Personal and Ubiquitous Computing (2023) 27:177–191
https://doi.org/10.1007/s00779-020-01447-7

ORIGINAL ARTICLE

End-to-end multivariate time series classification via hybrid deep learning architectures

Mehak Khan [1] · Hongzhi Wang [1] · Alladoumbaye Ngueilbaye [1] · Aya Elfatyany [1]

Received: 30 October 2019 / Accepted: 22 August 2020 / Published online: 11 September 2020

Abstract

Deep learning has revolutionized many areas, including time series data mining. Multivariate time series classification (MTSC) remained to be a well-known problem in the time series data mining community, due to its availability in various practical applications such as healthcare, finance, geoscience, and bioinformatics. Recently, multivariate long short-term memory with fully convolutional network (MLSTM-FCN) and multivariate attention long short-term memory with fully convolutional network (MALSTM-FCN) have shown superior results over various state-of-the-art methods. So, in this paper, we explore the usage of recurrent neural network (RNN), and its variants, such as bidirectional recurrent neural network (BiRNN), bidirectional long short-term memory (BiLSTM), gated recurrent unit (GRU), and bidirectional gated recurrent unit (BiGRU). We augment these RNN variants separately by replacing long short-term memory (LSTM) in MLSTM-FCN, which is the combination of LSTM, squeeze-and-excitation (SE) block, and fully convolutional network (FCN). Moreover, we integrate the SE block within FCN to leverage its high performance for the MTSC task. The resulting algorithms do not require heavy pre-processing or feature crafting. Thus, they could be easily deployed on real-time systems. We conduct a comprehensive evaluation with a large number of standard datasets and demonstrate that our approaches achieve notable results over the current best MTSC approach.

Keywords Convolutional neural network · Squeeze-and-excitation · Recurrent neural networks · Long short-term memory · Gated recurrent unit · Multivariate time series classification

1 Introduction

Time series classification (TSC) has received much attentiveness in data mining, in which the goal is to classify the data points over time based on its behavior [1]. Time series data is ubiquitous and used in statistics, finance, weather forecasting, pattern recognition, econometrics, astronomy, earthquake prediction, signal processing, and others. In short, practically any field which includes temporal measurements [2]. A time series dataset could be univariate, in which a single observation recorded sequentially over the equal time interval, whereas multivariate, where multiple time series observations are available simultaneously. The complexity of MTSC is increased due to the data type. The MTSC data comprises of interactions between multiple values at the single timestamp.

The problem of MTSC is an open challenge. To solve this issue, numerous research methods have been proposed for improving classification performance in practical environments. The most basic approach is principal component analysis (PCA), which merges all the dimensions of multivariate time series (MTS) to obtain a univariate time series (UTS) [3]. In traditional methods, the naive logistic model (NL) and fisher kernel learning (FKL) showed superior performance; NL predicts classes by sum-up the inner products among feature vectors and model weights over time. The FKL is best to use for TSC when it is based on Hidden Markov models (HMM). FKL also trains a linear support vector machine (SVM) for features or representation to make a prediction [4, 5].

✉ Mehak Khan
mehakkhan@hit.edu.cn

Hongzhi Wang
wangzh@hit.edu.cn

Alladoumbaye Ngueilbaye
angueilbaye@hit.edu.cn

Aya Elfatyany
ayaelfatyany@hit.edu.cn

[1] School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

The distance-based techniques using k-nearest neighbors (k-NN) have also proven to be successful for MTSC; dynamic time warping (DTW) along with k-NN is the best distance-based method [6]. Among feature-based methods, hidden-unit logistic model (HULM) and hidden-state conditional random field (HCRF) and [7] are the state-of-the-art methods on many benchmark datasets.

Another well-known algorithm for MTSC is a symbolic representation for MTS (SMTS) [8], which uses the random forest to partition MTS into leaf nodes; each leaf node is denoted by a word of the codebook. There are also some shapelet-based methods proposed for the MTSC problem, one of them is ultra-fast shapelets (UFS) [9]; first, it extracts random shapelets and then uses a linear SVM or a random forest as a classification method. The autoregressive forests for MTS modeling (mvARF) kernel [10] propose a tree ensemble that is trained on autoregressive models, with different time lags. Word extraction for time series classification-multivariate unsupervised symbols and derivatives (WEASEL-MUSE) [3] is another MTSC approach that builds multivariate feature vector sizes using a bag of patterns to capture discrete features by using various sliding window sizes and known as one of the best shapelet-based approaches.

Deep learning has revolutionized many areas, including MTSC. Some well-known methods are multi-channel deep convolutional neural network (MCDCNN) [11]; this model first learns features from each channel of UTS and then combines information collected from all channels as feature representation and applied into a multilayer perceptron (MLP) to perform classification. Recently, Karim et al. proposed two deep learning frameworks: the MLSTM-FCN and MALSTM-FCN; these methods are based on LSTM, FCN, attention mechanism, and SE block. Their methods are the present state-of-the-art methods for MTSC [12].

In this paper, we propose new hybrid deep learning models that depict comparable performance with existing methods. The proposed models do not require substantial data pre-processing or feature crafting. We tested the proposed models on 35 datasets from diverse domains. The performance metrics are classification testing error loss and f1 score obtained on a particular dataset.

1.1 Contributions

Our main contributions in this paper are as follows:

1. We propose novel hybrid deep learning frameworks that enable efficient MTSC. In this study, we exploit RNN and its variants in hybrid models. This study also shows the comparison of LSTM with RNN, BiRNN, BiLSTM, GRU, and BiGRU, in hybrid deep learning frameworks for various MTSC tasks such as action recognition, activity recognition, ECG/EEG classification, robot failure recognition, and speech recognition.
2. We further investigate the performance of the SE block when integrated within FCN (SE-FCN).
3. The proposed models are end-to-end and do not require heavy data pre-processing or feature crafting. Thus, they could be easily deployed on real-time systems. We validate the effectiveness of our proposed approaches by conducting a comprehensive evaluation on 35 datasets from diverse domains. The results demonstrate that our approaches achieve remarkable results over the present best MTSC approach.

The rest of the paper is organized as follows: Section 2 presents proposed work with their background components, Section 3 briefly explains experiments and results, and we conclude our work and points in Section 4.

2 Methodology

Recently, MLSTM-FCN and MALSTM-FCN [12] have shown state-of-the-art performance in classifying MTSC. To evaluate the effectiveness of different variants of RNN on hybrid MLSTM-FCN deep learning architecture, we propose five new deep learning end-to-end frameworks by replacing the LSTM unit to RNN/BiRNN/BiLSTM/GRU/BiGRU, named as MRNN-FCN, MBiRNN-FCN, MBiLSTM-FCN, MGRU-FCN, and MBiGRU-FCN. Additionally, we investigate the performance of the SE block separately when integrating it within FCN (SE-FCN) for MTSC. For a fair comparison, the architecture of the newly proposed frameworks is kept similar to existing state-of-the-art methods: MLSTM-FCN and MALSTM-FCN. In this section, we first formulate the MTSC problem and then discuss the details of the proposed models with their background components.

2.1 Problem formulation

Definition 1 A multivariate time series $X = (x_1, x_2, \ldots., x_M)$, where $x_M \in \mathbb{R}^n$ and n is the variable dimension.

Definition 2 A dataset $D = \{(a_1, b_1), (a_2, b_2), \ldots\ldots, (a_\mathrm{N}, b_\mathrm{N})\}$ is a group of pairs $(A_\mathrm{i}, B_\mathrm{i})$, where A_i is a multivariate time series with B_i to its corresponding one hot label vector.

Definition 3 The MTSC problem is defined as follows: given a set of classes K, a training data τ of multivariate time series X_i associated with their class labels $y(X_\mathrm{i}) \in \mathrm{K}$, i.e., $D = \{(a_1, b_1), (a_2, b_2), \ldots\ldots, (a_\mathrm{N}, b_\mathrm{N})\}$ the goal is to find a function f, which is a classifier or model, so that $f(X) = y(X)$ and for multivariate time series $X \notin \tau$.

2.2 SE-FCN

We propose a new hybrid deep learning model by integrating the SE block within FCN (SE-FCN) to leverage its high performance for the MTSC problem. The proposed architecture consists of FCN and SE block, which is represented in Fig. 1.

2.2.1 Fully convolutional neural network

FCN was initially presented by Wang et al. [13] for time series classification. FCNs are mostly applied in the temporal domain and has ended up to be useful for dealing with the temporal dimension for TSC without any immense data pre-processing and feature engineering. The FCN is the convolutional part of the model used as a feature extractor in this model.

For MTSC, FCN is described as follows:

$$t = w \odot x + b \tag{1}$$

$$a = BN(t) \tag{2}$$

$$y = \mathrm{Re}LU(a) \tag{3}$$

The architecture consists of temporal convolutional blocks that use as a feature extractor. These temporal convolutional blocks are based on three convolutional 1D kernels with the sizes (8, 5, 3) without striding. Each layer is followed by a batch normalization [14] to improve the speed, performance, and stability of the model and a rectified linear unit (ReLU) activation layer [15] to fix vanishing gradient problem. We build the final network by stacking three convolutional blocks with the filter sizes of 128, 256, and 128, respectively. After the convolutional blocks, the features feed into a global average pooling [16] layer, and the final label is produced from a SoftMax layer.

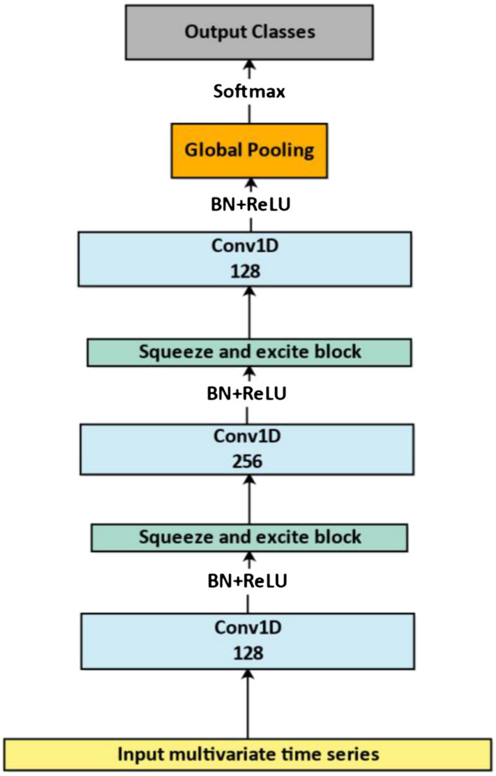


Fig. 1 SE block with FCN (SE-FCN)

2.2.2 Squeeze-and-excitation block

A SE block was introduced by Hu et al. [17] as an architectural component that can easily be integrated into any type of Convolutional Neural Network (CNN). Therefore, to leverage the usage of the SE block and to validate its efficiency over various MTSC datasets, we integrated it within FCN. The SE block features the spatial dependency to learn a channel-specific descriptor by global average pooling that later used to recalibrate the feature maps to highlight essential channels.

$$u_c = v_c * X = \sum_{s=1}^{C'} v_c^s * x^s. \tag{4}$$

A SE block works as a computational unit made upon a transformation F_{tr}, map on an input feature $X \in \mathbb{R}^{W' \times H' \times C'}$ to generate output feature map $U \in \mathbb{R}^{W \times H \times C}$. The output can be written as $U = [\mathrm{u}_1, \mathrm{u}_2, \ldots., \mathrm{u}_C]$.

Here $*$ is the convolution operator, $V_c = \left[v_c^1, v_c^2, \ldots., v_c^{C'}\right]$, $X = \left[\mathrm{x}^1, \mathrm{x}^2, \ldots., \mathrm{x}^{C'}\right]$, and $U \in \mathbb{R}^{W \times H}$. A 2D spatial kernel v_c^s represents a single channel v_c that acts as a corresponding channel X. The basic phenomena of the SE block can be explained in two steps: (1) squeeze and (2) excitation.

The squeeze operation is proposed to squeeze global spatial information into a channel-specific descriptor by using global average pooling in channel-wise statistics. For time series data [12], the transformation output, U, can be shrunk through spatial dimension T for the computation of channel-wise statistics, $z \in \mathbb{R}^C$, and then, the cth element of z is calculated as follows:

$$z_c = F_{sq}(\mathrm{u}_c) = \frac{1}{T} \sum_{t=1}^{T} \mathrm{u}_c(\mathrm{t}). \tag{5}$$

The excitation operation makes the full use of aggregated information from squeeze operation by fully capturing channel-wise dependencies and to achieve that the function must be flexible to learn a nonlinear and non-mutually exclusive relationship between multiple channels. To attain this, a single-gating mechanism with a sigmoid activation is used:

$$s = F_{ex}(z, W) = \sigma(g(z, W)) = \sigma(W_1 \delta(W_1, z)), \tag{6}$$

where δ refers to the ReLU activation function, $W_1 \in \mathbb{R}^{\frac{C}{r} \times C}$ and $W_2 \in \mathbb{R}^{\frac{C}{r} \times C}$. W^1 and $W_2$are used to optimize model complexity and help with the generalization. F_{ex} is a neural network, σ is the sigmoid function, and r is the reduction ratio.

$$\widetilde{x}_c = F_{scale}(u_c, s_c) = s_c \cdot u_c, \qquad (7)$$

Finally, the output of the SE block is obtained after rescaling U with the activation s:where $\widetilde{X} = [\widetilde{x}_1, \widetilde{x}_2, \ldots\ldots, \widetilde{x}_c$] and F_{scale} (u_{c}, s_{c}) refer to the channel-wise multiplication among the feature map $u_c \in \mathbb{R}^T$ and the scalar s_c.

2.2.3 Integration of SE block within FCN

We propose the integration of the SE block within FCN. This model has never been applied to the MTSC problem. The proposed model's architecture consists of FCN and SE block.

The input to the FCN block is a multivariate time series dataset with Q time steps and M variables per time step. The FCN is responsible for feature extraction and contains three temporal convolutional blocks. These temporal convolutional blocks are based on convolutional 1D kernels with sizes (8, 5, 3) and filter sizes of 128, 256, and 128, respectively. The uniform initializer [18] was used to initialize weights in all three convolutional kernels. After each of the first two convolutional blocks, the SE block is integrated. The reduction ratio is denoted as r which is set to 16 for SE blocks. Every layer is followed by batch normalization and the ReLU activation layer; then, the last FCN block is followed by a global average pooling layer. Finally, the output is produced from a SoftMax layer. A diagram illustrating the architecture of SE-FCN is shown in Fig. 1.

The SE block is integrated into FCN and recalibrates the feature maps to highlight essential channels. The additional parameters introduced due to the integration of the SE block into FCN can increase the size of the model. The total number of parameters can be computed as:

$$\frac{2}{r} \sum_{s=1}^{S} N_s . {C_s}^2, \qquad (8)$$

where r is the reduction ratio, S denotes the number of stages, which is the collection of blocks operating on a common spatial dimension, C_{s} refers to the dimension of the output channel in stage s, and N_{s} depicts the number of repeated blocks for the stage s. Due to the fact that FCN blocks are kept constant for all the models, we can easily compute the additional parameters as $\frac{2}{16} * \left\{ (128)^2 + (256)^2 \right\} = 10240$ for the SE-FCN.

The basic idea behind proposing this model is the SE block is useful in recalibrating feature maps as a whole and suppresses the less informative ones; besides, FCN is already proven better for time series classification as a baseline [13]; therefore, infusion, they can show comparable performance for various MTSC tasks.

2.3 RNN-based multivariate hybrid deep learning architectures

We propose new hybrid models by using different variants of RNN. The proposed model architecture consists of FCN, RNN/BiRNN/BiLSTM/GRU/BiGRU, and SE-FCN, respectively.

2.3.1 RNN and its variant

The RNN is a class of neural networks that connect units in the form of the directed cycle; this nature allows it to work with time series data. RNN has a major downside, called vanishing gradient problem, which prevents it from being accurate. RNN is expanded by the integration of edges that extend adjacent time steps, acquaint with a notion of time to the model, as depicted in Fig. 2a.

As stated by Lipton et al. [19], two equations determine all estimations essential for the computation of simple recurrent neural network at each time step on the forward pass, formulated as:

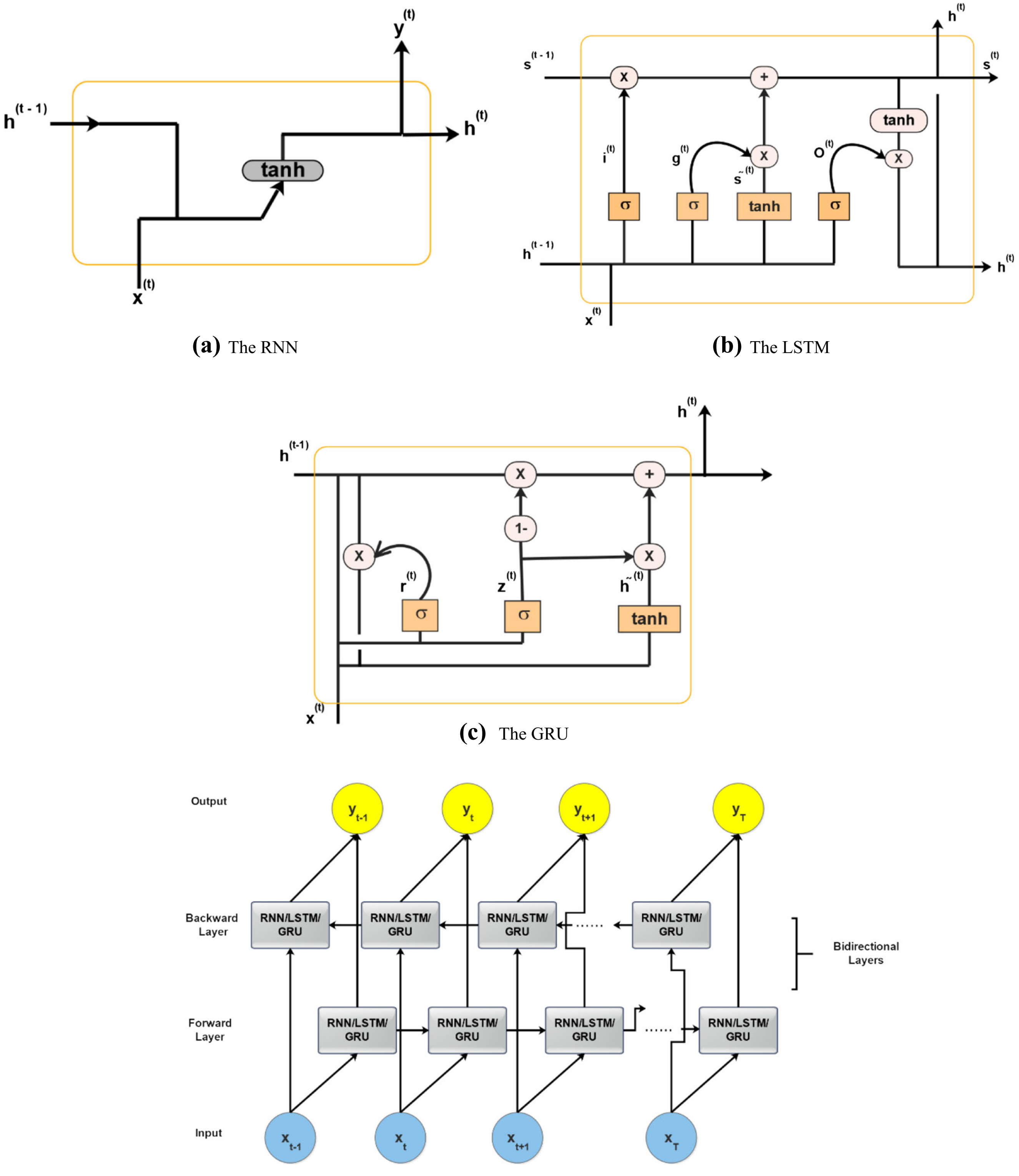
$$h^{(t)} = \sigma\left(W^{hx} x^{(t)} + W^{hh} h^{(t-1)} + b^h\right) \qquad (9)$$

$$\widehat{y}^{(t)} = \text{softmax}\left(W^{yx} h^{(t)} + b_y\right) \qquad (10)$$

where $x^{(t)}$ and $h^{(t-1)}$ are data values and recurrent hidden state at time step t. The output $\widehat{y}^{(t)}$ to each time t is calculated by the given hidden node values $h^{(t)}$ at time step t. Input $x^{(t)}$ at the time step $t-1$ can have an effect on the output $\widehat{y}^{(t)}$ at time step t and later by method for the recurrent connections. W^{hx} is the coefficient matrices of predictable weights between the input layer and the hidden layer and W^{hh} is the matrix of recurrent weights between the hidden layer and itself at adjacent time steps. The b_h and b_y are hidden later bias vectors that alter every node to find out an offset.

BiRNN was first introduced by [20] to present a structure that unfolds to become a bidirectional deep neural network. When it is applied to time series data, not solely the information can be passed following the natural temporal sequences, but additional information can also reversely provide information to previous time steps. BiRNN comprises of two hidden layers; both hidden layers are connected to input and output, as illustrated in Fig. 2d. These layers are differentiated in the way that the first has recurrent connections from past time step, while the second is flipped, passing activation backward on the sequence. BiRNN can be trained by regular backpropagation after unfolding across time. The subsequent three equations describe a BiRNN [19] as follows:

$$h^{(t)} = \sigma\left(W^{hx} x^{(t)} + W^{hh} h^{(t-1)} + b_h\right) \qquad (11)$$



(a) The RNN

(b) The LSTM

(c) The GRU

(d) The Bidirectional RNN/LSTM/GRU

Fig. 2 **a** The RNN. **b** The LSTM. **c** The GRU. **d** The bidirectional RNN/LSTM/GRU

$$z^{(t)} = \sigma\left(W^{zx}x^{(t)} + W^{zz}z^{(t+1)} + b_z\right) \tag{12}$$

$$\widehat{y}^{(t)} = \text{softmax}\left(W^{yx}h^{(t)} + W^{yx}z^{(t)} + b_y\right) \tag{13}$$

where $h^{(t)}$ and $z^{(t)}$ are the values of the hidden layers in the forward and backward directions, respectively.

Hochreiter and Schmidhuber introduced the LSTM primarily to overcome the vanishing gradient problem [21]. LSTM is a variant of RNN that has an identical type of input and output, as shown in Fig. 2b. However, in distinction to RNN, LSTM has an input gate, a forget gate, and an output gate. Therefore, it can control what has to be kept and what has to be forgotten. That is why LSTM can capture much longer range dependencies, whereas RNN cannot [22].

Put formally, computation within the LSTM model yield in keeping with the subsequent calculations that are performed at each time step. These calculations offer the complete algorithm for a modern LSTM with forget gates [19]:

$$g^{(t)} = \tanh\left(\mathrm{W}^{\mathrm{gx}}\mathrm{x}^{(\mathrm{t})} + \mathrm{W}^{\mathrm{gh}}\mathrm{h}^{(\mathrm{t-1})} + \mathrm{b}_{\mathrm{g}}\right) \tag{14}$$

$$i^{(t)} = \sigma\left(\mathrm{W}^{\mathrm{ix}}\mathrm{x}^{(\mathrm{t})} + \mathrm{W}^{\mathrm{ih}}\mathrm{h}^{(\mathrm{t-1})} + \mathrm{b}_{i}\right) \tag{15}$$

$$f^{(t)} = \sigma\left(\mathrm{W}^{\mathrm{fx}}\mathrm{x}^{(\mathrm{t})} + \mathrm{W}^{\mathrm{fh}}\mathrm{h}^{(\mathrm{t-1})} + \mathrm{b}_{f}\right) \tag{16}$$

$$o^{(t)} = \sigma\left(\mathrm{W}^{\mathrm{ox}}\mathrm{x}^{(\mathrm{t})} + \mathrm{W}^{\mathrm{oh}}\mathrm{h}^{(\mathrm{t-1})} + \mathrm{b}_{o}\right) \tag{17}$$

$$s^{(t)} = g^{(t)} \odot i^{(\mathrm{i})} + s^{(t-1)} \odot f^{(t)} \tag{18}$$

$$\mathrm{h}^{(\mathrm{t})} = \tanh\left(s^{(t)}\right) \odot o^{(t)} \tag{19}$$

where σ is the sigmoid function and $\odot$ is the element-wise multiplication. $\mathrm{h}^{(\mathrm{t})}$ is the value of the hidden layer of the LSTM at t the time step is the vector, while $\mathrm{h}^{(\mathrm{t-1})}$ is the output values by each memory cell in the hidden layer at the previous time.

Using LSTM as the network architecture in a BiRNN yields BiLSTM. Combining the advantages of BiRNN and LSTM, BiLSTM-based RNNs were designed [23]. Combining BiRNN with LSTM network can significantly improve the model's performance. A BiLSTM processes sequence data in each forward and backward direction with two separate hidden layers to capture past and future information, respectively. Subsequently, the two hidden states are concatenated to produce the final output like as shown in Fig. 2d. It has been proven that the bidirectional networks are considerably better than unidirectional ones in many fields; we use BiLSTM because it provides access to the long-range context in both input directions and outcomes with full learning on the particular problem. Unidirectional LSTM processed data based on the preserved information solely from the past. In issues, where all time steps of the input sequences are available, BiLSTM train two instead of one LSTMs on the input sequence.

A GRU is a gating mechanism in RNNs, introduced by [24] to solve the vanishing gradient problem of a standard RNN. It is similar to LSTM but has a smaller architecture and similar performance to LSTM. It consists of two gates: reset and update, but LSTM has three gates: input, output, and forget. Furthermore, it contains one activation unit, and LSTM contains two activation units. Therefore, a GRU needs fewer parameters to train a network and much less training time as compared with LSTM. A GRU can be formulated as follows:

$$z^{(t)} = \sigma\left(\mathrm{W}_z x^{(t)} + U_z h^{(t-1)} + b_z\right) \tag{20}$$

$$r^{(t)} = \sigma\left(\mathrm{W}_r x^{(t)} + U_r h^{(t-1)} + b_r\right) \tag{21}$$

$$\widetilde{h}^{(t)} = \tanh\left(\mathrm{W}_x x^{(t)} + U_h\left(r^{(t)} \odot h^{(t-1)}\right) + b_h\right) \tag{22}$$

$$h^{(t)} = \left(1 - z^{(t)}\right) \odot h^{(t-1)} + z^{(t)} \odot \widetilde{h}^{(t)} \tag{23}$$

where $x^{(t)}$ is an input vector, $h^{(t)}$ is an output vector, $z^{(t)}$ is the update gate vector, and $r^{(t)}$ is a reset gate vector at a time step t, respectively. W, U, and b are feedforward weights, recurrent weights, and biases, respectively. $\widetilde{h}^{(t)}$ depicts an output candidate activation. Figure 2c shows the architecture of a GRU.

The BiGRU is a type of BiRNN, which consists of two basic GRUs and processes, the input time series from both forward and backward directions, as depicted in Fig. 2d. Processing data from both directions enables the model to capture the pattern that may be ignored while using unidirectional GRU. Thus, for our problem statement, it can significantly improve classification performance.

2.3.2 Augmentation of different variants of RNN with SE-FCN

As we have already stated in Section 1, we replace the unidirectional LSTM to the other variants of RNN in MLSTM-FCN, one variant at each time, as shown in Fig. 3. The variants which we exploit are RNN, BiRNN, BiLSTM, GRU, and BiGRU. This architecture consists of two parallel modules: RNN and SE-FCN modules.

In RNN module, the input to the model is a multivariate time series dataset with Q time steps and M variables per time step, which passes through a dimension shuffle layer followed by a RNN module (which is comprising of either RNN, BiRNN, BiLSTM, GRU, or BiGRU). The dimension shuffle layer transforms the temporal dimension of the input data. The RNN block requires Q time steps to process M variables at each time step. However, due to the dimension shuffle layer, it transforms the input and requires M time steps to process Q variables per time step. A dimension shuffle helps to improve the efficiency of the network and aids to train a model in less time. After dimension shuffle layer, input provides the complete history to RNN module (that could be any variant, selected during execution), which is followed by dropout of 0.8 [25], to avoid overfitting and then pass through concatenation layer to the SoftMax classifier using the categorical cross-entropy loss function.

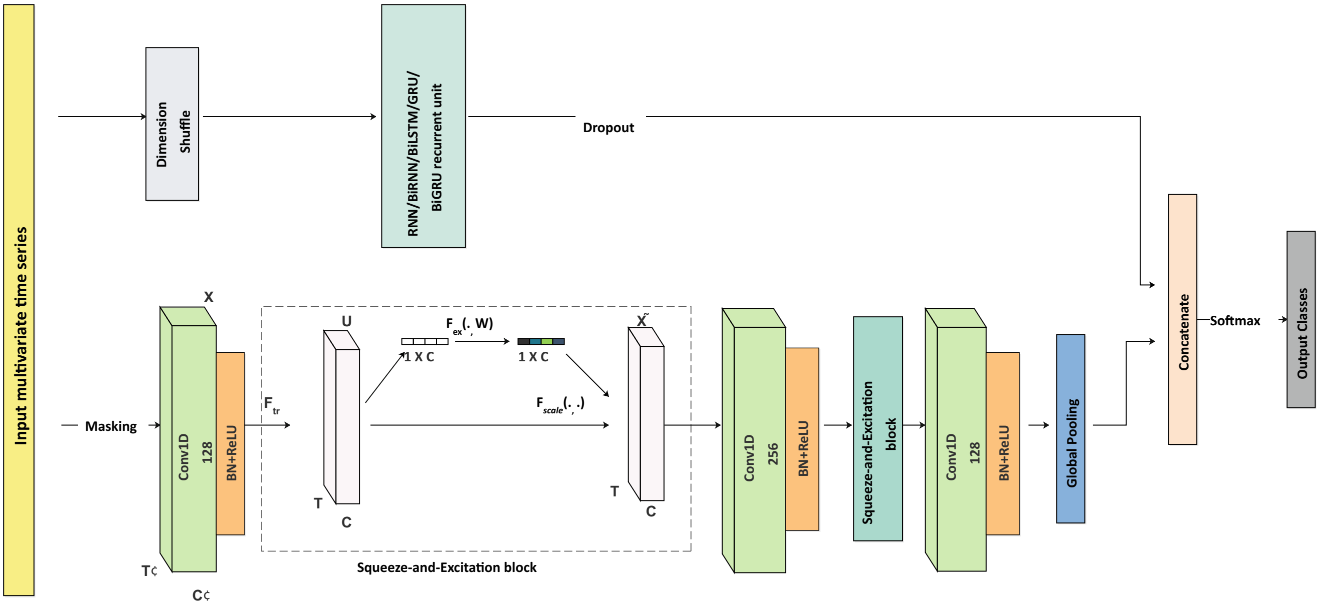


Fig. 3 Network architecture of MRNN-FCN, MBiRNN-FCN, MBiLSTM-FCN, MGRU-FCN, and MBiGRU-FCN

In the SE-FCN module, the input passes through the masking layer to the fully convolutional block, and masking allows to handle variable length inputs. The remaining architecture is the same as SE-FCN (explained in Section 2.2.3). After the final convolutional block, we use the global average pooling layer to interpret the classes and lessen the number of parameters as compared with the fully connected layer and feed them to the concatenation layer. The concatenation layer concatenates the received input from two parallel modules and feeds it to the SoftMax classifier and receives the produced output.

The goal to propose these models is to exploit different RNNs functionality in a hybrid architecture for the MTSC problem. As stated in Section 2.3.1, each RNN variant has a different flavor, and these are used mainly for sequential tasks; time series data is one of the kinds of sequential data, so we tested it in a manner to evaluate their performance in a similar hybrid architecture.

3 Experiments and results

3.1 Datasets

To evaluate the performance of proposed models, we tested them on 35 multivariate time series datasets used and preprocessed by Karim et al. [12]. These datasets were collected from multiple sources [3, 7, 26] and also belong from different domains to perform various classification tasks; a detailed description is presented in Table 1.

3.2 Experimental settings

In this paper, we implemented our models by modifying MLSTM-FCN and MALSTM-FCN [12]. In order to compare with the present state-of-the-art methods, the experimental settings kept unchanged. The models were trained from scratch using the Adam Optimizer [27] with an initial learning rate of 0.001, which reduced to the minimum learning rate 0.0001 and with the batch size of 128. The training epochs were set between 500 and 1000 for all the datasets. The proposed models were experimented using single GPU GTX 1060, Keras library [28] with the TensorFlow [29] in the back end.

3.3 Evaluation metrics

In this paper, we evaluate proposed models and other comparative approaches using testing classification error loss rate, f1 score, arithmetic mean rank, time complexity, and mean per class error (MPCE).

The f1 score depicts an overall measure of the accuracy of the model's performance by combining both precision and recall. The f1 sore is calculated as follows:

$$precision = \frac{TP}{TP + FP} \tag{24}$$

$$recall = \frac{TP}{TP + FN} \tag{25}$$

$$f1\text{−}score = 2 \times \frac{precision \times recall}{precision + recall} \tag{26}$$

where TP, FP, and FN are defined as true positive, false

Table 1 A detailed description of the multivariate time series datasets [12]

Datasets	No. of classes	No. of variables	Maximum training length	MTSC task	Train and test split
AREM	7	7	480	Activity recognition	50–50 split
Daily Sport	19	45	125	Activity recognition	50–50 split
EEG	2	13	117	EEG classification	50–50 split
EEG2	2	64	256	EEG classification	20–80 split
Gesture Phase	5	18	214	Gesture recognition	50–50 split
HAR	6	9	128	Activity recognition	71–29 split
HT Sensor	3	11	5396	Food classification	50–50 split
Movement AAL	2	4	119	Movement classification	50–50 split
Occupancy	2	5	3758	Occupancy classification	35–65 split
Ozone	2	72	291	Weather classification	50–50 split
MSR Activity	16	570	337	Activity recognition	5 ppl in train; rest in test
MSR Action	20	570	100	Action recognition	5 ppl in train; rest in test
Cohn-Kanade AU-coded Expression (CK+)	7	136	71	Facial expression classification	10-fold
Arabic-Voice	88	39	91	Speaker recognition	75–25 split
OHC	20	30	173	Handwriting classification	10-fold
ArabicDigits	10	13	93	Digit recognition	75–25 split
AUSLAN	95	22	96	Sign language recognition	44–56 split
CharacterTrajectories	20	3	205	Handwriting classification	10–90 split
CMUsubject16	2	62	534	Action recognition	50–50 split
DigitShape	4	2	97	Action recognition	60–40 split
ECG	2	2	147	ECG classification	50–50 split
JapaneseVowels	9	12	26	Speech recognition	42–58 split
KickvsPunch	2	62	761	Action recognition	62–38 split
LIBRAS	15	2	45	Sign language recognition	38–62 split
LP1	4	6	15	Robot failure recognition	43–57 split
LP2	5	6	15	Robot failure recognition	36–64 split
LP3	4	6	15	Robot failure recognition	36–64 split
LP4	3	6	15	Robot failure recognition	36–64 split
LP5	5	6	15	Robot failure recognition	39–61 split
NetFlow	2	4	994	Action recognition	60–40 split
PenDigits	10	2	8	Digit recognition	2–98 split
Shapes	3	2	97	Action recognition	60–40 split
U wave	8	3	315	Gesture recognition	20–80 split
Wafer	2	6	198	Manufacturing classification	25–75 split
WalkVsRun	2	62	1918	Action recognition	64–36 split

positive, and false negative, respectively.

MPCE is an evaluation metric introduced by [13] to evaluate the performance of the specific classifier on multiple datasets. MPCE is the arithmetic mean of PCE, which is calculated as follows:

$$PCE_n = \frac{e_n}{c_n} \tag{27}$$

$$MPCE = \frac{1}{N} \sum PCE_n \tag{28}$$

where e_n refers to error rate, c_n is the number of class labels in a dataset, n refers to each dataset, and N is the total number of datasets tested on a specific model. We are using the testing error loss rate to calculate the MPCE, so the lowest score is better.

We further examine the performance by comparing the existing state-of-the-art method with proposed models using the Wilcoxon signed-rank test, which is a non-parametric statistical hypothesis test, and a critical difference diagram to show the pairwise statistical difference comparison among models and used to visualize the classifier's rank.

3.4 Results and analysis

All the proposed models are evaluated on different evaluation metrics. The results are compared against present MTSC state-of-the-art methods MLSTM-FCN, MALSTM-FCN [12], and FCN as a baseline.

Tables 2 and 3 list the quantitative results by testing error loss rate, MPCE, and arithmetic mean, and f1-score also gives a comparative analysis of FCN and state-of-the-art methods and our proposed models.

For testing error loss rate, MRNN-FCN obtained superior performance over 13 datasets, and both SE-FCN and MGRU-FCN over 11 datasets, respectively. Meanwhile, state-of-the-

Table 2 Performance comparison of proposed models with other methods in terms of testing error loss and the mean per class error (MPCE)

Datasets	SE-FCN	MRNN-FCN	MBiRNN-FCN	MBiLSTM-FCN	MGRU-FCN	MBiGRU-FCN	FCN	MLSTM-FCN	MALSTM-FCN
AREM	0.1282	*0.1025*	0.1282	*0.1025*	0.1282	0.1282	0.1282	0.1282	0.1282
Daily Sport	0.0046	*0.0043*	0.0048	0.0046	*0.0043*	*0.0043*	0.0050	0.0048	0.0057
EEG	0.4375	0.4843	0.5000	0.4531	0.4375	0.4687	0.5000	0.4375	*0.4218*
EEG2	0.0899	0.0899	0.0883	0.0816	0.0933	0.0950	*0.0616*	0.0883	0.0883
Gesture Phase	*0.4646*	0.5101	0.5050	0.4797	0.4696	0.4747	0.4747	0.5151	0.4848
HAR	0.0424	0.0478	0.0437	0.0393	0.0356	0.0380	0.0556	0.0400	*0.0352*
HT Sensor	0.3600	0.3799	*0.2799*	0.3000	0.3199	0.3600	0.3799	0.3999	*0.2799*
Movement AAL	0.2420	0.2420	0.2101	0.2547	*0.2038*	0.2356	0.2292	0.2292	0.2292
Occupancy	0.3947	0.3947	0.3947	*0.1842*	*0.1842*	*0.1842*	0.3947	0.1973	0.3947
Ozone	0.2427	0.2427	0.2138	0.2196	*0.2080*	*0.2080*	0.2658	0.2427	*0.2080*
Activity	0.4187	*0.4000*	0.4625	0.4250	0.4438	0.4499	0.4375	0.4312	0.4937
Action 3d	0.2727	0.3063	0.3164	0.3400	0.2659	0.2558	*0.2356*	0.2794	0.3198
CK+	*0.0357*	*0.0357*	0.0714	0.1071	0.0714	0.0714	0.1071	0.0714	0.0714
Arabic-Voice	0.0227	0.0213	0.0222	0.0218	0.0195	0.0195	*0.0181*	0.0240	0.0209
OHC	*0.0035*	*0.0035*	0.0359	0.0071	0.0071	*0.0035*	*0.0035*	0.0071	0.0071
ArabicDigits	*0.0045*	0.0050	0.0050	0.0059	0.0077	0.0068	0.0077	0.0072	0.0095
AUSLAN	0.0519	0.0687	0.0442	0.0554	0.0477	0.0610	*0.0357*	0.0610	0.0428
CharacterTrajectories	0.0093	*0.0007*	0.0066	0.0074	0.0070	0.0066	0.0168	0.0066	0.0086
CMUsubject16	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*
DigitShapes	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*
ECG	0.1499	0.1600	*0.1399*	0.1499	0.1600	0.1499	0.1499	0.1499	*0.1399*
JapaneseVowels	0.0081	0.0108	0.0054	*0.0027*	0.0108	0.0135	0.0054	0.0054	0.0081
KickvsPunch	*0.1000*	*0.1000*	*0.1000*	*0.1000*	*0.1000*	*0.1000*	*0.1000*	*0.1000*	*0.1000*
Libras	0.0341	0.0273	0.0307	*0.0239*	0.0307	0.0273	0.0341	*0.0239*	0.0341
LPl	0.1600	0.1600	*0.1399*	0.1800	0.1800	0.1800	0.1800	0.1600	0.1600
LP2	0.1999	*0.1666*	0.1999	*0.1666*	*0.1666*	0.1999	0.1999	*0.1666*	0.2333
LP3	0.2666	0.3000	0.2666	0.2666	0.2666	0.2666	0.3666	0.2666	*0.2333*
LP4	0.0666	0.0666	0.0933	0.1066	0.0799	*0.0533*	0.1333	0.1200	0.0800
LP5	*0.3199*	0.3299	0.3399	0.3399	0.3299	0.3600	0.3999	0.3600	0.3399
NetFlow	*0.0430*	0.0617	0.0816	0.0711	0.0561	0.0823	0.0449	0.0580	0.0880
PenDlgits	0.0357	0.0355	0.0374	0.0351	0.0370	0.0361	*0.0349*	0.0361	0.0353
Shapes	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*
U wave	0.0254	0.0242	0.0245	0.0242	0.0254	0.0251	0.0234	0.0242	*0.0231*
Wafer	0.0089	*0.0078*	*0.0078*	0.0089	*0.0078*	0.0100	0.0111	*0.0078*	0.0111
WalkvsRun	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*
Wins/Ties	11	*13*	09	10	11	10	11	08	12
MPCE	0.0394	0.0410	0.0399	0.0370	*0.0357*	0.0374	0.0427	0.0380	0.0390
Arithmetic mean	3.6000	3.8000	4.0285	3.7714	*3.4285*	3.8857	4.4857	3.8857	4.0000

The instances in italics show the best performance

Table 3 Performance comparison of proposed models with other methods in terms of f1 score

Datasets	SE-FCN	MRNN-FCN	MBiRNN-FCN	MBiLSTM-FCN	MGRU-FCN	MBiGRU-FCN	FCN	MLSTM-FCN	MALSTM-FCN
AREM	0.8717	*0.8974*	0.8717	*0.8974*	0.8571	0.8717	0.8717	0.8717	0.8717
Daily Sport	0.9952	*0.9956*	0.9952	0.9953	*0.9956*	*0.9956*	0.9949	0.9951	0.9942
EEG	0.5625	0.5156	0.5000	0.5468	0.5625	0.5312	0.5000	0.5625	*0.5781*
EEG2	0.9100	0.9100	0.9116	0.9183	0.9066	0.9049	*0.9383*	0.9116	0.9116
Gesture Phase	*0.5357*	0.4803	0.4956	0.5089	0.5342	0.5269	0.5317	0.4848	0.5204
HAR	0.9575	0.9523	0.9562	0.9606	0.9639	0.9621	0.9443	0.9598	*0.9647*
HT Sensor	0.6122	0.6262	*0.7272*	0.6804	0.6868	0.6399	0.6262	0.6000	0.6938
Movement AAL	0.7579	0.7579	0.7898	0.7452	*0.7961*	0.7643	0.7707	0.7707	0.7707
Occupancy	0.6052	0.6052	0.6052	*0.8157*	*0.8157*	*0.8157*	0.6052	0.8026	0.6052
Ozone	0.7572	0.7572	0.7861	0.7803	*0.7919*	*0.7919*	0.7341	0.7572	*0.7919*
Activity	0.5889	*0.5908*	0.5359	0.5754	0.5705	0.5609	0.5808	0.5714	0.5180
Action 3d	0.7282	0.6959	0.6931	0.6663	0.7326	*0.7508*	0.7476	0.7236	0.6894
CK+	0.9454	*0.9642*	0.9285	0.8928	0.9285	0.9285	0.8928	0.9285	0.9285
Arabic-Voice	0.9770	0.9785	0.9781	0.9776	0.9799	*0.9808*	*0.9824*	0.9760	0.9793
OHC	*0.9964*	*0.9964*	*0.9964*	0.9928	0.9928	*0.9964*	*0.9964*	0.9928	0.9928
ArabicDigits	*0.9954*	0.9952	0.9950	0.9940	0.9922	0.9931	0.9922	0.9927	0.9904
AUSLAN	0.9479	0.9328	0.9556	0.9459	0.9536	0.9393	*0.9641*	0.9393	0.9571
CharacterTrajectories	0.9904	0.9923	0.9931	0.9921	*0.9933*	0.9929	0.9848	0.9931	0.9905
CMUsubject16	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*
DigitShapes	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*
ECG	0.8500	0.8399	*0.8600*	0.8500	0.8399	0.8500	0.8500	0.8500	*0.8600*
JapaneseVowels	0.9918	0.9891	0.9945	*0.9972*	0.9891	0.9891	0.9932	0.9932	0.9918
KickvsPunch	*0.8999*	*0.8999*	*0.8999*	*0.8999*	*0.8999*	*0.8999*	*0.8999*	*0.8999*	*0.8999*
Libras	0.9621	0.9725	0.9690	0.9742	0.9692	*0.9757*	0.9674	0.9705	0.9656
LPl	0.8399	0.8399	*0.8600*	0.8199	0.8199	0.8199	0.8199	0.8399	0.8399
LP2	0.8000	0.8135	0.8000	0.8135	*0.8333*	0.8000	0.7796	0.8135	0.7796
LP3	0.7333	0.6666	0.7333	0.7118	0.7118	0.7118	0.6333	0.7241	*0.7666*
LP4	0.9333	0.9333	0.9066	0.8933	0.9200	*0.9466*	0.8724	0.8724	0.9200
LP5	*0.6868*	0.6700	0.6633	0.6633	0.6700	0.6464	0.6000	0.6432	0.6600
NetFlow	*0.9569*	0.9382	0.9138	0.9288	0.9438	0.9176	0.9550	0.9419	0.9119
PenDlgits	0.9654	0.9650	0.9628	0.9652	0.9642	0.9633	*0.9661*	0.9646	0.9646
Shapes	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*
UWave	0.9748	0.9761	0.9758	0.9763	0.9745	0.9753	*0.9773*	0.9765	0.9768
Wafer	0.9910	*0.9921*	*0.9921*	0.9910	*0.9921*	0.9899	0.9888	*0.9921*	0.9888
WalkvsRun	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*	*1.0000*
Wins/Ties	10	11	10	8	12	*13*	11	6	10

The instances in italics show the best performance

art methods, MLSTM-FCN and MALSTM-FCN, showed the best performance over 08 and 12 datasets, respectively. MGRU-FCN secured the lowest MPCE score of 0.0357, MBiLSTM-FCN with second lowest has 0.0370, and MBiGRU-FCN and SE-FCN have 0.0374 and 0.0394, respectively, whereas MLSTM-FCN and MALSTM-FCN have MPCE of 0.0380 and 0.0390, respectively. Similarly, MGRU-FCN has the lowest arithmetic mean rank of 3.4285, and SE-FCN has the second lowest has 3.6000, whereas MLSTM-FCN and MALSTM-FCN have 3.8857 and 4.000, respectively. FCN showed best results over 11 datasets with the MPCE and arithmetic rank of 0.0427 and 4.4857, respectively.

In terms of f1 score, MBiGRU-FCN showed superior performance by winning over 13 datasets, MGRU-FCN on 12 and MRNN-FCNN and SE-FCN on 11 and 10, respectively.

MLSTM-FCN and MALSTM-FCN showed the best performance over 6 and 10 datasets, respectively. The baseline, FCN win over 11 datasets that demonstrate that FCN is itself robust enough as a baseline to perform better than some hybrid deep learning models.

Figure 4a and b show the training and validation error loss of proposed models and current state-of-the-art models over the movement AAL (movement classification) and wafer (manufacturing classification) datasets. The proposed models work well; therefore, they would provide lower time complexity values as it is illustrated in Fig. 5 and Table 4. From Fig. 5, it has been noted that the comparison metric is analyzed by the existing methods and the proposed models by means of the time complexity. In the x-axis, methods have been taken and, in the y-axis, time complexity value has been plotted. As a single module, FCN shows the lowest time complexity among all the models. Besides, SE-FCN is showing 2nd lowest since it contains less number of parameters and a combination of SE block and FCN only, not any RNNs variant is

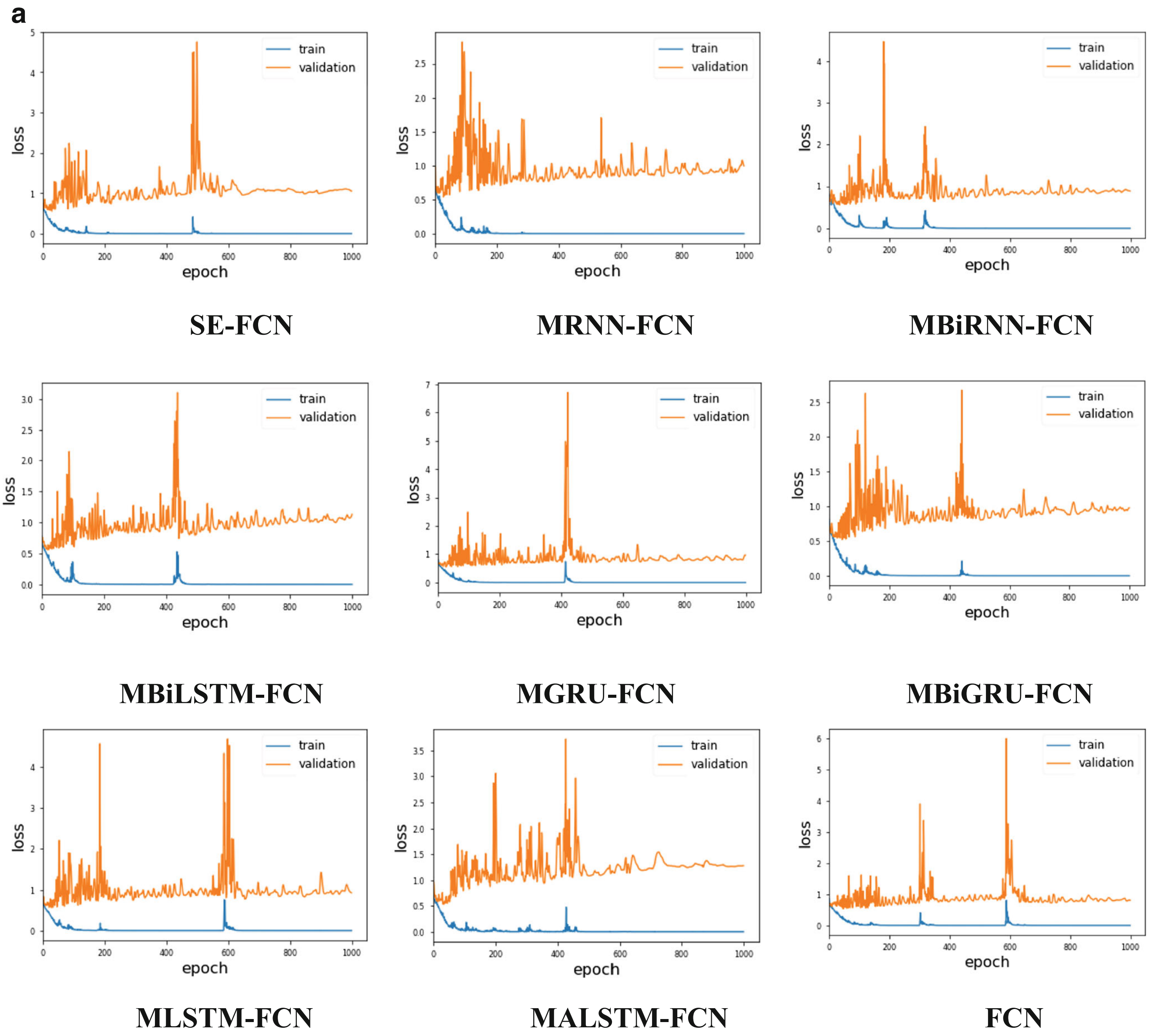
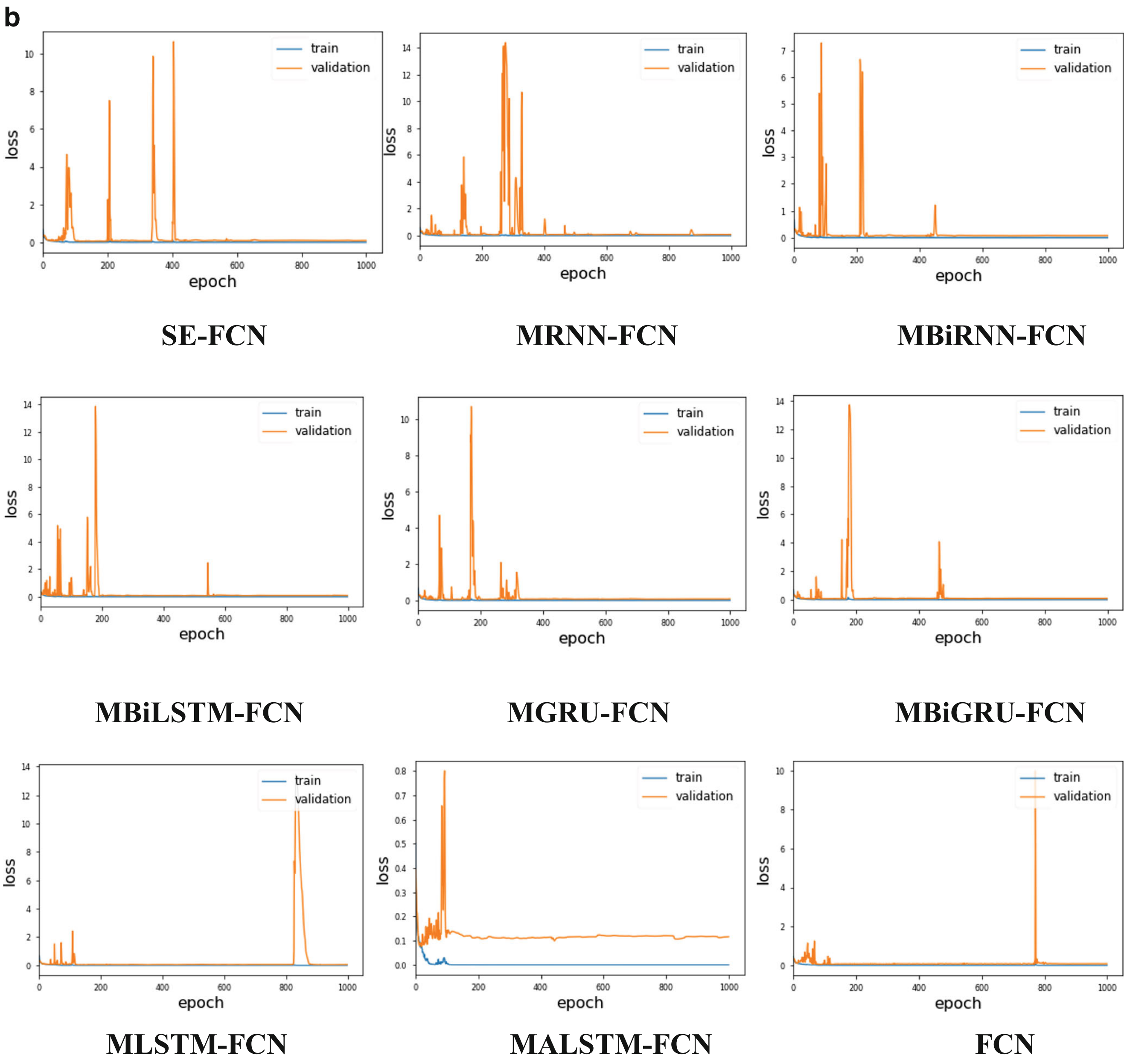


Fig. 4 **a** The training and validation error loss of proposed models and current state-of-the-art models over the movement AAL dataset (movement classification). **b** The training and validation error loss of proposed models and current state-of-the-art models over the wafer dataset (manufacturing classification)



SE-FCN MRNN-FCN MBiRNN-FCN

MBiLSTM-FCN MGRU-FCN MBiGRU-FCN

MLSTM-FCN MALSTM-FCN FCN

Fig. 4 (continued)

augmented in this model. Among all the RNN variant-based hybrid models, MRNN-FCN depicts the lowest time complexity, afterward MBiRNN-FCN and MGRU-FCN. MLSTM-FCN and MALSTM-FCN rank 6th and 9th, respectively, in means of time complexity. Figure 6 depicts the remarkable performance of the proposed models over existing methods through a critical difference diagram. Furthermore, Table 5 represents the performance by comparing the existing state-of-the-art method with proposed models using a non-parametric statistical hypothesis test and Wilcoxon signed-rank test.

Surprisingly, all the proposed models, state-of-the-art methods, and baseline show identical performance for action recognition MTSC tasks on CMUsubject16, DigitShape, KickvsPunch, Shapes, and WalkVsRun datasets, in terms of classification testing error loss and f1 score.

The results indicate that the other variants of RNN can perform better than unidirectional LSTM in a hybrid deep learning model. Also, the squeeze-and-excitation block can perform better when integrated separately within FCN for MTSC problems. Therefore, the proposed models are assumed as significantly better than the present state-of-the-art

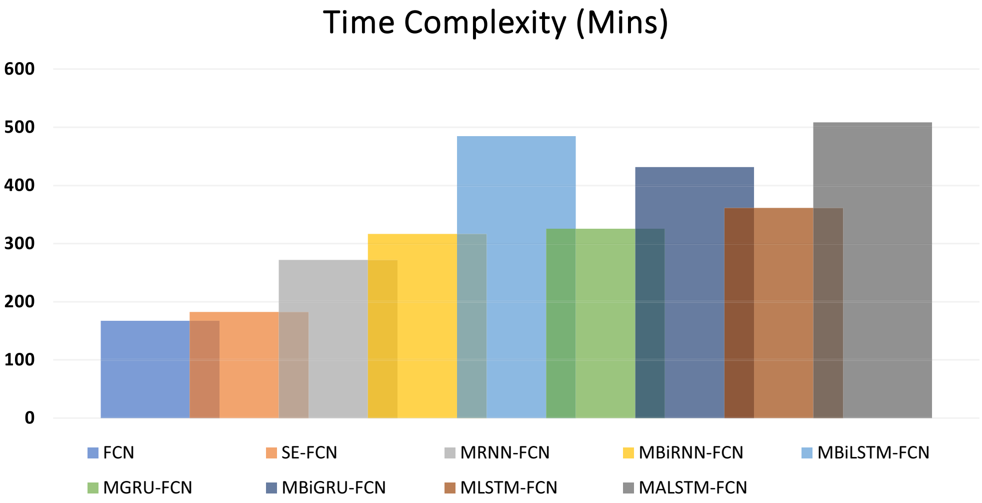


Fig. 5 Time complexity of proposed models and current state-of-the-art methods

Table 4 Time complexity (minutes) of proposed models with other methods

Datasets	SE-FCN	MRNN-FCN	MBiRNN-FCN	MBiLSTM-FCN	MGRU-FCN	MBiGRU-FCN	FCN	MLSTM-FCN	MALSTM-FCN
AREM	1.0009	1.1253	*0.8271*	1.1865	1.4899	1.2322	0.8581	1.6135	1.4922
Daily Sport	14.3541	26.7251	32.663	48.8458	32.4843	41.8243	*13.2986*	36.2198	64.4306
EEG	1.1554	0.991	0.725	1.454	1.352	1.413	*0.6582*	1.525	1.465
EEG2	4.1782	6.687	*0.9116*	11.3925	7.7341	10.031	3.8823	8.766	17.5955
Gesture Phase	1.4152	1.6941	1.8891	2.0421	1.7475	1.8	*0.9727*	1.8733	2.5271
HAR	18.6179	23.9535	26.4922	31.9032	26.123	29.4516	*16.7021*	27.655	30.4561
HT Sensor	6.6996	*5.8321*	5.8616	6.3211	6.3848	6.228	6.0315	6.6408	7.2468
Movement AAL	1.108	1.5761	1.3843	1.7635	1.3619	1.7204	*0.9735*	1.3731	2.1173
Occupancy	6.4431	6.8369	6.5918	6.7778	6.253	6.8902	*6.0745*	6.8452	6.6693
Ozone	*1.9476*	3.441	4.1393	7.5962	4.1035	6.4127	1.9507	4.9965	9.3786
Activity	10.6331	28	36.5314	63.646	39.4	53.7833	*10.4978*	45.55	54.08
Action 3d	*3.589*	18.933	28.1701	55.3875	28.853	57.7384	3.6656	34.2848	37.97
CK+	1.3659	3.7077	5.4	11.4571	5.8799	9.1856	*0.9698*	6.96	10.9334
Arabic-Voice	15.3993	30.897	40.308	78.9153	40.4081	63.2938	*13.7258*	47.21	72.592
OHC	8.6968	12.8955	15.2373	20.5246	14.9311	18.1734	*8.0163*	16.2985	22.7612
ArabicDigits	13.8846	19.3204	23.0815	31.736	21.6078	28.112	*12.5324*	22.7382	28.38
AUSLAN	6.3013	10.2081	12.203	15.4587	11.5422	13.6071	*5.7927*	12.47	16.58
CharacterTrajectories	6.8235	7.2918	7.5029	8.4364	7.5247	8.449	*6.2197*	7.8158	8.58
CMUsubject16	2.2335	*2.1992*	3.1302	5.2185	2.8487	4.1761	4.7502	3.3278	3.4105
DigitShapes	1.4504	*0.5824*	0.9036	0.8323	0.6906	0.6395	0.8646	0.7756	0.8053
ECG	1.8604	*0.9538*	1.0018	1.8605	1.0873	1.1836	1.1994	1.19	1.44
JapaneseVowels	0.9145	1.8518	2.3425	3.3699	2.3268	2.812	*0.8099*	2.4868	3.15
KickvsPunch	1.7275	2.3219	2.5973	5.0665	3.0995	4.2442	*1.604*	3.6457	4.9406
Libras	*1.6463*	1.9392	1.676	2.1462	2.1019	1.8491	2.3512	2.30	2.89
LPl	1.0711	1.5374	1.7092	1.3158	1.2281	0.9978	*0.9038*	1.12	2.27
LP2	1.1532	0.929	*0.6682*	0.9954	0.9849	0.8375	0.9009	0.8565	1.27
LP3	1.0952	0.9321	*0.6696*	0.9686	0.9943	1.0826	0.9085	0.8482	1.34
LP4	1.1098	*0.6709*	1.3057	1.6389	0.8185	1.5579	0.906	1.16	2.05
LP5	1.1996	*0.9295*	1.0045	1.2147	1.2666	1.1466	0.9423	1.39	2.05
NetFlow	23.1853	23.3537	24.0085	24.9215	23.5955	24.671	*20.4973*	24.31	26.11
PenDlgits	3.224	5.2808	5.9999	8.1473	5.8068	6.6869	*3.1156*	6.0989	7.1158
Shapes	1.4694	*0.5573*	0.9061	0.773	0.6418	0.6092	0.866	0.7367	0.7727
UWave	8.3379	9.6249	9.8204	9.9811	10.0006	10.0099	*7.7732*	10.4779	10.9849
Wafer	3.5294	4.5766	4.8697	5.5326	4.848	4.978	*3.313*	5.1206	5.6961
WalkvsRun	3.6735	3.7369	4.1948	5.7603	4.1688	4.8023	*2.7931*	4.6358	37.1573
Sum (min)	*182.4945*	*272.093*	316.7272	484.5874	325.6895	431.6302	*167.3213*	361.3156	508.7023

The instances in italics show the best performance

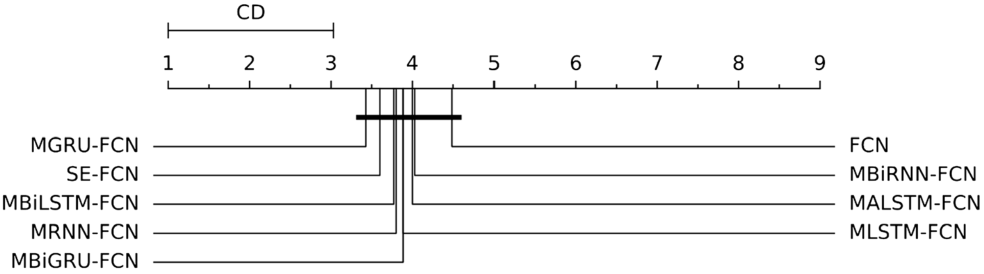


Fig. 6 Critical difference diagram showing the performance of proposed models to the current state-of-the-art classifier of multivariate time series data

Table 5 Wilcoxon signed-ranked test on proposed models and benchmark models over 35 multivariate time series datasets

	SE-FCN	MRNN-FCN	MBiRNN-FCN	MBiLSTM-FCN	MGRU-FCN	MBiGRU-FCN	MLSTM-FCN	MALSTM-FCN
FCN	*4.87E-02*	1.95E-01	2.42E-01	8.37E-02	*5.46E-02*	2.19E-01	2.00E-01	4.43E-01
SE-FCN		*1.89E-01*	5.09E-01	6.20E-01	9.49E-01	4.24E-01	*2.41E-01*	6.00E-01
MRNN-FCN			*6.14E-01*	7.55E-01	*2.41E-01*	6.74E-01	8.82E-01	9.09E-01
MBiRNN-FCN				5.69E-01	*6.53E-02*	6.57E-01	7.09E-01	9.76E-01
MBiLSTM-FCN					*3.07E-01*	4.39E-01	*3.17E-01*	6.57E-01
MGRU-FCN						*8.24E-02*	*1.07E-01*	4.24E-01
MBiGRU-FCN							*8.84E-01*	9.81E-01
MLSTM-FCN								*7.32E-01*

The instances in italics show the best performance

methods in terms of classification testing error loss, f1 score, MPCE score, arithmetic mean rank, and time complexity.

4 Conclusions

In this study, we have proposed new hybrid deep learning frameworks that depict superior performance on various datasets for the MTSC problem. This study also shows that the other variants of RNN can perform better in hybrid models than a unidirectional LSTM for MTSC problems such as action recognition, activity recognition, ECG/EEG classification, robot failure recognition, speech recognition. Moreover, SE-FCN shows better performance than MLSTM-FCN and MALSTM-FCN, which indicates that the SE block can work self-sufficiently when integrated within FCN for our problem statement. The proposed models are end-to-end and do not require heavy data pre-processing or feature crafting. Therefore, they could be easily deployed on real-time systems. We validate the effectiveness of our proposed approaches by conducting a comprehensive evaluation on 35 datasets from diverse domains. The results demonstrate that our approaches achieve remarkable results over the present state-of-the-art MTSC approaches.

Funding This paper was partially supported by the NSFC grant U1509216, U1866602, 61602129, and Microsoft Research Asia.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Fawaz HI et al (2019) Deep learning for time series classification: a review. Data Min Knowl Disc 33(4):917–963
2. De Gooijer JG, Hyndman RJ (2006) 25 years of time series forecasting. Int J Forecast 22(3):443–473
3. Schäfer P, Leser U (2017) Multivariate time series classification with WEASEL+ MUSE. arXiv preprint arXiv:1711.11343
4. Jaakkola T, Diekhans M, Haussler D (2000) A discriminative framework for detecting remote protein homologies. J Comput Biol 7(1–2):95–114
5. Van Der Maaten L (2011) Learning discriminative fisher kernels. in ICML
6. Orsenigo C, Vercellis C (2010) Combining discrete SVM and fixed cardinality warping distances for multivariate time series classification. Pattern Recogn 43(11):3787–3794
7. Pei W, Dibeklioglu H, Tax DMJ, van der Maaten L (2017) Multivariate time-series classification using the hidden-unit logistic

model. IEEE Transactions on Neural Networks and Learning Systems 29(4):920–931

8. Baydogan MG, Runger G (2015) Learning a symbolic representation for multivariate time series classification. Data Min Knowl Disc 29(2):400–422
9. Wistuba M, Grabocka J, Schmidt-Thieme L (2015) Ultra-fast shapelets for time series classification. arXiv preprint arXiv:1503.05018
10. Tuncel KS, Baydogan MG (2018) Autoregressive forests for multivariate time series modeling. Pattern Recogn 73:202–215
11. Zheng Y, et al. (2014) Time series classification using multichannels deep convolutional neural networks. in International Conference on Web-Age Information Management. 2014. Springer
12. Karim F, Majumdar S, Darabi H, Harford S (2019) Multivariate lstm-fcns for time series classification. Neural Netw 116:237–245
13. Wang Z, Yan W, Oates T (2017) Time series classification from scratch with deep neural networks: a strong baseline. in 2017 International Joint Conference on Neural Networks (IJCNN). IEEE
14. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167
15. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. in Proceedings of the 27th international conference on machine learning (ICML-10)
16. Lin M, Chen Q, Yan S (2013) Network in network. arXiv preprint arXiv:1312.4400
17. Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. in Proceedings of the IEEE conference on computer vision and pattern recognition
18. He K, et al. (2015) Delving deep into rectifiers: surpassing human-level performance on imagenet classification. in Proceedings of the IEEE international conference on computer vision
19. Lipton ZC, Berkowitz J, Elkan C (2015) A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019
20. Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Trans Signal Process 45(11):2673–2681
21. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
22. Zhao Y, Yang R, Chevalier G, Shah RC, Romijnders R (2018) Applying deep bidirectional LSTM and mixture density network for basketball trajectory prediction. Optik 158:266–272
23. Graves A, Schmidhuber J (2005) Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw 18(5–6):602–610
24. Chung J, et al. (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555
25. Srivastava N et al (2014) Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1):1929–1958
26. Dua, DAG, Casey (2017) UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences
27. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
28. Chollet F (2015) *Keras*. Available from: https://github.com/fchollet/keras
29. Abadi M, et al. (2016) Tensorflow: a system for large-scale machine learning. in 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)