Libraries of Approximate Circuits: Automated Design and Application in CNN Accelerators

Vojtech Mrazek, *Member, IEEE*, Lukas Sekanina, *Senior Member, IEEE*, and Zdenek Vasicek, *Member, IEEE*

***Abstract*— Libraries of approximate circuits are composed of fully characterized digital circuits that can be used as building blocks of energy-efficient implementations of hardware accelerators. They can be employed not only to speed up the accelerator development but also to analyze how an accelerator responds to introducing various approximate operations. In this paper, we present a methodology that automatically builds comprehensive libraries of approximate circuits with desired properties. Target approximate circuits are generated using Cartesian genetic programming. In addition to extending the EvoApprox8b library that contains common approximate arithmetic circuits, we show how to generate more specific approximate circuits; in particular, MxN-bit approximate multipliers that exhibit promising results when deployed in convolutional neural networks. By means of the evolved approximate multipliers, we perform a detailed error resilience analysis of five different ResNet networks. We identify the convolutional layers that are good candidates for adopting the approximate multipliers and suggest particular approximate multipliers whose application can lead to the best trade-offs between the classification accuracy and energy requirements. Experiments are reported for CIFAR-10 and CIFAR-100 data sets.**

***Index Terms*— Approximate circuit, genetic programming, convolutional neural network, hardware accelerator, optimization.**

I. Introduction

THE proper application of *approximate circuits* enables designers to obtain excellent trade-offs among power consumption, performance, and quality of service in many error resilient applications such as image recognition, video processing, data mining, and deep learning [1]. However, the design of approximate implementations of digital circuits is a time-demanding challenge even for experienced circuit designers. Inspired in the common circuit design flow, which is based on reusing pre-designed parts stored in a component library, *libraries of approximate circuits* have been introduced to accelerate the design process [2]–[4]. Approximate circuits intended for such libraries are typically obtained by the so-called *functional approximation*, which is a technology-independent simplification or modification of the exact version of the circuit. Functional approximation methods are being devised for either one type of circuits (e.g., multipliers) or arbitrary circuits. In the latter case, these methods algorithmically simplify, prune, or resynthesize the original (exact) implementation of the circuit. The designer can specify various objectives and constraints for target circuits. The resulting library of approximate circuits then contains many implementations showing different trade-offs among power consumption, performance, quality of service, and other criteria. The key benefit is that the user can quickly choose the most suitable approximate implementation for a given application, without repeating the design process.

An open problem remains how to systematically and effectively build comprehensive libraries of approximate circuits that are useful in real-world applications. In particular, we will focus on the EvoApprox8b library which is a collection of 8-bit approximate adders and multipliers that was automatically generated using Cartesian genetic programming (CGP) in 2017 [2]. EvoApprox8b contains hundreds of fully synthesized and characterized approximate circuits showing high-quality trade-offs, as documented in several independent case studies [5]–[7].

The aim of this article is twofold. First, we present a methodology that enables us to extend the original version of the library in several directions and in such a way that new types of arithmetic circuits utilizing various bit widths (from 8 to 32 bits, in some cases to 128 bits) can be included. Considering the fact that these circuits can be optimized for many different error metrics (and their combinations), for various target fabrication technologies and that hundreds of unique trade-offs can be generated for every specification, the resulting library of arithmetic circuits would contain tens of thousands of circuits. One of the challenges is determining the (exact) error for more complex approximate circuits. This problem is addressed by means of formal error analysis methods based on satisfiability (SAT) problem solving or binary decision diagram (BDD) construction and analysis [8]. In addition to standard arithmetic circuits having two n-bit operands, the methodology is capable of generating non-standard circuits with a different number of bits for each operand.

Second, the aim is to show how libraries of approximate circuits can be utilized in cutting-edge applications. We chose the most challenging one from those usually presented by the approximate computing community—hardware acceleration of

Manuscript received June 29, 2020; revised September 3, 2020 and October 7, 2020; accepted October 7, 2020. Date of publication October 20, 2020; date of current version December 11, 2020. This work was supported by the Czech Science Foundation under Grant 19-10137S. This article was recommended by Guest Editor T. Serrano-Gotarredona. *(Corresponding author: Zdenek Vasicek.)*

The authors are with the IT4Innovations Centre of Excellence, Faculty of Information Technology, Brno University of Technology, 60190 Brno, Czechia (e-mail: mrazek@fit.vutbr.cz; sekanina@fit.vutbr.cz; vasicek@fit.vutbr.cz).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2020.3032495

complex *convolutional neural networks* (CNNs). In the case of CNNs, approximate implementations have been proposed at the level of CNN architecture, data representation, arithmetic operations, memory access, and memory cells [9]–[11]. In our previous work, we focused on selecting the most suitable 8-bit approximate multiplier for a particular ResNet CNN [12]. In order to introduce suitable approximations to CNNs, a resilience analysis is conducted before any implementation steps. The resilience analysis of CNNs is usually carried out by removing some neurons, weights, memory accesses, or inserting some noise to neurons [9], [13] and observing the impact on the quality of service. In our case study, we show how a large set of approximate multipliers can be used to perform the resilience analysis of a hardware accelerator of five ResNet networks and to select the most suitable approximate multiplier for a given CNN layer. Contrasted to our previous work [12], [14], we also consider the non-standard approximate multipliers (the first input has 8 bits and the second has 7, 6, 5 or 4 bits) that we evolved for this purpose. These non-standard multipliers can contribute not only in reducing the overall energy of the multiplication operations conducted in CNNs, but also in reducing the memory usage as fewer bits are needed to store the weights. To summarize our key contributions:

- We present a methodology capable of extending the original EvoApprox8b library to contain more unique circuits on different bit widths. We also report the basic parameters of newly generated approximate circuits and compare them with EvoApprox8b and other approximate circuits.
- We evolve and characterize non-standard approximate multipliers (8 bit vs 7 – 4 bit operands) to extend the library and utilize them in a more detailed study on error resilience of CNNs.
- We perform a detailed error resilience analysis of five CNNs (ResNet-8, ResNet-14, ResNet-20, ResNet-26, and ResNet-164 v2) using relevant multipliers taken from the extended library of approximate circuits. Results are reported for the CIFAR-10 and CIFAR-100 benchmark problems [15].

The rest of the paper is organized as follows. Section II surveys related research, particularly the principles of approximate circuit design, error metrics and error analysis, existing libraries of approximate circuits, and their use in CNN design. Section III is devoted to CGP and its use for the design of approximate circuits. In Section IV, we present the methodology enabling us to systematically extend the library of approximate circuits and the results obtained using the methodology. Section V is devoted to the resilience analysis of ResNet CNNs using evolved approximate multipliers. Conclusions are given in Section VI.

II. Related Work

Approximate computing exploits the gap between the level of accuracy required by the applications/users and that provided by the computing system, for achieving diverse optimizations [1]. It has been developed in different ways and at various levels of the computing stack. Our state of the art survey is focused on approximate implementations of combinational circuits, their design and use in hardware accelerators of CNNs.

The approximations can be introduced to a circuit during various steps of the common circuit design flow. In this work, we primarily focus on the technology-independent logic synthesis step in which *functional approximations* can be applied. The advantage is that the approximate circuit can be implemented in an arbitrary application-specific integrated circuit (ASIC) or a field programmable gate array (FPGA) because it is assumed that the technology-dependent implementation is performed by means of common open source or commercial tools after the approximation is finished.

The methods performing the functional approximations can be classified as single-purpose or general-purpose (automated). The single-purpose (ad-hoc) methods have been developed for specific circuit components such as adders, multipliers, and dividers [16], [17]. On the other hand, the automated methods use some general-purpose circuit simplification, resynthesis, or approximation techniques and enable us to approximate arbitrary circuits. These methods start with an original (exact) circuit and, typically iteratively, modify its structure to reach the desired trade-off between the error, power consumption, and other objectives.

A. Error Analysis

Prior to any approximation is introduced, it is necessary to define (i) the error metric(s) for guiding the approximation process, (ii) the constraint(s) allowing to identify infeasible solutions and (iii) the error analysis method capable of quickly and reliably determining the error for all candidate approximate circuits.

1) Error Metrics and Constraints: The quality of approximate combinational circuits is typically expressed using one or several error metrics, where the most commonly used ones are: the error rate (ER) and the arithmetic errors such as the mean absolute error (MAE), the mean square error (MSE), the mean relative error (MRE), the worst-case error (WCE), and the worst-case relative error (WCRE). The equation for determining WCE is shown in Eq. 1, in which the output of the approximate circuit and original (exact) circuit is O_{approx} and O_{orig}, n_i is the number of primary inputs and $\forall x$ enumerates all possible input vectors. The remaining definitions can be found in [8].

$$\mathrm{WCE} = \max_{\forall x \in \mathbb{B}^{n_i}} \left| O_{\text{approx}}(x) - O_{\text{orig}}(x) \right| \tag{1}$$

An error metric can also serve as a constraint. For example, if the objective is to minimize MAE and WCE must be kept below a predefined threshold value, then WCE effectively constrains and thus reduces the design space. Apart from these standard metrics, we can employ additional application-specific metrics. As shown in [18], for example, the accurate multiplying by zero is an essential condition for the successful integration of approximate multipliers into neural networks. We can determine validity of this constraint by calculating WCE only for those input combinations

$x \in \mathbb{B}^{n_i}$ that lead to the zero output (see Eq. 2). The accurate multiplying by zero is guaranteed if $\mathrm{WCE}_{\mathrm{zr}} = 0$.

$$\mathrm{WCE}_{\mathrm{zr}} = \max_{\forall x \in \mathbb{B}^{n_i} : O_{\mathrm{orig}}(x)=0} O_{\mathrm{approx}}(x) \tag{2}$$

In addition to an error metric used at the circuit level, the error is also evaluated at the application level, e.g., for the complete CNN image classifier. This application-level error is then the primary quality indicator of the entire approximate implementation. As we aim to develop a general-purpose library of approximate circuits, we will consider all the listed circuit-level error metrics.

2) Error Analysis Methods: Most error analysis methods only estimate an approximate implementation error as determining the *exact error* is very time-consuming. The error estimate is obtained by circuit simulation across a reasonably inclusive subset of input vectors. For example, 10 million out of $2^{32} \approx 4295$ million vectors were employed for 16-bit multipliers [16]. The exact error can be obtained by exhaustive circuit simulation, but this approach is not scalable. For a few particular implementations of approximate adders and multipliers, a detailed probability analysis was performed, and probability error models were derived, e.g., [19]. Knowledge of these error models becomes very useful if such a circuit is (re)used in a more complex application, and one needs to perform reasoning about the application-level error based on the probability models available at the component level. An obvious disadvantage is that a lot of human effort is required to construct reliable probabilistic models for particular circuits.

The general-purpose exact error analysis methods are based on *equivalence checking*, i.e., checking whether a mathematical model of a circuit under design meets a given specification [8], [20], [21]. Two main approaches have been developed in this direction—-techniques based on Reduced Ordered Binary Decision Diagrams (ROBDD) and satisfiability (SAT) solvers. In approximate computing, this concept is extended to *relaxed equivalence checking*, by stressing that the considered circuits are checked to be equal up to some bound w.r.t. a suitably chosen distance (error) metric such as WCE or MAE. As ROBDDs are inefficient in representing classes of circuits for which the number of nodes in ROBDD is growing exponentially with the number of input variables (such as multipliers and dividers), their use in relaxed equivalence checking is typically possible for adders and other less structurally complex functions. For example, even 128-bit adders can be quickly analyzed in terms of all relevant error metrics [8]. Common SAT solvers are, in principle, applicable to the worst-case analysis only. However, this approach is more scalable than ROBDDs for the error analysis of multipliers [8].

In order to apply the formal techniques, we need to transform the problem into a Boolean satisfiability problem. This is achieved by constructing the so-called *approximation miter*, a circuit computing the difference between the accurate and approximate circuit. The approximation miter typically consists of three components: the exact and approximate circuits whose outputs are fed into an error computation block whose structure depends on the chosen metric. For WCE or MAE, e.g., the error computation block consists of the subtracter followed by a circuit that computes the absolute value. As soon as the miter is constructed, it can either be converted to a conjunction normal form and submitted to a SAT solver or represented as an ROBDD. The selection of a proper formal apparatus depends, in general, on the chosen error metric. The SAT solvers typically provide a binary output and can be thus applied to prove whether some threshold is exceeded. This is suitable, e.g., for calculating WCE or determining whether WCE is bounded by some threshold. On the other hand, the ROBDDs can be used not only to prove the satisfiability but also to quickly calculate the number of input assignments causing the erroneous output. This is necessary for calculating MAE and other statistically oriented metrics.

B. Automated Design of Approximate Circuits

Automated functional approximation methods start with a common (exact) circuit implementation and define one or several design objectives and constraints. The initial circuit is modified by an iterative approximation algorithm to produce an approximate implementation satisfying all design objectives and constraints. The basic algorithmic approximation techniques are pruning (i.e., removing some parts of the circuit), component replacement (i.e., complex subcircuits are replaced with simpler subcircuits) and approximate re-synthesis. However, if the circuit is provided in a behavioral HDL representation, other more software-oriented techniques (such as loop perforation and memorization) can be applied. The automated approximation methods select either randomly or heuristically, which parts of the circuit have to be removed, re-connected, or replaced. Examples of such methods are SALSA [20], SASIMI [22], ABACUS [23], ALFANS [24] and CGP-based methods [25]–[27]. Section III is devoted to introducing CGP for this purpose.

C. Libraries of Approximate Circuits

Benchmark suites and libraries have frequently been used to compare approximation methods and their results, i.e. approximate circuits and applications. Regarding the benchmarking of approximation methods, AxBench,[1] which contains benchmark problems for processors and GPUs as well as benchmark circuits, is one of the most popular collections. However, only circuit description and simulation scripts are given for eight circuits, i.e., the website contains neither the quality parameters nor the electrical parameters of the circuits.

In order to provide approximate circuits that can be routinely re-used, several libraries of approximate circuits have been developed. They are primarily focused on approximate arithmetic circuits because these circuits are typical building blocks of complex digital systems. Furthermore, it makes sense to provide their various approximate implementations that differ in the bit width, error, power consumption, delay and other parameters. A common approach to their design is developing a parameterizable approximate circuit and providing its approximate implementations by various settings

[1]axbench.org

of the parameter(s). In the case of approximate multipliers, suitable circuits of this type are, e.g., broken array multipliers (BAM) [17], rounding-based approximate multipliers (RoBA) [28], truncation- and rounding-based scalable approximate multipliers (TOSAM) [29] and dynamic range unbiased multipliers (DRUM) [30]. For approximate adders, see, e.g. [31]. The following libraries are open and can be obtained from the internet.

lpACLib[2] library contains the VHDL description of accurate and approximate versions of several manually constructed arithmetic modules (like adders and multiplier of different bit-widths) and the corresponding software implementations developed in C and MATLAB. All circuits are configurable and different configurations represent different approximate circuits [3].

GeAr[3] is a low-latency generic accuracy configurable adder that provides a higher number of potential configurations compared to state-of-the-art approximate adders. Circuit implementations are available in MATLAB and Verilog [32].

SMApproxLib[4] is an open source library of approximate multipliers with different bit-widths, output accuracies and performance gains targeting FPGAs [4].

EvoApprox8b[5] library was completely generated by CGP. Its first version from 2017 contains 430 non-dominated 8-bit approximate adders evolved from 13 conventional adders, and 471 non-dominated 8-bit approximate multipliers evolved from 6 conventional multipliers. All circuits are fully characterized in terms of area, delay, power consumption, and seven error metrics and they are available for download in C, Verilog, and Matlab.

D. Approximate Implementations of CNNs

CNNs are deep neural networks containing, in addition to other layers, the so-called convolutional layers. CNNs show superior performance, especially in image and video processing tasks such as image classification. As the state of the art CNNs consists of hundreds of layers and millions of network elements, they are demanding in terms of the execution time and energy requirements. For example, the inference phase of a trained CNN such as ResNet-50 (see Section V for details) requires performing $3.9 \cdot 10^9$ multiply-and-accumulate (MAC) operations to classify one single input image [10]. Training of CNNs is significantly more time and resource demanding.

Arithmetic operations conducted during the inference of a CNN are responsible for 10%-40% of energy, depending on a given CNN architecture and the CNN accelerator employed [10]. Various fixed-point as well as floating-point number representations were evaluated for CNNs to reduce power consumption [10], [33]. A detailed analysis of precision-scalable MAC architectures for CNNs was performed in [34]. The Ristretto tool helped in determining the optimum number of bits for arithmetic operations [13]. Further savings in energy are obtained not only by bit width reduction of arithmetic operations but also by introducing approximate operations, particularly to the multiplication circuits, which is currently a fresh research topic [18], [35], [36].

Libraries of approximate circuits have systematically been exploited in the CNN design in the following directions. (i) Mrazek *et al.* introduced an optimization algorithm for choosing a suitable approximate multiplier from a library of approximate multipliers in such a way that one approximate multiplier serves several layers, and the overall classification error and energy consumption are minimized [12]. At the same time it is ensured that no re-training is needed after introducing the approximate multipliers into CNN. (ii) Using a library of approximate multipliers, Ansari at al. identified the features in an approximate multiplier that tend to make it superior to others with respect to CNN accuracy [7]. A predictor was then built to forecast how well a multiplier is likely to work in a given CNN. This predictor was verified by classifying 114 approximate multipliers based on their performance in LeNet-5 and AlexNet CNNs on the SVHN and ImageNet data sets, respectively. (iii) A subset of the EvoApprox8b library was used in the resilience analysis of several ResNet networks as introduced in our preliminary conference report on this topic [14].

III. Circuit Approximation Using CGP

Cartesian genetic programming (CGP) is a branch of genetic programming primarily developed for the design and optimization of digital circuits [37]. CGP can also be used as a general-purpose approximation method for combinational circuits [25], [27]. CGP represents candidate circuits as directed acyclic graphs. These graphs are iteratively modified using mutation operator(s) to meet the design objectives and ensure that various constraints that can be imposed on the error, area, delay, or any other circuit property are not violated.

A. Circuit Representation

In the most common version of CGP, a candidate circuit is represented as a string of integers (the so-called chromosome) which fully specifies a logic network containing up to N nodes. These nodes are organized in a two-dimensional grid of n_c columns and n_r rows ($N \leq n_c \cdot n_r$). The number of primary inputs and outputs of the circuit is denoted n_i and n_o. Each node implements one of the functions specified in the set of functions Γ and has up to n_a inputs and a single output. For gate-level circuits, Γ usually contains a set of binary logic functions ($n_a = 2$). Any node input can be connected to any node in the previous $1 \ldots L$ columns, where L is a user-defined parameter. As Fig. 1 shows, all primary inputs and node outputs are assigned with a specific index, which enables us to specify the circuit encoding. Three integers are reserved for defining every node (two input codes and one function code). Finally, n_o integers specify where the primary outputs are connected.

B. Search Algorithm

Every candidate circuit represents one design point in the design space. In CGP, new designs are created by introducing

[2]sourceforge.net/projects/lpaclib/
[3]sourceforge.net/projects/approxadderlib
[4]cfaed.tu-dresden.de/pd-downloads
[5]fit.vutbr.cz/research/groups/ehw/approxlib/

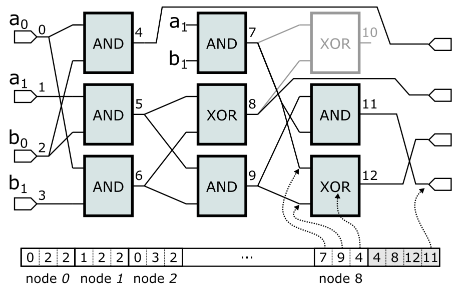

Fig. 1. A two-bit multiplier represented in CGP with parameters: $n_i = n_o = 4, n_c = n_r = 3, n_a = 2$, $\Gamma = \{0^{identity}, 1^{not}, 2^{and}, 3^{or}, 4^{xor}, 5^{nand}, 6^{nor}, 7^{xnor}, 8^{const0}, 9^{const1}\}$.

small random modifications to the chromosome. This operation is called the mutation and typically modifies h integers of the chromosome. Note that all changes must lead to valid circuits, i.e., only valid function codes and connections can be created.

The search method is based on the $(1 + \lambda)$ evolutionary strategy in which λ offspring circuits are created from one parent [37]. The search algorithm can start with either a randomly generated initial population or existing designs. The population size is $1 + \lambda$. After evaluating the initial population (i.e., measuring the circuit functionality and cost in the fitness function) the following steps are repeated until the termination condition is not satisfied: (i) the best-scored circuit (called the parent) is selected; (ii) λ offspring circuits are created from the parent using mutation; (iii) all offspring circuits are evaluated.

C. Evolutionary Circuit Approximation

The approximation process typically starts with an accurate circuit or several different versions of the accurate circuit. Candidate approximate circuits are generated from the original circuit in the course of a CGP run. As thousands of candidate circuits can be generated, the error evaluation must be fast. For small circuits, the exhaustive circuit simulation utilizing all possible input vectors is the quickest option. The scalability issues typical for more complex circuits are addressed by means of formal methods introduced in Section II-A. The electrical parameters (such as delay and area) of candidate circuits are estimated in the fitness function. Their precise evaluation is conducted by a professional design tool at the end of evolution, but only for the best-evolved designs. This strategy significantly reduces the design time [27].

If a single-objective CGP is applied, the target error range (e.g., the MAE), determined by e_{min} and e_{max}, is specified by the user. The goal is to minimize the number of gates (or area or power consumption, depending on the specification) while the error of the circuits is kept between the target values e_{min} and e_{max}. If various trade-offs between the objectives are requested, CGP is executed several times with different error bounds as the control parameters.

The multi-objective CGP allows us to optimize the error and other key circuit parameters (area, delay, and power consumption) together in one run. We are primarily interested in the approximate circuits belonging to the *Pareto front* that contains the so-called *non-dominated solutions*. For example, consider two circuits C1 and C2. Circuit C1 *dominates* circuit C2 if: (1) C1 is no worse than C2 in all objectives, and (2) C1 is strictly better than C2 in at least one objective. The search algorithm of CGP is then modified to build Pareto fronts in the course of evolution continuously. This approach was adopted to generate the EvoApprox8b library [2].

IV. Extending the Library of Approximate Circuits

To extend the EvoApprox8b library, we used a single objective CGP. The goal was to design approximate versions of various multipliers and adders that exhibit WCE no worse than a given threshold. Compared to EvoApprox8b, we considered not only unsigned but also signed adders and multipliers. Twenty thresholds linearly sampled in the log space were used for every circuit instance. The lowest threshold's value equals 1 while the highest value equals $2^{w+1} - 1$ for adders and $2^{2w} - 1$ for multipliers, where w is the bit-width. Ten independent runs of CGP were executed for every threshold. The WCE error metric was not chosen arbitrarily. We employed the fact that the other essential error metrics such as MAE and MSE (and also ER in case of adders) highly correlate with WCE [8]. In addition to that, the computation of WCE violation is of modest complexity compared to evaluating other error metrics, especially the statistical ones such as MAE or ER. This property is important, especially for instances with more than 12-bit operands where we cannot employ the exhaustive simulation in the evolutionary loop to determine WCE due to high runtime requirements. As shown in [26], deciding whether a candidate circuit violates the fixed WCE threshold can be done within few seconds even for tough verification problems such as 32-bit multipliers where the complete formal verification may represent a time-demanding process requiring several minutes or hours.

At the end of the evolutionary runs, we gathered all the logged results for a particular design problem, filtered out redundant solutions using estimated circuit parameters and available error parameters, and stored the netlists in a library. The complete process is illustrated in Fig. 2. The extended version of the library, called EvoApproxLib, is created as follows. The netlists are converted to corresponding HW and SW models. The HW models are synthesized by a common design tool (we used Synopsys Design Compiler, 45 nm process, V_{dd}=1V). The SW models allow determining all relevant quality parameters. After synthesis, we calculate the final Pareto front using real circuit parameters obtained by the design tool. This usually gives us a large set (see Tab. II) of non-dominated approximate circuits which nearly covers the complete design space (see the gray points in Fig. 4). The synthesis step can be repeated for different technology libraries (see the technology-specific custom library in Fig. 2). Moreover, we can also utilize a SW model provided by the user and calculate application-specific error parameters that are needed to create an application-specific library of approximate components. For example, we can evaluate the approximate multipliers in a CNN and use the classification accuracy as an application-specific quality parameter.

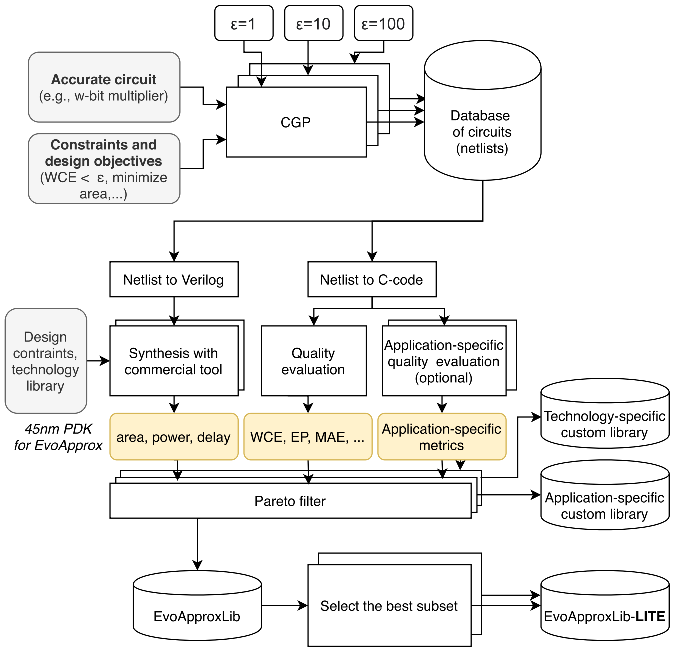
ε=1
ε=10
ε=100
Accurate circuit (e.g., w-bit multiplier)
Constraints and design objectives (WCE < ε, minimize area,...)
CGP
Database of circuits (netlists)
Netlist to Verilog
Netlist to C-code
Design contraints, technology library
Synthesis with commercial tool
Quality evaluation
Application-specific quality evaluation (optional)
45nm PDK for EvoApprox
area, power, delay
WCE, EP, MAE, ...
Application-specific metrics
Technology-specific custom library
Pareto filter
Application-specific custom library
EvoApproxLib
Select the best subset
EvoApproxLib-LITE

Fig. 2. The methodology of creating the library of approximate circuits.

As the number of instances in EvoApproxLib is high (e.g., 51,882 pieces of 8-bit approximate multipliers), the selection of the most suitable circuit for a given application could be a challenging combinatorial problem. In order to simplify this task and help the designers, we identified a subset of circuits and composed the so-called EvoApproxLib$^{\text{Lite}}$ library. The selection follows the principles of Pareto optimality with respect to several objectives in which power consumption is compared against ER, MAE, WCE, MSE and MRE. For each of the five subsets of components, ten circuit instances evenly distributed along the power axis were included to EvoApproxLib$^{\text{Lite}}$.

Section IV-A presents the CGP setup, evaluation of CGP performance, and selected approximate circuits that we obtained when extending the library of approximate circuits with very specific components—approximate multipliers having their operands on different bit widths. These multipliers are especially useful in CNNs, as it will be shown in Section V. Section IV-B deals with extending the EvoApproxLib to support more complex approximate adders and multipliers. Finally, all results are summarized in Section IV-C.

A. The NxM-Bit Approximate Multipliers

The objective is to design a set of approximate multipliers with operands on w_a and w_b bits, where $w_a + w_b = n_i = n_o$. Considering their application in CNNs, we chose $w_a = 8$ and $w_b = 4$, 5, 6, and 7.

1) Setup: For each configuration of bit-widths, we generated six different (but exact) multiplier architectures, including ripple-carry array multiplier (RCAM), two carry-save array multipliers (CSAM), and three Wallace tree architectures (WTM). In total, 24 netlists of exact multipliers were generated and used to seed the CGP. For each seed, 20 target error levels ε were considered. For every ε, 10 independent CGP runs were executed with the following parameters: $N = k$, where k is the number of gates of the original (exact) circuit with $n_i = w_a + w_b$ primary inputs and $n_o = n_i$ primary outputs, $\lambda = 4$, $h = 1$, Γ contains AND, OR, XOR gates, their inverted versions and inverter. At most, 10^6 generations were produced. All experiments were conducted on a server equipped with 2.4 GHz Intel Xeon CPU. The average duration of a single CGP run, considering this setup, ranges from 7.8 to 147.7 minutes depending on ε.

TABLE I
THE NUMBER OF GENERATED / NON-DOMINATED IMPLEMENTATIONS OF APPROXIMATE 8XN MULTIPLIERS IN EVOAPPROXLIB

	approximate mutliplier $w_a \times w_b$			
seed	**8x7**	**8x6**	**8x5**	**8x4**
RCAM	841 / 607	1,074 / 770	1,135 / 866	1,190 / 886
CSAM[1]	955 / 646	1,049 / 765	1,078 / 777	1,395 / 1,012
CSAM[2]	888 / 604	1,027 / 723	1,263 / 831	1,235 / 880
WTM[1]	1,038 / 709	1,136 / 778	1,281 / 906	1,250 / 931
WTM[2]	916 / 621	1,202 / 817	1,241 / 894	1,359 / 935
WTM[3]	1,051 / 699	1,099 / 810	1,231 / 913	1,241 / 941
total	5,689 / 2,192	6,587 / 2,749	7,229 / 3,402	7,670 / 3,676

As the approximate multipliers are supposed to be used in neural networks, we integrated the requirement for the accurate multiplying by zero (as introduced in [18]) together with the WCE constraint in the fitness function F as follows:

$$F(\widetilde{M}, \varepsilon) = \begin{cases} cost(\widetilde{M}) & \text{if } WCE(\widetilde{M}) \leq \varepsilon \ \wedge \\ & WCE_{zr}(\widetilde{M}) = 0 \\ \infty & \text{otherwise,} \end{cases} \tag{3}$$

where $cost(\widetilde{M})$ is the cost of a candidate solution $\widetilde{M}$. The cost is estimated as the sum of weighted areas of the gates used in the circuit. The objective is to minimize F. The validity of both conditions is checked using a single pass of exhaustive simulation.

At the end of evolution, the best-scored circuit is synthesized. Because the error parameters are evaluated using the exhaustive simulation, one can calculate all relevant error metrics in a reasonable time. Less than 2 ms are required, for example, to obtain the response of the largest 8×7 bit multiplier to all input combinations.

2) Results: The number of generated circuits for each configuration of w_a and w_b is given in Tab. I. This table also reports the number of non-dominated implementations that are included in EvoApproxLib. For the possible power reduction, please refer to Tab. IV. As evident from Tab. I, the usage of different seeds is beneficial because more unique circuits are obtained at the end; see the number of non-dominated solutions in the row denoted 'total' in comparison with the remaining ones. For example, for 8×7 multiplier, the resulting Pareto front contains 2,192 circuits, whereas 607 circuits are obtained from RCAM. On the other hand, the Pareto front size is much smaller than the sum of non-dominated solutions in the individual rows. That means that similar design points can be reached from different seeds.

Fig. 3 shows convergence curves of CGP for five error levels when seeded with 8×7-bit ripple-carry array multipliers.

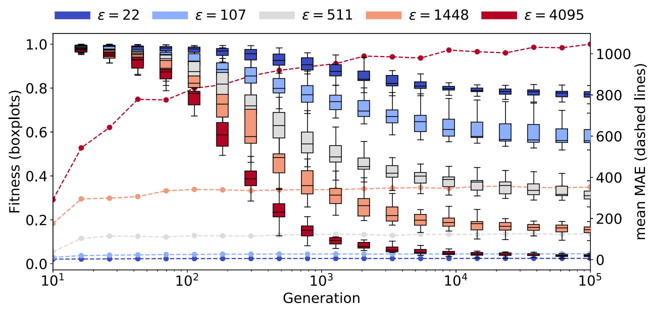

Fig. 3. The progress of the evolutionary design of 8 × 7 bit approximate multipliers. The box plots are constructed from 10 runs carried out for each of the five target error values ε.

Each of the five convergence curves (depicted using boxplot charts) shows the evolutionary search's progress aggregated from 10 independent runs. The convergence rate depends on the chosen ε. The fitness value decreases (i.e., the circuit shrinks) gradually for lower error levels and increases steeply for higher error levels. Only a small improvement in fitness is achieved in the last decade (i.e., in the range from 10^4 to 10^5), which means that the search probably reached the best-possible solution enabled by our experimental setup.

Fig. 3 also shows the development of MAE to illustrate the behavior of the error parameter whose value is not directly optimized or constrained. For the sake of better readability, the mean value of MAE is reported instead of boxplots. The MAE remains nearly constant for the lower values of ε. On the other hand, MAE gradually increases if $\varepsilon = 4095$. It seems that sub-optimal solutions are obtained at the end of the evolution when an excessive number of generations is used. However, a more detailed analysis revealed that MAE's upper bound depends on the value of WCE. Thus, it is guaranteed that the MAE stays at a reasonable level even if we use substantially more generations than necessary. Moreover, we store the best chromosome at every improvement of the fitness value. CGP thus provides a set of chromosomes in the course of evolution. The low-quality circuits are filtered out at the end of the evolution, and only the non-dominated solutions are kept.

B. Increasing the Bit-Width

Evolutionary approximation of arithmetic circuits operating on more than 12 bits requires applying a different approach to the error evaluation. We present a CGP-based approach that employs formal error analysis methods to provide approximate adders with up to 128 bits and approximate multipliers with up to 32 bits. More complex instances were not considered as they are less relevant for practice.

1) Approximation: For higher bit widths, we used the same CGP setup as described in Section IV-A. The only differences are that more generations (10^6) were used, and a SAT solver was employed to determine whether a candidate solution violates a target WCE.

In addition to the CGP-based approach, a scalable divide-and-conquer strategy was used for synthesizing 16-bit and 32-bit approximate multipliers [38]. The $2n$-bit operands are divided into four n-bit chunks (each operand has a lower and higher part) that are independently processed using four multipliers whose outputs are reduced using two adders with one n-bit and one $2n$-bit operand each. This method's key advantage is that if accurate adders are employed and some of the n-bit multipliers are arbitrarily chosen approximate multipliers with known WCE, the upper bound of WCE of the $2n$-bit approximate multiplier can be derived. If only one type of approximate multipliers is used then WCE can be calculated exactly. Moreover, this construction provides high-quality trade-offs between the area and error compared to many state-of-the-art approximate multipliers [38].

2) Error Parameters Calculation: The limit of exhaustive simulation is 32 inputs, i.e. two 16-bit operands. In the case of more complex approximate adders, all relevant error metrics are evaluated using ROBDD-based error analysis algorithms because the adders are structurally simple circuits, and all ROBDD-based error analysis algorithms scale well. Corresponding algorithms are summarized and assessed in [8]. For example, determining MAE for an approximate 16-bit and 32-bit adder takes 3 ms and 198 ms on average. In the case of more complex approximate multipliers, we used SAT solving to determine WCE using the algorithm proposed in [8]. Other metrics are estimated using simulation with a randomly generated subset of all possible input combinations.

C. The EvoApproxLib Library

Table II summarizes the number of approximate circuits that were generated and the number of non-dominated instances included in EvoApproxLib. Each of them represents a unique trade-off when some subset of objectives and constraints is considered. We can not remove any circuit without losing a multiplier or adder exhibiting some unique combination of properties. Please note that significantly different numbers of circuits of different types (e.g. 8-bit unsigned multipliers vs. 8-bit signed multipliers) are caused by our needs of particular circuits in different applications rather than non-existence of suitable circuits.

In Fig. 4, the black points (corresponding with the EvoApproxLib$^{\text{Lite}}$) are contrasted with the original circuits of EvoApprox8b (red points), conventional broken array multipliers (green points), truncated multipliers (blue points), lpACLib multipliers [3], and other multipliers analyzed in [16] (approximate multipliers (AM), error-tolerant multipliers (ETM), and underdesigned multipliers (UDM)). The grey points in Fig. 4 show all 16,833 non-dominated implementations. To make

TABLE II
THE NUMBER OF GENERATED / NON-DOMINATED IMPLEMENTATIONS OF ARITHMETIC CIRCUITS IN EVOAPPROXLIB

	operand bit-width			
	8	**12**	**16**	**32**
unsigned circuits				
adders	7,614 / 912	5434 / 1,516	2,098 / 939	175 / 84
multipliers	51,882 / 16,833	2,827 / 1,892	17,647 / 4,010	350 / 132
signed circuits				
adders	2,460 / 1,325	509 / 333	777 / 524	565 / 314
multipliers	220 / 108	476 / 220	828 / 387	n/a

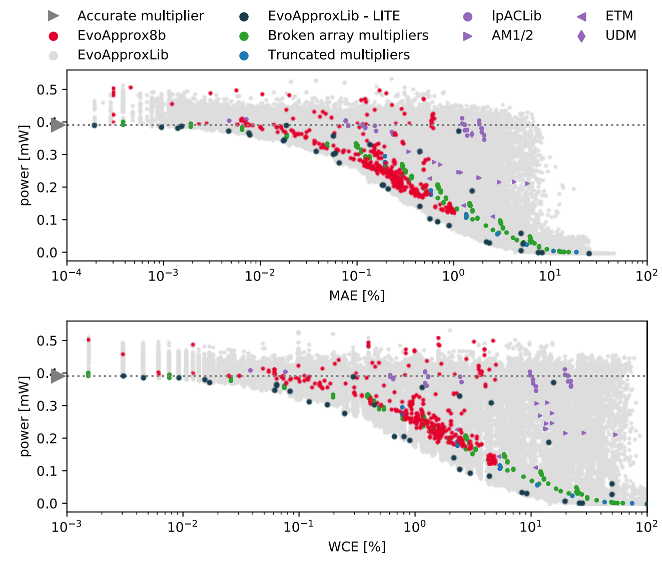

Fig. 4. The 8-bit approximate multipliers (black points) that were selected to EvoApproxLib$^{\text{Lite}}$ from all the discovered approximate multipliers (grey points) and compared to the former version of EvoApprox8b library (red points), broken array multipliers (green points), truncated multipliers (blue points) and other selected approximate multipliers from lpAClib [3] and [16]. Two objectives (MAE vs. power and WCE vs. power) are projected.

Fig. 4 readable, only two objectives (MAE vs. power and WCE vs. power) are projected. Note that EvoApprox8b and results of CGP were also compared with the state of art approximate circuits in [2], [26], [27].

Selected approximate adders and multipliers (as well as other circuits [39]) and their various parameters can be obtained from https://ehw.fit.vutbr.cz/evoapproxlib. The library provides circuit models in Verilog, Matlab, Python, and C. This enables the user to integrate the approximate circuits to hardware as well as software projects and design tools. All approximate circuits can thus be simulated to obtain their other parameters that are not listed on the web site (e.g., the errors under different error metrics or power consumption for another fabrication technology). Every circuit is assigned with a short permanent alphanumeric identifier to identify it uniquely.

V. CNN Resilience Analysis With the Library of Approximate Multipliers

The proposed library allows us to perform the resilience analysis in CNNs in a more complex way than previous methods. Regarding the use of approximate multipliers in CNNs, previous papers studied the impact of the bit width reduction [10], [13], [40] and considered a very limited set of approximate multipliers [9], [36]. Our previous studies utilizing the EvoApprox8b library only applied the approximate multipliers having both operands on the same number of bits [7], [12], [14], [18]. As we generated many different approximate multipliers in Section IV, we can immediately analyze the impact of their utilization not only on the accuracy of classification but also on the power consumption reduction.

In our case studies, we investigate how the approximate multipliers that are introduced to convolutional layers of five ResNet networks [41] can affect the classification accuracy and power consumption. Please note that ResNet CNNs introduced new modules (the so-called residual modules) containing an identity connection such that some layers can effectively be skipped. This technique enabled the update of the weights of earlier layers in very deep CNNs that would normally suffer from the vanishing gradient during training [10]. Fig. 5 shows a typical architecture of ResNet CNN used in our experiments.

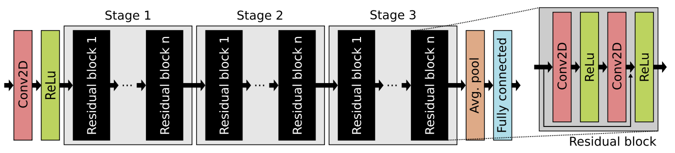

Fig. 5. Architecture of ResNet convolutional neural network with 3 stages and n residual blocks per stage.

TABLE III
Parameters of ResNet CNNs. The Accuracy Is Given on CIFAR-10 (for ResNet-8/-14/-20/-26) and CIFAR-100 (for ResNet-164 v2)

ResNet instance	**# conv. layers**	**mults** $\times 10^6$	**accuracy [%] (floating-point)**	**accuracy [%] (qint-8)**
ResNet-8	7	21.1	83.42	82.85
ResNet-14	13	35.3	85.93	85.81
ResNet-20	19	49.5	88.32	88.09
ResNet-26	25	63.6	90.05	89.70
ResNet-164 v2	163	592.6	74.46	74.27

A. Experimental Setup

Table III summarizes basic parameters of ResNet networks and their classification accuracies after training with the floating-point multiplication and after applying the 8-bit exact multiplier (qint-8). The experiments are performed with ResNet-8, ResNet-14, ResNet-20, and ResNet-26 trained on CIFAR-10 and ResNet-164 v2 trained on CIFAR-100 using TensorFlow [42]. A ResNet CNN utilizing the 8-bit exact multiplier is considered as a golden solution, and all proposed approximations are compared against it. Note that retraining was performed after introducing neither the 8-bit exact multiplier nor approximate multipliers in this CNN error-resilience study. Please note that retraining is, in principle, possible [43].

Hardware accelerators developed to speed up a CNN inference process primarily focus on the MAC operations that are essential in convolutional and fully connected layers. The computation is typically accelerated using a two-dimensional array of processing elements (PE). A typical PE multiplies the input with its weight and updates the sum maintained in each layer. The PE array can be operated in several ways, see [10], [44]. In our study, we suppose that the CNN accelerator running the ResNet is organized as a generic PE array. Since the multipliers participate in the PE energy more than the adders (approx. 6.5x considering 8-bit multipliers and 16-bit adders, also depending on the selected fabrication technology), only the multipliers will be approximated to achieve power savings.

Fig. 6 presents the approach developed to evaluate CNNs utilizing approximate multipliers. The input architectures of neural networks (ResNet-8, -14, etc.) are trained and quantized

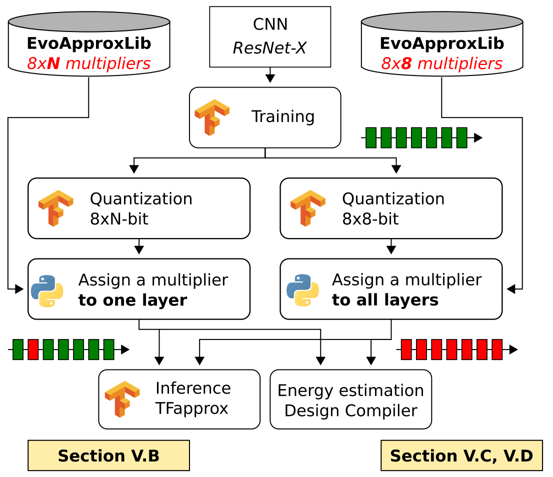

Fig. 6. The methodology developed for error resilience analysis of CNNs utilizing approximate multipliers.

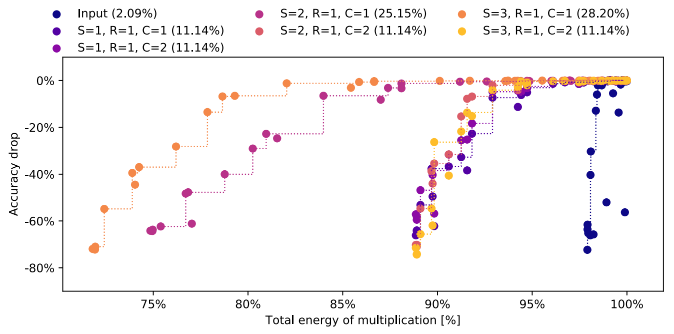

Fig. 7. The classification accuracy drop on CIFAR-10 and the energy of multiplications measured when approximate multipliers are used in one layer of ResNet-8 (with the reference classification accuracy 82.85%). Different layers are represented using different colors and characterized in terms of the number of stages (S), residual blocks (R), convolutional layers (C) and percentage of multiplications.

to 8-bit operations using the TensorFlow framework. By means of approximate multipliers taken for the library, the first experiment reveals to what extent the individual convolutional layers of ResNet are error-resilient (Section V-B). The second experiment reports energy savings in the inference path of ResNet by approximating all convolutional layers in the network using the same multiplier (Section V-C and V-D). In both cases, the accuracy of the CNN employing the approximate convolutional layers is evaluated using the TFApprox tool, which is a Tensorflow extension developed for fast emulation of approximate CNNs on a GPU [45]. The reference energy consumption of one multiplication is obtained by Synopsys Design Compiler (45 nm fabrication technology).

B. Case Study 1: One ResNet Layer Under Approximation

In order to identify a reasonable and diverse subset of 8-bit approximate multipliers, we started with the EvoApproxLib$^{\text{Lite}}$ library. After removing duplicate circuits we ended up with 35 approximate multipliers showing high-quality tradeoffs between power and the five error metrics (Section IV-C).

All exact multiplications of a given layer of ResNet-8 were then replaced by one of the approximate multiplication implementations. This process has been repeated for all the layers and all the 35 approximate multipliers, but only one layer was modified and one type of approximate multipliers was used in each experiment. Fig. 7 shows that the most interesting approximations are obtained if the convolutional layer of the third stage is approximated (see S=3, R=1, C=1 in Fig. 7). As this layer executes 28.2% of all the multiplications, it should undergo the approximation with the highest priority. Introducing the approximate multipliers to the first layer (which performs only 2.09% multiplications) makes a negligible contribution.

C. Case Study 2: All ResNet Layers Under Approximation

This case study deals with a situation in which all 8-bit multiplications of all convolutional layers are replaced with one particular approximate implementation of the multiplier.

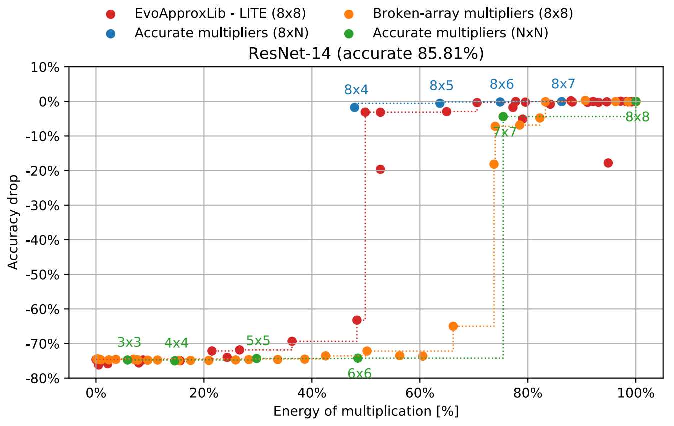

Fig. 8. The classification accuracy drop on CIFAR-10 and the energy of multiplication measured when all multiplications in all convolutional layers of ResNet-14 are performed with the same multiplier.

On the case of ResNet-14 CNN, Fig. 8 shows various trade-offs that can be obtained using a common truncation (green points), BAM multipliers [17] (orange points), 8-bit approximate multipliers taken from our pre-selected set of 35 approximate multipliers (red points) and *accurate* 8xN-bit multipliers (blue points), where $N = \{7, 6, 5, 4\}$ bits are devoted to the weights. Please note that the rounding to N bits is performed offline before the N-bit weights are stored to the weight memory. Unused bits are truncated.

As the 8xN-bit exact multipliers provide high-quality results, it is also worth to analyze the impact of employing the 8xN-bit approximate multipliers on the CNN accuracy and energy requirements. For these purposes, we re-used the approximate multipliers that were evolved in Section IV-A. From all the results reported in Tab. I, 6,345 non-dominated circuits were selected for each 8xN-bit approximate multiplier and utilized in four versions of ResNet. Fig. 9 shows that it is always better to pick a suitable 8xN-bit approximate multiplier than an 8 × 8-bit approximate multiplier because more energy can be saved for a given accuracy drop. Furthermore, the 8xN-bit approximate multipliers accept the weights on fewer bits, and thus, memory requirements are significantly reduced.

TABLE IV

PARAMETERS OF SELECTED APPROXIMATE MULTIPLIERS (EMPLOYED TO MULTIPLY TWO 8-BIT OPERANDS) EXPRESSED WITH RESPECT TO THE EXACT 8-BIT MULTIPLIER AND THE CLASSIFICATION ACCURACY (ON CIFAR-10) OF VARIOUS RESNET NETWORKS UTILIZING THESE CIRCUITS. MUL8U AND MULT8X ARE EVOLVED MULTIPLIERS AND BAM MULTIPLIERS (h AND v ARE THE HORIZONTAL AND VERTICAL BREAK LEVELS) ARE CONSTRUCTED ACCORDING TO [17]

Multiplier	**Relative**	**Error parameters**					**Classification accuracy [%]**			
	Power [%]	*MAE [%]*	*WCE [%]*	*MRE [%]*	*WCRE [%]*	*ER [%]*	*ResNet-8*	*ResNet-14*	*ResNet-20*	*ResNet-26*
8 bit (exact)	100.0	0.00	0.00	0.00	0.00	0.00	82.85	85.81	88.09	89.70
mul8u_QJD	88.0	0.017	0.082	0.51	200.00	74.80	82.61	85.99	88.17	89.96
mul8u_ZFB	77.7	0.059	0.45	0.80	43.56	69.26	82.03	85.76	87.96	89.63
mul8u_NGR	70.6	0.065	0.25	1.90	150.00	96.37	81.02	85.48	88.00	89.76
mul8u_185Q	52.7	0.18	0.79	4.16	125.00	98.05	72.69	82.68	77.55	72.57
mul8u_DM1	49.9	0.20	0.89	4.73	700.00	98.16	62.13	82.71	84.94	83.03
mul8u_12N4	36.3	0.43	2.15	4.20	80.00	87.31	19.30	16.47	22.52	20.80
Accurate 7x7	75.4	0.19	0.78	2.65	100.00	74.61	48.64	77.38	72.32	60.75
Accurate 6x6	48.5	0.58	2.32	7.00	100.00	93.16	12.09	9.99	11.77	11.16
Accurate 5x5	29.7	1.34	5.37	14.12	100.00	97.75	12.95	11.24	11.30	9.78
Accurate 4x4	14.6	2.83	11.33	25.41	100.00	98.88	12.63	11.24	10.85	9.79
Accurate 8x7	86.0	0.097	0.39	1.33	100.00	49.80	81.95	85.73	87.90	89.20
Accurate 8x6	68.4	0.29	1.17	3.54	100.00	74.71	82.28	85.62	87.80	89.57
Accurate 8x5	52.1	0.68	2.72	7.27	100.00	87.16	81.67	85.26	87.88	89.32
Accurate 8x4	35.1	1.46	5.84	13.53	100.00	93.38	76.51	84.05	87.04	88.57
BAM h=0, v=4	90.6	0.019	0.075	0.56	100.00	81.25	82.86	86.07	88.19	89.90
BAM h=0, v=5	83.2	0.049	0.20	1.26	100.00	89.06	80.16	85.72	87.56	89.53
BAM h=0, v=6	73.9	0.12	0.49	2.64	100.00	93.75	63.51	78.60	71.27	62.92
BAM h=1, v=5	73.7	0.13	0.54	2.28	100.00	90.43	41.72	67.65	54.95	44.88
BAM h=1, v=6	66.2	0.20	0.78	3.43	100.00	94.34	16.34	20.80	27.26	22.81
mul8x6u_4Z1	62.7	0.30	1.21	3.90	100.00	94.32	82.52	85.63	87.78	89.44
mul8x5u_410	46.3	0.70	2.80	7.74	300.00	95.36	81.62	84.95	87.80	89.55
mul8x5u_37F	43.7	0.70	2.87	8.05	1100.00	97.34	80.94	85.06	87.75	89.18
mul8x5u_44B	41.4	0.69	2.96	8.31	100.00	97.56	80.16	84.87	87.16	88.77
mul8x5u_43Z	39.9	0.70	2.86	8.46	100.00	97.84	78.90	84.71	87.12	88.87
mul8x5u_31H	39.6	0.70	2.85	8.40	100.00	97.81	77.95	84.11	86.79	88.58
mul8x4u_2YS	31.3	1.47	5.91	13.96	100.00	96.51	76.92	84.09	86.94	88.67
mul8x4u_30G	30.6	1.48	6.01	14.18	100.00	97.13	75.93	83.94	86.71	88.45
mul8x4u_3FA	29.8	1.48	6.11	14.31	100.00	97.95	74.58	83.67	86.19	88.49
mul8x4u_4GC	27.8	1.50	6.13	14.85	700.00	98.31	72.94	82.84	85.51	87.28
mul8x4u_3JN	26.1	1.47	6.11	14.73	100.00	98.64	71.58	82.79	85.75	86.47
mul8x4u_2JV	25.5	1.50	6.23	15.41	100.00	98.49	68.57	81.54	84.59	85.39

Another critical question is if a better trade-off (between the classification accuracy and the energy needed for all the multiplications) is obtained if one introduces more aggressive approximations to a larger ResNet or decent approximations to a smaller ResNet. We analyzed these trade-offs in Fig. 10, which plots the classification accuracy against the total energy of multiplication in all convolutional layers. In this experiment, we considered all multipliers situated on the Pareto fronts visible in Fig. 9. In Fig. 10, one can observe a set of solutions in which a careful approximation of a smaller network always represents a better trade-off than a heavily approximated larger network. Furthermore, a suitable approximate multiplier still provides a better trade-off than an accurate multiplier operated on a reduced bit width. These results are comparable with the ALWANN method [12] (applied on ResNet-8, ResNet-14, and ResNet-50 networks), which is a highly-specialized algorithm for the selection of 8-bit approximate multipliers for each layer of a particular CNN.

Table IV gives a detailed characterization of selected approximate multipliers discussed in this paper and the classification accuracy if these multipliers are employed in all convolutional layers of various instances of ResNet CNNs which are evaluated on the CIFAR-10 data set. Evolved approximate multipliers are compared with common approximate multipliers based on the truncation and BAM algorithm [17]. For example, a 1.83% drop in the accuracy of ResNet-8 can be exchanged for a 29.4% improvement in power consumption if an 8-bit approximate multiplier mul8u_NGR is chosen. For 8×5-bit approximate multiplier mul8 $\times$ 5u_43Z, the power improvement is 60.1% while we lose 3.95% accuracy. The most interesting trade-off is provided by 8×6-bit approximate multiplier mul8 $\times$ 6u_4Z1 whose usage reduces power consumption by 38.3% while the accuracy drop is only 0.33%.

D. Case Study 3: Complex CNNs

In the final experiment, we assess if the findings of the previous section also hold for a more complex network and a more complex problem. We chose a 164-layer ResNet of Bottleneck architecture (denoted as ResNet-164 v2) proposed in [46] for classification on a more challenging CIFAR-100 data set. This moderate-size CNN contains 163 convolutional layers with more than 592.6 million multiplications and achieves 74.46% classification accuracy on CIFAR-100. Fig. 11 shows that ResNet-164 v2 evaluated on CIFAR-100 is much less error resilient than the smaller ResNet networks

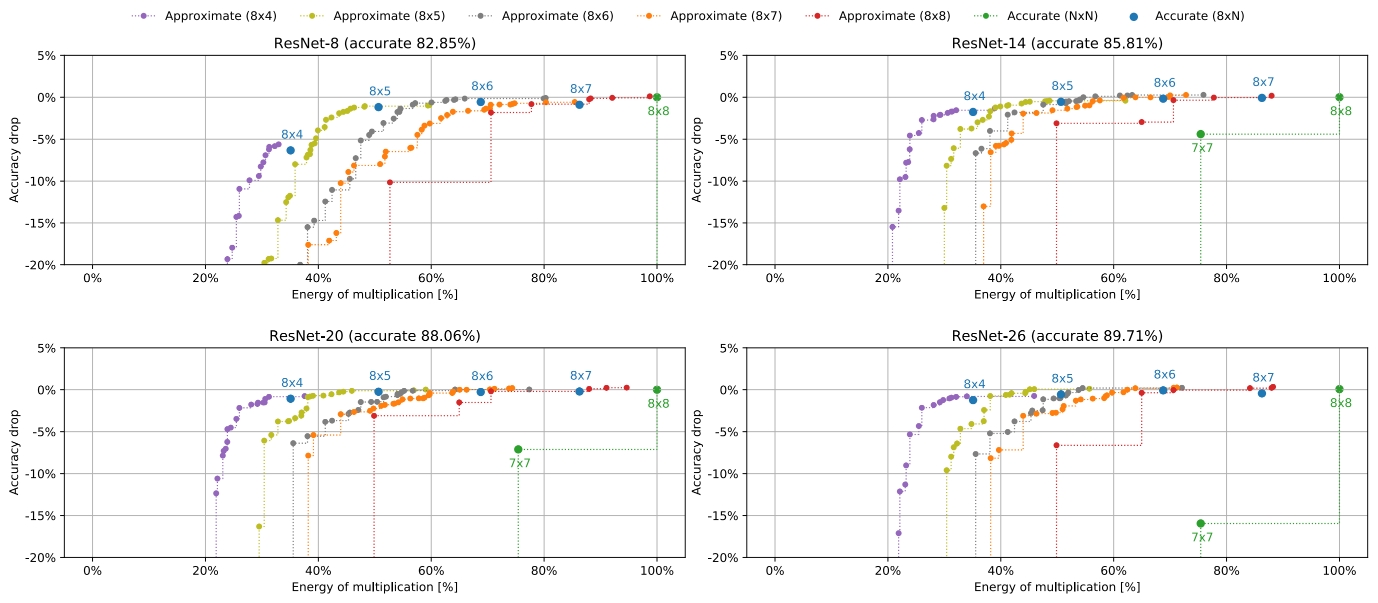

Fig. 9. The classification accuracy drop on CIFAR-10 and the energy of multiplication measured when all multiplications in all convolutional layers of various ResNet CNNs are performed with the same approximate multiplier.

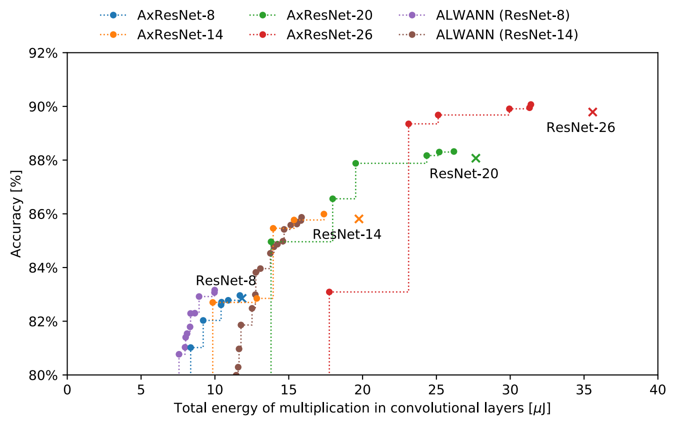

Fig. 10. The classification accuracy on CIFAR-10 and the total energy of multiplication in various approximate implementations of ResNet CNNs (points) and accurate, but quantized ResNet CNNs (crosses) contrasted with the ALWANN method [12].

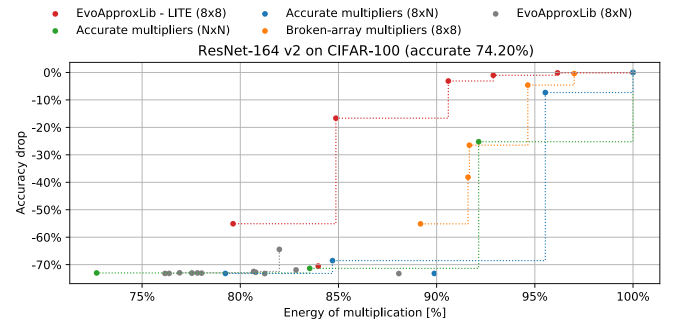

Fig. 11. The classification accuracy drop on CIFAR-100 and the energy of multiplication measured when all multiplications in all convolutional layers of ResNet-164 v2 are performed with the same multiplier.

evaluated on CIFAR-10. A 10% reduction in the energy of multiplication is associated with a non-negligible drop in the accuracy. The 8xN-bit approximate multipliers (we tested $N = \{7, 6, 5, 4\}$) that are beneficial for the smaller ResNet networks are not suitable in this task. This result is, however, somewhat expected and should be interpreted that one has to introduce approximate multipliers to this kind of non-trivial use cases, such as ResNet-164 v2 applied on CIFAR-100, very carefully. Either only some layers should be approximated or the most suitable approximate multiplier should be separately identified for each layer.

VI. Conclusions

In this paper, we presented a large library of approximate adders and multipliers primarily intended to accelerate the design process of energy-efficient hardware accelerators for CNNs and other signal, image, and video processing applications. In greater detail, we focused on the automated design of MxN-bit approximate multipliers. The new version of the library contains 19,067 (including 5,633 non-dominated) approximate adders, 74,230 (including 23,582 non-dominated) N-bit approximate multipliers and 27,175 (including 12,019 non-dominated) 8x{7,6,5,4}-bit approximate multipliers; i.e., in total 101,405 (including 35,601 non-dominated) approximate multipliers. It provides a large collection of circuits (showing various trade-offs between the error and other parameters) and thus implementation options that can easily be exploited in various applications. The user can choose, by means of the user interface available via the library website, the most suitable approximate circuit or a set of approximate circuits for a particular application.

We used the evolved approximate multipliers to analyze how various ResNet versions are error-resilient if approximate multiplication operations are introduced. We also identified the most promising combination of a particular ResNet version and an approximate multiplier for a given energy budget. The MxN-bit approximate multipliers consistently provided better trade-offs than N-bit approximate multiplies when deployed in the convolutional layers of smaller ResNet networks evaluated on CIFAR-10. For the largest ResNet network (ResNet-164 v2) that we evaluated on CIFAR-100, only 8-bit approximate multipliers provided an acceptable solution. Nevertheless, the utilization of the MxN-bit approximate multipliers allows us to reduce the number of bits that have to be available in some registers of the accelerator and also the memory space needed to store the weights of ResNet.

This work was motivated by the fact that multiplication is the key operation of CNN datapaths and, hence, many studies dealing with approximate multipliers in CNNs are available in the literature. In future work, we could apply the proposed approach to analyze approximate adders' usage or various combinations of approximate components in CNNs. In particular, our future work will be focused on searching for the best trade-off between the classification accuracy and energy requirements of a CNN accelerator under the assumption that various CNN architectures can be combined with suitable approximate MxN-bit multipliers (and other approximate components) and memory subsystem organizations. We believe that the usage of suitable correctly-sized MxN-bit approximate multipliers can reduce not only the memory consumption but also the latency and power consumption of the memory subsystem compared with the NxN-bit multipliers.

REFERENCES

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[2] 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 *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 258–261.

[3] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," in *Proc. DAC*, 2016, pp. 1–6.

[4] S. Ullah, S. S. Murthy, and A. Kumar, "SMApproxLib: Library of FPGA-based approximate multipliers," in *Proc. 55th Annu. Design Autom. Conf.*, New York, NY, USA, 2018, pp. 1–6.

[5] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "A test pattern generation technique for approximate circuits based on an ILP-formulated pattern selection procedure," *IEEE Trans. Nanotechnol.*, vol. 18, pp. 849–857, 2019.

[6] P. Detterer, C. Erdin, M. Nabi, J. P. de Gyvez, T. Basten, and H. Jiao, "Trading digital accuracy for power in an RSSI computation of a sensor network transceiver," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 102–107.

[7] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 317–328, Feb. 2020.

[8] Z. Vasicek, "Formal methods for exact analysis of approximate circuits," *IEEE Access*, vol. 7, no. 1, pp. 177309–177331, 2019.

[9] P. Panda *et al.*, "Cross-layer approximations for neuromorphic computing: From devices to circuits and systems," in *Proc. 53nd Design Autom. Conf.*, 2016, pp. 1–6.

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

[11] J. Castro-Godínez, D. Hernández-Araya, M. Shafique, and J. Henkel, "Approximate acceleration for CNN-based applications on IoT edge devices," in *Proc. IEEE 11th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Feb. 2020, pp. 1–4.

[12] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.

[13] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5784–5789, Nov. 2018.

[14] V. Mrazek, L. Sekanina, and Z. Vasicek, "Using libraries of approximate circuits in design of hardware accelerators of deep neural networks," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 243–247.

[15] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[16] H. Jiang, L. Liu, F. Lombardi, and J. Han, "Approximate arithmetic circuits: Design and evaluation," in *Approximate Circuits, Methodologies and CAD*, S. Reda and M. Shafique, Eds. Cham, Switzerland: Springer, 2019, pp. 67–98. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-99322-5_4

[17] 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 Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.

[18] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proc. ICCAD*, 2016, pp. 81:1–81:7.

[19] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "Probabilistic error modeling for approximate adders," *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 515–530, Mar. 2017.

[20] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *Proc. DAC*, 2012, pp. 796–801.

[21] M. Soeken, D. Grobe, A. Chandrasekharan, and R. Drechsler, "BDD minimization for approximate computing," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2016, pp. 1–6.

[22] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2013, pp. 1367–1372.

[23] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda, "Automated high-level generation of low-power approximate computing circuits," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 18–30, Jan. 2019.

[24] Y. Wu and W. Qian, "ALFANS: Multilevel approximate logic synthesis framework by approximate node simplification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1470–1483, Jul. 2019.

[25] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 432–444, Jun. 2015.

[26] M. Ceska, J. Matyas, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 416–423.

[27] L. Sekanina, Z. Vasicek, and V. Mrazek, "Automated search-based functional approximation for digital circuits," in *Approximate Circuits, Methodologies and CAD*, S. Reda and M. Shafique, Eds. Springer, 2019, pp. 175–203.

[28] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 393–401, Feb. 2017.

[29] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "TOSAM: An energy-efficient truncation- and rounding-based scalable approximate multiplier," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1161–1173, May 2019.

[30] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 418–425.

[31] M. A. Hanif, R. Hafiz, O. Hasan, and M. Shafique, "Quad: Design and analysis of quality-area optimal low-latency approximate adders," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[32] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proc. DAC*, 2015, pp. 86:1–86:6.

[33] M. Riaz *et al.*, "CAxCNN: Towards the use of canonic sign digit based approximation for hardware-friendly convolutional neural networks," *IEEE Access*, vol. 8, pp. 127014–127021, 2020.

[34] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 4, pp. 697–711, Dec. 2019.

[35] S. Shakib Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 145–150.

[36] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 2, p. 16:1–16:23, 2018.

[37] J. F. Miller, *Cartesian Genetic Programming*. Berlin, Germany: Springer, 2011.

[38] V. Mrazek, Z. Vasicek, L. Sekanina, H. Jiang, and J. Han, "Scalable construction of approximate multipliers with formally guaranteed worst case error," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 11, pp. 2572–2576, Nov. 2018.

[39] M. Ceska, J. Matyas, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "Adaptive verifiability-driven strategy for evolutionary approximation of arithmetic circuits," *Appl. Soft Comput.*, vol. 95, Oct. 2020, Art. no. 106466.

[40] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Proc. EDAA DATE*, Mar. 2017, pp. 1478–1483.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun./Jul. 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[42] M. Abadi *et al.* (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: https://www.tensorflow.org/

[43] C. De la Parra, A. Guntoro, and A. Kumar, "Full approximation of deep neural networks through efficient optimization," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5, doi: 10.1109/iscas45731.2020.9181236.

[44] M. A. Hanif, F. Khalid, and M. Shafique, "Cann: Curable approximations for high-performance deep neural network accelerators," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[45] F. Vaverka, V. Mrazek, Z. Vasicek, and L. Sekanina, "TFApprox: Towards a fast emulation of DNN approximate hardware accelerators on GPU," in *Proc. EDAA DATE*, Mar. 2020, pp. 1–4.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision—ECCV*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 630–645.

Vojtech Mrazek (Member, IEEE) received the Ing. and Ph.D. degrees in information technology from the Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic, in 2014 and 2018, respectively. He is currently a Researcher with the Evolvable Hardware Group, Faculty of Information Technology, Brno University of Technology. From 2018 to 2019, he was also a Visiting Postdoctoral Researcher with the Department of Informatics, Institute of Computer Engineering, Technische Universität Wien (TU Wien), Vienna, Austria. He has authored or coauthored over 30 conference/journal papers focused on approximate computing and evolvable hardware. His current research interests include approximate computing, genetic programming, and machine learning. He received several awards for his research in approximate computing, including the Joseph Fourier Award for research in computer science and engineering in 2018.

Lukas Sekanina (Senior Member, IEEE) received the Ing. and Ph.D. degrees from the Brno University of Technology, Brno, Czech Republic, in 1999 and 2002, respectively. He was a Visiting Professor with Pennsylvania State University, Erie, PA, USA, in 2001. He received the Fulbright Scholarship to work with the NASA Jet Propulsion Laboratory, Caltech, in 2004. He is currently a Full Professor and the Head of the Department of Computer Systems, Faculty of Information Technology, Brno University of Technology. He has served as an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2011 to 2014, *Genetic Programming and Evolvable Machines Journal*, and *International Journal of Innovative Computing and Applications*. He coauthored over 200 articles mainly on evolvable hardware, evolutionary computation, and approximate computing, and one patent.

Zdenek Vasicek (Member, IEEE) received the M.S. degree in computer science and engineering and the Ph.D. degree from the Brno University of Technology, Czech Republic, in 2006 and 2012, respectively. He is currently an Associate professor with the Brno University of Technology. His research interests include formal verification techniques and application of evolutionary approaches in areas related to the design and optimization of complex digital circuits and systems. He is an active PC member of several evolutionary conferences such as EuroGP, GECCO, and ICES. He received the Silver and Gold medals at HUMIES, in 2011 and 2015, respectively.