NeuroPower: Designing Energy Efficient Convolutional Neural Network Architecture for Embedded Systems

Mohammad Loni[1⋆], Ali Zoljodi[2], Sima Sinaei[1], Masoud Daneshtalab[1], and Mikael Sjödin[1]

[1] School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
{mohammad.loni, sima.sinaei, masoud.daneshtalab, mikael.sjodin}@mdh.se
[2] Shiraz University of Technology, Shiraz, Iran
ali.zoljodi@sutech.ac.ir

Abstract. Convolutional Neural Networks (CNNs) suffer from energy-hungry implementation due to their computation and memory intensive processing patterns. This problem is even more significant by the proliferation of CNNs on embedded platforms. To overcome this problem, we offer NeuroPower as an automatic framework that designs a highly optimized and energy efficient set of CNN architectures for embedded systems. NeuroPower explores and prunes the design space to find improved set of neural architectures. Toward this aim, a multi-objective optimization strategy is integrated to solve Neural Architecture Search (NAS) problem by near-optimal tuning network hyperparameters. The main objectives of the optimization algorithm are network accuracy and number of parameters in the network. The evaluation results show the effectiveness of NeuroPower on energy consumption, compacting rate and inference time compared to other cutting-edge approaches. In comparison with the best results on CIFAR-10/CIFAR-100 datasets, a generated network by NeuroPower presents up to 2.1x/1.56x compression rate, 1.59x/3.46x speedup and 1.52x/1.82x power saving while loses 2.4%/-0.6% accuracy, respectively.

Keywords: Convolutional neural networks (CNNs) · Neural Architecture Search (NAS) · Embedded Systems · and Multi-Objective Optimization

1 Introduction

CNNs have penetrated in a wide spectrum of platforms from workstations to embedded devices due to influential learning capabilities. However, modern CNN architectures are becoming more complex to provide superior accuracy leading to remarkable energy consumption. Dealing with huge computing throughput demand of upcoming complex learning models will be more critical where the failure of traditional energy and performance scaling paradigm in affording of modern applications requirements leads computing landscape towards inefficiency [4]. Approximate computing is one possible propitious alternative to cope with these challenges by amortizing outputs quality of imprecision-tolerant applications such as objects recognition, image processing, and

⋆ Corresponding Author

data analytics. Generally, three different approximation based strategies are proposed to diminish CNN computational complexity and/or improve energy saving: ① pruning network weights and quantization [7,16], ② employing customized hardware accelerators [28], and ③ optimizing network architecture at design-time since the performance (energy consumption and inference time) and output quality of CNNs are immensely affected by network architecture [18–20].

To benefit from these approaches, we propose NeuroPower, a CNN acceleration framework aiming to automatically explore the design space in order to design an energy efficient CNN architecture. NeuroPower solves the NAS problem and explore the design space considering better accuracy level and less network architectural complexity as the optimization objectives. Previous NAS solutions mainly focus on improving the network accuracy, while NeuroPower considers network architectural complexity, represented by the number of parameters in the network, as the second optimization objective since there is a strong correlation between energy consumption and network architectural complexity (see Section 3.1). For this, NeuroPower is equipped with a neuro-evolutionary Multi-Objective Optimization (MO^2) mechanism which produces a set of Pareto-optimal curves wherein each point on the curve is a vector with elements of the CNN hyperparameters. NeuroPower adaptively selects a suitable CNN architecture regarding power budget limitations and/or response-time of embedded hardware platform. Network pruning is a popular solution for diminishing the amount of network computation. In addition to design space exploration, NeuroPower can apply a network pruning method on a dense architecture to achieve further level of network optimization. In order to guarantee designing a light-weight architecture and to boost the optimization process, the design space has been trimmed by taking inspirations from DenseNet architecture [13].

Fig. 1.a illustrates an overview of the proposed NeuroPower framework. The MO^2 starts exploring design space after setting predefined learning and optimization parameters. Displayed NeuroPower controller in Fig. 1.a verifies the optimization termination condition by getting energy consumption and/or inference time of candidates. The optimization procedure will be continue until satisfying user criteria or the maximum number of iterations is reached. To verify the impact of NeuroPower on energy consumption and inference time, three COTS embedded platforms are utilized including a many-core NVIDIA Quadro K5100M GPU, a multi-core high-performance Intel Core processor i7-4940MX, and an ARM Cortex-A15. ARM architecture is one of the immensely popular embedded processors due to low power consumption, and providing reasonable performance. However, ARM is not suitable to process computational intensive CNN models. On the other hand, GPUs are popular performance-centric accelerators for machine learning applications refereed as another possibility to deal with reducing efficiency trend in the multi-core era. Although GPUs offer a higher level of programmability and memory bandwidth, they suffer from huge power consumption [28]. To tackle these challenges, our proposed framework demonstrates considerable performance gain over GPU, high-performance Intel CPU, and embedded ARM processor. In a nutshell, our main contributions in NeuroPower are:

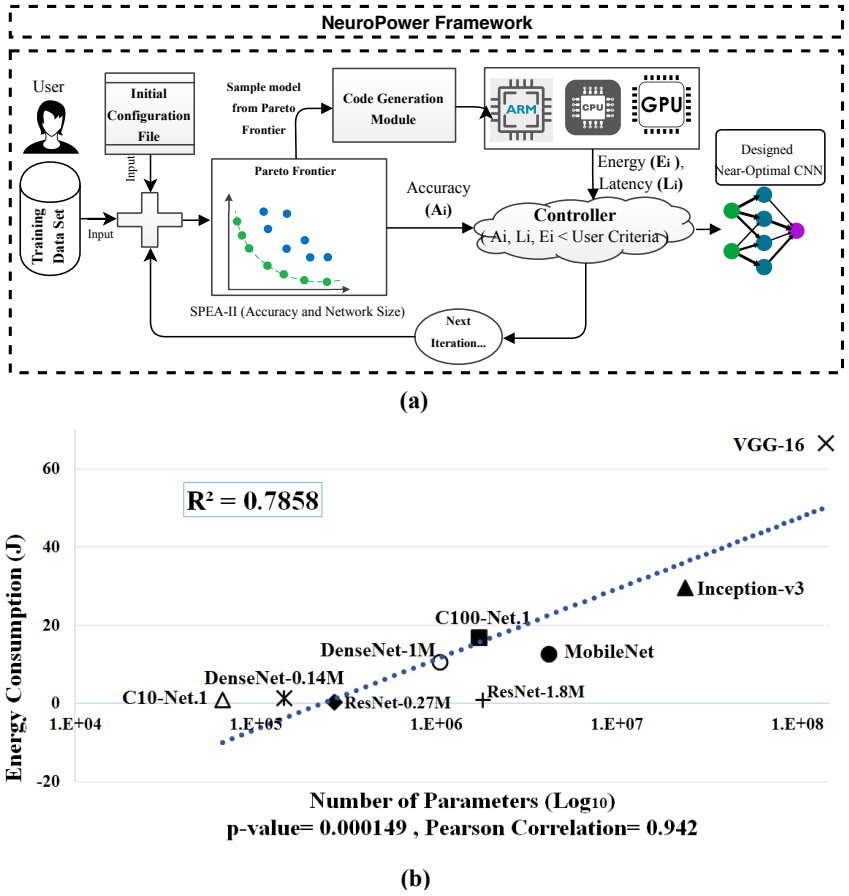

Fig. 1. (a) The overview of NeuroPower framework, (b) Energy consumption vs. # of network parameters.

- Presenting a meta-heuristic MO^2 method to explore energy aware CNN architectures. NeuroPower generates a set of network architectures optimized for a wide range of architectural complexities fitting different hardware resource budgets.
- Proposing novel activation functions which significantly increase the network accuracy.
- Developing a cutting-edge network pruning method on the neural network architecture to obtain less complex network with acceptable accuracy.

The remainder of this paper is organized as follows: Section 2 reviews related work in this scope. Section 3 gives background on CNN and the MO^2 algorithm. Details of the proposed framework are presented in Section 4 which consist of two solutions for network optimization: Design Space Exploration and Design Space Pruning. The experimental results are presented in Section 5, after which Section 6 concludes the paper.

2 Related Work

In the field of neural network design space exploration, different automated NAS approaches have been proposed including Bayesian optimization, Reinforcement Learning (RL), and neuro-evolutionary methods. Bayesian-based methods suffer from immense computational cost, suitable only for search models with a fixed-length space, and focuses on low-dimensional continuous problems [3, 18]. RL-based NAS methods provides high-quality results for image classification applications compared to the best hand-crafted CNN accuracy results [2, 29, 30, 33]. Despite their success, these models are mainly slow and require considerable computational resources in both exploration and training steps [2]. The most of the neuro-evolutionary methods leverage evolutionary algorithms for optimizing the neural architecture by evolving a population of improved candidates [18–20, 22, 25]. There has been proposed other multi-objective neuro-evolutionary frameworks [18–20] considering the number of network parameters as the second optimization objective. NeuroPower is more efficient compare to [18, 19] in terms of both compression rate and exploration time. Plus, NeuroPower can generate more compact architectures compared to [20], while takes roughly equal exploration time.

Moreover, As mentioned in previous section, the proposed framework has the ability of design space pruning to obtain less complex network with acceptable accuracy. In the field of neural network design space pruning also several methods have been presented. In [16] a method is proposed that prune CNN filters in two levels. It first clusters network filters by enforcing the K-means algorithm, then retain the filter which is the closest to the cluster center and pruning some of the others randomly. In [24] a data-free approach is proposed to carry out CNN model compression. They managed to avoid employing any training data by minimizing the expected squared difference of logits. In [8] a pruning approach by applying L1/L2-norm regularizations is introduced to remove the small weights. Although the performance is inspiring, the pruning would result in unstructured patterns in weights connectivity. Research in [17] tries to select the best filters for pruning, for example, uses absolute weight summation to evaluate the impact of a filter. In this method very low differences in weights are affected too much, thus, the work presented in [11] introduced Average Percentage of Zeros to assess the importance of each filter. This work needs lots of extra calculations and the compression ratio is not satisfying.

3 Background

3.1 An Overview of CNNs

A typical CNN is composed of multiple layers running in sequence, where input data is fed to the first layer and output is a series of feature extraction kernels applied on the input image. Convolution, normalization, pooling, and activation layers are responsible for feature extraction, while fully-connected layers are in charge of classification. VGG-16 [23] is a well-known CNN containing 13 and 3 convolutional and fully-connected layers respectively. Computational analysis of VGG-16 demonstrates that convolutional layers are extremely computation intensive containing 99.3% of the total computation

while fully-connected layers are memory-intensive which utilize more than 80% of data movements [18]. Thereby, for optimizing architectural complexity of a CNN, convolutional parameters including *the number of convolutional layers, the sizes of each layer, activation function, convolutional filter size*, and *learning rate* should be considered as the networks optimization hyperparameters.

Table 1. The CNN hyperparameters used as exploring neural design space parameters.

Parameter	Value
Activation Function (Left, Right) **[21]**	Relu, Elu, Sigmoid, Tanh, Swish, Selu, Linear
# Condense Layer	1, 2, 3, 4
# Feature Extraction Layer (FEL)	16, 28, 40, 52
Kernel Size	3x3, 5x5
Optimizer **[6]**	Adam, SGD, Adagrad, Adamax, Nadam

Table 1 lists considered hyperparameters and their corresponding values where the value range of each parameter has been limited to prune the design space, moreover, huge hyperparameter values does not always provide highly accurate networks [3]. Based on practical evaluations, these selected hyperparameters strongly influence accuracy and inference time [18, 19]. In addition, we realized a strong relationship between energy consumption/inference time and the number of parameters of a CNN. Fig. 1.b illustrates the relationship between energy consumption per each forward query and the number of parameters executed on an NVIDIA Quadro K5100M GPU. The results are plotted in the logarithmic scale to improve visual comprehension. These results imply that the number of parameters in a network is a strong proxy for network architectural complexity [18, 20, 27]. While the main focus of this paper is on diminishing CNN power consumption, the experimental results indicate that NeuroPower efficiently decreases inference time (see Section 5).

3.2 Strength Pareto Evolutionary Algorithm-II (SPEA-II)

to make the best balance between the network accuracy and architectural complexity, an optimization approach is needed. Computability is highly challenging especially in complex problems since there is no guarantee that NP-hard complex problems such as NAS problem can be solved in a satisfactory manner in a limited time. To improve solving such problems several techniques have been proposed. Among them, Evolutionary Computing (EC) methods are more prominent [18, 22, 25]. EC comprises a set of optimization algorithms mimicking the survival of the nature fittest principle, as some characteristics of this process can be utilized in optimization problems. Strength Pareto Evolutionary Algorithm-II (SPEA-II) [32] is a powerful meta-heuristic EC solving MO^2 problems. In this work, accuracy and the number parameters are considered as the optimization objectives. SPEA-II provides slightly superior optimization mechanism compared to NSGA-II [18, 31] by obtaining more diverse solutions in the archi-

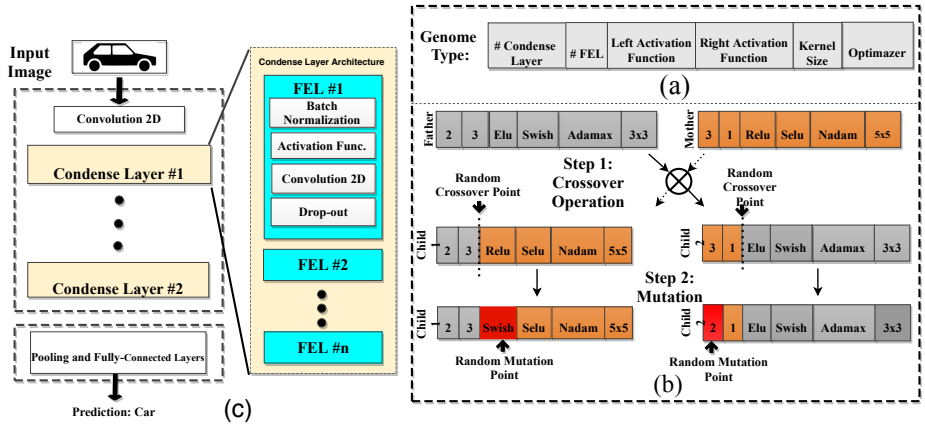

Fig. 2. (a) A genome type representing NAS hyperparameters. (b) Crossover & forced mutation operators between two genomes for SPEA-II optimization algorithm. (c) Inspired template architecture of a generated network.

tectural design space of CNNs. Algorithm 1 represents the pseudo-code of NeuroPower framework integrating SPEA-II as the optimization engine. SPEA-II is explained as following steps: ***Step 0:*** Creating an initial population U_0 with size N, and Y_0 as a null population. To generate the initial random population, network hyperparameters are represented as a string of genomes using direct encoding shown as genome type in Fig. 2.a. ***Step 1:*** Calculating the fitness values of individuals in the U_0 and Y_0. The fitness function is described in (1) as follows;

$$Score = \frac{Net\_Acc}{\#Net\_Params} \tag{1}$$

where Net_Acc is network accuracy and $\#Net\_Params$ is the number of network parameters. The $Score$ factor is SPEA-II fitness function for selection process, where architectures with higher accuracy and smaller network parameters are desirable. ***Step 2:*** $Y_{t+1}=\{x_i \mid x_i \in \{Y_t \cup U_t\}\ AND\ x_i\ is\ a\ nondominant\ individual\}$, and adjusting Y_{t+1} size to N. ***Step 3:*** The NeuroPower controller terminates optimization procedure if one of the individuals in Y_{t+1} satisfies user criteria or maximum number of iterations is reached. otherwise, the procedure progresses as long as satisfying termination condition. ***Step 4:*** Put individuals from Y_{t+1} into the mating pool with substituted binary championship rule. ***Step 5:*** Applying $Crossover$ and $ForcedMutation$ operators on the mating pool individuals to create the next population U_{t+1}. Network hyperparameters are represented as a genomes string and the recombination of these genes occurs with one-point crossover and one-point mutation operators shown in Fig. 2. We defined the new $ForcedMutation$ operator to change at least one hyperparameter node in genome type while guarantee the probability of evaluating equal architectures being zero. $ForcedMutation$ operator improves the exploration capability due to pushing the SPEA-II to find new candidates.

Algorithm 1: Pseudo Code of NeuroPowers Design Space Exploration

Input: **N**: Archive Size, **G**: Max. Number of Iterations, **H**: Hyperparameters List, $\mathbf{Eng_{user}}$, $\mathbf{Acc_{user}}$: User Criteria
Output: A Non-Dominated Set of Optimal Architectures on Pareto Frontier
Function `Optimization_Engine(`N, G, H`)`**:**
  Step 0: U_0= **Random_Population** (N, H); Y_0= **Empty_Population**;
  $t = 0$; //Iteration Number
  while *True* **do**
    Step 1: **Fitness_Values_Calculation** (U_t, Y_t, N);
    Step 2: Y_{t+1}=**Environmental_Selection** (U_t, Y_0, N);
    Step 3: **if** $(\exists i \in Y_{t+1} \mid E_i, I_i, A_i \neg satisfies\ Eng_{user}, Acc_{user})$ **then**
      $BreakWhileLoop$; // Terminate NeuroPower
    Step 4: P_t=**Mating_Selection** (Y_{t+1});
    Step 5: U_{t+1}=**Crossover&Forced_Mutation** (P_t);
    $t = t + 1$;
  return Y_{t+1};

4 NeuroPower: The Proposed Framework

4.1 Design Space Exploration (DSE) Algorithm

The NeuroPower framework consists of a controller, optimization engine and code generation module (Fig. 1.a). Predefined parameters of NeuroPower are specified in the configuration file including optimization and network training parameters such as number of epochs, learning rate, and valid range of network hyperparameters. After providing the input dataset and initiating the configuration file by user, the engine function will start to explore the design space of CNN architectures. At the end of each iteration, the energy consumption and inference time of each individual on Pareto curve will be measured by passing the network configuration to code generation module to generate specific run-time execution code for each platform. The code generation module uses Tensorflow library [1] to automatically generate kernel code for NVIDIA GPUs. Then, the controller checks whether the designed architecture satisfies user criteria or not. In the case of non-satisfaction, the optimization module will be called again to find the next iteration of the optimized solutions (Algorithm 1). SPEA-II explores the design space over a template architecture inspired from DenseNet for decreasing the probability of generating giant architectures. **Template Architecture:** The template consists of multiple Condense Layers where each Condense Layer contains back-to-back Feature Extraction Layers (FELs) [18]. FEL includes *Batch Normalization, Activation Function, 2D Convolution, Drop-out*, respectively. Obviously, for classification max-pooling and fully-connected layers are integrated as the last layers with softmax activation function. To share maximum learned knowledge between layers, all the layers are connected in a feed-forward manner to each other such that each layer receives the additional feature map information from the whole former layers in the Condense Layer and using concatenation layer to merging shared data.

4.2 Design Space Pruning Algorithm

In general, neural network pruning techniques try to reduce the storage and computation required by neural networks without considerable affecting on the network accuracy by learning the influential weights. To take advantages of network pruning on the designed architectures by NeuroPower, we proposed a pruning method which basically uses the idea presented in [16]. This technique tries to select and remove redundant filters which affected zero or very low in the network results by utilizing K-means++ algorithm for selecting appropriate filters for pruning. The proposed pruning algorithm works as follows: First, it employs the K-means++ algorithm to enforce the filters to enter specific clusters. Second, it will retain the filter which is the closest to the cluster center and prune the others in every cluster. Then the pruned model will be fine-tuned to recover accuracy.

5 Experimental Results

In this section the experimental results of design space exploration and design space pruning of the proposed framework are presented respectively. NeuroPower framework has been evaluated using well-known datasets and compare with cutting-edge architectures. The experiments have been performed on the following data sets: MNIST [15], CIFAR-10 [14] and CIFAR-100 [14].

5.1 Training Datasets

MNIST [15]: This is a dataset of black and white images for handwritten digit recognition containing 60,000 training and 10,000 testing images. Each image is a 28x28 pixels with ten labeled output as 0 to 9 numbers. ***CIFAR-10 [14]:*** This is a complex colorful dataset of natural images, each with 32x32 pixels which is mainly used for object recognition. This benchmark contains ten labeled output classes containing 50000 and 1000 images for training and testing, respectively. ***CIFAR-100 [14]:*** CIFAR-100 is similar to CIFAR-10, but with 100 classes while each class has 500 instead of 5,000 as in CIFAR10 making the classification more challenging.

5.2 Design Space Exploration Results

The design space roughly consists of 8000 different design points. The average training time is one hour for each design point by employing NVIDIA GTX 1080ti. Thus, leveraging exhaustive search is not reasonable since exploration takes 8000 GPU-hours, while NeuroPower required 360 GPU-hours for generating the experimental results demonstrating 22x reduction in exploration time. NeuroPower trains each network during exploration step using only 16 epochs since approximately 90% of the maximum achievable accuracy is obtained after 16 epochs [18]. The full training step will be applied to only the best selected architecture with 250 epochs. The NeuroPower's configuration file was set with the following parameters: batch size=128, maximum number of iterations=8, and random initial population=45.

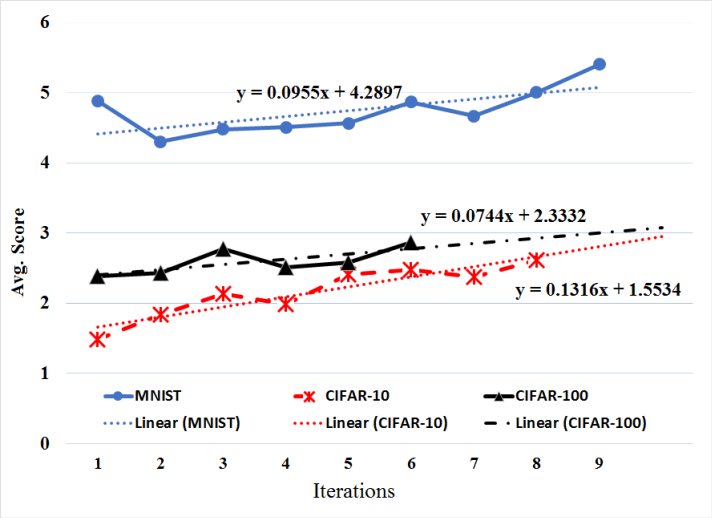

Fig. 3. Score convergence for MNIST, CIFAR-10 and IFAR-100 datasets. The linear equations demonstrate the overall improvement of the $Score$ fitness function over proceeding iterations.

Table 2 presents network designing strategy, error rate, number of parameters and network compression rate of solutions generated by NeuroPower compared to the other cutting-edge approaches. For MNIST dataset, M_Net_1 generated by Neuropower is 2x more compressed (with 0.3% accuracy loss) in comparison with a highly optimized network ADONN-Arch.3 [18]. C10_Net.1, C10_Net.2, and C10_Net.3 are different nodes of Pareto frontier selected from seventh iteration for CIFAR-10 dataset. C10_Net.1 is the most dense architecture provides up to 47.7x compression rate with 13% accuracy loss compared to a cutting-edge squeezed network, named CondenseNet^{Light} [12]. C10-Net.3 provides the best accuracy for CIFAR-10 dataset which is 3.1x compressed with loosing only 3% accuracy. The compression rate of C100-Net.1 and C100-Net.2 architectures for CIFAR-100 dataset is not significant compare to the other solutions, but NeuroPower still provides more accurate networks. In nutshell, NeuroPower strikes better balance between network accuracy and network size compare to RL and EC strategies and hand-crafted designs.

Fig. 3 illustrates the continuous proceeding improvement of the results and demonstrates that the convergency diagram guaranteeing the score/fitness function is approaching toward near-optimal points for the considered datasets. Fig. 4 illustrates the tuning of piece-wise activation functions toward designing more accurate architectures for CIFAR-10 dataset. Obviously, Swish is the dominant activation function for the input data greater than zero (right activation function) in initial iterations, however, Relu was replaced after fifth iteration meaning that NeuroPower finds Relu a superior option. However, Swish strongly overcomes other activations functions for the input data less than zero (left activation function) for all iterations.

Table 2. Error rate and Compression rate for different datasets.

Dataset	Approach	Solutions	#Params ($\times 10^6$)	Error Rate (%)	Compression Rate* ‡
	Hand-Crafted	Wan et al. [26]	-	0.21	-
	RL	MetaQNN [2]	5.59	0.35	0.023x
MNIST	MO^2-EC	* ADONN-Arch.3 [18]	0.13	0.41	-
	MO^2-EC	**Our M_Net_1**	**0.065**	**0.71**	**2x**
	Hand-Crafted	* CondenseNet$^{\text{Light}}$ [12]	3.1	3.46	-
	Hand-Crafted	SimpleNet [9]	5.48	4.68	0.53x
	Hand-Crafted	DenseNet (k=12)-40 [13]	1.0	7.0	3.1x
	Hand-Crafted	ResNet-20 [10]	0.27	8.75	11.48x
	Hand-Crafted	ResNet-110 [10]	1.7	6.43	1.82x
	Hand-Crafted	Gastaldi et al. [5]	26.4	2.86	0.117x
	RL	Block-QNN-22L [30]	39.8	3.54	0.078x
CIFAR-10	RL	MetaQNN [2]	6.92	11.18	0.45x
	RL	NAS-v1/v3 [33]	4.2/37.4	5.50/3.65	0.7x/0.083x
	RL	Block-QNN-S [30]	6.1	4.38	0.5x
	EC	Real et al. [22]	5.4	5.4	0.57x
	MO^2-EC	NSGA-Net [20]	3.3	3.85	0.94x
	MO^2-EC	ADONN-Arch.3 [18]	0.14	14.1	22.14x
	MO^2-EC	Loni et al. [19]	0.56	13.8	5.5x
	MO^2-EC	**Our C10-Net.1**	**0.065**	**16.49**	**47.7x**
	MO^2-EC	**Our C10-Net.2**	**0.21**	**11.05**	**14.7x**
	MO^2-EC	**Our C10-Net.3**	**1.0**	**6.81**	**3.1x**
	RL	MetaQNN [2]	11.18	27.14	0.28x
	RL	Block-QNN-S [30]	6.1	20.65	0.5x
	Hand-Crafted	* CondenseNet$^{\text{Light}}$ [12]	3.1	17.55	-
	Hand-Crafted	DenseNet (k=12)-40 [13]	1.0	27.55	3.1x
CIFAR-100	Hand-Crafted	DenseNet (k=12)-100 [13]	7.0	23.79	0.44x
	Hand-Crafted	SimpleNet [9]	5.48	26.58	0.53x
	MO^2-EC	NSGA-Net [20]	3.3	20.74	0.94x
	MO^2-EC	**Our C100-Net.1**	**1.1**	**26.63**	**2.82x**
	MO^2-EC	**Our C100-Net.2**	**1.89**	**24.87**	**1.64x**

* The baseline for comparing the compressing rate.
‡ The values more than 1.0 indicate improvement. Best results are in **bold**.

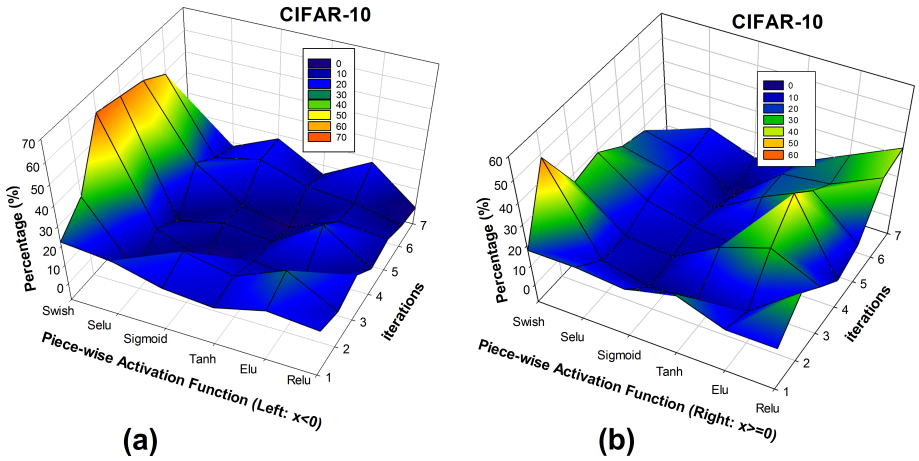

Fig. 4. The percentage of participation of (a) Left activation functions and (b) Right activation functions in the population for the CIFAR-10 dataset.

Table 3 presents the hardware implementation results for MNIST, CIFAR-10 and CIFAR-100 datasets. For each dataset, the architecture with highest *Network-Information-Density* (NID) is employed as the baseline of the comparisons. NID (accuracy per parameters) is a yield factor highlighting that the capacity of an architecture to utilize better its parametric space [27]. In order to measure energy consumption of the tested platforms, we used the NVIDIA Management Library (NVML), Intel Running Average Power Limit (RAPL) and kill-a-watt P4400 device to obtain the average power of GPU, Intel CPU and ARM during benchmark execution. In this paper, the kernel time is used for reporting run-time results, however, the overhead of communication time should be considered for embedded implementations, especially for mainly latency-oriented. Therefore, To demonstrate the positive impact of squeezing network architecture on GPU Device-to-Host (D2H) and Host-to-Device (H2D) data communications, the achieved data communication speedup after leveraging NeuroPower is reported in Table 3.

We can conclude: ① the networks with more NID provide higher energy efficiency. ② For CIFAR-10 and CIFAR-100 our architectures did not obtain maximum achievable energy efficiency, compare to the ResNet architecture. However, if we take the total execution time (kernel + communication) into account, we can achieve higher speedup. In Addition, the obtained energy efficiency results of our designed architectures are showing remarkable improvement on the ARM and CPU platforms. ③ Compacting an architecture potentially diminish the overhead of communication time (D2H/H2D) since less number of data packets need to be copied via PCI-Express Bus. We obtain 1.7x and 1.9x speedup on average for D2H and H2D data communications, respectively.

Table 3. Implementation results for different platforms

Platform			GPU			Intel ® CPU		ARM Processor	
Dataset	Network	NID ($\times 10^6$)	Energy Efficiency	Speedup (Kernel)	Speedup: D2H/H2D (Comm.)	Energy Efficiency	Speedup (Inference Time)	Energy Efficiency	Speedup (Inference Time)
	* ADONN-Arch.3 [18]	7.06	-	-	-	-	-	-	-
MNIST	**M_Net_1**	**15.25**	**1.55x**	**1.49x**	**1.59x/2.95x**	**1.49x**	**1.51x**	**1.56x**	**1.38x**
	ResNet-20 [10]	3.37	4.2x	4.25x	0.62x/0.79x	0.75x	0.81x	1.05x	0.87x
	ResNet-110 [10]	0.55	1.53x	1.64x	0.108x/0.15x	0.093x	0.95x	0.16x	0.87x
CIFAR-10	DenseNet (k=12)-100	0.135	0.25x	0.027x	0.05x/0.073x	0.099x	0.112x	0.07x	0.115x
	* ADONN-Arch.3 [18]	6.13	-	-	-	-	-	-	-
	C10_Net_1	**13**	1.54x	1.51x	**1.61x/1.57x**	**1.5x**	**1.59x**	**1.52x**	**1.32x**
	C10_Net_3	0.93	0.14x	0.154x	0.25x/0.313x	0.3x	0.372x	0.3x	0.37x
	* ResNet-110 [10]	0.43	-	-	-	-	-	-	-
CIFAR-100	DenseNet (k=12)-100	0.109	0.017x	0.016x	0.44x/0.46x	0.92x	1.03x	0.34x	0.74x
	C100-Net.1	**0.66**	0.11x	0.1x	**2.28x/2x**	**3.2x**	**3.46x**	**1.89x**	**2.38x**
	C100-Net.2	**0.4**	0.06x	0.06x	**1.45x/1.39x**	**2.45x**	**2.53x**	**1.35x**	**1.83x**

* The baseline for comparing the energy efficiency of different architectures. The values more than 1.0 indicate improvement. Best results are in **bold**.

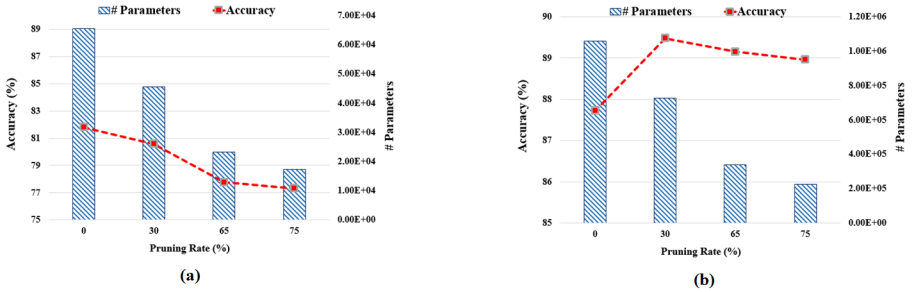

Fig. 5. The impact of the network pruning on the accuracy level of (a) C10-Net.1 and (b) C10-Net.3.

5.3 Pruning Results

The pruning method has been evaluated on two different architectures designed for CIFAR-10 dataset including C10-Net.1 and C10-Net.3. The first architecture is a compressed architecture with 0.065 million parameters, while the second one is a larger architecture with 1 million parameters. First, these two architectures are trained separately with 200 epochs and the accuracy rates are obtained 81.79% for C10-Net.1 and 87.72% for C10-Net.3. Then, the filter pruning technique is used by assuming cluster factor =0.9 and number of fine tune epochs=5 and pruning iteration=10 as the constant configuration and maximum pruning percent is equal to 30, 65, 75 as the threshold on weight pruning for both studied architectures. Fig. 5 illustrates the impact of the network pruning on the accuracy level. The number of networks parameters decreases with increasing the pruning rate and the NID of pruned architectures increases with increasing maximum pruning percent. Obviously, the impact of pruning on the accuracy rate of larger architecture, C10-Net.3, is much better than the other one. However, the denser architecture still provides 3.1x higher NID level compared to the pruned architecture of the large network with 75% pruning rate.

6 Conclusion

In this paper, we proposed the NeuroPower framework which automatically generates a highly-optimized CNN for commercial embedded devices. In the presented framework, two algorithms are proposed for design space exploration and design space pruning. NeuroPower alleviates the huge computational cost of CNNs by squeezing the network architecture while delivers acceptable accuracy level. In order to achieve an energy efficient network, a novel fitness function is used to consider energy consumption during exploration procedure. Experimental results show that, in comparison with the best results on CIFAR-10/CIFAR-100 datasets, NeuroPower presents up to 1.59x/3.46x speedup and 1.52x/1.82x power saving while loses 2.4%/-0.6% accuracy, respectively.

7 Acknowledgment

This Paper is supported by KKS within DeepMaker and DPAC projects.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 265–283 (2016)
2. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167 (2016)
3. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. arXiv preprint arXiv:1808.05377 (2018)
4. Esmaeilzadeh, H., Blem, E., Amant, R.S., Sankaralingam, K., Burger, D.: Power challenges may end the multicore era. vol. 56, p. 93 (2013). https://doi.org/10.1145/2408776.2408797, http://dl.acm.org/citation.cfm?doid=2408776.2408797
5. Gastaldi, X.: Shake-shake regularization. arXiv preprint arXiv:1705.07485 (2017)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
7. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. pp. 1135–1143 (2015)
8. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. pp. 1135–1143 (2015)
9. Hasanpour, S.H., Rouhani, M., Fayyaz, M., Sabokrou, M.: Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. arXiv preprint arXiv:1608.06037 (2016)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
11. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016)
12. Huang, G., Liu, S., Van der Maaten, L., Weinberger, K.Q.: Condensenet: An efficient densenet using learned group convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2752–2761 (2018)

13. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 (2017). https://doi.org/10.1109/CVPR.2017.243
14. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 and cifar-100 datasets. URl: https://www. cs. toronto. edu/kriz/cifar. html **6** (2009)
15. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
16. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
17. Li, L., Xu, Y., Zhu, J.: Filter level pruning based on similar feature extraction for convolutional neural networks. IEICE TRANSACTIONS on Information and Systems **101**(4), 1203–1206 (2018)
18. Loni, M., Daneshtalab, M., Sjodin, M.: ADONN: Adaptive design of optimized deep neural networks for embedded systems. In: Proceedings - 21st Euromicro Conference on Digital System Design, DSD 2018 (2018). https://doi.org/10.1109/DSD.2018.00074
19. Loni, M., Majd, A., Loni, A., Daneshtalab, M., Sjodin, M., Troubitsyna, E.: Designing compact convolutional neural network for embedded stereo vision systems. In: Proceedings - 2018 IEEE 12th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2018 (2018). https://doi.org/10.1109/MCSoC2018.2018.00049
20. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: Nsga-net: a multi-objective genetic algorithm for neural architecture search. arXiv preprint arXiv:1810.03522 (2018)
21. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions. arXiv preprint arXiv:1710.05941 (2017)
22. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2902–2911. JMLR. org (2017)
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
24. Srinivas, S., Babu, R.V.: Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149 (2015)
25. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 497–504. ACM (2017)
26. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: Regularization of neural networks using dropconnect. In: International conference on machine learning. pp. 1058–1066 (2013)
27. Wong, A.: Netscore: Towards universal metrics for large-scale performance analysis of deep neural networks for practical usage. arXiv preprint arXiv:1806.05512 (2018)
28. Yazdanbakhsh, A., Park, J., Sharma, H., Lotfi-Kamran, P., Esmaeilzadeh, H.: Neural acceleration for GPU throughput processors (2016). https://doi.org/10.1145/2830772.2830810
29. Zhong, Z., Yan, J., Liu, C.L.: Practical network blocks design with q-learning. arXiv preprint arXiv:1708.05552 **1**(2), 5 (2017)
30. Zhong, Z., Yan, J., Wu, W., Shao, J., Liu, C.L.: Practical block-wise neural network architecture generation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2423–2432 (2018)
31. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation **8**(2), 173–195 (2000)
32. Zitzler, E., Laumanns, M., Thiele, L.: Spea2: Improving the strength pareto evolutionary algorithm. TIK-report **103** (2001)
33. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)