Time series classification using multi-channels deep convolutional neural networks

Article · January 2014

CITATIONS
218

READS
731

5 authors, including:

Enhong Chen
University of Science and Technology of China
413 PUBLICATIONS **6,229** CITATIONS

SEE PROFILE

Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks

Yi Zheng[1,2], Qi Liu[1], Enhong Chen[1], Yong Ge[3], and J. Leon Zhao[2]

[1] University of Science and Technology of China
xiaoe@mail.ustc.edu.cn, {qiliuql,cheneh}@ustc.edu.cn
[2] City University of Hong Kong, Hong Kong
jlzhao@cityu.edu.hk
[3] The University of North Carolina at Charlotte
yong.ge@uncc.edu

Abstract. Time series (particularly multivariate) classification has drawn a lot of attention in the literature because of its broad applications for different domains, such as health informatics and bioinformatics. Thus, many algorithms have been developed for this task. Among them, nearest neighbor classification (particularly *1*-NN) combined with Dynamic Time Warping (DTW) achieves the state of the art performance. However, when data set grows larger, the time consumption of *1*-NN with DTW grows linearly. Compared to *1*-NN with DTW, the traditional feature-based classification methods are usually more efficient but less effective since their performance is usually dependent on the quality of hand-crafted features. To that end, in this paper, we explore the feature learning techniques to improve the performance of traditional feature-based approaches. Specifically, we propose a novel deep learning framework for multivariate time series classification. We conduct two groups of experiments on real-world data sets from different application domains. The final results show that our model is not only more efficient than the state of the art but also competitive in accuracy. It also demonstrates that feature learning is worth to investigate for time series classification.

1 Introduction

As a large amount of time series data have been collected in many domains such as finance and bioinformatics, time series data mining has drawn a lot of attention in the literature. Particularly, multivariate time series classification is becoming very important in a broad range of real-world applications, such as health care and activity recognition [1–3].

In recent years, a plenty of classification algorithms for time series data have been developed. Among these classification methods, the distance-based method k-Nearest Neighbor (k-NN) classification has been empirically proven to be very difficult to beat [4, 5]. Also, more and more evidences have shown that the Dynamic Time Warping (DTW) is the best sequence distance measurement in most domains [4–7]. Thus, the simple combination of k-NN and DTW could

F. Li et al. (Eds.): WAIM 2014, LNCS 8485, pp. 298–310, 2014.

reach the best performance of classification in most domains [6]. Other than sequence distance based methods, feature-based classification methods [8] follow the traditional classification framework. As is known to all, the performance of traditional feature-based methods depends on the quality of hand-crafted features. However, unlike other applications, it is difficult to design good features to capture intrinsic properties embedded in various time series data. Therefore, the accuracy of feature-based methods is usually worse than that of sequence distance based ones, particularly *1*-NN with DTW method. On the other hand, although many research works use *1*-NN and DTW, both of them cause too much computation for many real-world applications [7].

Motivation. *Is it possible to improve the accuracy of feature-based methods? So that the feature-based methods are not only superior to 1-NN with DTW in efficiency but also competitive to it in accuracy.*

Inspired by the deep feature learning for image classification [9–11], in this paper, we explore a deep learning framework for multivariate time series classification. Deep learning does not need any hand-crafted features by people, instead it can learn a hierarchical feature representation from raw data automatically. Specifically, we propose an effective Multi-Channels Deep Convolution Neural Networks (MC-DCNN) model, each channel of which takes a single dimension of multivariate time series as input and learns features individually. Then the MC-DCNN model combines the learnt features of each channel and feeds them into a Multilayer Perceptron (MLP) to perform classification finally. To estimate the parameters, we utilize the gradient-based method to train our MC-DCNN model. We evaluate the performance of our MC-DCNN model on two real-world data sets. The experimental results on both data sets show that our MC-DCNN model outperforms the baseline methods with significant margins and has a good generalization, especially for weakly labeled data.

The rest of the paper is organized as follows. Section 2 depicts the definitions and notations used in the paper. In section 3, we present the architecture of MC-DCNN, and describe how to train the neural networks. In section 4, we conduct experiments on two real-world data sets and evaluate the performance of each model. We make a short review of related work in section 5. Finally, we conclude the paper and discuss future work in section 6.

2 Definitions and Notations

In this section, we introduce the definitions and notations used in the paper.

Definition 1 *Univariate time series is a sequence of data points, measured typically at successive points in time spaced at uniform time intervals. A univariate time series can be denoted as* $\mathbf{T} = \{t_1, t_2, ..., t_n\}$*, and* n *is the length of* $\mathbf{T}$*.*

Definition 2 *Multivariate time series is a set of time series with the same timestamps. For a multivariate time series* $\mathbf{M}$*, each element* $\mathbf{m}_i$ *is a univariate time series. At any timestamp* t*,* $\mathbf{m}_{.t} = \{m_{1t}, m_{2t}, ..., m_{lt}\}$*, where* l *is the number of univariate time series in* $\mathbf{M}$*.*

As previous works shown [12], it's common to extract subsequences from long time series to do classification instead of classifying with the whole sequence.

Definition 3 *Subsequence is a sequence of consecutive points which are extracted from time series* $\mathbf{T}$ *and can be denoted as* $\mathbf{S} = \{t_i, t_{i+1}, ..., t_{i+k-1}\}$*, where* k *is the length of subsequence. Similarly, multivariate subsequence can be denoted as* $\mathbf{Y} = \{\mathbf{m}_{\cdot i}, \mathbf{m}_{\cdot i+1}, ..., \mathbf{m}_{\cdot i+k-1}\}$*, where* $\mathbf{m}_{\cdot i}$ *is defined in Definition 2.*

Since we perform classification on multivariate subsequences in our work, in remainder of the paper, we use subsequence standing for both univariate and multivariate subsequence for short according to the context. For a long-term time series, domain experts may manually label and align subsequences based on experience. We define this type of data as *well aligned and labeled data*.

Definition 4 *Well aligned and labeled data: Subsequences are labeled by domain experts, and different subsequences belonging to same pattern are well aligned.*

Fig.1 shows a snippet of time series extracted from BIDMC Congestive Heart Failure data set [13]. Each subsequence is extracted and labeled according to the red dotted line by medical staffs. However, to acquire the well aligned and labeled data, it always needs great manual cost.

Fig. 1. A snippet of time series which contains two types of heartbeat: normal (N) and ventricular fibrillation (V)

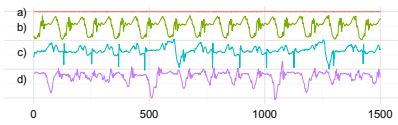

Fig. 2. Four 1D samples of 3D weakly labeled physical activities: a) '*standing*', b) '*walking*', c) '*ascending stairs*', d) '*descending stairs*'

In contrast to well aligned and labeled data, in practice, weakly labeled data can be obtained more easily [12, 1]. We define it as follows.

Definition 5 *Weakly labeled data: A long-term time series is associated with a single global label as shown in Fig.2.*

Due to the alignment-free property of weakly labeled data, it requires to extract subsequences by specific algorithm. The most widely used algorithm is sliding window [14]. By specifying *sliding* **step**, we can extract large amount of redundant subsequences from long-term time series.

In summary, in this paper, we will primarily concentrate on the time series of the same length and conduct experiments on both labeled data that is well aligned and weakly labeled data.

3 Multi-Channels Deep Convolutional Neural Networks

In this section, we will introduce a deep learning framework for multivariate time series classification: Multi-Channels Deep Convolutional Neural Networks (MC-DCNN). Traditional Convolutional Neural Networks (CNN) usually include two

parts. One is a feature extractor, which learns features from raw data automatically. And the other is a trainable fully-connected MLP, which performs classification based on the learned features from the previous part. Generally, the feature extractor is composed of multiple similar stages, and each stage is made up of three cascading layers: filter layer, activation layer and pooling layer. The input and output of each layer are called *feature maps* [11]. In the previous work of CNN [11], the feature extractor usually contains one, two or three such 3-layers stages. Due to space constraint, we only introduce the components of CNN briefly. More details of CNN can be referred to [11, 15].

3.1 Architecture

In contrast to image classification, the input of multivariate time series classification are multiple 1D subsequences but not 2D image pixels. We modify the traditional CNN and apply it to multivariate time series classification task in this way: we separate multivariate time series into univariate ones and perform feature learning on each univariate series individually. Then we concatenate a normal MLP at the end of feature learning to do classification. To be understood easily, we illustrate the architecture of MC-DCNN in Fig. 3. Specifically, this is an example of 2-stages MC-DCNN for activity classification. It includes 3-channels inputs and the length of each input is 256. For each channel, the input (i.e., the univariate time series) is fed into a 2-stages feature extractor, which learns hierarchical features through filter, activation and pooling layers. At the end of feature extractor, we flatten the feature maps of each channel and combine them as the input of subsequent MLP for classification. Note that in Fig. 3, the activation layer is embedded into filter layer in the form of non-linear operation on each feature map. Next, we describe how each layer works.

Filter Layer. The input of each filter is a univariate time series, which is denoted $\mathbf{x}_i^l \in \Re^{n_2^l}$, $1 \leq i \leq n_1^l$, where l denotes the layer which the time series comes from, n_1^l and n_2^l are number and length of input time series. To capture local temporal information, it requires to restrict each trainable filter $\mathbf{k}_{ij}$ with a small size, which is denoted m_2^l, and the number of filter at layer l is denoted m_1^l. Recalling the example described in Fig. 3, in first stage of channel 1, we have $n_1^l = 1$, $n_2^l = 256$, $m_2^l = 5$ and $m_1^l = 8$. We compute the output of each filter according to this: $\sum_i \mathbf{x}_i^{l-1} * \mathbf{k}_{ij}^l + b_j^l$, where the $*$ is convolution operator and b_j^l is the bias term.

Activation Layer. The activation function introduces the non-linearity into neural networks and allows it to learn more complex model. The most widely used activation functions are $sigmoid(t) = \frac{1}{1+e^{-t}}$ and $tanh(\cdot)$. In this paper, we adopt $sigmoid(\cdot)$ function in all activation layers due to its simplicity.

Pooling Layer. Pooling is also called subsampling because it usually subsamples the input feature maps by a specific factor. The purpose of pooling layer is to reduce the resolution of input time series, and make it robust to small variations for previous learned features. The simplest yet most popular method

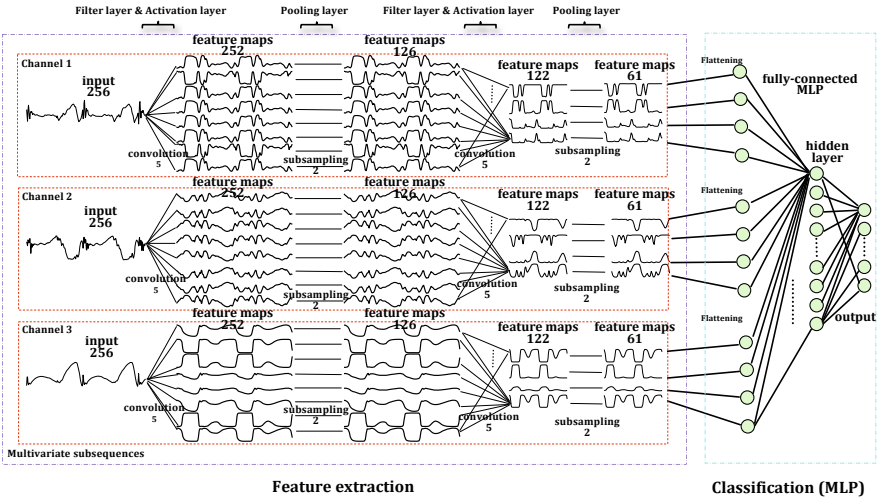

Fig. 3. A 2-stages MC-DCNN architecture for activity classification. This architecture consists of 3 channels input, 2 filter layers, 2 pooling layers and 2 fully-connected layers. This architecture is denoted as *8(5)-2-4(5)-2-732-4* based on the template *C1*(*Size*)-*S1*-*C2*(*Size*)-*S2*-*H*-*O*, where *C1* and *C2* are numbers of filters in first and second stage, *Size* denotes the kernel size, *S1* and *S2* are subsampling factors, *H* and *O* denote the numbers of units in hidden and output layers of MLP.

is to compute *average* value in each neighborhood at different positions with or without overlapping. The neighborhood is usually constructed by splitting input feature maps into equal length (larger than 1) subsequences. We utilize *average* pooling without overlapping for all stages in our work.

3.2 Gradient-Based Learning of MC-DCNN

The same as traditional MLP, for multi-class classification task, the loss function of our MC-DCNN model is defined as: $E = -\sum_t \sum_k y_k^*(t) \log(y_k(t))$, where $y_k^*(t)$ and $y_k(t)$ are the target and predicted values of t-th training example at k-th class, respectively. To estimate parameters of models, we utilize gradient-based optimization method to minimize the loss function. Specifically, we use simple backpropagation algorithm to train our MC-DCNN model, since it is efficient and most widely used in neural networks [16]. We adopt stochastic gradient descent (SGD) instead of full-batch version to update the parameters. Because SGD could converge faster than full-batch for large scale data sets [16].

A full cycle of parameter updating procedure includes three cascaded phases [17]: feedforward pass, backpropagation pass and the gradient applied.

Feedforward Pass. The objective of feedforward pass is to determine the predicted output of MC-DCNN on input vectors. Specifically, it computes feature maps from layer to layer and stage to stage until obtaining the output. As shown in the previous content, each stage contains three cascaded layers, and activation

layer is embedded into filter layer in form of non-linear operation on each feature map. We compute output feature map of each layer as follows:

$$\mathbf{z}_j^l = \sum_i \mathbf{x}_i^{l-1} * \mathbf{k}_{ij}^l + b_j^l, \quad \mathbf{x}_j^l = sigmoid(\mathbf{z}_j^l), \quad \mathbf{x}_j^{l+1} = down(\mathbf{x}_j^l)$$

where $down(\cdot)$ represents the subsampling function for *average* pooling, $\mathbf{x}_i^{l-1}$ and $\mathbf{z}_j^l$ denote the input and output of filter layer, $\mathbf{z}_j^l$ and $\mathbf{x}_j^l$ denote the input and output of activation layer, $\mathbf{x}_j^l$ and $\mathbf{x}_j^{l+1}$ denote the input and output of pooling layer.

Eventually, a 2-layer fully-connected MLP is concatenated to feature extractor. Since feedforward pass of MLP is standard and also the space is limited, more details of MLP can be referred to [16, 17].

Backpropagation Pass. Once acquiring predicted output $\mathbf{y}$, the predicted error E can be calculated according to the loss function. By taking advantage of chain-rule of derivative, the predicted error propagates back on each parameter of each layer one by one, which can be used to work out the derivatives of them. We still don't present backpropagation pass of final MLP for the same reason of feedforward pass.

For pooling layer in the second stage of feature extractor, the derivative of $\mathbf{x}_j^{l-1}$ is computed by the upsampling function $up(\cdot)$, which is an inverse operation opposite to the subsampling function $down(\cdot)$ for the backward propagation of errors in this layer.

$$\frac{\partial E}{\partial \mathbf{x}_j^{l-1}} = up(\frac{\partial E}{\partial \mathbf{x}_j^l})$$

For filter layer in second stage of feature extractor, derivative of $\mathbf{z}_j^l$ is computed similar to that of MLP's hidden layer:

$$\boldsymbol{\delta}_j^l = \frac{\partial E}{\partial \mathbf{z}_j^l} = \frac{\partial E}{\partial \mathbf{x}_j^l}\frac{\partial \mathbf{x}_j^l}{\partial \mathbf{z}_j^l} = sigmoid^{'}(\mathbf{z}_j^l) \circ up(\frac{\partial E}{\partial \mathbf{x}_j^{l+1}})$$

where $\circ$ denotes element-wise product. Since the bias is a scalar, to compute its derivative, we should summate over all entries in $\boldsymbol{\delta}_j^l$ as follows:

$$\frac{\partial E}{\partial b_j^l} = \sum_u (\boldsymbol{\delta}_j^l)_u$$

The difference between kernel weight $\mathbf{k}_{ij}^l$ and MLP's weight w_{ij}^l is the weight sharing constraint, which means the weights between $(\mathbf{k}_{ij}^l)_u$ and each entry of $\mathbf{x}_j^l$ must be the same. Due to this constraint, the number of parameters is reduced by comparing with the fully-connected MLP, Therefore, to compute the derivative of kernel weight $\mathbf{k}_{ij}^l$, it needs to summate over all quantities related to this kernel. We perform this with convolution operation:

$$\frac{\partial E}{\partial \mathbf{k}_{ij}^l} = \frac{\partial E}{\partial \mathbf{z}_j^l}\frac{\partial \mathbf{z}_j^l}{\partial \mathbf{k}_{ij}^l} = \boldsymbol{\delta}_j^l * reverse(\mathbf{x}_i^{l-1})$$

where $reverse(\cdot)$ is the function of reversing corresponding feature map. Finally, we compute the derivative of $\mathbf{x}_i^{l-1}$ as follows:

$$\frac{\partial E}{\partial \mathbf{x}_i^{l-1}} = \sum_j \frac{\partial E}{\partial \mathbf{z}_j^l} \frac{\partial \mathbf{z}_j^l}{\partial \mathbf{x}_i^{l-1}} = \sum_j pad(\boldsymbol{\delta}_j^l) * reverse(\mathbf{k}_{ij}^l)$$

where $pad(\cdot)$ is a function which pads zeros into $\boldsymbol{\delta}_j^l$ from two ends, e.g., if the size of $\mathbf{k}_{ij}^l$ is n_2^l, then this function will pad each end of $\boldsymbol{\delta}_j^l$ with $n_2^l - 1$ zeros.

Gradients Applied. Once we obtain the derivatives of parameters, it's time to apply them to update parameters. To converge fast, we utilize *decay* and *momentum* strategies [16]. The weight w_{ij}^l in MLP is updated in this way:

$$w_{ij}^l = w_{ij}^l + \Delta w_{ij}^l$$

$$\Delta w_{ij}^l = momentum \cdot \Delta w_{ij}^l - decay \cdot \epsilon \cdot w_{ij}^l - \epsilon \cdot \frac{\partial E}{\partial w_{ij}^l}$$

where w_{ij}^l represents the weight between x_i^{l-1} and x_j^l, Δw_{ij}^l denotes the gradient of w_{ij}^l, and ϵ denotes the learning rate. The kernel weight $\mathbf{k}_{ij}^l$, the bias term b_j^l in filter layer and b^l in MLP are updated similar to the way of w_{ij}^l. The same as [18], we set $momentum = 0.9$, $decay = 0.0005$ and $\epsilon = 0.01$ for our experiments. It is noted that [19] claimed that both the initialization and the momentum are crucial for deep neural networks, hence, we consider how to select these values as a part of our future work.

4 Experiments

In this section, we will conduct two groups of experiments on real-world data sets from two different application domains. Particularly, we will show the performance of our methods via comparing with other baseline models in terms of both efficiency and accuracy.

To the best of our knowledge, indeed, there are many public time series data sets available, e.g., the UCR Suite [20]. However, we decide not using the UCR Suite for the following reasons. First, we focus on the classification of multivariate time series, whereas most data sets in UCR Suite only contain univariate time series. Second, data sets in UCR Suite are usually small and CNN may not work well on such small data sets [21]. Thus, we choose two data sets which are collected from real-world applications, and we will introduce the data sets in the next subsections.

We consider three approaches as baseline methods for evaluation: *1*-NN (ED), *1*-NN (DTW-5%) and MLP. Here, *1*-NN (ED) and *1*-NN (DTW-5%) are the methods that combine Euclidean Distance and Window Constraint DTW [7]) [1] with *1*-NN, respectively. Besides these two state-of-the-art methods, MLP is

[1] Following the discoveries in [7], we set the optimal window constraint r as 5%.

chosen to demonstrate that the feature learning process can improve the classification accuracy effectively. For the purpose of comparison, we record the performance of each method by tuning their parameters. Notice that some other classifiers are not considered here, since it is difficult to construct hand-crafted features for time series and many previous works have claimed that feature-based methods cannot achieve the accuracy as high as *1*-NN methods. Also, we do not choose the full DTW due to its expensive time consumption. Actually, at least more than a month will be cost if we use full DTW in our experiments.

4.1 Activity Classification (Weakly Labeled Data)

Data Set. We use the weakly labeled PAMAP2 data set for activity classification . It records 19 physical activities performed by 9 subjects. On a machine with Intel I5-2410 (2.3GHz) CPU and 8G Memory (our experimental platform), according to the estimation, it will cost nearly a month for *1*-NN (DTW-5%) on this data set if we use all the 19 physical activities. Hence, currently, we only consider 4 out of these 19 physical activities in our work, which are '*standing*', '*walking*', '*ascending stairs*' and '*descending stairs*'. And each physical activity corresponds to a **3D** time series. Moreover, 7 out of these 9 subjects are chosen. Because the other two either have different physical activities or have different dominant hand/foot.

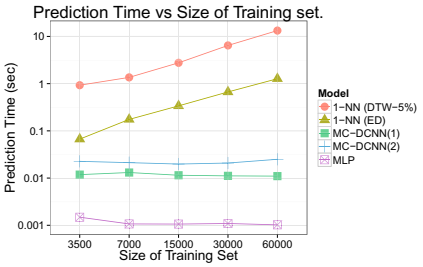

Fig. 4. Prediction time of each model on training sets with different size

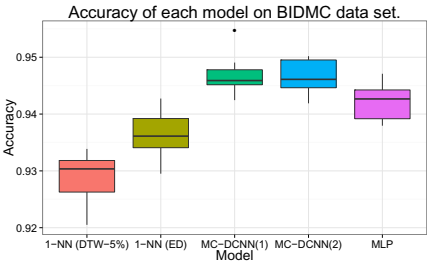

Fig. 5. The box-and-whisker plot of classification accuracy on BIDMC data set

Experiment Setup. We normalize each dimension of **3D** time series as $\frac{x-\mu}{\sigma}$, where μ and σ are mean and standard deviation of time series. Then we apply the sliding window algorithm to extract subsequences from **3D** time series with different *sliding steps*. To evaluate the performance of different models, we adopt the *leave-one-out* cross validation (LOOCV) technique. Specifically, each time we use one subject's physical activities as test data, and the physical activities of remaining subjects as training data. Then we repeat this for every subject. To glance the impact of depths, we evaluate two models: MC-DCNN(1), MC-DCNN(2). They are 1-stage and 2-stages feature learning models, respectively.

Experimental Results To evaluate efficiency and scalability of each model, we get five data splits with different volumes by setting *sliding step* as 128, 64,

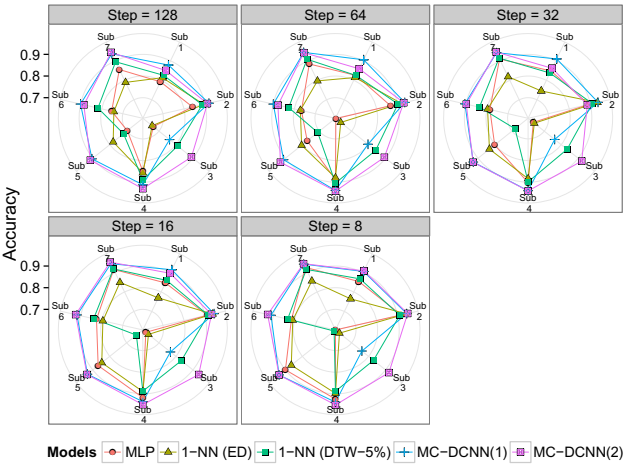

Fig. 6. Classification accuracy on each subject with different *sliding steps*

32, 16, 8, respectively. In addition, to ensure each subsequence to cover at least one pattern of time series, we set the sliding window length as 256.

As is well known, feature-based models have an advantage over lazy classification models (e.g., *k*-NN) in efficiency. As shown in Fig. 4, the prediction time of *1*-NN model increases linearly as the size of training data set grows. In contrast, the prediction time of our MC-DCNN model is almost constant no matter how large the training data is.

We also evaluate accuracy of each model on these five data splits. Fig. 6 shows the detailed accuracy comparisons of each subject at different *step* settings. From this figure we can see that for each subject our MC-DCNN model is either the most accuracy one or very close to the most accuracy one. Especially, for subject 3, the 2-stages MC-DCNN leads to much better accuracy than other approaches. We suppose that 2-stages MC-DCNN may learn high-level and robust feature representations so that it has a good generalization. We also show the average and standard deviation of accuracy in Table 1. From the table we can see that our model leads to the highest average accuracy and the lowest standard deviation.

Table 1. Average and standard deviation of accuracy of each model at different *sliding step*. **Bold** numbers represent the best results.

Step	*1*-NN (DTW-5%)	MLP	*1*-NN (ED)	MC-DCNN(1)	MC-DCNN(2)
128	83.46 (0.063)	77.89 (0.076)	79.05 (0.076)	88.73 (0.057)	**90.34 (0.031)**
64	84.51 (0.070)	80.09 (0.098)	80.25 (0.089)	90.38 (0.050)	**91.00 (0.033)**
32	84.44 (0.080)	82.49 (0.096)	80.74 (0.094)	90.28 (0.063)	**91.14 (0.031)**
16	84.16 (0.094)	84.34 (0.104)	81.74 (0.096)	90.75 (0.062)	**93.15 (0.019)**
8	83.61 (0.104)	84.83 (0.115)	82.28 (0.103)	90.53 (0.065)	**93.36 (0.015)**

4.2 Congestive Heart Failure Detection (Well Aligned Data)

Data Set Well aligned BIDMC data set was downloaded from Congestive Heart Failure database [2] [13]. Long-term electrocardiograph (ECG) data was recorded from 15 subjects, each of them suffers severe Congestive Heart Failure. Different from PAMAP2 data, in BIDMC data set, each type of heart failure corresponds to a **2D** time series. In this experiment, we consider four types of heartbeats to evaluate all the models: 'N', 'V', 'S', 'r'.

Experiment Setup. We still normalize each univariate of **2D** time series as mentioned before. Different from weakly data, we extract subsequences centered at aligned marks (red dotted line in Fig. 1). And each subsequence still has a length of 256. Similar to the classification of individuals' heartbeats [12], we mix all data of 15 subjects and randomly split it into 10 folds to perform 10-folds cross validation. Because as [12] noted, it can be able to obtain huge amounts of labeled data in this way and a unhealthy individual may have many different *types* of heartbeats. To glance the impact of depths, we also evaluate two models: MC-DCNN(1), MC-DCNN(2). The former performs 1-stage feature learning, and the latter performs 2-stages. To determine the epochs, we separate one third of training data as validation set. As shown in Fig. 7, we set epoch to 40 and 80 for 1-stage and 2-stages MC-DCNN models respectively. Since the test error is stable when epochs are greater than them.

Experimental Results. We illustrate the accuracy of each model on BIDMC data set in Fig. 5. From this figure, we can see that accuracies of 1-stage MC-DCNN and 2-stages MC-DCNN models are 94.67% and 94.65%, which are also higher than the accuracies of *1*-NN(ED) (93.64%), *1*-NN(DTW-5%) (92.90%) and MLP (94.22%). Due to the space limit we do not report the prediction time of each model on BIDMC data set. However, the result is similar to Fig. 4 and it also supports that feature-based models have an advantage over lazy classification models (e.g., *k*-NN) in efficiency.

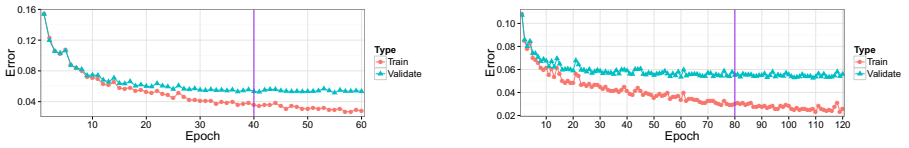

Fig. 7. Test error on validation set, left) 1-stage MC-DCNN model and right) 2-stages MC-DCNN model. The vertical purple line indicates the determined epoch.

5 Related Work

Many time series classification methods have been proposed based on different sequence distance measurements. Among these previous works, some researchers

[2] `http://www.physionet.org/physiobank/database/chfdb/`

claimed that *1*-NN combined DTW is the current state of the art [6, 7]. However, the biggest weakness of *1*-NN with DTW model is its expensive computation [7]. To overcome this drawback, a part of researchers explored to speed up the computation of distance measure (e.g., DTW) in certain methods (e.g., with boundary conditions) [7]. While another part of researchers tried to reduce the computation of *1*-NN by constructing data dictionary [12, 7, 14, 22]. When the data set grows large, all these approaches improve the performance significantly in contrast to simple *1*-NN with DTW. Some feature-based models have been explored for time series classification [2, 23], however, most of previous works extracted the hand-crafted statistical features based on domain knowledge, and achieved the performance not as well as sequence distance based models.

Feature learning (or representation learning) is becoming an important field in machine learning community in recent years [9]. The most successful feature learning framework is deep neural networks, which build hierarchical representations from raw data [10, 11, 24]. Particularly, as a supervised feature learning model, deep convolutional neural networks achieve remarkable successes in many tasks such as digit and object recognition [18], which motivates us to investigate the deep learning in time series field. In the literature, there are few works on time series classification using deep learning. Ref.[25] explored an unsupervised feature learning method with convolutional deep belief networks for audio classification, but in frequency domain rather than in time domain. Ref.[3] adopted a special time delay neural network (TDNN) model for electroencephalography (EEG) classification. However, their TDNN model only included a single hidden layer, which is not deep enough to learn good hierarchical features. To the best of our knowledge, none of existing works on time series classification has considered the supervised feature learning from raw data. In this paper, we explore a MC-DCNN model for multivariate time series classification and intend to investigate this problem in another way.

6 Conclusion and Future Work

Time series classification is becoming very important in a broad range of real-world applications, such as health care and activity recognition. However, most existing methods have high computational complexity or low prediction accuracy. To this end, we developed a novel deep learning framework (MC-DCNN) to classify multivariate time series in the paper. This model learns features from individual univariate time series in each channel automatically, and combines information from all channels as feature representation at final layer. A traditional MLP is concatenated to perform classification. We evaluated our MC-DCNN model on two real-world data sets. Experimental results show that our MC-DCNN model outperforms the competing baseline methods on both data sets, especially, the improvement of accuracy on weakly labeled data set is significant. Also, we showed that 2-stages MC-DCNN is superior to 1-stage MC-DCNN. It provides the evidence that the deeper architecture can learn more robust high-level features, which is helpful for improving performance of classification.

There are several research directions for future work. First, in this paper we simply use the 1-stage and 2-stages feature learning for better illustration, and in the future we plan to study and extend other deep learning models for multivariate time series classification on more data sets. Second, we also intend to perform unsupervised algorithms on unlabeled data to pre-train the networks.

Acknowledgement. This research was partially supported by grants from the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61325010), the National High Technology Research and Development Program of China (Grant No.2014AA015203), the Anhui Provincial Natural Science Foundation (Grant No. 1408085QF110), the Science and Technology Development of Anhui Province, China (Grants No. 13Z02008-5 and 1301022064), and the International Science & Technology Cooperation Plan of Anhui Province (Grant No. 1303063008).

References

1. Reiss, A., Stricker, D.: Introducing a modular activity monitoring system. In: 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC, pp. 5621–5624 (2011)
2. Kampouraki, A., Manis, G., Nikou, C.: Heartbeat time series classification with support vector machines. IEEE Transactions on Information Technology in Biomedicine 13(4), 512–518 (2009)
3. Haselsteiner, E., Pfurtscheller, G.: Using time-dependent neural networks for EEG classification. IEEE Transactions on Rehabilitation Engineering 8(4), 457–463 (2000)
4. Batista, G.E.A.P.A., Wang, X., Keogh, E.J.: A complexity-invariant distance measure for time series. In: SIAM Conf. Data Mining (2011)
5. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.: Querying and mining of time series data: experimental comparison of representations and distance measures. In: Proceedings of the VLDB Endowment, vol. 1(2), pp. 1542–1552 (2008)
6. Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: 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, KDD 2012, p. 262 (2012)
7. Xi, X., Keogh, E.J., Shelton, C.R., Wei, L., Ratanamahatana, C.A.: Fast time series classification using numerosity reduction. In: International Conference on Machine Learning, pp. 1033–1040 (2006)
8. Xing, Z., Pei, J., Keogh, E.J.: A brief survey on sequence classification. Sigkdd Explorations 12(1), 40–48 (2010)
9. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. arXiv preprint arXiv:1206.5538 (2012)
10. LeCun, Y., Bengio, Y.: Convolutional networks for images, speech, and time series. The Handbook of Brain Theory and Neural Networks 3361 (1995)

11. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 253–256. IEEE (2010)
12. Hu, B., Chen, Y., Keogh, E.: Time Series Classification under More Realistic Assumptions. In: SIAM International Conference on Data Mining, p. 578 (2013)
13. Goldberger, A.L., Amaral, L.A.N., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.: PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation 101(23), e215–e220 (2000)
14. Ye, L., Keogh, E.: Time series shapelets: a new primitive for data mining. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 947–956. ACM (2009)
15. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
16. LeCun, Y.A., Bottou, L., Orr, G.B., Müller, K.-R.: Efficient backProp. In: Orr, G.B., Müller, K.-R. (eds.) NIPS-WS 1996. LNCS, vol. 1524, pp. 9–50. Springer, Heidelberg (1998)
17. Bouvrie, J.: Notes on convolutional neural networks (2006)
18. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems 25, 1106–1114 (2012)
19. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, June 16-21, vol. 28 (2013)
20. Keogh, E., Zhu, Q., Hu, B., Hao, Y., Xi, X., Wei, L., Ratanamahatana, C.A.: The UCR Time Series Classification/Clustering Homepage (2011), http://www.cs.ucr.edu/~eamonn/time_series_data/
21. Pinto, N., Cox, D.D., DiCarlo, J.J.: Why is real-world visual object recognition hard? PLoS Computational Biology 4(1), e27 (2008)
22. Lines, J., Davis, L.M., Hills, J., Bagnall, A.: A shapelet transform for time series classification. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 289–297 (2012)
23. Nanopoulos, A., Alcock, R.O.B., Manolopoulos, Y.: Feature-based classification of time-series data. Information Processing and Technology 0056, 49–61 (2001)
24. Lee, H., Grosse, R., Ranganath, R., Ng, A.Y.: Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 609–616. ACM (2009)
25. Lee, H., Largman, Y., Pham, P., Ng, A.Y.: Unsupervised Feature Learning for Audio Classification using Convolutional Deep Belief Networks. Advances in Neural Information Processing Systems 22, 1096–1104 (2009)