Low-Cost Reservoir Computing Using Cellular Automata and Random Forests

Ángel López García-Arias*, Jaehoon Yu†, and Masanori Hashimoto*
* Osaka University, Osaka, Japan
Email: {a-lopez, hasimoto}@ist.osaka-u.ac.jp
† Tokyo Institute of Technology, Tokyo, Japan
Email: yu.jaehoon@artic.iir.titech.ac.jp

***Abstract*—High-performance image classification models involve massive computation and an energy cost that are unaffordable for resource-limited platforms. As a solution, reservoir computing based on cellular automata has been proposed, but there is still room for improvement in terms of classification cost. This research builds on the previous work introducing enhancements at both the algorithmic and architectural level. Using a random forest classifier with binary features completely eliminates multiplication operations and 97% of addition operations. Also, memory usage can be decreased by pruning 82% of the least relevant augmented features. An architecture with an increased level of parallelism which processes images in a single pass reduces memory accesses, and reduces 60% of logic by optimizing FPGA mapping. These speed, power, and memory optimizations come at an accuracy tradeoff of a mere 0.6%**

I. Introduction

Image classification models have followed the trend of increasing their computational complexity to achieve better accuracy. Despite the rising demand for computer vision applications for embedded and mobile systems, this trend is difficult to follow for these resource-limited platforms. Therefore, models that prioritize processing efficiency rather than accuracy are demanded. Thanks to the simplicity of its computation, reservoir computing has attracted attention as a framework for efficient pattern recognition. In this research we focus on a reservoir based on cellular automata (ReCA) for fast and energy-efficient image classification [1].

Although cellular automata provide a low computation cost reservoir, the inflated amount of features considerably increases computation in the classifier. This research proposes methods for optimizing computation cost and memory usage of the ReCA image classifier by using random forest with binary features as the output pattern analyzer. Also, it presents an architecture targeted at FPGA that improves the reservoir's data usage and hardware mapping.

The rest of this paper is structured as follows. Section II introduces related works and how to reproduce the existing method. Section III proposes algorithmic optimizations to the previous model, and Section IV proposes a hardware accelerator architecture that enhances FPGA implementation efficiency. Finally, Section V concludes this paper and discusses future work.

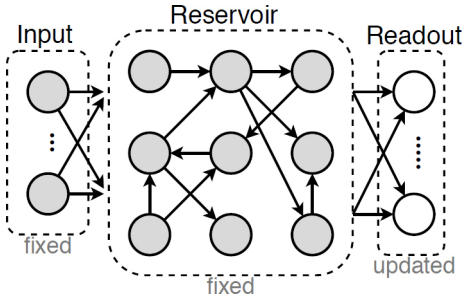

Fig. 1. Reservoir Computing model.

II. Related Work

This section briefly describes reservoir computing, cellular automata, and the previous work on ReCA applied to image classification, in order.

A. Reservoir Computing

Reservoir computing (RC) is a way of circumventing the difficulty of learning the parameters of recurrent layers in recurrent neural networks. As shown in Fig. 1, RC uses fixed weights in the input and hidden layers, only updating in the learning process the weights of the output layer. The fixed hidden layers are referred to as the reservoir, which has to be able to retain the information effectively. The output layer is referred to as the readout, and it consists of a simple classifier.

RC models are not limited to neural networks. There exist algorithmically and physically different types of dynamical systems that serve as reservoirs. Theoretically, any dynamical system can serve as a reservoir, as long as it maps inputs non-linearly into a higher-dimensionality feature space, and has fading memory. Most importantly, it is generally recommended to use a system with behavior on the edge between stability and chaos [2]. In RC models, the simplicity of the readout classifier can be leveraged to implement efficient hardware by coupling it with a computationally cheap reservoir. ReCA has been proposed for this purpose in [3] and [4].

B. Cellular Automata and Elementary Cellular Automata

Cellular automata (CA) are simple dynamical systems with a discrete state that evolves in discrete time steps. The evolution is determined by a simple rule based on the state of the

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

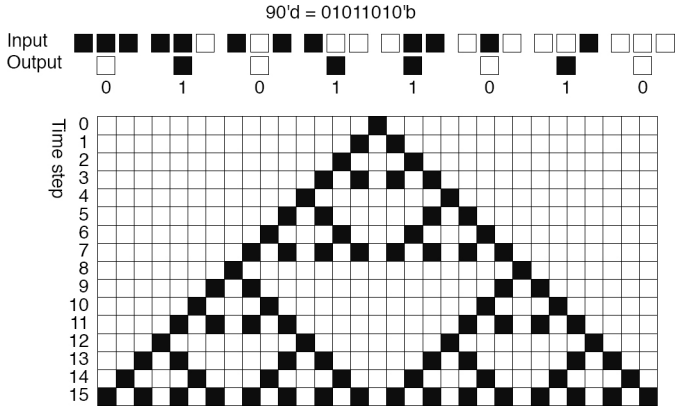

Fig. 2. Rule 90, its Wolfram Code, and the fractal pattern it generates.

neighboring cells and its own. Depending on the rule used, CA can exhibit a wide variety of behaviors, some being capable of computation. These rules, which are close to the edge of chaos [5], can be used effectively as reservoirs .

Elementary cellular automata (ECA) are one-dimensional binary CA. Their evolution is determined by the state of the two neighboring cells and their own, and thus all the $2^{2^3} = 256$ possible rules can be named Rule 0 to Rule 255 by a convention (known as Wolfram code [6]), illustrated in Fig. 2.

C. ReCA Applied to Image Classification

Although RC is mainly used for processing sequential data, ReCA has been used for implementing static image classifiers. The reason for this is the low computational complexity of the model compared to other image classifiers, which can be too high for mobile or embedded systems, as is the case of convolutional neural networks.

Applying CA to images is not trivial. This paper follows the same approach as [1], explained hereafter and illustrated on Fig. 3, to process the 28×28 8-bit grayscale images of the MNIST handwritten digit dataset [7].

First, the input image is split into eight channels: one for each of the 8 bits of the pixels. Each channel is processed independently. The data of each layer serves as the initial state of the ECA, which is applied to rows and columns separately with a fixed boundary condition. The two resulting images are combined with a bitwise XOR. This process is repeated for M iterations, which in addition to the original input, form a set of $M + 1$ images. Lastly, a pooling layer with a 2×2 window, stride 2, and no padding is applied to this set to get the feature vector for the readout.

After testing with all ECA rules, [1] found Rule 90 to be the most effective reservoir, which is consistent with our reproduction experiment. Experiments in [3], [4], [8], and [9] also found Rule 90 standing out in different tasks. One explanation for the better performance of Rule 90 might be its self-replicating nature. From a single non-zero cell, this rule generates the fractal known as the Sierpiński triangle shown in Fig. 2. When applied to multiple pixels, replicated data

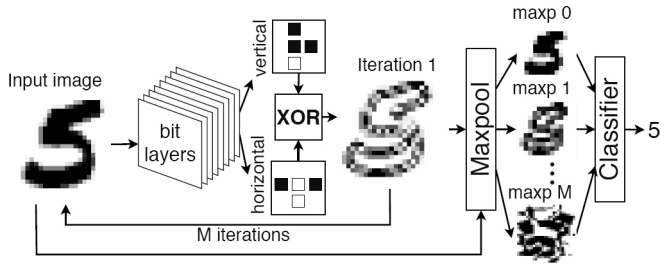

Fig. 3. ReCA image classification scheme.

expands while also interacting with other replicated bits, which may explain the efficiency at feature expansion of Rule 90.

The original implementation uses a linear classifier as readout, trained using a softmax regressor implemented with the limited memory BFGS optimizer provided by the Scikit-Learn [10] library for Python. The MNIST dataset is split between training and test data at a $6:1$ ratio, and the training dataset is augmented twice using elastic distortion. The reproduction of this implementation was able to obtain a test accuracy of 97.3% using the reported hyperparameters (a regularization of $C = 0.04$ and a maximum number of iterations of 1000).

This readout uses all the pixels of the $M + 1$ images to feed a linear classifier, thus having a feature vector of $f = \frac{(M+1)\cdot h\cdot w}{4}$ elements, where h and w are image height and width, respectively. This vector is linearly combined with a $f \times c$ weight matrix to get a score for each of the $c = 10$ classes, being the highest scoring one the classification result. Although this classifier is simple compared to multi-layer neural network classifiers, it has a high computational cost compared to the reservoir. The numbers of multiplications and additions are given by

$$\#\text{ multiplications} = f \cdot c \quad \text{and} \quad \#\text{ additions} = (f-1)\cdot c. \quad (1)$$

The number of operations needed for linear combination scale linearly as the number of features increases, making it ill-suited for a feature expansion model such as ReCA.

III. Proposed Model Optimization

For solving the computational cost and scalability issues mentioned above, the proposed ReCA classifier uses random forests with binary features extracted from a set of features augmented by ECA. This section explains the details of the proposed method and its evaluation results.

A. Random Forest Classifier

Aiming to reduce the computational cost of the ReCA readout, this paper proposes to use instead a random forest (RF) classifier instead of a linear model. RF consists of an ensemble of decision trees, which use comparators as their fundamental operation. Since comparison is performed similarly to addition in standard processors, we consider them equivalent operations. Then, the number of operations in the worst case for RF is given by

$$\#\text{ multiplications} = 0 \quad \text{and} \quad \#\text{ additions} = t \cdot d, \quad (2)$$

where t is the number of decision trees, and d is their maximum depth. RF entirely eliminates multiplication operations, which are the most computationally expensive. Furthermore, its complexity is solely determined by the number and size of the trees, independently of the number of features.

Using $M = 16$ ReCA iterations, an RF classifier with $t = 400$, $d = 20$, and entropy as criterion for splitting nodes, achieved a test accuracy of 97.0%, showing only a slight change in accuracy despite the large computation reduction.

B. Single-bit Features

Since RF uses no multiplication operations, it is unnecessary to merge back the bits coming out of the ECA layers to interpret them as 8-bit unsigned integers. Treating each output pixel as a feature reduces the number of operations required for classification. The number of comparisons needed in the worst case is still determined by $t \cdot d$, but with single-bit features, these comparators are single-bit. Since an 8-bit comparator is built with eight single-bit comparators (full adders), in specialized hardware the number of operations can be reduced by a factor of 8 if using single-bit features. Making the computation at a bit level is straightforward in the hardware implementation, and also simplifies the maxpool layer by avoiding the 8-bit unsigned reinterpretation.

The RF classifier described in Section III-A, trained with single-bit features, achieved a test accuracy of 96.0%.

C. Iteration Pruning

Unlike data augmentation, which only adds computation to the training phase, feature augmentation may increase the complexity of the model both in training and inference. Although this is not a concern with an RF readout, extra features also require more memory and data transfers. Therefore, it is important to have a mechanism that discerns the relevant augmented features and prunes the rest. Since RF does this by design, being able to rank the features by importance, a straightforward approach is to prune the features regarded by the RF as less relevant. Nonetheless, this research suggests a more aggressive approach based on experimental observation.

Not all ECA iterations contribute equally to the readout. Training independent linear classifiers for each iteration of Rule 90 and comparing their classification error suggests that iterations 0 (the original data), 8, and 16 make a significantly bigger contribution than the rest (Fig. 4).

Training the RF classifier with only on iterations 0, 8 and 16, thus pruning 82% of features, achieves a test accuracy of 96.7%, only slightly altering classification accuracy. This suggests that the RF was not selecting these features in the first place, considering them less important. Pruning features can vastly reduce the memory usage in hardware, improving speed and power efficiency.

It is interesting to notice how the classification error per iteration loosely follows the tendency of self-similar Gould's sequence, which counts the number of active cells at each layer of Sierpiński's triangle. The intuition behind this is that valleys of this sequence (found at powers of two) correspond to the iterations when the reservoir is less chaotic. This observation could be used to predict the iterations with a higher contribution to accuracy, pruning the rest to keep the feature vector size from exploding when increasing the number of ECA iterations in the reservoir.

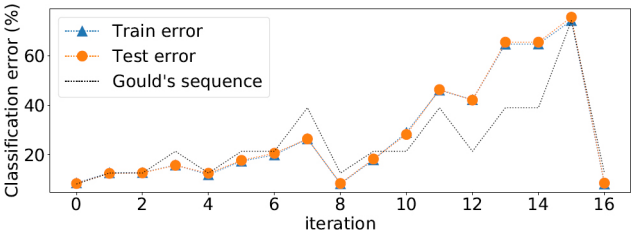

Fig. 4. Classification error for each of the single-iteration trained classifiers, and Gould's sequence scaled for comparison.

TABLE I
NUMBER OF OPERATIONS FOR DIFFERENT CLASSIFIERS

Classifier	Accuracy	# Mult.	# 1b Add.	Features (Bytes)
Linear Combination	97.3%	33,320	266,480	3,332
Random Forest	97.0%	0	64,000	3,332
RF, binary features	96.0%	0	8,000	3,332
RF, pruned bin. features	96.7%	0	8,000	588

D. Evaluation Results

Table I compares the accuracy, number of operations, and size of the feature vector for all the classifiers described in this section. The shown number of additions corresponds to the number of single-bit add operations. After the algorithmic improvements introduced, the ReCA classifier uses 100% fewer multiplications, i.e. no multiplications, 97% fewer additions, and 82% fewer features, with a slight accuracy drop of 0.6%. These reductions directly translate into cuts in computation and memory usage, guaranteeing a faster and more energy-efficient hardware implementation.

IV. Architecture for ReCA Feature Generation

This section describes the hardware architecture for ReCA feature generation, which targets at FPGA implementation.

A. Architectural Principles

In order to achieve a fast and efficient implementation, this architecture has been designed with the following three considerations:

1) Level of Parallelism: The ReCA model can be highly parallelized, since all the ECA are independent of each other, and channels are processed independently. Ideally, a matrix of 28×28 ECA processing units could be employed for each layer, performing an iteration in a single cycle. However, that circuit would have a large area. The proposed architecture parallelizes at layer and line levels by using an array of 8 processing elements (Figs. 5 and 6), which process the 8 channels of one line in parallel.

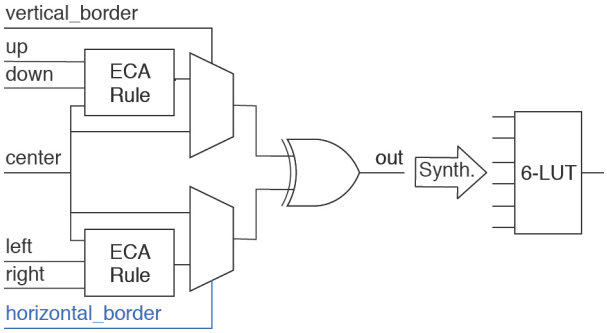

Fig. 5. ECA processing unit. Since the horizontal border is fixed, this circuit is synthesized as a 6-input LUT.

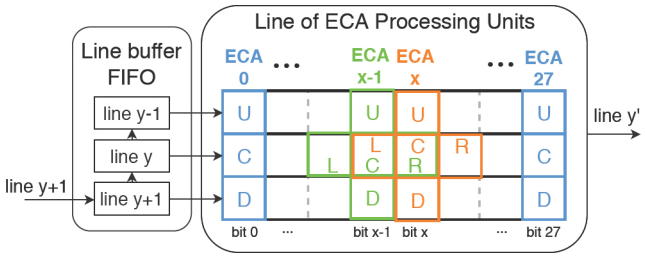

Fig. 6. The Processing Element is formed by a 3-line buffer FIFO and a line of 28 ECA processing units. U, C, D, L and R stand for up, center, down, left and right.

2) FPGA Mapping Optimization: For reducing FPGA resource utilization, the ECA processing unit has been designed to make maximal use of the fundamental building block of FPGAs: the logic block (LB). Specifically, it targets FPGAs whose LB has a 6-input look-up table (6-LUT). Since the design parallelizes at the line level, the horizontal boundary is statically determined by the horizontal position of the ECA processing unit. By setting the horizontal boundary condition as a fixed input, each ECA processing unit is synthesized as a 6-input, 1-output circuit, and thus is efficiently mapped to a 6-LUT (Fig. 5). It shall be noted that this is independent of the chosen ECA rule.

3) Memory Transfer Optimization: Memory read and write operations constitute the main bottleneck for hardware acceleration, in addition to being the most energy expensive. This issue is worse in the case of off-chip memory reads. For this reason, it is crucial to reduce to a minimum the number of times that the accelerator loads data. The proposed architecture does so by processing each iteration in a single pass, loading each pixel only once.

To achieve this, an FIFO buffers three lines of the input (Fig. 6), which is all the data needed to output one line. This data is fed to an array of 28 ECA processing units, which calculate the next state for each pixel of the central line in a single cycle, after what the next line is loaded. At the time a pixel leaves the line buffer, all its neighboring ECAs have been updated, and thus is never loaded again.

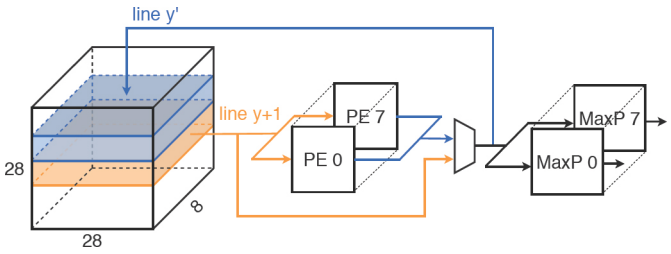

Fig. 7. Diagram of the ReCA accelerator architecture.

TABLE II
FPGA RESOURCE UTILIZATION COMPARISON

	# CLB LUTs	# CLB Registers	# BRAM
This work	392	822	8
Reproduction of [1]	971	822	8

B. ReCA Architecture Design

As illustrated in Fig. 7, the proposed hardware processes each of the eight layers parallelly. The pipeline starts at a double-port BRAM, which contains the input image. Every clock cycle, the PE fetches and processes one image line. The processed line is written back to the BRAM for the next iteration and also fed to the maxpooling unit. Since the maxpool window has a height of 2 and no overlap, the maxpool unit outputs feature data every two cycles.

This design takes 28 cycles to process each iteration, totaling to $17 \cdot 28 = 476$ cycles per input. The first line of the original input is fed in parallelly to the PE and maxpooling unit to avoid an extra latency cycle.

Table II compares resource utilization for the synthesis of the proposed design and a reproduction of the architecture proposed in [1]. By combining vertical and horizontal ECA processing in a single unit, and optimizing it for 6-LUT, this design reduces almost 60% of LUT usage.

V. Conclusion

This paper proposes algorithmic optimizations that eliminate multiplications and largely reduce the rest computation and memory requirements of the ReCA classifier in exchange for a minimal accuracy tradeoff. Additionally, it describes an architecture that optimizes dataflow and FPGA mapping.

Future work will implement in FPGA the proposed architecture in addition to an RF classifier, in pursuance of measures of power consumption and speed.

Further experiments to improve the model may include using different mappings from the input to the reservoir, as the ones employed in [4] and [8], using a multilayer architecture, such as the one proposed in [9], or applying a combination of different CA rules, as explored in [8] and [11]. Additionally, tests with more complex datasets must be performed.

Acknowledgement

This work was supported by JST CREST Grant Number JPMJCR18K2, Japan.

REFERENCES

[1] A. Morán Costoya, C. F. Frasser, M. Roca, and J. L. Rossello, "Energy-efficient pattern recognition hardware with elementary cellular automata," *IEEE Transactions on Computers*, pp. 1–1, 2019.

[2] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, p. 100–123, Jul 2019. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2019.03.005

[3] Ö. Yilmaz, "Reservoir computing using cellular automata," *CoRR*, vol. abs/1410.0162, 2014. [Online]. Available: http://arxiv.org/abs/1410.0162

[4] Mrwan Margem and Özgür Yilmaz, "An experimental study on cellular automata reservoir in pathological sequence learning tasks," 2016.

[5] C. G. Langton, "Computation at the edge of chaos: Phase transitions and emergent computation," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1, pp. 12 – 37, 1990.

[6] S. Wolfram, *A New Kind of Science*. Wolfram Media, 2002.

[7] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[8] S. Nichele and M. S. Gundersen, "Reservoir computing using non-uniform binary cellular automata," *CoRR*, vol. abs/1702.03812, 2017. [Online]. Available: http://arxiv.org/abs/1702.03812

[9] S. Nichele and A. Molund, "Deep reservoir computing using cellular automata," *CoRR*, vol. abs/1703.02806, 2017. [Online]. Available: http://arxiv.org/abs/1703.02806

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] N. R. McDonald, "Reservoir computing and extreme learning machines using pairs of cellular automata rules," *CoRR*, vol. abs/1703.05807, 2017. [Online]. Available: http://arxiv.org/abs/1703.05807