Dynamic Sparse Network for Time Series Classification: Learning What to "See"

Qiao Xiao[1,*], **Boqian Wu**[2,*], **Yu Zhang** [3,4], **Shiwei Liu**[5,1], **Mykola Pechenizkiy**[1], **Elena Mocanu**[2], **Decebal Constantin Mocanu** [1,2]
[1] Eindhoven University of Technology, [2] University of Twente
[3] Southern University of Science and Technology
[4] Peng Cheng Laboratory, [5] University of Texas at Austin
{q.xiao,m.pechenizkiy}@tue.nl
{boqianwu333,yu.zhang.ust}@gmail.com
shiwei.liu@austin.utexas.edu
{e.mocanu,d.c.mocanu}@utwente.nl

Abstract

The receptive field (RF), which determines the region of time series to be "seen" and used, is critical to improve the performance for time series classification (TSC). However, the variation of signal scales across and within time series data, makes it challenging to decide proper RF sizes for TSC. In this paper, we propose a dynamic sparse network (DSN) with sparse connections for TSC, which can learn to cover various RF without cumbersome hyper-parameters tuning. The kernels in each sparse layer are sparse and can be explored under the constraint regions by dynamic sparse training, which makes it possible to reduce the resource cost. The experimental results show that the proposed DSN model can achieve the state-of-art performance on both univariate and multivariate TSC datasets with less than 50% computational cost compared with recent baseline methods, opening the path towards more accurate resource-aware methods for time series analyses. Our code is publicly available at: `https://github.com/QiaoXiao7282/DSN`.

1 Introduction

Time series classification (TSC) as an important research topic in the data mining communities [26], has a wide range of applications from health monitoring [27, 14], and public security [60], to grid energy [43], and remote sensing [45]. In the last decade, neural networks, especially deep neural networks, have achieved competitive or even better performance than traditional TSC approaches (e.g. DTW [48], BOSS [49], COTE [1]) in many cases [58, 26].

However, how to discover and exploit the various scaled signals hidden in time series (TS) is still a significant challenge for TSC tasks [54, 39, 21]. One main reason lies in several variances, like sampling rate and record length, which are natural during time-series collection [49, 34, 5]. Furthermore, the amplitude offset, warping and occlusion of data points are unavoidable (see Figure 1). So determining the optimal scales for feature extraction is difficult but significant in the TSC task. One main solution is to cover as many receptive field (RF) scales as possible, so it does not ignore any useful signals from the time series inputs [54, 7, 8, 28].

Grounding on the motivation that multiple kernels in combination can capture discriminative patterns despite complex warping, Fawaz et al. [13] tried to ensemble convolutional neural networks with a large number and variety of kernels. Tang et al. [55] have stacked several OS-CNN blocks, each of

[*]Equal contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

which consisted of a list of kernel sizes to cover all scales of RF and achieve better performance on several TSC tasks. In Multi-scale Convolutional Neural Network (MCNN) [4], a grid search was applied to find suitable kernel sizes. Without elaborate setting and fine-grid search, Dempster et al. [7] transformed time series by using random convolutional kernels and then trained a time series classifier. Even though the use of kernels with various sizes can help to extract latent hierarchical features of multiple resolutions, it tends to cause a huge computational problem and overfits the datasets when a limited amount of training data is available [28]. To address this problem, dilated convolution [62] has been used to reduce the computational cost while keeping the receptive field size. Franceschi et al. [15] presented an unsupervised learning model with convolutional kernels for time series feature transformation, in which the dilation factors of kernels increased exponentially layer by layer. To date, there is still an unresolved challenge with hyperparameter selections (e.g. kernel sizes and dilated factors) to capture patterns at different scales. This raises a foundational question: ***Can we design a scalable method for TSC that achieves a trade-off between the computation and performance without cumbersome hyperparameter tuning?***

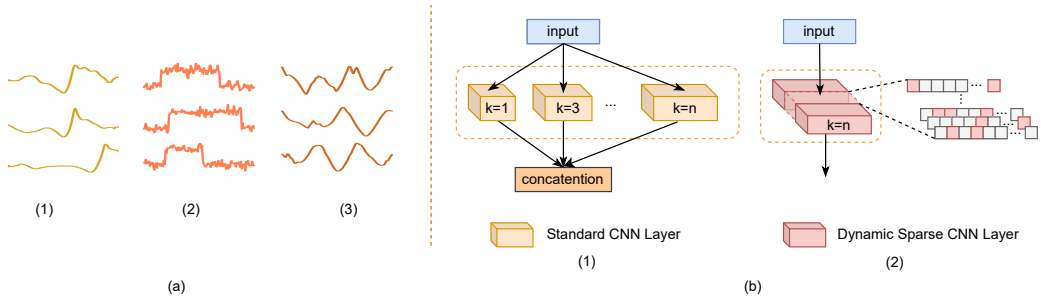

Figure 1: (a) Toy example illustrates the various time scales characteristic for TS data. Data in (1), (2) and (3) are sampled from three different TS datasets. The sequences with same color represent the same class. We can find that several sources of variances like phase, warping and offset are common in TS data. (b) The comparison between other methods [55, 28] (shown in b1) and our proposed DSN (shown in b2) for TSC. To cover various RF, DSN does not need to be configured with a series of kernel sizes k.

Inspired by sparse neural network models which can match the performance of dense counterparts with fewer connections [42, 37, 35, 63], we propose a dynamic sparse network (DSN) that is composed of CNN layers with large but dynamic sparse kernel for TSC tasks, which can automatically learn sparse connections in terms of covering various RF. The proposed DSN is trained by a dynamic sparse training (DST) strategy, which makes it possible to reduce the computational cost (e.g. floating-point operations (FLOPs)) for training and inference as well. Different from the traditional dynamic sparse training methods, which explore the connections in a layer-wise manner [42, 36, 12] and can hardly cover smaller RF, we propose a more fine-grained sparse training strategy for TSC. Specifically, the CNN kernels in each layer are split into several groups, each of which is sparse and can be explored under a constraint region during the course of sparse training. In this way, each layer can extract features in a more scalable and diverse way, and then by cascading, the model can cover more various scales of RF. To sum up, our main contributions are:

- We propose a novel dynamic sparse CNN layer with various effective neighbour RF, which can learn to "see" selectively and adaptively for more accurate TSC.
- By stacking the above dynamic sparse CNN layers, we introduce a novel DSN model natively for TSC, which is trained by DST to reduce computational and memory cost.
- To bridge gap between resource awareness and accuracy, we propose a more fine-grained training strategy for DSN, which can more easily cover diverse effective RF for TSC.
- We perform extensive empirical evaluation on both univariate and multivariate TSC tasks, showing that DSN achieves superior performance in terms of accuracy and resource cost.

2 Related Work

2.1 Time Series Classification

In the last decade, the success of deep learning encourages researchers to explore and extend its application on TSC tasks [31, 26, 58, 24, 32]. For univariate TSC, deep learning-based models attempt to directly transform raw time series data into low-dimensional feature representation via convolutional or recurrent networks [28, 47, 44]. For multivariate TSC, LSTM and attention layers are often stacked with CNN layers to extract complementary features from TS data [29, 65].

Recently, many works attempt to extract features in a more scalable manner as TS data is unavoidably composed of signals on various scales [49, 5, 54]. One main solution is to configure suitable kernels of various sizes such that they increase the probability to capture the proper scales [54, 55, 28]. Rocket-based methods [7, 8, 9] aim to use random kernels with several sizes and dilation factors to cover diverse RF for TSC. Different from these works, our proposed dynamic sparse CNN layer can adaptively learn the various RF with deformable dilation factors and trade-off both computation and performance without cumbersome hyperparameter tuning.

2.2 Sparse Training

Recently, the lottery ticket hypothesis has shown that it is possible to train a sparse sub-network to match the performance of dense counterpart with less computational cost [16]. Rather than iteratively pruning from a dense network [18, 66, 16], recent works try to find an initial mask with one-shot pruning based on gradient information during training [33, 57]. After pruning, the topology of the neural network will be fixed during training. However, this kind of models can hardly match the accuracy achieved by their dense counterparts [41, 57, 53].

Introduced as a new training paradigm before the lottery ticket hypothesis, DST starts from a sparse network and allows the sparse connectivity to be evolved dynamically with a fixed number of parameters throughout training [42, 12, 37, 63]. Nowadays, DST attracts increasing attentions from other research fields such as reinforcement learning [51] and continual learning [50], due to its potential of outperforming dense neural networks training [42, 63, 37]. Different from the traditional DST methods, our proposed DSN is trained with a fine-grained sparse training strategy rather than a traditional layer-wise manner to capture more diverse RF for TSC.

2.3 Adaptive Receptive Field

The RF, which can be adaptively changed during the training, has proven to be effective in many domains [56, 46, 59]. The adaptive RF can be usually captured by learning the optimal kernel sizes or the masks of kernels during the course of training [52, 19, 59]. However, neither the kernels nor the masks are sparse, which may cause the huge computational problem when larger RF needed. Different from these methods, our dynamic sparse CNN layers in DSN can be trained with DST, which can learn to capture variable RF with deformable dilation factors. What's more, the kernels are always sparse during training and inference, which makes it possible to reduce the computational cost.

3 The Proposed Model

3.1 Problem Definition

Definition 1. ***(Time Series Classification (TSC))****. Given a TS instance $\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\} \in \mathbb{R}^{n \times m}$, where m denotes the number of variates and n denotes the number of time steps, TSC aims to accurately predict the class label $y \in \{1, \ldots, c\}$ from c classes. When m equals 1, TSC is univariate and otherwise it is multivariate.*

Definition 2. ***(Time Series (TS) Training Set)****. A training set $\mathcal{D} = \{(\mathbf{X}^{(1)}, y^{(1)}), \ldots, (\mathbf{X}^{(N)}, y^{(N)})\}$ consists of N time series instances, where $\mathbf{X}^{(i)} \in \mathbb{R}^{n \times m}$ could either be a univariate or multivariate time series instance with its corresponding label $y^{(i)} \in \{1, \ldots, c\}$.*

Note that, in our case, all instances have the same number of time steps in a TS dataset. Without the loss of generality, given a training set, we aim to train a CNN classifier with adaptive and various RF with a low resource cost (e.g. memory and computation) for the TSC task.

3.2 Dynamic Sparse CNN Layer with Adaptive Receptive Field

A straightforward strategy to cover various RF is to apply multi-sized kernels in each CNN layer, but it has several limitations. Firstly, TS instances from different TS datasets do not have the same length and cycles [2] in most cases, making it difficult to set a fixed kernel configuration for all the datasets even with prior knowledge. Secondly, obtaining a large receptive field commonly requires a large kernel or stacking more layers, which introduces more parameters and thus increases storage and computational costs.

To tackle those challenges, the proposed dynamic sparse CNN layer possesses large but sparse kernels, which are learnable to capture adaptive RF. Specifically, given the input feature map $x^l \in \mathbb{R}^{c_{l-1} \times h \times w}$ in the lth layer (h equals 1 for univariate TSC and c_{l-1} is the number of input channels) , and kernel weights $\Theta^l \in \mathbb{R}^{c_{l-1} \times c_l \times 1 \times k}$ (k is the kernel size and c_l is the number of output channels), the convolution with stride 1 and padding in our proposed dynamic sparse CNN layer is formulated as

$$\mathbf{O}_j = \sum_{0 < i \leq c_{l-1}, i \in \mathbb{Z}} \left(\mathbf{I}^l(\Theta^l)_{i,j} \odot \Theta^l_{i,j} \right) \cdot x^l_i, \tag{1}$$

where $\mathbf{O}_j \in \mathbb{R}^{h \times w}$ denotes the output feature representation at the j-th output channel, $\mathbb{Z}$ denotes the set of integers, $\mathbf{I}^l(\cdot) : \mathbb{R}^{c_{l-1} \times c_l \times 1 \times k} \rightarrow \{0,1\}^{c_{l-1} \times c_l \times 1 \times k}$ is an indicator function, $\mathbf{I}^l(\Theta^l)_{i,j}$ indicates activated weights for $\Theta^l_{i,j} \in \mathbb{R}^{1 \times k}$ which is the kernel in the (i,j)-th channel, $\odot$ denotes the element-wise product, and $\cdot$ denotes the convolution operator. The indicator function $\mathbf{I}^l(\cdot)$, which is learned during the training of the proposed DSN, satisfies that $\|\mathbf{I}^l(\cdot)\|_0 \leq (1-S)N_l$ where $0 \leq S < 1$ is the sparsity ratio, $\|.\|_0$ denotes the L_0 norm and $N_l = c_{l-1} * c_l * 1 * k$. When $S > 0$, the kernel is sparse, and we can use a large k in the dynamic sparse CNN layer to obtain a large RF with reduced computational cost.

Remark 1. ***(Effective Neighbour Receptive Field (eNRF) size in the dynamic sparse CNN layer).*** *The receptive field is defined as the region in the input that the feature of a CNN model is looking at. We defined the Neighbour Receptive Field (NRF) as the region considered by each feature in the successive layer. Specifically, the NRF size is equivalent to the kernel size in the standard CNN layer (Considering the case that dilation equals to 1). Differently, the NRF size of the proposed dynamic sparse CNN layer is smaller than the kernel size, when the first or last weight in a kernel is not activated, e.g., $\exists i \in \{1, \ldots, c_{l-1}\}$ and $j \in \{1, \ldots, c_l\}$, such that $I^l(\Theta^l)_{i,j,1,1} = 0$ or $I^l(\Theta^l)_{i,j,1,k} = 0$. As illustrated in Figure 2, we defined the eNRF size $f^l_{i,j}$ of the kernel $\Theta^l_{i,j}$ as the distance between the first activated weight and last activated weight in the l-th CNN layer as*

$$f^l_{i,j} := \begin{cases} \max(Ind^l_{i,j}) - \min(Ind^l_{i,j}) + 1, & \text{if } Ind^l_{i,j} \neq \varnothing \\ 0, & \text{otherwise} \end{cases}, \tag{2}$$

where $Ind^l_{i,j}$ denotes the set of indices corresponding to non-zero weights in the kernel $\Theta^l_{i,j}$. Obviously we have $0 \leq f^l_{i,j} \leq k$.

The eNRF size set of l-th dynamic sparse CNN layer is denoted by $\mathbb{F}^{(l)}$, which satisfies that $0 \leq \min(\mathbb{F}^{(l)})$ and $\max(\mathbb{F}^{(l)}) \leq k$. Take the case in the Figure 2 as an simple example for l-th layer, then $\mathbb{F}^{(l)} = \{0,1,2,3,4,5,6\}$. Each dynamic sparse CNN layer is likely to cover various eNRF sizes ranging from 1 to k. When global information is expected, $\mathbf{I}^l(\cdot)$ could activate weights more dispersed to get a larger eNRF and exploit the input features selectively. To capture the local context, a smaller eNRF is expected, thus the activated weights tend to be

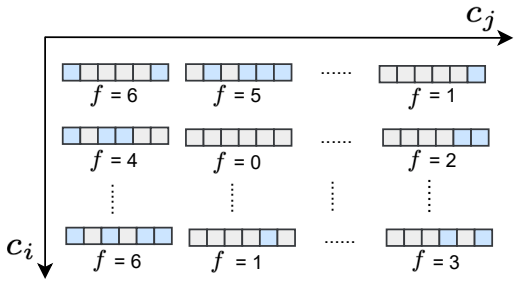

Figure 2: Illustrations for the eNRF size. The weights with blue color are activated.

[2] The term 'cycle' means that the data exhibit rising and falling that are not of a fixed frequency.

concentrated. By taking $k = 5$ as an example, $\mathbf{I}^l(\Theta^l)_{i,j}$ may be [1, 0, 1, 0, 1] for global context, while it may be [0, 0, 1, 1, 0] for local context. In this way, the eNRF can be adaptively adjusted.

3.3 Architecture in DSN

The proposed DSN model consists of three sparse CNN modules, each of which composes of a dynamic sparse CNN layer and a 1×1 CNN layer. Following the stacked sparse CNN modules, there is an additional dynamic sparse CNN layer, two adaptive average pooling layers, and one 1×1 convolution layer acting as the classifier in the DSN model. The overall architecture is shown in Figure 3.

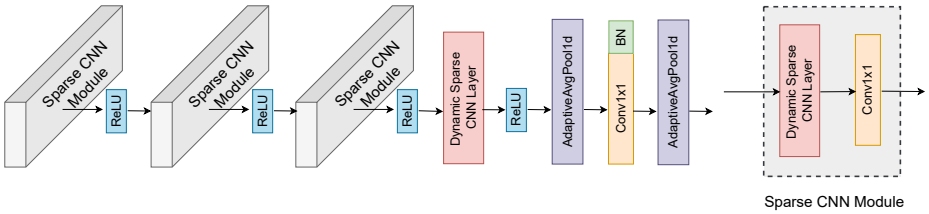

Figure 3: The proposed DSN model (left), which includes several sparse CNN modules (right) followed by a dynamic sparse CNN layer, two adaptive average pooling layers, and one 1×1 convolution layer.

The set of eNRF sizes $\mathbb{S}^{(l)}$ in the lth sparse CNN module is equal to that in the dynamic sparse CNN layer, since the eNRF size of 1×1 convolution is constantly equal to 1. $\mathbb{S}^{(l)}$satisfies that $0 \leq \min(\mathbb{S}^{(l)})$ and $\max(\mathbb{S}^{(l)}) \leq k$. Then, the set of eNRF sizes for the three successively stacked sparse CNN modules can be described by $\mathbb{RF}$ as

$$\mathbb{RF} = \left\{ \max(0, s^{(1)} + s^{(2)} + s^{(3)} - 2) \mid s^{(i)} \in \mathbb{S}^{(i)}, i \in \{1, 2, 3\} \right\}. \tag{3}$$

According to Eq. (3), we can see that stacking multiple sparse CNN modules can increase the size of $\mathbb{RF}$ linearly, where lth dynamic sparse CNN layer increases by the size of $\mathbb{S}^{(l)}$. For simplicity, the kernel size in each dynamic sparse CNN layer is consistently set to k in our study. Then, $\mathbb{RF}$ satisfies that $\max(\mathbb{RF}) \leq 3k - 2$ and $0 \leq \min(\mathbb{RF})$. Thus, a large k in sparse CNN modules can increase the range of the eNRF sizes for stacked sparse CNN modules.

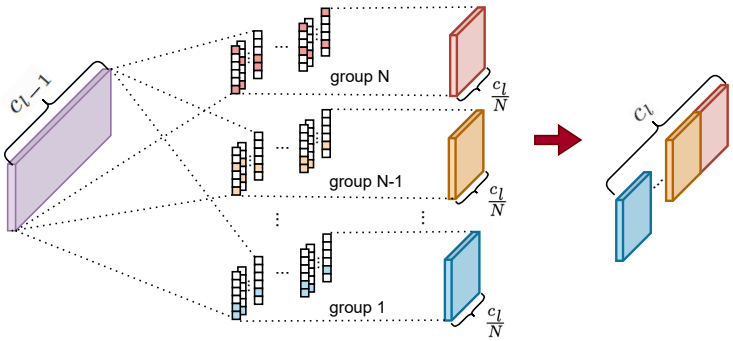

Figure 4: An illustration of the dynamic sparse CNN layer. In a dynamic sparse CNN layer, the kernels are split into N groups, each of which is with sparse connections learned by dynamic sparse training across specific constraint regions. The connections with color indicate non-zero values.

3.4 Dynamic Sparse Training for DSN Model

In this section, we present the training strategy to discovery the weights required to be activated in order to ensure well-performing RF. That is, we need to study how to update the indicator function $\mathbf{I}^l(\cdot)$ during the training of the proposed DSN.

We sparsely train the proposed DSN model from scratch to keep kernel sparse following the main idea of DST methods [42]. By design, during the training phase, the total number of activated weights must not exceed $N_l(1-S)$. However, we observe that the tiny eNRF is hard to be captured by DST methods with a layer-wise exploration manner, namely activated weights are discovered layer by layer, especially when the sparsity ratio S is undersized (more analysis in Section 4.5). Motivated by this observation, the kernels in each dynamic sparse CNN layer are divided into different groups, whose corresponding exploration regions are of different sizes as shown in Figure 4. Contrary to the DST methods, in the DSN the exploration of activated weights is separately performed in different kernel groups, which is a more fine-grained strategy. Specifically, the kernel weights $\Theta^l \in \mathbb{R}^{c_{l-1}\times c_l\times 1\times k}$ in the lth layer are split into N groups along the output channel, that is, $\Theta_1^l, \ldots, \Theta_N^l \in \mathbb{R}^{c_{l-1}\times \frac{c_l}{N}\times 1\times k}$ with corresponding exploration regions $\mathcal{R}_1^l, \ldots, \mathcal{R}_N^l$. The exploration region $\mathcal{R}_i^l$ in the ith group of the lth layer is defined as the first $\frac{i\times k}{N}$ positions of each kernel in this group. By taking $k=6$ and $N=3$ for example, activated weights in the first group are only within the first two positions of each kernel in Θ_1^l, while they are within the whole kernel for the last group (as shown in Figure 4). In this way, the various eNRF can be covered, and exploration space can be reduced (more details can be found in Appendix A.1) to improve exploration efficiency.

Algorithm 1 Dynamic Sparse Training for DSN Model

Input: Dataset $\mathcal{D}$, Network f_Θ, Sparsity ratio S
Exploration regions in layer l: $\{\mathcal{R}_1^l, ..., \mathcal{R}_N^l\}$
Exploration schedule: T, ΔT, α, f_{decay}
$\theta_i^l \leftarrow$ initialize activated weights in Θ_i^l using S and $\mathcal{R}_i^l$
for $t = 1$ **to** T **do**
  Sample a batch $B_t \sim \mathcal{D}$
  $L_t = \sum_{i\sim B_t} L\left(\left(f_\theta\left(x_i\right), y_i\right)\right.$
  if $(t \bmod \Delta T) == 0$ **then**
    for each layer $l = 1$ **to** L **do**
      for each group $i = 1$ **to** N **do**
        $u = n(\mathcal{R}_i^l) f_{\text{decay}}\left(t; \alpha, T\right)(1-S)$
        $\mathbb{I}_{\text{prune}} = \text{ArgTopK}\left(-\left|\theta_i^l\right|, u\right)$
        $\mathbb{I}_{\text{grow}} = \text{RandomK}\left(\mathcal{R}_i^l \backslash \theta_i^l, u\right)$
        $\mathbf{I}^l(.) \leftarrow$ Update $\mathbf{I}^l(.)$ using $\mathbb{I}_{\text{prune}}$ and $\mathbb{I}_{\text{grow}}$
        $\theta_i^l \leftarrow$ Update activated weights $\mathbf{I}^l(\Theta^l)_i \odot \Theta_i^l$
      end for
    end for
  else
    $\theta = \theta - \alpha \nabla_\theta L_t$
  end if
end for

Given the weight exploration regions $\mathcal{R}_1^l, \ldots, \mathcal{R}_N^l$ and the sparsity ratio S, we train the proposed DSN model as shown in Algorithm 1. The activated weights are explored within the exploration regions and updated every Δ_t iterations. The fraction of updated weight decays over time according the function $f_{\text{decay}}\left(t; \alpha, T\right)$, which follows the cosine annealing [10] as

$$f_{decay}\left(t; \alpha, T\right) = \frac{\alpha}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right), \tag{4}$$

where α is the initial fraction of activated weights updated, t is the current training iteration, and T is the number of training iterations. Thus, during the tth iteration, the number of updated activated weights in the ith group of the lth layer is $n(\mathcal{R}_i^l) f_{\text{decay}}\left(t; \alpha, T\right)(1-S)$, where $n(\mathcal{R}_i^l)$ is the number of weights can be explored in region R_i^l. During the update of activated weights, we firstly prune the activated weights determined by $\text{ArgTopK}\left(-\left|\theta_i^l\right|, u\right)$ and then randomly grow new activated weights by $\text{RandomK}\left(\mathcal{R}_i^l \backslash \theta_i^l, u\right)$, where $\text{ArgTopK}(\mathbf{v}, u)$ gives the indices of top u elements with larger values in a vector $\mathbf{v}$, $\text{RandomK}(\mathbf{v}, u)$ outputs the indices of random u elements in a vector $\mathbf{v}$, and $\mathcal{R}_i^l \backslash \theta_i^l$ denotes the weights within $\mathcal{R}_i^l$ but except θ_i^l.

The exploration for activated weights is straightforward. Firstly, pruning weights with small magnitudes is intuitive, because the contribution of weights with smaller magnitudes is insignificant or even negligible. Considering the pruning recoverability, we randomly regrow new weights with the same number as the pruned weights to achieve better activated weight exploration. In this way, the exploration of weights is dynamic and of plasticity, compared with the methods of pruning weights before and after training (more explanations can be found in Appendix A.3).

4 Experiments

In this section, we evaluate the resource cost and accuracy of the proposed DSN model with recent baseline methods on both univariate and multivariate TSC data.

4.1 Datasets and Baselines

The details of each kind of dataset are as follows.

- **Univariate TS dataset from UCR 85 archive [3]**: This archive consists of 85 univariate TS datasets, which are collected from various domains (e.g. health monitoring and remote sensing), have distinguishable characteristics, and various levels of complexity. The number of instances in the training set varies from 16 to 8926, while their time step resolutions range from 24 to 2709.
- **Three multivariate TS datasets from UCI [11]**: The EEG2 dataset contains 1200 instances of 2 categories and 64 variates. The Human Activity Recognition (HAR) dataset consists of 10,299 instances with 6 categories, and the number of variates is 9. The Daily Sport dataset contains 9,120 instances of 19 categories, and its number of variates is 45. The processing of these dataset is the same as in [29].

Univariate TSC baselines: The performance of the following three baseline methods are reported, OS-CNN [54], InceptionTime [28], and ResNet[58], as they are widely used for univariate TSC. **Multivariate TSC baselines:** Similar with [65], the following three multivariate baselines are selected: OS-CNN, TapNet [65], MLSTM-FCN [29]. To avoid an unfair comparison, we let outside the ensemble methods. However, based on our best knowledge, our performance analysis accounts for most of state-of-the-art methods.

4.2 Experimental Settings and Implementation Details

For optimization, we use the Adam optimizer with an initial learning rate of 3×10^{-4}, cosine decayed to 10^{-4}. Our model is trained for 1,000 epochs with a mini-batch size of 16. As in [28, 29], the best model, which corresponds to the minimum training loss, is used to evaluate performance over the testing sets. Inspired by [28], the kernel sizes k in each dynamic sparse CNN layer are set to 39. The number of kernel groups N in Algorithm 1 is set to 3, which helps to cover the small, medium and large eNRF. Each setting is repeated for five times and the average results are reported.

4.3 Results and Analysis

We show the performance on both univariate and multivariate TSC benchmarks in Table 1-2. For univariate TSC, we can see that our method outperforms the baseline methods in most cases with a smaller number of parameters (e.g. Params [3]) and less computation cost (e.g. FLOPs) on UCR 85 archive datasets. For multivariate TSC, our proposed DSN method achieves better performance on EEG2 and HAR datasets, while for Daily Sport, MLSTM-FCN was with 0.46% more accurate. It is worth highlighting that the resource cost (i.e. Params and FLOPs) of our proposed DSN is much smaller than that of baseline methods. Due to the page limitation, the detailed results including test accuracy and resource cost on each dataset of UCR 85 archive can be found in Appendix C.

Table 1: Pairwise comparison of test accuracy (%) and mean resource cost (i.e. Params (K)↓ and FLOPs (M)↓) of our method and other univariate TS baseline methods on UCR 85 archive datasets. Note that all the accuracies in comparison are rounded to two decimal places.

Archive	Methods	Baseline wins	**DSN** wins	Tie	Params	FLOPs
UCR 85 archive	ResNet	29	**53**	3	479.20	803.38
	InceptionTime	**40**	**40**	5	389.34	325.49
	OS-CNN	33	**49**	3	262.14	200.12
	DSN (ours)	-	-	-	**126.22**	**104.88**

4.4 Sensitivity Analysis

Effect of Sparsity Ratio: From Table 3, we can find out how the sparsity ratio of dynamic sparse CNN layer affects the final test accuracy and the resource cost in multivariate TS datasets. Here, we analyze the trade-off between test accuracy and the resource cost under various sparsity ratios

[3] The amount of non-zero parameters in the model as in [51, 30, 33].

Table 2: Test accuracy (ACC(%)) and resource cost (i.e. Params $(K)\downarrow$ and FLOPs $(M)\downarrow$) of different models on three multivariate TS datasets from UCI.

	Daily Sport			EEG2			HAR		
Methods	ACC	Params	FLOPs	ACC	Params	FLOPs	ACC	Params	FLOPs
TapNet	99.56	1885.58	346.97	85.30	1369.21	780.19	93.33	1275.78	310.61
MLSTM-FCN	**99.65**	324.77	70.59	91.00	342.51	161.53	96.71	284.97	63.38
OS-CNN	99.30	317.05	78.01	93.03	320.34	162.04	96.37	280.39	70.98
DSN (ours)	99.19	**163.68**	**40.48**	**99.10**	**162.14**	**82.31**	**96.82**	**160.13**	**40.64**

(i.e. $S \in [50\%, 80\%, 90\%]$) and a dense DSN model. Note that the dense model is defined exactly as our DSN model with dense connections along entire kernels. We can see that it is not that the more parameters of the model, the better performance it will achieve. This is mainly because more parameters will lead to overfitting and the proper sparse ratio will easily cover the desired receptive field (more demonstrations can be seen in Appendix B.1). Taking EEG2 data set as an example, with the decrease of sparse rate, the performance on it drops sharply. We hypothesise that this data set has more expectations for features from small receptive fields (the validation can be found in Appendix B.2). According to the experiments, a 80% sparsity ratio reflects a good trade-off between accuracy and resource cost, which is the default setting for the dynamic sparse CNN layer in our DSN model.

Table 3: Test accuracy (ACC(%)) and resource cost (i.e. Params $(K)\downarrow$ and FLOPs $(M)\downarrow$) of the proposed DSN model with different sparsity ratios on three multivariate TS datasets from UCI.

	Daily Sport			EEG2			HAR		
Methods	ACC	Params	FLOPs	ACC	Params	FLOPs	ACC	Params	FLOPs
Dense	99.19	1059.65	262.10	89.60	1058.11	536.17	96.06	1056.10	267.57
$S = 50\%$	**99.28**	370.44	91.63	96.23	368.90	187.05	96.62	366.89	93.01
$S = 80\%$	99.19	163.68	40.48	**99.10**	162.14	82.31	**96.82**	160.13	40.64
$S = 90\%$	99.25	**94.76**	**23.44**	98.90	**93.22**	**46.64**	96.75	**91.21**	**23.18**

Effect of Architecture: To show how the architecture of DSN affects the final performance, we adopt the critical difference diagram for a detailed comparison, which is usually used for evalution in TSC [54, 28]. From Figure 5, we can find that the DSN model with 4 dynamic sparse CNN layers and 177 output channels in each layer performs consistently better than the other structures. The proposed model with more output channels seems to be crucial for preserving the capacity of the network, while requires more parameters and FLOPs compared to other counterparts. This highlights an interesting trade-off between accuracy and computational efficiency. According to the experiments, the DSN model with 4 dynamic sparse CNN layers and 141 output channels in each layer is the trade-off between the performance and resource cost, which is the default setting for our DSN model.

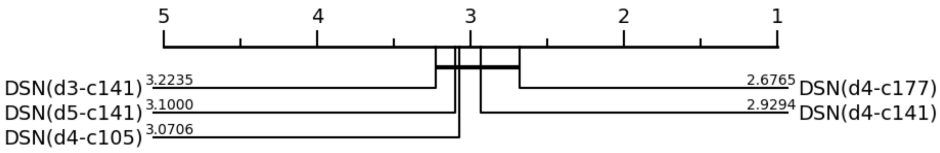

Figure 5: The critical difference diagram shows the average rank of the proposed DSN model with different architectures in UCR 85 archive. DSN(di-cj) represents the DSN model with i dynamic sparse CNN layers and j output channels in each layer. The smaller average rank corresponds to better performance of the model.

Effect of Kernel Group: As shown in Figure 6, training DSN without kernel group hampers the capture of small eNRF, which can not guarantee satisfactory performance on the datasets such as EEG2 expecting more local information. With the decrease in sparsity ratio S, the phenomenon that large RF occupies the majority becomes more severe (more demonstrations see in Appendix B.1),

which increases the performance gap between training with group and without it as shown in Table 4. Grouping the kernels can achieve various RF coverage and adorable accuracy for all datasets.

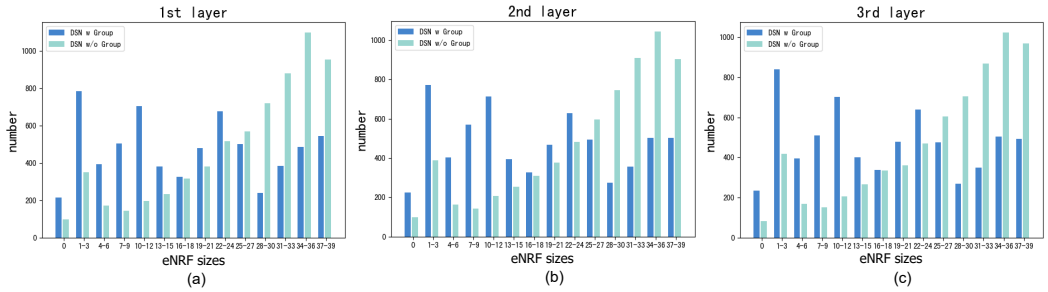

Figure 6: The distribution of eNRF sizes in different sparse CNN layers between DSN models training with and without grouping on HAR dataset. The number on y-axis corresponds to the frequency of eNRF sizes.

4.5 Ablation Study

Effect of Dynamic Sparse Training: Furthermore, we study the effect of the dynamic sparse training in our DSN method. We conduct ablation studies to compare DSN with its static variant (i.e. DSN^{fix}), which fixes its topology during training after activated weights initialization. In other words, the indicator function $\mathbf{I}^l(\cdot)$ in Eq. (1) will not update during the training process. According to the results shown in Figure 7 (a), we can find that the DSN model mostly outperforms the DSN^{fix} on UCR 85 archive, indicating that what are learned by dynamic sparse training is the suitable activated weights together with their values.

Table 4: Average accuracies (%) of the proposed DSN model affected by kernel grouping.

Sparsity ratio	Methods	Daily Sport	EEG2	HAR
$S = 60\%$	DSN w Group	**99.27**	**97.87**	96.53
	DSN w/o Group	99.23	96.77	**96.87**
$S = 70\%$	DSN w Group	99.20	**98.70**	**97.14**
	DSN w/o Group	**99.25**	97.27	96.52
$S = 80\%$	DSN w Group	99.19	**99.10**	**96.82**
	DSN w/o Group	**99.28**	98.77	96.79

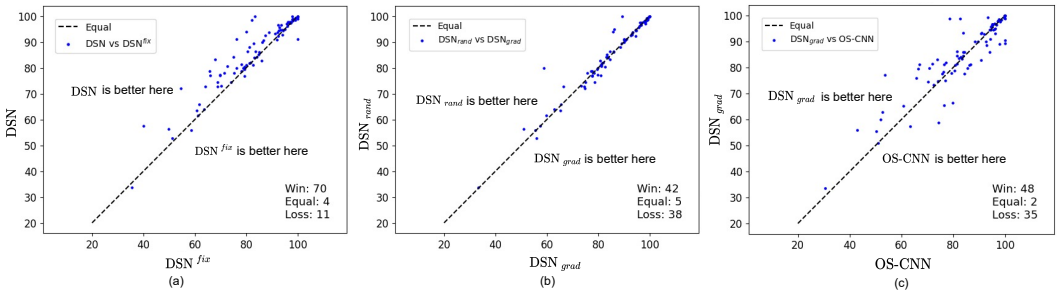

Figure 7: Accuracy plotting showing how the proposed DSN model is affected by the dynamic sparse training in UCR 85 archive.

4.6 Case Study on other DST Method

Likewise, the effectiveness of the proposed DSN method is also analyzed in the presence of other DST methods, such as RigL [12]. Rather than growing weights randomly (as in the line 9 of Algorithm 1), RigL grows them with the highest magnitude gradients. From Figure 7 (c), we can see that growing weights according to gradients (i.e. DSN$_{grad}$) can outperforms the OS-CNN method in most cases on

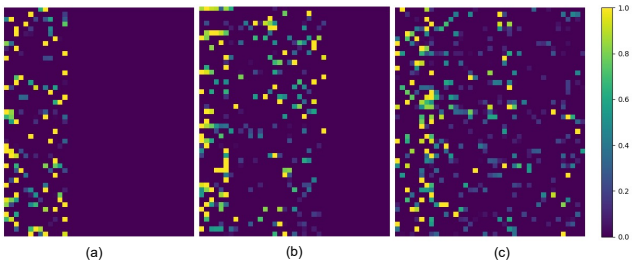

Figure 8: The activated weights in different kernel groups.

UCR 85 archive. Furthermore, as shown in Figure 7 (b), we can find that DSN_{grad} has a similar performance as growing weights randomly (i.e. DSN_{rand}). Note that, DSN_{rand} is the main setting of the proposed method in this paper with the reason that we don't need to compute the full gradients at any moment in time. In summary, the proposed DSN with fine-grained sparse strategy can be easily combined with the existing DST techniques, showing the potential for further improvements by merging it with other advanced DST methods in the future.

4.7 Visualization of Activated Weights

Figure 8 shows the normalized activated weights of the DSN model after sparse training on HAR dataset. (a), (b) and (c) correspond to the activated weights of the three kernel groups in the first dynamic sparse CNN layer, and each row of the figure represents one kernel. We can see that the weights in each kernel are sparse, and can be activated within the constraint region defined in Section 3.4.

5 Conclusions

This paper proposes dynamic sparse network (DSN) to cover diverse effective neighbour receptive field (eNRF) for time series classification (TSC) without the cumbersome hyper-parameters tuning. The proposed DSN can achieve about $2\times$ reduction in memory and computational cost, while achieving superior performance in terms of accuracy on both univariate and multivariate TSC datasets. Furthermore, DSN explores the activated weights in a more fine-grained strategy, which can be easily merged with existing DST techniques, showing a potential for further improvements when combined with advanced technology. DSN provides a feasible solution to bridge gap between resource awareness and various eNRF coverage for TSC, hoping to inspire other researches in different fields. It is worthwhile extending DSN to other research domains (e.g. time series forecasting and computer vision). Our DSN model has not yet been executed on real edge devices, and the performance in terms of accuracy and memory footprint on edge devices is not investigated. In the future, we hope to extend our work to other fields and decentralise training on edge devices.

Acknowledgements

This work has been partly funded by the NWO Perpectief project MEGAMIND.

References

[1] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015.

[2] Anthony J. Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn J. Keogh. The UEA multivariate time series classification archive, 2018. *CoRR*, abs/1811.00075, 2018.

[3] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[4] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-scale convolutional neural networks for time series classification. *arXiv preprint arXiv:1603.06995*, 2016.

[5] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

[6] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. The ucr time series classification archive, October 2018. `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`.

[7] Angus Dempster, François Petitjean, and Geoffrey I Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

[8] Angus Dempster, François Petitjean, and Geoffrey I. Webb. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Min. Knowl. Discov.*, 34(5):1454–1495, 2020.

[9] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 248–257, 2021.

[10] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *CoRR*, abs/1907.04840, 2019.

[11] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[12] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 2943–2952. PMLR, 2020.

[13] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep neural network ensembles for time series classification. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2019.

[14] Germain Forestier, François Petitjean, Pavel Senin, Fabien Despinoy, Arnaud Huaulmé, Hassan Ismail Fawaz, Jonathan Weber, Lhassane Idoumghar, Pierre-Alain Muller, and Pierre Jannin. Surgical motion analysis using discriminative interpretable patterns. *Artificial intelligence in medicine*, 91:3–11, 2018.

[15] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems*, 32, 2019.

[16] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

[17] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[18] Sharath Girish, Shishira R. Maiya, Kamal Gupta, Hao Chen, Larry S. Davis, and Abhinav Shrivastava. The lottery ticket hypothesis for object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 762–771, 2021.

[19] Shizhong Han, Zibo Meng, Zhiyuan Li, James O'Reilly, Jie Cai, Xiaofeng Wang, and Yan Tong. Optimizing filter size in convolutional neural networks for facial action unit recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*, pages 5070–5078, 2018.

[20] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[21] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony J. Bagnall. Classification of time series by shapelet transformation. *Data Min. Knowl. Discov.*, 28(4):851–881, 2014.

[22] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network.

[23] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[24] Seth H Huang, Xu Lingjie, and Jiang Congwei. Residual attention net for superior cross-domain time sequence modeling. *arXiv preprint arXiv:2001.04077*, 2020.

[25] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[26] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

[27] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, François Petitjean, Lhassane Idoumghar, and Pierre-Alain Muller. Automatic alignment of surgical videos using kinematic data. In *Conference on Artificial Intelligence in Medicine in Europe*, pages 104–113. Springer, 2019.

[28] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020.

[29] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 116:237–245, 2019.

[30] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham M. Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119, pages 5544–5555, 2020.

[31] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[32] Dongha Lee, Seonghyeon Lee, and Hwanjo Yu. Learnable dynamic temporal pooling for time series classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8288–8296, 2021.

[33] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[34] Jason Lines, Luke M Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–297, 2012.

[35] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. *Advances in Neural Information Processing Systems*, 34, 2021.

[36] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse RNN training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML*, volume 139, pages 6893–6904. PMLR, 2021.

[37] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *International Conference on Machine Learning*, pages 6989–7000. PMLR, 2021.

[38] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[39] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Abbas Zaidi, Bart Goethals, François Petitjean, and Geoffrey I. Webb. Proximity forest: an effective and scalable distance-based classifier for time series. *Data Min. Knowl. Discov.*, 33(3):607–635, 2019.

[40] Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110(11):3211–3243, 2021.

[41] Decebal Constantin Mocanu, Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. A topological insight into restricted boltzmann machines. *Machine Learning*, 104(2):243–270, 2016.

[42] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[43] Elena Mocanu, Decebal Constantin Mocanu, Phuong H Nguyen, Antonio Liotta, Michael E Webber, Madeleine Gibescu, and Johannes G Slootweg. On-line building energy optimization using deep reinforcement learning. *IEEE transactions on smart grid*, 10(4):3698–3708, 2018.

[44] Kewei Ouyang, Yi Hou, Shilin Zhou, and Ye Zhang. Convolutional neural network with an elastic matching mechanism for time series classification. *Algorithms*, 14(7):192, 2021.

[45] Charlotte Pelletier, Geoffrey I Webb, and François Petitjean. Temporal convolutional neural network for the classification of satellite image time series. *Remote Sensing*, 11(5):523, 2019.

[46] Silvia L. Pintea, Nergis Tomen, Stanley F. Goes, Marco Loog, and Jan C. van Gemert. Resolution learning in deep convolutional networks using scale-space theory. *IEEE Trans. Image Process.*, 30:8342–8353, 2021.

[47] Elahe Rahimian, Soheil Zabihi, Seyed Farokh Atashzar, Amir Asif, and Arash Mohammadi. Xceptiontime: Independent time-window xceptiontime architecture for hand gesture classification. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1304–1308, 2020.

[48] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270, 2012.

[49] Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015.

[50] Jonathan Schwarz, Siddhant M. Jayakumar, Razvan Pascanu, Peter E. Latham, and Yee Teh. Powerpropagation: A sparsity inducing weight reparameterisation. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, pages 28889–28903, 2021.

[51] Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peterk Stone. Dynamic sparse training for deep reinforcement learning. In *31st International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence*, 2022.

[52] Domen Tabernik, Matej Kristan, and Ales Leonardis. Spatially-adaptive filter units for compact and efficient deep neural networks. *Int. J. Comput. Vis.*, 128(8):2049–2067, 2020.

[53] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.

[54] Wensi Tang, Guodong Long, Lu Liu, Tianyi Zhou, Michael Blumenstein, and Jing Jiang. Omni-scale cnns: a simple and effective kernel size configuration for time series classification. In *International Conference on Learning Representations*, 2022.

[55] Wensi Tang, Guodong Long, Lu Liu, Tianyi Zhou, Jing Jiang, and Michael Blumenstein. Rethinking 1d-cnn for time series classification: A stronger baseline. *arXiv preprint arXiv:2002.10061*, 2020.

[56] Nergis Tomen, Silvia-Laura Pintea, and Jan van Gemert. Deep continuous networks. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139, pages 10324–10335, 2021.

[57] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019.

[58] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[59] Zhitong Xiong, Yuan Yuan, Nianhui Guo, and Qi Wang. Variational context-deformable convnets for indoor scene parsing. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020*, pages 3991–4001. Computer Vision Foundation / IEEE, 2020.

[60] Fei Yi, Zhiwen Yu, Fuzhen Zhuang, Xiao Zhang, and Hui Xiong. An integrated model for crime prediction using temporal and spatial factors. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1386–1391. IEEE, 2018.

[61] Sean Young, Zhe Wang, David Taubman, and Bernd Girod. Transform quantization for cnn compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[62] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, May 2016.

[63] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. *Advances in Neural Information Processing Systems*, 34, 2021.

[64] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2114–2124, 2021.

[65] Xuchao Zhang, Yifeng Gao, Jessica Lin, and Chang-Tien Lu. Tapnet: Multivariate time series classification with attentional prototypical network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6845–6852, 2020.

[66] Zhenyu Zhang, Xuxi Chen, Tianlong Chen, and Zhangyang Wang. Efficient lottery ticket finding: Less data is more. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 12380–12390. PMLR, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.2
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Preliminaries

A.1 Preliminaries on Comparison of Exploration Space

The size of exploration space for the layer-wise manner in SET algorithm is:

$$\left(\begin{array}{c} N_l \\ N_l(1-S) \end{array} \right) = \frac{N_l!}{N_l(1-S)!N_lS!}. \tag{5}$$

If we divide the kernels in one layer into N groups and explore $\frac{N_l(1-S)}{N}$ activated weights in each group, then the size of the exploration space becomes:

$$\left(\begin{array}{c} \frac{N_l}{N} \\ \frac{N_l(1-S)}{N} \end{array} \right)^N = \left(\frac{\frac{N_l}{N}!}{\frac{N_l(1-S)}{N}!\frac{N_lS}{N}!}\right)^N \leq \left(\begin{array}{c} N_l \\ N_l(1-S) \end{array} \right). \tag{6}$$

When the corresponding exploration region of each group is defined as in Section 3.4, the exploration space can be further reduced to:

$$\left(\begin{array}{c} \frac{N_l}{N^2} \\ \frac{N_l(1-S)}{N} \end{array} \right) \left(\begin{array}{c} \frac{2N_l}{N^2} \\ \frac{N_l(1-S)}{N} \end{array} \right) \cdots \left(\begin{array}{c} \frac{N_l}{N} \\ \frac{N_l(1-S)}{N} \end{array} \right) < \left(\begin{array}{c} \frac{N_l}{N} \\ \frac{N_l(1-S)}{N} \end{array} \right)^N. \tag{7}$$

A.2 Preliminaries on Calculation of Parameters

The summation of the number of non-zero weights (activated weights in sparse CNN layer + weights in other layers) in the network is used to estimate the size of network as follows:

$$Params = \sum_{l \notin \mathbb{L}} \|\theta^l\|_0 + \sum_{l \in \mathbb{L}} \sum_{i=1}^{N} \|\mathbf{I}^l(\Theta^l)_i \odot \Theta_i^l\|_0 = \|\theta\|_0, \tag{8}$$

where $\mathbb{L}$ contains a set of sparse CNN layer indexes, and θ is the parameter of our DSN which includes an amounts of zero values. $\|\mathbf{I}^l(\Theta^l)_i \odot \Theta_i^l\|_0$ is controlled by the sparsity ratio S and the corresponding exploration regions $\mathbf{R}_i^l$.

A.3 Preliminaries on Existing Compression Solutions

The existing compression solutions can be categorized into after-training, during-training and before-training methods.

- **After-training methods** obtain a sparse or small neural network from a pre-trained network by pruning the unimportant weights [20], quantifying the weights [61] or distillation [22]. Normally, this schedule type uses a standard dense training procedure followed by a re-trained procedure after compression to reach higher accuracy.
- **During-training methods**, which discover the sparse network during training without pre-trained network, can be divided into dense-to-sparse and sparse-to-sparse training strategy. Dense-to-sparse training strategy sparsifies the neural network during the training process through gradual magnitude pruning [17] or regularization techniques [38]. Sparse-to-sparse training strategy initializes a random (topology and weights) sparse network, and dynamically optimizes the topology as well as the weights simultaneously during training [42, 12]. Our DSN model is based on the sparse-to-sparse training strategy.
- **Before-training methods** aim to obtain a trainable small or sparse neural network before the main training process. Squeezenet [25] and MobileNets [23] are specifically engineered dense networks that are smaller than standard architectures and can run with limited resources on the edge device. Furthermore, motivated by [41, 33], many works [53, 57] have shown that pruning at initialization without optimizing network topology during training can also perform equally well.

B More Experiments

B.1 The Statistics of eNRF Sizes with Different Sparsity Ratios

The distributions of eNRF sizes in the first dynamic sparse CNN layer for HAR, Daily Sport and EEG2 datasets are shown in Figure 9, Figure 10 and Figure 11, respectively. We can see that DSN model with kernel grouping can cover more diverse eNRF sizes than it does without kernel grouping under different sparsity ratios S. With the decrease of S, the DSN model without kernel grouping tends to capture the large eNRF, while ignoring the small eNRF, which harms the performance of DSN model. Under different sparsity ratios, the DSN model with kernel grouping can guarantee a satisfactory performance for different datasets by covering various eNRF.

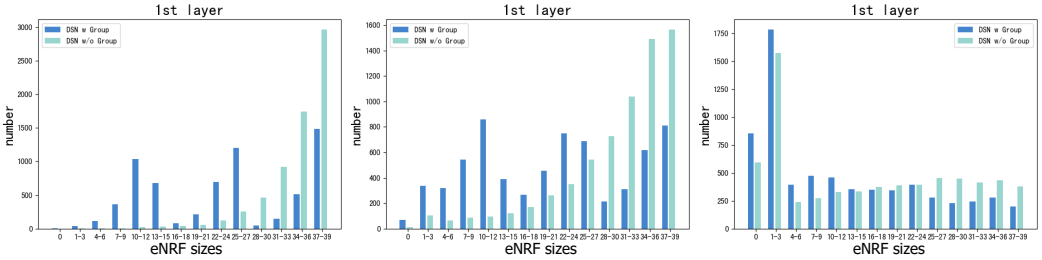

Figure 9: The statistics of eNRF sizes for the HAR dataset. The eNRF sizes shown in a sub-figure are from the first dynamic sparse CNN layer with a specific sparsity ratio (e.g. $S = 50\%$ (left), $S = 70\%$ (middle), and $S = 90\%$ (right)).

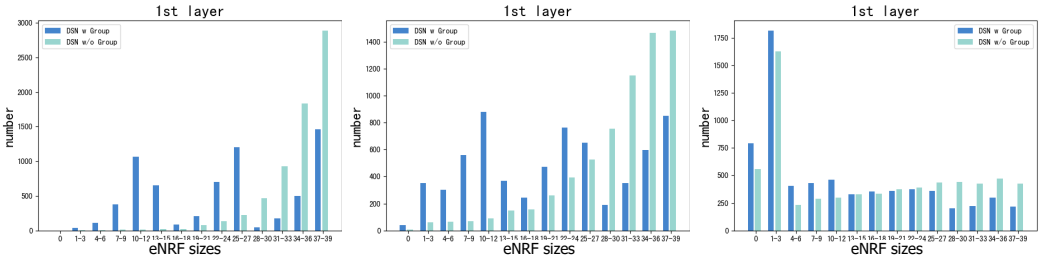

Figure 10: The statistics of eNRF sizes for the Daily Sport dataset.The eNRF sizes shown in a sub-figure are from the first dynamic sparse CNN layer with a specific sparsity ratio (e.g. $S = 50\%$ (left), $S = 70\%$ (middle), and $S = 90\%$ (right)).

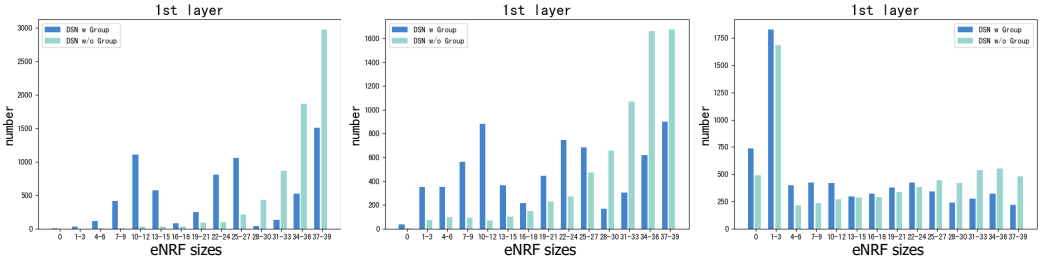

Figure 11: The statistics of eNRF sizes for the EEG2 dataset.The eNRF sizes shown in a sub-figure are from the first dynamic sparse CNN layer with a specific sparsity ratio (e.g. $S = 50\%$ (left), $S = 70\%$ (middle), and $S = 90\%$ (right)).

B.2 Better Kernel Sizes for EEG2 Dataset

We study the effect of kernel size k on the performance of spares/dense DSN for EEG2 dataset to verify our hypothesis that this dataset expects the local information. The dense DSN model is defined exactly as our DSN model with dense connections along entire kernels. In this way, the RF size in each layer equals to kernel size. As shown in Figure 12, the dense DSN with a kernel size of 9 achieves the highest test accuracy. However, with the increase of kernel size (RF size), the test accuracy drops dramatically. It is obvious that the local context plays a significant role for more accurate classification for EEG2 dataset. In contrast, our proposed sparse DSN model (sparsity ratio $S = 80\%$) with a large kernel size can still achieve satisfactory performance, with the main reason that it can learn the adaptive eNRF for the dataset.

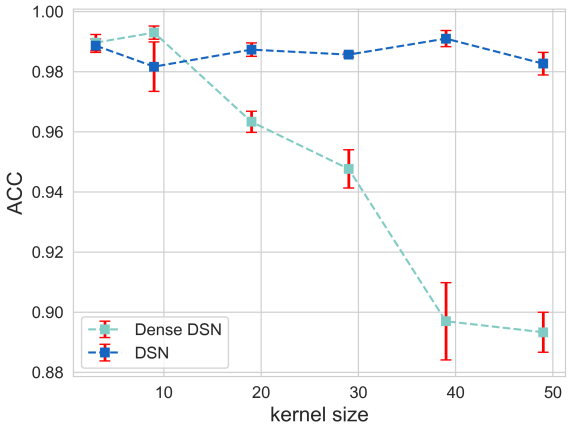

Figure 12: The test accuracies of EGG2 dataset in dense/sparse DSN with different kernel sizes k.

B.3 Can Various eNRF Outperform the Optimal NRF?

To demonstrate the advantage of various eNRF coverage, we compare the performances in terms of test accuracy between DSN models with various eNRF coverage and optimal NRF. The dense DSN models with different kernel sizes (from 5 to 40 with step 5) in each layer are used to search for optimal NRF in different datasets on the UCR 85 archive. As shown in Figure 13, by capturing various eNRF, our DSN model can achieve similar performance to the optimal NRF model without the cumbersome hyper parameters.

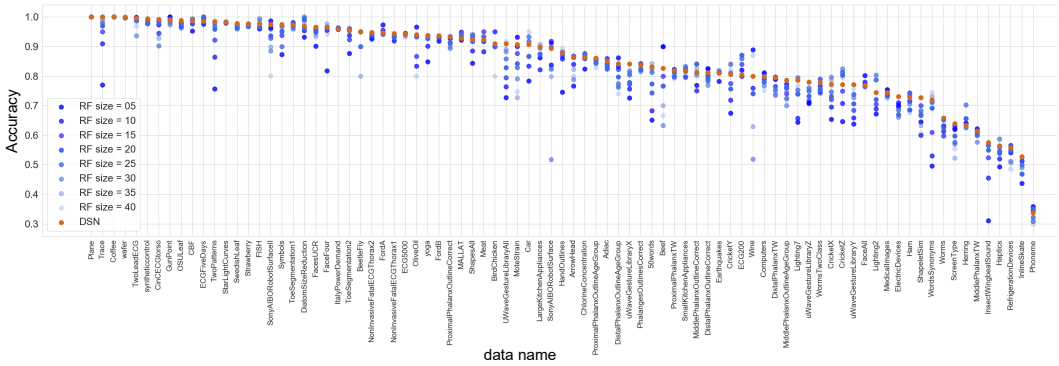

Figure 13: Classification accuracies on UCR 85 archive for the comparison of various eNRF and optimal NRF.

B.4 The Effect of Kernel Size for DSN

From Figure 14, we can find out how the kernel size k in the proposed DSN model affects the final test accuracy for UCR 85 archive. We can see that the performance of DSN model with small kernel sizes (e.g. 15 and 21) is worse than that of it with large kernel sizes (e.g. 39 and 45). It is because that the model with larger kernel sizes can cover larger eNRF which is useful to capture global information for some datasets. However, when the kernel size is oversized (e.g. 45), some of the small eNRFs can be lost in a predefined sparsity ratio (e.g. 80% in here), which will slightly degrade the performance instead.

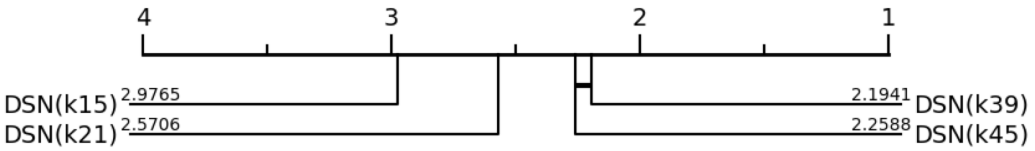

Figure 14: The critical difference diagram shows the average rank of the proposed DSN model with different kernel sizes in UCR 85 archive. DSN(ki) illustrates that the kernel size in each dynamic sparse CNN layer equals to i.

B.5 The Effect of Kernel Group for DSN

We study how the number of kernel groups N defined in Algorithm 1 affects the performance of proposed DSN model. From Figure 15, we can see that DSN model with $N = 2$ outperforms other counterparts. As discussed in 4.5 and B.1, grouping the kernels in each dynamic sparse CNN layer can help to achieve various NRF coverage especially with smaller sparsity ratio S. However, when the number of kernel groups (e.g. 4) is oversized, the distribution of eNRF sizes is nearly uniform, which will degrade the performance of DSN.

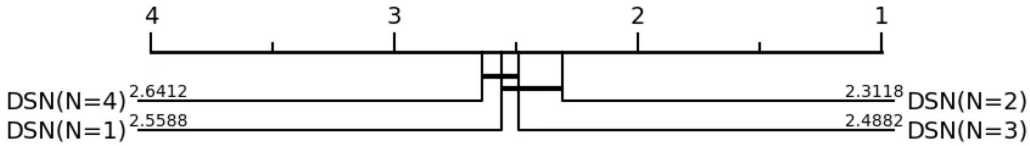

Figure 15: The critical difference diagram shows the average rank of the proposed DSN model with different kernel groups in UCR 85 archive. DSN($N = i$) illustrates that the kernels in each dynamic sparse CNN are split into i groups.

B.6 The Effect of Topology Initialization

As shown in Figure 16, the activated weights can be initialized randomly and sequentially before exploration, namely DSN$^{init}_{rand}$ and DSN$^{init}_{seq}$ respectively. After initialization, the topologies initialized by DSN$^{init}_{rand}$ and DSN$^{init}_{seq}$ are different. In detail, DSN$^{init}_{rand}$ randomly activates the weights within the exploration regions, while DSN$^{init}_{seq}$ activates the first $\frac{i \times k}{N} S$ weights of kernels in ith group. Figure 16 exemplifies the case where $k = 16$, $N = 4$ and $S = 50\%$. We further investigate which type of topology initialization are more suitable for the proposed DSN model. According to the results shown in Figure 17 (left) and Figure 18, we can see that the proposed DSN model with sequential initialization performs better than that with random initialization. Based on this, we adopt sequential initialization as the default setting for DSN model in all of the experiments.

Intuitively, the DSN method with random topology can also cover various eNRF. We further study the different performances of NRF which is from dynamic sparse training and from random topology in DSN method. We conduct ablation studies to compare DSN with its static variant (i.e. DSN$^{fix}_{rand}$), which fixes its topology during training after activated weights random initialization. According to

the results shown in Figure 17 (right), we can find that the DSN model outperforms the DSN^{fix}_{rand} on UCR 85 archive, indicating that what is learned by dynamic sparse training is the suitable activated weights together with their values.

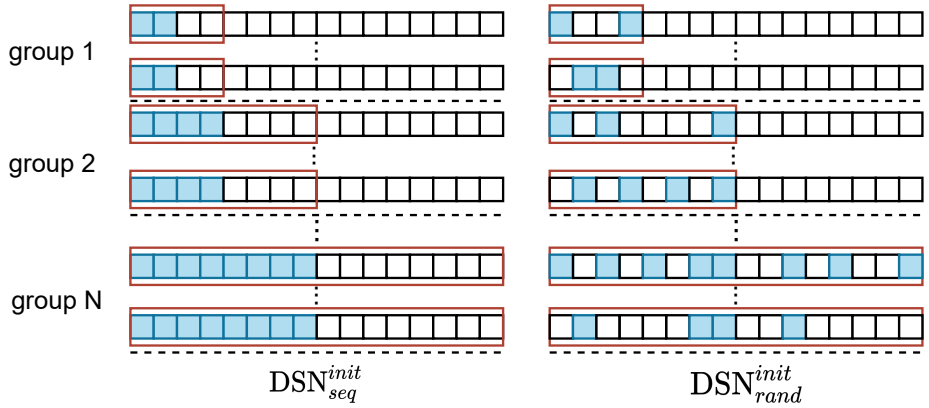

Figure 16: Visualization of different topology initialization manners, including sequential and random. The corresponding exploration regions for each group are shown in the red boxes. The blue color indicates the activated weights at initialization.

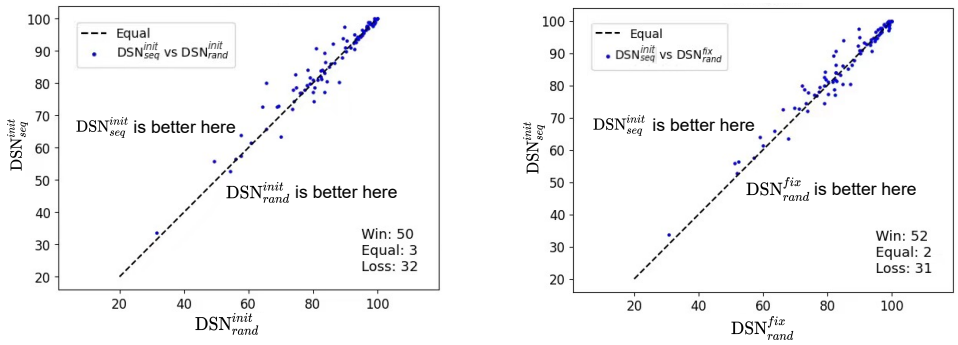

Figure 17: Accuracy plotting showing how the proposed DSN model is affected by the topology initialization on UCR 85 archive.

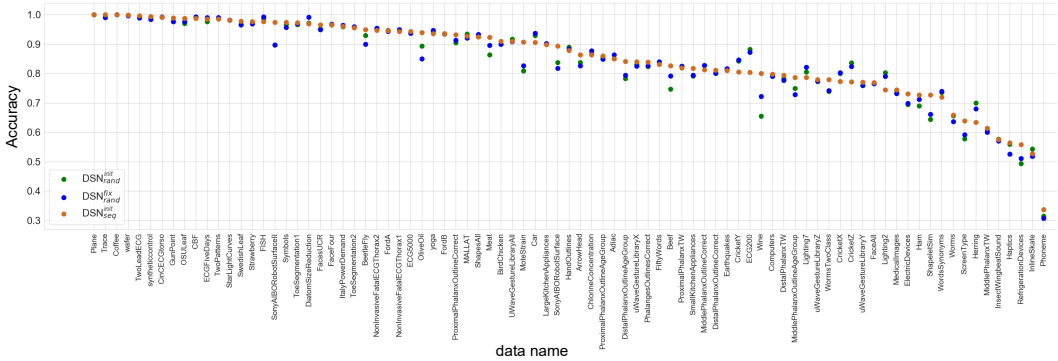

Figure 18: Classification accuracies on UCR 85 archive for the comparison of DSN^{init}_{seq} and DSN^{init}_{rand} with/without dynamic exploration.

B.7 Extremely Sparse DSN

We study how the performance of our DSN is affected under extremely sparse ratios (e.g. $S = 90\%$, $S = 95\%$ and $S = 97\%$). As shown in Table 5, we can see that our DSN with sparsity ratio $S = 95\%$ still performs better than OS-CNN while achieving more than $5\times$ resource cost reduction (e.g. Parameters and FLOPs). When sparsity ratio $S = 97\%$, which means that 97% weights in sparse CNN layer are set to zeros, the performance of our DSN model is only slightly worse than that of OS-CNN, but the resource cost is nearly $7\times$ less. From Figure 19, we can see that most blue dots (corresponding to accuracies of OS-CNN) are below the orange dots (corresponding to accuracies of DSN with extremely sparsity ratio), which means that OS-CNN performs worse than extremely sparse DSN in most dataset. What's more, DSN is far superior to OS-CNN in some datasets (e.g. CinCECGtorso, MiddlePhalanxOutlineAgeGroup and Earthquakes).

Table 5: Pairwise comparison of test accuracy (%) and mean resource cost (i.e. Params $(K)\downarrow$ and FLOPs $(M)\downarrow$) of our method with extremely sparsity ratio and oS-CNN on UCR 85 archive datasets. Note that all the accuracies in comparison are rounded to two decimal places.

Archive	Methods	OS-CNN wins	**DSN**(S) wins	Tie	Params	FLOPs
UCR 85 archive	OS-CNN	-	-	-	262.14	200.12
	DSN($S = 80\%$)	33	**49**	3	126.22	104.88
	DSN($S = 85\%$)	33	**49**	3	100.19	83.21
	DSN($S = 90\%$)	36	**47**	2	74.16	61.54
	DSN($S = 95\%$)	38	**43**	4	48.13	39.87
	DSN($S = 97\%$)	**41**	40	4	**37.72**	**31.20**

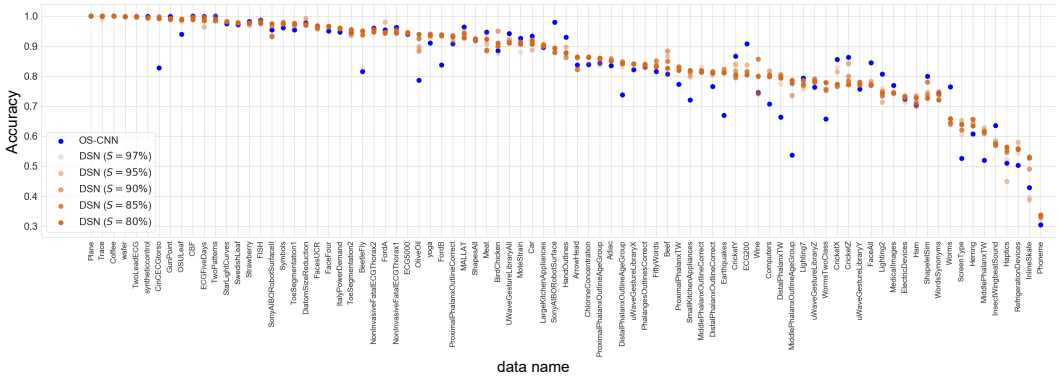

Figure 19: Classification accuracies on UCR 85 archive for DSN with different sparsity ratios.

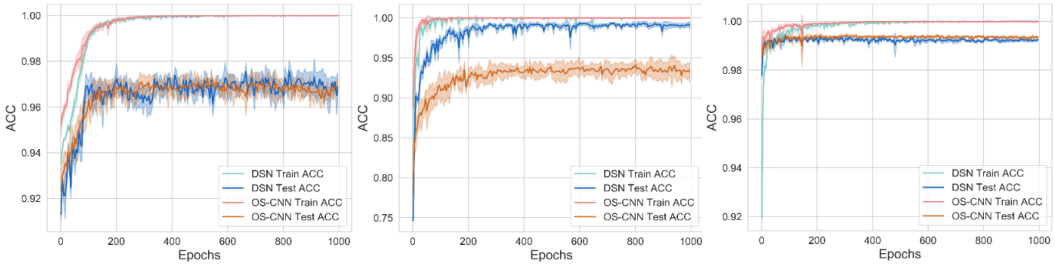

Figure 20: Learning curves of DSN models on different datasets, HAR(left), EEG2(middle) and Daily Sports(right). The standard deviation of the accuracy over 5 runs is shown in the shaded region.

B.8 Learning Curve on Three Multivariate TS Dataset

Figure 20 shows the learning behaviors of our DSN model and OS-CNN. We can see that our DSN model has a convergence speed similar to that of OS-CNN, while the resource requirements (memory and computational cost) are much less. Furthermore, the performance of DSN is far superior to that of OS-CNN for EEG2 datasets .

B.9 Additional Experiments on other TSC datasets

In order to evaluate if the good learning capabilities of DSN can be generalized over other larger scale of univariate and multivariate time-series datasets we evaluate it on the following datasets:

- **UCR 2018 archive** [6]. This archive consists of 128 univariate TS datasets, which is the updated version of the UCR 85 archive. Among them, 15 datasets are with unequal length and one (Fungi) has a single instance per class in the training files. As shown in [40], most implementations are not set up to handle these kinds of data, we also only report our results on the remaining 112 problems in UCR 2018 archive.
- **University of East Anglia (UEA) 30 archive** [2]. It consists of 30 multivariate TS datasets with distinguishable characteristics, and various levels of complexity. The class numbers vary from 2 to 39, and their numbers of variates vary from 2 to 963.

For UEA 30 archive, we add the resource awareness method Rocket [7] and the transformer based method TST [64] as our baselines.We summarize the performance on both additional univariate and multivariate TSC benchmarks in Table 6-7. We can see that, in terms of test accuracy, our method can still match the state-of-the-art methods, while having much smaller computational cost (e.g. the number of parameters and FLOPs). It is worth highlighting our proposed DSN method achieves a trade-off between the resource cost and performance in both univariate and multivariate TSC benchmarks.

Table 6: Pairwise comparison of test accuracy (%) and mean resource cost (i.e. Params $(K)\downarrow$ and FLOPs $(M)\downarrow$) of our method and other univariate TS baseline methods on UCR 112 archive datasets. Note that all the accuracies in comparison are rounded to two decimal places.

Archive	Methods	Baseline wins	**DSN** wins	Tie	Params	FLOPs
UCR 112 archive	ResNet	43	**64**	5	479.28	1049.12
	InceptionTime	**52**	51	9	389.42	425.06
	OS-CNN	48	**59**	5	264.19	259.30
	DSN (ours)	-	-	-	**126.30**	**136.91**

Table 7: Pairwise comparison of test accuracy (%) and mean resource cost (i.e. Params $(K)\downarrow$ and FLOPs $(M)\downarrow$) of our method and other multivariate TS baseline methods on UEA 30 archive datasets. Note that all the accuracies in comparison are rounded to two decimal places.

Archive	Methods	Baseline wins	**DSN** wins	Tie	Params	FLOPs
UEA 30 archive	TapNet	11	**17**	2	1344.67	966.46
	MLSTM-FCN	7	**23**	0	383.41	638.53
	OS-CNN	**15**	14	1	389.65	547.34
	TST	12	**18**	0	343.78	628.95
	Rocket	**15**	14	1	-	526.87[4]
	DSN (ours)	-	-	-	**148.74**	**305.37**

[4]For transform operation with 1000 random convolution kernels.

C Detailed Results

For UCR 85 archive, the reported results of ResNet and InceptionTime are from [7], while that of OS-CNN are from the official repository [5]. For three multivariate datasets from UCI, the reported results of LSTM-FCN are from the original paper, and that of TapNet and OS-CNN are from the implementation of official repository with default settings. For UEA 30 archive, the detailed results including test accuracy and resource cost on each dataset can be found in Table 9.

The detailed results for UCR 85 archive are given in Table 8. We also visualize the detailed results in Figure 21, we can see that the accuracies of DSN are nearly on the top in most of the datasets. In addition, it is worth noting that DSN loses by a slight margin in some datasets, but wins by a big margin in others.

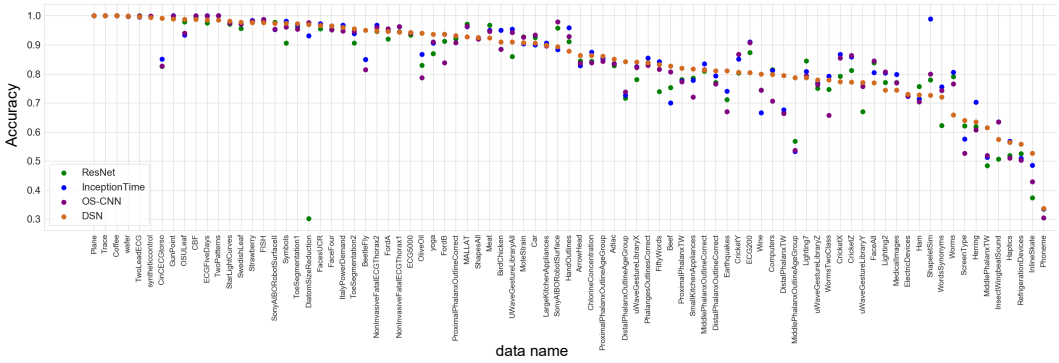

Figure 21: Visualizing classification accuracies on UCR 85 archive for the comparison of DSN and the baselines.

D More Implementation Details

In our dynamic sparse training algorithm 1, we choose the cosine annealing defined in Eq. 4 with $\Delta T = 5$ epochs and $\alpha = 0.5$. For UEA 30 archive, in each dynamic sparse CNN layer, the output channel is set to 177 with sparsity ratio $S = 90\%$. For UCR 112 archive, except for each CNN layer is padded with zero-value, other hyper-parameters are the same as in UCR 85 archive. We use PyTorch[6] to implement our method and run our experiments on Nvidia Tesla P100.

[5] https://github.com/Wensi-Tang/OS-CNN.

[6] https://pytorch.org/

Table 8: Test accuracies (ACC(%)) for UCR 85 archive and resource cost (i.e. Params (K) and FLOPs (M)) of our method. Test accuracies of our method are run five times and reported with (mean±std).

	ResNet	InceptionTime	OS-CNN	DSN (ours)	Params	FLOPs
50Words	73.96	84.18	81.60	83.16±0.69	132.27	68.36
Adiac	82.89	83.63	83.45	85.06±0.73	130.41	44.68
ArrowHead	84.46	82.86	83.77	86.40±0.23	125.55	62.31
Beef	75.33	70.00	80.67	82.67±2.49	125.83	116.65
BeetleFly	85.00	85.00	81.50	95.00±0.00	125.40	126.97
BirdChicken	88.50	95.00	88.50	91.00±2.00	125.40	126.97
CBF	99.50	99.89	99.99	98.78±0.31	125.55	31.82
Car	92.50	90.00	93.33	90.67±2.26	125.69	143.14
ChlorineConcentration	84.36	87.53	83.87	86.33±0.20	125.55	41.24
CinCECGtorso	82.61	85.14	82.75	99.20±0.18	125.69	406.37
Coffee	100.00	100.00	100.00	100.00±0.00	125.40	70.95
Computers	81.48	81.20	70.68	79.84±0.54	125.40	178.53
CricketX	79.13	86.67	85.51	77.28±1.57	126.83	74.71
CricketY	80.33	85.13	86.72	80.51±1.04	126.83	74.71
CricketZ	81.15	85.90	86.31	77.13±1.85	126.83	74.71
DiatomSizeReduction	30.13	93.14	97.71	96.99±0.48	125.69	85.63
DistalPhalanxOutlineAgeGroup	71.65	72.66	73.81	84.20±0.33	125.55	19.92
DistalPhalanxOutlineCorrect	77.10	79.35	76.59	81.13±1.09	125.40	19.89
DistalPhalanxTW	66.47	67.63	66.40	79.40±0.37	125.98	20.00
ECG200	87.40	91.00	90.80	80.40±0.49	125.40	23.85
ECG5000	93.42	94.09	94.01	94.32±0.12	125.83	34.85
ECGFiveDays	97.48	100.00	99.95	98.51±0.48	125.40	33.77
Earthquakes	71.15	74.10	66.98	81.06±0.00	125.40	126.97
ElectricDevices	72.91	72.27	72.36	73.02±0.74	126.12	24.00
FISH	97.94	98.29	98.74	97.71±0.72	126.12	114.97
FaceAll	83.88	80.41	84.47	76.91±0.16	127.12	32.87
FaceFour	95.45	96.59	95.11	96.59±0.00	125.69	86.88
FacesUCR	95.47	97.32	96.74	96.62±0.23	127.12	32.87
FordA	92.05	94.83	95.48	94.64±0.16	125.40	124.00
FordB	91.31	93.65	83.79	93.62±0.67	125.40	124.00
GunPoint	99.07	100.00	99.93	98.93±0.53	125.40	37.24
Ham	75.71	71.43	70.38	72.76±1.66	125.40	106.89
HandOutlines	91.11	95.95	92.95	87.90±0.68	125.40	671.54
Haptics	51.88	56.82	51.01	56.43±2.30	125.83	270.82
Herring	61.88	70.31	60.78	63.44±4.49	125.40	126.97
InlineSkate	37.31	48.55	42.93	52.73±4.52	126.12	466.69
InsectWingbeatSound	50.65	63.48	63.52	57.49±0.52	126.69	63.77
ItalyPowerDemand	96.30	96.79	94.72	96.05±0.13	125.40	6.01
LargeKitchenAppliances	89.97	90.67	89.57	89.92±0.26	125.55	178.56
Lighting2	77.05	80.33	80.66	74.43±2.45	125.40	157.95
Lighting7	84.52	80.82	79.32	78.63±1.40	126.12	79.27
MALLAT	97.16	96.29	96.38	92.71±1.83	126.26	254.05
Meat	96.83	95.00	94.67	92.33±2.91	125.55	111.13
MedicalImages	77.03	79.87	76.95	74.39±0.24	126.55	24.82
MiddlePhalanxOutlineAgeGroup	56.88	53.25	53.64	78.65±0.77	125.55	19.92
MiddlePhalanxOutlineCorrect	80.89	83.51	81.41	81.33±1.27	125.40	19.89
MiddlePhalanxTW	48.44	51.30	51.95	61.45±0.40	125.98	20.00
MoteStrain	92.76	90.34	92.64	90.73±0.43	125.40	20.88
NonInvasiveFatalECGThorax1	94.54	96.23	96.27	94.36±0.50	131.12	187.11
NonInvasiveFatalECGThorax2	94.61	96.74	96.01	94.72±0.33	131.12	187.11
OSULeaf	97.85	93.39	94.01	98.84±0.31	125.98	106.01
OliveOil	83.00	86.67	78.67	94.00±3.27	125.69	141.40
PhalangesOutlinesCorrect	83.90	85.43	82.97	83.87±0.78	125.40	19.89
Phoneme	33.43	33.54	30.45	33.70±0.79	130.70	254.93
Plane	100.00	100.00	100.00	100.00±0.00	126.12	35.89
ProximalPhalanxOutlineAgeGroup	85.32	85.37	84.39	86.05±0.50	125.55	19.92
ProximalPhalanxOutlineCorrect	92.13	93.13	90.79	93.20±0.14	125.40	19.89
ProximalPhalanxTW	78.05	77.56	77.32	81.95±0.29	125.98	20.00
RefrigerationDevices	52.53	50.93	50.29	55.84±1.27	125.55	178.56
ScreenType	62.16	57.60	52.64	63.95±1.11	125.55	178.56
ShapeletSim	77.94	98.89	79.94	72.67±4.87	125.40	124.00
ShapesAll	92.13	92.50	92.03	92.50±0.61	133.70	128.63
SmallKitchenAppliances	78.61	77.87	72.08	81.76±0.40	125.55	178.56

	ResNet	InceptionTime	OS-CNN	DSN (ours)	Params	FLOPs
SonyAIBORobotSurface	95.81	88.35	97.95	89.32±1.82	125.40	17.41
SonyAIBORobotSurfaceII	97.78	95.28	95.38	97.46±0.75	125.40	16.17
StarLightCurves	97.18	97.92	97.50	98.20±0.09	125.55	253.91
Strawberry	98.05	98.38	98.19	97.72±0.29	125.40	58.31
SwedishLeaf	95.63	97.12	97.12	97.79±0.19	127.26	32.16
Symbols	90.64	98.19	96.12	97.43±0.40	125.98	98.82
ToeSegmentation1	96.27	96.93	95.39	97.28±0.33	125.40	68.72
ToeSegmentation2	90.62	93.85	94.62	95.54±0.58	125.40	85.08
Trace	100.00	100.00	100.00	100.00±0.00	125.69	68.28
TwoLeadECG	100.00	99.56	99.92	99.70±0.07	125.40	20.38
TwoPatterns	99.99	100.00	100.00	98.50±0.26	125.69	31.84
UWaveGestureLibraryAll	85.95	95.45	94.25	90.99±0.38	126.26	234.46
Wine	74.44	66.67	74.44	80.00±2.96	125.40	58.06
WordsSynonyms	62.24	75.55	74.23	72.01±1.38	128.69	67.64
Worms	79.09	80.52	76.49	65.86±1.83	125.83	223.23
WormsTwoClass	74.68	79.22	65.71	77.90±1.35	125.40	223.15
syntheticcontrol	99.83	99.67	99.93	99.40±0.13	125.98	15.05
uWaveGestureLibraryX	78.05	82.47	82.18	84.06±0.39	126.26	78.31
uWaveGestureLibraryY	67.01	76.88	75.72	77.10±0.64	126.26	78.31
uWaveGestureLibraryZ	75.01	76.97	76.36	77.98±0.41	126.26	78.31
wafer	99.86	99.87	99.84	99.91±0.02	125.40	37.74
yoga	87.02	90.57	91.06	93.63±0.48	125.40	105.65

Table 9: Test accuracies (ACC(%)) for UEA 30 archive and resource cost (i.e. Params (K) and FLOPs (M)) of our method. Test accuracies of our method are run five times and reported with (mean±std).

	MLSTM-FCN	TapNet	OS-CNN	DSN (ours)	Params	FLOPs
ArticularyWordRecognition	97.30	98.70	98.75	98.40±0.25	146.35	41.58
AtrialFibrillation	26.70	33.30	23.33	6.67±0.00	142.00	180.42
BasicMotions	95.00	100.00	100.00	100.00±0.00	142.42	28.37
CharacterTrajectories	98.50	99.70	99.76	99.39±0.07	145.10	52.02
Cricket	91.70	95.80	99.31	98.89±0.56	143.85	338.23
DuckDuckGeese	67.50	57.50	54.00	56.80±3.25	221.60	119.04
ERing	13.30	13.30	88.15	92.22±1.12	142.66	18.55
EigenWorms	50.40	48.90	41.41	39.08±11.21	142.59	5075.34
Epilepsy	76.10	97.10	98.01	99.86±0.29	142.24	58.21
EthanolConcentration	37.30	32.30	24.05	24.49±0.89	142.24	493.68
FaceDetection	54.50	55.60	57.50	63.49±0.70	150.20	18.58
FingerMovements	58.00	53.00	56.75	49.20±1.17	143.36	14.32
HandMovementDirection	36.50	37.80	44.26	37.30±2.20	142.65	113.22
Handwriting	28.60	35.70	66.82	33.65±0.84	146.18	43.78
Heartbeat	66.30	75.10	48.90	78.34±0.79	145.30	117.00
InsectWingbeat	16.70	20.80	66.66	38.62±10.65	154.94	7.07
JapaneseVowels	97.60	96.50	99.12	98.86±0.11	143.66	8.53
LSST	37.30	56.80	41.25	60.26±4.53	144.21	10.66
Libras	85.60	85.00	95.00	96.44±0.27	144.15	13.22
MotorImagery	51.00	59.00	53.50	57.40±2.58	145.48	867.23
NATOPS	88.90	93.90	96.81	97.78±0.93	143.84	14.72
PEMS-SF	69.90	75.10	76.01	80.12±1.25	199.42	57.15
PenDigits	97.80	98.00	98.55	98.73±0.10	143.25	2.62
PhonemeSpectra	11.00	17.50	29.93	31.97±0.36	148.98	62.77
RacketSports	80.30	86.80	87.66	86.18±1.32	142.42	8.61
SelfRegulationSCP1	87.40	65.20	83.53	71.74±0.79	142.06	252.93
SelfRegulationSCP2	47.20	55.00	53.19	46.44±4.54	142.12	325.32
SpokenArabicDigits	99.00	98.30	99.66	99.10±0.21	143.90	23.85
StandWalkJump	6.70	40.00	38.33	38.67±4.99	142.12	705.04
UWaveGestureLibrary	89.10	89.40	92.66	91.56±0.44	142.95	89.08