A Customized Approach to Anomalies Detection by using Autoencoders

Roberto Aureli[a], Nicolo' Brandizzi[a], Giorgio De Magistris[a] and Rafał Brociek[b]

[a]Department of Computer, Control and Management Engineering, Sapienza University of Rome, Via Ariosto 25, 00135, Rome, Italy
[b]Department of Mathematics Applications and Methods for Artificial Intelligence, Faculty of Applied Mathematics, Silesian University of Technology, 44-100 Gliwice, Poland

Abstract
When dealing with sensor's data, it's important to keep track of what it's really happening in the tracked environments since failures, interruptions and misreadings must be expected at any time. Especially with logging processes involving extremely voluminous reports, an automatic method to detect entries that are not following the normal distribution of data (i.e. anomalies) should be the ideal solution. In the presented work the task performed by the autoencoder is to generate a reproduction error, used as metric for the classification of a sample in one of two classes: anomalous or non-anomalous.

Keywords
Parallel DSP, TI-ADC, FPGA, ASIC

1. Introduction

When dealing with sensor's data, it's important to keep track of what it's really happening in the tracked environments since failures, interruptions and misreadings must be expected at any time. Especially with logging processes involving extremely voluminous reports, an automatic method to detect entries that are not following the normal distribution of data (i.e. anomalies) should be the ideal solution.
Neural Networks can be used in this type of task as detectors for the distance of the sample from the natural distribution underlying the dataset.
In particular, *autoencoders* [1] are a type of neural network capable of compressing the input into a reduced, meaningful representation and finally decoding it back, reproducing it with the minimum error possible [2, 3]. This type of networks has currently been used successfully for image denoising [4, 5, 6], NLP's tasks and generic dimensionality reduction [7, 8]. The first use of this type of network dates back to the 80s, however its origins and authors are unclear, caused by changes in nomenclatures and definitions.
In this work, the reproduction error (i.e. the error between the input sample and the output of the autoencoder) over a set of sample is exploited to discriminate which samples are anomalous in the given set and which not.
Ideally, the anomalies are a minimal part of a dataset with generally low probabilities to be drawn from the distribution describing the set: this scarcity implies a big reproduction error from the autoencoder. Moreover, the more a gradient descent is performed over a set of inputs, the more the loss should decrease (until it hits its minimum), vice versa, if a datum is not common, the loss is greater with respect to other well known data.
This method is shown to work over a real life, unlabelled dataset, posing the problem in the unsupervised learning landscape.

2. Related Works

Anomalies detection tasks have been already studied and solved with neural networks exploiting the *reproduction error*: the difference between a generic sample and a reconstruction of itself performed by some mathematical model.
In [9], a module made up by stacked LSTMs networks is trained over non-anomalous data and its prediction error over the future steps is used as an indicator for the anomaly of the sample. However, this approach needs the dataset to be labelled, increasing the work needed in the creation and the difficulty of application in real life scenarios.
Similarly, in [10], they proposed a novel architecture called ALAD (Adversarially Learned Anomaly Detection), an approach based on generative adversarial networks. The GAN generates an *adversarially learned* set of features used to project the high-dimensional original space of the dataset into a reduced one. The reduced representations are then decoded and the reproduction error is used as an anomaly indicator.

SYSTEM 2021 @ Scholar's Yearly Symposium of Technology, Engineering and Mathematics. July 27–29, 2021, Catania, IT
aureli.1757131@studenti.uniroma1.it (R. Aureli);
brandizzi@diag.uniroma1.it (N. Brandizzi);
demagistris@diag.uniroma1.it (G. D. Magistris);
rafal.brociek@polsl.pl (R. Brociek)
0000-0002-3076-4509 (G. D. Magistris); 0000-0002-7255-6951 (R. Brociek)

More similar to the approach proposed in this work but extremely more advanced, [11] makes use of robust techniques paired with autoencoders to detect anomalies and type of anomalies like random corruptions, recurrent corruptions (i.e. corruptions present in more that one instance) and so on.
In the presented work the task performed by the autoencoder is to generate a reproduction error, used as metric for the classification of a sample in one of two classes: anomalous or non-anomalous.

3. Background and method description

3.1. Autoencoder

An autoencoder is made up by two functions: an *encoder* and a *decoder*. The goal of the encoding function is to map (i.e. to encode) the input into a different space. Symmetrically, the decoder must map the encoded vector back to the original input, without losing information. Conveniently, the encoding space is often chosen smaller in dimensions than the original space, making the autoencoder performing also a dimensionality reduction.
More formally, given an encoding function $E(x)$ and a decoding function $D(x)$, $D(E(x))$ must return back to the original x.
The two mentioned above functions can be approximated by symmetric neural networks, solving the following optimization problem:

$$\min_{\theta_D, \theta_E} \|x - D(E(x))\|^2$$

where θ_D and θ_E are the parameters of the respective neural networks.

3.2. Architecture

The only constraint that need to be taken in consideration is the presence of a bottleneck, a layer smaller than all the other layers in the network, essential for the dimensionality reduction. Without a bottleneck, the network is not "forced" to ignore useless or non-representative features in the input, losing the capacity of mapping the input in a denser space.
The networks' architectures are completely adaptable to the problem. Normally, the input (e.g. time-series, images) is embedded into a vectorial representation and then reduced till the bottleneck. This representation is the projection of the input into a different space carried out by the *encoder* part. The latent vector (i.e. the output of the bottleneck) is then passed to a generally symmetric network, returning a representation belonging to the original input space.

3.3. Training

The training procedure is the standard training procedure used for generic neural networks. A sample x is drawn from the dataset, sampled from the distribution $p_{dataset}(X)$. The latent vector (i.e. the output of the bottleneck layer) h is generated by the *encoder* part of the autoencoder, generating $h = E(x)$. This representation, at the end of the training, can be used as an approximated reduced representation of the original input. Finally, h is used as input for the *decoder* D, generating $x' = D(h)$, belonging to the same space of x. In this case, the L1 error between x and x' is minimized to train the parameters of the network. It's crucial that the goal of the training is to guarantee the minimum difference between x and x'.

3.4. Anomalies Detection

The key concept of the method is in the notion of *scarcity*: an anomaly, to be defined as such, needs to be a noteworthy event. In practical cases, it could be an unpredictable spike in a time series, a set of burned pixels in a photo, the saturation of a sensor, an unexpected down time of a link and so on.
It's worth nothing to say that, when sampling an element from a distribution, it's far more probable to sample a normal entry instead of an anomalous case (if not, other methods must be used or some problems could be in the dataset). With these premises, the autoencoder capacity to capture a distribution and projecting it in another space is exploited: during the training phase, multiple epochs are performed over all the samples in dataset, meaning that the network will experience a gradient descent over the loss generated by the same samples multiple times but without modifying the ratio between non-anomalous samples and anomalous ones. Moreover, the reproduction error will be minimized over the most prominent distribution in the dataset, generating larger errors in the anomalous subsets.
As shown in **Figure 1**, the scarcity is the fundamental parameter that separates a normal sample from an anomalous one. When inverting the contamination proportion, the originally *good samples* results in a loss distributed higher than the respective *anomalous* one. This example is purely demonstrative of the analysis made on the loss: a real-life dataset has a percentage of anomalies sensibly lower than the ones used here.
The last thing needed to define an anomaly is the *loss threshold*: after analyzing the distribution of the autoencoder's reproduction loss on all the samples in the dataset, a threshold must be manually imposed, where a sample with a loss beyond the latter is considered anomalous. A compromise must be reached, since all the anomalous samples must be included without including non-anomalous samples.

3.5. Workflow

The method is fast and with big enough datasets a single training can be performed to perform analysis on future, unseen events.
The preprocessing of the dataset is unavoidable: normalizing data and orders of magnitude is necessarily to have a good, consistent loss analysis. Samples bigger in module can importantly alter the distribution.
Finally, the model is fine-tuned (i.e. autoencoder architecture, bottleneck dimensions and so on) and a standard training procedure must be performed. The loss distribution is analyzed and the threshold is manually selected by the user.
Once new data arrive, there is no need to retrain the autoencoder: the sample is normalized and passed to the model, finally it's classified according to the threshold.
It may happen that long-term changes in the dataset distribution could completely alter the outcome of the process, requiring a new training and threshold selection.

4. Experiment

4.1. Dataset

The dataset is composed by information gathered from real car sensors.
Each car is assigned to a *client* that can have multiple *fleets* and, every day that the car has been used, the *total time* of connection to the network is logged, saving the duration in seconds, the *start date* and the *end date* of the session. In case of disconnections during the same day, multiple entries will show the multiple sessions.

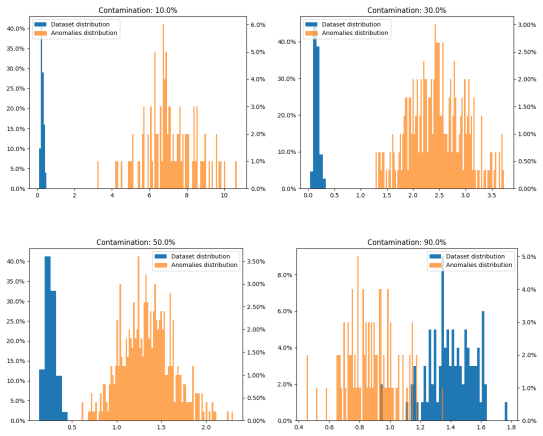

Figure 1: The losses distribution of an autoencoder trained on an artificially contaminated labelled dataset composed by normalized vectors of dimension 24 sampled by a multivariate Gaussian for the non-anomalous data and from an uniform distribution for the anomalous data.

- **VID**: An integer representing a single car in an unique way.
- **First Ping**: A date in the format `yyyy-dd-mm` representing the day of the session
- **TotalTime**: Seconds (integer) elapsed from the first connection to the last disconnection, same for all the sessions of the day.
- **start**: timestamp representing the start of the session.
- **end**: timestamp representing the end of the session.
- **env**: An integer representing the client owning the car in an unique way.
- **Service**: An integer representing the fleet of the client, unique for the given client.

4.2. Preprocessing

The structure of the dataset could resemble a time-series in which, eventually, each day is made up by more entries for the same car. To tackle this complexity, a reduction to a single entry for each car in a given day is performed in such a way to get a single point for each pair of `(VID, First Ping)` (`First Ping` is then dropped at the end of the preprocessing phase). However, to not lose any important information, some fields have been added to record what is implicit in the original dataset while `VID`, `TotalTime`, `env` and `Service` are kept the same. Since `start` and `end` are removed, `TotalTime_OFF` has been added to record the total time in which the car has been disconnected from the network. Moreover, `n_disconnections` is added to remember the number of sessions in a single day for the given car. The final attribute added is `C_v_off_time`, namely the *coefficient of variation* of the down-times between each session, an index of dispersion.
The coefficient is defined as:

$$C_v(x) = \frac{\sigma(x)}{\mu(x)} \tag{1}$$

where x is a set of data, σ is the standard deviation of x and μ is the mean. Practically, an higher C_v means that the data are unbalanced, implying a big difference across each element in the set. For example, $C_v(10, 10, 10, 10, 10) = 0$ meaning no dispersion in the data, while $C_v(35, 5, 5, 2, 3) = 1.4$, evidencing a set of data more scattered around the mean.
The meaning given to the coefficient of variation can vary based on the needs: in some cases it's better to have short disconnections than a long one since it could be easier to deduct missing locations or missing data, preferring therefore lower C_vs. Contrary to the last sentence,

a C_v near 0 could also represent a set of long disconnection times followed by small ones: this value alone is not enough to get an idea of how a link is performing since it doesn't contain information regarding the quantities in the set, only a normalized index of dispersion.
After the preprocessing, the dataset has been reduced from 573064 entries to 99386. As for the training process, the autoencoder is trained over `TotalTime`, `TotalTime_C_v_off_time` and `n_disconnections` normalized between 0 and 1. This choice is justified by the fact that the remaining variables are categorical one hence completely arbitrarily values used only for an identification purpose.

4.3. Model architecture and Training

The simplicity of the reduced dataset permits the use of a fully connected network. An hidden layer (for each network) is enough to capture the dimensionality, ending in a bidimensional bottleneck. The nonlinearity is introduced by a *ReLU* function and a *Sigmoid* [12] at the output of the two networks. To ease the training process and evade possible loacal-minima situations, a dropout layer is used in order to randomly drop to 0 the weights of the network with a probability of 25%. The associated loss is an L1-distance defined as $d(x, y) = \sum_{i=1}^{m} |x_i - y_i|$, where x and y are vectors of length m.
The training process is supervisioned by an early stopping mechanism, keeping the best model before reaching a situation of overfit. The performances are tracked by computing the loss over a set of unseen samples, extracted with a proportion of 30% from the original dataset.

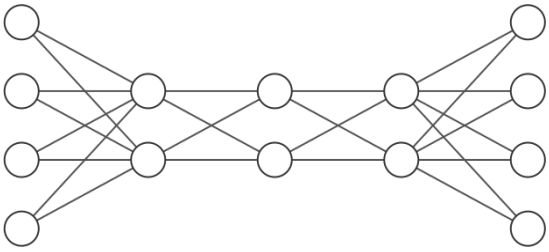

Figure 2: Autoencoder's layers and dimensions

4.4. Training results

As mentioned before, after the training the loss distribution must be analyzed. The histogram in **Figure 4** shows a decreasing trend before 0.10 after a plateau that drops at 0.25, followed by some sparse samples. The quality of the training is confirmed by the peak of the loss distribution near 0, evidencing an overall low loss.
By an empirical choice, the threshold is defined at 0.10,

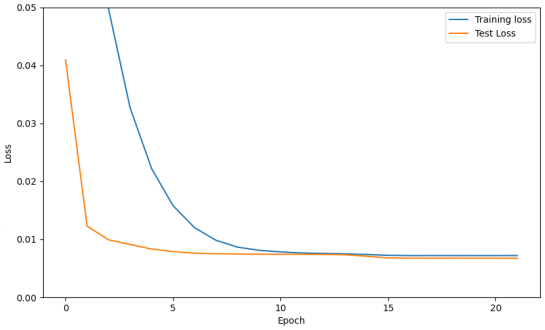

Figure 3: Training profile of the autoencoder's losses: there isn't a significant overfit in the last part.

where 1803 anomalies are found, the 1.84% of the whole dataset.

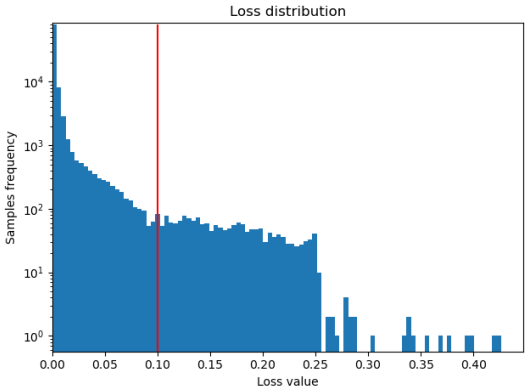

Figure 4: Histogram showing the distribution of the loss over all the dataset. Logarithmic scale on the y-axis.

5. Results

The following results are extracted from the dataset after the classification process mentioned before: each sample is expanded with the associated previously removed categorical variables, in order to contextualize the results in the dataset domain.
Each analysis starts from the comparison of the distribution in each class: anomalous and non-anomalous data. In each plot, the blue distribution represents the whole dataset distribution (labels on the left y-axis) while the orange one shows the anomalies (labels on the right y-axis).

5.1. Time anomalies

As expected, the notion of anomaly in a *connection to network* domain is in direct correlation with the duration of the down-time.

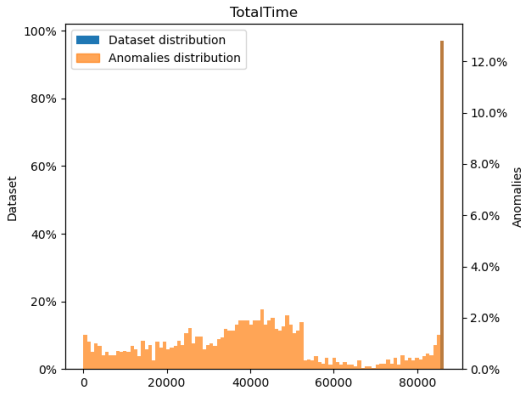

Figure 5: Distribution of the Total Time,

Figure 5 shows a blue peak of the distribution over the maximum value admissible (i.e 86400 seconds in a day), followed by a peak in the same position of the anomalies. The interesting part of the plot is where the anomalies are distributed more than the dataset itself, hence in the lower values of the x-axis. As expected, an important number of entries with a low total connection time is classified as anomaly.
Symmetrically, the plot of the down-time represents a similar situation:

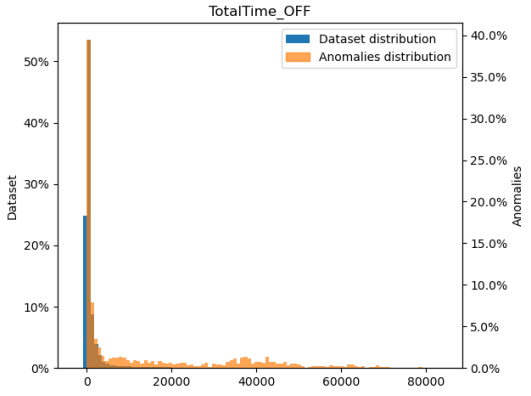

Figure 6: Distribution of the Total Time of disconnection

Also in this case, there is an important anomalies' peak near the dataset distribution followed by many samples on the right part of the x-axis. An anomalous entry is also described by an high disconnection time.

To better see the motivation of the overlapping peaks, a new type of plot can be introduced:

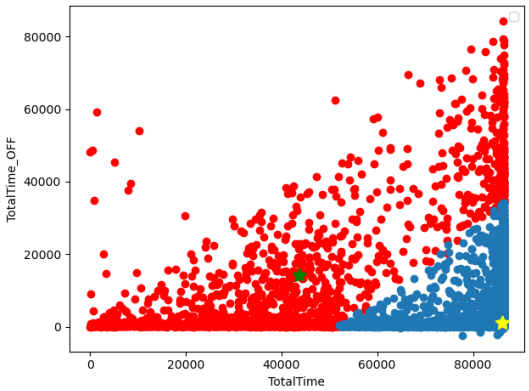

Figure 7: Scattered representation of each entry with `TotalTime` and `TotalTime_OFF` as coordinates for each point. Non-anomalous points in blue, anomalous points in red.

In this plot each entry is scattered on a plane, representing `TotalTime` on the x-axis and `TotalTime_OFF` on the y-axis. An ideal entry has the maximum `TotalTime` and the minimum `TotalTime_OFF`, posing itself on the rightmost lower corner of the plot.
The yellow star represents the center of mass of the non-anomalous distribution (in blue), very near to the ideal point, while the green star represents the center of mass of the anomalies distribution. There are $\sim 80k$ blue points while only $\sim 2k$ red ones, evidencing a big difference in the concentration.
It's worth nothing to say that a different threshold would have moved the frontier of the two clusters up or down. The reason of the overlapping peaks lies in the fact that a big number of anomalies is in the vertical line over the maximum of `TotalTime` (i.e. $x = 86400$) and on the horizontal line over the minimum `TotalTime_OFF` (i.e. $y = 0$), meaning that one variable is in a good range while the other one not. The worst anomalies are the ones lying near the center of the plot, containing a discrepancy in both the variables.

5.2. Number of disconnections

A counterintuitive results is shown in the next plots

In the left plot there aren't any notable results, the two distributions appear the same. The number of disconnections is evenly distributed over each entry in the dataset and in the anomalies.
When computing the distribution over the average disconnection time for each car, the histograms show an important difference: the peak of the anomalies is lower

than the peak of the dataset, meaning that the anomalous entries have a lower number of disconnections. However this could cause some confusion since a larger number is expected when talking about this type of variable but analyzing the results paired with the ones obtained in the previous subsection, the lower number of disconnections reveal a longer down-time. This result is confirmed by the following plot where the anomalies distribution is slightly translated to the right, meaning a less uniform number of disconnections that can be caused by a longer disconnection time followed by a set of short times.

5.3. Categorical analysis

A final analysis can be made on the categorical variables, answering the practical question: "Are there bad cars or bad clients?".
Following the previous analysis, it's possible to retrieve an overview on the presence of a single car in the anomalies by plotting the distribution of the VIDs. In **Figure 11 (left)**, the anomalies are concentrated in the first part of the x-axis, showing a peak near 0. A set of car that appear only less than 10.0% in the dataset is responsible of the $\sim$ 30.0% of the anomalies. A zoomed version of the same plot can be seen on **Figure 11 (right)**, where only the cars with a VID less than 2000 are showed.
In a practical way, this result can help focusing more on the subset of car that is more present in the anomalies, helping saving time on the analysis.
From this result it's possible to derive the conclusion on the final result, the client analysis. In **Figure 10** the biggest percentage of anomalies is covered by the client number 1. The duality on the results can show that the most of the first 2000 cars are assigned to the first client, notion that may help with the understanding of the failures.

5.4. Reproducibility

The method explained here is implemented in *PyTorch* [13].

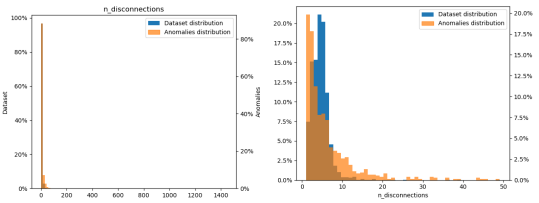

Figure 8: Plot of the distributions of the number of disconnections (left), plot of the average number of disconnections for each car (right)

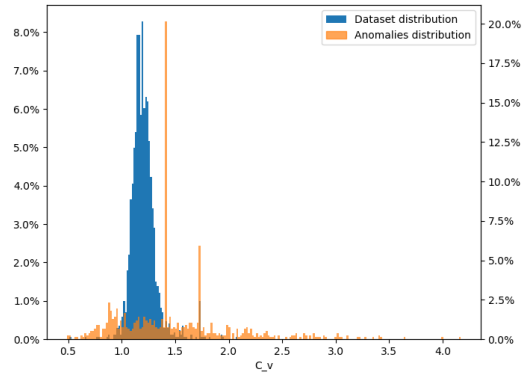

Figure 9: Plot of the distribution of the average coefficient of variation for each car.

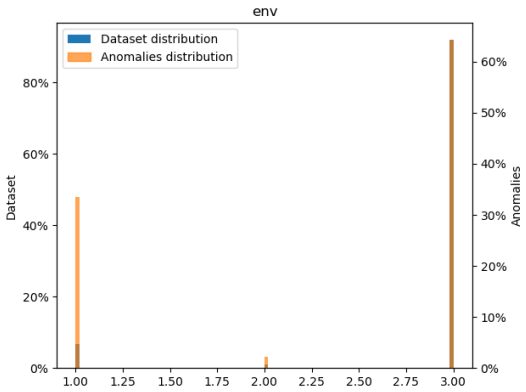

Figure 10: Plot of the distribution of the clients id.

All the results mentioned are perfectly reproducible by utilizing the same saved model (i.e. same architecture with same weights loaded in) and the same threshold. The only stochastic variable in the model is the dropout layer that must be deactivated before the evaluation. An useful thing is that the threshold is not an hyperparameter of

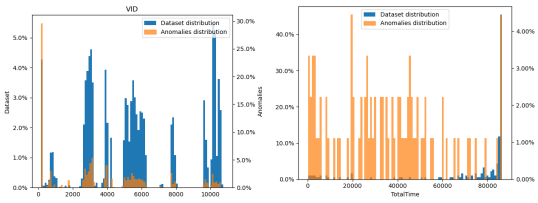

Figure 11: Plot of the distributions of the VIDs(left), plot of the distributions of the VIDs showed on the first 2000 vehicles (right).

the network, meaning that it can be changed, according to the needs of the user, after the training phase. A variation on the anomalies threshold could completely alter the result by including more or less entries in the anomalous set, creating a more severe (or less) detection system.
Finally, every evaluation can be made in real-time (after the training), with times that can vary according to the hardware and the architecture used. On an NVIDIA MX150, the evaluation over all the dataset takes approximately 20 seconds.

It's possible that the model needs a retraining if the distributions in the dataset changes in an unexpected way (e.g. logging temperatures can require a retraining between summer and winter if the entries are not enough for each season).

6. Conclusions

The method presented in this paper has shown the ability to perform a classification of an unlabelled dataset. It's a fast way to identify outliers in datasets or in a real-time data feed (after a first training over a big enough dataset composed by recorded logs).
Its flexibility can be a great incentive to its utilization, being tied only to the use of an autoencoder, architecture that can be expanded and customized at each eventuality. Moreover, the fact that the threshold must be imposed after the training can be exploited to increase or decrease the severity of the system in real-time, following changes in the needs of the user.
However, the manual choice of a loss threshold could be an element of imprecision, looking at the fact that a slightly alteration could extremely change the samples considered as anomalous. Another downside could be the needing of a big enough dataset, since with a little one there could be difficulties in learning the right distribution. However, this is a common problem among all the autoencoder applications.
Finally, the system is sensible to the dataset's dimensions, requiring an adequate normalization that must be applied also to real-time samples.
In conclusion, as future work the method can be expanded to automatically detect the threshold, removing the manual component that could completely change the outcome of the process.

References

[1] D. Bank, N. Koenigstein, R. Giryes, Autoencoders, CoRR abs/2003.05991 (2020). URL: https://arxiv.org/abs/2003.05991. arXiv:2003.05991.

[2] B. Nowak, R. Nowicki, M. Woźniak, C. Napoli, Multi-class nearest neighbour classifier for incomplete data handling, volume 9119, 2015, pp. 469–480.

[3] S. Russo, S. Illari, R. Avanzato, C. Napoli, Reducing the psychological burden of isolated oncological patients by means of decision trees, volume 2768, 2020, pp. 46–53.

[4] L. Gondara, Medical image denoising using convolutional denoising autoencoders, in: 2016 IEEE 16th international conference on data mining workshops (ICDMW), IEEE, 2016, pp. 241–246.

[5] G. Capizzi, G. Lo Sciuto, C. Napoli, E. Tramontana, M. Woźniak, A novel neural networks-based texture image processing algorithm for orange defects classification, International Journal of Computer Science and Applications 13 (2016) 45–60.

[6] R. Avanzato, F. Beritelli, M. Russo, S. Russo, M. Vaccaro, Yolov3-based mask and face recognition algorithm for individual protection applications, volume 2768, 2020, pp. 41–45.

[7] Y. Wang, H. Yao, S. Zhao, Auto-encoder based dimensionality reduction, Neurocomputing 184 (2016) 232–242.

[8] G. Capizzi, S. Coco, G. Sciuto, C. Napoli, A new iterative fir filter design approach using a gaussian approximation, IEEE Signal Processing Letters 25 (2018) 1615–1619.

[9] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long short term memory networks for anomaly detection in time series, 2015.

[10] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, V. Chandrasekhar, Adversarially learned anomaly detection, in: 2018 IEEE International Conference on Data Mining (ICDM), 2018, pp. 727–736. doi:10.1109/ICDM.2018.00088.

[11] C. Zhou, R. C. Paffenroth, Anomaly detection with robust deep autoencoders, in: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, pp. 665–674.

[12] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, Activation functions: Comparison of trends in practice and research for deep learning, 2018. arXiv:1811.03378.

[13] A. e. a. Paszke, Pytorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.