Echo State Networks-based Reservoir Computing for MNIST Handwritten Digits Recognition

Nils Schaetti*, Michel Salomon†, and Raphaël Couturier‡
FEMTO-ST Institute, UMR 6174 CNRS - University Bourgogne Franche-Comté, Belfort, France
DISC (department of computer science and complex systems) - AND team
Email: *n.schaetti@gmail.com, †michel.salomon@univ-fcomte.fr, ‡raphael.couturier@univ-fcomte.fr

Abstract—**Reservoir Computing is an attractive paradigm of recurrent neural network architecture, due to the ease of training and existing neuromorphic implementations. Successively applied on speech recognition and time series forecasting, few works have so far studied the behavior of such networks on computer vision tasks. Therefore we decided to investigate the ability of Echo State Networks to classify the digits of the MNIST database. We show that even if ESNs are not able to outperform state-of-the-art convolutional networks, they allow low error thanks to a suitable preprocessing of images. The best performance is obtained with a large reservoir of 4,000 neurons, but committees of smaller reservoirs are also appealing and might be further investigated.**

I. Introduction

For the past decade a branch of machine learning called deep learning [1], [2] has gained increasing popularity, becoming a breakthrough technology thanks to its ability to solve very efficiently various tasks such as computer vision. A deep neural network learns the targeted task by automatically extracting the most suitable representation of the input data [3]. It consists of multiple layers of processing units, or neurons, with connections following a feedforward or a recurrent model. A key reason of the current success of deep learning approaches is the increasing availability of more and more powerful computing architectures, especially general purpose graphical processing units (GPU) [4], which allow to scale-up and train such networks.

Various neural network architectures belong to the deep learning family, like Deep Belief Networks [5], Convolutional Neural Networks (ConvNet or CNN) [6], Stacked autoencoders [7], etc., and somehow the less known Reservoir Computing [8], [9] approach with the emergence of deep Reservoir Computing Networks (RCNs) obtained by chaining several reservoirs [10]. Convolutional Neural Networks are known to be very fruitful for solving classification or recognition tasks on datasets of images or videos. They are the state-of-the-art for many standard datasets such as the MNIST digits dataset [11], or the CIFAR-10 and CIFAR-100 datasets [12].

When dealing with tasks such as speech recognition or time series forecasting (like financial market time ones), input data are temporal series, in that case recurrent neural networks are usually the most suited to model temporal correlations. However, compared to feedforward networks, recurrent ones are known to be difficult to train. For example, BackPropagation Through Time (BPTT) unfolds a recurrent network in time into a feedforward multilayer network trained using the backpropagation process. Therefore, Reservoir Computing is appealing because the recurrent part of a RCN is not trained.

From the machine learning point of view, Reservoir Computing refers to Echo State Networks (ESN) [13], [14], while Liquid State Machines (LSM) [15] are the counterparts in computational neuroscience. The background idea in RCNs is to feed a large and sparsely connected recurrent network, called the reservoir, with input data to produce nonlinear signals among the neurons, which are then combined linearly together in the output layer to obtain, after training, the expected outputs. The training of a RCN is thus not only easier, since it is focused on the output layer, but also because it results in solving a system of linear equations.

Reservoir Computing is interesting from the neuromorphic computing point of view as well, since different physical implementations have been already proposed, in particular optoelectronic [16] and fully-optical ones [17]. The design of such neuromorphic processors, which allow an energy-efficient and high-speed processing of input data, has been investigated more recently for convolutional networks by IBM with the TrueNorth chip [18]. This one embeds 4,096 cores, each of them simulating 256 individually programmable neurons, consumes about 70 milliwatts, and is claimed to be the first step towards a brain-inspired computer.

As noticed previously, Reservoir Computing is able to deal with real-time problems and neuromorphic implementations are available. Therefore why not use this kind of networks to fulfill computer vision tasks? In this paper, as a first step, we study the use of ESNs for the classification of images representing handwritten digits. Among the questions which have to be answered we can notice the temporal encoding of an image or the need of preprocessing the images to exhibit relevant classification results.

The paper is organized as follows. In the next section we briefly discuss some related works on the use of Reservoir Computing for solving the MNIST problem. Section III presents the Echo State Networks, how they are trained and setup to perform Reservoir Computing. Section IV describes how we propose to apply an ESN on the MNIST problem and gives the obtained results. We discuss our proposal in Section V and compare it to other approaches, in particular state-of-the-art convolutional neural networks. The final section summarizes our contributions and gives some suggestions for future works.

II. Related Works

Several research works have already investigated the application of the reservoir computing paradigm on MNIST.

In [10], Jalalvand *et al.* studied how RCNs can be used for real-time event detection in low quality videos using the raw image pixels. In order to show that such networks are relevant alternatives to state-of-the-art methods in computer vision object recognition, they also looked the performance of RCNs on the task of handwritten digits recognition. More precisely, they applied to the MNIST image classification problem a deep RCN consisting of three stacked reservoirs, each of them composed of 16K neurons. For the inputs, since RCNs are dynamical systems, an image (28×28 pixels) must be fed as a stream of inputs through time. Therefore, by considering that 5 successive columns define the inputs for a time step, the input vector has a size of $5 \times 28 = 140$ pixels. Finally, an output vector of 11 scores, one for each possible digit value and one for the blank space present in the image, is obtained by accumulating the deep RCN outputs across time. As each reservoir produces such an output vector, with the outputs of a reservoir which are the inputs of the next one in the stack, there are 528K trainable parameters in this deep RCN. Using this 3-layer architecture, the authors were able to reach a digit error rate of 0.92% on the MNIST problem.

In [19], another work related to the PhD thesis of Jalalvand, the noise robustness of RCNs for recognition tasks is addressed. For handwriting recognition, several deep RCN architectures are evaluated, with different ways of combining horizontal and vertical image scanning. Horizontal scanning means that the image is processed column-wise like in the previous work, while in the vertical case it is row-wise. The best result was gained with two 2 layer systems of 16K reservoirs in each layer, one for each scanning direction, followed by a RCN of 16K neurons which takes as inputs the outputs of both 2 layer systems. This deep RCN of 5 reservoirs which has 880K trainable parameters, since each reservoir produces an output vector of 11 components likewise the reservoirs of the above chained deep RCN, allows to slightly improve performances by achieving an error rate of 0.81%.

Hermans *et al.* have also considered the MNIST problem as case study in [20], to assess the relevance of optimizing the inputs of photonic delay systems implementing the reservoir computing paradigm. As physical device they used a delayed feedback electro-optical system exhibiting Ikeda-type dynamics as reservoir, driven by an input time series encoded into a continuous time signal. The output of the device is then recorded and transformed into a high-dimensional feature set, in order to be processed with linear regression like in other RC approaches. In this context, the authors showed that an optimization of the input encoding using BPTT provides a significant improvement in the performance in comparison with random input encodings. In fact, they trained both input and output encodings in simulations with gradient descent using backpropagation, and then applied them successfully to the real physical device, obtaining an error rate of 1.16%.

III. Implementation of Reservoir Computing with Echo State Networks

A. Echo State Networks

The main type of reservoir network used in this paper is the *Echo State Network* (ESN), introduced by Herbert Jaeger in 2011 [13], which derives from a non-linear expansion vector model where the state vector is defined by:

$$x_t = g(\overset{in}{W} u_t + W x_{t-1}), \qquad t = 1, ..., T \tag{1}$$

where $x_t \in \mathbb{R}^{N_x}$, with N_x the number of neurons in the reservoir, is the activation vector (or state) of the reservoir's neurons at time *t*. Then, the matrix $\overset{in}{W} \in \mathbb{R}^{N_x \times N_u}$, with N_u as the dimension of the input signal, represents the weights applied to the network's inputs. Finally, $W \in \mathbb{R}^{N_x \times N_x}$ is the weight matrix of the connections between the neurons inside the reservoir. Usually, the reservoir is initialized with a null state $x_0 = 0$.

The readout (or output vector) $\hat{y}_t$ is a linear combination of the neuron's activation x_t:

$$\hat{y}_t = g(\overset{out}{W} x_t), \tag{2}$$

where $\overset{out}{W} \in \mathbb{R}^{N_y \times N_x}$, with N_y the number of readouts, is the weight matrix of the connections between the reservoir's neurons and the readouts.

This model can be extended to include bias weights and feedback between the output signal and the reservoir, as follows:

$$x_t = g(\overset{in}{W} u_t + W x_{t-1} + \overset{ofb}{W} y_{t-1} + \overset{bias}{W}), \qquad t = 1, ..., T \tag{3}$$

where $\overset{ofb}{W} \in \mathbb{R}^{N_x \times N_y}$ is the weight matrix of the connections between the outputs and the reservoir's neurons. Feedback connections are used in a mode called *free-run* where the ESN is used to generate an output signal. The matrix $\overset{bias}{W} \in \mathbb{R}^{N_x}$ represents the biases of the neurons from the reservoir.

Regarding the output, direct connections can be added between the inputs and the outputs:

$$\hat{y}_t = g(\overset{out}{W} [x_t; u_t]). \tag{4}$$

The output matrix is then $\overset{out}{W} \in \mathbb{R}^{N_y \times (N_x + N_u)}$ and the operation $[\cdot;\cdot]$ is defined as the vertical concatenation of two vectors. Figure 1 gives a good overview of the architecture of the complete ESN.

A common issue in Reservoir Computing paradigm is that the dynamic of the reservoir is not adapted to the dynamic of the input signal. The idea behind *Li-ESNs* is then to adapt the network to the temporal characteristics of target signal y.

Formally, the dynamic of a *Li-ESN* in continuous time is defined in [14] by:

$$\dot{x} = \frac{1}{c}(-ax + f(\overset{in}{W} u + Wx + \overset{ofb}{W} y)) \tag{5}$$

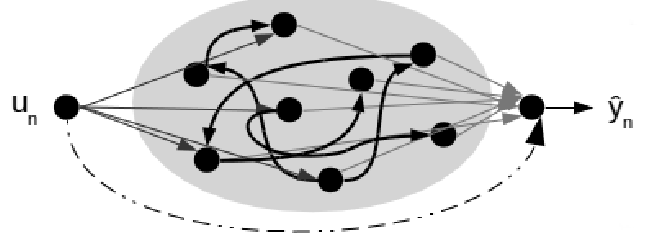

Fig. 1. ESN architecture: the inputs u_n are connected to a recurrent network of sigmoid units, while the output vector $\hat{y}_t$ is the linear combination of the units at time t.

where $c > 0$ is a time constant shared by all network's units, $a > 0$ is the *leaking rate* of the neurons, and f is the sigmoid function ($tanh(\cdot)$). The same dynamics discretized with a step δ and a sampled input signal $u_{t\delta}$ is given by:

$$x_{t+1} = \left(1 - \frac{a\delta}{c}\right) x_t + \frac{\delta}{c} f(\overset{in}{W} u_{(t+1)\delta}) + W x_t + \overset{ofb}{W} y_t) \quad (6)$$

and with $\delta = c$ we get the complete Li-ESN equation:

$$x_{t+1} = (1-a)x_t + f(\overset{in}{W} u_{t+1} + W x_t + \overset{ofb}{W} y_t). \quad (7)$$

The *leaking rate* a allows to modify the influence of past states and the dynamics of the reservoir will slow down as it goes towards zero.

B. Learning phases

The training phase consists in finding the weight matrix $\overset{out}{W}$ which minimizes the error $E(y, \hat{y})$ between the output $\hat{y}$ and the target signal y. Therefore, with the output weights matrix $\overset{out}{W} \in \mathbb{R}^{N_y \times N_x}$, $X \in \mathbb{R}^{N_x \times T}$ the matrix containing all reservoir states during training phase, and $Y \in \mathbb{R}^{N_y \times T}$ the matrix containing the corresponding outputs (with $N_x < T$), the training is defined by the following equation:

$$\overset{out}{W} X = Y, \quad (8)$$

which can be rewritten in normal form:

$$\overset{out}{W} XX^T = YX^T. \quad (9)$$

To solve this equation, a first solution is to compute the inverse matrix in order to find $\overset{out}{W}$ as follows:

$$\overset{out}{W} = (YX^T)(XX^T)^{-1}. \quad (10)$$

To get a better numerical stability, it is possible to rather use the Moore-Penrose pseudo-inverse:

$$\overset{out}{W} = YX^T(XX^T)^{+}. \quad (11)$$

However, the most common way to compute $\overset{out}{W}$ is to use *Ridge Regression* (or Thikonov regularization) [21]. By adding an additional cost to the least square optimization, *Ridge Regression* minimizes the amplitude of the weights of the $\overset{out}{W}$ matrix, mitigating the sensibility to noise and overfitting. The training equation (10) becomes then:

$$\overset{out}{W} = YX^T(XX^T + \lambda I)^{-1} \quad (12)$$

where I is the identity matrix ($I \in \mathbb{R}^{N_x \times N_x}$) and λ is the regularization factor. The regularization parameter λ has no absolute meaning and should be optimized for each reservoir.

C. Reservoir Computing method and conditions

The original Reservoir Computing method introduced with ESN in [13] includes the following steps :

1) Generate random input matrix and reservoir ($\overset{in}{W}$,W);
2) Compute reservoir states for each time step using the input signal u_t and save them successively in matrix X;
3) Compute the output matrix $\overset{out}{W}$ from states collected in 2) with a linear regression method, minimizing the average quadratic error $E(y, \hat{y})$ between the output signal $\hat{y}$ and the target signal y;
4) Use the trained network on a new input signal u_t and computes the output $\hat{y}_t$ with the $\overset{out}{W}$ output matrix.

Several parameters can be taken into consideration, but it is particularly important to ensure that the reservoir has the *Echo State Property* (ESP) which establishes that the effect of past states x_t and past inputs u_t on future states x_{t+k} vanishes gradually over time, and is not amplified.

The first method introduced to establish conditions to ensure that a reservoir has this echo state property is the *spectral radius*, noted $\rho(W)$, which is the maximum absolute eigenvalue of the matrix W. In this case, the formulation of conditions to get the echo state property has been the object of a lot confusion in publications about Reservoir Computing as it is commonly asserted that the condition $\rho(W) < 1$ is sufficient and necessary to get the echo state property. Nevertheless, it has been shown that this condition only applies to autonomous systems ($u_t = 0$), and is neither a sufficient nor a necessary condition for the ESP. However, since it has been shown empirically that the likelihood of the state transitions being contractive is higher when $\rho(W) < 1$, this condition is said to be *sufficient in practice*. Then, the algorithm to generate the reservoir weight matrix W is:

1) Generate a random matrix W;
2) Divide the matrix W by the spectral radius $\rho(W)$.

Other studies and conditions have been proposed to get a good reservoir. In fact, as randomly created reservoirs have not the same performances on a task, the search for methods allowing to create optimal reservoirs for a given task has been the center of a lot of attention in the past few years. Unfortunately, no such method has been found so far. Nonetheless, some conditions and measures have been proposed, most of them based on the border of chaos and the Lyapunov exponent.

The border of chaos is a region of parameters of a dynamical system in which the network operates at the limit between ordered and chaotic behaviors. It is commonly asserted in Reservoir Computing literature that a reservoir at this border has a greater computing power and a maximum memory capacity. This point has been the object of confusion in the application of Reservoir Computing which has led to the common method to set the value of spectral radius of the reservoir connectivity matrix near 1, but just below, so that the system dynamics be near the border of chaos. More precisely, research works on LSMs showed that reservoirs operating between an ordered and a chaotic behavior have optimal computing performances. Concerning ESNs, it has been suggested that they are optimal in a stable, but near chaos, dynamic because they have richer dynamics without drowning information. However, some doubts have been put on the border of chaos. Mainly because it has been shown that ESNs have better results with small spectral radius, especially when requiring quick reservoir answers instead of an extended memory. Therefore, it is not always beneficial to set the spectral radius so that the reservoir dynamics are at the border of chaos.

D. *Committees of ESNs*

In machine learning, committees of classifier are commonly used to improve performance. In Reservoir Computing, it has been used for voice recognition and financial time series prediction, in [14] and [22] respectively. The output of a committee of N_c members is defined by the average of all ESN outputs. For example, in a classification problem, if $Pr_c(C = i)$ is the probability given by the reservoir c that the class C of the current signal is i, then the probability of the same event given by the committee satisfies:

$$Pr(C = i) = \frac{1}{N_c} \sum_{c=1}^{N_c} Pr_c(C = i). \tag{13}$$

The learning phase is done separately on each reservoir with the same input signal u and target output y.

IV. Application on the MNIST Problem

The MNIST database is commonly used to evaluate performances of machine learning methods to recognize handwritten digits. It is composed of a training set of 60,000 images, and a test set of 10,000 images, with the corresponding labels going from zero to nine. Classifiers are evaluated on their error rate and the best score on MNIST, an error rate of 0.21%, is currently held by a committee of 5 neural networks [11]. Each network consists of two convolutional layers with 32-64 feature maps in each respectively, followed by a fully connected layer of 150 rectifier linear units sparsely connected thanks to DropConnect, and a final softmax classification layer.

In this section, we first present the method used to classify images with an ESN, and then we describe the transformations applied to images to improve the performances. Third, we specify the various kinds of output layers evaluated, and finally the performances of each reservoir.

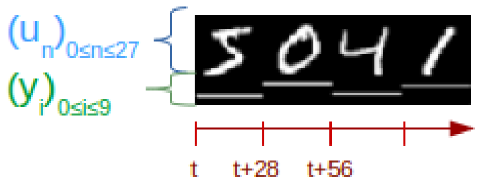

Fig. 2. Inputs and outputs of a ESN for image classification, with $N_i = 28$.

A. *From images to temporal signals*

Each digit to classify is represented in the MNIST database as a 28×28 image where each pixel has a value between 0 (white) and 255 (black). The first modification is to transform each image into a matrix $I_n \in \mathbb{R}^{N_i \times N_i}$, where n denotes the image index and $N_i = 28$ is the image size, with normalized pixel values. The corresponding label y_n is an integer between 0 and 9 according to the digit represented by I_n.

The problem is then to classify each matrix I_n in the class y_n corresponding to the digit that it represents. It is therefore a classification problem where the classifier $\hat{c} : I_n \to \hat{y}_n$, with $\hat{y}_n \in \{0...9\}$, has to establish to which class the matrix I_n belongs to. We can then evaluate the classifier performances, expressed as the percentage of matrix classified in the wrong class ($E(y, \hat{y}) \in [0, 100]$), on the training and test sets.

To give matrices to the reservoirs, we have to transform images into a temporal signal. To achieve this, we transform one spatial dimension to a temporal one, passing images column by column (or row by row depending on the selected orientation). As we consider a processing column-wise (horizonal scanning), we have $N_u = N_i$ reservoir inputs and for each input $(u_i)_t = (I_n)_{ij}$, where $t = n \times N_i + j$. Hence, the i-th reservoir input takes, at time step t, the value of the (i, j) element of the n-th matrix with both n and j defined according to t. Then, as a result we get N_i time series composed of all images put side by side. Figure 2 gives a better overview of this process, we can see that for each time step t which is a multiple of N_i a new image is introduced.
The idea behind this method is to put one dimension of the images in the memory of the reservoir so that it can create a set of features of the current image and use these features to classify the image. Therefore, the reservoir must have enough memory to store a full image.

B. *Images transformations*

To allow reservoirs to see images under different aspects and extract various features, we introduce in this section two kinds of image deformations. The resulting images are finally introduced in the reservoir.

The first type of deformation (S_x) consists in the reframing and resizing of images. The top line of Figure 4 gives three examples of this transformation. Both reframing and resizing are done on the part of the image which contains non-black pixels, removing the borders. For example, this allows a digit contained in a 22×22 square to be fitted into a 15×15 square.

The second type of transformation (R_θ) consists in the rotation of the image, keeping the size constant. In Figure 4,

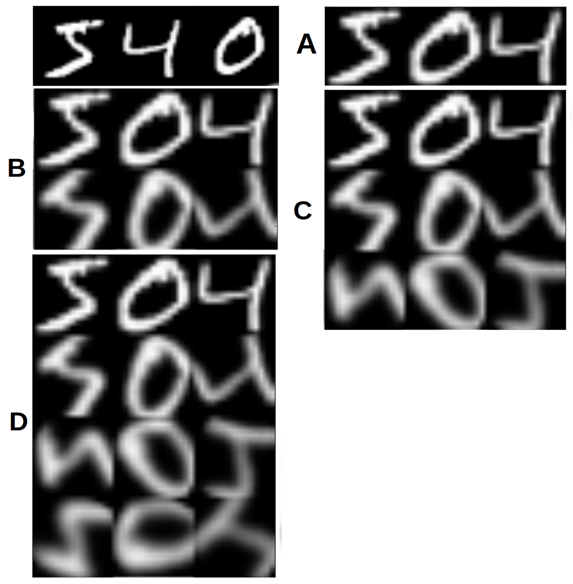

Fig. 3. Examples of image deformations and stacking. **Example B** Two deformations, S_{15} and $S_{15}+S_{30}$ stacked to give a $N_u = 30$ reservoir inputs. **Example C** Three deformations, S_{15}, $S_{15}+S_{30}$ and $S_{15}+S_{90}$, stacked give $N_u = 45$ reservoir inputs.

the middle line represents the deformation R_{30} where each image has been rotated by an angle of 30 degrees. The key idea behind this transformation is to present the digit to the reservoir through another angle and point of view to increase the number of efficient features for classification.

Finally, we can create a framework from these deformations by combining and multiplying them. The bottom line of Figure 4 shows the combination of both deformations. We can also give multiple representations to the reservoir by stacking the transformed images. The entries B, C and D in Figure 3 present examples of this idea.

C. Output layer

For classification problems, the usual output layer consists of one output per class $N_y = 10$, where each output returns the likelihood that the signal at time t belongs to the corresponding class. In addition to this classical output layer, we also evaluated two other output layers obtained by joining reservoir states. In the first case all the reservoir states are considered, whereas in the second one only some selected reservoir states are retained.

For the classical output layer, we trained 10 outputs $(\hat{y}_i)_t$ with $0 \leq i \leq 9$, one per digit, to copy a target 10-dimensional signal y_i where $(y_i)_t = 1$ if the current image at time t belongs to the class i, $(y_i)_t = 0$ otherwise. Therefore we get the probability that image n belongs to a class i by averaging each output over time between the time where the first image column was introduced ($t_{begin} = N_i \times n$) and the time where the last column was fed in the ESN ($t_{end} = N_i \times (n+1)$). More formally, the likelihood that the class C_n of image I_n be c is $Pr(C_n = c) = \frac{1}{N_i} \sum_{t_{begin}}^{t_{end}} (\hat{y}_i)_t$.

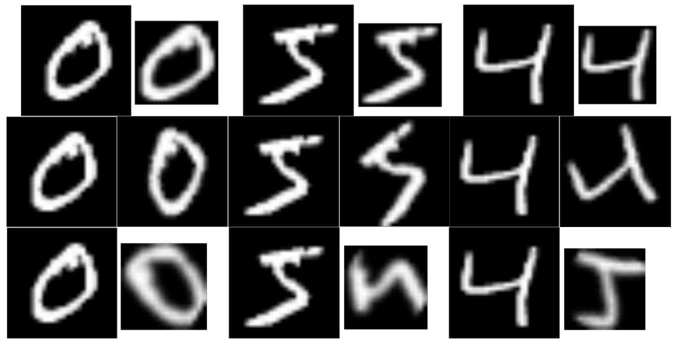

Fig. 4. **Top.** S_{20} deformation examples : image reframing and resizing, from 28×28 to 20×20 images. **Middle.** R_{30} deformation examples : 30 degrees image rotation. **Bottom.** Combined deformation $(S_{18}, R_{30}, S_{18}, R_{60})$.

The second kind of output layer, named JS, which was evaluated is obtained by joining all states resulting from the feeding of the image into the reservoir. Hence, the joined state is defined as $x^* = x_{t_{begin}} \cup x_{t_{begin}+1} \cup ... \cup x_{t_{end}}$, where $t_{end} - t_{begin} = N_i$. The learning phase consists therefore in solving the equation $Y = \overset{out}{W} X^*$, where $\overset{out}{W} \in \mathbb{R}^{N_y \times (N_x \times N_i)}$ and $X^* \in \mathbb{R}^{(N_x \times N_i) \times T}$. The idea behind this kind of layer is to compute the class probability not on a single state, but on the trajectory of a dynamical system.

The joined states output layer has an important drawback as it increases enormously the complexity of the learning phase, whether in terms of temporal or spatial complexity. Consequently we introduced a new output layer, named MTS (*Mixed Three States*), which consists in joining the last state and two intermediary states between the first and the last states. More formally, this mixed state is defined by $x^{mts} = x_{t_{begin}+\lfloor \frac{1}{3} N_i \rfloor} \cup x_{t_{begin}+\lfloor \frac{2}{3} N_i \rfloor} \cup x_{t_{end}}$. The training phase is done on this mixed state and the temporal and spatial complexity is hugely reduced compared to the JS output layer.

D. Results

To apply the kind of previously defined reservoir, we first sought good parameters with a grid-search for the following parameters : input scaling, proportion of zero in both $\overset{in}{W}$ and W, bias, spectral radius and leaking rate. The best configuration corresponds to a spectral radius of 1.3, a proportion of zero in $\overset{in}{W}$ of 90% and 10% in W, a input scaling of 0.6, bias of 1.0, and a leaking rate of 0.2. It is worth noticing that the spectral radius is here above one, much higher than the famous border of chaos, and that the low leaking rate indicates that this task requires a slow dynamic.

The average performance of 30 Li-ESN is 21.233% for a reservoir of 100 neurons ($N_x = 100$), whereas it is 9.48% for a reservoir of 1,200 neurons. A 1,200 neurons ESN shows a performance slightly better than a linear regression. We must then use different image transformations to allow the ESN to take into account different aspects of the images and extract improved features.

For this purpose, we selected the four input transformations presented in Figure 3, where the digits are resized to a size of

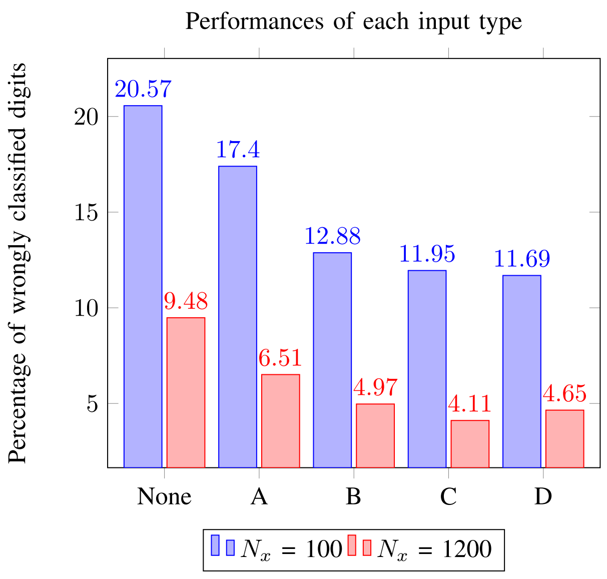

Fig. 5. Error rate of the different image deformations.

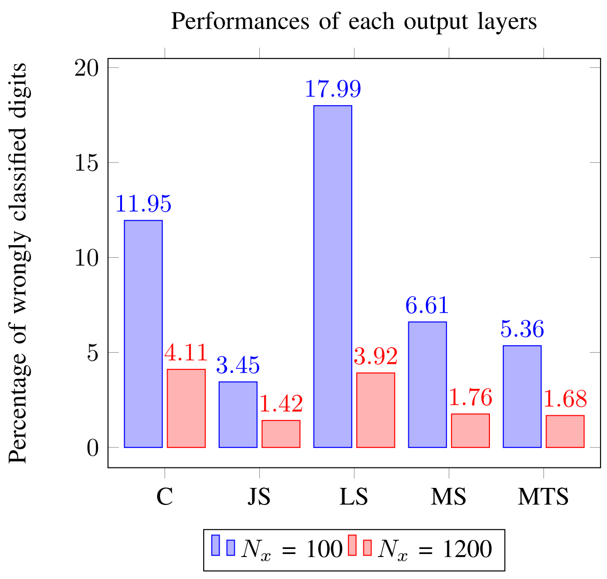

Fig. 6. Error rate of the different outputs.

15×15. As it is reasonable to make the assumption that each transformation changes the dynamics of the input signal, we therefore search for the best spectral radius and leaky rate for each kind of transformation. For the transformations A to D, the best spectral radius is respectively 1.2, 0.2, 0.3, and 0.1 (0.1 for no transformation), and the best leaky rate is 0.4 for all transformations (compared to 0.2 for no transformation). The increase of 0.2 of the leaky rate shows that resizing the digits from 28×28 to 15×15 accelerates the input dynamics.

We can then evaluate the average performances of each kind of input on 100 randomly generated reservoirs. Figure 5 shows the result of this experience. The input A reduces the error rate from 20.57% to 17.4% (-3.17%). The inputs B, C, and D reduce respectively the error rate to 12.88% (-7.69%), 11.95% (-8.62%) and 11.69% (-8.88%). For a reservoir of 1,200 neurons, the error rates for each transformation are respectively 6.51%, 4.97%, 4.11%, and 4.65%.

Two remarks can be made on these results. First, the last type of inputs, denoted by D, is less efficient than the type C, which could be explained by the fact that additional transformations of this kind do not give features and information useful for classification. Second, the error rate goes down fast as we increase the reservoir size, however, beyond 500 neurons, adding neurons has a much less significant impact on the performance.

We evaluated the average error rate of 100 randomly generated reservoirs ($N_x = 100$) for each type of output layer with the transformation C as input. In addition to JS and MTS, two other output layers were considered: the *Last State* (LS) and a concatenation of the middle and last states denoted by *Mixed States* (MS). Figure 6 shows the obtained results. We can see the good performance of the JS output, which reduces the error rate from 11.95% to 3.45% (-8.5%). The LS output increases the error rate (+6.05%), while the two other outputs, MS, and MTS, have good performances with respectively 6.61% (-5.34%) and 5.36% (-6.59%).

We introduced in Section II the commonly used committee method where N_c ESNs are used together to reduce the error rate. So we tested reservoirs of 50, 100, and 1,000 neurons, for committees of 2 to 50 members. For reservoirs of 50 neurons, the lower error rate is achieved by a committee of 10 members with 3.83% of wrongly classified digits, compared to 5.55% (-0.79%) for a single reservoir. For reservoirs of 100 neurons, the best performance is achieved by a committee of 35 members with 2.77%, compared to 3.54% (-0.77) for a single reservoir. For reservoirs of 1,000 neurons, the best performance is gained by a committee of 21 members with 1.29% (-0.27) of wrongly classified digits against 1.56% for a single reservoir. By using a majority voting rather than the average probability given by equation (13), the committee of 21 members gives a slightly improved error rate of 1.25%. Figure 7 shows typical examples of misclassified digits.

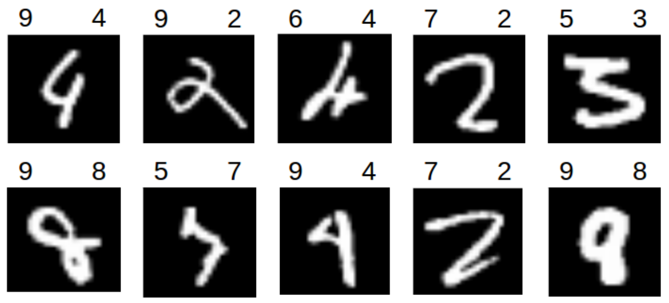

Fig. 7. Examples of wrongly classified digits by an ESN of 1,000 neurons with a joined state output layer. Digits on the top-left side are the digits found by the ESN, and the one on the top-right side are the real digits found in the MNIST database.

E. Using larger reservoirs: a case study with 4,000 neurons

In order to see the influence of an increase in the number of neurons, we have used an optimization loop in order to optimize some parameters of the reservoir with a genetic algorithm. These parameters are the number of neurons and the leaking rate a. The lower and upper bounds are respectively set to $1,000$ and $5,000$ for the number of neurons. The optimal parameters found are $4,000$ for the number of neurons and 0.4 for a. With these parameters, the error rate on the MNIST problem is equal to 0.93%. It should be noticed that these parameters also depend on the seed used to initialize the values of the reservoir, its connectivity, and the input matrix $\overset{in}{W}$.

V. Discussion

The obtained results can be used to compare the Reservoir Computing method to other machine learning approaches. To this end, we selected results achieved by several commonly used methods in the field of image classification.

On our side, we have chosen different networks of the work presented in this paper: reservoirs of 1,200 neurons with both MTS and JS output layers, a committee of 21 reservoirs of 1,000 neurons with a JS output layer, and finally a large reservoir of 4,000 neurons. Table I presents the error rate and the complexity, in terms of addition/multiplication operations to classify a single character, for each of these networks compared to a set of other methods including deep learning networks, support vector machines, and common feedforward neural networks.

Table I also shows that our approach outperforms the first deep learning network with respectively 1.68% and 1.42%, compared to 1.7% for the *LeNet-1*. The complexity is higher for the RC approach, but it is worth noticing that the sparsity of the matrices in equation (7) has no impact on the error rate as long as the number of non-zero elements does not drop below a critical value. To reduce the complexity of our method, the sparsity has been pushed from 10% to 0.3%, and from 90% to 5%, for the matrix W and $\overset{in}{W}$. The sparsity can still be increased and we are therefore confident that the complexity could be reduced to the level of the *LeNet-1*. We can observe that the different RC networks studied in this paper are able to reach good results for handwritten digits recognition.

As regards to other machine learning approaches, several remarks can be made. First, a support vector machine can reach a very good error rate of 1.1%, but with a huge complexity. Second, deep learning networks hold the top places, even if our best reservoir computing approach, the one with a reservoir of 4,000 neurons, performs slightly better than the *LeNet-4* and *LeNet-5* which have respectively, an error rate of 1.1% and 0.95%. In fact, Convolutional Neural Networks are the best suited for the MNIST problem, leading to impressive results, thanks to a committee decision. Hence, the three places on the podium are occupied by committees of 7, 35, and 5 members, yielding respective error rates of 0.27%, 0.23%, and 0.21%. This last value is the best performance so far and is nearly the human capacity ($\approx 0.2\%$).

TABLE I
Comparison of different machine learning approaches.
Reservoir computing approaches studied in this work are in bold.

Classifier	Error rate	Add/mult
LeNet-1 (deep learning)	1.7%	1.6×10^5
RC-aESN-MTS 1,200 neurons	**1.68%**	2.5×10^5
RC-aESN-JS 1,200 neurons	**1.42%**	4.5×10^5
21 × RC-aESN-JS 1,000 neurons	**1.25%**	15×10^6
SVM degree 4	1.1%	14×10^6
LeNet-4	1.1%	1.6×10^5
LeNet-5	0.95%	4.0×10^5
RC-aESN 4,000 neurons	**0.93%**	
CNN 551 neurons	0.35%	
7 × CNN 221 neurons	0.27%	
35 × CNN 221 neurons	0.23%	
5 × CNN with DropConnect	0.21%	

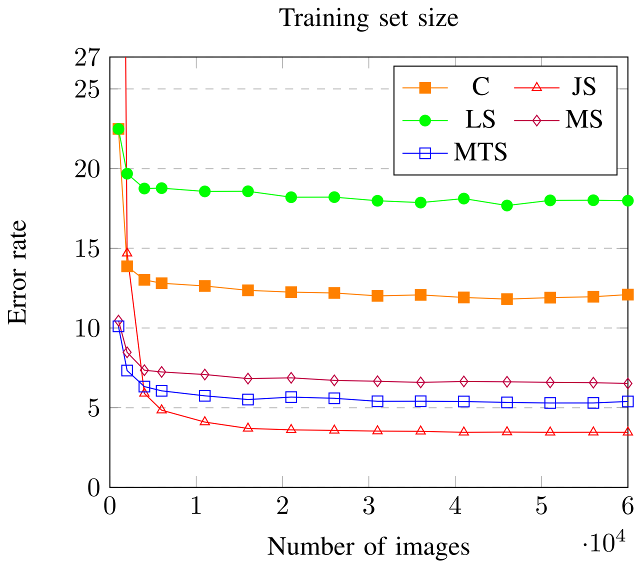

Fig. 8. Error rate of the different outputs.

However, convolutional networks have a huge learning time, stretching from a few hours to several weeks. They can be used nowadays thanks to GPU implementation. Our method on the contrary has a learning time of only about 20 minutes for a reservoir of 1,200 neurons. Moreover, better performances can be achieved by extending the learning set with affine and elastic transformations. Our method also shows an impressive capacity to learn quickly with a small training set. Figure 8 shows the error rate for the network using transformations of type C as input and each specific output layer. The error rate drops quickly as the learning set size grows and reaches values near its minimum with only one third of the set. This demonstrates a huge capacity of generalization.

VI. Conclusion and Future Works

We have introduced a new image classification approach using the Reservoir Computing paradigm and evaluated it on the MNIST. This approach shows an interesting and encouraging error rate of 1.42% for a reservoir of 1,200 neurons and 0.93% when using 4,000 neurons. It is worth noticing that fast learning networks like ESN can reach such good performances for a first application where lots of parameters can be optimized.

Moreover, the performance analysis shows that the way we presented image to the reservoir has an important impact on the error rate. Deformation and rotation allow significant drops of the number of wrongly classified digits. We think that these transformations allow the reservoir to extract a set of useful features for classification. The output layer seems also to have a significant impact on the performances, and applying the learning phase on joined state can lower the error rate of 70%.

Among other methods commonly used for image classification, our approach outperforms already the simple feedforward networks in terms of error rate and complexity, and can bear the comparison with the first deep learning networks such as the *LeNet-1*. It is worth noticing that deep learning networks are highly specialized, as they are trained with semi-supervised algorithms to extract the most significant features. In comparison, the medium used for computation in our approach, the reservoir, can be used for other tasks or to extract new information about the input signal without creating another neural network. Furthermore, *ESN* can be used as computation and memory devices, a feature mandatory to build future truly intelligent systems.

A large field of possibilities is still there to improve the performances of our networks. First, we used the old fashion method of selecting manually the set of transformations applied to the digits, unable to determine which features are the most useful for the task. Deep learning found a groundbreaking solution to this problem by creating abstraction layers where the deformations are selected by learning algorithms. In the future, we will use methods inspired by deep learning to create input layers that extract the most effective features for reservoir networks. Another track is to define and evaluate hybrid Reservoir Computing and deep learning networks: the reservoir could be clustered into abstraction layers, or a temporal dimension could be added to deep learning networks. Second, we could largely reduce the time of the learning phase by optimizing the implementation using GPU and CUDA, and we could also reduce the memory used with recursive least square algorithms. Finally, as the output layer seems to have a significant impact, extended researches are possible to determine which reservoir states should be used for the learning phase. Indeed, as each state is a highly nonlinear representation of the input signal and of its history as well, we can reasonably infer that each state is useful for classification.

We also plan to test a different committee method using three reservoirs. A first reservoir as standard classifier, a second one trained only on the digits wrongly classified by the first reservoir, and a last reservoir trained to decide which of the two reservoirs has the right answer for each digit.

References

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, Aug 2013.

[4] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014. [Online]. Available: http://arxiv.org/abs/1410.0759

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

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

[7] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[8] D. Verstraeten, B. Schrauwen, M. d'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[9] M. LukošEvičIus and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[10] A. Jalalvand, G. Van Wallendael, and R. Van de Walle, "Real-time reservoir computing network-based systems for detection tasks on visual contents," in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on.* IEEE, 2015, pp. 146–151.

[11] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

[12] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *ArXiv e-prints*, Nov. 2015.

[13] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, p. 34, 2001.

[14] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.

[15] W. Maass, "Liquid state machines: motivation, theory, and applications," *Computability in context: computation and logic in the real world*, pp. 275–296, 2010.

[16] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutierrez, L. Pesquera, C. R. Mirasso, and I. Fischer, "Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing," *Opt. Express*, vol. 20, no. 3, pp. 3241–3249, Jan 2012. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-20-3-3241

[17] D. Brunner, M. C. Soriano, C. R. Mirasso, and I. Fischer, "Parallel photonic information processing at gigabyte per second data rates using transient states," *Nature communications*, vol. 4, p. 1364, 2013.

[18] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing," *ArXiv e-prints*, Mar. 2016.

[19] A. Jalalvand, K. Demuynck, W. De Neve, R. Van de Walle, and J.-P. Martens, "Design of reservoir computing systems for noise-robust speech and handwriting recognition," in *28th Conference on Graphics, Patterns and Images (accepted in the Workshop of Theses and Dissertations (WTD)).* Sociedade Brasileira de Computação, 2015.

[20] M. Hermans, M. C. Soriano, J. Dambre, P. Bienstman, and I. Fischer, "Photonic delay systems as machine learning implementations," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 2081–2097, Jan. 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789272.2886817

[21] A. Tikhonov, *Solutions of ill-posed problems.*

[22] F. Wyffels and B. Schrauwen, "A comparative study of reservoir computing strategies for monthly time series prediction," *Neurocomputing*, vol. 73, no. 10, pp. 1958–1964, 2010.