Pattern Recognition Letters 36 (2014) 213–220

Contents lists available at SciVerse ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Efficient descriptor tree growing for fast action recognition

S. Ubalde *, N.A. Goussies, M.E. Mejail

Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina

ARTICLE INFO

Article history:
Available online 22 May 2013

Communicated by Luis Gomez Deniz

Keywords:
Action recognition
Nearest neighbor
Instance-to-Class distance

ABSTRACT

Video and image classification based on *Instance-to-Class* (I2C) distance attracted many recent studies, due to the good generalization capabilities it provides for non-parametric classifiers. In this work we propose a method for action recognition. Our approach needs no intensive learning stage, and its classification performance is comparable to the state-of-the-art. A smart organization of training data allows the classifier to achieve reasonable computation times when working with large training databases. An efficient method for organizing training data in such a way is proposed. We perform thorough experiments on two popular action recognition datasets: the KTH dataset and the IXMAS dataset, and we study the influence of one of the key parameters of the method on classification performance.

1. Introduction

Action recognition is a growing topic in computer vision research. Given a list of possible actions (e.g. running, sitting, clapping hands, etc.) and a video showing an actor performing any one of them, the goal is to produce an algorithm for the recognition of the action being performed. Automatic identification of actions in videos is a key aspect in many applications such as video summarization, video indexing, vigilance based on the analysis of security cam-captured videos, interaction with computers via movement, etc. The problem poses several challenges. On the one hand, the appearance of an action can vary considerably in different videos. This may be due to changes in lighting conditions, viewpoint, actor's clothing, etc. On the other hand, different actions can look very similar to each other. Occlusion and bad lighting conditions can add further difficulty to the problem.

Among recent works, those inspired on the *bag-of-features* approach became popular because of their simplicity and good performance. These studies base classification on measuring the distance between the query instance and each of the training instances – *Instance-to-Instance* (I2I) distance- and require quantizing instance features into a fixed-length vector for representation. Recently, Boiman et al. (2008) suggested that the use of I2I distance and feature quantization can severely hurt performance and proposed a method to overcome these issues. Their approach deals directly with unquantized features and computes *Instance-to-Class* (I2C) distances (instead of Instance-to-Instance) for classification. Apart from dealing with these problems, the method (referred to as *Naive–Bayes Nearest-Neighbor* or *NBNN*) presents several attractive features. First, it is a *non-parametric* classifier, which means that it needs no intensive learning phase. This is extremely useful when working with large training databases that are subject to frequent updates. Second, it achieves a performance comparable to that of the top *learning-based* methods. Learning based methods require an intensive parameter learning phase, and can usually achieve a better classification performance than non-parametric methods. As an extra advantage, the idea behind the method is fairly simple.

Despite its good qualities, the NBNN method is not well-suited for most real-world problems (Wang et al., 2009). The number of training features required at those scenarios to achieve a state-of-the-art performance is usually very large. This makes I2C computation expensive and results in prohibitive classification times. In Ubalde and Goussies (2012) we proposed an alternative method to NBNN (named *NBNNTree*), by which we aimed at lowering the amount of time consumed for classification. Broadly, the strategy followed to achieve such improvement was that of organizing the training features in a particular fashion.

Our approach successfully reduced the time complexity of NBNN while achieving a similar classification accuracy. However, the training stage suggested in Ubalde and Goussies (2012) had a few loose ends, that limited its use to databases with a low number of actions. In Section 4, after a short introduction to our original method, we present a strategy to overcome these problems.

As far as we are aware, only (Wang et al., 2009; Yuan et al., 2011) have used NBNN for action recognition. In Section 5 we thoroughly test our improved version of the NBNNTree method on two very popular action recognition datasets: the KTH dataset and IXMAS multiview dataset. We compare its performance and computation times with those achieved by NBNN, and we investigate the influence of one of its main parameters on classification performance.

* Corresponding author. Tel.: +54 11 4541 1113; fax: +54 11 4576 3359.
E-mail addresses: seubalde@dc.uba.ar, sebastian.ubalde@gmail.com (S. Ubalde), ngoussies@dc.uba.ar (N.A. Goussies), marta@dc.uba.ar (M.E. Mejail).

0167-8655/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.patrec.2013.05.007

2. Previous work

Automatic analysis of actions and behaviors in video has been extensively analyzed in previous works. Existing approaches to address the problem are varied.

An entire body of work is based on a global representation of the video. Using tracking or background subtraction the actor is localized in the video, and movement information is encoded as a whole. The approach presented in Davis and Bobick (1997) is based on the so-called *temporal templates*. They extract silhouettes from several frames, and aggregate differences among them yielding two images that encode action information: motion history image (MHI) and motion energy image (MEI). Hu moments are used to compare two templates. Euclidean distance is used by Weinland et al. (2007) to match two silhouettes. More examples based on silhouettes can be found in Zhu et al. (2009), Souvenir and Babbs (2008) and Wang and Suter (2006). Several works use spatio-temporal volumes to represent an action. A spatio-temporal volume is formed by stacking frames over a given sequence. Examples of this approach can be found in Yilmaz and Shah (2008), Yan et al. (2008), Grundmann et al. (2008) and Zelnik-manor and Irani (2001).

More related to our work are those approaches based on local descriptors. Such approaches are derived from techniques used in image classification. The basic idea is to characterize a video using descriptors of spatio-temporal patches extracted from certain interest points. In the work of Laptev (2005), the Harris detector (Harris and Stephens, 1988) is extended to the space–time domain. Interest points are located using the extended detector. The resulting points can be thought of as spatiotemporal corners. Patches around them are expected to correspond to video objects whose movement is changing direction. Dollar et al. (2005) detect interest points using a Gaussian filter to the spatial dimension and a Gabor filter to the temporal dimension. Chomat et al. (2000) use the responses after applying spatio-temporal receptive fields. Rapantzikos et al. (2007) apply discrete wavelet transforms in each of the three directions of a video volume.

Patches around interest points are usually represented using descriptors. Descriptors are intended to provide distinctive information about the patch, while being invariant to appearance, occlusion, rotation and scale. In Laptev (2005) histograms of oriented flow and gradients are used as descriptors. Dollar et al. (2005) use image gradients and PCA to reduce descriptor dimensionality (Willems et al., 2008) use an extension of SURF features (Bay et al., 2006) to 3D.

Many works use descriptor quantization in order to work with low-dimensional data. In Dollar et al. (2005), Laptev et al. (2008), Sivic and Zisserman (2003), Niebles et al. (2006), Schuldt et al. (2004) and Liu and Shah (2008), descriptors are clustered and cluster centers are selected as codewords. Videos are therefore represented as histograms of codewords. This approach is commonly known as *bag-of-features*. A classifier is trained using the set of histograms from the training videos. Nearest neighbor (NN) and support vector machines (SVM) are among the most used classifiers. While the first ones are easier to train, the last ones often achieve a better performance.

In Yuan et al. (2009) and Wang et al. (2009), descriptors are not clustered. Instead, they are used directly for classifying the video. This is based on the idea of Boiman et al. (2008) and presents several advantages over the bag-of-features approach, as discussed later in this paper.

3. The NBNN method

While in the work of Boiman et al. NBNN is used for image classification, this paper deals with action recognition. Because of that, we use a slightly different terminology here, to reflect the fact that we are working with *videos* and *actions* instead of *images* and *classes*.

Let V be a query video, and let $d_1, d_2, \ldots, d_n$ be its local descriptors. The NBNN method chooses the action $\widehat{A}$ performed in V according to the following equation:

$$\widehat{A} = \underset{A}{\operatorname{argmin}} \sum_{i=1}^{n} \|d_i - NN_A(d_i)\|^2, \tag{1}$$

where $NN_A(d_i)$ is the nearest neighbor of d_i within the descriptors of action A. Descriptors of action A are gathered from every training video labeled with A. As Boiman et al. show the summation in (1) approximates a *Video-to-Action* (V2A) KL-distance (Boiman et al., 2008). In other words, NBNN computes an approximated distance from V to every possible action, and chooses the action with the minimum distance.

3.1. NBNN drawbacks

As shown in Wang et al. (2009), NBNN requires a large number of local descriptors in the training set to achieve state of the art performance. This makes the computation of $NN_A(d_i)$ in (1) very expensive for real-world sets (which are usually built extracting more than 10,000 descriptors per training instance). This is the main computational bottleneck, even when approximate searches (using KD-trees as in Boiman et al. (2008)) are performed.

Based on the previous observation, it seems reasonable to expect that a reduction in the number of NN searches would lead to a more time efficient method. A first step in this direction is to notice the sequential nature of the NBNN method. Only after computing the V2A distance for every action, the method chooses the closest action. This may seem like a fair strategy, but it does not take advantage of a very common phenomena in action recognition problems. In most of them, actions can be easily arranged in sets of look-alike actions, each set containing actions similar to each other but not similar to actions in other sets.

For example, in the KTH dataset (Laptev, 2005) two sets are distinguishable at first glance: the one consisting of actions *boxing*, *hand waving* and *hand clapping* and the one consisting of actions *running*, *jogging* and *walking*. It would take a very bad classifier to classify a *running* video as belonging to any action in the first set, or a *boxing* video as belonging to any action in the second set. Taking this into account, it seems inefficient to compute the V2A distance for every action. It would be much more efficient to quickly discard the wrong set, concentrating the efforts in choosing an action within the right set. This is precisely the idea behind our proposed method.

4. The NBNNTree method

Our method is based on a particular organization of the descriptors in the training dataset. Instead of grouping descriptors according to their action (as in NBNN), we group them according to their *action-set*. An action-set is just a set of actions (e.g. the set {*boxing*, *hand waving*, *hand clapping*}).

The method requires a training step in which all actions are organized in an *action-set tree*. An action-set tree is a binary tree in which every subtree is labeled at its root with an action-set. For the purposes of our method, we are interested only in those action-set trees which are *valid*. A valid action-set tree t can be described as follows. If t is a leaf, then it should be labeled with an action-set consisting of a single action. If t is not a leaf, then the following conditions should be met:

1. Let s be the action-set label of t. Let s_l and s_r be the action-set labels of t's left and right subtree respectively. Then, $\{s_l, s_r\}$ should be a partition of s, with $|s_l| = \lfloor \frac{|s|}{2} \rfloor$ and $|s_r| = \lceil \frac{|s|}{2} \rceil$ where $|.|$ denotes cardinality, and $\lfloor . \rfloor$ and $\lceil . \rceil$ are the floor and ceiling functions respectively.
2. The left subtree of s should be a valid action-set tree.
3. The right subtree of s should be a valid action-set tree.

Note that a valid action-set tree will always be balanced. Fig. 1 shows two examples of action-set trees. Every action-set in the picture is tagged with a letter for future reference. Fig. 1(b) shows an invalid tree. The tree is invalid for several reasons. First, $\{B, C\}$ is not a partition of A because they have an action in common (*boxing*). Also, $|C| \neq \lceil |A|/2 \rceil$. Second, C has an action (*boxing*) that cannot be found neither in D or E. Therefore, $\{D, E\}$ is not a partition of C. And third, B is at a leaf, but it contains more than one action.

Based on the action-set tree, a *descriptor tree* is built. A descriptor tree is just an action-set tree in which every subtree stores at its root descriptors of the actions in its associated action-set. Descriptors are selected randomly from the training descriptor dataset. For example, for the tree of Fig. 1(a), the node labeled with B stores at its root descriptors selected randomly from the training dataset of actions *boxing* and *running*. Likewise, the node labeled with C stores at its root descriptors selected randomly from the training dataset of *boxing*. The exact number q of descriptors to be selected is chosen based on the confusion matrix obtained from our NBNN tests. Let s, s_l and s_r be defined as before, let z be the number of descriptors per action in the training dataset, and m the confusion matrix for NBNN. The degree of confusion between actions in s_l and actions in s_r is given by $d(s_l, s_r, m) = \sum_{(a,b)\in(s_l \times s_r)} m(a,b) + m(b,a)$. To compute q, we first normalize $d(s_l, s_r, m)$, dividing it by $2|s_l||s_r|\alpha$, where α is just a reasonable confusion value between actions (we used 1% for all our experiments). Let y be the normalized value obtained as a result of this division. We then set $q = yz$. Whenever q is larger than z (i.e. whenever $y > 1$), we simply discard it and use z instead.

Algorithm 1. NBNNTree algorithm

NBNNTree$(d_1, \ldots, d_n, t)$
Returns an action for a given set of descriptors $d_1, \ldots, d_n$ and a descriptor tree t Steps:
1. If t is a leaf, return the only action in $action - set(t)$.
2. Let $L = descriptors(left(t))$ and $R = descriptors(right(t))$.
3. $\forall d_i$ compute the nearest neighbor of d_i in $L : NN_L(d_i)$.
4. Compute $D_L = \sum_{i=1}^{n} \|d_i - NN_L(d_i)\|^2$.
5. $\forall d_i$ compute the nearest neighbor of d_i in $R : NN_R(d_i)$.
6. Compute $D_R = \sum_{i=1}^{n} \|d_i - NN_R(d_i)\|^2$.
7. If $D_L < D_R, t_{next} = left(t)$, else $t_{next} = right(t)$.
8. Recursively call *NBNNTree*$(d_1, \ldots, d_n, t_{next})$.

Given a query video, both its local descriptors and the descriptor tree are used by the NBNNTree method for classification. The algorithm for the NBNNTree classifier is detailed in Algorithm 1. For a given descriptor tree t, we use $left(t)$ and $right(t)$ to designate the left and right subtrees of t respectively, $action - set(t)$ to designate the label at the root of t and $descriptors(t)$ to designate the set of descriptors stored at the root of t.

The algorithm starts from the root of the tree and descends one level at a time. At each level, the distance D_L between the video and the action-set in the left subtree is compared to the distance D_R between the video and the action-set in the right subtree. The algorithm descends to the subtree with smaller distance and the process is repeated. When a leaf is reached, its action-set (consisting of a single action) is returned.

4.1. Building the action-set tree

Our method avoids the sequential strategy of the NBNN method. This was motivated by the observation that actions can be arranged into increasingly smaller sets of look-alike actions, yielding an action-set tree. But the question remains of how to build that tree. In Ubalde and Goussies (2012) we used a method based on the confusion matrix from our NBNN tests. At each step, the algorithm evaluated every possible partition of a set of actions s into a pair of sets s_l and s_r. The sets s_l and s_r were chosen in such a way that actions belonging to the same set were often confused with each other by the NBNN classifier.

The brute force nature of the method presented in Ubalde and Goussies (2012) make it a reasonable option for databases with a low number of actions. When working with a larger number of actions, however, a better strategy is required. As another disadvantage, the method relies heavily on the results of another method (NBNN). In order to avoid these problems, we present an heuristic to efficiently estimate the similarity of two given actions based solely on the descriptors computed for each of them.

Given two actions A and B, we define the dissimilarity $h(A, B)$ between them as:

$$h(A,B) = \sum_{d \in A^+} \|d - NN_B(d)\|^2 + \sum_{d \in B^+} \|d - NN_A(d)\|^2, \tag{2}$$

where A^+ is the set of all descriptors extracted from videos of action A, B^+ is the set of all descriptors extracted from videos of action B, $NN_A(d)$ is the nearest neighbor of d within the descriptors of action A and $NN_B(d)$ is the nearest neighbor of d within the descriptors of action B.

While this approach works well for databases with a low number of descriptors per action, it is computationally too expensive for real world problems (where the amount of descriptors required per action is usually very large).

In order to speed this up, we take into account only a small subset of A^+ and B^+ to compute $h(A, B)$. The criteria used to select this subset is not trivial. We found that randomly choosing descriptors does not work well enough. This is probably due to the large number of non-informative descriptors present in typical databases. As shown in Boiman et al. (2008), the most frequent descriptors are the ones that provide low class discriminativity.

We propose using a smarter criteria to build the subset. Following Yuan et al. (2009), we choose descriptors based on its *purity*. Denote by A^- the set of all descriptors extracted from videos of an action different from A. For a given descriptor d, denote its ϵ-nearest neighbors in A as $NN_\epsilon^A(d) = \{d_j \in A^+ : \|d - d_j\| \leqslant \epsilon\}$. The set of all d ϵ-nearest neighbors is denoted as: $NN_\epsilon(d) = \{d_j \in A^+ \cup A^- : \|d - d_j\| \leqslant \epsilon\}$. For a given $d \in A^+$, the ϵ-purity of d is defined by $w_\epsilon(d) = \frac{|NN_\epsilon^A(d)|}{|NN_\epsilon(d)|}$, where $|.|$ denotes cardinality. Note that, since $NN_\epsilon^A(d) \subseteq NN_\epsilon(d)$, we have $0 < w_\epsilon(d) \leqslant 1$. Basically, $w_\epsilon(d)$ describes the purity of d as a descriptor of its class. The larger $w_\epsilon(d)$, the purer d is. When computing the dissimilarity between two actions A and B, we consider only the p purest descriptors of A^+ and B^+, with $p \ll |A^+| = |B^+|$. In other words, we define the approximate dissimilarity between two actions A and B as:

$$h_p(A,B) = \sum_{d \in A_p^+} \|d - NN_B(d)\|^2 + \sum_{d \in B_p^+} \|d - NN_A(d)\|^2, \tag{3}$$

where A_p^+ and B_p^+ are the sets containing the p purest descriptors of A^+ and B^+ respectively.

Algorithm 2 shows how we obtain a partition of a given $action - set\ s$ into a pair of $action - sets\ s_l$ and s_r. The method greedily chooses an action and moves it from s_r to s_l. The action is chosen

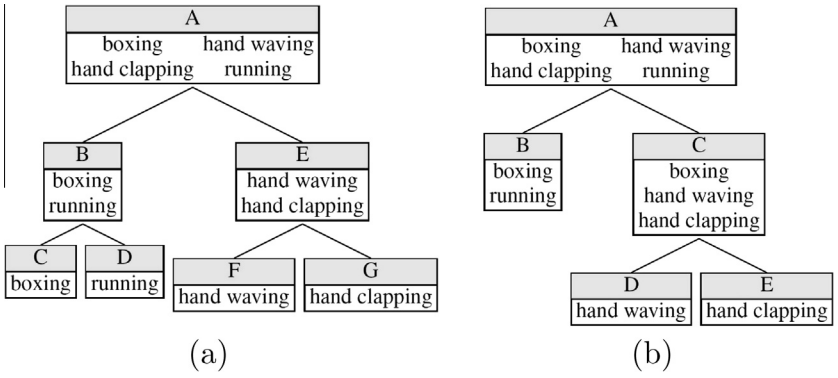

Fig. 1. Examples of action-set trees. (a) A valid tree. (b) An invalid tree.

based only on its similarity with the actions in s_l. This procedure is repeated until s_l and s_r have the same number of elements.

Algorithm 2. Splitting an action-set

SplitAction − Set(s)
Split s into two *action-sets* s_l and s_r.
Steps:
1. Randomly choose an action A in s and set $s_l = \{A\}$ and $s_r = s - \{A\}$.
2. Repeat while $|s_l| < |s_r|$
2.1 Find the action B in s_r such that $\sum_{A \in s_l} h_p(A, B)$ is minimal.
2.2 Let $s_l = s_l \cup \{B\}$ and $s_r = s_r - \{B\}$.

Finally, the steps to build the action-set tree are shown in Algorithm 3. The method partitionates the given action-set s into a pair of action-sets s_l and s_r, using Algorithm 2, and recursively builds trees from s_l and s_r.

Algorithm 3. Building the action-set tree

BuildTree(s)
Builds an action-set tree t from a given set of actions s.
Steps:
1. $action - set(t) = s$.
2. If $|s| = 1$ return.
3. Split s into two subsets s_l and s_r, using the method explained in Algorithm 2.
4. $left(t) = BuildTree(s_l, m), right(t) = BuildTree(s_r, m)$.

4.2. Time complexity of the NBNNTree method

As shown in the previous section, NBNNTree performs several steps at each level of the tree (numbered from 1 to 8 in Algorithm 1). Of these steps, 1, 2, 7 and 8 are clearly $\mathcal{O}(1)$. Step 3 computes $NN_L(d_i)$ for each of the n descriptors of the query video. Following Boiman et al. (2008), we use an approximate nearest neighbor algorithm (Muja and Lowe, 2009) for the computation of $NN_L(d_i)$. The expected time for this nearest neighbor search is logarithmic in $|L|$. Thus, step 3 is $\mathcal{O}(n \log(|L|))$. Step 4 is clearly $\mathcal{O}(n)$, because it involves only $\mathcal{O}(1)$ computations over n values. Step 3 to 4 together are therefore $\mathcal{O}(n \log(|L|))$. Likewise, steps 5 to 6 are $\mathcal{O}(n \log(|R|))$. In our experiments, every node in the descriptor tree stores at most z descriptors, where z is the number of training descriptors for a single action. Replacing $|L|$ and $|R|$ for z in the previous expressions yields a time complexity of $\mathcal{O}(n \log(z))$ for the steps 1–8 performed at each level of the tree. Since t is built from a valid action-set tree, its height is logarithmic in the total number of actions k. Thus, the time complexity of the NBNNTree classifier is $\mathcal{O}(\log(k) n \log(z))$. This is a substantial speed-up over the NBNN method, which has a time complexity of $\mathcal{O}(kn \log(z))$.

5. Experiments

In this section we evaluate the NNBNN and NBNNTree methods on two well-known action recognition datasets: the KTH dataset (Laptev, 2005) and the IXMAS dataset (Zelnik Manor et al., 2006).

Following the procedure in Dollar et al. (2005), we carry out a training stage to build a descriptor dataset per action. Specifically, given an action A we perform the following steps for every training video V labeled with A:

1. Detect n interest points in V.
2. Compute n descriptors $d_1, d_2, \ldots, d_n$ for the detected interest points.
3. Store $d_1, d_2, \ldots, d_n$ in the dataset for action A.

Steps 1 and 2 are implemented as suggested in Dollar et al. (2005). We detect interest points using a Gaussian filter over the spatial dimension and a Gabor filter over the temporal dimension. Patches around interest points are represented with image gradients. PCA is used to reduce descriptor dimensionality.

When classifying a video V, we first extract n descriptors from V as in the training stage. Then, based on the extracted descriptors and the descriptor datasets from the training stage, an action is chosen using NBNN or NBNNTree. Parameters for the approximate nearest neighbor search were set as proposed in Dollar et al. (2005).

The action-set and descriptor tree used by NBNNTree were built as shown in Section 4.1. A key aspect of the method followed to build the descriptor tree is the criteria chosen to select the number of descriptors per node. In Section 5.1 we explain the experiments performed to validate this criteria.

5.1. Choosing the right number of descriptors per node

As explained in Section 4, every subtree in a descriptor tree stores at its root descriptors of the actions in its associated action-set. The exact number q of descriptors to select from the training dataset is chosen based on the confusion matrix from the NBNN method. Given t_l and t_r the left and right subtrees of any given subtree, the number of descriptors to store at their roots is determined by the degree of confusion between the action-set associated with t_l and the action-set associated with t_r. The bigger the degree of confusion, the larger the number of descriptors. This was inspired by the intuition that action-sets that are harder to distinguish require more descriptors for correctly classifying a new video.

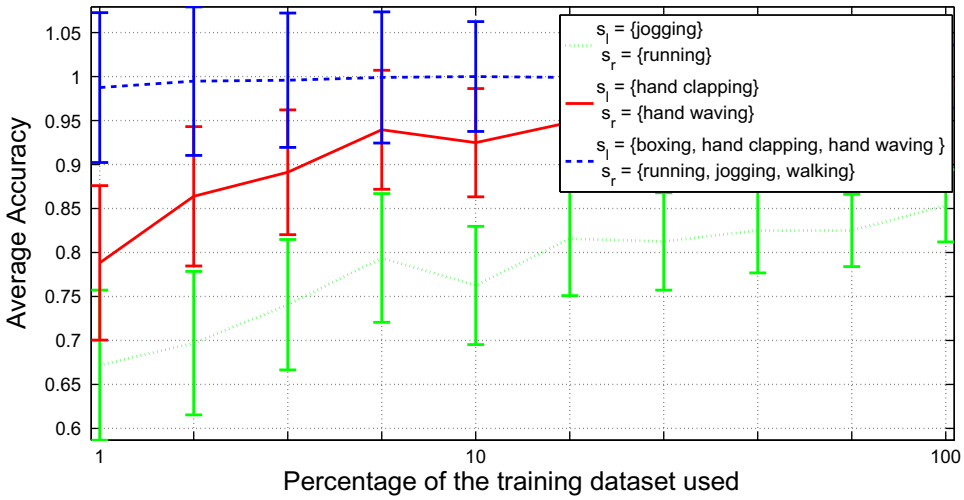

Fig. 2. Average accuracy for three representative pairs of action-sets, as a function of the percentage of descriptors employed for classification.

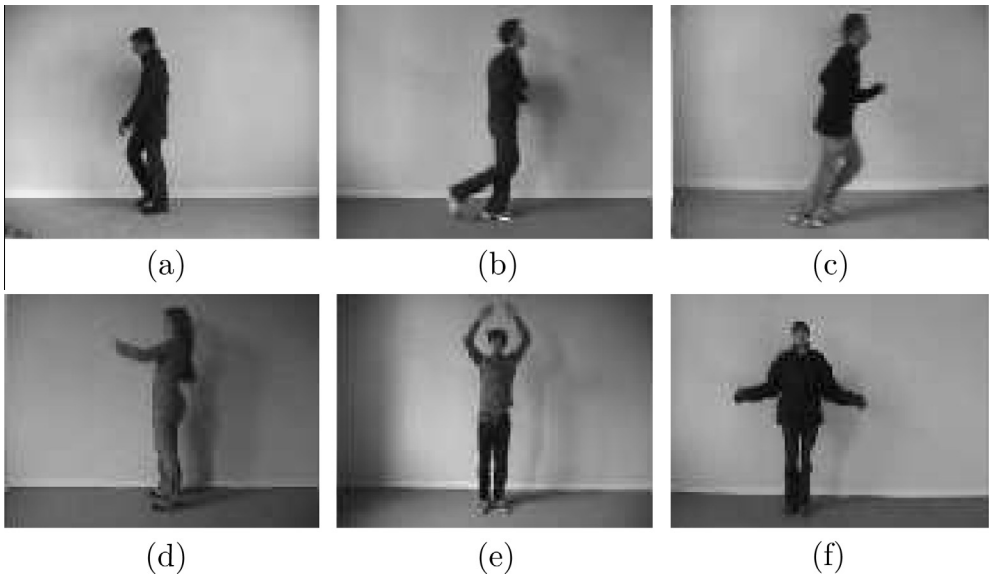

Fig. 3. Example frames from the KTH dataset: *walking* (3a), *jogging* (3b), *running* (3c), *boxing* (3d), *hand waving* (3e) and *hand clapping* (3f).

In order to verify this intuition, we carry out a series of experiments that study the influence of the number of descriptors on the classification accuracy. For a given node, we gradually change the number of descriptors stored at its subtree roots, computing the average accuracy for that node over all the training videos. Beginning at 1% of the total number of descriptors, we compute the average accuracy for different percentages of the descriptor dataset. Results for three representative nodes are shown in Fig. 2.

As expected, accuracy variation is higher for those action-sets that are often confused with each other by NBNN. For example, action-sets {*jogging*} and {*running*} are among the most confusing ones. For those action-sets, accuracy ranges from 67% when using 1% of the original descriptors to 85% when using the whole descriptor dataset. In contrast, accuracy variation is low for action-sets that are easily distinguishable from each other. For example, videos associated with {*boxing*, *hand clapping*, *hand waving*} are usually very different than videos associated with {*running*, *jogging*, *walking*}. This is reflected in a variation of less than 2% in accuracy (from 98.75% when using 1% of the descriptors, to 99.5%, when using 100% of the dataset).

These results support our idea of using a low number of descriptors for those action-sets that NBNN finds hard to distinguish from each other. Since using a high percentage of the dataset will not imply a significant improvement on classification accuracy, it seems reasonable to use just a small part of it and achieve better computation times.

5.2. KTH dataset

The KTH dataset contains 6 actions, performed several times by 25 actors in four different scenarios of appearance, illumination and scale. Both camera location and orientation of actors remain constant for most videos. In total, the dataset consists of 2391 videos. Fig. 3 shows some example frames.

We used leave-one-out cross-validation (LOOCV) on the actors. That is, videos were divided into 25 sets, each including exactly the videos of one actor. In each of 25 experiments, the classifier was trained using the videos from 24 sets and tested on the videos from the remaining set. We report average precision (both global and by action) over the 25 experiments.

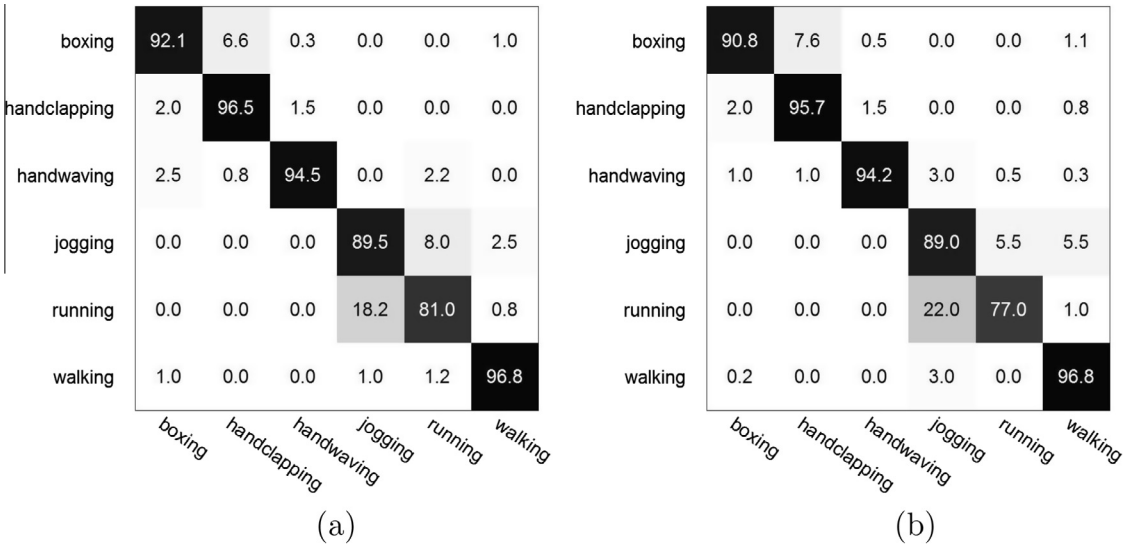

Fig. 4. Confusion matrices for the KTH dataset using NBNN (a) and NBNNTree (b).

Table 1
Average precision (in %) of different methods on the KTH dataset.

Method	Precision
Dollar et al. (2005)	80.66
Liu and Shah (2008)	94.16
Laptev et al. (2008)	91.8
Nowozin et al. (2007)	84.72
NBNN	91.73
NBNNTree	90.58

Fig. 4(a) shows the confusion matrix for the NBNN method. Average precision is 91.73%. Fig. 4(b) shows the confusion matrix for the NBNNTree method. Our method obtains similar results to those of NBNN for all the actions, and achieves 90.58% average precision.

Table 1 compares our results on the KTH dataset with those of existing approaches. Despite its simple training stage, NBNN achieves a state-of-the-art performance. NBNNTree shows similar results, outperforming most of the previous approaches.

The average running time for classification is reduced by 10% when using NBNNTree, compared to NBNN. This improvement came almost entirely from the strategy used to select the number of stored descriptors at each node, detailed in Section 4. For some of the nodes, only 8% of the total number of descriptors for that node action-set were enough to achieve the results reported in Fig. 4(b), which is impressive. Because of the small number of actions of the KTH dataset, the main improvement in time complexity provided by NBNNTree is not fully exploited.

5.3. IXMAS

The IXMAS dataset contains 13 actions, performed 3 times by 12 actors. Each action execution was recorded by fixed cameras at 5 different positions. We use the videos of the 4 cameras usually considered in literature. Actors arbitrarily chose position and

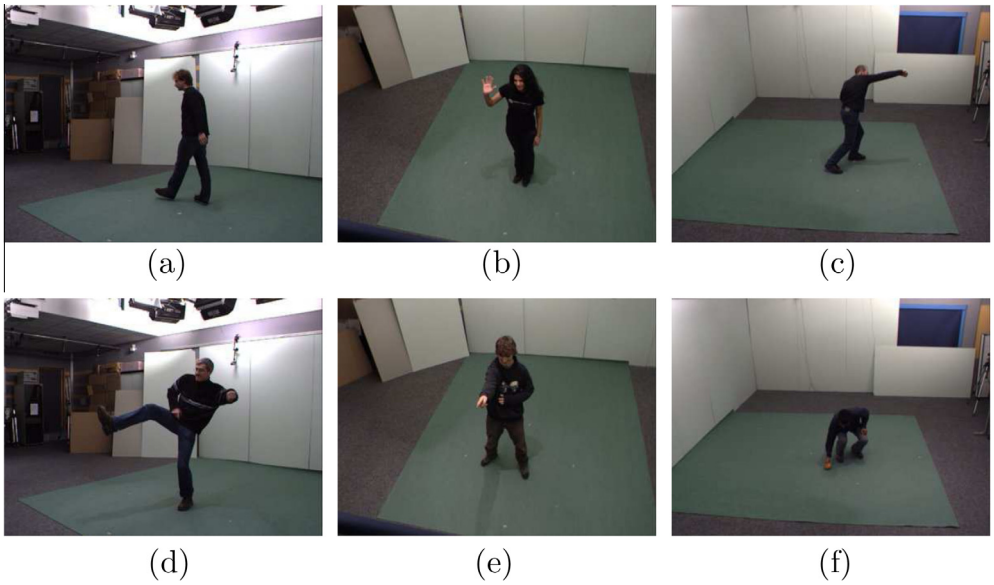

Fig. 5. Example frames for the dataset IXMAS: *walk* (a), *wave* (b), *punch* (c), *kick* (d), *point* (e) and *pick up* (f).

Table 2
Average precision (in %) of different methods on the IXMAS dataset.

Method	Cam 0	Cam 1	Cam 2	Cam 3
Weinland et al. (2007)	65.4	70	54.3	66
Yan et al. (2008)	72	53	68	63
Liu and Shah (2008)	76.67	73.29	71.97	72.99
NBNN	79.4	76.15	74.04	74.13
NBNNTree	78.8	75.83	71.07	71.75

orientation. The high number of actions and the great variety both in appearance and orientation make IXMAS one of the most challenging datasets. Fig. 5 shows some example frames.

We used 6-fold cross-validation on the actors. Namely, videos were divided into six sets, each including exactly the videos of two actors. In each of six experiments, the classifier was trained using the videos from five sets and tested on the videos from the remaining set. We report average precision (both global and by action) over the six experiments.

We used two testing schemes suggested in Liu and Shah (2008): (1) learning from four cameras and (2) learning from three cameras. In both cases we used only one camera for recognition.

In the first scheme, all the videos taken by the four cameras were used to train the classifier. The trained classifier was then tested on the videos of one designated camera. Table 2 shows the average precision values for NBNN and NBNNTree, and

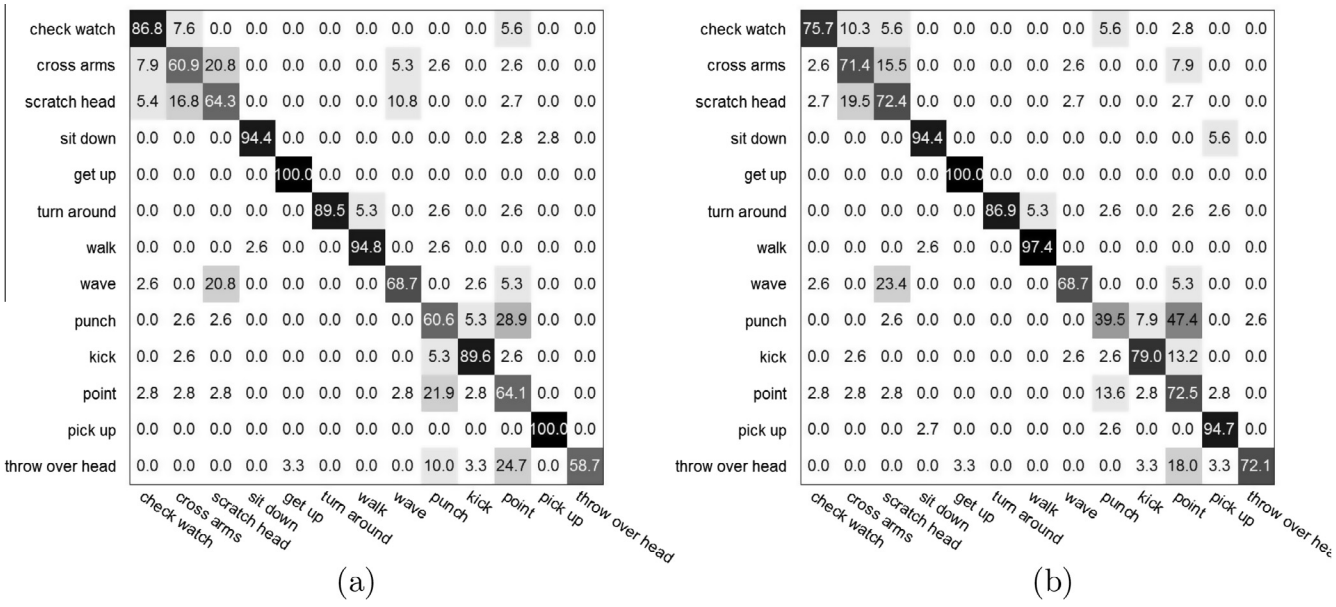

Fig. 6. Confusion matrices for the IXMAS dataset, using NBNN (a) and NBNNTree (b).

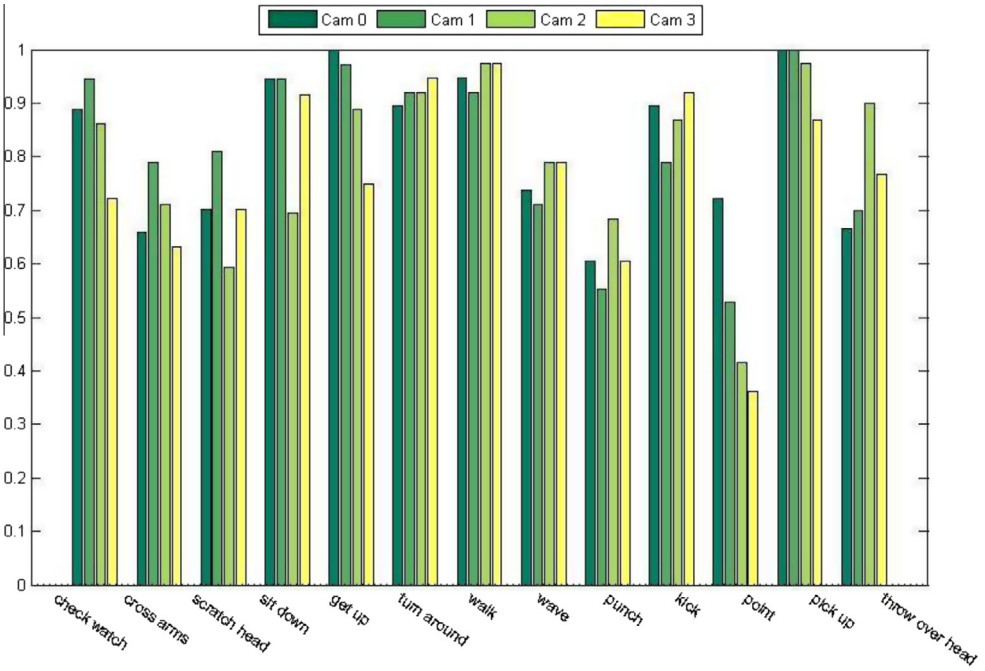

Fig. 7. Average precision by action for the NBNN method, training with 4 cameras on the IXMAS database.

Table 3
Comparative results on the IXMAS dataset. Average precision (in %) is shown according to the camera used for recognition. Videos from three cameras were used for training.

Method	Camera 0	Camera 1	Camera 2	Camera 3
Liu and Shah (2008)	72.29	61.22	64.27	70.99
NBNN	74.2	72.58	56.12	64.31
NBNNTree	74	68.5	52.21	66.13

compares them with the results achieved by other approaches. Both NBNN and NBNNTree perform better than the rest of the methods.

The best recognition performance is obtained with cameras 0 and 1. This is probably due to the higher level of self occlusion presented by cameras 2 and 3. This makes distinction between actions involving subtle arm movements more difficult (for example, actions *scratch head* and *wave* are often confused with each other). Motions usually appear more salient in cameras 0 and 1.

Fig. 6(a) shows the confusion matrix for the NBNN method. Average precision is 82.18%. Fig. 6(b) shows the confusion matrix for the NBNNTree method. Our method obtains similar results to those of NBNN for most actions, and achieves 81.34% average precision. Fig. 7 shows the average precision for each action using NBNN.

In the second scheme, we train the classifier using the videos from three cameras, and use the videos from the remaining camera to test it. This scheme is more challenging than the previous one, because there are no videos registered with the testing camera among the training set. Table 3 shows the average precision values for NBNN, NBNNTree and the approach in Liu and Shah (2008). The latter outperforms both NBNN and NBNNTree for cameras 2 and 3, but achieves lower precision for cameras 0 and 1. Our method shows similar performance to NBNN for all the cameras.

6. Conclusions

The usage of Instance-to-Class distance and unquantized descriptors greatly improves performance of non-parametric methods. This make it possible to achieve state-of-the-art results using simple classifiers like those based on nearest-neighbor. However, existing methods require a huge number of training features to obtain reasonable accuracy. This leads to high computation times, making them not-suitable for real world problems.

We focused on a particular method (NBNN) and proposed an alternative approach (NBNNTree) that significantly reduces its time complexity while achieving similar classification accuracy. Our approach benefits from the good qualities of non-parametric methods, and handles big training datasets in reasonable times. On the challenging IXMAS dataset, the average running time is reduced by 50% when using our method, compared to NBNN. The training dataset has to be structured in a specific way in order to be used by NBNNTree. We presented an efficient method to organize descriptors and explored the influence of an important parameter of that method on classification accuracy.

References

Bay, H., Tuytelaars, T., Van Gool, L., 2006. Surf: speeded up robust features. In: ECCV06, pp. I: 404–417.

Boiman, O., Shechtman, E., Irani, M., 2008. In defense of nearest-neighbor based image classification. In: CVPR08, pp. 1–8.

Chomat, O., Martin, J., Crowley, J., 2000. A probabilistic sensor for the perception and the recognition of activities. In: ECCV00, pp. I: 487–503.

Davis, J., Bobick, A., 1997. The representation and recognition of action using temporal templates. In: CVPR97, pp. 928–934.

Dollar, P., Rabaud, V., Cottrell, G., Belongie, S., 2005. Behavior recognition via sparse spatio-temporal features. In: PETS05, pp. 65–72.

Grundmann, M., Meier, F., Essa, I., 2008. 3d Shape context and distance transform for action recognition. In: ICPR0, pp. 1–4.

Harris, C., Stephens, M., 1988. A combined corner and edge detector. In: Alvey88, pp. 147–152.

Laptev, I., 2005. On space-time interest points. IJCV 64 (2–3), 107–123.

Laptev, I., Marszalek, M., Schmid, C., Rozenfeld, B., 2008. Learning realistic human actions from movies. In: CVPR08, pp. 1–8.

Liu, J., Shah, M., 2008. Learning human actions via information maximization. In: CVPR08, pp. 1–8.

Muja, M., Lowe, D.G., 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In: International Conference on Computer Vision Theory and Application VISSAPP'09). INSTICC Press, pp. 331–340.

Niebles, J., Wang, H., Fei Fei, L., 2006. Unsupervised learning of human action categories using spatial-temporal words. In: BMVC06, p. III: 1249.

Nowozin, S., Bakir, G., Tsuda, K., 2007. Discriminative subsequence mining for action classification. In: ICCV07, pp. 1–8.

Rapantzikos, K., Avrithis, Y., Kollias, S., 2007. Spatiotemporal saliency for event detection and representation in the 3d wavelet domain: potential in human action recognition. In: CIVR07, pp. 294–301.

Schuldt, C., Laptev, I., Caputo, B., 2004. Recognizing human actions: a local svm approach. In: ICPR04, pp. III: 32–36.

Sivic, J., Zisserman, A., 2003. Video google: A text retrieval approach to object matching in videos, In: ICCV03. pp. 1470–1477.

Souvenir, R., Babbs, J., 2008. Learning the viewpoint manifold for action recognition. In: CVPR08, pp. 1–7.

Ubalde, S., Goussies, N., 2012. Fast non-parametric action recognition. In: CIARP, pp. 268–275.

Wang, L., Suter, D., 2006. Informative shape representations for human action recognition. In: ICPR06, pp. II: 1266–1269.

Wang, Z., Hu, Y., Chia, L., 2009. Learning instance-to-class distance for human action recognition. In: ICIP09, pp. 3545–3548.

Weinland, D., Boyer, E., Ronfard, R., 2007. Action recognition from arbitrary views using 3d exemplars. In: ICCV07, pp. 1–7.

Willems, G., Tuytelaars, T., Van Gool, L., 2008. An efficient dense and scale-invariant spatio-temporal interest point detector. In: ECCV08, pp. II: 650–663.

Yan, P., Khan, S., Shah, M., 2008. Learning 4d action feature models for arbitrary view action recognition. In: CVPR08, pp. 1–7.

Yilmaz, A., Shah, M., 2008. A differential geometric approach to representing the human actions. CVIU 109 (3), 335–351.

Yuan, J., Liu, Z., Wu, Y., 2009. Discriminative subvolume search for efficient action detection. In: CVPR09, pp. 2442–2449.

Yuan, J., Liu, Z., Wu, Y., 2011. Discriminative video pattern search for efficient action detection. PAMI 33 (9), 1728–1743.

Zelnik-manor, L., Irani, M., 2001. Event-based analysis of video. In: Proc. CVPR, pp. 123–130.

Zelnik Manor, L., Irani, M., Weinland, D., Ronfard, R., Boyer, E., 2006. Free viewpoint action recognition using motion history volumes. CVIU 103 (2–3), 249–257.

Zhu, P., Hu, W., Li, L., Wei, Q., 2009. Human activity recognition based on r transform and fourier mellin transform. In: ISVC09, pp. II: 631–640.