Debiased Contrastive Learning for Time-Series Representation Learning and Fault Detection

Kexin Zhang, *Student Member, IEEE*, Rongyao Cai, Chunlin Zhou, and Yong Liu, *Member, IEEE*

***Abstract*—Building reliable fault detection systems through deep neural networks is an appealing topic in industrial scenarios. In these contexts, the representations extracted by neural networks on available labeled time-series data can reflect system states. However, this endeavor remains challenging due to the necessity of labeled data. Self-supervised contrastive learning (SSCL) is one of the effective approaches to deal with this challenge, but existing SSCL-based models suffer from sampling bias and representation bias problems. This article introduces a debiased contrastive learning framework for time-series data and applies it to industrial fault detection tasks. This framework first develops the multigranularity augmented view generation method to generate augmented views at different granularities. It then introduces the momentum clustering contrastive learning strategy and the expert knowledge guidance mechanism to mitigate sampling bias and representation bias, respectively. Finally, the experiments on a public bearing fault detection dataset and a widely used valve stiction detection dataset show the effectiveness of the proposed feature learning framework.**

***Index Terms*—Data augmentation, expert knowledge, industrial fault detection, self-supervised contrastive learning (SSCL), time-series representation learning.**

I. Introduction

In RECENT years, deep learning has shown impressive performance in extracting hidden patterns and features from time-series data. It has also been proven to be one of the effective techniques for constructing data-driven industrial fault detection methods [1], [2]. Generally, the availability of a large amount of labeled data is one of the critical factors for reliable diagnosis. However, this requirement is difficult to meet in some industrial scenarios because obtaining sufficient labeled data takes time. Therefore, feature learning methods that rely on a limited amount of labeled data have garnered increased attention.

Manuscript received 16 October 2023; accepted 17 January 2024. Date of publication 21 February 2024; date of current version 6 May 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB2012300 and in part by the National Natural Science Foundation of China under Grant 62103363. Paper no. TII-23-4036. *(Corresponding author: Yong Liu.)*

The authors are with the Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China (e-mail: zhangkexin@zju.edu.cn; rycai@zju.edu.cn; c_zhou@zju.edu.cn; yongliu@iipc.zju.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2024.3359409.

Digital Object Identifier 10.1109/TII.2024.3359409

An autoencoder (AE) is an unsupervised artificial neural network that consists of an encoder and a decoder [3], [4]. The AE is trained by minimizing the error between the reconstructed input and the original input, thereby reducing the high dependence on labeled data. Due to this property, AEs are widely used in industrial fault detection tasks, where there is a lack of sufficient labeled data [5], [6], [7]. AE-based fault detection methods can be broadly categorized into those based on intermediate representation extraction (IRE) and those based on reconstruction error comparison (REC). IRE-based methods treat the trained encoder as a feature extractor, taking the intermediate representation Z as the feature of the original input X, and subsequently build a classifier for fault detection tasks. For instance, Yu et al. [6] developed a 1-D residual convolutional AE for gearbox fault diagnosis. REC-based methods, on the other hand, directly compare the reconstruction error with a predefined threshold. Typically, a sample with a reconstruction error exceeding this threshold is identified as a faulty sample. Examples include the denoising AE for wind turbine fault detection [5] and an ensemble of AEs for bearing fault detection [7].

These AE-based unsupervised fault diagnosis methods reduce the dependence on labeled data. However, some challenges in AE-based methods remain. First, IRE-based methods treat AE's encoder as a feature extractor, thereby encoding the raw input into a more compact and informative representation for downstream tasks. However, the commonly used reconstruction loss is not directly related to the downstream task, making the extracted representation suboptimal. Second, REC-based methods usually only include normal data in the training stage so that the model can reconstruct normal samples very well. In contrast, the faulty samples are difficult to reconstruct, which helps us realize the identification of normal samples and faulty samples. However, REC-based methods cannot further identify types of faulty samples. Third, although obtaining a large amount of high-quality labeled data in most industrial scenarios is challenging, a small amount of labeled data is usually still available. The available labeled data is insufficient to train a reliable fault detection model but contains valuable expert knowledge. The current AE-based methods completely discard these available labeled data. However, expert knowledge of these available data is still beneficial when building an AE-based fault detection model.

As a potential feature learning method, self-supervised contrastive learning (SSCL) pretraining on unlabeled data followed by supervised fine-tuning on labeled data is a popular paradigm for learning from limited labeled examples. SSCL based on multiview invariance is one of the most widely used contrastive strategies. It utilizes data augmentation techniques to generate different views of an input sample; then, it learns representations by maximizing the similarity of the views from the same sample and minimizing the similarity of the views from different samples [8], [9]. Importantly, SSCL can train models and extract features without the need for additional labeled data. In industrial fault detection tasks, Liu et al. [10] proposed a time-frequency contrastive method for rotating machinery fault detection. Peng et al. [11] used a linear classifier to identify faults in chemical processes after obtaining the compact and informative representations under the SSCL framework. However, there are challenges when using SSCL for fault detection tasks. First, existing methods employ a biased feature learning framework, suffering from sampling bias, which results in suboptimal representations. Second, these methods lack the guidance of expert knowledge, meaning they cannot guarantee that the extracted features is relevant to the downstream fault detection task. This is what we call the representation bias problem.

This article proposes a new feature learning method for time-series data. It first builds a pretraining model under a newly designed debiased SSCL framework. Then, expert knowledge from the labeled data is then harnessed to guide weight updates throughout the training process. The primary function of the pretrained encoder is to serve as a feature extractor for subsequent fault detection tasks. In particular, we introduce a time-series augmentation technique termed multigranularity augmented view generation (MGAVG). MGAVG first produces augmented views across various granularities, followed by a secondary augmentation involving window warping and slicing techniques. To counter the sampling bias inherent when applying SSCL for time-series feature learning, we propose a debiased contrastive learning approach called momentum clustering contrastive learning (MCCL). MCCL employs a new momentum clustering algorithm to generate pseudolabels, aiding the accurate selection of true negative samples during model training. In addition, we incorporate expert knowledge, sourced from the limited labeled data during the pretraining phase, to adjust the weights of the pretrained encoder in every training epoch. This strategy ensures that the features extracted by our encoder are optimally aligned with downstream tasks and is named the expert knowledge guidance mechanism (EKGM). Finally, we validate the effectiveness of our proposed method using a public bearing fault detection dataset and a widely used valve stiction detection dataset. The main contributions of this article are as follows:

1) A new time-series augmentation method called MGAVG is introduced. This method combines the advantages of two traditional augmentation methods, window warping and window slicing, while offering additional perspectives through the proposed multigranularity approach.
2) We propose a new debiased contrastive feature learning strategy called MCCL, which alleviates the sampling bias problem in SSCL-based feature learning methods and encourages the extracted representations to exhibit a cluster-friendly distribution in the feature space.
3) A new mechanism for incorporating expert knowledge during the self-supervised feature learning process is developed, which can ensure that the final extracted features are directly related to downstream target tasks, thereby alleviating the representation bias problem.
4) The experimental results on both the public industrial bearing fault detection dataset and the widely used valve stiction detection dataset demonstrate that our method can create a reliable feature learning model using a small amount of labeled data and a large amount of unlabeled data. This showcases the practicality of our proposed method in real industrial scenarios.

The rest of this article is organized as follows. Section II gives the problem definition and overall framework. Section III describes the proposed augmentation method. Section IV gives the details of the debiased feature learning framework. Comperhensive experiments are implemented in Section V. Finally, Section VI concludes this article.

II. Overall Framework

A. Problem Definition

Consider $\mathbf{X}_i \in \mathbb{R}^{L\times D}, i = 1, 2, \ldots, N$ as a single time-series sample with a length of L and dimension of D. Here, N represents the total number of samples, and y_i is the true label of $\mathbf{X}_i$. The set $\mathcal{X} = \mathcal{X}_l \cup \mathcal{X}_u$ encompasses all available samples. $\mathcal{X}_l$ denotes the subset of time-series samples with true labels, and $\mathcal{X}_u$ denotes the subset without true labels. The number of samples in $\mathcal{X}_l$ is denoted as N_l, while the number of samples in $\mathcal{X}_u$ is N_u, where $N_l < N_u$. The goal is to develop a feature learning framework tailored for time-series data and subsequently apply it to industrial fault detection tasks. This framework should avoid manual feature engineering and make full use of $\mathcal{X}_u$ and $\mathcal{X}_l$ to build a data-driven detection model.

B. Feature Learning Framework

The proposed feature learning framework is shown in Fig. 1. We first divide all samples into a labeled set $\mathcal{X}_l$ and an unlabeled set $\mathcal{X}_u$. The whole feature learning is split into two branches: one for expert knowledge acquisition based on supervised learning and the other for representation learning based on self-supervised learning. The expert knowledge acquisition branch constructs a label prediction network (LPN) to acquire expert knowledge from $\mathcal{X}_l$. The representation learning branch utilizes $\mathcal{X}_u$ to train the feature extraction network (FEN) under the SSCL framework. Its purpose is to establish a mapping model from raw time-series data to compact and informative representations. Compared with the traditional SSCL-based biased feature learning method, we introduce MCCL and EKGM to mitigate the problems of sampling and representation biases. The MCCL module employs a new momentum clustering algorithm to produce pseudolabels for each training batch, selects true positive and negative samples based on these pseudolabels, and updates the weights of the FEN using the debiased contrastive loss. The EKGM leverages the LPN, which contains expert knowledge, to update the weights of the FEN, allowing the FEN to incorporate expert knowledge during training. Finally, the trained FEN is used to extract features from raw time-series

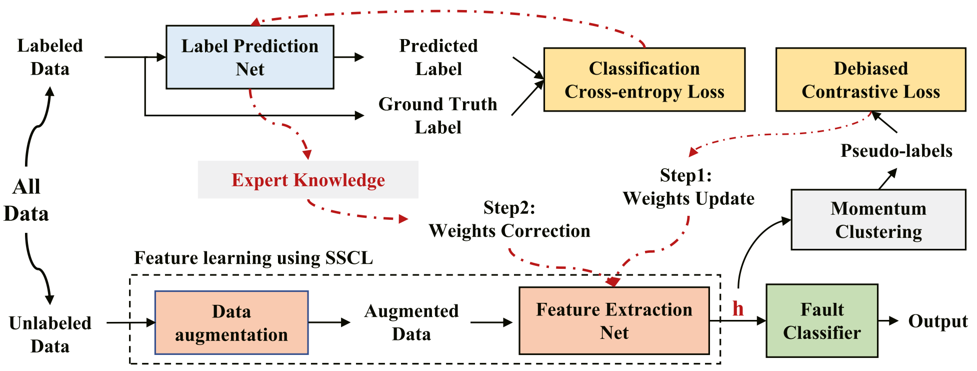


Fig. 1. Feature learning framework.

samples, which are then used as inputs to construct a fault classifier.

III. Data Augmentation for Time-Series

In multiview invariance-based SSCL, data augmentation is a commonly used method to generate different views of an input sample. Iwana et al. [12] pointed out that the window warping and window slicing algorithms are highly recommended. In this study, we continue to use these two effective algorithms. In addition, we develop a new multigranularity segmentation method for time-series data to further increase the diversity of augmented views.

A. MGAVG

This study employs the temporally weighted hierarchical clustering algorithm to segment time-series data at multiple granularities. This algorithm was the first to successfully identify adaptive segmentation for different action stages in videos [13]. We extend it to achieve multigranularity segmentations and to generate augmented views of time-series data.

Given a time-series sample $\mathbf{X}_i = \{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots, \mathbf{x}_{i,L}\} \in \mathbb{R}^{L\times D}$, where L represents the length and D denotes the dimension. Let $\mathbf{x}_{i,t} \in \mathbb{R}^D$ be the observation at time t. For the sake of simplicity, we will omit the subscript i in the subsequent description. We start by defining a matrix W of size $L \times L$. The element located in the jth row and kth column is represented as w_{jk} that can take the value of 0 or 1. If $w_{jk} = 1$, it signifies that $\mathbf{x}_k$ is the observation most similar to $\mathbf{x}_j$ out of all observations. The similarity between two observations is gauged by both the feature distance and the temporal distance. The definition of the feature distance is

$$G^f(j,k) = 1 - \mathcal{S}\langle \mathbf{x}_j, \mathbf{x}_k\rangle \tag{1}$$

where $G^f(j,k)$ represents the feature distance between $\mathbf{x}_j$ and $\mathbf{x}_k$. $\mathcal{S}\langle \mathbf{x}_j, \mathbf{x}_k\rangle$ is the cosine distance. While the feature distance can measure the similarity between any two observations, it does not account for temporal information. Intuitively, two consecutive (or very close) observations are more likely to be similar than two distant ones. Therefore, the temporal distance is introduced

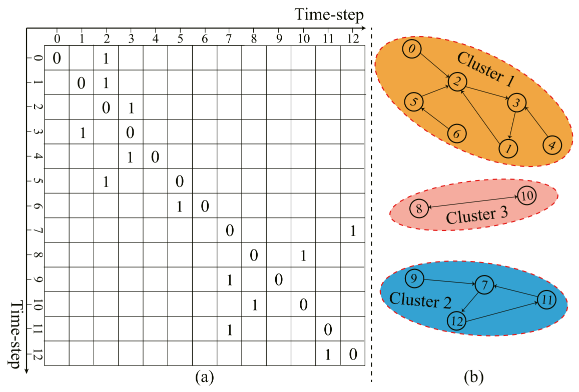


Fig. 2. Segmentation examples for the 12-steps time-series. (a) The sparse matrix obtained by Eq. (4). (b) Three directed graphs constructed from the sparse matrix, each directed graph forming a cluster.

by

$$G^t(j,k) = \exp(|j-k|/L) \tag{2}$$

where $|j-k|/L$ provides a weighing mechanism relative to the length of a time-series sample. Then, the modulated distance between any two observations is

$$G(j,k) = G^f(j,k) \cdot G^t(j,k). \tag{3}$$

We use (3) to calculate the modulated distance between every pair of observations and then populate the matrix W with these distance values. For each row of W, we retain only the smallest value and set all other values to zero, i.e.,

$$w_{jk} = \begin{cases} 0 & \text{if } G(j,k) > \min\limits_{\forall k} G(j,k) \\ 1 & \text{otherwise} \end{cases}. \tag{4}$$

We obtain the matrix W which consists solely of 0 s and 1 s. A straightforward example is depicted in Fig. 2(a). It can be seen that each row contains just one element with a value of 1, while all other elements are set to 0. For instance, if $w_{02} = 1$, it indicates that the observation most similar to $\mathbf{x}_0$ is $\mathbf{x}_2$. In this case, the observations $\mathbf{x}_0$ and $\mathbf{x}_2$ can be linked to form a cluster. During the clustering, all possible connections are explored, resulting in the formation of multiple clusters, as illustrated in Fig. 2(b). The initial segmentation has the finest granularity, allowing the

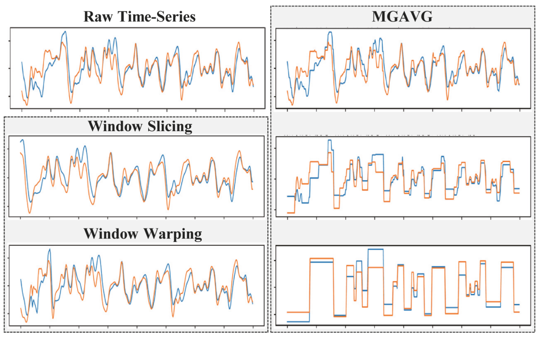


Fig. 3. Visualization of data augmentation.

original series to be divided into several segments. Then, we repeatedly merge the similar clusters to obtain multigranularity segmentations. Finally, we can split the original time-series into multiple segments, denoted as $\mathbf{X} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_P\}$. P is the number of segments. The raw values of all observations in each segment is replaced by the mean of all observations in that segment, which is defined as

$$\mathbf{X}_{[a^{\mathcal{S}_p}:b^{\mathcal{S}_p}]} = \frac{1}{b^{\mathcal{S}_p} - a^{\mathcal{S}_p} + 1} \sum_{t=a^{\mathcal{S}_p}}^{b^{\mathcal{S}_p}} \mathbf{X}_t \tag{5}$$

where $a^{\mathcal{S}_p}$ and $b^{\mathcal{S}_p}$ denote the start and end timestamps of the segment $\mathcal{S}_p$, respectively. For illustration, consider Fig. 2(b), where the initial segmentation segregates the original series into three clusters comprising six segments: $\mathbf{x} = \{\mathcal{S}_{[0:6]}, \mathcal{S}_{[7]}, \mathcal{S}_{[8]}, \mathcal{S}_{[9]}, \mathcal{S}_{[10]}, \mathcal{S}_{[11:12]}\}$. Even though $\mathbf{x}_8$ and $\mathbf{x}_{10}$ are part of the same cluster, they are noncontiguous in the time dimension, leading them to be categorized into separate segments. MGAVG is adept at constructing augmented views of time-series across varying granularities, while maintaining the temporal attributes inherent to the original series. Examples of such augmented views can be seen in Fig. 3.

B. Window Warping and Window Slicing

The augmented views generated by the MGAVG maintain the global pattern of the original time-series but overlook the local patterns. Thus, based on the multigranularity augmented views, we apply window warping and window slicing algorithms for further transformations of these views. Window warping involves modifying a randomly selected window of a time-series by either speeding it up or slowing it down, i.e.,

$$\mathbf{X}_{[t:t+S_{wp}*R_{wp}]} = \text{Warping}(\mathbf{X}_{[t:t+w]}, S_{wp}, R_{wp}) \tag{6}$$

where S_{wp} represents the size of the selected window. R_{wp} is warping ratios, which are typically $\frac{1}{2}$ or 2. $\mathbf{X}_{[t:t+S_{wp}*R_{wp}]}$ refers to the warped window. Unlike window warping, the process of window slicing involves randomly sampling a slicing window from the original time-series, i.e.,

$$\mathbf{X}_{[t:t+S_{ws}]} = \text{Slicing}(\mathbf{X}, S_{ws}) \tag{7}$$

Algorithm 1: Debiased Feature Learning Framework.

```
Input: The labeled time-series examples
  $\mathcal{X}_l = \{(\mathbf{X}_i, y_i) : i \in 1, 2, \ldots, N_l\}$, the unlabeled
  time-series examples $\mathcal{X}_u = \{(\mathbf{X}_j) : j \in 1, 2, \ldots, N_u\}$,
  augmentation operation $\mathcal{T}(\cdot)$, FEN'encoder $f^u_{\theta[E]}(\cdot)$ and
  LPN'encoder $f^s_{\phi[E]}(\cdot)$, projector $g(\cdot)$, classifier $c(\cdot)$,
  constant $\alpha$ and $\beta$
 1: for e, sampled batch in Enumerate($\mathcal{X}_u$) do
 2:    Sampled batch $\mathcal{X}^e_u$;
 3:    Get augmentations, representations, and projections
       $\hat{\mathcal{X}}^e_u = \mathcal{T}(\mathcal{X}^e_u)$, $\mathcal{B}^h_{(e)} = f^u_{\theta[E]}(\hat{\mathcal{X}}^e_u)$, $\mathcal{B}^z_{(e)} = g(\hat{\mathcal{X}}^e_u)$;
 4:    Substitute $\mathcal{B}^h_{(e)}$ into (13) to obtain the cluster centers
       $\mathcal{C}_{(e)}$ of the current batch;
 5:    Substitute $\mathcal{C}_{(e)}, \mathcal{C}_{(e-1)}$ and $\alpha$ into (14) to obtain the
       updated cluster centers $\hat{\mathcal{C}}_{(e)}$;
 6:    Use $\hat{\mathcal{C}}_{(e)}$ to obtain the pseudo-labels $\hat{\mathcal{Y}}_{(e)}$ of $\mathcal{X}^e_u$;
 7:    Substitute $\hat{\mathcal{Y}}_{(e)}$ and $\mathcal{B}^z_{(e)}$ into (16) to calculate
       debiased contrastive loss $\mathcal{L}_u$;
 8:    Update encoder $f^u_{\theta[E]}(\cdot)$ and projector $g(\cdot)$ to
       minimize $\mathcal{L}_u$;
 9: end for
10: for k, sampled batch in Enumerate($\mathcal{X}_l$) do
11:    Sampled batch $\mathcal{X}^k_l$;
12:    Get representations $\mathcal{B}^h_{(k)} = f^s_{\phi[E]}(\mathcal{X}^k_l)$;
13:    Get predicted probabilities $P_k = c(\mathcal{B}^h_{(k)})$;
14:    Calculate supervised classification loss $\mathcal{L}_s$ using
       Cross-Entropy loss;
15:    Update encoder $f^s_{\phi[E]}(\cdot)$ and classifier $c(\cdot)$ to
       minimize $\mathcal{L}_s$;
16: end for
17: Substitute $f^u_{\theta[E]}(\cdot)$, $f^s_{\phi[E]}(\cdot)$, and $\beta$ into (17) to obtain
    the debiased encoder $f^{u_d}_{\theta[E]}(\cdot)$;
Output: FEN's debiased encoder $f^{u_d}_{\theta[E]}(\cdot)$
```

where S_{ws} represents the size of the sampled slicing window. Illustrations of window warping and window slicing can be found in Fig. 3.

IV. Debiased Feature Learning Framework

This section introduces a debiased feature learning framework for time-series data. This framework comprises two networks: LPN and EEN. We also introduce a method for generating pseudolabels based on clustering, as well as a weight correction mechanism informed by expert knowledge. These are designed to address sampling and representation biases, respectively. A detailed pseudocode of this framework can be found in Algorithm 1.

A. Dilated Causal Convolution

Convolutional neural networks (CNNs) are commonly used in data-driven fault detection methods [14], [15], [16], [17], [18]. However, traditional 2D-CNNs encounter certain limitations

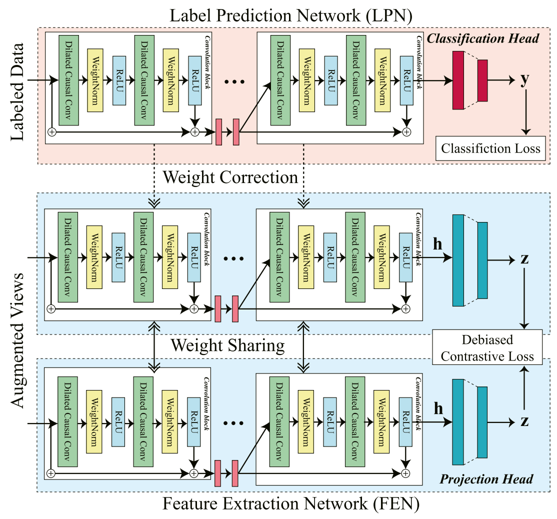


Fig. 4. Model structure.

when processing time-series data. This article introduces a dilated causal convolution (DCC) to construct both LPN and FEN. DCC's strength lies in its ability to directly extract temporal features from time-series data, even if the input series have varying lengths. Compared with the widely used long short-term memory networks, DCC allows efficient parallelization on GPUs, further improving computational efficiency. The DCC broadens its receptive field via an exponentially expanding convolution kernel, and it can capture long-range temporal dependencies by deepening the model. A typical DCC block encompasses two DCC layers, two weight normalization layers, two ReLU layers, and a 1-D convolution layer.

B. LPN and FEN

The model structure is depicted in Fig. 4, LPN is defined as $f_\phi^s(\cdot)$ and is represented by the red part, while FEN is defined as $f_\theta^u(\cdot)$ and is represented by the blue part. Both LPN and FEN possess the same encoder structure, but they have different output layers. The encoder comprises three stacked DCC blocks and is denoted as

$$\mathbf{h}_i = f^{(3)[k_3,d_3]}(f^{(2)[k_2,d_2]}(f^{(1)[k_1,d_1]}(\mathbf{X}_i))) \tag{8}$$

where k represents the size of the convolution kernel. d is dilation factor. $f^{(h)}$ represents the hth DCC block. The output layer of LPN is a classification head $c(\cdot)$, which is a three-layer fully connected neural network. Its purpose is to directly output the predicted probabilities of the input time-series. $c(\cdot)$ is defined as

$$\mathbf{o}_i = c(\mathbf{h}_i) = \text{Softmax}(W_c^{(3)}\text{ReLU}(W_c^{(2)}\text{ReLU}((W_c^{(1)}\mathbf{h_i})))) \tag{9}$$

where $\mathbf{o}_i$ is the predicted probabilities of each possible class for $\mathbf{X}_i$. The final predicted class corresponds to the class with the highest probability. The output layer of FEN is a projection head, denoted as $g(\cdot)$. $g(\cdot)$ is a two-layer fully connected neural network that maps the representation $\mathbf{h}_i$ to $\mathbf{z}_i$, where the debiased contrastive loss is applied, i.e.,

$$\mathbf{z_i} = g(\mathbf{h}_i) = W_g^{(2)}\text{ReLU}((W_g^{(1)}\mathbf{h}_i)). \tag{10}$$

C. Debiased Contrastive Learning

1) Biased Contrastive Learning: Given a time-series sample $\mathbf{X}_i$, and its two augmented views, $\mathbf{X}_i^{T_1}$ and $\mathbf{X}_i^{T_2}$. We first obtain the projections $\mathbf{z}_{i(1)}$ and $\mathbf{z}_{i(2)}$ of the augmented views according to (8) and (10). $\mathbf{z}_{i(1)}$ and $\mathbf{z}_{i(2)}$ are considered as a positive pair. In the biased contrastive learning framework, negative samples are indispensable. An implicit sampling mechanism is adopted. Specifically, given a batch of N_b samples, we first perform augmentations on all samples within the batch, resulting in $2N_b$ augmented samples. When a positive pair is assigned, the rest $2(N_b - 1)$ samples within the batch are considered negative samples. The biased contrastive loss for $\mathbf{X}_i^{T_1}$ is defined as

$$\mathcal{L}_{i(1)} = -\log \frac{\exp\left(\text{sim}\left(\mathbf{z}_{i(1)}, \mathbf{z}_{i(2)}\right)/\tau\right)}{\sum_{j=1}^{2N_b} \mathbb{1}_{[j\neq i]} \exp\left(\text{sim}\left(\mathbf{z}_{i(1)}, \mathbf{z}_{j(*)}\right)/\tau\right)} \tag{11}$$

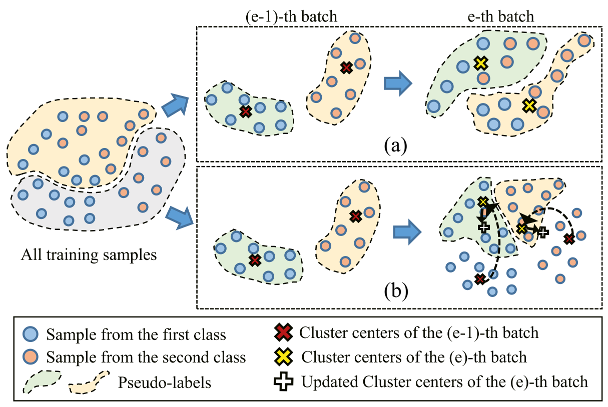
where τ is the temperature parameter. $\text{sim}(\cdot)$ represents the similarity between two vectors, typically measured using cosine similarity. $\mathbf{z}_{j(*)}$ denotes samples other than $\mathbf{z}_{i(1)}$ and $\mathbf{z}_{i(2)}$ within a training batch. (11) suffers from issues related to sampling and representation biases, leading it to be termed a biased contrastive learning loss. On one hand, the idea behind (11) is that all samples within a training batch belong to different classes, thus, augmented views from distinct samples are directly regarded as negative samples. This assumption is often unreasonable, as $\mathbf{X}_i$ and $\mathbf{X}_j$ might belong to the same class, making $\mathbf{X}_j$ a false negative sample. Furthermore, (11) encourages all samples to form a uniform distribution in the feature space, while the actual data distribution should be clustered. On the other hand, (11) lacks a direct relationship with downstream tasks. Therefore, a trained model trained may not ensure that the extracted features align with the needs of downstream tasks. This also makes (11) vulnerable to representation bias.

2) MCCL: We first introduce the MCCL module to alleviate the sampling bias problem. Assuming that the labels of all training samples are known, we can accurately select the true negative samples based on these labels. Supervised contrastive learning (SCL) [19] is a constraint based on known labels. it is defined as

$$\mathcal{L} = \sum_{i\in\mathcal{I}} \frac{1}{|\mathcal{P}(i)|} \sum_{p\in\mathcal{P}(i)} -\log \frac{\exp\left(\text{sim}\left(\mathbf{z}_i, \mathbf{z}_p\right)/\tau\right)}{\sum_{k\in\mathcal{A}(i)} \exp(\text{sim}\left(\mathbf{z}_i, \mathbf{z}_k\right)/\tau)} \tag{12}$$

where $\mathcal{I}$ is the index set of all samples within a batch, and $\mathcal{P}(i)$ represent the index set of samples belonging to the same class as $\mathbf{X}_i$. $\mathcal{A}(i)$ stands for an index set containing all samples. While (12) provides an ideal solution when labels are available, it is difficult to implement in practice, especially for time-series data. Thus, this study employs a clustering algorithm to generate pseudolabels capable of distinguishing between positive and negative samples during model training.

Define the set of representations of all augmented views within a training batch as $\mathcal{B}^h = \{\mathbf{h}_k\}, k \in 1, 2, \ldots, N_b$. The set of projections by $g(\cdot)$ is $\mathcal{B}^z = \{\mathbf{z}_k\}, k \in 1, 2, \ldots, N_b$. We first use a clustering algorithm to cluster all samples in $\mathcal{B}^h$, here we use the K-means algorithm and the number of clusters is set to C_b. The clustering pseudolabels and the cluster centers of the



(e-1)-th batch
e-th batch
(a)
(b)
All training samples
Sample from the first class
Sample from the second class
Pseudo-labels
Cluster centers of the (e-1)-th batch
Cluster centers of the (e)-th batch
Updated Cluster centers of the (e)-th batch

Fig. 5. Illustration of MCCL. (a) Clustering contrastive learning. (b) MCCL.

eth training batch can be expressed as

$$\mathcal{Y}_{(e)}, \mathcal{C}_{(e)} \leftarrow \text{Clustring}(\mathcal{B}^h_{(e)}) \tag{13}$$

where $\mathcal{B}^h_{(e)}$ represents a set of all augmented views within the eth training batch.

In the model training phase, $f^u_\theta(\cdot)$ is updated with each training batch. If the pseudolabels in two consecutive training batches are complete independent, it can lead to unstable model training. This is because the model has to adapt to different objective functions. Inspired by the momentum update algorithm in MoCo [20], this study employs the momentum update strategy to generate a more stable pseudolabel set. The cluster centers $\mathcal{C}_{(e)}$ is updated by

$$\hat{\mathcal{C}}_{(e)} = \alpha\mathcal{C}_{(e-1)} + (1-\alpha)\mathcal{C}_{(e)} \tag{14}$$

where $\alpha \in [0, 1]$ represent the cluster center update coefficient. The updated cluster centers are denoted by $\hat{\mathcal{C}}_{(e)}$. The pseudolabels of each training batch are influenced not only by the current training batch but also by the previous one. This strategy ensures that past information is considered when generating pseudolabels. Finally, $\hat{\mathcal{C}}_{(e)}$ is used to re-generate pseudolabels for the e-th training batch, denoted as

$$\hat{\mathcal{Y}}_{(e)} = \mathcal{G}(\hat{\mathcal{C}}_{(e)}, \mathcal{B}^h_{(e)}) \tag{15}$$

where $\mathcal{G}$ is the pseudolabel assignment function, i.e., assigning each samples to the cluster with the nearest distance. Finally, the (12) can be rewritten as

$$\mathcal{L} = \sum_{i\in\mathcal{I}} \frac{1}{\left|\hat{\mathcal{Y}}(i)\right|} \sum_{p\in\hat{\mathcal{Y}}(i)} -\log \frac{\exp\left(\text{sim}\left(\mathbf{z}_i, \mathbf{z}_p\right)/\tau\right)}{\sum_{k\in\mathcal{A}(i)} \exp(\text{sim}\left(\mathbf{z}_i, \mathbf{z}_k\right)/\tau)} \tag{16}$$

where $\hat{\mathcal{Y}}(i)$ is the index set of samples belonging to the same class as $\mathbf{X}_i$ according to the momentum clustering results.

Fig. 5 provides an illustration of the MCCL. The motivation behind MCCL can be divided into two aspects. The first is the clustering phase, also known as clustering contrastive learning. The purpose of this phase is to encourage a clustered distribution of the feature space among the samples, rather than a uniform distribution. The second aspect involves using momentum clustering to ensure that the results of generating pseudolabels are more accurate and stable. During the model training phase, samples in each training batch are randomly sampled from the complete dataset. Therefore, the cluster centers of the samples in this training batch may deviate from the true cluster centers of the complete dataset. As shown in Fig. 5(b), in the feature space, we use the clustering results from the previous batch to correct the cluster centers of the current batch. We aim for slow changes in clustering centers to ensure training stability. If the clustering results from the previous batch are accurate, the cluster centers of the current batch will be closer to the true cluster centers, ensuring the accuracy of generating pseudolabels.

3) EKGM: Although (16) encourages $f^u_\theta(\cdot)$ to encode the original data into a feature space with multiple clusters, the clustering algorithm remains fundamentally an unsupervised learning strategy. As a result, the extracted representation $\mathbf{h}$ is not guaranteed to be directly relevant to downstream fault detection tasks. To address this, this article leverages the limited labeled data $\mathcal{X}_l$, which contains label information deemed as expert knowledge, to further correct $f^u_\theta(\cdot)$. We first employ the supervised classification loss to train LPN $f^s_\phi(\cdot)$. Since LPN and FEN share the same encoder structure, we adjust the weights of FEN's encoder by

$$f^{u_d}_{\theta[E]}(\cdot) = \beta f^u_{\theta[E]}(\cdot) + (1-\beta) f^s_{\phi[E]}(\cdot) \tag{17}$$

where $\beta \in [0, 1]$ represents the weight correction coefficient. $f^{u_d}_{\theta[E]}(\cdot)$ is the debiased encoder.

D. Fault Classifier

Given $f^{u_d}_{\theta[E]}(\cdot)$ and labeled data $\mathcal{X}_l$, we first obtain the representations of $\mathcal{X}_l$, i.e.,

$$\mathcal{H}_l = f^{u_d}_{\theta[E]}(\mathcal{X}_l). \tag{18}$$

Then, $\mathcal{H}_l$ and its true label set $\mathcal{Y}_l$ are used to construct a fault detection classifier $Clf(\cdot)$, i.e.,

$$\mathcal{Y}_l = Clf(\mathcal{H}_l). \tag{19}$$

V. Experiments

A. Dataset Description

Two time-series datasets are used to demonstrate the effectiveness of the proposed method. The first is a well-known public dataset from the case western reserve university (CWRU) bearing data center. This dataset collects various vibration signals, including drive end accelerometer data, fan end accelerometer data, and base accelerometer data, under different working conditions with a sampling rate of 12 kHz. In this article, we construct one normal baseline category and nine drive end bearing fault categories by selecting different fault diameters (specifically 7, 14, and 21 mils). Each fault diameter is associated with three fault types: inner race defect (IRD), outer race defect (ORD), and ball defect (BD).

The second dataset is the international stiction data base (ISDB), which is supported by [21] and is a well-known benchmark for validation of novel methods concerning control loop performance assessment. These loops were collected from various process industries, including chemical plants (CHEM),

TABLE I
TEST LOOPS IN THE ISDB DATASET

Loop	Type	Malfunction	Loop	Type	Malfunction
CHEM 1	Fic	Stiction	CHEM 24	Fic	Likely stiction
CHEM 2	Fic	Stiction	CHEM 26	Lev	Likely stiction
CHEM 3	Tem	Quantisation	CHEM 29	Fic	Stiction
CHEM 4	Lev	Tuning problem	CHEM 32	Fic	Likely stiction
CHEM 5	Fic	Stiction	CHEM 33	Fic	Disturbance
CHEM 6	Fic	Stiction	CHEM 34	Fic	Disturbance
CHEM 10	Pre	Stiction	CHEM 58	Fic	No oscillation
CHEM 11	Fic	Stiction	MIN 1	Tem	Stiction
CHEM 12	Fic	Stiction	PAP 2	Fic	Stiction
CHEM 13	Ana	Faulty sensor	PAP 4	Con	Deadzone
CHEM 14	Fic	Faulty sensor	PAP 5	Con	Stiction
CHEM 16	Pre	Interaction	PAP 7	Fic	Disturbance
CHEM 23	Fic	Likely stiction	PAP 9	Tem	Non-stiction

pulp and paper mills (PAP), buildings (BAS), mining (MIN), and power plants (POW). Our goal is to identify stiction loops and nonstiction loops. Note that being nonstiction does not necessarily mean that the loop is normal, it may have other issues, such as external disturbances or sensor failures.

B. Data Collection and Preprocessing

For the CWRU dataset, we randomly sampled 450 time-series samples under each condition, each with a length of 512 and a dimension of 2. To simulate scenarios where high-quality training data is difficult to obtain at most industrial sites, we selected only 50 from each class for the training data, with the remaining 400 serving as the test data. The final constructed dataset consists of 500 training samples and 4000 test samples. For the ISDB dataset, we selected a total of 85 control loops with available data. From each control loop, we randomly sampled 60 time-series samples, each of which has a length of 800 and a dimension of 2. The final dataset contains 5100 samples. To facilitate comparisons with other valve stiction detection methods, we selected 26 control loops (1560 samples) as the test set and the remaining 59 control loops (3540 samples) as the training set. The main details of the test loops are described in Table I, where Tem, Fic, Pre, Lev, Con, and Ana denote the temperature, flow, pressure, level, concentration, and analyzer control, respectively.

In our experiments, we consider all available training samples as unlabeled data. Therefore, for the CWRU dataset, the available unlabeled data consists of 500 samples, while for the ISDB dataset, there are 3540 unlabeled samples. To acquire labeled data, our approach is to sample from all the training samples according to a predefined ratio. For this purpose, we introduce a new hyperparameter: availability of labels (AOL). As an example, with an AOL of 0.5, for the CWRU dataset, the available labeled data would be 250 samples ($500 \times 0.5 = 250$), whereas for the ISDB dataset, it would be 1770 samples ($3540 \times 0.5 = 1770$).

In the data preprocessing stage, we use Z-score standardization to scale the raw data to fit a standard normal distribution,

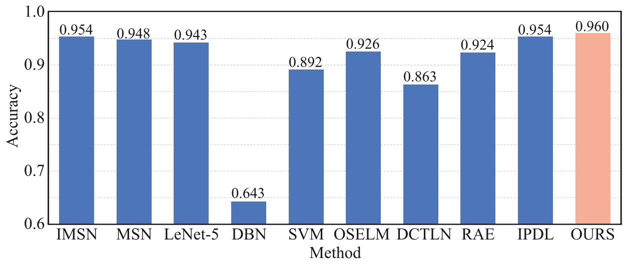


Fig. 6. Detection and comparison results for the CWRU dataset.

which is beneficial for most deep learning models. Given a sample $\mathbf{x} \in \mathbb{R}^{L\times D}$. The standard score of $\mathbf{x}^j$ (jth feature of $\mathbf{x}$) is calculated as

$$\mathbf{z}^j = \frac{\mathbf{x}^j - \bar{\mathbf{x}}^j}{s^j} \tag{20}$$

where $\bar{\mathbf{x}}^j$ is the mean of $\mathbf{x}^j$, and s^j is the standard deviation of $\mathbf{x}^j$.

C. Evaluation Metric

The classification accuracy is used to evaluate the proposed method, i.e.,

$$Acc = \frac{1}{n}\sum_{i=0}^{n-1} \mathbb{1}_{[\hat{y}_i = y_i]} \tag{21}$$

where $\mathbb{1} \in \{0, 1\}$ is the indicator function evaluating to 1 if $\hat{y}_i = y_i$. $\hat{y}_i$ is predicted label and y_i is the true label of ith time-series sample.

D. Detection and Comparison Results

This study first presents the detection results of the CWRU dataset and also provides comparison results with existing fault detection methods. The selected comparison methods include IMSN [17], MSN [22], LeNet-5, DBN [23], SVM, OSELM [24], DCTLN [25], RAE [26], and IPDL [27]. The results are displayed in Fig. 6. Both RAE and IPDL are methods designed to handle uncertainty in time-series data and were originally applied to wind speed forecasting. In this article, the neural network model outputs from these two methods serve as representation vectors and are connected to a classification head for the fault detection task. IMSN, MSN, LeNet-5, OSELM, and DCTLN were originally designed for classification tasks and can thus be directly applied to fault detection tasks. From the results, it can be seen that our method achieved the highest detection accuracy. Moreover, compared to OSELM and DCTLN, our method is designed for a ten-class classification task, which is more challenging than a four-class classification task. It should be noted that the primary objective of this article is not to achieve the highest possible detection accuracy. The experiments in this subsection aim solely to demonstrate that the proposed method can indeed surpass the state-of-the-art methods in fault detection tasks. The results in Fig. 6 are provided without a fine-tuning of the hyperparameters. However, further improvements in detection accuracy can be achieved through hyperparameter tuning,

TABLE II
DETECTION AND COMPARISON RESULTS FOR THE ISDB DATASET

Method	Accuracy	NOT tested loops
Higher-order statistics method	0.7917	2
Cross-correlation method	05833	2
Statistics method	0.6400	1
Curve fitting method	0.4800	1
Peak slope method	0.5600	1
Zone Segmentation method	0.6000	1
D-value ANN method	0.7917	2
Relay-based method	0.6538	0
Waveform shape analysis method	0.4231	0
PSD/ACF method	0.6923	0
BSD-CNN method	0.7692	0
Multiple-timescale CNN	0.8076	0
RAE	0.6538	0
IPDL	0.8076	0
MTFCC	0.8461	0
Ours	**0.8461**	0

The bold values indicate that these are the highest accuracies in the experimental results.

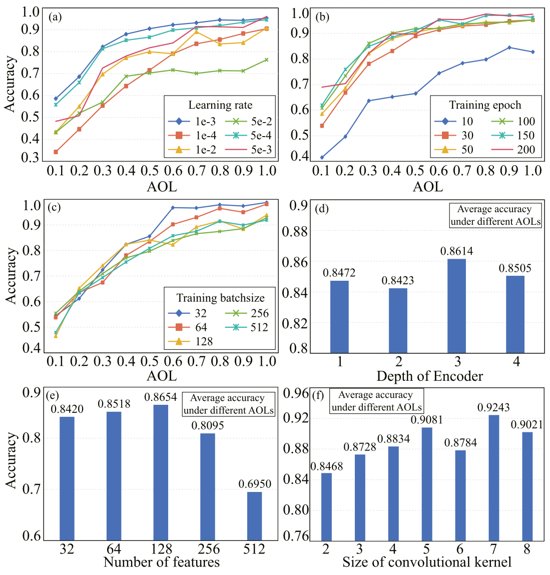


Fig. 7. Results of the hyperparameter selection experiment.

as discussed in Section V-E. It is also worth noting that the reported detection accuracies of other methods in Fig. 6 might differ from their original work. This difference arises from our use of a different data collection method, especially the selection of a smaller amount of training data.

For the ISDB dataset, we use the classification accuracy of the control loop as the evaluation metric. The comparison results with fifteen other detection methods are listed in Table II. Our method outperforms the existing methods with an accuracy of 0.8461, which is the highest among the considered methods. It can be observed that only nine methods can be applied to all test control loops (“Not tested loops” is 0), while the remaining seven methods are applicable only to some control loops. This demonstrates that our method can be applied to a broader range of industrial systems. Similar to our approach, D-value ANN, Multiple-timescale CNN, BSD-CNN, and MTFCC all utilize deep learning techniques to construct their detection models. However, D-value ANN, Multiple-timescale CNN, and BSD-CNN employ a fully supervised strategy, which significantly increases their dependence on labeled data. MTFCC conducts feature extraction under a SCCL framework and achieves the same accuracy as our method. Nevertheless, it does not address issues related to representation bias and sampling bias. Consequently, there remain potential risks when using MTFCC for other detection tasks.

E. Hyperparameters Study

The proposed method has several important hyperparameters, including the learning rate, training epoch, batch size, depth of the encoder, number of features in the representation vector, and size of the convolutional kernel. In this section, we conducted experiments on the CWRU dataset to demonstrate the impact of these hyperparameters on detection results.

1) Learning Rate: The learning rate is a tuning parameter in optimization algorithms that determines the step size at each iteration when moving toward a minimum of a loss function. Generally speaking, if the learning rate is too small, the convergence process can be very slow. Conversely, if the learning rate is too large, the model might struggle to converge. We tested the learning rate with six different values: 1e-3, 1e-4, 1e-2, 5e-3, 5e-4, and 5e-2. The detection results for these different learning rates are shown in Fig. 7(a). From the results, it is evident that a learning rate of 1e-3 offers the best performance for the detection model. In contrast, when the learning rate is set too high (5e-2, represented by the green line) or too low (1e-4, represented by the orange line), the model’s performance significantly deteriorates. Based on these findings, we set the learning rate to 1e-3 in this study.

2) Training Epoch: The number of training epochs is also considered a hyperparameter. It specifies how many times the entire training dataset is processed by the learning algorithm. Too few epochs might not yield a satisfactory detection model, while too many can be time-consuming. We tested six different values for the training epochs: 10, 30, 50, 100, 150, and 200. The detection results for these epochs are presented in Fig. 7(b). It can be seen that when the number of epochs exceeds 50, the detection model will exhibit reliable performance. Furthermore, when AOL is low, increasing the training epochs improves the final detection accuracy. However, while more epochs can enhance accuracy, they also require more time. Thus, this article strikes a balance between detection accuracy and training time by setting the number of epochs to 100.

3) Batch Size: The batch size defines the number of training samples that are propagated through the network. In the proposed feature framework, the batch size is closely related to both the momentum clustering process and contrastive learning loss. We tested the batch size with five different values: 32, 64, 128, 256, and 512. The detection results are shown in Fig. 7(c). From the results, it appears that our method performs better with a smaller batch size. We believe this superior performance is due to the fact that a smaller batch size ensures the accuracy of the clustering results, which in turn makes the model loss, as calculated by (16), more precise. Therefore, this article recommends a batch size of 32.

4) Depth of FEN'encoder: For a neural network model, a deeper structure implies better nonlinear expressive capability, allowing it to capture more complex relationships. However, a complex network structure not only extends the training time but also makes the training process more challenging. We set the encoder depth to four distinct values: 1, 2, 3, and 4. The detection results are presented in Fig. 7(d). It's worth noting that the reported detection accuracy represents the average value across different AOLs. Observably, as the depth increases, the model's performance enhances. At a depth of 3, the average detection rate peaks. Therefore, this study concludes that an encoder depth of 3 is a reasonable choice.

5) Number of Features in the Representation Vector: The number of features in the representation vector directly impacts the expressive capability of the model. The greater the number of features, the more powerful its representation capability becomes. However, with too many features, the representation vector might contain excessive redundant or irrelevant information. On the other hand, if the number of features is too few, the representation capability is compromised, and there's a risk of omitting useful information. We tested the model with representation vectors of five different feature sizes: 32, 64, 128, 256, and 512. The detection results are displayed in Fig. 7(e). Notably, as the number of features rises, the detection accuracy generally improves. However, after a certain point, an increase in features leads to a decline in average accuracy, suggesting that the model's performance may suffer. Based on our observations, a representation vector with 128 features appears optimal.

6) Size of Convolutional Kernel: The kernel size is a hyperparameter for both the LPN and the FEN. It is important as it determines the receptive field of a layer. We set the number of features to seven different values: 2, 3, 4, 5, 6, 7, and 8. The detection results are shown in Fig. 7(f). From the figure, it can be seen that a larger convolutional kernel size enhances detection accuracy. However, this also leads to an increase in the number of parameters that require training. Conversely, a smaller convolutional kernel is simpler to train. Yet, it is challenging for a small convolutional kernel to extract global information without increasing the model's depth, which in turn raises computational costs. As a result, in this article, we are more inclined to set the convolutional kernel size to 5.

7) Cluster Center Update Coefficient: This study delves deeper into the impact of different cluster center update coefficients on detection accuracy. The primary settings for this experiment are as follows: the data augmentation method used is WW, the number of clusters is set at 10, and the EKGM is not implemented. According to the definition in (14), a larger value of α suggests a heightened reliance on the previous training batch. Fig. 8 presents the detection results, with the tested coefficients being 0.0, 0.1, 0.3, 0.5, 0.7, 0.9, and 1.0. The red star in Fig. 8 denotes the coefficient with the highest detection accuracy. In most scenarios, utilizing the previous training batch to minimally update the clusters of the current batch often results in better detection accuracy, notably when $\alpha = 0.1$ or $\alpha = 0.3$. However, when $\alpha > 0.3$, the detection accuracy diminishes due to an excessive dependence on the prior training batch, preventing the correct clustering pseudolabels from being assigned to the current batch.

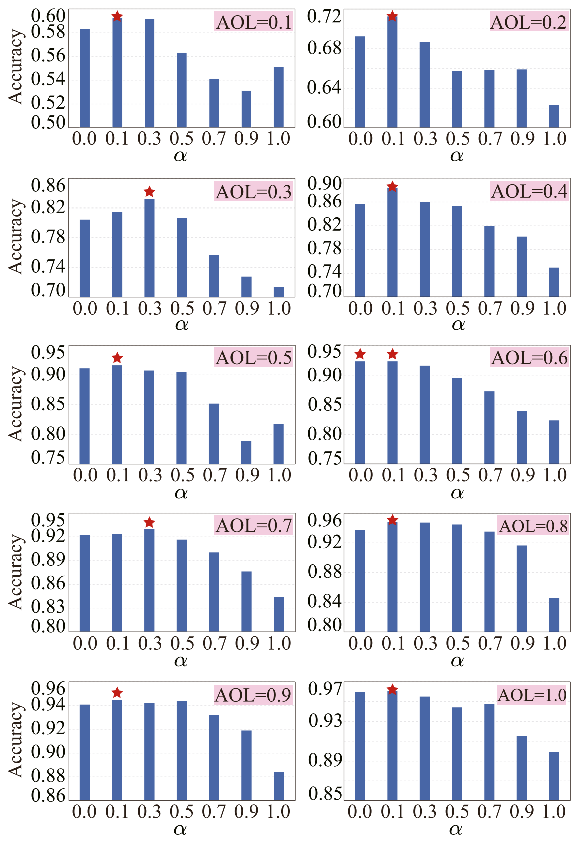


Fig. 8. Accuracy under different cluster center update coefficients.

8) Weight Correction Coefficient: The choice of the weight correction coefficient poses a significant challenge. This study conducts experiments using various weight correction coefficients to observe their effects on detection accuracy. According to (17), a larger coefficient implies a lower degree of weight correction. For instance, when $\beta = 0.9$, the FEN $f_\theta^s(\cdot)$ is less influenced by the LPN $f_\phi^u(\cdot)$. Fig. 9 presents the experimental results. When AOL is low ($\beta \leq 0.4$), the detection accuracy tends to increase as β decreases. This suggests that the expert information from the limited labeled data remains valuable. When AOL is relatively high ($\alpha > 0.4$), a decrease in β leads to an increase in detection accuracy, which then stabilizes. This indicates that while the information provided by the LPN is beneficial for the FEN, its advantage is restricted. At AOL values of 0.5 and 0.7, an excessive reliance on the LPN diminishes the final detection accuracy.

F. Ablation Study

MGAVG, MCCL, and EKGM are the main modules of our proposed method. In this section, we demonstrate the effectiveness of the proposed modules through ablation experiments on the CWRU and ISDB datasets.

1) Effectiveness of MGAVG: This study initially validates the effectiveness of the proposed data augmentation method using the CWRU and ISDB datasets. The primary settings for this experiment are as follows: the number of clusters is set at

TABLE III
DETECTION RESULTS OF DIFFERENT DATA AUGMENTATION METHODS

Dataset	AOL	J	S	R	WW	WS	WW+WS	M	M+WW	M+WS	M+WW+WS
CWRU	0.1	0.4455	0.4205	0.4272	0.4945	0.4991	0.492	0.4172	0.5167	0.5720	**0.6245**
	0.2	0.5460	0.5210	0.5255	0.6510	0.6777	0.6457	0.5320	0.6797	0.7680	**0.8005**
	0.3	0.6653	0.6518	0.6725	0.7535	0.7891	0.7632	0.6487	0.7912	0.8355	**0.8442**
	0.4	0.7085	0.7038	0.7112	0.7951	0.8277	0.8232	0.7040	0.8322	0.8657	**0.8845**
	0.5	0.7563	0.7805	0.7445	0.8207	0.8695	0.8610	0.7612	0.8627	0.9037	**0.9092**
	0.6	0.7695	0.8237	0.8277	0.8597	0.8845	0.8915	0.7937	0.9010	**0.9182**	0.9177
	0.7	0.8355	0.8557	0.8302	0.8750	0.8967	0.9100	0.8405	0.9030	0.9165	**0.9217**
	0.8	0.8425	0.8677	0.8812	0.9045	0.9081	0.9187	0.8595	0.9327	**0.9522**	0.9510
	0.9	0.8610	0.8925	0.8812	0.9097	0.9042	0.9280	0.8837	0.9312	**0.9365**	0.9330
	1.0	0.8853	0.9212	0.8912	0.9265	0.9275	0.9372	0.9072	0.9382	0.9312	**0.9497**
ISDB	0.1	0.5822	0.6250	0.6479	0.6687	0.6802	0.6677	0.6208	0.7104	0.7145	**0.7250**
	0.2	0.6708	0.6802	0.6885	0.6854	0.6989	0.7208	0.6364	0.7468	0.7479	**0.7583**
	0.3	0.6968	0.6708	0.6781	0.7083	0.7395	0.6989	0.6708	0.7281	**0.7906**	0.7302
	0.4	0.6593	0.7052	0.6802	0.7323	0.7281	0.7302	0.6593	0.6979	**0.7843**	0.7562
	0.5	0.6781	0.6989	0.6572	0.7406	0.7177	0.7395	0.6770	0.7489	**0.7687**	0.7656
	0.6	0.7052	0.7270	0.7072	0.7531	0.7083	0.7333	0.6697	0.6770	0.7510	**0.7593**
	0.7	0.7125	0.7166	0.6510	0.7197	0.6916	0.7260	0.6447	0.7614	0.7583	**0.8062**
	0.8	0.7145	0.7156	0.6770	0.7854	0.7270	0.7822	0.6750	0.7458	0.7562	**0.8031**
	0.9	0.7635	0.7083	0.6708	0.7625	0.7500	0.7312	0.6812	0.7708	0.7666	**0.7906**
	1.0	0.7041	0.7291	0.7020	0.7645	0.7679	0.7208	0.6885	0.7312	0.7416	**0.7977**

The bold values indicate that these are the highest accuracies in the experimental results.

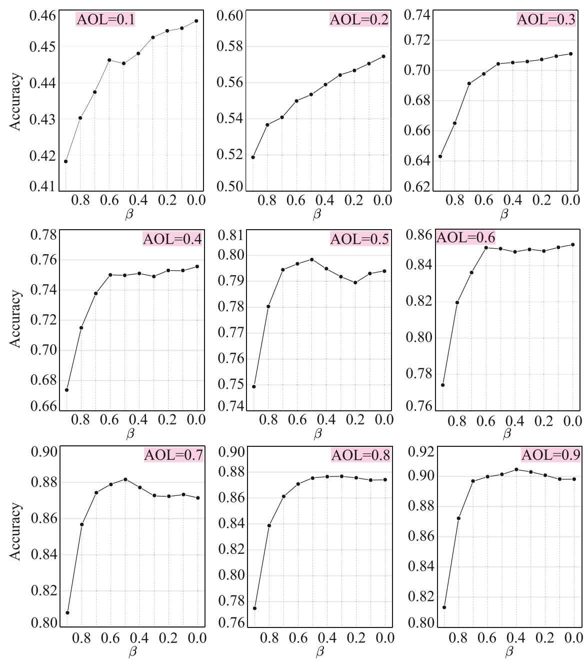


Fig. 9. Accuracy under different weight correction coefficients.

10, the cluster center update coefficient is 0.1, and the EKGM strategy is not employed. The data augmentation methods used in this experiment encompass jitter (J), scaling (S), rotation (R), window warping (WW), window slicing (WS), the combination of window warping and window slicing (WW+WS), MGAVG (M), MGAVG with window warping (M+WW), MGAVG with window slicing (M+WS), and MGAVG with both window warping and window slicing (M+WW+WS). These results can be found in Table III. The results suggest that WW and WS outperform J, S, and R on both datasets. While using M alone cannot provide the highest detection accuracy, pairing M with WW and WS (M+WW and M+WS) boosts the detection accuracy. For the CWRU dataset, M+WS records the highest detection accuracy at AOL settings of 0.6, 0.8, and 0.9. For the ISDB dataset, M+WS tops the detection accuracy at AOL values of 0.3, 0.4, and 0.5. In addition, the detection accuracy experiences significant improvement when M+WW is combined with M+WS (M+WW+WS). The highest detection accuracy is achieved under seven AOL settings for both datasets. Thus, integrating multiple augmentation methods proves more advantageous for the pretraining model in acquiring superior representations.

2) Effectiveness of MCCL: This study then validates the effectiveness of the MCCL. The main experimental settings are as follows: the data augmentation method is WW, the number of clusters is 10, the cluster center update coefficient is 0.1, and the EKGM is not adopted. The contrastive methods compared in this experiment include biased contrastive learning (BCL) [9], debiased contrastive learning (DCL) [28], hard contrastive learning (HCL) [29], SCL [19], and the proposed MCCL. In this context, MCCL_k indicates that the number of clusters is set to k. The results are provided in Table IV. On the CWRU dataset, SCL achieves the highest detection accuracy across all AOL settings. This superior performance can be attributed to its entirely unbiased contrastive learning framework, although it requires additional labels during model training. Conversely, BCL performs the poorest due to its completely biased property. Both DCL and HCL, being implicit debiased methods, exhibit detection accuracies superior to that of BCL. The detection accuracy of our proposed MCCL varies with the number of clusters. Specifically, when the number of clusters is small (such as MCCL_2 and MCCL_4), the accuracy is notably lower than with a larger number of clusters. This suggests that MCCL benefits from increasing the number of clusters. For the ISDB dataset, SCL is unable to achieve the highest detection accuracy across all AOL settings, possibly due to erroneous labels in

TABLE IV
DETECTION RESULTS OF DIFFERENT CONTRASTIVE METHODS

Dataset	AOL	BCL [9]	DCL [28]	HCL [29]	SCL [19]	MCCL$_2$	MCCL$_4$	MCCL$_6$	MCCL$_8$	MCCL$_{10}$
CWRU	0.1	0.4635	0.5017	0.5090	0.6572	0.4777	0.4865	0.5200	0.4660	**0.5705**
	0.2	0.6007	0.6527	0.6395	0.7767	0.5915	0.6225	0.6280	**0.6830**	0.6680
	0.3	0.7362	**0.7675**	0.7627	0.8315	0.7290	0.7115	0.7420	0.7580	0.7285
	0.4	0.7825	0.8010	0.7912	0.8705	0.7610	0.7742	0.7555	0.7950	**0.8020**
	0.5	0.8230	0.8282	0.8295	0.9067	0.8012	0.8062	0.8333	0.8307	**0.8477**
	0.6	0.8582	0.8682	0.8517	0.9237	0.8207	0.8465	0.8418	**0.8900**	0.8662
	0.7	0.8750	0.8765	0.8732	0.9367	0.8352	0.8835	0.8645	**0.8947**	0.8827
	0.8	0.8852	0.8975	0.8975	0.9552	0.8800	0.8942	**0.9225**	0.9177	0.9200
	0.9	0.8940	0.8967	0.8867	0.9387	0.9157	0.9150	**0.9165**	0.9072	0.8955
	1.0	0.9177	0.9175	0.9095	0.9515	0.9198	0.9300	0.9215	**0.9340**	0.9182
ISDB	0.1	0.6677	0.7218	0.7062	0.7833	0.6229	0.7177	0.7520	0.7281	**0.7541**
	0.2	0.7208	0.7041	0.7302	0.8208	0.7302	0.6843	0.7114	**0.7937**	0.7135
	0.3	0.6989	0.7333	0.7322	0.7635	0.7427	**0.7697**	0.7187	0.7229	0.7281
	0.4	0.7302	0.7312	0.7239	0.7343	0.7260	0.7156	**0.7604**	0.7427	0.6552
	0.5	0.7395	0.7604	0.7166	0.7479	**0.7937**	0.7479	0.7406	0.7854	0.7343
	0.6	0.7333	0.7458	0.7427	0.7875	0.7927	**0.8020**	0.7645	0.7104	0.7145
	0.7	0.7260	0.7729	0.7479	0.8437	0.7645	0.7572	**0.7854**	0.7270	0.7114
	0.8	0.7822	0.7552	**0.7885**	0.7520	0.7677	0.7625	0.7562	0.7510	0.7052
	0.9	0.7312	0.7656	0.7135	0.7458	0.7752	0.7313	0.7364	0.7343	**0.7802**
	1.0	0.7208	0.7541	0.7771	0.7604	0.7354	0.7281	0.7541	0.7239	**0.7854**

The bold values indicate that these are the highest accuracies in the experimental results.

TABLE V
IMPACT OF EKGM ON DETECTION RESULTS UNDER DIFFERENT AOLS

Dataset	AOL	No β	Use β (max)	Use β (mean)
CWRU	0.1	0.4238	0.4570 (+0.0332)	0.4440 (+0.0202)
	0.2	0.5368	0.5745 (+0.0377)	0.5535 (+0.0167)
	0.3	0.6608	0.7110 (+0.0502)	0.6940 (+0.0332)
	0.4	0.7165	0.7555 (+0.0390)	0.7388 (+0.0223)
	0.5	0.7843	0.7985 (+0.0142)	0.7883 (+0.0040)
	0.6	0.8173	0.8515 (+0.0342)	0.8374 (+0.0201)
	0.7	0.8598	0.8817 (+0.0219)	0.8666 (+0.0068)
	0.8	0.8626	0.8767 (+0.0141)	0.8691 (+0.0065)
	0.9	0.9003	0.9045 (+0.0042)	0.8888 (-0.0115)
	1.0	0.9180	0.9345 (+0.0165)	0.9215 (+0.0035)
ISDB	0.1	0.6948	0.7625 (+0.0677)	0.6931 (-0.0017)
	0.2	0.6614	0.7771 (+0.1157)	0.7453 (+0.0839)
	0.3	0.6197	0.7989 (+0.1792)	0.7321 (+0.1124)
	0.4	0.7354	0.7937 (+0.0583)	0.7209 (-0.0145)
	0.5	0.6937	0.7406 (+0.0469)	0.7103 (+0.0166)
	0.6	0.6448	0.8468 (+0.2020)	0.7543 (+0.1095)
	0.7	0.7260	0.7542 (+0.0282)	0.7377 (+0.0117)
	0.8	0.8312	0.8197 (-0.0115)	0.7642 (-0.0670)
	0.9	0.7000	0.7791 (+0.0791)	0.7442 (+0.0442)
	1.0	0.7229	0.7948 (+0.0719)	0.7652 (+0.0423)

the dataset. Yet, the proposed MCCL still achieved the top detection accuracy at various AOLs. Compared to the CWRU dataset, the ISDB dataset poses greater challenges, particularly in determining the optimal number of cluster centers. Overall, the ablation experiment results from both the ISDB and CWRU datasets underscore the effectiveness of MCCL.

3) Effectiveness of EKGM: This study validates the effectiveness of the EKGM strategy both with and without the use of the weight correction mechanism during the pretraining stage. The results are presented in Table V. The third column displays the detection results without EKGM (denoted by No β), the fourth column shows the peak detection accuracy among all weight correction coefficients [labeled as Use β (max)], and the fifth column presents the average detection accuracy [represented by Use β (mean)]. It can be observed that the proposed EKGM strategy is effective across all AOLs for the CWUR dataset. However, for the ISDB dataset, the EKGM strategy becomes ineffective when the AOL is set to 0.8. We hypothesize that this ineffectiveness arises from incorrect labeling within the training dataset, which causes the model to assimilate inaccurate information. When the AOL is set to 0.9 or higher, the presence of more accurately labeled data counteracts the earlier mistakes, ensuring the sustained effectiveness of the EKGM.

G. Time Efficiency Analysis

Time efficiency is also one of the key factors in assessing model performance. In this article, we analyze the constructed model using asymptotic notation, specifically Big O notation. In the method we propose, convolutional layers serve as the cornerstone of the detection model. Therefore, this article primarily focuses on the time complexity analysis of convolutional layers, and the time complexity of all convolutional layers is represented as

$$O\left(\sum_{l=1}^{d} C_{l-1} \cdot C_l \cdot S_l^2 \cdot H_l^2\right) \tag{22}$$

where l is the index of a convolutional layer, and d is the depth of the encoder. C_{l-1} is the input channels of the l-th layer. C_l is the output channels of the lth layer. S_l is the spatial size of the convolutional kernel, H_l is the spatial size of the output feature map. Comparing the running times with other methods is helpful for model evaluation, but obtaining fair comparison results is challenging. The actual algorithm runtime depends on various factors, including hardware performance, model architecture, and even some easily overlooked hyperparameters. Therefore, this article primarily provides the training time and inference time of our proposed method under different architectures. All

TABLE VI
TRAINING TIME AND INFERENCE TIME AT 50 TRAINING EPOCH

Dataset	Module	Training (s)	Inference (s)
CWRU	Base	11.11	0.0012
	+ MGAVG	26.24	0.0012
	+ MCCL	13.51	0.0007
	+ EKGM(AOL=0.2)	11.79	0.0012
	+ EKGM(AOL=0.4)	12.49	0.0012
	+ EKGM(AOL=0.6)	12.95	0.0012
	+ EKGM(AOL=0.8)	13.56	0.0012
	+ EKGM(AOL=1.0)	13.96	0.0012
ISDB	Base	73.48	0.0006
	+ MGAVG	162.12	0.0006
	+ MCCL	77.99	0.0004
	+ EKGM(AOL=0.2)	76.59	0.0005
	+ EKGM(AOL=0.4)	79.01	0.0005
	+ EKGM(AOL=0.6)	81.56	0.0005
	+ EKGM(AOL=0.8)	84.05	0.0005
	+ EKGM(AOL=1.0)	87.22	0.0005

code was written in Python using the PyTorch framework. The main hardware devices include an NVIDIA GeForce RTX 4090 GPU, an Intel Core i7-13700KF CPU, and 32 GB RAM. The running time of our method is shown in Table VI. The results reported in Table VI are based on a training epoch setting of 50. Based on Table VI, MGAVG significantly increases the training time because it requires data segmentation and generation at multiple time scales. On the other hand, the introduction of MCCL and EKGM has a relatively minor impact on training time, while the runtime of the EKGM module also depends on the utilization of labeled data. Additionally, both training and inference times are influenced by the length and the number of time-series samples. This is also the reason for the difference in inference times between the ISDB dataset and the CWRU dataset.

VI. CONCLUSION

This article presents a debiased feature learning framework for industrial time-series that can be deployed in industrial fault detection scenarios. The key idea is to improve the SSCL framework by incorporating MGAVG, MCCL, and EKGM. MGAVG retains the advantages of existing data augmentation methods and provides additional multigranularity views. MCCL introduces an explicit debiased contrastive learning framework based on clustering and pseudolabel generation, which alleviates the sampling bias issue. EKGM integrates expert knowledge during the model training stage to guide the weight update of the feature extraction model. This ensures that the extracted features are directly relevant to the downstream tasks and helps reduce the representation bias problem. Experiments conducted on two datasets have demonstrated the effectiveness of the proposed method.

We outline two directions worthy of further investigation. 1) Integrating inductive biases into time-series representation learning frameworks enhances their efficacy. Although many fault detection techniques predominantly rely on a data-driven modeling approach, the incorporation of reasonable inductive biases or priors proves beneficial for deep neural network-based fault detection models. While purely data-driven models are versatile across different tasks, they demand substantial training data. 2) Exploring adversarial attacks and robust analysis in data-driven fault detection models has gained attention. With the widespread use of deep neural networks in industrial fault detection, the susceptibility of these models to adversarial samples is garnering increasing attention. Studying adversarial attacks and defenses in data-driven fault detection is promising, yet current literature on this topic is sparse. Most existing research is concentrated in the domain of computer vision and natural language processing. The ramifications of adversarial attacks on industrial intelligence systems are still unclear.

REFERENCES

[1] Q. Sun and Z. Ge, “A survey on deep learning for data-driven soft sensors,” *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, pp. 5853–5866, Sep. 2021.

[2] J. Jiao, M. Zhao, J. Lin, and K. Liang, “A comprehensive review on convolutional neural network in machine fault diagnosis,” *Neurocomputing*, vol. 417, pp. 36–63, 2020.

[3] J. Zhai, S. Zhang, J. Chen, and Q. He, “Autoencoder and its various variants,” in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2018, pp. 415–419.

[4] P. Baldi, “Autoencoders, unsupervised learning, and deep architectures,” in *Proc. ICML Workshop Unsupervised Transfer Learn., Ser. Int. Conf. Mach. Learn.*, 2012, vol. 27, pp. 37–49.

[5] G. Jiang, P. Xie, H. He, and J. Yan, “Wind turbine fault detection using a denoising autoencoder with temporal information,” *IEEE/ASME Trans. Mechatronics*, vol. 23, no. 1, pp. 89–100, Feb. 2018.

[6] J. Yu and X. Zhou, “One-dimensional residual convolutional autoencoder based feature learning for gearbox fault diagnosis,” *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6347–6358, Oct. 2020.

[7] S. Plakias and Y. S. Boutalis, “A novel information processing method based on an ensemble of auto-encoders for unsupervised fault detection,” *Comput. Ind.*, vol. 142, 2022, Art. no. 103743.

[8] Y. Ding, J. Zhuang, P. Ding, and M. Jia, “Self-supervised pretraining via contrast learning for intelligent incipient fault detection of bearings,” *Rel. Eng. Syst. Saf.*, vol. 218, 2022, Art. no. 108126.

[9] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 1597–1607.

[10] Y. Liu, W. Wen, Y. Bai, and Q. Meng, “Self-supervised feature extraction via time–frequency contrast for intelligent fault diagnosis of rotating machinery,” *Measurement*, vol. 210, 2023, Art. no. 112551.

[11] P. Peng et al., “Progressively balanced supervised contrastive representation learning for long-tailed fault diagnosis,” *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–12, 2022, doi: 10.1109/TIM.2022.3151946.

[12] B. K. Iwana and S. Uchida, “An empirical survey of data augmentation for time series classification with neural networks,” *PLoS One*, vol. 16, no. 7, Jul. 2021, Art. no. e0254841.

[13] M. S. Sarfraz, N. Murray, V. Sharma, A. Diba, L. Van Gool, and R. Stiefelhagen, “Temporally-weighted hierarchical clustering for unsupervised action segmentation,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 11220–11229.

[14] L. Wen, X. Li, L. Gao, and Y. Zhang, “A new convolutional neural network-based data-driven fault diagnosis method,” *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5990–5998, Jul. 2018.

[15] W. Qiu, Q. Tang, J. Liu, and W. Yao, “An automatic identification framework for complex power quality disturbances based on multifusion convolutional neural network,” *IEEE Trans. Ind. Informat.*, vol. 16, no. 5, pp. 3233–3241, May 2020.

[16] R. Liu, F. Wang, B. Yang, and S. J. Qin, “Multiscale kernel based residual convolutional neural network for motor fault diagnosis under nonstationary conditions,” *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 3797–3806, Jun. 2020.

[17] Z.-X. Hu, Y. Wang, M.-F. Ge, and J. Liu, “Data-driven fault diagnosis method based on compressed sensing and improved multiscale network,” *IEEE Trans. Ind. Electron.*, vol. 67, no. 4, pp. 3216–3225, Apr. 2020.

[18] K. Zhang, Y. Liu, Y. Gu, J. Wang, and X. Ruan, “Valve stiction detection using multitimescale feature consistent constraint for time-series data,” *IEEE/ASME Trans. Mechatronics*, vol. 28, no. 3, pp. 1488–1499, Jun. 2023.

[19] P. Khosla et al., “Supervised contrastive learning,” in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 18661–18673.
[20] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 9726–9735.
[21] M. Jelali and B. Huang, *Detection and Diagnosis of Stiction in Control Loops: State of the Art and Advanced Methods*. London, U.K.: Springer-Verlag, 2010.
[22] Z. Hu, H. Youmin, B. Wu, J. Liu, D. Han, and T. Kurfess, “Hand pose estimation with multi-scale network,” *Appl. Intell.*, vol. 48, no. 8, pp. 2501–2515, 2018.
[23] H. Shao, H. Jiang, X. Zhang, and M. Niu, “Rolling bearing fault diagnosis using an optimization deep belief network,” *Meas. Sci. Technol.*, vol. 26, no. 11, 2015, Art. no. 115002.
[24] W. Mao, L. He, Y. Yan, and J. Wang, “Online sequential prediction of bearings imbalanced fault diagnosis by extreme learning machine,” *Mech. Syst. Signal Process.*, vol. 83, pp. 450–473, 2017.
[25] L. Guo, Y. Lei, S. Xing, T. Yan, and N. Li, “Deep convolutional transfer learning network: A new method for intelligent fault diagnosis of machines with unlabeled data,” *IEEE Trans. Ind. Electron.*, vol. 66, no. 9, pp. 7316–7325, Sep. 2019.
[26] M. Khodayar, O. Kaynak, and M. E. Khodayar, “Rough deep neural architecture for short-term wind speed forecasting,” *IEEE Trans. Ind. Informat.*, vol. 13, no. 6, pp. 2770–2779, Dec. 2017.
[27] M. Khodayar, J. Wang, and M. Manthouri, “Interval deep generative neural network for wind speed forecasting,” *IEEE Trans. Smart Grid*, vol. 10, no. 4, pp. 3974–3989, Jul. 2019.
[28] C.-Y. Chuang, J. Robinson, Y.-C. Lin, A. Torralba, and S. Jegelka, “Debiased contrastive learning,” in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 8765–8775.
[29] J. D. Robinson, C.-Y. Chuang, S. Sra, and S. Jegelka, “Contrastive learning with hard negative samples,” in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–13.

Kexin Zhang (Student Member, IEEE) received the B.S. and the M.S. degrees in engineering from Chine University of Geosciences, Wuhan, China, in 2016 and 2019, respectively, and the Ph.D. degree in control engineering and science from Zhejiang University, Hangzhou, China in 2023.

His major research interests include intelligent time series analysis, deep learning, data-driven industrial fault diagnosis, and artificial intelligence security.

Rongyao Cai received the B.S. degrees in chemical engineering from Zhejiang University of Technology, Hangzhou, China, in 2022. He is currently working toward the M.S. degree in control engineering with College of Control Science and Engineering, Zhejiang University, Hangzhou, China.

His major research interests include data mining, machine learning on industrial time-series data.

Chunlin Zhou received the B.S. and M.Eng. degrees in mechatronics engineering from Shanghai Jiaotong University, Shanghai, China in 2003 and 2006, respectively, and the Ph.D. degree in mechatronics engineering from Singapore Nanyang Technological University, Singapore, in 2012.

He is currently an Associate Professor with the College of Control Science and Engineering, Zhejiang University, Hangzhou, China. He is the author of more than 30 academic papers and the inventor of five Chinese/US patents. He is currently the Deputy Sectary of University Students Robotics Competition Committee of Zhejiang Province. His research interests include mechatronics design, motion control and their applications in surgical robots and biomimetic robots.

Dr. Zhou was the recipient of the 2014 Higher Education Prize of Zhejiang Province and 2018 Higher Education Prize of China.

Yong Liu (Member, IEEE) received the B.S. degree in computer science and engineering and the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2001 and 2007, respectively.

He is currently a Professor with the Institute of Cyber Systems and Control, Department of Control Science and Engineering, Zhejiang University. He has authored or coauthored more than 100 research papers in machine learning, computer vision, information fusion, and robotics. His current research interests include machine learning, robotics vision, information processing, and granular computing.