Anomaly Detection and Classification with Deep Learning Techniques

by

Nima Amini

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Chemical Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

Statement of contributions

This thesis consists in part of two manuscripts written for publication, which are co-authored by myself, and my supervisor.

One work has been published by Elsevier in Neurocomputing on Nov. 27, 2021. Amini, N. and Zhu, Q., 2021. "Fault detection and diagnosis with a novel source-aware autoencoder and deep residual neural network."

The other work is submitted to Computers & Chemical Engineering on Feb. 4, 2022. Amini, N. and Zhu, Q., 2022. "SA^{2}C: Self-adversarial autoencoding classifier for unsupervised anomaly detection."

Abstract

The capability of deep learning (DL) techniques for dealing with non-linear, dynamic and correlated data has paved the way for developing DL-based solutions for real-world problems. Among them, anomaly detection has been received increasing attention due to its wide applications from process safety to video surveillance. Anomaly detection is the task to detect deviations and outliers from normal data, and can be divided into two groups: supervised and unsupervised techniques. In unsupervised techniques, the training data consists of normal data (with or without a small proportion of anomalies), while in supervised techniques it is assumed that labelled data from normal samples and anomalies is available. In this thesis, we propose two techniques for unsupervised and supervised anomaly detection, respectively.

Self-adversarial autoencoding classifier (SAAC) is proposed for unsupervised anomaly detection with an end-to-end training phase. SAAC employs an adversarial autoencoder (AAE) as an anomaly generator without having access to real anomalous samples. An autoencoding binary classifier (ABC) is trained to differentiate between the generated anomalous samples by the AAE and the normal samples. AAE and ABC are trained in an adversarial way, and three datasets, namely the Tennessee-Eastman process, Credit Card, and CIFAR10, are utilized to demonstrate its superiority over other existing techniques in terms of anomaly detection.

As a supervised technique, a two-step technique for fault detection and diagnosis (FDD) is proposed. A source-aware autoencoder (SAAE) is proposed as an extension of autoencoders to incorporate faulty samples in their training stage. In SAAE, flexibility in tuning recall and precision trade-off, ability to detect unseen faults and applicability in imbalanced data sets are achieved. Bidirectional long short-term memory (BiLSTM) with skip connections is designed as the structure of the fault detection network in SAAE. Further, a deep network with BiLSTM and residual neural network (ResNet) is proposed for the subsequent fault diagnosis step to avoid randomness imposed by the order of the input features. A framework for combining fault detection and fault diagnosis networks is also presented without the assumption of having a perfect fault detection network. A comprehensive comparison among relevant existing techniques in the literature and SAAE-ResNet is also conducted on the Tennessee-Eastman process, which shows the superiority of the proposed FDD method.

Acknowledgements

I would like to sincerely thank my supervisor, Dr. Qinqin Zhu, for her constant support and guidance throughout this research.

I gratefully acknowledge the financial support by the Natural Sciences and Engineering Research Council of Canada (NSERC) as well as the University of Waterloo, which made this research possible.

Dedication

This is dedicated to my honorable mother, and my beloved sister.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of Anomaly detection and Classification

An anomaly is considered as a deviation from a normal notion, and the task to differentiate anomalies is referred to as anomaly detection. The effective design of anomaly detection techniques has been received increasing importance by the research community, and these techniques have wide applications in a variety of fields such as process monitoring, spam detection, intrusion detection and video surveillance [1, 2, 3, 4, 5].

Anomaly detection and classification, also known as fault detection and diagnosis (FDD) in chemical engineering, plays an essential role in industry to avoid frequent shut-downs and maintenance, damages to the environment and equipment, and disastrous accidents. In FDD, fault detection aims to differentiate between normal and faulty samples, while fault diagnosis is to locate the root causes of faulty samples to reduce economic losses and improve process safety. An illustration of the effects and tasks involved in FDD is presented in Figure 1.1. It is noted that precise labelling is important for efficient and accurate FDD. On one hand, mistakenly labeling normal samples as faults could result in unnecessary operation disruptions, and lead to unnecessary extra labor costs. On the other hand, regarding faulty samples as normal ones may also lead to severe consequences such as exposing equipment to detrimental operating conditions. Therefore, the accurate and robust design of FDD algorithms is highly demanded.

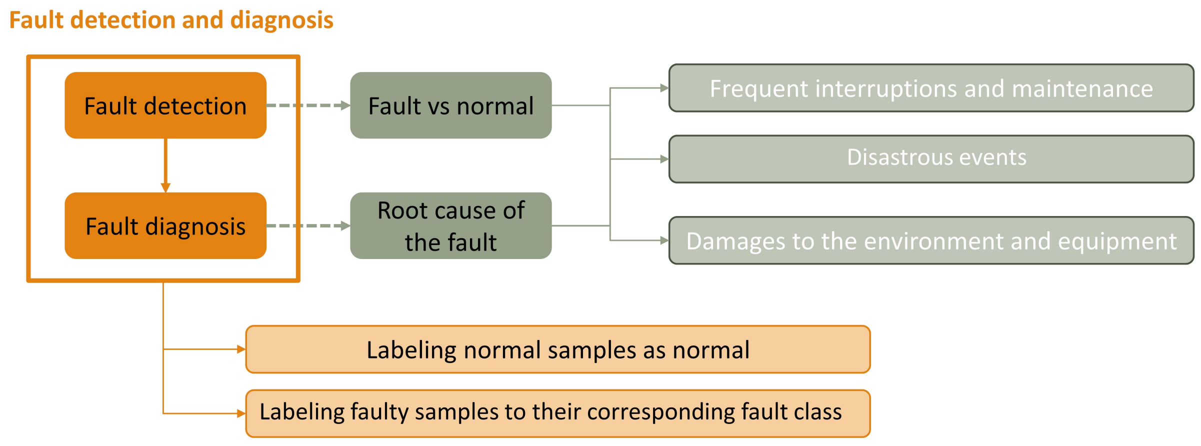
Fault detection and diagnosis
Fault detection
Fault diagnosis
Fault vs normal
Root cause of the fault
Frequent interruptions and maintenance
Disastrous events
Damages to the environment and equipment
Labeling normal samples as normal
Labeling faulty samples to their corresponding fault class

Figure 1.1: Steps and objectives in FDD.

Several series of techniques are developed for FDD purposes, namely knowledge-based ones and data-driven ones. Knowledge-based approaches, which require in-depth understanding of the process, have been proposed in the literature [6, 7, 8]. However, their applications are limited due to their strong dependence on domain-specific knowledge and complexities of the system.

Nowadays, with the continual increase in the computation power of computers and the amount of the acquired high-dimensional data, data-driven techniques serve as an alternative for knowledge-based techniques. Among them, multivariate statistical methods are widely studied due to their effectiveness to process high-dimensional data. For instance, principal component analysis (PCA) is an unsupervised method that converts highly correlated data into linearly uncorrelated features by successively maximizing the variance [9, 10]. Several variants of PCA such as dynamic PCA, nonlinear PCA and batch PCA were proposed in the literature to model different process characteristics [11, 12, 13, 14]. Independent component analysis (ICA), as another unsupervised method, extracts mutually independent components with the utilization of higher-order statistics, and it is applicable for non-Gaussian process monitoring [15, 16]. Dynamic ICA, robust ICA, and hybrid ICA are also developed for process monitoring [16, 17, 18]. Different from PCA and ICA, partial least squares (PLS) works as a supervised learning method, and it projects process and quality spaces into a new subspace through maximizing their covariance [19, 20, 21]. Other machine learning-based algorithms are also widely applied for FDD, including Fisher discriminant analysis, support vector machines, canonical variate analysis, and latent variable regression [22, 23, 2, 11, 24, 1]. A comparison study of some data-driven methods is conducted by Yin et al. [25].

In the past decades, neural network (NN) based soft sensors have been received increasing attention due to their strong capability to process tremendous volume of high-dimensional data. The history of employing NNs in chemical engineering processes for FDD is back to the late 80s by Hoskins and Himmelblau [26] and Venkatasubramanian and Chan [27]. Other NN-based models are also proposed for FDD, such as fuzzy NN, hierarchical artificial NN (ANN) and recurrent NN (RNN) [28, 29, 30]. Eslamloueyan [31] proposed a duty-oriented hierarchical ANN (DOHANN) method for fault diagnosis. In DOHANN, a new subspace is formed for each fault pattern with fuzzy C-means clustering algorithm, and then NNs are trained in each sub-space for fault diagnosis, where the NNs are activated through a supervisory Network. Rad and Yazdanpanah [32] utilized an ICA model for differentiating between faulty and normal samples and another ICA model to detect unseen faults. Then expectation maximization clustering technique assigns samples with known fault classes to clusters, whose labels are determined by their corresponding local multi-layer perceptron. Jiang et al. [33] designed an active learning criterion together with stacked denoising auto-encoder (AE) for fault diagnosis, and it is declared that the method shows superior results with less labeled data. Wu and Zhao [34] proposed a fault diagnosis method based on convolutional NNs (CNNs). Park et al. [35] proposed a combination of AE as fault detection network and long short-term memory (LSTM) as fault diagnosis network. Bidirectional RNNs and LSTM are studied in the work of Zhang et al.

[36] and Chadha et al. [37]. Wang et al. [38] proposed an extended deep belief network to fully exploit valuable information in data for FDD. Agarwal et al. [39] proposed a FDD approach based on the fusion of supervised deep recurrent AE NN and pseudo-random binary signal.

AE has demonstrated its advantages for fault detection, and various AE-based techniques are developed. Malhotra et al. [40] employed LSTM-based AE for anomaly detection for time series data. Yan et al. [41] applied denoising AE (DAE) and contractive AE (CAE) for process monitoring, and concluded that both DAE and CAE obtain more sensitive and robust monitoring performance over PCA. Variational recurrent AE is proposed by Cheng et al. [42] for process monitoring to account for nonlinearity and dynamics in probability space. Yu and Zhang [43] proposed manifold regularized SAE for fault detection to capture geometric information of the data. Yu et al. [44] proposed convolutional LSTM-AE to extract more effective features and applied it for fault detection.

A common feature of these AE-based fault detection techniques is that labeled faulty data cannot be exploited, and they are trained only with the data under normal operating condition. Since only the pattern of normal samples is observed during the training phase, AE-based fault detection techniques are prone to misclassify faulty samples. To address this issue, Munawar et al. [45] applied a negative learning phase, but its training phase is unstable. Yamanaka et al. [46] proposed an autoencoding binary classifiers (ABC), which assumes that the conditional probability of label given input follows the Bernoulli distribution. Ruff et al. [47] proposed a deep semi-supervised anomaly detection (Deep SAD) that tries to diverge anomalies from a predetermined center, which is determined without the information from auxiliary anomalous data set. Additionally, Deep SAD is not suitable to be used together with bounded activation functions. Kim et al. [48] proposed an anomaly detection technique based on generative adversarial network (GAN) and AE by employing two discriminators for normal and anomaly samples. However, this technique is not suitable for re-training existing AEs that have already been applied for fault detection.

In addition to AE-based unsupervised anomaly detection techniques, other techniques are also available, such as the deep one class techniques [49, 50, 51], and self-supervised approaches [52, 53, 54, 55, 56]. Generative deep learning models, such as variational AE (VAE) [57] and generative adversarial net (GAN) [58], are also adopted for anomaly detection purpose. The generative power of VAE enabled An and Cho [59] to employ reconstruction probability for anomaly detection. In AnoGAN developed by Schlegl et al. [60], a GAN is trained to learn the distribution of normal data, and in the testing phase, samples are mapped to the latent space and their fitness to the learned distribution specifies their anomaly scores. Adversarially learned anomaly detection (ALAD) was proposed by Zenati et al. [61] to adversarially learn features according to bi-directional GANs, which shows lower inference cost compared to AnoGAN. Another fast mapping technique for GAN-based anomaly detection was proposed in the work of Schlegl et al. [62]. GANomaly was designed by Akcay et al. [63] that leverages an encoder-decoder-encoder architecture for anomaly detection.

Along with employing generative models as feature extractor or re-constructor, an alternative approach is to generate instances based on the notion of normal instances only. Minimum likelihood GAN was designed by Wang et al. [64] to enhance the anomaly generation ability in GAN's generator. However, it is prone to instability, which is not desirable for anomaly detection. Ngo et al. [65] proposed a Fence GAN (FenceGAN) to generate samples at the boundaries of the normal data distribution by modifying GAN loss, and its discriminator is employed as the anomaly detector. The dispersion loss in FenceGAN, which is designed to stimulate generating samples far from its center of mass, can be sub-optimal since generated samples may be grouped around more than one center. Schulze et al. [66] proposed double-adversarial activation anomaly detection (DA^{3}D) to generate anomalous data using the decoder of a trained adversarial autoencoder (AAE) by training an anomaly generator network. The anomaly generator in DA^{3}D has to fool two different networks in its training phase to produce effective anomalies, which may not be straightforward in some cases and require an exhaustive hyper-parameter selection. Furthermore, the performance of DA^{3}D is influenced by the ability of the trained AAE in producing realistic data samples.

1.2 Research Outcomes

In order to adjust the aforementioned issues in existing techniques, in this work we first propose a self-adversarial autoencoding classifier (SAAC) for anomaly detection with an end-to-end training phase. SAAC takes advantage of generative models as anomaly generators before their convergence, and adversarially trains a network to differentiate between the generated samples and real data. Specifically, the decoder of an AAE is used as the anomaly generator network and the AAE is trained together with an autoencoding binary classifier (ABC) in an adversarial manner. In the testing phase, the trained ABC can be utilized as the anomaly detector. In contrast to previous works, SAAC does not make any modifications to the loss functions of generative models, and it tries to exploit them before their convergence. Besides, SAAC works in an unsupervised manner, and it does not require the access to anomalous samples for training ABC.

As we discussed, labels are hard and time-consuming to obtain. However, once the labelling information is available, it is important to fully exploit it. Thus, in addition to the unsupervised SAAC, we also proposed a supervised technique to employ faulty samples for fault detection. In this work, an extension of AE, referred to as source-aware autoencoder (SAAE), is proposed. SAAE is a type of AEs that is aware of both faulty and normal samples and employs them in its training phase in such a way that it can handle the imbalanced number of normal and faulty samples. Similar to other conventional AEs, the trade-off between precision and recall, as a process-specific decision, can be addressed by employing SAAE as the fault detection network. Moreover, SAAE is capable to detect unseen fault classes. Bidirectional LSTM (BiLSTM) equipped with skip connections is selected for the architecture of SAAEs as the fault detection network. For fault diagnosis, a BiLSTM layer followed by a residual neural work (ResNet) network is designed to avoid the randomness caused by the order of features in CNNs. A sequential framework for combining the fault detection and the fault diagnosis networks is also presented, where the fault diagnosis network takes the results of the fault detection network as its input.

1.3 Thesis Outline

The rest of the thesis is organized as follows. Background information including some relevant techniques is reviewed in Chapter 2. Details and experimental evaluations of the proposed methods for unsupervised anomaly detection and supervised FDD are presented in Chapters 3 and 4, respectively. These sections also summarize the results of comprehensive comparisons between existing approaches in the literature and the proposed methods. Finally, major conclusions are wrapped up in Chapter 5.

Chapter 2

Preliminaries

In this chapter, some important techniques that are relevant to the proposed methods are briefly reviewed.

2.1 Artificial Neural Network

Artificial neural networks (ANNs) are designed to mimic nervous systems, and they consist of learnable parameters to produce predictions through linear and nonlinear functions. The parameters of ANNs are usually acquired by optimizing a cost function, which varies considerably task-to-task. Each neuron in an ANN can be divided into a linear transformation and an activation function. The output of the neuron is the result of applying the activation function to the linear transformation of the inputs in the neuron. Figure 2.1 shows an illustration of a neuron with 2-dimensional inputs and its components.

It is observed that the decision rule, which is performed by a single neuron, is ineffective in dealing with complex tasks, and several neurons and layers are designed in ANNs. In this way, each layer consists of several neurons, and the layers can be stacked one after

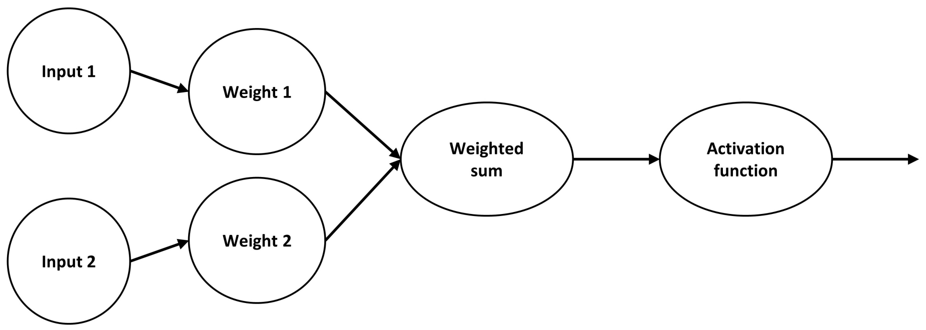

Figure 2.1: Schematic of a neuron in ANNs.

another in such a way that the outputs of the previous layer become the inputs of the next layer, and the input layer is generally the data points. Each neuron in the ANN can have a specific activation function. However, it is more common to have the same activation function for neurons in each particular layer.

Linear activation function is a linear transformation with known weight a and bias b, and can be expressed as follows:

$$\phi_{\text{linear}}(z) = az + b \tag{2.1}$$

In most cases, linear activation function is the identity map with $a = 1$ and $b = 0$. In order to account for non-linearities, Tanh and Sigmoid functions, which are S-shaped, can be utilized. The outputs of Tanh and Sigmoid functions are between -1 to 1 and 0 to 1, respectively, and their expressions are

$$\phi_{\text{Tanh}}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.2}$$

$$\phi(z)_{\text{Sigmoid}} = \mathrm{S}(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

The derivative of Tanh and Sigmoid functions for small and large values are close to 0, which can deteriorate the learning process of the ANNs. Alternatively, rectified linear unit (ReLU) [67] and scaled exponential linear units (SELU) [68] are introduced to address the vanishing and exploding gradient problems. ReLU and SELU functions are formulated as follows:

$$\phi_{\text{ReLU}}(z) = \max(0, z) \tag{2.4}$$

$$\phi_{\text{SELU}}(z) = \lambda_{\text{SELU}} \times (\max(0, z) + \min(0, \alpha_{\text{SELU}} \times (e^z - 1))) \tag{2.5}$$

where $\lambda_{\text{SELU}} \approx 1.05070$ and $\alpha_{\text{SELU}} \approx 1.67326$.

2.2 Long Short-Term Memory

In contrast to feedforward NNs, previously stored information in RNNs can be used to calculate the next outputs. Thus, RNNs can deal with time-series data by considering the time dependencies deep-rooted in the data. Long short-term memory (LSTM) is a variation of RNNs that aims to address gradient vanishing and gradient explosion problems arising during the training of traditional RNNs [69]. LSTMs are effective in many applications such as language modeling and translation, audio analysis, and traffic forecasting [70, 71, 72].

Each LSTM cell consists of input $\mathbf{x}_{[t]} \in \mathbb{R}^p$, cell state $\mathbf{c}_{[t]} \in \mathbb{R}^n$, hidden state $\mathbf{h}_{[t]} \in \mathbb{R}^n$, forget gate $\mathbf{f}_{[t]}$, input gate $\mathbf{i}_{[t]}$, and output gate $\mathbf{o}_{[t]}$, where p and n denote the number of input features and the number of neurons in the LSTM cell, respectively. $\mathbf{c}_{[t]}$ allows the network to extract long- and short-term time dependencies. The nonlinear gates $\mathbf{f}_{[t]}$, $\mathbf{i}_{[t]}$ and $\mathbf{o}_{[t]}$ control the flow of information, and decide what information should be kept from the previous memory once a new input vector $\mathbf{x}_{[t]}$ and the preceding hidden state $\mathbf{h}_{[t-1]}$ is observed. The mathematical formulation of these gates are presented as

$$\mathbf{f}_{[t]} = \mathrm{S}(\mathbf{W}_{g,f}\mathbf{x}_{[t]} + \mathbf{W}_{h,f}\mathbf{h}_{[t-1]} + \mathbf{b}_f) \tag{2.6}$$

$$\mathbf{i}_{[t]} = \mathrm{S}(\mathbf{W}_{g,i}\mathbf{x}_{[t]} + \mathbf{W}_{h,i}\mathbf{h}_{[t-1]} + \mathbf{b}_i) \tag{2.7}$$

$$\mathbf{o}_{[t]} = \mathrm{S}(\mathbf{W}_{g,o}\mathbf{x}_{[t]} + \mathbf{W}_{h,o}\mathbf{h}_{[t-1]} + \mathbf{b}_o) \tag{2.8}$$

where $\mathrm{S}(\cdot)$ is the element-wise sigmoid function. With Eqs. (2.6) - (2.8), $\mathbf{c}_{[t]}$ and $\mathbf{h}_{[t]}$ can

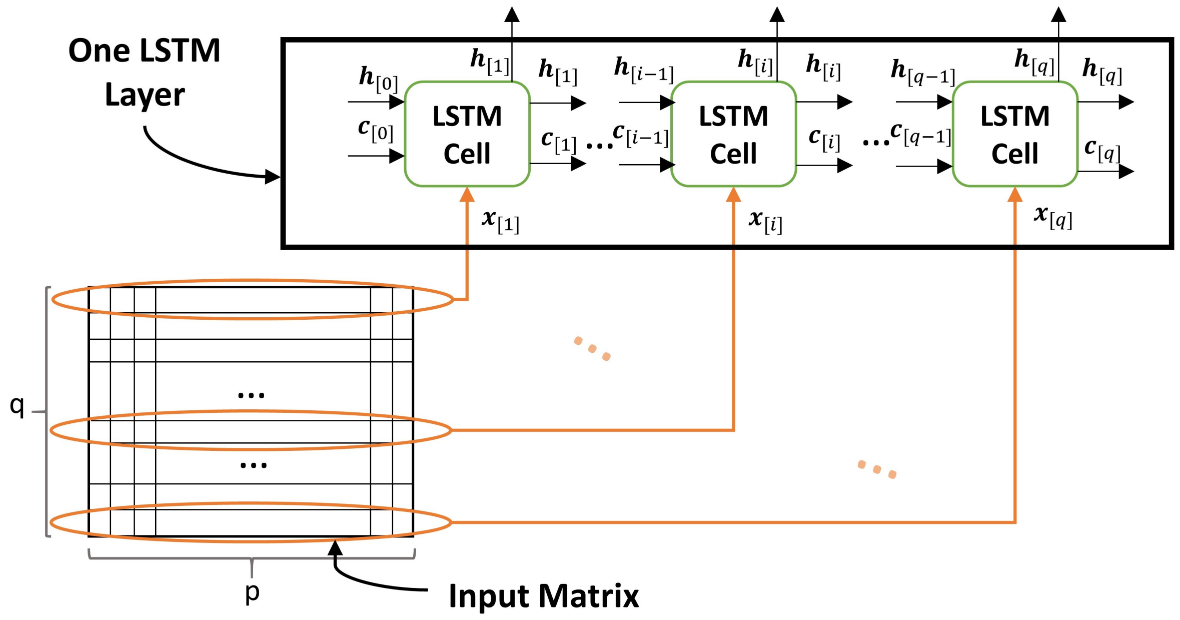

Figure 2.2: Schematic of many to many design of a LSTM layer.

be calculated as follows:

$$\mathbf{c}_{[t]} = \mathbf{c}_{[t-1]} \odot \mathbf{f}_{[t]} + \tanh(\mathbf{W}_{c,c}\mathbf{x}_{[t]} + \mathbf{W}_{h,c}\mathbf{h}_{[t-1]} + \mathbf{b}_c) \odot \mathbf{i}_{[t]} \tag{2.9}$$

$$\mathbf{h}_{[t]} = \tanh(\mathbf{c}_{[t]}) \odot \mathbf{o}_{[t]} \tag{2.10}$$

where $\mathbf{W}_{g,f}, \mathbf{W}_{g,i}, \mathbf{W}_{g,o}, \mathbf{W}_{c,c} \in \mathbb{R}^{n\times p}$ and $\mathbf{W}_{h,f}, \mathbf{W}_{h,i}, \mathbf{W}_{h,o}, \mathbf{W}_{h,c} \in \mathbb{R}^{n\times n}$ are the weights of the cell, $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_o, \mathbf{b}_c \in \mathbb{R}^n$ are symbolized biases of the the cell, $\odot$ indicates element-wise product and tanh denotes element-wise hyperbolic tangent function. $\mathbf{c}_{[t]}$ and $\mathbf{h}_{[t]}$ are then fed into the next cell. The structure of one LSTM layer with many to many design is shown in Figure 2.2, where the initial hidden state $\mathbf{h}_{[0]}$ and cell state $\mathbf{c}_{[0]}$ are set to zero vectors. Output's activation function and forget gate are considered as the most essential component of a LSTM cell, and removal of them will noticeably deteriorate model's performance [73].

2.3 Bidirectional Long Short-Term Memory

A unidirectional LSTM unit takes the previous hidden state, cell state, and current variable vector as inputs and returns current hidden and cell states. That is, it only takes inputs in a chronological order, and thus the output of each LSTM cell depends only on its preceding cells. However, data in reverse chronological order also contribute to the extraction of more relevant features in some applications.

Schuster and Paliwal [74] extended RNNs to bidirectional RNNs, where two RNNs are trained with the right to left and left to right sequence data, and thus the resulting outputs contain information from both time directions. The effectiveness of the bidirectional networks has been proven in many applications [75, 76, 77, 78].

For a bidirectional LSTM network (BiLSTM), two hidden states and two cell states in both time directions can be connected through several operations such as addition, linear combination, average and merge. In this work, concatenation is selected. The hidden states that are calculated from forward and backward time directions are denoted by $\mathbf{h}_{[t],f}$, and $\mathbf{h}_{[t],b}$. In this way, the hidden state of this cell ($\mathbf{h}_{[t]}$) can be computed as follows:

$$\mathbf{h}_{[t]}^{\top} = [\mathbf{h}_{[t],f}^{\top}, \mathbf{h}_{[t],b}^{\top}] \tag{2.11}$$

where superscript $\top$ denotes transpose operation. Initial hidden and cell state in both time directions are the same and equal to zero matrices.

2.4 Autoencoder

Autoencoder (AE), as a type of NNs, consists of an encoder and a decoder, where the encoder learns a nonlinear transformation to a latent space while the decoder tries to reconstruct the inputs from the latent space [79]. The structure of an AE is demonstrated in Figure 2.3, in which each neuron is connected to all neurons in its previous and next

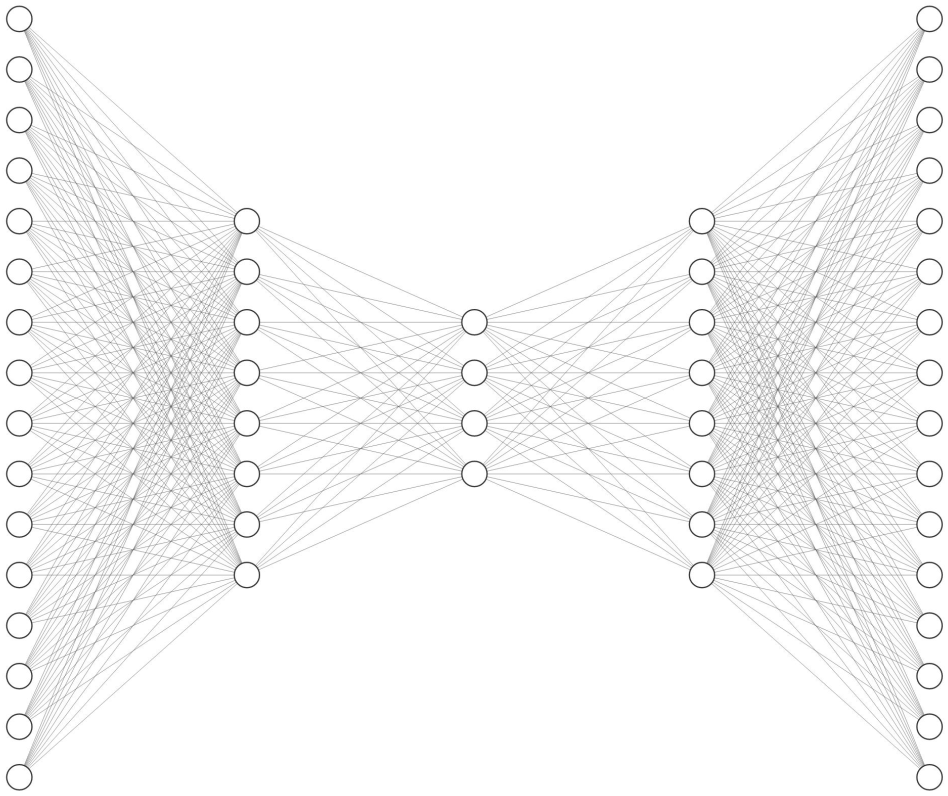

Figure 2.3: The structure of an AE with fully connected layers.

layers. AEs have gained great attention and applied for many tasks such as feature learning, information retrieval and anomaly detection [80, 81, 82].

In terms of anomaly detection, AEs have superior attributes over their counterpart technique called binary classification. The network that is trained with normal data (with or without a small proportion of anomalies) captures normal behaviors, and it is assumed that unseen anomaly patterns are different from normal ones [83]. AEs are effective without faulty data, and thus they are inherently capable of detecting unseen fault types to some extend. This attribute is more precious when no or a few faulty data are available or faulty data are expensive to achieve or an imbalance data set is available. Moreover, handling the trade-off between precision and recall, which could be a key to avoid disastrous events and economic losses due to the nature of the process, is also feasible with AEs. The details of evaluation metrics such as precision and recall are reviewed in Section 2.14.

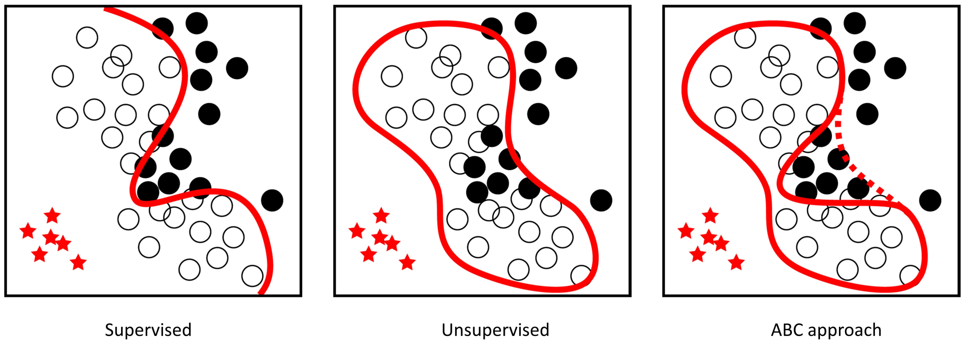

Figure 2.4: Comparison of (a) supervised, (b) unsupervised and (c) ABC in detecting unseen and known anomalies. The decision boundaries are shown by red lines. White circles, black circles, and red stars represent normal data points, known anomalies, and unseen anomalies, respectively.

2.5 Autoencoding Binary Classifier

AEs have demonstrated their potential for unsupervised anomaly detection, and thus they are effective to detect unseen anomaly types that are not available in the training phase. A common way for AE-based anomaly detection methods is to train an AE for the reconstruction of the normal samples. It is expected that the trained AE fails to reconstruct the anomalous data samples well. However, it is observed that trained AEs can reconstruct anomalous data samples that share similar patterns with the normal ones. Besides, it is declared that having access to anomalies in the training phase contributes to detecting anomalies [84]. Thus, designing AEs that can work in unsupervised settings and simultaneously exploit the information from the labelled anomaly samples is desired.

ABC is a supervised AE with the assumption of Bernoulli distribution for the conditional probability of the label variable given a data point. Figure 2.4 from the work of

Yamanaka et al. [46] compares the traditional supervised and unsupervised techniques with ABC. ABC is designed to form the boundaries of known normal samples with both unseen and known anomalies. Assume that a data set $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ is given, where n is the number of data points, and $x^{(i)} \in \mathbb{R}^m$ and $y^{(i)} \in \{0, 1\}$ denote the ith data point and its corresponding label. Abnormal and normal data points are labelled as 0 and 1, respectively. Let $f_\theta(x_i) : \mathbb{R}^m \to [0, 1]$ be an approximation for the conditional probability of x_i belonging to the normal class given x_i, where $f_\theta(x_i) \approx p(y_i = 1|x_i)$. The loss function for ABC can be expressed as

$$\mathcal{L}(x_i, y_i) = -y_i \log f_\theta(x_i) - (1 - y_i) \log(1 - f_\theta(x_i)) \tag{2.12}$$

f_θ can be defined in various ways as a function of AE's parameters. One common choice for $f_\theta(x)$ is $f_\theta(x) = e^{-\frac{\mathcal{L}_{\text{rec}}(x)}{\lambda_{\text{scale}}}}$, where $\mathcal{L}_{\text{rec}}(x)$ denotes the reconstruction error for x in the AE, and λ_{scale} is a normalization hyperparameter to account for the scale of $\mathcal{L}_{\text{rec}}(x)$. In this case, Eq. (2.12) can be written as

$$\mathcal{L}(x_i, y_i) = \underbrace{\frac{1}{\lambda_{\text{scale}}} y_i \mathcal{L}_{\text{rec}}(x_i)}_{\text{for normal samples}} - \underbrace{(1 - y_i) \log(1 - e^{-\frac{\mathcal{L}_{\text{rec}}(x_i)}{\lambda_{\text{scale}}}})}_{\text{for anomalies}} \tag{2.13}$$

The ABC is trained to minimize Eq. (2.13). The first term in the equation is for reducing reconstruction error for normal samples, while the second term stimulates the ABC to increase the reconstruction error for anomalies. In the special case when no anomalies are available, ABC is equivalent to the traditional AE.

It is worth pointing out that λ_{scale} is newly designed in our work, which is observed to improve SAAC model's performance, and it was not introduced in the work of Yamanaka et al. [46].

2.6 Generative Adversarial Network

Generating samples that resemble a data distribution could help in enriching data sets. One way is to train a generator to generate samples based on random noise, and a classifier (discriminator network) to decide on the quality of the generated and real samples. This is the underlying idea of the generative adversarial networks (GANs) [58], in which the discriminator and generator networks are trained to compete with each other and gradually become stronger in this competition.

Assume that a data set $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{n}$ is given, where n is the number of data points, and $\mathbf{x}^{(i)} \in \mathbb{R}^m$ denotes the ith data point. Random noise $\mathbf{z} \in \mathbb{R}^{p_{noise}}$ is the input of generator (G), and output m-dimensional samples. Assume data distrubution and random noise distribution are denoted as follows: $\mathbf{x} \sim p_{data}(\mathbf{x})$, $\mathbf{z} \sim p_z(\mathbf{z})$. The objective of GAN is to generate samples $\hat{\mathbf{x}}$ similar to real samples $\mathbf{x}$ from $\mathbf{z}$. The discriminator (D) in GAN is a binary classifier that decides on the similarity of $\mathbf{x}$ and $\hat{\mathbf{x}}$. That is, $D : x \rightarrow [0, 1]$ classifies generated and real samples, and output 1 for real samples and 0 for generated ones as follows:

$$D(\mathbf{x}) := \begin{cases} 1 & \text{for real sample } \mathbf{x} \\ 0 & \text{for fake (generated) sample } \mathbf{x} \end{cases} \tag{2.14}$$

In the case that D classifies fake samples as fakes (output 0) and real samples as real (output 1), a perfect D is obtained. However, this indicates the poor performance of G in generating samples similar to real ones. G and D are trained adversarially to compete with each other and gradually improve themselves. That is, G tries to fool D by generating realistic samples while D differentiates between the generated and real samples. The structure of GAN is provided in Figure 2.5 from the work of Ghojogh et al. [85]. G generates samples merely based on the random noise $\mathbf{z}$ without having access to real data $\mathbf{x}$. In this way, G receives feedbacks from D, while D updates itself by taking both $\mathbf{x}$ and $\hat{\mathbf{x}}$ as its inputs. The loss function for training a GAN can be expressed as

$$\min_{G} \max_{D} \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{2.15}$$

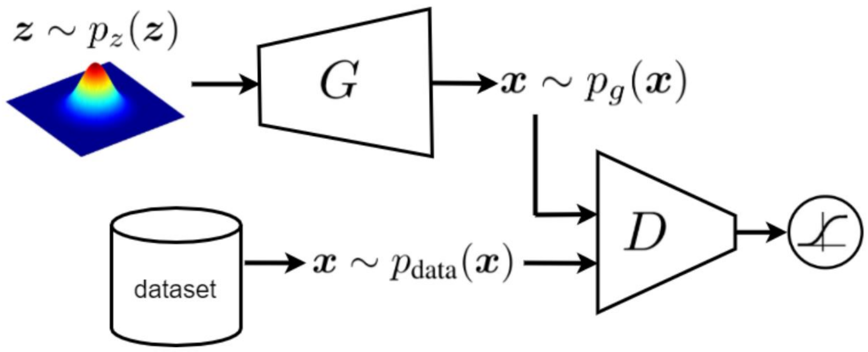

Figure 2.5: The combination of different networks in GAN.

where $\mathbb{E}[.]$ denotes the expectation operation. In Eq. (2.15), the first term is for the discriminator only, and it tries to increase the output of D for real data since real data samples have the label 1. In the second term of Eq. (2.15), the output of the discriminator from the generated samples from the random noise is calculated as $D(G(\mathbf{z}))$. The discriminator tries to maximize $\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$ to assign labels closer to 0 to the generated samples. On the other hand, the generator tries to minimize $\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$ to fool the discriminator, and generates samples which look realistic by the discriminator. D and G can be trained iteratively by employing mini-batches of real and generated data.

In general, two deep neural networks can be designed as the architecture of the generator and discriminator. The last layer of the discriminator can be activated with a Sigmoid function, and these networks can be trained by backpropagation in deep learning. In practical scenarios, the distribution of the random noise $\mathbf{z}$ is selected to be Gaussian.

Several extensions of GAN are proposed in the literature. Conditional GAN [86] is designed to generate samples by conditioning both generator and discriminator on label information. In particular, the label information is provided as additional inputs to both networks. Assume that the additional label information is provided as $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$, where $\mathbf{y}^{(i)}$ denotes the ith data point one-hot encoded label vector. The loss function of conditional GAN can be formulated as follows:

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log(D(\mathbf{x} \mid \mathbf{y}))] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z} \mid \mathbf{y})))] \tag{2.16}$$

Deep convolutional GAN (DCGAN) [87] enabled training deeper GANs for generating images with higher resolution. Strided convolutions are used instead of pooling layers in DCGAN, and layers are convolutional networks and no fully-connected layer is employed. Batch normalization, which is introduced in Section 2.11, is applied to all layers except the last layer of the generator and the first layer of the discriminator. In DCGAN, LeakyReLU [88] is the activation function for the layers in the discriminator, and the output layer of the generator is activated by Tanh and its other layers have ReLU activation function.

A common limitation in GANs is that they are prone to the problem of mode collapse [89] in cases that the data distributions $p_{data}(\mathbf{x})$ has several modes. As an example, a GAN that is trained on all 10 digits of the MNIST data set, may fail to generate samples of some digits. Wasserstein GAN [90] employs Earth-Mover distance to introduce an alternative to GANs. WGAN avoids the problem of mode collapse and has interpretable learning curves. WGAN replaces the discriminator of GANs by a critic network, and imposes Lipschitz constraint on the critic. However, poor sample generation or failure to converge are the issues of WGAN. As the further step, WGAN with gradient penalty [91] is designed, which provides an alternative method to weight clipping in WGAN, and it stabilizes the training phase of WGAN.

2.7 Adversarial Autoencoder

Makhzani et al. [92] augmented reconstruction-based autoencoders by an adversarial training procedure, and referred to it as adversarial autoencoders (AAE). Motivated by the adversarial training of GANs, AAE tries to match its code layer with an arbitrary prior distribution, and as shown in the structure of AAE in Figure 2.6, a discriminator network is integrated on top of the code vector of the autoencoder. The encoder of AAE transforms the input to the code layer and the decoder learns a transformation to the data distribution. The discriminator in AAE matches the code layer and prior distribution through adversarial training while its encoder and decoder attempt to minimize the reconstruction error. The training of an AAE model consists of three phases: i. updating the parameters of its encoder and decoder to minimize the reconstruction error; ii. updating the parameters of the discriminator to differentiate between the code layer and the prior distribution; and iii. updating the parameters of the encoder to fool the discriminator.

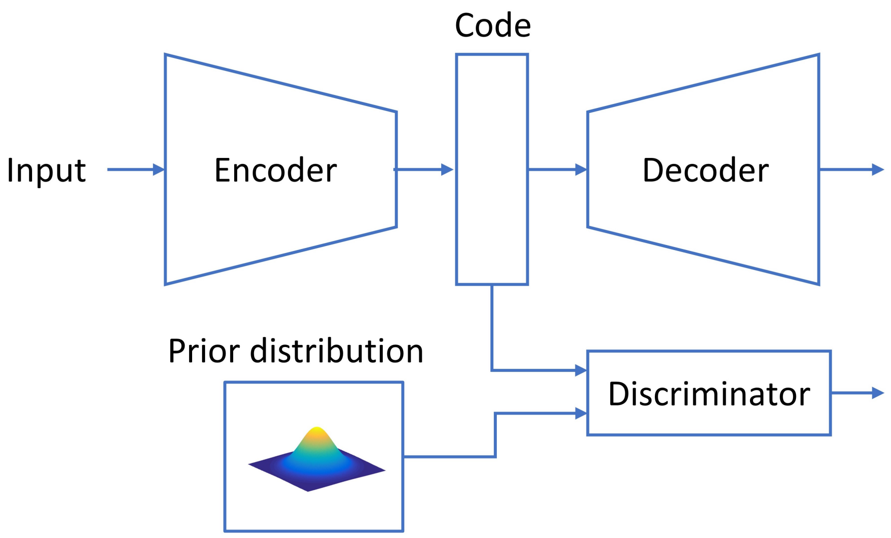

Figure 2.6: The integrated structure of encoder, decoder and discriminator in an AAE.

AAE has been employed for a variety of tasks such as video anomaly detection and localization [93], group anomaly detection [94], and generation of new molecules [95]. While it is used for supervised anomaly detection [96], AAE is also employed in an unsupervised way. For instance, Jang et al. [97] employed AAE for unsupervised fault detection due to its effectiveness in extracting features that represent the data manifold well.

2.8 Generate to Detect

As previously discussed, GANs are generally known for their ability to generate realistic data points with respect to a target class. The training of a GAN is based on a min-max game between competing networks in an adversarial manner. During the training, the generator (G), one component of the GAN, gradually learns the distribution of real samples, to achieve tasks such as augmenting the target class data. After convergence, G is expected to generate samples with a distribution close to the distribution of the target class. Nevertheless, before the convergence, G tends to generate irregularities due to the fact that it has not learned the distribution of the target class yet. Alternatively, generate to detect (G2D) model [98] regards these as anomalies. When G fails to generate normal samples, it is referred to as an irregularity generator. Then a binary classification network is trained to differentiate between normal samples and anomalies using the normal data and the constructed anomaly sets by irregularity generators, through minimizing the binary cross-entropy loss. The output of this binary classification network is defined as the anomaly score of G2D. WGAN is employed due to its more stable training phase and interpretable learning curves compared to conventional GANs in G2D. Figure 2.7 from [98] shows the loss values during the training of a WGAN on the MNIST data set. This figure shows the distinction between random irregularities, irregularities close to boundaries, and inliers.

2.9 Skip Connection

Deep neural networks have demonstrated superior results over their shallower counterparts in various areas such as image classification and object detection [99, 100]. However, training deep NNs are susceptible to gradient vanishing issue, and different methods are proposed to address this issue, such as ResNet [99], highway network [101], stochastic depth [102], FractalNet [103] and DenseNet [104].

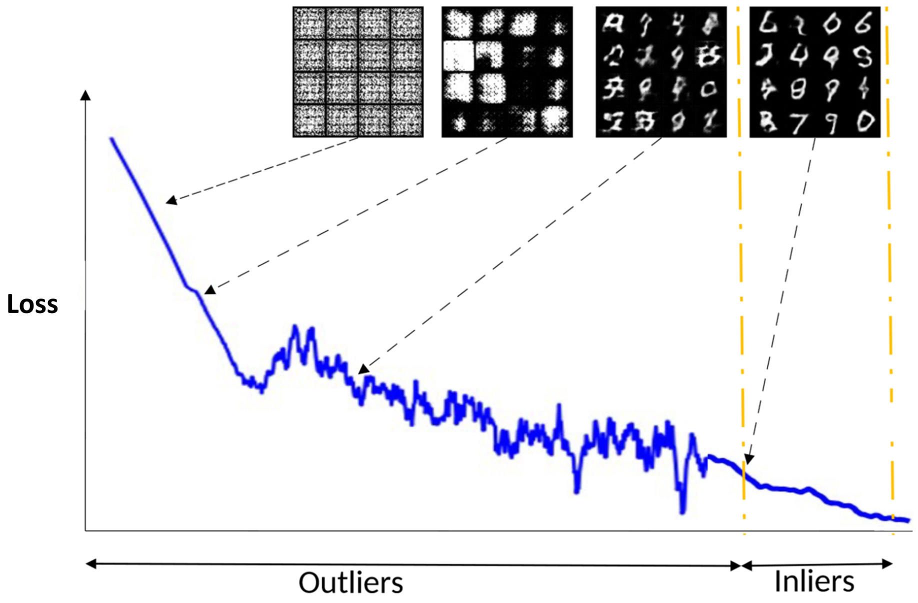

Figure 2.7: The loss values of a WGAN, which is trained on the MNIST data set, during its training phase.

Skip connections (also known as residual connection) is an efficient method to address the gradient vanishing problem. A skip connection skips some nonlinear layers in NN, and provides another path for the backpropagation of gradient. Assume that the input ω undertakes a nonlinear transformation through a function Φ. The output resulting from a skip function between ω and $\Phi(\omega)$ is depicted in Figure 2.8. A skip connections links the top layer to the bottom layer, and thus facilitates the transmission of gradient and information throughout the network [105]. It is noted that proper usage of skip connections not only alleviates the difficulties of training very deep NNs, but also can improve their generalization [106]. Orhan and Pitkow [107] also indicated that singularity elimination is a beneficial attribute of skip connection for training deep NNs.

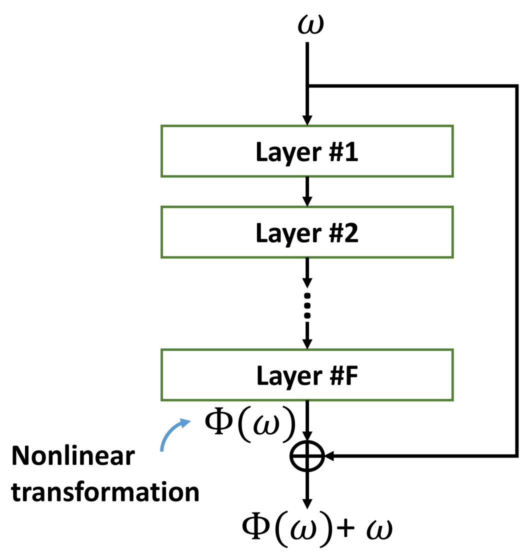

Figure 2.8: Skip connection between input and output layers.

2.10 Dropout

Over-fitting occurs when the performance of NNs differs noticeably between training and testing samples. An indication of over-fitting in FDD is that a divergence is displayed in the precision and recall values for training and testing sets. Srivastava et al. [108] proposed dropout to overcome the over-fitting issue, where each unit and its connections are removed from the NN with the probability of p_{drop} at each training iteration. Moreover, outputs are also scaled by $\frac{1}{1-p_{\text{drop}}}$ for training the network. It is noted that dropout is only applied for the training phase while for the testing phase it is ignored. Dropout technique is illustrated in Figure 2.9.

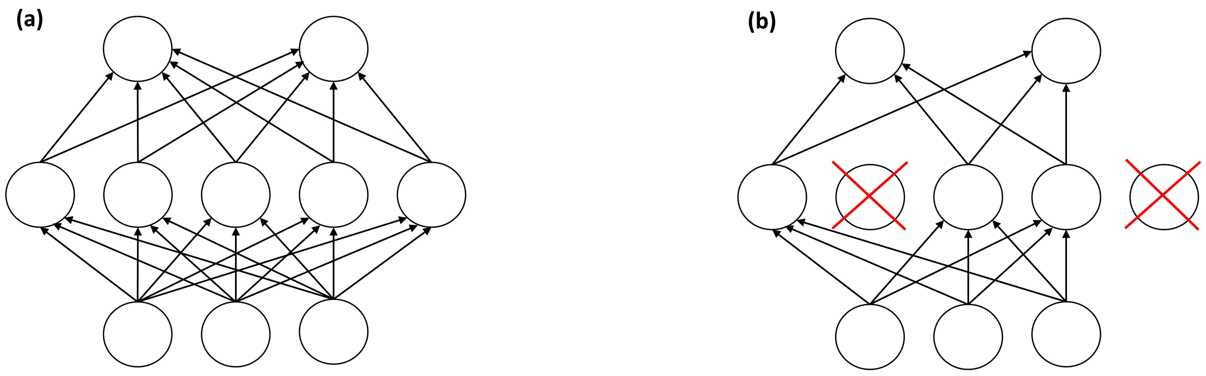

Figure 2.9: A NN with and without dropout is illustrated in subfigures a and b, respectively.

2.11 Batch Normalization

Batch normalization that aims to accelerate network training is proposed by Ioffe and Szegedy [109]. In this method, incoming batch of inputs are scaled and shifted to have zero-mean and univariance. Then, they undergo two re-scaling and re-shifting transformations. For a batch of input $\mathcal{B} = \{x^{(i)}\}_{i=1}^{n_b}$, its mean and variance are calculated by

$$\mu_{\mathcal{B}} = \frac{1}{n_b}\sum_{i=1}^{n_b} x^{(i)}, \quad \sigma_{\mathcal{B}}^2 = \frac{1}{n_b}\sum_{i=1}^{n_b}(x^{(i)} - \mu_{\mathcal{B}})^2 \tag{2.17}$$

Then, elements of $\mathcal{B}$ are normalized to have zero-mean and univariance.

$$\bar{x}^{(i)} = \frac{x^{(i)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2.18}$$

where ϵ is a constant designed for numerical stability. Further, two learnable parameters γ and ψ are employed in Ioffe and Szegedy [109] to scale and shift the normalized values as follows:

$$\hat{y}^{(i)} = \gamma\bar{x}^{(i)} + \psi \tag{2.19}$$

During the training phase, general mean and variance are estimated through exponentially weighted moving average with momentum value of m_{BN}, which are used for the normalization in the testing phase. Layers or neurons of NNs can be equipped with batch normalization.

2.12 Softmax Activation Function and Cross-Entropy Loss

Softmax activation function can be employed to represent the probability distribution of the predicted classes, which is widely adopted in multi-class classification tasks. It converts an n dimensional vector to another n dimensional exponentialized vector, which is normalized by the sum of elements of the elementwise exponentialized vector. Mathematically, the softmax activation function $\sigma : \mathbb{R}^n \rightarrow [0,1]^n$ is expressed as

$$\sigma(\mathbf{z}, i) = \frac{\exp(z_i)}{\Sigma_{j=1}^{n} \exp(z_j)} \tag{2.20}$$

where $\mathbf{z} \in \mathbb{R}^n$ denotes the input vector, and z^i is the i^{th} variable of $\mathbf{z}$.

Weighted categorical cross-entropy loss can be employed in various multi-class classification algorithm, and it is formulated as follows:

$$\text{loss}(\mathbf{y}, \mathbf{p}, \mathbf{w}) = -\Sigma_{c=1}^{n} w_c \times y_c \times \log(p_c) \tag{2.21}$$

where $\mathbf{w} \in [0, \infty)^n$ is the weight vector, and $\mathbf{y}$ and $\mathbf{p}$ are lable and probability vectors. Eq. (2.20) is usually served as the activation function of the last layer of a multi-class classification NN, which is often trained to minimize Eq. (2.21). During the testing phase, each data sample is assigned to the class that has the highest corresponding probability.

2.13 Adam Optimizer

Gradient descent is a computationally effective method for minimizing or maximizing objective functions since it requires first-order partial derivatives. In dealing with large data sets, the objective function can be optimized with subsets of the data set, and this leads to leveraging stochastic gradient descent (SGD) for optimization problems. It is common to have high dimensional parameters space, and large data sets in the field of deep learning, and Kingman and Ba [110] proposed Adam as a memory-efficient stochastic optimization technique, which is suitable to be employed together with mini-batches, by combining the potentials of AdaGrad [111] and RMSProp [112] to deal with sparse gradients and non-stationary objectives.

Assume that the function $f(\theta)$ is differentiable with respect to parameter θ for a minimization optimization problem. Let $f_i(\theta)$ denote the stochastic function at time-step i. Adam requires three hyper-parameters: α_{Adam} as the step size, and β_1 and β_2 as the exponential decay rates for the 1^{st} and 2^{nd} raw moments. Exponential moving averages of the gradient (m_t) and squared gradient (v_t) are initialized as 0's vectors, and β_1 and β_2 are used to determine their updated values. At each iteration, parameter θ_t can be updated through the following steps:

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}) \tag{2.22}$$

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{2.23}$$

$$g_t \leftarrow \beta_2 \cdot g_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{2.24}$$

$$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t) \tag{2.25}$$

$$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t) \tag{2.26}$$

$$\theta_t \leftarrow \theta_{t-1} - \alpha_{\text{Adam}} \cdot \hat{m}_t / (\hat{v}_t^{0.5} + \epsilon) \tag{2.27}$$

where ϵ is a parameter for stabilization. As default values, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The experimental results in the work of Kingman and Ba [110] show the success of Adam in the optimization of deep learning models.

2.14 Evaluation Metrics

In order to make a fair comparison of different algorithms, recall, precision and F_β-score are selected as evaluation metrics, which are formulated as

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.28}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.29}$$

$$F_\beta = (1 + \beta^2)\frac{\text{recall} \times \text{precision}}{\beta^2 \times \text{recall} + \text{precision}} \tag{2.30}$$

where TP and TN denote the number of correctly classified samples with positive and negative labels, respectively. FP and FN refer to misclassified samples with actual negative and positive labels. F_β is a weighted F measure, and β is a non-negative real constant, where F_1 ($\beta = 1$) is chosen in this work. F_1 together with recall and precision provide a reasonable set of metrics for evaluating the classification performance.

For the comparison between unsupervised anomaly detection techniques, area under the receiver operating characteristics curve (AUC-ROC) and maximum balanced accuracy (mBA) can be adopted as the evaluation metrics. In ROC curve, each data point illustrates true positive and false positive rates at a fixed threshold, and AUC-ROC represents a summary of the ROC curve. On the other hand, mBA displays the performance at a threshold where the arithmetic mean between sensitivity and specificity is maximum. A perfect anomaly detection technique has AUC-ROC and mBA of 1.

Chapter 3

SAAC for Unsupervised Anomaly Detection

Anomaly detection models are developed with a collected training data set, which contains normal samples only or a mixture of normal and faulty ones. These models can obtain satisfactory detection results when the test samples share the same anomaly classes as in the training data. However, in some practical scenarios, the existence of unseen anomalies in the test phase is common, since it is hard to exhaust all types of anomalies, and unexpected failures may happen. Thus, unsupervised anomaly detection techniques are proposed to account for unseen anomalies, which are generally trained using data collected under normal conditions (with or without a small proportion of anomalies). On the other hand, supervised AEs exploit both normal and anomalous samples, and demonstrate that unseen anomalies can be more detectable if the auxiliary data set of anomalous samples are given. Thus, considering the fact that acquiring a data set containing anomalous samples is an expensive task in some cases, in this chapter we focus on the design of unsupervised anomaly detection methods from a different perspective.

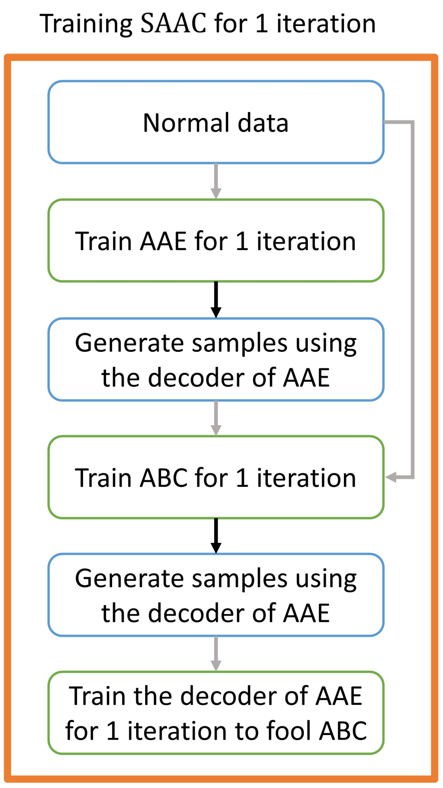
Training SAAC for 1 iteration
Normal data
Train AAE for 1 iteration
Generate samples using the decoder of AAE
Train ABC for 1 iteration
Generate samples using the decoder of AAE
Train the decoder of AAE for 1 iteration to fool ABC

Figure 3.1: The required steps in training SAAC for 1 iteration.

3.1 The Proposed Methods

In our proposed self-adversarial autoencoding classifier (SAAC) model, fake anomalies are generated by only observing normal samples, and an anomaly detector network is trained to leverage both normal samples and fake anomalies. SAAC includes two main parts, i.e., anomaly generator and anomaly detector. The objective of the anomaly generator is to generate fake anomalies from the normal data while the anomaly detector differentiates them from normal data. AAE is selected as the anomaly generator in SAAC, since it can generate anomalies before convergence, and ABC is selected as the anomaly detector. The details of AAE and ABC are described in sections 2.7 and 2.5 respectively, and they are

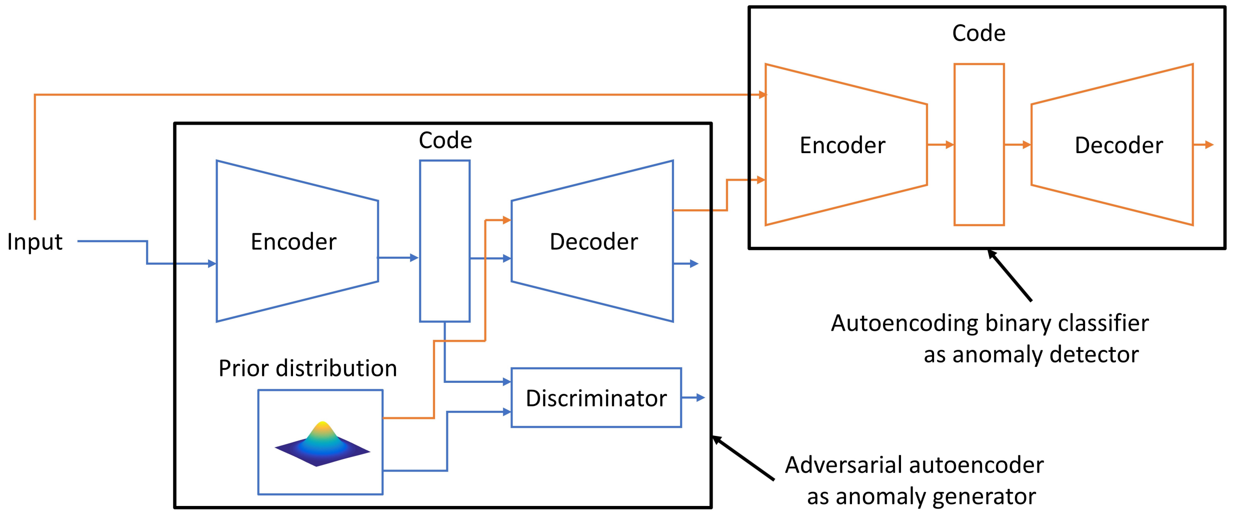
Code
Encoder
Decoder
Input
Code
Encoder
Decoder
Prior distribution
Discriminator
Autoencoding binary classifier as anomaly detector
Adversarial autoencoder as anomaly generator

Figure 3.2: The overall architecture of SAAC. SAAC consists of an AAE as the anomaly generator and an ABC as the anomaly detector, which are trained in an adversarial manner.

trained in an adversarial manner. In our work, the deterministic encoder is selected in AAE, in which the data distribution is considered as the source of stochasticity, and the prior distribution is chosen as a Gaussian distribution with zero-mean and unit-variance.

In each iteration, the AAE is trained to learn the distribution of the normal samples, and fake anomalies are generated using the decoder of the AAE from its prior, and the ABC is trained on the generated samples (for anomalies) and normal data to distinguish them. Then, new samples are generated again by the decoder of the AAE, which are used to update the parameters of the decoder of the AAE to minimize their reconstruction errors by the ABC as the adversarial training phase. The details of one iteration in SAAC are summarized in Figure 3.1, and the whole architecture of SAAC is depicted in Figure 3.2.

During the training phase of SAAC, the AAE generates fake anomalies, which are used to assist the ABC to learn the decision boundary to classify normal and anomalous samples. For the first training iteration, the AAE generates random samples, and as the training evolves, the AAE learns to generate samples more similar to real ones which are however

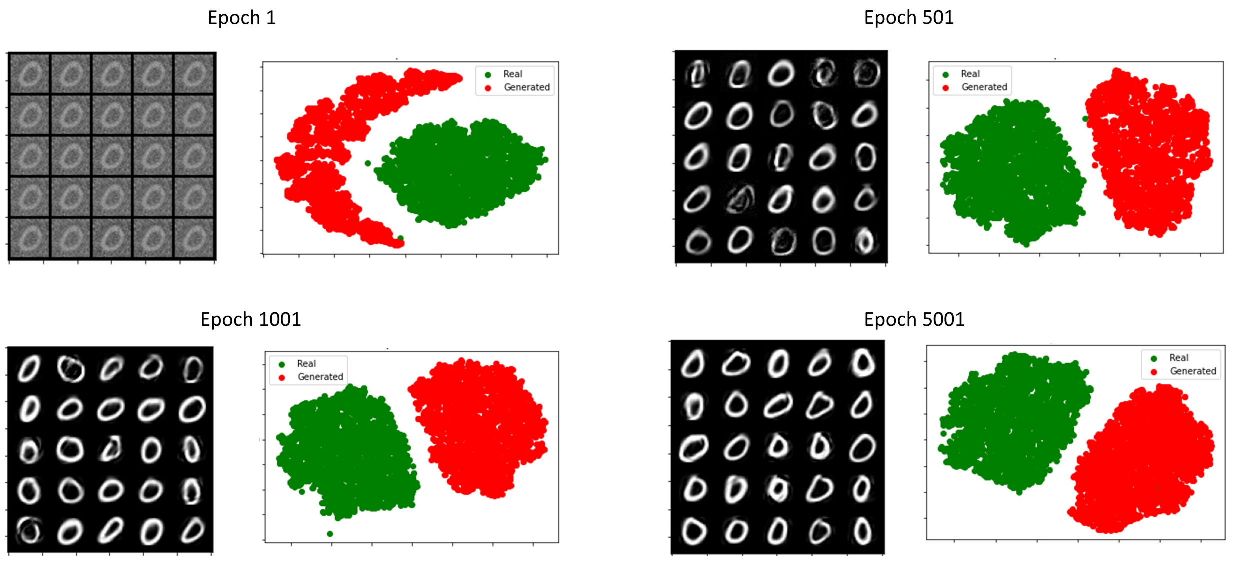

Figure 3.3: t-SNE visualization of generated samples by a simple AAE in different training epochs. This AAE is trained on digit "0" of MNIST data set.

still anomalies. After the convergence is achieved, the AAE has learned the distribution of the normal samples and can generate samples following the same distribution. It is noted that for all iterations, the fake anomalies are utilized to train the ABC anomaly detector, and the training of the ABC model stops before the convergence of AAE. In the testing phase, the reconstruction error by the trained ABC is considered as the anomaly score for a test sample.

The adversarial training between the AAE and ABC is designed to prevent finding shortcut solutions by the ABC for the task of anomaly detection. In this way, the ABC is trained based on the updated AAE, which tries to minimize the reconstruction error of its outputs by the ABC in the adversarial training step, and thus, should continuously update its decision rules during the training phase. Another attribute of the adversarial training is to generate samples by the decoder of the AAE according to the prior distribution and update the decoder according to the feedback of the ABC. Consequently, blind spots of the prior distribution in the decoder of the AAE can be avoided, and this attribute makes the decoder more effective.

Remark 1. In general, a complex structure of the AAE is not required in SAAC since the AAE does not need to converge to generate realistic samples. In our experiments, simpler AAE architectures are designed, which are incapable of generating realistic samples. Figure 3.3 illustrates t-SNE [113] visualization of generated sample by a simple AAE, which was trained on digit '0' of MNIST data set. From this figure, it is observed that the generated samples by the AAE and real normal data can be differentiated even after 4000 training epochs.

Remark 2. Prior to the proposed work of SAAC, ABC was employed in supervised settings, in which both anomalies and normal samples were accessible. Even though ABC shows superior performance in detecting unseen anomalies, its applicability was limited since it requires real anomalies. Moreover, AAE was primarily utilized for data augmentation for anomalies or unsupervised anomaly detection. Anomalies are required in the former one. For the latter one, AAE is trained to learn the distribution of the normal samples and this task is not anomaly detection-oriented, which makes its performance sub-optimal. In contrast to previous works related to ABC and AAE, SAAC is a novel anomaly detection technique that removes the necessity of having access to anomalies in ABC and it is particularly trained for the task of anomaly detection.

3.2 Experimental Results and Discussions

In order to evaluate the performance of the proposed SAAC, three benchmark data sets are chosen, namely the Tennessee-Eastman Process (TEP) [114], Credit Card (CC) [115] and CIFAR10 data [116], which range from chemical process data to image data. These data sets differ in the number of features, existence of class imbalance, and variability in their domains, which can provide more insights for the comparison.

For each data set, SAAC is compared to several competing techniques from the literature. In the cases that we use different data preparation steps in our experiments to the corresponding literature, those techniques are re-implemented. Hence, in the subsequent subsections, the results are generally based on our experiments unless it is explicitly stated.

For the TEP and CC data sets, training and testing data sets are transformed into the range of $[-1, 1]$ by MinMax transformation based on the normal training data. The details of the hyper-parameters and training settings are provided in the corresponding subsections. To be consistent with the previous works, similar pre-processing steps to the work of Massoli et al. [117] are applied for CIFAR10.

3.2.1 The Tennessee-Eastman Process

In TEP, four reactants and an inert are fed into the process, and two liquid products and a byproduct are formed through four exothermic reactions. The five main unit operators involved in TEP are reactor, condenser, compressor, separator, and stripper, and the schematic of TEP is presented in Figure 3.4. This process contains 41 measured variables and 11 manipulated variables. The description of manipulated and measured variables are provided in Tables 3.1 and 3.2, respectively.

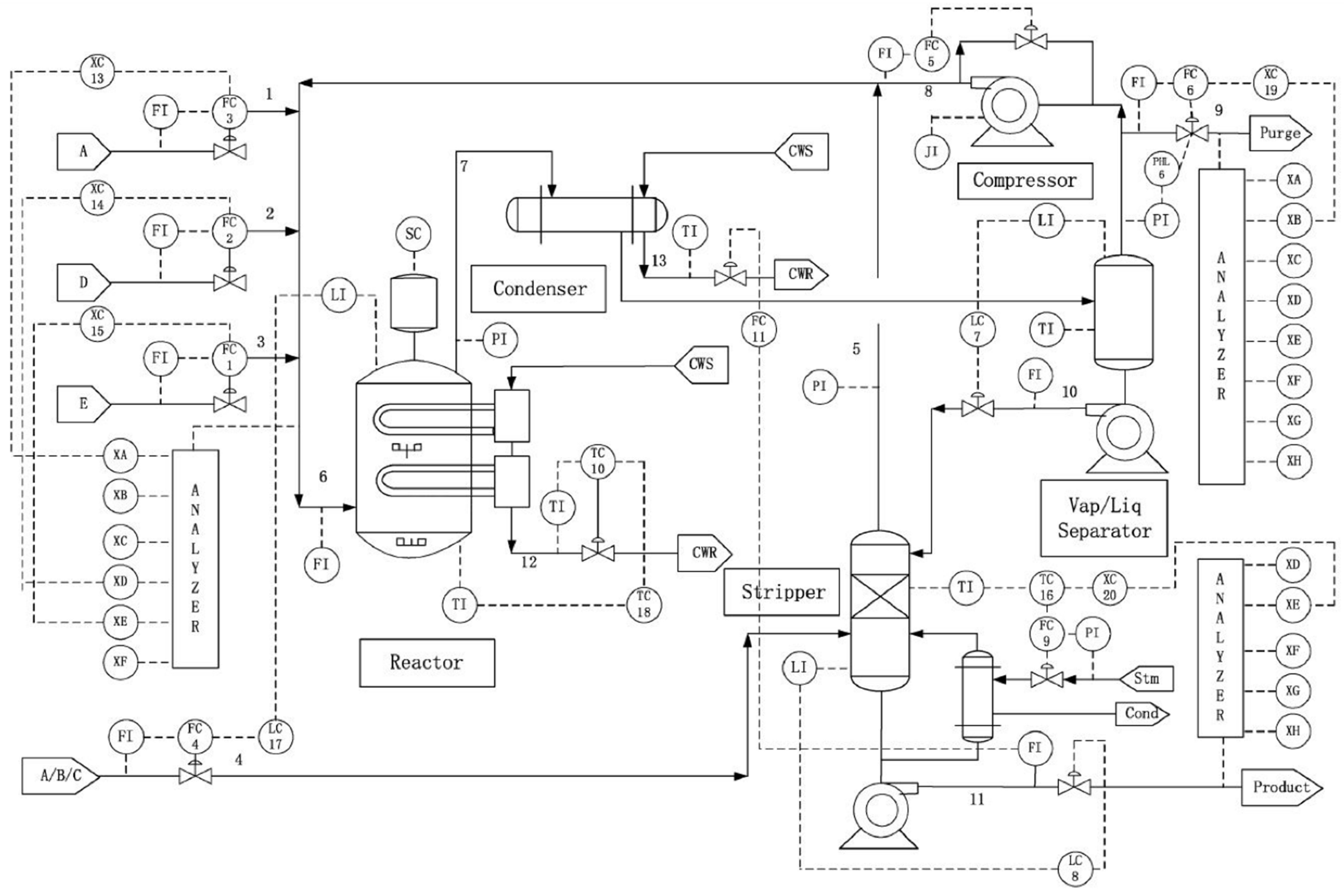

Figure 3.4: Flowsheet of TEP.

Table 3.1: Manipulated variables in TEP.

Description	Label
D feed flow	XMV(1)
E feed flow	XMV(2)
A feed flow	XMV(3)
A and C feed flow	XMV(4)
Compressor recycle valve	XMV(5)
Purge valve	XMV(6)
Separator pot liquid flow	XMV(7)
Stripper liquid product flow	XMV(8)
Stripper steam valve	XMV(9)
Reactor cooling water flow	XMV(10)
Condenser cooling water flow	XMV(11)

In this work, publicly available "additional data set" for TEP[1] is used, in which samples from 500 simulation runs with different random seeds are presented. 20 classes of faults are employed in this work, and their types are summarized in Table 3.3 (more details are accessible in the work of Downs and Vogel [118]). Each simulation run contains 500 faulty/normal training samples and 960 faulty/normal testing samples that are sampled every 3 minutes during the sampling period. For faulty cases, faults are introduced after 1 and 8 hours for training and testing runs, respectively.

In our experiments, 20 normal training simulation runs are selected as training data with in total 20×500 normal training samples, and 10 testing simulation runs are selected from both the normal and faulty test data sets, which consists of 10×960 normal samples and $10 \times 800 \times 20$ faulty samples.

The ABC and AAE in SAAC for the TEP experiments are symmetric AEs with the encoder architectures of $(50, 48, 40, 36, 30)$, and $(40, 30, 20, 15, 10, 5)$. The layers are acti-

[1]https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1

Table 3.2: Measured variables in TEP.

Description	Label	Description	Label
A feed	XMEAS(1)	Separator cooling water outlet temperature	XMEAS(22)
D feed	XMEAS(2)	Reactor feed analysis of A	XMEAS(23)
E feed	XMEAS(3)	Reactor feed analysis of B	XMEAS(24)
A and C feed	XMEAS(4)	Reactor feed analysis of C	XMEAS(25)
Recycle flow	XMEAS(5)	Reactor feed analysis of D	XMEAS(26)
Reactor feed rate	XMEAS(6)	Reactor feed analysis of E	XMEAS(27)
Reactor pressure	XMEAS(7)	Reactor feed analysis of F	XMEAS(28)
Reactor level	XMEAS(8)	Purge gas analysis of A	XMEAS(29)
Reactor temperature	XMEAS(9)	Purge gas analysis of B	XMEAS(30)
Purge rate	XMEAS(10)	Purge gas analysis of C	XMEAS(31)
Product separator temperature	XMEAS(11)	Purge gas analysis of D	XMEAS(32)
Product separator level	XMEAS(12)	Purge gas analysis of E	XMEAS(33)
Product separator pressure	XMEAS(13)	Purge gas analysis of F	XMEAS(34)
Product separator underflow	XMEAS(14)	Purge gas analysis of G	XMEAS(35)
Stripper level	XMEAS(15)	Purge gas analysis of H	XMEAS(36)
Stripper pressure	XMEAS(16)	Product analysis of D	XMEAS(37)
Stripper underflow	XMEAS(17)	Product analysis of E	XMEAS(38)
Stripper temperature	XMEAS(18)	Product analysis of F	XMEAS(39)
Stripper steam flow	XMEAS(19)	Product analysis of G	XMEAS(40)
Compressor work	XMEAS(20)	Product analysis of H	XMEAS(41)
Reactor cooling water outlet temperature	XMEAS(21)		

vated with SELU (except the last layer of the encoders and decoders), and batch size is set to 128. No dropout or batch normalization is applied.

To compare the performance of SAAC with other existing anomaly detection techniques, isolation forest (IF) [119], GANomaly [63], VAE [59], VAE-based deep SVDD (VAE-SVDD) [120], AnoGAN [60], FenceGAN [65] and AAE [97] are selected. The architecture of the GANomaly, VAE, VAE-SVDD and AAE are similar to the SAAC. For IF, the number of estimators is set to 120. Generator and discriminator of AnoGAN have the architecture of $(60, 36, 40, 48, 50, 52)$ and $(300, 150, 50, 1)$, respectively. For FenceGAN, feed-forward neural networks with $(80, 80, 52)$ and $(120, 100, 80, 50, 1)$ neurons are selected for the architecture of its generator and discriminator, respectively.

Table 3.3: Fault types in TEP.

Description	Disturbance	Type
A/C feed ratio, B composition constant	IDV(1)	Step
B composition, A/C ratio constant	IDV(2)	Step
D feed temperature	IDV(3)	Step
Reactor cooling water inlet temperature	IDV(4)	Step
Condenser cooling water inlet temperature	IDV(5)	Step
A feed loss	IDV(6)	Step
C header pressure loss-reduced availability	IDV(7)	Step
A, B, C feed composition	IDV(8)	Random variations
D feed temperature	IDV(9)	Random variations
C feed temperature	IDV(10)	Random variations
Reactor cooling water inlet temperature	IDV(11)	Random variations
Condenser cooling water inlet temperature	IDV(12)	Random variations
Reaction kinetics	IDV(13)	Slow drift
Reactor cooling water valve	IDV(14)	Sticking
Condenser cooling water valve	IDV(15)	Sticking
Unknown	IDV(16)	Unknown
Unknown	IDV(17)	Unknown
Unknown	IDV(18)	Unknown
Unknown	IDV(19)	Unknown
Unknown	IDV(20)	Unknown

Table 3.4 demonstrates the AUC-ROC results for different techniques on the TEP, where the normal and faulty classes are provided in the data set. As summarized from the table, SAAC shows the highest average AUC-ROC among them. For Fault 5, SAAC achieves AUC-ROC of 100.0%, while the highest AUC-ROC for the other techniques is 81.6% by GANomaly. For Faults 3, 9 and 15, all the techniques show comparable and relatively low AUC-ROC, which is expected since these faults are considered as hard-to-

Table 3.4: The AUC-ROC comparison results of various anomaly detection techniques for 20 faults in TEP.

Status Index	IF	GANomaly	VAE	VAE-SVDD	AnoGAN	FenceGAN	AAE	SAAC
Fault 1	99.6%	99.8%	99.9%	99.8%	99.9%	99.9%	99.9%	99.9%
Fault 2	99.4%	99.1%	99.4%	99.4%	99.4%	99.5%	99.5%	99.5%
Fault 3	51.8%	51.6%	50.4%	50.5%	50.5%	51.6%	50.2%	50.5%
Fault 4	63.9%	68.7%	100.0%	88.2%	94.7%	95.5%	99.9%	99.8%
Fault 5	68.8%	81.6%	75.4%	66.5%	59.5%	76.4%	70.1%	100.0%
Fault 6	99.9%	100.0%	100.0%	99.7%	100.0%	100.0%	100.0%	100.0%
Fault 7	81.1%	96.9%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Fault 8	98.6%	98.0%	98.7%	98.7%	98.6%	98.6%	98.8%	98.8%
Fault 9	52.3%	51.4%	50.6%	50.9%	50.9%	52.1%	50.5%	50.8%
Fault 10	80.2%	94.8%	79.9%	72.9%	69.4%	84.1%	77.1%	87.4%
Fault 11	74.6%	68.3%	92.6%	81.8%	86.1%	86.7%	92.3%	92.2%
Fault 12	99.5%	99.4%	99.6%	99.5%	99.4%	99.5%	99.5%	99.7%
Fault 13	97.8%	97.0%	97.6%	97.6%	97.6%	97.8%	97.6%	97.7%
Fault 14	89.5%	98.5%	100.0%	99.7%	100.0%	100.0%	100.0%	100.0%
Fault 15	53.7%	52.5%	51.1%	52.2%	51.7%	53.5%	51.2%	52.2%
Fault 16	69.7%	92.1%	76.2%	62.6%	59.3%	76.3%	71.5%	82.0%
Fault 17	87.9%	92.7%	97.2%	89.9%	94.2%	95.0%	96.5%	96.8%
Fault 18	96.9%	96.4%	97.1%	96.8%	97.0%	97.1%	97.0%	97.2%
Fault 19	67.6%	88.8%	73.6%	69.6%	64.0%	70.5%	80.3%	74.9%
Fault 20	76.5%	83.0%	84.4%	72.5%	75.4%	81.2%	81.7%	89.5%
Average	80.5%	85.5%	86.2%	82.4%	82.4%	85.8%	85.7%	**88.4%**

detect faults [121, 122]. As another performance metric, the average mBAs for Faults 1-20 of TEP are presented in Figure 3.5. It is observed that IF and SAAC have the lowest and highest average mBA, respectively.

The true positive rates (TPRs) of various techniques for Faults 1-20 are presented in Figure 3.6 and Table 3.5. These results are acquired in the case that their true negative rate (TNR) is chosen as 90.0%, as a practical scenario. TPRs of SAAC for Faults 5 and 20 are 40.3% and 19.6% higher than the best of the other techniques. GANomaly shows

Table 3.5: TPRs on TEP in the case that TNR is set to 90.0% for different techniques.

Fault Index	IF	GANomaly	VAE	VAE-SVDD	AnoGAN	FenceGAN	AAE	SAAC
Fault 1	99.6%	99.5%	99.9%	99.5%	99.2%	99.6%	99.8%	99.8%
Fault 2	98.8%	98.0%	98.9%	99.0%	97.7%	98.9%	99.1%	99.0%
Fault 3	10.9%	10.8%	10.2%	10.6%	1.0%	11.1%	10.1%	10.3%
Fault 4	18.1%	24.7%	100.0%	65.2%	26.8%	87.9%	100.0%	100.0%
Fault 5	38.0%	59.7%	40.0%	36.0%	18.7%	44.4%	36.5%	100.0%
Fault 6	99.7%	100.0%	100.0%	99.3%	99.5%	100.0%	100.0%	100.0%
Fault 7	54.8%	89.7%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Fault 8	97.1%	95.5%	97.4%	97.2%	94.9%	97.2%	97.5%	97.4%
Fault 9	11.9%	10.8%	10.5%	10.6%	1.1%	11.7%	10.3%	10.4%
Fault 10	51.7%	88.2%	52.1%	40.5%	9.0%	61.0%	47.1%	75.7%
Fault 11	31.6%	29.2%	82.4%	61.1%	37.2%	68.4%	81.8%	81.3%
Fault 12	98.8%	98.5%	99.1%	98.9%	95.8%	99.1%	99.0%	99.4%
Fault 13	95.6%	93.9%	95.5%	95.3%	92.9%	95.7%	95.6%	95.5%
Fault 14	63.5%	95.9%	99.9%	99.0%	99.8%	99.9%	100.0%	99.9%
Fault 15	13.0%	11.7%	10.9%	11.7%	1.2%	12.9%	10.8%	12.0%
Fault 16	30.3%	82.6%	43.2%	23.1%	1.9%	42.8%	34.5%	63.7%
Fault 17	63.4%	82.5%	93.9%	77.7%	72.5%	88.1%	92.3%	93.2%
Fault 18	93.6%	93.1%	94.6%	93.7%	92.6%	94.3%	94.4%	94.7%
Fault 19	24.5%	72.5%	34.2%	31.9%	2.8%	29.2%	49.4%	40.0%
Fault 20	43.6%	62.2%	64.8%	41.7%	22.4%	58.2%	59.7%	84.4%

Table 3.6: Average TPRs for 20 faults of TEP for different techniques when TNR is 90.0%.

Technique	IF	GANomaly	VAE	VAE-SVDD	AnoGAN	FenceGAN	AAE	SAAC
Average TPRs	56.9%	70.0%	71.4%	64.6%	53.3%	70.0%	70.9%	**77.8%**

superior results in Faults 10, 16, and 19, and SAAC shows slightly better or equivalent results to the best of other techniques for other faults. The average TPRs of these models are provided in Table 3.6 for all faults in the case that their TNR is set to 90.0%. It is observed that SAAC shows the average TPR of 77.8%, which is 6.4% higher than the

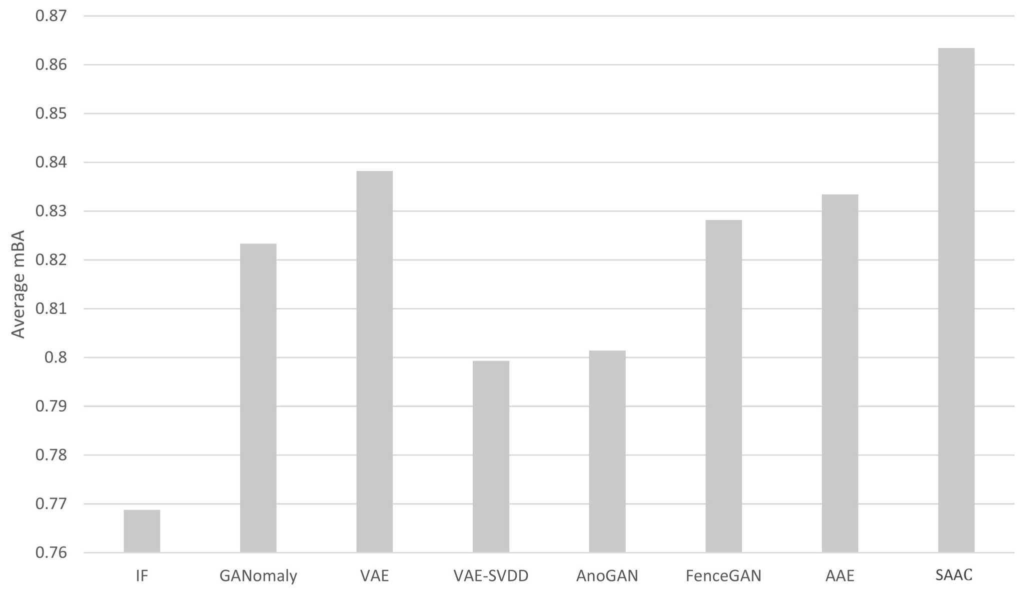

Figure 3.5: Average mBA for different techniques using all 20 faults in TEP.

second-best technique. In the light of these results, SAAC outperforms the aforementioned techniques in terms of average AUC-ROC, average mBA and average TPR for TEP.

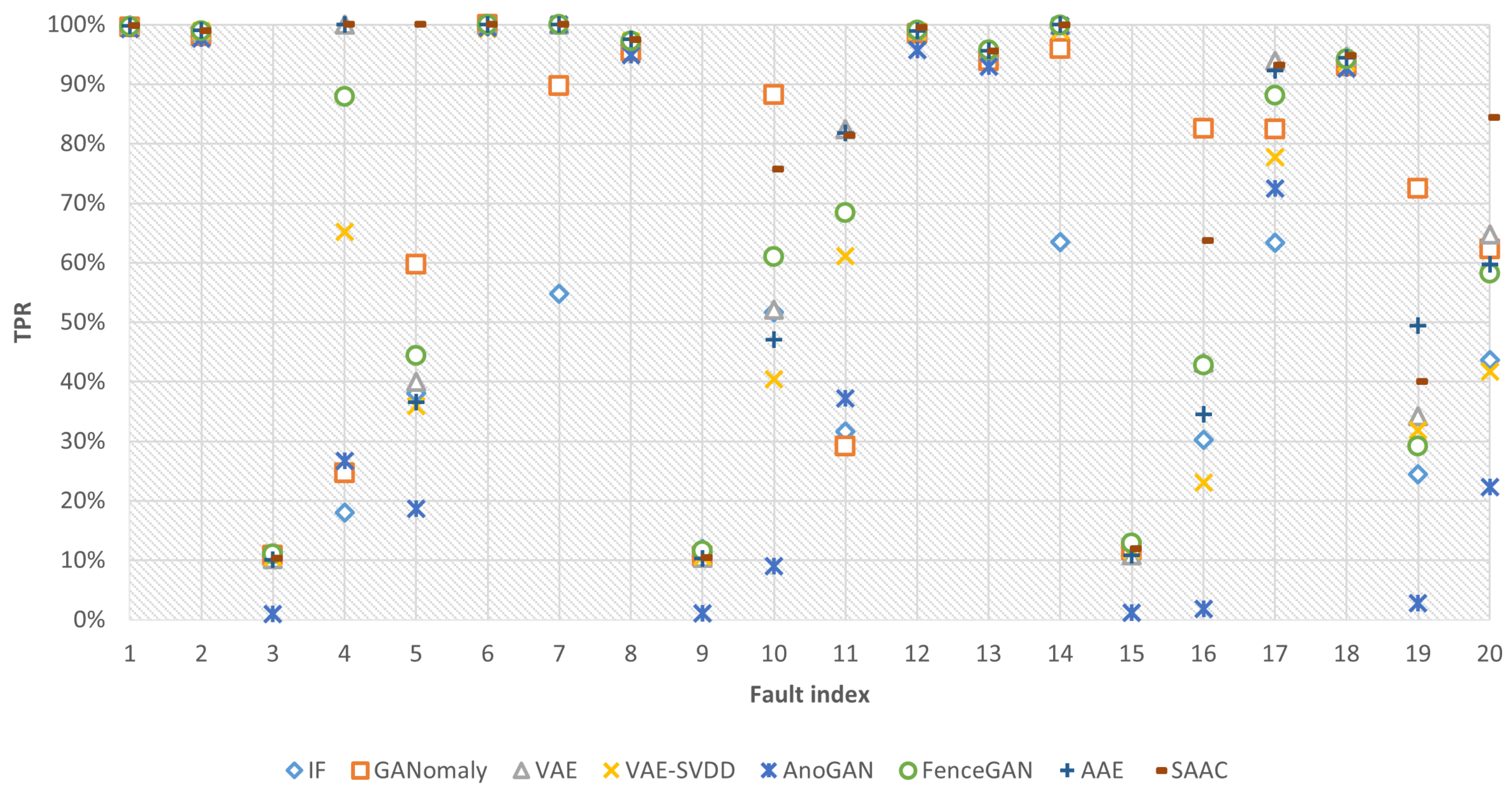

Figure 3.6: TPRs for different techniques on TEP in the case that TNR is set to 90.0%.

3.2.2 Credit Card Dataset

This data set contains 284,315 normal and 492 fraudulent transactions from credit card transactions, which is considered as a highly imbalanced data set. In our work, 80% of the normal samples are selected randomly for training while the rest are used as the testing set. For this experiment, the architecture of SAAC, AE, GANomaly [63], MOCCA [117], VAE [59], VAE-SVDD [120], and AAE [97] are similar to the ones described in Subsection 3.2.1.

The comparison results between SAAC and other techniques are presented in Table 3.7. In the light of these results, Soft MOCCA and IF show the lowest AUC-ROC and mBA, respectively, and SAAC is superior to other techniques in terms of average AUC-ROC and mBA.

Table 3.7: Average AUC-ROC and mBA (%) for different techniques averaged from 10 independent runs on Credit Card data set.

	AE	IF	GANomaly	VAE	VAE-SVDD	Soft MOCCA	AAE	SAAC
AUC-ROC	95.3	94.7	95.4	95.5	95.0	92.1	94.7	**96.2**
mBA	91.3	90.0	91.2	91.3	91.4	90.1	90.7	**92.1**

3.2.3 CIFAR10

The CIFAR10 data set contains 32×32 color images from 10 classes with 5000 and 1000 images per class for training and testing, respectively. The classes in CIFAR10 are mutually exclusive, and there is no overlap between them. An illustration of images in CIFAR10 is provided in Figure 3.7[2]. For the anomaly detection task, one class from the training set is regarded as normal, and all the classes are employed to evaluate the models' performance in the testing phase. This procedure is repeated for all the 10 classes in the data set.

[2]This image is available online at the following link: https://www.cs.toronto.edu/ kriz/cifar.html.

Similar architecture (without batch normalization layers) and pre-processing steps to the work of Massoli et al. [117] are implemented in this case study.

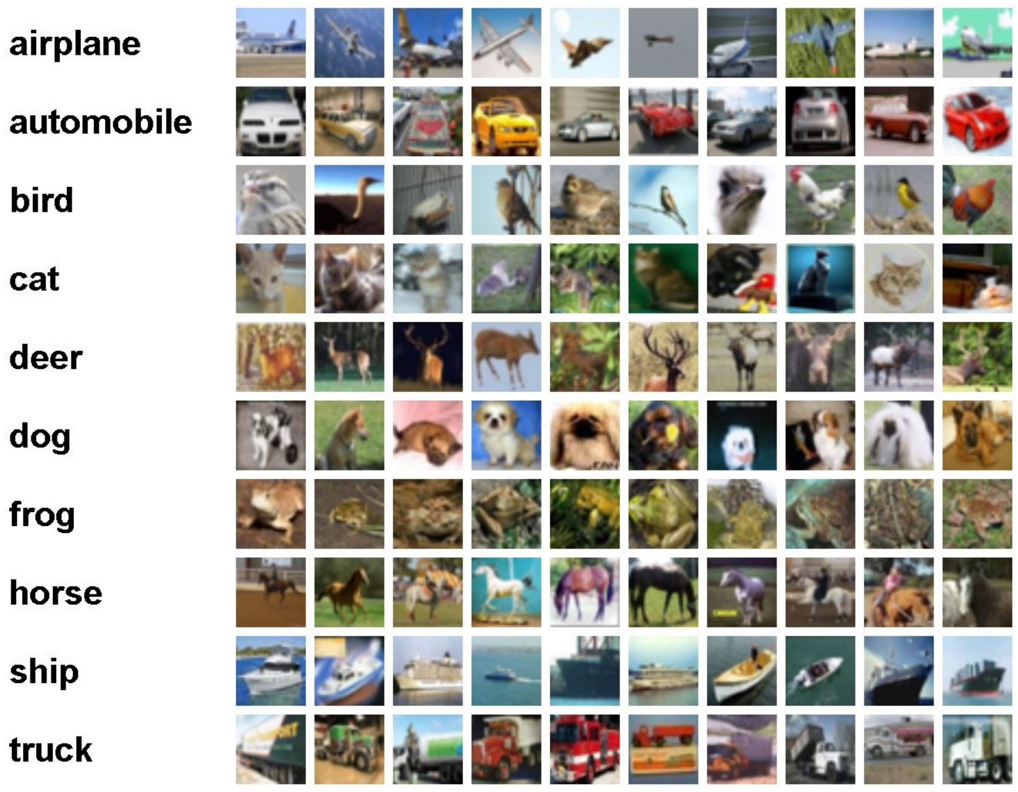

Figure 3.7: Sample images in CIFAR10 for different classes.

Table 3.8 shows the experimental results of SAAC and other techniques in terms of AUC-ROC. To capture the stochasticity caused by weight initialization and random number generation, averages and standard deviations from 10 independent runs are reported in this table for SAAC. For DCAE [123, 124], AnoGAN and OC Deep SVDD [50], values are reported from the work of [50]. For LSA [125], Soft and Hard MOCCA [117], OC-GAN [126], PIADE [127], DAGPR [128], and DOC^3 [129] values are taken from their original article. From this table, it can be noted that SAAC shows the highest average AUC-ROC compared with other existing techniques on CIFAR-10. Moreover, SAAC achieves the best performance for classes *Automobile*, *Horse*, and *Truck*. Since similar architecture as

MOCCA [117] is employed, it is concluded that the better performance achieved by SAAC is due to the improvements of the technique, and it is not because more complex architecture is utilized. It is also worth noting that SAAC demonstrates better performance compared to LSA, which employs larger models compared to SAAC.

Table 3.8: AUC-ROC and standard deviations (%) averaged from 10 independent runs on CIFAR10 data set based on one-vs-rest approach.

	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	Average
DCAE	59.1±5.1	57.4±2.9	48.9±2.4	58.4±1.2	54.0±1.3	62.2±1.8	51.2±5.2	58.6±2.9	76.8±1.4	67.3±3.0	59.4
AnoGAN	67.1±2.5	54.7±3.4	52.9±3.0	54.5±1.9	65.1±3.2	60.3±2.6	58.5±1.4	62.5±0.8	75.8±4.1	66.5±2.8	61.8
OC-Deep SVDD	61.7±4.1	65.9±2.1	50.8±0.8	59.1±1.4	60.9±1.1	**65.7±2.5**	67.7±2.6	67.3±0.9	75.9±1.2	73.1±1.2	64.8
LSA	73.5	58.0	69.0	54.2	76.1	54.6	75.1	53.5	71.7	54.8	64.1
Soft MOCCA	62.6±2.1	74.6±0.8	57.5±1.8	57.8±1.1	61.5±1.2	**66.3±1.0**	67.4±1.2	72.1±0.4	79.1±1.2	77.3±1.0	67.6
Hard MOCCA	66.0±1.5	70.5±1.3	52.4±1	60.1±0.6	60.9±1.2	**68.4±1.6**	67.1±0.5	68.5±1.0	79.2±0.8	75.8±0.7	66.9
OC-GAN	75.7	53.1	64.0	**62.0**	72.3	62.0	72.3	57.5	**82.0**	55.4	65.7
PIADE	75.1	55.0	**70.8**	60.9	**80.5**	64.5	72.9	65.1	77.1	53.2	67.5
DAGPR	75.1	73.7	59.5	56.4	69.2	57.2	69.2	53.1	76.7	79.3	66.9
DOC^3	**81.3±0.5**	74.2±1.3	66.0±0.6	**62.1±0.4**	74.0±1.6	**63.0±4.6**	**76.7±0.3**	67.6±1.8	81.1±0.6	76.8±2	72.3
SAAC	78.1±1.4	**78.6±0.6**	64.0±1.0	61.1±0.2	77.3±2.5	**67.1±1.7**	**75.3±4.1**	**76.6±1.4**	80.9±0.7	**80.6±1.1**	**74.0**

Chapter 4

SAAE for Supervised Fault Detection and Fault Diagnosis Methodology

4.1 The Proposed Methods

As we discussed earlier, labels are hard and costly to achieve. However, as the process proceed faulty samples may emerge. Thus, in addition to the unsupervised SAAC, we also proposed a supervised technique to employ faulty samples in FDD with the assumption of having a labelled data set.

4.1.1 Fault Detection

AE-based fault detection is an effective approach for detecting faulty samples that display distinguishable behaviors from normal samples. However, some faulty samples that exhibit similar patterns to normal ones can be misclassified by AEs. Furthermore, normal samples often dominate in the collected training data since faults occur infrequently in normal operating conditions. Thus, utilizing all samples for training could lead to the imbalanced data issue in a binary classification NN. Additionally, faulty samples may be costly to obtain, which will bring severe consequences to real-world industrial processes. In these cases, designing an effective method capable of producing reasonable results without faulty samples and reusable once faulty samples become accessible is crucial.

In this subsection, a new fault detection method, referred to as source-aware autoencoder (SAAE), is proposed. SAAE shares the following characteristics: i. tuning precision and recall trade-off, ii. training without faulty samples and fine-tuning once faulty samples emerge, iii. working effectively for the data sets with imbalanced number of normal and faulty data, iv. detecting unseen fault types, and v. exploiting all normal and faulty samples. Additionally, the proposed approach is effective for fine-tuning existing AEs utilized for fault detection tasks. In contrast to unsupervised AEs, SAAE is designed to extract fault-relevant information in a supervised manner to differentiate between normal and faulty samples. It is noted that the applications of SAAE are not limited to FDD and it can be used for any kinds of two-class classification tasks.

Assume that normalized normal and faulty data are denoted as $\mathcal{D}_N = \{\mathbf{x}_N^{(i)}\}_{i=1}^{n_1}$ and $\mathcal{D}_F = \{\mathbf{x}_F^{(i)}\}_{i=1}^{n_2}$, where $\mathbf{x}_N, \mathbf{x}_F \in \mathbb{R}^p$, p is the number of variables, and n_1 and n_2 are number of normal and faulty samples, respectively. The superscript (i) is the index of the sample. $\mathcal{D}_N$ and $\mathcal{D}_F$ are transformed into the following matrices with the aid of time window transformation.

$$\begin{aligned}
\mathbf{X}_N^{(i)} &= [\mathbf{x}_N^{(i-q+1)}, \mathbf{x}_N^{(i-q+2)}, \dots, \mathbf{x}_N^{(i)}]^\top, (i = q, q+1, \dots, n_1) \\
\mathbf{X}_F^{(i)} &= [\mathbf{x}_F^{(i-q+1)}, \mathbf{x}_F^{(i-q+2)}, \dots, \mathbf{x}_F^{(i)}]^\top, (i = q, q+1, \dots, n_2) \\
\mathcal{X}_N &= \{\mathbf{X}_N^{(i)}\}_{i=q}^{n_2} \\
\mathcal{X}_F &= \{\mathbf{X}_F^{(i)}\}_{i=q}^{n_2}
\end{aligned}$$

where q is the size of the sliding window. An SAAE with a pre-designed architecture is trained with normal data by minimizing the mean squared error (MSE) between training data and their reconstructed values with the following cost function:

$$C_N = \frac{1}{n_1 - q + 1} \sum_{i=q}^{n_1} \mathcal{L}_N \left(\mathbf{X}_N^{(i)}, \hat{\mathbf{X}}_N^{(i)} \right) \tag{4.1}$$

where

$$\mathcal{L}_N \left(\mathbf{X}_N^{(i)}, \hat{\mathbf{X}}_N^{(i)} \right) = \mathrm{MSE} \left(\mathbf{X}_N^{(i)}, \hat{\mathbf{X}}_N^{(i)} \right) \tag{4.2}$$

$\hat{\mathbf{X}}_N$ is the predicted value of $\mathbf{X}_N$ by the SAAE. After this training step, the obtained network is called *pre-trained network*.

Define mean(t_k) and std(t_k) as the mean and standard deviation of $\mathcal{L}_N$ for normal training data at iteration t_k, respectively, and denote MSE_F as the MSE for faulty samples. Now, the goal is to define a limit in such a way that $\mathcal{L}_N$ gets values lower than the limit and MSE_F returns values higher than the limit. This limit changes as the number of iterations increases, and it is connected to its previous states by adding momentum. The limit is symbolized by limit(t_k) at iteration t_k, and it is formulated as follows:

$$\mathrm{limit}(t_k) = m \times (\mathrm{mean}(t_k) + \beta \times \mathrm{std}(t_k)) + (1 - m) \times \mathrm{limit}(t_{k-1}) \quad \text{for} \quad t_k > 1 \tag{4.3}$$

$$\mathrm{limit}(t_1) = \mathrm{mean}(t_1) + \beta \times \mathrm{std}(t_1) \quad \text{for} \quad t_1 = 1 \tag{4.4}$$

where β and m are constants. The loss function for faulty samples can be formulated as follows:

$$\mathcal{L}_{F,t_k} \left(\mathbf{X}_F^{(i)}, \hat{\mathbf{X}}_F^{(i)} \right) = \max \left(0, \alpha \times \mathrm{limit}(t_k) - \mathrm{MSE}_F \left(\mathbf{X}_F^{(i)}, \hat{\mathbf{X}}_F^{(i)} \right) \right) \tag{4.5}$$

where α is a constant, and $\mathcal{L}_{F,t_k} \left(\mathbf{X}_F^{(i)}, \hat{\mathbf{X}}_F^{(i)} \right)$ is the loss function at iteration t_k. The max function in Eq. (4.5) penalizes faulty data that has lower reconstruction errors than $\alpha \times$ limit(t_k). When $\mathrm{MSE}_F \left(\mathbf{X}_F^{(i)}, \hat{\mathbf{X}}_F^{(i)} \right)$ increases, $\mathcal{L}_{F,t_k} \left(\mathbf{X}_F^{(i)}, \hat{\mathbf{X}}_F^{(i)} \right)$ decreases. It also indicates that Eq. (4.5) does not encourage the network to produce a very high reconstruction error for faulty data, and thus does not stimulate the network to get high weights and biases. In other words, if MSE_F reaches values lower than $\alpha \times$ limit, the network is encouraged to increase their MSEs.

After defining $\mathcal{L}_{F,t_k}$, the overall cost function is defined to minimize the MSEs for normal samples and to impose the penalty for the faulty samples simultaneously:

$$C(t_k) = \frac{1}{n_1 - q + 1} \sum_{i=q}^{n_1} \mathcal{L}_N \left(\mathbf{X}_N^{(i)}, \hat{\mathbf{X}}_N^{(i)} \right) + \frac{\lambda}{n_2 - q + 1} \sum_{i=q}^{n_2} \mathcal{L}_{F,t_k} \left(\mathbf{X}_F^{(i)}, \hat{\mathbf{X}}_F^{(i)} \right) \tag{4.6}$$

where $C(t_k)$ denotes the cost function at iteration t_k. λ is a non-negative real number that serves as the trade-off parameter between $\mathcal{L}_N$ and $\mathcal{L}_{F,t_k}$. When λ is large, the network pays more attention to reduce $\mathcal{L}_{F,t_k}$, which may lead to a high reconstruction error for normal data and the network may become ineffective. On the other hand, small λ encourages the network to focus more on the reconstruction of normal samples, leaving the faulty samples unexploited. Parameters of the *pre-trained network* are kept. Then, this network is trained to minimize Eq. (4.6).

Once training of the SAAE is completed, assume that the ratio of normal data that must be classified as normal is determined as p_r. Then l^* is the value that corresponds to the objective that $(1 - p_r) \times 100\%$ of members of $\mathcal{F}$ get values higher that l^*, where $\mathcal{F} = \{\text{MSE}(\mathbf{X}_N^{(i)}, \hat{\mathbf{X}}_N^{(i)})\}_{i=q}^{n_1}$. It is recommended that the value of l^* be determined from validation normal set.

The overall steps for training an SAAE and obtaining l^* are described in Algorithm 1. In this work, BiLSTM SAAE with skip connection is selected for fault detection network.

Remark 3. In this work, α, β and m are predefined values, and generally, λ is the only tuning hyper-parameter, whose value depends on the characteristics of input data and the value of p_r. β is set to 3, and the intuition behind it is as follows. If MSE for normal samples follows a normal distribution, 99.7% of them lie between (mean $-$ 3 $\times$ std) and (mean + 3 $\times$ std). The momentum coefficient m makes the limit(t_k) dependent on its previous values. Its range is $[0, 1]$, and it determines how much the value of limit(t_k) is related with its previous values. Greater the momentum coefficient m in Eq. (4.3) is, less dependency between limit(t_k) and limit(t_{k-1}) exists. The value for α is determined heuristically, which is calculated by

$$\alpha = \frac{r}{\text{mean}(t_0) + \beta \times \text{std}(t_0)} \tag{4.7}$$

Algorithm 1: Algorithm for Training SAAE

```
Input: Normalized normal and faulty sample matrices denoted by $\mathcal{D}_N$ and $\mathcal{D}_F$,
       SAAE with pre-designed architecture, $\lambda$, $m$, $\beta$ and $p_r$
Output: Trained SAAE, $l^*$
1  Transform $\mathcal{D}_N$ and $\mathcal{D}_F$ using time window transformation (if applicable), and
    construct normalized normal and faulty sample matrices $\mathcal{X}_N$ and $\mathcal{X}_F$;
2  Initialize parameters of the SAAE;
3  for $i = 1, 2, \dots, i_1$ do
4      Train SAAE through minimizing Eq. (4.1) for 1 iteration
5  Calculate mean($t_0$) and std($t_0$)
6      $\alpha \leftarrow \frac{r}{\text{mean}(t_0)+\beta\times\text{std}(t_0)}$, where $r$ is calculated from $p_r$
7  for $j = 1, 2, \dots, i_2$ do
8      Calculate
9          $\text{mean}(t_j) \leftarrow \text{mean}\Big(\mathcal{L}_N\left(\mathbf{X}_N^{(q)}, \hat{\mathbf{X}}_N^{(q)}\right), \cdots, \mathcal{L}_N\left(\mathbf{X}_N^{(n_1)}, \hat{\mathbf{X}}_N^{(n_1)}\right)\Big)$
10         $\text{std}(t_j) \leftarrow \text{std}\Big(\mathcal{L}_N\left(\mathbf{X}_N^{(q)}, \hat{\mathbf{X}}_N^{(q)}\right), \cdots, \mathcal{L}_N\left(\mathbf{X}_N^{(n_1)}, \hat{\mathbf{X}}_N^{(n_1)}\right)\Big)$
11     if $j = 1$ then
12         $\text{limit}(t_j) \leftarrow \text{mean}(t_j) + \beta \times \text{std}(t_j)$
13     else
14         Update limit($t_j$) with $m \times (\text{mean}(t_j) + \beta \times \text{std}(t_j)) + (1 - m) \times \text{limit}(t_{j-1})$
15     Calculate $C(t_j)$ with Eqs. (4.5) and (4.6)
16     Train SAAE through minimizing Eq. (4.6) for 1 iteration
17 Output the trained SAAE model
18 Calculate $\mathcal{F} = \{\text{MSE}(\mathbf{X}_N^{(i)}, \hat{\mathbf{X}}_N^{(i)})\}_{i=q}^{n_1}$
19 Determine $l^*$ in such a way that $p_r$ percent of members of $\mathcal{F}$ are lower or equal to
    $l^*$.
```

where r is the corresponding l^* for p_r in the *pre-trained network*.

Remark 4. Even though that the training step of SAAE is explained by employing full data sets, it can be used together with mini-batches of training data.

4.1.2 Fault Diagnosis

The objective of the fault diagnosis network is to determine the classes of faulty samples that are detected in the fault detection network, and in this work, fault diagnosis is regarded as a multi-class classification task. Assume that the number of fault classes is denoted by n_c. Denote FDR_i as the fault detection rate (FDR) obtained from the fault detection network for fault class i ($i = 1, \cdots, n_c$), and FDR of the normal class is denoted by FDR_0. Then, a deep NN is designed for the classification task. The last layer of the NN model contains $n_c + 1$ neurons, and the softmax function defined in Eq. (2.20) is adopted as its activation function. This network is trained to minimize the weighted categorical cross-entropy loss in Eq. (2.21) where the weight vector $\mathbf{w}$ is defined as

$$\mathbf{w} = \begin{bmatrix} \theta \times (1 - \mathrm{FDR}_0) \\ \theta \times (1 + \mathrm{FDR}_1) \\ \vdots \\ \theta \times (1 + \mathrm{FDR}_{N_c}) \end{bmatrix} \tag{4.8}$$

where $\theta \geq 1$ is a constant dependant on the data and fault diagnosis network. It is noted from Eq. (4.8) that the outputs of the fault detection network will affect the training of the fault diagnosis network. If a fault is detected with a higher FDR, a higher weight will be attached to this class in the diagnosis network. Thus the fault detection results are incorporated into the fault diagnosis network. It is also worth noting that the fault diagnosis network is trained on both normal and faulty training data. Nevertheless, the weight for the normal class is lower compared to fault classes, which implies that the main focus of the fault diagnosis network is to classify fault classes correctly.

In this work, the architecture of the fault diagnosis network is designed as a combination of BiLSTM and ResNet, and it aims to prevent the randomness caused by the random proximity of input features in CNNs. BiLSTM has two recurrent components, a forward recurrent component and a backward one. The forward component computes the hidden and cell states similar to a standard unidirectional LSTM, whereas the backward component computes them by taking the input sequence in reverse-chronological order (i.e. starting from time step T to 1). The intuition of using a backward component is that the network is set up in a way where it sees future data and learns its weights accordingly, which helps the network to capture some dependencies which otherwise wouldn't have been captured by the standard (forward) LSTM. In addition to the importance of modeling bidirectional time dependence, the order of features also plays an important role, which cannot be guaranteed in CNNs (except 1D-CNNs) due to the local operations. However, the outputs of BiLSTM are not based on local operations, and all features are employed. Therefore, BiLSTM is employed in our work to avoid the randomness caused by the feature order.

It has been observed that NNs with a high number of parameters can demonstrate superior results. For instance, 110-layer Res-Net with 1.7 million parameters outperforms 20-layers Res-Net with 0.27 million parameters on CIFAR10 in the work of He et al. [99]. A similar pattern has been reported in the work of Huang et al. [104]. Motivated by these works, a deep NN is designed for the fault diagnosis task, and its details are described in Subsection 4.2.2.

4.1.3 Fault Detection and Diagnosis

In this subsection, the fault detection and fault diagnosis approaches in Subsections 4.1.1 and 4.1.2 are integrated. The FDD framework contains two phases, offline and online phases, which are depicted in Figure 4.1. Its main steps are summarized as follows:

- Offline phase:

    1. Separate the collected samples into normal and faulty sets, and normalize them;
    2. Transform normal and faulty sets with a time window of size q, which produces sample matrices with the dimension of $q \times p$;

3. Construct training, validation and testing sets from sample matrices;
4. Design SAAE architecture and select its parameters m and β with the methods described in Subsection 4.1.1; Choose the value of p_r and λ ;
5. Train the SAAE fault detection network with Algorithm 1 using faulty and normal training samples;
6. Determine l^* as the value that p_r percent of validation (or training) normal samples can be classified correctly;
7. Calculate weights for weighted categorical cross-entropy loss $\mathbf{w}$ after selecting θ from Eq. 4.8 and the trained SAAE;
8. After designing fault diagnosis network, train the fault diagnosis network by minimizing weighted categorical cross-entropy loss with normal and faulty training samples.

- Online phase:

1. Collect online data (or samples from the testing set), and normalize them with the same procedure as in the offline step;
2. Convert the online data to sample matrices with the size of $q \times p$ with the time window q;
3. If the sample class is known, update the offline model by going to Step 5 in the offline phase; otherwise continue to Step 4 in the online phase;
4. Compute the MSE for online data with the trained SAAE-based fault detection model;
5. Classify the sample as a faulty one if its MSE is higher than l^*; otherwise label it as a normal sample;
6. Determine the class of the detected sample with the fault diagnosis network.

In this work, a BiLSTM-based SAAE with skip connections is selected for fault detection network, and the combination of BiLSTM and ResNet is chosen as the fault diagnosis

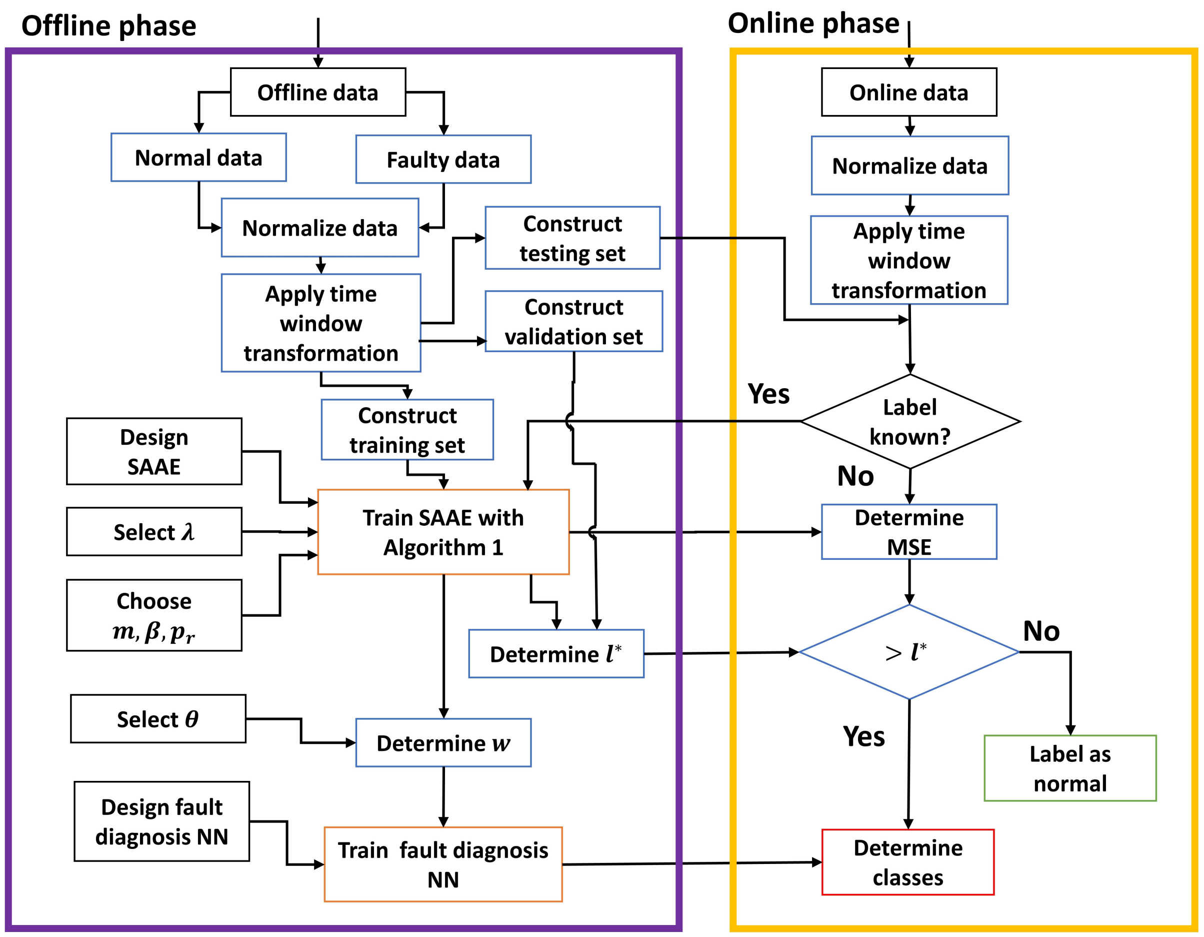

Figure 4.1: Flowchart of fault detection and diagnosis in online and offline phases.

network, and the integration of these two networks through the steps described in this subsection is named as SAAE-ResNet.

Remark 5. In the online phase, a sample that is detected as fault in the fault detection network may still be classified as a normal sample by the fault diagnosis network.

4.2 Experimental Results and Discussions

TEP has been regarded as a benchmark for comparing various fault detection and fault diagnosis approaches. Relying only on a data set from one simulation run for training and testing could lead to biased results, since random noises with different order of magnitudes will affect industrial processes, and the results achieved from various simulation runs could vary considerably [130]. To overcome these problems, training and testing data sets are enriched with data from independent simulation runs. 100 and 30 simulation runs from the training data set are utilized as training and validation data, and 40 simulation runs from the testing data set are chosen as testing data. Further, samples under normal operating conditions are discarded for faulty data sets. A time window of 40 is also introduced to reshape the data, leading to 46100, 13830 and 36840 normal samples and 44100, 13230 and 30440 faulty samples (for each fault) for training, validation and testing data, respectively. It is noted that for each fault, 10% of faulty samples are selected randomly from each simulation run to reduce computation cost. This is a common situation in many real-world industrial data since faulty data are not always available from the beginning of their introduction. Therefore, for each fault class 4400 training samples are employed while the testing set contains 30440 faulty samples.

Then the following transformation is applied for the j^{th} variable of i^{th} sample, $x_j^{(i)}$, to ensure that the value of the normal samples is in the range of $[-1, 1]$.

$$\bar{x}_j^{(i)} = 2 \times \frac{x_j^{(i)} - \min_{i=1}^{n_1} x_{N,j}^{(i)}}{\max_{i=1}^{n_1} x_{N,j}^{(i)} - \min_{i=1}^{n_1} x_{N,j}^{(i)}} - 1 \tag{4.9}$$

In this study, the experiments are performed in Python 3.8 on a desktop computer with Intel Core i7-6700HQ (2.60GHz, 12GB RAM) under 64Bit Windows 10 operating system. NumPy [131], scikit-learn [132], and PyTorch [133] are used for implementation of the experiments. Throughout this work, Adam [110] is chosen as the optimizer.

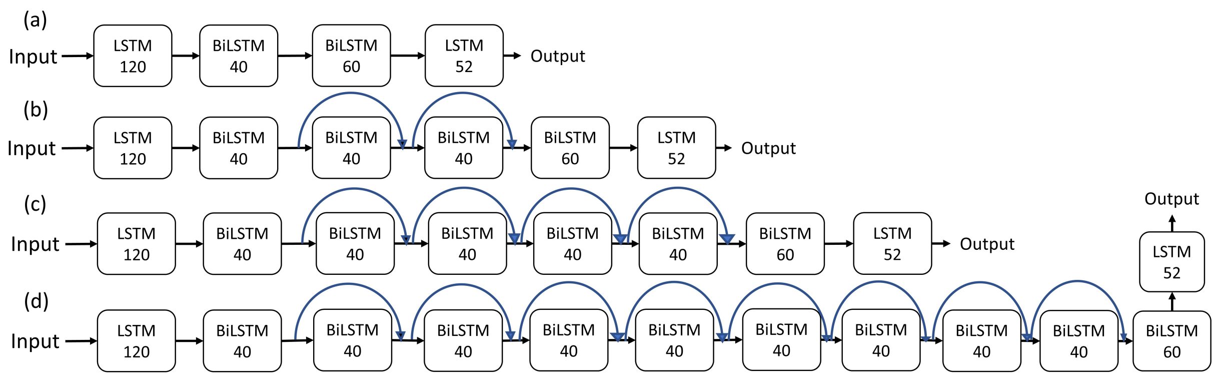

Figure 4.2: Architecture of SAAEs for the fault detection networks with (a) 4, (b) 6, (c) 8, (d) 12 layers.

4.2.1 Fault Detection Results

In this section, the proposed SAAE is tested on TEP for fault detection, and its structure is chosen based on BiLSTM layers with skip connections. To assess its performance, 4 SAAEs with 4, 6, 8 and 12 layers are designed and depicted in Figure 4.2. For instance, in the SAAE with 8 layers, normalized input matrices with the dimension of 40×52 are fed into an LSTM layer with 120 hidden size. Then, its output serves as the input of a BiLSTM layer with 40 hidden neurons. Then, the output of this layer, which has the size of 40×80 is added to the output of the third layer and makes the input of the fourth layer. This pattern repeats for layers 4, 5 and 6. Next, the output of the last BiLSTM layer with 40 hidden neurons (after applying skip connection) is fed to a BiLSTM layer and results in output matrices with the size of 40×120. Then, an LSTM layer produces output matrices with the identical size as the model's input matrices. Based on the described normalization technique in Eq. (4.9), the output values of the normalized normal samples still lie in $[-1, 1]$. However, normalized faulty samples can fall out of $[-1, 1]$, which is compatible with the objective of SAAEs.

The parameters of SAAEs are set as $m = 0.7$, $\beta = 3$, and the range of λ is $[4, 20]$. p_r, as a user-defined parameter, provides the flexibility for SAAEs to account for the precision and recall trade-off, and its value is highly interwoven with process specifications. Batch size for normal and faulty samples are set to 128 and 244 respectively. In Algorithm 1, SAAE is trained for 5 epochs ($i_1 = 5$) with learning rate of 0.001. In the second training loop of this algorithm, training epochs is increased to 60 ($i_2 = 60$), and the learning rate is reduced to 0.0001.

Figure 4.3 shows FDRs of the pre-designed 4 SAAEs with $p_r = 0.995$. As shown from the figure, FDRs for Faults 1, 2, 4-8, 11, 12, 14 and 19 are 100.0% in all 4 SAAEs. All the models show relatively low FDR for Fault 15, which may be related to the high user-defined value for p_r. It is observed that increasing the depth of SAAEs does not noticeably change FDRs for Faults 10, 13, 16, 17 and 20. But for Faults 3, 9 and 18, FDRs are improved by 1.7%, 10.1% and 0.6% from 4 to 8 layers SAAEs. Moreover, FDRs for Faults 3, 9 and 18 are decreased from 8 to 12 layers SAAEs, which indicates that increasing the number of layers in SAAEs does not necessarily enhance FDRs. Faults 3, 9 and 15 are identified to be difficult to detect [121, 134, 122]. However, it is observed that Faults 3 and 9 are detectable with SAAE even in the case that FDR for normal samples is set to 0.995. The average FDRs for all fault classes and normal validation set are 93.3%, 93.7%, 93.9% and 93.7% for SAAEs with 4, 6, 8 and 12 layers respectively, and SAAE with 8 layers is chosen as the fault detection network for subsequent analysis since it shows superior results in terms of the average FDR.

The comparison between AE and SAAE with 8 layers is illustrated in Figure 4.4. The AE with 8 layers showed relatively low FDRs for Faults 3, 9 and 15, which reveals that AEs are prone to misclassifying faults that behave similarly to normal samples. On the other hand, FDRs for Faults 3 and 9 are improved considerably in the SAAE with 8 layers. Average FDRs for all faults are improved from 84.2% in the AE to 93.6% in the SAAE. Therefore, it is experimentally observed that the SAAE performs better than the AE in terms of fault detection. Hence, the auxilary faulty training data set can contribute to the anomaly detection task.

Table 4.1: FDRs of SAAE, AE and binary classification for unseen fault groups on testing set.

	FDR for normal class	Average FDR of unseen faults		
		SAAE	AE	Binary classification
Unseen Faults 1-5	89.8%	99.4%	82.3%	82.2%
Unseen Faults 6-10	92.5%	96.0%	81.7%	86.5%
Unseen Faults 11-15	98.6%	80.3%	80.0%	80.1%
Unseen Faults 16-20	86.7%	99.4%	99.3%	86.0%
Average		**93.8%**	**85.8%**	**83.7%**

Figures 4.5 and 4.6 show the monitoring results using SAAE and AE for one simulation run on the testing data. From the results, SAAE shows superior monitoring performance compared to AE, and it is more sensitive to faults than AE, especially for Faults 3 and 9. For Fault 15, the MSEs of SAAE are closer to its limit compared to AE. Since the FDRs are affected by the control limit, whose value is determined by p_r, it is hypothesized that low FDR for Fault 15 in the SAAE is related to the high value of p_r.

Another important feature of a fault detection technique is its ability to detect unseen (new) fault classes. To evaluate the performance of SAAE, AE and binary classification NN, 5 fault classes are held out and the networks are trained on the remaining fault classes. The architecture of the binary classification NN is similar to the encoder part of the SAAE with 8 layers with one additional fully connected (FC) layer with 2 neurons and softmax activation function. Average FDRs for unseen fault classes for four groups of unseen faults (Faults 1-5, Faults 6-10, Faults 11-15 and Faults 16-20) are presented in Table 4.1. For each group, FDRs for the normal class are the same in all three networks. It can be observed that the SAAE possesses a higher average FDR of unseen faults in all fault groups, and the AE has a higher FDR for unseen faults compared to the binary classification NN on average. Therefore, it is concluded that SAAE demonstrates better fault detection performance for unseen faults compared to both AE and binary classification NN.

Table 4.2: Performance of the SAAE and binary classification NN for fault detection with imbalanced number of normal and faulty samples.

$\frac{\text{\# of faulty samples}}{\text{\# of normal samples}}$	FDR for normal class	Average FDR for faults	
		SAAE	Binary classification
19.1%	98.9%	93.4%	92.6%
9.5%	99.5%	92.5%	83.5%
4.8%	99.9%	90.8%	82.9%
2.4%	100.0%	90.4%	76.2%
Average		**91.8%**	**83.8%**

Table 4.2 summarizes the results of the SAAE and binary classification NN for data sets with imbalanced number of training normal and faulty samples matrices. 4 experiments with different values for the ratio of normal to faulty samples are conducted. It is observed that decreasing this ratio reduces the average FDR in the binary classification network, but it decreases less noticeably in the SAAE. Further, SAAE shows 8% higher average FDR in comparison to the binary classification network. Thus, SAAE is less sensitive to the ratio of the number of faulty to normal data samples compared to the binary classification network.

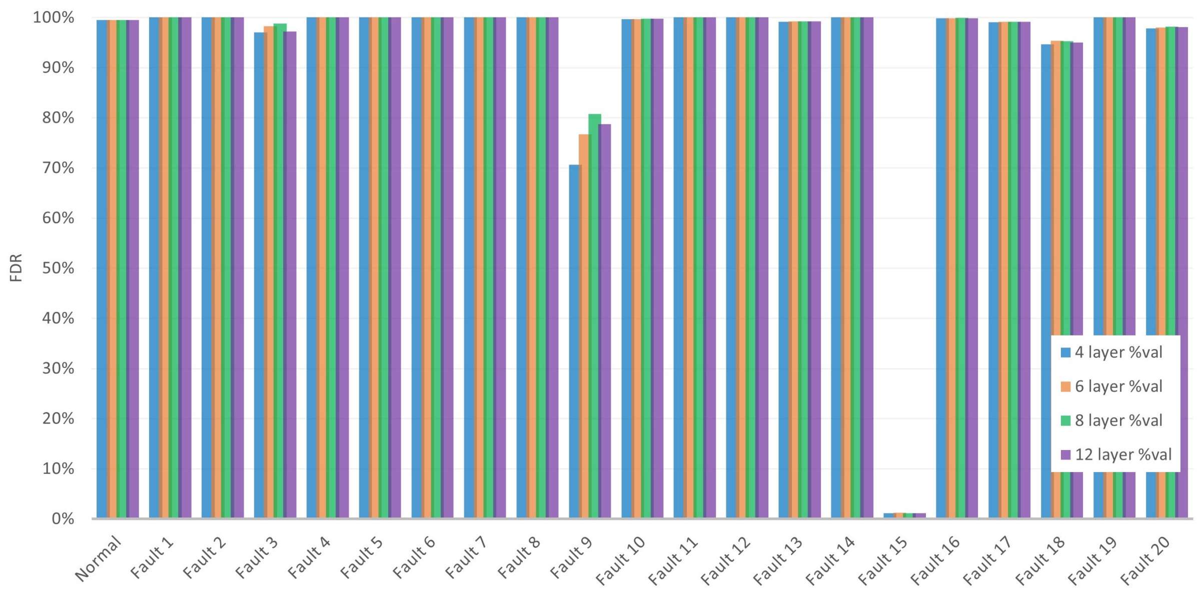

Figure 4.3: Fault detection rates for normal and faulty validation samples using 4, 6, 8 and 12 layer SAAEs ($p_r = 0.995$).

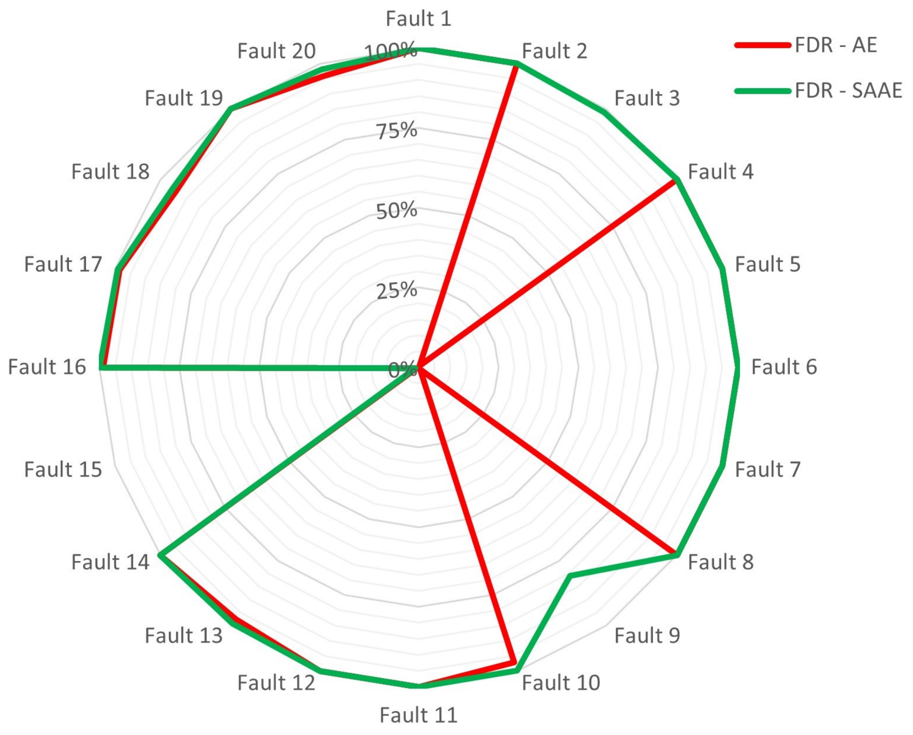

Figure 4.4: Radar plot of FDRs for fault detection AE and SAAE with 8 layers ($p_r = 0.995$).

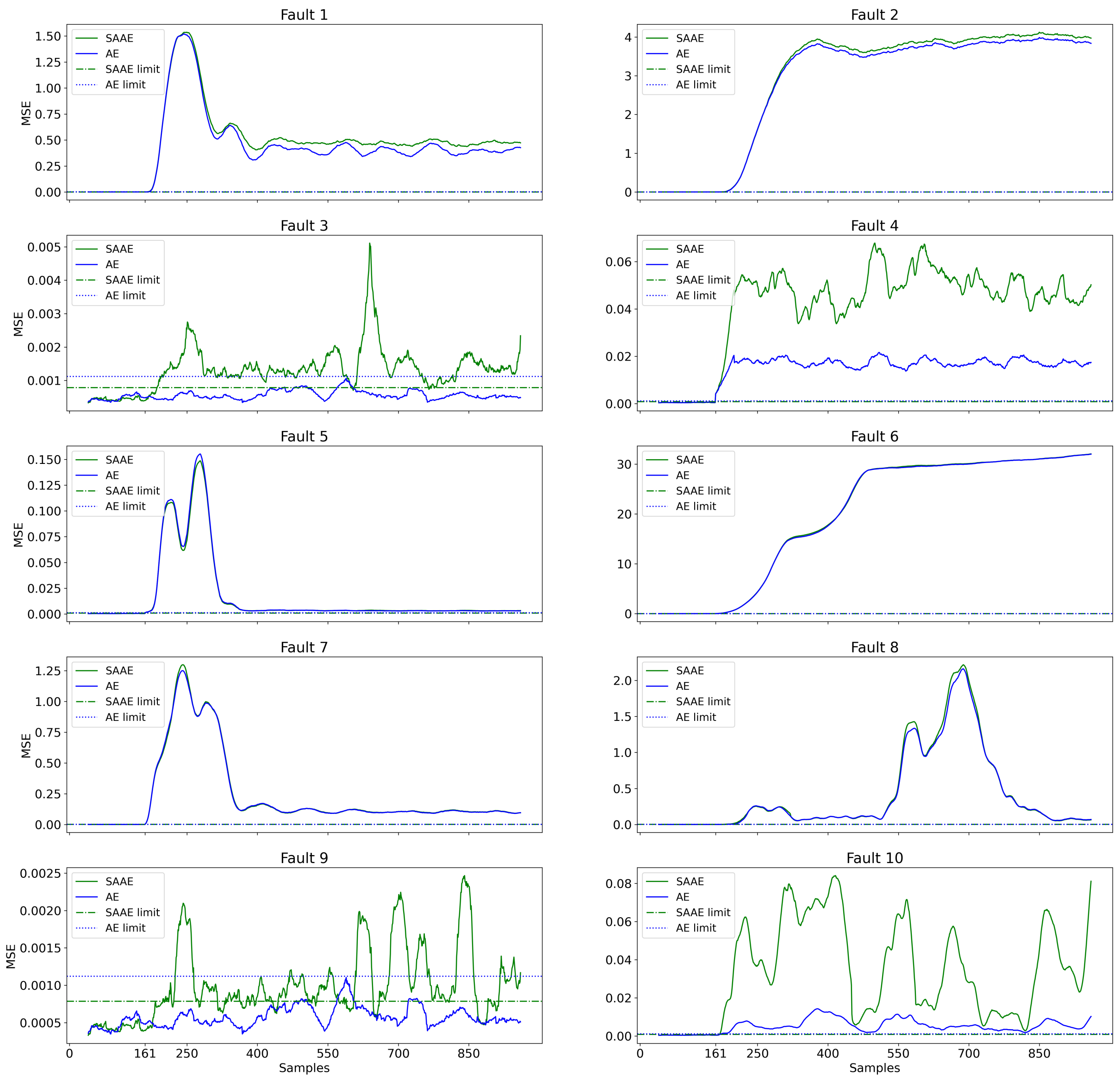

Figure 4.5: Process monitoring using SAAE and AE ($p_r = 0.995$) for Faults 1-10.

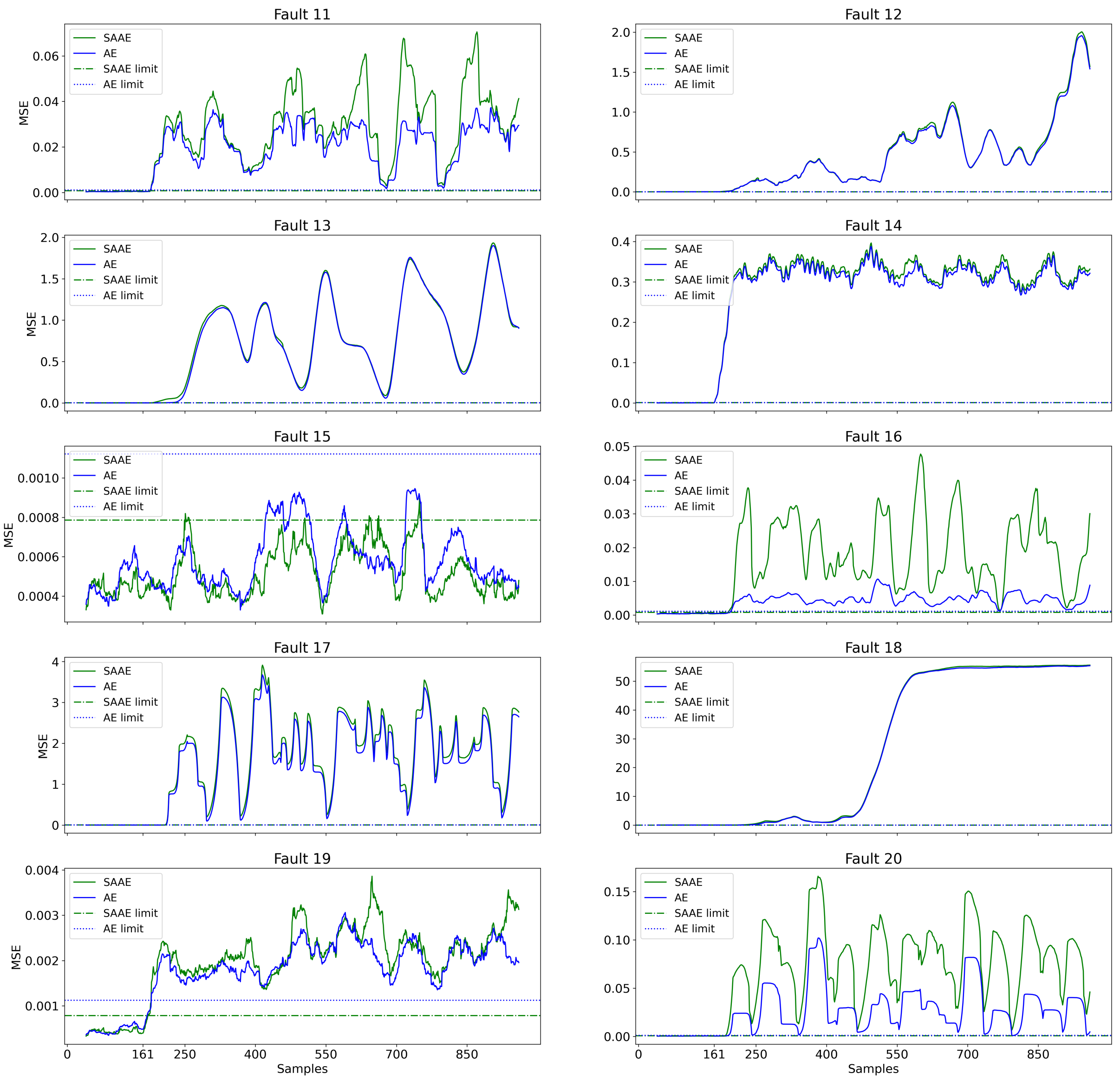

Figure 4.6: Process monitoring using SAAE and AE ($p_r = 0.995$) for Faults 11-20.

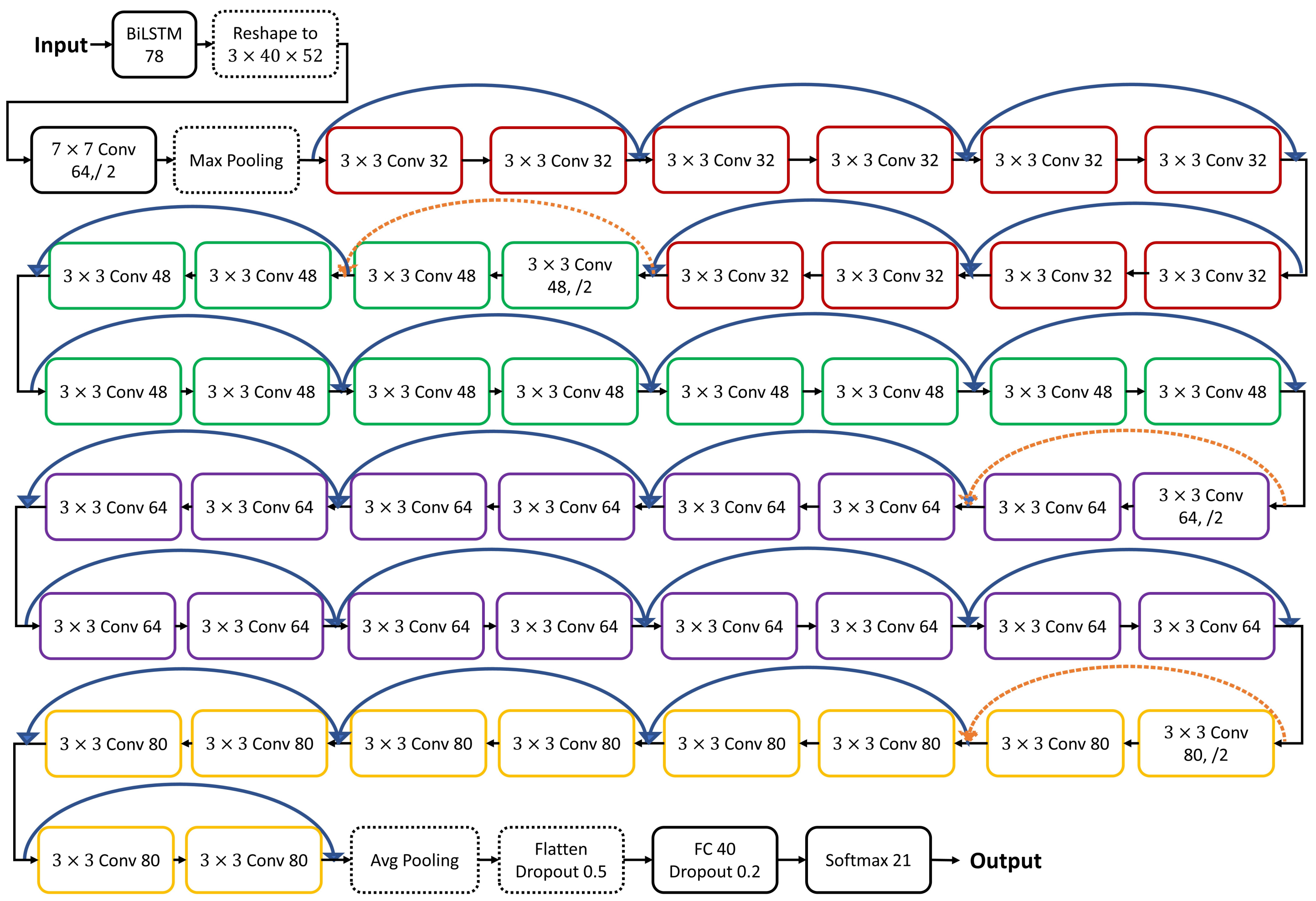

Figure 4.7: Architecture of BiLSTM-ResNet for fault diagnosis.

4.2.2 Fault Diagnosis Results

In this work, a combination of BiLSTM and ResNet is used as the fault diagnosis network. The BiLSTM layer aims to provide inputs to the ResNet in such a way that proximity of the features could reveal relevant information and avoid randomness caused by the order of features. Then, ResNet is used for the classification of samples. This architecture is referred to as BiLSTM-ResNet, which is shown in Figure 4.7. It is noted that the BiLSTM-ResNet is a very deep network with 52 layers, trained with both normal and faulty samples. Each *conv* block consists of a convolution, batch normalization and activation function. Residual connections are applied before the activation function of the corresponding *conv* layers. BiLSTM layer provides 3D inputs to *conv* blocks. The output of the last *conv* block is flattened and fed into a FC layer. The flatten operation and FC layer are equipped with Dropout. The output layer, which is activated with softmax function, determines the output of the BiLSTM-ResNet. Rectified linear unit (ReLU) is selected as the activation function in the *conv* blocks and FC layer. Dotted lines in Figure 4.7 indicate that 1×1 convolution with the stride of 2 is added to the input for applying residual connection operation.

For training the BiLSTM-ResNet, number of epochs, learning rate, batch size, m_{BN} and ϵ are set to 15, 0.001, 128, 0.1 and 10^{-5}, respectively. In this subsection, categorical cross-entropy loss is selected to make the results of the fault diagnosis network independent from the fault detection network. Results of BiLSTM-ResNet are presented in Figure 4.8. FDRs for all the faults (except Fault 15) are high and close in both training and testing sets. Average FDRs on training and testing sets are 96.5% and 94.3%, which reflects the good generalization ability of BiLSTM-ResNet. Moreover, testing FDRs for Faults 3, 9 and 15 are 95.8%, 81.2% and 34.9%, respectively, which implies that BiLSTM-ResNet is able to diagnose Faults 3 and 9 to some extend.

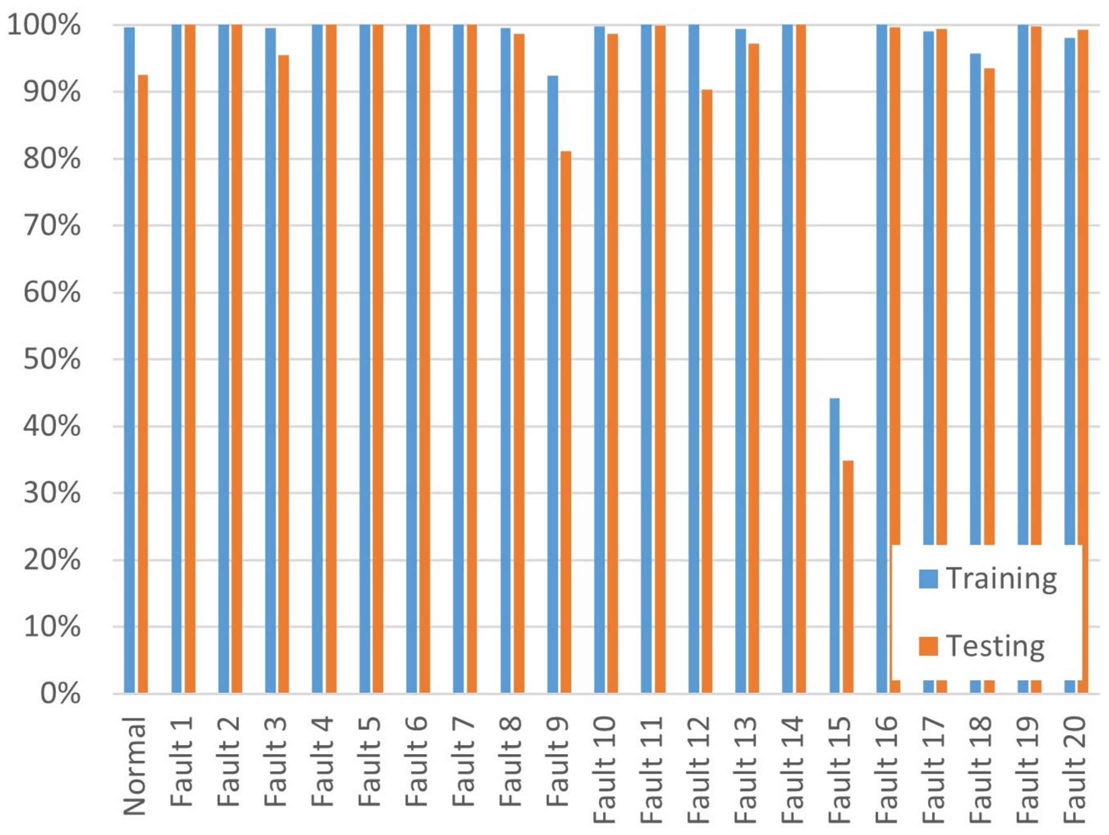

Figure 4.8: Fault diagnosis results of BiLSTM-ResNet on training and testing sets.

4.2.3 Fault Detection and Diagnosis Results

In the FDD framework, 8 layer SAAE and BiLSTM-ResNet are selected for fault detection and fault diagnosis respectively, which is referred to as SAAE-ResNet. Results are acquired for $p_r = 0.995$, 0.93 and 0.905. Parameters of the 8 layer SAAE for $p_r = 0.995$ are discussed in Subsection 4.2.1. For $p_r = 0.93$ and 0.905, λ and i_2 are set to 110 and 40, respectively. Once SAAEs are trained, 3 BiLSTM-ResNets are trained to minimize the corresponding weighed categorical cross-entropy loss (Eqs. (2.21) and (4.8)) for $p_r = 0.995$, 0.93 and 0.905. The results of the SAAE-ResNet are discussed in the next subsection together with other existing fault diagnosis methods.

4.2.4 Comparison Study

Several articles have reported the comparison results between deep learning and traditional machine learning methods. Heo and Lee [135] pointed out that ANN outperforms modified PLS by 10.5% lower false alarm rate. Wang et al. [136] compared stacked supervised auto-encoder (SSAE) with SVM, and the results show the superiority of SSAE by 4.2% on hydrocracking process and 39.6% on TEP in terms of classification accuracy. The analysis reported by Zhang et al. [137] showed that stacked LSTM obtains higher FDR over PCA, dynamic PCA, ICA, and dynamic ICA. According to these results, we mainly focus on the discussion and comparison of deep learning-based techniques in this thesis. To further illustrate the advantages of the proposed method over some traditional machine learning methods, FDRs of the proposed SAAE, linear discriminant analysis (LDA) and kernel SVM (K-SVM) for fault detection are reported in Table 4.3. It is noted that LDA and K-SVM utilize both normal and faulty data for process modeling. As shown in the table, the proposed SAAE achieves higher FDRs than LDA and K-SVM.

In the remaining of this subsection, a comprehensive comparison between SAAE-ResNet and other deep learning-based approaches is conducted. Instead of reporting the numbers in the corresponding papers, all the selected FDD models are re-simulated on the same data set for the following reasons.

- Several articles excluded some fault classes or just considered the fault diagnosis problem.
- The way that existing articles account for the dynamics of the TEP data (e.g., length of the time window) may be different.
- The total number of samples used for training and testing is not consistent in the literature.
- Some models are tested on limited number of data which may lead to biased results.

Table 4.3: FDRs of LDA, K-SVM, and SAAE for fault detection.

Status index	LDA	K-SVM	SAAE
Normal	40.4%	98.9%	99.5%
Fault 1	100.0%	100.0%	100.0%
Fault 2	100.0%	100.0%	100.0%
Fault 3	63.0%	2.1%	98.1%
Fault 4	64.9%	100.0%	100.0%
Fault 5	79.7%	100.0%	100.0%
Fault 6	95.4%	100.0%	100.0%
Fault 7	100.0%	100.0%	100.0%
Fault 8	91.3%	99.6%	100.0%
Fault 9	60.2%	1.6%	80.2%
Fault 10	63.8%	52.7%	99.9%
Fault 11	59.6%	81.6%	100.0%
Fault 12	92.0%	100.0%	100.0%
Fault 13	97.0%	97.6%	99.5%
Fault 14	60.8%	100.0%	100.0%
Fault 15	60.3%	1.8%	1.9%
Fault 16	60.1%	27.4%	100.0%
Fault 17	99.6%	99.1%	99.6%
Fault 18	91.9%	96.9%	97.9%
Fault 19	59.7%	5.4%	100.0%
Fault 20	99.7%	97.9%	99.3%
Average FDR for faults	80.0%	73.2%	93.8%

- Fault detection time, as an important aspect of a FDD model, is not reported frequently in the literature.

In order to make a fair comparison between various approaches, both FDR and F_1 are taken into account, and the testing set is enriched with samples from different simulation runs. For comparing the proposed approach with existing approaches, their FDR on the normal data is tuned to comparable values, and multiple comparisons with different FDRs on the normal class are conducted if possible. Suggesting specific procedures for conducting a fair comparison is inevitable, which are demonstrated in the following.

In a work by Wu and Zhao [34], 12 different architectures of CNNs were tested. For making the comparison, the architecture that showed the highest testing average FDR is selected. This architecture encompasses 3 convolutional layers with a 3×3 kernel size, 1 max pooling layer with a 2×2 kernel size, and 2 fully-connected layers. The output of these convolutional layers is flattened, and the FC layer with 300 neurons and a dropout rate of 0.5 produces the input of the next layer. In the next layer, an FC layer with the softmax activation function produces the output of the network. The number of neurons in this layer is set to 21. 2 networks with n_f equals to 128 and 150 are trained, where n_f is an integer denoting the number of filters in the convolutional layers. Each network is trained for 50 epochs. Batch size and learning rate are set to 128 and 0.001, respectively.

The comparison between unidirectional and bidirectional RNN showed the superiority of the later one by Zhang et al. [36]. Table 4.4 describes two architectures based on bidirectional gated recurrent unit (BiGRU), called BiGRU 1 and BiGRU 2, which are used for the comparison in this subsection. It is notable that linear transformation is selected to connect outputs from both time directions in BiGRU 1 and BiGRU 2. Dropout rate, number of epochs, batch size and learning rate are set to 0.5, 50, 100 and 0.0004, respectively.

The work of Heo and Lee [135] is selected for ANN comparison, and two architectures of (102-50-40-21) and (400-200-200-21) are selected for ANNs. The number of epochs, batch size and learning rate are set to 70, 100 and 0.001, respectively.

Wang et al. [136] proposed SSAE to extract relevant features for classification task, and declared that it outperforms stacked AE (SAE). Three stacked layers with 660, 330 and 165 neurons that are activated by tanh together with a softmax activated output layer with

Table 4.4: Structures of NNs for comparing with BiGRU.

Layer	Type	Dropout rate	Architectures	
			BiGRU 1	BiGRU 2
1	BiGRU	0.5	40×40	40×48
2	BiGRU	0.5	40×30	40×36
3	BiGRU	0.5	40×21	40×21
	Averaging		21	21
	Softmax		21	21

21 neurons are selected to test the effectiveness of SSAE and SAE. For training both SSAE and SAE networks, learning rate and batch size are set to 0.0001 and 64, respectively. For the pre-training step, the number of epochs is set to 30, and it is increased to 70 for the fine-tuning step.

Wu and Zhao [138] proposed process topology convolutional network (PTCN) to take advantage of the process flowchart information. Three stacked graph convolutional layers with the hidden dimension of 40 followed by an FC layer with 300 neurons equipped with the dropout rate of 0.5 and an output layer with softmax activation function and 21 neurons is selected to evaluate the performance of PTCN. Learning rate, batch size, and the number of epochs are set to 0.001, 128 and 50, respectively. Unobserved variables are initialized to zero and the adjacency matrix for TEP is transposed.

Multiblock temporal convolution network (MBTCN) was proposed by He et al. [139] for fault diagnosis. The comparison is made by using the official implementation, which is provided by the authors. The architecture and parameters of MBTCN are identical to the original MBTCN except for the batch size, number of epochs and learning rate. For the first 33 epochs, batch size equals 24, and it is increased to 64 for the next 7 epochs. The learning rate is reduced to 0.0001 from 0.001 in the last 2 epochs.

The results of the aforementioned techniques are summarized in Table 4.5. As shown in

Table 4.5: Comparison results of fault detection and diagnosis between different methods based on the testing set for TEP.

Status index	**Proposed SAAE-ResNet** p_r**=0.905**	**Proposed SAAE-ResNet** p_r**=0.93**	**Proposed SAAE-ResNet** p_r**=0.995**	**BiRNN**	**MBTCN**	**ANN**	**PTCN**	**SAE**	**SSAE**	**CNN**
	FDR	FDR	FDR	FDR	FDR	FDR	FDR	FDR	FDR	FDR
Normal	94.7%	96.0%	99.2%	93.1%	97.6%	91.9%	97.8%	93.5%	95.2%	91.5%
Fault 1	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Fault 2	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Fault 3	97.6%	97.0%	97.6%	96.8%	85.4%	90.4%	83.5%	87.8%	94.6%	50.1%
Fault 4	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Fault 5	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	99.8%	100.0%	99.9%
Fault 6	100.0%	100.0%	100.0%	98.5%	100.0%	100.0%	100.0%	99.2%	100.0%	100.0%
Fault 7	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Fault 8	99.4%	99.4%	99.4%	98.6%	99.4%	99.0%	99.6%	97.9%	97.9%	98.6%
Fault 9	78.7%	76.3%	72.7%	74.4%	57.7%	30.8%	17.7%	40.3%	21.7%	2.5%
Fault 10	98.6%	98.6%	98.6%	97.7%	98.8%	95.2%	94.3%	96.2%	94.0%	94.6%
Fault 11	99.9%	99.9%	99.9%	99.4%	99.9%	99.6%	99.7%	96.8%	98.5%	99.6%
Fault 12	93.4%	93.4%	93.4%	85.0%	96.0%	92.4%	93.7%	86.2%	85.5%	94.4%
Fault 13	96.7%	96.7%	96.7%	95.8%	96.2%	97.0%	96.8%	96.2%	95.6%	96.3%
Fault 14	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	99.9%	99.9%	100.0%
Fault 15	25.7%	21.2%	1.3%	28.1%	0.2%	9.9%	4.3%	8.7%	5.0%	8.0%
Fault 16	99.5%	99.5%	99.5%	99.1%	99.7%	95.4%	95.6%	94.5%	92.7%	88.5%
Fault 17	99.5%	99.5%	99.5%	99.4%	99.5%	99.3%	99.3%	99.3%	99.3%	99.3%
Fault 18	97.4%	97.4%	97.0%	97.8%	98.1%	97.9%	98.1%	97.9%	97.5%	97.7%
Fault 19	99.6%	99.6%	99.6%	97.3%	100.0%	81.3%	98.4%	74.7%	67.8%	99.8%
Fault 20	99.3%	99.3%	99.3%	99.3%	99.2%	99.1%	98.9%	99.0%	99.1%	98.6%
Average FDR for faults	**94.3%**	**93.9%**	**92.7%**	**93.4%**	**91.5%**	**89.4%**	**89.0%**	**88.7%**	**87.5%**	**86.4%**
Average FDR	**94.3%**	**94.0%**	**93.0%**	**93.3%**	**91.8%**	**89.5%**	**89.4%**	**89.0%**	**87.8%**	**86.6%**
Average F1	**94.1%**	**93.7%**	**92.0%**	**93.3%**	**90.8%**	**89.1%**	**88.5%**	**88.5%**	**87.2%**	**85.8%**

the table, the CNN shows the lowest FDR on normal class, average FDR and F_1, and it has a lower FDR for Faults 3, 9, 15 and 16 compared to other techniques. The inferior FDD performance of CNN may be related to the randomness brought by the order of features. To address this defect, MBTCN uses 1D convolutions which eliminates the influence of the proximity of features in CNNs. In SAAE-ResNet, a BiLSTM layer is placed before the CNN layers to assist the network to provide proximity-relevant inputs for CNN layers.

The pre-training step in the training of SAEs tries to extract features in an unsupervised manner, and SSAE is an extension of SAE in which the labels of inputs are used in the pre-training step. FDR for Fault 3 is higher in the SSAE, but it is lower in Fault 9 compared to the SAE. The average of FDRs for faults and F_1s are higher in the SAE but FDR for the normal class is higher in the SAE. This indicates that the results do not support the superiority of SAE or SSAE over each other for FDD in our experiment.

The PTCN takes into account the adjacency information from the process flowchart. It outperforms both SAE and SSAE since the FDR for normal class, average FDRs for faults and F_1s are higher in the PTCN. However, FDR for Fault 9 is much higher in the SAE compared to the PTCN.

The ANN obtains a lower FDR for the normal class and higher average FDRs and F_1 compared to the PTCN, SAE and SSAE. Thus, the performance of the ANN does not necessarily surpass the performance of the PTCN, SAE and SSAE. It is notable to point out that the SAE and SSAE, which are based on ANNs, and the ANN present lower FDRs for Fault 19 compared to other techniques.

The MBTCN takes use of the process flowchart to make its blocks and 1D convolutions to avoid the randomness caused by the order of features. The MBTCN shows a higher FDR for the normal class and average FDR and F_1 compared to the ANN, SAE and SSAE, which shows that the MBTCN outperforms them. Also, FDR for Fault 9 is much higher in the MBTCN compared to the PTCN.

In a classification task, BiRNNs make the decision on the classes of the inputs by extracting information from both time directions, and it achieves a higher FDR for incipient Fault 15.

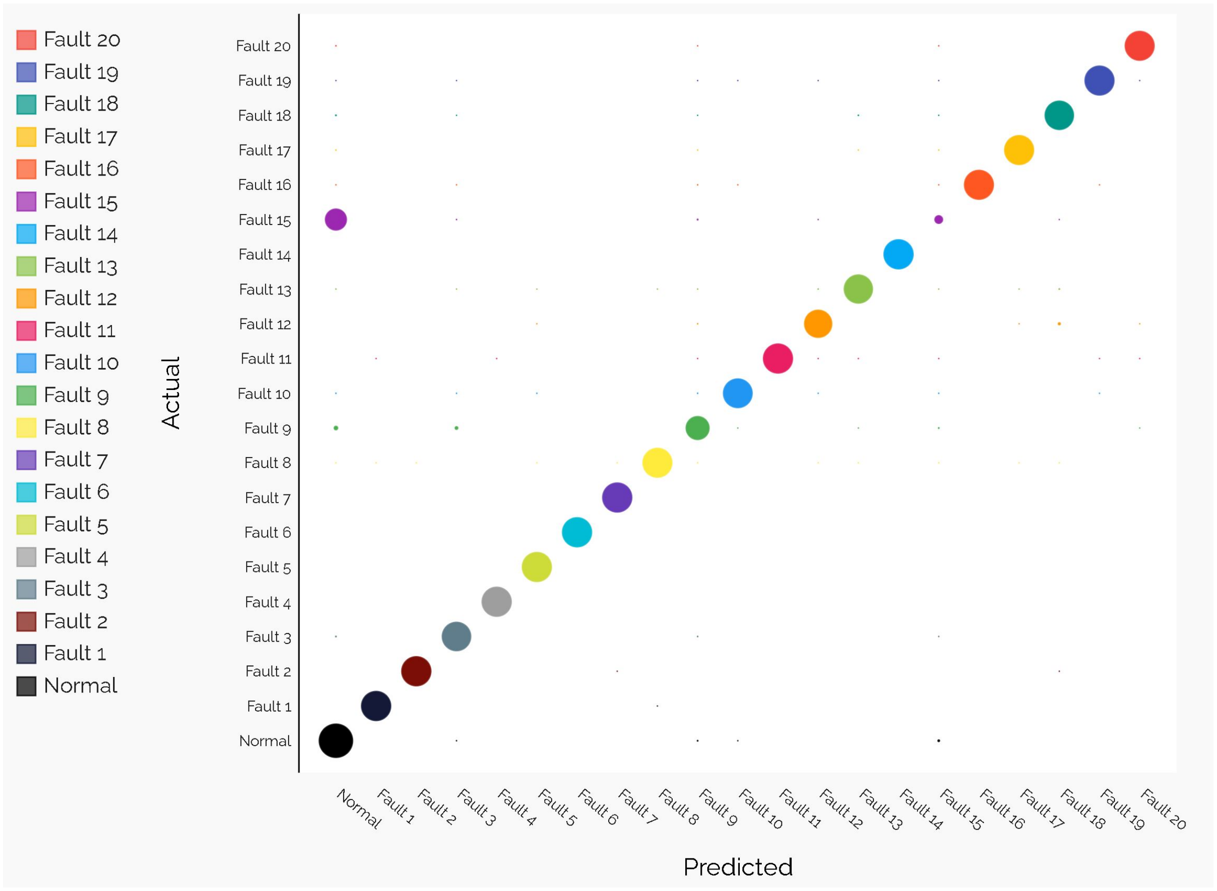

Figure 4.9: The details of FDD results on testing set for SAAE-ResNet with $p_r = 0.905$.

The performance of the proposed method is evaluated in three levels of FDR for the normal class. It is observed that the proposed method with $p_r = 0.995$ outperforms the MBTCN, ANN, PTCN, SAE, SSAE and CNN. For $p_r = 0.93$ or $p_r = 0.905$, FDR on normal class and average FDR and F_1 are higher for the proposed method compared to the BiRNN. Hence, SAAE-ResNet outperforms the other existing techniques in terms of average FDR. Moreover, the average FDR for incipient Faults 3, 9 and 15 is the highest in SAAE-ResNet with $p_r = 0.905$. The results of this network is further elaborated in Figure

4.9. From this figure, it is observed that samples with actual class of Fault 15 are mainly predicted as normal by the SAAE.

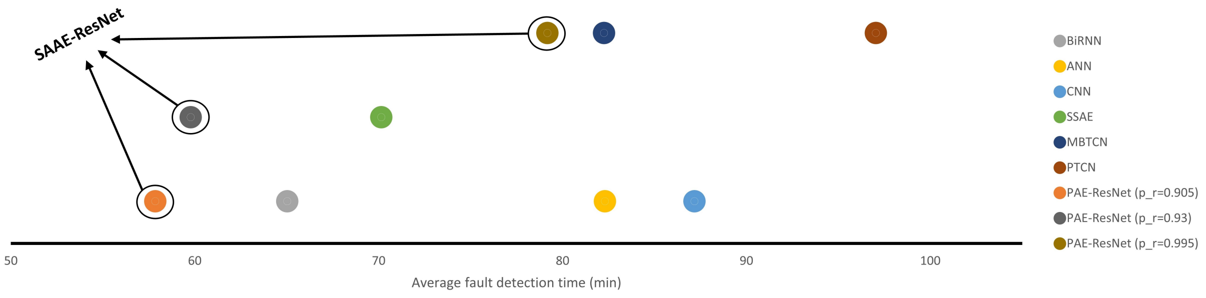

Figure 4.10: Average fault detection time for different techniques on testing set for TEP.

Another important metric of an FDD technique is how fast it can detect faults, since faster response is highly demanded in actual industrial processes. In this work, the criterion for calculating fault detection time is that the model detect 3 consecutive samples as fault. Average fault detection time is calculated as the average time required for a model to meet this criterion from the introduction of faults for all fault types. Figure 4.10 shows the average fault detection time for different FDD techniques. For $p_r = 0.995$, the FDR for the normal class is higher and the average fault detection time is lower in SAAE-ResNet, compared to the MBTCN and PTCN, indicating its superiority over the MBTCN and PTCN in terms of average fault detection time. Similarly, SAAE-ResNet outperforms the SSAE for $p_r = 0.93$ and the BiRNN, SAE, ANN, CNN for $p_r = 0.905$. Hence, SAAE-ResNet has a lower average fault detection time compared to other techniques.

Figure 4.11 shows the performance of various FDD techniques on training and testing data sets. The difference in average FDR on training and testing sets is considered as a metric of generalization ability. It is noted that to avoid overfitting issue, MBTCN and PTCN use knowledge from the process flowchart, and BiRNN and CNN employ dropout technique. From Figure 4.11, the difference between average FDR on training and testing sets in MBTCN and SAAE-ResNet with $p_r = 0.995$ are 1.2% and 0.7%, which indicates the superior generalization ability of SAAE-ResNet over MBTCN.

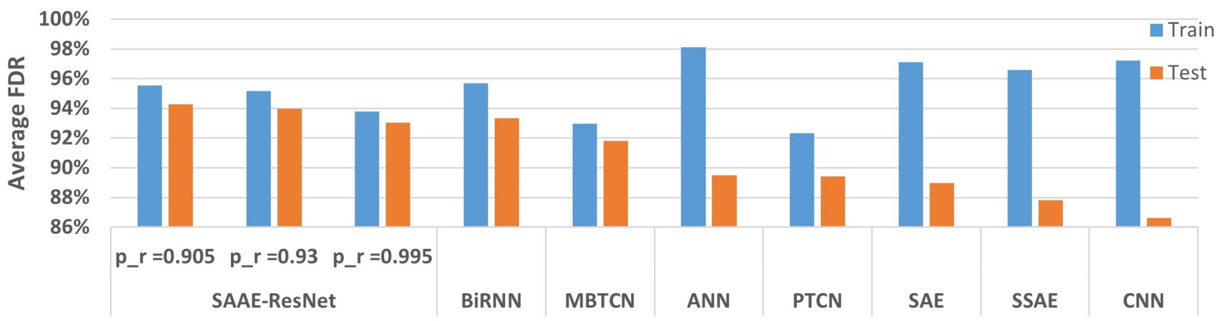

Figure 4.11: Average FDR from different techniques on training and testing sets.

Based on the discussions, SAAE-ResNet shows superior results in terms of the average FDR and F_1, generalization ability, average fault detection time, and average FDR for incipient faults. This technique also provides a framework for handling precision and recall trade-offs.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

In this work, an unsupervised anomaly detection technique, referred to as self-adversarial autoencoding classifier (SAAC), is proposed, and it is trained on normal samples in an end-to-end manner. The ability of adversarial autoencoders (AAE) for irregularity generation before convergence is coupled with the competence of the autoencoding binary classifier (ABC) to detect unseen anomalies, which enable utilizing ABC in an unsupervised manner. In this thesis, ABC is improved by introducing a new hyper-parameter, and the AAE is employed as an irregularity generator. Moreover, a framework to adversarially train these two networks is also proposed in this work. The superior anomaly detection performance of SAAC is demonstrated through the comparison with other strong techniques on three benchmark data sets.

To fully exploit the labelling information once it becomes available, a two-step and supervised fault detection and diagnosis method is proposed. For fault detection, autoencoders are extended to source-aware autoencoder, which has the following advantages:

providing the flexibility to adjust precision and recall trade-off, being able to work with and without faulty samples, being convenient for imbalanced data sets in terms of number of faulty and normal samples, and being re-trainable once a new fault type emerges or more faulty samples become accessible. Bidirectional long short-term memory with skip connections is designed for fault detection. Integration of BiLSTM and ResNet is designed for the fault diagnosis task, and the BiLSTM layer is designed to address the issue of randomness caused by the order of features. A FDD framework is also discussed, where the fault diagnosis network is stimulated to be trained based on the results of the fault detection network. Thus, the assumption of employing fault detection and fault diagnosis techniques separately is removed in the SAAE-ResNet.

Tennessee-Eastman process is selected to investigate the efficacy of the proposed FDD methods. SAAE shows better performance in terms of the average FDR compared to AE, and it is more competent than the AE and binary classification networks for detecting unseen fault classes, and is less sensitive to imbalance data sets. The results of the comprehensive comparison between different techniques on TEP not only show that SAAE-ResNet is capable of addressing the problem of tuning precision and recall trade-offs, but also demonstrate its superiority with lower fault detection time, higher average FDR and F_1, and generalization ability.

5.2 Future works

The proposed SAAE shows superior fault detection results with imbalanced data sets. However, imbalanced data sets could exist in the subsequent fault diagnosis step as well. One approach to account for the imbalanced number of fault-fault samples is to employ GANs for data generation. In this way, the class that has fewer samples can be enriched. Several promising research directions include i. exploitation of data from different modes of a process in the design of GANs, and ii. incorporation of process knowledge into GANs. Furthermore, evaluating the quality of generated samples from GANs is still an open research topic.

It is common to encounter a chemical process working in different operating conditions (modes) to achieve certain output specifications. To develop FDD techniques for each mode, having access to an enriched data set is a must. To fully exploit the data acquired from the process, it is important to take advantage of the data gathered from other modes of the process in the FDD technique of a specific mode. In these scenarios, transfer learning is a solution to transfer knowledge from a source task (in this case other modes of the process) to a target task (in this case the selected mode of the process). In this research direction, proposing combinations of GANs with the concept of transfer learning to generate samples from one mode of the process with employing the data from other modes is demanded.

Furthermore, self-supervised techniques have received increasing attention during recent years due to their generalization ability, in which the data is used as the source of supervision to itself. Thus, the expensive work of data labeling can be abandoned in self-supervised approaches compared to supervised ones. In the pre-text task of self-supervised learning, a model is trained with the unlabeled data, and the acquired knowledge is transferred in the down-stream step. Even though self-supervised learning shows a considerable improvement in the field of natural language processing and computer vision there exists a few works regarding time-series anomaly detection with self-supervised approaches, namely the works of Liu et al. [140] and Bailly et al. [52]. Designing novel pre-text tasks suitable for time-series chemical process data, employing the process knowledge in the pre-text step, and utilizing GANs, before and after convergence like SAAC, for self-labeling in the pre-text step can be explored as the future directions.

References

[1] S. J. Qin, Y. Dong, Q. Zhu, J. Wang, and Q. Liu, "Bridging systems theory and data science: A unifying review of dynamic latent variable analytics and process monitoring," *Annual Reviews in Control*, 2020.

[2] Q. Zhu, Q. Liu, and S. J. Qin, "Concurrent quality and process monitoring with canonical correlation analysis," *Journal of Process Control*, vol. 60, pp. 95–103, 2017.

[3] T. Wu, S. Liu, J. Zhang, and Y. Xiang, "Twitter spam detection based on deep learning," in *Proceedings of the australasian computer science week multiconference*, pp. 1–8, 2017.

[4] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE transactions on emerging topics in computational intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[5] R. Nawaratne, D. Alahakoon, D. De Silva, and X. Yu, "Spatiotemporal anomaly detection using deep learning for real-time video surveillance," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 393–402, 2019.

[6] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part i: Quantitative model-based methods," *Computers & chemical engineering*, vol. 27, no. 3, pp. 293–311, 2003.

[7] T. Ramesh, S. Shum, and J. Davis, “A structured framework for efficient problem solving in diagnostic expert systems,” *Computers & Chemical Engineering*, vol. 12, no. 9-10, pp. 891–902, 1988.

[8] L. W. Chen and M. Modarres, “Hierarchical decision process for fault administration,” *Computers & chemical engineering*, vol. 16, no. 5, pp. 425–448, 1992.

[9] B. M. Wise, N. Ricker, D. Veltkamp, and B. R. Kowalski, “A theoretical basis for the use of principal component models for monitoring multivariate processes,” *Process control and quality*, vol. 1, no. 1, pp. 41–51, 1990.

[10] P. Nomikos and J. F. MacGregor, “Monitoring batch processes using multiway principal component analysis,” *AIChE Journal*, vol. 40, no. 8, pp. 1361–1375, 1994.

[11] E. L. Russell, L. H. Chiang, and R. D. Braatz, “Fault detection in industrial processes using canonical variate analysis and dynamic principal component analysis,” *Chemometrics and intelligent laboratory systems*, vol. 51, no. 1, pp. 81–93, 2000.

[12] W. Ku, R. H. Storer, and C. Georgakis, “Disturbance detection and isolation by dynamic principal component analysis,” *Chemometrics and intelligent laboratory systems*, vol. 30, no. 1, pp. 179–196, 1995.

[13] S. W. Choi and I.-B. Lee, “Nonlinear dynamic process monitoring based on dynamic kernel PCA,” *Chemical engineering science*, vol. 59, no. 24, pp. 5897–5908, 2004.

[14] M. Jia, F. Chu, F. Wang, and W. Wang, “On-line batch process monitoring using batch dynamic kernel principal component analysis,” *Chemometrics and Intelligent Laboratory Systems*, vol. 101, no. 2, pp. 110–122, 2010.

[15] M. Kano, S. Tanaka, S. Hasebe, I. Hashimoto, and H. Ohno, “Monitoring independent components for fault detection,” *AIChE Journal*, vol. 49, no. 4, pp. 969–976, 2003.

[16] J.-M. Lee, C. Yoo, and I.-B. Lee, “Statistical process monitoring with independent component analysis,” *Journal of process control*, vol. 14, no. 5, pp. 467–485, 2004.

[17] L. Cai and X. Tian, "A new fault detection method for non-Gaussian process based on robust independent component analysis," *Process Safety and Environmental Protection*, vol. 92, no. 6, pp. 645–658, 2014.

[18] S. Zhang and C. Zhao, "Hybrid independent component analysis (H-ICA) with simultaneous analysis of high-order and second-order statistics for industrial process monitoring," *Chemometrics and Intelligent Laboratory Systems*, vol. 185, pp. 47–58, 2019.

[19] J. F. MacGregor, C. Jaeckle, C. Kiparissides, and M. Koutoudi, "Process monitoring and diagnosis by multiblock PLS methods," *AIChE Journal*, vol. 40, no. 5, pp. 826–838, 1994.

[20] S. J. Qin and T. J. McAvoy, "Nonlinear PLS modeling using neural networks," *Computers & Chemical Engineering*, vol. 16, no. 4, pp. 379–391, 1992.

[21] B. Xu and Q. Zhu, "Online quality-relevant monitoring with dynamic weighted partial least squares," *Industrial & Engineering Chemistry Research*, vol. 59, no. 48, pp. 21124–21132, 2020.

[22] L. H. Chiang, E. L. Russell, and R. D. Braatz, "Fault diagnosis in chemical processes using Fisher discriminant analysis, discriminant partial least squares, and principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 50, no. 2, pp. 243–252, 2000.

[23] M. Onel, C. A. Kieslich, and E. N. Pistikopoulos, "A nonlinear support vector machine-based feature selection approach for fault detection and diagnosis: Application to the tennessee eastman process," *AIChE Journal*, vol. 65, no. 3, pp. 992–1005, 2019.

[24] Q. Zhu, S. J. Qin, and Y. Dong, "Dynamic latent variable regression for inferential sensor modeling and monitoring," *Computers & Chemical Engineering*, vol. 137, p. 106809, 2020.

[25] S. Yin, S. X. Ding, A. Haghani, H. Hao, and P. Zhang, "A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process," *Journal of process control*, vol. 22, no. 9, pp. 1567–1581, 2012.

[26] J. C. Hoskins and D. M. Himmelblau, "Artificial neural network models of knowledge representation in chemical engineering," *Computers & Chemical Engineering*, vol. 12, no. 9-10, pp. 881–890, 1988.

[27] V. Venkatasubramanian and K. Chan, "A neural network methodology for process fault diagnosis," *AIChE Journal*, vol. 35, no. 12, pp. 1993–2002, 1989.

[28] J. Zhang, A. J. Morris, and G. A. Montague, "Fault diagnosis of a CSTR using fuzzy neural networks," in *Artificial Intelligence in Real-Time Control 1994*, pp. 153–158, Elsevier, 1995.

[29] K. Watanabe, S. Hirota, L. Hou, and D. Himmelblau, "Diagnosis of multiple simultaneous fault via hierarchical artificial neural networks," *AIChE Journal*, vol. 40, no. 5, pp. 839–848, 1994.

[30] C.-S. Tsai and C.-T. Chang, "Dynamic process diagnosis via integrated neural networks," *Computers & chemical engineering*, vol. 19, pp. 747–752, 1995.

[31] R. Eslamloueyan, "Designing a hierarchical neural network based on fuzzy clustering for fault diagnosis of the tennessee–eastman process," *Applied soft computing*, vol. 11, no. 1, pp. 1407–1415, 2011.

[32] M. A. A. Rad and M. J. Yazdanpanah, "Designing supervised local neural network classifiers based on EM clustering for fault diagnosis of tennessee eastman process," *Chemometrics and Intelligent Laboratory Systems*, vol. 146, pp. 149–157, 2015.

[33] P. Jiang, Z. Hu, J. Liu, S. Yu, and F. Wu, "Fault diagnosis based on chemical sensor data with an active deep neural network," *Sensors*, vol. 16, no. 10, p. 1695, 2016.

[34] H. Wu and J. Zhao, "Deep convolutional neural network model based chemical process fault diagnosis," *Computers & chemical engineering*, vol. 115, pp. 185–197, 2018.

[35] P. Park, P. D. Marco, H. Shin, and J. Bang, "Fault detection and diagnosis using combined autoencoder and long short-term memory network," *Sensors*, vol. 19, no. 21, p. 4612, 2019.

[36] S. Zhang, K. Bi, and T. Qiu, "Bidirectional recurrent neural network-based chemical process fault diagnosis," *Industrial & Engineering Chemistry Research*, vol. 59, no. 2, pp. 824–834, 2019.

[37] G. S. Chadha, A. Panambilly, A. Schwung, and S. X. Ding, "Bidirectional deep recurrent neural networks for process fault classification," *ISA transactions*, vol. 106, pp. 330–342, 2020.

[38] Y. Wang, Z. Pan, X. Yuan, C. Yang, and W. Gui, "A novel deep learning based fault diagnosis approach for chemical process with extended deep belief network," *ISA transactions*, vol. 96, pp. 457–467, 2020.

[39] P. Agarwal, J. I. M. Gonzalez, A. Elkamel, and H. Budman, "Hierarchical deep recurrent neural network based method for fault detection and diagnosis," *arXiv preprint arXiv:2012.03861*, 2020.

[40] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[41] W. Yan, P. Guo, Z. Li, *et al.*, "Nonlinear and robust statistical process monitoring based on variant autoencoders," *Chemometrics and Intelligent Laboratory Systems*, vol. 158, pp. 31–40, 2016.

[42] F. Cheng, Q. P. He, and J. Zhao, "A novel process monitoring approach based on variational recurrent autoencoder," *Computers & Chemical Engineering*, vol. 129, p. 106515, 2019.

[43] J. Yu and C. Zhang, "Manifold regularized stacked autoencoders-based feature learning for fault detection in industrial processes," *Journal of Process Control*, vol. 92, pp. 119–136, 2020.

[44] J. Yu, X. Liu, and L. Ye, "Convolutional long short-term memory autoencoder-based feature learning for fault detection in industrial processes," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–15, 2020.

[45] A. Munawar, P. Vinayavekhin, and G. De Magistris, "Limiting the reconstruction capability of generative neural network using negative learning," in *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2017.

[46] Y. Yamanaka, T. Iwata, H. Takahashi, M. Yamada, and S. Kanai, "Autoencoding binary classifiers for supervised anomaly detection," in *Pacific Rim International Conference on Artificial Intelligence*, pp. 647–659, Springer, 2019.

[47] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft, "Deep semi-supervised anomaly detection," *arXiv preprint arXiv:1906.02694*, 2019.

[48] J. Kim, K. Jeong, H. Choi, and K. Seo, "GAN-based anomaly detection in imbalance problems," in *European Conference on Computer Vision*, pp. 128–145, Springer, 2020.

[49] P. Wu, J. Liu, and F. Shen, "A deep one-class neural network for anomalous event detection in complex scenes," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 7, pp. 2609–2622, 2019.

[50] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*, pp. 4393–4402, PMLR, 2018.

[51] P. Liznerski, L. Ruff, R. A. Vandermeulen, B. J. Franks, M. Kloft, and K.-R. Müller, "Explainable deep one-class classification," *arXiv preprint arXiv:2007.01760*, 2020.

[52] R. Bailly, M. Malfante, C. Allier, L. Ghenim, and J. Mars, "Deep anomaly detection using self-supervised learning: application to time series of cellular data," in *3rd International Conference on Advances in Signal Processing and Artificial Intelligence*, 2021.

[53] I. Golan and R. El-Yaniv, "Deep anomaly detection using geometric transformations," *arXiv preprint arXiv:1805.10917*, 2018.

[54] J. Tack, S. Mo, J. Jeong, and J. Shin, "Csi: Novelty detection via contrastive learning on distributionally shifted instances," *arXiv preprint arXiv:2007.08176*, 2020.

[55] A.-S. Collin and C. De Vleeschouwer, "Improved anomaly detection by training an autoencoder with skip connections on images corrupted with stain-shaped noise," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 7915–7922, IEEE, 2021.

[56] M. Salehi, A. Eftekhar, N. Sadjadi, M. H. Rohban, and H. R. Rabiee, "Puzzle-AE: Novelty detection in images through solving puzzles," *arXiv preprint arXiv:2008.12959*, 2020.

[57] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[58] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[59] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[60] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker

discovery," in *International conference on information processing in medical imaging*, pp. 146–157, Springer, 2017.

[61] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar, "Adversarially learned anomaly detection," in *2018 IEEE International conference on data mining (ICDM)*, pp. 727–736, IEEE, 2018.

[62] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks," *Medical image analysis*, vol. 54, pp. 30–44, 2019.

[63] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Asian conference on computer vision*, pp. 622–637, Springer, 2018.

[64] C. Wang, Y.-M. Zhang, and C.-L. Liu, "Anomaly detection via minimum likelihood generative adversarial networks," in *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 1121–1126, IEEE, 2018.

[65] P. C. Ngo, A. A. Winarto, C. K. L. Kou, S. Park, F. Akram, and H. K. Lee, "Fence GAN: Towards better anomaly detection," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 141–148, IEEE, 2019.

[66] J.-P. Schulze, P. Sperl, and K. Böttinger, "Double-adversarial activation anomaly detection: Adversarial autoencoders are anomaly generators," *arXiv preprint arXiv:2101.04645*, 2021.

[67] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[68] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[69] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[70] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[71] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba, "Addressing the rare word problem in neural machine translation," *arXiv preprint arXiv:1410.8206*, 2014.

[72] Z. Cui, R. Ke, Z. Pu, and Y. Wang, "Stacked bidirectional and unidirectional LSTM recurrent neural network for forecasting network-wide traffic state with missing values," *Transportation Research Part C: Emerging Technologies*, vol. 118, p. 102674, 2020.

[73] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[74] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[75] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional LSTM networks for improved phoneme classification and recognition," in *International conference on artificial neural networks*, pp. 799–804, Springer, 2005.

[76] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *2013 IEEE workshop on automatic speech recognition and understanding*, pp. 273–278, IEEE, 2013.

[77] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller, "Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks," in *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 2164–2168, IEEE, 2014.

[78] F. Grisoni, M. Moret, R. Lingwood, and G. Schneider, "Bidirectional molecule generation with recurrent neural networks," *Journal of chemical information and modeling*, vol. 60, no. 3, pp. 1175–1183, 2020.

[79] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[80] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[81] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.

[82] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, pp. 4–11, 2014.

[83] A. Mallak and M. Fathi, "Sensor and component fault detection and diagnosis for hydraulic machinery integrating LSTM autoencoder detector and diagnostic classifiers," *Sensors*, vol. 21, no. 2, p. 433, 2021.

[84] L. Ruff, R. A. Vandermeulen, B. J. Franks, K.-R. Müller, and M. Kloft, "Rethinking assumptions in deep anomaly detection," *arXiv preprint arXiv:2006.00339*, 2020.

[85] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Generative adversarial networks and adversarial autoencoders: Tutorial and survey," *arXiv preprint arXiv:2111.13282*, 2021.

[86] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[87] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[88] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, Citeseer, 2013.

[89] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," *arXiv preprint arXiv:1611.02163*, 2016.

[90] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, pp. 214–223, PMLR, 2017.

[91] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein GANs," *Advances in neural information processing systems*, vol. 30, 2017.

[92] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[93] N. Li and F. Chang, "Video anomaly detection and localization via multivariate Gaussian fully convolution adversarial autoencoder," *Neurocomputing*, vol. 369, pp. 92–105, 2019.

[94] R. Chalapathy, E. Toth, and S. Chawla, "Group anomaly detection using deep generative models," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 173–189, Springer, 2018.

[95] A. Kadurin, S. Nikolenko, K. Khrabrov, A. Aliper, and A. Zhavoronkov, "druGAN: an advanced generative adversarial autoencoder model for de novo generation of new molecules with desired molecular properties in silico," *Molecular pharmaceutics*, vol. 14, no. 9, pp. 3098–3104, 2017.

[96] S. K. Lim, Y. Loo, N.-T. Tran, N.-M. Cheung, G. Roig, and Y. Elovici, "Doping: Generative data augmentation for unsupervised anomaly detection with GAN," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 1122–1127, IEEE, 2018.

[97] K. Jang, S. Hong, M. Kim, J. Na, and I. Moon, "Adversarial autoencoder-based feature learning for fault detection in industrial processes," *IEEE Transactions on Industrial Informatics*, 2021.

[98] M. Pourreza, B. Mohammadi, M. Khaki, S. Bouindour, H. Snoussi, and M. Sabokrou, "G2D: Generate to detect anomaly," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2003–2012, 2021.

[99] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[100] A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta, "Beyond skip connections: Top-down modulation for object detection," *arXiv preprint arXiv:1612.06851*, 2016.

[101] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," 2015.

[102] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European conference on computer vision*, pp. 646–661, Springer, 2016.

[103] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *arXiv preprint arXiv:1605.07648*, 2016.

[104] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[105] T. Tong, G. Li, X. Liu, and Q. Gao, "Image super-resolution using dense skip connections," in *Proceedings of the IEEE international conference on computer vision*, pp. 4799–4807, 2017.

[106] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, pp. 630–645, Springer, 2016.

[107] A. E. Orhan and X. Pitkow, "Skip connections eliminate singularities," *arXiv preprint arXiv:1701.09175*, 2017.

[108] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[109] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[110] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[111] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[112] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[113] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[114] C. Rieth, B. Amsel, R. Tran, and M. Cook, "Additional tennessee eastman process simulation data for anomaly detection evaluation," *Harvard Dataverse*, vol. 1, 2017.

[115] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating probability with undersampling for unbalanced classification," in *2015 IEEE Symposium Series on Computational Intelligence*, pp. 159–166, IEEE, 2015.

[116] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[117] F. V. Massoli, F. Falchi, A. Kantarci, Ş. Akti, H. K. Ekenel, and G. Amato, "MOCCA: Multi-layer one-class classification for anomaly detection," *arXiv preprint arXiv:2012.12111*, 2020.

[118] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & chemical engineering*, vol. 17, no. 3, pp. 245–255, 1993.

[119] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, pp. 413–422, IEEE, 2008.

[120] Y. Zhou, X. Liang, W. Zhang, L. Zhang, and X. Song, "VAE-based deep SVDD for anomaly detection," *Neurocomputing*, vol. 453, pp. 131–140, 2021.

[121] Y. Zhang, "Enhanced statistical analysis of nonlinear processes using KPCA, KICA and SVM," *Chemical Engineering Science*, vol. 64, no. 5, pp. 801–811, 2009.

[122] N. Basha, M. Z. Sheriff, C. Kravaris, H. Nounou, and M. Nounou, "Multiclass data classification using fault detection-based techniques," *Computers & Chemical Engineering*, vol. 136, p. 106786, 2020.

[123] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional autoencoders for hierarchical feature extraction," in *International conference on artificial neural networks*, pp. 52–59, Springer, 2011.

[124] A. Makhzani and B. Frey, "Winner-take-all autoencoders," *arXiv preprint arXiv:1409.2752*, 2014.

[125] D. Abati, A. Porrello, S. Calderara, and R. Cucchiara, "Latent space autoregression for novelty detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 481–490, 2019.

[126] P. Perera, R. Nallapati, and B. Xiang, "OCGAN: One-class novelty detection using GANs with constrained latent representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2898–2906, 2019.

[127] P. Mishra, C. Piciarelli, and G. L. Foresti, "A neural network for image anomaly detection with deep pyramidal representations and dynamic routing," *International Journal of Neural Systems*, vol. 30, no. 10, p. 2050060, 2020.

[128] J. Fan, Q. Zhang, J. Zhu, M. Zhang, Z. Yang, and H. Cao, "Robust deep auto-encoding Gaussian process regression for unsupervised anomaly detection," *Neurocomputing*, vol. 376, pp. 180–190, 2020.

[129] S. Dhar and B. G. Torres, "DOC3-deep one class classification using contradictions," *arXiv preprint arXiv:2105.07636*, 2021.

[130] C. A. Rieth, B. D. Amsel, R. Tran, and M. B. Cook, "Issues and advances in anomaly detection evaluation for joint human-automated systems," in *International Conference on Applied Human Factors and Ergonomics*, pp. 52–63, Springer, 2017.

[131] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[132] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[133] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[134] Y. Du and D. Du, "Fault detection using empirical mode decomposition based PCA and CUSUM with application to the tennessee eastman process," *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 488–493, 2018.

[135] S. Heo and J. H. Lee, "Fault detection and classification using artificial neural networks," *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 470–475, 2018.

[136] Y. Wang, H. Yang, X. Yuan, Y. Schardt, C. Yang, and W. Gui, "Deep learning for fault-relevant feature extraction and fault classification with stacked supervised auto-encoder," *Journal of Process Control*, vol. 92, pp. 79–89, 2020.

[137] Q. Zhang, J. Zhang, J. Zou, and S. Fan, "A novel fault diagnosis method based on stacked lstm," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 790–795, 2020.

[138] D. Wu and J. Zhao, "Process topology convolutional network model for chemical process fault diagnosis," *Process Safety and Environmental Protection*, vol. 150, pp. 93–109, 2021.

[139] Y. He, H. Shi, S. Tan, B. Song, and J. Zhu, "Multiblock temporal convolution network-based temporal-correlated feature learning for fault diagnosis of multivariate processes," *Journal of the Taiwan Institute of Chemical Engineers*, 2021.

[140] M. Liu, Z. Xu, and Q. Xu, "DeepFIB: Self-imputation for time series anomaly detection," *arXiv preprint arXiv:2112.06247*, 2021.

[141] J. Yin and X. Yan, "Mutual information–dynamic stacked sparse autoencoders for fault detection," *Industrial & Engineering Chemistry Research*, vol. 58, no. 47, pp. 21614–21624, 2019.

[142] C.-C. Huang, T. Chen, and Y. Yao, "Mixture discriminant monitoring: A hybrid method for statistical process monitoring and fault diagnosis/isolation," *Industrial & Engineering Chemistry Research*, vol. 52, no. 31, pp. 10720–10731, 2013.

[143] N. Amini and Q. Zhu, "Fault detection and diagnosis with a novel source-aware autoencoder and deep residual neural network," *Neurocomputing*, 2021.

[144] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pp. 4–11, 2014.

[145] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.

[146] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network

with a local denoising criterion.," *Journal of machine learning research*, vol. 11, no. 12, 2010.