Towards efficient deep autoencoders for multivariate time series anomaly detection

Marcin Pietroń, Dominik Żurek, Kamil Faber, Roberto Corizzo

AGH University of Krakow, Poland
{pietron, dzurek, kfaber}@agh.edu.pl
American University, Washington DC
{rcorizzo@american.edu}

Abstract. Multivariate time series anomaly detection is a crucial problem in many industrial and research applications. Timely detection of anomalies allows, for instance, to prevent defects in manufacturing processes and failures in cyberphysical systems. Deep learning methods are preferred among others for their accuracy and robustness for the analysis of complex multivariate data. However, a key aspect is being able to extract predictions in a timely manner, to accommodate real-time requirements in different applications. In the case of deep learning models, model reduction is extremely important to achieve optimal results in real-time systems with limited time and memory constraints. In this paper, we address this issue by proposing a compression method for deep autoencoders that involves three key factors. First, pruning reduces the number of weights, while preventing catastrophic drops in accuracy by means of a fast search process that identifies high sparsity levels. Second, linear and non-linear quantization reduces model complexity by reducing the number of bits for every single weight. The combined contribution of these three aspects allow the model size to be reduced, by removing a subset of the weights (pruning), and decreasing their bit-width (quantization). As a result, the compressed model is faster and easier to adopt in highly constrained hardware environments. Experiments performed on popular multivariate anomaly detection benchmarks, show that our method is capable of achieving significant model compression ratio (between 80% and 95%) without a significant reduction in the anomaly detection performance.

Keywords: Deep learning · Autoencoders · Anomaly detection · Pruning

1 Introduction

Multivariate time series anomaly detection is a very popular machine learning problem in many industry sectors. Therefore, many research works have been proposed in this field [9,2,4] Recent works highlight that the best results in terms of detection accuracy are achieved with deep autoencoders [4] based on convolutional layers. Other models with satisfactory results are autoencoders based on graph neural networks [3,13] and recurrent layers [9,7]. It is worth noting that their effectiveness depends heavily on the specific characteristics of the dataset they are assessed on. Neuroevolution provides a valuable way to address this issue, with the potential to extract optimized models for any given dataset. A notable example is the AD-NEv framework [11], which supports multiple layer types: CNN-based, LSTM based and GNN-based. One potential burden of deep autoencoder models is that each additional layer or channel inside the layer slows down the training and inference process, which negatively affects their efficiency in real-time or embedded systems. In fact, any delay in their inference can have a significant impact on the operation of the reliability of these systems. To this end, compression algorithms can significantly help in reducing the number of CPU cycles required to process input data. The second advantage of their adoption is the memory footprint reduction they provide. This aspect is extremely important in cases where models are exploited in dedicated hardware e.g. IoT, edge, etc. Many works focus on compression for deep learning [8,1,14],particularly for image-based data and natural language processing. The most efficient techniques are pruning [8,14,6] and quantization [8,1]. The pruning process presented in these works can be divided into structured pruning and unstructured pruning. The quantisation can be linear or non-linear. These works show that many deep learning models may present redundant weights which can be removed without any significant drop in detection accuracy. Additionally, given the robustness of these models to noise in input data, weights can be quantized to a lower bit format, further decreasing the memory footprint. However, studies focusing on reducing the complexity of deep learning models are still limited in the anomaly detection field. To the best of our knowledge, there is no work devoted to compressing models on multivariate anomaly detection benchmarks. To this end, in this paper we propose a compression workflow based on pruning and quantization. We adopt convolutional and graph autoencoders which have shown to be the most robust model architectures in anomaly detection tasks. In our work, pruning is incorporated in the training process, while linear and non-linear quantization based on nearest neighbour rounding is run on pruned and pre-trained models. Our experiments leveraging state-of-the-art base model architectures [4] show that compression techniques like pruning and quantization can significantly reduce the complexity of deep model architectures in multivariate anomaly detection tasks.

2 Method

In this section, we describe our proposed compression method for anomaly detection autoencoder models. Autoencoders learn a compressed representation, i.e., latent space Z of raw input data X in an unsupervised manner. Autoencoders are made of two parts: the encoder E, which transforms (encodes) the original input to the latent space, and the decoder D, which transforms (decodes) the latent space Z to the original feature space:

$$Z = E_{\Theta}(X) = e_{\theta_L}(e_{\theta_{L-1}}...(e_{\theta_0}(X))) \tag{1}$$

$$AE_{\Theta}(X) = D_{\Theta}(Z) = d_{\theta'_0}(d_{\theta'_1}...(d_{\theta'_L}(Z))) \tag{2}$$

The objective of the training is the minimization of the *reconstruction loss*, which corresponds to the difference between the decoder's output and the original input data.

2.1 Pruning

The goal of the first stage is to carry out a pruning process, which allows the retention of the most relevant parameters, thus saving computational resources involved for model inference. The general idea is to identify a subset of weights that yield a similar anomaly detection performance to the full model. By doing so, it is possible to discard the remaining weights and reduce the model size, which facilitates its provisioning in resource-limited environments such as edge and IoT. To achieve this goal, in this stage we devise pruning algorithms that involve: *i)* identification of a separate sparsity level for each layer, and *ii)* pruning with model retraining, to foster a more effective identification of the sparsity levels [10].

The representation of the pruned model AE^p_{Θ} is a tuple $AE^p_{\Theta} = (AE_{\Theta}, M)$, where AE_{Θ} is the original model composed by convolutional, fully-connected or LSTM layers e_{θ_i} and d_{θ_i}, arranged in the specified order. The weights for each layer are represented by the Θ tensor consisting of parameters from the encoder and decoder:

$$\Theta = \{\theta_0, \theta_1, ..., \theta_L, \theta'_L, ..., \theta'_1, \theta'_0\} \tag{3}$$

The mask tensor M contains $'0'$ and $'1'$ entries, which denote, for a given layer, weights that are either pruned or retained, respectively:

$$M = \{M_{e_{\theta_0}}, M_{e_{\theta_1}}, \dots, M_{e_{\theta_L}}, M_{d_{\theta_L}}, \dots, M_{d_{\theta_1}}, M_{d_{\theta_0}}\}. \tag{4}$$

We note that, in our work, a *sparsity level* υ_i for a layer i is defined as the ratio between the number of utilized weights and the total number of weights at that layer:

$$\upsilon_i = \frac{\sum_j M_{i,j}}{|M_i|} \quad , \quad \upsilon_{ws} = \sum_{j=0}^{L} \frac{|\theta_j| \cdot \upsilon_j}{|\Theta|} + \sum_{j=0}^{L} \frac{|\theta'_j| \cdot \upsilon_j}{|\Theta|}. \tag{5}$$

The weighted sparsity υ_{ws} is computed as a sum of two ratios which define the local sparsity for the encoder and decoder counterparts of the model.

We can find a threshold ϵ_i which ensures to retain the proper of weights (having a value greater than x) according to the sparsity υ_i.

$$\epsilon_i = x \quad , \quad \text{where} \quad \frac{|abs(\theta_i) > x|}{|\theta_i|} = \upsilon_i. \tag{6}$$

We leverage ϵ_i to identify the strongest weights which should be retained for a given layer. It defines mask M_i which is assigned to a specific layer and each entry $M_{i,j}$ can be regarded as binary value:

$$M_{i,j} = \begin{cases} 0 & \text{if } abs(\theta_{i,j}) < \epsilon_i \\ 1 & \text{if } abs(\theta_{i,j}) > \epsilon_i \end{cases} \quad (7)$$

Algorithm 1 Pruning - main algorithm

```
Require: $AE$, $P_S$, $V_{MIN}$, $V_{MAX}$, $N$, $N_{it}$, $O$, $B$, $P_S$
 1: $\Lambda \leftarrow P_S$ copies of $AE$, $K \leftarrow \emptyset$
 2: for $i = 0$ to $P_S$ do
 3:     $M \leftarrow \emptyset$
 4:     for $l = 0$ to $2 \cdot L$ do
 5:         $min_l, max_l \leftarrow V_{MIN}[l], V_{MAX}[l]$
 6:         $mean_l, std_l \leftarrow min_l + \frac{max_l - min_l}{2}, mean_l + \frac{max_l - min_l}{6}$
 7:         $\upsilon_l \leftarrow$ bound(sample($\mathcal{N}(mean_l, std_l)$), $min_l$, $max_l$)
 8:         $\epsilon_l \leftarrow$ compute threshold based on $\upsilon_l$ (see Equation 6)
 9:         $M_l \leftarrow$ generate mask based on $\epsilon_l$ (see Equation 7)
10:         $M \leftarrow M \cup M_l$
11:     end for
12:     for $epoch = 0$ to $N_{it}$ do
13:         for $b = 0$ to $B$ do
14:             $\Theta \leftarrow$ train batch $b$ with $O$
15:             for $l = 0$ to $2 \cdot L$ do
16:                 $\theta_l = \theta_l \odot M_l$        // Prune weights for layer $l$ according to mask
17:             end for
18:         end for
19:     end for
20:     $\Lambda \leftarrow \Lambda \cup (AE_\Theta, M, O)$
21:     $K \leftarrow K \cup (F_1 + \alpha \cdot \upsilon_{ws})$
22: end for
23: $i \leftarrow$ argmax($K$)
24: for $epoch = 0$ to $N$ do
25:     for $b = 0$ to $B$ do
26:         $\Theta_i \leftarrow$ train batch $b$ with $O_i$
27:         for $l = 0$ to $2 \cdot L$ do
28:             $\theta_l = \theta_l \odot M_l$        // Prune weights for layer $l$ according to mask
29:         end for
30:     end for
31: end for
```

Algorithm 1 starts with the initialization of the model population, and it sets up the empty list for pruned model quality metrics (line 1 and 2). Then, in a loop, the models are pruned following a retraining process (lines 2–22). At the beginning of the loop, the algorithm goes through the autoencoder layers (see internal loop: lines 4–10) and sets up the initial sparsity levels for each encoder and decoder layer. The sparsity levels are generated by normal distribution using specified mean and standard deviation (line 7). These parameters are computed based on predefined sparsity boundaries given as input parameters (line 6). Once the sparsity is computed, the algorithm uses Equation 7 to define ϵ_l value for each layer (line 8). Afterwards, the layer mask is set (line 9). The next internal loop is responsible for short training with predefined N_{it} epochs (lines from 12 to 19). The batch training is performed After each batch, the chosen layer weights are set to zero using element-wise multiplication with the layer mask (line 16). Then, the evaluation of the shortly pruned model is performed. The achieved F1 metric is added to the list (line 21). At the end, in lines from 24 to 31, the model with the best efficiency from the population is taken to the final long-term training stage with N epochs (note: the model is taken with its mask tensor, N»N_{it}).

2.2 Quantization

The quantization process allows our models to be processed further, reducing their complexity. Quantization is a viable process to reduce a complete floating point representation of values to a format with fewer bits. In this paper, we present two types of quantization: linear and non-linear.

Linear quantization can be thought of as a mapping function from a floating-point value $x \in \mathcal{S}$ to a fixed-point $q \in \mathcal{Q}$ through a function $f_{\mathcal{Q}} : \mathcal{S} \to \mathcal{Q}$:

$$q = f_{\mathcal{Q}}(x) = \mu + \sigma \cdot \mathrm{round}(\sigma^{-1} \cdot (x - \mu)), \tag{8}$$

where $\mu = 0$ and $\sigma = 2^{-\textbf{frac\_bits}}$, σ is a scaling factor (shift up or down). Integer bit-width can be defined as:

$$\textbf{int\_bits} = \mathrm{ceil}(\log_2(\max_{x \in \mathcal{S}} |x|)). \tag{9}$$

The second type of quantization we present in this paper is the non-linear method inspired by [12]. At first, we cluster weights in each layer leveraging the k-Means algorithm according to the desired number of clusters ω. Then, we assign an identifier of the cluster to each weight in a layer, selecting the closest cluster to the original value. The next step quantizes cluster centroids to ψ, which defines the bit-width format. Finally, we create a codebook, which contains a mapping between each original weight w and it corresponding quantized cluster's centroid w_q. A similar approach has been adopted in [12] and showcased effective compression capabilities in the context of NLP models and GPU-based DL models acceleration, respectively.

3 Results and discussion

The research questions posed by our paper are the following:

RQ1. How efficient dynamic pruning can be in anomaly detection auto-encoder architectures?

RQ2. How effectively can deep state-of-the-art anomaly detection models be reduced by means of quantization?

RQ3. What is the efficiency of linear and non-linear quantization on a pretrained autoencoder?

In our experiments, we consider state-of-the-art architectures in recent benchmarks for anomaly detection [11,5], i.e. convolutional autoencoders (CNN AE) and graph-based autoencoders (GDN). We adopt popular benchmark datasets: SWAT, WADI-2019, MSL, SMAP. The CNN AE for SWAT and WADI-2019 consist of 6 layers, for SMAP and WADI they have 12 layers. All our experiments were executed on a workstation equipped with Nvidia Tesla V100-SXM2-32GB GPUs using PyTorch framework. The V_{MIN} and V_{MAX} parameters were set to 0.2 and 0.8, respectively. The population size P_S was set to 16. In the pruning experiments these models were trained from scratch by Algorithm 1. The linear and nonlinear quantization were run on pretrained models. In all quantization experiments the output neuron activations are in 16-bit format. The baseline results achieved by CNN AE are the best among all models tested on the analyzed datasets [11]. The GDN achieves the second result in the case of the WADI-2019 and SWAT [11]. The CNN AE achieves following point-wise F1: WADI-62.0, SWAT-82.0, MSL-77.0, SMAP-57.0. The GDN gives F1: WADI-57.0, SWAT-81.0, MSL-30.0, SMAP-33.0.

Results in Table 1 show the performance of models following the pruning stage of our proposed compression workflow with different Sparsity levels. We observe that with a sparsity level of 0.2 the anomaly detection performance drops slightly in SWAT: from 82.0 to 81.45 (CNN AE) and from 81.0 to 80.51 (GDN). The performance drop is more significant for WADI-2019: from 62.0 to 56.28 (CNN AE) and from 57.0 to 53.5 (GDN). These results show that is difficult to reduce significant number of weights for both models on WADI-2019. The drop for Sparity level 0.2 can be acceptable for SWAT (about 0.5). The higher Sparsity level increases the drop further for both datasets. In case of SMAP and MSL when Sparsity increases to 0.75, the performance is still at the same level as in baseline models (for both CNN AE and GDN). These surprising results can be motivated as pruning can, in some cases, provide a noise reduction capability in the presence of noisy data in multivariate time series datasets, resulting in a more robust model. Overall, our experimental result show that pruning can be an effective strategy to compress deep autoencoder models for anomaly detection, especially for MSL and SMAP datasets (**RQ1**).

Results in Table 2 show the performance of models following the quantization stage of our proposed compression workflow with different bit width configurations. Overall, our experimental results show that 16-bit and 8-bit quantization can be quite effective in reducing the complexity of deep autoencoder models used for anomaly detection tasks (**RQ2**). In case of 5-bit and 4-bit quantization there is significant drop on WADI-2019 and SWAT. Both models CNN AE and GDN are robust for 4-bit quantization and give the F1-score at the same level as the baseline counterparts. It can be observed that for nonlinear 16-bit and 8-bit there is no drop in accuracy for both models and datasets (**RQ2**). The drop in case of

Table 1. Model performance in terms of F1-Score with proposed pruning workflow and different sparsity levels applied to each layer.

Sparsity=0.2		Sparsity=0.75	
CNN AE	**GDN**	**CNN AE**	**GDN**
81.45 (SWAT)	80.51 (SWAT)	57.01 (MSL)	30.2 (MSL)
56.28 (WADI)	53.5 (WADI)	77.02 (SMAP)	32.9 (SMAP)

Table 2. Experimental results (F1-score) with linear (left|) and non-linear (|right) quantization and different bit-width configurations.

	16-bit		8-bit	
Datasets	**CNN AE**	**GDN**	**CNN AE**	**GDN**
SWAT	81.90 | 80.36	80.80 | 80.70	81.98 | 80.27	80.70 | 80.75
WADI	62.13 | 57.56	56.90 | 55.01	62.06 | 59.11	56.80 | 55.50
SMAP	77.15 | 77.02	32.85 | 32.95	77.05 | 76.81	32.82 | 32.92
MSL	57.21 | 56.97	29.95 | 29.91	57.14 | 56.98	29.91 | 29.92
	5-bit		4-bit	
Datasets	**CNN AE**	**GDN**	**CNN AE**	**GDN**
SWAT	80.70 | 78.45	79.80 | 79.54	16.44 | 16.35	23.51 | 21.43
WADI	54.52 | 17.87	55.52 | 31.52	48.20 | 10.89	45.51 | 24.45
SMAP	76.09 | 76.21	32.65 | 32.71	75.61 | 75.89	32.49 | 32.63
MSL	56.45 | 56.15	29.69 | 29.67	56.09 | 56.91	29.55 | 29.63

4-bit quantization is acceptable only for MSL and SMAP. The research results presented in [12] show that sparse 1D convolutional layers can be speed up on GPU using sparse convolution. The sparsity above 70% guarantees that sparse convolution outperforms standard CuDnn implementation. Additionally, it shows that GPU can make usage from reduced precision format. These two aspects allows to improve models time efficiency on GPU (**RQ3**).

4 Conclusions and future works

In this paper we proposed a compression workflow leveraging pruning and quantization stages. While pruning is incorporated in the training process, linear and non-linear quantization is performed on pruned and pre-trained models. Our experiments leveraging state-of-the-art convolutional and graph autoencoder model architectures revealed the trade-off between model compression and anomaly detection performance that pruning and quantization techniques can achieve in benchmark multivariate anomaly detection settings. Among key findings, we observed that pruning can be quite effective with MSL and SMAP datasets, and 16-bit and 8-bit quantization only impacted in a small drop in terms of F1 score. Additionally, the 4-bit quantization gives the same accuracy levels as in floating point mode. On the other hand, pruning was not effective with the WADI dataset. The presented results show that anomaly detection autoencoders can be reduced from 80% (8-bit quantization and 20% sparsity level, WADI-2019 and SWAT) to about 95% (4-bit quantization and 75% sparsity level, MSL and SMAP). Future work will focus on more advanced quantization techniques based on model retraining, which could decrease the drop in F1-Score for lower bit-widths.

Acknowledgments

This paper was realized with funds of Polish Ministry of Science and Higher Education assigned to AGH University and it was supported by PLGrid Infrastructure.

References

1. Al-Hami, M., Pietron, M., Casas, R., Wielgosz, M.: Methodologies of Compressing a Stable Performance Convolutional Neural Networks in Image Classification (2020)
2. Audibert, J., Michiardi, P., Guyard, F., Marti, S., Zuluaga, M.A.: Usad: Unsupervised anomaly detection on multivariate time series. In: Proceedings of the 26th ACM SIGKDD. p. 3395–3404. KDD '20, New York, NY, USA (2020)
3. Deng, A., Hooi, B.: Graph neural network-based anomaly detection in multivariate time series. AAAI International Conference on Artificial Intelligence (2021)
4. Faber, K., Pietron, M., Zurek, D.: Ensemble neuroevolution-based approach for multivariate time series anomaly detection. Entropy **23(11)** (Nov 2021). https://doi.org/https://doi.org/10.3390/e23111466
5. Faber, K., Pietron, M., Zurek, D.: Ensemble neuroevolution-based approach for multivariate time series anomaly detection. Entropy **23**(11), 1466 (2021)
6. Frankle, J., Dziugaite, G., Roy, D., Carbin, M.: The Lottery Ticket Hypothesis at Scale (March 2019)
7. Garg, A., Zhang, W., Samaran, J., Savitha, R., Foo, C.S.: An evaluation of anomaly detection and diagnosis in multivariate time series. IEEE Transactions on Neural Networks and Learning Systems pp. 1–10 (2021). https://doi.org/10.1109/TNNLS.2021.3105827
8. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network (2015)
9. Hundman, K., Constantinou, V., Laporte, C., Colwell, I., Soderstrom, T.: Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. Proceedings of the 24th ACM SIGKDD p. 387–395 (2018)
10. Pietron, M., Wielgosz, M.: Retrain or not retrain? – efficient pruning methods of deep CNN networks (2020)
11. Pietron, M., Zurek, D., Faber, K., Corizzo, R.: Ad-nev: A scalable multi-level neuroevolution framework for multivariate anomaly detection. arXiv preprint arXiv:2305.16497 (2023)
12. Pietron, M., Zurek, D., Sniezynski, B.: Speedup deep learning models on GPU by taking advantage of efficient unstructured pruning and bit-width reduction, vol. 67. Elsevier (Mar 2023). https://doi.org/https://doi.org/10.1016/j.jocs.2023.101971
13. Ren, Z., Li, X., Peng, J., Chen, K., Tan, Q., Wu, X., Shi, C.: Graph autoencoder with mirror temporal convolutional networks for traffic anomaly detection. Scientific reports **14**(1), 1247 (2024)
14. Renda, A., Frankle, J., Carbin, M.: Comparing fine-tuning and rewinding in neural network pruning (2020)