Image classification using Echo State Networks and CNNs

Jorge Calvimontes
7024517
Model -Driven Deep Learning Lab
Final Project

Abstract

Echo State Networks, a type of neural network commonly used on time series task, has the special feature that it only need to train a small portion of its neurons without the trade off of losing precision or generalization. This feature could be used on other type of tasks to leverage training times and memory usage. This works presents the theoretical background for implementing this type of network, and also touches on the consideration for using it on the image classification task. As a result a TensorFlow model was implemented and a series of experiments were conducted to asses its performance on this type of task.

1. Introduction

Among computer vision one of the most studied task is image classification. Convolutional Neural Networks (CNN) [13] has been widely studied for this task. With many popular models implementing this paradigm [11] [3], [20] [14]. Their success can be attributed to it's ability to extract relevant features of an image thought different type of kernels. As such, in most cases they serve as a feature extractor stage with its output being passed to a classification head, which can be of different type of architectures.

On the other hand, to solve task related to time-series analysis recurrent neural networks, (RNN) are among the most used. Their main idea consist of implementing a memory of past data through recurrent connections. This allows a high dimensional representation of the data. One big challenge for RNNs is their training, given that a gradient based procedure results in the vanishing gradient problem [1]. Regularization techniques have been proposed [2] to alleviate training, which impose constraints for training or limitations to the architecture itself. One of such constrains proposes that a random initialization of the network has a bias towards a finite memory and good generalization properties [2], which implies that training of RNN can be simplified. With that idea, a Echo State Network (ESN) [5] [15] is proposed as a model that maintains finite memory and generalization. Under these conditions, it is sufficient to reduce training complexity to only a single linear output layer. This layer is therefore trained to fit the state (feature space projection) of the ESN to its target. This approach used on time-series data has already been demonstrated [6] [12] [18].

This work aims to combine the two paradigms previously discussed for the task on image classification. More specifically, use a CNN as a preprocessing and feature extraction stage of images and a ESN as a feature map and classification head. The main objective of the work is to assess if such architecture can be implemented effectively to solve the image classification task. For that purpose these specific objectives have been identified:

- Implementation of ESN on TensorFlow.
- Implementation of the proposed architecture for image classification.
- Hyper parameter optimization.
- Experimentation on different datasets.

2. Related Works

Classification of images thought ESN have already been studied on different publication [10] [9]. Also on a phisical sense as Liquid State Machine [1] has been implemented for this task [4]. These works didn't considered a prepossessing stage, thus the raw pixel values were feed to the network. A preprocessing stage was proposed by [16] which consisted of applying different transformations (re framing and re scaling) to images. Finally a preprocessing stage based on CNN was proposed by [19].

In this work we will implement the preprocessing stage proposed by [19] and combine it with the ESN architecture and procedure proposed by [16]. The reason behind

[1]Echo State networks and Liquid state machines both belong to the same paradigm known as reservoir computing. The former is used more on the machine learning realm, while the latter is used more in opto electronics and neuroscience.

this decision, is attributed to the fact that the former author didn't emphasize on the details of the ESN implementation, thus the procedure of feeding the features extracted from the CNN to the ESN are unclear. In contrast the latter author enters in great detail on this procedure and also proposed some optimization which were not considered by the other author.

It's also worth noting that on all the works mentioned the dataset used corresponds to MNIST. It's believed that this dataset is used given that this approach to solve the image classification task is still on an early stage. More challenging datasets will required a well consolidated architecture. This work will try to use a more challenging dataset.

3. Architecture Implementation

3.1. Echo State Network

A basic ESN as proposed by [5] generates a network state x_t at a given time step t; considering an activation function g, the network input matrix $\mathbf{W_{in}}$, input vector u_t, the reservoir matrix $\mathbf{W}$ and previous state of the network x_{t-1}. Formerly:

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

where $x_t \in \mathbb{R}^{N_x}$ representing the activation vector or state at time t, with N_x being the number of neurons on the reservoir. Then $\mathbf{W_{in}} \in \mathbb{R}^{N_x \times N_u}$ is the transformation applied to the input, with N_u as the dimension of the input signal. Finally, $\mathbf{W} \in \mathbb{R}^{N_x \times N_x}$.

The model proposed in 1 can be extended to include a bias:

$$x_t = g(\mathbf{W_{in}}u_t + \mathbf{W}x_{t-1} + \mathbf{W_{bias}}) \tag{2}$$

with $\mathbf{W_{bias}} \in \mathbb{R}^{N_x}$. A common problem on ESN is that the dynamics represented on previous states of the network doesn't adapt to the dynamics of the actual input. In other words, previous states of the network don't have a great influence on the current transformation of the input signal. This problem can be approached using a leaking rate proposed by [7]. Formely:

$$x_t = a(x_{t-1}) + a \cdot g(\mathbf{W_{in}}u_t + \mathbf{W}x_{t-1} + \mathbf{W_{bias}}) \tag{3}$$

with $a \in [0, 1]$. The leaking rate help regulate the influence between these two dynamics. It also depends on the type of signal, reason why it's optimal value can vary depending on the type of task and signal.

3.2. Readout

The output of the ESN consists of a readout layer (output vector) which considers the state of the network at a time step t and makes a linear transformation with an output matrix:

$$\hat{y_t} = \mathbf{W_{out}}x_t \tag{4}$$

where $\mathbf{W_{out}} \in \mathbb{R}^{N_y \times N_x}$ with N_y being the number of readouts (output signals). The training phase of an ESN consist in finding a $\mathbf{W_{out}}$ such that the generated output $\hat{y}$ approximates a target signal y i.e. minimizing the error between both.

In order to find the an optimal output matrix the training procedure consists on collecting all the states of the network $\mathbf{X} \in \mathbb{R}^{N_x \times T}$, and all the target signals $\mathbf{Y} \in \mathbb{R}^{N_y \times T}$, such that:

$$\mathbf{W_{out}}\mathbf{X} = \mathbf{Y}, \tag{5}$$

which can be expressed as:

$$\mathbf{W_{out}}\mathbf{X}\mathbf{X}^T = \mathbf{Y}\mathbf{X}^T.$$

To solve for the output matrix, the inverse matrix must be first calculated:

$$\mathbf{W_{out}} = (\mathbf{Y}\mathbf{X}^T)(\mathbf{X}\mathbf{X}^T)^{-1}.$$

Given that the output matrix is intended to be used beyond the training data, a regularization factor can be applied to overcome over-fitting and have a better generalization. Regularization can applied using Ridge Regression (Thikonov regularization). Thus the training equation becomes:

$$\mathbf{W_{out}} = (\mathbf{Y}\mathbf{X}^T)(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}, \tag{6}$$

where λ is the regularization factor and $\mathbf{I}$, the identity matrix ($\mathbf{I} \in \mathbb{R}^{N_x \times N_x}$)

3.3. Method and conditions

The general methodology for an ESN as introduced by [5] consists of:

1. Generate a random input and reservoir matrix ($\mathbf{W_{in}}, \mathbf{W}$), using a normalized distribution.
2. Initialize with zeros the first state (x_0) and matrices used on the read out ($\mathbf{Y}\mathbf{X}^T, \mathbf{X}\mathbf{X}^T$).
3. For each input signal i to be analyzed ($u_{i,t}, i = 1, .., N_s$), compute the reservoir states on each time step, then collect all the states for each signal on matrix $\mathbf{X}^* \in \mathbb{R}^{N_s \times (N_x \times T)}$.
4. Collect targets for each signal $\mathbf{Y}^* \in \mathbb{R}^{N_s \times (N_y \times T)}$.
5. Compute $\mathbf{Y}\mathbf{X}^T$ and $\mathbf{X}\mathbf{X}^T$ for each signal then sum over the signals.

$$\mathbf{Y}\mathbf{X}^T = \sum_{i=0}^{N_s} \mathbf{Y_i^*}\mathbf{X_i^{*T}}$$

$$\mathbf{X}\mathbf{X}^T = \sum_{i=0}^{N_s} \mathbf{X_i^*}\mathbf{X_i^{*T}}$$

6. Calculate the inverse using a preferred algorithm, and compute the output matrix.

7. Use the trained network on new data.

A condition to ensure that the dynamics of previous states remains thought new states (thus guaranteeing a memory), is to ensure the echo state property of the network. That is, making the spectral radius (maximum eigenvalue) of the reservoir weight matrix have a value below 1. To apply this condition once generated the reservoir weight matrix, it's corresponding spectral radius is calculated and then the matrix divided by this value.

Another condition that sometimes is required consists of making the input and reservoir matrix sparse. That is, removing connections between matrices and their corresponding inputs. To achieve this, random columns and rows are selected and their corresponding cells set to zero. Accordingly the percentage of dead connections is measured as to correlate with a given connectivity percentage.

4. Application on Image Classification

4.1. Convolution as feature extractor

On [16] transformation to the original image were made as a way to extract features and reduce the dimension of the input signal to network. From the training perspective, feeding the reservoir a signal with a low input dimensionality (N_u) signifies less use of memory which can optimize training. For the MNIST dataset, each image was reduced from 28x28 dimension to a 15x15 dimension. In contrast by using a multi layer CNN the output features can have even lower dimensions and preserve important semantics of the original image. On [19] the authors used a 3 layer CNN, which consisted used a 3x3 kernel with stride of 1 and activation function of hyperbolic tangent. In addition between the first and second layer a max pooling operation layers was used; then between the second and third an average polling layer was used, with kernel 3x3 and stride 2x2 accordingly. With this configuration a 28x28 image could be reduced to a dimension of 2x2. As for the number for channels the authors set it to 84, but on this work that will be a variable to be studied. This variable represents the length of each image signal, which means that high values influence training time and memory usage as it will be explain below. It is also worth noting that the convolution layers are not trained and their value is initialized using a random distribution. According to the authors this is enough to obtain relevant features to be interpreted by a ESN.

4.2. From Images to temporal signals

The first mayor clarification that the authors in [16] mention is how to interpret images as signals to the reservoir. It

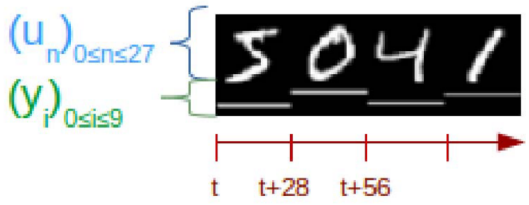

Figure 1. Image as inputs u_n and one hot outputs y_i, image courtesy of [16]

can be considered that each image is a signal to the reservoir, with N_s being the total amount of signals. Then this will imply that for each signal, a reservoir states is calculated. The first suggestion of the authors is to consider these signals as joined horizontally as seen on Fig. 1. Thus for each signal we don't calculate a new reservoir state, instead the last state of a signal becomes the initial on the next one, i.e the reservoir states are not reset for new signals. With this approach the reservoir is capable of capturing more dynamics given that it processes a long signal.

This approach combined with the CNN features will imply first, that the input dimension of the signal (N_y) will be determined by the convolution operations. For a 28x28 image this is reduced to 2x2, which can be flattened to a dimension of 4. Second, the length of each image signal will correspond to the number of output channels generated by the convolutions N_{ch}. This has the consequence that the total number of times steps (T) for the whole training will depend on the number of images (N_i) an the number of channels (N_{ch}). If working with a dataset with great number of images, having an elevated number of channels will increase training time and usage of memory.

4.3. Readout

In order to train the output matrix the target matrix (Y) must be composed of different output channels (N_y) for each time step. For the task of image classification each of these channels can correspond to a particular class, with N_y being the number of classes. On that sense, the target signal for each image i can be interpreted as a matrix $\mathbf{y_i} \in \mathbb{R}^{N_y \times N_{ch}}$ composed of repeated one hot vectors for a particular label. And as a hole, the collection of image signals $\mathbf{Y} \in \mathbb{R}^{N_y \times T}$ with $T = N_i \times N_{ch}$. A representation of the output can be observed in Fig. 1.

An optimization proposed by [16] suggest that instead of training the network to predict a repeated signal of one hot vectors thought the time frame of each image; to predict a single one hot vector by joining the states of the reservoir for such time frame. This implies the collection of reservoir states will expand its dimension $\mathbf{X} \in \mathbb{R}^{(N_x \times N_{ch}) \times N_i}$. This approach will be evaluated on the following sections.

One last consideration for the readout, given that ridge regression is performed throughout the hole collection of target signals. The output vector $\hat{y}$ will not necessarily have a predominant channel with value 1 as the one hot target. In order to evaluate the correctly guessed classes a softmax operation is performed on this vector as to extract the predominant class, given that the dataset used in this work only have one class per image.

4.4. Committee

Given that an ESN is initialized with random weights it is highly possible that the results obtained on different networks with same parameters differs. As such [16] suggests the implementation of committees which consists running the training and prediction of the network a specified number of times (N_c), on each run collecting the predictions of the network ($\hat{Y}$). Then for each image it is calculated the probability that its prediction belongs to a certain class. Formally:

$$Py(C = i) = \frac{1}{N_c} \sum_{c=1}^{N_c} Py_c(C = i)$$

On this work we consider the highest probability for each image.

5. TensorFlow implementation

In order to design the architecture proposed, the implementation consisted of generating a dataset, implementing a CNN based feature extractor, implementing a custom ESN which considers the optimizations exposed above and implementing helping functions such as measurement in order to conduct the experiments.

For the image datasets, fortunately the framework already has wrappers that help download and organize data into training and testing. Further processing was made to extract from training data a validation (corresponding to 20% of training data). Target data was transformed to one hot vectors corresponding to each class. A dataset object was generated which could be processed in batches. The feature extractor based on CNN was relatively easy to implement by using the keras library, given that it already gives all the necessary operations. A sequential model was used in which the convolution and polling operations were staked. The CNN kernels and strides explained before were used. The generated dataset was fed to the sequential model which generates the features used in the ESN.

Although the framework offers a RNN abstraction layer, the ESN implementation did not used it, given that the training procedure differs from the one imposed by this abstraction layer. Either way, the convention of using *call* and *build* methods was maintained. Throughout the network a same datatype is used corresponding to float64. All the matrix and vector operations are implemented using Einstein summation convention (einsum), as this convention simplifies and makes it more clear the dimensions involved in an operation.

To facilitate the training procedure a wrapper class (ESNFlow) was implemented. This receives the features from the CNN, passes them to the reservoir and collects the states. Training is done by preparing the readout (train) as explained in step 5) of Sec. 3.3 and calculating the output matrix (stop training). This wrapper also has a prediction method (call) which prepares new data, calculates the states and outputs a prediction considering softmax.

6. Experiments

Three types of experiments were held. The first one, uses a chaotic time series function in order to prove if the ESN architecture was correctly implemented. The next ones are aimed for image classification, one uses the MNIST dataset and the other the CIFAR-10.

6.1. Narma30

The 30th order narma function is a chaotic function in which its output depend on past outputs an inputs. Formally:

$$y_n = 0.2y_{n-1} + 0.4y_{n-1} \sum_{i=n-30}^{n-1} y_i + 1.5x_{n-29}x_{n-1}, \quad (7)$$

with the input chosen randomly $x = \{x_0, ..., x_n\} \in_R \mathbb{R}$ and $\{y_0, ..., y_{30}\} = 0$. Given that in order to learn the dynamics of the function it is necessary a memory, it is a good candidate to test if the ESN was correctly implemented. For this experiment 10 different signals were used, 9 of them for training and one for testing. For training, each signal was feed to the network without using joined states and resetting the reservoir states after each signal was processed. The reservoir parameter used are similar to the ones used by [17]. The Normalized Mean Root Square Error was measured during training and testing.

6.2. MNIST

For this experiment the training and validation datasets of MNIST is used to evaluate the impact of each hyper parameter on the network. A base configuration is set based on the reviewed literature. From this configuration each new experiment changes the values of one particular hyper parameter in order to assess the overall impact it caused. The percentage of wrongly guesses classes is used as a measure. A committee of 5 is used throughout all the experiments. Finally the best configuration for each hyper parameter is evaluated and used on a testing dataset.

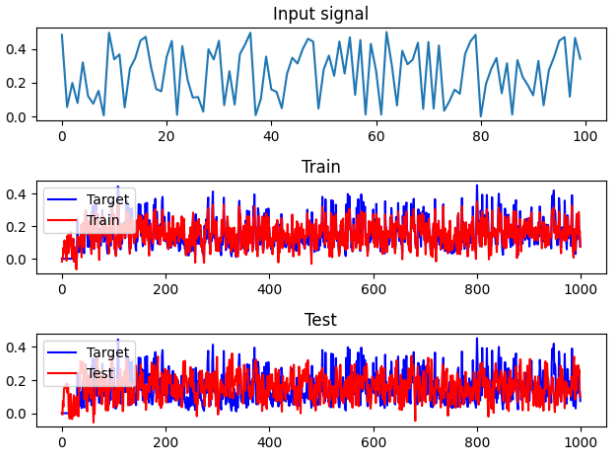

Figure 2. Approximation for the Narma30 function

6.3. CIFAR-10

Experiments on this dataset will not focus on hyper parameter optimization, the best configuration for MNIST will be used. The same metrics used before will be employed for testing and training data. The aim of these experiment is to assess if there is any impact on the performance of the network by changing from a gray scale image to a RGB one. Considering that the features obtained from the CNN will be equal for both dataset, it is believed it shouldn't have a great impact.

7. Results

7.1. Narma30

On Fig. 2 it can be shown a sample signal of the Narma30 function and how the ESN approximates on a training and testing signals. The procedure for obtaining results consisted on training the network, predicting on training and test signal, measuring the NMRSE for both and repeating the experiment 10 times. Then the mean NMRSE was measured across the 10 repetitions. The obtained error were 0.46 and 0.476 for training and testing respectively. These results are similar to the ones obtained by [17] on the same task, with similar parameter. This indicates that the implementation of the ESN was done correctly.

7.2. MNIST

An initial experiment considered the impact without joining states, the model yielded an error of 0.80 and 0.82 respectively for training and validation. Then, using joined states, the performance improved, reason why it was considered as part of the base experiment.

The parameters used in the base experiment are shown Tab. 1. For each of these parameters variants where made,

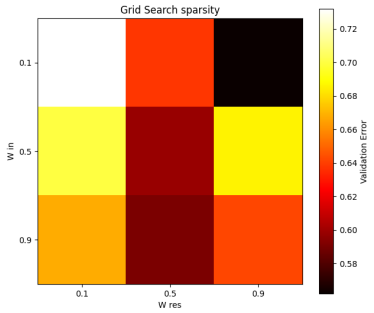

Figure 3. Grid search for sparsity between Win and W

Parameters	Value
reservoir size	200
input scaling	0.05
win sparsity	0.0
w sparsity	0.0
leaky rate	0.0
channels	15
joined states	True

Table 1. Base parameters

Parameter	Best Value	Train error	Validation error
Reservoir size	100	0.651	0.680
Input scaling	0.6	0.64	0.533
Spectral Radius	0.99	0.66	0.62
Bias scaling	1	0.622	0.652
Sparsity	win=0.1, w = 0.9	0.675	0.562
Leaky rate	0.2	0.675	0.663
Channels	20	0.631	0.553

Table 2. Best parameters

	Train Error	Test Error
Base	0.655	0.695
Optimized	0.576	0.528

Table 3. Comparison training and testing

Tab. 2 shows the best performance achieved (with respect to validation error) for a particular variation and its corresponding train and validation error. For the case of sparsity a grid search of both variables was performed as it the literature points they have a strong correlation. A graph of the grid search is shown in Fig. 3. Finally results for training and testing between the base and optimized network are shown in Tab. 3. It can be observed that with hyper parameter optimization there is improvement in comparison to the base. Although the overall performance is poor.

7.3. CIFAR-10

For this experiment no hyper parameter optimization was made, the network was trained using the best configuration for MNIST. The errors obtained are 0.74 and 0.822 for training and testing respectively. It can be these errors have increased in comparison of the results obtained for MNIST. This may indicate that this type of dataset required further preprocessing in order for the ESN to capture the dynamic of their features. One possible solution can be to change the CNN for this dataset, either by adding more layers or changing the kernel and stride parameters. Or most likely, increasing the number of channels.

8. Discussion

Based on the results presented on the previous sections, it can be claimed that the implementation of the ESN for a time series task was correct, but it's implementation for image classification didn't yield expected results. For a time series task, the ESN was able to approximate chaotic signals and yield a similar results presented in the literature. This indicates that the implemented ESN architecture works as expected for this type of task.

The two main references [16] [19] for image classification using ESN claim to obtain percentage errors below 8%. For our case the lowest error was 52.3% which presents a great difference. It is believed that the main cause of this difference is caused by the number of channels used on the feature extractor and the reservoir size. The authors that implemented the CNN aprroach used 84 channels with a reservoir size of 8000. In our case the maximum number of channels were 30 and a reservoir size of 300. Increasing these parameters further results in out of memory errors. This can be attributed to the fact that the architecture was designed considering the time series approach, in which the number of signals rarely exceed 100. For image classification, given that each image represents a signal, a dataset of more than 10.000 entries (the case for MNIST and CIFAR10) represent a huge memory load. An architecture which outputs states and updates the readout in batches is a better and more wise approach.

Regardless of the memory usage it was also observed that the time consumed for calculating all the reservoir states and training the readout, took an average of 5 min for a image signal composed of 15 channels. By increasing the number of channels the time to compute increases linearly. Further, if implementing the committee approach the time consumed can increase exponentially. For this reason using an output channel of 84 and a committee of 100 as the authors consulted did, could results on a training that can last days.

A final observation considers the way the output signal is interpreted. On this work a softmax transformation was used in which the class with highest probability value is used. Both authors don't mention the way they interpret the output of the network. Which leaves space to many interpretations.

9. Future Work

The main optimization that will be implemented on future work is the processing of reservoir and readout data through batches. This will allow to increase the reservoir size and the number of output channels of the CNN. Which ultimately can lead to more precise results. Another optimization entails parallelization given that the consumption of cpu (or gpu) is very low during states calculation and only high while calculating the output matrix (due to the inverse). This could easily be achieved using queue jobs. In regards to the hyper parameters, it was proven that they present an influence on the performance on the network so future work can consider finer hyper parameter optimization.

Other ideas consider, the use optimized ESN architectures like intrinsic plasticity [17] or multi stage reservoir [8]. Trying other types of feature extractors, like deep CNNs or auto encoders can also have promising results.

10. Conclusions

Throughout this work an overview the Echo State Network has been presented. This reflect the consideration and design details taken through the implementation of a TensorFlow model of this type of network. Further considerations have been presented for implementing this type of network on the task of image classification. Experiments to test the performance of the network on the tasks of time series prediction and image classification, were held.

Although the proposed objectives were completed, it cannot be claimed that the main objective was completed given that the results obtained show poor performance. The main cause and possible solution have been discussed. Thus it can be concluded, for now, that the architecture implemented has the potential to solve the image classification task. It has been proven that hyper parameter optimization has an impact on the performance of the network, thus further work needs to be done to completely assess the performance of the implemented model on image classification.

References

[1] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. 1

[2] Barbara Hammer and Jochen J Steil. Tutorial: Perspectives on learning with rnns. In *Proc. ESANN*, pages 357–368. Citeseer, 2002. 1

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[4] Michiel Hermans, Miguel C Soriano, Joni Dambre, Peter Bienstman, and Ingo Fischer. Photonic delay systems as machine learning implementations. 2015. 1

[5] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001. 1, 2

[6] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004. 1

[7] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352, 2007. 2

[8] Azarakhsh Jalalvand, Kris Demuynck, Wesley De Neve, and Jean-Pierre Martens. On the application of reservoir computing networks for noisy image recognition. *Neurocomputing*, 277:237–248, 2018. 6

[9] Azarakhsh Jalalvand, Kris Demuynck, Wesley De Neve, Rik Van de Walle, and Jean-Pierre Martens. Design of reservoir computing systems for noise-robust speech and handwriting recognition. In *Proceedings of the 28th Conference on Graphics, Patterns and Images (accepted in the Workshop of Theses and Dissertations (WTD)), Sociedade Brasileira de Computaçao, Salvador, Brazil*, pages 26–29, 2015. 1

[10] Azarakhsh Jalalvand, Glenn Van Wallendael, and Rik Van de Walle. Real-time reservoir computing network-based systems for detection tasks on visual contents. In *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, pages 146–151. IEEE, 2015. 1

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 1

[12] Laurent Larger, Miguel C Soriano, Daniel Brunner, Lennert Appeltant, Jose M Gutiérrez, Luis Pesquera, Claudio R Mirasso, and Ingo Fischer. Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. *Optics express*, 20(3):3241–3249, 2012. 1

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[14] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022. 1

[15] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002. 1

[16] Nils Schaetti, Michel Salomon, and Raphaël Couturier. 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)*, pages 484–491. IEEE, 2016. 1, 3, 4, 6

[17] Benjamin Schrauwen, Marion Wardermann, David Verstraeten, Jochen J Steil, and Dirk Stroobandt. Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71(7-9):1159–1171, 2008. 4, 5, 6

[18] Chunyang Sheng, Jun Zhao, Ying Liu, and Wei Wang. Prediction for noisy nonlinear time series by echo state network based on dual estimation. *Neurocomputing*, 82:186–195, 2012. 1

[19] Zhiqiang Tong and Gouhei Tanaka. Reservoir computing with untrained convolutional neural networks for image recognition. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 1289–1294. IEEE, 2018. 1, 3, 6

[20] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 1