An ESN approach for Audio Classification in Construction Sites

Neural Network Project - La Sapienza

Edoardo Bini, Marco Ferraro, Alessandro Giannetti

25/05/2020

Abstract

Echo State Networks (ESN) are very simple to implement and are readily provided by many libraries on the network. The aim of the project is to test whether ESNs can be useful tools for the audio data classification problem. For this purpose we developed an application for the classification of different types of construction vehicles and tools, through an Echo State Network. The proposed approach consists in splitting the audio data into fragments and sampling them into spectrogram representation to then be classified into the different types of machinery. This approach exhibited great potential in environmental sound classification (ESC) achieving significant accuracy.

1 Introduction

Audio classification has been a growing field lately since better and cheaper sensors are becoming widely available. The amount of applications for this technology has already proven to be staggering, from events monitoring to speech recognition the amount of practical uses keeps growing [1]. In an effort to continue exploring the use of machine learning techniques for environmental sound classification, our contribution focuses on the implementation of a Recurrent Neural Network (RNN) [2] to classify active machinery in construction sites.

In detail, this paper will describe our first approach at implementing an Echo State Network using a plethora of audio data collected on-site to establish if this technology can be effective in the recognition of different kinds machinery sounds.

2 Proposed approach

The methodology we adopted consists in transforming the audio data into their visual representation through the use of spectrograms and proceeding to classify them using an Echo state network.

2.1 ESN generalities

Our efforts focus on Echo State Networks, which provide a simplified approach to Recurrent Neural Networks by initializing the recurrent layer with the use of randomization processes [3]. Our problem is supervised, therefore for any given input signal $u(n) \in R^{N_u}$ a target output label $\mathbf{y}^{target}(n) \in R^{N_y}$ is provided. The task is to train a model such that its output $\mathbf{y}(n) \in R^{N_y}$ matches the real labels as often as possible, minimizing the error and, more importantly, that is able to conserve the same capabilities on unseen data. The error is commonly the Mean-Square Error (MSE) like the RMSE:

$$E(y, y^{target}) = \frac{1}{N_y} \sum_{i=1}^{N_y} \sqrt{\frac{1}{T} \sum_{n=1}^{T} (y_i(n) - y_i^{target}(n))^2} \tag{1}$$

which is also averaged over the N dimensions i of output here. ESNs use an RNN type with leaky-integrated discrete-time continuous-value units. The typical update equations are:

$$\widetilde{\mathbf{x}}(n) = tanh(\mathbf{W}^{in}[1; \mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)) \tag{2}$$

$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\widetilde{\mathbf{x}}(n) \tag{3}$$

where $\mathbf{x} \in R^{N_x}$ is a vector of reservoir neuron activation and $\widetilde{\mathbf{x}}(n) \in R^{N_x}$ is its update, $\mathbf{W}$ and $\mathbf{W^{in}}$ are the input and recurrent weight matrices and α is the leaking rate. The linear readout layer is defined as:

$$\mathbf{y}(n) = \mathbf{W}^{out}[1; \mathbf{u}(n); \boldsymbol{x}(n)] \tag{4}$$

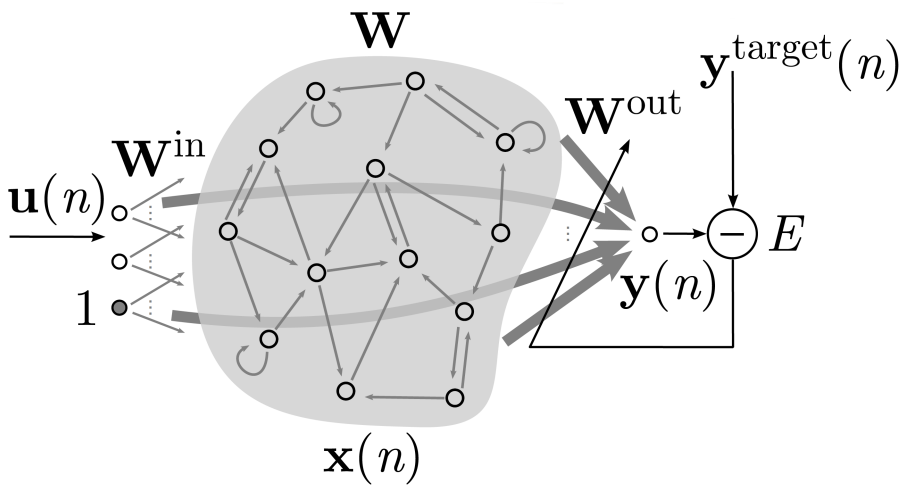


Fig.1 An echo state network

The original method of reservoir computing (RC) introduced with ESNs is the following:

1 generate a large random reservoir RNN($\mathbf{W^{in}}$, $\mathbf{W}$, α);

2 run it using the training input $\mathbf{u}(n)$ and collect the corresponding reservoir activation states $x(n)$;

3 compute the linear readout weights $\mathbf{W^{out}}$ from the reservoir using linear regression, minimizing the MSE between $\mathbf{y}(n)$ and $\mathbf{y}^{target}(n)$

4 use the trained network on new input data $\mathbf{u}(n)$ computing $\mathbf{y}(n)$ by employing the trained output weights $\mathbf{W^{out}}$.

In an ESN, an important component are the reservoirs. They act as a nonlinear expansion and as a memory input at the same time. A reservoir can be seen as a nonlinear high-dimensional expansion x(n) of the input signal u(n) and it can serve as a memory, providing temporal context. These aspects can give us a rich and relevant enough space in x(n), such that $\mathrm{y}^{\mathrm{target}}$ (n) could be obtained by linear combination. A reservoir is defined as (W^{in}, W, α): The input and the recurrent connection matrices are generated randomly according to some parameters, the leaking rate is selected as a free parameter itself. The parameters are: the size $\mathrm{N_x}$, sparsity, distribution of nonzero elements, the spectral radius of W, scalings of W^{in} and the leaking rate. The general wisdom is that the bigger the reservoir, the better the obtainable performance, provided appropriate regularization measures are

taken against over-fitting. The scaling of the input weight matrix W^{in} is another key parameter to optimize in an ESN. For uniformly distributed W^{in} we usually define the input scaling "a" as the range of the interval [-a, a] from which values of W^{in} are sampled; for normal distributed input weights one may take the standard deviation as a scaling measure. The leaking rate α of the reservoir nodes in can be regarded as the speed of the reservoir update dynamics discretized in time. We can describe the reservoir update dynamics in continuous time as an Ordinary Differential Equation (ODE)

$$\dot{\mathbf{x}} = -\mathbf{x} + tanh(\mathbf{W}^{in}[1; \mathbf{u}] + \mathbf{W}\mathbf{x}) \tag{5}$$

If we make an Euler's discretization of ODE in time, taking

$$\frac{\Delta \mathbf{x}}{\Delta t} = \frac{\boldsymbol{x}(n+1) - \mathbf{x}(n)}{\Delta t} \approx \dot{\mathbf{x}} \tag{6}$$

we arrive at exactly (up to some time indexing conventions) the discrete time equations with α taking the place of the sampling interval delta t. Thus α can be regarded as the time interval in the continuous world between two consecutive time steps in the discrete realization. The most important parameters are: input scalings, spectral radius, leaking rate. These three parameters are very important for a good performance and are quite task-specific. The reservoir size N_x almost comes as an external restriction, and the rest of the parameters can be set to reasonable default values: reservoir sparseness, weight distribution, or details of the model. It is still worth investigating several options for them, as a lower priority. As explained before, the performance can also be additionally improved in many cases by "splitting" a single parameter into several. Setting different scalings to the columns of W^{in} (corresponding to the bias input and possibly to different dimensions of input if they are of different nature) can go a long way. Also, setting leaking rates α differently for different units (e.g., by splitting them to several sub-populations with constant value) can help a lot in multi-timescale tasks [3].

2.2 Spectrogram extraction

To extrapolate a spectrogram from an audio signal, the Python library "Librosa" offers multiple functions that were used to calculate the mel-scale spectrograms [4][5]. All frames are sampled to 44,100Hz, with a window of size 1024, hop-size of 512 and 60 mel-bands. For every 30ms audio frame, a spectrogram of 60 rows and 3 time buckets is produced.

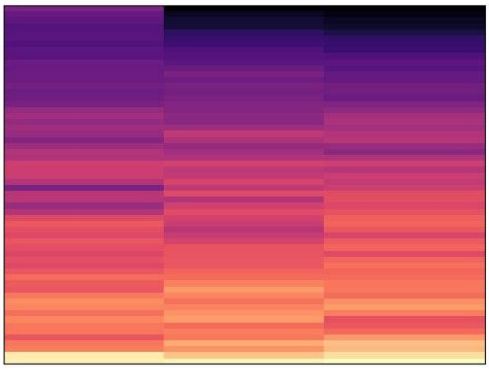


Figure 1: log-mel-spectrogram of a single 30ms segment of audio

3 Implementation

3.1 Utah audio data

The data-set used for this experiment is the Utah Audio Data. The audio contained in the data-set has been collected from different construction machines and equipment in real working scenarios, which implies a lot of background noise generated from workers, weather, environment etc. Since this data-set has already been used for audio classification [6], we chose to use the same classes that had been used for previous research. As a result, these are the classes that were considered: Backhoe JD50D Compact, Compactor Ingersoll Rand, Concrete Mixer, Excavator Cat 320E, Excavator Hitachi 50U. These classes provide the least noisy data available for more accurate results.

3.2 Data preprocessing

To avoid using the same data for training and testing, the data-set was split into two separate categories, one for each purpose. The data was then augmented by splitting each audio file into 30 milliseconds audio segments, each of which overlaps the subsequent one by 15 ms. We then compute the Root Mean Square (RMS) of every signal of these frames, and drop the ones with too small power with respect to the average RMS of the different segments, in order to remove the frames which contain mostly silence.

Afterwards, the dataset is balanced by taking N samples for each class, where N is the number of elements contained in the class with the least amount of samples. Balancing the dataset ensures that no bias can be present towards the classification of any class of machinery. Using the Python library Librosa, we extract the waveform of the audio tracks from the audio samples and, using the same library, we generate the log-scaled mel-spectrogram of the signal. The input of the ESN is a Numpy Array concatenating the values of the three time buckets of the spectrogram, one mel-band at a time. The labels corresponding to each segment consist of the one-hot-encoding representation of the specific class, also in the form of a Numpy Array.

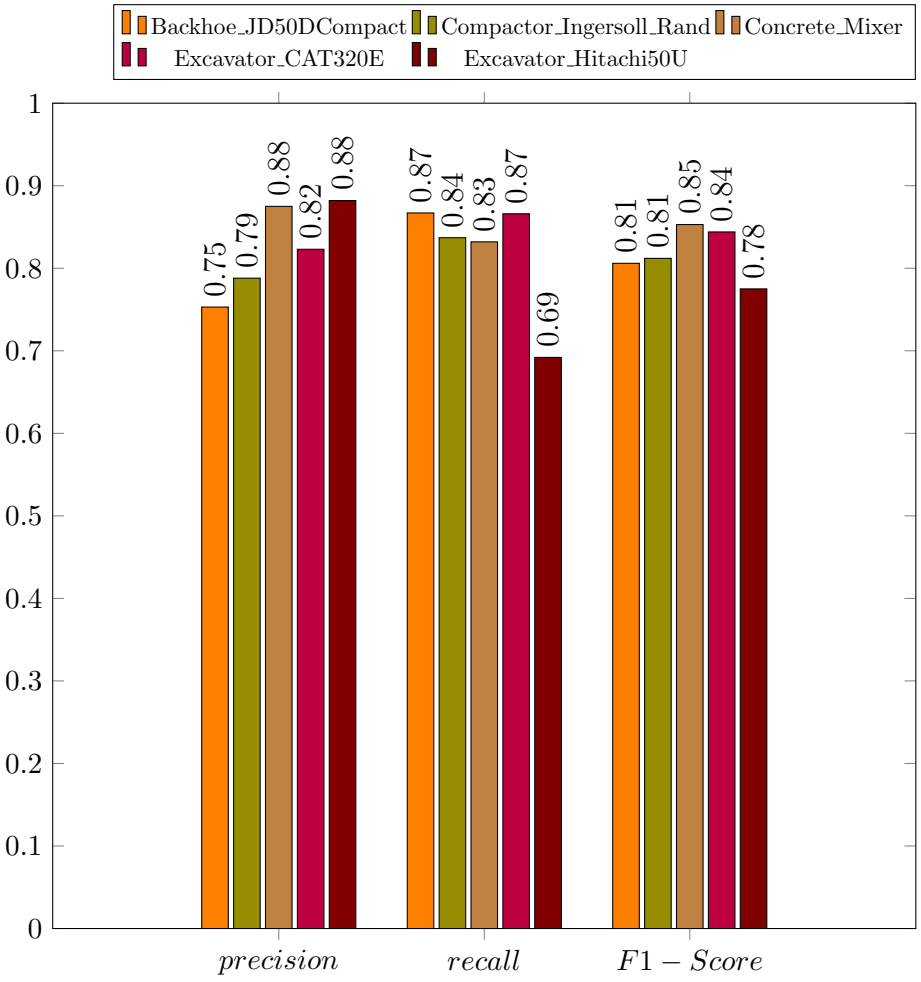
3.3 Memory usage

Our practical implementation of the network makes use of the "EasyESN" library which, like most available ESN libraries, loads the entire training data-set into memory before starting the training process without possibility of batch training [7]. Given the reservoir data expansion, our training was forced into a trade-off between the sizes of the training data and the reservoir. Through empirical research, we found that 300 audio segments for each class (a total of 1500 audio samples from the training set) and a reservoir size of 1200 were the most performing settings we could adopt for this particular data-set and application. We also established that 0.1 was the best value for the leaking rate of the RC.

Because of this severe limitation in our approach, the obtained results of this research might be improved by a significant amount by loading the training data in batches using custom-made ESN applications. Therefore, further research is highly encouraged and, in fact, needed.

4 Numerical Results

After training the classification model, we tested its performance metrics on 150,000 audio segments:



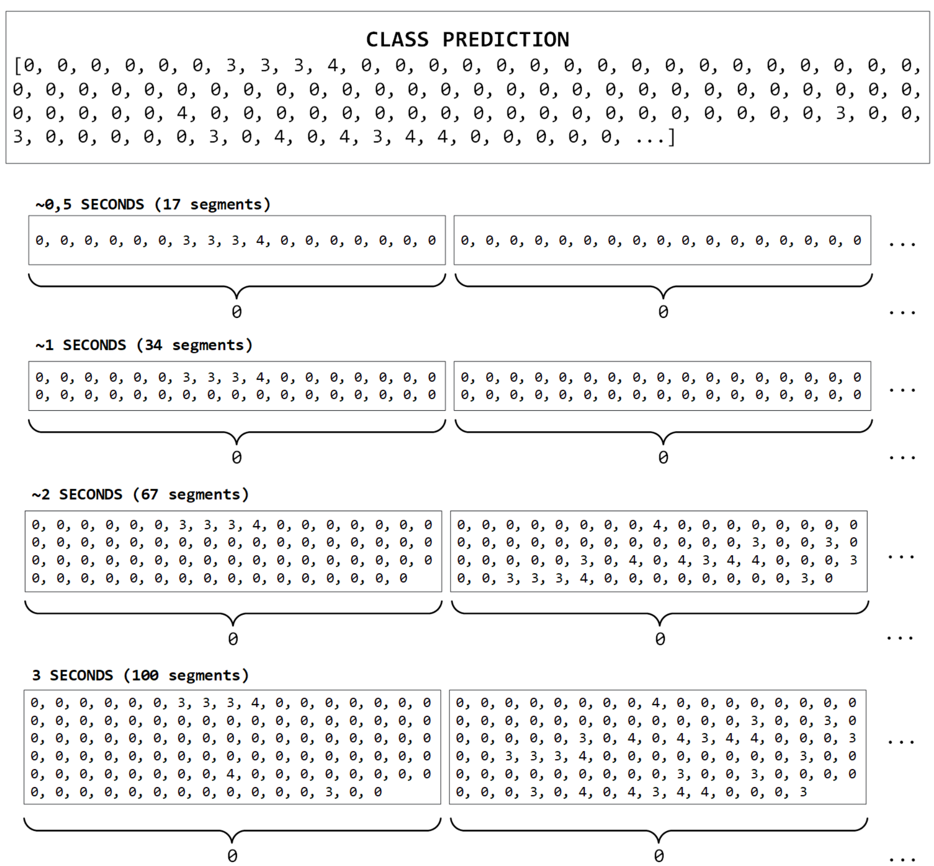
	class 1	class 2	class 3	class 4	class 5
class 1	26019	451	204	1157	2169
class 2	572	25119	1903	2214	192
class 3	296	3217	24977	1376	134
class 4	1597	954	1185	26004	260
class 5	6047	2123	258	810	20762

These results show an overall accuracy of 81.92%, an average precision of 82.48%, an average recall of 81,92% (equal to the overall accuracy since the data-set is perfectly balanced) and an average F1-score of 0,8185 [8].

Accuracy	81.92%	122881/150000
Average Precision	82.48%	
Average Recall	81.92%	
Average F1-Score	0.8185	

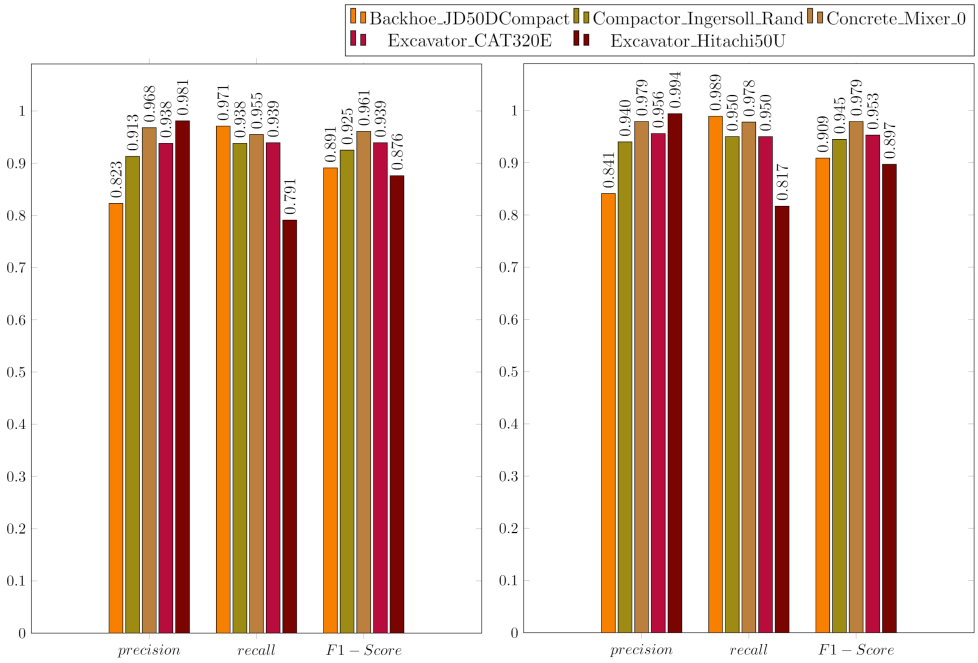
4.1 Majority voting

Depending on the practical application of this technology, it might be possible to augment the capabilities of the classifier by implementing a majority voting system [9].



Given that the classifier evaluates audio every 30 ms, one might consider to wait for the classification of multiple samples before judging the class affiliation to improve the results. In non-instantaneous applications for example, it might be possible to classify data after a few seconds, and even in the case of instantaneous applications, a small delay of 500ms might be acceptable.

These are some of the results for various amounts of time:

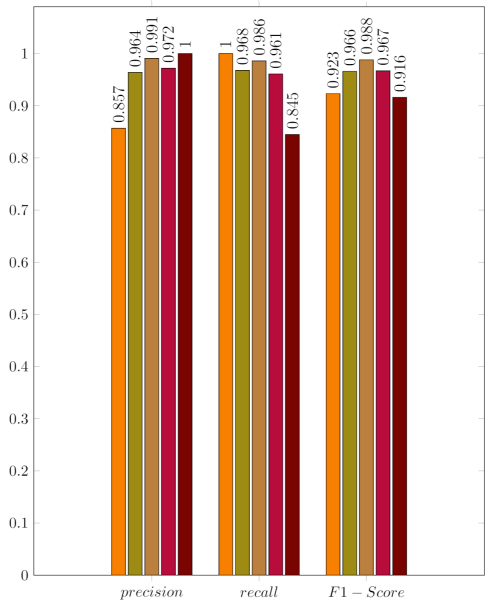


Majority voting every ∼0.5 seconds (17 segments)

Accuracy:	91.9%	8107/8820
Average Precision:	92.5%	
Average Recall:	91.9%	
Average F1-Score:	0.918	

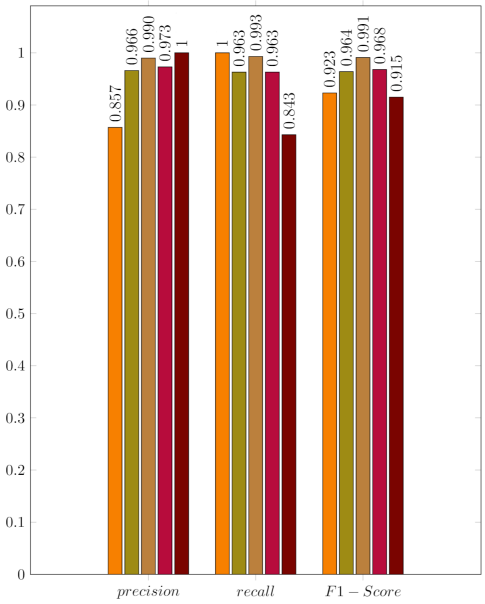
Majority voting every ∼1 second (34 segments)

Accuracy:	93.7%	4133/4410
Average Precision:	94.2%	
Average Recall:	93.7%	
Average F1-Score:	0.936	



Majority voting every ∼2 seconds (67 segments)

Accuracy:	95.26%	2129/2235
Average Precision:	95.72%	
Average Recall:	95.26%	
Average F1-Score:	0.952	



Majority voting every 3 seconds (100 segments)

Accuracy:	95.26%	1429/1500
Average Precision:	95.74%	
Average Recall:	95.27%	
Average F1-Score:	0.952	

These numerical results do not represent the general capability of the majority voting system since the amount of samples isn't large enough to guarantee statistical significance, but they are still indicative of a substantial benefit on every metric of performance.

5 Conclusions and future work

ESNs demonstrated a remarkable versatility by showing their potential in the audio data recognition field, especially when considering the data-set and memory constraints of this particular application. There is virtually no doubt that this technology could be used in other practical scenarios and we encourage their research.

6 References

1. M. Lukosevicius and H. Jaeger, "Overview of reservoir recipes", School of Engineering and Science, Jacobs University, Technical Report No.11, 2007.

2. Danilo P. Mandic, Jonathon A. Chambers, "Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability", August 2001

3. Mantas Lukosevicius, "A Practical Guide to Applying Echo State Networks", Jacobs University Bremen

4. "Large-Scale Weakly Supervised Audio Classification Using Gated Convolutional Neural Network", Yong Xu, Qiuqiang Kong, Wenwu Wang, Mark D. Plumbley 2018 IEEE International Conference on Acoustics

5. Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, Oriol Nieto "librosa: Audio and Music Signal Analysis in Python", Proc. of the 14th Python in Science Conf. (SCIPY 2015)

6. Alessandro Maccagno, Andrea Mastropietro, Umberto Mazziotta, Michele Scarpiniti, Yong-Cheol Lee, and Aurelio Uncini "A CNN Approach for Audio Classification in Construction Sites", Sapienza University of Rome, https://github.com/AndMastro/WreckingNet

7. EasyESN, https://github.com/kalekiu/easyesn

8. E. Alpaydin, Introduction to Machine Learning, Mit Press, 3rd Ed., 2014.

9. D Ruta, B Gabrys, Information fusion, Elsevier 2005.