RNTS: Robust Neural Temporal Search for Time Series Classification

Zhiwen Xiao[1], Xin Xu[2], Huanlai Xing[1,*], Rong Qu[3], Fuhong Song[1], and Bowen Zhao[1]
[1]School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China
[2]School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221166, China
[3]The University of Nottingham, Nottingham NG7 2RD 455356, UK
[*]Corresponding author: hxx@home.swjtu.edu.cn

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9534392

***Abstract*—Over the years, a large number of deep learning algorithms have been developed for time series classification (TSC). These algorithms were usually invented by researchers with prior knowledge and experience. However, it is a critical challenge for beginners to design decent structures to address various TSC problems. To this end, we propose a robust neural temporal search (RNTS) framework for identifying the relationships and features in TSC data, which mainly contains a temporal search network and an attentional LSTM network. To be specific, inspired by the idea of neural architecture search (NAS), the temporal search network automatically transforms its structure for each dataset according to its characteristics, responsible for extracting basic features. The attentional LSTM network is used to explore the complex shapelets and relationships the former may ignore. Experimental results demonstrate that RNTS achieves the best overall performance on 24 standard datasets selected from the UCR 2018 archive, in terms of three measures based on the top-1 accuracy, compared with a number of state-of-the-art approaches.**

***Index Terms*—Data mining, time series classification, deep learning, neural architecture search, convolutional neural network, LSTM**

I. Introduction

Time series data have been widely used in a variety of real-world applications, for example, electrocardiograph (ECG) detection [1], traffic prediction [2], energy detection [3], optical data analysis [4] and grid fault prediction [5]. To well exploit time series data relies on accurate recognition, usually offered by an efficient algorithm. In the past decades, time series classification has attracted increasingly more research attention and hundreds of deep learning algorithms have been proposed [6].

However, for robust time series classification, it is quite challenging for deep learning algorithms to explore hidden and representative shapelets from input data [6]. These algorithms can roughly be classified into two categories, namely single-network-based models and dual-network-based models. The former pays more attention to mining basic regularizations and representation hierarchy of the data, without performance guarantee. For example, InceptionTime [7] and OS-CNN [8] are not stable as they cannot always achieve excellent performance on different datasets. The latter consists of a feature extraction network and a relation extraction network, where feature extraction focuses on shallow and simple features, and relation extraction explores the underlying and complex shapelets and relationships hidden in data. Algorithms based on dual-network-based models make use of both networks, successfully adapting for a number of standard datasets, such as fcn-lstm-based networks [9], transformer-based models [10] and RTFN-based models [11].

Previously, in order to obtain a decent neural architecture, heavy intervention from professionals with prior knowledge and experience was necessary, no matter which model, i.e. single-network-based or dual-network-based, was used [12]. It is, however, challenging for beginners to re-design a given neural architecture so as to improve its performance. To this end, the neural architecture search (NAS) technique was introduced, responsible for automatically generating an appropriate neural architecture for a specific task, e.g. few-shot image classification [12]. Fortunately, NAS requires limited computing resources and as little human intervention as possible. This technique aims at striking a balance between overall performance and time efficiency [12] [13].

MetaQNN [14] and NAS-RL [15], as two pioneering work in NAS, use reinforcement learning (RL) methods to construct promising neural networks, ensuring high accuracy for image classification tasks. In general, a RL-based NAS approach leads to heavy consumption of computing resources, e.g., Real et al. [16] and Zoph et al. [17] required 2,000 and 3,150 GPU days to obtain excellent networks on CIFAR10 and ImageNet, respectively. However, for ordinary researchers without enough computing resources at hand, the cost incurred by applying RL-based approaches to design a decent neural architecture might not be affordable. A considerable amount of research work has thus been dedicated to reducing the problem complexity while improving the architecture construction efficiency from a number of aspects, including search space structure [18] and weight sharing/inheritance [19]. NAS has been adopted in various application domains, such as object detection [20] and semantic segmentation [21].

On the other hand, time series data are usually collected from various real-world applications. Each time series dataset has its specific characteristics, e.g. length, variance, etc., which may be significantly different from others. Besides, the dual-network-based model, one of the most commonly used methods for time series classification [10], uses an identical feature

extraction network to mine features for different datasets. This is obviously not effective as the characteristics of these datasets might be completely different. Instead, it is more reasonable to design a network architecture tailor-made for each dataset's requirements. This is because it is easier and more efficient for a problem-specific network architecture to extract more features and relationships from a certain dataset, rather than multiple datasets with different characteristics. Nevertheless, this idea has not drawn enough attention from the time series data mining community [22].

This paper proposes an efficient architecture search approach for time series classification, called robust neural temporal search (RNTS), which consists of a temporal search network and an attentional LSTM network. To be specific, based on the idea of NAS, the former as a feature extraction network, automatically selects its building blocks to extract abundant basic features from the input data, according to the actual requirements of each dataset. The latter as a relation extraction network [11], is predetermined and pays attention to the complex and hidden relationships the former may miss. By adopting the gradient descent method, RNTS is able to search for a decent neural architecture in a continuous search space. Compared with RL-based approaches, RNTS has smaller search space to explore, thus consumes less computing resources, e.g. our experiments were run on a desktop with a single GTX-1080ti GPU with 11G RAM. Our main contributions are summarized as follows:

- Different from traditional time series classification models that use a fixed neural network for all datasets, our RNTS adaptively constructs a proper temporal search network for each dataset according to its characteristics/requirements. Possible building blocks include convolutional, residual and direct connection cells.
- Different from DARTS that regards validation loss as the only objective for minimization [23], RNTS evaluates an architecture by not only the validation loss, but also the validation accuracy and number of parameters. This helps to strike a balance between accuracy and model complexity.
- The experimental results demonstrate that RNTS outperforms a number of state-of-the-art algorithms on 24 well known datasets selected in the UCR 2018 archive, in terms of the mean accuracy, 'win'/'tie'/'lose' measure, and AVG_rank, which are all based on the top-1 accuracy. This shows the effectiveness of RNTS in addressing time series classification problems.

II. RNTS

This section first overviews the proposed framework, and then describes the temporal search network, its search space, basic computation cells, and constraints and complexity. Later on, the attentional LSTM is briefly reviewed. Finally, the validation and weight update and score calculation of RNTS are introduced.

A. Overview

The overview of RNTS is illustrated in Fig. 1(a). There are primarily two parts, namely the temporal search network for feature extraction and the attentional LSTM network for identifying relationships in the data. Based on a set of basic computation cells, the former transforms itself to adapt to each dataset according to its characteristics, so as to mine as many basic features as possible from the input data. In this way, different feature extraction networks are automatically generated to handle different datasets. The latter, on the other hand, is predetermined and responsible for exploring the weak and in-depth connections and shapelets that might be extremely difficult for the former to discover. The two networks above complement each other to obtain sufficient features.

B. Temporal Search Network

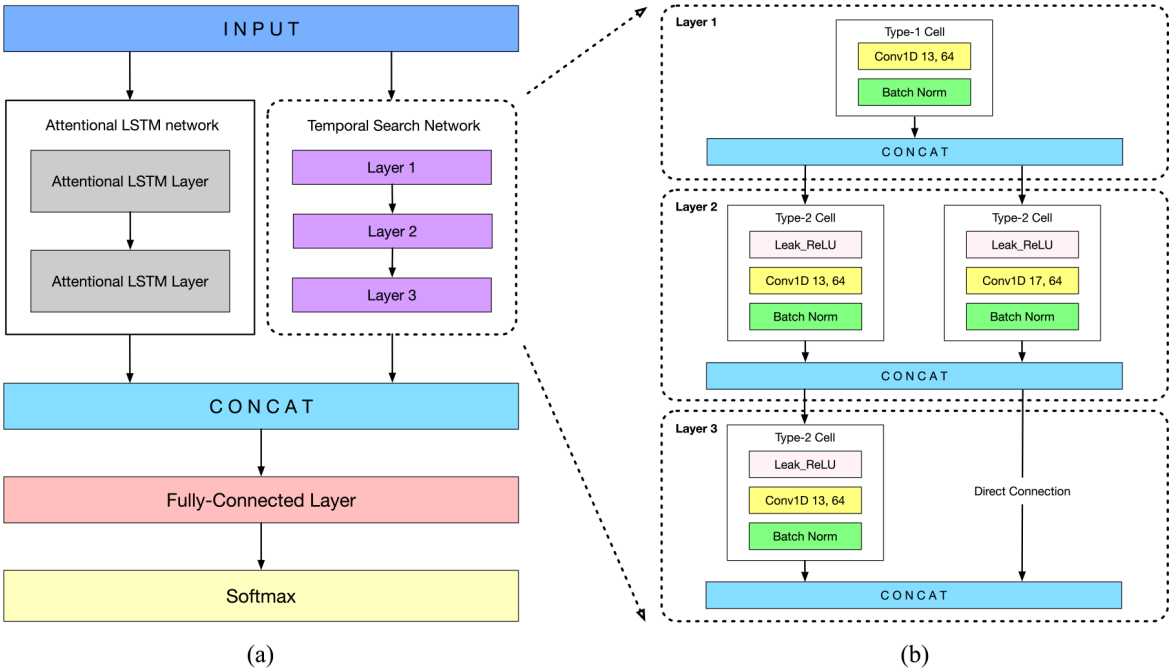
In the well known fcn-lstm architecture, it is observed that adopting three convolutional neural networks (CNN) layers achieves decent performance when addressing time series classification problems [9]. Inspired by this, we propose a three-layer structure as the temporal search network, where each layer is determined by selecting one or more components from a set of basic computation cells. Actually, determining the structure for three layers only requires limited computational expenses, especially when the number of candidate computation cells for selection is small. Besides, in each layer, a reduction operation (e.g. concatenation) is applied to concatenate the output of all computation cells within the layer. Details can be found in Subsections *Search Space*, *Basic Computation Cells*, and *Constraints and Complexity* below.

C. Search Space

Similar to a few works [16] [17], our task is to iteratively select a cell from a set of basic computation cells and then embed it in the temporal search network being constructed as a building block. Let $C = \{c_1, c_2, ..., c_N\}$ be the set of basic computation cells, where c_k is the k-th computation cell in C and N is the cardinality of C. In this paper, C is designed in advance and fixed during the architecture search process. A number of commonly used computation cells are included, namely convolutional cells with different kernels, residual cell, and direct connection cell.

A temporal search network can be viewed as a directed acyclic graph, consisting of a number of computation cells selected from C. Let $G = \{G_1, G_2, ..., G_M\}$ be the set of candidate graphs, where M denotes the cardinality of G and $G_t \in G$ is the t-th graph. G_t corresponds to an explicit temporal search network structure. In this paper, RNTS aims at finding an appropriate graph $G_t \in G$. As mentioned above, there are three layers in our temporal search network. For an arbitrary $G_t \in G$, its definition is written in Eq. (1).

$$G_t = \begin{pmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,N-1} & g_{1,N} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,N-1} & g_{2,N} \\ g_{3,1} & g_{3,2} & \cdots & g_{3,N-1} & g_{3,N} \end{pmatrix} \quad (1)$$



(a) (b)

Fig. 1. (a) Schematic diagram of the proposed RNTS; (b) an example structure of the temporal search network on dataset 'Rock' in the UCR 2018 archive. In (b), 'conv1d 13,64' represents a 1-dimension convolutional neural network, where its filter and channel sizes are set to 13 and 64, respectively; 'Batch norm' is a batch normalization regularization unit; the four convolutional cells and the direct connection are selected from a set of basic computation cells. The structure in (b) is automatically designed to meet the dataset requirements of 'Rock', which forms an explicit temporal search network for basic feature extraction.

where, $g_{i,j}$ denotes whether the j-th cell, $c_j, j = 1, 2, ..., N$, is selected as a building block of layer i, $i = 1, 2, 3$. $g_{i,j} = 1$ means c_j is selected and $g_{i,j} = 0$, otherwise.

D. Basic Computation Cells

The design of computation cells in C significantly influences the performance of the temporal search network as they are candidate building blocks of the three-layer structure. If these cells are not carefully designed, poor feature extraction performance could be resulted. We design three classes of cells for C, namely the convolutional, residual and direct connection cells.

1) Convolutional Cells: We design two types of convelutional cells, namely type-1 and type-2, to extract temporal features. Type-1 cells are used in layer 1 only, to extract temporal shapelets from input data. A type-1 cell primarily consists of a 1-dimension convolutional neural network (Conv1D) and a batch normalization unit, as defined in Eq. (2).

$$f_{type\_1}(x) = f_{bn}(w \otimes x + b) \tag{2}$$

where, x, w and b are the input data, set of learning weights and set of biases of Conv1D, respectively. $\otimes$ is the convolutional computation operation. In addition, the batch normalization eliminates internal covariate shift and is thus able to provide fast training process, which also helps to regularize the proposed RNTS and enhance its accuracy on different datasets.

Type-2 cells are used in layers 2 and 3 only, to mine temporal features from the representations already obtained, as defined in Eq. (3).

$$f_{type\_2}(x) = f_{bn}(w \otimes f_{leak\_relu}(x) + b) \tag{3}$$

where, f_{leak_relu} is a leaky $ReLU$ ($LeakReLU$) function to activate neural units. Different from $ReLU$, the $LeakReLU$ function prevents all of the gradients for the cell from becoming zero if all inputs fall on the negative tail of the activation function.

In order to facilitate accuracy and robustness of the feature extraction, we design three type-1 and three type-2 convolutional cells with different kernel sizes, respectively. To be specific, a type-1 cell is with a kernel size of 13, 15 or 17. Also, a type-2 cell has a kernal size of 13, 15, or 17. So, there are in total 6 convolutional cells in C.

2) Residual Cell: Inspired by the idea of residual networks [24], we allow features of a certain layer to flow from this layer to the terminal layer through a residual cell, making up the loss of the extracted features during their transformation, as defined in Eq. (4).

$$f_{res}(x) = f_{hidden}(x) + f_{channel}(x) \tag{4}$$

where, f_{hidden} is a feature extraction function to extract the hidden representations from x. $f_{channel}$ is a function to change the channel of x, so that the transformed x can be added to the extracted hidden representations.

3) Direct Connection Cell: In order to merge multi-scale temporal features obtained by different layers, we also offer a direct connection option. Different from the residual cell above, this simple connectivity pattern delivers a copy of the features extracted by a certain layer to the terminal layer for concatenation. By directly connecting two layers, we maximize the information flow between them within the temporal search network.

E. Constraints and Complexity

There are six convolutional cells, one residual cell and one direct connection cell in C. The number of cells in C, N, is thus eight. A graph $G_t \in G$ can be represented by a matrix indicating whether a computation cell is selected from C as a building block of a certain layer. For layer i, $i \in \{1, 2, 3\}$, we associate $g_{i,1}, ..., g_{i,6}$ with the 6 convolutional cells in C, respectively, where $g_{i,j}$ and $g_{i,j+3}$, $j = 1, 2, 3$, belong to type-1 and type-2 cells, respectively; we also associate $g_{i,7}$ and $g_{i,8}$ with the residual cell and direct connection cell, respectively.

1) Constraints: As type-2 convolutional cells cannot be used in layer 1, we set $g_{1,j} = 0$, $j \in \{4, 5, 6\}$, for each $G_t \in G$. Similarly, as type-1 convolutional cells cannot be used in layers 2 and 3, we set $g_{i,j} = 0$, $i \in \{2, 3\}$, $j \in \{1, 2, 3\}$, for each $G_t \in G$. Note that both residual and direct connection cells cannot co-exist in neither layer 2 nor layer 3 since their patterns conflict with each other. This constraint is defined in Eq. (5).

$$g_{i,7} + g_{i,8} < 2, \qquad \forall i \in \{1, 2\} \tag{5}$$

Obviously, there is no need to employ either residual cell or direct connection cell in layer 3 as it is the terminal layer. So, we set $g_{3,7} = 0$ and $g_{3,8} = 0$, for each $G_t \in G$.

2) Search Space Size: Different from the well known NAS algorithms [16] [17] that usually consume extremely high computing resources to navigate through a large search space, our temporal search network corresponds to a small search space since we limit the number of candidate computation cells in C, e.g., $N = 8$ in this paper. As a result, only a single 1080ti GPU is sufficient to address a medium-scale time series classification problem in as fast as several hours.

The following analyzes the search space size of our architecture search. As layers 1 and 2 are both constrained by Eq. (5), there are $2^3 \times (2^2 - 1)$ possible combinations of computation cells to form each of them. Layer 3 corresponds to 2^3 possible combinations of computation cells due to the constraints of $g_{3,7}$ and $g_{3,8}$. Besides, it is meaningless that every element in G_t is set to 0 since in this case the three layers are empty. So, the case with all zeros in G_t is not considered in this paper. The search space size, i.e. the size of G, is calculated in Eq. (6).

$$\begin{aligned} M &= (2^3 \times (2^2 - 1))^2 \times 2^3 - 1 \\ &= 4607 \end{aligned} \tag{6}$$

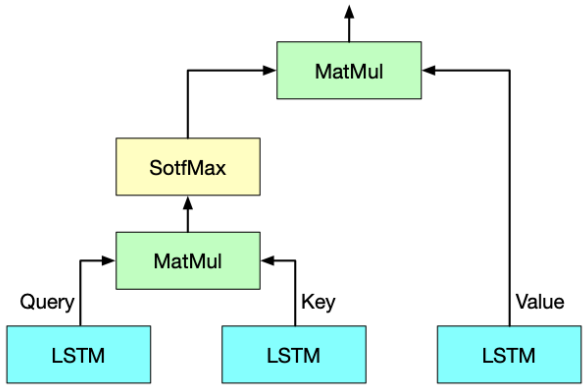


Fig. 2. Attentional LSTM.

F. Attentional LSTM

Nowadays, the most common way for integrating attention mechanism and LSTM networks is to simply stack them one after the other [9]. In contrast, Xiao et al. (2020) [11] proposed an attentional LSTM that deeply merges LSTM networks into a structure based on attention mechanism. This structure has shown excellent performance in exploiting the complicated shapelets and patterns hidden deeply in data that usually cannot be recognized by classical feature extraction meta-networks. This is why we also incorporate the attentional LSTM into the proposed RNTS architecture, as shown in Fig. 1(a).

To be specific, an attentional LSTM can be described as mapping a temporal query and a set of key-value pairs to the output, where the query, key and value are obtained from feature extraction in the LSTM networks. Besides, a compatibility function is used to explore complex relationships between a query and its corresponding key, with the robustness of learning capacities guaranteed. Its definition is written in Eq. (7). Fig. 2 shows the structure of the attentional LSTM, where 'MatMul' is a matrix multiplication operation.

$$ALSTM(f_Q, f_K, f_V) = softmax(f_Q \cdot f_K^T) \cdot f_V \tag{7}$$

where, f_Q,f_K, and f_V are the matrices of the queries, keys and values, respectively. Details can be found in [11].

G. Validation

Prediction As aforementioned, the proposed RNTS mainly consists of the temporal search network and attentional LSTM network. The former is adaptively constructed according to the specific requirements of a time series dataset while the latter is fixed. Hence, each temporal search network results in an explicit RNTS architecture. Let $A = \{A_1, A_2, ..., A_M\}$ denote the set of candidate RNTS architecutres, where the t-th candidate, A_t, as shown in Fig.1(a), adopts G_t as the temporal search network. Let $CLASS = \{class_1, class_2, ..., class_{n_c}\}$ represent the set of class labels on a certain dataset, where n_c is the number of classes. An arbitrary segment of features extracted by A_t can be mapped to a probability distribution

over $CLASS$ by giving a specific predicted class label $\hat{y} = class_j, j = 1, 2, ..., n_c$, through a softmax classifier (e.g. a softmax function). This classifier is defined in Eq. (8).

$$p(\hat{y} = class_j | x_i; \theta^{A_t}) = \frac{e^{\theta_j^{A_t} x_i}}{\sum_{k=1}^{n} e^{\theta_k^{A_t} x_i}} \tag{8}$$

where, x_i is the i-th segment of features obtained by A_t. Besides, θ^{A_t} is the parameter vector of A_t, where $\theta_j^{A_t} \in \theta^{A_t}$ corresponds to the j-th predicted class.

Loss Like many supervised algorithms, the difference between the prediction result and ground truth label, $\mathcal{L}$, is computed according to the cross entropy function, as written in Eq. (9).

$$\mathcal{L} = -\frac{1}{n_{in}} \sum_{i=1}^{n_{in}} y_i log(\hat{y}^{(i)}) \tag{9}$$

where, y_i and $\hat{y}^{(i)}$ are the ground truth label and the predicted label of the i-th sample of the input, respectively, and n_{in} is the size of the input.

In order to evaluate each architecture in A, we design a comprehensive performance indicator $score$, where $score_t$ stands for the performance of A_t on a specific time series classification problem. Unlike DARTS that uses validation loss as the only performance indicator [23], we are concerned with not only the validation loss, but also the validation accuracy and model complexity. To be specific, our RNTS aims at finding an architecture A_t with the lowest mean validation loss, $\mathcal{L}_{val}^{A_t}$, the highest mean validation accuracy, $\alpha_{val}^{A_t}$, and the smallest number of parameters, $n_{par}^{A_t}$. Our score function is defined in Eq. (10).

$$score_t = \alpha_{val}^{A_t} - \mathcal{L}_{val}^{A_t} - n_{par}^{A_t} \tag{10}$$

where, $n_{par}^{A_t}$ is measured in millions. An architecture with the highest score, namely $score_{best}$, is chosen to address the corresponding time series classification problem.

H. Weight Update and Score Calculation

Gradient descent is commonly used to train parameters of a learning model by finding proper weights for them [25]. This technique has achieved decent performance in a variety of application domains, e.g., routing optimization [26] and dynamic shapelets [27]. Hence, our RNTS also applies gradient descent to tune parameters.

Specifically, to evaluate a neural architecture constructed by RNTS is to calculate its performance score via the score function in the input data, D^{in}. For an arbitrary architecture A_t, it is initialized with weights $w_0^{A_t}$. Then, D^{in} is fed into A_t to obtain an initial score, $score_t(w_0^{A_t})$. After that, gradient descent is applied to tune the weights in the first epoch, $w_1^{A_t}$, based on $w_0^{A_t}$. In this way, the weights of A_t are iteratively updated until the end of training, i.e. $w_z^{A_t} \leftarrow w_{z-1}^{A_t}$, where $w_z^{A_t}$ is the weights of the z-th epoch, $z = 1, 2, ..., n_{total}$. Details of the weight update process are shown in Eqs. (11), (12).

$$w_z^{A_t} = w_{z-1}^{A_t} - l_{rate} \nabla_{w_{z-1}^{A_t}} \mathcal{L}_{train}^{A_t}(w_{z-1}^{A_t}) \tag{11}$$

$$\nabla_{w_{z-1}^{A_t}} \approx \frac{\mathcal{L}_{train}^{A_t}(w_{z-1}^{A_t} + \xi) - \mathcal{L}_{train}^{A_t}(w_{z-1}^{A_t} - \xi)}{2\xi} \tag{12}$$

where, $\mathcal{L}_{train}^{A_t}$ is the training loss of A_t, computed using Eq. (9), l_{rate} is the corresponding learning rate, and ξ>0 is an arbitrarily small number. Note that ξ can be replaced with l_{rate} in this paper. After training, we obtain $w_{n_{total}}^{A_t}$, which is used to calculate the validation score $score_t(w_{n_{total}}^{A_t})$. The RNTS-based temporal search network generation is written in Alg. 1.

Algorithm 1 RNTS-based Structure Generation

Input: Raw data D^{raw}; ▷ D^{raw}: raw data of a given dataset
Output: The best temporal search network G_{best};

```
Initialize all global variables;
Obtain D^in by the preprocessing operation;  ▷ D^in: the input data of RNTS
Generate candidate graph set G = {G_1, G_2, ..., G_M};
Initialize score set Ω_score = ∅;
for t = 1 to M do
    Construct A_t based on G_t;
    Initialize A_t with w_0^{A_t};
    for z = 1 to n_total do
        Compute L_train^{A_t}(w_z^{A_t}) using Eq. (9);
        Update w_z^{A_t} using Eqs. (11), (12);
    end for
    end for
    Calculate score_t(w_{n_total}^{A_t}) using Eq. (10);
    Set score_t = score_t(w_{n_total}^{A_t});
    Set Ω_score = Ω_score ∪ {score_t};
Obtain Score_best = max(Ω_score);
return G_best associated with Score_best;
```

TABLE I
DETAILS OF THE 24 SELECTED DATASETS.

Dataset	TrainSize	TestSize	Classes	SeriesLength	Type
Adiac	390	391	17	176	Image
ArrowHead	36	175	3	251	Image
Beef	30	30	5	470	Spectro
BirdChicken	20	20	2	512	Image
Car	60	60	4	577	Sensor
ChlorineCo.	467	3840	3	166	Sensor
Coffee	28	28	2	286	Spectro
Earthquakes	322	139	2	512	Sensor
ECG200	100	100	2	96	ECG
Ham	109	105	2	431	Spectro
Herring	64	64	2	512	Image
ItalyPow.D.	67	1029	2	24	Sensor
Meat	60	60	3	448	Spectro
OliveOil	30	30	4	570	Spectro
Plane	105	105	7	144	Sensor
ProximalPha.O.A.	400	205	3	80	Image
ProximalPha.TW	400	205	6	80	Image
Strawberry	613	370	2	235	Spectro
Wine	57	54	2	234	Spectro
Chinatown	20	345	2	24	Traffic
DodgerLoopD.	78	80	7	288	Sensor
PowerCons	180	180	2	144	Power
Rock	20	50	4	2844	Rock

III. Performance Evaluation

This section explains the experimental setup first and the ablation study on RNTS later, and finally analyzes the experimental results.

A. Experimental Setup

1) Preprocessing: The UCR 2018 archive[1] [28] is one of the most popular time series archives, including 128 datasets with various lengths in different application domains. To verify the performance of the proposed RNTS on different applications, we select 24 well recognized datasets from the archive. The details of these datasets are shown in Table I.

Let D^{raw} denote the raw data of a given dataset. A preprocessing operation is applied to convert D^{raw} to input data $D^{in} = \{S^{train}, S^{val}, S^{test}\}$, where S^{train}, S^{val}, and S^{test} are the data for training, validation and testing, respectively. In particular, we set $S^{val} = S^{test}$ during model training. Especially, this paper adopts data normalization technique to eliminate the negative effects of the bizarre samples.

2) Learning Rate: As known, learning rate has a significant impact on updating the weights of parameters of a model. This paper dynamically tunes the learning rate to adapt to the situations of different epochs during training. Let n_{total} and n_{decay} denote the number of the total training epochs and size of each decay period, respectively. The learning rate of the k-th decay period, $l_{rate}(k)$, $k = 2, ..., K$, is defined in Eq. (13), where $K = \lceil n_{total}/n_{decay} \rceil$.

$$l_{rate}(k) = (1 - decay) \times l_{rate}(k-1) \tag{13}$$

where, $decay$ is the decay rate of the learning rate and K is the total number of the decay periods. In this paper, we set $l_{rate}(1)$ = 0.01 and $decay$ = 0.1.

Besides, all experiments were run on a desktop with Ubuntu OS 18.04, Nvidia-GTX 1080ti with 11G RAM and AMD R5 1400 CPU with 16G RAM. Python 3.6 [2] and Tensorflow 1.14 [3] were used to implement our experiments.

B. Ablation Study

TABLE II
Results of the ablation study.

Alg. Num.	Alg. Name	Beef	Car	Earthquake	Herring	OliveOil	Rock	MeanAcc
1	RNTS with α_{val}	**0.9**	0.85	**0.794367**	**0.765625**	0.933333	0.88	0.853888
2	RNTS with $\mathcal{L}_{val}$	**0.9**	**0.9**	0.769784	**0.765625**	**0.966667**	0.86	0.860346
3	RNTS with n_{par}	**0.9**	0.833333	0.755396	0.75	0.933333	0.80	0.828677
4	RNTS with α_{val} & $\mathcal{L}_{val}$	**0.9**	**0.9**	**0.794367**	**0.765625**	**0.966667**	**0.92**	**0.874443**
5	RNTS with α_{val} & n_{par}	**0.9**	0.816667	0.741007	0.75	0.933333	0.86	0.833501
6	RNTS with $\mathcal{L}_{val}$ & n_{par}	**0.9**	0.816667	0.769784	**0.765625**	0.933333	0.86	0.840901
7	Supervised RTFN w/o ALSTM	0.8	0.5	0.741007	0.625	0.833333	0.62	0.686557
8	RNTS w/o ALSTM	0.866667	0.833333	0.755396	0.65625	0.9	0.68	0.781941
9	Ours	**0.9**	**0.9**	**0.794367**	**0.765625**	**0.966667**	**0.92**	**0.874443**

To evaluate the impact of each performance metric on RNTS, we select six long time series datasets with lengths over 500, including Beef, Car, Earthquake, Herring, OliveOil and Rock. To justify our main contribution in this paper, i.e., the temporal search network, we add RNTS without attentional LSTM (RNTS w/o ALSTM) and the supervised RTFN structure [11] without attentional LSTM (Supervised RTFN w/o ALSTM) to the competition. We compare RNTS with eight algorithms in terms of the mean accuracy, i.e., MeanACC, as shown in Table II.

TABLE III
Numbers of parameters with different algorithms (measured in millions).

Alg. Num.	Beef	Car	Earthquake	Herring	OliveOil	Rock
1	1.027	1.092	1.053	1.308	1.049	3.110
2	0.887	0.980	1.185	0.926	1.081	2.946
4	0.887	0.980	1.053	0.926	1.042	2.990
5	0.887	1.030	1.024	0.923	1.102	2.973
6	0.887	0.978	0.975	0.926	1.020	2.942
9	0.887	0.980	1.053	0.926	1.041	2.986

If focusing on RNTS with a single performance metric, i.e., α_{val}, $\mathcal{L}_{val}$, or n_{par}, one can see that none of them achieves stable performance when considering all the six datasets. RNTS with $\mathcal{L}_{val}$ is the best among the three, winning four datasets only. The reason behind is that $\mathcal{L}_{val}$ pays more attention to direct connections than residual ones, leading to structures with less residual cells embedded. However, it is difficult for such a network to compensate for the loss of features during their transmission process. Besides, RNTS with α_{val} concerns more about the relationships between direct correlation layers, ignoring those relationships between indirect correlation ones. It is more likely to result in a temporal search network with none or a few direct connections only. Nevertheless, this could not help much to provide RNTS with sufficient features, leading to possible performance deterioration.

If considering RNTS with two performance metrics, i.e., α_{val} & $\mathcal{L}_{val}$, α_{val} & n_{par}, or $\mathcal{L}_{val}$ & n_{par}, one can find that RNTS with α_{val} & $\mathcal{L}_{val}$ performs the best on each dataset. This is because it takes into account both the interrelationships between the corresponding layers and the loss of features in the feature transmission process, which helps RNTS to extract sufficient basic features. Unfortunately, to obtain enough features from representations between layers requires additional computation cells from C, which, in turn, leads to higher computing resource consumption. This is why we introduce n_{par} into $score$, i.e., Ours as defined in Eq. (10), for the purpose of striking a balance between feature quality and model complexity. Although unnecessary connections are removed from RNTS, ours is also able to obtain equivalent performance with RNTS with α_{val} & $\mathcal{L}_{val}$, i.e., both of them have the same MeanACC value on each of the six datasets. This is because those unnecessary connections may carry redundant features, which is trivial to the feature extraction process. Besides, for the six RNTS-based approaches, their numbers of parameters are shown in Table III. If comparing RNTS w/o ALSTM and the supervised RTFN w/o ALSTM, one can easily observe that the former outperforms the latter on all datasets. This is because the former is able to obtain more basic features from the input data since it adaptively finds a promising feature extraction network for each dataset case

[1] http://www.timeseriesclassification.com/
[2] https://www.python.org/
[3] https://tensorflow.google.cn/

TABLE IV
EXPERIMENTAL RESULTS OF DIFFERENT ALGORITHMS ON 24 DATASETS.

Name	Existing SOTA	Inception-Time	Best:lstm -fcn	Vanilla:ResNet -Transformer	ResNet -Tran1	ResNet -Tran2	ResNet -Tran3	OS-CNN	Supervised RTFN	Ours
Adiac	0.857	0.841432	**0.869565**	0.84399	0.849105	0.849105	0.849105	0.838875	0.792839	0.808184
ArrowHead	0.88	0.845714	**0.925714**	0.891429	0.891429	0.891429	0.891429	0.862857	0.851429	0.88
Beef	**0.9**	0.7	**0.9**	0.866667	0.866667	0.866667	0.866667	0.833333	**0.9**	**0.9**
BeetleFly	0.95	0.8	**1**	**1**	0.95	0.95	**1**	0.9	**1**	**1**
Car	0.933	0.883333	**0.966667**	0.95	0.883333	0.866667	0.3	0.933333	0.883333	0.9
ChlorineCo.	0.872	0.876563	0.816146	0.849479	0.863281	0.409375	0.861719	0.85651	0.894271	**0.921094**
Coffee	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Earthquakes	0.801	0.741007	**0.81295**	0.755396	0.755396	0.76259	0.755396	0.705036	0.776978	0.794367
ECG200	0.92	0.93	0.91	0.94	**0.95**	0.94	0.93	0.93	0.92	0.93
Ham	0.781	0.714286	**0.809524**	0.761905	0.780952	0.619048	0.514286	0.714286	**0.809524**	**0.809524**
Herring	0.703	0.671875	0.75	0.703125	0.734375	0.65625	0.703125	0.671875	0.75	**0.796875**
ItalyPow.D.	0.97	0.965015	0.963071	0.965015	0.969874	0.962099	**0.971817**	0.951409	0.964043	0.965015
Lightning2	**0.8853**	0.770492	0.819672	0.852459	0.852459	0.754098	0.868852	0.819672	0.836066	0.852459
Meat	**1**	0.933333	0.883333	**1**	**1**	**1**	**1**	0.983333	**1**	**1**
OliveOil	0.9333	0.833333	0.766667	**0.966667**	0.9	0.933333	0.9	0.866667	**0.966667**	**0.966667**
Plane	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
ProximalPha.O.A.	0.8832	0.84878	0.887805	0.887805	**0.892683**	0.882927	0.892683	0.853658	0.878049	**0.892683**
ProximalPha.TW	0.815	0.77561	**0.843902**	0.819512	0.814634	0.819512	0.819512	0.785366	0.834146	**0.843902**
Strawberry	0.976	0.983784	**0.986486**	**0.986486**	**0.986486**	**0.986486**	**0.986486**	**0.986486**	**0.986486**	**0.986486**
Wine	0.889	0.611112	0.833333	0.851852	0.87037	0.87037	0.907407	0.814815	0.907407	**0.925926**
Chinatown	0.9565	0.985423	0.982609	0.985507	0.985507	0.985507	0.985507	0.97971	0.985507	**0.988406**
DodgerLoopD.	0.5125	0.15	0.6375	0.5357	0.55	0.4625	0.5	0.6125	0.675	**0.7125**
PowerCons	0.9722	0.944444	0.994444	0.933333	0.944444	0.927778	0.927778	**1**	**1**	**1**
Rock	0.84	0.8	**0.92**	0.78	**0.92**	0.82	0.76	0.7	0.88	**0.92**
MeanACC	0.884583	0.816897	0.886641	0.880264	0.883791	0.842323	0.841324	0.858322	0.895489	**0.908087**
Win	1	0	4	0	1	0	1	0	0	**5**
Tie	4	2	6	6	6	4	6	4	9	**12**
Lose	19	22	14	18	17	20	17	20	15	**7**
Best	5	2	10	6	7	4	7	4	9	**17**
AVG_rank	5.187500	7.916667	4.791667	5.125000	4.833333	6.458333	5.416667	7.354316	4.604167	**3.312500**

according to its characteristics. On the contrary, the supervised RTFN w/o ALSTM uses a fixed network architecture for feature extraction, no matter how many datasets with different requirements are considered. This, to some extent, also unveils the difference between the NAS-based and human experience based feature extraction approaches.

C. Result Analysis and Discussion

To evaluate the performance of RNTS, we compare it with a number of state-of-the-art algorithms on the 24 datasets selected from the UCR 2018 archive, in terms of the top-1 accuracy, as shown in Table IV. This paper adopts not only MeanACC but also 'win'/'tie'/'lose' and AVG_rank as three performance measures. To be specific, 'win', 'tie' and 'lose' are used to rank all algorithms, namely based on how many datasets a specific algorithm achieves performance better than, equivalent to or worse than the others. For an arbitrary algorithm, its 'best' value is equal to the summation of the corresponding 'win' and 'tie' values. Besides, according to the average Geo-ranking method, AVG_rank measures the average difference between the accuracy of an algorithm and the best accuracy among all algorithms [9] [10] [11].

Table IV collects the MeanACC, 'win'/'tie'/'lose' and AVG_rank results on the 24 selected datasets. For a certain dataset, the existing SOTA stands for the algorithm with the best performance on it, mainly including BOSS [10], FCN [6], ConvTimeNet [10], Time2Graph [27], EE [10], SC-ResNet [29], COTE [10].; similarly, the Best: lstm-fcn is the best algorithm on that dataset, e.g., LSTM-FCN and Normalized LSTM-FCN are included [9].

If considering all the three performance measures above, our RNTS is no doubt the best among all algorithms for comparison since ours obtains the highest MeanACC and 'best' values, namely 0.908087 and 17, and the smallest AVG_rank value, namely 3.312500. Meanwhile, the supervised RTFN and Best: lstm-fcn are the second and third best algorithms while InceptionTime and OS-CNN are the two worst ones. Besides, ResNet-Trans1 overweighs the other three transformer-based algorithms in terms of MeanACC, 'best' and AVG_rank, including Vanilla: ResNet-Transformer, ResNet-Trans2 and ResNet-Trans3.

The following explains the reasons behind the findings above. As a dual-network-based model, RNTS automatically generates different temporal search networks to handle different datasets with various requirements, helping to extract as many basic features as possible in each dataset case. In addition, it adopts an attentional LSTM network to explore those complex shapelets and relationships that are usually ignored by the corresponding temporal search network. This is why RNTS outperforms the other algorithms on the 24 well known datasets. On the other hand, the supervised RTFN, Best: lstm-fcn, and ResNet-Trans1 also belong to dual-network-based models. They are also concerned with the extraction of both basic and complex features, helping them to gain decent overall performance. Each of them, however, uses a fixed feature extraction network, which may lead to severe loss of basic features during the feature transmission process on some datasets. This is because to design a single feature extraction network that is well adapted to all datasets is quite challenging. Meanwhile, InceptionTime and OS-CNN are single-network-

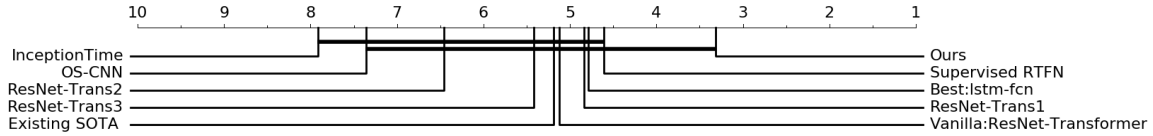


Fig. 3. Critical difference diagram of the average ranks of various algorithms on 24 datasets.

based models, which are not capable of discovering those relationships hidding in the representations already obtained. Besides, both of them also suffer from the drawback of adapting one feature extraction network to different datasets. The AVG_rank results are shown in Fig. 3.

IV. Conclusion

This paper presents a dual-network-based framework for time series classification, RNTS, which mainly consists of a temporal search network and an attentional LSTM network. In particular, the first network adaptively transforms itself to adapt to the requirements of each dataset. By considering not only the validation loss and accuracy but also the model complexity, RNTS aims at striking a balance between overall performance and time efficiency. In the experiments, compared with other state-of-the-art algorithms, RNTS ranks first on all 24 datasets with respect to both the mean accuracy and AVG_rank, and on 17 datasets in terms of the 'win'/'tie'/'lose' measure, respectively. This also unveils the potential of RNTS to be applied to time series classification problems in various real world domains.

References

[1] S. Wu, P. Chen, A. L. Swindlehurst and P. Hung. Cancelable biometric recognition with ECGs: subspace-based approaches," *IEEE Trans. Inf. Foren. Sec.*, vol. 14, no. 5, pp. 1323-1336, May 2019

[2] S. Du, T. Li, Y. Yang, X. Gong and S.-J. Horng. An LSTM based encoder-decoder model for multistep traffic flow prediction. In *Proc. IEEE IJCNN 2019*, Budapest, Hungary, pp. 1-8, 2019.

[3] D. Jiang, L. Huo, P. Zhang and Z. Lv. Energy-efficient heterogeneous networking for electric vehicles networks in smart future cities. *IEEE Trans. Intell. Transp.*, vol. 22, no. 3, pp. 1868-1880, 2021.

[4] Y. Ji, R. Gu, Z. Yang, J. Li, H. Li, and M. Zhang. Artificial intelligence-driven autonomous optical networks: 3S architecture and key technologies. *Sci. China Inf. Sci.*, vol. 63, article 160301, 2020.

[5] H. Jiang, Z. Wang and H. He. An evolutionary computation approach for smart grid cascading failure vulnerability analysis. In *Proc. IEEE SSCI*, Xiamen, China, pp. 332-338, 2019.

[6] H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar and P.A. Muller. Deep learning for time series classification: a reviewer. *Data Min. Knowl. Disc.*, vol. 33, pp. 917-963, 2019.

[7] H.I. Fawaz, B. Lucas, G. Forestier, C. Pelletier, D.F. Schmidt, J. Weber, G.I. Webb, L. Idoumghar, P. Muller and F. Petitjean. InceptionTime: finding AlexNet for time series classification. *Data Min. Knowl. Disc.*, vol. 34, pp. 1936–1962, 2020.

[8] W. Tang, G. Long, L. Liu, T. Zhou, J. Jiang and M. Blumenstein. Rethinking 1D-CNN for time series classification: a stronger baseline. *arXiv preprint arXiv: 2002.10061*, 2020.

[9] F. Karim, S. Majumdar and H. Darabi. Insights into LSTM fully convolutional networks for time series classification. *IEEE Access*, vol. 7, pp. 67718-67725, 2019.

[10] S.H. Huang, L. Xu and C. Jiang. Residual attention net for superior cross-domain time sequence modeling. *arXiv preprint arXiv: 2001.040771*, 2020.

[11] Z. Xiao, X. Xu, H. Xing and J. Chen. RTFN: robust temporal feature network. *arXiv preprint arXiv: 2008.07707*, 2020.

[12] P. Ren, Y. Xiao, X. Chang, P. Huang, Z. Li, X. Chen and X. Wang. A comprehensive survey of neural architecture search: challenges and solutions. *arXiv preprint arXiv: 2006.02903*, 2020.

[13] L. Fedorov, R.P. Adams, M. Mattina and P. Whatmough. Sparse: sparse architecture search for CNNs on resource constrained microcontrollers. In Proc. *NeurIPS 2019*, Vancouver, Canada, pp. 4978-4990, 2019.

[14] B. Baker, Q. Gupta, N. Naik and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv: 1611.02167*, 2016.

[15] B. Zoph and Q.V. Le. Neural architecture search with reinforcement learning. In Proc. *ICLR 2017*, Toulon, French, 2017.

[16] E. Real, A. Aggarwal, Y. Huang and Q.V. Le. Regularized evolution for image classifier architecture search. In Proc. *AAAI 2019*, pp. 4780-4789, 2019.

[17] B. Zoph, V. Vasudevan, J. Shlens and Q.V. Le. Learning transferable architectures for scalable image recognition. In Proc. *IEEE CVPR 2018*, Salt Lake City, USA, pp. 8697-8710, 2018.

[18] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In Proc. *ICLR 2018*, Vancouver, Canada, 2018.

[19] H. Pham, M.Y. Guan, B. Zoph, Q.V. Le and J. Dean. Efficient neural architecture search via parameter sharing. In Proc. *ICML 2018*, Stockholm, Sweden, 2018.

[20] J. Peng, M. Sun, Z. Zhang, T. Tan and J. Yan. Efficient neural architecture transformation search in channel-level for object detection. In Proc. *NeurIPS 2019*, Vancouver, Canada, pp. 14290-14299, 2019.

[21] Y. Zhang, Z. Qiu, J. Liu, T. Yao, D. Liu and T. Mei. Customizable architecture search for semantic segmentation. In Proc. *IEEE CVPR 2019*, Long Beach, CA, USA, pp. 11633-11642, 2019.

[22] H. Rakhshani, H.I. Fawaz, L. Idoumghar, G. Forestier, J. Lepagnot, J. Weber, M. Brévilliers and P.-A. Muller. Neural architecture search for time series classification. In Proc. *IJCNN 2020*, Glasgow, United Kingdom, pp. 1-8, 2020.

[23] H. Liu, K. Simonyan and Y. Yang. DARTS: differentiable architecture search. In Proc. *ICLR 2019*, New Orleans, USA, 2019.

[24] K. He, X. Zhang, S. Ren and J. Sun. Deep residual learning for image recognition. In Proc. *IEEE CVPR 2016*, Las Vegas, USA, pp. 770-778, 2016.

[25] Y. LeCun, Y. Bengio and G. Hinton. Deep learning. Nature, pp. 436-444, 2015.

[26] J. Chen, Z. Xiao, H. Xing, P. Dai, S. Luo and M.A. Iqbal. STDPG: a spatio-temporal deterministic policy gradient agent for dynamic routing in SDN. In Proc. *IEEE ICC 2020*, Dublin, Ireland, pp. 1-6, 2020.

[27] Z. Cheng, Y. Yang, W. Wang, W. Hu, Y. Zhuang and G. Song. Time2Graph: revisiting time series modeling with dynamic shapelets. In Proc. *AAAI 2020*, New York, USA, 2020.

[28] H.A. Dau, A. Bagnall, K. Kamgar, C.-C.M. Yeh, Y. Zhu, S. Gharghabi, C.A. Ratanamahatana and E. Keogh. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 6, pp. 1293-1305, 2019.

[29] M. Wenninger, S.P. Bayerl, J. Schmidt and K. Riehammer. Timage – a robust time series classification pipeline. In Proc. *ICANN 2019*, Munich, Germany, pp. 10-18, 2019.