Unsupervised Time Series Outlier Detection with Diversity-Driven Convolutional Ensembles

David Campos[1*], Tung Kieu[1*], Chenjuan Guo[1+], Feiteng Huang[2], Kai Zheng[3], Bin Yang[1], and Christian S. Jensen[1]
[1]Aalborg University, Denmark [2]Huawei Cloud Database Innovation Lab, China
[3]University of Electronic Science and Technology of China, China
[1]{dgcc, tungkvt, cguo, byang, csj}@cs.aau.dk, [2]huangfeiteng@huawei.com, [3]zhengkai@uestc.edu.cn

ABSTRACT

With the sweeping digitalization of societal, medical, industrial, and scientific processes, sensing technologies are being deployed that produce increasing volumes of time series data, thus fueling a plethora of new or improved applications. In this setting, outlier detection is frequently important, and while solutions based on neural networks exist, they leave room for improvement in terms of both accuracy and efficiency. With the objective of achieving such improvements, we propose a diversity-driven, convolutional ensemble. To improve accuracy, the ensemble employs multiple basic outlier detection models built on convolutional sequence-to-sequence autoencoders that can capture temporal dependencies in time series. Further, a novel diversity-driven training method maintains diversity among the basic models, with the aim of improving the ensemble's accuracy. To improve efficiency, the approach enables a high degree of parallelism during training. In addition, it is able to transfer some model parameters from one basic model to another, which reduces training time. We report on extensive experiments using real-world multivariate time series that offer insight into the design choices underlying the new approach and offer evidence that it is capable of improved accuracy and efficiency.

PVLDB Reference Format:
David Campos, Tung Kieu, Chenjuan Guo, Feiteng Huang, Kai Zheng, Bin Yang, and Christian S. Jensen. Unsupervised Time Series Outlier Detection with Diversity-Driven Convolutional Ensembles. PVLDB, 15(3): 611 - 623, 2022.
doi:10.14778/3494124.3494142

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/d-gcc/CAE-Ensemble.

1 INTRODUCTION

As part of the continued digitization, processes are increasingly being instrumented with sensors, which offer monitoring capabilities with the objective of supporting a variety of applications [25, 33, 50]. The monitoring of processes results in time series data [9, 21]. In many cases, it is important to be able to use this data to identify outliers or anomalies [1]. Outliers are relatively uncommon observations that differ from the remaining observations in a series. The accurate detection of outliers is critical in applications such as health, manufacturing, and transportation [15, 37, 38].

We consider *unsupervised* outlier detection, meaning that we do not rely on outlier labeled data for training, which offers two benefits. First, labeling by human experts is time consuming and labor intensive; thus, time series often come without outlier labels. Second, the unsupervised setting eliminates the reliance on manual labeling, where some outliers may go unnoticed, thus potentially enabling the detection of unanticipated types of outliers.

Unsupervised outlier detection has traditionally been supported by linear methods [16] that require human experts to set model parameters for the specific scenarios. The recent use of deep learning techniques enables substantial detection accuracy improvements over the linear models because they are able to learn non-linear features from the data, such as complex temporal dependencies, without requiring explicit supervision by human experts.

Autoencoders (AE) [17, 26] represent a popular and powerful deep learning based approach to unsupervised outlier detection. An AE consists of an *encoding* phase that compresses an original time series $\mathcal{T}$ into a compact representation and a subsequent *decoding* phase that reconstructs an output time series $\hat{\mathcal{T}}$ from the compact representation. The compact representation is only able to capture representative patterns that reflect general patterns in the original time series. As a result, the reconstructed time series $\hat{\mathcal{T}}$ is unlikely to capture outliers that by virtue of being outliers are not representative. Thus, if the difference between observations in $\mathcal{T}$ and in $\hat{\mathcal{T}}$, called the reconstruction error, exceeds a threshold ϵ, this suggests an outlier. For example, this occurs at time 3 in Figure 1.

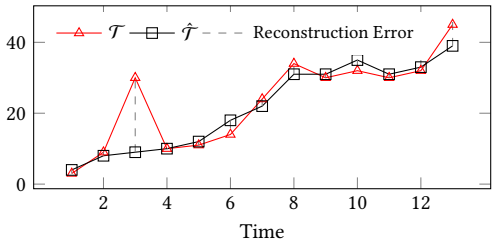


Figure 1: AE-based unsupervised outlier detection.

An autoencoder ensemble (AE-Ensemble) employs multiple AEs to avoid a single AE being overfit to original data [7, 36], which often further improves accuracy. In particular, RAE-Ensemble achieve

∗: Equal contributions. +: Corresponding author.

Proceedings of the VLDB Endowment, Vol. 15, No. 3 ISSN 2150-8097.
doi:10.14778/3494124.3494142

high accuracy using an ensemble of recurrent neural network based autoencoders (RAEs) [27], where a RAE is able to capture temporal dependencies among the observations in time series [28, 35],e.g., the increasing trend shown in Figure 1. This is a highly desirable property compared to classic AEs that use, e.g., feedforward neural networks, which fail to capture temporal dependencies.

However, RAE-Ensemble may still be improved in terms of accuracy and efficiency. First, while an ensemble often benefits from diverse basic models [2], the basic models in RAE-Ensemble may not be sufficiently diverse. Specifically, RAE-Ensemble generates basic models with randomly different network structures by randomly adding skip connections. However, the resulting basic models may not necessarily return diverse results, offering room for higher accuracy. Second, the basic models in RAE-Ensemble are based on Recurrent Neural Networks (RNNs), which are *inefficient*, as an RNN must process the time series data sequentially, as opposed to in parallel, due to its recurrence nature.

We propose CAE-Ensemble, a novel autoencoder ensemble approach that uses convolutional neural networks and aims to improve accuracy and efficiency by overcoming the limitations of RAE-Ensemble. First, we introduce a diversity metric that makes possible to assess the suitability of a newly generated basic model given the current state of the ensemble. This ensures the diversity among the basic models. Second, we replace the RNN framework of RAE-Ensemble with Convolutional Neural Networks (CNN) [14, 31], in order to capture temporal dependencies, with the benefit of achieving a high degree of training parallelism, thus improving efficiency. Third, to further reduce the training time, CAE-Ensemble integrates a transfer learning method based on so-called born-again networks [13] that share a portion of the trained parameters from one basic model with other basic models. Fourth, we propose a fully unsupervised strategy for selecting hyperparameters without relying on ground truth outlier labels.

To summarize, we make the following contributions. First, we propose a convolutional sequence-to-sequence autoencoder CAE for outlier detection that is capable to capture temporal dependencies with a high degree of training parallelism. Second, to improve the accuracy and efficiency, we propose a diversity-driven ensemble using CAEs as its basic models, along with a parameter transfer based training strategy. Third, we propose a fully unsupervised strategy to select hyper-parameters, and finally, we report insights from extensive experiments with real-world time series data sets to justify design choices in our proposal.

The paper is organized as follows. Section 2 formalizes the problem. Section 3 elaborates CAE-Ensemble. Section 4 reports on the experiments. Section 5 reviews related work. Section 6 concludes.

2 PRELIMINARIES

Time Series Outlier Detection. A time series $\mathcal{T} = \langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_C \rangle$ is a sequence of C observations, where each observation $\mathbf{s}_t \in \mathbb{R}^D$ is a D-dimensional vector. If $D = 1$, $\mathcal{T}$ is *univariate*, and if $D > 1$, $\mathcal{T}$ is *multivariate*, or *multidimensional*. Given a time series $\mathcal{T} = \langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_C \rangle$, the outlier score $\mathcal{OS}(\mathbf{s}_t)$ for observation $\mathbf{s}_t$ is a value such that the higher $\mathcal{OS}(\mathbf{s}_t)$ is, the more likely it is that $\mathbf{s}_t$ is an outlier. Given an outlier score threshold ϵ, outliers are the observations in $\mathcal{T}$ whose outlier score exceeds ϵ. Usually, threshold ϵ is defined according to domain knowledge.

In the experiments, to evaluate accuracy in a fair and meaningful manner, we use different metrics, including metrics that (1) require such specific thresholds and (2) do not rely on thresholds, e.g., when domain knowledge on selecting such thresholds is unavailable.

Autoencoders. A classic autoencoder AE consists of an encoder and a decoder, where each is a feed-forward neural network. The encoder transforms a D-dimensional input $\mathbf{x}$ to an intermediate and compact vector $\mathbf{h} \in R^M$, where $M < D$. Then, the decoder transforms the intermediate vector $\mathbf{h}$ to an output vector $\hat{\mathbf{x}} \in R^D$ that approximates the input vector. The learning goal is to minimize the difference between the input $\mathbf{x}$ and the reconstructed $\hat{\mathbf{x}}$.

Given a set of training inputs $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_C]$, the corresponding autoencoder outputs $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_C]$, where $\hat{\mathbf{x}}_i = \mathrm{AE}(\mathbf{x}_i)$, and the learnable parameters θ_{AE} of the AE, the objective function $\mathcal{L}_{AE}$ used for training is formulated in Equation 1.

$$\arg\min_{\theta_{AE}} \mathcal{L}_{AE} = \arg\min_{\theta_{AE}} \frac{1}{C} \sum_{i=1}^{C} (\mathbf{x}_i - \hat{\mathbf{x}}_i)^2 \tag{1}$$

The reconstruction error (*RE*) is defined as $||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2^2$. If the *RE* exceeds threshold ϵ, $\mathbf{x}_i$ is considered as an outlier [39].

Applications of AEs often relate to non-sequential data, such as images, and AEs do not consider relationships between observations over time, e.g., the increasing trend as shown in Figure 1, as summarized in Table 1.

Table 1: Autoencoder-based models for outlier detection.

	Temporal-dependencies	*Efficiency*	*Diversity*
AE	✗	✓	✗
RAE	✓	✗	✗
CAE	✓	✓	✗
AE-Ensemble	✗	✓	✓ (Implicit)
RAE-Ensemble	✓	✗	✓ (Implicit)
CAE-Ensemble	✓	✓	✓ (Explicit)

Recurrent Autoencoders. A sequence-to-sequence model targets the analysis of sequential data, by preserving the relationships surrounding each data point, e.g., temporal dependencies in time series. For example, if a time series is growing, e.g., as shown in Figure 1, such a model is capable to taking this growing tendency into account when making predictions or detecting outliers.

A recurrent autoencoder (RAE) follows a sequence-to-sequence model using a recurrent neural network (RNN). In the encoder, an observation $\mathbf{s}_t$ in time series $\mathcal{T}$ and a hidden state at the previous timestamp $\mathbf{h}_{t-1}^{(E)}$ are fed into an RNN unit to compute the current hidden state $\mathbf{h}_t^{(E)}$ at timestamp t using Equation 2.

$$\mathbf{h}_t^{(E)} = \mathrm{RNN}(\mathbf{s}_t, \mathbf{h}_{t-1}^{(E)}). \tag{2}$$

Here, $\mathrm{RNN}(\cdot)$ is an abstraction that can be a Long Short Term Memory (LSTM) [20] or a Gated Recurrent Unit (GRU) [11]. Once $\mathbf{h}_t^{(E)}$ is computed, it is fed into the next RNN unit to compute the hidden state at timestamp $t + 1$.

In the decoder, the time series is reconstructed in reverse order. Specifically, the last hidden state in the encoder $\mathbf{h}_C^{(E)}$ is used as the first hidden state of the decoder $\mathbf{h}_C^{(D)}$, i.e., $\mathbf{h}_C^{(E)} = \mathbf{h}_C^{(D)}$. Based on the previous hidden state $\mathbf{h}_{t+1}^{(D)}$ of the decoder and the previous reconstructed observation $\hat{\mathbf{s}}_{t+1}$, the decoder reconstructs the current observation $\hat{\mathbf{s}}_t = \text{RNN}(\hat{\mathbf{s}}_{t+1}, \mathbf{h}_{t+1}^{(D)})$ using an RNN unit.

Similar to an AE, an RAE identifies the learnable parameters that minimize the difference between the reconstructed time series and the original time series. Then, given the reconstructed observations represented by $\hat{\mathcal{T}} = \langle \hat{\mathbf{s}_1}, \hat{\mathbf{s}_2}, \ldots, \hat{\mathbf{s}_C} \rangle$, it is possible to calculate the differences from the original time series $\mathcal{T}$. Specifically, the outlier score for observation $\mathbf{s}_i$ is defined as $||\mathbf{s}_i - \hat{\mathbf{s}_i}||_2^2$.

Despite the good accuracy achieved by RAE [27], they suffer from low efficiency. In an RNN, the computation of one state takes as input the previous state, making RNN learning unparallelizable [35]. To improve efficiency, we propose a convolutional sequence-to-sequence autoencoder CAE in Section 3.1 that captures temporal dependencies without recursive computations.

Autoencoder Ensembles. Ensemble models combine multiple, diverse basic models and often achieve better accuracy than a single model by avoiding overfiting and underfitting (a.k.a., achieving better bias-variance trade-off) . To achieve improved accuracy, ensembles need to be designed so that the individual basic models are diverse and contribute differently to solving the problem [2].

Existing autoencoder ensembles often create different basic models randomly, without explicitly quantifying their diversity. For example, AE-Ensemble [7] creates multiple AEs by removing connections between neurons at random in feed-forward neural networks, and RAE-Ensemble [27] incorporates random skip connections to generate RAEs with different network structures. In other words, existing autoencoder ensembles consider diversity only implicitly. In Section 3.2, we propose a metric to quantify the diversities among different basic models and incorporate this metric explicitly into the objective function, making diversity part of the learning goals. This explains the "Diversity" column in Table 1.

Summary. Table 1 summarizes the existing autoencoder based models. The first three do not use ensembles and thus do not offer any diversity. AE is efficient, as matrix computations can be parallelized easily, but is unable to capture temporal dependencies. RAE is able to capture temporal dependencies, but is inefficient due to its recursive computations. We propose an efficient CAE that is able to capture temporal dependencies. We offer empirical evidence to justify this in Section 4.

The last three models are ensembles whose basic models employ the first three models, respectively. Thus, they have the same properties as their corresponding basic models. AE-Ensemble and RAE-Ensemble offer limited diversity because they generate basic models with randomly selected network structures without explicitly quantify the diversity in the learning processes. In contrast, we propose a metric that quantifies the diversity between basic models, and we judiciously design an objective function to incorporate the diversity metric to make CAE-Ensemble a diversity-driven ensemble. In addition, we propose a parameter sharing mechanism to improve the training efficiency of CAE-Ensemble.

3 THE CAE-ENSEMBLE FRAMEWORK

This section details the proposed CAE-Ensemble. In Section 3.1, we first cover basic CAE models using convolutional sequence-to-sequence autoencoders that are able to capture temporal dependencies in time series data while ensuring efficiency. Next, in Section 3.2, we describe a diversity-driven ensemble CAE-Ensemble that consists of the basic CAE models while taking into account diversity during training, along with an efficient training method using parameter transferring for the proposed ensemble. Finally, in Section 3.3, we propose an strategy for tuning hyperparameters in a fully unsupervised manner. The complete process is summarized in Algorithm 1 and a framework overview is shown in Figure 2.

Algorithm 1: CAE-Ensemble

Input : Raw time series $\mathcal{T}$, Number of Basic Models M
Output: Outlier scores

$outlierScores[\,] \leftarrow \emptyset$, $savedParam \leftarrow \emptyset$;
$\mathcal{T} \leftarrow ReScale(\mathcal{T})$;
$\beta, \lambda, w \leftarrow HyperparameterSelection(\mathcal{T})$; /* Sec 3.3 */
$\mathcal{T}_{windows} \leftarrow SplitIntoWindows(\mathcal{T}, w)$;
$\mathbf{X} \leftarrow Embedding(\mathcal{T}_{windows})$;
for $i \leftarrow 1$ **to** M **do**
  if $i = 1$ **then**
    /* Build the first basic model CAE f_1. */
    $\theta_{f_1} \leftarrow OptimizeNormal\,(f_1)$;
    $\hat{\mathbf{X}}^{(1)} \leftarrow f_1(\mathbf{X})$;
  else
    /* Build the i-th basic model CAE f_i. */
    $\theta_{f_i} \leftarrow OptimizeDiverse(f_i, \lambda, savedParam)$;
      /* *OptimizeDiverse* uses Eq. 13. */
    $\hat{\mathbf{X}}^{(i)} \leftarrow f_i(\mathbf{X})$;
  $outlierScores[i] \leftarrow \langle \|\mathbf{x}_1 - \hat{\mathbf{x}}_1^{(i)}\|_2^2, \ldots, \|\mathbf{x}_w - \hat{\mathbf{x}}_w^{(i)}\|_2^2 \rangle$;
  $savedParam \leftarrow$ Randomly select the fraction β of the parameters θ_{f_i} from basic model CAE f_i;
return $median(outlierScores)$

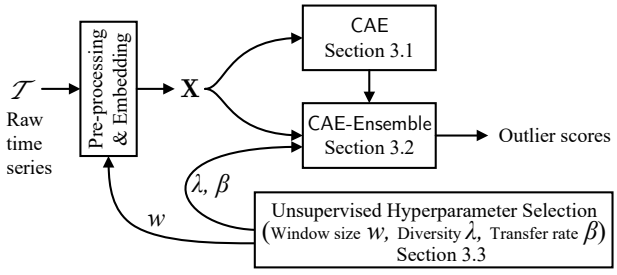


Figure 2: Framework overview.

Before introducing the model, we cover a pre-processing step: a *raw time series* is pre-processed into *time series windows* that are then used for training and testing [8]. The pre-processing first re-scales an observation x in the time series to $z = \frac{x - \mu}{\sigma}$, where μ is the mean and σ is the standard deviation of the observations

in the training time series. This is to prevent that magnitude differences among different dimensions in a time series affect the reconstruction errors differently, which is a common pre-processing technique [40].

We then create sliding windows of size w that slide one observation at a time. For example, for $\mathcal{T} = \langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_C \rangle$, the first window is $\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_w \rangle$ and the second is $\langle \mathbf{s}_2, \mathbf{s}_3, \ldots, \mathbf{s}_{w+1} \rangle$, etc.

3.1 Convolutional Autoencoder CAE

We build the CAE-Ensemble from basic models that use convolution sequence-to-sequence autoencoders CAEs. A CAE combines convolutional neural networks (CNNs) with a sequence-to-sequence architecture, as shown in Figure 3. First, the CAE embeds a time series window $\mathcal{T}$ into vectors that capture both the content and positions of the observations in $\mathcal{T}$. These vectors are then fed to the encoder that employs a 1D CNN to extract features that capture temporal dependencies in $\mathcal{T}$ and then outputs the features as hidden states. Next, the decoder employs another 1D CNN to extract features from both the embedded vectors and the hidden states from the encoder. The output from the decoder is another set of hidden states. As a 1D CNN does not involve recursive computations, it is possible to perform 1D CNN for different timestamps in parallel. This improves efficiency. Finally, an attention layer is applied to combine the hidden states from the encoder and the decoder, the result of which is then used for reconstructing the time series.

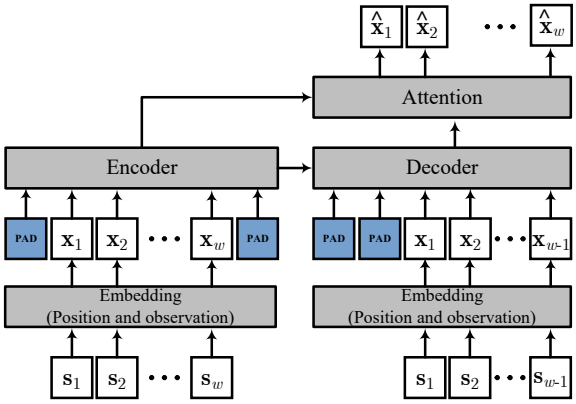


Figure 3: Convolutional autoencoder CAE.

3.1.1 Embedding. The embedding encompasses: *observation embedding* and *position embedding*.

Observation Embedding: The input of the encoder is a time series window $\mathcal{T} = \langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_w \rangle$, with vectors $\mathbf{s}_t \in \mathbb{R}^D$, which is then embedded into $\mathbf{V} = \langle \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_w \rangle$, where each vector $\mathbf{v}_t \in \mathbb{R}^{D'}$ is a D' dimensional representation for the original D dimensional observation $\mathbf{s}_t$ that captures its most typical features [23].

Specifically, we have $\mathbf{v}_t = f_s(\mathbf{W}_v \cdot \mathbf{s}_t + \mathbf{b}_v)$, where f_s represents a non-linear activation function that maps the original time series observation $\mathbf{s}_t$ to the embedding and $\mathbf{W}_v \in \mathbb{R}^{D' \times D}$ and $\mathbf{b}_v \in \mathbb{R}^{D'}$ are learnable parameters.

Position Embedding: When using CNNs instead of RNNs to model time series, we need to capture explicitly the positions of the observations in time series. Thus, we define a positional entry t for each observation in a time series. For example, the 5-th observation in a time series has the position entry 5. Specifically, positional entries $1, 2, \ldots, w$ are embedded into $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_w$, respectively, where each $\mathbf{p}_t \in \mathbb{R}^{D'}$, with $t \in [1, w]$, is a D' dimensional representation of the position of an observation. The transformation $\mathbf{p}_t = f_t(\mathbf{W}_p \cdot t + \mathbf{b}_p)$ is similar to the observation embedding.

Final Embedding: The observation and position embeddings are combined to obtain $\mathbf{X} = \langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_w \rangle = \langle \mathbf{v}_1 + \mathbf{p}_1, \mathbf{v}_2 + \mathbf{p}_2, \ldots, \mathbf{v}_w + \mathbf{p}_w \rangle$ as the input to the CNN. We use sum, instead of, e.g., concatenation, as existing studies show that summing achieves high effectiveness [14, 46]. A summary of the process is shown in Figure 4.

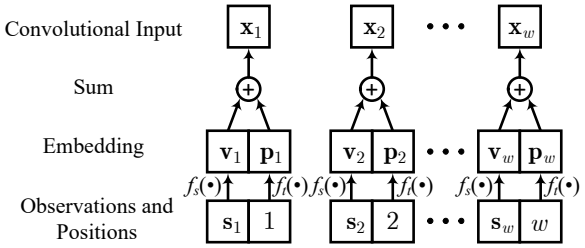


Figure 4: Embedding process.

3.1.2 Encoder. The encoder employs a stack of convolutional layers to capture typical temporal features and trends of time series. Figure 5 shows an encoder with 3 convolutional layers. Each layer

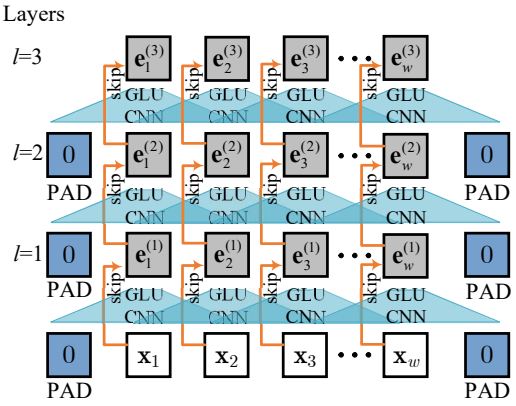


Figure 5: An example encoder with 3 layers.

is framed by a 1D convolution followed by a non-linear activation function. More specifically, the l-th layer takes as input a vector $\mathbf{E}^{(l)} \in \mathbb{R}^{w \times D'}$ and feeds a new vector $\mathbf{E}^{(l+1)} \in \mathbb{R}^{w \times D'}$ into the next layer, i.e., $(l+1)$-th convolution layer. The $\mathbf{E}$ vectors are the hidden states of the encoder.

$$\mathbf{E}^{(l+1)} = f_E(\mathbf{W}_E^{(l)} \otimes \mathrm{GLU}(\mathbf{E}^{(l)}) + \mathbf{b}_E^{(l)}) \tag{3}$$

The input to the first layer, $\mathbf{E}^{(0)}$, is the embedded input vector $\mathbf{X}$ with zero padding. We use padding in all layers to ensure that the output of the convolution has the same length as the input. Next, $\otimes$ denotes the 1D convolutional operator; k is the kernel size of the convolution operator; and $\mathbf{W}_E^{(l)} \in \mathbb{R}^{D' \times k}$ and $\mathbf{b}_E^{(l)} \in \mathbb{R}^{D'}$ are the kernel matrix and bias vector, respectively, at the l-th layer. Multiple different kernel matrices are often used. $f_E(\cdot)$ is a non-linear activation function.

To capture temporal information flow in time series better, we integrate Gated Linear Units (GLUs) [11] into the convolution layers. Specifically, a GLU is applied to $\mathbf{E}^{(l)}$ before applying the convolution operation. A GLU mimics the gating mechanisms used in RNNs and controls how much information along the temporal dimension should be kept or forgotten. First, from $\mathbf{E}^{(l)}$, we produce $\mathbf{A}_1^{(l)}$ and $\mathbf{A}_2^{(l)}$ by applying convolution (cf. Equation 5). Here, $\mathbf{W}_{A_i}^{(l)} \in \mathbb{R}^{D' \times k}$ and $\mathbf{b}_{A_i} \in \mathbb{R}^{D'}$ are the kernel matrix and bias vector, respectively. Then, we perform element-wise matrix multiplication, denoted by $\odot$, between $\mathbf{A}_1^{(l)}$ and the sigmoid function applied to $\mathbf{A}_2^{(l)}$, thus controlling how much information should be kept or forgotten (see Equation 4). GLU is specified in Equations 4 and 5.

$$\text{GLU}(\mathbf{E}^{(l)}) = \mathbf{A}_1^{(l)} \odot \sigma(\mathbf{A}_2^{(l)}) \tag{4}$$

$$\mathbf{A}_1^{(l)} = \mathbf{W}_{A_1}^{(l)} \otimes \mathbf{E}^{(l)} + \mathbf{b}_{A_1},\ \mathbf{A}_2^{(l)} = \mathbf{W}_{A_2}^{(l)} \otimes \mathbf{E}^{(l)} + \mathbf{b}_{A_2} \tag{5}$$

Finally, at each convolution layer, the output of the current convolution layer is added to the output from the previous convolution layer by using a skip or residual connection. Formally, we have $\mathbf{E}^{(l+1)} = \mathbf{E}^{(l+1)} + \mathbf{E}^{(l)}$. Using skip connections establish a flow for gradients across layers in the network, reducing the vanishing and the exploding gradients issues [18].

The encoder process is summarized in Figure 5, where the blue triangles represent the 1D convolution with GLU and the arrows indicate the skip connections.

3.1.3 Decoder. The decoder shown in Figure 6 is slightly different from the encoder. In the decoder, we ensure that the convolution at timestamp t only uses observations that appear no later than t. This aligns with the time series setting, where observations only to be seen in the future cannot be utilized. Thus, we apply padding to the input vector only before the first observation and then compute the hidden states for the decoder $\mathbf{D} \in \mathbb{R}^{w \times D'}$ as follows.

$$\mathbf{D}^{(l+1)} = f_D(\mathbf{W}_D^{(l)} \otimes \text{GLU}(\mathbf{D}^{(l)}) + \mathbf{b}_D^{(l)} + \mathbf{E}^{(l)}) \tag{6}$$

Here, $\otimes$ denotes the 1D convolution; k is the kernel size of the convolution operator; $\mathbf{W}_D^{(l)}$ and $\mathbf{b}_D^{(l)}$ are the kernel matrix and the bias vector at the l-th layer, respectively; $\mathbf{E}^{(l)}$ is the hidden state from the encoder at the same layer; and $f_D(\cdot)$ is a non-linear activation function.

As for the encoder, at each layer, the output of the current layer is added to the output from the previous layer by using a skip connection, i.e., $\mathbf{D}^{(l+1)} = \mathbf{D}^{(l+1)} + \mathbf{D}^{(l)}$.

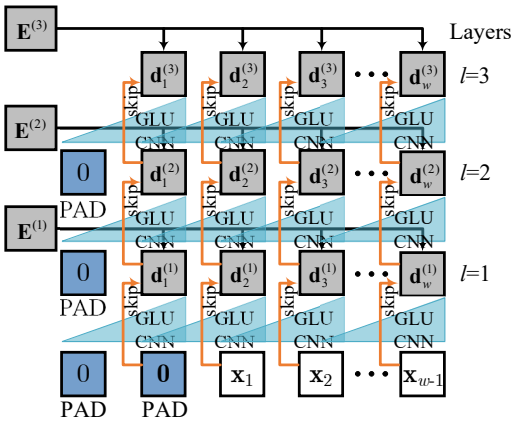


Figure 6: An example decoder with 3 layers.

3.1.4 Attention. We apply an attention mechanism [34] in the convolutional sequence-to-sequence autoencoder to capture local temporal patterns, e.g., periodicity, in an input time series by identifying similar observations that reoccur. Thus, the attention should consider observations that are relatively more important than others, given different specific settings.

To achieve this, we adopt a global attention method [34] that computes a context vector $\mathbf{c}_t^{(l)}$ that captures the relative importance of the observation at each timestamp t for each decoder layer l. Specifically, we first compute a state summary $\mathbf{z}_t^{(l)} = \mathbf{W}_z^{(l)} \cdot \mathbf{d}_t^{(l)} + \mathbf{b}_z^{(l)}$ using the decoder hidden state $\mathbf{d}_t^{(l)}$ at each timestamp. Then, we compute an attention score $\alpha_{tt'}^{(l)}$ that captures the temporal correlation between the observation at timestamp t at decoder layer l and the observation at timestamp t' at encoder layer l. Attention score $\alpha_{tt'}^{(l)}$ is computed as follows.

$$\alpha_{tt'}^{(l)} = \frac{\exp(\mathbf{z}_t^{(l)} \cdot \mathbf{e}_{t'}^{(l)})}{\sum_{t'=1}^{w} \exp(\mathbf{z}_t^{(l)} \cdot \mathbf{e}_{t'}^{(l)})} \tag{7}$$

Here, $\mathbf{e}_{t'}^{(l)}$, $t' \in [1, w]$, is the output of encoder layer l at timestamp t, and $\cdot$ denotes the dot product.

Next, a context vector $\mathbf{c}_t^{(l)} = \sum_{t'=1}^{w} \alpha_{tt'}^{(l)} \mathbf{e}_{t'}^{(l)}$ captures the temporal correlation between the observation at timestamp t at the decoder layer l and all observations at the encoder layer l. Once $\mathbf{c}_t^{(l)}$ is computed, we update the output of the corresponding decoder layer $\mathbf{d}_t^{(l)}$ by replacing $\mathbf{d}_t^{(l)}$ by $\mathbf{c}_t^{(l)} + \mathbf{d}_t^{(l)}$, i.e., $\mathbf{D}^{(l)} = \mathbf{C}^{(l)} + \mathbf{D}^{(l)}$.

In Figure 7, we summarize the attention mechanism. The attention score $\alpha_t^{(l)}$ between the decoder state summary and the encoder hidden states is calculated and weighted with the embedding information to obtain the context vector $\mathbf{c}_t^{(l)}$. Finally, represented by the orange lines, the output for the decoder is updated using the context to obtain the reconstruction $\hat{\mathbf{X}}$.

3.1.5 Reconstruction. At the last L-th convolution layer in the decoder, we obtain a final hidden state $\mathbf{D}^{(L+1)}$. We reconstruct the time series from this state using a simple fully connected neural network. Specifically, we have $\hat{\mathbf{X}} = f_R(\mathbf{W}_R \otimes \text{GLU}(\mathbf{D}^{(L+1)}) + \mathbf{b}_R)$.

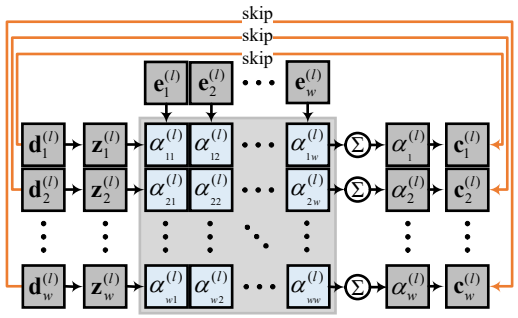


Figure 7: Attention structure.

Here, $\hat{\mathbf{X}} = \langle \hat{\mathbf{x}_1}, \hat{\mathbf{x}_2}, \ldots, \hat{\mathbf{x}_w} \rangle$ is the reconstruction of time series $\mathbf{X}$; $\mathbf{W}_R$ and $\mathbf{b}_R$ are the weight matrix and bias vector of the reconstruction layer, respectively; function $f_R(\cdot)$ is a non-linear activation function. The reconstruction errors between the elements of $\hat{\mathbf{X}}$ and $\mathbf{X}$ are used as outlier scores.

3.2 Diversity-Driven Ensembles

We use the proposed CAEs as basic models in the ensemble called `CAE-Ensemble`. Ensembles are generally able to improve overall accuracy by combining the outputs from individual basic models [7]. A naive approach is to first train multiple basic models. Then, each basic model is used to reconstruct embedded time series $\mathbf{X}$. The average over the reconstructed time series from all basic models is used as the final reconstructed time series for the ensemble, as defined in Equation 8.

$$F(\mathbf{X}) = \frac{1}{M} \sum_{m=1}^{M} f_m(\mathbf{X}), \tag{8}$$

where M is the total number of basic models, $f_m(\cdot)$ refers to the m-th basic model, and $F(\cdot)$ is the output of the ensemble.

This naive approach has two limitations: (1) If the basic models are similar, the accuracy of the ensemble is similar to those of the basic models, meaning that the ensemble is not substantially better than each basic model. (2) The computational cost of the ensemble is often M times that of a single basic model. When M is large, the ensemble can be very expensive to train.

The first limitation has been addressed partially by using basic models with different structures, e.g., by randomly modifying the connections among computational units [7, 27]. However, basic models with different structures may not produce diverse outputs.

We introduce a diversity-driven ensemble to address the above two limitations, as depicted in Figure 8. Rather than training different basic models independently [7, 27], we generate the basic models one by one. When training a basic model, we design an objective function that considers not only the accuracy of the model but also its difference compared to the previous basic models, which ensures diversity among basic models, thus addressing the first limitation. In addition, when training a basic model, we transfer a portion of the parameters from the previous basic model to the model instead of training the model from scratch. This helps significantly reduce the training time, thus addressing the second limitation.

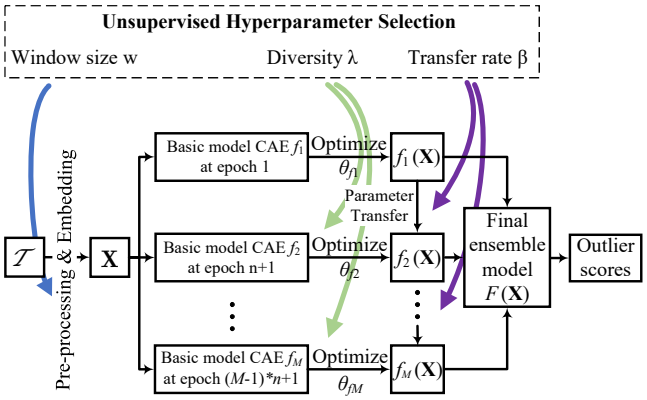


Figure 8: `CAE-Ensemble` overview.

3.2.1 Basic Model Generation. Inspired by Born-again Neural Networks [13] (`BANN`) and AdaBoost Negative Correlation [47], we iteratively generate basic models in the model training epochs. For example, we may generate a basic model per n training epochs. If $n = 10$, when training an ensemble using 200 epochs, we can then generate 20 basic models.

In the first epoch, we create the first basic model $f_1(\cdot)$, and we start training this basic model using embedded time series $\mathbf{X}$. We then add the first basic model to the ensemble. After n epochs, the first basic model is partially trained. We continue to generate the second basic model $f_2(\cdot)$ by transferring a portion of the parameters learned in the first basic model to $f_2(\cdot)$. Then, we train the second basic model. The training and basic model generation processes continue until the last training epoch.

More specifically, given a basic model $f_{m-1}(\cdot)$ with $\theta_{f_{m-1}}$ trained parameters obtained in the previous epochs, the newly generated basic model $f_m(\cdot)$ receives a randomly selected fraction β of its parameters from $f_{m-1}(\cdot)$. Then, the remaining fraction $1-\beta$ of the parameters must be trained in subsequent epochs. This approach enables us to train a large number of ensemble components with low training time. Figure 9 shows the process of generating new basic models with parameter transfer.

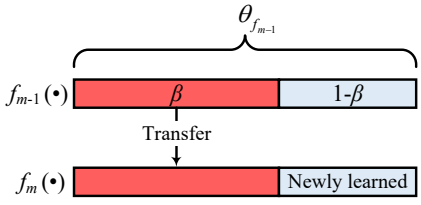


Figure 9: Basic model generation with parameter transfer.

Note that the proposed ensemble differs from a Snapshot Ensemble [24], where the trained parameters are transferred completely among the basic models.

3.2.2 Diversity Metric. Based on existing studies [2], an ensemble benefits from consisting of diverse basic models. The more diverse the basic models in an ensemble are, the more accurate the ensemble

often achieves. To quantify the diversity explicitly, we define a diversity metric $DIV_{f_m,f_n}(\cdot)$ to measure the dissimilarity between two basic models $f_m(\cdot)$ and $f_n(\cdot)$.

$$DIV_{f_m,f_n}(\mathbf{X}) = ||f_m(\mathbf{X}) - f_n(\mathbf{X})||_2, \tag{9}$$

Here, a larger $DIV_{f_m,f_n}(\mathbf{X})$ value indicates a larger difference between the outputs of basic models $f_m(\cdot)$ and $f_n(\cdot)$.

The design of diversity metric $DIV_{f_m,f_n}(\cdot)$ is inspired by the supervised diversity metric [55]. However, the supervised diversity metric computes the softmax outputs of two basic models and compares the outputs with ground-truth labels to quantify the diversity. In our unsupervised setting, we do not have ground truth labels. Instead, we aim for different basic models with different reconstructions, i.e., basic models with diverse outputs.

Next, we extend the diversity metric $DIV_F(\cdot)$ to measure the diversity of an ensemble model $F(\cdot)$ as follows.

$$DIV_F(\mathbf{X}) = \frac{2}{M(M-1)} \sum_{m=1}^{M} \sum_{n=m+1}^{M} DIV_{f_m,f_n}(\mathbf{X}), \tag{10}$$

where a large $DIV_F(\cdot)$ value indicates that ensemble model $F(\cdot)$ is more diverse.

3.2.3 Diversity-Driven Objective Function. Based on the proposed diversity metric, we define the objective function for training each basic model in the ensemble in two parts.

First, a basic model $f_m(\cdot)$ should reconstruct the time series accurately, which is the same as for all autoencoder models. Thus, the objective function $\mathcal{J}_{f_m}$ measures the difference between the input embedded time series vectors and the reconstructed vectors.

$$\mathcal{J}_{f_m} = ||\mathbf{X} - \hat{\mathbf{X}}||_2^2 = ||\mathbf{X} - f_m(\mathbf{X})||_2^2 \tag{11}$$

Second, the basic model should increase the diversity of the current ensemble. Thus, the objective function $\mathcal{K}_{f_m}$ utilizing Equation 9, measures the diversity between the basic model $f_m(\mathbf{X})$ and the current ensemble $F(\mathbf{X})$.

$$\mathcal{K}_{f_m} = ||f_m(\mathbf{X}) - F(\mathbf{X})||_2^2 \tag{12}$$

Finally, the objective function O_{f_m} for training the basic model $f_m(\cdot)$ is defined as a combination of $\mathcal{J}_{f_m}$ and $\mathcal{K}_{f_m}$.

$$\arg\min_{\theta_{f_m}} \mathcal{L}_{f_m} = \arg\min_{\theta_{f_m}} \mathcal{J}_{f_m} - \lambda \mathcal{K}_{f_m}, \tag{13}$$

where λ is a factor that controls the importance of the diversity, and θ_{f_m} represents the learnable parameters of model $f_m(\cdot)$.

3.2.4 Ensemble Outlier Score. We aggregate the outlier scores from all basic models to obtain the final outlier score. Given M basic models that reconstruct the embedded time series $\mathbf{X} = \langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_w \rangle$, we obtain M reconstructions $\hat{\mathbf{X}}^{(m)} = \langle \hat{\mathbf{x}}_1^{(m)}, \hat{\mathbf{x}}_2^{(m)}, \ldots, \hat{\mathbf{x}}_w^{(m)} \rangle$, where $1 \le m \le M$. For each vector $\mathbf{x}_t$ in the embedded time series $\mathbf{X}$, we obtain M reconstruction errors as follows.

$$\langle \|\mathbf{x}_t - \hat{\mathbf{x}}_t^{(1)}\|_2^2, \|\mathbf{x}_t - \hat{\mathbf{x}}_t^{(2)}\|_2^2, \ldots, \|\mathbf{x}_t - \hat{\mathbf{x}}_t^{(M)}\|_2^2 \rangle \tag{14}$$

We use the median of the M errors as the final outlier score of vector $\mathbf{x}_t$ as follows.

$$OS(\mathbf{x}_t) = \text{median}\left(\|\mathbf{x}_t - \hat{\mathbf{x}}_t^{(1)}\|_2^2, \ldots, \|\mathbf{x}_t - \hat{\mathbf{x}}_t^{(M)}\|_2^2 \right) \tag{15}$$

We use median$(\cdot)$ instead of mean$(\cdot)$ because median$(\cdot)$ reduces the influence of the reconstruction errors from the basic models that overfit to the original time series [27].

3.3 Unsupervised Hyperparameter Selection

Hyperparameter selection is an important step in deep learning. In our setting, important hyperparmeters include, e.g., the window size w, the parameter transfer percentage β, and parameter λ that controls the importance of the diversity in the loss function (cf. Equation 13).

Hyperparameter selection is often conducted on a validation set that is separated from the training set. We first train a model, under a specific hyperparameter setting, using the training set, and then we use the model to obtain a quality score on the validation set. Since the training and validation sets are independent, the quality score indicates how good the model, under the specific hyperparameter setting, would perform on unseen testing data. Based on the quality scores for different hyperparameter settings, we then select the hyperparameter setting that has the highest quality score.

The above strategy works well in supervised settings where ground truth labels in the validation set can be used to compute directly the quality scores, e.g., mean square errors or F1 scores. However, in an unsupervised setting, this becomes nontrivial as no ground truth labels are available.

Most existing studies for unsupervised outlier detection only provide the hyperparameters used in experiments, with no specification about how to select them. We summarize two strategies that exist in the literature for unsupervised outlier detection, identifying their limitation and propose a new strategy.

First, as there are no ground truth outlier labels available in the validation set to guide the best hyperparameter, they randomly select hyperparameters [7, 27]. However, if we are unlucky, we may choose inadequate hyperparameters, conducting to reduced accuracy.

Second, using small amounts of testing data for validation such that the ground truth labels in the testing data enables the computation of the quality scores. However, this strategy is no longer fully unsupervised—although the training is still unsupervised with no ground truth outlier labels, the hyperparameter selection needs the ground truth outlier labels [6].

We propose to use the reconstruction errors on the validation set as quality scores. This is fully unsupervised, since computing the reconstruction errors does not require ground truth labels. We may choose the hyperparameters that give the *lowest* reconstruction errors. However, we observe in our experiments that this strategy often leads to sub-optimal outlier detection accuracy. When the reconstruction errors are too small, the model may overfit to the specifics in the training time series, including outliers, thus making it difficult to distinguish outliers from regular patterns.

Our proposal uses the hyperparameters with the *median* reconstruction error among all hyperparameter settings. Although this strategy does not guarantee that the selected hyperparameters produce a model with the best accuracy on the testing data, it often provides a good enough accuracy and often outperforms the model using the hyperparameters with the lowest reconstruction errors, as it is shown in experiments in Section 4.2.4.

We employ three important hyperparameters in the model, as shown in Figure 8: the window size w, the diversity factor λ (cf. Equation 13) and the fraction of parameter transfer between ensemble members β (cf. Figure 9). The process is detailed in Algorithm 2.

Algorithm 2: Unsupervised Hyperparameter Selection

Input : A time series $\mathcal{T}$
Output: Selected hyperparameter values

$reconst[\,] \leftarrow \emptyset$;
$\mathcal{T}_{training}, \mathcal{T}_{validation} \leftarrow Split(\mathcal{T})$;
/* Finding default hyperparameter values using random search. */
while Randomly Select (w, β, λ) from ($[w_1, w_a]$, $[\beta_1, \beta_b]$, $[\lambda_1, \lambda_c]$) **do**
  Train CAE-Ensemble($\mathcal{T}_{training}$, w, β, λ);
  $reconst[w, \beta, \lambda] \leftarrow$ CAE-Ensemble($\mathcal{T}_{validation}$);
$(w_{def}, \beta_{def}, \lambda_{def}) \leftarrow arg\ median(reconst)$;
/* Choosing the Optimal Hyperparameters. */
$w_{opt} \leftarrow arg\ median_{w \in [w_1, w_a]} ReconError(w, \beta_{def}, \lambda_{def})$;
$\beta_{opt} \leftarrow arg\ median_{\beta \in [\beta_1, \beta_b]} ReconError(w_{def}, \beta, \lambda_{def})$;
$\lambda_{opt} \leftarrow arg\ median_{\lambda \in [\lambda_1, \lambda_c]} ReconError(w_{def}, \beta_{def}, \lambda)$;
return $(w_{opt}, \beta_{opt}, \lambda_{opt})$

The dataset is first split into training and validation sets, both without ground-truth labels. This ensures that the hyperparameter selection remains unsupervised. Then we define a range for each hyperparameter, i.e., $[w_1, w_a]$ for window size w. Next, we use random search [3] to identify a specific hyperparameter combination $(w_{def}, \beta_{def}, \lambda_{def})$ with the median reconstruction error among all hyperparameter combinations considered in the random search. This triple is used as the default hyperparmeters. Here, we use random search instead of grid search that requires enumeration of all possible hyperparameter combinations, which is too costly. Finally, we identify the optimal value for each hyperparameter. To do so, we fix the other two hyperparameters to their default values and vary the hyperparameter in its range. For example, when identifying the optimal w, we fix β and λ to their default values, and vary w in range $[w_1, w_a]$. We then choose the w_{opt} that gives the median reconstruction error as the optimal value for hyperparameter w.

4 EXPERIMENTS

We conduct extensive experiments to support the design choices for CAE-Ensemble. We evaluate both accuracy the efficiency.

4.1 Experiment Setup

4.1.1 Data sources. We use five different multivariate public real-world time series datasets, comprising settings from different domains such as server metrics, health, and embedded systems monitoring. All datasets, except *ECG*, include a train and test sets, considering ground-truth labels indicating outliers only for testing. We do not use the labels when training, just to calculate the accuracy. For *ECG*, the same set is used for both steps, ignoring the labels during training. For each data set, we reserve 30% of the training set as the validation set to enable the selection of optimal hyperparameters using the proposed median strategy.

- *ECG*[1] is a two-dimensional time series related to electrocardiogram readings for seven patients. Each time series contains 3,700-5,400 observations. The outlier ratio is 4.88%.
- *SMD*[2] is a public dataset for server metrics consisting of 28 time series, where each has 38 dimensions. It contains 708,405 observations for training and 708,420 for testing. The outlier ratio is 4.16%.
- *MSL*[3] consists on telemetry data from the Mars Curiosity Rover. It comprises 36 time series with 55 dimensions, containing 58,317 observations for training and 73,729 for testing. The outlier ratio is 9.17%.
- *SMAP*[3] is a dataset from a Soil Moisture satellite consisting on 69 time series with 25 different variables distributed in ten subsets. In total, they consist on 138,004 training and 435,826 testing observations. The outlier ratio is 12.27%.
- *WADI*[4] consists on two time series with 127 dimensions, representing a water distribution system under normal operation (1,994,172 observations) and during intrusion attacks (345,604 testing readings). The outlier ratio is 5.76% and it is sampled every ten timestamps, given its extensive size.

4.1.2 Baselines. We compare the proposed CAE-Ensemble with the following unsupervised outlier detection models.

- Isolation Forest (IS) [32]: An ensemble of randomized clustering trees that isolates outliers in sparse clusters. We use 100 base estimators for the ensemble;
- Local Outlier Factor (LOF) [5]: A density clustering based method that detects outliers according to local deviations from neighbors. The number of neighbors is 20 and we use Euclidean distance;
- One-Class SVM (SVM) [42]: A one-class classification method that employs Support Vector Machines to learn the boundary normal data points. We use a radial basis function (RBF) kernel with $\nu = 0.5$ [41];
- Moving Average Smoothing (MAS): A method where the values that deviates from a moving average window are likely to be considered as outliers;
- Autoencoder Ensemble (AE-Ensemble) [7]: An ensemble that consists of feed forward autoencoders with 20% of the connections randomly removed (cf. Table 1 in Section 2);
- Recurrent Autoencoder (RAE) [35]: A recurrent autoencoder using LSTM units to reconstruct time series via a sequence-to-sequence architecture (cf. Table 1 in Section 2);
- Convolutional Autoencoder (CAE): The proposed convolutional sequence-to-sequence autoencoder without using an ensemble (cf. Section 3.1);
- Correlation Matrices Recurrent Autoencoder (MSCRED) [54]: A state-of-the-art method for multivariate time series outlier detection that uses an autoencoder to reconstruct correlation matrices instead of using the time series directly. Matrices have length 16 with 5 steps in-between;
- Variational Recurrent Autoencoder (RNNVAE) [43]: The model establishes a stochastic latent component in the autoencoder

[1] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
[2] https://github.com/NetManAIOps/OmniAnomaly/
[3] https://github.com/khundman/telemanom
[4] https://itrust.sutd.edu.sg/itrust-labs_datasets/

for learning a distribution to improve the reconstruction output. The hidden and stochastic spaces are 64 with a regularization of 0.0001;
- Temporal Variational Autoencoder (OMNIANOMALY) [44]: The method extends the previous variational modeling with an additional component to capture temporal dependencies in the context of stochastic variables. The hidden space is 32 with 16 stochastic variables and a regularization of 0.0001;
- Recurrent Autoencoder Ensembles (RAE-Ensemble) [27]: A state-of-the-art recurrent autoencoder ensemble (cf. Table 1 in Section 2), where 20% of the skip connections are randomly dropped.

4.1.3 Evaluation Metrics. We consider two categories of metrics.
All thresholds: We consider a setting where prior knowledge on selecting a specific threshold is not provided. In this case, we use two metrics that consider all possible thresholds [7, 27]—Area Under the Curve of Precision-Recall (*PR*) and Area Under the Curve of Receiver Operating Characteristic (*ROC*) [40]. Both metrics are defined in terms of true positives (*TP*) (i.e., outliers predicted as outliers), true negatives (*TN*) (i.e., inliers predicted as inliers), false positives (*FP*) (i.e., inliers predicted as outliers), and false negatives (*FN*) (i.e., outliers predicted as inliers). The difference between *PR* and *ROC* is that *PR* disregards *TN*. The higher the *PR* and *ROC* are, the more accurate a method is.
Specific thresholds: We consider a setting where a specific threshold is chosen and we measure *Precision*, *Recall*, and *F1* [40] with the ground-truth outlier labels using the specific threshold. Higher values are desired. Here, we consider two settings. First, we select a threshold, among all possible thresholds, that gives the "best" *F1*, i.e., the best possible threshold [44, 49]. We use this threshold to report the best *F1*, along with the corresponding *Precision* and *Recall*. Second, we consider a specific threshold that is decided by the prior knowledge on the outlier ratio. In particular, if we know that a data set has K% of outliers, we choose a threshold such that there are K% of data whose ouliter scores are higher than the threshold.

4.1.4 Hyperparameters. The transfer β and diversity λ parameters (see Equation 13) are set according to the median strategy presented in Section 3.3, evaluating the cases $\beta = i/10, i \in [1, 9]$ and $\lambda = 2^j, j \in [0, 6]$ for each data set. Then, each time series is divided into windows of size w with $w - 1$ overlapped observations with respect to the previous window. For each window, we consider the reconstruction of the last observation in the window when calculating the outlier scores, except in the case of the first window, where all reconstructions are considered. The w is also chosen according to the median strategy, considering $w = 2^k, k \in [2, 8]$. The selected hyperparameters for different data sets are shown in Table 2. The details of hyperparameter selection are offered in Sections 4.2.4 and 4.2.5.

Table 2: Selected hyperparameters, median strategy.

	ECG	*MSL*	*SMAP*	*SMD*	*WADI*
β	0.5	0.7	0.9	0.2	0.5
λ	2	16	2	32	1
w	16	16	16	32	32

4.1.5 Implementation Details. We set the number of convolution layers in CAE to 10 for both the encoder and the decoder, with the kernel size for the convolution operator in all layers set to 3. The dimension of embedded vector D' is set to 256, and the batch sizes to 64. By default we use 8 basic models, with a new basic model for every 50 epochs. This setting ensures that a newly generated basic model is learned from previously well-trained basic models. We also vary the number of basic models in Section 4.2.6. We use Adam [29], a Stochastic Gradient Descent variant, as the optimizer. The learning rate is set to 0.001.

The experiments are run on a Ubuntu 18.04 server with a 10-cores CPU Intel Xeon W-2155, 128GB RAM, and two GPU NVIDIA TITAN RTXs. The code was written in Python 3.9.1 using PyTorch 1.7.1. The source code is publicly available at https://github.com/d-gcc/CAE-Ensemble.

4.2 Experiment Results

4.2.1 Accuracy. We report the accuracy results for the five datasets, respectively, as well as the average accuracy over all datasets. We report all-threshold metrics *PR* and *ROC* and threshold-dependent metrics *Precision*, *Recall*, and *F1*, when using the threshold that gives the best *F1*. We use bold and underline to highlight the highest and second highest values.

The results of *ECG* are shown in the first section of Table 3. The neural network based methods outperform the non-neural network based methods in almost all metrics. The relatively high *Recall* for ISF comes with a price with low *Precision*, which means that many instances that are not outliers are mistakenly considered as outliers. On average, *CAE-Ensemble* achieves the best result for *Precision*, *F1*, and *PR* with a competitive *ROC* case, showing a very good performance overall in comparison to the baselines.

The results for *SMD* are shown in the middle of Table 3. where *CAE* and *CAE-Ensemble* achieve higher accuracy on most of the metrics, in particular, *Precision*, *F1*, and *PR*, demonstrating the model functionality in server-related data sets. The high *Recall* in *MSCRED* reflects the case detailed before, i.e., with the price of low *Precision*.

For *MSL*, in the last part of Table 3, and *SMAP* in the first part of Table 4, our proposals show a great performance on the all-threshold metrics *PR* and *ROC* and also on the threshold-dependent metric *F1* that represents a balance between *Precision* and *Recall*, compared to all baselines. Then, for *WADI*, in the middle section of Table 4, the proposed models also present a robust performance, suggesting that the proposals also work in an application domain of water distribution.

The last part of Table 4 summarizes the overall performance for all the time series. The results show that our method outperforms all the other baselines w.r.t. *Precision*, *F1*, *PR*, and *ROC*, suggesting that CAE-Ensemble is a very competitive solution in terms of quality. For the *Recall* metric, the better performance shown by MSCRED and ISF come with very low *Precision*, making them not competitive in real application scenarios. This occurs when such methods are overly pessimistic and identify many observations as outliers, although some observations are not outliers. This produces more true positives at the cost of obtanining many false positives. This gives high *Recall* but low *Precision*. In addition, CAE-Ensemble performs not so well on *WADI*, where it has a very low *Recall* and thus low

Table 3: *ECG*, *SMD*, and *MSL* accuracy results.

Data set	*ECG*					*SMD*					*MSL*				
Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*	*Precision*	*Recall*	*F1*	*PR*	*ROC*	*Precision*	*Recall*	*F1*	*PR*	*ROC*
ISF	0.0543	**0.7199**	0.0999	0.0501	0.5062	0.0880	0.4571	0.1079	0.0591	0.5066	0.1553	0.6512	0.1895	0.1085	0.5036
LOF	0.0539	0.6539	0.0962	0.0500	0.4912	0.2494	0.2571	0.1764	0.1203	0.5695	0.2463	0.5316	0.2358	0.1431	0.5268
MAS	0.0670	0.6276	0.1159	0.0578	0.5342	0.4720	0.4099	0.3716	0.3253	**0.7520**	**0.2959**	0.5537	0.2525	0.1595	0.5469
OCSVM	0.0825	0.4987	0.1309	0.0588	0.5342	0.3414	0.2944	0.2626	0.1927	0.5783	0.2847	0.5149	0.2616	0.1581	0.5629
MSCRED	0.1789	0.6651	0.2303	0.1055	0.5166	0.0631	**0.7719**	0.1100	0.0395	0.5000	0.1243	**0.7747**	0.1874	0.1166	0.5072
OMNIANOMALY	0.2220	0.4938	0.2042	0.1409	0.5584	0.2432	0.3328	0.2110	0.1503	0.6148	0.1936	0.6297	0.2414	0.1609	0.5429
RNNVAE	0.1768	0.4222	0.1439	0.0895	0.5500	0.4334	0.3194	0.3045	0.2406	0.6917	0.1641	0.5639	0.2125	0.1378	0.5335
AE-Ensemble	0.1583	0.5398	0.1907	0.1302	**0.5952**	0.3713	0.3709	0.2832	0.2349	0.6823	0.1775	0.6936	0.2424	0.1404	0.5360
RAE	0.1297	0.5394	0.1669	0.0936	0.5922	0.4466	0.3037	0.3078	0.2424	0.6836	0.2069	0.6091	0.2423	0.1503	0.5575
RAE-Ensemble	0.2003	0.5838	0.1864	0.1176	0.5372	0.4684	0.3318	0.3332	0.2639	0.6998	0.2085	0.5633	0.2495	0.1572	0.5714
CAE	0.1919	0.4574	0.1954	0.1297	0.5633	0.4625	0.3804	**0.3895**	**0.3299**	0.7416	0.2223	0.5273	0.2649	**0.1641**	0.5843
CAE-Ensemble	**0.2522**	0.4924	**0.2521**	**0.1887**	0.5715	**0.4924**	0.3739	0.3770	0.3246	0.7375	0.2501	0.5343	**0.2713**	0.1633	**0.5963**

Table 4: *SMAP*, *WADI*, and *Overall* accuracy results.

Data set	*SMAP*					*WADI*					*Overall*				
Model	*Precision*	*Recall*	*F1*	*PR*	*ROC*	*Precision*	*Recall*	*F1*	*PR*	*ROC*	*Precision*	*Recall*	*F1*	*PR*	*ROC*
ISF	0.1396	0.5298	0.1986	0.1300	0.4979	0.0667	0.4765	0.1170	0.0610	0.5248	0.1008	0.5669	0.1426	0.0818	0.5078
LOF	0.2261	0.5178	0.2027	0.1289	0.5005	0.0736	0.3155	0.1193	0.0702	0.5284	0.1698	0.4552	0.1661	0.1025	0.5233
MAS	0.2819	0.5174	0.2542	0.1655	0.5233	0.2586	0.1555	0.1942	0.1490	0.5788	0.2751	0.4528	0.2377	0.1714	0.5870
OCSVM	0.2561	0.5722	0.2302	0.1461	0.4924	0.0980	0.2955	0.1472	0.1192	0.5754	0.2125	0.4351	0.2065	0.1350	0.5487
MSCRED	0.1266	**0.8199**	0.1914	0.1028	0.4403	0.1382	**0.8590**	0.2377	0.0993	0.6730	0.1262	**0.7781**	0.1913	0.0927	0.5274
OMNIANOMALY	0.2307	0.6222	0.2681	0.1556	0.5402	0.2996	0.3976	**0.3404**	0.1723	**0.7261**	0.2378	0.4952	0.2530	0.1560	0.5965
RNNVAE	0.1622	0.5646	0.1971	0.1192	0.5119	0.2881	0.3147	0.2867	0.1734	0.5739	0.2449	0.4370	0.2289	0.1521	0.5722
AE-Ensemble	0.3134	0.5895	0.2939	0.1780	0.5496	0.1619	0.2398	0.1928	0.0911	0.5102	0.2404	0.4727	0.2379	0.1498	0.6078
RAE	0.2071	0.6316	0.2381	0.1476	0.5390	0.2118	0.2799	0.2342	0.1150	0.6667	0.2365	0.4867	0.2406	0.1549	0.5747
RAE-Ensemble	0.2603	0.6604	0.2529	0.1628	0.5716	0.2999	0.2535	0.2707	0.1580	0.6516	0.2875	0.4786	0.2585	0.1719	0.6063
CAE	0.3175	0.5912	0.3170	0.2135	0.5892	0.2350	0.3019	0.2004	0.1243	0.5994	0.2858	0.4516	0.2735	0.1923	0.6156
CAE-Ensemble	**0.3387**	0.6187	**0.3327**	**0.2223**	**0.6080**	**0.5006**	0.1995	0.2853	**0.1911**	0.6023	**0.3668**	0.4438	**0.3037**	**0.2180**	**0.6231**

F1 and *ROC*. This is due to how the ground-truth is provided in the data set—the ground-truth marks all observations in long intervals as outliers, whereas only a few observations in the middle of the long intervals are real outliers. Although CAE-Ensemble is often able to detect the real outliers, it still has a low recall.

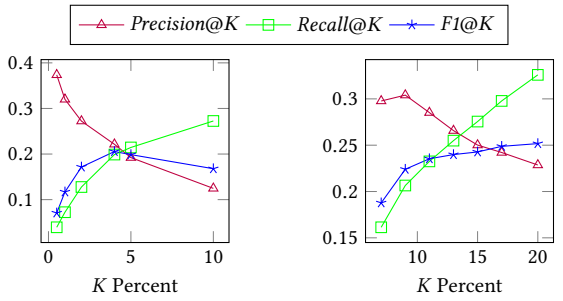
Due to the space limitation, in the following experiments, we only report results on two data sets, *ECG* and *SMAP*, as other data sets show similar results.

4.2.2 Outlier Score Threshold Sensitivity. To characterize the metrics related to specific thresholds, in Figure 10, we show their evolution as we select the top K percentage of the largest outlier scores as the threshold and consider the observations which have larger outlier scores as outliers.

For *ECG*, Figure 10(a) shows convergence at $K = 5$ percent, which is close to the actual outlier ratio (i.e., 4.88% cf. Section 4.1.1). *SMAP* includes multiple subsets whose outlier ratios differ significantly, ranging from 0.8% to 21.9%. Figure 10(b) shows a representative subset from *SMAP* with an outlier ratio of 12%, which is close to the average outlier ratio of *SMAP* (cf. Section 4.1.1).

Both figures suggest that when the outlier ratio is available, choosing it as the K percentage is a good choice.

4.2.3 Ablation Study. To evaluate our design choices, we study the effect of each module of CAE-Ensemble by removing model components. Specifically, we (1) remove the attention module (*No attention*); (2) remove the parameter transfer learning and without using the diversity metric on the objective function in the ensemble such that all basic models are trained independently (*No diversity*); (3) disregard the ensemble, i.e., using a single basic model CAE (*No ensemble*), and (4) remove the re-scaling in the pre-processing step and use the raw time series directly (*No re-scaling*).



(a) *ECG* **(b) *SMAP***

Figure 10: Outlier score threshold sensitivity with top K% largest outlier scores.

Table 5: Ablation study, *ECG* and *SMAP*.

		Precision	*Recall*	*F1*	*PR*	*ROC*
ECG	`No attention`	0.1440	0.4809	0.1840	0.1037	0.5606
	`No diversity`	0.1683	0.4714	0.1819	0.1244	**0.5939**
	`No ensemble`	0.1919	0.4574	0.1954	0.1297	0.5633
	`No re-scaling`	0.1806	0.4819	0.1741	0.1130	0.5379
	`CAE-Ensemble`	**0.2522**	**0.4924**	**0.2521**	**0.1887**	0.5715
SMAP	`No attention`	0.3290	0.5763	0.3049	0.1957	0.5605
	`No diversity`	0.3241	0.5841	0.3210	0.2186	0.5832
	`No ensemble`	0.3175	0.5912	0.3170	0.2135	0.5892
	`No re-scaling`	0.3252	0.5689	0.2872	0.1938	0.5666
	`CAE-Ensemble`	**0.3387**	**0.6187**	**0.3327**	**0.2223**	**0.6080**

Table 5 shows that the model with all components achieves the best results on almost all metrics. The only exception is the *ROC* metric for *ECG* when no diversity is considered. The diversity may introduce a few noisy false positives at some thresholds, which adversely affects the average performance.

We evaluate the diversity metric using Equation 10 for the two ensemble models in Table 6. A higher value indicates an ensemble has more diverse basic models, which is more desirable [2]. As expected, the full `CAE-Ensemble` model that considers diversity in the loss function achieves higher diversity. In contrast, for `No Diversity` ensemble, the basic models are trained independently using different random seeds to initialize their model parameters, which still results in different, but less diverse, basic models.

Table 6: Quantifying the diversity.

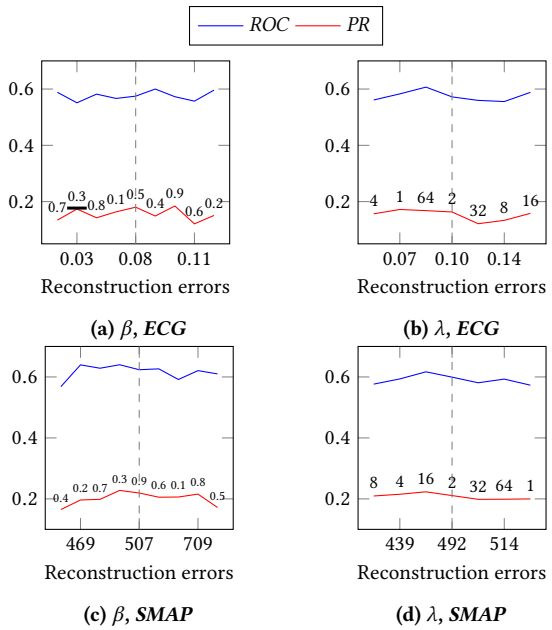
	ECG	*SMAP*
`No Diversity`	57.0118	16.3409
`CAE-Ensemble`	**94.7425**	**52.0796**

4.2.4 Hyperparameters Selection for β and λ. Figure 11 shows the β and λ parameter choices for the *ECG* and *SMAP* datasets according to the median reconstruction error strategy. Here, we consider metrics *PR* and *ROC* to evaluate the accuracy of the outlier detection. Note that computing *PR* and *ROC* requires ground-truth outlier labels, which are unavailable during validation.

The results in Figure 11 are ordered by the associated reconstruction errors. Then, the median reconstruction error is marked by the dashed vertical line in the middle of each figure. The numbers along the *PR* curve show specific values for the considered hyperparameter. For example, in Figure 11(a), when $\beta = 0.3$ (underlined), its corresponding reconstruction error is 0.03, the corresponding *PR* is a bit more than 0.17, and the corresponding *ROC* is below 0.57.

Figure 11 shows that the median strategy achieves good enough *PR* and *ROC*, although the selected β and λ are not the optimal values that give the highest *PR* or *ROC*. Compared to the strategy that selects the values with the smallest reconstruction errors, the median strategy often achieves higher *PR* and *ROC*.

More specifically, the results for β in Figures 11(a) and 11(c) are relatively unstable, showing sudden changes mainly for *ECG*. However, the median case is balanced between the best and worst cases, suggesting that it is a robust strategy. The λ parameter, in Figures 11(b) and 11(d), seems to have a trend, where the better results are achieved just before the median validation error. That observation can be useful for defining enhanced criteria using the validation errors for selecting λ. Again, the median strategy is a robust and good enough strategy.

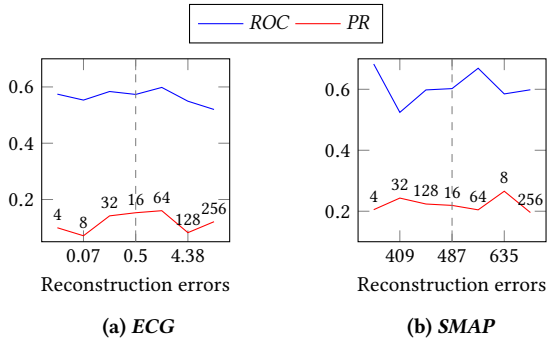


(a) β, *ECG* **(b) λ, *ECG***

(c) β, *SMAP* **(d) λ, *SMAP***

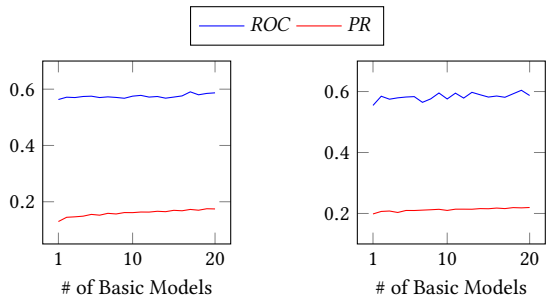
Figure 11: Hyperparameters selection—the numbers over the *PR* curve are candidate values for β and λ.

4.2.5 Hyperparameters Selection for Window Size w. We examine the effect of different window sizes. The results are shown in Figure 12, where we notice that the window size w, using the median strategy is not the best, since other configurations exist with higher PR and ROC values. Even so, the median strategy is reliable since the results are balanced among all the evaluated cases.



(a) *ECG* **(b) *SMAP***

Figure 12: Hyperparameter selection—the numbers over the *PR* curve are candidate values for window size w.



(a) *ECG* **(b) *SMAP***

Figure 13: Effect of the number of basic models.

4.2.6 Effect of the number of Basic Models. Figure 13 shows the performance for *PR*, and *ROC* as the number of basic models in the ensemble grows during training. For both datasets, there is an increasing performance as we include more basic models. For *ECG*, the results for *PR* are steadier and clearer as the number of basic models is greater. Then, for *ROC*, the metric shows a positive trend, but it is more unstable, as there are sudden changes between cases. For *SMAP*, the results are similar as the number of basic models is growing, with a clear and stable trend in *PR* with some fluctuations in *ROC*. This suggests that more basic models often help improve outlier detection accuracy, while training more basic models also needs more time.

4.2.7 Run Time. In Table 7, we report the training time of RAE, CAE, RAE-Ensemble, and CAE-Ensemble on five datasets, with eight basic models for each case. The ensemble models take longer time than their corresponding basic models, which is expected. However, CAE-Ensemble is much faster than RAE-Ensemble. For example, in cases such as *SMD*, the training for CAE-Ensemble finishes in a quarter of the time required for RAE-Ensemble. The results suggest that CAE-Ensemble is very efficient in comparison to the recurrent autoencoder. Also, the runtime ratio of RAE-Ensemble and RAE is on average 7.82, while the ratio between the CAE-Ensemble and CAE is 5.91, indicating that the parameter transfer in the CAE-Ensemble is effective reducing the training time. Thus, the convolution autoencoder and the parameter transfer are effective and succeed in reducing the running time in comparison to the recurrent model.

In the online phase, it takes around 50 microseconds to process a window of observations, making CAE and CAE-Ensemble possible to support real time outlier detection on streaming data.

Table 7: Training Time Comparison (in minutes).

Model	*ECG*	*MSL*	*SMAP*	*SMD*	*WADI*
RAE	7.84	16.63	32.19	246.43	72.32
RAE-Ensemble	59.66	129.99	254.83	1959.13	566.89
Ratio	7.60	7.82	7.92	7.95	7.84
CAE	4.16	7.65	20.36	74.34	22.37
CAE-Ensemble	24.05	45.45	122.13	452.06	129.58
Ratio	5.78	5.94	6.00	6.08	5.79

5 RELATED WORK

Outlier Detection in Time Series. Several traditional methods exist [16] that rely mainly on parametric methodologies to detect outliers and further to repair outliers [53]. Thus, they generally achieve good results given proper refinement and tuning according to specific guidelines [30, 56]. Some studies [45, 52] focus on improving the efficiency of distance based outlier detection methods to better support streaming data settings. CAE-Ensemble is also able to support streaming settings since the training is performed offline. Other studies exist that target semi-supervised [19] and supervised settings [25], which require ground-truth labels. In contrast, we study unsupervised outlier detection. Deep learning based methods are summarized in Table 1.

Sequence-to-sequence (seq2seq) Processing. Seq2seq modeling is introduced to model sequences, which is intrinsically sequential. Recurrent seq2seq architectures [10, 22, 23, 35, 51] require expensive recursions. A convolutional architecture [14] makes it possible to take advantage of parallel execution. Our framework differs from other seq2seq models by focusing on the unsupervised time series outlier detection problem using the convolution process. To the best of our knowledge, our study is the first attempt to apply seq2seq convolution in unsupervised time series outlier detection.

Ensemble Learning. Ensembles are used in a broad range of applications, generally covering supervised settings such as classification and regression [36]. Although several ensemble methods exist, such as bagging [4], boosting [12], and stacking [48], these methods all depend on labeled data to aggregate basic models. As a result, it is not trivial to apply these methods for unsupervised outlier detection. Existing unsupervised ensemble studies [7, 27] form an ensemble by creating a large quantity of basic models with random architectures. However, these methods is sub-optimal because the obtained basic models can be strongly alike thus limiting the diversity. A recent study [55] proposes EDGE, a solution to this problem using a diversity measure among basic models, which inspires our work. However, EDGE targets a supervised setting that differs substantially from our setting of unsupervised outlier detection. In addition, EDGE does not work for sequential data.

6 CONCLUSION AND FUTURE WORK

We propose a diversity-driven ensemble model built on convolutional sequence-to-sequence autoencoders for unsupervised outlier detection in time series data, along with an unsupervised hyperparameter selection method. An extensive empirical study offers evidence that the proposal is capable of improving both accuracy and efficiency compared to existing autoencoder based methods.

In future work, it is of interest to enable unsupervised time series cleaning by repairing detected outliers. It is also of interest to study more advanced unsupervised hyperparameter selection, e.g., exploring the relationships between the outlier ratio and the diversity metric.

ACKNOWLEDGMENTS

This work was partially supported by Independent Research Fund Denmark under agreements 8022-00246B and 8048-00038B, the VILLUM FONDEN under agreement 34328, Huawei Cloud Database Innovation Lab, and the Innovation Fund Denmark centre, DIREC.

REFERENCES

[1] Charu C. Aggarwal. 2013. *Outlier Analysis.*
[2] Charu C. Aggarwal and Saket Sathe. 2017. *Outlier Ensembles - An Introduction.*
[3] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305.
[4] Leo Breiman. 1996. Bagging Predictors. *Mach. Learn.* 24, 2 (1996), 123–140.
[5] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-Based Local Outliers. In *SIGMOD*. 93–104.
[6] Hakan Cevikalp, Burak Benligiray, and Omer Nezih Gerek. 2020. Semi-supervised Robust Deep Neural Networks for Multi-label Image Classification. *Pattern Recognition* 100 (2020), 107164.
[7] Jinghui Chen, Saket Sathe, Charu C. Aggarwal, and Deepak S. Turaga. 2017. Outlier Detection with Autoencoder Ensembles. In *SDM*. 90–98.
[8] Razvan-Gabriel Cirstea, Darius-Valer Micu, Gabriel-Marcel Muresan, Chenjuan Guo, and Bin Yang. 2018. Correlated Time Series Forecasting using Multi-Task Deep Neural Networks. In *CIKM*. 1527–1530.
[9] Razvan-Gabriel Cirstea, Tung Kieu, Chenjuan Guo, Bin Yang, and Sinno Jialin Pan. 2021. EnhanceNet: Plugin Neural Networks for Enhancing Correlated Time Series Forecasting.. In *ICDE*. 1739–1750.
[10] Razvan-Gabriel Cirstea, Bin Yang, and Chenjuan Guo. 2019. Graph Attention Recurrent Neural Networks for Correlated Time Series Forecasting.. In *MileTS19@KDD*.
[11] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language Modeling with Gated Convolutional Networks. In *ICML*. 933–941.
[12] Yoav Freund and Robert E. Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. Syst. Sci.* 55, 1 (1997), 119–139.
[13] Tommaso Furlanello, Zachary Chase Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born-Again Neural Networks. In *ICML*. 1602–1611.
[14] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *ICML*. 1243–1252.
[15] Chenjuan Guo, Bin Yang, Jilin Hu, Christian S. Jensen, and Lu Chen. 2020. Context-aware, preference-based vehicle routing. *VLDB J.* 29, 5 (2020), 1149–1170.
[16] Manish Gupta, Jing Gao, Charu C. Aggarwal, and Jiawei Han. 2014. Outlier Detection for Temporal Data: A Survey. *IEEE Trans. Knowl. Data Eng.* 26, 9 (2014), 2250–2267.
[17] Simon Hawkins, Hongxing He, Graham J. Williams, and Rohan A. Baxter. 2002. Outlier Detection Using Replicator Neural Networks. In *DAWAK*. 170–180.
[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.
[19] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. 2019. Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty. In *NIPS*. 15637–15648.
[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.
[21] Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. *VLDB J.* 27, 2 (2018), 179–200.
[22] Jilin Hu, Bin Yang, Chenjuan Guo, Christian S. Jensen, and Hui Xiong. 2020. Stochastic Origin-Destination Matrix Forecasting Using Dual-Stage Graph Convolutional, Recurrent Neural Networks. In *ICDE*. 1417–1428.
[23] Renjun Hu, Charu C. Aggarwal, Shuai Ma, and Jinpeng Huai. 2016. An embedding approach to anomaly detection. In *ICDE*. 385–396.
[24] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. Snapshot Ensembles: Train 1, Get M for Free. In *ICLR*. pp. 14.
[25] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Söderström. 2018. Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. In *SIGKDD*. 387–395.
[26] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Distinguishing Trajectories from Different Drivers using Incompletely Labeled Trajectories. In *CIKM*. 863–872.
[27] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2019. Outlier Detection for Time Series with Recurrent Autoencoder Ensembles. In *IJCAI*. 2725–2732.
[28] Tung Kieu, Bin Yang, and Christian S. Jensen. 2018. Outlier Detection for Multi-dimensional Time Series Using Deep Neural Networks. In *MDM*. 125–134.
[29] Diederik P. Kingma and Jimmy Ba. 2015. Adam: a Method for Stochastic Optimization. In *ICLR*. pp. 15.
[30] Kim-Hung Le and Paolo Papotti. 2020. User-driven Error Detection for Time Series with Events. In *ICDE*. 745–757.
[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[32] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *ICDM*. 413–422.
[33] Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. 2018. Finding Top-k Optimal Sequenced Routes. In *ICDE*. 569–580.
[34] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *EMNLP*. 1412–1421.
[35] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam M. Shroff. 2016. LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. In *ICML Anomaly Detection Workshop*. 5.
[36] Oleg Okun, Giorgio Valentini, and Matteo Ré (Eds.). 2011. *Ensembles in Machine Learning Applications*. Studies in Computational Intelligence, Vol. 373.
[37] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Anytime Stochastic Routing with Hybrid Learning. *Proc. VLDB Endow.* 13, 9 (2020), 1555–1567.
[38] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Fast stochastic routing under time-varying uncertainty. *VLDB J.* 29, 4 (2020), 819–839.
[39] Mayu Sakurada and Takehisa Yairi. 2014. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. In *MLSDA*. 4–11.
[40] Claude Sammut and Geoffrey I. Webb (Eds.). 2017. *Encyclopedia of Machine Learning and Data Mining*.
[41] Bernhard Schölkopf, Alexander J. Smola, Robert C. Williamson, and Peter L. Bartlett. 2000. New Support Vector Algorithms. *Neural Comput.* 12, 5 (2000), 1207–1245.
[42] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, and John C. Platt. 1999. Support Vector Method for Novelty Detection. In *NIPS*. 582–588.
[43] Maximilian Soelch, Justin Bayer, Marvin Ludersdorfer, and Patrick van der Smagt. 2016. Variational Inference for On-line Anomaly Detection in High-Dimensional Time Series. *CoRR* abs/1602.07109 (2016), 4.
[44] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network. In *SIGKDD*. 2828–2837.
[45] Luan Tran, Minyoung Mun, and Cyrus Shahabi. 2020. Real-Time Distance-Based Outlier Detection in Data Streams. *Proc. VLDB Endow.* 14, 2 (2020), 141–153.
[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*. 5998–6008.
[47] Shuo Wang, Huanhuan Chen, and Xin Yao. 2010. Negative correlation learning for classification ensembles. In *IJCNN*. 1–8.
[48] David H. Wolpert. 1992. Stacked generalization. *Neural Networks* 5, 2 (1992), 241–259.
[49] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *WWW*. 187–196.
[50] Sean Bin Yang, Chenjuan Guo, Jilin Hu, Jian Tang, and Bin Yang. 2021. Unsupervised Path Representation Learning with Curriculum Negative Sampling. In *IJCAI*. 3286–3292.
[51] Sean Bin Yang, Chenjuan Guo, and Bin Yang. 2020. Context-Aware Path Ranking in Road Networks. *IEEE Trans. Knowl. Data Eng.* (2020).
[52] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. 2019. NETS: Extremely Fast Outlier Detection from a Data Stream via Set-Based Processing. *Proc. VLDB Endow.* 12, 11 (2019), 1303–1315.
[53] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S. Yu. 2017. Time Series Data Cleaning: From Anomaly Detection to Anomaly Repairing. *Proc. VLDB Endow.* 10, 10 (2017), 1046–1057.
[54] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V. Chawla. 2019. A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data. In *AAAI*. 1409–1416.
[55] Wentao Zhang, Jiawei Jiang, Yingxia Shao, and Bin Cui. 2020. Efficient Diversity-Driven Ensemble for Deep Neural Networks. In *ICDE*. 73–84.
[56] Xuyun Zhang, Wan-Chun Dou, Qiang He, Rui Zhou, Christopher Leckie, Kotagiri Ramamohanarao, and Zoran A. Salcic. 2017. LSHiForest: A Generic Framework for Fast Tree Isolation Based Ensemble Anomaly Analysis. In *ICDE*. 983–994.