Published version available at: https://doi.org/10.1016/j.eswa.2020.113378

Scalable Auto-Encoders for Gravitational Waves Detection from Time Series Data

Roberto Corizzo[a,b,d,*], Michelangelo Ceci[b,d,e], Eftim Zdravevski[c], Nathalie Japkowicz[a]

[a]*Department of Computer Science, American University, 4400 Massachusetts Ave NW, Washington, DC 20016, United States*
[b]*Department of Computer Science, University of Bari Aldo Moro, Via E. Orabona, 4, 70125 Bari, Italy*
[c]*Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Rugjer Boshkovik 16, 1000 Skopje, North Macedonia*
[d]*National Interuniversity Consortium for Informatics (CINI), Via Volturno, 58, 00185 Roma, Italy*
[e]*Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia*

Abstract

Gravitational waves represent a new opportunity to study and interpret phenomena from the universe. In order to efficiently detect and analyze them, advanced and automatic signal processing and machine learning techniques could help to support standard tools and techniques. Another challenge relates to the large volume of data collected by the detectors on a daily basis, which creates a gap between the amount of data generated and effectively analyzed. In this paper, we propose two approaches involving deep auto-encoder models to analyze time series collected from Gravitational Waves detectors and provide a classification label (noise or real signal). The purpose is to discard noisy time series accurately and identify time series that potentially contain a real phenomenon. Experiments carried out on three datasets show that the proposed approaches implemented using the Apache Spark framework, represent a valuable machine learning tool for astrophysi-

*Corresponding author
Email addresses: rcorizzo@american.edu - Phone: +1 (202) 849-1080 (Roberto Corizzo), michelangelo.ceci@uniba.it (Michelangelo Ceci), eftim@finki.ukim.mk (Eftim Zdravevski), japkowic@american.edu (Nathalie Japkowicz)

cal analysis, offering competitive accuracy and scalability performances with respect to state-of-the-art methods.

---

1. Introduction

In September 2015, Gravitational Waves (GWs) were observed for the first time in the universe, following to Albert Einstein's intuition dated a hundred years ago, and described in his general theory of relativity.

During the first observation run (O1) of Advanced Laser Interferometer Gravitational-Wave Observatory (LIGO), which took place from September 2015 to January 2016, the first detections of GWs from stellar-mass binary black holes (BBHs) took place, as documented by Abbott et al. (2016c), Abbott et al. (2016a), and Abbott et al. (2016b). The second observation run (O2) started in November 2016, and ended in August 2017, when the first binary neutron star inspiral was observed by Abbott et al. (2016d) and Abbott et al. (2018).

GWs are the manifestation of disruptions in spacetime caused by accelerating masses, and they represent a totally new opportunity to study and interpret phenomena from the universe.

For this reason, GWs are attracting increasing interest in the fields of astrophysics. The adoption of automatic signal processing and machine learning techniques could be beneficial for their detection and analysis, and reduce the effort and the cost of standard approaches such as template matching. Such techiniques have been investigated by George & Huerta (2018), Zevin et al. (2017), and Bahaadini et al. (2018a). However, these two tasks (detection and analysis) pose several challenges and require specific expertise, since the data collected by detectors (interferometers) are in the form of time series affected by the presence of environmental and instrumental noise. Fig. 1 (left) shows that the strain time series representing real GWs immersed in noise are impossible to distinguish, even for the human eye, without data pre-processing. Fig. 1 (right) instead, shows the whitened time series representation of the same GWs signal presented in Fig. 1 (left) (ID GW151226).

This implies that, for instance, in order to effectively perform manual filtering, knowledge about the underlying type of noise present in data is required, as well as which frequencies could be relevant (or irrelevant) for the detection of a GW. Additional knowledge is required to perform other recurrent tasks, such as spectrogram analysis, filtering, and whitening, as

outlined by Cuoco et al. (2001a) and Cuoco et al. (2001b). Moreover, even though a large volume of data, in terms of petabytes per day, are collected by detectors, there is a consistent gap between the amount of data generated and adequately analyzed. This volume requires new automatic and scalable methods, which are capable of analyzing data in a timely manner, performing pre-processing and detection tasks that were typically executed manually.

In this paper, we address the challenges related to the GWs detection process from a machine learning perspective. More specifically, we propose two approaches involving deep auto-encoder models to analyze time series collected from detectors and provide a classification label (noise or real signal). The purpose is to discard noisy time series accurately, and identify potentially interesting time series that could be manually inspected.

The paper is structured as follows. In Section II, we provide an overview of related machine learning approaches for GWs detection and anomaly detection. In Section III, we present our proposed method. In Section IV we describe the datasets and the experimental results obtained in our study. Finally, Section V concludes the paper.

2. Related Works

The roots of this work are in astrophysical data analysis and in classification and anomaly detection of time series data. In the following, we discuss the related work in both research topics.

2.1. Astrophysical data analysis

In astrophysical data analysis, one of the most difficult challenges is to work with data in the form of time series characterized by high levels of noise.

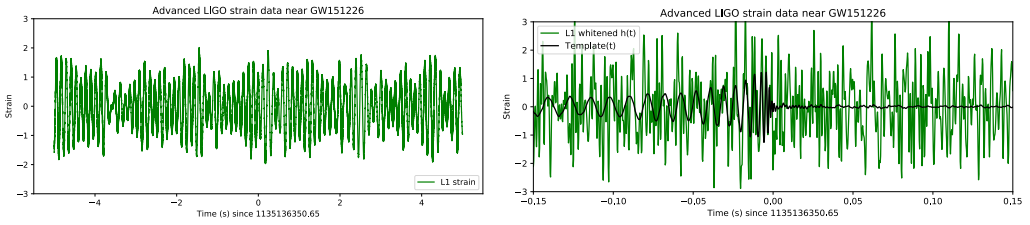

Figure 1: (left) Example of a time series containing a GWs signal immersed in LIGO noise (ID GW151226). Since data are dominated by noise, it is not possible to see the signal without some signal processing. (right) Whitened time series containing a GWs signal (ID GW151226) and the best matching template identified in the LIGO search pipeline.

Time-frequency representation of the data is one of the techniques used for detector characterization purposes and can represent the same information in a different domain (see Fig. 2). Data whitening, addressed by Cuoco et al. (2001a) and Cuoco et al. (2001b), is a typical step in astrophysical data analysis for detection and parameter estimation.

There are techniques which implement the whitening in time domain, as done by Cuoco et al. (2001a), or in frequency domain, as explained by Biwer et al. (2019). In frequency domain we can whiten the data by dividing each point by the noise amplitude spectrum in the Fourier domain. The purpose of the whitening procedure it to remove all the stationary noise, in order to possibly reveal weak signals hidden in other bands (see Fig. 2 right). One additional step is removing high-frequency noise by applying a bandpass filter to the data. In Fig. 2, a Butterworth bandpass filter has been applied to filter signal frequencies below 43Hz and above 300Hz. More details about typical signal processing techniques involved in gravitational waves analysis can be found in a study conducted by Allen & Romano (1999).

Data analysis for GWs requires to perform different tasks, such as noise removal, anomaly detection, and classification, which can all be performed applying machine learning techniques. In the remainder of the paper, we mainly discuss approaches that solve the classification task, which is the main objective of the present study.

In the GW literature, classification algorithms are either used to distinguish real signal from noise or to identify glitches[1]. In both cases, we can distinguish methods that work on spectrogram data (images) and methods that work directly on time series data.

Focusing on methods that work on spectrogram data, one of the tasks addressed in Gravity Spy by Zevin et al. (2017) and Bahaadini et al. (2018a) was to accurately classify glitches using convolutional neural networks, in order to subsequently avoid false positive gravitational wave detections. Supervised convolutional neural networks for glitch classification were also addressed by Razzano & Cuoco (2018) and Gabbard et al. (2018a). In their approaches, the network is trained via transfer learning, and used as a feature extractor for unsupervised clustering methods to discover new classes of glitches based on their morphology. A multi-view deep convolutional neural network for glitch classification was proposed by Bahaadini et al. (2017). The model

[1]Glitches are transient and non-Gaussian disturbances that mimic GWs morphology.

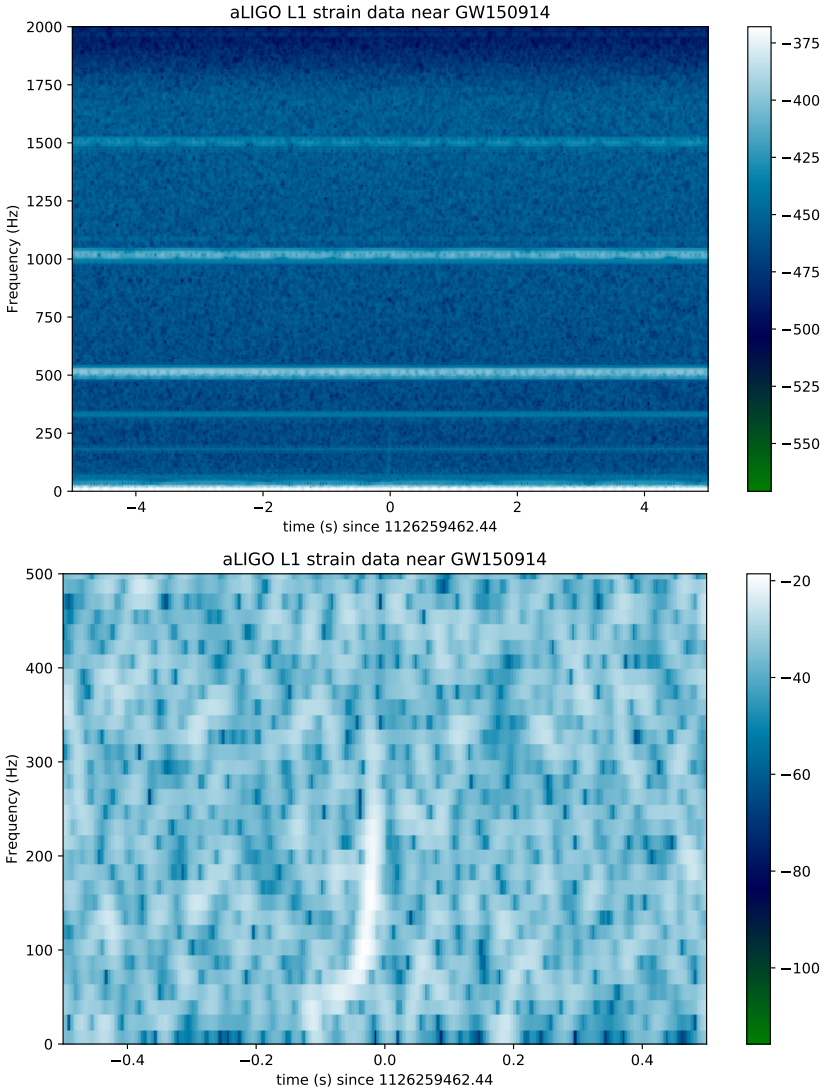

Figure 2: A raw spectrogram representation of the event GW150914 (top) and its whitened version (bottom). In the whitened version it is possible to clearly identify the gravitational wave corresponding to the event BBH (binary black hole mergers). These representations are extracted following the LOSC event tutorial (see `https://www.gw-openscience.org/s/events/GW150914/LOSC_Event_tutorial_GW150914.html`).

exploits four different time durations available for each glitch, from 0.5 to 4 seconds. Bahaadini et al. (2018b) focused on learning discriminative embedding functions for feature extraction. The task is domain adaptation, using a labeled set of glitch classes as a source domain and a pool of unlabeled glitch samples as a target domain (glitch classes).

The proposed methods are capable of obtaining nearly perfect classification accuracy, although data are represented in the form of cleaned spectrograms, that is, images in which glitches are clearly visible, thanks to manual operations such as filtering for spectral lines removal, whitening, and so forth. In Fig. 2, the left part shows a raw spectrogram, while the right part shows a GW emerging after bandpass filtering and whitening operations. In this figure, it looks clear that, with no such pre-processing, any phenomenon is not clearly visible in the raw data representation.

However, assuming that human-intensive pre-processing operations are always feasible is a simplistic assumption, since detectors continuously collect data in the form of strain time series that include noise, and a classification is required in real-time. In this paper, we address the problem of working directly on strain data, and we assess model performance in different conditions.

Shifting the focus to methods that work on time series data, noise classification approaches were proposed by Powell et al. (2017) and Mukherjee et al. (2010), who applied feature extraction techniques to data. In particular, principal component analysis and wavelet transform were considered to extract coefficients that are used as features to train a machine learning classifier. Convolutional neural networks for detection and parameter estimation were proposed by George & Huerta (2018) and Gabbard et al. (2018b), who performed experiments on datasets of waveform templates revealed that the proposed model is capable of obtaining similar performances compared to matched-filtering while being far more computationally efficient. A fully convolutional neural network architecture with dilated kernels that works directly on time series strain data to identify simulated GWs signals was proposed by Gebhard et al. (2017). A different approach was proposed by Shen et al. (2017), where a denoising auto-encoder based on sequence-to-sequence bi-directional Long-Short-Term-Memory recurrent neural networks is investigated. This work shows the superior effectiveness of auto-encoders trained on Gaussian noise, to extract cleaned GW signals from noisy time series collected on a single LIGO detector.

2.2. Classification and anomaly detection from time series data

Methods for time series classification can be divided into three main categories according to Xing et al. (2010): feature-based, distance-based and model-based. Feature-based methods transform time series into a feature vector which can be processed by conventional classification methods, such as decision trees, neural networks or Support Vector Machines. Bagnall et al. (2017) performed an extensive experimental study with methods in this category on a large number of datasets, showing that ensemble methods, such as Random Forest, obtain the best overall performance.

Distance-based methods work by defining a distance function to measure the similarity between a pair of time series and exploit such a distance function for classification. The k-Nearest Neighbor classifier (kNN) falls into this category. Finally, model-based methods adopt generative models, such as Naive Bayes or Hidden Markov Models (HMM), which assume that such underlying models generate the time series of a class.

Supervised time series classification methods are suitable when the available data consists of fully labeled training and test data sets. Common approaches in the literature for this task are based on neural networks and support vector machines, as outlined by Goldstein & Uchida (2016).

Unlike them, unsupervised methods present a great potential in many real-world domains and tasks in which data labels are not available. This problem is particularly important in the context of data streams since the classification is required in the same moment data are received. In particular, the study by Japkowicz (2001) demonstrated that when the data analyzed belongs to two classes, the binary classification task can be reformulated as an unsupervised anomaly detection task. In this case, the purpose is to discriminate among two classes by analyzing data over time and detecting if the current behavior is as expected (normal) or it deviates from the expected distribution (anomaly).

In the anomaly detection task, Chandola et al. (2009) classified anomalies in three general classes: point, contextual, and collective anomalies. This paper focuses on collective anomalies, that is, a collection of data points with respect to the entire time series analyzed, that are anomalies. The conceptualization of anomaly is clearly related to the specific task and problem addressed, and is related to its data structure.

In this context, different studies in recent literature, conducted by Najafabadi et al. (2015), Zhou & Paffenroth (2017), Principi et al. (2019), Chen

et al. (2019), and Corizzo et al. (2019) demonstrated the high performance of auto-encoders. The success of auto-encoders is also theoretically motivated by their inherent ability to learn representations with a low reconstruction error, by means of non-linear combinations of input characteristics, as explained by Y. Bengio et al. (2009).

2.3. Contribution

The reconstruction-based modeling capabilities of auto-encoders and their recognized potential in solving tasks such as data denoising, as shown by Shen et al. (2017), anomaly detection, as shown by Corizzo et al. (2019); Principi et al. (2019); An & Cho (2015); Zhou & Paffenroth (2017); Khan & Taati (2017) and feature extraction, as shown by Corizzo et al. (2019); Masci et al. (2011) with time series data, make them valid candidates for the analysis of astrophysical data, collected in the form of time series characterized by high levels of noise.

Following this analysis, our paper proposes two approaches (one unsupervised and one supervised) involving auto-encoder models in combination with classification models. Our aim is to analyze time series and classify them as noise or real signals (e.g., glitches).

The main reason that led us to prefer classification over a denoising approach is that, in this way, we can perform automatic analysis of data continuously observed in form of time series, and enable astro physicians and data scientists to focus only on potentially interesting time series and further process only them with already established methods. On the other hand, a denoising approach would still require manual intervention in order to filter out time series related to empty signals, and maintain time series that contain significant signals.

More specifically, the machine learning approaches for GWs detection we propose are able to: *i)* detect GWs automatically, directly from time series data; *ii)* recognize as many noise types as possible, being robust to unseen data from different distributions; *iii)* efficiently scale as the volume of data increases, with a distributed implementation that exploits the Apache Spark framework.

Differently from the approaches by Zevin et al. (2017), Bahaadini et al. (2018a), Razzano & Cuoco (2018), Gabbard et al. (2018a), Bahaadini et al. (2017) and Bahaadini et al. (2018b) described above, our approaches do not require any pre-processing step, such as the generation of spectrograms, filtering and whitening, to be performed on data collected by detectors.

From a model perspective, approaches based on auto-encoders were proposed by Shen et al. (2017), but applied for the denoising task. Focusing on the classification task, the most similar works with respect to our proposal are those proposed by George & Huerta (2018) and Gebhard et al. (2017). However, the focus in the work by George & Huerta (2018) and Gebhard et al. (2017) is on a supervised approach based on convolutional neural networks, whereas in this paper we propose two different approaches (unsupervised and supervised) for GWs detection based on auto-encoder neural networks, and we compare their performances, also with convolutional neural networks.

Another significant difference with respect to the study carried out by Gebhard et al. (2017) is that we do not analyze simulated waveforms, but real annotated GWs events, whereas Gebhard et al. (2017) simulated waveforms are injected in time series recorded from Hanford and Livingstone detectors.

Finally, to the best of our knowledge, this is the first paper that proposes auto-encoders for GWs classification on time series data. Moreover, on the contrary of existing works, we tackle the issue of processing and analyzing the high-volume of data collected by detectors. From this perspective, our effort consists in the proposal of a distributed implementation in the Apache Spark framework that is capable of processing large-scale data efficiently.

Although model architectures based on convolutional neural networks and recurrent neural networks are typically more suitable for data including temporal information, this implies in a higher model complexity, which results in a higher training time. With the aim to provide a valid tradeoff between accuracy and scalability, we concentrate on a simpler model, i.e. autoencoders, since they can be efficiently trained in a distributed manner on a cluster of computational nodes.

3. Method

In this section, we describe the two proposed approaches for GWs detection in time series. The first approach exploits auto-encoder models to classify strain time series data with an unsupervised anomaly detection strategy, whereas the second one uses auto-encoders for feature extraction and performs supervised classification.

3.1. Time series classification via anomaly detection (AE)

In this approach, we adopt auto-encoders as in Y. Bengio et al. (2009), motivated by their recognized effectiveness when learning to reconstruct a

specific input data representation with high accuracy, as demonstrated by Najafabadi et al. (2015), and for their potential in terms of feature extraction capabilities, as shown by Hinton & Salakhutdinov (2006). This feature has been used in the literature for anomaly detection, as shown by Japkowicz (2001) and Zhou & Paffenroth (2017), by analyzing the reconstruction error. The approach consists in training the auto-encoder with one-class data. Afterward, for a new data instance at prediction time, if a high reconstruction error of the auto-encoder is observed, then this data instance is assumed to belong to a different data distribution than that associated to training data. When this situation happens, the new data instance is labeled as an anomaly, i.e. a data instance that potentially contain a real GWs signal. The rationale of this approach is that we can train an auto-encoder with data representing different types of noise (i.e., the negative class, also regarded as the normal class) which are abundant, in contrast to few GWs signals observed so far, and classify unseen time series by evaluating their reconstruction error given by the auto-encoder model. Typically, noise is continuously observed, whereas GW signals appear in a limited time frame of a time series. According to the anomaly detection setting, a time series classified as anomaly corresponds to a time series that contains a real GWs signal.

Each auto-encoder has an encoding function γ and a decoding function δ. The goal of the auto-encoder is the following:

$$\begin{aligned} \gamma : \mathcal{X} \to \mathcal{F}, \qquad \delta : \mathcal{F} \to \mathcal{X}, \\ \gamma, \delta = \arg\min_{\gamma,\delta} \|X - \delta(\gamma(X))\|^2, \end{aligned} \tag{1}$$

where $\mathcal{X}$ is the data input space (each instance is representing a time series), and $\mathcal{F}$ is the encoding space (features learned by the auto-encoder).

The functions γ and δ are chosen to be parametric and differentiable with respect to a distance function. In this way, the parameters of the encoding and decoding functions can be optimized in order to minimize the reconstruction loss.

The learning process for an auto-encoder takes place via backpropagation. The process results in a first layer trained with raw data, which extracts new vectors of reduced dimensionality, by means of the activation function of the hidden neurons.

The architecture can consist of multiple hidden layers. In the case of two hidden layers, the output of the second hidden layer will be a second level encoding of the input data. The final layer of the auto-encoder is a layer

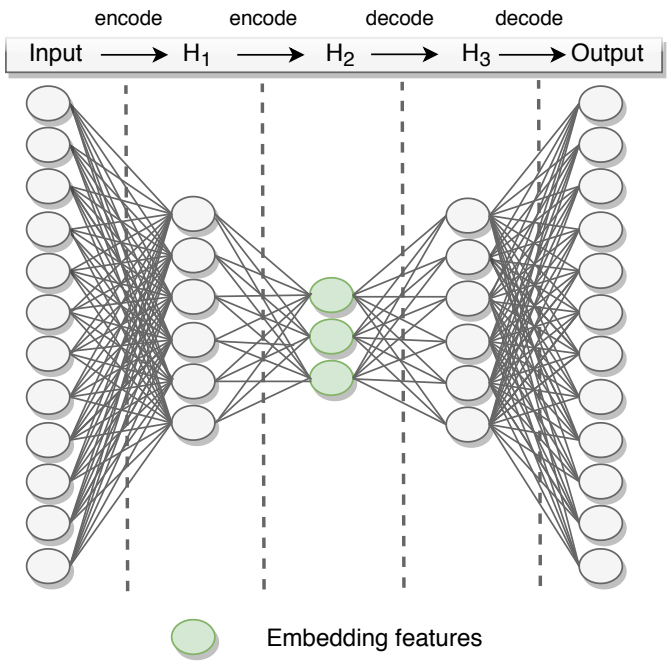

Figure 3: Feature extraction process performed using the encoding function of the trained auto-encoder.

of the same size of input data and provides the reconstruction of the input. When the purpose is classification, usually only the encoding part is used, and the autoencoder is solely used for initialization, as shown in Fig. 3.

In the AE approach described in this subsection, the final layer of the architecture has the same number of neurons as the input layer, such that the decoding stage can be used to reconstruct time series. Instead, the AE-FE approach presented in the following subsection solely uses the encoding stage to extract features that are in turn exploited by classification models.

With one hidden layer, the encoding stage of an auto-encoder takes the input $\mathbf{x} \in \mathbb{R}^d = \mathcal{X}$ and maps it to an hidden representation $\mathbf{z} \in \mathbb{R}^p = \mathcal{F}$, where σ is a sigmoid or a rectified linear unit activation function, W is a weight matrix, and b is a bias vector:

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}). \tag{2}$$

The decoding stage reconstructs $\mathbf{x}$ from $\mathbf{z}$ as:

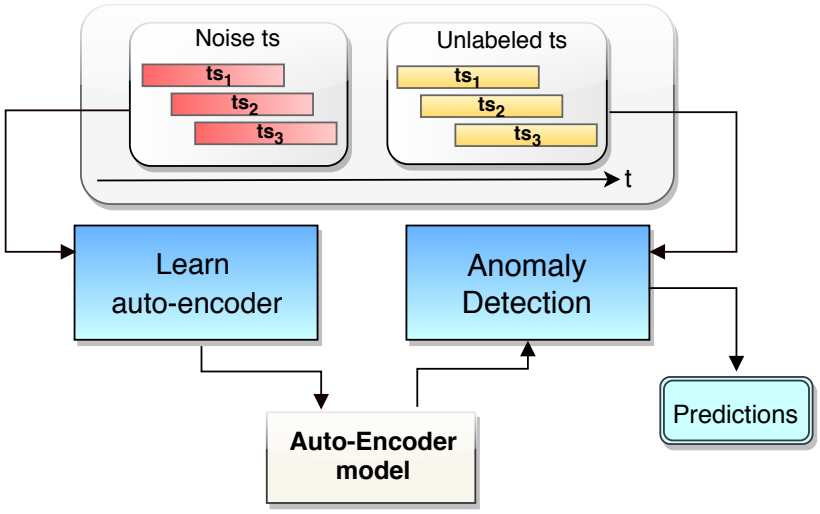

Figure 4: Workflow of the AE proposed method. Negative class time series (red) are exploited to train an auto-encoder model that accurately reconstructs noisy time series. At prediction time, a new time series of unknown class (yellow) is provided as input to the model, and the prediction class is returned, depending on the reconstruction error observed.

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}), \tag{3}$$

such that the following loss is minimized:

$$\mathcal{L}_2(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2. \tag{4}$$

Each time the auto-encoder is trained using training data, a threshold representing the maximum allowed distance is calculated. This threshold allows to define an upper bound to decide whether a data instance can be assumed to belong to the noise data distribution, or not. More precisely, when a data instance presents a reconstruction error that exceeds the defined threshold, it is classified as non-noise (real GW time series). Otherwise, it will be classified as noise. The flow of this method of classification is shown in Fig. 4.

It is important to stress that a valid decision on the threshold must take into account the data distribution of reconstruction errors. In order to take

into account possible changes of such distribution over time (i.e, concept drift), we recalculate the threshold each time the auto-encoder is trained as: $[\overline{e} + f \cdot \sigma_e]$, following a one-tailed sigma rule as described by Pukelsheim (1994), where f is the factor allowed for the standard deviation, $\overline{e}$ is the average reconstruction error of training instances and σ_e is the standard deviation of such errors.

This mechanism allows us to avoid defining the threshold manually, which could lead to a degradation in performance over time, due to changes in the data distribution. Specifying the threshold in terms of number of standard deviations from the mean, allows us to calculate the actual reconstruction error threshold automatically from training data, based on its error distribution. In general, the threshold selection is an open problem that is tackled in different ways in the literature and has a significant impact on the performance of the anomaly detection task, as shown by An & Cho (2015), Khan & Taati (2017), Bontemps et al. (2016) and Clark et al. (2018).

Although the reconstruction error approach for anomaly detection has been already used in past literature, it was, to the best of our knowledge, never applied in the context of GWs. Moreover, one of the significant drawbacks of this approach is that it requires an accurate estimation of the threshold. Therefore, by estimating it automatically, we solve this issue, also taking into account possible changes in the learned distribution.

3.2. Feature extraction with supervised classification (AE-FE)

In deep neural networks, hidden layers represent latent structures that incorporate newly extracted features that are increasingly abstract. Therefore, the entire set of hidden layers can be seen as a feature hierarchy with an increasing degree of complexity. Gehring et al. (2013), Masci et al. (2011) and Bertsekas (2019) have shown that deep auto-encoders preserve the feature extraction capability of traditional auto-encoders, thus allowing to extract a low-dimensional embedding feature space, as done in works by Hinton & Salakhutdinov (2006) and Japkowicz et al. (2000).

In this approach, we exploit auto-encoders exclusively for feature extraction. For such purpose, we define the hidden layers with a lower number of neurons, and they represent a new feature space with reduced dimensionality for input data, also called bottleneck features.

The feature extraction process extracts features recurring to the encoding function of the previously trained auto-encoder. More specifically, considering an auto-encoder architecture composed by one or two hidden layers, the

encoding process transforms the input data feature space F, with $|F|$ features, in a new feature space H_1 of dimensionality $|H1| \ll |F|$. A second encoding stage allows to obtain a new feature space H_2 of dimensionality $|H2| \ll |H1|$.

Adopting auto-encoders for feature extraction allows us to extract a new embedding feature space that describes data at a higher level of abstraction, and with a reduced dimensionality compared to the original input feature space. The primary goal of this process is to mitigate the collinearity phenomenon between features, as done by Corizzo et al. (2019) and, as a consequence, improve its reliability for learning tasks. More details on the collinearity phenomenon can be found in studies by Mason & Perreault Jr (1991) and Belsley et al. (1980). Once the features have been extracted, they can be exploited by a prediction model to perform classification in the new feature space.

In the proposed method, we adopt different methods as prediction models, taking place to different alternatives of the algorithm. We define a suffix in the method name to identify the selected classifier. For example, when Gradient-Boosted Trees (GBTs) are selected as a prediction model, the resulting method is identified as AE-GBT. GBTs are an ensemble-based method that trains decision trees iteratively, by minimizing a loss function, and following a boosting process, as described by Li et al. (2017). The ensemble improves iteratively by predicting target values at each iteration, and relabeling the dataset according to a specific loss function, in order to facilitate an improvement of the next decision tree, taking into account weak predictions made in the past. In this way, GBTs are able to reduce the loss function after each training iteration. One important benefit provided by GBTs is to be able to model non-linear interactions between independent and dependent variables.

Our motivation to include GBTs in our selection of algorithms stands in their demonstrated high performance in predictive modeling and forecasting tasks, as shown in the studies by Ganjisaffar et al. (2011) and Chen & Guestrin (2016). Zieba et al. (2016) and Ceci et al. (2017) obtained high performance in bankruptcy prediction and energy forecasting applications, respectively. Alternative classification algorithms used in our work include Logistic Regression (AE-LR), ExtraTrees (AE-ERT), Random Forest (AE-RF), XGBoost (AE-XGB) and Support Vector Machines (AE-SVM). In the experimental results, we refer to the AE-FE method using the notation of the specific classification algorithm selected for the classification step (i.e.

AE-LR, AE-ERT, AE-GBT, AE-XGB, AE-SVM)

The distributed GBTs implementation used in this work is that available in the Apache Spark Mllib library[2].

Although the most common approach for classification consists in the adoption of a final classification layer to the auto-encoder model, this choice requires a two-step optimization process: the first for the auto-encoder training, based on the reconstruction loss, and the second for the classification model, based on the classification loss. Since our main focus in this paper is to provide high accuracy and scalability at the same time, we preferred the adoption of traditional machine learning methods for the classification step, after observing that they are capable to achieve a high predictive accuracy while requiring a lower training time compared to the aforementioned approach.

The feature extraction step is represented in Fig. 3, while the AE-FE approach is represented in Fig. 5. From this figure it is possible to understand an essential aspect of our approach: supervised models are trained on features extracted by auto-encoders, even on data representing different classes. In fact, even if the auto-encoder has been trained on time series of the negative class (noise), it extracts features also for time series of the positive class, assuming that some labeled data for the positive class is available. The rationale is that, since GWs signals are immersed in noise, and thus resemble the morphology of the noise, the encoded representation of the extracted features allows to highlight important patterns that result crucial for the classification step. In this way, the supervised model learns to discriminate classes based on the difference in the encoded vector representation extracted by the auto-encoder.

4. Experiments

In this section, we describe the datasets and the experimental results obtained in our study, with the two proposed approaches: time series classification via anomaly detection (AE) and feature extraction with supervised classification (AE-FE).

[2] `https://spark.apache.org/docs/latest/mllib-ensembles.html`

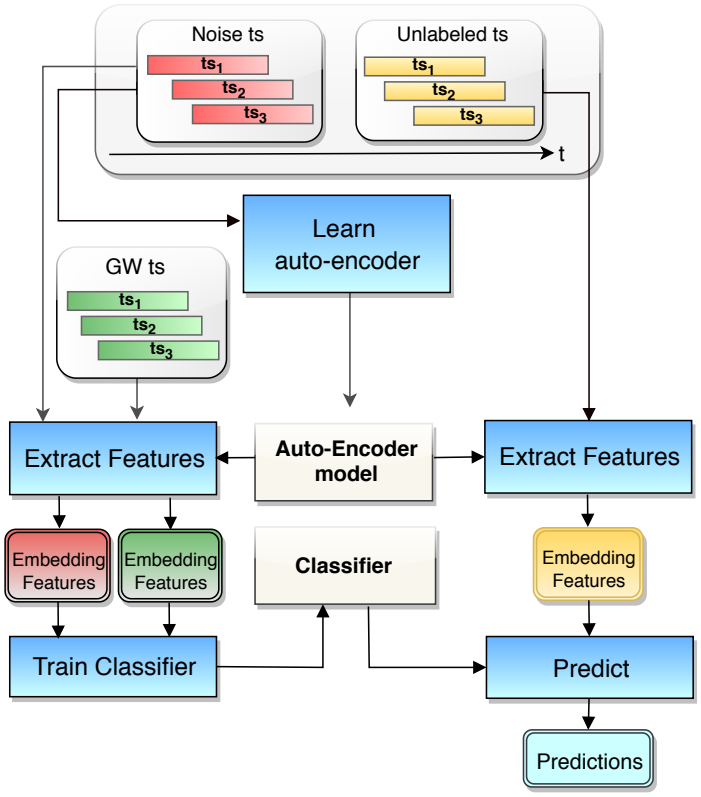

Figure 5: Workflow of the AE-FE proposed method. Negative class time series (red) are exploited to train an auto-encoder model that accurately reconstructs noisy time series. The auto-encoder encodes both negative class (red) and positive class (green) time series and the resulting representation is used to train a classifier. At prediction time, a new time series of unknown class (yellow) is provided to the classification model, which returns its prediction.

4.1. Data description

In this paper, we analyze time series data representing the gravitational strain collected by detectors, that is, the fractional changes in the lengths along the two axes (x-arm and y-arm cavities):

$$h(t) = \frac{\Delta L_y - \Delta L_x}{L}. \tag{5}$$

GWs raw data are also available in the hdf5 data format[3]. This tree-like

[3] `https://www.gw-openscience.org/data/`

format contains the following three files:

- **meta**: various metadata describing the time series observed, the detector, etc;
- **quality**: information about the quality of the data collected;
- **strain**: the actual time series collected by the detector.

In particular, our dataset includes positive class time series related to four published gravitational wave discoveries (GW150914, LVT151012, GW151226, and GW170104) detected from LIGO interferometers, as described by Acernese et al. (2015) and Aasi et al. (2015) , all of which are short-duration BBH (Binary Black Hole) events[4]. Time series of 1 and 3 seconds are collected at a sample rate of 4096 Hz, which leads to time series of 4096 and 12288 elements, respectively. Each event presents four time series, representing the data strain collected at H1 (Hanford) and L1 (Livingstone) interferometers, and their corresponding whitened version, following to pre-processing steps.

Inspired by Shen et al. (2017) and George et al. (2017), we perform data augmentation to enhance the size of the dataset. In particular, time series are shifted in time by an offset in seconds $o \in \{-0.25, -0.125, 0.125, 0.25\}$. This augmentation leads, for each time granularity, to 20 time series for each event, and a total of 80 time series.

In order to learn from noise and generalize over different types of noise, the dataset also includes negative class time series, corresponding to 70 different types of noise, generated (and simulated) by the PyCBC Python library, specialized for GWs data generation[5]. This library has been extensively used in the literature by Nielsen et al. (2019), Nitz et al. (2017), Usman et al. (2016) and Gebhard et al. (2017) to generate reliable GWs data for analysis.

Alternatively, we extract real gravitational noise located in the near proximity (3-15 seconds) of real GWs events directly from the related time series data.

Time series are then normalized using min-max normalization.

In the end, three datasets are created:

[4] `https://losc.ligo.org/s/events/GW150914/LOSC_Event_tutorial_GW150914.html`

[5] `http://pycbc.org/pycbc/latest/html/noise.html`

- **GW1**: It contains positive class and negative class time series as explained above.

- **GW2**: Time series for the negative class correspond to those in GW1, whereas time series for the positive class P are obtained blending whitened signals time series W with the different noisy time series N, at different noise rates ($\alpha = 0.1, 0.25, 0.50$), using an element-wise weighted average approach. Therefore, given a combination of a whitened time series $\mathbf{w} \in W$ and a noisy time series $\mathbf{n} \in N$, the following formula is applied for each element i in order to obtain the resulting time series $\mathbf{p}$:

$$\mathbf{p}[i] = (1 - \alpha) \cdot \mathbf{w}[t] + \alpha \cdot \mathbf{n}[t]. \tag{6}$$

- **GW3**: Time series for the positive class correspond to those in GW1. For the negative class, the time series are obtained by sampling gravitational noise in the near proximity (3-15 seconds) of the events GW170729, GW170809, GW170817, GW170608, GW170814, GW170818, and GW170823. We intentionally included noise located very closely to the events, in order to perform a more realistic evaluation. In fact, the 3-second buffer around an event is meant to make sure that no traces of the GW event are included in the noise data.

- **GW4**: In order to perform scalability experiments on a larger sized dataset, we propose an additional dataset obtained replicating all negative class time series of 1 sec. length (4096 feature values) available from the GW1 dataset multiple times, resulting in up to 50.000 time series. In the replication process, we simulate the presence of new time series belonging to the same data distribution of the original ones.

An overview on the datasets used is described in Table 1 and Table 2. Depending on the learning approach used, time series are used for one-class learning on the negative class and anomaly detection on newly arriving data (AE approach), or for feature extraction and subsequent supervised classification on both classes (AE-FE approach). The three datasets involved in our analysis (GW1, GW2, GW3) are ordered by difficulty, that is, from the simplest to the most challenging. Since GW4 is a larger scaled version of GW1, it is exploited exclusively for scalability experiments (see Section 4.4).

4.2. Experimental setup

For the two approaches, we consider different configurations, in terms of model architectures for the auto-encoder (1 or 2 hidden layers for encoding and 1 or 2 hidden layers for decoding, respectively). In the case of one hidden layer, the number of hidden units is set to 512 or 1024. In the case of two hidden layers, the number of hidden units is set to 512 for the first layer, and 256 for the second layer, or to 1024 for the first layer, and 512 for the second layer.

A 5-fold cross-validation scheme was used in the experiments on the GW1 and GW2 dataset. In the GW3 dataset, the split generation procedure is iterative and it selected at each fold all the representations of one specific signal (i.e., a GW event) as test set, and all the remaining signals as training set. This choice guarantees that all the representations of the same signal are either in the training set or in the test set. Since the signals in the positive class are four, the experimental setup relates to a 4-fold cross-validation scheme. In other words, this is a leave-one-subject-out cross-validation scheme where a subject refers to a GW event signal.

Considering that the time series analyzed are collected at a 4096 Hz sample rate, 512 hidden units correspond to $\frac{1}{8}$ of the 1-second time series length,

Table 1: Overview of the datasets used in the experiments. All variants of GW2 with varying noise rates present the same characteristics.

Dataset	**Class**	**Number of time series**	**Length of time series (1 sec.)**	**Length of time series (3 sec.)**
GW1 / GW2	Signal	80	4096	12288
GW1 / GW2	Noise	70	4096	12288
GW3	Signal	104	4096	12288
GW3	Noise	92	4096	12288
GW4	Noise	50,000	4096	NA

Table 2: Gravitational waves events considered in the GW1, GW2 and GW3 datasets, and their Signal-to-Noise Ratio (SNR) observed at the interferometers.

Event ID	**Signal-to-Noise Ratio (SNR)**
GW150914	24
LVT151012	9.7
GW151226	13
GW170104	13

Table 3: Sensitivity study on the threshold f and its impact on classification performance for the AE approach on all datasets with time series length of 1 second. The model architecture adopted consists of one hidden layer with 512 units. Best results in terms of F-Score are marked in bold.

Dataset	**Threshold f**	**Precision**	**Recall**	**F-Score**
GW1	1.5	0.9938	0.9933	**0.9933**
GW1	3.0	0.8641	0.6600	0.6934
GW1	4.5	0.9558	0.5200	0.6400
GW2 (N=0.1)	1.5	0.9761	0.9733	**0.9734**
GW2 (N=0.1)	3.0	0.9190	0.4400	0.5950
GW2 (N=0.1)	4.5	1	0.4666	0.6363
GW2 (N=0.25)	1.5	0.8902	0.7066	**0.7471**
GW2 (N=0.25)	3.0	0.9190	0.4400	0.5950
GW2 (N=0.25)	4.5	1	0.4666	0.6363
GW2 (N=0.5)	1.5	0.8786	0.4933	0.5993
GW2 (N=0.5)	3.0	0.9190	0.4400	0.5950
GW2 (N=0.5)	4.5	1	0.4666	**0.6363**
GW3	1.5	1.0	0.4285	**0.6000**
GW3	3.0	1.0	0.4285	0.6000
GW3	4.5	1.0	0.4285	0.6000

or $\frac{1}{24}$ of the 3 seconds time series length. Accordingly, 1024 hidden units correspond to $\frac{1}{4}$ of the 1-second time series length, or $\frac{1}{12}$ of the 3 seconds time series length.

The auto-encoder training procedure takes place with the LBFGS optimizer, which follows the minimization of the reconstruction error using training data (non-anomaly instances), stops if the maximum number of iterations is reached (500) or a stopping criterion is reached ($tol < 10e-5$). In particular, the tolerance criterion allows for early stopping if the algorithm is unable to reduce the training error by a specified tolerance at a certain iteration.

In detection mode, we set the factor for the standard deviation to $f = 1.5$. This choice is motivated by a preliminary sensitivity study on the impact of different threshold values on the classification performance (see Table 3). In fact, the best classification results are obtained with $f = 1.5$. There is only one case in which $f = 1.5$ does not reveal the best performance in terms of F-Score (GW2 dataset with noise N=0.5), but even in this circumstance it is still capable to achieve the best performance in terms of Recall.

Regarding the feature extraction approach, the features obtained are exploited by different classifiers. Specifically, we perform experiments with Lo-

gistic Regression (AE-LR), Gradient-Boosted Trees (AE-GBT), ExtraTrees (AE-ERT), Random Forest (AE-RF), XGBoost (AE-XGB) and Support Vector Machines (AE-SVM).

We compared our approach with one-dimensional convolutional neural networks. In the experiments, we refer to this method as Conv1D.

For all methods, we perform hyperparameter selection via grid search (see Lameski et al. (2015)). A summary of the configurations used are summarized in Table 4.

The selected values for the grid search are motivated by studies in the literature that provide specific heuristics. Specifically, Bengio (2012) observed that a valid default value for the batch size is 32, and other powers of 2 are feasible candidates. For learning rate, the author suggests to start from a default value of 0.01 and experiment with a decreasing factor (negative power of 10), considering 10^{-6} as extremely minimum value. Regarding dropout[6], the study by Srivastava et al. (2014) shows that values below 50% should be preferred, in order to avoid the removal of too many neurons that would cause an under-learning phenomenon. The hyperparameters are optimized according to a nested cross-validation scheme.

Moreover, we performed experiments with Deep Filtering (DF), introduced by George & Huerta (2018), a state-of-the-art method for gravitational waves analysis capable of working directly data in the form of time series. This method features a deep architecture based on one-dimensional convolutional neural networks. Specifically, it presents 4 convolution layers with dilations, filter sizes of 64, 128, 256, and 512 respectively, and 2 fully connected layers with sizes 128 and 64. More details on the architecture can be found in the study by George & Huerta (2018).

Our methods are implemented using the Scala programming language, exploiting Apache Spark as a distributed programming framework, whereas all competitors methods are implemented using the Python language exploiting the Keras library (see Chollet et al. (2015)).

Table 5 shows a summary of the best experimental results in terms of F-Score obtained with the optimal configuration for each method and dataset. The nonparametric statistical tests performed with the toolkit proposed by García et al. (2010), are shown in Table 6. Detailed scores of all algorithms

[6]Dropout, as described by Srivastava et al. (2014) is a regularization technique that reduces overfitting in neural networks by dropping out units (both hidden and visible).

Methods	Parameter	Min	Max	Step	Set of values
Conv1D, **Conv1D (2)**	Conv Layers	/	/	/	{2}
	Activation Function (Conv)	/	/	/	{ReLU}
	Filters	20	40	20	/
	Filter length	3	6	3	/
	Max Pooling Layers Size	/	/	/	{2}
	Dense Layers	/	/	/	{(512,256) , (1024,512)}
	Activation Function (Output)	/	/	/	{Softmax}
Conv1D, **Conv1D (2)**, **Deep Filtering**	Learning rate	10^{-4}	10^{-1}	$\{10^1\}$	/
	Dropout	0.1	0.3	0.2	/
	Batch Size	8	32	8	/
	Dense Units	512	1024	512	/
AE	Threshold f	1.5	4.5	1.5	/
AE, **AE-FE**	Hidden Units	512	1024	512	/
	Hidden Layers	1	2	1	/
	Max Epochs	/	/	500	/
	tol	/	/	/	$\{10e-5\}$
AE-LR	*maxIter*	/	/	/	{100}
AE-ERT, **AE-RF**	*numEstimators*	/	/	/	$\{10, 100, 1000\}$
AE-GBT	maxIter	/	/	/	{10}
	maxDepth	/	/	/	{5}
	maxBins	/	/	/	{32}
	minInstancesPerNode	/	/	/	{1}
	lossType	/	/	/	{logistic}
AE-XGB	*numEstimators*	/	/	/	{1000}
AE-SVM	Kernel Type	/	/	/	{RBF}
	C	1	100	10^1	/
	Gamma	0.0001	0.1	10^1	/

Table 4: Summary of hyperparameter configurations for all methods.

on each dataset and different configurations are presented in Appendix in Table 7, Table 8, Table 9, Table 10, and Table 11.

4.3. Accuracy results

Table 7 reports results in terms of Precision, Recall and F-score for the two proposed approaches (AE and AE-FE), for two implementations of one-dimensional convolutional neural networks (Conv1D), and for Deep Filtering (DF) with the GW1 dataset.

Common perception is that, in astrophysical data analysis, a higher Recall should be preferred compared to a higher Precision. Under this assumption, a better coverage in the detection of real GW would be obtained at the cost of a higher number of False Positives, that should be subsequently discarded manually. However, a perfect Recall performance could be obtained by a model that systematically predicts the positive class. Moreover, all collected gravitational data is being stored permanently, so retrospective processing of historical data, and potentially detecting missed events is still possible. Therefore, having a non-perfect Recall is acceptable. On the other hand, the results of detection of GW with the proposed methods could be used by astronomers as feedback on what to focus on. Consequently, they could decide to modify the positioning of a telescope system in order to observe a potential phenomenon (e.g., a source of GWs). This could be a costly operation, therefore the cost of false positives could be devastating, hence the importance of the Precision. Therefore, a tradeoff between Precision and Recall is necessary, also in order to avoid a too expensive manual intervention in filtering False Positives. This motivates the adoption of the F-Score as a reference metric to compare all approaches, since it provides a balance between Precision and Recall.

The results show that both the proposed approaches can obtain very high accuracy, classifying noise and real GWs time series effectively. Concerning the AE approach, once the auto-encoder has been trained with time series representing different types of noise, the detection approach based on reconstruction error with automatic threshold estimation can discriminate effectively unseen time series between noise and GWs. Moreover, there is no significant difference between the different auto-encoder architectures in terms of the predictive accuracy obtained. The same consideration holds when comparing AE, AE-FE, and Conv1D, which obtain similar results. Overall, the best results in terms of F-Score are obtained by AE-FE. However, it is important to stress that the AE approach assumes no prior knowledge about the

data distribution of the positive class, representing GWs phenomena, and it provides classification solely based on the reconstruction error of unseen time series. This feature is particularly useful considering that, currently, very few time series representing GWs phenomena have been observed and validated, and a supervised approach considering the two classes might be subject to degradation. In this case, the detection of unseen phenomena might result in difficulty, whereas it could still be performed accurately relying on the reconstruction error approach.

Dataset	Optimal Configuration	Conv1D	Conv1D (2)	Deep Filtering	AE	AE-LR	AE-GBT	AE-ERT	AE-RF	AE-XGB	AE-SVM
GW1	**Time series length (s)**	1	1	3	1	3	3	3	3	3	3
	Hidden layers	1	2	2	1	1	1	1	1	1	1
	Neurons	512	512-256	128-64	1024	512	512	512	512	512	512
	F-Score	0.9933	0.9932	0.9932	0.9934	0.9933	**1**	**1**	**1**	**1**	**1**
GW2	**Time series length (s)**	1	1	3	1 or 3	3	1	3	1 or 3	1	3
(N=10%)	**Hidden layers**	1	2	2	1	1	1	1	1	1	1
	Neurons	512	512-256	128-64	512	512	512	512	512	512	512
	F-Score	0.9797	0.9796	0.7205	0.9735	**0.9933**	**0.9933**	**0.9933**	**0.9933**	**0.9933**	**0.9933**
GW2	**Time series length (s)**	1	1	3	1	1	1	3	3	3	3
(N=25%)	**Hidden layers**	1	2	2	2	1	1	1	1	1	1
	Neurons	512	1024-512	128-64	512-256	512	1024	1024	512	1024	512
	F-Score	0.939	0.9796	0.85	0.7469	0.973	**0.9933**	0.9798	0.9821	0.9709	0.9888
GW2	**Time series length (s)**	1	1 or 3	3	3	3	3	3	3	3	3
(N=50%)	**Hidden layers**	1	2	2	1	1	2	1	1	1	1
	Neurons	1024	512-256	128-64	512	1024	512-256	1024	512	1024	512
	F-Score	0.847	0.9659	0.6782	0.6021	0.946	**0.9866**	0.9595	0.9597	0.9463	0.9798
GW3	**Time series length**	1 sec	3	3	1	3	3	3	3	3	3
	Hidden layers	1	2	2	1	1	2	1	1	1	1
	Neurons	1024	512-256	128-64	512	1024	512-256	1024	1024	512	1024
	F-Score	0.5889	0.5522	0.4155	0.6388	0.7797	0.6559	0.7947	0.7877	**0.8194**	0.7934

Table 5: Summary of experimental results in terms of F-Score obtained with the optimal configuration for each method and dataset. The best F-Score obtained for each dataset is marked in bold. In cases where more than one configuration obtained the same optimal performance, we report the simplest configuration in terms of model architecture (least number of hidden layers and neurons). The number of hidden layers and neurons for Conv1D, Conv1D (2) and Deep Filtering (DF) refer to their final dense layers.

Table 6: Average Rankings of the algorithms (Friedman). The lowest ranking is marked in bold.

Algorithm	Ranking
Conv1D	8.643
Conv1D (2)	6.725
Deep Filtering	9.062
AE	7.474
AE-LR	5.200
AE-GBT	3.843
AE-ERT	3.075
AE-RF	3.150
AE-XGB	5.312
AE-SVM	**2.512**

Table 8, Table 9, and Table 10 report the results obtained with the GW2 dataset, with noise levels of 10%, 25% and 50%, respectively. The results show that AE and AE-FE are still comparable when the noise rate is 10%, whereas the performances of AE and Conv1D start degrading. The difference in terms of F-Score between AE-FE and the competitor methods appears more consistent in this dataset, compared to the GW1 dataset. These results further confirm the fact that AE-FE is the best approach.

The degradation of AE is particularly visible in terms of Recall, while Precision is still relatively high. This behavior means that the AE model still returns the correct class when it returns a prediction for the positive class, but it misses the recognition of positive class time series (i.e., the noise is impacting the model in terms of an increase in the number of false negatives).

This result was expected since, in the GW2 dataset, positive class time series are obtained by contaminating whitened signals with the same noise distribution learned by the auto-encoder (AE approach). In particular, the strong perturbation leads the distributions of real GWs signals and noise to get closer. This perturbation also leads to a significant reduction of the differences in terms of reconstruction error among time series of different classes. Therefore, the reconstruction error threshold learned for the AE approach results no longer accurate when used to distinguish between the two classes. On the other hand, the AE-FE approach maintains optimal performance, also when noise increases: the model is capable of learning accurately the mapping between variations in features values extracted via

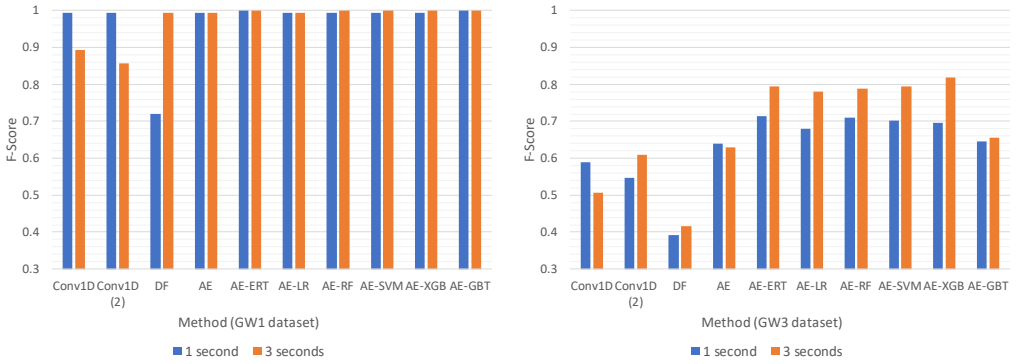

Figure 6: Summary of experimental results with the GW1 (a) and GW3 (b) datasets for all considered methods with time series of length 1 second and 3 seconds. The bar charts show the best F-Score obtained by each method selecting its best performing configuration.

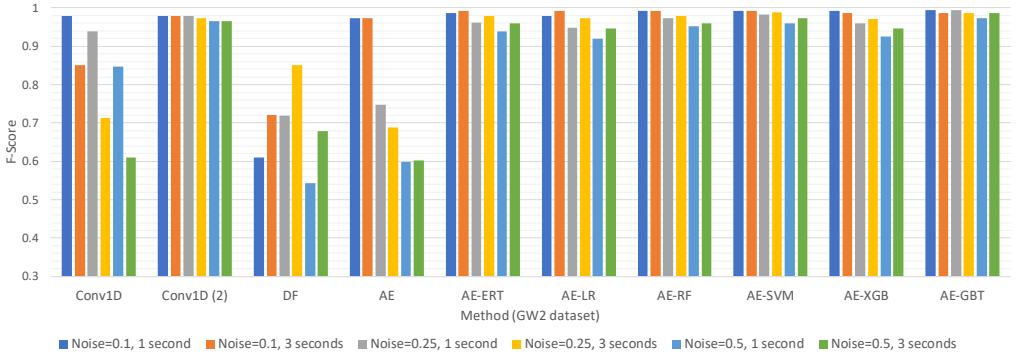

Figure 7: Summary of experimental results with the GW2 dataset for all considered methods with time series of length 1 second and 3 seconds, and noise levels 0.1, 0.25 and 0.5. The bar chart shows the best F-Score obtained by each method selecting its best performing configuration.

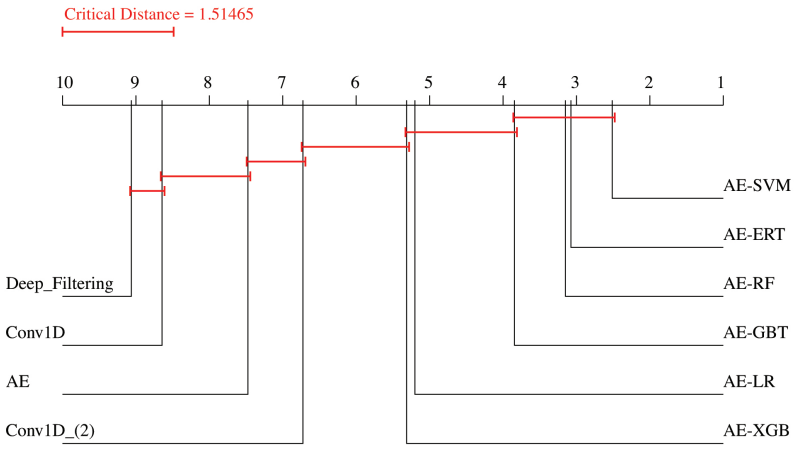

Figure 8: Nemenyi post-hoc critical distance diagram. If the distance between methods is less than the critical distance (at p-value = 0.05), there is no statistically significant difference between them.

the auto-encoder and the corresponding class, even when they are small, thanks to the positively labeled time series used during the training phase of gradient-boosted trees.

Another consideration regarding the AE approach is that its performance appears in some cases worse when it processes time series of 3 seconds. One explanation of this result is that, especially in presence of a high noise rate, the contribution of the noise in terms of high reconstruction error is consistent, whereas the contribution of the signal appears negligible, since it represents a fraction of the overall duration of the time series. This aspect implies that the classification task is more challenging with time series of 3 seconds. On the other hand, the convolutional filters of the Deep Filtering method perform better with longer time series, although the overall classification result is still much worse than our proposed approaches.

It is also possible to observe that the AE approach appears in some cases worse when the network size increases in terms of neurons or layers. This result implies that, for the particular model and optimization algorithm we adopted, larger-sized networks have a better likelihood to propagate data noise at further layers, resulting in a less accurate classification performance, whereas smaller-sized networks appear more capable to reduce noise propagation and extract a more compact and reliable data representation that results in more accurate classifications.

Table 11 reports the results obtained with the GW3 dataset. It is possible to observe a further degradation in the performance of all methods, deriving from the increased complexity of the task. However, AE-FE still appears to be the best approach, with satisfactory classification performance in terms of F-Score.

A summary of experimental results obtained for all methods considered and with all datasets is shown in Figure 6 and 7. The bar charts show the best F-Score obtained by each method selecting its best performing configuration. It is noteworthy that, even if the competitor methods analyzed in our study utilize directly time series of the two classes, and are free to learn characteristics of both noise and GWs, they exhibit lower classification performance.

In order to statistically validate the results obtained, in Table 12 we report the p-values of the Wilcoxon Signed Rank tests. The results show that, overall, both the proposed AE and AE-FE significantly outperform the competitor methods, and that AE-FE significantly outperforms AE.

Although AE-FE results appear favorable, it is noteworthy that the AE-FE approach is based on the assumption that the distribution of positive class phenomena is known since, differently from the AE approach, it requires data of both classes during the training phase. This assumption holds also for the Conv1D and the Deep Filtering methods.

However, this assumption might be too strong in the context of GWs analysis, since it is not always possible to know the morphology of the expected phenomena, given the current limited knowledge based on a small number of verified phenomena. In this perspective, the AE approach could be a valid alternative to AE-FE. In fact, the results obtained, which are generally in favor of the AE-FE approach, might represent a scenario that may be more or less realistic, depending on the context.

Additional nonparametric statistical tests performed with the toolkit proposed by García et al. (2010), are shown in Table 6. In order to statistically confirm the results obtained, we used the Friedman test with the Nemenyi post-hoc test at $\alpha = 0.05$. In Figure 8, we depict the result of the test, which shows that the methods exploiting our proposed feature extraction approach significantly outperform other methods.

4.4. Scalability results

In order to assess the execution performance of our distributed implementation, we performed scalability experiments on CPUs.

Considering that the analytic task we address is computationally and memory intensive, depending on the amount of data required for training, the purpose is to evaluate if our method is capable of scaling with increasing data size, and with an increasing number of CPUs. Moreover, we are interested in measuring the advantage in terms of running time for a distributed execution, with respect to a local execution of the method (i.e., running on a single machine).

The focus of the experiment is on the AE approach, which trains auto-encoders and constitutes the core of our contribution in terms of code implementation. In fact, AE-FE is a combination of the AE approach with classification models that are available as stable implementations in different machine learning libraries and frameworks.

As explained in Section 4, our experiments are performed on the GW4 dataset, obtained replicating all negative class time series of 1 sec. length (4096 feature values) available from the GW1 dataset multiple times, resulting in up to 50.000 time series. In the replication process, we simulate the presence of new time series belonging to the same data distribution of the original ones. For this purpose, for each time series selected randomly from the original dataset, the new time series is obtained by introducing, at most, 1% perturbation of each element of the time series. This choice was necessary in order to avoid full replication of time series, which could result in an unrealistically fast convergence of the loss function.

In the experiments, we adopted two Spark cluster configurations on Microsoft Azure HDInsight cloud infrastructure[7].

For the local setting, we adopt a configuration with 2 D12 head nodes, and one worker node of the same instance type (D12).

A D12 instance is equipped with four cores and 28 GB of RAM, whereas a D13 instance has eight cores and 56 GB of RAM, and a D14 instance has 16 cores and 112 GB RAM.

The first cluster configuration consists of 2 D12 head nodes and 4 D13 worker nodes. This cluster presents a scale factor of 8 compared to the local counterpart.

The second cluster configuration consists of 2 D12 head nodes and 8 D14 worker nodes. This cluster presents a scale factor of 16 compared to the local

[7] `https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/`

counterpart and a scale factor of 2 compared to the first cluster configuration.

For all cluster sizes and dataset sizes, we evaluated different configurations for the Spark job, including the number of executors (4, 8, 16), the number of executor cores (4, 8 and 16) and the executor memory (4 GB, 8 GB, 16 GB, and 32 GB). We argue that in a real-life deployment scenario the data volume to be collected and analyzed per time unit is known upfront, considering that the sample rate of the detectors is predefined. Therefore, configuring cluster parameters to the expected data volume is a possible choice.

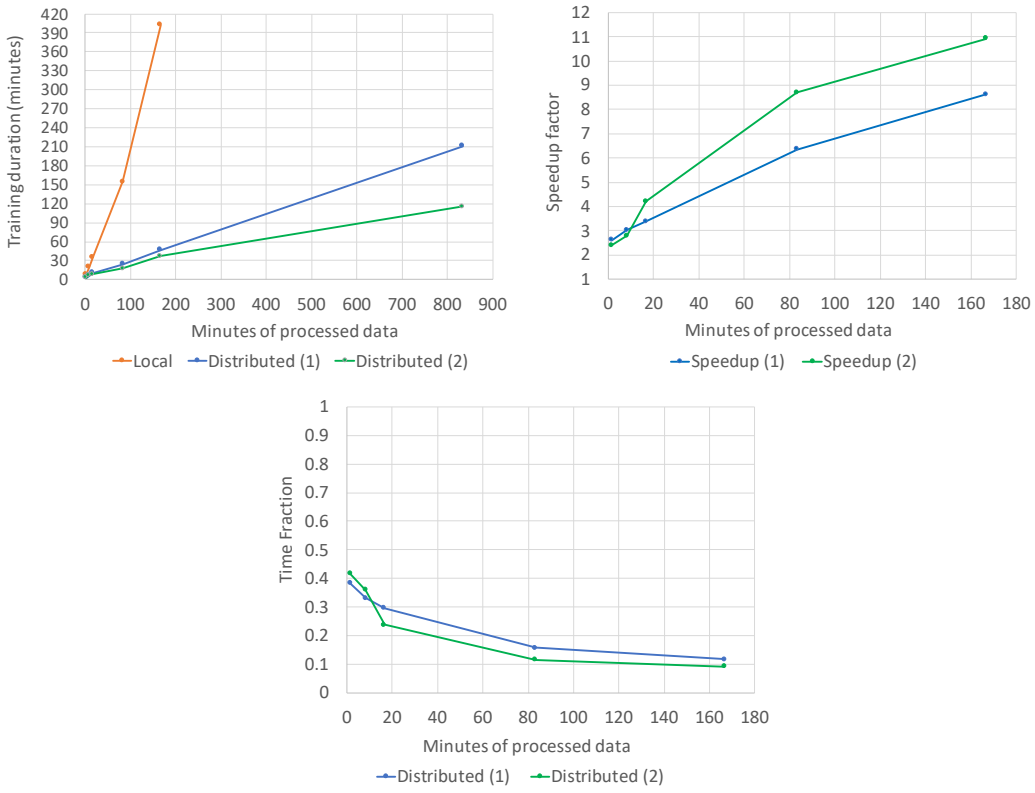

Figure 9: Scalability results in terms of execution time (left), speedup (right), and time fraction between distributed and local executions (bottom) for the AE approach, obtained with two cluster configurations. The two curves (1) and (2) refer to the different Spark cluster configurations on Azure HDInsight.

Fig. 9 shows the results in terms of execution time (left) and the speedup factor (right) observed with the best parameter values on the two clusters. The speedup graph does not contain the last point, related to 50.000 time

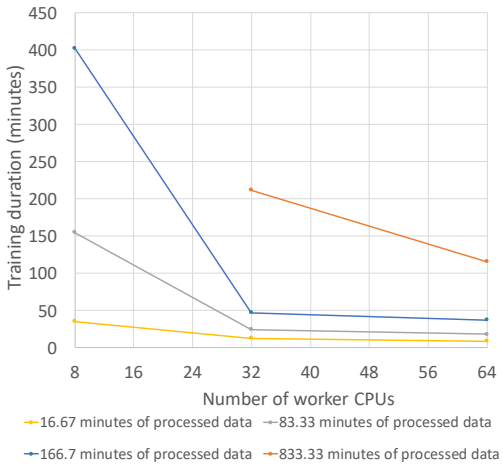

Figure 10: Scalability results in terms of execution time with a varying number of CPUs: 8 - Local, 32 - Distributed (1), 64 - Distributed (2). The missing point of the last series in the graph (833.33 minutes of processed data) denotes that the Local configuration with 8 worker CPUs could not complete the training process successfully due to an excessive memory and CPU overhead.

series, since the local execution could not be completed successfully due to an excessive memory and CPU overhead. Fig. 10 presents a different perspective in terms of execution time with a varying number of worker CPUs and different amounts of processed data.

Overall, it is possible to observe that the proposed implementation is capable of scaling linearly as data increases. Moreover, the speedup factor obtained with the distributed execution is consistently high, and it still increases with the addition of worker cores, considering the improvement obtained by cluster configuration. This performance confirms that our approach benefits from a cluster environment and scales well with large datasets.

4.5. Availability

The distributed implementation of the proposed approaches and the datasets are available to replicate the experiments at the following link:

`http://www.di.uniba.it/~corizzo/gw`.

5. Conclusion

In this paper, we proposed two approaches to perform unsupervised and supervised GWs detection directly from strain data collected in the form of time series, that require no manual effort in pre-processing and noise removal steps.

The results have shown that both approaches AE and AE-FE outperform one-dimensional convolutional neural networks and the Deep Filtering method. Moreover, AE is best suited when no prior knowledge is available about the data distribution of GWs phenomena. This assumption is quite realistic, given the current scarce availability of labeled time series representing verified phenomena. In such case, the models, trained using different types of noise, are sensitive to deviations from such data distributions, and therefore they are capable of detecting potentially interesting signals hidden in noisy time series, that may correspond to real phenomena.

On the other hand, when knowledge about real GWs phenomena representing the positive class is available, the AE-FE approach, that involves classification models learned using features extracted from an auto-encoder, outperforms the AE approach. Specifically, this method obtains the best detection performance when a phenomenon, already observed in the past, appears again in a newly observed time series. Additional experiments in the presence of varying noise rates introduced in time series of whitened signals highlighted that the AE approach starts to degrade significantly when noise reaches 25% of the signal (or more), while the AE-FE approach is more robust to noise.

However, the AE-FE approach assumes the availability of positive time series in order to train a supervised classifier on data for two classes. Realistically, we assume that its near perfect predictive performance obtained in our experiments, is also subject to degradation over time, in the case in which the morphology of GWs changes significantly in newly observed time series never seen before.

The proposed approaches implemented exploiting the Apache Spark framework primitives, also exhibit good scalability performances when operating with large-scale data and, therefore, are particularly suited for analytic tasks involving GWs time series, that are produced at a high volume.

For future work, we aim to extend our study addressing the glitch classification task with time series data. Moreover, we aim to investigate alternative neural network architectures, specifically optimized for the detection task.

Acknowledgment

This article/publication is based upon work from COST Action CA17137 "A network for Gravitational Waves, Geophysics and Machine Learning", supported by COST (European Cooperation in Science and Technology).

We also acknowledge the support of the Ministry of Education, Universities and Research (MIUR) through the project ComESto - Community Energy Storage: Gestione Aggregata di Sistemi dAccumulo dellEnergia in Power Cloud (Grant No. ARS01_01259) and the PON Ricerca e Innovazione 2014–2020 project "CLOSE – Close to the Earth" (ARS01_001413), funded by the Italian Ministry for Universities and Research (MIUR).

The author affiliated with the Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University acknowledges its support for the work on this project.

References

Aasi, J., Abbott, B., Abbott, R., Abbott, T., Abernathy, M., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R. et al. (2015). Advanced ligo. *Classical and quantum gravity*, *32*, 074001.

Abbott, B., Abbott, R., Abbott, T., Abraham, S., Acernese, F., Ackley, K., Adams, C., Adhikari, R., Adya, V., Affeldt, C. et al. (2018). Gwtc-1: A gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs. *arXiv preprint arXiv:1811.12907*, .

Abbott, B., Jawahar, S., Lockerbie, N., & Tokmakov, K. (2016a). Ligo scientific collaboration and virgo collaboration (2016) gw150914: first results from the search for binary black hole coalescence with advanced ligo. physical review d, 93 (12). issn 1550-2368. *PHYSICAL REVIEW D Phys Rev D*, *93*, 122003.

Abbott, B. P., Abbott, R., Abbott, T., Abernathy, M., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R. et al. (2016b). Binary black hole mergers in the first advanced ligo observing run. *Physical Review X*, *6*, 041015.

Abbott, B. P., Abbott, R., Abbott, T., Abernathy, M., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R. et al. (2016c).

Gw151226: Observation of gravitational waves from a 22-solar-mass binary black hole coalescence. *Physical review letters*, *116*, 241103.

Abbott, B. P., Abbott, R., Abbott, T., Abernathy, M., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R. et al. (2016d). Observation of gravitational waves from a binary black hole merger. *Physical review letters*, *116*, 061102.

Acernese, F., Adams, T., Agathos, M., Agatsuma, K., Allocca, A., Astone, P., Ballardin, G., Barone, F., Barsuglia, M., Basti, A. et al. (2015). The advanced virgo detector. In *Journal of Physics: Conference Series* (p. 012014). IOP Publishing volume 610-1.

Allen, B., & Romano, J. D. (1999). Detecting a stochastic background of gravitational radiation: Signal processing strategies and sensitivities. *Physical Review D*, *59*, 102001.

An, J., & Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, *2*, 1–18.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *31*, 606–660.

Bahaadini, S., Noroozi, V., Rohani, N., Coughlin, S., Zevin, M., Smith, J., Kalogera, V., & Katsaggelos, A. (2018a). Machine learning for gravity spy: Glitch classification and dataset. *Information Sciences*, *444*, 172–186.

Bahaadini, S., Rohani, N., Coughlin, S., Zevin, M., Kalogera, V., & Katsaggelos, A. K. (2017). Deep multi-view models for glitch classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2931–2935). IEEE.

Bahaadini, S., Rohani, N., Katsaggelos, A. K., Noroozi, V., Coughlin, S., & Zevin, M. (2018b). Direct: Deep discriminative embedding for clustering of ligo data. In *2018 25th IEEE International Conference on Image Processing (ICIP)* (pp. 748–752). IEEE.

Belsley, D. A., Kuh, E., & Welsch, R. E. (1980). Regression diagnostics: Identifying influential data and sources of collinearity.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade* (pp. 437–478). Springer.

Bertsekas, D. P. (2019). Feature-based aggregation and deep reinforcement learning: a survey and some new implementations. *IEEE/CAA Journal of Automatica Sinica*, *6*, 1–31.

Biwer, C., Capano, C. D., De, S., Cabero, M., Brown, D. A., Nitz, A. H., & Raymond, V. (2019). Pycbc inference: A python-based parameter estimation toolkit for compact binary coalescence signals. *Publications of the Astronomical Society of the Pacific*, *131*, 024503.

Bontemps, L., McDermott, J., Le-Khac, N.-A. et al. (2016). Collective anomaly detection based on long short-term memory recurrent neural networks. In *International Conference on Future Data and Security Engineering* (pp. 141–152). Springer.

Ceci, M., Corizzo, R., Fumarola, F., Malerba, D., & Rashkovska, A. (2017). Predictive modeling of PV energy production: How to set up the learning task for a better prediction? *IEEE Trans. Industrial Informatics*, *13*, 956–966. doi:10.1109/TII.2016.2604758.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*, 15.

Chen, L., Hu, X., Tian, W., Wang, H., Cao, D., & Wang, F.-Y. (2019). Parallel planning: a new motion planning framework for autonomous driving. *IEEE/CAA Journal of Automatica Sinica*, *6*, 236–246.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794). ACM.

Chollet, F. et al. (2015). Keras. https://keras.io.

Clark, J. I. W., Liu, Z., & Japkowicz, N. (2018). Adaptive threshold for outlier detection on data streams. *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, (pp. 41–49).

Corizzo, R., Ceci, M., & Japkowicz, N. (2019). Anomaly detection and repair for accurate predictions in geo-distributed big data. *Big Data Research*, .

Cuoco, E., Calamai, G., Fabbroni, L., Losurdo, G., Mazzoni, M., Stanga, R., & Vetrano, F. (2001a). On-line power spectra identification and whitening for the noise in interferometric gravitational wave detectors. *Classical and Quantum Gravity*, *18*, 1727.

Cuoco, E., Losurdo, G., Calamai, G., Fabbroni, L., Mazzoni, M., Stanga, R., Guidi, G., & Vetrano, F. (2001b). Noise parametric identification and whitening for ligo 40-m interferometer data. *Physical Review D*, *64*, 122002.

Gabbard, H., Williams, M., Hayes, F., & Messenger, C. (2018a). Matching matched filtering with deep networks for gravitational-wave astronomy. *Physical review letters*, *120*, 141103.

Gabbard, H., Williams, M., Hayes, F., & Messenger, C. (2018b). Matching matched filtering with deep networks for gravitational-wave astronomy. *Physical review letters*, *120*, 141103.

Ganjisaffar, Y., Caruana, R., & Lopes, C. V. (2011). Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* (pp. 85–94). ACM.

García, S., Fernández, A., Luengo, J., & Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, *180*, 2044–2064.

Gebhard, T., Kilbertus, N., Parascandolo, G., Harry, I., & Schölkopf, B. (2017). Convwave: Searching for gravitational waves with fully convolutional neural nets. In *Workshop on Deep Learning for Physical Sciences (DLPS) at the 31st Conference on Neural Information Processing Systems (NIPS)* (pp. 1–6).

Gehring, J., Miao, Y., Metze, F., & Waibel, A. (2013). Extracting deep bottleneck features using stacked auto-encoders. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 3377–3381). IEEE.

George, D., & Huerta, E. (2018). Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced ligo data. *Physics Letters B*, *778*, 64–70.

George, D., Shen, H., & Huerta, E. (2017). Deep transfer learning: A new deep learning glitch classification method for advanced ligo. *arXiv preprint arXiv:1706.07446*, .

Goldstein, M., & Uchida, S. (2016). A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, *11*, 1–31. doi:10.1371/journal.pone.0152173.

Hinton, G., & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*, 504–507.

Japkowicz, N. (2001). Supervised versus unsupervised binary-learning by feedforward neural networks. *Machine Learning*, *42*, 97–122. doi:10.1023/A:1007660820062.

Japkowicz, N., Hanson, S. J., & Gluck, M. A. (2000). Nonlinear autoassociation is not equivalent to PCA. *Neural Computation*, *12*, 531–545. doi:10.1162/089976600300015691.

Khan, S. S., & Taati, B. (2017). Detecting unseen falls from wearable devices using channel-wise ensemble of autoencoders. *Expert Systems with Applications*, *87*, 280–290.

Lameski, P., Zdravevski, E., Mingov, R., & Kulakov, A. (2015). Svm parameter tuning with grid search and its impact on reduction of model overfitting. In Y. Yao, Q. Hu, H. Yu, & J. W. Grzymala-Busse (Eds.), *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing* (pp. 464–474). Cham: Springer International Publishing.

Li, L., Lin, Y., Zheng, N., & Wang, F.-Y. (2017). Parallel learning: a perspective and a framework. *IEEE/CAA Journal of Automatica Sinica*, *4*, 389–395.

Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. *Artificial Neural Networks and Machine Learning–ICANN 2011*, (pp. 52–59).

Mason, C. H., & Perreault Jr, W. D. (1991). Collinearity, power, and interpretation of multiple regression analysis. *Journal of marketing research*, (pp. 268–280).

Mukherjee, S., Obaid, R., & Matkarimov, B. (2010). Classification of glitch waveforms in gravitational wave detector characterization. In *Journal of Physics: Conference Series* (p. 012006). IOP Publishing volume 243-1.

Najafabadi, M., Villanustre, F., Khoshgoftaar, T., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. *Journal of Big Data*, *2*, 1.

Nielsen, A. B., Nitz, A. H., Capano, C. D., & Brown, D. A. (2019). Investigating the noise residuals around the gravitational wave event gw150914. *Journal of Cosmology and Astroparticle Physics*, *2019*, 019.

Nitz, A. H., Dent, T., Dal Canton, T., Fairhurst, S., & Brown, D. A. (2017). Detecting binary compact-object mergers with gravitational waves: Understanding and improving the sensitivity of the pycbc search. *The Astrophysical Journal*, *849*, 118.

Powell, J., Torres-Forné, A., Lynch, R., Trifirò, D., Cuoco, E., Cavaglià, M., Heng, I. S., & Font, J. A. (2017). Classification methods for noise transients in advanced gravitational-wave detectors ii: performance tests on advanced ligo data. *Classical and Quantum Gravity*, *34*, 034002.

Principi, E., Rossetti, D., Squartini, S., & Piazza, F. (2019). Unsupervised electric motor fault detection by using deep autoencoders. *IEEE/CAA Journal of Automatica Sinica*, *6*, 441–451.

Pukelsheim, F. (1994). The three sigma rule. *The American Statistician*, *48*, 88–91.

Razzano, M., & Cuoco, E. (2018). Image-based deep learning for classification of noise transients in gravitational wave detectors. *Classical and Quantum Gravity*, *35*, 095016.

Shen, H., George, D., Huerta, E., & Zhao, Z. (2017). Denoising gravitational waves using deep learning with recurrent denoising autoencoders. *arXiv preprint arXiv:1711.09919*, .

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*, 1929–1958.

Usman, S. A., Nitz, A. H., Harry, I. W., Biwer, C. M., Brown, D. A., Cabero, M., Capano, C. D., Dal Canton, T., Dent, T., Fairhurst, S. et al. (2016). The pycbc search for gravitational waves from compact binary coalescence. *Classical and Quantum Gravity*, *33*, 215004.

Xing, Z., Pei, J., & Keogh, E. (2010). A brief survey on sequence classification. *SIGKDD Explor. Newsl.*, *12*, 40–48. doi:10.1145/1882471.1882478.

Y. Bengio, Y. et al. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, *2*, 1–127.

Zevin, M., Coughlin, S., Bahaadini, S., Besler, E., Rohani, N., Allen, S., Cabero, M., Crowston, K., Katsaggelos, A. K., Larson, S. L. et al. (2017). Gravity spy: integrating advanced ligo detector characterization, machine learning, and citizen science. *Classical and quantum gravity*, *34*, 064003.

Zhou, C., & Paffenroth, R. (2017). Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 665–674). ACM.

Zieba, M., Tomczak, S. K., & Tomczak, J. M. (2016). Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction. *Expert Systems with Applications*, *58*, 93–101.

Appendix: Additional experimental results

Table 7: Experimental results with the GW1 dataset for the proposed methods (AE, AE-FE), considering different model architectures, noise rates and time series lengths. Best results in terms of F-Score are marked in bold.

	Time series length					
Method	1 sec			3 secs		
Conv1D optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.9941	0.9928	0.9933	0.5474	0.6928	0.5960
1024 dense units	0.9941	0.9928	0.9933	0.5541	0.6928	0.6019
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.9941	0.9928	0.9932	0.8474	0.8928	0.8628
1024 dense units	0.9941	0.9928	0.9932	0.2533	0.500	0.3360
Conv1D (2) optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.9941	0.9928	0.9932	0.5541	0.6928	0.6019
1024 - 512 dense units	0.9941	0.9928	0.9932	0.5533	0.7000	0.6027
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.9941	0.9928	0.9932	0.8407	0.8928	0.8569
1024 - 512 dense units	0.9941	0.9928	0.9932	0.6888	0.7857	0.7196
Deep Filtering optimized	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
	0.6888	0.7857	0.7196	0.9941	0.9928	0.9932
Proposed method (AE)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9938	0.9933	0.9933	0.9824	0.9800	0.9801
2 hidden layers - 512 units	0.9737	0.9400	0.9444	0.9823	0.9800	0.9801
1 hidden layer - 1024 units	0.9938	0.9933	0.9934	0.9938	0.9933	0.9933
2 hidden layers - 1024 units	0.9938	0.9933	0.9933	0.9938	0.9933	0.9933
Proposed method (AE-LR)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9889	0.9857	0.9864	0.9941	0.9929	0.9933
2 hidden layers - 512 units	0.9941	0.9929	0.9933	0.9941	0.9929	0.9933
1 hidden layer - 1024 units	0.9889	0.9857	0.9864	0.9941	0.9929	0.9933
2 hidden layers - 1024 units	0.9941	0.9929	0.9933	0.9941	0.9929	0.9933
Proposed method (AE-GBT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	1	1	**1**	1	1	**1**
2 hidden layers - 512 units	0.9938	0.9933	0.9933	1	1	**1**
1 hidden layer - 1024 units	0.9938	0.9933	0.9933	0.9937	0.9933	0.9933
2 hidden layers - 1024 units	0.9938	0.9933	0.9933	1	1	**1**
Proposed method (AE-ERT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	1	1	**1**	1	1	**1**
2 hidden layers - 512 units	0.9941	0.9929	0.9933	1	1	**1**
1 hidden layer - 1024 units	0.9941	0.9929	0.9933	0.9941	0.9929	0.9933
2 hidden layers - 1024 units	0.9941	0.9929	0.9933	1	1	**1**
Proposed method (AE-RF)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9941	0.9929	0.9933	1	1	**1**
2 hidden layers - 512 units	0.9941	0.9929	0.9933	1	1	**1**
1 hidden layer - 1024 units	0.9941	0.9929	0.9933	0.9941	0.9929	0.9933
2 hidden layers - 1024 units	0.9941	0.9929	0.9933	1	1	**1**
Proposed method (AE-XGB)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9875	0.9866	0.9866	1	1	**1**
2 hidden layers - 512 units	0.9875	0.9866	0.9866	1	1	**1**
1 hidden layer - 1024 units	0.9941	0.9929	0.9933	0.9941	0.9929	0.9933
2 hidden layers - 1024 units	0.9941	0.9929	0.9933	0.9941	0.9929	0.9933
Proposed method (AE-SVM)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9941	0.9929	0.9933	1	1	**1**
2 hidden layers - 512 units	0.9941	0.9929	0.9933	1	1	**1**
1 hidden layer - 1024 units	0.9941	0.9929	0.9933	1	1	**1**
2 hidden layers - 1024 units	0.9941	0.9929	0.9933	1	1	**1**

Table 8: Experimental results with the GW2 dataset (noise rate: 10%) for the proposed methods (AE, AE-FE), considering different model architectures, noise rates and time series lengths. Best results in terms of F-Score are marked in bold.

Noise rate	**Time series length**					
10%	1 sec			3 secs		
Conv1D optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.9830	0.9786	0.9797	0.8355	0.8857	0.8501
1024 dense units	0.5375	0.6786	0.5822	0.5363	0.6786	0.5824
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.8297	0.8786	0.8433	0.5230	0.6786	0.5706
1024 dense units	0.8349	0.8857	0.8502	0.6874	0.7928	0.7205
Conv1D (2) optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.9830	0.9785	0.9796	0.8474	0.8928	0.8628
1024 - 512 dense units	0.9783	0.9714	0.9726	0.8363	0.8785	0.8492
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.9830	0.9785	0.9796	0.9783	0.9714	0.9726
1024 - 512 dense units	0.9830	0.9785	0.9796	0.9830	0.9785	0.9796
Deep Filtering optimized	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
	0.6492	0.7000	0.6109	0.6874	0.7928	0.7205
Proposed method (AE)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9762	0.9733	0.9735	0.9762	0.9733	0.9735
2 hidden layers - 512 units	0.9762	0.9733	0.9735	0.9762	0.9733	0.9735
1 hidden layer - 1024 units	0.9762	0.9733	0.9735	0.9762	0.9733	0.9735
2 hidden layers - 1024 units	0.9302	0.9000	0.9038	0.9699	0.9666	0.9668
Proposed method (AE-LR)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9712	0.9643	0.9662	0.9941	0.9929	**0.9933**
2 hidden layers - 512 units	0.9771	0.9714	0.9730	0.9941	0.9929	**0.9933**
1 hidden layer - 1024 units	0.9601	0.9500	0.9526	0.9941	0.9929	**0.9933**
2 hidden layers - 1024 units	0.9824	0.9786	0.9798	0.9941	0.9929	**0.9933**
Proposed method (AE-GBT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9937	0.9933	**0.9933**	0.9814	0.9800	0.9800
2 hidden layers - 512 units	0.9866	0.9866	0.9866	0.9875	0.9866	0.9867
1 hidden layer - 1024 units	0.9805	0.9800	0.9800	0.9714	0.9666	0.9668
2 hidden layers - 1024 units	0.9751	0.9733	0.9734	0.9814	0.9800	0.9800
Proposed method (AE-ERT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9824	0.9786	0.9798	0.9941	0.9929	**0.9933**
2 hidden layers - 512 units	0.9830	0.9786	0.9797	0.9941	0.9929	**0.9933**
1 hidden layer - 1024 units	0.9882	0.9857	0.9865	0.9941	0.9929	**0.9933**
2 hidden layers - 1024 units	0.9882	0.9857	0.9865	0.9941	0.9929	**0.9933**
Proposed method (AE-RF)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9941	0.9929	**0.9933**	0.9941	0.9929	**0.9933**
2 hidden layers - 512 units	0.9882	0.9857	0.9865	0.9941	0.9929	**0.9933**
1 hidden layer - 1024 units	0.9941	0.9929	**0.9933**	0.9941	0.9929	**0.9933**
2 hidden layers - 1024 units	0.9882	0.9857	0.9865	0.9941	0.9929	**0.9933**
Proposed method (AE-XGB)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9941	0.9929	**0.9933**	0.9675	0.9670	0.9666
2 hidden layers - 512 units	0.9680	0.9661	0.9664	0.9755	0.9723	0.9730
1 hidden layer - 1024 units	0.9667	0.9607	0.9596	0.9882	0.9857	0.9865
2 hidden layers - 1024 units	0.9665	0.9670	0.9665	0.9866	0.9866	0.9866
Proposed method (AE-SVM)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9941	0.9929	**0.9933**	0.9941	0.9929	**0.9933**
2 hidden layers - 512 units	0.9941	0.9929	**0.9933**	0.9941	0.9929	**0.9933**
1 hidden layer - 1024 units	0.9941	0.9929	**0.9933**	0.9941	0.9929	**0.9933**
2 hidden layers - 1024 units	0.9875	0.9866	0.9866	0.9941	0.9929	**0.9933**

Table 9: Experimental results with the GW2 dataset (noise rate: 25%) for the proposed methods (AE, AE-FE), considering different model architectures, noise rates and time series lengths. Best results in terms of F-Score are marked in bold.

Noise rate	**Time series length**					
25%	1 sec			3 secs		
Conv1D optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.8783	0.7964	0.7505	0.6842	0.7786	0.7126
1024 dense units	0.7833	0.7991	0.7391	0.5311	0.6750	0.5760
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.9530	0.9411	0.9390	0.6698	0.7741	0.7005
1024 dense units	0.8355	0.8857	0.8501	0.5533	0.700	0.6027
Conv1D (2) optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.9312	0.8848	0.8565	0.9718	0.9642	0.9661
1024 - 512 dense units	0.9830	0.9785	0.9796	0.7007	0.7928	0.7323
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.9830	0.9785	0.9796	0.9783	0.9714	0.9726
1024 - 512 dense units	0.9830	0.9785	0.9796	0.5249	0.6714	0.5695
Deep Filtering optimized	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
	0.6896	0.7785	0.7188	0.8355	0.8857	0.8500
Proposed method (AE)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.8903	0.7067	0.7471	0.9051	0.6066	0.6846
2 hidden layers - 512 units	0.8974	0.7000	0.7469	0.8936	0.5866	0.6688
1 hidden layer - 1024 units	0.8828	0.6466	0.6987	0.9031	0.6133	0.6884
2 hidden layers - 1024 units	0.9105	0.6600	0.7260	0.9031	0.6133	0.6884
Proposed method (AE-LR)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9541	0.9408	0.9430	0.9765	0.9717	0.9730
2 hidden layers - 512 units	0.9510	0.9420	0.9435	0.9679	0.9661	0.9665
1 hidden layer - 1024 units	0.9451	0.9318	0.9342	0.9743	0.9693	0.9707
2 hidden layers - 1024 units	0.9538	0.9464	0.9482	0.9666	0.9631	0.9641
Proposed method (AE-GBT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9759	0.9733	0.9733	0.9762	0.9733	0.9735
2 hidden layers - 512 units	0.9626	0.9600	0.9600	0.9876	0.9866	0.9867
1 hidden layer - 1024 units	0.9937	0.9933	**0.9933**	0.9742	0.9733	0.9733
2 hidden layers - 1024 units	0.9514	0.9466	0.9470	0.9546	0.9533	0.9533
Proposed method (AE-ERT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9669	0.9598	0.9617	0.9828	0.9786	0.9797
2 hidden layers - 512 units	0.9564	0.9488	0.9503	0.9790	0.9771	0.9776
1 hidden layer - 1024 units	0.9683	0.9595	0.9615	0.9826	0.9786	0.9798
2 hidden layers - 1024 units	0.9620	0.9557	0.9572	0.9615	0.9595	0.9598
Proposed method (AE-RF)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9769	0.9714	0.9730	0.9838	0.9813	0.9821
2 hidden layers - 512 units	0.9548	0.9494	0.9505	0.9745	0.9726	0.9731
1 hidden layer - 1024 units	0.9744	0.9693	0.9708	0.9820	0.9789	0.9798
2 hidden layers - 1024 units	0.9595	0.9536	0.9550	0.9572	0.9551	0.9553
Proposed method (AE-XGB)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9622	0.9592	0.9597	0.9546	0.9533	0.9530
2 hidden layers - 512 units	0.9282	0.9262	0.9260	0.9548	0.9524	0.9529
1 hidden layer - 1024 units	0.9594	0.9551	0.9552	0.9723	0.9702	0.9709
2 hidden layers - 1024 units	0.9321	0.9304	0.9306	0.9503	0.9494	0.9487
Proposed method (AE-SVM)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9832	0.9815	0.9821	0.9902	0.9881	**0.9888**
2 hidden layers - 512 units	0.9536	0.9473	0.9482	0.9745	0.9726	0.9731
1 hidden layer - 1024 units	0.9832	0.9815	0.9821	0.9863	0.9833	0.9843
2 hidden layers - 1024 units	0.9586	0.9542	0.9551	0.9701	0.9682	0.9687

Table 10: Experimental results with the GW2 dataset (noise rate: 50%) for the proposed methods (AE, AE-FE), considering different model architectures, noise rates and time series lengths. Best results in terms of F-Score are marked in bold.

Noise rate	**Time series length**					
50%	1 sec			3 secs		
Conv1D optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.5982	0.5303	0.3962	0.6081	0.6920	0.6102
1024 dense units	0.2898	0.4991	0.3334	0.5274	0.6643	0.5682
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.8491	0.7884	0.7493	0.3822	0.5857	0.4469
1024 dense units	0.8330	0.8786	0.8470	0.2467	0.500	0.3300
Conv1D (2) optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.9724	0.9642	0.9659	0.9724	0.9642	0.9659
1024 - 512 dense units	0.9483	0.9339	0.9319	0.6357	0.6991	0.5974
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.8308	0.8785	0.8430	0.8422	0.8857	0.8559
1024 - 512 dense units	0.8229	0.8651	0.8358	0.6783	0.7714	0.7058
Deep Filtering optimized	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
	0.5200	0.6428	0.5427	0.7490	0.7419	0.6782
Proposed method (AE)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.8787	0.4933	0.5994	0.8759	0.5000	0.6021
2 hidden layers - 512 units	0.8787	0.4933	0.5994	0.8759	0.5000	0.6021
1 hidden layer - 1024 units	0.8787	0.4933	0.5994	0.8759	0.5000	0.6021
2 hidden layers - 1024 units	0.8787	0.4933	0.5994	0.8759	0.5000	0.6021
Proposed method (AE-LR)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9310	0.9080	0.9101	0.9466	0.9366	0.9392
2 hidden layers - 512 units	0.9098	0.8973	0.8983	0.9279	0.9259	0.9263
1 hidden layer - 1024 units	0.9198	0.9027	0.9043	0.9518	0.9438	0.9460
2 hidden layers - 1024 units	0.9265	0.9170	0.9188	0.9284	0.9250	0.9261
Proposed method (AE-GBT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9511	0.9466	0.9468	0.9814	0.9800	0.9800
2 hidden layers - 512 units	0.9628	0.9600	0.9602	0.9875	0.9866	**0.9866**
1 hidden layer - 1024 units	0.9743	0.97333	**0.9734**	0.9552	0.9466	0.9474
2 hidden layers - 1024 units	0.8916	0.8800	0.8809	0.9813	0.9800	0.9800
Proposed method (AE-ERT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9471	0.9366	0.9390	0.9601	0.9500	0.9526
2 hidden layers - 512 units	0.9223	0.9098	0.9116	0.9556	0.9518	0.9530
1 hidden layer - 1024 units	0.9512	0.9357	0.9384	0.9654	0.9571	0.9595
2 hidden layers - 1024 units	0.9341	0.9232	0.9254	0.9296	0.9259	0.9263
Proposed method (AE-RF)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9601	0.9500	0.9526	0.9631	0.9580	0.9597
2 hidden layers - 512 units	0.9265	0.9170	0.9188	0.9420	0.9384	0.9395
1 hidden layer - 1024 units	0.9526	0.9438	0.9460	0.9578	0.9509	0.9528
2 hidden layers - 1024 units	0.9341	0.9232	0.9254	0.9237	0.9196	0.9196
Proposed method (AE-XGB)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9228	0.9188	0.9195	0.9288	0.9259	0.9259
2 hidden layers - 512 units	0.8900	0.8848	0.8852	0.9150	0.9116	0.9127
1 hidden layer - 1024 units	0.9307	0.9241	0.9259	0.9479	0.9455	0.9463
2 hidden layers - 1024 units	0.8884	0.8848	0.8856	0.9095	0.9071	0.9063
Proposed method (AE-SVM)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.9613	0.9589	0.9597	0.9824	0.9786	0.9798
2 hidden layers - 512 units	0.9200	0.9098	0.9121	0.9345	0.9321	0.9329
1 hidden layer - 1024 units	0.9613	0.9589	0.9597	0.9765	0.9714	0.9731
2 hidden layers - 1024 units	0.9321	0.9241	0.9257	0.9377	0.9313	0.9327

Table 11: Experimental results with the GW3 dataset for the proposed methods (AE, AE-FE), considering different model architectures and time series lengths. Best results in terms of F-Score are marked in bold.

	Time series length					
Method	1 sec			3 secs		
Conv1D optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.5553	0.6135	0.5622	0.2500	0.5000	0.3330
1024 dense units	0.4948	0.5395	0.4884	0.2653	0.5000	0.3466
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 dense units	0.5671	0.5374	0.5202	0.5668	0.5608	0.4975
1024 dense units	0.6145	0.5972	0.5889	0.5069	0.5791	0.5061
Conv1D (2) optimized						
Dropout: 0.1	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.5264	0.5652	0.4566	0.6027	0.5702	0.5522
1024 - 512 dense units	0.7059	0.5940	0.5463	0.4250	0.4966	0.3898
Dropout: 0.3	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
512 - 256 dense units	0.5390	0.5499	0.4869	0.6764	0.6262	0.6096
1024 - 512 dense units	0.5372	0.5700	0.5211	0.5576	0.5976	0.5521
Deep Filtering optimized	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
	0.3344	0.5020	0.3911	0.3704	0.5344	0.4155
Proposed method (AE)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	1	0.4693	0.6388	0.9474	0.4897	0.6292
2 hidden layers - 512 units	0.7782	0.4285	0.5409	0.7954	0.4183	0.5375
1 hidden layer - 1024 units	0.9793	0.4693	0.6302	0.8994	0.4744	0.6040
2 hidden layers - 1024 units	0.8818	0.4489	0.5854	0.8897	0.4285	0.5590
Proposed method (AE-LR)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.6836	0.6764	0.6744	0.7935	0.7597	0.7629
2 hidden layers - 512 units	0.6945	0.6819	0.6795	0.6458	0.6194	0.5881
1 hidden layer - 1024 units	0.6290	0.6597	0.6342	0.7984	0.7778	0.7797
2 hidden layers - 1024 units	0.5936	0.6181	0.5866	0.7744	0.7583	0.7607
Proposed method (AE-GBT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.6828	0.6020	0.6256	0.6125	0.5663	0.5789
2 hidden layers - 512 units	0.6139	0.5867	0.5925	0.6800	0.6479	0.6559
1 hidden layer - 1024 units	0.6687	0.6020	0.6192	0.6910	0.6173	0.6353
2 hidden layers - 1024 units	0.6630	0.6428	0.6464	0.6917	0.6326	0.6481
Proposed method (AE-ERT)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.7065	0.6931	0.6938	0.8130	0.7771	0.7805
2 hidden layers - 512 units	0.7161	0.7104	0.7090	0.6847	0.6181	0.5804
1 hidden layer - 1024 units	0.7243	0.7139	0.7144	0.8266	0.7917	0.7947
2 hidden layers - 1024 units	0.7054	0.6875	0.6855	0.7482	0.7375	0.7355
Proposed method (AE-RF)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.7112	0.7042	0.7021	0.7846	0.7597	0.7598
2 hidden layers - 512 units	0.7203	0.7208	**0.7160**	0.6697	0.6111	0.5730
1 hidden layer - 1024 units	0.7203	0.7111	0.7106	0.8167	0.7875	0.7877
2 hidden layers - 1024 units	0.6963	0.6806	0.6809	0.7444	0.7361	0.7345
Proposed method (AE-XGB)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.6970	0.6972	0.6918	0.8236	0.8194	**0.8194**
2 hidden layers - 512 units	0.6969	0.6931	0.6916	0.6456	0.6083	0.5806
1 hidden layer - 1024 units	0.7031	0.6972	0.6969	0.8177	0.8069	0.8063
2 hidden layers - 1024 units	0.6820	0.6750	0.6684	0.7077	0.6986	0.6980
Proposed method (AE-SVM)	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
1 hidden layer - 512 units	0.7040	0.6917	0.6932	0.7776	0.7653	0.7666
2 hidden layers - 512 units	0.7063	0.7028	0.7021	0.6723	0.6375	0.6105
1 hidden layer - 1024 units	0.6234	0.6639	0.6354	0.8176	0.7917	0.7934
2 hidden layers - 1024 units	0.6737	0.6486	0.6403	0.7869	0.7653	0.7683

Table 12: Wilcoxon Signed Rank Tests (all datasets). Bold: improvement is statistically significant when the p-value is smaller than 0.01.

Pairwise comparison	p-value	winner
RMSE criterion		
Conv1D *vs* AE	**4.54E-06**	AE
Conv1D (2) *vs* AE	9.80E-02	Conv1D (2)
Deep Filtering *vs* AE	**1.82E-06**	AE
Conv1D vs AE-LR	**1.18E-13**	AE-LR
Conv1D (2) vs AE-LR	**1.21E-06**	AE-LR
AE vs AE-LR	**5.26E-12**	AE-LR
Deep Filtering vs AE-LR	**7.85E-15**	AE-LR
Conv1D *vs* AE-GBT	**2.46E-14**	AE-GBT
Conv1D (2) *vs* AE-GBT	**3.87E-10**	AE-GBT
AE *vs* AE-GBT	**1.27E-11**	AE-GBT
Deep Filtering *vs* AE-GBT	**7.85E-15**	AE-GBT
Conv1D vs AE-ERT	**3.61E-14**	AE-ERT
Conv1D (2) vs AE-ERT	**1.97E-09**	AE-ERT
AE vs AE-ERT	**4.06E-14**	AE-ERT
Deep Filtering vs AE-ERT	**7.85E-15**	AE-ERT
Conv1D vs AE-RF	**3.61E-14**	AE-RF
Conv1D (2) vs AE-RF	**3.99E-10**	AE-RF
AE vs AE-RF	**8.74E-14**	AE-RF
Deep Filtering vs AE-RF	**7.85E-15**	AE-RF
Conv1D vs AE-XGB	**3.93E-14**	AE-XGB
Conv1D (2) vs AE-XGB	**7.99E-06**	AE-XGB
AE vs AE-XGB	**2.02E-12**	AE-XGB
Deep Filtering vs AE-XGB	**7.85E-15**	AE-XGB
Conv1D vs AE-SVM	**3.61E-14**	AE-SVM
Conv1D (2) vs AE-SVM	**1.45E-11**	AE-SVM
AE vs AE-SVM	**4.06E-14**	AE-SVM
Deep Filtering vs AE-SVM	**7.85E-15**	AE-SVM