Deep Learning-Based Circuit Recognition Using Sparse Mapping and Level-Dependent Decaying Sum Circuit Representations

Arash Fayyazi, Soheil Shababi, Pierluigi Nuzzo, Shahin Nazarian, and Massoud Pedram
Department of Electrical Engineering, University of Southern California
{fayyazi, shababi, nuzzo, shahin, pedram}@usc.edu

Abstract— **Efficiently recognizing the functionality of a circuit is key to many applications, such as formal verification, reverse engineering, and security. We present a scalable framework for gate-level circuit recognition that leverages deep learning and a convolutional neural network (CNN)-based circuit representation. Given a standard cell library, we present a *sparse mapping* algorithm to improve the time and memory efficiency of the CNN-based circuit representation. Sparse mapping allows encoding only the logic cell functionality, independently of implementation parameters such as timing or area. We further propose a data structure, termed *level-dependent decaying sum (LDDS) existence vector*, which can compactly represent information about the circuit topology. Given a reference gate in the circuit, an LDDS vector can capture the function of the gates in the input and output cones as well as their distance (number of stages) from the reference. Compared to the baseline approach, our framework obtains more than an-order-of-magnitude reduction in the average training time and 2× improvement in the average runtime for generating CNN-based representations from gate-level circuits, while achieving 10% higher accuracy on a set of benchmarks including EPFL and ISCAS'85 circuits.**

I. Introduction

Reliance on third-party resources, including third-party intellectual property (IP) cores and fabrication foundries as well as commercial off-the-shelf components, has raised concerns about the insertion of hardware Trojans into fabricated chips. To confront this threat, an approach relies on reverse engineering as a means to rebuild the full functionality of the netlist for further analysis [1].

Gate-level reverse engineering consists of identifying the main functional blocks composing a circuit and their interconnection. The space of possible functional blocks can be defined via a library of components that can include, for example, commonly-used hardware design patterns or custom finite-state machine blocks. The identification problem is usually addressed in two steps [2]–[4]. First, a set of *candidate matches* are identified by mapping candidate blocks of the unknown circuit to components in the library. These candidates are then justified via formal verification based on a formal notion of matching between an unknown circuit block and a library component. Exhaustively justifying all the candidate matches via formal verification may turn into a time-consuming and computationally-demanding task for large circuits; a major challenge in reverse engineering is then to devise fast and efficient methods that can effectively point to a small set of candidate solutions and alleviate the burden of formal verification. This challenge offers the motivation for this work.

A set of structural and functional approaches have been proposed in the literature to match an unknown sub-netlist against an abstract component library, including mining behavioral patterns from simulation or execution traces [5], word-level structure reconstruction [3], or structural and functional analysis of individual gates and sub-modules [6]. In this paper, *we address the problem of deriving a functional description of a circuit from an unstructured netlist by leveraging deep learning and circuit representations based on convolutional neural networks (CNNs).* In doing so, we are motivated by the state-of-the-art performance of machine learning (ML) techniques, based on both convolutional and deep neural networks, for solving challenging problems including classification, language processing, and decision making in a variety of applications – from business, to social work, medicine, and engineering [7] [8].

Recent work shows that ML also promises to reduce the execution time for solving certain problems in electronic design automation (EDA) and VLSI design, such as circuit recognition [9], [10]. Specifically, we are motivated by the work of Dai and Brayton [9], who have recently proposed the use of CNNs for circuit recognition. In spite of the remarkable memory and time efficiency of ML algorithms, a major challenge of CNN-based circuit recognition, in both the training and deployment phases, is to construct compact and efficient circuit representations that scale well for large circuits and prevent overfitting, i.e., do not impair the model's ability to accurately classify data that are outside of the training set. In this work, we address this challenge by investigating the effectiveness of deep learning for the identification of the functionality of datapath elements of a design from a gate-level netlist.

We focus on datapath elements as they include the majority of logic gates in microprocessor-like designs. We propose a novel sparse mapping algorithm, which allows to only encode information about the logic cell functionality, independently of implementation parameters such as timing or area, thus increasing the space and area efficiency of the CNN-based circuit representation used in our algorithms. We further propose a data structure, termed level-dependent decaying sum (LDDS) existence vector, which can compactly represent information about the circuit topology. Given a reference gate, an LDDS vector can capture the function of the gates in the input and output cones as well as their distance (number of stages) from

978-3-9819263-2-3/DATE19/©2019 EDAA

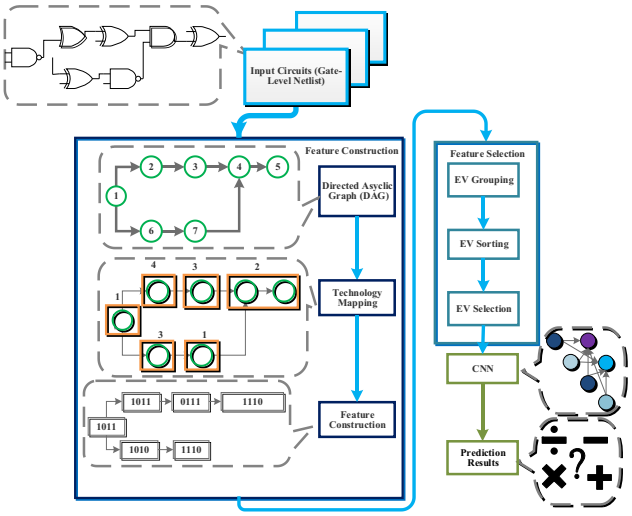

Fig. 1. Flowchart illustrating the proposed framework.

the reference. We implement a deep learning framework that relies on the above data structure and algorithm to recognize the functionality of digital datapath circuits, and can perform better than the state of the art [9] in terms of average training time, execution time, and accuracy.

II. Preliminaries

A flowchart illustrating the proposed framework is shown in Fig. 1. The CNN requires as input a compact vector-based representation of the data to be classified. We call this vector-based representation *existence vector (EV)* and generate it as a part of the feature construction step. In the feature selection step, the fixed-size data, a matrix consisting of multiple EVs, is pre-processed to be fed to the CNN. The CNN leverages this information to classify the operation of each circuit which, in this paper, can be a multiplier, adder, subtractor, modulo, divider, or an unknown operator. The details of each block will be provided in the following subsections.

A. Feature Construction

A critical step in using CNNs for circuit recognition (i.e., to differentiate between different types of circuits) is to convert the circuit structure into a format that is suitable for CNNs, namely a fixed-size real-valued matrix. Naive approaches, based on the adjacency matrix associated with the circuit graph or the AIGER format, may present scalability issues, since their size increases with the size of the circuits.

One approach is to construct these features from smaller circuit elements such as the nodes in a directed acyclic graph (DAG) associated with the circuit, a data structure that is also used in technology mapping [11]. Technology mapping converts a circuit into a DAG with indexed nodes. The idea is then to restrict the set of Boolean functions available for the implementation of the circuit nodes to the ones of a specific standard cell library and map any input circuit to cells in the selected library.

Deriving the functionality of each node in the circuit DAG by technology mapping is not sufficient, since we also need information about how the different nodes are connected, i.e., the edges of the graph. We then use EVs as a data structure, and

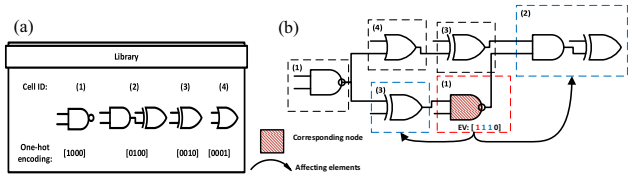

Fig. 2. (a) One-hot encoding of standard cells. (b) Assigning an EV to a circuit node.

generate a vector for each circuit node, following the approach proposed in the literature [9]. To generate an EV for a node, we use one-hot binary coding, as shown in Fig. 2 (a). The i^{th} entry of the EV is set to one if and only if the corresponding node is implemented using the i^{th} cell element from the library. Then, we perform a bitwise OR operation between the one-hot code associated with the node and the ones of all the neighbours, to incorporate information about the nearest neighbours of each gate. An example circuit with the EV of one of its nodes is shown in Fig. 2 (b).

B. Feature Selection

An EV encodes functional and structural features of a circuit node. The number of EVs in a circuit is then equal to the number of gates (nodes) m in the circuit graph. Because the input data to the CNN must have a fixed size regardless of the original circuit size, we need a mechanism to select a fixed-size subset of EVs in each circuit.

The idea is to partition the circuit into a fixed number p of groups, by using topological sort as a suitable approach for group ordering and for sorting the circuit vertices [9]. We then select k representative EVs from each group to be given as features for the CNN, where k can be an arbitrary number. To do so, we use the ranking heuristics suggested in the literature [9], based on selecting the EVs that occur most frequently in each group, or have the highest number of elements set to 1. We finally include these representative EVs, a total of kp vectors, into a matrix with fixed row and column sizes of kp and $|EV|$, respectively.

As the size of the circuit increases, each group covers a larger region, and some representative features of the operator to be recognized can be over-shadowed by sub-circuits in the same group. To increase the accuracy of classification, one solution is to increase the number of groups, which corresponds to increasing the size of the data matrices and the CNNs, hence the resource consumption and runtime. In the following, we describe two approaches that can further reduce the size of the CNN circuit representation while including more information about the circuit structure. Specifically, we present the LDDS encoding approach as an alternative solution to increasing p by enhancing the structural information content encoded for each group.

III. Circuit Representation Improvements

A. Sparse Mapping

We observe that most of the column entries in the CNN input matrices reported in the literature [9] are zeros, which suggests that the cell library based on 4-input lookup tables (4-LUTs), proposed in the literature as a candidate library for mapping, gives unnecessarily high degrees of freedom. To validate this conjecture, we use the forests of trees method to evaluate the

Algorithm 1 Sparse Mapping (SM)

Sparse Mapping (In: network N, library L; Out: Sparsely mapped network: M_s)
1: L_s = PreSparseProcessing(L)
2: G = TransformNetworkIntoAIG(N)
3: ComputeCutsAndMatches (G,L_s)
4: M1 = MappingForMinDelay (G, L_s)
5: M_s = TransformToMappedNetwork (G, M1)
6: Return M_s

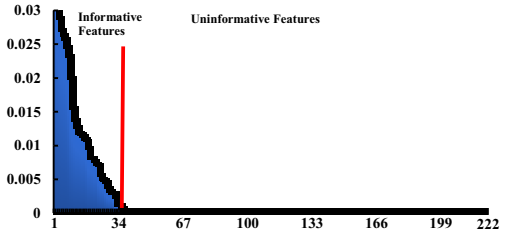

Fig. 3. Importance of constructed features in the baseline model. X-axis show the feature number (the features are sorted based on their importance.)

importance of different features on an artificial classification task [12]. In this method, feature permutation is used to test the actual significance of a feature in the presence of noise (obtained by shuffling the feature of samples). We compute the feature importance as the difference between the baseline (model that was fit to the training dataset) performance and the performance on the permuted dataset. Results show that most of the features extracted in the baseline model are not important (see Fig. 3) and may end up with decreasing the accuracy of the CNN models.

We then propose to use a sparse mapping (SM) algorithm, which is based on the standard cell library mapping version in the ABC framework. The pseudocode of the sparse mapping heuristic is shown in Algorithm 1. To increase the sparsity of the CNN input matrix, we only consider the functionality of the cells in the library as a feature for constructing EVs. There are many cells in the standard cell library with the same functionality (e.g., Inv1, Inv2, …) but different implementation parameters such as delay. In our approach, we categorize all these cells as representing the same cell. This step is termed pre-sparse processing in Algorithm 1 and is performed on library cells. Our compact modeling procedure has multiple benefits. It reduces overfitting and accuracy, since it decreases the impact of noisy, redundant data on decisions [13], in addition to reducing the training time. We leverage ABC [11] to transform a network into an AIG format or to enumerate cuts for the DAG of the input netlist, as reported in the literature [14], but remove all the heuristic iterative optimizations used for delay or area minimization to shorten the generation time of our matrix representation.

B. Level-Dependent Decaying Sum Existence Vectors

We introduce level-dependent decaying sum (LDDS) existence vectors to incorporate information about the circuit structure in a compact way. Given a circuit DAG and a reference node *r,* the level with respect to *r* is the distance (number of stages) from *r* of each vertex in the output or input logic cone of the reference node itself. We construct LDDS vectors as summarized in Algorithm 2. The *mapping* function returns the index of each vertex in the library (i.e., the cell ID). The *FindParents* and *FindChildren* functions return the children and parents of their input vertex. For each vertex, we start by looking

Algorithm 2 Level Dependent Decaying Sum EV

Function LLDS (In: Base vertex, maxLevelDiff, DAG G; Out: EV_{LLDS} representation of input vertex)
LevelDiff = 0; $parents_{number}$ = 1; $Q = \emptyset$
7: $Q = Q \cup base \cup base$
8: While ($Q \neq \emptyset$) do
9: $Q_{temp} = \emptyset$
10: $count = 0$
11: Foreach node $\in Q$
12: If (LevelDiff > 1)
13: $EV_{LLDS}[mapping(node)] += 2^{-\text{LevelDiff}}$
14: Else
15: $EV_{LLDS}[mapping(node)] = 1$
16: If ($LevelDiff < maxLevelDiff$)
17: If (count < $parents_{number}$)
18: If ($FindParents(node)! = G.Roots()$)
19: $Q_{temp} = Q_{temp} \cup FindParents(node)$
20: $parents_{number_{temp}} += Size(FindParents(node))$
21: $count = count + 1$
22: Else If ($FindChildren(node)! = G.Leaves()$)
23: $Q_{temp} = Q_{temp} \cup FindChildren(node)$
24: End Foreach
25: $parents_{number} = parents_{number_{temp}}$
26: $Q = Q_{temp}$
27: $LevelDiff = LevelDiff + 1$
28: End While
29: Return EV_{LLDS}

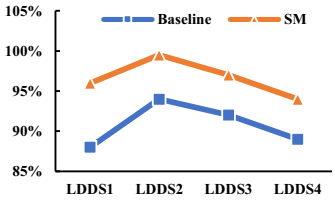

Fig. 4. Decrease of accuracy for LDDS with more than two levels. *LDDSn* means that *n* levels are considered in the computation.

at the immediate neighbors (parent and children nodes) and assigning them a score of 1. As we progress toward further levels (lines 16-24) with respect to base vertex, the scores corresponding to the cells at that level are divided by a constant, which is 2 in our approach (line 13). Therefore, the second-level parents and children will have a score of 1/2, and so on, until we reach roots (leaves) which are primary fanins (fanouts) of the original circuit. The rationale behind this scaling factor (i.e., 2 in our approach) is to force the numerical entries of EV_{LLDS} to be normalized within a fixed range and prevent corruption of the encoded information.

The LDDS approach discounts the effect of the presence of cells "far away" with respect to the current vertex in favour of cells that are "closer." In this study, we stick to 2 levels, since increasing the number of levels resulted in loss of accuracy, as shown in Fig. 4, as well as significantly decreased time efficiency.

IV. Results and Discussion

A. Benchmarks and Simulation Setup

Our framework utilizes ABC [11] to perform technology mapping and then executes the algorithms in Sec. III to compute the CNN input matrices. We build the CNNs and train the models using the Tensorflow package [15]. We select six sets of benchmarks. Division and modulo circuits are based on the reference publication [9]; they are randomly generated in word-

TABLE I RUNTIME FOR CONVERTING AIGS INTO CNN INPUT MATRICES AS WELL AS REQUIRED TIME FOR TRAINING THE CNN

Runtime (s)	LDDS2	SM	LDDS2+SM
Converting AIG to Bool Matrix (78 gates)	0.011	0.001011	0.0057
CNN training (100 epochs)	630.66	51.63	54.36

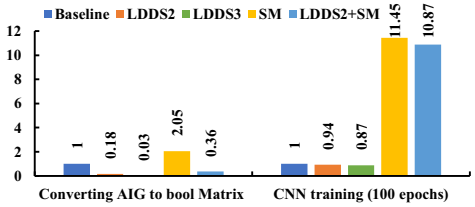

Fig. 5. Speedup for converting AIGs into CNN input matrices and required time for training the CNN with the proposed approaches as well as the baseline approach.

TABLE II THE AVERAGE AND STANDARD DEVIATION OF ACCURACY RATES

Method	Baseline	LDDS2	SM	LDDS2+SM
Accuracy	88.2±3.6	94.3±1	96.4±0.8	99.0±1.0

level Verilog and synthesized into gate-level circuits by Yosys [16]. The same approach is used to generate adder and subtractor circuits. We also used multiplier circuits from [17] and [18], including three multiplier types, "btor", "sp-ar-rc," and "abc." We finally added a separate class consisting of circuits from the ISCAS'85 [19] and EPFL [20] benchmarks, which do not belong to the aforementioned classes and are marked with "unknown." The circuits operate on word lengths ranging from 2 to 32 bits. Numerical experiments are executed on a core i7 3.2-GHz CPU with NVIDIA Tesla K40c GPU. The technology mapping library is a generic 180-nm technology [21]. The number of groups p is 40. For each group, we take the three most representative EVs ($k = 3$). Therefore, the dimension of each input matrix is 120×28.

We build CNNs to classify the type of the circuit, namely multiplier, divider, modulo, adder, subtractor, or unknown operator. All circuits are partitioned into a training set, a validation set, and a testing set, where the size of the validation or testing set is fixed at 50 samples, and the size of the training set is 900 samples. The training set is randomly selected (out of 4500 samples) in each run. For all the experiments, we build CNNs with the following layers in series: (1) a convolution layer with sixty-four 8×8 filters, (2) a Rectified Linear Unit (ReLU) activation layer, (3) a max-pooling layer with filters of size 2×2, (4) a dropout layer with dropout fraction 0.25, (5) a fully connected layer with 32 outputs, (6) a ReLU activation layer, (7) a dropout layer with dropout fraction 0.5, (8) a fully connected layer whose number of outputs is equal to the number of expected classes (6 in our case). The training method follows the ADAM routine [22] while the loss function is the softmax cross entropy.

B. *Training Accuracy and Convergence Speed*

The runtime for converting circuit netlists into CNN input matrices as well as the required time for training the CNN with the proposed approach are shown in TABLE I. The comparison with the baseline approach is in Fig. 5. The pre-processing time grows linearly with the circuit size. The results indicate that the training (conversion) time for our method is 11.44× (2.04×) faster than the baseline method [9]. We believe that this improvement is due to sparse mapping, which produces smaller CNN input feature sizes and a faster training process.

TABLE II lists the average and standard deviation of the accuracy rates for each of the approaches on 100 runs, where all data sets are re-partitioned and reshuffled. The average accuracy of the proposed approach is at least 6% larger than the one previously reported in the literature [9]. We believe this is motivated by the richer structural information encoded via the LDDS approach and the smaller number of features enabled by the SM algorithm. Overall, the numerical results show that the proposed method can distinguish all the mathematical operators, even if they may have similar structure (e.g., adders and subtractors). Moreover, building compact circuit representations indeed helps improve the performance of our implementation.

V. CONCLUSIONS

We presented a deep learning-based circuit recognition framework consisting of a feature extraction stage, a feature selection stage, and a standard CNN. We proposed compact data structures and algorithms to generate circuit representations that can improve the training time and processing time by orders of magnitude with respect to the state of the art. As a future work, we would like to investigate the effectiveness of the proposed approach by training CNN models on a set of circuits with hidden trojans to help detect and locate malware in hardware designs.

REFERENCES

[1] K. Bernstein, "Integrity and Reliability of Integrated Circuits (IRIS)," *DARPA*, 2011.
[2] T. Meade *et al.*, "Gate-Level Netlist Reverse Engineering Tool Set for Functionality Recovery and Malicious Logic Detection," *Int. Symp. Test. Fail. Anal.*, 2016.
[3] W. Li *et al.*, "WordRev: Finding word-level structures in a sea of bit-level gates," in *IEE HOST*, 2013, pp. 67–74.
[4] H. Kong *et al.*,"A Universal Macro Block Mapping Scheme for Arithmetic Circuits," in *IEEE DATE*, 2015, pp. 1629–1634.
[5] W. Li *et al.*, "Reverse engineering circuits using behavioral pattern mining," in *IEEE HOST*, 2012, pp. 83–88.
[6] P. Subramanyan *et al.*, "Reverse engineering digital circuits using structural and functional analyses," *IEEE TETC.*, vol. 2, no. 1, pp. 63–80, Mar. 2014.
[7] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[8] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 Dataset," *online http//www. cs. toronto. edu/kriz/cifar. html*, 2014.
[9] Y.-Y. Dai and R. K. Braytont, "Circuit recognition with deep learning," in *IEE HOST*, 2017, pp. 162–162.
[10] P. A. Beerel and M. Pedram, "Opportunities for Machine Learning in Electronic Design Automation," in *IEEE ISCAS*, 2018, pp. 1–5.
[11] BVSRC - Berkeley Verification and Synthesis Research Center, "ABC - A System for Sequential Synthesis and Verification," *Release 70930*, pp. 1–19, 2013.
[12] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Neural Information Processing Systems*, 2013, pp. 1–9.
[13] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *J. Mach. Learn. Res.*, vol. 3, no. 3, pp. 1157–1182, 2003.
[14] S. Chatterjee *et al.*, "Reducing structural bias in technology mapping," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2894–2902, Dec. 2006.
[15] M. Abadi *et al.*, "TensorFlow: A System for Large-Scale Machine Learning TensorFlow: A system for large-scale machine learning," in *OSDI* , 2016, vol. 16, pp. 265–283.
[16] C. Wolf, "Yosys Open SYnthesis Suite." 2016.
[17] M. Ciesielski *et al.*, "Verification of gate-level arithmetic circuits by function extraction," in *ACM/IEEE DAC*, 2015, pp. 1–6.
[18] D. Ritirc, A. Biere, and M. Kauers, "Column-wise verification of multipliers using computer algebra," in *ACM/IEEE FMCAD*, 2017, pp. 23–30.
[19] D. Bryan, "The ISCAS ' 85 benchmark circuits and netlist format," *North Carolina State Univ.*, vol. 25, no. June, pp. 695–698, 1985.
[20] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL Combinational Benchmark Suite," in *IWLS, number EPFL-CONF-207551*, no. ii, 2015.
[21] C. D. Systems, "Cadence Repository for Electronic Technical Education." [Online]. Available: http://crete.cadence.com.
[22] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *Proc. ICLR*, 2014.