SN Computer Science (2023) 4:832
https://doi.org/10.1007/s42979-023-02159-4

ORIGINAL RESEARCH

# A Deep Convolutional Neural Network for Time Series Classification with Intermediate Targets

**Aboozar Taherkhani**[1] · **Georgina Cosma**[2] · **T. M. McGinnity**[3,4]

Received: 12 May 2021 / Accepted: 18 July 2023


## Abstract

Deep Convolutional Neural Networks (CNNs) have been successfully used in different applications, including image recognition. Time series data, which are generated in many applications, such as tasks using sensor data, have different characteristics compared to image data, and accordingly, there is a need for specific CNN structures to address their processing. This paper proposes a new CNN for classifying time series data. It is proposed to have new intermediate outputs extracted from different hidden layers instead of having a single output to control weight adjustment in the hidden layers during training. Intermediate targets are used to act as labels for the intermediate outputs to improve the performance of the method. The intermediate targets are different from the main target. Additionally, the proposed method artificially increases the number of training instances using the original training samples and the intermediate targets. The proposed approach converts a classification task with original training samples to a new (but equivalent) classification task that contains two classes with a high number of training instances. The proposed CNN for Time Series classification, called CNN-TS, extracts features depending the distance of two time series. CNN-TS was evaluated on various benchmark time series datasets. The proposed CNN-TS achieved 5.1% higher overall accuracy compared to the CNN base method (without an intermediate layer). Additionally, CNN-TS achieved 21.1% higher average accuracy compared to classical machine-learning methods, i.e., linear SVM, RBF SVM, and RF. Moreover, CNN-TS was on average 8.43 times faster in training time compared to the ResNet method.

**Keywords** Classification · Convolutional neural network · Intermediate targets · Time series analysis

## Introduction

Time series (TS) datasets are obtained by recording a series of time-dependent observations. TS covers a broad range of applications, such as investigation of market prices, prediction of epidemic spread, speech signal processing, electrocardiogram (ECG) investigation, understanding the brightness of a target star, manufacturing, and weather forecasting [18, 42, 47, 48, 82]. Time Series Classification (TSC) is an important part of TS data mining that has been used in many areas. For example, in medical science, classification of TS datasets generated from Electrocardiogram (ECG) data has been used for heart disease diagnosis [59, 73], and Electroencephalogram (EEG) signal data are used as a primary tool for seizure onset detection [3]. TSC has been used for different applications, such as activity recognition [2, 57, 69]; and in industry, classification of TS generated from different sensors such as gas pressure sensors, and thermometers play important roles in industrial control processes [47, 48].

Deep learning (DL) algorithms have exhibited impressive capabilities in image processing and big data analysis [61, 70]. DL uses several layers of processing elements to extract high-level abstraction from input data. One of the most popular approaches in DL is the Convolutional Neural Network (CNN). CNN is inspired by biological visual systems and has

✉ Aboozar Taherkhani
aboozar.taherkhani@dmu.ac.uk

✉ Georgina Cosma
g.cosma@lboro.ac.uk

T. M. McGinnity
tm.mcginnity@ulster.ac.uk

1 School of Computer Science and Informatics, De Montfort University, Leicester, UK

2 Department of Computer Science, Loughborough University, Loughborough, UK

3 School of Science and Technology, Nottingham Trent University, Nottingham, UK

4 Intelligent Systems Research Centre, Ulster University, Derry, UK

Published online: 28 October 2023

been used in various machine vision tasks [20, 40, 42, 65, 72]. The CNN has gained much interest and popularity due to its capabilities of processing raw data, eliminating the requirement to pre-process data using feature extraction methods. Indeed, the feature extraction property of convolutional layers, and the powerful training methods of the CNN, are reasons for the high performance of CNNs on image classification. CNNs have been widely used in computer vision tasks [16, 33], object detection [35], bioinformatics [54], economics [33], and natural language processing [75].

The focus of deep CNNs has mainly been on image processing, and the application of CNNs for TS data is only now starting to emerge [4]. TS datasets have different characteristics compared to images, and therefore, specific CNN structures are required to optimally process such data [47, 48].

Deep CNNs usually have numerous learning parameters and consequently need large training datasets. Some TS tasks, such as EEG classification, usually do not have large volumes of data for training deep learning algorithms with good generalizing ability. Yannick et al. [78] found that many authors who adopt deep learning methods for EEG processing have suggested that more training data would improve the performance of their deep models. Additionally, the number of training data is important when intra-subject models are used. In intra-subject models, the data of a single subject are used to train the model related to the subject. The data of intra-subject models have less variability which can lead to high performance [78]. However, in intra-subject models, each model is trained with a limited number of training samples, corresponding to a single subject, which often contains a small number of signal recordings. This need for a large training sample set poses a specific problem for intra-subject situations [46, 60]. However, in general, there are many application areas where the collection of large training datasets may not be feasible. For example, in medical situations where data are collected in hospitals from patients with epilepsy, it may be difficult to collect a high number of training samples from a specific patient. Collecting such data could include years of recordings collected from different subjects with a specific disease [78].

Moreover, a deep CNN usually uses a supervised learning approach that requires labeled data, and labeling is a time-consuming task, particularly for large datasets. Therefore, there is a need for new techniques that can be used to train deep neural networks with a relatively small number of training samples. Liu et al. [49] have highlighted that designing deep learning models to learn from fewer training samples will have a significate effect on the future progress of deep learning methods.

In this paper, inspired by the intermediate concept of the brain [43], intermediate outputs are constructed to control learning in hidden layers of a deep learning method to improve the performance of the method. It is proposed to have new intermediate outputs extracted from different hidden layers instead of having a single output to control weight adjustment in the hidden layers during training. Intermediate targets which are different from the main target are used as labels to train the intermediate outputs. The intermediate targets control the creation of features in the hidden layers of deep learning methods to generate more informative features in hidden layers. Consequently, they improve accuracy of the deep learning. Additionally, the proposed CNN artificially increases the number of training instances using the original training samples. The proposed CNN-TS approach converts a classification task with original training samples to a new (but equivalent) classification task that contains two classes with a high number of training instances. The proposed method receives two TS as inputs, and it extracts features from the two applied inputs using intermediate outputs and subtracts the features to measure the distance of two TS. Distance-based methods are well-known methods in classical TSC but have not been explored in detail in the deep learning domain.

The structure of the paper is as follows. In the section "Related Works" a brief review of TSC methods is presented. The proposed method, CNN-TS, is described in the section "Proposed Method". Simulation results are demonstrated in the section "Results" before the conclusion in the section "Discussion and Conclusion".

## Related Works

### Time Series Classification

The high-dimensional and ordered properties of TS data and the redundancy in TS resulting from their highly autocorrelated properties make TSC a challenging task [46]. TSC requires machine-learning methods that are compatible with the characteristics of TS to process a sequence of observations following each other in time [46]. There exist different methods to classify TS data, which can be summarized into three main categories: model-based, feature-based, and distance-based techniques [37, 76, 82].

In model-based classification methods, a collection of TS data is used to build a model. Usually, a model is built for each class using the TS belonging to that class. Then the class of an unknown data sample is determined by each model (i.e., built for each class) to evaluate which is the best fit for the unknown data sample [37, 76]. For example, the Autoregressive model is a model-based method that is used for TSC. In Autoregressive models, it is assumed that the TS satisfies the stationary assumption, which cannot be followed in every situation [82].

The Markov Model (MM) and the Hidden Markov Model (HMM) are two other model-based methods which are used for non-stationarity TS [5].

Feature-based techniques are commonly used in classical TSC approaches to reduce the dimensionality of samples in TS data using different feature extraction methods. Simple statistical methods such as mean and variance, or more complicated methods such as spectral feature extraction methods can be used in the feature-based techniques [46]. Discrete Fourier Transform and Discrete Wavelet Transform (DWT) are two examples of spectral feature extraction methods. The spectral methods usually transform the time domain into the frequency domain and take a number of low-frequency harmonics that contain most of the TS energy. Eigenvalue methods, such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD), are other dimensionality reduction methods that can be applied to TS; these usually have better performance compared to spectral methods as they are calculated in an optimal way. However, they are not suitable for large datasets [37].

In distance-based classification methods, distances between TS are measured, and a method such as k-Nearest Neighbors (k-NN) is used to classify an unknown sample [46]. The distances between an unknown sample and the training samples are calculated and the unknown sample is classified based on its distances from the training samples [7]. Euclidean distance and a Dynamic Time Warping distance (DTW) have also been successfully used with one nearest-neighbor classifier for TSC [34]. DTW has been shown to be robust to TS variation generated by translations or dilations and it is considered a strong solution for TS distance measurement. DTW performs a local comparison instead of measuring similarity by considering the high-level structure in a long TS [7].

In addition to the above-mentioned k-NN distance-based classification method, the distance features method for TSC is another main distance-based TSC. In distance features method, new representations of TSs are created using the distance between TSs. In the global distance features method, the distances between a time series and other time series in the training data are calculated, for the full length of time series, to extract global distance features, which are then used as a feature vector to represent the time series. This learning method is from a general learning approach which is called learning in dissimilarity space [10, 62]. Gudmundsson et al. [25] have used two distance measure methods based on DTW to create global distance features. Then, an SVM model is used to classify the global features. Kate [36] has used different distance measures including DTW and Euclidian Distance (ED) to construct global distance features to be classified by SVM. Giusti et al.'s [22] generalized Kate's [36] approach by extending the distance features method to other domains, such as frequency, which is different from the time domain in the previous global distance feature method. The computation cost of the global distance feature method is a significant drawback of the method. A high number of training samples and consequently a high number of pairs lead to a large input dimensions. Jain and Spiegel [32] proposed to use Principal Component Analysis (PCA) to reduce the dimensionality of global distance features created using DTW method to be classified by SVM. Kenji et al. [38] reduce the computation cost by calculating the distance between a time series and n subset of time series as prototypes out of the total time series. They consider the prototype selection as a feature selection method.

Local Distance Features (LDFs) are used as another sub-group of distance features method for TSC [1]. In LDF methods, the distance between some local patterns is calculated. Ye and Keogh [79] have proposed an important LDF method where sub sequences of time series called shapelets are selected to represent different classes. Specific shapes in time series, i.e., specific subsequence of time series, can be determined by experts and they can be used to identify the class of the time series. The original shapelets method [79] enumerates all possible subsequences of time series to find the appropriate shapelets; this has a high computation cost. Consequently, other works have been carried out to reduce the computational cost of the shapelets method [28, 56, 64, 80], and to learn appropriate shapelets [23]. Hills et al. [29] proposed a method for finding the most discriminative shapelets. Then, a vector of feature matrices is constructed by the distance between each time series and the selected shapelets. The minimum distance between a shapelet and all the subsequences of a time series with the size of the shapelet is considered as the distance between the time series and the shapelet. Li and Lin [44] have used an evolutionary method to find shapelets called Separating References (SRs) that effectively separate different classes. The distance between the SRs and the series from different classes are such features that can separate the classes with large margins. Despite the research summarized above, there remains room for applying distance features approaches to deep learning neural networks for TSC.

## Deep Neural Networks for Time Series Classification

Classical feature-based methods for TSC do feature extraction and classification separately, and their performance relies on the quality of the extracted features. There is no specific method for extracting high-quality features for different TS, and different tasks need particular expertise to extract appropriate features [46]. However, a CNN can merge feature extraction and classification into a single process and the network is trained to extract appropriate features to improve the network's performance. For instance, Lin et al. [46] proposed an end-to-end deep learning structure called Group-Constrained Convolutional Recurrent Neural Network (GCRNN) for TSC. A

network of convolutional layers is used to extract features from the input TS. The extracted features are input to a recurrent network to capture the temporal characteristics of TS. The output of the recurrent network is fed to a fully connected network with sparse group lasso regularization.

Long Short-Term Memory (LSTM) is a recurrent neural network that is designed for analyzing TS. LSTM requires more computing resources than CNN, and training an LSTM is more computationally expensive than training a CNN. Additionally, recent research has shown that certain convolutional architectures for different applications, such as audio synthesis, machine translation, and skeleton-based action prediction, can reach state-of-the-art accuracies [15, 17, 19, 21, 48, 58]. For instance, Liu et al. [77] have proposed a CNN for fault diagnosis by proposing a dislocate layer at the input level. Their proposed layer extracts windows of TS in different intervals of an original signal. Liu et al.'s [77] experimental results have shown that their proposed CNN-based method has good performance in such industrial applications.

The end-to-end Multi-scale Convolutional Neural Network (MCNN) [13] applies different transformations using downsampling transformation in the time domain; additionally, it performs spectral transformation in the frequency domain on an input signal. Then, different convolutional layers are used to extract high-level features from the original input and the transformed versions of the input. The extracted features are fed to a fully connected layer, and then, a Softmax output layer is used to classify the input.

Liu et al. [47, 48] proposed a deep learning method called Multivariate Convolutional Neural Network (MVCNN) that considers the multivariate properties of TS data. They utilized a $1 \times 1$ convolution filter for layers that are close to the input layer to extract features that specifically come from each variant. However, the shared filter among different variants can mix the data from different sources during training.

The main property of the above-mentioned methods in this section is the use of a CNN's ability for automatic feature extraction, and for this reason, they can be considered as feature-based techniques for TSC using the deep learning method.

In model-based classifiers or generative models, the first goal is to find a suitable representation of TS before training a classifier [17, 41]. In a model-based method, an unsupervised method is often used to model the TS. For example, some deep learning methods used stacked denoising auto-encoders (SDAEs) to model input signals [8, 30]. RNN auto-encoders have also been used to generate a representative TS, and then, a classifier such as SVM was used for classification [52, 53, 63]. Echo State Networks (ESNs) project the input TS inside a reservoir of a recurrent neural network to reconstruct a representation of the input TS, and then, the learned representation of input TS is used for classification [6, 9, 12, 51]. Antoniades et al. [3] have proposed an Asymmetric–Symmetric Autoencoder (ASAE) to map a scalp EEG to an intracranial EEG (iEEG), since recording an iEEG is an invasive method to record the brain activity and is also expensive to implement. The model is used for the classification of Intracranial Epileptic Discharges (IEDs) and non-IED. Wang et al. [68] and Mittelman [55] have designed deep neural networks that reconstruct a multivariate TS using a deconvolutional operation followed by an upsampling method.

In summary, the literature shows that there exist several deep learning methods to extract features and to classify TS. These methods, like feature-based methods in classical TSC, are focused on features that are extracted by a number of convolution layers. Additionally, the literature review revealed that there exist a considerable number of deep learning methods that classify TS using model-based techniques. Although distance-based classification of TS data is thoroughly investigated in traditional TSC methods, distance-based methods have not been investigated in the deep learning field as much as they have been studied in classical methods for classification of TS. The review shows that it would be useful to design deep learning methods that are based on the principle of distance-based classification methods. This paper proposes a CNN that takes as input a pair of TS, evaluates their distance, and predicts whether these two TS are close enough to be in the same class. The ability of the proposed CNN to take two TS as inputs increases the number of different instances that are available to train the proposed network. The proposed method is described in detail in "Time Series Classification".

## Proposed Method

In this section, a technique to synthetically increase the number of training instances and create an extended dataset is first described. Then, a CNN called CNN-TS is proposed to classify the extended TS data. The structure of the proposed CNN is designed to be compatible with the extended dataset.

## A Method for Synthetically Increasing the Number of Training Samples

Suppose a training dataset is $X = \{(\boldsymbol{x}_1, c_1), ..., (\boldsymbol{x}_N, c_N)\}$, where $\boldsymbol{x}_i$ is a p-dimensional input vector, i.e., $\boldsymbol{x}_i \in R^p$, and $c_i$ is the output corresponding to $\boldsymbol{x}_i$, and $c_i \in \{1, 2, \ldots, K\}$, where $K$ is the total number of classes. $\boldsymbol{x}_i$ contains $p$ sequential elements of a TS. $N$ is the number of training samples in the dataset. The training goal is to fit a classifier, $C(x)$, using training data. The trained classifier can then be used to find the class labels of unseen testing data samples.

Different permutations with replacement of two samples from the original dataset, $X$, are picked to construct a new training dataset called $X^n$. Each instance from the new training set, $X^n$, contains two samples from the original dataset, $X$. Therefore, the new dataset has $T = P^R(N, r = 2) = N^r = N^2$ training instances, where $P^R$ stands for Permutations with Replacement, $N$ is the number of samples in the original dataset, and $r = 2$ is the number of samples that are selected. Therefore, the newly constructed dataset has a higher number of training instances, $N^r$ compared to $N$, which is suitable for a deep neural network since it needs a high number of training samples.

The structure of a constructed training dataset is shown in (1)

$$\begin{aligned}\tilde{X}^n = \{&(\boldsymbol{x}_1, \boldsymbol{x}_1, c_1, c_1), (\boldsymbol{x}_1, \boldsymbol{x}_2, c_1, c_2) \ldots \\ &(\boldsymbol{x}_2, \boldsymbol{x}_1, c_2, c_1), (\boldsymbol{x}_2, \boldsymbol{x}_2, c_2, c_2) \ldots \\ &(\boldsymbol{x}_N, \boldsymbol{x}_1, c_N, c_1), (\boldsymbol{x}_N, \boldsymbol{x}_2, c_N, c_2) \ldots \\ &(\boldsymbol{x}_N, \boldsymbol{x}_N, c_N, c_N)\},\end{aligned} \tag{1}$$

where $\tilde{X}^n$ consists of several data arrangements denoted as $(\boldsymbol{x}_i, \boldsymbol{x}_j, c_i, c_j)$, where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, are two training samples from the original dataset, $X$, along with $c_i$ and $c_j$ which are the labels corresponding to each input, respectively. The two inputs, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, within an arrangement could belong to the same class or they could belong to two different classes. A third label, $c_{ij}$, is constructed based on this arrangement, and it shows whether $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same class. The new training set is as follows:

$$\begin{aligned}X^n = \{&(\boldsymbol{x}_1, \boldsymbol{x}_1, c_1, c_1, c_{11}), (\boldsymbol{x}_1, \boldsymbol{x}_2, c_1, c_2, c_{12}) \ldots \\ &(\boldsymbol{x}_2, \boldsymbol{x}_1, c_2, c_1, c_{21}), (\boldsymbol{x}_2, \boldsymbol{x}_2, c_2, c_2, c_{22}) \ldots \\ &(\boldsymbol{x}_N, \boldsymbol{x}_1, c_N, c_1, c_{N1}), (\boldsymbol{x}_N, \boldsymbol{x}_2, c_N, c_2, c_{N2}) \ldots \\ &(\boldsymbol{x}_N, \boldsymbol{x}_N, c_N, c_N, c_{NN})\},\end{aligned} \tag{2}$$

where

$$c_{ij} = \begin{cases} 1, c_i = c_j \\ 0, c_i \neq c_j \end{cases} i,j = 1, \ldots, N. \tag{3}$$

Therefore, a new binary classification task emerges from the original multi-class classification task. In the new training set shown in (2) each training instance, i.e., $(\boldsymbol{x}_i, \boldsymbol{x}_j, c_i, c_j, c_{ij})$, includes: the first sample, $\boldsymbol{x}_i$, and the second sample, $\boldsymbol{x}_j$, and their corresponding labels, $c_i$ and$c_j$, from the original dataset, $X$. Additionally, each training instance contains the fifth element, $c_{ij}$, which holds a binary value, 1 if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same class and 0 otherwise. Note that in this paper, 'sample' is used to refer to each item in the original dataset, i.e., $X$, and 'instance' is used to refer to each item in the newly constructed dataset, i.e., $X^n$. A figure-based description of the permutation on the input samples used

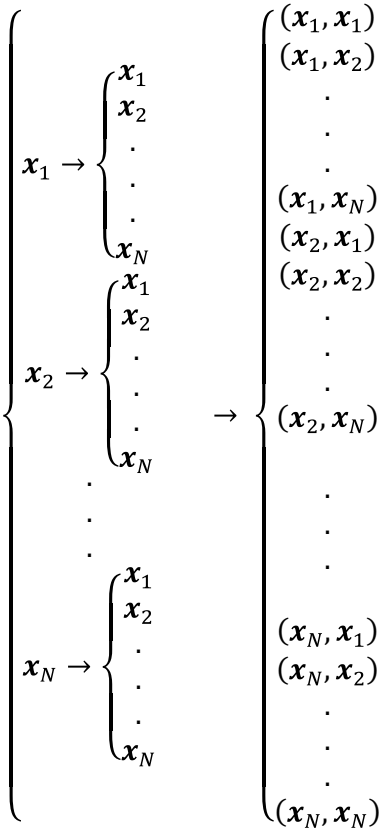


**Fig. 1** A figure-based description of the permutation on the input samples used to construct the new large data described in (2)

to construct the new large data described in (2) is shown in Fig. 1.

The method increases the number of training samples by $N^2$ training instances that are generated by selecting 2 samples from an original training dataset of N samples but may excessively boost the number of training data if N is large. When N grows, the number of samples for the proposed method grows by the power of 2, i.e., $N^2$. Given the impact of $N^2$, an original training dataset with a high number of training samples could increase the number of generated training instances substantially, and not all the newly generated training instances may be required. To control the number of generated training instances, an under-sampling method can be used to control the number of newly generated instances. Thus, instead of selecting two samples from all the original training samples, a subset of representative samples that has similar characteristics as the original samples are selected using the under-sampling technique proposed by Zhang and Mani [81]. This approach controls the number of generated training instances, especially when the original training dataset has a high number of training samples. It helps to continue the learning for higher numbers of training epochs in shorter time duration, because of a lower number of training instances. If the number of representative samples is $M$ which is smaller than N, then the total number of new instances being generated for the proposed method is $N \times M < N^2$. The proposed method can increase the number of training instances intensively for small data by setting a high value for $M$, while providing a smaller increase in the number of training instances for datasets that already have a high number of training samples, by setting $M$ to a small value.

## Proposed Method for Time Series Classification Using Synthetically Extended Training Samples

In this section, a structure for CNN is designed to be trained on the new training set, $X^n$, which contains a high number of training instances. Then, proposed intermediate targets are described. The proposed network has two inputs, $\text{In}_1$ and $\text{In}_2$, and it accepts two training samples, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ which are in each instance of $X^n$, i.e., $(\boldsymbol{x}_i, \boldsymbol{x}_j, c_i, c_j, c_{ij})$. The structure of the proposed deep neural network is shown in Fig. 2. The network compares the two inputs, $\text{In}_1$ and $\text{In}_2$, and returns a main output, i.e., $O_m$. The main output, i.e., $O_m$, corresponds to label $c_{ij}$.

Block 1 and Block 2 in the CNN network shown in Fig. 2 are composed of a number of layers of neurons including convolutional layers, and the blocks extract high-level features from pairs of TS inputs. The extracted features are subtracted to make a set of features that reflect the distance of the two inputs to assist the network to make an accurate comparison between the two inputs.

Then, the extracted features are processed by the next three components (Block 3, Block 4, and Global Average Pooling) to generate the main output, $O_m$. The network generates the main output based on the comparison of the two inputs. The main output, $O_m$, shows whether the two applied inputs belong to the same class.

The structure inside each block used in the previous network (see Fig. 2) is shown in Fig. 3. The structure is inspired by ResNet (deep Residual Network) [27] blocks. ResNet is a deep CNN that uses shortcut connections in its blocks, which are called residual blocks. The shortcut connections help the gradient flow directly to the bottom layers. ResNet [27] is a well-known deep structure for CNN and it has achieved state-of-the-art results in image processing tasks. Note that the network shown in Fig. 2, which has been designed to be trained on the increased number of training instances in $X^n$ by accepting two inputs, is called the base network. The base network does not have the proposed intermediate targets which are described in the next section. The structure of the residual blocks which are used in this paper is shown in Fig. 3. The main branch is composed of three pairs of a 1-dimensional convolutional

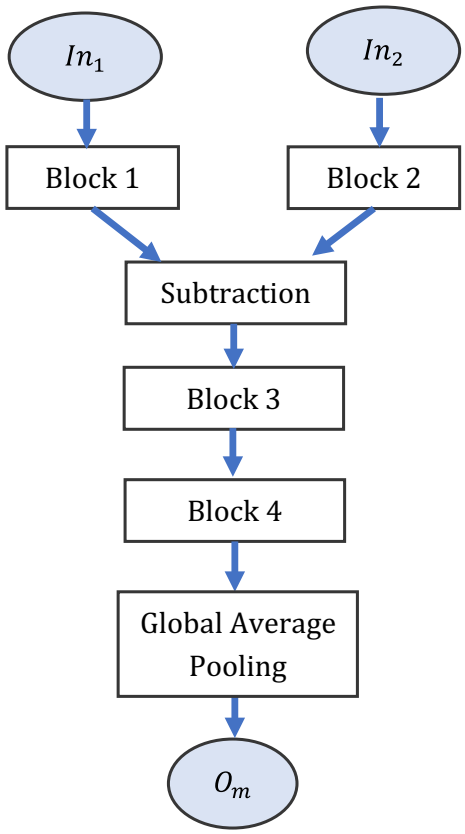


**Fig. 2** The structure of the base deep neural network which is trained on the new training dataset which uses the synthetically increased training samples. The structure of Blocks 1–4 is shown in Fig. 3. The base network does not have intermediate targets

layer (1D-Conv) coupled with batch normalization. The outputs of the first two pairs are passed through ReLU (Rectified Linear Unit) activation function, as shown in Fig. 3. The shortcut connection on the right side of Fig. 3 comprises a pair of 1D-Conv and a batch normalization layer. The outputs of the main branch and the shortcut connection are added. The results are passed through an ReLU activation function to make the output of the block.

### CNN-TS: Proposed Intermediate Target Concept for TSC Using CNN

Convolutional neural networks (CNNs) are end-to-end learning machines. During the learning process on a usual CNN, an input is applied to the first end, and a label at the other end is used to calculate loss and to tune the learning parameters. Usually, in CNN, inherent intermediate representation is generated without control and observation. However, human visual systems work based on perceptual organization. The process of extraction of low-level features in the intermediate level of the vision system has been referred by different names, such as perceptual organization, or feature grouping [66]. Determining how emerging low-level features in the intermediate levels of vision systems leading to perceptual organization remains a challenging problem in vision research. Perceptual organization is not generated randomly and it follows some rules [43]. In this work, applying controls on the intermediate features in a CNN can improve the processing ability of the CNN while making it similar to its natural counterpart. In this work, the proposed method is used for TS processing.

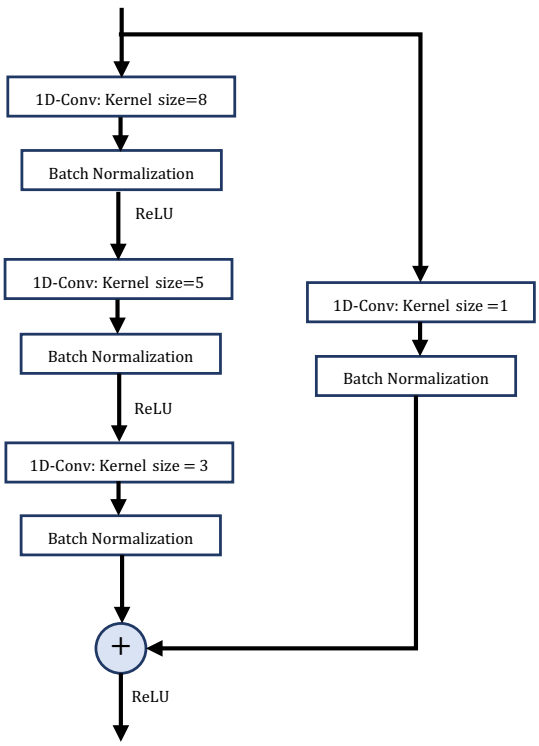


**Fig. 3** The structure of each block used in this research

In this paper, in addition to the method to increase the number of training instances described in "Time Series Classification".A, intermediate targets are constructed to improve the performance of CNN for TSC. The intermediate targets are used to train hidden layers of the CNN. The original CNN without intermediate targets shown in Fig. 2 is used as a base network, while the concept of "intermediate targets" is used to design a novel CNN structure for classification of TS, which is called CNN-TS.

The proposed network has two intermediate outputs, which are shown by $O_1$ and $O_2$ in Fig. 4. The two intermediate outputs, i.e., $O_1$ and $O_2$, are used to guide the training of the layers in Block 1 and Block 2, respectively. The labels of the training samples that are applied to $\text{In}_1$ are used for $O_1$. Therefore, the features generated in the output of Block1 are controlled by the label of $\text{In}_1$, and they contain information about the label of $\text{In}_1$.

On the other hand, the labels of the input samples which are applied to $\text{In}_2$ are used for $O_2$. Therefore, the features generated at the output of Block 2 are affected by the label of $\text{In}_2$, and generate features that contain information about label of $\text{In}_2$. The output features of Block 1 and Block 2 that reflect the labels of the two samples applied to the two inputs are subtracted. The subtracted features which reflect the distance of the two applied inputs are processed by Block 3 and Block 4 and Global Average Pooling to generate the main output $O_m$, and then, the main output is trained to evaluate whether the two inputs are from the same class or not.

When using deep neural networks for TSC where there are a high number of layers in the network, the error back propagated from the final output of the network should travel through a high number of layers to reach the input layer, and this could vanish the propagated error. Consequently, the training of the layers far from the outputs was not effective as it is expected. Using the intermediate outputs helps the learning algorithm to control the errors for the intermediate layers and creates more accurate backpropagated errors to train the network.

### Training the Proposed CNN-TS with Class-Related Coefficients

In the proposed CNN-TS, the inputs are first applied to their corresponding CNN layer. The CNN layer maps an input to a feature map with shared weights called a kernel, i.e., $\boldsymbol{W}$. In the $l$th layer, there are a number of feature maps, and (4) calculates the output of the $i$th feature map in the $l$th layer, $\boldsymbol{y}_i^l$

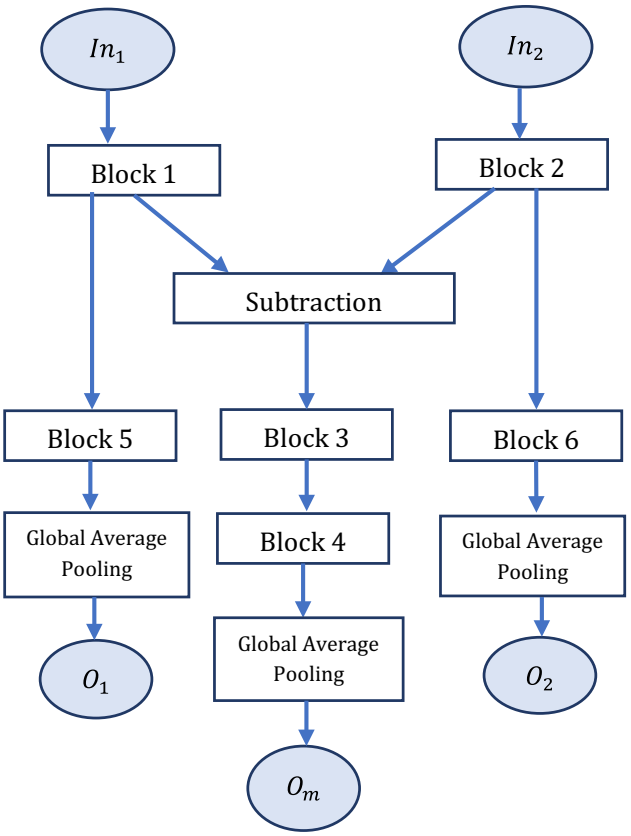


**Fig. 4** The proposed method with intermediate targets. The labels related to the two inputs applied to $In_1$ and $In_2$ are used as labels for the two intermediate targets, $O_1$ and $O_2$, respectively. The main target shows if the two inputs are from the same class or from different classes

$$\boldsymbol{y}_i^l = \sum_j (\boldsymbol{w}_{i,j}^l * \boldsymbol{y}_j^{l-1} + \boldsymbol{b}_i^l), \tag{4}$$

where $\boldsymbol{w}_{i,j}^l$ is the convolutional kernel used to map the $j$th feature map in the $(l-1)$th layer to the $i$th feature map in the next layer (the $l$th layer), $\boldsymbol{b}_i^l$ is the bias related to the $i$th feature map in the $l$th layer. The '*' is the convolutional operator sign. As shown in Fig. 3, there is a batch normalization layer after each convolution layer. The batch normalization layer normalizes the output of its previous layer to maintain the mean and standard deviation (Std.) close to 0 and 1, respectively.

After a batch normalization, ReLU function is used to generate the activation for the next layer (see Fig. 3). The activation is passed through different 1D-Conv, Batch normalization layers, and ReLU functions before reaching the three outputs of the proposed network (see Fig. 4). Figure 4 shows that before each output layer, there is a Global Average Pooling (GAP) layer [23]. In the GAP layer, the average of each feature map is calculated to represent the feature map to reduce the number of features. After this operation, the number of outputs returned by the GAP layer is reduced to the number of feature maps in the previous layer.

Three logistic regression models are placed on the top of the previous layers to construct three categorical outputs (see Fig. 4). A SoftMax function is used for the $k$th output as shown in (5)

$$\boldsymbol{O}^k = \text{softmax}\left(\boldsymbol{W}^k \left(\boldsymbol{G}^k\right)^T + b^k\right), \tag{5}$$

where $\boldsymbol{O}^k$ is the output vector of the $k$th output of the proposed network, and $k \in \{1, 2, m\}$ as the network has three outputs (see Fig. 4). The number of elements in $\boldsymbol{O}^k$ is equal to the number of classes for the $k$th output. For instance, the main output has two classes, and $\boldsymbol{O}^{k=m}$ has two elements. $\boldsymbol{G}^K$ is the output of the GAP layer before the $k$th output layer, $\boldsymbol{W}^k$ is the weight matrix that connects the output of the previous corresponding GAP layer to the $k$th output layer, and $b^k$ is the bias related to the $k$th output layer.

Adam (A Method for Stochastic Optimization [39]) for back-propagation learning is used to train the proposed CNN. Categorical cross-entropy is used to calculate loss function to train the network. Three loss functions corresponding to the three outputs of the proposed method are used to generate the total value of the loss of the network, i.e., $L$

$$L = \sum_{k \in \{1,2,m\}} \alpha_k L_k, \tag{6}$$

where $\alpha_k$ is a coefficient that weights the effect of the loss related to the $k$th output on the total value of the loss. As $\boldsymbol{O}^{k=m}$ is the main output, $\alpha_{k=m} = 1$ and the coefficients related to the two intermediate targets are set to 0.5, i.e., $\alpha_1 = \alpha_2 = 0.5$, which are half the value for the main output. $L_k$ is categorical cross-entropy loss related to the $k$th output.

The increased number of instances generated from the original data results in an imbalanced dataset, in $\boldsymbol{O}^m$. Additionally, real-world data are more likely to be imbalanced. Accordingly, class-related coefficients are used in the loss function of each output to improve the ability of the proposed method to process imbalanced data. Equation (7) shows the loss function of the $k$th output for the $i$th input instance, i.e., $L_k^i$, that includes the class-related coefficient

$$L_k^i = -C_k^i \sum_{c=1}^{N_k} t_k^{c,i} \log\left(z_k^{c,i}\right), \tag{7}$$

where $N_k$ is the number of classes for the $k$th output, $t_k^{c,i}$ is the $c$th element of the label vector corresponding to the $k$th output for the $i$th input instance, $z_k^{c,i}$ is the $c$th element of the predicted output vector related to the $k$th output for the $i$th input instance, and $C_k^i$ is a class-related coefficient related to the class of the $i$th input instance in the $k$th output. $L_k$

for each batch of data is calculated by summing $L_k^i$ over the number of samples in the batch of training instances.

The class-related coefficient has a high value when there are a low number of training samples in its corresponding class. The high value for a minority class causes a high loss value generated for an error related to the minority class, and consequently, the proposed method puts more attention on the class with a low number of instances. The class coefficient, $C_k^i$, is calculated using (8)

$$C_k^i = \frac{T_k^i}{T}, \tag{8}$$

where $T$ is the total number of training instances, and $T_k^i$ is the number of training instances out of the total number of training instances that have the same class as the $i$th input instance for the $k$th output. All the samples that belong to the same class according to the label for the $k$th output have the same class-related coefficient.

Note that selecting pairs of inputs from different classes to find if they are or they are not from the same class could result in an imbalanced classification task. Suppose that there are $N_c$ classes in the original dataset with equal numbers of training samples in different classes, i.e., $t_1 = \cdots = t_{N_c} = t$ where $t_i$ is the number of training samples in the $i$th class, and the total number of training samples in the original training set is $T = t_1 + \cdots + t_{N_c}$. In this case, the total number of training instances for the generated data corresponding to the main output is $T_t = T \times T = T^2$. The number of pairs of training samples that are from the same class can be calculated using (9)

$$T_s = N_c(t \times t) = N_c t^2, \tag{9}$$

where $T_s$ is the number of pairs of inputs that are from the same class. On the other hand, the number of pairs of inputs that are from different classes, i.e., $T_d$, can be calculated using (10)

$$T_d = N_c(t \times (T - t)) = N_c(tT - t^2) = N_c(t(N_c t) - t^2) = (N_c - 1)N_c t^2. \tag{10}$$

The two different values for the number of instances in the two classes related to the main output, i.e., $T_s$ and $T_d$, imply that the resulted classification task is an imbalanced classification problem when $N_c$>2. When the number of classes in the original data is increased, i.e., $N_c \gg 2$, the level of imbalance will be increased. The proposed method used (7) to overcome the imbalance in the generated data.

### Classifying a Test Input Based on the Main Output

During testing, a similar data structure described by (1) and (2) for training will be used. A test sample, $x_t^t$, is applying to the first input, i.e., $\text{In}_i$, and the representative subset of training samples that are selected to be applied to the second input for training the network is used. While a test sample is applied to the first input, each of the training samples in the representative subset is applied to the second input and the main output of the network predicts if the testing sample, $x_t^t$, is from the class of a sample from the representative subset. Whenever it is predicted that an applied sample (from representative subset of the training data) has the same class as the testing sample, the class of the test sample will be predicted based on the class of the sample from the representative subset of training data.

A subset of representative samples from the training dataset was selected using an under-sampling called NearMiss method [31, 81]. Two samples from each class were selected using the method. The number of samples in the representative subset is set to be small, i.e., 2 samples from each class, to prevent an intense increase in the number of input pairs. To obtain the results for the proposed method in Tables 2, 3, and 4, each testing sample is compared with the samples in the representative subset. Then, using the final output of the proposed method, it is predicted that the applied test sample has the same class label as one of the samples in the subset. If the applied test sample is predicted to have a similar class with more than one sample from the subset, it is assigned based on the vote appointed by the samples from the subset that is predicted to have the same class as the applied test input.

During testing, a test input is applied to $\text{In}_1$ (see Fig. 4), and $\overline{N}$ training samples from the original training dataset, i.e., $X$, are applied to $\text{In}_2$ one by one, and the network predicts whether the test data are from the same class as each training sample which is applied to $\text{In}_2$. Therefore, the number of predicted main outputs, $O_m$, for a testing input is equal to the number of the training data applied to $\text{In}_2$, i.e., $\overline{N}$. The training samples out of the $\overline{N}$ samples applied to $\text{In}_2$ that are predicted to have the same class as the testing sample are considered to decide about the label of the testing input. Suppose that $N_s$ training samples out of the $\overline{N}$ samples are predicted to have the same class as the testing input. The $N_s$ training samples may belong to different classes because of the prediction error. The testing input is assigned to a class according to the maximum voting over the label of the $N_s$ training samples.

## Results

The experimental results are presented in this section. First, the datasets, which are used in the experiments, are introduced. Then, the proposed method is compared with the base method, as shown in Fig. 2. In the subsection, the effect of the intermediate targets is investigated. In the third part of this section, the proposed method is compared to other state-of-the-art

methods. Finally, the effect of the number of training samples is investigated.

## Dataset

The experiments in this section are run on TS datasets obtained from the UCR Time series Classification Archive [14]. The characteristics of the datasets, which are used in the following experiments, are provided in Table 1. The first column of the table shows the type of TS. The name of each dataset is shown in the second column. Each dataset has a training set and a testing set, and the number of training and testing samples is shown in columns four and five, respectively. The number of classes in each dataset is shown in the fifth column. The length of TS in each dataset is shown in the last column of Table 1.

## Comparison of CNN-TS with the Base Deep Neural Network

Table 2 compares the accuracy of the final proposed method (Fig. 4) with the base network that does not have the intermediate output (Fig. 2). Note that the base method is essentially the standard approach, which is trained by a large set of training instances generated by the proposed approach, i.e., both methods are trained with the same number of training instances. In the following experiments, down-sampling is used to select 2 samples, i.e., $M = 2$, from each class of a training dataset to construct new instances for the proposed network. For instance, Table 1 shows that the 'InsectWingbeatSound' [14] dataset contains 220 training samples. $220^2 = 48,400$ different pairs of samples can be selected from the training dataset to be applied to the two inputs of the proposed network. However, if the number of instances for the second input is restricted to 22, the number of samples in the new dataset (composed of two inputs) is $220 \times 22 = 4840$ which is 10 times smaller than the previous situation—hence, training can be performed by 10 times less computation cost. In this section, the effect of the intermediate targets is investigated on different TS datasets, and the results are shown in Table 2. The results show that in most datasets (17 out of 20), the proposed method with intermediate targets has higher accuracy compared to when the intermediate targets are removed from the proposed structure. The proposed CNN-TS method achieved an average accuracy of 80.6%, outperforming the base method which achieved an average accuracy of 77.1%. The bold numbers in Table 2 show the largest values of Accuracy (A), precision (P), and recall (R) in each row. As shown in Table 2, intermediate targets have increased the classification accuracy by 22.40% on the 'InlineSkate' dataset, and from 91.67 to 97.37% on the 'ToeSegmentation1' dataset.

The learning parameters of the hidden layers in Block 1 are affected by the $O_1$ and $O_m$ during training. The error generated in $O_1$ is backpropagated to Block 1 through Block 5. The intermediate output $O_1$ controls the output of Block1 (based on the labels provided for $O_1$), and therefore, the intermediate output $O_1$ keeps the features in Block 1 that contain information about the label of $In_1$. Similarly, the output $O_2$ controls the features

**Table 1** Characteristic of Time series Datasets, i.e., the number of samples in the training and test data, the number of classes, and the length of time series in each dataset

| Type | Dataset | # Train | # Test | # Class | Length |
|---|---|---|---|---|---|
| TS Images[a] | ArrowHead | 36 | 175 | 3 | 251 |
| Food spectrographs | Beef | 30 | 30 | 5 | 470 |
| Sensor | Car | 60 | 60 | 4 | 577 |
| ECG | ECG200 | 100 | 100 | 2 | 96 |
| Food spectrographs | Ham | 109 | 105 | 2 | 431 |
| TS Images | Herring | 64 | 64 | 2 | 512 |
| Sensor | ItalyPowerDemand | 67 | 1029 | 2 | 24 |
| Sensor | Lightning2 | 60 | 61 | 2 | 637 |
| Sensor | Lightning7 | 70 | 73 | 7 | 319 |
| Sensor | MoteStrain | 20 | 1252 | 2 | 84 |
| Motion | ToeSegmentation1 | 40 | 228 | 2 | 277 |
| Sensor | Earthquakes | 139 | 322 | 2 | 512 |
| Motion | Haptics | 155 | 308 | 5 | 1092 |
| Motion | InlineSkate | 100 | 550 | 7 | 1882 |
| Sensor | InsectWingbeatSound | 220 | 1980 | 11 | 256 |
| Simulated | Mallat | 55 | 2345 | 8 | 1024 |
| Sensor | CinC_ECG_torso | 40 | 1380 | 4 | 1639 |
| TS Images | SwedishLeaf | 500 | 625 | 15 | 128 |
| TS Images | FaceAll | 560 | 1690 | 14 | 131 |
| Sensor | ChlorineConcentration | 467 | 3840 | 3 | 166 |

[a] http://www.timeseriesclassification.com/description.php?Dataset=ArrowHead

**Table 2** The effect of the intermediate targets on the proposed method

| Dataset | Proposed CNN-TS[a] | | | Base DNN[b] | | | CNN-TS vs. Base DNN difference |
|---|---|---|---|---|---|---|---|
| | *A* | *P* | *R* | *A* | *P* | *R* | dA |
| ArrowHead | **79.4** | **0.827** | **0.827** | 76.0 | 0.820 | 0.810 | 3.4 |
| Beef | **90.0** | **0.920** | **0.920** | 87.0 | 0.910 | 0.900 | 3 |
| Car | 88.3 | 0.930 | **0.930** | **90.0** | **0.960** | 0.857 | − 1.7 |
| ECG200 | **90.0** | 0.880 | **0.880** | 85.0 | **0.890** | 0.870 | 5 |
| Ham | **77.1** | **0.793** | **0.793** | 73.3 | 0.760 | 0.760 | 3.8 |
| Herring | **53.1** | 0.452 | 0.452 | 46.9 | **0.470** | **0.470** | 6.2 |
| ItalyPowerDemand | 96.4 | **0.967** | **0.967** | **96.0** | 0.960 | 0.960 | 0.4 |
| Lightning2 | **77.1** | **0.802** | **0.805** | 75.4 | 0.733 | 0.751 | 1.7 |
| Lightning7 | **82.2** | **0.872** | **0.858** | 76.7 | 0.870 | 0.780 | 5.5 |
| MoteStrain | **89.4** | **0.924** | **0.916** | 83.4 | 0.903 | 0.897 | 6 |
| ToeSegmentation1 | **97.4** | **0.957** | **0.955** | 91.7 | 0.800 | 0.770 | 5.7 |
| Earthquakes | **82.0** | 0.564 | **0.525** | 81.7 | **0.910** | 0.510 | 0.3 |
| Haptics | **47.7** | 0.428 | 0.397 | 45.8 | **0.470** | **0.460** | 1.9 |
| InlineSkate | **53.1** | 0.216 | 0.223 | 30.7 | **0.330** | **0.320** | 22.4 |
| InsectWingbeatSound | **62.5** | **0.641** | **0.641** | 60.7 | 0.640 | 0.640 | 1.8 |
| Mallat | **97.3** | **0.968** | **0.968** | 96.9 | 0.960 | 0.960 | 0.4 |
| CinC_ECG_torso | 93.3 | 0.940 | 0.933 | **95.4** | **0.960** | **0.960** | − 2.1 |
| SwedishLeaf | **95.5** | **0.968** | **0.968** | 94.4 | 0.950 | 0.950 | 1.1 |
| FaceAll | **82.1** | **0.963** | **0.957** | 79.9 | 0.930 | 0.930 | 2.2 |
| ChlorineConcentration | **78.4** | 0.395 | 0.541 | 74.2 | **0.850** | **0.770** | 4.2 |
| Average | **80.6** | 0.770 | **0.773** | 77.1 | **0.804** | 0.767 | 3.4 |
| Std. | 15.3 | 0.234 | 0.227 | 18.3 | 0.187 | 0.191 | 3 |
| Win | 17 | 12 | 15 | 3 | 8 | 5 | |

Accuracy (*A*), precision (*P*), and recall (*R*) for the proposed CNN-TS, and the Base DNN were reported

[a]With intermediate targets

[b]Without intermediate targets

generated by 'Block 2' to generate features that have information about the label of the second input. This strategy helps the network make more accurate decisions on whether two inputs belong to the same class.

## Comparison of CNN-TS with Other Methods

In this section, the proposed method, CNN-TS, is compared with different methods. First, three classical machine-learning methods are trained on different time series data sets. The three methods are a linear Support Vector Machine (SVM) with a linear kernel, SVM with the Radial Basis Function (RBF) kernel, and a Random Forest (RF) with 50 estimators. Raw time series data have high dimensionality and cannot be directly fitted using classical machine-learning methods. Time-domain, frequency-domain, and time–frequency-domain features which are employed in [83] are extracted from each time series, and the three classical machine-learning methods are trained by the extracted features. Root mean square (RMS), Variance, the Maximum value, Skewness, Kurtosis, and Peak-to-Peak difference are six features in time domain that are utilized. Spectral Skewness, Kurtosis and power, which are calculated using the Fast Fourier transform (FFT), are the three frequency-domain features that are used in this research. Wavelet Energy is also used to extract a time–frequency-domain feature. The accuracies of these methods relative to the proposed method, when applied to various datasets obtained from the USR database, are shown in Table 3. The last row, i.e., 'Wins' row, in Table 3 shows the number of times each method reached the highest accuracy among the other methods. The proposed method has achieved the highest accuracy on 18 datasets, followed by the SVM with linear Kernel method, which reached the highest accuracy on 2 datasets. The 'Average' row in Table 3 shows the mean value of accuracy achieved by each method across all datasets. The results show that, on average, the proposed CNN-TS achieved the highest accuracy, i.e., 80.6%, followed by the RF method which achieved an average accuracy of 59.5%, i.e., on average, the proposed CNN-TS approach achieves 21.1% higher accuracy compared the best results achieved among the three methods. Note that the proposed method was directly applied to raw time series data with a high

**Table 3** Comparison of the proposed CNN-TS method with three classical machine-learning methods

| Dataset | Proposed method | | | SVM (linear) | | | SVM (RBF) | | | RF | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* |
| ArrowHead | **79.4** | **0.827** | **0.827** | 53.1 | 54.9 | 51.9 | 54.3 | 58.0 | 52.9 | 64.0 | 65.3 | 63.9 |
| Beef | **90.0** | **0.920** | **0.920** | 40.0 | 44.8 | 40.0 | 36.7 | 34.3 | 36.7 | 30.0 | 9.3 | 0.0 |
| Car | **88.3** | **0.930** | **0.930** | 50.0 | 54.9 | 49.2 | 21.7 | 5.4 | 5.0 | 68.3 | 71.6 | 8.7 |
| ECG200 | **90.0** | **0.880** | **0.880** | 76.0 | 75.7 | 70.3 | 70.0 | 67.7 | 63.2 | 80.0 | 80.3 | 75.3 |
| Ham | **77.1** | **0.793** | **0.793** | 61.0 | 63.7 | 60.2 | 54.3 | 54.9 | 53.5 | 58.1 | 58.1 | 57.9 |
| Herring | 53.1 | 0.452 | 0.452 | 59.4 | 29.7 | 50.0 | 59.4 | 29.7 | 50.0 | **64.1** | **62.7** | **59.4** |
| ItalyPowerDemand | **96.4** | **0.967** | **0.967** | 91.3 | 91.5 | 91.3 | 49.9 | 24.9 | 50.0 | 90.4 | 90.5 | 90.4 |
| Lightning2 | 77.1 | **0.802** | **0.805** | **78.7** | 79.5 | 77.9 | 57.4 | 59.6 | 54.4 | 75.4 | 77.8 | 74.0 |
| Lightning7 | **82.2** | **0.872** | **0.858** | 56.2 | 59.4 | 55.9 | 38.4 | 38.9 | 32.4 | 54.8 | 51.4 | 51.3 |
| MoteStrain | **89.4** | **0.924** | **0.916** | 81.0 | 81.4 | 80.4 | 74.8 | 75.6 | 73.8 | 78.0 | 77.9 | 77.8 |
| ToeSegmentation1 | **97.4** | **0.957** | **0.955** | 72.4 | 73.2 | 71.8 | 54.0 | 53.7 | 53.7 | 64.5 | 64.5 | 64.5 |
| Earthquakes | **82.0** | 0.564 | 0.525 | **82.0** | 41.0 | 50.0 | **82.0** | 41.0 | 50.0 | 78.3 | **57.4** | **54.5** |
| Haptics | **47.7** | 0.428 | 0.397 | 41.2 | **43.8** | **41.2** | 29.9 | 24.1 | 28.8 | 37.0 | 37.9 | 36.6 |
| InlineSkate | **53.1** | 0.216 | 0.223 | 22.6 | **35.6** | 21.1 | 18.2 | 8.8 | 16.3 | 29.5 | 28.5 | **28.8** |
| InsectWingbeatSound | **62.5** | **0.641** | **0.641** | 17.2 | 18.4 | 17.2 | 13.7 | 7.4 | 13.7 | 13.9 | 3.8 | 3.9 |
| Mallat | **97.3** | **0.968** | **0.968** | 68.2 | 69.0 | 68.3 | 12.3 | 1.5 | 12.5 | 74.8 | 76.5 | 74.9 |
| CinC_ECG_torso | **93.3** | **0.940** | **0.933** | 52.4 | 55.3 | 52.4 | 42.8 | 32.0 | 42.9 | 55.9 | 60.5 | 55.9 |
| SwedishLeaf | **95.5** | **0.968** | **0.968** | 67.5 | 67.1 | 67.1 | 22.6 | 13.8 | 24.0 | 72.2 | 71.7 | 72.2 |
| FaceAll | **82.1** | **0.963** | **0.957** | 45.8 | 45.7 | 52.1 | 25.9 | 32.5 | 30.3 | 46.9 | 47.5 | 52.7 |
| ChlorineConcentration | **78.4** | 0.395 | **0.541** | 54.5 | 51.3 | 35.2 | 54.9 | **51.4** | 35.7 | 53.3 | 45.7 | 42.4 |
| Average | 80.6 | 0.770 | 0.773 | 58.5 | 56.8 | 55.2 | 43.6 | 35.8 | 40.0 | 59.5 | 58.4 | 57.2 |
| Std. | 15.33 | 0.234 | 0.227 | 18.95 | 18.2 | 18.5 | 20.1 | 21.3 | 16.6 | 19.3 | 19.4 | 18.8 |
| Wins | 18 | 15 | 16 | 2 | 2 | 1 | 1 | 1 | 0 | 1 | 2 | 3 |

dimension; however, the classical machine-learning methods used the features extracted from the specified feature extraction methods (because the dimension of the raw time series data is high, such approaches cannot directly be applied to the raw data).

In the next experiments, the proposed method is compared with other methods that can directly be applied to raw data. Table 4 presents the results when applying the proposed method and other methods to TS benchmark datasets. In Table 4, 'MLP' stands for Multi-Layer Perceptron (MLP) which is a traditional form of DNN. The MLP for TSC proposed by Wang et al. [74] is used in the comparison. The MLP has 4 fully connected layers and a Softmax layer as the output layer of the network. 'FCN' stands for Fully Convolutional Neural Network (FCN) [50], and it is extended by Wang et al. [74] for TSC. FCNs do not have any pooling layers and contain 5 layers. 'ResNet' is a deep Residual Network (ResNet) for TSC proposed in [74]. The ResNet has 11 layers and it is the deepest architecture used in this paper. The original ResNet [27] has achieved state-of-the-art results in image processing, and it is a well-known method.

'Encoder' is the Encoder network [67] with 5 layers include 3 convolutional layers. Encoder is similar to FCNs [50]; however, it uses Parametric Rectified Linear Unit (PReLU) activation function where an additional parameter is added to enable learning of the slop of each function. Additionally, the Encoder network uses dropout regularization and max-pooling operation. MCNN is the Multi-scale Convolutional Neural Network (MCNN) [13] which comprises 4 layers. t-LeNet is the Time Le-Net [26] which has 4 layers. MCDCNN is a Multi-Channel Deep Convolutional Neural Network [84]. Time-CNN [17] has 3 layers, uses the mean-squared error loss function with a Fully Convolutional layer with sigmoid activation function in its final layer. Finally, 'TWIESN' is a Time Warping Invariant Echo State Network [71] with 3 layers.

Table 4 shows that the proposed method has reached the highest accuracy on 8 datasets, followed by the ResNet [74] which has reached the highest accuracy on 7 datasets. The results in Table 4 show that, on average, the proposed method has achieved the highest accuracy, i.e., 80.6%, followed by the ResNet method which achieved an average accuracy of 79.0%. Table 4 also compares the Precision, and Recall of the proposed method and ResNet.

The critical difference diagram (CDD) used in [17] was implemented to statistically compare all the classifiers over all the data sets. The code in https://github.com/hfawaz/cd-diagram was used to do the statistic test. First, Friedman test was performed to reject a null hypothesis, i.e., there are no differences between the accuracies to depict the CDD. After the rejection of the null hypothesis, Wilcoxon–Holm method was used to perfume a post hoc analysis. The result is reported in Fig. 5. The thick horizontal lines in

**Table 4** Comparison of the proposed CNN-TS method with the state-of-the-arts methods

| Dataset | Proposed method | | | MLP [74] | | | FCN [74] | | | ResNet [74] | | | Encoder [67] | | | MCNN [13] | | | t-LeNet [26] | | | MCDCNN [84] | | | Time-CNN [17] | | | TWIESN [71] | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* | *A* | *P* | *R* |
| ArrowHead | 79.4 | 0.827 | 0.827 | 77.8 | 0.673 | 0.658 | 84.3 | 0.836 | 0.828 | **84.5** | **0.854** | **0.858** | 80.4 | 0.794 | 0.798 | 33.9 | 0.100 | 0.333 | 30.3 | 0.101 | 0.333 | 68.5 | 0.701 | 0.608 | 72.3 | 0.730 | 0.733 | 65.9 | 0.621 | 0.618 |
| Beef | **90.0** | **0.920** | **0.920** | 72.0 | 0.703 | 0.700 | 69.7 | 0.683 | 0.667 | 75.3 | 0.776 | 0.767 | 64.3 | 0.717 | 0.633 | 20.0 | 0.054 | 0.250 | 20.0 | 0.040 | 0.200 | 56.3 | 0.04 | 0.200 | 76.3 | 0.805 | 0.733 | 53.7 | 0.453 | 0.433 |
| Car | 88.3 | 0.930 | **0.930** | 76.7 | 0.766 | 0.769 | 90.5 | **0.945** | 0.926 | **92.5** | 0.932 | 0.911 | 75.8 | 0.779 | 0.777 | 24.0 | 0.054 | 0.250 | 31.7 | 0.079 | 0.250 | 73.0 | 0.636 | 0.649 | 78.2 | 0.767 | 0.760 | 78.3 | 0.753 | 0.740 |
| ECG200 | 90.0 | 0.880 | 0.880 | 91.6 | 0.879 | 0.858 | 88.9 | 0.873 | 0.864 | 87.4 | 0.886 | 0.852 | **92.3** | **0.917** | **0.887** | 64.0 | 0.320 | 0.500 | 64.0 | 0.320 | 0.50 | 83.3 | 0.814 | 0.787 | 81.4 | 0.777 | 0.757 | 84.2 | 0.862 | 0.810 |
| Ham | **77.1** | **0.793** | **0.793** | 69.1 | 0.714 | 0.714 | 71.8 | 0.724 | 0.724 | 75.7 | 0.772 | 0.771 | 72.7 | 0.739 | 0.731 | 50.6 | 0.257 | 0.500 | 51.4 | 0.257 | 0.500 | 73.3 | 0.677 | 0.675 | 71.1 | 0.707 | 0.706 | 72.3 | 0.790 | 0.778 |
| Herring | 53.1 | 0.452 | 0.452 | 52.8 | 0.60 | 0.603 | 60.8 | **0.641** | **0.632** | **61.9** | 0.588 | 0.590 | 58.6 | 0.556 | 0.530 | 59.4 | 0.297 | 0.500 | 59.4 | 0.297 | 0.500 | 60.0 | 0.297 | 0.500 | 53.9 | 0.537 | 0.534 | 59.1 | 0.593 | 0.537 |
| ItalyPowerDemand | 96.4 | 0.967 | 0.967 | 95.4 | 0.962 | 0.961 | 96.1 | 0.960 | 0.960 | 96.3 | **0.968** | **0.968** | **96.5** | 0.968 | 0.968 | 50.0 | 0.251 | 0.500 | 49.9 | 0.249 | 0.500 | 95.5 | 0.948 | 0.948 | 95.5 | 0.953 | 0.952 | 88.0 | 0.924 | 0.923 |
| Lightning2 | **77.1** | **0.802** | **0.805** | 67.0 | 0.703 | 0.700 | 73.9 | 0.737 | 0.733 | 77.0 | 0.753 | 0.751 | 69.2 | 0.687 | 0.656 | 55.7 | 0.271 | 0.500 | 54.1 | 0.271 | 0.500 | 63.0 | 0.752 | 0.628 | 63.6 | 0.639 | 0.629 | 70.3 | 0.648 | 0.600 |
| Lightning7 | 82.2 | **0.872** | **0.858** | 63.0 | 0.581 | 0.608 | 82.7 | 0.813 | 0.796 | **84.5** | 0.856 | 0.813 | 62.5 | 0.615 | 0.602 | 31.0 | 0.037 | 0.143 | 26.0 | 0.037 | 0.143 | 53.4 | 0.567 | 0.580 | 65.1 | 0.687 | 0.695 | 66.4 | 0.805 | 0.682 |
| MoteStrain | 89.4 | 0.924 | 0.916 | 85.8 | 0.865 | 0.860 | **93.7** | 0.929 | 0.928 | 92.8 | **0.935** | **0.933** | 84.0 | 0.853 | 0.848 | 50.8 | 0.230 | 0.500 | 53.9 | 0.270 | 0.500 | 76.5 | 0.846 | 0.814 | 88.2 | 0.901 | 0.888 | 78.5 | 0.833 | 0.829 |
| ToeSegmentation1 | **97.4** | **0.957** | 0.955 | 58.3 | 0.596 | 0.596 | 96.1 | 0.969 | **0.969** | 96.3 | 0.956 | 0.957 | 65.9 | 0.666 | 0.664 | 50.5 | 0.263 | 0.500 | 52.6 | 0.263 | 0.500 | 49.0 | 0.563 | 0.561 | 59.5 | 0.614 | 0.614 | 86.5 | 0.905 | 0.902 |
| Earthquakes | **82.0** | 0.564 | 0.525 | 71.7 | 0.511 | 0.504 | 72.7 | **0.582** | **0.581** | 71.2 | 0.501 | 0.500 | 74.8 | 0.566 | 0.517 | 74.8 | 0.410 | 0.500 | 74.8 | 0.410 | 0.500 | 74.9 | 0.410 | 0.500 | 70.0 | 0.542 | 0.552 | 74.8 | 0.526 | 0.507 |
| Haptics | 47.7 | 0.428 | 0.397 | 43.3 | 0.421 | 0.436 | 48.0 | **0.540** | **0.490** | **51.9** | 0.530 | 0.482 | 42.7 | 0.418 | 0.427 | 20.9 | 0.038 | 0.200 | 20.8 | 0.042 | 0.200 | 40.4 | 0.373 | 0.396 | 36.6 | 0.374 | 0.381 | 40.4 | 0.461 | 0.429 |
| InlineSkate | **53.1** | 0.216 | 0.223 | 33.7 | 0.361 | 0.334 | 33.9 | 0.344 | **0.350** | 37.3 | 0.253 | 0.246 | 29.2 | 0.362 | 0.312 | 16.7 | 0.026 | 0.143 | 16.5 | 0.022 | 0.143 | 21.5 | 0.288 | 0.120 | 28.7 | 0.329 | 0.313 | 33.0 | **0.492** | 0.280 |
| InsectWingbeatSound | 62.5 | 0.641 | 0.641 | 60.7 | **0.655** | **0.645** | 39.3 | 0.403 | 0.399 | 50.7 | 0.514 | 0.512 | **63.3** | 0.649 | 0.642 | 15.8 | 0.008 | 0.091 | 9.1 | 0.008 | 0.091 | 58.3 | 0.593 | 0.608 | 58.3 | 0.585 | 0.587 | 43.7 | 0.360 | 0.326 |
| Mallat | **97.3** | 0.968 | 0.968 | 91.8 | 0.930 | 0.905 | 96.7 | **0.975** | **0.973** | 97.2 | 0.973 | 0.969 | 87.6 | 0.909 | 0.873 | 13.5 | 0.015 | 0.125 | 12.3 | 0.015 | 0.125 | 90.1 | 0.943 | 0.929 | 92.0 | 0.935 | 0.923 | 59.6 | 0.556 | 0.643 |
| CinC_ECG_torso | **93.3** | 0.940 | 0.933 | 84.0 | 0.864 | 0.832 | 82.4 | 0.828 | 0.827 | 82.6 | 0.788 | 0.789 | 91.1 | 0.913 | 0.897 | 38.1 | 0.062 | 0.250 | 25.0 | 0.063 | 0.250 | 73.6 | **0.952** | **0.953** | 74.5 | 0.695 | 0.609 | 30.0 | 0.286 | 0.290 |
| SwedishLeaf | 95.5 | 0.968 | 0.968 | 85.1 | 0.807 | 0.782 | **96.9** | **0.971** | **0.970** | 95.6 | 0.952 | 0.949 | 93.0 | 0.930 | 0.924 | 11.8 | 0.004 | 0.67 | 6.5 | 0.004 | 0.067 | 84.6 | 0.883 | 0.887 | 88.4 | 0.870 | 0.873 | 82.5 | 0.859 | 0.818 |
| FaceAll | 82.1 | **0.963** | 0.957 | 79.3 | 0.922 | **0.962** | 94.5 | 0.923 | 0.962 | **83.9** | 0.866 | 0.927 | 79.3 | 0.820 | 0.871 | 17.0 | 0.006 | 0.071 | 8.0 | 0.006 | 0.071 | 71.7 | 0.755 | 0.804 | 76.8 | 0.756 | 0.863 | 65.7 | 0.698 | 0.763 |
| ChlorineConcentration | 78.4 | 0.395 | 0.541 | 80.2 | 0.626 | 0.404 | 81.4 | 0.791 | 0.794 | **84.4** | **0.827** | **0.825** | 57.3 | 0.609 | 0.416 | 53.3 | 0.178 | 0.333 | 53.3 | 0.178 | 0.333 | 64.3 | 0.607 | 0.595 | 60.0 | 0.635 | 0.481 | 55.3 | 0.515 | 0.361 |
| Average | 80.6 | 0.770 | 0.773 | 72.0 | 0.707 | 0.692 | 77.7 | 0.773 | 0.769 | 79.0 | 0.774 | 0.769 | 72.0 | 0.723 | 0.699 | 37.6 | 0.144 | 0.343 | 36.0 | 0.147 | 0.310 | 66.6 | 0.632 | 0.637 | 69.5 | 0.692 | 0.679 | 64.41 | 0.647 | 0.614 |
| Std. | 15.3 | 0.234 | 0.227 | 16.4 | 0.169 | 0.181 | 19.2 | 0.191 | 0.195 | 16.9 | 0.197 | 0.201 | 17.3 | 0.171 | 0.190 | 19.5 | 0.131 | 0.180 | 21.2 | 0.132 | 0.174 | 17.4 | 0.250 | 0.229 | 17.2 | 0.168 | 0.176 | 17.461 | 0.190 | 0.208 |
| Wins | 8 | 6 | 5 | 0 | 1 | 2 | 2 | 6 | 7 | 7 | 4 | 4 | 3 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

Fig. 5 show different groups of classifiers whose accuracies are not significantly different.

The t-SNE visualization implemented in [11] is used to intuitively evaluate the effectiveness of the proposed method. The t-SNE visualization is applied to the $O_1$ and $O_m$ output of the proposed method to find the separability of the method. Additionally, t_SNE visualization is applied to the output of the ResNet. The t-SNE visualizations for different data sets are reported in Figs. 6, 7, and Appendix 1.

Next, the computation time of the proposed method is compared to ResNet, and the results are shown in Table 5. The number of training epochs and the required time duration for training both methods are shown in Table 5. The 'Improvement' column in Table 5 shows the difference between the time duration of ResNet and the proposed method, i.e., the time duration of ResNet is subtracted from the time duration of the proposed method. The last column 'Times of Improvement' in Table 5 shows how many times the time duration of ResNet is higher than the time duration of the proposed method. 'Times of Improvement' is obtained by dividing the time duration of ResNet by the time duration of the proposed method.

The results in Table 5 show that for 18 out of 20 datasets, the proposed method performs its learning in shorter time durations than ResNet. ResNet can be trained in a shorter time than the proposed method only for two datasets, i.e., 'SwedishLeaf' and 'FaceAll' datasets. Note that these two datasets, 'SwedishLeaf' and 'FaceAll', have 15 and 14 classes, respectively, and thus contain a relatively high number of classes compared to the other datasets (see Table 1). Therefore, there are $15 \times 2 = 30$ selections for the second input for 'SwedishLeaf' dataset. Note that two samples are selected from each class using the mentioned down-sampling described in "Time Series Classification".A to represent the class. As the 'SwedishLeaf' dataset contains 500 training samples, the new training dataset for the two inputs of the proposed network has $500 \times 30 = 15{,}000$ instances, which is a high number compared to the original training dataset that contains 500 samples. The high number of available selections for the second input (which occurs due to the high number of classes in the dataset) increases the number of new training instances, and it consequently increases the training computation time. Therefore, the proposed method has a high training duration for datasets with a high number of classes.

Using the mentioned down-sampling can reduce the number of training samples and consequently reduce the training time. However, when the number of classes is increased, the reduction in the number of training samples is limited; because an appropriate number of training samples from each class is required. The high number of classes prevents a reduction in the number of generated training instances.

One method to further reduce the computation time for the proposed method is to reduce the number of selections for the second input from 30 to 15. To reach this aim, and thus to reduce the number of selections for the second input, instead of selecting two samples from each class, a single training sample will be selected from each class—this consequently reduces the number of instances from 1500 to $500 \times 15 = 7500$ which is the half of the previous one. Therefore, the computation time can also be reduced up to half. However, reducing the number of training samples might impact the model's accuracy (see the "Investigation of the Effect of the Number of Input Samples").

For 18 out of 20 datasets in Table 5, the proposed method performed the learning task in a shorter duration compared to ResNet. For instance, the proposed method has a reduction of 24,425.7 s and 23,112.9 s in learning time duration for 'Mallat' and 'CinC_ECG_torso' datasets, respectively, while the proposed method can reach a higher accuracy than ResNet on the datasets. In the last columns of Table 5, the results show that the proposed method can reach up to 46.9 times faster processing time than ResNet. For instance, the training time duration of ResNet for 'ItalyPowerDemand' dataset is 2673.7 s which is 15.3 times higher than of the time duration required by the proposed method. The proposed method only requires 175.0 s to reach an accuracy higher than the ResNet accuracy.

## Investigation of the Effect of the Number of Input Samples

Deep CNNs usually have a high number of layers of neurons, and consequently, they have a high number of training parameters. The high number of training parameters needs a high number of training samples to train the neural network. The proposed method combines different training samples and generates a new training set with a high number of training samples, and it increases the ability of the proposed method to learn with a comparably low number of original training samples. In this section, the ability of the proposed method is investigated when the number of training samples is reduced. In this simulation, the

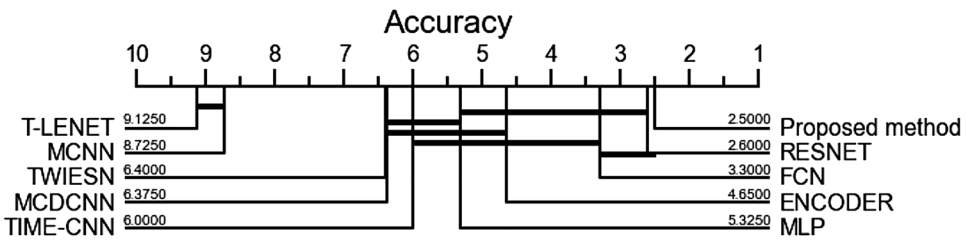


**Fig. 5** CDD for comparing the accuracy of the proposed method with other learning methods on different datasets. Each thick line shows a group of methods that do not have significant differences in their accuracies

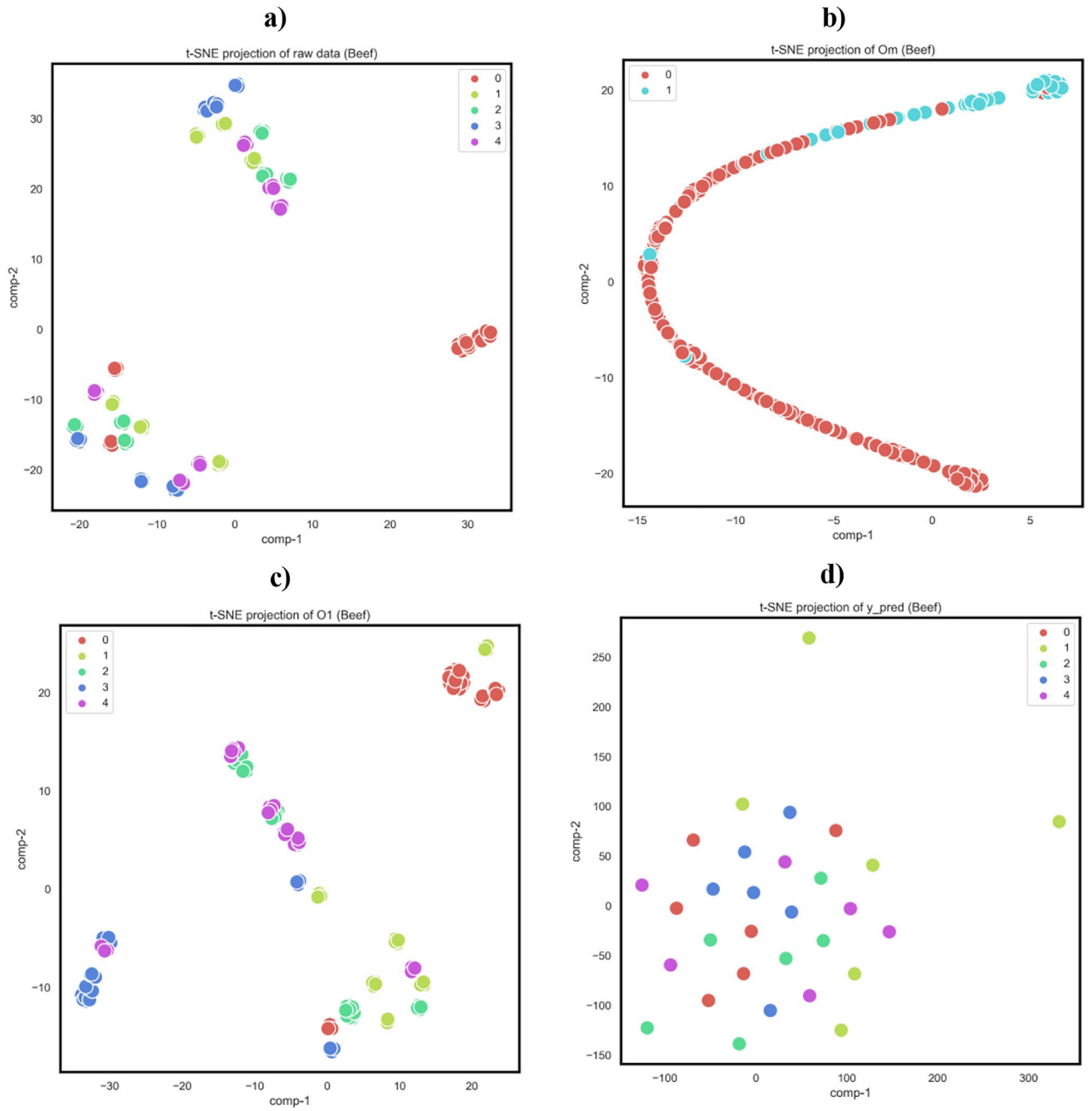


**Fig. 6** The t-SNE projection of **a** the raw data, **b** $O_m$ output of the proposed method, **c** $O_1$ output of the proposed method, and **d** the output of ResNet for 'Beef' data set

number of training samples in the 'CinC_ECG_torso' dataset from the UCR data archive is reduced gradually, and then, the accuracy of the proposed method is obtained for different reduced numbers of training samples, i.e., the network is trained by the remainder of the training set. The number of training samples is reduced by 2, 4, 6, and 8. The first class in the 'CinC_ECG_torso' dataset has a small number of training samples (5 training samples), so it is kept unchanged and the training samples from the second class which has a higher number of training samples are reduced. The accuracy of the proposed method on the reduced number of training samples is then compared to the base method. Figure 8a shows that the accuracy of the base method is reduced when the number of removed training samples is increased, i.e., the number of remaining training samples is reduced. However, in comparison, the proposed method is relatively stable in accuracy values related to the reduced numbers of training samples when it is compared to the base method. The proposed method has 12.68% higher accuracy compared to the base method for

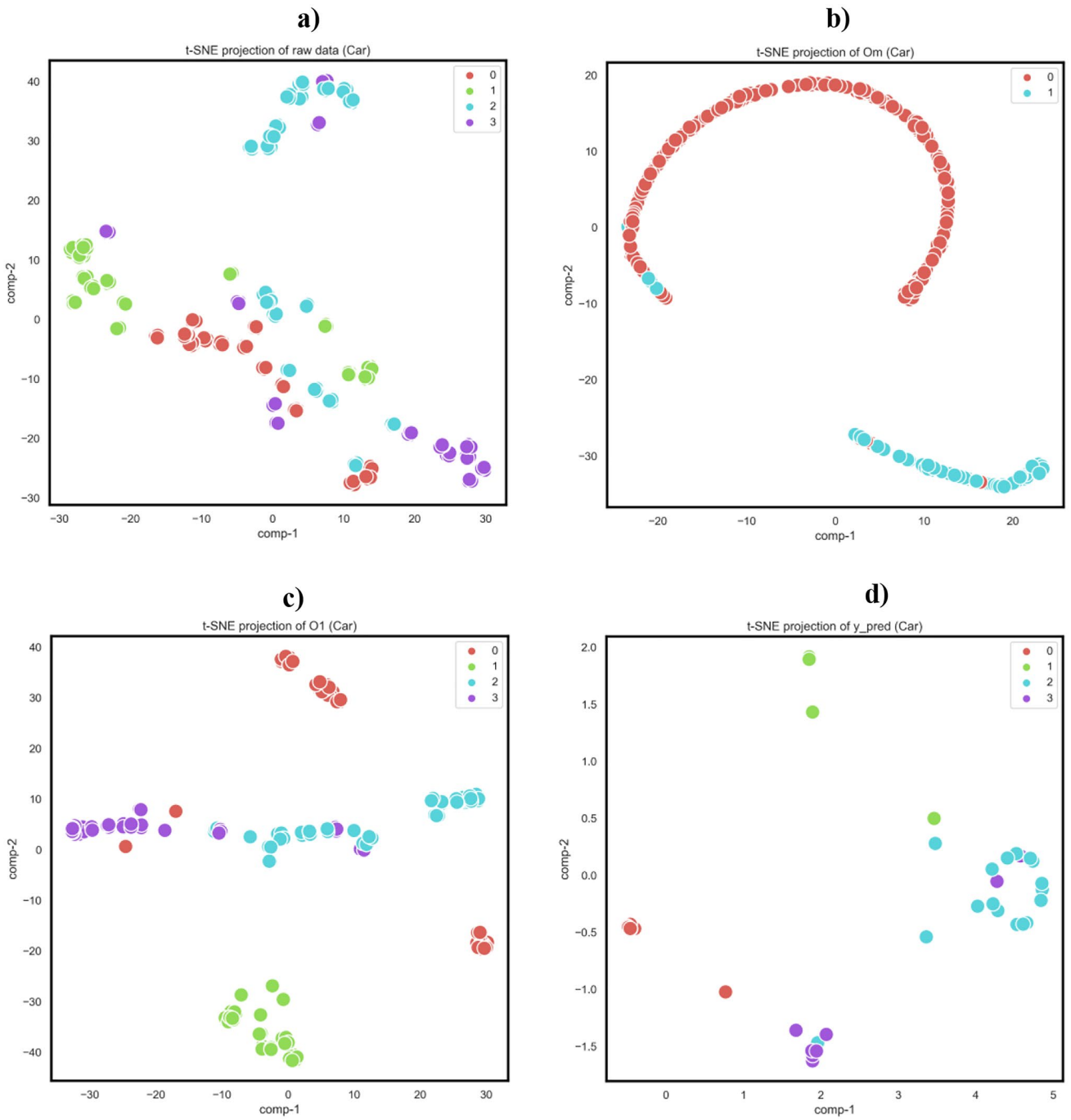


**Fig. 7** The t-SNE projection of **a** the raw data, **b** $O_m$ output of the proposed method, **c** $O_1$ output of the proposed method, and **d** the output of ResNet for 'Car' data set

the original dataset ('# Reduced Samples = 0') which contains all the training samples. The improvement of the proposed method is increased compared to the base method when the number of training samples is reduced. For instance, when the number of training samples is reduced from 40 to 32, the testing accuracy of the proposed method is 20.29% higher than the base method. The testing set in 'CinC_ECG_torso' contains 1380 samples. The proposed method can recognize 280 more testing instances correctly than the base method. The effect of the number of training samples is tested on other datasets and the results are shown in Fig. 8.

**Table 5** Comparison of computation time, i.e., duration

| Dataset | Proposed method | | ResNet | | Improvement (sec.) | Times of improvement |
|---|---|---|---|---|---|---|
| | # Epochs | Duration (sec.) | # Epochs | Duration (Sec.) | | |
| ArrowHead | 294 | 298.5 | 1492 | 1375.9 | 1077.4 | 4.6 |
| Beef | 277 | 304.1 | 1442 | 795.5 | 491.4 | 2.6 |
| Car | 298 | 709.6 | 1442 | 1005.5 | 295.9 | 1.4 |
| ECG200 | 291 | 185.4 | 1398 | 729.4 | 544.0 | 3.9 |
| Ham | 184 | 380.9 | 1405 | 1231.9 | 851.0 | 3.2 |
| Herring | 298 | 310.3 | 1323 | 1037.1 | 726.8 | 3.3 |
| ItalyPowerDemand | 231 | 175.0 | 1487 | 2673.7 | 2498.7 | 15.3 |
| Lighting2 | 243 | 325.1 | 1405 | 1739.4 | 1414.4 | 5.4 |
| Lighting7 | 226 | 536.4 | 1433 | 1297.5 | 761.0 | 2.4 |
| MoteStrain | 286 | 166.5 | 1442 | 7814.5 | 7648.0 | 46.9 |
| ToeSegmentation1 | 280 | 262.4 | 1429 | 1832.0 | 1569.6 | 7.0 |
| Earthquakes | 178 | 498.3 | 1159 | 3245.0 | 2746.6 | 6.5 |
| Haptics | 299 | 2457.3 | 1329 | 5530.3 | 3073.0 | 2.3 |
| InlineSkate | 299 | 3729.1 | 1358 | 11,050.0 | 7320.9 | 3.0 |
| InsectWingbeatSound | 296 | 2100.7 | 1498 | 7113.3 | 5012.6 | 3.4 |
| Mallat | 293 | 1305.9 | 1490 | 25,731.6 | 24,425.7 | 19.7 |
| CinC_ECG_torso | 296 | 829.5 | 1439 | 23,942.4 | 23,112.9 | 28.9 |
| SwedishLeaf | 287 | 3893.3 | 1440 | 2693.1 | − 1200.2 | 1.2 |
| FaceAll | 263 | 4165.2 | 1487 | 3975.3 | − 189.8 | 0.7 |
| ChlorineConcentration | 293 | 905.5 | 1451 | 6409.7 | 5504.1 | 1.0 |
| Average | 270.6 | 1177.0 | 1417.5 | 5561.2 | 4384.2 | 8.43 |
| Std. | 37.15 | 1305.8 | 77.2 | 6991.9 | 6880.0 | 11.20 |

## Discussion and Conclusion

In this paper, a method (described in the section "A Method for Synthetically Increasing the Number of Training Samples") is used to synthetically transform an initially labeled training dataset to improve the training process for time series classification. In particular, the method selects pairs of raw TS from the original training dataset. The higher number of available selections of two TS helps to increase the number of training instances. Therefore, the method increases the number of training instances by the power of two of the number of initial training samples. A deep CNN is a data-hungry method and it needs a high number of labeled training samples, and the proposed method makes more training data available for CNNs.

Then, a new CNN, called CNN-TS, is proposed to work with the increased number of training data. CNN-TS compares the two TS in each pair and predicts whether the two TS are from the same class. Moreover, the proposed CNN-TS benefits from intermediate targets which are set based on the new learning task. Two intermediate targets are set corresponding to the two TS which are applied as inputs of the proposed method. The intermediate targets supervise the intermediate features which are extracted from each input to increase the overall classification accuracy of the proposed method. The intermediate targets use the label of their corresponding input to train the network.

The proposed method can be considered as a deep distance-based TSC. In a classical distance-based TSC, a classification is performed based on the distance between a test sample and training samples. The main element in a classical distance-based TSC is its distance measurement method. Measuring the distance between two TS is not a straight-forward task, because the method should be invariant against translation in the TS or it should be insensitive to the speed of performing similar tasks. In the classical method usually, the distance between two samples is calculated, and then, an analysis is performed on the measured distances to decide in which class a sample belongs. However, the proposed method, CNN-TS, automatically evaluates the distance between two TS and performs the distance measurement and classification jointly in a network to increases its accuracy and to improve CNN's abilities. In fact, the intermediate targets in the proposed method control the features extracted in the intermediate layers to reflect information related to the labels of the applied inputs as the labels are available to the intermediate targets during training. Then, in the next layer, the intermediate features are subtracted to generate features that reflect the distance between the two inputs. In the following layers, the distance-related features are used to decide whether the two inputs are from the same class. The proposed method adjusts the learning parameters to learn the distance and classification in a CNN for TSC.

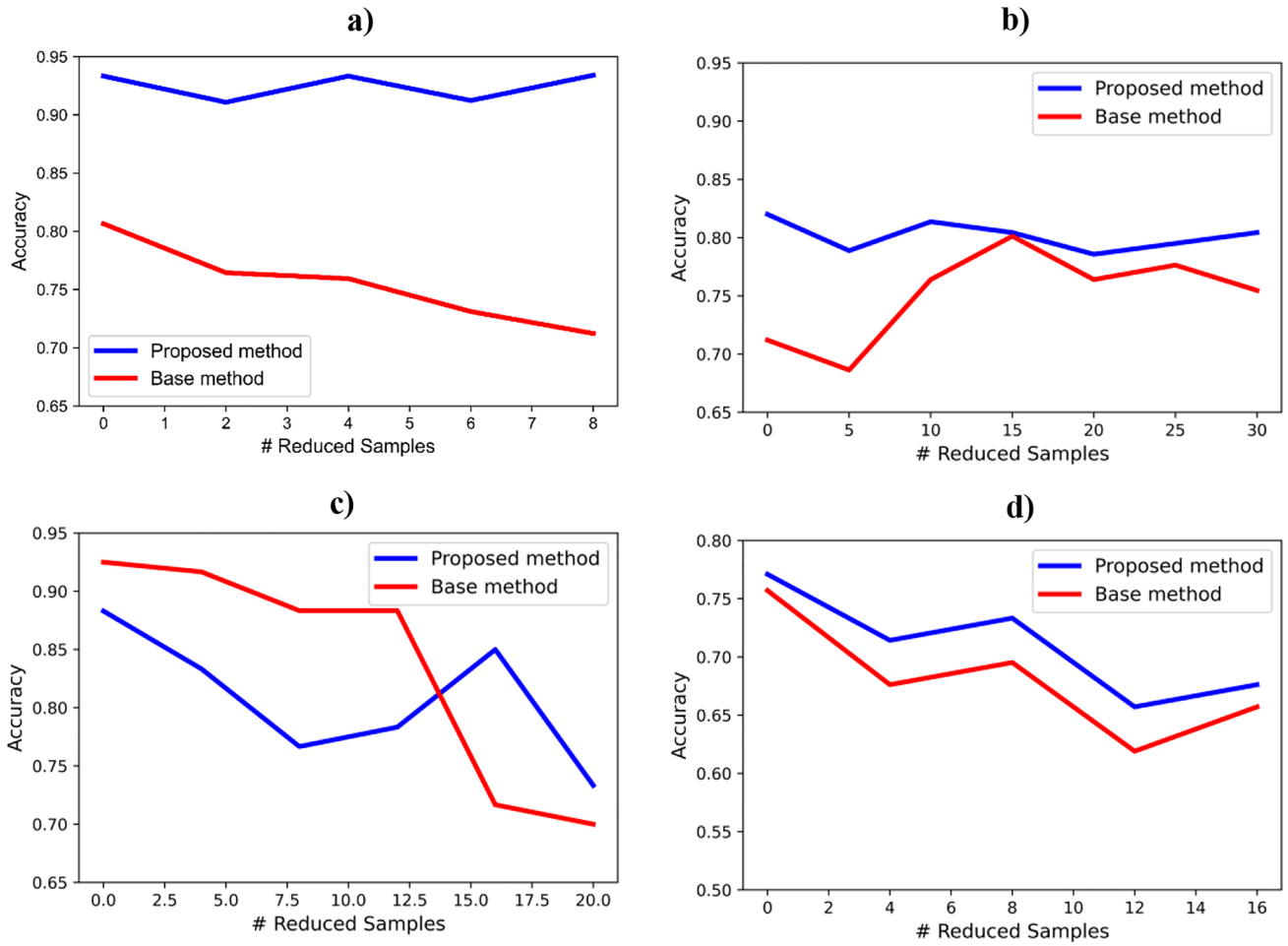


**Fig. 8** Comparison of the accuracy of the proposed method and the base method on **a** CinC_ECG_torso, **b** earthquakes, **c** car, and **d** Ham data sets when the number of training samples is reduced. Note that when '# Reduced Samples = 0', there is no reduction in the number of training samples, and thus, all the original training samples are used

Siamese neural networks have the ability to evaluate similarity between inputs [24]. A Siamese neural network learns to determine the probability of its applied pair of inputs belonging to the same class or different classes. The Siamese neural network proposed in [24] does not take into account the imbalance property which is generated as the result of selecting pairs of inputs from different classes. The severity of the imbalance will be increased when the number of classes is increased [see (9) and (10)]. Additionally, the method proposed in [24] does not use the extra knowledge that exists in the labels of each input in an applied pair of inputs to the network. However, the proposed method in this paper deals with the imbalance in the generated data using (7). Moreover, the proposed method in this paper has used the labels of inputs in each pair as intermediate targets to train hidden layers.

The proposed CNN-TS method is evaluated on different datasets obtained from the UCR time series classification archive. First, CNN-TS is compared to a base method, which is a similar CNN to the proposed method but without the intermediate targets. Experimental results show improvement in the accuracy of the proposed method compared to the base method on 17 out of 20 datasets. Additionally, the proposed method is compared with three classical machine-learning methods namely linear SVM, RBF SVM, and RF. The results show that, on average, the proposed method achieved 21.1% higher accuracy than that achieved by the other methods. The proposed method is also compared to other state-of-the-art methods, and the experiment results show that it has achieved higher accuracies on various different datasets compared to the best results achieved by the other methods. Moreover, CNN-TS achieved higher accuracies with a shorter training time duration, which is on average 8.43 times shorter than the time duration required for the method with the best accuracy among the other state-of-the-art methods, i.e., ResNet.

Although the classical distance-based methods are known to perform well in the traditional TSC, they have not been considered thoroughly in the literature of CNN methods for TSC

to date. Investigating the different aspects of distance-based TSC and reflecting them in CNN can be a new direction for future research. Experimental results have shown that intermediate targets can improve the performance of a CNN, because intermediate targets supervise the generation of features in the intermediate layers instead of allowing the features to generate without control. Thus, finding appropriate targets for intermediate layers in different applications of CNN can be another direction for future research.

## Appendix 1: The t-SNE Projection on 'ArrowHead', 'ECG200', and 'Ham' Datasets

See Figs. 9, 10 and 11.

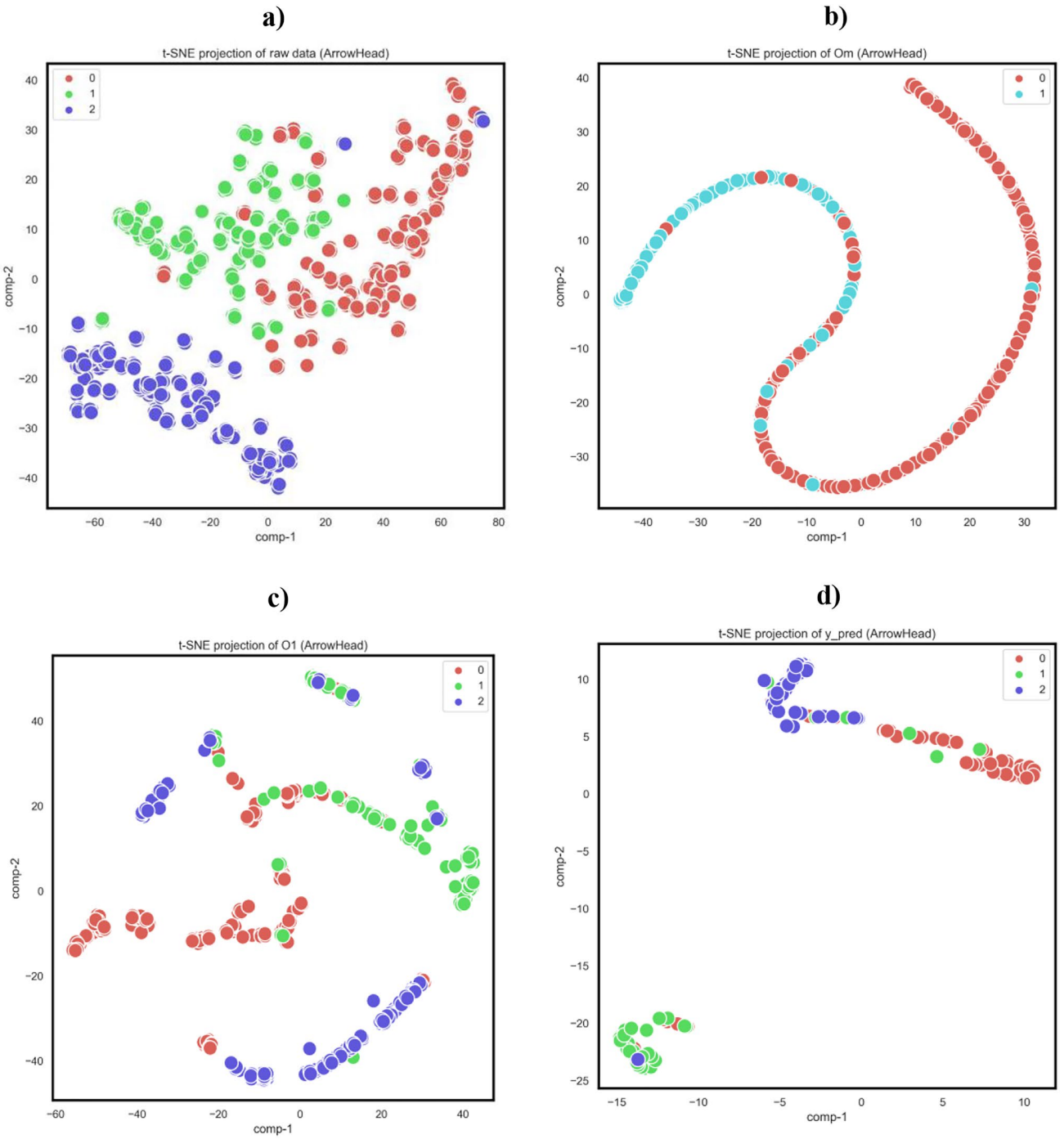


**Fig. 9** The t-SNE projection of **a** the raw data, **b** $O_m$ output of the proposed method, **c** $O_1$ output of the proposed method, and **d** the output of ResNet for 'ArrowHead' data set

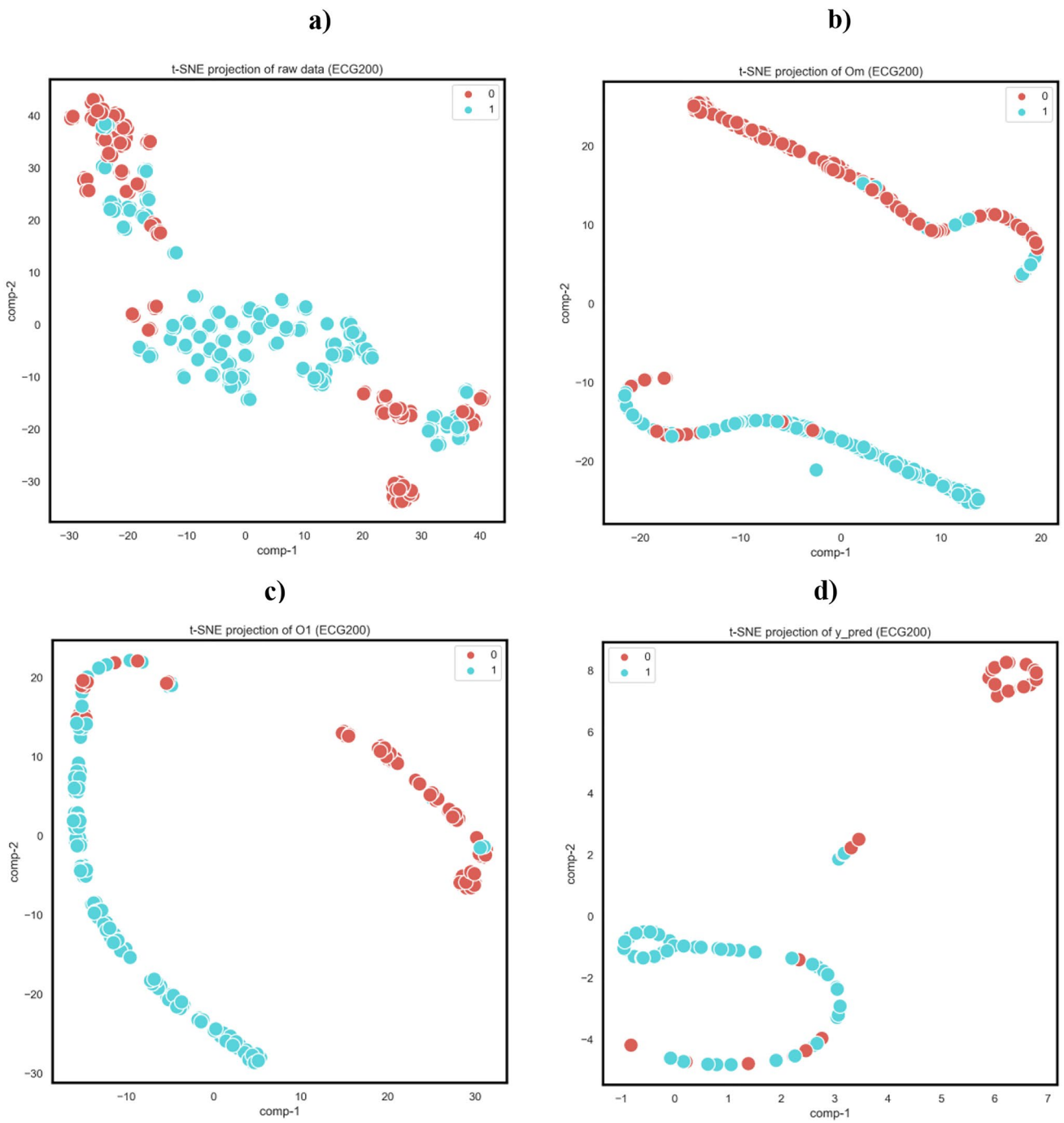


**Fig. 10** The t-SNE projection of **a** the raw data, **b** $O_{\mathrm{m}}$ output of the proposed method, **c** $O_1$ output of the proposed method, and **d** the output of ResNet for 'ECG200' data set

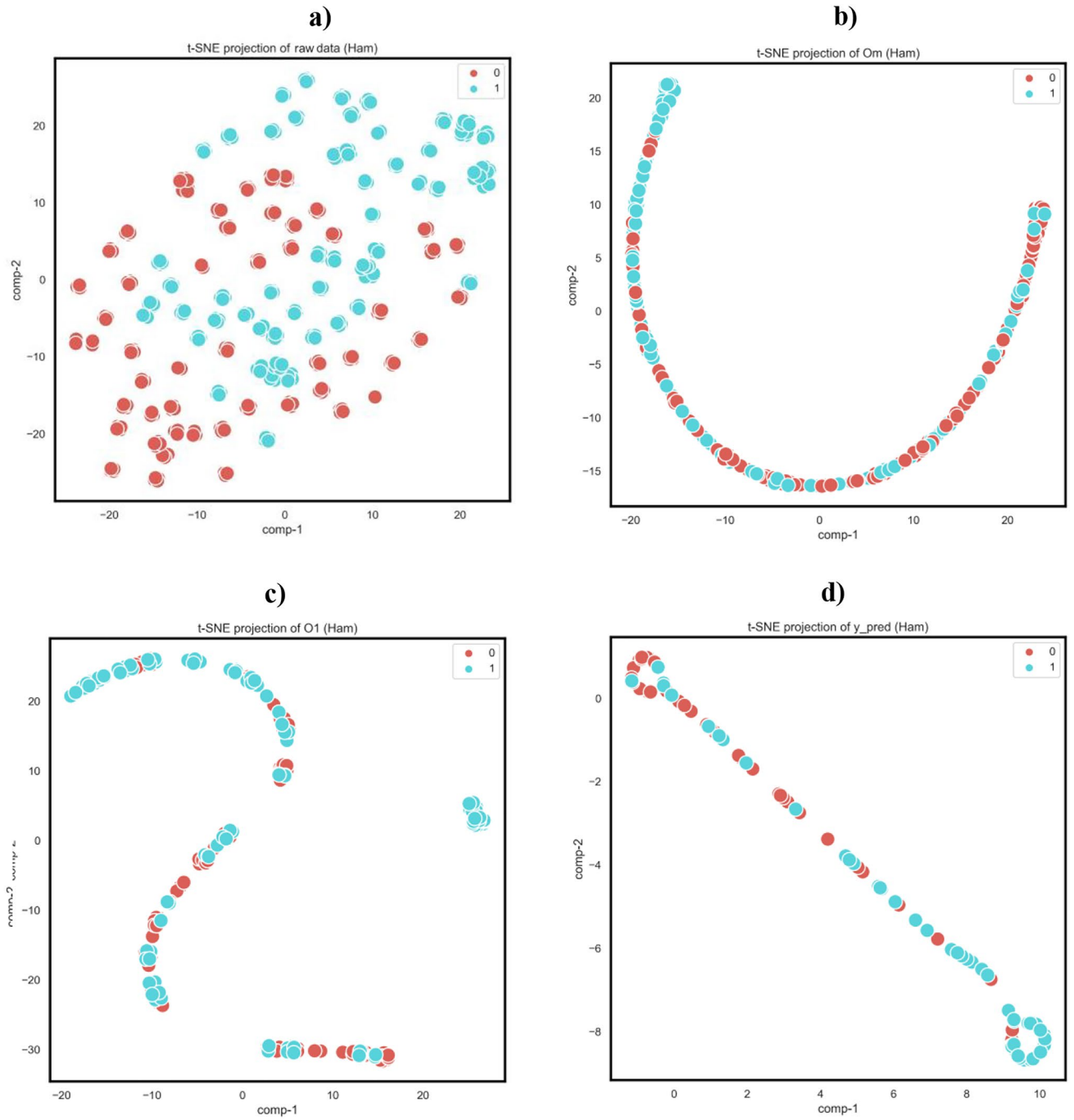


**Fig. 11** The t-SNE projection of **a** the raw data, **b** $O_m$ output of the proposed method, **c** $O_1$ output of the proposed method, and **d** the output of ResNet for 'Ham' data set

**Author Contributions** AT: conceptualization, methodology, software, formal analysis, investigation, writing—original draft, and visualization; GC: conceptualisation, resources, writing—review and editing, project administration, and funding acquisition; TMM: validation, resources, writing—review and editing, and project administration.

**Funding** The work was funded by The Leverhulme Trust Research Project under Grant RPG-2016-252 entitled "Novel Approaches for Constructing Optimised Multimodal Data Spaces".

**Availability of Data and Materials** N/A.

**Code Availability** Code will be available publicly in github after acceptance.

## Declarations

**Conflict of Interest** On behalf of all authors, the corresponding author states that there is no conflict of interest.



## References

1. Abanda A, Mori U, Lozano JA. A review on distance based time series classification. Data Min Knowl Disc. 2019;33(2):378–412. https://doi.org/10.1007/s10618-018-0596-4.
2. Alani AA, Cosma G, Taherkhani A. Classifying imbalanced multimodal sensor data for human activity recognition in a smart home using deep learning. Proc Int Jt Conf Neural Netw. 2020. https://doi.org/10.1109/IJCNN48605.2020.9207697.
3. Antoniades A, Spyrou L, Martin-Lopez D, Valentin A, Alarcon G, Sanei S, Took CC. Detection of interictal discharges with convolutional neural networks using discrete ordered multichannel intracranial EEG. IEEE Trans Neural Syst Rehabil Eng. 2017;4320(c):1–10. https://doi.org/10.1109/TNSRE.2017.2755770.
4. Antoniades A, Spyrou L, Martin-Lopez D, Valentin A, Alarcon G, Sanei S, Took CC. Deep neural architectures for mapping scalp to intracranial EEG. Int J Neural Syst. 2018;0(0):1850009. https://doi.org/10.1142/S0129065718500090.
5. Antonucci A, De Rosa R, Giusti A, Cuzzolin F. Robust classification of multivariate time series by imprecise hidden Markov models. Int J Approx Reason. 2015;56(PB):249–63. https://doi.org/10.1016/j.ijar.2014.07.005.
6. Aswolinskiy W, Reinhart RF, Steil J. Time series classification in reservoir- and model-space: a comparison. In: Schwenker F, Abbas HM, El Gayar N, Trentin E, editors. Artificial neural networks in pattern recognition. Cham: Springer International Publishing; 2016. p. 197–208.
7. Baydogan MG, Runger G, Tuv E. A bag-of-features framework to classify time series. IEEE Trans Pattern Anal Mach Intell. 2013;35(11):2796–802.
8. Bengio Y, Yao L, Alain G, Vincent P (2013) Generalized denoising auto-encoders as generative models. Advances in neural information processing systems, pp. 899–907.
9. Bianchi FM, Scardapane S, Jenssen R. Reservoir computing approaches for representation and classification of multivariate time series. 2018. https://arxiv.org/pdf/1803.07870.pdf
10. Chen Y, Garcia EK, Gupta MR, Rahimi A, Cazzanti L. Similarity-based classification: concepts and algorithms. J Mach Learn Res. 2009;10:747–76.
11. Chen Z, Liu Y, Zhu J, Zhang Y, Jin R, He X, et al. Time-frequency deep metric learning for multivariate time series classification. Neurocomputing. 2021;462:221–37. https://doi.org/10.1016/j.neucom.2021.07.073.
12. Chouikhi N, Ammar B, Alimi AM, Member S (2018) Genesis of basic and multi-layer echo state network recurrent autoencoder for efficient data representations. https://arxiv.org/ftp/arxiv/papers/1804/1804.08996.pdf
13. Cui Z, Chen W, Chen Y. Multi-scale convolutional neural networks for time series classification. 2016. https://doi.org/10.3724/SP.J.1077.2009.00909
14. Dau HA, Keogh E, Kamgar K, Yeh C-CM, Zhu Y, Gharghabi S et al. The UCR Time Series Classification Archive. 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
15. Dauphin YN, Fan A, Auli M, Grangier D. Language modeling with gated convolutional networks. In: The 34th international conference on machine learning—volume 70 (ICML'17), 2017. (pp. 933–41).
16. Ding C, Tao D. Robust face recognition via multimodal deep face representation. IEEE Trans Multimedia. 2015;17(11):2049–58. https://doi.org/10.1109/TMM.2015.2477042.
17. Fawaz HI, Forestier G, Weber J, Idoumghar L, Muller P-A. Deep learning for time series classification: a review. Data Min Knowl Discov. 2019. https://doi.org/10.1007/s10618-019-00619-1.
18. Fu TC. A review on time series data mining. Eng Appl Artif Intell. 2011;24(1):164–81. https://doi.org/10.1016/j.engappai.2010.09.007.
19. Gao Z, Wang X, Yang Y, Mu C, Cai Q, Dang W, Zuo S. EEG-based spatio-temporal convolutional neural network for driver fatigue evaluation. IEEE Trans Neural Netw Learn Syst. 2019. https://doi.org/10.1109/TNNLS.2018.2886414.
20. Garcia-Gasulla D, Parés F, Vilalta A, Moreno J, Ayguadé E, Labarta J, et al. On the behavior of convolutional nets for feature extraction. J Artif Intell Res. 2018;61:563–92.
21. Gehring J, Auli M, Grangier D, Yarats D, Dauphin YN. Convolutional sequence to sequence learning. Int Conf Mach Learn (ICML). 2017. https://doi.org/10.18653/v1/P16-1220.
22. Giusti R, Silva DF, Batista GEAPA. Improved time series classification with representation diversity and SVM. In: Proceedings—2016 15th IEEE international conference on machine learning and applications, ICMLA 2016, 2016, (1), 1–6. https://doi.org/10.1109/ICMLA.2016.108
23. Grabocka J, Schilling N, Wistuba M, Schmidt-Thieme L. Learning time-series shapelets. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, 2014, pp. 392–401. https://doi.org/10.1145/2623330.2623613
24. Gregory K, Zemel R, Salakhutdinov R. Siamese neural networks for one-shot image recognition gregorylation. In: 32th international conference on machine learning, Vol. 37; 2013. Lille, France, p. 1355. https://doi.org/10.1136/bmj.2.5108.1355-c.
25. Gudmundsson S, Runarsson TP, Sigurdsson S. Support vector machines and dynamic time warping for time series. In: 2008 IEEE international joint conference on neural networks (IEEE World congress on computational intelligence; 2008, pp. 2772–277662. https://doi.org/10.4018/978-1-5225-2498-4.ch012.
26. Le Guennec A, Malinowski S, Tavenard R. Data augmentation for time series classification using convolutional neural networks. In: ECML/PKDD workshop on advanced analytics and learning on temporal data; 2016. Riva Del Garda.
27. He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. In: IEEE conference on computer vision and pattern recognition (CVPR); 2016, pp. 770–778. Las Vegas. https://arxiv.org/pdf/1512.03385.pdf.
28. He Q, Dong Z, Zhuang F, Shang T, Shi Z. Fast Time Series Classification Based on Infrequent Shapelets. In: In 2012 11th international conference on machine learning and applications; 2012 (pp. 215–219). https://doi.org/10.1109/ICMLA.2012.44

29. Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A. Classification of time series by shapelet transformation. Data Min Knowl Disc. 2014;28(4):851–81. https://doi.org/10.1007/s10618-013-0322-1.
30. Hu Q, Zhang R, Zhou Y. Transfer learning for short-term wind speed prediction with deep neural networks. Renewable Energy. 2016;85:83–95. https://doi.org/10.1016/j.renene.2015.06.034.
31. Imblearn. Class to perform under-sampling based on NearMiss methods. 2003. https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.NearMiss.html?highlight=nearmiss
32. Jain B, Spiegel S. Dimension reduction in dissimilarity spaces for time series classification. In: International workshop on advanced analysis and learning on temporal data; 2015 (pp. 31–46).
33. Jean N, Burke M, Xie M, Davis WM, Lobell BD, Ermon S. Combining satellite imagery and machine learning to predict poverty. Science. 2016;353(6301):790–4.
34. Jeong YS, Jeong MK, Omitaomu OA. Weighted dynamic time warping for time series classification. Pattern Recogn. 2011;44(9):2231–40. https://doi.org/10.1016/j.patcog.2010.09.022.
35. Jonathan T, Goroshin R, Jain A, LeCun Y, Bregler C. Efficient object localization using convolutional networks. In: IEEE conference on computer vision and pattern recognition (CVPR); 2015 (pp. 648–656). Boston. https://doi.org/10.1109/CVPR.2015.7298664.
36. Kate RJ. Using dynamic time warping distances as features for improved time series classification. Data Min Knowl Disc. 2016;30(2):283–312. https://doi.org/10.1007/s10618-015-0418-x.
37. Kaya H, Gündüz-Öğüdücü Ş. A distance based time series classification framework. Inf Syst. 2015;51:27–42. https://doi.org/10.1016/j.is.2015.02.005.
38. Kenji B, Frinken V, Riesen K, Uchida S. Efficient temporal pattern recognition by means of dissimilarity space embedding with discriminative prototypes. Pattern Recogn. 2017;64(January 2016):268–76. https://doi.org/10.1016/j.patcog.2016.11.013.
39. Kingma DP, Ba J. Adam: a method for stochastic optimization. In: The 3rd international conference on learning representations (ICLR); 2014 (pp. 1–15). Banff. https://doi.org/10.1145/1830483.1830503
40. Krizhevsky A, Sutskever I, Geoffrey EH. ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems 25 (NIPS2012) (pp. 1–9); 2012. https://doi.org/10.1109/5.726791.
41. Längkvist M, Karlsson L, Loutfi A. A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recogn Lett. 2014;42(1):11–24. https://doi.org/10.1016/j.patrec.2014.01.008.
42. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–44. https://doi.org/10.1038/nature14539.
43. Li C, Zia MZ, Tran Q-H, Yu X, Hager GD, Chandraker MM. Deep supervision with intermediate concepts. IEEE Trans Pattern Anal Mach Intell. 2019;41(8):1828–43. https://doi.org/10.1109/CVPR.2017.49.
44. Li X, Lin J. Evolving separating references for time series classification. SIAM Int Conf Data Min SDM. 2018;2018:243–51. https://doi.org/10.1137/1.9781611975321.28.
45. Lin M, Chen Q, Yan S (2014). Network in network. In: International conference on learning representations (ICLR) (pp. 1–10). Banff.
46. Lin S, Runger GC. GCRNN: Group-constrained convolutional recurrent neural network. IEEE Trans Neural Netw Learn Syst. 2018;29(10):4709–18. https://doi.org/10.1109/TNNLS.2017.2772336.
47. Liu CL, Hsaio WH, Tu YC. Time series classification with multivariate convolutional neural network. IEEE Trans Industr Electron. 2019;66(6):4788–97. https://doi.org/10.1109/TIE.2018.2864702.
48. Liu J, Shahroudy A, Wang G, Duan L-Y, Kot AC. Skeleton-based online action prediction using scale selection network. IEEE Trans Pattern Anal Mach Intell. 2019;8828(c):1–15. https://doi.org/10.1109/CVPR.2018.00871.
49. Liu W, Wang Z, Liu X, Zeng N, Liu Y, Alsaadi FE. A survey of deep neural network architectures and their applications. Neurocomputing. 2017;234(October 2016):11–26. https://doi.org/10.1016/j.neucom.2016.12.038.
50. Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. Proc Int Jt Conf Neural Netw. 2015. https://doi.org/10.1109/IJCNN.2017.7966367.
51. Ma Q, Shen L, Chen W, Wang J, Wei J, Yu Z. Functional echo state network for time series classification. Inf Sci. 2016;373:1–20. https://doi.org/10.1016/j.ins.2016.08.081.
52. Malhotra P, Vig L, Agarwal P, Shroff G. TimeNet: pre-trained deep recurrent neural network for time series classification. In: 25th European symposium on artificial neural networks, computational intelligence and machine learning; 2017.
53. Mehdiyev N, Lahann J, Emrich A, Enke D, Fettke P, Loos P. ScienceDirect ScienceDirect time series classification using deep learning for process planning: a case from the process industry. Proc Comput Sci. 2017;114:242–9. https://doi.org/10.1016/j.procs.2017.09.066.
54. Min S, Lee B, Yoon S. Deep learning in bioinformatics. Brief Bioinform. 2017;18(5):851–69. https://doi.org/10.1093/bib/bbw068.
55. Mittelman, R. Time-series modeling with undecimated fully convolutional neural networks. 2015. https://arxiv.org/pdf/1508.00317.pdf
56. Mueen A, Young N (n.d.). Logical-Shapelets: an expressive primitive for time series classification, 1154–62.
57. Nweke HF, Teh YW, Al-garadi MA, Alo UR. Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: state of the art and research challenges. Expert Syst Appl. 2018;105:233–61. https://doi.org/10.1016/j.eswa.2018.03.056.
58. van den Oord A, Dieleman S, Zen H, Simonyan, K., Vinyals, O., Graves, A., et al. WaveNet: a generative model for raw audio. In: Speech Synthesis Workshop (SSW); 2016 (pp. 1–15). https://doi.org/10.1109/ICASSP.2009.4960364.
59. Özbay Y, Ceylan R, Karlik B. A fuzzy clustering neural network architecture for classification of ECG arrhythmias. Comput Biol Med. 2006;36(4):376–88. https://doi.org/10.1016/j.compbiomed.2005.01.006.
60. Page A, Shea C, Mohsenin T. Wearable seizure detection using convolutional neural networks with transfer learning. In: Proceedings—IEEE international symposium on circuits and systems, 2016-July, 2016, pp 1086–1089. https://doi.org/10.1109/ISCAS.2016.7527433
61. Peng S, Jiang H, Wang H, Alwageed H, Zhou Y, Sebdani MM, Yao YD. Modulation classification based on signal constellation diagrams and deep learning. IEEE Trans Neural Netw Learn Syst. 2018;30(3):718–27. https://doi.org/10.1109/TNNLS.2018.2850703.
62. Pw DR, Elzbieta P. Dissimilarity representation for pattern recognition, the: foundations and applications, Vol. 64; 2005. World scientific, Singapore.
63. Rajan D, Thiagarajan JJ. A Generative Modeling Approach to Limited Channel ECG Classification. In: 40th annual international conference of the IEEE engineering in medicine and biology society (EMBC); 2018 (pp. 2571–2574).
64. Rakthanmanon T (n.d.). Fast shapelets: a scalable algorithm for discovering time series shapelets, pp. 668–676.
65. Rios-Navarro A, Corradi F, Aimar A, Delbruck T, Milde MB, Tapiador-Morales R, et al. NullHop: a flexible convolutional neural network accelerator based on sparse representations of feature maps. IEEE Trans Neural Netw Learn Syst. 2018;30(3):1–13. https://doi.org/10.1109/tnnls.2018.2852335.

66. Sarkar S, Soundararajan P. Supervised learning of large perceptual organization: graph spectral partitioning and learning automata. IEEE Trans Pattern Anal Mach Intell. 2000;22(5):504–25. https://doi.org/10.1109/34.857006.
67. Serrà J, Pascual S, Karatzoglou A. Towards a Universal neural network encoder for time series. Artif Intell Res Dev Curr Challenges New Trends Appl. 2018;308:120–9. https://doi.org/10.3233/978-1-61499-918-8-120.
68. Song W, Wang Z, Liu L, Zhang F, Xue J, Ye Y, et al. Representation learning with deconvolution for multivariate time series classification and visualization. 2016. https://arxiv.org/pdf/1610.07258.pdf.
69. Taherkhani A, Cosma G, Alani AA, McGinnity TM. Activity recognition from multi-modal sensor data using a deep convolutional neural network. Adv Intell Syst Comput. https://doi.org/10.1007/978-3-030-01177-2_15.
70. Taherkhani A, Cosma G, McGinnity TM. AdaBoost-CNN: an adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. Neurocomputing. 2020. https://doi.org/10.1016/j.neucom.2020.03.064.
71. Tanisaro P, Heidemann G. Time series classification using time warping invariant Echo State Networks. In: Proceedings—2016 15th IEEE international conference on machine learning and applications, ICMLA 2016; 2017, pp. 831–836. https://doi.org/10.1109/ICMLA.2016.166.
72. Tian Y, Wang X, Wu J, Wang R, Yang B. Multi-scale hierarchical residual network for dense captioning. J Artif Intell Res. 2019;64:181–96.
73. Wang J, Ping L, She MFH, Nahavandi S, Kouzani A. Bag-of-words representation for biomedical time series classification. Biomed Signal Process Control. 2013;8(6):634–44. https://doi.org/10.1016/j.bspc.2013.06.004.
74. Wang Z, Yan W, Oates T. Time series classification from scratch with deep neural networks: a strong baseline. In: Proceedings of the international joint conference on neural networks, 2017-May; 2017, pp. 1578–1585. https://doi.org/10.1109/IJCNN.2017.7966039.
75. Wu Y, Schuster M, Chen Z, Le QV, Norouzi M, Macherey W et al. Google's neural machine translation system: bridging the gap between human and machine translation. 2016. http://arxiv.org/abs/1609.08144
76. Xing Z, Pei J, Keogh E. A brief survey on sequence classification. ACM SIGKDD Explorations Newsl. 2010;12(1):40. https://doi.org/10.1145/1882471.1882478.
77. Yang B, Liu R, Sun C, Meng G, Chen X. Dislocated time series convolutional neural architecture: an intelligent fault diagnosis approach for electric machine. IEEE Trans Industr Inf. 2017;13(3):1310–20. https://doi.org/10.1109/tii.2016.2645238.
78. Yannick R, Hubert B, Isabela A, Alexandre GHFT, Jocelyn F. Deep learning-based electroencephalography analysis: a systematic review. 2019. http://arxiv.org/abs/1901.05498
79. Ye L, Keogh E. Time series Shapelets: a new primitive for data mining. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining; 2009. (pp. 947–956).
80. Ye L, Keogh E. Time series shapelets : a novel technique that allows accurate, interpretable and fast classification. Data Min Knowl Disc. 2011;22:149–82. https://doi.org/10.1007/s10618-010-0179-5.
81. Zhang J, Mani I. kNN approach to unbalanced data distributions: a case study involving information extraction. In: Workshop on learning from imbalanced datasets; 2003. Washington DC.
82. Zhao B, Lu H, Chen S, Liu J, Wu D. Convolutional neural networks for time series classification. J Syst Eng Electron. 2017;28(1):162–9. https://doi.org/10.21629/JSEE.2017.01.18.
83. Zhao R, Yan R, Chen Z, Mao K, Wang P, Gao RX. Deep learning and its applications to machine health monitoring. Mech Syst Signal Process. 2019;115:213–37. https://doi.org/10.1016/j.ymssp.2018.05.050.
84. Zheng Y, Liu Q, Chen E, Ge Y, Zhao JL. Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. Front Comp Sci. 2016;10(1):96–112. https://doi.org/10.1007/s11704-015-4478-2.