Front.Comput.Sci.
DOI

RESEARCH ARTICLE

Exploiting Multi-Channels Deep Convolutional Neural Networks for Multivariate Time Series Classification

Yi ZHENG[1,3], Qi LIU[1], Enhong CHEN[1](✉), Yong GE[2], J. Leon ZHAO[3]

1 School of Computer Science and Technology, University of Science and Technology of China, Hefei, 230027, China
2 Department of Computer Science, University of North Carolina at Charlotte
3 Department of Information Systems, City University of Hong Kong, Hong Kong

Abstract Time series classification is related to many different domains, such as health informatics, finance, and bioinformatics. Due to its broad applications, researchers have developed many algorithms for this kind of tasks, e.g., multivariate time series classification. Among the classification algorithms, Nearest Neighbor (*k*-NN) classification (particularly *1*-NN) combined with Dynamic Time Warping (DTW) achieves the state of the art performance. The deficiency is that when the data set grows large, the time consumption of *1*-NN with DTW will be very expensive. In contrast to *1*-NN with DTW, it is more efficient but less effective for feature-based classification methods since their performance usually depends on the quality of hand-crafted features. In this paper, we aim to improve the performance of traditional feature-based approaches through the feature learning techniques. Specifically, we propose a novel deep learning framework, Multi-Channels Deep Convolutional Neural Networks (MC-DCNN), for multivariate time series classification. This model first learns features from individual univariate time series in each channel, and combines information from all channels as feature representation at the final layer. Then, the learnt features are applied into a Multilayer Perceptron (MLP) for classification. Finally, the extensive experiments on real-world data sets show that our model is not only more efficient than the state of the art but also competitive in accuracy. This study implies that feature learning is worth to be investigated for the problem of time series classification.

Received October 27, 2014; accepted month dd, yyyy

E-mail: cheneh@ustc.edu.cn

Keywords Convolutional Neural Networks,Time Series Classification,Feature Learning,Deep Learning

1 Introduction

As the development of information technology, sensors become cheaper and more prevalent in recent years. Hence, a large amount of time series data (e.g., Electrocardiograph) can be collected from different domains such as bioinformatics and finance. Indeed, the topic of time series data mining, e.g., univariate time series classification and multivariate time series classification, has drawn a lot of attention [1–4].

Particularly, compared to univariate time series, multivariate time series can provide more patterns and views of the same underlying phenomena, and help improve the classification performance. Therefore, multivariate time series classification is becoming more and more important in a broad range of applications, such as activity recognition and healthcare [5–7]. In this paper, we focus on the classification of multivariate time series. Along this line, there has been a number of classification algorithms developed. As many previous studies claimed [2, 8], among these methods, the distance-based method *k*-Nearest Neighbor (*k*-NN) classification is very difficult to beat. On the other hand, more evidences have also shown that the Dynamic Time Warping (DTW) is the best sequence distance measurement for time series in many domains [2, 8–10]. Hence, it could reach the best performance of classification through combining the *k*-NN and DTW in most scenarios [9]. As contrasted to the sequence distance based methods, traditional feature-

based classification methods could also be applied for time series [1], and the performance of these methods depends on the quality of hand-crafted features heavily. However, different from other applications, for time series data, it is difficult to manually design good features to capture the intrinsic properties. As a consequence, the classification accuracy of feature-based approaches is usually worse than that of sequence distance based approaches, particularly *1*-NN with DTW method. Recall that both of *1*-NN and DTW are effective but cause too much computation in many applications [10]. Hence, we have the following motivation.

Motivation. *Is it possible to improve the accuracy of feature-based methods for multivariate time series? 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 [11–13], in our preliminary work [14], we introduced a deep learning framework for multivariate time series classification. Deep learning does not need any hand-crafted features, as it can learn a hierarchical feature representation from raw data automatically. Specifically, we proposed an effective Multi-Channels Deep Convolutional Neural Networks (MC-DCNN) model [1)], each channel of which takes a single dimension of multivariate time series as input and learns features individually. After that, the MC-DCNN model combines the learnt features of each channel and feeds them into a Multilayer Perceptron (MLP) to perform further classification. We adopted the gradient-based method to estimate the parameters of the model. Finally, we evaluated the performance of our MC-DCNN model on several real-world data sets. The experimental results on these data sets reveal that our MC-DCNN model outperforms the baseline methods with significant margins and has a good generalization, especially for weakly labeled data. In this extended version, we integrate novel activation function and pooling strategy, and meanwhile, we compare its rate of convergence with the traditional combinations of activation functions and pooling strategies. To further improve the performance, we also apply an unsupervised initialization to pretrain the convolutional neural networks and propose the pretrained version of MC-DCNN model. Moreover, in order to understand the learnt features intuitively, we provide visualizations of them at two filter layers. Through the new investigation, we summarize several discussions of current study and they guide the directions for our future work.

The rest of the paper is organized as follows. In Section 2, we provide some preliminaries. 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 several real-world data sets and discuss the performance of each model. We make a short review of related work in Section 5. Finally, we conclude the paper in Section 6.

[1)] A preliminary version of this work has been published in the proceedings of WAIM 2014 (full paper).

2 Preliminaries

In this section, we first introduce some definitions and notations. Then, we define two distance measures used in the paper.

2.1 Definitions and Notations

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* $T = \{t_1, t_2, ..., t_n\}$.

Definition 2. *Multivariate time series is a set of time series with the same timestamps. A multivariate time series M is a* $n \times l$ *matrice where the* j^{th} *row and* i^{th} *column of M are* $m_{j\cdot}$ *and* $m_{\cdot i}$. *While n and l show the length and the number of single univariate time series in M. Correspondingly,* $m_{j\cdot}$ *and* $m_{\cdot i}$ *represent the* j^{th} *univariate time series as shown in Definitions 1 and the points of these univariate time series at time i.*

We follow previous work [15] and extract subsequences from long time series to perform classification instead of directly classifying with the entire sequence.

Definition 3. *Subsequence is a sequence of consecutive points which are extracted from time series T and can be denoted as* $S = \{t_i, t_{i+1}, ..., t_{i+k-1}\}$, *where k is the length of subsequence. Similarly, multivariate subsequence can be denoted as* $Y = \{m_{\cdot i}, m_{\cdot i+1}, ..., m_{\cdot i+k-1}\}$, *where* $m_{\cdot i}$ *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 with the same pattern are well aligned.*

Algorithm 1 Sliding window algorithm

```
1: procedure [s]=SlidingWindow(T, L, P)
2:     i := 0, m := 0;                          ▷ m is the number of subsequences.
3:     s := empty;                              ▷ The set of extracted subsequences.
4:     while i + L ⩽ length(T) do
5:                                              ▷ L is the length of sliding window.
6:         s[m] := T[i..(i + L − 1)];
7:         i := i + P, m := m + 1;              ▷ P is the step.
8:     end while
9: end procedure
```

Fig.1 shows a snippet of time series extracted from BIDMC Congestive Heart Failure data set [16]. Each subsequence is extracted and labeled according to the dotted line by medical staffs. However, to acquire the well aligned and labeled data, it always needs great manual cost.

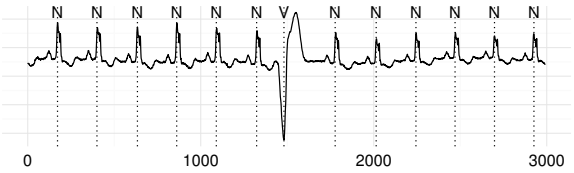


Fig. 1 A snippet of time series which contains two types of heartbeat: normal (N) and ventricular fibrillation (V).

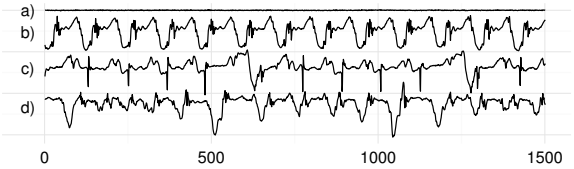


Fig. 2 Four 1D samples of 3D weakly labeled physical activities: a) '*standing*', b) '*walking*', c) '*ascending stairs*', d) '*descending stairs*'.

In contrast to well aligned and labeled data, in practice, weakly labeled data can be obtained more easily [7, 15], which is defined 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 [17], by which a large number of redundant subsequences may be extracted and all kinds of potential subsequences in alignment-free pattern space are covered. We illustrate the pseudo-code of sliding window algorithm in Algorithm 1. The parameters T, L and P denote the long-term time series, the window length and the sliding step. Supposing the length of T is n, it is easy to find that the number of extracted subsequences is $\lceil \frac{n-L+1}{P} \rceil$.

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.

2.2 Time Series Distance Measure

For time series data, Euclidean distance is the most widely used measure. Suppose we have two univariate time series Q and C, of same length n, where $Q = \{q_1, q_2, ..., q_i, ..., q_n\}$ and $C = \{c_1, c_2, ..., c_i, ..., c_n\}$. The Euclidean distance between Q and C is:

$$\mathrm{ED}(C, Q) = \sqrt{\sum_{i=1}^{n} (c_i - q_i)^2}$$

Furthermore, for two multivariate time series X and Y, the Euclidean distance between them could be defined as follows:

$$\mathrm{ED}(X, Y) = \sum_{i=1}^{l} \mathrm{ED}(x_i, y_i)$$

where l denotes the number of components, and x_i and y_i represent the i^{th} univariate time series of them, respectively. As a simple measure, Euclidean distance suffers from several problems [2]. In the literature, DTW is considered as an alternative that is more robust than Euclidean distance and can align two sequences effectively [2, 9, 10]. To align two sequences (e.g., Q and C) using DTW, we need to construct a distance matrix first, which is shown as follows.

$$\begin{pmatrix} d(q_1, c_1) & d(q_1, c_2) & \cdots & d(q_1, c_n) \\ d(q_2, c_1) & d(q_2, c_2) & \cdots & d(q_2, c_n) \\ \vdots & \vdots & \ddots & \vdots \\ d(q_n, c_1) & d(q_n, c_2) & \cdots & d(q_n, c_n) \end{pmatrix}$$

Each element $d(q_i, c_j)$ of the matrix corresponds to the distance between the i-th point of Q and the j-th point of C, i.e., $d(q_i, c_j) = (q_i - c_j)^2$. In this paper we primarily concentrate on the time series of the same length, thus we use two equal length time series to explain DTW for convenience. Notice that DTW can also be applied to time series with different length.

A warping path W is a sequence of contiguous matrix elements which defines a mapping between Q and C: $W = \{w_1, w_2, ...w_k, ..., w_{2n-1}\}$. Each element of W corresponds to a certain element of the distance matrix (i.e., $w_k = d(q_i, c_j)$). There are three constraints that the warping path should satisfy, which are shown as follows.

- Boundary : $w_1 = d(q_1, c_1)$ and $w_{2n-1} = d(q_n, c_n)$.
- Continuity: Given $w_k = d(q_i, c_j)$ and $w_{k-1} = d(q_{i'}, c_{j'})$, then $i - i' \leqslant 1$ and $j - j' \leqslant 1$.
- Monotonicity: Given $w_k = d(q_i, c_j)$ and $w_{k-1} = d(q_{i'}, c_{j'})$, then $i - i' \geqslant 0$ and $j - j' \geqslant 0$.

Since there may exist exponentially many warping paths, the aim of DTW is to find out the one which has the minimal warping cost:

$$\mathrm{DTW}(Q, C) = \underset{W=\{w_1, w_2, ..., w_{2n-1}\}}{\text{minimize}} \sum_{k=1}^{2n-1} w_k$$

where W should satisfy these three constraints above. One step further, for two multivariate time series X and Y, similar to Euclidean distance, DTW between X and Y can be defined as follows:

$$\mathrm{DTW}(X, Y) = \sum_{i=1}^{l} \mathrm{DTW}(x_i, y_i)$$

where l denotes the number of components in multivariate time series, and both of x_i and y_i represent the i^{th} univariate time series of them, respectively.

It is common to apply dynamic programming to compute $\mathrm{DTW}(Q, C)$ (or $\mathrm{DTW}(X, Y)$), which is very efficient and has a time complexity $O(n^2)$ in this context. However, when the size of data set grows large and the length of time series becomes long, it is very time consuming to compute DTW combined with k-NN method. Hence, to reduce the time consumption, window constraint DTW has been adopted widely instead of full DTW in many previous work [10, 18–20]. On the other hand, from the intuition, the warping path is unlikely to go very far from the diagonal of the distance matrix [10]. In other words, for any element $w_k = d(q_i, c_j)$ in the warping path, the difference between i and j should not be too large. By limiting the warping path to a warping window, some previous work [10, 19] showed that relatively tight warping windows actually improve the classification accuracy.

According to above discussions, we consider both Euclidean distance and window constraint DTW as the default distance measures in the following.

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. 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* [13]. In the previous work of CNN [13], the feature extractor usually contains one, two or three such 3-layers stages. For remainder of this section, we first introduce the components of CNN briefly and more details of CNN can be referred to [13, 21]. Then, we show the gradient-based learning of our model. After that, the related unsupervised pretraining is given at the end of this section.

3.1 Architecture

In contrast to image classification, the inputs 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, and then a traditional MLP is concatenated at the end of feature learning that is used to do the 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 with pretraining for activity classification. Once the pretraining is completed, the initial weights of network are obtained. Then, the inputs of 3-channels are fed into a 2-stages feature extractor, which learns hierarchical features through filter, activation and pooling layers. At the end of feature extractor, the feature maps of each channel are flatten and combined 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. We describe how each layer works in the following subsections.

3.1.1 Filter Layer

The input of each filter is a univariate time series, which is denoted $\mathbf{x}_i^l \in \Re^{n_2^l}$, $1 \leqslant i \leqslant n_1^l$, 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. To determine the size of each filter $\mathbf{k}_{ij}$, we follow the earlier studies [22] and set it to 5 ($m_2 = 5$) as they suggested.

3.1.2 Activation Layer

The activation function introduces the non-linearity into neural networks and allows it to learn more complex model.

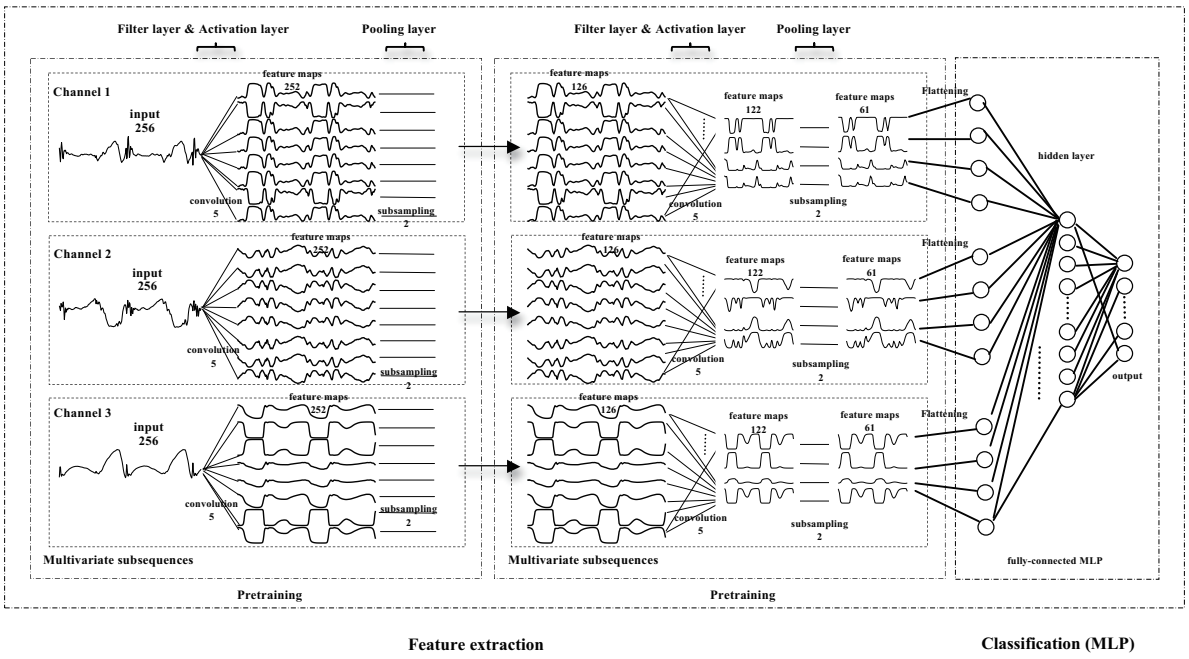


Fig. 3 A 2-stages MC-DCNN architecture for activity classification. This architecture consists of 3 channels input, 2 filter layers, 2 pooling layers and 2 fully-connected layers. Pretraining is performed for two stages gradually and then supervised learning is applied. This architecture is denoted as *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.

Two activation functions $sigmoid(x) = \frac{1}{1+e^{-x}}$ and $tanh(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}}$ are most widely used in artificial neural networks for a long time. In recent years, a novel activation function named as "rectified linear units" (ReLU) has been attempted by many studies [23, 24]. ReLU is defined as: $f(x) = \max(0, x)$. When comparing with traditional activation functions (i.e., $sigmoid(\cdot)$ and $tanh(\cdot)$), the merit of ReLU is that it can improve the generalization and make the training of networks become faster and simpler [23, 24]. Besides, this activation function could avoid vanishing gradient issue and be computed efficiently. For easy understanding, we illustrate ReLU and these two common activation functions in Fig. 4. In this extended version, we consider integrating ReLU into our MC-DCNN models to improve the performance of time series classification.

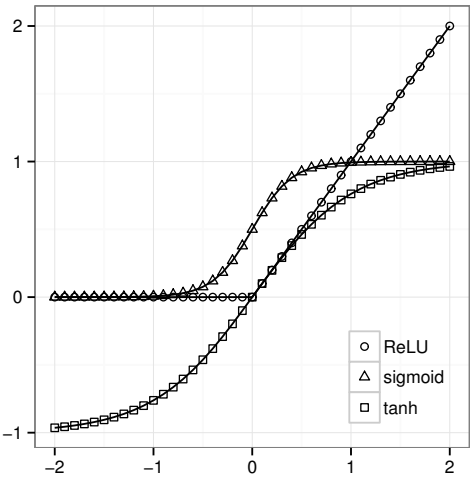


Fig. 4 Illustration of non-linearity, $sigmoid(\cdot)$, $tanh(\cdot)$ and ReLU functions.

3.1.3 Pooling Layer

Pooling is also called as 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 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. Another widely used method is *max* pooling which computes the maximum in the neighborhood in contrast to *average* pooling. Previous studies [25, 26] claimed that *max* pooling is superior to *average* pooling for image classification and it also leads to faster convergence rate by selecting superior invariant features that improve generalization performance. Hence, to further improve the performance, in this extended version, we apply the *max* pooling strategy in our deep convolutional

neural networks.

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 [27]. 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 [27].

A full cycle of parameter updating procedure includes three cascaded phases [28]: feedforward pass, backpropagation pass and the gradient applied.

3.2.1 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:

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

where $down(\cdot)$ represents the subsampling function for either *average* or *max* pooling, $\phi(\cdot)$ represents the activation function (either $sigmoid(\cdot)$ or ReLU here), $\mathbf{x}_i^{l-1}$ 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. Due to feedforward pass of MLP is standard and the space is limited, more details of MLP can be referred to [27, 28].

3.2.2 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 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:

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

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 to reverse the sequence with respect to each 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.

3.2.3 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 [27]. 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 [29], we set $momentum = 0.9$, $decay = 0.0005$ and $\epsilon = 0.01$ for our experiments. It is noted that Ref. [30] claimed that both the initialization and the momentum are crucial for deep neural networks, hence, we consider the way of selecting these values as a part of our future work.

3.3 Pretraining

One challenge of neural networks especially for deep architectures is how to avoid bad local minima during the learning process. Many previous studies claimed that a greedy layer-wise unsupervised initialization could alleviate the local minima issue and achieve better performance [31, 32]. In this subsection, we consider such a kind of unsupervised pretraining for our deep neural networks. Specifically, we first recall conventional pretraining methods (Auto-Encoder and Denoising Auto-Encoder) and then explain how to pretrain the CNN with stacked Convolutional Auto-Encoder (CAE) [33].

3.3.1 Auto-Encoder

Basically, an Auto-Encoder is a three layers artificial neural network, which is very similar to traditional MLP. The aim of Auto-Encoder is to learn a transformation from original space to a lower-dimensional space. Through the learnt transformation, the compressed low-dimensional representations could be used as features for many other tasks. Meanwhile, another usage of Auto-Encoder is to initialize the neural networks for better performance. By stacking a series of Auto-Encoders, the constructed deep architecture could be further fine-tuned in the form of supervised learning. Suppose the input is $\mathbf{x} \in \mathcal{R}^d$ and the hidden representation is denoted as $\mathbf{h} \in \mathcal{R}^{d'}$. In an Auto-Encoder, $\mathbf{h} = \phi(\mathbf{W}\mathbf{x} + b)$ and $\mathbf{y} = \phi(\mathbf{W}'\mathbf{h} + b')$, where $\mathbf{y}$ represents the reconstruction of input $\mathbf{x}$ and ϕ denotes the activation function. The only constraint here is $\mathbf{W}' = \mathbf{W}^T$. Analogous to traditional neural networks, we could learn the parameters $\theta = \{\mathbf{W}, b, b'\}$ through minimizing the reconstructed error: $E(\theta) = ||\mathbf{x} - \mathbf{y}||^2$.

3.3.2 Denoising Auto-Encoder

The deficiency of conventional Auto-Encoder is that it may learn the identity transformation if we do not apply any constraints on it. The common solution for this is either adding a regularized sparsity to the cost function or adding random noisy to the inputs. The latter one named as Denoising Auto-Encoder is more popular due to its simplicity. Since the only modification is to add a variable v to each element of inputs. The variable v could follow either a binomial distribution for binary input or common Gaussian distribution for continuous value. The learning process of Denoising Auto-Encoder is identical to that of conventional Auto-Encoder.

3.3.3 Convolutional Auto-Encoder

The elements in both Auto-Encoder and Denoising Auto-Encoder are fully connected. Different from these two models, Convolutional Auto-Encoder (CAE) has a constrain on the connections, i.e., the weights between two layers are shared in form of convolution. As is shown, such a convolution operation could preserve the spatial and temporal locality. Thus CNN has the advantage of time-shift invariance. Due to the convolutional and pooling operations, the pretraining of CNN with stacked CAEs is different from the other two models. At the same time, the reconstruction of CAE is also different from formal forward and backward passes of CNN. For the convolutional encoder, the hidden feature maps of CAE can be calculated as follows.

$$\mathbf{h}_j = \phi(\mathbf{z}_j) = \phi(\sum_i pad(\mathbf{x}_i) * \mathbf{k}_{ij} + b_j)$$

where $\mathbf{x}_i$ represents the i-th element of input and $\mathbf{h}_j$ denotes the j-th filter map after convolution and activation. The function $pad(\cdot)$ will pad $\mathbf{x}_i$ with zeros from two ends, e.g., if the size of $\mathbf{k}_{ih}$ is n_2 then this function will pad each end of $\mathbf{x}_i$ with $n_2 - 1$ zeros. Other symbols have already been explained in previous sections. After that, $\mathbf{h}_j$ will be processed through a pooling layer. Analogous to conventional Auto-Encoder, the reconstruction of $\mathbf{x}_i$ should be built based on $\mathbf{h}_j$ in a reversed procedure of convolutional encoder. The convolutional decoder works in form of the equation below.

$$\mathbf{y}_i = \phi(\sum_j \mathbf{h}_j * reverse(\mathbf{k}_{ij}) + c_i)$$

where the $reverse(\cdot)$ has been mentioned before and c_i is the bias term for the reconstruction $\mathbf{y}_i$. For the sake of learning the parameters, we need to minimize the cost function of CAE: $E(\theta) = \sum_i \|\mathbf{x}_i - \mathbf{y}_i\|^2$. Similar to CNN, by applying the backpropagation algorithm, the gradient of each parameter could be obtained and then the parameters will be updated until final convergence of CAE.

Our MC-DCNN model could include multiple stages. Hence, to pretrain the deep architecture we require stacking several CAEs. Recall Fig. 3, the first stage of this model is pretrained based on CAE at the beginning and then its hidden representations will be fed into the subsequent CAE for the second stage as its input. For MC-DCNN with more stages, the unsupervised pretraining would be performed for these CAEs one by one following the greedy, layer-wise fashion. Once the unsupervised pretraining of all CAEs has been done, then the learnt parameters will be used to initialize our MC-DCNN model. After that, the supervised learning of CNN will be performed as shown in Section 3.1 and 3.2.

4 Experiments

We conduct experiments on real-world data sets from two application domains, and we will demonstrate: (1) The performance of our MC-DCNN via comparing with baselines on weakly labeled data (Section 4.1) and well aligned data (Section 4.2 and 4.3), respectively; (2) The evaluation of activation functions and pooling strategies (Section 4.4); (3) The visualization of learnt features (Section 4.5); (4) A brief discussion (Section 4.6).

To the best of our knowledge, there are many public time series data sets available, e.g., the UCR Suite [34]. 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 [35]. One exception is the Non-Invasive Fetal ECG data set. In the UCR Suite, there are two data sets related to Non-Invasive Fetal ECG, both of which were created by recording the heartbeat information. One corresponds to the heartbeat data from left thorax and the other corresponds to that from right. Then, we could combine these two single data sets and obtain a **2D** time series data set. Besides, we also choose two data sets that are collected from real-world applications. And we will describe each of these data sets in 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 [2] [10] with *1*-NN, respectively. Besides these two state-of-the-art methods, MLP is 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 much previous work have claimed that feature-based methods cannot achieve the accuracy as high as *1*-NN methods. Moreover, we do not choose the full DTW due to its expensive time consumption.

4.1 Activity Classification (Weakly Labeled Data)

4.1.1 Data Set

We use the weakly labeled PAMAP2 data set for activity classification[3] [7]. 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 more than a month for *1*-NN (DTW-5%) on this data set if we use all the 19 physical activities[4]. Hence, without loss of generality, currently, we only consider 4 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.

4.1.2 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 as shown in Algorithm 1 to extract subsequences from **3D** time series with different *sliding steps*. Table 1 shows the number of subsequences of each activity for each subject when sliding step is set to 8. 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

[2] Following the discoveries in [10], we set the optimal window constraint r as 5%.

[3] http://www.pamap.org/demo.html

[4] There are 240,000 and 60,000 samples in training and test sets, respectively. Each sample includes three components with the length of 256.

Table 1 The number of subsequences of each activity.

Subject	standing	walking	ascending stairs	descending stairs
1	2,683	2,750	1,922	1,798
2	3,166	4,035	2,104	1,838
3	2,535	3,598	1,235	1,814
4	3,057	3,960	2,023	1,722
5	2,735	3,972	1,722	1,527
6	3,013	3,183	1,598	1,345
7	3,187	4,183	2,141	1,389
Total	20,376	25,681	12,745	11,433

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 kinds of models: MC-DCNN(1)/MC-DCNN(1)-Pre, MC-DCNN(2)/MC-DCNN(2)-Pre, with respect to 1-stage and 2-stages non-pretrained/pretrained models.

4.1.3 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, 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 L as 256.

Feature-based models have an advantage over lazy classification models (e.g., k-NN) in efficiency. As shown in Fig. 5, 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.

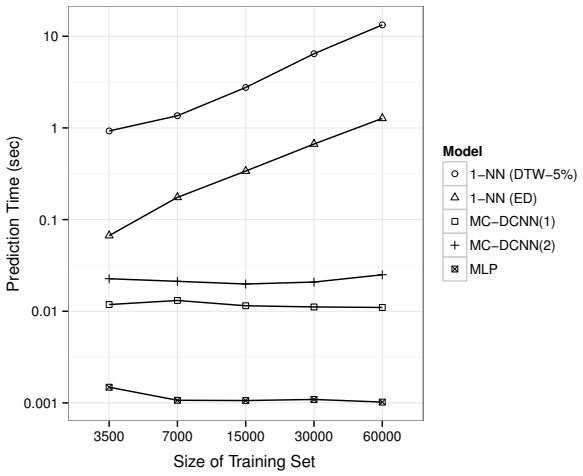


Fig. 5 Prediction time of each model on training sets with different size.

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 (data splits). From this figure we can see that for each subject our MC-DCNN model, particularly the pretrained 2-stages one, is either the most accuracy one or very close to the most accuracy one. Especially, for subject 3, the 2-stages MC-DCNNs lead to much better classification accuracy than other approaches. We suppose that 2-stages MC-DCNNs may learn higher level and more robust feature representations so that it has a good generalization. The average and standard deviation of accuracy is shown in Table 2. From the table we can see that our MC-DCNN leads to the highest average accuracy and the lowest standard deviation. Especially, the pretrained MC-DCNN is superior to others including the non-pretrained models, which demonstrates that such a unsupervised initialization could indeed improve the performance to some extent. By comparing to 1-stage MC-DCNN, the 2-stages models perform better at each *sliding step*. This is consistent to the statement of previous studies [36, 37] that deep neural networks trained by simple back-propagation work better than more shallow ones. On the other hand, by comparing MC-DCNN with traditional MLP, the superiority of MC-DCNN demonstrates that the feature learning process could indeed improve the performance of classification and the MC-DCNN model can learn good internal representations.

4.2 Congestive Heart Failure Detection (Well Aligned Data)

4.2.1 Data Set

The well aligned BIDMC data set was downloaded from Congestive Heart Failure database[5] [16]. 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, which was recorded by medical instruments. Table 3 shows the number of subsequences of each type. In this experiment, we consider four types of heartbeats to evaluate all the models: ‘N’, ‘V’, ‘S’, ‘r’.

Table 3 The number of subsequences of each heartbeat type.

Type	N	V	S	r	+	Q	E
Number	31,563	28,166	5,314	10,353	258	293	5

4.2.2 Experiment Setup

We still normalize each univariate of **2D** time series as $\frac{x-\mu}{\sigma}$, where μ and σ are mean and standard deviation of time series. Different from weakly data, we extract subsequences

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

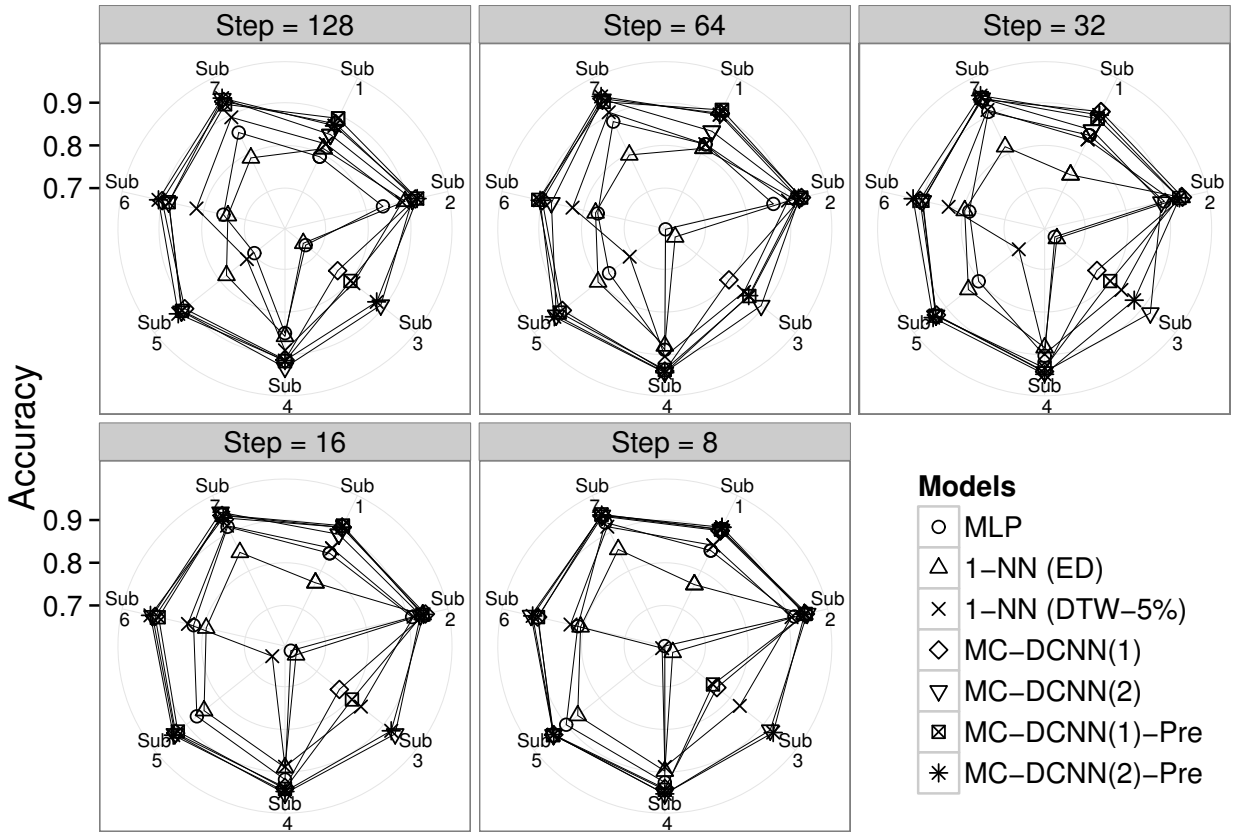


Fig. 6 Classification accuracy on each subject with different *sliding steps*.

Table 2 Average and standard deviation of accuracy at different *sliding step*. **Bold** numbers represent the best results.

Method	Step=128	Step=64	Step=32	Step=16	Step=8
1-NN (ED)	79.05 (0.076)	80.25 (0.089)	80.74 (0.094)	81.74 (0.096)	82.28 (0.103)
1-NN (DTW-5%)	83.46 (0.063)	84.51 (0.070)	84.44 (0.080)	84.16 (0.094)	83.61 (0.104)
MLP	77.89 (0.076)	80.09 (0.098)	82.49 (0.096)	84.34 (0.104)	84.83 (0.115)
MC-DCNN(1)	88.73 (0.057)	90.38 (0.050)	90.28 (0.063)	90.75 (0.062)	90.53 (0.065)
MC-DCNN(1)-Pre	89.37 (0.044)	91.47 (0.027)	90.38 (0.049)	91.11 (0.048)	90.55 (0.071)
MC-DCNN(2)	90.34 (0.031)	91.00 (0.033)	91.14 (0.031)	93.15 (0.019)	93.36 (0.015)
MC-DCNN(2)-Pre	**90.94 (0.025)**	**91.63 (0.032)**	**92.27 (0.028)**	**93.22 (0.011)**	**93.43 (0.013)**

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 [15], we mix all data of 15 subjects and randomly split it into 10 folds to perform 10-folds cross validation. Because as Ref. [15] 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. Similar to the previous experiment, we also evaluate these models to glance the impact of depths and pretraining on this data set. Moreover, to determine the epochs, we separate one third of training data as validation set. As shown in Fig. 8, 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.

4.2.3 Experimental Results

We show the classification accuracy of each model on BIDMC data set in Fig. 7. The accuracies of 1-stage and 2-stages non-pretrained MC-DCNN models are 94.67% and 94.65%. After we pretrained the MC-DCNN models, the MC-DCNN(1)-Pre and MC-DCNN(2)-Pre models improve the classification accuracies to 95.04% and 95.35%. All these MC-DCNN models have better performance than other baseline models, i.e., *1*-NN(ED) (93.64%), *1*-NN(DTW-5%) (92.90%) and MLP (94.22%). It is also noted that all of these learning-based methods including MLP perform better than distance-based methods. We consider that it may be due to the precise alignment of each heartbeat signal. For the well aligned data, these learning-based methods could

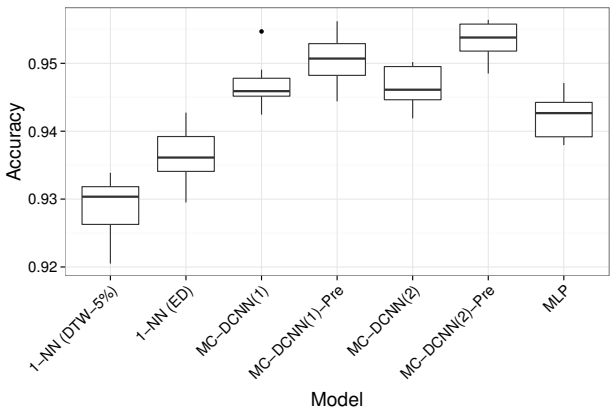


Fig. 7 The box-and-whisker plot of classification accuracy on BIDMC.

capture the potential important information and obtain good feature representations. Still, MC-DCNN is superior to MLP, which also demonstrates that the feature learning is beneficial for improving the classification accuracy. The experimental result proves that the pretraining process improve the performance indeed. We do not report the prediction time of each model on BIDMC data set. Since, the result is similar to Fig. 5 and it also supports that feature-based models have an advantage over lazy classification methods such as *k*-NN in efficiency.

4.3 Classification of Non-Invasive Fetal ECG (Well Aligned Data)

4.3.1 Data Set

Another well aligned data set is Non-Invasive Fetal ECG. For simplicity, we name this data set as NIFE (**N**on-**I**nvasive **F**etal **E**CG). Specifically, we combined two univariate time series data sets and each of them corresponds to the record of the ECG from left and right thorax. Hence, in this data set, each type of non-invasive fetal ECG corresponds to a **2D** time series. There are 42 types of non-invasive fetal ECG in the data set and the length of each univariate time series is 750. The size of the training set is 1,800 and there are 1,965 samples that can be used to test. Different from previous two experiments, we do not need normalization for this data set. Since the creators of this data set had processed for that and they also separated the training and test data in the randomized manner. That's the reason why we do not consider resampling the data for further cross validation. Another reason is many researchers in time series classification directly evaluated their models with the training and test data but not used the cross validation strategy. Hence, we follow them and do not apply cross validation here.

4.3.2 Experimental Results

We show the classification accuracy of each model on NIFE data set in Table. 4.

Table 4 Classification accuracy of each model on NIFE.

Method	Precision
1-NN (DTW-5%)	88.19
MLP	95.32
1-NN (ED)	89.62
MC-DCNN(1)	95.93
MC-DCNN(1)-Pre	96.03
MC-DCNN(2)	96.13
MC-DCNN(2)-Pre	96.23

This result is analogous to that of the previous BIDMC data set. All of the learning-based methods achieve higher accuracy than that of distance-based methods. As we said, the reason of such result may be because the ECG data is aligned precisely. For such well aligned data, these three learning-based methods may capture the potential important information and acquire good feature representations. Hence, the learning-based methods perform better than distance-based ones. The MC-DCNN models including the pretrained ones are superior to MLP, which proves that feature learning can improves the classification accuracy. Moreover, the pretrained 2-stages MC-DCNN outperforms other models and achieves the highest accuracy, which not only proves that 2-stages MC-DCNN is superior to 1-stage model to some extent but also demonstrates the pretraining can improve the classification accuracy indeed. Nonetheless, the improvement of MC-DCNN models is not that remarkable, especially for MLP, which proves our assumption, i.e., with precise alignment time series data (that means without the time-shift involved), even the traditional MLP could obtain good feature representations of data and achieve good classification accuracy.

4.4 Evaluation of Activation functions and Pooling strategies

We treat the traditional *sigmoid*(·) function and *average* pooling as a default choice for previous experiments. To further evaluate the effect of activation functions and pooling strategies, in the following, we conduct several experiments on PAMAP2 data set. Specifically, we focus on testing the rate of convergence of the MC-DCNN models integrated with different activation functions and pooling strategies.

For a better illustration of convergence, we select one subject from PAMAP2 and show the rate of convergence

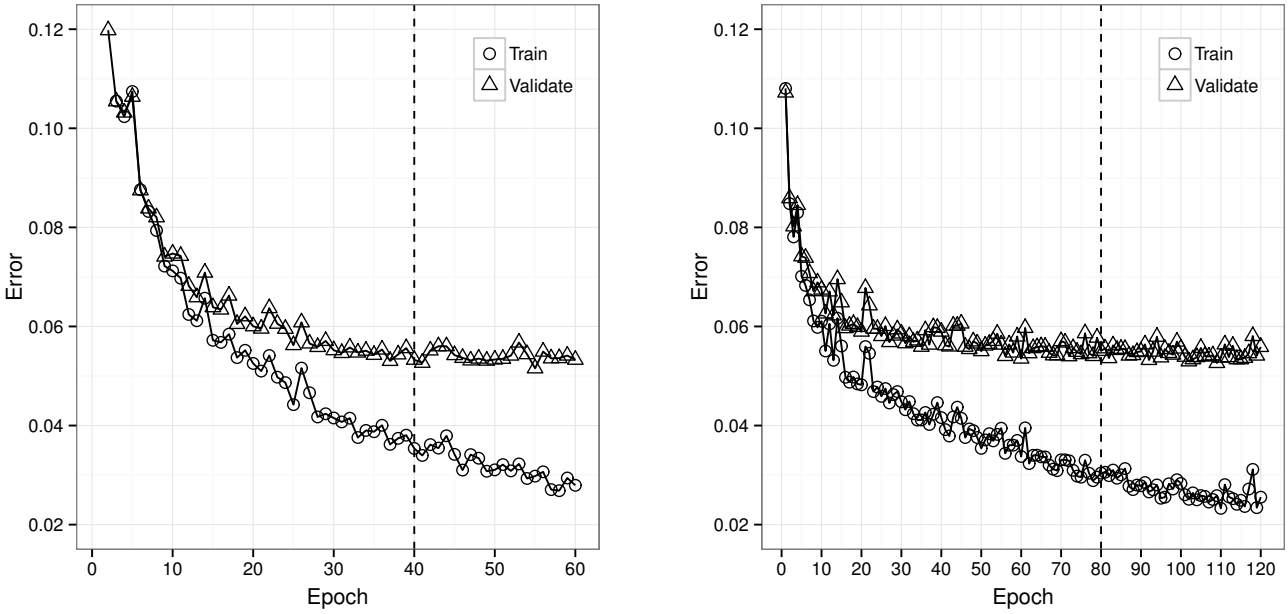


Fig. 8 Test error on validation set: Left) 1-stage MC-DCNN model; Right) 2-stages MC-DCNN model. The vertical dashed line is the determined epoch.

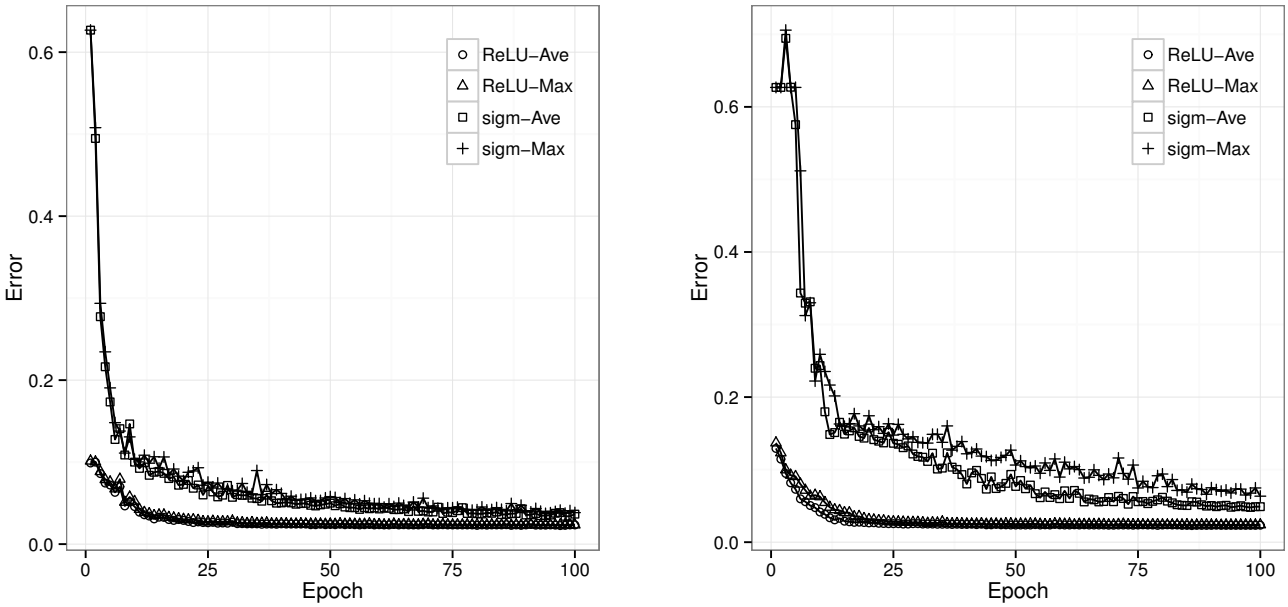


Fig. 9 Training error of 1-stage (*left*) and 2-stages (*right*) MC-DCNNs with *sigmoid*(·), ReLU activation functions and *average*, *max* pooling strategies.

of each model in Fig. 9. We consider two activation functions (i.e., ReLU and *sigmoid*(·)) and two sorts of pooling strategies (*max* and *average* poolings) here. As the epoch grows from 1 to 100, for each setting of activation function and pooling strategy, the training error decreases with different speed. From this figure, we can see that ReLU is superior to *sigmoid*(·) function since the training error of ReLU converges faster than that of *sigmoid*(·) for both 1-stage and 2-stages MC-DCNN models. For the pooling strategy, *average* pooling converges a little faster than *max* pooling but the difference is subtle. By the consideration of generalization ability that mentioned in previous studies, we believe that *max* pooling is a good choice for our study. Hence, we combine ReLU and *max* pooling in our MC-DCNN models.

4.5 Visualization

To visualize the learnt features of MC-DCNN model, we trained a two-stages MC-DCNN model on the PAMAP2 data set. Both of the first and the second filter layers contain 20

kernels in this model. We illustrate the learnt features of corresponding stage in Fig. 10 and Fig. 11, respectively.

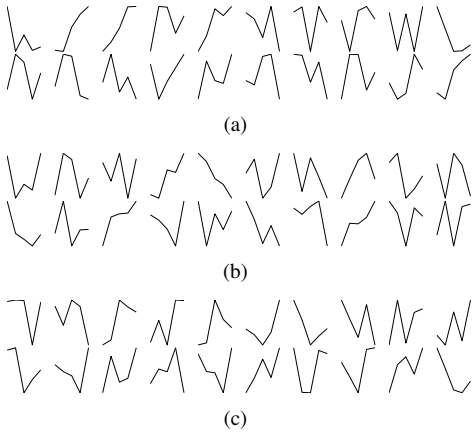


Fig. 10 Visualization of the first filter layer in the model trained on PAMAP2 data set. a) Channel one 20 filter weights with the length of 5, b) and c) represent the filter weights in the other two channels, respectively.

For the first filter layer, as shown in Fig. 10, each channel of MC-DCNN model contains twenty filters with the length of 5. Moreover, the features of each channel are learnt independently. From the figure, we can see that different local patterns (shapes) of time series are captured automatically by our model. In Fig. 11, we show the learnt features of the second filter layer in 20 groups of filter planes, each of which connects to all 20 second filter layer feature maps. Thus, each channel contains 20×20 filter weights in total, and over completed features have been learnt independently. Through the final MLP layers, the MC-DCNN model could combine the extracted features from each channel and determine the importance of each feature for different classes by the supervised learning. According to the experimental results, our MC-DCNN, a kind of feature learning method, can improve the accuracy of classification.

4.6 Discussion

We first discuss the advantages of our method intuitively. In MC-DCNN, the multi-channels can learn feature representation for each univariate time series automatically and individually. Then, the traditional MLP is used to combine these features and obtain a better representation for each class. According to the experimental results, such feature learning and feature combining approaches improve the classification performance for multivariate time series. We believe that our model can obtain better feature representations from each channel for each univariate time series and also the final feature combining can further learn the weights for each class, which makes our model can distinguish different patterns of the different underlying phenomena effectively and improves the performance finally.

In the following, we discuss the limitations of this study, and we believe that the discussions will lead to many future work. First, although we conducted these experiments with different parameter settings, there are still many other parameters in our MC-DCNN model. According to the suggestions of previous studies, some parameters were set to constants, while this may not be the optimal choice for our problem; Second, it is time consuming to train the neural networks since we did not utilize the parallel techniques (e.g., speedup by GPU) but implemented all the models in Matlab. This is also one of the reasons why we only constructed at most 2-stages MC-DCNN in the experiments. Hence, in the future work, we plan to study and extend other deep learning models for multivariate time series classification on more data sets and parameter settings.

5 Related Work

We first briefly review previous studies of time series classification methods. Then, we summarize the existing research interest on feature learning by deep neural networks and introduce the related pretraining methods at the end of this section. Many time series classification methods have been proposed based on different sequence distance measurements. Among the previous work, some researchers claimed that *1*-NN combined DTW is the current state of the art [9, 10]. However, the biggest weakness of *1*-NN with DTW model is its expensive computation [10]. To overcome this drawback, some of the researchers explored to speed up the computation of distance measure (e.g., DTW) in certain methods (e.g., with boundary conditions) [10]. While some of other researchers tried to reduce the computation of *1*-NN by constructing data dictionary [10, 15, 17, 38]. When the data set grows large, all these approaches improve the performance significantly in contrast to simple *1*-NN with DTW. Though many feature-based models have been explored for time series classification [6, 39], most of previous work 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 for recent years [11]. The most successful feature learning framework is deep neural networks, which build hierarchical

representations from raw data [12, 13, 40]. Particularly, as a supervised feature learning model, deep convolutional neural networks achieve remarkable successes in many tasks such as digit and object recognition [29], which motivates us to investigate the deep learning in time series field. In current literature, there are few studies on time series classification using feature learning and deep neural networks. Ref. [41] explored an unsupervised feature learning method with convolutional deep belief networks for audio classification, but in frequency domain rather than in time domain. Ref. [42] proposed a time-delay neural networks (TDNN) for phoneme recognition. The TDNN can be considered as a simplified model of CNN, since it only contains one or two tied connection hidden layers but does not perform pooling like traditional CNN, which makes it does not have a good shift invariant ability as well as CNN. Ref. [5] adopted a special 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.

One challenge of neural networks especially for deep architectures is how to avoid bad local minima during the learning process. To alleviate this issue, a better initialization of weights in neural networks is needed, which can further improve classification performance [31, 32]. For CNN, a greedy layer-wise unsupervised initialization named stacked Convolutional Auto-Encoder (CAE) can be used to pretrain the networks, which has been shown the effectiveness to improve the classification performance [33]. To the best of our knowledge, none of existing studies on time series classification has considered the supervised feature learning from raw data and also pretraining of networks. Hence, in this paper, we explore a MC-DCNN model for multivariate time series classification and intend to investigate this problem from feature learning view.

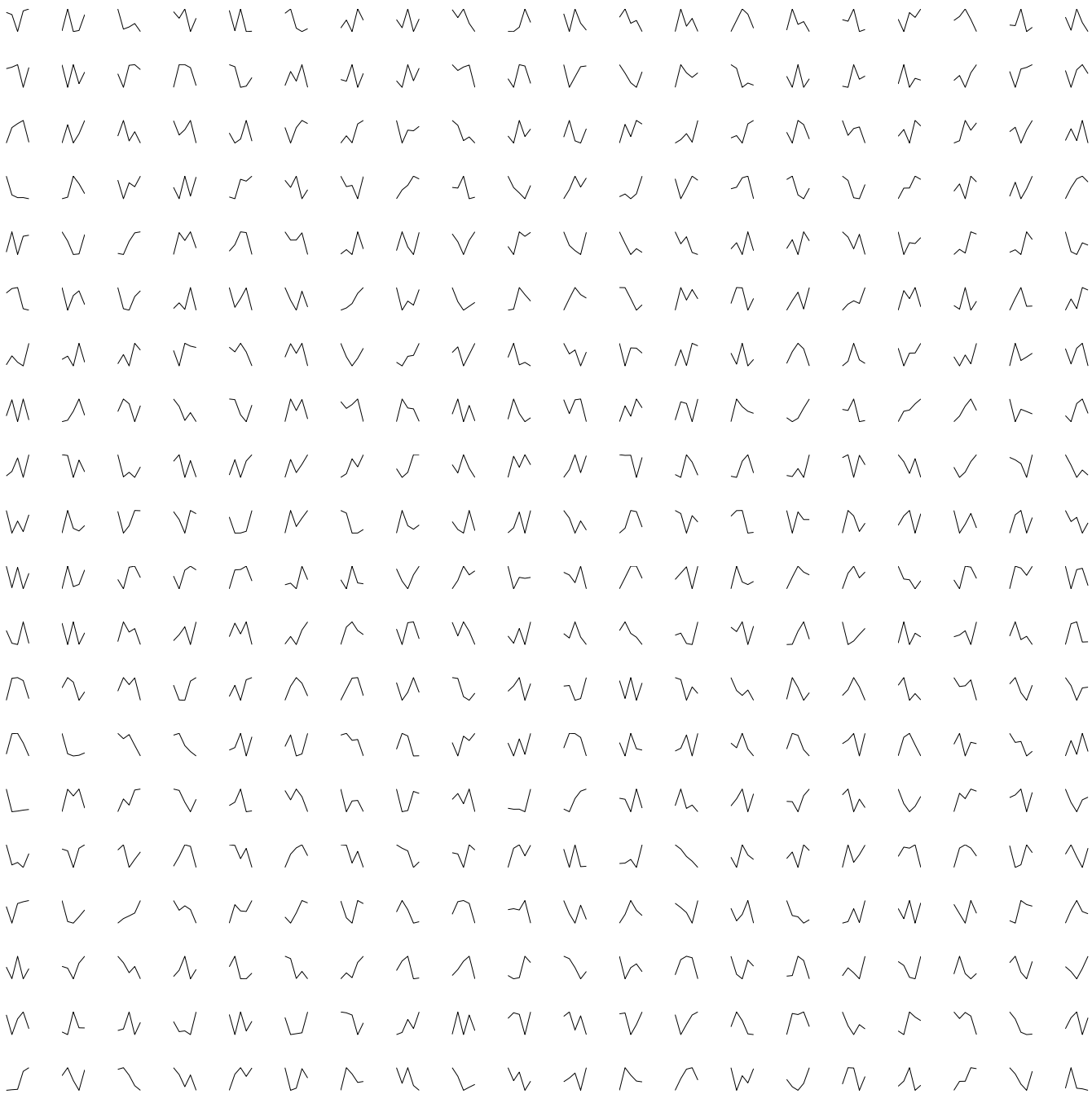
6 Conclusion

In this paper, we developed a novel deep learning framework (MC-DCNN) to classify multivariate time series. Through learning features from individual univariate time series in each channel automatically, this model then combines the outputs of all channels as feature representation at final layer. After that, a traditional MLP concatenated to the final layer of feature representation performs the classification. Meanwhile, we applied an unsupervised initialization to pretrain CNN and proposed the pretrained version of MC-DCNN model. Finally, extensive experimental results on several real-world data sets revealed that the MC-DCNN model indeed outperformed the competing baseline methods, and the improvement of accuracy on weakly labeled data set is significant. We found that the pretrained models outperform the non-pretrained ones, which shows the effectiveness of pretraining to improve the classification performance. We also observed that 2-stages MC-DCNN is superior to 1-stage model to some extent, which provides the evidence that deeper architecture could learn more robust high-level features for improving the classification. We hope that this study could lead to more future work.

Acknowledgements 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 Natural Science Foundation of China (Grant No. 61403358) and the Fundamental Research Funds for the Central Universities of China (Grants No. WK2350000001 and WK0110000042).

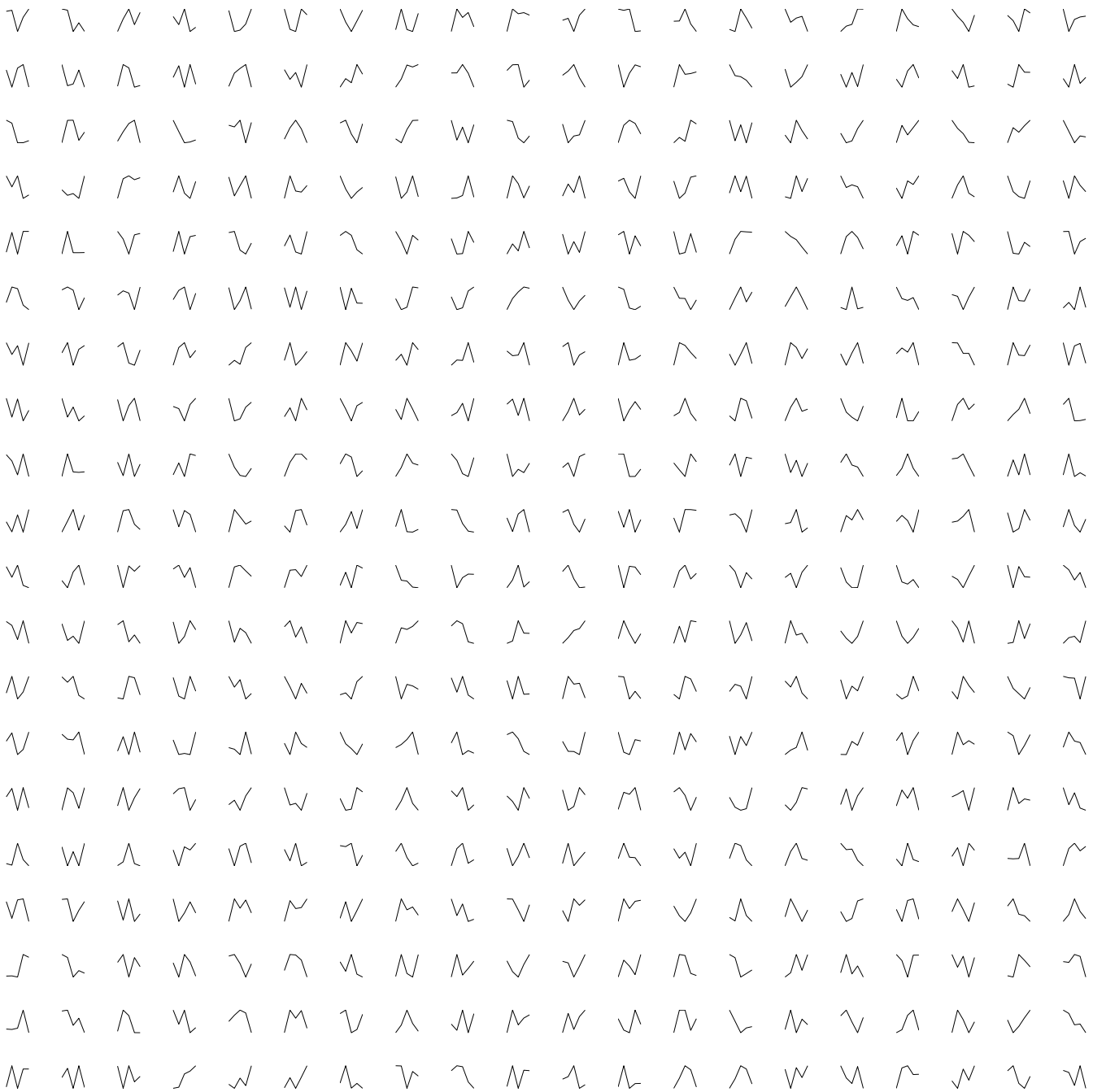
References

1. Xing Z, Pei J, Keogh E. A brief survey on sequence classification. ACM SIGKDD Explorations Newsletter, 2010, 12(1): 40–48
2. Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E. Querying and mining of time series data: experimental comparison of representations and distance measures. Proceedings of the VLDB Endowment, 2008, 1(2): 1542–1552
3. Orsenigo C, Vercellis C. Combining discrete svm and fixed cardinality warping distances for multivariate time series classification. Pattern Recognition, 2010, 43(11): 3787–3794
4. Batal I, Sacchi L, Bellazzi R, Hauskrecht M. Multivariate time series classification with temporal abstractions. In: FLAIRS Conference. 2009
5. Haselsteiner E, Pfurtscheller G. Using time-dependent neural networks for EEG classification. Rehabilitation Engineering, IEEE Transactions on, 2000, 8(4): 457–463
6. Kampouraki A, Manis G, Nikou C. Heartbeat time series classification with support vector machines. Information Technology in Biomedicine, IEEE Transactions on, 2009, 13(4): 512–518
7. Reiss A, Stricker D. Introducing a modular activity monitoring system. In: Engineering in Medicine and Biology Society,EMBC, 2011 Annual International Conference of the IEEE. 2011, 5621–5624
8. 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
9. 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. Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12, 2012, 262
10. Xi X, Keogh E J, Shelton C R, Wei L, Ratanamahatana C A. Fast time series classification using numerosity reduction. In: International



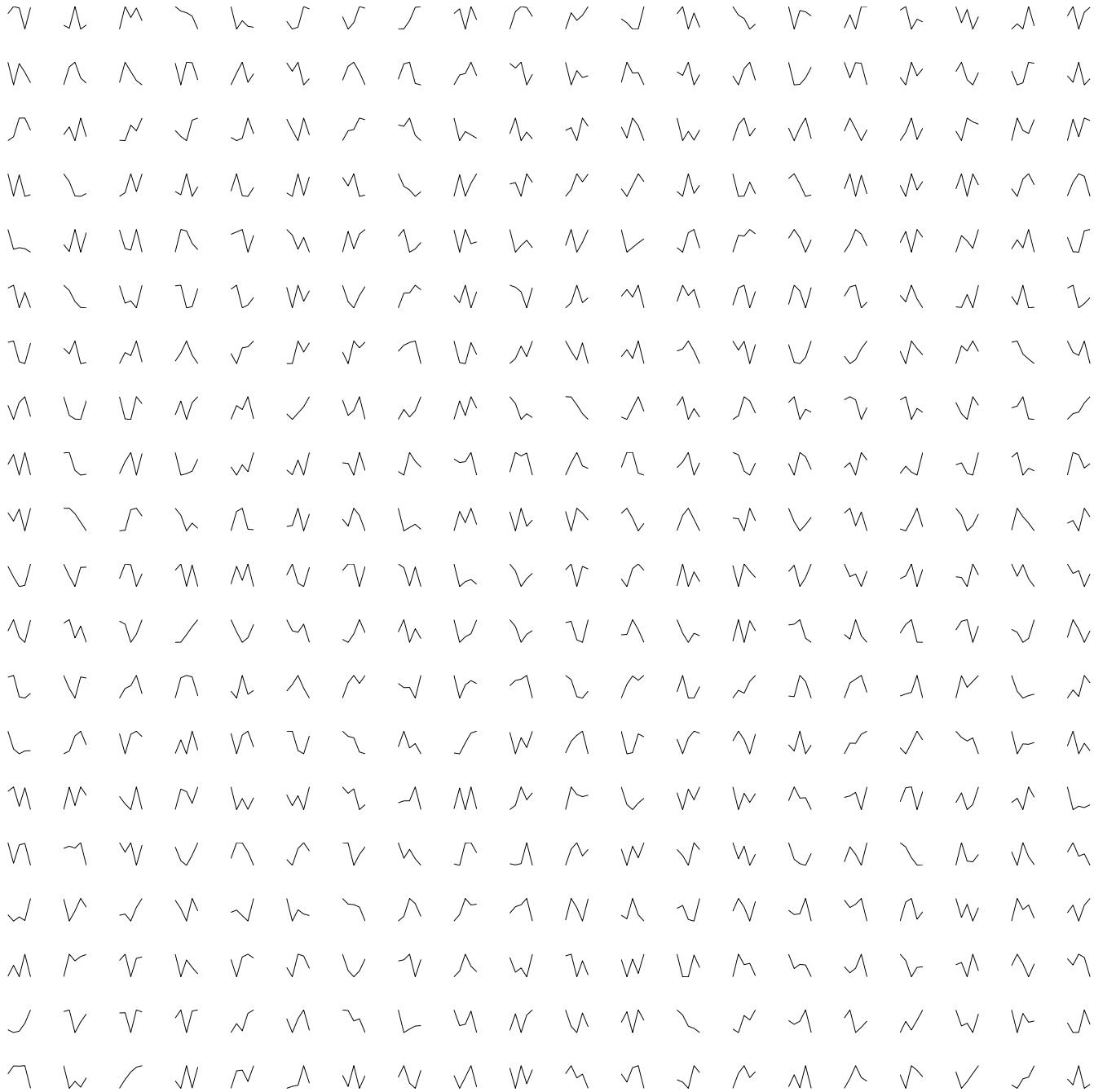
(a)

Fig. 11 Visualization of the second filter layer in the model trained on PAMAP2 data set. a) Channel one 20 x 20 filter weights with the length of 5, b) and c) represent the filter weights in the other two channels, respectively.



(b)

Fig. 11 Visualization of the second filter layer in the model trained on PAMAP2 data set. a) Channel one 20 x 20 filter weights with the length of 5, b) and c) represent the filter weights in the other two channels, respectively.

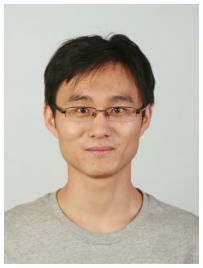


(c)

Fig. 11 Visualization of the second filter layer in the model trained on PAMAP2 data set. a) Channel one 20 x 20 filter weights with the length of 5, b) and c) represent the filter weights in the other two channels, respectively.

Conference on Machine Learning. 2006, 1033–1040

11. Bengio Y, Courville A, Vincent P. Representation learning: A review and new perspectives. arXiv preprint arXiv:1206.5538, 2012
12. LeCun Y, Bengio Y. Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 1995, 3361
13. LeCun Y, Kavukcuoglu K, Farabet C. Convolutional networks and applications in vision. In: Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on. 2010, 253–256
14. Zheng Y, Liu Q, Chen E, Ge Y, Zhao J. Time series classification using multi-channels deep convolutional neural networks. In: Li F, Li G, Hwang S w, Yao B, Zhang Z, eds, Web-Age Information Management, volume 8485 of *Lecture Notes in Computer Science*, 298–310. Springer International Publishing, 2014
15. Hu B, Chen Y, Keogh E. Time Series Classification under More Realistic Assumptions. SIAM International Conference on Data Mining, 2013, 578
16. 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, 2000, 101(23): e215—-e220
17. 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. 2009, 947–956
18. Ratanamahatana C A, Keogh E. Making time-series classification more accurate using learned constraints. 2004
19. Ratanamahatana C A, Keogh E. Three myths about dynamic time warping data mining. In: Proceedings of SIAM International Conference on Data Mining (SDMâĂŹ05). 2005, 506–510
20. Yu D, Yu X, Hu Q, Liu J, Wu A. Dynamic time warping constraint learning for large margin nearest neighbor classification. Information Sciences, 2011, 181(13): 2787–2796
21. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86(11): 2278–2324
22. Simard P Y, Steinkraus D, Platt J C. Best practices for convolutional neural networks applied to visual document analysis. In: Proceedings of the seventh international conference on document analysis and recognition. 2003, 958–962
23. Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines. In: Proc. 27th International Conference on Machine Learning, number 3. 2010, 807–814
24. Zeiler M D, Ranzato M, Monga R, Mao M, Yang K, Le Q, Nguyen P, Senior A, Vanhoucke V, Dean J, others . On rectified linear units for speech processing. In: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. 2013, 3517–3521
25. Scherer D, Müller A, Behnke S. Evaluation of pooling operations in convolutional architectures for object recognition. In: Artificial Neural Networks–ICANN 2010, 92–101. Springer, 2010
26. Nagi J, Ducatelle F, Di Caro G A, Ciresan D, Meier U, Giusti A, Nagi F, Schmidhuber J, Gambardella L M. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In: Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on. 2011, 342–347
27. LeCun Y, Bottou L, Orr G B, Müller K R. Efficient backprop. In: Neural networks: Tricks of the trade, 9–50. Springer, 1998
28. Bouvrie J. Notes on convolutional neural networks. 2006
29. Krizhevsky A, Sutskever I, Hinton G. Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25. 2012, 1106–1114
30. Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013, 2013, 28
31. Erhan D, Bengio Y, Courville A, Manzagol P A, Vincent P, Bengio S. Why does unsupervised pre-training help deep learning? The Journal of Machine Learning Research, 2010, 11: 625–660
32. Hinton G E, Salakhutdinov R R. Reducing the dimensionality of data with neural networks. Science, 2006, 313(5786): 504–507
33. Masci J, Meier U, Cireşan D, Schmidhuber J. Stacked convolutional auto-encoders for hierarchical feature extraction. In: Artificial Neural Networks and Machine Learning–ICANN 2011, 52–59. Springer, 2011
34. Keogh E, Zhu Q, Hu B, Hao Y, Xi x , Wei L, Ratanamahatana C A. The UCR Time Series Classification/Clustering Homepage: www.cs.ucr.edu/˜eamonn/time_series_data/. 2011
35. Pinto N, Cox D D, DiCarlo J J. Why is real-world visual object recognition hard? PLoS computational biology, 2008, 4(1): e27
36. Cireşan D C, Meier U, Masci J, Gambardella L M, Schmidhuber J. Flexible, high performance convolutional neural networks for image classification. In: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two. 2011, 1237–1242
37. Cireşan D, Meier U, Masci J, Schmidhuber J. Multi-column deep neural network for traffic sign classification. Neural Networks, 2012, 32: 333–338
38. Lines J, Davis L M, Hills J, Bagnall A. A shapelet transform for time series classification. Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012, 289–297
39. Nanopoulos A, Alcock R O B, Manolopoulos Y. Feature-based classification of time-series data. Information processing and technology, 2001, 0056: 49–61
40. 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. 2009, 609–616
41. Lee H, Largman Y, Pham P, Ng A Y. Unsupervised Feature Learning for Audio Classification using Convolutional Deep Belief Networks. In: Advances in Neural Information Processing Systems 22, 1096–1104. 2009
42. Waibel A, Hanazawa T, Hinton G, Shikano K, Lang K J. Phoneme recognition using time-delay neural networks. Acoustics, Speech and Signal Processing, IEEE Transactions on, 1989, 37(3): 328–339



Yi Zheng received the B.E. degree in Computer Science and Technology in 2009 from Harbin Institute of Technology, Heilongjiang, P.R. China. He is currently a Ph.D. student in the School of Computer Science and Technology at University of Science and Technology of China (USTC), P.R. China. His major research interests include time series data mining and deep learning. He has published several papers in refereed conference proceedings and journals, such as WAIM'14, PAKDD'15 and Nature Communications.

Qi Liu is an Associate Researcher in University of Science and Technology of China (USTC). He received his Ph.D. in Computer Science from USTC. His general area of research is data mining and knowledge discovery. He has published prolifically in refereed journals and conference proceedings, e.g., TKDE, TOIS, TKDD, TIST, SIGKDD, IJCAI, ICDM, and CIKM. He has served regularly in the program committees of a number of conferences, and is a reviewer for the leading academic journals in his fields. He is a member of ACM and IEEE. Dr. Liu is the recipient of the ICDM'11 Best Research Paper Award, the Special Prize of President Scholarship for Postgraduate Students, Chinese Academy of Sciences (CAS) and the Distinguished Doctoral Dissertation Award of CAS.

Enhong Chen received the PhD degree in computer science from USTC, the master's degree from the Hefei University of Technology and the BS degree from Anhui University. He is currently a professor and the vice dean of the School of Computer Science, the vice director of the National Engineering Laboratory for Speech and Language Information Processing of USTC, winner of the National Science Fund for Distinguished Young Scholars of China. His research interests include data mining and machine learning, social network analysis and recommender systems. He has published lots of papers on refereed journals and conferences, including IEEE TKDE, TMC, KDD, ICDM, NIPS, CIKM and Nature Communications. He was on program committees of numerous conferences including KDD, ICDM, SDM. He received the Best Application Paper Award on KDD'08 and Best Research Paper Award on ICDM'11. He is a senior member of the IEEE.

Dr. Yong Ge received his Ph.D. in Information Technology from Rutgers, The State University of New Jersey in 2013, the M.S. degree in Signal and Information Processing from the University of Science and Technology of China (USTC) in 2008, and the B.E. degree in Information Engineering from Xi'an Jiao Tong University in 2005. He is currently an Assistant Professor at the University of North Carolina at Charlotte. His research interests include data mining and business analytics. He has published prolifically in refereed journals and conference proceedings, such as IEEE TKDE, ACM TOIS, ACM TKDD, ACM TIST, ACM SIGKDD, SIAM SDM, IEEE ICDM, and ACM RecSys.

J. Leon Zhao is Head and Chair Professor in the Department of Information Systems, City University of Hong Kong. He was Interim Head and Eller Professor in Management Information Systems, University of Arizona. He holds Ph.D. from Haas School of Business, University of California at Berkeley. His research is on information technology and management, with a particular focus on collaboration and workflow technologies and business information services. He is director of Lab on Enterprise Process Innovation and Computing funded by NSF, RGC, SAP, and IBM among other sponsors. He received IBM Faculty Award in 2005 and was awarded Chang Jiang Scholar Chair Professorship at Tsinghua University in 2009.