Class-Specific Attention (CSA) for Time-Series Classification

Yifan Hao*
yifan@nmsu.edu
New Mexico State University
Las Cruces, NM, USA

Huiping Cao
hcao@nmsu.edu
New Mexico State University
Las Cruces, NM, USA

K. Selçuk Candan
candan@asu.edu
Arizona State University
Tempe, AZ, USA

Jiefei Liu
jiefei@nmsu.edu
New Mexico State University
Las Cruces, NM, USA

Huiying Chen
hchen@nmsu.edu
New Mexico State University
Las Cruces, NM, USA

ABSTRACT

Most neural network-based classifiers extract features using several hidden layers and make predictions at the output layer by utilizing these extracted features. We observe that not all features are equally pronounced in all classes; we call such features *class-specific features*. Existing models do not fully utilize the class-specific differences in features as they feed all extracted features from the hidden layers equally to the output layers. Recent attention mechanisms allow giving different emphasis (or attention) to different features, but these attention models are themselves class-agnostic. In this paper, we propose a novel class-specific attention (CSA) module to capture significant class-specific features and improve the overall classification performance of time series. The CSA module is designed in a way such that it can be adopted in the existing neural network (NN) based model to conduct time series classification. In the experiments, this module is plugged into five start-of-the-art neural network models for time series classification to test its effectiveness by using 40 different real datasets. Extensive experiments show that a model embedded with the CSA module can improve the base model in most cases and the accuracy improvement can be up to 42%.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Time Series analysis** → *Attention.*

KEYWORDS

Time series classification; Neural networks; Attention

ACM Reference Format:
Yifan Hao, Huiping Cao, K. Selçuk Candan, Jiefei Liu, and Huiying Chen. 2022. Class-Specific Attention (CSA) for Time-Series Classification. In *Proceedings of SIGKDD International workshop on mining and learning from time series (MiLeTS).* Washington DC, USA, 9 pages.

MiLeTS, Aug 15, 2022, Washington DC

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

1 INTRODUCTION

Time series analysis (e.g., classification, anomaly detection) is becoming more critical in applications that collect increasing amount of time series data. Different neural network (NN)-based approaches, such as Long Short-Term Memory (LSTM) model [9], Convolutional Neural Networks (CNNs) [19], have been introduced for time series analysis. Fully-Convolutional Networks (FCN) are considered as a strong baseline in time series classification [17]. Most neural network models, either CNN or LSTM based, have several *hidden layers* (e.g., convolutional layers or LSTM layers) to generate temporal features – only the last several layers (e.g., global pooling layers or fully-connected layers) are used as *output layers* to make predictions [6, 10, 11, 19, 22].

Naturally, each extracted feature may not equally contribute to the time series classification of instances to different classes [8]: some features may be more effective for one class while other features may be more pronounced in instances of another class. If one feature mostly comes from instances in one specific class, in this paper, we refer to this feature as a *class-specific feature.*

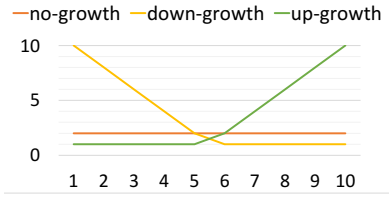


Figure 1: Time-series examples from different classes

Example 1.1 (Class-specific features). Fig. 1 shows the stock prices of three companies. The red curve shows the stock prices from a stable performing but no-growth company, without any price changes. The green curve shows the prices from an up-growth company and the yellow curve shows the prices of a down-growth company. It is easy to see that, in this example, the sub-sequences from time step 1 to 5 can easily separate the down-growth company from the other two; sub-sequences from 6 to 10 would be more helpful to differentiate the up-growth company from the other two companies; and the whole time range from 1 to 10 may be needed to differentiate the no-growth company from the down-growth and up-growth companies.

While the existence of class-specific features have been acknowledged and leveraged in the context of improving model explainability [8], unfortunately they are not well-utilized for the primary classification tasks by the conventional neural networks models, which generally feed all the features generated by the hidden layers to the output layers [10, 11, 17].

In recent years, attention mechanisms [2, 6, 16] have been introduced and widely leveraged to allow providing different levels of emphasis (or attention) to different features. Extensive research has shown that such attention mechanisms can improve the performance of neural network models in different applications. However, these attention mechanisms themselves tend to be class-agnostic and, to the best of our knowledge, for a general time-series classification task, no existing attention mechanism explicitly incorporates class-specific features for generating emphasis.

To make use of the class-specific features to improve the performance of general NN models, this paper proposes a novel Class-Specific Attention (CSA) module that captures class-specific features to learn class-specific attention, without negatively impacting the testing phase. The CSA module is different from existing attention designs [2, 6, 16], since CSA utilizes class labels to calculate attention.

The main **contributions** of this paper are:

- We propose a Class-Specific Attention (CSA) module to capture class-specific features and learn class-specific attention from (multivariate) time series data. This module is general and can be adopted in most neural network based models for time series classification (with or without attention) to improve their performance.
- We design a Class-differentiation component to better identify the specific and important features to one class. The CSA module leans class-specific features in the hidden layers during training and easy to be used in testing.
- The CSA module is applied to five state-of-the-art neural network models for time series classification on 40 real datasets (28 multivariate time-series (MTS) and 12 univariate time-series (UTS)). Extensive experiments show that a model with CSA module provides better overall performance (with accuracy gains up to 42%) compared to the same model without CSA.

The rest of this paper is organized as follows. Section 2 discusses the related works. Section 3 presents the architecture, the learning algorithm, and the utilization of the newly proposed CSA module. Section 4 reports our experimental results to evaluate the performance of the CSA module. Section 5 concludes this work and discusses future directions.

2 RELATED WORKS

Time series classification is a well-studied problem with extensive literature on the topic. Recently, deep learning models have shown great success in time series classification. Convolutional Neural Networks (CNNs) have been exploited in time series classification because shallow convolutional layers can capture short-term temporal dependencies and deep convolutional layers can encode long-term temporal dependencies. Various CNN models are designed to classify multivariate time series (MTS) data [19, 22]. In [17], a Fully-Convolutional Network (FCN) is proposed. It is considered as a strong baseline for time series classification.

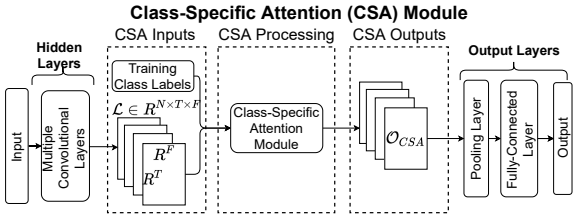


Figure 2: The overview of the an FCN Model, extended with Class-Specific Attention (CSA)

Long Short-Term Memory (LSTM) [9, 13, 14] and Gated Recurrent Unit (GRU) [5] based models are also widely used in temporal analysis. Both *LSTM* and *GRU* capture temporal features and makes predictions by using gating functions in their dynamics states. To make good use of the *FCN* and *LSTM* models, Karim et al. [10, 11] further propose two state-of-the-art models, *LSTM-FCN* and *MLSTM-FCN* by combining both *FCN* and *LSTM* to classify uni-variate and multivariate time series. Among these models, class-specific features are not utilized.

Recently, the attention mechanism [2] has been widely deployed to solve various types of problems, including regression analysis (e.g., [12, 14]) and time series analysis [4, 6, 10, 11, 15, 18, 21]. Despite all the success that attention mechanisms bring, existing attention mechanisms cannot be directly applied to capture class-specific features.

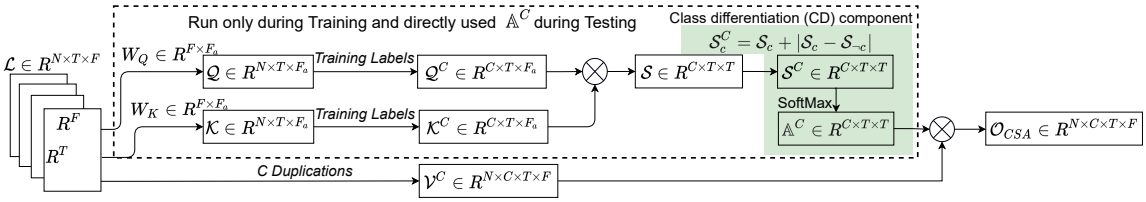
Using class-label information in classification models has been explored. [8] and [7] apply class information for well-trained models to improve either the model explainability or the classification performance. TapNet [20] uses class label information to build a prototype (a component inside their model) for each class; they average the features of the instances for the same class label and apply softmax on the different prototypes. This is different from our design of calculating attention from class-specific key and query features [2].

3 CLASS-SPECIFIC ATTENTION (CSA)

It is possible that features generated from hidden layers of neural network models do not equally contribute to classifying different classes. Furthermore, features that are important to classify instances belonging to one class may not be useful in classifying instances belonging to other classes [8]. This phenomenon becomes more obvious in time-series classification when a sub-sequence of a time series, instead of the whole sequence, is more useful to classify instances from specific classes (as in Eg. 1.1). In this section, we describe a *Class-Specific Attention (CSA)* module that builds on the above observations to identify and leverage class-specific features in time series classification.

3.1 Overall Design

We explain the core idea and overall design of the CSA module using Fig. 2. The CSA module is designed in a way such that it can be adopted in existing neural network (NN) based time series classification models. Without loss of generality, we use the FCN [17]



⊗ denotes matrix multiplication and ⊕ denotes matrix addition

Figure 3: The architecture of the Class-Specific Attention (CSA) Module – notation convention: (1) A symbol using AMS blackboard bold font ($\mathbb{A}^C$) represents a tensor learning from training instances, and directly used for testing; (2)the other tensors are represented using math calligraphic font style (e.g., $\mathcal{L}$, $\mathcal{K}$, $\mathcal{V}$, $\mathcal{S}$); (3) class-specific tensors are denoted with a superscript C (e.g., $\mathcal{Q}^C$, $\mathcal{K}^C$, $\mathcal{V}^C$, $\mathcal{S}^C$); (4) kernel weights are represented using W with a subscript (W_Q, W_K); (5) dimension sizes are represented as superscript for R (e.g., $R^{N\times T}$).

model as a representative NN model to explain the design of our newly proposed CSA module. In this figure, the CSA module is used to extend the FCN model for time series classification for illustration purposes. The CSA module is shown in the three dotted-line boxes at the center of the figure. Note that, while CSA can be placed after any hidden layer of a neural network model, in this figure (and in the experiments reported in the paper), we place it after the last hidden layer of the neural model, as later layers capture more higher-level features than the previous layers [8]. CSA takes a special form of input that leverages the class label information of training instances – such labels are generally used by existing neural network models for evaluating the loss value from the output layer, but are not explicitly utilized for learning hidden features. In contrast to the state-of-the-art, the CSA input data integrates class label information of training instances with the features learned from previous hidden layers (the detailed explanation of the matrix $\mathcal{L}$ can be found in Section 3).

CSA gives different attention (or emphasis) to features generated in the hidden layers to learn class-specific features, which are then used to evaluate the probabilities of the data instances belonging to different classes. The design of the CSA module adopts the basic idea of self-attention mechanism [16] where the features in the three hidden spaces (key, value, query) are learned from the same input data. On the other hand, CSA differs from self-attention mechanisms in two major aspects:

- First, CSA is designed to learn the class-specific features in the training stage and preserve these features in the *hidden query space*. Yet, the design of the query space, which encodes the class-specific features, can be directly utilized in the testing stage without knowing the class labels of testing instances.
- Typical attention mechanisms calculate attention values by using the similarity between the key features and the query features; however, such a calculation cannot differentiate features that commonly exist in all the instances from the class-specific features that only exist in instances of specific classes. In contrast, attention calculation in the CSA module *differentiates the importance of class-specific features from other features.*

Table 1 lists the symbols used in the method description.

Symbol	Meaning
N	# of instances in a TS dataset
C	# of distinct classes in a dataset
V	# of variables in a TS dataset
T	# of time points of one time series in X
$\mathcal{L}$	Feature matrix from hidden layers
F	# of features at each time point in X
F_a	# of features in the hidden key and query spaces
B	# of instances in one batch

Table 1: Symbols used in this paper

3.2 Overview of the CSA Architecture

Fig. 3 presents a detailed view of the CSA architecture. In this scenario, the features from the hidden layers of the FCN model are fed as input to the CSA module – in particular, we assume that features are generated by the hidden layers that are immediately before the CSA module in the architecture.

The CSA architecture outlined in Fig. 3 embodies our key design decisions:

Design Decision #1. Unlike the existing self-attention mechanisms, CSA introduces the class-specific attention $\mathbb{A}^C$ to differentiate class-specific features from other features. In particular, after learning the features $\mathcal{Q}$ and $\mathcal{K}$ in the latent query and key spaces, CSA aggregates the instances in the same class in these two feature spaces to get two class-specific feature tensors $\mathcal{Q}^C$ and $\mathcal{K}^C$ so that they can be further utilized in calculating the class-specific attention $\mathbb{A}^C$ and the updated features $\mathcal{O}_{CSA}$.

Design Decision #2. The class-specific attention $\mathbb{A}^C$ is designed to be a global tensor. It is updated during the training stage and can be directly used in testing. Consequently, the CSA design avoids the need for the testing instances to have class labels.

3.3 Training Process

The ultimate goal of the CSA training is to convert regular features (denoted as $\mathcal{L}$ in Fig. 3) to class-specific features ($\mathcal{O}_{CSA}$ in Fig. 3). This conversion utilizes the class-specific attention $\mathbb{A}^C$.

The *CSA_Calculation* algorithm, shown in Algorithm 1, provides an overview of the learning process for CSA training. First, for

Algorithm 1 *CSA_Calculation*

Input: $\mathcal{L} \in R^{N \times T \times F}$: feature tensor from the hidden layers
Output: $O_{CSA} \in R^{N \times C \times T \times F}$: feature tensor adjusted using class-specific attention

1: $\mathcal{K} = \mathcal{L} \cdot W_K$ where $W_K \in R^{F \times F_a}$
2: $\mathcal{Q} = \mathcal{L} \cdot W_Q$ where $W_Q \in R^{F \times F_a}$
3: Initialize $\mathcal{K}^C \in R^{C \times T \times F}$, $\mathcal{Q}^C \in R^{C \times T \times F}$
4: **for** each class label c **do**
5: $\mathcal{L}_c$ = all the instances belonging to class c.
6: $\mathcal{K}^C[c, :, :] = average(\mathcal{K}[\mathcal{L}_c, :, :])$ on the first dimension
7: $\mathcal{Q}^C[c, :, :] = average(\mathcal{Q}[\mathcal{L}_c, :, :])$ on the first dimension
8: **end for**
9: $\mathcal{S} = \mathcal{K}^C \cdot (\mathcal{Q}^C)^T$ where $\mathcal{S} \in R^{C \times T \times T}$
10: Initialize $\mathcal{S}^C \in R^{C \times T \times T}$
11: **for** each class label c **do**
12: $\mathcal{S}_c = \mathcal{S}[c, :, :]$
13: $\mathcal{S}_{\neg c} = \frac{SUM(\mathcal{S}) - \mathcal{S}_c}{C-1}$
14: $\mathcal{S}^C[c, :, :] = \mathcal{S}_c + abs(\mathcal{S}_c - \mathcal{S}_{\neg c})$
15: **end for**
16: $\mathbb{A}^C = SoftMax(\mathcal{S}^C) \in R^{C \times T \times T}$
17: $\mathcal{V} = \mathcal{L} \cdot W_V$, where $W_V \in R^{F \times F}$
18: $\mathcal{V}^C$ = shape C copies of $\mathcal{V}$ to $R^{N \times C \times T \times F}$
19: $\mathcal{L}^C$ = shape C copies of $\mathcal{L}$ to $R^{N \times C \times T \times F}$
20: $O_{CSA} = \mathcal{L}^C + \sigma \times (\mathbb{A}^C \cdot \mathcal{V}^C)$
21: **return** O_{CSA}

the training instances, the features in the hidden query space ($\mathcal{Q}$) and key space ($\mathcal{K}$) are learned (Lines 1-2). Then, these two feature tensors are transformed to class-specific query features $\mathcal{Q}^C$ and class-specific key $\mathcal{K}^C$ features (Lines 4-8). Next, $\mathcal{Q}^C$ are combined with $\mathcal{K}^C$ to calculate the class-specific attention $\mathbb{A}^C$ (Lines 9-16). Finally, the attention $\mathbb{A}^C$ is applied to the value features $\mathcal{V}^C$ (which is reshaped from the original feature tensor $\mathcal{L}$) to get O_{CSA} (Line 20).

In what follows, we provide details of the learning processes for calculating the class-specific attention $\mathbb{A}^C$.

3.3.1 Learning Process for $\mathbb{A}^C$. The calculation of the class-specific attention $\mathbb{A}^C$ utilizes the class-specific query features $\mathcal{Q}^C$ and the key features $\mathcal{K}^C$.

To calculate the similarity between $\mathcal{Q}^C$ and $\mathcal{K}^C$, a feature matrix is calculated as $\mathcal{S} = \mathcal{K}^C \cdot (\mathbb{Q}^C)^T$. Different from existing attention mechanisms, we introduce a class differentiation (CD) component to post-process the feature matrix $\mathcal{S}$, as shown as the right shaded area in Fig. 3. Existing attention mechanisms calculate the attention values directly from $\mathcal{S}$. However, calculating the similarity between the key and query features is not sufficient for CSA. If a feature is learned from instances in multiple (or all) classes, finding this feature may not be very helpful when making predictions. To better separate instances from one class, CSA needs to identify specific features from instances within this class, which are not commonly found from instances belonging to other classes. By adding the absolute differences between features for class c and all the other classes ($\neg c$) (Lines 11-15), we seek features that can differentiate instances from class c and instances from other classes ($\neg c$) while still making use of the original features. The significant class-specific features are kept in $\mathcal{S}^C$. $\mathcal{S}^C$ is further used to calculate the class-specific attention $\mathbb{A}^C$ (Line 16).

3.3.2 Updating of $\mathcal{L}$ to O_{CSA}. Class-specific attention $\mathbb{A}^C$ is used to adjust the value features $\mathcal{V}$ from $\mathcal{L}$ to get O_{CSA} (Lines 17-21).

The value features $\mathcal{V}$ and the features in $\mathcal{L}$ are transformed to $\mathcal{V}^C$ and $\mathcal{L}^C$ by repeat C copies. The output features in O_{CSA} are calculated using $O_{CSA} = \mathcal{L}^C + \sigma \times (\mathbb{A}^C \cdot \mathcal{V}^C)$ where σ is a learnable scalar value.

3.4 Utilization of CSA for the Output of NN

In the *CSA-embedded FCN* model (i.e., FCN model that embeds the CSA module, as shown in Figure 2), the features in O_{CSA} are adjusted using class-specific attention and are fed to output layers (global pooling layers and fully connected layers). More specifically, the output layers of the *CSA-embedded FCN* consist of one global pooling layer and one fully connected layer. The design of the global pooling layer follows the common design practice in existing models [10, 11, 17]: O_{CSA} ($\in R^{N \times C \times T \times F}$) is converted to a condensed feature tensor $\mathcal{G} \in R^{N \times C \times F}$ by averaging the features in the time dimension.

Where CSA differs from the conventional design is in the fully connected layer: the features in $\mathcal{G}$, if directly fed to a traditional fully connected layer, cannot make good use of the features from specific classes because fully-connected layers directly combine all features without considering their differences. Instead, we design a layer by introducing class-specific weights and biases to better utilize class-specific features. In particular, for each class c, a weight matrix $\Omega \in R^{F \times 1}$ and a bias β are used to calculate a value for this class ($val = \mathcal{G} \cdot \Omega + \beta$). This value is used to get the probability of an instance belonging to class c. Compared with the traditional fully-connected layer, this design can make better use of the class-specific features from O_{CSA} because features from different classes are not combined. Instead, the probability of an instance belonging to one class is calculated only using features specific to this class for most cases. Meanwhile, the weight parameters (in Ω and β) are not more than those in a traditional fully-connected layer – therefore, model training time does not increase because of this special design.

Prediction for a testing instance. The class-specific features O_{CSA} for a testing instance are calculated by directly utilizing the global class-specific attention $\mathbb{A}^C$ and the value features of this instance. Note that the testing instance does not need any class label to leverage class-specific attention. Next, these class-specific features O_{CSA} are passed to the specially designed fully connected layer described above to make class predictions.

4 EXPERIMENTS

We evaluate the effectiveness of the proposed CSA module. All methods are implemented using *Python* 3.7 and tested on a server with Intel Xeon Gold 5218 2.3G CPUs, 192GB RAM, and one Nvidia Tesla V100 GPU with 32GB memory. PyTorch 1.10 is used to build all the models. To acquire stable results, every number (accuracy or running time) we report is an *average of five* runs.

4.1 Experimental Settings

Datasets. We report results on 40 datasets (28 MTS and 12 UTS) which are randomly chosen from the time series repositories (UCR [3] and UEA [1]).

Dataset	N	C	V	T
ArtWordRec	575	25	9	144
BasicMotions	80	4	6	100
CharTraj	2858	20	3	182
Cricket	180	12	6	1197
DuckDuckGeese	100	5	270	1345
EigenWorms	259	5	6	17984
Epilepsy	275	4	3	206
EthanolConc	1751	4	3	1751
FaceDetection	9414	2	62	144
FingerMovements	416	2	28	50
HandMovement	234	4	10	400
Handwriting	1000	26	3	152
Heartbeat	409	2	61	405
InsectWingbeat	50000	10	30	200
JapaneseVowels	640	10	26	12
LSST	4925	14	6	36
Libras	360	15	2	45
MotorImagery	378	2	64	3000
NATOPS	360	6	24	51
PEMS-SF	440	7	963	144
PenDigits	10992	10	2	8
Phoneme	6668	39	11	217
RacketSports	303	4	6	30
SelfRegSCP1	561	2	6	896
SelfRegSCP2	380	2	7	1152
SpokenArab	8798	10	13	93
StandWalkJump	27	3	4	2500
UWaveGesture	440	8	3	315
MedicalImages	1141	10	1	99
MelPedestrian	3633	10	1	24
MidPhalanxoutGrp	554	3	1	80
MidPhalanxoutCor	891	2	1	80
MidPhalanxtw	553	6	1	80
OsuLeaf	442	6	1	427
PhalangesCor	2658	2	1	80
Powercons	360	2	1	144
ProximalPhaGrp	605	3	1	80
ProximalPhaCor	891	2	1	80
ProximalPhaTw	605	6	1	80
RefrigerationDev	750	3	1	720

Table 2: Statistics of 28 MTS and 12 UTS Datasets

Methods for Comparison. The CSA module is designed to be compatible with, and improve the time series classification performance of, existing models. Many NN models exist to classify time series, it is not realistic to embed CSA to all of them for testing. In our experiments, we embed the CSA module in five *representative* state-of-the-art models, namely (1) Fully Convolutional Networks (*FCN*) [17], (2) Multivariate Long Short-Term Memory (*MLSTM*) [9], (3) *MLSTM-FCN* [11], (4) CNN with attention (*CNN-ATN*) [6], and (5) *TapNet* [20]. The first three models are widely used NN models without attention mechanism, while *CNN-ATN* leverages attention in MTS classification, *TapNet* incorporates both attention and class label information in training. For these five models, we get their implementation from their published source code. If the version is not comparable, we convert the code to PyTorch.

Datasets	AI_{FCN}	AI_{MLSTM}	$AI_{MLSTM\text{-}FCN}$	AI_{TapNet}	$AI_{CNN\text{-}ATN}$
ArtWordRec	**0.204**	**4.902**	**1.029**	**0.816**	**0.407**
BasicMotions	-0.207	**0.948**	**0.207**	-0.600	**1.833**
CharTraj	0.000	**0.616**	**0.403**	0.000	0.000
Cricket	0.000	**0.568**	**2.050**	-1.895	**2.228**
DuckDuckGeese	**3.514**	**3.274**	**1.662**	-6.630	-
EigenWorms	**1.471**	0.000	**9.125**	-3.333	-
Epilepsy	**6.045**	**3.003**	**1.157**	-5.517	**0.421**
EthanolConc	**5.449**	**1.170**	**5.145**	**13.043**	-1.923
FaceDetection	0.000	**0.692**	**1.079**	**1.423**	-
FingerMovements	**1.235**	0.000	0.000	**3.583**	**2.990**
HandMovement	**4.274**	**11.892**	**5.579**	**12.069**	**4.739**
Handwriting	**1.408**	**7.692**	**4.965**	**40.404**	**0.676**
Heartbeat	**0.739**	**0.244**	**0.741**	-2.139	0.000
InsectWingbeat	**27.778**	0.000	**17.241**	-	-
JapaneseVowels	**0.907**	**1.493**	**0.215**	**0.405**	0.000
LSST	**2.703**	**0.379**	**5.283**	**42.667**	**1.506**
Libras	**0.443**	**28.261**	-0.442	**3.797**	0.000
MotorImagery	**1.250**	**2.632**	0.000	-3.459	-
NATOPS	**1.354**	**3.535**	**1.814**	0.000	**0.423**
PEMS-SF	**1.720**	**22.222**	**1.724**	0.000	-
PenDigits	-0.203	0.000	0.000	0.000	0.000
Phoneme	**4.545**	**6.000**	**18.750**	**31.707**	-
RacketSports	**3.476**	**1.542**	**4.416**	**15.909**	0.000
SelfRegSCP1	**0.229**	**0.668**	**0.452**	-13.587	**5.783**
SelfRegSCP2	**3.169**	**4.196**	0.000	**2.076**	**2.198**
SpokenArab	**0.821**	0.000	**0.407**	0.000	0.000
StandWalkJump	0.000	-5.780	-3.271	**15.748**	-
UWaveGesture	-0.962	**0.737**	-0.260	-0.907	-0.477
Wins	**21/28**	**22/28**	**21/28**	**13/27**	**11/20**
Average	**2.549**	**3.603**	**2.838**	**5.392**	**1.040**

Table 3: AI using CSA for all the models on MTS datasets (bold entries indicate scenarios where a model with CSA provides positive accuracy improvements)

Hyper-Parameter Setting. We use the hyper-parameter settings reported in [11], and [6]: The convolutional layers of the *FCN* model contain three 1-D kernels with sizes 8, 5, 3 – the corresponding numbers of kernels are 128, 256, and 128. The pooling filter in the global pooling layer has the same as the length as the time series output from the previous layer. The 2D kernels in the *CNN-ATN* model have sizes (8×1), (5×1), and (3×1). The number of the hidden states of *MLSTM* model is 128. The F parameter for $\mathcal{L}$ (Fig. 3) for all the models is set to 128. For TapNet, the number of variable subsets is set to 3.

Evaluation Measure. We consider a commonly used classification performance measure, accuracy ($Acc \in [0, 1]$). More specifically, we compute and report accuracy improvement, AI, to compare two models, A and B, with accuracy values Acc_A and Acc_B:

$$AI(A, B) = \frac{Acc_A - Acc_B}{Acc_B} \tag{1}$$

Since we are generally interested in measuring the improvement provided by the CSA module over the conventional baselines, we use AI_{Model} to denote $AI(Model\text{-}CSA, Model)$. Each experiment were run **five** times on each dataset and the average improvements for each dataset are reported.

4.2 Effectiveness Analysis

This section presents the effectiveness of the CSA module.

4.2.1 MTS datasets. Table 3 presents the detailed AI values for all MTS datasets. In this table, the results on several datasets for *CNN-ATN* model are missing (denoted with '-') because the models cannot finish utilizing the GPU memory. *CNN-ATN* uses 2D convolutional operations, as well as attentions over variable and temporal dimensions, which requires a lot of memory.

The table clearly show that the AI values is bigger than zero for most datasets for each base model. This means that the CSA module helps improve all these different base models. The average AI for the different models ranges from 1.04% to 5.39%. There are a few cases where the base models show slightly better performance than their corresponding CSA model; we conjecture that this is due to the increase in the number of model parameters when using CSA extension.

The accuracy improvement over the *TapNet* method is the highest. This is because *TapNet* generates the $\mathcal{L}$ tensor from three subsets of the input MTS data, which is three times larger than the $\mathcal{L} \in R^{N \times T \times F}$ from other models. The large $\mathcal{L}$ provides more information for our CSA module to utilize.

Datasets	AI_{FCN}	AI_{LSTM}	$AI_{LSTM\text{-}FCN}$	AI_{TapNet}	$AI_{CNN\text{-}ATN}$
MedicalImages	**0.253**	**11.111**	**0.509**	**0.267**	**1.044**
MelPedestrian	**0.909**	**8.434**	**0.251**	**3.788**	**1.412**
MidPhalanxoutGrp	**5.298**	**35.514**	-0.323	**0.608**	**2.096**
MidPhalanxoutCor	**0.246**	0.000	**0.247**	**0.474**	**0.242**
MidPhalanxtw	0.000	**20.961**	**3.200**	-1.294	**0.370**
OsuLeaf	**0.828**	0.000	**1.245**	**1.029**	**2.004**
PhalangesCor	**0.249**	0.000	**1.003**	**2.332**	**1.985**
Powercons	**0.432**	**1.277**	**0.648**	**1.695**	**1.089**
ProximalPhaGrp	**0.955**	**25.085**	**0.238**	-0.234	**0.235**
ProximalPhaCor	**2.128**	**4.348**	**3.118**	**0.671**	-1.089
ProximalPhaTw	-2.151	**12.262**	**3.261**	**0.756**	**1.222**
RefrigerationDev	-0.769	**13.725**	**1.515**	0.000	-2.405
Wins	**9/12**	**9/12**	**11/12**	**9/12**	**10/12**
Average	**0.644**	**11.251**	**1.175**	**0.776**	**0.631**

Table 4: AI using CSA for all the models on UTS datasets (*LSTM* model on UTS corresponds to *MLSTM* model for MTS; *LSTM-FCN* model on UTS corresponds to *MLSTM-FCN* model for MTS)

4.2.2 UTS datasets. Table 4 presents the AI values on UTS for all the models (the detailed accuracy results can be found from the appendix). The results show that the proposed CSA module can also improve the classification performance of all the base models on UTS datasets. The average AI for the different models ranges from 0.6% to 11% Compared with the results on the MTS datasets, the AI is slightly smaller on MTS datasets. This is because MTS datasets normally have more features than the UTS datasets.

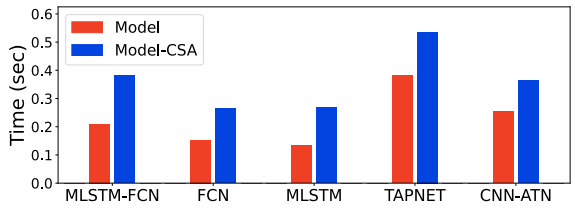
We want to particular mention that the improvement on the *TapNet* model is low on UTS. The reason is that there is only one variable in a UTS. Thus, we cannot create multiple variable subsets to generate $\mathcal{L}$. The best improvement is observed on *LSTM* model (which corresponds to the *MLSTM* model for MTS). This is because *LSTM* is the worst performing base model on UTS (detailed accuracy information can be found in the appendix).

4.3 Ablation Study

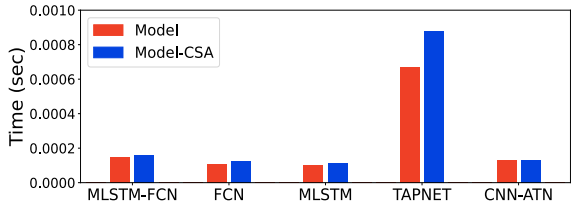
Datasets	*FCN-CSA*		
	w/o CD	*w CD*	*AI*
ArtWordRec	0.980	0.982	**0.204**
BasicMotions	0.962	0.966	**0.416**
CharTraj	0.990	0.990	0.000
Cricket	0.914	0.910	-0.438
DuckDuckGeese	0.742	0.766	**3.235**
EigenWorms	0.562	0.552	-1.780
Epilepsy	0.818	0.842	**2.997**
EthanolConc	0.658	0.658	0.000
FaceDetection	0.560	0.562	**0.357**
FingerMovements	0.653	0.656	**0.536**
HandMovement	0.475	0.488	**2.737**
Handwriting	0.280	0.288	**2.857**
Heartbeat	0.813	0.818	**0.677**
InsectWingbeat	0.134	0.138	**2.985**
JapaneseVowels	0.890	0.890	0.000
LSST	0.478	0.456	-4.50
Libras	0.896	0.906	**1.116**
MotorImagery	0.642	0.648	**0.935**
NATOPS	0.898	0.898	0.000
PEMS-SF	0.943	0.946	**0.371**
PenDigits	0.980	0.982	**0.204**
Phoneme	0.086	0.092	**6.977**
RacketSports	0.764	0.774	**1.309**
SelfRegSCP1	0.886	0.874	-1.354
SelfRegSCP2	0.573	0.586	**2.358**
SpokenArab	0.972	0.982	**1.029**
StandWalkJump	0.418	0.400	-4.192
UWaveGesture	0.612	0.618	**0.980**
Wins			**19/28**
Average			**0.715**

Table 5: AI comparison for *FCN-CSA* models with and w/o CD component on MTS datasets

CSA relies on a class-differentiation (CD) component to detect the features that can differentiate a class c and the remaining ones $\neg c$. In order to assess the the impact of the CD component, we implement a version of CSA without CD, *Model-CSA-NoCD* as follows. We calculate the CSA outputs by applying the *SoftMax* function directly on S to calculate $\mathcal{A}^C$, without calculating the feature differences between class c and classes $\neg c$ (i.e., without computing $\mathcal{S}_c^C$ and $\mathcal{S}^C$ in Fig. 3). Without loss of generality, we choose one base model (*FCN model*) and run *FCN-CSA* and *FCN-CSA-NoCD* over all the MTS datasets. As we expect, for most data sets, the CD component helps improve the performance. As shown in Table 5, on average, *FCN-CSA* increase the accuracy of *FCN-CSA-NoCD* with 0.72%. Despite that this number is not big, we note that among all the MTS datasets, *FCN-CSA* outperforms *FCN-CSA-NoCD* on 19 datasets and they have same performance on 4 datasets. These results show that the CD component helps improve the time series classification performance in general.



(a) Training time per iteration



(b) Testing time per instance

Figure 4: Running time (averaged on all the datasets)

4.4 Efficiency Analysis

The above results show that class-specific attention generally has positive impact on classification accuracy. In this section, we examine the efficiency of the proposed CSA module.

Fig. 4 (a) reports the training time for the baseline models (i.e., *Model*) and *Model-CSA* per iteration. As we would expect, CSA requires additional time to calculate the tensors and learning parameters in the CSA architecture. *Model-CSA* uses up to 0.2 seconds more than *Model* to train the models in one iteration. While class-specific attention has an impact on training times, there are no significant differences between the testing times of baseline models with models extended with CSA. Fig. 4 (b) shows *Model-CSA* uses almost the same time as *Model* to make predictions for one instance for all models except the TapNet. TapNet has a bigger difference because it has more features (three times) than other models to learn in the CSA module. This confirms that class-specific attention can be used effectively in practice for accurate time series classification.

5 CONCLUSIONS AND FUTURE WORKS

This paper presents a class-specific attention (CSA) module to improve the classification performance of neural network models for time series classification. The proposed module can be embedded to neural network model for time series classification (including those that leverage other forms of attention) to automatically capture significant features to differentiate instances of one class from the instances of the other classes. CSA identifies class-specific features leveraging training labels, while avoiding the need to access label information during testing phase. To the best of our knowledge, this is the first attention design that leverages the class label information in the hidden layers to generate class-specific features. Experiments on 40 real datasets have shown that CSA generally boosts accuracy – in the experiments, we have seen performance improvements up to 42%.

REFERENCES

[1] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. 2018. The UEA multivariate time series classification archive, 2018. arXiv:1811.00075 [cs.LG]

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*. http://arxiv.org/abs/1409.0473

[3] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. 2015. The UCR Time Series Classification Archive. www.cs.ucr.edu/~eamonn/time_series_data/.

[4] Xu Cheng, Peihua Han, Guoyuan Li, Shengyong Chen, and Houxiang Zhang. 2020. A Novel Channel and Temporal-Wise Attention in Convolutional Networks for Multivariate Time Series Classification. *IEEE Access* 8 (2020), 212247–212257. https://doi.org/10.1109/ACCESS.2020.3040515

[5] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR* abs/1409.1259 (2014). http://arxiv.org/abs/1409.1259

[6] Yifan Hao and Huiping Cao. 2020. A New Attention Mechanism to Classify Multivariate Time Series. In *IJCAI*. 1999–2005. https://doi.org/10.24963/ijcai.2020/277

[7] Yifan Hao, Huiping Cao, and Erick Draayer. 2020. CNN Approaches to Classify Multivariate Time Series Using Class-specific Features. In *2020 IEEE International Conference on Smart Data Services (SMDS)*. 1–8. https://doi.org/10.1109/SMDS49396.2020.00008

[8] Yifan Hao, Huiping Cao, Abdullah Mueen, and Sukumar Brahma. 2019. Identify Significant Phenomenon-Specific Variables for Multivariate Time Series. *IEEE Trans. Knowl. Data Eng.* 31 (2019), 1–13. https://doi.org/10.1109/TKDE.2019.2934464

[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[10] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. 2018. LSTM Fully Convolutional Networks for Time Series Classification. *IEEE Access* 6 (2018), 1662–1669. https://doi.org/10.1109/ACCESS.2017.2779939

[11] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. 2019. Multivariate LSTM-FCNs for time series classification. *Neural Networks* 116 (2019), 237–245. https://doi.org/10.1016/j.neunet.2019.04.014

[12] Yi-Ju Lu and Cheng-Te Li. 2020. AGSTN: Learning Attention-adjusted Graph Spatio-Temporal Networks for Short-term Urban Sensor Value Forecasting. In *ICDM*. IEEE, 1148–1153. https://doi.org/10.1109/ICDM50108.2020.00140

[13] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2014. How to Construct Deep Recurrent Neural Networks. In *2nd International Conference on Learning Representations (ICLR)*. http://arxiv.org/abs/1312.6026

[14] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. 2017. A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. In *IJCAI*. 2627–2633. https://doi.org/10.24963/ijcai.2017/366

[15] Achyut Mani Tripathi and Rashmi Dutta Baruah. 2020. Multivariate Time Series Classification With An Attention-Based Multivariate Convolutional Neural Network. In *2020 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN48605.2020.9206725

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPs*. 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need

[17] Zhiguang Wang, Weizhong Yan, and Tim Oates. 2017. Time series classification from scratch with deep neural networks: A strong baseline. In *International Joint Conference on Neural Networks (IJCNN)*. 1578–1585. https://doi.org/10.1109/IJCNN.2017.7966039

[18] Zhiwen Xiao, Xin Xu, Huanlai Xing, Shouxi Luo, Penglin Dai, and Dawei Zhan. 2020. RTFN: A Robust Temporal Feature Network for Time Series Classification. arXiv:2011.11829 [cs.LG]

[19] Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiaoli Li, and Shonali Krishnaswamy. 2015. Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition. In *IJCAI*. 3995–4001. http://ijcai.org/Abstract/15/561

[20] Xuchao Zhang, Yifeng Gao, Jessica Lin, and Chang-Tien Lu. 2020. TapNet: Multivariate Time Series Classification with Attentional Prototypical Network. In *The Thirty-Fourth Conference on Artificial Intelligence, AAAI 2020, February 7-12, 2020*. 6845–6852. https://aaai.org/ojs/index.php/AAAI/article/view/6165

[21] Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. 2020. Multivariate Time-series Anomaly Detection via Graph Attention Network. In *ICDM*. IEEE, 841–850. https://doi.org/10.1109/ICDM50108.2020.00093

[22] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. 2016. Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. *Frontiers Comput. Sci.* 10, 1 (2016), 96–112. https://doi.org/10.1007/s11704-015-4478-2

A DETAILED RESULTS

A.1 Univariate Time Series Results

Table 6 presents all the accuracy and AI results on 12 UTS datasets.

Datasets	*FCN*			*MLSTM*			*MLSTM-FCN*		
	w/o CSA	*w CSA*	*AI*	*w/o CSA*	*w CSA*	*AI*	*w/o CSA*	*w CSA*	*AI*
MedicalImages	0.792	0.794	**0.253**	0.522	0.580	**11.111**	0.786	0.790	**0.509**
MelPedestrian	0.660	0.666	**0.909**	0.830	0.900	**8.434**	0.796	0.798	**0.251**
MidPhalanxoutGrp	0.604	0.636	**5.298**	0.428	0.580	**35.514**	0.618	0.616	-0.323
MidPhalanxoutCor	0.814	0.816	**0.246**	0.570	0.570	0.000	0.810	0.812	**0.247**
MidPhalanxtw	0.516	0.516	0.000	0.458	0.554	**20.961**	0.500	0.516	**3.200**
OsuLeaf	0.966	0.974	**0.828**	0.210	0.210	0.000	0.964	0.976	**1.245**
PhalangesCor	0.804	0.806	**0.249**	0.600	0.600	0.000	0.798	0.806	**1.003**
Powercons	0.926	0.930	**0.432**	0.940	0.952	**1.277**	0.926	0.932	**0.648**
ProximalPhaGrp	0.838	0.846	**0.955**	0.590	0.738	**25.085**	0.840	0.842	**0.238**
ProximalPhaCor	0.846	0.864	**2.128**	0.690	0.720	**4.348**	0.834	0.860	**3.118**
ProximalPhaTw	0.744	0.728	-2.151	0.734	0.824	**12.262**	0.718	0.744	**3.261**
RefrigerationDev	0.520	0.516	-0.769	0.408	0.464	**13.725**	0.528	0.536	**1.515**

Datasets	*TAPNET*			*CNN-ATN*		
	w/o CSA	*w CSA*	*AI*	*w/o CSA*	*w CSA*	*AI*
MedicalImages	0.750	0.752	**0.267**	0.766	0.774	**1.044**
MelPedestrian	0.792	0.822	**3.788**	0.850	0.862	**1.412**
MidPhalanxoutGrp	0.658	0.662	**0.608**	0.668	0.682	**2.096**
MidPhalanxoutCor	0.844	0.848	**0.474**	0.826	0.828	**0.242**
MidPhalanxtw	0.618	0.610	-1.294	0.540	0.542	**0.370**
OsuLeaf	0.972	0.982	**1.029**	0.898	0.916	**2.004**
PhalangesCor	0.772	0.790	**2.332**	0.806	0.822	**1.985**
Powercons	0.944	0.960	**1.695**	0.918	0.928	**1.089**
ProximalPhaGrp	0.856	0.854	-0.234	0.850	0.852	**0.235**
ProximalPhaCor	0.894	0.900	**0.671**	0.918	0.908	-1.089
ProximalPhaTw	0.794	0.800	**0.756**	0.818	0.828	**1.222**
RefrigerationDev	0.584	0.584	0.000	0.582	0.568	-2.405

Table 6: AI comparison for models with and w/o CSA on UTS

A.2 Multivariate Time Series Results

Table 7 presents all the accuracy and AI results on 28 MTS datasets.

Datasets	*FCN*			*MLSTM*			*MLSTM-FCN*		
	w/o CSA	*w CSA*	*AI*	*w/o CSA*	*w CSA*	*AI*	*w/o CSA*	*w CSA*	*AI*
ArtWordRec	0.980	0.982	**0.204**	0.816	0.856	**4.902**	0.972	0.982	**1.029**
BasicMotions	0.968	0.966	-0.207	0.844	0.852	**0.948**	0.966	0.968	**0.207**
CharTraj	0.990	0.990	0.000	0.974	0.980	**0.616**	0.992	0.996	**0.403**
Cricket	0.910	0.910	0.000	0.704	0.708	**0.568**	0.878	0.896	**2.050**
DuckDuckGeese	0.740	0.766	**3.514**	0.672	0.694	**3.274**	0.722	0.734	**1.662**
EigenWorms	0.544	0.552	**1.471**	0.464	0.464	0.000	0.526	0.574	**9.125**
Epilepsy	0.794	0.842	**6.045**	0.666	0.686	**3.003**	0.864	0.874	**1.157**
EthanolConc	0.624	0.658	**5.449**	0.342	0.346	**1.170**	0.622	0.654	**5.145**
FaceDetection	0.562	0.562	0.000	0.578	0.582	**0.692**	0.556	0.562	**1.079**
FingerMovements	0.648	0.656	**1.235**	0.614	0.614	0.000	0.642	0.642	0.000
HandMovement	0.468	0.488	**4.274**	0.370	0.414	**11.892**	0.466	0.492	**5.579**
Handwriting	0.284	0.288	**1.408**	0.208	0.224	**7.692**	0.282	0.296	**4.965**
Heartbeat	0.812	0.818	**0.739**	0.818	0.820	**0.244**	0.810	0.816	**0.741**
InsectWingbeat	0.108	0.138	**27.778**	0.124	0.124	0.000	0.116	0.136	**17.241**
JapaneseVowels	0.882	0.890	**0.907**	0.938	0.952	**1.493**	0.932	0.934	**0.215**
LSST	0.444	0.456	**2.703**	0.528	0.530	**0.379**	0.530	0.558	**5.283**
Libras	0.902	0.906	**0.443**	0.552	0.708	**28.261**	0.906	0.902	-0.442
MotorImagery	0.640	0.648	**1.250**	0.608	0.624	**2.632**	0.638	0.638	0.000
NATOPS	0.886	0.898	**1.354**	0.792	0.820	**3.535**	0.882	0.898	**1.814**
PEMS-SF	0.930	0.946	**1.720**	0.576	0.704	**22.222**	0.928	0.944	**1.724**
PenDigits	0.984	0.982	-0.203	0.976	0.976	0.000	0.980	0.980	0.000
Phoneme	0.088	0.092	**4.545**	0.100	0.106	**6.000**	0.096	0.114	**18.750**
RacketSports	0.748	0.774	**3.476**	0.778	0.790	**1.542**	0.770	0.804	**4.416**
SelfRegSCP1	0.872	0.874	**0.229**	0.898	0.904	**0.668**	0.884	0.888	**0.452**
SelfRegSCP2	0.568	0.586	**3.169**	0.572	0.596	**4.196**	0.592	0.592	0.000
SpokenArab	0.974	0.982	**0.821**	0.960	0.960	0.000	0.982	0.986	**0.407**
StandWalkJump	0.400	0.400	0.000	0.692	0.652	-5.780	0.428	0.414	-3.271
UWaveGesture	0.624	0.618	-0.962	0.814	0.820	**0.737**	0.768	0.766	-0.260

Datasets	*TAPNET*			*CNN-ATN*		
	w/o CSA	*w CSA*	*AI*	*w/o CSA*	*w CSA*	*AI*
ArtWordRec	0.980	0.988	**0.816**	0.984	0.988	**0.407**
BasicMotions	1.000	0.994	-0.600	0.982	1.000	**1.833**
CharTraj	0.990	0.990	0.000	0.990	0.990	0.000
Cricket	0.950	0.932	-1.895	0.808	0.826	**2.228**
DuckDuckGeese	0.724	0.676	-6.630	-	-	-
EigenWorms	0.600	0.580	-3.333	-	-	-
Epilepsy	0.870	0.822	-5.517	0.950	0.954	**0.421**
EthanolConc	0.322	0.364	**13.043**	0.520	0.510	-1.923
FaceDetection	0.562	0.570	**1.423**	-	-	-
FingerMovements	0.614	0.636	**3.583**	0.602	0.620	**2.990**
HandMovement	0.464	0.520	**12.069**	0.422	0.442	**4.739**
Handwriting	0.198	0.278	**40.404**	0.296	0.298	**0.676**
Heartbeat	0.748	0.732	-2.139	0.820	0.820	0.000
InsectWingbeat	-	-	-	-	-	-
JapaneseVowels	0.988	0.992	**0.405**	0.990	0.990	0.000
LSST	0.450	0.642	**42.667**	0.664	0.674	**1.506**
Libras	0.790	0.820	**3.797**	0.790	0.790	0.000
MotorImagery	0.636	0.614	-3.459	-	-	-
NATOPS	0.890	0.890	0.000	0.946	0.950	**0.423**
PEMS-SF	0.912	0.912	0.000	-	-	-
PenDigits	0.940	0.940	0.000	0.970	0.970	0.000
Phoneme	0.164	0.216	**31.707**	-	-	-
RacketSports	0.704	0.816	**15.909**	0.888	0.888	0.000
SelfRegSCP1	0.736	0.636	-13.587	0.830	0.878	**5.783**
SelfRegSCP2	0.578	0.590	**2.076**	0.546	0.558	**2.198**
SpokenArab	0.982	0.982	0.000	0.990	0.990	0.000
StandWalkJump	0.508	0.588	**15.748**	-	-	-
UWaveGesture	0.882	0.874	-0.907	0.838	0.834	-0.477

Table 7: AI comparison for models with and w/o CSA on MTS