Unsupervised Random Forest Indexing for Fast Action Search

Gang Yu, Junsong Yuan
School of Electrical and Electronic Engineering
Nanyang Technological University, Singapore
gyu1@e.ntu.edu.sg, jsyuan@ntu.edu.sg

Zicheng Liu
Microsoft Research
Redmond, WA, USA
zliu@microsoft.com

Abstract

Despite recent successes of searching small object in images, it remains a challenging problem to search and locate actions in crowded videos because of (1) the large variations of human actions and (2) the intensive computational cost of searching the video space. To address these challenges, we propose a fast action search and localization method that supports relevance feedback from the user. By characterizing videos as spatio-temporal interest points and building a random forest to index and match these points, our query matching is robust and efficient. To enable efficient action localization, we propose a coarse-to-fine subvolume search scheme, which is several orders faster than the existing video branch and bound search. The challenging cross-dataset search of several actions validates the effectiveness and efficiency of our method.

1. Introduction

The development of invariant local features and the recent advances in fast subwindow search algorithms [26] have allowed us to search and locate small visual objects within large image databases. It is natural to ask if we can search and locate complex video patterns, e.g., human actions, in a large corpus of videos. Despite previous work in action recognition [2] [5] [9] [12] [25] and detection [14] [10] [7] [28], efficient search of actions remains a largely unsolved problem, mainly because of the following three unique challenges.

First of all, for action search, usually only a single query example is provided. In such a case, the amount of training data is extremely limited and only available at the time of query, whereas in action detection and classification [2] [21], a lot of positive and negative training examples can be leveraged. Therefore it is much more difficult to identify and locate a specific action example in videos. Furthermore, possible action variations such as scale and view point changes, style changes and partial occlusions only worsen the problem, let alone cluttered and dynamic backgrounds.

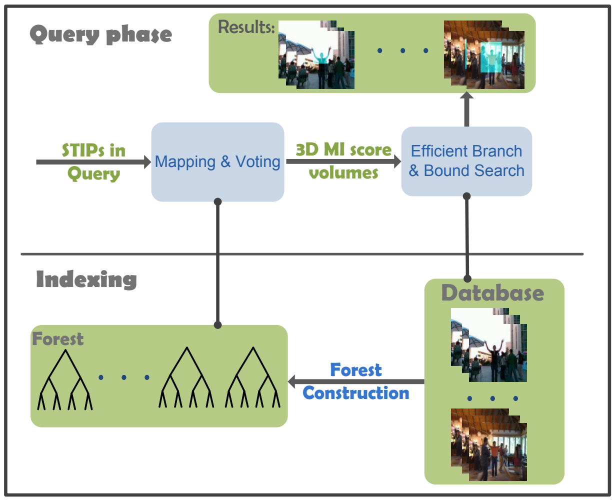

Figure 1. Overview of our algorithm.

Secondly, a retrieval system must have a fast response time because otherwise the user experience would suffer. Unlike video shot or event search, where the goal is to rank pre-segmented video clips, action search is much more difficult as we need not only find the target video, but also locate the action accurately, i.e. identify the spatio-temporal location of the action in the video. For a dataset consisting of tens of hours of videos, such a process is expected to be finished in only a few seconds.

Finally, a retrieval process typically involves user interactions, which allows the user to clarify and update their preferences. Thus, a practical system must have the flexibility to refine the retrieval results by leveraging the labels resulting from subsequent user feedback. Although relevance feedback is popular in image search, there is much less work that supports interactive action search.

We build an action search engine that addresses the above three challenges. An overview of our system is depicted in Fig. 1. Each video is characterized by a collection of interest points, which will be labeled according to the exemplar action in the query phase. The spatio-temporal video subvolumes are cropped out as detections, if most of the interest points inside them match well with the query action. To enable efficient labeling, a random forest is constructed to index these interest points. By improving the branch-and-bound search in [7] and proposing a coarse-to-fine subvolume search strategy, our proposed search method significantly improves the efficiency of the state-of-the-art action detection methods, with comparable search performance. With a single desktop computer, our method can search an hour long video within only 25 seconds. Finally, our method can be easily extended to support interactive search by incrementally adding user labeled actions to the query set. Experiments on cross-dataset search validate the effectiveness and efficiency of our proposed method.

2. Related Work

Action recognition and detection have been active research topics and a lot of work has been done. In [10], a 3D Haar feature based optical flow is proposed to represent 3D volumes. [9] presents a maximum average correlation height filter, with which the intra-class variability is well captured. A visual spacetime oriented energy structure representation is proposed in [16], which is robust to scene clutter and rapid dynamics. In addition to these global template based action representations, local feature description [1] based representations have also been widely used. [2] employs the bag of words (BOW) models based on local feature points [1] and SVM as the classifier. Similar bag of words representations are also discussed in [3] [21]. Apart from BOW representations, Gaussian Mixture Models (GMM) [14] and nearest neighbor (NN) search [7] are alternative solutions to action recognition.

Despite great successes in action recognition and detection, action retrieval, on the other hand, is less exploited. We can roughly categorize most of the existing action retrieval algorithms into two classes based on the number of query samples. Algorithms in the first category [6] [16] perform the sliding window search on the database with a single query sample. One limitation of these techniques is that with a single query sample, it is impossible to model action variations. Besides, an action retrieval system usually involves user interactions but their approaches do not have the capability to incrementally refine their models based on the user feedback. The other category of action retrieval algorithms, for example [19], is based on a set of query samples, usually including both positive and negative samples. Despite the fact that they work well in uncontrolled videos, the computational cost is high and they would fail if insufficient number of query samples are provided. Apart from the above work, there exist some other algorithms in the literature. For example, [13] [22] [23] rely on auxiliary tools like storyboard sketches, semantic words and movie transcripts for action retrieval, while [11] is specifically focused on quasi-periodic events. [24] is doing action retrieval based on static images.

3. Video Representation and Matching

In our method, an action is represented by a set of spatial-temporal interest points (STIP) [1], denoted as $V = \{d_i \in \mathbb{R}^n\}$. Each STIP point d is described by two kinds of features: HOG (Histogram of Gradient) and HOF (Histogram of Flow) and the feature dimension n is 162. For action retrieval, we are given a database with N video clips (each video clip is denoted as $\mathcal{V}_i$), $\mathcal{D} = \{\mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_N\}$. These video clips may contain various types of actions such as handwaving, boxing, and walking. Our objective is, given one or more query videos, referred to as $\mathcal{Q}$, to extract all the sub-volumes which are similar to the query. Formally, that is to find:

$$V^* = \max_{V \subset \mathcal{D}} s(\mathcal{Q}, V), \tag{1}$$

where $s(\mathcal{Q}, V)$ is a similarity function between a set of query video clips $\mathcal{Q}$ and a subvolume V in the database.

Unlike previous single template action detection and retrieval [16], which can only take one positive sample for query, our approach can integrate multiple query samples and even negative ones. By introducing negative samples during the query phase, our algorithm is more discriminative. In addition, this approach enables interactive search by leveraging the labels obtained from user feedbacks.

Following our previous work in [7], we use the mutual information as the similarity function for $s(\mathcal{Q}, V)$. So we have:

$$\begin{aligned} V^* &= \max_{V \subset \mathcal{D}} MI(\mathbf{C} = c_{\mathcal{Q}}, V) \\ &= \max_{V \subset \mathcal{D}} \log \frac{P(V|\mathbf{C}=c_{\mathcal{Q}})}{P(V)} \\ &= \max_{V \subset \mathcal{D}} \log \frac{\prod_{d_i \in V} P(d_i|\mathbf{C}=c_{\mathcal{Q}})}{\prod_{d_i \in V} P(d_i)} \\ &= \max_{V \subset \mathcal{D}} \sum_{d_i \in V} \log \frac{P(d_i|\mathbf{C}=c_{\mathcal{Q}})}{P(d_i)}. \end{aligned} \tag{2}$$

We refer to $s^{c_{\mathcal{Q}}}(d_i) = \log \frac{P(d_i|\mathbf{C}=c_{\mathcal{Q}})}{P(d_i)}$ as the mutual information between STIP d_i and query set $\mathcal{Q}$. In [7], $s^{c_{\mathcal{Q}}}(d_i)$ is computed based on one positive nearest neighbor point and one negative nearest neighbor point from d_i. However, nearest neighbor search in high dimensional space is very time consuming even with the advanced local sensitive hashing (LSH) technique [7]. Second, this approach is sensitive to noise, since only two points are used to compute its score. In order to address these problems, we formulate $s^{c_{\mathcal{Q}}}(d_i)$ as:

$$\begin{aligned} s^{c_{\mathcal{Q}}}(d_i) &= \log \frac{P(d_i|\mathbf{C}=c_{\mathcal{Q}})}{P(d_i)} \\ &= \log \frac{P(d_i|\mathbf{C}=c_{\mathcal{Q}})P(\mathbf{C}=c_{\mathcal{Q}})}{P(d_i)P(\mathbf{C}=c_{\mathcal{Q}})} \\ &= \log \frac{P(\mathbf{C}=c_{\mathcal{Q}}|d_i)}{P(\mathbf{C}=c_{\mathcal{Q}})}. \end{aligned} \tag{3}$$

In Eq. 3, $P(\mathbf{C} = c_{\mathcal{Q}})$ is the prior probability that can be computed as the ratio of the number of positive query

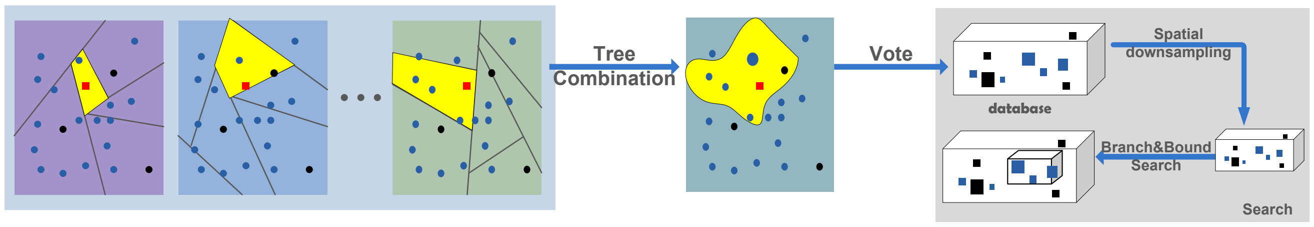

Figure 2. A schematic illustration of random forest based indexing and action search.

STIPs to the total number of query STIPs. In order to estimate $P(\mathbf{C} = c_{\mathcal{Q}}|d_i)$ efficiently, we propose a random forest based space partition strategy.

4. Random Forest based Indexing

Random forest was first proposed to solve the classification problem [17]. Later, it was extended to handle regression and other applications. In recent years, there have been a lot of applications in computer vision, which employ the random forest as the basic data structure, like [4] [20] [18] [27]. In this paper, random forest behaves like a mapping function to estimate $P(\mathbf{C} = c_{\mathcal{Q}}|d_i)$, which is an essential component of our algorithm as illustrated in Fig. 1.

A random forest with N_T trees is built offline from the database. At the query stage, all the STIP points in the query set $\mathcal{Q} = \mathcal{Q}_P \cup \mathcal{Q}_N$ (where $\mathcal{Q}_P$ and $\mathcal{Q}_N$ refer to positive query and negative query, respectively) are first extracted and distributed into the forest. Fig. 2 gives a two-dimension example where blue and black dot points represent the positive and negative STIPs, respectively. Each STIP point $d_i \in \mathcal{D}$ (red sqaure in Fig. 2) falls into one of the leaves of a tree in the forest. Each leaf node contains several STIP points $d_q \in \mathcal{Q}$. In order to compute the posterior $P(\mathbf{C} = c_{\mathcal{Q}}|d_i)$, we integrate the information from all the leaves which contain d_i. Suppose d_i falls into a leaf with N_k^+ positive query STIP points and N_k^- negative points for tree T_k, then $P(\mathbf{C} = c_{\mathcal{Q}}|d_i)$ can be computed as:

$$P(\mathbf{C} = c_{\mathcal{Q}}|d_i) = \frac{1}{N_T}\sum_{k=1}^{N_T}\frac{N_k^+}{N_k^+ + N_k^-}. \tag{4}$$

As can be seen from Eq. 4, our voting strategy can integrate negative query samples, which makes our algorithm more discriminative.

Eq. 3 can hence be rewritten as:

$$\begin{aligned} s^{c_{\mathcal{Q}}}(d_i) &= \log P(\mathbf{C} = c_{\mathcal{Q}}|d_i) - \log P(\mathbf{C} = c_{\mathcal{Q}}) \\ &= \log \tfrac{1}{N_T}\textstyle\sum_{k=1}^{N_T}\tfrac{N_k^+}{N_k^+ + N_k^-} - \log P(\mathbf{C} = c_{\mathcal{Q}}). \end{aligned} \tag{5}$$

However, in the case where there are no negative query samples available ($\mathcal{Q}_N = \emptyset$), we slightly modify Eq. 4 to:

$$P(\mathbf{C} = c_{\mathcal{Q}}|d_i) = \frac{1}{N_T}\sum_{k=1}^{N_T}\frac{N_k^+}{M}, \tag{6}$$

where M is a normalization parameter. And Eq. 5 can be written as

$$\begin{aligned} s^{c_{\mathcal{Q}}}(d_i) &= \log \tfrac{1}{N_T}\textstyle\sum_{k=1}^{N_T}\tfrac{N_k^+}{M} - \log P(\mathbf{C} = c_{\mathcal{Q}}) \\ &= \log \tfrac{1}{N_T}\textstyle\sum_{k=1}^{N_T} N_k^+ - \log M - \log P(\mathbf{C} = c_{\mathcal{Q}}). \end{aligned} \tag{7}$$

We further introduce a parameter $A = -\log M - \log P(\mathbf{C} = c_{\mathcal{Q}})$, which will be tuned in the experiments.

To build a random forest, we use a method motivated by [4], where the random forest is used for voting object locations. The differences from [4] will be discussed at the end of this section.

Assume we have N_D STIP points in the dataset, denoted as $\{x_i = (x_i^1, x_i^2), i = 1, 2, \cdots, N_D\}$; $x_i^1 \in \mathbb{R}^{72}$ and $x_i^2 \in \mathbb{R}^{90}$ are the HOG feature and HOF feature, respectively. In order to build a tree and split the dataset, a random number $\tau \in \{1, 2\}$ is first generated to indicate which kind of feature to use for splitting ($x_i^{\tau=1}$ refers to HOG feature and $x_i^{\tau=2}$ means HOF feature.) Then two more random numbers e_1 and e_2 will be generated which are the dimension indices of the feature descriptor (either HOG feature or HOF feature depending on the value of τ.) After that, a "feature difference" can be evaluated with $D_i = x_i^\tau(e_1) - x_i^\tau(e_2), i = 1, 2, \cdots, N_D$. Based on all the D_i, we can estimate the mean and variance of the feature difference.

To put it briefly, a hypothesis (with variables τ, e_1 and e_2) can be generated with the following three steps:

- Generate $\tau \in \{1, 2\}$ to indicate the type of feature to use
- Generate the dimension indexes e_1 and e_2 and compute the feature difference $D_i = x_i^\tau(e_1) - x_i^\tau(e_2), i = 1, 2, \cdots, N_D$
- Split the dataset into two parts based on the mean of feature differences and obtain a variance

We generate γ hypotheses ($\gamma = 50$ in our experiments) and find the one with the largest variance on feature difference. Usually, a larger variance means that the data distribution spreads out more and the feature difference is more significant. Therefore the corresponding mean is used as the threshold to split the dataset. After this, one node will be built and the dataset will be partitioned into two parts. For each part, a new node will be further constructed in the same way. This process is repeated until the predefined maximum depth is reached.

Each tree can be thought of as a feature space partition as shown in Fig. 2. Usually, STIP points in the same leaf node are similar. The score evaluation (Eq. 5) on the forest can be explained intuitively by a dyeing process. We can think of each positive query STIP point as having a blue color and a negative point as having a black color. For each query point, we pass it down each tree in the forest. The leaf that the point falls in is dyed in the same color as the query point. Each leaf keeps a count of the number of times it is dyed by blue and a count of the number of times it is dyed by black after we pass all the positive and negative query points down the trees. If a leaf's blue count is larger than the black count, it is more likely to belong to the positive region, and vice versa. Given a point d_i (red square point in Fig. 2) in the dataset, to compute its score with respect to the positive queries, we pass it down each tree of the forest. From each tree, we find the leaf that d_i falls in. The blue counts and black counts of all the leafs in all the trees that d_i falls in are combined to estimate its posterior $P(\mathbf{C} = c_{\mathcal{Q}}|d_i)$. The function of random forest is like a special kernel, as shown by the yellow regions in Fig. 2.

The general benefits of random forest are numerous, most of which have already been discussed in [4]. In this paper, we point out four properties of the random forest that are essential for us. First, each tree in the forest is almost independent to others, which is an important property for us to evaluate $P(\mathbf{C} = c_{\mathcal{Q}}|d_i)$, because the independence can greatly reduce the variance of estimation. Second, a random forest is fast to evaluate during the query stage. The computation time only depends on the number of trees and the depth of each tree. Hence, it is usually faster than LSH based nearest neighbor search [7]. In the experiments, we will show that our random forest based weighting approach is over 300 times faster than LSH based approaches. This is of great importance if we want to perform real-time action analysis. Another advantage of random forest compared with LSH is that, during the construction of each tree, data distribution of the STIPs is integrated, which means the tree construction is guided by the data density. This is one reason why random forest has great speed gain but little performance loss. From that aspect, the structure of random forest is flexible. Finally, by adding more trees to the forest, we can alleviate the affection of lacking query samples. As shown in Fig. 2, for each tree, only a small portion of nearest neighbors can be found. One main benefit of using multiple trees is to improve the accuracy of the nearest neighbor search.

We further compare our technique with other work that also use random forests. First, our random forest is constructed in an unsupervised manner for class-independent video database indexing, while traditional random forests are constructed in a supervised manner for object detection (e.g. [4].) Second, in [4], random forest is used to vote for the hypothesized center positions through Hough voting, while our random forest does not rely on Hough voting. Thus, the non-trivial scale estimation of [4] is partially solved through our branch and bound search, which avoids an exhaustive search of all possible scales. Third, our random forest generates both positive and negative voting scores, thus it is more discriminative compared to [4], which generates only positive votes based on the frequency. Finally, we use random forest for density estimation, which has been less exploited before.

5. Efficient Action Search

5.1. Coarse-to-fine Subvolume Search Scheme

After computing the scores for all the STIP points in the database, we follow the approach in [7] to search for subvolumes in each video in the database. However, as stated in [15], there are two limitations in the subvolume search method proposed by [7]. First, we need to run multiple rounds of branch and bound search if we want to detect more than one instance. In addition, the computational cost is extremely high when the video resolution is high. In this paper, we extend the ideas from [15] to address the above two problems. Spatial-downsampling is presented in [15] to handle the high resolution videos and reduce the computational cost. Suppose the downsampling factor is s, for each $s \times s$ patches in the original video space, we add them together to form one point in the downsampled space. [15] proposed an error bound for this downsampling strategy:

$$f^s(\tilde{V}^*) \geq (1 - \frac{s * h + s * w + s^2}{wh}) f(V^*), \qquad (8)$$

where $\tilde{V}^* = argmax_{V \in \mathcal{D}^s} f^s(V)$ denotes the optimal subvolume in the downsampled search space $\mathcal{D}^s$, $f(V) = MI(\mathcal{Q}, V)$, and V^* refers to the optimal subvolume in the original search space with width w and height h.

Suppose we want to retrieve the top K results from the database, we divide our search into two steps. In the first step, we employ the top-K search in [15] and with spatial downsampling $s = 16$ to obtain the first round results. After that, we can estimate a threshold, denoted as θ, based on the Kth largest subvolume score (denoted as $f^s(\tilde{V}^{(K)})$.) For example, if we set downsampling factor $s = 16$ and assume

$w = h = 64$, then our approximation has an average error:

$$\frac{s*h + s*w + s^2}{wh} = 56.3\%. \tag{9}$$

So we choose $\theta = 0.437 f^s(\tilde{V}^{(K)})$ to filter the first round results. Then, for each remaining subvolume $\tilde{V}^{(k)}$ from the first round, we extend the spatial size with 8 pixels in each direction and perform a λ search [15] with $\lambda = f^s(\tilde{V}^{(K)})$ and downsampling factor $s = 8$. As shown in Table 2, our efficient two-round branch-and-bound search only costs 24.1 seconds to search a database of one hour long 320×240 videos.

5.2. Refinement with Hough Voting

Although our search algorithm can successfully locate the retrieved actions, the localization step may not be accurate enough, as can be seen from the first row of Fig. 6. This motivates us to add a refinement step. Suppose we already have the initial results from the down-sampled branch and bound search, for all the STIP points within the detected subvolume, we match with the query video clip, either by random forest or Nearest Neighbor search. Then the shift from the matched STIPs in the query will vote for the center of the retrieved action. After considering all the votes, the center of the retrieved action is the position with the largest vote. The spatial scale of the action is extended to include the initial retrieved region and the temporal scale is fixed to the initial retrieved result. Both quantitative results, as shown in Fig. 4, and empirical results in Fig. 6 show that the refinement step can successfully improve our retrieved results.

5.3. Interactive Search

The performance of our action retrieval system is constrained by the limited number of queries. To show that our retrieval system can achieve better results when more queries are provided, we add an interaction step to facilitate human interaction. There are two major advantages of the interaction step. The first is to allow the user to express what kind of action he/she wants to retrieve. Another advantage is that our system can benefit from more query samples after each round of interaction.

To implement the system, we first perform one round of action retrieval based on a few query samples. After that, the user would label D (D=3 in our experiments) detections with the largest scores. Then the D newly labeled subvolumes will be added into the query set for the next round of retrieval. Detailed results will be discussed in the experiment section.

6. Experimental results

To validate our proposed algorithm, two experiments are discussed in this section. In order to give a quantitative comparison with other work, we present a cross-dataset action detection experiment first. After that, we show the performance of our action retrieval system. Finally some retrieval results are provided.

6.1. Action Detection

We validate our random forest based indexing strategy with a challenging action detection experiment. Since handwaving is one of the actions, that have a lot of practical uses, we propose to detect handwaving actions in this experiment. We first train the model with KTH dataset (with 16 persons in the training part) and then perform experiments on a challenging dataset (MSR II) of 54 video sequences, where each video consists of several actions performed by different people in a crowded environment. Each video is approximately one minute long.

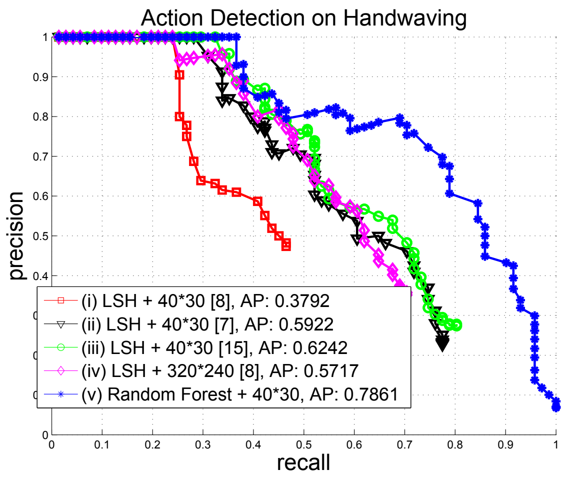

Figure 3. Precision-recall curves for handwaving detection. AP in the legend means the average precision.

Fig. 3 compares the precision-recall curves for the following methods (the resolution for the original videos is 320×240):

(i) ASTBB (Accelerated Spatio-Temporal Branch-and-Bound search) [8] in low resolution score volume (frame size 40×30),

(ii) Multi-round branch-and-bound search [7] in low-resolution score volume (frame size 40×30),

(iii) Top-K search in down-sampled score volume [15] (size 40×30),

(iv) ASTBB [8] in 320×240 videos,

(v) Random forest based voting followed by Top-K search in down-sampled score volume (size 40×30).

The first four methods ((i)-(iv)) employ the LSH based voting strategy [7]. The measurement of precision and recall is the same as those described in [7]. To compute the precision we consider a true detection if : $\frac{\text{Volume}(V^* \cap G)}{\text{Volume}(G)} > \frac{1}{8}$,

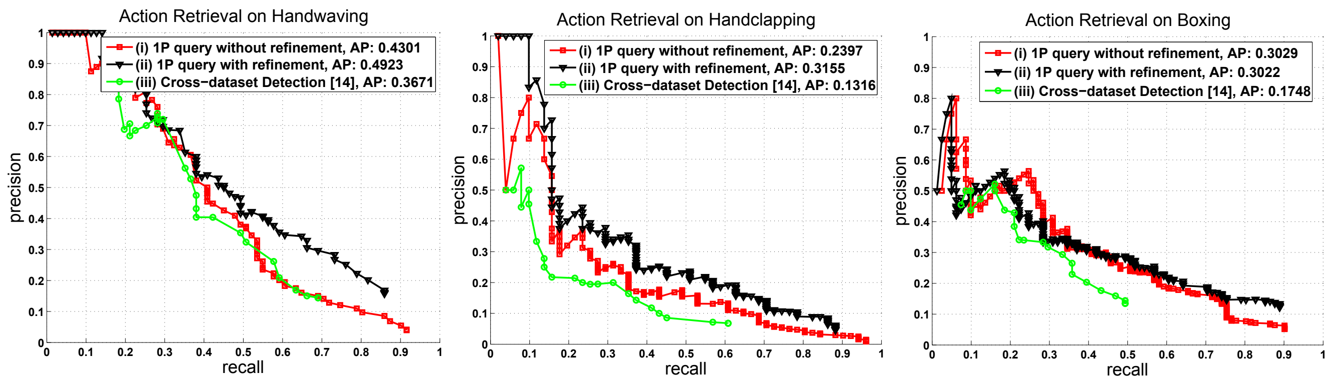

Figure 4. Precision-recall curves for action search.

where G is the annotated ground truth subvolume, and V^* is the detected subvolume. On the other hand, to compute the recall we consider a hit if: $\frac{\text{Volume}(V^* \cap G)}{\text{Volume}(V^*)} > \frac{1}{8}$. According to Fig. 3, our random forest based action detection outperforms the other algorithms. Compared with LSH voting strategy ((i)-(iv)), it shows that our unsupervised random forest based voting score is more discriminative and robust. The underlying reason is that our random forest is data-sensitive, i.e. we model the data distribution when constructing the trees. Besides, since LSH only uses two nearest neighbors for voting, the results are easily corrupted by noise. In [28], random forest is also employed to do action detection. However, the difference is that trees in [28] are constructed in a supervised manner, which means that the label information is utilized when splitting the nodes, while our random forest is unsupervised built with the purpose of modeling the underlying data distribution.

6.2. Action Retrieval

To give a quantitative result for our action retrieval system, we use videos from MSR II as the database. The query samples are randomly drawn from KTH dataset. As little work has been done before on MSR II dataset for action retrieval, to verify our system, we decide to compare our retrieval results with several action detection results from previous researches. The evaluation is the same as that for action detection. For the implementations of our random forest, we set the number of trees in a forest $N_T = 550$ and the maximum tree depth to 18. Fig. 4 compare the following three strategies on handwaving, handclapping and boxing actions (for the boxing action, we flip each frame in the query video so that we can retrieve the boxing coming from both directions), respectively.

(i) One positive query example without Hough refinement,
(ii) One positive query example with Hough refinement,
(iii) Cross-Dataset detection [14][1],

[1]The STIP features in [14] are extracted in video resolution of 160 × 120 but 320 × 240 for other methods

As shown in Fig. 4, with a single query, our results ((i) and (ii)) are already comparable to (iii) for all three action types. This is quite encouraging because (iii) used all the training data while we only use a single query. Besides, our Hough refinement scheme (ii) improves the results without Hough refinement (i).

Fig. 5 shows the experimental results of interactive action retrieval. The following six strategies (all of them are performed without Hough refinement) are compared.

(i) One query example with random forest based voting,
(ii) One query example with NN based voting,
(iii) One positive and one negative query examples
(iv) Two positive and two negative query examples,
(v) One iteration of user interaction after (i),
(vi) Two iterations of user interaction after (i).

We can see that when there is only one query example, our random forest based voting strategy (i) is superior to NN based voting strategy (ii). When there are two query examples (one positive and one negative,) the retrieval results become worse than the one query case. The reason is that negative action type is hard to describe and a single example is usually not enough. However, the performance of our system increases as more query samples are given. In particular, after two interaction steps, our retrieval results are better than the results obtained by other action detection systems ((i)-(iv) in Fig. 3), which utilize all the training data (256 examples).

We also provide some search results in Fig. 6 and Fig. 7 for our action retrieval system on different kinds of actions. For each query, seven subvolumes with the highest scores are listed in the figure. The retrieved subvolumes are marked by colored rectangles. The rectangle with cyan background indicates a “correct” retrieval. As shown in the first row of Fig. 6, some of the cyan results are focused on a subregion of the action region. But this can be relieved with Hough refinement as indicated in the second row. In short, our action retrieval system can get very good results among the top retrieved subvolumes on various actions types.

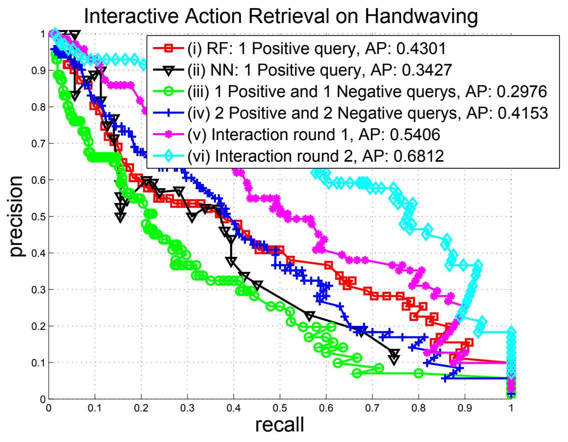

Figure 5. Precision-recall curves for the interactive action retrieval.

6.3. Computational Cost

For our action retrieval system, there are two major runtime costs: voting and searching. With the help of random forest structure, our voting cost is very low (Table 1). This is a major improvement over the time cost of LSH. On the other hand, as shown in Table 2, our coarse-to-fine subvolume search scheme only costs 24.1s for all 54 video clips in MSR II, while Top-K search in [15] takes 26mins. This is even 2800 times faster than the 3D branch and bound search in [7]. To set the parameter θ in Section 5.1, we average the w and h among the top K results and obtain an error bound based on the estimated w and h. With this error bound, we can compute θ similarly as in Eq. 9.

The total computational cost for our system is listed in Table 2. The testing environment is as follows. We use one query video, which is approximately 20 seconds long. The database consists of 54 sequences with 320×240 resolution from MSR II. We use a PC with 2.6G CPU and 3G memory. As shown in Table 2, it takes only 24.7s to retrieve the top-7 results in a database and another 2s to refine them. Besides, the search time is independent of the duration of the query videos. This means, when there are more queries, the total computation time only grows linearly with the feature extraction time, which is around 30s for a 20s sequence. For a very large database, like Youtube, it has little impact to our voting cost since the voting cost mainly depends on the number of trees and the depth of each tree. In order to deal with the increasing search complexity, parallel computing can be utilized in the first step of branch and bound search since the search for different video clips are mutually independent. As the number of candidates for search in the second step of our branch and bound search only depends on the number of retrieved results required by the user, the database size has little impact on the runtime cost for the second step.

Method	Voting Time (ms)	One sequence (s)
LSH [7]	173.48±423.71	1734.8
Random Forest	0.537±0.14	5.37

Table 1. CPU time consumed by STIP voting in a database that consists of 870,000 STIPs. The second column is the CPU time for computing the votes of all STIPs in the database with respect to a single STIP in the query. The third column is the CPU time for computing the votes with respect to a 1 minute long query video (approximately 10,000 STIPs).

Voting time (s)	0.6
Search time (s)	24.1
Refinement time (s)	2
Total Computation Time (s)	26.7

Table 2. Total computation time of our retrieval system. Suppose the query video is around 20s and the database consists of 54 high resolution videos with total length of one hour. Our algorithm retrieves the top 7 subvolumes from the database as shown in Fig. 6.

7. Conclusion

We have developed a random forest based voting technique for action detection and search. This method has the unique property that it is very easy to leverage feedback from the user. In addition, the interest point matching is much faster than the existing nearest-neighbor-based method. To handle the computational cost in searching the large video space, we propose a coarse-to-fine subvolume search scheme, which results in a dramatic speedup over the existing video branch-and-bound method. Cross-dataset experiments demonstrate that our proposed method is not only fast to search higher-resolution videos, but also robust to action variations, partial occlusions, and cluttered and dynamic backgrounds. The handwaving detection results on the MSR II action dataset show that our proposed approach outperforms the state-of-the-art methods. Finally, our interactive action search results show that the detection performance can be improved significantly after only a few rounds of relevance feedback.

Acknowledgement

This work is supported in part by the Nanyang Assistant Professorship (SUG M58040015) to Dr. Junsong Yuan.

References

[1] I. Laptev, "On space-time interest points," *International Journal of Computer Vision*, vol. 64, no. 2-3, pp. 107–123, 2005.

[2] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local svm approach," in *Proc. IEEE Conf. on Pattern Recognition*, 2004.

[3] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *Proc. IEEE Conf. on CVPR*, 2008.

[4] J. Gall, and V. Lempitsky, "Class-specific Hough forests for object detection," in *Proc. IEEE Conf. on CVPR*, 2009.

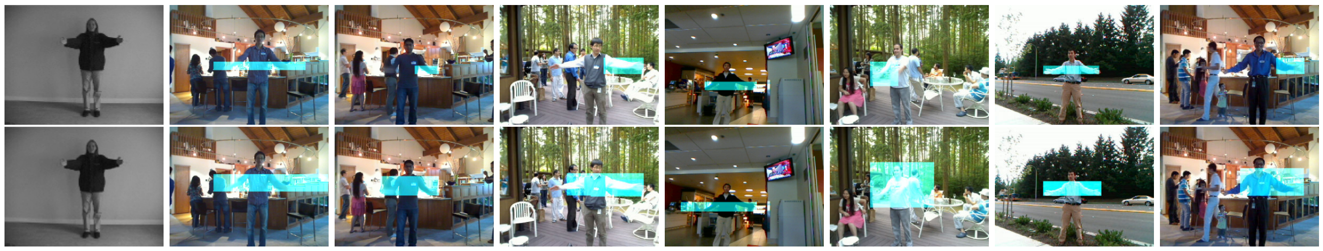

Figure 6. Comparison of retrieval results without and with Hough refinement. For each row, the first image indicates the query sample and the following 7 images refer to the highest ranked retrieved results. All the experiments are done with only one query sample without user feedback. The upper and lower rows are the experiments on handclapping without and with Hough refinement, respectively. The query clip (first column) is from KTH while the database is from MSR II.

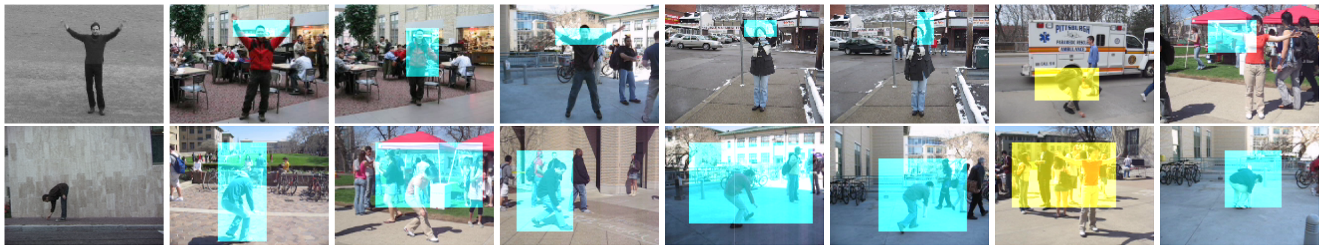

Figure 7. Retrieval results given a single query video. For each row, the first image indicates the query sample and the following 7 images refer to the highest ranked retrieved results. All the experiments are done with only one query sample without any user feedback. For the first row, the query clip with handwaving action is from KTH while the database is from [6]. For the second row, the query clip with bending action is from Weizmann Dataset while the database is from [6].

[5] K. Reddy, J. Liu, and M. Shah, “Incremental action recognition using feature-tree,” in *Proc. IEEE Intl. Conf. on Computer Vision*, 2009.

[6] Y. Ke, R. Sukthankar, and M. Hebert, “Event detection in crowded videos,” in *Proc. IEEE International Conf. on Computer Vision*, 2007.

[7] J. Yuan, Z. Liu, and Y. Wu, “Discriminative subvolume search for efficient action detection,” in *Proc. IEEE Conf. on CVPR*, pp. 2442-2449, 2009.

[8] J. Yuan, Z. Liu, Y. Wu, and Z. Zhang., “Speeding up spatio-temporal sliding-window search for efficient event detection in crowded videos,” in *ACM Multimeida Workshop on Events in Multimedia*, 2009.

[9] D. Mikel, J. Ahmed, and M. Shah, “Action mach a spatio-temporal maximum average correlation height filter for action recognition,” in *Proc. IEEE Conf. on CVPR*, 2008.

[10] Y. Ke, R. Sukthankar, and M. Hebert, “Efficient visual event detection using volumetric features,” in *Proc. IEEE ICCV*, 2005.

[11] P. Wang, G.D. Abowd and J.M. Rehg, “Quasi-periodic event analysis for social game retrieval,” in *Intl. Conf. on Computer Vision*, 2009.

[12] Z. Lin, Z. Jiang, and L. Davis, “Recognizing actions by shape-motion prototype trees,” in *Proc. IEEE Intl. Conf. on Computer Vision*, 2009.

[13] J. Collomosse, and G. McNeill, and Y. Qian, “Storyboard sketches for content based video retrieval,” in *IEEE Intl. Conf. on Computer Vision*, 2009.

[14] L. Cao, Z. Liu and T. Huang, “Cross-dataset Action Detection,” in *Proc. IEEE Proc. CVPR*, 2010.

[15] A. Norbert, Z. Liu, and J. Yuan, “Efficient search of Top-K video subvolumes for multi-instance action detection,” in *Proc. IEEE Conf. on Multimedia Expo (ICME)*, pp. 328-333, 2010.

[16] K.G. Derpanis, M. Sizintsev, K. Cannons, and R.P. Wildes, “Efficient action spotting based on a spacetime oriented structure representation,” in *Computer Vision and Pattern Recognition*, 2010.

[17] L. Breiman, “Random forests,” in *Machine learning*, Vol.45, pp.5-32, 2001.

[18] K. Mikolajczyk, and H. Uemura, “Action recognition with motion-appearance vocabulary forest”,” in *Computer Vision and Pattern Recognition (CVPR)*, 2008.

[19] I. Laptev and P. Prez, “Retrieving actions in movies,” in *Proc. IEEE Intl. Conf. on Computer Vision (ICCV)*, 2007.

[20] A. Yao, J. Gall and L. V. Gool, “A Hough Transform-Based Voting Framework for Action Recognition”,” in *Computer Vision and Pattern Recognition (CVPR)*, 2010.

[21] J.C. Niebles, H. Wang, and L. Fei-Fei, “Unsupervised learning of human action categories using spatial-temporal words,” in *International Journal of Computer Vision*, Vol.3, pp.299-318, 2008.

[22] Y. Aytar, M. Shah, J. Luo, “Utilizing semantic word similarity measures for video retrieval,” in *Computer Vision and Pattern Recognition*, 2008.

[23] A. Gaidon, M. Marszalek and C. Schmid, “Mining visual actions from movies” in *British Machine Vision Conference*, 2009.

[24] N. Ikizler-Cinbis, R.G. Cinbis, and S. Sclaroff, “Learning actions from the web,” in *Proc. IEEE Intl. Conf. on Computer Vision*, 2009.

[25] A. Kovashka and K. Grauman, “Learning a hierarchy of discriminative space-time neighborhood features for human action recognition,” in *Computer Vision and Pattern Recognition (CVPR)*, 2010.

[26] C.H. Lampert, “Detecting Objects in Large Image Collections and Videos by Efficient Subimage Retrieval,” in *Proc. IEEE Intl. Conf. on Computer Vision*, 2009.

[27] C. Marsala, and M. Detyniecki, “High scale video mining with forests of fuzzy decision trees,” in *Proc. Intl. Conf. on Soft computing as transdisciplinary science and technology*, 2008.

[28] G. Yu, A. Norberto, J. Yuan, Z. Liu, “Fast Action Detection via Discriminative Random Forest Voting and Top-K Subvolume Search,” in *IEEE Trans. on Multimedia*, 2011 (in press).