Information Sciences 588 (2022) 159–173

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Denoising temporal convolutional recurrent autoencoders for time series classification

Zhong Zheng[a], Zijun Zhang[a,*], Long Wang[b], Xiong Luo[b]

[a] School of Data Science, City University of Hong Kong, Hong Kong Special Administrative Region, China
[b] School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China

A R T I C L E I N F O

Article history:
Received 14 March 2020
Received in revised form 4 March 2021
Accepted 13 December 2021
Available online 17 December 2021

Keywords:
Neural networks
Denoising autoencoders
Time series classification
Temporal convolutional networks
Gated Recurrent Unit

A B S T R A C T

In this paper, a denoising temporal convolutional recurrent autoencoder (DTCRAE) is proposed to improve the performance of the temporal convolutional network (TCN) on time series classification (TSC). The DTCRAE consists of a TCN encoder and a Gated Recurrent Unit (GRU) decoder. Training the DTCRAE for TSC includes two phases, an unsupervised pre-training phase based on a DTCRAE and a supervised training phase for developing a TCN classifier. Computational studies are conducted to prove the effectiveness of DTCRAEs for TSC based on three datasets, the Sequential MNIST, Permuted MNIST, and Sequential CIFAR-10. Computational results demonstrate that the pre-trained DTCRAE provides a better initial structure for a TCN classifier, in terms of its higher precisions, recalls, F1-scores, and accuracies. The sensitivity analysis on the validation set shows that the pre-trained DTCRAE is robust to changes of the batch size, noisy rate, and dropout rate. DTCRAEs offer best TSC accuracies on two of three datasets and an accuracy comparable to the best one on another dataset by benchmarking against a number of state-of-the-art algorithms. Results verify the advantage of applying DTCRAEs to enhance the TSC performance of the TCN.

1. Introduction

Time series classification (TSC) is one of frequently encountered problems in data-driven applications of various fields. Solving TSC has a significant practical value but is challenging, which attracts a great amount of research efforts from data mining and machine learning communities [11,27]. To achieve a better TSC accuracy, numerous algorithms have been developed in the literature and can be categorized into two major groups, the distance-based and feature-based methods. Distance-based methods focus on training a classifier, such as the k nearest neighbor (kNN) and Support Vector Machines (SVM) [18], coupled with a distance function [26]. Feature-based methods target on extracting different types of local features from time series for classifiers. Conventional feature-based methods majorly employ different coefficients, such as wavelet coefficients [40] and Fourier coefficients [4], to extract features. Distance-based and conventional feature-based methods may require different feature extractions according to application needs, which are usually arbitrary and empirical.

Deep neural networks (DNNs) have achieved great successes in a number of classification tasks [23,41], which motivates the recent development of deep learning models for TSC [42]. Among different types of developed DNNs, three DNN architectures, Multilayer Perceptrons (MLPs), Recurrent Neural Networks (RNNs), and Convolutional Neural Networks

* Corresponding author.
E-mail address: zijzhang@cityu.edu.hk (Z. Zhang).

https://doi.org/10.1016/j.ins.2021.12.061
0020-0255/© 2021 Elsevier Inc. All rights reserved.

(CNNs), are widely adopted. MLPs are fully connected networks and one major concern of adopting MLPs for time series data is that their structures do not exhibit any spatial invariance [11]. RNNs are commonly considered options for sequence modeling tasks [2], such as the prediction task [8,35] and the classification task [19,22,38,43], in deep learning. However, the generic RNN is designed to predict an output for each element (time stamp) in the time series [21]. In addition, RNNs frequently suffer from the gradient vanishing problem due to training on long time series. The difficulty of training high quality RNNs and parallelizing their training constrains the application of generic RNNs [32,33]. Although a number of RNN variants, such as the Long Short Term Memory (LSTM) [16] and the Gated Recurrent Unit (GRU) [7], have been developed to mitigate the gradient vanishing problem to some extent, the capability of existing RNNs in capturing long-term dependencies still needs further improvements. Compared with RNNs, convolutions in CNNs can be processed in parallel to enhance the computational efficiency as the filter utilized at each layer is identical. Due to the success of convolutional architectures in various applications, especially in computer vision, extending CNNs for time series analyses has recently emerged [9,12,45]. Among convolutional architectures, the temporal convolutional network (TCN) has demonstrated its comparable performance to RNN variants in a number of sequential modeling tasks [1–3,31]. A TCN which adopted the 1D fully-convolutional network (FCN) architecture [28], causal convolutions, and dilated convolutions [31], was able to obtain a higher accuracy than RNNs based on the Sequential MNIST and Permuted MNIST datasets [2]. The TrellisNet, a variant of TCN, was developed to achieve a higher accuracy based on the Sequential MNIST, Permuted MNIST, and Sequential CIFAR-10 datasets [3]. These studies have demonstrated the great potential of the TCN in offering better TSC results.

Besides the design of network architecture, in deep learning, an unsupervised pre-training phase has been widely applied to obtain a decent representation of time series to further benefit the deep classifier training [21]. A Deep Belief Network initialized by stacking Restricted Boltzmann Machines resulted in an impressive performance [15]. Denoising autoencoders (DAEs) [38] and stacked denoising autoencoders (SDAEs) [39] considering a layer-wise initialization were introduced to learn more robust representations of inputs. An unsupervised pre-training by SDAEs was applied to initialize a model for the time series classification [5,17]. The unsupervised pre-training via autoencoders has also been successfully applied into RNNs to extract better representations for time series. Classifiers, such as the Support Vector Machine and Random Forest, were constructed on top of representations obtained by an RNN autoencoder [30,34]. LSTMs pre-trained by sequential autoencoders were shown to offer better classification results than those of LSTMs randomly initialized [10]. The multi-layer GRU autoencoder was developed on different sets of time series via an unsupervised process to serve as an off-the-shelf generic feature extractor for time series [29]. These studies demonstrated that the unsupervised pre-training by recurrent autoencoders could facilitate RNNs to obtain better representations for time series and achieve better classification results.

In the literature, integrating the TCN and the autoencoder based pre-training to achieve a better performance for TSC has not been explored. This paper aims to fill up such research gap and develops the denoising temporal convolutional recurrent autoencoders (DTCRAEs) to upgrade the performance of the TCN for TSC. A denoising temporal convolutional autoencoder (DTCRAE) is majorly composed of a TCN encoder and a GRU decoder. Training the DTCRAE for TSC includes two phases, the unsupervised training phase based on the DTCRAE and the supervised training phase for developing the TCN based classifier. The developed TCN classifier includes the TCN encoder and a fully connected layer on top of the TCN encoder. In the unsupervised pre-training phase, the raw input is firstly corrupted by forcing a fixed number of elements to zero [38]. Next, a TCN encoder is trained to map the corrupted input into a hidden vector representation and a GRU decoder is trained to reconstruct the raw input based on such vector representation via minimizing the reconstruction error. In the supervised training phase, a TCN classifier is trained via minimizing the cross-entropy loss. Computational studies are conducted based on three publicly accessible datasets to validate the advantages of the developed model. Results demonstrate that the TCN pre-trained by the DTCRAE achieves better classification accuracies than the TCN randomly initialized based on all datasets. The sensitivity analysis on the validation dataset indicates that the pre-trained DTCRAE is robust to changes of the batch size and noisy rate. Moreover, the developed DTCRAE for TSC achieves best classification accuracies on two of three considered datasets and offers an accuracy comparable to the best one on another dataset, which prove the effectiveness and benefits of DTCRAEs for TSC problems.

2. Preliminaries

In this section, a brief description of TSC is offered firstly. Two major components utilized in the DTCRAE, the GRU and the TCN, as well as structures of autoencoders (AEs) and DAEs are next introduced in detail.

A. Description of Time Series Classification

Given a time series $\mathrm{x} = (x_1, x_2, \cdots, x_T)$, the goal of time series classification is to compute a prediction vector q according to Eq. (1)

$$\mathrm{q} = f(\mathrm{x}), \tag{1}$$

where $\mathrm{q} = (q_1, q_2, \cdots, q_C)$ and C denotes the total number of classes. The q_c presents the probability that a given x belongs to the class c and $\sum_{c=1}^{C} q_c = 1$. The $f(\cdot)$ denotes the classification model which predicts the class of each time series. Based on the prediction vector $\mathrm{q} = (q_1, q_2, \cdots, q_C)$, the time series x is labelled with the class having the highest probability.

B. Gated Recurrent Unit

As shown in Fig. 1, the Gated Recurrent Unit (GRU) is a variant of RNN which incorporates two gates, a reset gate r_i and an update gate u_i, into the vanilla RNN to control the transformation of hidden states at different time steps. Given an input sequence, $x_1, x_2, \cdots, x_T$, the goal of the GRU is to produce the corresponding hidden states, $h_1, h_2, \cdots, h_T$, at each time. Hidden states of the GRU are updated according to Eqs. (2)–(5).

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \tag{2}$$

$$u_t = \sigma(W_u x_t + U_u h_{t-1} + b_u), \tag{3}$$

$$\widetilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1}) + b), \tag{4}$$

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \widetilde{h}_t. \tag{5}$$

An optional output y_t at each time step t is computed by adding a linear layer on top of the recurrent layer:

$$y_t = W_y h_t + b_y. \tag{6}$$

In Eqs. (2)–(6), matrices, W, U, W_r, U_r, W_u, U_u and W_y as well as vectors, b, b_r, b_u, and b_y, are hyper parameters which need to be optimized. The σ denotes the element-wise sigmoid function and the $\odot$ represents the Hadamard Product (an element-wise multiplication). The tanh denotes the hyperbolic tangent function, which serves as an activation function.

C. Temporal Convolutional Network

Given an input sequence $\mathrm{x} = (x_1, x_2, \cdots, x_T)$, the goal of the TCN is to predict some corresponding outputs $\mathrm{y} = (y_1, y_2, \cdots, y_T)$, at each time. As similar as the RNN, the output y_t only depends on inputs $x_1, x_2, \cdots, x_t$. Briefly speaking, a TCN maps an input of a length T to an output of the same length while prevents the information leakage from the future at each time t. The learning goal of a TCN is typically minimizing an expected loss between targets and predictions.

A TCN is constructed by stacking multiple layers of causal convolutions, where an output at time t is convolved only with elements from time t and earlier in the previous layer. The 1D fully-convolutional network (FCN) architecture [28] is adopted to ensure that the output has the same length as the input. Moreover, dilated convolutions are incorporated into the TCN to enable an exponentially large receptive field [44]. Given a 1-D sequence input $\mathrm{x} \in \mathbb{R}^n$ and a filter $f : \{0, \cdots \mathrm{k} - 1\} \to \mathbb{R}$, the dilated causal convolutional operation F on the element s of a sequence x is defined as

$$F(\mathrm{x}_s) = (\mathrm{x} *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot \mathrm{x}_{s-d \cdot i}, \ \mathrm{x}_{\le 0} := 0 \tag{7}$$

$$\mathrm{u} = (F(\mathrm{x}_1), F(\mathrm{x}_2), \cdots, F(\mathrm{x}_n)), \tag{8}$$

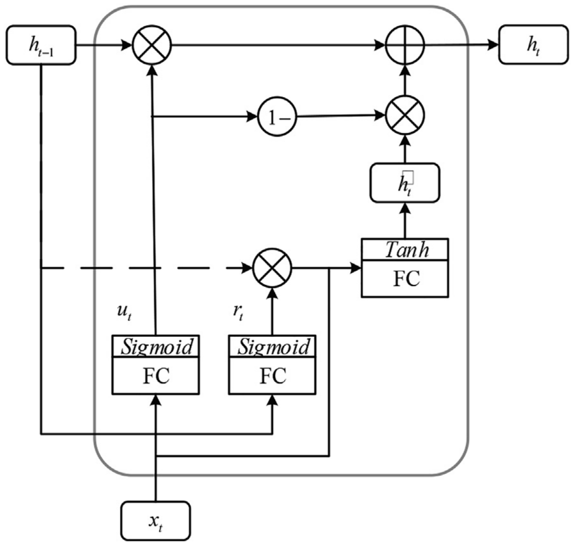


Fig. 1. The schematic diagram of the Gated Recurrent Unit.

where d is the dilation factor, k is the filter size, $s - d \cdot i$ counts the number of directions from previous nodes to the current node, and u is the output sequence of a length n. In simple terms, dilated convolutions introduce a fixed step between every two adjacent filter taps. In addition, the residual block [14], which has been widely employed in the CNN, is also adopted by the TCN. The residual connection allows layers to learn modifications to the identity mapping rather than the entire transformation. The respective field of the TCN could be increased by either choosing a larger filter size k or increasing the dilation factor d. Fig. 2 depicts the structure of the TCN, in which each residual block has two dilated causal convolutional layers.

D. Autoencoders and Denoising Autoencoders

An AE typically has a symmetric network structure composed of an encoder and a decoder. An encoder neural network maps the input vector x into a hidden vector representation h, which is assumed to be able to capture all information contained in the input sequence. From this hidden vector representation, a decoder neural network is next employed to reconstruct the input vector by minimizing the reconstruction error. The encoder stage and decoder stage are represented by Eqs. (9) and (10):

$$\mathrm{h} = f_e(\mathrm{x}), \tag{9}$$

$$\mathrm{z} = f_d(\mathrm{h}), \tag{10}$$

where z is the decoded result, $f_e(\cdot)$ and $f_d(\cdot)$ denote the encoder neural network and decoder neural network, respectively.

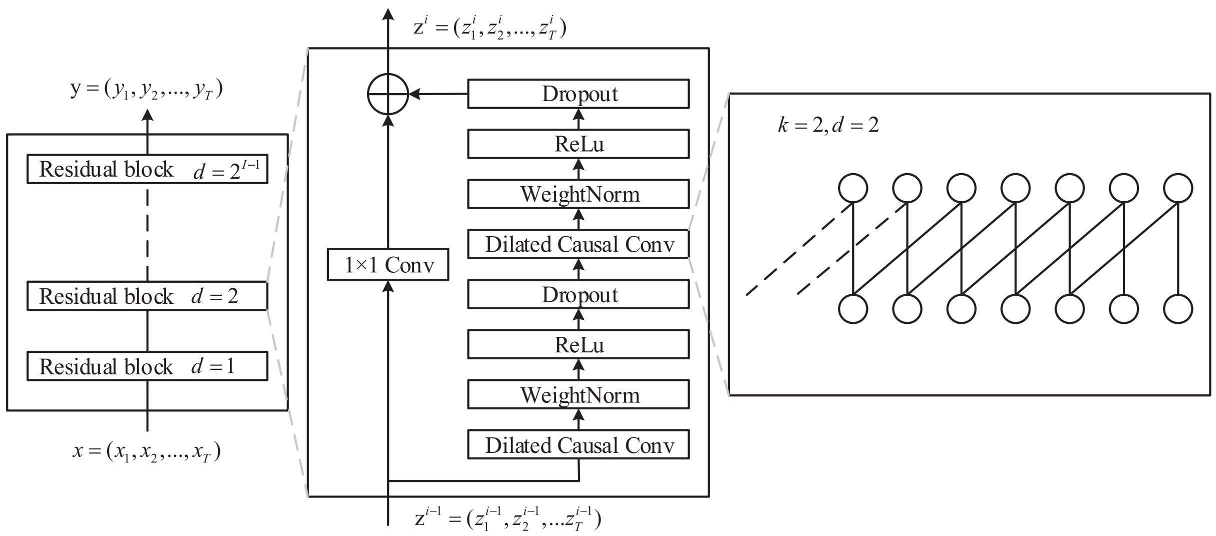
DAEs naturally extend AEs by training the network to repair the input from a corrupted or partially destroyed one [38]. Corruptions of inputs can be implemented by randomly forcing a fixed number of elements in each input to zero or adding distributional noises, such as a zero-mean Gaussian noise, to the input data [13]. From a manifold learning perspective, DAEs learn a robust mapping function from noisy examples to the clean data manifold. The capability of pre-trained DAEs for significantly improving the classification accuracy has been demonstrated in previous studies [38,39]. The network architecture of a DAE is shown in Fig. 3. Stages of the DAE including inputs corruption, encoding, and decoding are described by Eqs. (11)–(13).

$$\widehat{\mathrm{x}} = f_c(\mathrm{x}), \tag{11}$$

$$\mathrm{h} = f_e(\widehat{\mathrm{x}}), \tag{12}$$

$$\mathrm{z} = f_d(\mathrm{h}), \tag{13}$$

where $\widehat{\mathrm{x}}$ indicates the corrupted input and $f_c(\cdot)$ denotes the corruption function.



$\mathrm{y} = (y_1, y_2, \ldots, y_T)$
Residual block $d = 2^{l-1}$
Residual block $d = 2$
Residual block $d = 1$
$x = (x_1, x_2, \ldots, x_T)$
$\mathrm{z}^i = (z_1^i, z_2^i, \ldots, z_T^i)$
Dropout
ReLu
WeightNorm
Dilated Causal Conv
1×1 Conv
Dropout
ReLu
WeightNorm
Dilated Causal Conv
$\mathrm{z}^{i-1} = (z_1^{i-1}, z_2^{i-1}, \ldots z_T^{i-1})$
$k = 2, d = 2$

Fig. 2. Residual blocks in the TCN.

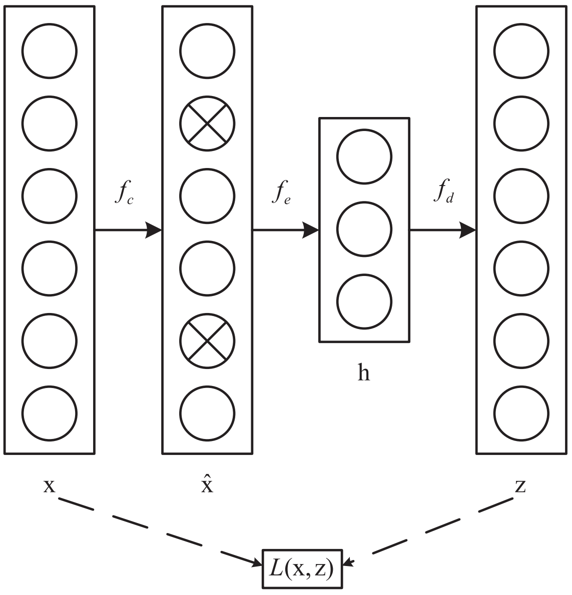


Fig. 3. The network architecture of a DAE.

3. Denoising temporal convolutional recurrent autoencoders for time series classification

We firstly describe the principle of the proposed DTCRAE and next introduce details of the framework, Pseudo Code, training criteria, as well as evaluation metrics of DTCRAEs for TSC.

A. DTCRAEs

As shown in Fig. 4, the DTCRAE model proposed in this paper consists of three stages, the inputs corruption stage, the TCN encoding stage, and the GRU decoding stage. Let $\mathrm{x} = (x_1, x_2, \cdots, x_T) \in \mathbb{R}^{D \times T}$ denote a multivariate time series with D variables of the length T, $x_t \in \mathbb{R}^D$ presents the t-th observations of all variables, $t \in \{1, 2, \cdots, T\}$, and $h_t \in \mathbb{R}^S$ denotes the hidden state of the encoder at time t, where S presents the number of hidden units in the encoder. The input sequence x is firstly corrupted by randomly forcing a fixed number of elements to zero. The corrupted input $\widehat{\mathrm{x}} = (\widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_T) \in \mathbb{R}^{D \times T}$ is then mapped into a set of hidden vector representations $\mathrm{h} = (h_1, h_2, \cdots h_\mathrm{T}) \in \mathbb{R}^{S \times T}$ by the TCN encoder. The hidden vector representation h_T at time T is supposed to encode all information in the input sequence. Next, the h_T is fed into the GRU decoder to iteratively reproduce the input sequence $\mathrm{z} = (z_1, z_2, \cdots, z_T) \in \mathbb{R}^{D \times T}$ via a reverse order [36]. The learning goal is to minimize the reconstruction loss between the input sequence x and the reconstructed input sequence z. Next, we would like to introduce the inputs corruption, TCN encoding, and GRU decoding stages of the proposed DTCRAE model.

Inputs corruption: Let v denote the noisy rate in the inputs corruption stage. A fixed number of elements in each input x, vDT, are chosen randomly and their values are set to zero while other elements in x remain the same. Based on this process, the corrupted input $\widehat{\mathrm{x}} = (\widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_T)$ is obtained by stochastically adding some zero noises to a raw input $\mathrm{x} = (x_1, x_2, \cdots, x_T)$ according to Eq. (11).

TCN encoding: Assuming that the TCN encoder is constructed by stacking a total number of I dilated causal layers. Given the corrupted input $\widehat{\mathrm{x}}$ obtained from the inputs corruption stage, hidden states of each hidden layer in the TCN encoder are calculated based on Eqs. (14) and (15).

$$h_1^{(i+1)}, h_2^{(i+1)}, \cdots h_T^{(i+1)} = \left(F\left(h_1^{(i)}\right), F\left(h_2^{(i)}\right), \cdots, F\left(h_T^{(i)}\right)\right), \quad 0 \le i \le I - 1, \tag{14}$$

$$h_1^{(0)}, h_2^{(0)}, \cdots h_T^{(0)} = \widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_T, \tag{15}$$

where hidden states of the layer $i+1$, $h_1^{(i+1)}, h_2^{(i+1)}, \cdots h_T^{(i+1)}$, are obtained by the dilated causal convolution considering hidden states of the previous layer, $h_1^{(i)}, h_2^{(i)}, \cdots h_T^{(i)}$, as inputs. The hidden state at time T of the last layer I, $h_T^{(I)}$, is next regarded as the initial hidden state of the GRU decoder.

GRU decoding: The DTCRAE employs a GRU, which only takes the initial hidden state as the input [29], as a decoder. Based on the hidden vector representation, $h_T^{(I)}$, obtained by the TCN encoder, the GRU decoder is iterated for T steps to obtain the reconstructed input z in a reverse order. Updating functions of the GRU decoding stage are presented in Eqs. (16)–(21).

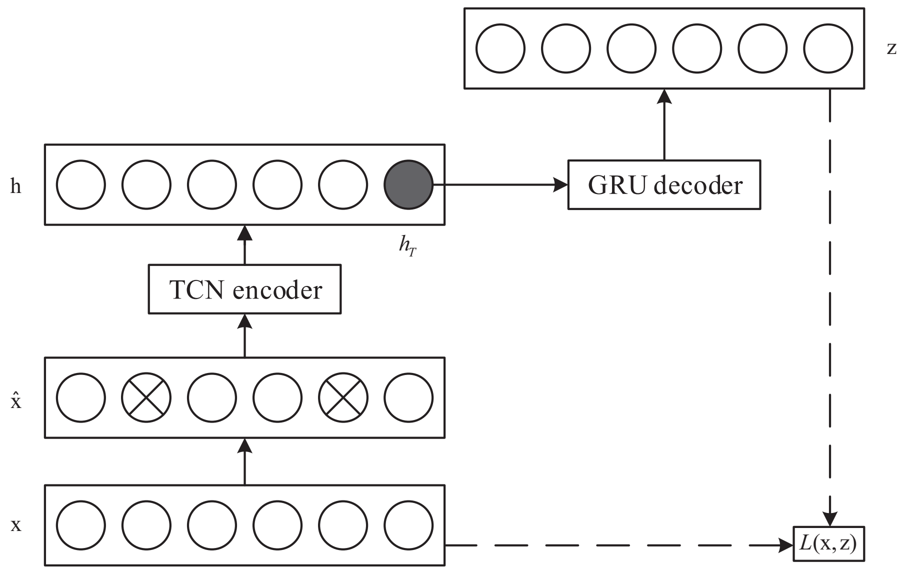


Fig. 4. The illustration of the DTCRAE.

$$r_t = \sigma\left(U_r h'_{t+1} + b_r\right), \quad 1 \le t \le T \tag{16}$$

$$u_t = \sigma\left(U_u h'_{t+1} + b_u\right), \quad 1 \le t \le T \tag{17}$$

$$\widetilde{h}_t = \tanh(U(r_t \odot h'_{t+1}) + b), \quad 1 \le t \le T \tag{18}$$

$$\widetilde{h}_t = (1 - u_t) \odot h'_{t+1} + u_t \odot \widetilde{h}_t, \quad 1 \le t \le T \tag{19}$$

$$h'_{T+1} = \tanh\left(U_i h_T^{(l)} + b_i\right), \tag{20}$$

$$z_t = W_z h'_t + b_z. \tag{21}$$

where U_i and b_i are parameters of a linear layer for keeping the hidden state $h_T^{(l)}$ obtained by the TCN encoder with the same dimension of the hidden state of the GRU decoder. The W_z and b_z are parameters of the output layer on top the GRU decoder for reconstructing the input.

B. The framework of DTCRAEs for TSC

The procedure of training a DTCRAE for TSC has two phases, the unsupervised pre-training phase via the DTCRAE and the supervised training phase for developing a TCN classifier. The TCN classifier consists of a TCN encoder and a fully connected layer on top of such TCN encoder. In the unsupervised pre-training phase, the input sequence $\mathrm{x} = (x_1, x_2, \cdots, x_T)$ is firstly processed to the corrupted input $\widehat{\mathrm{x}} = (\widehat{x}_1, \widehat{x}_2, \cdots, \widehat{x}_T)$ and next mapped to a hidden vector representation, $h_T^{(l)}$, via the TCN encoder. A GRU decoder is initialized based on the hidden vector representation $h_T^{(l)}$ to reconstruct an input sequence $\mathrm{z} = (z_1, z_2, \cdots, z_T)$. The unsupervised pre-training via the DTCRAE aims to minimize the MSE loss between the raw and reconstructed input sequences. In the supervised training phase, a TCN classifier is trained by minimizing the cross entropy loss between the true target p and the predicted classification vector q. The performance of the proposed DTCRAE for TSC is evaluated by examining the classification accuracy of the TCN classifier based on the test dataset. The framework and Pseudo Code of the DTCRAE for TSC are offered in Fig. 5 and **Algorithm 1**.

Algorithm 1 The DTCRAE for TSC

```
Inputs: Training dataset including inputs X and targets P
Parameters: mini batch size $m$, noisy rate $v$, total pre-training epochs $e_p$, total training epochs $e_t$
Output: the trained TCN classifier
1: for $e_p$ epochs do
2:    for each mini batch samples $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ in X do
3:      for $i \in \{1, \cdots, m\}$ do
4:          Inputs corruption: Randomly choose $vDT$ elements of the sample $x^{(i)}$, set their values to zero to obtain the
            corrupted input $\widehat{x}^i$.
5:          TCN encoding: Given the corrupted input $\widehat{x}^i$, the hidden vector representation $h_T^l$ is computed according to
            Eqs. (14) and (15).
6:          GRU decoding: Given the hidden vector representation $h_T^l$, the reconstructed output $z^{(i)}$ is computed from
            Eqs. (16)–(21).
7:      end for
8:    Compute the MSE reconstruction loss $L_{\mathrm{MSE}} = \sum_{i=1}^{m} \|x^{(i)} - z^{(i)}\|_2^2 / mT$.
9:    Compute the gradient of the loss $L_{\mathrm{MSE}}$ with respect to parameters of the TCN encoder and the GRU decoder. Update
      these parameters by the Adam optimizer.
10:   end for
11: end for
12: for $e_t$ epochs do
13:   for each mini batch samples $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ in X and corresponding targets $\{p^{(1)}, p^{(2)}, \cdots, p^{(m)}\}$ in P do
14:      Given the input $x^{(i)}$, compute the hidden vector representation $h_T^l$ according to Eqs. (14) and (15).
15:      Compute the classification vector $q^{(i)}$ via the fully connected layer based on the hidden vector representation $h_T^l$.
16:   end for
17:   Compute the cross entropy loss $L_{\mathrm{H}} = -\sum_{i=1}^{m} \sum_{c=1}^{C} \left[ p_c^{(i)} log q_c^{(i)} + \left(1 - p_c^{(i)}\right) log\left(1 - q_c^{(i)}\right)\right] / m$.
18:   Compute the gradient of the loss $L_{\mathrm{H}}$ with respect to parameters of the TCN encoder and fully connected layer.
      Update these parameters by the Adam optimizer.
19: end for
20: Return the trained TCN classifier
```

C. Training Criteria and Evaluation Metrics

We consider the Mean Squared Error (MSE) as the reconstruction loss in the unsupervised pre-training phase and cross entropy loss as the classification loss in the supervised training phase. The definition of the MSE and cross entropy loss are expressed by Eqs. (22) and (23).

Given an input sequence $x = (x_1, x_2, \cdots, x_T) \in \mathbb{R}^{D \times T}$ and a reconstructed input $z = (z_1, z_2, \cdots, z_T) \in \mathbb{R}^{D \times T}$, the MSE loss is defined as

$$L_{\mathrm{MSE}} = \sum_{t=1}^{T} \sum_{d=1}^{D} \left(x_t^d - z_t^d\right)^2 / T. \tag{22}$$

Given the true one-hot target $p \in \{0, 1\}^C$ and classification Bernoulli vector $q \in [0, 1]^C$, the cross entropy loss is defined as

$$L_{\mathrm{H}} = -\sum_{c=1}^{C} [p_c log q_c + (1 - p_c) log(1 - q_c)], \tag{23}$$

where C denotes the total number of classes in the classification problem.

Four evaluation metrics, the classification accuracy, precision, recall, and the F1-score, are applied to evaluate the model performance based on the test dataset.

Classification accuracy: The classification accuracy is defined in Eq. (24).

$$\text{Accuracy} = \frac{N_r}{N} \times 100\%, \tag{24}$$

where N_r and N denote the number of correctly classified samples and total number of samples in the dataset.

Precision, **recall,** and **F1-score**: Definitions of the precision, recall, and F1-score are presented in Eqs. (25)–(27).

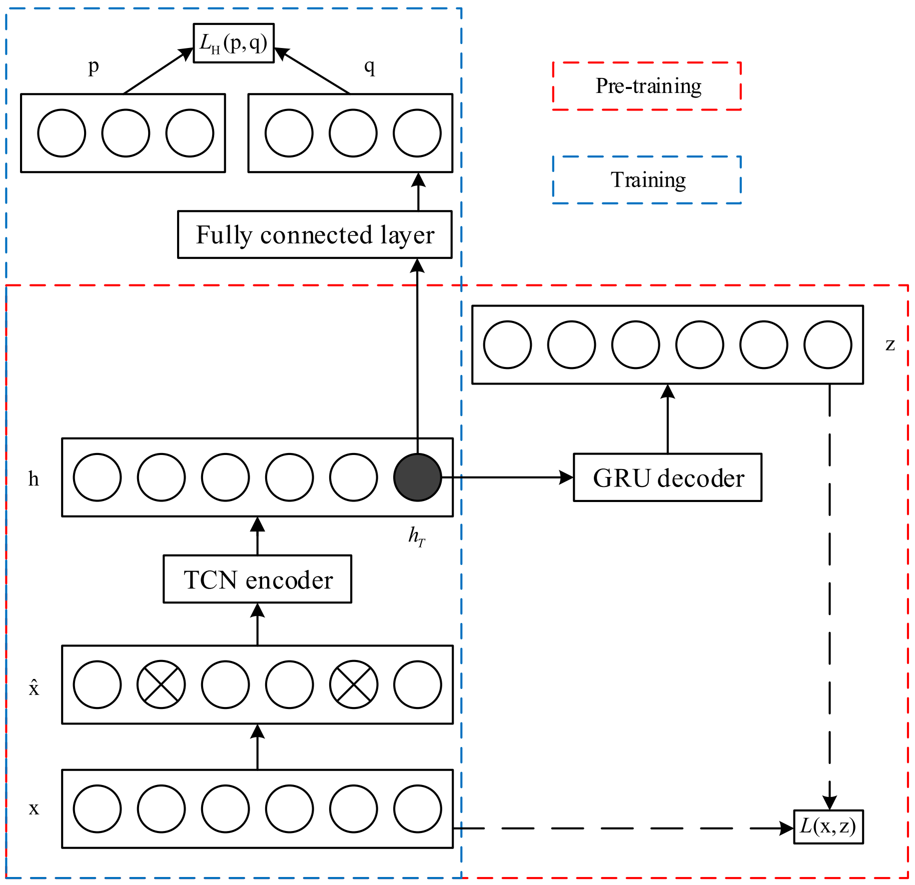


Fig. 5. The framework of the DTCRAE for TSC.

$$\text{Precision} = \frac{TP}{TP + FP} \times 100\%, \tag{25}$$

$$\text{Recall} = \frac{TP}{TP + FN} \times 100, \tag{26}$$

$$F_1 = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}, \tag{27}$$

where TP, FP, and FN present the number of true positive samples, false positive samples. and false negative samples, respectively.

4. Computational experiments

The proposed DTCRAE for TSC is assessed based on three widely utilized benchmark TSC datasets, the Sequential MNIST[1] (Seq. MNIST), Permuted MNIST (Per. MNIST) and Sequential CIFAR-10[2] (Seq. CIFAR-10). Classification accuracies of the proposed DTCRAE for TSC and state-of-the-art methods in the literature, which are considered as benchmarks, are computed and compared.

A. Benchmarks

The proposed DTCRAE is compared with 7 different benchmark models, the dilated recurrent neural network (dilated RNN [6]), independently recurrent neural network (IndRNN [25]), TCN, Long Short Term Memory Networks with

[1] http://yann.lecun.com/exdb/mnist/
[2] https://www.cs.toronto.edu/~kriz/cifar.html

reconstruction auxiliary loss (r-LSTM [37]), Transformer ([46]), TrellisNet ([3]), and Dense IndRNN ([24]). Besides the TCN[3], which has been described in Section 2, other six benchmark models are briefly introduced as follows.

Dilated GRU[4]: The dilated GRU is a recurrent connection structure characterized by multi-resolution dilated recurrent skip connections. The dilated recurrent skip connection can be described by Eq. (28), where $c_t^{(l)}$ denotes the cell in layer l at time t and $s^{(l)}$ refers to the skip length (dilation) of layer l. The $f(\cdot)$ is a nonlinear function implemented via the GRU cell. Multiple dilated recurrent layers with hierarchical dilations are stacked together to construct a dilated recurrent neural networks. Compared with conventional RNNs, the dilated GRU has been shown to be able to extend the range of temporal dependencies with fewer parameters and significantly improve the computational efficiency.

$$c_t^{(l)} = f\left(x_t^{(l)}, c_{t-s^{(l)}}^{(l)}\right). \tag{28}$$

IndRNN[5]: The IndRNN is a new type of RNN in which neurons in the same layer are independent of each other and they are connected across layers. The hidden state in the IndRNN is updated based on Eq. (29), where $h_{n,t}$ is the hidden state for the n-th neuron at time step t. The σ denotes the element-wise sigmoid function. The IndRNN has been empirically shown to be able to capture long-term dependencies as well as prevent the gradient vanishing and exploding problems in Vanilla RNN. A dense IndRNN is a specialized IndRNN, which incorporates dense connections into the deep layers.

$$h_{n,t} = \sigma(W_n x_t + u_n h_{n,t-1} + b_n). \tag{29}$$

r-LSTM: The r-LSTM incorporates an auxiliary loss at random anchor points into the LSTM to force the LSTM to reconstruct previous events. The intuition is that if events to be predicted are close enough to the anchor point, the number of back propagation through time (BPTT) steps needed for the decoder to reconstruct past events can be quite small [37]. Empirically, the r-LSTM could improve the ability of RNNs to capture long-term dependencies.

Transformer: A Transformer adopts multi-head self-attention layers and positional-wise feed-forward layers in the encoder-decoder structure to replace recurrent layers for sequence transduction tasks. In the encoder, the multi-head self-attention layer is realized based on Eqs. (30)–(33), where M is the number of heads and $h \in \mathbb{R}^{n\times d}$ is the input vector of dimension d. The Q_m, K_m, V_m and O_m represent the query, key, value, and output metrics of the m-th head, respectively. The $d_k = d/M$, W_m^Q, W_m^K and W_m^V are learnable parameters. The softmax denotes a normalized exponential function to normalize the output of a network to a probability distribution over predicted output classes. A positional-wise feed-forward layer is utilized to conduct a linear projection on the concatenation of $(O_1, O_2, \cdots, O_M)$ to obtain the output of the multi-head self-attention layer. The decoder is implemented in a similar way as the encoder. Compared with RNNs and CNNs, the Transformer is more parallelizable and requires significantly less time to train.

$$Q_m = hW_m^Q, \quad 1 \le \mathrm{m} \le M \tag{30}$$

$$K_m = hW_m^K, \quad 1 \le \mathrm{m} \le M \tag{31}$$

$$V_m = hW_m^V, \quad 1 \le \mathrm{m} \le M \tag{32}$$

$$O_m = \mathrm{softmax}\left(\frac{Q_m K_m^T}{\sqrt{d_k}}\right) V_m, \quad 1 \le \mathrm{m} \le M \tag{33}$$

TrellisNet[6]: As a special TCN, the TrellisNet directly injects the input into each of the hidden layers and adopts weight tying across the depth. The feature vectors in the TrellisNet are computed based on Eqs. (34) and (35), where x_t is the input vector and $z_t^{(i)}$ is the feature vector in the i-th layer of the network, at time step t. The W_1 and W_1 are learnable parameters and $f(\cdot)$ is a nonlinear function implemented via the LSTM cell. It has been shown that the TrellisNet generalizes to a truncated recurrent network. Empirical experiments have demonstrated that the TrellisNet outperforms the current state-of-the-art models in a variety of sequence modeling tasks.

$$\widehat{z}_{t+1}^{(i+1)} = W_1\left[x_t, z_t^{(i)}\right] + W_2\left[x_{t+1}, z_{t+1}^{(i)}\right] \tag{34}$$

$$z_{t+1}^{(i+1)} = f\left(\widehat{z}_{t+1}^{(i+1)}, z_t^{(i)}\right) \tag{35}$$

[3] http://github.com/locuslab/TCN.
[4] https://github.com/code-terminator/DilatedRNN
[5] https://github.com/Sunnydreamrain/IndRNN_pytorch
[6] https://github.com/locuslab/trellisnet

B. Dataset Description

Classification tasks on Sequential MNIST, Permuted MNIST, and Sequential CIFAR10 datasets have been widely utilized to evaluate the capability of a model on capturing long dependencies [2]. The MNIST is a dataset of 70,000 Gy scale images with a size of 28 $\times$ 28 pixels. It is a widely considered classification dataset for handwritten digits ranging from 0 to 9 and having 10 patterns in total. A digit image selected from the MNIST dataset and the corresponding sequential image are shown in Fig. 6 as illustrative examples. In the Sequential MNIST dataset, MNIST images [23] are presented to the model as a 1 $\times$ 784 sequence for the digit classification. In the Permuted MNIST dataset, the order of the sequence is permuted at random [20]. The CIFAR-10 dataset contains 60,000 color images with a size of 32 $\times$ 32 pixels and 3 channels. These images include 10 different classes representing airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6000 images for each of these 10 classes. An object image selected from the CIFAR-10 dataset and the corresponding sequential image are shown in Fig. 7 as illustrative examples. In the Sequential CIFAR-10 dataset, images are processed as a 3 $\times$ 1024 sequence [3]. In each of three datasets, the raw training dataset is split into the training and validation datasets with a ratio 9:1. Statistical summaries of three datasets are produced and reported in Table 1.

In Table 1, the *N* represents the total number of samples in the dataset while *L* and *D* denote the length of time series and the dimension of the input.

C. Model Development

TCN encoder in the DTCRAE is composed of 8 residual blocks. The number of channels of each dilated causal convolutional layer is set to 64. The kernel size of the TCN encoder is set to 7 to ensure a sufficient respective field. In the GRU decoder, the dimension of the hidden state is set to 150. The initial learning rate for the unsupervised pre-training phase and supervised training phase are set to 0.0001 and 0.001, respectively. The Adam optimizer is applied into both of unsupervised pre-training and supervised training phases. The total number of training epochs in the unsupervised pre-training and supervised training are set to 300 and 50 respectively.

We conduct a comparative analysis to study the impact of three key hyper parameters, the batch size, and noisy rate of the mini batch training, as well as the dropout rate on training the TCN. At the input corruption stage for training the DTCRAE, a set of candidate settings of the noisy rate, {0.0, 0.1, 0.2, 0.3}, is considered. Two different types of batch sizes, 32 and 64, are tried for both of DTCRAE and TCN without pre-training. In the TCN, the dropout rate is searched with considering a set of candidate options {0.0, 0.1, 0.2}. Best hyper parameters are next selected according to the classification performance based on the validation dataset. The comparison of precisions, recalls, F1-scores and classification accuracies of different models is conducted based on the test dataset.

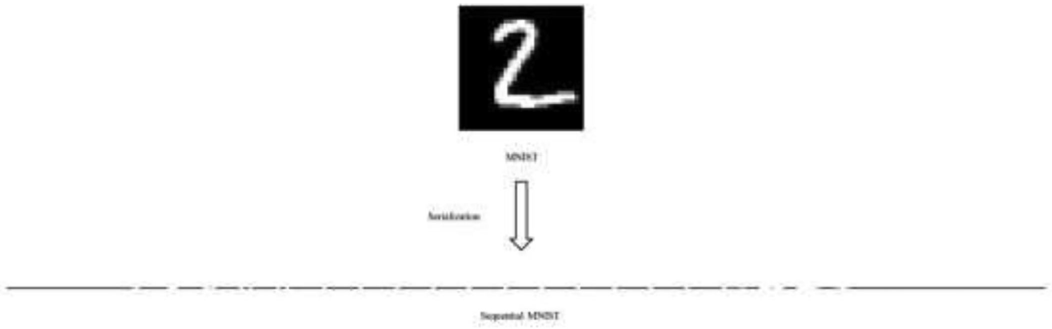


Fig. 6. A digit image of "2" of size 1 $\times$ 28 $\times$ 28 from MNIST dataset and the corresponding sequential image of size 1 $\times$ 784.

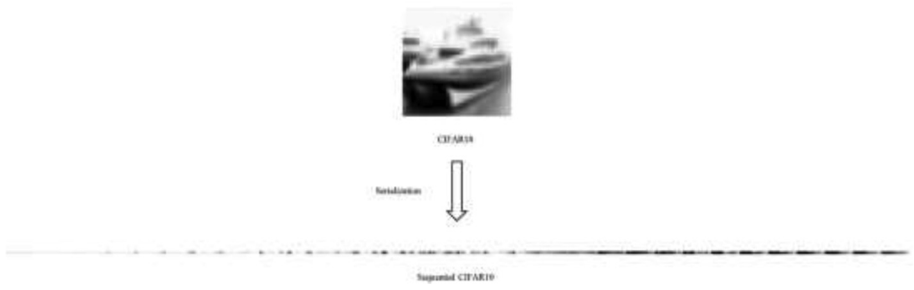


Fig. 7. A ship image of size 3 $\times$ 32 $\times$ 32 from the CIFAR-10 dataset and the corresponding sequential image of size 3 $\times$ 1024.

Table 1
Dataset statistics.

Datasets	*N* of training	*N* of validation	*N* of test	*L*	*D*
Seq. MNIST	54,000	6000	10,000	784	1
Permuted MNIST	54,000	6000	10,000	784	1
Seq. CIFAR-10	45,000	5000	10,000	1024	3

D. Experimental Results

The proposed DTCRAE for TSC is implemented on the Sequential MNIST, Permuted MNIST, and Sequential CIFAR-10 datasets The Receiver Operating Characteristic (ROC) curves and the Area Under The Curve (AUC) on all datasets are shown in Fig. 8. The pre-training losses and training losses of the DTCRAE are presented in Figs. 9–11. The comparative analysis of settings of the batch size, noisy rate, and dropout rate based on the validation dataset is reported in Table 2. The precision, recall, and F1-score of the TCN and DTCRAE on the test dataset are compared in Table 3. In Table 4, we compare classification accuracies of the proposed model and considered benchmark models based on the test dataset.

Fig. 8 shows that the proposed DTCRAE achieves very nice performances on three datasets. Specifically, the AUC obtained by the DTCRAE is 1 on the Seq. MNIST and Permuted MNIST dataset as well as 0.97 on the CIFAR-10 dataset. These results demonstrate that the well trained DTCRAE model has a strong capability of separating data of different classes on considered datasets.

Figs. 9–11 demonstrate the convergence of the DTCRAE based on three datasets. These results validate the effectiveness of the hidden representations learned by the DTCRAE for reproducing inputs. This good hidden representation serves as a great initialization for the next classification task.

As shown in Table 2, based on different batch sizes, noisy rates, and dropout rates, the TCN classifier pre-trained by a DTCRAE consistently offers a better classification accuracy than that of the TCN classifier initialized randomly based on validation datasets derived from three considered datasets. Table 3 shows that the proposed DTCRAE obtains better precisions, recalls, and F1-scores than the TCN. These results imply that the pre-trained DTCRAE provides a better initialization for the TCN classifier by comparing with the random initialization. It is also observable that the pre-trained DTCRAE is robust to changes of the batch size, noisy rate, and dropout rate.

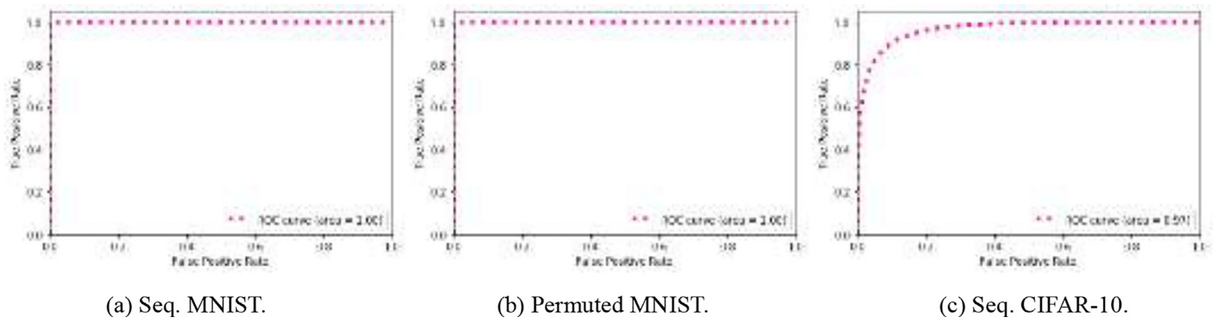


Fig. 8. ROC curves and AUC on (a) Seq. MNIST, (b) Permuted MNIST and (c) Seq. CIFAR-10, respectively.

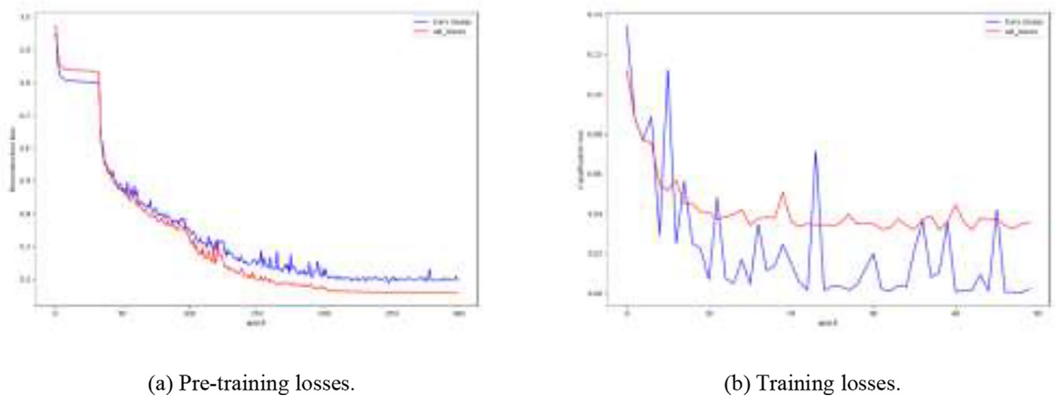
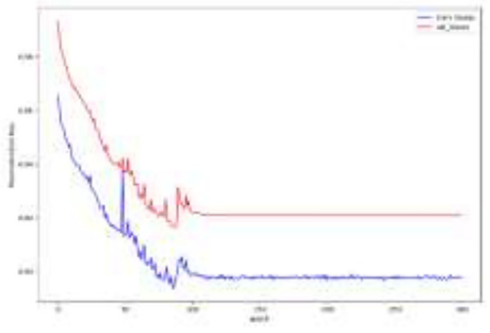


Fig. 9. Pre-training and training losses of the DTCRAE on the Seq. MNIST dataset.


(a) Pre-training losses.

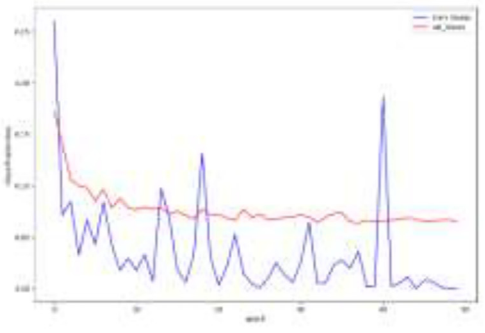
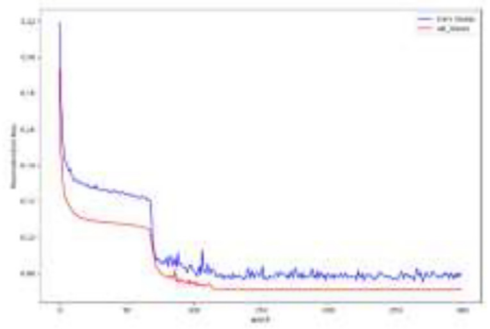

(b) Training losses.

Fig. 10. Pre-training and training losses of the DTCRAE on the Permuted MNIST dataset.


(a) Pre-training losses.

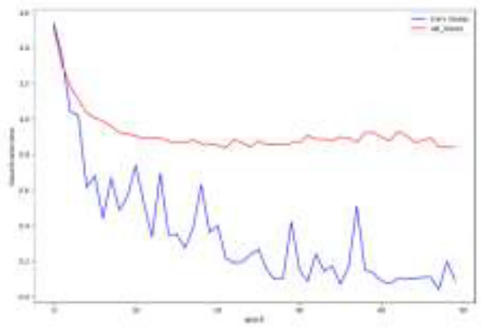

(b) Training losses.

Fig. 11. Pre-training and training losses of the DTCRAE on the Seq. CIFAR-10 dataset.

Table 2
The sensitivity analysis of the batch size, noisy rate and the dropout rate based on the validation dataset.

Dataset	Dropout	Batch size	Model (noisy rate)				
			TCN	DTCRAE (0.0)	DTCRAE (0.1)	DTCRAE (0.2)	DTCRAE (0.3)
Seq. MNIST	0	32	98.63	**98.88**	**99.1**	**99.03**	**98.77**
		64	98.82	98.77	**99.03**	98.70	**99.05**
	0.1	32	99.00	**99.22**	**99.03**	98.93	**99.10**
		64	98.88	**99.37**	**99.08**	**99.25**	**99.03**
	0.2	32	99.00	**99.12**	**99.23**	**99.25**	**99.33**
		64	99.08	**99.18**	**99.23**	**99.20**	**99.15**
Permuted MNIST	0	32	97.38	97.03	**98.07**	**98.07**	**98.07**
		64	97.33	**97.77**	**97.8**	**98.03**	**98.00**
	0.1	32	98.22	98,07	**98.23**	97.92	97.85
		64	98.25	98.10	**98.37**	**98.30**	98.13
	0.2	32	98.00	97.71	**98.23**	**98.18**	**98.35**
		64	98.18	**98.35**	98.07	**98.22**	**98.20**
Seq. CIFAR-10	0	32	63.24	**63.44**	**66.24**	**63.60**	63.18
		64	60.26	**63.54**	**63.84**	**64.46**	**63.22**
	0.1	32	69.50	**71.98**	**69.6**	**72.50**	**69.86**
		64	69.22	**70.08**	**71.26**	**70.48**	**70.00**
	0.2	32	68.74	**71.22**	**71.3**	**74.48**	**73.66**
		64	72.96	72.48	**74.46**	72.36	**73.18**

Table 3
Comparison of precision, recall and F1-score of the TCN and the proposed DTCRAE.

Dataset	Model	Precision	Recall	F1-score
Seq. MNIST	TCN	99.14	99.13	99.13
	DTCRAE	**99.21**	**99.20**	**99.20**
Permuted MNIST	TCN	98.11	98.20	98.15
	DTCRAE	**98.50**	**98.40**	**98.45**
Seq. CIFAR-10	TCN	72.68	72.49	72.58
	DTCRAE	**74.24**	**73.81**	**74.02**

Table 4
Comparison of classification accuracy (%) of different models on the test dataset.

Model	Seq. MNIST	Permuted MNIST	Seq. CIFAR-10
Dilated GRU [6]	99.00	94.60	None
IndRnn [25]	99.00	96.00	None
Generic TCN [2] (re-implemented)	99.14	98.26	72.49
r-LSTM w/ Aux. Loss [37]	98.40	95.20	72.20
Transformer (self-attention) [37]	98.90	97.90	62.20
TrellisNet [3]	99.20	98.13	73.42
Dense IndRNN [24]	**99.48**	97.20	None
TCN pre-trained by DTCRAE	99.21	**98.40**	**73.81**

As shown in Table 4, it is clear that the DTCRAE achieves the best classification accuracies based on the test dataset derived from Permuted MNIST and Sequential CIFAR-10 datasets. Based on the test dataset derive from the Sequential MNIST dataset, the DTCRAE offers a comparable classification accuracy to the best one, which is obtained by the dense IndRNN [24]. These results validate the effectiveness and advantage of the proposed DTCRAE for TSC.

5. Conclusions

In this paper, a denoising temporal convolutional recur-rent autoencoder (DTCRAE) was proposed for time series classification (TSC). The training of the proposed DTCRAE had two phases, the unsupervised pre-training phase via a DTCRAE and the supervised training phase for developing a TCN classifier. The TCN classifier was composed of a TCN encoder and a fully connected layer. In the unsupervised pre-training phase, the DTCRAE utilized a TCN encoder to map the corrupted input into a hidden vector representation, from which a GRU decoder for reconstructing the input was next constructed. In the supervised training phase, a TCN classifier was trained by minimizing the cross-entropy classification error. The classification accuracy, precision, recall, and the F1-score were applied to evaluate the performance of the model based on the test dataset.

The developed DTCRAE for TSC was compared with different benchmarks based on three widely utilized publicly accessible datasets, the Sequential MNIST, Permuted MNIST and Sequential CIFAR-10 datasets. Computational results demonstrated that the pre-trained DTCRAE could improve the TSC performance of a TCN classifier because the pre-trained DTCRAE provided a better initialization for training a TCN classifier. The sensitivity analysis based on the validation dataset demonstrated that the pre-trained DTCRAE was robust to changes of the batch size, noisy rate, and the dropout rate in the TCN. In addition, DTCRAEs offered best TSC accuracies on two of three datasets and an accuracy comparable to the best one on another dataset by benchmarking against a number of state-of-the-art algorithms. These results verified the advantage of pre-trained DTCRAEs in enhancing the TSC performance of the TCN. One implementation issue about adopting the DTCRAE is that the kernel size in the TCN should be large enough to ensure a sufficient respective field. The proposed DTCRAE is promising to achieve a decent classification performance especially on time series datasets which demonstrate long time dependences.

CRediT authorship contribution statement

Zhong Zheng: Methodology, Validation, Formal analysis, Writing – original draft. **Zijun Zhang:** Conceptualization, Supervision, Writing – review & editing, Funding acquisition. **Long Wang:** Writing – review & editing, Funding acquisition. **Xiong Luo:** Writing – review & editing, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China Youth Scientist Fund Project with No. 52007160, in part by the Hong Kong Research Grants Council General Research Fund Projects with No. 11215418 and 11204419, in part by the Laboratory for AI-Powerd Financial Technologies Limited, as well as inpart by the Interdisciplinary Research Project for Young Teachers of USTB (Fundamental Research Funds for the Central Universities) under Grant FRF-IDRY-19-017.

References

[1] E. Aksan, O. Hilliges, STCN: Stochastic Temporal Convolutional Networks, ArXiv Prepr. ArXiv190206568 (2019).
[2] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, ArXiv Prepr. ArXiv180301271 (2018).
[3] S. Bai, J.Z. Kolter, V. Koltun, Trellis networks for sequence modeling, ArXiv Prepr. ArXiv181006682 (2018).
[4] A. Bailly, S. Malinowski, R. Tavenard, L. Chapel, T. Guyet, Dense bag-of-temporal-sift-words for time series classification, in: Int. Workshop Adv. Anal. Learn. Temporal Data, Springer, 2015, pp. 17–30.
[5] Y. Bengio, L. Yao, G. Alain, P. Vincent, Generalized denoising auto-encoders as generative models, in: Adv. Neural Inf. Process. Syst., 2013, pp. 899–907.
[6] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M.A. Hasegawa-Johnson, T.S. Huang, Dilated recurrent neural networks, in: Adv. Neural Inf. Process. Syst., 2017, pp. 77–87.
[7] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, ArXiv Prepr. ArXiv14123555 (2014).
[8] J.T. Connor, R.D. Martin, L.E. Atlas, Recurrent neural networks and robust time series prediction, IEEE Trans. Neural Netw. 5 (1994) 240–254.
[9] Z. Cui, W. Chen, Y. Chen, Multi-scale convolutional neural networks for time series classification, ArXiv Prepr. ArXiv160306995 (2016).
[10] A.M. Dai, Q.V. Le, Semi-supervised sequence learning, in: Adv. Neural Inf. Process. Syst., 2015, pp. 3079–3087.
[11] H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Deep learning for time series classification: a review, Data Min. Knowl. Discov. 33 (2019) 917–963.
[12] J.C.B. Gamboa, Deep learning for time-series analysis, ArXiv Prepr. ArXiv170101887 (2017).
[13] L. Gondara, K. Wang, Multiple imputation using deep denoising autoencoders, ArXiv Prepr. ArXiv170502737 (2017).
[14] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.
[15] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (2006) 1527–1554.
[16] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[17] Q. Hu, R. Zhang, Y. Zhou, Transfer learning for short-term wind speed prediction with deep neural networks, Renew. Energy. 85 (2016) 83–95.
[18] C. Huang, L. Wang, L.L. Lai, Data-driven short-term solar irradiance forecasting based on information of neighboring sites, IEEE Trans. Ind. Electron. 66 (2018) 9918–9927.
[19] M. Hüsken, P. Stagge, Recurrent neural networks for time series classification, Neurocomputing 50 (2003) 223–235.
[20] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N.R. Ke, A. Goyal, Y. Bengio, A. Courville, C. Pal, Zoneout: Regularizing rnns by randomly preserving hidden activations, ArXiv Prepr. ArXiv160601305 (2016).
[21] M. Längkvist, L. Karlsson, A. Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling, Pattern Recogn. Lett. 42 (2014) 11–24.
[22] S. Lawrence, C.L. Giles, S. Fong, Natural language grammatical inference with recurrent neural networks, IEEE Trans. Knowl. Data Eng. 12 (2000) 126–140.
[23] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE. 86 (1998) 2278–2324.
[24] S. Li, W. Li, C. Cook, Y. Gao, C. Zhu, Deep Independently Recurrent Neural Network (IndRNN), ArXiv Prepr. ArXiv191006251 (2019).
[25] S. Li, W. Li, C. Cook, C. Zhu, Y. Gao, Independently recurrent neural network (indrnn): Building a longer and deeper rnn, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 5457–5466.
[26] J. Lines, A. Bagnall, Time series classification with ensembles of elastic distance measures, Data Min. Knowl. Discov. 29 (2015) 565–592.
[27] H. Long, L. Sang, Z. Wu, W. Gu, Image-based Abnormal Data Detection and Cleaning Algorithm via Wind Power Curve, IEEE Trans. Sustain Energy (2019).
[28] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 3431–3440.
[29] P. Malhotra, V. TV, L. Vig, P. Agarwal, G. Shroff, TimeNet: Pre-trained deep recurrent neural network for time series classification, ArXiv Prepr. ArXiv170608838 (2017).
[30] N. Mehdiyev, J. Lahann, A. Emrich, D. Enke, P. Fettke, P. Loos, Time series classification using deep learning for process planning: a case from the process industry, Proc. Comput. Sci. 114 (2017) 242–249.
[31] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, Wavenet: A generative model for raw audio, ArXiv Prepr. ArXiv160903499 (2016).
[32] R. Pascanu, T. Mikolov, Y. Bengio, Understanding the exploding gradient problem, CoRR Abs12115063. 2 (2012).
[33] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: Int. Conf. Mach. Learn., 2013, pp. 1310–1318.
[34] D. Rajan, J.J. Thiagarajan, A generative modeling approach to limited channel ecg classification, in: 2018 40th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBC, IEEE, 2018, pp. 2571–2574.
[35] E.W. Saad, D.V. Prokhorov, D.C. Wunsch, Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks, IEEE Trans. Neural Netw. 9 (1998) 1456–1470.
[36] I. Sutskever, O. Vinyals, Q. Le, Sequence to sequence learning with neural networks, Adv. NIPS (2014).
[37] T.H. Trinh, A.M. Dai, M.-T. Luong, Q.V. Le, Learning longer-term dependencies in rnns with auxiliary losses, ArXiv Prepr. ArXiv180300144 (2018).
[38] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proc. 25th Int. Conf. Mach. Learn., ACM, 2008, pp. 1096–1103.
[39] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2010) 3371–3408.
[40] J. Wang, P. Liu, M.F. She, S. Nahavandi, A. Kouzani, Bag-of-words representation for biomedical time series classification, Biomed. Signal. Process. Control. 8 (2013) 634–644.
[41] L. Wang, Z. Zhang, J. Xu, R. Liu, Wind turbine blade breakage monitoring with deep autoencoders, IEEE Trans. Smart Grid. 9 (2016) 2824–2833.
[42] Z. Wang, W. Yan, T. Oates, Time series classification from scratch with deep neural networks: A strong baseline, in: 2017 Int. Jt. Conf. Neural Netw. IJCNN, IEEE, 2017, pp. 1578–1585.
[43] R. Watrous, G. Kuhn, Induction of nitestate automata using second order recurrent networks, nips4, n.d.
[44] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, ArXiv Prepr. ArXiv151107122 (2015).

[45] Y. Zheng, Q. Liu, E. Chen, Y. Ge, J.L. Zhao, Time series classification using multi-channels deep convolutional neural networks, in: Int. Conf. Web-Age Inf. Manag., Springer, 2014, pp. 298–310.
[46] [1706.03762] Attention Is All You Need, (n.d.).