ADONN: Adaptive Design of Optimized Deep Neural Networks for Embedded Systems

Mohammad Loni, Masoud Daneshtalab, Mikael Sjödin
School of Innovation, Design and Engineering, Mälardalen University
Västerås, Sweden
{mohammad.loni, masoud.daneshtalab, mikael.sjodin}@mdh.se

***Abstract*—Nowadays, many modern applications, e.g. autonomous system, and cloud data services need to capture and process a big amount of raw data at runtime, that ultimately necessitates a high-performance computing model. Deep Neural Network (DNN) has already revealed its learning capabilities in runtime data processing for modern applications. However, DNNs are becoming more deep sophisticated models for gaining higher accuracy which require a remarkable computing capacity. Considering high-performance cloud infrastructure as a supplier of required computational throughput is often not feasible. Instead, we intend to find a near-sensor processing solution which will lower the need for network bandwidth and increase privacy and power efficiency, as well as guaranteeing worst-case response-times. Toward this goal, we introduce ADONN framework, which aims to automatically design a highly robust DNN architecture for embedded devices as the closest processing unit to the sensors. ADONN adroitly searches the design space to find improved neural architectures. Our proposed framework takes advantage of a multi-objective evolutionary approach, which exploits a pruned design space inspired by a dense architecture. Unlike recent works that mainly have tried to generate highly accurate networks, ADONN also considers the network size factor as the second objective to build a highly optimized network fitting with limited computational resource budgets while delivers comparable accuracy level. In comparison with the best result on CIFAR-10 dataset, a generated network by ADONN presents up to 26.4 compression rate while loses only 4% accuracy. In addition, ADONN maps the generated DNN on the commodity programmable devices including ARM Processor, Hiph-Performance CPU, GPU, and FPGA.**

***Index Terms*—Neural Architectural Search, Approximation Computing, Neural Processing Unit, Multi-Objective Optimization**

I. Introduction

According to International Data Corporation (IDC) in 2017, the size of world's information is increasing with an explosive growth and would be 140 ZB by 2050 [1] where the conventional computing model cannot scale-up anymore due to the failure of Dennard scaling and Moore's law [2]. Approximate computing can be counted as a propitious alternative for the context of big data while the failure of traditional energy and performance scaling paradigm in affording of modern applications requirements leads computing landscape towards inefficiency. In addition, most of new technologies such as quantum computing, die stacking and 3D chip technologies, diamond transistors, and neuromorphic computing attempt to overcome the scaling challenges, however, they are not sufficiently mature. These technologies require huge changes across the system stack, from software systems to new hardware architectures, which is incompatible with the nature of current software structures [3]. Approximation computing attempts to improve power and performance utilization by depreciating outputs quality of imprecision tolerance applications such as objects recognition, data analytics, search engines and autonomous systems. Although there exist a variety of approximation techniques, finding the best approach has remained as a challenging issue since it requires a considerable try to balance a trade-off between efficiency and accuracy. Among approximate techniques, DNN provides a higher accuracy, a greater power/performance efficiency, and a better robustness [4], particularly when the approximation targets applications like vision and speech recognition functions. A DNN model highly depends on the approximate function which can be a Multi-Layer Perceptron (MLP) neural network with a single layer down to a many back-to-back deep layers.

Nevertheless, DNNs are ever-evolving, memory intensive, and complex processing models containing thousand to millions operations for the entire model which make their implementation overwhelming even for expert developers. Generally, there are two approaches aiming to tackle these challenges: ① diminishing the network size by leveraging network pruning techniques during training phase [5] and ② employing customized hardware accelerators [13], [9], [35]. However optimizing the network architecture at *design time* should be taken into account as the third approach since the choice of the architecture strongly impacts on both the performance and the output quality of DNNs. To benefit from this opportunity, we propose a neural acceleration framework, named ADONN, which automatically generates a robust DNN in terms of network accuracy and network size, then maps the generated network to an embedded device. Unlike previous neural architectural solutions that their focus are only on improving the accuracy level, ADONN also considers network size as the second objective of the search space in order to adaptively find a fit DNN for limited resource embedded devices. For this, ADONN is equipped with a multi-objective optimization (MO^2) method to solve neural architectural search problem by finding a set of Pareto-optimal surfaces. The design space has been pruned by taking inspirations from a cutting-edge architecture, DenseNet [6], to boost the convergence speed to an optimal result. To the best of our knowledge, ADONN is the first framework which uses a multi-objective neuro-

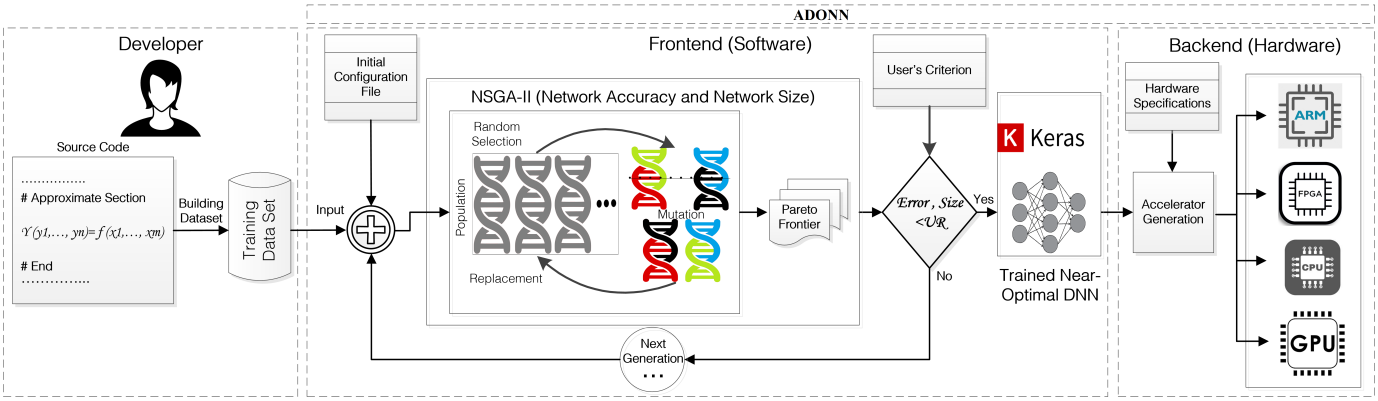

Fig. 1. The overview of ADONN framework.

evolutionary approach for the space exploration of finding optimal deep neural architectures while mapping the generated network to the given hardware.

An overview of the proposed framework is illustrated in Fig. 1. The configuration file of ADONN comprises predefined parameters for the MO^2 algorithm and network training parameters. As shown in Fig. 1, the input of the framework is a dataset for generating a neural network. To approximate execution of an application, developer first needs to identify the approximation region of the code, then provides a training dataset for the specified code block in order to be mimicked by a DNN generated by ADONN. Approximation region of the code should be both hotspot and less sensitive to quality loss in both data and operations. We can define a hotspot as a code region which consumes considerable energy or occupies the main part of execution time [7]. In nutshell, our main contributions in ADONN are threefold:

- Developed a multi-objective neuro-evolutionary method to discover near-optimal DNN architectures in terms of the accuracy and the network size.
- Supporting both Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN) models fitting with the required accuracy of diverse applications from mathematical function to image classification.
- Adaptive finding the best architecture regarding resource budget and execution time constraints. Then, mapping the generated network on different platforms to evaluate the applicability of ADONN is our last contribution.

The organization of the paper is as following. Section II gives background information on CNN and the MO^2 algorithm. Section III describes ADONN framework. Section IV presents and analyzes the experimental results. Section V reviews related work in this scope. Finally, section VI summarizes conclusion and future work.

II. Background

A. An Overview of CNNs

CNN is one of the most popular DNN models used to represent the information in a supervised manner suited for visual recognition. The CNN is composed of multiple back-to-back layers, where input image is fed to the first layer. Each layer gets feature maps information from previous layers, and generates new output feature maps by using a filter kernel. The convolution, pooling, normalization, and activation layers are used for feature extraction, and fully connected layers are responsible for classification. The performance criteria of a DNN include the ability to classify data that has never seen before, inference time, and learning rate which all depend on the multiple hyper-parameters of network architecture.

After computational analysis of a popular CNN, VGG-16 [31], we can conclude that convolutional layers (Conv.) are extremely computational intensive which contain 99.3% of total computation while Fully-Connected Layers (FCLs) are mainly memory-intensive that utilize more than 80% of data. Thus, for optimizing a CNN architecture, convolutional parameters including the number of convolutional layers, the sizes of each layer, and the filter size should be considered as the networks optimization hyperparameters. Moreover, the choice of activation functions in DNNs outstandingly influence on the training performance since the heart of neural networks is an activation function applied to a linear transformation. So, the activation function is also considered as a pivotal metric in designing the DNN architecture.

B. Multi-Objective Optimization (MO^2)

In this context, we use MO^2 to solve neural architectural search problem by finding a set of Pareto-optimal sets of network hyperparameters. In this work, we mainly consider two key objectives for the network optimization, classification accuracy and network size. Non-Dominated Sorting Genetic Algorithm (NSGA-II) [8], is a powerful meta-heuristic population-based evolutionary algorithm solving MO^2 problems which aims to adaptively fit a set of candidates to Pareto frontier. NSGA-II works as follows. In the first step, an offspring population U_t is formed from a parent population P_t by using Genetic Programming (GP), both with size N. Then we combine U_t and P_t to devise a third population R_t of size 2*N. Next, NSGA-II extracts a population (with size N) from R_t by employing a multiple objectives non-dominated

sorting and crowding distance comparison. The main aim of non-dominated sorting is to find a set of solution which cannot dominant each other. Moreover, by doing crowding distance sorting, we can orchestrate the density of solution for each Pareto front. NSGA-II selects the best N candidates for generating the next population called P_{t+1}. This procedure is repeated for the next generations until exceeds a predefined maximum number of generations or satisfies developer's criterion including a desired level of accuracy/network size. Although ADONN walks toward an optimal solution, it does not always guarantee to reach developer's criterion. Fig. 2 plots the Pareto frontier of improved individuals (P_i) for three different generations.

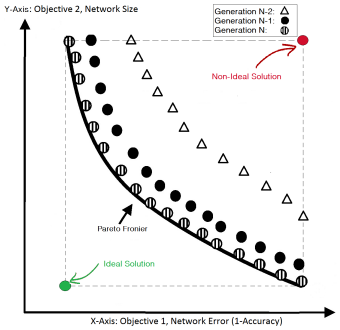

Fig. 2. Illustration of Pareto Front for the improved set of solutions based on NSGA-II algorithm.

III. ADONN

This section describes the ADONN framework which is composed of frontend and backend layers. The frontend is responsible to generate the optimized DNN while the backend layer deals with hardware configuration and mapping.

The hand-craft designing of DNN architectures needs a deep expertise and a large number of trial and error imposing a considerable design cost and efficiency risk. Thereby, tailoring the DNN architecture automatically has emerged as an efficient alternative solution in the machine learning community. This approach is considered for the frontend layer of our framework in which we propose a MO^2 based on a template architecture to generate DNNs.

a) Template Architecture: We investigate an evolutionary-based approach to search the design space inspired from DenseNet to vanish the probability of generating colossal networks which passed the one hundred deep layers obstacle. This decision leads ADONN to generate compact-inclined networks in a reasonable time by gaining from human experience in designing efficient DNNs. The basic template architecture of the network is shown in Fig. 3a. The generated network consist of back-to-back Condense_Layers for feature extraction while each layer consists of multiple Convolution_Layers. Each Convolution Block includes Batch Normalization, Activation Function, 2D Convolution and Dropout layers, respectively. The final classification is integrated by the max-pooling and the fullyconnected layers as the output layer with the softmax activation function. To pass maximum information between layers in the network, all the layers are connected to each other in a feed-forward manner such that each layer receives the additional feature map information from the whole former layer and combining them by using a concatenation layer. This structure leading us to enlarge sharing information and shorten path from the first layer to the last layer.

b) Search Space Alg1orithm: The main architectural hyperparameters of DNNs are listed in Table I. For cutting back the search space, the range of each hyperparameter is limited. Different combinations of these parameters form several architectures with various performances. Finding a near-optimal network architecture of the combination of these hyperparameters is the main goal of the search algorithm. In the other word, we can model the DNN architecture selection problem as the hyperparameter optimization problem. ADONN is equipped with the fast and multi-objective GP, NSGA-II, to discover near-optimal set of hyperparameters considering both the accuracy and the network size as the objectives. Total trainable network weights is defined as the network size objective since the performance and energy efficiency of the backend accelerator highly rely on inner product operations which are execution bottleneck of DNNs [9].

TABLE I
THE CNN HYPERPARAMETERS USED AS SEARCHING NEURAL DESIGN SPACE PARAMETERS.

Parameter	Deep CNN
Activation Function	hard-sigmoid, relu, elu, tanh, sigmoid, softplus, linear
# Condense_Layer	1, 2, 3, 4
# Convolution_Layer	16, 28, 40, 52
Kernel Size	3x3, 5x5
Optimizer	rmsprop, adam, sgd, adagrad, adadelta, adamax, nadam

Network hyperparameters are represented as a string of genomes using direct encoding and the recombination of these genes occurs with one-point crossover operation shown in Fig. 3b. The neural architectural exploration algorithm is explained in four steps as following: ① After generating random initial parent population P_t with size N, ADONN generates a network model based on the hyperparameters of each genome in the parent population. Then ADONN trains each individual model to calculate the network accuracy and network size for all the models. ② The offspring populating U_t will be created by using GP including crossover and mutation steps. ③ The NSGA-II sorts the combination of T_t and P_t to find the next generation parent population of N acceptable individuals which cannot dominant each other in terms of accuracy and network size. ④ This process will continue until attaining the predefined maximum number of generations. The entire search procedure is summarized in Algorithm 1. Fig. 3c illustrates a schematic of generated architecture including one Condense_Layer and five Convolution_Layers. Compare to DenseNet, ADONN generates more accurate networks with

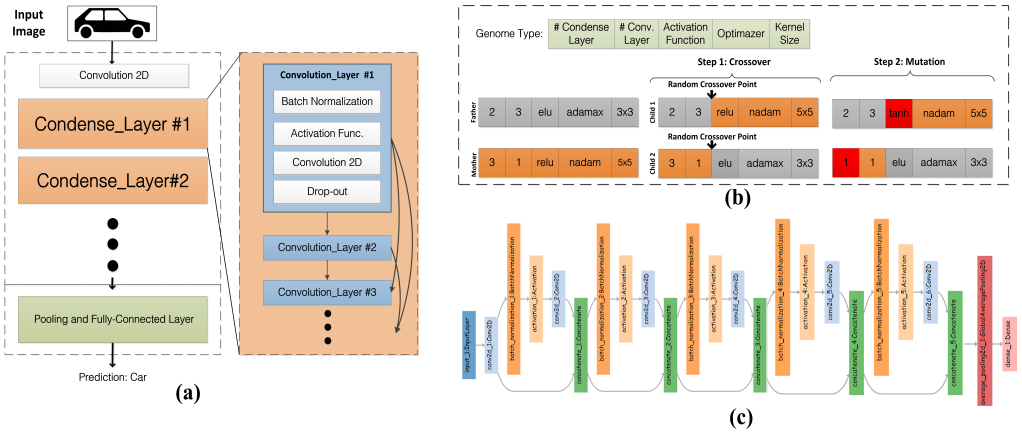

Fig. 3. (a) The template architecture of generated networks. (b) Genome type. (c) The schematic of a generated network.

superior flexibility regarding resource limitation of the back-end platform. To increase the rate of optimal discovering, we monitor all genomes in all previous generations. The output of the frontend layer is a set of improved network architectures on the Pareto curve with different network accuracies and sizes.

Efficient mapping of the generated network on hardware is the next step. Using Application-Specific Integrated Circuit (ASIC) as a customized ADONNs backend accelerator can gain considerable power and performance efficiency, nonetheless, ASIC cannot be reconfigured and reprogrammed. Graphic Processing Units (GPUs) are popular performance-centric accelerators refereed as another possibility to cope with diminishing the efficiency trend in the multi-core era [11]. Although GPUs offer a higher level of programmability and memory bandwidth, they suffer from huge power consumption and are efficient only for data parallel kernels and dense data structures [12]. On the other hand, the combination of supporting arbitrary forms of parallelism, flexibility, and power efficiency of off-the-shelf Field-Programmable Gate Arrays (FPGAs) provide a promising opportunity for efficient neural network implementation. Unfortunately, on-chip memory limitation, relatively primitive memory abstraction model, and the lack of efficient high-level APIs are the major bottlenecks of FPGA as a neural-based accelerator [9]. In fact, each of these hardware devices offers various capabilities for the real work problems. Section IV.c presents implementation results on different processing platforms.

IV. Methodology

To get the sense of practicality of the results, we need to evaluate ADONN using well-known datasets and compare with cutting-edge architectures. We first introduce employed training datasets, then the classification and implementation results will be presented. ADONN searches the optimal network architecture using partial training by using just 16 epochs since this epoch number is enough for making the decision. Fig. 4 plots the validation loss and validation accuracy progression by increasing the number of epochs for Net-CNN-Arch.3 with 0.14 million parameters. We got roughly 90% of maximum achievable accuracy after 16 epochs. ADONN utilizes the Keras Library [10] for training the network.

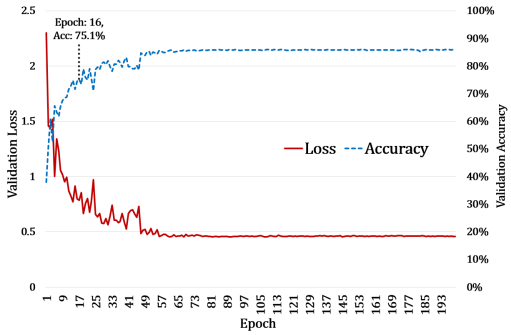

Fig. 4. Accuracy and loss Validation for Net-CNN-3 with 0.14M parameters.

A. Training Datasets

a) MNIST [14]: This is a dataset of black and white images for handwritten digit recognition containing 60,000 training and 10,000 testing images, respectively. Each image in the MNIST dataset is a 28x28 pixels with ten labeled output as 0 to 9 numbers.

b) CIFAR-10 [15]: This is a complex colorful benchmark dataset of natural images, each with 32x32 pixels which is mainly used for object recognition. This benchmark contains ten labeled output classes. CIFAR-10 training and testing datasets contain 50000 and 1000 images, respectively.

B. Classification Results

The near-optimal Pareto frontier results are illustrated in Fig. 5 to Fig. 8 on MNIST and CIFAR-10 datasets after just five generations. We got these results where ADONNs configuration file was set with the following parameters: dropout=0.2, epoch=16, batch size=128, number of generations=5, and random initial population with the size equal to 30. As can be seen, Pareto-optimal curves shifted toward left implying that our results have gotten improved set of network architecture candidates.

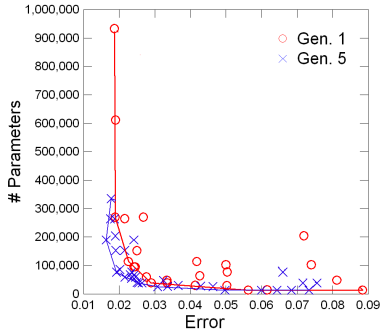

Fig. 5. Pareto frontier plots for MLP architecture generated for MNIST dataset.

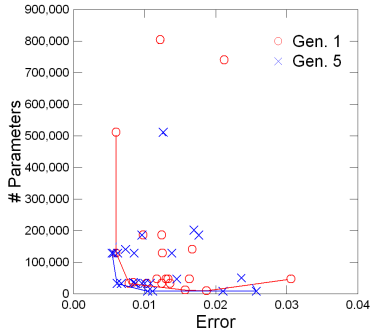

Fig. 6. Pareto frontier plots for CNN architecture generated for MNIST dataset.

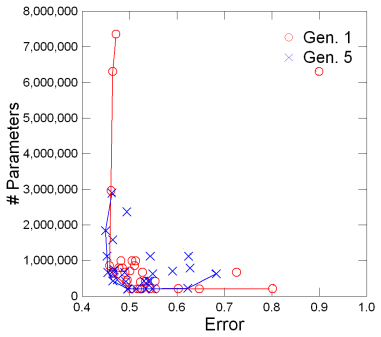

Fig. 7. Pareto frontier plots for MLP architecture generated for CIFAR-10 dataset.

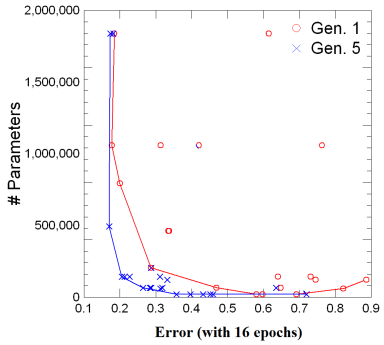

Fig. 8. Pareto frontier plots for CNN architecture generated for CIFAR-10 dataset.

Algorithm 1: Pseudo Code of ADONN Procedure

Input: **N**: Population Size, **G**: Max. Number of Generations, **H**: Possible Hyperparameters

Output: A Set of Optimal Architectures on Pareto Frontier

Function `ADONN` (N, G, H) **:**

- P_0= **Random_Population** (N, H); //Creating initial random solutions with size N
- **Objectives_Function** (P_0, Size (P_0)); //Evaluating the objectives of each solution in the population
- U_0= **Selection_Crossover_Mutation** (P_0); //Generating the offspring population by doing random crossover and mutation
- t=1;
- **while** *($t < G$) or (Criterion Not satisfied)* **do**
  - R_t=**Combine** (P_t, U_t); //Merging Parent and Offspring population, the size of P_{t+1} is $2 * N$
  - **Objectives_Function** (P_{t+1}, Size (P_{t+1}));
  - $Sort_t$=**Non_Dominant_Sort**(P_{t+1}); //Sorting the first population in fronts
  - $pfs[t]$=**Crowding_Distance_Sorting**($Sort_t$); //Symmetric disturbing offspring population by crowding distance sort to build Pareto frontier and save it in pfs
  - P_{t+1}=$pfs[t]$;// Creating Next Population
  - U_{t+1}=**Selection_Crossover_Mutation** (P_{t+1});
  - **Objectives_Function** (U_{t+1}, Size (U_{t+1}));
- **return** $pfs[G]$;

Function `Objectives_Function` (*Population P, Size N*) **:**

- i=1;
- **while** *($i < N$)* **do**
  - List [i]= **Extract_Network_Parameters**(P_i);
  - $model[i]$= **Create_Model** (List [i]); //Generating a DNN model using network hyperparameters
  - $Acc.[i]$, $\#Params[i]$=**Train_Evaluate** ($model[i]$); // Train the ntwork to get validation accuracy and num. network parameters
- **return** $Accuracy$, $\#Parameters$;

We have verified the effectiveness of ADONN compare to the error rate and total number of trainable parameters of the other cutting-edge approaches shown in Table II. For getting the results presented in Table II, we fully trained networks with 300 epochs. Network architectures with highest accuracy are employed as the baseline of the comparisons. Compare to a reinforcement learning solution, MetaQNN, we lost 0.06% accuracy for MNIST dataset while we have 43x compression rate. Not only compare to MetaQNN, but only also the superiority of ADONN's optimization rate is clear for

MNIST dataset.

Net-CNN-Arch.1, Net-CNN-Arch.2, and Net-CNN-Arch.3 are three different nodes of Pareto frontier selected from fifth generation. These three nodes have different network objectives which give a vast authority to ADONN to select the most appropriate architecture based on the execution time constraints or resource limitation of the target hardware platform. Net-CNN-Arch.1 loses 4% accuracy compared to the most accurate networks [26], while has 26.4x less parameters. Moreover, MLP model presents comparable accuracy for MNIST, but it cannot provide acceptable accuracy for CIFAR-10, revealing we need more complex architectures for modern dataset. In nutshell, ADONN strikes better the balance between network accuracy and network size compare to reinforcement learning-based solutions, evolutionary-based approaches and hand-craft designs.

TABLE II
COMPARISON RESULTS OF ERROR RATE ON MNIST AND CIFAR-10 DATASETS.

Dataset	Method	#Params (x10^6)	Error (%)
MNIST	MetaQNN [21]	5.59	0.35
	EDEN [28]	1.8	1.6
	SimpleNet [29]	0.3	0.25
	Wan et al. [30]	-	0.21
	Our MNIST-MLP	0.19	1.2
	Our MNIST-CNN	**0.13**	**0.41**
CIFAR-10	NAS-v1/v3 [22]	4.2/37.4	5.50/3.65
	SimpleNet [29]	5.48	4.68
	VGG-16 [31]	138	7.55
	DenseNet (k=12)-40 [6]	1.0	7.0
	DenseNet (k=12)-100 [6]	7	5.77
	DenseNet (k=24)-100 [6]	27.2	5.83
	EDEN [28]	0.17	25.6
	ResNet-20 [27]	0.27	8.75
	ResNet-110 [27]	1.7	6.43
	Masanori et al. [24]	1.68	5.98
	Block-QNN-22L [23]	39.8	3.54
	MetaQNN [21]	6.92	11.18
	Real et al. [25]	5.4	5.4
	Gastaldi et al. [26]	26.4	2.86
	Our Net-MLP	0.66	37.0
	Our Net-CNN-Arch.1	**1.0**	**6.9**
	Our Net-CNN-Arch.2	**0.49**	**8.7**
	Our Net-CNN-Arch.3	**0.14**	**14.1**

C. Implementation Results

To verify the practical impact of ADONN, we used four prevalent hardware platforms, Xilinx UltraScale plus FPGA, NVIDIA Tesla M60 GPU, Intel Core i7-7820, and ARM Cortex-A15. Table III summarizes the specification of test platforms. We picked out four congruent networks offering better accuracy per parameters including ResNet-20, ResNet-110, DenseNet (k=12)-100, and DenseNet (k=24)-100 to compare with the generated networks by ADONN. We also did not use any network compression technique to only assess the influence of network architecture on inference time. Due to the sake of brevity, we just present the implementation results of the more complex dataset, CIFAR-10. Keras framework automatically uses cuDNN to compile a neural network for GPU. For getting FPGA results, the Amazon EC2 deep learning F1.2xlarge instance has been used.

TABLE III
HARDWARE PLATFORM DETAILS.

Platform	CPU	GPU	ARM	FPGA
Frequency (GHz)	2.9	1.178	1.9	.8
Technology (nm)	14	28	28	16 (FinFET+)
TDP (W)	45	300	5	-
Cores/Total Thread	4/8	4096 CUDA Cores	8/8	FF= 2.5(x10^6) LUT= 1.18(x10^6) DSP= 6800
Memory	8MB Cache	16GB GDDR5	2.5MB Cache	BRAM= 75.5 Mb
Approx. Price (USD)	378$	7,532$	60$/board	-

Unlike CPUs, we do need an initialization phase to copy data to GPU/FPGA's internal memory, before lunching processing kernel. Usually, kernel time is used for reporting runtime results, however, considering the communication time is vital for embedded implementations, especially for mission critical applications since these applications are mainly latency-oriented. Due to this reason, the total execution time must be taken into account as the evaluation metric. In addition, we believe compacting a network potentially could diminish the overhead of communication time since less number of data packets need to be copied via PCI-Express bus. To increase the precision of results, we got them for 10000 times and the average time is leveraged for presenting the results. Fig. 9 to Fig. 12 plot accuracy, the logarithmic scale (to improve visual comprehension) of the number of parameters and the speedup compared to the baseline, DenseNet (k=12)-100. The main reason of selecting DenseNet (k=12)-100 as the ideal baseline is that it delivers better accuracy-parameters trade-off in comparison with the other networks. Unlike accuracy and the number of parameters, execution time is a platform aware metric and highly depends on hardware implementation, compiler, and the software stack. Therefore, there is no exact speedup similarity among different hardware platforms. The results show that for each hardware platform there is a firm relation among inference time, network accuracy and network parameters. In nutshell, we can conclude: ① the networks with more parameters have higher accuracy, ② after getting a network more complex, the speedup rate will be decrease, e.g. we got maximum speedup up to 39% on FPGA platform with minimum number of parameters for Net-CNN-3, while DenseNet (k=24)-100 with the best accuracy result always has shown at least 0.33 speed-down. ③ The execution time is scaled by changing the number of parameters demonstrating the considering network size as a design objective decreases both the communication and kernel execution times.

V. RELATED WORK

A. Automatic Designing Deep Neural Network

In this section, we address state-of-the-art approaches pointing to automatically design the architecture of DNNs. These approaches could be categorized into the hyper-parameter optimization, reinforcement learning and evolutionary approaches.

a) Hyperparameter Optimization: From machine learning point of view, we can model DNN architecture designing problem as the hyperparameter optimization. There have been proposed many hyperparameter tuning methods, such as Grid

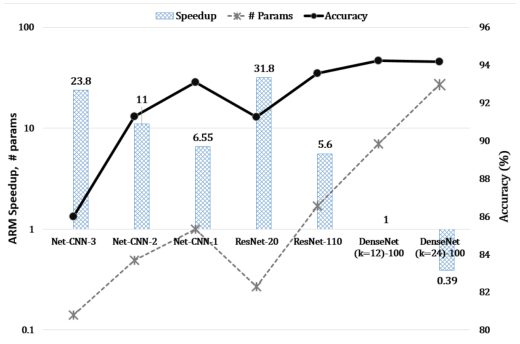

Fig. 9. Speedup of ADONN generated networks in comparison to network size and accuracy on ARM platforms.

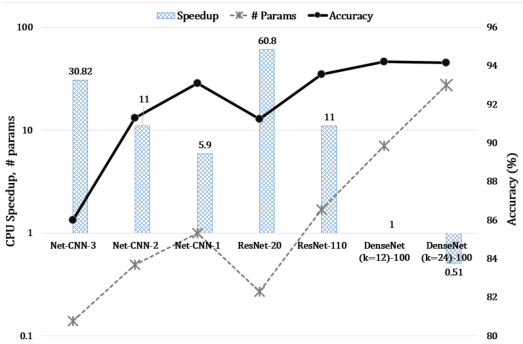

Fig. 10. Speedup of ADONN generated networks in comparison to network size and accuracy on CPU platforms.

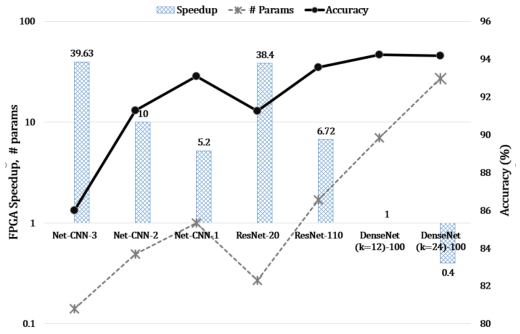

Fig. 11. Speedup of ADONN generated networks in comparison to network size and accuracy on FPGA platforms.

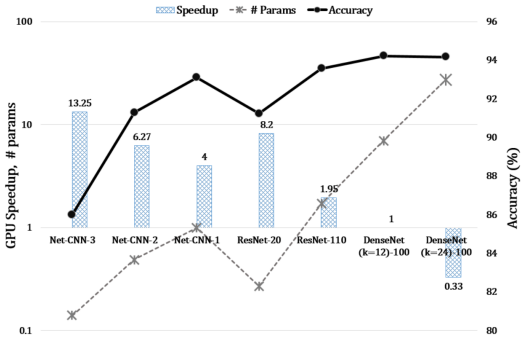

Fig. 12. Speedup of ADONN generated networks in comparison to network size and accuracy on GPU platforms.

Search (GS) [16], gradient search [17], Random Search (RS) [18], and Bayesian optimization-based method [19]. However GS is relatively slow, using RS is challenging due to extremely random sampling in the search space, and Bayesian-based methods suffer from immense computational cost. In addition, these methods are suitable only for search models with a fixed-length space and hard to design more flexible architectures from scratch [20].

b) Reinforcement Learning: Recently there has been much work at the intersection of reinforcement learning and deep learning which show better results for image classification applications compared to best hand-craft DNN accuracy results. Baker et al. [21] have proposed a meta-modeling approach based on reinforcement learning to produce CNN architectures. In this paper A Q-learning agent explores and exploits a space of model architectures with greedy strategy and experience replay. In [22], a recurrent neural network (RNN) was used to generate neural network architectures, and the RNN was trained with reinforcement learning to maximize the expected accuracy on a learning task. This method uses distributed training and asynchronous parameter updates with 800 graphic processing units (GPUs) to accelerate the reinforcement learning process. In [23], a block-wise network generation pipeline called BlockQNN has been provided to automatically build high-performance networks using the Q-Learning paradigm with epsilon-greedy exploration strategy. Despite their success, these models are considerably too slow and require huge computational resources in both training and prediction steps, e.g. MetaQNN [21] contains 11.18 M trainable parameters and used 10 GPUs for 8-10 days to train a CIFAR-10 classifier.

c) Evolutionary-based approaches: Suganuma et al. [24] tried to automatically construct CNN architectures for an image classification task based on Cartesian genetic programming (CGP). The CNN structure and connectivity represented by the CGP encoding method are optimized to maximize the validation accuracy. Sun et al. [20] proposed a new method using genetic algorithms for evolving the architectures and connection weight initialization values of a deep CNN. In their proposed algorithm, an efficient variable-length gene encoding strategy is designed to represent the different building blocks and the unpredictable optimal depth in convolutional neural networks. In addition, a new representation scheme is developed for effectively initializing connection weights which is expected to avoid networks getting stuck into local minima. Real et al [25] proposed y simple evolutionary techniques at unprecedented scales to discover models for the CIFAR-10 and CIFAR-100 datasets. They used novel and intuitive mutation operators that navigate large search spaces.

B. DNN Acceleration

After reviewing literature, various approximation code accelerators have been found [32], [33], [34], and [35]. However the main weakness of them is the NN architecture selection procedure. Prior work mainly used a simple search methodology to explore a small design space which is not applicable for

real-world applications. Moreover, they just generate a deep multi-layer network which is obsolete and does not produce competitive accurate results for modern applications such as object recognition.

VI. Conclusion and Future Work

DNNs are both computational and memory intensive processing patterns, leading to difficult implementation especially on embedded devises. The importance of the problem will be more highlighted when we need to keep processing close to sensors due to guaranteeing privacy, increasing energy efficiency and ensuring worst-case response-times. To tackle this problem, we proposed ADONN, a framework which automatically generates a highly-optimized DNN for commercial of-the-shelf embedded devices. ADONN alleviates the huge computational cost of DNNs by benefiting from squeezing the network architecture at design time. To reach this goal, ADONN integrates a multi-objective optimization strategy to optimally search the design space of DNNs. For generating an embedded implementable accelerator, ADONN considers the number of trainable parameters of the network as the second objective of search algorithm in order to find a highly optimized DNN. The evaluation results demonstrate the effectiveness of ADONN on both pure kernel time and communication time. In addition, providing an optimized set of solutions fitting with various Developer's criterion is an attractive benefits of ADONN. Implementing an automatic profiler for exploring approximation regions of code will be also remain as future work.

Acknowledgment

This work has been supported by KKS within the projects DeepMaker and DPAC.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, Data Age 2025 - The Evolution of Data to Life-Critical: Don not Focus on Big Data; Focus on the Data That is Big, IDC White Pap., no. April, pp. 125, 2017.

[2] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, Dark silicon and the end of multicore scaling, IEEE Micro, vol. 32, no. 3, pp. 122134, 2012.

[3] Ungerer, T., Fey, I.D., Knebel, M., Bagherzadeh, N., Bartolini, S., Bertels, K., Bradatsch, C., Carrasco, J.M.G., De Bosschere, K., Duranton, M., DACLE, C.L.: Report on Disruptive Technologies for years 2020-2030.

[4] M. Wyse, "Modeling Approximate Computing Techniques," Academic paper.

[5] J. T. and W. D. Song Han, Jeff Pool, Learning both Weights and Connections for Efficient Neural Networks, in Advances in Neural Information Processing Systems, 2015, vol. 50, no. 2, pp. 1135–1143.

[6] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, Densely connected convolutional networks, In Proceedings of the IEEE conference on computer vision and pattern recognition, vol. 1, no. 2, p. 3. 2017.

[7] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, AxBench: A multiplatform benchmark suite for approximate computing, IEEE Des. Test, vol. 34, no. 2, pp. 6068, 2017.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182197, 2002.

[9] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, Caffeine:Towards Uniformed Representation and Acceleration for Deep Convolutional Neural Networks, Proc. 35th Int. Conf. Comput. Des. - ICCAD 16, no. August, pp. 18, 2016.

[10] F. Chollet, Keras, GitHub, 2015. [Online]. Available: https://github.com/fchollet/keras.

[11] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, Power challenges may end the multicore era, Commun. ACM, vol. 56, no. 2, p. 93, 2013.

[12] B. Falsafi, B. Dally, D. Singh, D. Chiou, J. J. Yi, and R. Sendag, FPGAs versus GPUs in Data centers, IEEE Micro, vol. 37, no. 1, pp. 6072, 2017.

[13] H. Sharma Jongse Park Emmanuel Amaro Bradley Thwaites Praneetha Kotha Anmol Gupta Joon Kyung Kim Asit Mishra Hadi Esmaeilzadeh, DNNWEAVER: From High-Level Deep Network Models to FPGA Acceleration, IEEE Int. Conf. Mechatronics, Electron. Automot. Eng., no. 2, pp. 7680, 2015.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient Based Learning Applied to Document Recognition, Proc. IEEE, vol. 86, no. 11, pp. 22782324, 1998.

[15] A. Krizhevsky and G. Hinton. Cifar-10 dataset. https://www.cs.toronto.edu/ kriz/cifar.html.

[16] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kgl, Algorithms for Hyper-Parameter Optimization, in Advances in Neural Information Processing Systems (NIPS), 2011, pp. 25462554.

[17] Y. Bengio, Gradient-based optimization of hyperparameters, in Neural computation, no. 8, pp. 1889-1900, 2000.

[18] J. Bergstra JAMESBERGSTRA and U. Yoshua Bengio YOSHUABENGIO, Random Search for Hyper-Parameter Optimization, J. Mach. Learn. Res., vol. 13, pp. 281305, 2012.

[19] J. Snoek, H. Larochelle, and R. P. Adams, Practical Bayesian Optimization of Machine Learning Algorithms, Adv. Neural Inf. Process. Syst., vol. 25, pp. 29602968, 2012.

[20] Y. Sun, B. Xue, and M. Zhang, Evolving Deep Convolutional Neural Networks for Image Classification, arXiv prepr. arXiv:1710.10741, 2017.

[21] B. Baker, O. Gupta, N. Naik, and R. Raskar, Designing Neural Network Architectures using Reinforcement Learning, arXiv Prepr., pp. 116, 2016.

[22] B. Zoph, and Q.V. Le, Neural architecture search with reinforcement learning, arXiv prepr. arXiv:1611.01578, 2016.

[23] Z. Zhong, J. Yan, and C.L. Liu, Practical Network Blocks Design with Q-Learning, arXiv prepr. arXiv:1708.05552, 2017.

[24] M. Suganuma, S. Shirakawa, and T. Nagao, A genetic programming approach to designing convolutional neural network architectures, GECCO 2017 - Proc. 2017 Genet. Evol. Comput. Conf., pp. 497504, 2017.

[25] E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, Q. Le, and A. Kurakin, Large-scale evolution of image classifiers, arXiv prepr. arXiv:1703.01041, 2017.

[26] X. Gastaldi, Shake-shake regularization, arXiv prepr. arXiv:1705.07485, 2017.

[27] K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770778.

[28] E. Dufourq, and B.A. Bassett, EDEN: Evolutionary Deep Networks for Efficient Machine Learning, arXiv prepr. arXiv:1709.09161, 2017.

[29] S.H. Hasanpour, M. Rouhani, M. Fayyaz, and M. Sabokrou, Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures, arXiv prepr. arXiv:1608.06037, 2016.

[30] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, Regularization of neural networks using dropconnect, Icml, no. 1, pp. 109111, 2013.

[31] K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, Int. Conf. Learn. Represent., pp. 114, 2015.

[32] B. Grigorian and G. Reinman, Accelerating divergent applications on SIMD architectures using neural networks, in 2014 32nd IEEE International Conference on Computer Design, ICCD 2014, 2014, pp. 317323.

[33] Z. Du, A. Lingamneni, Y. Chen, K. V. Palem, O. Temam, and C. Wu, Leveraging the Error Resilience of Neural Networks for Designing Highly Energy Efficient Accelerators, IEEE Trans. Comput. Des. Integr. Circuits Syst., vol. 34, no. 8, pp. 12231235, 2015.

[34] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, SNNAP: Approximate computing on programmable SoCs via neural acceleration, in 2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015, 2015, pp. 603614.

[35] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, Neural acceleration for GPU throughput processors, in Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48, 2015, pp. 482493.