# Probabilistic Classification of Skeleton Sequences

Jan Sedmidubsky(✉) and Pavel Zezula

Masaryk University, Brno, Czech Republic
{xsedmid,zezula}@fi.muni.cz

**Abstract.** Automatic classification of 3D skeleton sequences of human motions has applications in many domains, ranging from entertainment to medicine. The classification is a difficult problem as the motions belonging to the same class needn't be well segmented and can be performed by subjects of various body sizes in different styles and speeds. The state-of-the-art recognition approaches commonly solve this problem by training recurrent neural networks to learn the contextual dependency in both spatial and temporal domains. In this paper, we employ a distance-based similarity measure, based on deep convolutional features, to search for the *k*-nearest motions with respect to a query motion being classified. The retrieved neighbors are analyzed and re-ranked by additional measures that are automatically chosen for individual queries. The combination of deep features, dynamism in the similarity-measure selection, and a new *k*NN classifier brings the highest classification accuracy on a challenging dataset with 130 classes. Moreover, the proposed approach can promptly react to changing training data without any need for a retraining process.

## 1 Introduction and Related Work

Motion capture (MoCap) data (shortly *motion data*) are sequences of skeletons that consist of 3D positions of human body joints. These spatio-temporal data constitute a model, that on one hand substantially simplifies the view on the complex structure of human motion, but on the other hand enables automated and computer-aided processing. The increasing accuracy and availability of capturing devices have facilitated recording motion data in a variety of application domains, such as military, sports, medicine, law inforcement and smarthomes.

Such application domains require computer-aided operations to analyze the motion data automatically. One of the most fundamental operations is *action classification*, sometimes referred to as *action recognition*. It is the problem of inferring the kind of movement action, based on pre-classified training data. Solving this problem is challenging as the motions of the same kind can be performed by various subjects in different styles, speeds, and initial body postures.

The key part of the classification process is an efficient extraction of robust and sufficiently discriminative features to effectively model the spatial and temporal evolutions of different actions [15]. The survey in [10] categorizes existing

S. Hartmann et al. (Eds.): DEXA 2018, LNCS 11030, pp. 50–65, 2018.
https://doi.org/10.1007/978-3-319-98812-2_4

features into: (1) *joint-based* representations keeping the correlations among 3D joint locations within an action, (2) *mined-joint-based* features learning what subsets of body joints are more informative to discriminate a given action from the other ones, and (3) *dynamics-based* representations employing the advantage of the way the 3D joint locations move over time and modeling an action as a set of 3D trajectories, e.g., implemented by a 2D motion image approximating 3D joint positions by specific colors of the RGB color space [6,13].

The extracted features can then be compared for similarity by distance measures, such as the Dynamic Time Warping [2], to find the most similar training samples with respect to a query being classified. The retrieved samples are analyzed by $k$-nearest-neighbor classifiers to determine the class of the query [13]. On the other hand, the $k$-nearest neighbor ($k$NN) classifiers have been gradually effaced by the increasing success of deep neural networks. Several different neural-network architectures have recently been proposed for motion classification. Specifically, deep *convolutional* networks are trained by the 2D-motion-image features that are also classified by the network [6]. Most attempts suggest to employ the architecture of *recurrent* neural networks to better model the contextual dependency in the temporal domain [8]. This architecture can be enriched by the Long Short-Term Memory (LSTM) to better learn long-term temporal dependencies [9,15,18]. To further handle the noise and occlusion of skeleton sequences, *gating mechanisms* are integrated into LSTM to learn the reliability of the sequential data and accordingly adjust their effect on updating the long-term context information stored in the LSTM memory cell [8]. To benefit from different network architectures at the same time, the combination of convolutional and LSTM networks is proposed [12]. Recently, the recurrent neural networks have been enriched by *attention-based* mechanisms to additionally detect the most discriminative moments within an action [1,9,15].

## Our Contribution

The state-of-the-art motion-data classifiers constitute either purposely-learned neural networks [1,6,9], or 1NN classifiers dependent on a single similarity measure [13,14]. Even if the machine-learning classifiers generally achieve a higher accuracy, they have to be retrained each time when new classes or even only class samples are added, which is a very time-consuming process making the real-time processing prohibited. We plan to overcome this problem by proposing new $k$-nearest-neighbor classifiers that can dynamically react to changing training data, while benefiting from the similarity concept originally learned on a deep convolutional neural network. In particular, we extend the 1NN approach by introducing two new $k$NN classifiers that additionally output a probability distribution over classes to which a query motion should belong. More importantly, we propose a third *confusion-based* classifier that employs the $k$NN-based probability distribution to automatically select more convenient similarity measures for classifying the query motion, rather than relying only on some global similarity. Despite reaching the best accuracy on a challenging dataset with 130 classes, the proposed approach does not require large volumes of training data and can react to data changes immediately in contrast to the recent approaches.

## 2 Representation and Similarity of Skeleton Sequences

A *motion sequence* (or simply *motion*) $M$ is represented by a sequence of consequent *skeletons*. The total number $|M|$ of skeletons determines the motion *length*. The $t$-th skeleton $\in \mathbb{R}^{31\times 3}$ captured at the time moment $t$ $(1 \leq t \leq |M|)$ consists of 3D coordinates of 31 tracked *joints*.

To compare a pair of skeleton sequences, we adopt the state-of-the-art similarity measure which was originally proposed in [14] and improved in [13]. It uses the Euclidean distance to compare $4,096$-dimensional *feature vectors* extracted from motions of variable lengths. A feature vector is extracted from a motion sequence $M$ in the following three steps.

1. *Normalization* – Each skeleton is normalized to become independent of both position and orientation in a 3D space. The skeleton size is additionally scaled to keep the same body proportions over all motions.
2. *Motion-image construction* – The 3D space covering all possible normalized skeletons is then mapped into the RGB color space to approximate any joint position by specific color. The positions of all joints changing over time are projected into a 2D image, so-called *motion image*. This image is generated by the "WeightedJointsFixed" variant [13] to occupy $256 \times |M|$ pixels.
3. *Feature extraction* – The generated motion image is finally processed by the *reference model*[1] of the well-known Krizhevsky's deep convolutional neural network [5] to extract a deep $4,096$-dimensional feature vector $F_M^{4\text{K}}$, as the output of the last hidden network layer.

To achieve a higher descriptive power of feature vectors, the reference model originally trained on common photographs is additionally *fine-tuned* by a training set of motion images, as described in the experiments. The feature extraction and fine-tuning processes are deeply analyzed in [13,14].

The extracted $4,096$D feature vectors $F_{M_1}^{4\text{K}}$ and $F_{M_2}^{4\text{K}}$ of motions $M_1$ and $M_2$ are compared by the Euclidean distance as:

$$compDist^{4\text{K}}\left(F_{M_1}^{4\text{K}}, F_{M_2}^{4\text{K}}\right) = \left\| F_{M_1}^{4\text{K}} - F_{M_2}^{4\text{K}} \right\| .$$

We call this similarity measure as *original* and employ it by all the classifiers proposed in this paper.

## 3 Classification Baseline

We formally define the problem of motion classification and introduce the baseline 1NN classification approach, which has already been evaluated on the motion-image similarity measure in [13].

[1] https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet.

*Classification* is the problem of identifying a single *class* (sometimes referred to as *category*) to which a given motion sequence belongs, based on a set of already categorized motion sequences, i.e., *training data*. Formally, a general classifier $classify$ determining the class of a *query* motion is defined as:

$$classify : \mathcal{M} \rightarrow \mathcal{C}, \tag{1}$$

where $\mathcal{M} = \{M_1, \ldots, M_n\}$ denotes the input domain of $n$ motions, while $\mathcal{C} = \{C_1, \ldots, C_m\}$ is the output domain of $m$ classes.

## 3.1 1NN Classifier

To implement any search-based classifier, there is a need to retrieve the most similar motions – *nearest neighbors* (NN) – with respect to a query motion. The following $1NN$ function searches a training set $\mathcal{M}^{\mathrm{TR}}$ of categorized motion sequences and returns the one that is the most similar to the query motion $M^{\mathrm{Q}}$:

$$\begin{aligned} 1NN\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}\right) = \Big\{ M' \in \mathcal{M}^{\mathrm{TR}} \mid \forall M \in \mathcal{M}^{\mathrm{TR}} : \\ compDist^{4\mathrm{K}}\left(F^{4\mathrm{K}}_{M^{\mathrm{Q}}}, F^{4\mathrm{K}}_{M'}\right) \leq compDist^{4\mathrm{K}}\left(F^{4\mathrm{K}}_{M^{\mathrm{Q}}}, F^{4\mathrm{K}}_{M}\right) \Big\}. \end{aligned} \tag{2}$$

The 1-*nearest-neighbor classifier*, denoted as $classify^{1\mathrm{NN}}$, then simply assigns the query motion the class of the most similar motion:

$$classify^{1\mathrm{NN}}\left(M^{\mathrm{Q}}\right) = getClass\left(1NN\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}\right)\right), \tag{3}$$

where the function *getClass* returns the known class $C_i \in \mathcal{C}\,(i \in \{1, \ldots, m\})$ of motion passed in the argument.

## 3.2 Experimental Evaluation

We first introduce the dataset and methodology that are used for evaluation throughout this paper. Then we present the accuracy results of the 1NN classifier.

**Dataset.** The classification scenario is evaluated on 3D skeletal data of the publicly available HDM05 [11] motion capture dataset. This dataset provides the ground truth HDM05-130, which categorizes 2,345 short motion sequences into 130 categories of specific actions, such as the turn left, sit down on a chair, or clap with hands five times. The average action length is 2.17 s – it corresponds to about 260 frames with the dataset sampling frequency of 120 Hz.

We select this HDM05-130 ground truth because it is very challenging – it contains the highest number of 130 categories when compared to other datasets, such as CMU (30 categories), NTU RGB + D (60 categories), MSR (20 categories) or MHAD (11 categories) [16]. Moreover, it is characterized with a subtle categorization, containing for example separate classes for kicking with left or right leg, to the front or to the side.

As suggested in [13,14], we additionally ignore 8 categories with less than 10 motion samples. Thus our resulting ground truth HDM05-122 contains 122 categories with 2,328 motions in total.

**Methodology.** To be consistent with previous works [6,7,13], we use 50% of motion sequences for training by applying the 2-fold cross validation procedure. In particular, we *randomly* partition all 2,328 motion samples into halves, i.e., into two sets (folds) of 1,164 motions. In the first pass, the first fold is used for training ($\mathcal{M}^{\mathrm{TR}}$) and the second one for testing. In the second pass, the training and test folds are swapped. For each pass, we preprocess the ground truth to:

1. Generate motion images (see Sect. 2) for both training and test motions;
2. Fine-tune the reference model of the neural network by the motion images from the training set;
3. Extract a 4,096D feature vector for each of training and test images based on the fine-tuned network model.

The test motions are then used as query arguments of the $classify^{\mathrm{1NN}}$ classifier. The accuracy of a single-motion classification is either 100%, or 0% based on the fact whether the classified category equals to the category of the query motion. The accuracy of the given pass is then measured as an average accuracy over 1,164 test queries. The accuracy of the classifier is finally expressed as an average over both passes.

**Evaluation of Classification Accuracy.** As already demonstrated in [13], the 4,096D deep features compared by the Euclidean distance achieves the 1NN classification accuracy of 87.84%. This result serves as the baseline for other classifiers proposed in this paper.

## 4 Probabilistic *k*NN Classifiers

Although the 1NN classifier is simple and quite accurate, it needn't be convenient when (1) the query-closest neighbors have almost the same distance while belonging to different classes or when (2) the correct class is confusable with another class, e.g., "grab a thing" with"deposit a thing". In that cases, it is useful to analyze the categories and similarities of more nearest neighbors, rather than relying only on the most similar one. This additionally brings the possibility to determine a probability distribution over a set of classes that the query should belong to. Formally, a *probabilistic classifier* is defined as:

$$classify : \mathcal{M} \rightarrow \{(\mathcal{C} \times [0,1])\}\,. \tag{4}$$

The result of classification is a set of pairs associating the classes with their probabilities of being the correct match. Within this section, we propose two probabilistic classifiers $classify^{\mathrm{kNN}}$ and $classify^{\mathrm{kNN\_TMC}}$.

### 4.1 Weighted-Distance *k*NN Classifier

The idea is to classify the query by a majority vote of its nearest neighbors. We consider not only the number of votes but also the similarity of neighbors with respect to the query.

To obtain the $k$-nearest neighbors from the categorized training set $\mathcal{M}^{\mathrm{TR}}$ to the query $M^{\mathrm{Q}}$, the following $kNN$ function is evaluated:

$$\begin{aligned} kNN\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right) = \Big\{ & M' \in \mathcal{M}' \mid \mathcal{M}' \subset \mathcal{M}^{\mathrm{TR}}, |\mathcal{M}'| = k, \\ & \forall M' \in \mathcal{M}', \forall M \in \mathcal{M}^{\mathrm{TR}}/\mathcal{M}' : \\ & \mathit{compDist}^{\mathrm{4K}}\left(F^{\mathrm{4K}}_{M^{\mathrm{Q}}}, F^{\mathrm{4K}}_{M'}\right) \leq \mathit{compDist}^{\mathrm{4K}}\left(F^{\mathrm{4K}}_{M^{\mathrm{Q}}}, F^{\mathrm{4K}}_{M}\right) \Big\}. \end{aligned} \tag{5}$$

We extend this $kNN$ function by the additional class parameter $C \in \mathcal{C}$ to determine the subset of nearest neighbors that belong to the specified class $C$:

$$kNN(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k, C) = \left\{ M \in kNN(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k) : \mathit{getClass}(M) = C \right\}. \tag{6}$$

**Relevance of Neighbors.** As the $k$-nearest neighbors are retrieved, the *relevance* of each neighbor for classification is determined. This relevance is computed by normalizing the neighbor distance with respect to the distance of the $k$-th neighbor. Such query-dependent normalization is very effective in situations where distances of nearest neighbors vary a lot across different categories, rather than a normalization based on the maximum possible distance. The relevance of the neighbor $M$ is computed by the function $\mathit{compRel} : \mathcal{M} \rightarrow [0, 1]$ as:

$$\begin{aligned} \mathit{compRel}(M) &= 1 - \frac{\mathit{compDist}^{\mathrm{4K}}(F^{\mathrm{4K}}_{M^{\mathrm{Q}}}, F^{\mathrm{4K}}_{M})}{\mathit{kNeighborDist} \cdot \mathit{kNeighborDistWeight}}, \\ \mathit{kNeighborDist} &= \max\left\{ \mathit{compDist}^{\mathrm{4K}}(F^{\mathrm{4K}}_{M^{\mathrm{Q}}}, F^{\mathrm{4K}}_{M}) : M \in kNN\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right) \right\}, \end{aligned} \tag{7}$$

where $\mathit{kNeighborDist}$ is the precomputed distance to the $k$-th neighbor and $\mathit{kNeighborDistWeight} \in [1, \infty)$ is a user-defined parameter determining the importance of distances to classification. We fix this parameter to 1.1 to put a high emphasis on the distances, which at the same time decreases relevance of the $k$-th neighbor a lot.

The result relevance approaching the value 1 denotes a high importance of the specific neighbor and with a decreasing value, the neighbor importance goes down.

**Aggregation of Relevances of Neighbors.** To compute classification probabilities, relevances of neighbors belonging to the same class are summed and finally normalized across all categories. The following *compRels* function returns the pairs associating the summed relevance $r_i$ with the class $C_i$:

$$\begin{aligned} \mathit{compRels}\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right) = \Big\{ & (C_i, r_i) \;\Big|\; C_i \in \mathcal{C}, \\ & r_i = \sum_{M \in kNN(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k, C_i)} \mathit{compRel}(M) \Big\}. \end{aligned} \tag{8}$$

To normalize the relevances into probabilities, the following *normRels* function is employed:

$$normRels\left(\{(C_1, r_1), \ldots, (C_m, r_m)\}\right) = \\ \left\{ (C_i, p_i) \;\middle|\; i \in [1, m], p_i = r_i / \textstyle\sum_{j=1}^{m} r_j \right\}. \tag{9}$$

It transforms the relevance $r_i$ of each class $C_i$ into probability $p_i \in [0, 1]$, computed as a ratio between the class relevance and the sum of relevances of all the classes. Such normalization additionally ensures that the sum of probabilities of all the classes is equal to 1. The classifier returning such probabilities of classes is denoted as *weighted-distance classifier* ($classify^{\mathrm{kNN}}$) and defined as:

$$classify^{\mathrm{kNN}}\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right) = normRels\left(compRels\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right)\right). \tag{10}$$

## 4.2 Training-Class-Sizes *k*NN Classifier

The disadvantage of the previous weighted-distance classifier is that the computed class probabilities can be influenced by different sizes of categories, in terms of the number of training samples. For example, assume that the query $M^{\mathrm{Q}}$ should be assigned to the class $C_1$ and that the classes $C_1$ and $C_2$ contain 1 and 100 training samples, respectively. Then by considering $k = 15$, the 15 nearest neighbors to $M^{\mathrm{Q}}$ are retrieved. Even if the most similar neighbor belongs to the correct class $C_1$, the class $C_2$ is evaluated as the most probable because next 14 neighbors belonging to $C_2$ overweight just one sample. This can arise when the sizes of training classes are not balanced.

In order to avoid such situation, we compute for each class the ratio between the number of its samples being among the $k$-nearest neighbors and the number of the available training samples of that class. The function *updRels* updates the originally-computed relevances $r_i$ by the square root of such ratios in order to calculate new relevances $r'_i$ as:

$$updRels\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right) = \left\{ (C_i, r'_i) \;\middle|\; (C_i, r_i) \in compRels\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right), \right. \\ \left. r'_i = r_i \cdot \sqrt{\frac{|kNN(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k, C_i)|}{|\{M \in \mathcal{M}^{\mathrm{TR}} : getClass(M) = C_i\}|}} \right\}. \tag{11}$$

The classifier considering the sizes of classes is denoted as *training-class-sizes classifier* ($classify^{\mathrm{kNN\_TCS}}$) and formally defined as:

$$classify^{\mathrm{kNN\_TCS}}\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right) = normRels\left(updRels\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right)\right). \tag{12}$$

## 4.3 Evaluation of Classification Accuracy

Both the proposed *k*NN classifiers ($classify^{\mathrm{kNN}}$ and $classify^{\mathrm{kNN\_TCS}}$) compute the probabilities of classes of being the correct match. To evaluate the classification scenario, we consider the class of the highest probability. Figure 1a illustrates the differences between the classifiers depending on an increasing value of

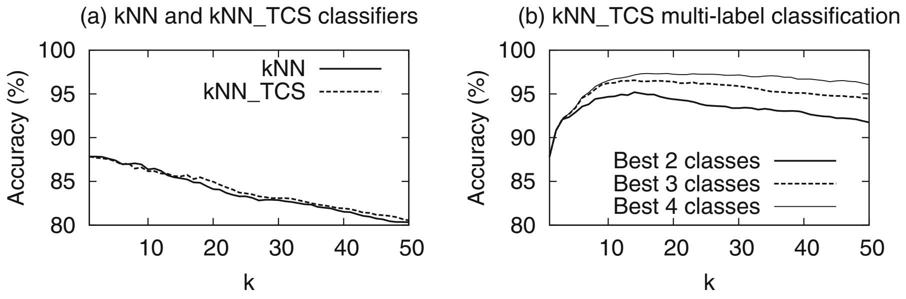

**Fig. 1.** Accuracy of $k$NN classification based on a varying value of $k$: (a) Accuracy of $classify^{\text{kNN}}$ and $classify^{\text{kNN\_TCS}}$ classifiers. (b) Accuracy of the $classify^{\text{kNN\_TCS}}$ classifier when the correct match is among the two/three/four highest-probable classes.

$k$. As expected, the differences are relatively small due to a quite balanced sizes of training classes. In particular, the differences are negligible up to the value $k = 15$, which is the number approaching the average number of class samples. With the value $k > 15$, the sizes of training classes play a more important role, which leads to a slightly better accuracy of the $classify^{\text{kNN\_TCS}}$ classifier.

Both classifiers reach the highest accuracy when $k = 1$ and with an increasing value, their accuracy decreases. Even the results do not outperform the baseline 1NN classifier (the accuracy of 87.84%), the probabilities of individual classes bring new possibilities for enhancements. Based on these $k$NN classifiers, the confusion-based $k$NN classifier is proposed in next section.

## 5 Confusion-Based Classifier

In case there is no clear decision on what class is a query motion, the $k$NN classifiers provide more classes ranked by their probabilities of being the correct match. If we tolerate that the correct match is among the two top ranked classes (instead of considering only the highest-probable class), the classification accuracy can significantly increase. Figure 1b illustrates the $k$NN_TCS classification accuracies when the correct match is included in the two/three/four top ranked classes. We can see that for $k = 15$, the achieved accuracy is higher than 95%, even when only the two top ranked classes are considered.

To significantly improve the accuracy from the baseline value of 87.84% up to 95%, we would need a magic stick choosing the correct class from the two top ranked classes obtained by the 15NN_TCS classifier. The idea of approximating the magic stick is to *re-rank* the $k$-nearest neighbors based on different similarity measures which can better separate the two top ranked classes, than the original measure presented in Sect. 2. Such suitable measures are automatically selected for each pair of classes based on *confusion matrices* learned from the training data. The class of the neighbor with the smallest re-ranked distance is finally considered as the classification result.

### 5.1 Training Phase: Learning Confusion Matrices

To increase the classification accuracy, additional distance measures can be provided. We consider the set $D$ of additional descriptors $\{D_1, \ldots, D_d\}$ to represent each motion $M$ by additional feature vectors $F_M^{D_1}, \ldots, F_M^{D_d}$. For the $i$-th descriptor $D_i$ $(i \in [1, d])$, the similarity of feature vectors $F_{M_1}^{D_i}$ and $F_{M_2}^{D_i}$ of motions $M_1$ and $M_2$ is calculated using a predefined distance measure, denoted as $compDist^{D_i}$.

Within the training phase, the *confusion matrix* $cm^{D_i}$ is calculated for each of the measures. Each matrix has the size of $m \times m$ elements, where $m$ is the number of training classes. The element $cm^{D_i}[C_x, C_y] \in [0, 1]$ of the matrix expresses how much the class $C_x$ is *confusable* with the class $C_y$ $(x, y \in [1, m])$, with respect to the distance measure $compDist^{D_i}$. The value of 0 means that the measure perfectly separates the training motions of both classes and with an increasing value, separability decreases. We compute such value by evaluating the average 1NN classification accuracy over all training motions categorized in both classes. Formally, we compute the value as:

$$cm^{D_i}[C_x, C_y] = \frac{|\{M \in \mathcal{M}^{C_x} \mid getClass(M) \neq classify^{1\mathrm{NN}}(M, \mathcal{M}^{C_{xy}}/\{M\})\}|}{|\mathcal{M}^{C_x}|},$$
$$\mathcal{M}^{C_x} = \left\{M \in \mathcal{M}^{\mathrm{TR}} \mid getClass(M) = C_x\right\}, \quad (13)$$
$$\mathcal{M}^{C_{xy}} = \left\{M \in \mathcal{M}^{\mathrm{TR}} \mid getClass(M) \in \{C_x, C_y\}\right\},$$

where $\mathcal{M}^{C_x}$ denotes the training motions belonging to the class $C_x$, while $\mathcal{M}^{C_{xy}}$ represents motions of both classes $C_x$ and $C_y$.

The way of matrix calculation does not guarantee the symmetry, i.e., it may happen that $cm^{D_i}[C_x, C_y] \neq cm^{D_i}[C_y, C_x]$. Since this is not convenient for further processing, we make the matrix symmetric by considering the maximum value, which can increase the confusion value of both classes:

$$cm^{D_i}[C_x, C_y] = cm^{D_i}[C_y, C_x] = \max\left\{cm^{D_i}[C_x, C_y], cm^{D_i}[C_y, C_x]\right\}. \quad (14)$$

### 5.2 Classification Phase: Analyzing the Nearest Neighbors

Within the classification phase, the $k$-nearest neighbors are retrieved and classified by the $classify^{\mathrm{kNN\text{-}TCS}}$ classifier. The correct match is considered to be among the two top ranked classes. Based on the confusion values of these two classes, the weights of the additionally provided similarity measures are computed. These weights are used to re-rank the nearest neighbors. The whole process is described in the following paragraphs.

**Identifying the Most Ranked Classes.** The two top ranked classes $C'$ and $C''$ $(C', C'' \in \mathcal{C})$ are selected from the $classify^{\text{kNN\_TCS}}$ classification result as:

$$\begin{aligned} get2MostRankedClasses\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right) = \Big\{ C', C'' \ \Big| \\ (C', p'), (C'', p'') \in classify^{\text{kNN\_TCS}}(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k) : p' \geq p'', \\ \forall (C_i, p_i) \in classify^{\text{kNN\_TCS}}(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k) / \{(C', p')\} : p'' \geq p_i \ \Big\} . \end{aligned} \tag{15}$$

**Weighting Similarity Measures.** The most suitable similarity measure(s) for re-ranking should have the lowest (ideally zero) confusion value for the two top ranked classes $C'$ and $C''$ – the lower the value $cm^{D_i}[C', C'']$, the better separation ability of the $D_i$ descriptor. Such lowest confusion value, denoted as $minConf$, can be easily obtained by this formula: $minConf = \min_{i=1}^{d} cm^{D_i}[C', C'']$.

The lowest confusion value is used to determine the *weight* $w^{D_i} \in [0, 1]$ for each descriptor $D_i$. Since there can be a high number of provided descriptors, we select only the best one(s) of the lowest confusion value in order not to suppress important movement features by aggregating many measures. At the same time, the best-available descriptor(s) for the query motion needn't be still optimal and can have a quite low weight, i.e., the two top ranked classes cannot be simply separated by any of the additionally provided measures. For these reasons, we decrease the weight to the power of 3 to more suppress the influence of the best-available measure(s) having "lower" weights. The descriptor weight $w^{D_i}$ is finally determined as:

$$w^{D_i} = \begin{cases} 0 & cm^{D_i}[C', C''] > minConf, \\ (1 - minConf)^3 & cm^{D_i}[C', C''] = minConf. \end{cases} \tag{16}$$

If the best-available measure(s) have a "low" weight, the original motion-image similarity measure is more important. The weight $w^{4\mathrm{K}}$ of such original measure is determined as:

$$w^{4\mathrm{K}} = \max\left\{ \left(1 - cm^{4\mathrm{K}}[C', C'']\right)^3, 1 - (1 - minConf)^3 \right\}. \tag{17}$$

This value is the maximum between the weight computed based on the confusion matrix of the original descriptor and the inverse value of the weight of the best-available additional measure. In other words, the original similarity measure is preferred if it well separates the two top ranked classes or in cases when a strong additional descriptor is not available.

**Re-ranking and Classifying Neighbors.** The calculated weights of the original similarity measure and additional measures are used to re-rank the list of the retrieved neighbors. The re-ranking process is based on weighting and normalizing the distances separately for each descriptor and summing such updated distances together. Formally, to re-rank the neighbor $M$ with respect to the query $M^{\mathrm{Q}}$, the following *reRank* function is applied:

$$\begin{aligned} reRank(M^{\mathrm{Q}}, M) =& w^{4K} \cdot compDist^{4\mathrm{K}}(F^{4\mathrm{K}}_{M^{\mathrm{Q}}}, F^{4\mathrm{K}}_{M})/md^{4\mathrm{K}}[C', C'']+ \\ & \sum_{i=1}^{d} \left( w^{D_i} \cdot compDist^{D_i}(F^{D_i}_{M^{\mathrm{Q}}}, F^{D_i}_{M})/md^{D_i}[C', C''] \right), \end{aligned} \tag{18}$$

where $md^{D_i}$ is the matrix of class-pairwise maximum distances. In particular, the value $md^{D_i}[C', C'']$ is the maximum of distances computed among all pairs of training motions belonging to classes $C'$ or $C''$, with respect to the distance measure $compDist^{D_i}$. Such matrices are simply precomputed for the original motion-image measure (matrix $md^{4\mathrm{K}}$) and each additional measure (matrices $md^{D_i}$) in the training phase as:

$$md^{D_i}[C', C''] = \max \left\{ compDist^{D^i}(F^{D_i}_{M_1}, F^{D_i}_{M_2}) \;\middle|\; M_1 \in C', M_2 \in C'' \right\}. \tag{19}$$

The maximum distances computed separately for each pair of classes more effectively normalize the distances, rather than considering a global maximum. The neighbors are than sorted based on their re-ranked distance and the class of that with the lowest distance is considered as the final classification result. We call this approach as the *confusion-based* classifier $classify^{\mathrm{CONF}}$ and define it formally as:

$$\begin{aligned} & classify^{\mathrm{CONF}}\left(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k\right) = getClass\left(\{\, M' \in kNN(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k) \mid \right. \\ & \left. \forall M \in kNN(M^{\mathrm{Q}}, \mathcal{M}^{\mathrm{TR}}, k) : reRank\left(M^{\mathrm{Q}}, M'\right) \leq reRank\left(M^{\mathrm{Q}}, M\right) \,\}\right). \end{aligned} \tag{20}$$

Due to the concentration on the first re-ranked neighbor only, the confusion-based classifier does not support probabilistic classification. However, it would be possible if the re-ranked list was processed by another $k$NN classifier.

### 5.3 Experimental Evaluation

We describe the additional three measures used for re-ranking, evaluate the accuracy of the confusion-based classifier, and outline the efficiency results.

**Description of Additional Similarity Measures.** To verify the suitability of the confusion-based classifier, we implement the following three simple descriptors whose features are compared by the Manhattan distance, i.e., $L_1$ metric.

- *Joint trajectory length* ($D_1$) – the 31D feature vector, where each dimension corresponds to the total length of the trajectory of the specific joint (out of 31 joints). The trajectory length is computed by summing the Euclidean distances between the 3D joint positions in consecutive skeletons.

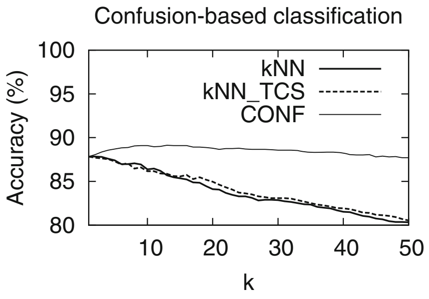

**Fig. 2.** Accuracy between $k$NN-based and confusion-based classifiers.

- *Normalized joint trajectory length* ($D_2$) – the 31D feature vector constructed in the same way as the previous $D_1$ descriptor. The total trajectory lengths are additionally normalized by the length of the motion sequence, i.e., by the number of captured skeletons.
- *Maximum axis distance* ($D_3$) – the 93D feature vector whose dimensions correspond to the maximum reachable coordinate separately in the $x/y/z$ axis of each joint. When comparing with the feature of another motion, only the differences in the 8 most distant dimensions are considered, which implies that this similarity measure is not symmetric.

While the first two descriptors ($D_1$ and $D_2$) are computed based on the original motion data, the third descriptor ($D_3$) is extracted from normalized skeleton-centric view-invariant data, described in Sect. 2.

**Classification Accuracy.** Figure 2 illustrates the contrast between the accuracies of both $k$NN classifiers (introduced in Sect. 4) and the confusion-based classifier. The confusion-based classifier increases the accuracy with an increasing value of $k$ up to $k = 15$ and then the accuracy slightly decreases. When $k = 15$, the classification reaches the 89.09% accuracy. When compared to the baseline 1NN accuracy of 87.84%, the error in classification is decreased by 10%.

**Efficiency of Learning and Classification.** To pre-compute a confusion matrix $cm^{D_i}$ ($m \times m$) for a given descriptor $D_i$, there is a need to evaluate $m^2$ confusion values, where $m$ denotes the number of provided classes. Assuming that all the classes contain approximately the same number of $|\mathcal{M}^{\mathrm{TR}}|/m$ training motion samples, a single matrix cell requires $((|\mathcal{M}^{\mathrm{TR}}| - 1)/m) \cdot |\mathcal{M}^{\mathrm{TR}}|/m$ evaluations of the $D_i$ distance measure. In total, the following number of:

$$m^2 \cdot \left( \frac{|\mathcal{M}^{\mathrm{TR}}| - 1}{m} \cdot \frac{|\mathcal{M}^{\mathrm{TR}}|}{m} \right) \cong |\mathcal{M}^{\mathrm{TR}}|^2$$

*compDist*$^{D_i}$ distance computations is performed in the training phase. Since in our case $|\mathcal{M}^{\mathrm{TR}}| = 1{,}164$, we perform about 1.4 M distance computations for the original as well as each of the three additional descriptors. The total time needed to learn all the matrices takes only about 10 s. For example, the clearly most expensive *compDist*$^{4K}$ measure applied to the 4,096D features needs roughly 7 s on commodity hardware (i7 960 at 3.2 GHz).

The $md^{D_i}$ matrix keeping the maximum class-pairwise distances can be computed during the process of calculation of the confusion matrix.

Also the classification time is not much influenced by additional descriptors that need roughly 1 ms in total for re-ranking. The most expensive operation is the searching for $k$-nearest neighbors, which requires 6 ms. However, this time can be further decreased by indexing the original features by any metric-based structure [17] to scale to large databases of training samples. Sometimes, the feature extraction process is included in classification time, mainly when queries cannot be preprocessed in advance. In our case, the extraction of all the features, also including the original 4,096D feature, takes about 30 ms per a query motion.

## 6 Comparison with the State-of-the-Art Approaches

We compare the accuracy of the best-performing confusion-based classifier with the most recent approaches [4,6,7,13,14] that evaluate the same 2-fold cross validation procedure on the challenging HDM05 ground truth with 122 or 130 categories. Table 1 shows that we beat not only the 1NN classifiers based on the motion-image concept [13,14] but even the most recent purposely-trained classifiers [4,6,7] based on neural networks. From the recognition-error point of view, we decrease the error by 54%, 33%, 15%, 10%, and 19% with respect to the methods in [4,6,7,13,14], respectively.

**Table 1.** Comparison with the state-of-the-art methods using the 2-fold cross validation (i.e., using 50% of training data).

| Method | Accuracy (%) | |
|---|---|---|
| | HDM05-122 | HDM05-130 |
| Huang et al. [4] | N/A | 75.78 |
| Laraba et al. [6] | N/A | 83.33 |
| Sedmidubsky et al. [14] | 87.24 | N/A |
| Sedmidubsky et al. [13] | 87.84 | N/A |
| Li et al. [7] | N/A | 86.17 |
| Our approach | **89.09** | **88.78** |

Although there are other well-performing classifiers, such as [1,3,9,12,15], they do not evaluate the recognition accuracy on the challenging set of 122 or 130 HDM05 categories. One of the reasons is probably a limited amount of training data, with less than twenty samples per a single class. Moreover, such approaches pay a little attention to efficiency and applicability issues. Specifically, our confusion-based classifier has the following advantages when compared to the state-of-the-art classifiers.

– *Dynamic measure selection* – additional similarity measures are dynamically and automatically weighted to select the best possible one(s) for re-ranking, with respect to a query motion being classified. It is much more convenient than training a single descriptor that needs to recognize any class, as used in existing works.
– *Robustness* – if there are some misclassified training samples, they can degrade the classification accuracy. In our approach, the class of such samples can simply be repaired causing the immediate impact on classification, without the necessity of any long-term retraining like in [9,12].
– *Limited amount of training data* – the original descriptor does not need large amounts of training data due to the utilization of pre-trained neural network on a different image domain. Additional measures can also separate well classes having a limited number of training samples. In particular, we use 50% of the dataset for training in comparison with other approaches utilizing, e.g., 80% or 90% [14] of data for training.
– *Efficiency and indexability* – the comparison of the original 4,096D feature vectors by the Euclidean distance enables indexing such features by any metric-based structure [17] to efficiently evaluate $k$-nearest neighbor queries, even in very large databases of training samples.

## 7 Conclusions

The state-of-the-art similarity measure for motion data [13,14], based on 4,096D deep features extracted using a convolutional neural network, achieves a high accuracy even when used with the baseline 1NN classifier. In this paper, we employ this original measure to search for the $k$-nearest training samples with respect to an unlabeled query. The retrieved neighbors are analyzed by the proposed $k$NN_TCS classifier to determine the two top probable classes for the processed query. To select the correct class, we employ additional simple similarity measures. Based on class-pairwise confusion matrices automatically learned for each similarity measure, the correct class is selected in 94% of cases. This helps decrease the error in classification by 10% on the challenging HDM05-122 dataset, when compared to the baseline 1NN classifier.

The proposed approach, despite being very effective in classification, has several other advantages in comparison with existing classifiers. In particular, the training sample set can be (1) *small* (less than 10 samples per class) to reach a high classification accuracy or (2) *large* to search the $k$-nearest neighbors efficiently due to indexability of the original similarity measure, and (3) *dynamic* (even in sense of adding samples of new classes) with the immediate impact to classification, without any time-consuming retraining process. Moreover, the additional similarity measures are dynamically utilized in classification only when the original measure does not provide clear results. The provided additional measures are automatically integrated into the confusion-based classifier, without any need of supervision.

**Acknowledgment.** This research was supported by GBP103/12/G084.

## References

1. Baradel, F., Wolf, C., Mille, J.: Human action recognition: pose-based attention draws focus to hands. In: ICCV Workshop on Hands in Action, Venice, Italy (2017)
2. Barnachon, M., Bouakaz, S., Boufama, B., Guillou, E.: Ongoing human action recognition with motion capture. Pattern Recogn. **47**(1), 238–247 (2014)
3. Bütepage, J., Black, M.J., Kragic, D., Kjellström, H.: Deep representation learning for human motion prediction and classification. CoRR abs/1702.07486 (2017)
4. Huang, Z., Wan, C., Probst, T., Gool, L.V.: Deep learning on lie groups for skeleton-based action recognition. CoRR abs/1612.05877 (2016)
5. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 25, pp. 1097–1105. Curran Associates, Inc. (2012)
6. Laraba, S., Brahimi, M., Tilmanne, J., Dutoit, T.: 3D skeleton-based action recognition by representing motion capture sequences as 2D-RGB images. Comput. Anim. Virtual Worlds **28**(3–4), e1782 (2017)
7. Li, C., Cui, Z., Zheng, W., Xu, C., Yang, J.: Spatio-temporal graph convolution for skeleton based action recognition. In: 32nd Conference on Artificial Intelligence (AAAI 2018). AAAI Press (2018)
8. Liu, J., Shahroudy, A., Xu, D., Wang, G.: Spatio-temporal LSTM with trust gates for 3D human action recognition. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9907, pp. 816–833. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46487-9_50
9. Liu, J., Wang, G., Duan, L., Hu, P., Kot, A.C.: Skeleton based human action recognition with global context-aware attention LSTM networks. IEEE Trans. Image Process. **27**(4), 1586–1599 (2018)
10. Lo Presti, L., La Cascia, M.: 3D skeleton-based human action classification. Pattern Recognit. **53**(C), 130–147 (2016)
11. Müller, M., Röder, T., Clausen, M., Eberhardt, B., Krüger, B., Weber, A.: Documentation Mocap Database HDM05. Technical report CG-2007-2, Universität Bonn (2007)
12. Nez, J.C., Cabido, R., Pantrigo, J.J., Montemayor, A.S., Vlez, J.F.: Convolutional neural networks and long short-term memory for skeleton-based human activity and hand gesture recognition. Pattern Recognit. **76**, 80–94 (2018)
13. Sedmidubsky, J., Elias, P., Zezula, P.: Enhancing effectiveness of descriptors for searching and recognition in motion capture data. In: 19th International Symposium on Multimedia, pp. 240–243. IEEE Computer Society (2017)
14. Sedmidubsky, J., Elias, P., Zezula, P.: Effective and efficient similarity searching in motion capture data. Multimed. Tools Appl. **77**(10), 12073–12094 (2018). https://doi.org/10.1007/s11042-017-4859-7
15. Song, S., Lan, C., Xing, J., Zeng, W., Liu, J.: An End-to-End spatio-temporal attention model for human action recognition from skeleton data. CoRR abs/1611.06067 (2016). http://arxiv.org/abs/1611.06067

16. Wang, P., Li, W., Ogunbona, P.O., Wan, J., Escalera, S.: RGB-D-based motion recognition with deep learning: a survey. Int. J. Comput. Vis. **PP**(99), 1–34 (2017)
17. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems, vol. 32. Springer, Boston (2006). https://doi.org/10.1007/0-387-29151-2
18. Zhu, W., et al.: Co-Occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks. In: 30th Conference on Artificial Intelligence (AAAI 2016), pp. 3697–3703. AAAI Press (2016)