DeepAD: A Generic Framework based on Deep Learning for Time Series Anomaly Detection

Teodora Sandra Buda[1], Bora Caglayan[1], and Haytham Assem[1] *

[1] Cognitive Computing Group, Innovation Exchange,IBM Ireland
{tbuda@ie., bora.caglayan@, haythama@ie.} ibm.com

Abstract. This paper presents a generic anomaly detection approach for time-series data. Existing anomaly detection approaches have several drawbacks such as a large number of false positives, parameters tuning difficulties, the need for a labeled dataset for training, use-case restrictions, or difficulty of use. We propose DeepAD, an anomaly detection framework that leverages a plethora of time-series forecasting models in order to detect anomalies more accurately, irrespective of the underlying complex patterns to be learnt. Our solution does not rely on the labels of the anomalous class for training the model, nor for optimizing the threshold based on highest detection given the labels in the training data. We compare our framework against EGADS framework on real and synthetic data with varying time-series characteristics. Results show significant improvements on average of 25% and up to 40-50% in F_1-*score*, precision, and recall on the Yahoo Webscope Benchmark.

1 Introduction

A well-known characterization of an outlier is given by Hawkins as, "an observation which deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism" [10]. An anomaly represents a non-conforming pattern that deviates from the expected behavior, and is often referred to as an outlier or exception [5]. Detecting and mitigating these anomalies is fundamental in various domains (e.g., health, performance, security), and translates to potentially saving lives by detecting critical conditions, revenue and reputation by avoiding downtime, or improvements in application performance.

A popular approach for anomaly detection is employing explicit generalization models [1], where a summarized model is created up front to capture the normal behavior of the monitored instance, and further using the deviation between the expected normal behavior and actual behavior as error metric for anomaly detection. Typically the deviation is then monitored and fitted to a particular distribution (e.g., Gaussian [13]) and then a threshold is identified based on optimizing the precision and recall in the training data through the

* This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 700381 (ASGARD) and No. 671625 (CogNet).

use of past labelled anomalous instances. The use of the labels of the anomalous class, also referred to as *golden labels* is a requirement for most of the anomaly detection techniques, either for identifying a threshold or for building a classifier to detect anomalies based on anomalous patterns in the past. This however limits the applicability of these techniques to datasets where these labels have been collected and in addition, many times suffering from the class imbalance problem, since the normal instances typically overweight the abnormal ones. Moreover, besides the need for golden labels, existing anomaly detection approaches are typically suitable for a particular type of data or anomaly to capture, which makes their application more limited in practice [1, 5].

This paper introduces a novel *Deep* learning-based *A*nomaly *D*etection framework, named DeepAD. The DeepAD framework discovers anomalies without the need of golden labels, while maintaining the highest levels of true anomaly detection, and reducing the number of false positives compared to the best available technique. DeepAD employs various explicit generalization models to learn the normal behaviour of the data and utilizes a dynamic sliding window for determining a dynamic threshold fitted for each time-series under analysis. The dynamic window is adjusted for each point to contain past rescaled squared errors to ensure the accuracy is highest. To the best of our knowledge, DeepAD represents the first framework of its kind that utilizes multiple advanced prediction models allowing multivariate inputs without the specific use of golden labels. The use of multiple models, combined with the dynamic threshold on rescaled errors increases F_1-*score*, precision and recall beyond the state of art. The key characteristics of DeepAD are identified below:

1. This framework leverages state-of-the-art *deep learning* models such as long short term memory (LSTM) neural networks, which are renown for their ability to remember relevant information in temporal sequence data even with large gaps in between using memory gates.
2. The model learns the normal behaviour of the monitored instance and deviations from this normal behaviour are signalled as anomalous data points. The framework does not use the *ground truth* of actual anomaly locations neither for training the model nor for determining the dynamic thresholds.
3. The framework does not set hard thresholds which makes it more adaptable to varying patterns in the dataset considering an *online* setting.
4. DeepAD supports *multivariate* analysis since it can receive as input more than one feature if needed, e.g., through LSTM, and hence can surpass the first limitation of approaches limited to univariate analysis.
5. The framework combines the predictions of multiple forecasting techniques, including autoregressive models and triple exponential smoothing, in order to offer a *generic* extensible approach for forecasting.

2 Related Work

Advanced anomaly detection techniques usually employ machine learning, which can be divided into three classes: supervised, semi-supervised and unsupervised.

Anomaly detection with supervised learning [9] requires a dataset where each instance is labelled and typically it involves training a classifier on training set. Semi-supervised algorithms such as [14] construct a model to represent the normal behaviour from an input training dataset; following the model is used to calculate the likelihood of the testing dataset to be generated by the model. Unsupervised models such as [3] do not require a labelled dataset and operate under the assumption that the majority of the data points are normal (e.g., employing clustering techniques [15]) and return the remaining ones as outliers.

LSTMs have captured the attention of researchers recently in anomaly detection. For instance, [13] utilize LSTM for predicting time series and use the prediction errors for anomaly detection. They assumed that the resulting prediction errors have a Gaussian distribution, which were used then to assess the likelihood of anomalous behavior. Then a threshold is learnt based on the validation dataset to maximize the F-score, which was calculated based on the golden labels within the validation dataset. The approach was validated on four time series. Moreover, [6] follows a similar approach applied to ECG time series, where the prediction errors are fit to a Gaussian distribution, and then the threshold is determined based on optimizing the F-score on the validation set, which similarly was calculated based on the given golden labels. Furthermore, [12] utilizes an LSTM-based encoder-decoder for multi-sensor anomaly detection. When enough anomalous sequences are available, a threshold is learnt by maximizing precision and recall. The use of recurrent neural networks is also common for intrusion detection, such as in [2], with the aim of detecting and classifying attacks. However, the approaches identified above utilize the golden labels for optimizing the threshold against the prediction errors or building classifiers.

Two major limitations exist in current techniques: 1) Most approaches, such as statistical and probabilistic models, are typically suitable only for univariate datasets where a single metric is monitored at a time. This can be extended to multiple metrics by building a model for each metric. However, this would not consider any correlations between metrics. Hence these approaches cannot easily be extended to multivariate analysis where correlations among metrics can be used to identify potential anomalous behaviour. This is avoided as DeepAD can receive as input multiple features, since it can use a single LSTM model that can capture anomalies across multiple features, which makes it multivariate. 2) Existing approaches typically rely on datasets that contain the ground truth labels, where the anomalies are specifically pin pointed to a data point. This can be difficult to gather in real-life scenarios as labelled data is expensive and requires expert knowledge which yet might be affected by human errors in labelling the data. Moreover, the amount of data to be monitored and labelled would be unrealistic. In addition, the initial model might not generalize to new types of anomalies unless retrained and hence requiring expert knowledge for the entire duration of the deployment of the anomaly detection model, making these approaches unrealistic to be deployed in dynamic environments. This is avoided with our dynamic threshold-based anomaly detection approach since no labels are required for training or detecting the thresholds.

3 DeepAD Framework

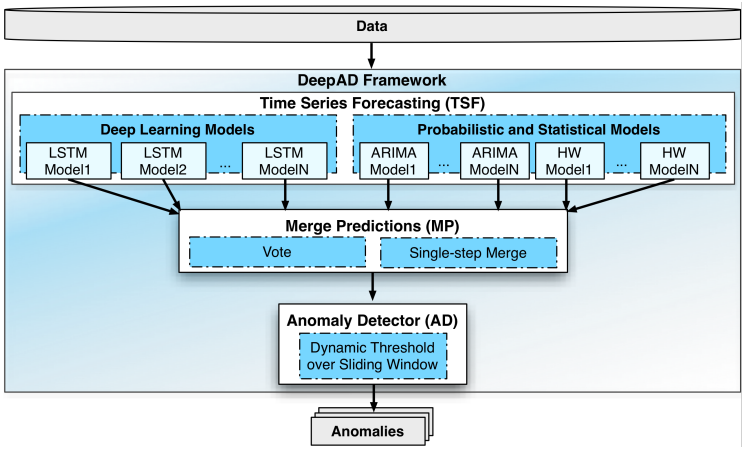
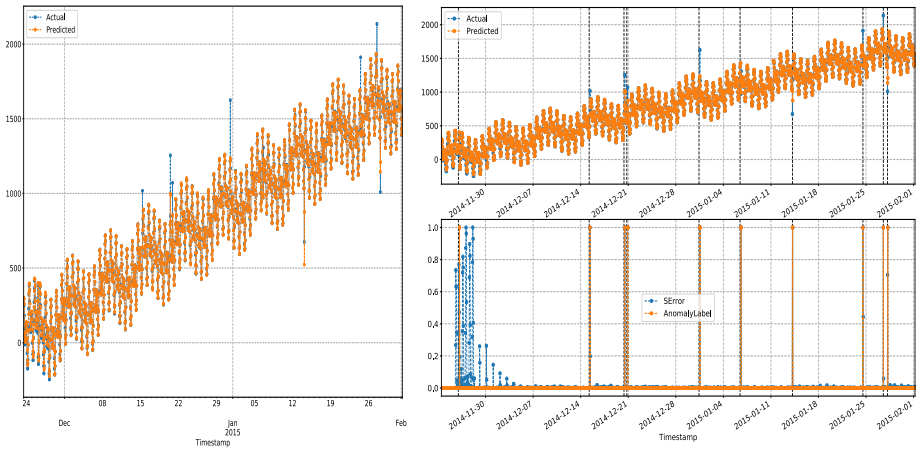


Fig. 1: DeepAD Framework Overview.

The DeepAD framework is illustrated in Figure 1 and has three main phases, detailed in the following subsections:

1. **Time series forecasting (TSF)**: The first phase employs various different explicit generalization models. We train the probabilistic and statistical models and the LSTM models utilizing different architectures for learning the normal behaviour of the monitored environment and then apply them on incoming streaming data for scoring. Through this approach, our framework supports plugging in different TSF models and can leverage *multivariate* models for forecasting.
2. **Merge Predictions (MP)**: The second phase combines the predictions of the multiple models, since some techniques provide better results than others depending on the dataset characteristics. This phase is crucial as it enables DeepAD to be a *generic* framework in the sense that it does not depend on a specific time series forecasting model.
3. **Anomaly Detector (AD)**: The third phase employs extreme value analysis for computing a dynamic threshold, as follows: it compares the actual values and the predicted values and when the distance is above a certain threshold the framework reports the current value as anomalous. The distance represents the squared error between the actual and predicted value, normalized between 0 and 1, and the threshold is computed at each time step on the past scaled squared error. Through this approach, our framework is *independent* of the golden labels and hence can be applied to any time series data irrespective of them containing anomalous labels in the past.

3.1 Time series forecasting (TSF)



(a) TSF+Single-step Merge. (b) TSF+Single-step Merge+AD.

Fig. 2: DeepAD$_{Merge}$: Time Series Forecasting with Single-step Merge and AD output on a sample time series (#90) from A3 Benchmark.

Given a dataset D, the TSF phase aims to learn the normal behavior of the system under analysis. The output of each TSF model is a one-step ahead prediction which will contain what the value is expected to be at the next timestamp. For this purpose, DeepAD supports plugging in different models to enable the prediction. Currently, DeepAD utilizes the following techniques: Long-short term memory (LSTM), autoregressive integrated moving average (ARIMA) and triple exponential smoothing, also commonly referred to in the literature as Holt-Winters (HW), as the models can complement each other depending on the dataset. For instance, deep neural networks such as LSTM may provide best results given large training data, whereas given small datasets, ARIMA and HW may provide better forecasts.

In the case of LSTM, the *look_back* parameter needs to be specified, which represents the number of previous time steps to use as input values to predict the next time step value. DeepAD utilizes the following LSTM architectures: (i) LSTM simple: 1 hidden layer with n neurons. The following three variations of this architecture were plugged into DeepAD: $n = \{4, 10, 16\}$, (ii) LSTM wide: 3 hidden layers with 64, 256, and 100 neurons, respectively, and (iii) LSTM deep: 7 hidden layers with 16, 48, 48, 96, 96, 48, and 16 neurons, respectively. The objective is to use simple, wide and deep architectures. For the each architecture we have trained two models, one with a *look_back* of 1 and another with a

look_back of 3, respectively, and for all we have used *rmsprop*[1] as optimizer, since these resulted in the lowest RMSE. We also evaluated the following *look_back* variations: 1, 3, 12, 24, 60.

Furthermore, in the case of ARIMA and HW, DeepAD utilizes the past $24 \cdot 5$ values for forecasting, in case of hourly measurements, which leads to utilizing the past 5 days of data for the next prediction. In particular for ARIMA we utilize the following values for building the different models: $p = \{0,1\}$, $d = \{1\}$, and $q = \{1,2\}$, where p is the number of time lags of the autoregressive model, d is the degree of differencing, and q is the order of the moving-average model. Moreover, for HW we utilize: $\alpha = 1$, $\beta = 0$, $\gamma = 0.7$ and $\alpha = 0.716$, $\beta = 0.029$, $\gamma = 0.993$, since these resulted in the lowest RMSE. For both ARIMA and HW, more models can be plugged in with other parameters values combinations.

We illustrate the outputs of the TSF phase in Figure 2a, where the *Actual* values are highlighted with orange, and the *Predicted* with blue. In this phase, we can observe that the predicted values typically follow the actual values, except for most of the sudden spikes in the data.

3.2 Merging Predictions (MP)

Similarly to an ensemble, the second phase combines the predictions of the multiple models following two distinct approaches:

1. *Single-step merge (DeepAD$_{Merge}$)*: This strategy aims to combine the outputs of multiple models in order to get a more accurate forecast for a single dataset. For this purpose, this strategy compares the predicted values produced by each individual model with the actual value and selects the prediction with the lowest RMSE to forward to the AD phase at each timestamp.
2. *Vote (DeepAD$_{Vote}$)*: This strategy aims to select the use of a single model for a given dataset. For this purpose, this strategy follows a voting approach, keeping only the model that provided the most accurate predictions in terms of RMSE for the training dataset to be utilized further for forecasting.

3.3 Anomaly Detector (AD)

Once the predictions are merged, a dynamic threshold is determined based on the squared error as follows: for each predicted value, a queue representing the sliding window of the previous squared errors is maintained. A scaler is applied to fit and transform the past squared errors from the sliding window between 0 and 1. In order to ensure DeepAD is not bound to the underlying distribution of the errors, we leverage Chebyshev's inequality [7]. In contrast to the 68-95-99 rule, also referred to the empirical rule [8], which applies to normal distributions only, the Chebyshev's inequality guarantees that, for a wide class of probability distributions, no more than a certain fraction of values can be more than a certain distance from the mean. In order to allow our framework to work with

[1] http://ruder.io/optimizing-gradient-descent/index.html#rmsprop

a variety of distributions, we utilize this inequality to determine the threshold. We identify that 99%(i.e., $1 - \frac{1}{10^2}$) of the values must lie within 10 times the standard deviation, and hence to identify the $< 1\%$ that might lie outside, we use 10 times the standard deviation of the errors as dynamic threshold. This confirmed optimum results for detecting anomalies across the 367 time series analysed in Section 4.

Algorithm 1: isAnomaly($actualValue$, $pastValues$, $squaredErrors$, $predictedValue$, $look\_back$, $slidingWindow$)

$scaler \leftarrow$ `MinMaxScaler`$(feature\_range = (0, 1))$);
`//Rescale errors from sliding window for dynamic threshold fitting`;
$scaledSErrors \leftarrow scaler$.`fitTransform`$(squaredErrors[-slidingWindow :])$;
`//Compute dynamic threshold as 10 times standard deviation`;
$dynamic\_thresh \leftarrow 10 \cdot$ `numpy.std`$(scaledErrors)$;
`//Calculate squared error and apply transformation on current error`
$crtSError \leftarrow (actualValue - predictedValue) \wedge 2$;
$crtScaledSError \leftarrow scaler$.`transform`$(crtSError)$;
`//If current error bigger than dynamic threshold signal return` $True$;
if $crtScaledSError \geq dynamic\_thresh$ **then return** $True$;
`// Otherwise add non scaled squared error to the queue`;
$squaredErrors \leftarrow squaredErrors$.`put`$(crtSError)$;
return $False$

Following, if the squared error of the predicted value is higher than 10 times the standard deviation of the previous squared scaled errors then the module signals the instance as anomalous. Hence the squared errors and threshold are dynamic and generally change at every prediction to adapt for the new values and increase accuracy. The module is set to wait for a period of 50 timestamps before calculating the standard deviation in order to make sure the standard deviation calculated has sufficient values to derive it and also that there are not too many false positives reported at the beginning runtime of AD. This wait period is a tuneable parameter, however we observed that waiting for 50 timestamps was sufficient for the considered datasets. The step is described in Algorithm 1. Moreover, we illustrate the output of the AD phase in Figure 2b, where the upper part of the diagram illustrates the TSF outputs (i.e., the actual and predicted values), and the lower part of the diagram illustrates AD outputs, i.e., the squared error (SError) and the anomaly label (AnomalyLabel), which is 1 for detected anomalous data points and 0 for normal points. The dashed vertical lines represent the actual anomalous instances from the ground truth. We observe that the AnomalyLabel produced by DeepAD$_{Merge}$ follows the dashed lines either at the time of the anomaly or slightly after.

4 Evaluation

This section presents the evaluation of our proposed framework DeepAD. We compare our framework to a recently published generic and scalable anomaly detection framework called EGADS [11], since it follows similar steps to DeepAD for detecting anomalies. The framework compares against the Anomaly Detection R library[2] released by Twitter, change point methods, and outlier detectors with static threshold, on the Yahoo Webscope Benchmark, claiming to provide highest accuracy levels, irrespective of the dataset.

In addition, we compare DeepAD$_{Merge}$ and DeepAD$_{Vote}$ against the results of three of the individual TSF models coupled with the AD based on dynamic threshold. In this way, we illustrate the benefits of the MP phase of our framework compared to each individual TSF model. Since ARIMA+AD and HW+AD showed similar results across all evaluation metrics, we only illustrate the results of ARIMA+AD, further denoted by DeepAD$_{ARIMA}$. In addition, we illustrate the results of the simple and deep LSTM architectures, denoted further by DeepAD$_{LSTM-S}$ and DeepAD$_{LSTM-D}$, as each was more suitable for a particular dataset, based on the evaluation metric.

Finally, we ranked the performances of the six compared approaches based on the evaluation metrics. We chose *modified competition ranking* as ranking methodology (also known as "1334" ranking). In this ranking methodology, a model's rank is equal to the lowest rank of the model(s) it has a tie with. The modified competition ranking approach guarantees that: a) The results of the ranking would be deterministic, b) The best model would be ranked 1^{st} and the worst model would be ranked 6^{th} for all of the datasets, thus making it possible to aggregate the results.

4.1 Dataset

We utilized the Yahoo Webscope Benchmark[3] for our evaluation since this benchmark has been widely referenced in the community and consists of a wide set of time-series with tagged anomaly points. The benchmark is suitable for testing the detection accuracy of various anomaly-types including outliers and change-points. The benchmark consists of a total of 367 time series, split into four main benchmarks. The A1 Benchmark is based on the real production traffic to some of the Yahoo properties. The other three benchmarks are based on synthetic time-series. A2 and A3 Benchmarks include outliers, while the A4 Benchmark includes change-point anomalies. The synthetic time-series generated have varying length, magnitude, number of anomalies, anomaly type, anomaly magnitude, noise level, trend and seasonality. The real dataset is comprised of Yahoo Membership Login (YML) data and it tracks the aggregate status of user logins to the Yahoo network. Both the synthetic and real time-series contain 3000 data-points each, which for the YML data represents 3 months worth of data-points.

[2] https://github.com/twitter/AnomalyDetection

[3] Yahoo! Webscope dataset ydata-labeled-time-series-anomalies-v1_0. http://webscope.sandbox.yahoo.com

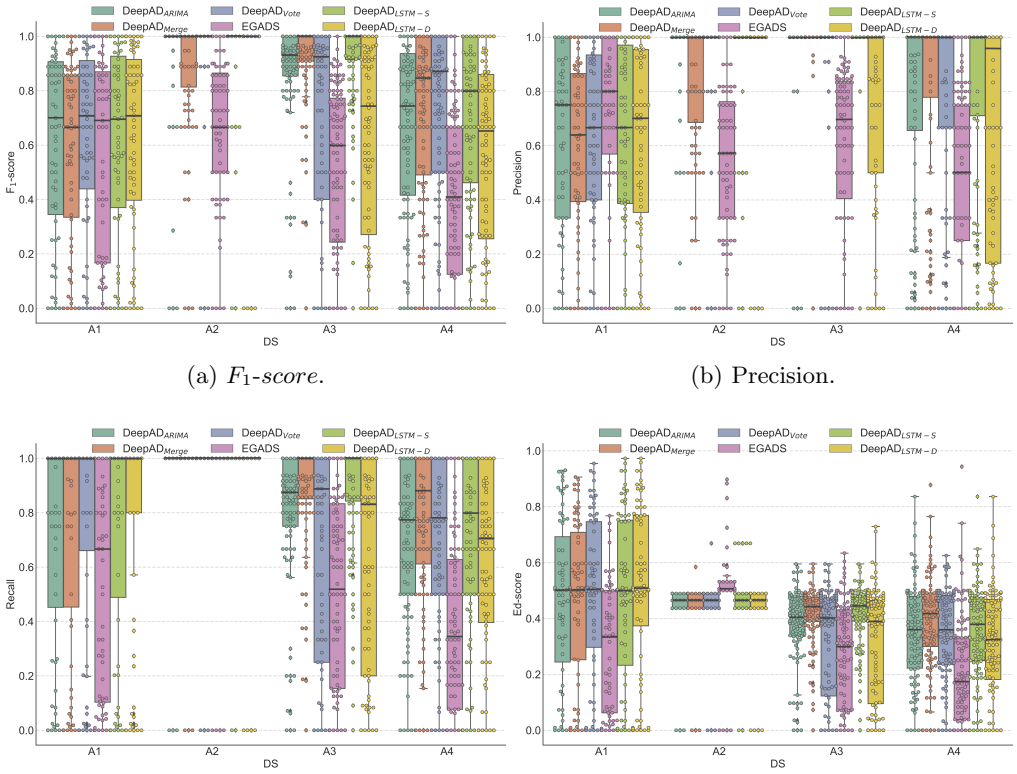
4.2 Evaluation Metrics

We evaluate the techniques based on the standard measures of precision, recall and F_1-*score*. Furthermore, we evaluate the early detection of a technique with the *Ed-score* defined in [4]. The *Ed-score* evaluates how early an anomaly was detected relative to the anomaly window. The *Ed-score* is between 0 and 1, where 1 represents that the anomaly was discovered at the beginning of the interval and 0 at the end of the interval. In this way, the techniques are compared against even if they discover the anomaly after it had occurred (i.e., *Ed-score* less than 0.5). The *Ed-score* is relative to the time interval, i.e., a 10% increase in *Ed-score* means that a technique detected an anomaly 10% of the time interval earlier on average.

4.3 Results

Figure 3a, Figure 3b, and Figure 3c present the DeepAD results compared to EGADS for F_1-*score*, precision and recall, respectively. First, we observe that DeepAD achieves an improvement on average across all datasets as follows by metric: *i)* F_1-*score*: 26%, with a median improvement from 2% in A1 to 40% and 44% in A3 and A4, respectively, *ii)* precision: 25%, with a median improvement from −13% in A1 to 50% in A4, and *iii)* recall: 24%, with a median improvement from 0 in A2 to 53% in A4. Note that only for A1 in the case of precision, EGADS achieves a higher median by 13% compared to DeepAD. This suggests that the framework may be biased towards some datasets than others. However, it can be observed from Figure 3c that the higher median in precision resulted in a less stable and lower median for recall for EGADS in A1. Second, we observe that the performance of some individual TSF models is unstable across different datasets for various evaluation metrics: e.g., for the A1 benchmark consisting of real time series, DeepAD$_{LSTM-D}$ provides better results than DeepAD$_{LSTM-S}$ in terms of recall in Figure 3c, however it provides worse results for the other benchmarks. DeepAD$_{Merge}$ and DeepAD$_{Vote}$ aim to address this commonly found challenge of instability through their ensemble strategy by employing multiple prediction models and results show a more stable performance across datasets and evaluation metrics. Third, depending on the requirements, different MP strategy can be followed: *i)* DeepAD$_{Merge}$ typically maintains a higher level of recall than DeepAD$_{Vote}$ for all datasets due to picking the closest prediction to the actual value at each timestamp, since for the true anomalies typically the TSF predictions are far from the actual value which is expected, and *ii)* DeepAD$_{Vote}$ typically maintains a higher level of precision than DeepAD$_{Merge}$ for all datasets, since it avoids the case of low RMSE TSF models that don't quite learn the underlying patterns but report close to actual values at each time stamp (e.g., a model that learns that the next timestamp has a close value to the current one).

Furthermore, Figure 3d illustrates the early detection score for all techniques. We observe that for the A1 benchmark, the models powered by AD have reached a median of 0.51, compared to 0.34 for EGADS, as the A1 corresponds to the real dataset contain more dynamic realistic patterns. In A2, the performance



(a) F_1-*score*. (b) Precision.

(c) Recall. (d) *Ed-score*.

Fig. 3: Evaluation results in terms of F_1-*score*, precision, recall and early detection score.

of the models was very close, with EGADS reaching an *Ed-score* higher with 0.04 than the rest of the models. However, for A3 and A4 none of the models managed to reach a higher value than 0.5, with a median up to 0.44 in A3 and 0.42 in A4 for DeepAD and 0.3 in A3 and 0.17 in A4 for EGADS, leading to the observation that most anomalies have been detected slightly after their occurrence. We observe that in general DeepAD outperforms EGADS in terms of early detection score across all benchmarks reaching the highest difference of 0.24 in A4.

Figure 4 shows the distribution of ranks for the four performance measures and for all datasets. The figure illustrates the number of datasets for which a model scored a rank between 1 and 6, where rank 1 represents the best model and rank 6 represents the worst model for a given dataset. It should be noted that each model has one or more wins (i.e., rank 1) and one or more lowest rank (i.e., rank 6) for all of the performance measures. This result shows that there

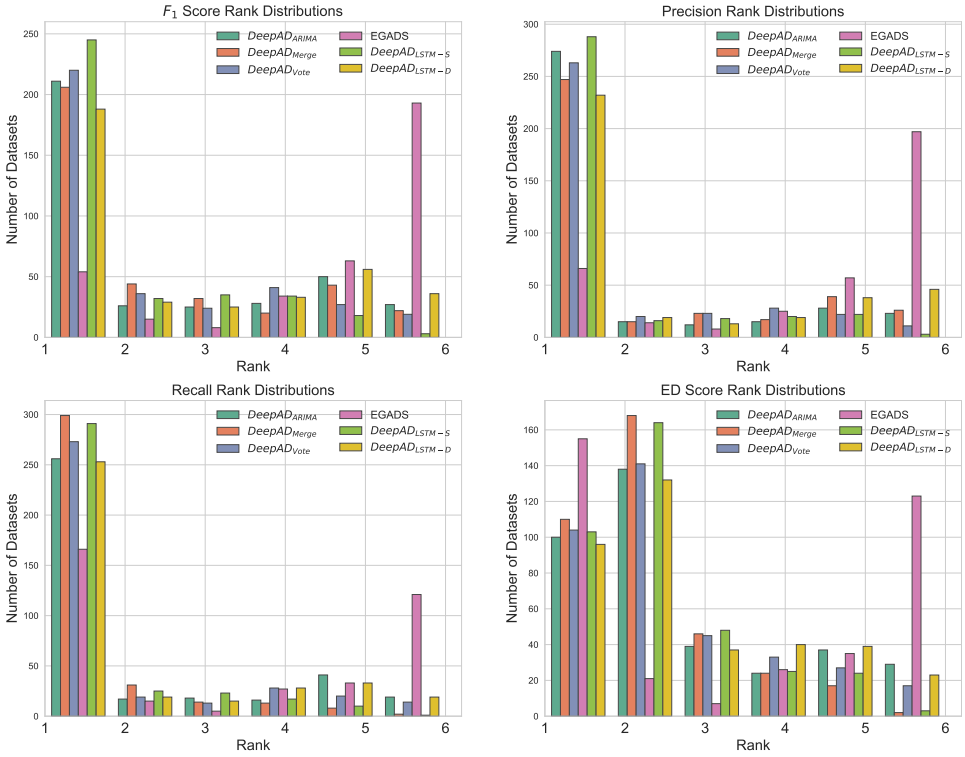


Fig. 4: Modified Competition Ranking of the models for all datasets.

is no model that categorically perform best or worst. However, the distribution illustrates the probability of lower and higher rankings. EGADS had the lowest number of wins for and highest number of lowest ranks among the six models based on F_1-*score*, precision and recall. Surprisingly, for *Ed-score*, EGADS has both the highest number of wins and highest number of lowest rank cases. This suggests once again that EGADS may be biased towards certain datasets. For all the performance measures, EGADS has the lowest median and mean rank overall. EGADS had a mean rank of 4.67 for F_1-*score*, 3.30 for recall, 4.59 for precision and 3.36 for *Ed-score*. EGADS had a median rank of 6 for F_1-*score* and precision, 3 for recall and 4 for *Ed-score*. Lastly, we found that the rank distribution of EGADS is significantly lower than all the other models based on DeepAD using Wilcoxon test ($P < 0.001$). This result shows that on the considered benchmark datasets, picking EGADS would not be the optimal choice. Moreover, the median rankings for all the DeepAD models are 1 for precision, recall and F_1-*score* and 2 for *Ed-score*. The mean ranking difference between the best and worst DeepAD model is less than 1, which shows similar ranking across all DeepAD models.

5 Conclusion

This paper presented a generic anomaly detection framework based on deep-learning (DeepAD) that does not utilize the prior knowledge of the anomalous class neither for training the model nor for determining the threshold. We compared our framework against a state-of-the-art anomaly detection framework EGADS [11] on the Yahoo Webscope Benchmark. We observed that DeepAD generally outperformed and outranked the EGADS framework in terms of early detection score, precision, recall and F_1-*score*. As future work, we plan to plug in other TSF models into the framework, such as convolutional neural networks which can be leveraged in spatiotemporal datasets.

References

1. Aggarwal, C.C.: An introduction to outlier analysis. In: Outlier analysis, pp. 1–40. Springer (2013)
2. Al-Jarrah, O., Arafat, A.: Network intrusion detection system using neural network classification of attack behavior. Journal of Advances in Information Technology Vol 6(1) (2015)
3. Amer, M., Goldstein, M., Abdennadher, S.: Enhancing one-class support vector machines for unsupervised anomaly detection. In: ACM SIGKDD. pp. 8–15 (2013)
4. Buda, T.S., Assem, H., Xu, L.: Ade: An ensemble approach for early anomaly detection. In: IFIP/IEEE IM. pp. 442–448 (2017)
5. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) 41(3), 15 (2009)
6. Chauhan, S., Vig, L.: Anomaly detection in ecg time signals via deep long short-term memory networks. In: IEEE DSAA. pp. 1–7 (2015)
7. Dixon, W.J., Massey Frank, J.: Introduction To Statistical Analsis. McGraw-Hill Book Company, Inc; New York (1950)
8. Dunlop, N.: Statistical Calculations, pp. 203–224. Apress, Berkeley, CA (2015)
9. Görnitz, N., Kloft, M.M., Rieck, K., Brefeld, U.: Toward supervised anomaly detection. Journal of Artificial Intelligence Research (2013)
10. Hawkins, D.M.: Identification of outliers, vol. 11. Springer (1980)
11. Laptev, N., Amizadeh, S., Flint, I.: Generic and scalable framework for automated time-series anomaly detection. In: ACM SIGKDD. pp. 1939–1947 (2015)
12. Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G.: Lstm-based encoder-decoder for multi-sensor anomaly detection. arXiv preprint arXiv:1607.00148 (2016)
13. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series. In: ESANN (2015)
14. Noto, K., Brodley, C., Slonim, D.: Frac: a feature-modeling approach for semi-supervised and unsupervised anomaly detection. Data mining and knowledge discovery 25(1), 109–133 (2012)
15. Rajasegarar, S., Leckie, C., Palaniswami, M.: Hyperspherical cluster based distributed anomaly detection in wireless sensor networks. Journal of Parallel and Distributed Computing 74(1), 1833–1847 (2014)