Chapter 2
Supervised Trees for Human Action Recognition and Detection

Abstract Efficient action detection in unconstrained videos is as challenging problem due to the cluttered backgrounds and the large intra-class variations of human actions. In this chapter, we characterize a video as a collection of spatio-temporal interest points, and locate actions via searching for spatio-temporal video subvolumes of the highest mutual information score toward each action class. A random forest is constructed to efficiently generate discriminative votes from individual interest points, and a fast top-K subvolume search algorithm is developed to find all action instances in a single round of search. Without significantly degrading the performance, such a top-K search can be performed on down-sampled score volumes for more efficient localization. Experiments on a challenging MSR Action Dataset II validate the effectiveness of our proposed multiclass action detection method.

Keywords Action detection · Top-K subvolume search · 3D Branch-and-bound search · Spatial downsampling · Random forest voting

2.1 Introduction

According to the literature, human action recognition and detection have been widely exploited. Template matching-based approach [4, 12] and tracking-based approach [14] work well in some constrained environment. However, there are still two major challenges to address.

First of all, for the template matching method, only a single template is usually provided to perform action recognition and detection [4, 12]. In such a case, a single template cannot well characterize the intra-class variations of an action and is not discriminative enough for classification. Second, different from object detection, the search space in the spatio-temporal video space is extremely large. It thus greatly increases the computational cost for these template-based approaches. For example, it is very time-consuming to search actions of different spatial scales and different temporal durations in the video space. Although the recently proposed spatio-temporal branch-and-bound search method [16, 18] can significantly improve the search speed, it is still not fast enough to handle high-resolution videos (e.g., 640 ×

G. Yu et al., *Human Action Analysis with Randomized Trees*,
SpringerBriefs in Signal Processing, DOI 10.1007/978-981-287-167-1_2

480 and higher). Considering that the spatio-temporal localization is computationally more demanding for high resolution videos, it is important to provide efficient solutions for high-resolution videos. Moreover, given a video dataset containing multiple action instances, it is desirable to efficiently detect all of them in one round of search.

To address the above challenges in action recognition and detection, we propose a random forest-based template matching method, as shown in Fig. 2.1. Without performing background subtraction or human body tracking, each video sequence is characterized by a collection of spatio-temporal interest points (STIPs). During the training phase, random forest is constructed to leverage the distribution of the STIPs from both positive and negative classes in the high-dimensional feature space. During the testing phase, each individual point matches the query class through the pre-built random forest, and provides an individual voting score toward each action type. Following the mutual information maximization formulation in [16], action detection becomes finding the spatio-temporal video subvolume with the maximum mutual information score.

Compared with the nearest neighbor-based matching scheme in [16], our proposed random forest-based approach enables a much more efficient interest point matching without degrading the matching quality. Meanwhile, as both positive and negative action samples are taken into account while building the random forest, our proposed method not only handles intra-class action variations well, but also provides more discriminative matching to detect action instances. To reduce the computational overhead in searching high resolution videos, we improve the original spatio-temporal branch-and-bound search method in [16] on two aspects. First, instead of performing branch-and-bound search in the original score volume, we propose to search a down-sampled score volume for efficient action localization. Our theoretical analysis shows that the error between the optimal solution of the down-sampled volume and that of the original volume can be upper bounded. Second, we propose a top-K search method to enable the detection of multiple action instances simultaneously in a single round of branch-and-bound search. It provides an efficient solution for multiclass multiple instance action detection.

To evaluate the efficiency and generalization ability of our proposed method, we perform a cross-dataset action detection test: our algorithm is trained on the KTH dataset and tested on the MSR action dataset II, which contains 54 challenging video sequences of both indoor and outdoor scenes. The extensive multiclass action detection results show that, ignoring the feature extraction cost, our proposed method can search a one-hour 320×240 video sequence in less than half an hour. It can detect actions of varying spatial scales, and can well handle the intra-class action variations including performing style and speed variations, and even partial occlusions. It also can handle cluttered and dynamic backgrounds. The proposed Top-K volume search algorithm is general and can be used for any other applications of video pattern search.

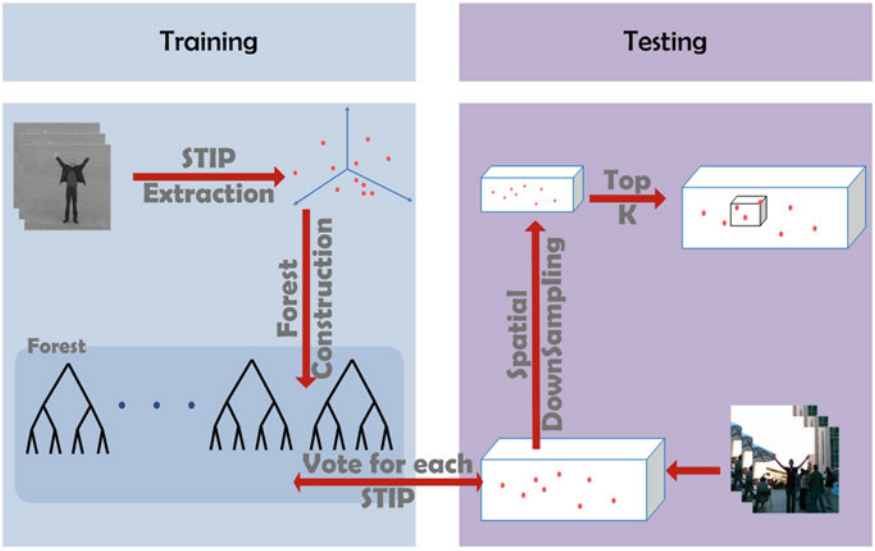

Fig. 2.1 The overview of our random forest-based video subvolume search

2.2 Multiclass Action Recognition

2.2.1 Mutual Information-Based Classification

We represent an action as a collection of spatio-temporal interest points (STIP) [7], where $d \in \mathbb{R}^N$ denotes an N-dimensional feature vector describing a STIP. The reasons to use STIP feature is its superior performance for human action recognition and detection. Besides, the descriptor for STIP, i.e., HOG and HOF, is suitable for our random forest-based framework. A comparison of different detectors and descriptors can be seen in [13]. Denote the class label set as $\mathcal{C} = \{1, 2, \ldots, C\}$.

In order to recognize different action classes, we evaluate the pointwise mutual information between a testing video clip $\mathcal{Q} = \{d_q\}$ and one action class $c \in \mathcal{C}$:

$$\begin{aligned} \mathrm{MI}(\mathbf{C} = c, \mathcal{Q}) &= \log \frac{P(\mathcal{Q}|\mathbf{C}=c)}{P(\mathcal{Q})} \\ &= \log \frac{\prod_{d_q \in \mathcal{Q}} P(d_q|\mathbf{C}=c)}{\prod_{d_q \in \mathcal{Q}} P(d_q)} \\ &= \sum_{d_q \in \mathcal{Q}} \log \frac{P(d_q|\mathbf{C}=c)}{P(d_q)}, \end{aligned} \tag{2.1}$$

where d_q refers to the STIP point in $\mathcal{Q}$ and we assume d_q is independent from each other. Each $s^c(d_q) = \log\frac{P(d_q|\mathbf{C}=c)}{P(d_q)}$ is the point-wise mutual information between a STIP point d_q and a specific class c.

In the previous work [16], $s^c(d_q)$ is computed as follows:

$$s^c(d_q) = \mathrm{MI}(\mathbf{C} = c, d_q) = \log\frac{C}{1 + \frac{P(d_q|\mathbf{C}\neq c)}{P(d_q|\mathbf{C}=c)}(C-1)}, \tag{2.2}$$

where C is the number of classes. The likelihood ratio in Eq. 2.2 is calculated as:

$$\frac{P(d_q|\mathbf{C}\neq c)}{P(d_q|\mathbf{C}=c)} \approx \lambda^c \exp^{-\frac{1}{\sigma^2}(||d_q - d_{NN}^{c-}||^2 - ||d_q - d_{NN}^{c+}||^2)}, \tag{2.3}$$

where d_{NN}^{c+} and d_{NN}^{c-} are the nearest neighbors of d_q in the positive class and negative class, respectively, and λ^c is the ratio of the number of positive STIP to the number of negative STIP in the training dataset.

Despite its good performance, Eq. 2.3 has two limitations:

- In order to calculate the likelihood ratio in Eq. 2.3, we need to search the nearest neighbors d_{NN}^{c+} and d_{NN}^{c-}. Although locality sensitive hash (LSH) has been employed for fast nearest neighbor search, it is still time-consuming for large high dimensional dataset.
- Only two STIPs are used to approximate the likelihood ratio in Eq. 2.3, which is not accurate.

To address the two problems, we reformulate the voting score $s^c(d_q)$ in Eq. 2.2 as:

$$\begin{aligned} s^c(d_q) &= \mathrm{MI}(\mathbf{C} = c, d_q) = \log\frac{P(d_q|\mathbf{C}=c)}{P(d_q)} \\ &= \log\frac{P(\mathbf{C}=c, d_q)}{P(\mathbf{C}=c)P(d_q)} \\ &= \log\frac{P(\mathbf{C}=c|d_q)}{P(\mathbf{C}=c)} \\ &= \log P(\mathbf{C} = c|d_q) - \log P(\mathbf{C} = c). \end{aligned} \tag{2.4}$$

As $P(\mathbf{C} = c)$ is a constant prior, the problem boils down to computing the posterior $P(\mathbf{C} = c|d_q)$. To enable an efficient computation, we approximate this probability with a random forest.

2.2.2 Random Forest-Based Voting

Random forest was first proposed to solve the classification problem [2]. Later, it has been extended to handle regression problems and is used for many multimedia applications, like [1, 5, 8–11, 14]. In this section, random forest is used to estimate the posterior probability $P(\mathbf{C} = c|d_q)$.

To build the forest from a training dataset, we use a method motivated by [5]. However, compared with [5], which treats a random forest as a classifier and votes for the hypothesis given a feature point, our random forest is used to estimate the posterior distribution of each STIP point.

Two kinds of descriptors for STIP: Histogram of Gradient (HoG) and Histogram of Flow (HoF), are used to build the random forest. In the following, we first describe how to build a single decision tree, then the forest is constructed by M independent trees. Assume we have N STIP points in the training set, defined as $\{(x_i, y_i), i = 1, 2, \ldots, N\}$, where $x_i = (x_i^1, x_i^2)$; $x_i^1 \in R^{72}$ and $x_i^2 \in R^{90}$ refer to the HOG feature and HOF feature, respectively; $y_i \in \mathcal{C}$ is the label of the STIP (if we want to detect actions from category $\mathbf{C} = c$, we consider STIPs with $y_i = c$ as positive examples and other STIPs as negative examples). In order to build a tree and split the training set, 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 $x_i^{\tau=2}$ refers to HOF feature). Then two more random integer numbers e_1 and e_2 will be generated, indicating the dimension indices of either HOG or HOF feature. After that, a "feature difference" can be evaluated with $D_i = x_i^\tau(e_1) - x_i^\tau(e_2), i = 1, 2, \ldots, N$. For each x_i, we assign it to the left child node if $x_i^\tau(e_1) - x_i^\tau(e_2) \geq \theta$ or right child node if $x_i^\tau(e_1) - x_i^\tau(e_2) < \theta$.

The threshold θ is selected by minimizing the binary classification error:

$$\theta^* = \text{argmin}_\theta(\min\{\mathcal{E}(c)^L + \mathcal{E}(\bar{c})^R, \mathcal{E}(c)^R + \mathcal{E}(\bar{c})^L\}), \tag{2.5}$$

where:

$$\begin{aligned}
\mathcal{E}(c)^L &= \sum_{i=1}^{N} I(y_i \neq c) I(x_i^\tau(e_1) - x_i^\tau(e_2) \geq \theta) \\
\mathcal{E}(c)^R &= \sum_{i=1}^{N} I(y_i \neq c) I(x_i^\tau(e_1) - x_i^\tau(e_2) < \theta) \\
\mathcal{E}(\bar{c})^L &= \sum_{i=1}^{N} I(y_i = c) I(x_i^\tau(e_1) - x_i^\tau(e_2) \geq \theta) \\
\mathcal{E}(\bar{c})^R &= \sum_{i=1}^{N} I(y_i = c) I(x_i^\tau(e_1) - x_i^\tau(e_2) < \theta).
\end{aligned} \tag{2.6}$$

In Eq. 2.6, $I(x)$ is a indicator function, that is $I(x) = 1$ if $x = 1$ and 0 otherwise. And c is the action type we want to detect. The first two terms refer to the misclassification errors of the left node and right node, respectively, when the labels of the nodes are both c. The last two terms refer to the misclassification errors of the left node and right node, respectively, when the labels of the nodes are not c.

The above three parameters (τ, e_1 and e_2) can be integrated into a single hypothesis. For example, we can generate a hypothesis to partition the dataset using the following three steps:

- Generate $\tau \in \{1, 2\}$ to indicate the feature type to use
- Generate the dimension index e_1 and e_2 and compute the feature difference $D_i = x_i^\tau(e_1) - x_i^\tau(e_2), i = 1, 2, \ldots, N$

- Split the dataset into two parts based on a threshold on feature difference and obtain a misclassification error

We generate γ hypotheses independently ($\gamma = 200$ in our experiments) and select the one with the smallest misclassification error. After this, one node will be built and the training set 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 any of the two conditions below is satisfied: (1) the depth of the tree reaches the maximum number or (2) the number of points in the node is smaller than a predefined threshold.

Now, we discuss how to compute $P(\mathbf{C} = c|d_q)$ with a random forest. Suppose we have M trees in a forest and the STIP d_q will fall in one of the leaves in a tree. Assume that for a tree T_i, the STIP point d_q falls in a leaf with N_i^+ positive samples and N_i^- negative samples. The posterior distribution of d_q can be approximated by the average density of the M nodes in M different trees:

$$P(\mathbf{C} = c|d_q) \approx \frac{1}{M} \sum_{i=1}^{M} \frac{N_i^+}{N_i^+ + N_i^-}. \tag{2.7}$$

Then Eq. 2.4 can be replaced with:

$$\begin{aligned} S^c(d_q) &= \log P(\mathbf{C} = c|d_q) - \log P(\mathbf{C} = c) \\ &= \log \tfrac{1}{M} \sum_{i=1}^{M} \tfrac{N_i^+}{N_i^+ + N_i^-} - \log P(\mathbf{C} = c). \end{aligned} \tag{2.8}$$

In the training dataset, the numbers of STIP points are different for different action classes. Therefore, it is inaccurate to compute the prior probability $P(\mathbf{C} = c)$ directly from the distribution of training dataset. In our experiments, we introduce the parameter $A = -\log P(\mathbf{C} = c)$ and optimize it in the experiments.

The benefits of using the random forest are numerous. First, each tree in the forest is independent to other trees when evaluating $P(\mathbf{C} = c|d_q)$ in Eq. 2.7. The average of them thus reduces the variance of the estimation. Second, random forest is fast to evaluate during the testing stage. The runtime cost for each STIP only depends on the depth of each tree and the number of trees. It is not affected by the number of points in the training data. Hence, it is much faster than LSH-based nearest neighbor search. In the experiment section, we will show that random forest-based voting approach is over 4,000 times faster than the LSH-based approach. Another advantage of random forest compared with LSH is that, when constructing the trees, the label information of x_i can be integrated. Thus, the trees follow the data distribution of the training data. This improves the generalization ability. Finally, the construction of random forest is flexible. Besides the label information, it is easy to combine other types of feature descriptors and spatial information of STIPs.

According to the literature, [10, 14, 15] also employ tree structures for action recognition. We first consider the differences between [14] and our work. The feature [14] employs is densely sampled while we use the sparse STIP features. Second, [14]

votes for the center of the action while our random forest weighs each STIP point so that the non-trivial scale estimation can be partially solved with branch and bound search. Third, the votes in [14] are estimated from the frequency view so that it would generate positive votes even for the background. On the contrary, our votes employs the mutual information-based measure (Eq. 2.4), which is more discriminative thanks to the introduction of negative votes. The trees in [10] are used for indexing and searching nearest neighbor while trees in [15] serves as a codebook. Since we employ random forest to weigh each STIP points, the motivations and implementations are different from [10, 15]. Besides, our work can deal with not only action classification but also action detection, while [10, 15] are only applicable to action recognition.

After obtaining the individual voting score of each STIP, the spatio-temporal location and scale of the target action will be determined by the branch-and-bound search as described in next section.

2.3 Action Detection and Localization

The purpose of action detection is to find a subvolume V with the maximum similarity to the pre-defined action type. Following [16], with each STIP being associated with an individual score $s^c(d)$, our goal is to find the video subvolume with the maximum score:

$$V^* = \text{argmax}_{V \subset \mathcal{V}} f(V), \tag{2.9}$$

where, $V = [T, B] \times [L, R] \times [S, E]$ is a video subvolume, where L, R, T, B, S and E are the left, right, top, bottom, start and end positions of V; $f(V) = \sum_{d \in V} s^c(d)$ and $\mathcal{V}$ is the whole video space. A subvolume V is said to be *maximal* if there does not exist any other subvolume V' such that $f(V') > f(V)$ and $V' \cap V \neq \emptyset$. The action detection problem is to find all the maximal subvolumes whose scores are above a certain threshold.

A spatio-temporal branch-and-bound algorithm was proposed in [16] to solve the single subvolume search problem. Instead of performing a branch-and-bound search directly in the 6-dimensional parameter space Λ, the method performs a branch-and-bound search in the four-dimensional spatial parameter space. In other words, it finds the spatial window W^* that maximizes the following function:

$$F(W) = \max_{T \subseteq \mathbb{T}} f(W \times T), \tag{2.10}$$

where $W = [T, B] \times [L, R]$ is the spatial window; $T = [S, E]$ is the temporal segment, and $\mathbb{T} = [0, t - 1]$.

One advantage of separating the parameter space is that the worst case complexity is reduced from $O(m^2n^2t^2)$ to $O(m^2n^2t)$. The complexity is linear in t, which is usually the largest of the three dimensions. For this reason, it is efficient in processing long videos, but when the spatial resolution of the video increases, the

complexity goes up quickly. The method in [16] was tested on videos with low resolution (160 × 120). In this section, we are interested in higher resolution videos (320 × 240 or higher). We found that for videos taken under challenging lighting conditions with crowded background such as those in the publicly available MSR Action dataset II,[1] the action detection rates on 320 × 240 resolution videos are much better than those on 160 × 120. Unfortunately, the subvolume search for 320 × 240 videos is much slower. For example, [16] takes 20 h to search the MSR Action dataset II which consists of 54 video sequences of 1 min long each with 320 × 240 resolution.

Moreover, in [16], the multi-instance detection problem was converted to a series of single subvolume search problem. They first find the optimal subvolume V_1 such that $f(V_1) = max_V f(V)$. After that, it sets the scores of all the points in V_1 to 0, and finds the optimal subvolume V_2, and so on. To further speed up the search process during the branch-and-bound iterations, a heuristic was used in [18]. If a candidate window $\mathbb{W}$ with a score larger than the detection threshold is found, the subsequent searches are limited to the subwindows contained in $\mathbb{W}$. It guarantees that it will find a valid detection, but the detected subvolume is not guaranteed to be optimal.

In the next two subsections, we present two techniques to speed up the subvolume search algorithm. The combination of the two techniques allows us to perform subvolume search on 320 × 240 videos in real time.

2.3.1 Spatial Down-Sampling

To handle high-resolution videos, the technique is to spatially down-sample the video space by a factor s before the branch-and-bound search. Note that the interest point detection, descriptor extraction, and the scores are all done in the original video sequence.

For a video volume $\mathcal{V}$ of size $m \times n \times t$, the size of the down-sampled volume $\mathcal{V}^s$ with scale factor s is $\frac{m}{s} \times \frac{n}{s} \times t$. For any point $(i, j, k) \in \mathcal{V}^s$ where $i \in [0, \frac{m}{s} - 1]$, $j \in [\frac{n}{s} - 1]$, and $k \in [0, t - 1]$, its score is defined as the sum of the scores of the $s \times s$ points in $\mathcal{V}$, that is, $f^s(i, j, k)$ is defined as

$$f^s(i, j, k) = \sum_{x=0}^{s-1} \sum_{y=0}^{s-1} f(s * i + x, s * j + y, k). \tag{2.11}$$

Given any subvolume $V^s = [L, R] \times [T, B] \times [S, E] \subset \mathcal{V}^s$, where L, R, T, B, S and E are the left, right, top, bottom, start and end positions of V^s, respectively, denote $\xi(V^s)$ to be its corresponding subvolume in original video $\mathcal{V}$, that is,

$$\xi(V^s) = [s * L, s * (R + 1) - 1] \times [s * T, s * (B + 1) - 1] \times [S, E]. \tag{2.12}$$

[1] The MSR action dataset II is available at http://research.microsoft.com/en-us/um/people/zliu/ActionRecoRsrc/default.htm.

As they are the same subvolume, it is easy to see that

$$f^s(V^s) = f(\xi(V^s)). \tag{2.13}$$

A subvolume $V = [X_1, X_2] \times [Y_1, Y_2] \times [T_1, T_2] \subset \mathcal{V}$ is called an *s-aligned* subvolume if X_1 and Y_1 are multiples of s and the width $X_2 - X_1 + 1$ and height $Y_2 - Y_1 + 1$ are also multiples of s. Equation 2.12 provides a one-to-one mapping between the volumes in $\mathcal{V}^s$ and the s-aligned subvolumes in $\mathcal{V}$.

Instead of searching the original video space, we can search the down-sampled video space $\mathcal{V}^s$ of a much smaller size $\frac{m}{s} \times \frac{n}{s} \times t$. However, as the down-sampling process also introduces the approximation errors, it affects the search results. In general, for any $V^s \subset \mathcal{V}^s$, there exists a $V = \xi(V^s) \subset \mathcal{V}$. It thus shows that the maximum subvolume found in the down-sampled space is at most as good as the one found in the original space:

$$\max_{V^s \subset \mathcal{V}^s} f^s(V^s) \leq \max_{V \subset \mathcal{V}} f(V). \tag{2.14}$$

We illustrate a concrete example in Fig. 2.2. For simplicity, in Fig. 2.2, we choose the down-sampling factor $s = 2$ and discuss the problem in the $2D$ space (only one frame is considered). The left figure shows the original video space and its down-sampled version is in the right figure. Each pixel is associated with a voting score. The orange rectangle highlights the optimal solution in the original video space, namely the bounding box of the highest total sum. After the down-sampling, the gray rectangle is the detection result in the down-sampled video. By mapping it back to the original space, we obtain an approximate solution highlighted by the red rectangle. It overlaps with the optimal solution in the original space, but the total sum is slightly less. To further quantify the approximation error, we derive the upper bound of the error caused by the down-sampling, as explained in Theorem 2.1.

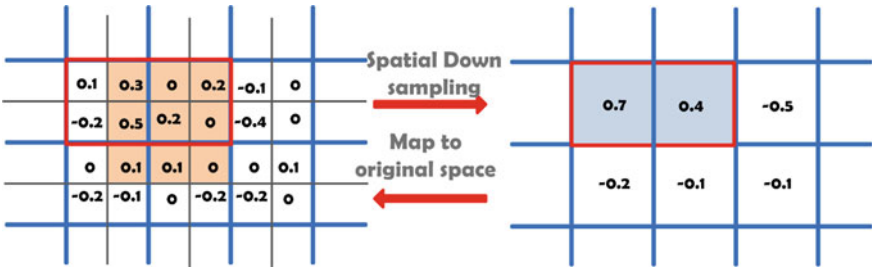

Fig. 2.2 Approximation of the spatial down-sampling. *Left* figure shows the score image in the original resolution and *right* figure shows the down-sampled score image. Every four small pixels in a cell from the original resolution sum up to one score in the low resolution, for example, the value in the *top-left* pixel from the *right* figure $0.5 = 0.1 + 0.3 - 0.4 + 0.5$. We notice that the optimal solution found in the down-sampled video space is worse than that in the original space ($f^s(\tilde{V}^*) = 1.1 < f(V^*) = 1.4$)

Theorem 2.1 *Bound of the approximation error. Let V^* denote the optimal subvolume in $\mathcal{V}$, that is, $f(V^*) = max_{V \subset \mathcal{V}} f(V)$. Assume $V^* = [x_1, x_1 + w - 1] \times [y_1, y_1 + h - 1] \times [t_1, t_2]$ where w and h are the width and height of V^*, respectively and further assume the total score of a subvolume is on average proportional to its size. Then, there exists an s-aligned subvolume $\tilde{V}$ satisfying*:

$$f(\tilde{V}) \geq (1 - \frac{s * h + s * w + s^2}{wh}) f(V^*). \tag{2.15}$$

Proof Let V^* denote the optimal subvolume in $\mathcal{V}$, that is, $f(V^*) = \max_{V \subset \mathcal{V}} f(V)$. Assume $V^* = [x_1, x_1 + w - 1] \times [y_1, y_1 + h - 1] \times [t_1, t_2]$ where w and h are the width and height of V^*, respectively. Let $|V|$ denote the number of voxels in V. It can be shown that there exists an s-aligned subvolume $\tilde{V} = [\tilde{x}_1, \tilde{x}_1 + \tilde{w} - 1] \times [\tilde{y}_1, \tilde{y}_1 + \tilde{h} - 1] \times [t_1, t_2]$ such that

$$|(V^* \setminus \tilde{V}) \cup (\tilde{V} \setminus V^*)| \leq (s * h + s * w + s^2)(t_2 - t_1). \tag{2.16}$$

Therefore,

$$\frac{|(V^* \setminus \tilde{V}) \cup (\tilde{V} \setminus V^*)|}{|V^*|} \leq \frac{s * h + s * w + s^2}{wh}. \tag{2.17}$$

If we assume the total score of a subvolume is on average proportional to its size, then

$$\frac{f((V^* \setminus \tilde{V}) \cup (\tilde{V} \setminus V^*))}{f(V^*)} \leq \frac{s * h + s * w + s^2}{wh}. \tag{2.18}$$

Therefore

$$\frac{f(V^*) - f(\tilde{V})}{f(V^*)} \leq \frac{s * h + s * w + s^2}{wh}. \tag{2.19}$$

After a re-arrangement of the items, we have:

$$f(\tilde{V}) \geq (1 - \frac{s * h + s * w + s^2}{wh}) f(V^*). \tag{2.20}$$

Let $\tilde{V}^* = \text{argmax}_{V \in \mathcal{V}^s} f^s(V)$ denote the optimal subvolume in $\mathcal{V}^s$. Based on Eq. 2.15, we have

$$f^s(\tilde{V}^*) \geq (1 - \frac{s * h + s * w + s^2}{wh}) f(V^*). \tag{2.21}$$

As an example, suppose the spatial dimension of V is 320×240, and the scale factor $s = 8$. The spatial dimension of the down-sampled volume is 40×30. If we assume the window size of the optimal subvolume V^* is 64×64, then the average relative error is at most

$$\frac{s*h+s*w+s^2}{wh} = \frac{8*64+8*64+8^2}{64^2} \approx 25\,\%. \tag{2.22}$$

We have run numerical experiments to measure the relative error of the optimal solutions in the down-sampled volumes. We used 30 video sequences of resolution 320×240. There are three action types. For each video sequence and each action type, we obtain a 3D volume of scores as defined in Eq. 2.8. We choose $s = 8$, and down-sample each 3D volume to spatial resolution of 40×30. There are 113 actions in total. For each action, we compute its corresponding down-sampled subvolume, and evaluate the relative error which is the score difference divided by the original action score. The mean is 23 % and the standard deviation is 26 %. We can see that the numerical experiments are consistent with the theoretical analysis.

2.3.2 Top-K Search Algorithm

The multi-instance search algorithm in [16] repeatedly applies the single-instance algorithm many times until some stop criteria is met. In practice, there are typically two different stop conditions that can be used. The first is to stop after k iterations where k is a user-specified integer. The second is to stop when the detection score is smaller than a user-specified detection threshold λ. In either case, suppose the number of detected instances is k, then the worst case complexity of the algorithm is $O(kn^2m^2t)$.

We notice that in 1D case, Brodal and Jorgensen [3] developed an algorithm that finds the Top-K subarrays in $O(n+k)$ time. This is much more efficient than repeatedly applying the single-instance algorithm k times which has the complexity $O(kn)$. In 3D case, we would also like to have an algorithm that is more efficient than simply applying the single-instance algorithm k times. We consider two different variants corresponding to the two stop criteria. The first, called λ search, can be applied when we are interested in finding all the subvolumes above a user-specified threshold λ. The second, called Top-K search, can be applied when we are interested in finding the Top-K subvolumes.

2.3.2.1 λ Search

In this section, we describe an algorithm that finds all of the subvolumes with scores larger than a user-specified threshold λ. The pseudo-code of the algorithm is shown in Algorithm 1. Following the notation in [16], we use $\mathbb{W}$ to denote a collection of spatial windows, which is defined by 4 intervals that specify the parameter ranges for the left, right, top, and bottom positions, respectively. Given any set of windows $\mathbb{W}$, we use $\hat{F}(\mathbb{W})$ to denote its upper bound which is estimated in the same way as in [16, 17]. We use $W_{\max}$ to denote the largest window among all the windows in $\mathbb{W}$. Initially, $\mathbb{W}^*$ is equal to the set of all the possible windows on the image and $F*$ is the corresponding upper bound, as in Line 5 of Algorithm 1. From Line 6–19, we split and store the results if the top state $\mathbb{W}$ is over a threshold λ and iterate this

process. From Line 20–22, we have a subvolume (V^*) detected. The whole process iterates until the score for the detected subvolume is below the threshold.

Algorithm 1 λ search.

1: Initialize P as empty priority queue
2: set $\mathbb{W} = [T, B, L, R] = [0, m] \times [0, m] \times [0, n] \times [0, n]$
3: push($\mathbb{W}$, $\hat{F}(\mathbb{W})$) into P
4: **repeat**
5: Initialize current best solution F^*, W^*
6: **repeat**
7: retrieve top state $\mathbb{W}$ from P based on $\hat{F}(\mathbb{W})$
8: **if** $\hat{F}(\mathbb{W}) > \lambda$ **then**
9: split $\mathbb{W}$ into $\mathbb{W}^1 \cup \mathbb{W}^2$
10: **if** $\hat{F}(\mathbb{W}^1) > \lambda$ **then**
11: push ($\mathbb{W}^1$, $\hat{F}(\mathbb{W}^1)$) into P
12: update current best solution $\{W^*, F^*\}$
13: **end if**
14: **if** $\hat{F}(\mathbb{W}^2) > \lambda$ **then**
15: push ($\mathbb{W}^2$, $\hat{F}(\mathbb{W}^2)$) into P
16: update current best solution $\{W^*, F^*\}$
17: **end if**
18: **end if**
19: **until** $\hat{F}(\mathbb{W}) \leq F^*$
20: $T^* = argmax_{T \in [0,t]} f(W^*, T)$;
21: add $V^* = [W^*, T^*]$ to the list of detected subvolumes.
22: for each point $(i, j, k) \in V^*$, set $f(i, j, k) = 0$.
23: **until** $\hat{F}(\mathbb{W}) \leq \lambda$

In terms of the worst case complexity, the number of branches of this algorithm is no larger than $O(n^2m^2)$ since the algorithm does not re-start the priority queue P. Each time it branches, the algorithm has to compute the upper bound whose complexity is $O(t)$. Therefore the worst complexity involved in branch and bound is the same as [16]: $O(n^2m^2t)$. In addition, each time when it detects a subvolume, the algorithm has to update the scores of the video volume which has complexity $O(nmt)$. If there are k detected subvolumes, the complexity for updating the scores is $O(kmnt)$. Overall, the worst case complexity of this algorithm is $O(n^2m^2t) + O(kmnt)$. When k is large, this is much better than $O(kn^2m^2t)$.

2.3.2.2 Top-K Search

In this section, we describe how to modify Algorithm 2 for the case when we are interested in finding the Top-K actions, and we assume we do not know the threshold λ.

The pseudo-code of the algorithm is shown in Algorithm 2. The algorithm is similar to Algorithm 2. In Line 6, $(\{W_i^*, F_i^*\})_{i=c...k}$ are set as all the possible windows on the image and its upper bound score, respectively. From Line 6–20, we split and

store the results if the top state $\mathbb{W}$ is over the Kth top score and iterate this process. From Line 21–24, we have a subvolume (V^*) detected. The whole process iterates until K subvolumes are detected. There are four major differences. First, instead of maintaining a single current best solution, it maintains k-best current solutions. Second, it replaces the criteria $\hat{F}(\mathbb{W}) > \lambda$ with $\hat{F}(\mathbb{W}) > F_k^*$ to determine whether we need to insert $\mathbb{W}^1$ or $\mathbb{W}^2$ into the queue P. Third, it replaces the inner-loop stop criteria $\hat{F}(\mathbb{W}) \leq F^*$ with $\hat{F}(\mathbb{W}) \leq F_c^*$. Finally, the outer-loop stop criteria $\hat{F}(\mathbb{W}) \leq \lambda$ is replaced with $c > k$. In this algorithm, the number of outer loops is k. So the worst case complexity is also $O(n^2m^2t) + O(kmnt)$.

Algorithm 2 Top-K Search.

1: Initialize P as empty priority queue
2: set $\mathbb{W} = [T, B, L, R] = [0, m] \times [0, m] \times [0, n] \times [0, n]$
3: push($\mathbb{W}$, $\hat{F}(\mathbb{W})$) into P
4: c=1
5: **repeat**
6: Initialize $(\{W_i^*, F_i^*\})_{i=c...k}$ where $F_k^* \leq ... \leq F_c^*$
7: **repeat**
8: retrieve top state $\mathbb{W}$ from P based on $\hat{F}(\mathbb{W})$
9: **if** $\hat{F}(\mathbb{W}) > F_k^*$ **then**
10: split $\mathbb{W}$ into $\mathbb{W}^1 \cup \mathbb{W}^2$
11: **if** $\hat{F}(\mathbb{W}^1) > F_k^*$ **then**
12: push $(\mathbb{W}^1, \hat{F}(\mathbb{W}^1))$ into P
13: update $(\{W_i^*, F_i^*\})_{i=c...k}$
14: **end if**
15: **if** $\hat{F}(\mathbb{W}^2) > F_k^*$ **then**
16: push $(\mathbb{W}^2, \hat{F}(\mathbb{W}^2))$ into P
17: update $(\{W_i^*, F_i^*\})_{i=c...k}$
18: **end if**
19: **end if**
20: **until** $\hat{F}(\mathbb{W}) \leq F_c^*$
21: $T^* = argmax_{T \in [0,t]} f(W^*, T)$;
22: output $V_c^* = [W^*, T^*]$ as the c-th detected subvolume
23: for each point $(i, j, k) \in V_c^*$, set $f(i, j, k) = 0$.
24: c = c+1
25: **until** $c > k$

2.4 Experiments

2.4.1 Action Classification

To evaluate our proposed random forest-based approach for multiclass action classification, we test on the benchmark KTH dataset. The experiment setup is the same as [7, 16] where clips from 16 persons are used for training and the other 9 persons

Table 2.1 Confusion matrix for KTH action dataset

	Clap	Wave	Box	Run	Jog	Walk
Clap	137	1	6	0	0	0
Wave	7	137	0	0	0	0
Box	0	0	144	0	0	0
Run	0	0	0	95	47	2
Jog	0	0	0	4	136	4
Walk	0	0	0	0	0	144

The total accuracy is 91.8 %

Table 2.2 Comparison of different reported results on KTH dataset

Method	Mean accuracy (%)
Our method	91.8
Yuan et al.'s [16]	93.3
Reddy et al.'s [10]	90.3
Laptev et al.'s [6]	91.8

are used for testing. The confusion matrix is listed in Table 2.1. We also compare our results with the state-of-the-art results in Table 2.2. With the same input features, our method performs as well as the method using support vector machine for classification [6]. Although our performance is slightly worse than the nearest neighbor-based classification in [16], as will be shown later, our approach is significantly faster as it avoids the nearest neighbor search.

2.4.2 Action Detection

To evaluate our multiclass action detection and localization, we perform cross-dataset training and testing. We first build a random forest using the KTH dataset (with the 16 persons in the training part) and then test on a challenging dataset (MSRII) 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. The videos contain three different types of actions: handwaving, handclapping, and boxing. Some videos contain different people performing different actions simultaneously. There are also instances where a person performs two different actions consecutively.

For all of our experiments we have fixed $K = 3$, $\lambda = 3.0$. Moreover, unless explicitly mentioned we down-sample the score volume to 40×30 pixels.

Figure 2.3 compares the precision-recall for the following methods (the original videos are of high resolution 320×240):

(i) ASTBB (Accelerated Spatio-Temporal Branch-and-Bound search) of [18] in low resolution score volume (frame size 40×30),
(ii) ASTBB of [18] in 320×240 videos,

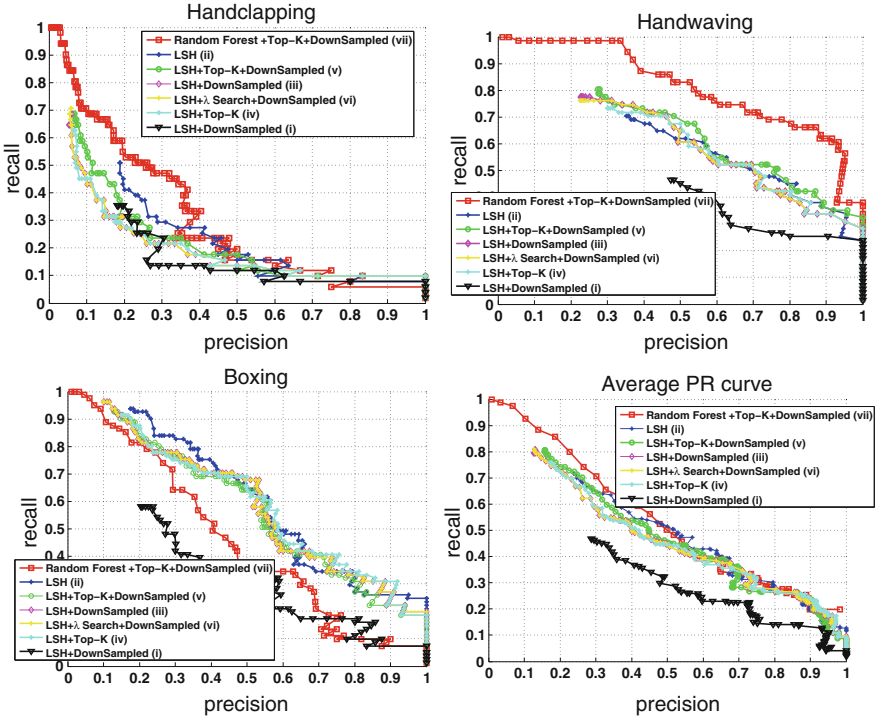

Fig. 2.3 Precision-recall curves for action detections with different methods

(iii) multiround branch-and-bound search of [16] in low-resolution score volume (frame size 40×30),
(iv) Top-K search at original size 320×240,
(v) Top-K search at down-sampled score volume (size 40×30),
(vi) λ search at down-sampled score volume (size 40×30),
(vii) random forest-based weighting followed by Top-K search at down-sampled score volume (size 40×30).

Except for (vii), which uses our random forest-based voting score, the other methods apply the LSH-based nearest neighbor voting score as in [16]. The parameter $A = -\log P(C = c)$ in Eq. 2.8 for method (vii) is set to 2.1, 1.7 and 0.9 for handclapping, handwaving and boxing respectively. Also, we use the walking actions from KTH as the negative dataset when constructing forests. For the purpose of generating precision-recall curves, we modified the outer-loop stop criteria (Line 25, Algorithm 2) to repeat until $\hat{F}(\mathbb{W}) \leq \lambda$ where λ is a small threshold. In this way, it outputs more than K subvolumes which is necessary for plotting the precision-recall curve. Some sample detection results obtained by our approach (vii) are shown in Fig. 2.4. To demonstrate the capability of handling non-stationary actions, we show a walking detection result at the bottom row. The detection is done by using KTH

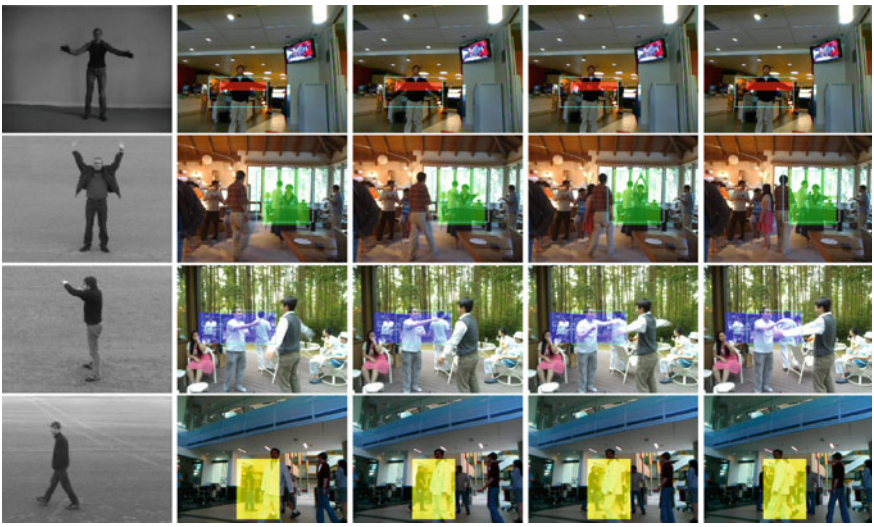

Fig. 2.4 Detection results (Random Forest+Top-K) of handclapping (*1st row*), handwaving (*2nd row*), boxing (*3rd row*) and walking (*4th row*) are listed in 2-5 columns with *red*, *green*, *blue* and *yellow* colors to show the bounding boxes, respectively. The *cyan dash* regions are the ground truths. The first column are sample images from training set

walking as the positive training data while the KTH handwaving, handclapping, and boxing are used as the negative training data.

The measurement of precision and recall is the same as what is described in [16]. For the computation of the precision we consider a true detection if : $\frac{\text{Volume}(V^* \cap G)}{\text{Volume}(G)} > \frac{1}{8}$ where G is the annotated ground truth subvolume, and V^* is the detected subvolume. On the other side, for the computation of the recall we consider a hit if: $\frac{\text{Volume}(V^* \cap G)}{\text{Volume}(V^*)} > \frac{1}{8}$.

We first compare the results based on LSH voting approaches. Figure 2.3 lists the Precision-Recall curves for the three different action classes, respectively. The average PR curve is computed by averaging precision and recall results among the three action types while adjusting a threshold. This can give a general idea of the overall performance for different algorithms. From the precision-recall curves, we can see that although the accelerated search of [18] provides excellent results in high resolution videos, its performance on down-sampled low resolution videos is poor compared with other search schemes. Moreover, all the methods applied to the high resolution videos provide similar performance. In particular, the methods of Top-K search with branch-and-bound search at down-sampled size (v) and λ search with branch-and-bound search at down-sampled size (vi) are among the best ones. These results justify our proposed λ search and Top-K search algorithms. Although the branch-and-bound is performed in the down-sampled size videos, it still provides good performance. However, the search speed is much faster. To compare the performance of action detection between LSH and random forest, (v) and (vii)

are two search schemes with the same environment but different voting approaches. Random forest (vii) is superior to LSH (v) in handwaving but poorer in boxing. Since the boxing action is highly biased in KTH dataset (much more boxing actions are performed from right to left), it reduces the discriminative ability of the trees. For LSH, however, because it searches only one nearest positive and negative in the neighborhood, the effect of such bias can almost be ignored.

2.4.3 Computational Cost

The feature extraction step is performed with publicly available code in [7]. Although their code may not be very fast, there are faster implementations available. Therefore, the computation time for feature extraction is not considered in this section. We suppose all the STIP points are already extracted and stored in the memory. Then, the computational time of our algorithms is dominated by two operations, computing the score for each STIP and branch-and-bound search. For the first part, LSH takes on average 18.667 ms per STIP point while random forest only takes 0.0042 ms. To deal with a video clip with 10,000 STIPs, it will take around 186.67 s for LSH but only 42 ms for random forest. That is, random forest-based approach is 4,000 times faster than LSH-based approach.

Table 2.4 shows the time consumed for the search part. All of the algorithms are implemented using C++, performed on a single PC of dual-core and 4G main memory: (a) accelerated λ search of [18] in low resolution videos (frame size 40 $\times$ 30), (b) accelerated λ search of [18] in high resolution videos, (c) multiround branch-and-bound search of [16] in low-resolution videos (frame size 40 $\times$ 30), (d) λ search, with branch-and-bound search at down-sampled size 40 $\times$ 30, (e) Top-K search, with branch-and-bound search at down-sampled size 80 $\times$ 60, (f) Top-K search, with branch-and-bound search at down-sampled size 40 $\times$ 30.

Table 2.4 shows that although the method of [18] works well for low resolution videos, the search speed becomes much slower for high resolution videos. Moreover as shown in Fig. 2.3, when performing on the down-sampled score volumes, the heuristic method of [18] (curve (i)) is a lot worse than the other methods. This is an indication that it is not a good idea to perform heuristic search on down-sampled score volumes. In comparison, λ search provides much better search quality. Among all the search schemes, the fastest method is the Top-K search with branch-and-bound

Table 2.3 Time consumed for voting one STIP and one video sequence (for example, 10,000 STIP points)

Method	Voting time (ms)	One sequence (s)
LSH	18.667 $\pm$ 8.4105	186.67
Random forest	0.0042 $\pm$ 0.0032	0.042

Only CPU time is considered

Table 2.4 Time consumed for each method to search actions in the 54 videos

Method	Running time
Low resolution (40 × 30) [18]	40 min
High resolution (320 × 240) [18]	20 h
Down-sampled B&B (40 × 30)	10 h
λ search + down-sampled B&B (40 × 30)	1 h 20 min
Top-K + down-sampled B&B (80 × 60)	6 h
Top-K + down-sampled B&B (40 × 30)	26 min

Table 2.5 Comparison of total time cost for action detection

Method	Voting time (mins)	Search time (min)	Total time (min)
LSH+B&B [18]	271	1,200	1, 471
LSH+Top-K (our algorithm)	271	26	297
Random forest+Top-K (our algorithm)	0.62	26	26.62

Only CPU time is considered

at down-sampled score volume of 40 × 30. It takes only 26 min to process the 54 sequences whose total length is about one hour in total.

Finally, we compare LSH and random forest in terms of total computation time in Table 2.5, including the runtime cost for computing scores and the runtime cost for top-K search. For the previous method [18], it takes at least 1,471 min to search all the actions for 54 videos in MSRII. In contrast, the total computation time of our proposed algorithm is 26.62 min.

2.5 Summary of this Chapter

We have developed a novel system for human action recognition and spatio-temporal localization in video sequences. The system improves upon the state-of-arts on two aspects. First, we proposed a random forest-based voting technique to compute the scores of the interest points, which achieves multiple orders-of-magnitude speed-up compared to the nearest neighbor-based scoring scheme. Second, we proposed a top-k search technique which detects multiple action instances simultaneously with a single round of branch-and-bound search. To reduce the computational complexity of searching higher resolution videos, we performed subvolume search on the down-sampled score volumes. We have presented experimental results on challenging videos with crowded background. The results showed that our proposed system is robust to dynamic and cluttered background and is able to perform efficient action detection on high resolution videos. In next chapter, we will further improve the computational speed for action localization with a coarse-to-fine branch and bound search strategy, and extend the idea for the human action search based on one query sample.

References

1. A. Bosch, A. Zisserman, X. Munoz, Image classification using random forests and ferns, in *Proceedings of the IEEE International Conference on Computer Vision* (2007)
2. L. Breiman, Random forests. Mach. Learn. **45**, 5–32 (2001)
3. G. Brodal, A. Jørgensen, A linear time algorithm for the k maximal sums problem. Math. Found. Comput. Sci. **4707**, 442–453 (2007)
4. K.G. Derpanis, M. Sizintsev, K. Cannons, R.P. Wildes, Efficient action spotting based on a spacetime oriented structure representation, in *Computer Vision and Pattern Recognition* (*CVPR*) (2010)
5. J. Gall, V. Lempitsky, Class-specific hough forests for object detection, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2009)
6. I. Laptev, M. Marszalek, C. Schmid, B. Rozenfeld, Learning realistic human actions from movies, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2008)
7. I. Laptev, On space-time interest points. Int. J. Comput. Vis. **64**(2–3), 107–123 (2005)
8. V. Lepetit, P. Lagger, P. Fua, Randomized trees for real-time keypoint recognition, in *Computer Vision and Pattern Recognition*(textitCVPR) (2005)
9. K. Mikolajczyk, H. Uemura, Action recognition with motion-appearance vocabulary forest, in *Computer Vision and Pattern Recognition* (*CVPR*) (2008)
10. K.K. Reddy, J. Liu, M. Shah, Incremental action recognition using feature-tree, in *Proceedings of the IEEE International Conference on Computer Vision* (2009)
11. F. Schroff, A. Criminisi, A. Zisserman, Object class segmentation using random forests, in *Proceedings of the British Machine Vision Conference* (2008)
12. H.J. Seo, P. Milanfar, Action recognition from one example, in *IEEE Transactions on Pattern Analysis and Machine Intelligence* (*PAMI*) (2010)
13. H. Wang, M.M. Ullah, A. Klaser, I. Laptev, C. Schmid, Evaluation of local spatio-temporal features for action recognition, in *Proceedings of the British Machine Vision Conference* (2009)
14. A. Yao, J. Gall, L. Van Gool, A hough transform-based voting framework for action recognition, in *Computer Vistion and Pattern Recogition* (*CVPR*) (2010)
15. T.H. Yu, T.K. Kim, R. Cipolla, Real-time action recognition by spatitemporal sematic and structural forest, in *BMVC* (2010)
16. J. Yuan, Z. Liu, Y. Wu, Discriminative subvolume search for efficient action detection, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2009)
17. J. Yuan, Z. Liu, Y. Wu, Discriminative video pattern search for efficient action detection, in *IEEE Transactions on Pattern Analysis and Machine Intelligence* (*PAMI*) (in press)
18. J. Yuan, Z. Liu, Y. Wu, 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)

http://www.springer.com/978-981-287-166-4

Human Action Analysis with Randomized Trees

Yu, G.; Yuan, J.; Liu, Z.

2015, VIII, 83 p. 30 illus. in color., Softcover
ISBN: 978-981-287-166-4