Designing Compact Convolutional Neural Network for Embedded Stereo Vision Systems

Mohammad Loni*, Amin Majd†, Abdolah Loni‡, Masoud Daneshtalab*, Mikael Sjödin* and Elena Troubitsyna∓
*School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
†Department of Information Technology, Åbo Akademi University, Turku, Finland
‡Computer Science and Mathematics Department, Allameh Tabatabai University, Tehran, Iran
∓Theoretical Computer Science Department, KTH Royal Institute of Technology, Stockholm, Sweden
Email: *{mohammad.loni, masoud.daneshtalab, mikael.sjodin}@mdh.se, †{amin.majd, Elena.troubitsyna}@abo.fi

***Abstract*—Autonomous systems are used in a wide range of domains from indoor utensils to autonomous robot surgeries and self-driving cars. Stereo vision cameras probably are the most flexible sensing way in these systems since they can extract depth, luminance, color, and shape information. However, stereo vision based applications suffer from huge image sizes and computational complexity leading system to higher power consumption. To tackle these challenges, in the first step, GIMME2 stereo vision system [1] is employed. GIMME2 is a high-throughput and cost efficient FPGA-based stereo vision embedded system. In the next step, we present a framework for designing an optimized Deep Convolutional Neural Network (DCNN) for time constraint applications and/or limited resource budget platforms. Our framework tries to automatically generate a highly robust DCNN architecture for image data receiving from stereo vision cameras. Our proposed framework takes advantage of a multi-objective evolutionary optimization approach to design a near-optimal network architecture for both the accuracy and network size objectives. Unlike recent works aiming to generate a highly accurate network, we also considered the network size parameters to build a highly compact architecture. After designing a robust network, our proposed framework maps generated network on a multi/many core heterogeneous System-on-Chip (SoC). In addition, we have integrated our framework to the GIMME2 processing pipeline such that it can also estimate the distance of detected objects. The generated network by our framework offers up to 24x compression rate while losing only 5% accuracy compare to the best result on the CIFAR-10 dataset.**

***Index Terms*—Neural Network Architecture Search, Deep Convolutional Neural Network, Neural Processing Unit, Stereo Vision Systems**

I. Introduction

Stereo vision systems are multi-modal sensing way allowing for extracting three-dimensional (3-D) information, luminance, color, distance, and shape. Compare to different sensing ways such as LIDAR, RADAR, and cameras, stereo vision cameras are more attractive due to providing heterogeneous information simultaneously that are beneficial for many applications to have comprehensive information to explore unknown surrounding environment and dynamic navigation. For example autonomous vehicles can benefit from stereo vision systems to move on the right path, detect dynamic objects such as pedestrians and other vehicles, and estimating the distance between the vehicle and recognized objects.

Today, Deep Convolutional Neural Networks (DCNNs) construct the skeleton of visual recognition, decision making and prediction algorithms because of providing higher accuracy and more flexibility compared to old-fashioned solutions. However, DCNNs are complex and ever-evolving processing models containing up to millions operations for the entire model making their implementation unexciting. In addition, current stereo cameras produce high-resolution images requiring huge computational throughput and considerable energy consumption which are main obstacles for embedded system implementation.

To overcome these challenges, GIMME2 embedded system [1] has been utilized and adapted in the first step. GIMME2 is a power efficient FPGA-based stereo vision system. There exist generally two strategies approaching to overcome DCNN implementation barriers: ① employing customized hardware accelerators [7], [9], [30]. ② decreasing the network computation by using network pruning techniques in training step [5]. However, optimizing the network architecture should be also taken into account as the third solution because the choice of the architecture strongly effects on the inference time, memory usage, hardware footprint, the accuracy level, and the prediction quality of DCNNs. For this, we introduce an evolutionary-based accelerator which is also compatible with GIMME2. Our proposed solution tries to design a near-optimal DCNN for image classification algorithms in term of network size while guaranteeing acceptable level of accuracy by employing Cartesian Genetic Programming (CGP) based method. Our framework then maps the generated network to a multi/many core SoC.

There are other neural approximation accelerators [27]–[30]. However, they fail to generate an efficient DCNN architecture. Previous works mainly employed a simple restricted design space exploration methodology to diminish the time of generating accelerator. Plus, they just generate a deep multi-layer networks which is relatively obsolete and do not offer acceptable accurate results for many modern applications. Unlike prior works, network size also is considered as the second objective to design a compact neural network since network size is important for embedded platforms. Toward this purpose, our framework uses a multi-objective optimization method to solve neural architecture search problem. The output of multi-objective algorithm will be a set of Pareto-optimal curves while each curve comprises n different architectures

978-1-5386-6689-0/18/$31.00 ©2018 IEEE
DOI 10.1109/MCSoC2018.2018.00049

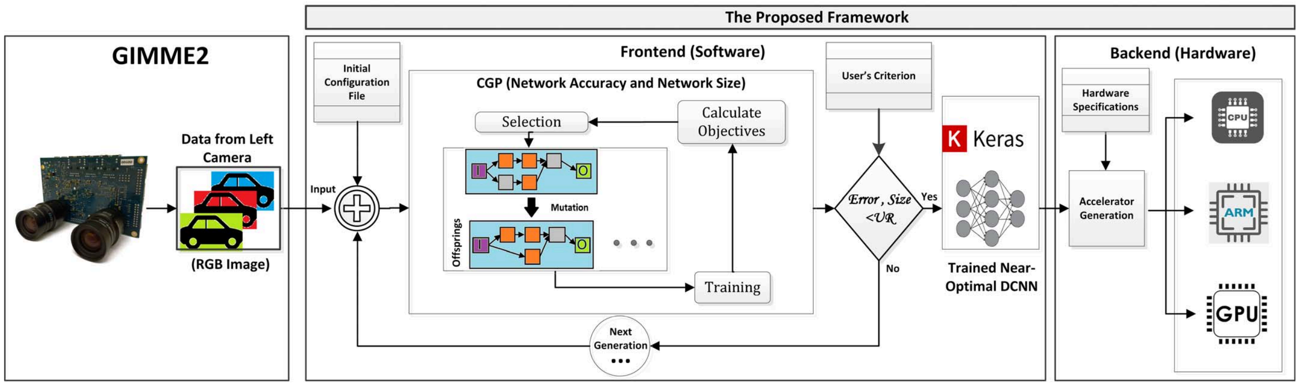


Fig. 1. The overview of the proposed framework.

with various accuracies and network sizes.

The overview of the proposed framework is depicted in Fig. 1. The configuration file contains predefined values for DCNN training, e.g. number of epoch, learning rate, total number of generation and the number of offspring. The input of the proposed framework is an image stream receiving from stereo vision camera. In nutshell, our main contributions in this paper are twofold:

- Developing a multi-objective neuro-evolutionary optimization approach to search design space of DCNN architectures by considering the accuracy and the network size as the key objectives.
- Finally, the designed DCNN is embed into GIMME2 processing system such that it can estimate the distance of detected objects.

We make the structure of the rest of the paper as follows. Section 2 gives background information on multi-objective algorithm, DCNN, and GIMME2 embedded device. Section 3 reviews related work in this scope. Section 4 describes CGP approach to design a DCNN. Section 5 presents and analyzes the experimental results. Finally, section 6 summarizes conclusion and future work.

II. Background

A. DCNN

DCNN is one of the most popular supervised deep learning models which are mainly used for object classification/recognition. Multiple back-to-back hidden layers compose the DCNN architecture, where input image is fed to the first layer. Each layer is responsible for receiving feature maps information from previous layer, and after applying special operations on the data, e.g. filtration, resizing, and etc., then deliver them to the next layer in the chain. The fully connected layers are responsible for classification, while the convolution, pooling, normalization, and activation layers are used for feature extraction and feature map resizing. The ability to classify data that has never seen before, inference time, and learning rate of a DCNN are all depend on the network architecture, therefore, optimizing the network architecture potentially provides considerable performance. To find computational bottlenecks of DCNNs, we have analyzed a well-known DCNN, VGG-16 [26]. As the first conclusion, Fully-Connected Layers (FCLs) are memory-intensive containing more than 80% of memory accesses, however, convolutional layers (Conv.) are strongly computational intensive which utilize 99.3% of total computation. This is illustrated in Fig. 2, the comparison between computational volume, memory accesses, and pure software execution time for the VGG-16 architecture.

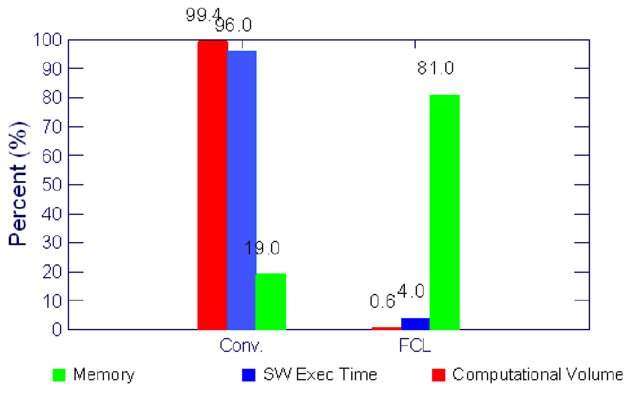


Fig. 2. The percentage of memory usage, computational volume, and software execution time of VGG-16.

B. Multi-Objective Cartesian Genetic Programming

We need an optimized approach to make the best balance between network size and the accuracy. Computability is a significant challenge especially in complex problems; there are no guarantees that such problems can be solved in a satisfactory manner in a limited time. Several techniques have been proposed to improve solving of NP-hard complex problems. Among them, evolutionary computing (EC) methods are more prominent [19], [21]. There are population-based search methods that mimic the process of natural selection and evolution, as some characteristics of this process can be utilized in optimization problems. There exists different type of evolutionary algorithms like Genetic algorithm (GA), imperialist competitive algorithm [34], memetic algorithm and etc. ECs are divided based on application and type of problems, for example discrete problems and continues problems. Genetic algorithm and Cuckoo are used on discrete problems and PSO and ICA are used for contentious problems. Finding the efficient network is a discrete problem therefore we select a superb combination of GA and Cuckoo. In other words, we select the simple and fast operation of GA (mutation) to explore the search space more accurate than other exploration methods with greedy search of Cuckoo together. Coding and encoding are the key parameters to achieve the best results. With a suitable coding, algorithm can be simulated from phenotype to genotype and can make the phenotype from genotype by decoding. Binary coding, tree coding, real coding are more popular type of coding models but they cannot become fix on our problem. Our problem needs a variable length coding method with different type of elements that generate different network architectures with various sizes and parameters. The more suitable coding method for optimizing the network is Cartesian coding. Based on the Cartesian coding [32], the algorithm can have perfect chromosomes that build different possible network architectures with various sizes and parameters.

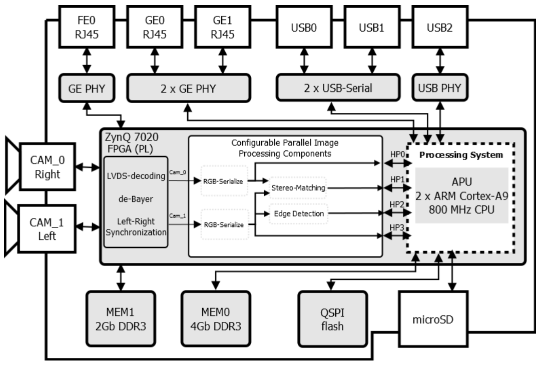


Fig. 3. The overview of GIMME2 Architecture.

C. GIMME2 Architecture

GIMME2 is an embedded power efficient stereo vision system based on the Xilinx Zynq System-on-Chip (SoC) FPGA that provides megapixel pictures and supports a high speed image processing pipeline. Fig. 3 illustrates the overview of GIMME2 architecture. Providing high resolution images, low power consumption, and standardized interface are the main properties of GIMME2, allowing vision-based sensing feasible. GIMME2 is equipped with two Aptina MT9J003 image sensors which are CMOS sensors with 15fps@10MP (3856 x 2764 bayer pattern array) or 60fps@1080p, with 12-bit color resolution. Each sensor produce 2.8 Gb/s data over four 240 MHz LVDS-lanes. A simple Zero-mean Sum of Absolute Differences (ZSAD) stereo-matching is utilized for full HD real-time depth estimation [4]. Table I describes GIMME2 key features. Frame rate could be varied for different applications, e.g. it can provide up to 240 frame/second for images with 640x400 pixels. We cannot use the FPGA of GIMME2 since it is filled by preprocessing logics. Thus we aim to design a compact network to be processed on the ARM processor in the reasonable time.

TABLE I
THE SPECIFICATION OF GIMME2.

Frame rate	15fps@10MP, 60fps@1080p
Processing Unit	Xilinx Zynq 7020
Programmable Logic	Artix-7 85K Logic Cells
Processing System	Dual ARM Cortex-A9 (766MHz)
Memory	4Gb DDR3, 2Gb DDR3, QSPI, SD-card
Communication	2xGBE, 1xFE, 3xUSB 2.0 (one host)
Power consumption	18W@24V
Physical dimensions	130x82mm

III. 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: DNN architecture designing problem can be modeled as the hyperparameter optimization. There are some solutions for hyperparameter tuning problems, like gradient search [12], Grid Search (GS) [11], Random Search (RS) [13], and Bayesian optimization-based method [14]. GS is relatively slow, Bayesian-based methods suffer from massive computational cost, and using RS is difficult because of extremely random sampling in the search space. In addition, these methods are suitable only for search models with a fixed-length space and hard to design more flexible DCNN architectures from scratch [15].

b) Reinforcement Learning: Recently there have been many works at the intersection of reinforcement learning and designing DCNN providing better results for image classification in comparison with the best hand-craft DCNN results. In [17], 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. Baker et al. [16] have proposed a meta-modeling approach based on reinforcement learning to produce DCNN architectures. In this paper a Q-learning agent explores and exploits a space of model architectures with greedy strategy and experience replay. In [18], 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. However, these models are considerably too slow and require huge computational resources in both training and prediction steps, e.g. MetaQNN [16] contains 11.18 M trainable parameters and used 10 GPUs for up to 10 days to train a CIFAR-10 dataset.

c) Evolutionary-based approaches: Sun et al. [15] proposed a new method using genetic algorithms for evolving the architectures and weight connections of a DCNN. In their proposed algorithm, an efficient variable-length gene encoding strategy is designed to represent the different building blocks and the unpredictable optimal depth. Suganuma et al. [19] attempted to automatically build a DCNN architectures for an image classification task based on Cartesian genetic programming (CGP). The DCNN architecture and connectivity represented by the CGP encoding method are optimized to maximize the validation accuracy. Real et al [21] proposed a simple evolutionary techniques for discovering models on the CIFAR-10 and CIFAR-100 datasets. They used novel and intuitive mutation operators which navigate vast search spaces. Finally, in a forthcoming accepted publication [20], we will show how we can use ordinary GP for designing a optimized DNN for embedded systems.

IV. Designing DCNN Using Multi-Objective CGP

In this paper, we tried to present the multi-objective version of a CGP approach to design a near-optimal DCNN based on the work of Suganuma et al. [19]. The DCNN architecture is represented by direct encoding, where CGP is used to search the architecture space by employing one point mutation operator to modify genome parameters. We briefly describe the structure of multi-objective CGP and our modifications which enhance the DCNN performance and memory footprint.

As mentioned in section II.B, the CGP structure is used for encoding the genome type by representing a network in directed graph that is mapped in two-dimensional matrix. The fixed length genome type is pictured in Fig. 4.(a) with three columns and two rows, where the number of processing nodes could be different from genome length. In this representation, we have two different nodes, active and inactive, where active node should be in the chain, from input to the output. The functionality of each node block can be composed of max-pooling, average pooling, concatenation, activation, summation, ConvBlock, and ResBlock. The convBlock is composed of an ordinary convolution layer, batch normalization and activation, respectively. Moreover, ResBlock is a bit more complex where it consists of ConvBlock, convolution layer, batch normalization, and summation to perform element-wise addition on ConvBlock input and the normalization output. Fig. 4.(b) displays the architecture of ConvBlock and ResBlock. Finally, the output node is always a fully connected layer with the softmax activation function. The valid range of node blocks and network training parameters is shown in Table II. Adding convolution layer with different settings to the node block list based on our prior knowledge is a minor contribution to increase the efficiency of final results.

In the evolution procedure, we first randomly generate a network architecture as a single parent. Then in each iteration, the parent will be mutated by randomly changing connections and node blocks to generate n offsprings. Then, the parent will be replaced with the most privileged child. This process will be continued until satisfying user criteria, e.g. we have to classify at least 10 frames/second and thus our framework should generate a compact network with an acceptable accuracy level and 600 millisecond inference time. Unlike the ordinary mutation operator of CGP that has the possibility to only modify inactive nodes, a forced mutation operator has been defined to change at least one active node. The fitness function inspired by meta-heuristic Cuckoo idea is described in (1), and (2) as follows;

$$net\_param = \left(\frac{max\_param - net\_param}{max\_param}\right) \tag{1}$$

$$Score = \alpha \times (net\_param) + \beta \times (val\_acc) \tag{2}$$

net_param is the optimization metric for the network size, where smaller network sizes can be inferred from higher values. max_param is the maximum number of trainable parameters for the biggest generated network in the worse case and it depends on the size of CGP representation matrix. In addition, The $Score$ factor is CGP fitness function for selection process which has been designed to be in the range of 0 to 1. In the second equations, the a and b parameters are practical coefficients which should be tuned for each dataset. In this paper, we have done a simple exploration to adjust the a and b values. Fig. 5 plots the validation accuracy and the total number of network parameters after 512 iterations for each setting. The a coefficient should be more than the b if we aim to generate a highly dense network, while more accurate networks will be designed by increasing b value. Fig. 5 clearly reflects that more accurate networks are more complex. In this paper, we have picked up two distinct settings to evaluate the effectiveness of our framework after implementation, with a=0.1; b=0.9 and a=0.9; b=0.1. To increase the speed of the procedure of designing networks, we made search decisions based on a limited number of training epochs which was set to 30, and the full training step with 250 epochs will be done after finding a satisfactory architecture.

V. Experimental Results

We have verified the real impact of the proposed framework by using a well-known dataset, CIFAR-10 [10], then we have compared the results with cutting-edge architectures. In the next, CIFAR-10 training dataset is introduced, then the classification and implementation results are presented. Our framework utilizes the Keras Library [8] for training the networks.

a) CIFAR-10: This is a complex colorful images dataset utilized as visual recognition with 32x32 pixel sizes. This benchmark contains ten output classes labeled as airplane, automobile, bird, cat, deer, dog, frog, ship, truck, and horse categories. CIFAR-10 is divided to testing and training datasets with 1000 and 50000 images, respectively. CIFAR-10 has been selected because of providing natural images of different prevalent categories.

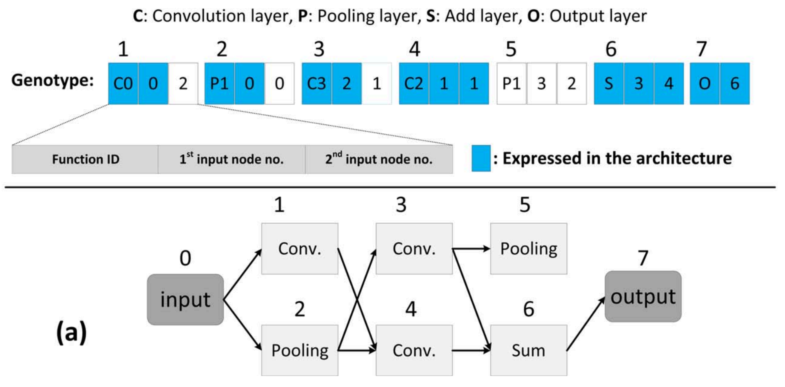
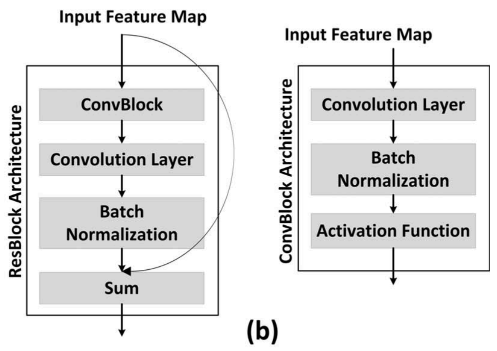

C: Convolution layer, P: Pooling layer, S: Add layer, O: Output layer
Genotype:
C0 0 2 | P1 0 0 | C3 2 1 | C2 1 1 | P1 3 2 | S 3 4 | O 6
Function ID
1st input node no.
2nd input node no.
: Expressed in the architecture
input
Conv.
Pooling
Conv.
Conv.
Pooling
Sum
output
(a)


Fig. 4. (a) The genome type example representing network architecture. (b) The architecture of ResBlock and ConvBlock.

Algorithm 1: Pseudo Code of Multi Objective CGP

```
Input: G: Max. Number of Iterations, H: Possible
       Genome parameters
Output: An Optimal Network Architectures
Function Seach($G$, $H$):
    $P_0$= Random_Parent (H); //Creating initial
      random parent
    t=1;
    while ($t < G$ ) or (achieved near-optimal net.) do
        Fitness_Function ($P_0$); //Calculating the
          fitness function of parent
        $U_0$= Force_Mutation ($P_0$);
          //Generating the offspring population by doing
          force mutation operation
        Fitness_Function ($U_0$);
        $U_1$= Force_Mutation ($P_0$);
        Fitness_Function ($U_1$);
        $P_0$= Max ($U_0$, $U_1$, $P_0$); //Replacing the best
          offspring with the parent
    return $P_0$;
```

A. Classification Results

Fig. 6 and Fig. 7 illustrate the continuous proceeding improvement of our results. Fig. 6 is the convergence diagram demonstrating the score/fitness function is approaching toward near-optimal points. Not only the fitness function, but also the accuracy and loss metrics show promising improvement by increasing the number of iterations (Fig. 7). Although we have some stops or failures in gaining better result in each iteration, the general progression of multi objective CGP always moves toward better outcomes. The near-optimal Pareto frontiers are illustrated in Fig. 8 on CIFAR-10 dataset after just five iterations. We achieved these results with the configuration of Table II, where the random initial offsprings size is equal to 50 in this test. Clearly, Pareto-optimal curves shifted toward left meaning our results have got an improved set of network architecture candidates with less error rate.

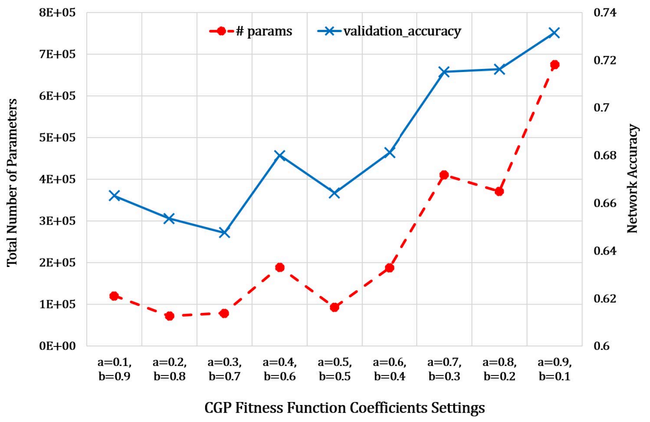


Fig. 5. Design space exploration for addressing the impact of a and b on accuracy and network parameters.

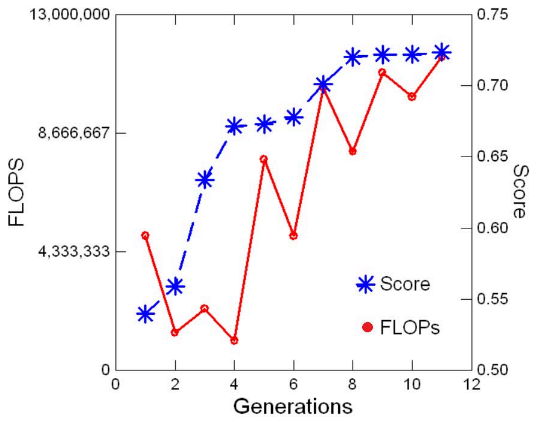


Fig. 6. Improvement proceeding of convergence metric (score) and network floating point operation for the best child in each iteration

TABLE II
THE PARAMETER RANGE FOR NODE BLOCKS AND CGP.

Training Parameters	***Epoch***	30
	Learning Rate	0.0005
	Batch Size	256
CGP Parameters	***# Columns***	30
	# offsprings	2
	# Rows	10
Network Node Blocks	***ConvBlock (a,b)***	a ∈ 32,64,128
		b ∈ 3x3,5x5
	ResBlock (a,b)	a ∈ 32,64,128
		b ∈ 3x3,5x5
	Activation	relu [31]
	Convolution_3x3 (a,s)	a ∈ 32,64
		s ∈ 1,2,3
	Pooling (Max, Average)	2x2, 3x3
a: # Output Channels; ***b***: Kernel Size; ***s***: Stride		

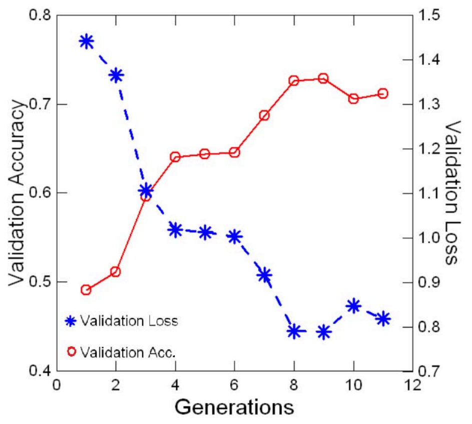


Fig. 7. Improvement proceeding of accuracy and loss for the best child in each iteration.

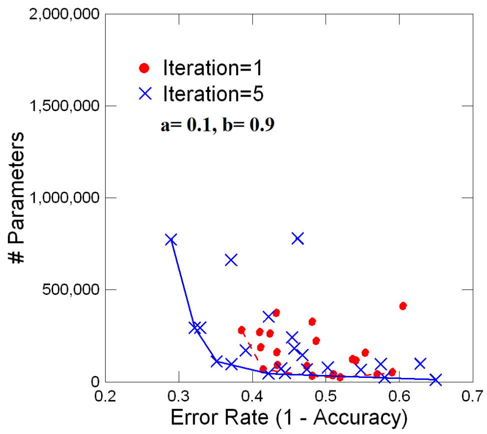


Fig. 8. Pareto frontier plots for DCNN architecture with 50 offsprings.

To evaluate the effectiveness of proposed the framework, we compared the error rate, network floating-point operations and the total number of trainable parameters of the designed networks with the other cutting-edge approaches shown in Table III. we consider the latest accurate networks as the comparison baselines listed in Table III. DCNN-Arch.1, and DCNN-Arch.2 are two architectures generated by different fitness function policies. DCNN-Arch.2 is a highly compact network, while DCNN-Arch.1 is the most accurate network generated by our framework. The main intention of this paper is not to achieve cutting-edge accuracies, but to generate a dense network with delivering competitable accuracy by profiting from imprecision tolerance nature of image processing algorithms. Our framework has a wide authority to select the most appropriate architecture based on the execution time constraints of the target hardware platform. DCNN-Arch.1 loses $\simeq$5% accuracy compared to the most accurate networks [22], while has 24x less parameters. Compare to the reinforcement learning based solutions, MetaQNN, we got better results in terms of accuracy and network size. In nutshell, our proposed framework aims to better strike the balance between network size and accuracy in comparison with the evolutionary-based solutions, hand-craft designs, and reinforcement learning-based methods. Due to requiring a huge time for running search algorithm, we limited the number of days for searching the best architecture. However, the ascending progression of improvements give auspicious news that by running the search algorithm for more days, we can get better results.

TABLE III
COMPARISON RESULTS OF ERROR RATE ON CIFAR-10 DATASET.

Dataset	**Method**	**#Params (x10^6)**	**Error (%)**
CIFAR-10	NAS-v1/v3 [17]	4.2/37.4	5.50/3.65
	SimpleNet [25]	5.48	4.68
	DenseNet (k=24)-100 [6]	27.2	5.83
	EDEN [24]	0.17	25.6
	ResNet-20 [23]	0.27	8.75
	ResNet-110 [23]	1.7	6.43
	Masanori et al. [19]	1.68	5.98
	Block-QNN-22L [18]	39.8	3.54
	MetaQNN [16]	6.92	11.18
	Real et al. [21]	5.4	5.4
	Gastaldi et al. [22]	26.4	2.86
	Our DCNN-Arch.1 (*a*=0.1; *b*=0.9)	**1.1**	**8.1**
	Our DCNN-Arch.2 (*a*=0.9; *b*=0.1)	**0.56**	**13.8**

B. Implementation Results

Three popular many/multi core platforms are used to evaluate the implementation results, NVIDIA Tesla M60 GPU, Intel Core i7-7820, and ARM Cortex-A15. Table IV summarizes the hardware specification of each one. Three different congruent networks that offer better accuracy per parameters including ResNet-20, ResNet-110, and DenseNet (k=24)-100 are chosen to compare with the DCNN-Arch.1. We also did not use any network compression technique to only assess the influence of network architecture on inference time. Keras framework automatically uses cuDNN to compile a neural network for GPUs. We only present the implementation result of DCNN-Arch.1 for the sake of brevity. Pure kernel time usually is leveraged to report runtime results, but considering the overhead of communication time is vital for embedded implementations, especially for mission critical applications since these applications are mainly latency-oriented.

TABLE IV
HARDWARE PLATFORM DETAILS.

Platform	**CPU**	**GPU**	**ARM**
Frequency (GHz)	2.9	1.178	1.9
Technology (nm)	14	28	28
TDP (W)	45	300	5
Cores/Total Thread	4/8	4096 CUDA Cores	8/8
Memory	8MB Cache	16GB GDDR5	2.5MB Cache
Approx. Price (USD)	378$	7,532$	60$/board

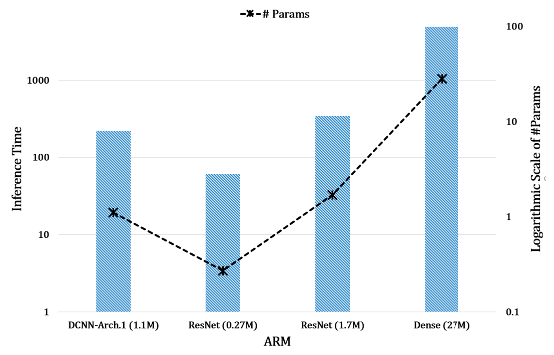


Fig. 9. Speedup of DCNN-Arch.1 in comparison to network size and accuracy on ARM platforms.

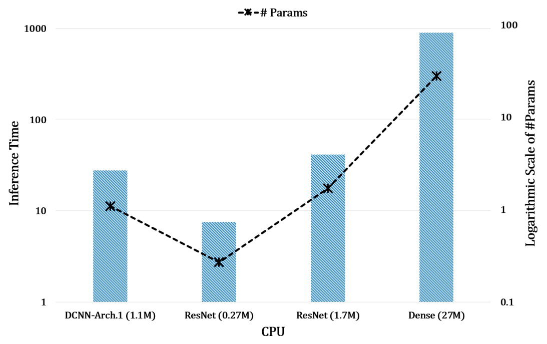


Fig. 10. Speedup of DCNN-Arch.1 in comparison to network size and accuracy on CPU platforms.

Therefore, the total execution time has been taken into account as the runtime metric. The average time is used in order to report the results and for increasing results precision, we achieved them for 10000 times. Fig. 9 to Fig. 11 plot execution time, and the logarithmic scale (to improve visual comprehension) of the number of parameters. Unlike the accuracy and the network size which are highly depend on the software stack, compiler optimizations, and hardware implementation, inference time is a platform aware factor. Thus, there is no exact similarity among different hardware platforms. The

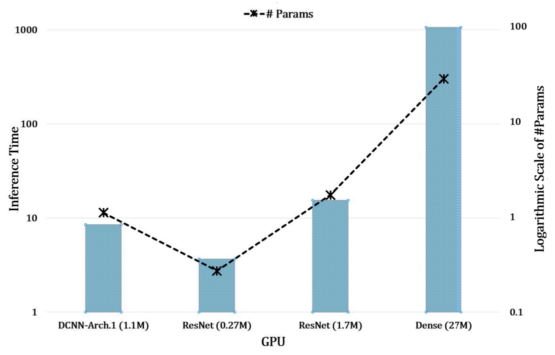


Fig. 11. Speedup of DCNN-Arch.1 in comparison to network size and accuracy GPU platforms.

results demonstrate there is a firm relation among network parameters, and inference time for each hardware platform. In nutshell, we can conclude the networks with the more number of parameters have higher inference time. In addition, our framework can adaptively find a suitable architecture regarding the timing constraints for some applications, e.g. real-time systems, by loosing negligible amount of accuracy.

C. Stereo Vision Application

A snapshot of the left camera of GIMME2 is illustrated in the Fig. 12.(a). In this configuration, we designed a DCNN for face detection. Fig. 12.(b) plots the disparity map containing distance information. Red pixels represent close distances, while blue pixels display far points. We designed a DCNN for face/gender detection by our framework implemented on the ARM processor of the Xilinx Zynq SoC. After extracting the head position in the image and matching with disparity map image, we can easily measure the distance by averaging the the distance to each pixel in the detected area. In this configuration, we set the image size receiving from camera 640x400 pixels, and 14 frame per second.

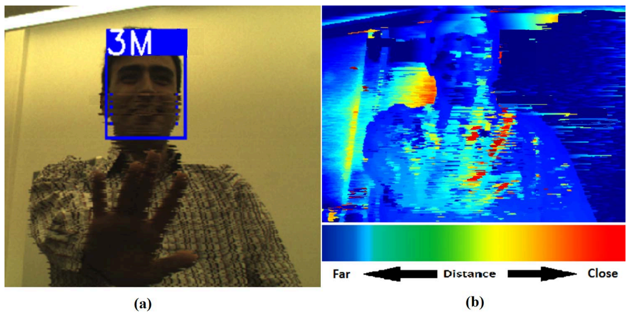


Fig. 12. (a) GIMME2 left camera snapshot. (b) Disparity map.

VI. CONCLUSION

Deep convolutional neural networks are complex processing models which their implementation is challenging especially on embedded devices. Plus, executing these models on multi-core platforms is extincted due to not providing enough computational throughput. The importance of the problem will be more highlighted when we have to deal with big data problem [33]. Stereo vision applications are one domain which suffer from the implementation barriers. To tackle these challenges, we proposed a multi objective CGP approach which automatically generates a highly optimized DCNN for commercial of-the-shelf multi/many core SoCs. Our framework aims to search the vast design space of DCNN architectures to find a network fitting with timing constraints and/or limited resource budget of embedded fabrics. We considered the total number of trainable parameters of the network as the second objective of search algorithm in order to find a highly optimized DCNN. The evaluation results demonstrate the effectiveness of the multi-objective CGP which continuously improves the network architecture in each iteration.

References

[1] C. Ahlberg, F. Ekstrand, M. Ekstrom, G. Spampinato, and L. Asplund, GIMME2 - An embedded system for stereo vision and processing of megapixel images with FPGA-acceleration, in 2015 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015, 2016.

[2] 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.

[3] 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.

[4] P. Greisen, S. Heinzle, M. Gross, and A. Burg, An FPGA-based processing pipeline for high-definition stereo video, EURASIP Journal on Image and Video Processing, vol. 2011, no. 1, 2011. [Online]. Available: http://dx.doi.org/10.1186/1687-5281-2011-18

[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] 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.

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

[9] 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.

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

[11] 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.

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

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

[14] 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.

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

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

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

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

[19] 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.

[20] M. Loni, M. Daneshtalab, and M. Sjödin, ADONN: Adaptive Design of Optimized Deep Neural Networks for Embedded Systems, Euromicro Conference on Digital System Design (DSD), Prague, Czech, 2018.

[21] 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.

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

[23] 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.

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

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] P. Ramachandran, B. Zoph, and Q. V. Le, Searching for Activation Functions, arXiv prepr. arXiv:1710.05941, 2017.

[32] J. F. Miller, Cartesian Genetic Programming, in Cartesian Genetic Programming, 2011, pp. 1734.

[33] 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.

[34] A. Majd, G. Sahebi, M. Daneshtalab, J. Plosila, S. Lotfi, and H. Tenhunen, Parallel imperialist competitive algorithms, in Concurrency Computation, 2018, vol. 30, no. 7.