C02029: Doctor of Philosophy
CRICOS Code: 0000
32903 PhD Thesis: Analytics
Feb. 2024

A Study on

Deep Learning Methods for Multivariate Time Series Classification

Chao Yang

School of Computer Science
Faculty of Engg. & IT
University of Technology Sydney
NSW 2007, Australia

A Study on
Deep Learning Methods for Multivariate Time Series Classification

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy
in
Analytics

by

Chao Yang

to

School of Computer Science
Faculty of Engineering and Information Technology
University of Technology Sydney
NSW 2007, Australia

Feb. 2024

Certificate of Original Authorship

I, *Chao Yang* declare that this thesis, submitted in fulfillment of the requirements for the award of Doctor of Philosophy, in the *School of computer science*, *Faculty of Engineering and Information Technology* at the University of Technology Sydney. This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis. This document has not been submitted for qualifications at any other academic institution. This research is supported by the Australian Government Research Training Program.

SIGNATURE: Production Note: Signature removed prior to publication.

DATE: 7th Feb., 2024

PLACE: Sydney, Australia

Acknowledgments

I would like to acknowledge everyone who gave me passion, encouragement, and blessing to help me finish this thesis.

First, I would like to say the most sincere thanks to my supervisor, Dr. Xianzhi Wang, who has provided me with guidance and suggestions not only for my Ph.D. career but also for my future plans. He always stands with me, shares his suggestions from my perspective, and pays more attention to my gains and losses than his own. His kindness and patience make me feel like working with a friend instead of a supervisor. I more than appreciate his providing me with an amazing and unbelievable environment for doing research work.

I would like to thank my colleagues, Mr. Zihao Li, Miss Yakun Chen, and Miss Ruotong Hu, for their help. This is my first time living and studying overseas for more than one year. I have encountered many unexpected challenges in living and research, and I can not imagine how to handle these without their help.

I would like to thank my friends, Dr. Xiaoming Chen and Dr. Liang Shen, for their help. We finished our bachelor's degrees at the same University and have been in contact for about seven years. Although our research interests are quite different, we never stop sharing our encouragements with each other and trying to become the source of passion and power for each other. I sincerely cherish this friendship and wish it can last forever.

I am grateful to the University of Technology Sydney for providing technical and device support to help me finish my research work conveniently. I also want to thank the China Scholarship Council for its financial support.

I would like to thank Dr. Angela for her help. I appreciate her providing the tutor jobs for me to support my living expenses. Although she is not my supervisor or co-supervisor, she generously provided assistance to me both in research and living aspects.

I would like to thank my wife, Miss Xin Wang, for her help during my whole Ph.D. journey in Sydney. It's hard to describe my gratitude to her. Although she is not good at English, she tried her best to order small cakes for me when I felt upset. She gave me so many surprises and made me feel happy when I was living in a completely strange environment. We have been in love for seven years, and most of the time, we were not living in the same city. But we always trust each other and believe we will have a good result. It is a big fortune for me to have her accompanied both in my life in Sydney and my life in the future.

Last but not least, I would like to thank my parents, Mr. Minle Yang and Dr. Fang Wang, for their help in my whole life. Their love and support are the most important for

me to finish all that I have done.

List of Publications

Related to the Thesis :

1. **Yang, C.**, Wang, X., Yao, L., Long, G., Jiang, J., & Xu, G. In ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 3308-3312), IEEE.

2. **Yang, C.**, Wang, X., Yao, L., Long, G., Jiang, J., & Xu, G. Attentional Gated Res2Net for Multivariate Time Series Classification. Neural Processing Letters, 1-25.

3. **Yang, C.**, Wang, X., Yao, L., Jiang, J., & Xu, G. An Explanation Module for Deep Neural Networks Facing Multivariate Time Series Classification. In AI 2021: Advances in Artificial Intelligence: 34th Australasian Joint Conference, AI 2021, Sydney, NSW, Australia, 2022, Proceedings (pp. 3-14). Cham: Springer International Publishing.

4. **Yang, C.**, Wang, X., Chen, Y., Li Z., Yao, L., Jiang, J., Long G., & Xu, G. Positional Embedding May Not Be All You Need: An Empirical Study on Positional Embedding for Transformer. Neurocomputing and Applications, 2024. Under review.

5. **Yang, C.**, Chen, Y., Li Z., & Wang, X. Exploring the Effectiveness of Positional Embedding on Transformer-based Architectures for Multivariate Time Series Classification. In International Conference on Advanced Data Mining and Applications (pp. 34-47). Cham: Springer Nature Switzerland.

6. **Yang, C.**, Wang, X., Jiang, J., Long G., & Xu, G. From Time Series to Multi-Modality: Classifying Multivariate Time Series via Both 1D and 2D Representations. In International Conference on Advanced Data Mining and Applications (pp. 19-33). Cham: Springer Nature Switzerland.

7. **Yang, C.**, Wang, X., Yao, L., Jiang, J., Long G., & Xu, G. Dyformer: A Dynamic Transformer-based Architecture for Multivariate Time Series Classification. Information Sciences, 656, 119881.

OTHERS :

8. Chen, Y., Li, Z., **Yang, C.**, Wang, X., Long, G., & Xu, G. Adaptive Graph Recurrent Network for Multivariate Time Series Imputation. In Neural Information Processing: 29th International Conference, ICONIP 2022, Virtual Event, November 22–26, 2022, Proceedings, Part V (pp. 64-73). Singapore: Springer Nature Singapore.

9. Li, Z., Wang, X., **Yang, C.**, Yao, L., McAuley, J., & Xu, G. Exploiting Explicit and Implicit Item Relationships for Session-based Recommendation. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining (pp. 553-561).

10. Wang, J., Guo, Z., **Yang, C.**, Wang W., & Li X. Multi-Scale Hybrid Fusion Network for Mandarin Audio-Visual Speech Recognition. In 2023 IEEE International Conference on Multimedia and Expo (ICME) (pp. 642-647). IEEE.

11. Wang, W., Guo, Z., **Yang, C.**, Wang, J., Jiang, S., & Zhang, T. Joint-semantics Multi-Similarity Hashing for Cross-modal Retrieval. International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2024. Accepted.

12. Wang, J., **Yang, C.**, Guo, Z., Li, X., & Wang, W. (2023, October). An End-to-End Mandarin Audio-Visual Speech Recognition Model with a Feature Enhancement Module. In 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 572-577). IEEE.

Abstract

Multivariate time series classification is a crucial task with broad applications in areas such as finance, medicine, and engineering. It is inherently challenging to classify multivariate time series as it requires considering temporal patterns and inter-relations among multiple variables simultaneously. Deep learning has recently gained popularity in computer vision, natural language processing, and data mining thanks to its advantages in capturing complicated, nonlinear relations from massive data. However, existing methods can not effectively leverage the characteristic of multivariate time series data to provide satisfactory solutions for classifying sequences with different classes. They can not fully leverage the multi-level temporal dependencies, lack structure adaptation on various time series sequences, ignore the inherent frequency information, and suffer for the explainability. In this thesis, I will introduce deep learning methods to leverage the inherent temporal information of the time series sequences for accurate multivariate time series classification, specifically via 1) feature-map-wise attention, 2) dynamic architecture, 3) explanation module, 4) utilization of 2D representations with frequency information, 5) exploration on positional embedding. The proposed methods are evaluated using the datasets from the UEA time series classification repository and UCI data resource, which contain a large multitude of public multivariate time series classification datasets covering a wide range of applications including, EEG signal classification, activity recognition, disease diagnosis, etc. These datasets concern different domains and reflect diverse data characteristics in terms of sequence lengths and variable numbers for comprehensive evaluation. Experimental results demonstrate that the proposed methods are superior to the traditional machine learning methods and recently proposed deep learning methods. In summary, in this thesis, I propose five deep learning methods for classifying multivariate time series. The detailed contents are as follows:

1. A novel convolutional neural network architecture called Attentional Gated Res2Net for multivariate time series classification. The proposed model uses hierarchical residual-like connections to achieve multi-scale receptive fields and capture multi-granular temporal information. The gating mechanism enables the model to consider the relations between the feature maps extracted by receptive fields of multiple sizes for information fusion. Further, Two types of attention modules, channel-wise attention and block-wise attention, are proposed to leverage the multi-granular temporal patterns better. The experimental results on 14 benchmark multivariate time series datasets show that the proposed model outperforms

several baselines and state-of-the-art methods by a large margin. Besides, the proposed model improves the performance of existing models when used as a plugin.

2. A novel dynamic transformer-based architecture called Dyformer to address the limitations of traditional transformers in multivariate time series classification. Transformer is promising for time series classification, but as a generic approach, they have limited capability to effectively capture the distinctive characteristics inherent in time series data and adapt to diverse architectural requirements. Dyformer incorporates hierarchical pooling to decompose time series into subsequences with different frequency components. Then, it employs Dyformer modules to achieve adaptive learning strategies for different frequency components based on a dynamic architecture. Further, feature-map-wise attention is implemented to capture multi-granular temporal patterns and a joint loss function to facilitate model training. Extensive experiments conducted on 30 benchmark datasets show the proposed model outperforms a multitude of state-of-the-art methods and baselines. Dyformer also copes well with limited training samples when pre-trained.

3. A summary of six different Transformer-based variants designed for time series-related tasks and explore the impact of positional embedding on the vanilla Transformer and variants using 30 public multivariate time series classification datasets and 4 time series forecasting datasets. The experimental results demonstrate that the positional embedding has a positive impact on the performance of the vanilla Transformer in both classification and forecasting, but a negative impact on the Transformer-based variants in the classification and a look-back window size-dependent impact on the variants in the forecasting. Additionally, the impact of spurious sequential order information is investigated on these models by inverting the time series data without positional embedding. The experimental results demonstrate that the variants' performance degrades more drastically than the vanilla Transformer, suggesting the variants can automatically capture position information, rendering positional embedding redundant. The findings highlight the importance of carefully considering the utilization of positional embedding in Transformer-based models and suggest that its effectiveness may vary based on the specific architecture and task.

4. A novel method for classifying multivariate time series using both temporal and frequency information. The proposed approach involves transforming the time series into spectrograms using the Short-Time Fourier Transform (STFT), which generates a 2D representation containing frequency components and their temporal positions. Three different window sizes are used to generate multiple spectrograms with varying frequency and temporal resolutions for each variable. This creates a new modality in addition to the 1D time series, allowing us to convert the multivariate time series classification into a multi-modality data classification task and borrow powerful backbones from computer vision fields. A dual-stream network is constructed based on the ResNet architecture to leverage both 1D and 2D rep-

resentations for accurate multivariate time series classification. The experiments conducted on 30 public datasets demonstrate that the proposed method outperforms several competitive baseline methods, achieving state-of-the-art performance.

5. A novel explanation module pluggable into existing deep neural networks to explore variable importance for explaining multivariate time series classification. The proposed module is evaluated using popular deep neural networks on both real-world and synthetic datasets to demonstrate its effectiveness in generating explanations for multivariate time series classification. The experiments also show the module improves the classification accuracy of existing models due to the comprehensive incorporation of temporal features.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Multivariate time series data is a group of chronologically recorded and synchronous variables. Real-world applications relying on the analysis of time series include speech recognition [14, 117, 130], gesture recognition [36, 123, 131], health care [15, 16, 138], etc. Compared with other types of sequential data such as language or video, multivariate time series is recorded chronologically while each time point saves some scalars. Besides the inter-relations among multiple variables, it contains temporal variation reflecting the inherent properties of time series, such as continuity, periodicity, trend, etc. Specifically, the temporal information indicates the trends and the pulses across multiple time steps, while the frequency information represents the periodical regulations. Additionally, the temporal and frequency information demonstrates multi-range characteristics-they both contain long-term and short-term information. Hence, it is inherently challenging to classify multivariate time series as it requires simultaneously incorporating temporal patterns and inter-relations among multiple variables to learn the differences of the time series sequences with various classes for obtaining accurate results. Although current studies propose some methods for multivariate time series classification, they either can not effectively leverage the specific characteristics of the time series, ignore inherent information from the frequency domain, or lack explainability and generalization.

1.2 Existing Efforts and Limitations

Traditional methods for time series classification include distance-based models (e.g., k-nearest neighbors) and feature-based models (e.g., random forest [8] and support vector machine [69]). These models highly rely on manually-defined features, which are heuristic and task-dependent [6]. Also, it takes the expertise and considerable time of domain experts to design such features. Furthermore, conventional machine learning (ML) techniques have limitations in processing high-dimension data and representing complicated functions efficiently [10].

In recent years, many deep learning methods have been proposed to address the critical challenges for multivariate time series classification. They can be categorized as convolutional neural net-based methods, recurrent neural net-based methods, transformer-based methods, and ensemble methods. Convolutional neural net-based methods such as Omni-Scale Convolutional Neural Network (OS-CNN) [109] and Inception-time [43] can capture short-term dependencies but is challenging to harness long-term dependencies. Recurrent neural net-based methods such as Long Short-Term Memory (LSTM) [56] and Gated Recurrent Unit [24] can leverage long-term dependencies but is time-consuming for training and inference. Transformer-based methods such as Time Series Transformer [151] and Gated Transformer [87] can leverage multi-scale temporal dependencies, but its attention mechanism based on dot product can not fully leverage the characteristics of the time series. Ensemble models such as LSTM-FCN [71] combine different methods to take advantage of their respective strengths, but the construction manner remains an open problem.

Furthermore, existing deep learning-based classifiers rely solely on temporal information while disregarding clues from the frequency perspective, making them impossible to leverage the inherent temporal variations of the time series sufficiently. Additionally, current methods can only update their parameters instead of their architectures during training, which limits representation learning and models' generalization ability.

Besides the above limitations, the black-box characteristic of deep learning models impedes humans from obtaining insights into the internal regulation and decisions made by classifiers. Although some studies have sought explanations for classification problems [1, 144], they mostly design separate architectures that are specific to certain deep neural-network types or re-design a new backbone architecture leading to extra efforts.

1.3 Our Solutions

This thesis aims to address the challenges in multivariate time series classification. An attentional architecture is first proposed to leverage multi-scale temporal information sufficiently. The Short-time Fourier Transform is then implemented to extract frequency information and construct a dual-steam architecture to harness both temporal and frequency information. To realize adaptive learning strategies for different frequency components, a time series decomposition method is designed to decompose the time series into several subsequences with different frequency components. Then, a dynamic transformer-based architecture that can adjust its structure during training and inference is introduced. Additionally, a comprehensive empirical study on the effectiveness of positional embedding on the current transformer-based variants is conducted to explore the necessity of positional embedding under different scenarios. Furthermore, a pluggable explanation module is proposed to explore the importance of the variables for the classifiers.

1.4 Thesis Overview

We introduce the existing efforts and challenges in Chapter 2 and detail our proposed methods in the following chapters. Chapter 3 introduces the proposed attention architecture, Chapter 4 introduces the proposed method to harness the frequency information of the time series, Chapter 5 introduces a dynamic transformer-based architecture to enable an adaptive information flow and learning strategy, Chapter 6 introduces an empirical study on the positional embeddings for transformer-based methods, and Chapter 7 introduces a pluggable explanation module. Finally, Chapter 8 gives the conclusions. Specifically, this thesis is organized in the following structure:

- **Chapter 2**: we review the existing methods to address the challenges in the multivariate time series classification. We first introduce the traditional machine learning methods based on statistical theory. Then, we introduce deep learning-based methods such as convolutional neural net-based methods and recurrent neural net-based methods. We additionally introduce the attentional mechanisms developed for time series data and the time series decomposition to extract different components of the time series. At last, we introduce several methods to enable the explainability of the classifiers.

- **Chapter 3**: we present a novel attnetional architecture called Attentional Gated Res2Net for multivariate time series classification. Our model uses hierarchical residual-like connections to achieve multi-scale receptive fields and capture multi-granular temporal information. The gating mechanism enables the model to consider the relations between the feature maps extracted by receptive fields of multiple sizes for information fusion. Further, we propose two types of attention modules, channel-wise attention and block-wise attention, to better leverage the multi-granular temporal patterns.

- **Chapter 4**: we propose a novel method for classifying multivariate time series leveraging both temporal and frequency information. We first apply Short-Time Fourier Transform (STFT) to transform time series into spectrograms, which contain a 2D representation of frequency components and their temporal positions. In particular, for each variable, we generate spectrograms with varying frequencies and temporal resolutions under different window sizes. The transformation essentially adds a new modality to 1D time series and converts the multivariate time series classification into a multi-modality data classification task, making it possible to bring powerful backbones from computer vision fields to solve the time series classification problem. We then construct a dual-stream network based on the ResNet architecture that takes in both 1D and 2D representations for accurate multivariate time series classification.

- **Chapter 5**: we propose a novel dynamic transformer-based architecture called Dyformer to realize adaptive learning strategies for different frequency components. Dyformer incorporates hierarchical pooling to decompose time series into subsequences with different frequency components. Then, it employs Dyformer modules to realize a dynamic architecture and information flow facing different frequency components. Furthermore, we introduce feature-map-wise attention mechanisms to capture multi-scale temporal dependencies and a joint loss function to facilitate model training.

- **Chapter 6**: we summarize six different Transformer-based variants designed for time series-related tasks and explore the impact of positional embedding on the vanilla Transformer and variants using thirty public multivariate time series classification datasets and four time series forecasting datasets. Our experimental results demonstrate that the positional embedding has a positive impact on the vanilla Transformer's performance in both classification and forecasting. In contrast, it has a negative impact on the Transformer-based variants in the classification and an input sequence length-dependent impact on the variants in the forecasting. Additionally, we investigate the impact of spurious sequential order information on these models by inverting the time series data without positional embedding. The experimental results demonstrate that the variants' performance degrades more drastically than the vanilla Transformer, suggesting the variants can automatically capture position information, rendering positional embedding redundant. Our findings reveal the varying effectiveness of positional embedding on different model architectures and tasks, highlighting the significance of using positional embedding cautiously in Transformer-based models.

- **Chapter 7**: we propose a novel explanation module pluggable into existing deep neural networks to explore variable importance for explaining multivariate time series classification. We evaluate our module with popular deep neural networks on both real-world and synthetic datasets to demonstrate its effectiveness in generating explanations for multivariate time series classification. Our proposed module can also improve the classification accuracy of existing models due to the comprehensive incorporation of temporal features.

- **Chapter 8**: we summarize this thesis along with several future directions.

1.5 Key Contributions and Impact

The main contributions of this thesis include novel architectures to leverage temporal dependencies and frequency information, an empirical study on the positional embedding's impact on current transformer-based methods, and the explanation module. Our efforts address the inherent challenges we introduced and realize precious and interpretable time series classification.

- **Attentional architecture**. The proposed Attentional Gated Res2Net can capture temporal dependencies over various ranges and exploit the inter-variable relations to achieve high performance on time series of various lengths, making it feasible for various real-world challenges.

- **Dual-Stream architecture**. We employ Short-Time Fourier Transform (STFT) with varying window sizes to generate 2D representations containing frequency components and corresponding temporal positions at multiple resolutions and then construct a dual-stream architecture based on ResNet to leverage both temporal and frequency information.

- **Time series decomposition method**. We design a hierarchical pooling layer to decompose time series into several subsequences with different frequency components. Hierarchical pooling differs from previous efforts in leveraging all the frequency components in the time series without information loss, offering the potential to achieve more accurate multivariate time series classification.
- **Dynamic architecture**. We propose a dynamic transformer-based architecture called Dyformer, which varies its learning strategies by adapting its structures to the specific characteristics of time series sequences with various frequency components for multivariate time series classification.
- **Empirical study on positional embedding**. We summarize six kinds of transformer-based variants designed for time series-related tasks and explore the effectiveness of the positional embedding on them as well as the vanilla transformer to explore the scenarios in which positional embedding positively impacts the model's performance to facilitate researchers and practitioners in making informed decisions on whether to incorporate positional embedding in their models.
- **Explanation module**. We propose an explanation module that can be plugged into existing popular deep neural networks to infer the importance of variables in multivariate time series classification automatically. Our module enables the models to leverage the temporal features for achieving better classification accuracy.
- **Extensive experiments**. We conduct extensive experiments on a large number of public datasets to study the behavior and performance of the proposed methods. Experimental results demonstrate the superiority of the proposed methods compared to the competitive baselines and state-of-the-art techniques.

Chapter 2

Related Work

In this chapter, we summarize the current studies proposed for addressing multivariate time series classification-related challenges.

2.1 Traditional Machine Learning Methods

Statistical and traditional machine learning methods have been extensively employed for multivariate time series classification. Distance-based approaches, such as k-Nearest Neighbors (KNN) [112], as well as feature-based methods, including Support Vector Machine (SVM) [152], Time Series Combination of Heterogeneous and Integrated Embedding Forest (TS-CHIEF) [104], Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [85], and Random Convolutional Kernel Transform (ROCKET) [30], have been used. Dynamic time warping (DTW) [112] is the most popular distance function to compensate for possible confounding offset by allowing some realignment of the series that allows KNNN [41, 48] to classify various time series sequences. Depending on the warping path for different dimensions, DTW can be divided into independent warping (DTW_I), dependent warping (DTW_D), and adaptive warping ((DTW_A). SVM, which is the most used feature-based method in recent years, implements kernel functions to map the raw data into the high-dimension space, making finding a classification boundary easier. ROCKET implements a group of convolutional kernels whose parameters are randomly initialized and not updated during training for feature extraction, resulting in a significant efficiency compared to deep neural nets.

ROCKET can be further accelerated by generating more kernels with smaller kernel sizes, called miniROCKET [31]. From the signal processing perspective, the untrained kernels can be recognized as the basis of the transform, which maps the raw time series into other domains to facilitate the classification. However, these methods typically rely on manually crafted features and face difficulties in capturing complex relationships efficiently from high-dimensional data [10].

2.2 Deep Learning Methods

Deep learning methods for MTSC can be categorized into four classes: Convolutional Neural Network-based models, Recurrent Neural Network-based models, cascade models, and Transformer-based models. We introduce more details in the following sections.

2.2.1 Convolutional Neural Network-based models

Existing CNN-related studies either design new CNN-based architecture or directly borrow models from other domains. InceptionTime [43, 66] borrows the idea of Inception Network [107] used in computer vision and uses multiple convolutional kernels with different sizes to realize multi-granular temporal information extraction. ResNet [55] is another example that is modified to make it feasible for time series classification [122]. As such, constructing a very deep convolutional network to facilitate the model's classification capacity is practicable. The fully Convolutional Network (FCN) [21] is proposed in the computer vision field, which is constructed using convolutional layers only. As the convolutional layers have parameter sharing and locally connected characteristics, it is much more efficient than other convolutional neural networks. In this case, FCN [70] is considered for time series classification as an efficient and effective classifier. Multi-channel convolutional network [156] is proposed for time series classification, which regards multivariate as the input of multi-channels to harness the inter-relationships of the variables and the temporal features. Temporal Convolutional Network (TCN) [115] is first proposed for the time series forecasting task. It implements casual convolutional kernels to capture long-term temporal dependencies. Due to its capacity to harness temporal features, it is also used for multivariate time series classification [76]. Time series attentional prototype network (Tapnet) [155] is a specifically designed convolutional-based architecture for time series classification. It implements the attentional mechanism in the convolutional neural network to harness the multi-range temporal features. Multi-scale convolutional neural network (MCNN) [28] is another convolutional-based model designed for time series classification. It builds a multi-branch layer to harness multi-scale temporal features. CNNs can handle temporal patterns and inter-variable relationships. However, as it only focuses on the temporal information that lies in the receptive field, it faces challenges in capturing long-range temporal dependencies [86].

2.2.2 Recurrent Neural Network-based models

Recurrent neural networks feed the current output information into the next input data to harness the temporal dependencies. It is first used in natural language processing tasks such as machine translation, emotion classification, etc. Well-known recurrent neural networks include Gated Recurrent Unit (GRU) [24] and Long Short-Term Memory (LSTM) [56], which employ gate mechanisms to capture long-range temporal patterns. As the time series data contains temporal information, recurrent neural networks are naturally feasible for classifying time series sequences. Models based on LSTM and GRU [19, 39, 72, 78, 145] are popularly used for time series classification. Highway network [105] is another RNN-based method that enables the information flow through different layers. Existing highway network-based methods [64, 110] have been implemented to solve time series-related challenges. Nevertheless, RNN-based methods are time-consuming for training and inference due to their recurrent architectures. And they are likely to forget the temporal information facing extremely long time series sequences.

2.2.3 Cascade models

To address the shortcomings of convolutional neural nets and recurrent neural nets, existing efforts have tried to construct cascade models by combining them. A hybrid architecture is proposed for time series classification [38] through parallelly constructing convolutional neural nets and recurrent neural nets. Another cascade model [140] constructs the convolutional neural network and the recurrent neural network in a series manner. It borrows the idea of the Inception network via stacking multiple convolutional kernels with multiple sizes to capture multi-granular temporal features to enhance performance. LSTM-FCN [71] constructs LSTM and FCN parallelly and implements an attentional layer called Squeeze-and-excitation networks (SENet) [60] to improve the models' capacity. It demonstrates a strong baseline facing various scenarios. Thanks to its performance, existing works have tried to use it to address real-world challenges that are related to time series data [7, 74].

2.2.4 Transformer-based models

Transformer [113] first emerged for natural language processing [33]. It implements self-attention operation to harness both long-term and short-term temporal information, hence can combine the merits of CNNs and RNNs. But until now, transformers have rarely been applied to MTSC. Gated transformer [87] is designed in a two-tower manner to harness the channel-wise and step-wise interrelationships, respectively. It then implements a gate to fuse the information from two towers. AutoTransformer [97] uses network architecture search technology to automatically construct the optimal structure of the model facing various time series datasets to make it more generalizable. Multi-modal fusion transformer [68] uses Gramian Angular Field (GAF) to convert time series into 2D representations, then builds a dual-path transformer architecture to harness both 1D representations and 2D representations. Besides, Recent works that are designed for time series representation learning regard multivariate time series classification as a downstream task. Representative models include Task-Aware Reconstruction for Time-Series Transformer (TARNet) [27] and Time Series Transformer (TST) [151]. TST uses the same architecture as the vanilla transformer. It randomly masks some time steps and then guides the model to recover the time series sequences to learn the representations. TARNet uses the downstream task to guide the training to make the transformer learn the important representations according to the specific task. It achieves promising classification accuracy on several public datasets.

2.3 Time Series Decomposition

Traditional time series methods decompose time series into trend, seasonal, and bias terms [96]. Recent deep learning methods borrow the ideas for representation learning. For example, Autoformer [136] designs a deep learning-based block to simulate the decomposition process. The block eliminates the long-term trend-cyclical part and highlights the seasonal parts for representation learning. COST [125] employs two types of blocks, Trend Feature Disentangler and Seasonal Feature Disentangler, to obtain trend and seasonal terms, respectively. It then uses contrastive learning to learn the representations of the two terms for downstream tasks. FEDformer [158] decomposes time series and designs a sampling strategy for the frequency domain for seasonal representation learning. Unlike the above, Ts2Vec [147] recurrently decomposes the learned representations for hierarchical contrastive learning. The decomposed representations contain temporal patterns of different granularities, enabling representation learning at various scales.

2.4 Attention Mechanism

The attention mechanism was first used in the seq2seq model for machine translation [24]. A vanilla seq2seq model first feeds the input sequence to an encoder (which consists of multiple recurrent layers) [24] to generate hidden states and outputs. It then collects the hidden states of all the steps to represent the information of the input. An attention mechanism forces the model to learn the weights of hidden states in the decoder part during this process. Thus, the model can focus on a specific region of the input sequence, leading to a significant performance improvement. Recent studies have designed different attention modules and applied them to various domains [61, 126]. Among them, Squeeze-and-Excitation Block Networks (SENet) [60] is widely used for various tasks thanks to its easiness of implementation and the power to enhance the model's performance. SENet works in two steps. First, it uses global average pooling to obtain an information vector of feature maps from different channels. Then, it employs fully connected layers to capture the inter-relations between feature maps to learn the weights of feature maps and highlight the critical information. Many deep learning methods such as LSTM-FCN [71] and Geo-CBAM-CNN [120] for multivariate time series classification have used SENet to enhance the performance. There exist many works for improving the performance of the SENet, including cSENet, sSENet and csSENet [98]. Later, self-attention [113] is proposed and demonstrates significantly better performance compared with other attention modules. Self-attention uses point-wise attention to measure the similarities of the data on all time steps and hence can effectively leverage the temporal information. As point-wise attention can not leverage the rich temporal patterns contained in time series effectively, to make the transformer better harness the temporal features, LogTrans [82] and Block Recurrent Transformer [65] use the convolutional layer or recurrent layer to leverage local temporal information for overcoming the limitations of point-wise attention.

2.5 Explanation Methods

There have been several efforts exploring explanation methods for deep neural networks in various tasks. Some efforts have been made to figure out the effect of the input on the output [79, 149]. Gradient-based methods have been used for exploring the influence of the input changes [101]. However, these methods are only feasible for certain kinds of neural networks, which limits the application scopes.

Another explanation approach is to design a separate architecture for explanation purposes. Some studies [18, 52, 102, 135, 141, 143] 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 [5, 25], 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 [59]: 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.

CHAPTER 3

ATTENTIONAL GATED RES2NET FOR MULTIVARIATE TIME SERIES CLASSIFICATION

3.1 Introduction

Multivariate time series contain temporal information from different sources, hence, measuring the interaction of sources and learning the temporal representations are the keys to realizing accurate multivariate time series classification (MTSC) [9]. Different tasks have different requirements for the classifier, making building a generalized used classifier a challenge. For example, EEG signal-based MTSC can be focused on many different goals such as the recognition of emotion [3, 73], decoding cognitive skills [4], recognition, investigation of sustained attention, detection of sleep disorder, decoding of cognitive tasks in brain-computer-interface, etc. In EEG classification, the performance is sensitive to many parameters such as the number of recording channels, i.e., feature dimension, recording time length, i.e., the number of features, number of individuals in each group, feature extraction method, and classifier's architecture.

Recently, deep learning has gained popularity in computer vision, natural language processing, and data mining thanks to its advantages in capturing complicated, non-linear relations from massive data [80]. Deep neural networks usually stack multiple neural layers for automatic feature extraction and representation learning [57]. Many neural network architectures, such as Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Transformer [113], Long Short-Term Memory (LSTM) [56], and Gated Recurrent Unit (GRU) [24], have been applied for time series analysis. In particular, RNN sends the prior output to the next input layer to facilitate temporal feature extraction; therefore, it takes a long training time and cannot support parallel computation. CNN can extract temporal features and be parallelized during training to fully exploit the power of Graphics Processing Units (GPUs); however, it faces challenges in capturing long-range temporal dependencies and is, therefore, less used for time series classification. Transformer [113] has recently emerged as a promising solution to multivariate time series classification. While the transformer supports both parallel computing and efficient temporal feature extraction, it requires massive parameters for the multiple fully connected layers, making the training extremely time-consuming. Furthermore, the transformer suffers overfitting on small datasets [137] and faces challenges in capturing short-range temporal information [34].

Besides, existing solutions to MTSC commonly require careful adjustments of architectures and parameters to deal with time series of various lengths. This is a critical yet little studied issue in existing time series classification research.

To summarize, among the DL methods, CNN is efficient for training and inferencing but challenging for capturing long dependencies; RNN can effectively learn the temporal representations of long temporal features but is computationally expensive; transformer contains too many parameters, making it easy to prone to overfitting on small size datasets. We aim for accurate MTSC that can adapt to time series of various lengths to address the above deficiencies of existing studies.

To this end, we propose a novel CNN architecture called Attentional Gated Res2Net (AGRes2Net) for MTSC. Our model can overcome the shortcoming of the standard CNN architecture by enabling the extraction of both global and local temporal features. It also has the capability to leverage multi-granular feature maps through channel-wise and block-wise attention mechanisms. In a nutshell, we make the following contributions in this work:

- We propose a novel AGRes2Net architecture for accurate MTSC. Our model can capture dependencies over various ranges and exploit the inter-variable relations to achieve high performance on time series of various lengths, making it feasible for various tasks.

- We propose two attention mechanisms, namely channel-wise attention and block-wise attention, to leverage multi-granular temporal information for tasks with different data characteristics. The former has advantages on datasets with many variables, while the latter can effectively prevent overfitting on datasets with very few variables.

- We conducted extensive experiments on 14 benchmark datasets to evaluate the model. A comparison with several baselines and state-of-the-art methods shows the superior performance of our model. Besides, plugging our model into MLSTM-FCN, a state-of-the-art CNN-RNN parallel model, demonstrates the model's capability to improve existing models' performance.

3.2 Our Approach

We propose Attentional Gated Res2Net for accurate classification of multivariate time series of various lengths. In particular, we incorporate gating and attention mechanisms on top of Res2Net [44], where *gate*s control the information flow across the groups of convolutional filters, and the *attention* module harnesses the feature maps at different levels of granularity.

The overall architecture of AGRes2Net (shown in Figure 3.1) consists of two stages: Convolution and Attention. We illustrate these two stages in the following subsections, respectively.

3.2.1 Convolution stage

We design the convolution stage based on Res2Net [44], a CNN backbone specially designed to achieve multi-scale receptive fields based on group convolution. Group convolution first appeared in AlexNet [77] and significantly reduced the number of parameters in that model. It has since been adopted in many lightweight networks [26, 58] to generate a large number of feature maps with a small number of parameters.

Unlike conventional CNNs, which use a single set of filters to work on all channels, Res2Net includes multiple groups of filters and uses a separate group to handle each subset of channels. These filter groups are connected in a hierarchical, residual-like style, and they work as follows: First, a convolutional layer takes the input data and outputs a feature map for channel expansion. Then, the feature map is split into groups along the channel, generating groups of feature maps, i.e., *input feature map*s. Finally, for each input feature map, a separate group of filters extracts features and generates the corresponding output, i.e., an *output feature map*. In particular, when extracting features from an input feature map, the filter group also takes into account the output of

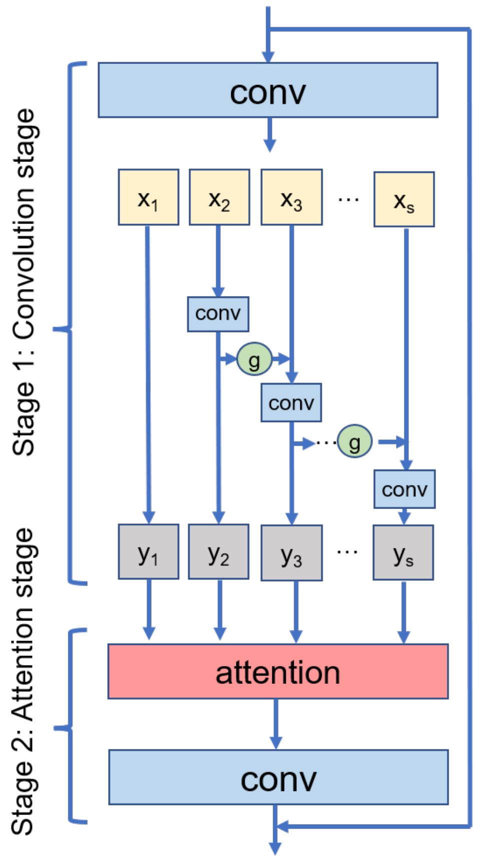


Figure 3.1: The structure of Attentional Gated Res2Net. It consists of two stages: convolution and attention. The convolution stage feeds the input to a convolutional layer for channel expansion and then groups the output along the channel. Each group (except the first) conducts convolution based on its input and its precedent group's output (passed through gates). The attention stage forces the model to consider the temporal information at different levels of granularity. Finally, the network uses a convolutional layer for channel compression and information fusion.

the filter group that comes immediately before it. The whole process repeats until all input feature maps are processed.

Suppose $\mathbf{X}$ is the feature map obtained from channel expansion, and $\mathbf{X}$ is evenly divided into s groups, $\{\mathbf{x}_i\}_{i=1}^{s}$, where $\mathbf{x}_i$ denotes the ith group. Each group contains an input feature map that has the same temporal size but contains only $1/s$ of the channels in $\mathbf{X}$. Let $\mathbf{K}_i$ be the convolution operation. Then, given an input feature map $\mathbf{x}_i$, the convolution output, $\mathbf{y}_i$, is calculated as follows:

$$\mathbf{y}_i = \begin{cases} \mathbf{x}_i & i = 1 \\ \mathbf{K}_i(\mathbf{x}_i) & i = 2 \\ \mathbf{K}_i(\mathbf{x}_i + \mathbf{y}_{i-1}) & 2 < i \leqslant s. \end{cases} \tag{3.1}$$

By feeding the concatenation of all the outputs to a convolutional layer, Res2Net achieves multi-scale receptive fields to facilitate multivariate time series classification.

However, it has difficulty in controlling the information flow between the feature-map groups—at each step, $\mathbf{y}_i$ is always fully sent to the next group regardless of whether it avails or harms the model's performance.

Addressing this limitation is important as it enables to model to control how to weigh the precedent output feature map against the current input feature map in an input-dependent manner. This, in turn, mitigates the problem of vanishing gradients without having to take long delays.

To this end, we introduce the gating mechanism [139] into Res2Net at the convolutional stage to enhance feature extraction. Specifically, in our model (shown in Figure 3.1), all groups of feature maps (except the first) are sent to convolutional layers for feature extraction, and a gating unit lies between each pair of adjacent feature-map groups to control how much information flows from the precedent to the current group. Given a feature-map group (or more specifically, input feature map), $\mathbf{x}_i$, the value of the corresponding gate, $\mathbf{g}_i$, is calculated as follows:

$$\mathbf{g}_i = \tanh(a(\text{concat}(a(\mathbf{y}_{i-1}), a(\mathbf{x}_i)))). \tag{3.2}$$

where a can be either fully-connected or 1-D convolutional layers, $concat$ is the concatenation operation, and $tanh$ is the activation function commonly used for gates.

Note that, we only use the precedent output feature map $\mathbf{y}_{i-1}$ and the current input feature map $\mathbf{x}_i$ to calculate the gate—this is different from the gating mechanism in [139]. More specifically, we omit the undivided feature map $\mathbf{X}$ as it contains redundant information and does not significantly improve the performance. Eventually, after the convolution stage, we obtain $\mathbf{y}_i$ as follows:

$$\mathbf{y}_i = \begin{cases} \mathbf{x}_i & i = 1 \\ \mathbf{K}_i(\mathbf{x}_i) & i = 2 \\ \mathbf{K}_i(\mathbf{x}_i + \mathbf{g}_i \cdot \mathbf{y}_{i-1}) & 2 < i \leqslant s. \end{cases} \tag{3.3}$$

3.2.2 Attention stage

The convolution stage only considers the information flow between adjacent feature-map groups. As such, it limits the model's ability to capture the dependencies between groups that have long distances in-between. In this regard, we design an attention stage to attend to a certain part when processing output feature maps. In particular, we propose two types of attention modules, namely channel-wise attention module and block-wise attention module, to harness multi-granular temporal patterns effectively.

3.2.2.1 Channel-wise attention

Channel-wise attention captures the relations between channels of the convolution stage's output, i.e., output feature maps, $\{\mathbf{y}_i\}_{i=1}^{s}$, where s is the number of feature-map groups in the convolution stage.

Suppose every $\mathbf{y}_i$ contains the same number of channels, say J channels—this is reasonable as we divide the original feature map $\mathbf{X}$ evenly along the channel.

Let $\mathbf{h}_{i,j}$ be the feature map for the jth channel of $\mathbf{y}_i$. We use three fully-connected layers to learn the query, key, and value of $\mathbf{h}_{i,j}$ (denoted by $\mathbf{q}_{i,j}$, $\mathbf{k}_{i,j}$, and $\mathbf{v}_{i,j}$, respectively). Similarly, we denote by $\mathbf{q}_{m,n}$, $\mathbf{k}_{m,n}$, and $\mathbf{v}_{m,n}$ the query, key, and value of $\mathbf{h}_{m,n}$, and the feature map for the nth channel of $\mathbf{y}_m$. Given two different feature maps, $\mathbf{h}_{i,j}$ and $\mathbf{h}_{m,n}$, we calculate the channel-wise attention as follows:

$$\mathbf{attention}\left(\mathbf{q}_{i,j},\mathbf{k}_{m,n}\right)=\frac{\mathbf{q}_{i,j}\mathbf{k}_{m,n}^{T}}{\sqrt{J}} \tag{3.4}$$

Once computed, we can update the feature map of every channel according to its relations with all the other feature maps. As feature maps contain temporal information within various ranges, channel-wise attention can capture temporal dependencies at multiple levels of granularity, realizing a **feature-map-wise attention**. Based on the above, the updated feature map $\tilde{\mathbf{h}}_{i,j}$ can be calculated as follows:

$$\tilde{\mathbf{h}}_{i,j}=\sum_{s}\sum_{J}\text{Softmax}\left(\frac{\mathbf{attention}\left(\mathbf{q}_{i,j},\mathbf{k}_{m,n}\right)}{\sum_s\sum_J\mathbf{attention}\left(\mathbf{q}_{i,j},\mathbf{k}_{m,n}\right)}\right)\mathbf{v}_{m,n} \tag{3.5}$$

Given s output feature maps each having J channels with k dimensions, the total number of feature maps for channel-wise attention is $s\times J$, resulting in the computational complexity of $\mathcal{O}\left((s\times J)^2k\right)$.

3.2.2.2 Block-wise attention

Block-wise attention regards each $\mathbf{y}_i$ as an individual block that contains temporal information at a certain granularity. Instead of calculating attention values along the channel, block-wise attention directly feeds $\mathbf{y}_i$ to the fully-connected layers to calculate the corresponding query, key, and value. Block-wise attention has advantages in mitigating *overfitting* as it considers sparse relations when computing the attention.

Suppose $\mathbf{y}_i$ and $\mathbf{y}_m$ are two output feature maps. We denote by $\mathbf{q}_i$, $\mathbf{k}_i$ and $\mathbf{v}_i$ the query, key and value of $\mathbf{y}_i$; similarly, we denote by $\mathbf{q}_m$, $\mathbf{k}_m$ and $\mathbf{v}_m$ the query, key and value of $\mathbf{y}_m$. Then, we calculate the block-wise attention as follows:

$$\mathbf{attention}(\mathbf{q}_i, \mathbf{k}_m) = \frac{\mathbf{q}_i \mathbf{k}_m^T}{\sqrt{s}} \tag{3.6}$$

Once computed, we can update the feature maps of every block according to their relations with all the other feature maps. And the updated feature map for each block, $\tilde{\mathbf{y}}_i$, can be calculated as follows:

$$\tilde{\mathbf{y}}_i = \sum_s \text{Softmax}\left(\frac{\mathbf{attention}\left(\mathbf{q}_i, \mathbf{k}_j\right)}{\sum_s \mathbf{attention}\left(\mathbf{q}_i, \mathbf{k}_j\right)}\right)\mathbf{v}_j \tag{3.7}$$

Given s feature maps, each having J channels with k dimensions, the computational complexity of block-wise attention is $\mathcal{O}\left(s^2 Jk\right)$.

3.3 Experiments

This section reports our extensive experiments to evaluate our proposed approach, including comparisons against baselines, ablation studies, and parameter studies on several public time-series datasets. We demonstrate that our approach can be used as a plugin to improve the performance of state-of-the-art methods and provide practical advice on how to adapt our approach to a specific problem.

3.3.1 Datasets

We conducted experiments on 14 public multivariate time series datasets (summarized in Table 3.1). These datasets cover various tasks from different application domains, such as activity recognition, EEG classification, and weather forecasting. They contain time series of various lengths with different numbers of variables. We have carefully selected these datasets to reflect applications in various domains and ensure that they

Table 3.1: A list of our experimental datasets.

Dataset	Task	#Classes	#Variables	Length	Train-test ratio	SOTA
Action 3D [83]	Action Recognition	20	570	100	48:52	MALSTM-FCN [71]
Ozone[1]	Weather Forecasting	2	72	291	50:50	MLSTM-FCN[71]
AREM[1]	Activity Recognition	7	7	480	50:50	MALSTM-FCN [71]
LP5[1]	Failure Detection	5	6	15	39:61	MUSE [99]
EEG[1]	EEG Classification	2	13	117	50:50	MLSTM-FCN [71]
Gesture Phase[1]	Gesture Recognition	5	18	214	50:50	MLSTM-FCN [71]
ECG[2]	ECG Classification	2	2	147	50:50	MUSE [99]
FingerMovements[3] [29]	Movement Classification	2	28	50	76:24	InceptionTime [43]
DuckDuckGeese[3] [29]	Audio Classification	5	1345	270	50:50	InceptionTime [43]
HeartBeat[3] [29]	Audio Classification	2	61	405	49:51	Canonical Interval Forest [91]
LSST[3] [29]	Signal Classification	14	36	6	50:50	MUSE [99]
MotorImagery[3] [29]	EEG Classification	2	3000	64	74:26	Time Series Forest [32]
SelfRegulationSCP2[3] [29]	EEG Classification	2	1152	7	53:47	DTW [92]
StandWalkJump[3] [29]	Activity Recognition	3	2500	4	45:55	ROCKET [30]

are diverse enough in length and variable number of time series to reflect different difficulty levels in real-world multivariate time-series classification problems.

3.3.2 Baseline Methods

We selected several competitive baselines and state-of-the-art (SOTA) methods to compare with our approach.

- **Res2Net** [44]: this is a CNN backbone that uses group convolution and hierarchical residual-like connections between convolutional filter groups to achieve multi-scale receptive fields.
- **GRes2Net** [139]: this work incorporates gates in Res2Net, where the gates' values are calculated based on a different method from ours—it additionally takes into account the original feature map before it is divided into groups when calculating gates' values.
- **Res2Net+SE**: this work combines Res2Net with a Squeeze-and-Excitation Block (SE) [60] to leverage the effectiveness of attention modules.
- **GRes2Net+SE**: this work combines GRes2Net with SE to leverage the effectiveness of attention modules.

We briefly introduce the SOTA methods for the experimental datasets below. A full list of SOTA methods is given in Table 3.1.

[1]https://archive.ics.uci.edu/ml/index.php
[2]http://www.cs.cmu.edu/~bobski
[3]https://www.cs.ucr.edu/~eamonn/time_series_data_2018

- **MLSTM-FCN [71]**: a multivariate LSTM fully convolutional network that concatenates the outputs of two parallel blocks: a fully convolutional block (embedded with SEs) and an LSTM block. It is a variant of LSTM-FCN.
- **MALSTM-FCN [71]**: a multivariate attention LSTM fully convolutional network, which resembles MLSTM-FCN but replaces LSTM cells with attention LSTM cells.
- **MUSE [99]**: a model that extracts and filters multivariate features by encoding context information into each feature.
- **InceptionTime [43]**: a CNN-based model transferred from computer vision to time series classification, which stacks multiple parallel convolutional filters for temporal feature extraction.
- **Time Series Forest [32]**: an ensemble tree-based method that employs a combination of entropy gain and a distance measure to evaluate the differences between time-series sequences.
- **Canonical Interval Forest [91]**: a model that refines *Time Series Forest* by upgrading the interval-based component.
- **Dynamic Time Warping (DTW)[92]**: a traditional distance-based machine learning method for time series analysis.
- **Random Convolutional Kernel Transform (ROCKET) [30]**: a CNN-based model that uses random convolutional kernels to extract multi-granular temporal features.

3.3.3 Model Configuration and Evaluation Metric

We followed the methods as illustrated in the SOTA methods to preprocess the datasets. In particular, we normalized each dataset to zero mean and unit standard deviation. We also applied zero paddings to cope with sequences with various lengths in the same training set. The experimental results of each method were obtained under the optimal or suggested settings as provided in the original paper.

To ensure a fair comparison, we set all the models based on Res2Net, GRes2Net, and our approach contained the same number of feature-map groups and used identical filters for each group.

We used our model as the backbone for feature extraction and trained our model for 500 training epochs using Adam [75] optimizer. The learning rate was set to 0.001 and

Table 3.2: Experiment configuration settings

Dataset	Number of Layers	Number of Groups	Dropout Rate
FingerMovements	4	4	0.2
DuckDuckGeese	4	64	0.3
HeartBeat	2	64	0.2
LSST	2	8	0.25
MotorImagery	6	64	0.2
SelfRegulationSCP2	4	8	0.5
StandWalkJump	4	64	0.2
Action 3D	4	8	0.3
Ozone	4	8	0.25
AREM	6	64	0.25
LP5	4	2	0.4
EEG	4	4	0.5
Gesture Phase	5	8	0.4
ECG	4	8	0.5

adjusted to 1/10 of itself after every 100 epochs. The dropout rate was set to 0.4 to avoid possible overfitting. We repeated the training and test processes five times and took the average of multiple runs as the final results; this mitigates the impact of randomized parameter initialization. The details including the number of layers, the number of convolutional groups, and the dropout rate settings can be found in Table 3.2.

We used *accuracy*, which is currently used by all the SOTA methods on the experimental datasets, as the metric for evaluating the methods. Accuracy measures the ratio of correctly predicted instances to the total instances, which can be described as:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}} \tag{3.8}$$

However, accuracy is not comprehensive enough to measure the performance of the classifier. Although the vast majority of the related work uses accuracy as the only evaluation metric, we additionally use *precision*, *recall*, and *F1-Score* in our parameter and ablation studies to gain further insights into how our model performs. Precision measures the ratio of correctly predicted positive observations to the total predicted positive observations, which can be described as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.9}$$

Recall measures the ratio of correctly predicted positive observations to all actual positives, which can be described as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.10}$$

Table 3.3: Accuracy of different models on 14 benchmark datasets. AGRes2Net+CA and AGRes2Net+BA represent our Attentional Gated Res2Net model incorporated with channel-wise attention and block-wise attention, respectively. The improvement is the comparison between SOTA and the proposed model.

Dataset	Res2Net	Res2Net+SE	GRes2Net	GRes2Net+SE	SOTA	AGRes2Net+BA	AGRes2Net+CA	Improvement (%)
FingerMovements	0.5240	0.5280	0.5340	0.5480	0.5613	**0.6240**	0.5820	11.17
DuckDuckGeese	0.6360	0.6680	0.6560	0.6800	0.6347	0.6880	**0.7080**	11.55
HeartBeat	0.6463	0.7415	0.7512	0.7561	0.7652	0.7853	**0.8663**	13.21
LSST	0.5268	0.5447	0.5341	0.5799	0.6362	0.5843	**0.6671**	4.86
MotorImagery	0.5220	0.5340	0.5380	0.5740	0.5380	0.6240	**0.6280**	16.73
SelfRegulationSCP2	0.5367	0.5522	0.5444	0.5555	0.5369	0.5711	**0.6210**	15.66
StandWalkJump	0.3333	0.4000	0.3333	0.3333	0.4556	**0.4667**	0.3333	2.44
Action 3D	0.7457	0.7182	0.7301	0.8037	0.7542	0.8350	**0.8617**	14.25
Ozone	0.7989	0.8264	0.8034	0.8390	0.8150	0.8494	**0.8620**	5.77
AREM	0.7692	0.8462	0.8205	0.8717	0.8462	**0.9231**	0.8974	9.09
LP5	0.5642	0.5799	0.684	0.6328	0.7100	**0.7396**	0.7326	4.17
EEG	0.5781	0.5469	0.6094	0.6406	0.6563	**0.6719**	**0.6719**	2.38
Gesture Phase	0.5859	0.5898	0.6601	0.6641	0.5353	0.6445	**0.6953**	29.89
ECG	0.7200	0.8000	0.8400	0.8300	0.9300	0.8500	**0.9400**	1.08

The F1-Score is the harmonic mean of precision and recall, providing a balance between the two metrics, which can be described as:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.11}$$

3.3.4 Comparison of Different Methods

Table 3.3 shows a performance comparison of all the methods on the experimental datasets. Our proposed model, using either channel-wise or block-wise attention, consistently outperformed all the other compared methods on all 14 datasets, demonstrating our model's superiority in solving MTSC in diverse contexts regardless of the lengths of time-series sequences.

Channel-wise attention favors longer time-series sequences, as it beats block-wise attention on all the top-8 datasets with the longest sequences. The results conform to our intuition that channel-wise attention may have an edge on capturing multi-granular temporal information.

Block-wise attention tends to excel on datasets that contain fewer variables. Among the top-4 datasets with the least variables, it beats channel-wise attention on 3 of them (AREM, LP5, and EEG); this is also consistent with our intuition that block-wise attention may have advantages in preventing overfitting thanks to the sparse relations considered in its attention calculation.

An exception occurs on the ECG dataset, which has as few as two variables; this reason lies in that this dataset contains abundant sequences that allow for channel-wise attention to fully exploit the training data without causing overfitting.

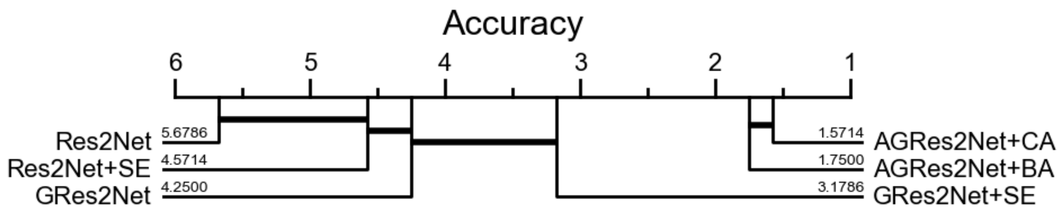


Figure 3.2: Critical difference diagram of the arithmetic means of the ranks on all datasets.

Figure 3.2 shows the result of the Wilcoxon signed-rank test on the baseline methods' performance. It shows that, overall, our model achieves similar classification performance when using channel-wise attention and block-wise attention. Either way, our model performs significantly better than the baselines. This result demonstrates the effectiveness of harnessing inter-dependencies between variables and multi-granular feature maps (as our model does use gates, attention, and group convolution) in improving classification performance on sequences of various lengths.

3.3.5 Impact of Depth and Width of Model

In this experiment, we study how the depth and width of our model impact the classification performance. Generally, a deeper and wider model has a stronger capability to capture complex relations from data. Our model becomes more complex as we increase its depth (by stacking more layers), width (by expanding the number of feature-map groups), or both.

We trained our model under different width and depth settings and applied different types of attention in the experiment. Considering the many experimental datasets, we only show the results on two representative datasets, **Action 3D** and **Heartbeat**. The former has medium-length sequences and a large number of variables; in contrast, the latter has long sequences but a medium number of variables, making them ideal for exemplifying the experimental results. In particular, we show the results of our model after applying channel-wise attention and block-wise attention on **Heartbeat** and **Action 3D** datasets, respectively.

Our results (Table 3.4) show that wider models beat deeper models in both the training and test phases. While stacking multiple layers leads to large receptive fields that can capture dependencies in a larger range, a wider model can achieve receptive fields with multiple sizes and fuse the feature maps from different convolution filters

Table 3.4: Training and test results under varying widths and depths.

Dataset	Configuration		Train				Test			
	Width	Depth	Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
HeartBeat	8	8	0.9482	0.8933	0.9108	0.9019	0.7622	0.6848	0.6953	0.6901
	16	4	0.9712	0.9221	0.9423	0.9321	0.8443	0.6859	0.6879	0.6869
	32	2	**0.9742**	**0.9298**	**0.9491**	**0.9394**	**0.8570**	**0.8602**	**0.8551**	**0.8576**
Action 3D	8	8	0.8525	0.8225	0.8213	0.8219	0.7490	0.7767	0.7458	0.7609
	16	4	0.9003	0.8864	0.8848	0.8856	0.8434	0.8171	0.8091	0.8130
	32	2	**0.9068**	**0.8908**	**0.8908**	**0.8908**	**0.8515**	**0.8701**	**0.8462**	**0.8579**

to learn multi-granular temporal patterns. In comparison, a wider model leverages the temporal features of time-series sequences more effectively, making it generally a better choice. Several studies [132, 154] in the computer vision field draw similar conclusions.

3.3.6 Impact of Group Number

In this experiment, we further explore the impact of the hyperparameter s, which determines the number of feature-map groups (as well as the number of filter groups) in our model. Intuitively, a larger s gives a wider model that can fuse more temporal features extracted by convolutional filters with multiple sizes of receptive fields, thus facilitating the capturing long-range dependencies.

We kept all other settings (e.g., number of layers, epochs, learning rate, dropout rate) unchanged while varying the value of s to explore its influence on classification results. Similar to the precedent experiment, we show the experimental results on four datasets that have significantly different lengths of sequences (namely **LP5**, **AREM**, **Ozone**, and **Action 3D**) to avoid information overload. We used block-wise attention on the first two datasets and channel-wise attention on the last two.

Our results (Table 3.5) show our model consistently achieved better performance during training as s increased. We can easily tune our model towards capturing a broader range of temporal information by allowing for more groups with a greater s. However, greater values of s bring the risk of overfitting, demonstrated by decreased performance in the test phase, e.g., in the case of the Qzone and Action 3D datasets. The results suggest the necessity of tuning this hyperparameter s given a specific dataset to gain the best performance.

Beyond the above results, we may consider our model as *recurrent* because each group's output feature map is sent to the subsequent group. Following this idea, we may regard group number s as the number of steps that the model takes during its recurrent computation. While traditional convolutional neural networks obtain larger receptive

Table 3.5: Training and test results under different s. We set greater s values for the AREM dataset as it has much longer sequences than the others do. We set 6 layers for Ozone, 6 layers for AREM, 4 layers for Action 3D, and 4 layers for LP5.

Dataset	s	Train				Test			
		Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
Ozone	2	0.9425	0.9434	0.9437	0.9436	0.8436	0.8299	0.8281	0.8289
	4	0.9529	0.9501	0.9428	0.9464	0.8563	0.8609	0.8598	0.8603
	8	0.9635	0.9699	0.9640	0.9669	**0.8570**	**0.8602**	**0.8551**	**0.8576**
	16	0.9792	0.9771	0.9774	0.9772	0.8257	0.8346	0.8243	0.8294
	32	**0.9844**	**0.9827**	**0.9830**	**0.9829**	0.8257	0.8302	0.8372	0.8337
AREM	4	0.7907	0.7798	0.7534	0.7664	0.7949	0.7007	00.7171	0.7088
	8	0.8139	0.8452	0.8250	0.8350	0.7435	0.6871	0.8268	0.7505
	16	0.8605	0.8870	0.8397	0.8627	0.8205	0.7756	0.8648	0.8178
	32	0.8837	0.9048	0.8939	0.0.8993	0.8718	0.8163	0.9056	0.8586
	64	**0.9767**	**0.9821**	**0.9841**	**0.9831**	**0.8692**	**0.9619**	**0.8452**	**0.8998**
Action 3D	2	0.8138	0.7968	0.8426	0.8191	0.7088	0.7212	0.7352	0.7281
	4	0.8219	0.8149	0.8595	0.8366	0.7207	0.7255	0.7391	0.7322
	8	0.8232	0.8109	0.8592	0.8344	**0.8617**	**0.8764**	**0.8711**	**0.8737**
	16	0.8263	0.8201	0.8638	0.8414	0.8318	0.8359	0.8131	0.8243
	32	**0.8350**	**0.8296**	**0.8681**	**0.8484**	0.8252	0.8329	0.7991	0.8156
LP5	2	0.7813	0.7964	0.8125	0.8043	**0.7396**	**0.7818**	**0.7214**	**0.7504**
	4	0.8125	0.8511	0.8398	0.8313	0.7309	0.7014	0.7568	0.7158
	8	0.8594	0.8750	0.8593	0.8671	0.7014	0.7052	0.7665	0.7033
	16	0.8906	0.9142	0.8809	0.9022	0.6771	0.7022	0.7237	0.6894
	32	**0.9531**	**0.9714**	**0.9418**	**0.9622**	0.6215	0.6187	0.6454	0.6201

fields by stacking multiple layers or employing dilation convolution layers, they are not as flexible or effective as our model in capturing multi-granular temporal information.

3.3.7 Impact of Attention Modules

The superiority of our attention modules over SE is indicated by our model outperforming those baselines that incorporate SE [60] (see Table 3.3). Specifically, the SE module uses *global average pooling*, which generates a scalar to represent the feature map of each channel. In comparison, our attention mechanisms (channel-wise and block-wise attention) avoid using global average pooling, thus preventing the information loss caused by the pooling operation.

Table 3.6 further shows our model's performance when using the two attention modules during the training and test phases. We choose to show the results on three datasets, which cover a large range of variable numbers (7 for AREM, 72 for Ozone, and 570 for Action 3D). The results (Table 3.6) are consistent with our findings in Section 3.3.4 that channel-wise attention generally beats block-wise attention except for small datasets with very few variables.

Table 3.6: Training and test results of our model with different attention modules.

Dataset	Attention	Train				Test			
		Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
Ozone	Block-wise	0.7088	0.7212	0.7352	0.7281	0.6793	0.6644	0.6685	0.6664
	Channel-wise	**0.8232**	**0.8109**	**0.8592**	**0.8344**	**0.8604**	**0.8669**	**0.8437**	**0.8551**
AREM	Block-wise	0.8837	0.9048	0.8939	0.8993	**0.8718**	**0.8163**	**0.9056**	**0.8586**
	Channel-wise	**0.9767**	**0.9821**	**0.9841**	**0.9831**	0.8205	0.7483	0.8772	0.8076
Action 3D	Block-wise	0.9091	0.8908	0.9091	0.8996	0.8181	0.8306	0.8150	0.8227
	Channel-wise	**0.9635**	**0.9699**	**0.9640**	**0.9669**	**0.8570**	**0.8602**	**0.8551**	**0.8576**

Table 3.7: Ablation test for our model.

Dataset	Model	Accuracy	Recall	Precision	F1-Score
EEG	Res2Net	0.5781	0.5713	0.5882	0.5796
	Res2Net + Gates	0.5938	0.5943	0.5943	0.5943
	Res2Net + channel-wise attention	0.6094	0.6105	0.6417	0.6257
	Res2Net + Gates + channel-wise attention	**0.6719**	**0.6750**	**0.6833**	**0.6791**
AREM	Res2Net	0.7692	0.7469	0.7639	0.7530
	Res2Net + Gates	0.8205	0.7551	0.8762	0.8112
	Res2Net + block-wise attention	0.8718	0.8163	0.8929	0.8529
	Res2Net + Gates + block-wise attention	**0.9231**	**0.8762**	**0.9571**	**0.9149**

As for this experiment, both Ozone and Action 3D datasets contain many variables (72 and 570) and sufficient sequences during training for channel-wise attention to perform well. In contrast, AREM contains only 43 sequences that cover as many as seven classes. The number of sequences is extremely limited for each class, making channel-wise attention easily lead to overfitting.

3.3.8 Ablation Study

We conducted ablation studies to explore the effectiveness of gates and our attention modules. The model without gates and attention module is the same as vanilla Res2Net. We separately incorporate gates, attention, and both attention and gates in Res2Net and compare the results.

Again, we only present the results on **EEG** and **AREM** datasets to avoid information overload. For each dataset, we tested the attention mechanism that led to inferior performance to the other, i.e., channel-wise attention on the EEG dataset and block-wise attention on the AREM dataset, to make the comparisons more evident.

Our results (Table 3.7) show that attention modules contribute slights more than gates in improving the performance of Res2Net, but every component contributes significantly to the improved performance.

Table 3.8: Time consumption comparison with attention modules and without attention modules on DuckDuckGeese, CA means channel-wise attention and BA means block-wise attention. The data in the brackets is the standard deviation.

	Without Attention	With CA	With BA
Training time consumption (s)	1.3991 (0.1627)	3.0911 (0.2772)	2.8689 (0.2391)
Test time consumption (s)	0.7675 (0.0977)	1.1226 (0.0737)	0.980 (0.0860)

Table 3.9: Time consumption comparison with attention modules and without attention modules on MotorImagery, CA means channel-wise attention and BA means block-wise attention. The data in the brackets is the standard deviation.

	Without Attention	With CA	With BA
Training time consumption (s)	37.7661 (1.7530)	186.5941 (9.0698)	75.9330 (2.808)
Test time consumption (s)	7.1772 (0.3879)	22.1444 (3.5978)	8.0369 (0.4131)

3.3.9 Time Consumption of Attention Modules

We conducted experiments to analyze the extra time consumption of the attention modules. We select two datasets, **MotorImagery** and **DuckDuckGeese**, because their length and variable number are significantly large. We trained the models on i7-8700K CPU instead of GPU because GPUs are too powerful that can alleviate the impact. We stacked 4 layers and used 64 groups of convolutional filters at each layer. We trained the model with channel-wise attention, with block-wise attention, and without attention module 300 epochs separately and recorded the training time and test time per epoch. We calculate and give the average time consumption and the standard deviation. The results are shown in Table 3.8 and Table 3.9.

According to the results, we can see that time consumption significantly increases when using the attention module. Among the two attention modules, channel-wise attention is more computationally expensive. Compared with the model without any attention module, the time consumption of channel-wise attention for training is about 2.2 times on DuckDuckGeese and is 4.9 times on MotorImagery. While the time consumption of block-wise attention for training is 2.1 times on DuckDuckGeese and is 2 times on MotorImagery. Although attention modules improve the performance (shown in Section 3.3.8), they also make the model less efficient, which brings challenges for employing the model on devices with limited computing resources.

Table 3.10: Performance Comparison between the data with PCA and without PCA on SelfRegulationSCP2. The data in the brackets is the standard deviation.

	Training					Test				
	Accuracy	Precision	Recall	F1-Score	Time per epoch	Accuracy	Precision	Recall	F1-Score	Time per epoch (s)
Without PCA	**0.9386**	**0.9300**	**0.9297**	**0.9298**	18.3621 (1.0645)	**0.6210**	**0.6167**	**0.6132**	**0.6149**	6.0603 (0.2679)
With PCA	0.8958	0.8799	0.8788	0.8793	**0.6664 (0.8050)**	0.5667	0.5611	0.5556	0.5583	**0.1981 (0.0235)**

3.3.10 Impact of Feature Dimension Reduction

As discussed in Section 3.3.9, we find that our model is less efficient when the time series contains too many variables. So, we conducted experiments to explore the impact of combining feature dimension reduction algorithms with the AGRes2Net. We select **SelfRegulationSCP2** dataset as it contains 1152 variables. We used Principal Component Analysis (PCA) to reduce the number of variables from 1152 to 28. we stacked 4 layers, and each layer has 8 groups of convoltuional filters. We use the same dropout rate and experiment settings that are described in Section 3.3.3. We trained the model on i7-8700K CPU. We recorded the performance, including accuracy, precision, recall, F1-Score, and time consumption of both the training phase and test phase. The results are given in Table 3.10.

According to the results, all the performances go poorer, but the time consumption is significantly reduced. Specifically, the accuracy after using PCA decreases 9.59%, but the test speed of the model is about 33 times faster. So, dimension reduction algorithms (such as PCA) are practicable for dropping some features if we want to make the model more efficient in facing the time series that contain too many variables.

3.3.11 Effectiveness of Our Model as a Plugin

We use MLSTM-FCN, the SOTA architecture on most datasets (as shown in Table 3.1), to demonstrate the effectiveness of our model as a plugin. The original MLSTM-FCN follows a CNN-LSTM parallel architecture. The input goes through multiple LSTMs and CNNs, and the outputs are concatenated and go through a fully connected layer for information fusion. We conducted this experiment by replacing the original convolutional modules of MLSTM-FCN with our model while preserving the architecture and all the other parts in MLSTM-FCN.

We show the comparison results on two datasets, **AREM** and **Gesture Phase**, to demonstrate the impact of our model on the overall performance of MLSTM-FCN. Specifically, we adopted block-wise attention on the AREM dataset and channel-wise attention on the Gesture Phase dataset without particular reasons. We omit to show the

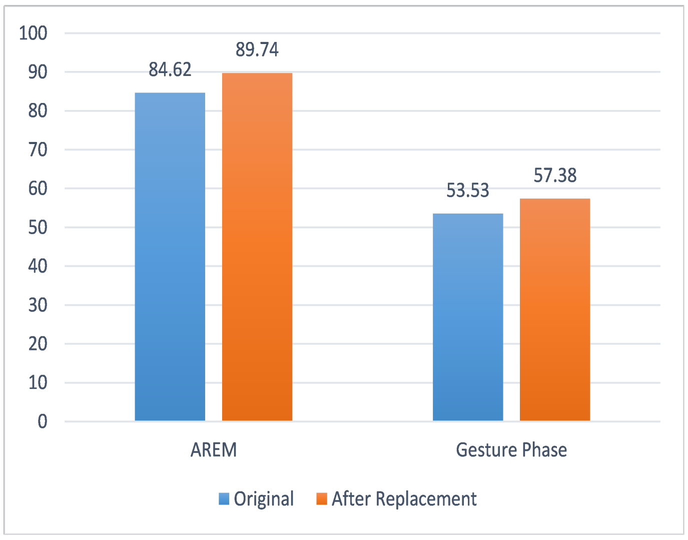


Figure 3.3: Accuracy comparison between the vanilla MLSTM-FCN (blue bar) and the MLSTM-FCN where our model replaces the convolutional modules (orange bar). Block-wise attention and channel-wise attention are applied to the AREM dataset and the Gesture Phase dataset, respectively.

results on other datasets as they draw similar conclusions.

The results (Figure 3.3) show a significant improvement in the classification accuracy of MLSTM-FCN on both datasets after the replacement, demonstrating the positive effect of our model on the performance of existing multivariate time series classification models when used as a plugin.

3.3.12 Exploring the Threshold for Choosing Channel-wise Attention and Block-wise Attention

As discussed in the previous section, channel-wise attention performs better and vice versa. This section further explores whether a standard threshold exists for choosing the proper attention module. We select two datasets, **LSST** and **HeartBeat**, for experiments because they contain many variables and channel-wise attention performs better than block-wise attention, and we can use dimension reduction methods to tune the variable numbers to find when the block-wise attention performs better. We use PCA to gradually control the variable numbers. The results can be seen in Table 3.11 and Table 3.12.

According to the results, we can see the thresholds of the two datasets are different (3 for LSST and 2 for HeartBeat). Besides, when we reduce the variable number to 3 on LSST, the performance significantly decreases, making the results less convincing. According to the results, we can see the thresholds of the two datasets are different (3

Table 3.11: Performance comparison based on the different variable numbers on LSST

	Block-wise Attention				Channel-wise Attention			
Variable Number	Accuracy	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score
3	0.2173	0.1328	0.1221	0.1272	0.1473	0.1316	0.1154	0.1229
4	0.5799	0.5795	0.5722	0.5758	0.6553	0.6375	0.6358	0.6367

Table 3.12: Performance comparison based on the different variable numbers on Heart-Beat

	Block-wise Attention				Channel-wise Attention			
Variable Number	Accuracy	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score
2	0.7456	0.6207	0.6321	0.6264	0.6451	0.5560	0.5496	0.5528
3	0.6369	0.5970	0.5975	0.5972	0.7130	0.6072	0.6156	0.6114

for LSST and 2 for HeartBeat). Besides, when reducing the variable number to 3 on LSST, the performance of both attention modules is significantly decreased, making the results less convincing. Besides, from the results given in Table 3.3.4, we can see on the FingerMovements dataset, the block-wise attention performs better, while on the ECG dataset, the channel-wise attention outperforms block-wise attention. However, Finger-Movements contains 28 variables, while ECG contains only 2 variables. To summarize, the threshold is case-by-case, and the standard threshold does not exist. Although we can follow a rule that using channel-wise attention is preferable in facing a dataset that has lots of variables (such as SelfRegulationSCP2, Action 3D, DuckDuckGeese, etc.), we still need to do empirical studies on each dataset to choose the proper attention module.

3.3.13 Practical Advice

We offer several suggestions on applying our model to broader scenarios based on the above experimental results and our analysis:

- *Avoid very deep models*: based on the results given in Section 3.3.5, a wider model is generally more capable than a deeper model of addressing a general multivariate time series classification. We should prioritize constructing wider models rather than stacking more layers when faced with a new problem.

- *Focus on tuning the hyperparameter s*: setting a larger s increases the number of convolutional-filter groups, leading to multiple receptive fields that capture temporal patterns in various ranges. The experimental results given in Section 3.3.6 suggest that tuning the hyperparameter s is especially important for long time-series sequences to achieve the best possible performance. It is generally worthwhile to tune s ahead of investigating the optimal settings of other parameters.

- *Choose attention module based on variable number*: the number of variables is, by far, the most useful single criterion for deciding which attention module to choose for our model based on our experiments delivered in Section 3.3.7. As discussed, block-wise attention is preferred for sequences with a small number of variables, and channel-wise attention is more suitable for sequences with massive variables. More criteria include the number of sequences available for training, the number of classes, and the length of sequences, which must be figured out case by case.

To conclude, the application and implementation settings depend on the amount of training data and the variable numbers. Besides, the optimal hyperparameter is scenario-dependent and should be carefully considered to realize better performance.

3.4 Conclusion

In this paper, we propose a novel deep learning architecture called *Attentional Gated Res2Net* for accurate multivariate time series classification. Our model comprehensively incorporates gates and two types of attention modules to capture multi-granular temporal information. We evaluate the model on diverse datasets that contain sequences of various lengths with a wide range of variable numbers. Our experiments show the model outperforms several baselines and state-of-the-art methods by a large margin. We thoroughly investigate the effect of different components and settings on the model's performance and provide hands-on advice on applying our model to a new problem. Our test on plugging the model into a state-of-the-art architecture, MLSTM-FCN, demonstrates the potential for using our model as a plugin to improve the performance of existing models.

CHAPTER 4

From Time Series to Multi-Modality: Classifying Multivariate Time Series via Both 1D and 2D Representations

4.1 Introduction

A typical multivariate time series contains a sequence of data points at regular time intervals, where values of multiple variables or measurements from multiple sensors exist at the same time points. While traditional methods for multivariate time series classification have been based on statistical or machine learning methods, deep learning-based methods, represented by Long Short-Term Memory (LSTM) [56], Inception-time [43], and Time Series Transformer (TST) [151] have gained prevalence recently thanks to their outstanding capability to extract effective features and learn representation in complex scenarios. Until now, all the existing approaches have been focusing on the temporal information of multivariate time series data while disregarding the underlying frequency information, which is proven invaluable in many domains like signal processing [95]. Intuitively, real-world multivariate time series data often exhibit periodicity that is challenging to detect and model from a purely temporal perspective. This highlights the necessity of incorporating frequency information into the classifier to model and classify multivariate time series data accurately. All the above inspires us to develop a novel approach that can leverage temporal and frequency information comprehensively for more accurate multivariate time series classification.

Existing methods that extract frequency information from time series data generally aim for time series forecasting, represented by ETSformer [45] and COST [125]. These methods are commonly based on Fourier Transform [12], which decomposes time series into a set of sine functions representing different frequencies, with the amplitude of each sine function indicating the intensities of the frequency components. Fourier Transform, however, can only observe time series' global frequency components without their temporal positions, resulting in insufficient frequency information that limits the accuracy of multivariate time series classification. Therefore, it calls for new approaches that can incorporate more comprehensive frequency information to improve classification performance.

In light of the above, we aim to classify multivariate time series sequences by leveraging both temporal and frequency information. Specifically, we adopt the Short-Time Fourier Transform (STFT) [47] to address the limitations of the Fourier Transform. STFT divides time series into overlapping segments, applies a Fourier transform to each segment, and finally concatenates the resulting 2D frequency domain representations to provide more comprehensive information that covers both the frequency components and their temporal positions. In particular, we use three different window sizes to generate spectrograms for each variable; these spectrograms carry multi-resolution frequency information that reflects multi-scale temporal patterns of time series which is crucial for modeling time series [23]. Through the above transformation, we create a new data modality and transform the time series classification task into a multi-modality classification task. This further allows us to bring computer vision backbones into time series classification, which have shown effectiveness in exploiting 2D representations [127]. We further construct a dual-stream architecture based on ResNet [55], a widely used computer vision method, to leverage the power of both 2D representations (with frequency information) and 1D representations (with temporal information) of time series. The combination of 2D and 1D representations enables us to classify time series effectively, demonstrated by our proposed method consistently outperforming state-of-the-art baselines on 30 public multivariate time series datasets.

4.2 Methodology

The proposed method is based on a dual-stream architecture consisting of a spectrogram stream and a time series stream, as illustrated in Figure 4.1. We first implement the STFT using three different window sizes to generate a set of 2D spectrograms with

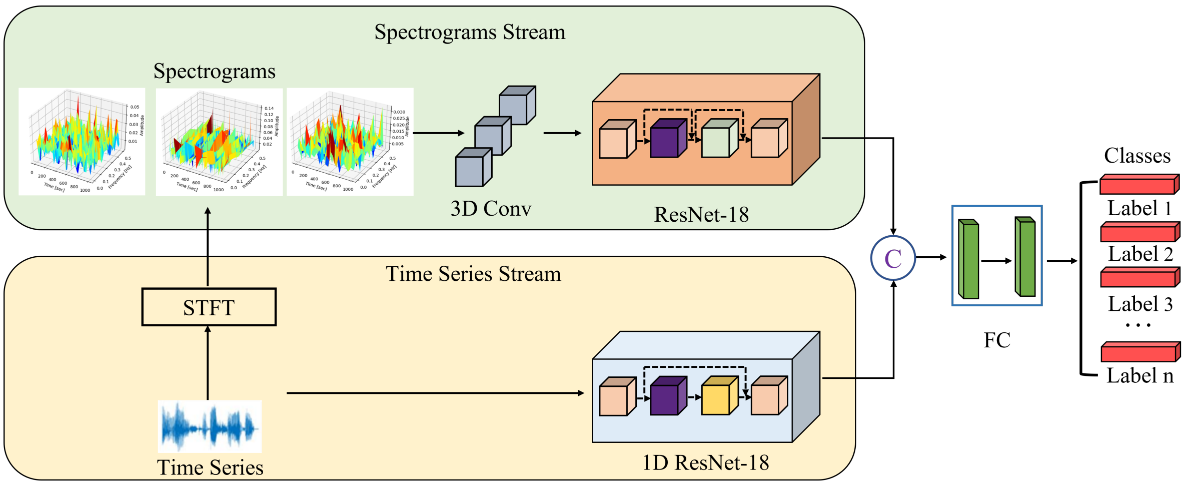


Figure 4.1: The architecture of the proposed method. We use the time series with one variable to illustrate our method for simplicity. We employ Short-Time Fourier Transform with three different window sizes to generate a set of 2D spectrograms with multiple resolutions. We construct a dual-stream architecture based on ResNet to leverage both 1D representations and 2D representations. In the spectrogram stream, we use a 3D convolutional layer to fuse the spectrogram information from the resolution perspective and feed the output to ResNet-18. In the time series stream, we follow the architecture of ResNet-18 while replacing the 2D convolutional kernels using 1D convolutional kernels to adapt the shape of time series data. Finally, the output feature maps from two streams are concatenated (C in this Figure means concatenation) and fed into a fully-connected layer to map the output to the probability distribution of the classes.

varying temporal and frequency resolutions. Following this, a 3D convolutional layer is utilized to fuse the resolution-wise information of the spectrograms, while the output is fed into a ResNet-18 network to leverage 2D representations. Concurrently, the time series data is fed into a 1D ResNet-18 network that leverages 1D representations using 1D convolutional kernels. The output feature maps of both streams are concatenated, and a fully-connected layer with softmax function is applied to map the output to the probability distribution of the classes. We elaborate on each component of the proposed method in the following sections.

4.2.1 Short-Time Fourier Transform

Real-world time series data are typically sampled from continuous data streams at specific sampling rates. In signal processing, the Discrete Fourier Transform (DFT) is commonly used to extract frequency components from time series data, which can be

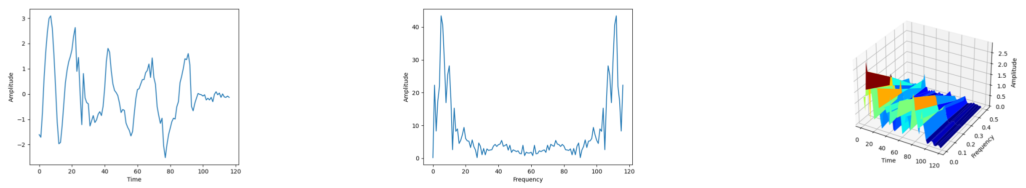

(a) Time Domain (b) Spectrogram generated by Fourier Transform (c) Spectrogram generated by STFT

Figure 4.2: The time domain of a time series sampled from the Handwriting dataset and the spectrograms generated by the Fourier Transform and Short-Time Fourier Transform (STFT). The spectrogram generated by the STFT provides more comprehensive frequency information, including both the frequency components and their temporal positions, in contrast to the spectrogram generated by the Fourier Transform

described as follows:

$$X(k) = \sum_{t=0}^{T-1} x(t)e^{-i2\pi kt/T} \tag{4.1}$$

where x_t is the time series sequence, and $t \in (0, T-1)$, T is the length of the time series. $X(k)$ is the frequency component obtained after DFT, while k is the index. However, the DFT lacks temporal position information of the frequency components, resulting in insufficient frequency information. To address this limitation, the Short-Time Fourier Transform (STFT) is performed, which involves using a sliding window to divide a time series sequence into short time intervals and performing the Fourier Transform on each interval to obtain the frequency components and their temporal positions. The STFT can be described as:

$$X(j,\omega) = \sum_{t=0}^{L-1} x(t)w(j-t)e^{-i\omega t} \tag{4.2}$$

where $x(t)$ represents the input time series in the time domain, $w(j-t)$ represents truncating the time series $x(t)$ with a window function in time to obtain the short time interval $x(t)w(j-t)$, L is the length of the window, j represents the center position of the current window, and ω represents the frequency of interest. In this case, STFT provides more sufficient frequency information including the frequency components and their temporal positions compared with the Fourier Transform. We applied the Fourier Transform and STFT to a sequence sampled from the **Handwriting** dataset to illustrate the differences between the spectrograms obtained through the Fourier Transform and STFT, and the results are shown in Figure 4.2. The STFT requires a balance between frequency and temporal resolutions, which presents a challenge in selecting an optimal window size. A larger window size provides more precise frequency information but results in poorer temporal resolution, while a smaller window size provides better temporal resolution but less precise frequency information. We follow a traditional signal processing approach [54] to address this issue, where the window size is chosen based on the time series's bandwidth. To select an appropriate window size, we calculate the maximum bandwidth among all variables, which can be described as:

$$\begin{gathered}\text{Bandwidth }_n = \lceil f^n_{\max} - f^n_{\min} \rceil \\ \text{Bandwidth } = \max(\text{ Bandwidth }_0, \text{ Bandwidth }_1, \ldots, \text{ Bandwidth }_N)\end{gathered} \tag{4.3}$$

where f^n_{max} and f^n_{min} are the maximum frequency and the minimum frequency present in the time series's nth variable, respectively, and N is the variable number. We then use three window sizes: the two, three, and four times the time series's frequency bandwidth, respectively, with an overlap of 50%, to generate three spectrograms with multi-level resolutions. For a time series sequence $x \in \mathbb{R}^{N \times T}$, where N is the variable number and T is the sequence length, the corresponding spectrogram generated by STFT is $s \in \mathbb{R}^{N \times 3 \times H \times W}$ where 3 means three different window sizes that we use, and H and W are the spectrogram's height and width. In this way, we extract the frequency components and their temporal positions from the time series and create a new data modality by converting the 1D time series sequence into a set of 2D representations, enabling us to borrow the powerful backbones from the computer vision field for leveraging 2D representation.

4.2.2 ResNet-18

We propose a dual-stream architecture based on ResNet [55] to leverage both the 2D representations in the spectrogram stream and the 1D representations in the time series stream for representation learning. ResNet is a popular deep neural network architecture that addresses the issue of vanishing gradients, which arises when the gradients become too small to effectively update the weights during backpropagation, particularly in very deep networks. This property has made it a competitive backbone for various computer vision tasks, motivating us to adopt it in our approach. In the spectrogram stream, the sets of 2D representations generated by the STFT are first fed into a 3D convolutional layer for resolution-wise information fusion. This layer down-samples the input spectrograms from the resolution perspective and generates a single 2D representation for each variable. The calculation process can be described as:

$$y = W * x + b \tag{4.4}$$

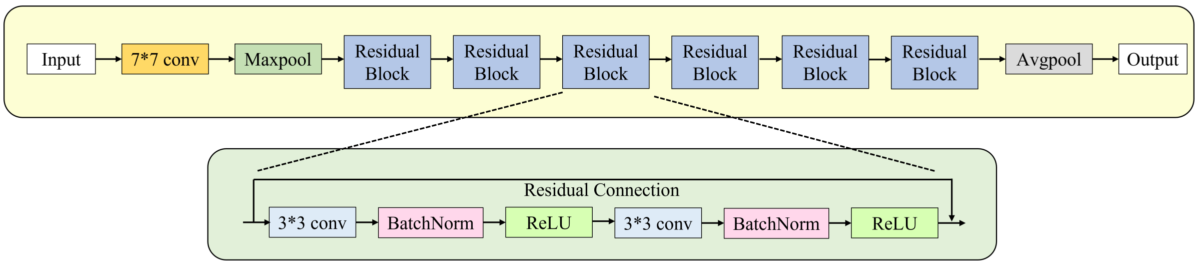


Figure 4.3: The architecture of ResNet-18.

where $x \in \mathbb{R}^{N\times 3\times H\times W}$ and $y \in \mathbb{R}^{N\times H\times W}$, and W and b are the convolutional kernel and bias term, respectively.

The resulting 2D representation and the original 1D time series are then fed into two separate neural networks, namely ResNet-18 and 1D ResNet-18, respectively. The architecture of ResNet-18, illustrated in Figure 4.3, comprises six residual blocks, each consisting of two convolutional layers with a kernel size 3×3. The output feature maps are fed into an average pooling layer for down-sampling from the spatial perspective, generating the latent vector of the input feature maps. In the 1D ResNet-18, we follow the same architecture as ResNet-18 but replace the 2D convolutional kernels with 1D convolutional kernels to accommodate the shape of the 1D representations. This enables us to process the time series data while retaining the advantages of ResNet-18's architecture. We then concatenate the output of the two streams and feed them into the fully-connected layer with a softmax function to map them to the probability distribution of the classes.

4.3 Experiments

4.3.1 Datasets

We evaluated our method using the UEA Time Series Classification Repository [29], which contains 30 public multivariate time series datasets. These datasets concern different domains and reflect diverse data characteristics in terms of sequence lengths and variable numbers, etc. All datasets had been preprocessed and split into training and test sets.

We further normalized them to zero mean and unit standard deviation and applied zero paddings to ensure that each dataset contains sequences of the same lengths. Table 4.1 shows the details of the datasets.

Table 4.1: Statistics of the 30 UEA datasets used in experimentation.

Dataset	Train Cases	Test Cases	Dimensions	Length	Classes
ArticularyWordRecognition	275	300	9	144	25
AtrialFibrillation	15	15	2	640	3
BasicMotions	40	40	6	100	4
CharacterTrajectories	1,422	1,436	3	182	20
Cricket	108	72	6	1,197	12
DuckDuckGeese	60	40	1,345	270	5
EigenWorms	128	131	6	17,984	5
Epilepsy	137	138	3	206	4
EthanolConcentration	261	263	3	1,751	4
ERing	30	30	4	65	6
FaceDetection	5,890	3,524	144	62	2
FingerMovements	316	100	28	50	2
HandMovementDirection	320	147	10	400	4
Handwriting	150	850	3	152	26
Heartbeat	204	205	61	405	2
JapaneseVowels	270	370	12	29	9
Libras	180	180	2	45	15
LSST	2,459	2,466	6	36	14
InsectWingbeat	30,000	20,000	200	78	10
MotorImagery	278	100	64	3,000	2
NATOPS	180	180	24	51	6
PenDigits	7,494	3,498	2	8	10
PEMS-SF	267	173	963	144	7
Phoneme	3,315	3,353	11	217	39
RacketSports	151	152	6	30	4
SelfRegulationSCP1	268	293	6	896	2
SelfRegulationSCP2	200	180	7	1,152	2
SpokenArabicDigits	6,599	2,199	13	93	10
StandWalkJump	12	15	4	2,500	3
UWaveGestureLibrary	120	320	3	315	8

4.3.2 Baselines

We consider several popular machine learning methods and recently proposed deep learning models as baselines. The selected competitive methods include ROCKET [30], Time Series Transformer (TST) [151], ShapeNet [81], Dynamic Time Warping (DTW), TS2Vec [147], MLSTM-FCN [71], OS-CNN [109], TapNet [155], Temporal Neighborhood Coding (TNC) [111], and WEASEL+ MUSE [99].

4.3.3 Model Configuration and Evaluation Metric

We trained our model for 500 training epochs using Adam [75] optimizer. The learning rate is initialized to 0.001; it scales down with a coefficient of 0.1 every 50 epochs after the first 100 epochs. We repeated the training and test processes five times and took the average of multiple runs as the final results to mitigate the impact of randomized parameter initialization. We used dropout to avoid possible overfitting. Training and

Table 4.2: Accuracy of different models on 30 benchmark datasets. The best performance values are bolded, and the second-best performance values are underlined.

Dataset	Ours	WEASEL+MUSE	TST	ROCKET	DTW	TS2Vec	MLSTM-FCN	OS-CNN	TapNet	TNC	ShapeNet
ArticularyWordRecognition	**0.996**	<u>0.990</u>	0.977	**0.996**	0.987	0.987	0.973	0.988	0.987	0.973	0.987
AtrialFibrillation	**0.524**	0.333	0.067	0.249	0.200	0.200	0.267	0.233	0.333	0.133	<u>0.400</u>
BasicMotions	**1.000**	**1.000**	0.975	<u>0.990</u>	0.975	0.975	0.950	**1.000**	**1.000**	0.975	**1.000**
CharacterTrajectories	<u>0.997</u>	0.990	0.975	0.967	0.989	0.995	0.985	**0.998**	<u>0.997</u>	0.967	0.980
Cricket	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.972	0.917	<u>0.993</u>	0.958	0.958	0.986
DuckDuckGeese	**0.767**	0.575	0.562	0.461	0.492	0.680	0.675	0.540	0.575	0.460	<u>0.725</u>
EigenWorms	**0.897**	<u>0.890</u>	0.748	0.863	0.618	0.847	0.504	0.414	0.489	0.840	0.878
Epilepsy	**1.000**	**1.000**	0.949	<u>0.991</u>	0.964	0.964	0.761	0.980	0.971	0.957	0.987
Ering	0.875	0.133	**0.964**	0.447	0.133	0.874	0.133	<u>0.881</u>	0.133	0.852	0.133
EthanolConcentration	**0.476**	0.430	0.326	<u>0.452</u>	0.323	0.308	0.373	0.240	0.323	0.297	0.312
FaceDetection	**0.683**	0.545	<u>0.681</u>	0.647	0.529	0.501	0.545	0.575	0.556	0.536	0.602
FingerMovements	**0.601**	0.490	0.560	0.553	0.530	0.480	<u>0.580</u>	0.568	0.530	0.470	<u>0.580</u>
HandMovementDirection	<u>0.443</u>	0.365	0.243	**0.446**	0.231	0.338	0.365	0.443	0.378	0.324	0.338
HandWriting	**0.672**	0.605	0.359	0.567	0.286	0.515	0.286	<u>0.668</u>	0.357	0.249	0.451
HeartBeat	**0.863**	0.727	<u>0.776</u>	0.717	0.717	0.515	0.663	0.489	0.751	0.746	0.756
JapaneseVowels	0.967	0.984	**0.994**	0.962	0.949	0.984	0.976	<u>0.991</u>	0.965	0.978	0.984
Libras	**0.981**	<u>0.973</u>	0.656	0.906	0.870	0.867	0.856	0.950	0.850	0.817	0.856
LSST	<u>0.782</u>	**0.878**	0.408	0.632	0.551	0.537	0.373	0.413	0.568	0.595	0.590
MotorImagery	**0.632**	0.590	0.500	0.531	0.500	0.510	0.510	0.535	0.590	0.500	<u>0.610</u>
NATOPS	<u>0.941</u>	0.500	0.850	0.885	0.883	0.928	0.889	**0.968**	0.939	0.911	0.883
PEMS-SF	**0.932**	0.870	<u>0.919</u>	0.751	0.711	0.682	0.699	0.760	0.751	0.699	0.751
PenDigits	<u>0.991</u>	0.968	0.560	**0.996**	0.977	0.989	0.978	0.985	0.980	0.979	0.977
Phoneme	0.287	0.190	0.085	0.284	0.151	0.233	0.110	**0.299**	0.175	0.207	<u>0.298</u>
RacketSports	**0.934**	0.190	0.809	<u>0.928</u>	0.803	0.855	0.803	0.877	0.868	0.776	0.882
SelfRegulationSCP1	**0.961**	<u>0.934</u>	0.925	0.908	0.775	0.812	0.874	0.835	0.652	0.799	0.782
SelfRegulationSCP2	**0.738**	<u>0.710</u>	0.589	0.533	0.539	0.578	0.472	0.532	0.550	0.550	0.578
SpokenArabicDigits	<u>0.994</u>	0.460	0.993	0.712	0.963	0.988	0.990	**0.997**	0.983	0.934	0.975
StandWalkJump	**0.659**	0.333	0.267	0.456	0.200	0.467	0.067	0.383	0.400	0.400	<u>0.533</u>
UWaveGestureLibrary	**0.951**	0.916	0.903	<u>0.944</u>	0.903	0.906	0.891	0.927	0.894	0.759	0.906
InsectWingBeat	**0.697**	0.163	0.105	0.168	0.105	0.466	0.167	<u>0.667</u>	0.208	0.469	0.250
Average Accuracy	**0.808**	0.658	0.658	0.698	0.628	0.698	0.621	<u>0.704</u>	0.657	0.670	0.699
Average Rank	**1.57**	4.93	6.63	4.93	7.83	6.17	7.77	<u>4.77</u>	6.07	8.00	4.83

testing are done on a single Nvidia GTX 3080 Ti.

We use *accuracy*, which is currently used by all baseline methods, as the metric for comparison. We additionally use macro *precision*, *recall*, and *F1-Score* in our parameter and ablation studies to gain further insights into our model's performance.

4.3.4 Comparison Results

The performance comparison results (shown in Table 4.2) reveal that our model has demonstrated superior performance to all the baseline methods across a wide range of experimental datasets. Specifically, our model achieved the best results on 21 datasets, the second-best performance on six datasets, and the third-best on two datasets out of 30 experimental datasets. It demonstrated superior performance compared to all baselines, achieving a 14.7% increase in average classification accuracy compared to the second-best method, OS-CNN, and a 15.6% increase compared to the third-best method, ShapeNet. Furthermore, our model achieved an average rank of 1.57, outperforming the second-best method, OS-CNN, which had an average rank of 4.77. Figure 4.4 shows the result of the Wilcoxon signed-rank test (with a confidence level of 95%) on the baseline methods'

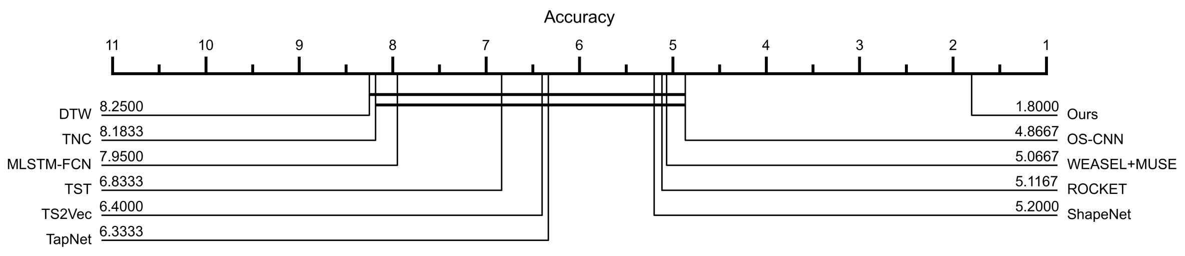


Figure 4.4: Critical Difference (CD) diagram of the selected baselines and our method with a confidence level of 95%.

performance, consistently showing that our method achieved the highest classification performance among all the compared methods.

Traditional machine learning methods, including WEASEL+MUSE, DTW, and ROCKET, are limited in handling such large datasets, reflected in their inferior performance on datasets including InsectWingBeat and FaceDetection, which contain 50,000 and 9,114 samples, respectively. Furthermore, existing deep learning models often ignore the inherent frequency information in time series data, which can be crucial for accurately classifying time series with significant differences in the frequency domain rather than the time domain.

We attribute this improvement to two key factors. First, our method's dual-stream architecture effectively captures both temporal and frequency information, enhancing its ability to discriminate time series sequences between different classes. Second, by utilizing the Short-Time Fourier Transform (STFT), our method leverages the frequency components and their temporal locations of the time series to provide more comprehensive frequency information compared to the Fourier Transform. Our results from the Wilcoxon signed-rank test, conducted with a confidence level of 95%, further confirm that our method achieved the best classification performance among all compared methods.

4.3.5 Convolutional Backbone Selection Sensitivity

We replaced the ResNet with other popular computer vision backbones including ResNeXt [133], Res2Net [44], ResNeSt [153], and Inception [108] to explore the impact of the backbone selection on the performance. We conducted experiments on three datasets, including **DuckDuckGeese**, **HeartBeat**, and **HandWriting**. The results can be found in Table 4.3. The tested backbones have more complex architectures and parameters compared to ResNet, leading to better performance during the training phase but overfitting on the test sets. We believe that with the increase of the dataset scale, implementing more

Table 4.3: The Training and test results of different backbones from the computer vision field. The best performance values are bolded.

		Training				Test			
Dataset	Models	Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
DuckDuckGeese	ResNet	0.862	0.813	0.764	0.788	**0.767**	**0.741**	0.736	**0.738**
	ResNeXt	0.866	0.809	**0.866**	0.837	0.673	0.645	0.639	0.642
	Res2Net	0.894	0.815	0.827	0.821	0.639	0.613	0.632	0.622
	ResNeSt	**0.907**	**0.915**	0.855	**0.884**	0.692	0.704	**0.761**	0.731
	Inception	0.859	0.897	0.811	0.852	0.734	0.729	0.736	0.732
HeartBeat	ResNet	0.906	0.891	0.882	0.886	0.863	0.772	0.795	0.783
	ResNeXt	0.916	0.902	0.914	0.908	0.741	0.726	0.719	0.722
	Res2Net	**0.931**	**0.919**	0.922	0.920	0.714	0.678	0.669	0.673
	ResNeSt	0.928	0.917	**0.927**	**0.922**	0.665	0.640	0.608	0.624
	Inception	0.909	0.874	0.907	0.890	**0.782**	**0.738**	**0.806**	**0.771**
HandWriting	ResNet	0.735	0.702	0.744	0.722	**0.672**	**0.654**	**0.661**	**0.657**
	ResNeXt	0.849	0.865	0.872	0.868	0.533	0.542	0.591	0.565
	Res2Net	0.856	0.802	0.874	0.836	0.592	0.607	0.586	0.596
	ResNeSt	**0.857**	**0.886**	**0.883**	**0.884**	0.557	0.573	0.605	0.589
	Inception	0.764	0.753	0.773	0.763	0.597	0.612	0.596	0.604

complicated backbones may enhance the classifier's classification capacity. As most of the datasets we use contain limited samples in the training set (fewer than 1000), we selected ResNet as the optimal solution based on our evaluation of the performance metrics.

4.3.6 Impact of Our Spectrogram Stream as a Plugin

We incorporate the spectrogram stream as a plugin into the existing architectures including TST [151] and MLSTM-FCN [71] to evaluate the effectiveness of the 2D representations with frequency information in improving the performance of the existing methods. We conducted experiments on two datasets: **EigenWorms** and **RacketSports**. The outcomes of our investigation, as presented in Table 4.4, indicate a significant improvement in the average classification accuracy and the F1-Score of both methods during both the training and testing phases. Specifically, we observed an increase of 8.6% and 7.3% in the average classification accuracy and F1-Score, respectively, during the training phase, and an increase of 9.6% and 10.4% in the average classification accuracy and F1-Score, respectively, during the test phase. These findings suggest that the utilization of 2D representations with frequency information can enhance the performance of existing methods.

Table 4.4: The experimental results when using our frequency stream with 2D representations as a plugin. W/o means that the method does not contain the spectrogram stream and vice versa. The best performance values are bolded.

		Train				Test			
Dataset	Method	Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
EigenWorms	MLSTM-FCN (w/o)	0.587	0.574	0.624	0.598	0.504	0.519	0.479	0.498
	MLSTM-FCN (w)	**0.721**	**0.714**	**0.677**	**0.695**	**0.629**	**0.624**	**0.595**	**0.609**
	TST (w/o)	0.839	0.832	0.816	0.824	0.748	0.791	0.778	0.784
	TST (w)	**0.882**	**0.893**	**0.885**	**0.889**	**0.826**	**0.828**	**0.819**	**0.823**
RacketSports	MLSTM-FCN (w/o)	0.828	0.779	**0.833**	0.805	0.803	0.709	0.702	0.705
	MLSTM-FCN (w)	**0.843**	**0.811**	0.805	**0.808**	**0.814**	**0.727**	**0.751**	**0.739**
	TST (w/o)	0.854	0.819	0.822	0.820	0.809	0.712	0.705	0.708
	TST (w)	**0.894**	**0.833**	**0.882**	**0.857**	**0.824**	**0.762**	**0.793**	**0.777**

Table 4.5: Ablation test for our method. Fourier Transform means we use Fourier Transform instead of STFT to extract frequency information. Single window size means we only use one window size (three times the bandwidth) to generate the spectrogram. Time Series and Spectrogram Stream only mean using information from one stream separately instead of both to classify time series. The best performance values are bolded.

Dataset	Model	Accuracy	Precision	Recall	F1-Score
DuckDuckGeese	Fourier Transform	0.675	0.669	0.688	0.678
	Single Window Size	0.689	0.707	0.725	0.716
	Time Series Stream Only	0.632	0.619	0.661	0.639
	Spectrogram Stream Only	0.718	0.711	0.724	0.717
	Ours	**0.767**	**0.741**	**0.736**	**0.738**
FaceDetection	Fourier Transform	0.575	0.602	0.552	0.576
	Single Window Size	0.627	0.585	0.673	0.626
	Time Series Stream Only	0.630	0.615	0.622	0.618
	Spectrogram Stream Only	0.647	0.651	0.642	0.646
	Ours	**0.681**	**0.622**	**0.716**	**0.666**
PEMS-SF	Fourier Transform	0.751	0.643	0.637	0.640
	Single Window Size	0.794	0.718	0.698	0.708
	Time Series Stream Only	0.819	0.822	0.803	0.812
	Spectrogram Stream Only	0.874	0.856	0.877	0.866
	Ours	**0.932**	**0.957**	**0.889**	**0.922**

4.3.7 Ablation Study

We conducted ablation studies on three datasets, including **DuckDuckGeese**, **FaceDetection**, and **PEMS-SF**, to investigate the effectiveness of individual components of our proposed method. We compared the performance of the method with the use of Fourier Transform instead of STFT to extract frequency information. Besides, for STFT, we use a single window size (three times the bandwidth) for spectrogram generation instead of three window sizes. Additionally, we tried to use information from one single stream (either the time series or spectrogram stream) individually to classify time series instead of both. The experimental results are summarized in Table 4.5.

Our analysis reveals that each component improves the classifier's performance. Notably, STFT demonstrates a more significant impact on the classification accuracy of the model on two of the datasets. This finding implies that the utilization of 2D representations with frequency information provided by STFT is crucial for enhancing the classification capacity of the model.

4.4 Conclusion

This study proposes a novel dual-stream architecture for accurately classifying multivariate time series sequences. The method leverages the inherent frequency information in the time series data by implementing STFT to obtain the frequency components and their temporal positions. We construct a dual-stream architecture based on ResNet, which can leverage both 1D and 2D representations effectively to classify multivariate time series sequences. We evaluate the proposed model on diverse datasets containing sequences of various lengths and variable numbers. The experimental results show that our method outperforms several baseline and state-of-the-art methods by a significant margin. We also conduct a thorough investigation of the effect of different components and settings on the model's performance. Through the proposed method, we can effectively leverage the inherent frequency information of the time series to realize more accurate classification.

CHAPTER 5

DYFORMER: A DYNAMIC TRANSFORMER-BASED ARCHITECTURE FOR MULTIVARIATE TIME SERIES CLASSIFICATION

5.1 Introduction

Multivariate time series classification is inherently challenging as it requires incorporating temporal patterns and inter-relations among multiple variables simultaneously to obtain accurate results. Transformer-based models are increasingly applied to multivariate time series classification, given their success in computer vision and natural language processing in recent years [2, 113]. Examples of such models include Time Series Transformer (TST) [151], and Task-Aware Reconstruction for Time-Series Transformer (TARNet) [27]. The former adapts the vanilla transformer architecture for multivariate time series classification, while the latter uses multivariate time series classification as the downstream task to guide the pre-training.

Recent studies [114, 119, 134, 159] show that dynamic architectures are crucial for enhancing representation learning and models' generalization ability. These methods can adjust their parameters and architectures during training, resulting in improved representation power [53]. Besides, the dynamic architecture of a model allows itself to better capture the patterns and features of the data it is trained on, making it more generalizable to diverse types of data. However, existing transformer-based models for multivariate time series classification have limited ability to adapt to specific datasets.

While the models' parameters are optimized through training, their structures remain unchanged during the training process. This hinders their representation learning capacity.

We aim to design a dynamic architecture to achieve better adaptivity of transformer models for multivariate time series classification. Consequently, we propose *Dyformer*, a dynamic transformer-based architecture that incorporates adaptive learning strategies for different frequency components of time series to accommodate characteristics of specific datasets. In particular, we design a hierarchical pooling layer to decompose time series sequences into subsequences that contain different frequency components (or frequency patterns), laying the foundation for adapting the learning strategies for different frequency components. We devise a novel Dyformer module consisting of a gate and a Dyformer block to enable the adaptation of the model structure for each frequency component, where the Dyformer block leverages the architecture introduced in Chapter 3 to realize feature-map-wise attention for capturing multi-scale temporal dependencies and the gate enables the bypassing of Dyformer blocks. As such, a Dyformer module can adjust its structure dynamically for each subsequence, generating an output (the extracted features) specific to the respective subsequence. We also design a joint loss function, which adds a gate penalization term to the task loss to train the proposed model.

5.2 Methodology

The architecture of *Dyformer* (Figure 5.1) consists of three main components: hierarchical pooling, Dyformer modules, and information fusion layer. *Dyformer* works as follows: 1) Hierarchical pooling decomposes the input time series into subsequences representing different frequency patterns; 2) Each subsequence is processed by a stack of Dyformer modules, with each Dyformer module containing a gate and a Dyformer block. The gate determines whether to feed the Dyformer module's input to the Dyformer Block, while the Dyformer block achieves feature-map-wise attention to leverage multi-level temporal dependencies; 3) The information fusion layer takes the concatenated outputs of Dyformer modules to obtain the final output; 4) A fully-connected layer with *Softmax* maps the output to label probability distributions.

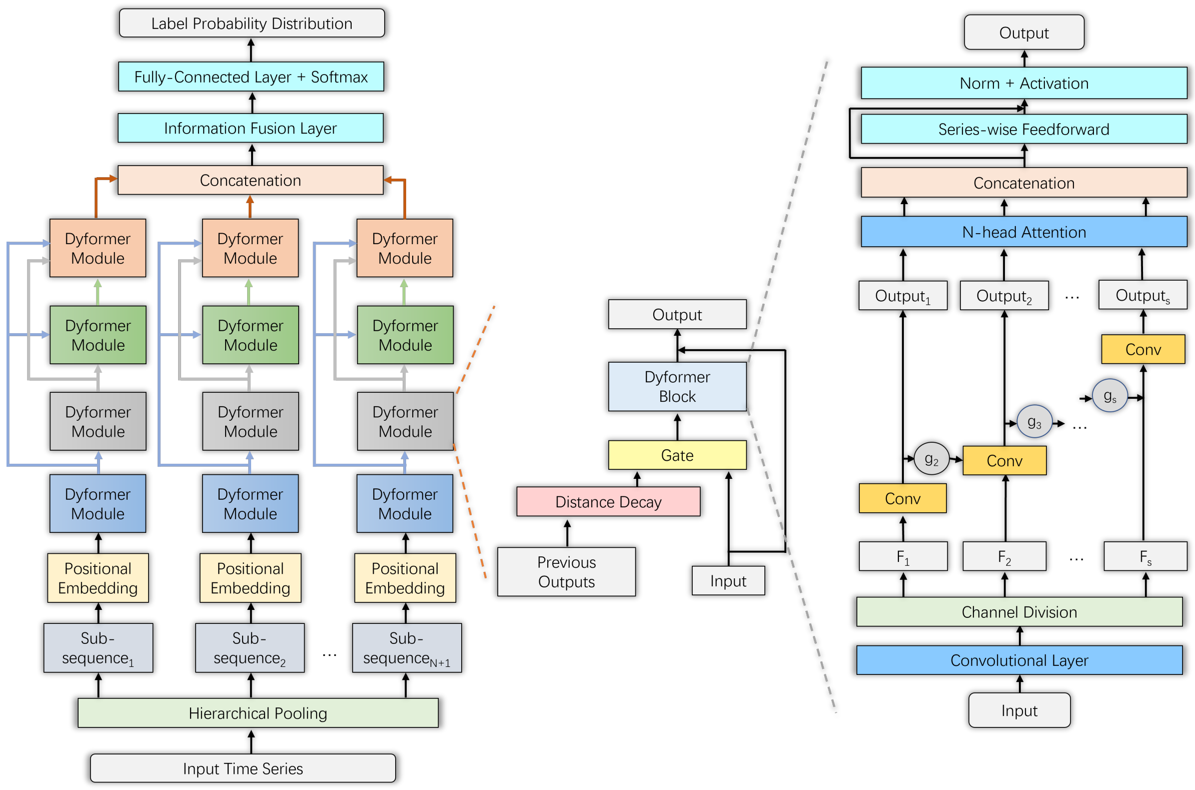


Figure 5.1: **Left**: The overall structure of Dyformer. First, the hierarchical pooling layer decomposes the input time series sequences into N+1 subsequences. Then, each subsequence is processed by a stack of multiple Dyformer modules (there are four in this illustrative example) for feature extraction. Finally, an information fusion layer takes the concatenation of all the output feature maps, and a fully-connected layer with $Softmax$ obtains the classification results in terms of label probability distributions. **Middle**: The structure of the Dyformer module. A Dyformer module contains a hard gate (in yellow) and a Dyformer block. The output feature maps of all previous Dyformer modules determine the gate's status. If the gate is open, the Dyformer block takes the prior Dyformer module's output for feature extraction; otherwise, it directly outputs its immediate input. **Right**: The structure of the Dyformer block. A Dyformer block uses hierarchically connected group convolution to achieve feature-map-wise multi-head attention for leveraging multi-level temporal dependencies and soft gates (in gray) to control the information flow and series-wise feedforward, enabling it to focus on series-wise temporal patterns.

5.2.1 Hierarchical Pooling

Traditionally, a time series decomposition method [17] decomposes time series into three distinct components, namely the trend, seasonal, and bias terms. To the best of our knowledge, there is no deep learning-based method that incorporates decomposition methods for multivariate time series classification yet. Although some time series forecasting methods [45, 118, 125] have incorporated decomposition techniques, they utilize separate modules to handle trend and seasonal terms while considering the bias term as high-frequency noise and ignoring it.

However, it is crucial for a time series decomposition method to consider all the frequency components instead of certain particular components. While a time series forecasting problem focuses on temporal variation, multivariate time series classification involves multiple time series that belong to different classes. Thus, multivariate time series classification requires capturing the distinct characteristics among various classes manifested in all frequency bands.

We use examples from two real-world datasets to illustrate the significance of considering different frequency components in multivariate time series classification. Specifically, we sample time series sequences from two public real-world datasets **AtrialFibrillation**[1] and **BasicMotions**[2]. Then, we explore the differences of sequences from the time and frequency domains, respectively. Figure 5.2 and Figure 5.3 show samples of time series data from the **AtrialFibrillation** dataset and the **BasicMotions** dataset, respectively. Each figure shows two random samples from the same dataset yet belonging to different classes and depicts these samples in the time domain and the frequency domain, respectively. We also show the difference between the two samples in each domain. The two figures exemplify how different frequency components may play different roles in different classification problems. Specifically, Figure 5.2 (f) shows the two samples mainly differ in the high-frequency and low-frequency parts, while in the case of Figure 5.3 (f), the difference between the two samples spans across the full frequency spectrum, making it unjustifiable to consider any frequency band as noises for multivariate time series classification. Beyond the above example, a recent study on computer vision [116] has shown that high-frequency components can contribute to the generalization ability of image classification models. This finding further motivates us to preserve all the frequency components for multivariate time series classification.

[1]`http://www.timeseriesclassification.com/description.php?Dataset=AtrialFibrillation`

[2]`http://www.timeseriesclassification.com/description.php?Dataset=BasicMotions`

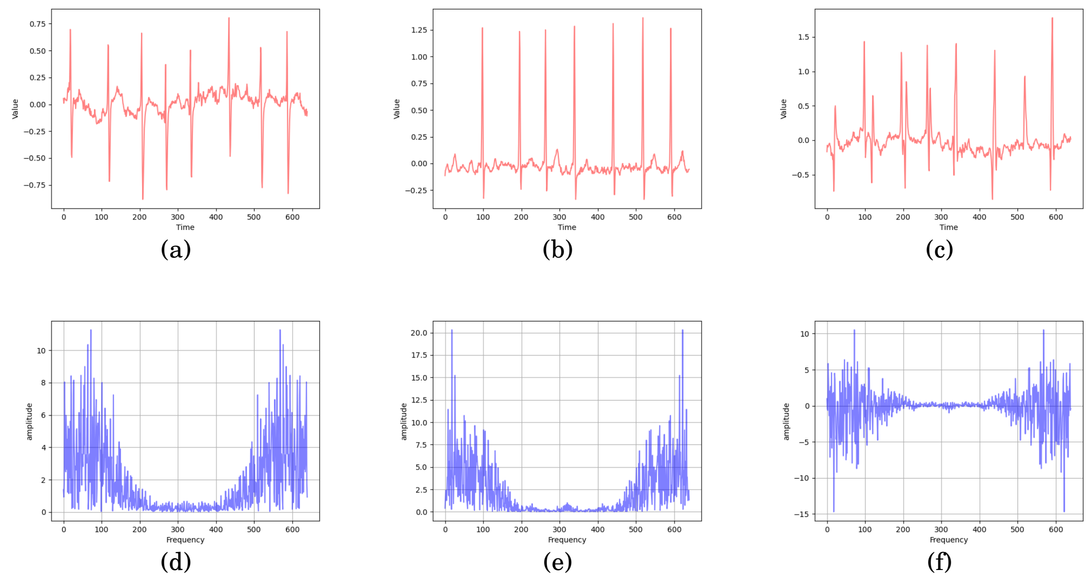


Figure 5.2: (a) A sample of class 1 (non-termination atrial fibrillation); (b) A sample of class 2 (atrial fibrillation that self-terminates at least one minute after the recording process); (c) The difference between the two samples in the time domain; (d) The spectrogram of the first sample (from class 1); (e) The spectrogram of the second sample (from class 2); (f) The difference between the spectrograms of the two samples.

Given that time series decomposition is an effective approach for time series analysis, we propose a novel hierarchical pooling layer that decomposes time series to improve the performance of multivariate time series classification. We are not limited to the three terms (namely trend, seasonal, and bias) but develop a more general approach to decompose time series. More specifically, we obtain a series of subsequences that represent different frequency components without information loss[3].

Let $Input_1$ be the original input. The hierarchical pooling layer outputs a series of subsequences:

$$Output = \left[Output_1, \dots, Output_N, Input_{N+1}\right], \tag{5.1}$$

where N is a tunable hyperparameter that controls the number of subsequences. We apply *average pooling* N times to generate $N+1$ subsequences as follows:

$$\begin{aligned} Output_n &= AvgPooling\left(Input_n\right) \\ Input_{n+1} &= Input_n - Output_n, \end{aligned} \tag{5.2}$$

[3]A proof for the no-information-loss property can be found in section **??**

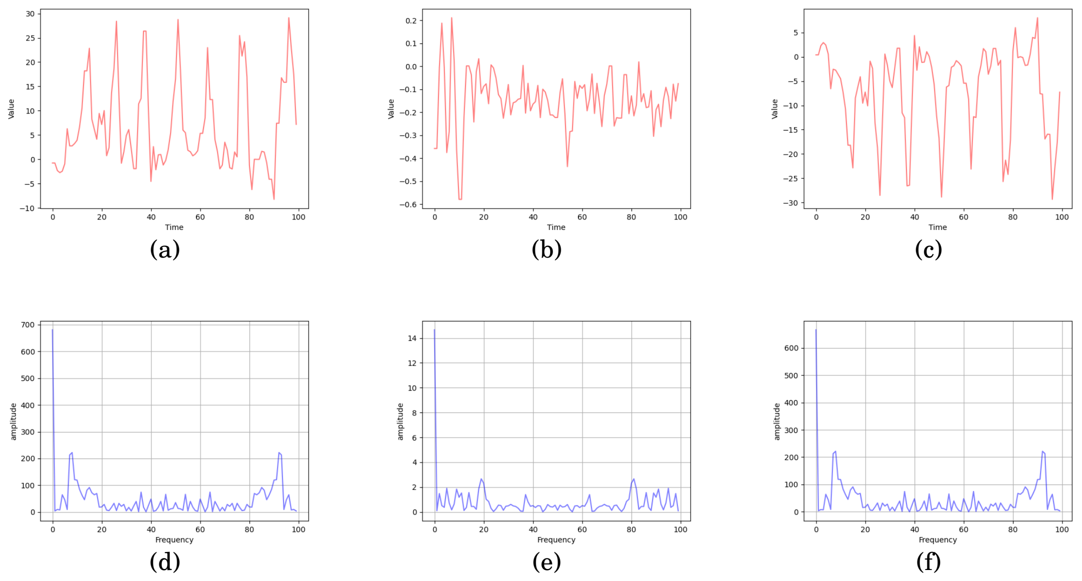


Figure 5.3: (a) A sample of class 3 (running); (b) A sample of class 4 (badminton); (c) The difference between the two samples in the time domain; (d) The spectrogram of the first sample (from class 3); (e) The spectrogram of the second sample (from class 4); (f) The difference between the spectrograms of the two samples.

where $n \in \{1,\ldots,N\}$. The hierarchical pooling layer offers two advantages: 1) it can effectively decompose a time series into multiple subsequences with various frequency components; 2) it can be easily implemented using the widely-used average pooling layer, ensuring its seamless integration into any deep learning architecture to support end-to-end training.

5.2.2 Dyformer Module

The Dyformer module is the essential component to achieve dynamic structure and adaptive learning strategies for different frequency components in our approach. A Dyformer module contains a gate and a Dyformer block: the former determines whether the input can be fed into the latter for feature extraction via feature-map-wise attention; if not, the Dyformer block of the module will be skipped and the Dyformer module's output will be the same as the input. As such, our Dyformer can realize a dynamic architecture for time series sequences with different frequency components to realize adaptive learning strategies. We give more details about each component of the Dyformer module in the following subsections.

5.2.2.1 Gate

We implement a gate before every Dyformer block to provide flexibility in activating Dyformer blocks. The gate takes in previous Dyformer modules' output feature maps to determine the accessibility of the next Dyformer block. By deciding the gate status, the model can dynamically adjust the network structure, varying learning strategies for different frequency patterns.

Specifically, the gate (denoted by $Gate_M$) for the Mth Dyformer block considers all the previous output feature maps $[F_1, F_2, ..., F_{M\text{-}1}]$, where $F \in \mathbb{R}^{C \times L}$ stands for the output feature map that decides the gate's status, C is the number of channels, and L is the length of the feature map. We propose *Distance Decay* to represent the importance of each feature map based on its distance to the current Dyformer module. The above gate design poses two advantages: 1) it considers the information more comprehensively; 2) the next Dyformer module will not take in the same feature maps for gate calculation if the previous Dyformer block is skipped. In contrast, if only the previous output feature maps are considered, skipping a Dyformer block will result in the subsequent gates taking in the same feature map, leading to their status being the same.

Distance Decay is described below:

$$F'_m = F_m \times e^{k_m(m-M+1)}, \tag{5.3}$$

where $e^{k_m(m\text{-}M+1)}$ is the decay term, and k_m is a learnable parameter initialized to 1. The decay term for the closest output feature map $F_{M\text{-}1}$ is always 1 (which means no decay), while values of all the other output feature maps are exponentially reduced. A convolutional layer takes the sum of all the decayed feature maps to generate a decision value V_M for the $Gate_M$:

$$V_M = \tanh\Big(\text{Mean}\big(\text{Conv}(\sum_{1}^{M-1} F_m \times e^{k_m*(m-M+1)})\big)\Big), \tag{5.4}$$

and we use it to calculate the $Gate_M$:

$$Gate_M = \begin{cases} 1, & V_M \geq 0 \\ 0, & V_M < 0. \end{cases} \tag{5.5}$$

We feed the $F_{M\text{-}1}$ to the current Dyformer block for feature extraction when $Gate_M$ is 1; otherwise, we skip the Mth Dyformer block and feed the prior feature map $F_{M\text{-}1}$ directly to the next Dyformer module. Since Eq. (5.5) is non-differentiable, we use *reparameterization* [67] to update the gate status during training.

5.2.2.2 Dyformer Block

The Dyformer block adopts the architecture introduced in Chapter 3 (Figure 5.1) which constructs groups of convolutional filters in a hierarchy and feeds the outputs to a multi-head attention layer to implement feature-map-wise attention. As the point-wise attention used in TST [151] can not capture the rich temporal variations of the time series sequences, existing works are prone to use convolutional layers [82] or recurrent layers [65] to harness the locality temporal information for attention calculation. However, the feature maps obtained after feeding time series into neural networks contain multi-level temporal information, i.e., long- and short-term dependencies, which convolutional or recurrent layers can not fully leverage. Hence, we propose feature-map-wise attention to capture multi-level temporal dependencies.

Specifically, given the input feature map, it is first fed to a 1D convolutional layer, expanding the channels to n. We then split the feature map into groups along the channel to obtain s groups of feature maps. Specifically, we construct s groups of convolutional filters (kernel size 3, stride 1, padding 1) with w channels (to avoid information loss, we set $n = s \times w$). A separate group of filters harnesses the temporal information for each input feature map and generates the corresponding output. Each filter group considers the output of the filter group that comes immediately before it when extracting features from an input feature map. The above process repeats until all input feature maps are processed. We use s-1 soft gates for controlling the information flow between adjacent groups of filters—the first group of convolutional filters does not have a gate.

The above computational process can be formulated below:

$$y_i = \begin{cases} \text{Conv}(x_i) & i = 1 \\ \text{Conv}(x_i + g_i \cdot y_{i-1}) & 1 < i \leqslant s \end{cases} \tag{5.6}$$

where x_i is the input feature map, y_i is the output feature map, g_i is the corresponding soft gate, $i \in \{1, 2, \cdots, s\}$.

Given an input feature-map group, x_i, the value of the corresponding gate, g_i, is calculated by:

$$g_i = \tanh(\text{Conv}(\text{concat}(\text{Conv}(y_{i-1}), \text{Conv}(x_i)))), \tag{5.7}$$

where y_{i-1} is the prior output group of feature maps.

All the output feature maps $\{y_i\}$ are sent to a standard multi-head attention layer to achieve feature-map-wise attention, where the Dyformer block calculates $query$, key, and $value$ of feature maps (rather than a single time step) to capture both long-term and short-term temporal dependencies. A series-wise feedforward layer follows by processing the multi-head attention layer's output based on a convolutional layer, which focuses on a region—which is in contrast to a point-wise feedforward layer—and is thus more suitable for handling time series data. The series-wise feedforward can be described as:

$$y = W * x + b \tag{5.8}$$

where $*$ means convolution, x and y are the input and output, respectively, and W and b are learnable parameters. Finally, normalization and ReLU activation functions generate the final output for the entire Dyformer block. As such, a Dyformer Block's output contains a combination of receptive fields of various sizes to fuse multi-granular feature maps. As we construct more groups, the input of each group of convolutional filters contains its own input and the outputs of all the previous groups. In this way, the Dyformer Block progressively fuses temporal dependencies at various levels of granularity and achieves effective temporal feature extraction.

5.2.3 Information Fusion Layer

The information fusion layer takes the concatenation of the output feature maps of all paths and sends the output to a fully-connected layer and $Softmax$ to obtain the label probability distribution. We propose a two-stage information fusion strategy. Accordingly, the information fusion layer stacks one 2D convolutional layer and one 1D convolutional layer to fuse the feature maps progressively: 1) the 2D convolutional layer fuses the temporal information from the path's perspective; 2) the 1D convolutional layer fuses the information in the channel dimension.

5.2.4 Loss Function

We train Dyformer in a supervised manner using a joint loss function that consists of two terms: task loss term and gate penalization term. The cross-entropy loss is used for the classification task, and we design a new term to penalize the open probabilities of hard gates (i.e., the gates before Dyformer blocks). The overall loss is as follows:

$$\begin{aligned} \text{Loss } = & -\frac{1}{\#samples} \sum_{i=1}^{\#samples} \sum_{c=1}^{C} y_{ic} \log(p_{ic}) + \\ & \gamma \frac{1}{M*(N+1)} \sum_{1}^{N+1} \sum_{1}^{M} V_{mn}, \end{aligned} \tag{5.9}$$

where $\#samples$ is the number of training samples, C is the number of classes, and c denotes the class index. i denotes the ith sample, m and n are the indices of the Dyformer module and subsequence, respectively. y_{ic} indicates whether the ith sample in the cth class, while p_{ic} denotes the corresponding probability. γ is the hyperparameter controlling the influence of the penalty term. M is the number of Dyformer blocks in each path, and m denotes the mth Dyformer block. N+1 is the number of subsequences, and n denotes the nth subsequence.

5.2.5 Unsupervised Pre-training

Alternatively, we can train Dyformer in an unsupervised pre-training manner. To this end, we randomly mask every time step in the time series at a probability of 0.5, which is a commonly used probability for unsupervised time series representation learning [148], and replace $Softmax$ following the information fusion layer with a fully-connected layer, forcing the model to reconstruct the input time series sequence. Inspired by Reinforcement Learning, We first set all gates open to learn the temporal representation for the stable "environment" and regard time series reconstruction as a regression task with Mean Square Error (MSE) loss.

$$\text{Loss} = \frac{1}{\#samples} \sum_{i=1}^{\#\text{sample}} (\breve{y}_i - y_i)^2, \tag{5.10}$$

where $\breve{y}_i$ is the output, and y_i is the ground truth label. Once pre-trained, we fine-tune our model using the loss function as specified in Eq. (5.9).

5.3 Experiments

5.3.1 Datasets

We evaluated Dyformer using the UEA Time Series Classification Repository [29][4], which contains 30 public multivariate time series datasets. The datasets under consideration pertain to a range of domains and exhibit various data characteristics with respect to the length of sequences and the number of variables. All datasets had been preprocessed and split into training and test sets. In addition, we performed normalization of the datasets to establish a zero mean and unit standard deviation. We applied zero paddings to ensure that the datasets contained sequences of equal lengths.

[4]`http://www.timeseriesclassification.com/dataset.php`

5.3.2 Baseline Methods

We carefully select several recent and competitive machine learning and deep learning models as the baselines to evaluate our approach. The selected methods include ROCKET [30], Time Series Transformer (TST) [151], Task-Aware Reconstruction for Time-Series Transformer (TARNet) [27], ShapeNet [81], Dynamic Time Warping (DTW), TS2Vec [147], MLSTM-FCN [71], OS-CNN [109], TapNet [155], Temporal Neighborhood Coding (TNC) [111], Temporal and Contextual Contrasting (TS-TCC) [37], and WEASEL+ MUSE [99]

To avoid suboptimal performance outcomes due to potential disparities in implementation details, we meticulously gathered and relied upon the experimental results of these baseline methods as reported in other reputable publications instead of undertaking reimplementation efforts. We meticulously gathered and relied upon the experimental results of these baseline methods as reported in other reputable publications. Specifically, we leveraged resources pertaining to the experimental results of all baseline methods from two primary sources: OS-CNN and TARNet. This approach ensured the credibility and consistency of our comparative analyses.

5.3.3 Model Configuration and Evaluation Metric

We used Adam [75] optimizer to train our model for 600 epochs. The initial learning rate is 0.001; it scales down with a coefficient of 0.1 every 50 epochs after the first 100 epochs. We trained and tested the model for five times and calculated the average of multiple runs as the final results for mitigating the influence of randomized parameter initialization. We used dropout to avoid possible overfitting. Training and testing are done on a single Nvidia GTX 3080 Ti.

For the hierarchical pooling layer, we set kernel size=5, padding=2, and stride=1 to keep the lengths of the decomposed subsequences unchanged. For pre-training, we first select the hyperparameters to minimize the MSE loss, and then we use the same configuration of supervised training for fine-tuning. Configuration details are given in Table 5.1.

We use *accuracy*, which is currently used by all baseline methods, as the metric for comparison. We additionally use macro *precision*, *recall*, and *F1-Score* in our parameter and ablation studies to gain further insights into our model's performance.

Table 5.1: Model configuration

Dataset	#subsequences	Dropout rate	#Dyformer Modules	#convolutional filter groups	#attention heads	γ
ArticularyWordRecognition	4	0.3	4	32	8	0.2
AtrialFibrillation	4	0.5	5	64	4	0.2
BasicMotions	3	0.5	4	16	4	0.2
CharacterTrajectories	5	0.2	4	32	4	0.3
Cricket	5	0.4	6	64	4	0.3
DuckDuckGeese	4	0.5	4	32	16	0.2
EigenWorms	5	0.3	8	64	4	0.2
Epilepsy	4	0.3	5	32	4	0.3
EthanolConcentration	5	0.5	5	64	4	0.3
ERing	3	0.5	4	16	4	0.2
FaceDetection	4	0.4	4	16	8	0.2
FingerMovements	3	0.3	4	16	8	0.3
HandMovementDirection	4	0.3	6	64	4	0.2
Handwriting	4	0.3	4	32	4	0.3
Heartbeat	3	0.3	6	64	8	0.1
JapaneseVowels	3	0.4	4	8	4	0.2
Libras	3	0.2	4	8	4	0.2
LSST	3	0.3	4	16	4	0.2
MotorImagery	5	0.4	8	64	8	0.1
NATOPS	3	0.3	4	16	8	0.3
PenDigits	3	0.4	4	4	4	0.1
PEMS-SF	5	0.4	4	32	16	0.2
Phoneme	4	0.2	4	32	4	0.2
RacketSports	3	0.4	4	8	4	0.1
SelfRegulationSCP1	5	0.5	4	64	4	0.2
SelfRegulationSCP2	5	0.4	6	64	4	0.2
SpokenArabicDigits	3	0.2	6	16	4	0.2
StandWalkJump	5	0.5	6	64	4	0.3
UWaveGestureLibrary	4	0.3	4	32	4	0.3
InsectWingbeat	5	0.2	4	8	16	0.3

5.3.4 Comparison Results

The performance comparison of all the methods (Table 5.2) shows that our model outperformed all the baselines, achieving the best results on 21 datasets, second-best on three datasets, and third-best on five datasets (out of 30 experimental datasets). Also, it achieved an average rank of 1.5, followed by the second-best of 4.9 achieved by TARNet. Figure 5.4 shows the result of the Wilcoxon signed-rank test (with a confidence level of 95%) on the baseline methods' performance. Overall, it shows that Dyformer trained in a supervised manner achieves the best classification performance. The results demonstrate the effectiveness of implementing adaptive learning strategies in multivariate time series classification, given that our model is the only one among the compared methods to achieve a dynamic architecture with time series decomposition for multivariate time series classification.

Traditional machine learning methods (e.g., DTW, ROCKET, and WEASEL+MUSE) did not perform well on large datasets, such as InsectWingBeat and FaceDetection, which contain 50,000 and 9,114 samples, respectively. CNN- and RNN-based models (e.g., MLSTM-FCN, OS-CNN, and TapNet) faced challenges dealing with long-ranged dependencies, reflected by poor classification accuracy on datasets containing long sequences

Table 5.2: Accuracy of different models on 30 benchmark datasets. The best performance values are bolded and the second-best performance values are underlined. *sup* and *pre* indicate supervised training and pre-training, respectively. The best performance of Dyformer and TST is used for rank calculation.

Dataset	Dyformer (sup)	Dyformer (pre)	TST (sup)	TST (pre)	ROCKET	DTW	TS2Vec	MLSTM -FCN	OS-CNN	TapNet	TNC	TS-TCC	WEASEL +MUSE	TARNet	ShapeNet
ArticularyWordRecognition	**0.997**	0.983	0.977	0.973	0.996	0.987	0.987	0.973	0.988	0.987	0.973	0.953	0.990	0.977	0.987
AtrialFibrillation	0.267	0.333	0.067	0.133	0.249	0.200	0.200	0.267	0.233	0.333	0.133	0.267	0.333	**1.000**	0.400
BasicMotions	**1.000**	**1.000**	0.975	0.925	0.990	0.975	0.975	0.950	**1.000**	**1.000**	0.975	**1.000**	**1.000**	**1.000**	**1.000**
CharacterTrajectories	**0.999**	0.996	0.975	0.972	0.967	0.989	0.995	0.985	0.998	0.997	0.967	0.985	0.990	0.994	0.980
Cricket	**1.000**	0.972	**1.000**	0.987	**1.000**	**1.000**	0.972	0.917	0.993	0.958	0.958	0.917	**1.000**	**1.000**	0.986
DuckDuckGeese	0.788	**0.818**	0.562	0.539	0.461	0.492	0.680	0.675	0.540	0.575	0.460	0.380	0.575	0.750	0.725
EigenWorms	**0.901**	0.870	0.748	0.756	0.863	0.618	0.847	0.504	0.414	0.489	0.840	0.779	0.890	0.420	0.878
Epilepsy	**1.000**	0.993	0.949	0.935	0.991	0.964	0.964	0.761	0.980	0.971	0.957	0.957	**1.000**	**1.000**	0.987
Ering	**0.970**	0.963	0.964	0.941	0.447	0.133	0.874	0.133	0.881	0.133	0.852	0.904	0.133	0.919	0.133
EthanolConcentration	0.397	0.432	0.326	0.337	**0.452**	0.323	0.308	0.373	0.240	0.323	0.297	0.285	0.430	0.323	0.312
FaceDetection	**0.732**	0.727	0.681	0.689	0.647	0.529	0.501	0.545	0.575	0.556	0.536	0.544	0.545	0.641	0.602
FingerMovements	**0.650**	0.570	0.560	0.530	0.553	0.530	0.480	0.580	0.568	0.530	0.470	0.460	0.490	0.620	0.580
HandMovementDirection	0.459	**0.473**	0.243	0.270	0.446	0.231	0.338	0.365	0.443	0.378	0.324	0.243	0.365	0.392	0.338
HandWriting	0.593	0.567	0.359	0.305	0.567	0.286	0.515	0.286	**0.668**	0.357	0.249	0.498	0.605	0.281	0.451
HeartBeat	**0.816**	0.785	0.776	0.776	0.717	0.717	0.515	0.663	0.489	0.751	0.746	0.751	0.727	0.780	0.756
JapaneseVowels	0.989	**0.998**	0.994	0.997	0.962	0.949	0.984	0.976	0.991	0.965	0.978	0.930	0.984	0.992	0.984
Libras	0.944	0.928	0.656	0.672	0.906	0.870	0.867	0.856	0.950	0.850	0.817	0.822	0.973	**1.000**	0.856
LSST	0.758	0.729	0.408	0.482	0.632	0.551	0.537	0.373	0.413	0.568	0.595	0.474	0.878	**0.976**	0.590
MotorImagery	**0.640**	0.610	0.500	0.510	0.531	0.500	0.510	0.510	0.535	0.590	0.500	0.610	0.590	0.630	0.610
NATOPS	0.933	0.911	0.850	0.806	0.885	0.883	0.928	0.889	**0.968**	0.939	0.911	0.822	0.500	0.911	0.883
PEMS-SF	**0.952**	0.935	0.919	0.896	0.751	0.711	0.682	0.699	0.760	0.751	0.699	0.734	0.870	0.936	0.751
PenDigits	**0.997**	0.995	0.560	0.512	0.996	0.977	0.989	0.978	0.985	0.980	0.979	0.974	0.968	0.976	0.977
Phoneme	**0.309**	0.288	0.085	0.149	0.284	0.151	0.233	0.110	0.299	0.175	0.207	0.252	0.190	0.165	0.298
RacketSports	0.954	0.933	0.809	0.849	0.928	0.803	0.855	0.803	0.877	0.868	0.776	0.816	0.190	**0.987**	0.882
SelfRegulationSCP1	**0.944**	0.930	0.925	0.922	0.908	0.775	0.812	0.874	0.835	0.652	0.799	0.823	0.934	0.816	0.782
SelfRegulationSCP2	0.628	0.619	0.589	0.604	0.533	0.539	0.578	0.472	0.532	0.550	0.550	0.533	**0.710**	0.622	0.578
SpokenArabicDigits	0.996	0.993	0.993	**0.998**	0.712	0.963	0.988	0.990	0.997	0.983	0.934	0.970	0.460	0.985	0.975
StandWalkJump	0.433	**0.567**	0.267	0.267	0.456	0.200	0.467	0.067	0.383	0.400	0.400	0.333	0.333	0.333	0.533
UWaveGestureLibrary	**0.962**	0.949	0.903	0.913	0.944	0.903	0.906	0.891	0.927	0.894	0.759	0.753	0.916	0.878	0.906
InsectWingBeat	**0.711**	0.701	0.105	0.158	0.168	0.105	0.466	0.167	0.667	0.208	0.469	0.264	0.163	0.137	0.250
Average Accuracy	**0.791**	0.786	0.658	0.660	0.698	0.628	0.698	0.621	0.704	0.657	0.670	0.668	0.683	0.748	0.699
Average Rank	**1.5**		7.4		5.8	9.2	7.2	9.0	5.7	7.0	9.4	9.0	5.5	4.9	5.7

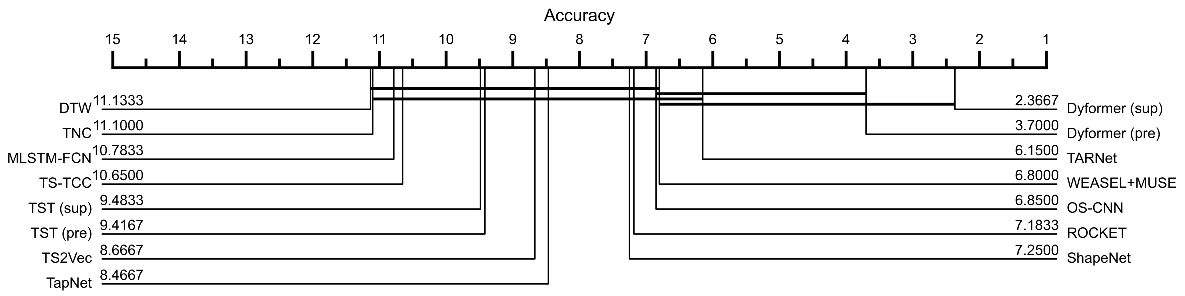


Figure 5.4: Critical Difference (CD) diagram of the selected baselines and Dyformer with a confidence level of 95%.

(e.g., EigenWorms and StandWalkJump). Since contrastive learning-based methods (TNC and TS-TCC) impose strong inductive bias, such as transformation-invariance, they may not fit various scenarios, resulting in low average ranks of 9.4 and 9.0, respectively.

Dyformer outperformed the other transformer-based models including TST and TARNet. Compared with TST, the proposed Dyformer achieves 11.8% and 11.4% improvement with the supervised training and unsupervised pre-training modes, respectively, in classification accuracy. While compared with TARNet, the improvement is 5.7% and 5.1%. We attribute the improvement to two-fold: 1) The capability of Dyformer's feature-map-wise attention for capturing multi-granular relationships—this is opposed to the content-based, dot-product attention, which falls less effective in detecting temporal dependencies [45]; 2) Incorporating time series decomposition and dynamic architecture enables adaptive learning strategies for different frequency components, realizing more effective feature extraction and harnessing than static models.

Our results of the Wilcoxon signed-rank test with a confidence level of 95% further confirm that Dyformer—either the supervised training version or the unsupervised pre-trained version achieved the best classification performance.

The training method (supervised or unsupervised) does not significantly affect Dyformer's performance.

5.3.5 Proof of Effectiveness of Hierarchical Pooling

We use Discrete Fourier Transform (DFT) [106], a commonly used method to explore the frequency components of the time series. The process can be described as follows:

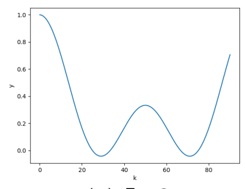
$$X(k) = \sum_{t=0}^{T-1} x(t) e^{-i2\pi kt/T} \tag{5.11}$$

where $x(t)$ is the time series sequence, and $t \in (0, T-1)$, T is the length of the time series. $X(k)$ is the frequency component obtained via DFT, and k is the index. Suppose $y(t)$ is the time series obtained after average pooling layer, L is the kernel size of the pooling layer, $\mathscr{F}$ is DFT, then $y(t)$ can be described as:

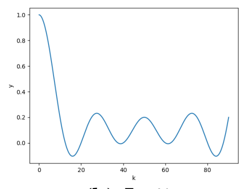
$$y(t) = \frac{1}{L} \sum_{l=0}^{L-1} x(t-l) \tag{5.12}$$

Then:

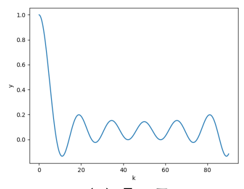
$$\begin{aligned} Y(k) = \mathscr{F} y(t) &= \frac{1}{L} \sum_{l=0}^{L-1} \mathscr{F} x(t-l) \\ &= \frac{1}{L} \sum_{l=0}^{L-1} X(k) e^{-i2\pi kl/T} \\ &= X(k) \frac{1}{L} \sum_{l=0}^{L-1} e^{-i2\pi kl/T} \\ &= X(k) \frac{\left(1 - e^{-\frac{i2\pi kL}{T}}\right)}{L\left(1 - e^{-\frac{i2\pi k}{T}}\right)} \end{aligned} \tag{5.13}$$



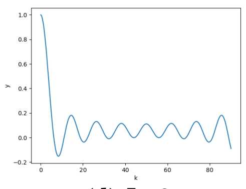
(a) L=3



(b) L=5



(c) L=7



(d) L=9

Figure 5.5: The curves of impact term (Eq. (5.14)) under kernel size of the pooling layer L=3, 5, 7, and 9. k is the frequency index, and y is the value of the impact term.

According to Eq. (5.13), from the frequency perspective, the influence of the average pooling layer is an impact term described as:

$$y = f(k, L) = \frac{\left(1 - e^{-\frac{i2\pi kL}{T}}\right)}{L\left(1 - e^{-\frac{i2\pi k}{T}}\right)} \tag{5.14}$$

To explore the effectiveness of the impact term (Eq. (5.14)), we set the kernel size of the pooling layer L 3, 5, 7, 9, separately and the corresponding curves are shown in Figure 5.5. The analysis shows that the impact factor tends to enhance the low-frequency component and reduce the high-frequency component. Additionally, the average pooling method more significantly penalizes high-frequency components with the increasing kernel size of the pooling layer.

5.3.6 An illustration of Hierarchical Pooling

We sample two sequences from **DuckDuckGeese** and **FaceDetection** to gain further insights regarding the effectiveness of the hierarchical pooling layer. The sampled sequences and their spectrograms are shown in Figure 5.6. We feed $Input_1$ into the hierarchical pooling layer and obtain $Output_1$, $Output_2$, $Input_3$, and $Input_4$. We use DFT to explore their frequency components. The illustrations of their time domain and frequency domain are shown in Figure 5.7 and Figure 5.8. Additionally, we provide the details of the top-5 frequency components of the original input and the decomposed subsequences in Table 5.3 and Table 5.4. Based on the results, we find that $Output_1$ contains the frequency components with high amplitudes, while the other outputs (i.e., $Output_2$ and $Output_3$ for both datasets) contain the frequency components that are different from $Output_1$. Because, after the first iteration, we subtract the $Output_1$ from the original time series $Input_1$, then the remaining $Input_2$ contains the different frequency components compared with $Output_1$. In this way, we can decompose the time

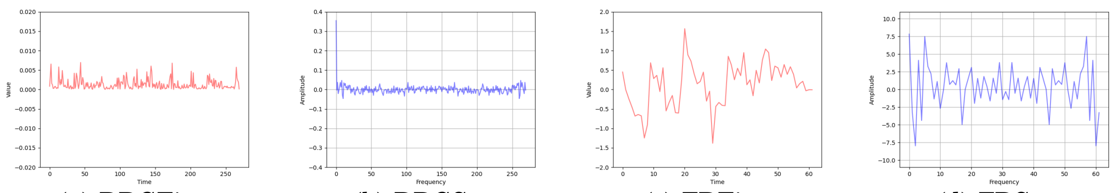

(a) DDGTime (b) DDGSpe (c) FDTime (d) FDSpe

Figure 5.6: Time domain and the spectrogram of the samples. DDGTime means the time domain of the sample from the DuckDuckGeese dataset, and DDGSpe means its spectrogram. And so on for the sample from the FaceDetection dataset.

Table 5.3: Top-5 Frequency Components of the original time series and the subsequences obtained after hierarchical pooling from the DuckDuckGeese dataset.

Input_1		Output_1		Output_2		Output_3		Input_4	
Frequency	Amplitude	Frequency	Amplitude	Frequency	Amplitude	Frequency	Amplitude	Frequency	Amplitude
0	0.35	0	0.35	142	0.012	142	0.009	102	0.055
8	0.047	8	0.045	128	0.012	128	0.009	168	0.054
262	0.047	262	0.045	134	0.007	208	0.007	135	0.046
268	0.037	268	0.036	136	0.007	62	0.007	124	0.039
2	0.036	2	0.036	111	0.007	159	0.006	146	0.039

Table 5.4: Top-5 Frequency Components of the original time series and the subsequences obtained after hierarchical pooling from the FaceDetection dataset.

Input_1		Output_1		Output_2		Output3		Input_4	
Frequency	Amplitude	Frequency	Amplitude	Frequency	Amplitude	Frequency	Amplitude	Frequency	Amplitude
0	7.83	0	7.68	57	1.79	5	0.81	12	6.56
57	7.49	57	1.13	5	1.79	57	0.81	50	6.56
5	7.49	5	4.13	6	0.75	10	0.48	29	5.03
59	4.14	59	3.30	56	0.75	52	0.48	33	5.03
3	4.14	3	3.30	59	0.53	56	0.46	46	3.81

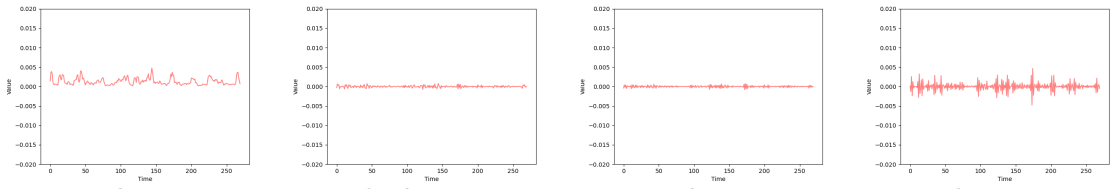
series sequence into a set of subsequences with different frequency components through the hierarchical pooling layer.

5.3.7 Proof of non-Information Loss of Hierarchical Pooling

According to the Eq. (5.2), we can find that:

$$\text{Input}_{1} = \sum_{n=1}^{N} \text{Output}_{n} + \text{Input}_{N+1} \tag{5.15}$$

where $Input_1$ is the original time series sequence, and $Output_n$ ($n \in (1,N)$) and $Input_{N+1}$ are the decomposed subsequences. Hence, from the time domain perspective, the hierarchical pooling layer does not lead to any information loss. From the frequency domain perspective, according to the Eq. (5.11), the DFT of the original time series sequence can


(a) Output$_1$ (b) Output$_2$ (c) Output$_3$ (d) Input$_4$

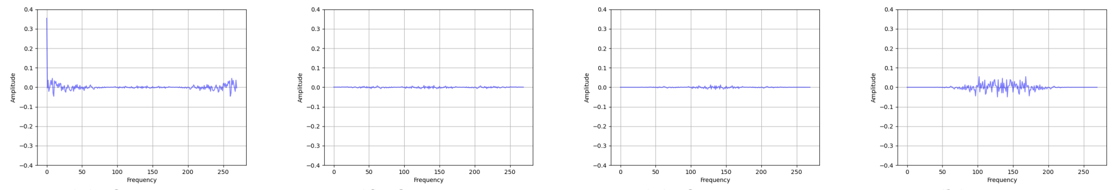
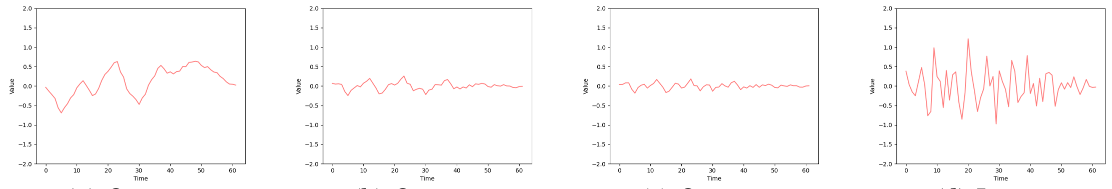

(e) Output$_2$ (f) Output$_3$ (g) Output$_3$ (h) Input$_4$

Figure 5.7: The generated subsequences (Output$_1$, Output$_2$, Output$_3$, and Input$_4$) of the hierarchical pooling layer from DuckDuckGeese dataset. The two axes refer to the time domain and frequency domain, respectively.


(a) Output$_1$ (b) Output$_2$ (c) Output$_3$ (d) Input$_4$

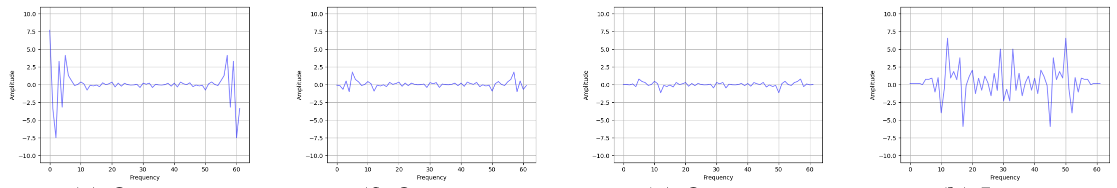

(e) Output$_2$ (f) Output$_3$ (g) Output$_3$ (h) Input$_4$

Figure 5.8: The generated subsequences (Output$_1$, Output$_2$, Output$_3$, and Input$_4$) of the hierarchical pooling layer from FaceDetection dataset. The two axes refer to the time domain and frequency domain, respectively.

be described as:

$$\text{Input}_{\,1}(k) = \sum_{t=0}^{T-1} \text{Input}_{\,1} e^{-\frac{i2\pi kt}{T}} \tag{5.16}$$

While the DFT of the decomposed time series sequences can be described as:

$$\begin{aligned}
&\sum_{t=0}^{T-1}\left(\sum_{n=1}^{N} \text{Output}_{\,n} e^{-\frac{i2\pi kt}{T}} + \text{Input}_{N+1} e^{-\frac{i2\pi kt}{T}}\right)\\
&= \sum_{t=0}^{T-1} e^{-\frac{i2\pi kt}{T}}\left(\sum_{n=1}^{N} \text{Output}_{\,n} + \text{Input}_{\,N+1}\right)\\
&= \sum_{t=0}^{T-1} \text{Input}_{\,1} e^{-\frac{i2\pi kt}{T}}
\end{aligned} \tag{5.17}$$

Hence, from the frequency domain perspective, the hierarchical pooling layer does not lead to any information loss.

5.3.8 Effectiveness of feature-map-wise Attention

To explore the effectiveness of the feature-map-wise attention, we use recurrent layers and convolutional layers to replace the feature-map-wise attention and explore the impact on the performance. Specifically, the process of using the convolutional layer to calculate the query, key, and value can be described as follows:

$$\begin{aligned}
Q &= W^Q * U_t + b^Q\\
K &= W^K * U_t + b^K\\
V &= W^V * U_t + b^V
\end{aligned} \tag{5.18}$$

where $*$ means convolution, and W^Q, W^K, W^V, b^Q, b^K, and b^V are learnable parameters.

When using a commonly used recurrent layer, the Gate Recurrent Unit (GRU) [24], as the calculation process of the query, key, and value matrices is similar, for simplicity, we only describe the calculation process of query matrix Q in the projection layer as:

$$\begin{aligned}
r_t &= \sigma\left(W^Q_{ir} U_t + b^Q_{ir} + W^Q_{hr} U_{(t-1)} + b^Q_{hr}\right)\\
z_t &= \sigma\left(W^Q_{iz} U_t + b^Q_{iz} + W^Q_{hz} h_{(t-1)} + b^Q_{hz}\right)\\
n_t &= \tanh\left(W^Q_{in} U_t + b^Q_{in} + r_t \circ \left(W^Q_{hn} h_{(t-1)} + b^Q_{hn}\right)\right)\\
h_t &= (1 - z_t) \circ n_t + z_t \circ h_{(t-1)}\\
Q &= Concat(h_1, h_2, \ldots, h_T)
\end{aligned} \tag{5.19}$$

where W^Q_{ir}, W^Q_{iz}, W^Q_{in}, W^Q_{hr}, W^Q_{hz}, W^Q_{hn}, b^Q_{ir}, b^Q_{hr}, b^Q_{iz}, b^Q_{hz}, b^Q_{in}, and b^Q_{hn} are learnable parameters. We select **ArticularyWordRecognition** and **MotorImagery** for experiments as they are diverse enough in variable number (ArticularyWordRecognition 9, MotorImagery 64) and length (ArticularyWordRecognition 144, MotorImagery 3, 000). For the

Table 5.5: Training and test results under varying attention manner. Convolution and recurrent means using the convolutional and recurrent layers to calculate the query, key, and value matrices, respectively.

Dataset	Attention Manner	Train				Test			
		Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
ArticularyWordRecognition	Convolution	0.5654	0.7055	0.6166	0.6581	0.5413	0.6167	0.5512	0.5821
	Recurrent	0.8492	0.9055	0.8726	0.8887	0.7548	0.7900	0.7638	0.7767
	Feature-map-wise	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**0.9970**	**0.9960**	**0.9980**	**0.9970**
MotorImagery	Convolution	0.7206	0.7194	0.7191	0.7192	0.5420	0.5400	0.5387	0.5393
	Recurrent	0.7099	0.7086	0.7084	0.7085	0.5384	0.5100	0.3988	0.4476
	Feature-map-wise	**0.7306**	**0.7294**	**0.7291**	**0.7292**	**0.6400**	**0.6100**	**0.6300**	**0.6198**

Table 5.6: Training and test results under varying group numbers.

Dataset	#Groups	Train				Test			
		Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
DuckDuckGeese	4	0.762	0.722	0.686	0.704	0.654	0.627	0.592	0.609
	8	0.803	0.790	0.766	0.778	0.722	0.665	0.621	0.642
	16	0.868	0.825	0.813	0.819	0.761	0.717	0.694	0.705
	32	0.907	0.893	0.855	0.873	**0.818**	**0.801**	**0.806**	**0.804**
	64	**0.942**	**0.929**	**0.904**	**0.916**	0.743	0.719	0.766	0.742
HeartBeat	4	0.862	0.871	0.856	0.864	0.699	0.669	0.654	0.661
	8	0.927	0.888	0.879	0.884	0.717	0.661	0.658	0.659
	16	0.951	0.929	0.912	0.920	0.734	0.678	0.669	0.674
	32	0.965	0.936	0.914	0.925	0.765	0.697	0.706	0.702
	64	**0.971**	**0.942**	**0.921**	**0.932**	**0.786**	**0.704**	**0.714**	**0.709**

convolutional layer, we stack two layers with kernel size 3, and we use padding to ensure that the length of the input and output remains consistent; for GRU, we stack two bi-directional GRU layers. We follow the same configurations in Table 5.1 for all the other settings. According to the experimental results in Table 5.5, feature-map-wise attention realizes the best performance on both datasets. The convolutional and recurrent layers can only capture short-term dependencies, ignoring long-term dependencies. Our feature-map-wise attention can effectively leverage long- and short-term dependencies through progressively enlarged receptive fields to realize multi-scale attention, resulting in better performance.

5.3.9 Impact of Group Number

As mentioned, Dyformer uses feature-map-wise attention by hierarchical-connected group convolution to leverage the inter-relationships of various granular feature maps. In this experiment, we explore the impact of the group number (s in Eq. (5.6)) while keeping all the other settings unchanged on two datasets: **DuckDuckGeese** and **HeartBeat**. These datasets contain long time series sequences (over 200 time steps) that allow us to explore the impact of group numbers in a larger range.

Table 5.7: Training and test results under varying γ values.

Dataset	γ	Train				Test			
		Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
LSST	0	**0.897**	**0.877**	**0.892**	**0.884**	0.617	0.623	0.605	0.614
	0.1	0.882	0.868	0.863	0.865	0.634	0.635	0.632	0.633
	0.2	0.853	0.843	0.828	0.835	**0.758**	**0.721**	**0.749**	**0.735**
	0.3	0.824	0.814	0.823	0.818	0.667	0.655	0.630	0.642
	0.4	0.765	0.740	0.759	0.749	0.637	0.629	0.631	0.630
FingerMovements	0	**0.927**	**0.934**	**0.924**	**0.929**	0.540	0.550	0.530	0.540
	0.1	0.908	0.911	0.905	0.908	0.580	0.570	0.590	0.580
	0.2	0.896	0.889	0.902	0.895	**0.650**	**0.620**	**0.650**	**0.635**
	0.3	0.864	0.858	0.861	0.859	0.590	0.580	0.570	0.575
	0.4	0.797	0.791	0.801	0.796	0.530	0.540	0.520	0.530

Intuitively, more groups of feature maps result in convolutional filters with more sizes of receptive fields. However, constructing more groups brings the risk of overfitting. Our results (Table 5.6) show that Dyformer became overfitted when the group number increased to 64, indicated by a performance drop from s=32 to s=64. Since HeartBeat has longer sequences than DuckDuckGeese, no overfitting occurred when s went as large as 64.

5.3.10 Impact of Penalty Weight

We selected two datasets, **LSST** and **FingerMovements**, as examples to explore the impact of the hyperparameter γ, which impacts how likely the gates before the Dyformer blocks will open.

Results (Table 5.7) show that our model tends to underfit as γ increases. Intuitively, γ can be regarded as the block-wise dropout, i.e., a large γ means the Dyformer block is more likely to be skipped. Hence, we can follow the same fine-tuning rule as the dropout rate. To summarize, we tend to use small γ on large datasets to improve the training performance and vice versa. Nevertheless, overfitting should be considered when increasing γ. In particular, when γ is 0, the gate penalization term is omitted from the loss function.

5.3.11 Impact of Training Method

Table 5.2 shows the comparison between the Dyformer trained in a supervised manner and that trained in the (unsupervised) pre-training manner. Dyformer with supervised training outperforms the pre-trained version, as demonstrated by the higher average accuracy.

Table 5.8: Training and test results using different feedforward layers.

Dataset	Feedforward Layer	Train				Test			
		Accuracy	Recall	Precision	F1-Score	Accuracy	Recall	Precision	F1-Score
StandWalkJump	Point-wise feedforward	0.599	0.586	0.601	0.593	0.397	0.388	0.382	0.385
	Series-wise feedforward	**0.612**	**0.637**	**0.629**	**0.633**	**0.433**	**0.428**	**0.439**	**0.433**
HandWriting	Point-wise feedforward	0.677	0.682	0.669	0.675	0.547	0.552	0.541	0.546
	Series-wise feedforward	**0.698**	**0.719**	**0.681**	**0.699**	**0.593**	**0.582**	**0.601**	**0.591**
DuckDuckGeese	Point-wise feedforward	0.854	0.837	0.844	0.840	0.769	0.765	0.774	0.769
	Series-wise feedforward	**0.887**	**0.893**	**0.865**	**0.879**	**0.788**	**0.767**	**0.806**	**0.786**

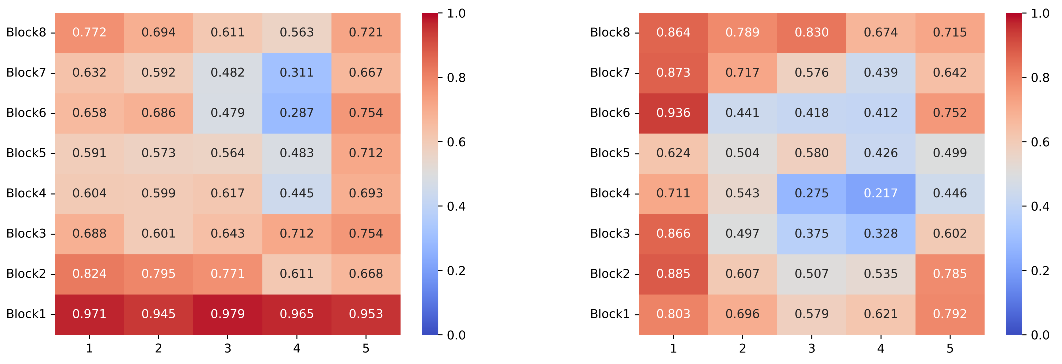
Unsupervised training is generally better on datasets containing limited samples, such as **AtrialFibrillation**, **BasicMotions**, and **StandWalkJump**. That indicates the unsupervised training method may better cope with limited samples, thus enhancing the classification performance.

5.3.12 Impact of the Series-wise Feedforward.

Dyformer introduces a novel approach by replacing the conventional point-wise feedforward layer, commonly employed in transformer-based architectures, with the series-wise feedforward layer. This innovation is particularly well-suited for handling time series data due to its inherent temporal dependencies. In this experiment, we investigate the impact of adopting the series-wise feedforward layer while keeping all other settings unchanged. We evaluate its performance on three diverse datasets: **DuckDuckGeese**, **HandWriting**, and **StandWalkJump**. These datasets vary in terms of sequence length and the number of variables, providing a comprehensive result. The experimental results presented in Table 5.8 consistently demonstrate that the series-wise feedforward layer outperforms the point-wise attention in both the training and test phases, demonstrating its effectiveness in enhancing model performance.

5.3.13 Layer Activation Frequency Study

We further explored the activation status of the Dyformer blocks in different paths. We selected **EigenWorms** and **MotorImagery** as they have the longest and second-longest time series sequences (17,984 and 3,000, respectively) among all the datasets. Hence, we decomposed the time series, obtained the maximum number of subsequences, and constructed the most Dyformer modules among all the datasets. We believe exploring the most complicated architecture demonstrates more comprehensive results than other datasets. Following the configuration in Table 5.1, we tested all the datasets' samples and recorded the number of activation times of all the Dyformer blocks. Then, we calculated



(a) EigenWorms (b) MotorImagery

Figure 5.9: Heatmap for Activation Frequency of all Dyformer Blocks on EigenWorms and MotorImagery. 1, 2, 3, 4, and 5 mean the paths for the first through fifth subsequences.

Table 5.9: Ablation test for Dyformer. GC means group convolution, and HP means hierarchical pooling.

Dataset	Model	Accuracy	Precision	Recall	F1-Score
DuckDuckGeese	Dyformer (w/o GC, HP)	0.585	0.569	0.588	0.578
	Dyformer (w/o GC)	0.643	0.629	0.637	0.632
	Dyformer (w/o HP)	0.701	0.665	0.696	0.680
	Dyformer	**0.818**	**0.775**	**0.791**	**0.783**
FaceDetection	Dyformer (w/o GC, HP)	0.494	0.521	0.481	0.500
	Dyformer (w/o GC)	0.561	0.574	0.546	0.560
	Dyformer (w/o HP)	0.612	0.607	0.615	0.611
	Dyformer	**0.737**	**0.709**	**0.715**	**0.712**
PEMS-SF	Dyformer (w/o GC, HP)	0.644	0.621	0.613	0.617
	Dyformer (w/o GC)	0.673	0.647	0.639	0.643
	Dyformer (w/o HP)	0.775	0.763	0.754	0.758
	Dyformer	**0.935**	**0.893**	**0.929**	**0.911**

the average number of activation times as the activation frequency of all the Dyformer blocks.

Our results (Figure 5.9) show that Dyformer tends to activate the first and last Dyformer blocks more than the middle ones, and *amplitude* positively impacts the Dyformer block's activation frequency.

Examples are subsequence$_1$ and subsequence$_5$—their frequency components have larger amplitudes; meanwhile, more Dyformer blocks are activated in the paths for them.

5.3.14 Ablation Study

We conducted ablation studies on three datasets including **DuckDuckGeese**, **FaceDetection**, and **PEMS-SF**, to explore the effectiveness of group convolution and hierarchical pooling. We separately incorporate group convolution and hierarchical pooling, and both in Dyformer. Specifically, without group convolution, we feed the input time series sequences to standard convolutional layers to obtain series-wise queries, keys, and values and calculate standard multi-head attention. Without hierarchical pooling, we feed the original time series to the model instead of pre-decomposition. We trained Dyformer in a supervised manner.

The results (Table 5.9) suggest that each component improves the performance, while group convolution impacts more significantly on all three datasets' performance than hierarchical pooling. Hence, feature-map-wise attention is likely to play a more critical role in enhancing the classification ability of the model.

5.4 Conclusion and Future Work

This work proposes a novel dynamic transformer-based architecture called Dyformer for accurate multivariate time series classification. We first design a hierarchical pooling layer for time series decomposition. We further propose a Dyformer Module to achieve adaptive learning strategies for different frequency patterns and incorporate feature-map-wise attention for capturing multi-scale temporal dependencies. We conduct a comprehensive evaluation of our model on a wide range of datasets comprising sequences of varying lengths and variable numbers. Our experimental results show Dyformer outperforms multiple baselines and state-of-the-art methods by a large margin. We thoroughly investigate the influence of various components and configurations on the model's performance. We also demonstrate that through unsupervised pre-training, Dyformer gains an improvement over fully supervised learning on datasets with fewer samples.

We have identified several weaknesses in the proposed Dyformer framework. Firstly, the architecture's reliance on parallel construction of multi-paths results in a time consumption profile that is heavily contingent upon the training speed of the path that activates the maximum number of Dyformer blocks. Consequently, the activation of fewer Dyformer blocks along certain paths may not yield a significant acceleration in the overall training speed. This trade-off between training speed and performance enhancement is a notable limitation of Dyformer, as it prioritizes the latter at the expense of the former. In the face of real-world challenges, it becomes imperative to carefully consider the trade-off between accuracy and efficiency when selecting appropriate methodologies. Additionally, a noteworthy deficiency in the proposed Dyformer framework lies in its lack of interpretability. This absence of interpretability can restrict the applicability of the method in specific scenarios, particularly in fields such as medical care, where transparency and interpretability are of paramount importance. In the future, we aim to address these identified shortcomings to enhance the applicability and utility of the proposed Dyformer framework.

CHAPTER 6

An Empirical Study on Positional Embedding for Transformer

6.1 Introduction

In recent years, Transformer-based methods have made remarkable breakthroughs in time series-related tasks such as classification [22, 62, 87, 146] and forecasting [40, 82, 142, 157]. Since Transformer is position-insensitive, positional embedding [113] was introduced to allow the model to learn the relative position of tokens. Positional embedding generally injects position information into sequence data. It takes the form of sinusoidal functions of different frequencies, with each embedding dimension corresponding to a sinusoid whose wavelengths form a geometric progression. To date, positional embedding has been a routine for Transformer-based models [84, 87, 129] that deal with time series sequences.

Despite the widespread use, there have been some debates [45, 150] around the necessity of positional embedding, and a comprehensive investigation of positional embedding's effectiveness on various Transformer-based models is still to be developed. Firstly, Transformer-based models [84, 157] that contain position-sensitive modules (e.g., convolutional and recurrent layers) can automatically learn the potential position information, making positional embedding redundant to some extent. This point is supported by studies [94, 128] in other fields, suggesting positional embedding may be unnecessary and replaced with position-sensitive layers. Secondly, positional embedding has inherent limitations that may potentially impair the model's performance. Since positional embedding is hand-crafted, it may bring inductive bias that may adversely impact the model's performance in some cases. Such as for classification, positional embedding injects the same position tokens into the time series of different classes, which poses additional challenges to the classifier in figuring out the differences between sequences with different class labels.

In this work, we review the existing Transformer-based variants containing position-sensitive layers and summarize six kinds of Transformer-based variants. We then conduct comprehensive experiments on thirty time series classification datasets and four time series forecasting datasets under three different input sequence lengths to explore the impact of positional embedding on the variants and the vanilla Transformer. Our results show that positional embedding positively impacts the vanilla Transformer in both classification and forecasting, but it negatively influences the performance of the variants in classification and shows different impacts on the variants' performance under different input sequence lengths in forecasting.

To gain further insight into the effectiveness of positional embedding, we reverse the sequential order of the time series and investigate the impact of spurious position information caused by it. We find that the spurious position information drastically degrades the performance of the variants and significantly impacts the attention distribution, but it has an insignificant effect on the vanilla Transformer.

Through this work, we hope to explore the scenarios in which positional embedding positively impacts the model's performance to facilitate researchers and practitioners in making informed decisions on whether to incorporate positional embedding in their models for time series analysis.

6.2 Background

In this section, we review the positional embedding techniques and various Transformer-based variants.

6.2.1 Positional Embedding

Positional embedding was first proposed for Transformer in [113], which uses fixed sine and cosine functions of different frequencies to represent the position information, as

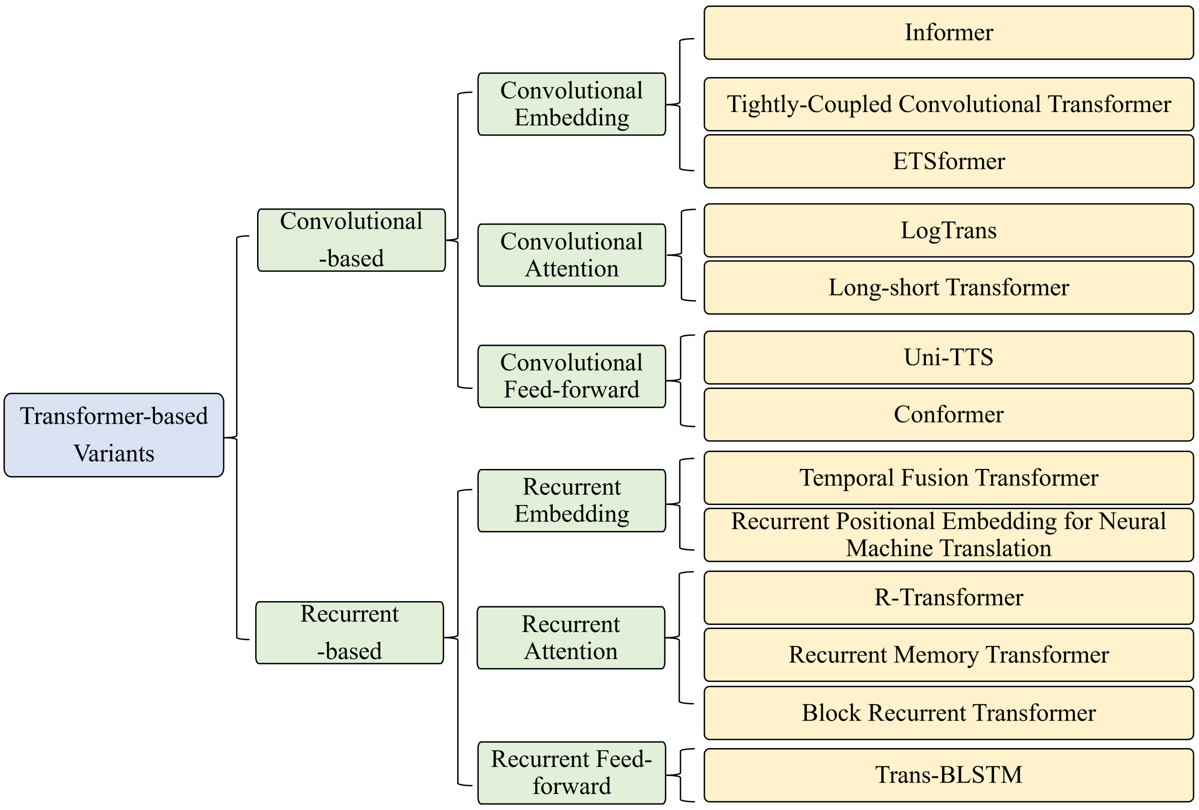

Transformer-based Variants
Convolutional -based
Recurrent -based
Convolutional Embedding
Convolutional Attention
Convolutional Feed-forward
Recurrent Embedding
Recurrent Attention
Recurrent Feed-forward
Informer
Tightly-Coupled Convolutional Transformer
ETSformer
LogTrans
Long-short Transformer
Uni-TTS
Conformer
Temporal Fusion Transformer
Recurrent Positional Embedding for Neural Machine Translation
R-Transformer
Recurrent Memory Transformer
Block Recurrent Transformer
Trans-BLSTM

Figure 6.1: A summary of six types of Transformer-based variants for modeling sequential data.

described below:

$$
\begin{aligned}
PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\
PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right)
\end{aligned}
\tag{6.1}
$$

where pos and i are the position and the dimension indices, respectively, and d_{model} is the dimensionality of the input time series. That is, each dimension of the positional embedding corresponds to a sinusoid. Since for any fixed offset k, $PE_{\text{pos}+k}$ can be represented as a linear function of PE_{pos}, allowing the model to learn the relative positions easily. The positional embedding is then added to the input time series as the input of the Transformer.

Considering hand-crafted positional embedding is generally less expressive and adaptive [124], Time Series Transformer (TST) [151] enhances the vanilla Transformer [113] by implementing learnable positional embedding. Specifically, TST shares the same architecture as the vanilla Transformer, which stacks several basic blocks, each consisting of scaled dot-product multi-head attention and a feed-forward network (FFN) to leverage temporal data. But it differs in initializing the positional embedding using fixed values and then updating the embedding jointly with other model parameters through the training procedure.

Informer [157] introduces a variant of positional embedding known as time stamp embedding, which encodes both hierarchical time stamps (such as week, month, and year) and agnostic time stamps (such as holidays and events) to generate global information. This approach has been shown to mitigate performance degradation when dealing with long time series sequences. Time stamp embedding is commonly used in time series forecasting, but is less frequently used in classification tasks because the time series sequences from different classification datasets are sampled based on various sampling rates, bringing challenges to present a uniform time stamp embedding.

6.2.2 Transformer-based Variants

Current studies often incorporate convolutional or recurrent layers in the vanilla Transformer architecture in dealing with sequence-related tasks, including time series analysis. Figure 6.1 summarizes these Transformer-based variants into six categories of representative methods, as detailed below.

- **Convolutional Embedding**: Methods in this category, namely Informer [157], Tightly-Coupled Convolutional Transformer (TCCT) [103], and ETSformer [45], implement a convolutional layer to obtain convolutional embeddings, which map the raw input sequences to a latent space before feeding them to the transformer block.

- **Convolutional Attention**: Instead of calculating the point-wise attention, LogTrans [82] and Long-short Transformer [160] use the convolutional layer to calculate the attention matrix (including queries, keys, and values) of segments to leverage the local temporal information.

- **Convolutional Feed-forward**: Uni-TTS [89] and Conformer [49] implement a convolutional layer after the multi-head attention as the feed-forward layer (or part of the feed-forward layer) to capture local temporal correlations.

- **Recurrent Embedding**: Temporal Fusion Transformer (TFT) [84] and the work in [20] use a recurrent layer to encode content-based order dependencies into the input sequence.

Table 6.1: Structure of the basic model and six variants (ConvEmbedding means Convolutional Embedding Variant, RecEmbedding means Recurrent Embedding Variant, and so on).

Model	Input Embedding	Projection	Feed-forward
Basic Model	Linear	Linear	Linear
ConvEmbedding	Convolutional Layer	Linear	Linear
ConvAttention	Linear	Convolutional Layer	Linear
ConvFFD	Linear	Linear	Convolutional Layer
RecEmbedding	Gated Recurrent Unit	Linear	Linear
RecAttention	Linear	Gated Recurrent Unit	Linear
RecFFD	Linear	Linear	Gated Recurrent Unit

- **Recurrent Attention**: Recurrent Memory Transformer [13], Block Recurrent Transformer [65], and R-Transformer [121] use a recurrent neural net to calculate the attention matrix, which harnesses the temporal information more effectively when compared with the point-wise attention.

- **Recurrent Feed-forward**: Instead of point-wise feed-forward, TRANS-BLSTM [63] uses a recurrent layer after multi-head attention to harness the non-linear temporal dependencies.

6.3 Methodology

We adopt the architecture of TST [151] as the **basic model**. Following the idea of the existing methods, we design six Transformer-based variants: Convolutional Embedding, Convolutional Attention, Convolutional Feed-forward, Recurrent Embedding, Recurrent Attention, and Recurrent Feed-forward. The modifiable components of the Transformer architecture (shown in Figure 6.2) include the input embedding layer, which projects the input time series into the latent space, the projection layer, which calculates the attention matrix, and the feed-forward layer, which leverages non-linear relationships. For each variant, we use different layers in each component, while details are given in Table 6.1. We provide more details in the following sections.

6.3.1 Basic Model

The **basic model** adopts linear layers in all three components. In this case, for each sample $\mathbf{x_t} \in \mathbb{R}^M : \mathbf{X} \in \mathbb{R}^{M\times T} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x_T}]$, where T is the sequence length and M is the

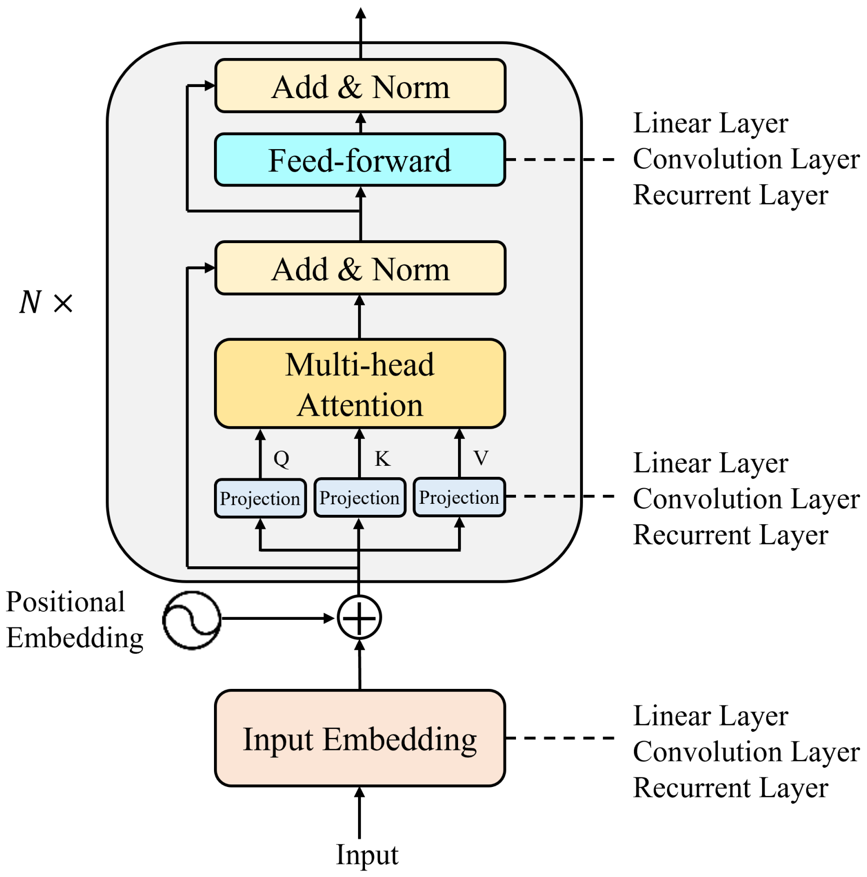


Figure 6.2: Architecture of the transformer-based variants for modeling sequential data

variable number. The input embedding can be described as:

$$U_t = W^x x_t + b^x \tag{6.2}$$

where $t = 0,1,...,T$ is the time stamp index, $W^x \in \mathbb{R}^{M \times d_k}$ and $b^x \in \mathbb{R}^{d_k}$ are learnable parameters. The projection layer can be described as:

$$\begin{aligned} Q &= W^Q U_t + b^Q \\ K &= W^K U_t + b^K \\ V &= W^V U_t + b^V \end{aligned} \tag{6.3}$$

where $W^Q \in \mathbb{R}^{d_k \times d_k}$, $W^K \in \mathbb{R}^{d_k \times d_k}$, $W^V \in \mathbb{R}^{d_k \times d_k}$, $b^Q \in \mathbb{R}^{d_k}$, $b^K \in \mathbb{R}^{d_k}$, and $b^V \in \mathbb{R}^{d_k}$ are are learnable parameters. We use standard scaled Dot-Product attention proposed in the vanilla Transformer [113] for self-attention calculation:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{6.4}$$

The feed-forward layer can be described as:

$$FFN(x) = \text{ReLU}(W_1 x + b_1) W_2 + b_2 \tag{6.5}$$

where leanable parameters are: $W_1 \in \mathbb{R}^{d_k \times d_k}$, $W_2 \in \mathbb{R}^{d_k \times d_k}$, $b_1 \in \mathbb{R}^{d_k}$, and $b_2 \in \mathbb{R}^{d_k}$.

6.3.2 Convolutional-based Variants

We refer to the architectures that employ convolutional layers in any of the three components (input embedding layer, projection layer, or feed-forward layer) as convolutional-based variants. Here, we utilize a one-dimensional convolutional layer with a kernel size of 3. We also set the padding to 1 to preserve the lengths of representations. In the following, we illustrate our convolutional-based variants one by one.

Convolutional Embedding Variant replaces the linear layer with the convolution layer in the input embedding layer, which is formulated below:

$$U_t = W^x * x_t + b^x \tag{6.6}$$

where $*$ is the convolutional operation, $W^x \in \mathbb{R}^{M \times d_k \times P}$ and $b^x \in \mathbb{R}^M$ are learnable parameters, and P is the kernel size.

Convolutional Attention Variant replaces the linear layer with the convolution layer in the projection layer, which is formulated below:

$$\begin{aligned} Q &= W^Q * U_t + b^Q \\ K &= W^K * U_t + b^K \\ V &= W^V * U_t + b^V \end{aligned} \tag{6.7}$$

where $W^Q \in \mathbb{R}^{d_k \times d_k \times P}$, $W^K \in \mathbb{R}^{d_k \times d_k \times P}$, $W^V \in \mathbb{R}^{d_k \times d_k \times P}$, $b^Q \in \mathbb{R}^{d_k}$, $b^K \in \mathbb{R}^{d_k}$, and $b^V \in \mathbb{R}^{d_k}$ are learnable parameters.

Convolutional Feed-forward Variant formulated the linear layer with the convolution layer in the feed-forward layer, which is described below:

$$FFN(x) = \text{ReLU}(W_1 * x + b_1) * W_2 + b_2 \tag{6.8}$$

where leanable parameters are: $W_1 \in \mathbb{R}^{d_k \times d_k \times P}$, $W_2 \in \mathbb{R}^{d_k \times d_k \times P}$, $b_1 \in \mathbb{R}^{d_k}$, and $b_2 \in \mathbb{R}^{d_k}$.

6.3.3 Recurrent-based Variants

We name the architectures that use recurrent layers in any of the three components (input embedding layer, projection layer, or feed-forward layer) as recurrent-based variants. Here, we use Gate Recurrent Unit (GRU) [24] as the recurrent layer. In the following, we illustrate our recurrent-based Variants one by one.

Recurrent Embedding Variant replaces the linear layer with the GRU in the input embedding layer, which is formulated below:

$$\begin{aligned}
r_t &= \sigma\left(W_{ir}^{x} x_t + b_{ir}^{x} + W_{hr}^{x} U_{(t-1)} + b_{hr}^{x}\right)\\
z_t &= \sigma\left(W_{iz}^{x} x_t + b_{iz}^{x} + W_{hz}^{x} h_{(t-1)} + b_{hz}^{x}\right)\\
n_t &= \tanh\left(W_{in}^{x} x_t + b_{in}^{x} + r_t \circ \left(W_{hn}^{x} h_{(t-1)} + b_{hn}^{x}\right)\right)\\
h_t &= (1 - z_t) \circ n_t + z_t \circ h_{(t-1)}\\
U_t &= Concat(h_1, h_2, \ldots, h_T)
\end{aligned} \tag{6.9}$$

where $W_{ir}^{x} \in \mathbb{R}^{M \times d_k}$, $W_{iz}^{x} \in \mathbb{R}^{M \times d_k}$, $W_{in}^{x} \in \mathbb{R}^{M \times d_k}$, $W_{hr}^{x} \in \mathbb{R}^{d_k \times d_k}$, $W_{hz}^{x} \in \mathbb{R}^{d_k \times d_k}$, $W_{hn}^{x} \in \mathbb{R}^{d_k \times d_k}$, $b_{ir}^{x} \in \mathbb{R}^{d_k}$, $b_{hr}^{x} \in \mathbb{R}^{d_k}$, $b_{iz}^{x} \in \mathbb{R}^{d_k}$, $b_{hz}^{x} \in \mathbb{R}^{d_k}$, $b_{in}^{x} \in \mathbb{R}^{d_k}$, and $b_{hn}^{x} \in \mathbb{R}^{d_k}$ are learnable parameters, $\circ$ is the Hadamard product.

Recurrent Attention Variant replaces the linear layer with the GRU in the projection layer. Since the calculation processes of all the matrices are similar, for simplicity, we only present the calculation process of the query matrix Q in the projection layer below:

$$\begin{aligned}
r_t &= \sigma\left(W_{ir}^{Q} U_t + b_{ir}^{Q} + W_{hr}^{Q} U_{(t-1)} + b_{hr}^{Q}\right)\\
z_t &= \sigma\left(W_{iz}^{Q} U_t + b_{iz}^{Q} + W_{hz}^{Q} h_{(t-1)} + b_{hz}^{Q}\right)\\
n_t &= \tanh\left(W_{in}^{Q} U_t + b_{in}^{Q} + r_t \circ \left(W_{hn}^{Q} h_{(t-1)} + b_{hn}^{Q}\right)\right)\\
h_t &= (1 - z_t) \circ n_t + z_t \circ h_{(t-1)}\\
Q &= Concat(h_1, h_2, \ldots, h_T)
\end{aligned} \tag{6.10}$$

where $W_{ir}^{Q} \in \mathbb{R}^{d_k \times d_k}$, $W_{iz}^{Q} \in \mathbb{R}^{d_k \times d_k}$, $W_{in}^{Q} \in \mathbb{R}^{d_k \times d_k}$, $W_{hr}^{Q} \in \mathbb{R}^{d_k \times d_k}$, $W_{hz}^{Q} \in \mathbb{R}^{d_k \times d_k}$, $W_{hn}^{Q} \in \mathbb{R}^{d_k \times d_k}$, $b_{ir}^{Q} \in \mathbb{R}^{d_k}$, $b_{hr}^{Q} \in \mathbb{R}^{d_k}$, $b_{iz}^{Q} \in \mathbb{R}^{d_k}$, $b_{hz}^{Q} \in \mathbb{R}^{d_k}$, $b_{in}^{Q} \in \mathbb{R}^{d_k}$, and $b_{hn}^{Q} \in \mathbb{R}^{d_k}$ are learnable parameters.

Recurrent Feed-forward Variant replaces the linear layer with the GRU in the feed-forward layer, which is formulated below:

$$
\begin{aligned}
r_t &= \sigma\left(W_{ir}U_t + b_{ir} + W_{hr}U_{(t-1)} + b_{hr}\right) \\
z_t &= \sigma\left(W_{iz}U_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}\right) \\
n_t &= \tanh\left(W_{in}U_t + b_{in} + r_t \circ \left(W_{hn}h_{(t-1)} + b_{hn}\right)\right) \\
h_t &= (1 - z_t) \circ n_t + z_t \circ h_{(t-1)} \\
O &= Concat(h_1, h_2, ..., h_T)
\end{aligned}
\tag{6.11}
$$

where $W_{ir} \in \mathbb{R}^{d_k \times d_k}$, $W_{iz} \in \mathbb{R}^{d_k \times d_k}$, $W_{in} \in \mathbb{R}^{d_k \times d_k}$, $W_{hr} \in \mathbb{R}^{d_k \times d_k}$, $W_{hz} \in \mathbb{R}^{d_k \times d_k}$, $W_{hn} \in \mathbb{R}^{d_k \times d_k}$, $b_{ir} \in \mathbb{R}^{d_k}$, $b_{hr} \in \mathbb{R}^{d_k}$, $b_{iz} \in \mathbb{R}^{d_k}$, $b_{hz} \in \mathbb{R}^{d_k}$, $b_{in} \in \mathbb{R}^{d_k}$, and $b_{hn} \in \mathbb{R}^{d_k}$ are learnable parameters, and O is the final output of the feed-forward layer.

6.4 Experiments

We investigate the impact of positional embedding on all methods for both time series classification and forecasting. We choose the following datasets for comprehensive experimental results.

6.4.1 Datasets

We empirically evaluate the impact of positional embedding on the performance of the basic model and transformer-based variants (illustrated in Section 6.3) for time series classification and forecasting. We report our experimental configurations and discuss the results in the following subsections.

- **Classification**: We selected 30 public multivariate time series datasets from the UEA Time Series Classification Repository [29]. All datasets were pre-split into training and test sets[1] . We normalized all datasets to zero mean and unit standard deviation and applied zero padding to ensure all the sequences in each dataset bear the same length.
- **Forecasting**: We choose the Electricity Transformer Temperature (ETT) dataset [157] and create separate datasets as {ETTh1, ETTh2} for 1-hour level and {ETTm1, ETTm2} for 15-minutes-level to explore the impact of positional embedding at different granularities. Each data point consists of the target value "temperature" and six power load features. The train/val/test is 12/4/4 months. The input of each

[1]details can be found at `http://www.timeseriesclassification.com/dataset.php`

Table 6.2: Model configuration.

Dataset	Learning Rate	#layer	Batch Size	Dropout	#attention head
ArticularyWordRecognition	0.01	3	32	0.01	2
AtrialFibrillation	0.01	2	16	0.01	2
BasicMotions	0.00001	2	16	0.01	2
CharacterTrajectories	0.01	2	16	0.01	2
Cricket	0.01	2	16	0.01	2
DuckDuckGeese	0.001	4	8	0.3	5
EigenWorms	0.01	1	1	0.01	2
Epilepsy	0.00001	4	16	0.01	2
EthanolConcentration	0.001	2	16	0.01	4
ERing	0.00001	2	16	0.01	2
FaceDetection	0.00001	2	16	0.01	2
FingerMovements	0.001	2	16	0.01	2
HandMovementDirection	0.01	2	16	0.1	2
Handwriting	0.01	5	16	0.01	2
Heartbeat	0.00001	2	16	0.01	2
JapaneseVowels	0.01	3	16	0.3	2
Libras	0.01	5	16	0.01	2
LSST	0.01	2	16	0.01	2
MotorImagery	0.00001	2	16	0.01	2
NATOPS	0.00001	3	16	0.01	2
PenDigits	0.001	2	16	0.01	2
PEMS-SF	0.00001	2	16	0.01	2
Phoneme	0.00001	3	16	0.01	2
RacketSports	0.00001	2	16	0.1	4
SelfRegulationSCP1	0.00001	3	16	0.1	2
SelfRegulationSCP2	0.00001	2	16	0.01	2
SpokenArabicDigits	0.00001	3	16	0.1	2
StandWalkJump	0.01	3	16	0.01	2
UWaveGestureLibrary	0.01	2	16	0.01	2
ETTh1	0.01	2	4	0.05	2
ETTh2	0.01	2	4	0.05	2
ETTm1	0.01	2	4	0.05	2
ETTm2	0.01	2	4	0.05	2

dataset is zero-mean normalized. We set three different input sequence lengths: 96 (4 days for ETTh and 24 hours for ETTm), 192, and 336, and set the prediction window size to 48 (2 days for ETTh and 12 hours for ETTm).

6.4.2 Model Configuration and Evaluation Metric

We trained the basic model and six variants for 500 epochs using the Adam optimizer [75] on all the datasets with and without the learnable positional embedding. For classification, we selected the learnable positional embedding. Additionally, we incorporated the time stamp embedding for forecasting.

Besides, we applied an adaptive learning rate, which was reduced by a factor of 10 after every 100 epochs, and employed dropout regularization to prevent overfitting. Table 6.2 summarizes our model configurations for each dataset.

For classification, we evaluate the models using two metrics: *accuracy* and macro *F1-Score*, while for forecasting, we use *mean square error* (MSE) and *mean absolute error* (MAE). All metrics are commonly used for evaluating the model's performance. To mitigate the effect of randomized parameter initialization, we repeated the training and test procedures five times and took the average as the final results.

6.5 Results and Analysis

6.5.1 Classification

Table 6.3 and Table 6.4 show the evaluation results of all methods on 30 datasets. The results indicate positional embedding positively impacts the basic model—with positional embedding, the basic model's performance improves by 17.5% and 14.3% in accuracy and macro F1-Score, respectively. This reveals the significance of enabling the basic model to leverage the position information (e.g., via positional embedding) in solving the multivariate time series classification problem.

In contrast, positional embedding negatively impacts the performance of the Transformer-based variants. Without positional embedding, convolutional embedding (i.e., ConvEmbedding in Table 6.1) and recurrent embedding (i.e., RecEmbedding in Table 6.1) models outperformed all other variants, achieving the best accuracy of 56.21% and 56.17%, respectively, and the best macro F1-Scores of 0.528 and 0.5375, respectively. These two models differ from all other models in that their input embedding layers encode the position information when projecting the raw data to a latent space, making the position information accessible by subsequent layers for feature extraction and resulting in superior performance.

Incorporating positional embedding decreased the average accuracy of the variants by 12.7% (convolutional embedding), 9.1% (convolutional attention), 18.6% (convolutional feed-forward), 22.1% (recurrent embedding), 21.5% (recurrent attention), and 15.7% (recurrent feed-forward), respectively. Results for the macro F1-Score show similar trends. Since the convolutional and recurrent layers can inherently capture the position information from sequential data, it is natural to consider positional embedding redundant for

Table 6.3: Accuracy of different models on 30 benchmark datasets.

Model	ArticularyWordRecognition	AtrialFibrillation	BasicMotions	CharacterTrajectories	Cricket	DuckDuckGeese
BasicModel (w/ PE)	0.4788	0.4524	1.0000	0.5140	0.9643	0.7667
BasicModel (w/o PE)	0.1280	0.2949	1.0000	0.4684	0.9395	0.5847
ConvEmbedding (w/ PE)	0.5125	0.4333	1.0000	0.4560	0.7468	0.6922
ConvEmbedding (w/o PE)	0.6774	0.7037	1.0000	0.6207	0.8016	0.7145
ConvAttention (w/ PE)	0.5413	0.5238	1.0000	0.2120	0.6515	0.6257
ConvAttention (w/o PE)	0.5091	0.4000	1.0000	0.1975	0.7433	0.3902
ConvFFD (w/ PE)	0.5232	0.4524	1.0000	0.2834	0.8775	0.5858
ConvFFD (w/o PE)	0.6483	0.5238	1.0000	0.3000	0.9623	0.6472
RecEmbedding (w/ PE)	0.5580	0.4524	1.0000	0.4227	0.8442	0.4800
RecEmbedding (w/o PE)	0.7321	0.6111	1.0000	0.5478	0.9339	0.6608
RecAttention (w/ PE)	0.7312	0.6444	1.0000	0.3842	0.6938	0.3120
RecAttention (w/o PE)	0.7548	0.6768	1.0000	0.7450	0.8371	0.6444
RecFFD (w/ PE)	0.6052	0.4167	1.0000	0.3405	0.6660	0.6065
RecFFD (w/o PE)	0.6890	0.5500	1.0000	0.5001	0.7978	0.6419
Model	EigenWorms	Epilepsy	JapaneseVowels	Libras	LSST	MotorImagery
BasicModel (w/ PE)	0.4447	0.8153	0.9616	0.0113	0.1716	0.5801
BasicModel (w/o PE)	0.4186	0.7753	0.9346	0.1527	0.1770	0.4332
ConvEmbedding (w/ PE)	0.3556	0.8156	0.9526	0.0317	0.1719	0.4264
ConvEmbedding (w/o PE)	0.4843	0.8725	0.9633	0.0760	0.1086	0.7525
ConvAttention (w/ PE)	0.3792	0.7377	0.7418	0.0878	0.1365	0.5000
ConvAttention (w/o PE)	0.3844	0.5564	0.7491	0.1352	0.1845	0.6420
ConvFFD (w/ PE)	0.4716	0.8041	0.9688	0.0298	0.0987	0.4391
ConvFFD (w/o PE)	0.4872	0.8954	0.9721	0.0490	0.1269	0.4585
RecEmbedding (w/ PE)	0.6724	0.8037	0.9566	0.0691	0.0461	0.5525
RecEmbedding (w/o PE)	0.7053	0.8538	0.9669	0.1070	0.0943	0.6326
RecAttention (w/ PE)	0.4171	0.6734	0.9668	0.1023	0.1006	0.6302
RecAttention (w/o PE)	0.6420	0.8163	0.9608	0.1897	0.0904	0.7551
RecFFD (w/ PE)	0.4199	0.7711	0.9702	0.1014	0.1370	0.4688
RecFFD (w/o PE)	0.6545	0.8353	0.9709	0.1159	0.1664	0.4719
Model	NATOPS	PenDigits	PEMS-SF	Phoneme	EthanolConcentration	ERing
BasicModel (w/ PE)	0.2834	0.1639	0.6204	0.0164	0.3880	0.6659
BasicModel (w/o PE)	0.1198	0.1203	0.8091	0.0110	0.0627	0.6065
ConvEmbedding (w/ PE)	0.2249	0.6416	0.8076	0.0126	0.1086	0.6982
ConvEmbedding (w/o PE)	0.3074	0.6452	0.8770	0.0291	0.1979	0.7695
ConvAttention (w/ PE)	0.2111	0.0784	0.8717	0.0170	0.0632	0.6660
ConvAttention (w/o PE)	0.2699	0.5021	0.8953	0.0211	0.1254	0.6913
ConvFFD (w/ PE)	0.2339	0.2036	0.8155	0.0142	0.0627	0.4966
ConvFFD (w/o PE)	0.2796	0.3388	0.8816	0.0142	0.1340	0.7357
RecEmbedding (w/ PE)	0.2951	0.3561	0.6314	0.0059	0.0618	0.4318
RecEmbedding (w/o PE)	0.4215	0.4066	0.8277	0.0106	0.4755	0.8008
RecAttention (w/ PE)	0.2638	0.1491	0.5946	0.0111	0.0985	0.6291
RecAttention (w/o PE)	0.4958	0.1711	0.8348	0.0243	0.1236	0.6488
RecFFD (w/ PE)	0.2797	0.1003	0.5953	0.0074	0.0618	0.7222
RecFFD (w/o PE)	0.4231	0.2755	0.8333	0.0123	0.1254	0.7735
Model	FaceDetection	FingerMovements	HandMovementDirection	Handwriting	Heartbeat	RacketSports
BasicModel (w/ PE)	0.6285	0.6075	0.2691	0.0239	0.7528	0.0879
BasicModel (w/o PE)	0.5487	0.3925	0.1830	0.0460	0.8627	0.0546
ConvEmbedding (w/ PE)	0.6461	0.5405	0.3032	0.0266	0.8663	0.4439
ConvEmbedding (w/o PE)	0.5589	0.5861	0.3224	0.0307	0.7506	0.4530
ConvAttention (w/ PE)	0.6762	0.7082	0.2908	0.0154	0.7238	0.2111
ConvAttention (w/o PE)	0.6816	0.7552	0.3442	0.0664	0.8663	0.5574
ConvFFD (w/ PE)	0.6457	0.6725	0.2858	0.0364	0.7258	0.1691
ConvFFD (w/o PE)	0.6347	0.7242	0.2950	0.4540	0.7272	0.3109
RecEmbedding (w/ PE)	0.6793	0.5187	0.4222	0.0400	0.3610	0.1716
RecEmbedding (w/o PE)	0.5500	0.5233	0.3779	0.0570	0.7096	0.2121
RecAttention (w/ PE)	0.6797	0.5028	0.2083	0.0318	0.3610	0.1970
RecAttention (w/o PE)	0.5425	0.6731	0.2559	0.0513	0.7528	0.3110
RecFFD (w/ PE)	0.6648	0.2550	0.4335	0.0189	0.7238	0.1253
RecFFD (w/o PE)	0.5527	0.4167	0.4454	0.0352	0.7435	0.2003
Model	SelfRegulationSCP1	SelfRegulationSCP2	SpokenArabicDigits	StandWalkJump	UWaveGestureLibrary	Average
BasicModel (w/ PE)	0.8494	0.5259	0.1111	0.5694	0.5284	0.4915
BasicModel (w/o PE)	0.8176	0.5125	0.1111	0.2778	0.2865	0.4183
ConvEmbedding (w/ PE)	0.8720	0.5847	0.4433	0.2778	0.3656	0.4986
ConvEmbedding (w/o PE)	0.8846	0.5694	0.5988	0.5500	0.3938	0.5621
ConvAttention (w/ PE)	0.8188	0.5207	0.4681	0.4524	0.4763	0.4623
ConvAttention (w/o PE)	0.8592	0.5000	0.6620	0.5500	0.3818	0.5042
ConvFFD (w/ PE)	0.8375	0.5250	0.5591	0.1944	0.3472	0.4607
ConvFFD (w/o PE)	0.8979	0.6080	0.6418	0.5500	0.5496	0.5465
RecEmbedding (w/ PE)	0.7935	0.2500	0.5833	0.4778	0.4031	0.4600
RecEmbedding (w/o PE)	0.8472	0.2500	0.7232	0.8182	0.4338	0.5617
RecAttention (w/ PE)	0.8494	0.5000	0.5112	0.5500	0.4113	0.4553
RecAttention (w/o PE)	0.8746	0.5710	0.5420	0.7167	0.3374	0.5531
RecFFD (w/ PE)	0.8658	0.4486	0.4491	0.4667	0.3620	0.4512
RecFFD (w/o PE)	0.8843	0.5752	0.4555	0.4615	0.5370	0.5222

Table 6.4: Macro F1-Score of different models on 30 benchmark datasets.

Model	ArticularyWordRecognition	AtrialFibrillation	BasicMotions	CharacterTrajectories	Cricket	DuckDuckGeese
BasicModel (w/ PE)	0.5395	0.3523	1.0000	0.5783	0.9581	0.7177
BasicModel (w/o PE)	0.0988	0.3297	1.0000	0.5687	0.9156	0.4478
ConvEmbedding (w/ PE)	0.6170	0.4139	1.0000	0.5554	0.7208	0.5365
ConvEmbedding (w/o PE)	0.7226	0.6035	1.0000	0.6734	0.8285	0.6379
ConvAttention (w/ PE)	0.5094	0.4000	1.0000	0.1879	0.6382	0.3959
ConvAttention (w/o PE)	0.5821	0.5157	1.0000	0.3141	0.7758	0.4760
ConvFFD (w/ PE)	0.5921	0.3529	1.0000	0.3892	0.8719	0.5365
ConvFFD (w/o PE)	0.7152	0.3333	1.0000	0.4318	0.9582	0.6360
RecEmbedding (w/ PE)	0.6272	0.3591	1.0000	0.4510	0.8194	0.4748
RecEmbedding (w/o PE)	0.7605	0.4577	1.0000	0.6335	0.9025	0.5304
RecAttention (w/ PE)	0.7599	0.4603	1.0000	0.4473	0.7108	0.3354
RecAttention (w/o PE)	0.7767	0.4553	1.0000	0.7868	0.8600	0.5338
RecFFD (w/ PE)	0.6615	0.4142	1.0000	0.4351	0.7121	0.5338
RecFFD (w/o PE)	0.7314	0.5161	1.0000	0.5712	0.8176	0.6114

Model	EigenWorms	Epilepsy	JapaneseVowels	Libras	LSST	MotorImagery
BasicModel (w/ PE)	0.4020	0.8169	0.9670	0.0321	0.2009	0.4325
BasicModel (w/o PE)	0.3912	0.7667	0.9439	0.1307	0.1297	0.5799
ConvEmbedding (w/ PE)	0.2471	0.7568	0.9514	0.0148	0.1504	0.4265
ConvEmbedding (w/o PE)	0.2849	0.8345	0.9600	0.0403	0.1220	0.4500
ConvAttention (w/ PE)	0.2460	0.5912	0.7222	0.0581	0.0555	0.4120
ConvAttention (w/o PE)	0.3592	0.7100	0.6509	0.0158	0.1317	0.6393
ConvFFD (w/ PE)	0.3695	0.7667	0.9656	0.0568	0.0585	0.4390
ConvFFD (w/o PE)	0.4923	0.8941	0.9744	0.0744	0.0829	0.4623
RecEmbedding (w/ PE)	0.5984	0.7479	0.9566	0.1339	0.0752	0.5498
RecEmbedding (w/o PE)	0.6606	0.8557	0.9679	0.1508	0.0681	0.6291
RecAttention (w/ PE)	0.3231	0.5496	0.9686	0.0690	0.0950	0.4630
RecAttention (w/o PE)	0.5197	0.8072	0.9673	0.2292	0.0830	0.5526
RecFFD (w/ PE)	0.3738	0.7462	0.9721	0.1448	0.1312	0.4533
RecFFD (w/o PE)	0.6519	0.8410	0.9741	0.1512	0.0724	0.4547

Model	NATOPS	PenDigits	PEMS-SF	Phoneme	EthanolConcentration	ERing
BasicModel (w/ PE)	0.3127	0.0869	0.7693	0.0236	0.2152	0.6486
BasicModel (w/o PE)	0.2291	0.0747	0.6615	0.0191	0.1432	0.5273
ConvEmbedding (w/ PE)	0.3257	0.6173	0.7872	0.0234	0.1739	0.6498
ConvEmbedding (w/o PE)	0.3974	0.6262	0.8655	0.0247	0.2096	0.7679
ConvAttention (w/ PE)	0.2333	0.1291	0.8543	0.0099	0.0777	0.5992
ConvAttention (w/o PE)	0.3031	0.4512	0.8900	0.0259	0.2224	0.6600
ConvFFD (w/ PE)	0.2121	0.2591	0.7932	0.0249	0.1432	0.5523
ConvFFD (w/o PE)	0.3564	0.3129	0.8763	0.0207	0.1581	0.7092
RecEmbedding (w/ PE)	0.3635	0.4070	0.6814	0.0142	0.1419	0.4622
RecEmbedding (w/o PE)	0.3715	0.4176	0.7929	0.0149	0.2600	0.7814
RecAttention (w/ PE)	0.1894	0.2101	0.6375	0.0205	0.1483	0.5415
RecAttention (w/o PE)	0.3423	0.2118	0.8277	0.0234	0.2217	0.6511
RecFFD (w/ PE)	0.2344	0.1380	0.6328	0.0165	0.1419	0.6853
RecFFD (w/o PE)	0.4271	0.3123	0.7916	0.0238	0.2224	0.7615

Model	FaceDetection	FingerMovements	HandMovementDirection	Handwriting	Heartbeat	RacketSports
BasicModel (w/ PE)	0.6270	0.5580	0.2603	0.0441	0.6119	0.1502
BasicModel (w/o PE)	0.5483	0.4029	0.1593	0.0680	0.6167	0.1304
ConvEmbedding (w/ PE)	0.6454	0.4931	0.2703	0.0549	0.6167	0.2044
ConvEmbedding (w/o PE)	0.5563	0.5310	0.2960	0.0598	0.6119	0.3170
ConvAttention (w/ PE)	0.6814	0.5710	0.3052	0.0369	0.7351	0.2400
ConvAttention (w/o PE)	0.6762	0.5181	0.1970	0.0622	0.5950	0.2582
ConvFFD (w/ PE)	0.6396	0.5612	0.3102	0.0454	0.6144	0.2732
ConvFFD (w/o PE)	0.5544	0.5923	0.2349	0.0631	0.6093	0.2814
RecEmbedding (w/ PE)	0.6793	0.5154	0.2097	0.0713	0.4561	0.1677
RecEmbedding (w/o PE)	0.5499	0.5211	0.3298	0.0795	0.5702	0.2394
RecAttention (w/ PE)	0.6793	0.4997	0.1763	0.0594	0.4561	0.2551
RecAttention (w/o PE)	0.5419	0.5457	0.2305	0.0653	0.6091	0.2499
RecFFD (w/ PE)	0.6642	0.4031	0.3617	0.0465	0.5950	0.1940
RecFFD (w/o PE)	0.5523	0.4174	0.3231	0.0609	0.5986	0.3186

Model	SelfRegulationSCP1	SelfRegulationSCP2	SpokenArabicDigits	StandWalkJump	UWaveGestureLibrary	Average
BasicModel (w/ PE)	0.8167	0.4669	0.2222	0.4260	0.5309	0.4748
BasicModel (w/o PE)	0.8120	0.5040	0.2222	0.2947	0.3282	0.4153
ConvEmbedding (w/ PE)	0.8632	0.5131	0.5001	0.2871	0.3785	0.4757
ConvEmbedding (w/o PE)	0.8839	0.5448	0.6328	0.4611	0.3677	0.5280
ConvAttention (w/ PE)	0.2548	0.4346	0.3021	0.2353	0.3867	0.3898
ConvAttention (w/o PE)	0.8119	0.4434	0.4141	0.2559	0.4812	0.4633
ConvFFD (w/ PE)	0.8249	0.5217	0.6121	0.2508	0.3023	0.4600
ConvFFD (w/o PE)	0.8976	0.5740	0.6497	0.4496	0.5897	0.5167
RecEmbedding (w/ PE)	0.7074	0.4000	0.5586	0.4680	0.4077	0.4657
RecEmbedding (w/o PE)	0.8132	0.4000	0.6771	0.7370	0.4143	0.5375
RecAttention (w/ PE)	0.8167	0.4511	0.4883	0.4128	0.3663	0.4341
RecAttention (w/o PE)	0.8736	0.5425	0.5469	0.5220	0.3617	0.5151
RecFFD (w/ PE)	0.8524	0.4133	0.5315	0.2353	0.3765	0.4517
RecFFD (w/o PE)	0.8839	0.5533	0.5451	0.4163	0.5811	0.5235

Table 6.5: The mean absolute error of different models on selected datasets under input sequence length size 96.

Model	ETTh1	ETTh2	ETTm1	ETTm2	Average
BasicModel (w/ PE)	1.3801	0.8784	1.0757	0.8758	1.0525
BasicModel (w/o PE)	1.3885	1.1257	1.3596	1.2475	1.2803
ConvEmbedding (w/ PE)	0.4807	0.7875	0.7366	0.8524	0.7143
ConvEmbedding (w/o PE)	0.4805	0.7584	0.6611	0.6645	0.6411
ConvAttention (w/ PE)	0.4804	0.7687	0.8027	0.9308	0.7457
ConvAttention (w/o PE)	0.4806	0.7844	0.7462	0.9210	0.7331
ConvFFD (w/ PE)	0.4815	0.7116	0.7042	0.9825	0.7200
ConvFFD (w/o PE)	0.4727	0.7132	0.6966	0.8596	0.6855
RecEmbedding (w/ PE)	0.4803	0.6917	0.8004	0.8825	0.7137
RecEmbedding (w/o PE)	0.4794	0.7089	0.6445	0.7038	0.6342
RecAttention (w/ PE)	0.4808	0.7106	0.7788	0.6473	0.6544
RecAttention (w/o PE)	0.4806	0.6975	0.7289	0.6821	0.6473
RecFFD (w/ PE)	0.4816	0.7464	0.7844	0.9309	0.7358
RecFFD (w/o PE)	0.4803	0.7437	0.7905	0.8641	0.7197

Transformer-based variants. Besides, positional embedding risks introducing inductive bias and contaminating the original data. Specifically, positional embedding injects the same information into sequences of different classes, bringing new challenges to the classifiers; this may also contribute to performance degradation.

Further reflecting on the results, we suggest that positional embedding may not be necessary for Transformer-based variants that already contain position-sensitive modules. In particular, for time series classification tasks, while the classifier focuses on the differences between time series sequences across different classes, positional embedding is content-irrelevant, adding the same position information to all sequences regardless of their class labels. As position-sensitive modules generally consider content information when encoding the position information, redundant content-irrelevant positional embedding may lead the model toward capturing spurious correlations that potentially hinder the classifier's performance.

6.5.2 Forecasting

Table 6.5, Table 6.6, Table 6.7, Table 6.8, Table 6.9, and Table 6.10 present the evaluation results of all methods on the selected datasets different input sequence lengths. We observe that positional embedding improves the basic model's performance signif-

Table 6.6: The mean square error of different models on selected datasets under input sequence length 96.

Model	ETTh1	ETTh2	ETTm1	ETTm2	Average
BasicModel (w/ PE)	2.0213	1.1170	1.3299	1.1110	1.3948
BasicModel (w/o PE)	2.1449	1.5866	1.9667	1.8203	1.8796
ConvEmbedding (w/ PE)	0.3382	1.1074	1.0889	0.7914	0.8315
ConvEmbedding (w/o PE)	0.3380	1.1048	0.6735	0.6945	0.7027
ConvAttention (w/ PE)	0.3378	1.1061	1.0896	1.1309	0.9161
ConvAttention (w/o PE)	0.3381	1.1072	1.0993	1.1028	0.9119
ConvFFD (w/ PE)	0.3481	1.0137	1.0925	1.1416	0.8990
ConvFFD (w/o PE)	0.3315	1.0144	0.7983	1.0292	0.7934
RecEmbedding (w/ PE)	0.3377	0.9779	1.1280	1.0369	0.8701
RecEmbedding (w/o PE)	0.3359	1.0116	0.6481	0.9908	0.7466
RecAttention (w/ PE)	0.3383	1.0124	0.9726	0.5389	0.7156
RecAttention (w/o PE)	0.3379	0.9882	0.8847	0.5976	0.7021
RecFFD (w/ PE)	0.3482	1.0848	1.1227	1.2708	0.9566
RecFFD (w/o PE)	0.3378	1.0779	1.1596	1.0102	0.8964

icantly while demonstrating input length-dependent impact on variants. Specifically, without the positional embedding, the basic model experiences an average decrease of 25.8% in MSE and 17.8% in MAE, respectively, under input sequence length 96, an average decrease of 38.5% in MSE and 24.4% in MAE, respectively, under input sequence length 192, an average decrease of 25.7% in MSE and 17.7% in MAE, respectively under input sequence length 336. In contrast, for all variants, when the input sequence length equals to 96, the average MSE and MAE of the variants increase by 18.3% and 21.6% (convolutional embedding), 0.5% and 0.2% (convolutional attention), 13.3% and 6% (convolutional feed-forward), 16.5% and 12.5% (recurrent embedding), 1.9% and 1.1% (recurrent attention), and 6.7% and 2.3% (recurrent feed-forward) after removing the positional embedding. When the input sequence length equals to 192, the average MSE and MAE of all variants show insignificant differences (MSE and MAE decrease by 6% and 1%, respectively) after removing the positional embedding. When the input sequence length equals to 336, the average MSE and MAE of all variants decrease by 8.2% and 13.1% (convolutional embedding), 14.9% and 16.5% (convolutional attention), 10.6% and 6.8% (convolutional feed-forward), 15.6% and 9.1% (recurrent embedding), 16.9% and 18.6% (recurrent attention), and 15.1% and 20.9% (recurrent feed-forward) after removing the positional embedding. Similar to classification, the convolutional em-

Table 6.7: The mean absolute error of different models on selected datasets under input sequence length 192.

Model	ETTh1	ETTh2	ETTm1	ETTm2	Average
BasicModel (w/ PE)	1.1132	0.7851	1.2331	0.8086	1.0733
BasicModel (w/o PE)	1.9907	2.0043	1.9369	1.8127	1.4203
ConvEmbedding (w/ PE)	0.3789	0.5231	0.5219	0.3868	0.4527
ConvEmbedding (w/o PE)	0.3497	0.4298	0.4426	0.3641	0.3966
ConvAttention (w/ PE)	0.3482	0.4556	0.5623	0.3619	0.4321
ConvAttention (w/o PE)	0.3628	0.4479	0.5167	0.3523	0.4199
ConvFFD (w/ PE)	0.4176	0.5739	0.4956	0.3496	0.4592
ConvFFD (w/o PE)	0.4649	0.5895	0.5059	0.3721	0.4831
RecEmbedding (w/ PE)	0.4493	0.5084	0.5498	0.3669	0.4686
RecEmbedding (w/o PE)	0.4081	0.4445	0.4941	0.3586	0.4263
RecAttention (w/ PE)	0.4362	0.4261	0.4753	0.3936	0.4328
RecAttention (w/o PE)	0.3934	0.4471	0.5152	0.3862	0.4355
RecFFD (w/ PE)	0.4168	0.4367	0.4573	0.3941	0.4262
RecFFD (w/o PE)	0.4086	0.4289	0.4654	0.3843	0.4218

bedding and recurrent embedding models without the positional embedding achieve the best performance in terms of average MAE and MSE. Our results suggest that encoding position information before multi-head attention and feed-forward layers significantly improves the models' performance in both forecasting and classification.

6.5.3 Analysis

The positional embedding enhances the basic model's performance for both classification and forecasting tasks, while its impact on the variants is task-wise. For all variants, the decreases in average accuracy and average macro F1-Score caused by positional embedding are 16.6% and 15.4%, respectively, in classification. For forecasting, the performance degradation varies under different input sequence lengths. The MAE and MSE have an average decrease of 5.2% and 8.3%, respectively, when the input sequence length is 96. While the positional embedding slightly improves the variants' performances with an average increase of 6% and 1% in MSE and MAE, respectively with the input sequence length 192. When the input sequence length is 336, the positional embedding positively impacts the variants' performance, increasing the average MSE and MAE by 14.1% and 12.6%, respectively. With the increase of the input sequence length, all the methods realize better performances, but the positional embedding shows a different

Table 6.8: The mean square error of different models on selected datasets under input sequence length 192.

Model	ETTh1	ETTh2	ETTm1	ETTm2	Average
BasicModel (w/ PE)	2.9371	1.2642	2.9371	1.6074	2.1865
BasicModel (w/o PE)	3.6902	3.4508	3.5946	3.4794	3.5538
ConvEmbedding (w/ PE)	0.2986	0.4283	0.5803	0.4026	0.4275
ConvEmbedding (w/o PE)	0.2965	0.3371	0.5467	0.3492	0.3824
ConvAttention (w/ PE)	0.3532	0.3648	0.4699	0.3494	0.3843
ConvAttention (w/o PE)	0.4145	0.3279	0.4482	0.3379	0.3821
ConvFFD (w/ PE)	0.2901	0.5072	0.4866	0.3351	0.4048
ConvFFD (w/o PE)	0.3296	0.5698	0.5741	0.3755	0.4623
RecEmbedding (w/ PE)	0.3472	0.5386	0.5108	0.4243	0.4552
RecEmbedding (w/o PE)	0.3336	0.4269	0.5069	0.3769	0.4111
RecAttention (w/ PE)	0.3939	0.4331	0.4817	0.4248	0.4334
RecAttention (w/o PE)	0.3682	0.4774	0.5442	0.3985	0.4471
RecFFD (w/ PE)	0.3941	0.4156	0.4818	0.4149	0.4266
RecFFD (w/o PE)	0.3518	0.4069	0.4907	0.3915	0.4102

Table 6.9: The mean absolute error of different models on selected datasets under input sequence length 336.

Model	ETTh1	ETTh2	ETTm1	ETTm2	Average
BasicModel (w/ PE)	1.3801	0.8784	1.0757	0.8758	1.0525
BasicModel (w/o PE)	1.3885	1.1257	1.3596	1.2475	1.2803
ConvEmbedding (w/ PE)	0.3049	0.3475	0.3051	0.2901	0.3119
ConvEmbedding (w/o PE)	0.3237	0.3805	0.3182	0.3362	0.3397
ConvAttention (w/ PE)	0.3123	0.3145	0.3273	0.2943	0.3121
ConvAttention (w/o PE)	0.3491	0.3819	0.3924	0.3449	0.3671
ConvFFD (w/ PE)	0.3089	0.3571	0.3486	0.3355	0.3375
ConvFFD (w/o PE)	0.3812	0.3785	0.3936	0.3567	0.3775
RecEmbedding (w/ PE)	0.3131	0.3145	0.3437	0.3275	0.3247
RecEmbedding (w/o PE)	0.3842	0.3916	0.3748	0.3887	0.3848
RecAttention (w/ PE)	0.3062	0.3071	0.3269	0.2919	0.3080
RecAttention (w/o PE)	0.3783	0.3645	0.3910	0.3494	0.3708
RecFFD (w/ PE)	0.3141	0.3124	0.3291	0.2981	0.3134
RecFFD (w/o PE)	0.3804	0.3721	0.4162	0.3082	0.3692

Table 6.10: The mean square error of different models on selected datasets under input sequence length 336.

Model	ETTh1	ETTh2	ETTm1	ETTm2	Average
BasicModel (w/ PE)	2.0213	1.1170	1.3299	1.1110	1.3948
BasicModel (w/o PE)	2.1449	1.5866	1.9667	1.8203	1.8796
ConvEmbedding (w/ PE)	0.3087	0.3198	0.3149	0.2968	0.3101
ConvEmbedding (w/o PE)	0.3679	0.4031	0.3276	0.3279	0.3566
ConvAttention (w/ PE)	0.3165	0.2906	0.2954	0.2998	0.3006
ConvAttention (w/o PE)	0.3557	0.3715	0.3957	0.3174	0.3601
ConvFFD (w/ PE)	0.2955	0.3369	0.3349	0.2901	0.3144
ConvFFD (w/o PE)	0.3254	0.3402	0.3731	0.3113	0.3375
RecEmbedding (w/ PE)	0.2957	0.3289	0.3257	0.2854	0.3089
RecEmbedding (w/o PE)	0.3467	0.3294	0.3559	0.3261	0.3395
RecAttention (w/ PE)	0.3044	0.3188	0.3187	0.2798	0.3054
RecAttention (w/o PE)	0.3978	0.3921	0.3891	0.3211	0.3750
RecFFD (w/ PE)	0.2964	0.3184	0.3186	0.2935	0.3067
RecFFD (w/o PE)	0.4076	0.3942	0.4177	0.3304	0.3875

Table 6.11: The classification experimental results comparison between the models on the original data and the inverted data (degradation is the comparison between the model under original data and inverted data).

Model	Accuracy	Precision	Recall	macro F1-Score	Degradation (Accuracy)	Degradation (macro F1-Score)
BasicModel (inverse)	0.6031	0.5519	0.5338	0.5427	0.56%	-2.92%
BasicModel	0.6065	0.5593	0.4988	0.5273		
ConvEmbedding (inverse)	0.6172	0.5870	0.5794	0.5832	19.79%	24.05%
ConvEmbedding	0.7695	0.7685	0.7674	0.7679		
ConvAttention (inverse)	0.5766	0.5574	0.4704	0.5102	16.59%	22.70%
ConvAttention	0.6913	0.6741	0.6464	0.6600		
ConvFFD (inverse)	0.5255	0.5093	0.4987	0.5039	28.57%	28.95%
ConvFFD	0.7357	0.7148	0.7036	0.7092		
RecEmbedding (inverse)	0.4749	0.4667	0.4263	0.4456	40.70%	42.97%
RecEmbedding	0.8008	0.7852	0.7776	0.7814		
RecAttention (inverse)	0.5505	0.5778	0.5367	0.5565	15.15%	14.53%
RecAttention	0.6488	0.6556	0.6466	0.6511		
RecFFD (inverse)	0.6590	0.6741	0.6504	0.6620	14.80%	13.07%
RecFFD	0.7735	0.7630	0.7600	0.7615		

impact on the variants. We believe that long-time series sequences lead to challenges for the convolutional layer and the recurrent layer to encode long-term position information as they lack the capacity to capture long-term dependencies [11, 88], making the variants relying on the sequential order information provided by the positional embedding.

Noted, on some classification datasets such as Cricket (the sequence's length is 1197),

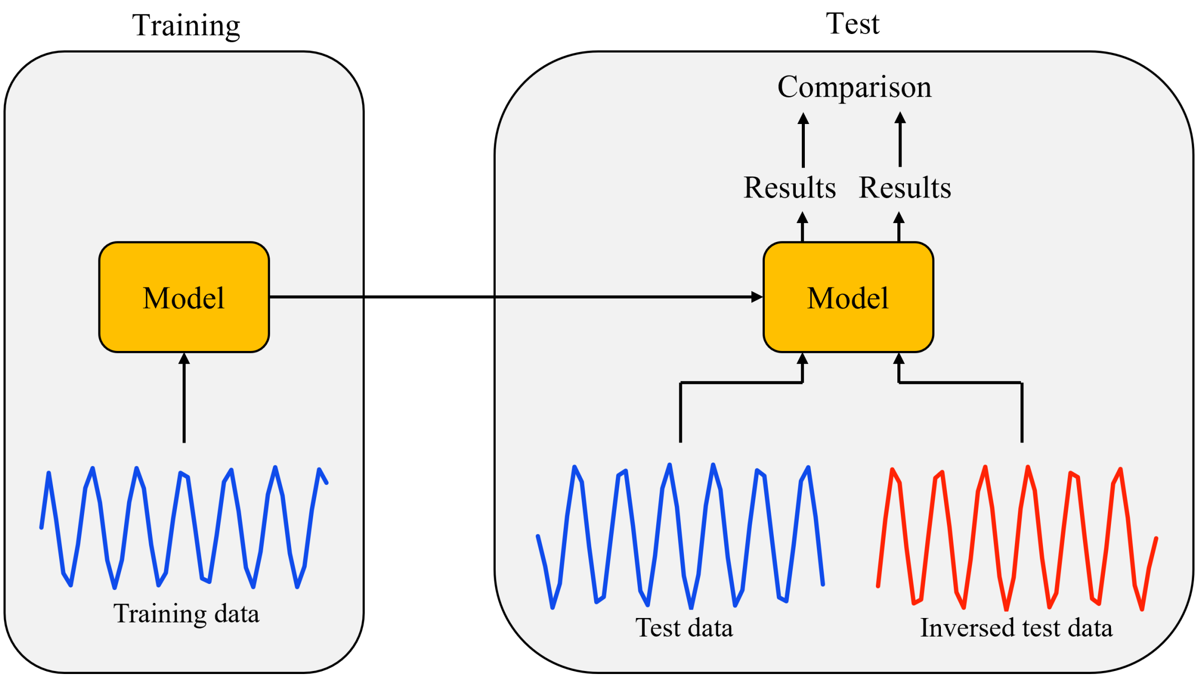


Figure 6.3: The pipeline of the time series inverse study. We first train all methods using the samples from the training set without positional embedding. We then use the samples and the inverted samples in the test set to evaluate the trained models and explore the performance differences.

EigenWorms (the sequence's length is 17984), and MotorImagery (the sequence's length is 3000), although the length of time series sequences from these datasets is much longer than 336, the variants realize better performances without the positional embedding. We think that is because the classification task does not require strict position information compared with forecasting. Hence, the convolutional layer or recurrent layer has enough capacity to encode the position information to model time series sequences.

Besides, we find that the convolutional embedding and the recurrent embedding achieve the best performance among the convolutional-based and recurrent-based variants and they both significantly outperform the basic model with the positional embedding, indicating the importance of encoding the position information before attention calculation and feed-forward layer.

We believe that the impact of the positional embedding on model performance varies across classification and forecasting tasks due to differences in the temporal information that the model focuses on. Specifically, for classification tasks, the model concentrates on the differences between time series data across different classes, whereas positional embedding is content-irrelevant, i.e., adding the same position information to data with

Table 6.12: The forecasting experimental results comparison between the models on the original data and the inverted data (degradation is the comparison between the model under original data and inverted data).

Model	MSE	MAE	Degradation (MSE)	Degradation (MAE)
BasicModel (inverse)	2.1659	1.4596	0.98%	5.12%
BasicModel	2.1449	1.3885		
ConvEmbedding (inverse)	1.5588	1.3756	361.23%	186.30%
ConvEmbedding	0.3380	0.4805		
ConvAttention (inverse)	0.9672	0.9910	186.07%	106.20%
ConvAttention	0.3381	0.4806		
ConvFFD (inverse)	1.5957	1.1928	381.36%	152.34%
ConvFFD	0.3315	0.4727		
RecEmbedding (inverse)	1.4649	1.2381	336.11%	158.26%
RecEmbedding	0.3359	0.4794		
RecAttention (inverse)	1.3485	1.2235	299.08%	154.58%
RecAttention	0.3379	0.4806		
RecFFD (inverse)	0.7811	0.8320	131.23%	73.23%
RecFFD	0.3378	0.4803		

various labels. Because the convolutional layer and the recurrent layer consider content information when encoding the position information, redundant content-irrelevant positional embedding may lead to capturing spurious correlations that could harm the performance. On the other hand, time series forecasting requires the model to focus on the temporal patterns of the data itself rather than the differences between classes and to harness the strict sequential order information to infer future time series data. Consequently, utilizing positional embedding does not significantly harm the model's forecasting performance when the length of the input time series is relatively short, while it improves variants' performance facing long input time series sequences.

6.6 Time Series Inverse Study

We further examined the ability to capture position information of the basic model and its variants through the order inverse study illustrated in Figure 6.3. Specifically, we trained all models without positional embedding and subsequently tested them after inverting the sequential order of the input data in the test set. We recorded the performance of each model and the attention distribution heat maps of the last self-attention layer. By comparing the results obtained from the original and inverted test data, we aimed to investigate the impact of sequential order information on all methods. Further details are presented in the following sections.

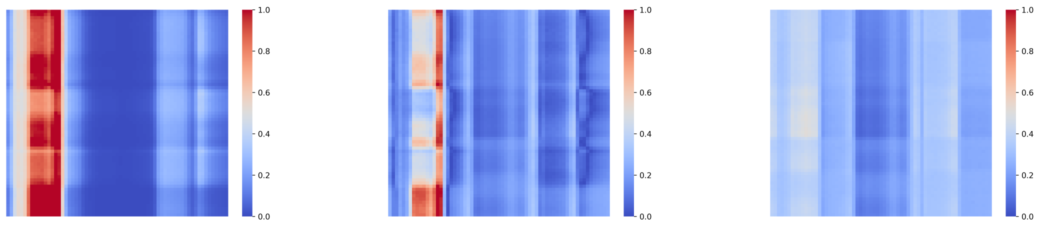
6.6.1 Classification

We chose **ERing** for our experiments for two main reasons: Firstly, its time series has a length of 65, allowing for more intuitive and interpretable attention heat maps than longer sequences. Secondly, all the tested models demonstrated satisfactory performance (with an average accuracy of 0.7180). This suggests that the models focused on the most critical slots of the time series, ensuring the attention heat maps produced are convincing. We additionally use *precision* and *recall* to gain further insights into how the models perform. The results of the basic model, convolutional-based variants and recurrent-based variants can be found in Table 6.11.

The evaluation results demonstrate that inverting the order of the time series data has nearly no impact on the basic model, but it causes a drastic degradation in the variants' performance. Specifically, the average accuracy and macro F1-Score of the variants decrease by 22.6% and 24.4%, respectively.

To investigate the variants' sensitivity to the changes in the sequential order information, we explore the attention distributions of all models under the original and inverted time series data and the difference between them. The attention distribution heat maps are shown in Figure 6.4, Figure 6.5, and Figure 6.6. We find that the variants are indeed sensitive to changes in data order, as the attention distribution varies significantly after inverting the sequential order of the test data. We believe that the variants, which contain recurrent or convolutional layers, can capture the potential position information of the data. Therefore, the inverted sequential order negatively impacts the model's performance by misleading it into focusing on unreasonable time slots. In contrast, the results of the basic model are different. Figure 6.4 (c) shows that the basic model's attention distribution varies slightly after inverting the test data compared with the variants, indicating that the vanilla Transformer is insensitive to the position information of the data.

From either classification results and attention distribution heat maps, we find that spurious position information caused by inverting the data's sequential order impacts the basic model and the variants differently. The variants' performance deteriorates significantly, while the basic model's performance is slightly affected. We believe the variants' ability to capture potential position information makes it possible to be deceived by spurious position information. While inverting the order cannot significantly affect the



(a) BasicModel (b) BasicModel (inverse) (c) BasicModel (diff)

Figure 6.4: Attention distribution heat maps of the basic model under original and inverted test data in classification. Diff means difference, resulting from the subtraction between the attention weights under the original and inverted data.

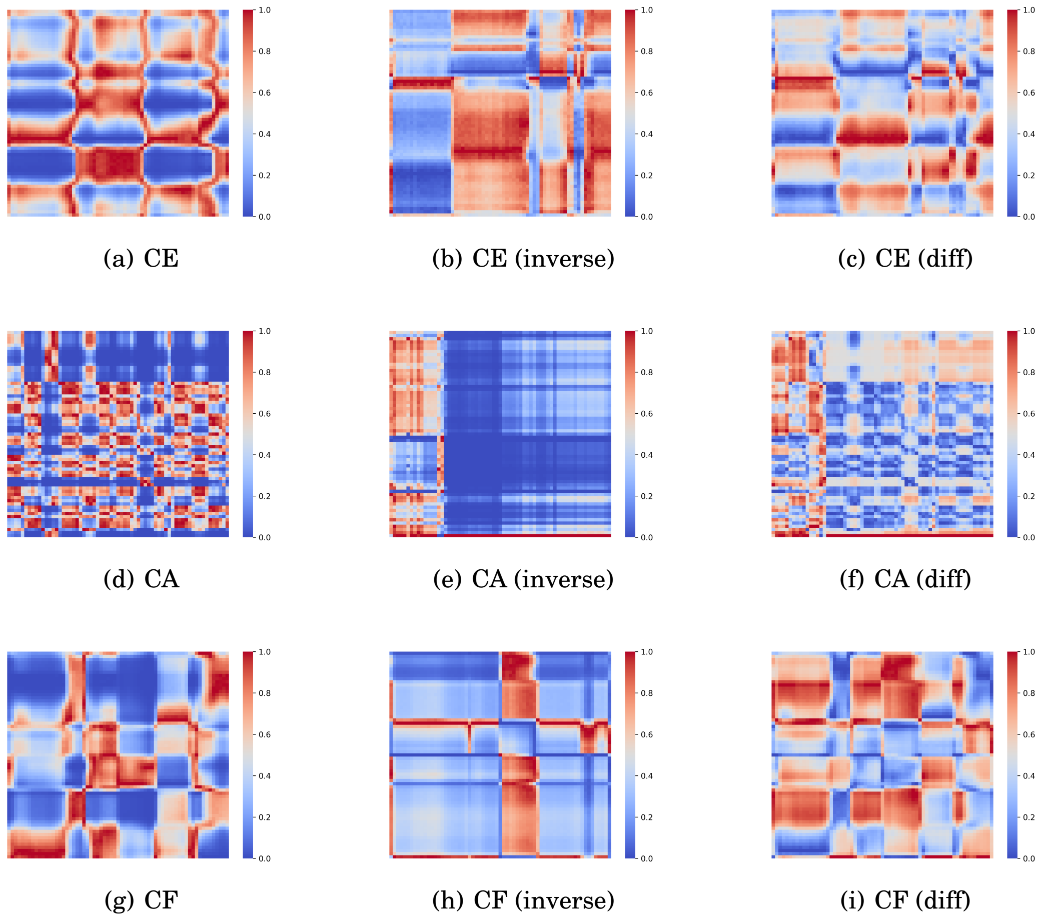
basic model as it is insensitive to position information and can not observe the changes in the sequential order.

6.6.2 Forecasting

Our experiments on the **ETTh1** dataset under the input sequence length 96 used the same pipeline as the classification task, and the results in Table 6.12, Figure 6.7, Figure 6.8, and Figure 6.9 support our previous conclusions. However, we observe that the performance degradation of all variants is more significant in the forecasting than in classification. Specifically, the average degradation of mean square error and mean absolute error are 282.51% and 138.48%, respectively, while the average degradation of accuracy and macro F1-Score are 22.60% and 24.38%, respectively. We believe that the strict and precise sequential order required for inferring future time series data in the forecasting task may account for this difference, whereas the classification task focuses more on differences between the time series sequences with different classes rather than their sequential order.

6.7 Conclusion

Current Transformer-based architectures containing position-sensitive layers routinely use positional embedding without comprehensively considering its effectiveness. In this work, we investigate the impact of positional embedding on the vanilla Transformer architecture and six types of Transformer-based variants in time series classification and forecasting. Our experimental results on 30 public classification datasets and 4 forecasting datasets revealed that the positional embedding positively influences the vanilla



(a) CE (b) CE (inverse) (c) CE (diff)

(d) CA (e) CA (inverse) (f) CA (diff)

(g) CF (h) CF (inverse) (i) CF (diff)

Figure 6.5: Attention distribution heat maps of convolutional-based variants under original and inverted test data in classification. CE, CA, and CF mean convolutional embedding, convolutional attention, and convolutional feed-forward, respectively. Diff means difference, resulting from the subtraction between the attention weights under the original and inverted data.

Transformer architecture in both tasks, while its influence on the variants is negative and input sequence length-dependent for classification and forecasting, respectively. Our analysis of the model's performance and attention distributions under inverted time series data shows that the ability of the variants to capture potential position information resulted in focusing on the spurious sequential order information caused by the inverted sequential order, leading to a more drastic degradation in their performance compared to the basic model. Our findings suggest that position-sensitive layers in Transformer-based architectures can encode the potential position information, likely making positional embedding unnecessary in classification but its utilization may depend on the length

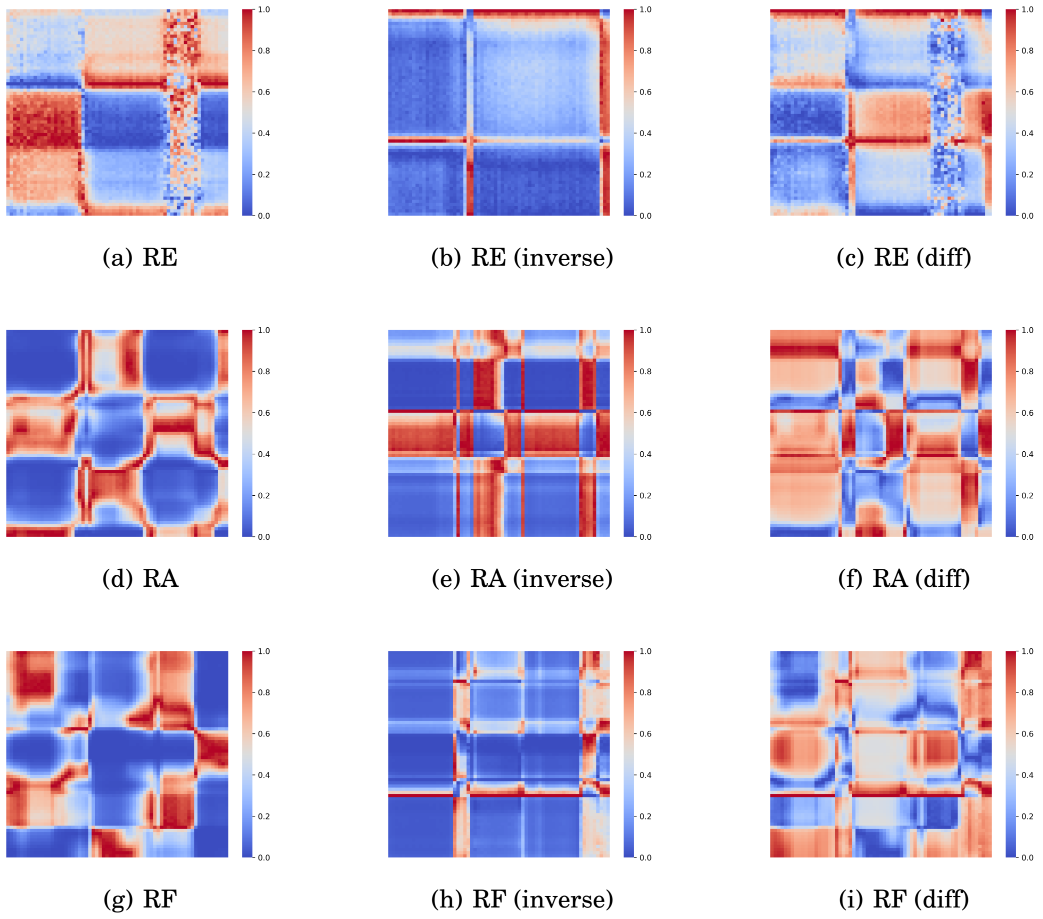


Figure 6.6: Attention distribution heat maps of recurrent-based variants under original and inverted test data in classification. RE, RA, and RF mean recurrent embedding, recurrent attention, and recurrent feed-forward, respectively. Diff means difference, resulting from the subtraction between the attention weights under the original and inverted data.

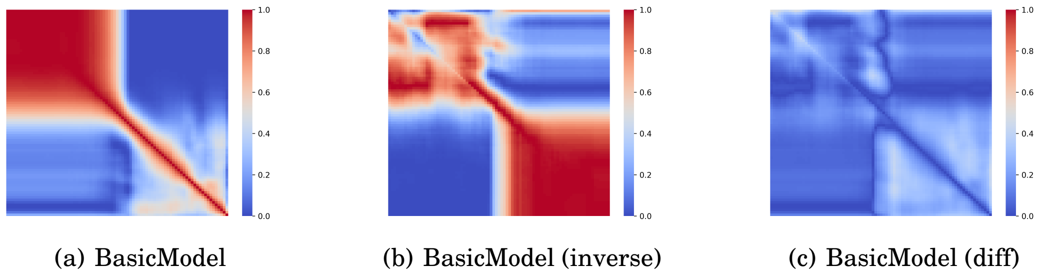
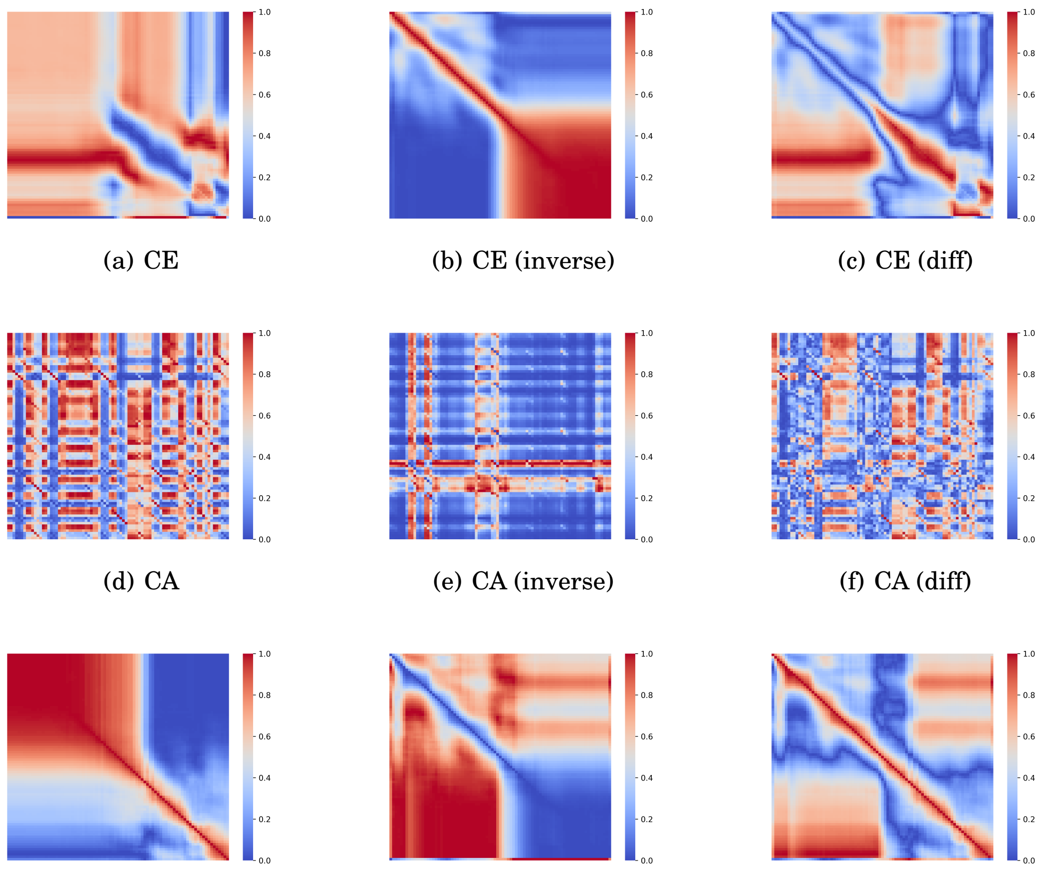


Figure 6.7: Attention distribution heat maps of the basic model under original and inverted test data in forecasting. Diff means difference, resulting from the subtraction between the attention weights under the original and inverted data.



(a) CE (b) CE (inverse) (c) CE (diff)

(d) CA (e) CA (inverse) (f) CA (diff)

(g) CF (h) CF (inverse) (i) CF (diff)

Figure 6.8: Attention distribution heat maps of convolutional-based variants under original and inverted test data in forecasting. CE, CA, and CF mean convolutional embedding, convolutional attention, and convolutional feed-forward, respectively. Diff means difference, resulting from the subtraction between the attention weights under the original and inverted data.

of the input time series in forecasting. Additionally, the convolutional embedding and the recurrent embedding outperform other variants, making them preferable when considering position-sensitive layers in the Transformer-based architecture. As most existing transformer-based architectures use positional embedding routinely without consideration of its effectiveness, through this study, we hope to bring some insights regarding how to implement positional embedding facing specific scenarios.

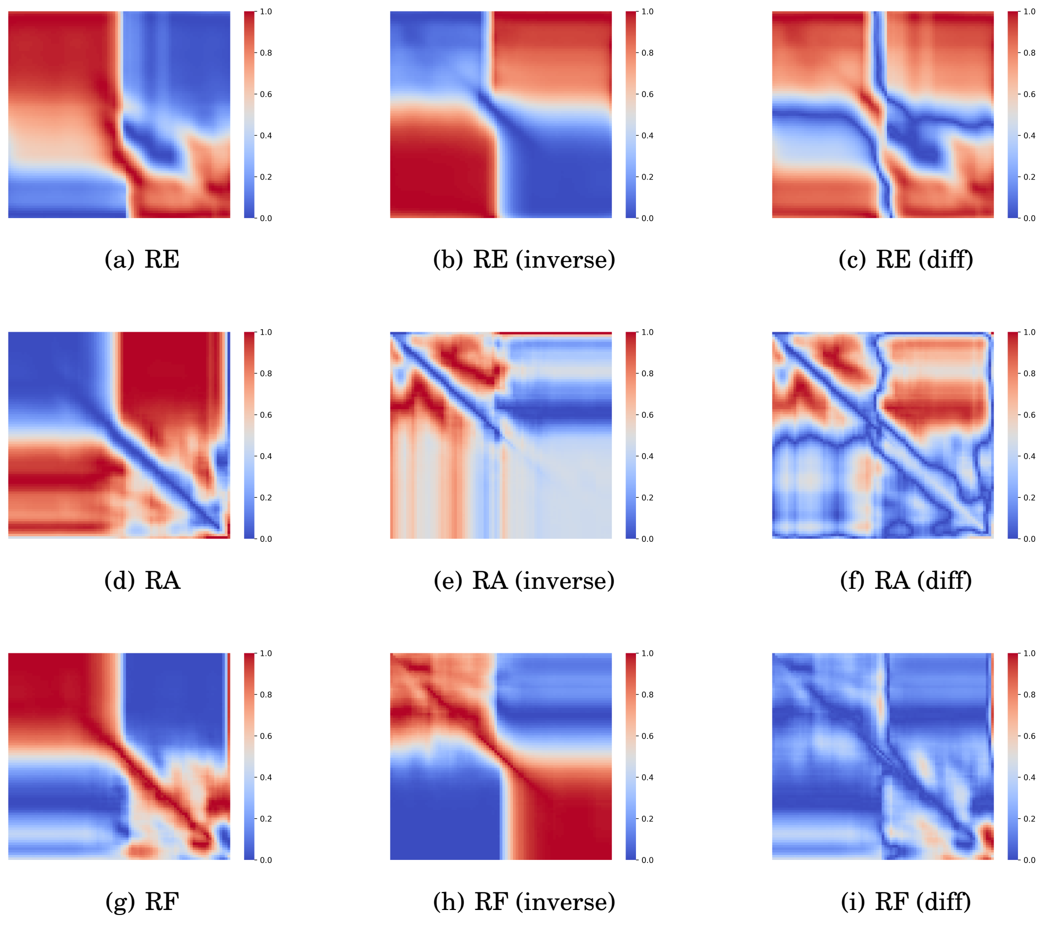


Figure 6.9: Attention distribution heat maps of recurrent-based variants under original and inverted test data in forecasting. RE, RA, and RF mean recurrent embedding, recurrent attention, and recurrent feed-forward, respectively. Diff means difference, resulting from the subtraction between the attention weights under the original and inverted data.

CHAPTER 7

An Explanation Module for Deep Neural Networks for Multivariate Time Series Classification

7.1 Introduction

Currently, deep neural networks have been widely adopted for multivariate time series classification [42] and achieved state-of-the-art performance in various tasks, thanks to the ability to capture complicated, non-linear relations between inputs and outputs [80]. 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 [100] 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 [50, 51] and predict the growth trend by solving a multivariate time series classification 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 [46, 90] call for explainable multivariate time series classification 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 harness the temporal features of the input time series effectively, 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 explanations for classification problems [1, 144], 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 the 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 multivariate time series classification. In this regard, we propose an explanation module that can be seamlessly integrated into deep neural networks to gain the importance of variables in multivariate time series classification. We propose an explanation module that can be plugged into existing popular deep neural networks, such as convolutional neural networks and recurrent neural networks, to infer the importance of variables in multivariate time series classification automatically. We conducted experiments on four benchmark multivariate time series datasets using four variants of convolutional neural network and recurrent neural network 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 multivariate time series classification 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.

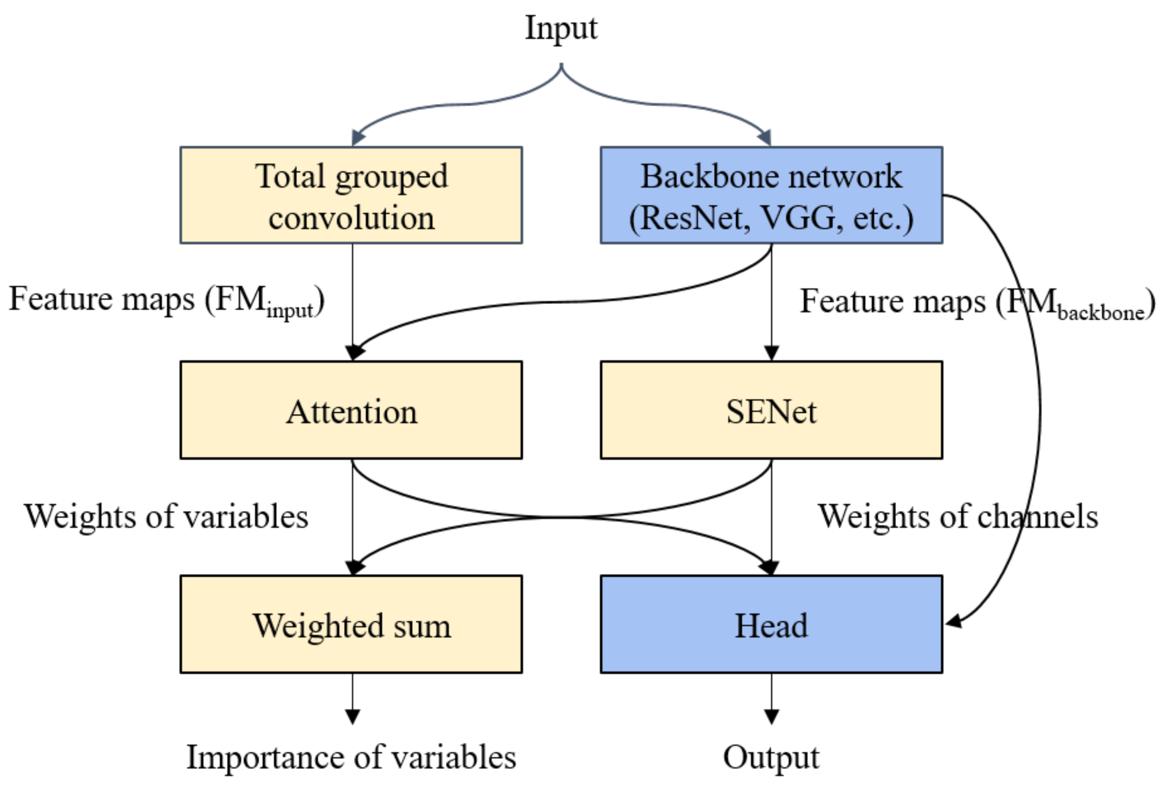


Figure 7.1: The proposed module is pluggable into any existing deep learning model (i.e., *backbone network*) in a backbone-head fashion. Yellow blocks represent our proposed module, and Blue ones stand for the original neural network model. The module obtains the importance of variables by calculating 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.

7.2 Methodology

In a typical multivariate time series classification process (represented by blue blocks in Figure 7.1), the input firstly goes through the backbone (e.g., convolutional neural network or recurrent neural network) 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.

Our proposed module (represented in yellow blocks in Figure 7.1) aims to explore the importance of variables for pluggable explanation in multivariate time series classification. 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 the channel of the input data and does convolutions on each channel. In this manner, the number of filters is equal to the number of 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 **M** is the number of variables, and **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 learned according to the inter-relationship of the channels and the importance of different channels regarding 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 the channels of the $\mathbf{FM}_{backbone}$ $\mathbf{Attn}_{nm}$ as well as the importance of the channels of the $\mathbf{FM}_{backbone}$ $\mathbf{SelfAttn}_{n}$. In the last step, the module does the weighted sum to obtain the importance of the variables as Eq. (7.1):

$$\text{Importance}_{\ m} = \sum_{1}^{N} \text{Attn}_{\ nm} \times \text{Self Attn}_{\ n} \tag{7.1}$$

where $\mathbf{Importance}_{m} \in \mathbb{R}^{M}$. In this way, the importance of the **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 or limitation for combining the proposed module with the existing models to figure out the importance of multivariate time series classification variables.

7.3 Experiments

7.3.1 Datasets

We conducted experiments on four carefully selected public multivariate time series datasets (Table 7.1), which are representative of different sizes and domains. Table 7.1 includes the number of classes, the number of variables for each input sequence, the length of the sequence, and the train-test split ratio. More details are as follows:

- **AREM [35]**: The AREM dataset contains time series sequences recorded by sensors placed in different positions of the body to recognize the activities. The dataset consists of six activity types: cycling, lying, sitting, standing, walking, bending1, and bending2.

- **LP5 [35]**: The 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 [35]**: The 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 [93]**: 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 and 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.

Please add the following required packages to your document preamble:

Table 7.1: Experimental datasets

Dataset	Classes	Number of variables	Sequence length	Train-test ratio
AREM [35]	7	7	480	50:50
LP5 [35]	5	6	15	39:61
ArabicDigits [35]	10	13	93	75:25
Wafer [93]	5	18	214	25:75

7.3.2 Baseline Methods

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

- **ResNet** [55] and **Res2Net** [44]: Popular Convolutional Neural Network-based models. We train ResNet on the AREM dataset and train Res2Net on the 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** [56] and **GRU** [24]: 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 recurrent neural network 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.

7.3.3 Evaluation Procedure

For each model, we followed the given train-test split regulation and first trained 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 varies 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 first trained and evaluated all the methods on the selected datasets and recorded the performances. After that, we sample the noise data from the normal distribution, add it to the variables separately, and test 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 of 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 multivariate time series classification. We tested the model five times on the datasets and calculated the average accuracy and the standard deviation. Since we concentrate on the effectiveness of the proposed module, accuracy suffices to suggest the quality of different methods' results.

7.3.4 Results

7.3.4.1 Performance on real datasets

Table 7.2, Table 7.3, Table 7.4, and Table 7.5 show our experimental results on the four datasets. Our module can be implemented to be combined with various models without much extra effort. 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 for constructing 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 of 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 7.2) show that the classification accuracy is 99.87% on the selected class. The results produced by the module indicate that the 5th and 6th variables are the 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.

7.3.4.2 Validating explanation ability

We tested adding noises to the variables manually, which resulted 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-

Table 7.2: 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 7.3: 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 7.4: 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

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

It is worth noting that the accuracy of all the datasets 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.

7.3.4.3 Impact on performance

Besides explanation ability, our module can improve the performance of the baseline models on the respective datasets. Specifically, Figure 7.2 suggests that all the selected classifiers achieve better classification accuracy on each dataset, indicating the proposed module's superiority. 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.

Table 7.5: 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	

Table 7.6: Training time consumption comparison between the model with the proposed module and without the proposed module on the Wafer dataset

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

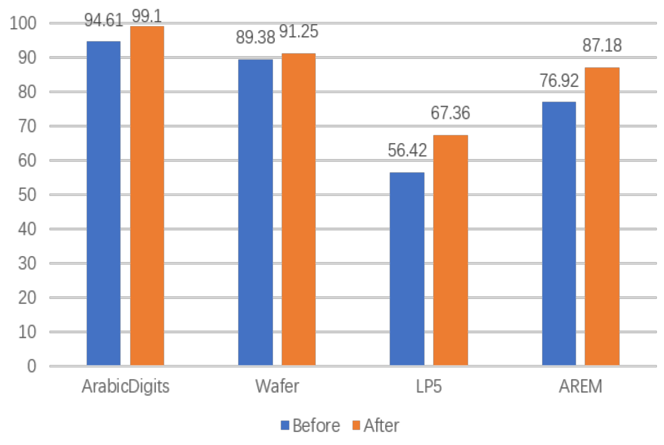
The module helps the original model fuse various feature maps with different levels and improve the classifier's performance. The accuracy given in Figure 7.2 is the average accuracy of all the classes instead of a specific class. Hence, the accuracy differs from the results of the tables shown in the previous contents.

Besides, the proposed module does not significantly cause extra time consumption. To indicate that, we record the training time consumption shown in Table 7.6 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.

7.4 Conclusion

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

Figure 7.2: Accuracy comparison between the original model and the model combined with our proposed module



information of the input. We also provide some tricks for implementing our module.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we introduce how to fully leverage the inherent information of the time series sequences for multivariate time series classification. We propose five methods to address the challenges, including leveraging multi-scale temporal dependencies, adaptive learning strategies for different frequency components, the effectiveness of positional embedding, leveraging both temporal and frequency information, and ensuring the model's explainability. Specifically, the proposed attentional architecture can harness multi-scale temporal dependencies through a hierarchically connected convolutional structure; the proposed dynamic structure can realize adaptive learning strategies through flexible information flow; the dual-stream architecture can generate a new modality containing frequency information from time series to harness both frequency and temporal information for classification; the empirical study on transformer-based models can provide practical suggestions on how to implement positional embedding effectively; and the pluggable explanation module can improve the interoperability of the existing methods. We selected multiple public benchmark datasets for the assessment to demonstrate the superiority of the proposed methods. Specifically, we make the following contributions:

- **Attentional Gated Res2Net**: we propose the Attentional Gated Res2Net using hierarchical residual-like connections to achieve multi-scale receptive fields and capture multi-granular temporal information. We evaluate the model on diverse datasets that contain sequences of various lengths with a wide range of variable numbers. The experimental results show the model outperforms several baselines and state-of-the-art methods by a large margin. We thoroughly investigate the effect of different components and settings on the model's performance and provide hands-on advice on applying our model to a new problem. Our test on plugging the model into the state-of-the-art architecture demonstrates the potential for using our model as a plugin to improve the performance of existing models.

- **Leveraging temporal and frequency information**: we propose a novel dual-stream architecture for accurately classifying multivariate time series sequences. The method leverages the inherent frequency information of the time series data by implementing Short-Time Fourier Transform to obtain the frequency components and their temporal positions. We construct a dual-stream architecture based on ResNet, which can leverage both 1D and 2D representations effectively to classify multivariate time series sequences. We evaluate the proposed model on diverse datasets containing sequences of various lengths and variable numbers. The experimental results show that our method significant outperforms several baseline and state-of-the-art methods. We also comprehensively investigate the effectiveness of the model's components and settings on the performance.

- **Dyformer**: we propose a novel dynamic transformer-based architecture called Dyformer. We first design the hierarchical pooling layer for time series decomposition. We further propose the Dyformer module to achieve adaptive learning strategies for different frequency patterns. We also incorporate the proposed Attentional gated Res2Net to realize the feature-map-wise attention for capturing multi-scale temporal dependencies. We conduct comprehensive evaluations on a wide range of datasets comprising sequences of varying lengths and variable numbers. Our experimental results show that Dyformer achieves better performance compared to multiple baselines and state-of-the-art methods. Furthermore, We explore the influence of various components and configurations on the model's performance. We also demonstrate that through unsupervised pre-training, Dyformer gains an improvement over fully supervised learning on datasets with fewer samples.

- **An empirical study on positional embedding**: we investigate the impact of positional embedding on the vanilla Transformer architecture and six types of Transformer-based variants designed for time series-related tasks. Our experimental results on 30 public classification datasets and 4 forecasting datasets revealed that the positional embedding positively influences the vanilla Transformer architecture in both tasks, while its influence on the variants is negative and input sequence length-dependent for classification and forecasting, respectively. Our analysis of the model's performance and attention distributions under inverted time series data shows that the ability of the variants to capture potential position information resulted in focusing on the spurious sequential order information caused by the inverted sequential order, leading to a more drastic degradation in their performance compared to the vanilla transformer. Our findings suggest that position-sensitive layers in Transformer-based architectures can encode the potential position information, likely making positional embedding unnecessary in classification, while its utilization may depend on the length of the input time series in forecasting. Additionally, the convolutional embedding and the recurrent embedding outperform other variants, making them preferable when considering position-sensitive layers in the Transformer-based architecture.

- **Explanation module**: 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 to enhance the existing deep learning-based classifiers' interpretability. Besides, the module can improve the model's performance as it benefits the models by facilitating leveraging the temporal information of the input.

8.2 Future Work

We consider future research from the following perspectives:

- We plan to explore the interpretability of our proposed method for leveraging both temporal and frequency information through visualization technologies for convolutional neural networks from the computer vision field. As such, the proposed method can figure out the importance of the time series slots for obtaining more convincing results. Additionally, we plan to extend our work to more time series-related tasks, such as time series imputation, forecasting, and abnormal detection. As these tasks also require capturing multi-range temporal information, we believe the proposed method can realize good performances.

- We plan to refine the explanation module to make it feasible to figure out the importance of both temporal aspects and variable aspects. As some tasks require importance from both temporal and variable perspectives, this can enlarge the application scope of the proposed methods.
- We plan to propose an unsupervised time series representation method to avoid the manually designed positive-pair selection strategy, which may bring inductive bias that may negatively impact the generalization. We will evaluate it using various downstream tasks, including classification, forecasting, etc.
- We aim to improve the efficiency of the proposed Dyformer. Dyformer's reliance on parallel construction of multi-paths results in a time consumption profile that is heavily contingent upon the training speed of the path that activates the maximum number of Dyformer blocks. Consequently, the activation of fewer Dyformer blocks along certain paths may not yield a significant acceleration in the overall training speed. This trade-off between training speed and performance enhancement is a notable limitation of Dyformer, as it prioritizes the latter at the expense of the former. This will make Dyformer applicable for computational resource-limited devices.

Bibliography

[1] M. Ancona, C. Oztireli, and M. Gross, *Explaining deep neural networks with a polynomial time algorithm for shapley value approximation*, in International Conference on Machine Learning, PMLR, 2019, pp. 272–281.

[2] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, *Vivit: A video vision transformer*, arXiv preprint arXiv:2103.15691, (2021).

[3] S. Aydin, *Deep learning classification of neuro-emotional phase domain complexity levels induced by affective video film clips*, IEEE Journal of Biomedical and Health Informatics, 24 (2019), pp. 1695–1702.

[4] S. Aydn, *Cross-validated adaboost classification of emotion regulation strategies identified by spectral coherence in resting-state*, Neuroinformatics, (2021), pp. 1–13.

[5] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473, (2014).

[6] L. Bai, L. Yao, X. Wang, S. S. Kanhere, and Y. Xiao, *Prototype similarity learning for activity recognition*, in Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2020, pp. 649–661.

[7] T. Bai and P. Tahmasebi, *Attention-based lstm-fcn for earthquake detection and location*, Geophysical Journal International, 228 (2022), pp. 1568–1576.

[8] M. G. Baydogan, G. Runger, and E. Tuv, *A bag-of-features framework to classify time series*, IEEE transactions on pattern analysis and machine intelligence, 35 (2013), pp. 2796–2802.

[9] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: A review and new perspectives*, IEEE transactions on pattern analysis and machine intelligence, 35 (2013), pp. 1798–1828.

[10] Y. BENGIO, Y. LECUN, ET AL., *Scaling learning algorithms towards ai*, Large-scale kernel machines, 34 (2007), pp. 1–41.

[11] S. BIRNBAUM, V. KULESHOV, Z. ENAM, P. W. W. KOH, AND S. ERMON, *Temporal film: Capturing long-range sequence dependencies with feature-wise modulations.*, Advances in Neural Information Processing Systems, 32 (2019).

[12] R. N. BRACEWELL AND R. N. BRACEWELL, *The Fourier transform and its applications*, vol. 31999, McGraw-Hill New York, 1986.

[13] A. BULATOV, Y. KURATOV, AND M. S. BURTSEV, *Recurrent memory transformer*, arXiv preprint arXiv:2207.06881, (2022).

[14] M. BURCHI AND R. TIMOFTE, *Audio-visual efficient conformer for robust speech recognition*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2023, pp. 2258–2267.

[15] K. L. CABALLERO BARAJAS AND R. AKELLA, *Dynamically modeling patient's health state from electronic medical records: A time series approach*, in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, pp. 69–78.

[16] S. CHAMBON, M. N. GALTIER, P. J. ARNAL, G. WAINRIB, AND A. GRAMFORT, *A deep learning architecture for temporal sleep stage classification using multivariate and multimodal time series*, IEEE Transactions on Neural Systems and Rehabilitation Engineering, 26 (2018), pp. 758–769.

[17] C. CHATFIELD, *The analysis of time series: an introduction*, Chapman and Hall/CRC, 2003.

[18] A. CHATTOPADHAY, A. SARKAR, P. HOWLADER, AND V. N. BALASUBRAMANIAN, *Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks*, in 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2018, pp. 839–847.

[19] Z. CHE, S. PURUSHOTHAM, K. CHO, D. SONTAG, AND Y. LIU, *Recurrent neural networks for multivariate time series with missing values*, Scientific reports, 8 (2018), p. 6085.

[20] K. Chen, R. Wang, M. Utiyama, and E. Sumita, *Recurrent positional embedding for neural machine translation*, in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 1361–1367.

[21] Q. Chen, J. Xu, and V. Koltun, *Fast image processing with fully-convolutional networks*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2497–2506.

[22] Z. Chen, D. Chen, X. Zhang, Z. Yuan, and X. Cheng, *Learning graph structures with transformer for multivariate time series anomaly detection in iot*, IEEE Internet of Things Journal, (2021).

[23] Z. Chen, Q. Ma, and Z. Lin, *Time-aware multi-scale rnns for time series modeling.*, in IJCAI, 2021, pp. 2285–2291.

[24] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, arXiv preprint arXiv:1406.1078, (2014).

[25] E. Choi, M. T. Bahadori, J. A. Kulas, A. Schuetz, W. F. Stewart, and J. Sun, *Retain: An interpretable predictive model for healthcare using reverse time attention mechanism*, arXiv preprint arXiv:1608.05745, (2016).

[26] F. Chollet, *Xception: Deep learning with depthwise separable convolutions*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.

[27] R. R. Chowdhury, X. Zhang, J. Shang, R. K. Gupta, and D. Hong, *Tarnet: Task-aware reconstruction for time-series transformer*, in Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022, pp. 212–220.

[28] Z. Cui, W. Chen, and Y. Chen, *Multi-scale convolutional neural networks for time series classification*, arXiv preprint arXiv:1603.06995, (2016).

[29] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, and Ratanamahatana, *The ucr time series classification archive*, October 2018.

[30] A. DEMPSTER, F. PETITJEAN, AND G. I. WEBB, *Rocket: exceptionally fast and accurate time series classification using random convolutional kernels*, Data Mining and Knowledge Discovery, 34 (2020), pp. 1454–1495.

[31] A. DEMPSTER, D. F. SCHMIDT, AND G. I. WEBB, *Minirocket: A very fast (almost) deterministic transform for time series classification*, in Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining, 2021, pp. 248–257.

[32] H. DENG, G. RUNGER, E. TUV, AND M. VLADIMIR, *A time series forest for classification and feature extraction*, Information Sciences, 239 (2013), pp. 142–153.

[33] J. DEVLIN, M.-W. CHANG, K. LEE, AND K. TOUTANOVA, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805, (2018).

[34] M. A. DI GANGI, M. NEGRI, R. CATTONI, R. DESSI, AND M. TURCHI, *Enhancing transformer for end-to-end speech-to-text translation*, in Proceedings of Machine Translation Summit XVII: Research Track, 2019, pp. 21–31.

[35] D. DUA AND C. GRAFF, *UCI machine learning repository*, 2017.

[36] S. DUAN, L. WU, B. XUE, A. LIU, R. QIAN, AND X. CHEN, *A hybrid multimodal fusion framework for semg-acc-based hand gesture recognition*, IEEE Sensors Journal, 23 (2023), pp. 2773–2782.

[37] E. ELDELE, M. RAGAB, Z. CHEN, M. WU, C. K. KWOH, X. LI, AND C. GUAN, *Time-series representation learning via temporal and contextual contrasting*, in Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, International Joint Conferences on Artificial Intelligence Organization, 8 2021, pp. 2352–2359.

[38] N. ELSAYED, A. S. MAIDA, AND M. BAYOUMI, *Deep gated recurrent and convolutional network hybrid model for univariate time series classification*, arXiv preprint arXiv:1812.07683, (2018).

[39] ——, *Gated recurrent neural networks empirical utilization for time series classification*, in 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber,

Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2019, pp. 1207–1210.

[40] C. Fang and C. Wang, *Time series data imputation: A survey on deep learning approaches*, arXiv preprint arXiv:2011.11347, (2020).

[41] M. J. Fard, A. K. Pandya, R. B. Chinnam, M. D. Klein, and R. D. Ellis, *Distance-based time series classification approach for task recognition with application in surgical robot autonomy*, The International Journal of Medical Robotics and Computer Assisted Surgery, 13 (2017), p. e1766.

[42] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, *Deep learning for time series classification: a review*, Data mining and knowledge discovery, 33 (2019), pp. 917–963.

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

[44] S. Gao, M.-M. Cheng, K. Zhao, X.-Y. Zhang, M.-H. Yang, and P. H. Torr, *Res2net: A new multi-scale backbone architecture*, IEEE transactions on pattern analysis and machine intelligence, (2019).

[45] W. Gerald, C. Liu, D. Sahoo, A. Kumar, and S. Hoi, *Etsformer: Exponential smoothing transformers for time-series forecasting*, arXiv preprint arXiv:2202.01381, (2022).

[46] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, *Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals*, circulation, 101 (2000), pp. e215–e220.

[47] D. Griffin and J. Lim, *Signal estimation from modified short-time fourier transform*, IEEE Transactions on acoustics, speech, and signal processing, 32 (1984), pp. 236–243.

[48] S. Gudmundsson, T. P. Runarsson, and S. Sigurdsson, *Support vector machines and dynamic time warping for time series*, in 2008 IEEE International

Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 2772–2776.

[49] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, et al., *Conformer: Convolution-augmented transformer for speech recognition*, arXiv preprint arXiv:2005.08100, (2020).

[50] B. Guo, Y. Mu, F. Wang, and S. Dong, *Effect of periodic light color change on the molting frequency and growth of litopenaeus vannamei*, Aquaculture, 362 (2012), pp. 67–71.

[51] B. Guo, F. Wang, Y. Li, and S. Dong, *Effect of periodic light intensity change on the molting frequency and growth of litopenaeus vannamei*, Aquaculture, 396 (2013), pp. 66–70.

[52] M. Han and X. Liu, *Feature selection techniques with class separability for multivariate time series*, Neurocomputing, 110 (2013), pp. 29–34.

[53] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, *Dynamic neural networks: A survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 44 (2021), pp. 7436–7456.

[54] F. J. Harris, *On the use of windows for harmonic analysis with the discrete fourier transform*, Proceedings of the IEEE, 66 (1978), pp. 51–83.

[55] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[56] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural computation, 9 (1997), pp. 1735–1780.

[57] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural networks, 4 (1991), pp. 251–257.

[58] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, arXiv preprint arXiv:1704.04861, (2017).

[59] T.-Y. Hsieh, S. Wang, Y. Sun, and V. Honavar, *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, 2021, pp. 607–615.

[60] J. Hu, L. Shen, and G. Sun, *Squeeze-and-excitation networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7132–7141.

[61] J. Hu and W. Zheng, *A deep learning model to effectively capture mutation information in multivariate time series prediction*, Knowledge-Based Systems, 203 (2020), p. 106139.

[62] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, *Hitanomaly: Hierarchical transformers for anomaly detection in system log*, IEEE Transactions on Network and Service Management, 17 (2020), pp. 2064–2076.

[63] Z. Huang, P. Xu, D. Liang, A. Mishra, and B. Xiang, *Trans-blstm: Transformer with bidirectional lstm for language understanding*, arXiv preprint arXiv:2003.07000, (2020).

[64] R. S. Hussain, M. A. Quddus, M. P. Enoch, K. D. Ruikar, N. Brien, and D. Gartside, *Time series analysis of local authority policy interventions on highway works durations*, in Proceedings of the Institution of Civil Engineers-Transport, vol. 174, Thomas Telford Ltd, 2021, pp. 283–293.

[65] D. Hutchins, I. Schlag, Y. Wu, E. Dyer, and B. Neyshabur, *Block-recurrent transformers*, arXiv preprint arXiv:2203.07852, (2022).

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

[67] E. Jang, S. Gu, and B. Poole, *Categorical reparameterization with gumbel-softmax*, arXiv preprint arXiv:1611.01144, (2016).

[68] H. Jiang, L. Liu, and C. Lian, *Multi-modal fusion transformer for multivariate time series classification*, in 2022 14th International Conference on Advanced Computational Intelligence (ICACI), IEEE, 2022, pp. 284–288.

[69] A. KAMPOURAKI, G. MANIS, AND C. NIKOU, *Heartbeat time series classification with support vector machines*, IEEE Transactions on Information Technology in Biomedicine, 13 (2008), pp. 512–518.

[70] F. KARIM, S. MAJUMDAR, H. DARABI, AND S. CHEN, *Lstm fully convolutional networks for time series classification*, IEEE access, 6 (2017), pp. 1662–1669.

[71] F. KARIM, S. MAJUMDAR, H. DARABI, AND S. HARFORD, *Multivariate lstm-fcns for time series classification*, Neural Networks, 116 (2019), pp. 237–245.

[72] M. KHAN, H. WANG, A. RIAZ, A. ELFATYANY, AND S. KARIM, *Bidirectional lstm-rnn-based hybrid deep learning frameworks for univariate time series classification*, The Journal of Supercomputing, 77 (2021), pp. 7021–7045.

[73] B. KILIÇ AND S. AYDIN, *Classification of contrasting discrete emotional states indicated by eeg based graph theoretical network measures*, Neuroinformatics, (2022), pp. 1–15.

[74] Y. KIM, J. SA, Y. CHUNG, D. PARK, AND S. LEE, *Resource-efficient pet dog sound events classification using lstm-fcn based on time-series data*, Sensors, 18 (2018), p. 4019.

[75] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[76] B. H. D. KOH, C. L. P. LIM, H. RAHIMI, W. L. WOO, AND B. GAO, *Deep temporal convolution network for time series classification*, Sensors, 21 (2021), p. 603.

[77] A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems, 25 (2012), pp. 1097–1105.

[78] D.-H. KWON, J.-B. KIM, J.-S. HEO, C.-M. KIM, AND Y.-H. HAN, *Time series classification of cryptocurrency price trend based on a recurrent lstm neural network*, Journal of Information Processing Systems, 15 (2019), pp. 694–706.

[79] T. LE NGUYEN, S. GSPONER, I. ILIE, M. O,ÄÔREILLY, AND G. IFRIM, *Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations*, Data mining and knowledge discovery, 33 (2019), pp. 1183–1222.

[80] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, nature, 521 (2015), pp. 436–444.

[81] G. Li, B. Choi, J. Xu, S. S. Bhowmick, K.-P. Chun, and G. L.-H. Wong, *Shapenet: A shapelet-neural network approach for multivariate time series classification*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, 2021, pp. 8375–8383.

[82] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, *Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting*, Advances in neural information processing systems, 32 (2019).

[83] W. Li, Z. Zhang, and Z. Liu, *Action recognition based on a bag of 3d points*, in 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops, IEEE, 2010, pp. 9–14.

[84] B. Lim, S. Ö. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, International Journal of Forecasting, 37 (2021), pp. 1748–1764.

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

[86] C.-L. Liu, W.-H. Hsaio, and Y.-C. Tu, *Time series classification with multivariate convolutional neural network*, IEEE Transactions on Industrial Electronics, 66 (2018), pp. 4788–4797.

[87] M. Liu, S. Ren, S. Ma, J. Jiao, Y. Chen, Z. Wang, and W. Song, *Gated transformer networks for multivariate time series classification*, arXiv preprint arXiv:2103.14438, (2021).

[88] S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. X. Liu, and S. Dustdar, *Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting*, in International conference on learning representations, 2021.

[89] Y. Liu, Z. Xu, G. Wang, K. Chen, B. Li, X. Tan, J. Li, L. He, and S. Zhao, *Delightfultts: The microsoft speech synthesis system for blizzard challenge 2021*, arXiv preprint arXiv:2110.12612, (2021).

[90] P. MAJOR AND E. A. THIELE, *Seizures in children: Laboratory*, Pediatrics in review, 28 (2007), p. 405.

[91] M. MIDDLEHURST, J. LARGE, AND A. BAGNALL, *The canonical interval forest (cif) classifier for time series classification*, in 2020 IEEE International Conference on Big Data (Big Data), IEEE, 2020, pp. 188–195.

[92] M. MÜLLER, *Dynamic time warping*, Information retrieval for music and motion, (2007), pp. 69–84.

[93] R. T. OLSZEWSKI, *Bobski's world*.
[EB/OL], 2012.
`http://www.cs.cmu.edu/~bobski/`.

[94] Z. PAN, J. CAI, AND B. ZHUANG, *Fast vision transformers with hilo attention*, arXiv preprint arXiv:2205.13213, (2022).

[95] H. PURWINS, B. LI, T. VIRTANEN, J. SCHLÜTER, S.-Y. CHANG, AND T. SAINATH, *Deep learning for audio signal processing*, IEEE Journal of Selected Topics in Signal Processing, 13 (2019), pp. 206–219.

[96] J. QIU, S. R. JAMMALAMADAKA, AND N. NING, *Multivariate bayesian structural time series model.*, J. Mach. Learn. Res., 19 (2018), pp. 2744–2776.

[97] Y. REN, L. LI, X. YANG, AND J. ZHOU, *Autotransformer: Automatic transformer architecture design for time series classification*, in Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2022, pp. 143–155.

[98] A. G. ROY, N. NAVAB, AND C. WACHINGER, *Concurrent spatial and channel ‚Äòsqueeze & excitation‚Äôin fully convolutional networks*, in Medical Image Computing and Computer Assisted Intervention–MICCAI 2018: 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part I, Springer, 2018, pp. 421–429.

[99] P. SCHÄFER AND U. LESER, *Multivariate time series classification with weasel+ muse*, arXiv preprint arXiv:1711.11343, (2017).

[100] J. SCHMIDHUBER, *Deep learning in neural networks: An overview*, Neural networks, 61 (2015), pp. 85–117.

[101] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, *Grad-cam: Visual explanations from deep networks via gradient-based localization*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 618–626.

[102] P. Senin and S. Malinchik, *Sax-vsm: Interpretable time series classification using sax and vector space model*, in 2013 IEEE 13th international conference on data mining, IEEE, 2013, pp. 1175–1180.

[103] L. Shen and Y. Wang, *Tcct: Tightly-coupled convolutional transformer on time series forecasting*, Neurocomputing, 480 (2022), pp. 131–145.

[104] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, *Ts-chief: a scalable and accurate forest algorithm for time series classification*, Data Mining and Knowledge Discovery, 34 (2020), pp. 742–775.

[105] R. K. Srivastava, K. Greff, and J. Schmidhuber, *Highway networks*, arXiv preprint arXiv:1505.00387, (2015).

[106] D. Sundararajan, *The discrete Fourier transform: theory, algorithms and applications*, World Scientific, 2001.

[107] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[108] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826.

[109] W. Tang, G. Long, L. Liu, T. Zhou, M. Blumenstein, and J. Jiang, *Omni-scale cnns: a simple and effective kernel size configuration for time series classification*, in International Conference on Learning Representations, 2021.

[110] Y. Tao, L. Ma, W. Zhang, J. Liu, W. Liu, and Q. Du, *Hierarchical attention-based recurrent highway networks for time series prediction*, arXiv preprint arXiv:1806.00685, (2018).

[111] S. Tonekaboni, D. Eytan, and A. Goldenberg, *Unsupervised representation learning for time series with temporal neighborhood coding*, in International Conference on Learning Representations, 2020.

[112] T. M. Tran, X.-M. T. Le, H. T. Nguyen, and V.-N. Huynh, *A novel non-parametric method for time series classification based on k-nearest neighbors and dynamic time warping barycenter averaging*, Engineering Applications of Artificial Intelligence, 78 (2019), pp. 173–185.

[113] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in neural information processing systems, 2017, pp. 5998–6008.

[114] A. Veit and S. Belongie, *Convolutional networks with adaptive inference graphs*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 3–18.

[115] R. Wan, S. Mei, J. Wang, M. Liu, and F. Yang, *Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting*, Electronics, 8 (2019), p. 876.

[116] H. Wang, X. Wu, Z. Huang, and E. P. Xing, *High-frequency component helps explain the generalization of convolutional neural networks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 8684–8694.

[117] J. Wang, Z. Guo, C. Yang, X. Li, and Z. Cui, *Multi-scale hybrid fusion network for mandarin audio-visual speech recognition*, in 2023 IEEE International Conference on Multimedia and Expo (ICME), IEEE, 2023, pp. 642–647.

[118] X. Wang, H. Liu, J. Du, Z. Yang, and X. Dong, *Clformer: Locally grouped auto-correlation and convolutional transformer for long-term multivariate time series forecasting*, Engineering Applications of Artificial Intelligence, 121 (2023), p. 106042.

[119] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, *Skipnet: Learning dynamic routing in convolutional networks*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 409–424.

[120] Y. Wang, Z. Zhang, L. Feng, Y. Ma, and Q. Du, *A new attention-based cnn approach for crop mapping using time series sentinel-2 images*, Computers and electronics in agriculture, 184 (2021), p. 106090.

[121] Z. Wang, Y. Ma, Z. Liu, and J. Tang, *R-transformer: Recurrent neural network enhanced transformer*, arXiv preprint arXiv:1907.05572, (2019).

[122] Z. Wang, W. Yan, and T. Oates, *Time series classification from scratch with deep neural networks: A strong baseline*, in 2017 International joint conference on neural networks (IJCNN), IEEE, 2017, pp. 1578–1585.

[123] S. Wei, Y. Zhang, and H. Liu, *A multimodal multilevel converged attention network for hand gesture recognition with hybrid semg and a-mode ultrasound sensing*, IEEE Transactions on Cybernetics, (2022).

[124] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, *Transformers in time series: A survey*, arXiv preprint arXiv:2202.07125, (2022).

[125] G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. Hoi, *Cost: Contrastive learning of disentangled seasonal-trend representations for time series forecasting*, arXiv preprint arXiv:2202.01575, (2022).

[126] S. Woo, J. Park, J.-Y. Lee, and I. So Kweon, *Cbam: Convolutional block attention module*, in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 3–19.

[127] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, *Timesnet: Temporal 2d-variation modeling for general time series analysis*, arXiv preprint arXiv:2210.02186, (2022).

[128] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, *Cvt: Introducing convolutions to vision transformers*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 22–31.

[129] H. Wu, J. Xu, J. Wang, and M. Long, *Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting*, Advances in Neural Information Processing Systems, 34 (2021), pp. 22419–22430.

[130] P. Wu, H. Liu, X. Li, T. Fan, and X. Zhang, *A novel lip descriptor for audio-visual keyword spotting based on adaptive decision fusion*, IEEE Transactions on Multimedia, 18 (2016), pp. 326–338.

[131] R. Xi, M. Li, M. Hou, M. Fu, H. Qu, D. Liu, and C. R. Haruna, *Deep dilation on multimodality time series for human activity recognition*, IEEE Access, 6 (2018), pp. 53381–53396.

[132] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, *Aggregated residual transformations for deep neural networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[133] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, *Aggregated residual transformations for deep neural networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1492–1500.

[134] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, *Deebert: Dynamic early exiting for accelerating bert inference*, in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 2246–2251.

[135] Z. Xing, J. Pei, P. S. Yu, and K. Wang, *Extracting interpretable features for early classification on time series*, in Proceedings of the 2011 SIAM international conference on data mining, SIAM, 2011, pp. 247–258.

[136] J. Xu, J. Wang, M. Long, et al., *Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting*, Advances in Neural Information Processing Systems, 34 (2021).

[137] P. Xu, D. Kumar, W. Yang, W. Zi, K. Tang, C. Huang, J. C. K. Cheung, S. J. Prince, and Y. Cao, *Optimizing deeper transformers on small datasets*, in ACL/IJCNLP (1), 2021.

[138] R. Yan, F. Li, D. D. Zhou, T. Ristaniemi, and F. Cong, *Automatic sleep scoring: A deep learning architecture for multi-modality time series*, Journal of neuroscience methods, 348 (2021), p. 108971.

[139] C. Yang, M. Jiang, Z. Guo, and Y. Liu, *Gated res2net for multivariate time series analysis*, in 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–7.

[140] C. Yang, W. Jiang, and Z. Guo, *Time series data classification based on dual path cnn-rnn cascade network*, IEEE Access, 7 (2019), pp. 155304–155312.

[141] L. Ye and E. Keogh, *Time series shapelets: a novel technique that allows accurate, interpretable and fast classification*, Data mining and knowledge discovery, 22 (2011), pp. 149–182.

[142] A. Y. Yildiz, E. Koç, and A. Koç, *Multivariate time series imputation with transformers*, IEEE Signal Processing Letters, 29 (2022), pp. 2517–2521.

[143] H. Yoon and C. Shahabi, *Feature subset selection on multivariate time series with extremely large spatial features*, in Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06), IEEE, 2006, pp. 337–342.

[144] J. Yoon, J. Jordon, and M. van der Schaar, *Invase: Instance-wise variable selection using neural networks*, in International Conference on Learning Representations, 2018.

[145] Y. Yu, X. Zeng, X. Xue, and J. Ma, *Lstm-based intrusion detection system for vanets: a time series classification approach to false message detection*, IEEE Transactions on Intelligent Transportation Systems, 23 (2022), pp. 23906–23918.

[146] Y. Yuan and L. Lin, *Self-supervised pretraining of transformers for satellite image time series classification*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 14 (2020), pp. 474–487.

[147] Z. Yue, Y. Wang, J. Duan, T. Yang, C. Huang, Y. Tong, and B. Xu, *Ts2vec: Towards universal representation of time series*, arXiv preprint arXiv:2106.10466, (2021).

[148] ——, *Ts2vec: Towards universal representation of time series*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 8980–8987.

[149] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, in European conference on computer vision, Springer, 2014, pp. 818–833.

[150] A. Zeng, M. Chen, L. Zhang, and Q. Xu, *Are transformers effective for time series forecasting?*, arXiv preprint arXiv:2205.13504, (2022).

[151] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, *A transformer-based framework for multivariate time series representation*

learning, in Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 2114–2124.

[152] D. Zhang, W. Zuo, D. Zhang, and H. Zhang, *Time series classification using support vector machine with gaussian elastic metric kernel*, in 2010 20th International Conference on Pattern Recognition, IEEE, 2010, pp. 29–32.

[153] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha, et al., *Resnest: Split-attention networks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 2736–2746.

[154] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha, M. Li, and A. Smola, *Resnest: Split-attention networks*, 2020.

[155] X. Zhang, Y. Gao, J. Lin, and C.-T. Lu, *Tapnet: Multivariate time series classification with attentional prototypical network*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 6845–6852.

[156] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, *Time series classification using multi-channels deep convolutional neural networks*, in International conference on web-age information management, Springer, 2014, pp. 298–310.

[157] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, *Informer: Beyond efficient transformer for long sequence time-series forecasting*, in Proceedings of AAAI, 2021.

[158] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, *Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting*, arXiv preprint arXiv:2201.12740, (2022).

[159] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei, *Bert loses patience: Fast and robust inference with early exit*, Advances in Neural Information Processing Systems, 33 (2020), pp. 18330–18341.

[160] C. Zhu, W. Ping, C. Xiao, M. Shoeybi, T. Goldstein, A. Anandkumar, and B. Catanzaro, *Long-short transformer: Efficient transformers for language and vision*, Advances in Neural Information Processing Systems, 34 (2021), pp. 17723–17736.