Automated Circuit Approximation Method Driven by Data Distribution

Zdenek Vasicek, Vojtech Mrazek and Lukas Sekanina
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno, Czech Republic
Email: {vasicek, imrazek, sekanina}@fit.vutbr.cz

***Abstract*—We propose an application-tailored data-driven fully automated method for functional approximation of combinational circuits. We demonstrate how an application-level error metric such as the classification accuracy can be translated to a component-level error metric needed for an efficient and fast search in the space of approximate low-level components that are used in the application. This is possible by employing a weighted mean error distance (WMED) metric for steering the circuit approximation process which is conducted by means of genetic programming. WMED introduces a set of weights (calculated from the data distribution measured on a selected signal in a given application) determining the importance of each input vector for the approximation process. The method is evaluated using synthetic benchmarks and application-specific approximate MAC (multiply-and-accumulate) units that are designed to provide the best trade-offs between the classification accuracy and power consumption of two image classifiers based on neural networks.**

I. Introduction

Approximate computing exploits the fact that there are many *error-resilient* applications (such as image recognition, video processing and data mining) in which *quality of service* can be traded for performance or power consumption. Adopting the principles of approximate computing thus enables to significantly improve energy efficiency of complex computer applications. In order to obtain an approximate implementation, a common practice is to replace selected components of the original (exact) implementation by their approximate versions. For this purpose, approximate components based on various approximation principles have been introduced (for example, see a recent survey of approximate adders and multipliers [1]). Even open circuit libraries nowadays provide various sorts of approximate circuits [2], [3]. However, it is important to emphasize that these circuits have (almost always) been optimized with respect to general-purpose error metrics and evaluated under the assumption of uniformly distributed input values. Applying these prefabricated approximate circuits can bring some improvements in power consumption or performance, but much better trade-offs are always obtained if the approximate circuit is deliberately developed and optimized for a given application and if it exploits some knowledge about the application, for example, a particular (non-uniform) distribution of input vectors.

These application-specific approximate circuits (ASACs) can, in principle, be obtained using automated circuit approximation methods such as ABACUS [4], SALSA [5], CGP [6], [3] etc. if a suitable error metric is provided. Contrasted to the manual approximation approach (represented by, e.g., [1]) these methods automatically generate and evaluate candidate designs until the implementation showing acceptable trade-offs between design objectives is obtained. Let us consider an example in which the objective is to create highly efficient approximate multipliers for an image classifier based on a Convolutional Neural Network (CNN). Papers [6], [7] already shown that employing approximate multipliers optimized with respect to a given CNN can reduce (in comparison with a common truncation) the overall power consumption with a negligible impact on the accuracy. When automated approximate circuit design techniques are applied in this context, the key question is how to define the error metric for the approximation procedure working at the level of components (multipliers). It is evident that the error metrics cannot be based on the classification accuracy (i.e. at the CNN level) as obtaining this parameter requires to perform a very time consuming evaluation for each CNN containing a new candidate approximate multiplier. This approach enables to explore only a very limited number of candidate designs (within the available time) and obtain a low quality solution. On the other hand, if a common error metric is applied at the level of multipliers, the approximation algorithm has no way to exploit the particular data distribution observed in a given CNN.

In general, we are looking for an easy-to-calculate error metric applicable at the level of components, but providing highly correlated outputs with the quality measure used in the application containing these components. This application-tailored, but component-level error metric is then used in the circuit approximation method.

This paper deals with an automated design of ASACs using Cartesian Genetic Programming (CGP). In the context of CGP-based approximations, we propose a new error metric – a *weighted mean error distance* (WMED) – for steering the circuit approximation process. WMED introduces a set of weights (derived from the data distribution measured on a selected signal in a given application) determining the importance of each input vector for the approximation process. The principle is to allow more aggressive approximations for less important inputs (lower weights are assigned to them) and gentle approximations for highly important inputs (higher weights are assigned to them).

The proposed method is evaluated using (1) synthetic benchmark problems and (2) two instances of neural image classifiers. In the case of synthetic benchmark problems, the objective is to design an approximate multiplier $\widetilde{M}(x,y)$ showing high-quality trade-offs between WMED and power consumption. The weights used in WMED reflect the importance of particular input vectors on x input which is modeled using a probability mass function D. In other words, $\widetilde{M}$ is designed, optimized and approximated for a user-given D. This is highly relevant for applications in which one operand of the multiplier is an arbitrary input value and the importance of the second operand (roughly) follows D. For example, in image filters, signal filters, or artificial neurons there is always an input multiplied by a certain value (i.e. a filter coefficient or a synaptic weight) which can be statistically characterized for a given application. At the same time it is required that all multipliers have to be identical in these applications in order to obtain uniform circuit structures suitable for hardware implementation.

In the case of neural image classifiers, application-specific approximate MAC (multiply-and-accumulate) units are designed to provide the best trade-offs between the classification accuracy and power consumption. The definition of WMED is based on the distribution of weights across all NN layers.

II. Related work

This paper deals with *functional approximation* which is a technology-independent circuit approximation method. Its purpose is to modify the implementation (function) of a given circuit in such a way that the quality of service is kept at desired level while power consumption is reduced (or performance is increased) with respect to the original implementation.

A. Functional approximation

Approximations have been introduced to circuits described at the transistor, gate [5], [6], register-transfer and behavioral [4] levels. Many authors have introduced approximate operations directly at the level of abstract circuit representations such as binary decision diagrams and and-invert graphs [8]. Basic functional approximation principles are: (i) truncation, which is based on reducing bit widths of registers and all operations of the data path; (ii) pruning, which lies in removing some parts of the circuit; (iii) component replacement, in which exact components are replaced with approximate components available in a library of approximate components; (iv) re-synthesis, in which the original logic function is replaced by a cheaper implementation; (v) other techniques such as table lookup etc.

The automated approximation methods are often constructed as iterative methods in which many candidate approximate circuits have to be generated and evaluated. This is, in fact, a multi-objective search process. Examples of elementary circuit modifications (i.e. steps in the search space) are replacing a gate by another one, reconnecting an internal signal or reconnecting a circuit output. It has been shown that this kind of search can effectively be performed by means of Cartesian genetic programming [9], [3], [6]. Details on CGP will be given in Section III.

B. Approximate CNNs

With the rapid development of artificial intelligence methods based on deep CNNs, a lot of attention has been focused on efficient hardware implementations of neural networks [10]. CNNs employ multiple layers of computational elements performing the convolution operation, pooling (selection/subsampling), non-linear transformations and the final classification based on a common multi-layer perceptron (MLP).

One of the key challenges in this area is to provide fast and energy efficient *inference phase* (i.e. the application of an already trained network). The reason is that trained CNNs are employed in embedded systems and have to process enormous volumes of data in a real-time scenario. As CNNs are highly error resilient, a good strategy is to reduce the bit width for all involved operations and storage elements. This approach has been taken by the Tensor Processing Unit (TPU), where only 8-bit operations are implemented in MAC units. The highly parallel processing enabled by TPU exploits a systolic array composed of 65,536 8-bit MAC units [11].

Approximation techniques developed for circuit implementations of NNs were surveyed in [12]. In the case of approximate multipliers for NNs, they are implemented either as multiplier-less multipliers [7], truncated multipliers [11] or application-specific multipliers [6]. For example, Mrazek et al. developed approximate multipliers that perform exact multiplication by zero (which is important as many weights are zero and no error is thus distributed to subsequent processing layers) and deep approximations are allowed for all the remaining operand values [6]. On two benchmark problems, this strategy provided better trade-offs (energy vs. accuracy) than the multiplier-less multipliers [7], [6].

III. Design of application-specific approximate circuits

The proposed design method based on CGP is developed for combinational circuits. For the sake of simplicity, we will focus on approximate multipliers in this section.

A. Weighted mean error distance

We propose WMED as an extension of the conventional *mean error distance* (MED). Let I and J be discrete random variables representing data at the inputs of a multiplier M. Let D be a probability mass function of I defined as $D(x) = Pr(I = x)$. Given D and a signed approximate w-bit multiplier $\widetilde{M}$, WMED is defined as

$$\mathrm{WMED_D}(\widetilde{\mathrm{M}}) = \frac{1}{2^{2w}} \sum_{i=-2^{w-1}}^{2^{w-1}-1} \sum_{j=-2^{w-1}}^{2^{w-1}-1} \alpha_{i,j} \cdot |i \cdot j - \widetilde{M}(i,j)|$$

where $\widetilde{M}(i,j)$ is the output of a signed approximate multiplier for inputs i and j, and $0 \leq \alpha_{i,j} \leq 1$ is the weight determined by the probability mass function D. In our case $\alpha_{i,j} = D(i)$

($\sum D(i) = 1$), but a different approach can be chosen in general. The WMED for an unsigned approximate multiplier is constructed accordingly. Note that $0 \leq \mathrm{WMED_D} \leq 1$.

B. Circuit representation in CGP

In CGP [9], a combinational circuit is modeled as a two-dimensional grid of nodes (see the example in Fig. 1), where the type of nodes depends on the level of abstraction used in modeling (the gates are used in our case). The circuit utilizes n_i primary inputs and n_o primary outputs. A unique address is assigned to all primary inputs (0 – 4 in Fig. 1) and to the outputs of all nodes (5 – 16 in Fig. 1) to define an addressing system enabling circuit topologies to be specified. As no feedback connections are allowed in the basic version of CGP, only combinational circuits can be created. Each candidate circuit is represented using $S = r \times c \times (n_a + 1) + n_o$ integers, where c is the number of columns, r is the number of rows and n_a is the maximum arity of node functions. All supported node functions are defined in the function set Γ. In this representation, the $n_a + 1$ integers specify one programmable node in such a way that n_a integers specify source addresses for its inputs and one integer determines the function of the node. This circuit representation can be seen as a netlist in which redundant components are allowed.

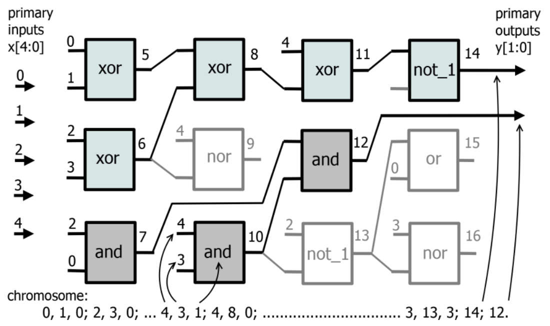

Fig. 1. Combinational circuit represented in CGP with parameters: n_i = 5, n_o = 2, c = 4, r = 3, n_a = 2, Γ = {xor (encoded with 0), and (1), or (2), nor (3), not_1 (4)}. Nodes 9, 13, 15 and 16 are inactive.

C. Search algorithm and fitness function

Having a candidate circuit represented as a string of integers, new candidate circuits are created by a random modification of this string – the so-called *mutation*. It is important to ensure that all randomly created numbers are within a legal interval, i.e. a valid candidate circuit is always produced.

CGP employs a simple search algorithm denoted $(1 + \lambda)$ which operates with a set of $1 + \lambda$ candidate circuits (the so-called population) [9]. Starting with the original circuit (the so-called parent), a new population is created by applying the mutation operator on the original circuit and creating λ offspring circuits. The mutation operator randomly modifies up to h randomly selected integers of the string. These offspring are evaluated in terms of functionality and electrical parameters and the so-called fitness score is assigned to them. The best performing individual is taken as a new parent. These steps are repeated until the time available for the evolution is exhausted.

The goal of the design process is to find an approximate circuit minimizing the area on a chip and keeping WMED below a predefined threshold. The area parameter is chosen because it is highly correlated with power consumption and can quickly be estimated using the technology library (see the methodology proposed in [6]). The design process is repeated for several target approximation errors E_i in order to construct Pareto front (the error vs. the area). The fitness value $\mathbf{F}$ of a candidate approximate multiplier $\widetilde{M}$ is defined as

$$\mathbf{F}(\widetilde{M}) = \begin{cases} A_{\widetilde{M}} & \text{if } \mathrm{WMED}_D(\widetilde{M}) \leq E_i \\ \infty & \text{otherwise,} \end{cases} \tag{1}$$

where $A_{\widetilde{M}}$ is estimated area of $\widetilde{M}$ and the objective is to minimize $\mathbf{F}$.

IV. Case Study 1: Data distribution driven approximate multipliers

The objective of this section is to show that better trade-offs (between key parameters of multipliers) can be obtained in comparison with the conventional approximation methods (which are assuming uniformly distributed input data) if a non-uniform data distribution is used in the WMED definition. Figure 2 shows the data distributions used in our experiments. D_1 and D_2 are arbitrarily chosen normal and half-normal distributions. The uniform distribution (D_u) will serve as a reference in all experiments.

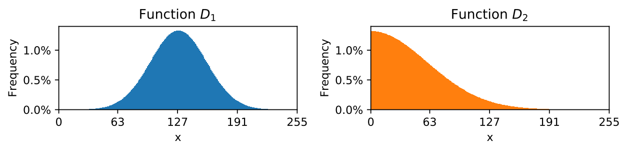

Fig. 2. Probability mass function D_1 and D_2

Approximate 8-bit multipliers are evolved using CGP which utilizes standard parameter setting as recommended in the literature [9], [6]: n_i = 16 (two 8-bit inputs), n_o = 16, c = 320 ... 490 depending on the initial multiplier, r = 1, n_a = 2, Γ = {all standard two-input gates}, h = 5 mutations/individual, λ = 4. The initial population of CGP is seeded with different conventional implementations of exact multipliers. The fitness function is defined according to Eq. 1. For all 14 target WMED values, we repeated the CGP-based design ten times (one CGP run took 1 hour). The best evolved circuits were re-synthesized with Synopsys Design Compiler (45 nm process; V_{dd} =1V) to obtain their power consumption and other parameters (Fig. 3). In order to investigate the impact of selected distributions D on properties of resulting multipliers, each multiplier is also evaluated using the remaining WMEDs that were not considered during the design. For both D_1 and D_2 we confirmed that CGP can evolve approximate multipliers showing better trade-offs than the approximate multipliers evolved for D_u and top-quality approximate multipliers available in [1].

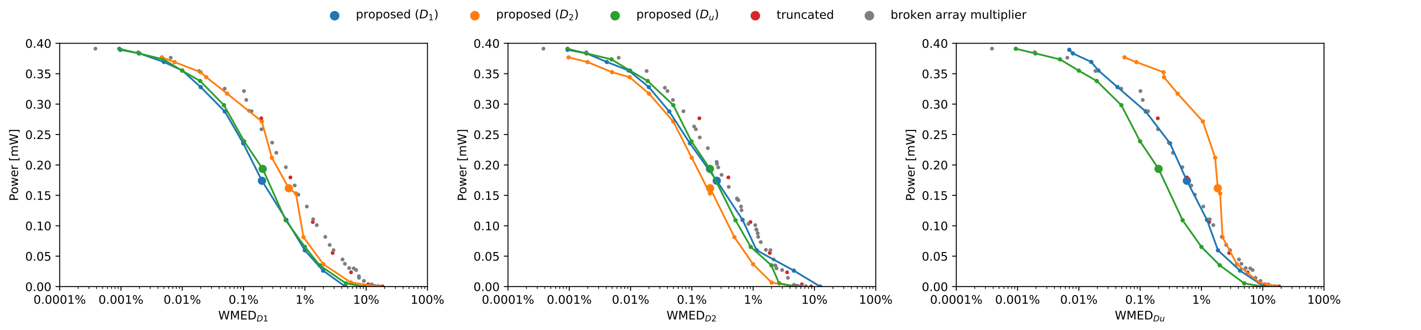

Fig. 3. Parameters of approximate multipliers that were evolved according to selected distributions (WMED_{D_1}, WMED_{D_2} and WMED_{D_u}) and conventional approximate multipliers (truncated array multiplier [1], broken-array multiplier [13]).

The heat maps on Fig. 4 show for selected multipliers (see the highlighted points in Fig. 3) how the resulting approximation error is reflecting the data distribution applied in the approximation process. In the case of D_1, if the operand is around 127 the product shows a low error, but higher errors are visible for operands near to 0 and 255. In the case of D_2, low errors are visible for $x < 127$. In the case of D_u, the error is spread more uniformly.

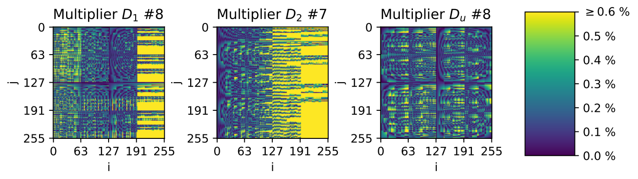

Fig. 4. Approximation errors for all combinations of input vectors of selected approximate multipliers. Note that these selected multipliers are very similar in terms of power consumption and WMED.

Intuitively, approximate multipliers optimized for error distribution D_2 should provide better trade-offs than other multipliers when used in the image filter which is constructed to eliminate Gaussian noise. The reason is that Gaussian filters employ a $k \times k$-pixel filtering window with many close-to-zero coefficients whose sum has to be less than 256. If results of approximate multiplication by these coefficients are almost exact (the error can be arbitrarily high for non-coefficients) then the quality of filtering is higher than if the filter contains approximate multipliers showing uniformly distributed errors. Hence, we compared the impact of various approximate multipliers on the quality of filtering conducted with the approximate Gaussian filter. We used a standard Gaussian filter implementation in which 3×3 pixels are multiplied by nine constants. Figure 5 clearly shows that Gaussian filters employing approximate multipliers (which were evolved according to distribution D_2) show better trade-offs between Peak Signal to Noise Ratio (PSNR) and power consumption (given for the complete image filter implementation) than other implementations. PSNR is calculated as the mean value from 25 images. Please note that we have not designed any specialized approximate multipliers for this task; we just applied the approximate multipliers presented in Fig. 3.

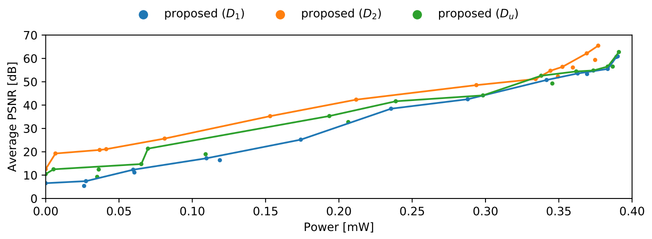

Fig. 5. Average PSNR obtained using approximate Gaussian image filters employing various implementations of approximate multipliers.

V. Case Study 2: Approximate MAC units for CNNs

When applying automated approximate circuit design techniques in the context of neural network based image classifiers, the key question is how to define an easy-to-calculate error metric for the approximation procedure working at the level of components (such as MACs and multipliers) because obtaining the classification accuracy of the whole NN is very time consuming. We will apply the CGP-based circuit approximation utilizing WMED to evolve approximate multipliers tailored for a particular trained NN.

A. Image classification benchmarks

Our method will be evaluated in the task of image classification (digits 0 – 9). Two NN architectures – a popular Multi-Layer Perceptron (MLP) applied on the MNIST benchmark and CNN LeNet-5 [14] applied on the more challenging Google's SVHN benchmark – will be addressed. This setup will allow us to compare our results with [6]. We used the MLP network with 28×28 input neurons, 300 neurons in the hidden layer and 10 output neurons whose outputs are interpreted as the probability of each of 10 target classes. We modified LeNet-5 to be able to process 32×32 pixel images stored in SVHN. The LeNet-5 consists of five layers – three convolution layers, two pooling layers used for data subsampling and one fully connected layer. The latter layer consists of 120 neurons outputting 10 values that are interpreted as the probability of each of 10 target classes. In LeNet-5, more than 278 thousand multiplication operations have to be executed to classify a single input image. A common MLP implementation shows 98 % accuracy on the MNIST data set. In the case of LeNet-5, 90.8 – 92.7 % accuracy is typically reported on SVHN [6].

B. Reference implementation

Common implementations of neural networks typically use a 32-bit floating-point representation of real numbers for data storage and manipulation. For both considered neural networks, we firstly apply a quantization process with Ristretto tool, which performs a fully automated trimming analysis of a given network [15]. The analysis using different bit-widths revealed that 8-bit fixed point signed values provide sufficient classification accuracy (only a 0.01 % resp. 0.1 % accuracy drop for MNIST, resp. SVHN reported). At the end of this process, we obtained models that can be accelerated in HW using a systolic array of processing elements. Each processing element consists of an 8-bit MAC unit and n-bit register (such as in [11]). Each MAC includes an 8-bit signed multiplier and n-bit adder, where $n = 8 + \log_2 d$ and d is the maximum number of products that have to be summed up. In the case of fully connected layers and MLP, d equals to the maximum number of weights that can be connected to a neuron. In the case of convolution layers, d is the number of items in a kernel.

C. Applying available approximate circuits

We replaced the exact multipliers with top-quality approximate 8-bit multipliers that have been proposed in literature. In particular, we considered broken-array multipliers [13] and EvoApprox8b library [3]. We also utilized the approximate multipliers in which the exact multiplication by zero is guaranteed [6]. Then we evaluated the accuracy of the neural network containing these multipliers on test data sets. Results are presented in Fig. 7.

D. Evolutionary design of approximate multipliers

We employed CGP to evolve application-tailored 8-bit approximate multipliers with the WMED error metric reflecting the properties of our target neural networks and data sets. In order to establish WMED, we analyzed the distribution of weights across all convolutional CNN layers / MLP neurons in fully trained NNs. The resulting distributions are shown in Fig. 6 (Top). In the case of SVHN, the distribution of weights is close to the normal distribution with zero mean, but MNIST has 92 % the most frequent values within the interval (-0.08 ... 0.08).

CGP was used with the following parameters: n_i = 16 (two 8-bit inputs), n_o = 16, c = 320 ... 490 depending on the initial multiplier, r = 1, n_a = 2, Γ = {all standard two-input gates}, $h = 5$ mutations/individual, λ = 4, 10^6 iterations/run. The fitness function is defined as proposed in Section III.

The best discovered multipliers were integrated into MAC units and relevant design parameters were obtained with Synopsys Design Compiler (45 nm process). Fig. 6 (Bottom) shows Power Delay Product (PDP) by means of box plot graphs for resulting approximate multipliers evolved for desired WMED. Each box plot was constructed from 25 independent CGP runs. For example, if WMED is constrained to 0.2%, PDP can be reduced by 50 % in the case of LeNet-5 on SVHN.

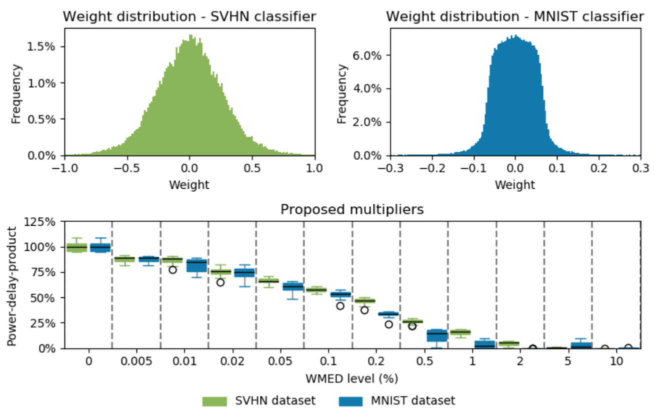

Fig. 6. Top: Weight distribution in neural networks trained with SVHN (left) and MNIST (right). Bottom: Relative power-delay-products of multipliers obtained from 25 independent CGP runs for a given WMED.

E. Integration of approximate MACs to CNNs

The best non-dominated MACs were integrated to both neural network architectures whose classification accuracy was then calculated using test sets (Table 1). We can observe that CNN accuracy remains practically unchanged for WMED $\leq$ 0.5%. However, corresponding PDP of MAC units was reduced by 55 %. If a deeper approximation is allowed (WMED = 2%), a 70 % reduction of PDP is reported.

The fine-tuning of the NN weights can, in principle, improve the accuracy drop introduced by quantization. During this fine-tuning, the network learns how to classify images with approximate multipliers. Table I shows that the effect of fine-tuning (10 iterations employed) is enormous especially in the case of 5 % and 10 % error. For 10 % error, for example, the accuracy was improved from -62.99 % to -5.04 % for SVHN and from -61.14 % to -1.24 % for MNIST. As it is acceptable to tolerate a 1 % accuracy drop in practice, we can achieve more than 70 % power and PDP reduction for SVHN (WMED = 2%) and 85 % reduction for MNIST (WMED = 5%). Fig. 7 compares the classification accuracy (obtained by LeNet-5 on SVHN and MLP on MNIST) and relative power consumption when different approximate multipliers are employed in MAC units. Solutions obtained with the proposed method are clearly dominating.

VI. Conclusions

By means of the proposed error metric – WMED – we demonstrated how an application-level error metric can be translated to a component level and exploited in searching for high quality application-specific approximate circuits. The method has been evaluated in the design of approximate multipliers in which the importance of one of the operands is determined using a probability mass function. Under this scenario we evolved approximate multipliers showing better trade-offs than (i) approximate multipliers evolved with a common error metric and (ii) high-quality conventionally designed approximate multipliers. The impact of the method was

TABLE I

RELATION BETWEEN WMED OF BEST APPROXIMATE MULTIPLIERS AND CLASSIFICATION ACCURACY OF APPROXIMATE NEURAL NETWORKS BEFORE AND AFTER FINE-TUNING. [1]The accuracy as well as other parameters are expressed relatively to the original NN (negative value = degradation, positive value = improvement, 0 % = equal to the parameters of NN when exact multipliers employed). [2]The design parameters are reported for the MAC units.

MAC WMED level (%)	**SVHN data set**					**MNIST data set**				
	Initial accuracy	**After finetuning**	**PDP**	**Power**	**Area**	**Initial accuracy**	**After finetuning**	**PDP**	**Power**	**Area**
0	0.00 %	0.24 %	0 %	0 %	0 %	0.00 %	0.09 %	0 %	0 %	0 %
0.005	0.02 %	0.36 %	-4 %	-8 %	-3 %	0.00 %	0.09 %	-1 %	-12 %	-3 %
0.01	0.00 %	0.44 %	-4 %	-14 %	-5 %	0.00 %	0.10 %	-14 %	-16 %	-6 %
0.05	0.00 %	0.51 %	-26 %	-26 %	-16 %	0.03 %	0.14 %	-28 %	-27 %	-11 %
0.1	0.07 %	0.41 %	-29 %	-37 %	-27 %	0.05 %	0.10 %	-35 %	-32 %	-13 %
0.5	0.08 %	0.31 %	-55 %	-57 %	-38 %	-0.01 %	0.10 %	-60 %	-65 %	-45 %
1	0.13 %	0.20 %	-60 %	-65 %	-45 %	-0.42 %	0.12 %	-70 %	-71 %	-49 %
2	-0.82 %	-0.41 %	-70 %	-71 %	-49 %	-4.79 %	-0.02 %	-79 %	-75 %	-53 %
5	-18.56 %	-1.85 %	-90 %	-86 %	-70 %	-3.70 %	-0.30 %	-85 %	-83 %	-66 %
10	-62.99 %	-5.04 %	-89 %	-87 %	-66 %	-61.14 %	-1.24 %	-91 %	-89 %	-70 %

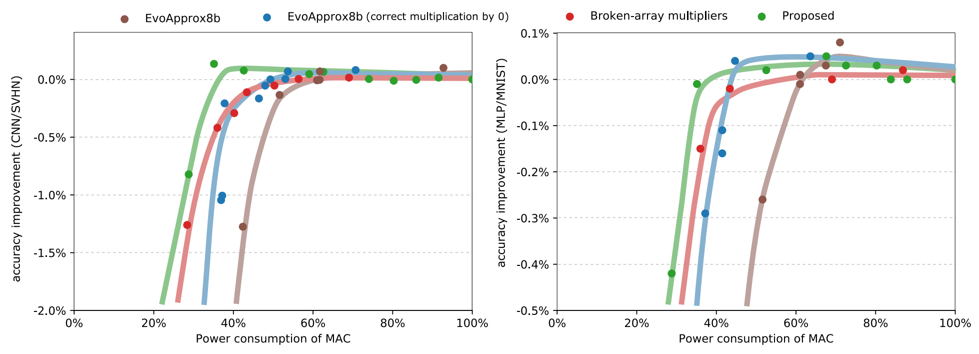

Fig. 7. Classification accuracy of CNN on SVHN (left) and MLP on MNIST (right) and relative power consumption when different approximate multipliers (EvoApprox8b [3], [6], broken-array multipliers [13]) are employed in MAC units. The NN accuracy is expressed relatively to the quantized model employing 8-bit accurate multiplication.

demonstrated in the approximate implementation of Gaussian image filters. We also showed that when evolved MAC units are used in NN-based classifiers, 65 % power consumption reduction is obtained (in the MAC units), with a negligible impact on the accuracy of classification.

This work was supported by Czech Science Foundation project 19-10137S.

REFERENCES

[1] H. Jiang, C. Liu *et al.*, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, Aug. 2017.

[2] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, pp. 86:1–86:6.

[3] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017*, 2017, pp. 258–261.

[4] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE'14. EDA Consortium, 2014, pp. 1–6.

[5] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: systematic logic synthesis of approximate circuits," in *The 49th Design Automation Conference*. ACM, 2012, pp. 796–801.

[6] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ACM, 2016, pp. 811–817.

[7] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," in *Proc. of the Design, Automation & Test in Europe Conference*. EDA Consortium, 2016, pp. 1–6.

[8] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of aigs for error tolerant applications," in *Proc. of ICCAD'16*. ACM, 2016, pp. 83:1–83:8.

[9] J. F. Miller, *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[10] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[11] N. P. Jouppi, C. Young, N. Patil *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. of the 44th Annual Int. Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.

[12] P. Panda, A. Sengupta, S. S. Sarwar, G. Srinivasan, S. Venkataramani, A. Raghunathan, and K. Roy, "Invited – cross-layer approximations for neuromorphic computing: From devices to circuits and systems," in *53nd Design Automation Conference*. IEEE, 2016, pp. 1–6.

[13] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, April 2010.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5784–5789, 2018.