Enhancing Self-Supervised Learning Representation Efficiency for Time Series via Similarity-Based Compression

Brooklyn Berry [*] Yifeng Gao [*]

Abstract

Time series data perhaps is one of the most broad data types that exist and is studied by diverse research fields. Recently, Self-Supervised Learning (SSL) training frameworks, the training frameworks to pre-train deep learning model without human annotations, have been proposed. Because human annotation for time series is typically associated with being costly, there is a growing interest in developing effective SSL for time series data. In SSL, the pre-trained model will often produce a time series embedding summarized from the original time series to ensure temporal information is preserved. Although such representation can effectively capture the semantic information, the potential correlation between each time series sample can induce redundancy in the final representation. In this paper, we propose a compression algorithm named SimCompress to compress the redundant encoder embedding series by removing similar consecutive embeddings. The proposed compression algorithm is easy to implement, and can be adapted to any SSL framework that produces an embedding series. The experiment shows that the proposed process can possibly significantly decrease the amount of space required for data storage without reducing the performance accuracy on downstream tasks.

1 Introduction

Time series data is one of the most widespread data types that exist and is studied across many research domains, such as astronomy [18, 12], medicine [13, 1, 15], and meteorology [24, 19]. With sensing technological advancement, time series data has become more abundant. Due to the rich information that time series holds, the exploration of different time series related tasks may lead to advancement in various domains.

One of the bottleneck issues in mining time series data is the lack of labeling. Often times, only domain experts are capable of providing accurate labels. Due to a lack of expert knowledge or high manual labor costs, deep learning model performance is often significantly impacted as it requires annotated data samples. To combat these obstacles, Contrastive Self-Supervised Learning (SSL) training frameworks for time series data mining has become a popular research field. Contrastive SSL frameworks consists of a pretraining step and a fine-tuning step. In the pre-training step, the framework aims to train a model using both the original and augmented versions of data to understand the underlying patterns of time series data with little human guidance. In the fine-tuning step, the model trains a task-specific head (e.g. a linear classification layer) with a small amount of task-related information (e.g. labels) to adapt to the downstream task. By relying on SSL frameworks as opposed to human labeling needed in the downstream task, the costs to train deep learning models can potentially be reduced.

In SSL pre-training frameworks, the pre-training model often outputs an embedding series summarized from the original time series to preserve the temporal information and ensure all characteristics and semantic information are presented in the embedding. While SSL is often promising, the full length of the output embedding series likely isn't utilized. As a result, the potential correlation between each time series segments can induce redundancy in the final representation. In this paper, we find that the resultant encoder embedding series contains redundant embeddings that do not introduce new information. Based on this observation, we propose an algorithm called SimCompress to compress the embedding series by removing similar consecutive embeddings that do not provide new information. Specifically, we utilize similarity between all samples in a time stamp and all samples in its neighboring time stamp to determine whether new semantic information is presented. If embeddings in a time stamp contain no new information, then its neighboring time stamp will be disregarded, resulting in a compacted version of the original output embedding series. The proposed compression algorithm is easy to implement and can be adapted to any SSL framework that produce an embedding series. The experiment further shows that the proposed process can potentially significantly decrease the amount of space required for data storage without seriously damaging the performance accuracy of down-

[*]University of Texas Rio Grande Valley

stream tasks.

In summary, the contribution of this paper can be summarized as following:

- We proposed a novel compression algorithm SimCompress to create a more space efficient embedding series for downstream tasks.
- The algorithm is independent of the training process. After pre-training, we apply our method to the resultant embedding series to remove unnecessary and duplicate embeddings.
- The proposed method can achieve similar downstream task performance accuracy using only a portion of the entire embedding series to do so.

The rest of the paper is organized as follows: Section 2 discusses recent developments in regards to self-supervised learning, followed by the discussion of the problem and our high-level idea to combat it in Section 3. In Section 4, we introduce our proposed method with Section 5 where experimentation is discussed follows. Finally, we conclude the paper in Section 6.

2 Related Work

Recently, there is a growth interest in self-supervised learning (SSL) frameworks [26, 16, 11, 2, 10, 3, 8] for time series data. In time series data focused SSL framework [25, 7, 20, 6], most research work focuses on pre-training an encoder such that it could transfer the original time series into a meta time series that can represent its semantical meaning in different levels of resolution. To achieve this ability, recently proposed SSL frameworks start to focus on training an encoder that can produce feature maps that could be useful for downstream tasks. Yue et al. [25] proposed a framework named ts2vec, which firstly aims to extract semantic information in all resolutions and store it into the embedding series. Woo et al [23] developed a new framework called CoST. CoST designs a contrastive loss for learning discriminative trends via time domain representation, as well as seasonal representations via frequency domain representation. Wang et al.[22] proposed a task-driven multi-resolution contrastive learning framework, which aims at contrasting semantic information in different levels for medical time series. Similarly, SimMTM[4] proposed by Dong et al, is a pre-training framework for Masked Time series modeling that aims to reconstruct masked time points through the weighted aggregation of the masked data's augmentations to ensure the construction loss based training is meaningful. Kamarathi et al [17] proposed a model named Large Pre-Trained Time Series Models (LPTM) which introduces adaptive segmentation. This technique recognizes optimal segmentation for dataset specific time series in pre-training to efficiently train models with data across various domains. Duan et al [5] introduces MF-CLR, a framework for learning valuable representations of multi-frequency time series in a self-supervised model through use of a hierarchical mechanism that spans the feature dimensions across different frequencies. Grover et al [9] introduces Segment, Shuffle, and Switch (S3), a plug-and-play neural network layer that performs learnable shuffles and schema reattachment to improve time series representation learning. Liu et al[14] proposed TimesURL, a self-supervised framework that uses both frequency-based augmentation, as well as double Universums produced by instance and temporal-wise mixing to create hard negatives for enhancing contrastive learning.

Existing time series self-supervised frameworks often focus on bettering self-supervised learning schema. However, how to effectively save the learned embedding series is not studied. This problem is especially critical when the length of time series data is large. The redundant information saved in the embedding may also increase the cost of performing downstream tasks.

3 Problem Definition

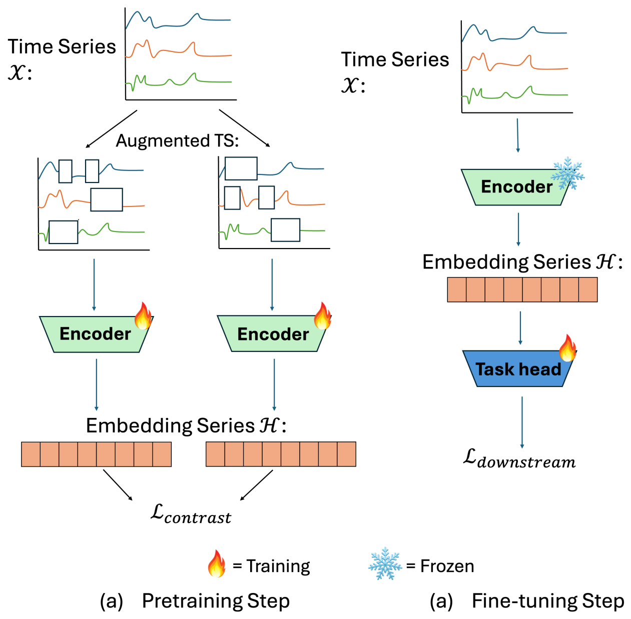
The overall self-supervised learning framework is illustrated in Figure 1. Given B number of time series data $\mathcal{X} = \{x_k\}_{k=1}^{B}$ where $x_k \in R^{C\times L}$, the contrastive self-supervised learning framework aims to pre-train an $Encoder(.)$ such that a feature maps $\mathcal{H}$ via:

$$\mathcal{H} = Encoder(\mathcal{X}) \tag{3.1}$$

where $\mathcal{H}$ consists of the embedding series of all samples ($\mathcal{H} \in R^{B\times C\times L}$). In this process, $Encoder(.)$ aims to preserve all semantic information without the guidance of human annotation. This is achieved by enforcing the embedding feature maps generated from two augmented version of time series are similar with each other, whereas the unrelated samples are far away from each other (Figure 1.a). The feature map $\mathcal{H}$ will be used in downstream tasks by fine-tuning a task-specific head (usually a linear layer or a simple MLP) (Figure 1.b).

In addition to obtaining $\mathcal{H}$, our goal is to obtain a compressed version of $\mathcal{H}$ denoted as $\mathcal{H}'$ ($\mathcal{H}' \in R^{B\times C\times L'}$) where $L' < L$. The obtained $\mathcal{H}'$ is more space efficient than $\mathcal{H}$ while still similar in downstream task performance accuracy. By compressing $\mathcal{H}$ into $\mathcal{H}'$, we aim to preserve crucial embedding representations of the time series while simultaneously reducing the amount necessary for downstream tasks by generating a compacted version of the feature map. In the rest of the paper, since we emphasize on time evolving behavior in the embedding space, we consider $\mathcal{H} = \{h_t\}_{t=1}^{L}$ as the a set of embedding features obtained through the encoder recording over time. Each $h_t \in R^{B\times C}$ represents all the embeddings obtained at a time stamp t. Furthermore, the k_{th} sample's embedding at time stamp i is denoted as $h_{i,k}$.

4 Methodology



Time Series $\mathcal{X}$:
Augmented TS:
Encoder
Encoder
Embedding Series $\mathcal{H}$:
$\mathcal{L}_{contrast}$
Time Series $\mathcal{X}$:
Encoder
Embedding Series $\mathcal{H}$:
Task head
$\mathcal{L}_{downstream}$
= Training
= Frozen
(a) Pretraining Step
(a) Fine-tuning Step

Figure 1: Contrastive Based Self-supervised Learning Framework

4.1 Motivation Intuitively, because time series data are often highly correlated, it is very likely that the produced embedding of two consecutive time stamps may be very similar. Therefore, we could generate a compacted version of $\mathcal{H}$ by removing consecutive similar embeddings. To illustrate the existence of redundancy, the t-SNE is shown in Figure 2.

Figure 2.a shows an embedding series $\mathcal{H}$ obtained from a time series dataset $\mathcal{X}$. Due to correlation existing at each time stamp, many time-stamp-wise embeddings h_t share strong similarities with their neighbors (e.g. h_{t-1} or h_{t+1}). For example, h_1 and h_2 highlighted by the blue arrow are similiar with each other. A similiar observation is found for h_2, h_3, and h_4. This observation led to the question of *whether or not it is necessary to retain all embeddings*, despite certain ones essentially being duplicates and providing no new semantic information. To produce a more space efficient embedding series, we remove the duplicate embeddings (Figure 2.b) by keeping only a single copy of the repetitive embeddings, forming a unique embedding series $\mathcal{H}'$ that is only half the size of original embedding series (shown in Figure 2.c). The compressed series still contains important and unique information obtained through model. To visualize similarity, the produced final embeddings of the original and compressed series are illustrated in Figure 2.d and Figure 2.e respectively. The final compressed embedding shares very similar behavior compared with the one generated through the original embedding series. This observation shows that the repetitive embeddings may contain less meaningful information, as well as that the removal of duplicates could minimize the necessary data storage requirement while having a small performance impact.

4.2 Proposed Compression Algorithm The overall structure of SimCompress is shown in Algorithm 1. The algorithm checks the similarity of the latest compressed embedding series with the current embedding in the original series, iteratively updating the current embedding. If the score is less than the established threshold, the current embedding is added to our compressed embedding series (Line 6-8). If the similarity score is greater than or equal to the threshold, then it skips the current embedding (Line 9-10). The algorithm does this for all embeddings in the original series.

The input for SimCompress is the original dataset embedding series, denoted as $h \in R^{B\times C\times L}$ (Line 1). On this input, the algorithm creates an array idx to store the indices of the elements used in our compressed embedding series, as well as our compressed embeddings tensor $h' \in R^{B\times C\times L'}$. idx is initialized as the index of the first element of h (Line 2), and h' is initialized as h_1 (Line 3). Next, the algorithm iterates through all h_n, at each iteration checking the similarity of h_i and the last element of h' (Line 5-6). If the similarity is less than our threshold δ, h_i is added to h', and i is added to idx (Line 6-8). If the similarity is greater than or equal to δ, the algorithm does nothing (Line 10). Once the similarities of all embeddings have been checked, the algorithm returns h' and idx (Line 13).

Intuitively, SimCompress computes similarity scores between different embeddings in the original series h, adding dissimilar embeddings determined by threshold δ to a compressed embedding series, h'. In addition to this, the idx array is continuously updated with the indices of the dissimilar h embeddings.

4.3 Similarity used in Line 6 Next, we introduce the proposed similarity measure for Algorithm 1. Given h_i and h_j represents the embedding of all data at time

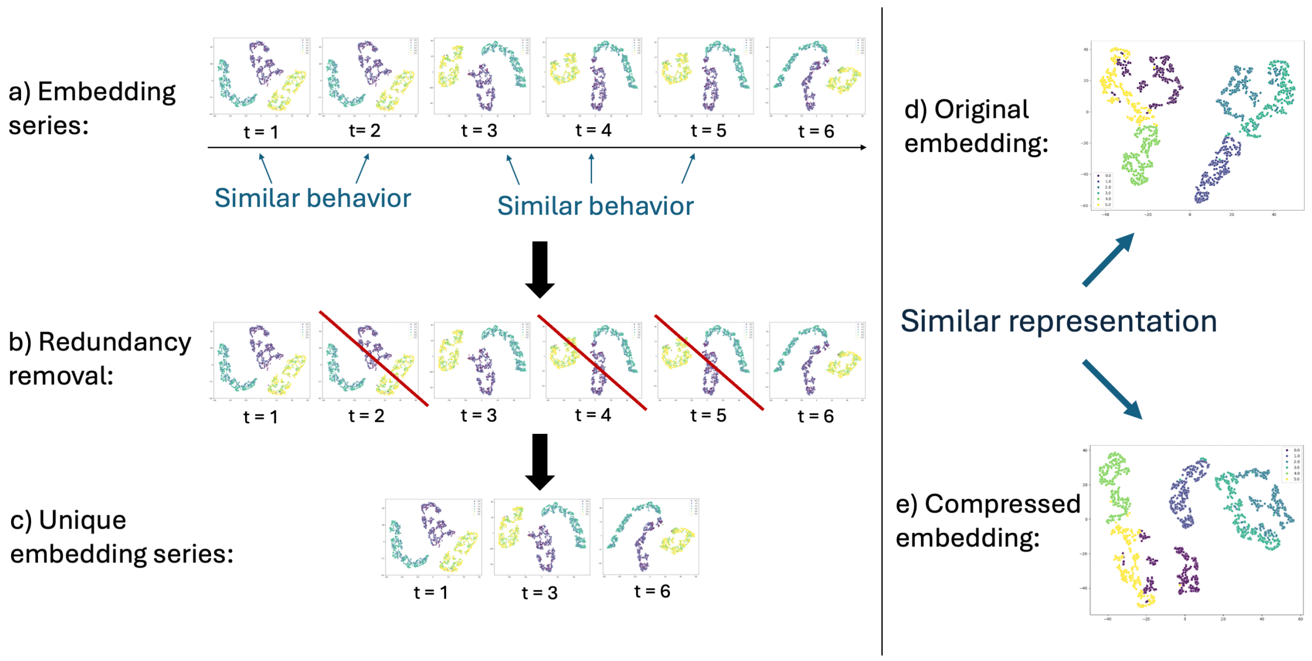


Figure 2: Redundancy Existed in Pretraining Feature Map

stamp i and j respectively, we produced our similarity scores by calculating the average cosine similarity between $h_{i,k} \in h_i$ and $h_{j,k} \in h_j$ for all samples:

$$sim(h_i, h_j) = \frac{1}{B} \sum_{k=1}^{B} \frac{h_{i,k}^T h_{j,k}}{||h_{i,k}|| |h_{j,k}||} \tag{4.2}$$

Intuitively, we can measure the distance of different time stamps in the embedding space to determine those that are similar to one another, therefore repetitive and unnecessary to store and use in downstream tasks.

5 Experiment Evaluation

In this section, we demonstrate that SimCompress can have better space efficiency while maintaining similar downstream task performance. Following the existing work, we use TCN as our backbone. We set the embedding size to 128, learning rate is 0.001, and the NVIDIA L4 GPU is used in the experiment. All our experiments are conducted via the Google Cloud Platform.

5.1 Experiment Setting For all the experiments unless otherwise mentioned, the threshold used for the proposed compression algorithm is 0.6. If the algorithm finds that the embedding series is not compressed (i.e., compressed series length is equal to the original length), the algorithm will decrement the threshold by 0.1 until a threshold results in the compressed series length being different than the original series length. Throughout the experiment, we used the SimMTM framework with their default parameters for pre-training. If the dataset has large size, we adjust the number of masked data augmentations to 2. For the datasets having a large number of samples, the pretraining step only uses 5000 samples per dataset.

Algorithm 1 SimCompress

```
1:  **Input**: embedding series $h \in R^{B \times C \times L}$
2:  **Output**: compressed embedding series: $h' \in R^{B \times C \times L'}$ where $L' < L$ and $idx$ array $idx$ consists of the sampled time stamp.
    /* Initialize index array and compressed series */
3:  $idx = [1]$
4:  $h' = h_1$ where $h_1 \in R^{B \times C}$ // For embedding vectors at time stamp $i$ in the series $h$
5:  **for** all $h_i \in h$ **do**
6:      **if** $sim(h'.\text{last}(), h_i) < \delta$ **then**
7:          $h'.\text{add}(h_i)$ /* Add $h_i$ to Compressed series */
8:          $idx.\text{add}(i)$ /* Add idx $i$ to idx array */
9:      **else**
10:         continue /* Skip $h_i$ */
11:     **end if**
12: **end for**
13: return $h'$ and $idx$
```

Table 1: Overall Performance Evaluation

Dataset/Method	Accuracy		Series Length	
	SimMTM	SimMTM+compression	SimMTM	SimMTM+compression
CinCECGTorso	0.9217	0.9275	70	**15**
ECG200	0.87	0.86	6	**3**
ECG5000	0.9238	0.9278	8	**7**
ECGFiveDays	0.8479	0.8815	8	**5**
NonInvasiveFetalECGThorax1	0.5481	0.4478	33	**8**
NonInvasiveFetalECGThorax2	0.569	0.4758	33	**10**
TwoLeadECG	0.7217	0.7155	6	**4**
Car	0.7833	0.7667	26	**14**
DodgerLoopDay	0.5975	0.5974	14	**9**
DodgerLoopGame	0.8898	0.8583	14	**4**
DodgerLoopWeekend	0.9603	0.944	14	**7**
Earthquakes	0.7194	0.7194	23	**21**
ElectricDeviceDetection	0.8601	0.8593	13	**9**
FordB	0.6815	0.6864	23	**11**
KeplerLightCurves	0.4586	0.3509	201	**1**
Lightning2	0.8033	0.7377	29	**6**
Lightning7	0.7534	0.7671	15	**6**
Plane	0.962	0.962	8	**5**
Trace	0.99	0.95	14	**6**
Wafer	0.983	0.9825	8	**4**
UMD	0.9861	0.9792	8	**4**
BME	0.9533	0.9333	7	**3**
ShapeletSim	0.5	0.5278	23	**20**
CBF	0.9822	0.9844	7	**5**
Mallat	0.4354	0.4183	45	**20**
ChlorineConcentration	0.5307	0.5544	9	**8**
TwoPatterns	0.9862	0.978	7	**5**
Avg.	**0.785**	0.77 (-2%)	24.89	**8.18 (x3.04)**

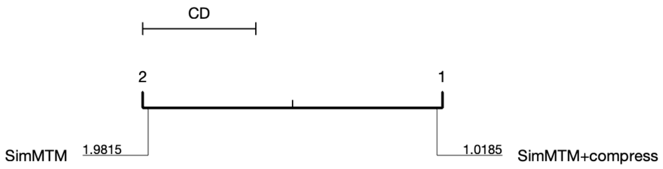
5.2 Datasets The methods are tested on the datasets that belong to the ECG, Sensor, or Simulated time series type in the UCR Time Series Classification Archive. Because we focus on datasets with larger lengths to be compressed, we omit datasets with a length less than or equal to 100 for the experiment. We use the original testing/training split to evaluate our method.

5.3 Baselines : To the best of our knowledge, there is not existing work specifically designed to compress embedding space series. We compared our method to the simple solution, without compression. In all the experiments, we used the latest state-of-the-art SSL framework, SimMTM, as our pre-training framework to test the proposed algorithm.

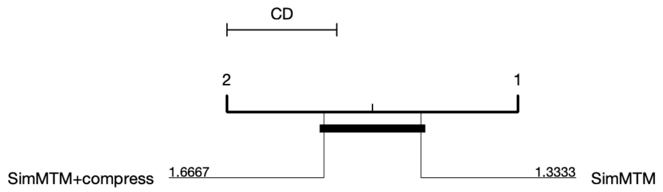
5.4 Evaluation Criteria We evaluate our proposed method against baselines for the semi-supervised classification task after the embedding series is obtained. The model is pretrained on the entire dataset without using labels. Then, given a training set associated with labels, a feedforward network is trained with 100 epochs to perform the classification task in the fine-tuning step. We evaluate the model in the testing set via the following two criteria:

- **accuracy**: The classification accuracy on the default testing dataset is used to evaluate the performance of our proposed method vs. original uncompressed result.
- **compression rate**: The length of the compressed series divided by the total number of original embeddings.

5.5 Experiment Result The comparsion result is shown in Table 1. According to the results, SimCompress has the ability to significantly reduce the size of



(a) Critical Difference Diagram for Compression Rate

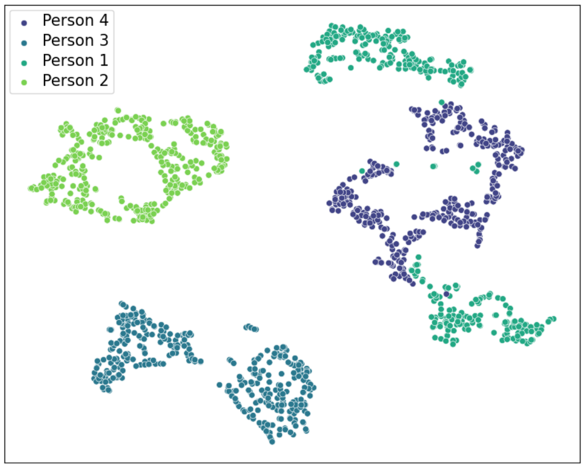


(b) Critical Difference Diagram for Accuracy

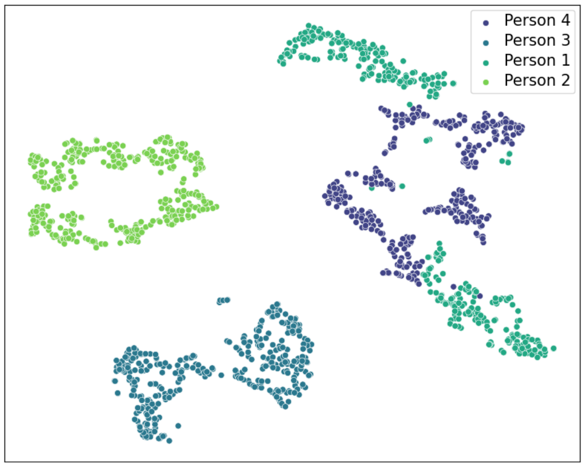
Figure 3: Critical Difference Diagram comparison between with and without compression

the embedding series. In average, the series length in the original embedding series is 24.89. After compression, it reduces to 8.18 (approximately 3 times less of the original embeddings on average). Besides the compression results, the accuracy performance results are very similar for SimMTM and SimMTM+compression. The original method has an average accuracy of 0.785 whereas the compression algorithm can still achieve an average accuracy of 0.77 (reduced approximately 2% of performance). Furthermore, a critical difference testing with significant level 0.05 is conducted on both accuracy and the compression rate. The results are shown in Figure 3b and Figure 3a respectively. All the methods fall in the bold bar, indicating they have statistically similar performance between each other. According to the figure, the proposed method has significantly shorter series length compared with the original method, whereas the accuracy performance is statistically similar to the original SimMTM solution. Overall, this result demonstrates that the proposed method can achieve similar classification accuracy results while simultaneously cutting out repetitive embeddings in the series, leading to saving space usage.

5.6 Parameter Testing In this experiment, we evaluate the impacts of our parameter (δ) to the model performance and compression ratio. We enumerate all possible thresholds from 0.1 to 1.0 by incrementally increasing by 0.1 and report both the downstream classification accuracy, as well as the compression ratio obtained through SimCompress. The result is shown in Table 2. According to the table, with thresholds over 0.5, our method continues to have high accuracy performance in all cases. The higher the threshold is, a lower compression rate will be achieved by the proposed algorithm, but the performance will potentially be higher. When setting a threshold too small, (i.e. smaller than 0.5), the performance accuracy reduces significantly because the proposed method overly focuses on compression (e.g. with threshold values from 0.1 to 0.3, all have a compression rate of 1/70). While the desired threshold ultimately depends on the data and user demand, according to the result threshold around 0.5 or 0.6 is capable of generously compressing the embedding series without significant performance loss.



(a) CinCECG t-SNE Visualization of Original Embedding Series



(b) CinCECG t-SNE Visualization of Compressed Embedding Series

Figure 4: t-SNE visualization of compressed vs. original embedding series

5.7 Data Embedding Visualization In this experiment, we demonstrate that the embedding repre-

Table 2: Parameter Testing

Threshold δ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Acc.	0.3036	0.2732	0.263	0.8754	0.9072	0.9246	0.9275	0.9225	0.9362	0.908
Compression Rate	1/70	1/70	1/70	6/70	11/70	15/70	19/70	28/70	41/70	70/70

sentation after compression is similar to the original embedding. We evaluated the performance of SimCompress on the CinCECGTorso dataset. The t-SNE visualization[21] of vectors with and without compression is shown in Figure 4. Figure 4b shows the t-SNE visualization of embedding vectors after compression, and Figure 4a shows the t-SNE visualization of embedding vectors before compression. The labels of each sample are highlighted in different colors. According to the figure, the separation of classes in both Figure 4a and Figure 4b is visually similar, and in addition, the location of data points in t-SNE figures are relatively the same. The result implies that the embeddings generated by both solutions are similar and the proposed approach is capable of removing repetitive embeddings, furthermore freeing up storage without significantly changing the performance.

6 Conclusion

In this paper, we introduce a compression algorithm named SimCompress. The algorithm aims at reducing the number of time stamps saved in the final embedding space by removing consecutive similar embeddings. The algorithm is independent of the pre-training process and can be adapted to any SSL framework that produces an embedding series. The experiment shows that SimCompress can achieve similar downstream task performance while using the compressed series that is much more compacted than the original embedding series.

References

[1] Wanpracha Art Chaovalitwongse, Oleg A Prokopyev, and Panos M Pardalos. Electroencephalogram (eeg) time series classification: Applications in epilepsy. *Annals of Operations Research*, 148(1):227–250, 2006.

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[3] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[4] Jiaxiang Dong, Haixu Wu, Haoran Zhang, Li Zhang, Jianmin Wang, and Mingsheng Long. Simmtm: A simple pre-training framework for masked time-series modeling. *Advances in Neural Information Processing Systems*, 36:29996–30025, 2023.

[5] Jufang Duan, Wei Zheng, Yangzhou Du, Wenfa Wu, Haipeng Jiang, and Hongsheng Qi. Mf-clr: multi-frequency contrastive learning representation for time series. In *Forty-first International Conference on Machine Learning*, 2024.

[6] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. Time-series representation learning via temporal and contextual contrasting. *arXiv preprint arXiv:2106.14112*, 2021.

[7] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems*, 32, 2019.

[8] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[9] Shivam Grover, Amin Jalali, and Ali Etemad. Segment, shuffle, and stitch: A simple layer for improving time-series representations. *Advances in Neural Information Processing Systems*, 37:4878–4905, 2024.

[10] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[11] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. *Advances in neural information processing systems*, 32, 2019.

[12] Sara Jamal and Joshua S Bloom. On neural architectures for astronomical time-series classification with application to variable stars. *The Astrophysical Journal Supplement Series*, 250(2):30, 2020.

[13] Argyro Kampouraki, George Manis, and Christophoros Nikou. Heartbeat time series classification with support vector machines. *IEEE transactions on information technology in biomedicine*, 13(4):512–518, 2008.

[14] Jiexi Liu and Songcan Chen. Timesurl: Self-supervised contrastive learning for universal time series representation learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 13918–13926, 2024.

[15] Pengfei Liu, Xiaoming Sun, Yang Han, Zhishuai He,

Weifeng Zhang, and Chenxu Wu. Arrhythmia classification of lstm autoencoder based on time series anomaly detection. *Biomedical Signal Processing and Control*, 71:103228, 2022.

[16] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6707–6717, 2020.

[17] Harshavardhan Prabhakar Kamarthi and B Aditya Prakash. Large pre-trained time series models for cross-domain time series analysis tasks. *Advances in Neural Information Processing Systems*, 37:56190–56214, 2024.

[18] Joseph W Richards, Dan L Starr, Nathaniel R Butler, Joshua S Bloom, John M Brewer, Arien Crellin-Quick, Justin Higgins, Rachel Kennedy, and Maxime Rischard. On machine-learned classification of variable stars with sparse and noisy time-series data. *The Astrophysical Journal*, 733(1):10, 2011.

[19] Stephanie Thiesen, Paul Darscheid, and Uwe Ehret. Identifying rainfall-runoff events in discharge time series: a data-driven method based on information theory. *Hydrology and Earth System Sciences*, 23(2):1015–1034, 2019.

[20] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. *arXiv preprint arXiv:2106.00750*, 2021.

[21] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[22] Yihe Wang, Yu Han, Haishuai Wang, and Xiang Zhang. Contrast everything: A hierarchical contrastive framework for medical time-series. *Advances in Neural Information Processing Systems*, 36, 2024.

[23] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. Cost: Contrastive learning of disentangled seasonal-trend representations for time series forecasting. *arXiv preprint arXiv:2202.01575*, 2022.

[24] CL Wu and Kwok-Wing Chau. Prediction of rainfall time series using modular soft computingmethods. *Engineering applications of artificial intelligence*, 26(3):997–1007, 2013.

[25] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8980–8987, 2022.

[26] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1476–1485, 2019.