Machine Learning in Applications Detection of Anomalous Behaviour in Industrial Robot Project Report

Federica Lupoli s318018, Edoardo Franco S310228, Bordero Giovanni s313010
Politecnico di Torino

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Machine Learning in Applications Detection of Anomalous Behaviour in Industrial Robot Project Report

Abstract—**This study investigates the application of Adversarial AutoEncoders (AAEs) for anomaly detection in Kuka industrial robots, comparing their performance to traditional AutoEncoders (AEs). We utilized time-series sensor data from robot operations, encompassing both normal and anomalous behavior. A grid search like approach was conducted to optimize hyperparameters, including window sizes, learning rates, and training epochs. Both models demonstrated high efficacy in anomaly detection, with the AAE achieving a quite perfect peak in F1.5 score under optimal conditions. Contrary to initial expectations, we found that larger window sizes did not necessarily improve performance, suggesting the importance of capturing relevant temporal patterns within a concise context. Additionally, feature-wise thresholding techniques were utilized to distinguish between normal and anomalous behavior based on reconstruction error, yielding insightful results. Our findings indicate that AAEs may offer slight advantages over AEs in certain configurations, potentially due to enhanced stability and generalization capabilities from adversarial training.**

I. Introduction

In modern industrial settings, the reliability and precision of robots are crucial for maintaining high productivity and quality standards. Kuka industrial robots, widely used across various industries, rely on complex, synchronized movements driven by sophisticated sensor data. However, these robots are not immune to issues arising from configuration errors or natural wear and tear, which can result in slower or less precise movements. Detecting such anomalies early can prevent significant downtime and costly repairs.

This project focuses on implementing an adversarial AutoEncoder (AAE) for anomaly detection using a dataset derived from a Kuka industrial robot. The dataset encompasses time-series data from the robot's sensors, capturing joint angle positions, velocities, currents, and power usage values. The primary challenge is to identify deviations in the robot's performance that can indicate potential anomalies.

An adversarial AutoEncoder is a machine learning model that combines the capabilities of an AutoEncoder with those of an adversarial network. The AutoEncoder component is trained to learn the normal operational patterns of the robot by encoding and decoding the sensor data. The adversarial component aims to improve this process making the encoded representations more robust, potentially improving the model's ability to detect anomalies.

The goal of this study is to evaluate whether incorporating an adversarial component into a traditional AutoEncoder framework improves the performance and reliability of anomaly detection. This involves not only developing and training the AAE model but also determining the most suitable metrics to evaluate its performance. Through rigorous experimentation and analysis, the project aims to provide insights into the effectiveness of adversarial AutoEncoders in enhancing anomaly detection for industrial robots.

II. Background

Anomaly detection in time-series data is a crucial task that has gathered significant attention due to its applications in various domains such as cybersecurity, finance, and healthcare. A typical anomaly detection (AD) setting assumes that only normal data are accessible during the training phase, making unsupervised methods particularly appropriate. These methods train models to learn patterns from normal signals, with the goal of assigning low anomaly scores to normal inputs and high scores to abnormal ones. [1] [2]

AutoEncoders (AEs) have emerged as a key reconstruction-based method, leveraging their ability to learn compressed representations of data. In time-series contexts, AutoEncoders are trained to reconstruct input sequences, and anomalies are identified by their high reconstruction errors. Practical implementations, such as those by *Audibert et al. (2020)* [3] and *Torabi et al.* [4], demonstrate the effectiveness of AEs in various applications by focusing on vector reconstruction errors to identify anomalies.

Adversarial AutoEncoders (AAEs) build on this concept by incorporating adversarial training, where a Discriminator network works alongside the AutoEncoder to distinguish between real and reconstructed data. This approach enhances the model's ability to produce accurate reconstructions, thus improving anomaly detection performance. For instance, the *USAD* [3] method showcases the effectiveness of adversarial training in multivariate time-series anomaly detection, achieving robustness and scalability across different datasets.

Variational AutoEncoders (VAEs) introduce a probabilistic layer to the AutoEncoder framework, learning distributions over the latent space rather than fixed representations. Anomaly detection systems leveraging Variational AutoEncoders (VAEs) [5] [6] have shown remarkable potential for this task. Though typically used for classification, their generative nature also supports sampling. Additionally, the flexibility of their neural architecture allows for easy adaptation across various domains. VAE-based anomaly detection systems have demonstrated superior effectiveness and resilience, outperforming PCA-based methods [5], random forests [6], traditional deep neural networks, and one-class SVMs [7].

Generative Adversarial Networks (GANs) [8], particularly frameworks like *ALGAN (Adjusted-LSTM GAN)* [1], use a generator to produce synthetic data and a Discriminator to differentiate between real and synthetic data. GANs like *AL-GAN* leverage LSTM units to manage temporal dependencies effectively, addressing common issues such as vanishing gradients and mode collapse, thus improving anomaly detection in time-series data.

Transformer-based Models, such as the Anomaly Transformer [9], employ attention mechanisms to capture long-range dependencies and complex temporal patterns. However, transformers sometimes struggle with maintaining the order and continuity essential for time-series data, leading to the use of LSTM-based GANs in some scenarios for better temporal modeling.

These advancements highlight the ongoing evolution of time-series anomaly detection methodologies. The comparative study of adversarial AutoEncoders and other state-of-the-art techniques, underscores the continuous efforts to improve robustness and accuracy in detecting anomalies within complex, real-world time-series data.

III. Materials and methods

A. Dataset

For this study, we utilized a dataset collected from a Kuka industrial robot operating under different conditions. The dataset is composed of sensory and operational data recorded during both normal and anomalous operations.

The data is categorized into two subsets: *KukaNormal* and *KukaSlow*, each representing distinct operational states:

- *KukaNormal*: This subset includes 233,792 samples and captures the robot's behavior under normal operating conditions. Each sample is described by 86 features.
- *KukaSlow*: This subset contains 41,538 samples and reflects the robot's behavior under slowed operational conditions. Each sample in this subset is described by 87 features, where the last one is the label identifying the anomaly dataset.

Features
The features in the dataset are categorized into two main groups: those related to the robot's operational metrics and those obtained from various sensors attached to the robot. Below is a detailed description of the features:

Operational Metrics
These include power-related parameters such as apparent power, current, frequency, phase angle, power, power factor, reactive power, and voltage.

Sensor Data
These features are obtained from seven sensors (*sensor_id1* to *sensor_id7*) mounted on different parts of the robot. Each sensor records the following:

- Accelerometer readings (*AccX, AccY, AccZ*)
- Gyroscope readings (*GyroX, GyroY, GyroZ*)
- Quaternion data representing the sensor's orientation (*q1, q2, q3, q4*)
- Temperature (*temp*)

Test Set Preparation
For the test set, a combination of normal and anomalous data was used, with a focus on creating a diverse and challenging evaluation dataset. The preparation process involved the following steps:

- *Data Windowing*: Both normal and anomalous data were processed using a sliding window approach. This technique allows for the capture of temporal dependencies in the data.
- *Test Set Composition*: The test set was constructed by combining three types of data:
  - *Pure Anomalous Data*: Taken directly from the KukaSlow dataset.
  - *Pure Normal Data*: A subset of the KukaNormal dataset, not used in training.
  - *Mixed Data*: Created by accurately inserting anomalous segments into normal operation data.
- *Mixed Data Generation*: To create challenging test cases, anomalous segments were inserted into normal operation data. This process involved:
  - Selecting a subset of normal data.
  - Inserting short anomalous segments (10% of the window size) at random positions within each window.
  - Ensuring that inserted anomalies match the action type of the normal data they replace, maintaining operational context.
- *Data Balancing*: The test set was balanced to include equal proportions of pure anomalous, pure normal, and mixed data.
- *Shuffling*: The final test set was shuffled to ensure random distribution of different data types during evaluation.

This methodology ensures a comprehensive and challenging test set that evaluates the model's ability to detect both clear anomalies and subtle deviations in robot behavior, providing a robust assessment of the model's performance in real-world scenarios.

B. Models

We employed two models in our research: an AutoEncoder (AE) and an Adversarial AutoEncoder (AAE). These models are implemented using TensorFlow and are tailored to handle time-series data through the use of several 1D convolutional layers.

AutoEncoder
The AutoEncoder is composed of an Encoder and a Decoder.

The Encoder is tasked with mapping the input data into a compressed representation, referred to as the latent space. This process involves the transformation of high-dimensional input data into a lower-dimensional format. The Encoder, typically structured as a series of convolutional or fully connected layers, reduces the input dimensions through nonlinear mappings, thereby capturing the essential features and underlying structure of the data while discarding noise and redundancy.

As opposite, the Decoder's role is to reconstruct the original input data from this compressed latent space representation.

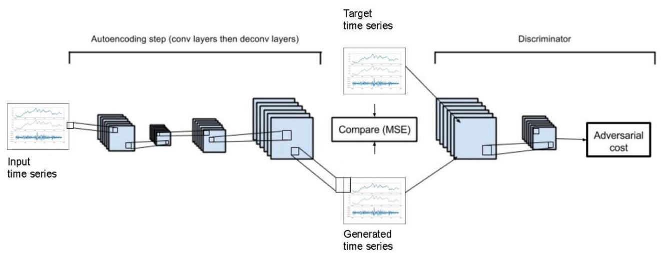

Fig. 1. Adversarial AutoEncoder Architecture

This reconstruction aims to be as accurate as possible, minimizing the loss of information that occurred during the encoding phase. The Decoder, mirroring the architecture of the Encoder, employs a sequence of layers that progressively upscale and refine the latent space representation back to the original input dimensions. The efficacy of the AutoEncoder is measured by the reconstruction error, which quantifies the difference between the input and the output.

For the Encoder, we employed a series of Conv1D layers with progressively increasing filter sizes (from 2 times the window size up to 32 times the window size), using varying kernel sizes and strides to capture temporal features effectively. Each Conv1D layer is followed by batch normalization to stabilize and accelerate training, and ReLU activation to introduce non-linearity.

The Decoder uses Conv1DTranspose layers, which are essentially the transposed counterparts of Conv1D layers, allowing the model to upsample and reconstruct the original input. The filter sizes decrease progressively in the Conv1DTranspose layers (from 16 times the window size down to the window size), and batch normalization is applied after each layer except the last. ReLU activation is also used after each layer except the final one, ensuring the reconstructed data retains the necessary temporal structure.

Adversarial AutoEncoder
The Adversarial AutoEncoder extends the AutoEncoder by incorporating a Discriminator, creating an adversarial training framework as represented in Figure 1.

The Discriminator is designed to distinguish between real and reconstructed data, thereby enhancing the AutoEncoder's ability to best reconstruct the input samples. It comprises Conv1D layers similar to those in the Encoder but also includes dense layers to process the flattened output of the convolutional layers. Each Conv1D and dense layer is followed by batch normalization to ensure stable training, and the dense layers use ReLU activation. The output layer employs a sigmoid activation to provide a probability score indicating the authenticity of the input.

The AAE training process involves minimizing the reconstruction loss to improve the AutoEncoder and maximizing the discrimination accuracy to enhance the Discriminator.

A custom training step was implemented, passing the real data X_{real} through the AutoEncoder and producing the reconstructed X_{fake}. The Discriminator is then used to evaluate both the real and the generated data, producing outputs $C(X_{real})$ and $C(X_{fake})$, respectively.

The AutoEncoder loss L_{AE} comprises two components: the adversarial loss and the reconstruction loss. The adversarial loss encourages the AutoEncoder to generate data that the Discriminator cannot distinguish from real data. This is mathematically expressed as:

$$L_{adv} = E_{X_{real}}[\max(0, 1 - C(X_{fake}))] \tag{1}$$

The reconstruction loss is measured using the mean squared error (MSE) between the real data and the generated data:

$$L_{rec} = E_{X_{real}}[\|X_{real} - X_{fake}\|_2^2] \tag{2}$$

Thus, the total AutoEncoder loss is given by:

$$L_{AE} = L_{adv} + L_{rec} \tag{3}$$

The Discriminator loss L_D is designed to maximize the difference between the Discriminator's outputs on real and generated data. It is expressed as:

$$L_D = E_{X_{real}}[\max(0, 1 - C(X_{real})) + \max(0, 1 + C(X_{fake}))] \tag{4}$$

C. Metrics

To evaluate the performance of our anomaly detection framework, we employed several metrics that are particularly suited to imbalanced datasets, which are common in anomaly detection scenarios where normal operations outnumber anomalous events.

Given the predictions and the true labels, we compute the following metrics:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

where TP, TN, FP, and FN represent True Positives (correctly identified anomalies), True Negatives (correctly identified normal operations), False Positives (normal operations incorrectly flagged as anomalies), and False Negatives (missed anomalies), respectively.

It's important to note that in anomaly detection, accuracy can be misleading due to class imbalance. A model that always predicts ”normal” could achieve high accuracy in a dataset where anomalies are rare, but would be useless for detecting actual anomalies. Therefore, we place more emphasis on the following metrics:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

Precision measures the proportion of correctly identified anomalies among all instances flagged as anomalous, while recall measures the proportion of actual anomalies that were correctly identified.

The F1 score, which is the harmonic mean of precision and recall, provides a balanced measure:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{8}$$

In the industrial context of Kuka robots, where missing an anomaly (false negative) could lead to costly production errors or equipment damage, we prioritize the avoidance of false negatives over false positives. However, we still maintain a good weight for false positives to minimize unnecessary maintenance checks. To achieve this balance, we employ the F$_\beta$ score with $\beta = 1.5$:

$$F_\beta = (1+\beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} \tag{9}$$

where $\beta = 1.5$ makes recall one and a half times as important as precision. This choice of β reflects our emphasis on minimizing missed anomalies in the industrial robot setting, where failing to detect a malfunction could have more severe consequences than triggering a false alarm.

Note that while we still compute accuracy, we do not heavily rely on it for model evaluation due to its potential to be misleading in imbalanced datasets. Instead, we focus on precision, recall, F1, and in particular on F1.5 score, which provide a more nuanced and reliable assessment of our model's performance in detecting rare anomalies in Kuka industrial robot operations.

IV. Results and discussion

A. Data Pre-Processing

Data pre-processing is a crucial step in preparing the dataset for training deep learning models. It involves several steps to ensure the data is clean, normalized, and properly structured for input into the model. In this study, the following steps were undertaken:

Normalization is applied to both the normal and anomalous datasets, ensuring that each feature in the dataset contributes equally to the learning process by scaling the data to a range between 0 and 1. Specifically, the normalization formula used is:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

where x represents the original feature values, and x' represents the normalized values. This step prevents features with larger ranges from dominating those with smaller ranges.

The data is also split into training and testing sets with a split ratio of 70:30 to ensure that the model has enough data to learn from while also being able to generalize well on unseen data.

Finally, after applying windowing, the dataset is shuffled to ensure that the training process is unbiased and that the model does not learn any unintended patterns from the order of the data.

B. Experiments

Our experimental setup was designed to evaluate and compare the performance of two anomaly detection models: an AutoEncoder (AE) and an Adversarial AutoEncoder (AAE). A grid search-like approach was employed in order to identify the optimal hyperparameters for each model.

Experimental Setup
We utilized two datasets, preprocessed using sliding windows, with window sizes ranging from 10 to 200 time steps, and step sizes set to one-quarter of the window size in order to exploit a trade-off between the total avoidance of overlapping and the computational costs associated with a step size of one. For each model architecture, we explored the following hyperparameter space:

- Window sizes: [10, 20, 30, 50, 100, 200]
- Learning rates: [10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}]
- Number of epochs: [2, 5, 10, 20, 30, 50]

We implemented early stopping with a patience of 3 epochs to prevent overfitting and a batch size of 256. The models were trained using the Adam optimizer with the specified learning rates and the number of steps per epoch derived from batch size and the train dataset shape, in order to fully cover the whole dataset in each epoch.

Evaluation Metrics
To assess model performance, we primarily used the F1.5 score, which places 1.5 times more importance on recall compared to precision. This choice reflects our preference for minimizing false negatives in anomaly detection while also keeping consideration of false positives. We also computed accuracy, precision, recall, and the standard F1 score for a comprehensive evaluation, as well as the area under the receiver operating characteristic curve (AUROC).

Threshold Selection
In order to discriminate between anomaly and normal behavior we used an thresholding approach based on reconstruction error. In particular, we marked as anomalies all those sample that have a reconsruction error greater than the selected threshold. We explored two threshold selection methods:

- Mean Threshold: We calculated the mean reconstruction error on the training data and set the threshold as the mean plus three times the standard deviations, following the common knowledge on outlier identification.
- Feature-wise Threshold: Starting from the idea of Tarobi et al. [4] we computed thresholds for each feature independently exploring various strategies, such as: mean plus three standard deviations, and the maximum between the mean plus one standard deviation and the maximum observed value for the considered feature.

Analysis
To gain deeper insights into model performance, we generated several visualizations:

1) Averaged reconstruction error plots to compare normal and anomalous data.
2) Loss distribution plots for training and test data, highlighting the separation between normal and anomalous samples.

3) Feature-wise loss plots to identify which features contribute most to anomaly detection.
4) Confusion matrices to visualize classification performance.
5) ROC curves to assess the trade-off between true positive and false positive rates.

These visualizations helped us understand the models' behavior and validate their effectiveness in distinguishing between normal and anomalous operations.

Results

For each configuration, we trained the models and evaluated their performance on the test set using multiple metrics. We considered accuracy (though it's less effective in unbalanced scenarios), recall, precision, F1, and F1.5 scores. AUROC was not used as a determining factor, as well-performing models consistently achieved a value of 1.00. The best configurations for each model were ultimately determined based on the highest F1.5 score achieved, which we found to be the most informative metric for our specific use case.

The AutoEncoder (AE) model demonstrated optimal performance with a learning rate (λ) of 10^{-3}, revealing also a nuanced relationship between window size (W) and model performance, as illustrated in Figure 2. Indeed contrary to initial expectations, we found that excessively large window sizes did not necessarily lead to improved results. Larger windows required substantially more training time and computational resources to achieve satisfactory performance, often without commensurate gains in the considered metrics.

The best-performing AE configuration utilized a window size of 10, trained for 5 epochs and reaching a F1.5 score of 0.9991, as shown in Table I. This configuration struck an optimal balance between capturing sufficient temporal context and maintaining computational efficiency. The current dimension of window size allowed the model to focus on the most relevant temporal patterns without being overwhelmed by extraneous information or noise present in larger windows.

These results underscore the AE's remarkable capacity to accurately reconstruct input windows, a critical ability for effective anomaly detection. By learning to faithfully reproduce normal operational patterns, the AE becomes highly sensitive to deviations that may indicate anomalies.

The Adversarial AutoEncoder (AAE) demonstrated remarkable efficacy in anomaly detection, with its performance significantly influenced by key hyperparameters: window size, learning rate, and number of epochs. Through extensive experimentation, we identified an optimal configuration that achieved exceptional results. This configuration, utilizing a window size of 20, a learning rate of 10^{-3}, and trained for 50 epochs, yielded an impressive F1.5 score of 0.9991. The corresponding performance metrics were equally notable, with an accuracy of 0.9983, precision of 0.9972, and a perfect recall of 1.0, resulting in an F1 score of 0.9986, as detailed in Table II..

Our investigation revealed that the AAE exhibits a nuanced relationship with window size, as depicted in Figure 2. While a window size of 20 consistently yielded superior performance, we observed a marked degradation in model efficacy when employing larger window sizes, particularly 100 and 200. This degradation was most pronounced in the model's recall, suggesting that the AAE encounters difficulties in capturing relevant patterns when presented with excessively long input sequences. This finding underscores the critical nature of appropriate window size selection in maximizing the AAE's anomaly detection capabilities.

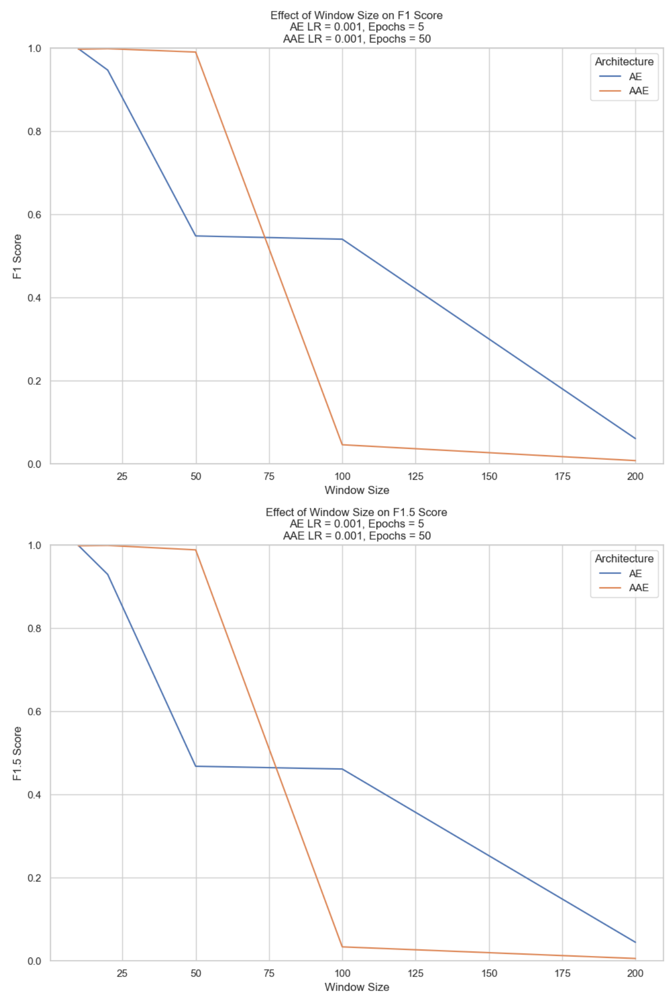

Fig. 2. Window size Tuning - Comparison between AE and AAE

The learning rate emerged as another crucial factor in optimizing AAE performance. Our experiments consistently demonstrated that a learning rate of 10^{-3} produced the most favorable outcomes. Deviations from this value, whether higher (10^{-2}) or lower (10^{-4}, 10^{-5}), resulted in sub-optimal performance. This observation highlights the importance of meticulous learning rate tuning in achieving peak AAE performance.

The number of training epochs exhibited a generally positive correlation with model performance, as illustrated in Figure 3. While the pinnacle of performance was attained at 50 epochs, it is noteworthy that significant improvements were already discernible at 10 epochs. This finding suggests the potential for striking an effective balance between computational efficiency and model performance by judiciously selecting the number of training epochs.

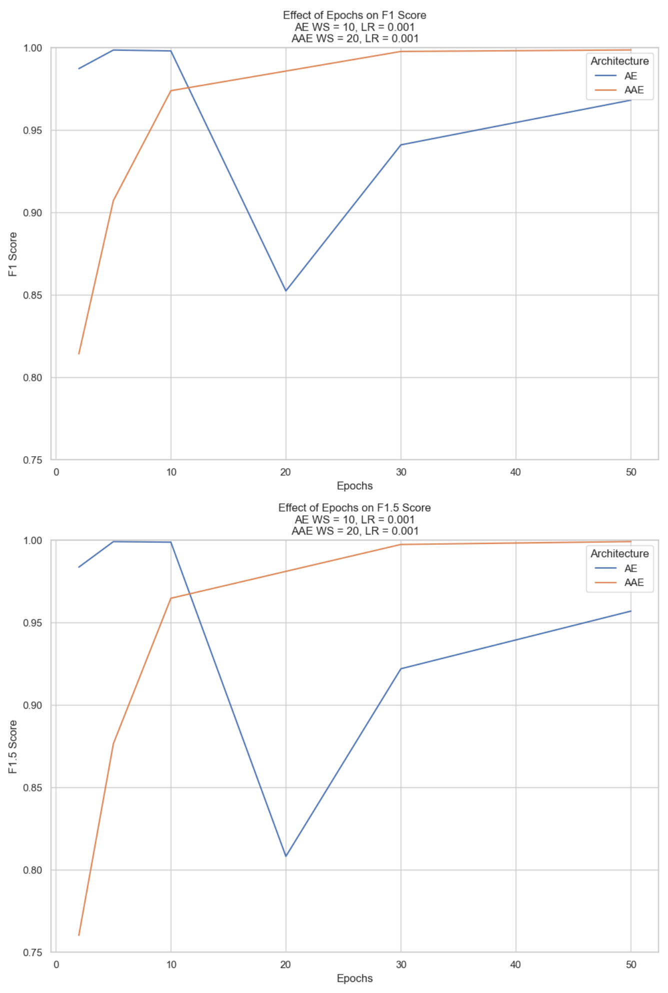

Fig. 3. Epochs Tuning - Comparison between AE and AAE

W	λ	Epochs	Acc. (%)	Prec. (%)	Recall (%)	F1 (%)	F1.5 (%)
10	10^{-2}	2	45.82	59.91	0.28	0.55	0.40
	10^{-3}		98.64	99.69	97.79	98.73	98.37
	10^{-4}		74.52	98.94	53.56	69.50	62.36
	10^{-5}		67.97	99.48	41.21	58.28	50.27
10	10^{-3}	2	98.64	99.69	97.79	98.73	98.37
20			88.35	99.78	80.52	89.12	85.60
30			62.20	98.60	37.87	54.73	46.73
50			61.69	98.04	37.63	54.38	46.43
100			40.61	100.00	2.84	5.52	4.05
200			41.43	100.00	4.54	8.68	6.42
10	10^{-3}	2	98.64	99.69	97.79	98.73	98.37
		5	99.84	99.71	100.00	99.86	99.91
		10	99.79	99.61	100.00	99.80	99.88
		20	84.47	99.39	74.62	85.24	80.82
		30	93.26	99.44	89.31	94.10	92.20
		50	96.29	99.85	93.96	96.82	95.70
10	10^{-3}	5	99.84	99.71	100.00	99.86	99.91
20			94.03	99.66	90.24	94.72	92.94
50			62.11	98.89	37.93	54.83	46.81
100			61.20	97.47	37.39	54.05	46.14
200			40.57	100.00	3.14	6.10	4.48

TABLE I
COMPREHENSIVE COMPARISON FOR AE MODEL

A distinguishing characteristic of the AAE's performance was its consistently high precision, often exceeding 0.99 across various configurations, as evident in Table II. This high precision indicates a remarkably low false positive rate, a critical factor in many real-world anomaly detection scenarios where false alarms can be costly or disruptive. Conversely, the model's recall exhibited considerable variability, ranging from as low as 0.0037 to a perfect 1.0, depending on the hyperparameter configuration. This variability in recall underscores the AAE's sensitivity to hyperparameter tuning, particularly in its capacity to identify all anomalies present in the dataset.

The F1.5 score, which places greater emphasis on recall, displayed a clear upward trend correlating with increased epochs and appropriate window size selection, as shown in Figures 3 and 2. This trend culminated in near-perfect performance in the optimal configuration, highlighting the AAE's potential for high-stakes applications where the cost of missing anomalies is substantial.

In comparison to the standard AutoEncoder results discussed earlier, the AAE demonstrated superior performance, particularly in its ability to achieve perfect recall without compromising precision. This finding suggests that the adversarial training approach enhances the model's discriminative power, enabling it to distinguish anomalies across a broader spectrum of scenarios.

Additionally, we explored the impact of feature-wise vector thresholds on the AAE model's performance, as presented in Table III. With this approach we marked as anomaly all those features that overcome the threshold even only by a single feature. Building upon our previous experiments with mean thresholds, we selected the optimal hyperparameter configuration (learning rate of 10^{-3} and 50 epochs) and investigated the effect of varying window sizes on feature-wise thresholding approaches. Two thresholding approaches were evaluated: a standard deviation-based method $(3 \times std)$ and a maximum-based method. The latter, denoted as "max" in the table is formally expressed as $\max(\mu + std, \max(x))$ where μ is the mean, std is the standard deviation, and x represents the reconstruction error of the considered feature. Results indicate that the max-based threshold consistently outperformed the standard deviation approach across different window sizes, with the best performance achieved at a window size of 30, yielding an F1.5 score of 98.31%.

V. CONCLUSIONS AND FUTURE WORKS

This study explored the potential of Adversarial AutoEncoders (AAEs) for anomaly detection in Kuka industrial robots, comparing their performance with traditional AutoEncoders (AEs). Our findings suggest that both models may be capable of achieving high performance in detecting anomalies within time-series sensor data, with the AAE potentially offering slight advantages in certain configurations.

Interestingly, our results indicate that larger window sizes might not always lead to improved performance as initially hypothesized. This observation could point to the importance of capturing relevant temporal patterns within a concise context. However, it's worth noting that as window sizes increase,

W	λ	Epochs	Acc. (%)	Prec. (%)	Recall (%)	F1 (%)	F1.5 (%)
20	10^{-2}	2	40.87	76.19	0.37	0.74	0.54
	10^{-3}		81.43	99.85	68.74	81.43	**76.03**
	10^{-4}		49.76	99.46	15.32	26.54	20.71
	10^{-5}		44.81	99.42	7.11	13.27	9.95
10	10^{-3}	2	69.70	98.46	44.69	61.48	53.72
20			81.43	99.85	68.74	81.43	**76.03**
30			62.33	98.93	37.91	54.82	46.79
50			61.79	98.41	37.64	54.45	46.47
100			40.59	100.00	2.88	5.60	4.11
200			40.41	100.00	2.88	5.59	4.10
20	10^{-3}	2	81.43	99.85	68.74	81.43	76.03
		5	89.93	99.82	83.15	90.72	87.65
		10	96.98	99.85	95.05	97.39	96.48
		30	99.73	99.85	99.69	99.77	99.74
		50	99.83	99.72	100.00	99.86	**99.91**
10	10^{-3}	50	99.70	99.44	100.00	99.72	99.83
20			99.83	99.72	100.00	99.86	**99.91**
50			98.85	99.57	98.53	99.05	98.85
100			40.29	100.00	2.36	4.62	3.38
200			38.59	45.00	0.40	0.80	0.58

TABLE II
COMPREHENSIVE COMPARISON FOR AAE MODEL

W	Threshold	Acc. (%)	Prec. (%)	Recall (%)	F1 (%)	F1.5 (%)
10	$3 \times std$	84.25	77.49	100.00	87.32	91.80
	max	96.74	99.90	94.08	96.90	95.80
20	$3 \times std$	86.35	81.28	100.00	89.67	93.38
	max	97.27	99.85	95.53	97.64	96.82
30	$3 \times std$	87.05	82.31	100.00	90.30	93.80
	max	98.50	99.86	97.64	98.74	98.31
50	$3 \times std$	85.75	80.98	100.00	89.49	93.25
	max	99.42	99.63	99.42	99.52	**99.49**
100	$3 \times std$	87.40	82.91	100.00	90.66	94.04
	max	99.19	99.40	99.28	99.34	99.31

TABLE III
COMPARISON OF AAE RESULTS WITH FEATURE-WISE THRESHOLD $\lambda = 10^{-3}$, EPOCHS = 50

the models face growing computational challenges. One of the main hurdles appears to be the need to design larger networks to handle the increased data volume within each time frame. This scaling may require thoughtful redesign of network architectures to effectively capture intricate temporal patterns and dependencies without compromising performance.

The study also hints at the potential benefits of careful hyperparameter optimization, particularly for learning rates and training epochs, in achieving optimal performance for both models. The AAE's performance in our experiments could suggest enhanced stability and generalization capabilities, potentially showcasing the benefits of adversarial training in anomaly detection tasks.

Also, an adaptive thresholding techniques can further enhance the AAE's anomaly detection capabilities, particularly in scenarios where feature distributions may exhibit varying degrees of skewness or outlier presence.

Future work could focus on redesigning the network architecture to better accommodate larger time frames and window sizes. This might involve increasing the number of convolutional layers to capture more complex temporal relationships in the data. Additionally, given the importance of interpretability in industrial applications, further research could explore methods to enhance the explainability of these models, allowing decision-makers to better understand and trust the system's outputs.

REFERENCES

[1] M. A. Bashar and R. Nayak, "Algan: Time series anomaly detection with adjusted-lstm gan," 2023. [Online]. Available: https://arxiv.org/abs/2308.06663

[2] Z. Z. Darban, G. I. Webb, S. Pan, C. C. Aggarwal, and M. Salehi, "Deep learning for time series anomaly detection: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2211.05244

[3] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Usad: Unsupervised anomaly detection on multivariate time series," 2020. [Online]. Available: https://doi.org/10.1145/3394486.3403392

[4] H. Torabi, S. L. Mirtaheri1, and S. Greco, "Practical autoencoder based anomaly detection by using vector reconstruction error," 2023. [Online]. Available: https://doi.org/10.1186/s42400-022-00134-9

[5] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:36663713

[6] H. Xu, Y. Feng, J. Chen, Z. Wang, H. Qiao, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, and D. Pei, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," 2018. [Online]. Available: http://dx.doi.org/10.1145/3178876.3185996

[7] Y. Yamanaka, T. Iwata, H. Takahashi, M. Yamada, and S. Kanai, "Autoencoding binary classifiers for supervised anomaly detection," 2019. [Online]. Available: https://arxiv.org/abs/1903.10709

[8] E. Brophy, Z. Wang, Q. She, and T. Ward, "Generative adversarial networks in time series: A survey and taxonomy," 2021. [Online]. Available: https://arxiv.org/abs/2107.11098

[9] J. Xu, H. Wu, J. Wang, and M. Long, "Anomaly transformer: Time series anomaly detection with association discrepancy," 2022. [Online]. Available: https://arxiv.org/abs/2110.02642