International Association of Applied Mathematics and Mechanics – Archive for Students –

Reservoir Computing for Image Classification

Mehdi Ghorbani[a,⋆], Manish Yadav[a], Merten Stender[a]

[a] Technische Universität Berlin, Berlin, Germany

received 20.10.2025, accepted 26.02.2026, published 01.03.2026

⋆ corresponding author: mahdi.ghorbaniaghooyeh@campus.tu-berlin.de
supervisor: Merten Stender and Manish Yadav, Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin, Berlin, Germany

Abstract: *Reservoir Computing (RC) is a machine learning framework designed to solve time-dependent tasks, but it remains less explored for time-independent tasks. In this work, we adapt RC for image classification by flattening input images into one-dimensional vectors and feeding the entire vector in a single step into an RC. We evaluate our approach on the MNIST and COIL-100 datasets, demonstrating that RC achieves competitive accuracy while requiring significantly shorter training times compared to traditional neural networks. Our experiments show that the difference between the worst and best performance across reservoir network densities is 1–3%. Increasing the reservoir size generally improves accuracy; for instance, increasing the reservoir from 100 to 1000 typically yields a 10–15% improvement, whereas increasing it further from 1000 to 2000 results in only a 3–5% gain. This work explores RC as a potential option for image classification on standard and moderately sized datasets, particularly in compute-constrained scenarios, though its accuracy is notably lower than typical deep learning approaches.*

Keywords: network theory, classification, reservoir computing, random network, scale-free network

1 Introduction

Image processing using deep learning has seen significant advancements in recent years, driven by the development of powerful neural network architectures [24]. RC represents an advancement in neural networks, particularly for handling temporal data [9]. At its core, RC has a reservoir that is a fixed, randomly generated network [19]. This reservoir acts as a dynamic memory system that processes input data through a series of transformations [25]. Unlike conventional recurrent neural networks (RNN), where both recurrent connections and output weights are typically trained through backpropagation in time [31], RC simplifies the training process significantly.

The reservoir consists of fixed, randomly generated weights and connections that remain unchanged over the learning process. This structure is crucial because it defines a nonlinear dynamical system capable of capturing temporal dependencies [33] in the input data. By keeping the reservoir fixed, RC avoids the computationally intensive task of adjusting recurrent connections during training. Instead, the focus is primarily on training the output weights, which connect the reservoir's internal states to the desired output [25]. RC typically employs a learning rule such as ridge regression [16] or Echo State Networks (ESNs) [19] to train the output weights [20]. These methods leverage the rich dynamics of the reservoir to map input sequences to output predictions efficiently. RC has found applications in a wide range of fields such as time-series prediction [32], computational neuroscience [6], and signal processing

[11]. Its ability to effectively process temporal data with minimal computational overhead [21] makes it particularly suitable for real-time applications and scenarios where computational resources are limited. One of the key advantages of RC over traditional RNNs lies in its training simplicity and efficiency [20]. By leveraging a pre-existing randomly initialized reservoir and focusing on training only the output weights, RC reduces the risk of overfitting and accelerates convergence [25] during training. Despite its advantages, RC may underperform in tasks requiring hierarchical feature extraction, such as high-resolution image classification. The performance heavily depends on the reservoir's connectivity and size; however, a larger reservoir does not necessarily lead to better performance [18]. The performance also depends on the choice of training algorithm and the reservoir state. The time increment updates the reservoir state as

$$r(t+\triangle t) = (1-\alpha)\cdot r(t) + \alpha\cdot\sigma[W_{res}\cdot r(t) + W_{in}\cdot x(t)] \quad (1)$$

where $r(t)$ represents the reservoir state at time t, and each element $r_i(t)$ denotes the activation level of reservoir node i. The reservoir state at time $t+\Delta t$, denoted by $r(t+\Delta t)$, represents how the reservoir node activations evolve over a small time increment Δt. The leakage rate α, also known as the integration time constant, determines how quickly information from the current state $r(t)$ decays or persists into the next time step $r(t+\Delta t)$ [5]. When $\alpha = 0$, the reservoir states remain unchanged over time, indicating no leakage [5]. As α increases towards 1, the reservoir states evolve rapidly, reflecting a higher degree of temporal integration. The reservoir connectivity matrix is denoted by W_{res}, where $W_{res_{ij}}$ represents the weight of the connection from node j to node i [14]. W_{res} defines the intrinsic dynamics and interactions between reservoir nodes, influencing how information propagates and transforms within the reservoir. The input weight matrix W_{in} connects the input $x(t)$ to the reservoir nodes [19]. The input vector $x(t)$ influences the activation levels of reservoir nodes through the weighted connections $W_{in}\cdot x(t)$. The activation function $\sigma[\cdot]$ is applied element-wise to the sum of inputs to each reservoir node. Typically, $\sigma[\cdot]$ is a nonlinear function (e.g., sigmoid, tanh) that introduces nonlinearity into the reservoir dynamics, crucial for capturing complex input-output mappings and enhancing computational capabilities. The time increment equation, Eq. 1, governs how reservoir states evolve over successive time steps in RC. At each time t, reservoir nodes integrate information from their current state $r(t)$, influenced by both internal dynamics $W_{res}\cdot r(t)$ and external inputs $W_{in}\cdot x(t)$ [25]. The parameter α balances the influence of current state retention $(1-\alpha)\cdot r(t)$ with new information integration $\alpha\cdot\sigma[W_{res}\cdot r(t) + W_{in}\cdot x(t)]$, modulating the reservoir's responsiveness to temporal dependencies and input signals [5]. During training, the output weights are computed as

$$y_{train} = W_{out}\cdot X_{res} \quad (2)$$

where the output weight matrix W_{out} maps the reservoir states X_{res} [25] to the target outputs y_{train} during the training phase. It represents the trainable parameters [4] that adjust the transformation from reservoir responses to desired output predictions, essential for achieving accurate and effective learning in RC. The matrix of reservoir states X_{res} is collected over time, where each column represents the activation states of reservoir nodes for a specific time instance or input sequence. These reservoir states encapsulate the dynamic responses of the reservoir network to input signals, encoding temporal patterns [2] [32] and features crucial for subsequent prediction or classification tasks. The vector or matrix of target training outputs y_{train} corresponds to the input sequences or instances that drive the reservoir network during training. It serves as the ground truth or desired response that the RC aims to approximate or predict accurately through its learning process [20]. The output weights W_{out} are computed by performing a linear regression [20] of reservoir states X_{res} against target outputs y_{train}. This regression minimizes the squared error between predicted and actual outputs, guiding the RC toward convergence and improving its accuracy in predicting future outputs or classifications. In the last layer, ridge regression is used for final prediction. Ridge regression is a form of linear regression that applies L2 regularization to the model, which penalizes large coefficients in order to reduce overfitting. This regularization term is controlled by a hyperparameter λ, which balances the trade-off between fitting the model to the data and maintaining smaller coefficients. In the context of RC, ridge regression is often used to train the output layer of the reservoir, where the complex dynamics of the reservoir's state are combined with the input signals to predict the output. Ridge regression helps ensure that the readout weights are not too large, improving the model's generalization and stability [16]. Ridge regression is implemented in the output layer of the reservoir that is connected to the predictions.

By optimizing W_{out}, the RC learns to map the complex temporal patterns captured in X_{res} to the corresponding outputs y_{train}. This adaptation ensures that the reservoir network effectively learns the underlying relationships and dependencies within the input data. Typical use cases include time series prediction, pattern recognition, and sequential data analysis, where capturing and leveraging temporal dependencies are essential for achieving high-performance outcomes [2]. For inference, a simple matrix multiplication is used, in which output weights are multiplied by the reservoir state as

$$y(t) = W_{out} \cdot r(t) \tag{3}$$

where the output vector $y(t)$ represents the predicted values or classifications generated by the RC at time t. It encapsulates the RC's output response based on the current reservoir state $r(t)$, aiming to approximate or predict the target outputs associated with the input data [25]. The output weight matrix W_{out}, learned during the training phase, maps the reservoir states $r(t)$ to the predicted outputs $y(t)$. It represents the optimized parameters that enable the RC to transform reservoir activations into meaningful predictions. The reservoir state vector $r(t)$ at time t comprises the activation levels of reservoir nodes determined by the input signals and internal dynamics. These reservoir states encode temporal patterns and features crucial for predicting future outputs or classifications, reflecting the RC's internal representations. The Eq. 3 embodies the prediction phase of RC, where the reservoir states $r(t)$ are transformed into output predictions $y(t)$ using the learned output weights W_{out} [25]. This process aims to generalize the patterns and relationships captured during training to unseen data instances or future time steps. By multiplying W_{out} with $r(t)$, the RC computes $y(t)$ as a linear combination of reservoir activations, leveraging the optimized weights to generate accurate predictions. The prediction $y(t)$ reflects the RC's real-time response based on the current reservoir state $r(t)$ [19], enabling continuous adaptation and prediction refinement over time. The effectiveness of $y(t)$ depends on the quality of W_{out}, which encapsulates the model's ability to generalize learned patterns and make reliable predictions across diverse input scenarios [19]. As Fig. 1 shows, inputs are fed into the reservoir at randomly selected nodes.

Although RC is typically formulated for temporal processing, the proposed formulation for image classification is conceptually related to Extreme Learning Machines (ELM) [17] . Similar to ELMs, the internal network weights are randomly initialized and kept fixed, while only the output layer is trained using linear regression. In contrast to standard ELMs, the recurrent reservoir dynamics are not removed but attenuated by choosing leakage rates of between 0.85 and 0.95, thereby reducing temporal effects while preserving weak dynamical interactions.

In this study, we propose an alternative approach: rather than feeding data points sequentially over time steps, we flatten the entire input into a one-dimensional array and inject it into the reservoir all at once. We aim to make RC as effective for classification tasks as state-of-the-art neural networks. We present experiments to demonstrate the capabilities of our approach and to support our primary claims, including that our method is much faster in training. Additionally, our approach—utilizing a fixed reservoir state and treating input data as a one-dimensional array—achieves high accuracy.

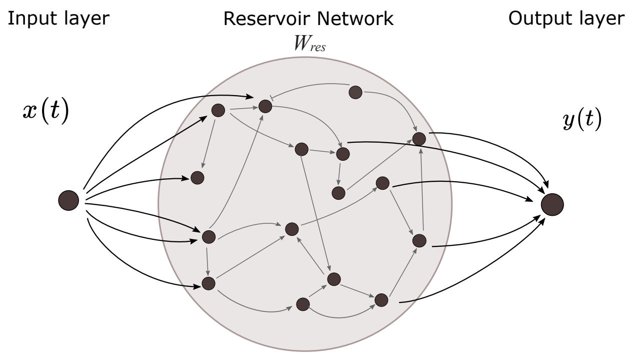

Figure 1 – The architecture of an RC with three components: input, reservoir, and decoders. The input layer receives external signals, $x_1(t), x_2(t), \ldots, x_m(t)$, which are fed into the reservoir, resulting in an internal reservoir state $S(t)$. This state is read by the output layer, producing output signals, $y_1(t), y_2(t), \ldots, y_n(t)$, as the system's response.

Numerous studies have explored the use of the RC framework for image classification.

The paper by Schaetti et al. [28] investigates the use of Echo State Networks (ESNs) as an RC for recognizing MNIST handwritten digits. Their method involves encoding the 28x28 MNIST images as temporal signals and feeding them into ESNs. By transforming each image row into a time-series input, the ESN generates a "memory" of image features, which aids in classification. The authors explore image transformations (such as resizing and rotating) to improve feature extraction by allowing the ESN to view images from varied perspectives. They found that the model performs best with specific parameters—particularly a spectral radius above 1 and a low leaking rate, indicating a slow dynamic suitable for the task. Different output layers were tested, with the "Joined States" (JS) and "Mixed Three States" (MTS) output layers offering richer dynamic insights but at a cost in complexity. The results show that a 1,200-neuron ESN combined with image transformations achieved competitive performance, with an error rate around 4%, using a committee of ESNs for further accuracy. This study provides insights into using ESNs for image classification tasks and highlights the benefits of dynamic and transformed input signals in enhancing recognition accuracy.

Gardner et al. [8] introduce a modified Echo State Network (ESN) designed for static image classification, overcoming the constraints of conventional ESNs when applied to time-independent inputs. Their method splits input images into multiple equal parts, which are processed by parallel reservoirs, allowing neuron states to converge with minimal pre-processing. A Gaussian white noise filter is applied multiple times to each image to improve feature extraction, with the final classification generated after convergence through ridge regression. This approach achieved a 95.3% classification rate on the MNIST dataset with only 10,000 samples and a training time of approximately 5 minutes. The study demonstrates that parallel reservoir processing and noise injection enable ESNs to handle time-independent image data efficiently, providing competitive accuracy with reduced computational resources compared to traditional feed-forward networks.

2 Contents

2.1 Implementation of RC for classification

The main approach in this study is to flatten the image into a one-dimensional vector instead of transforming it into a sequence. For an image of size $h \times w$, the flattened vector $x(t)$ has dimensionality hw (Fig. 2). The reservoir receives this vector as input, transforming the image data directly into the internal state of the reservoir. The flattened layer will then be fed into the randomly initialized reservoir network and transported to the output layer. The output of the model consists of class scores generated by ridge regression, where the predicted class is directly determined by the highest score. This linear approach eliminates the need for softmax normalization while maintaining competitive accuracy.

In standard RC, input data is typically fed into the reservoir sequentially over multiple time steps, allowing the reservoir state to evolve dynamically. However, for tasks like image classification that lack a natural temporal structure, this time-based processing may be unnecessary. Instead, we propose a simplified approach where the entire one-dimensional input vector is provided to the reservoir in a single step. This means the reservoir state is computed once using the full input, which can be expressed as

$$r = \sigma(W_{res} \cdot r_0 + W_{in} \cdot x) \tag{4}$$

where r is the reservoir state as defined in Eq. 1, and r_0 denotes the initial reservoir state. All other variables are as defined in Eq. 1.

Network density is an essential parameter in RC, influencing the structure of the recurrent network. It represents the proportion of possible connections between nodes, indicating the level of interconnectivity within the network. In this study, various densities, including 0.1, 0.25, 0.5, 0.75, 0.9, and 0.99, were explored to analyze their effect on the model's performance. The network was generated using the ‘networkx‘ library [12], with different graph models such as ’Erdős–Rényi’, ’small-world’, and ’scale-free’, which offer diverse connectivity patterns. The reservoir's weight matrix W was derived from the chosen graph, scaled by the spectral radius of W to normalize its eigenvalues, ensuring stability in the network's dynamics. Testing different densities allows us to investigate how varying the network's connectivity influences its memory capacity, nonlinearity, and ability to generalize, ultimately helping identify the optimal configuration for a classification task.

3 Results

3.1 Dataset

Two different datasets were primarily used for training and testing the models. The MNIST dataset [23] is a widely used benchmark in computer vision. It consists of 70K grayscale images of handwritten digits, each sized 28x28 pixels, representing the digits 0 through 9. The dataset is divided into a training set of 60,000 images and a test set of 10K images. MNIST has been essential for evaluating image classification models due to its simplicity, uniformity, and clear distinction between different classes. The COIL-100 dataset [26] is a standard dataset for object recognition tasks in computer vision. It contains 7.2K color images of 100 different objects, where each object was placed on a turntable and photographed from 72 different angles at 5-degree increments. The images are 128x128 pixels in size and depict a variety of everyday items, including tools, toys, and household objects. COIL-100 is widely used to assess the performance of algorithms in tasks like object detection, pose estimation, and 3D object recognition.

3.2 Deep learning models for comparison

To compare the performance of different types of RC, we also evaluate some deep learning models, including CNN, LeNet, small ResNet, and Mini-VGG.

Convolutional neural networks (CNN) [1] are a type of deep learning models designed to process structured grid data, such as images. CNN consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. In CNN, the convolutional layers

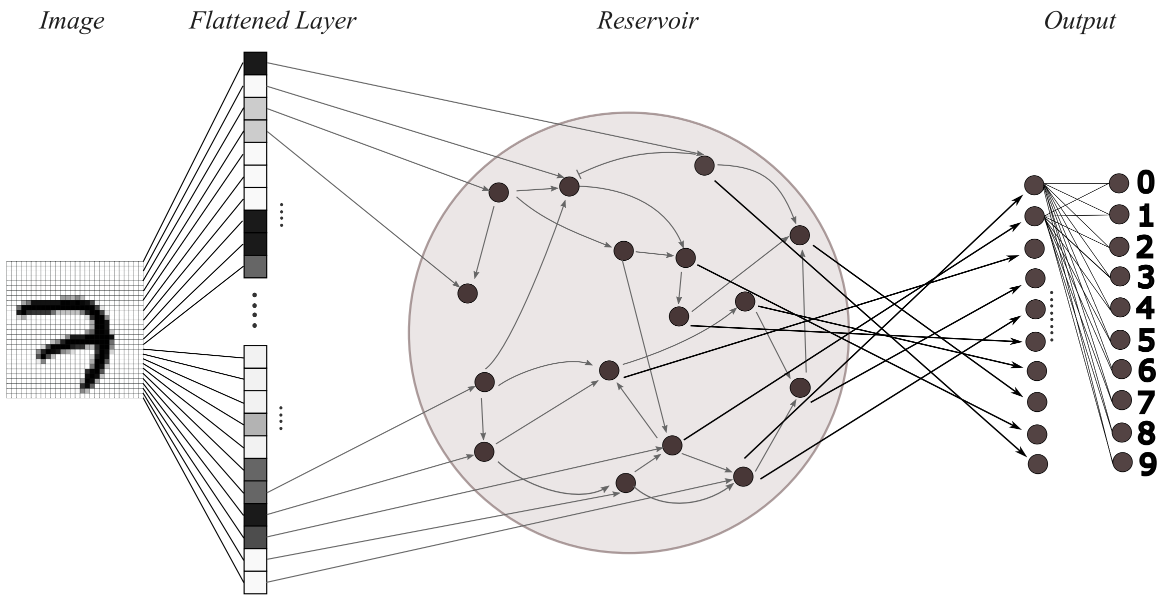

Figure 2 – Flattened layer is used as input to the reservoir. The time-dependent feature of RC is not used here and all the image pixels are fed into the network as a flattened layer.

use filters (also called kernels) to automatically extract features from input data. These features are detected through local receptive fields, where each neuron in the convolutional layer is connected to a local region of the input. After convolution, pooling layers are applied to reduce the dimensionality of the feature maps. Finally, the fully connected layer combines the learned features and computes the final output, often through classification. LeNet-5 [22] is one of the earliest CNN architectures developed for image recognition. It consists of seven layers: an input layer, two convolutional layers (C1 and C3), two pooling layers (S2 and S4), and two fully connected layers (F5 and F6). The network starts with a 32x32 input image, and through convolutional layers, it extracts features using convolutional kernels. These features are then downsampled using pooling layers to reduce the dimensionality. The fully connected layers at the end of the network classify the features into predefined categories. The Mini-VGG network is a smaller version of the popular VGG [29] architecture, designed to be computationally more efficient while maintaining much of the performance characteristics. Unlike VGG-19, which has 19 layers, the Mini-VGG typically has fewer convolutional layers, making it a lightweight model. It consists of a few convolutional layers, each followed by ReLU activation and max pooling operations. The convolutional layers use small 3×3 filters, which help capture fine-grained features in images. A small ResNet (Residual Network) is a compact version of the ResNet [15] architecture, which is designed to address the vanishing gradient problem in deep neural networks by introducing residual connections, or shortcuts, that bypass one or more layers. In a small ResNet, the architecture typically consists of fewer layers compared to larger ResNet models but retains the key feature of residual connections. These residual connections enable the model to learn identity mappings, which helps preserve information and improve the flow of gradients. All models were implemented using PyTorch [27].

3.3 Performance of RC in comparison to deep learning models

We begin by training the RC using three different network types: small-world, scale-free, and Erdős-Rényi networks.

An Erdős-Rényi [7] network is a type of random graph where each pair of nodes has an equal probability p of being connected. This model is defined by two parameters: the number of nodes N and the probability p of creating an edge between any two nodes. As the probability p increases, the network becomes more densely connected, and at a critical value, it undergoes a phase transition where the network becomes fully connected. A small-world network is characterized by a high clustering coefficient and short average path lengths, similar to real-world networks such as social and biological networks. The model was introduced by Watts and Strogatz [30], who proposed a way to generate networks that are highly clustered, like regular lattices, but also have the small diameter typical of random graphs. The network is created by starting with a regular lattice and randomly rewiring some of the edges with a probability p. This rewiring creates "shortcuts" that significantly

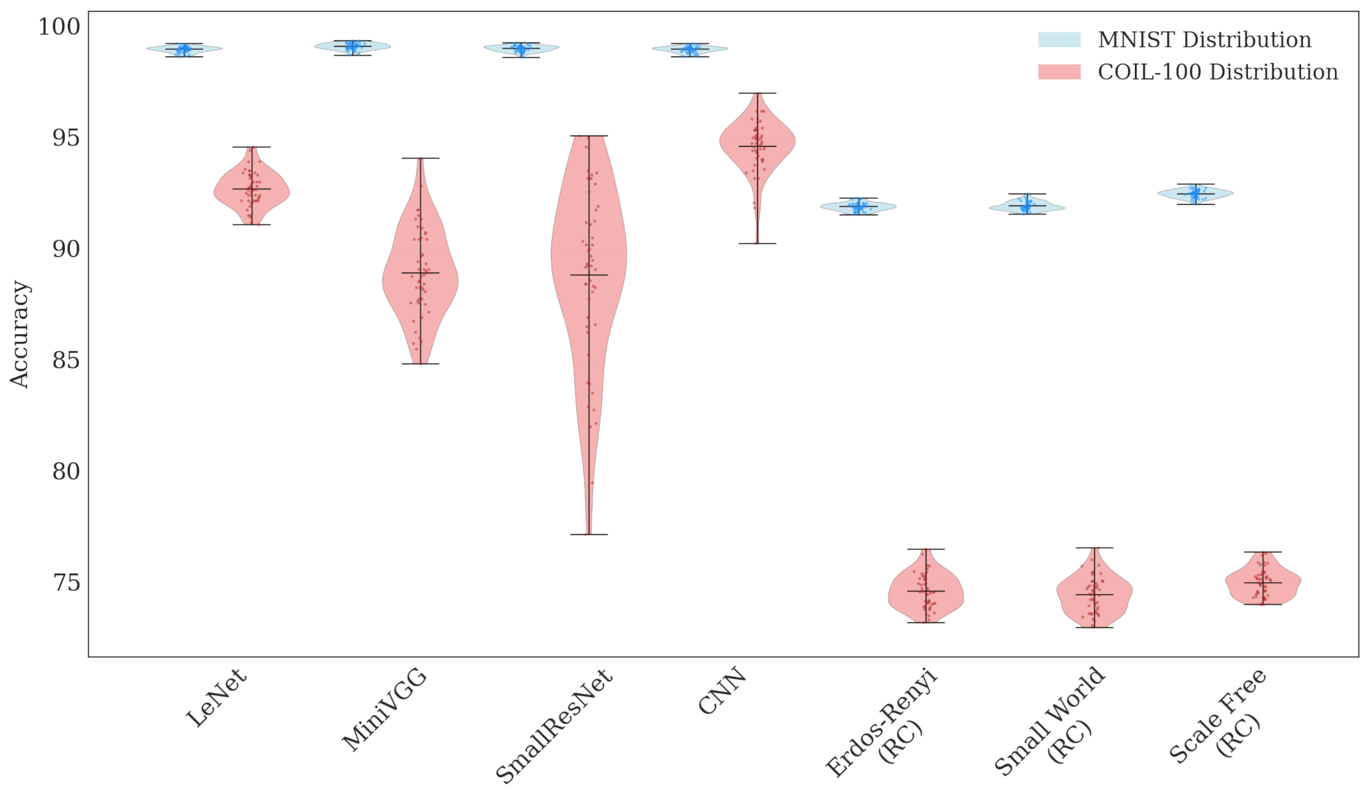

Figure 3 – Comparison of classification accuracy distribution between multiple neural networks and RC on MNIST and COIL-100 datasets. Each RC network type is of size 1000, and a density of 0.9. Each network was tested 30 times to ensure reproducibility, other model details are provided in Table 1.

reduce the average path length without destroying the high clustering. A scale-free network is a type of network where the degree distribution follows a power law, meaning a few nodes have many connections, while most nodes have only a few. The model was introduced by Barabási and Albert [3], who showed that scale-free networks emerge through a process called preferential attachment, where new nodes are more likely to connect to existing nodes with high degrees. This results in a small number of highly connected nodes and a large number of nodes with few connections. These networks were generated using NetworkX: Erdős-Rényi networks with the erdos-renyi-graph function, small-world networks via watts-strogatz-graph, and scale-free networks through barabasi-albert-graph.

We evaluated all models on two datasets: MNIST and COIL-100. Despite RC's simplicity and short training time, RC achieved impressive results, with approximately 90% accuracy on MNIST (Fig. 3) and 70–75% accuracy on COIL-100 as listed in Table 1. For a fair comparison with existing deep learning approaches, we use the deep learning models described in Section 3.2. Deep learning models outperformed RC in classification tasks due to their high capacity for feature extraction. This is particularly noteworthy given that RC was originally developed for time-dependent tasks such as time-series prediction.

3.4 Reservoir structural effect on performance

The density δ of a directed network is defined as the ratio of the number of edges M in the network to the maximum possible number of edges in the network. For a network with N nodes, this is expressed as $\delta = \frac{M}{N(N-1)}$ where M represents the total number of edges in the network. In simple words, the density provides a measure of how sparse or connected a network is. A key observation is that the density δ inversely scales with the number of nodes N, following a hyperbolic relationship. This scaling is influenced by the average degree μ of the nodes in the network, where $M = \mu N$. Substituting this relationship back into the density equation yields $\delta \approx \frac{\mu}{N}$. As networks grow, new nodes add a variable number of edges, affecting the average degree μ. Performance-dependent growth strategies produce sparser networks (lower density) with reduced average degrees [33]. We varied the density to investigate how the connectivity level of the reservoir influences classification accuracy, as different network topologies may respond differently to sparsity or redundancy in connections. As illustrated in Fig. 4, the accuracy improves with increasing reser-

Model	Num. Parameters	Training Time (s)	MNIST Accuracy	COIL-100 Accuracy
LeNet	~ 1.6M	48	98.42%	92.69%
Mini-VGG	~ 2M	67	99.10%	88.93%
Small ResNet	~ 1.85M	51	99.00%	88.41%
CNN	~ 420K	33	98.96%	94.60%
Erdős–Rényi	1000 Nodes	66	91.89%	74.62%
Small-World	1000 Nodes	59	91.92%	74.45%
Scale-Free	1000 Nodes	71	92.47%	74.99%

Table 1 – Average classification accuracies on the test set and training times measured on the training set for neural networks (NN) and RC on MNIST and COIL-100, each evaluated over 50 independent runs for reproducibility. For both datasets, 90% of the data was used for training and 10% for testing. Neural networks were trained on a GPU for faster benchmarking, while RC was executed on a CPU due to more efficient NumPy-based implementations. Despite this, reservoir models achieve competitive test accuracy with significantly shorter training times. A detailed description of the hardware setup is provided in Appendix 5.1.

voir density in scale-free networks, likely due to their enhanced ability to capture diverse input patterns. In contrast, as seen in Fig. 4, small-world and Erdős–Rényi networks show no significant sensitivity to changes in reservoir density.

4 Summary and outlook

In this study, we explored the application of RC for image classification tasks, particularly focusing on the MNIST dataset. Our pixel-wise processing approach achieved promising results with relatively low computational cost, highlighting RC's potential for resource-constrained environments. The reservoir of randomly interconnected nonlinear units proved effective at capturing temporal dependencies, despite the non-sequential nature of the task.

However, RC's pixel-wise approach poses limitations in capturing spatial dependencies within images, which are essential for more complex visual tasks. Future research could investigate hybrid models that integrate RC with convolutional or deep learning architectures to improve spatial feature extraction. Additionally, optimizing the reservoir architecture, such as exploring dynamic or adaptive reservoirs, could enhance model performance further.

Finally, incorporating GPU acceleration through libraries like CuPy, as well as optimizing the reservoir state selection and leveraging recursive least squares methods, could reduce training times and memory requirements. Future work could also include exploring how larger reservoir sizes impact accuracy and further improve the model's scalability in real-time applications.

The comparison of RC with more complex models, such as CNN and LeNet architectures, reveals interesting results. While CNN outperform RC in terms of accuracy, particularly in capturing spatial dependencies in image data, the RC excels in terms of training speed and resource efficiency. RC achieved a solid performance on the MNIST dataset with significantly fewer training durations and lower computational complexity compared to other neural network models, which often require training on GPUs for extended periods.

Our experiments showed that different reservoir network architectures (Erdős–Rényi, small-world, and scale-free) yielded similar performance; however, scale-free networks demonstrated greater robustness, as classification accuracy consistently improved with increased network density. Regarding reservoir size, we found that a range between 700 and 1000 nodes provided the best trade-off between accuracy and training time. Smaller reservoirs trained faster but underperformed in classification accuracy, while larger networks offered only marginal accuracy gains at a significant computational cost compared to smaller networks. While pruning has been successfully applied in time-series RC tasks to simplify large networks [34], its effectiveness in image classification remains underexplored. Nonetheless, it is plausible that pruning techniques—especially those guided by node importance or activity metrics—could reduce redundancy in large random, small-world, or scale-free reservoirs, leading to more compact and efficient models without significant loss in accuracy. Future work should investigate pruning strategies tailored to static image tasks and their impact on spatial feature representation.

Nonetheless, the limitations of RC, particularly its difficulty in capturing spatial relationships due to pixel-wise processing, should be addressed in future work. While RC is effective in tasks that require fast processing and lower computational costs, it may not be the opti-

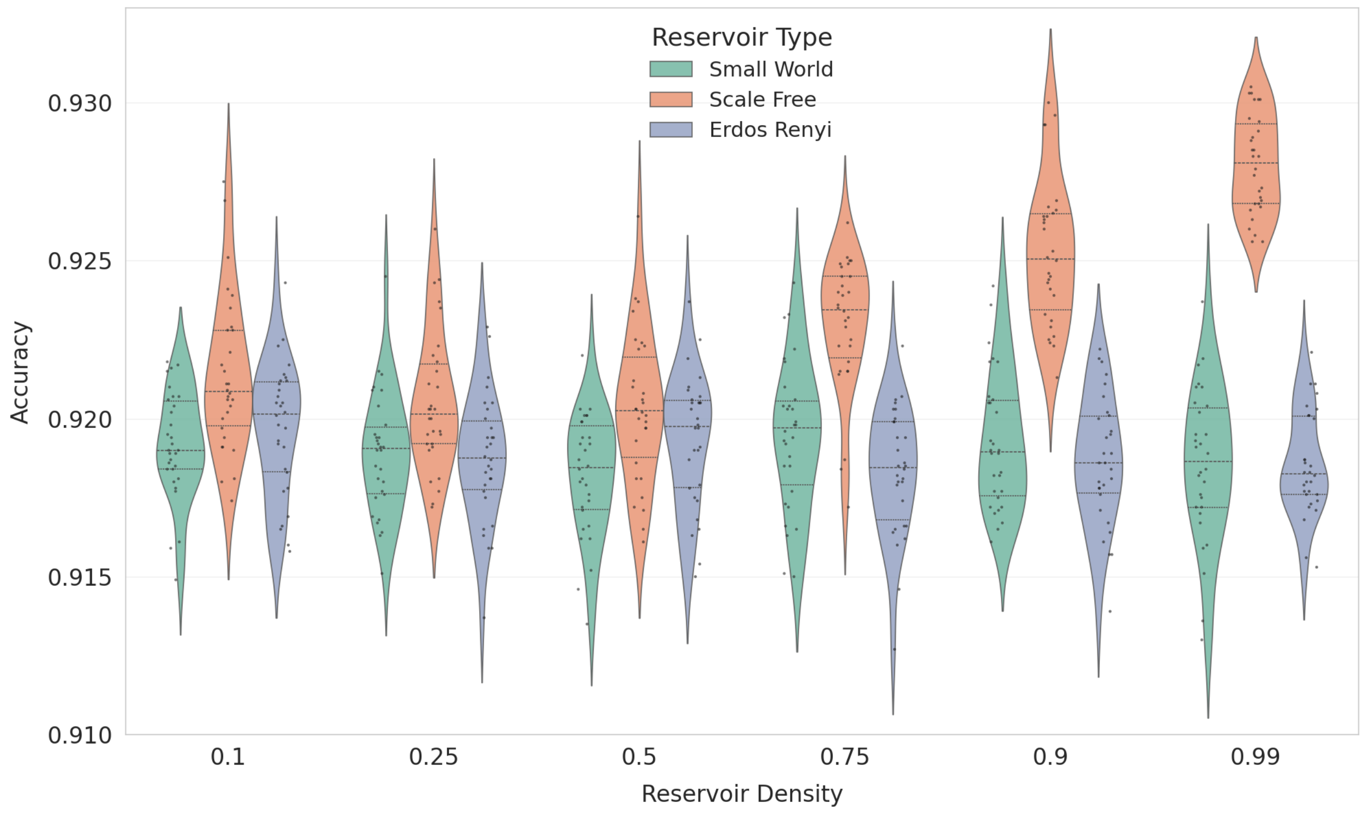

Figure 4 – Classification accuracy of scale-free, small-world, and Erdős–Rényi networks on MNIST for varying network densities. Each density was tested 30 times to ensure reproducibility. After a density of 0.5, scale-free accuracy improves steadily with increasing density.

mal solution for more complex image recognition tasks that require an understanding of the spatial structure within the data.

Finally, optimizing the reservoir configuration, experimenting with different architectures, and leveraging GPU acceleration via libraries like CuPy could further improve the model's scalability and efficiency. Overall, RC demonstrates promising potential in specific domains, but further development is required to compete with more advanced deep-learning models in complex visual tasks.

5 Appendix

5.1 Hardware specifications

In this study, RC was implemented primarily using the Numpy library [13], allowing training to rely solely on the CPU, which is often sufficient given the relatively small size of most RC implementations. The training and inference processes of the RC were carried out on a system equipped with an AMD Ryzen 5 2400G processor with 8 threads and a clock speed of 3.80 GHz, along with 32 GB of system RAM. For training the CNN, LeNet-5, small ResNet, and Mini-VGG models in this study, an NVIDIA Tesla P40 24GB GPU was utilized on the same setup.

Code Availability: The implementation is provided as a jupyter notebook and is publicly available on GitHub [10].

CRediT author statement: Mehdi Ghorbani, Conceptualization, Methodology, Software, Validation, Writing – original draft, Visualization; Manish Yadav, Supervision, Writing – review & editing; Merten Stender, Supervision, Writing – review & editing. The main author (Mehdi Ghorbani) is a student at Technische Universität Berlin. Both supervisors, Manish Yadav and Merten Stender, are listed as co-authors and served as supervising senior researchers in accordance with GAMMAS submission guidelines.

Acknowledgements: Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) supported Manish Yadav under the Special Priority Program (SPP 2353, project number 501847579, led by Merten Stender).

References

[1] I. Sutskever A. Krizhevsky and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, pages 1097–1105, 2012.

[2] T. Akiyama and G. Tanaka. Analysis on characteristics of multi-step learning echo state networks for nonlinear time series prediction. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019. (in press).

[3] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] Büsing, L. Schrauwen, and R. Legenstein. Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons. *Neural Computation*, 22:1272–1311, 2010.

[5] M. Dale, S. O'Keefe, A. Sebald, S. Stepney, and M. A. Trefzer. Reservoir computing quality: connectivity and topology. *Natural Computing*, 20(2):205–216, 2021. ISSN 1572-9796.

[6] Fabrizio Damicelli, Claus C. Hilgetag, and Alexandros Goulas. Brain connectivity meets reservoir computing. *PLOS Computational Biology*, 18(11):e1010639, 2022.

[7] P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

[8] Steven D. Gardner, Mohammad R. Haider, Lee Moradi, and Vladimir Vantsevich. A modified echo state network for time independent image classification. In *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 255–258, 2021.

[9] A. Gaurav, X. Song, S. Manhas, A. Gilra, E. Vasilaki, P. Roy, and M. M. De Souza. Reservoir computing for temporal data classification using a dynamic solid electrolyte zno thin film transistor. *Frontiers in Electronics*, 3, 2022.

[10] Mehdi Ghorbani. Reservoir computing for object classification. `https://github.com/MehdiGhorb/Reservoir-Computing-for-Object-Classification`, 2024. GitHub repository.

[11] Alireza Goudarzi, Peter Banda, Rosara Lakin, Christof Teuscher, and Darko Stefanovic. A comparative study of reservoir computing for temporal signal processing. 01 2014.

[12] Aric A Hagberg, Daniel A Schult, and Pieter J Swart. Exploring network structure, dynamics, and function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA, USA, 2008.

[13] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, and et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[14] Hassan, Li, and Chen. Hardware implementation of echo state networks using memristor double crossbar arrays. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 2171–2177. IEEE, 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[16] R. W. Hoerl. Ridge regression: A historical context. *Technometrics*, 62:420–425, 11 2020.

[17] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[18] T. Hülser, F. Köster, K. Lüdge, and L. Jaurigue. Deriving task specific performance from the information processing capacity of a reservoir computer. *Nanophotonics*, 12(5):937–947, 2023.

[19] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.

[20] H. Jaeger. Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the echo state network approach. Technical Report GMD Report 159, German National Research Center for Information Technology, 2002.

[21] A. Katumba, M. Freiberger, P. Bienstman, and J. Dambre. A multiple-input strategy to efficient integrated photonic reservoir computing. *Cognitive Computation*, 9:307–314, 2017.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. 2010.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[25] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3:127–149, 2009.

[26] S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (coil-100). Technical Report CUCS-006-96, Columbia University, February 1996.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, 2019.

[28] N. Schaetti, M. Salomon, and R. Couturier. Echo state networks-based reservoir computing for mnist handwritten digits recognition. In *International Conference on Computational Science and Engineering*, Paris, France, August 2016.

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. arXiv:1409.1556 [cs.CV].

[30] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[31] Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[32] M. Yadav, S. Chauhan, M. D. Shrimali, and M. Stender. Predicting multi-parametric dynamics of externally forced oscillator using reservoir computing and minimal data. *Nonlinear Dynamics*, 2024.

[33] M. Yadav, S. Sinha, and M. Stender. Evolution beats random chance: Performance-dependent network evolution for enhanced computational capacity. *Physical Review E*, 111:014320, 2025.

[34] Manish Yadav and Merten Stender. Task-specific node pruning enhances computational efficiency of reservoir computing networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 35(8):083123, 08 2025. ISSN 1054-1500.