Pluggable Explanation for Deep Neural Networks-based Multivariate Time Series Classification

Chao Yang[1], Xianzhi Wang[1], Lina Yao[2], Jing Jiang[3], and Guandong Xu[4]

[1] School of Computer Science, University of Technology Sydney, Australia
[2] School of Computer Science and Engineering, UNSW, Australia
[3] Australian AI Institute, University of Technology Sydney, Australia
[4] Data Science Institute, University of Technology Sydney, Australia

Abstract. Deep neural networks currently achieve state-of-the-art performance in many multivariate time series classification (MTSC) tasks, which are crucial for various real-world applications. However, the black-box characteristic of deep learning models impedes humans from obtaining insights into the internal regulation and decisions made by classifiers. Existing explainability research generally requires constructing separate explanation models to work with deep learning models or process their results, thus calling for additional development efforts. We propose a novel explanation module pluggable into existing deep neural networks to explore variable importance for explaining MTSC. We evaluate our module with popular deep neural networks on both real-world and synthetic datasets to demonstrate its effectiveness in generating explanations for MTSC. Our experiments also show the module improves the classification accuracy of existing models due to the comprehensive incorporation of temporal features.

1 Introduction

The past decade has seen multivariate time series classification (MTSC) becoming one of the most critical issues in data mining [11]. MTSC finds significance in various practical tasks, such as activity recognition [40], disease diagnosis [31], and weather forecasting [25]. Currently, deep neural networks have been widely adopted for MTSC [12] and achieved state-of-the-art performance in various tasks, thanks to the ability to capture complicated, non-linear relations between inputs and outputs [29]. Generally, deep neural networks stack multiple neural layers to automate feature extraction and representation learning, and their internal mechanisms remain unrevealed to the end-user. Nevertheless, many real-world applications find the significance of gaining insights into the critical variables that impact the decisions of classifiers [37] to approach a better understanding of specific domains. For example, in aquaculture, multiple environmental conditions (e.g., light) jointly affect the creature's growth. However, although researchers can monitor environmental variables and growth of creatures [16, 17] and predict the growth trend by solving a multivariate time series classification

(MTSC) problem, it is more desirable to derive human-understandable interpretations of which factors play the major role in determining the classification outcomes. Various applications in other domains, e.g., healthcare and medical diagnosis [15, 32] call for explainable MTSC as well.

A deep neural network for multivariate time series classification usually consists of two components: backbone and head. The backbone is responsible for extracting temporal features and harnessing the inter-relationship of the variables to learn the representations of the input data, called feature-maps. The head can map the feature-maps to the possibility distribution of the output labels, i. e. the classes. The backbone conducts feature extraction by fusing the temporal features from different variables. While it is beneficial for the model to effectively harness the temporal features of the input time series, it leads to challenges in finding the important information from variables. For example, in convolutional neural networks, the filter in the first layer will harness all the channels' information simultaneously—the channels are fully connected for information fusion across all the channels. Hence, as the networks go deep, it is nearly impossible for the typical convolutional neural network to track the variables' importance during inference.

Although many studies have sought explanation for classification problems [42, 1], they mostly design separate architectures that are specific to certain deep neural-network types. They need to re-design the backbone architecture (following the ad-hoc approach) or propose post-hoc techniques, which lack the flexibility to be applied to different deep neural networks. Besides, the whole architecture has to be re-evaluated when task or circumstance changes, leading to extra efforts for model adaptation. All the above deficiencies call for a generic module that is pluggable into various deep neural networks for MTSC. In this regard, we propose an explanation module that can be seamlessly integrated into deep neural networks to gain the importance of variables in MTSC. We make the following contributions in this paper:

- We propose an explanation module that can be plugged into existing popular deep neural networks, such as CNN, RNN, to infer the importance of variables in MTSC automatically.
- We conducted experiments on four benchmark multivariate time series datasets using four variants of CNN and RNN to evaluate our proposed module. Our experiments on input variables with added noises validate the effectiveness of the module.
- Besides adding explainability, the experimental results show that our module enables the MTSC models to better leverage the temporal feature and achieve better accuracy.
- We provide implementation details of the proposed module and related experiments to ensure our module can be re-implemented conveniently.

The rest of this paper is organized as follows: Section 2 introduces some related works and techniques; Section 3 presents the structure of the proposed explanation module; Section 4 reports our experiments; finally, Section 5 concludes our works.

2 Related Work

2.1 Multivariate Time Series Classification

Convolutional Neural Networks (CNNs) are firstly used for image recognition [27]. Recent studies have found that 1D CNN can be used for temporal feature extraction [19, 5, 22], hence inspiring researchers to use CNN for time series classification [13, 44, 28]. For 1D CNN, the convolution computation can harness the potential temporal patterns while the information fusion across the channels is helpful to tackle the inter-relations of the variables. As CNNs focus on the information in the receptive field, it is challenging to capture a relatively long-range time series.

Recurrent Neural Network (RNN) is a structure specifically designed for temporal data [30, 33]. Two most well-known variants are called Long Short-term Memory (LSTM) [21] and Recurrent Gated Unit (GRU) [8] which are widely used in dealing with time series sequences [39, 6, 36]. RNNs have the shortcomings of containing massive parameters. Besides, it is difficult to apply parallel computation to RNNs, which further degrades the time consumption [35].

The combination of CNN and RNN represents one effort to fix the shortcomings [4, 3]. CNN and RNN are constructed in the parallel or cascade style to exploit the advantages of both CNN and RNN. This architecture is beneficial for capturing various ranges of temporal feature extraction. LSTM-FCNs [26] construct CNN and RNN in a two parallel stream style. Combining this architecture with the attention layer called Squeeze-and-Excitation Net [24] can achieve state-of-the-art performance on several benchmark multivariate time series classification datasets.

2.2 Explanation Methods

There have been several efforts exploring explanation methods for deep neural networks in various tasks. Some efforts have tried to figure out the effect of the input on the output [43]. Gradient-based methods have been used for exploring the influence of the input changes [38, 7]. However, these types of methods are only feasible for convolutional neural networks.

Another explanation approach is to design a separate architecture for explanation purposes. Some studies [41, 18] select a critical subset of features to figure out the most influential variables. While some work embeds attention mechanisms to evaluate the effectiveness of the input data [2, 9], it may take considerable efforts to design a new architecture, not to mention the potential adverse impact of the explanation module on performance. An example is LAXCAT [23]: although it can visualize critical variables based on fully-grouped convolutions and attention mechanisms, it lacks the ability to exploit the inter-relationship among variables, resulting in suboptimal performance.

3 Our Approach

In a typical multivariate time series classification (MTSC) process (represented by blue blocks in Fig. 1), the input firstly goes through the backbone (e.g., CNN or RNN) to generate feature-maps (denoted by $\mathbf{FM}_{backbone} \in \mathbb{R}^{N \times L}$). Then, the head (usually, a fully-connected layer or 1D convolutional layer) maps feature-maps to a probability distribution of classes.

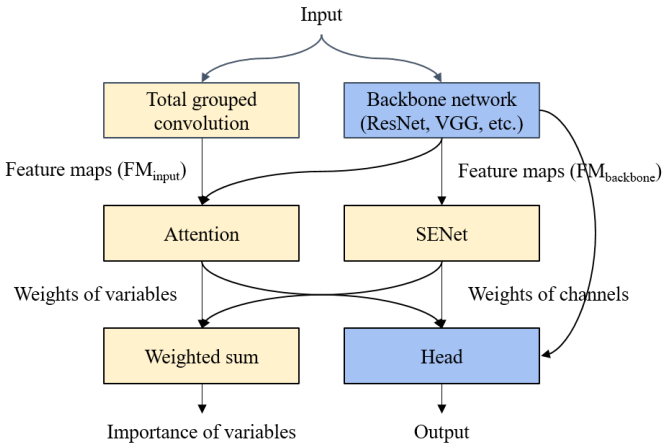

Fig. 1. The proposed module is pluggable into any existing deep learning model (i.e., *backbone network*) in a backbone-head fashion. Yellow blocks represent our proposed module and Blue ones stand for the original neural-network model. The module obtains the importance of variables by calculating attention on top of the feature-maps extracted by the backbone ($\mathbf{FM}_{backbone}$) and by the grouped convolution layers ($\mathbf{FM}_{input}$), respectively. The module updates $\mathbf{FM}_{backbone}$ twice according to the outputs of *Attention* and *SENet* to enhance the backbone network's performance.

Our proposed module (represented yellow blocks in Fig. 1) aims to explore the importance of variables for pluggable explanation in MTSC. Our module works in the following steps. Given input fed to a total grouped convolution layer, the convolution filters conduct separate calculations on each variable. Normal convolution is fully connected in the channel perspective causing the information flow among the channels. Total grouped convolution splits channel of the input data and does convolutions on each channel. In this manner, the number of filters is equal to the number of the variables. Hence, during this process, the module does not consider any inter-relationship of the variables. The feature-map of each channel is the representation of each variable. The output feature-map is indicated by $\mathbf{FM}_{input} \in \mathbb{R}^{M \times L}$, where $\mathbf{M}$ is the number of variables, and $\mathbf{L}$ is the length of the feature-map. Noted, the length of the $\mathbf{FM}_{input}$ should be the same as $\mathbf{FM}_{backbone}$ for the attention calculation. Generally, the backbone

downsamples the input that leads to the small length of $\mathbf{FM}_{backbone}$ than the input time series sequence. If necessary, the module will adjust the kernel size of the total grouped convolution according to the two feature-maps to ensure their lengths are equal. Typically, using a large stride for downsampling helps ensure the feature-maps' lengths meet the module's requirements. Since no information flows across variables, each channel of $\mathbf{FM}_{input}$ can be considered as the vectorized representation of the corresponding variable.

After this, the attention between the $\mathbf{FM}_{backbone}$ and $\mathbf{FM}_{input}$ is calculated. In this step, we have the importance of the variables according to the channels of $\mathbf{FM}_{backbone}$, indicated by $\mathbf{Attn}_{nm} \in \mathbb{R}^{N \times M}$, it also can be seen as the weights of the variables. Besides, $\mathbf{FM}_{backbone}$ is updated based on the results. In the next step, $\mathbf{FM}_{backbone}$ is sent to the SENet to obtain the importance of the channels or the weights of the channels. As the importance is learnt according to the interrelationship of the channels and the importance of different channels regarding to the output, we indicate it using $\mathbf{SelfAttn}_n \in \mathbb{R}^N$, while the $\mathbf{FM}_{backbone}$ is updated the second time. Then, we have the importance of the variables regarding to the channels of the $\mathbf{FM}_{backbone}$ $\mathbf{Attn}_{nm}$ as well as the importance of the channels of the $\mathbf{FM}_{backbone}$ $\mathbf{SelfAttn}_n$. At the last step, the module does weight sum to obtain the importance of the variables as Eq. (1):

$$\text{Importance}_{\ m} = \sum_{1}^{N} \text{Attn}_{\ nm} \times \ \text{Self Attn}_{\ n} \tag{1}$$

where $\mathbf{Importance}_m \in \mathbb{R}^M$. In this way, the importance of the $\mathbf{M}$ variables on the decision-making process of the classifier is obtained.

As the feature-maps $\mathbf{FM}_{backbone}$ are updated twice based on the results of two attention calculation steps, the model can utilize the temporal information more effectively to achieve better performance. Specifically, the $\mathbf{FM}_{input}$ contains different granular feature-maps compared with $\mathbf{FM}_{backbone}$. Hence, the module can harness more feature-maps to achieve better classification accuracy. Besides, our module can be integrated into the classifier following a backbone-head style. It has nearly no difficulty and limitation for combining the proposed module with the existing models to figure out the importance of MTSC variables.

4 Experiments

4.1 Datasets

We conducted experiments on four carefully selected public multivariate time series datasets (Table 1), which are representative of different sizes and domains. Table 1 includes the number of the classes, number of the variables for each input sequence, the length of the sequence, and the train-test split ratio. More details are as follows:

– **AREM [10]**: AREM dataset contains time series sequences recorded by sensors placed in different positions of the body to recognize the activities.

Table 1. Experimental datasets

Dataset	Classes	Number of variables	Sequence length	Train-test ratio
AREM [10]	7	7	480	50:50
LP5 [10]	5	6	15	39:61
ArabicDigits [10]	10	13	93	75:25
Wafer [34]	5	18	214	25:75

The dataset consists of six activity types: cycling, lying, sitting, standing, walking, bending1, and bending2.
- **LP5 [10]**: LP5 dataset is for robot failures detection in motion. It contains five classes, including normal, bottom collision, bottom obstruction, collision in part, and collision in tool.
- **ArabicDigits [10]**: ArabicDigits is used to detect which Arabic digits the writer is writing. So it is very intuitive that the dataset contains 10 classes, including the digits ranging from 0 to 9.
- **Wafer [34]**: The wafer database comprises a collection of time-series data sets where each file contains the sequence of measurements recorded by one vacuum-chamber sensor during the etch process applied to one silicon wafer during the manufacture of semiconductor microelectronics. It contains two classes: normal or abnormal.

For each dataset, we normalized it to zero mean and unit standard deviation; we also applied zero paddings to cope with sequences with different lengths.

4.2 Baseline Methods

We select two representative variants of CNN and two representative variants of RNN to demonstrate the feasibility of plugging our proposed module into existing models. These models also serve as baseline methods for comparative experiments.

- **ResNet** [20] and **Res2Net** [14]: Popular Convolutional Neural Network-based models. We train ResNet on AREM dataset and train Res2Net on LP5 dataset. ResNet and Res2Net contain 4 convolutional layers. Each convolutional layer is 1D convolution to ensure the model is adaptable for time series data.
- **LSTM** [21] and **GRU** [8]: Popular Recurrent Neural Network-based models. We train LSTM on the ArabicDigits dataset and train GRU on the Wafer dataset. As LSTM and GRU contain 2 RNN layers, we use the last hidden state as the information vector. The vector is sent to a fully connected layer to map the information vector to the probability distribution of the classes.

4.3 Evaluation Procedure

For each model, we followed the given train-test split regulation and firstly train it on the training set. We train our model on a single GTX 3090 GPU with 24

GB memory. We apply *oversampling* to classes with fewer samples to mitigate the impact of imbalanced class distribution. Then, considering the importance of variables vary across classes, for each dataset, we select a particular class from the test set to evaluate the importance of variables produced by our module. Specifically, we select the sixth class (i.e., bending2), the third class (i.e., bottom obstruction), the first class (i.e., digit 0), and the first class (normal) from the four datasets (AREM, LP5, ArabicDigits, and Wafer), respectively, to evaluate our experimental results.

We also generate synthetic datasets by adding random noises to the original datasets to further validate the soundness of the importance of variables produced by our module. Specifically, we sample the noise data from normal distribution and add to the variables separately and text the model with the contaminated data. Intuitively, if the variable is important when adding the noise to it, it should dramatically influence the decision-making by the classifier. In other words, if the influence of the variable is significant, then the accuracy will fall significantly when the variable is affected by the noise and vice versa.

We use *accuracy* as the evaluation metric, which is commonly used as the sole performance indicator in MTSC. We tested the model for five times on the datasets and calculate the average accuracy and the standard deviation. Since we concentrate on the effectiveness of the proposed module, accuracy suffice to suggest the quality of different methods' results.

4.4 Results

Performance on real datasets Tables 2–5 show our experimental results on the four datasets. Our module can be implemented to be combined with various models without much extra efforts. Through the utilization of our module, we can obtain the importance of the variables quantitatively. To make the results more convenient to understand, we execute softmax on the results. Thus, all the weights are in [0,1], and the sum is 1. We select a specific class and use the synthetic dataset and test the importance given by the module, i.e., we add noise manually to the variables separately; then, we explore the accuracy changes thanks to the noise. We present the accuracy on the specific dataset before and after adding noise to the variable separately. Intuitively, a more critical variable bears a great change in classification accuracy. Our results on the Wafer dataset (shown in Table 2) show a classification accuracy is 99.87% on the selected class. The results produced by the module indicate that the 5th and 6th variables are most important and least important, respectively. As we repeat the experiments five times, we also give the standard deviations of the average accuracy with and without the noise.

Validating explanation ability We tested adding noises to the variables manually, which results in drastic changes in the accuracy on the 5th variable while little change on the 6th variable. The results indicate that noises can influence the crucial variables, and the regulation has well-matched the hypothesis. On the

Table 2. Experimental results on Wafer dataset

Accuracy (sd) (w/o noise)	99.87% (0.68%)					
Variable id	1	2	3	4	5	6
Importance of variables	0.101	0.129	0.158	0.261	0.276	0.075
Accuracy (%) (w/ noise)	99.01	96.38	96.38	96.38	94.38	99.75
Standard Deviation (w/noise)	0.05	0.39	0.38	0.42	0.76	0.04
Accuracy change (Δ%)	0.86	3.49	3.49	3.49	4.9	0.12

Table 3. Experimental results on LP5 dataset

Accuracy (sd) (w/o noise)	96.15% (2.58%)					
Variable id	1	2	3	4	5	6
Importance of variables	0.225	0.157	0.186	0.082	0.148	0.202
Accuracy (%) (w/ noise)	46.15	88.46	76.92	92.30	92.30	73.07
Standard Deviation (%) (w/noise)	2.81	2.59	2.57	2.96	2.28	2.70
Accuracy change (Δ%)	50.00	7.69	19.23	3.85	3.85	23.08

Table 4. Experimental results on AREM dataset

Accuracy (sd) (w/o noise)	100% (0)						
Variable id	1	2	3	4	5	6	7
Importance of variables	0.215	0.150	0.060	0.115	0.174	0.229	0.057
Accuracy (%) (w/ noise)	0.00	0.00	71.43	71.43	14.28	0.00	71.43
Standard Deviation (%) (w/noise)	6.32	4.90	6.46	7.07	4.99	6.91	6.18
Accuracy change (Δ%)	100.00	100.00	28.57	28.57	85.71	100.00	28.57

Table 5. Experimental results on ArabicDigits dataset

Accuracy (sd) (w/o noise)	100% (0)						
Variable id	1	2	3	4	5	6	7
Importance of variables	0.071	0.001	0.074	0.042	0.076	0.386	0.024
Accuracy (%) (w/ noise)	98.83	99.10	98.83	99.10	97.54	29.24	98.83
Standard Deviation (%) (w/noise)	0.35	0.21	0.25	0.24	0.34	0.33	0.36
Accuracy change (Δ%)	0.27	0.00	0.27	0.00	1.56	69.86	0.27
Variable id	8	9	10	11	12	13	
Importance of variables	0.100	0.030	0.110	0.031	0.052	0.002	
Accuracy (%) (w/ noise)	85.10	98.04	61.72	98.43	98.83	98.83	
Standard Deviation (%) (w/noise)	0.38	0.33	0.49	0.46	0.91	0.45	
Accuracy change (Δ%)	14.00	1.06	37.38	0.67	0.27	0.27	

other datasets, the results are similar. Hence, we can say that the importance the module obtains is convincing.

It is worth noting that the accuracy of all the datasets given in Tables 2–5 is quite high. That is because we select the specific class to evaluate our results. We have found that a satisfying performance is crucial to obtain reasonable results. Because when the classification accuracy is high, that means the model focuses on the right variables.

Table 6. Training time consumption comparison between the model with the proposed module and without the proposed module on Wafer dataset

	Average Training Time (s)	Standard Deviation (s)
Without the module	32.95	3.9
With the module	36.02	2.76

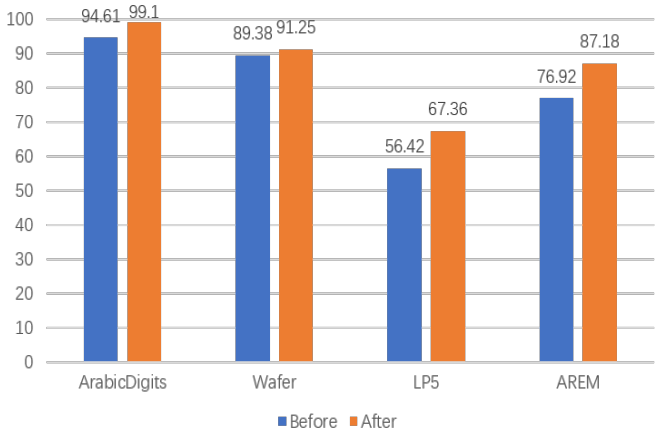

Fig. 2. Accuracy comparison between the original model and the model combined with our proposed module

Impact on performance Besides explanation ability, our module can improve the performance of the baseline models on the respective datasets. Specifically, Fig. 2 suggests that all the selected classifiers achieve better classification accuracy on each dataset, which indicates the superiority of the proposed module. The model better harnesses the temporal features as it updates the feature-maps (extracted by the backbone) twice according to the self-attention of the feature-maps and the attention between feature-maps and input. The module helps the original model to fuse various levels of feature-maps and improve the classifier's performance. The accuracy given in Fig. 2 is the average accuracy of all the classes instead of a specific class. Hence the accuracy is different from the results of the tables shown in the previous contents.

Besides, the proposed module does not significantly cause extra time consumption. To indicate that, we record the training time consumption, which is shown in Table 6, on the Wafer dataset. The corresponding method we use on the Wafer dataset is the LSTM with three layers. We train the model on Intel Core i7-8550 with 16GB RAM instead of GTX 3090 GPU, because the GPU is too powerful to demonstrate the time consumption difference. In Table 6, we can see the average training time increased by 9%, thus we can say the proposed module is efficient.

5 Conclusion and Future Work

We propose an explanation module to explore the importance of the variables for multivariate time series classification. Our module can be easily plugged into the existing models and quantitatively figure out the importance of the variables for classification. Our extensive experiments demonstrate its effectiveness. Besides, the module can improve the model's performance further, as it is beneficial for leveraging the temporal information of the input. We also provide some tricks for implementing our module. However, the module is not feasible for finding the important time interval for the outcome, which leads to limitations. In the future, we plan to refine the module to make it feasible for figuring out the importance of both temporal aspects and variable aspects.

References

1. Ancona, M., Oztireli, C., Gross, M.: Explaining deep neural networks with a polynomial time algorithm for shapley value approximation. In: International Conference on Machine Learning. pp. 272–281. PMLR (2019)
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
3. Bai, L., Yao, L., Kanhere, S.S., Wang, X., Yang, Z.: Automatic device classification from network traffic streams of internet of things. In: 2018 IEEE 43rd Conference on Local Computer Networks (LCN). pp. 1–9. IEEE (2018)
4. Bai, L., Yao, L., Wang, X., Kanhere, S.S., Xiao, Y.: Prototype similarity learning for activity recognition. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 649–661. Springer (2020)
5. Borovykh, A., Bohte, S., Oosterlee, C.W.: Conditional time series forecasting with convolutional neural networks. arXiv preprint arXiv:1703.04691 (2017)
6. Cao, J., Li, Z., Li, J.: Financial time series forecasting model based on ceemdan and lstm. Physica A: Statistical Mechanics and its Applications **519**, 127–139 (2019)
7. Chattopadhay, A., Sarkar, A., Howlader, P., Balasubramanian, V.N.: Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 839–847. IEEE (2018)
8. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
9. Choi, E., Bahadori, M.T., Kulas, J.A., Schuetz, A., Stewart, W.F., Sun, J.: Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. arXiv preprint arXiv:1608.05745 (2016)
10. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
11. Esling, P., Agon, C.: Time-series data mining. ACM Computing Surveys (CSUR) **45**(1), 1–34 (2012)
12. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data Mining and Knowledge Discovery **33**(4), 917–963 (2019)
13. Gamboa, J.C.B.: Deep learning for time-series analysis. arXiv preprint arXiv:1701.01887 (2017)

14. Gao, S., Cheng, M.M., Zhao, K., Zhang, X.Y., Yang, M.H., Torr, P.H.: Res2net: A new multi-scale backbone architecture. IEEE transactions on pattern analysis and machine intelligence (2019)
15. Goldberger, A.L., Amaral, L.A., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.: Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. circulation **101**(23), e215–e220 (2000)
16. Guo, B., Mu, Y., Wang, F., Dong, S.: Effect of periodic light color change on the molting frequency and growth of litopenaeus vannamei. Aquaculture **362**, 67–71 (2012)
17. Guo, B., Wang, F., Li, Y., Dong, S.: Effect of periodic light intensity change on the molting frequency and growth of litopenaeus vannamei. Aquaculture **396**, 66–70 (2013)
18. Han, M., Liu, X.: Feature selection techniques with class separability for multivariate time series. Neurocomputing **110**, 29–34 (2013)
19. Han, Z., Zhao, J., Leung, H., Ma, K.F., Wang, W.: A review of deep learning models for time series prediction. IEEE Sensors Journal (2019)
20. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
21. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
22. Hoermann, S., Bach, M., Dietmayer, K.: Dynamic occupancy grid prediction for urban autonomous driving: A deep learning approach with fully automatic labeling. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 2056–2063. IEEE (2018)
23. Hsieh, T.Y., Wang, S., Sun, Y., Honavar, V.: Explainable multivariate time series classification: A deep neural network which learns to attend to important variables as well as time intervals. In: Proceedings of the 14th ACM International Conference on Web Search and Data Mining. pp. 607–615 (2021)
24. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018)
25. Karevan, Z., Suykens, J.A.: Transductive lstm for time-series prediction: An application to weather forecasting. Neural Networks **125**, 1–9 (2020)
26. Karim, F., Majumdar, S., Darabi, H., Harford, S.: Multivariate lstm-fcns for time series classification. Neural Networks **116**, 237–245 (2019)
27. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25**, 1097–1105 (2012)
28. Lea, C., Vidal, R., Reiter, A., Hager, G.D.: Temporal convolutional networks: A unified approach to action segmentation. In: European Conference on Computer Vision. pp. 47–54. Springer (2016)
29. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
30. Lipton, Z.C., Berkowitz, J., Elkan, C.: A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019 (2015)
31. Lipton, Z.C., Kale, D.C., Elkan, C., Wetzel, R.: Learning to diagnose with lstm recurrent neural networks. arXiv preprint arXiv:1511.03677 (2015)
32. Major, P., Thiele, E.A.: Seizures in children: Laboratory. Pediatrics in review **28**(11), 405 (2007)

33. Malhotra, P., TV, V., Vig, L., Agarwal, P., Shroff, G.: Timenet: Pre-trained deep recurrent neural network for time series classification. arXiv preprint arXiv:1706.08838 (2017)
34. Olszewski, R.T.: Bobski's world. [EB/OL] (2012), http://www.cs.cmu.edu/ bobski/
35. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International conference on machine learning. pp. 1310–1318. PMLR (2013)
36. Sagheer, A., Kotb, M.: Time series forecasting of petroleum production using deep lstm recurrent networks. Neurocomputing **323**, 203–213 (2019)
37. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural networks **61**, 85–117 (2015)
38. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE international conference on computer vision. pp. 618–626 (2017)
39. Siami-Namini, S., Tavakoli, N., Namin, A.S.: A comparison of arima and lstm in forecasting time series. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). pp. 1394–1401. IEEE (2018)
40. Yang, C., Jiang, W., Guo, Z.: Time series data classification based on dual path cnn-rnn cascade network. IEEE Access **7**, 155304–155312 (2019)
41. Yoon, H., Shahabi, C.: Feature subset selection on multivariate time series with extremely large spatial features. In: Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06). pp. 337–342. IEEE (2006)
42. Yoon, J., Jordon, J., van der Schaar, M.: Invase: Instance-wise variable selection using neural networks. In: International Conference on Learning Representations (2018)
43. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision. pp. 818–833. Springer (2014)
44. Zheng, Y., Liu, Q., Chen, E., Ge, Y., Zhao, J.L.: Time series classification using multi-channels deep convolutional neural networks. In: International conference on web-age information management. pp. 298–310. Springer (2014)