Focus on Your Negative Samples in Time-Series Representation Learning

Seonggye Lee[1], **Pilsung Kang**[1*]

[1]Korea University, Seoul, Republic of Korea
{seonggye_lee, pilsung_kang}@korea.ac.kr

Abstract

With the advancement of deep learning, unsupervised time-series representation learning has become an active area of research. Various models have been proposed, and many of them are based on contrastive learning. The key factor of contrastive learning is the construction of appropriate positive and negative pairs. Most methods carefully construct positive pairs, but the construction of negative pairs is surprisingly simple. While the benefits of hard negative samples have been studied in computer vision (CV), natural language processing (NLP), and graph domains, there is a lack of research in the time-series domain. To address this issue, we propose the FoN (Focus on Negative samples) loss, a loss function that assigns training weights to negative samples that are similar to anchors through K-means clustering. We applied the FoN loss to the TS2Vec, TS-TCC, and CoST, to perform tasks involving multivariate forecasting and univariate/multivariate classification. The experimental results show an average performance improvement of 4.27% in terms of mean squared error for multivariate forecasting and 1.6%p in terms of accuracy for classification. Notably, performance improvements were observed across all models and tasks, indicating that the FoN loss is model-agnostic as well as task-agnostic.

Introduction

Time-series data holds significant importance in diverse fields such as finance (Sezer, Gudelek, and Ozbayoglu 2020), manufacturing (Wang et al. 2022), and healthcare (Seh et al. 2020). Due to the importance of time-series data, various studies were conducted to solve tasks such as forecasting (Zhou et al. 2021, 2022; Zeng et al. 2023), classification (Dempster, Petitjean, and Webb 2020; Chen et al. 2013), and anomaly detection (Xu et al. 2022; Shen et al. 2021). In addition to researching end-to-end models that target specific tasks, there were also active studies of time-series representation learning methodologies for generating high-quality representations (Franceschi, Dieuleveut, and Jaggi 2019; Tonekaboni, Eytan, and Goldenberg 2021; Yue et al. 2022; Eldele et al. 2021; Woo et al. 2022).

Similar to domains like computer vision (CV), natural language processing (NLP), and graph, many methods based on contrastive learning have been studied for learning time-series representations (Franceschi, Dieuleveut, and Jaggi 2019; Tonekaboni, Eytan, and Goldenberg 2021; Yue et al. 2022; Eldele et al. 2021; Woo et al. 2022). A key factor of contrastive learning is the construction of appropriate positive and negative samples (Jaiswal et al. 2020). To construct positive samples, methods such as using subseries of anchors (Franceschi, Dieuleveut, and Jaggi 2019), estimating distributions through statistical tests (Tonekaboni, Eytan, and Goldenberg 2021), and data augmentation (Yue et al. 2022; Eldele et al. 2021; Woo et al. 2022) are used. Especially, methods that use data augmentation propose methods to capture the characteristics of time-series. TS2Vec (Yue et al. 2022) employs hierarchical contrastive learning in both the instance and temporal domains. CoST (Woo et al. 2022) utilizes seasonal-trend decomposition. In the case of TS-TCC (Eldele et al. 2021), it performs cross-view contrasting on weak augmentation and strong augmentation. On the other hand, the construction of negative samples is surprisingly simple compared to positive samples. Negative samples are constructed by randomly selecting from data different from the anchor (Franceschi, Dieuleveut, and Jaggi 2019; Eldele et al. 2021), by selecting data with different time indexes or instances (Yue et al. 2022), and by using a memory bank (He et al. 2020) to contain negative samples (Woo et al. 2022). The aforementioned approaches utilize limited information regarding the negative sample, resulting in a substantial element of randomness in this process. To address this issue, the study was proposed in which the negative sample is chosen to be a sample with a large Euclidean distance from the anchor (Chang et al. 2022). This method can alleviate the sampling bias problem. However, the problem is that hard negative samples cannot be properly considered during the learning process. While studies have reported that hard negative samples can achieve high performance and fast convergence in CV, NLP, and graph domains (Robinson et al.

*Pilsung Kang is the corresponding author.

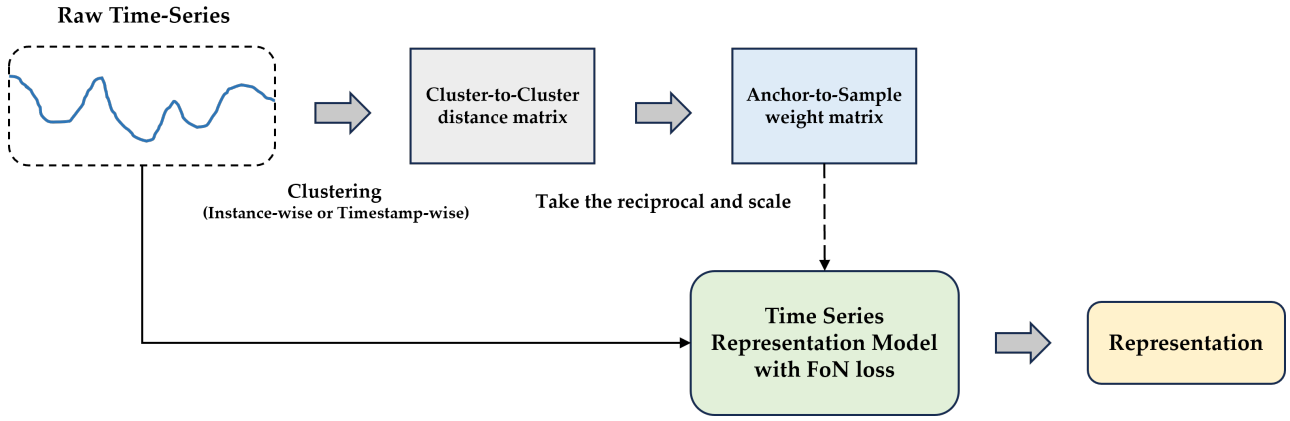

Figure 1: Framework of time-series representation learning with FoN loss. K-means clustering is performed on the raw time-series to obtain an Anchor-to-Sample weight matrix. Train time-series representation learning models with FoN loss using the Anchor-to-Sample weight matrix as a lookup table.

2021; Kalantidis et al. 2020; Xiong et al. 2021; Liu et al. 2023), there is a lack of research on hard negative samples in time-series representation learning.

To address this issue, we propose FoN (Focus on Negative samples) loss, a loss function that gives training weight to negative samples similar to anchors based on K-means clustering. K-means clustering is performed on the raw time-series, and the task determines which dimensions to cluster on. Time-series are usually composed of three dimensions: instance, timestamp, and feature, and the input and output change depending on which of these dimensions the clustering is performed on. Considering this, in this study, clustering was performed by determining the appropriate dimensions according to the task. For example, in the case of classification, the task is to classify which group each instance belongs to, so K-means clustering was performed based on the instance dimension. On the other hand, for the prediction task, K-means clustering was performed based on the timestamp dimension, since the task is to predict the time-series at a future point in time. Through this process, we obtain cluster information from raw time-series, calculate learning weights, and incorporate them into the contrastive learning loss to produce the FoN loss. Specifically, It calculates the training weight as the reciprocal of the distance between the centroid of the cluster to which the anchor belongs and the centroid of the cluster to which the negative sample belongs. By multiplying the calculated training weights by the similarity term of the contrastive loss, it is designed to reflect more negative samples close to the anchor in the representation learning. Since FoN loss does not use an additional hard negative sampling method, it is a universal loss function that can be easily applied to contrastive learning-based representation learning methods. We applied the FoN loss to contrastive learning-based time-series representation learning models, including TS2Vec (Yue et al. 2022), TS-TCC (Eldele et al. 2021), and CoST (Woo et al. 2022). All three models have publicly available official code and were utilized for univariate/multivariate classification and multivariate forecasting tasks. FoN loss consistently outperforms baseline models across combinations of models and tasks. This experimental evidence underscores its model-agnostic and task-agnostic nature. Looking at the specific performance improvements by task, we found 1.5%p improvement in average accuracy for univariate classification, 1.7%p improvement in average accuracy for multivariate classification, and 4.27% improvement in average MSE for multivariate forecasting. In addition, FoN loss can achieve higher performance with fewer training epochs compared to the baseline without FoN loss. We experimentally demonstrated that by utilizing only 50% of epochs in multivariate time-series classification tasks and 75% in univariate time-series classification and multivariate time-series prediction tasks, the FoN loss can outperform the baseline model. The contributions of this paper are as follows:

- We propose a loss function called FoN loss, which effectively reflects the similarity between anchor and negative samples through K-means clustering in contrastive learning-based time-series representation learning.
- We apply FoN loss to existing contrastive learning-based time-series representation learning methods TS2Vec, CoST, and TS-TCC to record performance improvements in univariate classification, multivariate classification, and multivariate forecasting tasks, demonstrating that FoN loss is model-agnostic as well as task-agnostic.
- We found that FoN loss achieves higher performance to the baseline model with as few as 50% of training epochs in multivariate time-series classification task, thus experimentally demonstrating that superior performance can be expected with fewer training epochs through FoN loss.

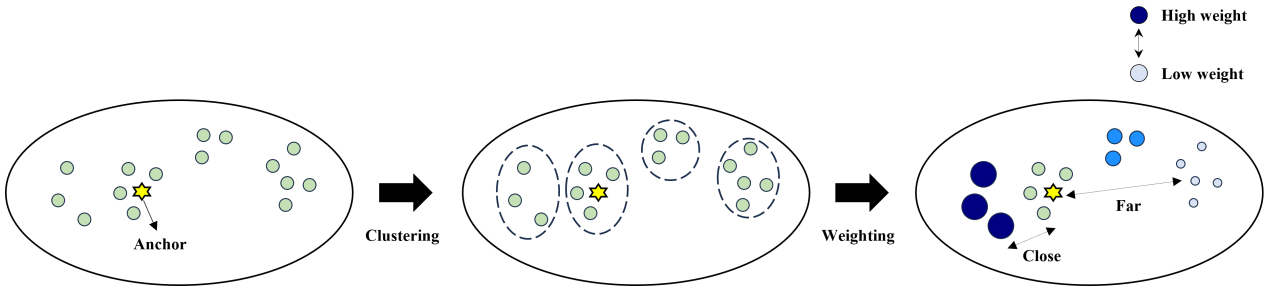

Figure 2: Conceptual diagram of FoN loss. FoN loss is a loss function that clusters the raw time-series through clustering, giving large training weights to samples close to the cluster to which the anchor belongs and small training weights to samples farther away.

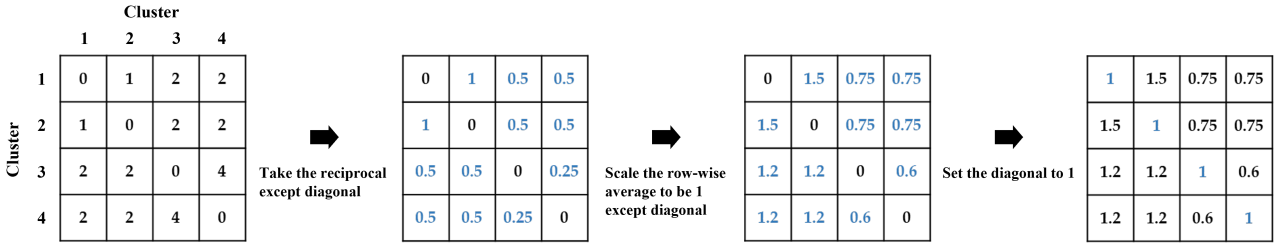

Figure 3: The process of deriving the Anchor-to-Sample weight matrix from the cluster-to-cluster distance matrix. The Anchor-to-Sample weight matrix is calculated by taking the inverse of the components except for the distance to self (diagonal of the matrix), scaling them row-wise so that the average is 1, and changing the weight to self (diagonal of the matrix) to 1.

Proposed Method

Figure 1 depicts the training process of a time-series representation learning model with FoN loss. K-means clustering is performed by instance-wise or timestamp-wise, and the anchor-to-sample weight matrix is calculated. After that, the anchor-to-sample weight matrix is used as a lookup table to train a time-series representation learning model through FoN loss. The specific instructions on performing K-means clustering, calculating FoN loss, and applying FoN loss are provided in detail in Section *Algorithm of FoN Loss*, and Section *Application to Task*.

Problem Definition

Given a time-series $\left(\mathbf{X} \in \mathbb{R}^{N \times T \times F}\right)$ consisting of N instances, T timestamps, and F features. Our goal is to derive a function f_θ that maps time-series to a representation using contrastive learning. Specifically, the time-series $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$ and the corresponding representation $\mathbf{R} = \{r_1, r_2, \ldots, r_N\}$ can be denoted by instance, where the time-series for the i-th instance is expressed as $x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,T}\}$, and its corresponding representation is given by $r_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,T}\}$. In conclusion, f_θ maps $x_{i,t} \in \mathbb{R}^F$ to $r_{i,t} \in \mathbb{R}^K$.

Algorithm of FoN Loss

FoN loss leverages the cluster information of the raw time-series generated by K-means clustering to give greater weight to negative samples with high similarity to anchors (Figure 2). It is a universal loss function that can be applied without altering the concepts of existing contrastive learning-based time-series representation learning methods. This is because it employs a technique that assigns weights to the negative samples already sampled, rather than imposing restrictions on the sampling method. Here is the notation for FoN loss along with its specific formula.

Let i_m be the instance of the raw time-series of the m-th sample, t_m the timestamp, r_{i_m,t_m} the representation of the m-th sample, and r'_{i_m,t_m} the positive pair. Also, assume that the negative pairs are other samples in the mini-batch. Letting temperature be τ, batch size be M, and similarity of two vectors x, y be $sim(x, y)$, the contrastive loss of the s-th batch using the m-th sample as an anchor can be expressed as Eq. (1):

$$\ell^s = -\sum_{m=1}^{M} \log \frac{\exp\left(\text{sim}\left(r_{i_m,t_m} \cdot r'_{i_m,t_m}\right)/\tau\right)}{\sum_{b=1}^{M} \exp\left(\text{sim}\left(r_{i_m,t_m} \cdot r_{i_b,t_b}\right)/\tau\right)}. \quad (1)$$

FoN loss reflects the training importance of each negative sample by multiplying the contrastive loss by the training weight α calculated based on K-means clustering. When the training weights of r_{i_m,t_m} (anchor) to r_{i_b,t_b} (sample) are defined as $\alpha_{m,b}$, changing Eq. (1) to

Task	128 UCR dataset				29 UEA dataset			
Baseline Model	TS2Vec		TS-TCC		TS2Vec		TS-TCC	
FoN loss	-	✓	-	✓	-	✓	-	✓
Avg. ACC	0.820	**0.832**	0.678	**0.695**	0.698	**0.716**	0.605	**0.620**
ACC gap	**1.2%p**		**1.7%p**		**1.8%p**		**1.5%p**	
Ours 1 to 1 wins	86		94		21		16	
Ours 1 to 1 draws	17		18		3		5	
Ours 1 to 1 losses	25		16		5		8	
Wilcoxon Test p-value	2.30E-06		6.79E-15		0.000854		0.0120	

Table 1: Univariate/multivariate time-series classification results.

FoN loss is equivalent to Eq. (2):

$$FoN^s = -\sum_{m=1}^{M} \log \frac{\exp\left(\text{sim}\left(r_{i_m,t_m} \cdot r'_{i_m,t_m}\right)/\tau\right)}{\sum_{b=1}^{M} \alpha_{m,b} \exp\left(\text{sim}\left(r_{i_m,t_m} \cdot r_{i_b,t_b}\right)/\tau\right)} \,. \tag{2}$$

As shown in Eq. (2), the key to FoN loss is the training weight $\alpha_{m,b}$. The training weight $\alpha_{m,b}$ is calculated through K-means clustering, and the specific calculation process is as follows:

First, set the domains to perform K-means clustering on the raw time-series $\mathbf{X}$. Since $\mathbf{X}$ consists of three dimensions $(instance, timestamp, feature)$, we can perform K-means clustering on three domains: instance, timestamp, and feature. We set the domain that can best reflect the characteristics of each task to perform K-means clustering. The specific domain settings and reasons for them, as well as the K-means clustering method, are described in detail in the section of the *Application to Task*. The centroid of each cluster obtained through K-means clustering is used to calculate the cluster-to-cluster distance matrix. Each component of the cluster-to-cluster distance matrix uses the Euclidean distance between the centroids of the clusters. For example, if the number of clusters is set to 4, the cluster-to-cluster distance matrix is a matrix with a size of 4×4 as shown in the leftmost of Figure 3. Then, all values except the diagonal are reciprocalized and scaled so that the row-wise average of all values except the diagonal is 1. Finally, by changing the diagonal to 1, the Anchor-to-Sample weight matrix is calculated (Figure 3). The training weights calculated in this way can be used to reflect more appropriate hard negative samples in learning. According to Robinson et al. (2021), the conditions for appropriate hard negative samples are ① negative samples that are similar to an anchor and difficult to distinguish, and ② true negative samples with a different label than the anchor. The FoN loss is a loss function designed to satisfy these conditions. To satisfy condition ①, we used the reciprocal of the distance between the centroids of the clusters. To satisfy condition ②, we used a training weight of 1, which is the average of all training weights, when the anchor and the negative sample belong to the same cluster. The reason for setting condition ② is that negative samples that belong to the same cluster as an anchor have a higher probability of being useful samples for learning because they are similar negative samples to the anchor, but they also have a higher probability of not being true negative samples. Therefore, there is a trade-off between hard negative and false negative. To reflect this, we set the training weight to 1 (average of all training weights) when the anchor and sample are in the same cluster. The Anchor-to-Sample weight matrix is used as a lookup table, and the FoN loss is calculated by multiplying the training weights between the anchor and sample by the similarity term of contrastive loss (Eq. (2)).

Application to Task

We applied FoN loss to classification and forecasting tasks to compare their performance. The specific application of FoN loss in time-series classification and forecasting tasks is as follows:

Time-series Classification Classification tasks are to classify each instance, which is organized in a 2-dimensional form of $(timestamp, feature)$. Therefore, it is important to know which cluster each instance belongs to. Thus, we performed K-means clustering on the instance domain. Also, for the multivariate classification task, we flattened the 2-dimensional form of $(timestamp, feature)$ to the 1-dimensional form of $(timestamp \times feature)$. To summarize, we flatten the raw time-series data $\mathbf{X} \in \mathbb{R}^{N \times T \times F}$ to generate $\mathbf{X}_{\text{flatten}} \in \mathbb{R}^{N \times TF}$ and perform K-means clustering on $\mathbf{X}_{\text{flatten}}$ based on instance domain to obtain an N-dimensional vector $L_{\text{instance}} \in \mathbb{R}^N$ containing cluster labels for each instance.

Time-series Forecasting Forecasting tasks are to predict the value at a future period through a time-series consisting of $(timestamp, feature)$ in a 2-dimensional form. Therefore, the most important part is to identify the characteristics of the data as it changes over time. To capture the information about the change of time, K-means clustering was performed on the timestamp domain in the case of forecasting. If the number of instances is more than one, K-means clustering can be performed by flattening the 2-dimensional form of $(instance, feature)$ to $(instance \times feature)$, similar to the case of multivariate classification. For the reasons mentioned, we perform K-means clustering on the

Dataset	ADF Test Stat.	H	TS2Vec				CoST				Number of clusters	
			MSE		MAE		MSE		MAE			
			B/L	FoN	B/L	FoN	B/L	FoN	B/L	FoN	TS2Vec	CoST
ETTh1	-3.488	24	0.589	**0.581**	0.539	**0.531**	0.391	**0.382**	0.428	**0.427**	8	8
		48	0.635	**0.624**	0.570	**0.561**	0.444	**0.437**	0.464	**0.464**		
		168	0.775	**0.761**	0.652	**0.642**	0.634	**0.631**	0.575	**0.576**		
		336	0.937	**0.925**	0.736	**0.728**	0.815	**0.801**	0.676	**0.671**		
		720	1.076	**1.018**	0.809	**0.789**	**0.903**	0.921	**0.750**	0.750		
		Avg	0.802	**0.782**	0.661	**0.650**	0.637	**0.634**	0.578	**0.578**		
ETTh2	-3.597	24	0.431	**0.381**	0.482	**0.460**	0.428	**0.420**	0.490	**0.487**	10	8
		48	0.636	**0.572**	0.606	**0.577**	0.669	**0.651**	0.627	**0.620**		
		168	1.905	**1.887**	1.069	**1.061**	1.720	**1.691**	1.018	**1.010**		
		336	**2.422**	2.456	**1.262**	1.268	1.898	**1.870**	1.079	**1.073**		
		720	2.495	**2.378**	1.349	**1.295**	2.029	**2.019**	1.100	**1.100**		
		Avg	1.578	**1.535**	0.954	**0.932**	1.349	**1.330**	0.863	**0.858**		
ETTm1	-4.191	24	0.412	**0.388**	0.425	**0.412**	0.254	**0.252**	0.333	**0.332**	8	6
		48	0.556	**0.518**	0.504	**0.489**	0.334	**0.334**	0.388	**0.387**		
		96	0.593	**0.569**	0.539	**0.526**	0.383	**0.379**	0.422	**0.419**		
		288	0.676	**0.638**	0.593	**0.575**	0.474	**0.473**	0.487	**0.486**		
		672	0.751	**0.725**	0.642	**0.630**	0.626	**0.624**	0.577	**0.576**		
		Avg	0.598	**0.568**	0.541	**0.526**	0.414	**0.413**	0.441	**0.440**		
Weather	-6.382	24	0.311	**0.308**	0.361	**0.359**	0.298	**0.298**	0.361	**0.360**	6	6
		48	0.380	**0.374**	0.415	**0.413**	0.359	**0.359**	0.410	**0.410**		
		168	0.503	**0.492**	0.506	**0.503**	0.466	**0.466**	0.493	**0.493**		
		336	0.553	**0.535**	0.537	**0.533**	0.501	**0.501**	0.520	**0.519**		
		720	0.601	**0.576**	0.563	**0.558**	0.536	**0.536**	0.544	**0.544**		
		Avg	0.469	**0.457**	0.476	**0.473**	0.432	**0.432**	0.465	**0.465**		
Exchange-rate	-1.728	24	0.131	**0.082**	0.283	**0.212**	0.161	**0.150**	0.305	**0.293**	8	6
		48	0.218	**0.157**	0.359	**0.291**	0.263	**0.255**	0.394	**0.384**		
		168	0.779	**0.686**	0.648	**0.599**	0.849	**0.823**	0.735	**0.714**		
		336	1.858	**1.561**	1.026	**0.944**	1.367	**1.334**	0.931	**0.911**		
		720	2.521	**2.005**	1.218	**1.105**	1.863	**1.841**	1.059	**1.049**		
		Avg	1.102	**0.898**	0.707	**0.630**	0.901	**0.880**	0.685	**0.670**		
Average performance increase			**6.80%**		**3.80%**		**1.18%**		**0.70%**		-	-

Table 2: Multivariate time-series forecasting results. B/L in the columns stands for baseline. For TS2Vec, we achieved an average performance improvement of 6.80% by MSE and 3.80% by MAE, and for CoST, we achieved a performance improvement of 1.18% and 0.70%. When calculating the average of all models, the performance improvement is 4.27% by MSE and 2.32% by MAE.

timestamp domain, which results in a T-dimensional vector $L_{\text{timestamp}} \in \mathbb{R}^T$ containing the cluster label for each timestamp.

Experiments

This section describes the results of the classification and forecasting tasks by applying FoN loss to TS2Vec (Yue et al. 2022), TS-TCC (Eldele et al. 2021), and CoST (Woo et al. 2022), which are contrastive learning-based time-series representation learning models. The final performance was calculated using logistic regression for classification and ridge regression for forecasting using the produced representations. The specific experimental setup, experimental results, and analysis for each model and task are as follows:

Time-series Classification

We used TS2Vec and TS-TCC as baseline models for time-series classification. We compared the performance of each model with and without FoN loss. For fair comparisons, we used the same hyperparameters shared by baseline and FoN loss for each model. The specific hyperparameter settings are shown in the *Appendix*. In the case of FoN loss, the number of clusters with the highest validation accuracy among $\{3, 4, 5, 6, 7, 8, 9\}$ was used as the final number of clusters. We used 128 datasets in UCR archive (Dau et al. 2019) for the univariate classification, and 29 datasets in UEA archive (Bagnall et al. 2018) for the multivariate classification. Datasets are already divided into training sets and test sets. Therefore, we divided the training set by 8:2 for validation.

The average accuracy of classifications and 1 to 1 comparison with and without FoN loss for each base-

line model are shown in Table 1. Avg. ACC in Table 1 means the average accuracy of all datasets in each task, and Ours 1 to 1 wins/draws/losses mean the number of datasets with higher/tied/lower accuracy when FoN loss is used in order. The results of this experiment show that regardless of the baseline model and task, the performance improves when FoN loss is used. In the case of TS2Vec, the average accuracy improvement with FoN loss is 1.5%p, and in TS-TCC, the average accuracy improvement is 1.6%p. This confirms that FoN loss works model-agnostically in classification tasks. Furthermore, comparing the average accuracy by task, the average improvement is 1.5%p for univariate time-series classification and 1.7%p for multivariate time-series classification. In both cases, we recorded performance gains, but multivariate is slightly higher. When comparing each dataset on a 1 to 1 basis, the difference between FoN loss and without FoN loss is noticeable. Approximately 70% of the datasets recorded performance improvement with FoN loss, while 17% of the datasets recorded performance degradation with FoN loss. In addition, the p-value of the Wilcoxon signed test is low enough for all model and dataset combinations, proving that FoN loss contributes to performance improvement in a statistically significant way.

Time-series Forecasting

Time-series forecasting tasks are categorized into univariate forecasting, which predicts future values of a specific target feature based on its past values, and multivariate forecasting, which predicts values for all features. In this paper, only multivariate forecasting was performed. This is because in the case of univariate forecasting, the data exists as a scalar in the time domain, so performing clustering simply sorts the values in order of magnitude, and the meaning of clustering is lost.

We used TS2Vec and CoST as baseline models for multivariate time-series forecasting. We compared the performance of each model with and without FoN loss. For fair comparisons, we used the same hyperparameters shared by baseline and FoN loss for each model. The specific hyperparameter settings are shown in the *Appendix*. In the case of FoN loss, the number of clusters was determined based on the validation. Among the options $\{6, 8, 10, 12\}$, the number of clusters that resulted in the lowest sum of MSE and MAE for the prediction slice was chosen. The training set, validation set, and test set were divided into 6:2:2 for the entire dataset. We used ETT (Electricity Transformer Temperature) datasets ETTh1, ETTh2, and ETTm1 (Zhou et al. 2021), which are organized by county and unit of measure; Exchange-rate (Lai et al. 2018), a dataset containing exchange rate information for eight countries on a daily basis from 1990 to 2016; and Weather[1], a dataset with hourly weather information for approximately 1,600 locations in the United States.

The experimental results of multivariate time-series forecasting are shown in Table 2. The performance improvement of 4.27% in MSE and 2.32% in MAE when applying FoN loss is averaged across all datasets and models. When comparing the performance improvements based on the dataset, we can see that Exchange-rate, ETTm1, ETTh2, ETTh1, and Weather, are in that order. This is very similar to the results of the ADF test, which suggests that the lower the stationarity, the higher the performance improvement when applying FoN loss. This shows that the learning method of FoN loss, which gives more weight to negative samples that are similar to the anchor, leads to learning a robust representation even for datasets with changing distributions.

Analysis

How does the Anchor-to-Sample weight matrix affect performance?

Weight matrix	Exchange-rate	UCR	UEA
	Avg. MSE	Avg. ACC	
Ours	**0.898**	**0.833**	**0.716**
w/ Uniform	1.102 (-22.64%)	0.820 (-1.3%p)	0.698 (-1.8%p)
w/ Random	1.039 (-15.73%)	0.794 (-3.9%p)	0.701 (-1.5%p)
w/ Reverse	0.986 (-9.81%)	0.805 (-2.8%p)	0.710 (-0.6%p)

Table 3: Analysis of Anchor-to-Sample weight matrix.

To analyze the effectiveness of the FoN loss, which assigns higher training weights to negative samples that are closer to anchors, we compare the performance of the Anchor-to-Sample weight matrix using *Uniform*, *Random*, and *Reverse*. The performance changes in the multivariate forecasting (Exchange-rate), univariate classification, and multivariate classification tasks when changing the Anchor-to-Sample weight matrix are shown in Table 3. The *Uniform* is the case where the values of the Anchor-to-Sample weight matrix are all set to 1, which is the same as the case of training with the baseline model. The *Random* means that the Anchor-to-Sample weight matrix is changed to a random matrix using the NumPy random function and scaled row-wise. We report the average of the performance recorded when the NumPy seed is changed to [12, 22, 32, 42, 52]. Finally, the *Reverse* means that instead of taking the reciprocal of the distance between clusters, the raw distance is scaled to calculate the Anchor-to-Sample weight matrix. This allows negative samples with lower similarity to the anchor to be more heavily weighted in the training. From this analysis, we can see that the Anchor-to-Sample weight matrix calculated by the method proposed by FoN loss has the best performance regardless of the task. In addition, the performance of *Reverse* is always higher than *Random*, but it is lower than the baseline in univariate classification. This analysis shows that FoN loss, a structure that can reflect more negative samples similar to anchors in learning, is the most effective in improving performance.

[1] www.ncei.noaa.gov/data/local-climatological-data/

Can FoN loss reflect the benefits of hard negative samples?

Task	Method	10%	25%	50%	75%	100%
Multivariate forecasting	B/L	**0.872**	0.897	0.890	0.887	0.910
	FoN	0.880	**0.874**	**0.877**	**0.861**	**0.848**
Univariate classification	B/L	**0.792**	**0.804**	**0.814**	0.817	0.820
	FoN	0.781	0.798	0.813	**0.821**	**0.833**
Multivariate classification	B/L	0.678	**0.689**	0.691	0.692	0.698
	FoN	**0.682**	0.689	**0.706**	**0.714**	**0.716**

Table 4: Performance changes over representation training epochs. For multivariate forecasting, we report the average MSE of all datasets used in this paper, and for univariate/multivariate classification, we report the average accuracy.

In contrastive representation learning methods in CV, NLP, and graph domains, it is mentioned that higher performance and training convergence with fewer training epochs can be expected with hard negative samples (Robinson et al. 2021; Kalantidis et al. 2020; Xiong et al. 2021).

Table 4 shows the results of the performance change for each task as a function of the representation training epoch. For both forecasting and classification, we conducted experiments using TS2Vec as the baseline model. For forecasting, we measured the performance by epoch for all the datasets used in this paper (ETT datasets, Exchange-Rate, and Weather) and recorded the average MSE of all the datasets. For classification, we ran experiments on 128 datasets included in the UCR archive and 29 datasets from the UEA archive and reported the average accuracy. In the case of forecasting, we can see that the MSE is gradually decreasing when FoN loss is applied, while the baseline model shows an unstable learning behavior in which the MSE increases and decreases despite increasing training epochs. In the case of classification, both the baseline and FoN loss cases show a gradual increase in average accuracy. The training epochs at which the FoN loss outperforms 100% of the baseline are 10% for multivariate prediction, 75% for univariate classification, and 50% for multivariate classification. This suggests that FoN loss can outperform the baseline model with fewer training epochs.

Figure 4 illustrates the results of a two-dimensional visualization using t-SNE of the ERing dataset from the UEA archive, showcasing the representation's progression across training epochs. The visualization reveals that even after 75% of the training epochs with the baseline model, the orange classes remain scattered. In contrast, when using the FoN loss, the orange classes are already clustered after only 10% of the training epochs. Taken together, both performance over training epochs and visualization results show that FoN loss contributes to performance and training convergence. This suggests that the FoN loss can properly reflect the benefits of hard negative samples.

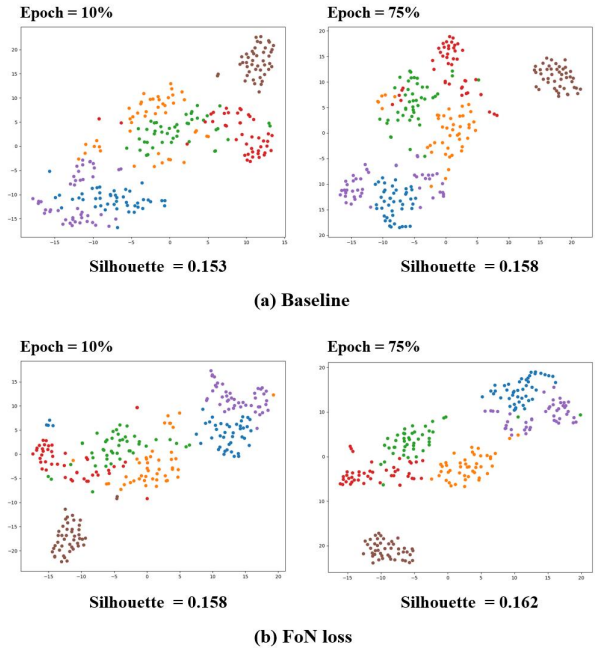

Figure 4: Visualization of changes in training epoch for the ERing dataset from the UEA archive using t-SNE.

Conclusion

In this paper, we propose FoN loss, a loss function that uses K-means clustering to weight learning on hard negative samples. FoN loss is a universal loss function that can be applied to all existing contrastive learning-based time-series representation learning methods. Since it only takes the cluster information of the raw data and reflects it in the contrastive loss without additional hard negative sampling. Not only is it universal, but we also demonstrated that it is a loss function that can be expected to improve performance task-agnostic as well as model-agnostic by recording performance improvement in all task and model combinations. Specifically, we recorded a performance improvement of 1.5%p in average accuracy for univariate classification, 1.7%p in average accuracy for multivariate classification, and 4.27% in average MSE for multivariate forecasting. In addition, it outperformed the baseline model in the multivariate classification task by training only 50% of the epochs and demonstrated convergence in fewer epochs through the performance trend as the number of epochs changed. The FoN loss can be applied to contrastive learning-based time-series representation learning methods to improve performance. Therefore, we expect that it will further improve the performance of contrastive learning-based time-series representation learning that has been published and will be published in the future.

References

Abualigah, L. M. Q.; et al. 2019. Feature selection and enhanced krill herd algorithm for text document clustering.

Aghabozorgi, S.; Shirkhorshidi, A. S.; and Wah, T. Y. 2015. Time-series clustering–a decade review. *Information systems*, 53: 16–38.

Bagnall, A.; Dau, H. A.; Lines, J.; Flynn, M.; Large, J.; Bostrom, A.; Southam, P.; and Keogh, E. 2018. The UEA multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*.

Chang, Y.-C.; Subramanian, D.; Pavuluri, R.; and Dinger, T. 2022. Time series representation learning with contrastive triplet selection. In *5th Joint International Conference on Data Science & Management of Data* (*9th ACM IKDD CODS and 27th COMAD*), 46–53.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, 1597–1607. PMLR.

Chen, Y.; Hu, B.; Keogh, E.; and Batista, G. E. 2013. Dtw-d: time series semi-supervised learning from a single example. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 383–391.

Dau, H. A.; Bagnall, A.; Kamgar, K.; Yeh, C.-C. M.; Zhu, Y.; Gharghabi, S.; Ratanamahatana, C. A.; and Keogh, E. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6): 1293–1305.

Dempster, A.; Petitjean, F.; and Webb, G. I. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5): 1454–1495.

Eldele, E.; Ragab, M.; Chen, Z.; Wu, M.; Kwoh, C. K.; Li, X.; and Guan, C. 2021. Time-series representation learning via temporal and contextual contrasting. *arXiv preprint arXiv:2106.14112*.

Ezugwu, A. E.; Ikotun, A. M.; Oyelade, O. O.; Abualigah, L.; Agushaka, J. O.; Eke, C. I.; and Akinyelu, A. A. 2022. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110: 104743.

Fang, H.; Wang, S.; Zhou, M.; Ding, J.; and Xie, P. 2020. Cert: Contrastive self-supervised learning for language understanding. *arXiv preprint arXiv:2005.12766*.

Franceschi, J.-Y.; Dieuleveut, A.; and Jaggi, M. 2019. Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems*, 32.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9729–9738.

Jaiswal, A.; Babu, A. R.; Zadeh, M. Z.; Banerjee, D.; and Makedon, F. 2020. A survey on contrastive self-supervised learning. *Technologies*, 9(1): 2.

Kalantidis, Y.; Sariyildiz, M. B.; Pion, N.; Weinzaepfel, P.; and Larlus, D. 2020. Hard negative mixing for contrastive learning. *Advances in Neural Information Processing Systems*, 33: 21798–21809.

Lai, G.; Chang, W.-C.; Yang, Y.; and Liu, H. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, 95–104.

Le-Khac, P. H.; Healy, G.; and Smeaton, A. F. 2020. Contrastive representation learning: A framework and review. *Ieee Access*, 8: 193907–193934.

Liu, Y.; Yang, X.; Zhou, S.; Liu, X.; Wang, Z.; Liang, K.; Tu, W.; Li, L.; Duan, J.; and Chen, C. 2023. Hard sample aware network for contrastive deep graph clustering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, 8914–8922.

Robinson, J. D.; Chuang, C.-Y.; Sra, S.; and Jegelka, S. 2021. Contrastive Learning with Hard Negative Samples. In *International Conference on Learning Representations*.

Seh, A. H.; Zarour, M.; Alenezi, M.; Sarkar, A. K.; Agrawal, A.; Kumar, R.; and Ahmad Khan, R. 2020. Healthcare data breaches: insights and implications. In *Healthcare*, volume 8, 133. MDPI.

Sezer, O. B.; Gudelek, M. U.; and Ozbayoglu, A. M. 2020. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied soft computing*, 90: 106181.

Shen, L.; Yu, Z.; Ma, Q.; and Kwok, J. T. 2021. Time series anomaly detection with multiresolution ensemble decoding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 9567–9575.

Tonekaboni, S.; Eytan, D.; and Goldenberg, A. 2021. Unsupervised Representation Learning for Time Series with Temporal Neighborhood Coding. In *International Conference on Learning Representations*.

Wang, J.; Xu, C.; Zhang, J.; and Zhong, R. 2022. Big data analytics for intelligent manufacturing systems: A review. *Journal of Manufacturing Systems*, 62: 738–752.

Woo, G.; Liu, C.; Sahoo, D.; Kumar, A.; and Hoi, S. 2022. CoST: Contrastive Learning of Disentangled Seasonal-Trend Representations for Time Series Forecasting. In *International Conference on Learning Representations*.

Xia, W.; Wang, Q.; Gao, Q.; Yang, M.; and Gao, X. 2022. Self-consistent contrastive attributed graph clustering with pseudo-label prompt. *IEEE Transactions on Multimedia*.

Xiong, L.; Xiong, C.; Li, Y.; Tang, K.-F.; Liu, J.; Bennett, P. N.; Ahmed, J.; and Overwijk, A. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *International Conference on Learning Representations*.

Xu, J.; Wu, H.; Wang, J.; and Long, M. 2022. Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy. In *International Conference on Learning Representations*.

Yue, Z.; Wang, Y.; Duan, J.; Yang, T.; Huang, C.; Tong, Y.; and Xu, B. 2022. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 8980–8987.

Zeng, A.; Chen, M.; Zhang, L.; and Xu, Q. 2023. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, 11121–11128.

Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; and Zhang, W. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 11106–11115.

Zhou, T.; Ma, Z.; Wen, Q.; Wang, X.; Sun, L.; and Jin, R. 2022. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, 27268–27286. PMLR.

Zolhavarieh, S.; Aghabozorgi, S.; Teh, Y. W.; et al. 2014. A review of subsequence time series clustering. *The Scientific World Journal*, 2014.

Related Work

Time-Series Clustering

Clustering is a method of grouping similar data through criteria such as data features and characteristics (Abualigah et al. 2019). Clustering methods have been widely employed across diverse domains, such as text, image, and graph. Moreover, time-series data has been subjected to various clustering approaches for analysis and interpretation (Ezugwu et al. 2022). In particular, time-series data clustering can be categorized into three types depending on the dimension of the data to be clustered: whole time-series clustering, subsequence time-series clustering, and time point clustering (Zolhavarieh et al. 2014). In the case of the whole time-series clustering, this means performing clustering on each whole time-series consisting of instances, timestamps, and features, while for subsequence time-series clustering, this means clustering a subset of the time-series in windows rather than the entire time-series. Finally, time point clustering is a slightly different concept than the first two categories. This involves clustering timestamps based on their temporal proximity or value in the data (Aghabozorgi, Shirkhorshidi, and Wah 2015). The clustering methodology used in this study is subsequence time-series clustering, specifically when the window size is 1. Subsequence time-series clustering can be categorized into Hierarchical clustering, Partitioning clustering, Density-based clustering, and Pattern discovery according to the clustering method (Zolhavarieh et al. 2014). In this paper, Euclidean distance (ED)- / dynamic time wrapping (DTW)- based K-means clustering was considered based on the time complexity of the algorithm and how well it can capture the characteristics of time-series data. However, DTW-based methodologies were excluded due to high time complexity. In conclusion, we adopted ED-based K-means clustering as the clustering algorithm considering its time complexity and suitability with time-series data.

Contrastive Time-Series Representation Learning

Contrastive representation learning has become an important method in various domains, including CV, NLP, and graph, as well as in the time-series domain (Chen et al. 2020; Fang et al. 2020; Xia et al. 2022). Contrastive learning refers to a methodology that trains the embedding space to be closer to the anchor and positive samples and further away from negative samples (Le-Khac, Healy, and Smeaton 2020). Here, a positive sample is a sample with the same label as the anchor, and a negative sample is a sample with a different label than the anchor. Since most contrastive representation learning is performed in an unsupervised setting, constructing appropriate positive/negative samples is a key part of the process (Jaiswal et al. 2020). In the field of the time-series domain, various methodologies have been proposed to construct positive samples. In T-loss (Franceschi, Dieuleveut, and Jaggi 2019), a subseries of the anchors is randomly selected and considered as a positive sample. In the case of TNC (Tonekaboni, Eytan, and Goldenberg 2021), positive samples are constructed by determining the presence or absence of neighbors with anchors through the ADF test. In addition, TS2Vec (Yue et al. 2022), TS-TCC (Eldele et al. 2021), and CoST (Woo et al. 2022) construct positive samples by augmenting data through the application of data augmentation techniques to anchor. Specifically, TS2Vec utilizes time cropping and masking to perform data augmentation, and TS-TCC applies two data augmentation methodologies: weak augmentation through jitter-and-scale strategy and strong augmentation through permutation-and-jitter strategy. Finally, CoST uses scale, shift, and jitter to augment the data. Methodologies that utilize such data augmentation propose a methodology for capturing the characteristics of a time-series. Specifically, TS2Vec employs hierarchical contrastive learning in both the instance and temporal domains, and TS-TCC performs cross-view contrasting on weak augmentation and strong augmentation. In the case of the CoST, it utilizes seasonal-trend decomposition. On the other hand, the construction of negative samples is relatively simple compared to positive samples. Methodologies that consider all other data in a batch as negative samples and proceed with training without analyzing the data selected as negative samples (Franceschi, Dieuleveut, and Jaggi 2019; Eldele et al. 2021), and use samples with different timestamps or in-

Dataset	Number of clusters								Avg	Std	Over rate
	3	4	5	6	7	8	9	B/L			
Crop	0.733	0.740	0.740	0.736	0.735	0.737	0.732	**0.760**	0.736	0.003	0.00%
ElectricDevices	0.696	0.699	**0.703**	0.699	0.700	0.700	0.702	0.697	0.700	0.002	85.71%
FordB	0.786	0.773	0.788	0.783	**0.799**	0.790	0.774	0.785	0.785	0.009	57.14%
FaceDetection	0.514	0.514	0.516	0.526	**0.528**	0.516	0.522	0.513	0.519	0.006	100.00%
PhonemeSpectra	0.231	0.235	**0.239**	0.234	0.238	0.229	0.233	0.227	0.234	0.004	100.00%
SpokenArabicDigits	0.986	0.989	0.988	0.988	**0.989**	0.985	0.984	0.988	0.987	0.002	28.57%
Avg	0.658	0.658	0.662	0.661	**0.665**	0.660	0.658	0.662	0.660	0.004	61.90%

Table 5: Accuracy changes over the number of clusters on a classification task. The Over rate column is the percentage of clusters that outperformed the B/L (baseline).

Dataset	H	Number of clusters					Avg	Std	Over rate
		6	8	10	12	B/L			
Exchange-rate	24	**0.075**	0.082	0.095	0.075	0.131	0.081	0.008	100%
	48	0.158	**0.157**	0.179	0.183	0.218	0.169	0.012	100%
	168	**0.659**	0.686	0.754	0.707	0.779	0.701	0.035	100%
	336	1.686	**1.561**	1.660	1.700	1.858	1.652	0.055	100%
	720	2.336	**2.005**	2.027	2.227	2.521	2.149	0.138	100%
	Avg	0.982	**0.898**	0.943	0.979	1.102	0.951	0.034	100%

Table 6: MSE changes over the number of clusters on a forecasting task. The Over rate column is the percentage of clusters that outperformed the B/L (baseline).

stances as negative samples are being used (Yue et al. 2022). In addition, CoST (Woo et al. 2022) uses the memory bank proposed by MoCO (He et al. 2020), but it is a methodology that simply increases the number of negative samples, and there is no separate validation of negative samples. In conclusion, we can see that even the latest time-series representation learning methodologies are mainly based on the randomness of negative samples. When negative samples are heavily influenced by randomness in this manner, a challenge arises in effectively incorporating hard negative samples into the training process. Here, the hard negative sample refers to a negative sample that is challenging to differentiate from an anchor. Numerous studies have empirically demonstrated that utilizing hard negative samples can lead to improved performance and training convergence (Robinson et al. 2021; Kalantidis et al. 2020; Xiong et al. 2021; Liu et al. 2023). Since these studies are not specific to one domain but come from various domains such as CV, NLP, and graph, we can expect that hard negative samples also play an important role in the time-series domain.

Experiments Detail

In this paper, we used TS2Vec (Yue et al. 2022), CoST (Woo et al. 2022), and TS-TCC (Eldele et al. 2021) as baseline models. We followed the hyperparameter settings suggested in the paper for the baseline model as much as possible. We also set the same hyperparameters for both the baseline model and the case with FoN loss. Experiments were conducted on an Nvidia GeForce RTX 2080 Ti GPU and an Intel(R) Core(TM) i7-11700K CPU @ 3.60GHz environment. The specific hyperparameter settings for each model are as follows:

TS2Vec

For TS2Vec, we used most of the same settings as in the paper (Yue et al. 2022), but due to the computational cost, we changed only the batch size and max train length to perform the experiment. The specific hyperparameter settings are batch size = 4, learning rate = 0.001, representation dimension = 320, max train length = 201, optimizer = AdamW. For the number of iterations, we used 200 iterations if the size of the entire dataset (product of the number of instances, the number of timestamps, and the number of features) is less than 100,000, and 600 iterations if it is more than 100,000.

CoST

In the case of CoST, we used most of the same settings as in the original paper (Woo et al. 2022), but due to the computational cost, only the batch size was changed to perform the experiment. The specific hyperparameter settings are batch size = 128, learning rate = 0.001, representation dimension = 320, max train

length = 201, and optimizer = AdamW. For the number of iterations, we used 200 iterations if the size of the entire dataset (product of the number of instances, the number of timestamps, and the number of features) is less than 100,000, and 600 iterations if it is more than 100,000. We also set α, a hyperparameter that refers to the ratio of the losses calculated in the time domain and frequency domain, to 0.0005, following the original paper (Woo et al. 2022) setting.

TS-TCC

For the experiment of TS-TCC, we set the hyperparameters by referring to the hyperparameter setting of HAR dataset. In the case of TS-TCC, the prediction of future timesteps for strong augmentation and weak augmentation is performed during the temporal contrasting process. In this process, it is necessary to set a hyperparameter for how much to set the future timestep. The original paper (Eldele et al. 2021) recommends a level of 40% of the feature length, so we set the same in this experiment. We also set the weight $\lambda_1 = 1$ for the loss calculated by temporal contrasting and $\lambda_2 = 0.7$ for the loss calculated by contextual contrasting, which is the same as the setting suggested in the original paper. The specific hyperparameter settings are batch size = 8 (UCR), 4 (UEA), number of epochs = 40, optimizer = Adam, learning rate = 0.0003, jitter and scale ratio = 1.1 (jitter ratio used in weak augmentation), jitter ratio = 0.8 (jitter ratio used in strong augmentation), and maximum segment length 8.

Performance Variation With the Number of Clusters

Table 5 and Table 6 show the performance of forecasting and classification tasks according to the number of clusters when TS2Vec is used as the baseline model. The over rate in each table means the percentage of clusters that outperformed the baseline for each item. In Table 5, Crop, ElectricDevices, and FordB are univariate classification datasets from the UCR archive, and FaceDetection, PhonemeSpectra, and SpokenArabicDigits are multivariate classification datasets from the UEA archive. For univariate classification, we selected the top three datasets with the highest number of instances. For multivariate classification, we also selected the dataset with the highest number of instances, but to better reflect the multivariate nature of the classification, we selected the top three datasets with the highest number of instances, except for PenDigits, which has two features. For Classification, we outperformed the baseline in an average of 61.90% of the clusters. In case of the forecasting, we outperformed the baseline for all horizon and cluster combinations. The validation allows us to choose the number of clusters that perform better, but overall, it outperforms the baseline regardless of the number of clusters, suggesting that performance improvements can be expected without additional hyperparameter tuning.

Full Results

Table 7 and Table 8 show the performance of all datasets in the UCR archive and UEA archive, respectively, and the optimal number of clusters obtained through validation. About 70% of the datasets showed a performance improvement with FoN loss, and only about 17% showed a performance decrease.

Dataset	TS2Vec			TS-TCC		
	B/L	FoN	# of clusters	B/L	FoN	# of clusters
ACSF1	**0.850**	0.830	8	0.404	**0.471**	4
Adiac	**0.744**	0.678	8	0.185	**0.267**	4
AllGestureWiimoteX	**0.720**	0.717	9	**0.510**	0.503	4
AllGestureWiimoteY	**0.766**	0.759	8	**0.545**	0.543	4
AllGestureWiimoteZ	**0.740**	0.711	3	**0.429**	0.403	4
ArrowHead	0.806	**0.829**	8	0.657	**0.685**	6
Beef	0.700	**0.833**	9	0.458	0.458	3
BeetleFly	0.900	0.900	9	0.750	**0.833**	4
BirdChicken	0.800	**0.950**	6	0.583	**0.708**	3
BME	0.967	**0.993**	9	0.752	**0.779**	9
Car	0.783	**0.917**	7	0.703	**0.750**	8
CBF	1.000	1.000	9	0.867	**0.882**	7
Chinatown	0.971	**0.985**	8	0.772	**0.833**	4
ChlorineConcentration	**0.793**	0.791	6	0.562	**0.570**	9
CinCECGTorso	**0.841**	0.776	7	0.969	**0.971**	9
Coffee	1.000	1.000	9	0.969	0.969	5
Computers	0.660	**0.672**	9	0.543	**0.547**	6
CricketX	0.769	**0.797**	4	0.635	**0.648**	5
CricketY	0.767	**0.772**	4	0.641	**0.646**	7
CricketZ	0.787	**0.813**	9	0.669	**0.674**	3
Crop	**0.760**	0.740	4	**0.640**	0.638	8
DiatomSizeReduction	0.980	**0.987**	7	0.798	**0.846**	8
DistalPhalanxOutlineAgeGroup	0.719	**0.734**	4	0.722	**0.736**	6
DistalPhalanxOutlineCorrect	0.772	**0.783**	7	0.646	**0.654**	9
DistalPhalanxTW	0.698	**0.705**	3	0.715	**0.722**	9
DodgerLoopDay	0.513	**0.550**	4	**0.700**	0.688	9
DodgerLoopGame	0.884	**0.891**	8	0.722	0.722	3
DodgerLoopWeekend	0.971	**0.978**	7	0.965	0.965	3
Earthquakes	0.748	0.748	9	0.736	**0.757**	4
ECG200	0.930	0.930	7	0.817	**0.846**	4
ECG5000	0.936	**0.942**	4	**0.941**	0.940	8
ECGFiveDays	1.000	1.000	9	0.712	**0.745**	8
ElectricDevices	0.697	**0.703**	5	0.646	**0.669**	4
EOGHorizontalSignal	0.511	**0.544**	7	0.429	**0.432**	3
EOGVerticalSignal	0.470	**0.503**	5	0.356	**0.364**	3
EthanolLevel	0.530	**0.564**	3	**0.288**	0.286	3
FaceAll	0.795	**0.798**	9	0.761	**0.763**	3
FaceFour	0.909	**0.920**	5	0.818	**0.841**	4
FacesUCR	**0.927**	0.893	4	0.872	**0.876**	6
FiftyWords	0.813	**0.815**	5	0.699	**0.716**	4
Fish	0.931	**0.949**	7	0.687	0.687	4
FordA	0.936	**0.934**	7	0.915	**0.920**	7
FordB	0.785	**0.799**	7	0.776	**0.784**	6

Dataset	TS2Vec			TS-TCC		
	B/L	FoN	# of clusters	B/L	FoN	# of clusters
FreezerRegularTrain	0.985	**0.992**	3	0.864	**0.891**	7
FreezerSmallTrain	0.885	**0.943**	5	0.669	**0.678**	7
Fungi	0.941	**0.962**	5	0.344	**0.370**	5
GestureMidAirD1	0.654	**0.685**	6	0.632	**0.647**	6
GestureMidAirD2	0.538	**0.631**	3	0.500	**0.537**	7
GestureMidAirD3	0.323	**0.392**	3	0.294	**0.331**	7
GesturePebbleZ1	0.843	**0.907**	6	0.824	**0.830**	9
GesturePebbleZ2	0.867	**0.892**	3	0.794	**0.825**	4
GunPoint	0.980	**0.993**	9	0.822	**0.842**	4
GunPointAgeSpan	0.994	**1.000**	9	0.884	**0.894**	3
GunPointMaleVersusFemale	0.997	**1.000**	3	0.959	**0.972**	3
GunPointOldVersusYoung	1.000	1.000	9	1.000	1.000	3
Ham	0.667	**0.781**	3	0.714	**0.732**	6
HandOutlines	0.914	**0.924**	6	0.886	**0.904**	7
Haptics	**0.539**	0.321	9	0.433	**0.439**	5
Herring	0.609	**0.625**	6	0.594	**0.641**	8
HouseTwenty	**0.950**	0.933	6	0.765	**0.792**	4
InlineSkate	0.395	**0.407**	4	0.290	**0.307**	4
InsectEPGRegularTrain	1.000	1.000	9	1.000	1.000	3
InsectEPGSmallTrain	1.000	1.000	9	1.000	1.000	3
InsectWingbeatSound	0.627	**0.645**	8	0.642	**0.646**	3
ItalyPowerDemand	**0.967**	0.951	3	0.915	**0.919**	8
LargeKitchenAppliances	0.859	**0.875**	3	0.584	**0.608**	7
Lightning2	0.852	**0.885**	4	0.603	**0.691**	7
Lightning7	**0.822**	0.808	9	0.762	**0.788**	7
Mallat	0.948	**0.962**	6	0.694	**0.725**	9
Meat	0.933	0.933	8	0.313	0.313	3
MedicalImages	0.771	**0.801**	8	0.705	**0.711**	9
MelbournePedestrian	**0.961**	0.898	7	0.820	**0.832**	8
MiddlePhalanxOutlineAgeGroup	0.636	**0.656**	3	0.650	0.650	8
MiddlePhalanxOutlineCorrect	**0.808**	0.777	3	0.566	0.566	8
MiddlePhalanxTW	0.571	**0.604**	5	0.575	**0.606**	4
MixedShapesRegularTrain	0.916	**0.925**	4	0.859	**0.862**	5
MixedShapesSmallTrain	0.874	**0.885**	5	**0.829**	0.828	8
MoteStrain	0.859	**0.867**	7	0.748	**0.755**	4
NonInvasiveFetalECGThorax1	0.929	0.929	5	0.780	**0.873**	4
NonInvasiveFetalECGThorax2	**0.938**	0.922	5	0.862	**0.882**	5
OliveOil	**0.900**	0.867	5	0.438	0.438	3
OSULeaf	**0.822**	0.810	5	**0.488**	0.480	3
PhalangesOutlinesCorrect	**0.802**	0.774	5	**0.647**	0.644	6
Phoneme	**0.289**	0.280	5	0.195	**0.197**	6
PickupGestureWiimoteZ	**0.860**	0.820	7	0.679	0.679	3
PigAirwayPressure	0.351	**0.476**	5	0.091	**0.101**	5
PigArtPressure	0.923	**0.947**	3	**0.192**	0.188	4
PigCVP	0.721	**0.745**	9	0.096	**0.101**	4
PLAID	0.523	**0.534**	7	**0.397**	0.379	5

Dataset	TS2Vec			TS-TCC		
	B/L	FoN	# of clusters	B/L	FoN	# of clusters
Plane	0.971	**1.000**	8	0.964	**0.973**	3
PowerCons	0.961	**0.989**	8	1.000	1.000	3
ProximalPhalanxOutlineAgeGroup	0.849	**0.859**	9	**0.737**	0.723	7
ProximalPhalanxOutlineCorrect	0.876	0.876	8	0.775	**0.798**	7
ProximalPhalanxTW	0.785	**0.810**	9	0.688	0.688	4
RefrigerationDevices	0.611	**0.613**	9	0.463	**0.500**	7
Rock	**0.740**	0.720	9	0.339	**0.375**	4
ScreenType	0.403	**0.405**	9	0.409	**0.417**	6
SemgHandGenderCh2	0.950	**0.960**	8	0.935	**0.953**	3
SemgHandMovementCh2	0.880	**0.887**	6	0.647	**0.706**	4
SemgHandSubjectCh2	0.942	**0.944**	6	0.882	**0.919**	3
ShakeGestureWiimoteZ	**0.940**	0.920	6	0.625	**0.643**	3
ShapeletSim	0.978	**0.994**	4	0.522	**0.538**	8
ShapesAll	0.893	**0.900**	6	0.723	**0.733**	7
SmallKitchenAppliances	0.728	**0.731**	5	0.606	0.606	4
SmoothSubspace	0.967	**0.973**	6	0.868	0.868	7
SonyAIBORobotSurface1	0.884	**0.894**	8	0.490	**0.505**	9
SonyAIBORobotSurface2	0.867	**0.881**	5	0.760	**0.771**	4
StarLightCurves	0.963	**0.972**	6	**0.947**	0.944	3
Strawberry	0.959	**0.965**	6	0.896	**0.904**	6
SwedishLeaf	0.928	**0.939**	7	0.861	**0.884**	8
Symbols	0.962	**0.972**	5	0.844	**0.848**	6
SyntheticControl	0.997	0.997	7	0.974	**0.987**	6
ToeSegmentation1	0.921	0.921	4	0.578	**0.595**	7
ToeSegmentation2	0.900	**0.931**	7	0.654	**0.691**	6
Trace	1.000	1.000	9	0.692	0.692	3
TwoLeadECG	**0.996**	0.993	5	0.592	**0.598**	4
TwoPatterns	1.000	1.000	9	0.998	**0.999**	5
UMD	1.000	1.000	9	0.833	**0.910**	9
UWaveGestureLibraryAll	0.951	**0.956**	5	0.939	**0.945**	8
UWaveGestureLibraryX	0.812	**0.818**	3	0.788	**0.789**	9
UWaveGestureLibraryY	0.719	**0.740**	3	**0.653**	0.647	9
UWaveGestureLibraryZ	0.762	**0.775**	4	0.678	**0.684**	7
Wafer	0.998	0.998	8	0.992	**0.994**	5
Wine	0.815	**1.000**	5	0.482	**0.643**	9
WordSynonyms	0.699	**0.704**	3	0.613	**0.621**	8
Worms	0.727	**0.753**	6	0.575	**0.600**	9
WormsTwoClass	0.701	**0.792**	9	0.625	**0.650**	6
Yoga	0.856	**0.888**	3	**0.757**	0.753	4
Average	0.820	**0.833**	6.234	0.678	**0.695**	5.586

Table 7: Full results of 128 datasets from the UCR archive. B/L in the columns stands for baseline. Applying FoN loss increased the average accuracy by 1.3%p for TS2Vec and 1.7%p for TS-TCC.

Dataset	TS2Vec			TS-TCC		
	B/L	FoN	# of clusters	B/L	FoN	# of clusters
ArticularyWordRecognition	0.980	**0.990**	4	0.921	**0.944**	8
AtrialFibrillation	0.133	**0.333**	7	0.339	0.339	4
BasicMotions	0.975	**1.000**	5	0.975	0.975	5
CharacterTrajectories	0.994	0.994	4	0.912	**0.967**	3
Cricket	**1.000**	0.986	3	0.861	**0.889**	9
DuckDuckGeese	0.520	**0.540**	3	0.375	**0.393**	8
EigenWorms	**0.908**	0.893	3	0.197	**0.394**	6
Epilepsy	0.957	0.957	3	**0.938**	0.931	6
ERing	0.841	**0.870**	4	0.888	**0.904**	5
EthanolConcentration	0.266	**0.323**	4	0.312	**0.327**	9
FaceDetection	0.513	**0.516**	8	**0.555**	0.540	7
FingerMovements	0.510	**0.520**	8	0.490	**0.529**	9
HandMovementDirection	0.297	**0.365**	7	**0.463**	0.412	6
Handwriting	**0.505**	0.499	7	**0.290**	0.276	6
Heartbeat	0.668	**0.683**	7	**0.713**	0.709	4
JapaneseVowels	0.981	0.981	7	0.082	0.082	3
Libras	0.894	**0.900**	8	0.625	**0.630**	3
LSST	0.527	**0.539**	7	**0.367**	0.365	3
MotorImagery	0.500	**0.530**	8	0.510	0.510	9
NATOPS	0.939	**0.950**	8	**0.734**	0.728	3
PEMS-SF	0.694	**0.711**	7	**0.661**	0.639	4
PenDigits	**0.989**	0.868	7	0.925	**0.932**	3
PhonemeSpectra	0.227	**0.239**	5	0.091	**0.105**	3
RacketSports	0.822	**0.842**	9	0.776	**0.783**	8
SelfRegulationSCP1	0.751	**0.771**	9	0.878	**0.899**	3
SelfRegulationSCP2	0.517	**0.544**	8	0.576	**0.592**	8
SpokenArabicDigits	**0.988**	0.985	8	0.940	**0.953**	3
StandWalkJump	0.467	**0.533**	3	0.321	**0.393**	3
UWaveGestureLibrary	0.881	**0.906**	8	**0.831**	0.831	5
Average	0.698	**0.716**	6.172	0.605	**0.620**	5.379

Table 8: Full results of 29 datasets from the UEA archive. B/L in the columns stands for baseline. Applying FoN loss increased the average accuracy by 1.8%p for TS2Vec and 1.5%p for TS-TCC.