Neural Networks 154 (2022) 481–490

Contents lists available at ScienceDirect

Neural Networks

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

Multivariate time-series classification with hierarchical variational graph pooling

Ziheng Duan [a,b,1], Haoyan Xu [a,b,c,1], Yueyang Wang [a,*], Yida Huang [a], Anni Ren [b], Zhongbin Xu [b], Yizhou Sun [c], Wei Wang [c]

[a] School of Big Data and Software Engineering, Chongqing University, Chongqing, 401331, China
[b] College of Energy Engineering, Zhejiang University, Zhejiang, 310027, China
[c] Department of Computer Science, University of California, Los Angeles, CA 90095, USA

ARTICLE INFO

Article history:
Received 5 November 2021
Received in revised form 22 March 2022
Accepted 26 July 2022
Available online 2 August 2022

Keywords:
Multivariate time series classification
Graph neural networks
Graph pooling
Graph classification

ABSTRACT

In recent years, multivariate time-series classification (MTSC) has attracted considerable attention owing to the advancement of sensing technology. Existing deep-learning-based MTSC techniques, which mostly rely on convolutional or recurrent neural networks, focus primarily on the temporal dependency of a single time series. Based on this, complex pairwise dependencies among multivariate variables can be better described using advanced graph methods, where each variable is regarded as a node in the graph, and their dependencies are regarded as edges. Furthermore, current spatial–temporal modeling (e.g., graph classification) methodologies based on graph neural networks (GNNs) are inherently flat and cannot hierarchically aggregate node information. To address these limitations, we propose a novel graph-pooling-based framework, MTPool, to obtain an expressive global representation of MTS. We first convert MTS slices into graphs using the interactions of variables via a graph structure learning module and obtain the spatial–temporal graph node features via a temporal convolutional module. To obtain global graph-level representation, we design an "encoder-decoder"-based variational graph pooling module to create adaptive centroids for cluster assignments. Then, we combine GNNs and our proposed variational graph pooling layers for joint graph representation learning and graph coarsening, after which the graph is progressively coarsened to one node. Finally, a differentiable classifier uses this coarsened representation to obtain the final predicted class. Experiments on ten benchmark datasets showed that MTPool outperforms state-of-the-art strategies in the MTSC task.

1. Introduction

Multivariate time series (MTS), which are obtained from numerous variables or sensors in our daily lives, have been utilized in different investigations (Feng & Niu, 2021; Li et al., 2021; Shi et al., 2021; Zhou et al., 2020). Attributable to cutting-edge sensing techniques, the multivariate time-series classification (MTSC) issue, which recognizes the labels for MTS records, has attracted considerable attention in recent years (Baldán & Benítez, 2019; Mori, Mendiburu, Miranda, & Lozano, 2019; Zhang, Gao, Lin, & Lu, 2020). MTSC models have been applied in a broad range of real-world applications (Iwana, Frinken, & Uchida, 2020; Zhou, Wang, Huang, & Liu, 2021), such as sleep stage identification (Sun, Chen, Li, Fan, & Chen, 2019), healthcare (Kang & Choi, 2014), and action recognition (Yu & Lee, 2015). Specifically, MTS has the following two significant attributes: (1) each univariate time series has an *inner temporal reliance* mode, and (2) there always exist *complex hidden dependency* relationships among different MTS variables. Capturing these two attributes is a significant contribution to achieving better classification performance but is also a challenging task.

Several MTS classification methods have been proposed in previous studies. Distance-based methods, such as dynamic time warping (DTW) with k-NN (Seto, Zhang, & Zhou, 2015) and feature-based methods such as the hidden unit logistic model (HULM) (Pei, Dibeklioğlu, Tax, & van der Maaten, 2017), are successful in classification tasks on many benchmark MTS datasets. In any case, these methodologies require hefty crafting of data preprocessing and feature engineering, and they cannot fully explore the *inner temporal reliance* mode in individual univariate time series. Recently, many deep-learning-based methods have been exploited for end-to-end MTS classification. Fully convolutional networks (FCNs) and residual networks (ResNets) can achieve comparable or better performances than traditional methods (Wang, Yan, & Oates, 2017). The MLSTM-FCN (Karim, Majumdar, Darabi, & Harford, 2019) utilizes an LSTM layer and

* Corresponding author.
E-mail address: yueyangwang@cqu.edu.cn (Y. Wang).
[1] These authors contributed equally to this research.

https://doi.org/10.1016/j.neunet.2022.07.032
0893-6080/

a stacked convolutional neural network (CNN) layer to obtain representations. MLSTM-FCN also uses squeeze-and-excitation blocks to enhance the performance by modeling the interdependencies between the variables. These deep learning-based methods have performed outstandingly in MTSC tasks. In general, the current deep learning-based strategies mainly regard MTS as sequences and design specific neural network layers for sequences to mine the pairwise dependencies among multivariate variables.

Intuitively, graphs could more naturally express complex relationships of nodes than that of sequence. The use of graph models to measure *complex hidden dependencies* among multivariate variables is promising (Wang et al., 2020) owing to the rapid development of graph-based techniques (Zhou et al., 2020). Based on this assumption, we propose to construct graphs from MTS instead of directly utilizing the sequences of MTS. Thus, variables from MTS are constructed as nodes in the converted graph, and they are interlinked through their hidden relations. Graphs are a particular type of information that portrays the relationships between various entities or nodes. Existing mining graph methods, such as graph neural networks (GNNs) (Scarselli, Gori, Tsoi, Hagenbuchner, & Monfardini, 2008) and aggregating feature information of adjoining nodes to learn node or graph representation, have been shown to successfully capture the hidden relations of nodes. For example, GNNs are widely used in community detection (Liu et al., 2020; Su et al., 2021), graph anomaly detection (Ma et al., 2021), and recommendation (Duan et al., 2021) owing to their ability to capture complex graph-based data. Consequently, mining MTS information using graph neural networks can be a promising method to save their temporal patterns while exploiting the interdependency among time-series variables (Wang et al., 2020; Wu et al., 2020; Xu et al., 2020). Therefore, this study proposes the conversion of MTS slices into graphs through graph structure learning and viewing the MTS classification task as a graph classification task.

The graph classification task aims to predict the type of the entire graph, where the graph structure and all of the initial node-level representations are inputs. For example, given a molecule, the task could be to anticipate whether it is poisonous (Ranjan, Sanyal, & Talukdar, 2020). Although current GNN-based spatial–temporal modeling strategies can aggregate the graph structure and node-level representations, they are still inherently flat and cannot aggregate hub data in a hierarchical manner (Ying et al., 2018). Thus they cannot fully capture the hidden features of the MTS when aggregating node-level to graph-level representations. Recently, some hierarchical pooling methods have been proposed and have achieved promising results in many graph classification tasks, such as gPool (Gao & Ji, 2019), DiffPool (Ying et al., 2018), and MemGNN (Khasahmadi, Hassani, Moradi, Lee, & Morris, 2020). These methods design specific hierarchical graph pooling layers, where nodes are recursively aggregated to frame a cluster that addresses a hub in the pooled graph. gPool downsamples by selecting the most important nodes, while DiffPool downsamples by clustering nodes using GNNs. As with DiffPool, MemGNN utilizes a multihead exhibit of memory keys and a convolution operator to sum the soft cluster assignments from various heads and calculate the attention scores among nodes and clusters.

However, we observe that the key process of most hierarchical pooling mechanisms, i.e., the generation of centroids for soft cluster assignments, is not related to the input graphs. This is unexpected because the centroids of the different graphs should be different. MemGNN randomly initializes centroids (keys) before training and makes the centroids learnable during training. The centroids cannot change during testing; therefore, the testing graphs can only use the same centroids as train graphs for cluster assignments. Intuitively, each input graph should have specific centroids based on its topology structure and node features. In general, the pooling mechanism should maintain three significant properties of graphs: (1) *Permutation invariance*: The centroids of the same graph should be invariant when the permutation of the nodes changes. (2) *Input correlation*: Centroids should change if the input graph changes. (3) *Dimensional adjustability*. The dimensions of the centroid matrix should be adjustable according to the dimensions of the input and coarsened graphs.

To this end, we propose a novel MTS classification framework called MTPool (*M*ultivariate *T*ime Series Classification with a Variational Graph *Pool*ing). The goal of MTPool is to model *inner temporal reliance* mode and *complex hidden dependency* relationships among MTS variables and obtain MTS's global representation. Specifically, MTPool first constructs a graph based on MTS data through the graph structure learning module and learns the spatial–temporal features of MTS through the temporal convolution module. Then, to retain the three properties mentioned above, we designed a novel pooling layer, variational pooling, for graph coarsening. This pooling layer contains an "encoder–decoder" architecture, which enables the generation of centroids to be input-related and makes the model more inductive. Next, MTPool adapts the GNNs to jointly learn graph representations, after which the graph is progressively coarsened to one node. Finally, these final coarsened graph-level representations can be used as feature inputs for a differentiable classifier for MTSC tasks. In summary, the fundamental contributions of our study are as follows:

(1) To the best of our knowledge, we first propose a hierarchical graph-pooling-based framework to model MTS and hierarchically generate its global representation for MTS classification.
(2) We design MTPool as an end-to-end joint framework for graph structure learning, temporal convolution, graph representation learning, and graph coarsening.
(3) We propose a novel pooling method, which is called variational pooling. The centroids for the cluster assignments are input-related to the input graphs, making the model more inductive and leading to better performance.
(4) We conduct extensive experiments on MTS benchmark datasets. Experiments on ten benchmark datasets show that MTPool outperforms cutting-edge strategies in the MTSC task.

2. Related work

2.1. Multivariate time-series classification

Most MTSC methods can be classified as distance-based, feature-based, and deep-learning-based approaches. Here, we discuss only deep-learning-based strategies.

Currently, two popular deep learning models, CNNs and recurrent neural networks (RNNs), are widely used for MTS classification. These models, regarding the MTS as sequences, often use an LSTM layer and a stacked CNN layer to extract time-series features, and a softmax layer is then applied to predict the label (Zhang et al., 2020). For example, the MLSTM-FCN (Karim et al., 2019) utilizes an LSTM layer and a stacked CNN layer alongside squeeze-and-excitation blocks to obtain representations. TapNet (Zhang et al., 2020) also constructed an LSTM layer and a stacked CNN layer, followed by an attentional prototype network.

Deep-learning-based methods require less domain knowledge in time-series data than traditional methods. Based on these methods, we propose the use of a graph to model complex hidden

dependencies among multivariate variables. The dependencies are explicitly described by the edges in the graph. Advanced graph techniques also provide a variety of ways to describe the dependencies.

2.2. Graph neural networks

Scarselli et al. (2008) first proposed the idea and the concept of a GNN, which broadened existing neural networks with respect to the handling of the information addressed in graph areas. GNNs follow a local aggregation mechanism, where the embedding vector of a node is processed by recursively aggregating and transforming the embedding vectors of its neighboring nodes (Duan et al., 2021; Xu et al., 2021). Numerous GNN variations have been proposed and have accomplished cutting-edge results for both node and graph classification assignments (Wang, Duan, Liao, Wu, & Zhuang, 2019). For example, a graph convolutional network (GCN) (Kipf & Welling, 2016) can be viewed as an approximation of the spectral-domain convolution of graph data. GraphSAGE (Hamilton, Ying, & Leskovec, 2017) and FastGCN (Chen, Ma, & Xiao, 2018) sample and aggregate the neighborhood information while enabling training in batches yet forfeiting some time-proficiency. Graph attention networks (GATs) (Veličković et al., 2017) are used to design a new method of gathering neighbors through self-attention. Subsequently, a graph isomorphism network (GIN) (Xu, Hu, Leskovec, & Jegelka, 2018) and k-GNNs (Morris et al., 2019) were developed, presenting more complex and different types of aggregation.

2.3. Graph pooling

Graph pooling strategies can be classified as topology-based, global, and hierarchical pooling. Here, we only discuss hierarchical pooling methods. DiffPool (Ying et al., 2018) trains two parallel GNNs to obtain the node-level representations and cluster assignments. gPool (Gao & Ji, 2019) and SAGPool (Lee, Lee, & Kang, 2019) dropped nodes from the input graph as opposed to bunching various nodes to frame a cluster in the pooled graph. They devised a top-K node choice method to create an initiated subgraph for subsequent layers. Although they are more efficient than DiffPool, they do not gather nodes or calculate soft-edge weights. This makes it difficult for them to preserve node and edge information effectively. MemGNN (Khasahmadi et al., 2020) are soft cluster assignments, and they utilize a clustering-friendly distribution to determine the attention scores among nodes and clusters. However, they generate centroids (which are used to calculate soft assignments) without involving input graphs, which is not logical intuitively because the centroids for different input graphs should be different. We propose variational pooling using an "encoder–decoder" architecture to obtain centroids to address this limitation. It can maintain permutation invariance while inputting centroids to graphs (Xu et al., 2020).

3. Framework

In this section, we present the proposed MTPool in detail. MTPool can be divided into four parts: graph structure learning, temporal convolution, spatial–temporal modeling, and variational graph pooling. The schematic of MTPool is shown in Fig. 1.

3.1. Problem formulation

An MTS can be represented as a matrix $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\} \in \mathbb{R}^{n\times T}$, where T is the length of the time series, and n is the number of MTS variables. Each MTS is associated with a class label y from a predefined label set. Given a group of MTS $\boldsymbol{\mathcal{X}} = \{\boldsymbol{X_1}, \boldsymbol{X_2}, \ldots, \boldsymbol{X_N}\} \in \mathbb{R}^{N\times n\times T}$, where N is the number of MTS slices in the group, and the corresponding labels $\mathcal{Y} = \{y_1, y_2, \ldots, y_N\} \in \mathbb{R}^N$, the research goal is to learn the mapping relationship between $\boldsymbol{\mathcal{X}}$ and $\mathcal{Y}$ based on the proposed model.

3.2. Graph structure learning

We propose a adaptive relation embedding strategy that adaptively learns the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ to model complex hidden dependency relationships in the MTS sample $\boldsymbol{X}$. The learned graph structure (adaptive adjacency matrix) $\boldsymbol{A}$ is defined as

$$\boldsymbol{A} = Embed_1(\boldsymbol{X}), \tag{1}$$

where $Embed_1()$ represents the graph structure learning function. For the input MTS, $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\} \in \mathbb{R}^{n\times T}$, we first calculate the similarity matrix $\boldsymbol{C}$ between sampled MTS variables:

$$C_{ij} = \frac{exp\left(-\sigma(distance(\boldsymbol{x_i}, \boldsymbol{x_j}))\right)}{\sum_{p=1}^{n} exp\left(-\sigma(distance(\boldsymbol{x_i}, \boldsymbol{x_p}))\right)}, \tag{2}$$

where $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ denote the ith and jth MTS variables ($i, j = 1, 2, \ldots, n$), and *distance* denotes a distance metric such as the Euclidean distance, absolute value distance, and dynamic time warping. Then, the adaptive adjacency matrix $\boldsymbol{A}$ can be calculated as:

$$\boldsymbol{A} = \sigma(\boldsymbol{C}\boldsymbol{W_{adj}}), \tag{3}$$

where σ is an activation function and $\boldsymbol{W_{adj}} \in \mathbb{R}^{n\times n}$ are learnable model parameters that adaptively generate an adjacency matrix based on the input features. Moreover, to improve the training efficiency, reduce the noise impact, and make the model more robust, the threshold c_1 is set to make the adjacency matrix sparse:

$$A_{ij} = \begin{cases} A_{ij}, & A_{ij} \geq c_1. \\ 0, & A_{ij} < c_1. \end{cases} \tag{4}$$

Finally, row normalization is applied to $\mathbf{A}$.

3.3. Temporal convolution

This stage aims to extract temporal features that are associated with or change over time, as well as to construct a feature matrix $\boldsymbol{X_{TC}} \in \mathbb{R}^{n\times d}$, where d is the obtained feature dimension. With temporal convolution, we can obtain each MTS feature matrix as follows:

$$\boldsymbol{X_{TC}} = Embed_2(\boldsymbol{X}), \tag{5}$$

where $Embed_2()$ is the temporal convolution function. When investigating a time series, it is essential to consider its numerical value as well as its pattern over the long haul. Time series in the real world typically have numerous concurrent periodicities. For example, the number of inhabitants in a specific city shows a particular pattern each day. However, a significant example can be seen by observing it over multiple weeks or one month. In this manner, it is important to separate the highlights of the time arrangement into units of numerous periods. To mimic the present circumstances, we utilize numerous CNN channels with various responsive fields, namely kernel sizes, to extract features at multiple time scales.

For the ith CNN filter (the number of CNN filters is q and $i = 1, 2, \ldots, q$), given the input time series $\boldsymbol{X}$, the feature vector $\boldsymbol{f_i}$ is expressed as follows: $\boldsymbol{f_i} = \sigma(\boldsymbol{W_i} * \boldsymbol{X} + \boldsymbol{b})$, where $*$ denotes the convolution operation, σ is a nonlinear activation function, such as $RELU(x) = max(0, x)$, $\boldsymbol{W_i} \in \mathbb{R}^{1\times ks}$ represents the ith CNN kernel, ks is the kernel size, and $\boldsymbol{b}$ is the bias. The final feature vector can be expressed as $\boldsymbol{X_{TC}} = \left(\mathop{\Vert}_{i=1}^{|q|} \boldsymbol{f_i}\right)$, where $\mathop{\Vert}_{i=1}^{|q|}$ is the concatenation operation from feature $\boldsymbol{f_1}$ to feature $\boldsymbol{f_q}$.

In this manner, temporal features under various periods are extracted, providing a powerful reference for time-series classification.

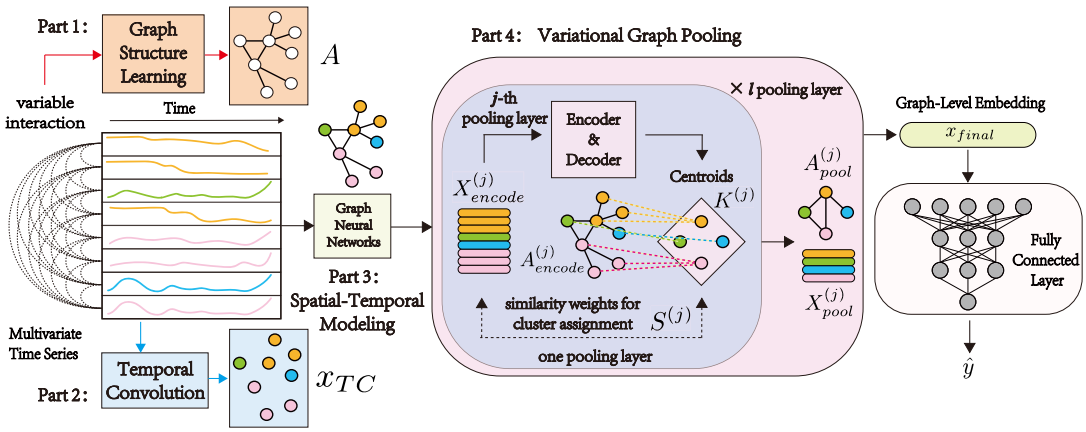


Fig. 1. The general architecture of MTPool. The adjacency and feature matrices are constructed through graph structure learning and temporal convolution, respectively. Then, GNNs aggregate and fuse the spatial–temporal features. After several pooling layers, the input graphs were hierarchically coarsened to one node to attain their graph-level representations. Finally, these final output graph-level embeddings can be used as feature inputs for a differentiable classifier for MTSC tasks. MTPool is an end-to-end joint framework for graph structure learning, temporal convolution, graph representation learning, and graph coarsening.

3.4. Spatial–temporal modeling

In this stage, m GNN layers $(G_1, G_2, \ldots, G_m)$ are employed on the input graphs (denoted as $\boldsymbol{X_{TC}}, \boldsymbol{A}$) for spatial–temporal modeling. GNN layers can fuse spatial dependencies and temporal patterns to embed the features of nodes and transform the feature dimension of nodes to d_{encode}, as shown in Eq. (6):

$$\boldsymbol{Z} = G_m(G_{m-1}(\ldots G_1(\boldsymbol{X_{TC}}, \boldsymbol{A})\ldots)), \tag{6}$$

where $\boldsymbol{Z} \in \mathbb{R}^{n \times d_{encode}}$ represents the learned node embedding from the spatial–temporal modeling. G_j $(j = 1, 2, \ldots, m)$ consists of a GNN layer and a batch normalization layer. The GNN can be GCN (Defferrard, Bresson, & Vandergheynst, 2016), GAT (Veličković et al., 2017), and GIN (Xu et al., 2018). Inspired by the k-GNN (Morris et al., 2019) model, this study uses the following propagation mechanism to calculate the forward-pass update of a node denoted by $\boldsymbol{v_i}$:

$$\boldsymbol{z_i^{(j+1)}} = \sigma\left(\boldsymbol{z_i^{(j)}}\boldsymbol{W_1^{(j)}} + \sum_{r \in \boldsymbol{N(i)}} \boldsymbol{z_r^{(j)}}\boldsymbol{W_2^{(j)}}\right) \tag{7}$$

where $\boldsymbol{W_1^{(j)}}$ and $\boldsymbol{W_2^{(j)}}$ are parameter matrices of the jth GNN layer, $\boldsymbol{z_i^{(j)}}$ is the hidden state of node $\boldsymbol{v_i}$ in the jth layer, and $\boldsymbol{N(i)}$ denotes the neighbors of node i. K-GNNs only perform information fusion between a specific node and its neighbors, ignoring the information of other non-neighbor nodes. This design highlights the relationships among variables, effectively avoiding information redundancy caused by high dimensions.

3.5. Variational graph pooling

3.5.1. Overall transformation

In this stage, the encoded graph is hierarchically pooled into a single node to generate a graph-level representation. This successive process can be expressed as:

$$\boldsymbol{x_{final}} = P_l(P_{l-1}(\ldots P_1(\boldsymbol{Z}, \boldsymbol{A})\ldots)), \tag{8}$$

where P_j $(j = 1, 2, \ldots, l)$ represents one pooling layer. After stacking l pooling layers, we obtain the graph-level representation vector $\boldsymbol{x_{final}}$. For the jth pooling layer, $P_j()$ can pool each encoded graph into a specific coarsened graph. The overall transformation of $P_j()$ is given by Eq. (9):

$$\begin{aligned} \boldsymbol{X_{pool}^{(j)}} &= \sigma\left(\boldsymbol{S^{(j)}}\boldsymbol{X_{encode}^{(j)}}\boldsymbol{W_{pool}^{(j)}}\right), \\ \boldsymbol{A_{pool}^{(j)}} &= \sigma\left(\boldsymbol{S^{(j)}}\boldsymbol{A_{encode}^{(j)}}(\boldsymbol{S^{(j)}})^{\boldsymbol{T}}\right), \end{aligned} \tag{9}$$

where σ is a nonlinear activation function, $\boldsymbol{W_{pool}^{(j)}} \in \mathbb{R}^{d_{encode}^{(j)} \times d_{pool}^{(j)}}$ is a trainable parameter matrix that represents a linear transformation, and $\boldsymbol{S^{(j)}} \in \mathbb{R}^{n_{pool}^{(j)} \times n_{encode}^{(j)}}$ is an assignment matrix representing a projection from the original nodes to pooled nodes (clusters). The details of calculating $S^{(j)}$ are presented in the next Section 3.5.2. $\boldsymbol{X_{encode}^{(j)}} \in \mathbb{R}^{n_{encode}^{(j)} \times d_{encode}^{(j)}}$ and $\boldsymbol{A_{encode}^{(j)}} \in \mathbb{R}^{n_{encode}^{(j)} \times n_{encode}^{(j)}}$ represent the encoded feature and adjacency matrix of the input graph in the jth pooling layer; $\boldsymbol{X_{pool}^{(j)}} \in \mathbb{R}^{n_{pool}^{(j)} \times d_{pool}^{(j)}}$ and $\boldsymbol{A_{pool}^{(j)}} \in \mathbb{R}^{n_{pool}^{(j)} \times n_{pool}^{(j)}}$ represent the pooled feature and adjacency matrix of the pooled graph in the jth pooling layer. In most cases, the pooled graph contains fewer nodes than the input graph ($n_{pool}^{(j)} < n_{encode}^{(j)}$). For the first pooling layer, we have $\boldsymbol{X_{encode}^{(0)}} = \boldsymbol{Z}$ and $\boldsymbol{A_{encode}^{(0)}} = \boldsymbol{A}$.

3.5.2. Computation of assignment matrix

In this section, we propose a new pooling method, variational pooling, to address the limitations of existing methods, as discussed in Section 2.3, and to calculate $\boldsymbol{S^{(j)}}$ in a more effective way. Although many advanced graph pooling methods have been proposed in recent years, they all have some limitations. For example, MemGNN (Khasahmadi et al., 2020) utilizes a multihead exhibit of memory keys and a convolution operator to sum the soft cluster assignments from various heads. MemGNN utilizes a clustering-friendly distribution to determine the attention scores among nodes and clusters and performs better than DiffPool in many tasks. Nevertheless, we observe that it generates memory heads, which represent the new centroids in the space of pooled graphs, without the involvement of the input graphs. However, the centroids for different graphs should be different, and each input graph should have its corresponding centroids based on its topology structure and node features. To address the limitations mentioned above, we first generate h batches of centroids $\boldsymbol{K^{(j)}} = [\boldsymbol{K_1^{(j)}}, \boldsymbol{K_2^{(j)}}, \ldots, \boldsymbol{K_h^{(j)}}]^T \in \mathbb{R}^{h \times n_{pool}^{(j)} \times d_{encode}^{(j)}}$ based on the input graph in $P_j()$, and we then compute and aggregate the relationship between every batch of centroids and the encoded graph for the assignment matrix $\boldsymbol{S^{(j)}}$.

3.5.3. Generation of centroids

The generation of centroids $\boldsymbol{K^{(j)}}$ should satisfy the following properties:

(1) **Permutation invariance.** The same graph can be addressed using various adjacency matrices by permuting the sequences of the nodes. The centroids of the same graph should be invariant to such changes.
(2) **Input correlation.** The generation of centroids is based on an input graph. If the input graph changes, the centroids should change accordingly to effectively capture global features.
(3) **Dimensional adjustability.** The dimension of the centroid matrix $\boldsymbol{K}^{(j)}$ should be adjusted according to the dimensions of the input and output coarsened graphs.

Therefore, we propose an "encoder–decoder" architecture to compute the centroid matrix $\boldsymbol{K}^{(j)}$. In general, the *encoder* ensures the property of permutation invariance and makes the centroids adaptive to input graphs, whereas the *decoder* controls the dimension of the output coarsened graph:

$$\boldsymbol{K}^{(j)} = Decoder\left(Encoder\left(\boldsymbol{X}^{(j)}_{\boldsymbol{encode}}\right)\right). \tag{10}$$

As shown in Eq. (10), an *encoder* is deployed over the input graph, transforming $\boldsymbol{X}^{(j)}_{\boldsymbol{encode}} \in \mathbb{R}^{n^{(j)}_{encode}\times d^{(j)}_{encode}}$ into $\boldsymbol{X}^{(j)}_{\boldsymbol{g}} \in \mathbb{R}^{1\times d^{(j)}_{encode}}$: Here, $\boldsymbol{X}^{(j)}_{\boldsymbol{g}}$ is the feature that incorporates the input information. Subsequently, a *decoder* is applied to map $\boldsymbol{X}^{(j)}_{\boldsymbol{g}}$ to $\boldsymbol{K}^{(j)} \in \mathbb{R}^{(h\times n^{(j)}_{pool})\times d^{(j)}_{encode}}$, and $\boldsymbol{K}^{(j)}$ is then reshaped to $\mathbb{R}^{h\times n^{(j)}_{pool}\times d^{(j)}_{encode}}$. Referring to Bai et al. (2019), the expressions for the encoder and decoder used in this study are as follows:

$$\begin{aligned} Encoder: \boldsymbol{X}^{(j)}_{\boldsymbol{g}} &= \sum_{i=1}^{n} \sigma\left((\boldsymbol{u}^{(j)}_{\boldsymbol{i}})^{T}\boldsymbol{x}^{(j)}_{\boldsymbol{avg}}\boldsymbol{u}^{(j)}_{\boldsymbol{i}}\right) \\ &= \sum_{i=1}^{n} \sigma_1\left((\boldsymbol{u}^{(j)}_{\boldsymbol{i}})^{T}\sigma_2\left(\left(\frac{1}{n}\sum_{j=1}^{n}\boldsymbol{u}^{(j)}_{\boldsymbol{j}}\right)\boldsymbol{W}^{(j)}_{\boldsymbol{avg}}\right)\boldsymbol{u}^{(j)}_{\boldsymbol{i}}\right), \\ Decoder: MLP, \end{aligned} \tag{11}$$

where $\boldsymbol{u}^{(j)}_{\boldsymbol{i}}$ is the embedding of node i of the input graph in the jth pooling layer, σ_1 is the *sigmoid* function, σ_2 is the *tanh* activation function, and $\boldsymbol{W}^{(j)}_{\boldsymbol{avg}} \in \mathbb{R}^{d^{(j)}_{encode}\times d^{(j)}_{encode}}$ is the learnable weight matrix. For the encoder, we used the attention mechanism to guide the model to learn weights under specific tasks in order to generate a graph-level representation $\boldsymbol{X}^{(j)}_{\boldsymbol{g}}$ for centroids. For the decoder, we used a multilayer perceptron (MLP) to reshape matrix $\boldsymbol{K}^{(j)}$ to the required size.

3.5.4. Computation of assignment matrix

With centroid matrix $\boldsymbol{K}^{(j)}$, we can compute the relationship $\boldsymbol{S}^{(j)}_{\boldsymbol{p}} \in \mathbb{R}^{n^{(j)}_{pool}\times n^{(j)}_{encode}}$ $(p = 1, 2, \ldots, h)$ between centroids $\boldsymbol{K}^{(j)}_{\boldsymbol{p}} \in \mathbb{R}^{n^{(j)}_{pool}\times d^{(j)}_{encode}}$ $(p = 1, 2, \ldots, h)$ and $\boldsymbol{X}^{(j)}_{\boldsymbol{encode}} \in \mathbb{R}^{n^{(j)}_{encode}\times d^{(j)}_{encode}}$ in the jth pooling layer. We use the cosine similarity to evaluate the relationship between input node embeddings and centroids, as described in Eq. (12), followed by a row normalization deployed in the resulting assignment matrix:

$$\begin{aligned} (\boldsymbol{S}^{(j')}_{\boldsymbol{p}})' &= \text{cosine}\left(\boldsymbol{K}^{(j)}_{\boldsymbol{p}}, \boldsymbol{X}^{(j)}_{\boldsymbol{encode}}\right), \\ \boldsymbol{S}^{(j)}_{\boldsymbol{p}} &= \text{normalize}_{\text{row}}\left((\boldsymbol{S}^{(j')}_{\boldsymbol{p}})'\right), \end{aligned} \tag{12}$$

where $(\boldsymbol{S}^{(j')}_{\boldsymbol{p}})' \in \mathbb{R}^{n_{pool(j)}\times n^{(j)}_{encode}}$ $(p = 1, 2, \ldots, h)$ denotes the assignment matrix before normalization. Finally, we aggregate the information pertaining to the h relationship $\boldsymbol{S}^{(j)}_{\boldsymbol{p}}$:

$$\boldsymbol{S}^{(j)} = \Gamma_{\phi}\left(\Big\|_{p=1}^{|h|}\boldsymbol{S}^{(j)}_{\boldsymbol{p}}\right). \tag{13}$$

In Eq. (13), we concatenate $\boldsymbol{S}^{(j)}_{\boldsymbol{p}}$ $(p = 1, 2, \ldots, h)$ and apply a trainable weighted sum Γ_{ϕ} to the concatenated matrix, leading to the final assignment matrix $\boldsymbol{S}^{(j)}$.

3.6. Differentiable classifier

In this stage, the graph-level embedding vector $\boldsymbol{x}_{\boldsymbol{final}}$ is applied to a specific predicted class number $\hat{y}$. A standard *MLP* is used to transform the dimension of the graph-level embedding into the number of classes. Then, we compare the predicted class number against the ground-truth label. The loss function is as follows:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j} log\hat{y}_{i,j}, \tag{14}$$

where N is the set of training samples, M is the number of classes, y is the true label, and $\hat{y}$ is the value predicted by the model. The whole algorithm framework is shown in Algorithm 1.

3.7. Time complexity analysis

In this subsection, we analyze the time complexity of MTPool, and the real-time consumption is presented in Section 4.2.2. Given a multivariate time series $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\} \in \mathbb{R}^{n\times T}$, where T is the length of the time series and n is the number of MTS variables.

First, for the graph structure learning module, we require $O(n \times n \times T)$ to compute the similarity matrix C among the MTS variables. To adaptively generate an adjacency matrix based on the input features, we design a learnable matrix $\boldsymbol{W_{adj}}$, which introduces $O(n\times n\times n)$ time complexity. Thus, the graph structure learning module requires $O\big(n \times n \times (T + n)\big)$. For the temporal convolution module, there are $O(q \times n \times T)$ operations, where q denotes the number of CNN filters. Suppose that there are m GNN layers in the spatial–temporal modeling part, we need $O\Big(m \times (n \times n \times T + n \times T \times d_{encode})\Big)$ operations to combine spatial dependencies and temporal patterns, where d_{encode} is the final feature dimension. Considering that q, m, and d_{encode} are set to small numbers in our experiment, and T is often larger than n, we can conclude that the total time complexity before the variational graph pooling module is $O(n \times n \times T)$.

For the variational graph pooling module, we first show the time complexity of a specific layer. Suppose that for the jth pooling layer, $P_j()$ can pool each encoded graph $\boldsymbol{X}^{(j)}_{\boldsymbol{encode}} \in \mathbb{R}^{n^{(j)}_{encode}\times d^{(j)}_{encode}}$, $\boldsymbol{A}^{(j)}_{\boldsymbol{encode}} \in \mathbb{R}^{n^{(j)}_{encode}\times n^{(j)}_{encode}}$, to the coarsened graph $\boldsymbol{X}^{(j)}_{\boldsymbol{pool}} \in \mathbb{R}^{n^{(j)}_{pool}\times d^{(j)}_{pool}}$, $\boldsymbol{A}^{(j)}_{\boldsymbol{pool}} \in \mathbb{R}^{n^{(j)}_{pool}\times n^{(j)}_{pool}}$. Assume that we already obtained the assignment matrix $\boldsymbol{S}^{(j)} \in \mathbb{R}^{n^{(j)}_{pool}\times n^{(j)}_{encode}}$, we need $O\Big((n^{(j)}_{pool} \times d^{(j)}_{encode} \times (n^{(j)}_{encode} + d^{(j)}_{pool}) + n^{(j)}_{encode} \times n^{(j)}_{pool} \times (n^{(j)}_{encode} + n^{(j)}_{pool}))\Big)$ to generate $\boldsymbol{X}^{(j)}_{\boldsymbol{pool}}$ and $\boldsymbol{A}^{(j)}_{\boldsymbol{pool}}$. Prior to obtaining the assignment matrix $\boldsymbol{S}^{(j)}$, it is necessary to compute the h batches of centroids $\boldsymbol{K}^{(j)} = [\boldsymbol{K}^{(j)}_{\boldsymbol{1}}, \boldsymbol{K}^{(j)}_{\boldsymbol{2}}, \ldots, \boldsymbol{K}^{(j)}_{\boldsymbol{h}}]^{T} \in \mathbb{R}^{h\times n^{(j)}_{pool}\times d^{(j)}_{encode}}$. To generate centroids, the encoder needs $O(n^{(j)}_{encode} \times d^{(j)}_{encode} \times d^{(j)}_{encode})$, and the decoder takes $O(h \times n^{(j)}_{pool} \times d^{(j)}_{encode})$. Then, computing the relationship $\boldsymbol{S}^{(j)}_{\boldsymbol{p}} \in \mathbb{R}^{n^{(j)}_{pool}\times n^{(j)}_{encode}}$ $(p = 1, 2, \ldots, h)$ between centroids $\boldsymbol{K}^{(j)}_{\boldsymbol{p}} \in \mathbb{R}^{n^{(j)}_{pool}\times d^{(j)}_{encode}}$ $(p = 1, 2, \ldots, h)$ and $\boldsymbol{X}^{(j)}_{\boldsymbol{encode}} \in \mathbb{R}^{n^{(j)}_{encode}\times d^{(j)}_{encode}}$ in the jth pooling layer requires $O(h \times n^{(j)}_{pool} \times n^{(j)}_{encode} \times d^{(j)}_{encode})$. From this analysis, we can see that the centroid-generating part that we designed has less time complexity than the pooling operation after obtaining the assignment matrix $\boldsymbol{S}^{(j)}$, which occupies a dominant position. Therefore, if we choose

Algorithm 1: MTPool algorithm framework.

Input : The multivariate time-series dataset of N MTS slices, $\boldsymbol{\mathcal{X}} = \{\boldsymbol{X_1}, \boldsymbol{X_2}, ..., \boldsymbol{X_N}\} \in \mathbb{R}^{N\times n\times T}$;
Output: The corresponding predict labels, $\hat{\mathcal{Y}} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_N\} \in \mathbb{R}^N$

$\hat{\mathcal{Y}} = \{\}$; // Record predict labels of MTS slices
for $\boldsymbol{X_k}$ *in* $\{\boldsymbol{X_1}, \boldsymbol{X_2}, ..., \boldsymbol{X_N}\}, \boldsymbol{X_k} \in \mathbb{R}^{n\times T}$ **do**
  for $\boldsymbol{x_i}$ *in* $\boldsymbol{X_k} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}, \boldsymbol{x_i} \in \mathbb{R}^T$ **do**
    for $\boldsymbol{x_j}$ *in* $\boldsymbol{X_k} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}, \boldsymbol{x_j} \in \mathbb{R}^T$ **do**
      $C_{ij} = \frac{exp(-\sigma(distance(\boldsymbol{x_i}, \boldsymbol{x_j})))}{\sum_{p=0}^{n} exp(-\sigma(distance(\boldsymbol{x_i}, \boldsymbol{x_p})))}$ // Calculating the similarity matrix
    end
  end
  $\boldsymbol{A} = \sigma(\boldsymbol{C}\boldsymbol{W_{adj}})$; // Calculating the adaptive adjacency matrix
  $\boldsymbol{A} = Normalize_{row}(Threshold(\boldsymbol{A}))$; // Graph structure learning
  $\boldsymbol{X_{TC}} = Temporal\_Convolution(\boldsymbol{X_k}) \in \mathbb{R}^{n\times d}$; // Temporal convolution
  $\boldsymbol{Z} = G_m(G_{m-1}(...G_1(\boldsymbol{X_{TC}}, \boldsymbol{A})...))$; // Spatial-temporal modeling
  $\boldsymbol{X_{encode}^{(0)}} = \boldsymbol{Z}, \boldsymbol{A_{encode}^{(0)}} = \boldsymbol{A}$; // The input of the first pooling layer
  for P_j *in Pooling_Layers* $= \{P_1, P_2, ..., P_l\}$ **do**
    $\boldsymbol{K^{(j)}} = [\boldsymbol{K_1^{(j)}}, \boldsymbol{K_2^{(j)}}, ..., \boldsymbol{K_h^{(j)}}]^T = decoder\left(encoder\left(\boldsymbol{X_{encode}^{(j)}}\right)\right)$, $Encoder : \boldsymbol{X_g^{(j)}} = \sum_{i=1}^{n} \sigma_1\left((\boldsymbol{u_i^{(j)}})^T \sigma_2((\frac{1}{n}\sum_{j=1}^{n} \boldsymbol{u_j^{(j)}})\boldsymbol{W_{avg}^{(j)}})\boldsymbol{u_i^{(j)}}\right)$,
    $Decoder : MLP$;
    // Generating centroids
    $\boldsymbol{S^{(j)}} = \Gamma_\phi\left(\Big\|_{p=1}^{|h|} \boldsymbol{S_p^{(j)}}\right)$, and $\boldsymbol{S_p^{(j)}} = Normalize_{row}\left(Cosine\left(\boldsymbol{K_p^{(j)}}, \boldsymbol{X_{encode}^{(j)}}\right)\right)$. $p = 1, 2, ..., h$;
    // Computing assignment matrix
    $\boldsymbol{X_{pool}^{(j)}} = \sigma\left(\boldsymbol{S^{(j)}}\boldsymbol{X_{encode}^{(j)}}\boldsymbol{W_{pool}^{(j)}}\right)$;
    $\boldsymbol{A_{pool}^{(j)}} = \sigma\left(\boldsymbol{S^{(j)}}\boldsymbol{A_{encode}^{(j)}}(\boldsymbol{S^{(j)}})^T\right)$;
    // Variational graph pooling
    $\boldsymbol{X_{encode}^{(j+1)}} = \boldsymbol{X_{pool}^{(j)}}, \boldsymbol{A_{encode}^{(j+1)}} = \boldsymbol{A_{pool}^{(j)}}$; // Obtain the input of the next pooling layer
  end
  $\boldsymbol{x_{final}} = \boldsymbol{X_{pool}^{(j)}}$;
  $\hat{y}_k = MLP(\boldsymbol{x_{final}})$;
  $\hat{\mathcal{Y}} \leftarrow \hat{y}_k$;
  // Add $\hat{y}_k$ to $\hat{\mathcal{Y}}$
end
return $\hat{\mathcal{Y}}$

Table 1
Summary of the 10 UEA datasets used in experiments.

	Name	Train size	Test size	Num series	Series length	Classes
AF	AtrialFibrillation	15	15	2	640	3
FM	FingerMovements	316	100	28	50	2
HMD	HandMovementDirection	160	74	10	400	4
HB	Heartbeat	204	205	61	405	2
LIB	Libras	180	180	2	45	15
MI	MotorImagery	278	100	64	3000	2
NATO	NATOPS	180	180	24	51	6
PD	PenDigits	7494	3498	2	8	10
SRS2	SelfRegulationSCP2	200	180	7	1152	2
SWJ	StandWalkJump	12	15	4	2500	3

small values of pooling layers l, d_{encode}, and d_{pool}, the time complexity of the variational graph pooling module is $O(n \times n \times n)$, where n is the number of MTS variables. Considering that the time-series length T is usually larger than the number of MTS variables n, the total time complexity of MTPool is $O(n \times n \times T)$.

4. Experiments

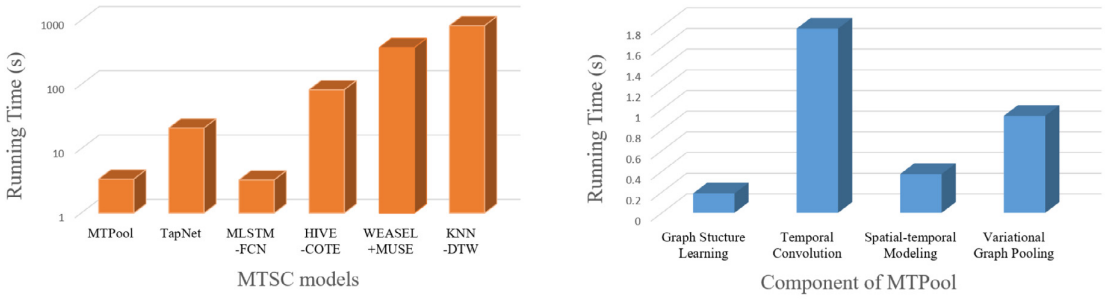
In this section, we provide a comprehensive analysis of 10 benchmark datasets for MTS classification and compare the evaluation results of our model (MTPool) with other baselines.

4.1. Experiment settings

4.1.1. Datasets

We use 10 publicly available benchmark datasets from the UEA MTS classification archive.[2] The main characteristics of each dataset are summarized in Table 1. The train and test sizes represent the number of MTS slices in the training and test datasets,

[2] Datasets are available at http://timeseriesclassification.com. We excluded data with extremely long lengths, unequal lengths, high-dimensional sizes, and in-balance splits.



(a) Time consumption for different MTSC models

(b) Time consumption for each component of MTPool

Fig. 2. The real time consumption of the Heartbeat dataset. We use the log-scale running time in Fig. 2(a) for the better comparison of different models.

respectively. Num series refers to the number of variables in each MTS slice; the series length refers to the length or feature dimension of each variable in each MTS slice. The class refers to the number of types of MTS slices.

4.1.2. Metrics

Similar to other MTS classification methods (Zhang et al., 2020), we used the classification accuracy as an evaluation metric:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{15}$$

where TP, TN, FP, and FN denote the true positive, true negative, false positive, and false negative, respectively.

4.1.3. Methods for comparison

We used the following implementations of the MTS classifiers, including the common distance-based classifiers, latest bag-of-patterns model, and deep learning models

(1) **ED, DTW$_I$, DTW$_D$, - with and without normalization (norm)** (Bagnall et al., 2018) are the common distance-based models. 1-Nearest Neighbor with distance functions includes Euclidean (ED), dimension-independent dynamic time warping (DTW$_I$), and dimension-dependent dynamic time warping (DTW$_D$).
(2) **WEASEL+MUSE** (Schäfer & Leser, 2017) represents the Word ExtrAction for time SEries cLassification (WEASEL) with a Multivariate Unsupervised Symbols and dErivatives (MUSE). It is the most effective bag-of-pattern algorithm for MTSC.
(3) **HIVE-COTE** (Bagnall, Flynn, Large, Lines, & Middlehurst, 2020) is a heterogeneous meta-ensemble for time-series classification. This approach is a good baseline for assessing bespoke MTSC.
(4) **MLSTM-FCN** (Karim et al., 2019) is a famous deep-learning framework for MTSC. It utilizes an LSTM layer and a stacked CNN layer alongside squeeze-and-excitation blocks to obtain the representations.
(5) **TapNet** (Zhang et al., 2020) draws on the strengths of both traditional and deep learning approaches. It also constructs an LSTM layer and a stacked CNN layer, followed by an attentional prototype network.
(6) **MTPool** is the MTPool framework obtained with our proposed variational pooling and adaptive adjacency matrix.

4.1.4. Training details

All of the networks were implemented using Pytorch 1.4.0 in Python 3.6.2 and trained with 10 000 epochs (computing infrastructure: Ubuntu 18.04 operating system, GPU NVIDIA GeForce RTX 2080 Ti with 8 GB GRAM and 32 GB of RAM). We used {3, 5, 7} as the convolutional kernel size, and 10 as the channel number. The threshold to make the adjacency matrix sparse was chosen from {0.05, 0.1, 0.2}, and the output dimension of the GNNs layer was 128. For the pooling layers, the heads of centroids are chosen from {1, 2, 4}, the reduction factor is chosen from {2, 3, 6}, and the number of nodes in the final pooling layer is 1. The initial learning rate was 10^{-4}, and the categorical cross-entropy loss and Adam optimization were used to optimize the parameters of our models. For each method, we performed 10 times and reported the best accuracy.

4.2. Main results

4.2.1. Effectiveness

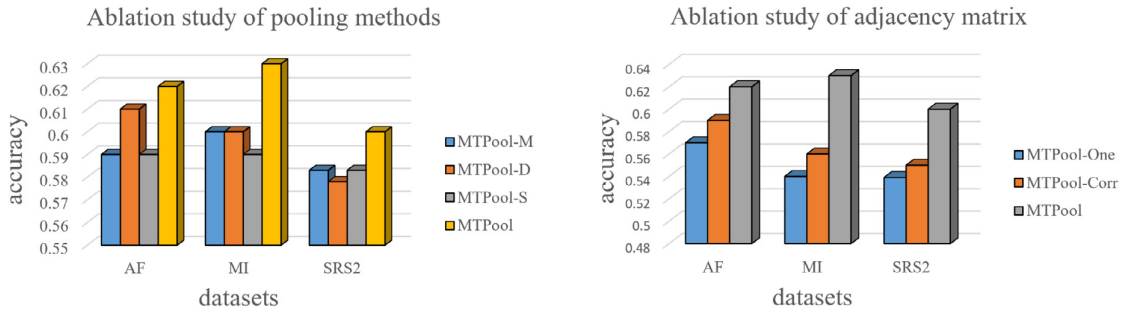
Table 2 shows the accuracy results obtained for the selected 10 UEA datasets of the MTPool and other MTS classifiers. "Avg. Rank" in the table refers to the average ranking of different models under all datasets, and "Wins/Ties" indicates the number of datasets for which this model achieves the best performance. For some approaches that ran out of memory, we referred to Ruiz, Flynn, and Bagnall (2020) for the corresponding results. The best performance in each dataset is shown in bold.

First, we can observe that MTPool outperforms several other advanced MTS classification methods (such as TapNet or MLSTM-FCN) on eight datasets. The "Avg. Rank" indicates the superiority of MTPool over the existing state-of-the-art models (the second-highest ranked model, MLSTM-FCN, has an average ranking of 4.40). We also found that methods that are based on deep learning proposed in recent years, such as TapNet and MLSTM-FCN, perform better than methods based on nearest neighbors, proving the effectiveness of deep learning in extracting MTS features.

More importantly, with different datasets, the number of variables and lengths span an extensive range. For instance, the number of variables ranged from 2 to 64, minimum length of the MTS was eight, and maximum was 300. In the case of variable dataset parameters, our framework maintains considerable competitiveness, thereby proving robustness to our model. More specifically, for a large number of variables, MTPool can handle this by increasing the pooling layers and making the graph pooling process more hierarchical, thereby performing well. For instance, MTPool achieved the best and second-best accuracies for MotorImagery (64 variables) and Heartbeat (61 variables). This means that when the number of variables is relatively large, the hierarchical pooling framework that we provide can better capture the graph's structure and generate time-series embeddings with more robust characterization capabilities. Notably, when the number of variables is relatively small, by reducing the number of pooling layers, MTPool can also achieve good accuracy. MTPool achieves the best accuracy on PenDigits (two variables), SelfRegulationSCP2

Table 2
Accuracy of the 11 algorithms on the default training/test datasets of 10 selected UEA MTSC archives. The best performance is shown in bold.

Methods/Datasets	AF	FM	HMD	HB	LIB	MI	NATO	PD	SRS2	SWJ	Avg. Rank	Win/Ties
ED	0.267	0.519	0.279	0.620	0.833	0.510	0.850	0.973	0.483	0.333	7.50	0
DTW_I	0.267	0.513	0.297	0.659	0.894	0.390	0.850	0.939	0.533	0.200	7.70	0
DTW_D	0.267	0.529	0.231	0.717	0.872	0.500	0.883	0.977	0.539	0.200	6.50	0
ED (norm)	0.200	0.510	0.278	0.619	0.833	0.510	0.850	0.973	0.483	0.333	8.25	0
DTW_I (norm)	0.267	0.520	0.297	0.658	0.894	0.390	0.850	0.939	0.533	0.200	7.60	0
DTW_D (norm)	0.267	0.530	0.231	0.717	0.870	0.500	0.883	0.977	0.539	0.200	6.50	0
WEASEL+MUSE	0.400	0.550	0.365	0.727	0.894	0.500	0.870	0.948	0.460	0.267	5.65	0
HIVE-COTE	0.133	0.550	0.446	0.722	**0.900**	0.610	0.889	0.934	0.461	0.333	5.40	1
MLSTM-FCN	0.333	0.580	**0.527**	0.663	0.850	0.510	0.900	0.978	0.472	0.400	4.40	1
TapNet	0.200	0.470	0.338	**0.751**	0.878	0.590	0.939	0.980	0.550	0.133	5.25	1
MTPool	**0.533**	**0.620**	0.486	0.742	**0.900**	**0.630**	**0.944**	**0.983**	**0.600**	**0.667**	**1.25**	**8**



(a) Ablation study of pooling methods

(b) Ablation study of adjacency matrix

Fig. 3. Ablation study of pooling methods and adjacency matrix. The AF, MI, and SRS2 datasets were used, and different colors represent different methods. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

(seven variables), and StandWalkJump (four variables) datasets. A more detailed analysis of MTPool is provided in the following subsection.

4.2.2. Efficiency

We analyzed the efficiency of MTPool using the Heartbeat dataset. As shown in Fig. 2, we ran the models on the heartbeat test dataset 10 times and recorded the average real-time consumption. Although there are differences in the specific implementation methods of different models and the complexities of models cannot be fully reflected using only the running time, we can draw the following conclusions. First, as shown in Fig. 2(a), the running times of machine-learning methods such as WEASEL+MUSE and KNN-DTW are significantly higher than those of recently proposed deep learning-based methods such as MTPool, TapNet, and MLSTM-FCN. This shows the efficiency of deep learning methods on MTSC tasks. From Fig. 2(b), we can see that, although variational graph pooling is introduced in MTPool, it does not contribute significantly to the running time. The temporal convolution dominates the total running time, which is consistent with our analysis of the time complexity.

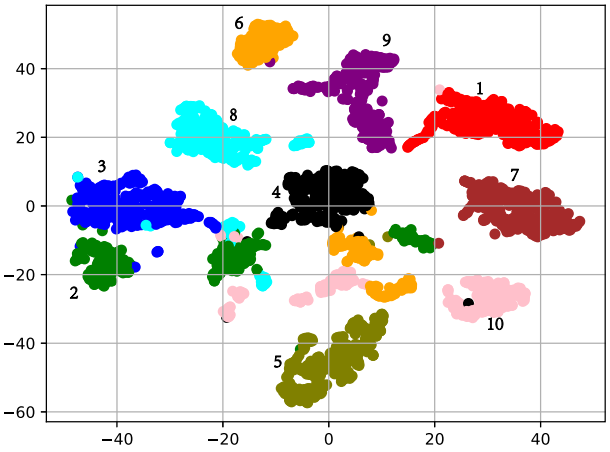


Fig. 4. Class Prototype Inspection: visualize the 128-dimension multivariate time-series embeddings learned for the PenDigits dataset in a two-dimensional image using t-SNE.

4.3. Ablation study

We conducted an ablation study to validate the effectiveness of key components that contribute to the improved outcomes of MTPool. First, we substituted our variational pooling layers with other advanced pooling methods in the MTPool framework. We then used a constant adjacency matrix for the initial graph structure instead of a adaptive adjacency matrix. The detailed setting of each variant model is as follows.

- **MTPool-M** is the MTPool framework with MemPool (Khasahmadi et al., 2020), which generates clustering centroids without involving the input graphs.
- **MTPool-D** is the MTPool framework with DiffPool (Ying et al., 2018), which trains two parallel GNNs to obtain node-level embeddings and cluster assignments.
- **MTPool-S** is the MTPool framework with SAGPool (Lee et al., 2019), which drops nodes from the input to pool the graph.
- **MTPool-One** is the MTPool framework with variational pooling and an all-one adjacency matrix.
- **MTPool-Corr** is the MTPool framework with variational pooling and a correlation coefficient adjacency matrix.
- **MTPool** is the MTPool framework with our proposed Variational Pooling and adaptive adjacency matrix.

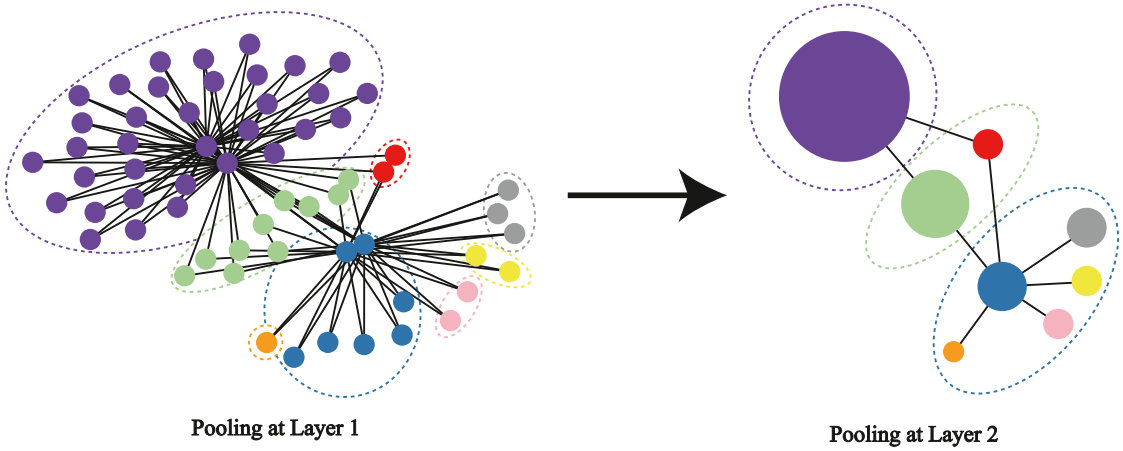


Fig. 5. Visualization of the graph pooling process in MTPool, using example graphs from Heartbeat dataset, which has 61 MTS variables, so the original graph has 61 nodes. The nodes in the second layer correspond to clusters in the first layer. We used the same color to represent nodes of the same cluster and dotted lines to indicate different clusters. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 3 shows the comparison results that were obtained. The important conclusions of these results are as follows:

(1) Different hierarchical graph pooling methods can be incorporated into our MTPool framework, and they exhibit comparable or better performance than state-of-the-art MTSC methods.
(2) Different adjacency matrices can be used in our MTPool framework. However, the performance of the all-one matrix was slightly worse than that of the correlation coefficient matrix, and our adaptive matrix achieved the best performance.
(3) Even if the three cutting-edge pooling methods (MemPool, DiffPool, and SAGPool) and two adjacency matrices (all-one and corr) have their own merits in the three selected datasets, our proposed variational pooling can achieve the best performance in all cases. This proves the effectiveness of our well-designed pooling method and the adaptive adjacency matrix.

4.4. Inspection of class prototype

This section visualizes the class prototype and its corresponding time-series embedding to prove the effectiveness of our well-trained time-series embedding. We used the t-SNE algorithm (Maaten & Hinton, 2008) to visualize the 128-dimensional time series embedded in the form of two-dimensional images. We used different colors to distinguish between the different categories. Fig. 4 shows the embeddings learned for the PenDigits dataset, which contains 3498 test samples of 10 different types. The following conclusions can be drawn based on the results obtained:

(1) The distance between data samples from different categories is much greater than the distance between data samples of the same type. Therefore, we can easily use the learned multivariate time series to embed the time-sequence classification.
(2) Low-dimensional time series embedding provides us with a more interpretable perspective to understand classifiers' problems. For example, categories two, six, and ten are not divided into complete pieces. Instead, these three categories are divided into several sub-parts. Thus, it helps to identify problems with the classifier and take further measures, such as adding more training samples to these three classes.

4.5. Case study: Visualizations

In this section, we visualize the graph pooling process using the Heartbeat dataset. Fig. 5 shows a visualization of node assignments in the first and second layers on a graph constructed from the Heartbeat dataset. We used the same color to represent nodes of the same cluster. The cluster membership of each node is determined by finding the argmax of its cluster assignment probability. We also observed that even if the final goal is to obtain graph-level embedding and the class to which MTS belongs, MTPool can still capture the hierarchical graph structure, which helps us further reveal the dependencies among different variables in MTS. Notably, the assignment matrix may not assign nodes to specific clusters. The column corresponding to the unused cluster has a lower value for all nodes. For example, in this case, the expected number of clusters we set in the first layer was 15 (greater than eight), but in fact, we obtained eight clusters. This reminds us that even if we define the expected cluster number in advance, MTPool automatically performs clustering to obtain the best coarser results suitable for this graph. Such characteristics can be adjusted for different inputs, and they thus have a strong generalization ability.

5. Conclusion

In this paper, we propose the first graph-pooling-based framework, MTPool, for MTS classification. The proposed framework can explicitly model pairwise dependencies among MTS variables and attain global embedding with strong interpretability and expressiveness. The experimental results demonstrate that the proposed model exhibits state-of-the-art performance compared to existing models.

Future research is promising for exploring and designing more powerful hierarchical graph pooling approaches that can be incorporated into our MTPool framework to attain a more expressive and interpretable global representation for MTS.

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.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No. 2019YFB2102600), the National Natural Science Foundation of China (No. 62002035), and the Natural Science Foundation of Chongqing, China (No. cstc2020jcyj-bshX0034).

References

Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., et al. (2018). The UEA multivariate time series classification archive, 2018. arXiv preprint arXiv:1811.00075.

Bagnall, A. J., Flynn, M., Large, J., Lines, J., & Middlehurst, M. (2020). A tale of two toolkits, report the third: on the usage and performance of HIVE-cote v1. 0. CoRR.

Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., & Wang, W. (2019). Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the twelfth ACM international conference on web search and data mining* (pp. 384–392).

Baldán, F. J., & Benítez, J. M. (2019). Distributed fastshapelet transform: a big data time series classification algorithm. *Information Sciences*, *496*, 451–463.

Chen, J., Ma, T., & Xiao, C. (2018). Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv preprint arXiv:1801.10247.

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems, Vol. 29* (pp. 3844–3852). Curran Associates, Inc..

Duan, Z., Wang, Y., Ye, W., Feng, Z., Fan, Q., & Li, X. (2021). Connecting latent ReLationships over heterogeneous attributed network for recommendation. arXiv preprint arXiv:2103.05749.

Feng, Z.-k., & Niu, W.-j. (2021). Hybrid artificial neural network and cooperation search algorithm for nonlinear river flow time series forecasting in humid and semi-humid regions. *Knowledge-Based Systems*, *211*, Article 106580.

Gao, H., & Ji, S. (2019). Graph u-nets. arXiv preprint arXiv:1905.05178.

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. CoRR abs/1706.02216. arXiv:1706.02216.

Iwana, B. K., Frinken, V., & Uchida, S. (2020). DTW-NN: A novel neural network for time series recognition using dynamic alignment between inputs and weights. *Knowledge-Based Systems*, *188*, Article 104971.

Kang, H., & Choi, S. (2014). Bayesian common spatial patterns for multi-subject EEG classification. *Neural Networks*, *57*, 39–50.

Karim, F., Majumdar, S., Darabi, H., & Harford, S. (2019). Multivariate LSTM-FCNs for time series classification. *Neural Networks*, *116*, 237–245.

Khasahmadi, A. H., Hassani, K., Moradi, P., Lee, L., & Morris, Q. (2020). Memory-based graph networks. In *International conference on learning representations*.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. CoRR abs/1609.02907. arXiv:1609.02907.

Lee, J., Lee, I., & Kang, J. (2019). Self-attention graph pooling. arXiv preprint arXiv:1904.08082.

Li, J., Yang, B., Li, H., Wang, Y., Qi, C., & Liu, Y. (2021). DTDR–ALSTM: Extracting dynamic time-delays to reconstruct multivariate data for improving attention-based LSTM industrial time series prediction models. *Knowledge-Based Systems*, *211*, Article 106508.

Liu, F., Xue, S., Wu, J., Zhou, C., Hu, W., Paris, C., et al. (2020). Deep learning for community detection: progress, challenges and opportunities. arXiv preprint arXiv:2005.08225.

Ma, X., Wu, J., Xue, S., Yang, J., Zhou, C., Sheng, Q. Z., et al. (2021). A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*.

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(Nov), 2579–2605.

Mori, U., Mendiburu, A., Miranda, I. M., & Lozano, J. A. (2019). Early classification of time series using multi-objective optimization techniques. *Information Sciences*, *492*, 204–218.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., et al. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 33* (pp. 4602–4609).

Pei, W., Dibeklioğlu, H., Tax, D. M., & van der Maaten, L. (2017). Multivariate time-series classification using the hidden-unit logistic model. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(4), 920–931.

Ranjan, E., Sanyal, S., & Talukdar, P. P. (2020). ASAP: Adaptive structure aware pooling for learning hierarchical graph representations. In *AAAI* (pp. 5470–5477).

Ruiz, A. P., Flynn, M., & Bagnall, A. (2020). Benchmarking multivariate time series classification algorithms. arXiv preprint arXiv:2007.13156.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80.

Schäfer, P., & Leser, U. (2017). Multivariate time series classification with weasel+ MUSE. arXiv preprint arXiv:1711.11343.

Seto, S., Zhang, W., & Zhou, Y. (2015). Multivariate time series classification using dynamic time warping template selection for human activity recognition. In *2015 IEEE symposium series on computational intelligence* (pp. 1399–1406). IEEE.

Shi, Z., Bai, Y., Jin, X., Wang, X., Su, T., & Kong, J. (2021). Parallel deep prediction with covariance intersection fusion on non-stationary time series. *Knowledge-Based Systems*, *211*, Article 106523.

Su, X., Xue, S., Liu, F., Wu, J., Yang, J., Zhou, C., et al. (2021). A comprehensive survey on community detection with deep learning. arXiv preprint arXiv:2105.12584.

Sun, C., Chen, C., Li, W., Fan, J., & Chen, W. (2019). A hierarchical neural network for sleep stage classification based on comprehensive feature learning and multi-flow sequence learning. *IEEE Journal of Biomedical and Health Informatics*, *24*(5), 1351–1366.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

Wang, Y., Duan, Z., Huang, Y., Xu, H., Feng, J., & Ren, A. (2020). MTHetGNN: A heterogeneous graph embedding framework for multivariate time series forecasting. arXiv e-Prints, arXiv–2008.

Wang, Y., Duan, Z., Liao, B., Wu, F., & Zhuang, Y. (2019). Heterogeneous attributed network embedding with graph convolutional networks. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 33* (pp. 10061–10062).

Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 international joint conference on neural networks (IJCNN)* (pp. 1578–1585). IEEE.

Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., & Zhang, C. (2020). Connecting the dots: Multivariate time series forecasting with graph neural networks. arXiv preprint arXiv:2005.11650.

Xu, H., Chen, R., Bai, Y., Duan, Z., Feng, J., Sun, Y., et al. (2020). CoSimGNN: Towards large-scale graph similarity computation. arXiv preprint arXiv:2005.07115.

Xu, H., Duan, Z., Wang, Y., Feng, J., Chen, R., Zhang, Q., et al. (2021). Graph partitioning and graph neural network based hierarchical graph matching for graph similarity computation. *Neurocomputing*, *439*, 348–362.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks? ArXiv arXiv:1810.00826.

Xu, H., Huang, Y., Duan, Z., Wang, X., Feng, J., & Song, P. (2020). Multivariate time series forecasting with transfer entropy graph. arXiv e-Prints, arXiv–2005.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems* (pp. 4800–4810).

Yu, Z., & Lee, M. (2015). Real-time human action classification using a dynamic neural model. *Neural Networks*, *69*, 29–43.

Zhang, X., Gao, Y., Lin, J., & Lu, C.-T. (2020). TapNet: Multivariate time series classification with attentional prototypical network. In *AAAI* (pp. 6845–6852).

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., et al. (2020). Graph neural networks: A review of methods and applications. *AI Open*, *1*, 57–81.

Zhou, Y., Duan, Z., Xu, H., Feng, J., Ren, A., Wang, Y., et al. (2020). Parallel extraction of long-term trends and short-term fluctuation framework for multivariate time series forecasting. arXiv preprint arXiv:2008.07730.

Zhou, K., Wang, W., Huang, L., & Liu, B. (2021). Comparative study on the time series forecasting of web traffic based on statistical model and generative adversarial model. *Knowledge-Based Systems*, *213*, Article 106467.