electronics

MDPI

Article

Online Learning of Oil Leak Anomalies in Wind Turbines with Block-Based Binary Reservoir

Matteo Cardoni [1,2], Danilo Pietro Pau [1,*], Laura Falaschetti [2], Claudio Turchetti [2] and Marco Lattuada [3]

[1] System Research and Applications, STMicroelectronics, Via C. Olivetti 2, I-20864 Agrate Brianza, Italy; matteocardo@gmail.com
[2] Department of Information Engineering, Università Politecnica delle Marche, Via Brecce Bianche 12, I-60131 Ancona, Italy; l.falaschetti@univpm.it (L.F.); c.turchetti@univpm.it (C.T.)
[3] System Research and Applications, STMicroelectronics, Via Tolomeo 1, I-20007 Cornaredo, Italy; marco.lattuada@st.com
* Correspondence: danilo.pau@st.com

Abstract: The focus of this work is to design a deeply quantized anomaly detector of oil leaks that may happen at the junction between the wind turbine high-speed shaft and the external bracket of the power generator. We propose a block-based binary shallow echo state network (BBS-ESN) architecture belonging to the reservoir computing (RC) category and, as we believe, it also extends the extreme learning machines (ELM) domain. Furthermore, BBS-ESN performs binary block-based online training using fixed and minimal computational complexity to achieve low power consumption and deployability on an off-the-shelf micro-controller (MCU). This has been achieved through binarization of the images and 1-bit quantization of the network weights and activations. 3D rendering has been used to generate a novel publicly available dataset of photo-realistic images similar to those potentially acquired by image sensors on the field while monitoring the junction, without and with oil leaks. Extensive experimentation has been conducted using a STM32H743ZI2 MCU running at 480 MHz and the results achieved show an accurate identification of anomalies, with a reduced computational cost per image and memory occupancy. Based on the obtained results, we conclude that BBS-ESN is feasible on off-the-shelf 32 bit MCUs. Moreover, the solution is also scalable in the number of image cameras to be deployed and to achieve accurate and fast oil leak detections from different viewpoints.

Keywords: anomaly detection; binary quantization; block-based processing; echo state networks; micro-controller; online learning

Citation: Cardoni, M.; Pau, D.P.; Falaschetti, L.; Turchetti, C.; Lattuada, M. Online Learning of Oil Leak Anomalies in Wind Turbines with Block-Based Binary Reservoir. *Electronics* **2021**, *10*, 2836. https://doi.org/10.3390/electronics10222836

Academic Editor: José L. Abellán

Received: 7 October 2021
Accepted: 13 November 2021
Published: 18 November 2021

1. Introduction

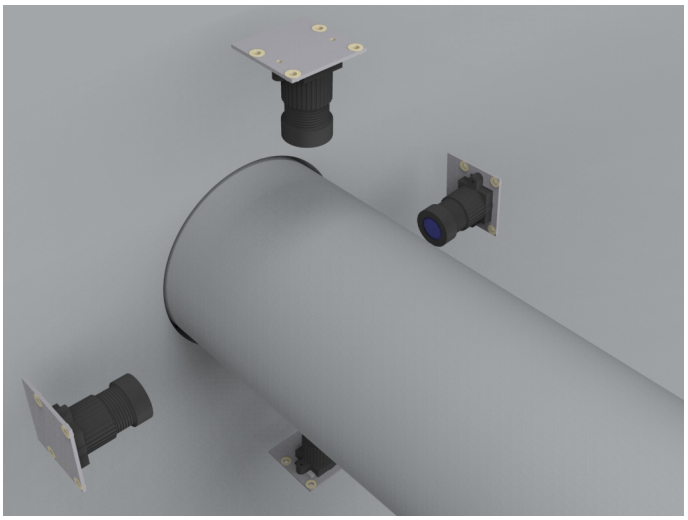
The wind industry is continuing its growth without peace. In 2017 there were 341,000 wind turbines on the planet [1]. In June 2021 alone, 3.5 gigawatts of new wind turbines were opened, proving that the need for wind energy and renewable resources is continuing to be a priority for many countries. In 2021, it is expected that the capacity of wind turbines will increase substantially [2]. They are typically clustered together in wind farms [3–5], each of which can generate enough electricity to power thousands of homes. Wind farms are usually located on top of a mountain or in an otherwise windy place, e.g., offshore in the sea, to take advantage of natural winds [6]. An example is the Walney Extension located in the Irish Sea approximately 19 km west of the northwest coast of England and covers a massive area of 149 square kilometers [7]. Each of the 87 wind turbines stands 195 m tall, making these offshore wind turbines some of the largest wind turbines in the world. The Walney Extension has the potential to generate 659 megawatts of power, which is enough to supply 600,000 homes in the United Kingdom with electricity. As a consequence, for those wind turbines inspection, monitoring, and repairing is a very complex and costly task performed in a harsh environment and represents a fast-growing

industry due to the high number of existing wind turbines requiring maintenance and approaching an advanced age. As these wind turbines age, predictive maintenance is emerging as a must-have requirement.

Predictive maintenance differs from preventive maintenance because it relies on the actual condition of equipment, rather than average or expected life statistics used to predict when maintenance will be required. Typically, Machine Learning approaches are adopted for the definition of the actual condition of the system [8–10] including the capability to detect anomalies [11–14].

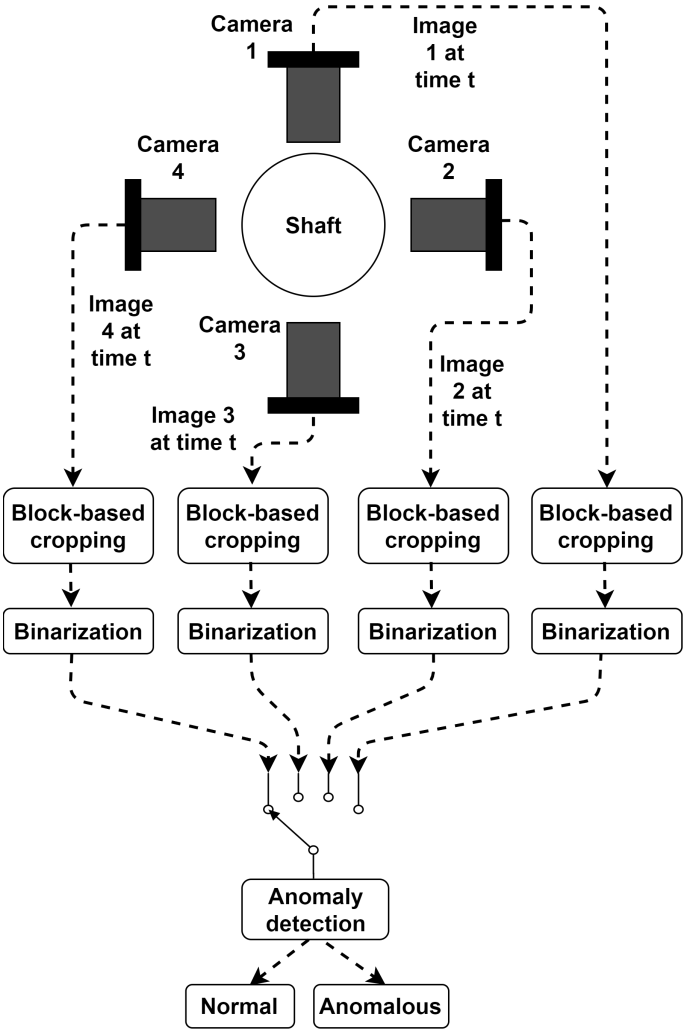
In particular oil leak detection in wind turbines is an interesting problem, since it can provoke fires inside the turbine and because human prompt intervene is very difficult to happen, being both dangerous and very expensive [15,16]. Furthermore, the access to the inside cabin of the generator is even more challenging as they may be located offshore or in hard-to-access locations. The inside of the generator offers an interesting opportunity: it is a closed, fully controlled environment suitable to be monitored using image cameras. Indeed, it is characterized by uniform color background, limited presence of dust and dirt, absence of stains, which make the conditions quite constant in absence of anomalies. This suggests that the deployment of image cameras to monitor them is much less affected by perspective and illumination changes than cameras operating in other open un-controlled environment scenarios, such for example autonomous driving. Therefore, they can be used to perform oil leak detection in a kind of laboratory-controlled working conditions, allowing the oil leak to be the only possible change to be detected. It is worth noting that despite the fully controlled conditions, the acquired images can still be characterized by the presence of noise introduced for example by the degradation of the image sensors. Once any anomaly will be detected, it will be promptly communicated to the maintenance service department and then a specialized operator intervention will be programmed.

The proposed system is conceived in such a way that image sensors are framing the junction between the shaft and the bracket of the power generator, where oil leaks would be located in the case of anomaly. The junction needs to be framed with a 360° field of view. For this reason, at least 3 or 4 cameras would be needed (the zones framed by the camera can also be overlapped) depending on the field of view. Figure 1 shows a simplified scheme with 4 cameras framing the junction.



(**a**)

Figure 1. *Cont.*



(**b**)

Figure 1. Exemplary installation with 4 cameras framing the junction where anomalies can happen. (**a**) The cameras framing the junction. (**b**) Block diagram of the cameras framing the junction, being synchronized in time.

Such system architecture needs to design a machine learning-based anomaly detector per each camera, to detect anomalies 360 degree.

Inside each image, a precisely defined zone of interest is defined, limiting the search area of the leaks, with also the advantage of reducing the processing complexity.

To the best of our knowledge, no public dataset about this problem exists in the state of the art and to simulate the zone of interest for the design and test of the anomaly detector. Collecting this type of image indeed would require direct access to the internal structure of wind turbines, limiting the possibilities of acquisition to companies performing their maintenance. To mitigate this limitation, the images used in this paper have been produced through 3D graphics photo-realistic rendering, reproducing scenarios with and without anomalies and from different viewpoints, i.e., camera positions. The noise both during the generation of the images and during their import in the system for validating the proposed approach tries to minimize their artificiality and makes them as close as possible to real images. Nevertheless, it is not guaranteed that the type of added noise covers whatever type of irregularities caused by unpredicted external factors in the considered environment. For this reason, availability in the future of real images datasets can actually help in validating the proposed approach.

The challenge we addressed consists of designing a system that can learn on site the normality with low and fixed computational complexity as well as fixed memory storage during its functioning. Thus, we investigated a system that can possibly run with low power consumption on a cheap and off-the-shelf micro-controller (MCU) which, in a preferred embodiment, is attached to all those image sensors instantiated into the system to achieve close-to-sensor machine learning computing.

The constraint of fixed complexity learning brought us to the choice of an echo state network, which belongs to the reservoir computing category, as will be specified in Section 3. Low complexity requirements are not only achieved by designing a neural network with parsimonious model size but also binarizing the images captured by the sensor before being processed by the neural network. This is also possible due to the controlled lighting and overall working conditions inside the generator cabin. The images binarization, in turn, creates another opportunity: to further reduce the complexity of the network by deeply quantizing and at most even binarizing its weights, including the reservoir layers ones, and the activations. To aggressively reduce the memory occupation and improve pipelined processing of the system with respect to a network working on the entire image, the framed zones of interest feeding the neural network are further decomposed into non-overlapped blocks to mimic a sequential scan in, for example, lexicographic order, or in any other as the sensor technology permits. Finally, we designed the system with the aim to be robust to defective phototransistors in the image sensor, in order to deal with the lifetime of the system. The sensor's phototransistors may in fact be damaged by an imperfect manufacturing process, aging, or averse ambient conditions such as temperature variations. In summary, the peculiarity of the proposed work is in the use of a deeply quantized (8 and 1 bit) block-based ESN network with more than one reservoir layer, capable of performing online normality (oil leak-free) learning with a cap on complexity and using images decomposed in a block-based temporally processed sequence, followed by a special purpose anomaly evaluator of the network output.

This paper is organized as follows: Section 2 introduces the system architecture and the requirements to be full-filled; Section 3 reviews related work and highlights major differences versus proposed work; Section 4 describes the novel dataset created to model the problem and in particular images with and without oil leaks; Section 5 describes the novel BBS-ESN pipeline in both full precision baseline and quantized versions. In Section 6 the metrics for evaluating anomaly detection capabilities are introduced; next in Section 7 the obtained results are reported and discussed and in Section 8 the implementation of the quantized BBS-ESN on a STM32 MCU is evaluated, with particular focus on the execution times. Finally, in Section 9 the conclusions and the possible future works are summarized.

2. System Architecture and Associated Requirements

The system architecture is shown in Figure 1. 360-degree monitoring of the junction is achieved, as an exemplary embodiment, using more than one camera with an adequate field of view (as explained in Section 4). The number of cameras to be used depends on their field view, so the system needs to be scalable in their number. Each camera outputs only a limited set of fixed position block-based zones of interest which are next binarized since the background is light and the dark shape of leaks is the only relevant information to solve the problem while there is no need to process color information which is considered redundant and deceitful. The binarization adopted in this study is explained in Section 4.2. Next, the binarized blocks are used as input by the micro-controller unit (MCU) whose aim consists of detecting the presence of any leaks exploiting the presented machine learning solution.

In Figure 2 the block diagram of the system architecture is shown, from the image acquisition to their classification as normal or anomalous. The outputs of the neural network are evaluated with a decision process, called Evaluator, which will be described in detail in Section 6.

The processing complexity that any MCU needs to support shall be minimized without compromising the accuracy of the solution. If that would be achieved, then the derived approach can be implemented on a low-power micro-controller considering limited available embedded RAM and FLASH memory.

To implement the system on a low-power MCU the processing complexity shall be minimized. In fact, the available embedded RAM and FLASH memory in a micro-controller is limited and cannot be expanded further during the aging of the device. Therefore, the neural network must be carefully designed. Besides limiting the network weights number, complexity and memory reduction can be achieved with quantization, even binarization in some cases, of the weights. To further decrease memory buffering complexity, block-based processing is used, mimicking each block decomposed zone of interest in a kind of temporal sequence.

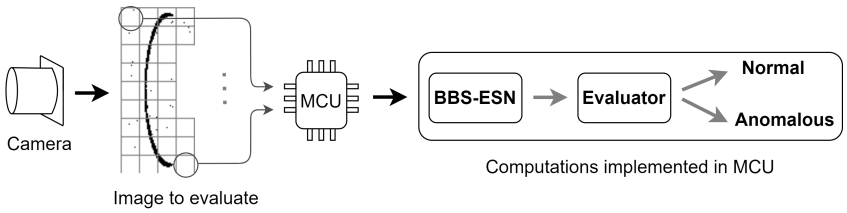


Figure 2. BBS-ESN network with camera inputs, processing them as temporal sequences of blocks.

The possibility to implement the detection in an MCU, with small power consumption, is also made possible by the absence of real-time constraints (unlike autonomous driving which requires inference in a few milliseconds). Therefore, once the zones of interest of an image are processed by the micro-controller, then that end can trigger the next zone of interest acquisition to happen. Indeed, another assumption is that the human maintainer intervention time is order of magnitudes greater than the detection time.

Furthermore, since the system must not require pre-knowledge or pre-calibration, the system must be able to learn the normality condition on site. After the learning is concluded in a fixed amount of time, it shall be able to detect anomalies with a low number of false negatives (to avoid missing dangerous anomalies) and false positives (to avoid triggering un-needed human intervention).

3. Related Work

Oil leak detection has been used studied in the context of satellites synthetic aperture radar images of the sea surface. One important process in these studies that happens before the evaluation of anomalies is the feature extraction that can take place through thresholding [17,18]. Another common process for these studies, also performed before the evaluation, is the image segmentation [17,19–21]. Segmentation divides the images blocks (of fixed or variable dimensions) into sub-images. The segmentation permits, also, the discarding of unuseful blocks, such as land zones [21].

In this case, however, blocks have been used as a single image, uncorrelated with the other blocks. Moreover, network used was not recurrent, since the images were not analyzed in a temporal fashion.

Apart from these fields of study, block-based image processing has been used in Compressed Sensing [22], Noise Removal [23], Image Recapture [24] and Artifact Removal and Image Compression [25,26].

Anomaly detection or outlier detection is an important problem that refers to the task of identifying patterns in data that do not comply with the expected behaviour [27–30]. The non-conforming and unexpected patterns are called outliers or anomalies. Anomaly detection has been extensively used in a wide variety of applications such as cyber-intrusion detection [31], fraud detection [32], medical anomaly detection [33,34], industrial damage detection [35], hyperspectral image analysis [36], sensor networks [37], image processing [38], to cite just a few. Outlier detection is very popular in industrial applications [39–43] since

it is critical to the efficient and secure operation of industrial equipment, integrated sensors, and the overall production process. Many industrial real problems focus on finding outliers from time series, i.e., data depending on time [29,44–48]. This is a challenging task since there are many requirements for time series dataset to meet: (i) the detection algorithm should satisfy real-time requirements, especially in some industrial monitoring, (ii) outliers are not necessary the maximum or minimum data, (iii) it cannot be assumed that data points are independent and identically distributed, due to strong continuity and correlation.

There are three distinct categories of anomaly approaches, depending on the existence of labels in training data, i.e., supervised, semi-supervised, or unsupervised anomaly detection. This paper deals with the last category of unsupervised anomaly detection, which is the most difficult scenario due to the absence of any label of the data. To tackle this problem, a great number of approaches exists based on analyzing segments or subsequences in a pattern sequence. However, the performance of such methods significantly degrades for high-dimensional data or in the presence of noise. To overcome such limitations deep learning solutions [49–53], have shown to be very effective in solving unsupervised anomaly detection problems.

Two different deep learning approaches have shown to be profitable to solve the above-discussed problem: extreme learning machines (ELMs) and deep echo state networks (DESNs). In this paper, an approach of the second time is presented which includes network quantization and use of intersection of confidence tests.

3.1. Extreme Learning Machines

Due to the problem and requirements discussed in Section 2, inference and learning need to be performed in a fixed and predictable amount of time and memory. These needs represent a major limitation in the choice of NN topologies since most training methods converge to the optimal solution in an a priori unknown amount of computational and storage resources. The typical training procedure of NN is based on backpropagation using stochastic gradient descend using multiple epochs [54] by processing large batches of data. This limits our interest to extreme learning machines (ELM)s. ELMs are defined as feedforward, not recurrent, neural networks for classification, regression, clustering, sparse approximation, compression, and feature learning with a single layer or multiple layers of hidden nodes, in which the parameters of the hidden nodes (not just the weights connecting inputs to hidden nodes) do not need to be tuned. These hidden nodes can be randomly assigned and never updated (i.e., they are random projection but with nonlinear transforms), just as can happen to RC weights, or can be inherited from their ancestors without being changed. In most cases, the output weights of hidden nodes are usually learned in a single step, which essentially amounts to learning a linear model, such as the Readout model, following a reservoir layer, is trained through a linear solver, e.g., Ridge regression [55]. The name "Extreme Learning Machines" was given to such models by its main inventor Guang-Bin Huang [56]. According to their creators, these models can produce good generalization performance and learn thousands of times faster than networks trained using backpropagation [57]. In the literature, it is also shown that these models can outperform support vector machines in both classification and regression applications [58–60].

The output function of an ELM is

$$f_L(x) = \sum_{i=1}^{L} \beta_i h_i(a_i, b_i, x) = h(x)\beta \tag{1}$$

where $\beta = [\beta_1, \ldots, \beta_L]^T$ is the output weight vector and $h(x) = [h_1(a_1, b_1, x), \ldots, h_L(a_L, b_L, x)]$ is the output vector of the hidden layer, with hidden node parameters (a_i, b_i). Basically, an ELM is trained in two main stages: (1) random feature mapping, and (2) linear parameters solving. In the first stage, the hidden node parameters are randomly generated

(independent of the training data) according to any continuous probability distribution instead of being explicitly trained, thus obtaining a remarkable efficiency compared to non-random neural networks. In the second stage of ELM learning, the weights β connecting to the hidden layer and the output layer are derived by solving the linear equation

$$T = H\beta \tag{2}$$

where H is the hidden layer output matrix and T is the training data target matrix [58].

As a comparison with the network designed for this study, Section 7.1.4 reports the results obtained using the same proposed network structure without the reservoirs recurrent connections.

3.2. Deep Echo State Networks

Echo state networks have been used for anomaly detection, in particular for situations where the network is trained only with data belonging to the normal class (for example in situations where the available data are imbalanced in the numbers for the normal and anomalous classes). Echo state network have been used for online training in [61–63]. Deep echo state networks extend the principles of ESN [64–66] as they embody more than one reservoir layer [67]. The rationale behind the stacking of reservoir layers is the processing of each temporal information in hierarchical feature abstractions [68].

Proposed BBS-ESN uses deep echo state networks (DeepESN). Its state at time t is computed with (3):

$$\boldsymbol{x}^{(r)}(t) = f\left(\boldsymbol{W}_{in}^{(r)}\boldsymbol{i}^{(r)}(t) + \boldsymbol{W}^{(r)}\boldsymbol{x}^{(r)}(t-1)\right) \quad , \tag{3}$$

where $\boldsymbol{W}_{in}^{(r)}$ and $\boldsymbol{W}^{r}$ represent respectively the input weights matrix and the recurrent matrix of the rth layer. $\boldsymbol{i}^{r}$ is the input to the rth layer. For the 1st layer it is the input $\boldsymbol{u}(t)$, which is a flattened block. For the subsequent layers $\boldsymbol{i}^{r}$ is the state of the reservoir in position $r-1$ at the same time, $\boldsymbol{x}^{(r-1)}(t)$. f represent an element-wise nonlinear function.

The network output is computed stacking the states of the reservoirs and multiplying the vector obtained for the readout matrix $\boldsymbol{W}_{out}$, as in (4):

$$\boldsymbol{y}(t) = \boldsymbol{W}_{out}\boldsymbol{x}(t) \tag{4}$$

The training takes place using the network states associated with the target outputs, as explained in [65], using the stacking of the reservoir layers states as network state.

The readout matrix $\boldsymbol{W}_{out}$, of an DeepESN is the only part of the network subject to online learning, while the other network weights are instantiated randomly. This is why we suggest considering DeepESN and BBS-ESN as examples belonging to the ELM family. The training of $\boldsymbol{W}_{out}$ is performed using a fixed number of blocks and therefore will achieve a fixed complexity differently to methods based on backpropagation using a variable number of epochs to converge to the optimal set of weights.

Considering the state at time t as the vertical stacking of the states of each reservoir at the same time (taken as column vectors):

$$\left[\boldsymbol{x}^{(1)}(t)\ \boldsymbol{x}^{(2)}(t)\ ...\ \boldsymbol{x}^{(N_R)}(t)\right]^T = \boldsymbol{x}(t) \in \mathbb{R}^{N_R N_N} \quad , \tag{5}$$

being N_R the number of reservoir layers and N_N the neurons in each reservoir (considered the same for all reservoirs). The training of the readout matrix takes place in batch, using the grouped states from T preceding instants in

$$\left[\boldsymbol{x}(1), \ldots, \boldsymbol{x}(T-1), \boldsymbol{x}(T)\right] = \boldsymbol{X} \in \mathbb{R}^{N_R N_N \times T} \tag{6}$$

The formula for the readout train is the following:

$$\boldsymbol{W}_{out} = \boldsymbol{Y}_{target}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \alpha^2\boldsymbol{I})^{-1} \quad , \tag{7}$$

where $\boldsymbol{I} \in \mathbb{N}^{N_R N_N \times N_R N_N}$ is an identity matrix, α^2 is a regularization term and $\boldsymbol{Y}_{target}$ is the matrix of the target outputs associated with the states matrix $\boldsymbol{X}$ [65].

3.3. Network Quantization

Quantization is known as the process of approximating a continuous signal by a set of discrete symbols or integer values. Most networks are trained using fp32 precision. In our case, an fp32 or even fp64 representation has greater precision than needed and requires 4 to 8 times more memory than an integer 8 bits model or 32 to 64 times than an integer 1 bits model. Therefore, converting fp32 parameters to lower bit representations, such as int8 and int1, can significantly reduce memory footprint, power consumption and offer the opportunity for an increased execution speed. However, this result cannot be achieved by compromising the accuracy of the full precision model. Quantization can be categorized into two areas: (1) quantization aware training (QAT) and (2) post-training quantization (PTQ). The difference depends on whether quantization is affecting weights learning during training. Alternatively, it is also possible to categorize quantization by where data are grouped for quantization: (1) layer-wise and (2) channel-wise. Finally, while evaluating parameter widths, another possible classification is by bit-depth, such N-bit quantization. With respect to network quantization, examples can be found in [69,70]. In [71], the authors present a study of an ESN designed with states represented by integer numbers and binary recurrence matrix.

3.4. Intersection of Confidence Tests

Anomaly Detection often requires analysis on the residual computed as the difference between the input and the output of an anomaly detector, implemented as an autoencoder [72,73]. Typically, the residual is analyzed using a change detection tests, i.e., statistical tools able to sequentially analyze data looking for changes [74]. Examples are, i.e., a majority-voting thresholding test and the ICI-based Change-Detection Test [75]. Such tests are characterized by lightweight computational load and reduced memory occupation and can be directly executed by a micro-controller; Some other examples can be found in [76–78].

4. Dataset Creation

A public dataset comprehensive of various framings of the junction object of this study, in the presence and absence of oil leaks, was not available, to the best of our knowledge. Creating a collection of real images would require direct access to the intern of wind turbines, which is possible only for their maintainers. For this reason, we created a specific one consisting of a collection of synthetic images photo realistically similar to those that would be acquired by a live image sensor, representing normality (absence of leaks) and abnormality (presence of leaks). The images have been produced through graphic rendering in RGB format. We produced these images in various framings, viewpoints, and surfaces textures and, for the anomalous images, various oil leak colors. Grayscale and binary images have been generated through image preprocessing, using the RGB images as a starting point. This dataset, named Oil leak dataset, is publicly available at [79]; the full details of the dataset and of its creation can be found in [80].

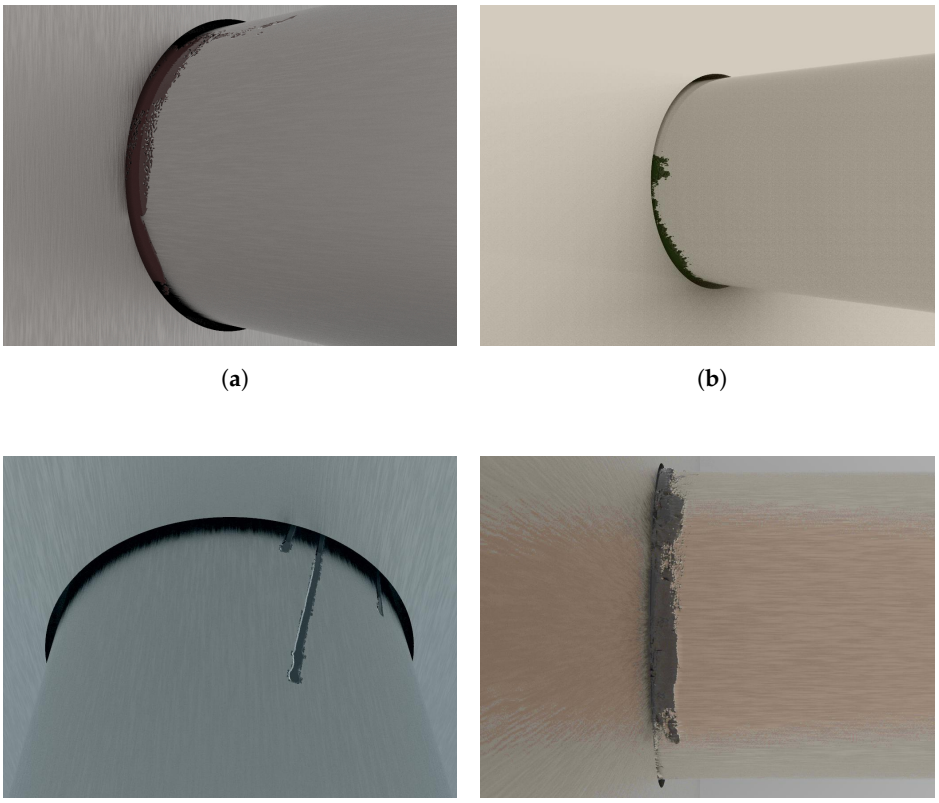
4.1. Generation of the RGB Images

A 3D model of the junction has been created as the renderings starting point. The oil leaks have been simulated with images produced on purpose and added as textures located in the junction proximity. The objective was to represent various types of anomalies, from slightly evident leaks near the junction to splashes spreading in the shaft.

Regarding the surface textures for the shaft and the bracket, a metallic texture has been applied in two different variations: one for not painted steel and another for steel painted in gray. 16 framing have been used as the rendering points of view for each parameter combination. Images preprocessing has been applied to simulate various light conditions.

To estimate of the camera distances to have a 360° view, we took advantage of the cylindrical nature of the junction. The rendered images have been generated with a field of view of 39.6°. In the hypothesis of positioning the cameras with view centered perpendicularly to the shaft surface, the condition for a with camera with field of view of 39.6° to have a 90° view of the shaft is to be at a distance from the shaft radius of $l = (r\sqrt{2}/2) \cdot \left(1 + \tan^{-1}(39.6/2)\right) = 2.67\,r$ (with r the shaft radius).

For the images in this dataset, the distances of the virtual cameras are variable and permit coverage of a view of 180° of the shaft, generating as a consequence overlapping in the covered zones. The picture resolution is 1024 × 768. In Figure 3 some examples are shown.



(**a**) (**b**)

(**c**) (**d**)

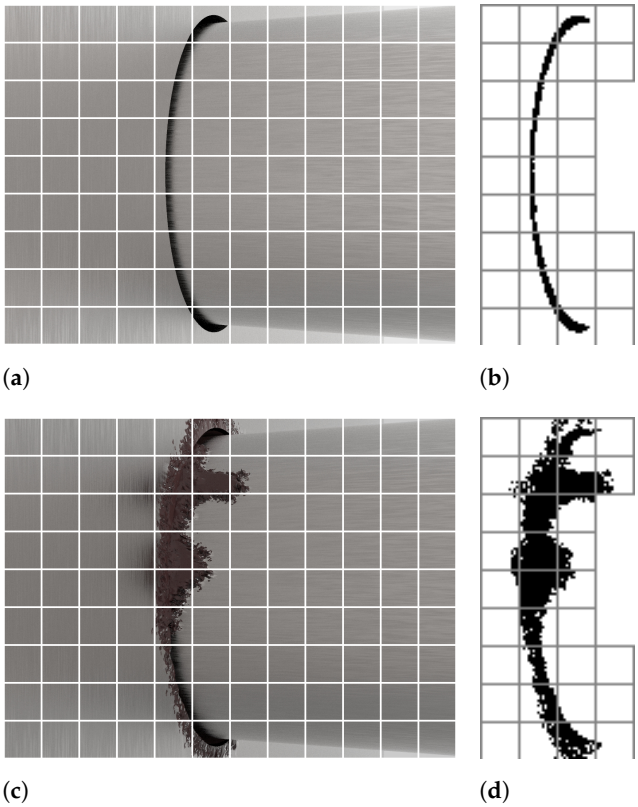
Figure 3. Some examples of RGB images of the generated dataset. (**a**) Brown leak, metallic texture, natural light. (**b**) Green leak, white paint texture, warm light. (**c**) Gray leak, white paint texture, cold light. (**d**) Brown leak, metallic texture, oversaturated.

4.2. Grayscale and Binary Images

RGB images have been subsequently elaborated to be converted to grayscale and to binary format. The conversion to grayscale is performed by eliminating the hue and saturation information, retaining the luminance value [81]. The algorithm chosen to binarize the grayscale images is the Otsu's method, which uses a globally determined threshold [82]. Due to the information loss related to color, the images used for this process have been the ones produced with brown leaks and only for the original light condition obtained from the renderings. This process could be done for other leak colors and light conditions with near equal results. The smaller number of grayscale and binary images, for this reason, brought us to apply data augmentation to these two datasets (even though only binary images have been used in this study).

For the network design, using binary pictures, the image original sizes have been scaled reducing each side to approximately 20% of the original size, to 192 × 144 resolution. Resizing has been used both for the baseline and the quantized versions of the BBS-ESN and

it has been performed using the bilinear interpolation technique [83]. Since we assumed the cameras to be mounted in fixed positions as shown in Figure 1a, we can decompose through a fixed mask in non-overlapped blocks the images and read from the sensor only those which enclose portions of interest of the junction. Figure 4 shows the images divided into blocks (both RGB and binarized), and that only the blocks outlining the junction have been chosen to be used as network inputs for both learning and detection of anomalies. In fact, they are the only ones carrying useful information for the purpose of this study.



(**a**) (**b**)

(**c**) (**d**)

Figure 4. Normal and anomalous images binarized and resized, with selected subset of blocks subject of further processing. (**a**) RGB normal image, 1024 × 768 pixels. (**b**) Binary normal image, 192 × 144 pixels. (**c**) RGB anomalous image, 1024 × 768 pixels. (**d**) Binary anomalous image, 192 × 144 pixels.

To test the BBS-ESN networks simulating the temporal propagation of oil leak anomalies with a gradual transition, image morphing has been used. Normal images have been used to create the starting sequence and anomalous ones with the same framing (and original metallic texture in the RGB images) have been used as ending sequence. In Table 1a,b are reported the images numbers with and without leaks.

It is important to point out that the images are used as network inputs in a binary representation of −1 and 1 values.

Table 1. Quantitative view of the dataset.

(**a**) **Number of images produced by rendering, preprocessing and binary data augmentation.**			
	RGB	Grayscale	Binary
With leak	11,520	20,480	30,720
Without leak	192	1024	1536
(**b**) **Number of morphed binary sequences**			
Morphed binary sequences		10	
Images per sequence		100	

5. Proposed Block-Based Binary Shallow Echo State Network (BBS-ESN)

Two different network versions have been developed, presenting the same architecture but differing in the weights and non-linearity precision. The two networks, named baseline and quantized BBS-ESN, also differ because the in the baseline version the reservoirs connectivity is 100%, meaning that the reservoir matrices have all non-zero coefficients, while in the quantized version has binary reservoir matrices with non-zero weights only on the matrix diagonals. This Section is organized as follows: in Section 5.1 the two networks architecture are introduced, in Section 5.2 the BBS-ESN training is explained, Section 5.3 introduces the baseline BBS-ESN and Section 5.4 introduces to the quantized BBS-ESN.

5.1. Architecture Overview

The BBS-ESN was designed, in both full precision baseline and a quantized version, with two reservoir layers. Figure 5 shows the top-level view of the BBS-ESN.

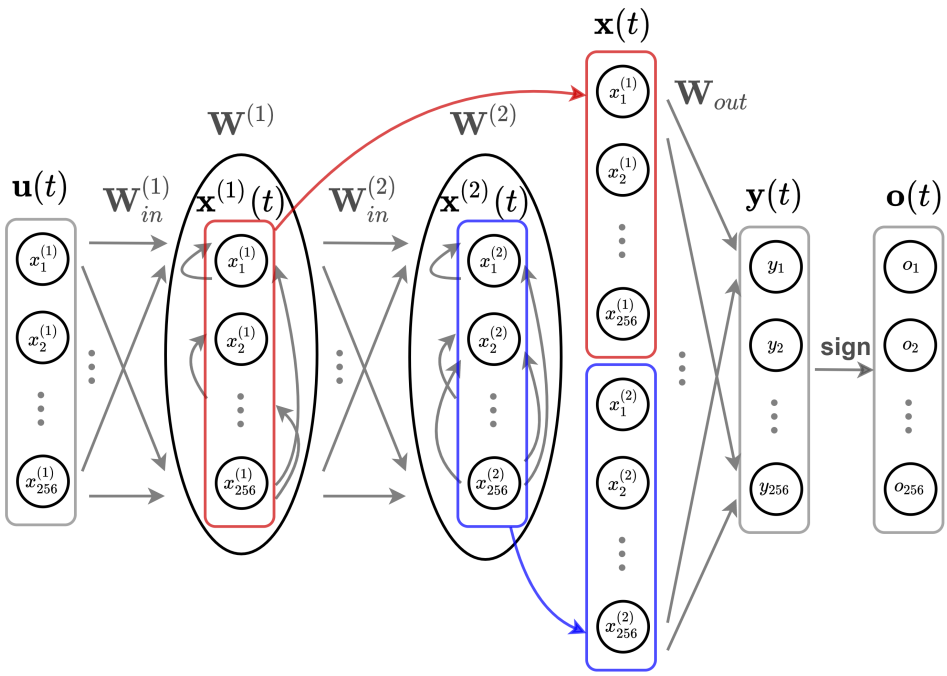


Figure 5. Top architecture of the BBS-ESN with 2 reservoir layers.

The network pipeline follows the description below:

- The input to the network, $\boldsymbol{u}(t)$, is a binary image block (of dimension 16×16) flattened in a column vector of dimension 256×1. The binary numbers are represented by -1 and 1 values. The input is mapped into the first reservoir layer by the matrix $\boldsymbol{W}_{in}^{(1)}$. The mapping of the input activations into the reservoir is added to the reservoir state at the previous time $\boldsymbol{x}^{(1)}(t-1)$, multiplied by a reservoir matrix $\boldsymbol{W}^{(1)}$. The weights of $\boldsymbol{W}^{(1)}$ represent the weighted connection between the network reservoirs, from which the state of each neuron depends on the state of the others. A non-linearity f is then applied to the sum $\boldsymbol{W}_{in}^{(r)}\boldsymbol{i}^{(r)}(t) + \boldsymbol{W}^{(r)}\boldsymbol{x}^{(r)}(t-1)$ as shown in (3).
- The same operations are performed in the second reservoir layer, except for its input, i.e., the first reservoir state at the same temporal instant (see (3)). The weights of the matrices $\boldsymbol{W}_{in}^{(1)}$, $\boldsymbol{W}^{(1)}$, $\boldsymbol{W}_{in}^{(2)}$ and $\boldsymbol{W}^{(2)}$ are composed by randomly instantiated weights that are kept always fixed during learning and inference.
- The network readout $\boldsymbol{y}(t)$ is computed using both the reservoir states stacked vertically, as per (8),

$$\boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{x}^{(1)}(t) & \boldsymbol{x}^{(2)}(t) & . \end{bmatrix} \tag{8}$$

The BBS-ESN output is expressed in (9)

$$\boldsymbol{y}(t) = \boldsymbol{W}_{out}\boldsymbol{x}(t) \quad . \tag{9}$$

The matrix $\boldsymbol{W}_{out}$ is the only element of the network that is subject to training.

- Finally, the sign extraction is applied to the network readout $\boldsymbol{y}(t)$ (10):

$$\boldsymbol{o}(t) = sign\big(\boldsymbol{y}(t)\big) \quad . \tag{10}$$

5.2. BBS-ESN Training

The network training modality is based on the assumption that the image sensor is initially set and turned on framing the junction while it is in normal state (i.e., without leaks). The rationale is that the network, through training, aims at approximating the input image in a situation without anomalies, while the framing of the camera remains the same. Once the learning is concluded, the network is put in inference mode.

An anomaly condition will be asynchronously identified if the output will be significantly different from the input. This condition will occur because the image under analysis is significantly different from the ones used by the training step, therefore being anomalous. This approach is similar to the ones adopted in [76–78] using ESN.

Due to this formulation of network training, the network states associated with input blocks are stacked into the state matrix $\boldsymbol{X}$, while the input blocks are used as train targets. We will have, therefore, that the states are disposed as in (11):

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^{(1)}(1) & \dots & \boldsymbol{x}^{(1)}(B) \\ \boldsymbol{x}^{(2)}(1) & \dots & \boldsymbol{x}^{(2)}(B) \end{bmatrix} \quad . \tag{11}$$

begin B the number of blocks used for the training. The matrix of the train targets $\boldsymbol{Y}_{targets}$ is represented in (12)

$$\boldsymbol{Y}_{target} = \begin{bmatrix} \boldsymbol{u}(1) & \dots & \boldsymbol{u}(B) & , \end{bmatrix} \tag{12}$$

where the blocks $\boldsymbol{u}(t)$ are acquired from more than one image and are placed in the same order they have been used as network inputs. These two matrices are used to compute $\boldsymbol{W}_{out}$ as in (7). The sign extraction of (10) does not have a role in the training. The blocks order of (12) can vary and is explained in Section 7.1. The choice on the number of blocks fell on 80 and it is explained in Section 7.1.3. As for the choice of stacking the states vertically, experiments have been conducted using the visualization technique t-SNE [84] to embed the states, treated as 256-dimensional values, in a 2-dimensional map. These experiments have shown a separation of the mapping zones of the 2 reservoir states, justifying the association of different weights to them (as the scheme in Figure 5 proposes).

5.3. Baseline BBS-ESN Implementation

The baseline BBS-ESN relies on double-precision floating point weights and computations and employs two reservoirs of 256 neurons each. The baseline BBS-ESN inputs are column vectors of dimension 256×1 (obtained from images blocks of dimension 16×16 that are flattened). The network output is also a column vector of dimension 256×1.

All the matrices are fully connected, which means that they have all non-zero coefficients. The matrices $\boldsymbol{W}_{in}^{(1)}$, $\boldsymbol{W}^{(1)}$, $\boldsymbol{W}_{in}^{(2)}$ and $\boldsymbol{W}^{(2)}$ are instantiated with a random uniform distribution between -1 and 1, using the MT19937 pseudorandom number generator [85]. The coefficients of $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ have been rescaled in order to have spectral radius [65,67] $\rho(\boldsymbol{W}^{(1)}) = \rho(\boldsymbol{W}^{(2)}) = 0.95$.

The baseline BBS-ESN parameters are summarized in Tables 2–4 ("bin[−1,1]", from here on, stays for "binary distribution of -1 and 1"). In Table 3 is reported that the matrices $\boldsymbol{W}_{in}^{(1)}$, $\boldsymbol{W}_{in}^{(2)}$, $\boldsymbol{W}^{(1)}$, and $\boldsymbol{W}^{(2)}$ are to be saved in Flash memory. The reason is that since they are random instantiated values that will never change, they can be used as constant values.

The readout training of the baseline BBS-ESN implements the operations reported in Table 5. These operations are the ones performed in (7), using $\alpha^2 = 1$.

Table 2. Baseline BBS-ESN parameters.

Blocks size	16×16 pixels
Blocks values distribution	bin.$[-1, 1]$
Input vector units *	256×1 pixels
Number of reservoirs	2
Neurons per reservoirs	256
Reservoirs Non-linearity	tanh
Output vector units	256
Output vector distribution	bin.$[-1,1]$
Reservoirs states in readout	Vertically stacked
$\boldsymbol{W}_{in}^{(1)}$, $\boldsymbol{W}_{in}^{(2)}$	Uniform distribution of fp64 between -1 and 1
$\rho(\boldsymbol{W}^{(1)})$, $\rho(\boldsymbol{W}^{(2)})$	0.95
Connectivity of reservoirs	100%

* The input vectors are obtained flattening the blocks.

Table 3. Baseline BBS-ESN weights and memory occupation.

Position	**Number**	**Type**	**Occupied Memory [KBytes]**	**Memory Type**
$\boldsymbol{W}_{in}^{(1)}$	65,536	fp64	512	Flash
$\boldsymbol{W}^{(1)}$	65,536	fp64	512	Flash
$\boldsymbol{W}_{in}^{(2)}$	65,536	fp64	512	Flash
$\boldsymbol{W}^{(2)}$	65,536	fp64	512	Flash
$\boldsymbol{W}_{out}$	131,072	fp64	1024	RAM
Total	393,216	fp64	2048 1024	Flash RAM

Table 4. Baseline BBS-ESN inference operations for one block.

Operation	Multiply	Accumulate	Non-Lin.
$\boldsymbol{a}(t) = \boldsymbol{W}_{in}^{(1)}\boldsymbol{u}(t)$	65,536 (fp64)	65,280 (fp64)	-
$\boldsymbol{b}(t) = \boldsymbol{W}^{(1)}\boldsymbol{x}^{(1)}(t-1)$	65,536 (fp64)	65,280 (fp64)	-
$\boldsymbol{x}^{(1)}(t) = \tanh(\boldsymbol{a}(t) + \boldsymbol{b}(t))$	-	256 (fp64)	tanh
$\boldsymbol{c}(t) = \boldsymbol{W}_{in}^{(2)}\boldsymbol{x}^{(1)}(t)$	65,536 (fp64)	65,280 (fp64)	-
$\boldsymbol{d}(t) = \boldsymbol{W}^{(2)}\boldsymbol{x}^{(2)}(t-1)$	65,536 (fp64)	65,280 (fp64)	-
$\boldsymbol{x}^{(2)}(t) = \tanh(\boldsymbol{c}(t) + \boldsymbol{d}(t))$	-	256 (fp64)	tanh
$\boldsymbol{y}(t) = \boldsymbol{W}_{out}\boldsymbol{x}(t)$ *	131,072 (fp64)	130,816 (fp64)	-
$\boldsymbol{o}(t) = \text{sign}(\boldsymbol{y}(t))$	-	-	sign
Total	393,216 (fp64)	392,448 (fp64)	-

* $\boldsymbol{x}(t) = \left[\boldsymbol{x}^{(1)}(t) \quad \boldsymbol{x}^{(2)}(t)\right]^T$.

Table 5. Operations performed for the readout training of the baseline BBS-ESN.

Position	Multiply	Accumulate	Division
$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T + \boldsymbol{I}$	20,971,520 (fp64)	20,709,632 (fp64)	-
$\boldsymbol{A} = \boldsymbol{K}^{-1}$		$\boldsymbol{O^3}$ order *	
$\boldsymbol{Z} = \boldsymbol{Y}_{target}\boldsymbol{X}^T$	10,485,760 (fp64)	10,354,688 (fp64)	-
$\boldsymbol{W}_{out} = \boldsymbol{Z}\boldsymbol{A}$	67,108,864 (fp64)	66,977,792 (fp64)	-

* The number of operations necessary for the matrix inversion depends on the algorithm used.

5.4. Quantized BBS-ESN Implementation

The baseline network has been simplified with quantized operations and its properties are summarized in Tables 6–8. As for the baseline BBS-ESN, in Table 7 is reported that the matrices $\boldsymbol{W}_{in}^{(1)}$ and $\boldsymbol{W}_{in}^{(2)}$ are to be saved in Flash since they are constant as in the baseline implementation.

5.4.1. Binarization of the Input Matrices

$\boldsymbol{W}_{in}^{(1)}$ and $\boldsymbol{W}_{in}^{(2)}$ have been set with uniform distributions of binary values (-1 and 1). The distributions have been obtained using the same pseudorandom number generator used for the baseline BBS-ESN, i.e., the MT11937 [85].

5.4.2. Binarization of the Reservoirs

As reported in [66], the connectivity of the reservoir $\boldsymbol{W}$, in an ESN, can be reduced leaving connections only in the diagonal matrix. In [71] the reservoir weights have been implemented as an identity matrix with all the rows shifted of a fixed position. In this study, the weights in both the reservoir matrices, $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$, have been implemented as an identity matrix. Therefore, $\rho(\boldsymbol{W}^{(1)}) = 1$ and $\rho(\boldsymbol{W}^{(2)}) = 1$. As in [71], the condition on the spectral radius has been relaxed from $\rho(\boldsymbol{W}^{(1)}) < 1$, $\rho(\boldsymbol{W}^{(2)}) < 1$ to $\rho(\boldsymbol{W}^{(1)}) \leq 1$, $\rho(\boldsymbol{W}^{(2)}) \leq 1$.

This weights distribution on $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ mean that the state of a neuron depends only on the state of the same neuron at the preceding instant.

Furthermore, the multiplication for an identity matrix does not need to be executed, nor it requires memory space to save its values. For this reason, these matrices are not listed in Tables 6 and 7.

Table 6. Quantized BBS-ESN parameters.

Blocks size	16×16 pixels
Blocks values distribution	bin.$[-1, 1]$
Input vector units *	256×1 pixels
Number of reservoirs	2
Neurons per reservoirs	256
Reservoirs Non-linearity	sign
Output vector units	256
Output vector distribution	bin.$[-1, 1]$
Reservoirs states in readout	Vertically stacked
$\boldsymbol{W}_{in}^{(1)}$, $\boldsymbol{W}_{in}^{(2)}$	Uniform distribution of binary -1 and 1
$\rho(\boldsymbol{W}^{(1)})$, $\rho(\boldsymbol{W}^{(2)})$	1.0
Connectivity of reservoirs	0.39%

* The input vectors are obtained flattening the blocks.

Table 7. Quantized BBS-ESN weights and memory occupation.

Position	Number	Type	Occupied Memory [KBytes]	Memory Type
$\boldsymbol{W}_{in}^{(1)}$	65,536	bin.$[-1, 1]$	8	Flash
$\boldsymbol{W}_{in}^{(2)}$	65,536	bin.$[-1, 1]$	8	Flash
$\boldsymbol{W}_{out}$	131,072	int8	128	RAM
Total	131,072	bin.$[-1, 1]$	16	Flash
	131,072	int8	128	RAM

The multiplication of the reservoirs state for the reservoirs matrices is not reported, since those matrices have been set to be identity matrices.

Table 8. Quantized BBS-ESN inference operations for each block.

Operation	Multiply	Accumulate	Non-Lin.
$\boldsymbol{a}(t) = \boldsymbol{W}_{in}^{(1)}\boldsymbol{u}(t)$	65,536 (bin.$[-1, 1]$)	65,280 (int8)	-
$\boldsymbol{x}^{(1)}(t) = \text{sign}\big(\boldsymbol{a}(t) + \boldsymbol{x}^{(1)}(t-1)\big)$	-	256 (int8)	sign
$\boldsymbol{c}(t) = \boldsymbol{W}_{in}^{(2)}\boldsymbol{x}^{(1)}(t)$	65,536 (bin.$[-1, 1]$)	65,280 (int8)	-
$\boldsymbol{x}^{(2)}(t) = \text{sign}\big(\boldsymbol{c}(t) + \boldsymbol{x}^{(2)}(t-1)\big)$	-	256(int8)	sign
$\boldsymbol{y}(t) = \boldsymbol{W}_{out}\boldsymbol{x}(t)$ *	131,072 (int8)	130,816 (int16)	-
$\boldsymbol{o}(t) = \text{sign}\big(\boldsymbol{y}(t)\big)$	-	-	sign
Total	131,072 (bin.$[-1, 1]$) 131,072 (int8)	131,072 (int8) 130,816 (int16)	-

* $\boldsymbol{x}(t) = \begin{bmatrix}\boldsymbol{x}^{(1)}(t) & \boldsymbol{x}^{(2)}(t)\end{bmatrix}^T$. The multiplication of thereservoir state for the reservoir matrices are not reported, since those matrices have been set to be identity matrices.

5.4.3. Quantized Readout Layer

The readout weights have been quantized to int8. The procedure follows the principle of normalization, dividing all the elements of the matrix for the smaller of the matrix entries, taken in modulus, called from now $\min(|W_{out\ i,j}|)$.

It has been assessed empirically that $\min(|W_{out\ i,j}|)$ is always smaller than 1. As a consequence, the rescaled matrix

$$\tilde{\boldsymbol{W}}_{out} = \frac{\boldsymbol{W}_{out}}{\min(|W_{out\ i,j}|)} \tag{13}$$

will be composed of elements that taken in modulus are greater or equal to 1. Therefore, the values $\tilde{\boldsymbol{y}}$, obtained using the readout $\tilde{\boldsymbol{W}}_{out}$, will be a scaled version of $\boldsymbol{y}$, obtained using the original readout $\boldsymbol{W}_{out}$. This is expressed in formulas in (14)

$$\tilde{\boldsymbol{W}}_{out}\boldsymbol{x}(t) = \tilde{\boldsymbol{y}}(t) = \frac{\boldsymbol{y}(t)}{\min(|W_{out\ i,j}|)} \quad . \tag{14}$$

Therefore, the sign extraction of $\tilde{\boldsymbol{y}}(t)$, from (10), would produce the same results of $\boldsymbol{o}(t)$ obtained with the baseline BBS-ESN, extracting the sign from $\boldsymbol{y}(t)$.

The elements of $\tilde{\boldsymbol{W}}_{out}$ are rescaled to int8 representation introducing, inevitably, a quantization error. The procedure for rescaling the $\tilde{\boldsymbol{W}}_{out}$ values and converting them to int8 representation is reported in (15)

$$\tilde{\boldsymbol{W}}_{out}^{\text{int8}} = (int8)\Big(\frac{\tilde{\boldsymbol{W}}_{out}}{\max(|W_{out\ i,j}|)} \cdot (2^7 - 1)\Big) \quad . \tag{15}$$

Since the output of $\tilde{\boldsymbol{W}}_{out} / \max(|W_{out\ i,j|})$ is at most 1, the $\tilde{\boldsymbol{W}}_{out}^{\text{int8}}$ matrix coefficients will be at most equal to $2^7 - 1$.

5.4.4. Quantized BBS-ESN Readout Training Implementation

The quantized BBS-ESN training implements the operations in Table 9. The matrix inversion in Table 9 is performed in fp32 numbers. Consequently, also the multiplication

involving these values use fp32. In Table 9 the $\boldsymbol{K}$ matrix inversion is not reported, since it can be implemented with many methods. Section 8 and Table 13 report the $\boldsymbol{K}$ inversion implementative details.

Table 9. Operations performed for the readout training of the quantized BBS-ESN.

Position	Multiply	Accumulate	Division
$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T + \boldsymbol{I}$	20,971,520 (bin.$[-1, 1]$)	20,709,632 (int8)	-
$\boldsymbol{A} = \boldsymbol{K}^{-1}$		$\boldsymbol{O^3}$ order *	
$\boldsymbol{Z} = \boldsymbol{Y}_{target}\boldsymbol{X}^T$	10,485,760 (bin.$[-1, 1]$)	10,354,688 (int8)	-
$\boldsymbol{W}_{out} = \boldsymbol{Z}\boldsymbol{A}$	67,108,864 (fp32 $\times$ int8)	66,977,792 (fp32 + int8)	-
Rescale $\boldsymbol{W}_{out}$ **	-	-	131.072 (fp32/fp32)

* The number of operations necessary for the matrix inversion depends on the algorithm used. ** The rescaling operation needs to be performed only for the quantized network.

The rescaling is the only additional operation compared to the baseline BBS-ESN readout training. It is needed to quantize the readout weights as explained in the Section 5.4.

The accumulations are performed in int8, even though the theoretical accumulation maximum could be overflowed, because the uniform distribution of binary values in the input matrices permits never reaching the accumulation maximums. This has been tested changing the accumulation type, in the training procedure, from int8 to int16 and the result analysis of Section 7 showed no changes.

5.4.5. Memory Savings

The weights memory occupation for the baseline BBS-ESN is 2048 KBytes of Flash and 1024 KBytes of RAM, while for quantized BBS-ESN these values are 16 KBytes of Flash and 128 KBytes of RAM. The quantized network weights, consequently, occupy only the 0.8% of the baseline Flash memory and only the 12.5% of the baseline RAM. Section 8 reports an analysis on the implementation of the quantized BBS-ESN.

6. Method for Detecting Anomalies

Anomalies are identified using a decision process called Evaluator, which works on the reconstruction error between the BBS-ESN input blocks and output blocks, mediated over all the blocks in an image. Specifically, the error metric is the number of pixels that are reproduced wrongly (i.e., with the opposite binary value since binarized) comparing the BBS-ESN output $\boldsymbol{o}(t)$ to the corresponding input $\boldsymbol{u}(t)$ per block basis.

The threshold of the error value is calculated, during the training of the Evaluator, from the error values of images considered normal, processed by the BBS-ESN, once its readout has been trained and set. For each of these images $\in \{T_1, \ldots, T_j, \ldots, T_n\}$ (T stays for training) the corresponding error value μ_{T_j} is computed, i.e., the mean number of wrongly reproduced pixels per block for the image T_j.

A window of error values for normal images is set, named Training Window (shown in Figure 6). This window will have as mean value μ_T, i.e., the mean of the μ_{T_j} for the images $\in \{T_1, \ldots, T_j, \ldots, T_n\}$. Its width will be set by a coefficient Γ multiplied by the variance on the μ_{T_j} values, named σ_T. Therefore, the Training Window will be computed as in (16)

$$\text{Training Window} = \left[\mu_T - \Gamma\,\sigma_T,\ \mu_T + \Gamma\,\sigma_T\right] \quad . \tag{16}$$

For each image to be tested, numbered $I_1, \ldots, I_j, \ldots$ (I stays for inference), it is calculated the mean number of wrongly reproduces pixels per image block, μ_{I_j}, and the image will be considered normal if this value is included within the normality window.

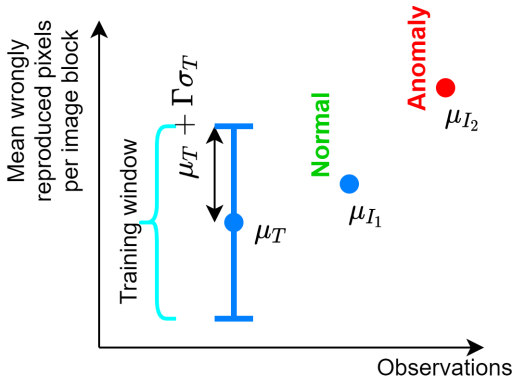


Figure 6. Detection of anomalies: an image is evaluated as anomalous if the mean number of wrongly reproduced pixels per block is outside the window of values named Training Window.

The rationale, therefore, is to perform the evaluation as the sampling from a pool of values, which can be normal or anomalous. The optimal value of Γ, depends on the defective pixel percentage of the image and will be discussed in Section 7. Figure 7 shows the pipeline for both the network training and the evaluator training.

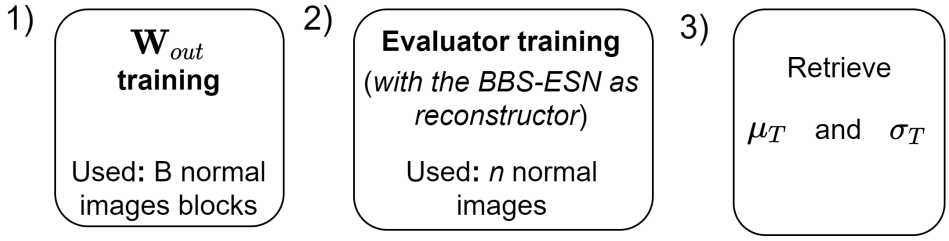
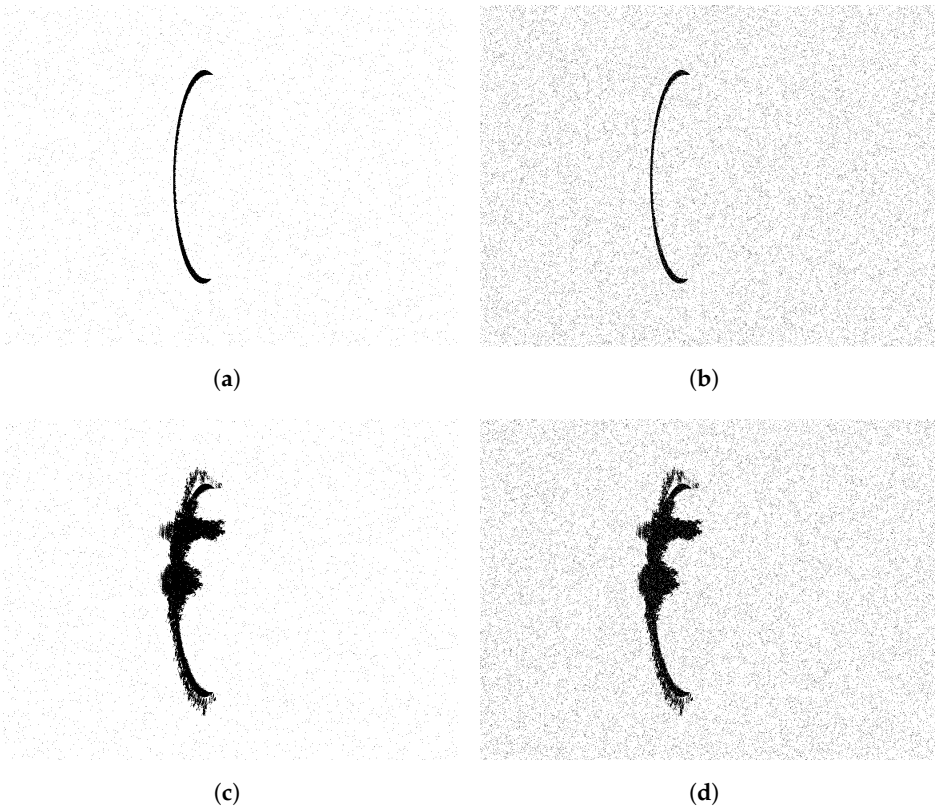


Figure 7. Pipeline for BBS-ESN (with three sequential blocks (**1**–**3**)) and evaluator training.

The Evaluator has been implemented using 100 normal images for its training, as a trade-off between test rapidity and accuracy results. In fact, the more images are used for the Evaluator training, the more the μ_T and σ_T values are computed using a wide samples population (that in this case are images). It is useful to remind that for each training the images are different from each other, despite the framing being the same, due to the time unrelated (from an image to the next one) noise simulating defective phototransistors. In particular, Salt & Pepper noise (i.e., setting to white pixels in black regions and vice-versa) has been applied to the binary images using scikit-image library [86]. Different amounts of noise (i.e., different percentage of noisy pixels) have been added to the input images. The considered values are 0.01%, 0.1%, 1%, and 10%. Example of noisy images are shown in Figure 8.



(**a**) (**b**)

(**c**) (**d**)

Figure 8. Normal and anomalous images with Salt & Pepper noise. (**a**) Binary normal image with 1% noise. (**b**) Binary normal image with 10% noise. (**c**) Binary anomalous image with 1% noise. (**d**) Binary anomalous image with 10% noise.

7. Anomaly Detection Accuracy Results

Accuracy tests, evaluating False Positives Rate (FPR) and False Negatives Rate (FNR) percentages on the evaluated images, have been performed using the binary morphed sequences. FPR in a sequence of images is defined as in (17)

$$\text{FPR} = \frac{\text{False positives}}{\text{Images in the sequence}} \cdot 100 \quad , \tag{17}$$

where "False positives" is the number of normal images evaluated as anomalous.

FNR is defined as in (18):

$$\text{FNR} = \frac{\text{False negatives}}{\text{Images in the sequence}} \cdot 100 \quad , \tag{18}$$

where "False negatives" is the number of anomalous images evaluated as normal.

7.1. Training and Testing Conditions

In these tests, every images sequence has been evaluated with the addition of various binary noise percentages, in order to simulate a variable number of defective phototransistors. It is important to point out that the tests have been conducted with various sequences (and as a consequence various leak shapes and framings) to assess the BBS-ESN accuracy in various conditions. More than one framing has been used to simulate different camera positions. In fact, during the system setup the cameras, while remaining in fixed positions, may be placed with a variable framing of the junction.

In all the tests a fixed number of 80 blocks, collected from more than one image to achieve that cap, have been used for BBS-ESN training. Before using the states from those blocks for the training, the states from the previous 50 blocks have been discarded as initial transient [64]. The choice of the training and transient blocks is explained in Section 7.1.3.

For the Evaluator training, instead, the blocks from 100 images have been used and the reason for this choice is reported in Section 6. The reason both the trainings use blocks from more than one image is due to the addition of binary noise to the images, which is unrelated from one image to the next. If there were no noise in the images both trainings could be performed only with blocks from a single image, since the camera positions are fixed and, consequently, the normal images from the same sensor would be all equal. Using only one image for the two trainings in noisy conditions, though, would cause overfitting to the BBS-ESN. As for the Evaluator, using a single image would cause the Training Window to be just a single value, treating as anomalous every image associated with a different μ_{I_j} value (Section 6). Clearly, these condition would be acceptable only in ideal conditions, i.e., with total absence of defective phototransistors.

Once both the trainings are completed, sequences of 70 images have been used to test the BBS-ESN accuracy (the value 70 has been chosen as a trade-off between a significant number of images and a brief enough simulations time). To test the anomaly detection process in different conditions, the number of normal images in each tested sequence (i.e., before the leak appearance) was different (keeping the total images for each sequence to 70.

For each network two different input blocks orders have been used to construct the train matrices $\boldsymbol{X}$ (with the states caused by the flattened input blocks) and $\boldsymbol{Y}_{target}$ (composed by the target vectors). The target vectors are the flattened input blocks themselves, since the aim of the training is the reconstruction of the input blocks (11) and (12). The two orders for the input blocks that have been tested are:

- Lexicographical order: starting from the top left block, sliding right in the same row, and repeating the process in the rows below. The process iterates for all the images used for the collection of training blocks.
- Random order: the blocks order, once the blocks from all the images have been saved, is shuffled randomly. The pseudorandom number generation algorithm used for the random shuffle is PCG64 [87].

This procedure has been done to test the sensibility of BBS-ESN to the input order. It is important to point out that the states in $\boldsymbol{X}$ follow the order of the input blocks in $\boldsymbol{Y}_{target}$, of which the states are a consequence.

7.1.1. Choice of Γ Coefficients

A different Γ coefficient has been chosen for each noise percentage and each network, to minimize FPR, FNR, and the sum of FPR and FNR percentages. The choice of the possible coefficients Γ is a trade-off result. In fact, by increasing it, this will lead to the inclusion of more μ_{I_j} corresponding to anomalous images in the Training Window, leading to an increment of the FNR. On the other hand, by lowering Γ, it will shrink the Training Window, leaving out some μ_{I_j} from normal images and increasing the FPR. The Γ values retrieved by these simulations can be used in the system deployment, depending on the defective phototransistors sensors percentage, which can be predicted from their aging. It is not possible, in fact, to retrieve the optimal Γ from the online Evaluator training. The reason is that it does not depend only on μ_T and σ_T, but also from the mean difference of the μ_{I_j} values of both normal and anomalous images, and the system only trains with normal images.

Both FPR and FNR can lead to negative effects, this is why Table 10 shows the results obtained minimizing each of the two values and the sum of both (results with defective pixels percentage of 0% and 0.001% have not been reported since they are always FPR = 0.0, FNR = 0.0).

Table 10. Accuracy results of the baseline and quantized BBS-ESN, compared with an anomaly detection algorithm that evaluates the residual image through image subtraction. A different Γ coefficient has been used for each network and each noise percentage, to minimize the FPR, the FNR and the sum of FPR and FNR.

Minimization	Input-Output Comparison	BBS-ESN Training Blocks Order	Noise 0.01%		Noise 0.1%		Noise 1%		Noise 10%	
			FPR	FNR	FPR	FNR	FPR	FNR	FPR	FNR
FPR	Baseline BBS-ESN	Random	0.0	0.0	0.0	0.0	0.0	0.0	0.0	28.7
		Lexicographical	0.0	0.0	0.0	0.0	0.0	0.0	0.0	28.7
	Quantized BBS-ESN	Random	0.0	0.0	0.0	0.0	0.0	0.0	0.0	33.1
		Lexicographical	0.0	0.0	0.0	0.0	0.0	0.0	0.0	33.1
	Residual Image Algorithm	-	0.0	0.0	0.0	22.7	0.0	54.4	0.0	92.5
FNR	Baseline BBS-ESN	Random	0.0	0.0	0.0	0.0	0.0	0.0	83.8	0.3
		Lexicographical	0.0	0.0	0.0	0.0	0.0	0.0	83.8	0.3
	Quantized BBS-ESN	Random	0.0	0.0	0.0	0.0	0.0	0.0	81.4	0.6
		Lexicographical	0.0	0.0	0.0	0.0	0.0	0.0	81.4	0.6
	Residual Image Algorithm	-	0.0	0.0	32.1	0.0	83.8	5.0	85.4	9.9
FPR + FNR	Baseline BBS-ESN	Random	0.0	0.0	0.0	0.0	0.0	0.0	2.7	13.2
		Lexicographical	0.0	0.0	0.0	0.0	0.0	0.0	2.7	13.2
	Quantized BBS-ESN	Random	0.0	0.0	0.0	0.0	0.0	0.0	4.1	14.6
		Lexicographical	0.0	0.0	0.0	0.0	0.0	0.0	4.1	14.6
	Residual Image Algorithm	-	0.0	0.0	1.2	5.6	6.6	28.9	30.7	38.6

Results with defective pixels percentage of 0% and 0.001% have not been reported since they are always FPR = 0.0, FNR = 0.0. The number of transient blocks is 50 for each BBS-ESN test. The number of blocks to train the readout is 80 for each BBS-ESN test. The number of images used to train the Evaluator is 100 in every case.

On one hand, the increase of false negatives would cause the detection of anomalies only when they are prominent, causing delays in the maintenance service of the wind turbine, at the risk of increasing occurrence of damages. On the other hand, too many false positives could cause false alarms and useless maintenance interventions, leading to inefficiencies.

Γ coefficients could be chosen, also, to minimize a weighted sum of FPR and FNR, obtaining a trade-off between detection speed and false alarms increments.

7.1.2. Test with Different Quantization Procedure

Experiments have been conducted with the training procedure of the quantized BBS-ESN. It has been attempted to rescale the matrix $\boldsymbol{K}^{-1}$ (Table 9) and to convert it to integer representation before the multiplication with $\boldsymbol{Z}$, with the aim of reducing the times required for executing the multiplication (the chosen representation has been *int32*, to reduce the overflow possibility). The results, using lexicographical training blocks order and choosing Γ coefficients to minimize the sum of FPR + FNR, achieved FPR = 0.8, FNR = 1.9 at a noise level of 1% and FPR = 3.2, FNR = 13.1 at a noise level of 10%. Due to the accuracy reduction towards the original procedure reported in Table 9, the original method has been kept. In fact, this new procedure would cause a performance reduction at noise level 1%, since using this noise level with the standard quantization procedure we have FPR = 0.0, FNR = 0.0.

7.1.3. Choice of the Number of Training and Transient Blocks

For the choice of the number of training blocks, various blocks number have been tested (200, 100, 90, 80, 60, 30) using the quantized BBS-ESN with a fixed transient of 50 blocks (that is approximately the number of blocks in two images for the images with a smaller blocks number, which has a mean of 29.4). 80 has been the only number that permitted FPR = 0.0 and FNR = 0.0 at the noise level of 1%.

As for the number of transient blocks, various values have been tested (100, 80, 50, 30, 0) using the BBS-ESN, trained with 80 blocks. 50 as number of transient blocks has been the only value allowing FPR = 0.0, FNR = 0.0.

7.1.4. Removal of the Recurrent Connections

Experiments have been conducted removing the recurrent connections with the baseline and quantized BBS-ESNs, reducing the network to be feedforward, a classic ELM (Section 3.1) with 2 hidden layers instead of 1. This also means that the elements $\boldsymbol{W}^{(1)}\boldsymbol{x}^{(1)}(t-1)$ and $\boldsymbol{W}^{(2)}\boldsymbol{x}^{(2)}(t-1)$ of (3) have been equaled to 0, transforming the operations for computing $\boldsymbol{x}^{(1)}(t)$ and $\boldsymbol{x}^{(2)}(t)$ in Tables 4 and 8 to the sole application of the element-wise sign extraction.

In Table 11 a comparison of the accuracy results of the feedforward baseline and quantized BBS-ESN with respect to the baseline and quantized BBS-ESN has been reported. Results show 0.0 values for both FPR and FNR in the conditions of sequences with 0% noise (not reported in Table 11), for both the baseline and quantized feedforward networks and for training blocks in lexicographical and random order. As for the conditions of 1% and 10% noise levels, FPR+FNR minimization, the baseline feedforward network, trained with random blocks order, scored results of FPR = 0.3, FNR = 0.0 for 1% noise and FPR = 3.2, FNR = 13.0 for 10% noise. Using a lexicographical blocks order the accuracy results were FPR = 0.3, FNR = 0.3 for 1% noise and FPR = 4.5, FNR = 12.3 for 10% noise. The quantized feedforward network scored results of FPR = 0.0, FNR = 0.3 for 1% noise and FPR = 3.9, FNR = 12.0 for 10% noise with a random training order. The results for the lexicographical training order with noise levels of 1% and 10% were, respectively, FPR = 0.3, FNR = 0.3 and FPR = 4.2, FNR = 11.7. These results were obtained using Gamma coefficient for the minimization of the sum of FPR and FNR.

Results obtained by the BBS-ESN baseline and quantized versions, both with and without recurrent connections, are marginally different. For noise level 1% the difference is under 1% for both FPR and FNR for both the baseline and quantized versions. For noise level 10% the difference is under 1% for both FPR and FNR for the baseline version and under 1% for FPR and under 3% for FNR for the quantized version.

As you can see, the network versions with recurrent connections always show a low FPR and therefore have the advantage to avoid false alarms and consequently useless maintenance interventions.

Table 11. Comparison of the accuracy results of the feedforward baseline/quantized BBS-ESN respect to the baseline/quantized BBS-ESN.

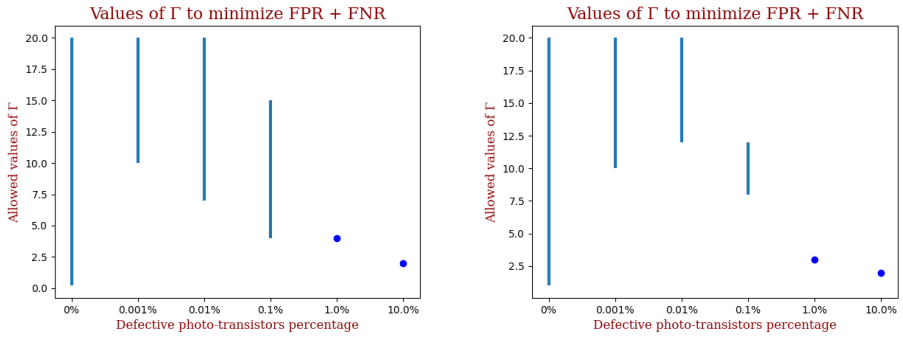
Minimization	Input-Output Comparison	BBS-ESN Training Blocks Order	Noise 1%		Noise 10%	
			FPR	FNR	FPR	FNR
FPR	Baseline feedforward BBS-ESN	Random	0.0	0.6	0.0	41.5
		Lexicographical	0.0	1.8	0.0	41.1
	Quantized feedforward BBS-ESN	Random	0.0	0.3	0.0	43.1
		Lexicographical	0.0	1.8	0.0	27.4
FNR	Baseline feedforward BBS-ESN	Random	0.3	0.0	83.0	0.6
		Lexicographical	0.8	0.0	86.4	0.3
	Quantized feedforward BBS-ESN	Random	0.8	0.0	84.5	0.8
		Lexicographical	1.4	0.0	78.9	0.3
FPR+FNR	Baseline feedforward BBS-ESN	Random	0.3	0.0	3.2	13.0
		Lexicographical	0.3	0.3	4.5	12.3
	Quantized feedforward BBS-ESN	Random	0.0	0.3	3.9	12.0
		Lexicographical	0.3	0.3	4.2	11.7
FPR	Baseline BBS-ESN	Random	0.0	0.0	0.0	28.7
		Lexicographical	0.0	0.0	0.0	28.7
	Quantized BBS-ESN	Random	0.0	0.0	0.0	33.1
		Lexicographical	0.0	0.0	0.0	33.1
FNR	Baseline BBS-ESN	Random	0.0	0.0	83.8	0.3
		Lexicographical	0.0	0.0	83.8	0.3
	Quantized BBS-ESN	Random	0.0	0.0	81.4	0.6
		Lexicographical	0.0	0.0	81.4	0.6
FPR+FNR	Baseline BBS-ESN	Random	0.0	0.0	2.7	13.2
		Lexicographical	0.0	0.0	2.7	13.2
	Quantized BBS-ESN	Random	0.0	0.0	4.1	14.6
		Lexicographical	0.0	0.0	4.1	14.6

Results with defective pixels percentage of 0% have not been reported since they are always FPR = 0.0, FNR = 0.0. The number transient blocks is 50 for each BBS-ESN test. The number of blocks to train the readout is 80 for each BBS-ESN test. The number of images used to train the Evaluator is 100 in every case.

7.2. Analysis of the Results

Table 10 shows the accuracy results for baseline and the quantized BBS-ESN versions and Figure 9 shows Γ values allowing the achievement of those results, for each noise level. For the sake of brevity, only the values used to minimize the sum of FPR and FNR are reported. The Γ coefficients used in this study span from 0.2 to 20.

As the noise increases it becomes increasingly difficult for the network to separate the μ_{I_j} values corresponding to normal and anomalous images (Section 6).



(**a**) (**b**)

Figure 9. Gamma values allowing the results obtained. The values are the same for both the blocks order used for the network training. (**a**) Gamma coefficients for the baseline BBS-ESN allowing the results obtained. (**b**) Gamma coefficients for the quantized BBS-ESN allowing the results obtained.

7.2.1. Robustness to Training Blocks Order

Following some experiments, both the BBS-ESN versions are robust to changes in the readout matrix training blocks order, since the results obtained are the same (at one decimal place in FPR and FNR, as reported in Table 10) in both the train condition.

7.2.2. Comparison with Residual Image Algorithm

The results have been compared with the ones achieved using a Residual Image Algorithm that compares a newly acquired image with a saved reference, similar to the method in [88]. For this algorithm, the Evaluator that has been used follows the same rules of the one used for the BBS-ESN, and it is summarized in Figure 10.

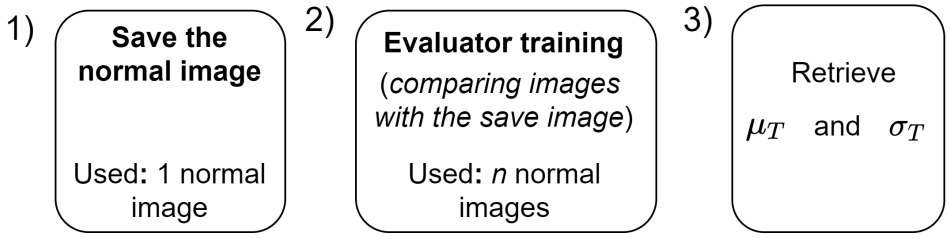


Figure 10. Pipeline for Residual Image Algorithm normal image acquisition (with three sequential blocks (**1**–**3**)) and Evaluator training.

All of BBS-ESN, full precision baseline and quantized versions, have achieved better results than the Residual Image Algorithm. This is due to a greater reconstruction error that BBS-ESN is capable of achieving for anomalous images and has a lower reconstruction error for normal images. This leads to a better separation between the μ_{I_j} from normal and anomalous images (Section 6), leading to overall better performance in terms of FPR and FNR percentages.

8. Feasibility Analysis on a Tiny Micro-Controller

The execution times of BBS-ENS quantized version have been estimated considering the implementation one the micro-controller STM32H743ZI2 running at 480 MHz, featuring ARM Cortex-M7 core, 1 MByte embedded RAM and 2 MByte embedded FLASH. Full precision is out of profiling scope due to the usage of fp64 that reduces performance.

By profiling a variety of neural layers at various bit-depth generated with the plugin X-CUBE-AI v7.0.0 part of STM32CubeMX [89], it has been calculated that by average the execution time on the considered target of a single- precision floating point (fp32) MACC is 10.4 ns, of a binary MACC is 1.5 ns, of a int8 MACC is 6 ns, and finally, a fp32 division takes

29 ns. On the basis of these values, the execution time required for training and inference of BBS-ESN can be estimated. As it is possible to implement up to 4 int8 sums in one clock cycle (using ARM SIMD instructions [90]), the time for an int8 sum can be approximated to be $\frac{1/4}{480\ \text{MHz}} = 0.52$ ns. In a similar way, 2 int16 sums can be performed in 1 clock cycle giving an estimated sum time of $\frac{1/2}{480\ \text{MHz}} = 1.04$ ns.

In this study, binary values (-1 and 1) are packed as 0 and 1 in 32 bit words but they are extended to int8 values in order to perform computations. The time required for packing and unpacking these values to and from 32-bit memory words has not been considered. It is important to notice, also that in these estimates the operations and the times for reading and writing into the registers are not considered, resulting in an optimistic analysis.

In Table 12 are reported the times for the operations spent during training and inference. The time necessary for a binary multiplication has been approximated with the time required for a binary MACC.

Table 12. Times estimated for the single operations spent for training and inference.

Operation	Time [ms]
int8 sum	0.52×10^{-6}
int16 sum	1.04×10^{-6}
bin.$[-1, 1]$ multiplication	1.5×10^{-6} *
fp32 MACC	10.4×10^{-6}
fp32 division	29×10^{-6}

* The time used for the binary multiplication is approximated to the time for a binary MACC.

8.1. Training Time Profile

Table 13 reports the execution times and the number of operations for the readout training. These operations are represented with the types used for the MCU implementation (i.e., accumulations of fp32 with int8 values are implemented with both numbers in fp32).

Table 13. Times required and respective operations for the readout training for the quantized BBS-ESN. The time for converting binary values, packed as 1-bit values, into int8 has not been considered in this estimation.

Position	Times [ms]	Multiply	Accumulate	Division
$\boldsymbol{K = XX^T + I}$	0.042×10^3	20,971,520 (bin.$[-1, 1]$)	20,709,888 (int8)	-
$\boldsymbol{A = K^{-1}}$ *	0.467×10^3 **	44,869,888 (fp32)	44,869,888 (fp32)	131,328 (fp32)
$\boldsymbol{Z = Y_{target}X^T}$	0.021×10^3	10,485,760 (bin.$[-1, 1]$)	10,354,688 (int8)	-
$\boldsymbol{W_{out} = ZA}$	0.698×10^3 **	67,108,864 (fp32)	66,977,792 (fp32)	-
Rescale $\boldsymbol{W_{out}}$	0.004×10^3	-	-	131,072 (fp32)
Total	1.232×10^3	31,457,280 (bin.$[-1, 1]$) 111,978,752 (fp32)	31,064,576 (int8) 111,847,680 (fp32)	262,400 (fp32)

* The Gauss–Jordan method has been considered for the inversion. ** The time required for fp32 multiplications and sums is obtained. approximating the fp32 MACC quantity with the multiplication quantity.

The times for fp32 operations represent the worst case since they do not consider that the number of multiplications are slightly greater than the number of sums. For the estimate of the matrix inversion time, the Gauss–Jordan method has been considered, since it is the method used for the inversion in the ARM CMSIS-5 library [91].

Using 100 images to be trained, the Evaluator would require a training time of 1.206 s.

8.2. Inference Time Profile

The times and operations for the inference for a single block, using the quantized BBS-ESN, are listed in Table 14. The multiplication of the reservoirs state for the reservoir matrices are not reported, since those matrices are set to be equal to identity matrices.

Table 14. Times required and respective operations for one block inference for the quantized BBS-ESN

Operation	Time [ms]	Multiply	Accumulate	Non-Lin.
$\boldsymbol{a}(t) = \boldsymbol{W}_{in}^{(1)}\boldsymbol{u}(t)$	0.034	65,536 (bin.$[-1, 1]$)	65,280 (int8)	-
$\boldsymbol{x}^{(1)}(t) = \text{sign}\big(\boldsymbol{a}(t) + \boldsymbol{x}^{(1)}(t-1)\big)$	0.001 *	-	256 (int8)	sign
$\boldsymbol{c}(t) = \boldsymbol{W}_{in}^{(2)}\boldsymbol{x}^{(1)}(t)$	0.034	65,536 (bin.$[-1, 1]$)	65,280 (int8)	-
$\boldsymbol{x}^{(2)}(t) = \text{sign}\big(\boldsymbol{c}(t) + \boldsymbol{x}^{(2)}(t-1)\big)$	0.001 *	-	256 (int8)	sign
$\boldsymbol{y}(t) = \boldsymbol{W}_{out}\boldsymbol{x}(t)$ **	0.333	131,072 (bin.$[-1, 1]$)	130,816 (int16)	-
$\boldsymbol{o}(t) = \text{sign}\big(\boldsymbol{y}(t)\big)$ ***	-	-	-	sign
Total	0.402	262,144 (bin.$[-1, 1]$)	131,072 (int8) 130,816 (int16)	-

* The time required for the int8 accumulation and sign extraction has been approximated with the time required for a binary multiplication and int8 sum. ** $\boldsymbol{x}(t) = \left[\boldsymbol{x}^{(1)}(t) \quad \boldsymbol{x}^{(2)}(t)\right]^T$. *** The times for the sign extraction of $\boldsymbol{y}(t)$ have been neglected, having overestimated the times for the int8 sum and sign extraction (in the numbers with the * symbol). The multiplication of the reservoirs state for the reservoir matrices are not reported, since we have set those matrices to be equal to identity matrices.

Since the estimated inference time for one block is 0.402 ms and the mean blocks number for the sequences we used is 29.4, then approximated to 30, the time required to perform the inference on one image is estimated to be 12.06 ms (the reason for having different blocks number in different sequences is illustrated in Section 7.1). As for the mean computational complexity, it is estimated with 7,864,320 binary multiplications, 3,932,160 int8 sums and 3,924,480 int16 sums. The time required to compare the result output $\boldsymbol{o}(t)$ with the input $\boldsymbol{u}(t)$ has been neglected, being a simple XOR operation plus a count of the 1 values in output from the XOR, of second order importance versus the other operations performed by the inference. Additionally, the computation of the mean number of these 1 values per block in an image (i.e., the μ_{I_j}, explained in Section 6) has been neglected for the same reason.

8.3. Required Memory

8.3.1. Training

The RAM required to perform the training (as described in Table 13) is reported in Table 15 with all the necessary steps to be executed. The maximum required RAM value is needed for the $\boldsymbol{K}$ matrix inversion and it is 2176 KBytes. This quantity has been estimated in the worst optimization case (i.e., in the need for storing simultaneously every value of $\boldsymbol{K}$ and $\boldsymbol{K}^{-1}$, with $\boldsymbol{K}$ coded as fp32 numbers).

Table 15. RAM required for the training of the BBS-ESN. $\boldsymbol{X}$ is the state matrix (11), $\boldsymbol{Y}_{target}$ is the target matrix (12).

Step	Matrices Needed	Dimension and Type	Required RAM for Matrix [KBytes]	Total Required RAM [KBytes]
Before training	$\boldsymbol{X}$	512×80 (bin.$[-1, 1]$)	5	7.5
	$\boldsymbol{Y}_{target}$	256×80 (bin.$[-1, 1]$)	2.5	
Computation of $\boldsymbol{Z} = \boldsymbol{Y}_{target}\boldsymbol{X}^T$	$\boldsymbol{X}$	512×80 (bin.$[-1, 1]$)	5	135.5
	$\boldsymbol{Y}_{target}$	256×80 (bin.$[-1, 1]$)	2.5	
	$\boldsymbol{Z}$	256×512 (int8)	128	
Computation of $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T + \boldsymbol{I}$	$\boldsymbol{X}$	512×80 (bin.$[-1, 1]$)	5	389
	$\boldsymbol{Z}$	256×512 (int8)	128	
	$\boldsymbol{K}$	512×512 (fp32)	256	
Computation of $\boldsymbol{K}^{-1}$ *	$\boldsymbol{K}$	512×512 (fp32)	1024	2176
	$\boldsymbol{K}^{-1}$	512×512 (fp32)	1024	
	$\boldsymbol{Z}$	256×512 (int8)	128	
Computation of $\boldsymbol{Z}\boldsymbol{K}^{-1} = \boldsymbol{W}_{out}$	$\boldsymbol{K}^{-1}$	512×512 (fp32)	1024	1664
	$\boldsymbol{Z}$	256×512 (int8) **	128	
	$\boldsymbol{Z}\boldsymbol{K}^{-1}$	256×512 (fp32)	512	
Rescaling of $\boldsymbol{W}_{out}$	$\boldsymbol{W}_{out}$	256×512 (fp32)	512	512
Conversion of $\boldsymbol{W}_{out}$ in int8	$\boldsymbol{W}_{out}$	256×512 (fp32)	512	512 ***
	$\boldsymbol{W}_{out}$	256×512 (int8)	128	
Final result: $\boldsymbol{W}_{out}$ in int8	$\boldsymbol{W}_{out}$	256×512 (int8)	128	128
-	-	-	Maximum required RAM	2176

* the RAM for the matrix inversion has been estimated in the worst optimization case (i.e., in the necessity of storing simultaneously every value of $\boldsymbol{K}$ and $\boldsymbol{K}^{-1}$, with $\boldsymbol{K}$ converted in fp32). ** $\boldsymbol{Z}$ can be saved in int8 and every element converted to fp32 before multiplying it. *** 512 KBytes is the maximum RAM required in this step because the elements of $\boldsymbol{W}_{out}$ can be converted to int8 overwriting the fp32 values.

8.3.2. Inference

By implementing the inference with the operations reported in Table 16, then the total RAM required is 129.2 KBytes, considering the readout matrix $\boldsymbol{W}_{out}$ and the input, state and output, respectively $\boldsymbol{u}(t)$, $\boldsymbol{x}(t)$ and $\boldsymbol{y}(t)$. These values are shown in Table 16. As for the Flash memory, the required quantity is 16 KBytes, as reported in Table 7.

Table 16. RAM required to perform the inference with the operations of Table 14.

Matrix or Vector	Dimension and Type	Required RAM [KBytes]
W_{out}	256×512 (int8)	128
$u(t)$ *	256×1 (bin.$[-1, 1]$)	0.03
$x(t)$ *	512×1 (bin.$[-1, 1]$)	0.06
$o(t)$ *	256×1 (bin.$[-1, 1]$)	0.03
-	Total required RAM	129.2

* The elements of $u(t)$, $x(t)$ and $o(t)$ can be saved as binary and converted to int8 performing operations W_{out}.

9. Conclusions

This study proved that the quantized BBS-ESN can be implemented as inference with an estimated number of operations, for each image, of 7,864,320 binary multiplications, 3,932,160 int8 sums and 3,924,480 int16 sums, and with an execution time of 12.06 ms.

The estimates on the required RAM and Flash are respectively of 129.2 KBytes and 16 KBytes. As for the BBS-ESN training, the estimated numbers and types of operations are 31,457,280 binary multiplications, 111,978,752 fp32 multiplications, 31,064,576 int8 sums and 111,847,680 fp32 sums, and with an execution time of 1.232 s. In this case, no additional space is required in Flash and the required RAM is estimated to be 2176 KBytes. Once the network has been trained, the decision process called Evaluator trains to set the reconstruction error threshold, to separate normal from anomalous images, in 1.260 s (Section 8.2).

The accuracy results on the synthetic dataset that have been obtained with the quantized BBS-ESN are of FPR = 0.0 and FNR = 0.0 in the best noise conditions (0%) and of FPR = 4.1 and FNR = 14.6 in the worst noise conditions (10%). These are the best results for the sum of FPR and FNR and they are achieved by setting the Evaluator in order to minimize this sum. In the same conditions, the baseline BBS-ESN has performance of FPR = 0.0, FNR = 0.0 with no noise and FPR = 2.7, FNR = 13.2 with a noise percentage of 10%.

The Flash memory saving for inference, between the baseline and the quantized BBS-ESN, is of 2032 KBytes, comparing the value of Table 7 with the value of Table 3. The RAM saving for inference, in terms of required weights memory, is of 896 KBytes, comparing the value of Table 7 to the value of Table 3.

It is important to note, as stated in Section 8, that the estimates are optimistic since the operations and the time for reading and writing the data in the registers are not considered. Additionally, the time and the operations required to convert binary values into int8, and vice-versa (Section 8) has been neglected. The reason for this is that these operations have a second order importance versus the others. This study has shown that the BBS-ESN deeply quantized can be implemented on an off-the-shelf MCU. This is feasible because the estimated time for the detection on one image is approximately 12.06 ms (Section 8.2) and the images could be acquired every minute or even less frequently, since the relatively slow process of oil leak motion. Slight accuracy differences have been noted between the BBS-ESN and the network without recurrent connections (both in the quantized and baseline versions).

The network training can be optimized using half-precision floating point numbers, or even fixed-point precision numbers. The training states could be reduced in smaller chunks making the training iterative between chunks until all of them are processed and

the weights computed. This is expected to reduce RAM needs. Moreover, more typologies could be tested, changing the combinations of reservoir layers, neurons per reservoir, and blocks dimensions. Finally, the network could be implemented in C-code, and therefore measuring on the real MCU since we presented complexity estimations.

Future works include the validation of the proposed approach on real oil leak images, as soon as they become available, to evaluate the accuracy of the presented network on more irregular samples and to improve it if necessary. Other possible future works regard the integration of the proposed approach with other anomaly detection techniques exploiting information coming from different types of sensors (e.g., pressure, temperature). Combining different techniques based on different information sources can mitigate the effects of degradation of the single sensors, reducing the number of false positives and false negatives.

Author Contributions: Conceptualization, M.C. and D.P.P.; Data curation, M.C.; Investigation, M.C. and D.P.P.; Methodology, M.C., D.P.P. and C.T.; Project administration, D.P.P. and C.T.; Software, M.C., D.P.P. and L.F.; Supervision, D.P.P., L.F., C.T. and M.L.; Validation, M.C., D.P.P., L.F., C.T. and M.L.; Visualization, M.C., D.P.P., L.F., C.T. and M.L.; Writing—original draft, M.C., D.P.P., L.F., C.T. and M.L.; Writing—review and editing, M.C., D.P.P., L.F., C.T. and M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: This work used data publicly available from [79].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Frangoul, A. There Are over 341,000 Wind Turbines on the Planet: Here's How Much of a Difference They're Actually Making. 2017. Available online: https://www.cnbc.com/2017/09/08/there-are-over-341000-wind-turbines-on-the-planet-why-they-matter.html (accessed on 6 September 2021).
2. Anistar Technologies. 5 Biggest Trends in Wind Energy Heading into 2021. 2020. Available online: https://www.anistar.com/hiring/5-biggest-trends-in-wind-energy-heading-into-2021/ (accessed on 6 September 2021).
3. Crespo, A.; Hernández, J.; Frandsen, S. Survey of modelling methods for wind turbine wakes and wind farms. *Wind Energy* **1999**, *2*, 1–24. [CrossRef]
4. Barthelmie, R.J.; Jensen, L.E. Evaluation of wind farm efficiency and wind turbine wakes at the Nysted offshore wind farm. *Wind Energy* **2010**, *13*, 573–586. [CrossRef]
5. Porté-Agel, F.; Bastankhah, M.; Shamsoddin, S. Wind-turbine and wind-farm flows: A review. *Bound.-Layer Meteorol.* **2020**, *174*, 1–59. [CrossRef]
6. Díaz, H.; Guedes Soares, C. Review of the current status, technology and future trends of offshore wind farms. *Ocean. Eng.* **2020**, *209*, 107381. [CrossRef]
7. Ørsted. Walney Extension Offshore Wind Farm. Technical Report, Ørsted. 2013. Available online: https://orstedcdn.azureedge.net/-/media/www/docs/corp/uk/updated-project-summaries-06-19/190515_walney-extension-web_aw.ashx?la=en&rev=ddb582211f2f4c3597a6f3d0293457a9&hash=A7EB7ADD81FA681447BDB0BD1DBB186A (accessed on 6 September 2021).
8. Stetco, A.; Dinmohammadi, F.; Zhao, X.; Robu, V.; Flynn, D.; Barnes, M.; Keane, J.; Nenadic, G. Machine learning methods for wind turbine condition monitoring: A review. *Renew. Energy* **2019**, *133*, 620–635. [CrossRef]
9. Qian, P.; Tian, X.; Kanfoud, J.; Lee, J.L.Y.; Gan, T.H. A Novel Condition Monitoring Method of Wind Turbines Based on Long Short-Term Memory Neural Network. *Energies* **2019**, *12*, 3411. [CrossRef]
10. Black, I.M.; Richmond, M.; Kolios, A. Condition monitoring systems: A systematic literature review on machine-learning methods improving offshore-wind turbine operational management. *Int. J. Sustain. Energy* **2021**, *40*, 1–24. [CrossRef]
11. Zhao, H.; Liu, H.; Hu, W.; Yan, X. Anomaly detection and fault analysis of wind turbine components based on deep learning network. *Renew. Energy* **2018**, *127*, 825–834. [CrossRef]
12. Chang, Y.; Chen, J.; Qu, C.; Pan, T. Intelligent fault diagnosis of Wind Turbines via a Deep Learning Network Using Parallel Convolution Layers with Multi-Scale Kernels. *Renew. Energy* **2020**, *153*, 205–213. [CrossRef]
13. Yu, W.; Huang, S.; Xiao, W. Fault Diagnosis Based on an Approach Combining a Spectrogram and a Convolutional Neural Network with Application to a Wind Turbine System. *Energies* **2018**, *11*, 2561. [CrossRef]
14. Jiang, G.; He, H.; Yan, J.; Xie, P. Multiscale Convolutional Neural Networks for Fault Diagnosis of Wind Turbine Gearbox. *IEEE Trans. Ind. Electron.* **2019**, *66*, 3196–3207. [CrossRef]
15. Gao, Y.Y.; Zhang, W.Z.; Wang, H. The Influence of Gearbox Oil-Leak to the Bolt Joint in Wind Turbine Tower. In *Advanced Materials Research*; Trans Tech Publications Ltd.: Freienbach, Switzerland, 2013; Volume 744, pp. 87–90.

16. del Álamo, J.R.; Duran, M.J.; Muñoz, F.J. Analysis of the Gearbox Oil Maintenance Procedures in Wind Energy. *Energies* **2020**, *13*, 3414. [CrossRef]
17. Zeng, K.; Wang, Y. A Deep Convolutional Neural Network for Oil Spill Detection from Spaceborne SAR Images. *Remote Sens.* **2020**, *12*, 1015. [CrossRef]
18. Solberg, A.; Storvik, G.; Solberg, R.; Volden, E. Automatic detection of oil spills in ERS SAR images. *IEEE Trans. Geosci. Remote Sens.* **1999**, *37*, 1916–1924. [CrossRef]
19. Kubát, M.; Holte, R.; Matwin, S. Machine Learning for the Detection of Oil Spills in Satellite Radar Images. *Mach. Learn.* **2004**, *30*, 195–215. [CrossRef]
20. Brekke, C.; Solberg, A.H. Oil spill detection by satellite remote sensing. *Remote Sens. Environ.* **2005**, *95*, 1–13. [CrossRef]
21. Cantorna, D.; Dafonte, C.; Iglesias, A.; Arcay, B. Oil spill segmentation in SAR images using convolutional neural networks. A comparative analysis with clustering and logistic regression algorithms. *Appl. Soft Comput.* **2019**, *84*, 105716. [CrossRef]
22. Adler, A.; Boublil, D.; Elad, M.; Zibulevsky, M. A Deep Learning Approach to Block-based Compressed Sensing of Images. *arXiv* **2016**, arXiv:1606.01519.
23. Ahn, B.; Cho, N.I. Block-Matching Convolutional Neural Network for Image Denoising. *arXiv* **2017**, arXiv:1704.00524.
24. Li, H.; Wang, S.; Kot, A. Image Recapture Detection with Convolutional and Recurrent Neural Networks. *Electron. Imaging* **2017**, *2017*, 87–91. [CrossRef]
25. Maleki, D.; Nadalian, S.; Derakhshani, M.M.; Sadeghi, M.A. BlockCNN: A Deep Network for Artifact Removal and Image Compression. *arXiv* **2018**, arXiv:1805.11091.
26. Quijas, J.; Fuentes, O. Removing JPEG blocking artifacts using machine learning. In Proceedings of the 2014 Southwest Symposium on Image Analysis and Interpretation, San Diego, CA, USA, 6–8 April 2014; pp. 77–80.
27. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly Detection: A Survey. *ACM Comput. Surv.* **2009**, *41*, 1–58. [CrossRef]
28. Zimek, A.; Schubert, E.; Kriegel, H.P. A survey on unsupervised outlier detection in high-dimensional numerical data. *Stat. Anal. Data Mining Asa Data Sci. J.* **2012**, *5*, 363–387. [CrossRef]
29. Gupta, M.; Gao, J.; Aggarwal, C.C.; Han, J. Outlier Detection for Temporal Data: A Survey. *IEEE Trans. Knowl. Data Eng.* **2014**, *26*, 2250–2267. [CrossRef]
30. Hodge, V.; Austin, J. A survey of outlier detection methodologies. *Artif. Intell. Rev.* **2004**, *22*, 85–126. [CrossRef]
31. Keshk, M.; Sitnikova, E.; Moustafa, N.; Hu, J.; Khalil, I. An Integrated Framework for Privacy-Preserving Based Anomaly Detection for Cyber-Physical Systems. *IEEE Trans. Sustain. Comput.* **2021**, *6*, 66–79. [CrossRef]
32. Huang, D.; Mu, D.; Yang, L.; Cai, X. CoDetect: Financial Fraud Detection With Anomaly Feature Detection. *IEEE Access* **2018**, *6*, 19161–19174. [CrossRef]
33. Quellec, G.; Lamard, M.; Cozic, M.; Coatrieux, G.; Cazuguel, G. Multiple-Instance Learning for Anomaly Detection in Digital Mammography. *IEEE Trans. Med. Imaging* **2016**, *35*, 1604–1614. [CrossRef]
34. Rasheed, W.; Tang, T.B. Anomaly Detection of Moderate Traumatic Brain Injury Using Auto-Regularized Multi-Instance One-Class SVM. *IEEE Trans. Neural Syst. Rehabil. Eng.* **2020**, *28*, 83–93. [CrossRef]
35. Lu, J.; Liang, B.; Lei, Q.; Li, X.; Liu, J.; Liu, J.; Xu, J.; Wang, W. SCueU-Net: Efficient Damage Detection Method for Railway Rail. *IEEE Access* **2020**, *8*, 125109–125120. [CrossRef]
36. Li, Z.; Zhang, Y. Hyperspectral Anomaly Detection via Image Super-Resolution Processing and Spatial Correlation. *IEEE Trans. Geosci. Remote Sens.* **2021**, *59*, 2307–2320. [CrossRef]
37. Wang, Y.; Liu, Z.; Wang, D.; Li, Y.; Yan, J. Anomaly Detection and Visual Perception for Landslide Monitoring Based on a Heterogeneous Sensor Network. *IEEE Sens. J.* **2017**, *17*, 4248–4257. [CrossRef]
38. Kim, H.; Park, J.; Min, K.; Huh, K. Anomaly Monitoring Framework in Lane Detection With a Generative Adversarial Network. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 1603–1615. [CrossRef]
39. Siegel, B. Industrial Anomaly Detection: A Comparison of Unsupervised Neural Network Architectures. *IEEE Sens. Lett.* **2020**, *4*, 1–4. [CrossRef]
40. Jiang, Y.; Wang, W.; Zhao, C. A Machine Vision-based Realtime Anomaly Detection Method for Industrial Products Using Deep Learning. In Proceedings of the 2019 Chinese Automation Congress (CAC), Hangzhou, China, 22–24 November 2019; pp. 4842–4847.
41. Muramatsu, H.; Kakimi, Y.; Takeuchi, I.; Katsura, S. Package Leak Detection Based on Aperiodic Anomaly Extraction. *IEEE J. Emerg. Sel. Top. Ind. Electron.* **2021**, *2*, 363–370. [CrossRef]
42. Peng, Y.; Ruan, S.; Cao, G.; Huang, S.; Kwok, N.; Zhou, S. Automated Product Boundary Defect Detection Based on Image Moment Feature Anomaly. *IEEE Access* **2019**, *7*, 52731–52742. [CrossRef]
43. Wu, D.; Jiang, Z.; Xie, X.; Wei, X.; Yu, W.; Li, R. LSTM Learning With Bayesian and Gaussian Processing for Anomaly Detection in Industrial IoT. *IEEE Trans. Ind. Inform.* **2020**, *16*, 5244–5253. [CrossRef]
44. Lu, H.; Liu, Y.; Fei, Z.; Guan, C. An Outlier Detection Algorithm Based on Cross-Correlation Analysis for Time Series Dataset. *IEEE Access* **2018**, *6*, 53593–53610. [CrossRef]
45. Rasheed, F.; Alhajj, R. A Framework for Periodic Outlier Pattern Detection in Time-Series Sequences. *IEEE Trans. Cybern.* **2014**, *44*, 569–582. [CrossRef]
46. Wang, S.; Li, C.; Lim, A. A Model for Non-Stationary Time Series and its Applications in Filtering and Anomaly Detection. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 1–11. [CrossRef]

47. Hu, M.; Ji, Z.; Yan, K.; Guo, Y.; Feng, X.; Gong, J.; Zhao, X.; Dong, L. Detecting Anomalies in Time Series Data via a Meta-Feature Based Approach. *IEEE Access* **2018**, *6*, 27760–27776. [CrossRef]
48. Akouemo, H.N.; Povinelli, R.J. Data Improving in Time Series Using ARX and ANN Models. *IEEE Trans. Power Syst.* **2017**, *32*, 3352–3359. [CrossRef]
49. Karadayı, Y.; Aydin, M.N.; Öğrenci, A.S. A Hybrid Deep Learning Framework for Unsupervised Anomaly Detection in Multivariate Spatio-Temporal Data. *Appl. Sci.* **2020**, *10*, 5191. [CrossRef]
50. Deecke, L.; Ruff, L.; Vandermeulen, R.A.; Bilen, H. Deep Anomaly Detection by Residual Adaptation. *arXiv* **2020**, arXiv:2010.02310.
51. Pang, G.; Shen, C.; Cao, L.; van den Hengel, A. Deep Learning for Anomaly Detection: A Review. *arXiv* **2020**, arXiv:2007.02500.
52. Ye, F.; Zheng, H.; Huang, C.; Zhang, Y. Deep Unsupervised Image Anomaly Detection: An Information Theoretic Framework. *arXiv* **2020**, arXiv:2012.04837.
53. Abati, D.; Porrello, A.; Calderara, S.; Cucchiara, R. Latent Space Autoregression for Novelty Detection. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 481–490.
54. LeCun, Y.; Bengio, Y.; Hinton, G. Deep Learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
55. Hilt, D.E.; Seegrist, D.W. *Ridge, a Computer Program for Calculating Ridge Regression Estimates*; Department of Agriculture, Forest Service, Northeastern Forest Experiment: Washington, DC, USA, 1977; Volume 236.
56. Huang, G.B.; Chen, L.; Siew, C.K. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Netw.* **2006**, *17*, 879–892. [CrossRef]
57. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [CrossRef]
58. Huang, G.B.; Zhou, H.; Ding, X.; Zhang, R. Extreme Learning Machine for Regression and Multiclass Classification. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **2012**, *42*, 513–529. [CrossRef]
59. Huang, G.B. What are Extreme Learning Machines? Filling the Gap Between Frank Rosenblatt's Dream and John von Neumann's Puzzle. *Cogn. Comput.* **2015**, *7*, 263–278. [CrossRef]
60. Huang, G.B. An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels. *Cogn. Comput.* **2014**, *6*, 376–390. [CrossRef]
61. Park, J.; Cho, D.; Kim, S.; Kim, Y.B.; Kim, P.Y.; Kim, H.J. utilizing online learning based on echo-state networks for the control of a hydraulic excavator. *Mechatronics* **2014**, *24*, 986–1000. [CrossRef]
62. Waegeman, T.; wyffels, F.; Schrauwen, B. Feedback Control by Online Learning an Inverse Model. *IEEE Trans. Neural Netw. Learn. Syst.* **2012**, *23*, 1637–1648. [CrossRef] [PubMed]
63. Jordanou, J.P.; Antonelo, E.A.; Camponogara, E. Online learning control with Echo State Networks of an oil production platform. *Eng. Appl. Artif. Intell.* **2019**, *85*, 214–228. [CrossRef]
64. Jaeger, H. *The "Echo State" Approach to Analysing and Training Recurrent Neural Networks-With an Erratum Note*; German National Research Center for Information Technology GMD Technical Report; German National Research Center for Information Technology: Bonn, Germany, 2001; Volume 148.
65. Lukoševičius, M.; Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.* **2009**, *3*, 127–149. [CrossRef]
66. Gallicchio, C.; Micheli, A. Architectural and Markovian factors of echo state networks. *Neural Netw.* **2011**, *24*, 440–456. [CrossRef]
67. Gallicchio, C.; Micheli, A.; Pedrelli, L. Deep reservoir computing: A critical experimental analysis. *Neurocomputing* **2017**, *268*, 87–99. [CrossRef]
68. Gallicchio, C.; Micheli, A. Deep Echo State Network (DeepESN): A Brief Survey. *arXiv* **2017**, arXiv:1712.04323.
69. Qin, H.; Gong, R.; Liu, X.; Bai, X.; Song, J.; Sebe, N. Binary Neural Networks: A Survey. *arXiv* **2020**, arXiv:2004.03333.
70. Courbariaux, M.; Bengio, Y. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or −1. *arXiv* **2016**, arXiv:1602.02830.
71. Kleyko, D.; Frady, E.P.; Kheffache, M.; Osipov, E. Integer Echo State Networks: Efficient Reservoir Computing for Digital Hardware. *arXiv* **2020**, arXiv:1706.00280.
72. Suh, S.; Chae, D.H.; Kang, H.G.; Choi, S. Echo-state conditional variational autoencoder for anomaly detection. In Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 24–29 July 2016; pp. 1015–1022.
73. Chen, Z.; Yeo, C.K.; Lee, B.S.; Lau, C.T. Autoencoder-based network anomaly detection. In Proceedings of the 2018 Wireless Telecommunications Symposium (WTS), Phoenix, AZ, USA, 17–20 April 2018; pp. 1–5.
74. Basseville, M.; Nikiforov, I.V. *Detection of Abrupt Changes: Theory and Application*; Prentice Hall Englewood Cliffs: Hoboken, NJ, USA, 1993; Volume 104.
75. Alippi, C.; Boracchi, G.; Roveri, M. Change detection tests using the ICI rule. In Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 18–23 July 2010; pp. 1–7. [CrossRef]
76. Chen, Q.; Zhang, A.; Huang, T.; He, Q.; Song, Y. Imbalanced dataset-based echo state networks for anomaly detection. *Neural Comput. Appl.* **2020**, *32*, 3685–3694. [CrossRef]
77. Heim, N.; Avery, J.E. Adaptive Anomaly Detection in Chaotic Time Series with a Spatially Aware Echo State Network. *arXiv* **2019**, arXiv:1909.01709.

78. Obst, O.; Wang, X.R.; Prokopenko, M. Using Echo State Networks for Anomaly Detection in Underground Coal Mines. In Proceedings of the 2008 International Conference on Information Processing in Sensor Networks (ipsn 2008), St. Louis, MO, USA, 22–24 April 2008; pp. 219–229.
79. Cardoni, M.; Pau, D.; Falaschetti, L.; Turchetti, C.; Lattuada, M. Oil Leak Dataset, Mendeley Data, V1. Available online: https://data.mendeley.com/datasets/nbxzxn3ffk/1 (accessed on 7 October 2021).
80. Cardoni, M.; Pau, D.; Falaschetti, L.; Turchetti, C.; Lattuada, M. Synthetic image dataset of shaft junctions inside wind turbines in presence or absence of oil leaks. *Data Brief* **2021**, *39*, 107538. [CrossRef]
81. Matlab. rgb2gray API. Available online: https://it.mathworks.com/help/matlab/ref/rgb2gray.html (accessed on 7 October 2021).
82. Otsu, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Trans. Syst. Man Cybern.* **1979**, *9*, 62–66. [CrossRef]
83. Fadnavis, S. Image Interpolation Techniques in Digital Image Processing: An Overview. *Int. J. Eng. Res. Appl.* **2014**, *4*, 2248–2270.
84. van der Maaten, L.; Hinton, G. Visualizing Data using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.
85. Matsumoto, M.; Nishimura, T. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Trans. Model. Comput. Simul.* **1998**, *8*, 3–30. [CrossRef]
86. Van der Walt, S.; Schönberger, J.L.; Nunez-Iglesias, J.; Boulogne, F.; Warner, J.D.; Yager, N.; Gouillart, E.; Yu, T. scikit-image: image processing in Python. *PeerJ* **2014**, *2*, e453. [CrossRef]
87. O'Neill, M.E. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*; Technical Report HMC-CS-2014-0905; Harvey Mudd College: Claremont, CA, USA, 2014.
88. Ehret, T.; Davy, A.; Delbracio, M.; Morel, J.M. How to Reduce Anomaly Detection in Images to Anomaly Detection in Noise. *Image Process. Line* **2019**, *9*, 391–412. [CrossRef]
89. STMicroelectronics. X-CUBE-AI, AI Expansion Pack for STM32CubeMX. Available online: https://www.st.com/en/embedded-software/x-cube-ai.html (accessed on 7 October 2021).
90. ARM. CMSIS-Core (Cortex-M)—Intrinsic Functions for SIMD Instructions. Available online: https://www.keil.com/pack/doc/CMSIS/Core/html/group__intrinsic__SIMD__gr.html (accessed on 7 October 2021).
91. ARM. CMSIS Version 5. Available online: https://github.com/ARM-software/CMSIS_5 (accessed on 7 October 2021).