Incremental Action Recognition Using Feature-Tree

Kishore K Reddy
Computer Vision Lab
University of Central Florida
kreddy@cs.ucf.edu

Jingen Liu
Computer Vision Lab
University of Central Florida
liujg@cs.ucf.edu

Mubarak Shah
Computer Vision Lab
University of Central Florida
shah@cs.ucf.edu

Abstract

Action recognition methods suffer from many drawbacks in practice, which include (1)the inability to cope with incremental recognition problems; (2)the requirement of an intensive training stage to obtain good performance; (3) the inability to recognize simultaneous multiple actions; and (4) difficulty in performing recognition frame by frame.

In order to overcome all these drawbacks using a single method, we propose a novel framework involving the feature-tree to index large scale motion features using Sphere/Rectangle-tree (SR-tree). The recognition consists of the following two steps: 1) recognizing the local features by non-parametric nearest neighbor (NN), 2) using a simple voting strategy to label the action. The proposed method can provide the localization of the action. Since our method does not require feature quantization, the feature-tree can be efficiently grown by adding features from new training examples of actions or categories. Our method provides an effective way for practical incremental action recognition. Furthermore, it can handle large scale datasets due to the fact that the SR-tree is a disk-based data structure. We have tested our approach on two publicly available datasets, the KTH and the IXMAS multi-view datasets, and obtained promising results.

1. Introduction

Growing interest in visual surveillance has led to research in many promising areas such as human action recognition, crowd behavior analysis, tracking of an individual in crowded scenes, etc. Human action recognition has been one of the most challenging problems studied over the last two decades for reliable and effective solutions. Current solutions have many limitations and are far from solving real world problems.

Most current approaches may not be very useful in solving realistic problems in practice. One problem is that most approaches are not easily scalable, due to the fact that they are based on a predefined training dataset, which is static in nature. However, this may not be the case in most real word problems, since the datasets are dynamically changing. For instance, if we want to add new examples to an existing training set or add an entirely new action category to the training set, one has to update the visual vocabulary and computed histograms (bag of visual words) for bag of words method, or construct new templates using new examples for a template based approach. Performing action recognition on a frame by frame basis is a very important requirement in real world videos. Furthermore, the most challenging task is dealing with videos containing multiple actions happening simultaneously and also having large occlusions. Most methods can only classify videos containing a single action and are not robust to large occlusions.

To overcome these real world challenges, we propose to use a feature-tree data structure. We first use the SR-tree (Sphere/ Rectangle) [1] to generate our feature-tree using the features (spatiotemporal Dollar features [3]) of the labeled training examples. SR-tree is a variant of R-tree [2], which has been widely used in the database community. Instead of splitting the high-dimensional feature space by hyper-planes as is done in earlier tree techniques, the feature points are organized into regions in the feature space, which are specified by the intersection of a bounding sphere and bounding rectangle. Hence, SR-tree can maintain smaller region volume to provide disjoint regions and larger diameter to support fast range and NN search in high-dimensional space. During the recognition phase, we extract features from an unknown action video and classify each feature into an action category using SR-tree. To recognize the action, we adopt a simple voting method by counting the labels of the features.

Our proposed method has many advantages compared to the existing action recognition approaches. First, we can effectively and efficiently integrate indexing and recognition using our proposed feature-tree. As a result of this, we successfully avoid any intensive training (vocabulary construction and category model training). Second, since the feature-tree grows when additional training features are added from the new training examples or a new category, the resultant tree is very useful for incremental action recognition in many realistic applications. Third, the tree provides a disk-based data structure, which makes our recognition system scalable for large scale datasets. Finally, the recognition can be performed nearly in real time. For example, for the KTH dataset, each feature query takes 0.02s on a single core 2 GHZ Pentium processor, and if each frame has about 3 features, it only takes 0.06s for the frame-based recognition, and the performance is still competitive. Our system can also be used to detect occurrence of multiple actions which are happening simulta-

neously and localize them.

The rest of the paper is organized as follows. Section 2 deals with the related work and section 3 presents our proposed method. The experiments are discussed in section 4. Finally, we conclude our work in section 5.

2. Related work

Different approaches have been proposed to solve the problem of action recognition over the past two decades. Spatio-temporal template based approach was one of the early solutions for action recognition problems. Initial work was done by Polana and Nelson [4] and later made popular by Bobick and Davis [5]. Though template based methods can perform detection and localization of multiple actions happening simultaneously, they are computationally intense and highly dependent on training examples to build effective and reliable templates.

Another approach for recognizing action, models the dynamics of human motion using finite state models [6] or hidden Markov models [7][8]. However a great deal of training data is required to build effective and reliable models. Recent work on representing human actions uses poses [9], employing a novel pose descriptor called Histogram-of-oriented-rectangles (HOR) with a velocity descriptor prior to the pose based classification step. In this paper, the subject has to be tracked and the body parts have to be identified, which is a problem when occlusions are present in a scene containing multiple people.

Most recently, bag of features (BOF) has been receiving increasing attention due to its simplicity and surprisingly good performance on object, scene and action recognition problems [3][14][15][22][20]. Inspired by the success of the bag of words approach in text categorization, computer vision researchers have widely and successfully applied the bag of *visual words* model beyond BOF for visual recognition. Vocabulary-based methods (bag of visual words) were proposed to reduce the number of features by quantization for compact visual representation [3][14][20][19][21]. It is generally a two-stage procedure, consisting of vocabulary construction, and category model learning (e.g. with SVM). Different vocabulary construction methods may lead to very different performance. Flat vocabulary, which is normally created by applying *k*-means on the subsamples of the training data [3][15][22] obtains good performance, but generating vocabularies is computationally expensive, mainly due to the cost of clustering of visual features from training examples, and assigning the visual descriptors to *visual words* (especially for generating very large vocabularies). Hierarchical vocabularies constructed using hierarchical clustering techniques [19][17] have been explored recently, and proven to be very efficient and effective in visual recognition.

Tree data structure has been widely used in visual recognition. For instance, *kd*-tree [10] has been used for fast image matching [11]. However, one may need to reconstruct the entire tree if it is seriously unbalanced due to dynamic update [13]. Therefore, *kd*-tree is not scalable and is unable to handle a large scale feature dataset efficiently. Recently, several tree-based approaches have been proposed to construct vocabulary trees. Nister et al. [19] used hierarchical k-means to quickly compute very large vocabularies. Random Forest models have been successfully applied for recognition. Instead of using Random Forest as a classifier, Moosmann et al. [18] used similar techniques to construct random vocabulary trees in a supervised mode. They obtained much better performance than using the unsupervised vocabulary construction approaches such as k-means. As most vocabulary construction methods are training-based, they are unable to update the vocabulary dynamically.

This extremely limits their applications in the real world. In order to overcome this, Yeh et al. [24] proposed a forest-based method to create adaptive vocabularies, which can update the vocabularies without reconstructing the entire trees. This method is efficient for image retrieval; however, it still requires re-training the category models for recognition when the vocabulary has been updated using the new training examples or categories, because the image representations may have changed with the update of vocabularies.

Using trees to index the bag of features and directly perform recognition has not been addressed extensively. Randomized trees have been used in [16] for image view recognition. They used simple binary trees and split a node by checking the entropy of point labels. Boiman et al [12] also used *kd*-tree for fast feature search in their image classification. However, both of the above methods work only when using simple trees which do not support efficient dynamic update. They are not suitable for large scale datasets or applications with dynamic environments.

Little work has been done in the action recognition area using trees to index spatiotemporal features. Instead of detecting spatiotemporal interest points, Mikolajczyk *et al.* [17] detected local static features from each frame and trained the vocabulary forest by hierarchical k-means. They obtained good results on the KTH dataset. However, their approach also faces the problem of dynamic updating of vocabulary, so it is unable to cope with incremental action recognition for most real applications. Besides, their system is not real time.

However, to the best of our knowledge, the use of tree data structures to integrate indexing, recognition, and localization has not been explored for action recognition.

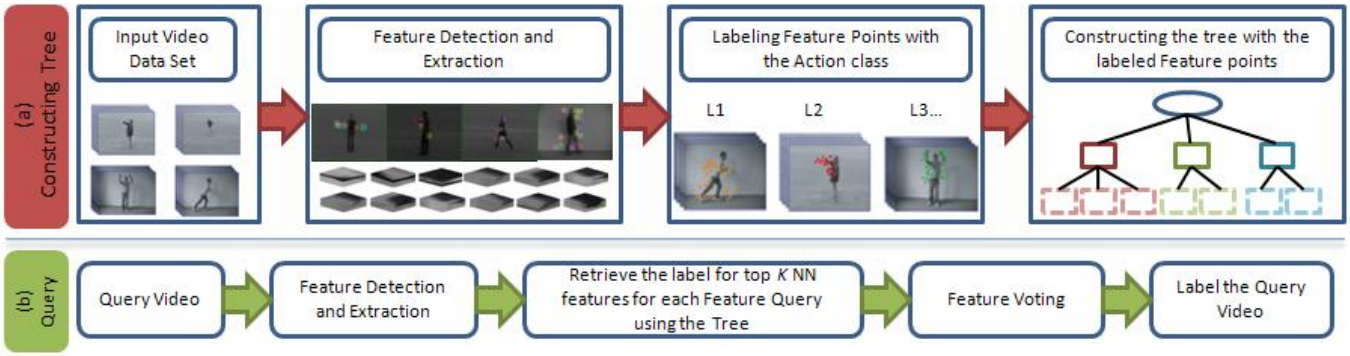


Fig. 1: The proposed framework for action recognition using feature-tree.

3. Proposed method

In this section, we describe our proposed method in detail. Fig. 1 shows the main steps in our method for action recognition using feature tree. There are two phases: Fig. 1 (a) depicts the procedure of construction of the feature-tree and Fig. 1 (b) describes the recognition steps. In the feature-tree construction stage, local spatiotemporal features are detected and extracted from each labeled video, and then each feature is represented by a pair [*d,l*] where *d* is the feature descriptor and *l* is the class label of the feature. Finally, we index all the labeled features using SR-tree. In the recognition stage, given an unknown action video, we first detect and extract local spatiotemporal features, and then for each feature we launch a query into the feature-tree. A set of nearest neighbor features and their corresponding labels are returned for each feature. Each returned nearest neighbor votes for its label. This process is repeated for each feature, and these votes can be weighted based on the importance of each nearest neighbor. Finally, a video is assigned a label, which receives the most votes. The entire procedure does not require intensive training. Therefore, we can easily apply incremental action recognition using the feature-tree. Besides, we also can easily localize the actions in the video, and detect multiple actions occurring simultaneously in the same video.

3.1. Feature Extraction

In order to detect an action in a given video, we use the spatiotemporal interest point detector proposed by Dollar *et al.*[3]. Compared to the 3D Harris-Corner detector [26], it produces dense features that can significantly improve the recognition performance in most cases. This detector utilizes 2-D Gaussian filter and 1-D Gabor filter separately in spatial and temporal directions respectively. A response value is given by the following function at every position *(x, y, t)*:

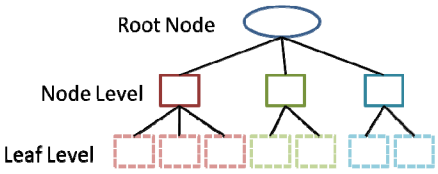
$$R = \{I * g_\sigma(x,y) * h_{ev}(\mathrm{t})\}^2 + \{I * g_\sigma(x,y) * h_{od}(\mathrm{t})\}^2,$$

where $g_\sigma(x,y)$ is the spatial Gaussian filter, and h_{ev} and h_{od} are a quadrature pair of the 1-D Gabor filter in time. It produces high responses to the temporal intensity change points. *N* interest points are selected at the locations of local maximal responses, and 3D cuboids are extracted around them. For simplicity, we used the flat gradient vectors $d_i \in R^N$ to describe the cuboids, and utilized PCA to reduce the descriptor dimension, so a video is represented by a set of cuboids $D = \{d_i\}$.

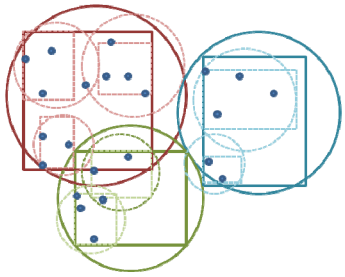
3.2. Constructing Feature-tree

Given a set of action videos $V = \{v_1, v_2,\ldots,v_M\}$ and its corresponding class label $l_i \in \{1,2,\ldots,C\}$, we extract *n* spatiotemporal features $d_i^j (1 \le j \le n)$ from video v_i. We associate each feature with its class label and get a two-elements feature tuple $x_i^j = [d_i^j \; l_c]$. Then we obtain a collection of labeled features $T = \{x_i^j\}$ to construct our feature-tree.

SR-tree is chosen to index the labeled feature collection *T*. It is a disk-based data structure having successfully integrated the advantages of the R-tree and SS-tree [28] data structures, so it is fast for large scale datasets which cannot be fit into the main memory. We briefly review its data structure as follows. Fig 2 shows the nested hierarchy structure with a root node, nodes, and leaves. Instead of splitting the feature space by a hyper plane, SR-tree organizes the data by hierarchical regions. The region is defined by the intersection of a bounding sphere (the smallest sphere covering a set of points) and a bounding rectangle. This design creates a balance between having larger region diameter and smaller region volume to improve the performance of nearest neighbors search. Each leaf of an SR-tree has a minimum and maximum limit on the number of entries it can take. Entries in the leaf store the points with their labels. A leaf is split and converted into a node if it reaches the maximum number of allowed entries. Nodes do have a limit on the number of children they can hold. Each entry in a node has four components: bounding sphere, bounding rectangle, number of points in the sub-tree, and pointers to all its children.



(a) Tree structure.



(b) Node and Leaf level graphical view.

Fig. 2: Tree structure defined by the intersection of bounding spheres and bounding rectangles.

We can consider the regions in our feature-tree as a cluster of feature points which are spatially close to each other.

We can imagine that the same type of actions will share several feature patterns, so each leaf node may have dominated class labels. The feature-tree can grow in three steps:

1. Inserting a new feature point into the tree.
2. Splitting a leaf when it is overcrowded.
3. Reinserting the feature points from step 2.

Insertion: A centroid-based algorithm is applied in the insertion step due to its effectiveness for nearest neighbor queries. The insertion algorithm is supposed to find the best sub-tree that can accommodate the new spatiotemporal feature point x_i^j from T. The sub-tree, whose centroid (i.e. the center of the feature points covered by it) is nearest to x_i^j , is selected. After insertion, the regions (the bounding sphere and the bounding rectangle of the parents) are updated. In fact, the region updates are propagated upwards until the root is reached. The bounding rectangle is updated the same way as it is done in R-tree. The bounding sphere, however, is updated using both the bounding sphere and bounding rectangle of its children. Using both the bounding sphere and bounding rectangle of the children helps the radius of the parent's bounding sphere to be smaller compared to the SS-tree, which in turn reduces the overlap of the bounding spheres.

Splitting: If a leaf is overcrowded, a portion of its entries are reinserted. If the reinsertion fails, it is split and converted into a node and the entries are reinserted.

Deletion: An entry is simply removed if the deletion of the entry causes no under-utilization of any leaf or node. If the deletion does cause under-utilization of a leaf or node, the leaf or node is removed from the tree and the orphaned entries are reinserted into the tree.

Table 1 Nearest Neighbor Search.

Objective: Given a query feature point, find the nearest neighbors
1. Start with the candidate set nearest to the query feature point d_q.
2. Search each leaf having its region intersecting the candidate set.
3. Start with the leaf closest to the query point d_q.
4. If K nearest neighbors d_r are found terminate.
5. If not, go to the next closest leaf whose region intersect with the candidate set, and do this till all the K nearest neighbors are found.

Table 2 Main steps for Action recognition using feature-tree.

Objective: Given a query video Q, assign it a class label.
Feature Extraction:
1. For a given video Q, detect the spatiotemporal interest points using the Dollar detector and extract the cuboids around them.
2. Compute gradient for each cuboid q, and use PCA to reduce the dimension, then represent it by a descriptor d_q.
3. Represent given video Q as a bag of features $\{d_q\}$.
Action Recognition:
Given the query features of Q: $\{d_q\}$
1. For each query feature d_q in Q retrieve the nearest neighbor d_r^q .
2. The class label of Q is decided by equation:
$\hat{C} = \arg\max_C \sum_{q=1}^{M} \sum_{r=1}^{K} \frac{\tau * I_q^r}{\|d_q - d_r^q\| + \epsilon}$.

3.3. Action Recognition

Given the feature-tree constructed during the training, the action recognition starts by extracting spatiotemporal features from an unknown action video. Suppose the query action video is represented by a set of spatiotemporal features, say $Q = \{d_q\}$ ($1 \le q \le M$), our recognition task is to find its class label. Instead of using a complicated learning method, we adopt simple voting schema.

For each query feature d_q, we retrieve the labels of the K nearest neighbors from the feature-tree, and assign it a class label based on the votes of the labels of the K returned features. Then, the label of Q is decided by the final voting of the class labels of the query features in Q. Note that, generally the query features of Q may contain good and bad features (good feature meaning discriminative features). In order to distinguish the votes from them, we can assign a class label to Q using the following equation:

$$\hat{C} = \arg\max_{C} \sum_{q=1}^{M} \sum_{r=1}^{K} \frac{\tau * I_q^r}{\|d_q - d_r^q\| + \epsilon} \qquad (1)$$

where $d_r^q \in \mathrm{NN}(d_q)$, I_q^r is an indicator function which is equal to 1 if the label of d_r is C, is zero otherwise, and ϵ, τ are constant numbers (NN - Nearest Neighbor). Therefore, the contribution of each feature also depends on how well it matches the query feature. Noisy features (bad features) usually do not have good matches.

The nearest neighbor search algorithm implements an ordered depth first traversal. First, it collects the candidate set, and secondly it revises the candidate set by visiting each leaf having its region intersected with the candidate set. The search is terminated if there are no more leaves to visit and the final candidate set is the search result. The search is performed by computing the distance of the search point x_i^j to the region of the child and visiting the child which is closer. Since the region for an SR-tree is the intersection of a bounding sphere and a bounding rectangle, the minimum distance from a search point to a region is defined as the maximum between the minimum distance to the bounding sphere and the minimum distance to the bounding rectangle. We summarize the nearest search procedure and the action recognition steps in Table 1 and Table 2.

(a) Our feature-tree (b) Random Forest

Fig. 3: Confusion table for KTH data set; sequences corresponding to 5 persons were used to construct the tree.

4. Experiments and results

We tested our approach on two publicly available datasets: the KTH dataset [22] and the IXMAS multi-view dataset [23]. The default experiment settings are as follows, unless and until otherwise specified. For each video we extract 200 interest points and corresponding cuboids. Gradient based feature descriptor is used in all the experiments.

4.1. Experiments on the KTH Dataset

We applied our approach to the KTH dataset, which contains six actions: *boxing*, *clapping*, *waving*, *jogging*, *walking,* and *running*. They are performed by 25 actors under four different scenarios of illumination, appearance, and scale changes. In total the data set contains 598 video sequences.

In our experiment we used sequences corresponding to 5 persons to construct the feature-tree and the remaining sequences of 20 persons for testing. This procedure was repeated five times by switching the persons for feature-tree construction and testing. The average accuracy was 87.8%. We used exhaustive NN search instead of the SR-tree with similar experiment setup and got an average performance of 87.8% similar to using SR-tree. But, SR-tree is approximately 30 times faster than exhaustive NN search and is nearly real-time. Instead of using SR-tree data structure to construct our feature-tree as we describe in section 3, we also used Random Forest [25] to create the feature-trees using 5 persons, and tested it on the remaining 20 persons. As before, we repeated this five times. Each time, there were 50 random forest feature-trees, and the final classification was made based on voting from all the feature-trees. The average accuracy was 72.9%. As we know, Random Forest uses binary trees which separate the feature space by a plane when splitting the space at each node to construct the feature-trees. Random forest may be weaker than our proposed feature-tree which separates the feature space by regions. We also did similar experiments using vocabulary-based approach which uses k-means to generate the flat vocabulary, and SVM to train the category model. The average accuracy was about 84.4%. We also observed that the increase in training examples had little effect on the performance. When we increase the training set to 10 persons the performance using proposed SR-tree increases to 90.3%.

Fig. 3 shows the confusion table for the KTH dataset using our feature-tree (a) and Random Forest (b). From the confusion tables provided in Table 2(a), one can observe that the actions clapping and waving are confused with boxing, and there is a lot of confusion between jogging and running. It's observed that the actions involving hand movement get confused with each other. This is also the case with leg movement actions; this is due to the similarity between the actions. This has also been observed in approaches like bag of video words. The Random Forest approach has a high rate of confusion with actions like jogging with running.

Table 3 The performance of the different bag of visual words approaches.

Method	Acc(%)	Method	Acc(%)
Our method	90.3	Nowozin, *et al.[27]*	84.7
Liu, *et al.[14]*	91.3	Dollar *et al.[3]*	80.6
wong, *et al.[22]*	83.9	Schuldt *et al.[15]*	71.7

We list the state-of-the-art results on the KTH dataset using bag of visual words related approaches in table 3. We cannot directly compare them due to different experiment settings. For example, [14] and [23] used 24 persons

for training. Also, in order to perform a fair comparison we do not list some results which were achieved by using spatiotemporal information, which is not used in our approach.

4.2. Effect of number of Features, feature dimensions and Nearest Neighbors

Without exception, in all our experiments in this paper, we extracted 200 spatiotemporal features from each video, and used one nearest neighbor for voting in the feature query phase. In these experiments, we tried to verify how the performance is affected by the number of extracted features and the K value in K nearest neighbors. Fig. 4 shows the results of using different number of features and different K values. It is very impressive to note that using 10 features we can still get about 75% average accuracy. It does not make too much difference when the number of extracted features is larger than 100.

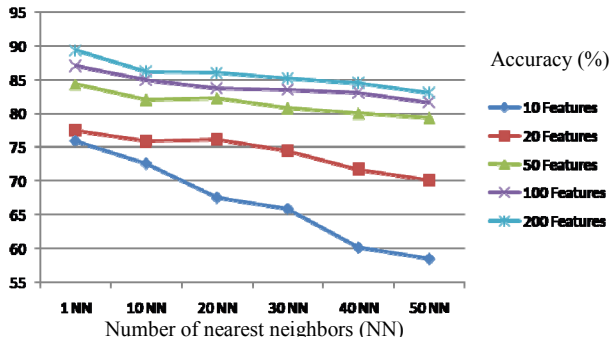


Fig. 4 The performance analysis of the proposed feature tree method. The number of features vary from 10 to 200 and the number of nearest neighbors vary from 1 to 50.

This implies that feature-tree performs well even if only very few feature points are available from each video. As the number of feature points increases to 200, the performance increases to about 89%.

Effect of feature dimension: For feature dimension of 100 we have a performance of 87.8% and we achieve almost the same with dimension of 50 i.e., 87.7%. Reduction in dimension of the feature vector has very small impact on the performance, which can be helpful to decrease the needed computations.

4.3. Incremental action recognition

In the real world we need systems which can adapt to or learn from the dynamic environment. Most vocabulary-based bag of feature methods cannot cope with this situation, since they normally need intensive training for vocabulary construction and/or category modeling using SVM. However, our feature-tree method can deal with this problem. In our approach the feature-tree can be updated with new training examples or new categories without reconstructing the entire tree. In this experiment we again use the KTH data set to show the advantages of having an incremental feature-tree. We first train our tree with 5 people on 4 actions (*boxing*, *clapping*, *waving*, *walking*). We add one person at a time from action 5 to the feature-tree, and analyze the impact. The test data set in this experiment is from 5 actions and the remaining 20 people not included in the training set. The results are shown in Fig. 5. Tree expansion is done by inserting the new feature points into the fully constructed tree. We expand the tree with features from 5 more persons for action 5 (Jogging), which were previously not included in the training set. From our experiment we observe that the performance increases as the tree is expanded with new training data.

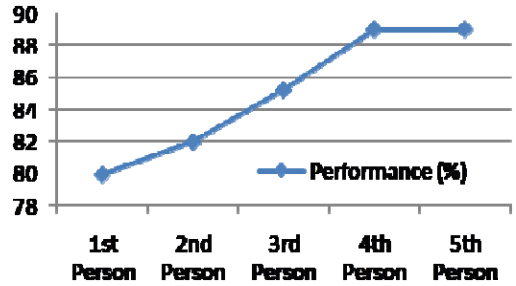


Fig. 5 Plot shows how the recognition rate increases when incremental examples from the new category are added into the feature-tree.

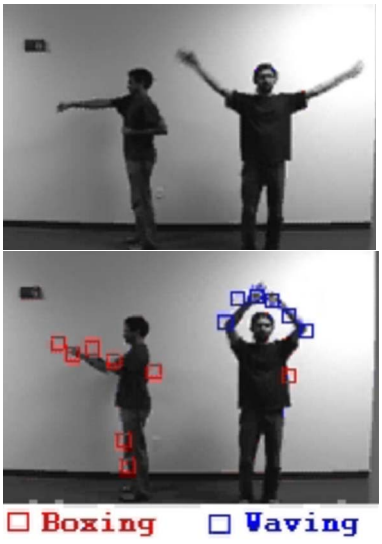


Fig. 6 Classification and localization of two actions happening in the same video. The features assigned boxing labels are shown by red, and the ones assigned waving are shown by blue.

4.4. Recognizing multiple actions in a video

As the aforementioned, the basic bag of video words approaches are unable to detect and localize the actions when multiple actions happen in the same video. However, our

feature-tree can be used to easily handle this case. Given a video with multiple actions, we extract m spatiotemporal features. We conduct feature queries and assign a class label to each of them. We rank them by the class frequency f_c. If $f_c > \Omega$ (a predefined threshold) then we were able to recognize the action and localize it by the locations of the voting features. To simulate this scenario and demonstrate this straightforward method, we recorded a video with two instances: *boxing* and *waving* happening simultaneously. Fig. 6 shows the localization and recognition. All red features correspond to boxing, while the blue correspond to waving. We extracted the feature points from the video. We queried the tree constructed using 5 persons and 6 actions. We indexed all the queried feature vectors with the first nearest neighbor's label and displayed on the video as shown in Fig. 6.

4.5. Action Recognition in frame by frame model

In this experiment, we show that action recognition can be performed in a frame-by-frame mode. This implies that, for an online system, the system does not need to wait for all the frames of the video before making a decision. We can make the decision and update it as more frames become available. Obviously, vocabulary-based approaches cannot be used in a frame-by-frame mode, since the histogram will be meaningful after enough observations on a number of frames are available. In order to verify this

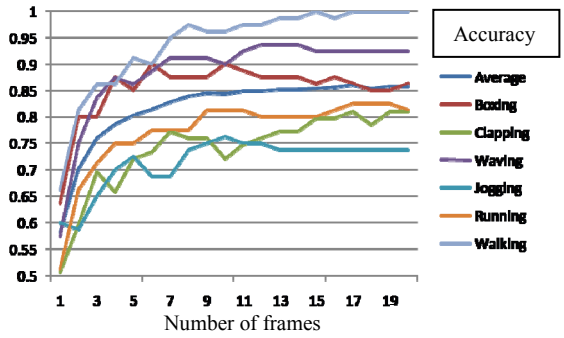


Fig. 7 Performance by increasing the number of frames considered in voting

function, we created a feature-tree with five persons. And then we tested it on the rest of videos in frame-by-frame mode. Fig. 7 shows the results for all the actions. It clearly shows the system can achieve good performance with very few frames, like as few as 3 frames. The recognition performance will be stable after frame 11. After this, more incoming frames do not help improve the performance. The recognition is nearly real-time. In our system, each feature query takes about 0.02s and each frame has about 3 features, and so it is totally 0.06s/frame.

4.6. Experiments on IXMAS Multi-view dataset

We also applied our method on the IXMAS multi-view dataset. The dataset contains 13 daily life actions, performed three times by 12 actors under four different views. In total it contains 1848 video sequences. Sample videos are shown in Fig 8. This data is challenging due to the variations in the camera view, and it also has very similar actions like checking watch, scratching head, and pointing. Here we conduct two different sets of experiments:

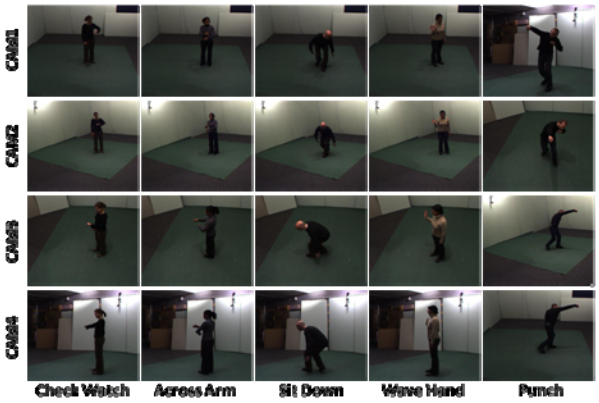


Fig. 8 Four views of five selected actions from IXMAS dataset.

Learning from four views: In the first experiment, learning is done using four views. We use the videos from 5 actors to construct the feature-tree and use the remaining 7 actors for testing. The experiments were repeated four times with videos from different actors to construct the feature-tree. First, we recognized the action using single view, and average accuracies for each camera 1-4 are 69.6%, 69.2%, 62.0% and 65.1%. These results are a little better than those in paper [23], but they are worse than results reported in [14]. However, both of these methods used videos from ten actors for training, where as we only used 5 actors. Besides, both methods require an intensive training process. Next, we tried to recognize actions using simple voting employing four views, and we were able to improve the total average accuracy to about 72.6%. Fig. 9 shows the confusion table for this experiment. Actions like sit-down, get up, turn around, kicking and pickup have an average performance of 89%, but the performance is not that good for the other actions.

Learning from three views: In the second experiment, we used videos from three camera views to construct the feature-tree, and the fourth camera view to test the system. This is repeated four times by changing the camera-views in the training and testing sets. In this experiment, we achieved an accuracy of 81.0%, 70.9%, 79.2%, and 64.9% for cameras 1, 2, 3, and 4 respectively, as shown in Fig 10. These are better than the results reported in [23] and [14].

5. Conclusion

In this paper, we proposed a framework for incremental action recognition using feature-tree. Our approach does not require intensive training (vocabulary construction and category modeling), but we still can get competitive performance and fast recognition. The proposed method is very practical since it can perform incremental action recognition. Also, our method is implemented using disk-based SR-tree data structure. The proposed method has very good scalability. Most importantly, our method can recognize multiple actions simultaneously happening in a video.

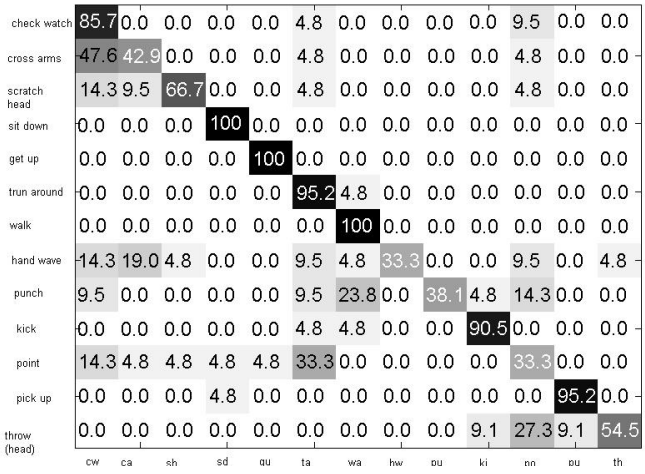


Fig. 9 Confusion table for IXMAS Multi-view dataset.

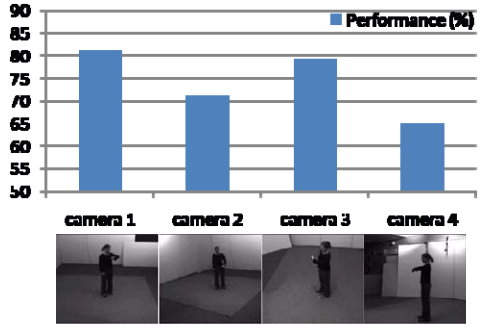


Fig. 10 Performance with a camera view missing.

References

[1] N. Katamaya and S. Satoh, The SR-tree: An index structure for high-dimensional nearest neighbor queries. SIGMOD, 1997.

[2] A. Guttman, R-Trees: A Dynamic Index Structure for Spatial Searching, ACM SIGMOD, 1984.

[3] P. Dollar, V. Rabaud, G. Cottrell and S. Belongie. Behavior recognition via sparse spatio-temporal features, In VS-PETS 2005.

[4] R. Polana, R. Nelson, Detecting activities, IEEE Conf. on Computer Vision and Pattern Recognition, pp. 2–7, 1993.

[5] A. Bobick, J. Davis, The recognition of human movement using temporal templates, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (3) 257–267, 2001.

[6] P. Hong, M. Turk, T. Huang, Gesture modeling and recognition using finite state machines, in: Int. Conf. Automatic Face and Gesture Recognition, pp. 410–415, 2000.

[7] M. Brand, N. Oliver, A. Pentland, Coupled hidden Markov models for complex action recognition, in: IEEE Conf. on Computer Vision and Pattern Recognition, pp. 994–999, 1997.

[8] A. Wilson, A. Bobick, Parametric hidden Markov models for gesture recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 21 (9), 884–900, 1999.

[9] N. Ikizler, P. Duygulu, Histogram of oriented rectangles: A new pose descriptor for human action recognition, Image and Vision Computing 27, 1515-1526, 2009.

[10] J.L. Bentley, Multidimensional Binary Search Trees in database applications, IEEE Trans. Soft. Eng., vol. SE-5, no. 4, pp.333-340, July 1979.

[11] J. S. Beis and D. Lowe, Indexing without invariants in 3D object recognition, IEEE Transaction on PAMI, 21, 10 (1999), pp 1000-1015.

[12] O. Boiman, E. Shechtman and M. Irani, In defense of nearest-neighbor based image classification, CVPR 2008.

[13] N. Katayama and S. Satoh, Experimental evaluation of disk-based data structures for nearest neighbor searching; AMS DIMACS Series, Vol. 59, pp.87-104, 2002.

[14] J. Liu and M. Shah. Learning human action via information maximization, CVPR 2008.

[15] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: A local SVM approach, ICPR 2004.

[16] V. Lepetit, P. Lagger and P. Fua, Randomized Trees for Real-time Keypoint Recognition, CVPR 2005

[17] K. Mikolajczyk and H. Uemura. Action recognition with motion-appearance vocabulary forest, CVPR 2008.

[18] F. Moosmann, B. Triggs and F. Jurie, Fast discriminative visual codebooks using Randomized Clustering Forests, NIPS 2006.

[19] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society.

[20] J. Liu and M. Shah, Scene modeling using co-clustering, ICCV, 2007.

[21] J. Liu, Y. Yang and M. Shah. Learning semantic visual vocabulary using diffusion distance, CVPR 2009.

[22] S. Wong, T. Kim, et al. Learning motion categories using both semantics and structural information, CVPR 2007.

[23] D. Weinland, E. Boyer and R. Ronfard. Action recognition from arbitrary views using 3D examplars, ICCV 2007.

[24] T. Yeh, J. Lee and T. Darrell, Adpative vocabulary forest for dynamic indexing and category learning, ICCV, 2007

[25] Breiman, L. Bagging predictors. Machine Learning, 1996. 26(2), 123–140.

[26] I. Laptev. On space-time interest points, IJCV, 64(2-3): 107-123, 2005.

[27] S. Nowozin, G. Bakir and K. Tsuda. Discrinative subsequence mining for action recognition, ICCV 2007.

[28] D. White and R. Jain, "Similarity Indexing with the SS-tree", IEEE International Conference on Data Engineering, New Orleans, LA. 516-523. February 1996.