2023 IEEE International Conference on Big Data and Smart Computing (BigComp) | 978-1-6654-7578-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/BigComp57234.2023.00050

Temporal Convolutional Network-Based Time-Series Segmentation

Hyangsuk Min*†, Jae-Gil Lee*
*KAIST, Korea
†HL Mando, Korea
Email: hs_min@kaist.ac.kr / hyangsuk.min@hlcompany.com, jaegil@kaist.ac.kr

***Abstract*—Time-series segmentation is useful to identify the underlying characteristics of time series and summarize time series as a sequence of states. Partitioning time series into the states makes the complex time series easily understandable and interpretable. However, as labels for time points are not normally available, it is challenging to figure out the accurate segments and their states. Therefore, we propose an unsupervised time-series segmentation using the inherent properties in times series. The states can be characterized by diverse length patterns inherent in time series, and thus capturing diverse patterns is crucial in an unsupervised time-series segmentation. We adopt a temporal convolutional network (TCN) as our key component to learn diverse length patterns since the intermediate layers in TCN contain both short and long patterns hierarchically. In this paper, we propose a novel unsupervised time-series segmentation *TCTS*, which is featured with the joint optimization of two modules. The TCN-based pattern learning module aims to grasp diverse length patterns that are characterized differently by the states, while the clustering-based classification module improves the separability of the representations between the states. We conduct experiments by comparing several baselines with multiple datasets and demonstrate the superiority of *TCTS*.**

***Index Terms*—Time series, segmentation, temporal clustering, unsupervised learning**

I. Introduction

Time series are generated in a wide variety of industries, including finance, healthcare, medicine, and smart factories [11], [29]. Time series are time-stamped observations with multiple sensors. These time series have temporally related to repeated states and can be summarized as a sequence of multiple states. Summarizing time series into multiple states helps understand underlying complex features easily. In the field of healthcare [29], body-worn wearable sensors help monitor the health of the elderly living alone by checking abnormalities of behaviors over time. For example, in Figure 1, time series with three features are collected through body-worn sensors on an ankle, a chest, and a wrist. The sensors record the different trends dependent on activities (states), each of which continues for a while. Since an activity repeats the same movements, the sensor creates a value that repeats for each activity. When a person stands still on the ground, the values are constant and do not fluctuate, and repeated patterns are easily observed. On the contrary, when a person starts moving, e.g., walking and running, the values are recorded differently.

• Jae-Gil Lee is the corresponding author.

Fig. 1: An example of time-series segmentation. There are three time series collected by wearable sensors which are worn on a wrist, an ankle, and a chest; four states, standing, jumping, walking, and running, have occurred.

Running triggers the values to oscillate a lot. In other words, 'standing' and 'running' generate patterns of different lengths. The standing has a shortly repetitive pattern as the movements are static, but running generates a fluctuating long pattern. Therefore, time series can be summarized as a sequence of states since values are collected in different repeated patterns according to activities (states).

Time series have multiple underlying distinctive properties depending on states. The properties can be defined in diverse patterns appearing over time. A state has a distinctive feature, which is a mixture of varying length patterns. These features are key components to figuring out the states in long time series. Thus, all of the states are composed of various length patterns and, therefore, are quite complex. There are plenty of applications in the field of time-series segmentation such as fitness tracking, network attack detection, sleep stage classification, and hand gesture detection [7], [13], [15]. However, the labels for each timestamp are generally not available, and the process of manually annotating labels for timestamps involves a lot of human labor. Finding the segments is challenging without labels for each timestamp because we cannot determine the start and finish points of a segment.

To overcome the label deficiency, there is an unsupervised time-series segmentation to identify the underlying properties of the time series. Time-series segmentation is one of the most promising research in time-series studies. Temporal clustering [9] is the one of methods in time-series segmentation. Temporal clustering partitions time series into non-overlapping multiple segments and clusters similar segments into the same groups. There are distance-based approaches and model-based approaches, both of which cluster a subsequence into a group and result in segments.

978-1-6654-7578-5/23/$31.00 ©2023 IEEE
DOI 10.1109/BigComp57234.2023.00050

Distance-based methods use k-means and distance metrics such as Euclidean and dynamic time warping (DTW) to calculate the distance between cluster centroids and timestamps and assign a state with a minimum distance [19]. Model-based methods such as the Gaussian mixture model (GMM) and Toeplitz inverse covariance-based clustering (TICC) adopt statistical properties such as mean, variance, and precision matrix and use maximum likelihood estimation to find the optimal state for a subsequence [8], [15].

However, existing studies are unaware of complex patterns appearing differently depending on the state. In Figure 1, the time series comprises both short and long patterns, and the pattern lengths are diverse depending on the states. Therefore, finding the optimal subsequence length is essential to have good segmentation quality. However, since existing studies use a fixed-length subsequence during the clustering process, they have several limitations.

First, using a short subsequence causes fragmentation problems. The fragmentation problem causes one state to break into multiple meaningless pieces and induces a chain effect confusing state allocation of the next subsequence. To prevent the fragmentation problem, there can be an effort to widen the subsequence length to cover long patterns. However, a long subsequence produces another challenge. First, learning a long subsequence results in the absence of time series transition points. The transition point is the time at which the state changes. As a result of the long subsequence's emphasis on recognizing long patterns as a whole, short patterns are likely forgotten. Second, because both distance-based and model-based approaches learn the representations whose size is given by a subsequence, they need to learn more parameters for longer subsequences.

To summarize, a fixed length of subsequences brings about three limitations: segment fragmentation, missing transition points, and high computational complexity. Consequently, identifying a mixture of varying length patterns is necessary to figure out the states correctly. Otherwise, time series can be partitioned into segments wrongly or broken into multiple meaningless fragments.

A temporal convolutional network (TCN) [17] is the most adequate method to solve the limitations of the previous studies. TCN is composed of multiple dilated convolutional layers. A dilated convolutional layer introduces a dilation rate that makes a space within a kernel. As stacking the multiple dilated convolutional layers, the receptive field increases, and thus, each layer has a receptive field of a different length. By mixing the results of each layer in TCN, both short and long patterns can be captured simultaneously, and thus, we can prevent fragmentation problems and properly capture transition points. Additionally, since TCN uses a shared kernel to learn long time series, TCN has fewer parameters than fully connected layers and RNN-based layers, thus reducing computational complexity.

Contributions of this paper are summarized as follows:

- We propose a novel unsupervised **T**emporal **C**onvolutional network-based **T**ime-series **S**egmentation, called ***TCTS***, which figures out varying length patterns inherent in time series without data annotation.
- We introduce TCN-based pattern learning to learn distinctive patterns depending on states and improve efficiency. Additionally, we adopt a pooling network to support TCN and clustering-based classification to make the representation suitable for clustering. TCN-based pattern learning and clustering-based classification are jointly optimized.
- We conduct various experiences based on three benchmark datasets for performance evaluation. The results show that *TCTS* outperforms baselines including time-series representation methods.

II. Related Work

In this section, we discuss previous studies related to unsupervised time-series segmentation. We specifically focus on research about temporal clustering. Temporal clustering is categorized into two approaches: distance-based approaches and model-based approaches. Additionally, there have been attempts to apply deep learning to time-series clustering. We first introduce distance-based approaches and model-based approaches and then explain deep time-series clustering.

A. Unsupervised Time-Series Segmentation

Distance-based approaches are dependent on the distance function to calculate the distance to centroids. A subsequence with a length of the window is considered as one vector in Euclidean space, and the method computes a distance between vectors and multiple centroids using a given distance function. k-means [27] is the most frequently used clustering algorithm and uses the Euclidean distance metric to find the optimal cluster. There were variants of the original k-means to adjust centroids more adaptable to time series. DTW barycenter averaging combined with k-means (k-DBA) was proposed for better alignment in time series, which adopts dynamic time warping (DTW) for barycenter averaging to overcome pairwise averaging of Euclidean distance [19], [27]. Also a variant of k-DBA, soft dynamic time warping (soft-DTW) [5] was proposed to consider more possible alignments unlike DTW considers only the optimal path. k-Spectral centroid clustering (KSC) [32] was introduced with a different shape-based distance measure that offers simultaneously pairwise scaling and shifting of time series. However, pairwise scaling and shifting were effective for a limited number of datasets. k-Shape [18] was proposed to focus on the shape of time series to capture similarities by calculating distance using DTW instead of focusing on real values. Kernel k-means [6] was proposed to overcome the limitations of k-means to non-linear separable data. Kernel k-means first projects time series onto a high-dimensional kernel space and performs k-means on this space. Global alignment kernel (GAK) [4] was introduced as the most adequate kernel to time series. However, distance-based approaches are sensitive to outliers and noise and computationally expensive since all time points are taken into account. Additionally, distance-based approaches are incapable

of identifying diverse length patterns in time series and tend to split long patterns into multiple pieces.

Model-based approaches are characterized to learn statistical properties for cluster representation. Learning the statistical properties often provides better results than distance-based approaches. Statistical properties are informative and flexible since they define cluster representation as a distribution and a hidden state. Simply calculating distances between time points and centroids is incapable of considering the complex relationships between features and correlations.

One of the most generally used clustering algorithms is the Gaussian mixture model (GMM) [15]. It assumes that time series is composed of a mixture of Gaussian distributions and uses multiple Gaussian distributions as a cluster representation. Each cluster is represented by a Gaussian distribution, and GMM calculates the maximum likelihood with a subsequence to find out which cluster is the most adequate to be involved. Auto-regressive moving average mixture model (ARMA mixture model) [31] was proposed to overcome time-series clustering with variable lengths using a mixture of an auto-regressive moving average. ARMA mixture model assumes time series is generated by k different ARMA models, and k points out the number of clusters. Hidden markov model (HMM) [24] uses internal hidden states which are not observed directly as cluster representation. Toeplitz inverse covariance-based clustering (TICC) [8], which is the state of the art, uses an inverse covariance matrix with Toeplitz constraints as a cluster representation. The Toeplitz constraint helps to reduce the computational complexity since it forces relationships between t and $t+1$ to be the same as $t+1$ and $t+2$, which partially restricts the ability to express time series. TICC focuses more on structure similarity between features in time series as an inverse covariance matrix contains the partial correlation between features at the same time points and different time points. However, these methods have shown several limitations; a fixed length of subsequences gives fragmentation results because of ignoring long patterns, and computational complexity increases depending on the subsequence length.

B. Deep Time-Series Clustering

Deep time-series clustering normally comprises two steps: feature extracting using deep learning and clustering. Features are initially extracted from time series using a deep learning model, and clustering algorithms are applied to the extracted features. Extracting features reduces the impact of noise or outliers and decreases the dimension of time series. Deep temporal clustering (DTC) [22] was the first study applying deep learning in temporal clustering. DTC consists of an auto-encoder and a clustering algorithm to learn representation with non-linearity in time series. The auto-encoder uses 1D-CNN and BI-LSTM to learn short-term features and latent representation. The clustering algorithm uses KL-divergence to compare similarities between the predicted and target distribution. Target distribution is created by sharpening the predicted distribution, which means that KL-divergence aims to improve the cohesion within clusters. Deep temporal clustering representation (DTCR) [16] was followed, which is more stable than DTC. Since DTC generates target distribution using the predicted distribution and the predicted distribution using the learned representation, the results fully depend on the performance of the auto-encoder. DTCR consists of the seq2seq model, classification model, and k-means objective and targets to learn cluster-oriented temporal representations. DTCR uses the k-means objective function to indirectly employ the separability and cohesion of k-means. Moreover, DTCR generates fake samples and classifies fake and real samples to improve the ability of the encoder.

However, DTC and DTCR focus on learning representation and clustering the whole time series. Therefore, DTC and DTCR are not adequate to split one long time series into multiple segments. Accordingly, they cannot be directly applied to our problem setting. Our proposed model, *TCTS*, targets to segment one long time series and clusters them correctly by figuring out distinctive patterns depending on the states. Therefore, an additional segmentation process needs to be included in DTC and DTCR to compare with *TCTS*. To the best of our knowledge, *TCTS* is the first to attempt to perform time-series segmentation and clustering simultaneously using deep neural networks.

III. Methodology: *TCTS*

A. Problem Definition

Given a time series X,

$$X = [x_1, x_2, \ldots, x_T],$$

where x_i is the i-th observation, and $x_i = \{x_i^1, x_i^2, \ldots, x_i^N\}$ has N features. Our goal is to partition a time series into multiple non-overlapping segments $S = \{s_1, s_2, \ldots, s_M\}$, where the length of s_j is arbitrary, and cluster S into K states, $C = \{c_1, c_2, ..., c_K\}$, where $c_k = \{s_1, \ldots, s_n\}$.

Instead of clustering adjacent time points together to consider diverse patterns, we focus on single time point clustering and seek the optimal latent representation that can include unique characteristics for each state. The latent representation is learned by deep learning in an unsupervised way and designed to contain distinct patterns inherent in each state over time. The latent representation is informative enough not to use neighboring time points together to consider the varying length patterns during clustering, which makes clustering converges faster. Additionally, the latent representation automatically encourages the adjacent time points to belong to the same state and makes up multiple long segments. Consequently, our proposed method concentrates on learning the latent representation that includes temporal context and unique features for the states and simultaneously segments and clusters time series.

B. Overview

In this section, we describe our model, *TCTS*. The overview of the model architecture is depicted in Figure 2. The overall architecture consists of two modules. First, the TCN-based pattern learning module learns a latent representation for every

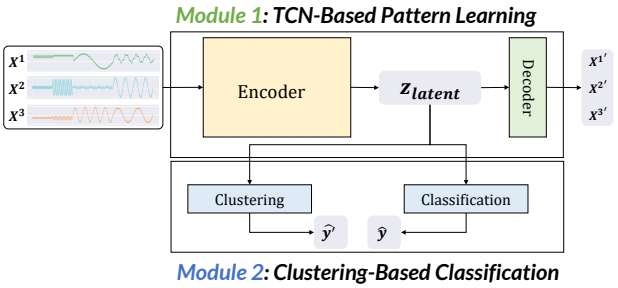

Fig. 2: Overall architecture of *TCTS*.

time point and each time point compactly represents the characteristics inherent in the state. Second, the clustering-based classification module improves the separability of the latent representations between the states and forces them to have unique representations.

C. Module 1: TCN-Based Pattern Learning

In Figure 3, the TCN-based pattern learning module consists of two networks, an encoder network and a decoder network. The encoder network has the Temporal convolutional network (TCN) [17] and the pooling network [23], and the decoder network has 1D convolutional layers. The encoder network receives time series as an input and generates the latent representation, which is imported into the decoder network to reproduce the original time series.

The TCN-based pattern learning module aims to capture the repeated patterns intrinsic in time series differently depending on states and generate the optimal latent representation for every state. Since the patterns have diverse lengths and shapes, considering time series in the multi-resolution is necessary to learn the optimal latent representation.

*1) **Temporal convolutional network (TCN)**:* TCN learns long patterns by stacking multiple 1D dilated convolutional layers with exponentially doubled dilation rates, $i.e.$, $1, 2, 4, 8, \ldots, 128$. The 1D dilated convolutional layer introduces a dilation rate, which defines the spaces within kernels. Increasing the dilation rate can broaden the view in the time dimension with a relatively short kernel. The 1D dilated convolutional layer with kernel size 3 and dilation rate 2 is the same as the 1D convolutional layer with kernel size 5. Therefore, stacking multiple dilated convolutional layers learns a long subsequence with relatively fewer layers than stacking multiple normal convolutional layers, thus needing much fewer parameters. A layer in TCN consists of a dilated convolutional layer, a batch normalization, a Relu activation function, and a dropout layer to make the learning process stable. Therefore, each layer is formulated as follows. (A batch normalization and a dropout layer are omitted for ease of exposition.)

$$H_l = ReLU(W_l * H_{l-1} + b_l), \tag{1}$$

where H_l is the output of l-th layer in TCN, $*$ denotes the convolution operator, $W_l \in \mathbf{R}^{F \times N \times D}$ are the weights of dilated convolution filters with a kernel size F, the number of channels N, and the number of convolutional filters D.

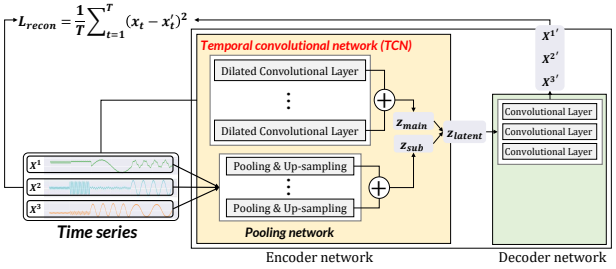

Fig. 3: Module 1: TCN-based pattern learning.

A layer has receptive fields of different lengths and learns patterns of different lengths. Since we target to learn diverse patterns occurring assorted depending on what states are activating, using the results of layers is helpful to capture diverse length patterns. We stack all the layers' outputs, average with learned weights by our model, and generate the optimal latent representations similar within the same states and discriminative between the states.

$$z_1 = H \cdot \alpha \in \mathbb{R}^{T \times D}, \tag{2}$$

where $H = [H_1, H_2, \ldots, H_L]^\top \in \mathbb{R}^{T \times D \times L}, \alpha \in \mathbb{R}^L$, T is the number of timestamps, and L is the number of layers.

*2) **Pooling network**:* A pooling network follows the architecture of temporal pyramid pooling. A pooling network mainly smooths time series and provides a general trend of each feature. A pooling network additionally learns patterns of diverse lengths for each feature by applying a pooling to features individually. A pooling summarizes long time series into short time series and keeps the important information.

$$\mathrm{P}_l = [\mathrm{Upsampling}_l(\mathrm{Pooling}_l(X_1)), \ldots, \mathrm{Upsampling}_l(\mathrm{Pooling}_l(X_N))]^\top \in \mathbb{R}^{T \times N}, \tag{3}$$

where P_l is the output of a l-th layer in the pooling network, T is the number of timestamps, and N is the number of features. Each feature has a max pooling layer applied, and an up-sampling layer is followed to match the length of time. The pool size decreases exponentially double, $i.e\ T, T/2, \ldots, T/16$. Summarizing long time series into short time series using multiple pool sizes helps recognize varying length patterns. After implementing the pooling network, all outputs are stacked and averaged. The results summarize and highlight the characteristic patterns of states.

$$z_2 = \frac{1}{L}\Sigma_1^L P \in \mathbb{R}^{T \times N}, \tag{4}$$

where $P = [P_1, P_2, \ldots, P_L]^\top \in \mathbb{R}^{T \times N \times L}$. After time series pass through the TCN and pooling network individually, the results are concatenated and transmitted to a 1D convolutional layer with kernel size = 3 and filter size = H to produce the final representation.

$$z = Conv1D([z_1, z_2]) \in \mathbb{R}^{T \times H}, \tag{5}$$

where H is the latent representation dimension. H is smaller than the dimension of the original time series. The final

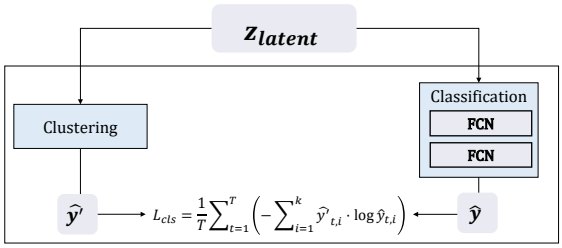

Fig. 4: Module 2: Clustering-based classification.

representation z compactly learns varying length patterns and contains distinctive representations depending on the states. Therefore, z is informative to partition time series into non-overlapped and non-fragmented segments with accurate transition points.

*3) **Decoder network**:* The decoder network aims to reproduce the original time series. Recreating the original time series correctly is necessary to contain the inherent characteristics in the representation since the time series is composed of diverse length patterns that appear differently depending on what states are activated. To achieve the goal, the decoder network has multiple 1D convolutional layers (1D-CNN) to regenerate the original time series. 1D-CNN is powerful to gather temporally spread information efficiently. We use three 1D-CNN layers with the kernel size set to 3 and the filter size set to N, which is the original time-series dimension, for the last layer and D for the remaining layers. Additionally, the residual connection from the previous layer is added to the decoder network.

$$\hat{H}_l = ReLU(W_l * \hat{H}_{l-1} + b_l) + \hat{H}_{l-1}, \tag{6}$$

where $\hat{H}_l$ is an output of the l-th 1D-CNN layer, and W_l is the kernel weight of the l-th 1D-CNN layer. Therefore, the final reconstructed time series is constructed as follows.

$$X' = \hat{H}_3 = ReLU(W_3 * \hat{H}_2 + b_3) + \hat{H}_2, \tag{7}$$

D. Module 2: Clustering-Based Classification

In Figure 4, a clustering-based classification consists of a clustering algorithm and a classification layer. Both layers receive the latent representation from Equation (5) and return clustering and classification results respectively. The clustering uses algorithms such as k-means and Toeplitz inverse covariance-based clustering (TICC) in an unsupervised way. The classification layer is trained with pseudo-labels from the clustering in a supervised way and targets to imitate the clustering results. Therefore, clustering-based classification helps z to be separable between different states and cohesive within the same states, which learns the discriminative power of the clustering algorithm. We first introduce the clustering part and then explain the classification layer.

*1) **Clustering**:* Any clustering algorithm can be utilized such as k-means and TICC. In *TCTS*, we specifically choose TICC [8]. Since TICC is the state of the art in temporal clustering, it shows the best performance among other approaches. We use TICC with a window size set to 1, which means that the number of parameters is small enough not to be taken into account, and a Toeplitz constraint, which restricts learning diverse length patterns, is ignored. The clustering algorithm receives the latent representations z and the clustering outputs of the previous training epoch and returns the clustering results adapted to the new latent representation starting from the previous outputs.

$$\hat{y}_s^{'} = Clustering(z, \hat{y}_{s-1}^{'}), \tag{8}$$

where $\hat{y}_s^{'}$ is the result of the clustering, and s means the training epoch. These clustering results affect the latent representation update indirectly and help to improve the separability of the latent representation by collaborating with the classification layer, which is introduced below.

*2) **Classification layer**:* The classification layer contains two fully connected layers and a softmax activation function. The classification layer receives the latent representation z as an input as in the clustering algorithm.

$$\hat{y} = argmax\{Softmax(FCN(Softmax(FCN(z))))\}. \tag{9}$$

The classification layer aims to predict the same results as the clustering outputs. The classification layer does not aim to improve classification performance but to decouple the latent representation, z, between the states. In other words, the classification layer assists to learn the clustering-oriented representation collaborating with clustering. Note that the clustering results are not consistent during training since the clustering does not explicitly result in the exact labels. The clustering provides information that the two points belong to the same cluster or different clusters. To enable the classification layer to learn the weights to imitate the clustering algorithm, we adjust the results of the clustering by hashing to maintain the implicit orders.

As a result, the clustering-based classification module is jointly optimized with the TCN-based pattern learning module since the latent representations are also affected by the results of the clustering-based classification module, and the newly updated latent representations pass through the clustering and the classification layer.

E. Objective Function

As an objective function, we use a combination of reconstruction loss and classification loss to jointly update module 1 and module 2. We use mean squared error as a reconstruction loss and cross-entropy as a classification loss. First, the reconstruction loss compares the original time series and a reproduced time series by the decoder.

Second, the classification loss compares the clustering results as true labels and classification outputs as predicted labels. Although the reconstruction error works well to learn the latent representations, we adopt clustering-based classification to make the latent representations more suitable for clustering. The final objective function is a combination of the aforementioned two losses,

TABLE I: Overall comparison with baselines with raw datasets and latent representation learned by an LSTM auto-encoder. k-means, k-DBA, GMM, and TICC trained with raw datasets and AE + $\{k$-means, k-DBA, GMM, TICC$\}$ trained with the latent representation.

	Metric	k-means	k-DBA	GMM	TICC	AE + k-means	AE + k-DBA	AE + GMM	AE + TICC	***TCTS***
mHealth	NMI	0.74 (±0.01)	0.69 (±0.02)	0.79 (±0.02)	0.83 (±0.01)	0.59 (±0.04)	0.60 (±0.04)	0.77 (±0.02)	0.78 (±0.05)	**0.94 (±0.03)**
	ARI	0.59 (±0.00)	0.53 (±0.03)	0.65 (±0.03)	0.63 (±0.02)	0.44 (±0.04)	0.44 (±0.05)	0.62 (±0.03)	0.52 (±0.08)	**0.89 (±0.00)**
	F1	0.61 (±0.02)	0.57 (±0.04)	0.65 (±0.05)	0.64 (±0.06)	0.51 (±0.04)	0.52 (±0.06)	0.66 (±0.03)	0.40 (±0.08)	**0.89 (±0.07)**
	ACC	0.64 (±0.02)	0.63 (±0.03)	0.69 (±0.04)	0.72 (±0.01)	0.57 (±0.03)	0.58 (±0.05)	0.70 (±0.03)	0.55 (±0.07)	**0.92 (±0.05)**
PAMAP2	NMI	0.64 (±0.02)	0.66 (±0.03)	0.64 (±0.03)	0.71 (±0.00)	0.62 (±0.02)	0.61 (±0.01)	0.62 (±0.02)	0.75 (±0.04)	**0.81 (±0.02)**
	ARI	0.50 (±0.03)	0.51 (±0.05)	0.50 (±0.04)	0.55 (±0.01)	0.46 (±0.02)	0.46 (±0.02)	0.44 (±0.02)	0.57 (±0.07)	**0.72 (±0.02)**
	F1	0.52 (±0.01)	0.55 (±0.03)	0.54 (±0.05)	0.55 (±0.03)	0.46 (±0.03)	0.50 (±0.01)	0.50 (±0.01)	0.56 (±0.05)	**0.68 (±0.02)**
	ACC	0.56 (±0.02)	0.60 (±0.03)	0.60 (±0.04)	0.63 (±0.01)	0.54 (±0.02)	0.56 (±0.01)	0.55 (±0.02)	0.66 (±0.04)	**0.76 (±0.02)**
SKODA	NMI	0.40 (±0.00)	0.42 (±0.02)	0.27 (±0.00)	0.64 (±0.04)	0.36 (±0.02)	0.35 (±0.03)	0.38 (±0.04)	0.75 (±0.07)	**0.89 (±0.03)**
	ARI	0.23 (±0.00)	0.26 (±0.02)	0.14 (±0.00)	0.37 (±0.02)	0.22 (±0.02)	0.20 (±0.02)	0.24 (±0.04)	0.58 (±0.11)	**0.86 (±0.02)**
	F1	0.41 (±0.00)	0.49 (±0.03)	0.33 (±0.02)	0.44 (±0.08)	0.39 (±0.04)	0.37 (±0.06)	0.41 (±0.07)	0.43 (±0.08)	**0.82 (±0.01)**
	ACC	0.42 (±0.00)	0.48 (±0.02)	0.36 (±0.03)	0.57 (±0.04)	0.42 (±0.02)	0.40 (±0.04)	0.44 (±0.05)	0.62 (±0.07)	**0.88 (±0.01)**

$$L = \lambda L_{recon} + (1 - \lambda) L_{cls}, \tag{10}$$

where λ balances between the reconstruction error and the classification loss for segmentation results.

IV. Evaluation

A. Dataset

We use three datasets to demonstrate that *TCTS* adaptively learns the varying length patterns. mHealth and PAMAP2 are related to healthcare monitoring, and SKODA is about a smart factory. The three datasets have different patterns depending on their states, and the patterns have diverse lengths.

- **mHealth** [1], [2] comprises body motion and vital signs recorded while performing physical activities, such as standing and climbing stairs. The dataset was collected using a 50Hz smartphone sensor worn around a waist. It has 35174 time points and 12 states.
- **PAMAP2** [20], [21] contains the physical activity of aging people. The dataset was collected using 100Hz inertia measurement units and a 9Hz heart rate monitor worn around a wrist, a chest, and an ankle. It has 51843 time points and consists of 12 different physical activities.
- **SKODA** [25] consists of the assembly-line workers' movement in a car production scenario. The dataset was collected using 100Hz sensors worn around arms. It has 107951 time points and eight different movements, such as opening and closing the engine hood and checking the trunk gap.

B. Evaluation Metrics

Following the common evaluation metrics selected by related studies [16], [33], we adopt normalized mutual information (NMI) [26] and adjusted rand index (ARI) [10]. Additionally, we use macro-F1 score (F1), and accuracy (ACC) [3], [30] with the Hungarian algorithm [12] to compare performances.

C. Baselines

Demonstrating the superiority of *TCTS*, we chose several unsupervised time-series segmentation algorithms that can be aligned with our problem definition. We ran baselines in two different ways. First, we ran k-means, k-DBA, GMM, and TICC with raw values. Second, we generated a time-series representation with an LSTM auto-encoder and ran four baselines mentioned above. The LSTM auto-encoder has three LSTM layers in both the encoder and decoder, and time-series representation is extracted from the last layer of the encoder.

D. Overall Performance Comparison

Table I presents the results of time-series segmentation grouped by three datasets in terms of NMI, ARI, F1, and ACC. From the third to sixth columns are the results with raw datasets, while from the seventh to tenth columns are the results with time-series representation.

First, k-means, k-DBA, GMM, and TICC give worse results than *TCTS*. Since the approaches are incapable of learning diverse patterns featured differently in states, the segmentation results are broken into small pieces resulting in low performance. Even if the model shows good performance in terms of NMI, they have relatively lower results in terms of ARI and F1. NMI concentrates on the purity of clusters, while ARI and F1 more focus on reducing the false positives and false negatives. ARI and F1 react more sensitive to imbalanced clustering results. TICC has the best performance among baselines with the mHealth dataset, but it has lower ARI and F1 compared to GMM implying that TICC has more imbalanced clustering results than GMM. Generally, *TCTS* shows the best performance on all four metrics. In the mHealth dataset, the ACC is improved by 27.78%–46.03%, in the PAMAP2 dataset, the ACC is improved by 20.06%–35.71%, and in the SKODA dataset, the ACC is improved by 54.38%–144.44%, which is an astonishing improvement.

Second, AE + k-means, AE + k-DBA, AE + GMM, and AE + TICC adversely give worse performance than the tests with raw datasets. Since the representation learning algorithm uses an auto-encoder only, it doesn't have enough ability to learn distinctive features in the state individually. Although the auto-encoder can learn long patterns easily, LSTM rather lumps some of the original characteristics inherent in time series. Therefore, hidden representations lose the clear differences and have obscure features which are not distinguished between the two states. The poorly learned representation harms the performance in most clustering algorithms.

TABLE II: Ablation study of module 2 in *TCTS*.

	Metric	TCTS (w/o Module 2)	TCTS	Degrade
mHealth	NMI	0.93 (±0.03)	0.94 (±0.03)	0.0097
	ARI	0.86 (±0.06)	0.89 (±0.00)	0.0315
	F1	0.83 (±0.06)	0.89 (±0.07)	0.0673
	ACC	0.89 (±0.05)	0.92 (±0.05)	0.0338
mHealth	NMI	0.78 (±0.04)	0.81 (±0.02)	0.0307
	ARI	0.66 (±0.08)	0.72 (±0.02)	0.0862
	F1	0.63 (±0.06)	0.68 (±0.02)	0.0825
	ACC	0.71 (±0.07)	0.76 (±0.02)	0.0752
SKODA	NMI	0.88 (±0.06)	0.89 (±0.03)	0.0159
	ARI	0.78 (±0.14)	0.86 (±0.02)	0.0974
	F1	0.80 (±0.04)	0.82 (±0.01)	0.0225
	ACC	0.84 (±0.06)	0.88 (±0.01)	0.0391

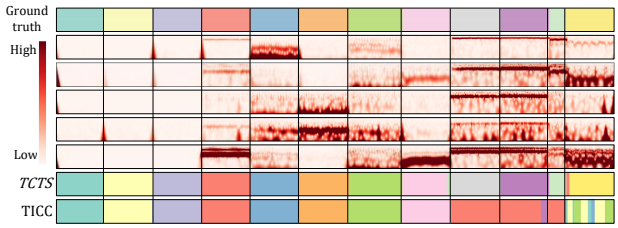

Fig. 5: The wavelet transform of mHealth datasets and segmentation results of *TCTS* and TICC.

E. Ablation Study

To figure out the fulfillment of module 2 proposed in Section III-D, we remove the classification layer of module 2 and break the interconnection between module 1 and module 2.

***TCTS* (w/o Module 2)** removes the classification layer of module 2. The loss function is revised to contain only reconstruction error. The training process consists of two parts, and the interconnection between two modules is interrupted. Therefore, module 1 is first trained in the same way as the baselines, and then, the clustering algorithm is performed. The performance of *TCTS* (w/o Module 2) is on average decreased by 2% in terms of NMI and 5% in terms of ACC, compared to *TCTS* in Table II. The result demonstrates the importance of joint optimization with a clustering algorithm.

F. Analysis

In this section, we discuss the qualitative results of our model, *TCTS*. We pointed out the three limitations of the previous work: segment fragmentation, missing transition points, and high computational complexity. We explore how the problems are solved by *TCTS* and describe the results in terms of accuracy and efficiency.

*1) **Accuracy**:* In Figure 5, the wavelet transform [14], [28] of five features in mHealth and the segmentation results are described. The wavelet transform shows the aspect of the patterns and how the pattern lengths differ according to the states. The location of the dark-colored rows indicates pattern length, and the dark-colored appearance at the top means a longer pattern. The dataset has varying length patterns since the dark color is appeared in different rows depending on the states. *TCTS* partitions all segments correctly compared to TICC. Since *TCTS* can adaptively learn varying length patterns, thus preventing fragmentation and capturing transition points. TICC missed the transition points between states in the third and fourth states from the back since TICC is incapable of understanding the mixture of varying length patterns and distinctive differences in the patterns between states. Moreover, TICC causes the fragmentation problem in the last segments since the last segment comprises much longer patterns which are the composite of short patterns. Consequently, *TCTS* prevents the fragmentation problem and captures the transition points exactly by adaptively learning diverse length patterns.

TABLE III: The number of model parameters and running time in seconds.

	Method	**Params**	**Converged Time**
mHealth	***TCTS***	$6.9K$	95s
	TICC	$28.7K$	418s
PAMAP2	***TCTS***	$14.4K$	193s
	TICC	$140.7K$	1398s
SKODA	***TCTS***	$15.0K$	200s
	TICC	$129.8K$	1177s

*2) **Efficiency**: TCTS* is composed of a 1D convolution layer and a pooling layer to learn short and long patterns, and has much fewer parameters than previous approaches such as TICC. In Table III, comparing the number of parameters which we use for PAMAP2 datasets, *TCTS* needs only $14.4K$ parameters whereas TICC needs $140.7K$ parameters. Since *TCTS* learns both short and long patterns using a short shared parameter, called a kernel in a convolution layer, we decrease the number of parameters. Additionally, we compress the underlying properties in time series appearing differently depending on the states into lower dimensional representation, and thus we use much fewer parameters. On average, we use 6.5 times fewer parameters for the three datasets. Due to fewer parameters, *TCTS* converges much faster than TICC. As a result, we solve the computational complexity problem and give clustering results much faster.

V. Conclusion

In this paper, we introduce *TCTS*, a novel unsupervised time-series segmentation. Based on time-series properties, we work on learning dynamically appearing patterns depending on the states. Depending on what states are behaved, the patterns have different lengths and amplitudes. To capture varying length patterns, we propose a TCN-based pattern learning module and a clustering-based classification module. We demonstrate that *TCTS* is aware of varying length patterns using the TCN-based pattern learning module by mixing from short to long patterns hierarchically and learning decoupled representation between the states using the clustering-based classification module. *TCTS* empirically show the superiority using three time-series datasets. Moreover, we show the segmentation results of mHealth datasets visually. Consequently, we demonstrate the performance of *TCTS* quantitatively and qualitatively.

ACKNOWLEDGMENT

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00862, DB4DL: High-Usability and Performance In-Memory Distributed DBMS for Deep Learning).

REFERENCES

[1] Oresti Banos, Rafael Garcia, Juan A Holgado-Terriza, Miguel Damas, Hector Pomares, Ignacio Rojas, Alejandro Saez, and Claudia Villalonga. mHealthDroid: a novel framework for agile development of mobile health applications. In *Proceedings of the 2014 International Workshop on Ambient Assisted Living (IWAAL)*, pages 91–98, 2014.

[2] Oresti Banos, Claudia Villalonga, Rafael Garcia, Alejandro Saez, Miguel Damas, Juan A Holgado-Terriza, Sungyong Lee, Hector Pomares, and Ignacio Rojas. Design, implementation and validation of a novel open framework for agile development of mobile health applications. *Biomedical Engineering Online*, 14(2):1–20, 2015.

[3] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586, 2010.

[4] Marco Cuturi. Fast global alignment kernels. In *Proceedings of the 2011 International Conference on Machine Learning (ICML)*, pages 929–936, 2011.

[5] Marco Cuturi and Mathieu Blondel. Soft-dtw: a differentiable loss function for time-series. In *Proceedings of the 2017 International Conference on Machine Learning (ICML)*, pages 894–903, 2017.

[6] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the 2004 ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 551–556, 2004.

[7] Tak-chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[8] David Hallac, Sagar Vare, Stephen Boyd, and Jure Leskovec. Toeplitz inverse covariance-based clustering of multivariate time series data. In *Proceedings of the 2017 ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 215–223, 2017.

[9] Minh Hoai and Fernando De la Torre. Maximum margin temporal clustering. In *Proceedings of the 2012 International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 520–528, 2012.

[10] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[11] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[12] Harold W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[13] Oscar D Lara and Miguel A Labrador. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys & Tutorials*, 15(3):1192–1209, 2012.

[14] Gregory Lee, Ralf Gommers, Filip Waselewski, Kai Wohlfahrt, and Aaron O'Leary. Pywavelets: A python package for wavelet analysis. *Journal of Open Source Software*, 4(36):1237, 2019.

[15] T Warren Liao. Clustering of time series data—a survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[16] Qianli Ma, Jiawei Zheng, Sen Li, and Gary W Cottrell. Learning representations for time series clustering. In *Advances in Neural Information Processing Systems 2019 (NeurIPS)*, pages 3776–3786, 2019.

[17] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[18] John Paparrizos and Luis Gravano. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM International Conference on Management of Data (SIGMOD)*, pages 1855–1870, 2015.

[19] François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678–693, 2011.

[20] Attila Reiss and Didier Stricker. Creating and benchmarking a new dataset for physical activity monitoring. In *Proceedings of the 2012 International Conference on PErvasive Technologies Related to Assistive Environments (PETRA)*, pages 1–8, 2012.

[21] Attila Reiss and Didier Stricker. Introducing a new benchmarked dataset for activity monitoring. In *Proceedings of the 2012 International Symposium on Wearable Computers (ISWC)*, pages 108–109, 2012.

[22] Naveen Sai Madiraju, Seid M Sadat, Dimitry Fisher, and Homa Karimabadi. Deep temporal clustering: Fully unsupervised learning of time-domain features. *arXiv preprint arXiv:1802.01059*, 2018.

[23] Dipika Singhania, Rahul Rahaman, and Angela Yao. Coarse to fine multi-resolution temporal convolutional network. *arXiv preprint arXiv:2105.10859*, 2021.

[24] Padhraic Smyth. Clustering sequences with hidden markov models. In *Advances in Neural Information Processing Systems 1996 (NeurIPS)*, pages 648–654, 1996.

[25] Thomas Stiefmeier, Daniel Roggen, Georg Ogris, Paul Lukowicz, and Gerhard Tröster. Wearable activity tracking in car manufacturing. *IEEE Pervasive Computing*, 7(2):42–50, 2008.

[26] Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

[27] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, et al. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6, 2020.

[28] Christopher Torrence and Gilbert P Compo. A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society*, 79(1):61–78, 1998.

[29] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, 2019.

[30] Mingrui Wu and Bernhard Schölkopf. A local learning approach for clustering. In *Advances in Neural Information Processing Systems 2006 (NeurIPS)*, pages 1529–1536, 2006.

[31] Yimin Xiong and Dit-Yan Yeung. Time series clustering with arma mixtures. *Pattern Recognition*, 37(8):1675–1689, 2004.

[32] Jaewon Yang and Jure Leskovec. Patterns of temporal variation in online media. In *Proceedings of the 2011 International Conference on Web Search and Data Mining (WSDM)*, pages 177–186, 2011.

[33] Hui Zhang, Tu Bao Ho, Yang Zhang, and M-S Lin. Unsupervised feature extraction for time series clustering using orthogonal wavelet transform. *Informatica*, 30(3), 2006.