Neural Computing and Applications
https://doi.org/10.1007/s00521-024-09656-4

ORIGINAL ARTICLE

Exploring deep echo state networks for image classification: a multi-reservoir approach

E. J. López-Ortiz[1] · M. Perea-Trigo[2] · L. M. Soria-Morillo[2] · F. Sancho-Caparrini[1] · J. J. Vegas-Olmos[3]

Received: 1 November 2023 / Accepted: 25 March 2024

Abstract
Echo state networks (ESNs) belong to the class of recurrent neural networks and have demonstrated robust performance in time series prediction tasks. In this study, we investigate the capability of different ESN architectures to capture spatial relationships in images without transforming them into temporal sequences. We begin with three pre-existing ESN-based architectures and enhance their design by incorporating multiple output layers, customising them for a classification task. Our investigation involves an examination of the behaviour of these modified networks, coupled with a comprehensive performance comparison against the baseline vanilla ESN architecture. Our experiments on the MNIST data set reveal that a network with multiple independent reservoirs working in parallel outperforms other ESN-based architectures for this task, achieving a classification accuracy of 98.43%. This improvement on the classical ESN architecture is accompanied by reduced training times. While the accuracy of ESN-based architectures lags behind that of convolutional neural network-based architectures, the significantly lower training times of ESNs with multiple reservoirs operating in parallel make them a compelling choice for learning spatial relationships in scenarios prioritising energy efficiency and rapid training. This multi-reservoir ESN architecture overcomes standard ESN limitations regarding memory requirements and training times for large networks, providing more accurate predictions than other ESN-based models. These findings contribute to a deeper understanding of the potential of ESNs as a tool for image classification.

Keywords ESN · GNN · Image classification · MNIST

1 Introduction

Deep Neural Network (DNN)-based models have become the standard for solving many real-world problems [1–5], including computer vision tasks [6, 7], due to their high accuracy. However, their high computational and energy requirements [8] make them resource-intensive to train, highlighting the importance of exploring alternative models that can provide more sustainable solutions. One such model is the echo state network (ESN) [9], a type of recurrent neural network (RNN) that has gained popularity due to its effectiveness in solving time series prediction problems [10]. The lightness of the model, as well as its fast training, makes it ideal for use on resource-constrained devices, even without GPUs, which is difficult to achieve with DNN-based models. Like traditional RNNs, ESNs have an internal feedback matrix that allows them to store information about previous states, making them well suited

✉ E. J. López-Ortiz
elortiz@us.es

M. Perea-Trigo
mptrigo@us.es

L. M. Soria-Morillo
lsoria@us.es

F. Sancho-Caparrini
fsancho@us.es

J. J. Vegas-Olmos
juanj@nvidia.com

[1] Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Avda. Reina Mercedes, SN, 41012 Sevilla, Sevilla, Spain

[2] Department of Languages and Computer Systems, Universidad de Sevilla, Avda. Reina Mercedes, SN, 41012 Sevilla, Sevilla, Spain

[3] NVIDIA Corporation, NVIDIA, Hermon Building, 20692 Yokneam, Yokneam, Israel

Published online: 18 April 2024

to provide predictions based on both current inputs and historical information.

Despite their success in predicting time series [11], their potential to effectively capture spatial relationships in data has been relatively understudied. This study aims to investigate the ability of ESNs to handle pure spatial information in image data, which has important implications for real-world applications. In particular, we seek to assess the performance of ESNs in handling pure spatial tasks using the MNIST and FashionMNIST data sets. This thorough evaluation will offer insight into ESN's capabilities and limitations in image processing, guiding future research in this field.

The main contributions of this article are the following:

1. Exploration of the vanilla ESN architecture in a domain beyond time series, namely in an image classification task. We study its behaviour for different sets of hyperparameters and test the influence of each of them on the performance of the architecture, shedding light on its adaptability to spatial tasks.
2. Implementation and refinement of three established ESN-based architectures. The modifications include the integration of multiple output layers, tailoring these architectures for best performance in an image classification task.
3. Evaluation of the aforementioned architectures in the context of image classification, providing an analysis of their performance and highlighting the advantages of the parallel reservoir architecture over other ESN-based architectures.
4. Advancement of the state of the art through the application of ESNs for image classification, showcasing a significant contribution to the field.

The remainder of this article is structured as follows: Sect. 2 presents an overview of related works on ESN. In Sect. 3, we detail our proposed ESN architectures for image processing. Section 4 presents and analyses the results of our experiments. Finally, Sect. 5 summarises our findings and provides insights for future work.

2 Previous works

Echo state networks were initially introduced in the early 2000 s [9] as a type of recurrent neural network. The inception of ESNs was motivated by the need to address the challenges associated with training RNNs, whose recurrent connections present complexities when attempting to apply backpropagation to large sequences.

The key feature of ESN is the use of a randomly generated sparse connectivity matrix in the recurrent layer and a constant reaction of the network state to the input and its previous states, allowing the network to capture complex temporal dependencies while reducing the number of trainable parameters in the network. The training of an ESN can be summarised in four phases: initialisation, reservoir computation, weight adjustment, and testing.

2.1 ESN architecture

2.1.1 Initialisation

The randomness with which both the input layer and the reservoir are generated, generally using a normal or uniform distribution, is one of the factors influencing rapid training of ESNs. However, initial randomness can sometimes lead to suboptimal results, so careful selection of some parameters is necessary to achieve the desired results. Factors such as the number of neurons, the density of the connection, and the spectral radius of the connection matrix all play a crucial role in determining the reservoir behaviour, which subsequently affects the overall performance of the network. Next, we will briefly discuss them to understand their impact (see [12] for an extended explanation of each hyperparameter):

- *Number of Nodes:* Having too many or too few nodes can lead to poor results. The number of nodes determines the size and complexity of the reservoir.
- *Density:* The ratio of connections between reservoir nodes, also known as the connection density, can greatly influence the performance of the network.
- *Spectral Radius:* Usually denoted by $\boldsymbol{\rho}$. The spectral radius of the connection matrix helps to avoid the predictions of the network tend to infinity or zero when the network works in recurrent mode.
- *Input Scaling:* Ensuring that the input data falls within a suitable range for the network to process is crucial for accurate predictions. It is usually referred to as $\boldsymbol{\gamma}$.
- *Leaking Rate:* The extent to which information from the previous time-step is retained in the current time-step plays an important role in the network's performance. It is usually called $\boldsymbol{\alpha}$.
- *Regularisation coefficient:* This hyperparameter helps to prevent overfitting during the computation of the weights of the output layer. Regularisation is especially important when working with small data sets. Usually denoted by $\boldsymbol{\beta}$.

Optimising the hyperparameters of echo state networks remains a challenging task. Various methods, such as genetic algorithms (GA) [13], particle swarm optimisation (PSO) [14], as well as gradient-based [15] or grid search approaches, have been used to fine-tune the hyperparameters and improve network performance.

In this work, a grid search was used to observe the response of the architecture to the different hyperparameters.

2.1.2 Reservoir computation

During training, input sequences are fed into the network and the activations of the reservoir neurons are computed. These activations and inputs are then collected and stored in a matrix, which is a historical record of the dynamic behaviour of the network (as shown in Fig. 1). This matrix is referred to as H for convenience. Upon completion of the training phase, these data determine the weights of the output layer. As a result, the performance of the ESN model depends on the information contained in H, making it a crucial element of the training phase.

The most commonly used approach to update states in ESNs is the leaking-integration variation, introduced by [16]. This approach updates the state of the network at each time step using:

$$x_t = (1-\alpha)x_{t-1} + \alpha\lambda(W_{\text{in}} \cdot u_t + W \cdot x_{t-1}) \quad (1)$$

In the given equation, the function λ is typically chosen as a sigmoid function, with the hyperbolic tangent function being a commonly used example. If we let K represent the size of the input and N denote the number of nodes in the reservoir, the vector $u_t \in \mathbb{R}^K$ denotes the input at time t, and $W_{\text{in}} \in \mathbb{R}^{N\times K}$ represents the weights of the input layer. The matrix $W \in \mathbb{R}^{N\times N}$ is the adjacency matrix that describes the connections between neurons (this structure is commonly referred to as the reservoir) and $x_{t-1} \in \mathbb{R}^N$ represents the state of the neurons in the previous time step. The parameter $\alpha \in [0, 1]$ controls the network sensitivity to new inputs. A low value of α results in a less responsive network to new inputs, tending to retain more information about its previous state. On the other hand, a high value of α leads to a network that responds more strongly to new inputs. Since the calculation of new states is partly based on the previous state of the network, each new state contains an "echo" of the previous one. This is why these networks are called echo state networks.

2.1.3 Finding the weights of the output Layer

In an ESN, the input layer and the reservoir work together to generate a singular state that is highly dependent on both the current input and the previous state of the reservoir. During the training phase, these states are stored in the matrix H, which is then used to calculate the weights of the output layer. The task of the output layer is to decipher the network response to inputs and to provide an accurate output prediction. This is achieved by solving an ordinary differential equation (ODE) system that relates the network response to input data (stored in H) to the desired output (Fig. 2).

The accuracy of the output layer is critical for the performance of the ESN, as it directly impacts the network's ability to generate accurate predictions. A linear regression algorithm is often used to train the output layer, which is less computationally expensive than backpropagation. The Tikhonov regularisation method (expressed in Eq. 2) is frequently used for stability:

$$W_{\text{out}} = (H^\intercal H + \beta I)^{-1} H^\intercal Y \quad (2)$$

where W_{out} represents the output layer and $H^\intercal$ is the transpose of H, the regularisation coefficient, β, is typically a small value between 10^{-4} and 10^{-10}, and the target vector is denoted by Y.

2.1.4 Testing

During the test phase, the ESN generates predictions for novel input sequences. Each new input produces a reaction

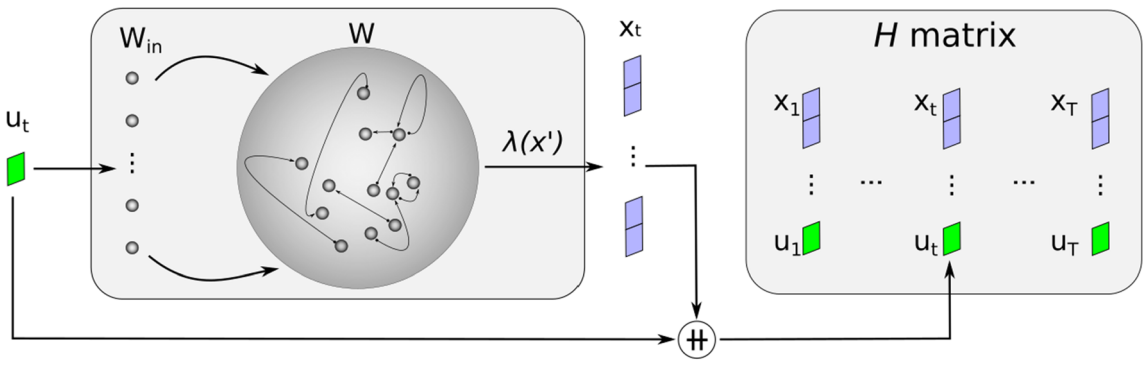

Fig. 1 ESN. Basic architecture. Reservoir computing phase. The symbol $+\!\!+$ is used to express a vector concatenation, while λ is used to express the computation of the new state. At each time t, the new input causes a reaction in the state of the neurons, and this reaction, together with the input that caused it, is stored in the H matrix

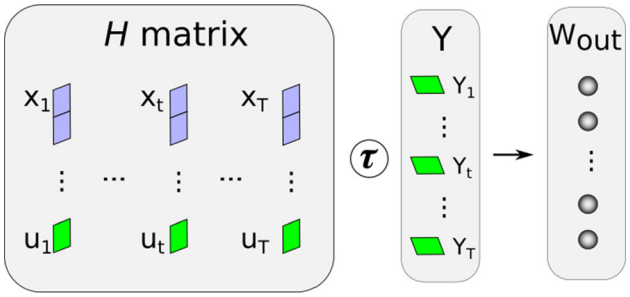

Fig. 2 ESN. Basic architecture. Regression with Tikhonov regularisation (expressed as τ) is used to calculate the weights of the output layer

of neurons within the network, and the output layer processes the resulting state beside the input to produce a prediction. Typically, this is the next step when working with time series. Figure 3 illustrates the test phase in the vanilla ESN architecture.

In some cases, the network takes each prediction as the next input, allowing the ESN to work in a generative mode (predicting sequences rather than just the next value in the series).

2.2 ESN applications

In the context of neural networks, DNNs and ESNs represent two distinct approaches to learning through training. In DNNs, a complex structure of layers of neurons connected by edges is constructed. These networks are trained by backpropagation to adjust the weights of the connections and improve the prediction accuracy. However, ESNs comprise three fundamental components: the input layer, a group of neurons organised in a graph-like structure (the reservoir), and the output layer. The input layer and the reservoir are generated randomly and their weights remain unchanged during training. The learning process in ESN focuses on only determining the set of weights for the output layer though linear regression algorithm (which has a lower complexity compared to backpropagation methods) that enables accurate predictions of the input data.

In summary, while DNNs aim to fine-tune all network weights during training, ESNs rely on a fixed and random structure to make predictions, adjusting only the output layer. In this way, ESN-based architectures offer a significant advantage over traditional DNNs, reducing the training time and computation required and, consequently, the size of data sets for training. The simplicity and robustness of ESNs, with their short training times, low resource consumption, and ability to approximate complex system dynamics, have made them attractive for researchers to apply in a wide range of scenarios. We can find applications of ESN in diverse fields such as speech recognition [17], natural language processing [18, 19], control systems [20], anomaly detection [21], image classification [22], music generation [23], phone call prediction [24] and bioinformatics [25].

Its popularity in recent years has led to several proposals based on the original architecture [26]. In [27], three ESN-based architectures were proposed and their ability to generate states was investigated. These architectures are:

- deepESN, where a series of layers are generated and fed with the output of the previous layer (Fig. 4a).
- deepESN-IA, similar to the previous one, but each layer receives the original input along with the output of the previous layer (Fig. 4b).
- groupedESN, where a series of reservoirs are generated and work in parallel to produce a single state (Fig. 4c).

In some cases, ESNs have been used in combination with other types of neural networks to improve their performance. For example, in [28] an ESN was integrated with a multi-layer perceptron (MLP) to address the task of colour image segmentation. Similarly, [29] used a combination of long short-term memory (LSTM) and ESN to control the temperature of an industrial hot-blast furnace. The integration of ESN with convolutional neural networks

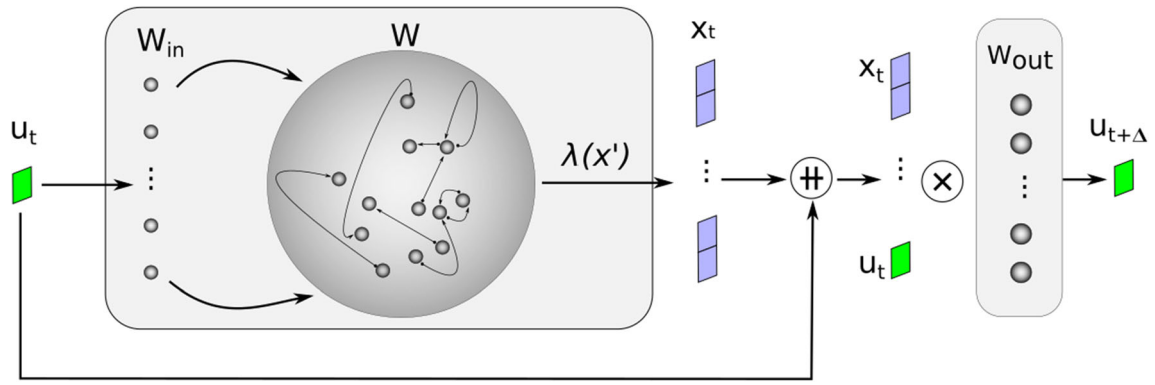

Fig. 3 ESN. Basic architecture. Testing. In the test phase, each new input causes a response in the state of the neurons, and this response, together with the input that caused it, is processed by the output layer to produce the final output

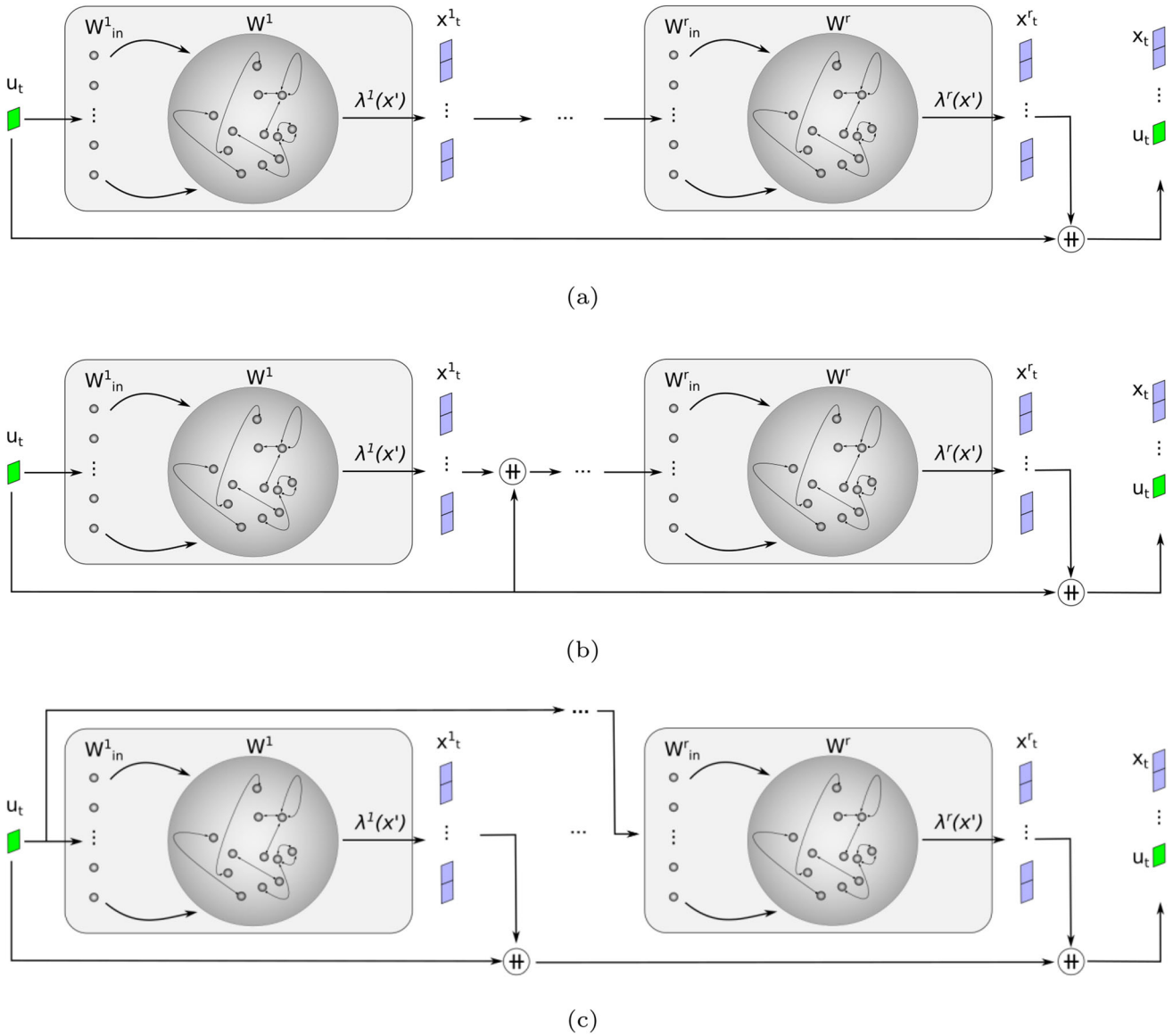

Fig. 4 Multi-Layer ESN-based architectures. **a** deepESN: The first layer processes the input as in the vanilla ESN, and the other layers receive the state of the previous reservoir as input. **b** deepESN-IA: The first layer processes the input as in the vanilla ESN, the other layers receive the state of the previous reservoir along with the original input, **c** groupedESN: a group of reservoirs operate independently as in vanilla ESNs, together they form a single state vector

(CNNs) has also been explored, as demonstrated in the research conducted by [30], where the network was applied to a solar energy prediction task.

ESNs have also been shown to be effective in image processing, as evidenced by [31, 32], after transforming images into temporal series. [33] also proposed a combination of ESNs with CNNs for image classification. A summary of these works and their scopes can be seen in Table 1.

Although, as a type of recurrent neural network, ESNs excel at processing sequential data, research on their capability to comprehend spatial relationships remains limited. The purpose of this paper is to push ESNs beyond their traditional use and to evaluate their performance in scenarios devoid of temporal information. Specifically, we investigate the network's capacity in image classification, where the emphasis shifts from temporal relationships to spatial understanding. As our experimental phase will demonstrate, this change makes certain network parameters, typically crucial for generating echo states (e.g., *leaking rate* or *spectral radius*), less significant in this context than in time series applications.

In the next section, we present a multi-reservoir echo state network (MRESN) architecture based on groupedESNs [27], which uses multiple reservoirs working in parallel. Each reservoir is tasked with processing the entire

Table 1 Previous work overview

Work	Architecture	Target	Context	Metric
[17]	ESN	Classification	Audio signal	WER
[18]	ESN	Prediction	Word sequences	Cosine, AUC, ...
[19]	ESN	Classification	Word sequences	Error rate
[20]	ESN	Prediction	Time series	NMSE
[21]	ESN	Prediction	Heart rate	Error rate
[22]	ESN	Classification	Image	Error rate
[23]	ESN	Prediction	Time series	RMSE
[25]	ESN	Prediction	Feature vector	AUC
[28]	ESN + MLP	Segmentation	Image	Accuracy
[29]	ESN + LSTM	Prediction	Time series	RMSE, MAE
[30]	ESN + CNN	Prediction	Time series	MAE, MAPE, ...
[31]	ESN	Classification	Image	Error rate
[32]	ESN	Predict./Classif	Various	Various
[33]	ESN + CNN	Classification	Images	Error rate
[34]	ESN	Prediction	Time series	NRMSE
[35]	ESN	Classification	Images	Error rate

input image, allowing the network to capture a variety of spatial features simultaneously. Unlike parallelESN [35], which partitions the input and distributes it to the reservoirs, each reservoir in our proposed architecture processes the entire image in a single step. This approach is reminiscent of using an ESN ensemble to determine the output, but with the key difference that the ensemble information is taken into account when finding the output layer weights, allowing us to obtain high-dimensional state vectors while significantly reducing training times without negatively affecting network performance.

3 Multi-reservoir ESN (MRESN) architecture for image classification

As we have seen, in ESNs hyperparameters have a direct impact on the performance of the model. Although most of these parameters strongly influence its behaviour when analysing time series, they become less relevant when trying to capture spatial relationships in images. In particular, as will be confirmed in the experimentation section, the spectral radius or the leaking rate do not have a significant impact on the results. This is because in this task we do not have a sequence in which the points depend on the previous ones, such as the benchmark signal Mackey glass, etc. In our case, there is independence between the different input examples, and they are analysed in a single step, and therefore the echoes containing the states become less relevant when analysing static data such as the image. However, the number of nodes in the reservoir becomes critical, and large reservoirs will be necessary to obtain good results. The biggest drawback is that as we increase the number of nodes in the reservoir, the computation required to train the model increases, directly affecting both the training time and the required memory space. The simplicity in calculating the weights of an ESN is both an advantage and a disadvantage because the larger the

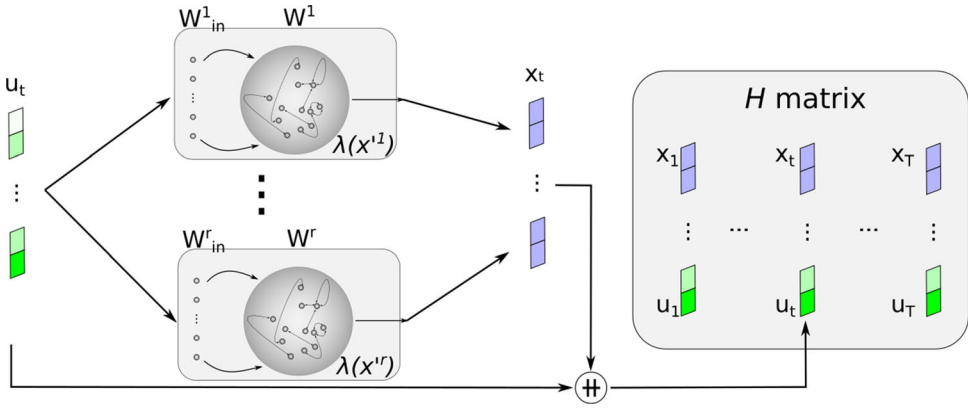

Fig. 5 groupedESN and MRESN architecture. A concatenation of each reservoir state yields the final state, which is stored in the H matrix during the training phase

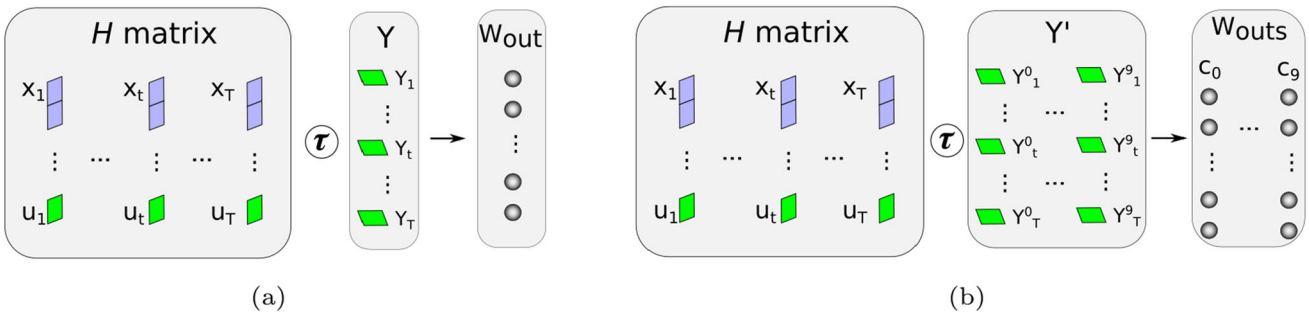

Fig. 6 **a** One layer for all classes. **b** One layer per class

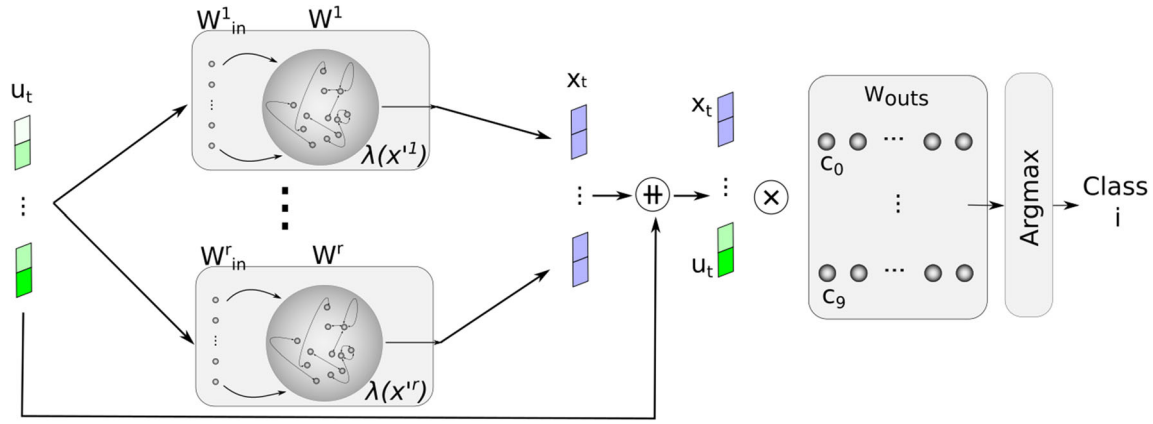

Fig. 7 MRESN architecture. The output layer is a matrix where each column is specialised in a particular class. Argmax or Softmax can be used as the final layer

number of nodes in the network, the larger the reservoir (which will grow quadratically with respect to the number of nodes) and the size of H, so the configuration of these networks will be limited by the hardware resources used.

In this study, we implemented the three architectures previously presented, making necessary modifications to tailor them for an image classification task that involves multiple output layers. Although deepESNs have been shown to be very robust in the time series domain [34] and even in real-world problems [36], they have not performed well in the image classification task proposed in this work.

Fig. 8 **a** MNIST 3 mm **b** FashionMNIST

However, the parallel architecture allows us to achieve high performance compared to ESNs.

Our MRESN architecture is based on groupedESN. It uses multiple reservoirs that work in parallel, each processing the entire input image. The high-dimensional state vectors produced play a crucial role in preserving spatial information and significantly reducing training times compared to DNNs and even ESNs with the same number of nodes. The process for generating the state of the MRESN architecture is described by:

$$\mathbf{x}_t^i = (1-\alpha^i)\mathbf{x}_{t-1}^i + \alpha^i \lambda^i (W_{\text{in}}^i \cdot \mathbf{u}_t + W^i \cdot \mathbf{x}_0), \quad i \in [1, r] \tag{3}$$

$$\mathbf{x}_t = (\mathbf{x}_t^1, \mathbf{x}_t^2, \ldots, \mathbf{x}_t^r) \in \mathbb{R}^{\sum_{i=1}^{r} |r_i|} \tag{4}$$

where r is the total number of reservoirs. At each time step t, the vector $\mathbf{x}_t$ is obtained by concatenating the vectors, $\mathbf{x}_t^i$, produced by each individual reservoir, W^i. This process results in a final vector of size $\sum_{i=1}^{r} |r_i|$. In the original ESN architecture, the size of the reservoir scales quadratically with the number of nodes, resulting in a significant increase in the memory footprint of the architecture when a large number of nodes are used. In contrast, groupedESN, parallelESN and MRESN require allocating the sum of the

squares of the nodes in each reservoir. As the number of nodes in each reservoir is typically much smaller than that required in the original architecture, the MRESN architecture exhibits a significantly reduced memory requirement.

$$\mathcal{O}_{\text{ESN}} = n \times n \tag{5}$$

$$\mathcal{O}_{\text{MRESN}} = \sum_{i=1}^{r} n_i \times n_i \tag{6}$$

where $n_i \ll n, \quad \text{for } i = 1, 2, ..., r$.

Despite the differences in state computing, the training phase of this architecture is similar to the original (see Fig. 5).

Another aspect to consider in image classification tasks is the independence among images from the data set. Consequently, it would not be advisable to rely on previous states of the network when processing a new image, which could lead to unexpected network behaviours. To avoid this, we use a zero vector $\mathbf{x}_0$ every time the network has to process a new image, and before computing the final state (which will be stored in the H matrix), a two-step initial transient is used to warm up the network.

Similarly to the vanilla ESN architecture, we store each input of the training set in the matrix H along with the resulting state. Once the H matrix is ready, we calculate the weights of the output layer in a single step using Tikhonov regularisation. However, a key difference from the basic architecture must be emphasised. Instead of using a target vector consisting of the next step in the input series, a target vector must contain the class of the image. This modification enables the ESN to perform classification tasks rather than prediction tasks (see Fig. 6a).

As suggested by [12], we can train specific output layers for each class instead of using a single output layer to decide the class of the image:

$$W_{\text{outs}}^{c} = (H^T H + \beta I)^{-1} H^T Y^c, \quad c \in [1, 2, \ldots, k] \tag{7}$$

In our experiments (MNIST and FashionMNIST), we will have $k = 10$. Training each output layer to specialise in a particular class improves accuracy. It does not require multiple passes through the data set and does not significantly increase the training process, despite the increase in k. We use the states already stored in the H matrix to train these layers and compare them with different target vectors derived from the original. These new target vectors are binary, containing 1 when the example belongs to the class we are training and 0 otherwise. In this way, we obtain specialised output layers for each class that can accurately decide whether the input belongs to a particular class or not (see Fig. 6b).

When we have multiple output layers, we need a mechanism to make a decision based on the output of the specialised output layers, for example, a softmax layer that takes the output of all the specialised output layers and returns a probability distribution over the classes. The class with the highest probability can then be chosen as the predicted class for the input (see Fig. 7).

4 Methodology

In this section, we evaluate the performance of the proposed architecture by conducting experiments on the MNIST and FashionMNIST data sets. To measure the effectiveness of our architecture, we will compare its

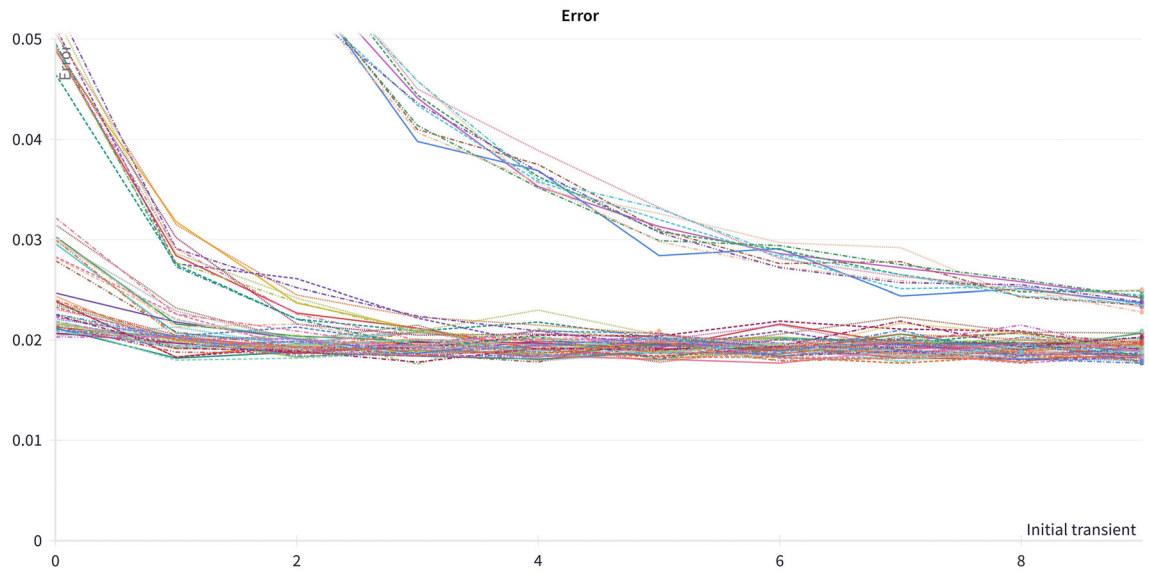

Fig. 9 Errors committed by networks with different values of α as the initial transient increases

Table 2 Range of hyperparameter values for grid search in vanilla ESN

Parameter	Minimum value	Maximum value
N (Nodes)	$5 \cdot 10^2$	$5 \cdot 10^3$
Density	0.1	0.7
α	0.1	0.9
β	10^{-6}	10^{-9}
ρ	0.5	1.5

results with those obtained with the original ESN architecture. Our experimental setup involves testing various parameters and configurations, and we conduct several trials to obtain reliable results.

4.1 MNIST and FashionMNIST data sets

The MNIST and FashionMNIST data sets are both commonly used benchmarks in image classification. MNIST consists of greyscale images of handwritten digits and is simple to use, making it a valuable benchmark for assessing the performance and robustness of image classification models. FashionMNIST includes greyscale images of clothing items instead of digits (see Fig. 8). Both data sets contain training and testing sets, and each example is associated with a label from a set number of classes. Both data sets include ten classes. There is no temporal relationship between the examples; they are independent of each other.

4.2 Experimentation

We have selected Julia for all the implementations: [12] for the original ESNs, and our own implementation for deepESN, deepESN-IA and MRESN. These architectures are trained and tested on the MNIST and FashionMNIST data sets using the same experimental setup: 60,000 examples for training and 10,000 for testing. Due to the poor results obtained for this task with deepESN and deepESN-IA, we focused on MRESN, which was compared to the vanilla ESN architecture. The same range of hyperparameters was tested for both architectures.

To ensure the robustness and reliability of our results, we conduct each experiment several times using different seeds to generate the random structures of the input layers and reservoirs and report the average performance of the architectures. Performance is evaluated using various metrics, including classification accuracy, training time, and memory usage.

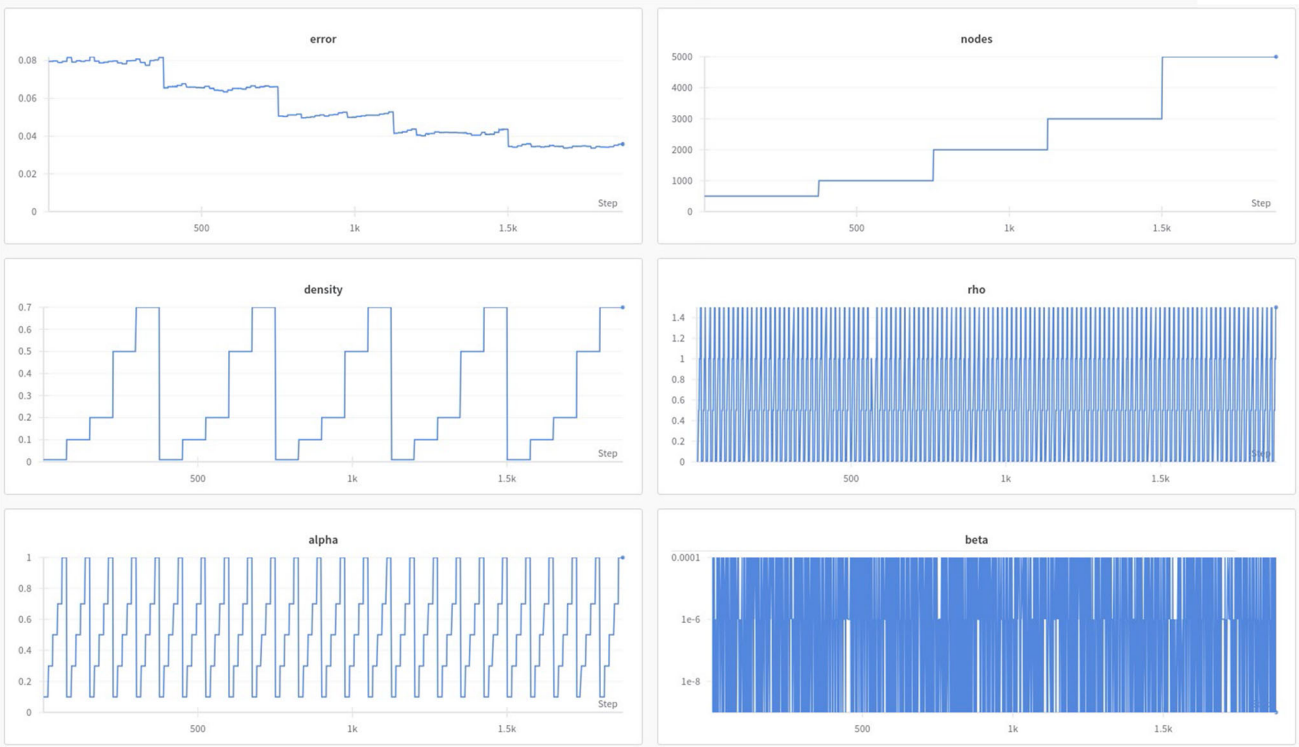

Fig. 10 Grid search Hyperparameters values

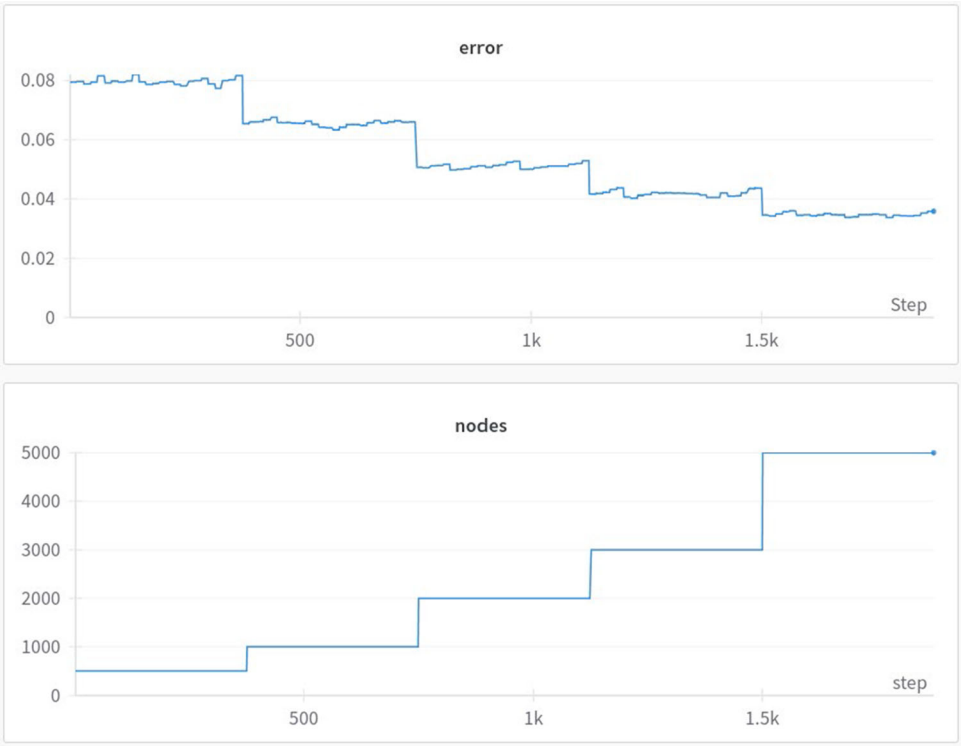

Fig. 11 Influence of the number of nodes (N) on network performance

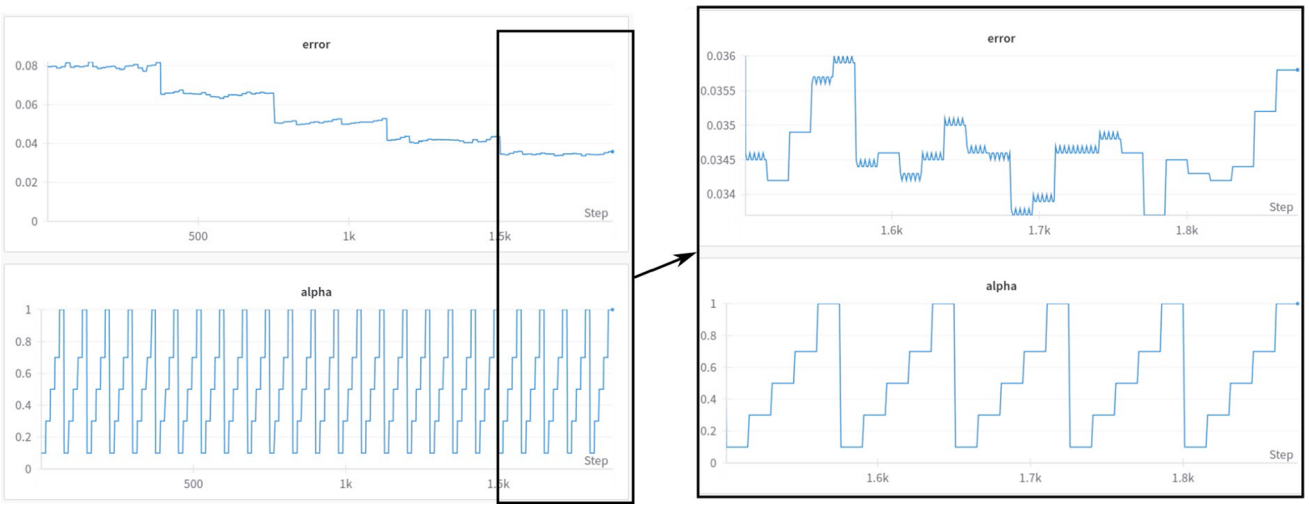

Fig. 12 Influence of the α parameter on network performance

All experiments are performed on a server equipped with an AMD Ryzen 7 5800X 16-core processor, 64 Gb of RAM, and an Nvidia GeForce RTX 4090 GPU card.

5 Results and discussion

One of the first questions that arose was whether it was necessary to have an initial transient. When using an ESN with leaky integrator [16] to analyse time series, it is interesting to note that at the beginning of training, the network has been tempered by a series of steps in which information is allowed to circulate through the network, but without registering these initial states in H. This allows more coherent states to be stored in H as they lack the noise that the cold start of the network can generate. In our case, lacking the temporal component, what we have done is to introduce the image into the network at each step and use the generated state along with the image to generate the next state. But is an initial transient necessary for this task?

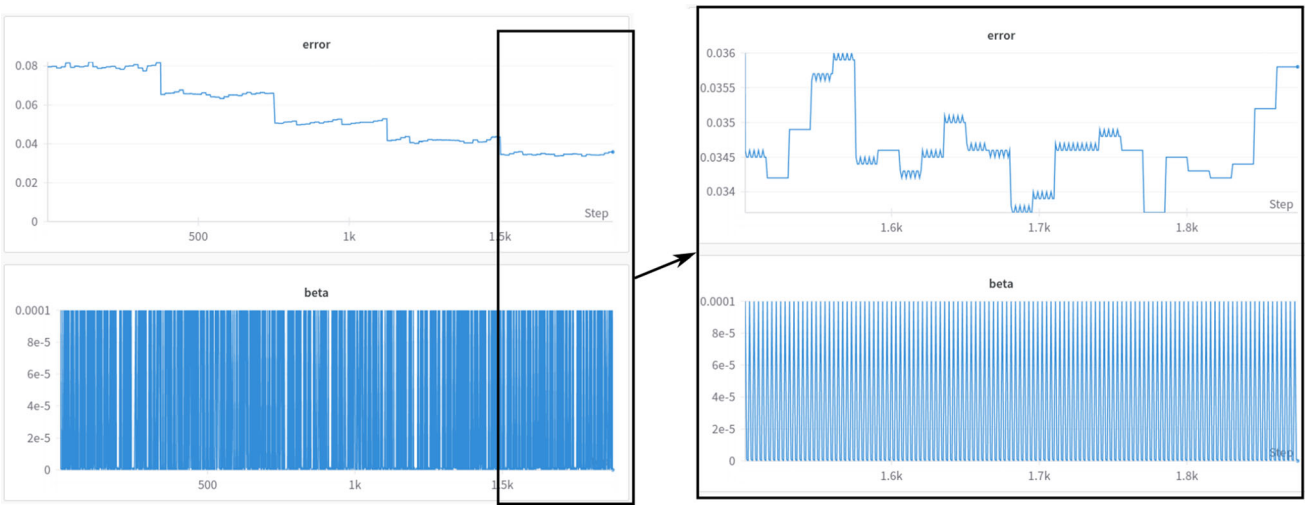

Fig. 13 Influence of the β parameter on network performance

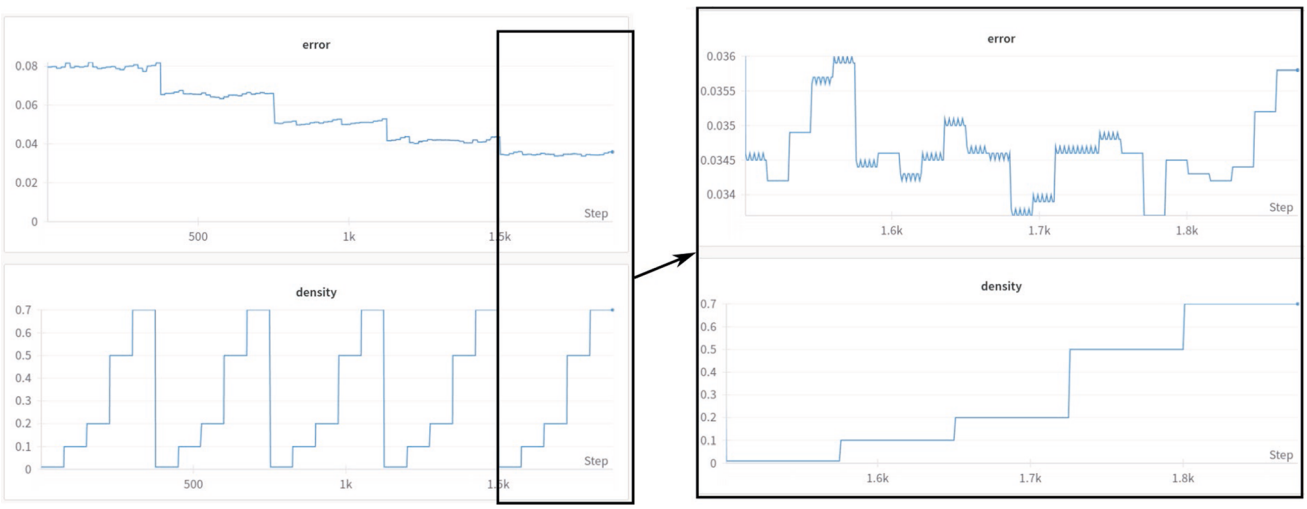

Fig. 14 Influence of the *density* of the network on performance

To answer this question, we conducted an experiment to test how different values for the initial transient affect the final result.

As α significantly influences the retention of information from the previous state within the network, we performed experiments with various networks, each assigned a value of $\alpha \in [0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 0.99]$. Illustrated in Fig. 9, the accuracy of these networks is observed as we vary the initial transient. The maximum accuracy is achieved by networks with $\alpha > 0.3$. These networks are depicted in the bottom section of Fig. 9. Networks with $0.1 \leq \alpha \leq 0.3$ exhibit a requirement for more initial transient steps to produce satisfactory results, as seen in the middle of Fig. 9. Notably, networks with $\alpha = 0.01$ do not reach the accuracy levels of other networks, even when subjected to high initial transient values. In any case, for networks with a sufficiently high value α, we can see that two initial transient steps are sufficient to obtain good results and that increasing this value would only lead to networks with longer training and inference times. For this reason, in this work we have chosen to set the value of the initial transient to 2 and to test the behaviour of the architecture by changing the rest of the parameters.

Once the initial transient value had been set, our next experiment was to analyse the performance of a vanilla ESN architecture with multiple output layers, one for each class. We measured its effectiveness over a range of different parameter sets. Based on [12], we identified a spectrum of possible values for each hyperparameter. Although this study provides a set of values that generally perform well, for specific problems and hyperparameters, outliers have been employed with success. In our study, we expanded the values suggested by the authors to explore

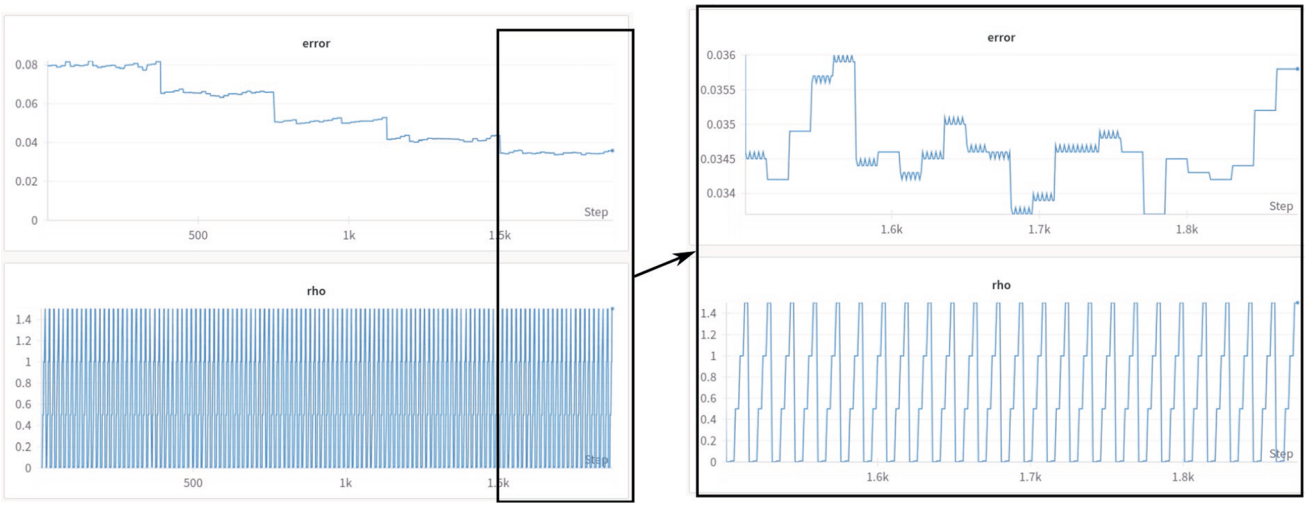

Fig. 15 Influence of the ρ parameter on network performance

Table 3 Hyperparameters used in deepESN and deepESN-IA

Nodes per layer	Density	Transient	ρ	α	β	γ
2000	0.2	0	[0.5, 1.5]	0.7	10^{-8}	[0.5, 1.5]

the limits of our grid search. The ranges we considered are summarised in Table 2 and Fig. 10.

The influence of each hyperparameter on the final performance of the network can be seen in Figs. 11–15.

Although these figures show the results obtained using MNIST data set, the same trends are observed in FashionMNIST data set.

For this range of hyperparameters, the best results were obtained with $N = 5000$ nodes, with an error ranging from 0.0337 to 0.0345. This network performed best with different sets of hyperparameters. Initially, when $\alpha = 0.7$ and density $= 0.5$, the results were consistently favourable, regardless of the values assigned to ρ and β (within the dimension chosen in the grid search). Similarly, the minimum error was also achieved when $\alpha = 0.5$, density $= 0.2$, and $\beta \in [10^{-9}, 10^{-6}]$, regardless of the specific value of ρ. It should be noted that the hyperparameter space is extremely rough and that different sets may work well for other reservoirs. However, the results indicate that the parameters *leaking-rate* (α), *regularisation coefficient* (β), *density*, and *spectral-radius* (ρ) do not have a significant impact on network performance. This is expected since ρ and α are typically more relevant for time series data, where the network needs to maintain a memory of previous states. In our approach, the network has only two steps of *initial-transient* before computing the final state, which reduces the risk of the state converging to atypical values, so some hyperparameters have little impact on the final results. On the other hand, N has a significant impact on the performance of the network. Increasing the number of nodes (even beyond the maximum used in this grid search) reduces the prediction error because the state vector becomes larger and can encode more information about the input.

The downside is that increasing N also increases the time and memory required for training since it affects the size of the structures used, especially H.

It should be noted that the number of nodes in the network appears to be the critical factor in capturing spatial relationships. This observation allows greater flexibility in adjusting other hyperparameters, such as *spectral radius* and *leaking rate*, in architectures designed for scenarios involving spatial and temporal relationships, such as video sequences.

After this exploration of the behaviour in vanilla ESNs, we implemented the architectures proposed by [27] (see Fig. 4) and enhanced their design by incorporating multiple specialised output layers, customising them for a classification task. We used an argmax layer as the final step. The reservoirs and the input layer are randomly generated. We fix $\alpha = 0.7$, but we give random values for ρ and λ (*input scaling*) to improve diversity. The range of hyperparameters used for the networks can be found in Table 3. In our first approach, we want to test if initial transient is relevant in these architectures and fix this parameter to 0.

These networks were tested for depths ranging from 1 to 10 layers. However, neither deepESN nor deepESN-IA yielded good results, and we observed that these results

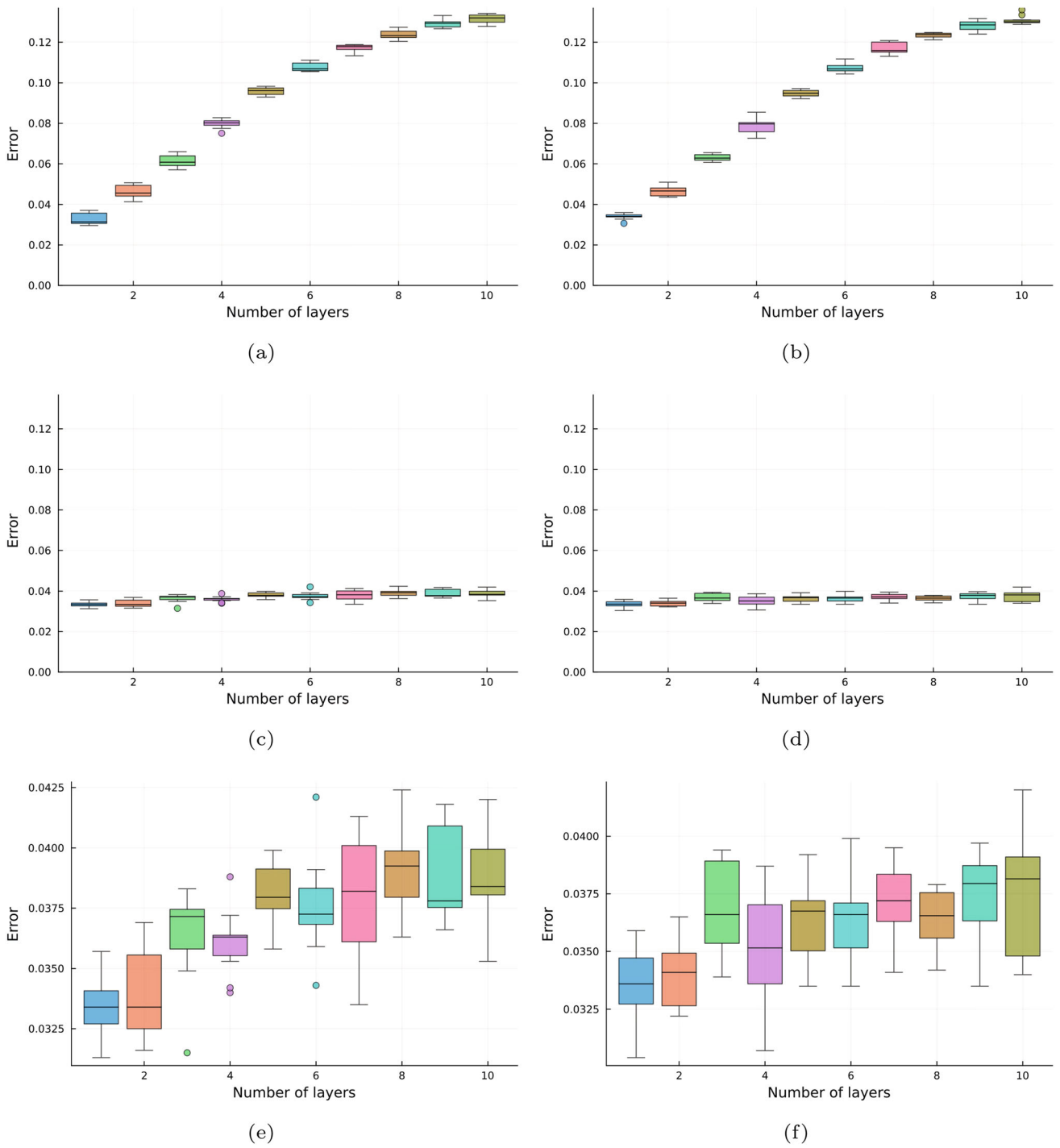

Fig. 16 **a** deepESN, **b** deepESN-IA, **c** deepESN with Identity matrix as Input Layer, **d** deepESN-IA with Identity matrix as Input Layer. **e** Same as **c** but in a shorter range, **f**) Same as **d** but in a shorter range

worsened as we increased the number of layers in the network (Fig. 16a and b). One possible explanation is that, in these architectures, there is a linear modification of the input data performed by the input layer of each ESN in the hidden layers, causing the error to increase as we increase the number of hidden layers. For example, if we replace the input layers in the reservoirs located in the hidden layers with an identity matrix that does not alter the input received (that is, the state of the previous layer), the results become more consistent (Fig. 16e and f). Anyway, lack of initial transient does not help the network make better predictions. As we will see later in the comparison, some

initial transient steps help the network perform better, but this was not enough to match the accuracy of other ESN-based models.

Although sequential architectures based on stacked ESN layers may not provide optimal performance for this task, the use of multiple ESNs in parallel proves to be a successful approach. The MRESN approach allows us to generate large state vectors and significantly reduce the required training time compared to the conventional ESN architecture. This time difference increases significantly compared to the vanilla-ESN architecture, where training times are significantly longer. Furthermore, the overall performance of the MRESN network improves with the same number of nodes compared to ESN or deepESN. The multi-reservoir architecture provides a promising solution to generate large state vectors while maintaining high accuracy and reducing training times. Tables 4 and 5 compare the three implemented architectures and the vanilla ESN in terms of execution time and accuracy, respectively.

Our experiments demonstrated that the proposed architecture outperforms the original ESN architecture in terms of classification accuracy on both data sets. Specifically, the proposed architecture achieved 98.43% accuracy in the MNIST data set, compared to 97.84% for the original ESN architecture. In the FashionMNIST data set, the proposed architecture achieved 89.12% accuracy, while the original ESN architecture achieved 88.1%. Notably, the new architecture not only performs better but also requires less training time and memory for the used structures.

Finally, Table 6 presents a comparative analysis of 3 MRESN architectures, one with 20 reservoirs of 1000 nodes, one with 6 reservoirs of 1000 nodes and finally a small network with 4 reservoirs of 500 nodes. These architectures are compared with other architectures based on ESN and DNN, focusing on two key metrics: accuracy and number of trainable parameters. This comparison provides insight into the effectiveness of the MRESN architecture in achieving competitive levels of accuracy while maintaining a manageable number of trainable parameters across different r x N configurations. A graphical representation of this table is shown in Fig. 17.

6 Conclusions

In this paper, we have investigated the behaviour of different ESN-based architectures and evaluated their performance in the context of image classification. For the basic architecture, we tested the network response under various sets of hyperparameters and analysed different modifications for deepESN-type networks.

Table 4 Global execution time

Model	r x Nodes	Time CPU	Time GPU	FPS CPU	FPS GPU
ESN	1 x 20000	04:12:00	00:40:59	4.63	28.47
MRESN	20 x 1000	**00:39:59**	00:11:30	**29.18**	101.45
deepESN	10 x 2000	01:02:03	**00:10:53**	18.80	**107.20**
deepESN-IA	10 x 2000	01:29:55	00:12:29	12.97	93.46
ESN	1 x 15000	02:16:00	00:20:42	8.58	56.36
MRESN	15 x 1000	**00:26:06**	00:07:30	**44.70**	155.56
deepESN	8 x 2000	00:46:29	**00:07:29**	25.10	**155.90**
deepESN-IA	8 x 2000	01:09:03	00:08:39	16.90	134.87
ESN	1 x 10000	01:11:00	00:08:36	16.43	135.66
MRESN	10 x 1000	**00:18:30**	00:04:00	**63.06**	291.67
deepESN	5 x 2000	00:26:48	**00:03:32**	43.53	**330.19**
deepESN-IA	5 x 2000	00:39:58	00:04:09	29.19	281.12
ESN	1 x 6000	00:31:30	00:02:24	37.04	486.10
MRESN	6 x 1000	00:16:12	00:02:01	72.02	578.50
deepESN	3 x 2000	**00:14:26**	**00:01:40**	**80.83**	**700.00**
deepESN-IA	3 x 2000	00:21:54	00:02:03	53.27	569.11
ESN	1 x 2000	**00:01:30**	**00:00:31**	**777.78**	**2258.06**
MRESN	4 x 500	00:01:45	00:00:51	666.67	1372.55
deepESN	2 x 1000	00:01:36	00:01:12	729.17	977.22
deepESN-IA	2 x 1000	00:02:17	00:01:14	510.95	945.95

Bold values represent the best results achieved in terms of accuracy, number of trainable parameters, FPS or time

Table 5 Networks accuracy for MNIST and FashionMNIST

Architecture	Reservoirs (r) x Nodes	MNIST	FashionMNIST
ESN	1 x 20000	97.70 ±0.10	88.70 ±0.10
MRESN	20 x 1000	**98.36 ±0.07**	**89.13 ±0.09**
deepESN	10 x 2000	89.97 ±0.011	69.20 ±0.016
deepESN-IA	10 x 2000	96.63 ±0.003	86.43 ±0.003
ESN	1 x 15000	97.55 ±0.04	88.48 ±0.06
MRESN	15 x 1000	**98.28 ±0.03**	**89.10 ±0.20**
deepESN	8 x 2000	89.46 ±0.013	68.03 ±0.014
deepESN-IA	8 x 2000	96.78 ±0.002	86.97 ±0.003
ESN	1 x 10000	97.28 ±0.15	88.36 ±0.13
MRESN	10 x 1000	**97.76 ±0.30**	**88.45 ±0.20**
deepESN	5 x 2000	90.54 ±0.014	69.42 ±0.021
deepESN-IA	5 x 2000	97.07 ±0.002	87.23 ±0.003
ESN	1 x 6000	96.80 ±0.10	**87.84 ±0.15**
MRESN	6 x 1000	**97.12 ±0.04**	87.50 ±0.15
deepESN	3 x 2000	90.86 ±0.009	69.58 ±0.016
deepESN-IA	3 x 2000	97.01 ±0.001	87.37 ±0.003
ESN	1 x 2000	95.00 ±0.14	86.44 ±0.30
MRESN	4 x 500	**96.64 ±0.001**	**86.77 ±0.13**
deepESN	2 x 1000	83.95 ±0.02	63.16 ±0.02
deepESN-IA	2 x 1000	95.99 ±0.003	82.59 ±0.001

Bold values represent the best results achieved in terms of accuracy, number of trainable parameters, FPS or time

Table 6 Architecture comparison with other S.O.T.A. methods

Architecture	Trainable parameters	Accuracy	Work	Year
Ensembled CNN	$> 10^6$	**99.9%**	[37] **SOTA**	2020
CNN + HVC	$1.5x10^6$	99.87	[38]	2023
ESN + CNN	90000	99.25%	[33]	2018
MRESN	20000	98.36%	**This work**	2023
DNN-5	175180	97.2	[39]	2023
MRESN	6000	97.12%	**This work**	2023
DNN-3	80568	97.0	[39]	2023
DNN-2	5500	96.4	[39]	2023
MRESN	2000	95.6%	**This work**	2023
ParallelESN	1600	95.3%	[35]	2021
ESN	2000	92.75%	[22]	2022
DeepESN	**400**	86%	[32]	2021

Bold values represent the best results achieved in terms of accuracy, number of trainable parameters, FPS or time

Building upon groupedESN, we introduced the MRESN architecture for image classification. This architecture uses multiple reservoirs that operate in parallel to capture pure spatial relationships in images without transforming them into time series. Our experiments on the MNIST data set demonstrated that the MRESN architecture achieved a classification accuracy of 98.43%, surpassing both the classical ESN architecture and other state-of-the-art ESN-based architectures, all while requiring less training time.

Unlike previous approaches that divide the image into parts and allocate one slice to each reservoir, the MRESN architecture processes the entire image in each reservoir. This method uses the information of the ensemble to determine the weights of the output layer, overcoming the limitations of vanilla ESNs and providing more accurate predictions. Furthermore, utilising a set of smaller reservoirs instead of a single large structure with many nodes results in reduced training and testing times for the network

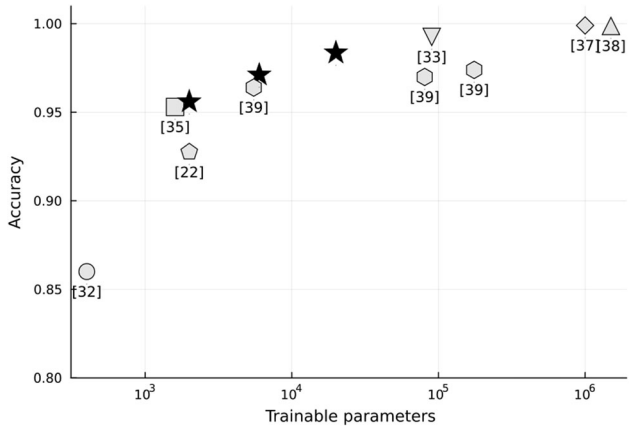

Fig. 17 Relationship between accuracy and trainable parameters for different models. Stars denote the MRESN architecture, while other symbols represent models identified by their reference numbers in Table 5

and significantly lowers the required disk space for model execution.

Our results suggest that the proposed MRESN architecture can serve as a viable alternative for learning spatial relationships in scenarios where energy efficiency and training time are critical concerns. The MRESN architecture holds promise for applications in image classification tasks. Future research could explore its potential for generalisation to more complex data sets. Furthermore, investigating the results when preprocessing data sets by applying techniques such as data augmentation [40] or feature extraction [41] could further enhance its utility.

In this study, we examine the response of the network to different sets of hyperparameters and demonstrate that the number of nodes within the network emerges as a critical factor in capturing spatial relationships. This discovery provides greater flexibility for adjusting other hyperparameters, such as *spectral radius* and *leaking rate*, in architectures designed for scenarios that involve not only spatial but also temporal relationships, as seen in video sequences. This adaptability should allow these networks to be optimised in various contexts without compromising their ability to capture spatial relationships, a hypothesis that we intend to test in our future work.

Acknowledgements This research has been partially funded by the projects DISARM (PDC2021-121197-C21) and HORUS (PID2021-126359OB-I00) funded by MCIN / AEI / 10.13039 / 501100011033, by the "European Union NextGenerationEU/PRTR" and Project ID2PPAC funded through the ECSEL framework program (Grant agreement ID: 101007254). This research project is also supported by the REPNIN++ network (RED2022-134355-T) within the framework of the Spanish State Program to Promote Scientific and Technical Research and its Transfer, part of the Innovation Research Plan 2021-2023. NVIDIA also supported this investigation through the donation of one NVIDIA A100 to our colleague Miguel A. Martinez-del-Amor.

Funding Funding for open access publishing: Universidad de Sevilla/CBUA.

Data availability The data sets used to train and evaluate these networks are publicly available at: (1)https://git-disl.github.io/GTDLBench/datasets/mnist_datasets/. (2) https://www.kaggle.com/datasets/zalando-research/fashionmnist. The code used in this work is available in: https://github.com/enrloport/MR-ESN

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

1. Guo Y, Yu H, Ma L, Zeng L, Luo X (2023) Thfe: a triple-hierarchy feature enhancement method for tiny boat detection. Eng Appl Artif Intell 123:106271. https://doi.org/10.1016/j.engappai.2023.106271
2. Lei Q, Guo Y, Ma L, Luo X (2023) Few-shot object detection via instance-wise and prototypical contrastive learning. In: Chang S (ed) The 35th international conference on software engineering and knowledge engineering, SEKE 2023, KSIR virtual conference center, USA, July 1–10. KSI Research Inc., pp 685–690. https://doi.org/10.18293/SEKE2023-129
3. Bhimavarapu U (2022) Irf-lstm: enhanced regularization function in lstm to predict the rainfall. Neural Comput Appl 34(22):20165–20177. https://doi.org/10.1007/s00521-022-07577-8
4. Sahin ME, Ulutas H, Yuce E, Erkoc MF (2023) Detection and classification of Covid-19 by using faster r-cnn and mask r-cnn on ct images. Neural Comput Appl 35(18):13597–13611. https://doi.org/10.1007/s00521-023-08450-y
5. Raju ASN, Jayavel K, Rajalakshmi T (2023) An advanced diagnostic colorectalcadx utilises cnn and unsupervised visual explanations to discover malignancies. Neural Comput Appl 35(28):20631–20662. https://doi.org/10.1007/s00521-023-08859-5
6. Beohar D, Rasool A (2021) Handwritten digit recognition of mnist dataset using deep learning state-of-the-art artificial neural network (ann) and convolutional neural network (cnn). In: 2021 International conference on emerging smart computing and informatics (ESCI), pp 542–548. https://doi.org/10.1109/ESCI50559.2021.9396870
7. Kaziha O, Bonny T (2019) A comparison of quantized convolutional and lstm recurrent neural network models using mnist. In: 2019 International conference on electrical and computing

technologies and applications (ICECTA), pp 1–5. https://doi.org/10.1109/ICECTA48151.2019.8959793
8. Coleman C, Narayanan D, Kang D, Zhao T, Zhang J, Nardi L, Bailis P, Olukotun K, Ré C, Zaharia M (2017) Dawnbench: an end-to-end deep learning benchmark and competition. Training 100(101):102
9. Jaeger H (2001) The "echo state" approach to analysing and training recurrent neural networks-with an erratum note'. www.researchgate.net
10. Kim T, King B (2020) Time series prediction using deep echo state networks. Neural Comput Appl. https://doi.org/10.1007/s00521-020-04948-x
11. Shahi S, Fenton FH, Cherry EM (2022) Prediction of chaotic time series using recurrent neural networks and reservoir computing techniques: a comparative study. Mach Learn Appl 8:100300. https://doi.org/10.1016/j.mlwa.2022.100300
12. Lukoševičius M (2012) In: Montavon G, Orr GB, Müller K-R (eds) A practical guide to applying echo state networks. Springer, Berlin, Heidelberg, pp 659–686. https://doi.org/10.1007/978-3-642-35289-8_36
13. Tian Z, Gao X, Li S, Wang Y (2015) Prediction method for network traffic based on genetic algorithm optimized echo state network. Jisuanji Yanjiu yu Fazhan/Comput Res Dev 52:1137–1145. https://doi.org/10.7544/issn1000-1239.2015.20131757
14. Xue Y, Zhang Q, Neri F (2021) Self-adaptive particle swarm optimization-based echo state network for time series prediction. Int J Neural Syst 31:234. https://doi.org/10.1142/S012906572150057X
15. Thiede LA, Parlitz U (2019) Gradient based hyperparameter optimization in echo state networks. Neural Netw 115:23–29. https://doi.org/10.1016/j.neunet.2019.02.001
16. Jaeger H, Lukoševičius M, Popovici D, Siewert U (2007) Optimization and applications of echo state networks with leaky-integrator neurons. Neural Netw 20(3):335–352. https://doi.org/10.1016/j.neunet.2007.04.016. Echo State Networks and Liquid State Machines
17. Shrivastava H, Garg A, Cao Y, Zhang Y, Sainath T (2021) Echo state speech recognition.https://doi.org/10.48550/ARXIV.2102.09114
18. Tong M, Bickett A, Christiansen E, Cottrell G (2007) Learning grammatical structure with echo state networks. Neural Netw 20:424–432. https://doi.org/10.1016/j.neunet.2007.04.013
19. Cabessa J, Hernault H, Kim H, Lamonato Y, Levy YZ (2021) Efficient text classification with echo state networks. In 2021 International joint conference on neural networks (IJCNN), pp 1–8 https://doi.org/10.1109/IJCNN52387.2021.9533958
20. Salmen M, Plöger P (2005) Echo state networks used for motor control. Robot Autom 18:1953–1958. https://doi.org/10.1109/ROBOT.2005.1570399
21. Chen Q, Zhang A, Huang T, He Q, Song Y (2020) Imbalanced dataset-based echo state networks for anomaly detection. Neural Comput Appl. https://doi.org/10.1007/s00521-018-3747-z
22. Sun J, Li L, Peng H (2021) An image classification method based on echo state network. In: 2021 International conference on neuromorphic computing (ICNC), pp 165–170 https://doi.org/10.1109/ICNC52316.2021.9607999
23. Krusna Lukosevicius M (2018) Predicting Mozart's next note via echo state networks. www.piano-midi.de
24. Bianchi FM, Scardapane S, Uncini A, Rizzi A, Sadeghian A (2015) Prediction of telephone calls load using echo state network with exogenous variables. Neural Netw 71:204–213. https://doi.org/10.1016/j.neunet.2015.08.010
25. Guo X, Qian Y, Tiwari P, Zou Q, Ding Y (2022) Kernel risk sensitive loss-based echo state networks for predicting therapeutic peptides with sparse learning. In: 2022 IEEE international conference on bioinformatics and biomedicine (BIBM), pp 6–11 . https://doi.org/10.1109/BIBM55620.2022.9994902
26. Sun C, Song M, Hong S, Li H (2020) A review of designs and applications of echo state networks. http://arxiv.org/abs/2012.02974
27. Gallicchio C, Micheli A, Pedrelli L (2017) Deep reservoir computing: a critical experimental analysis. Neurocomputing. https://doi.org/10.1016/j.neucom.2016.12.089
28. Souahlia A, Belatreche A, Benyettou A, Foitih Z, Benkhelifa E, Curran K (2020) Echo state network-based feature extraction for efficient color image segmentation. Concurr Comput Pract Exp 32:5719. https://doi.org/10.1002/cpe.5719
29. Yang Y, Zhao X, Liu X (2020) A novel exhaust gas temperature prediction method of hot blast stove. In: 2020 39th Chinese control conference (CCC), pp 5916–5921. https://doi.org/10.23919/CCC50068.2020.9189443
30. Mustaqeem Ishaq M, Kwon S (2022) A cnn-assisted deep echo state network using multiple time-scale dynamic learning reservoirs for generating short-term solar energy forecasting. Sustain Energy Technol Assessm 52:102275. https://doi.org/10.1016/j.seta.2022.102275
31. Schaetti N, Salomon M, Couturier R (2016) Echo state networks-based reservoir computing for mnist handwritten digits recognition. In: 2016 IEEE intl conference on computational science and engineering (CSE) and IEEE intl conference on embedded and ubiquitous computing (EUC) and 15th intl symposium on distributed computing and applications for business engineering (DCABES), pp 484–491. https://doi.org/10.1109/CSE-EUC-DCABES.2016.229
32. Barredo Arrieta A, Gil-Lopez S, Laña I, Bilbao N, Del Ser J (2022) On the post-hoc explainability of deep echo state networks for time series forecasting, image and video classification. Neural Comput Appl. https://doi.org/10.1007/s00521-021-06359-y
33. Tong Z, Tanaka G (2018) Reservoir computing with untrained convolutional neural networks for image recognition. In: 2018 24th international conference on pattern recognition (ICPR), pp 1289–1294. https://doi.org/10.1109/ICPR.2018.8545471
34. Hu R, Tang Z, Song X, Luo J, Wu E, Chang S (2021) Ensemble echo network with deep architecture for time-series modeling. Neural Comput Appl. https://doi.org/10.1007/s00521-020-05286-8
35. Gardner SD, Haider MR, Moradi L, Vantsevich V (2021) A modified echo state network for time independent image classification. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/MWSCAS47672.2021.9531776
36. Dettori S, Matino I, Colla V, Speets R (2022) A deep learning-based approach for forecasting off-gas production and consumption in the blast furnace. Neural Comput Appl. https://doi.org/10.1007/s00521-021-05984-x
37. An S, Lee M, Park S, Yang H, So J (2020) An ensemble of simple convolutional neural network models for MNIST digit recognition
38. Byerly A, Kalganova T, Dear I (2020) A branching and merging convolutional network with homogeneous filter capsules. arXiv:2001.09136v4
39. Pishchik E (2023) Trainable activations for image classification. https://doi.org/10.20944/preprints202301.0463.v1
40. Saini D, Malik R (2021) Image data augmentation techniques for deep learning-a mirror review. In: 2021 9th International conference on reliability, infocom technologies and optimization (trends and future directions) (ICRITO), pp 1–5. https://doi.org/10.1109/ICRITO51393.2021.9596262
41. Cao X, Guo Y, Yang W, Luo X, Xie S (2023) Intrinsic feature extraction for unsupervised domain adaptation. Inte J Web Inf Syst 19(5/6):173–189. https://doi.org/10.1108/IJWIS-04-2023-0062