Expert Systems With Applications 181 (2021) 115147

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Auto-adaptive multilayer perceptron for univariate time series classification

Felipe Arias del Campo, María Cristina Guevara Neri, Osslan Osiris Vergara Villegas *, Vianey Guadalupe Cruz Sánchez, Humberto de Jesús Ochoa Domínguez, Vicente García Jiménez

Universidad Autónoma de Ciudad Juárez, Instituto de Ingeniería y Tecnología, Avenida del Charro 450 Norte, Partido Romero, P.C. 32310 Ciudad Juárez, Chihuahua, Mexico

ARTICLE INFO

Keywords:
Time series
Time series classification
Multilayer perceptron
UCR data set

ABSTRACT

Time Series Classification (TSC) is an intricate problem that has encountered applications in various science fields. Accordingly, many researchers have presented interesting proposals to tackle the TSC problem. Nevertheless, most methods are hand-crafted to classify specific Time Series (TS) and are computationally expensive even for small data sets. In this paper, we propose a new approach to the Multilayer Perceptron (MLP) for TSC. The main novelty is that the hyperparameters related to batch size and the number of neurons in the hidden layers are auto-adapted according to the TS nature. We carried out an empirical study on 61 benchmark data sets from the University of California, Riverside (UCR). The experimental evaluation revealed that our proposal is competitive when we compare the accuracy versus 14 state-of-the-art methods. A non-parametric statistical test verifies that the proposed MLP ranked in fourth place and can be executed on standard computer equipment, making it simple, accessible, and competitive.

1. Introduction

Data classification refers to the activity of categorizing and organizing information for better analysis and decision-making. The aim is to generate a map from the input data to the desired output for a given training set (Li et al., 2021). The common data classification procedures are known as supervised and unsupervised.

In supervised classification, a prediction model is developed by learning from labeled data (training); hence, it is possible to make predictions for unlabeled examples (Uddin, Khan, Hossain, & Ali, 2019). Conversely, in unsupervised classification, the data is no tagged; hence, no training phase is required. The prediction model uses clustering to define the number of classes (Kucuk & Avdan, 2020).

Because many practical situations can be expressed as associations between two variables, supervised data classification has numerous applications, where the most common is image classification (Rawat & Wang, 2017). However, there are data known as Time Series (TS) whose characteristics make them unique. TS is a finite sequence of real values extracted from successive observations over a regular time interval. TS can be univariate, where at each time instant, only one real value is taken, or multivariate, where many real values are obtained simultaneously (Lahreche & Boucheham, 2021).

Time Series Classification (TSC) is different from traditional classification because the attributes are ordered. TSC implies learning a function that maps a series into a class from a predefined class set. The research in TSC has been of particular interest in various fields, including classification of weather readings (Soares, Costa, & Leite, 2018), biomedical signals (Azami, Fernández, & Escudero, 2017), financial records (Chang, Zhipeng, & Yuanjie, 2019), and psychological signals (Jebb, Tay, Wang, & Huang, 2015).

Different methods of grouping TSC algorithms have been proposed in the literature (Bagnall, Lines, Bostrom, Large, & Keogh, 2017). Nevertheless, in this paper, TSC techniques are grouped into three categories according to the 14 baseline algorithms employed for comparisons (i) based on features (Hu et al., 2018; Kenji & Uchida, 2020), (ii) based on ensembles (Bagnall, Lines, Hills, & Bostrom, 2015; Wei, Petitjean, & Webb, 2020), and (iii) based on deep learning (Chen & Shi, 2019; Liu, Hsaio, & Tu, 2019).

* Corresponding author.
E-mail addresses: al171515@alumnos.uacj.mx (F. Arias del Campo), al171517@alumnos.uacj.mx (M.C. Guevara Neri), overgara@uacj.mx (O.O. Vergara Villegas), vianey.cruz@uacj.mx (V.G. Cruz Sánchez), hochoa@uacj.mx (H.J. Ochoa Domínguez), vicente.jimenez@uacj.mx (V. García Jiménez).

https://doi.org/10.1016/j.eswa.2021.115147
Received 18 January 2021; Received in revised form 29 April 2021; Accepted 29 April 2021
Available online 19 May 2021

In feature-based methods, a discriminatory feature of the TS is chosen before the classification phase. The selected feature represents global or local patterns, which are passed through classifiers (Schäfer & Leser, 2020). The idea underlying these methods is the dimensionality reduction by using a set of features to represent the whole TS. The four algorithms selected in this category were: Word ExtrAction for time SEries cLassification (WEASEL) (Schäfer & Leser, 2017), Time Series Forest (TSF) (Deng, Runger, Tuv, & Vladimir, 2013), the CAnonical Time-series CHaracteristics (Catch22) (Lubba et al., 2019), and Shapelet Transform Classifier (STC) (Zhao, Pan, & Tao, 2020).

Conversely, ensemble-based approaches utilize a combination of different classifiers integrated to achieve greater classification accuracy. Seven algorithms were selected in this category, including the HIerarchical VotE Collective Of Transformation-based Ensembles (HIVE-COTE) (Lines, Taylor, & Bagnall, 2018), the Bag Of Symbolic Fourier approximation Symbols (BOSS) (Schäfer, 2015), Contract BOSS (cBOSS) (Middlehurst, Vickers, & Bagnall, 2019), Spatial BOSS (sBOSS) (Large, Bagnall, Malinowski, & Tavenard, 2019), Random Interval Spectral Ensemble (RISE) (Flynn, Large, & Bagnall, 2019), Proximity Forest (PF) (Lucas et al., 2019) and Time Series Combination of Heterogeneous and Integrated Embeddings Forest (TS-CHIEF) (Shifaz, Pelletier, Petitjean, & Webb, 2020).

In the last decade, the use of deep learning for TSC has grown considerably (Fawaz, Forestier, Weber, Idoumghar, & Muller, 2019). Deep neural networks are based on learning by transforming the raw input data into a more abstract representation, suitable for classification (LeCun, Bengio, & Hinton, 2015). A wide variety of deep neural network architectures, such as Convolutional Neural Network (CNN) models (Chen & Shi, 2019; Liu et al., 2019), and Residual Neural Network (ResNet) (Xiaowu, Zidong, Qi, & Weiguo, 2019), are commonly employed for TSC. The three selected methods in this category were: the ResNet (Xiaowu et al., 2019), the Inception Time (Fawaz et al., 2020), and the RandOm Convolutional KErnel Transform (ROCKET) (Dempster, Petitjean, & Webb, 2020).

In summary, numerous methods have attempted to solve the TSC problem. However, as a result of the literature review: (i) the methods have not yet achieved an entirely successful performance (Dau et al., 2018), (ii) classifiers are hand-crafted to solve specific problems. Hence, when the length or nature of the TS changes, the algorithm must be redesigned (Bagnall et al., 2017); (iii) computer equipment with high processing power (clusters) and a large amount of memory are needed to perform TSC even for small data sets (Dempster et al., 2020), (iv) recent algorithms are becoming more complex and challenging to understand, and (v) when the hyperparameters in an Artificial Neural Network (ANN) model have been fixedly adjusted to process the TS specific characteristics, they cannot be reused for a TS with different properties.

In machine learning, classification methods predict a class, while regression methods are designed to predict continuous numeric outputs. Although new algorithms for classification and regression have been developed, the Multilayer Perceptron (MLP) has remained a topic of interest for the scientific community. Multilayer perceptrons are often used due to their flexibility and the capability to fit a wide range of smooth, non-linear functions with high accuracy levels.

Recent examples on the use of the MLP for classification, include the detection of melanoma (Sánchez, Rodríguez, Salazar, Avecilla, & Pérez, 2020), liner surface defects in solid rocket motors (Simoes, Parquet, & Parquet, 2020), neonatal sleep-wake classification (Farooq et al., 2020), and thyroid disease diagnosis (Hosseinzadeh et al., 2021). On the other hand, MLP has been recently used as a regression method to predict the viscosity of a nanofluid (Hemmati, Varamesh, Husein, & Karan, 2018), the thermal conductivity of carbon dioxide (Nait, Jahanbani, & Zeraibi, 2020), and the solubility of carbon dioxide in crude oil (Mahdaviara et al., 2021).

The classification problem has been solved using robust methods including Least Square Support Vector Machines (LS-SVM) (Singh et al., 2020), Support Vector Machines (SVM) (Singh, Chandra, Kumar, Dass, & Bhatnagar, 2020), CNN's (Tang et al., 2020), and genetic algorithms (Ahn & Hur, 2020). However, we choose the MLP based on two main reasons. First, the implementation of most TSC techniques requires an advanced level of understanding of mathematical procedures. Second, the machine learning solutions implemented to solve the TSC problem require Graphics Processing Unit (GPU) cards to work around the intensive mathematical processing needed for training. In contrast, the MLP has the advantages that it does not require deep mathematical knowledge, the model complexity is lower than other methods, for example, the SVM (Zanaty, 2012), and finally, the MLP is more used than the SVM (Hesami, Naderi, Tohidfar, & Yoosefzadeh-Najafabadi, 2020).

In this work, we propose an alternative MLP for univariate TSC for one, two, and three layers. The input data do not require many cycles of mathematical processing, and the hyperparameters are dynamically adjusted depending on the size of the training set and the number of measurements in the TS. In summary, the contributions of this paper are as follows:

1. We explain how to auto-adapt the proposed MLP for univariate TSC.
2. We describe how to dynamically auto-adapt the MLP hyperparameters regarding batch size and the number of neurons in the hidden layer according to the TS nature.
3. We use 61 univariate UCR data sets to benchmark our MLP against 14 state-of-the-art methods.

The rest of the paper is organized as follows. A review of the works for hyperparameters optimization with different algorithms is presented in Section 2. Section 3 describes the materials used for the experiments. Section 4 shows the proposed MLP classifier architecture. The details of the experiments conducted and the corresponding results are presented in Section 5. Finally, Section 6 offers the concluding remarks.

2. Literature review

In the literature, many papers have addressed the problem of neural networks hyperparameter optimization. After the perusal conducted to analyze the newest papers (2016–2021), at least four main research branches were detected: (i) the search-based methods (grid (Pontes, Amorim, Balestrassi, Paiva, & Ferreira, 2016), random (Liashchynskyi & Liashchynskyi, 2019) and dynamic encoding (Yoo, 2019)); (ii) the sampling-based methods (genetic algorithms (Cui & Bai, 2019) and Bayes (Theckel, Rana, Gupta, & Venkatesh, 2020)); (iii) the model-based methods (reinforcement learning (Wu, Chen, & Liu, 2020)), and (iv) the auto-adaptive methods (apoptosis (Siegel, Daily, & Vishnu, 2010), batch size (Devarakonda, Naumov, & Garland, 2017), number of neurons (Garro, Sossa, & Vazquez, 2009), number of connections (Garro, Sossa, & Vazquez, 2011), and our proposal).

The search-based methods employ brute-force or a simple rule to select the best from a given subset of hyperparameter values. These methods are the most frequently used due to their simplicity. However, they tend to be inefficient because they take a long time to execute and yield high variance during calculation.

Pontes et al. (2016) proposed combining the design of experiments and focused grid search on tuning the hyperparameters of an MLP to predict average surface roughness. The learning rate, number of epochs, and neurons in hidden layers were optimized. Two data sets were employed for experimentation, and no information about the equipment utilized was offered. The results revealed a reduction of the prediction error between 71.5% and 82.3% compared to techniques currently used. A comparison between genetic algorithm and grid and random searches for neural networks hyperparameter optimization was presented by Liashchynskyi and Liashchynskyi (2019). The hyperparameters optimized were the number of convolutional and dense layers. The experiments used the CIFAR-10 data set in a powerful computer with Nvidia Tesla K80 GPU. The best results were obtained with the genetic

algorithm. Although, it took more time to compute the solution.

Yoo (2019) used a univariate Dynamic Encoding Algorithm for Searches (uDEAS) for hyperparameter optimization. The proposal was tested with an autoencoder and a CNN using the MNIST data set. The learning rate, batch size, number of hidden layers, number of filters, and the first and second convolution layers were optimized. The authors argued that few computational resources were employed to achieve fast convergence. However, no detail about the equipment used for testing was offered.

Sampling-based methods utilize a policy to guide the sampling process. The policy is updated by considering the evaluation of a new sample. These methods are better because they make smarter decisions than search-based algorithms. However, it is hard to determine the representation and size of the initial population, and there is no correct way to choose a prior.

Cui and Bai (2019) optimized the CNN hyperparameters by combining multiscale and multilevel evolutionary optimization with Gaussian process Bayesian optimization. The optimized hyperparameters were the size of layers and kernels, learning rate, momentum, weight decay, and dropout. The experiments were conducted with powerful workstations on three data sets: CIFAR-10, CIFAR-100, and ILSVRC-2012. Experimental results showed good performance and adaptability to optimize the hyperparameters with various numerical types.

Theckel et al., 2020 developed a framework for hyperparameter optimization of an MLP and a CNN based on Bayes theory. The optimization was performed on a small subset of training data and then with the whole data using directional derivative signs. The letter recognition, MNIST, adult income prediction, vehicle, and CIFAR-10 data sets were used to conduct the experiments. The hyperparameters tuned were the number of neurons, batch size, dropout, dropout weight, mini-batch size, learning rate, and momentum. No information about the computational resources to conduct the tests was offered. The results obtained outperformed standard Bayesian optimization.

The model-based methods allow learning a model of the environment to determine the search strategy by trial and error. However, these methods need feedback about the agent's action and require a lot of data and computation.

A model that applies reinforcement learning to adjust the hyperparameters of a CNN was presented by Wu et al. (2020). The optimization was treated as a Markov decision problem, and reinforcement learning was employed to select the hyperparameters sequentially. The batch size, convolutional stride, kernel channel, pooling stride, kernel type, fully connected layers nodes, and learning rate were optimized. A total of 101 data sets from the OpenML and UCI were employed for experimentation. The proposal achieved an accuracy of 86.1% for the 101 tasks. Unfortunately, no information about the equipment for experimentation was offered.

The auto-adaptive methods refer to auto-adapt the hyperparameters according to the features or nature of the data set to classify.

An approach to adaptively remove unnecessary neurons of CNNs was proposed by Siegel et al. (2010). The process of neuron pruning was called apoptosis, and after removal, it is not required to retrain the model. The data sets used were Higgs Bosson, Imagenet, and MNIST. The approach reduced the overall training time by 2-3x, obtaining a competitive accuracy. However, it must be executed in a supercomputer or similar large-scale system.

Devarakonda et al. (2017) presented an approach for AlexNet, ResNet, and VGG networks in which the batch size was doubled every 20-epoch, and the learning rate was adapted with a decay of 0.75. The data sets employed were CIFAR-10, CIFAR-100, and ImageNet. The batch size dimension increased to 524,288. Therefore, a powerful and expensive four NVIDIA Tesla P100 GPUs were needed. The running times of adaptive batch sizes were reduced with less than a 1% accuracy difference compared to fixed batch size.

Garro et al. (2009) implemented Particle Swarm Optimization (PSO) to find the best topology, the number of neurons, the transfer function for each neuron, and the synaptic weights of an ANN. Four data sets were used to evaluate the algorithm's accuracy: XOR, iris plant, wine, and breast cancer. The performance was compared against the Back-Propagation (BP) algorithm in an ANN composed of three layers, with a learning rate of 0.1, the same data sets, and the same number of epochs. The proposed algorithm performed better than the BP.

Garro et al. (2011) presented an Artificial Bee Colony (ABC) method that minimizes the number of connections and maximizes the accuracy in an ANN. The proposal evolved the synaptic weights, the ANN's architecture, and simultaneously the transfer functions of each neuron. The algorithm performance was evaluated with four data sets: iris plant, wine, breast cancer, and a real object recognition problem. The experiments were validated with two different fitness functions, the Mean Square Error (MSE) and the Classification ERror (CER). The experiments proved that the proposal provided a good optimization, given that it is possible to find the optimal values to construct an ANN automatically.

As aforementioned, several methods have been proposed to solve the task of neural networks hyperparameters optimization. Most of the methods were global optimizers that can deal with multi-objective, constrained, and high-dimensional problems. The algorithms reviewed were tested with various data sets, but there were no references about its performance on TS. Moreover, all the works analyzed proposed using very deep networks to solve specific problems, requiring considerable time and computer resources (e.g., GPUs, RAM, storage space) to be executed, and the number of optimized hyperparameters varied from two to nine.

Regarding auto-adaptive methods, four methodologies were detected. Neuron pruning and PSO were used to adapt the network topology. Another method doubled the batch size and decreased the learning rate by 0.75 every 20 epochs. Finally, ABC was used to minimize the number of connections in an ANN. In contrast, our proposal considers the nature of the TS to adapt the number of neurons in the hidden layers and the batch size. Moreover, only standard computer equipment is needed to conduct the experiments, and the accuracy obtained is competitive with 14 state-of-the-art methods.

3. Materials

In this section, we describe the materials used to conduct the experiments. First, we detail the computer equipment's characteristics. Then, we describe the software employed to implement the tests. Finally, we present the selected TS data sets.

3.1. Computer equipment

To evaluate the performance of the proposed method on a commercial computer, we used an Intel i7-4790 K 64-bit desktop computer with Windows 10. The computer included 20 GB of Random-Access Memory (RAM) and a Graphics Processing Unit (GPU) NVIDIA GeForce RTX 2070 with 8 GB of RAM.

3.2. Software

Different development tools can be used to implement Artificial Neural Networks (ANNs) models, including Caffe, Theano, Microsoft Cognitive Toolkit, Torch, and TensorFlow, to name a few. In this work, we used the TensorFlow 2.0 open-source software.

Also, to use the GPU's computing capacity, it was required to install additional drivers provided by the manufacturer NVIDIA, version 10.1 of Compute Unified Device Architecture (CUDA).

3.3. UCR data set

In this work, we selected the univariate subset of the TS data set published by the University of California, Riverside (UCR) (Dau et al.,

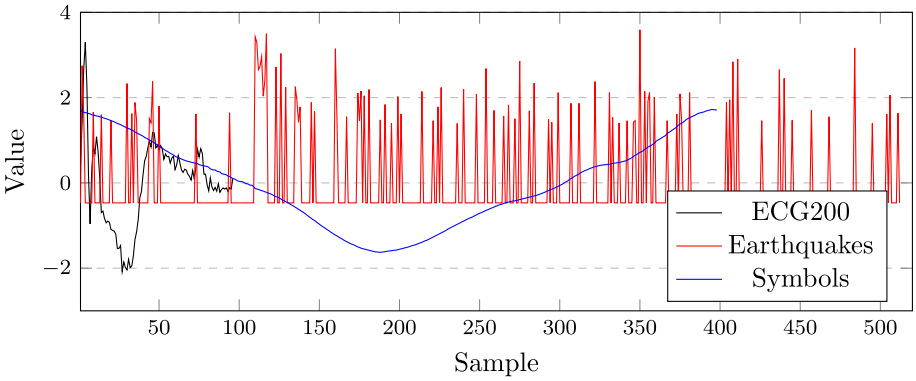

Fig. 1. The plot of three TS from the UCR data set.

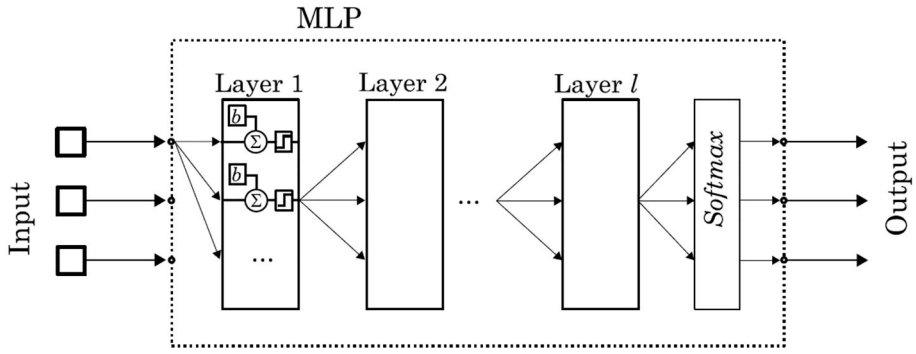

Fig. 2. The proposed MLP architecture.

2018). Because it is used for benchmarking purposes in other works (Bagnall et al., 2015; Bagnall et al., 2017; Abanda, Mori, & Lozano, 2019; Kenji & Uchida, 2020; Wei et al., 2020; Schäfer & Leser, 2020). The UCR data set contains 128 different TS, from which 108 have reported classification accuracy results using the 14 algorithms mentioned in Section 1.

The length of the UCR TS ranges from 24 to 2709. The training sets go from 16 to 8926, and the number of classes from 2 to 60. The UCR TS included audio, images, motion, ECG, HAR, spectrum, simulated sensors, traffic, EEG, financial, hemodynamics, and APG. From the 108 series, 47 were left out of this study because they did not comply with the inclusion criteria: (i) the amount of data in the series required more RAM than the memory available, or (ii) the time required to train the MLP exceeded the 15 min that were arbitrarily defined as the limit to perform the training.

The 61 selected series have different characteristics in terms of the number of samples in the training and validation sets, the number of samples per series, and the range of values between the measurements. Fig. 1 shows an example of three series randomly selected from the data set. The plots exhibit the diversity of values, the shape of graphs, and the number of samples. Note that the *Symbols* TS has a slight variation between two consecutive measurements; hence, the plot is a smooth curve. The *Earthquakes* TS has high and low values alternated very frequently. Unlike the other two series, *ECG200* TS has only 96 measurements in terms of the amount of data, representing almost a quarter of *Symbols* and a fifth of *Earthquakes*.

4. The proposed MLP architecture

The proposed model for UCR TSC is an MLP with one, two, and three layers. This section describes the structure of a traditional MLP and offers a detailed explanation of our proposal.

4.1. Multilayer perceptron

Due to its reliable performance, the MLP is one of the most frequently employed ANN (Nait, 2020). MLP evolves the simple perceptron and is part of the family of the feed-forward neural networks. Incorporating one or more hidden layers, the MLP can represent non-linear functions (Hemmati, Nait, Reza, Dai, & Zhang, 2020).

The traditional MLP is composed of one input layer (which receives the input data), l hidden layers (responsible for learning non-linear features), and one output layer (which provides the output). One hidden layer is sufficient to make an MLP an universal approximator (Guliyev & Smailov, 2018). The MLP receives a data vector as input, and the output is a vector that defines the class to which the input data belongs (Seo & Cho, 2020).

Fig. 2 shows the general concept of an MLP made of l hidden layers. Each layer has a certain number of identical elements known as neurons. Each neuron of one layer is connected to the neurons of the next layer without feedback.

For the sake of clarity on how an MLP works, consider w_{ij} as the weight between the input neuron i and the hidden neuron j; and w_{jk} as the weight between the hidden neuron j and the output neuron k.

All the layers have multiple neurons where addition and multiplication operations are conducted. When an input pattern $(x_1, x_2, \ldots x_n)$ is presented, it is passed using w_{ij} weights from the input to the hidden layer. The product is applied to all the elements of the input vector for each neuron. The values obtained are added up (dot product) to get an output O_j; this process is shown in Eq. (1).

$$O_j = f\left(\sum_{i=1}^{n^{[0]}} w_{ij}x_i + b_j\right) \tag{1}$$

where $n^{[0]}$ is the number of features in the input pattern, b is known as the bias that allows the shifting of linear combinations to either right or

left, and $f(.)$ is the activation function that determines whether the output of the neuron is activated or not.

Common activation functions employed in MLP are the sigmoid, the hyperbolic tangent, tan-sigmoid and log-sigmoid (Nait, Abdelfetah, & Ouaer, 2021). However, we used the Rectified Linear Unit (ReLU) to overcome the numerical problems related to the sigmoid such as the vanishing gradient.

Each neuron activation function's resulting values form the layer's output vector and become the input to the next layer in the model, where the same process is repeated. An MLP can contain any number of layers, and each layer can have a different number of neurons. Each neuron connects to all the neurons in the next layer, creating a dense connection. The layers in this model are also known as dense layers or fully connected layers.

The *lth* layer's output is computed by performing the dot product of the previously hidden layer's outputs and the actual hidden layer weights plus the actual bias. The result is the input to the activation function $f(.)$ as shown in Eq. (2).

$$\widehat{Y}_k = f\left(\sum_{j=1}^{n^{[l-1]}} w_{jk} O_j + b_k\right) \tag{2}$$

where $n^{[l-1]}$ is the number of hidden neurons in the previous layer.

The softmax activation function was used in the output layer. Thus, the MLP output is a probability vector (the sum of probabilities is equal to 1.0) of the same length as the total number of classes (Li et al., 2019). Once the probability vector is obtained, the highest value is located, and its position indicates the class to which the input belongs.

Only the vector element representing the correct class has a value of 1, and the remaining elements have a value of 0. Thus, the input string has a probability of 100% belonging to the specific class and 0% to the other classes.

The training process of the model is performed by comparing the last output vector against an expected vector. The expected vector is an array of dimensions equal to the number of classes of the series. All the computations until this step are known as the forward phase. After this point, the backward propagation phase starts.

The difference between the output and the desired vectors represents the model error, and it is used to adjust the values of the previous layer weights. The goal is to minimize the mean squared error between the obtained output and the desired output. The error is computed using Eq. (3).

$$E_p = \frac{1}{2}\sum_{k=1}^{m}\left(d_k - \widehat{Y}_k\right)^2 \tag{3}$$

where m is the number of examples in the data set, d_k is the desired output for the *kth* input example.

The gradient of the error is multiplied by the learning rate (η), and the result is used to update each neuron's weight in the corresponding layer under training. When the model is created, the weights are randomly initialized with a small value. The gradient is progressively backward passed to each layer, starting at the last layer until reaching the first layer to adjust the weights.

The gradient is a vector equal to the partial derivative of E_p regarding each weight and takes the direction of the fastest error increase, while the opposite direction determines the fastest error decrease. The error is reduced by adjusting each weight in the direction shown in Eq. (4).

$$-\eta\frac{\partial E_p}{\partial w_{ij}} \tag{4}$$

The weights are updated using Eq. (5).

$$w_{ij} := w_{ij} - \eta\frac{\partial E_p}{\partial w_{ij}} \tag{5}$$

The backpropagation process is performed starting from the last layer to the first one. That is why it is called backpropagation (short of backward propagation of error).

4.2. The proposed hyperparameter optimization method

As well as other neural networks, MLP has hyperparameters such as the number of hidden layers, the number of neurons in each hidden layer, type of activation function, batch size, dropout, and learning rate, which must be tuned.

Hyperparameter optimization is a challenging task that is frequently solved by experimented designers or by computational brute force. In this work, we propose a way to determine the number of hidden layers and the number of neurons in the MLP, and the batch size according to the TS nature. We have not considered the last layer of the MLP as part of the model; therefore, the value l was used to define the number of layers.

The proposed l-layer model is shown in Fig. 2. However, we experimented with one, two, and three layers to find the optimal number of layers. For a four-layer model, the memory requirements exceeded the RAM available in the GPU for larger TS. We also observed that the TS that could be tested with four layers showed a lower accuracy than that obtained for the model with three layers. The learning process of the network is supervised. The data set consists of 2-tuples, each with the values of the series (input vector) and the probability of the class this vector belongs to (output vector).

In the training stage, groups of one or more pairs of the training set are presented (input vector and expected vector); this group is called a batch. Once the whole batch has been presented to the model, the error is calculated, and the weights are updated.

The selection of the batch size hyperparameter influences the dynamics of the learning algorithm in two ways: (i) a large batch reduces the training time, and because the error is calculated using multiple pairs, the training is more efficient, and (ii) a small batch increases the training time, and because few elements are used to calculate the error, some noise is added to the process of weight updating. A small-batch can lead to the model working better for generalization (it can classify values that were not included in the training set) (Dostal et al., 2020; Kandel & Castelli, 2020).

An epoch occurs when all batches in the training set have been used. The training cycle consists of running several training epochs until the error obtained falls below an acceptable value or the maximum number of allowed epochs is reached. However, the epoch can also be defined as processing a defined number of batches and not necessarily all of them. For the experiments conducted, an epoch was considered as using all the batches.

Because the TS used to evaluate the model has different lengths, both in measurements and in the number of samples, the batch size selection could not be fixed for all the TS. It is also not recommended to set the batch to the size of the entire training data set.

It was discovered through the diverse experiments that the different TS data sets required different hyperparameters. That means there was no general solution to the TSC problem. Therefore, a large number of experiments were executed to evaluate different hyperparameters and the model performance. As a result, Eqs. (6) and (7) were proposed to calculate the batch length and the hidden layer size, respectively.

The equations were designed to involve the TS data sets attributes found to be distinctive, such as the size of the training set or the time series length. By including those attributes as variables in the equations, the MLP hyperparameters could be adjusted automatically for each TS data set.

Eq. (6) depicts the proposed adaptive calculation of the batch size concerning the size of the training set.

$$Batchsize = \left\lfloor \frac{Size\ \ of\ \ the\ \ training\ \ set}{Divisor} \right\rfloor \tag{6}$$

where the way to compute the *Divisor* is explained in this section. The

Table 1
The TS characteristics used to explain how the *Divisor* and *Multiplier* values were calculated.

Data set	No. of training sets	No. of test sets	No. of classes	Data points
Beef	30	30	5	470
FiftyWords	1800	858	2	80
PhalangesOutlinesCorrect	455	455	50	270

floor function $\lfloor . \rfloor$ returns the largest integer that is smaller than, or equal to, the result of the division. The minimum value that *Divisor* can take is one since the division by zero results in an undefined value.

On the other hand, establishing the number of neurons in the hidden layers is crucial because it affects the neural network's capability for generalization and the training time. If a reduced number of neurons is used, it might result in underfitting, meaning the neural network is incapable of learning the data variability. Conversely, if many neurons are used, it might result in overfitting, meaning that the neural network learns the detail and noise in the training data. Hence, during the validation stage, the network will not identify the new examples. Consequently, the generalization capabilities will be impaired. Moreover, many neurons in the hidden layers increase the time, making impossible the training of the network in standard computer equipment (Aalst et al., 2010).

Neural networks have been studied for several years. However, there is no fully valid method to determine the number of neurons in the hidden layers. Therefore, a trial and error method is usually employed to identify the suitable number of neurons in each hidden layer (Nait et al., 2021). Considering this, we propose the Eq. (7) to determine the number of neurons in hidden layers adaptively.

$$Neurons = TSLength * LayerIndex * Multiplier \tag{7}$$

where *TSLength* is the number of points measured in each TS, *LayerIndex* is an index assigned from the last to the first hidden layer. In the three-layer model, the assigned sequential values are 3, 2, 1. The *Multiplier* factor is computed later in this section.

In Eq. (6), a *Divisor* value of 24, and in Eq. (7), a *Multiplier* value of 9 were obtained after running several training cycles. For the sake of clarity on how these values were obtained, consider the information of the three TS shown in Table 1. The three TS selected represents the different shapes included in the data sets. *Beef* has a reduced number of training sets while *FiftyWords* has a larger number. *PhalangesOutlinesCorrect* has more classes to differentiate, and the training set size is between the size of the other two TS.

Each TS is used to train the MLP by changing the *Divisor* from 1 to 54 in steps of one. The training time and loss values were collected by averaging ten cycles (to improve the repeatability), as it is shown in Fig. 3. The solid line represents the loss values and the dotted line the time. Notice that TS shares the same type of marker for loss and time.

The solid vertical line in Fig. 3 shows the value of 24 for the *Divisor*. It establishes a trade-off between loss and time for the three TS.*Beef* is the shortest TS with 30 training sets. Therefore, when the *Divisor* goes higher than 15, the effective batch size is always one. Moreover, this value does not introduce an adverse effect on loss and time. The changes are only

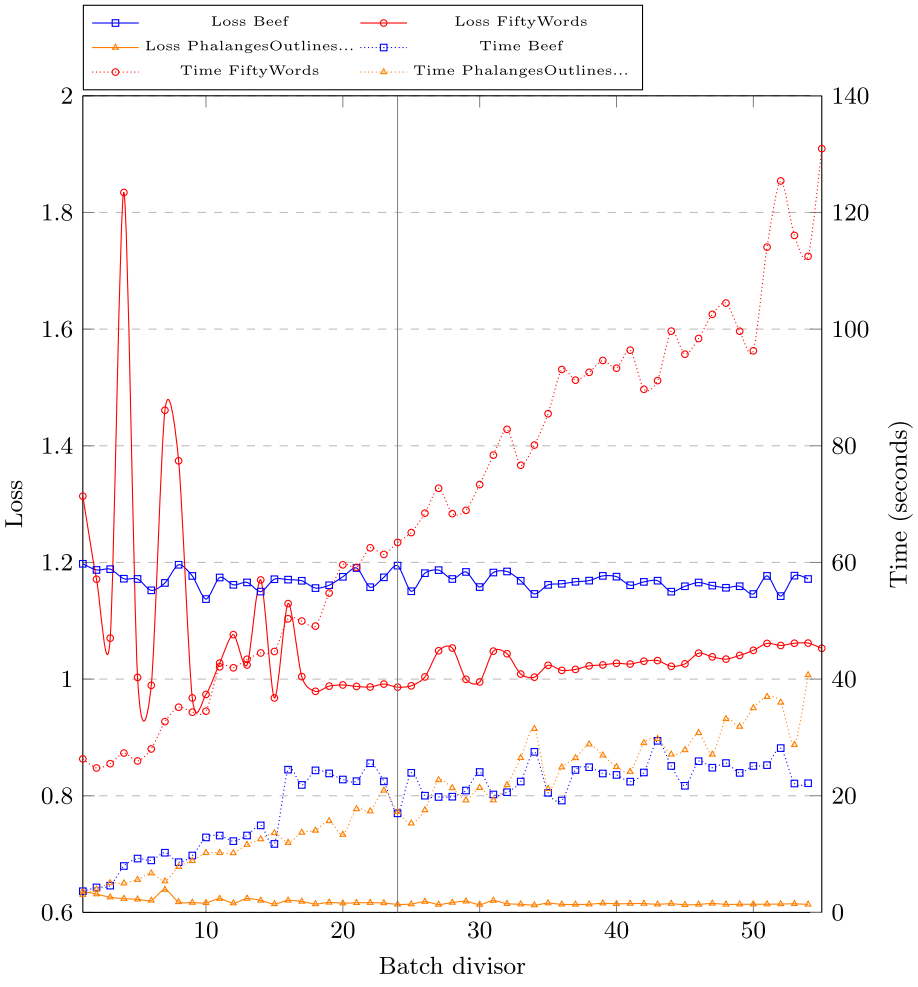

Fig. 3. The loss and time obtained by averaging ten training cycles for three TS with values from 1 to 54 for the Divisor.

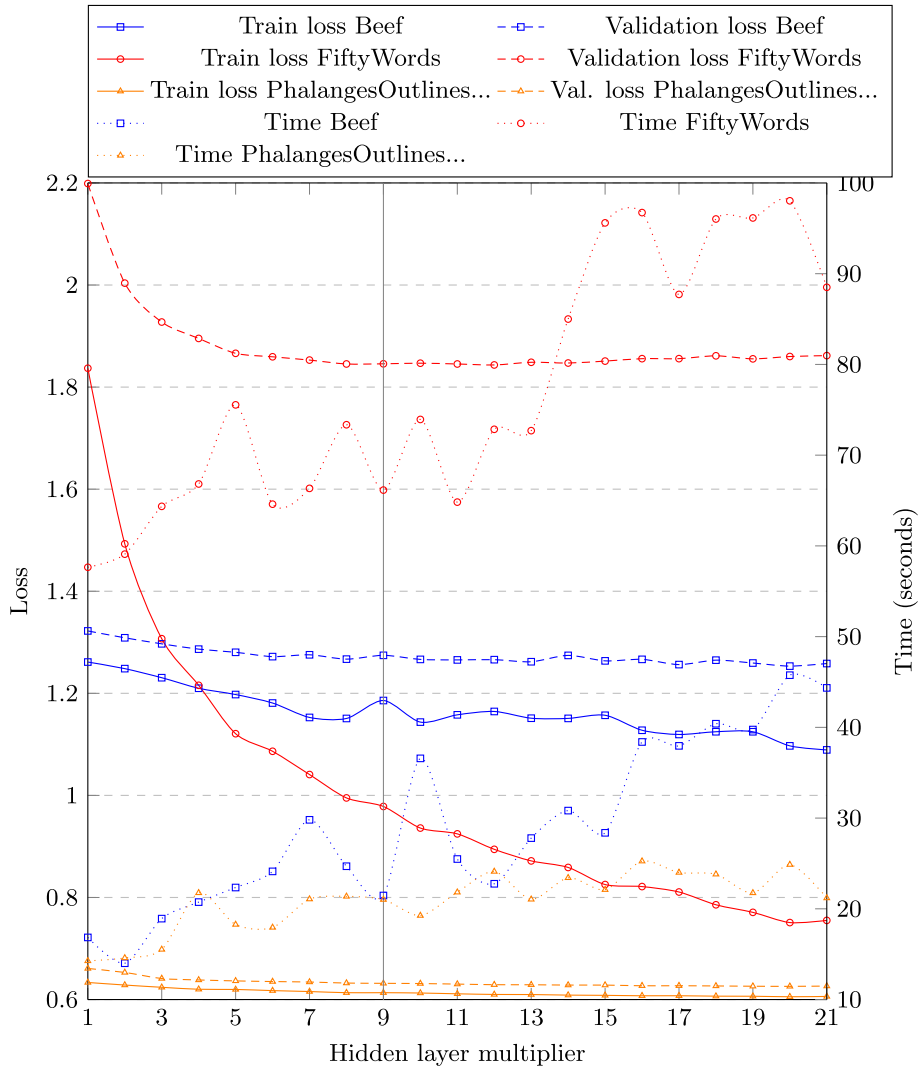

Fig. 4. Training loss, validation loss and time obtained for the three TS with Multiplier values from 1 to 21.

attributed to the random nature of the MLP initialization and the numeric resolution.

Small values of *Divisor* yield a significant loss variation for the *FiftyWords* TS. For instance, a maximum loss of 1.83 is obtained with a *Divisor* of four, while a *Divisor* of five gives a loss of 1.00. The variation decreases when the *Divisor* increases, with a stable value of 17. It is worth emphasizing that the variation shown was due to the MLP randomness. Hence, a *Divisor* value of 24 is appropriate because it is big enough to allow TS with a larger number of training sets to reach its stability while keeping the trade-off between low loss and low time. *PhalangesOutlinesCorrect* showed a smooth loss. The *Divisor* value of 24 is appropriate because it does not adversely affect the loss, and the highest values are already achieved. Moreover, the training time of 17 s is equal to the *Beef* and about one-fourth of *FiftyWords*.

Fig. 4 shows the results of the training loss (solid line), validation loss (dashed line), and time (dotted line). The experiment was carried out with *Multiplier* values ranging from 1 to 21 in steps of one. Unlike the *Divisor* experiment, the validation loss was also included because the training and validation loss curves do not follow the same pattern.

The solid vertical line in Fig. 4 shows the results with the proposed *Multiplier* value of nine. For larger values, the training loss decreases, and the validation loss reaches a steady level, as observed in *FiftyWords* TS. Low validation loss is needed to ensure generalization capability. At this point, the trade-off between validation loss and time is obtained for the three TS. It is also important to highlight that the training time is often increased as the *Multiplier* goes up. Therefore, it is desired to use the lowest *Multiplier* value that produces the smallest loss.

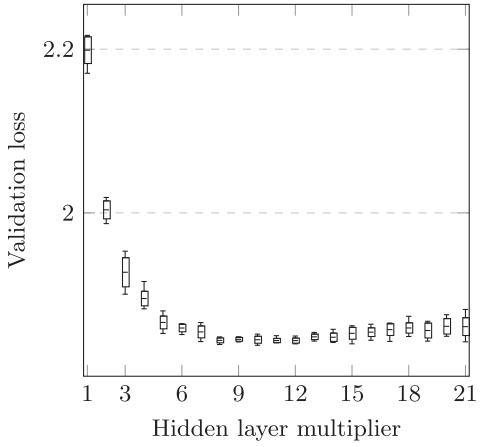

Fig. 5. Box and whisker plot for the validation loss using different values for Multiplier.

The box and whisker plot of Fig. 5 shows that when the *Multiplier*

Table 2
Test results part I: Average accuracy of the classifiers over 61 UCR data sets (Continued in Table 3).

	Reported methods							
Data set	TS-Chief	Rocket	Inception	S-Boss	ResNet	Hive-Cote	Proximity	Boss
			Time			v1.0	Forest	
Adiac	0.780	0.772	**0.822**	0.743	0.815	0.796	0.722	0.749
ArrowHead	0.881	0.859	0.880	**0.888**	0.859	0.876	0.884	0.869
Beef	0.632	0.760	0.682	0.656	0.677	0.736	0.594	0.612
BeetleFly	0.958	0.885	0.893	0.937	0.853	0.963	0.860	0.943
BirdChicken	0.963	0.882	0.952	0.968	0.945	0.940	0.903	**0.983**
Car	0.879	**0.912**	0.901	0.859	0.908	0.869	0.806	0.848
CBF	0.998	0.996	0.996	**0.999**	0.988	0.998	0.994	0.999
ChlorineConcentration	0.661	0.796	**0.864**	0.659	0.841	0.734	0.631	0.658
Coffee	0.990	**1.000**	0.999	0.981	0.996	0.993	0.992	0.986
CricketX	0.830	0.839	**0.853**	0.784	0.808	0.816	0.800	0.762
CricketY	0.817	0.845	**0.860**	0.771	0.810	0.810	0.800	0.750
CricketZ	0.838	0.853	**0.861**	0.787	0.813	0.834	0.803	0.769
DiatomSizeReduction	0.946	0.958	0.951	0.945	0.306	0.914	0.957	0.945
DistalPhalanxOutlineAgeGroup	**0.828**	0.812	0.766	0.821	0.776	0.824	0.802	0.821
DistalPhalanxOutlineCorrect	0.819	0.824	0.815	0.811	0.809	0.824	0.823	0.812
DistalPhalanxTW	0.692	0.701	0.665	0.670	0.667	0.696	0.692	0.671
Earthquakes	0.748	0.748	0.731	0.747	0.717	0.747	**0.750**	0.746
ECG200	0.855	0.899	0.897	0.872	0.884	0.859	0.873	0.878
ECG5000	**0.948**	0.947	0.942	0.941	0.937	0.946	0.940	0.940
ECGFiveDays	0.994	**0.996**	0.996	0.992	0.951	0.994	0.883	0.992
FaceAll	0.983	**0.988**	0.983	0.975	0.982	0.980	0.977	0.970
FaceFour	**1.000**	0.931	0.939	0.982	0.925	0.973	0.945	0.995
FacesUCR	0.973	0.971	**0.977**	0.957	0.964	0.961	0.956	0.951
FiftyWords	**0.843**	0.825	0.827	0.765	0.724	0.772	0.826	0.706
Fish	**0.982**	0.974	0.973	0.971	0.970	0.979	0.934	0.970
GunPoint	**1.000**	0.992	0.995	0.997	0.991	0.998	0.991	0.996
Ham	0.805	**0.855**	0.850	0.835	0.807	0.840	0.783	0.837
Herring	0.597	0.625	0.625	0.608	0.597	0.612	0.574	0.596
InsectWingbeatSound	0.632	**0.657**	0.627	0.519	0.491	0.640	0.607	0.512
ItalyPowerDemand	0.962	0.962	0.960	0.868	0.957	0.958	0.956	0.871
Lightning2	0.769	0.777	0.817	0.808	0.801	0.773	**0.849**	0.819
Lightning7	0.794	0.798	**0.821**	0.681	0.810	0.758	0.792	0.671
Meat	0.984	0.989	0.984	0.984	**0.994**	0.986	0.987	0.981
MedicalImages	0.799	**0.805**	0.796	0.717	0.792	0.740	0.771	0.716
MiddlePhalanxOutlineAgeGroup	0.694	**0.711**	0.594	0.659	0.597	0.698	0.659	0.656
MiddlePhalanxOutlineCorrect	0.806	0.834	0.834	0.807	0.824	0.813	0.824	0.810
MiddlePhalanxTW	0.573	0.590	0.527	0.542	0.531	0.584	0.549	0.532
OliveOil	**0.917**	0.902	0.874	0.874	0.862	0.883	0.879	0.876
OSULeaf	0.974	0.939	0.952	**0.977**	0.975	0.975	0.859	0.969
PhalangesOutlinesCorrect	0.825	0.845	**0.861**	0.819	0.848	0.826	0.829	0.817
Plane	**1.000**	**1.000**	0.997	0.998	**1.000**	**1.000**	**1.000**	0.998
ProximalPhalanxOutlineAgeGroup	0.846	0.852	0.822	0.833	0.817	0.856	0.840	0.828
ProximalPhalanxOutlineCorrect	0.875	0.899	**0.906**	0.866	0.906	0.885	0.866	0.866
ProximalPhalanxTW	0.811	0.804	0.782	0.775	0.789	0.816	0.791	0.769
ShapeletSim	**1.000**	0.998	0.924	**1.000**	0.727	**1.000**	0.789	**1.000**
SonyAIBORobotSurface1	0.890	0.958	0.954	0.895	**0.960**	0.826	0.920	0.898
SonyAIBORobotSurface2	0.901	0.935	0.951	0.884	**0.969**	0.937	0.899	0.879
Strawberry	0.974	**0.979**	0.975	0.966	0.975	0.975	0.960	0.970
SwedishLeaf	0.962	0.963	**0.970**	0.925	0.959	0.949	0.953	0.920
Symbols	**0.971**	0.969	0.970	0.964	0.947	0.969	0.967	0.963
SyntheticControl	**0.999**	0.998	0.996	0.965	0.994	0.994	0.998	0.967
ToeSegmentation1	**0.960**	0.933	0.953	0.920	0.954	0.960	0.836	0.925
ToeSegmentation2	0.963	0.933	0.964	0.963	0.953	**0.968**	0.886	0.962
Trace	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
UWaveGestureLibraryX	0.847	**0.857**	0.834	0.788	0.790	0.833	0.831	0.753
UWaveGestureLibraryY	**0.788**	0.784	0.771	0.715	0.676	0.755	0.767	0.662
UWaveGestureLibraryZ	0.791	**0.796**	0.773	0.736	0.751	0.775	0.767	0.695
Wafer	0.999	0.999	0.999	0.999	0.999	1.000	0.996	0.999
Wine	0.898	0.914	0.887	0.894	0.856	0.892	0.856	0.893
WordSynonyms	**0.794**	0.764	0.752	0.738	0.613	0.693	0.778	0.658
Yoga	0.873	0.914	0.912	**0.915**	0.877	0.912	0.887	0.910
Number of times the method wins	15	13	11	6	5	4	4	3

factor increases, the validation loss decreases. Also, the interval between the maximum and the minimum decreases as the *Multiplier* factor increases, and after it reaches 12 the spread increases again. It can be observed that the *Multiplier* factors of 9 and 11 exhibit the least spread of the quartiles as well as the least loss. However, a *Multiplier* factor of 9 was selected because, according to Eq. (7), a *Multiplier* factor of 11 would increase the number of neurons.

In the literature, underfitting is less discussed than overfitting because this problem can be alleviated by increasing the model's complexity. Contrarily, methods such as cross-validation, feature removal, data augmentation, early stopping, and ensembling have been proposed to prevent overfitting. However, the most widely used method is regularization, which performs slight modifications to the network's weights to find the best generalization (Nusrat & Jang, 2018).

Table 3
Test results part II (Continued from Table 2).

	Reported methods						Proposed methods		
Data set	cBoss	STC	Weasel	Catch22	TSF	Rise	1MLP	2MLP	3MLP
Adiac	0.746	0.793	0.799	0.685	0.712	0.758	0.524	0.691	0.739
ArrowHead	0.878	0.807	0.848	0.750	0.797	0.828	0.809	0.838	0.841
Beef	0.571	0.736	0.740	0.473	0.689	0.742	0.880	**0.900**	0.860
BeetleFly	**0.975**	0.933	0.887	0.840	0.833	0.872	0.860	0.850	0.900
BirdChicken	0.977	0.870	0.865	0.893	0.815	0.868	0.840	0.765	0.770
Car	0.843	0.858	0.834	0.746	0.766	0.753	0.872	0.882	0.873
CBF	0.998	0.985	0.980	0.954	0.972	0.949	0.888	0.891	0.899
ChlorineConcentration	0.665	0.735	0.755	0.598	0.723	0.765	0.487	0.753	0.785
Coffee	0.990	0.989	0.989	0.980	0.987	0.985	**1.000**	**1.000**	**1.000**
CricketX	0.764	0.792	0.776	0.609	0.693	0.706	0.544	0.574	0.588
CricketY	0.751	0.778	0.780	0.591	0.686	0.709	0.601	0.628	0.625
CricketZ	0.772	0.807	0.790	0.628	0.706	0.722	0.561	0.594	0.595
DiatomSizeReduction	0.884	0.859	0.908	0.925	0.942	0.932	0.965	**0.972**	0.966
DistalPhalanxOutlineAgeGroup	0.806	0.796	0.793	0.783	0.809	0.822	0.736	0.767	0.765
DistalPhalanxOutlineCorrect	0.780	**0.827**	0.819	0.812	0.806	0.811	0.746	0.789	0.786
DistalPhalanxTW	0.673	0.690	0.679	0.681	0.691	0.694	0.646	0.684	**0.711**
Earthquakes	0.747	0.742	0.747	0.739	0.747	0.748	0.750	0.743	0.745
ECG200	0.830	0.839	0.859	0.789	0.860	0.851	0.890	0.915	**0.920**
ECG5000	0.943	0.942	0.946	0.936	0.943	0.937	0.936	0.939	0.941
ECGFiveDays	0.984	0.978	0.994	0.816	0.952	0.973	0.958	0.974	0.977
FaceAll	0.969	0.954	0.973	0.811	0.950	0.965	0.837	0.869	0.877
FaceFour	0.997	0.656	0.981	0.680	0.907	0.877	0.847	0.835	0.839
FacesUCR	0.952	0.910	0.956	0.709	0.904	0.892	0.790	0.806	0.810
FiftyWords	0.718	0.737	0.777	0.598	0.722	0.667	0.697	0.711	0.709
Fish	0.974	0.950	0.951	0.773	0.830	0.859	0.892	0.878	0.870
GunPoint	1.000	0.986	0.993	0.943	0.955	0.981	0.908	0.949	0.951
Ham	0.811	0.811	0.821	0.694	0.799	0.820	0.769	0.758	0.750
Herring	0.574	0.633	0.602	0.556	0.604	0.598	**0.728**	0.719	0.722
InsectWingbeatSound	0.539	0.629	0.619	0.559	0.603	0.636	0.654	0.651	0.645
ItalyPowerDemand	0.926	0.954	0.947	0.877	0.959	0.945	0.967	0.973	**0.974**
Lightning2	0.797	0.658	0.627	0.745	0.764	0.682	0.734	0.738	0.725
Lightning7	0.720	0.743	0.713	0.646	0.721	0.698	0.671	0.677	0.660
Meat	0.977	0.968	0.977	0.943	0.984	0.987	0.945	0.947	0.910
MedicalImages	0.690	0.710	0.709	0.757	0.746	0.667	0.632	0.706	0.738
MiddlePhalanxOutlineAgeGroup	0.677	0.668	0.660	0.688	0.660	0.700	0.629	0.646	0.649
MiddlePhalanxOutlineCorrect	0.772	0.832	0.828	0.773	0.800	0.805	0.750	**0.848**	0.846
MiddlePhalanxTW	0.567	0.579	0.554	0.557	0.569	0.585	0.490	0.531	**0.623**
OliveOil	0.874	0.879	0.913	0.746	0.893	0.893	0.850	0.807	0.487
OSULeaf	0.960	0.956	0.852	0.724	0.643	0.654	0.575	0.568	0.583
PhalangesOutlinesCorrect	0.779	0.834	0.822	0.792	0.806	0.813	0.670	0.753	0.802
Plane	**1.000**	0.999	0.995	0.988	0.996	0.997	0.974	0.975	0.981
ProximalPhalanxOutlineAgeGroup	0.850	0.846	0.845	0.858	0.845	0.857	0.855	0.856	**0.863**
ProximalPhalanxOutlineCorrect	0.865	0.895	0.876	0.834	0.849	0.874	0.734	0.869	0.884
ProximalPhalanxTW	0.795	0.808	0.801	0.786	0.802	0.813	0.744	0.777	**0.838**
ShapeletSim	0.985	1.000	0.997	0.994	0.514	0.768	0.534	0.526	0.537
SonyAIBORobotSurface1	0.623	0.801	0.909	0.883	0.864	0.867	0.798	0.825	0.766
SonyAIBORobotSurface2	0.876	0.937	0.935	0.902	0.874	0.912	0.826	0.828	0.839
Strawberry	0.972	0.972	0.979	0.923	0.967	0.973	0.932	0.966	0.968
SwedishLeaf	0.911	0.934	0.958	0.880	0.898	0.923	0.787	0.874	0.890
Symbols	0.963	0.901	0.953	0.948	0.878	0.913	0.884	0.897	0.879
SyntheticControl	0.951	0.992	0.987	0.967	0.992	0.678	0.933	0.971	0.974
ToeSegmentation1	0.952	0.953	0.943	0.813	0.667	0.880	0.618	0.619	0.614
ToeSegmentation2	0.965	0.945	0.928	0.835	0.803	0.912	0.745	0.779	0.775
Trace	**1.000**	**1.000**	**1.000**	1.000	0.992	0.983	0.671	0.846	0.846
UWaveGestureLibraryX	0.774	0.820	0.818	0.769	0.800	0.634	0.741	0.779	0.774
UWaveGestureLibraryY	0.694	0.745	0.726	0.704	0.722	0.668	0.690	0.703	0.698
UWaveGestureLibraryZ	0.711	0.772	0.755	0.706	0.733	0.664	0.686	0.721	0.719
Wafer	0.999	**1.000**	1.000	0.997	0.997	0.995	0.995	0.995	0.996
Wine	0.878	0.886	**0.930**	0.700	0.862	0.871	0.724	0.680	0.667
WordSynonyms	0.668	0.623	0.713	0.544	0.648	0.592	0.587	0.591	0.595
Yoga	0.912	0.880	0.892	0.804	0.866	0.837	0.832	0.857	0.855
Number of times the method wins	3	3	2	0	0	0	2	4	7

A common technique employed for regularization is the dropout. Dropout consists of randomly ignoring some neurons in the layer and retaining the rest with a certain probability that needs to be optimized. The dropout value for each layer was calculated depending on the layer position. By observing the MLP from the input to the output (backward), the first layer has a dropout of 0.3. Then, the dropout was divided by two on each inner layer. In this way, the model with one layer uses a dropout of 0.3. The model with two layers uses a dropout of 0.3 and 0.15 for the last and penultimate layers, respectively. Finally, with three layers, the values used were 0.3, 0.15, and 0.075.

5. Experiments and results

To conduct the experiments, we used 61 TS from the UCR repository. We discarded the other sets due to the length and amount of data, the memory requirements, and the TS that requires more than 15 min to

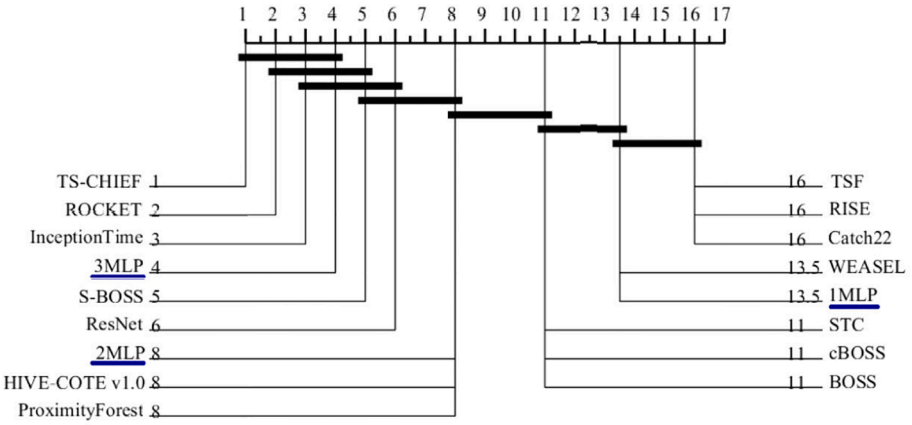

Fig. 6. Average ranges of the number of times won for each method.

train.

The configuration parameters to train the three MLP models are 4500 for the maximum number of epochs and a learning rate of 0.0001. Moreover, we configured a stopping point to prevent the training cycle from being extended to the maximum number of epochs and reduce overfitting. The stopping point occurs every time the loss value is less than 0.0008, with a patience factor of 35.

We added a function to increase the efficiency of the training cycle when the curve reaches a stagnation point. The function reduced the learning rate by 20% when the variation of the accuracy of the evaluation is lower than 0.01. This function is called *Reduce Learning Rate In Plateau*. We also added a waiting period of 9 cycles to avoid excessive adjustments. This occurs before applying the reduction again if the curve remained unchanged.

We used the 61 TS to independently train each of the three proposed models and the validation sets to measure the accuracy. Some validation sets were much larger than the training sets.

We executed ten times the training and validation cycles to obtain the average accuracy. We used the average because it gives more consistent and reproducible values. The difference between distinct cycles for the same series could become significant due to the random initialization of the weights. For example, for the Strawberry series, the highest value was 89.46%, and the lowest was 77.02%.

5.1. Results

A benchmark was conducted between the results obtained with the proposed MLP and 14 state-of-the-art algorithms. The website in Bagnall, Lines, Vickers, and Keogh, 2020 maintains a comprehensive repository for research into TSC, including Critical Difference Diagram (CDD) and the file with a collection of accuracy obtained with 14 algorithms on the UCR data set.

A summary of the accuracy is shown in Tables 2 and 3. The last three columns in Table 3 contain the values obtained by the MLP with one, two, and three layers, respectively. Boldface numbers represent the highest values obtained for each TS.

For the sake of clarity the decimal part of the accuracy was truncated to three digits. However, nine decimal digits were considered to compare maximum values. For example, the accuracy of the *TS-chief* and *InceptionTime* is 0.996 for *ECGFiveDays* in Table 3. However, the nine digit value for *TS-chief* is 0.995973674 and for *InceptionTime* is 0.995857530. Hence, the first is marked in boldface.

In the last row of Tables 2 and 3, we ranked the different methods according to how many times each method wins. The best performers were TS-Chief, Rocket, Inception time, and MLP with three layers. Each method won 15, 13, 11, and 7 times, respectively.

The reason why the proposed model placed fourth can be explained by observing the first three classification methods. TS-Chief uses an ensemble combining some of the best TSC methods. Rocket utilizes a CNN, and Inception Time employs a deep learning model based on the Inception-v4 architecture. These three methods have a more complex structure and, therefore, superior performance than the MLP, which consists of a simpler neural network architecture.

The use of hyperparameters calculated dynamically depending on the number of elements in the training set and the length of the TS allowed the MLP to automatically adjust to different TS without needing manual adjustments, generalizing the model for data sets of different shapes. Fig. 6 shows the average range of all the methods according to the number of times each method obtained the highest accuracy. It should be noted that the proposed methods occupy positions 4, 8, and 13.5. Observe that our simplest method (1MLP) is above STC, cBOSS, and BOSS methods, and our most complex method (3MLP) is just below the top three methods. However, the 3MLP method is computationally less expensive, which is encouraging to justify its use.

A comparison of the accuracy obtained with the MLP concerning the 14 state-of-the-art methods is shown in Table 4. The first column shows the highest value obtained among all the included methods. The following three columns show the difference between the highest accuracy of the reference and the accuracy obtained by the proposed methods. We show the values as percentages, and negative values indicate that the proposed method obtained a lower accuracy.

Typically, the CDD is used to evaluate the algorithms. However, the results may not be clear when a new method is evaluated. For instance, the diagram in Fig. 7 suggests that the proposed three-layer MLP method is ranked among the lower significance methods. However, in Table 3 and in Fig. 6, it is in fourth place. Expedited analysis can lead to a new method being ruled out prematurely.

As an additional way to evaluate the relevance of the proposed method, we suggest grouping by the range of normalized differences between the values obtained by the reference and the proposed methods. The difference can be converted into a percentage using the reference value (the maximum value of accuracy reported among all reference methods) as the value equivalent to 100%; this allows the conversion of the differences to a proportional part.

For example, if the accuracy of 0.30 was obtained with the reference method and 0.25with the proposed method, the difference is -0.05. Then, this difference is converted to a percentage with Eq. 8.

$$Percentage = \frac{Difference * 100\%}{Reference} \tag{8}$$

Therefore, the resulting percentage is -16.66%. On the other hand, if the reference method has an accuracy of 0.80, and the method to evaluate 0.75, the difference is also -0.05 and the percentage is -6.25%. The method to be evaluated, in the second example, has an accuracy

Table 4
A summary of the percentage differences among the proposed and the reference methods.

	Accuracy	Percentage difference		
Data set	max value	1MLP	2MLP	3MLP
Adiac	0.822	−36.285	−15.970	−10.142
ArrowHead	0.888	−8.925	−5.578	−5.278
Beef	0.760	15.789	18.421	13.158
BeetleFly	0.975	−11.795	−12.821	−7.692
BirdChicken	0.983	−14.576	−22.203	−21.695
Car	0.912	−4.388	−3.291	−4.205
CBF	0.999	−11.162	−10.829	−10.006
ChlorineConcentration	0.864	−43.626	−12.797	−9.066
Coffee	1.000	0.000	0.000	0.000
CricketX	0.853	−36.268	−32.692	−31.070
CricketY	0.860	−30.107	−26.968	−27.356
CricketZ	0.861	−34.895	−30.994	−30.904
DiatomSizeReduction	0.958	0.773	1.456	0.841
DistalPhalanxOutlineAgeGroup	0.828	−11.092	−7.385	−7.559
DistalPhalanxOutlineCorrect	0.827	−9.781	−4.613	−4.964
DistalPhalanxTW	0.701	−7.917	−2.428	1.368
Earthquakes	0.750	0.000	−0.864	−0.576
ECG200	0.899	−1.001	1.780	2.336
ECG5000	0.948	−1.270	−0.995	−0.765
ECGFiveDays	0.996	−3.840	−2.196	−1.928
FaceAll	0.988	−15.292	−12.076	−11.243
FaceFour	1.000	−15.246	−16.446	−16.105
FacesUCR	0.977	−19.134	−17.509	−17.075
FiftyWords	0.843	−17.326	−15.683	−15.918
Fish	0.982	−9.121	−10.576	−11.391
GunPoint	1.000	−9.200	−5.067	−4.933
Ham	0.855	−10.134	−11.359	−12.249
Herring	0.633	15.062	13.580	14.074
InsectWingbeatSound	0.657	−0.456	−0.833	−1.802
ItalyPowerDemand	0.962	0.491	1.087	1.198
Lightning2	0.849	−13.458	−13.071	−14.617
Lightning7	0.821	−18.197	−17.529	−19.533
Meat	0.994	−4.919	−4.751	−8.440
MedicalImages	0.805	−21.441	−12.273	−8.384
MiddlePhalanxOutlineAgeGroup	0.711	−11.480	−9.105	−8.648
MiddlePhalanxOutlineCorrect	0.834	−10.144	1.592	1.386
MiddlePhalanxTW	0.590	−16.991	−9.945	5.688
OliveOil	0.917	−7.273	−12.000	−46.909
OSULeaf	0.977	−41.159	−41.836	−40.271
PhalangesOutlinesCorrect	0.861	−22.263	−12.561	−6.932
Plane	1.000	−2.571	−2.476	−1.905
ProximalPhalanxOutlineAgeGroup	0.858	−0.423	−0.265	0.587
ProximalPhalanxOutlineCorrect	0.906	−19.009	−4.146	−2.477
ProximalPhalanxTW	0.816	−8.889	−4.842	2.690
ShapeletSim	1.000	−46.587	−47.389	−46.333
SonyAIBORobotSurface1	0.960	−16.875	−14.137	−20.201
SonyAIBORobotSurface2	0.969	−14.765	−14.495	−13.357
Strawberry	0.979	−4.731	−1.307	−1.141
SwedishLeaf	0.970	−18.891	−9.899	−8.200
Symbols	0.971	−8.972	−7.595	−9.417
SyntheticControl	0.999	−6.607	−2.769	−2.503
ToeSegmentation1	0.960	−35.659	−35.522	−36.024
ToeSegmentation2	0.968	−23.014	−19.518	−19.915
Trace	1.000	−32.900	−15.400	−15.400
UWaveGestureLibraryX	0.857	−13.515	−9.092	−9.740
UWaveGestureLibraryY	0.788	−12.407	−10.848	−11.404
UWaveGestureLibraryZ	0.796	−13.778	−9.386	−9.642
Wafer	1.000	−0.522	−0.483	−0.426
Wine	0.930	−22.163	−26.941	−28.334
WordSynonyms	0.794	−26.061	−25.548	−24.995
Yoga	0.915	−9.036	−6.334	−6.548

closer to the reference than the one obtained in the first example. As it can be observed, negative differences indicated that the method proposed obtained less accuracy than the reference. However, it does not imply that the CDD should not be used. Percentage comparison uses different information and provides a numerical reference on how close the reference result is.

Table 5 was generated from Table 4. The information was grouped into four categories: (i) the number of times the accuracy obtained by the proposed MLP was greater than the best reference method, (ii) the number of times when the accuracy was equal (methods achieved 100% accuracy), (iii) the difference was 1% lower than the best reference method, and (iv) the difference was 5% lower than the best reference method. The information can be considered as a histogram of the percentages.

We used the Friedman and Nemenyi tests to compare multiple classifiers (Demsar, 2006). The aim was to determine if there is any statistically significant difference between the ranks of the compared methods.

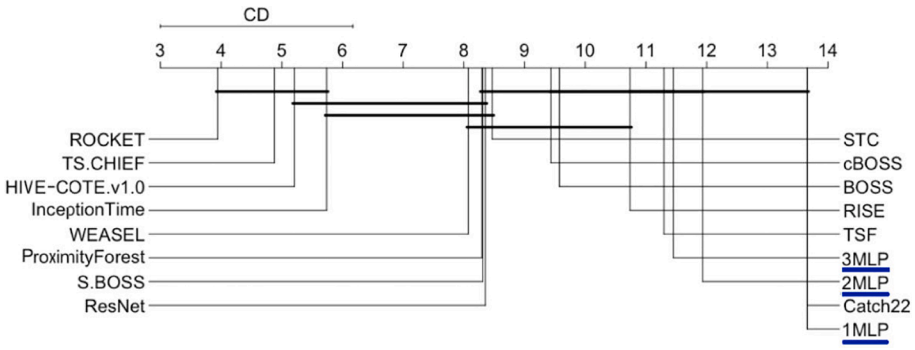

Fig. 7. Average accuracy ranges for the 14 state-of-the-art methods and the three proposed methods.

Table 5
Comparing the accuracy obtained with the proposed MLP and the reference methods by groups.

	1MLP		2MLP		3MLP	
Accuracy	Wins	Accumulated	Wins	Accumulated	Wins	Accumulated
Greater than	3	3	5	5	9	9
Equal to	2	5	1	6	1	10
Less than 1%	19	24	24	30	23	33
Less than 5%	32	56	26	56	23	56

Table 6
Friedman average rankings of the methods (Demsar, 2006).

Method	Ranking
ROCKET	3.9426
TS-CHIEF	4.8770
HIVE-COTE v1.0	5.2049
InceptionTime	5.7377
WEASEL	8.0738
ProximityForest	8.2951
S-BOSS	8.3115
ResNet	8.3525
STC	8.4672
cBOSS	9.4344
BOSS	9.5738
RISE	10.7377
TSF	11.2951
3MLP	11.4508
2MLP	11.9344
Catch22	13.6557
1MLP	13.6557

Friedman's test scores the methods for each data set separately. The best get the first rank, the second-best gets the second rank, and so on. We assigned the average ranges in the event of a tie. On the other hand, we used the Nemenyi test to compare the classifiers and find if the two classifiers' performance is significantly different. We need to know if the average ranges corresponding to both classifiers differ, at least in the critical difference, determined by Eq. (9).

$$CD = q_\alpha \sqrt{\frac{z(z+1)}{6N}} \tag{9}$$

where z corresponds to the number of methods to compare ($z = 17$), N is the total number of TS (61), the critical value q_α corresponding to the test used for the comparison of the methods, is given in (Zar, 2007) (Appendix Table B.15), and is equivalent to 3.562; and αrepresents the confidence interval over which the critical value q_α is selected. The number α commonly used in the analysis done on the reported methods is equal to 0.05. So that, we also selected it for the present study.

We obtained the average rankings of the algorithms to create the diagram depicted in Fig. 7. First, we separated the TS, and we performed a ranking of the methods according to the accuracy values obtained. Then, we obtained the sum and average of each of the method ranges. Finally, we ordered the ranges from highest to lowest. We positioned the highest performances on the left (the first one being the first in the ranking), and we placed the methods with the lowest-performing on the right (the last is the one that obtained the last place in the ranking). Table 6 shows the average rankings of the algorithms.

The horizontal bars in Fig. 7 show the classifiers' pairs where the difference between average ranges is less than the critical difference calculated with Eq. 9. The horizontal bars group the methods in which there is no significant statistical difference among the values. Even though the proposed 3MLP method is placed in the range 11.4508, there is no critical difference against the ProximityForest method positioned in the range 8.2951. This allows us to locate the proposed method competitively among the reported methods.

The statistical comparison of the algorithms carried out by the Friedman test shows a rejection of the null hypothesis, which means that the proposed MLP differs significantly from the highest-rated algorithm (ROCKET).

When the null hypothesis of Friedman's test is rejected, there is a wide variety of multiple comparisons that can be used to determine which treatments differ from each other. The p-values obtained by applying post hoc methods over the Friedman procedure results are presented In Table 7.

- Bonferroni-Dunn's procedure rejects the hypotheses with a p-value ≤ 0.003125.
- Holm's procedure rejects the hypotheses with a p-value ≤ 0.016667.
- Hochberg's procedure rejects the hypotheses with a p-value ≤ 0.0125.
- Hommel's procedure rejects the hypotheses with a p-value ≤ 0.016667.
- Holland's procedure rejects the hypotheses with a p-value ≤ 0.016952.
- Rom's procedure rejects the hypotheses with a p-value ≤ 0.013109.
- Finner's procedure rejects the hypotheses with a p-value ≤ 0.043889.
- Li's procedure rejects the hypotheses with a p-value ≤ 0.036484.

Table 7
Post Hoc comparison for $\alpha = 0.05$ (Friedman).

i	Algorithm	$z = \frac{(R_0 - R_i)}{SE}$	p	Holm Hochberg Hommel	Holland	Rom	Finner	Li
16	1MLP	10.62	< 0.00001	0.003	0.003	0.003	0.003	0.036
15	Catch22	10.62	< 0.00001	0.003	0.003	0.004	0.006	0.036
14	2MLP	8.74	< 0.00001	0.004	0.004	0.004	0.010	0.036
13	3MLP	8.21	< 0.00001	0.004	0.004	0.004	0.013	0.036
12	TSF	8.04	< 0.00001	0.004	0.004	0.004	0.016	0.036
11	RISE	7.43	< 0.00001	0.005	0.005	0.005	0.019	0.036
10	BOSS	6.16	< 0.00001	0.005	0.005	0.005	0.022	0.036
9	cBOSS	6.01	< 0.00001	0.006	0.006	0.006	0.025	0.036
8	STC	4.95	< 0.00001	0.006	0.006	0.007	0.028	0.036
7	ResNet	4.82	< 0.00001	0.007	0.007	0.008	0.031	0.036
6	S-BOSS	4.78	< 0.00001	0.008	0.009	0.009	0.035	0.036
5	ProximityForest	4.76	< 0.00001	0.010	0.010	0.011	0.038	0.036
4	WEASEL	4.52	0.00001	0.013	0.013	0.013	0.041	0.036
3	InceptionTime	1.96	0.04962	0.017	0.017	0.017	0.044	0.036
2	HIVE-COTE v1.0	1.38	0.16743	0.025	0.025	0.025	0.047	0.036
1	TS-CHIEF	1.02	0.30681	0.050	0.050	0.050	0.050	0.050

It is worth noting that even though the statistical data shown in Table 7 makes clear that the MLP's is not the top performer, the diagram in Fig. 7 also shows that the proposed 3MLP method can be commensurable to the highest-ranking algorithms. This can be seen in Fig. 6. According to the number of times the algorithm obtained the highest accuracy value when classifying a TS, our proposed method occupied fourth place in general.

In summary, the proposed three-layer MLP obtained fourth place according to accuracy (it wins ten times). However, in 23 TS our proposal is close to reaching the highest reference value.

At the end of the experimentation stage, we have observed at least three main disadvantages of our proposal. First, when the number of nodes and connections in the hidden layers increases, the memory required grows. Therefore, the training stage cannot be conducted on traditional computers with low memory. This is the reason why we have used only a three-layer MLP. Second, when the MLP competes against complex methods, it can be easily defeated. However, the complex methods can only be implemented in powerful computers. As was shown in our study, a correct hyperparameter tuning makes the MLP a competitive alternative. Third, the MLP, as other artificial neural networks, can fall in local minima. So that, it is needed to work to define methods to overcome this problem.

On the other hand, our proposal has many advantages. First, it does not require deep mathematical knowledge or the use of perplexing algorithms. Second, the model complexity and training time are relatively low compared to complex methods; and third, its simplicity makes it an accessible solution to a TSC problem.

6. Conclusions

In this paper, we proposed an auto-adaptive MLP for TSC on 61 UCR data sets. The hyperparameters for batch size and the number of neurons in the hidden layers are automatically adapted according to the TS nature. Hence, the proposed method solves different TSC tasks. We conducted a benchmark of our proposal against 14 state-of-the-art methods. From the results, we observed that the three-layer MLP ranked in fourth place, even when it is computationally simple. Also, we proposed a different alternative to compare the accuracy among methods.

Although the MLP was not among the three best performers, it is important to highlight the advantages of using our model. It is a simple architecture, easy to implement, accessible, affordable, competitive, and a valid solution to the TSC problem.

In the future, it will be desirable to test the proposed model with a different data set. It will also be interesting to modify the method to handle the TS excluded from the study. Furthermore, it will be desirable to perform comparisons of execution time. Finally, the comparison will include distance-based methods that were excluded in the current experiment.

CRediT authorship contribution statement

Felipe Arias del Campo: Conceptualization, Methodology, Software, Writing - original draft. **María Cristina Guevara Neri:** Conceptualization, Methodology, Formal analysis. **Osslan Osiris Vergara Villegas:** Methodology, Formal analysis, Writing - original draft. **Vianey Guadalupe Cruz Sánchez:** Methodology, Investigation, Visualization. **Humberto Jesús Ochoa Domínguez:** Methodology, Investigation, Writing - review & editing. **Vicente García Jiménez:** Validation, Formal analysis.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Aalst, W., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E., & Günther, C. (2010). Process mining: A two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling, 9*, 87–111.

Abanda, A., Mori, U., & Lozano, J. (2019). A review on distance based time series classification. *Data Mining and Knowledge Discovery, 33*, 378–412.

Ahn, G., & Hur, S. (2020). Efficient genetic algorithm for feature selection for early time series classification. *Computers & Industrial Engineering, 142*, 1–5.

Azami, H., Fernández, A., & Escudero, J. (2017). Refined multiscale fuzzy entropy based on standard deviation for biomedical signal analysis. *Medical & Biological Engineering & Computing, 55*, 2037–2052.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery, 31*, 606–660.

Bagnall, A., Lines, J., Hills, J., & Bostrom, A. (2015). Time-series classification with cote: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering, 27*, 2522–2535.

Bagnall, A., Lines, J., Vickers, W., & Keogh, E. (2020). The UEA & UCR time series classification repository. www.timeseriesclassification.com.

Chang, L., Zhipeng, J., & Yuanjie, Z. (2019). A novel reconstructed training-set svm with roulette cooperative coevolution for financial time series classification. *Expert Systems with Applications, 123*, 283–298.

Chen, W., & Shi, K. (2019). A deep learning framework for time series classification using relative position matrix and convolutional neural network. *Neurocomputing, 359*, 384–394.

Cui, H., & Bai, J. (2019). A new hyperparameters optimization method for convolutional neural networks. *Pattern Recognition Letters, 125*, 828–834.

Dau, H., Keogh, E., Kamgar, K., Yeh, C., Zhu, Y., Gharghabi, S., Ratanamahatana, C., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G., & Hexagon-ML (2018). The UCR time series classification archive. https://www.cs.ucr.edu/eamonn/time_series_data_2018/.

Dempster, A., Petitjean, F., & Webb, G. (2020). Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery, 34*, 1454–1495.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.

Deng, H., Runger, G., Tuv, E., & Vladimir, M. (2013). A time series forest for classification and feature extraction. *Information Sciences, 239*, 142–153.

Devarakonda, A., Naumov, M., & Garland, M. (2017). Adabatch: Adaptive batch sizes for training deep neural networks. CoRR, abs/1712.02029. http://arxiv.org/abs/1712.02029. arXiv:1712.02029.

Dostal, L., Grossert, H., Duecker, D., Grube, M., Kreuter, D., Sandmann, K., Zillman, B., & Seifried, R. (2020). Predictability of vibration loads from experimental data by means of reduced vehicle models and machine learning. *IEEE Access, 8*, 177180–177194.

Farooq, S., Ahmad, J., Tahir, A., Awais, M., Chen, C., Irfan, M., Ayesha, H., Bakar, A., Long, X., Yin, B., Akbarzadeh, S., Lu, C., Wang, L., & Chen, W. (2020). EEG-based neonatal sleep-wake classification using multilayer perceptron neural network. *IEEE Access, 8*, 183025–183034.

Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. (2019). Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery, 33*, 917–963.

Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D., Weber, J., Webb, G., Idoumghar, L., Muller, P., & Petitjean, F. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery, 34*, 1936–1962.

Flynn, M., Large, J., & Bagnall, T. (2019). The contract random interval spectral ensemble (c-RISE): The effect of contracting a classifier on accuracy. In *Proc. of the International Conference on Hybrid Artificial Intelligence Systems* (pp. 381–392). Springer.

Garro, B., Sossa, H., & Vazquez, R. (2009). Design of artificial neural networks using a modified particle swarm optimization algorithm. In *Proc. of the 2009 International Joint Conference on Neural Networks* (pp. 938–945). IEEE.

Garro, B., Sossa, H., & Vazquez, R. (2011). Artificial neural network synthesis by means of artificial bee colony (abc) algorithm. In *Proc. of the IEEE Congress of Evolutionary Computation* (pp. 331–338). IEEE.

Guliyev, N., & Smailov, V. (2018). On the approximation by single hidden layer feedforward neural networks with fixed weights. *Neural Networks, 98*, 296–304.

Hemmati, A., Nait, M., Reza, M., Dai, Z., & Zhang, X. (2020). Modeling co2 solubility in water at high pressure and temperature conditions. *Energy & Fuels, 34*, 4761–4776.

Hemmati, A., Varamesh, A., Husein, M., & Karan, K. (2018). On the evaluation of the viscosity of nanofluid systems: Modeling and data assessment. *Renewable and Sustainable Energy Reviews, 81*, 313–329.

Hesami, M., Naderi, R., Tohidfar, M., & Yoosefzadeh-Najafabadi, M. (2020). Development of support vector machine-based model and comparative analysis with artificial neural network for modeling the plant tissue culture procedures: effect of plant growth regulators on somatic embryogenesis of chrysanthemum, as a case study. *Plant Methods, 16*, 1–15.

Hosseinzadeh, M., Hassan, O., Yassin, M., Safara, F., Kamaran, H., Ali, S., Vo, B., & Chiang, H. (2021). A multiple multilayer perceptron neural network with an adaptive learning algorithm for thyroid disease diagnosis in the internet of medical things. *The Journal of Supercomputing, 77*, 3616–3637.

Hu, M., Ji, Z., Yan, K., Guo, Y., Feng, X., Gong, J., Zhao, X., & Dong, L. (2018). Detecting anomalies in time series data via a meta-feature based approach. *IEEE Access, 6*, 27760–27776.

Jebb, A., Tay, L., Wang, W., & Huang, Q. (2015). Time series analysis for psychological research: examining and forecasting change. *Frontiers in Psychology, 6*, 1–24.

Kandel, I., & Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express, 6*, 312–315.

Kenji, B., & Uchida, S. (2020). Time series classification using local distance-based features in multi-modal fusion networks. *Pattern Recognition, 97*, 1–12.

Kucuk, D., & Avdan, U. (2020). Optimization-based automated unsupervised classification method: A novel approach. *Expert Systems with Applications, 160*, 1–15.

Lahreche, A., & Boucheham, B. (2021). A fast and accurate similarity measure for long time series classification based on local extrema and dynamic time warping. *Expert Systems with Applications, 168*, 1–12.

Large, J., Bagnall, A., Malinowski, S., & Tavenard, R. (2019). On time series classification with dictionary-based classifiers. *Intelligent Data Analysis, 23*, 1073–1089.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*, 436–444.

Li, Q., Xiong, Q., Ji, S., Yu, Y., Wu, C., & Yi, H. (2021). A method for mixed data classification base on rbf-elm network. *Neurocomputing, 431*, 7–22.

Li, Y., Tang, G., Du, J., Zhou, N., Zhao, Y., & Wu, T. (2019). Multilayer perceptron method to estimate real-world fuel consumption rate of light duty vehicles. *IEEE Access, 7*, 63395–63402.

Liashchynskyi, P., & Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: A big comparison for NAS. CoRR, abs/1912.06059. http://arxiv.org/abs/1912.06059. arXiv:1912.06059.

Lines, J., Taylor, S., & Bagnall, A. (2018). Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data, 12*, 1–35.

Liu, C., Hsaio, W., & Tu, Y. (2019). Time series classification with multivariate convolutional neural network. *IEEE Transactions on Industrial Electronics, 66*, 4788–4797.

Lubba, C., Sethi, S., Knaute, P., Schultz, S., Fulcher, B., & Jones, N. (2019). catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery, 33*, 1821–1852.

Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., Petitjean, F., & Webb, G. (2019). Proximity forest: An effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery, 33*, 607–635.

Mahdaviara, M., Nait, M., Hemmati, A., Dai, Z., Zhang, C., Xiao, T., & Zhang, X. (2021). Toward smart schemes for modeling CO_2 solubility in crude oil: Application to carbon dioxide enhanced oil recovery. *Fuel, 285*, 1–16.

Middlehurst, M., Vickers, W., & Bagnall, A. (2019). Scalable dictionary classifiers for time series classification. In *Proc. of the International Conference on Intelligent Data Engineering and Automated Learning* (pp. 11–19). Springer.

Nait, M. (2020). Modeling solubility of sulfur in pure hydrogen sulfide and sour gas mixtures using rigorous machine learning methods. *International Journal of Hydrogen Energy, 45*, 33274–33287.

Nait, M., Abdelfetah, M., & Ouaer, H. (2021). On the evaluation of solubility of hydrogen sulfide in ionic liquids using advanced committee machine intelligent systems. *Journal of the Taiwan Institute of Chemical Engineers, 118*, 159–168.

Nait, M., Jahanbani, A., & Zeraibi, N. (2020). Predicting thermal conductivity of carbon dioxide using group of data-driven models. *Journal of the Taiwan Institute of Chemical Engineers, 113*, 165–177.

Nusrat, I., & Jang, S. (2018). A comparison of regularization techniques in deep neural networks. *Symmetry, 10*, 1–18.

Pontes, F., Amorim, G., Balestrassi, P., Paiva, A., & Ferreira, J. (2016). Design of experiments and focused grid search for neural network parameter optimization. *Neurocomputing, 186*, 22–34.

Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation, 29*, 2352–2449.

Schäfer, P. (2015). The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery, 29*, 1505–1530.

Schäfer, P., & Leser, U. (2017). Fast and accurate time series classification with weasel. In *Proc. of the 2017 ACM on Conference on Information and Knowledge Management* (pp. 637–646). ACM.

Schäfer, P., & Leser, U. (2020). Teaser: Early and accurate time series classification. *Data Mining and Knowledge Discovery, 34*, 1336–1362.

Seo, H., & Cho, D. (2020). Cancer-related gene signature selection based on boosted regression for multilayer perceptron. *IEEE Access, 8*, 64992–65004.

Shifaz, A., Pelletier, C., Petitjean, F., & Webb, G. (2020). Ts-chief: A scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery, 34*, 1–34.

Siegel, C., Daily, J., & Vishnu, A. (2010). Adaptive neuron apoptosis for accelerating deep learning on large scale systems. In *Proc. of the 2016 IEEE International Conference on Big Data* (pp. 753–762). IEEE.

Simoes, L., Parquet, F., & Parquet, J. (2020). Detection of liner surface defects in solid rocket motors using multilayer perceptron neural networks. *Polymer Testing, 88*, 1–13.

Singh, S., Singh, K., Singh, S., Kaur, J., Peshoria, S., & Kumar, J. (2020). Study of ARIMA and least square support vector machine (LS-SVM) models for the prediction of sars-cov-2 confirmed cases in the most affected countries. *Chaos, Solitons & Fractals, 139*, 1–9.

Singh, V., Chandra, R., Kumar, S., Dass, P., & Bhatnagar, P. A. V. (2020). Prediction of covid-19 corona virus pandemic based on time series data using support vector machine. *Journal of Discrete Mathematical Sciences and Cryptography, 23*, 1–15.

Soares, E., Jr., Costa, P. C. B., & Leite, D. (2018). Ensemble of evolving data clouds and fuzzy models for weather time series prediction. *Applied Soft Computing, 64*, 445–453.

Sánchez, L., Rodríguez, J., Salazar, S., Avecilla, G., & Pérez, G. (2020). A high-accuracy mathematical morphology and multilayer perceptron-based approach for melanoma detection. *Applied Sciences, 10*, 1–17.

Tang, W., Long, G., Liu, L., Zhou, T., Jiang, J., & Blumenstein, M. (2020). Rethinking 1d-cnn for time series classification: A stronger baseline. arXiv preprint arXiv: 2002.10061.

Theckel, T., Rana, S., Gupta, S., & Venkatesh, S. (2020). Fast hyperparameter tuning using bayesian optimization with directional derivatives. *Knowledge-Based Systems, 205*, 1–8.

Uddin, S., Khan, A., Hossain, M., & Ali, M. (2019). Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making, 19*, 1–16.

Wei, C., Petitjean, F., & Webb, G. (2020). Fastee: Fast ensembles of elastic distances for time series classification. *Data Mining and Knowledge Discovery, 34*, 231–272.

Wu, J., Chen, S., & Liu, X. (2020). Efficient hyperparameter optimization through model-based reinforcement learning. *Neurocomputing, 409*, 381–393.

Xiaowu, Z., Zidong, W., Qi, L., & Weiguo, S. (2019). Integration of residual network and convolutional neural network along with various activation functions and global pooling for time series classification. *Neurocomputing, 367*, 39–45.

Yoo, Y. (2019). Hyperparameter optimization of deep neural network using univariate dynamic encoding algorithm for searches. *Knowledge-Based Systems, 178*, 74–83.

Zanaty, E. (2012). Support vector machines (SVMs) versus multilayer perception (MLP) in data classification. *Egyptian Informatics Journal, 13*, 177–183.

Zar, J. (2007). *Biostatistical analysis* (1st ed.). USA: Prentice-Hall Inc.

Zhao, H., Pan, Z., & Tao, W. (2020). Regularized shapelet learning for scalable time series classification. *Computer Networks, 173*, 1–12.