Energy-Efficient Pattern Recognition Hardware With Elementary Cellular Automata

Alejandro Morán, *Student Member, IEEE*, Christiam F. Frasser, Miquel Roca, *Member, IEEE*, and Josep L. Rosselló, *Member, IEEE*

Abstract—The development of power-efficient Machine Learning Hardware is of high importance to provide Artificial Intelligence (AI) characteristics to those devices operating at the Edge. Unfortunately, state-of-the-art data-driven AI techniques such as deep learning are too costly in terms of hardware and energy requirements for Edge Computing (EC) devices. Recently, Cellular Automata (CA) have been proposed as a feasible way to implement Reservoir Computing (RC) systems in which the automaton rule is fixed and the training is performed using a linear regression model. In this work we show that Reservoir Computing based on CA may arise as a promising AI alternative for devices operating at the edge due to its intrinsic simplicity. For this purpose, a new low-power CA-based reservoir hardware is proposed and implemented in a FPGA (known as ReCA circuitry). The use of Elementary Cellular Automata (ECA) is able to further simplify the RC structure to implement a power efficient AI system suitable to be implemented in EC applications. Experiments have been conducted on the well-known MNIST handwritten digits database, obtaining competitive results in terms of processing time, circuit area, power and inference accuracy.

Index Terms—Reservoir computing, machine learning, pattern recognition, cellular automata, hardware implementation

1 Introduction

The proliferation of Internet of Things (IoT) has pushed the horizon of a new computing paradigm, Edge Computing (EC), that incorporates the data processing at the edge of the network [1]. In this new scenario, the development of Artificial Intelligence (AI) inference systems operating at the edge has captured lot of attention in the machine learning community since these systems may assist EC devices to further reduce its dependence with respect to the cloud. Since edge nodes normally present limited processing capability in terms of area and power, state-of-the-art deep learning approaches typically do not provide a feasible energy efficient solution for EC. Therefore, it is of high interest to develop new methodologies for the implementation of energy efficient machine learning hardware in a single chip.

Reservoir Computing (RC) [2], [3] is an attractive machine learning alternative due to its simplicity and computationally inexpensive learning process and that can be suitable for edge applications. In the standard RC approach the input is connected to a randomly initialized Recurrent Neural Network (RNN) and the training process is only applied to the output layer weights using linear or logistic regression. The RC learning methodology has been considered to be a practical alternative to the back-propagation through time (BPTT) algorithm. For these cases the feed forward neural network architecture (FFNN) is commonly used but with a cost of a more complex training methodology if compared with the linear regression that is normally applied to RC. RC has been successfully applied in numerous domains, such as robot control [4], image/video processing [5], wireless sensor networks [6], financial forecasting [7] or to the monitoring of physiological signals [8]. In this context, fast and efficient hardware designs implementing RC systems are an interesting alternative for many of these applications which require a real-time and intensive data processing and/or the use of low-power devices to ensure a long battery lifetime.

Elementary cellular automata (ECA) are the simplest class of 1-dimensional Cellullar Automata (CA) [9] that is found to provide a rich and complex dynamic behavior that is also reproduced by more complex CA schemes. ECA systems consists of a 1D string of cells that can be settled in two possible states (high or low) and evolve in discrete time steps guided by the interaction between nearest neighbors. Since there are a total of $2^3 = 8$ binary states for three cells (i.e., a given cell and its nearest neighbours), there are a total of $2^8 = 256$ possible rules of interaction, which are labeled from 0 to 255 following a standard convention. This convention is illustrated in Fig. 1, in which the interaction rule 30 is shown as an example of application.

Different image analysis and processing methods have been developed based on CA, e.g., [10], [11], [12], [13]. This is the case of the multiple attractor cellular automata (MACA) that has been used to classify 1-dimensional arrays of binary data [14], working as an associative memory and where the processing is restricted to limit cycle dynamics. Another application example in this line is two-dimensional clustering [15], [16], [17], which also relies on the associative memory approach using MACA.

• *The authors are with the Electronics Engineering Group, Department of Physics, University of the Balearic Islands, Ctra. Valldemossa Km 7.5, 07122, Palma de Mallorca, Spain. E-mail: a.moran@uib.eu, {christian.franco, miquel.roca}@uib.es, j.rossello@uib.com.*

Manuscript received 15 Feb. 2019; revised 5 Aug. 2019; accepted 17 Oct. 2019. Date of publication 25 Oct. 2019; date of current version 7 Feb. 2020. (Corresponding author: Alejandro Morán.)
Recommended for acceptance by M. Lipasti.
Digital Object Identifier no. 10.1109/TC.2019.2949300

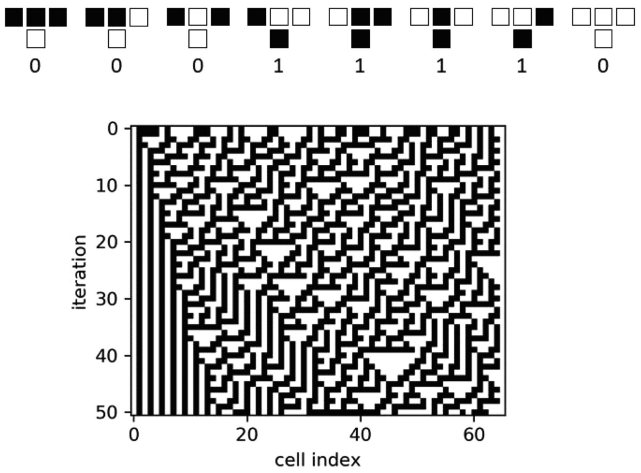


Fig. 1. Basic mechanism of elementary cellular automata (rule 30) showing its temporal evolution from random initial conditions and fixed boundary conditions.

In this work an efficient hardware implementation of a reservoir computing system based on cellular automata (ReCA) is proposed. ReCA systems differ from more traditional associative memory approaches in the inclusion of the transient and chaotic temporal evolution of the automata. Therefore, the dynamics are not restricted to fixed points and limit cycles but extended to transient and chaotic behaviors. Following the developments performed in [18], [19], [20], [21], [22], [23], [24], the temporal evolution of CA (both in the transient and stationary states) is used as a reservoir. The original idea of replacing a reservoir by a cellular automaton spatio-temporal volume was proposed by Yilmaz [20], whose work was extended to deep reservoir computing by Nichele and Molund [25]. In the proposed ReCA system the CA states are readout by an output layer and the CA's initial state is equal to the input signal to be processed. First, we develop an exhaustive study of the accuracy of different ECA rules when applied to a pattern recognition task. The most accurate rule that is finally selected (rule 90) is implemented using a simple digital circuitry. For this method, rule 90 is also the one that provides the best validation fitting using the MNIST database. Interestingly, the use of rule 90 is already proposed by Yilmaz for metric learning as a computationally efficient alternative to radial basis functions in support vector machines [20]. Although this is not the first time in which CA are implemented in digital circuitry for research purposes (see Ref. [26]), to the best of our knowledge this is the first time in which a full ReCA circuitry oriented for pattern recognition is implemented and tested in hardware. The proposed circuitry is synthesized in a medium-sized Field Programmable Gate Array (FPGA) and tested to processing the MNIST handwritten numbers dataset [27]. The proposed model is compared with previously published deep learning implementations showing highly competitive results.

2 Methods

2.1 Notation

We first define some common notation[28], [29], [30] followed in this work. In particular, we use bold capital letters (e.g., $\mathbf{X}$) and bold lowercase letters (e.g., $\mathbf{x}$) to denote matrices and vectors respectively. We employ non-bold letters (e.g., x) to represent scalars, greek letters (e.g., β) for parameters and the hat symbol is used to denote approximations performed by the machine learning system (e.g., $\hat{\mathbf{y}}$ is an approximation of $\mathbf{y}$). The u, x and y letters and all their variants are reserved to be referred to inputs to our system, internal states of the reservoir and output signals coming from the proposed system respectively.

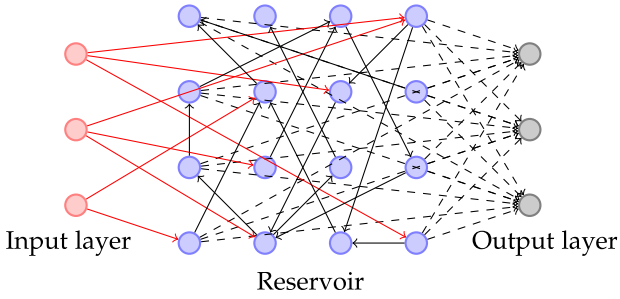


Fig. 2. Reservoir computing scheme. Three blocks: Input layer, reservoir, and the output layer compose the circuitry.

2.2 Reservoir Computing Principles

In the RC architecture, the internal weights between the computational nodes of the reservoir are kept fixed and only the connections to a non-recurrent output layer (the readout) are modified by learning (see Fig. 2). This strategic design avoids the use of complex back-propagation methods, thus reducing the training to a classical linear regression problem. Depending on the typology of the computational nodes, RC systems can be defined as Echo-State Networks, Liquid State Machines or ReCA (when using classical artificial neurons, spiking neurons or CA respectively).

The architecture of a RC system therefore consists of a total of N internal processing nodes, each one providing a given value $x_i(k)$ where $i \in \{1, 2, .., N\}$, N is the number of computational elements in the reservoir, k represents the evolution during time. The reservoir state at a given time evolves according to a nonlinear function:

$$\boldsymbol{x}(k) = F\big[\boldsymbol{x}(k-1), \boldsymbol{u}(k)\big]. \tag{1}$$

That is, the reservoir state depends on the input in this time step ($\boldsymbol{u}(k)$) and the previous reservoir state. The main objective of the reservoir is to correlate the actual input with Q categories to be discovered. For this purpose, and taking a total of $N \cdot M$ descriptors (values of N computational nodes at M different time steps) we construct the spatio-temporal state vector $\boldsymbol{x} \equiv \big(\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(M)\big)$ containing a total of $N \cdot M$ components. The result to be obtained at the output layer of the reservoir is a total of Q values [$\hat{\boldsymbol{y}} = \big(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_Q\big)$], each output representing a given category or variable to be recognized in the case of pattern recognition or a variable to be predicted for regression. The output layer generate the estimation $\hat{\boldsymbol{y}}$ by applying a linear combination of the reservoir states during the M time steps:

$$\hat{\boldsymbol{y}} = \boldsymbol{x}\boldsymbol{W}, \tag{2}$$

where, in the most general case, $\boldsymbol{x}$ is the spatio-temporal state vector composed by $M \cdot N$ components (i.e., the states

of all the neurons $x_i(k)$ values considered during the M time steps), and $\boldsymbol{W}$ is a proper $N \cdot M \times Q$ weight matrix, so that vector $\hat{\boldsymbol{y}}$ contains one prediction for each of the Q categories to infer.

In the present work the reservoir is evolved for each input sample separately, so that the update formula (1) for the ith training sample becomes

$$\boldsymbol{x}_i(k) = F\big[\boldsymbol{x}_i(k-1)\big], \tag{3}$$

with $\boldsymbol{x}_i(0) = G\big[\boldsymbol{u}_i\big]$ as initial condition, where G is any function that maps the input sample to the initial state of the reservoir. Notice that update method (3) is equivalent to imposing that $\boldsymbol{u}(k)$ in (1) changes every M time steps. In addition, and considering a total of L independent samples, we have that:

$$\hat{\boldsymbol{y}}_i = \boldsymbol{x}_i \boldsymbol{W} \quad i = 1, \ldots, L, \tag{4}$$

where these L independent samples will be used in the training set for the estimation of $\boldsymbol{W}$. Each one of the L spatio-temporal feature vectors ($\boldsymbol{x}_i$) are obtained via (3) for L different input samples ($\boldsymbol{u}_i$), and $\hat{\boldsymbol{y}}_i$ is the corresponding $1 \times Q$ vector to be predicted for each of the L measurements. Therefore, only one set of Q predictions is obtained for each independent input sample $\boldsymbol{u}_i$.

To estimate $\boldsymbol{W}$, the different (L) training samples must be considered to adjust these coefficients with respect to the Q expected outputs. Therefore, an $L \times Q$ matrix $\boldsymbol{Y}$ is needed as the ground truth. The qth column ($\boldsymbol{Y}^{(q)}$) is composed by a total of L expected responses for each sample taken as training set. For each sample, $\hat{\boldsymbol{y}}$ is the approximation provided by the reservoir activity so that the error with respect to the expected value $\boldsymbol{Y}_i$ (ith row of matrix $\boldsymbol{Y}$) must be minimized. One of the most simple ways to derive $\boldsymbol{W}$ is through the Moore-Penrose pseudo-inverse of $\boldsymbol{X}$ applied to $\boldsymbol{Y}$. We define the design matrix $\boldsymbol{X}$ as the L different spatio-temporal vector states of the reservoir that arises when the L different inputs $\boldsymbol{u}_i$ stimulate the reservoir. Therefore, X is a $L \times (MN)$ matrix, if no dimensionality reduction is applied. The, $(MN) \times Q$ weight matrix $\boldsymbol{W}$ may be obtained as:

$$\boldsymbol{W} = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{Y}. \tag{5}$$

In contrast, if a dimension reduction H is applied, the feature matrix becomes $H\big[\boldsymbol{X}\big]$, so that $\boldsymbol{X}$ would be replaced by $H\big[\boldsymbol{X}\big]$ in (5) along with the corresponding changes in (4) and the weight matrix dimensions.

Equation (5) is a one-step optimization technique that minimizes the mean squared error between predictions and targets to obtain the key coefficients $\boldsymbol{W}$. It is the standard RC training method for regression and continuous variable prediction tasks. Nevertheless, since the pseudo-inverse is based on the singular value decomposition algorithm, the computation time required to invert a dense matrix $\boldsymbol{X}^T\boldsymbol{X}$ grows with the third power of the number of features [31]. Fortunately, the computation time for first and approximate second order gradient descent approaches scales linearly with the number of features, so that the same least squares solution provided by (5) is obtained. Moreover, since this work targets a classification task with more than two mutually exclusive categories, it is solved using multinomial logistic regression or maximum entropy classifier, which is a linear model that generalizes the logistic regression method to multiclass classification problems [32], and is explained in Section 2.3. In contrast to the least squares objective function, the maximum entropy classifier relies on the minimization of a loss function that is more suitable for (multiple) binary valued labels. From the perspective of the maximum likelihood solution, the least squares optimization assumes that the errors of the linear model are normally distributed, while the multinomial or simple logistic optimization objective assumes a multinomial or binomial distribution for the errors of the linear model [32], respectively. The main difference between both optimization objectives is the effect of large errors. While the least squares loss function penalizes large errors with the square of the squared difference, the logistic loss function penalizes large errors to an asymptotically constant. In addition, rare events with high scores in favour of the correct label would contribute to modify the value of the weights in the case of the least squares optimization, however in the case of the (multinomial) logistic regression this has no consequences in the weight values. Therefore, the maximum entropy classifier presents better results than the Moore-Penrose method in terms of accuracy and computation time. An additional reason to choose this classifier is that it is easier to obtain a better precision for fixed point weights without a significative loss in accuracy if the algorithm relies on stochastic gradient based optimization, as is explained in Section 3.2.

2.3 Softmax Regression

Softmax regression [33], also known as multinomial logistic regression or maximum entropy classifier, is a linear model that generalizes the logistic regression method [32] to classify multiple categories.

We define the Softmax function S as:

$$S(\hat{y}^{(q)}) = \frac{\exp(\hat{y}^{(q)})}{\sum_{q'} \exp(\hat{y}^{(q')})}, \tag{6}$$

which is computed for all samples and categories, i.e., all the components of a prediction row vector $\hat{\boldsymbol{y}}$. From (6) and $\hat{\boldsymbol{Y}}$, we construct the matrix $S(\hat{\boldsymbol{Y}})$ with information of each sample i to belong to a certain category q.

Given the label matrix $\boldsymbol{L}$, the one-hot encoded label for the ith sample is denoted as $\boldsymbol{l}_i$. The outputs finally selected are the most likely categories, $\text{argmax}\,[S(\hat{\boldsymbol{Y}})]$, or the one-hot encoded output $\hat{\boldsymbol{L}} = \text{onehot}\{\text{argmax}\,[S(\hat{\boldsymbol{Y}})]\}$.

To explain the softmax regression, lets first consider we have only one training example ($L = 1$). The first step is to compute the logit vector $\hat{\boldsymbol{y}}$. Next, the logit vector is converted to probabilities via a Softmax function (6). The model is trained by minimizing the cross-entropy between $S(\hat{\boldsymbol{y}})$ and the known one-hot encoded label $\boldsymbol{l}$:

$$\mathcal{D}(S(\hat{\boldsymbol{y}}), \boldsymbol{l}) = -\sum_q l^{(q)} \log S(\hat{y}^{(q)}), \tag{7}$$

where $\boldsymbol{l}^{(q)}$ is one or zero depending whether the index q matches the corresponding category or not. Equation (7) measures the Kullback-Leibler divergence between the probability vector $S(\hat{\boldsymbol{y}})$ and the actual one-hot encoded label (this divergences may be understood as being a kind of

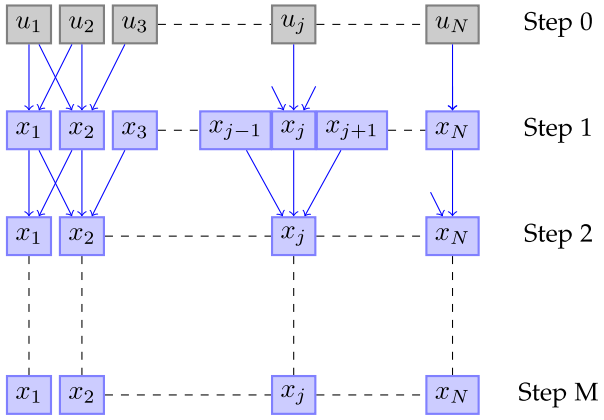


Fig. 3. Reservoir computing scheme using cellular automata. The input is stored in the reservoir at the beginning of the computation and evolved in time during different time steps.

distance between probability distribution functions). For the general case in which a total of L samples are considered, we minimize the average cross-entropy, which is often referred to as the cross-entropy loss. Generalizing to $L > 1$, a more general form of the loss function is

$$\mathcal{L}\big(S(\hat{\boldsymbol{Y}}), \boldsymbol{L}\big) = \frac{1}{L}\sum_{i=1}^{L} \mathcal{D}(S(\hat{\boldsymbol{y}}_i), \boldsymbol{l}_i) + \frac{1}{C}||\boldsymbol{W}||_F, \tag{8}$$

where the first term is the average cross-entropy and the second is a regularization term where $||\cdot||_F$ denotes the Frobenius norm [34]. Notice that a small value of the regularization parameter (C) penalizes large weights to avoid overfitting. This loss function can be minimized using gradient descent or more sophisticated optimization methods. In our case, we used the limited memory BFGS algorithm [35] provided by the Scikit-learn Python module [36] using double precision floating point numbers. However, for the hardware implementation we trained the model using the Adam stochastic optimization method [37] and direct (fake) trained quantization in 8 bits to decrease the number of logic elements and computational power.

Softmax regression has three main advantages (A to C). (A) It can handle multiclass classification problems. (B) Once the model is trained, it suffices to compute $\hat{\boldsymbol{Y}}$ to make a prediction, since the most likely categories are $\operatorname{argmax}(\hat{\boldsymbol{y}}_i)$. (C) The loss function (8) is convex [38], i.e., it has only one global minimum.

2.4 Fundamental Behavior of ReCA Systems

A ReCA system is obtained when the state of each internal node of the reservoir is binary (zero or one) and the dynamics of the reservoir obey one or more CA rules. For ECA, considering one input sample $\boldsymbol{u}$, Equation (3) is transformed to:

$$x_j(k) = F\Big[x_{j-1}(k-1), x_j(k-1), x_{j+1}(k-1)\Big], \tag{9}$$

where F is a boolean function and $x_j \in \{0, 1\}$ is the value of the jth cell. Therefore, as depicted in Fig. 3, the state of the jth cell depends on the state of its neighbors (x_{j-1} and x_{j+1}) and itself (x_j) evaluated in the previous time step for all the binary cells. The spatio-temporal reservoir state $\boldsymbol{x}$ constructed from the automaton evolution for M time steps $\boldsymbol{x} = [x_1(0), x_2(0), \ldots, x_N(0), x_1(1), \ldots, x_N(M)]$ is used to evaluate the logit vector $\hat{\boldsymbol{y}} = H[\boldsymbol{x}]\boldsymbol{W}$. Therefore, in the case of temporally uncorrelated input data, a ReCA system using ECA rules can be understood as being a time-multiplexed one-dimensional Deep Convolutional Neural Network (CNN) with fixed weights defined by the rule of the automaton.

2.5 ReCA Systems Applied to Image Classification

For the special case of RC implemented using the dynamic of ECA (i.e., ReCA systems restricted to ECA), the intrinsic one-dimensional structure with binary states of ECA systems has to be adapted to higher dimensions. In the case of grayscale images, we deal with two spatial dimensions where each pixel is characterized with a 8 bit signal value. In the case of RGB images, we should deal with three dimensions since each color channel (red, green and blue) has its own layer of intensities. So, the first question we address in this section is how do we extrapolate ECA, (optimum to iterate a 1-dimensional binary array) to process a higher dimensional signal.

For 2D grayscale images we iterate rows and columns independently using ECA rules with fixed boundary conditions and combine the resulting vectors using a bitwise XOR operation. The vectorized image $\boldsymbol{u}$ is decomposed in 8 layers corresponding to each binary digit of each pixel:

$$\boldsymbol{u} = \sum_{l=0}^{B-1} 2^l \boldsymbol{u}^{(l)}. \tag{10}$$

where $\boldsymbol{u}^{(B-1)}$ and $\boldsymbol{u}^{(0)}$ are the most and less significative bits of $\boldsymbol{u}$ respectively. We construct a total of B layers of reservoir automata that are evolved in time so that we define a time-varying integer state vector $\boldsymbol{x}'(k)$ as:

$$\boldsymbol{x}'(k) = \sum_{l=0}^{B-1} 2^l \boldsymbol{x}^{(l)}(k). \tag{11}$$

where $\boldsymbol{x}^{(l)}(k)$ is a boolean that is the result of combining two time-dependent ECA processing blocks (one for columns and the other for rows). Its initial value is $\boldsymbol{x}^{(l)}(0) = \boldsymbol{u}^{(l)}$ and the dynamics is governed by a cellular automaton. Therefore, the CA rules are applied to each layer, with no communication between layers. The dynamic behavior of the classifier is illustrated in Fig. 4 using a MNIST sample. As can be seen, the input image $\boldsymbol{u}$ is evolved in time during a total of M time steps (thus creating a time-dependent integer state signal $\boldsymbol{x}'(k)$). Note that in the particular case of grayscale images, the vectorized binary tensor $\boldsymbol{x}(k)$ has 3 dimensions (the width, w, the height, h, and the depth, B).

As commented, rows and columns are iterated independently applying the same automaton rule repeatedly for each iteration, starting from the input image $\boldsymbol{u} \in \mathbb{N}^{w \cdot h}$. Let g^k be the function g applied k times, e.g., $g^3(\cdot) = g(g(g(\cdot)))$. Then, we define the ReCA states associated to the rows with a time evolution defined by the next expression:

$$\begin{aligned} \boldsymbol{x}_r^{(l)}(k) &= g_r^k(\boldsymbol{u}^{(l)}) \\ &= \begin{cases} \boldsymbol{u}^{(l)} & k = 0 \\ g_r\big(\boldsymbol{x}_r^{(l)}(k-1)\big) & k > 0 \end{cases} \qquad \forall l \in [0, B-1]. \end{aligned} \tag{12}$$

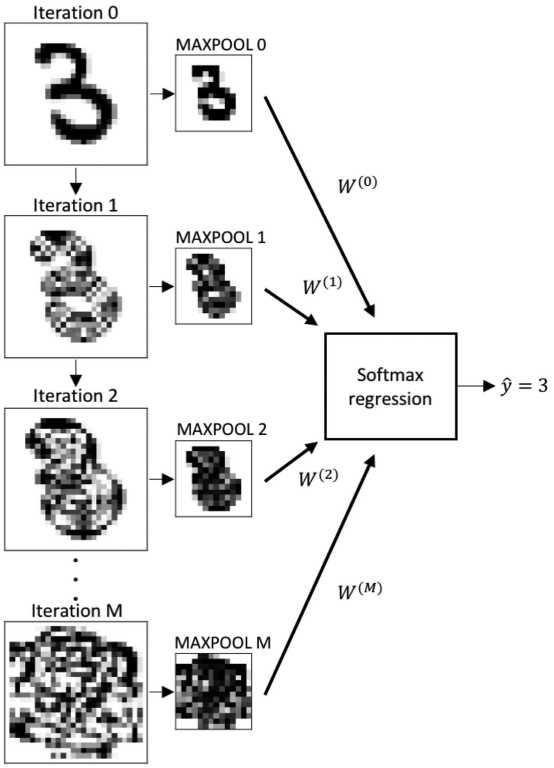


Fig. 4. Scheme of the proposed classifier applied to a MNIST sample.

where the function $g_r : \{0,1\}^{w \cdot h} \rightarrow \{0,1\}^{w \cdot h}$ iterates the rows of the binary layers of the input from the most to the less significative digits. Similarly, the columns' iterations of the input are given by

$$\begin{aligned} \boldsymbol{x}_c^{(l)}(k) &= g_c^k(\boldsymbol{u}^{(l)}) \\ &= \begin{cases} \boldsymbol{u}^{(l)} & k = 0 \\ g_c\big(\boldsymbol{x}_c^{(l)}(k-1)\big) & k > 0 \end{cases} \qquad \forall l \in [0, B-1], \end{aligned} \tag{13}$$

with $g_c : \{0,1\}^{w \cdot h} \rightarrow \{0,1\}^{w \cdot h}$.

Finally, these two half parts of the full ReCA system obtained independently $(\boldsymbol{x}_r(k), \boldsymbol{x}_c(k))$ are combined using a XOR function

$$\boldsymbol{x}^{(l)}(k) = \boldsymbol{x}_r^{(l)}(k) \oplus \boldsymbol{x}_c^{(l)}(k) \qquad \forall l \in [0, B-1], k > 0, \tag{14}$$

where $\oplus$ represents a bitwise exclussive-OR operation that is restricted to the intersection bit between the row and column vector. A particular case of $\boldsymbol{x}'(k)$ obtained via (11) is depicted in Fig. 4 for several iterations starting from the original image (iteration 0) to the final one (iteration M). It should also be noted that from the different boolean functions that were tried in the numerical experiments, both the XOR and XNOR logic operations correspond to the best results. In addition, the performance is similar using OR and NAND logic operations to combine $\boldsymbol{x}_r(k)$ and $\boldsymbol{x}_c(k)$, but this is not the case for AND and NOR logic operations, since the proportion of zeros is higher, so that the contribution of each iteration to the output layer is more sparse compared to XOR/XNOR or OR/NAND combination choices. For simplicity, we maintained equal both iterative rules g_c and g_r

The method can be generalized to higher dimensions (e.g., in the case of RGB images), following the same convention used for rows and columns in Equations (12) and (13), and then combining all the contributions using a single XOR to obtain $\boldsymbol{x}^{(l)}$, as in Equation (14).

Fig. 5. Some samples taken from the MNIST database.

To obtain invariance under small translations and reduce the number of weights in the output layer, we apply max pooling filters to the original image and after each iteration (illustrated in Fig. 4). Pooling layers are selected to have a stride of 2, zero padding and squared window of size 2.

In contrast to the evolution of the automaton iterated by rows and columns, max pooling is applied to each iteration. Therefore, after applying max pooling and considering that w and h are multiples of 2, (as is the case of the MNIST digits) then the input to the softmax classifier for M iterations is $H[\boldsymbol{x}] = \text{maxpool}(\boldsymbol{x}') \in \mathbb{N}^{\frac{w}{2}\frac{h}{2} \cdot M}$. Therefore, the total number of weights used by the model is $\frac{Mwh}{4}$. Ideally, the feature vector must not include redundant information, so that a rich non-linear behaviour to obtain a good classification accuracy is necessary.

3 Results

The study of the software and hadware implementations of the ReCA system have been done with the MNIST database and is divided in two parts. First, we use a double precision software simulation from which we select the ECA rule with better results. Next, the selected classifier is trained also by software using 8-bits precision weights that are used in a final FPGA implementation. The training is carried out using mini batches and Adam optimization method, reaching equivalent performance than with double precision weights.

The MNIST database consists of 70000 squared, grayscale digit images of 28×28 pixels, some examples are shown in Fig. 5. These 70000 images are divided in two sets: 60000 for optimization and 10000 for testing. The model is adjusted using a training and a validation set. In our case, 55000 samples of the first set are used to optimize the weights, while the other 5000 samples are used as validation set to tune the hyperparameters (e.g., automaton rule, iterations and regularization parameter).

3.1 Software Exploration

Using the reservoir methodology described in Section 2.5 we check the performance of all the ECA rules in the processing

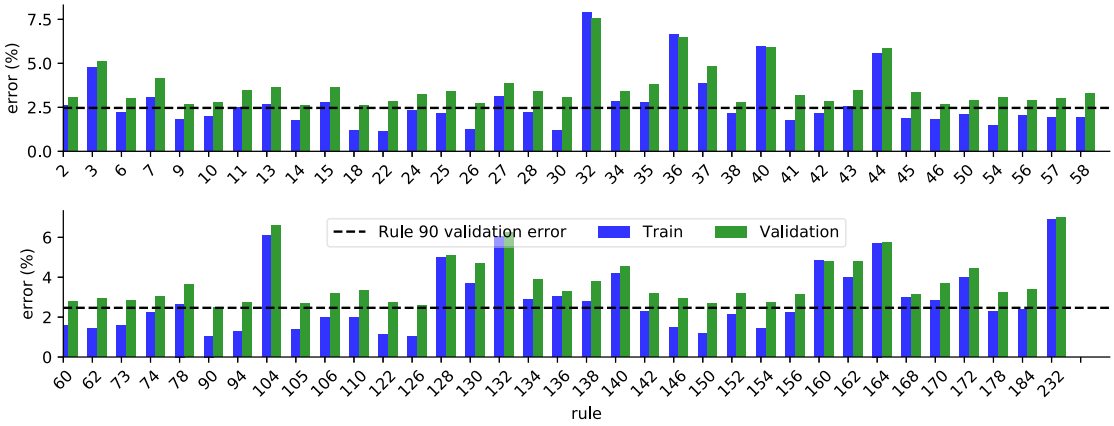


Fig. 6. Performance of the proposed architecture using ten iterations of different, non-symmetric elementary cellular automata rules. Rules with vanishing or period two and temporal evolution without transient state are not shown in this figure.

of the MNIST database. In Fig. 6 we show the results obtained for all the ECA rules that do not share any symmetry. Also, some class 1 and class 2 rules are not shown since they give worse results than the ones presented in this figure. These results have a clear correlation with the class of the ECA rule [9]:

- *Class 1*: Once the evolution reaches a uniform state, adding more iterations does not improve precision because new iterations are a linear combination of the previous iteration, e.g., rule 32.
- *Class 2*: There are two cases for this class of ECA rules. The first case is similar to class 1 ECA rules, once the evolution reaches a uniform or period two state, the precision is no longer improved as the number of iterations increase, e.g., rule 36. The second case are class 2 ECA rules with higher precision, for which the uniform or periodic behaviour is also reached, but the number of needed iterations is higher, e.g., rule 156.
- *Class 3*: Since the boundary conditions are fixed, at some point the behaviour will become cyclic, but the period is higher than for Class 2 ECA rules and the behaviour appears to be random starting from random initial conditions. In addition, since the dynamics are more complex than in previous cases, it is more complicated to find linear dependencies between the new iteration and linear combinations of previous iterations, thus providing potentially higher accuracy e.g., rule 90.

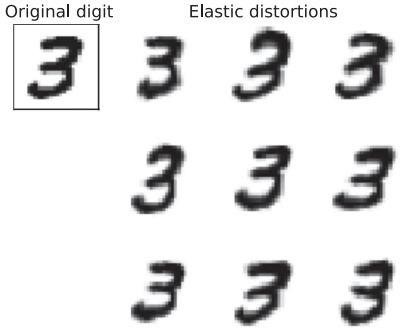


Fig. 7. Example of nine elastic distortions applied to a sample MNIST digit using $\alpha_d = 30$ and $\sigma_d = 5$.

- *Class 4*: As in the case of class 3 ECA rules, since the boundary conditions are fixed, they become cyclic after some period. These rules form areas of repetitive structures and structures that interact in complicated ways. Therefore, it is possible to obtain linearly independent new iterations for a long period, which might imply higher accuracy, e.g., rule 110.

From these results we select rule 90 since it is providing one of the lowest errors and a simple digital implementation.

To improve the training performance, we expand the training set through elastic distortions [39]. An example of this data augmentation (DA) technique is illustrated in Fig. 7 using two parameters. The first (α_d) quantifies the amount of displacement per pixel, and the second (σ_d) the standard deviation of a Gaussian kernel that is applied to convolve the image.

In Fig. 8 we show the error obtained as the size of the reservoir is increased (number of iterations). Also, the training set is augmented twice when $M = 11$ and $M = 14$. When the gap between the training and validation errors becomes large enough and results cannot be further improved via a higher regularization, we add elastic distortions (each training set increment is indicated with a vertical dashed line in Fig. 8). The training stage has been performed using 55000 samples for $M < 11$, 110000 samples for $11 \leq M < 14$ and 165000 for $M \geq 14$.

3.2 Hardware Implementation of the Proposed Model

The full ReCA system, which includes both the CA reservoir along with the output layer, is designed in hardware for its implementation in a FPGA device. The internal registers of the ReCA circuitry are stored with the weights obtained in the offline training performed by software. For the hardware implementation, an 8-bit weights model is utilized since the experimental results are comparable to the double precision ones obtained in the simulations.

The training is done using Adam [37], which is an algorithm for first-order gradient based stochastic optimization, applied to Equation (8) in our case. On the one hand, this algorithm is first-order because, unlike the limited memory BFGS method (in which an approximation of the Hessian matrix is computed) only the gradient and a few additional calculations are needed to update the weights. The gradient of the loss function (8) is computed using a randomly selected

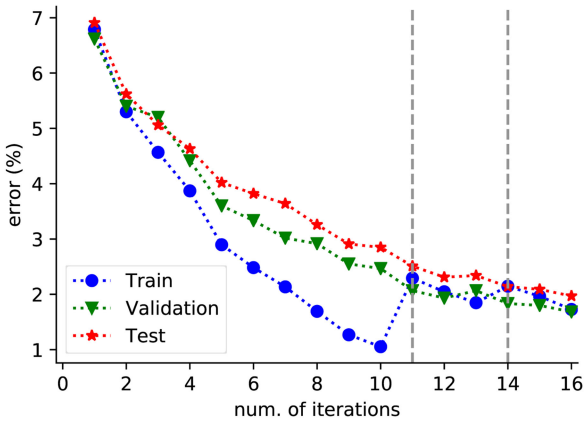


Fig. 8. MNIST training, validation and test error, using rule 90 as described in section 2.5, adding 55000 additional samples using elastic distortions every vertical dashed line (at $M = 11$ and $M = 14$). The regularization parameter is $C = 0.04$ and the maximum number of limited memory BFGS iterations is fixed to 1000.

fixed-size subset (or mini batch) taken from the training set. The weights are updated and the mini-batch changed each training step.

The loss function expressed in (8) ($\mathcal{L}(W)$ for short) is a noisy objective function when using batches. During the training process this function changes since both the input data and the parameters change. To indicate this dependence with time step t, we use $\mathcal{L}_t(W_t)$ instead, so that the gradients are $g_t = \nabla_W \mathcal{L}_t(W_t)$. Since the objective is to minimize the expected value $\mathbb{E}[\mathcal{L}_t(W_t)]$, the algorithm updates exponential moving averages of g_t and g_t^2, which are referred to as m_t and v_t respectively. To control the decay rates of these two exponential moving averages, two new hyper-parameters are introduced: $\beta_1, \beta_2 \in [0, 1)$, in addition to the learning rate α.

The algorithm is implemented as follows. At the beginning ($t = 0$), both moving averages are initialized to zero, i.e., $m_0 = v_0 = 0$, and the first mini batch is selected. Then, for each new training iteration, the gradients g_t are computed to update the moving averages as

$$m_t = \frac{\beta_1 m_{t-1} + (1 - \beta_1) g_t}{1 - (\beta_1)^t}, \tag{15}$$

$$v_t = \frac{\beta_2 v_{t-1} + (1 - \beta_2) g_t^2}{1 - (\beta_2)^t}. \tag{16}$$

TABLE 1
This Table Specifies the Hyperparameters of the Model with 8-Bits Weights

Hyperparameters of the quantized model	
CA rule	90
CA iterations (M)	16
Optimization method	Adam* ($\beta_1 = 0.9$, $\beta_2 = 0.999$)
Learning rate	0.008
Regularization strength	0.00012
Batch size**	17000
Distortions per image ($\alpha_d = 30$, $\sigma_d = 5$)	3

(*) We Used the Tensorflow [40] AdamOptimizer class. (**) Mini-Batches are Chosen at Random for each Optimization Step.

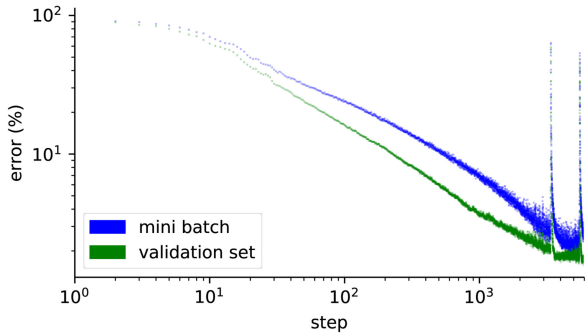


Fig. 9. Training process for the quantized model using the hyperparameters specified in Table 1.

Finally, the weights are updated as

$$W_t = W_{t-1} - \alpha \frac{m_t}{\sqrt{v_t}}, \tag{17}$$

and the process is repeated using a new mini batch until convergence of W_t. Note that the term $m_t/\sqrt{v_t}$ is a smoother version of the gradient that depends on previous computations. Thus, the variance of the stochastic trajectory of $\mathcal{L}_t(W_t)$ becomes smaller compared to simple stochastic gradient descent, which also results in faster convergence.

Rule 90 and a total of 16 iterations are implemented in the hardware (all the parameters are listed in Table 1). The training process using these parameters and Adam optimization method is illustrated in Fig. 9. In this figure we represent the error in each mini-batch (blue) and the error in the validation set (green) as a function of the number of optimization steps, i.e., the number of times the optimization algorithm has been applied. After a large number of optimization steps, we observed peaks of poor performance due to 8-bits quantization restrictions, which slows down the training process. As a result we obtained a 1.92 percent of error in the test set.

The proposed method has been implemented in hardware using weights estimated offline. Fig. 10 shows the general scheme of the implemented circuitry, which has been synthesized on a Cyclone V 5CSEBA6U23I7 SoC FPGA and reported a logic utilization of 22.6K ALM (20.5 percent of the available ALM resources). According to the *PowerPlay Early Power Estimator* [41] provided by Intel®, this inference hardware has a power consumption of 0.289 W. In this particular case, the design also makes use of 40 DSP blocks, 4 of them for each category, so that the contribution to the matrix multiplication of each iteration is parallelized across 4 DSP blocks for each output. Note that this choice is quite arbitrary and the number of DSP blocks is limited by the specific device. Depending on the dimensions of the input data and the number of required iterations for a specific task, the number of DSP blocks could be chosen to minimize the latency.

For the sake of completeness, a high-level scheme of the implemented hardware architecture is provided in Fig. 10, which together with core of the ReCA processing shown in Fig. 11 implements all the steps of the algorithm. The rule 90 processing unit (R90PU) depicted in Fig. 11 is used for each row and column, which are iterated independently, and the same structure is replicated for the 8 grayscale bits. From this figure we also emphasize that rule 90 is not only the one with the lowest prediction error, but also one with

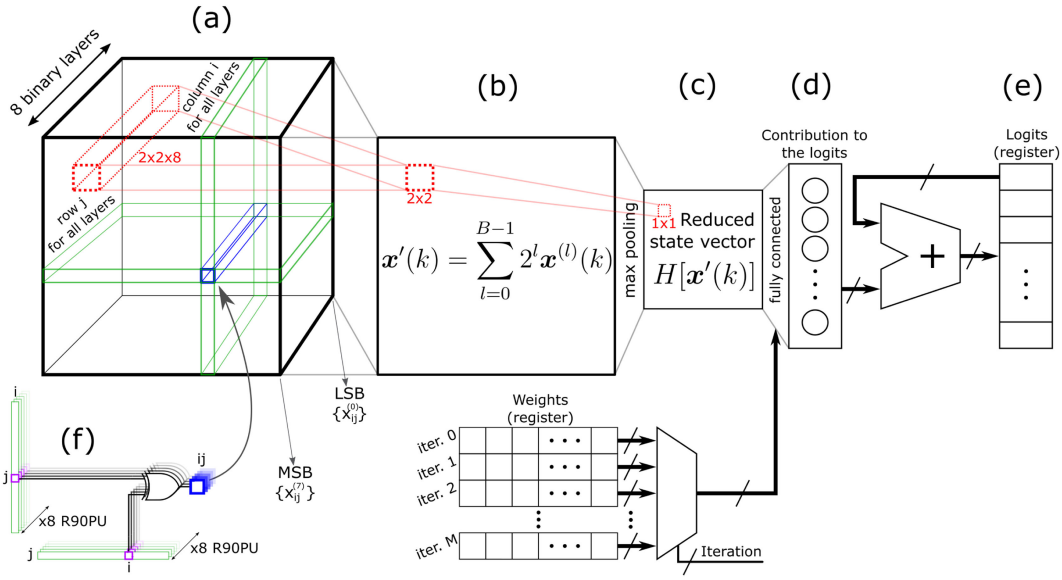


Fig. 10. A scheme of the hardware implementation of the proposed classifier.(a) 3D boolean tensor representation of 2D grayscale data, this tensor representation is organized in 2D layers from the most significative bits (MSB) to the less significative bits (LSB) of the grayscale image. (b) The 8 binary layers are interpreted as unsigned integers x_{ij}. (c) Max pooling filter applied to x'. (d) The contribution to the logits is computed using the pre-stored weights obtained offline. (e) The current contribution to the logits is accumulated until the final iteration is reached. (f) The rows and columns are iterated independently using rule 90 and combined to obtain an updated 3D boolean tensor using a bitwise XOR operation. This process is repeated until the final iteration is reached. Once the contibution of the final iteration is accumulated, the logits are valid data.

the most simplest ways to be implemented since only one XOR gate per cell is needed to implement this rule.

Going back to the high-level scheme, in Figs. 10a and 10f two R90PU layers are highlighted in green color, which correspond to iterations by rows (12) and columns (13) to generate $\boldsymbol{x}_r$ and $\boldsymbol{x}_c$ respectively. These independent row and column iterators produce the reservoir states depicted in green that are further combined into a single 3D boolean tensor $\boldsymbol{x}$ (14) whose components are depicted in blue in Figs. 10a and 10f. These boolean values are used to generate the integer state signal $\boldsymbol{x}'$ ((11)) that may be interpreted as greyscale pixel representations of the underlaying reservoir states (Fig. 10b). Therefore, in the case of the MNIST there are $2 \times 8 \times 28 = 448$ R90PUs with 28 registers ($R = 28$) and 26 XOR each one because boundary conditions are fixed and padding is not considered. So that $448 \times 26 = 11648$ XOR gates are required to implement all the R90PUs. Considering also the XOR gates of intersections, there is a total of $11648 + 28 \times 28 \times 8 = 17920$ XOR gates. In general, the number of required XOR gates for any grayscale image size would be $B(h(w-2) + w(h-2) + hw)$.

For each 2×2 pixels evaluated (Fig. 10b in red), the one with the highest value is selected (Fig. 10c). This reduced state vector obtained after the max pooling layer is multiplied by its specific weight matrix depending on the specific iteration (provided by the weights register). Finally, all these computations are accumulated sequentially in the output composed by $Q = 10$ registers to store the updated value of the logits (Fig. 10e). Notice that this process is initialized with the logits register (Fig. 10) at 0 and $\boldsymbol{x} = \boldsymbol{u}$, so that the first contribution to the logits is due to the input image $\boldsymbol{u}$ as depicted in Fig. 4. Then, the process is repeated M additional times, accumulating the contribution to the output registers, so that the logits are valid data when the contribution of the last iteration is accumulated.

Additionally, the present implementation is compared to previous publications implementing popular models of CNNs in terms of accuracy, latency, maximum performance, power, power-delay product, FPGA logic and DSP block utilization. Those quantities are specified in Table 2. In this Table *Accuracy* is the MNIST accuracy measured with the test set and reported with and without DA, *Clock frequency* may change depending on the development board, *Latency* is referred to as the classification time for one image, *Maximum performance* refers to the maximum number of thousands of inferences per second (KIPS) divided by the clock frequency, *power efficiency* refers to the number of thousands of inferences (KI) obtained per Joule invested, *Power-delay product* is

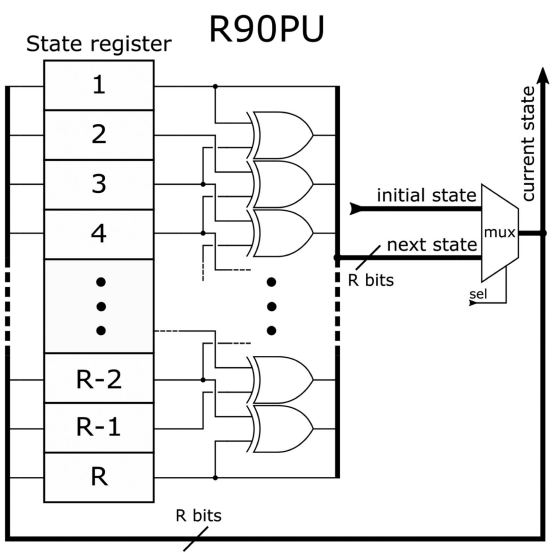


Fig. 11. Digital scheme of the rule 90 processing unit (R90PU). The present state is indexed from 1 to R ($R = 28$ in the case of the MNIST digits) and rule 90 is implemented by computing the XOR of the nearest neighbours except for the first and last elements (for which we consider fixed boundary conditions).

TABLE 2
FPGA Implementation of the Proposed Model Using 8-Bits Precission Weights Compared to Some Previous Works

Model	Accuracy (%)		Clock freq.	Latency	Maximum performance	Power	Power efficiency	Power-delay product	Logic	DSP
	w/o DA	w/ DA	(MHz)	(ms)	(KIPS/MHz)	(W)	(KI/J)	(mJ)	utilization	blocks
Ref. [42]	99.52	-	100	4-6 (approx.)	0.002	-	-	-	36.4K LUT + 41.1K FF	8
Ref. [43]	98.62	-	100	26.37	0.00038	-	-	-	14.8K LUT + 54.1K FF	20
Ref. [44]	98.32	-	150	0.0034	1.96	26.2	11.22	0.0891	182.3K ALM	20
This work	97.10	98.08	50	0.020	1.00	0.289	173	0.00578	22.6K ALM	40
Ref. [45]	96.80	-	150	0.0254	0.262	-	-	-	51.1K LUT + 66.3K FF	638
Ref. [46]	96.33	-	100	0.924	0.011	-	-	-	16.1K LUT + 6K FF	12

the energy consumption per inference, *Logic utilization* refers to the (vendor-specific) FPGA resources used in the final placement and *DSP blocks* is related to the number of multipliers synthesized in parallel. Note that the latency is approximately 10^3 times the clock period because the dot product implementation in the output layer is restricted to use 40 DSP blocks, i.e., 40 multiplications in parallel. From this table we can appreciate that our model is faster than most conventional CNN implementations and can provide good accuracy while maintaining a relatively low power consumption and resource utilization.

4 Conclusion

In this paper, we use a low-cost CA-based circuitry for pattern recognition. The proposed ReCA circuitry is implemented and validated using the MNIST dataset obtaining a competitive accuracy, low latency and low power consumption compared to previous works. From Table 2, the proposed architecture is comparable in processing speed to a high-end CNN FPGA design but consuming more than 15 times less energy and less FPGA resources. Nevertheless, there is a tradeoff between classification speed (number of CA iterations) and accuracy, which may increase the latency when dealing with more complex data sets.

Although the obtained accuracy is still not as good as that from state-of-the-art models, our classifier based on ECA reservoir, max pooling and softmax regression represents a proof of concept for the applicability of ReCA systems for image classification. Moreover, its hardware implementation represents a fast and efficient alternative compared with previously published hardware implementations of CNN image recognition systems. Its higher power efficiency (15 times better than CNN implementations) enable the technology to be applied to provide AI characteristics to those applications operating at the edge.

Further work in this line might consist of a similar hardware implementation using different image data sets (e.g., CIFAR-10 [47]) or time dependent data (e.g., ECG or EEG), considering a larger CA neighborhood, performing weight binarization [48], [49] to further reduce the number of logic elements and DSP blocks and/or adding fully connected hidden layers before the softmax layer.

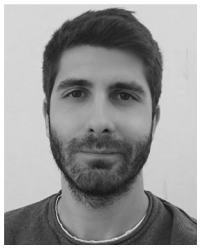
Acknowledgments

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO), and the Regional European Development Funds (FEDER), under Grant contract TEC2017-84877-R.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[2] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, Nov. 2002.
[3] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Review*, vol. 3, no. 3, pp. 127–149, 2009.
[4] E. A. Antonelo and B. Schrauwen, "On learning navigation behaviors for small mobile robots with reservoir computing architectures," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 4, pp. 763–780, Apr. 2015.
[5] A. Jalalvand, G. V. Wallendael, and R. V. D. Walle, "Real-time reservoir computing network-based systems for detection tasks on visual contents," in *Proc. 7th Int. Conf. Comput. Intell. Commun. Syst. Netw.*, 2015, pp. 146–151.
[6] D. Bacciu, P. Barsocchi, S. Chessa, C. Gallicchio, and A. Micheli, "An experimental characterization of reservoir computing in ambient assisted living applications," *Neural Comput. Appl.*, vol. 24, no. 6, pp. 1451–1464, 2014.
[7] X. Lin, Z. Yang, and Y. Song, "Short-term stock price prediction based on echo state networks," *Expert Syst. Appl.*, vol. 36, pp. 7313–7317, 2009.
[8] P. Buteneers et al., "Real-time detection of epileptic seizures in animal models using reservoir computing," *Epilepsy Res.*, vol. 103, no. 2–3, pp. 124–134, 2013.
[9] S. Wolfram, *A New Kind of Science*. Champaign, IL, USA: Wolfram Media, 2002, vol. 5.
[10] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image analysis using mathematical morphology," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-9, no. 4, pp. 532–550, Jul. 1987.
[11] W. Ge, R. T. Collins, and R. B. Ruback, "Vision-based analysis of small groups in pedestrian crowds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 5, pp. 1003–1016, May 2012.
[12] C. R. Dyer and A. Rosenfeld, "Parallel image processing by memory-augmented cellular automata," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-3, no. 1, pp. 29–41, Jan. 1981.
[13] T. Wang, A. Borji, L. Zhang, P. Zhang, and H. Lu, "A stagewise refinement model for detecting salient objects in images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4019–4028.
[14] N. Ganguly, P. Maji, S. Dhar, B. K. Sikdar, and P. P. Chaudhuri, "Evolving cellular automata as pattern classifier," in *Proc. Int. Conf. Cellular Automata*, 2002, pp. 56–68.
[15] P. G. Tzionas, P. G. Tsalides, and A. Thanailakis, "A new, cellular automaton-based, nearest neighbor pattern classifier and its VLSI implementation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 3, pp. 343–353, Sep. 1994.
[16] M. Chady and R. Poli, "Evolution of cellular-automaton-based associative memories," Cognitive Science Research Papers, Univ. Birminham, Birmingham, U.K., 1997.
[17] P. Maji, C. Shaw, N. Ganguly, B. K. Sikdar, and P. P. Chaudhuri," Theory and application of cellular automata for pattern classification," *Fundamenta Informaticae*, vol. 58, no. 3–4, pp. 321–354, 2003.
[18] O. Yilmaz,"Machine learning using cellular automata based feature expansion and reservoir computing," *J. Cellular Automata*, vol. 10, no. 5–6, pp. 435–472, 2015.

[19] O. Yilmaz, "Symbolic computation using cellular automata-based hyperdimensional computing," *Neural Comput.*, vol. 27, no. 12, pp. 2661–2692, 2015.
[20] O. Yilmaz, "Reservoir computing using cellular automata," *CoRR*, vol. abs/1410.0162, 2014. [Online]. Available: http://arxiv.org/abs/1410.0162
[21] O. Yilmaz, "Connectionist-symbolic machine intelligence using cellular automata based reservoir-hyperdimensional computing," *CoRR*, vol. abs/1503.00851, 2015. [Online]. Available: http://arxiv.org/abs/1503.00851
[22] S. Nichele and A. Molund, "Deep learning with cellular automaton-based reservoir computing," *Complex Syst. Publications*, vol. 26, no. 4, pp. 319–339, 2017.
[23] S. Nichele and M. S. Gundersen,"Reservoir computing using nonuniform binary cellular automata," *Complex Syst. Publications*, vol. 26, no. 3, pp. 225–245, 2017.
[24] N. McDonald, "Reservoir computing extreme learning machines using pairs of cellular automata rules," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 2429–2436.
[25] 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
[26] M. Halbach and R. Hoffmann, "Implementing cellular automata in FPGA logic," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, 2004, pp. 258–262.
[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[28] Y. Liu, L. Nie, L. Han, L. Zhang, and D. S. Rosenblum, "Action2Activity: Recognizing complex activities from sensor data," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 1617–1623.
[29] Y. Liu, L. Zhang, L. Nie, Y. Yan, and D. S. Rosenblum, "Fortune Teller: Predicting Your Career Path," in *Proc. 30th Conf. Artif. Intell.*, 2016, pp. 201–207.
[30] Y. Liu, L. Nie, L. Liu, and D. S. Rosenblum, "From action to activity: Sensor-based activity recognition," *Neurocomputing*, vol. 181, pp. 108–115, 2016.
[31] S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. Cambridge, U.K.: Cambridge Univ. Press, 2018.
[32] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*. Hoboken, NJ, USA: Wiley Sons, 2013, vol. 398.
[33] C. Kwak and A. Clayton-Matthews, "Multinomial logistic regression," *Nursing Res.*, vol. 51, no. 6, pp. 404–410, 2002.
[34] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*. Philadelphia, PA, USA: Siam, 1997, vol. 50.
[35] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, no. 1, pp. 503–528, 1989.
[36] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980* [cs.LG].
[38] N. M. Nasrabadi, "Pattern recognition and machine learning," *J. Electron. Imaging*, vol. 16, no. 4, 2007, Art. no. 049901.
[39] P. Y. Simard et al., "Best practices for convolutional neural networks applied to visual document analysis," in *Proc. 7th Int. Conf. Document Anal. Recognit.*, 2003, pp. 958–962.
[40] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Design Implementation*, 2016, pp. 265–283.
[41] Intel Corporation, "PowerPlay early power estimator user guide," UG-01070, Feb. 21, 2017.
[42] J.-H. Lin et al., "Binarized convolutional neural networks with separable filters for efficient hardware acceleration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jul. 2017.
[43] S. Ghaffari and S. Sharifian, "FPGA-based convolutional neural network accelerator design using high level synthesize," in *Proc. Int. Conf. Signal Process. Intell. Syst.*, 2016, pp. 1–6.
[44] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.
[45] Y. Zhou and J. Jiang, "An FPGA-based accelerator implementation for deep convolutional neural networks," in *Proc. 4th Int. Conf. Comput. Sci. Netw. Technol.*, 2015, pp. 829–832.
[46] T.-H. Tsai, Y.-C. Ho, and M.-H. Sheu, "Implementation of FPGA-based accelerator for deep neural networks," in *Proc. IEEE 22nd Int. Symp. Design Diagnostics Electron. Circuits Syst.*, 2019, pp. 1–4.
[47] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Rep. no. 4, vol. 1, no. 4, p. 7, 2009.
[48] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.
[49] M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830.



Alejandro Morán (S'19) received the BSc degree in physics from the University of the Balearic Islands, in 2016, and the MSc degree in physics of complex systems from the Institute for Cross-Disciplinary Physics and Complex Systems, University of the Balearic Islands, in 2017. He is currently working toward the PhD degree with the Electronic Engineering Group, Department of Physics, University of the Balearic Islands. His research interests include machine learning based on unconventional computing techniques, neuromorphic hardware, embedded systems and FPGA. He is a student member of the IEEE.

Christiam F. Frasser received the degree in electronics engineering from the University of Libertadores, Bogotá, Colombia, and the MS degree in electronics systems for smart environments from the University of Málaga, in 2010 and 2017 respectively. He is currently a predoctoral student at the University of the Balearic Islands in the Physics Department. His current research interests include machine learning implementations in embedded devices.

Miquel Roca (M'00) received the BSc degree in physics from the University of the Balearic Islands, in 1990, and the PhD degree from the University of the Balearic Islands, in 1994. After a research period at Electronic Engineering Department, Polytechnic University of Catalonia and a research stage at the Department of Electrical Engineering and Computer Science at INSA (Toulouse), he obtained a post of associate professor with the University of the Balearic Islands where currently he is a full professor with the Electronic Engineering research group. He has been working in Microelectronic design and Test, radiation dosimeters design. His research interests include neural networks based systems, neuromorphic hardware based on FPGA, and neural networks applications.

Josep L. Rosselló (M'00) received the PhD degree in physics from the University of the Balearic Islands (UIB), Palma, Spain, in 2002. He has been an associated professor of Electronic Technology with the Department of Physics, UIB, since 2002. He has authored dozens of research papers and patents in different fields with hundreds of citations and is currently the principal investigator of the Electronic Engeneering Group in the Physics Department at UIB. He is serving as AI consultant and developer for different technological companies and is part of the Organizing Committees of several conferences such as the Power and Timing Modeling Optimization and Simulation Conference and the International Joint Conference on Neural Networks. His current research interests include neuromorphic hardware, edge computing, stochastic computing and high-performace data mining.