Convolutional Neural Networks for Multivariate Time Series Classification using both Inter- and Intra-Channel Parallel Convolutions

Guillaume Devineau, Wang Xi, Fabien Moutarde, Jie Yang

To cite this version:
Guillaume Devineau, Wang Xi, Fabien Moutarde, Jie Yang. Convolutional Neural Networks for Multivariate Time Series Classification using both Inter- and Intra-Channel Parallel Convolutions. Reconnaissance des Formes, Image, Apprentissage et Perception (RFIA’2018), Jun 2018, Marne la Vallée, France. hal-01888862

HAL Id: hal-01888862
https://minesparis-psl.hal.science/hal-01888862
Submitted on 5 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Convolutional Neural Networks for Multivariate Time Series Classification using both Inter- & Intra-Channel Parallel Convolutions

G. Devineau\(^1\) W. Xi\(^2\) F. Moutarde\(^1\) J. Yang\(^2\)

\(^1\) MINES ParisTech, PSL Research University, Center for Robotics, Paris, France
\(^2\) Shanghai Jiao Tong University, School of Electronic Information and Electrical Engineering, China

{guillaume.devineau, wang.xi, fabien.moutarde}@mines-paristech.fr

Abstract

In this paper, we study a convolutional neural network we recently introduced in [9], intended to recognize 3D hand gestures via multivariate time series classification. The Convolutional Neural Network (CNN) we proposed processes sequences of hand-skeletal joints’ positions using parallel convolutions. We justify the model’s architecture and investigate its performance on hand gesture sequence classification tasks. Our model only uses hand-skeletal data and no depth image. Experimental results show that our approach achieves a state-of-the-art performance on a challenging dataset (DHG dataset from the SHREC 2017 3D Shape Retrieval Contest). Our model achieves a 91.28% classification accuracy for the 14 gesture classes case and an 84.35% classification accuracy for the 28 gesture classes case.

1 Introduction

Gesture is a natural way for a user to interact with one’s environment. One preferred way to infer the intent of a gesture is to use a taxonomy of gestures and to classify the unknown gesture into one of the existing categories based on the gesture data, e.g., using a neural network to perform the classification. In this paper we present and study a convolutional neural network architecture relying on intra- and inter-parallel processing of sequences of hand-skeletal joints’ positions to classify complete hand gestures. Where most existing deep learning approaches to gesture recognition use RGB-D image sequences to classify gestures [41], our neural network only uses hand (3D) skeletal data sequences which are quicker to process than image sequences. The rest of this paper is structured as follows. We first review common recognition methods in Section II. We then present the DHG dataset we used to evaluate our network in Section III. We detail our approach in Section IV in terms of motivations, architecture and results. Finally, we conclude in Section VI and discuss how our model can be improved and integrated into a realtime interactive system. Note that the contents of this paper are highly similar to that of [9], especially sections 1, 2 and 3, as well as the figure illustrating the network, however in this article we focus more on practical tips and on justifying the network architecture whereas the original paper focus was more centered on gesture-related aspects. Readers familiar with [9] can directly skip to the subsection Architecture Tuning of section IV, in which the network architecture is justified more thoroughly.

2 Definition & Related Work

We define a 3D skeletal data sequence \(s\) as a vector \(s = (p_1 \cdots p_n)^T\) whose components \(p_i\) are multivariate time sequences. Each component \(p_i = (p_i(t))_{t \in \mathbb{N}}\) represents a multivariate sequence with three (univariate sequences) components \(p_i = (x^{(i)}, y^{(i)}, z^{(i)})\) that altogether represent a time sequence of the positions \(p_i(t)\) of the \(i\)-th skeletal joint \(j_i\). Every skeletal joint \(j_i\) represents a distinct and precise articulation or part of one’s hand in the physical world. In the following subsections, we present a short review of some approaches to gesture recognition. Typical approaches to hand gesture recognition begin with the extraction of spatial and temporal features from raw data. The features are later classified by a Machine Learning algorithm. The feature extraction step can either be explicit, using hand-crafted features known to be useful for classification, or implicit, using (machine) learned features that describe the data without requiring human labor or expert knowledge. Deep Learning algorithms leverage such learned features to obtain hierarchical representations (features) that often describe the data better than hand-crafted features. As we work on skeletal data only, with a deep-learning perspective, this review pays limited attention to non deep-learning based approaches and to depth-based approaches; a survey on the former approaches can be found in [19] while several recent surveys on the latter approaches are listed in Neverova’s thesis [21].

2.1 Non-deep-learning methods using hand-crafted features

Various hand-crafted representations of skeletal data can be used for classification. These representations often describe physical attributes and constraints, or easily interpretable properties and correlations of the data, with an em-
phasis on geometric features and statistical features. Some commonly used features are the positions of the skeletal joints, the orientation of the joints, the distance between joints, the angles between joints, the curvature of the joints’ trajectories, the presence of symmetries in the skeletal, and more generally other features that involve a human interpretable metric calculated from the skeletal data [15, 16, 33]. For instance, in [37], Vemulapalli et al. propose a human skeletal representation within the Lie group $SE(3) \times \ldots \times SE(3)$, based on the idea that rigid body rotations and translations in 3D space are members of the Special Euclidean group $SE(3)$. Human actions are then viewed as curves in this manifold. Recognition (classification) is finally performed in the corresponding Lie algebra. In [8], Devanne et al. represent skeletal joints’ sequences as trajectories in a n-dimensional space; the trajectories of the joints are then interpreted in a Riemannian manifold. Similarities between the shape of trajectories in this shape space are then calculated with $k$-Nearest Neighbor ($k$-NN) to achieve the sequence classification. In [7], two approaches for gesture recognition -on the DHG dataset presented in the next section- are proposed. The first one, proposed by Guerry et al., is a deep-learning method presented in the next subsection. The second one, proposed by De Smedt et al., uses three hand-crafted descriptors: Shape of Connected Joints (SoCJ), Histogram of Hand Directions (HoHD) and Histogram of Wrist Rotations (HoWR), as well as Fisher Vectors (FV) for the final representation. Regardless of the features used, hand-crafted features are always fed into a classifier to perform the gesture recognition. In [5], Cippitelli et al. use a multi-class Support Vector Machine (SVM) for the final classification of activity features based on posture features. Other very frequently used classifiers [40] are Hidden Markov models (HMM), Conditional Random Fields (CRF), discrete distance-based methods, Naive Bayes, and even simple $k$-Nearest Neighbors ($k$-NN) with Dynamic Time Warping (DTW) discrepancy.

### 2.2 Deep-Learning based methods

Deep Learning, also known as Hierarchical Learning, is a subclass of Machine Learning where algorithms $f$ use a cascade of non-linear computational units $f_i$ (layers), e.g., using convolutions, for feature extraction and transformation: $f = f_1 \circ f_2 \circ \ldots \circ f_n$. A traditional Convolutional Neural Network (CNN, or ConvNet) model almost always involves a sequence of convolution and pooling layers, that are followed by dense layers. Convolution and pooling layers serve as feature extractors, whereas the dense layers, also called Multi Layer Perceptron (MLP), can be seen as a classifier. A strategy to mix deep-learning algorithms and (hand) gesture recognition consists in training convolutional neural networks [18] on RGB-D images. A direct example of hand gesture recognition via image CNNs can be found in the works of Strezoski et al. [32] where CNNs are simply applied on the RGB images of sequences to classify. Guerry et al. [7] propose a deep-learning approach for hand gesture recognition on the DHG dataset, which is described in section III of this paper. The Guerry et al. approach consists in concatenating the Red, Green, Blue and Depth channels of each RGB-D image. An already pretrained VGG [29] image classification model is then applied on sequences of 5 concatenated images consecutive in time. In [20], Molchanov et al. introduce a CNN architecture for RGB-D images where the classifier is made of two CNN networks (a high-resolution network and a low-resolution network) whose class-membership outputs are fused with an element-wise multiplication. Neverova et al. carry out a gesture classification task on multi-modal data (RGB-D images, audio streams and skeletal data) in [22, 23]. Each modality is processed independently with convolution layers at first, and then merged. To avoid meaningless co-adaptation of modalities a multi-modal dropout (ModDrop) is introduced. Nevertheless, these approaches use deep information where we only want to use skeletal data. In [38], Wang et al. color-code the joints of a 3D skeleton across time. The colored (3D) trajectories are projected on 2D planes in order to obtain images that serve as inputs of CNNs. Each CNN emits a gesture class-membership probability. Finally, a class score (probability) is obtained by the fusion of the CNNs scores.

Recurrent Neural Networks (RNN), e.g. networks that use Long Short-Term Memory (LSTM) [12] or Gated Recurrent Units (GRU) [4], have long been considered as the best way to achieve state-of-the-art results when working with neural networks on sequences like time series. Recently, the emergence of new neural networks architectures that use convolutions or attention mechanisms [35, 36] rather than recurrent cells has challenged this assumption, given that RNNs present some significant issues such as being sensitive to the first examples seen, having complex dynamics that can lead to chaotic behavior [17] or being models that are intrinsically sequential, which means that their internal state computations are difficult to parallelize, to name only a few of their issues. In [30], Song et al. elegantly combine the use of an LSTM-based neural network for human action recognition from skeleton data with a spatio-temporal attention mechanism. While this approach seems promising, we rather seek to find a convolution-only architecture rather than a recurrent one.

Zheng et al. propose a convolution-based architecture that does not involve recurrent cells in [42], although this architecture can easily be extended with recurrent cells: [25]. Zheng et al. introduce a general framework (Multi-Channel Deep Convolutional Neural Networks, or MCDCNN) for multivariate sequences classification. In MCDCNN, multivariate time series are seen as multiple univariate time series; as such, the neural network input consist of several 1D time series sequences. The feature learning step is executed on every univariate sequence individually. The respective learned features are later concatenated and merged using a classic MLP placed at the end of the feature extraction layers to perform classification. The major
of the hand (i.e. the 3D hand skeleton) only is competitive with other existing approaches that often focus on RGB-D images. The second goal of this paper is to propose a generic neural network that does not require recurrent cells to achieve this recognition.

4.2 Model Architecture

In [9] we introduce a multi-channel convolutional neural network with two feature extraction modules and a residual branch per channel. The whole model architecture is depicted in figure 1. The architecture is inspired by the high-resolution and low-resolution networks from [20]. The use of residual branches in our architecture is inspired from the original Residual Networks paper [11]. Residual branches make networks easier to optimize because of a better gradient backpropagation in the training phase; they empirically improve the accuracy of deep networks.

Our network inputs consist of multiple, fixed-length, 1D sequences \( s_1, s_2, \ldots, s_c \) where \( c \in \mathbb{N}^+ \) is the number of sequences (channels). Each of these sequences \( s_i \) is directly fed to three parallel branches. The first branch, improperly called residual branch in this paper, is almost an identity function. Instead of outputting exactly its input we perform a pooling on the input in order to reduce the risk of overfitting. The second and third branches both present a similar architecture, detailed below, designed for feature extraction. In these two branches, the input is processed as follows. The input is passed to a convolution layer, whose output is subsampled using a pooling layer. This process is repeated two more times. For a single branch, the difference between all the three convolutions resides in the number of feature maps used; the difference between the two branches resides in the size of the convolutions kernels. Having two kernel sizes for the time convolution layers allows the network to directly work at different time resolutions. Formally, let \( h^{(l)}(\beta) \) represent the input of the \( l \)-th convolution layer of the \( \beta \)-branch, \( \mathbf{F}^{(l)}(\beta) \) be the number of feature maps, \( W^{(l)}(\beta) \) the \( k \)-th convolution filter of the \( l \)-th convolution in the \( \beta \)-branch, and \( b^{(l)}(\beta) \) the bias shared for the \( k \)-th filter map. The output \( h^{(l+1)}(\beta) \) of the \( l \)-th convolution layer is calculated as

\[
h^{(l+1)}(\beta) = \sigma \left( h^{(l)}(\beta) * W^{(l)}(\beta) + b^{(l)}(\beta) \right)
\]

where \( \sigma \) is the activation function. This output \( h^{(l+1)}(\beta) \) serves as the input of the pooling layer that directly follows the convolution layer. For a single channel, the outputs of the three branches are concatenated into a single vector. Finally, a multi layer perceptron “merges” the -concatenated- vectors of all the channels together and acts as a classifier. There are as many MLP outputs as the number of gesture classes.

In our experiments, we have two branches (high resolution and low resolution branches) : \( \beta \in [1;2] \). 3 convolution

1. middle branch in figure 1
and pooling layers: \( l \in \{1; 3\} \), and \( K^{(l, \beta)} = 8 \) feature maps for \( l = 1 \) or \( l = 2 \) and \( K^{(3, \beta)} = 4 \) feature maps for \( l = 3 \). The multi layer perceptron has 1 hidden layer with 1996 hidden units. All of the neurons in our network use the ReLU activation function: \( \sigma(x) = \text{ReLU}(x) = \max(0, x) \), with the exception of the output neurons which use the soft-max activation function. All of the 3 \( \{2 \times 2 \times c\} \) subsampling layers use an average pooling with a temporal pool size of 2. Average pooling computes the average value of features in a neighborhood (of 2 time steps in our case), while max pooling extracts the maximum value of the features in the neighborhood. Empirically, it has been shown that max pooling outperforms average pooling in image recognition problems [1]. Nevertheless, experiments we conducted on the choice of the pooling method for our model showed that our model exhibits better results with average pooling (we see a 0.88% decrease in validation accuracy for the model with maximum pooling rather than the average one for the model configuration presented in this paper).

4.3 Architecture Tuning

To justify the model architecture, we study the performance of successive improvements of a MC-DCNN model, created by tweaking its architecture. We iteratively choose the input data format, the activation function used, the number of convolution layers and feature maps, the regularization rate, the pooling method, and finally we introduce a residual and two parallel CNN branches.

Window length. Our model takes a fixed length gesture sequence. In practice, since the hand gestures from the DHG dataset are relatively short, we do not use a sliding window over the sequences as they occur, but rather simply resample them in time to fit in a fixed time window. To determine the length of the window, we start with a simple mono-branch convolution-pooling model (with 4 convolutions instead of 3 in the final model) and evaluate its validation accuracy depending on the input duration after resampling (see table 1). The window length we chose is 100 timesteps.

**Table 1 – Accuracy after training the model for 50 epochs**

<table>
<thead>
<tr>
<th>Input Length</th>
<th>Train Acc.</th>
<th>Validation Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>97.30%</td>
<td>82.80%</td>
</tr>
<tr>
<td>50</td>
<td>98.88%</td>
<td>84.59%</td>
</tr>
<tr>
<td>75</td>
<td>98.42%</td>
<td>86.26%</td>
</tr>
<tr>
<td>100</td>
<td>98.78%</td>
<td>87.10%</td>
</tr>
<tr>
<td>150</td>
<td>98.37%</td>
<td>85.78%</td>
</tr>
<tr>
<td>200</td>
<td>98.01%</td>
<td>84.59%</td>
</tr>
</tbody>
</table>

**Activation function.** While the choice of the rectified linear unit (ReLU) function is a de facto standard in recent deep learning models, each type of activation function has its pros and cons [13]. We thus decided to investigate whether we should use the exponential linear unit (ELU), the rectified linear unit (ReLU) or the sigmoid activation function as the network activation function. As a reminder, these activations are defined as follows: \( \text{ReLU}(x) = x^+ = \max(0, x) \), \( \text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \), and \( \text{ELU}(x) = \begin{cases} \lambda(x^+) - 1 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \), where \( \lambda \geq 0 \) is a hyper-parameter to be tuned.

After training the same model for a limited -but sufficient-

---

**FIGURE 1 – Illustration (borrowed from [9] in which the network architecture is introduced) of the parallel convolutional neural network. Every channel is processed separately before the Multi Layer Perceptron. The parallel feature extraction module presented on the right is not shared between the 66 channels.**
number of epochs and evaluating its performances on the validation set (see table 2), we chose to use the ReLU function as the network activation function. ELU are ReLU yield to similar results, although ELU is more computationally expensive. Overall ReLU reveals to be a good choice, one of the reasons being that it introduces sparsity effect on the network.

<table>
<thead>
<tr>
<th>Activation Function</th>
<th>Training Acc.</th>
<th>Validation Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid</td>
<td>7.30%</td>
<td>6.81%</td>
</tr>
<tr>
<td>ReLU</td>
<td>91.63%</td>
<td>83.51%</td>
</tr>
<tr>
<td>ELU</td>
<td>94.74%</td>
<td>82.68%</td>
</tr>
</tbody>
</table>

TABLE 2 – Accuracy after training the model for 50 epochs

Other activations worth considering in future works include Leaky ReLUs or Parametric ReLUs, as well as the Swish activation function. Parametric ReLUs are similar to ReLUs but cells do not “die”, which can be the case with ReLU due to the backpropagation issue when \( x < 0 \). \( PReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ \mu x & \text{if } x < 0 \end{cases} \)

The coefficient \( \mu \) is learnt during the network training. The Swish activation function has empirically been shown to display better performances than the ReLU activation function [27] and is defined by \( \text{Swish}(x) = x \cdot \text{sigmoid}(\beta x) \).

**Number of layers.** Following a classical convolution-pooling architecture schema, with a single branch and without dropout, we evaluate the performance of the model for a few epochs. The models we evaluate are composed of convolutions with ReLU activations. We denote \( C_{ReLU}(8,5) \) (or simply \( C(8,5) \)) a convolution layer with 8 feature maps and a kernel of width 5, followed by a ReLU activation. We denote \( P(2) \) an average pooling layer of size 2.

Results are presented in table 3. We chose to use a depth of 3 convolution+pooling layers, since it leads to the best (average) validation accuracy. The depth is likely to be related to the sequence length though, and longer sequences may benefit from being processed by a deeper network, if one wanted to change the input length.

<table>
<thead>
<tr>
<th>Model</th>
<th>w/o Residual</th>
<th>w/o High Resolution</th>
<th>w/o Low Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 classes</td>
<td>-1.05%</td>
<td>-0.53%</td>
<td>-1.31%</td>
</tr>
<tr>
<td>28 classes</td>
<td>-5.24%</td>
<td>-6.38%</td>
<td>-4.96%</td>
</tr>
</tbody>
</table>

TABLE 5 – Model acc. degradation on branch ablation

**4.4 Training**

In this section we detail the hyperparameters we used as well as some other information related to the training.

**Weights Initialization & Batching.** Each training batch contained a set of 32 skeletal gesture sequences of length 100, where a skeletal gesture sequence is a list of \( 22 \times 3 = 66 \) unidimensional sequences.

For the training, we used the Xavier initialization (also known as GLOROT uniform initialization) [10] to set the initial random weights for all the weights of our model.
Implementation. Our model was implemented twice using either PyTorch or Keras as a high level library above TensorFlow. CUDA/CuDNN were used for GPU parallelization of the computations.

Hardware. We trained our model on one machine with a GPU (NVIDIA GeForce GTX 1080 Ti). Using the hyper-parameters presented in the paper, each training step took about 12 seconds. We trained the model for 1000 steps.

Loss & Optimizer. We selected negative log-likelihood as the cost function. To train our model, we used the Adam optimization algorithm [14] which calculates an exponential moving average of the gradient and the squared gradient. Two parameters, namely $\beta_1$ and $\beta_2$, are used to respectively weight the first moment (i.e. the mean) and the second moment (i.e. the uncentered variance) of the gradient. For the decay rates of the moving averages we used the parameters $\beta_1 = 0.9$, $\beta_2 = 0.98$. The values of other parameters were $\alpha = 10^{-3}$ for the learning rate, and $\epsilon = 10^{-8}$. All these values are either identical ($\alpha$, $\beta_1$, $\epsilon$) or close ($\beta_2 = \beta_2^{(default)} = 0.999$) to the default values proposed in the Adam article.

4.5 Results on the DHG dataset

We work on the DHG dataset presented on section III. All sequences are preprocessed as described in the previous section. Each resampled skeletal sequence is split into 22 joints’ sequences, and then into 3 × 1D sequences of the (x, y, z) positions of the joints. This leads to 66 = 22 × 3 input sequences that are fed to model we introduced. The outputs of the last layer of the MLP represent the labels we will attribute to the gesture corresponding to the 66-channel input. Since there are two classification tasks, depending on the number of classes, we use two different neural networks with the same architecture, except that one has 14 outputs and the other has 28 outputs.

On the DHG dataset, our model achieves a 91.28% classification accuracy for the 14 gesture classes case and a 84.35% classification accuracy for the 28 gesture classes case. These are the best recognition accuracy scores known for this challenging dataset at this day (see table 6).

In [26, 8, 7, 24, 6] more or less complex but handcrafted features are used in the classification pipelines. The main advantage of deep-learning approaches is to automatically discover such (sometimes complex) features. [3] is based on deep-learning architecture. It directly applies LSTMs - but without applying CNNs beforehand- to the skeletal data (and to the handcrafted features). Introducing a convolution step before the LSTMs could possibly improve the model in [3]. Our model likely uses more efficient representations due to the use of the parallel branches. A comparison of the different approaches is presented in table 6.

4.6 Discussion

Dataset Size. A common barrier to using deep-learning is small datasets. The DHG dataset has roughly 3000 balanced sequence instances (1960 train sequences + 837 test sequences) of 100 timesteps with 66 1D-channels. The proposed model has 13829454 free parameters in total, or 13829454/66 ≈ 209537 free parameters by channel. Given that each of the 1960 training sequences has 100 timesteps, and because of the regularization applied (DropOut with $p = 0.2$), the model likely does not overfit. Qualitatively speaking, no overfitting was observed experimentally. Zero-, One-, or Few-shot learning [39], as well as data augmentation, transfer learning, model compression and distillation techniques can help to reduce the minimum size of the dataset required for training and validating deep learning models.

Preprocessing, Average Pooling & Data Regularity. The input to the network assumes a sequence of poses, which are provided by the Intel RealSense camera. The poses can also be retrieved by using body-worn sensors or estimated by segmenting videos [28]. In that specific case, occlusion issues may arise, leading to wrong joint positions. Data augmentation may help reduce this problem. Invariance by rotation is not taken into account yet by the network.

We re-sampled signals to a vector of size 100 due to the nature of the motions that were all both relatively short as well as all being about the same duration in order of magnitude. This may not hold for motion capture data with very large number of joints.

Table 6 – Accuracy results on the DHG dataset

<table>
<thead>
<tr>
<th>Gesture</th>
<th>Ours</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Ours</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>72.4%</td>
<td>88.9%</td>
<td>82.1%</td>
<td>78.5%</td>
<td>76.53</td>
<td>87.0%</td>
<td>85.9%</td>
<td>84.35%</td>
</tr>
<tr>
<td>T</td>
<td>72.2%</td>
<td>77.0%</td>
<td>74.9%</td>
<td>85.2%</td>
<td>76.53</td>
<td>87.0%</td>
<td>86.1%</td>
<td>79.4%</td>
</tr>
<tr>
<td>P</td>
<td>84.7%</td>
<td>90.9%</td>
<td>87.7%</td>
<td>84.8%</td>
<td>78.5%</td>
<td>87.0%</td>
<td>85.9%</td>
<td>81.28%</td>
</tr>
<tr>
<td>RC</td>
<td>90.9%</td>
<td>78.4%</td>
<td>84.2%</td>
<td>95.4%</td>
<td>90.9%</td>
<td>85.9%</td>
<td>84.1%</td>
<td>91.28%</td>
</tr>
<tr>
<td>SR</td>
<td>91.2%</td>
<td>100.0%</td>
<td>95.4%</td>
<td>95.4%</td>
<td>91.2%</td>
<td>92.3%</td>
<td>88.6%</td>
<td>94.8%</td>
</tr>
<tr>
<td>SL</td>
<td>98.0%</td>
<td>88.9%</td>
<td>93.2%</td>
<td>93.2%</td>
<td>98.0%</td>
<td>85.5%</td>
<td>95.4%</td>
<td>90.9%</td>
</tr>
<tr>
<td>SU</td>
<td>98.2%</td>
<td>79.4%</td>
<td>87.8%</td>
<td>93.9%</td>
<td>98.0%</td>
<td>85.5%</td>
<td>97.8%</td>
<td>92.6%</td>
</tr>
<tr>
<td>SD</td>
<td>99.3%</td>
<td>91.8%</td>
<td>92.6%</td>
<td>93.8%</td>
<td>99.3%</td>
<td>88.0%</td>
<td>94.3%</td>
<td>92.0%</td>
</tr>
<tr>
<td>SX</td>
<td>100%</td>
<td>89.9%</td>
<td>94.3%</td>
<td>95.8%</td>
<td>100%</td>
<td>85.0%</td>
<td>90.1%</td>
<td>95.4%</td>
</tr>
<tr>
<td>Sx</td>
<td>98.3%</td>
<td>100.0%</td>
<td>99.1%</td>
<td>99.1%</td>
<td>98.3%</td>
<td>98.5%</td>
<td>94.1%</td>
<td>94.1%</td>
</tr>
<tr>
<td>SY</td>
<td>90.6%</td>
<td>100.0%</td>
<td>95.1%</td>
<td>93.2%</td>
<td>90.6%</td>
<td>92.0%</td>
<td>92.6%</td>
<td>81.90</td>
</tr>
<tr>
<td>Sh</td>
<td>96.2%</td>
<td>71.4%</td>
<td>82.0%</td>
<td>86.6%</td>
<td>96.2%</td>
<td>81.0%</td>
<td>84.7%</td>
<td>81.90</td>
</tr>
</tbody>
</table>

Table 7 – Comparison of F1 score in the 14 gesture classes case

F1 scores by gesture, for a version of the model without the “residual” branch; very similar results are obtained for the model with the “residual” branch.
variable time spans for which one may prefer encode with a convolution and memorize with an recurrent cell like an LSTM or a GRU.

Recurrence ; Speed. One of the goals of this paper was to study if a convolution-only network could lead to state-of-the-art results for gesture classification. While this result is established for short gestures with limited semantic meaning, the question remains open for gestures with very variable time span. For those cases, re-sampling the input might not always work and it might probably more efficient to insert recurrent cells in the model, e.g. after the convolutions, in order to benefit from the (long time range) context by keeping track of the processed input in a memory. In that case, one should carefully check if the model does not overfit, as memory cells are often harder to regularize [34]. Recurrent cells also tends to significantly increase the training time although there is ongoing work, e.g. [2], to alleviate this issue. LSTMs and GRUs can warp time through their gating mechanism, but since CNNs can have gating mechanisms too, it would be interesting to see if gestures with very variable time span and limited semantic meaning could be efficiently classified without involving any recurrent or auto-regressive mechanism. One of the main advanges of using sparse (skeletal) input data instead of dense (image) input data lies in inference speed. On a (good) Intel Xeon CPU E5-1630 v4 @ 3.70GHz processor, without any GPU, the inference time is as low as $10^{-3}$ s for a batch of 32 gestures, which is several orders of magnitude sufficient for real-time applications, even on less efficient processors for embedded systems.

5 Conclusion

We introduced and studied a new convolutional neural network which classifies (recognizes) hand gestures using skeletal data only. This neural network extends the MC-DCNN framework in several ways. First, it introduces parallel processing branches for each signal. The advantage of two convolutional branches over a single one seems to be that it allows the architecture to access different time resolutions of each signal. Second, the use of residual connection for each signal allows the gradient to better backpropagate in the neural network. Experimentally, it seems to be useful not only regarding the duration of network training, but also in terms of accuracy results. Finally, dropout is also used as a regularization technique. From a neural network perspective, we observe that (intra- and inter-) parallel processing of sequences using convolutional neural networks can be competitive with neural architectures that use cells specifically designed for sequences such as GRU and LSTM cells. We applied our model to perform hand gesture classification on a challenging hand gesture dataset (DHG dataset). Our method outperforms all existing published methods on this dataset. Our model achieves a 91.28% classification accuracy (+3.04% improvement) for the 14 gesture classes case and an 84.35% classification accuracy (+2.45% improvement) for the 28 gesture classes case.

Références
