2023 IEEE 23rd International Symposium on Computational Intelligence and Informatics (CINTI) | 979-8-3503-4294-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/CINTI59972.2023.10381935

NiaNetAD: Autoencoder architecture search for tabular anomaly detection powered by HPC

Sašo Pavlič
University of Maribor
Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, 2000 Maribor
Slovenia
Email: saso.pavlic@student.um.si

Sašo Karakatič
University of Maribor
Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, 2000 Maribor
Slovenia
Email: saso.karakatic@um.si

Iztok Fister Jr.
University of Maribor
Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, 2000 Maribor
Slovenia
Email: iztok.fister1@um.si

***Abstract*—In the era of Industry 4.0, predictive maintenance (PdM) has become increasingly important for ensuring the efficient and reliable operation of machinery. However, the complexity and diversity of industrial datasets acquired from these sensors can make it challenging to detect anomalies and predict when preventative maintenance is necessary. The proposed NiaNetAD (Nature-Inspired Algorithms for Deep Neural NeEtwork creaTion for Anomaly Detection) method addresses this challenge by using nature-inspired algorithms (NIAs) to construct an auto-encoder (AE) neural network, which identifies anomalies in the machine's operation. The anomalies in this case refers to unusual patterns in the machine state, which could indicate an eventual breakdown. By using NiaNetAD on industrial datasets, it can help identify potential failures and schedule maintenance before they occur, reducing downtime, improving equipment lifespan, and lowering maintenance costs. The results of the NiaNetAD experiment, which was conducted on a high-performance computing (HPC) platform with multiple NIAs, demonstrated the significance of an effective and dispersed search strategy when constructing an AE architecture for a specific task. When applied to unsupervised anomaly detection (AD) on a fault detection dataset, the results indicate that improving the AE's reconstruction loss, reducing the bottleneck size, and avoiding excessive complexity in the topology can result in better outcomes.**

***Index Terms*—predictive maintenance, autoencoder, anomaly detection, nature-inspired algorithms, optimization, high-performance computing, unsupervised learning**

I. Introduction

A. Current situation in Predictive maintenance

PdM is a technique that utilizes data from industrial production processes to predict machine failures and recommend preventive maintenance [1]. The system analyzes sensor data to detect abnormal machine behavior, allowing maintenance teams to schedule repairs before breakdowns occur [2]. This approach reduces costs associated with downtime and increases overall efficiency [3]. Machine learning techniques like random forest, artificial neural networks (ANNs), and support vector machines are commonly used for PdM, with vibration signal data being the preferred input for AD [4], [5]. However, challenges persist in automating outlier detection using neural architecture search (NAS) methods, including defining the search space, addressing unsupervised settings, handling imbalanced data, and ensuring sample efficiency [6]–[9].

This research was funded by the Slovenian Research Agency (research core funding No. P2-0057)

B. Our approaches to ongoing challenges

We enhance NiaNet [10] to find optimal AE architectures for detecting anomalies in fault detection datasets. Using evolutionary algorithms, we minimize the need for raw data analysis by identifying outliers. Our approach employs a multi-objective function to reduce reconstruction loss and complexity, enabling NiaNetAD to adapt automatically to diverse tabular datasets with minimal human intervention.

We conduct comprehensive experiments on fault detection datasets, comparing NiaNetAD's performance with a random classifier. Our contributions include extending previous work for AD in tabular datasets, applying NiaNetAD to a real-world dataset to obtain optimal AE architectures and threshold values, and proposing objective function formulation and experiment variables to evaluate the method's performance with multiple NIAs.

The remaining structure of this paper is as follows. Section II briefly describes the related works. Further information on challenges in PdM can be found in Section III. Section IV presents the proposed method NiaNetAD. Obtained results of the experiment are presented in Section V. The last Section VI contains the conclusion and future steps.

II. Related works

In Section II, we focus on summarizing strategies for automatically designing ANN architectures, known as NAS or neuroevolution techniques, specifically applied to AD problems. Anomaly detection presents distinct challenges and requirements that need effective differentiation between anomalous and normal data.

Jiao et al. [11] addressed challenges in NAS systems for outlier detection with limited data samples by introducing the AutoOD method. The authors employed a self-imitation learning-based experience replay mechanism to enhance sample efficiency and overcome issues like local optimality and

unfair bias. Their proposed framework was evaluated against random search on benchmark datasets.

Termritthikun et al. [9] proposed NAS techniques using MultiObjective Genetic Algorithm and Firefly Algorithm to develop an efficient CNN model for resource-constrained devices. Their experiments on various datasets, including crack detection datasets, showed reduced processing time and parameters with comparable classification accuracy, suggesting suitability for deployment on such devices.

III. Challanges of Predictive maintenance

One major challenge in PdM for equipment is managing large, complex datasets with increasing volume, leading to overfitting and poor model performance. The high dimensionality and sparsity of data further hinder identifying relevant patterns.

Non-stationary data, which changes over time, poses difficulties for traditional AD methods designed for stationary data. Integrating and pre-processing data from various sources, such as sensor data, maintenance records, and weather data, adds complexity to the task.

Interpretability is a significant issue as complex models like deep learning lack transparency, making it hard for domain experts to understand predictions and identify important features. Lastly, real-time PdM demands processing vast data volumes promptly to prevent equipment failure [1], [4], [5].

A. Architecture for anomaly detection

A range of DNN architectures can be used for AD on datasets. Serradilla et al. [12] mention three commonly used DNN architectures for this task: AE, CNN, RNN, and GAN. Combinations like VAE and LSTM can also be employed. In our research, we focused on the challenge of extracting meaningful patterns from high-dimensional data with few instances. We employed an AE model, which effectively reduces dimensionality [13]. AE architectures require fewer resources than CNN architectures for dimensionality reduction. To handle limited data instances, we used NIA's to discover the optimal AE architecture. The ideal AE architecture should be simple enough to capture patterns but not overly simplistic to avoid overfitting.

IV. Proposed method

We introduce NiaNetAD [1], an extension of our previous work [10], aimed at detecting anomalies in real-world datasets. NiaNetAD builds on our designed system, which identified topology and hyper-parameters for efficient encoding in an AE model. The proposed method utilizes a learning-based system to automatically find the most effective AE model for AD. It determines an optimal threshold value to distinguish normal and anomalous data instances. Additionally, we employ the PyTorch Lightning framework [14] to ensure adaptability across various hardware infrastructures, such as GPUs or HPC clusters, enabling exploration of a broader search space in less time.

[1]https://github.com/SasoPavlic/NiaNetAD

A. Pipeline Overview

The NiaNetAD method operates through a pipeline consisting of several key components, as shown in Fig. 1. The user provides a tabular dataset and initial search strategy parameters as inputs. NiaNetAD is initialized along with its main dependency, the NiaPy [15] framework for solution optimization. Solutions generated are transformed into working AE models, which are evaluated iteratively until a user-defined stopping criterion, like a specified number of iterations or performance metrics, is met. The optimal solution is then transformed into a final AE model and tested on a separate dataset to obtain reconstruction losses for each sample. These losses are used to determine the optimal threshold value, which is utilized in the AD process.

B. Representing individuals

The search space of the NiaNetAD method is bounded by an AE-shaped topology and a list of hyper-parameters. We propose the objective function formulation and a collection of nature-inspired algorithms to guide the search strategy direction away from falling into the curse of local optimum. With this search set, we tend to discover global optimum AE architectures for unsupervised AD in tabular datasets. Before moving forward, we need to understand what the solution array is. This is explained in the equation 1.

$$\chi_i^{(j)} = \left\{ x_{i,0}^{(j)}, \ldots, x_{i,n}^{(j)} \right\}, \text{for } i = 0, \ldots, N_p - 1 \tag{1}$$

Where each element of the solution is in the interval $\chi_{i,1}^{(j)} \in [0, 1]$. Real values in interval are then mapped according to equations [2-6], where y_1 stands for topology shape, y_2 layer neuron delta, y_3 for number of layers, y_4 for activation function, y_5 for number of epochs, y_6 for learning rate, y_7 for optimizer algorithm.

$$y_1, y_4, y_7 = \lfloor x[i] \rfloor; y_1 \in [0, 1] \tag{2}$$

$$y_2 = \lfloor \frac{x[i]}{features} \rfloor; y_2 \in [0, features] \tag{3}$$

$$y_3 = \lfloor \frac{x[i]}{maxLayers} \rfloor; y_{33} \in [0, maxLayers] \tag{4}$$

$$y_5 = \lfloor x[i] \cdot 10 + 100 \rfloor; y_5 \in [100, 200] \tag{5}$$

$$y_6 = \lfloor \frac{x[i]}{1000} \rfloor; y_6 \in [10^{-3}, 10^{-0}] \tag{6}$$

As in the NiaNet paper, the solution array $A = [x_1, x_2, x_3, x_4, x_5, x_6, x_7]$ is partitioned into two segments. The first three elements dictate the topology, while the last four specify the hyper-parameters, as seen in Fig. 2.

After the search strategy algorithm generates an array of continuous solutions, the encoding process begins. This process maps all variables, [y_1-y_7], using equations [2-6] by applying the binning process to each variable individually.

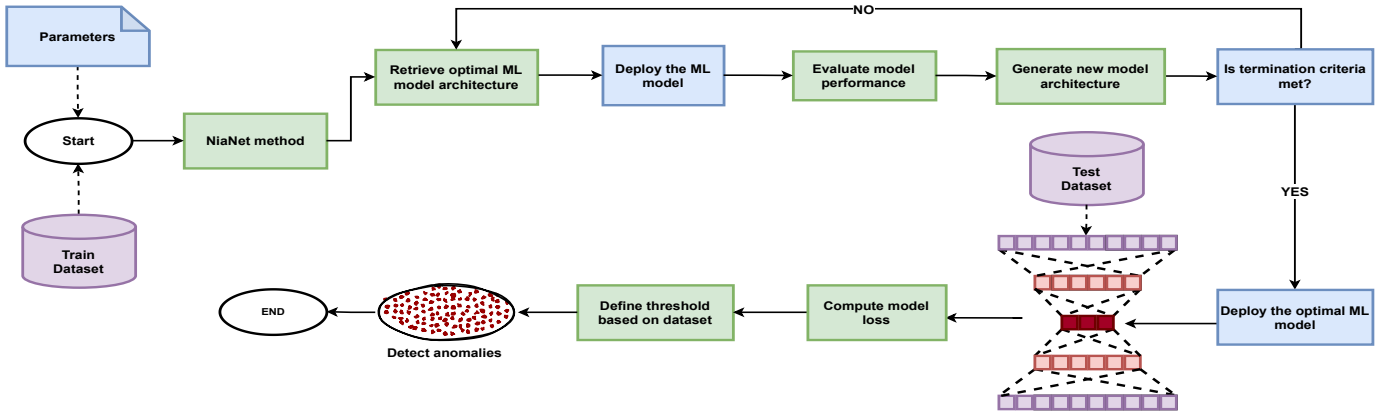

Fig. 1: An high-level overview of the NiaNetAD pipeline showcases the process of initiating a search process based on the dataset and initial parameters. As the search process undergoes iterations, the optimal solution is determined and mapped onto a functioning AE model for AD purposes.

To start, the binning process creates a specified number of bins within the interval of $\in [0, 1]$. The number of bins is determined by the number of possible values a single variable can hold. Next, the binning process takes each element value and maps it to a PyTorch [16] object, effectively encoding the continuous solution into a discrete format.

C. Fitness function

The fitness function, as expressed in equation 9, is utilized to evaluate the solutions generated during the evolutionary exploration of the search space. Two sub-functions, as presented in equations [7-8], are employed to find the optimal balance between the reconstruction error and the complexity of the model.

$$RMSE = \alpha_1 \cdot \sqrt{\frac{1}{N}\sum_{i=1}^{N} |x_i - \hat{x_i}|^2} \tag{7}$$

Where $RMSE$ represents the reconstruction error of the AE model and α_1 is weighting constant.

$$C = \frac{(y_5)^{\alpha_2} + (y_3 \cdot \alpha_3) + (bottleneck\_dim \cdot \alpha_4)}{\alpha_5} \tag{8}$$

Where C represents the topology complexity of the AE model and α_2,α_3,α_4,α_5 are weighting constants.

$$f(\chi_i^{(j)}) = \min E + C \tag{9}$$

Where $f(\chi_i^{(j)})$ represents the fitness value of an individual in evolution with goal to find the global minimum.

D. Detecting anomalies

Applications for AEs can be found across industries [17]–[19]. They are also effective for discovering anomalies in datasets. The method for AD using an AE assumes that the model learns to represent the majority of similar data well, but not as well for dispersed and abnormal minority samples. To detect anomalies, a trained AE model is utilized. Data is loaded in batches, and the model calculates the Root Mean Square Error (RMSE) for each sample. Anomalies are identified by RMSE values exceeding a set threshold. This threshold is determined by analyzing the distribution of reconstruction errors and setting a value higher than the population mean.

E. Algorithm pseudo-code

In order to understand what was changed from our previous work, we will first look into the pseudocode of NiaNetAD, which extends the functionalities. In this updated version, the main difference, is the capability to use the designed AE models for detecting anomalies in the dataset and setting the optimal threshold value. As we can see the Alg. 1.

Algorithm 1 Proposed method

Input: Tabular dataset, Search strategy parameters
Output: Abnormal data instances

```
NiaNetAD.init(parameters)
while terminationConditionNotMet do
    solution ← NiaNetAD.getBestSolution()
    AE ← Autoencoder(solution)
    AE.shape ← AE.mapShape()
    AE.layerStep ← AE.mapLayerStep()
    AE.layers ← AE.mapLayers(
    AE.activation ← AE.mapActivation()
    AE.epochs ← AE.mapEpochs()
    AE.LR ← AE.mapLearningRate()
    AE.optimizer ← AE.mapOptimizer()
    fitness ← TrainEvaluate(AE, dataset)
    NiaNetAD.generateNewSolution(fitness)
end while
AE ← NiaNetAD.getOptimalModel()
RMSE ← CalculateLoss(AE.predict(dataset) - outputs)
threshold ← RMSE.mean()
return dataset[RMSE > threshold]
```

V. Experiment and Results

The purpose of experimental work is to determine whether the proposed method is capable of designing competitive AE models for AD on fault detection datasets. We conducted the experiments to answer the following research questions:

1) Q1: Does lower reconstruction error provide better AD results?
2) Q2: Does increasing the number of algorithm iterations result in higher AUC score?
3) Q3: How effective is the fittest AE model, proposed by NiaNetAD compared with a random classifier?
4) Q4: How effective is the proposed search strategy when compared to random search?

A. Dataset Overview

In this section, we present the dataset used as input for our method, NiaNetAD, which proposes an optimal AE model. The dataset was chosen to address a real-world practical problem and consists of data samples from a sewage treatment plant in Istanbul. The data was collected using sensors to monitor bearing movement, capturing temperature and vibration. These sensors were mounted on both the pump's driving end (DE) and non-driving end (NDE) bearings. The dataset contains 60 features, 2152 instances, and is divided into two classes: normal (1221 instances) and anomalous (931 instances).

The dataset contains discrete numerical data representing each feature. It includes original temperature and vibration values, along with additional temporal features (TF). The TFs were extracted and added to the dataset using standard statistical measures such as mean, standard deviation (SD), range, skewness, kurtosis, maximum, minimum, and various percentiles (Q95, Q90, Q80). These temporal features enhance the fault detection dataset. The addition of these temporal features is a common practice in the field of fault detection [20]–[22]. The extracted features serve to provide valuable insights into the behavior of the system and enhance the accuracy of the fault detection process.

B. Enviromental setup

The proposed method, NiaNetAD, requires specific software and hardware prerequisites for implementation. We used various Python libraries, including NiaPy for optimizing solutions with NIAs, Scikit-learn for DNN model evaluation [23], NumPy for array operations [24], PyTorch for DNN initialization [16], and PyTorch Lightning for distributed training [14]. For hardware, we utilized a HPC with 6 dual-processor compute nodes, each equipped with 4 Nvidia V100 GPUs. Each node had 32 GB of GPU memory and ran on Rocky Linux 8.5 (Green Obsidian) [25]. This configuration enabled efficient search space exploration and faster experiment execution.

C. Experimental settings

In NiaPy experiments, we controlled various parameters: *DIM = 7* for dimensionality, *POP = default* for population size, and *ME = 1000* for maximum function evaluations. We used a single repetition *R = 1* and set LB = 0.0 and UB = 1.0 for lower and upper search space bounds.

The activation functions list included: *ELU*, *RELU*, *Leaky RELU*, *RRELU*, *SELU*, *CELU*, *GELU*, *Tanh*. Optimizers list contained: *Adam*, *Adagrad*, *SGD*, *RAdam*, *ASGD*, *Rprop*. Fixed training parameters: *batch size = 10*, data split ratio *65:25:10*, early stopping *patience = 50*, *min delta = 0.00*.

Weighting constants: $\alpha_1 = 1000$, $\alpha_2 = 2$, $\alpha_3 = 100$, $\alpha_4 = 10$, and $\alpha_5 = 100$.

D. Results

This section provides an overview of the findings from experimenting with our NiaNetAD method. The aim is to determine the key factors contributing to a robust AE model for AD, such as architecture, model complexity, and reconstruction error. Additionally, we aim to understand how the NiaNetAD method improves its models over the course of multiple iterations.

Duration of Exploration: Building upon the environmental setup described in the preceding subsection, the evaluation was carried out using specific control parameters. The total execution time under these experimental conditions amounted to 44 GPU days.

Results by the optimization algorithm: Our examination revealed the following outcomes. The top solutions produced by PSO, DE, FA, jDE, and GA algorithms are presented in Table I. The FA algorithm reached the lowest fitness value (minimization problem), followed by the DE and PSO algorithm. The GA and jDE algorithms reached significantly higher fitness scores. Upon conducting our experiments, we observed a lack of correlation between the AUC scores and the results obtained from our proposed fitness function. Specifically, the solutions identified as optimal by the AUC scores did not correspond to the lowest fitness values within the population. This observation leads us to conclude that an AE with a minimal RMSE and simple complexity may not be sufficient for effectively detecting anomalies in complex systems. With these findings, we concluded that the answer to the research question **Q1** is false.

To answer the question **Q2**, the Mann-Kendall test [26] was used to determine if there was a trend in the rolling statistics of the AUC metric over the iterations of the algorithms. The mean performance was calculated, and it was found that the PSO, DE, and jDE algorithms were gradually finding better solutions, while the FA and GA algorithms were showing a decreasing trend. Additionally, the SD metric was calculated to examine the stability of the search strategy. It was found that the DE and jDE algorithms were becoming more stable over time, while the PSO, FA, and GA algorithms were becoming less stable.

Fittest AE architecture: Following is the fittest AE architecture discovered by the NiaNetAD method. Among the optimization algorithms listed in Table I the DE algorithm discovered the most optimal topology and hyper-parameters for a given dataset. The Fittest AE architecture solution was discovered after 504 iterations of the search process, and it

TABLE I: Performance comparison of optimal solutions discovered by NiaNetAD using selected algorithms, as measured by high AUC scores.

Algorithm	AUC	Fitness value	RMSE	Bottleneck size	Layers in AE	Epochs	Activation function	Learning rate	Optimizer algorithm
DE	0.84	2116	1.92	17	2	140	ELU	0.07	Adagrad
PSO	0.83	2345	2.11	31	2	150	ELU	0.12	Adagrad
FA	0.81	1649	1.28	28	2	190	RRELU	0.01	RAdam
GA	0.79	2829	2.56	53	2	160	RELU	0.10	Adagrad
jDE	0.77	3751	3.62	7	2	110	SELU	0.48	Adagrad

reached the value of 0.84 in the AUC metric on a test dataset. The AE topology was mapped from DE's proposed solution based on the encoding strategy. The AE model was split into encoder and decoder parts, where the bottleneck size was 17. Both the encoder and decoder were mirrored versions of each other. If the encoder was encoding the 60-D input vector to the 17-D latent vector, then the decoder was decoding the latent vector back to the 60-D output vector. The selected bottleneck size was good enough to generalize input data into lower dimensional representation and then to decompress normal data instances with lower reconstruction error than anomalous ones.

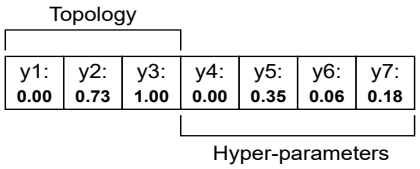

Fig. 2: A representation of the allocation of solution array indices [y_1-y_7] to variables, which are later utilized to map it to a PyTorch model.

The **topology** of the AE was based on the first three elements of the solution array. The derived AE model was divided into two distinct components, an encoder, and a decoder, with the bottleneck size set at 17. This can be seen in Fig. 3. Both the encoder and decoder were symmetric in design, with the encoder compressing a 60-D input vector into a 17-D latent vector, and the decoder decompressing the latent vector back into a 60-D output vector. The encoder and decoder both utilized a *Linear* layer, which is a type of fully connected layer commonly used in neural networks.

The final four elements of the solution array were utilized to map the **hyper-parameters** for the AE model. These were set as follows: activation function = *ELU*, epochs = 140, learning rate = 0.07, and optimization algorithm = *Adagrad*. The solution array element values are visible on Fig. 2.

Achieved performance: The selected bottleneck size was considered sufficient for effectively generalizing input data into a lower-dimensional representation, which resulted in decompressing normal data instances with a lower RMSE mean of 14.58, as compared to anomalous instances which had a higher mean RMSE of 17.90. Additionally, normal instances displayed an RMSE with SD of 2.46, while anomalous instances had a significantly higher SD of 7.51. Furthermore, we can view the RMSE and MSE scores for the entire train and test datasets in Table II.

TABLE II: Performance comparison for RMSE, MSE, and Accuracy metrics on all data points in the train and test datasets.

	Train	**Test**
RMSE	4.71	4.82
MSE	1.39	1.40
Acc	0.80	0.79

The dataset used to train the AE model had a relatively balanced distribution of anomalous and normal instances, which made it challenging to train the AE model effectively using an unsupervised technique. Despite this, the AE model was able to reach an AUC of 0.84, indicating a decent performance as illustrated in Fig. 4. The accuracy of the model on the training and testing datasets was found to be 0.80 and 0.79 respectively as seen in Table II when using the optimal threshold *FPR* = 0.24 and *TPR* = 0.84. The trapezoidal rule [27] returned the value of 15.6 for this threshold. This finding gives an answer to question **Q3**, on how effective is the fittest AE model in comparison with the random classifier.

Following are the findings to a question **Q4**. The findings demonstrate that not only does the top model produced by our NiaNetAD method outperform that of a random search, but the improvement in the average performance of the top models is much more pronounced. This demonstrates that NiaNetAD more effectively and efficiently discovers high-performing models compared to a random search. Additionally, there is a noticeable upward trend in the average performance of NiaNetAD over time, which is not present in the random search results. This suggests that our search controller continually improves and refines its strategies based on past search experiences, whereas the random search has a limited ability to uncover optimal models. Lastly, the SD of the search process is significantly reduced when using our search strategy, compared to the random search, which indicates a more consistent and stable search process.

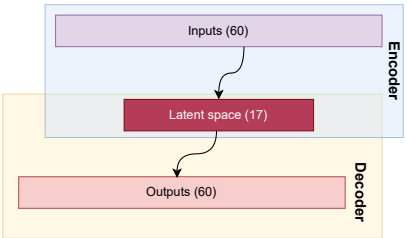

Fig. 3: The fittest topology discovered. Symmetric encoder and decoder, compressing 60-D input to 17-D latent vector and decompressing it back to 60-D output.

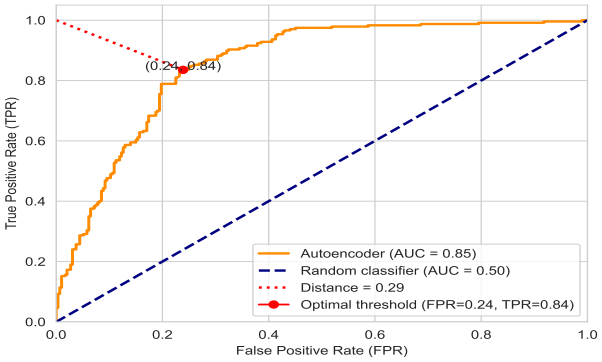

Fig. 4: Optimal AE model performance for AD on fault detection dataset: AUCROC metric calculated using FPR and TPR metrics with optimal threshold value.

VI. Conclusion

In conclusion, the suggested NiaNetAD approach offers a potentially effective way to handle the problems associated with PdM in the era of Industry 4.0. It can effectively recognize irregularities in the operation of machinery, which can assist predict probable breakdowns and planning maintenance before they happen. This is accomplished by employing NIAs to build an AE neural network. As a result, there may be less downtime, longer equipment life, and cheaper maintenance expenses.

In general, the utilization of NiaNetAD for PdM of machinery sensors can improve the functioning of machines by ensuring that they are consistently operating at optimal levels, reducing the likelihood of unanticipated breakdowns, enhancing productivity, and providing economic benefits.

Going forward, our goal is to further test our method using well-established benchmark datasets and compare its performance with other competitive methods. This will give us a deeper understanding of the strengths and areas for improvement of our approach.

References

[1] T. Zonta, C. A. da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li, "Predictive maintenance in the industry 4.0: A systematic literature review," vol. 150, p. 106889. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835220305787

[2] S. Namuduri, B. N. Narayanan, V. S. P. Davuluru, L. Burton, and S. Bhansali, "Review—deep learning methods for sensor based predictive maintenance and future perspectives for electrochemical sensors," vol. 167, no. 3, p. 037552, publisher: IOP Publishing. [Online]. Available: https://dx.doi.org/10.1149/1945-7111/ab67a8

[3] E. Florian, F. Sgarbossa, and I. Zennaro, "Machine learning-based predictive maintenance: A cost-oriented model for implementation," vol. 236, p. 108114. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925527321000906

[4] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. d. P. Francisco, J. P. Basto, and S. G. S. Alcalá, "A systematic literature review of machine learning methods applied to predictive maintenance," vol. 137, p. 106024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835219304838

[5] J. Dalzochio, R. Kunst, E. Pignaton, A. Binotto, S. Sanyal, J. Favilla, and J. Barbosa, "Machine learning and reasoning for predictive maintenance in industry 4.0: Current status and challenges," vol. 123, p. 103298. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361520305327

[6] H. T. Ünal and F. Başçiftçi, "Evolutionary design of neural network architectures: a review of three decades of research," vol. 55, no. 3, pp. 1723–1802. [Online]. Available: https://doi.org/10.1007/s10462-021-10049-5

[7] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions." [Online]. Available: http://arxiv.org/abs/2006.02903

[8] Y. Li, Z. Chen, D. Zha, K. Zhou, H. Jin, H. Chen, and X. Hu, "AutoOD: Neural architecture search for outlier detection," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2117–2122, ISSN: 2375-026X.

[9] C. Termritthikun, L. Xu, Y. Liu, and I. Lee, "Neural architecture search and multi-objective evolutionary algorithms for anomaly detection," in *2021 International Conference on Data Mining Workshops (ICDMW)*, pp. 1001–1008, ISSN: 2375-9259.

[10] S. Pavlič, I. F. Jr, and S. Karakatič, "NiaNet: A framework for constructing autoencoder architectures using nature-inspired algorithms," in *Annals of Computer Science and Information Systems*, vol. 30, pp. 109–116, ISSN: 2300-5963. [Online]. Available: https://annals-csis.org/Volume_30/drp/192.html

[11] Y. Jiao, K. Yang, D. Song, and D. Tao, "TimeAutoAD: Autonomous anomaly detection with self-supervised contrastive loss for multivariate time series," vol. 9, no. 3, pp. 1604–1619, conference Name: IEEE Transactions on Network Science and Engineering.

[12] O. Serradilla, E. Zugasti, J. Rodriguez, and U. Zurutuza, "Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects," vol. 52, no. 10, pp. 10 934–10 964. [Online]. Available: https://doi.org/10.1007/s10489-021-03004-y

[13] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. JMLR Workshop and Conference Proceedings, pp. 37–49, ISSN: 1938-7228. [Online]. Available: http://proceedings.mlr.press/v27/baldi12a.html

[14] W. Falcon and The PyTorch Lightning team, "PyTorch lightning." [Online]. Available: https://github.com/Lightning-AI/lightning

[15] G. Vrbančič, L. Brezočnik, U. Mlakar, D. Fister, and I. Fister, "NiaPy: Python microframework for building nature-inspired algorithms," *Journal of Open Source Software*, vol. 3, no. 23, p. 613, Mar. 2018. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.00613

[16] PyTorch. [Online]. Available: https://www.pytorch.org

[17] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," pp. 241–246. [Online]. Available: http://arxiv.org/abs/1608.04667

[18] T.-H. Song, V. Sanchez, H. EIDaly, and N. Rajpoot, "Hybrid deep autoencoder with curvature gaussian for detection of various types of cells in bone marrow trephine biopsy images."

[19] A. Radhakrishnan, K. Yang, M. Belkin, and C. Uhler, "Memorization in overparameterized autoencoders." [Online]. Available: http://arxiv.org/abs/1810.10333

[20] Y. Liu, Z. Pang, M. Karlsson, and S. Gong, "Anomaly detection based on machine learning in IoT-based vertical plant wall for indoor climate control," vol. 183, p. 107212. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360132320305837

[21] D. B. Araya, K. Grolinger, H. F. ElYamany, M. A. M. Capretz, and G. Bitsuamlak, "Collective contextual anomaly detection framework for smart buildings," in *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 511–518, ISSN: 2161-4407.

[22] W. Elmasry and M. Wadi, "Enhanced anomaly-based fault detection system in electrical power grids," vol. 2022, p. e1870136, publisher: Hindawi. [Online]. Available: https://www.hindawi.com/journals/itees/2022/1870136/

[23] "scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation." [Online]. Available: https://scikit-learn.org/stable/

[24] "NumPy." [Online]. Available: https://numpy.org/

[25] Rocky linux. [Online]. Available: https://rockylinux.org/

[26] M. Kendall, "Rank correlation methods. 2nd impression," *Charles Griffin and Company Ltd. London and High Wycombe*, 1975.

[27] W. L. England, "An exponential model used for optimal threshold selection on ROC curues," vol. 8, no. 2, pp. 120–131, publisher: SAGE Publications Inc STM. [Online]. Available: https://doi.org/10.1177/0272989X8800800208