

Multivariate Time Series Classification with Hierarchical Variational Graph Pooling

Ziheng Duan^{a,b,*}, Haoyan Xu^{a,b,c,*}, Yueyang Wang^{a,**}, Yida Huang^a, Anni Ren^b, Zhongbin Xu^b, Yizhou Sun^c and Wei Wang^c

^aSchool of Big Data and Software Engineering, Chongqing University, Chongqing, 401331, China

^bCollege of Energy Engineering, Zhejiang University, Zhejiang, 310027, China

^cDepartment of Computer Science, University of California, Los Angeles, CA 90095, USA

ARTICLE INFO

Keywords:

Multivariate Time Series Classification, Graph Neural Networks, Graph Pooling, Graph Classification

ABSTRACT

With the advancement of sensing technology, multivariate time series classification (MTSC) has recently received considerable attention. Existing deep learning-based MTSC techniques, which mostly rely on convolutional or recurrent neural networks, are primarily concerned with the temporal dependency of single time series. As a result, they struggle to express pairwise dependencies among multivariate variables directly. Furthermore, current spatial-temporal modeling (e.g., graph classification) methodologies based on Graph Neural Networks (GNNs) are inherently flat and cannot aggregate hub data in a hierarchical manner. To address these limitations, we propose a novel graph pooling-based framework MTPool to obtain the expressive global representation of MTS. We first convert MTS slices to graphs by utilizing interactions of variables via graph structure learning module and attain the spatial-temporal graph node features via temporal convolutional module. To get global graph-level representation, we design an "encoder-decoder" based variational graph pooling module for creating 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. At last, a differentiable classifier takes this coarsened representation to get the final predicted class. Experiments on ten benchmark datasets exhibit MTPool outperforms state-of-the-art strategies in the MTSC task.

1. Introduction

Multivariate time series (MTS), which are gathered from numerous variables or sensors in our day-by-day life, are utilized in different investigations [17, 27, 8, 42]. Attributable to cutting edge sensing techniques, the Multivariate Time Series Classification (MTSC) issue, recognizing the labels for MTS records, has pulled in a ton of consideration in late numerous years [40, 4, 19]. MTSC models have been applied in a broad range of real-world applications [41, 11], such as sleep stage identification [28], healthcare [12] and action recognition [39]. Specifically, MTS has the accompanying two significant attributes: (1) Each univariate time series has an *inner temporal reliance* mode; (2) There always exist *hidden dependency* relationships among different MTS variables. Capturing these two attributes is a significant contribution to getting better classification performance but also a challenging task.

A lot of MTS classification methods have been proposed throughout the long term. Distance-based methods, like Dynamic Time Warping (DTW) with k-NN [26], and feature-based methods such as Hidden Unit Logistic Model (HULM) [21] have proven to be successful in classification tasks on many benchmark MTS datasets. In any case, these methodologies need hefty crafting on data preprocessing and feature engineering and they cannot fully explore the *inner temporal reliance* mode in each univariate time series. Recently, many deep learning-based methods have been exploited for end-to-end MTS classification. Fully convolutional networks (FCN) and the residual networks (ResNet) can achieve comparable or better performance than traditional methods [32]. MLSTM-FCN [13] utilizes an LSTM layer and a stacked CNN layer alongside squeeze-and-excitation blocks to obtain representations. These deep learning-based methods have achieved promising performance in MTSC tasks. Notwithstanding, the current deep learning-based strategies don't show the *hidden dependency* relationships among different MTS variables. Their input of these models should be grid data, also limiting the representational ability.

*These authors contributed equally to this research.

**Corresponding author.

ORCID(s): 0000-0003-3210-0930 (Y. Wang)

To address the above problem, we first 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. As we know, graphs are a particular type of information that portrays the relationships between various entities or nodes. Existing mining graph methods, like Graph Neural Networks (GNNs) [24], aggregating feature information of adjoining nodes to learn node or graph representation, have been demonstrated successfully capturing the hidden relations of nodes. Consequently, mining MTS information by utilizing graph neural networks can be a promising method to save their temporal patterns while exploiting the interdependency among time series variables [36, 30, 33]. Therefore, this paper proposes converting MTS slices to 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 whole graph, where the graph structure and all the initial node-level representations are inputs. For instance, given a molecule, the task could be to anticipate whether it is poisonous [22]. Although current GNNs based spatial-temporal modeling strategies could aggregate the graph structure and node-level representations, they are still inherently flat and do not have the capacity of aggregate hub data in a hierarchical way [38]. Recently, some hierarchical pooling methods have been proposed and achieved promising results in many graph classification tasks, such as gPool [7], DiffPool [34], MemGNN [12]. These methods design specific hierarchical graph pooling layers, where nodes are recursively aggregated to frame a cluster addressing a hub in the pooled graph. gPool downsamples by selecting the most important nodes. DiffPool downsamples by clustering the nodes using GNNs. Like DiffPool, MemGNN utilizes a multi-head 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 notice that the key process of most hierarchical pooling mechanisms, generating centroids for soft cluster assignments, is not related to input graphs. This does not make sense because the centroid for different graphs should be different. Although MemGNN randomly initializes centroids (keys) before training and makes centroids learnable during training. The centroids cannot change when testing so that the testing graphs can only use the same centroids as train graphs for cluster assignments. Intuitively, each input graph should have its specific centroids based on its topology structure and node features. In general, the pooling mechanism should keep three significant properties of graphs: (1) *Permutation invariance*: The centroids of the same graph should be invariant when the permutation of nodes changes. (2) *Input correlation*: The centroids should change if the input graph changes. (3) *Dimensional adjustability*. The dimension of the centroids matrix should be adjustable by the dimension of the input graph and coarsened graph.

Towards this end, we propose a novel MTS classification framework called MTPool (*M*ultivariate *T*ime Series Classification with Variational Graph *P*ooling). The goal of MTPool is to model temporal patterns and hidden dependency relationships among MTS variables and obtain the MTS's global representation. Specifically, MTPool first constructs a graph based on MTS data through the graph structure learning module and learns spatial-temporal features of MTS through the temporal convolution module. Then, to keep the three properties mentioned above, we design a novel pooling layer, Variational Pooling, for graph coarsening. This pooling layer contains an "encoder-decoder" architecture, enabling the generation process of centroids to be input-related and making the model more inductive. Next, MTPool adapts GNNs for jointly learning graph representation, after which the graph is progressively coarsened to one node. Finally, these final coarsened graph-level representations can be used as features input to a differentiable classifier for MTSC tasks. To summarize, our fundamental contributions can be finished up as follows:

- (1) As far as we could possibly know, 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, Variational Pooling. The centroids for 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 exhibit MTPool outperforms cutting edge strategies in the MTSC task.

2. Related Work

2.1. Multivariate Time Series Classification

Most MTSC methods can be grouped into three categories: distance-based, feature-based, and deep learning-based approaches. Here we only discuss deep learning-based strategies.

Currently, two popular deep learning models, CNN and RNN, are widely used in MTS classification. These models often use an LSTM layer and stacked CNN layer to extract time-series features, and a softmax layer is then applied to predict the label [40]. For example, MLSTM-FCN [13] utilizes an LSTM layer and a stacked CNN layer alongside squeeze-and-excitation blocks to obtain representations. TapNet [40] also constructs 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. However, the limitation of the above models is obvious: they assume that the time series variables have the same effect among each other. Consequently, they cannot explicitly model the pairwise dependencies among variables. In this case, graphs are the most appropriate data structure for modeling MTS.

2.2. Graph Neural Networks

[24] first proposed the idea and the concept of graph neural network (GNN), which broadened existing neural networks for handling 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 embedding vectors of its neighbor nodes [35, 7]. Numerous GNN variations have been proposed and have accomplished cutting-edge results on both node and graph classification assignments [31]. For example, the Graph Convolutional Networks (GCN) [15] could be viewed as an approximation of spectral-domain convolution of the graph data. GraphSAGE [10], and Fast-GCN [5] sample and aggregate of the neighborhood information while enabling training in batches yet forfeiting some time-proficiency. Graph Attention Networks (GAT) [29] designs a new way to gather neighbors through self-attention. After this, Graph Isomorphism Network (GIN) [37] and k-GNNs [20] are developed, presenting more perplexing and different types of aggregation.

2.3. Graph Pooling

Graph pooling strategies can be characterized into three classes: topology based, global, and hierarchical pooling. Here we only discuss hierarchical pooling methods. DiffPool [38] trains two parallel GNNs to obtain node-level representations and cluster assignments. gPool [9], and SAGPool [16] drop nodes from the input graph as opposed to bunch various nodes to frame a cluster in the pooled graph. They devise a top-K node choice method to make an initiated sub-graph for the following layers. Although they are more efficient than DiffPool, they do not gather nodes nor calculate soft edge weights. This makes them unable to preserve node and edge information effectively. MemGNN [14] likewise soft cluster assignments, and they utilize a clustering-friendly distribution to figure the attention scores among nodes and clusters. However, they generate centroids (which are used to calculate soft assignments) without involving the input graphs, which does not make sense intuitively because the centroids for different input graphs should be different. We propose Variational Pooling, using an “encoder-decoder” architecture to get centroids to address this limitation. It can keep the property of permutation invariance while making centroids input-related to graphs [34].

3. Framework

In this part, 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 Figure 1.

3.1. Problem Formulation

A multivariate time series (MTS) can be represented as a matrix $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \in \mathbb{R}^{n \times T}$, in which 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 $\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_N\} \in \mathbb{R}^{N \times n \times T}$, where N is number of MTS slices in the group, and the corresponding labels $\mathcal{Y} = \{y_1, y_2, \dots, y_N\} \in \mathbb{R}^N$, the research goal is to learn the mapping relationship between \mathcal{X} and \mathcal{Y} based on the proposed model.

3.2. Graph Structure Learning

We propose a dynamic relation embedding strategy, which learns the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ adaptively to model latent relations in MTS sample \mathbf{X} . The learned graph structure (dynamic adjacency matrix) \mathbf{A} is defined as:

$$\mathbf{A} = \text{Embed}_1(\mathbf{X}), \quad (1)$$

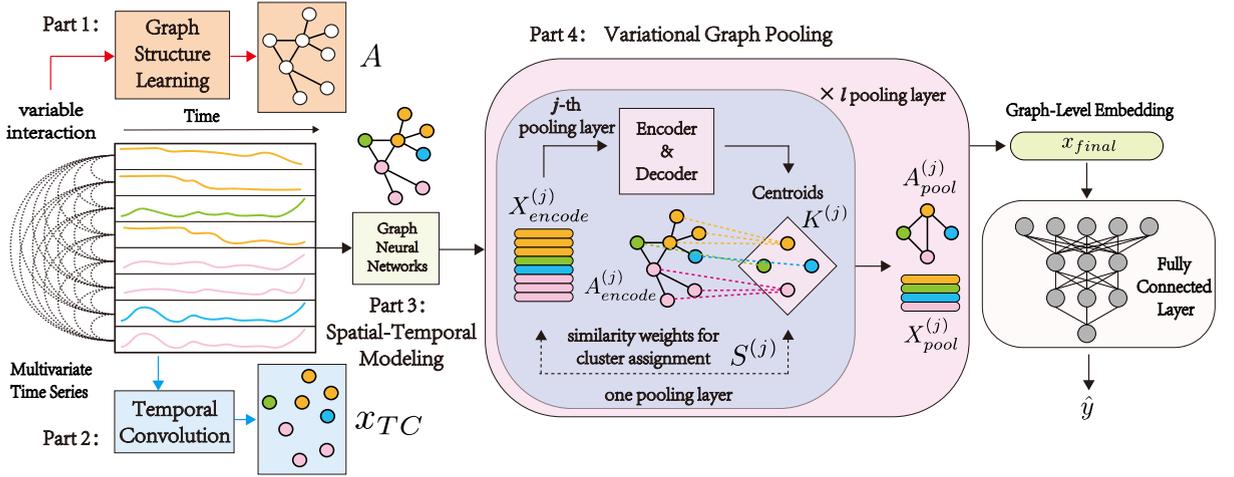


Figure 1: The general architecture of MTPool. Adjacency matrix and feature matrix are constructed through graph structure learning and temporal convolution respectively. Then GNNs aggregate and fuse spatial-temporal features. After several pooling layers, the input graphs are hierarchically coarsened to one node to attain their graph-level representations. Finally, these final output graph-level embeddings can be used as features input to 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.

where $Embed_1()$ represents the graph structure learning function. For the input MTS $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \in \mathbb{R}^{n \times T}$ we first calculate the similarity matrix \mathbf{C} between sampled MTS variables:

$$C_{ij} = \frac{\exp(-\sigma(\text{distance}(\mathbf{x}_i, \mathbf{x}_j)))}{\sum_{p=1}^n \exp(-\sigma(\text{distance}(\mathbf{x}_i, \mathbf{x}_p)))}, \quad (2)$$

where \mathbf{x}_i and \mathbf{x}_j denote the i^{th} and j^{th} MTS variables ($i, j = 1, 2, \dots, n$) and distance denotes the distance metric such as Euclidean Distance, Absolute Value Distance, Dynamic Time Warping, etc. Then the dynamic adjacency matrix \mathbf{A} can be calculated as:

$$\mathbf{A} = \sigma(\mathbf{C}\mathbf{W}_{adj}), \quad (3)$$

where σ is an activation function and $\mathbf{W}_{adj} \in \mathbb{R}^{n \times n}$ is learnable model parameters to dynamically generate adjacency matrix based on input features. What's more, to improve training efficiency, reduce noise impact and make the model more robust, threshold c_1 is set to make the adjacency matrix sparse:

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

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

3.3. Temporal Convolution

This stage aims to extract temporal features, which is associated with or change over time, and construct feature matrix $\mathbf{X}_{TC} \in \mathbb{R}^{n \times d}$, where d is the obtained feature dimension. With temporal convolution, we can get each MTS's feature matrix as follows:

$$\mathbf{X}_{TC} = Embed_2(\mathbf{X}), \quad (5)$$

where $Embed_2()$ is the temporal convolution function. When investigating 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 instance, the number of inhabitants in a specific city shows a particular pattern each day. Yet, a significant example can be seen by noticing it on the size of multi-week or one month. In this manner, it is sensible and vital to separate the highlights of the time arrangement in units of numerous particular periods. To mimic the present circumstance, we utilize numerous CNN channels with various responsive fields, namely kernel sizes, to extract features at multiple time scales.

For the i -th CNN filters (the number of CNN filters is q and $i = 1, 2, \dots, q$), given the input time series \mathbf{X} , the feature vector \mathbf{f}_i are expressed as follows: $\mathbf{f}_i = \sigma(\mathbf{W}_i * \mathbf{X} + \mathbf{b})$, where $*$ denotes the convolution operation, σ is a nonlinear activation function, such as $RELU(x) = \max(0, x)$, $\mathbf{W}_i \in \mathbb{R}^{1 \times ks}$ represents the i -th CNN kernel, ks is the kernel size and \mathbf{b} is the bias. And the final feature vector can be expressed as $\mathbf{X}_{TC} = \left(\begin{array}{c} |q| \\ \parallel \\ i=1 \end{array} \mathbf{f}_i \right)$, where \parallel means concatenate operation from the feature \mathbf{f}_1 to the feature \mathbf{f}_q . In this way, temporal features under various periods are extracted, providing powerful reference for time series classification.

3.4. Spatial-temporal Modeling

In this stage, m GNN layers (G_1, G_2, \dots, G_m) are employed on the input graphs (denoted as $\mathbf{X}_{TC}, \mathbf{A}$) for spatial-temporal modeling. GNN layers can fuse spatial dependencies and temporal patterns for embedding feature of nodes and transform the feature dimension of nodes to d_{encode} , just as shown in equation (6):

$$\mathbf{Z} = G_m(G_{m-1}(\dots G_1(\mathbf{X}_{TC}, \mathbf{A})\dots)), \quad (6)$$

where $\mathbf{Z} \in \mathbb{R}^{n \times d_{encode}}$, represents the learned node embedding from the spatial-temporal modeling. G_j ($j = 1, 2, \dots, m$) consists of a graph neural network layer (GNN) and a batch normalization layer. GNN can be such as GCN [6], GAT [29], GIN [37], etc. Inspired by k-GNNs [20] model, we use the following propagation mechanism in this paper to calculate the forward-pass update of a node denoted by \mathbf{v}_i :

$$\mathbf{z}_i^{(j+1)} = \sigma\left(\mathbf{z}_i^{(j)} \mathbf{W}_1^{(j)} + \sum_{r \in \mathbf{N}(i)} \mathbf{z}_r^{(j)} \mathbf{W}_2^{(j)}\right), \quad (7)$$

where $\mathbf{W}_1^{(j)}$ and $\mathbf{W}_2^{(j)}$ are parameter matrices of the j -th GNN layer, $\mathbf{z}_i^{(j)}$ is the hidden state of node \mathbf{v}_i in the j^{th} layer and $\mathbf{N}(i)$ 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 relationship among variables, effectively avoiding the information redundancy brought by high dimensions.

3.5. Variational Graph Pooling

3.5.1. Overall Transformation

In this stage, the encoded graph is hierarchically pool to a single node for generating its graph-level representation. This successive process can be expressed as:

$$\mathbf{x}_{final} = P_l(P_{l-1}(\dots P_1(\mathbf{Z}, \mathbf{A})\dots)), \quad (8)$$

where P_j ($j = 1, 2, \dots, l$) represents one pooling layer, and after stacking l pooling layers, we get its graph-level representation vector \mathbf{x}_{final} . For the j -th pooling layer, $P_j()$ can pool each encoded graph, to a specific coarsened graph. The overall transformation of $P_j()$ is shown in equation 9:

$$\begin{aligned} \mathbf{X}_{pool}^{(j)} &= \sigma\left(\mathbf{S}^{(j)} \mathbf{X}_{encode}^{(j)} \mathbf{W}_{pool}^{(j)}\right), \\ \mathbf{A}_{pool}^{(j)} &= \sigma\left(\mathbf{S}^{(j)} \mathbf{A}_{encode}^{(j)} (\mathbf{S}^{(j)})^T\right), \end{aligned} \quad (9)$$

where σ is a non-linear activation function, $\mathbf{W}_{pool}^{(j)} \in \mathbb{R}^{d_{encode}^{(j)} \times d_{pool}^{(j)}}$ is a trainable parameter matrix standing for a linear transformation and $\mathbf{S}^{(j)} \in \mathbb{R}^{n_{pool}^{(j)} \times n_{encode}^{(j)}}$ is the assignment matrix representing a projection from the original nodes to pooled nodes (clusters). The details of how to calculate $\mathbf{S}^{(j)}$ are in the next subsection 3.5.2. $\mathbf{X}_{encode}^{(j)} \in \mathbb{R}^{n_{encode}^{(j)} \times d_{encode}^{(j)}}$ and $\mathbf{A}_{encode}^{(j)} \in \mathbb{R}^{n_{encode}^{(j)} \times n_{encode}^{(j)}}$ represent the encoded feature and adjacency matrix of the input graph in the j -th pooling

layer respectively; $\mathbf{X}_{pool}^{(j)} \in \mathbb{R}^{n_{pool}^{(j)} \times d_{pool}^{(j)}}$ and $\mathbf{A}_{pool}^{(j)} \in \mathbb{R}^{n_{pool}^{(j)} \times n_{pool}^{(j)}}$ represent the pooled feature and adjacency matrix of the pooled graph in the j -th pooling layer respectively. In most cases, the pooled graph has less nodes than the input graph ($n_{pool}^{(j)} < n_{encode}^{(j)}$). For the first pooling layer, we have: $\mathbf{X}_{encode}^{(0)} = \mathbf{Z}$ and $\mathbf{A}_{encode}^{(0)} = \mathbf{A}$.

3.5.2. How to Compute Assignment Matrix

In this part, we propose a new pooling method: Variational Pooling, to address the limitation of existing methods as we discussed in section 2.3 and to calculate $\mathbf{S}^{(j)}$ in a more effective way. Although there are many advanced graph pooling methods proposed in recent years, they all have some limitations. For example, MemGNN [14] utilizes a multi-head exhibit of memory keys and a convolution operator to sum the soft cluster assignments from various heads. MemGNN utilizes a clustering-friendly distribution to figure the attention scores among nodes and clusters and performs better than DiffPool in many tasks. Still, we notice that it generates memory heads, which stand for the new centroids in the space of pooled graphs, without the involvement of the input graphs. However, 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 $\mathbf{K}^{(j)} = [\mathbf{K}_1^{(j)}, \mathbf{K}_2^{(j)}, \dots, \mathbf{K}_h^{(j)}]^T \in \mathbb{R}^{h \times n_{pool}^{(j)} \times d_{encode}^{(j)}}$ based on the input graph in $P_j()$ and then compute and aggregate the relationship between every batch of centroids and the encoded graph for assignment matrix $\mathbf{S}^{(j)}$.

3.5.3. How to Generate Centroids

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

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

Therefore, we propose an ‘‘encoder-decoder’’ architecture for computing centroids matrix $\mathbf{K}^{(j)}$. In general, the *encoder* ensures the property of permutation invariance and makes the centroids adaptive to input graphs while the *decoder* controls the dimension of the output coarsened graph:

$$\mathbf{K}^{(j)} = \text{Decoder} \left(\text{Encoder} \left(\mathbf{X}_{encode}^{(j)} \right) \right). \quad (10)$$

As shown in equation 10, an *encoder* is deployed over the input graph, transforming $\mathbf{X}_{encode}^{(j)} \in \mathbb{R}^{n_{encode}^{(j)} \times d_{encode}^{(j)}}$ to $\mathbf{X}_g^{(j)} \in \mathbb{R}^{1 \times d_{encode}^{(j)}}$. Here $\mathbf{X}_g^{(j)}$ is the feature that incorporates the input information. Then a *decoder* is applied to map $\mathbf{X}_g^{(j)}$ to $\mathbf{K}^{(j)} \in \mathbb{R}^{(h \times n_{pool}^{(j)}) \times d_{encode}^{(j)}}$, after which $\mathbf{K}^{(j)}$ is reshaped to $\mathbb{R}^{h \times n_{pool}^{(j)} \times d_{encode}^{(j)}}$. Refer to [3], the expression of encoder and decoder we use in this paper is as follows:

$$\begin{aligned} \text{Encoder} : \mathbf{X}_g^{(j)} &= \sum_{i=1}^n \sigma \left((\mathbf{u}_i^{(j)})^T \mathbf{x}_{avg}^{(j)} \mathbf{u}_i^{(j)} \right) \\ &= \sum_{i=1}^n \sigma_1 \left((\mathbf{u}_i^{(j)})^T \sigma_2 \left(\left(\frac{1}{n} \sum_{j=1}^n \mathbf{u}_j^{(j)} \right) \mathbf{W}_{avg}^{(j)} \mathbf{u}_i^{(j)} \right) \right), \end{aligned} \quad (11)$$

$$\text{Decoder} : \text{MLP},$$

where $\mathbf{u}_i^{(j)}$ is the embedding of node i of the input graph in the j -th pooling layer, σ_1 is the *sigmoid* function, σ_2 is the *tanh* activation function and $\mathbf{W}_{avg}^{(j)} \in \mathbb{R}^{d_{encode}^{(j)} \times d_{encode}^{(j)}}$ is a learnable weight matrix. For the encoder, we use the attention mechanism to guide the model to learn weights under specific tasks to generate graph-level representation $\mathbf{X}_g^{(j)}$ for centroids. For decoder, here we use Multiple Layer Perceptron (MLP) to reshape matrix $\mathbf{K}^{(j)}$ to the size we need.

3.5.4. Computing Assignment Matrix

With centroids matrix $\mathbf{K}^{(j)}$, we can compute the relationship $\mathbf{S}_p^{(j)} \in \mathbb{R}^{n_{pool}^{(j)} \times n_{encode}^{(j)}}$ ($p = 1, 2, \dots, h$) between centroids $\mathbf{K}_p^{(j)} \in \mathbb{R}^{n_{pool}^{(j)} \times d_{encode}^{(j)}}$ ($p = 1, 2, \dots, h$) and $\mathbf{X}_{encode}^{(j)} \in \mathbb{R}^{n_{encode}^{(j)} \times d_{encode}^{(j)}}$ in j -th pooling layer. We use cosine similarity to evaluate the relationship between input node embeddings and centroids, as described in equation 12, followed by a row normalization deployed in the resulting assignment matrix:

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

where $(\mathbf{S}_p^{(j)})' \in \mathbb{R}^{n_{pool}^{(j)} \times n_{encode}^{(j)}}$ ($p = 1, 2, \dots, h$) is the assignment matrix before normalization. We finally aggregate the information of h relationship $\mathbf{S}_p^{(j)}$:

$$\mathbf{S}^{(j)} = \Gamma_{\phi} \left(\begin{array}{c} |h| \\ \parallel \\ \mathbf{S}_p^{(j)} \end{array} \right). \quad (13)$$

In equation 13, we concatenate $\mathbf{S}_p^{(j)}$ ($p = 1, 2, \dots, h$) and perform a trainable weighted sum Γ_{ϕ} to the concatenated matrix, leading to the final assignment matrix $\mathbf{S}^{(j)}$.

3.6. Differentiable Classifier

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

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

where N is the set of training samples, M denotes 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.

4. Experiments

In this section, we lead comprehensive analyses on ten benchmark datasets for MTS classification and look at the evaluation results of our model (MTPool) with other baselines.

4.1. Experiment Settings

4.1.1. Datasets

We use ten publicly available benchmark datasets from the UEA MTS classification archive.¹ The main characteristics of each dataset are summarized in Table 1. The train and test sizes represent the number of MTS slices in the train and test datasets, respectively. Num series refers to the number of variables in each MTS slice; the series length refers to the length or the feature dimension of each variable in each MTS slice; class refers to the number of types of MTS slices.

4.1.2. Metrics

Like other MTS classification methods [40], we use classification accuracy as an evaluation metric:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \quad (15)$$

where TP , TN , FP and FN respectively stands for true positive, true negative, false positive and false negative.

¹Datasets are available at <http://timeseriesclassification.com>. We exclude data with extremely long length, unequal length, high dimension size, and in-balance split.

Algorithm 1: MTPool algorithm framework

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

```

1  $\hat{\mathcal{Y}} = \{\}$ ; // Record predict labels of MTS slices
2 for  $X_k$  in  $\{X_1, X_2, \dots, X_N\}$ ,  $X_k \in \mathbb{R}^{n \times T}$  do
3   for  $x_i$  in  $X_k = \{x_1, x_2, \dots, x_n\}$ ,  $x_i \in \mathbb{R}^T$  do
4     for  $x_j$  in  $X_k = \{x_1, x_2, \dots, x_n\}$ ,  $x_j \in \mathbb{R}^T$  do
5        $C_{ij} = \frac{\exp(-\sigma(\text{distance}(x_i, x_j)))}{\sum_{p=0}^n \exp(-\sigma(\text{distance}(x_i, x_p)))}$  // Calculating the similarity matrix
6     end
7   end
8    $A = \sigma(CW_{adj})$ ; // Calculating the dynamic adjacency matrix
9    $A = \text{Normalize}_{row}(\text{Threshold}(A))$ ; // Graph Structure Learning
10   $X_{TC} = \text{Temporal\_Convolution}(X_k) \in \mathbb{R}^{n \times d}$ ; // Temporal Convolution
11   $Z = G_m(G_{m-1}(\dots G_1(X_{TC}, A)\dots))$ ; // Spatial-temporal Modeling
12   $X_{encode}^{(0)} = Z$ ,  $A_{encode}^{(0)} = A$ ; // The input of the first pooling layer
13  for  $P_j$  in  $\text{Pooling\_Layers} = \{P_1, P_2, \dots, P_l\}$  do
14     $K^{(j)} = [K_1^{(j)}, K_2^{(j)}, \dots, K_h^{(j)}]^T = \text{Decoder}(\text{Encoder}(X_{encode}^{(j)}))$ ;
15     $\text{Encoder} : X_g^{(j)} = \sum_{i=1}^n \sigma_1((u_i^{(j)})^T \sigma_2((\frac{1}{n} \sum_{j=1}^n u_j^{(j)}) W_{avg}^{(j)}) u_i^{(j)})$ ,  $\text{Decoder} : \text{MLP}$ ;
16    // Generating Centroids
17     $S^{(j)} = \Gamma_\phi \left( \left\| \begin{matrix} |h| \\ \parallel \\ p=1 \end{matrix} S_p^{(j)} \right\| \right)$ ,  $S_p^{(j)} = \text{Normalize}_{row}(\text{Cosine}(K_p^{(j)}, X_{encode}^{(j)}))$ ,  $p = 1, 2, \dots, h$ ;
18    // Computing Assignment Matrix
19     $X_{pool}^{(j)} = \sigma(S^{(j)} X_{encode}^{(j)} W_{pool}^{(j)})$ ;
20     $A_{pool}^{(j)} = \sigma(S^{(j)} A_{encode}^{(j)} (S^{(j)})^T)$ ;
21    // Variational Graph Pooling
22     $X_{encode}^{(j+1)} = X_{pool}^{(j)}$ ,  $A_{encode}^{(j+1)} = A_{pool}^{(j)}$ ; // Get the input of the next pooling layer
23  end
24   $x_{final} = X_{pool}^{(j)}$ ;
25   $\hat{y}_k = \text{MLP}(x_{final})$ ;
26   $\hat{\mathcal{Y}} \leftarrow \hat{y}_k$ ;
27  // Add  $\hat{y}_k$  to  $\hat{\mathcal{Y}}$ 
28 end
29 return  $\hat{\mathcal{Y}}$ 

```

4.1.3. Methods for Comparison

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

- (1) **ED, DTW_I, DTW_D, - with and without normalization (norm)** [1] are the common distance-based models. 1-Nearest Neighbour with distance functions include Euclidean (ED); dimension-independent dynamic time warping (DTW_I); and dimension-dependent dynamic time warping (DTW_D).
- (2) **WEASEL+MUSE** [25] stands for the Word ExtrAction for time Series cLassification (WEASEL) with a Multivariate Unsupervised Symbols and dErivatives (MUSE). It is the most effective bag-of-patterns algorithm for MTSC.
- (3) **HIVE-COTE** [2] is a heterogeneous meta ensemble for time series classification. This approach is a good baseline

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

for assessing bespoke Multivariate Time Series Classification.

- (4) **MLSTM-FCN** [13] 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 representations.
- (5) **TapNet** [40] draws on the strengths of both traditional and deep learning approaches. It also constructs a LSTM layer and a stacked CNN layer, followed by an attentional prototype network.
- (6) **MTPool** is the MTPool framework with our proposed Variational Pooling and dynamic adjacency matrix.

4.1.4. Training Details

All the networks are implemented with Pytorch 1.4.0 in python 3.6.2 and trained with 10000 epochs (computing infrastructure: Ubuntu 18.04 operating system, GPU NVIDIA GeForce RTX 2080 Ti with 8 Gb GRAM and 32 Gb of RAM). We use {3, 5, 7} as the convolutional kernel size and ten as the channel number. The threshold to make the adjacency matrix sparse is chosen from {0.05, 0.1, 0.2} and the output dimension of the GNNs layer is 128. For pooling layers, 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 is 10^{-4} , and the categorical cross-entropy loss and the Adam optimization are used to optimize the parameters of our models.

4.2. Main Results

Table 2 shows the accuracy results on the selected 10 UEA datasets of MTPool and other MTS classifiers. "Avg. Rank" in the table means 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 refer to [23] for the corresponding results. The best performance in each dataset is bolded.

First of all, we can observe that MTPool wins on eight datasets, better than several other advanced MTS classification methods (such as TapNet or MLSTM-FCN wins on one dataset). 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 can also find that methods based on deep learning proposed in recent years, such as TapNet and MLSTM-FCN, have better performance than methods based on nearest neighbors, proving the effectiveness of deep learning in extracting MTS features.

More importantly, with the different datasets, the number of variables and lengths span an extensive range. For instance, the variables number ranges from 2 to 64, and the minimum length of MTS is eight, and the maximum is 300. In the case of variable dataset parameters, our framework maintains considerable competitiveness, proving our model's robustness. 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, thus performing well. For instance, MTPool achieved the best and second-best accuracy on the MotorImagery (64 variables) and Heartbeat (61 variables). This means that when the number of variables is relatively large, the hierarchical pooling framework we provide can better capture the graph's

Table 2

Accuracy of the 11 algorithms on the default training/test datasets of 10 selected UEA MTSC archives. The best performance is bolded.

Methods / Datasets	AF	FM	HMD	HB	LIB	MI	NATO	PD	SRS2	SWJ	Avg. Rank	Wins/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

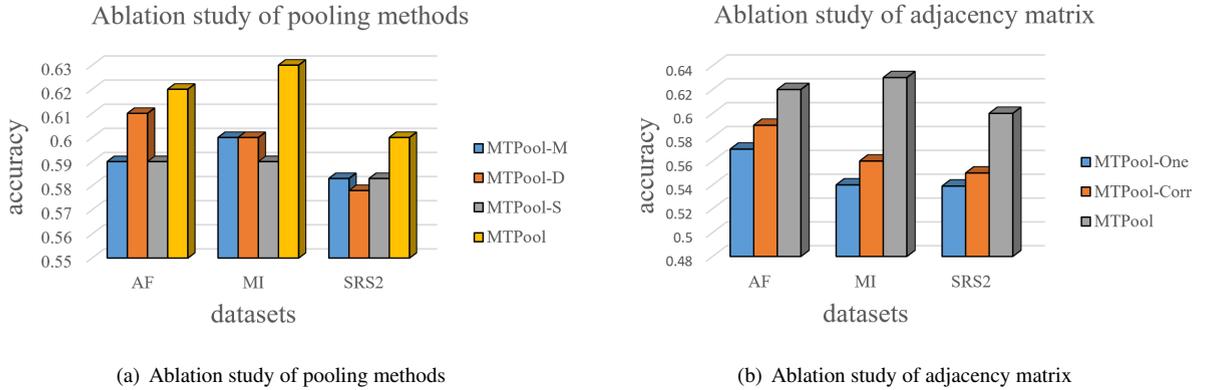


Figure 2: Ablation study of pooling methods and adjacency matrix. AF, MI and SRS2 datasets are used and different colors represent different methods.

structure and generate time series embeddings with more robust characterization capabilities. It is worth noting that when the number of variables is relatively small, by reducing the number of pooling layers, MTPool can also achieve good accuracy: MTPool obtains the best accuracy on PenDigits (2 variables), SelfRegulationSCP2 (7 variables), and StandWalkJump (4 variables) datasets. A more detailed analysis of MTPool is in the following subsection.

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 substitute our Variational Pooling layers with some other advanced pooling methods in the MTPool framework. Then we use a static adjacency matrix for the initial graph structure instead of a dynamic adjacency matrix. The detailed setting of each variant model is as follows.

- **MTPool-M** is the MTPool framework with MemPool [14], which generate clustering centroids without involving the input graphs.
- **MTPool-D** is the MTPool framework with DiffPool [38], which trains two parallel GNNs to get node-level embeddings and cluster assignments.
- **MTPool-S** is the MTPool framework with SAGPool [16], which drops nodes from the input to pool the graph.

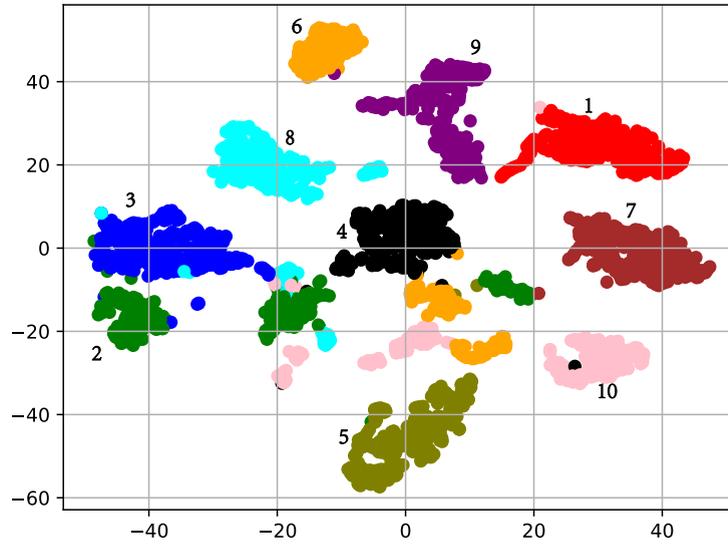


Figure 3: Class Prototype Inspection: visualize the 128-dimension multivariate time series embeddings learned for the PenDigits dataset in a two-dimensional image by t-SNE.

- **MTPool-One** is the MTPool framework with Variational Pooling and all-one adjacency matrix.
- **MTPool-Corr** is the MTPool framework with Variational Pooling and correlation coefficient adjacency matrix.
- **MTPool** is the MTPool framework with our proposed Variational Pooling and dynamic adjacency matrix.

Figure 2 shows the comparison results. The important conclusions of these results are as follows:

- (1) Different hierarchical graph pooling methods can be incorporated in our MTPool framework and can achieve comparable or better performance than state-of-the-art MTSC methods.
- (2) Different adjacency matrices can be used in our MTPool framework. However, the all-one matrix's performance is slightly worse than the correlation coefficient matrix, and our dynamic matrix achieves the best.
- (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 dynamic adjacency matrix.

4.4. Inspection of Class Prototype

This section visualizes the class prototype and its corresponding time series embedding to prove our well-trained time series embedding effectiveness. We use the t-SNE algorithm [18] to visualize the 128-dimensional time series embedded in the form of two-dimensional images. We use different colors to distinguish different categories. Figure 3 shows the embeddings learned for the PenDigits dataset, containing 3498 test samples in 10 different types. The following conclusions can be drawn from the results:

- (1) The distance between data samples from different categories is much greater than the distance between data samples from 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, we see that categories two, six, and ten are not divided into complete pieces. Instead, these three categories are all divided into several sub-parts. Thus, it helps identify problems with the classifier and take further measures, such as adding more training samples in these three classes.

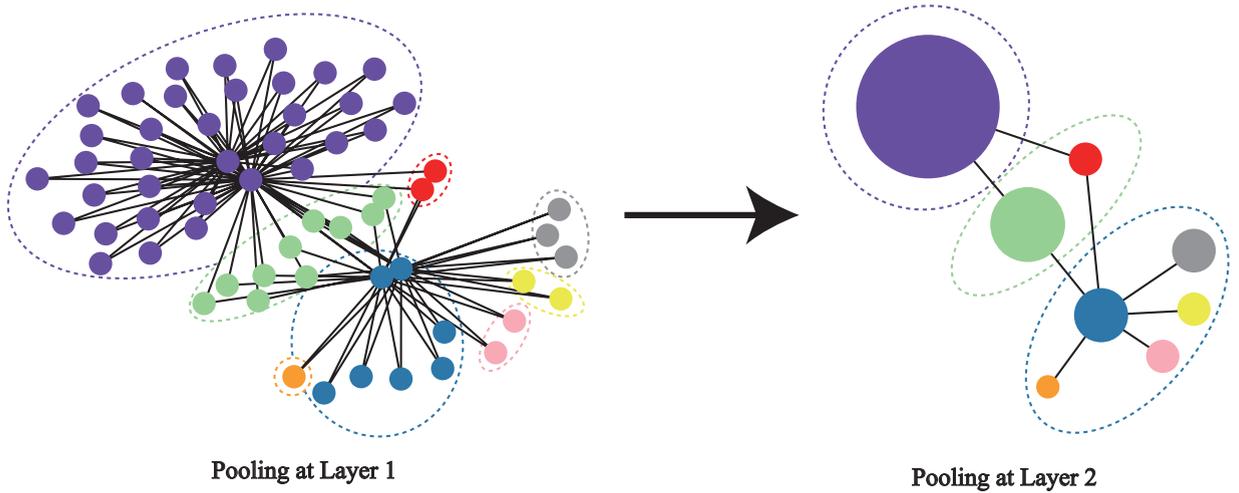


Figure 4: Visualization of the graph pooling process in MTPool, using example graphs from Heartbeat dataset, which has 61 MTS variables, so the original graph has 61 nodes. Nodes in the second layer correspond to clusters in the first layer. We use the same color to represent nodes of the same cluster and use dotted lines to indicate different clusters.

4.5. Case Study: Visualizations

In this section, we visualize the graph pooling process by using the Heartbeat dataset. Figure 4 shows a visualization of node assignments in the first and second layers on a graph constructed from the Heartbeat dataset. We use the same color to represent nodes of the same cluster. The cluster membership of each node is determined by the argmax of its cluster assignment probabilities. We also observed that even if the final goal is to get the graph-level embedding and get 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. It is worth mentioning that 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 is 15 (greater than 8), but in fact, we get 8 clusters. This reminds us that even if we define the expected cluster number in advance, MTPool will automatically perform clustering to get the best coarser results suitable for this graph. Such characteristics can be adjusted for different inputs, thus having a strong generalization ability.

5. Conclusion

In this paper, we propose the first graph pooling-based framework, MTPool, for MTS classification. It can explicitly model the pairwise dependencies among MTS variables and attain global embedding with strong interpretability and expressiveness. Experimental results demonstrate our model achieves state-of-the-art performance in the existing models.

Future research is promising to explore and design 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.

6. Acknowledgments

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

References

- [1] Bagnall, A., Dau, H.A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., Keogh, E., 2018. The uea multivariate time series classification archive, 2018. arXiv preprint arXiv:1811.00075 .

- [2] 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 .
- [3] 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.
- [4] Baldán, F.J., Benítez, J.M., 2019. Distributed fastshapelet transform: a big data time series classification algorithm. Information Sciences 496, 451–463.
- [5] Chen, J., Ma, T., Xiao, C., 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv preprint arXiv:1801.10247 .
- [6] Defferrard, M., Bresson, X., Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering, in: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 29. Curran Associates, Inc., pp. 3844–3852.
- [7] 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 .
- [8] 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, 106580.
- [9] Gao, H., Ji, S., 2019. Graph u-nets. arXiv preprint arXiv:1905.05178 .
- [10] Hamilton, W.L., Ying, R., Leskovec, J., 2017. Inductive representation learning on large graphs. CoRR abs/1706.02216. arXiv:1706.02216.
- [11] 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, 104971.
- [12] Kang, H., Choi, S., 2014. Bayesian common spatial patterns for multi-subject eeg classification. Neural Networks 57, 39–50.
- [13] Karim, F., Majumdar, S., Darabi, H., Harford, S., 2019. Multivariate lstm-fcns for time series classification. Neural Networks 116, 237–245.
- [14] Khasahmadi, A.H., Hassani, K., Moradi, P., Lee, L., Morris, Q., 2020. Memory-based graph networks, in: International Conference on Learning Representations.
- [15] Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. CoRR abs/1609.02907. arXiv:1609.02907.
- [16] Lee, J., Lee, I., Kang, J., 2019. Self-attention graph pooling. arXiv preprint arXiv:1904.08082 .
- [17] 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, 106508.
- [18] Maaten, L.v.d., Hinton, G., 2008. Visualizing data using t-sne. Journal of machine learning research 9, 2579–2605.
- [19] 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.
- [20] Morris, C., Ritzert, M., Fey, M., Hamilton, W.L., Lenssen, J.E., Rattan, G., Grohe, M., 2019. Weisfeiler and leman go neural: Higher-order graph neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 4602–4609.
- [21] 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, 920–931.
- [22] Ranjan, E., Sanyal, S., Talukdar, P.P., 2020. Asap: Adaptive structure aware pooling for learning hierarchical graph representations., in: AAAI, pp. 5470–5477.
- [23] Ruiz, A.P., Flynn, M., Bagnall, A., 2020. Benchmarking multivariate time series classification algorithms. arXiv preprint arXiv:2007.13156 .
- [24] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2008. The graph neural network model. IEEE Transactions on Neural Networks 20, 61–80.
- [25] Schäfer, P., Leser, U., 2017. Multivariate time series classification with weasel+ muse. arXiv preprint arXiv:1711.11343 .
- [26] 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, IEEE. pp. 1399–1406.
- [27] 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, 106523.
- [28] 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, 1351–1366.
- [29] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., 2017. Graph attention networks. arXiv preprint arXiv:1710.10903 .
- [30] Wang, Y., Duan, Z., Huang, Y., Xu, H., Feng, J., Ren, A., 2020. Mthetgcn: A heterogeneous graph embedding framework for multivariate time series forecasting. arXiv e-prints , arXiv:2008.
- [31] 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, pp. 10061–10062.
- [32] 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), IEEE. pp. 1578–1585.
- [33] 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 .
- [34] Xu, H., Chen, R., Bai, Y., Duan, Z., Feng, J., Sun, Y., Wang, W., 2020a. Cosimgnn: Towards large-scale graph similarity computation. arXiv preprint arXiv:2005.07115 .
- [35] Xu, H., Duan, Z., Wang, Y., Feng, J., Chen, R., Zhang, Q., Xu, Z., 2021. Graph partitioning and graph neural network based hierarchical graph matching for graph similarity computation. Neurocomputing 439, 348–362.
- [36] Xu, H., Huang, Y., Duan, Z., Wang, X., Feng, J., Song, P., 2020b. Multivariate time series forecasting with transfer entropy graph. arXiv e-prints , arXiv:2005.

- [37] Xu, K., Hu, W., Leskovec, J., Jegelka, S., 2018. How powerful are graph neural networks? ArXiv abs/1810.00826.
- [38] 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.
- [39] Yu, Z., Lee, M., 2015. Real-time human action classification using a dynamic neural model. Neural Networks 69, 29–43.
- [40] Zhang, X., Gao, Y., Lin, J., Lu, C.T., 2020. Tapnet: Multivariate time series classification with attentional prototypical network., in: AAAI, pp. 6845–6852.
- [41] 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, 106467.
- [42] Zhou, Y., Duan, Z., Xu, H., Feng, J., Ren, A., Wang, Y., Wang, X., 2020. Parallel extraction of long-term trends and short-term fluctuation framework for multivariate time series forecasting. arXiv preprint arXiv:2008.07730 .