CNN Approaches to Classify Multivariate Time Series Using Class-specific Features

Yifan Hao
Department of Computer Science
New Mexico State University, NMSU
Las Cruces, US
yifan@nmsu.edu

Huiping Cao
Department of Computer Science
New Mexico State University, NMSU
Las Cruces, US
hcao@nmsu.edu

Erick Draayer
Department of Computer Science
New Mexico State University, NMSU
Las Cruces, US
edraayer@nmsu.edu

***Abstract*—Many smart data services (e.g., smart energy, smart homes) collect and utilize time series data (e.g., energy production and consumption, human body movement) to conduct data analysis. Among such analysis tasks, classification is a widely utilized technique to provide data-driven solutions.**

Most existing classification methods extract a single set of features from the data and use this feature set for classification across multiple classes. This often ignores the reality that different and class-specific subsets of the initial feature set may better facilitate classification. In this paper, we propose two convolutional neural network (CNN) models using class-specific variables to solve the multi-class classification problem over multivariate time series (MTS) data. A new loss function is introduced for training the CNN models. We compare our proposed methods with 13 baseline approaches using 14 real datasets. The extensive experimental results show that our new approaches can not only outperform other methods on classification accuracy, but also successfully identify important class-specific variables.

***Keywords*-Smart data-services; Multivariate Time Series; Classification**

I. Introduction

Smart data-services provide data-driven solutions to decision makers by utilizing different types of data. Much of such data is in the form of multivariate time series (MTS) and collected through smart infrastructures, such as smart energy, smart homes, and smart healthcare. For example, multiple current and frequency waveforms (MTS data) collected in smart energy systems can be used to identify system faults; the movement information of the different parts of a human (MTS data) collected by sensors can help identify the type of movement this person is making; the amount of the many PM2.5 composition particles in a period of time (MTS data) can help monitor air quality. Analyzing MTS data has received increasing interest in the past decade due to the deployment of smart infrastructures. Extracting features and classifying MTS are applied in various smart data-services, including human activity recognition [1], [2], healthcare [3], voice processing [4] and many others. In MTS classification problems, besides generating accurate predictions, it is also important to understand the features or factors that are most critical for prediction interpretation or decision making. For example, recent research shows that different PM2.5 (particulate matter with diameter less than 2.5 micrometers) composition particles contribute to different types of diabetes [5]; One PM2.5 particle can be considered as one variable in some MTS data. Identifying which PM2.5 particles contribute to specific diabetes (e.g., Type-2 diabetes) is a challenging problem in this domain and revealing this information would be a major benefit to researchers.

Compared with other data, MTS data contains time-dependency in each time series. How to model and analyze this time-dependency is one major challenge in MTS classification. Many algorithms can successfully classify MTS data, but very few of them can directly capture the relationships between classes and specific features.

Convolutional neural networks (CNN) have shown high accuracy in time-series classification [2], [6], [7], [8]. The CNN approach is capable of automatically extracting features from the MTS data and utilizing such features to recognize different classes. In this paper, we propose two newly designed convolutional neural network (CNN) based approaches by leveraging class-specific variables. These two approaches redesign the fully connected layer in a typical CNN structure to reveal and leverage the class-specific variables. Identifying these class-specific variables is very important, especially in neural network-based algorithms [9].

We show that our CNN approaches can not only achieve better classification performance than state-of-the-art methods but also successfully identify the class-specific variables, which gives us a better understanding of the classes in the data. The main contributions of this paper are the following.

- We propose two novel CNN based approaches with class-specific variables. The class-specific variables are identified and leveraged during model training.
- A new loss function is designed to fit the structure of the proposed CNN models. The new loss function can better separate similar classes and address the issue with imbalanced datasets.
- The proposed approaches can identify important class-specific variables. Compared with other state-of-the-art methods, our approaches can identify the class-specific variables more effectively and efficiently.
- We evaluate the classification performance of the proposed approaches with four state-of-the-art methods

978-1-7281-8777-8/20/$31.00 ©2020 IEEE
DOI 10.1109/SMDS49396.2020.00008

and nine other baselines. Our experiments on 14 datasets show that the proposed approaches achieve the best averaged accuracy.
- We compare the identified class-specific variables from our approach with the features identified from state-of-the-art methods. Our approach can identify similar class-specific variables with much better efficiency.

The paper is organized as follows. Section II formally defines the problem and related terminology. Section III presents our proposed approaches. Section IV presents our experiments and shows the effectiveness and efficiency of our proposed approaches. Section V discusses the literature. Finally, Section VI concludes our work.

II. Problem Formulation and Terminology

Multivariate time series (MTS) data records values for multiple variables in a period of time. (e.g., daily temperature and humidity over one month). Each MTS consists of multiple time series in the form of $(v_1, v_2, \cdots, v_m)$ where v_i is a numerical value recorded for one variable at the i-th time point and m is the length of the time series.

When conducting classifications, it is important to extract useful features from MTS data. Choosing an optimal selection of features can impact both accuracy and performance of the classification, and may reveal underlying associations between features and classes. For example, an association between certain medical readings and the classification of a patient's disease could improve understanding of how that disease develops. However, determining optimal features in MTS data can be very challenging. This paper works on MTS classification and class-specific variable (CV) identification simultaneously.

Definition 1 (Research Problem): Given an MTS with multiple class labels, the problem is to accurately classify instances of the MTS by extracting and utilizing class-specific variables.

Symbol	Meaning
C	# of distinct classes in a dataset
V	# of variables in an MTS dataset
N	# of instances in an MTS dataset
m	length of one time series in an MTS dataset
Y	the actual class labels of the instances

Table I: Symbols used in this paper

III. Methodology

This section presents our CNN based approaches to conduct the multi-class classification of MTS data. Table 1 summarizes the meaning of major parameters. We use a toy dataset throughout this paper to explain the concepts and the computations of our algorithms.

Example 1 (MTS data): Table II shows a small dataset with three classes: Standing in Elevator (SE), Moving around in Elevator (ME), and Playing Basketball (PB). This dataset contains two variables representing the height of the sensors on someone's left arm (LA) and left leg (LL). Fig. 1 draws the LA sequences of the three instances.

Class	Variables	Time sequences
Standing in Elevator (SE)	LA	10, 20, 29, 39, 40, 40, 40
	LL	5, 15, 25, 33, 35, 35, 35
Moving around in Elevator (ME)	LA	12, 18, 31, 37, 42, 38, 41
	LL	7, 14, 27, 30, 37, 34, 37
Playing Basketball (PB)	LA	10, 14, 18, 13, 9, 7, 10
	LL	4, 6, 9, 8, 7, 3, 5

*LA/LL: the y-coordinate of the left arm/leg sensor

Table II: Example dataset

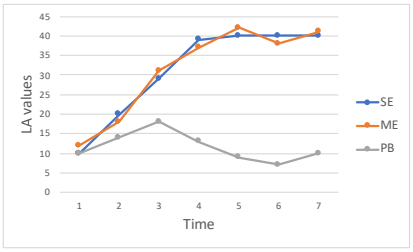

Figure 1: The LA sequences for the example dataset

A. Motivation of Proposed Approaches

In classification problems, when an instance is wrongly classified to a class label (say c), a common reason is that this instance's features are similar to the features of class c. We observe that while some instances in a dataset can be easily classified with the correct class labels (no matter which classifiers are used), other instances may be misclassified to classes with similar features. Let us use *fuzzy class* to denote the classes that can be easily confused with (and be classified as) other classes and use *non-fuzzy class* to denote the other classes.

Given the example, a CNN can benefit from class-specific variables that can differentiate the "SE" class and the "ME" class. One (or multiple) fully-connected layer(s) for all the classes cannot differentiate instances belonging to fuzzy classes. With such fuzzy-class instances, many iterations used to train a neural network model may not improve accuracy much. Instead, accuracy may become worse in later iterations. For example, in one iteration, many instances belonging to fuzzy classes are wrongly classified into a class with similar features. In the next iteration, the adaptations adjusted by the learning algorithm may help correctly classify instances in one fuzzy class, but may wrongly classify instances in another fuzzy class. A desirable model needs to identify features that help differentiate instances in the fuzzy classes. We design class-specific fully-connected components to alleviate this issue.

B. CNN with Class-Specific Fully-Connected Components

A CNN$_{mts}$ [2], [9] model for MTS consists of several convolutional layers, pooling layers, and fully-connected

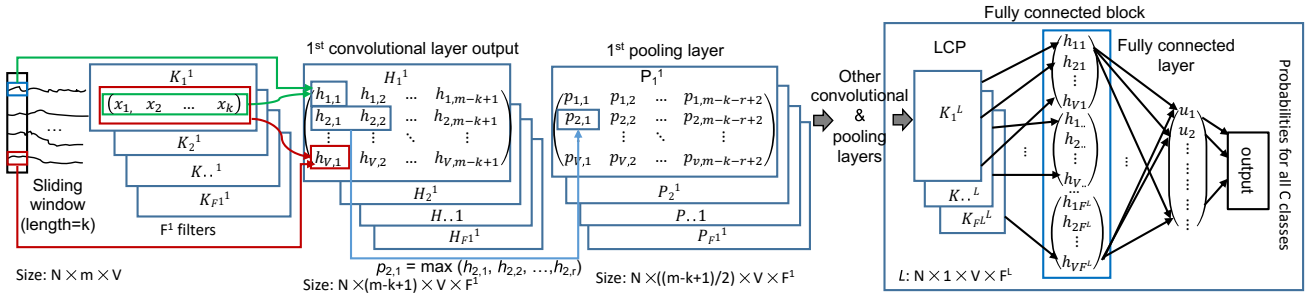

Figure 2: CNN structure for MTS using in [9] (LCP: last convolutional or pooling layer)

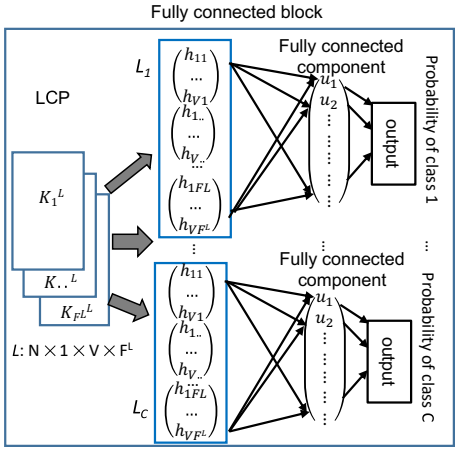

Figure 3: The FCC layer in CNN-CF

layers.

1) Overall Design of CNN-CF: The design of our CNN model with Class-specific Fully-connected components (*CNN-CF*) is built upon the CNN$_{mts}$ model [9] , as shown in Fig. 2. Given the MTS input with size $N \times m \times V$ (N instances where each instance keeps V sequences with length m), different from the regular CNN model for image classification, whose convolutional (pooling) filter size is $k \times k$ ($r \times r$), the convolutional (pooling) filter in CNN$_{mts}$ has size $k \times 1$ ($r \times 1$). The convolutional and pooling filters are applied to each time series variable. The last convolutional/pool (LCP) layer of CNN$_{mts}$ aggregates the time series output sequences from the previous convolutional layer. The output from the LCP is $N \times 1 \times V \times F^L$ (F^L is the number of filters in the LCP layer). When using a CNN$_{mts}$ model to conduct multi-class classification, the fully-connected layer (or the last fully-connected layer if there are multiple such layers) is utilized to calculate C probabilities for all C classes. The class with the highest probability is selected as the predicted class.

CNN-CF decomposes a large task (learning C probabilities for C classes) to smaller tasks with binary classifications. It creates C *fully-connected components (FCCs)* after the LCP layer. Each component conducts one binary classification task: differentiating one class from all the other classes. The FCC layer contains C fully-connected components. The c-th component is responsible for calculating the probability that one instance belongs to class c. Fig. 3 shows the structure of the FCC layer in our CNN-CF.

Algorithm: CNN-CF-Construction (X, Y)
Input: (1) X: the training multivariate time series data
(2) Y: the class labels for X
Output: a CNN-CF model

1) Set up the convolutional and pooling layers in the CNN$_{mts}$ model
2) Let $\mathcal{L}$ be the matrices of LCP, which are vectorized to $[(h_{11}, \cdots, h_{V1}), \cdots, (h_{1F^L}, \cdots, h_{VF^L})]$
3) Make C copies of $\mathcal{L}$: $\mathcal{L}_1, \cdots, \mathcal{L}_C$
4) Initialize an array M_{cf} to keep the weights for C fully-connected components
5) For each class label c ($c = 1 \cdots C$), construct a Y_c vector
   a) For the j-th instance in Y
      i) if ($Y[j] == c$) $Y_c[j] = 1$ else $Y_c[j] = 0$
   b) Use the weights in $M_{cf}[c]$ to connect $\mathcal{L}$ and Y_c
6) Train a constructed CNN model

Figure 4: Algorithm to construct a CNN-CF model

The pseudo-code of constructing a CNN-CF model is shown in Algorithm *CNN-CF-Construction* (Fig. 4). The convolutional and pooling layers are set as in CNN$_{mts}$ (Line 1). It flattens the matrices in LCP to vectors and makes C copies of the vectorized hidden units, $\mathcal{L}_1, \cdots, \mathcal{L}_C$. The c-th fully connected component links the $\mathcal{L}_c$ to the output layer of predicting an instance to be class c or not c.

2) Loss Function of CNN-CF: To train any neural network model, a loss function is needed to optimize the parameters (weights) of the neural network. The gradient descent optimization method trains the parameters (weights) of the neural network such that the loss function is minimized. Cross-Entropy (CE) is the most commonly used loss function. In MC problems, CE is formally defined as

$$CE = \sum_{i=1}^{N} \sum_{c=1}^{C} y_c[i] \times log(p_c[i])$$

Here, $y_c[i]$ is 1 if the i-th instance's actual class is c and is 0 otherwise, $p_c[i]$ is the probability of the i-th instance

belonging to class c. Directly applying CE as a loss function in CNN-CF generally produces high CE values because the CE calculation depends on the prediction of all the instances and the predictions of instances belonging to the fuzzy classes are generally poor.

Example 2: In the dataset shown in Example 1, instances from "SE" and "WE" are easily predicted to be one of the fuzzy classes in one iteration. The poor performance of "SE" and "WE" increases the overall CE value. A high CE value may cause a bad optimization of the parameters for "PB" in the next iteration.

The fully connected components that calculate C binary classifications need to address another issue where the instances belonging to a class c or $\neg c$ are imbalanced. The loss function needs to consider the imbalanced nature of the dataset. When a dataset has unbalanced instances, Weighted Cross-Entropy (WCE) is generally used. However, WCE can only alleviate the issue to some degree. This paper introduces a new measurement, Binary-class Cross-Entropy (BCE), to alleviate the effect of unbalanced data in binary classification problems. For each binary classification problem (the prediction is either c or $\neg c$), we obtain a set S_c with all the instances which either have c as the real label or are predicted to belong to c. BCE_c calculates the cross-entropy using only the instances from S_c, as shown in Eq. (1). N_c in Eq. (1) is the total number of instances in S_c and $\mathcal{L}_c$ is the flattened vector $\mathcal{L}$. The design of BCE_c can force the algorithm to learn better weights to separate class c from other classes which have similar features to class c.

$$BCE_c = \frac{1}{N_c}\sum_{i \in S_c} y_c[i]log(p_c[i]) + (1 - y_c[i])(1 - log(p_c[i])) \tag{1}$$

where $p_c[i] = \vec{\omega}_c \times \mathcal{L}_c$.

Example 3: For the dataset in Example 1, BCE for class "SE" very likely considers the instances from "SE" and "ME". The BCE for "SE" class focuses on separating "SE" and "ME" instances.

We further define a Class-combined Cross-Entropy (CCE) as the loss function in the training of the FCC layer with C fully-connected components. CCE adds up the C BCE values from all the binary classification problems (Equation 2).

$$CCE = \sum_{c=1}^{C} BCE_i \tag{2}$$

The derivative of CCE is the sum of the derivatives of the C BCEs based on the sum rule of differentiation operation [10].

$$\frac{\partial(CCE)}{\partial \omega_c} = \frac{\partial BCE_1}{\partial \omega_c} + ... + \frac{\partial BCE_C}{\partial \omega_c} = \frac{\partial BCE_c}{\partial \omega_c} \tag{3}$$

In the binary classification problem (c vs. $\neg c$), the weight and bias are trained using BCE_c. The weight and bias for class c are only depended on whether c can be separated over $\neg c$. Equation 3 shows the derivation calculation for ω_c.

C. CNN-CF with Class-Specific Features

Features from LCP do not equally contribute to classifying various classes. Furthermore, features that can help classify instances belonging to one class may not be helpful in classifying instances belonging to other classes [9]. It is critical that we utilize different features that are important in differentiating separate classes.

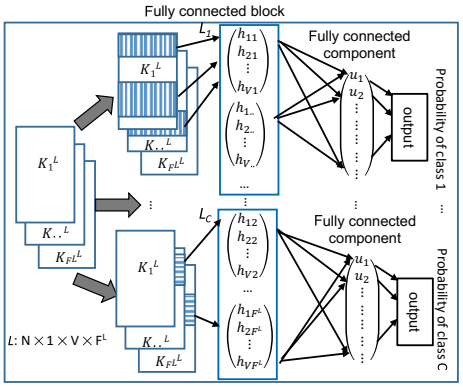

Figure 5: FCC layer of CNN-CF^2 (the shaded areas denote the important features (variables and filters)

We design a new CNN approach to incorporate class-specific variables in the model. This new approach is denoted as CNN-CF^2 (Fig. 5).

Building upon the *CNN-CF* model, it extracts important features from the LCP layer ($\mathcal{L}_1, \cdots, \mathcal{L}_C$) and feeds only the important features for class c to the corresponding c-th fully-connected component for prediction. The features are chosen from the LCP layer for two reasons. First, it is the last convolutional/pooling layer, thus its features are at higher-level and have more valuable information than the features in the previous layers. Second, the LCP hidden units absorb the time dimension through the previous layers, thus the time dependency does not need to be considered when choosing features in the LCP layer.

The top features are related to two factors, the variables, which are represented in the data, and the filters (or kernels), which are set in the *CNN-CF*^{2} model. Selecting the top features involves identifying (i) the top important variables for each filter without considering the specific classes (using Algorithm *CNN-CF*^{2}*-TVI* in Fig. 6) , (ii) the top filters for each class (using Algorithm *CNN-CF*^{2}*-TFI* in Fig. 7) , and (iii). the important variables for each class (Section III-D)

The first step is to identify the important variables for each filter (kernel) because one filter in the CNN model can capture significant features for some variables, but not for all the variables. For each filter f and each instance n, the algorithm (Fig. 6) retrieves the variables with high feature values because a high feature value indicates that the filter matches the shape of the sequence for that variable.

Algorithm: ***CNN-CF^2-TVI*** **($\mathcal{L}$, σ_V)**
Input: (1) $\mathcal{L}$: $N \times V \times F^L$ array from the last convolutional layer,
(3) σ_V the percentage of requested variables for each filter
Output: $\mathcal{L}'$: $N \times V \times F^L$ with the non-top-rated variables are zeros

1) Initialize an $N \times V \times F^L$ array $\mathcal{L}'$ with value zero
2) For each filter feature f (f=$1 \cdots F^L$)
   a) For each instance n (n=$1 \cdots N$)
      i) $\mathcal{L}_n^f$ = $\mathcal{L}[n, 1 \cdots V, f]$
      ii) top_n^f = the variable indies of the ($\sigma_V \times V$) highest values from $\mathcal{L}_n^f$
      iii) For each variable v in top_n^f
         A) $\mathcal{L}'[n, v, f] = \mathcal{L}[n, v, f]$
3) Return $\mathcal{L}'$

Figure 6: The top-variable identification algorithm

Algorithm: ***CNN-CF^2-TFI*** **($\mathcal{L}'$, Y, σ_F, C)**
Input: (1) $\mathcal{L}'$: $N \times V \times F^L$ output array from *CNN-CF^2-TVI*,
(2) Y: the class-label vector for N instances,
(3) σ_F the percentage of requested features for each class,
(4) C the total number of classes
Output: TF_{set} =$\{TF_1, TF_2, \cdots, TF_C\}$ where TF_c consists of $\lfloor \sigma_F \cdot F^L \rfloor$ top important filters for the class label c

1) Initialize an $C \times F^L$ array ω with score zero
2) For each class c ($c = 1, \cdots, C$)
   a) n_c = indexes of instances belonging to class c
   b) $n_{\neg c}$ = indexes of instances not belonging to class c
   c) For each filter f (f=$1 \cdots F^L$)
      i) $\mathcal{L}'^f_c$ = $\mathcal{L}'[\, n_c, 1 \cdots V, f]$
      ii) $\mathcal{L}'^f_{\neg c}$ = $\mathcal{L}'[n_{\neg c}, 1 \cdots V, f]$
      iii) *meanL$_c$*, *meanL$_{\neg c}$* = mean vector of $\mathcal{L}'^f_c$ and $\mathcal{L}'^f_{\neg c}$ by averaging on the first dimension respectively
      iv) $\omega[c, f]$ = dist(*meanL$_c$*, *meanL$_{\neg c}$*))
3) For each class c
   a) TF_c = the top $\lfloor \sigma_F \cdot F^L \rfloor$ highest values in $\omega[c, 1 \cdots F^L]$
4) Return TF_{set} =$\{TF_1, TF_2, \cdots, TF_C\}$

Figure 7: The top-filter identification algorithm

After selecting the important variables for all the filters, the second step is to choose important filters because not all the filters can generate significant features (Algorithm *CNN-CF^2-TFI* in Fig. 7). The importance of a filter is evaluated using the top variables chosen for that filter by the *CNN-CF^2-TVI* algorithm. *CNN-CF^2-TFI* first separates the training data into two groups, where the instances in one group belong to class c while the instances in the other group do not belong to c (Lines 2a-2b). Then, it calculates the distances of features from the c-class instances and the $\neg c$-class instances (Line 2c). The filters with larger feature distances between the c-class instances and the $\neg c$-class instances are considered more important because they can better differentiate class c and $\neg c$ (Line 3). Different from CNN-CF, CNN-CF^2 only feeds the important feature identified by *CNN-CF^2-TFI* to each fully connected component in model training.

D. Class-Specific Variable Identification

Besides improving the classification accuracy of MC problems on MTS data, we are also interested in reporting the features that are important to differentiate a specific class c from the other classes. We call this step Class-specific variable identification (*CVI*). The class-specific feature (CF) selection is conducted after *CNN-CF^2-TFI*. Please note that the significant variables selected at this step are class specific while the important variables selected through *CNN-CF^2-TVI* are for each filter, not class specific.

For each class c, we get n_c and $n_{\neg c}$. They denote the indexes of the instances belonging to c and not belonging to c respectively. For each class c and each variable v, we extract the features for all the variables from the top filters $\mathcal{L}[n_c, v, TF_c]$ and $\mathcal{L}[n_{\neg c}, v, TF_c]$, and calculate their mean vectors, $meanL_c$ and $meanL_{\neg c}$ (whose length is $|TF_c|$) by averaging on the first dimension. The score of a variable v is the euclidean distance between $meanL_c$ and $meanL_{\neg c}$. The higher this score is, the higher the differentiation function that v can play. The variables with the high score values can better differentiate class c from other classes.

Dataset	N	C	V	m
Action	560	20	570	100
Activity	320	16	570	337
Ara Voice	8800	88	39	91
Auslan	2565	95	22	96
Daliy Sport	9120	19	45	125
Ges	396	5	18	214
Har	10299	6	9	128
Ht Sensor	100	3	11	5396
JapaneseVowels	640	9	12	26
OHC	2858	20	30	173
Net	1337	2	4	994
Eeg	128	2	13	117
Eeg2	1200	2	64	256
Ozone	346	2	72	291

Table III: Datasets

IV. Experiments

All the methods are implemented using $Python$ 3.4, and tested on a server with Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz and 256 GB RAM. TensorFlow (www.tensorflow.org [11]) is used to build the CNN model and Adamoptimizer is used in the training process.

A. Methods for Comparison

The performance of the proposed approaches are compared with existing state-of-art methods: (i) Long Short Term Memory Fully Convolutional Networks (LSTM-FCN) [6] and (ii) an Attention LSTM-FCN (ALSTM-FCN) [6], which are defined for classifying univariate time series and have been adapted for MTS, (iii) Multivariate LSTM-FCN (MLSTM-FCN) [7], and (iv) Multivariate Attention LSTM-FCN (AMLSTM-FCN) [7]. Other than the previous four approaches, more Other Baseline (OB), WMUSE [12], ARKernel [13], LPS [14], mv-ARF [13], SMTS [15], HULM [16], DTW [1], SVM [17], and RF [18] are used in the comparison. Only the best results from the baseline approaches are presented in this section.

B. Experimental settings

(1) Datasets: We use 14 real datasets, which have at least 100 instances, to test the proposed approaches. The

Dataset	Methods							
	LSTM-FCN	MLSTM-FCN	ALSTM-FCN	MALSTM-FCN	Best-of-OB	CNN$_{mts}$	CNN-CF	CNN-CF^2
Action	0.717	0.754	0.727	0.747	0.707	0.747	0.798 [1]	0.795 [2]
Activity	0.531	0.619	0.556	0.588	0.663	0.581	0.606 [4]	0.613 [3]
Ara Voice	0.980	0.980	0.986	0.983	0.946	0.961	0.972 [5]	0.973 [4]
Auslan	0.970	0.970	0.960	0.960	0.980	0.947	0.970 [3]	0.972 [2]
Daily Sport	0.997	0.997	0.997	0.997	0.984	0.993	0.995 [3]	0.996 [2]
Ges	0.505	0.535	0.525	0.531	0.409	0.535	0.545 [2]	0.556 [1]
Har	0.960	0.967	0.955	0.967	0.816	0.946	0.960 [3]	0.961 [2]
HT Sensor	0.680	0.780	0.720	0.800	0.720	0.760	0.860 [1]	0.860 [1]
JapaneseVowels	0.990	1.000	0.990	0.990	0.980	0.980	0.990 [2]	1.000 [1]
OHC	1.000	1.000	1.000	1.000	0.990	0.990	0.999 [2]	1.000 [1]
Averaged	0.833 [7]	0.860 [3]	0.842 [6]	0.856 [4]	0.794 [8]	0.843 [5]	0.870 [2]	**0.872 [1]**

* Values in [] denote the ranks in each row

* To better display the results, we only show the ranks of the results from CNN-CF and CNN-CF^2 for each individual dataset.

Table IV: Multi-class classification (Best-of-OB: the best results from all the other baseline approaches)

Dataset	Methods							
	LSTM-FCN	MLSTM-FCN	ALSTM-FCN	MALSTM-FCN	Best-of-OB	CNN$_{mts}$	CNN-CF	CNN-CF^2
Eeg	0.609	0.656	0.641	0.641	0.625	0.578	0.584 [7]	0.615 [5]
Eeg2	0.907	0.910	0.907	0.913	0.775	0.941	0.972 [1]	0.967 [2]
Net	0.940	0.950	0.930	0.950	0.980	0.947	0.963 [3]	0.968 [2]
Ozone	0.676	0.815	0.792	0.798	0.751	0.791	0.815 [1]	0.815 [1]
Averaged	0.783 [7]	0.833 [3]	0.818 [5]	0.826 [4]	0.783 [7]	0.814 [6]	0.834 [2]	**0.841 [1]**

* Values in [] denote the ranks in each row

* To better display the results, we only show the ranks of the results from CNN-CF and CNN-CF^2 for each individual dataset.

Table V: Binary-class classification comparison

detailed statistics for the datasets are shown in Table III. **(2) Evaluation measurements:** The most commonly utilized classification metric, accuracy, is reported as there are no imbalanced datasets. **(3) Parameter setting:** The convolutional layers of the CNN model contain three layers with filter sizes $8*1$, $5*1$, and $3*1$, the corresponding numbers of filters for the three layers are 128, 256, and 128. The pooling filter in the global pooling layer is set to be the same as the length of the time series output from the previous convolutional layer. These settings in the convolutional and pooling layers are the same as those in [7]. Both the percentage for variables (σ_V) and features (σ_F) are set to be 50%.

C. Effectiveness Analysis

In this section, we evaluate the performance of the proposed approaches in two aspects: classification performance and the meaningfulness of the selected features.

1) Comparison on Multi-Class Classification: This section compares the classification accuracy on 10 multi-class MTS datasets of our proposed methods, CNN-CF and CNN-CF^2, with other state-of-the-art approaches. Table IV presents the accuracy values on classifying those multi-class datasets.

The ranks for the two proposed approaches and the average accuracy for all methods are provided. For all 10 MC datasets, CNN-CF^2 achieves the highest average accuracy (the last row in Table IV) and the best ranking. CNN-CF^2 gets the highest accuracy on four datasets and receives the second highest on another four datasets. CNN-CF^2 has the lowest rank on the "Ara Voice" dataset. However, compared with the highest accuracy from MLSTM-FCN (0.986), the accuracy result from CNN-CF^2 (0.973) is still comparable.

The results in the last two columns show that the CNN-CF^2 achieves better or the same performance as CNN-CF on most datasets (nine out of ten datasets). Both CNN-CF and CNN-CF^2 have better performance than the CNN$_{mts}$ with the same configuration. The results are consistent with our expectations and intuition.

2) Comparison on Binary-Class Classification: The proposed approaches are designed for the multi-class classification problem (MC). However, they can also be applied to conduct Binary classification (BC). We evaluate the performance of the proposed approaches on four datasets with two class labels. Table V shows the comparison results using all methods. On those four datasets, the CNN-CF and CNN-CF^2 still outperformed in terms of averaged accuracy although the improvement is less than that in MC classification.

3) Effect of the Loss Function: This section compares the performance of using the newly designed loss function, class-combined cross-entropy (CCE), with the commonly used weighted cross-entropy (WCE) for multi-class classifications. Due to space limitation, we report results only on three datasets, "Ht sensor" with the minimum number of classes (3 classes), "Auslan" with the maximum number of classes (95 classes), and "Daily Sport" with the median number of classes (19 classes).

Loss function	Datasets		
	Ht Sensor	Daily Sport	Auslan
WCE	0.800	0.991	0.948
CCE	**0.860**	**0.995**	**0.970**

Table VI: CNN-CF accuracy (different loss functions)

Table VI shows that the classification accuracy of CNN-CF using WCE and CCE as the loss function respectively.

CNN-CF is used as the classifier because the performance of CNN-CF^2 is closely related to the performance of CNN-CF. The results indicate that CNN-CF using CCE outperforms the CNN-CF using WCE on all three datasets.

4) Case Studies of the Extracted Features: In this section, we verify the usefulness of the extracted class-specific variables (CV) that are identified using the *CVI* algorithm (Section III-D). We compare the identified CV with the ones discovered using the state-of-the-art approach [19].

PB	CV_{ref}	CV_{cvi}
Top 1	y gyroscopes (left arm)	x gyroscopes (left arm)
Top 2	x gyroscopes (left arm)	y gyroscopes (right arm)
Top 3	y gyroscopes (right arm)	y gyroscopes (left arm)
Top 4	x gyroscopes (right arm)	x accelerometers (left arm)
Top 5	y accelerometers (left arm)	x accelerometers (right arm)
Top 6	y accelerometers (right arm)	x accelerometers (left leg)

(a) Playing Basketball

RM	CV_{ref}	CV_{cvi}
Top 1	x magnetometers (left leg)	x accelerometers (left arm)
Top 2	x magnetometers (right leg)	x accelerometers (right arm)
Top 3	x magnetometers (torso)	x magnetometers (left leg)
Top 4	x accelerometers (left arm)	x accelerometers (right leg)
Top 5	x accelerometers (right arm)	x accelerometers (left leg)
Top 6	x accelerometers (left leg)	x magnetometers (right leg)

(b) Using rowing machine

Table VII: Top 6 CV comparison

In the "playing basketball" activity, both arms are used. The second column in Table VII(a) (CV_{ref}) shows the identified CV from [19] and the last column (CV_{cvi}) shows the selected CV using *CVI*, from this work. Similar to CV_{ref}, the top-5 CV from CV_{cvi} are arm-related, although the order of the CV is sightly different from CV_{ref}.

Table VII (b) shows the results for the activity "using rowing machine", which is an activity using movements of both arms and legs. Both CV of CV_{ref} and CV_{cvi} are from accelerometers and magnetometers recording arm and leg movements. The results show that the algorithm *CVI* in this paper can identify similar CV as those identified in [19].

Method	Ht Sensor	Daily Sport	Auslan
CNN$_{mts}$	63982	631044	595878
CNN-CF^2	24160	52877	8235

Table VIII: CV identification time (sec.)

Method	Data					
	Ht Sensor		Daily Sport		Auslan	
	Train	Test	Train	Test	Train	Test
CNN$_{mts}$	23917.2	6.5	97040.7	59.1	11577.2	6.1
CNN-CF	24145.2	6.5	105174.7	59.2	15890.3	9.7
CNN-CF^2	24162.4	6.6	112254.9	62.6	21707.2	14.9

Table IX: Classification time (sec.)

D. Efficiency Analysis

For class-specific variable identification, we compared CNN$_{mts}$ [9] and our *CNN-CF^2* because other approaches cannot detect class-specific variables. Table VIII presents the identification time, which shows that *CNN-CF^2* is much more efficient than CNN$_{mts}$. This is because CNN$_{mts}$ requires training C models in order to identify the CV for all C classes, while *CNN-CF^2* only needs to train one model for all C classes. The class-specific variable identification time of *CNN-CF^2* is about ρ ($2 \leq \rho \leq C$) times faster than that of CNN$_{mts}$. Here, ρ is smaller than C. It is because there are C' models ($0 \leq C' \leq C$) in [9] which can converge earlier instead of reaching the epoch limitation.

Table IX shows the classification time results. CNN-CF needs more training time than CNN$_{mts}$. This is because there are more parameters in the FCC layer than a regular fully-connected layer in CNN$_{mts}$. CNN-CF takes more time on a dataset with more classes. The time to train and test a CNN-CF^2 model are the longest among all the three approaches. CNN-CF^2 contains post-processing (*CNN-CF^2-TVI* and *CNN-CF^2-TFI*). However, compared with the overall training and testing time of a CNN model, the increment of the training and testing time is not significant, especially when the number of classes of one dataset is small. For the Ht Sensor dataset, the training time of CNN-CF^2 is 101% of the training time of CNN$_{mts}$. For "Auslan" with 94 classes, the CNN-CF^2 training time is about 187% of the training time of CNN$_{mts}$, which is also acceptable.

V. Related Works

We briefly introduce the existing algorithms for MTS classification. These algorithms can be separated as either similarity-based methods, feature-based methods, or neural network-based methods.

Similarity-based methods calculate similarity scores between time series and use these scores for classification. Previous work has shown Dynamic Time Warping (DTW) along with K-Nearest Neighbors (KNN) to be successful in MTS classification problems [1]. Feature-based methods typically use features extracted from the MTS data for comparisons and classifications. Examples of such methods are many (e.g., [15], [14], [16], [20], [12], [13], [21]). However, none of these methods use class-specific features in the classification. Many recent success stories of MTS classification comes from neural network-based approaches [2], [8]. However, these methods suffer from the vanishing gradient problem. Long Short-Term Memory (LSTM) RNNs help counter this problem by using gating functions in their state dynamics [22]. LSTM RNNs is able to learn the temporal dependencies in MTS unless such dependencies are long-term. Attention mechanisms and Squeeze-and-Excitation Blocks [23] have recently been adapted for Neural Network classification on MTS and generate multiple models including LSTM-FC, ALSTM-FCN [6] and MLSTM-FCN and MALSTM-FCN [7].

Two recent methods use class-specific features when classifying MTS data. MASK [24] identifies shapelets from

MTS and uses the shapelets to evaluate the variables. Similar to this paper, [9] identify the class-specific variables based on the CNN features. This paper compares our new methods with [9], which was compared with [24].

VI. Conclusions

This paper presents two CNN models, *CNN-CF* and *CNN-CF*^{2}, to classify multi-class multivariate time series by utilizing class-specific variables. These two models present two new designs to replace the typical fully connected layers in CNN. To train these two models, a new loss function is introduced to classify instances in similar classes and to alleviate the effect of imbalanced datasets. The experiments on 14 real datasets show that the proposed approaches not only improve the overall accuracy of multi-class classification but also efficiently identify class-specific variables. The design of the new loss function helps maintain reasonable training time of the proposed models. The proposed methodology can be directly utilized in any smart data-services that make use of MTS data to make classifications and to extract features.

Acknowledgments

This work is supported by NSF awards #1633330, #1345232, #1757207, and #1914635.

References

[1] S. Seto, W. Zhang, and Y. Zhou, "Multivariate time series classification using dynamic time warping template selection for human activity recognition," in *IEEE Symposium Series on Computational Intelligence, SSCI*, 2015, pp. 1399–1406.

[2] J. Yang, M. N. Nguyen, P. P. San, X. Li, and S. Krishnaswamy, "Deep convolutional neural networks on multichannel time series for human activity recognition," in *Intl. Joint Conf. on Artificial Intelligence, IJCAI*, 2015, pp. 3995–4001.

[3] P. Ordóñez, M. desJardins, C. Feltes, C. U. Lehmann, and J. C. Fackler, "Visualizing multivariate time series data to detect specific medical conditions," in *AMIA 2008, American Medical Informatics Association Annual Symposium*, 2008.

[4] S. Fong, K. Lan, and R. Wong, "Classifying human voices by using hybrid sfx time-series preprocessing and ensemble feature selection," *BioMed research international*, vol. 2013, p. 720834, 10 2013.

[5] F. L. Yang and et al, "Long-term exposure to ambient fine particulate matter and incidence of diabetes in china: A cohort study," *Environment International*, vol. 126, pp. 568–575, 05 2019.

[6] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "LSTM fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2018.

[7] F. Karim, S. Majumdar, H. Darabi, and S. Harford, "Multivariate lstm-fcns for time series classification," *Neural Networks*, vol. 116, pp. 237–245, 2019.

[8] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *15th Web-Age Information Management, WAIM*, 2014, pp. 298–310.

[9] Y. Hao, H. Cao, A. Mueen, and S. Brahma, "Identify significant phenomenon-specific variables for multivariate time series," *IEEE Trans. Knowl. Data Eng.*, vol. 31, pp. 1–13, 2019.

[10] G. Strang, *Calculus*, ser. Open Textbook Library. Wellesley-Cambridge Press, 1991, no. v. 1.

[11] "Tensorflow, an open-source software library for machine intelligence. https://www.tensorflow.org/."

[12] P. Schäfer and U. Leser, "Multivariate time series classification with WEASEL+MUSE," *CoRR, abs/1711.11343*.

[13] K. S. Tuncel and M. G. Baydogan, "Autoregressive forests for multivariate time series modeling," *Pattern Recognition*, vol. 73, pp. 202–215, 2018.

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

[15] M. G. Baydogan and G. Runger, "Learning a symbolic representation for multivariate time series classification," *Data Min. Knowl. Discov.*, vol. 29, no. 2, pp. 400–422, 2015.

[16] W. Pei, H. Dibeklioglu, D. M. J. Tax, and L. van der Maaten, "Multivariate time-series classification using the hidden-unit logistic model," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 29, no. 4, pp. 920–931, 2018.

[17] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT. ACM, 1992, pp. 144–152.

[18] Tin Kam Ho, "Random decision forests," in *Proceedings of 3rd International Conf on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.

[19] Y. Hao, H. Cao, A. Mueen, and S. Brahma, "Technical report for phenomenon-specific variable identification from multivariate time series. https://kddlab.nmsu.edu/project/PVI."

[20] A. Quattoni, S. Wang, L. Morency, M. Collins, and T. Darrell, "Hidden conditional random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 10, pp. 1848–1852, Oct 2007.

[21] M. Wistuba, J. Grabocka, and L. Schmidt-Thieme, "Ultra-fast shapelets for time series classification," *CoRR*, vol. abs/1503.05018, 2015.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *CoRR*, vol. abs/1709.01507, 2017.

[24] D. S. Raychaudhuri, J. Grabocka, and L. Schmidt-Thieme, "Channel masking for multivariate time series shapelets," *CoRR*, vol. abs/1711.00812, 2017.