IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 20, 2020, accepted June 2, 2020, date of publication June 12, 2020, date of current version June 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3002095

Dynamic Multi-Scale Convolutional Neural Network for Time Series Classification

BIN QIAN[1], YONG XIAO[1], ZHENJING ZHENG[2], MI ZHOU[1], WANQING ZHUANG[2], SEN LI[2], AND QIANLI MA[2], (Member, IEEE)

[1]Electric Power Research Institute, China Southern Power Grid, Guangzhou 510080, China
[2]School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China

Corresponding author: Qianli Ma (qianlima@scut.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61502174 and Grant 61872148, in part by the Natural Science Foundation of Guangdong Province under Grant 2017A030313355, and Grant 2019A1515010768, in part by the Guangzhou Science and Technology Planning Project under Grant 201704030051 and Grant 201902010020, and in part by the Key Research and Development Program of Guangdong Province under Grant 2018B010107002.

ABSTRACT Time series classification is an essential task in many real-world application domains. As a popular deep learning network, convolutional neural networks have achieved excellent performance in time series classification tasks. The filters of the convolutional neural networks are fixed length and shared by each sample. However, each time series usually has different time scale features. Therefore, convolutional neural networks are not capable of extracting multi-scale features for each sample flexibly. In this paper, we propose dynamic multi-scale convolutional neural network to extract multi-scale feature representations existing in each time series dynamically. Specifically, we design a variable-length filters generator to produce a set of variable-length filters conditioned on the input time series. To make model differentiable, we use the learnable soft masks to control the lengths of variable-length filters. Therefore, the feature representations of different time scales can be captured through the variable-length filters. Then, the max-over-time pooling is used to select the most discriminative local patterns. Finally, the fully connected layer with softmax output is employed to calculate the final probability distribution for each class. Experiments conducted on extensive time series datasets show that our approach can improve the performance of time series classification through the learning of variable-length filters. Furthermore, we demonstrate the effectiveness of dynamically learning variable-length filters for each sample through the visualization analysis.

INDEX TERMS Convolutional neural networks, multi-scale temporal features, time series classification.

I. INTRODUCTION

Time series data is ubiquitous. Human activities and nature produce time series every day, such as medical observations, financial recordings, physiological signals, and weather data. Time series classification (TSC) is a fundamental task in many real-world application domains. Recent research showed that TSC could benefit to auxiliary medical diagnosis [1], [2], human activity recognition [3], speech analysis [4], etc. According to the number of variables in time series data, TSC tasks can be categorized into two types: univariate time series classification (UTSC) and multivariate time series classification (MTSC). We focus on UTSC tasks in this paper.

The associate editor coordinating the review of this manuscript and approving it for publication was Fan-Hsun Tseng.

In recent decades, many researchers have focused on TSC tasks and proposed various methods to solve them. As the earliest baseline, the distance-based methods classify a time series by nearest neighbor classifier or support vector machine with designed distances. Typical models of distance-based methods include the 1-Nearest Neighbor with Euclidean distance (1NN-ED) [5], the Dynamic Time Warping (1NN-DTW) [5], and the derivative distance-based Dynamic Time Warping (DD_{DTW}) [6]. DD_{DTW} uses the weighted combination of the DTW distance between two time series and the DTW distance between their corresponding first-order difference sequences. Unlike the distance-based methods, the feature-based methods extract discriminative features from time series for classification. Methods include the bag of SFA symbols (BOSS) [7], learned shapelets (LS) [8], and learned pattern similarity (LPS) [9].

The statistical time series methods [10]–[13] employ summary statistics of time series as features for classification. For example, time series forest (TSF) [11] and the time series bag of features (TSBF) [12] use simple statistical features such as mean, standard deviation, and slop for classification. Lei and Wu [13] use the statistical features (maximum, mean, polar difference, variance, etc.) from the subsequences to constitute new time series, and combine with the fully convolutional network for time series classification.

Although distance-based, feature-based, and statistical time series methods can achieve high accuracy and have acceptable interpretation, they need to define distance metrics explicitly or handcrafted features. They are error-prone and may not be suitable for time series of all domains.

To further improve the performance of TSC, the ensemble-based methods perform classification by ensemble multiple different classifiers. For example, Elastic Ensemble (EE) [14] is an ensemble classifier with 1NN based on 11 elastic distance measures. The Shapelet Transform (ST) [15] uses the shapelet transformation based on a heterogeneous ensemble. The collection of transformation ensembles (COTE) [16] ensembles 35 different classifiers constructed in the domain of time, frequency, change, and shapelet transformation, respectively. However, they are usually suffering from high computational complexity.

In recent years, deep learning architectures have been applied to a wide variety of tasks and achieved great success. Wang *et al.* [17] proposed a strong baseline for TSC tasks with deep neural networks. There are three different deep neural networks used for TSC, including Multilayer Perceptrons (MLP), Residual Network (ResNet), and Fully Convolutional Network (FCN). Among them, FCN achieves state-of-the-art performance on the TSC tasks.

However, multi-scale temporal information naturally exists in time series [18], which plays a vital role in the TSC tasks. For example, we visualize three instances from two categories in the Computers dataset in Figure 1. It shows that the feature scales of instance in class 1 are different from the ones of instances in class 2. In addition, the two instances of class 2 also have different scales of features. Therefore, not only different samples have different scales of features, but also, there are different scales of features within a sample. How to capture sample-specific multi-scale features is still a challenge for time series classification. FCN uses fixed-length filters for each layer, which is unable to extract multi-scale temporal features of time series. Some existing convolutional neural network-based approaches [18], [19] are dedicated to capturing multi-scale temporal features of time series and verify that multi-scale temporal features can effectively improve the performance of time series classification models. Moreover, in the field of image processing [20] and natural language processing [21], multi-scale features have also proved to be beneficial for better classification. However, these time series classification approaches rely on prior knowledge to manually select different scales for each dataset. Therefore, they are unable to learn

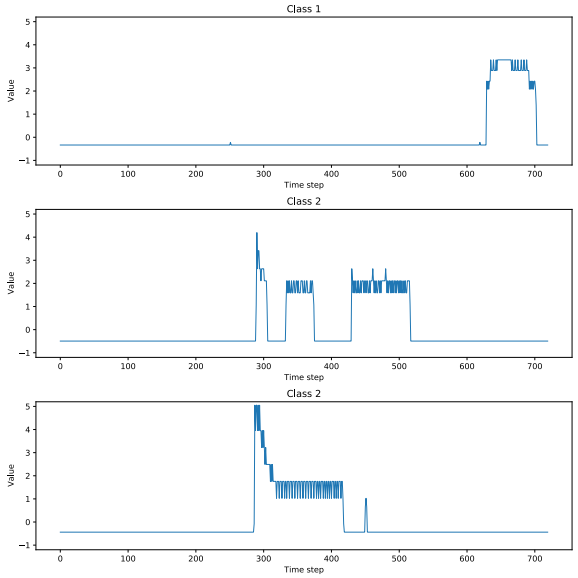


FIGURE 1. Three time series instances in the Computers dataset. The X-axis and Y-axis indicate time step and corresponding value, respectively. The instances from the same class and different classes have different scales of features.

variable-length filters based on each time series adaptively since all the samples share their filters. Hence, they cannot capture sample-specific multi-scale temporal features.

To address the above issues, in this paper, we propose a novel end-to-end variable-length filter learning model called Dynamic Multi-Scale Convolutional Neural Network (DMS-CNN), for TSC tasks. Specifically, we propose a variable-length filters generator to generate a set of variable-length filters conditioned on the input time series. The core of the variable-length generator is to determine the length of each filter based on each sample of time series. To this end, we use a subnet to learn the lengths of filters. After that, we employ the lengths of filters to generate the masks and apply them to control the lengths of the filters, thereby obtaining the variable-length filters. The variable-length filters are used to replace the filters of the conventional convolutional neural network to capture features with different time-scales in each time series. Then, max-over-time pooling [22] is used to maintain temporal invariance and select the most discriminative local patterns. Finally, we use a fully connected layer with softmax output to calculate the final probability distribution for each class. Compared with existing approaches, our approach is also dedicated to capturing multi-scale temporal features of time series for classification. The major difference is that our approach can extract sample-specific multi-scale feature representations existing in each time series. Specifically, it can be summarized as two points: (1) Our approach is to learn the scales of filters through data rather than relying on prior knowledge to select the scales of filters manually. (2) Our approach can learn sample-specific multi-scale feature representations for each time series. In contrast, the existing approaches require

a large number of filters of multiple scales to cover the scales of each sample as many as possible. Experiments conducted on 85 UCR time series datasets show that DMS-CNN can improve the performance of TSC tasks, and visualization analysis further demonstrates the effectiveness of the dynamically learning variable-length filters.

The remainder of this paper is organized as follows. In section II, The preliminary concepts related to our work are introduced. Section III presents the detail of our proposed model DMC-CNN. The experimental results and analysis on extensive time series datasets to demonstrate the effectiveness of DMC-CNN are given in section IV. Finally, Section V provides conclusion and future work of this paper.

II. PRELIMINARIES

A. NOTATION OF UNIVARIATE TIME SERIES CLASSIFICATION

Univariate time series classification is a task that takes a univariate time series sample as the input of the model and predicts its label.

A univariate time series dataset can be denoted as $\mathcal{D} = \{(\boldsymbol{T}_1, \boldsymbol{Y}_1), (\boldsymbol{T}_2, \boldsymbol{Y}_2), \cdots, (\boldsymbol{T}_N, \boldsymbol{Y}_N)\}$, where $\boldsymbol{T}_i = \{x_1, x_2, \cdots, x_L\}$ denotes the i-th sample, and $x_j \in \mathbb{R}, j = 1, 2, \cdots, L$. $\boldsymbol{Y}_i \in \mathbb{R}^c$ denotes the label of the i-th sample. N, L, and c are the number of samples, the length of input time series, and the number of categories, respectively. The time series dataset is usually divided into two sets: a training set and a testing set. The training set is used to train the model and obtain the parameters of the model, and the testing set is used only for final evaluation.

B. CONVOLUTIONAL NEURAL NETWORKS

Recently, deep learning architectures have been applied to a wide variety of tasks and achieved great success. As a popular deep learning architecture, convolutional neural networks (CNNs) [23] have achieved good performance in TSC tasks duo to its powerful feature extraction capability.

The convolutional layer and pooling layer are the main components of a convolutional neural network. In Figure 2, we show the architecture of conventional CNNs for TSC tasks. The model consists of an input layer, a convolutional layer, a pooling layer, and an output layer. First, the original time series is used as the input to the network. Then, the filters of two different lengths are used to convolve the input to extract multi-scale temporal features of the time series. After that, the max-over-time pooling is utilized to select the most discriminative local patterns, which have the most significant impact on classification. Finally, the output of the pooling layer is fed into the fully connected layer with softmax output for classification. Although CNNs use filters of multiple lengths to extract multi-scale temporal features of time series, they cannot adaptively learn variable-length filters conditioned on the input time series.

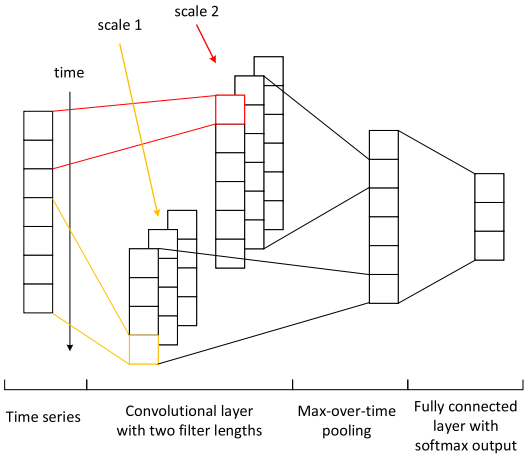


FIGURE 2. The conventional convolutional neural network that consists of a convolutional layer with filters of two different lengths, a max-over-time pooling layer, and a fully connected layer is used for time series classification.

III. DYNAMIC MULTI-SCALE CONVOLUTIONAL NEURAL NETWORK

The general architecture of a DMS-CNN is illustrated in Figure 3. The variable-length filters generator is used to generate a set of filters with different lengths conditioned on the input time series and the randomly initialized filters. Then the input time series is fed into the convolutional layer with variable-length filters to capture the multi-scale temporal features of the time series. After that, we use the max-over-time pooling to select the most discriminative features, which are the most significant for classification. Finally, the fully connected layer with softmax output is used to calculate the final probability distribution for each class.

A. VARIABLE-LENGTH FILTERS GENERATOR

To learn variable-length filters, we control the lengths of the filters by learning masks. The masks are generated based on the input time series and the randomly initialized filters to generate the sample-specific variable-length filters. Therefore, we first use convolution to obtain the embedded representations of the input time series. The subsequences of input time series are used as the input of the convolutional layer.

Given a univariate time series $\boldsymbol{T} = \{x_1, x_2, \cdots, x_L\}^{\mathrm{T}}$, we can obtain P subsequences of length l with a sliding window length l and stride of 1, where $P = L - l + 1$ is the number of subsequences. All the subsequences of time series are concatenated and represented as $\boldsymbol{S}$. We represent $\boldsymbol{S} \in \mathbb{R}^{P \times l}$ as follow:

$$\boldsymbol{S} = \boldsymbol{x}_{1:l} \oplus \boldsymbol{x}_{2:l+1} \oplus \cdots \oplus \boldsymbol{x}_{L-l+1:L}, \tag{1}$$

where $\boldsymbol{x}_{t:t+l-1} \in \mathbb{R}^{l \times 1}$ denotes the l-length subsequence of time series from time step t to $t + l - 1$ and $\oplus$ denotes the concatenation operator.

To obtain the embedded representation of the time series, we use a convolution operation on $\boldsymbol{S}$ with a stride of 1.

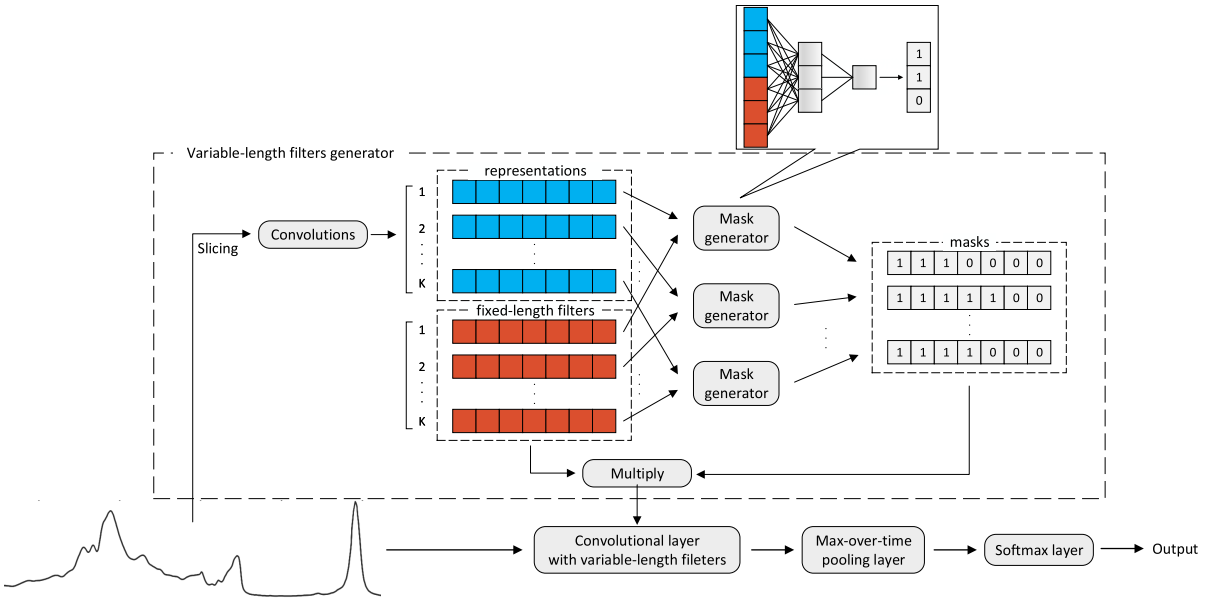


FIGURE 3. General architecture of the Dynamic Multi-Scale Convolutional Neural Network (DMS-CNN). The core of DMS-CNN is variable-length filters generator, which is used to generate a set of filters with different lengths conditioned on the input time series and the randomly initialized filters. Specifically, we first use convolution to obtain the embedded representations of the input time series, which are indicated with the blue lattices. Then, the embedded representations and randomly initialized filters are used to generate a set of filters with different lengths, and we use the red lattices to indicate the randomly initialized filters. Finally, the filters of the conventional CNN are replaced by the variable-length filters to capture features with different time-scales in each time series.

Let $\boldsymbol{W}_i \in \mathbb{R}^{\omega \times P}$ denotes the i-th filter with ω-length. The convolutional result with filter $\boldsymbol{W}_i$ on $\boldsymbol{S}$ is obtained by:

$$\boldsymbol{e}_i = \boldsymbol{W}_i * \boldsymbol{S} + b_i, \tag{2}$$

where b_i denotes the bias, and $*$ is the convolution operation. We use K filters to convolve $\boldsymbol{S}$, and all the convolutional results are denoted as:

$$\boldsymbol{E} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_K\}^{\mathrm{T}}, \tag{3}$$

where $\boldsymbol{E} \in \mathbb{R}^{K \times l}$ is the embedded representations of the input time series, and $\boldsymbol{e}_i \in \mathbb{R}^{l \times 1}, i = 1, 2, \cdots, K$ denotes the convolutional result with filter $\boldsymbol{W}_i$ on the subsequences of time series $\boldsymbol{S}$.

To generate the variable-length filters, a mask generator is proposed to learn the lengths of filters. We first randomly initialize fixed-length filters $\boldsymbol{W}^f = \{\boldsymbol{W}_1^f, \boldsymbol{W}_2^f, \cdots, \boldsymbol{W}_K^f\}^{\mathrm{T}}$ like the conventional CNNs, where $\boldsymbol{W}_i^f \in \mathbb{R}^{l \times 1}$ denotes the i-th fixed-length filter. We concat the embedded representations of time series and the fixed-length filters as inputs to the mask generator.

In the mask generator, we first design a function $m : \mathbb{R}^{2l} \rightarrow [0, 1]$, which determines the length used by each filter. In this work, the function m is implemented as two fully connected layers.

$$r_i = \boldsymbol{W}^{m2} f(\boldsymbol{W}^{m1} \cdot (\boldsymbol{e}_i \oplus \boldsymbol{W}_i^f) + \boldsymbol{b}^{m1}) + b^{m2}, \tag{4}$$

$$l_i = r_i \times l, \tag{5}$$

where $\boldsymbol{W}^{m1} \in \mathbb{R}^{2l \times l}$ and $\boldsymbol{W}^{m2} \in \mathbb{R}^{l \times 1}$ denote the connection weights of the first and second layers of the fully connected network, respectively, and $\boldsymbol{b}^{m1}$ and b^{m2} denote the corresponding bias. l_i denotes the length used by the i-th filter.

We can obtain the length of each filter by Eq. (4) and Eq. (5), and denote as $\boldsymbol{M} = \{l_1, l_2, \cdots, l_K\}$. However, how to enable the filters to have different lengths through learning is a challenging problem. Here we use the masks to control the lengths of different filters. In practice, if we generate the mask directly from the learned lengths, it will make the model non-differentiable and lead to complicated optimization. Instead, we use the soft mask proposed by Jernite *et al.* [24] to approximate this operation.

Given $l_i \in [0, l]$ and a sharpness parameter λ, the mask vector $\boldsymbol{m}_i \in \mathbb{R}^l$ is defined as:

$$\forall j \in 1, 2, \cdots, l, m_{ij} = \sigma(\lambda(l_i - j)), \tag{6}$$

where $\boldsymbol{m}_i = \{m_{i1}, m_{i2}, \cdots, m_{il}\}^{\mathrm{T}}$ denotes the mask vector, and $\sigma(\cdot)$ is the sigmoid function. By using Eq. (6), we can obtain the mask where the first l_i dimensions are closed to 1, and the last $(l - l_i)$ dimensions are closed to 0. As λ increases, the closer the soft mask is to the 0-1 mask. We denote all the masks as $\boldsymbol{M} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_K\}^{\mathrm{T}}$, where $\boldsymbol{m}_i \in \mathbb{R}^{l \times 1}$ denote the mask vector of the i-th filter. Then, the variable-length filters are generated as follow:

$$\boldsymbol{W}^v = \boldsymbol{W}^f \otimes \boldsymbol{M}, \tag{7}$$

where $\boldsymbol{W}^v = \{\boldsymbol{W}_1^v, \boldsymbol{W}_2^v, \cdots, \boldsymbol{W}_K^v\}$, $\boldsymbol{W}_i^v \in \mathbb{R}^{l \times 1}$ denotes the i-th variable-length filter, and $\otimes$ is the multiply operator.

TABLE 1. The details of 85 UCR datasets. N_{max}/N_{min} is the ratio between the numbers of samples of most and least frequent classes.

Name	Dataset type	#Train	#Test	#Class	Length	N_{max}/N_{min}
Adiac	Image	390	391	37	176	3.75
ArrowHead	Image	36	175	3	251	1.00
Beef	Spectro	30	30	5	470	1.00
BeetleFly	Image	20	20	2	512	1.00
BirdChicken	Image	20	20	2	512	1.00
Car	Sensor	60	60	4	577	1.55
CBF	Simulated	30	900	3	128	1.50
Chlorine	Sensor	467	3840	3	166	2.88
CinCECGTorso	Sensor	40	1380	4	1639	2.60
Coffee	Spectro	28	28	2	286	1.00
Computers	Device	250	250	2	720	1.00
CricketX	Motion	390	390	12	300	1.56
CricketY	Motion	390	390	12	300	1.29
CricketZ	Motion	390	390	12	300	1.64
DiatomSizeR	Image	16	306	4	345	6.00
DisPhxAgeGp	Image	139	400	3	80	4.33
DisPhxCorr	Image	276	600	2	80	1.40
DisPhxTW	Image	139	400	6	80	5.25
Earthquakes	Sensor	139	322	2	512	2.97
ECG200	ECG	100	100	2	96	2.23
ECG5000	ECG	500	4500	5	140	146.00
ECGFiveDays	ECG	23	861	2	136	1.56
ElectricDevices	Device	8926	7711	7	96	4.73
FaceAll	Image	560	1690	14	131	1.00
FaceFour	Image	24	88	4	350	2.67
FacesUCR	Image	200	2050	14	131	8.25
FiftyWords	Image	450	455	50	270	52.00
Fish	Image	175	175	7	463	1.33
FordA	Sensor	1320	3601	2	500	1.07
FordB	Sensor	810	3636	2	500	1.02
GunPoint	Motion	50	150	2	150	1.08
Ham	Spectro	109	105	2	431	1.10
HandOutlines	Image	370	1000	2	2709	1.78
Haptics	Motion	155	308	5	1092	2.00
Herring	Image	64	64	2	512	1.56
InlineSkate	Motion	100	550	7	1882	2.00
InsectWing	Sensor	220	1980	11	256	1.00
ItalyPower	Sensor	67	1029	2	24	1.03
LrgKitApp	Device	375	375	3	720	1.00
Lightning2	Sensor	60	61	2	637	22.00
Lightning7	Sensor	70	73	7	319	73.80
Mallat	Simulated	55	2345	8	1024	5.50
Meat	Spectro	60	60	3	448	1.00
MedicalImages	Image	381	760	10	99	33.83
MidPhxAgeGrp	Image	154	400	3	80	3.03
MidPhxCorr	Image	291	600	2	80	1.33
MidPhxTW	Image	154	399	6	80	4.89
MoteStrain	Sensor	20	1252	2	84	1.00
NonInv_Thor1	ECG	1800	1965	42	750	11.60
NonInv_Thor2	ECG	1800	1965	42	750	21.60
OliveOil	Spectro	30	30	4	570	3.25
OSULeaf	Image	200	242	6	427	3.53
PhalCorr	Image	1800	858	2	80	1.87
Phoneme	Sensor	214	1896	39	1024	24.00
Plane	Sensor	105	105	7	144	2.22
ProxPhxAgeGp	Image	400	205	3	80	2.63
ProxPhxCorr	Image	600	291	2	80	2.09
ProxPhxTW	Image	205	400	6	80	36.00
RefrigerationDevices	Device	375	375	3	720	1.00
ScreenType	Device	375	375	3	720	1.00
ShapeletSim	Simulated	20	180	2	500	1.00
ShapesAll	Image	600	600	60	512	1.00
SmlKitApp	Device	375	375	3	720	1.00
SonyAIBORobot1	Sensor	20	601	2	70	2.33
SonyAIBORobot2	Sensor	27	953	2	65	1.45
StarLightCurves	Sensor	1000	8236	3	1024	3.77
Strawberry	Spectro	370	613	2	235	1.80
SwedishLeaf	Image	500	625	15	128	1.62
Symbols	Image	25	995	6	398	2.67
SyntheticControl	Simulated	300	300	6	60	1.00
ToeSegmentation1	Motion	40	228	2	277	1.00
ToeSegmentation2	Motion	36	130	2	343	1.00
Trace	Sensor	100	100	4	275	1.48
TwoLeadECG	ECG	23	1139	2	82	1.09
TwoPatterns	Simulated	1000	4000	4	128	1.14
UWaveGestAll	Motion	896	3582	8	945	1.27
UWaveGest_X	Motion	896	3582	8	315	1.27
UWaveGest_Y	Motion	896	3582	8	315	1.27
UWaveGest_Z	Motion	896	3582	8	315	1.27
Wafer	Sensor	1000	6164	2	152	9.31
Wine	Spectro	57	54	2	234	1.11
WordSynonyms	Image	267	638	25	270	30.00
Worms	Motion	77	181	5	900	4.13
WormsTwoClass	Motion	77	181	2	900	1.33
Yoga	Image	300	3000	2	426	1.19

B. CONVOLUTIONAL LAYER WITH VARIABLE-LENGTH FILTERS

The variable-length filters are obtained with the variable-length filters generator. However, if the variable-length filters are generated by the masks, there will be some zeros in the last part of each filter, which hinders the convolution of time series. To solve this problem, we pad the input series with zeros up to the length of $(L + l)$, and denote as:

$$\boldsymbol{T}^p = \{x_1, x_2, \cdots, x_L, \boldsymbol{0}\}^{\mathrm{T}}, \tag{8}$$

where $\boldsymbol{T}^p \in \mathbb{R}^{(L+l)\times 1}$ denotes the time series after zero padding, and $\boldsymbol{0} \in \mathbb{R}^{l\times 1}$ is the zero vector. The zero padding will not affect DMS-CNN due to the max-over-time pooling layer in DMS-CNN, but retains the complete information of all time series.

We apply the variable-length filters $\boldsymbol{W}^v$ to $\boldsymbol{T}^p$, and the convolutional result with filter $\boldsymbol{W}_i^v$ on $\boldsymbol{T}^p$ can be denote as:

$$\boldsymbol{d}_i = \boldsymbol{W}_i^v * \boldsymbol{T}^p + b_i^v, \tag{9}$$

where $\boldsymbol{D} = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_K\}^{\mathrm{T}}$ denote all the convolutional results with variable-length filters $\boldsymbol{W}^v$ on $\boldsymbol{T}^p$, and b_i^v denote the bias.

To select the most important local pattern, we employ the max-over-time pooling. In this way, the pooling result of $\boldsymbol{d}_i$ can be defined as $p_i = \max(\boldsymbol{d}_i)$, and we collect all the pooling results into a single vector as:

$$\boldsymbol{P} = \{p_1, p_2, \cdots, p_K\}^{\mathrm{T}}. \tag{10}$$

Finally, the pooling results are fed into a softmax layer to obtain the conditional distribution over each category label as follows:

$$\hat{\boldsymbol{Y}} = \boldsymbol{W}^o\boldsymbol{P} + \boldsymbol{b}^o, \tag{11}$$

$$p(\boldsymbol{C}|\boldsymbol{T}) = softmax(\hat{\boldsymbol{Y}}), \tag{12}$$

where $\boldsymbol{W}^o \in \mathbb{R}^{K\times c}$ and $\boldsymbol{b}^o \in \mathbb{R}^c$ denote the weights and bias of the *softmax* layer, respectively, and $\hat{\boldsymbol{Y}}$ denotes the output vector. $\boldsymbol{C}$ denotes the classes of the input time series and $p(\boldsymbol{C}|\boldsymbol{U})$ is the conditional distribution over time series label.

C. TRAINING

We use $\mathcal{D}_{tr} = \{\boldsymbol{T}_1, \boldsymbol{T}_2, \cdots, \boldsymbol{T}_{N_{tr}}\}$ and $\mathcal{Y}_{tr} = \{\boldsymbol{Y}_1, \boldsymbol{Y}_2, \cdots, \boldsymbol{Y}_{N_{tr}}\}$ to represent the input time series and corresponding labels of the training set, respectively, where $\boldsymbol{T}_i \in \mathbb{R}^{L\times 1}$ denotes the i-th input time series, and $\boldsymbol{Y}_i \in \mathbb{R}^c$ denotes its label. N_{tr} and c are the number of samples in the training set and the number of categories, respectively.

Therefore, the loss function of the DMS-CNN is the negative logarithm likelihood function as follow:

$$\mathcal{L}(\boldsymbol{T}) = -\frac{1}{N_{tr}} \sum_{i=1}^{N_{tr}} \sum_{r=1}^{c} \mathbf{1}\{\boldsymbol{Y}_{i,r}=1\} log \frac{exp(\hat{\boldsymbol{Y}}_{i,r})}{\sum_{j=1}^{c} exp(\hat{\boldsymbol{Y}}_{i,j})}, \tag{13}$$

where $\mathcal{Y}_{out} = \{\hat{\boldsymbol{Y}}_1, \hat{\boldsymbol{Y}}_2, \cdots, \hat{\boldsymbol{Y}}_{N_{tr}}\}$ denote the predicted label of the DMS-CNN, and $\hat{\boldsymbol{Y}}_i \in \mathbb{R}^c$ is the predicted label of the i-th input time series.

Algorithm 1 Dynamic Multi-Scale Convolutional Neural Network

Input:

$\boldsymbol{T}$: input time series
K: number of variable-length filters
l: sliding window size
λ: the sharpness parameter

Output:

$p(\boldsymbol{C}|\boldsymbol{T})$: conditional distribution

Initialize the fixed-length filters $\boldsymbol{W}_f = \{\boldsymbol{W}_1^f, \boldsymbol{W}_2^f, \cdots, \boldsymbol{W}_K^f\}^{\mathrm{T}}$
Obtain the subsequences $\boldsymbol{S}$ of $\boldsymbol{T}$ with a sliding window length l and stride of 1
Compute the embedded representation $\boldsymbol{E} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots \boldsymbol{e}_K\}^{\mathrm{T}}$ using Eq. (2)
for $i = 1$ to K **do**
  $r_i = \boldsymbol{W}^{m2} f(\boldsymbol{W}^{m1} \cdot (\boldsymbol{e}_i \oplus \boldsymbol{W}_i^f) + \boldsymbol{b}^{m1}) + b^{m2}$
  $l_i = r_i \times l$
  for $j = 1$ to l **do**
    $m_{ij} = \sigma(\lambda(l_i - j))$
  end for
end for
Obtain the mask matrix: $\boldsymbol{M} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_K\}$
Compute the variable-length filters: $\boldsymbol{W}^v = \boldsymbol{W}^f \otimes \boldsymbol{M}$
Compute the convolutional result $\boldsymbol{D}$ with variable-length filters using Eq. (9)
for $i = 1$ to K **do**
  $p_i = \max(\boldsymbol{d}_i)$
end for
Obtain the pooling results: $\boldsymbol{P} = \{p_1, p_2, \cdots, p_K\}$
Compute conditional distribution $p(\boldsymbol{C}|\boldsymbol{T})$ using Eq. (11) and Eq. (12)
return $p(\boldsymbol{C}|\boldsymbol{T})$

The ADAM [25] optimizer is used to train DMS-CNN model with an initial learning rate of 0.01. To make the training of the model stable, the weight attenuation is adopted to train the model. Specifically, we attenuated the learning rate by 0.8 after every 50 epochs. Algorithm 1 shows the pseudo-code of Dynamic Multi-scale CNNs in detail.

IV. EXPERIMENTS

To evaluate the performance of DMS-CNN, experiments are conducted on the UCR time series classification archive to compare DMS-CNN with other methods. All the experiments are run on an Intel Core i7-6850K 3.60GHz CPU, 64GB RAM, and a GeForce GTX 1080-Ti 11G GPU.

The classification accuracy is used to evaluate the performance and defined by:

$$\text{Accuracy} = \frac{\#\text{correct classified samples}}{\#\text{testing samples}} \tag{14}$$

A. DATASET INTRODUCTION AND IMPLEMENTATION DETAILS

The UCR time series classification archive [26] contains 85 publicly available time series datasets, which vary by the number of classes, dataset types, number of samples, and length of time series. Each dataset was split into training and testing set using the standard split. Table 1 summarizes the details of the 85 datasets. We also use a histogram to describe the distribution of ratio ($N_{\max}/N_{\min}$) between the numbers of samples of most and least frequent classes for 85 datasets in Figure 4.

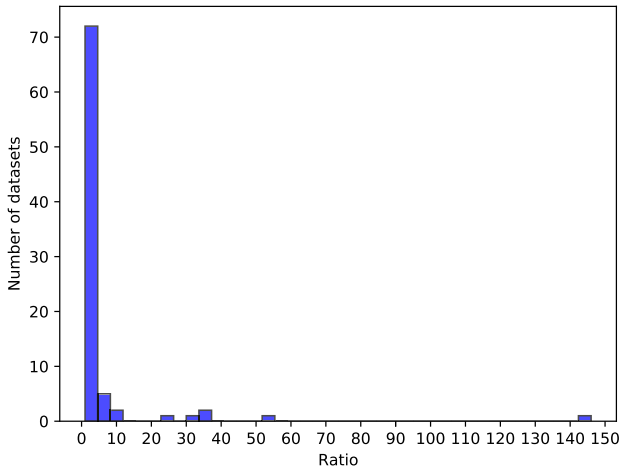


FIGURE 4. The distribution of ratio ($N_{\max}/N_{\min}$) between the numbers of samples of most and least frequent classes for 85 datasets. The X-axis and Y-axis indicate the ratio and number of datasets, respectively.

DMS-CNN involves the selection of several hyperparameters, including slice window size l, the length of filters to obtain the embedded representation of time series ω, the number of variable-length filters K, and the sharpness parameter λ. In our experiments, the slicing window sizes l are also the lengths of fixed-length filters, which are chosen according to the length of the time series. We set l as $0.4L$, where L denotes the length of the input time series. ω is fixed to 3. K is chosen from $K \in \{30, 60, 90, 120\}$. The sharpness parameter λ is chosen from $\lambda \in \{5, 10\}$. In addition, to improve the generalization capability, we apply dropout [27] and batch normalization [28] to the inputs of the softmax layer and the convolutional layer with variable-length filters respectively. The dropout rate is set to 0.2.

B. COMPARISON METHODS

For making a comprehensive evaluation, we first compare DMS-CNN with 12 representative machine learning methods. Then DMS-CNN is compared with several recent deep learning models.

For traditional machine learning methods, we compare DMS-CNN with two classical yet powerful baseline methods 1-Nearest Neighbor with Euclidean distance (1NN-ED) and 1-Nearest Neighbor with Dynamic Time Warping (1NN-DTW). In addition, ten state-of-the-art methods published in recent years are selected as the compared baselines. These methods can be divided into four categories: distance-based methods, feature-based methods, statistical time series methods, and ensemble-based methods. All the

TABLE 2. Accuracy of DMS-CNN compared with 12 conventional classifiers on 85 UCR datasets. The best accuracy is indicated as bold. ST is an ensemble of 8 classifiers, EE consists of 11 1-NN classifiers and COTE is an ensemble model with 35 different base models.

Datasets	ED	DTW	DD_DTW	DTD_C	LS	BOSS	LPS	TSF	TSBF	ST(8)	EE(11)	COTE(35)	DMS-CNNs
Adiac	0.611	0.604	0.701	0.701	0.522	0.765	0.770	0.731	0.770	0.783	0.665	**0.790**	0.691
ArrowHead	0.800	0.703	0.789	0.720	0.846	0.834	0.783	0.726	0.754	0.737	0.811	0.811	0.851
Beef	0.667	0.633	0.667	0.667	0.867	0.800	0.600	0.767	0.567	**0.900**	0.633	0.867	**0.733**
BeetleFly	0.750	0.700	0.650	0.650	0.800	**0.900**	0.800	0.750	0.800	**0.900**	0.750	0.800	0.850
BirdChicken	0.550	0.750	0.850	0.800	0.800	0.950	**1.000**	0.800	0.900	0.800	0.800	0.900	0.917
Car	0.733	0.733	0.800	0.783	0.767	0.833	0.850	0.767	0.783	**0.917**	0.833	0.900	0.867
CBF	0.852	0.997	0.997	0.980	0.991	0.998	**0.999**	0.994	0.988	0.974	0.998	0.996	0.998
Chlorine	0.650	0.648	0.708	0.713	0.592	0.661	0.608	0.720	0.692	0.700	0.656	0.727	**0.790**
CinC_ECG_torso	0.897	0.651	0.725	0.852	0.870	0.887	0.736	0.983	0.712	0.954	0.942	**0.995**	0.743
Coffee	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.964	**1.000**	0.964	**1.000**	**1.000**	**1.000**
Computers	0.576	0.700	0.716	0.716	0.584	**0.756**	0.680	0.720	**0.756**	0.736	0.708	0.740	0.683
CricketX	0.577	0.754	0.754	0.754	0.741	0.736	0.697	0.664	0.705	0.772	**0.813**	0.808	0.762
CricketY	0.567	0.744	0.777	0.774	0.718	0.754	0.767	0.672	0.736	0.779	0.805	**0.826**	0.756
CricketZ	0.587	0.754	0.774	0.774	0.741	0.746	0.754	0.672	0.715	0.787	0.782	**0.815**	0.760
DiatomSizeR	0.935	0.967	0.967	0.915	0.980	0.931	0.905	0.931	0.899	0.925	0.944	0.928	**0.987**
DistPhxAgeGp	0.626	0.770	0.705	0.662	0.719	0.748	0.669	0.748	0.712	0.770	0.691	0.748	**0.853**
DistPhxCorr	0.717	0.717	0.732	0.725	0.779	0.728	0.721	0.772	0.783	0.775	0.728	0.761	**0.802**
DistPhxTW	0.633	0.590	0.612	0.576	0.626	0.676	0.568	0.669	0.676	0.662	0.647	0.698	**0.798**
Earthquakes	0.712	0.719	0.705	0.705	0.741	0.748	0.640	0.748	0.748	0.741	0.741	0.748	**0.820**
ECG200	0.880	0.770	0.830	0.840	0.880	0.870	0.860	0.870	0.840	0.830	0.880	0.880	**0.883**
ECG5000	0.925	0.924	0.924	0.924	0.932	0.941	0.917	0.939	0.940	0.944	0.939	**0.946**	0.944
ECGFiveDays	0.797	0.768	0.769	0.822	**1.000**	**1.000**	0.879	0.956	0.877	0.984	0.820	0.999	**1.000**
ElectricDevices	0.552	0.602	0.592	0.594	0.587	**0.799**	0.681	0.693	0.703	0.747	0.663	0.713	0.693
FaceAll	0.714	0.808	0.902	0.899	0.749	0.782	0.767	0.751	0.744	0.779	0.849	**0.918**	0.822
FaceFour	0.784	0.830	0.830	0.818	0.966	**1.000**	0.943	0.932	**1.000**	0.852	0.909	0.898	0.822
FacesUCR	0.769	0.905	0.904	0.908	0.939	**0.957**	0.926	0.883	0.867	0.906	0.945	0.942	0.896
FiftyWords	0.631	0.690	0.754	0.754	0.730	0.705	0.818	0.741	0.758	0.705	**0.820**	0.798	0.738
Fish	0.783	0.823	0.943	0.926	0.960	**0.989**	0.943	0.794	0.834	**0.989**	0.966	0.983	0.924
FordA	0.665	0.555	0.723	0.765	0.957	0.930	0.873	0.815	0.850	**0.971**	0.738	0.957	0.913
FordB	0.606	0.620	0.667	0.653	**0.917**	0.711	0.711	0.688	0.599	0.807	0.662	0.804	0.889
Gun-Point	0.913	0.907	0.980	0.987	**1.000**	**1.000**	0.993	0.973	0.987	**1.000**	0.993	**1.000**	0.993
Ham	0.600	0.467	0.476	0.552	0.667	0.667	0.562	0.743	**0.762**	0.686	0.571	0.648	0.752
HandOutlines	0.862	0.881	0.868	0.865	0.481	0.903	0.881	0.919	0.854	**0.932**	0.889	0.919	0.885
Haptics	0.370	0.377	0.399	0.399	0.468	0.461	0.432	0.445	0.490	**0.523**	0.393	**0.523**	0.478
Herring	0.516	0.531	0.547	0.547	0.625	0.547	0.578	0.609	0.641	**0.672**	0.578	0.625	0.656
InlineSkate	0.342	0.384	**0.562**	0.509	0.438	0.516	0.500	0.376	0.385	0.373	0.460	0.495	0.427
InsectWing	0.562	0.355	0.355	0.473	0.606	0.523	0.551	0.633	0.625	0.627	0.595	**0.653**	0.638
ItalyPower	0.955	0.950	0.950	0.951	0.960	0.909	0.923	0.960	0.883	0.948	0.962	0.961	**0.970**
LrgKitApp	0.493	0.795	0.795	0.795	0.701	0.765	0.717	0.571	0.528	0.859	0.811	0.845	**0.880**
Lightning-2	0.754	0.869	0.869	0.869	0.820	0.836	0.820	0.803	0.738	0.738	**0.885**	0.869	0.842
Lightning-7	0.575	0.726	0.671	0.658	0.795	0.685	0.740	0.753	0.726	0.726	0.767	0.808	**0.836**
MALLAT	0.914	0.934	0.949	0.927	0.950	0.938	0.908	0.919	0.960	**0.964**	0.940	0.954	0.948
Meat	0.933	0.933	0.933	0.933	0.733	0.900	0.883	0.933	0.933	0.850	0.933	0.917	**0.944**
MedicalImages	0.684	0.737	0.737	0.745	0.664	0.718	0.746	0.755	0.705	0.670	0.742	**0.758**	0.739
MidPhxAgeGp	0.519	0.500	0.539	0.500	0.571	0.545	0.487	0.578	0.578	0.643	0.558	0.636	**0.800**
MidPhxCorr	0.766	0.698	0.732	0.742	0.780	0.780	0.773	**0.828**	0.814	0.794	0.784	0.804	0.549
MidPhxTW	0.513	0.506	0.487	0.500	0.506	0.545	0.526	0.565	0.597	0.519	0.513	0.571	**0.629**
MoteStrain	0.879	0.835	0.833	0.768	0.883	0.879	0.922	0.869	0.903	0.897	0.883	**0.937**	0.869
NonInv_Thor1	0.829	0.790	0.806	0.841	0.259	0.838	0.812	0.876	0.842	**0.950**	0.846	0.931	0.945
NonInv_Thor2	0.880	0.865	0.893	0.890	0.770	0.901	0.841	0.910	0.862	**0.951**	0.913	0.946	0.949
OliveOil	0.867	0.833	0.833	0.867	0.167	0.867	0.867	0.867	0.833	**0.900**	0.867	**0.900**	0.422
OSULeaf	0.521	0.591	0.880	0.884	0.777	0.955	0.740	0.583	0.760	**0.967**	0.806	**0.967**	0.726
PhalCorr	0.761	0.728	0.739	0.761	0.765	0.772	0.756	0.803	**0.830**	0.763	0.773	0.770	0.809
Phoneme	0.109	0.228	0.269	0.268	0.218	0.265	0.237	0.212	0.276	0.321	0.305	**0.349**	0.153
Plane	0.962	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
ProxPhxAgeGp	0.785	0.805	0.800	0.795	0.834	0.834	0.795	0.849	0.849	0.844	0.805	**0.854**	**0.854**
ProxPhxCorr	0.808	0.784	0.794	0.794	0.849	0.849	0.842	0.828	0.873	0.883	0.808	0.869	**0.893**
ProxPhxTW	0.707	0.761	0.766	0.771	0.776	0.800	0.732	**0.815**	0.810	0.805	0.766	0.780	**0.815**
RefrigerationDevices	0.395	0.464	0.445	0.445	0.515	0.499	0.459	**0.589**	0.472	0.581	0.437	0.547	0.492
ScreenType	0.360	0.397	0.429	0.437	0.429	0.464	0.416	0.456	0.509	0.520	0.445	**0.547**	0.458
ShapeletSim	0.539	0.650	0.611	0.600	0.950	**1.000**	0.867	0.478	0.961	0.956	0.817	0.961	0.922
ShapesAll	0.752	0.768	0.850	0.838	0.768	**0.908**	0.873	0.792	0.185	0.842	0.867	0.892	0.842
SmlKitApp	0.344	0.643	0.640	0.648	0.664	0.725	0.712	**0.811**	0.672	0.792	0.696	0.776	0.673
SonyAIBORobot1	0.696	0.725	0.742	0.710	0.810	0.632	0.774	0.787	0.795	0.844	0.704	0.845	**0.961**
SonyAIBORobot2	0.859	0.831	0.892	0.892	0.875	0.859	0.872	0.810	0.778	0.934	0.878	**0.952**	0.887
StarlightCurves	0.849	0.907	0.962	0.962	0.947	0.978	0.963	0.969	0.977	0.979	0.926	**0.980**	0.975
Strawberry	0.946	0.941	0.954	0.957	0.911	**0.976**	0.962	0.965	0.954	0.962	0.946	0.951	0.969
SwedishLeaf	0.789	0.792	0.901	0.896	0.907	0.922	0.920	0.914	0.915	0.928	0.915	**0.955**	0.929
Symbols	0.899	0.950	0.953	0.963	0.932	**0.967**	0.963	0.915	0.946	0.882	0.960	0.964	0.938
SyntheticControl	0.880	0.993	0.993	0.997	0.997	0.967	0.980	0.987	0.993	0.983	0.990	**1.000**	**1.000**
ToeSegmentation1	0.680	0.772	0.807	0.807	0.934	0.939	0.877	0.741	0.781	0.965	0.829	**0.974**	0.934
ToeSegmentation2	0.808	0.838	0.746	0.715	0.915	**0.962**	0.869	0.815	0.800	0.908	0.892	0.915	0.931
Trace	0.760	**1.000**	**1.000**	0.990	**1.000**	**1.000**	0.980	0.990	0.980	**1.000**	0.990	**1.000**	**1.000**
TwoLeadECG	0.747	0.905	0.978	0.985	0.996	0.981	0.948	0.759	0.866	**0.997**	0.971	0.993	**0.997**
TwoPatterns	0.907	**1.000**	**1.000**	**1.000**	0.993	0.993	0.982	0.991	0.976	0.955	**1.000**	**1.000**	0.999
UWaveGestAll	0.948	0.892	0.935	0.938	0.953	0.939	0.966	0.957	0.926	0.942	**0.968**	0.964	0.946
UWaveGest_X	0.739	0.728	0.779	0.775	0.791	0.762	0.829	0.804	**0.831**	0.803	0.805	0.822	0.791
UWaveGest_Y	0.662	0.634	0.716	0.698	0.703	0.685	**0.761**	0.727	0.736	0.730	0.726	0.759	0.706
UWaveGest_Z	0.650	0.658	0.696	0.679	0.747	0.695	0.768	0.743	**0.772**	0.748	0.724	0.750	0.742
Wafer	0.995	0.980	0.980	0.993	0.996	0.995	0.997	0.996	0.995	**1.000**	0.997	**1.000**	0.998
Wine	0.611	0.574	0.574	0.611	0.500	0.741	0.630	0.630	0.611	**0.796**	0.574	0.648	0.556
WordSynonyms	0.618	0.649	0.730	0.730	0.607	0.638	0.755	0.647	0.688	0.571	**0.779**	0.757	0.654
Worms	0.455	0.584	0.584	0.649	0.610	0.558	0.701	0.610	0.688	**0.740**	0.662	0.623	0.552
WormsTwoClass	0.610	0.623	0.649	0.623	0.727	**0.831**	0.753	0.623	0.753	**0.831**	0.688	0.805	0.766
Yoga	0.830	0.837	0.856	0.856	0.834	**0.918**	0.869	0.859	0.819	0.818	0.879	0.877	0.850

TABLE 3. Statistical results of DMS-CNN and 12 conventional classifiers on 85 UCR datasets.

Methods	ED	DTW	DD_{DTW}	DTD_C	LS	BOSS	LPS	TSF	TSBF	ST(8)	EE(11)	COTE(35)	DMS-CNN
Avg rank	10.906	9.835	8.194	8.359	7.335	5.747	7.312	7.035	7.018	4.988	6.229	3.176	4.865
Best	1	4	5	3	6	18	5	5	8	21	8	26	24

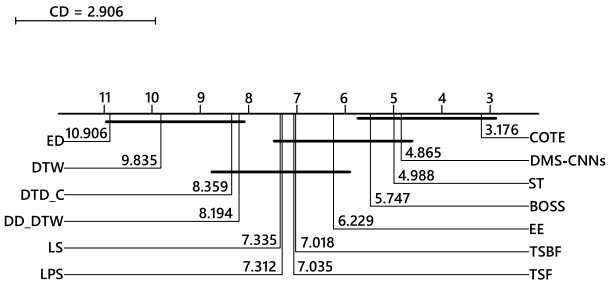


FIGURE 5. Critical difference diagram of the comparison with traditional machine learning methods. The critical difference is 2.906, which means that two classifiers are not significantly different at $p < 0.05$ level when the rank difference is less than 2.906.

experimental results of these methods are collect by Bagnall *et al.* [29].

- **Distance-based methods:** In addition to two classic methods 1NN-ED [5], and 1NN-DTW [5], two distance-based methods are selected, including derivative DTW (DD_{DTW}) [6], and derivative transform distance (DTD_C) [30]. DD_{DTW} is a method that uses the weighted combination of the DTW distance between two time series and the DTW distance between their corresponding first-order difference sequences. Based on DD_{DTW}, DTD_C takes into account the DTW distance of the sequences transformed by sin, cosine and Hilbert transformation.
- **Feature-based methods:** Three feature-based methods are selected, including learned shapelet (LS) [8], bag of SFA symbols (BOSS) [7], and learned pattern similarity (LPS) [9].
- **Statistical time series methods:** Two statistical time series methods are selected, including time series forest (TSF) [11] and time series bag of features (TSBF) [12]. The TSF classifies a time series by the random forest using some statistical features, such as mean, standard deviation, and slope. The TSBF is an extension of TSF that has multiple stages.
- **Ensemble-based methods:** These methods include Shapelet Transform (ST) [15], Elastic Ensemble (EE) [14] and the collective of transformation-based ensembles (COTE) [16]. ST uses shapelet transformation based on a heterogeneous ensemble. EE is an ensemble classifier with 1NN based on 11 elastic distance measures. COTE ensembles 35 different classifiers constructed in the domain of time, frequency, change, and shapelet transformation, respectively.

For deep learning methods, many effective deep learning models have been proposed and applied to TSC tasks. We compare DMS-CNN with Multilayer Perceptron (MLP), Residual Network (ResNet), and Fully Convolutional Network (FCN). The experimental results of the three methods are provided by Wang *et al.* [17]. The brief introduction to these three methods is given below.

- **Multilayer Perceptron (MLP)** stack three fully-connected layers with 500 neurons for each layer, and the softmax layer is used to obtain the classification results.
- **Residual Network (ResNet)** stack three residual blocks, each of which contains three convolution block. The number of filters in the three residual blocks are 64, 128, and 128, respectively. The global average pooling layer and a softmax layer are used to obtain classification results.
- **Fully Convolutional Network (FCN)** stack three convolution blocks with 128, 256, and 128 filters in each block, where the filter size in each block is 3, 5, and 8. the convolutional results are fed into the global average pooling layer and a softmax layer to get the classification results.

In addition, we construct a baseline model to verify the effectiveness of the variable-length filters. The architecture and hyperparameters of the baseline model are consistent with DMS-CNN. The only difference is that the baseline model uses fixed-length filters.

C. COMPARISON WITH TRADITIONAL METHODS

For traditional methods, the accuracy of DMS-CNN and 12 traditional methods on 85 UCR datasets are shown in Table 2. In addition, we also summarize the average rank of each method and the number of datasets on which each method achieves the best results in Table 3.

As shown in Table 3, DMS-CNN achieves much higher accuracy than any of the other methods on 24 of the 85 datasets and the average rank of 4.865. Compared to the distance-based methods, DMS-CNN is superior to these methods. Although DD_{DTW} achieves the best performance in the distance-based method, it only achieves the best results on 5 of the 85 datasets. Similarly, the performance of DMS-CNN is better than three feature-based methods and two statistical time series methods. The ensemble-based methods achieve good results on 85 UCR datasets. For example, COTE achieves the best results on 26 of the 85 datasets and the best average rank of 3.176, which is superior to DMS-CNN. However, COTE ensembles 35 different classifiers and thus inevitably suffers from high computational complexity. Even if the ensemble-based methods ensemble multiple classifiers, DMS-CNN still numerically superior in

TABLE 4. Accuracy of DMS-CNN compared with baseline and 3 deep learning methods on 85 UCR datasets. The best results are marked as bold.

Datasets	MLP	ResNet	FCN	Baseline	DMS-CNN
Adiac	0.752	0.826	**0.857**	0.720	0.691
ArrowHead	0.823	0.817	**0.880**	0.827	0.851
Beef	**0.833**	0.767	0.750	0.733	0.733
BeetleFly	0.850	0.800	**0.950**	0.817	0.850
BirdChicken	0.800	0.900	**0.950**	0.850	0.917
Car	0.833	**0.933**	0.917	0.844	0.867
CBF	0.860	0.994	**1.000**	0.995	0.998
Chlorine	**0.872**	0.828	0.843	0.848	0.790
CinC_ECG_torso	**0.842**	0.771	0.813	0.733	0.743
Coffee	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Computers	0.540	0.824	**0.848**	0.704	0.683
CricketX	0.569	**0.821**	0.815	0.750	0.762
CricketY	0.595	**0.805**	0.792	0.753	0.756
CricketZ	0.592	**0.813**	**0.813**	0.767	0.760
DiatomSizeR	0.964	0.931	0.930	0.951	**0.987**
DistPhxAgeGp	0.827	0.798	0.835	**0.853**	**0.853**
DistPhxCorr	0.810	**0.820**	0.812	0.808	0.802
DistPhxTW	0.747	0.740	0.790	0.785	**0.798**
Earthquakes	0.792	0.786	0.801	**0.820**	**0.820**
ECG200	**0.920**	0.870	0.900	0.873	0.883
ECG5000	0.935	0.931	0.941	0.943	**0.944**
ECGFiveDays	0.970	0.955	0.985	**1.000**	**1.000**
ElectricDevices	0.580	**0.728**	0.723	0.671	0.693
FaceAll	0.885	0.834	**0.929**	0.770	0.822
Face (four)	0.830	**0.932**	**0.932**	0.860	0.822
FacesUCR	0.815	**0.958**	0.948	0.899	0.896
FiftyWords	0.712	0.727	0.679	0.722	**0.738**
Fish	0.874	**0.989**	0.971	0.935	0.924
FordA	0.769	**0.928**	0.906	0.923	0.914
FordB	0.629	**0.900**	0.883	0.883	0.889
Gun-Point	0.933	0.993	**1.000**	0.991	0.993
Ham	0.714	**0.781**	0.762	0.705	0.752
HandOutlines	0.807	0.861	0.776	**0.893**	0.885
Haptics	0.461	0.506	**0.551**	0.440	0.478
Herring	0.687	0.594	**0.703**	0.620	0.656
InlineSkate	0.351	0.365	0.411	0.403	**0.427**
InsectWing	0.631	0.531	0.402	0.635	**0.638**
ItalyPower	0.966	0.960	**0.970**	0.969	**0.970**
LrgKitApp	0.480	0.893	**0.896**	0.825	0.880
Lightning-2	0.721	0.754	0.803	0.836	**0.842**
Lightning-7	0.644	0.836	**0.863**	0.817	0.836
MALLAT	0.936	0.979	**0.980**	0.931	0.948
Meat	0.933	**1.000**	0.967	0.922	0.944
MedicalImages	0.729	0.772	**0.792**	0.729	0.739
MidPhxAgeGp	0.735	0.760	0.768	**0.800**	**0.800**
MidPhxCorr	0.760	0.793	**0.795**	0.739	0.549
MidPhxTW	0.609	0.607	0.612	**0.642**	0.629
MoteStrain	0.869	0.895	**0.950**	0.882	0.869
NonInv_Thor1	0.942	0.948	**0.961**	0.921	0.945
NonInv_Thor2	0.943	0.951	**0.955**	0.938	0.949
OliveOil	0.400	**0.867**	0.833	0.400	0.422
OSULeaf	0.570	0.979	**0.988**	0.628	0.726
PhalCorr	**0.830**	0.825	0.826	0.800	0.809
Phoneme	0.098	0.324	**0.345**	0.151	0.153
Plane	0.981	**1.000**	**1.000**	**1.000**	**1.000**
ProxPhxAgeGp	0.824	0.849	0.849	**0.854**	**0.854**
ProxPhxCorr	0.887	**0.918**	0.900	0.913	0.893
ProxPhxTW	0.797	0.807	0.810	0.810	**0.815**
RefrigerationDevices	0.371	0.528	**0.533**	0.462	0.492
ScreenType	0.408	**0.707**	0.667	0.425	0.458
ShapeletSim	0.483	**1.000**	0.867	0.822	0.922
ShapesAll	0.775	**0.912**	0.898	0.828	0.842
SmlKitApp	0.389	0.797	**0.803**	0.664	0.673
SonyAIBORobot1	0.727	**0.985**	0.968	0.956	0.961
SonyAIBORobot2	0.839	**0.962**	**0.962**	0.885	0.887
StarLightCurves	0.957	**0.975**	0.967	0.972	**0.975**
Strawberry	0.967	0.958	**0.969**	0.968	**0.969**
SwedishLeaf	0.893	0.958	**0.966**	0.924	0.929
Symbols	0.853	0.872	**0.962**	0.915	0.938
SyntheticControl	0.950	**1.000**	0.990	**1.000**	**1.000**
ToeSegmentation1	0.601	0.965	**0.969**	0.923	0.934
ToeSegmentation2	0.746	0.862	0.915	0.908	**0.931**
Trace	0.820	**1.000**	**1.000**	**1.000**	**1.000**
TwoLeadECG	0.853	**1.000**	**1.000**	0.994	0.997
TwoPatterns	0.886	**1.000**	0.897	0.997	0.999
UWaveGestAll	**0.954**	0.868	0.826	0.940	0.946
UWaveGest_X	0.768	0.787	0.754	0.776	**0.791**
UWaveGest_Y	0.703	0.668	**0.725**	0.686	0.706
UWaveGest_Z	0.705	**0.755**	0.729	0.746	0.742
Wafer	0.996	0.997	0.997	0.997	**0.998**
Wine	0.796	0.796	**0.889**	0.728	0.556
WordSynonyms	0.594	0.632	0.580	0.633	**0.654**
Worms	0.343	0.619	**0.669**	0.501	0.552
WormsTwoClass	0.597	0.735	0.729	0.735	**0.766**
Yoga	0.855	**0.858**	0.845	0.803	0.850

TABLE 5. Statistical results of 3 deep learning methods, baseline and DMS-CNN on 85 UCR datasets.

Methods	MLP	ResNet	FCN	Baseline	DMS-CNN
Avg rank	4.124	2.600	2.253	3.371	2.653
Best	7	29	37	11	25

TABLE 6. Statistical results of 2 statistical methods, 3 deep learning methods and DMS-CNN on 85 UCR datasets.

Methods	TSF	TSBF	MLP	ResNet	FCN	DMS-CNN
Avg rank	4.153	4.053	4.618	2.788	2.429	2.959
Best	10	11	5	25	31	22

average rank to the ensemble-based methods (e.g., ST, EE) except for COTE. To have a more intuitive understanding of the results, we compare DMS-CNN with traditional methods in pairs. The pairwise accuracy plots between DMS-CNN and traditional methods are shown in the appendix. As shown in Figure 11-22, DMS-CNN obtains higher accuracy in the vast majority of datasets compared with the traditional classifiers except for COTE.

Furthermore, we conduct a non-parametric statistical test, the Nemenyi test [31] on the average ranks of the methods to make statistical comparisons. As shown in Figure 5, the critical difference is 2.906, which means that two classifiers are not significantly different at $p < 0.05$ level when the rank difference is less than 2.906. Therefore, we can conclude that DMS-CNN is significantly better than the distant-based methods and slightly better than the feature-based methods, the statistical time series methods, and the ensemble-based methods except for COTE. Although COTE ensembles 35 different classifiers, it has no statistically significant difference with DMS-CNN.

D. COMPARISON WITH DEEP LEARNING METHODS

For deep learning methods, the accuracy of DMS-CNN, baseline model, and three deep learning methods on 85 UCR datasets are shown in Table 4. In addition, we also summarize the average rank of each method and the number of datasets on which each method achieves the best results in Table 5.

As shown in Table 5, DMS-CNN achieves much higher accuracy than any of the other deep learning methods on 25 of the 85 datasets and the average rank of 2.653. The baseline model achieves the best results on 11 of 85 datasets, and the average rank is 3.371. Therefore, DMS-CNN is superior to the baseline model. This verifies that the variable-length filters can better capture the temporal feature of time series, thereby improving the accuracy of time series classification. Compared with the three deep learning methods, DMS-CNN is better than MLP and worse than ResNet and FCN. ResNet achieves the best results on 29 of 85 datasets, and the average rank is 2.6. Although ResNet uses multiple convolutional layers, the performance of DMS-CNN is very close to that of ResNet. Similarly, the pairwise accuracy plots between DMS-CNN and other deep learning methods are also shown in the Appendix. As shown in Figure 23-26, DMS-CNN

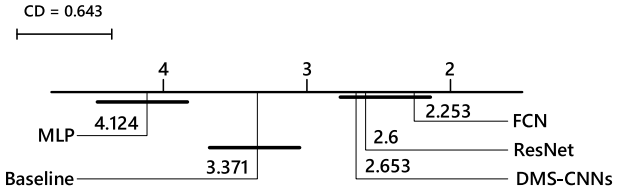


FIGURE 6. Critical difference diagram of the comparison with baseline and deep learning methods. The critical difference is 0.643, which means that two classifiers are not significantly different at $p < 0.05$ level when the rank difference is less than 0.643.

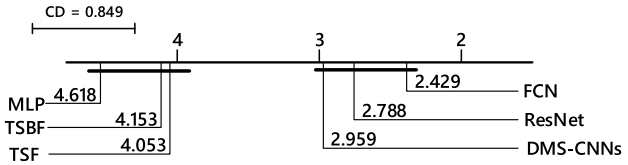


FIGURE 7. Critical difference diagram of the comparison with statistical methods and deep learning methods. The critical difference is 0.849, which means that two classifiers are not significantly different at $p < 0.05$ level when the rank difference is less than 0.849.

obtains higher accuracy in the vast majority of datasets compared with the MLP and baseline model. In addition, the comparisons are constructed to compare the traditional statistical methods with deep learning methods. As shown in Table 6, ResNet, FCN and DMS-CNN are superior to TSF, and TSBF, which indicates the deep learning methods have better performances than the traditional statistical methods.

Similarly, we also conduct the Nemenyi test on the average ranks of these five methods to make statistical comparisons. As shown in Figure 6, the critical difference is 0.643. From the Figure 6, we can draw the following conclusions. First, DMS-CNN, baseline, ResNet, and FCN are significantly better than MLP because convolution can better extract the temporal features of time series. Then, DMS-CNN is significantly better than the baseline model, which further verifies the effectiveness of the variable-length filters. Finally, although the results of DMS-CNN on the whole 85 UCR datasets can not beat ResNet and FCN, it has no statistically significant difference with them. For the comparisons between the traditional statistical methods with deep learning methods, as shown in Figure 7, ResNet, FCN and DMS-CNN are significantly better than the statistical methods, i.e., TSF and TSBF.

E. VISUALIZATION ANALYSIS

In this section, we conduct experiments on the ECGFiveDays dataset for the visualization analysis of the interpretability of the DMS-CNN. The visualization analysis consists of two parts. First, we will explore how DMS-CNN captures multi-scale temporal features. And then, we analyze how the dynamics of the filters affect DMS-CNN.

The multi-scale temporal information naturally exists in time series. For example, each cardiac cycle of the standard electrocardiogram (ECG) consists of a P wave, a QRS wave,

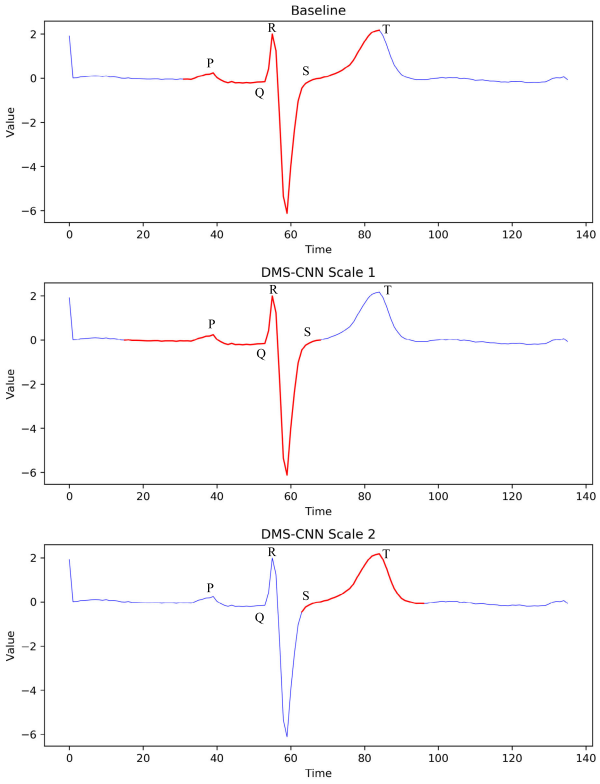


FIGURE 8. A time series sample that is form the ECGFiveDays dataset, and the important local patterns that are kept by the max-over-time pooling in the baseline model and DMS-CNN with different scales, respectively. We use the blue curves and red ones to indicate the time series samples and important local patterns, respectively. P, QRS and T are the critical waves in ECG. The X-axis and Y-axis are the time step and corresponding values, respectively.

and followed by a T wave [32]. In the DMS-CNN, the max-over-time pooling is used to select the most discriminative local patterns. Therefore, we can see whether the DMS-CNN has learned the multi-scale temporal information by the local patterns kept by max-over-time pooling. We visualize a sample in the ECGFiveDays dataset and the important local patterns that kept by the max-over-time pooling in Figure 8. As we can see, the baseline model uses the fixed-length filter, so it can only try its best to capture all the important local features, rather than learn each important local feature independently. Unlike the baseline model, DMS-CNN learns filters of two scales. The important local pattern captured by the filters of the first scale is P-wave and QRS wave and the filters of the second scale capture T-wave. Therefore, the important local features of time series may have different lengths. MDS-CNN can adaptively learn these features for each sample through the variable-length filters. In addition, CNNs can also use filters of multiple lengths like DMS-CNN to extract multi-scale temporal features of time series. However, the setting of the filter lengths depends on the prior knowledge, and needs to be set manually for each dataset.

To verify the effectiveness of dynamics, we construct a contrast model DMS-CNN without Dynamic (DMS-CNN

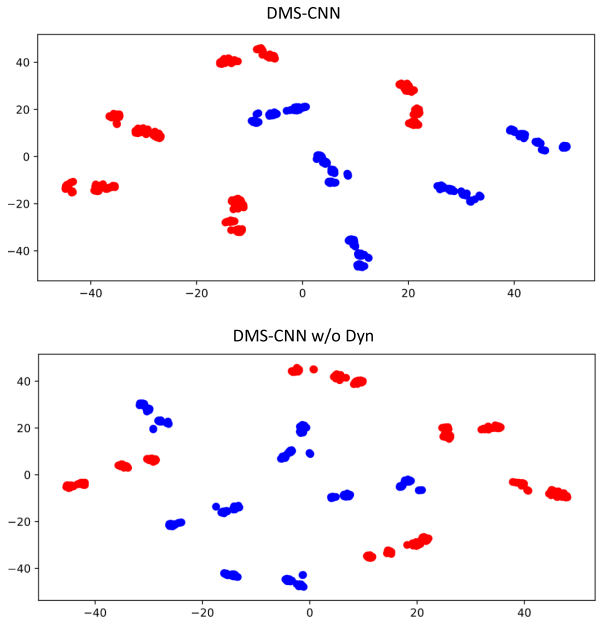


FIGURE 9. The 2-dimensions features of DMS-CNN and DMS-CNN w/o Dyn after dimension reduction. The feature maps of DMS-CNN have smaller intra-class distances and larger inter-class distances compared to the ones of DMS-CNN w/o Dyn. (We did not label the X-axis and Y-axis since they have no specific meanings.)

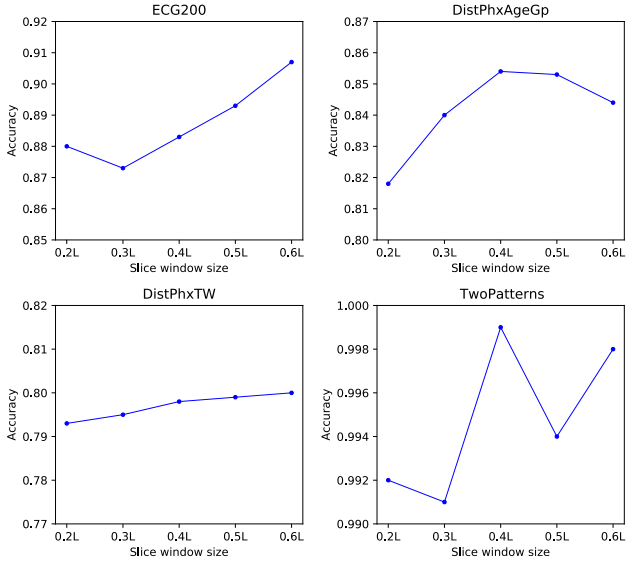


FIGURE 10. The classification accuracy of DMS-CNN on 4 UCR datasets when increasing the slice window size. For each subfigure, the X-axis and Y-axis indicate slice window size and corresponding accuracy, respectively. When l is set to 0.4L, DMS-CNN can basically obtain good results.

w/o Dyn). It only relies on the randomly initialized filters when generating the variable-length filters. That is, all samples use the same variable-length filters. The feature maps of a good classifier usually have a smaller intra-class distance and a larger inter-class distance. Therefore, we take the inputs of the output layers of the two models, respectively, and use t-SNE [33] to reduce their dimensions to 2-dimensions. The

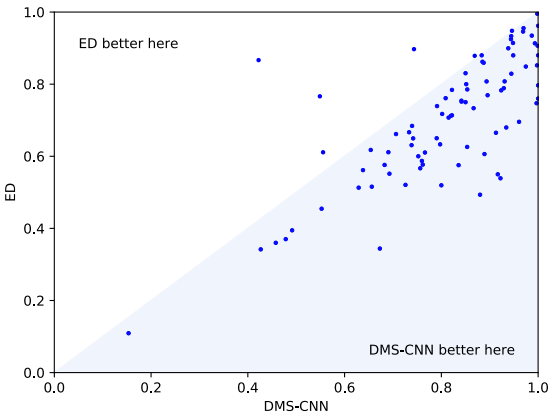


FIGURE 11. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and ED.

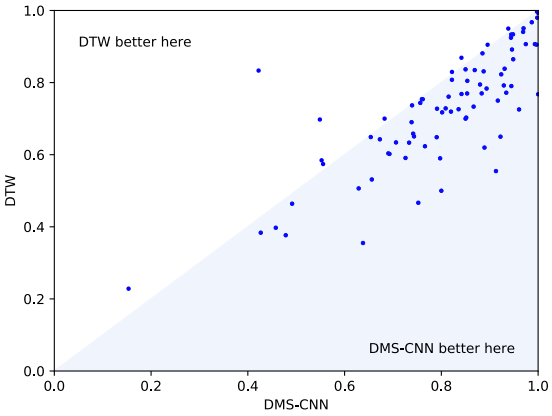


FIGURE 12. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and DTW.

features after dimensionality reduction are shown in Figure 9. We can find in Figure 9 that the feature maps of DMS-CNN have smaller intra-class distances and larger inter-class distances (e.g., the results of DMS-CNN have a fewer number of clusters than the ones of DMS-CNN w/o Dyn). Therefore, the dynamics of the filters are beneficial for the model to classify time series better.

F. EFFECT OF SLICE WINDOW SIZE

The slice window size l is also the length of fixed-length filters, and it is an essential hyperparameter. It determines the maximum length of the variable-length filters and thus influences the performance of DMS-CNN. To explore the effect of l on DMS-CNN, we conduct experiments with different l on four UCR datasets. These four datasets are ECG200, DistPhxAgeGp, DistPhxTW, and TwoPatterns, respectively. The hyperparameters of DMS-CNN, when applied to these four datasets, are described as follows. ω and λ of all the experiments are set to 3 and 10, respectively. The number of variable-length filters K is set to 120 when DMS-CNN is applied to ECG200, DistPhxAgeGp, and TwoPatterns. K is set to 90 when DMS-CNN is applied to DistPhxTW.

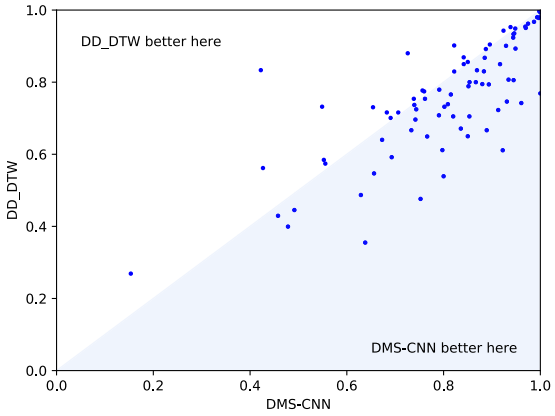


FIGURE 13. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and DD_DTW.

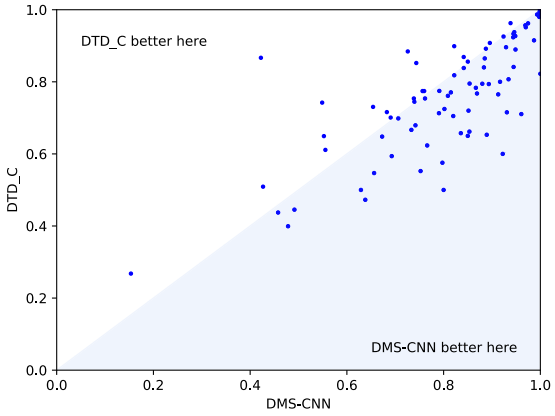


FIGURE 14. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and DTD_C.

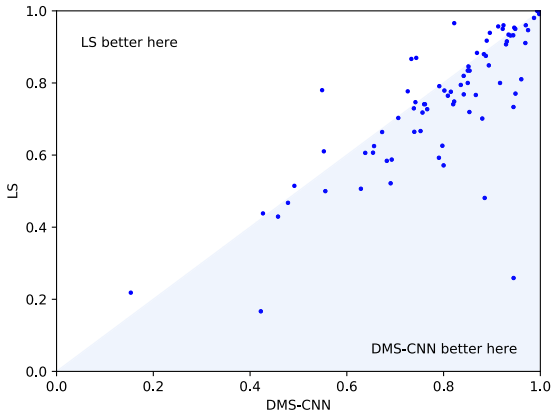


FIGURE 15. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and LS.

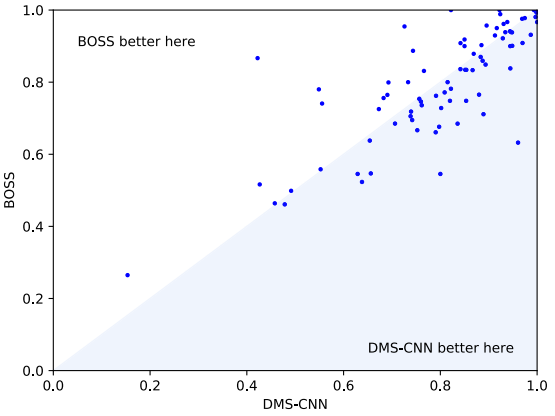


FIGURE 16. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and BOSS.

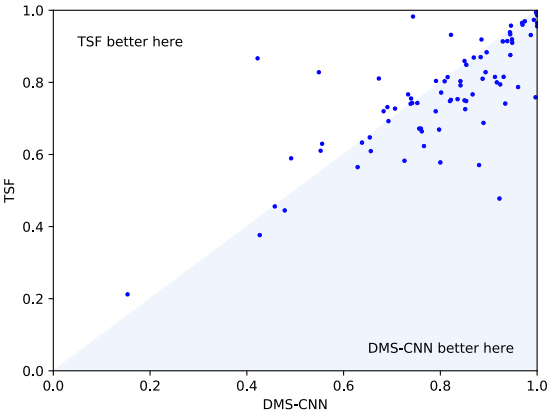


FIGURE 17. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and TSF.

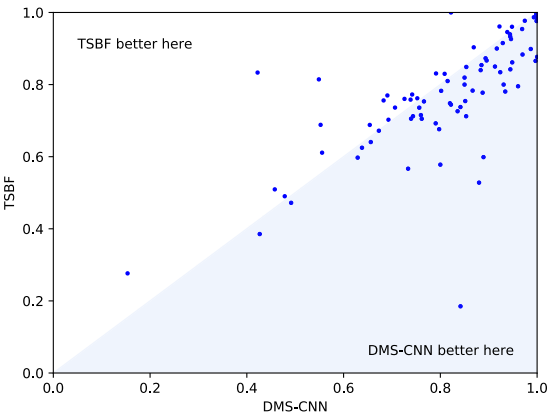


FIGURE 18. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and TSBF.

When increasing the slice window size l, the classification accuracies of DMS-CNN on 4 datasets are shown in Figure 10. For the ECG200 dataset, increasing the value of l can gradually improve the performance of DMS-CNN. For the DistPhxAgeGp dataset, increasing the value of l will increase the accuracy of DMS-CNN at first and then decrease. For the DistPhxTW dataset and the TwoPatterns dataset, the accuracies of DMS-CNN change little (note that the accuracies of the TwoPatterns are in a very small range of [0.99,

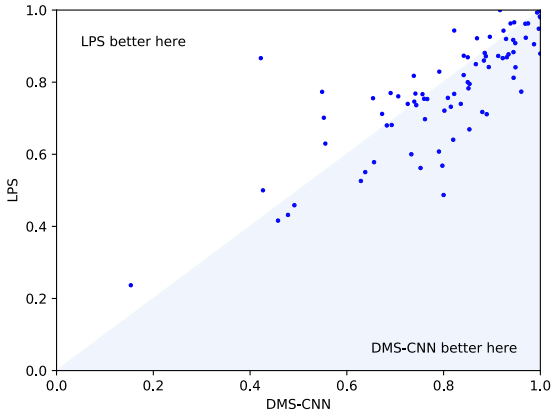


FIGURE 19. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and LPS.

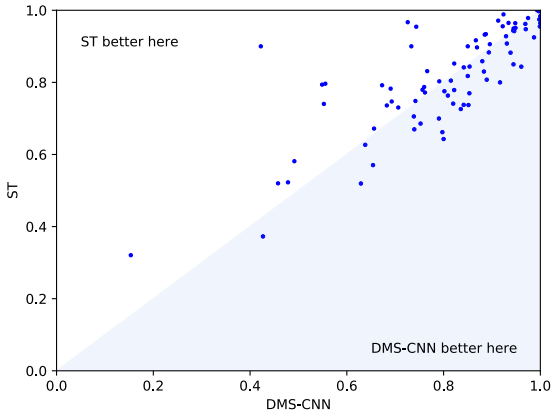


FIGURE 20. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and ST.

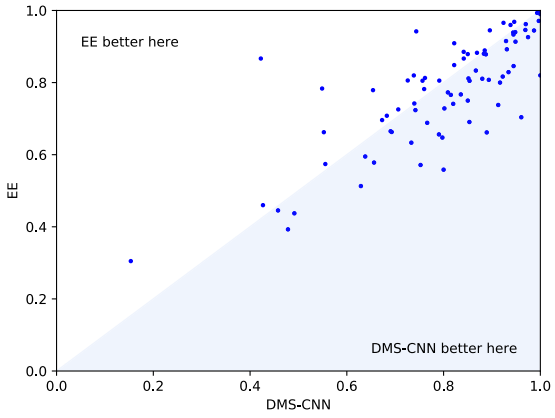


FIGURE 21. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and EE.

1.00]) with increasing the value of l. As shown in Figure 10, when l is set to $4L$, DMS-CNN can basically obtain good results. Therefore, l is set to $4L$ in all the UCR datasets to make a trade-off between model accuracy and efficiency.

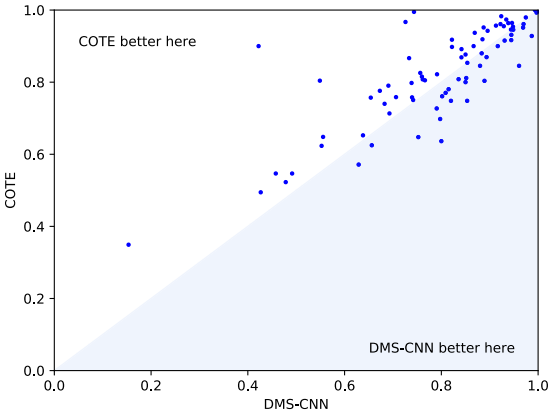


FIGURE 22. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and COTE.

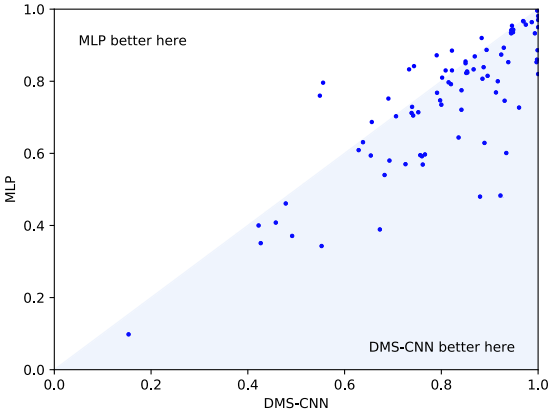


FIGURE 23. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and MLP.

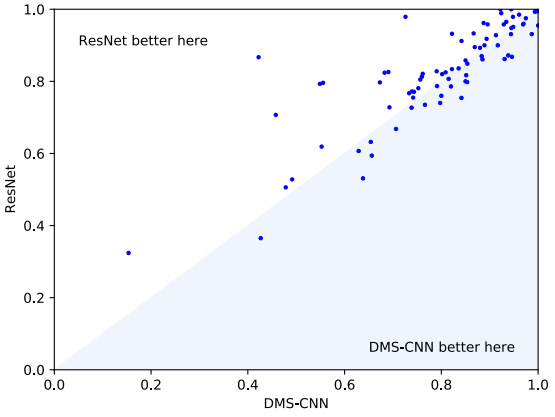


FIGURE 24. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and ResNet.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a convolution-based time series classification architecture called Dynamic Multi-scale CNNs (DMS-CNN) to solve the problem that CNNs cannot adaptively extract multi-scale temporal features for each time series. DMS-CNN is an end-to-end model that adaptively

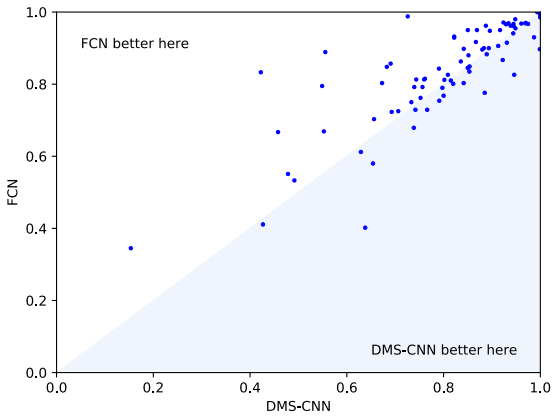


FIGURE 25. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and FCN.

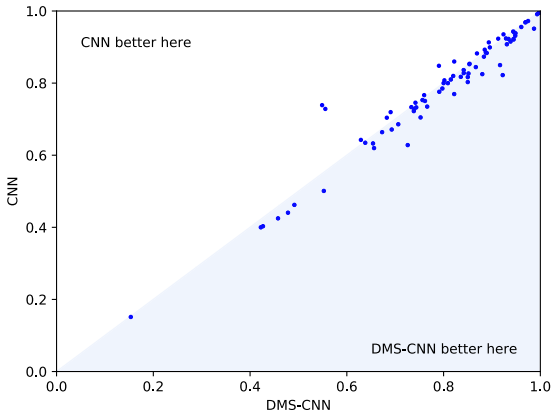


FIGURE 26. Pairwise accuracy plots on 85 UCR datasets between DMS-CNN and CNN.

learns multi-scale temporal features of each input time series through the dynamically learning of variable-length filters. Experimental results demonstrate that DMS-CNN outperforms all of the distance-based methods and feature-based methods and some of the ensemble-based methods, e.g., ST and EE. Although the results of DMS-CNN on the whole 85 UCR datasets can not beat COTE, ResNet, and FCN, it has no statistically significant difference with them. Moreover, DMS-CNN is significantly better than the baseline model, which verifies the effectiveness of the variable-length filters.

In the future, we will extend this approach to multivariate time series classification tasks, which is more challenging due to the correlation between multiple variables. In addition, we will explore how to learn variable-length filters in the deeper convolutional neural networks.

APPENDIX
PAIRWISE ACCURACY PLOTS BETWEEN DMS-CNN AND OTHER METHODS

To better understanding the results, we compare DMS-CNN with other methods in pairs. The pairwise-accuracy plots between DMS-CNN and other methods are shown in Figure 11-26.

REFERENCES

[1] K. Samiee, P. Kovacs, and M. Gabbouj, "Epileptic seizure classification of EEG time-series using rational discrete short-time Fourier transform," *IEEE Trans. Biomed. Eng.*, vol. 62, no. 2, pp. 541–552, Feb. 2015.

[2] O. Erkaymaz, M. Ozer, and M. Perc, "Performance of small-world feedforward neural networks for the diagnosis of diabetes," *Appl. Math. Comput.*, vol. 311, pp. 22–28, Oct. 2017.

[3] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognit. Lett.*, vol. 119, pp. 3–11, Mar. 2019.

[4] E. Trentin, S. Scherer, and F. Schwenker, "Emotion recognition from speech signals via a probabilistic echo-state network," *Pattern Recognit. Lett.*, vol. 66, pp. 4–12, Nov. 2015.

[5] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *Proc. KDD Workshop*, 1994, pp. 359–370.

[6] T. Górecki and M. Łuczak, "Using derivatives in time series classification," *Data Mining Knowl. Discovery*, vol. 26, no. 2, pp. 310–331, Mar. 2013.

[7] P. Schäfer, "The BOSS is concerned with time series classification in the presence of noise," *Data Mining Knowl. Discovery*, vol. 29, no. 6, pp. 1505–1530, Nov. 2015.

[8] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 392–401.

[9] M. G. Baydogan and G. Runger, "Time series representation and similarity based on local autopatterns," *Data Mining Knowl. Discovery*, vol. 30, no. 2, pp. 476–509, 2016.

[10] K. B. Venkataramana and C. C. Sekhar, "Large margin AR model for time series classification," in *Proc. 19th Int. Conf. Pattern Recognit.*, Dec. 2008, pp. 1–4.

[11] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Inf. Sci.*, vol. 239, pp. 142–153, Aug. 2013.

[12] M. G. Baydogan, G. Runger, and E. Tuv, "A bag-of-features framework to classify time series," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2796–2802, Nov. 2013.

[13] Y. Lei and Z. Wu, "Time series classification based on statistical features," *EURASIP J. Wireless Commun. Netw.*, vol. 2020, no. 1, pp. 1–13, Feb. 2020.

[14] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Mining Knowl. Discovery*, vol. 29, no. 3, pp. 565–592, May 2015.

[15] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, "Classification of time series by shapelet transformation," *Data Mining Knowl. Discovery*, vol. 28, no. 4, pp. 851–881, Jul. 2014.

[16] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with COTE: The collective of transformation-based ensembles," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 9, pp. 2522–2535, Sep. 2015.

[17] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 1578–1585.

[18] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," 2016, *arXiv:1603.06995*. [Online]. Available: http://arxiv.org/abs/1603.06995

[19] Q. Ma, W. Zhuang, L. Shen, and G. W. Cottrell, "Time series classification with echo memory networks," *Neural Netw.*, vol. 117, pp. 225–239, Sep. 2019.

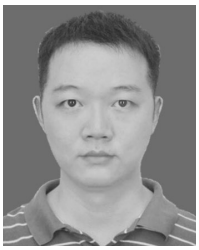
[20] X. Li, W. Wang, X. Hu, and J. Yang, "Selective kernel networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 510–519.

[21] S. Wang, M. Huang, and Z. Deng, "Densely connected CNN with multi-scale feature attention for text classification," in *Proc. 27th Int. Joint Conf. Artif. Intell.* San Mateo, CA, USA: Morgan Kaufmann, Jul. 2018, pp. 4468–4474.

[22] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12 pp. 2493–2537, Aug. 2011.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[24] Y. Jernite, E. Grave, A. Joulin, and T. Mikolov, "Variable computation in recurrent neural networks," 2016, *arXiv:1611.06188*. [Online]. Available: http://arxiv.org/abs/1611.06188
[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[26] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. (Jul. 2015). *The UCR Time Series Classification Archive*. [Online]. Available: https://www.cs.ucr.edu/~eamonn/time_series_data/
[27] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, *arXiv:1207.0580*. [Online]. Available: http://arxiv.org/abs/1207.0580
[28] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[29] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining Knowl. Discovery*, vol. 31, no. 3, pp. 606–660, May 2017.
[30] T. Górecki and M. Łuczak, "Non-isometric transforms in time series classification using DTW," *Knowl.-Based Syst.*, vol. 61, pp. 98–108, May 2014.
[31] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.
[32] D. Kaya, M. Türk, and T. Kaya, "Wavelet-based analysis method for heart rate detection of ecg signal using labview," in *Proc. 40th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, 2017, pp. 314–317.
[33] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.

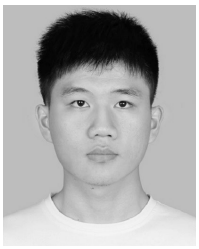


BIN QIAN received the master's degree in electrical engineering from the Huazhong University of Science and Technology.

He is currently the Head of the Measurement Equipment and Test Laboratory of Measurement Research Institute, Electric Power Research Institute, China Southern Power Grid, Guangzhou, China. His main research interests include electrical measurement and intelligent equipment detection technology research.

YONG XIAO received the Ph.D. degree in electrical engineering from Wuhan University.

He is currently the Deputy Director of the Measurement Research Institute, Electric Power Research Institute, China Southern Power Grid, Guangzhou, China. His main research interests include electrical measurement, research and development of new energy application technology, and related production and management. He has been employed as the Chief Science Communication Expert of the China Association of Science and Technology, and the Expert Member of the National Intelligent Measurement Technology Industry Innovation Alliance, and the National Power Substitution Industry Strategy Alliance, and the member of the National Power Demand Side Management Standard Committee, the China Electrical Engineering Society, and the National Electrical Instrument Standard Committee, and the Director of the Guangdong Measurement Society, and so on.



ZHENJING ZHENG is currently pursuing the master's degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China.

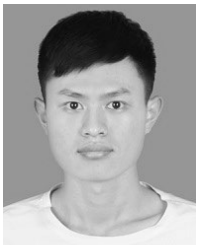
His current research interests include machine learning and deep learning.

MI ZHOU received the master's degree in information and communication engineering from the Beijing University of Posts and Telecommunications.

Her main research interests include electric energy data analysis and big data research.

WANQING ZHUANG is currently pursuing the master's degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China.

His current research interests include machine learning and deep learning.



SEN LI is currently pursuing the master's degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China.

His current research interests include machine learning and deep learning.

QIANLI MA (Member, IEEE) received the Ph.D. degree in computer science from the South China University of Technology, Guangzhou, China, in 2008.

From 2016 to 2017, he was a Visiting Scholar with the University of California at San Diego, La Jolla, CA, USA. He is currently a Professor with the School of Computer Science and Engineering, South China University of Technology. His current research interests include machine learning algorithms, data-mining methodologies, and time-series modeling and their applications.