Adaptive Resolution Loss: An Efficient and Effective Loss for Time Series Self-Supervised Learning Framework

Kevin Garcia
kevin.garcia09@utrgv.edu
The University of Texas Rio Grande Valley
Edinburg, TX, USA

Juan Manuel Perez
juan.m.perez02@utrgv.edu
The University of Texas Rio Grande Valley
Edinburg, TX, USA

Yifeng Gao
yifeng.gao@utrgv.edu
The University of Texas Rio Grande Valley
Edinburg, TX, USA

ABSTRACT

Time series data is a crucial form of information that has vast opportunities. With the widespread use of sensor networks, large-scale time series data has become ubiquitous. One of the most prominent problems in time series data mining is representation learning. Recently, with the introduction of self-supervised learning frameworks (SSL), numerous amounts of research have focused on designing an effective SSL for time series data. One of the current state-of-the-art SSL frameworks in time series is called TS2Vec. TS2Vec specially designs a hierarchical contrastive learning framework that uses loss-based training, which performs outstandingly against benchmark testing. However, the computational cost for TS2Vec is often significantly greater than other SSL frameworks. In this paper, we present a new self-supervised learning loss named, adaptive resolution loss. The proposed solution reduces the number of resolutions used for training the model via score functions, leading to an efficient adaptive resolution learning algorithm. The proposed method preserves the original model's integrity while significantly enhancing its training time.

CCS CONCEPTS

• **Computing Methodologies → Machine Learning**; • **Theory of Computation** → *Design and Analysis of Algorithms.*

KEYWORDS

TS2Vec, Time Series, Machine Learning, Data Mining, Algorithms

1 INTRODUCTION

Time series data is a crucial form of information that has vast opportunities [1, 6, 12, 14, 15, 18, 19] . Recently, with the widespread use of sensor networks, large-scale time series data have become ubiquitous. Such data gives us a dense amount of valuable information. The task of mining time series could help us harvest important trends, patterns, and crucial behaviors, which ultimately benefit various applications.

One of the most prominent problems in time series data mining is representation learning: transforming time series into low-dimensional representations that can represent their semantic similarity while benefiting various downstream tasks[20]. Recently, with the introduction of self-supervised learning frameworks (SSL) for image, video, and natural language representation learning [2, 3, 8–10, 13, 21], numerous research has focused on designing an effective SSL for time series data. One of the current state-of-the-art SSL frameworks in time series is called TS2Vec [20]. TS2Vec specially designs a hierarchical contrastive learning framework that uses loss-based training, which performs outstandingly against benchmark testing.

While TS2Vec outperforms existing state-of-the-art models, the model's computational cost is much heavier than other self-supervised learning frameworks. It utilizes hierarchical enumeration to compute the loss in each resolution of the time series, which significantly increases the computational burden.

This paper proposes a single resolution-based loss function to train the model. Intuitively, since every resolution series is highly correlated, we can train one resolution while improving the rest. In other words, instead of computing all resolutions, we designed an algorithm to adaptively select the resolutions, which can reduce the loss based on current and historical training loss. In the experiment section, we demonstrate how our proposed approach achieves similar accuracy and performance with classification tasks while showing that it is more efficient than the original TS2Vec model.

Our motivation behind this project lies within the improvement of the model, in order to advance Data mining as a whole. If our implementation is deemed suitable we can then improve similar models with a similar implementation leading to the availability of incredibly faster and robust models for the purpose of analyzing Time Series data.

In summary, the contribution of this paper is:

- Proposing an efficient self-supervised learning loss named adaptive resolution loss for time series self-supervised framework based on the TS2Vec framework.
- Evaluating the proposed method on time series classification tasks. The proposed approach achieves similar accuracy performance compared to TS2Vec while the execution speed is improved.

The following sections of the paper are organized as follows: Section 2 discusses the related work. Section 3 describes the problem definition. Section 4 describes the existing TS2Vec self-supervised learning framework. Section 5 introduces the proposed adaptive resolution loss. We present our experiments in Section 6 and provide a conclusive analysis of our research in Section 7.

2 RELATED WORK

Recently, the self-supervised learning (SSL) framework [2, 3, 8–10, 13, 21] is introduced for vision representation learning in the research domain of computer vision. The goal of SSL is to train a deep learning model to understand the semantic-level invariance characteristic through carefully designed pretext tasks from high-level semantic understanding related to image data (e.g. learning rotation-invariant representation for images) [7, 11, 21]. Recently, an increasing amount of time series representation learning research have been focused on designing the self-supervised deep learning framework [4, 5, 16?]. Most models are designed based on the unsupervised contrastive learning task [2].

Franceschi et al.[5] introduces an unsupervised contrastive learning framework by introducing a novel triplet selection approach based on segment's context. Similarly, Tonekaboni et al. proposed a framework named Temporal Neighborhood Coding (TNC) [16]. TNC aims to utilizes the temporal correlation along neighboring segments to learn the representation. Eldele et al. [4] introduces a Temporal and Contextual Contrast (TS-TCC) based framework. In TS-TCC, two types of augments, strong augmentation and weak augmentation are used to perform contrastive learning. Yue et al. [20] proposed a framework named ts2vec. The proposed framework introduces a random cropping based augmentation and a hierarchical loss to stabilize the obtained embedding. It achieved significantly better performance compared with previous methods.However, we found the computational burden for Ts2Vec is also higher than existing works.

3 PROBLEM STATEMENT

Next we will briefly describe the definitions and problem statement.

A **Time Series** $T = t_1, \ldots, t_L$ is a set of observations ordered by time where $t_i \in R^M$.

Similarly, we define a **Subsequence** $s_{p,q} = [t_p, \ldots, t_q]$ of a time series as a contiguous set of observations starting from position p and ends at q with length $l = p - q + 1$.

Given N time series data $\mathcal{D} = T_1, T_2, \ldots, T_N$, the goal of the **representation learning task** is to find a linear mapping $h(\cdot) : T_i \rightarrow e_i$ which maps all the *raw time series data* T_i into a K-dimensional latent space R^K such that all semantic levels are similar in the raw time series,and the data is preserved.

We next describe the SimCLR framework [2], the framework that is adopted in the TS2Vec model. It essentially creates the representations of the data without utilizing the label information. The framework is illustrated in Fig 1. Given a sample x from a training set $\mathcal{D}$, two different views of x, x_1 and x_2, are augmented based on a view augmentation operator $a(.)$. The views are treated as the data that shared the same semantic meaning as x. The latent representation of x_1 and x_2, h_1 and h_2, are then obtained through a nonlinear function $h(\cdot)$ modeled by a deep learning model. Another task-specific nonlinear mapping function $g(\cdot)$ is then applied to map h_1 and h_2 into z_1 and z_2 respectively. Lastly, the contrastive learning loss of x is defined based on z_1 and z_2 through a function $\ell(z_1, z_2)$.

4 TS2VEC: SELF-SUPERVISED LEARNING FOR TIME SERIES

4.1 TS2Vec Framework

Our model's base architecture is adopted from the original TS2Vec model architecture, which comprises of three main components: an Input Projection Layer $h(.)$, Random Cropping Augmentation $a_{croping}(.)$, and Time Stamp Masking Module $a_{mask}(.)$.

4.1.1 Random Cropping Augmentation $a_{croping}(.)$. :

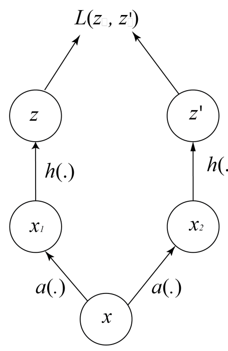


Figure 1: Contrastive Learning based Self-Supervised Learning Framework

Given an input time series, the model first generates two augmentations based on random cropping. Intuitively, the model generates two overlapped sub-sequences $s^i_{p,q}$ and $s^i_{p',q'}$ where $[p, q] \cap [p', q'] \neq \emptyset$.

4.1.2 Time Stamp Masking Module $a_{mask}(.)$. : A random masking is applied to generate an augmented context view by masking latent vectors at randomly selected timestamps. It essentially hides some of the information by creating a slightly different version of the data from the original, allowing the model to learn more robust representations.

Both the cropped-and-masked sub-sequences will pass through Input Projection Layer to obtain the embedding z_i and z'_i.

4.1.3 Input Projection Layer $h(.)$. Unlike classical contrastive learning, the function $h(.)$ generates an embedding series z_i (i.e. defined as another time series with K-dimensional observations in each time step) instead of simple K-dimensional vector. The generated series will be used to compute the hierarchical contrastive loss used in TS2Vec. Note that only the overlapping region shared by z_i and z'_i are used to compute loss. Following the architecture in TS2Vec, $h(.)$ function is modeled through a dilated convolution neuron network.

4.2 Hierarchical Loss Function

TS2Vec performs a hierarchical enumeration process to learn contrastive loss on top of the z_i, z'_i in multi-resolution.

Two different contrastive loss functions: temporal and instance-wise losses, are applied per resolution.

4.2.1 Temporal Loss: The Temporal Loss function regards two views of the same timestamp in z_i, z'_i as positive pairs, while those at differing timestamps as negative pairs. Concretely, the loss is written as:

$$\ell_r^{temp} = \sum_i \sum_t -log \frac{exp(d_{i,t} \cdot d'_{i,t})}{\sum_{t'} \left(exp(d_{i,t} \cdot d'_{i,t'}) + \mathbb{1}_{[t \neq t']} exp(d_{i,t} \cdot d_{i,t'}) \right)} \tag{1}$$

where d_i represents the time series down-sampled from z_i with average-based pooling of window size 2^{r-1}

4.2.2 Instance Loss: The Instance Loss function regards two embeddings generated from the same instance as positive pairs, while those at differing instances as negative.

$$\ell_r^{inst} = \sum_i \sum_t -log \frac{exp(d_{i,t} \cdot d'_{i,t})}{\sum_{j=1}^{N} \left(exp(d_{i,t} \cdot d'_{j,t}) + \mathbb{1}_{[i \neq j]} exp(d_{i,t} \cdot d_{j,t}) \right)} \tag{2}$$

4.2.3 Resolutions with relation to Loss Functions: With each iteration of the TS2Vec model, a general loss value is output which is produced by a combination of both instance and temporal loss functions per resolution of the dataset:

$$\ell_r^{overall} = \alpha \ell_r^{temp} + (1-\alpha) \ell_r^{inst} \tag{3}$$

where α is a hyper-parameter. And the final loss being:

$$\ell^{overall} = \sum_r \ell_r^{overall} \tag{4}$$

In the original TS2Vec model, the respective resolutions of the time series, whether the semantic information is stored in coarse or fine-grained resolutions, are not considered. Moreover, the current TS2Vec model utilizes all resolutions evenly throughout training, which are costly with larger datasets. In the next section, we will introduce our proposed loss adaptive resolution selection loss, which only computes a single resolution per epoch, which can significantly enhance the algorithm's efficiency.

5 ADAPTIVE RESOLUTION LOSS

In this section, we will present our proposed loss function. Developed to enhance the efficiency and accuracy of TS2Vec. Our function introduces a unique adaptive resolution selection process tailored to optimize computational power across different resolutions. By dynamically focusing on the most important resolutions during optimization, our method offers a substantial leap toward minimizing loss across all resolutions. The following subsections will discuss our solutions' specifics, motivation, core components, and the algorithm's complex operations. Furthermore, we will illustrate how this adaptive resolution loss reduces the model's training costs, demonstrating its advantage over existing methods.

5.1 Adaptive Resolution Selection

The algorithm summary is shown in Algorithm 1. Intuitively, by selecting the most important resolution to optimize, we can optimize the loss without using the full computation power in all resolutions. Since each resolution of d_i is highly correlated, training a specific resolution potentially minimizes the loss of all other resolutions. Specifically, we measure the resolution's importance by decreasing loss speed. If the loss is not decreasing or increasing, we will enforce the optimization algorithm to focus on this resolution. If the loss decreases very fast, we may pretend to prioritize other resolutions.

Our current implementation is split into multiple components with different sub-goals, which work together to create our final implementation. The algorithm consists of three steps, firstly, it aligns each resolutions based on a global resolution index. Secondly, it will assign an important score value to each resolution. Finally, the algorithm will randomly choose one resolution to optimize based on a multinomial distribution, which is determined by the important scores This process is repeated throughout the course of the training.

Algorithm 1 Overall Optimization Procedure

```
1: INPUT: $loss\_arr, net, data, a$
2: $loss\_arr \leftarrow Update\_Loss\_StopGrad(loss\_arr, net, data)$
3: $aligned, index \leftarrow Shift(loss\_arr, epochs, resolutions)$
4: $score \leftarrow Score(aligned, index, 1)$
5: $r \leftarrow Sampling(score)$
6: $loss \leftarrow loss\_arr[r].grad\_enabled()$
7: $model.update(loss)$
8: END
```

Algorithm 2 Shift function

```
1: INPUT: $arr, epochs, resolutions$
       {Flip the array. The index starts from left to right}
2: $flipped \leftarrow$ FlipVector($arr$)
3: $num\_resolution \leftarrow$ ObtainNumOfResolution($flipped$)
4: for i in range(epochs) do
5:     $aligned[i] \leftarrow$ FilledMissingRes($flipped[i]$, $flipped[i-1]$)
6: end for
7: OUTPUT: $aligned, num\_resolution$
8: END
```

5.1.1 Index shift function: Since the overlapping area between z and z' can be an arbitrary length between $[1, L-1]$, the resolution indices r in different epochs do not represent the same resolution. Therefore, the proposed algorithm will first align each resolution across different epochs. In order to accurately compute the score, we design a flip and shift function that properly reorders the resolutions. The algorithm shown in Algorithm 2 is a simplified representation. The algorithm consists of two steps. First, the algorithm reorders the resolutions and obtains the number of resolutions from the given dataset (Lines 2 & 3). Second, we interpolate the most current loss value for any resolution absent because of the variable length of the overlapping region (Lines 4 & 5). By properly pre-aligning the array of resolutions, we can now suitably assign them a score value.

Algorithm 3 Score Function

```
1: INPUT: $loss\_arr, index, epoch, a = 0.1$
2: $loss\_cur \leftarrow$ CroppedRes($loss\_arr$, $num\_res[epoch]$)
3: $loss\_prev \leftarrow$ CroppedRes($loss\_arr$, $num\_res[epoch-1]$)
4: $importance \leftarrow loss\_cur - loss\_prev$
5: $score \leftarrow softmax(a \cdot importance)$
6: OUTPUT: $score$
7: END
```

5.1.2 Score function: After aligning the resolutions across epochs, a score function is used to evaluate the importance of each resolution. The score function is described in Algorithm 3. Intuitively, at a given epoch, the algorithm computes the loss difference between current and previous epoch (Lines 2-4). Then the algorithm will compute the score via a softmax function which is based off of the resolutions loss. The greater the loss the greater the score is (Line 5). We can then use this newly generated probability array to pass on to sampling the resolution as seen in Algorithm 1 (Lines

4 & 5). Note that this function ignores any absented resolution in the original loss array, which means this array is usually smaller than $\log L$ size. Formally, the score is computed via $\frac{e^{x_j}}{\sum_k e^{x_k}}$.

5.1.3 Probability function: The probability function (named Sampling in Algorithm 1) is the final part of the adaptive resolution setting. It utilizes the probability array produced by the score function, as seen in Algorithm 1 (Line 4), to select the most weighted resolution (Line 5). It does so through a multinomial distribution random variable sampling function, which makes a selection based on the given probabilities. Finally, the chosen value is then used to update the weights of the resolutions in the model (Lines 6 & 7).

5.2 Advantage of Proposed Method

In the proposed learning framework, the back-propagation will only compute through a single-resolution loss function. Other than the wanted resolution, all other loss functions can be computed without considering gradient, which dramatically reduce the model's training cost. In the experiment, we demonstrate that the proposed method is significantly more efficient than the original TS2Vec model.

6 EXPERIMENT

We conducted our tests in a cloud-based environment offered by Google Colab. Google Colab provides an interactive environment that allows us to write and execute Python, which is ideal for machine learning and data analysis. We chose this environment as our platform because it can share work through different computers and access powerful computation resources. We evaluate the experiments with an NVIDIA Tesla T4 GPU in most of the experiments. When requiring more than 16GB GPU memory, we leveraged NVIDIA A100 GPU for our experiment.

6.1 Datasets

We use 6 different datasets from UEA/UCR Multivariate Time Series Classification Archive. The characteristic of the datasets are shown in Table 1. Each dataset exhibits variations in terms of dimension, number of samples, the length of the time series, and other characteristics.

6.2 Experiment Setup

We willingly adopt this approach to test our implementation's efficiency and effectiveness thoroughly. We compared our proposed method with the original TS2Vec framework through classification accuracy performance in the experiment. Following the evaluation protocol adopted in TS2Vec, we use the trained model $h(.)$ to convert the multivariate time series into K dimension representation. Then, the embedding is applied with a logistical regression classifier to perform the classification task. Regarding parameter optimization, we use the AdamW model with respective parameters and learning rate. For all comparison experiments, we repeated experiments five times for each dataset and reported the average performance.

Throughout the experiment, we set embedding size $K = 16$, number of dilated convolution layer to 2, and learning rate, within the AdamW optimization model, to $1e-3$. The final embedding is computed through global average pooling across all timestamps.

Datasets					
	Train Size	Test Size	Length	No. of Classes	Type
BasicMotions	40	40	100	4	HAR
ArticularyWordRecognition	275	300	144	25	MOTION
UWaveGestureLibrary	2238	2241	315	8	HAR
CharacterTrajectories	1422	1436	182	20	MOTION
NATOPS	180	180	51	6	HAR
HandMovementDirection	160	74	400	4	EEG

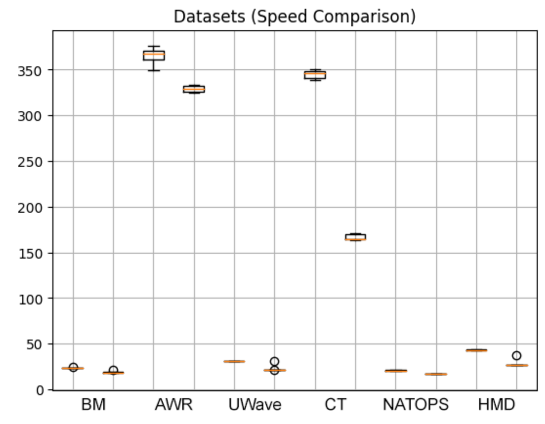
Table 1: Dataset information

6.3 Comparison Evaluation

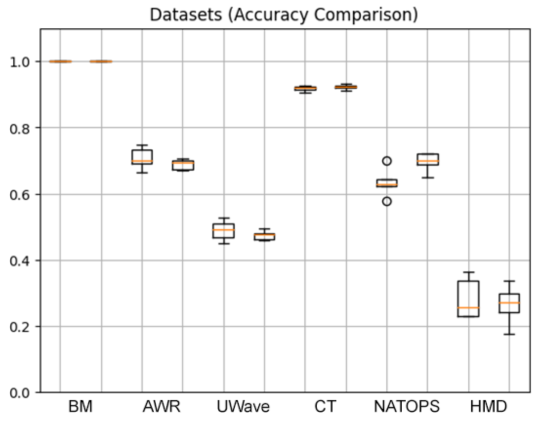
In this subsection, we will discuss our experiment's accuracy and efficiency results.

6.3.1 Accuracy: The average accuracy across five times experiments per dataset is shown in Table 2, and the boxplot of the accuracy is shown in Figure 2(b). The best performing dataset in terms of accuracy would be BasicMotions (Table 2), where both the proposed and original implementations scored an average accuracy of 100% . The worst performing dataset would be HandMovementDirection, where the proposed implementation scored an average of 26.49% accuracy while the original's implementation scored an average of 28.38%, which is slightly better. At its highest difference (when underperforming), the accuracy is about 1.89% (HMD) than the original model, which is slightly worse. In comparison, its lowest difference (not counting BM) would be 1.44% (UWave). At its highest difference (when succeeding), the accuracy is 6.23% (NATOPS), which exceeds any difference within underperformance. While at its lowest difference, the accuracy is 0.45%. Overall, the proposed method succeeds in maintaining a similar or even better accuracy than the original. However, this assessment does not account for the speed.

6.3.2 Efficiency: The average execution time across five times experiments per dataset is shown in (Table 2), and the boxplot of the accuracy is shown in Figure 2(b). The best performing dataset in terms of speed, would be NATOPS (Table 2), where our implementation finished its training with an average of 16.79 seconds, while the original's average training time was 20.88 seconds. The worst performing dataset would be AWR (Articulary Word Recognition). AWR had the highest training time for both the proposed implementation and the original, having a training time of 329.07 seconds for our implementation versus 364.85 seconds for the original. Although a notable difference in training times would be the ChartacterTrajectories dataset, where our implementation had a time of 166.80 seconds while the original had 344.98 seconds. This is a drastic difference when comparing both speed and accuracy since it performed very well while scoring a higher accuracy than the original. Notably, the speed for the proposed implementation always performed better than the original regardless of the dataset.



(a) Vs. Execution Time



(b) Vs. Performance

Figure 2: Comparison of speed and accuracy between the original TS2Vec model (left) and our proposed model (right) for different datasets.

	Proposed		Original	
Dataset	Speed (s)	Accuracy	Speed (s)	Accuracy
BasicMotions	**19.00**	**100%**	23.67	**100%**
ArticularyWordRecognition	**329.07**	68.87%	364.85	**70.67%**
UWaveGestureLibrary	**23.03**	47.50%	31.31	**48.94%**
CharacterTrajectories	**166.80**	**92.27%**	344.98	91.82%
NATOPS	**16.79**	**69.67%**	20.88	63.44%
HandMovementDirection	**28.88**	26.49%	43.24	**28.38%**

Table 2: Average Execution Time and Accuracy, Proposed Vs. TS2Vec

6.4 Embedding Visualization

To better understand the performance of our proposed modifications to the TS2Vec model, we visualize the embeddings generated by the model during the training process. TS2Vec generates two types of embedding - average pooling and without pooling, which we call flattened embedding. These are both high-dimensional, so to make sense of them visually, we used t-SNE (t-Distributed Stochastic Neighbor Embedding)[17], a powerful technique for dimensionality reduction particularly well suited for visualizing high-dimensional datasets.

We tested with six datasets to illustrate our model's ability to capture the inherent structure and relationships in the data.

We test two situations, global average pooling embedding, and without pooling embedding (flatten).

6.4.1 Average Pooling Embedding: For each dataset, we extract the average pooling embedding produced by the final layer of our model at the end of the training. We then apply t-SNE to reduce the dimensionality of these embeddings to two dimensions.

In the two-dimensional scatter plots, each point corresponds to a time series in the dataset, and the point's color indicates the class of the time series. The proximity of points in the plot reflects the similarity of their corresponding time series, as learned by the model through average pooling.

6.4.2 Without Pooling (Flatten) Embedding: We also visualize the embedding without pooling generated by the model. Similar to the average pooling embedding, we apply t-SNE to the flattened embedding and plot the resulting two-dimensional representation.

For both models in their average pooling and flatten embeddings, we observed that in the BasicMotions, and CharacterTrajectories datasets, the time series from different classes form distinct clusters, which indicates our model has effectively learned to differentiate between the classes in this dataset effectively.

We noticed the class separation is less pronounced for datasets such as ArticulatoryWordRecognition, UWaveGestureLibrary, HandMovementDirection, and NATOPS, reflecting these datasets' greater complexity and diversity.

However, even with the original TS2Vec, both model's accuracy is not too compromised, and our proposed method can work with these datasets at a faster speed, as is the case with CharacterTrajectories, which sees a slight improvement in its accuracy but manages to go through the datasets at a 50% faster rate. As a result, the visualizations in Figure 2 provide evidence that our modifications to the TS2Vec model maintain its ability to learn useful representations of time series data. Furthermore, they offer a valuable tool for interpreting the model's behavior and diagnosing any issues that may arise during training.

7 CONCLUSION

In this paper, we presented a method to improve the computational efficiency of the TS2Vec model, a state-of-the-art model for time series representation learning. Our method involves the use of adaptive resolution setting in the model's loss function, which allows us to reduce the computational load of the training process without sacrificing the model's performance.

Our experimental results confirm that our proposed method is effective. Our model achieved similar or better classification accuracy in a range of datasets compared to the original TS2Vec model while consistently reducing training time. These findings suggest that our method can be a valuable tool for researchers and practitioners working with large-scale time series data.

REFERENCES

[1] Stefan Baisch and Götz HR Bokelmann. 1999. Spectral analysis with incomplete time series: an example from seismology. *Computers & Geosciences* 25, 7 (1999), 739–750.

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

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

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

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

[6] André Gensler, Janosch Henze, Bernhard Sick, and Nils Raabe. 2016. Deep Learning for solar power forecasting—An approach using AutoEncoder and LSTM Neural Networks. In *2016 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 002858–002865.

[7] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. 2018. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728* (2018).

[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. 2020. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems* 33 (2020), 21271–21284.

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

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

[11] Eric Jang, Coline Devin, Vincent Vanhoucke, and Sergey Levine. 2018. Grasp2vec: Learning object representations from self-supervised grasping. *arXiv preprint arXiv:1811.06964* (2018).

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

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

[14] Hussein Sharadga, Shima Hajimirza, and Robert S Balog. 2020. Time series forecasting of solar power generation for large-scale photovoltaic plants. *Renewable Energy* 150 (2020), 797–807.

[15] Tetsuo Takanami and Genshiro Kitagawa. 1991. Estimation of the arrival times of seismic waves by multivariate time series model. *Annals of the Institute of Statistical mathematics* 43, 3 (1991), 407–433.

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

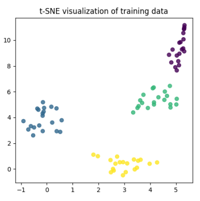
[17] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[18] Panayiotis Varotsos, Nicholas V Sarlis, and Efthimios S Skordas. 2011. *Natural time analysis: the new view of time: precursory seismic electric signals, earthquakes and other complex time series*. Springer Science & Business Media.

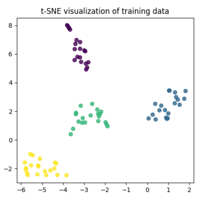
[19] Jin Wang, Ping Liu, Mary FH She, Saeid Nahavandi, and Abbas Kouzani. 2013. Bag-of-words representation for biomedical time series classification. *Biomedical Signal Processing and Control* 8, 6 (2013), 634–644.

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

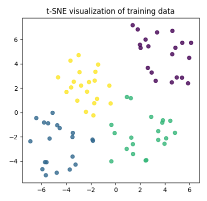
[21] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. 2019. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1476–1485.



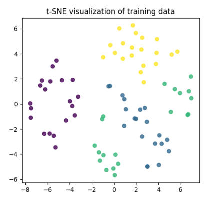
(a) BM TS2Vec (avg pool)



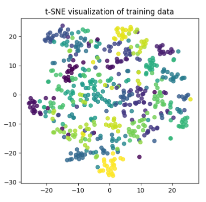
(b) BM Proposed (avg pool)



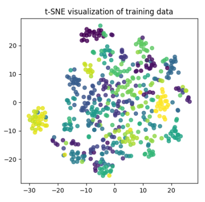
(c) BM TS2Vec (flatten)



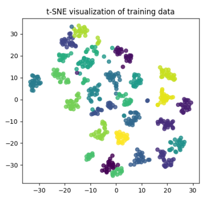
(d) BM Proposed (flatten)



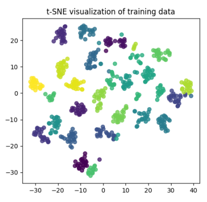
(e) AWR TS2Vec (avg pool)



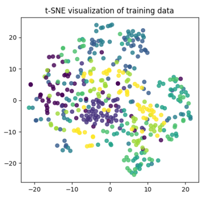
(f) AWR Proposed (avg pool)



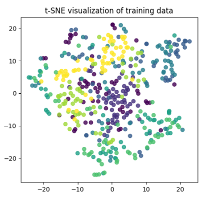
(g) AWR TS2Vec (flatten)



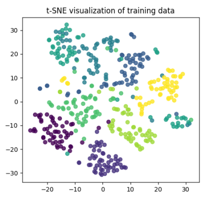
(h) AWR Proposed (flatten)



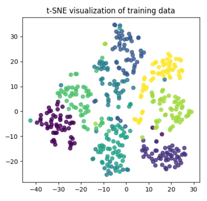
(i) UWave TS2Vec (avg pool)



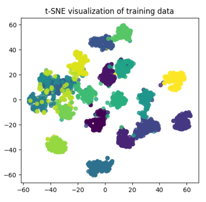
(j) UWave Proposed (avg pool)



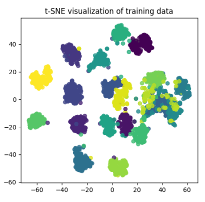
(k) UWave TS2Vec (flatten)



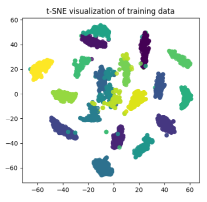
(l) UWave Proposed (flatten)



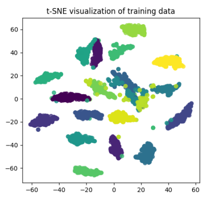
(m) CT TS2Vec (avg pool)



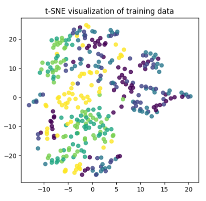
(n) CT Proposed (avg pool)



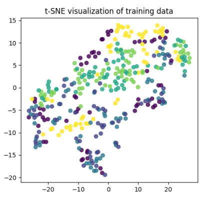
(o) CT TS2Vec (flatten)



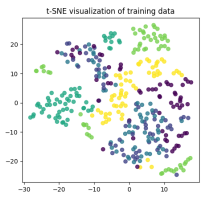
(p) CT Proposed (flatten)



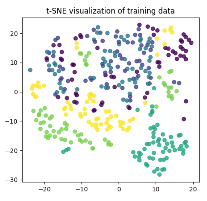
(q) NATOPS TS2Vec (avg pool)



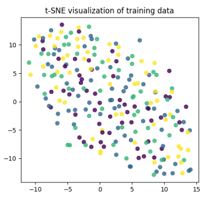
(r) NATOPS Proposed (avg pool)



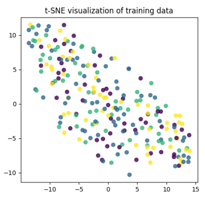
(s) NATOPS TS2Vec (flatten)



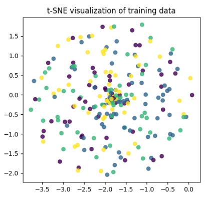
(t) NATOPS Proposed (flatten)



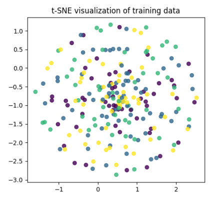
(u) HMD TS2Vec (avg pool)



(v) HMD Proposed (avg pool)



(w) HMD TS2Vec (flatten)



(x) HMD Proposed (flatten)

Figure 3: Visualizing embedding instances via t-SNE [17].