Presented at IEEE Symposium on Computational Intelligence in Cyber Security (IEEE CICS 2021), Orlando, Florida, USA, December 5-7 2021.
To be published in Proceedings of the 2021 IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2021).

A Performance-Oriented Comparison of Neural Network Approaches for Anomaly-based Intrusion Detection

Stefano Iannucci
Mississippi State University
stefano@cse.msstate.edu

Jesse Ables
Mississippi State University
jha92@msstate.edu

William Anderson
Mississippi State University
wha41@msstate.edu

Bhuvanesh Abburi
Mississippi State University
ba894@msstate.edu

Valeria Cardellini
University of Rome Tor Vergata
cardellini@ing.uniroma2.it

Ioana Banicescu
Mississippi State University
ioana@cse.msstate.edu

***Abstract*—Intrusion Detection Systems employ anomaly detection algorithms to detect malicious or unauthorized activities in real time. Anomaly detection algorithms that exploit artificial neural networks (ANN) have recently gained particular interest. These algorithms are usually evaluated and compared through effectiveness measures, which aim to quantify how well anomalies are identified based on detection capabilities. However, to the best of our knowledge, the performance characterization from the perspective of computational cost and space, training time, memory consumption, together with a quantitative analysis of the trade-offs between algorithm effectiveness and performance, have not been explored yet.**

In this work, we select four recently proposed unsupervised anomaly detection algorithms based on ANN, namely: REPresentations for a random nEarest Neighbor (*REPEN*), DevNet, OmniAnomaly, Multi-Objective Generative Adversarial Active Learning (*MO-GAAL*); we perform a variety of experiments to evaluate the trade-offs between the effectiveness and performance of the selected algorithms using two reference dataset: NSL-KDD and CIC-IDS-2017. Our results confirm the importance of this study, showing that none of the selected algorithms dominate the others in terms of both, effectiveness and performance. Furthermore, it shows that approaches based on Recurrent Neural Networks, which exploit the temporal dependency of the samples, have a clear advantage over the others in terms of effectiveness, while exhibiting the worst execution time.

***Index Terms*—Performance Assessment, Anomaly Detection, Intrusion Detection**

I. Introduction

In the cyber-security domain, Intrusion Detection Systems (IDSs) can be categorized as signature-based and anomaly-based. The signature-based IDSs rely on a database of signatures and perform pattern matching on the input data (e.g., network traffic, log traces, and so forth) to identify potential threats to the protected system. The anomaly-based IDSs instead employ some anomaly detection algorithm to identify deviations from the normal, baseline behavior. In particular, anomaly-based IDSs are recently gaining traction because they have the potential ability to identify zero-days attacks, that is, attacks for which signatures have not been released yet.

Anomaly detection algorithms based on Artificial Neural Networks (ANN) have recently gained particular interest. Indeed, according to [1], in the last decade, ANN-based approaches and their variants (e.g., auto-encoders, self-organizing maps, convolutional neural networks) covered 22.22% of the entire space of anomaly detection algorithms.

Usually, validation and comparison of anomaly detection algorithms are carried out using metrics, such as, *precision, recall, F1-score, AUC-ROC* [2]. However, in the field of intrusion detection, where performing a timely detection is paramount to avoid further negative consequences to the system under attack, metrics such as, *execution time, CPU usage, memory usage, disk usage, scalability*, also become particularly important. We refer to the former set as *effectiveness* metrics, and to the latter as *performance* metrics.

However, most of the works on anomaly detection applied to intrusion detection limit their study to the effectiveness of the proposed approaches, without carrying out a thorough assessment of their performance (e.g. [3], [4]). As a consequence, it is impossible to establish which approach provides the best trade-off between effectiveness and performance.

The main contribution of this paper is the presentation of what is, to the best of our knowledge, the first joint quantitative comparison between the effectiveness and performance of anomaly detection algorithms based on ANN, applied to the field of intrusion detection. Specifically, we compare the *F1-score*, the training time, and the memory usage of four state-of-the-art algorithms, whose source code has been published with an open-source license, using two attack datasets.

We found that none of the assessed approaches outperforms the others in terms of both, effectiveness and performance. Furthermore, there is evidence that approaches based on Recurrent Neural Networks (RNN), have a clear advantage in terms of effectiveness with respect to their counterparts which are not able to exploit the temporal dependence among data samples.

All the changes to the original source code of the selected approaches have been published and are accessible at: https://github.com/cse-msstate.

The studies more closely related to ours include [3]–[5]. Zoppi et al. [3] present a quantitative comparison of several anomaly detection algorithms applied on datasets of multiple attacks. Our work substantially differs from theirs in the set of selected algorithms, since we focus on ANN-based anomaly detection, and also on a performance characterization study. Similarly, Meira et al. [4] present an empirical study of unsupervised learning algorithms but focus only on their effectiveness in detecting anomalies. With respect to the study in [5], which compares the computation time of unsupervised anomaly detection algorithms, we consider recently proposed ANN-based algorithms and evaluate a larger number of performance metrics.

The paper is organized as follows. In Sections II and III, the motivations and the procedure for the selection of the approaches and of the datasets subject of our study are presented. In Section IV, details on the design of the experiments are provided; experimental results are discussed in Section V. Finally, conclusions are drawn and anticipated future works are described in Section VI.

II. Selection of the Anomaly Detection Techniques

In [6], a taxonomy of ANN-based approaches is proposed, in which they are primarily classified as: *generative, discriminative*, or *hybrid*. However, discriminative algorithms are based on supervised learning and, according to [7], they cannot be effectively used for anomaly-based intrusion detection. Indeed, the data distribution of the time series, which is the usual type of data analyzed by anomaly detection algorithms, is likely to change over time in this specific domain, resulting then in the need for the intrusion detection system to be able to recognize anomalies that have never appeared before.

For the above reasons, we focus our analysis on generative and hybrid algorithms based on ANN. The authors of [6] identify three sub-categories of generative approaches, namely: *Autoencoders (AE), Recurrent Neural Networks (RNN), Boltzmann Machines (BM)* and one sub-category of the hybrid approaches, namely, *Generative Adversarial Networks (GAN)*. This classification drives our research, and we select representative algorithms from each one of the aforementioned sub-categories.

The selection process entailed a search of approaches for which a recent implementation was published online. We searched on https://paperswithcode.com/ and used a combination of the keywords: *anomaly detection*, *deep learning*, *autoencoder*, *recurrent neural network*, *boltzmann machine*, *generative adversarial networks*, and restricted the results to works published in or after 2018. Of the resulting works, we picked those that had been published in highly ranked conferences or journals, and specifically we selected, for each-sub-category: REPresentations for a random nEarest Neighbor (*REPEN*) [8] and *DevNet* [9] for AE, *OmniAnomaly* [10] for RNN, Multi-Objective Generative Adversarial Active Learning (*MO-GAAL*) [11] for GAN. The BM sub-category has been less investigated with respect to the others. As a consequence, we were not able to find any published implementation for it. In the following, we provide a short summary of the main characteristics of each one of the considered approaches, and how they relate to each other.

A. REPEN

Pang et al. [8] note that most of the approaches to anomaly detection based on representation learning are not end-to-end. That is, data representation is learned without considering the outlier detection technique that is subsequently utilized. For this reason, they propose a framework, named *RAMODO*, which incorporates the outlier detection method into the objective function of the underlying representation learning algorithm. Furthermore, differently from classical unsupervised approaches, the proposed one can take advantage of few labeled samples, if present, in order to improve the detection performance. An implementation of the RAMODO framework named *REPEN* is then introduced, making use of the *Sp* distance-based outlier detection technique [12], which has shown state-of-the-art accuracy and scalability. Overall, the authors state that REPEN has a time complexity that is linear w.r.t. data dimensionality and size, while an estimate of the space complexity is not provided. However, performance data (e.g., execution time, speed-up, etc.) are not reported.

B. DevNet

Instead of focusing on learning the data representation, the same authors of [8] realized a novel approach that directly calculates a score for a given input sample [9]. The proposed framework, named *DevNet* from Deviation Networks, is composed of three main modules: (i) an anomaly scoring network, (ii) a reference score generation, and (iii) a deviation loss calculator. Module (i) is implemented using the encoder part of an autoencoder, which learns a M-dimensional representation of D-dimensional data, followed by a single-layer scoring function, which computes the score of the input sample as a linear combination of its M-dimensional representation. Module (ii) is designed to calculate the average score and the standard deviation of the normal samples, which will in turn be used as reference values for the comparison with the input sample that must be classified. Finally, module (iii) optimizes the scoring network by implementing a loss function based on the *z-score* of the input sample. It is noteworthy that DevNet does not properly handle time series-based data, because it performs a stratified random sampling of the input data.

According to the authors, the computational complexity of DevNet is $O(n\_epochs \times n\_batches \times b \times (Dh_1 + h_1 h_2 + \cdots + h + H))$ for the training phase and $O(I(Dh_1 + h_1 h_2 + \cdots + h_H))$ for the testing phase, where n_epochs is the number of training epochs, $n\_batches$ and b are respectively the number of training batches and the batch size, h_i is the number of neural units at the i-th hidden layer, D is the dimensionality of the data, and I is the data size of the test set.

This assessment is confirmed by experimental results, which show that DevNet exhibits a linear time complexity with respect to both data size and number of features, while at the same time outperforming REPEN by a factor of 10 to 20 on large datasets, due to a computationally efficient loss function.

C. OmniAnomaly

Su et al. [10] proposed an anomaly detection technique, named *OmniAnomaly*, to specifically handle multivariate time series and the temporal dependence between data instances. In details, OmniAnomaly is characterized by two phases: an off-line training and an on-line detection. During the off-line training, a dataset with normal data is first pre-processed (i.e., data is standardized and segmented using sliding windows), and then used to train a model using: (i) a Gated Recurrent Unit (GRU) [13], a variant of a general Recurrent Neural Network (RNN) [14] to capture temporal dependence between the input observations; (ii) a Bayesian variational autoencoder [15] to compress the input space into a latent space of reduced dimension; (iii) a linear Gaussian state space model to model the temporal dependence of the variables in the latent space; and (iv) a chain of planar mapping transformations to learn a non-Gaussian posterior distribution of the variables in the latent space. In the on-line detection, each sample is assigned with an anomaly score, which is subsequently used in combination with a dynamic threshold to finally classify input samples as normal or anomalies.

Notably, a study of the complexity of the proposed algorithm has not been conducted. Instead, an empirical evaluation of the performance including the training time for the three datasets considered in the experimentation has been reported, showing that the training time requires almost 90 minutes per epoch for the largest dataset and on the reference GPU.

D. SO-GAAL and MO-GAAL

Another approach to anomaly detection, which makes use of the Generative Adversarial Network paradigm, has been proposed in [11]. First, the authors re-define the concept of anomaly with respect to the density of samples in the sample space. Indeed, the assumption is that a sample that resides in a region with a low density can be considered an outlier. However, since computing the density function for the entire space could be computationally prohibitive, the problem is reformulated as a classification problem, where n data points are synthetically generated from a uniform reference distribution and added to the original dataset. This formulation still suffers from the curse of dimensionality because, in order for the synthetically generated outliers to be representative of the sample space, their amount should polynomially grow with the number of dimensions, leading again to a problem that is intractable. For this reason, instead of generating sparse outliers from a uniform distribution, a GAN is used to generate outliers that occur near the real data. A GAN is composed of two neural networks, a generator G and a discriminator D, which engage in a min-max game. Here, the generator is in charge of generating outlier samples, and the discriminator is in charge of detecting whether the input data is an outlier generated by G, or an actual sample representing normal data extracted from the dataset. One of the issues of this approach, named Single-Objective General Adversarial Active Learning (SO-GAAL), is that after a certain number of epochs, the generator will generate outlier samples that will be more similar to the real data, making it impossible for the discriminator to effectively distinguish between normal samples and anomalies. Hence, a stopping criterion should be introduced, such that the game terminates when the generated outliers provide enough information to the classifier to effectively detect anomalies. However, this is not possible because there is no prior information during the training stage.

For this reason, MO-GAAL is introduced. The latter is based on k generators $G_1, \ldots, G_k$, and a single discriminator D. Each generator works on a partition $X_i (i = 1, \ldots, k)$ of the dataset, with $X = \bigcup_i X_i$, where the samples within a given partition are similar to each other. Experiments show that the execution time of MO-GAAL exhibits a linear increase with respect to the data size, with a high initial overhead.

III. Attacks Datasets

The Intrusion Detection domain has historically suffered from the lack of publicly available datasets, due to privacy concerns [16]. For this reason, most of the anomaly-based intrusion detection techniques proposed in the last two decades, including several recent ones (e.g., [17]–[19]), have been validated using few, outdated datasets, such as, DARPA'98 and DARPA'99 [20], KDDCUP'99 [21], and NSL-KDD [22]. However, this trend is slowly changing, and new datasets are being made publicly available. Indeed, according to a recently published survey [23], 36 datasets have been released in the past two decades, with an increasing release rate in the last years. Each dataset is specialized on specific aspects or characteristics of the network traffic. For example, some of them are provided in raw packet capture format, while others are provided as a list of network flows. Some are built using real network traffic, others rely on some synthetic data generator or on traffic generated in a closed, emulated environment. For an extensive description of the datasets and a detailed classification, the interested reader can refer to [23].

In terms of performance, a proper evaluation of the scalability of the algorithms described in Section II requires using datasets of different sizes. For this reason, we select two datasets with sizes that range between 150K samples to 2.8M samples, namely: NSL-KDD and CIC-IDS-2017 [24].

A. NSL-KDD

NSL-KDD has been selected because, despite being old (it was created in 1999), and potentially not representative of current network patterns, it is the one that has been most frequently used in the past decade. Therefore, results obtained using this dataset can potentially be compared to other approaches published in the literature, at least from the effectiveness perspective. Its size is relatively small, counting roughly 150K samples. The number of features of NSL-KDD is 41, and its contamination rate is 46.5%, meaning that 46.5% of its samples are anomaly points.

B. CIC-IDS-2017

Similarly to NSL-KDD, CIC-IDS-2017 was created in an emulated environment. Two types of traffic were generated: background and malicious. The former is the result of the simulated activities of 25 users, working with standard protocols, such as, HTTP, FTP, and SSH. The latter contains various types of attacks, which range from Denial of Service (DoS), to Cross-Site Scripting (XSS), and brute force attacks to FTP and SSH servers. Different attack types are executed at different times, with no overlap on the dataset. The latter is divided into multiple segments, each one referring to a specific day, or time of the day, and contains a total of 2.8M records, each one representing a bi-directional network flow. Each record is characterized by 79 features, including the label, and the contamination rate is roughly 19.7%.

IV. Experiment Design

The primary focus of the evaluation conducted in this work is to assess the performance of the selected ANN-based anomaly detection algorithms. However, it is not possible to consider performance metrics without taking into account the effectiveness of the algorithms. Indeed, performance and effectiveness of ANN-based anomaly detection algorithms are highly intertwined, due to the existence of hyperparameters that affect both of them. Although it is out of the scope of this work to carry out an extensive analysis of the configuration of the hyperparameters, it is worth noting that all the algorithms that we have selected use a Stochastic Gradient Descent (SGD) approach to find the minimum of a loss function. SGD is an iterative algorithm, and the number of iterations employed during the training phase has an impact on both, the effectiveness and the performance of the approaches that use it. Ideally, the highest effectiveness is obtained when the loss function reaches a global minimum. However, the problem of finding a global minimum using SGD is undecidable when the loss function is not convex.

For this reason, the stopping criterion for the training phase is based on training time, rather than on the achievement of a given effectiveness objective. Specifically, we set an experiment time limit of 12 hours, for every algorithm and every dataset instance, as described in Section IV-A. This threshold has been chosen in order to balance the number of epochs that will be completed by each algorithm with the computational power needed to run all the experiments.

A. Data Preparation

For each dataset of 100% samples, we created data subsets by randomly extracting 20%, 40%, 60%, and 80% of the samples. This extraction is needed in order to analyze, for each algorithm and each dataset, the effectiveness and the performance with respect to the length of the data.

For all the experiments, with the exception of those regarding OmniAnomaly, we use 4-fold cross validation. Traditional K-fold cross validation cannot be used with OmniAnomaly, because it would break the temporal dependency of the samples. Therefore, for that specific algorithm, we used the time series cross-validator implemented in `scikit-learn` [25]. This cross-validator divides the dataset in $K > 2$ splits and then, $\forall n . n = 1, 2, \ldots, K-1$, it uses samples from splits in the interval $[1, n]$ for training and samples from split $n+1$ for testing. In our case, we set $K = 4$.

In order to replicate with the highest possible degree of fidelity the same environment described by the authors of the selected algorithms, we applied the minimum possible amount of pre-processing to the generated dataset instances. Specifically, the only pre-processing step that we always executed, and that was not always described in the considered works, is *one-hot encoding*. This step was needed in order to convert categorical data to numerical data, which is required by neural network-based algorithms. Furthermore, as indicated by the referenced papers or implemented in their corresponding source code, we performed the following pre-processing activities:

- REPEN - Data was normalized using min-max scaling, as reported in [8], [26].
- DevNet - No additional pre-processing steps were performed.
- OmniAnomaly - In [10], the authors mention a data standardization process, without providing any additional details. However, we found in the `utils.py` source file (line 78), that a min-max normalization was applied.
- SO-GAAL and MO-GAAL - Labels of the input dataset were flipped, in order to match the expected input.

B. CPU Affinity and Monitoring System

All the experiments have been conducted on CloudLab *c220g5* machines. These are nodes characterized by a Non-Uniform Memory Access (NUMA) architecture. Each node has 2 physical CPUs with 10 cores each, for a total of 40 cores when hyperthreading is enabled. The amount of RAM is 192GB. Since the machines have a password protected BIOS, hyperthreading could not be disabled. Therefore, we used the Linux `taskset` command to set the affinity of the experiment processes so that no more than one sibling was used for a given physical core[1]. Furthermore, in order to avoid most of the performance measurements issues due to the existence of multiple NUMA nodes, we set the affinity of the experiment processes to only use cores from a single physical CPU. Indeed, from preliminary experimental results we found that, when threads are distributed across multiple NUMA nodes, there is a non-negligible performance degradation due to the possibility that a thread may need to access memory that is non-local, thus increasing the memory access time.

[1] With hyperthreading enabled, each physical core is split into two virtual siblings; the map of the siblings can be obtained reading the `/proc/cpuinfo` file.

V. Experimental Results

In this section we illustrate, for each of the selected algorithms and datasets: (i) the trend of the learning curves, with respect to both, number of epochs and training time; (ii) the usage of memory over time; (iii) the usage of memory with subsets of different size; (iv) the training time with subsets of different size.

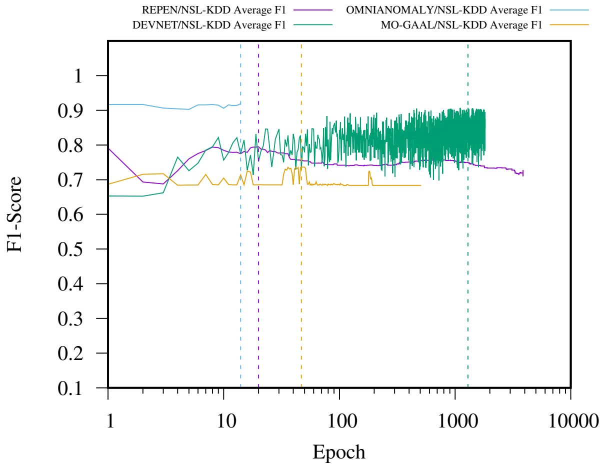

Fig. 1. Learning Curves vs Number of Epochs with NSL-KDD

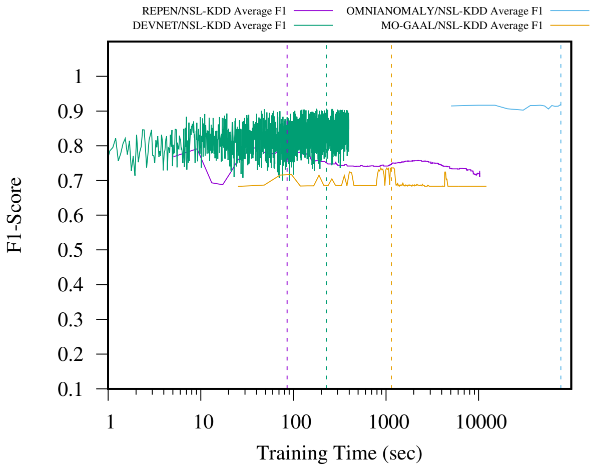

Fig. 2. Learning Curves vs vs Training Time with NSL-KDD

Figures 1 and 2 show the learning curves when the selected algorithms are fed with the 80% subset of the NSL-KDD dataset. The reason why we selected the 80% subset for this experiment, instead of the full dataset, is to carry out a fair comparison: on one hand, one of the algorithms that we are comparing, namely OmniAnomaly, uses a time-series cross validator which, with $K = 4$, allows a training process with up to 80% of the entire data. On the other hand, the largest size of CIC-IDS-2017 that MO-GAAL was able to handle was 80%. It is worth noting that the time reported in Figure 2 is training time, which is only one of the components of the total experimentation time. For this reason, the plotted traces in most cases have curves that do not reach a time of 12 hours.

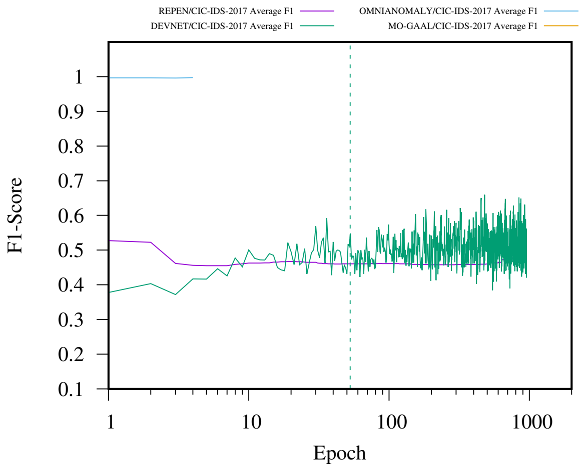

Fig. 3. Learning Curves vs Number of Epochs with CIC-IDS-2017

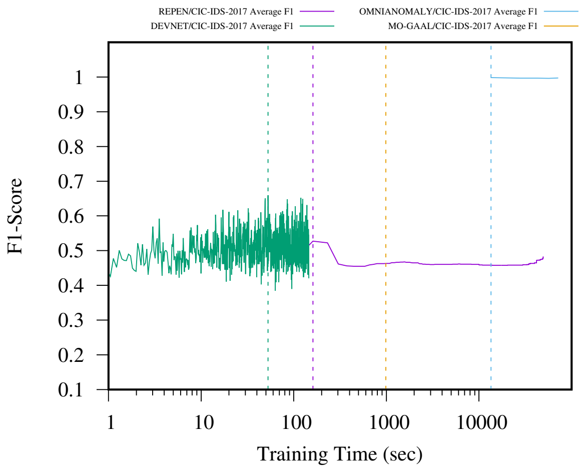

Fig. 4. Learning Curves vs Training Time with CIC-IDS-2017

Figure 1 shows how the F1-Score varies according to the number of epochs, and Figure 2 shows how the same score varies according to the training time. The dotted vertical bars correspond respectively to the epoch and to the time at which the maximum score has been obtained, for each algorithm. The highest score has been achieved by OmniAnomaly. Such a high score is due to its RNN-based architecture, which proves to be particularly indicated in the intrusion detection domain, where data samples are indeed temporally correlated. However, the drawback of this approach is an increased computational complexity, which leads to the highest training time among the chosen algorithms.

A similar observation can be made for the results shown in Figures 3 and 4. The latter show how the F1-Score varies according respectively to the number of epochs and to the training time, when the 80% subset of CIC-IDS-2017 was used as the input to the selected algorithms. OmniAnomaly clearly outperforms the other algorithms, achieving a nearly perfect F1-score. It is worth noting that, with the exception of DevNet, the vertical bars in Figure 3 are not visible because the highest score has been obtained either at epoch 1, in the case of REPEN, or at epoch 0, in the case of OmniAnomaly and MO-GAAL.

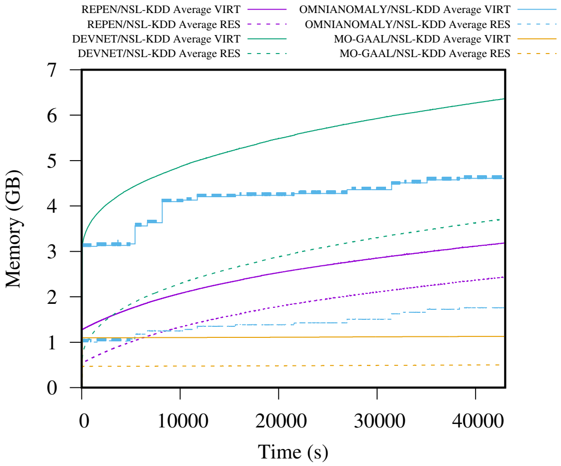

Fig. 5. Memory Usage Over Time with NSL-KDD

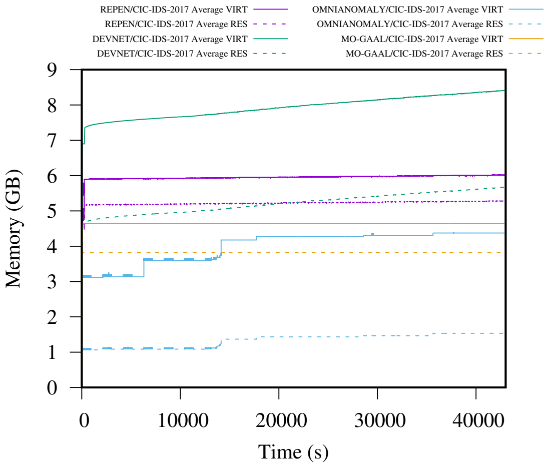

Fig. 6. Memory Usage Over Time Time with CIC-IDS-2017

Figures 5 and 6 show the memory traces of the processes running the experiments. Both the virtual size VIRT and the resident size RES of the processes were reported, for each algorithm. For the same reasons as above, the plotted memory traces are those related to the execution of the algorithms with the 80% subset of the datasets. The plots show that, despite being based on ANN, with the exception of MO-GAAL, the selected approaches have a memory usage that is directly proportional to the number of epochs used for training. Indeed, REPEN and DevNet, which are the fastest algorithms in terms of per-epoch execution time, manage to complete more epochs than their counterparts in the fixed 12-hours experiment time, as shown in Figures 1 and 3. Executing more epochs leads them to have the greatest increase in memory usage, both with NSL-KDD and CIC-IDS-2017. Furthermore, it is possible to observe how, despite OmniAnomaly only completes a relatively small number of epochs, its memory increase is not negligible. Finally, while the memory usage of MO-GAAL appears to be mostly constant, this is due to the fact that it is the one with the highest per-epoch execution time, with a single epoch completed when run against 80% of CIC-IDS-2017. This correspondence between memory usage and number of epochs is also evident from the plot of Figure 7. The latter shows how the memory usage varies according to the size of the selected subset of CIC-IDS-2017. Interestingly, the memory used by OmniAnomaly decreases with an increasing size of the input data. This is due to the fact that an increase in the input data size corresponds to a lower number of completed epochs, when the experiment has a fixed duration. Having a high per-epoch memory requirement leads then to a decrease in the memory usage when fewer epochs are completed. Similar conclusions can be drawn when NSL-KDD is used as the input dataset. For space reasons, we do not include its plot.

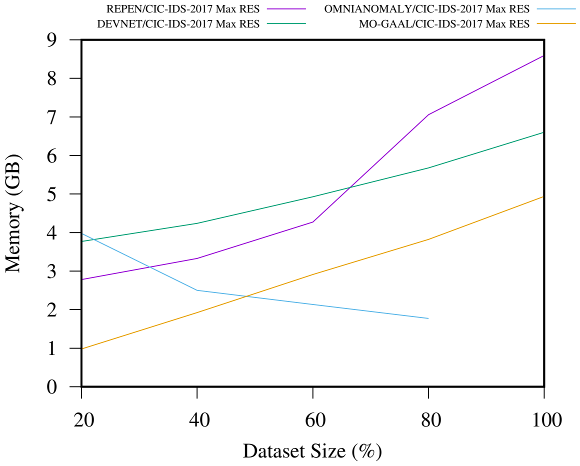

Fig. 7. Memory Usage vs CIC-IDS-2017 Subset Size

Finally, Figure 8 reports the training times according to the size of the selected subset of CIC-IDS-2017. Ideally, we would consider the training time as the minimum time needed to achieve the maximum F1-Score in the duration of the experiment. However, as shown in Figures 3 and 4, the F1-Score has non negligible fluctuations throughout the training phase. For this reason, we define the training time as the minimum time to reach a F1-Score that is at least 5% lower than the maximum reachable F1-Score. It is possible to see that REPEN has a linear time increase with respect to the size. MO-GAAL and OmniAnomaly instead appear to have a trend that is quadratic. Finally, the training time of DevNet does not exhibit a clear trend, due to the fact that its F1-Score has a fluctuation during the training that is higher than the 5% threshold we set on the F1-Score.

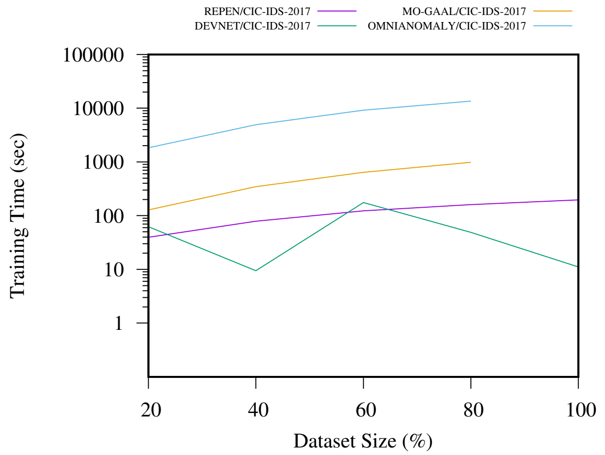

Fig. 8. Training Time vs CIC-IDS-2017 Subset Size

VI. Conclusion and Future Works

Deciding which anomaly detection approach to use for intrusion detection is a complex task, where a trade-off between effectiveness and performance must usually be accepted. In this work we carried out a joint assessment of the performance and of the effectiveness of four state-of-the-art ANN-based anomaly detection algorithms (REPEN, DevNet, OmniAnomaly, MO-GAAL) applied to two reference intrusion detection datasets (NSL-KDD and CIC-IDS-2017). Experimental results showed that none of the algorithms dominates the others in terms of both, effectiveness and performance. In details, OmniAnomaly, which is based on RNN, outperforms all the others in terms of effectiveness, by achieving the highest F1-Score. However, it also exhibits the highest training time and per-epoch memory usage. In contrast, REPEN appears to be one of the fastest algorithms in terms of training time. However, its maximum F1-Score is the lowest in the CIC-IDS-2017 case, and the second lowest in the NSL-KDD case.

As future work, we will extend the proposed analysis to assess the impact of the number of features and of the availability of multiple CPUs on both, effectiveness and performance. Furthermore, we anticipate a study on the energy consumption of anomaly detection algorithms. This study can also provide insights on how to select an anomaly detection algorithm for edge computing scenarios, which are usually based on decentralized, low-powered computational resources.

Acknowledgements

All the experiments have been conducted on the NSF-sponsored CloudLab platform [27].

References

[1] H. Hindy, D. Brosset, E. Bayne, A. K. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens, “A taxonomy of network threats and the effect of current datasets on intrusion detection systems,” *IEEE Access*, vol. 8, pp. 104 650–104 675, 2020.

[2] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[3] T. Zoppi, A. Ceccarelli, T. Capecchi, and A. Bondavalli, “Unsupervised anomaly detectors to detect intrusions in the current threat landscape,” *ACM/IMS Trans. Data Sci.*, vol. 2, no. 2, pp. 1–26, 2021.

[4] J. Meira, R. Andrade, I. Praça, J. Carneiro, V. Bolón-Canedo, A. Alonso-Betanzos, and G. Marreiros, “Performance evaluation of unsupervised techniques in cyber-attack anomaly detection,” *J. Ambient Intell. Humaniz. Comput.*, vol. 11, pp. 4477–4489, 2020.

[5] M. Goldstein and S. Uchida, “A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data,” *PLoS One*, vol. 11, no. 4, 2016.

[6] A. Aldweesh, A. Derhab, and A. Z. Emam, “Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues,” *Knowledge-Based Systems*, vol. 189, p. 105124, 2020.

[7] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou *et al.*, “Time-series anomaly detection service at Microsoft,” in *Proc. of ACM SIGKDD ’19*, 2019, pp. 3009–3017.

[8] G. Pang, L. Cao, L. Chen, and H. Liu, “Learning representations of ultrahigh-dimensional data for random distance-based outlier detection,” in *Proc. of ACM SIGKDD ’18*, 2018, pp. 2041–2050.

[9] G. Pang, C. Shen, and A. van den Hengel, “Deep anomaly detection with deviation networks,” in *Proc. of ACM SIGKDD ’19*, 2019, pp. 353–362.

[10] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, “Robust anomaly detection for multivariate time series through stochastic recurrent neural network,” in *Proc. of ACM SIGKDD ’19*, 2019, pp. 2828–2837.

[11] Y. Liu, Z. Li, C. Zhou, Y. Jiang, J. Sun, M. Wang, and X. He, “Generative adversarial active learning for unsupervised outlier detection,” *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 8, pp. 1517–1528, 2019.

[12] M. Sugiyama and K. Borgwardt, “Rapid distance-based outlier detection via sampling,” *Advances in Neural Information Processing Systems*, vol. 26, pp. 467–475, 2013.

[13] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[15] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *Proc. of 2014 Int’l Conf. on Learning Representations (ICLR ’14)*, 2014.

[16] S. Iannucci, H. A. Kholidy, A. D. Ghimire, R. Jia, S. Abdelwahed, and I. Banicescu, “A comparison of graph-based synthetic data generators for benchmarking next-generation intrusion detection systems,” in *Proc. of 2017 IEEE Int’l Conf. on Cluster Computing (CLUSTER ’17)*, 2017, pp. 278–289.

[17] Y. Li, R. Ma, and R. Jiao, “A hybrid malicious code detection method based on deep learning,” *International Journal of Security and Its Applications*, vol. 9, no. 5, pp. 205–216, 2015.

[18] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, “A deep learning approach to network intrusion detection,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[19] B. Zhang, Y. Yu, and J. Li, “Network intrusion detection based on stacked sparse autoencoder and binary tree ensemble method,” in *Proc. of 2018 IEEE Int’l Conf. on Communications Workshops*, 2018, pp. 1–6.

[20] “DARPA’98 and DARPA’99 datasets,” https://www.ll.mit.edu/r-d/datasets?keywords=DARPA.

[21] “KDDCUP’99 dataset,” https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[22] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *Proc. of 2009 IEEE Symp. on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[23] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, “A survey of network-based intrusion detection data sets,” *Computers & Security*, vol. 86, pp. 147–167, 2019.

[24] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, “Towards a reliable intrusion detection benchmark dataset,” *Software Networking*, vol. 2018, no. 1, pp. 177–200, 2018.

[25] “Sklearn time-series cross validator,” https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html.

[26] G. Pang, K. M. Ting, and D. Albrecht, “LeSiNN: Detecting anomalies by identifying least similar nearest neighbours,” in *Proc. of 2015 IEEE Int’l Conf. on Data Mining Workshops*, 2015, pp. 623–630.

[27] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide *et al.*, “The design and operation of CloudLab,” in *Proc. of 2019 USENIX Ann. Tech. Conf. (ATC ’19)*, 2019, pp. 1–14.