Exploiting Time Dynamics for One-Class and Open-Set Anomaly Detection

Lorenzo Brigato[1], Riccardo Sartea[2], Stefano Simonazzi[2], Alessandro Farinelli[2], Luca Iocchi[1], and Christian Napoli[1]

[1] Department of Computer, Automation and Management Engineering
La Sapienza University of Rome,
Viale Ariosto 35, 06121 Roma, Italy
{brigato; iocchi; cnapoli}@diag.uniroma1.it

[2] Department of Computer Science
University of Verona,
Strada le Grazie 15, 37134 Verona, Italy
{alessandro.farinelli; riccardo.sartea}@univr.it

Abstract. In this paper we describe and compare multiple one-class anomaly detection methods for Cyber-Physical Systems (CPS) that can be trained with data collected only during normal behaviors. We also consider the problem of detecting which group of sensors is most affected by the anomalous situation solving an open-set classification task. The proposed methods are domain independent and are based on a temporal analysis of data collected by the system. More specifically, we use different flavours of deep learning architectures, including recurrent neural networks (RNN), temporal convolutional networks (TCN), and autoencoders. Experimental results are conducted in three different scenarios with publicly available datasets: social robots, autonomous boats and water treatment plants (SWaT dataset). Quantitative results on these datasets show that our approach achieves comparable results with respect to state of the art approaches and promising results for open-set classification.

1 Introduction

Cyber-Physical Systems (CPS) such as robots, industrial machinery, autonomous vehicles, etc., are employed in an ever increasing variety of environments. In many cases these systems have a direct impact on society, e.g., power or water treatment plants. It is therefore crucial to ensure the correct and safe operation of CPS, as the consequence of unexpected failure or disruption could harm people and business. In this work we aim to detect behaviors of CPS that do not conform to the known normal operation, i.e., anomalies. Furthermore, since attacks to CPS cause unwanted behaviors that differ from the normal, detecting an anomaly allows also to detect the underlying attack. A key problem of anomaly detection is the lack of data regarding the anomalies. In fact, it is easy to profile the normal behavior of a CPS, however it is more complex to observe the CPS under a realistic anomalous behavior. Moreover, it is almost impossible to predict all the possible types of anomalies that may occur in complex CPS. For this reason, the anomaly detection problem is often addressed as a one-class classification problem, i.e., the only available data for training belongs to the class referring to the normal behavior. The works in [1] proposes a multivariate anomaly detection method based on Generative Adversarial Networks (GAN), using Long Short-Term Memory Recurring Neural Networks (LSTM-RNN) for generator and discriminator models. After the training process, the discriminator is used to distinguish between normal and abnormal behavior. The DAGMM approach proposed in [2] makes use of autoencoders and Gaussian Mixture Model (GMM) that takes the low-dimensional input from the compression network and outputs mixture membership prediction for each sample. Anomaly detection for CPS, and specifically robots, has been recently addressed in [3], where authors propose to extract system logs from a set of internal variables of a robotic system, transform such data into images, and train different Autoencoder architectures to detect anomalies. In this work, we propose a new transformation of the input specifically designed for Convolutional Autoencoders that takes time into account. We show that such transformation increases the robustness of Convolutional Autoencoders, without changing the underlying model. In this work we also propose a regression approach to the open-set classification of anomalies that is capable of identifying which sensors of a CPS are most likely the cause of a detected anomaly. This information is very valuable as it allows for an interpretation of the anomaly and helps the analysis of its causes. In particular, the open-set refers to the different classes of possible anomalies that are not known at training time since the training process is one-class, i.e., only normal behavior. In the empirical evaluation we compare our solutions with several statistical methods and state of the art approaches for anomaly detection, considering techniques that use both explicit [4],[5] and implicit [6],[7],[8],[9],[1] representations of the time dimension. We use three datasets for testing: autonomous water drone *Boat* [3], the *Pepper* social robot [3], and *SWaT* [10]. The first two were collected from robotic platforms[3], whereas the third one was collected from an industrial CPS of a water treatment plant. Moreover, we also show a use case for the *Pepper* social robot involved in a public demonstration subject to a simulated attack. This use case is employed as additional test set for the anomaly detection methods described in the paper. Crucially, in this test we run models trained on the mentioned dataset on a log that has been acquired in a very different situation, e.g., different environment, different means of interaction. The analysis confirms that the simulated attack can be detected by using our approach. Overall results of the experiments show that deep learning architectures are suitable to detect anomalies in one-class and open-set settings and that exploiting time dynamics generally improve such performance. In summary, the contributions of this paper are the following: *i)* we propose a new transformation of the input specifically designed for Convolutional Autoencoders that

[3] `sites.google.com/diag.uniroma1.it/robsec-data`

characterize the temporal evolution of system variables; *ii)* we propose a novel approach based on regression for the open-set classification of anomalies that is capable of identifying which sensors of a CPS are most likely the cause of a detected anomaly; *iii)* we conduct an extensive empirical evaluation of multiple techniques on real datasets of robotic platforms and industrial CSP. Moreover, we also present the results on a different instance of real usage scenario for the *Pepper* platform.

2 Problem definition

Let us consider a general CPS $\mathcal{S}$ that can be described by a set of variables V characterizing its internal state over time. We denote $K = |V|$. The evolution over time of V characterizes the behavior of S, that can either be *normal* or fall into a class of *anomalies*, generally denoted as *abnormal.* Let r_t be a record, i.e., a tuple of values of V collected at time t, note that $|r_t| = K$. We define $\pi_{t,d} = \{r_{t-d}, \dots, r_t\}$ a system log of a behavior containing the values of variables V recorded from time $(t - d)$ to time t. Let us consider a scenario in which the logs are taken when $\mathcal{S}$ is performing a normal behavior, we denote with D_r the set of collected normal behavior records, and D_t a set of unknown behaviors potentially containing both normal and abnormal records. The first problem we define aims at producing a binary classifier of normal and abnormal behaviors of system logs, trained only with normal samples. The practical importance of this problem is that it is not necessary to collect abnormal situations, which are often very difficult to acquire in a significant quantity and variety to train a reliable model.

Definition 1 One-class classification of system logs. *Given a dataset of system logs D_r, generate a model that is able to classify new instances $\pi_{t,d} \in D_t$.*

The second problem considers the use of the model computed as a solution to Problem Definition 1, to compute the subset of variables $H \in 2^V$ that better explains the anomaly, when this is detected. This problem is also of practical importance, as the information produced helps in directing the investigation of the causes towards the sensors that are most likely involved in the anomaly.

Definition 2 Open-set classification of system logs. *Given a dataset of system logs D_r, generate a model that is able to classify new instances $\pi_{t,d} \in D_t$ and, when an anomaly is detected, estimate the subset of variables $H \in 2^V$ that contribute the most to the anomaly.*

3 Log-to-temporal-image Anomaly Detection

We propose a new solution to the one-class classification problem based on transforming logs into images described in [3]. We focus our attention on the temporal aspect of the logs and aim to improve the performance of Convolutional Autoencoders that resulted to be the worst models in terms of performance [3]. A key element in our approach is the transformation of each record $r_t \in R$ into a squared image $I \in \mathcal{I}$ (denoting with $\mathcal{I}$ the set of images). To this end, we implemented a *log-to-image transformation function* $\sigma : R \rightarrow \mathcal{I}$ able to transform log records into images. $D_{r,I} = \{(\sigma(r_t), \mathit{normal}) \mid \forall\, r_t \in D_r\}$ computed from the system logs that will be used for generating the class model. More precisely, given dataset D_r we define a new dataset of labeled images $D_{r,I} = \{(\sigma(r_t), \mathit{normal}) \mid \forall\, r_t \in D_r\}$ computed from the system logs that will be used for generating the class model

3.1 Log-to-temporal image Transformation

Since logs represent temporal data, the temporal information includes important features that may be needed to detect anomalies. However, convolutions do not exploit such information since their filters work over spatial dimensions that do to take time into account. Moreover, features in data are expected to be dense. This condition is not fulfilled unless the input space is low dimensional. The standard *min_max* normalization implemented in [3] penalized Convolutional Autoencoders when tested on *Pepper* dataset. Indeed, 1) each image does not consider the values of the previous record, 2) the vector is simply reshaped to generate an image (padding if needed) so, values are located without following any reasoning. To solve the mentioned issues we define a log-to-image transformation function $\sigma : R \rightarrow \mathcal{I}$ implemented as follows. Let $(r_k)_t$ be the k-th variable within a record at time t and let $(\rho_k)_t = \frac{\partial (r_k)_t}{\partial t}$ and $(\theta_k)_t = \frac{\partial^2 (r_k)_t}{\partial t^2}$ be respectively the first and second time derivatives. They characterize the temporal evolution of the logs variables. Variables that follow similar patterns through time should be closely located to each other. On the other hand, variables whose temporal profile greatly differ should be mapped far from each other. In this manner, convolution kernels can extract the information needed to characterize the normality of the current sample. The first step regards the transformation of the derivatives to polar coordinates. The first derivatives are interpreted as radii, in polar coordinates, that originates at the image center. Since the image edge has a defined dimension I_e, $(\rho_k)_t$ is remapped to the interval $[0,\ \frac{I_e}{2} - 1]$. The absolute value of the second derivatives $|(\theta_k)_t|$ is then interpreted as the polar coordinates' angle, and therefore normalized accordingly to belong to $[0,\ 2\pi]$. The image row $(i_k)_t$ and column $(j_k)_t$ to which a variable of the log record $(r_k)_t$ is mapped, are computed as:

$$\begin{aligned} (i_k)_t &= \left\lfloor (\rho_k)_t \cos(\theta_k)_t + \tfrac{I_e}{2} \right\rfloor \\ (j_k)_t &= \left\lfloor (\rho_k)_t \sin(\theta_k)_t + \tfrac{I_e}{2} \right\rfloor \end{aligned} \tag{1}$$

Note that i and j are floored to integer values. Some variables could be mapped to the same pixel, for instance, constant variables. We will refer to this circumstance as pixel superposition. Generally, when such a superposition occurs it means that some variables are concurrently and consistently varying over time. In Equation (2) we tackle this problem and the related formal complications. We could consider such covaring variables as correlated noise. In this latter fashion, it is useful to drop out such variables since they can become detrimental for

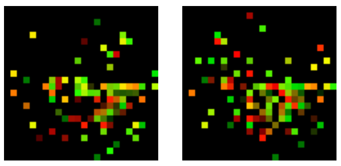

Fig. 1. Temporal image generated from a *normal* (left) and an *abnormal* (right) sample taken from *Pepper* dataset.

correct classification. On the other hand, the produced image size matters for the quantization error. In fact, small outputs will determine a high probability of pixel superposition whether or not variables covariate. We let the transformation overwrite values over the same pixels. Experiments proved that such mapping does not hurt performance. The image I is composed of three channels R, G and B. The red channel is filled with the normalized value of the log record $(\hat{r}_k)_t = \text{minmax}((r_k)_t)$. The green channel refers to the semantic meaning of the variable while the blue channel is filled with zeros. In order to compose a mathematically consistent formalism, let define

$$\begin{aligned} \Lambda_t &= \{((i_k)_t, (j_k)_t)\ \forall\ k \in [0, K] \cap \mathbb{N}\} \\ \bar{k}_{ijt} &\triangleq \max\{k : (i_k)_t = i, (j_k)_t = j\} \\ \chi_{ijt} &\triangleq [(\hat{r}_{\bar{k}_{ijt}})_t, \tfrac{\bar{k}_{ijt}}{K}, 0] \end{aligned} \tag{2}$$

It follows that for each pixel in position (i, j) at a time instant t is represented by an RGB vector computed as

$$\mathbf{I}_{ij}(t) \triangleq \begin{cases} \chi_{ijt} & (i, j) \in \Lambda_t \\ [0, 0, 0] & (i, j) \notin \Lambda_t \end{cases} \tag{3}$$

Normalization bounds (minimum and maximum) are based on the training data distribution. At testing time, some values might be mapped outside the desired intervals. In this latter circumstance, these values are forced to the closest allowed value. An example of a couple of generated images from the *Pepper* dataset is shown in Figure 1.

3.2 Class Modelling

Class modelling is performed through two different phases, as done in [3]. Since in this work we used images transformed following the previously described strategy, we will refer to this method as (*tConvEnc*). The network is trained over the majority of the one-class dataset D_r to learn its latent representation. Then, the affiliation to the *normal* class follows a decision rule based on a threshold. Such threshold is computed through the loss function and estimated on a smaller sample that we will call $D_{thr} \subset D_r$. D_{thr} is not fed to the autoencoder. Given

D_{thr} and the loss function $\mathcal{L}$ of a trained network, we compute $\mathbf{l} = \mathcal{L}(D_{thr})$, with $\mathbf{l}$ being a vector containing the loss values for each record $r_t \in D_{thr}$. We keep the approach used in [3] to compute the range of expected *normal* losses. We choose the upper and lower bounds as $\delta_u,\ \delta_l = \mu(\mathbf{l}) \pm z \cdot \sigma(\mathbf{l})$, where μ is the mean and σ is the standard deviation of the values in $\mathbf{l}$. The value of z can be tuned to vary the interval. A testing sample is classified as *normal* if the value of its loss l_t is in between δ_u and δ_l.

3.3 Architecture of the Autoencoders

Pepper: The *tConvEnc* is composed by eight convolutional layers with kernel size of dimensions $(6, 6)$. The first three are followed by Max-pooling layers with stride dimensions equal to 2 whereas the central three layers with Up-sampling. The kernels for each convolutional layer are respectively [100, 50, 30, 10, 30, 50, 100, 3]. The batch size was set to 64. **Boat**: The *tConvEnc* is composed by six convolutional layers with kernel size of dimensions $(3, 3)$. The first two are followed by Max-pooling layers with stride dimensions equal to 2 whereas the central two layers with Up-sampling. The kernels for each convolutional layer are respectively $[32, 16, 8, 16, 32, 3]$. The batch size was equal to 64. **SWaT**: The *tConvEnc* is composed by six convolutional layers with kernel size of dimensions $(6, 6)$. The first two are followed by Max-pooling layers with stride dimensions equal to 2 whereas the central two layers with Up-sampling. The kernels for each convolutional layer are respectively $[32, 16, 8, 16, 32, 3]$. The batch size was equal to 1024. All input images are zero-padded. Each network was trained with Binary Cross-Entropy Loss and Adam optimizer (default learning rate). We fixed the number of epochs to 30. We used ReLUs for all hidden layers and Sigmoids for the output layers.

4 Regression

As an alternative approach we propose to use regression temporal models in order to address both problem definitions of Section 2. We used three different models: a Recurrent Neural Network (RNN), a dense neural network and a Temporal Convolutional Networks (TCN). The RNN has two layers and is composed of Gated Recurrent Units (GRU) gating mechanism with a hidden layer size of 64. The hidden layer is connected to the network output through a dropout layer. The dense neural network has a batch-norm layer, two hidden layers of dimension 64 units, both followed by dropout. The TCN uses 3 layers of size 30 and has kernel size of 3. All models take in input a contiguous time sequence of size (K, t), where t is the number of time intervals considered $(1 < t)$ and return an output of size K. We trained RNN with Backpropagation Through Time (BPTT). On the other hand, we created and shuffled all possible contiguous subsequences of length t for TCN and dense network.

The model, used for regression, takes in input a sequence $\pi_{t-1,d-1}$ and returns in output the predicted value of r_t, called $\widehat{r}_t$. To train the networks, Mean

Squared Error (MSE) loss and Adam optimizer have been employed, using as labels the values in the log at time t, r_t. We fixed the dropout rate to 0.4, and the number of epochs has been tuned to maximize performance while avoiding over-fitting. The evaluation of a new sequence $\pi_{t,d} \in D_t$ follows these steps: if the MSE between the predicted values $\widehat{r}_t$ and the actual values $r_t \in \pi_{t,d}$ is above the threshold tr an attack is detected. The threshold is computed from the predictions $Pred_t$ on the training data D_r by selecting the p percentile (usually between 70 and 99.5).

$$Pred_t = \frac{\sum_{i=1}^{K}(r_{t,i} - \widehat{r}_{t,i})^2}{K} \tag{4}$$

$$tr = Perc_p(\{Pred_1, ..., Pred_{|D_r|}\}) \tag{5}$$

5 Open-set classification

To solve the problem in Definition 2, we consider a variation of the regression approach presented in Section 4 by changing how the results of the regression model are used (while the training does not change). In particular, instead of computing a single threshold tr as before, we compute a threshold t_i for each dimension $0 < i \leq K$ as follows

$$Pred_{t,i} = (r_{t,i} - \widehat{r}_{t,i})^2 \tag{6}$$

$$tr_i = Perc_p(\{Pred_{1,i}, ..., Pred_{|D_r|,i}\}) \tag{7}$$

Now, suppose we want to classify a new sequence $\pi_{t,d} \in D_t$. First, we retrieve the model prediction $\widehat{r}_t$ (just as in Section 4). Then, we compute the squared errors between every dimension $r_{t,i}$ and $\widehat{r}_{t,i}$ (as in Equation 6) and divide the result by the respective threshold tr_i, in order to normalize between the thresholds of the dimensions. After that, dimensions whose normalized values are greater than 1 are ranked according to such values, and the first k dimensions determine a subset of variables, called H. The value k is domain dependent and can be set by using different criteria, such as the number of variables present in the dataset, by fixing a threshold on the ranking value, or by considering the number of variables typically involved in an anomaly (or attack). In our empirical evaluation we use this later criteria (see Section 6.2).

6 Experimental Results

6.1 Anomaly detection results

In the experimental results we follow the standard anomaly detection nomenclature: we consider abnormal samples as positives and normal samples as negatives. Therefore, true positive are abnormal recognized abnormal. Due to unbalanced data, typically used in anomaly detection, in addition to precision and recall, we consider F1-score. For all domains, we computed the results by varying the

	t	5			20		
	perc.	P	R	F1	P	R	F1
	70	0.820	**0.970**	0.889	0.794	**0.971**	0.874
	80	0.852	0.949	**0.898**	0.852	0.947	0.897
dense	90	0.890	0.866	0.878	0.896	0.902	**0.899**
	95	**0.922**	0.784	0.847	0.921	0.750	0.827
	99	0.872	0.188	0.309	**0.970**	0.279	0.433
	70	0.841	**0.794**	**0.817**	0.827	**0.797**	**0.812**
	80	0.863	0.708	0.778	0.900	0.711	0.794
RNN	90	**0.894**	0.596	0.715	**0.923**	0.486	0.637
	95	0.884	0.388	0.539	0.920	0.394	0.552
	99	0.694	0.039	0.075	0.883	0.026	0.051
	70	0.822	**0.726**	**0.771**	0.890	**0.729**	**0.802**
	80	0.841	0.607	0.705	0.888	0.614	0.726
TCN	90	0.876	0.451	0.595	**0.929**	0.548	0.689
	95	**0.878**	0.370	0.521	0.917	0.373	0.530
	99	0.644	0.033	0.063	0.899	0.018	0.036

Table 1. One-class problem on *Pepper*. Results for the RNN, Dense and TCN. Results are given in terms of Precision (P), Recall (R), F1-score (F1).

number of records t and the percentile value used to derive the threshold. Table 1 shows the values for *Pepper*, the *SWaT* behavior is similar, while for *Boat* most values are close to 1. We can see that all models have similar results, with slightly better results for the dense model, particularly in the recall. The precision remains high, exceeding 92% in some cases. As expected, we can observe that, as the percentile threshold value increases, the precision increases while reducing the recall and vice-versa. Instead, against expectations, we do not notice big differences by changing the number t of records considered. The best results, considering F1-score, were achieved by choosing as a percentile a value of 80 for *Pepper* and *Boat* while 99.5 for *SWaT*. We suppose that this is affected primarily by two factors: (i) the different distribution of normal and attack data, dictated by the availability of data, (ii) *SWaT* has a regular periodic normal behavior, while for robots, their behaviors vary significantly over time. Table 2 presents the results obtained for the anomaly detection task. We compare our approach with several competitors: One-Class Support-Vector Machine (OC-SVM) [6], Isolation Forest (IF) [7], Kernel Density Estimator (KDE) [8, 9], Generative Adversarial Networks-based Anomaly Detection (GAN-AD) [1], Principal Component Analysis (PCA), K-Nearest Neighbours (KNN), Feature Bagging (FB), Autoencoder (AE), Multivariate Anomaly Detection for time series data with Generative Adversarial Networks (MAD-GAN) [4] and Efficient Gan-based anomaly detection (E-GAN) [5]. In the *SWaT* dataset we observed that one sensor (*AIT201*), from about half the normal test data, has values 4 times higher than the maximum value of the training data, so we try to use only the first n components, obtained with PCA, as proposed in state of the art approaches [1, 4, 5]. We can see that if we use all the original data our regression

	Boat			Pepper			Swat		
Method	P	R	F1	P	R	F1	P	R	F1
1SVM	0.609	1.000	0.757	0.982	0.366	0.533	0.149	0.907	0.256
LOF	0.918	0.071	0.132	0.885	0.323	0.473	0.122	0.996	0.217
EE	0.919	1.000	**0.958**	0.604	0.542	0.571	0.077	0.004	0.007
IF	0.538	0.446	0.487	0.823	0.512	**0.631**	0.256	0.791	0.387
GAN-AD [1]	–	–	–	–	–	–	0.933	0.636	0.750
PCA [4]	–	–	–	–	–	–	0.249	0.216	0.230
KNN [4]	–	–	–	–	–	–	0.078	0.078	0.080
FB [4]	–	–	–	–	–	–	0.102	0.102	0.100
AE [4]	–	–	–	–	–	–	0.726	0.526	0.610
EGAN [5]	–	–	–	–	–	–	0.406	0.677	0.510
MAD-GAN [4]	–	–	–	–	–	–	0.990	0.637	**0.770**
Enc [3]	–	–	**0.99**	–	–	0.90	0.973	0.612	0.751
DeepEnc [3]	–	–	**0.99**	–	–	**0.94**	0.985	0.608	0.752
ConvEnc [3]	–	–	**0.99**	–	–	0.77	0.992	0.618	**0.761**
tConvEnc	0.992	0.989	0.990	0.860	0.946	**0.901**	0.993	0.582	**0.734**
Dense	0.999	1.000	0.999	0.896	0.902	0.899	0.485	0.589	0.532
RNN	1.000	1.000	**1.000**	0.841	0.794	0.817	0.392	0.665	0.493
TCN	1.000	1.000	**1.000**	0.890	0.729	0.802	0.189	0.743	0.301
Dense (pc=8)	0.988	0.998	0.993	0.813	0.890	0.849	0.975	0.627	**0.763**
RNN (pc=8)	0.996	1.000	0.998	0.672	0.236	0.349	0.959	0.632	**0.762**
TCN (pc=8)	0.998	1.000	0.999	0.444	0.143	0.217	0.997	0.592	0.743

Table 2. Comparison of approaches and domains. Results are given in terms of Precision (P), Recall (R) and F1-score (F1). The results are shown in groups (top-down: literature, past experiments, new experiments and new experiments with PCA).

approach is not very effective on *SWaT*. This is not surprising because of the abnormal sensor in the normal data that is recognized. If we use only the first n components instead the anomaly tends to fade and our approach improves performance, reaching, in the case of *SWaT*, the best approaches considered using GAN. It should be noted, however, that our method does not require the use of the principal components and indeed we have verified, with the other datasets, that using only the first n components, the performance drops. The F1-score values also decrease accordingly, from about 0.9 using all sensors: 0.89 with 100 components, 0.86 with 30 components, 0.80 with 10 components, 0.61 with 3 components and 0.24 with only one component. With *SWaT*, the performance increase because by reducing dimensions we are removing the abnormal behavior of sensor *AIT201*. Considering the *tConvEnc*, we tried different image sizes, by changing the dimension of the edge I_e. We found out that large edges increase data sparsity and consequently reduce the performance of *tConvEnc*, while narrow edges increase the probability of pixel superposition. This is reasonable since convolutions prefer dense features. For the *Pepper* dataset we generated images with edge of size $I_e = 24$, for the *Boat* dataset $I_e = 16$ and for the *SWaT* dataset $I_e = 12$. The hyperparameter z has been varied in the range $[0, 6]$. The reported

Boat	dense	RNN	TCN	**Pepper**	dense	RNN	TCN
Dos	0.004	0.692	**0.985**	joint	0.445	0.533	0.494
DosPay	0.000	0.736	**0.963**	led	0.462	0.674	**0.785**
GpsDown	0.000	**0.997**	0.000	wheel	0.145	0.681	0.394
Stuck	0.454	0.550	**0.806**				
avg	*0.115*	*0.744*	*0.688*	avg	*0.351*	*0.630*	*0.558*
Swat	dense	RNN	TCN		dense	RNN	TCN
2	**0.933**	**0.867**	**0.878**	22	0.702	**0.904**	0.436
6	**0.875**	**0.825**	0.775	32	**0.926**	**0.876**	**0.843**
7	**0.931**	**0.897**	**0.885**	38	**0.807**	0.684	0.684
8	**0.974**	**0.959**	0.263	40	0.016	0.016	**0.869**
10	0.121	0.758	**1.000**	41	0.387	**0.834**	0.722
20	**0.925**	**0.875**	**0.888**	avg	*0.290*	*0.317*	*0.298*

Table 3. Open-set classification problem. Results are given in terms of Accuracy. Due to space constraints we only report a subset of all the attacks for $SWaT$

results are obtained choosing z that maximized the F1 score. It is clear that the new transformation increases the robustness of Convolutional Autoencoders. Indeed, *tConvEnc* reaches 0.901 F1-score while *ConvEnc* scored 0.77 on *Pepper*. For the *Boat* and $SWaT$ datasets results are comparable with the normalized *ConvEnc*. We suppose that *ConvEnc* manages to maintain a good performance in these two datasets due to the small input dimensionality and the consequent small image size. The proposed temporal input transformation allows the *tConvEnc* to match the performance of the other autoencoders and state of the art methods. For instance, *tConvEnc* reaches an F1-score of 0.734 which is close the MAD-GAN F1-score of 0.770 on $SWaT$.

6.2 Open-set classification results

In this set of experiments, we evaluate the performance of the system in providing classes of anomalies that are not seen at training time. In order to perform a quantitative analysis, we have used test sets with labels of specific attacks, corresponding to subsets of variables affected by such attacks. More specifically, for each kind of attack in the test set, e.g., an attack that takes control of the motors of the robots, we have specified the corresponding affected subset of variables $G \in 2^V$. We then compare all these subsets with the output of the open-set classification, i.e., the set H, by computing the Jaccard index J [11] between H and each subset G, and selecting the maximum. For these experiments the classification is evaluated with accuracy (since we are using the Jaccard index). However, the system has still been trained only with *normal* data, so test classes were never seen at training time. Table 3 reports the accuracy for each model with respect to each type of attack, whose names match those used in the documentation. From the *Boat* dataset, we can see that RNN correctly identifies each attack with accuracy from 55% (Stuck) to 99% (GpsDown). The TCN model has higher accuracy for some kinds of attacks, but it is not able to identify the

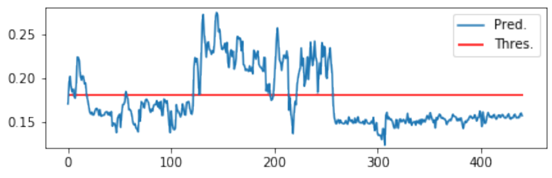

Fig. 2. Pepper prediction behavior under an attack: MSE value of Dense model (blue) vs. 99.7 percentile (red).

GpsDown attack. The dense model instead usually mis-classifies. Overall, the best model remains RNN. We also compared how the different types of attacks are predicted by the one-class classification (Problem Definition 1) and the open-set classification (Problem Definition 2). In particular, we observe that for some types of attacks, one approach is more effective than the other. For example, attack 28 on $SWaT$ is detected very well as an anomaly (one-class), although the responsible sensors identified are wrong (open-set). Conversely, attack 32 is not detected (one-class), but sensors involved in this attack are correctly identified (open-set). This suggests that an approach with higher performance for anomaly detection may not necessarily be the most useful to detect the source of anomaly and hence to suggest viable interventions to address it. As an additional experiment on the open-set classification problem we tried to verify if our methodologies correctly identify the most involved sensors, before applying the classification with the Jaccard index. For each time interval in a specific attack, we have assigned an incremental score to the first 10 sensors involved, from 1 for the tenth to 10 for the first. In 21 types of attack out of 35 in $SWaT$, at least one of the sensors involved is detected. This result gives an idea of the operation at the basis of the classification and can already be used to understand the nature of the attack.

6.3 Pepper use case

In this final experiment, we show a use case for the *Pepper* social robot involved in a public demonstration subject to a simulated attack. In contrast to the other experiments, here we run the regression models trained on the previously mentioned *Pepper*, testing them on an additional log that has been acquired in a very different situation: a public space where several people interact with the robot with diverse goals and modalities. Figure 2 shows the outcome of the system during a portion of time where the *Pepper* robot was manually pushed back by a user. As shown in the figure, the system was able to detect the anomaly happening in the middle of the logged period, i.e., between time 120 to 250. Moreover, running the method for open-set classification we notice that the first sensor that appears in the ranking is the sensor that evaluates the stiffness of the front left wheel [*WheelFLStiff*] followed by some of the laser sensors positioned on the shovel. This confirms that the method is capable of providing useful indications on the possible source for the anomaly.

7 Conclusions

In this work, we have presented several one-class anomaly detection methods that can be trained using logs of robotic platforms and CPS. Moreover, we consider the problem of detecting variables or sensors that are most affected by the anomalous behavior addressing an open-set classification task. Quantitative comparison with state of the art methods on several datasets shows that our approach achieves comparable results for anomaly detection and promising results for open-set classification. Our work paves the way for several research directions. In particular, we believe that addressing the open-set classification problem is a key step to go beyond anomaly detection and move towards system analysis and diagnosis for robotic platforms. We believe that moving in this direction is crucial to widen the practical use of robotic technologies in real-world applications.

Acknowledgment

Sapienza University of Tome - funding for scientific research - year 2020

References

1. D. Li *et al.*, "Anomaly detection with generative adversarial networks for multivariate time series," *CoRR*, vol. abs/1809.04758, 2018.
2. B. Zong *et al.*, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *Int. Conf. on Learning Representations*, 2018.
3. M. Olivato *et al.*, "A comparative analysis on the use of autoencoders for robot security anomaly detection," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.
4. D. Li *et al.*, "MAD-GAN: multivariate anomaly detection for time series data with generative adversarial networks," *CoRR*, vol. abs/1901.04997, 2019.
5. H. Zenati *et al.*, "Efficient gan-based anomaly detection," *CoRR*, vol. abs/1802.06222, 2018.
6. B. Schölkopf *et al.*, "Support vector method for novelty detection," in *Advances in neural information processing systems*, pp. 582–588, 2000.
7. F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, Dec 2008.
8. E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, pp. 1065–1076, 09 1962.
9. M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *Ann. Math. Statist.*, vol. 27, pp. 832–837, 09 1956.
10. J. Goh *et al.*, "A dataset to support research in the design of secure water treatment systems," in *Critical Information Infrastructures Security*, (Cham), pp. 88–99, Springer International Publishing, 2017.
11. P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining.* Pearson Education, 2006.