Deep-Aligned Convolutional Neural Network for Skeleton-Based Action Recognition and Segmentation

Babak Hosseini¹ · Romain Montagne² · Barbara Hammer¹

Received: 17 March 2020 / Revised: 30 April 2020 / Accepted: 5 May 2020 / Published online: 16 May 2020
© The Author(s) 2020

Abstract

Convolutional neural networks (CNNs) are deep learning frameworks which are well known for their notable performance in classification tasks. Hence, many skeleton-based action recognition and segmentation (SBARS) algorithms benefit from them in their designs. However, a shortcoming of such applications is the general lack of spatial relationships between the input features in such data types. Besides, non-uniform temporal scalings are a common issue in skeleton-based data streams which leads to having different input sizes even within one specific action category. In this work, we propose a novel deep-aligned convolutional neural network (DACNN) to tackle the above challenges for the particular problem of SBARS. Our network is designed by introducing a new type of filters in the context of CNNs which are trained based on their alignments to the local subsequences in the inputs. These filters result in efficient predictions as well as learning interpretable patterns in the data. Also, our DACNN framework can incrementally expand its deep structure based on the learning progress, which makes it flexible regarding different SBARS datasets. We empirically evaluate our framework on real-world benchmarks showing that the proposed DACNN algorithm obtains a competitive performance compared to the state of the art while benefiting from a less complicated yet more interpretable model.

Keywords Convolutional neural networks · Action recognition · Time series alignment · Action segmentation

Abbreviations

CNN	Convolutional neural networks
DACNN	Deep-aligned convolutional neural network
SBARS	Skeleton-based action recognition and segmentation
Al-filters	Alignment filters
Abs-filters	Abstract filters

1 Introduction

Action recognition is still a challenging problem in the area of computer vision despite its numerous applications in sports analysis, video surveillance, and human–machine interaction [1, 36]. A popular way to represent action data is through 3D skeleton sequences, which are the 3D positions of body joints through time. This representation medium can efficiently describe the underlying dynamics of the actions [21]. Recently, the development of motion capture technologies such as the Microsoft Kinect [51] along with the existing robust pose estimation methods [39] has considerably facilitated the collection of skeleton-based datasets. These technological advancements have motivated considerable research efforts being put into this area [2, 10].

In particular, skeleton-based action recognition has become an interesting problem for many deep learning algorithms such as convolutional neural networks (CNNs) [9, 22, 47] and recurrent neural networks (RNN) [10, 28, 41]. RNN methods can learn the temporal dynamics of the sequential data; nevertheless, they have practical shortcomings in the training of their stacked structures [30, 45].

✉ Babak Hosseini
bhosseini@techfak.uni-bielefeld.de

Romain Montagne
r.montagne@hotmail.fr

Barbara Hammer
bhammer@techfak.uni-bielefeld.de

¹ Center of Cognitive Interactive Technology (CITEC),
Bielefeld University, Bielefeld, Germany

² Eurodecision, Versailles, France

Compared to RNN architectures, CNN-based methods provide more effective solutions by extracting local features from their input and finding discriminative patterns in the data [13, 22]. However, regardless of the promising feature extraction capability of CNN, its specific convolutional structure is designed originally for image-based input data and primarily relies on spatial dependencies between the neighboring points. In contrast, such a direct relationship does not generally exist in skeleton-based action datasets. Although some works tried to solve this problem by using 1-dimensional filters (only for the temporal dimension), it is still not an efficient solution to this specific shortcoming of CNN-based frameworks [52].

A crucial step before analyzing any motion data is the temporal segmentation phase, in which we need to split the long streams of recorded data into meaningful actions [53]. Although there exist unsupervised algorithms for temporal segmentation of motion data [24, 53], they usually over-segment actions into smaller sub-actions or segment also the blank parts of the stream. To our knowledge, no major supervised segmentation method has been proposed, particularly for skeleton-based data. The temporal segmentation of skeleton-based sequences refers to predicting the action to which each time frame belongs. In other domains such as speech recognition and text analysis, this process is known as the sequence labeling of the data [13]. In works similar to [26, 32], they add a conditional random field (CRF [25]) layer to the last layer of an LSTM network to perform the sequence labeling. As a different method, the CRF layer is combined with a deep gated RNN in [48] for word labeling. In [13], the authors propose a temporal labeling architecture entirely based on CNNs and combined with gated units and attention modules. Although these methods are effective regarding the accuracy and the computational complexity, their implementation on skeleton-based motion data requires to perform problem-specific data preprocessing and apply substantial changes in the networks' structure.

Motivations An essential group of techniques for classification of the sequential data is time series alignment methods [35]. It is shown that via comparing each data sequence to some predefined or learned sequences, we can discriminate or segment the data samples with high accuracy [3, 37]. Also, in algorithms similar to [49], finding a small distinct subsequence in the input data can reveal its classification label. Therefore, it is logical to design filters in CNN architecture based on the alignment concept to extract essential features in sequential data.

Besides finding different patterns in the input space, one aim of increasing the depth (number of middle layers) in CNNs is to spatially/temporally compress the representation of the data. Consequently, the extracted feature maps are downscaled as deeper we go through the layers of CNN. Nevertheless, the necessary level of compression depends

on several factors, one of which is the temporal length of the essential patterns in the input data. Therefore, sequence length should help us to better decide on the required depth of such network in an SBARS task.

Contributions In this work, we propose a novel framework deep-aligned convolutional neural network (DACNN) as an efficient algorithm for skeleton-based action recognition and segmentation (SBARS) algorithm. As depicted in Fig. 1, DACNN is constructed upon the activation maps of the alignment kernels in its first layer following by a CNN with 1-dimensional convolutional filters (1D-CNN). This architecture makes DACNN flexible to the input size and additionally can smoothly increase the number of its deep layers (in 1D-CNN) automatically based on the structure of the given dataset.

To be more specific,

- We introduce the alignment kernels (AI-filters) in the context of CNN, which are more efficient than the convolutional filters regarding the temporal feature extraction and classification of skeleton-based action data.
- DACNN learns temporal subsequences in the data as essential local patterns which make the outcome of the network semantically interpretable and leads to more accurate predictions.
- As another crucial contribution to the state of the art, our DACNN architecture can incrementally extend its depth (number of middle layers) based on the quality and length of the learned AI-filters during the learning process and without disrupting the training phase.

In the next section, we summarize the most relevant work in the literature. In Sect. 3, we introduce the alignment filters while we explain the architecture of DACNN in Sect. 4. The empirical evaluation of DACNN is provided in Sect. 5, and we conclude the paper in the last section.

Fig. 1 General overview of DACNN framework. Alignment kernels preprocess the input streams. 1D-CNN performs temporal prediction based on the derived alignment map. The IDI module automatically increases the depth of 1D-CNN. The fine-prediction unit improves the resolution of output prediction

2 Related Work

Generally, it is possible to split the skeleton-based action recognition methods into two general categories:

The first group includes methods with a preprocessing step to extract features (usually hand-coded) which best represent the skeleton information. For instance, in [46] the local occupancy pattern was proposed based on the depth appearance of the joints along with an ensemble action recognition model. In [20], they proposed a discriminator based on the covariance matrix of joints locations, while in [44], the algorithm was designed based on the 3D geometric relationships between different regions of the body.

The second category contains deep learning feature extraction methods which are generally designed based on CNN and RNN models. Among RNN frameworks, a regularized LSTM architecture is proposed in [54] for co-occurrence feature extraction. A spatiotemporal attention-based model is utilized in [41] to assign different weights to different frames, and a trust gate technique was proposed in [28] to deal with the noise in skeleton-based data.

Regarding the approaches based on CNN models, Tang et al. [43] combined CNN with a reinforcement learning module to learn the most efficient video frames. In [22], cylindrical coordinates were utilized to present a new skeleton representation. The skeleton data were transformed into images in [30] to be more appropriate for CNN architecture, and Li et al. [45] combined two CNN models assigned to the joint position and velocity information individually.

A majority of the proposed deep learning algorithms for temporal segmentations benefit from the probabilistic power of CRF. In works similar to [19, 26, 32], they add a CRF layer to the last layer of a LSTM network to perform to perform sequence labeling. As a different method, the CRF layer is combined with a deep gated RNN in [48] for word labeling. In [13], they propose a temporal labeling architecture entirely based on CNNs and combined with gated units and attention modules. Their method is efficient regarding the accuracy and the computational complexity.

Nevertheless, direct application of these methods on skeleton-based motion data requires to choose a proper preprocessing and substantial changes in the networks' structure. Another common practice in many algorithms is to perform the segmentation phase using other methods as the preprocessing step while applying their action recognition model on the obtained segmented data [12, 33, 34].

As the difference to the current literature, our algorithm performs the segmentation and classification of the skeleton-based actions in a seamless architecture while finding essential local patterns in the input data.

3 Alignment Kernels for CNN

Generally, the skeleton-based data are an instance of multi-variate time series. Hence, we denote each action sequence by $\mathbf{X} = [\mathbf{x}(1) \dots \mathbf{x}(T)] \in \mathbb{R}^{n_f \times T}$, where n_f indicates the number of dimensions (related to the body joints) and T denotes the temporal length of the sequence. Although n_f is commonly higher than the number of joints, we may interchangeably use them in this document to facilitate the understanding of the main concepts. For each input sequence $\mathbf{X}$, there exists a corresponding frame-based labeling matrix $\mathbf{H} \in \mathbb{R}^{C \times T}$, where $h_{ct} = 1$ if the t -th frame of $\mathbf{X}$ belongs to class c (when having more than one action in $\mathbf{X}$). The matrix $\mathbf{H}$ is zero elsewhere and $c \in \{1, \dots, C\}$ in a C -class setting. Hence, in an SBARS task, we are interested in predicting the true value of $\mathbf{H}$ for each $\mathbf{X}$.

Studying the typical CNN architectures, the filters used in the first convolutional layer of the network mathematically behave as degree-1 polynomial kernels. In particular, applying a convolution filter $\mathbf{W}$ (conv-filter) with an $n \times n$ receptive field to an image patch $\mathbf{Y}$ of the same size computes the feature value v as

$$v = \sum_{i,j=1}^n \mathbf{y}_{ij} \mathbf{W}_{ij} + b = (\mathbf{y}^\top \mathbf{w} + b)^1 = \mathcal{K}_{pol1}(\mathbf{y}, \mathbf{w}), \quad (1)$$

where b is the filter bias and $(\mathbf{y}, \mathbf{w})$ are the vectorized forms of $(\mathbf{Y}, \mathbf{W})$, respectively. In (1), $\mathcal{K}_{pol1}(\mathbf{y}, \mathbf{w})$ denotes the polynomial kernel of degree-1 between $\mathbf{y}$ and $\mathbf{w}$, which measures the similarity between the input patch $\mathbf{Y}$ and the filter $\mathbf{W}$. Extending (1) to all the filters in the first layer of CNN and to all parts of the input data, the first layer of a CNN can be interpreted as a multiple-kernel function [14] which measures the similarity of the given input to some exemplars/filters. However, in kernel-based classification problems, it is known that employing the squared exponential kernel (A.K.A Gaussian kernels) can better discriminate the input space compared to polynomial kernels [4]. This superiority results from the flexibility and large input domain of Gaussian kernels. Therefore, we reshape the traditional CNN design to benefit from squared exponential filters to achieve better performance in SBARS tasks.

3.1 Alignment Filters

Principally, the input layer of a CNN consists of n_f different channels $\{\mathbf{x}_j\}_{j=1}^{n_f} \in \mathbb{R}^T$ each of which contains the temporal skeleton data related to one dimension of $\mathbf{X}$. Based on the discussed rationale in the previous section, we propose the following distance-based nonlinear kernel as the fundamental feature extraction unit of our DACNN architecture (alignment layer in Fig. 1):

Fig. 2 Alignment layer of the network. Each AI-filter f_i^1 is applied to all n_f channels in the input to form the i -th row in the alignment map V^1 . Adding V^1 to the alignment maps of other Abs-filters derives the augmented map $\bar{V}$

$$g(\mathbf{x}_j|_{t_0}^t, \mathbf{f}_i^1) = e^{-\|\mathbf{f}_i^1 - \mathbf{x}_j|_{t_0}^t\|_2^2}, \quad \forall i = 1, \dots, d_1, \quad (2)$$

where $\{\mathbf{f}_i^1\}_{i=1}^{d_1} \in \mathbb{R}^t$ are the alignment filters with the receptive field of t and $\mathbf{x}_j|_{t_0}^t$ denotes any temporal window of length t starting from the t_0 -th frame of a $\mathbf{x}_j$ channel. After scanning each channel of $\mathbf{X}$ by these filters with a stride s (Fig. 2), we obtain a tensor $\mathbf{V} \in \mathbb{R}^{T \times n_f \times d_1}$ as the activation map. Each entry v_{jik} from $\mathbf{V}$ represents the similarity between the j -th window in channel $\mathbf{x}_i$ and the filter $\mathbf{f}_k$, and summing $\mathbf{V}$ over its second dimension results in the more summarized activation map $\mathbf{V}^1 \in \mathbb{R}^{d_1 \times T}$ (Fig. 2).

To make an analogy to a regular CNN structure, we reformulate the introduced alignments as an l_2 -norm comparison layer followed by an activation layer of $f(x) = e^{-x}$ units. This reformulation is conceptually similar to the convolution and RELU layers of a regular CNN architecture. Therefore, a proper classification-based training scenario leads to finding discriminative patterns in $\mathbf{V}^1$ with high activation values (near 1). In other words, we learn filters $\mathbf{f}_i$ which can distinguish the data classes based on their similarities to the local parts in the data channels $\{\mathbf{x}_j\}_{j=1}^{n_f}$.

3.2 Vanishing Gradient and Saturated Activation

Updating the values of an AI-filters $\mathbf{f}$ in a gradient-based optimization routine, the activation function of (2) and its gradient obtain infinitesimal values if $\|\mathbf{f} - \mathbf{x}_j|_{t_0}^t\|_2^2$ become large. This condition leads to zero updates of $\mathbf{f}_i$ in the optimization process which may occur when a filter $\mathbf{f}_i$ has a large distance to all subsequences in $\mathbf{X}$ (e.g., lousy initialization) or the learning rate is too high.

As a systematic workaround, we compute the activation map $\mathbf{V}^1$ using the function

$$g(\mathbf{x}, \mathbf{f}) = (1 + a)e^{-\|\mathbf{x} - \mathbf{f}\|_2^2} - a \quad (3)$$

instead of (2) by choosing a small a (in implementations we use $a = 0.1$). The value of $g(\mathbf{x}, \mathbf{f})$ becomes $-a$ when $\|\mathbf{x} - \mathbf{f}\|_2^2$

is large, we preserve the sparseness effect of the activation function when computing $\mathbf{V}^1$, and it leads to faster convergence. In addition, in the backward phase of the training, we employ the conditional gradient

$$\nabla_{\mathbf{f}} g = \begin{cases} -2(1 + a)e^{-\|\delta\|_2^2} & \|\delta\|_2^2 \leq 0.5 \\ -2[(1 + a)e^{-\|\delta\|_2^2} + 1]\delta + 2\sqrt{0.5}\text{sign}(\delta) & \|\delta\|_2^2 > 0.5 \end{cases} \quad (4)$$

where $\delta = \mathbf{x} - \mathbf{f}$, and $\sqrt{0.5}$ is the point from which the gradient of (2) starts to decrease toward zero. In such case, the gradient becomes $-2(\mathbf{x}_j|_{t_0}^t - \mathbf{f})$ when $\|\mathbf{f} - \mathbf{x}_j|_{t_0}^t\|_2^2$ has a relatively large value and prevents the filter weights from becoming saturated. More detailed explanation of the above is provided in “Appendix.”

3.3 Abstract Filters

Although the proper training of the AI-filters can fit them to the small local patterns in $\mathbf{X}$ which are essential to the classification task, we are also interested in finding longer patterns in the action data. The benefits of finding these patterns are twofolds:

1. Applying longer filters on the data leads to sparser activation maps (Fig. 3: $\{v^4, v^3\}$ versus $\{v^1, v^2\}$).
2. Long patterns are semantically more interpretable than the short ones (Fig. 3: v^4 represents a complete action).

To that aim, we define the abstract filters (Abs-filter) $\{\mathbf{f}_i^p\}_{i=1}^{d_p}$ with the receptive field of pt , where $p \in \mathbb{N}$. Each $\mathbf{f}_i^p$ is a temporal concatenation of p smaller AI-filter of size t as $\mathbf{f}_i^p = \bigoplus_{j \in I} \mathbf{f}_j^1$, where $\bigoplus$ concatenates the AI-filters $\mathbf{f}_j^1$ with the index order given by the set I . To find $\mathbf{f}_i^p$ filters systematically, we select the potential candidates among the AI-filters as the first step.

Definition 1 Two AI-filters $\{\mathbf{f}_i^1, \mathbf{f}_j^1\}$ of size t are candidates to form an Abs-filter if we can find a window of size t starting at the time frame t_0 of a data channel $\mathbf{x}_k$ such that

Fig. 3 Abstract filters $\{\mathbf{f}_4, \mathbf{f}_3\}$ result in sparse alignment maps $\{v^4, v^3\}$, and $\mathbf{f}_4$ can be interpreted as the representative of a complete arm movement in the raise action

$$\rho(\mathbf{f}_i^1, \mathbf{f}_j^1, \mathbf{x}_k |_{t_0}^t) = g(\mathbf{x}_k |_{t_0}^t, \mathbf{f}_i^1)g(\mathbf{x}_k |_{t_0+t}^t, \mathbf{f}_j^1) \geq r, \quad (5)$$

where r is a scaler with a value sufficiently close to 1.

Based on the definition 1, when $\rho(\mathbf{f}_i^1, \mathbf{f}_j^1, \mathbf{x}_k |_{t_0}^t)$ has a value close to 1, there exists a temporal pattern in a channel of $\mathbf{X}$ which fits the concatenation of $(\mathbf{f}_i^1, \mathbf{f}_j^1)$. Accordingly, by using a moderate threshold in (5) (e.g., $r = 0.8$), we collect all the candidate AI-filters $\mathbf{f}_i^1$ in the forward pass (Fig. 1) and form a binary-weighted graph $\mathcal{G}$. In this graph, the filters $\{\mathbf{f}_i^1\}_{i=1}^{d_1}$ are the nodes, those of which correspond to the definition 1 have undirected links of weight -1 between them. Now, we find the existing Abs-filters of different sizes via finding shortest paths between connected nodes of $\mathcal{G}$, which is efficiently solved by the Floyd–Warshall algorithm [15]. So, those Abs-filters which are subsets of the longer ones are detected and eliminated.

After finding $\mathbf{M}$ Abs-filters, for each $p \leq \mathbf{M}$, we apply the created filters $\{\mathbf{f}_i^p\}_{i=1}^{d_p}$ on the channels of $\mathbf{X}$ analogous to the description in Sect. 3.1. Then, the generated activation maps $\{\mathbf{V}^p\}_{p=2}^{\mathbf{M}}$ are added to $\mathbf{V}^1$ to result in the augmented map $\tilde{\mathbf{V}}$ (Fig. 2). An immediate benefit of these long filters is the sparse alignments we obtain in each row of $\tilde{\mathbf{V}}$ (Fig. 2), which enriched the feature extraction part of our framework (Fig. 1) and increases the interpretability of the outcome (Fig. 3).

4 Deep-Aligned CNN

In Sect. 3, we introduced the alignment kernels as the important feature extraction layer of our skeleton-based action recognition algorithm (Fig. 1). Now, we discuss the role and rationale of the remaining parts in the DACNN architecture.

For each real-life skeleton-based motion data $\mathbf{X}$, different data channels (dimension) contain streams of continuous changes in the values of different joint orientations. These values particularly lay in the range of $[0, 2\pi]$ throughout normalization (or in $[\theta_0, 2\pi]$ due to physical limitations). Therefore, it is highly expected to find short subsequences in different dimensions and temporal locations in $\mathbf{X}$ (or long patterns in symmetrical joints), which have similar shapes or curvatures (Fig. 4). A similar characteristic can also be observed in the quaternion representation of the $\mathbf{X}$.

Based on this observation, we can extract the important patterns in dimension of $\mathbf{X}$ by applying each filter $\mathbf{f}_i^p$ to all the channels. As a direct benefit, this structure notably reduces down the network's number of parameters and avoids an unnecessary model complexity.

After deriving the augmented activation map $\tilde{\mathbf{V}}$ (Fig. 2), we feed it to a regular CNN which contains 1-dimensional convolution filters (1D-conv) as shown in Fig. 5. Each deep layer q of the network contains two 1D-conv layers following

Fig. 4 Examples of similar subsequences (red curves) which are found among different body joints and in different temporal locations related to a skeleton-based action

Fig. 5 Architecture of the 1D-CNN unit in the DACNN framework. It maps the augmented alignment map $\tilde{\mathbf{V}}$ (input) to the prediction map $\mathbf{O}$ (output)

a max-pooling layer with the stride of 2. This combination results in a feature map l_q for each deep layer q with the temporal size of $\mathbb{R}^{T/2^q}$. Hence, the data representation becomes more abstract as we go through these layers.

As a complementary choice to Sect. 3.2, we apply the ELU operator [7] to the input of each max-pooling layer as

$$\omega(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x < 0 \end{cases}. \quad (6)$$

while $\frac{d\omega}{dx}|_{x=0} = 1$ lets the gradients flow through the layers of 1D-CNN even if they belong to any saturated filter.

4.1 Prediction Layer

We want our network structure to take inputs of different sizes and also to fit both the segmentation and classification problems. To that aim, we design a convolutional prediction layer (inspired by [31]) to obtain a prediction map $\mathbf{O} \in \mathbb{R}^{C \times T}$ as the output of 1D-CNN (last layer in Fig. 5). Each entry o_{ct} should represent the confidence of the network in assigning the t th time frame of $\mathbf{X}$ to the class c .

To achieve the above, we first compute the score map S_1 by applying a convolution layer with C output channels to the last activation map (l_q in Fig. 5). This process maps the abstract features to a score map of size $C \times \frac{T}{2^q}$. We would like each entry i from the channel c of S_1 to represent a

probability for the downsampled time frame i to be of class c . Hence, the prediction map $\mathbf{O}$ is computed by applying a $\times 2^q$ upsampling on the S_1 score map. The upsampling is performed by using C deconvolution filters with the stride of 2^q [11]. Finally, to calculate the prediction error (cost) of the network, we compare the obtained prediction map $\mathbf{O}$ to the target label matrix $\mathbf{H}$ using a cross-entropies loss function as

$$\text{LOSS} = - \sum_{c=1}^C \sum_{t=1}^T h_{ct} \log(o_{ct}). \quad (7)$$

Therefore, after DACNN is converged to an optimal point, we can predict the label matrix $\tilde{\mathbf{H}}$ of a test data $\mathbf{X}$ as

$$\tilde{h}_{ct} = \begin{cases} 1 & c = \arg \max_c \|1 - o_{ct}\|_2^2 \\ 0 & \text{otherwise} \end{cases} \quad \forall c, t, \quad (8)$$

where $\tilde{h}_{ct}$ determines if an individual time frame t in $\mathbf{X}$ belongs to the activity class c . However, in a classification setting where the whole $\mathbf{X}$ sequence belongs to one class, we identify the class label of $\mathbf{X}$ as

$$c = \arg \min_c \|\mathbf{1}_{1 \times T} - \mathbf{o}_{c,:}\|_2^2, \quad (9)$$

in which $\mathbf{o}_{c,:}$ denotes the c -th row of $\mathbf{O}$. Nevertheless, the training phase of DACNN is the same for both segmentation and classification tasks using the defined cost term **LOSS** in (7). The complete scheme of the DACNN framework while putting all units together is presented in “Appendix.”

4.2 Fine-Prediction Module

According to the structure of 1D-CNN, each pooling layer downsamples its input feature map by the factor of 2. Hence, for a network with q deep layers, the resulted score map S_1 of width $\frac{T}{2^q}$ would be upsampled in one step to provide the prediction map $\mathbf{O}$ with T time frames. This one-step extreme upsampling, especially for large q , results in a large time frame abstraction for the entries of $\mathbf{O}$. Although this abstraction can be appreciated in a single-class activity recognition problem, it reduces prediction resolution of the network for segmentation problems. This issue becomes more problematic especially in segment borders or in alternative activity repetitions.

As a workaround, we employ a specific skip-connection structure via the fine-prediction module of our DACNN framework (Fig. 1) to have a fine-grained prediction map $\mathbf{O}$. As illustrated in Fig. 6, we obtain score maps for the available alignment map (related to learned Abs-filters) and fuse them with the upsampling path of 1D-CNN. More precisely, we start by the largest Abs-filters of length p , such that $p = 2^{\tilde{q}}$, $\tilde{q} \geq 2$. Then, after downsampling their alignment map $\mathbf{V}^p$ by a max-pooling of size $2^{\tilde{q}}$ and applying a

Fig. 6 Fine-prediction module of DACNN. The skip-connections from score maps of specific Abs-filters result in fine-grained segmentations compared to Fig. 5. The alignment map $\mathbf{V}^p$ in the block-diagram corresponds to the largest Abs-filters of size $p = 2^{\tilde{q}}$ s.t. $\tilde{q} \geq 2$. Nevertheless, the fine-grained process will be also extended for the smaller Abs-filters of such size

score-conv layer (with C output channels) we obtain the score map of $\tilde{S}_2 \in \mathbb{R}^{C \times \frac{T}{2^{\tilde{q}}}}$. On the other side, the score map S_1 of a q -layer 1D-CNN would be upsampled throughout deconvolutions of stride $2^{(q-\tilde{q})}$, which results in an intermediate score map $\tilde{S}_1$ with the width of $\frac{T}{2^{\tilde{q}}}$. Hence, $\tilde{S}_1$ would be added to S_1 to improve its current resolution.

As the rationale, each $\mathbf{V}^p$ has a sparse activation map with gap intervals of pt between the extracted matching patterns, especially when $p \geq 2$ (e.g., Fig. 3). Hence, p -factor downsampling preserves the essential information existing in $\mathbf{V}^p$ while it still increases the resolution of $\mathbf{O}$ by order of $2^{(q-\tilde{q})}$ when $\tilde{q} < q$. Additionally, this skip-connection module intensifies the role of the AI-filters in the prediction task due to the weight sharing between them and the Abs-filter, which increases their discriminative quality.

The above process will be extended by also using other alignment maps $\{\mathbf{V}^p, p = 2^{\tilde{q}}\}_{\tilde{q}=2}^q$ by applications of the downsampling and score convolution on each of these $\mathbf{V}^p$ maps (in parallel passes) to obtain their corresponding score maps $\tilde{S}_i, i = 1, \dots, \tilde{q}$. On the other side, S_1 will be upsampled sequentially by the factor of 2 (after the first $2^{(q-\tilde{q})}$ upsampling) to provide the score maps S_i with temporal resolutions corresponding to the above obtained $\tilde{S}_i$ maps.

4.3 Incremental Depth Increase

According to Fig. 5, the depth of 1D-CNN (number of its deep layers) defines the extent to which the temporal abstractions should be made through its layers. More precisely, the degree of this abstraction should coincide with the temporal length of the distinctive patterns in the dataset, which, on the other hand, has a close relation to the length of the learned Abs-filters $\mathbf{f}^p$. To benefit from this relationship, we add the incremental depth increase (IDI module) to DACNN framework (Fig. 1). This module increases the depth of 1D-CNN network as the training phase learns Abs-filters with bigger lengths.

Assume 1D-CNN has q deep layers in the current epoch, meaning that $S_1 \in \mathbb{R}^{C \times \frac{T}{2^q}}$ (Fig. 5), and the biggest Abs-filter is $\mathbf{f}^p$ where $p < 2^{(q+1)}$. Now, if in the next forward pass of DACNN we find an Abs-layer $\mathbf{f}^{\hat{p}}$ where $p = 2^{(q+1)}$, we construct the layer $q+1$ by replacing the first score-conv of 1D-CNN by another stack of 1D-conv and max-pooling layers followed by a new score-conv layer of the proper size. Hence, the new score map becomes $S_1 \in \mathbb{R}^{C \times \frac{T}{2^{(q+1)}}}$.

However, before connecting the $(q+1)$ -th layer to 1D-CNN, the IDI module first pretrains the initial weights of its layers in an isolated backpropagation loop (Fig. 7). Considering $\tilde{S}_1$ as the initial score map of layer $q+1$ and $\hat{S}_1$ as its factor-2 upsampled map, we train the weights of the $(q+1)$ th layer to minimize the following cost function:

$$\text{err}(q, q+1) = \|S_1 - \hat{S}_1\|_F^2, \quad \|\cdot\|_F : \text{Frobenius norm}, \quad (10)$$

where $\text{err}(q, q+1)$ indirectly indicates the current distance between the dynamic states of the new layer and DACNN.

We perform the above backpropagation in parallel to the main training loop of DACNN until $\text{err}(q, q+1)$ becomes sufficiently small. Afterward, the weights of layer $q+1$ are getting updated by the main backpropagation loop of DACNN. Figure 10 provides an example of the smooth learning curve resulted from using IDI module.

4.4 Sparse Activation Maps

As a crucial observation in SBARS tasks, it is rare for a temporal pattern to occur in many channels of $\{\mathbf{x}_i\}_{i=1}^{n_f}$ at a any given time frame t . Even in synchronized motions, this may happen only in 2–4 joints (e.g., simultaneously raising both hands or both legs). In addition, in each action segment, a specific temporal pattern may occur for a small number of times in each $\mathbf{x}_i$ channel. Even by considering the temporal repetitions of one action, this number would be neglectable compared to T as the temporal length of $\mathbf{x}_i$.

Therefore, we apply a sparsity objective $Sp = \sum_{p=1}^M \|\mathbf{V}^p\|_1$ to the backpropagation of DACNN to compute the optimal value of $\mathbf{f}^1_i$ as

$$\mathbf{f}^{1*}_i = \arg \min_{\mathbf{f}^1_i} \text{LOSS}(\mathbf{f}^1_i) + \lambda \text{Sp}(\mathbf{f}^1_i), \quad (11)$$

in which λ is the scalar which controls the sparsity gain in the training phase. We can reformulate $\text{Sp}(\mathbf{f}^1_i) = \sum_{t,k} \hat{v}_{tki}$, where $\hat{v}_{tki}$ considers the entries in all $\{\mathbf{V}^p\}_{p=1}^M$ in which $\mathbf{f}^1_i$ is involved. Consequently, we optimize Eq. (11) based on the following chain rule:

$$\frac{\partial \text{LOSS}(\mathbf{f}^1_i)}{\partial \mathbf{f}^1_i} = \sum_{t,k} \left(\frac{\partial \text{LOSS}(\mathbf{f}^1_i)}{\partial \hat{v}_{tki}} + \lambda \right) \frac{\partial \hat{v}_{tki}}{\partial \mathbf{f}^1_i}, \quad (12)$$

where $\frac{\partial \hat{v}_{tki}}{\partial \mathbf{f}^1_i} = -2g(\mathbf{x}_j|_t, \mathbf{f}^1_i)(\mathbf{x}_j|_t - \mathbf{f}^1_i)$. In practice, the sparsity term Sp redirects the training path toward the efficient use of the resources (AI-filters) in finding the most distinctive local patterns in the given data, which can significantly reduce the network's complexity and prevent it from overfitting [6].

5 Segmentation and Recognition Evaluation

In this section, we implement our DACNN framework on 4 well-known action recognition datasets in order to obtain empirical evaluation of its performance in SBARS tasks.

5.1 Datasets

CMU Mocap [8] This human motion capture dataset is an extensive collection of different human activities which are recorded by the Vicon system. The data is collected from 144 subjects each of which performs a sequence of different actions in separate sessions. Therefore, our task is to segment and recognize the performed actions in each individual session. We collect the sessions which contain actions from 15 highly observed categories in the whole dataset. These categories include human typical movements such as *walk*, *punch*, *wave*, *run*, *jump*, and *raise*. For the segmentation task, we treat the rest of action classes as blank spaces (gaps). The training set contains 75% of the sessions, while the rest is left for testing.

Montalbano V2 dataset [12] This dataset is related to the “ChaLearn Looking at People” challenge which is recorded with Kinect technology as described by [12]. It includes 13,858 samples of 20 different Italian sign gestures which are recorded in continues sequences. The task of this challenge is to perform SBARS for sign gestures based on the given test and training streams.

SYSU-3D Human-Object Interaction dataset (SYSU) The SYSU [16] contains 12 different action classes recorded in 480 video sequences. SYSU dataset is captured from 40 human subjects, and in each time frame, it represents the 3D coordinates of 20 body joints. We create the test and

Fig. 7 IDI module which increases the deep layers of 1D-CNN while measuring the error of this extension

train batches by splitting the number of subjects in half over 10 randomized splits.

NTU-RGB+D Dataset (NTU) [38] This action recognition dataset consists of 60 classes of actions captured from 40 human subjects. It has 56,000 sequences with 4 million frames in total, and the recorded data of 25 main body joints are used for the action recognition task. There are two typical evaluation protocols for this benchmark: the cross-subject (CS) recognition task which uses data of 20 subjects for training and the rest for testing and the cross-view (CV) task in which the recorded samples from cameras 2 and 3 constitute the training set and the rest is preserved for the test set.

5.2 Implementation Details

In the implementation of DACNN, we choose the kernel size of $t = 3$ for AI-filters and 1×3 for the conv-filters. This choice of t for AI-filters provides them enough angular freedom to learn any local curvature in the data. We also apply the dropout [42] of rate 0.65 to prevent DACNN from overfitting. We train DACNN using the Adam approach [23], and the skeleton data are normalized per dimension which allows the filters to be aligned to local parts of different joints. Regarding the implementation of the baseline algorithms, either we use their publicly available codes and tune their parameters with cross-validation on training set, or we refer to their reported results in the relevant publications. For each dataset, we use its typical evaluation setting reported in the literature. Implementation details of DACNN are available in “Appendix.”

5.3 Action Segmentation Results

In this set of experiments, we evaluate our designed DACNN framework with respect to the segmentation task. The segmentation task is applied to CMU Mocap and Montalbano V2 datasets, for which we predict the label of each time frame in the input motion $\mathbf{X}$ using (8). The action segmentation performance of each algorithm would be evaluated by using the Jaccard index

$$JI = \frac{\tilde{\mathbf{H}} \cap \mathbf{H}}{\tilde{\mathbf{H}} \cup \mathbf{H}}. \quad (13)$$

CMU Mocap The segmentation performance of DACNN on CMU Mocap dataset is evaluated by its comparison to SSSM [24] and HACA [53] as unsupervised temporal segmentation approaches, and to ConvS2S [13], End2End [32], LSTM-CRF [26], and RNN-CRF [48] as the deep sequence labeling frameworks. In the latter group, we replace the word embedding parts with the quaternion values of the time frames. As reported in Table 1, the performances of deep learning algorithms have substantial distances from the unsupervised

Table 1 Segmentation accuracy for Montalbano V2 and CMU Mocap based on Jaccard Index (JI)

Montalbano		CMU Mocap	
Method	JI	Method	JI
YNL [12]	27.1	HACA [53]	71.4
Terrier [12]	53.9	SSSM [24]	75.7
Quads [12]	74.6	LSTM-CRF [26]	85.9
Ismar [12]	74.7	End2End [32]	88.4
Gesture labeling [5]	78.4	RNN-CRF [48]	88.1
CNN+LSTM2 [34]	79.5	ConvS2S [13]	86.4
Moddrop [33]	83.3		
DACNN (proposed)	87.2	DACNN (proposed)	92.5

The best result (bold) is according to a two-valued t-test at a 5% significance level

Fig. 8 Segmentation results on the CMU dataset (Subject 86, trial 03). *GT* ground truth segments. *DACNN-nf* DACNN framework without the fine-prediction unit

methods. Compared to them, DACNN outperforms the best method (End2End) with a notable margin of 4.1%. This result supports the effectiveness of the DACNN architecture regarding the supervised segmentation of skeleton-based action data.

In Fig. 8, the segmentation results of some baselines are visually evaluated on one of the challenging sequences from the CMU dataset (Subject 86, trial 03). In comparison with the unsupervised method SSSM, the supervised algorithms better identify the gaps (insignificant actions) because they are optimized by the relevant label information in the training phase. Additionally, SSSM finds sub-clusters in some of the segments (e.g., *walk* and *kick*) which is not the desired outcome in supervised temporal segmentation.

The algorithms DACNN and DACNN-nf (without fine-prediction unit) have fewer mistakes in the results, especially in the gap areas. Besides, DACNN obtains less overlapping between the segments and better predictions in the gap areas compared to DACNN-nf, which emphasizes the positive effect of the fine-prediction module in DACNN. In comparison, End2End has lower performance than DACNN and DACNN-nf regarding the gap areas and segments.

Montalbano V2 dataset For this dataset, we compare DACNN to the following baselines regarding the temporal segmentation accuracy: YNL [12], Terrier [12], Quads

[12], Ismar [12], Gesture Labeling [5], Moddrop [33], and the CNN+LSTM2 algorithm [34]. The majority of these algorithms need to perform a separate segmentation on the data prior to the training phase, but the DACNN's training is independent of that preprocessing phase. According to Table 1, our proposed method obtained higher performance than the baselines (2.9% margin in the Jaccard index), while the best alternative approach (Moddrop) has a relevantly complex network architecture and uses three different modalities (video, Mocap, and audio).

5.4 Action Recognition Results

To empirically evaluate our DACNN algorithm for action recognition tasks, we evaluate it on SYSU and NTU datasets in a classification setting. Hence, for each test data sample $\mathbf{X}$, the network predicts its corresponding class label c via (9). The recognition performance of the utilized algorithms is evaluated based on the classification accuracy

$$\text{Acc} = 100 \times \frac{\text{\#correctly classified samples}}{\text{\#all test samples}}. \quad (14)$$

SYSU dataset For the empirical evaluations, we compare our DACNN algorithm to other baselines including CNN+DPRL [43], ST-LSTM+Trust Gate [28], Dynamic Skeletons [16], LAFF(SKL) [17], SR-TSL [40], VA-LSTM [50], and GCA-LSTM [29], which includes the most recent deep learning applications (CNN, LSTM, etc.) on this dataset. In Table 2, our proposed DACNN framework outperforms the state of the art with a 2.3% margin.

NTU dataset For the NTU dataset, we evaluate DACNN in comparison to the state-of-the-art methods from the literature: HBRNN-L [10], Dynamic Skeletons [16], LieNet-3Blocks [18], Part-aware LSTM [38], ST-LSTM+Trust Gate [28], CNN+LSTM2 [34], Two-Stream RNN [45], STA-LSTM [41], GCA-LSTM (step-wise) [29], Clips+CNN+MTLN [22], View invariant [30], CNN+DPRL [43], VA-LSTM [50], ST-GCN [47], CNN+LSTM [27], Two-Stream CNN [9], and SR-TSL [40]. As we can see in Table 3, DACNN did not beat SR-TSL

Table 2 Recognition accuracy (%) for SYSU-3D dataset

Method	Acc.	Method	Acc.
LAFF(SKL) [17]	55.2	CNN+DPRL [43]	76.7
Dynamic Sk. [16]	75.2	GCA-LSTM [29]	78.6
ST-LSTM+TG [28]	76.7	VA-LSTM [50]	77.8
SR-TSL [40]	82.0	DACNN (proposed)	84.3

The best result (bold) is according to a two-valued t-test at a 5% significance level

Table 3 Recognition accuracy (%) for NTU dataset regarding cross-view (CV) and cross-subject (CS)

Method	CS	CV	Method	CS	CV
HBRNN-L [10]	59.1	64.0	Clips+CNN [22]	79.6	84.8
Dynamic Sk. [16]	60.2	65.2	View invariant [30]	80.0	87.2
LieNet-3Blocks [18]	63.1	68.4	CNN+DPRL [43]	82.3	87.7
Part-aware LSTM [38]	62.9	70.3	VA-LSTM [50]	79.5	87.9
CNN+LSTM2 [34]	67.5	76.2	ST-GCN [47]	81.5	88.3
ST-LSTM+TG [28]	69.2	78.7	Two-Stream CNN [9]	83.1	89.1
Two-Stream RNN [45]	72.1	79.7	CNN+LSTM [27]	82.9	91.0
STA-LSTM [41]	74.1	81.8	SR-TSL [40]	84.8	92.4
GCA-LSTM [29]	76.3	84.5	DACNN (proposed)	83.8	90.7

The best result (bold) is according to a two-valued t-test at a 5% significance level

and CNN+LSTM in recognition accuracy. Nevertheless, it still achieves a competitive result compared to other recent state-of-the-art algorithms, such as ST-CGN, CNN+DRPL, and VA-LSTM, and even outperforms CNN+LSTM in the CS settings. It is important to notify that the NTU dataset is recorded in a very constrained experimental setting, which is an advantage for the methods which considerably rely on the spatial processing of human poses (such as SR-TSL and CNN+LSTM).

6 Further Empirical Evaluations

In this sections, we also analyze your DACNN framework from other empirical perspectives. Mainly, our concern is to study the interpretation of the Abs-filters, the performance of IDI module, and the effect of sparsity regulation. We also investigate the role of different modules of the DACNN architecture and their effect on the outcome.

6.1 Interpreting the Abs-Filters

Apart from the action recognition and segmentation performance, it is interesting to visualize the learned Abs-filters of DACNN in order to investigate any semantic pattern among them. Relevantly, another strength of DACNN compared to other deep neural networks is its simplicity in visualization and interpretation of its trained filters (Abs-filters). Specifically, we associate filter $\mathbf{f}_i^p$ to class c if

$$c = \arg \max_c \sum_{\mathbf{X} \in \text{class } c} \|\mathbf{v}^p(i, :)|_{\mathbf{X}}\|^2, \quad (15)$$

Fig. 9 Visualization of the Abs-filters learned by DACNN on the NTU dataset and the classes to which they are mostly related. Red links indicate the body parts in which the Abs-filters have high alignment values

Fig. 10 Effect of IDI module in the learning curve of DACNN on Montalbano V2. The squares show the epochs when a new deep layer is initialized, and the circles indicate when the IDI module finishes the pretraining of that layer and adds it to the main loop of DACNN

in which $\mathbf{v}^p(i, :)|_{\mathbf{X}}$ denotes the i -th row of the alignment map $\mathbf{V}^p$ after applying $\mathbf{f}^p_i$ on the dimensions of $\mathbf{X}$. In Fig. 9, we visualize some of the Abs-filters learned by DACNN after being trained on the NTU dataset. These filters are mostly related to the action classes *walking*, *waving*, *sitting*, *jumping*, and *throwing*. It is clear that each filter has learned a semantic subsequence from one joint of the whole action. For instance, one Abs-filter is aligned with the movement of the *left foot* in the *jumping* action before the subject leaves the ground, or another filter has learned half of a *walking* cycle on the *right foot*. Considering other Abs-filters which are illustrated in Fig. 9, each of them has recognized

a specific temporal pattern which facilitates the separation of that class from others.

6.2 Incremental Layer Extension

We investigate the individual effect of the IDI module on training performance via the implementation of DACNN on the Montalbano V2 dataset. As shown in Fig. 10, we initialize DACNN with 2 deep layers in its 1D-CNN module (depth 2). Then, as the network constructs the Abs-filters during the training, the IDI module increases its depth incrementally until this progress becomes saturated at a depth of 4.

Figure 10 also contains the accuracy curve related to the training of DACNN-5 and DACNN-direct without IDI modules. DACNN-5 has a fixed depth of 5 during its training, while we incrementally increase the depth of DACNN-direct, but without the pretraining (Fig. 7) of its new layers.

According to Fig. 10, although DACNN and DACNN-5 both converge to the same accuracy performance, DACNN (with IDI module) presents a notably faster convergence and also chooses only up to 4 deep layers as the necessary level of complexity for training DACNN on this dataset. Also, the trained DACNN-5 learns Abs-filters up to the size of $54 = 3 \times 18$. However, the minimum filter size which triggers the initiation of depth 5 is 3×2^5 . Hence, DACNN-5 couldn't find any discriminative pattern that is large enough to justify the necessity of having the 5th depth in its structure. Furthermore, DACNN-direct shows dramatic decreases in its learning curve each time a new deep layer is added to its network, which is due to the disturbance the new untrained initial weights cause in training.

6.3 Role of Sparsity

To better study the role of the sparsity loss $\mathcal{S}_p$ in the recognition performance, we repeat the experiment on the NTU dataset with varying the parameter λ of Eq. (11) in the range [0 1]. Based on Table 4, this term has a positive effect on the accuracy and convergence speed when λ is in the range [0.2 0.5]. However, very large or minimal values of λ lead to low accuracy and slow convergence, respectively.

Table 4 Effect of the sparsity parameter λ on accuracy (%) and convergence epoch (C.ep.) for the NTU dataset

λ	Acc.	C.ep.	λ	Acc.	C.ep.	λ	Acc.	C.ep.
0	89.3	310	0.4	90.4	231	0.8	87.8	176
0.1	89.8	289	0.5	90.0	210	0.9	87.7	169
0.2	90.5	273	0.6	88.4	195	1.0	86.5	161
0.3	90.7	254	0.7	88.1	190	–	–	–

The best result (bold) is according to a two-valued t-test at a 5% significance level

Table 5 Ablation study: prediction accuracies (%) for partial implementations of DACNN on three selected datasets

Method	SYS-3D	NTU CS	NTU CV	Mont.
DACNN-1D	71.8	70.5	75.5	70.54
DACNN-AI	78.4	77.4	85.3	82.53
DACNN-nf	82.5	81.8	88.9	84.15
DACNN (Fig. 1)	84.3	83.0	90.7	85.2

6.4 Ablation Study

To study individual roles of different modules in DACNN (Fig. 1), we perform an ablation study by repeating the recognition experiments (Sect. 5.4) for the variants of DACNN framework (Table 5). In DACNN-nf, we remove the fine-prediction module, and we do not employ any Abs-filter in DACNN-AI. The DACNN-1D network is similar to DACNN-AI, but it uses 1D-conv filters instead of the AI-filters. According to the results in Table 5, the lower performance of DACNN-nf compared to DACNN shows the positive effect of the Abs-filter skip-connections (Fig. 6) in improving the accuracy of the final prediction in 1D-CNN.

Nevertheless, the accuracy of DACNN-nf is still close to DACNN and is even higher than the state of the art for SYS-3D and Montalbano. Removing the Abs-filters from DACNN causes a notable decrease in the performance of DACNN-AI, which justifies the significant role of these filters in extracting the key patterns in the data. However, DACNN-AI obtains higher accuracies than DACNN-1D which proves how effective the alignment filters are regarding the SBARS problems. We already demonstrated the individual effects of the fine-prediction unit (in segmentation) and IDI module in Figs. 8 and 10, respectively.

6.5 Stability of DACNN

The DACNN framework has a stable training phase due to its specific structure. The 1D-CNN module has a similar structure to the fully convolutional neural networks [31], and the alignment layer contains convex operation units which are fully differentiable. Hence, the main body of DACNN follows a routine training procedure similar to other typical CNN networks. One example of the learning curve is provided in Fig. 10 which studies the training progress of the network with and without the IDI module.

Regarding the construction of the Abs-filters, fine-prediction layers, and the IDI module, it is crucial to notice that we designed these modules to become functional relying on some AI-filters which are trained sufficiently and reached stable regions of the optimization space. For instance, a portion of $\{\mathbf{f}_i^1\}_{i=1}^{d_1}$ can possess specific patterns inside the data, but they need to be fine-tuned. In particular, the IDI module starts to learn the initial parameters of a new deep layer when DACNN already reached the low-slope part of its learning curve (Fig. 10). Additionally, the new deep layer has fewer parameters to update than main DACNN framework which makes their notably fusion fast and smooth.

7 Conclusions

In this work, we proposed a deep-aligned convolutional neural network for skeleton-based action recognition and temporal segmentation. This network is constructed upon introducing the novel concept of temporal alignment filters for CNNs. These filters are efficient choices for classification of skeleton-based motion data compared to regular convolution filters. They extract crucial local patterns in the temporal dimensions of the data to better discriminate the action classes. Besides the competitive performance of our DACNN framework compared to the state of the art, its extracted features (learned Abs-filters) are easily interpretable regarding their semantic contents. Furthermore, we designed an IDI module to smoothly increase the depth of our network according to the data structure and without disrupting the training process. Our empirical evaluation of DCANN on different SBARS benchmarks supports our claims regarding the performance and benefits of our network. We believe that the idea of incorporating alignments in CNNs can be further studied in other relevant areas such as relevance analysis and generative adversarial networks.

Acknowledgements This research was supported by the Center of Cognitive Interaction Technology “CITEC” (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG).

Author Contributions BHM guided this research, BHS wrote the manuscript, performed the experiments, and conducted the analysis, and RM contributed to the design of the Abs-filter formation algorithm.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in

Fig. 11 Detailed description of the DACNN framework including all of its constituents (better to be seen in colors)

the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Appendix

By putting all the building units of Fig. 1 together, we provide a complete description of the framework in Fig. 11. Nevertheless, to find the design information or rationale for each specific parts in Fig. 11, please read the relevant subsection in Sects. 3 and 4.

Implementation Hints

We use a warm initialization for the AI-filters before starting the training phase. We assign their values by random choice of t -length subsequences from different $\mathbf{X}$ data sequences. This strategy prevents the filters from becoming saturated initially and results in a convergence speedup. Besides, we split each training set to leave 25% of its sequences for early stopping in the training phase. Additionally, it is practical to perform subsampling for high-resolution inputs. This strategy reduces the required depth of the network. As another solution, it is possible to choose a larger t (e.g., $t \in [5, 7, 11, 13]$) as the basic length

of the AI-filters ($\{\mathbf{f}_i^1\}$) which reduces the required depth in 1D-CNN unit without reducing the resolution of the data. However, the first solution is easier to implement.

Regarding the debugging of the framework, in general, the progress in learning the abs-filters shows whether the network structure suits the complexity of the given task. As a common observation, when no (or limited number of) Abs-filter is formed in the training, it is required to increase the number of used AI-filters (d_i).

Also regarding the initial design of the network, we can start the training without having the sparsity term ($\lambda = 0$) or the IDI module. Without using the IDI module, the length and quality of the learned Abs-filter can be checked manually to see if the learning progress is sensible, especially regarding the size of convolutional layers in 1D-CNN and the number of AI-filters used in the alignment layer of DACNN. Then, as the next step, we add the IDI module and the sparsity term in the loop to let DACNN grow its depth appropriately. Generally, we advise obtaining a satisfactory result without the sparsity term and tuning λ afterward to better improve the outcome. Although the above guideline is not always the most optimal step for tuning and debugging, it provides a straightforward routine to initialize the experiments for any new dataset.

Fig. 12 Value of the activation function $g(\mathbf{x}, \mathbf{f})$ for AI-filter of length 2 (a) and the original and modified gradients of a 1-dimensional $g(\mathbf{x}, \mathbf{f})$ (b). The original gradient (16) becomes zero for large values of $\|\mathbf{x} - \mathbf{f}\|_2^2$, but then its modified version (4) prevents the vanishing issue when the gradient tends to fade

Implementation Detail for Abs-Filters

The Abs-filters in our network share weights with their constituent AI-filters $\{\mathbf{f}_i^1\}_{i=1}^{d_1}$. For instance, if $\mathbf{f}_5^3 = [\mathbf{f}_4^1 \mathbf{f}_3^1 \mathbf{f}_1^1]$, in the backpropagation phase, the weights of $\{\mathbf{f}_4^1, \mathbf{f}_3^1, \mathbf{f}_1^1\}$ are updated instead of $\mathbf{f}_5^3$.

To have the same temporal length (T) for all alignment maps $\{\mathbf{V}^p\}_{p=2}^M$, we apply a zero padding of size $\frac{pt}{2}$ for each filter with the receptive field of size pt and put zeros for the remaining $\frac{pt}{2} + 1$ entries. This specific zero-padding gives the chance of checking the alignment between a filter and an unfinished input pattern at the end-border of input sequence.

In practice, the alignment maps $\{\mathbf{V}^p\}_{p=2}^M$ have different dimensions of $d_p \times T$, which makes the convenient summation of the alignment filters difficult. Therefore, in the sub-fusion unit of Fig. 2, we add the content of $\mathbf{v}^p(i, :)$ to the entries of $\mathbf{v}^1(s, :)$ if $\mathbf{f}_i^p$ has a form of $[\mathbf{f}_s^1 \dots]$, where $\mathbf{v}^p(i, :)$ denotes the i -th row of the alignment map $\mathbf{V}^p$ resulted from $\mathbf{f}_i^p$. By doing the same for all rows of $\{\mathbf{V}^p\}_{p=2}^M$, we obtain the augmented alignment map $\tilde{\mathbf{V}}$ in Fig. 2. In fact, if there is a significant pattern in the time frame t_0 of the data which closely matches $\mathbf{f}_i^p$, we should observe a relevantly large peak (near 1) in both $\mathbf{v}^p(i, t_0)$ and $\mathbf{v}^1(s, t_0)$, but not necessary in the t_0 frames of other rows of $\mathbf{V}^1$ which are related to other constituent AI-filters in $\mathbf{f}_i^p$. Consequently, this form of fusion intensifies the above relationship.

Tackling the Vanishing Gradient Issue

As discussed in Sect. 3.2, the original activation function for the AI-filters in (2) becomes zero when they do not fit any temporal part of the data ($\|\mathbf{f} - \mathbf{x}_j|_{t_0}^t\|_2^2$ is large). This behavior is observed in Fig. 12-a. Also, its gradient

$$\nabla_{\mathbf{f}} g = -2e^{-\|\mathbf{f} - \mathbf{x}_j|_{t_0}^t\|_2^2} (\mathbf{f} - \mathbf{x}_j|_{t_0}^t), \quad (16)$$

becomes zero as well in such cases [(Fig. 12b (blue))]. However, as stated in the paper, by computing $g(\mathbf{x}_j|_{t_0}^t, \mathbf{f}_i^1)$ based on (3) the tale of the activation function becomes saturated at a small negative value $-a$ which allows the filter $\mathbf{f}$ to still become updated in the backpropagation phase.

The gradient of (3) respect to each entry f_i from $\mathbf{f}$ is $\nabla_{f_i} g = -2(1+a)e^{-\|\mathbf{x} - \mathbf{f}\|_2^2} (x_i - f_i)$, and its second derivative respect to f_i is computed as $\nabla_{f_i}^2 g = [-2 + 4(x_i - f_i)^2](1+a)e^{-\|\mathbf{x} - \mathbf{f}\|_2^2}$. The value of $\nabla_{f_i}^2 g$ becomes zero when $(x_i - f_i)^2 = 0.5$ which also corresponds to the extremum points in Fig. 12b (blue). Extending this point to higher-dimensional activation functions, it corresponds to the contour of $\|\mathbf{x} - \mathbf{f}\|_2^2 = 0.5$, and we consider it as a threshold after which update of filter $\mathbf{f}$ becomes difficult due to the vanishing gradient issue. Therefore, in order to prevent this situation, we compensate the vanishing effect by adding a proportional term $-2(\mathbf{x} - \mathbf{f})$ to the gradient, which is equivalent to assuming a regularization term of $-\|\mathbf{x} - \mathbf{f}\|_2^2$ being added to the activation function. Hence, we derive the modified gradient for updating $\mathbf{f}$ as (4) in which the condition is employed based on the considered threshold contour. The extra term $2\sqrt{0.5}\text{sign}(\blacksquare)$ in (4) is used to preserve continuity of the gradient in the switching point. In Fig. 12b, we can compare the original gradient curve of (16) to the modified one in (4). The modified version compensates the fading effect of the gradient values smoothly starting from the point that it tends to decrease dramatically.

References

1. Aggarwal JK, Ryoo MS (2011) Human activity analysis: a review. *ACM Comput Surv CSUR* 43(3):16
2. Aggarwal JK, Xia L (2014) Human activity recognition from 3D data: a review. *Pattern Recognit Lett* 48:70–80
3. Anagnostopoulos A, Vlachos M, Hadjieleftheriou M, Keogh E, Yu PS (2006) Global distance-based segmentation of trajectories. In: *Proceedings of SIGKDD'06*. ACM, pp 34–43
4. Camps-Valls G, Bruzzone L (2005) Kernel-based methods for hyperspectral image classification. *IEEE TGRS* 43(6):1351–1362
5. Chang JY (2014) Nonparametric gesture labeling from multi-modal data. In: *Workshop at ECCV'14*. Springer, pp 503–517
6. Changpinyo S, Sandler M, Zhmoginov A (2017) The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*
7. Clevert DA, Unterthiner T, Hochreiter S (2015) Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*
8. CMU (2007) Carnegie Mellon University graphics lab: Motion capture database. <http://mocap.cs.cmu.edu>
9. Du Y, Fu Y, Wang L (2015) Skeleton based action recognition with convolutional neural network. In: *ACPR'15*. IEEE, pp 579–583
10. Du Y, Wang W, Wang L (2015) Hierarchical recurrent neural network for skeleton based action recognition. In: *Proceedings of CVPR'15*, pp 1110–1118

11. Dumoulin V, Visin F (2016) A guide to convolution arithmetic for deep learning. arXiv preprint [arXiv:1603.07285](https://arxiv.org/abs/1603.07285)
12. Escalera S, Baró X, Gonzalez J, Bautista MA, Madadi M, Reyes M, Ponce-López V, Escalante HJ, Shotton J, Guyon I (2014) Chalearn looking at people challenge 2014: dataset and results. In: Workshop at ECCV'14. Springer, pp 459–473
13. Gehring J, Auli M, Grangier D, Yarats D, Dauphin Y.N (2017) Convolutional sequence to sequence learning. In: Proceedings of ICML'17, pp 1243–1252
14. Gönen M, Alpaydın E (2011) Multiple kernel learning algorithms. JMLR 12(Jul):2211–2268
15. Hougardy S (2010) The Floyd–Warshall algorithm on graphs with negative cycles. Inf Process Lett 110(8–9):279–281
16. Hu JF, Zheng WS, Lai J, Zhang J (2015) Jointly learning heterogeneous features for RGB-D activity recognition. In: Proceedings of CVPR'15, pp 5344–5352
17. Hu JF, Zheng WS, Ma L, Wang G, Lai J (2016) Real-time RGB-D activity prediction by soft regression. In: ECCV'16. Springer, pp 280–296
18. Huang Z, Wan C, Probst T, Van Gool L (2017) Deep learning on lie groups for skeleton-based action recognition. In: Proceedings of CVPR'17. IEEE computer Society, pp 1243–1252
19. Huang Z, Xu W, Yu K (2015) Bidirectional LSTM-CRF models for sequence tagging. arXiv preprint [arXiv:1508.01991](https://arxiv.org/abs/1508.01991)
20. Hussein M.E, Torki M, Gawayyed M.A, El-Saban M (2013) Human action recognition using a temporal hierarchy of covariance descriptors on 3D joint locations. In: IJCAI'13, vol 13, pp 2466–2472
21. Johansson G (1973) Visual perception of biological motion and a model for its analysis. Percept Psychophys 14(2):201–211
22. Ke Q, Bennamoun M, An S, Sohel F, Boussaid F (2017) A new representation of skeleton sequences for 3D action recognition. In: CVPR'17. IEEE, pp 4570–4579
23. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980)
24. Krüger B, Vögele A, Willig T, Yao A, Klein R, Weber A (2017) Efficient unsupervised temporal segmentation of motion data. IEEE TMM 19(4):797–812
25. Lafferty JD, McCallum A, Pereira FCN (2001) Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of ICML'01, pp 282–289
26. Lample G, Ballesteros M, Subramanian S, Kawakami K, Dyer C (2016) Neural architectures for named entity recognition. In: Proceedings of NAACL-HLT, pp 260–270
27. Li C, Wang P, Wang S, Hou Y, Li W (2017) Skeleton-based action recognition using LSTM and CNN. In: ICMEW'17. IEEE, pp 585–590 (2017)
28. Liu J, Shahroudy A, Xu D, Kot AC, Wang G (2018) Skeleton-based action recognition using spatio-temporal LSTM network with trust gates. IEEE TPAMI 40(12):3007–3021
29. Liu J, Wang G, Hu P, Duan LY, Kot AC (2017) Global context-aware attention LSTM networks for 3D action recognition. In: CVPR'17, vol 7, p 43
30. Liu M, Liu H, Chen C (2017) Enhanced skeleton visualization for view invariant human action recognition. Pattern Recognit 68:346–362
31. Long J, Shelhamer E, Darrell T (2015) Fully convolutional networks for semantic segmentation. In: Proceedings of CVPR'15, pp 3431–3440
32. Ma X, Hovy E (2016) End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In: Proceedings of ACL'16, vol 1, pp 1064–1074
33. Neverova N, Wolf C, Taylor G, Nebout F (2016) Moddrop: adaptive multi-modal gesture recognition. IEEE TPAMI 38(8):1692–1706
34. Núñez JC, Cabido R, Pantrigo JJ, Montemayor AS, Vélez JF (2018) Convolutional neural networks and long short-term memory for skeleton-based human activity and hand gesture recognition. Pattern Recognit 76:80–94
35. Petitjean F, Forestier G, Webb GI, Nicholson AE, Chen Y, Keogh E (2016) Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. Knowl Inf Syst 47(1):1–26
36. Poppe R (2010) A survey on vision-based human action recognition. Image Vis Comput 28(6):976–990
37. Rakthanmanon T, Keogh EJ (2013) Data mining a trillion time series subsequences under dynamic time warping. In: IJCAI, pp 3047–3051
38. Shahroudy A, Liu J, Ng TT, Wang G (2016) NTU RGB+D: a large scale dataset for 3D human activity analysis. In: CVPR'16, pp 1010–1019
39. Shotton J, Fitzgibbon A, Cook M, Sharp T, Finocchio M, Moore R, Kipman A, Blake A (2011) Real-time human pose recognition in parts from single depth images. In: CVPR'11. IEEE, pp 1297–1304
40. Si C, Jing Y, Wang W, Wang L, Tan T (2018) Skeleton-based action recognition with spatial reasoning and temporal stack learning. arXiv preprint [arXiv:1805.02335](https://arxiv.org/abs/1805.02335)
41. Song S, Lan C, Xing J, Zeng W, Liu J (2017) An end-to-end spatio-temporal attention model for human action recognition from skeleton data. AAAI 1:4263–4270
42. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. JMLR 15(1):1929–1958
43. Tang Y, Tian Y, Lu J, Li P, Zhou J (2018) Deep progressive reinforcement learning for skeleton-based action recognition. In: CVPR'18, pp 5323–5332
44. Vemulapalli R, Arrate F, Chellappa R (2014) Human action recognition by representing 3D skeletons as points in a lie group. In: CVPR'14
45. Wang H, Wang L (2017) Modeling temporal dynamics and spatial configurations of actions using two-stream recurrent neural networks. In: CVPR'17
46. Wang J, Liu Z, Wu Y, Yuan J (2012) Mining actionlet ensemble for action recognition with depth cameras. In: CVPR'12. IEEE, pp 1290–1297
47. Yan S, Xiong Y, Lin D (2018) Spatial temporal graph convolutional networks for skeleton-based action recognition. arXiv preprint [arXiv:1801.07455](https://arxiv.org/abs/1801.07455)
48. Yang Z, Salakhutdinov R, Cohen W (2016) Multi-task cross-lingual sequence tagging from scratch. arXiv preprint [arXiv:1603.06270](https://arxiv.org/abs/1603.06270)
49. Ye L, Keogh E (2009) Time series shapelets: a new primitive for data mining. In: Proceedings of SIGKDD'09. ACM, pp 947–956
50. Zhang P, Lan C, Xing J, Zeng W, Xue J, Zheng N et al (2017) View adaptive recurrent neural networks for high performance human action recognition from skeleton data
51. Zhang Z (2012) Microsoft kinect sensor and its effect. IEEE Multimed 19(12):4–10
52. Zheng Y, Liu Q, Chen E, Ge Y, Zhao JL (2014) Time series classification using multi-channels deep convolutional neural networks. In: WAIM'14. Springer, pp 298–310
53. Zhou F, De la Torre F, Hodgins JK (2013) Hierarchical aligned cluster analysis for temporal clustering of human motion. IEEE TPAMI 35(3):582–596
54. Zhu W, Lan C, Xing J, Zeng W, Li Y, Shen L, Xie X et al (2016) Co-occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks. In: AAAI, vol 2, p 6