Chapter

CNN Approaches for Time Series classification

Lamyaa Sadouk

Abstract

Time series classification is an important field in time series data-mining which have covered broad applications so far. Although it has attracted great interests during last decades, it remains a challenging task and falls short of efficiency due to the nature of its data: high dimensionality, large in data size and updating continuously. With the advent of deep learning, new methods have been developed, especially Convolutional Neural Network (CNN) models. In this paper, we present a review of our time series CNN approaches including: (i) a data-level approach based on encoding time series into frequency-domain signals via the Stockwell transform, (ii) an algorithm-level approach based on an adaptive convolutional layer filter that suits the time series in hand, and (iii) another algorithm-level approach adapted to time series classification tasks with limited annotated data, which is a global, fast and light-weight framework based on a transfer learning technique with a source learning task similar or different but related to the target learning task. These approaches are implemented on identifying human activities including normal movements of typical subjects and disorder-related movements such as stereotypical motor movements of autistic subjects. Experimental results show that our approaches improve performance of time series classification.

Keywords: time series, classification, convolutional neural networks, transfer learning

1. Introduction

Time series is a series of data points which are collected by recordinga set of observations chronologically. Examples of time series include speech, human activities, electrocardiogram (ECG), etc. Recently, time series classification has attracted great interests and initiated various researches. However, the nature of time series data, including the large size of data, the high dimensionality and the continuously updating scheme of time series, makes time series classification a more challenging task.

Time series classification is widely applied in different fields such as in astronomy [1] to classify the brightness of a target star, in medical science to diagnose cardiac disorders [2] or to recognize human activities [3, 4], and in computer science for speech recognition [5, 6]. To handle time series classification, several techniques were proposed, which can be aggregated into three categories: model based, distance based and feature based.

The first category of time series classification approaches consists of building a model for each class by fitting its parameters to that class. Examples of such approaches are the autoregressive (AR) model [7] and the hidden Markov model (HMM) [8] which are limited to stationary and symbolic non-stationary time series respectively.

The second category relies on developing distance functions to measure the similarity (or dissimilarity) between two time series and on selecting a good classifier, such as dynamic time warping (DTW) distance [9, 10]. But these approaches are computationally expensive.

The third category consists of extracting meaningful features from the time series. Examples of such approaches include the discrete Fourier transform (DFT) [11], the Short-time Fourier transform (STFT) [12], the discrete wavelet transform (DWT), principal component analysis (PCA), singular value decomposition (SVD), sparse coding [13], and shapelets [14].

Meanwhile, automatic feature-based approaches using deep learning models rely have been successfully applied to time series classification, classification problems, especially convolutional neural networks (CNNs) which are regarded as the most successful and commonly used deep learning model. In [5, 6], authors address the problematic of speech recognition whereby speech signals have similar patterns within different frequency band locations which convey a different meaning. A solution to this problem is to employ a limited weight sharing CNN [6] where weight sharing is limited only to local filters which are close to each other and which are pooled together in the subsampling layer. Another approach based on tiled CNN architecture with a pre-training stage (an unsupervised learning algorithm named topographic ICA) was proposed by [15], which showed its superiority over traditional CNN on small time series datasets. A tiled CNN [16] is a CNN which unties weights locally and uses a regular "tiled" pattern of tied weights that requires only that hidden units *k* steps away from each other to have tied weights. Another relevant CNN architecture for time series classification named multi-scale convolutional neural network (MCNN) [17] was introduced where each of the three transformed versions of the input (which will be discussed in Section 3.1) is fed into a branch i.e., a set of consecutive convolutional and pooling layers, resulting in three outputs which are concatenated and further fed into more convolutional and pooling layers, fully connected layers and a softmax layer to generate the final output. Training all parameters is done jointly using back-propagation. Another attempt to enhance time series classification was proposed in [18], which employs the same idea of multiple branches within the CNN architecture, except that the input is not a different transformed version of the time series signal fed into each branch, but rather a duplicate of the same time series signal fed into all the branches (three branches). However, different convolutional filter sizes are applied per branch in order to capture the multi-scale characteristics of the time series. Two other CNN proposals to time series classification were suggested in [19], namely fully convolutional networks (FCN) without subsampling layers, and residual networks (ResNet). FCNs [20] are defined as networks which have convolutional layers only and no fully-connected layers, whereas ResNet [21] is a type of specialized neural network that solves the "vanishing gradient" problem when having many layers within the network, by using residual blocks which take advantage of residual mapping to preserve inputs. By adding batch normalization layers into FCN and ResNet, and by replacing the fully connected layers with a global pooling layer in the FCN, these two deep learning models seem to yield comparable or better results than MCNN [17]. An ensemble method of deep learning networks named LSTM-FCN is proposed in [22] is proposed and consists of feeding the same time series input into two branches: an FCN and Long Short Term Recurrent Neural Network (LSTM) block [23], producing two outputs which are concatenated and then passed onto a softmax classification layer. Another attempt to helps the CNN converge faster and better to the minima was made by Guennec et al. [24] who propose to perform data-augmentation techniques (further described in Section 3.1) and pre-train each layer in an unsupervised manner (using an auto-encoder) using unlabeled training time series from different datasets. For multivariate time series, only few research papers based on CNNs were published (such as [3, 4, 25, 26]). Zheng et al. [25] proposed a multi-channels deep convolution neural network (MC-DCNN), each branch of which takes a single dimension of the multivariate time series as input and learns features individually. Then the MC-DCNN model combines the learnt features of each branch and feeds them into a fully connected layer to perform classification. And, to further improve the performance, authors also suggested to pre-train the MC-DCNN first by applying an unsupervised initialization via the convolutional auto-encoder method. Meanwhile, a different CNN architecture for multivariate time series classification was introduced in [3, 4, 26], which treats the 3-, 12-, and 9-variate time series inputs (in [3, 4, 26] respectively) as a 3-, 12-, and 9-channel inputs and convolves them as a whole instead of convolving each channel of the input separately as performed in [25]. Authors of this architecture argue that, by separating multivariate time series into univariate ones just as in [25], the interrelationship between different univariate time series may be lost and thus will not be mined/extracted.

In this paper, we aim at presenting a review on our CNN approaches for time series classification. Our review discusses our CNN contributions at the data-level and at the algorithm-level. Our paper is organized as follows. In Section 2, some preliminaries about time series are introduced. In Section 3 reviews existent data-level techniques are presented, our data-level technique is reviewed, and experiments as well as results of our technique are laid out. Section 4 describes our algorithm-level approaches for time series classification, with experiments conducted and results analyzed. Section 4.3.3 concludes our paper with future perspectives.

2. Preliminaries/nature of time series data

Univariate and multivariate time series data. Time series inputs can be categorized into: (i) Univariate Time series which have only a single variable observed at each time and thus resulting in one channel per time series input, and (ii) Multivariate Time series which have two or more variables observed at each time, ending up with multiple channels per time series input. Most time series analysis methods focus on univariate data as it is the simplest to work with. Multivariate time series analysis considers simultaneously multiple time series, which, in general is much more complicated than univariate time series analysis as it is harder to model and often many of the classical methods do not perform well.

Raw data or extracted signals. A raw time series is a series of data points indexed in time order i.e., a sequence of discrete-time data taken at successive equally spaced points in time. In time series classification tasks, some authors choose to evaluate the performance of their approaches using raw time series data taken from a specific field/domain while some others prefer to use public datasets in which the raw time series is already segmented and converted into a set of fixed-length signals. Indeed, several research papers using CNNs [17–19, 22, 24, 26, 27] build their experimental studies on the UCR time series classification archive [28] which consists of extracted short signals. Nonetheless, this benchmark is composed of relatively small datasets (with a small number of instances), which makes the CNN less efficient knowing that CNNs require large training sets for training. Furthermore, in most of the cases, fixed-length signals cannot be further encoded into new representations (which are discussed in Section 3.1), as opposed to raw time series. These issues have led authors of [3–6, 15, 25, 29, 30] to use raw time series data instead.

3. Data-level approach

Throughout this section, we show several approaches used in the literature to pre-process time series by re-framing them into new representations for a further CNN implementation. Indeed, a raw time series needs to be converted into a set of fixed-length vector or matrix inputs before being fed into the CNN. Then, we discuss our data-level approach (of previous works [3, 4]) based on the Stockwell transform method.

3.1 Background on pre-processing methods for CNN

3.1.1 Basic pre-processing method: the sliding window

Given a sequence of values for a time series dataset, values at multiple time steps can be grouped to form an input vector while the output corresponds to a specific label given to this vector (generally provided by an expert). The use of time steps to predict the label (e.g., the class) is called the *Sliding Window* method and is explained in the algorithm below. The width of the sliding window L can be varied to include more or less previous time steps depending on the specificity of the dataset and the user preference.

Algorithm 1. Sliding window's algorithm

```
1.  procedure $SlidingWindow(T, L, s)$
2.      $i = 0; n = 0;$                  // $n$ is the number of frames/windows.
3.      $F = [];$                        // $F$ is the set of extracted frames/windows.
4.      while $i + L \leq length(T)$ do  // $L$ is the length of sliding window
5.          $F[n] = T[i..(i + L - 1)];$  // $T$ is the original time series data/sequence.
6.          $i = i + s; n = n + 1;$      // $s$ is the step.
7.      end while
8.      return $F$;
9.  end procedure
```

3.1.2 Other pre-processing methods

Several research papers have focused mainly on applying some pre-processing to raw time series before being fed into the CNN. In this subsection, we present some important contributions which demonstrated that applying changes to the signals can further improve the CNN performance.

Several attempts have been made in order to encode raw time series as a matrix representation (e.g., 2D images) such as the Gramian Angular Field (GAF) [15], the Markov Transition Field (MTF) [15], Recurrence Plots (RP) [27], and stacked time series signals [29, 31], multivariate time series are treated as a 2D time-space input signals with one dimension denoting discrete time flows and the other corresponding to different channels of the multivariate time series.

Another type of data pre-processing based on applying transformation to data is performed in order to augment the data, thereby ensuring a better CNN training and thus a higher performance. For instance, the *window slicing* method [24] trains the CNN using slices of the time series input, then at test time classifies each slice of the test time series using CNN, and performs majority voting to output the predicted label. The *window warping* method [24] consists of warping a randomly selected slice of a time series by speeding it up or down, producing a transformed raw time series. Then, this latter is further converted into fixed-length input signals/instances via window slicing. Another attempt of augmenting time series is suggested in [3] where either small noise or smoothing is applied to the raw time series. Other transformations were also considered in [17] such as down-sampling to generate versions of a time series at different time scales, and spectral transformations in the frequency domain by adopting low frequency to remove noise from time series inputs.

Knowing that random noise and high-frequency perturbations present in the time series data can interfere tremendously with the learning process and that it is hard to capture useful features with the presence of noise in raw time series data, some works ([5, 30]) proposed to apply the Fast Fourier transform (FFT) and convert the raw time series into a set of frequency domain signals which serve as inputs for the CNN training process.

3.2 Stockwell transform

Instead of employing the FFT which is restricted to a predefined fixed window length, we choose to adopt the Stockwell transform (ST) as our preprocessing method for CNN training [3, 4]. In this section, the ST method is defined, its implementation on real world applications is detailed, and its experimental results are analyzed.

3.2.1 Methodology

The advantage of the ST over the FFT is its ability to adaptively capture spectral changes over time without windowing of data, resulting in a better time-frequency resolution for non-stationary signals [32]. To illustrate the ST method, let $h[l] = h(l{\bullet}T)$, $l = 0, 1, \ldots, N-1$, be the samples of the continuous signal $h(t)$, where T is the sampling interval (i.e., the sampling interval of our sensor or measuring device). The discrete Fourier transform (DFT) can be written as,

$$H[m] = \sum_{l=0}^{N-1} h[l] e^{\frac{-i2\pi ml}{N}} \qquad (1)$$

where m is the discrete frequency index $m = 0, 1, \ldots, N-1$.

The discrete Stockwell transform (DST) is given by,

$$S[k,n] = \sum_{m=0}^{N-1} W[m]H[m+n]e^{\frac{i2\pi mk}{N}} \tag{2}$$

where k is the index for time translation and n is the index for frequency shift. The function $W[m] = e^{\frac{-2\pi^2 m^2}{n^2}}$ is the Gaussian window in the frequency domain.

Given a N-length signal, the DST coefficients are computed using the following steps:

1. Apply an N-point DFT to calculate the Fourier spectrum of the signal $H[m]$;

2. Multiply $H[m+n]$ with the Gaussian window function $W[m] = e^{\frac{-2\pi^2 m^2}{n^2}}$

3. For each fixed frequency shift $n = 0, 1, \cdots, \tau - 1$ (where τ is the number of frequency steps desired), apply an N-point inverse DFT to $W[m]H[m + n]$ in order to calculate the DST coefficients $S[k,n]$, where $k = 0, 1, \ldots, N-1$;

Note that there is a DST coefficient calculated for every pair of$\langle k, n \rangle$ in the time-frequency domain. Therefore, the result of the DST is a complex matrix of size $\tau \times N$ where the rows represent the frequencies for every frequency shift index n and the columns are the time values of every time translation index k i.e., each column is the "local spectrum" for that point in time. This results in N instances, each instance being represented as a $\tau \times 1 \times 1$ matrix. If the time series is multivariate with D channels, D DST matrices will be generated, and each instance is represented as a $\tau \times 1 \times D$ matrix.

3.2.2 Experiments

3.2.2.1 Stereotypical motor movement (SMM) recognition task [4]

A SMM is defined as a repetitive movement which is regarded as one of the most apparent and relevant atypical behaviors present within children on the Autism Spectrum. Thus, detecting SMM behaviors can play a major role in the screening and therapy of ASD, thus potentially improving the lives of children in the spectrum.

Dataset. The SMM dataset used for training the CNN is derived from [33] and consists of raw time series of acceleration signals collected by three-axis wireless accelerometers (located at the torso, left wrist and right wrist) from six atypical (e.g., autistic) subjects in a longitudinal study. Activities including SMMs (body rocking, hand flapping, or simultaneous body rocking and hand flapping) and non-SMMs were engaged by subjects and were labeled (annotated) by an expert as SMM or non-SMM. Two to three sessions (9 to 39-min long) were recorded per participant, except subject 6 who was observed only once in Study 2.

Pre-processing. The data collection called "Study1" and "Study2" were recorded at a sampling frequency of 60 and 90 Hz respectively. So, to equalize the data, the 60 Hz signals are resampled and interpolated to 90 Hz. Next, data of both sensors go through a high pass filter with a cut-off frequency of 0.1 Hz in order to get rid of noise.

Afterwards, data is turned into fixed-length vector samples either in time-domain (using the sliding window) or in frequency-domain (using ST). Time-domain samples are obtained by segmenting raw data using a one second window (e.g., $L = 90$, see algorithm1) and 88.9% overlap between consecutive data segments (e.g. $s = 10$, see algorithm1), resulting in 90 time-point samples. And, knowing that three 3-axis acceleration signals are measured per accelerometer, an input sample will be a $90 \times 1 \times 9$ matrix with 9 denoting the number of channels ($9 = 3 accelerometers \times 3 coordinates$).

On the other hand, *frequency-domain* samples are obtained by deriving ST for every other 10th sample, and by selecting the proper best frequency range. Considering that 98% of the FFT amplitude of human activity frequencies is contained below 10 Hz, the ST frequencies are first chosen to be between 0 and 10 Hz, which yields bad CNN classification performance. And, after considering Goodwin's observation that almost all SMMs are contained within a frequency range of 1–3 Hz, we chose a new frequency range of 0–3 Hz which produced higher CNN classification performance. So, computing the ST generates multiple input samples (vectors of length 50), each containing the power of 50 frequencies ($\tau = 50$) in the range of 0–3 Hz. Thus, each extracted frequency-domain sample is a $50 \times 1 \times 9$ matrix.

CNN training. The purpose is to analyze intersession variability of different SMMs by training one CNN per domain (time or frequency domain) per subject per study. In other words, feature learning that is performed is specific to one domain (time and frequency), one subject i and one study j. The goal of this experiment is to build deep networks that are capable of recognizing SMMs across multiple sessions within the same atypical subject i. Training is conducted using k-fold cross-validation of an atypical subject i for a study j such that k is the number of sessions for which a participant was observed within each study, and every fold consists of data from a specific session. Time and frequency domain CNN architectures are composed of three and two sets of convolution, ReLU and pooling layers respectively with the number of filters set to {96, 192, 300} and {96,192} respectively, followed by a fully connected layer with 500 neurons. Training is performed for 10 to 40 epochs with the following hyper-parameters: a dropout of 0.5, a momentum of 0.9, a learning rate of 0.01, a weight decay of 0.0005, and a mini-batch size of 150.

3.2.2.2 Human activity recognition (HAR) task [3]

Dataset. The dataset used for HAR is the PUC dataset [34] which consists of 8 hours of human activities collected at a sampling frequency of 8 Hz by 4 tri-axial ADXL335 accelerometers located at the waist, left thigh, right ankle, and right arm. The activities are: sitting, standing, sitting down, standing up, and walking.

Pre-processing. The PUC data is further converted into time and frequency domain signals. In time-domain, a 1 s time window (e.g., $L = 8$) with 125 ms overlapping (e.g., $s = 1$) is employed to generate 8×1 time-domain samples. However, knowing that an 8×1 input matrix is not a vector long enough for training a CNN, signals are resampled from 8 to 50 using an antialiasing FIR low-pass filter and compensating for the delay introduced by the filter. The resultant time-domain input samples are $50 \times 1 \times 12$ matrix where 12 stands for the number of channels ($14 accelerometers \times 3 coordinates$). In frequency-domain, raw signals are resampled from 8 to 16 Hz; then, the ST is computed to obtain, for each input sample, the power of 50 frequencies in the range of 0–8 Hz, resulting in frequency-domain input samples of size $50 \times 1 \times 12$.

CNN training. In this experiment, one CNN is trained for each domain (time and frequency domain), with a 10-fold cross-validation. CNN architecture and parameters are set the same as in the SMM recognition task.

3.2.3 Results

Table 1 summarizes accuracy and F1-score results of CNN (in both time and frequency domains) for both the SMM recognition and HAR tasks. For SMM

	F1-scores SMM recognition												Accuracies HAR
	Study 1						Study 2						
	S1	S2	S3	S4	S5	S6	S1	S2	S3	S4	S5	Mean	
Time-domain CNN	91.23	76.76	84.95	93.38	86.41	95.11	95.97	75.67	60.17	91.68	82.55	84.90	**99.90**
Frequency-domain CNN	**96.54**	**78.41**	**93.62**	**96.46**	**95.74**	**98.58**	**96.07**	**95.27**	**85.03**	**98.03**	**93.88**	**93.42**	95.98

Table 1.
Performance rates of time-, and frequency-domain CNNs for the SMM recognition (in terms of F1-score) and Human Activity Recognition referred to as HAR (in terms of accuracy). Highest rates are in bold.

recognition, as opposed to other subjects, Subject 6 within Study 2 had only one session recorded; thus, no CNN was trained for this subject. We observe that all frequency-domain CNNs (of all subjects in all studies) perform better than time-domain CNNs by 8.52% (in terms of the mean F1-score). This suggests that ST eliminates all noisy information and thus helps the CNN capture meaningful features.

However, as opposed to these results, comparing results of time and frequency domain CNNs on the Human Activity Recognition (HAR) task demonstrates the efficiency of time over frequency by 3.92% in terms of accuracy (as shown in **Table 1**). These contradictory results can be explained by the difference in the chosen ST frequency range for SMM recognition and that of HAR. Indeed, in SMM recognition, the frequency range of the ST was carefully chosen to cover almost all SMMs (0–3 Hz), resulting in optimal frequency-domain samples (containing full and noise-free information) which produced better CNN parameters. Meanwhile, the ST frequency range for HAR (0–8 Hz) may be a short/small range which generated frequency-domain samples that may have lost relevant information. Indeed human activity frequencies fall between 0 and 20 Hz (with 98% of the FFT amplitude contained below 10 Hz). Thus, in order to train CNNs with frequency-domain signals, it is necessary to analyze raw time series to come up with the proper ST frequency range which covers all valuable information needed for the recognition task.

4. Algorithm-level approach

4.1 Background on convolutional neural networks

4.1.1 Definition

CNNs were developed with the idea of local connectivity. Each node is connected only to a local region in the input. The local connectivity is achieved by replacing the weighted sums from the neural network with convolutions. In each layer of the CNN, the input is convolved with the weight matrix (e.g., the filter) to create a feature map. As opposed to regular neural networks, all the values in the output feature map share the same weights so that all nodes in the output detect exactly the same pattern. The local connectivity and shared weights aspect of CNNs reduce the total number of learnable parameters, resulting in more efficient training and learning in each layer a weight matrix which is capable of capturing the necessary, translation-invariant features from the input.

4.1.2 CNN structure

The input to a convolutional layer is usually taken to be three-dimensional: the height, weight and number of channels. In the first layer this input is convolved with a set of M_1 three-dimensional filters applied over all the input channels. In our case, we consider a one-dimensional time series $x = (x_t)_{t=0}^{N-1}$. Given a classification task and a model with parameter values w, the task for a classifier is to output the predicted class $\hat{y}$ based on the input time series, $x(0)$, ..., $x(t)$.

The output feature map from the first convolutional layer is then given by convolving each filter w_h^1 for $h = 1, \ldots, M_1$ with the input:

$$a^1(i,h) = \left(w_h^1 * x\right)(i) = \sum_{j=-\infty}^{\infty} w_h^1(j)x(i-j) \tag{3}$$

where $w_h^1 \in R^{1\times k\times 1}$ and $a^1 \in R^{1\times N-k+1\times M_1}$, i is the index of the feature map at the second dimension ($i = 1, \ldots, N - k + 1$) and h is the index of the feature map at the third dimension ($h = 1, \ldots, M_1$). Note that since the number of input channels in this case is one, the weight matrix also has only one channel. Similar to the feedforward neural network, this output is then passed through the non-linearity $h(\cdot)$ to give $f^1 = h(a^1)$.

In each subsequent layer $l = 2, \ldots, L$, the input feature map $f^{1-1} \in R^{1\times N_{l-1}\times M_{l-1}}$, where $1 \times N_{l-1} \times M_{l-1}$ is the size of the output filter map from the previous convolution with $N_{l-1} = N_{l-2} - k + 1$, is convolved with a set of M_l filters $w_h^l \in R^{1\times k\times M_{l-1}}$, $h = 1, \ldots, M_l$, to create a feature map $a^l \in R^{1\times N_l\times M_l}$,

$$a^l(i,h) = \left(w_h^l * f^{l-1}\right)(i) = \sum_{j=-\infty}^{\infty} \sum_{m=1}^{M_{l-1}} w_h^1(j,m) f^{l-1}(i-j,m) \tag{4}$$

This output is then passed through the non-linearity to give $f^l = h\left(a^l\right)$. The filter size parameter k thus controls the receptive field of each output node. Without zero padding, in every layer the convolution output has width $N_l = N_{l-1} - k + 1$ for $l = 1, .., L$. Since all the elements in the feature map share the same weights this allows for features to be detected in a time-invariant manner, while at the same time it reduces the number of trainable parameters.

The output is then fed into a pooling layer (usually a max-pooling layer) which acts as a subsampling layer. The output map $p^l(h)$ of a feature map h is achieved by computing maximum values over nearby inputs of the feature map as follows:

$$p^l(i,h) = \max_{r\in R}\left(f^l(i \times T + r, h)\right) \tag{5}$$

where R is the pooling size, T is the pooling stride, and i is the index of the resultant feature map at the second dimension.

Multiple convolution, ReLU and pooling layers can be stacked on top of one another to form a deep CNN architecture. Then, the output of these layers is fed into a fully connected layer and an activation layer. The output of the network after L layers will thus be the matrix f^L. Depending on what we want our model to learn, the weights in the model are trained to minimize the error between the output from the network f^L and the true output we are interested in, which is often denoted as the objective function (loss function). For instance, a softmax layer can be applied on top, followed by the entropy cost function which is an objective function computed based on the true labels of training instances and probabilistic outputs of softmax function.

4.2 CNN with the adaptive convolutional filter approach

In previous CNN works, several attempts have been made to extract the most relevant/meaningful features using different CNN architectures. While works [17, 24] transformed the time series signals (by applying down-sampling, slicing, or warping) so as to help the convolutional filters (especially the 1st convolutional layer filters) capture entire peaks (i.e., whole peaks) and fluctuations within the signals, the work of [18] proposed to keep time series data unchanged and rather feed them into three branches, each having a different 1st convolutional filter size, in order to capture the whole fluctuations within signals. An alternative is to find an adaptive 1st convolutional layer filter which has the most optimal size and is able to capture most of entire peaks present in the input signals. By obtaining the most appropriate 1st convolutional layer filter, there will be no need to apply multiple branches with different 1st convolutional layer filter sizes, and no need to apply transformations such as down-sampling, slicing and warping, thus requiring less computational resources. The question of how to compute this adaptive 1st convolutional layer filter is addressed in [4]. In this section, we will discuss the approach based on the adaptive 1st convolutional layer filter. Next, to prove the efficiency of this/our approach, an application on SMM recognition is conducted and results are analyzed.

4.2.1 Methodology

In CNNs, multiple hyper-parameters are to be chosen carefully in order to retrieve the best classification rate, including model hyper-parameters which define the CNN architecture, and optimization hyper-parameters such as the loss function, learning rate, etc. Model Hyper-parameter values are generally chosen based on the literature and on the trial-and-error process (through running experiments with multiple values). A conventional approach is to start with a CNN architecture which has already been adopted in a similar domain to ours, and then update hyper-parameters by experimentation.

In our study, we focus on the convolutional layer filter (also known as "Receptive field"). Conventionally, the 1st convolutional layer filter has one of the following sizes: 3×3, 5×5, 7×7 and 10×10, where small filter sizes capture very fine details of the input while large ones leave out minute details in the input. After all, the goal is to set the receptive field size such that the filters detect complete features (e.g., entire variations) within an object (which could be edges, colors or textures within an image, or peaks within a signal). Choosing a small 1st layer filter is good for capturing a short variation (e.g., a signal peak) within a time series input signal, but may capture only a slice of this variation by convolving only part of it, thus failing to detect the whole fluctuation within the signal. Conversely, a relatively large filter may convolve multiple signal peaks at once and therefore no fluctuation is detected either. So, the choice of the proper 1st layer receptive field size is crucial for find good CNN features and for maximizing recognition performance. In that sense, we need 1st layer filter that suits the input signals and variations present within them. In other words, we need to find the best filter size which convolves most of the entire signal peak within the input signals. To this end, it is necessary to find the optimal length of all signal peaks present within input signals. To do so, we apply *sampling*, a statistical procedure that uses characteristics of the sample (i.e., sample peak lengths taken from randomly selected signals) to estimate the characteristics of the population (i.e., the optimal peak length of all signals). The sample statistic can be the mean, median or mode.

Given a population of signals with mean μ and n random values x (n being sufficiently large: $n \geq 30$) sampled from the population with replacement, the mean $E(x)$ is a point estimate of μ and $E(x) = \mu$ where $E(x) = \frac{1}{n}\sum_{i=1}^{n} x_i$. However, using the sample mean $E(x)$ as the optimal length of peaks within time series signals (in time- and frequency-domains) and as the size of the 1st convolutional layer filter gives poor CNN performance since it is influenced by outliers or extremes values (i.e., by signals peaks whose length are either too small and too big). In this case, we use the sample median.

For the sample median $M_e(x)$ to be a point estimate of the population median M_e (with a small bias), the distribution of the sample values x should be normally distributed. Nonetheless, the asymptotic distribution of the sample median in the classical definition is well-known to be normal for absolutely continuous distributions only and not for discrete distributions (such as time series). A solution to this problem is to employ the definition of the sample median based on mid-distribution functions [35] which proves that the sample median has an asymptotically normal distribution, meaning that $M_e(x) \approx M_e$. Then, computing the sample median of signal peak lengths will give us the optimal size of the 1st convolutional layer filter.

4.2.2 Experiments

Experiments are conducted on the SMM Recognition task. The dataset and experimental setup are the same as in Section 3.2.2. The inputs used are either time-domains acceleration signals of size $90 \times 1 \times 9$ for time-domain CNN training or frequency-domain signals of size $50 \times 1 \times 9$ for frequency-domain CNN training. The goal is to find the optimal size of the 1st convolutional layer filter for both time-domain and frequency-domain CNNs.

As explained in the methodology, the first step to determine the size of the 1st convolutional layer filter is to collect 30 random signals (for each of the time and frequency domain SMM signals) that contain at least one peak and to randomly pick 30 peaks from these signals. Histograms (a) and (b) of **Figure 1** represent frequency distributions of the 30 peak lengths for time and frequency domain signals respectively. Afterwards, the computed time and frequency domain medians (9 and 10 respectively) are applied as the optimal size of the 1st convolutional layer filter for the time and frequency domain CNNs respectively.

4.2.3 Results

In order to prove the efficiency of this adaptive 1st convolutional layer filter approach, we run experiments on different time and frequency domain CNN architectures by varying the size of the 1st convolutional layer filter between 7 and 11 across both architectures. Performance rates in terms of the F1-score metric are displayed in **Figure 2**. In time-domain, an increase in the size of the 1st convolutional layer filter from 7 ($\sim$ a time span of 0.078 s) to 9 ($\sim$ a time span of 0.1 s) results in an increase of 3.26%, while an increase of the filter size from 9 to 10 ($\sim$ a time span of 0.11 s) and 11 ($\sim$ a time span of 0.12 s) diminishes the performance of the network. Therefore, the most optimal size of the 1st convolutional filter is equal to the sample median of signal peak lengths, suggesting that 0.1 is the best time span of the 1st convolutional layer to retrieve the whole acceleration peaks and the best acceleration changes. Similarly, in frequency domain, the 1st convolutional layer kernel yielding the highest F1-score is the one with size 10, which is simply the sample median ($M_e(x) = 10$). Thus, these results confirm the

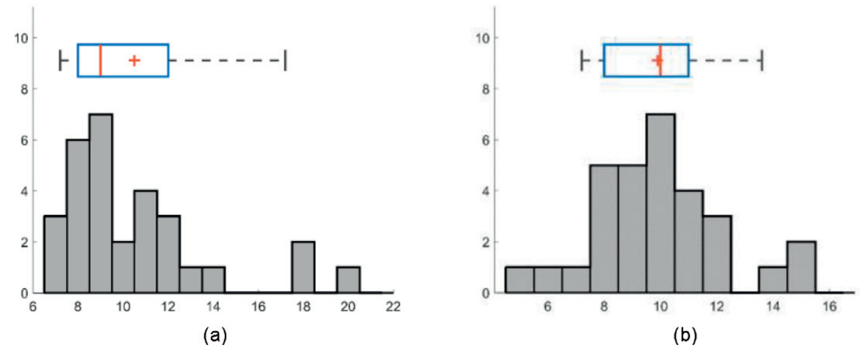

Figure 1.
(a) and (b) Histograms and box plots of the frequency distribution of 30 peak lengths present within 30 randomly selected time and frequency domain signals respectively.

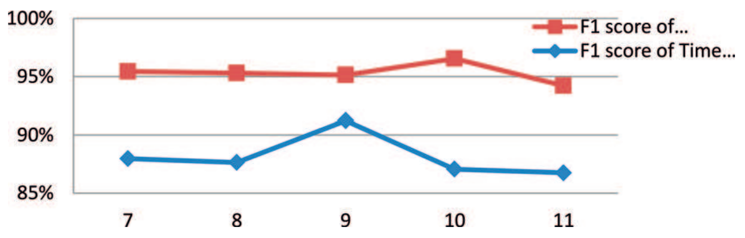

Figure 2.
Effect of the size of 1st convolutional layer kernel on SMM recognition performance.

	Study 1						Study 2				Mean
	S1	S2	S3	S4	S5	S6	S1	S2	S3	S4	S5	
CNN-Rad [30]	71	73	70	92	68	94	68	22	2	77	75	64.73
Time-domain CNN	91.23	76.76	84.95	93.38	86.41	95.11	95.97	75.67	60.17	91.68	82.55	84.90

Table 2.
Comparative results (F1-scores) between the CNN using the adaptive 1st convolutional filter approach and the CNN of Rad et al. [30].

superiority of the adaptive 1st convolutional layer filter approach for both time and frequency domain signals.

Furthermore, another way to show the efficiency of this adaptive 1st convolutional layer filter approach is to compare the performance of our time-domain CNN with the CNN of Rad et al. [30] which was trained on the same dataset as ours (in time-domain). **Table 2** displays F1-score results of CNNs trained per subject and per study using the optimal 1st convolutional layer filter size (denoted as "Time-domain CNN") and using the architecture of [30] (referred to as CNN-Rad). These results suggest that our time-domain CNN performs 20.17% higher in overall than the CNN of [30] and confirms the efficiency of the adaptive convolutional layer.

4.3 CNN approach for tasks with limited annotated data

CNNs have so far yielded outstanding performance in several time series applications. However, this deep learning technique is a data driven approach i.e., a supervised machine learning algorithm that requires excessive amount of labeled (e.g., annotated) data for proper training and for a good convergence of parameters. Although in recent years several labeled datasets have become available, some fields such as medicine experience a lack of annotated data as manually annotating a large set requires human expertise and is time consuming. For instance, labeling acceleration signals of autistic children as SMMs or non-SMMs requires knowledge of a specialist. The conventional approach to deal with this kind of problem is to perform data augmentation by applying transformations to the existing data, as shown in Section 3.1.2. Data augmentation achieves slightly better time series classification rates but still the CNN is prone to overfitting. In this section, we present another solution to this problem, a "knowledge transfer" framework which is a global, fast and light-weight framework that combines the transfer learning technique with an SVM classifier. Afterwards, this technique is further implemented on another type of SMM recognition task, which consists of recognizing SMMs across different atypical subjects rather than recognizing SMMs across multiple sessions within one subject (as performed in experiments of Sections 3.2.2 and 4.2.2).

4.3.1 Methodology

Transfer learning is a machine learning technique where a model trained on one task (a source domain) is re-purposed on a second related task (a target domain). Transfer learning is popular in deep learning, including Convolutional Neural Networks, given the enormous resources required to train deep learning models or the large and challenging datasets on which deep learning models are trained. For a CNN, given a source domain with a source learning task and a target domain with a target learning task (task of interest), transfer learning aims to improve learning of the target predictive function using the knowledge in the source domain which is the pre-trained CNN model containing features (parameters or weights) learned from the source domain task. This process works if these source domain features are general, meaningful and suitable to the target task. The pre-trained model can then be used as the starting point for a model on the target task. This may involve using all or parts of the pre-trained CNN model, depending on the modeling technique used. Accordingly, the questions that arise are: (i) which source learning task should be used for pre-training the CNN model given a target learning task, and (ii) which parts (e.g., learned features) of this model are common between the source and target learning tasks.

An answer to the first question is to propose two source learning tasks. One source learning task is chosen to be very close and similar to the target learning task. And, if this source learning task lacks annotated data, then another source learning task is introduced which is chosen to be different but related to the target learning task.

A solution to the second problematic is to assume that features shared across the source and target tasks correspond to low- and mid-level information (e.g., fine details or local patterns) contained within inputs of both tasks, whereas the unshared features are the high-level information (e.g., global patterns) contained within inputs. And, knowing that training a CNNs produces learned low-, mid- and high-level features located at (contained within) the first, intermediate and last hidden layers respectively, we therefore assume that the features shared between the source and target tasks are contained within the first and intermediate CNN layers while features distinguishing one task from the other are contained within the last CNN layer. For instance, in image classification, as the CNN learns low-level features (Gabor filters such as edges, corners) through the first hidden layers, mid-level features (squares, circles, etc.) through intermediates hidden layers, and high-level features (faces, text, etc.) through last hidden layers, scene recognition (source learning task) and object recognition (target learning task) will have the same first and intermediate layers' weights but different last layer weights. In time series, considering human activities where every activity is a combination of several basic continuous movements, with basic continuous movements corresponding to the smooth signals and the transitions or combinations among these movements causing the significant/salient changes of signal values, the purpose of the CNN will be to capture basic continuous movements through its low- and mid-level parameters (first and intermediate hidden layers) and the salience of the combination of basic movements through its high-level parameters (last hidden layers). Therefore, as an example, the CNN trained on recognizing basic human activities such as sitting, standing and walking (source learning task), and the one trained on recognizing SMMs (target learning task) will both have the same first and intermediate layer weights and different high layer weights. Another example is the CNN trained on SMMs of an atypical subject (source task) and the one trained on SMMs of another atypical subject (target task) which will have common first and intermediate hidden layers' weights and different last hidden layer weights, due to the inter-subject variability across atypical subjects.

In that sense, we propose a "Transfer learning with SVM read-out" framework which is composed of two parts: (i) the first part having first and intermediate layers' weights of a CNN already pre-trained on a source learning task, (the last CNN layer being discarded), and (ii) the second part composed of a support vector machine (SVM) classifier with RBF kernel which is connected to the end of the first part. Then, we feed the entire training dataset of the target task into this framework in order to train the SVM parameters. As opposed to training a CNN on the target task which requires updating all hidden layers' weights for several iterations using a large training set for all these weights to converge, our framework computes weights of the last layer(s) only, in one iteration only. Moreover the advantage of using SVM as the classifier is that it is fast and generally performs well on small training set since it only relies on the support vectors, which are the training samples that lay exactly on the hyperplanes used to define the margin. In addition, SVMs have the powerful RBF kernel, which allows to map the data to a very high dimension space in which the data can be separable by a hyperplane, hence guaranteeing convergence. Hence, our framework can be regarded as a global, fast and light-weight technique for time series classification where the target task has limited annotated/labeled data.

4.3.2 Experiments

We conduct this experiment again on the SMM recognition task. However, we will perform SMM recognition across multiple atypical subjects as opposed to SMM recognition within subjects which was developed in experiments of Sections 3.2. and 4.2. Indeed, due to the inter-subject variability of SMMs, a CNN trained on movements of an atypical subject i performs badly on detecting SMMs of another atypical subject and therefore cannot be applied on SMMs other than subject i's SMMs. Indeed, testing one of the trained CNNs of experiment 4.2.2 (let us say the trained CNN of subject i study j) on SMMs of a subject other than subject i produces a very low F1-score with less than 30%. This implies that CNN features learned from SMMs of one subject differ from the ones learned from another subject and that they are not general enough to detect SMMs of another subject. So, instead of training a CNN for each atypical subject individually (Sections 3.2.2 and 4.2.2), we use the "transfer learning with SVM read-out" framework for the detection of SMMs across subjects. Through this experiment, our goal is to prove that this framework is a global, fast and light-weight technique for time series classification tasks which experience a lack of labeled data.

The target learning task will be the recognition of SMMs of subject i, while the source learning task will be either a task close to the target task such as the recognition of SMMs of multiple subjects other than i, or a task different but related to the target task such as the recognition of basic human activities. Running the "TL SVM" framework with the former and the latter target tasks will be denoted as "*TL SVM similar domains*" and "*TL SVM across domains*" respectively. Through this experiment, we will also show that these chosen source learning tasks contribute in generating CNN features that are general/global enough to recognize SMMs of any new atypical subject.

Datasets. The dataset used for the target learning task is the same SMM dataset used in Section 3.2.2. The dataset used for the source domain of the "*TL SVM similar domains*" experiment is also the SMM dataset, whereas the one used for the source domain of the "TL SVM across domains" experiment is the PUC dataset which is described in Section 3.2.2.

When using the SMM dataset in the target and source learning tasks, we do not take into consideration signals of all accelerometers/sensors (torso, right and left wrist) but rather signals of the torso sensor, resulting in input samples with 3 channels instead of 9. So, with torso measurements only, the only stereotypical movements that could be captured are the rock and flap-rock SMMs (and no flap SMMs). Accordingly, only rock and flap-rock SMM instances will be used as inputs in this experiment.

When using the PUC dataset for the source learning task, only the waist accelerometer (waist being next to torso) is taken into account since the other accelerometers (located at the thigh, ankle and arm) will not be relevant to the SMM recognition task during transfer learning. We consider the waist location to be equivalent to the torso location so that the CNN pre-trained on the source learning dataset can further be transferred to the target learning task (SMM recognition). Accordingly, input instances will have 3 channels instead of 12.

Pre-processing. The pre-processing phase is the same as in Section 3.2.2.

Experimental setup. In experiments below, the architecture of the CNN model in time domain and frequency domain as well as training parameters are similar to the ones in Section 3.2.2. In addition, the target learning task consists of SMM recognition of a target subject i of study j where $i \in [1, 6]$ and $j \in [1, 2]$. Accordingly, one "transfer learning with SVM" framework will be run per domain (time or frequency domain) per subject per study. The training and testing sets of subject i (study j) are selected using the same k-fold cross-validation used in Section 3.2.2. However only a subset of the training set (10,000–30,000 instances) is used, where 2000 SMM instances are randomly selected from the overall training set for training.

In order to perform SMM recognition on a target subject using transfer knowledge from SMMs of other subjects, the following steps are performed in time and frequency domains for each study i within each study j:

- **Step 1:** we train a randomly initialized CNN in both time and frequency domains, for 5–15 epochs, using: (i) SMM instances of all 6 atypical subjects within study j except subject i for "*TL SVM similar domains*" framework, and (ii) basic human activities' instances for "TL SVM across domains" framework. This process results in a pre-trained CNN model.

- **Step 2:** we reuse all layers of this CNN except the last layer (which is a fully connected layer) which is removed and replaced by the SVM classifier. The SVM of the transfer learning framework is trained using a small subset of subject i's training data (i.e., 2000 SMM samples), which results in learned high-level features. Then, the remaining SMMs of subject i are implemented for testing the framework. Knowing that the input consists of only a subset of the original training dataset of subject i, we choose to run the SVM for 5 runs, with 2000 randomly selected samples in each run. In such a way, by aggregating F1-scores of the 5 runs, we provide more realistic results. This procedure is applied on every domain (time and frequency) on every subject i in every study j.

4.3.3 Results

"TL SVM similar domains". As depicted in **Table 3**, this framework (combining part of the pre-trained CNN with an SVM) is able to identify SMMs at a mean F1-score of 74.50 and 91.81% for time and frequency domains respectively. As opposed to the technique of directly applying the pre-trained CNN for classification which fails to recognize SMMs, "*TL SVM similar domains*" framework is able to capture relevant features for the recognition of SMMs across subjects. Thus, we can infer that low and mid-level SMM features share the same information from one subject to another and that "*TL SVM similar domains*" can be used as a global framework to identify SMMs of any new atypical subject. Furthermore, low- and mid-level features captured from a source learning task can be employed as low- and mid-level features of a target learning task close to the source task.

"TL SVM across domains". Training this framework produces satisfying results with a mean score of 72.29 and 79.78% in time and frequency domains respectively (**Table 3**). So, fixing low and mid-level features to features of basic movements and adjusting only the high-level features by an SVM seems to give satisfying classification results, which confirms that our framework has engaged feature detectors for finding stereotypical movements in signals. These results, especially the frequency-domains results, indicate that: (i) connecting low- and mid-level features of basic movements to an SVM classifier then feeding in 2000 instances for training the SVM generates a global framework which holds relevant and general representation that adapts to SMMs of any new atypical subject i, and (ii) both human and stereotypical movements may share low and mid-level features in common, suggesting that low- and mid-level information learned from a source target task by a CNN model can be directly applied as low- and mid-level features for a target learning task different but related to the source learning task, especially when there is a lack of labeled data within the target learning task.

Moreover, both our techniques are compared against the following methods:

i. The "*CNN with few data*" technique which consists of training a CNN in time and frequency domains with randomly initialized weights using the same target training data as the ones of "*TL SVM similar domains*" framework (i.e., 2000 SMM instances of the target subject i). The difference between this CNN and the CNN of Section 3.2.2 is that less data is used for training (2000 versus 10,000–30,000 training instances), only torso sensor measurements are applied in the former (compared to torso, right and left wrist sensor measurements in the latter), and only rock and flap-rock SMM instances are considered in the former (compared to rocking, flap-rock and flap SMM instances in the latter). We refer to this technique as "*CNN few data*".

ii. The "*transfer learning with full fine-tuning*" technique (referred to as "*TL full fine-tuning*") consists of identifying SMMs of subject i within study j by first training a CNN in time and frequency domains for 5–15 epochs using SMM instances of all 6 atypical subjects within study j except subject i (as in Step 1 of training "*TL SVM similar domains*" framework), then by fine-tuning (e.g., updating) weights of all CNN layers using same target training data.

iii. The "*transfer learning with limited fine-tuning*" technique (denoted as "*TL limited fine-tuning*") is the same as "*transfer learning with full fine-tuning*" except that the fine-tuning process is effective only on weights of the last CNN layer L while weights of other layers $1, \ldots, L-1$ are unchanged.

Results and properties of the three techniques are depicted in **Tables 3** and **4** respectively. From these results and properties, the following observations can be made:

- "*TL SVM similar domains*" framework performs higher than the three frameworks "*CNN few data*", "*TL full fine-tuning*" and "*TL limited fine-tuning*" in both time- and frequency-domain. This can be explained by the nature of the training process of the three frameworks, which relies on updating

Approaches	**Study 1**						**Study 2**					**Mean**
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S1**	**S2**	**S3**	**S4**	**S5**	
Time domain												
CNN few data	72.02	62.31	52.98	88.47	60.61	88.86	80.90	53.28	16.00	82.66	75.82	66.72
TL full fine-tuning	**75.73**	71.31	**59.04**	91.67	59.47	91.89	85.66	**63.84**	38.57	92.24	**82.31**	73.79
TL limited fine-tuning	75.44	56.50	50.86	**91.74**	**63.86**	**93.11**	**85.88**	62.62	27.14	**93.63**	81.16	71.09
TL SVM similar domains	75.37	**76.44**	56.53	**91.74**	63.37	92.76	84.86	62.97	**41.60**	93.32	80.55	**74.50**
TL SVM across domains	71.66	74.40	66.80	90.69	61.87	92.19	81.35	58.13	35.66	88.44	73.98	72.29
Frequency domain												
CNN few data	76.64	96.55	63.44	93.13	82.58	94.61	84.94	76.42	29.51	93.66	83.41	79.54
TL full fine-tuning	88.51	**97.22**	**88.15**	97.53	**91.29**	98.26	92.17	88.19	52.17	96.59	91.98	89.28
TL limited fine-tuning	87.98	94.59	62.70	**97.57**	87.94	98.36	91.62	87.08	40.00	97.82	90.82	85.14
TL SVM similar domain	**90.54**	**97.22**	83.86	95.24	86.19	**98.45**	**92.71**	**90.49**	**84.99**	**97.99**	**92.22**	**91.81**
TL SVM across domains	74.50	91.56	43.77	93.11	76.03	94.2	85.16	74.67	66.98	93.66	83.99	79.78

Table 3.
Results of CNN approaches used in this experiment per domain (time or frequency) per subject, per study. Highest rates are in bold.

Approaches	# parameters updated for one pass (1 batch)	# batches per iteration	# iterations (epochs)	Implementation on Android device
CNN few data	1.2e + 06 (time-domain) 7.1e + 05 (frequency-domain)	14 (2000/150)	20–35 (time-domain) 55–85 (frequency-domain)	No (too much memory consumption)
TL full fine-tuning	1.2e + 06 (time-domain) 7.1e + 05 (frequency-domain)	14 (2000/150)	5–15	No (too much memory consumption)
TL limited fine-tuning	1000 (500*2)	14 (2000/150)	5–15	No (hard to run back-propagation on mobile devices)
TL SVM (similar domains and across domains)	500	1	1	Yes (easy to train SVM on mobile devices)

Table 4.
Properties and resources used for the different techniques implemented in the experiment.

parameters using backpropagation. And, knowing that backpropagation requires abundant data for proper training, a lack of training data (2000 SMM instances) pushes the three frameworks to overfit and be less efficient.

- "*TL SVM similar domains*" and "*TL SVM across domains*" architectures perform better than "*CNN few data*" by 7.78 and 5.57% respectively in time-domain and by 12.27 and 0.24% respectively in frequency-domain. Therefore, both architectures engage in capturing more general features than "*CNN few data*". In terms of resources, one advantage of the two architectures over "*CNN few data*" is that the former converge much faster than the latter. Indeed, the former require 5–15 epochs (in both time and frequency domains) for full convergence while the latter needs 20–35 epochs and 55–85 epochs in time- and frequency-domain respectively for full convergence, as shown in **Table 4**. Another advantage resides in the number of parameters that have to be learned, which is 500 for the former (in both time and frequency domains) versus 1.2e + 06 and 7.1e + 05 for the latter in time and frequency domain respectively (**Table 4**). Hence, as opposed to "*TL full fine-tuning*" and "*TL limited fine-tuning*" frameworks, the "TL SVM" can be regarded as a global, fast and light-weight framework for SMM recognition across subjects.

- "*TL full fine-tuning*" has a slightly higher performance than "*TL limited fine-tuning*" by 2.71 and 4.14% in time- and frequency-domain respectively, suggesting that fine-tuning weights of layers 1, ..., $L - 1$ (where L is the number of CNN layers) is unnecessary since it does not improve SMM recognition significantly. This confirms the earliest assumption that atypical subjects have similar low- and mid-level features but different high-level features.

- "*TL SVM across domains*" framework yields a lower performance than "*TL SVM similar domains*" by 2.21% and 12.02% in time- and frequency-domain respectively. This implies the superiority in the SMM recognition task of low and mid-level features learned from SMMs over the ones learned from basic human movements. However, the latter features are more global. In time-domain, the small rate difference (2.21%) between "*TL SVM across domains*" and "*TL SVM similar domains*" suggests that the low- and mid-level feature space generated by human activities shares common details with the one generated by movements of specific atypical subjects. This is not the case for frequency-domain series, which can be explained by the difference in the frequency range between human activities and SMMs. Indeed, the FFT amplitude of human activities is contained below 10 Hz, pre-training the CNN on human activity frequency-signals from 0 to 3 Hz and not from 0 to 10 Hz results in imperfect human activity features which, combined with the SVM, do not seem to yield good classification results on the recognition of SMMs. If we were to have a new target learning task whose data signals are within the same frequency range as data signals of the source learning task, then "*TL SVM across domains*" would have achieved the same performance as "*TL SVM similar domains*".

- One advantage of "*TL SVM similar domains*" and "*TL SVM across domains*" is that they can be implemented in Android portable devices, as shown in **Table 4**. Indeed, an expert could receive continuous acceleration signals from the torso accelerometer of a subject, and label them on the fly (as SMM/non-SMM) as the subject performs his activities/movements. This results in annotated time series which are then preprocessed and fed into either "*TL SVM similar domains*" or "*TL SVM across domains*" for training. A one-minute recording of these signals is sufficient to train one of the two frameworks. Afterwards, this framework is ready to use for recognizing further SMMs on that same subject.

5. Conclusion

Time series pose important challenges to existing approaches which perform predictive modeling for classification tasks. In this paper we present a review on our previous works. Our contributions are aggregated into two categories: data-level and algorithm-level approaches. Our data-level approach consists of encoding time series using STin order to produce noise-free input signals which offer a more efficient CNN training. At the algorithm level, one approach is the adaptive convolutional layer filter approach which consists of determining the size of the filter based on an analysis of the input time series signals and fluctuations present within them. Indeed, choosing the proper 1st layer filter generates features maps which are more informative about the input signals and which capture the whole peaks within input signals. Furthermore, "*TL SVM similar domains*" and "*TL SVM across domains*" are algorithm-level approaches dealing with tasks with limited annotated data, which are regarded as two global, fast and light-weight techniques for these kinds of tasks. These two CNN approaches generate features general and global enough to recognize time series of the target learning task, given time series of a source learning task that is similar or different but related to the target learning task. All these approaches were implemented on the recognition of human activities, including normal activities performed by typical subjects and disorder-based activities performed by atypical subjects (such as SMMs of autistic subjects). Experimental results have showed the superiority of our techniques and their ability to extract relevant features from time series inputs. As a perspective, knowing that time series datasets often contain outliers either due to noisy time series or mislabeled time series (e.g. incorrect labels), we aim at studying a robust CNN that is insensitive to outliers. As opposed to our data-level CNN technique (mentioned in this paper) whose goal is to eliminate noise from time series, this robust CNN is an algorithm-level technique with acts at the level of loss functions by controlling high error values caused by outliers.

Conflict of interest

The authors declare that they have no conflicts of interest.

Author details

Lamyaa Sadouk
Faculty of Science and Technology, University Hassan 1st, Settat, Morocco

*Address all correspondence to: lamyaa.sadouk@gmail.com

References

[1] Kaya H, Gunduz-oguducu S. A distance based time series classification framework. Information Systems. 2015; **51**(C):27-42

[2] Wang S, Liu P, She MFH, et al. Bag-of-words representation for biomedical time series classification. Biomedical Signal Processing and Control. 2013; **8**(6):634-644

[3] Sadouk L, Gadi T, Essoufi EH. Convolutional neural networks for human activity recognition in time and frequency-domain. In: Proceedings of ICRTS. 2017. pp. 485-496

[4] Sadouk L, Gadi T, Essoufi EH. A novel deep learning approach for recognizing stereotypical motor movements within and across subjects on the autism spectrum disorder. Computational Intelligence and Neuroscience. 2018:16. DOI: 10.1155/2018/7186762. Article ID 7186762

[5] Abdel-Hamid O, Deng L, Yu D. Exploring convolutional neural network structures and optimization techniques for speech recognition. Interspeech. 2013;**2013**:1173-1175

[6] Abdel-Hamid O, Mohamed AR, Jiang H, Penn G. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In: Proceedings of ICASSP IEEE. 2012. pp. 4277-4280

[7] Kini BV, Sekhar CC. Large margin mixture of AR models for time series classification. Applied Soft Computing. 2013;**13**(1):361-371

[8] Antonucci A, De Rosa R, Giusti A, et al. Robust classification of multivariate time series by imprecise hidden Markov models. International Journal of Approximate Reasoning. 2015;**56**(B):249-263

[9] Rakthanmanon T, Campana B, Mueen A, et al. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. ACM Transactions on Knowledge Discovery from Data. 2013; **7**(3):1-31

[10] Jeong Y, Jeong MK, Omitaomu OA. Weighted dynamic time warping for time series classification. Pattern Recognition. 2011;**44**(9):2231-2240

[11] Schäfer P. The BOSS is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery. 2014;**29**(6):1505-1530

[12] Bailly A et al. Dense bag-of-temporal-SIFT-words for time series classification. In: Lecture Notes in Artificial Intelligence. 2016

[13] Chen Z, Zuo W, Hu Q, Lin L. Kernel sparse representation for time series classification. Information Sciences. 2015;**292**:15-26

[14] Hills J, Lines J, Baranauskas E, et al. Classification of time series by shapelet transformation. Data Mining and Knowledge Discovery. 2014;**28**(4): 851-881

[15] Wang Z, Oates T. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In: Workshops AAAI. 2015

[16] Ngiam J, Chen Z, Chia D, Koh PW, Le QV, Ng AY. Tiled convolutional neural networks. In: Advances in Neural Information Processing Systems. 2010: 1279-1287

[17] Cui Z, Chen W, Chen Y. Multi-scale convolutional neural networks for time series classification. 2016. arXiv preprint arXiv:1603.06995

[18] Wenlin W et al. Earliness-aware deep convolutional networks for early time series classification. 2016. arXiv preprint arXiv:1611.04578

[19] Wang Z, Yan W, Oates T. Time series classification from scratch with deep neural networks: A strong baseline. In: Proceedings of the IEEE IJCNN. 2017. pp. 1578-1585

[20] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on CVPR; 2015

[21] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on CVPR. 2016. pp. 770-778

[22] Karim F et al. Lstm fully convolutional networks for time series classification. IEEE Access. 2018;**6**:1662-1669

[23] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation. 1997;**9**:1735-1780

[24] Le Guennec A, Malinowski S, Tavenard R. Data augmentation for time series classification using convolutional neural networks. In: In: ECML/PKDD Workshop on AALTD. 2016

[25] Zheng Y, Liu Q, Chen E, et al. Time series classification using multi-channels deep convolutional neural networks. In: Proceedings of the 15th ICWAIM. 2014. pp. 298-310

[26] Zhao B et al. Convolutional neural networks for time series classification. Journal of Systems Engineering and Electronics. 2017;**28**(1):162-169

[27] Hatami N, Gavet Y, Debayle J. Classification of time series images using deep convolutional neural networks. In: Proceedings of ICMV 2017, vol. 10696; International Society for Optics and Photonics. 2018

[28] Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, et al. The UCR Time Series Classification Archive. Available from: www.cs.ucr.edu/~eamonn/time_series_data/

[29] Yang J, Nguyen MN, San PP, Li X, Krishnaswamy S. Deep convolutional neural networks on multichannel time series for human activity recognition. In: IJCAI. Vol. 15. 2015. pp. 3995-4001

[30] Rad NM, Kia SM, Zarbo C, van Laarhoven T, Jurman G, Venuti P, et al. Deep learning for automatic stereotypical motor movement detection using wearable sensors in autism spectrum disorders. Signal Processing. 2018;**144**:180-191

[31] Groß W, Lange S, Bödecker J, Blum M. Predicting time series with space-time convolutional and recurrent neural networks. In: Proceedings of the 25th ESANN. 2017. pp. 71-76

[32] Stockwell RG, Mansinha L, Lowe RP. Localization of the complex spectrum: The S transform. IEEE Transactions on Signal Processing. 1996; **44**(4):998-1001

[33] Goodwin MS, Haghighi M, Tang Q, Akcakaya M, Erdogmus D, Intille S. Moving towards a real-time system for automatically recognizing stereotypical motor movements in individuals on the autism spectrum using wireless accelerometry. UBICOMP. 2014;**14**

[34] Ugulino W, Cardador D, Vega K, Velloso E, Milidiú R, Fuks H. Wearable computing: Accelerometers' data classification of body postures and movements. In: Advances in Artificial Intelligence-SBIA. 2012, 2012. pp. 52-61

[35] Ma Y, Genton MG, Parzen E. Asymptotic properties of sample quantiles of discrete distributions. Annals of the Institute of Statistical Mathematics. 2011;**63**:227-243