International Journal of Computer Vision manuscript No.
(will be inserted by the editor)

Activity representation with motion hierarchies

Adrien Gaidon · Zaid Harchaoui · Cordelia Schmid

Received: date / Accepted: date

Abstract Complex activities, *e.g.*, pole vaulting, are composed of a variable number of sub-events connected by complex spatio-temporal relations, whereas simple actions can be represented as sequences of short temporal parts. In this paper, we learn hierarchical representations of activity videos in an unsupervised manner. These hierarchies of mid-level motion components are data-driven decompositions specific to each video. We introduce a spectral divisive clustering algorithm to efficiently extract a hierarchy over a large number of *tracklets* (*i.e.*, local trajectories). We use this structure to represent a video as an unordered binary tree. We model this tree using nested histograms of local motion features. We provide an efficient positive definite kernel that computes the structural and visual similarity of two hierarchical decompositions by relying on models of their parent-child relations. We present experimental results on four recent challenging benchmarks: the High Five dataset [Patron-Perez et al, 2010], the Olympics Sports dataset [Niebles et al, 2010], the Hollywood 2 dataset [Marszalek et al, 2009], and the HMDB dataset [Kuehne et al, 2011]. We show that per-video hierarchies provide additional information for activity recognition. Our approach improves over unstructured activity models, baselines using other motion decomposition algorithms, and the state of the art.

Keywords Action recognition · Video analysis · Motion decomposition · Spectral clustering · Kernel methods

A. Gaidon
Xerox Research Center Europe
e-mail: adrien.gaidon@xrce.xerox.com

Z. Harchaoui · C. Schmid
LEAR team, INRIA Grenoble Rhône-Alpes
655 Avenue de l'Europe 38330 Montbonnot, France
e-mail: {zaid.harchaoui, cordelia.schmid}@inria.fr

1 Introduction

Video content often relates to humans and their actions. Simple actions, *e.g.*, running, rarely convey enough meaning to interpret a dynamic scene, but instead form the building blocks of more complex activities, *e.g.*, pole vaulting. In this work, we aim at recognizing such *high-level activities*: spatio-temporal patterns composed of a *variable* number of *related* movements of actors, body parts, and objects. Automatically identifying those parts and exploiting both their content and their relations is a challenging and important problem for the recognition of complex activities. Therefore, we introduce an unsupervised approach to *hierarchically decompose the complex motion content of an activity*, and an efficient algorithm to compare two *tree-structured videos*.

Our method consists in, first, extracting short-term trajectories of densely sampled points (*tracklets* [Wang et al, 2013]), which describe most of the motion contained in a video. We, then, decompose the motion content of a video into a *hierarchy of data-driven parts* by using hierarchical clustering on the set of tracklets. Our main contribution is a *hierarchical divisive clustering* algorithm based on recursive bi-partitioning of an approximate multi-modal spectral embedding of tracklets. We use the resulting structure, called *cluster-tree* [Duda et al, 2001] (*cf.* Figure 1), to model a video as an unordered binary tree, called *BOF-tree*, which we represent by nested bag-of-features histograms of local motion features. Using this structured information presents several challenges. First, BOF-trees have a variable number of nodes (motion components), and a structure specific to each video. Second, there is no natural left-to-right ordering of the children of the same parent node. Therefore, we introduce a positive definite kernel on variable-sized, unordered binary trees. It consists in efficiently comparing all their sub-trees by approximating sub-trees with simple edge models, and leveraging the additive structure of BOF-

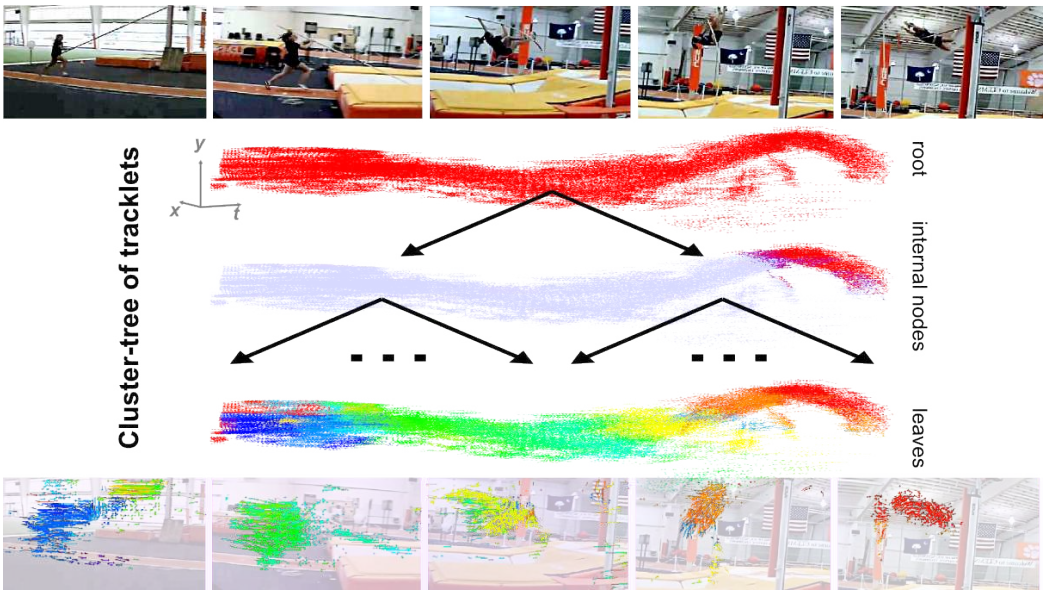

Fig. 1: *Example hierarchical motion decomposition obtained with our recursive bi-partitioning algorithm on dense tracklets.*

trees. We, then, use this kernel with Support Vector Machines (SVM) [Schölkopf and Smola, 2002] to learn powerful non-linear activity classifiers.

Related work

Our approach is weakly supervised, as it relies neither on part annotations, nor on a predefined action decomposition as in [Gaidon et al, 2011], but only requires training videos with activity labels. In addition, we use neither video segmentation techniques [Brendel and Todorovic, 2011, Brox and Malik, 2010, Grundmann et al, 2008, Lezama et al, 2011], nor manually pre-defined semantic attribute detectors [Prest et al, 2012, Sadanand and Corso, 2012]. We argue that *clusters of tracklets* provide a compromise between (i) detailed information available from segmentation and (ii) higher-level attributes, body part localizations, or object detections. Although our tracklets are of a short duration, the clusters obtained with our hierarchical decomposition capture complex and long-term motions of spatio-temporal parts, as well as their relations (*cf.* Figure 1 for an illustration).

Our method is based on the Nyström approximation for spectral clustering [Fowlkes et al, 2004] to compute an approximate spectral embedding from a fraction of the tracklets of a video. We do not require the number of clusters to be globally fixed and known a priori (in contrast to Fowlkes et al [2004]), and our hierarchical decomposition provides useful structural information relating the motion parts together, instead of unrelated clusters. Furthermore, our *divisive* method can scale to videos with hundreds of thousands of tracklets, whereas bottom-up agglomerative algorithms have a computational complexity at least quadratic in the number of points [Fradet et al, 2009, Hastie et al, 2008]. As we deal with tracklets, another difference with existing trajectory clustering approaches [Brox and Malik, 2010, Fradet et al, 2009, Lezama et al, 2011] is that we do not restrict trajectory comparisons to their common time span.

Several methods, *e.g.*, [Brendel and Todorovic, 2011, Laxton et al, 2007, Niebles and Fei-Fei, 2007, Oliver et al, 2000, Todorovic, 2012], represent the structure of complex activities using probabilistic graphical models. For instance, Niebles and Fei-Fei [2007] use a constellation of parts represented by BOFs over shape and motion features. They model a category using a probabilistic mixture of a fixed number of parts. They classify actions in controlled video conditions by maximizing the likelihood with respect to their generative model. More recently, Todorovic [2012] use background subtraction to learn a powerful generative model of an activity's local co-occurrence patterns between foreground motion features. Brendel and Todorovic [2011] propose another general graphical model of spatio-temporal relations between parts learned from hierarchical video segmentations. However, their video segmentation algorithm — and therefore the resulting action models — can only account for smooth motions of color-consistent parts forming continuous spatio-temporal "tubes". We do not make these restrictive assumptions on parts. Furthermore, we estimate and compare each video's specific structure, and learn a discriminative model instead of matching to a global action template.

Discriminative alternatives to these generative models, *e.g.*, [Bilen et al, 2011, Liu et al, 2011, Niebles et al, 2010, Raptis et al, 2012, Tang et al, 2012, Wang and Mori, 2011], often use techniques related to the popular deformable part model of Felzenszwalb et al [2010]. For instance, Liu et al

[2011] combine manually predefined attributes with data-driven ones obtained by clustering local features. They use a latent SVM [Felzenszwalb et al, 2010] to learn the importance of each part. Niebles et al [2010] discover temporal parts and learn an SVM classifier per video segment at latent temporal locations. In addition to discovering discriminative temporal segments, Tang et al [2012] model the transitions between hidden states and their durations using a semi-Markov Model [Hongeng and Nevatia, 2003], whose parameters are learned using a latent SVM. Bilen et al [2011] deal not only with temporal aspects, but also with spatial ones by finding the best crops and splits of spatio-temporal video volumes. More closely related to our work, Raptis et al [2012] extract clusters of long-term trajectories, and learn a latent model over a fixed number of parts. Their approach has a cubic time complexity in the number of trajectories, relies on bounding box annotations, and uses only a fixed small subset of clusters for all videos. Furthermore, they explicitly model pairwise relationships between clusters using their mean trajectory, whereas we use the full hierarchical structure resulting from our clustering.

In contrast to the previous approaches, we do not assume that all actions of the same category share strong geometrical and temporal relations between a fixed number of parts common to all training instances. Instead, *each video has its own decomposition structure*, and all parts — including their relationships — are used in each comparison between videos. This means we compare videos based on their structure and content, instead of trying to align or match video sub-volumes. In addition to its robustness to intra-class variability, this approach does not need to solve a complex inference problem for each test video.

We also differ from global hierarchies of local features [Mikolajczyk and Uemura, 2008, Reddy et al, 2009] and shape-motion prototypes [Jiang et al, 2012b]. These methods use global tree structures to speed up the computation of a matching score, where each feature casts a vote for an action category. Finally, we account for all local features jointly, and leverage instance-level relationships. In contrast to [Gilbert et al, 2010, Jiang et al, 2012a, Kovashka and Grauman, 2010, Matikainen et al, 2010], we do not explicitly model feature neighborhoods and co-occurrences.

Our approach is inspired by the work on hierarchical image segmentation of Pablo et al [2011], Shi and Malik [2000], which relies on spectral clustering to represent an image as a region tree. We, however, propose a different clustering algorithm adapted to activities. Furthermore, our goal is not just to build a hierarchical video representation, but also to leverage it for activity recognition. A preliminary version of this work appeared in [Gaidon et al, 2012].

This paper is organized as follows. Section 2 describes tracklets and our clustering algorithm. Section 3 details our hierarchical video model, the BOF-Tree, and our proposed kernel. Section 4 contains our experiments. We evaluate our method on the complex activities from the High Five [Patron-Perez et al, 2010] and Olympic Sports [Niebles et al, 2010] datasets. We also show that our approach can be successfully applied on a large set of atomic actions from the Hollywood 2 [Marszalek et al, 2009] and HMDB [Kuehne et al, 2011] datasets, although the performance gains are smaller.

2 Clustering dense tracklets

In this section, we first describe how we extract dense local point trajectories, called *tracklets* (*cf.* Figure 2) using dense optical flow fields. We then address the problem of efficiently and accurately clustering a video composed of a large number of tracklets in order to obtain a hierarchical decomposition of its motion components. We describe the two steps of our clustering algorithm: a non-linear projection of the tracklets on a multi-modal spectral embedding using the Nyström approximation, followed by a hierarchical divisive clustering algorithm. Note that the algorithm described in this section is applied to a single video at a time and does not require any other external data, such as other videos where the same action occurs.

2.1 Extracting dense tracklets

We summarize the approach of Wang et al [2013], which we use to track densely sampled points.

First, we compute the dense optical flow field of the video using the algorithm of [Farnebäck, 2003] as implemented in the OpenCV library [Bradski and Kaehler, 2008]. This iterative multi-scale approach is based on approximating the neighborhood of pixels by quadratic polynomials using polynomial expansion.

Second, we sample the pixels to be tracked on a dense spatial grid. Similar to Wang et al [2013], we observed that a sampling step-size of 5 pixels is a good compromise between density and speed given the dimensions of our videos — between 50 and 1000 frames for resolutions close to 640×480. As points in homogeneous regions cannot be reliably tracked, we filter them out following the criterion [Shi and Tomasi, 1994, Wang et al, 2013]: if the smallest eigenvalue of the autocorrelation matrix of a point is lower than a threshold, it is removed from the set of points to be tracked.

Third, we use a multi-scale video representation over 8 spatial scales, spaced by a factor $1/\sqrt{2}$. We do not use multiple temporal scales — *i.e.*, we track points from frame t to $t+1$ — as we are only interested in *local* point trajectories. Each point $P_t = (x_t, y_t)$ at frame t is robustly tracked in each scale by median filtering in the dense optical flow field $w = (u_t, v_t)$:

$$P_{t+1} = (x_{t+1}, y_{t+1}) = (x_t, y_t) + (M * w)|_{\bar{x}_t, \bar{y}_t}, \qquad (1)$$

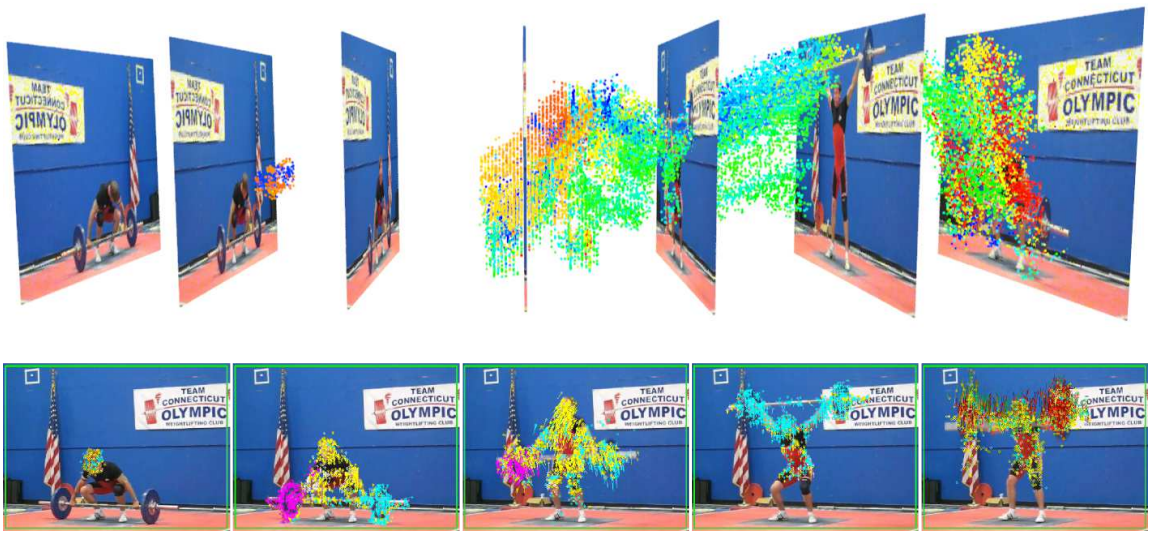

Fig. 2: *Tracklets of a weightlifting activity. Colors correspond to the most detailed clusters obtained with our algorithm.*

where M is the median filtering kernel and $(\bar{x}_t, \bar{y}_t)$ is the rounded position of (x_t, y_t). Note that this interpolation step is efficient once the dense optical flow field is extracted. This approach, therefore, allows to efficiently track points on a dense spatial grid. The new frame $t+1$ may then contain good regions void of points to be subsequently tracked. Indeed, points can be lost and new ones can appear due to a change in the scene. Therefore, and in order to maintain density, we fill these regions by adding new points re-sampled on the dense spatial grid.

Point trajectories interpolated from an optical flow field are often deviating from the underlying tracked pixel after a certain number of frames. In order to limit this drifting problem, Wang et al [2013] propose to track points only across a fixed small number of frames L. In their experiments, they show that using tracklets of duration $L = 15$ frames yields state-of-the-art action recognition results. We used the same setting in all our experiments except on the sports videos of the Olympic Sports dataset, as we observed that, in this case, shorter tracklets ($L = 5$) yield better results ($+5\%$ accuracy on average for all the methods we tested). Indeed, when dealing with fast motions, the displacements of the underlying pixels are large, motion blur causes point tracks to drift faster, and self-occlusions make it often impossible to track interesting body parts for more than a few frames. This is especially frequent in sports videos, *e.g.*, an athlete spinning on himself before throwing a discus.

Although such short duration tracklets convey less information, *clusters* of tracklets can encode mid-level information on par with variable-length longer-term trajectories (*cf.* Figure 2). Furthermore, errors in long-term trajectories due to drift cannot be easily recovered from at later stages. On the contrary, errors in short-term tracklets amount to noise that can be appropriately handled via some simple post-processing steps, as described in the following paragraphs. Similar to Wang et al [2013], we remove tracklets with static trajectories. They, indeed, convey no motion information, and are often associated either with a static textured background (*e.g.*, trees) or with noise. We also handle the other extremal case by removing tracklets containing large and sudden displacements.

In addition to these simple pruning steps followed by Wang et al [2013], we filter out tracklets that are in low-density regions. In more details, isolated but reliably tracked points — *e.g.*, due to block-like compression artifacts — are considered as outliers and removed. We use a simple sliding window outlier estimation technique that has proven effective in our experiments.

For each tracklet $\mathbf{P} = (P_t)_{t=F-L+1,\cdots,F}$ ending at frame F, we first estimate its approximate spatial k-nearest neighbors $\mathcal{N}_{k,r}(\mathbf{P})$ *restricted* to tracklets ending between frames $F-r$ and $F+r$. In our experiments, we used $k = 30$ neighbors, $r = 5$ frames, and the spatial distance between tracklets is estimated between their respective average positions $\bar{P} = (\bar{x}, \bar{y}) = \frac{1}{L}\sum_{t=1}^{L} P_t$. We then compute the local sparsity $s_{\mathbf{P}}$ of a tracklet as the mean spatial distance to its neighbors in the temporal window $[F-r, F+r]$:

$$s_{\mathbf{P}} = \frac{1}{k} \sum_{\mathbf{P}' \in \mathcal{N}_{k,r}(\mathbf{P})} \|\bar{P} - \bar{P}'\| \tag{2}$$

We consider a tracklet with local sparsity $s_{\mathbf{P}}$ as an outlier if $s_{\mathbf{P}} > \bar{s} + \bar{\sigma}_s$, where $(\bar{s}, \bar{\sigma}_s)$ is the mean and standard deviation of the spatial distance to the approximate spatial k-nearest neighbors of all tracklets in the entire video, irrespective of temporal position. The approximate k-nearest neighbors are computed *via* kd-trees, which are efficiently built in the space of average spatial positions of tracklets.

2.2 Tracklet descriptors for intra-video clustering

Once the tracklets have been extracted, we model them using multiple features describing their spatio-temporal position and shape. We use the following descriptors to represent trajectory information, where we separate the location and shape information along the x, y and time axes to allow for different normalizations per feature-type and per dimension:

- $\mathbf{x} = (x_1, \cdots, x_L)$, x positions over time,
- $\mathbf{y} = (y_1, \cdots, y_L)$, y positions over time,
- $\mathbf{z} = (t, \cdots, t + L - 1)$, temporal positions,
- $\mathbf{v_x} = (x_{k+1} - x_k)_{k=1:L-1}$, velocities along x-axis,
- $\mathbf{v_y} = (y_{k+1} - y_k)_{k=1:L-1}$, velocities along y-axis.

Our approach, however, is not specific to the set of features chosen to represent a tracklet. The only pre-requisite is the availability of a similarity function specific to each feature channel. We use Gaussian RBF kernels $k(f, f') = exp(-\gamma d(f, f')^2)$, where the distance $d(f, f')$ is the Euclidean distance. The γ parameter of each kernel is automatically fixed to $\gamma = 1/(2\bar{d})$, where $\bar{d}$ is an estimate of the median of the distances between the corresponding tracklet features. This normalization ensures that the different feature channels are comparable.

Note that the tracklet descriptors need not be particularly robust here, as the goal is to cluster tracklets to decompose the motion components *inside* a video. Hence, all comparisons are *intra-video* during this clustering stage, and descriptors with too many invariances would prevent the discovery of finer motion patterns. We, indeed, found that clustering based on robust local features commonly used for action recognition (*e.g.*, HOG, HOF, or MBH descriptors of tracklets [Wang et al, 2013]) yields less discriminative decompositions. We also found that using both the trajectory and the shape (velocities) of the tracklets yields better results than trajectory or shape alone, suggesting that these two aspects are complementary for clustering.

2.3 Multi-modal spectral embedding of tracklets

Once the features are extracted, the first step of our clustering algorithm is to project tracklets onto a low-dimensional space. This embedding relies on spectral properties of a similarity matrix between tracklets.

2.3.1 Similarity between tracklets

We define the similarity between tracklets as the *product of the per-feature similarities* in order to combine the multiple descriptors representing different characteristics of tracklets. As each feature-specific similarity is positive-definite, the product similarity is also a positive-definite kernel. It corresponds to a feature space that is the tensor product of the feature spaces induced by the individual kernels. Using this product kernel has the advantage of entailing an "and" effect, *i.e.*, tracklets close according to this similarity are close with respect to *all* the features used. This behavior is desirable as all our features convey meaningful information representing our priors on tracklets, such as "similar tracklets should be close *and* have similar shapes".

In contrast, using an "early fusion" approach — *i.e.*, concatenating the descriptors into a single vector or summing the per-feature kernels — has an "or" effect: tracklets are close according to *at least one* feature. In addition, early fusion mixes heterogeneous descriptors with various scales into a long vector, for which the Euclidean distance is not adequate. Therefore, central grouping methods like k-means are not adapted to our set-up, as they search for clusters with a Gaussian distribution in the Cartesian product of the input spaces. These shortcomings are addressed by resorting to a spectral clustering method, which operates on an embedding of the data points.

2.3.2 Spectral embedding

The spectral embedding of our tracklets is based on projections onto the leading eigenvectors of the graph Laplacian corresponding to a similarity matrix between tracklets. Let $W \in \mathbb{R}^{N \times N}$ be a symmetric similarity matrix between N tracklets. This matrix can be viewed as the weighted adjacency matrix of a graph, where the nodes are the tracklets and the edges are weighted by the pairwise affinity between tracklets. The normalized Laplacian of this graph is defined as $\mathcal{L} = I - D^{-1/2} W D^{-1/2}$, where D is the diagonal matrix of row-sums of W. Thresholding the eigenvector of $\mathcal{L}$ with the second smallest eigenvalue yields an approximate solution to the NP-Hard Normalized Cut (NCut) bi-partitioning problem [Shi and Malik, 2000]. Note that the leading eigenvalue-eigenvector pair of $\mathcal{L}$ is $(0, D^{1/2}\mathbf{1})$.

In order to obtain a predetermined number of clusters, one can either apply this technique recursively on the next leading eigenvectors, or use a distortion-minimization clustering algorithm — *e.g.*, k-means — on an embedding consisting of the projections on multiple leading eigenvectors.

For N data points, the N_E-dimensional spectral embedding ($N_E \ll N$) is obtained by computing the $N \times (N_E + 1)$ matrix V of the $N_E + 1$ leading eigenvectors and the $(N_E + 1) \times (N_E + 1)$ diagonal matrix Λ of eigenvalues of the system:

$$\left(D^{-1/2} W D^{-1/2}\right) V = V \Lambda. \tag{3}$$

The j^{th} embedding coordinate of the i^{th} tracklet is given by:

$$E_{i,j} = \frac{V_{i,j+1}}{\sqrt{D_{i,i}}}, \quad i = 1, \cdots, N, \; j = 1, \cdots, N_E, \tag{4}$$

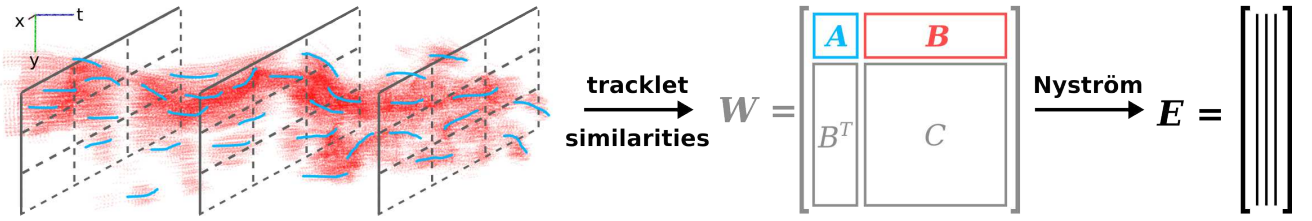

Fig. 3: *Illustration of the Nyström approximation. We first sample $n \ll N$ tracklets by randomly choosing p points in each cell of a regular spatio-temporal grid. We then compute the similarity between the selected tracklets (in blue), yielding matrix A, as well as with all other $N - n$ points (in red), yielding matrix B. The Nyström method allows to efficiently compute a spectral embedding E from the approximate leading eigenvectors $\hat{U}$ of the large $N \times N$ matrix W of all similarities using only the small $n \times N$ slice of it, composed of blocks A and B.*

where the eigenvectors are sorted by ascending eigenvalue. Thus, each tracklet is associated with a row of E, and a clustering algorithm can be applied on the rows of E to partition the data.

However, the spectral embedding cannot be directly computed this way in our case. Indeed, for N tracklets, solving equation 3 requires the computation and storage of a $N \times N$ similarity matrix. In addition, its computational complexity is in $O(N^3)$, which seems prohibitive in our setup, where the number of data points can be on the order of 10^6.

A mechanism to overcome this complexity is to use a sparse affinity matrix — *e.g.*, by thresholding its entries — and efficient sparse eigensolvers such as the Lanczos method [Shi and Malik, 1998]. With our dense tracklets, the affinity matrix is not sparse, and thresholding its entries still requires the evaluation of all similarities. Thresholding also has side-effects that are not well understood [Fowlkes et al, 2004]. For instance, thresholding a kernel matrix may not preserve its positive-definiteness. Therefore, we choose to use the Nyström method in order to efficiently compute an approximate spectral embedding.

2.3.3 Nyström approximation

The Nyström method is a technique for finding numerical approximations to eigenfunction problems. It is related to kernel PCA [Schölkopf et al, 1998, Williams and Seeger, 2001], and has been introduced in the context of spectral clustering in [Fowlkes et al, 2004]. It allows to extend an eigenvector computed for a subset of points to any arbitrary point, while requiring only a subset of the columns (or rows) of the similarity matrix W (see Figure 3 for an illustration).

Let A be the $n \times n$ similarity matrix between a subset of n randomly sampled tracklets, with $n \ll N$, and let B be the $n \times (N - n)$ similarity matrix between this subset and all other tracklets. With no loss of generality, we assume that the samples are sorted such that we can rewrite the full $N \times N$ similarity matrix W as:

$$W = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \tag{5}$$

with $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times (N-n)}$ and $C \in \mathbb{R}^{(N-n) \times (N-n)}$, C being in general too expensive to compute as $N \gg n$. Note that in our case W and A are positive-definite.

Let $A = U_A \Lambda_A U_A^T$ be the eigendecomposition of A. The matrix form of the Nyström extension is $B^T U_A \Lambda_A^{-1}$. The approximate eigenvectors $\hat{U}$ of W given by the Nyström method are:

$$\hat{U} = \begin{bmatrix} U_A \\ B^T U_A \Lambda_A^{-1} \end{bmatrix} \tag{6}$$

Note that $\hat{U}$ are also the eigenvectors of the approximation $\hat{W}$ of W:

$$\hat{W} = \hat{U} \Lambda_A \hat{U}^T = \begin{bmatrix} A & B \\ B^T & B^T A^{-1} B \end{bmatrix} \tag{7}$$

It shows that the Nyström method implicitly approximates C using $B^T A^{-1} B$. Thus, the quality of the approximation can be measured by the norm of the Schur complement $\|C - B^T A^{-1} B\|$, which quantifies how well C is spanned by the rows of B. Consequently, the subset of rows of W must correspond to tracklets spanning the clusters to be discovered. Therefore, to ensure good diversity and coverage, we randomly subsample p tracklets per cell of a spatio-temporal grid over the whole video (*cf.* Figure 3). The grid granularity and p are adapted dynamically to the total number of tracklets and the available computational resources. Similarly to Fowlkes et al [2004], we observed that, in practice, good results can be obtained with less than 1% of the total number of points.

As observed in [Fowlkes et al, 2004], the columns of $\hat{U}$ are not orthogonal. However, as A is positive-definite in our case, we can apply the "one-shot" technique of [Fowlkes et al, 2004] and compute the orthogonalized approximate eigenvectors of W in one step as follows.

Let $S = A + A^{-1/2}BB^TA^{-1/2}$, where $A^{-1/2}$ is a pseudo-inverse of a symmetric positive definite square root of A. Let $S = U_S\Lambda_S U_S^T$ be the eigendecomposition of S. As shown in [Fowlkes et al, 2004], $\hat{W}$ is diagonalized by $\hat{V}$ and Λ_S, where:

$$\hat{V} = \begin{bmatrix} A \\ B \end{bmatrix} A^{-1/2} U_S \Lambda_S^{-1/2} \tag{8}$$

Note that in the context of NCut, A and B must be normalized by the row-sums $\hat{d}$ of $\hat{W}$ beforehand. This can be done without computing the expensive $B^TA^{-1}B$ block of $\hat{W}$ by:

$$A_{i,j} \leftarrow \frac{A_{i,j}}{\sqrt{\hat{d}_i \hat{d}_j}}, \quad B_{k,l} \leftarrow \frac{B_{k,l}}{\sqrt{\hat{d}_k \hat{d}_{l+n}}}, \tag{9}$$
$$i,j,k = 1,\cdots,n, \quad l = 1,\cdots,N-n$$

where

$$\hat{d} = \hat{W}\mathbf{1}_N = \begin{bmatrix} a_r + b_r \\ b_c + B^TA^{-1}b_r \end{bmatrix} \tag{10}$$

with $\mathbf{1}_N$ the column vector of N ones, $a_r, b_r \in \mathbb{R}^n$ the row-sums of A and B, and $b_c \in \mathbb{R}^{N-n}$ the column-sums of B. After normalizing A and B (eq. 9, 10) and computing $\hat{V}$ (eq. 8), the spectral embedding E is:

$$E_{i,j} = \frac{\hat{V}_{i,j+1}}{\hat{V}_{i,1}}, \quad i = 1,\cdots,N,\ j = 1,\cdots,N_E, \tag{11}$$

In our set-up, special care must be taken with respect to numerical stability. Indeed, even though A is positive-definite, it is in general ill-conditioned. Therefore, we use a robust pseudo-inverse algorithm to compute A^{-1} by thresholding the lower-end of its spectrum. We also follow the useful guidelines from [Foster et al, 2009]. Note that the overall computational cost of this approximate spectral embedding is $O(n^2N)$ in time and $O(nN)$ in space, which is a large improvement over the complexity of the exact method, *i.e.*, $O(N^3)$ in time and $O(N^2)$ in space.

2.4 Hierarchical divisive clustering

Using the computed spectral embedding, we cluster tracklets via an efficient hierarchical divisive algorithm. The pseudo-code of our approach is given by algorithm 1 and 2.

We adopt a top-down approach that consists in recursively bi-partitioning the set of tracklets (*cf.* Figure 1). We split a set of tracklets in two by thresholding along an eigenvector, *i.e.*, along a dimension of the spectral embedding. As the second smallest eigenvector is the real-valued solution to the Normalized-Cut (NCut) problem, it is composed of two separated ranges of values indicating the optimal partition of the tracklets. This argument is also valid for the next leading eigenvectors and, theoretically, one can optimally sub-partition the data by recursively thresholding one eigenvector after the other.

Algorithm 1 spectral_division

Input: features of N tracklets (*cf.* section 2.2)
Output: the cluster-tree (hierarchical set of nodes)
Compute the spectral embedding
Compute the slice $[A \quad B]$ of W (*cf.* fig. 3)
Normalize A and B (eq. 9, 10)
Compute the embedding E (eq. 8, 11)
Initialize priority queue over nodes to be split
`to_split` ← empty priority queue of nodes
Push the root of the cluster-tree on `to_split`
# Recursively split nodes according to priority
while `to_split` is not empty **do**
  `node` ← pop highest priority node from `to_split`
  # Find the best split, *cf.* algorithm 2
  `left, right` = **find_best_split**(`node`)
  # Add children to the queue
  if `left` and `right` are not empty **then**
    Push `left` on `to_split`
    Push `right` on `to_split`
  end if
end while

Algorithm 2 find_best_split

Inputs: `node`: set of tracklets, E: embedding
Parameters: m & M: min & max leaf sizes
Output: `left`, `right`: nodes of best split found
# Determine if we cannot split further
if $|$`node`$| \leq 2m$ **then**
  return $\emptyset, \emptyset$ # `node` is a leaf
end if
# Determine score to improve upon
if $|$`node`$| \leq M$ **then**
  `best_score` ← score of `node`
else
  `best_score` ← $-\infty$ # force the split
end if
# Greedily search for the best split
`left, right` ← $\emptyset, \emptyset$
for $j = 1$ **to** N_E **do**
  for e in candidate thresholds **do**
    `sl` ← **connectedness**($\{i \in$ `node` ; $E_{i,j} < e\}$)
    `sr` ← **connectedness**($\{i \in$ `node` ; $E_{i,j} \geq e\}$)
    `split_score` ← min(`sl`, `sr`)
    if `split_score` > `best_score` **then**
      `best_score` ← `split_score` # best split so far
      `left` ← $\{i \in$ `node` ; $E_{i,j} < e\}$
      `right` ← $\{i \in$ `node` : $E_{i,j} \geq e\}$
    end if
  end for
  # Check the stopping criterion
  if an improving split was found **then**
    if $j > 1$ **or** split was not forced **then**
      return `left, right` # the best split found
    end if
  end if
end for
return `left, right`

However, several practical problems need to be taken into account while following such an approach. First, the optimal bi-partition is often unclear in realistic conditions and the eigenvectors tend to reflect this ambiguity by having smooth variations. Therefore, which threshold to use is in general unclear when looking only at eigenvector values. Furthermore, thresholding a smoothly varying eigenvector can lead to an unstable partition, as a small change in the threshold significantly modifies the partition.

The first technique we use to address this stability problem is based on a criterion similar to the one mentioned in [Shi and Malik, 2000]. If the variations of an eigenvector are below a small threshold, then we do not attempt to split along this eigenvector. We used 10^{-10} as threshold in our experiments. More importantly, we propose to circumvent the difficulties posed by smooth eigenvectors by finding the threshold that maximizes a model selection score based on spatio-temporal information (*cf.* Algorithm 2). We experimented with multiple spatio-temporal consistency criteria, such as inertia or label agreement amongst neighbors, but the one that yielded the best results in our experiments is the *connectedness* measure described in the following.

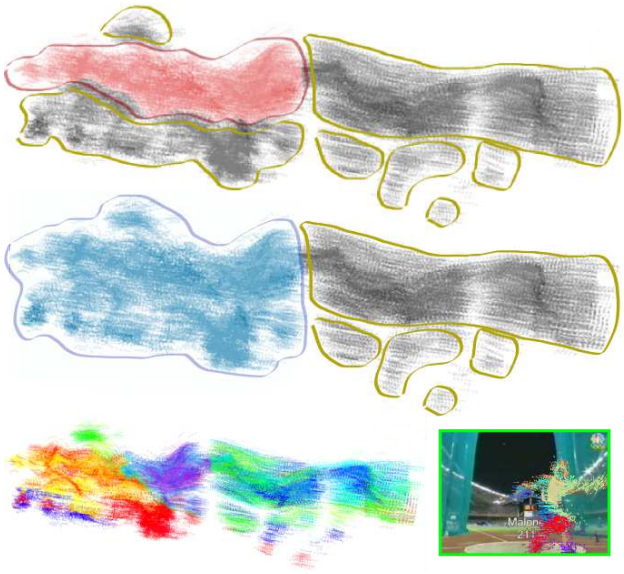

Fig. 4: *Spatio-temporally connected components (of a "discus throw" video) obtained by two different candidate bi-partitions (top two rows). Our method selects the second possibility, as it yields a higher connectedness ($1/5$ instead of $1/7$). The last row shows the leaves of our cluster-tree for this video and one of its frames.*

Connectedness score of a node

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a spatio-temporal k-nearest neighbor graph over the tracklets $\mathcal{V}$ in a video: $(v_i, v_j) \in \mathcal{E}$ if and only if $v_i, v_j \in \mathcal{V}$ are k-nearest neighbors according to their spatio-temporal positions. We define the connectedness score $c(\mathcal{V}')$ of a subset of tracklets $\mathcal{V}' \subset \mathcal{V}$ forming a node in the cluster-tree as $c(\mathcal{V}') = |\mathcal{C}(\mathcal{V}')|^{-1}$, where $\mathcal{C}(\mathcal{V}')$ is the set of connected components of the subgraph $\mathcal{G}(\mathcal{V}')$ of $\mathcal{G}$. Maximizing this connectedness score has a strong advantage over the minimization of distortion-based measures such as inertia: it does not constrain the shape of the clusters. In particular, it enforces neither spherical nor "tube-like" clusters, but only *encourages* spatio-temporal contiguity of the tracklets in order to avoid clearly spatio-temporally disjoint parts in the same node. Note also that connectedness is not biased with respect to node size. In addition, it can be efficiently obtained by pre-computing once — during an initialization step — the full spatio-temporal adjacency matrix of all tracklets in the video. We compute this sparse matrix using *kd-trees* on the average spatio-temporal position of tracklets to determine the graph of approximate k-nearest neighbors. As this is fast to compute, including for hundreds of thousands of tracklets, we select the smallest k such that the entire video has exactly one spatio-temporally connected component. This yields approximatively $k = 10$ neighbors on average over all videos. Once the neighborhood graph is built, computing the number of connected components of a subgraph is done in linear time by a depth-first search.

This connectedness score is specific to a node in the cluster-tree, and when deciding whether a split improves performance, we compare the score of the parent node with the lowest score of its candidate children (*cf.* Algorithm 2 and Figure 4). The score of a node is also employed to determine the order used to choose the nodes to split. As outlined in Algorithm 1, we use a simple min-heap-based priority queue to split nodes with the *lowest* score first, as they are those with the maximum expected gain. Ties in priority are handled by picking the largest node, then the node closer to the root. We observed that, in general, this choice of priority leads to a depth-first construction of the cluster-tree, with occasional backtracking to higher nodes. Note that we do not use the NCut value as score, as we do not compute the full similarity matrix W, and smooth variations in an eigenvector yields smooth variations in the NCut value.

The second issue faced with a recursive thresholding approach, is that the accumulation of approximation errors that makes higher eigenvectors less reliable. We, therefore, use this natural ordering of the eigenvectors in order to find a good split. We adopt a greedy strategy depicted in Algorithm 2. We first try candidate thresholds along the leading eigenvector. In our experiments, we use nine adaptive thresholds corresponding to the 10^{th}, 20^{th}, $\ldots$, 90^{th} percentile of the considered eigenvector values, in order to avoid severe imbalance between nodes at the same depth in the cluster-tree. If the best split along this eigenvector improves with respect to the score of the parent node that we try to split, then the split is registered with this eigenvector-threshold pair. Otherwise, we iteratively try to split along the next

leading eigenvector, until we reached the last one. We observed that less than 20 eigenvectors are generally used. For each node, we start again from the leading eigenvector, as suggested in [Shi and Malik, 2000], because it is the most reliable one, and we observe that, in practice, the leading eigenvectors might be decomposed over several plateaus.

The only hyper-parameters of our divisive algorithm are the minimum and maximum leaf sizes: m and M. They are employed in order to avoid "degenerate" — *i.e.*, too large or too small — leaves, a frequent problem faced by central grouping methods such as k-means. If a node to be split contains less than $2m$ tracklets, we automatically mark it as a leaf. If a node is larger than M, we force the split by using the best eigenvector-threshold pair, even if the scores of the children nodes are lower than the score of the parent (*cf.* Algorithm 2). We used $m = 200$ and $M = 2000$ in our experiments.

3 Classification of cluster-trees

We now explain how we represent each node in the cluster-tree as a histogram of quantized tracklet features, and introduce a tree kernel to efficiently compare cluster-trees for activity recognition.

3.1 Tracklet descriptors for classification

As described in Section 2.2, we use simple discriminative tracklet descriptors in order to discover the detailed motion structure of each video. During classification, however, we compare the content of different videos. Therefore, recognition requires more robust features in order to build models that can handle the large intra-class variability of activities in real-world videos. We represent a tracklet using Motion Boundary Histograms (MBH) [Dalal et al, 2006, Wang et al, 2013]. It consists of two histograms quantifying the gradients of the horizontal and vertical components of the optical flow. As shown in [Wang et al, 2013] for the bag-of-features model, using MBH to describe tracklets allows for better action recognition than when using trajectory information. As MBH relies on the derivatives of the optical flow, it suppresses constant motion information and focuses on representing motion boundaries, *i.e.*, local changes of orientation in the motion field. Therefore, it can be robust to a certain amount of camera translation, but not to more complex movements such as rotations. In addition, MBH allows to robustly handle the noise in the spatial derivatives of the optical flow *via* quantization. We use the same parameters as in [Wang et al, 2013] to efficiently compute tracklet-aligned MBH descriptors directly from the dense optical flow field used to obtain the tracklets. We could also combine MBH with other features (*e.g.*, with HOG and HOF [Wang et al, 2013]), but this results in moderate performance gains at a significantly increased computational cost.

Note that we found that MBH is not adapted as a clustering feature. Its robustness, indeed, can yield large clusters containing multiple distinct objects moving in the same manner, *e.g.*, soccer players running on a soccer field, which, despite their different spatio-temporal locations, cannot be separated according to MBH features.

3.2 BOF-Tree: tree of nested bag-of-features

As a node is a motion component composed of a diverse set of tracklets, we propose to efficiently encode this diversity with a bag-of-features (BOF). Note that we could also represent a node's content with the average of its tracklets. However, this has several limitations in our situation. First, there is a potentially large number of points per node, which makes a node qualitatively different from a "super-tracklet", especially at a low depth in the cluster-tree. Second, as our clustering algorithm uses different features than the ones used for classification, there is, in general, a high variability of tracklet descriptors in a node, *i.e.*, a large dispersion around the centroid. In addition, the spatio-temporal consistency of the nodes is not strictly enforced. Thus, the global shape of our nodes may not be discriminative, due to holes and noise caused by our approximate clustering algorithm. In contrast, the BOF representation has proven to be a reliable way to describe entire videos as well as smaller temporal action units [Gaidon et al, 2011].

From the set of MBH descriptors and the cluster-tree obtained by our spectral divisive algorithm, we extract a hierarchical representation of a video called *BOF-tree*. Its structure is the same as its corresponding cluster-tree. In addition, each node in the BOF-tree is described by a bag-of-features. See Figure 5 for an illustration. We first pre-compute a vocabulary of track-aligned MBH features on a random subset of the training features. We use an on-line k-means algorithm [Sculley, 2010] with $k = 4000$ to cluster 10^6 tracklets randomly sampled from the training videos. We then quantify all MBH features by assigning them to the closest "visual word" (centroid) in the learned vocabulary. Finally, each node is represented by a BOF, *i.e.*, its histogram of occurrences of visual words. Although we use a high-dimensional representation for better accuracy, this does not cause computational or memory problems. In practice, per-node histograms are indeed sparse, thus can be represented efficiently. Modeling a node with a BOF discards the "intermediate geometry" at the node level — *i.e.*, between neighboring tracklets — while local geometry is still captured by the local tracklet features, and global spatio-temporal structure information is captured by the cluster-tree.

As a consequence of the clustering, the left-to-right order in the BOF-tree does not have any geometrical interpre-

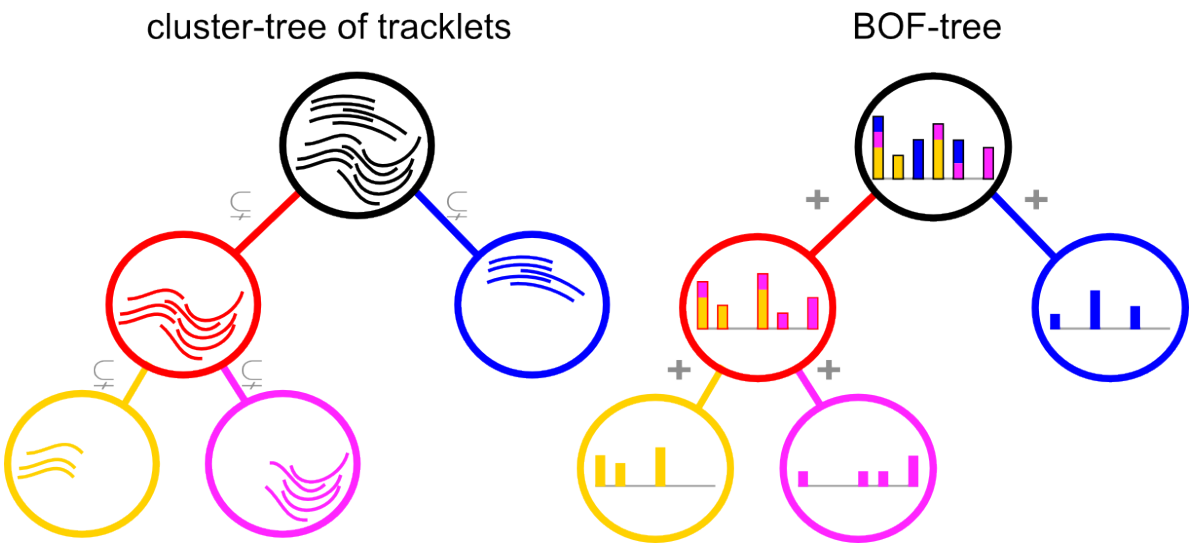

Fig. 5: *(left) A cluster-tree of tracklets, where edges between nodes represent a strict inclusion. (right) Its corresponding BOF-tree, where the BOF of a node is the sum of its children's BOFs.*

tation (the cluster-tree is *unordered*). The only semantic relation directly captured by our tree structure is the inclusion relation derived from the cluster-tree: the two children of a node correspond to a bi-partition of the tracklets of their parent node. This induces an additive property on the nodes of a BOF-tree: the BOF of a node in a BOF-tree is the sum of its children's BOFs (*cf.* Figure 5). We now show how to use this property to efficiently compare BOF-trees, while leveraging the hierarchical structure information between nodes.

3.3 Kernel between BOF-trees

Existing kernels on variable-sized trees are generally inspired by string kernels, *i.e.*, measure structural similarity by counting the number of common sub-structures [Shawe-Taylor and Cristianini, 2004]. In BOF-trees, however, this approach is not directly applicable, as siblings are unordered. In addition, our goal is to use the structure information to disambiguate comparisons between the *content* of the motion component hierarchies, not to directly compare the tree structures. Alternatively, summing over all pairwise node comparisons results in a valid kernel comparing the contents of BOF-Trees (which we use as a baseline, *cf.* Section 4.3), but it ignores their structure.

Instead, a general approach to compare such trees is to decompose them over the set of all possible paths, and compare the resulting *bags of paths* [Dupé and Brun, 2008, Suard et al, 2007]. Although this loses some structural information, it allows to compare graphs with positive definite kernels in polynomial time, whereas the general graph matching problem is NP-Hard [Diestel, 2005].

Let $\mathcal{T}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{T}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ be two BOF-trees, defined from their set of vertices (nodes) $\mathcal{V}_i$ and directed edges (parent-child relations) $\mathcal{E}_i$. Each node $v \in \mathcal{V}_i$ is represented by a BOF, $b(v)$, over its constitutive tracklets. We model a directed edge $e = (v_p, v_c) \in \mathcal{E}_i$ by the *concatenation* $b(e) = (b(v_p), b(v_c))$ of the BOF of the child node v_c with the BOF of its parent node v_p.

Let h be a kernel between BOF. We use the intersection kernel [Maji et al, 2008] between L_1-normalized histograms:

$$h(x, x') = \sum_j \min\left(\frac{x_j}{\|x\|_1}, \frac{x'_j}{\|x'\|_1}\right) \tag{12}$$

A path $s = (s_1, \cdots, s_m)$ of length $|s| = m$ in $\mathcal{T}_i$ is a sequence of m connected vertices $e_k = (s_k, s_{k+1}) \in \mathcal{E}_i$. We note $\mathcal{S}_i$ the set of all paths of $\mathcal{T}_i$, and define the following kernel between paths:

$$k_p(s, s') = \begin{cases} 0 & \text{if } |s| \neq |s'| \\ \frac{1}{m-1} \sum_{k=1}^{m-1} h(b(e_k), b(e'_k)) & \text{if } |s| = |s'| = m \end{cases} \tag{13}$$

Summing over all pairwise path comparisons yields a simple positive definite kernel between BOF-trees:

$$k_{all-paths}(\mathcal{T}_1, \mathcal{T}_2) = \frac{1}{|\mathcal{S}_1||\mathcal{S}_2|} \sum_{s \in \mathcal{S}_1} \sum_{s' \in \mathcal{S}_2} k_p(s, s') \tag{14}$$

This kernel, however, is expensive to compute in practice due to the large number of possible paths.

Instead, we propose the more efficient "All Tree Edge Pairs" (ATEP) kernel, which leverages the additive property of BOF-Trees. It consists in comparing the *edges* of BOF-trees, and can be seen as an approximation of the previous

kernel by considering only paths of length 1. Let $r_i \in \mathcal{V}_i$ be the root of $\mathcal{T}_i$, $i \in \{1, 2\}$. Our ATEP kernel is defined as:

$$k_{ATEP}(\mathcal{T}_1, \mathcal{T}_2) = w_r k_r(\mathcal{T}_1, \mathcal{T}_2) + (1 - w_r) k_e(\mathcal{T}_1, \mathcal{T}_2) \quad (15)$$

where

$$k_r(\mathcal{T}_1, \mathcal{T}_2) = h(b(r_1), b(r_2)) \quad (16)$$

is the intersection kernel between the roots of the trees and

$$k_e(\mathcal{T}_1, \mathcal{T}_2) = \frac{1}{|\mathcal{E}_1||\mathcal{E}_2|} \sum_{e \in \mathcal{E}_1} \sum_{e' \in \mathcal{E}_2} h(b(e), b(e')) \quad (17)$$

is the average of all pairwise edge comparisons. Note that the normalization by the number of edges makes ATEP unbiased with respect to the size of the trees.

As the roots have no parents, they are handled separately in this kernel: $w_r \in [0, 1]$ is a cross-validated parameter encoding a prior on the importance of the root-to-root comparisons. Note that the case $w_r = 1$ corresponds to the standard global BOF model with the intersection kernel.

This kernel relies on hierarchical relations: a node only depends on its parent. It can also be seen as a similarity between all sub-trees of two binary BOF-trees. Let a *direct family* $(v, s(v), p(v))$ denote, respectively, a non-root node $v \in \mathcal{V}_i \backslash \{r_i\}$, its only sibling $s(v)$, and its parent $p(v)$. The additive property of BOF-trees is formulated as $b(v) + b(s(v)) = b(p(v))$. Therefore, $(b(v), b(p(v)))$ completely characterizes a direct family. In addition, the BOF $b(v)$ is the sum of all the BOFs of its descendants. Consequently, $(b(v), b(p(v)))$ can be seen as an approximation of the content of the sub-tree rooted at $p(v)$, not just the path $p(v) \rightarrow v$. Therefore, the ATEP kernel efficiently compares all sub-trees by using only one level of hierarchy at a time to compare two motion components. Note that this theoretical interpretation is only valid for *binary* BOF-trees, whereas the previous one (approximation of order one of the all-paths kernel in eq. 14) is valid for any hierarchical decomposition and tree-structured representation.

Note that if the node kernel h is positive definite, then our ATEP kernel is also positive definite (it is a sum of positive definite kernels). We can, therefore, use it directly in conjunction with an SVM classifier [Schölkopf and Smola, 2002]. For multi-class classification, we adopt the One *v.s.* Rest approach.

4 Experiments

In this section, we report the results obtained with our approach, and compare it to both the state of the art and several baselines.

4.1 Datasets

We evaluate our method on four publicly available benchmarks — High Five [Patron-Perez et al, 2010], Olympic Sports [Niebles et al, 2010], Hollywood 2 [Marszalek et al, 2009], and HMDB [Kuehne et al, 2011] — that contain a total of 83 different action categories and 9639 real-world videos.

The **High Five** dataset [Patron-Perez et al, 2010] consists of 300 video clips, with an average duration of 90 frames, collected from 20 different TV shows. The activity categories are four human-human interactions: hand shakes, high fives, hugs, and kisses. Each activity is performed in 50 different clips, the remaining 100 "negative" clips containing other actions. The dataset authors propose a model using head orientations, and rely on manual annotations for discrete head orientation and upper body localization. We do not use this additional supervision in our experiments. Evaluation on this benchmark is conducted like in [Patron-Perez et al, 2010], *i.e.*, by computing the recognition performance in Average Precision (AP) using a 2-fold cross-validation with fixed folds provided with the dataset.

The **Olympic Sports** dataset [Niebles et al, 2010] contains 783 Youtube videos of athletes practicing 16 different sport activities such as springboard diving and weight lifting. This dataset contains longer videos (230 frames in average) of fast and complex articulated human motions, possibly involving interactions with objects (*e.g.*, a javelin). It presents several challenges, such as cluttered backgrounds, low quality videos, compression artifacts, and subtle distinctions between some categories (*e.g.*, triple jump and long jump). In addition, the activities are composed of multiple simpler actions (*e.g.*, running and jumping) that can be shared across categories, which is a challenge for part-based recognition. Different actions might, indeed, contain the same parts, but arranged differently. This justifies the need to leverage the spatio-temporal structure of activities in order to better distinguish them. In order to compare to the state of the art, we report both the mean accuracy (ACC) and AP on a fixed train/test split provided by the dataset authors.

The **Hollywood 2** [Marszalek et al, 2009] dataset consists of 1707 video clips — 823 for training, 884 for testing — extracted from 69 Hollywood movies. This benchmark contains particularly challenging video conditions due to large visual variability across movies. There are 12 categories: answering a phone, driving a car, eating, fighting, getting out of a car, hand shaking, hugging, kissing, running, sitting down, sitting up, and standing up. Some of these actions have a clear compositional nature (*e.g.*, kissing), which makes them amenable to decomposition, but other categories

are not as clearly structured (*e.g.*, fighting). Recognition performance is measured using mean Average Precision.

The **HMDB** dataset [Kuehne et al, 2011] is composed of 6849 clips from different real-world sources such as Youtube videos, TV footage, and movies. It contains 51 action categories that include general facial actions (*e.g.*, smile, laugh, talk), facial actions with object manipulation (*e.g.*, smoke, drink), full body movements (*e.g.*, cartwheel, climb, walk, wave), interactions with objects (*e.g.*, dribble, golf), and interactions between persons (*e.g.*, hug, kiss, punch). Each action is performed in at least 101 clips. Performance is evaluated by the mean accuracy over all classes, averaged over three fixed different train/test splits provided by the authors. HMDB is a challenging dataset, which includes poor video conditions (only 17% of high quality clips), partially visible actors (in 44% of the videos), camera motions, and a large variety of viewpoints. In addition, another difficulty is caused by the presence of close pairs of classes such as chewing *v.s.* talking, shooting a gun *v.s.* shooting a bow, fencing *v.s.* sword exercise. HMDB actions — some of which are parts of the more complex Olympic Sports activities — tend to be both atomic and short with 90 frames per action on average, but are numerous and varied.

4.2 Camera motion compensation

Although camera motion compensation is not necessary with our approach, it allows to remove the camera motion bias of a dataset. Some video sources — and sports videos in particular — are characterized by large camera motions strongly correlated with the action filmed. For instance, in the compensated frame of Figure 6 (bottom row), we can see the rapid downward tilt of the camera as it follows the landing of the jumper. In addition, amateur videos — *e.g.*, those posted on YouTube, some of which are included in the Olympic Sports and HMDB datasets — are often shot with handheld devices with significant motion due to camera shaking. When using dense tracklets, camera motion is directly encoded by the features. Furthermore, the background tracklets will share similar motion statistics, as they mostly correspond to the movement of the camera.

We, therefore, report results on videos where camera motion was compensated for the Olympic Sports and HMDB datasets. This ensures that our evaluation is less influenced by (positive or negative) camera motion biases, and only quantifies the efficiency of our action model. Note that the other datasets we use in our experiments are less prone to such camera motion biases. High Five and Hollywood 2, indeed, contain high-quality movies or TV videos, with less camera motion and shots focusing on the actors (*i.e.*, with less pixels on the background). We, therefore, present results on the original non-compensated video sequences for these benchmarks.

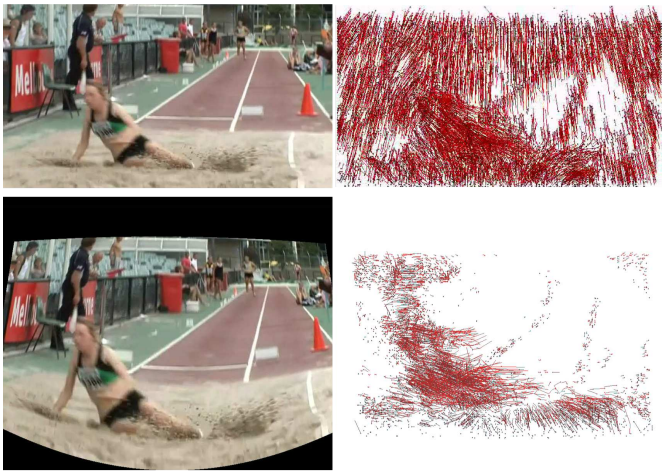

Fig. 6: *Tracklets from a video (top) and its compensated version.*

Camera motion estimation [Szeliski, 2010] in uncontrolled video conditions is a difficult task. Although efficient approaches exist for simple transformations, *e.g.*, phase correlation [De Castro and Morandi, 1987] for translations, only few address all kinds of complex camera motions, such as camera shake in amateur videos, or out-of-plane rotations coupled with zooms in movies. Here, we explicitly model the camera motion to generate compensated videos as in [Ikizler-Cinbis and Sclaroff, 2010, Kuehne et al, 2011]. We use simple and efficient image-stitching techniques to compute the approximate motion of a background plane. First, we extract salient 2D Harris interest points in frames which we represent with SIFT descriptors [Lowe, 2004]. We, then, robustly match the interest points using RANSAC [Szeliski, 2010] and estimate the transformation between all pairs of adjacent frames. Finally, we produce a compensated video where all frames are warped according to the estimated transformation. We then compute the optical flow in these videos, which corresponds mostly to the flow induced by the moving objects (*cf.* Figure 6), as the background flow (induced by the camera motion) is canceled by simple thresholding in the compensated video. This, then, allows us to extract compensated tracklets and their descriptors. Note that the trajectories of points on the warped frame boundaries are filtered out and not included in our models. In our experiments, we found that this motion compensation yielded a significant improvement in general (+5% accuracy for the "BOF tracklets" baseline on Olympic Sports and HMDB).

4.3 Baselines

We first compare to two simple baselines using a global BOF model over the entire video [Laptev et al, 2008, Wang et al, 2013]. The approach of Laptev et al [2008] uses sparse lo-

cal Spatio-Temporal Interest Points (STIPS) [Laptev, 2005] described by a concatenation of histogram of oriented gradients and optical flow, denoted "HOG/HOF". The approach of Wang et al [2013] uses dense tracklets represented by MBH descriptors. Note that this method corresponds to using only the model of the root node of our BOF-trees. In both cases, we use our own implementation of the method with the same vocabulary construction and size as mentioned in Section 3.2. In addition to these unstructured baselines, we compare our approach with decompositions obtained by alternative clustering methods. First, we compare to two standard "flat" clustering algorithms that produce a set of unrelated clusters: k-means and spectral clustering. Due to the large amount of tracklets per video, we adopt an efficient on-line variant of k-means [Sculley, 2010]. For spectral clustering, we adopt the approach of Fowlkes et al [2004]: we partition tracklets with (on-line) k-means on the same approximate spectral embedding used by our method. These two algorithms take as input a fixed number of clusters. However, different videos contain different numbers of motion components. Therefore, we use a number of clusters that depends linearly on the number of tracklets, such that the smallest videos have at least 2 clusters, whereas the largest ones have at most 1000 clusters.

We also compare our approach to a closely related baseline yielding a cluster-tree. Its steps are all similar to our method except for one: we replace the spectral divisive thresholding algorithm of Section 2.4 (noted SDT) by a recursive application of k-means. This amounts to splitting a node with (on-line) k-means ($k = 2$) on the spectral embedding. The algorithm is noted "SDKM" for Spectral Divisive K-Means. We also add another related baseline where we apply k-means recursively in the same manner, but directly on the concatenation of the features, *i.e.*, without spectral embedding in order to determine its contribution. This baseline is noted "BPKM" for bi-partitioning k-means. Like with our approach, we classify the resulting cluster-trees using an SVM with our ATEP kernel. Note that our thresholding algorithm is more computationally efficient than these alternatives based on k-means, as they additionally require to compute distances to centroids at each iteration.

In order to compare unrelated sets of clusters, we use a natural simplification of our ATEP kernel consisting in averaging all pairwise cluster comparisons between two videos. This baseline kernel is noted as "ACP" for "All Cluster Pairs". It can be applied on the clusters resulting from the "flat" clustering algorithms, the leaves of a BOF-tree, or on all nodes of a BOF-tree (including internal ones) in which case the hierarchical relations are ignored.

Table 1 contains the notations used to describe the methods and kernels compared in our experiments.

Tree-structured activity models	
SDT tree	BOF-Tree (Section 3.2) from Spectral Divisive Thresholding on tracklets (Section 2.4)
SDKM tree	BOF-Tree from Spectral Divisive K-Means on tracklets (Section 4.3)
BPKM tree	BOF-Tree from Bi-Partitioning K-Means on tracklets (Section 4.3)
Flat sets of bag-of-features	
SDT leaves	leaves of the SDT BOF-Trees
SDKM leaves	leaves of the SDKM BOF-Trees
spectral	clusters obtained by spectral clustering
kmeans	clusters obtained by k-means
Unstructured baselines	
BOF tracklets	BOF of tracklets (roots of BOF-trees)
BOF STIPS	BOF of spatio-temporal interest points
Kernels on part-based activity models	
ATEP	All Tree Edge Pairs kernel, mean of all pairwise edge comparisons, specific to BOF-Trees
ACP	All Cluster Pairs kernel, mean of all pairwise cluster comparisons

Table 1: *The methods we implemented and compared.*

	HighFive (AP)	Olympics (ACC)	Hollywood2 (AP)	HMDB (ACC)
SDT tree ATEP	62.4	**85.0**	**54.4**	**41.3**
SDKM tree ATEP	**64.1**	77.8	54.3	39.3
BPKM tree ATEP	62.9	77.9	52.6	39.4
SDT tree ACP	55.9	79.3	47.6	33.9
SDKM tree ACP	57.0	70.6	47.4	34.1
BPKM tree ACP	55.4	72.9	47.2	37.0
SDT leaves ACP	57.7	77.9	47.7	31.6
SDKM leaves ACP	58.1	72.2	46.1	33.6
BPKM leaves ACP	56.1	71.1	46.7	35.5
spectral ACP	58.2	71.7	47.9	30.5
kmeans ACP	56.4	70.8	45.4	34.4
BOF tracklets	56.0	76.6	52.4	38.7
BOF STIPS	36.9	61.3	45.0	26.9

Table 2: *Comparison of our method with different baselines.*

4.4 Recognition results

Table 2 compares our method to the aforementioned baselines. Table 3 compares our method with the state of the art. Additionally, Figure 7 and Figure 8 contain per-class details on the Olympic Sports and High Five datasets. "AP" stands for Average Precision and "ACC" for Accuracy, both averaged over all categories.

First, our approach significantly improves over the unstructured BOF baselines: $+4.9\%$ on average, with a root weight w_r (prior importance of global BOF / the root of the BOF-Tree, *cf.* eq. 15) of 45% on average over all actions.

	HighFive (AP)	Olympics (ACC)	Olympics (AP)	Hollywood2 (AP)	HMDB (ACC)
SDT tree ATEP	**62.4**	**85.0**	**85.5**	54.4	41.3
Wang et al [2013]	-	-	77.2	59.9	**48.3**
Jiang et al [2012a]	-	-	80.6	59.5	40.7
Vig et al [2012]	-	-	-	**60.0**	-
Sapienza et al [2012]	-	-	-	43.9	30.5
Todorovic [2012]	-	82.9	-	-	-
Kliper-Gross et al [2012]	-	-	-	-	29.2
Yu et al [2012]	56.0	-	-	-	-
Tang et al [2012]	-	-	66.8	-	-
Sadanand and Corso [2012]	-	-	-	-	26.9
Liu et al [2011]	-	65.1	74.4	-	-
Brendel and Todorovic [2011]	-	77.3	-	-	-
Kuehne et al [2011]	-	-	-	-	23.2
Patron-Perez et al [2010]	32.8	-	-	-	-

Table 3: *Comparison of our method with the state of the art.*

This confirms the importance of leveraging structure information to recognize complex activities. As visible in Figure 9, the improvement is particularly noticeable for complex sports activities involving different phases (*e.g.*, in "triple-jump", "vault", "snatch", "discusthrow", which all have a cross-validated root weight of 0, meaning that only the hierarchical relations are used, not the root of the BOF-Tree), for interactions between multiple actors (*e.g.*, "kick", "hug", "handshake"), and for articulated combinations of different motions (*e.g.*, "bowling", "sword", "cartwheel").

Our method fails to improve over the BOF baseline in two main scenarios: (i) for simple actions characterized by a single fast or fine-grained motion — *e.g.*, when answering a phone, when throwing a javelin, or when diving — that is often represented by a single cluster (these categories have a root weight of 100% in our experiments, meaning that ATEP is equivalent to the BOF tracklets baseline in these cases), and (ii) for unorganized collections of motion — *e.g.*, for the "fightperson" category — that do not have consistent relations between motion components. Note that most classes with a low performance improvement do not use only the root ($w_r < 100\%$), which indicates that only a few videos in these categories have a helpful decomposition.

Our results also point out that movies are the most challenging video source for our approach. Indeed, the increase in performance over BOF — although significant in some cases, *cf.* Figure 10 for precision-recall curves — is the smallest on Hollywood 2 categories. This is caused by background clutter and editing effects introducing noisy motion components that can be consistently observed across videos, irrespectively of the action performed. This indicates that our method would benefit from other scene interpretation techniques, *e.g.*, shot, actor, and object detectors.

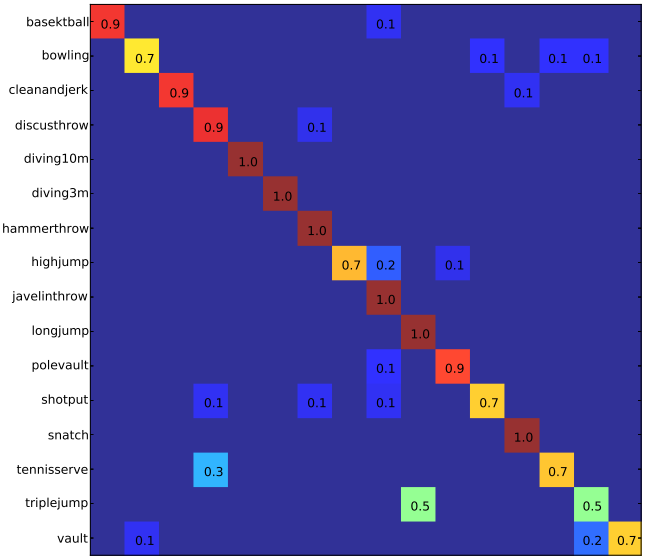

Fig. 7: *Confusion matrix for STD Tree ATEP on Olympic Sports.*

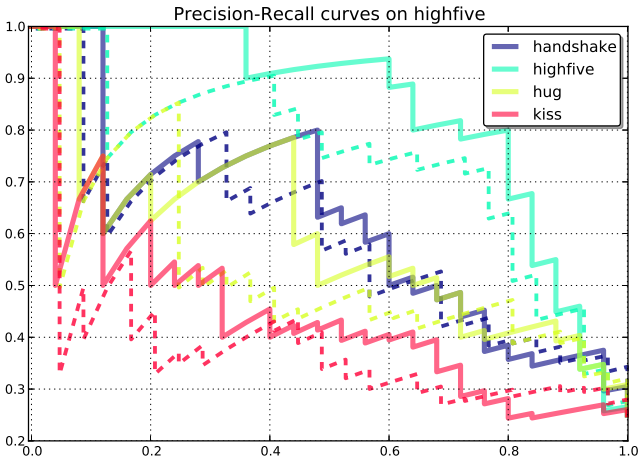

Fig. 8: *Precision - Recall curves for SDT Tree ATEP on the High Five activities. Dashed lines correspond to the "BOF tracklets" baseline.*

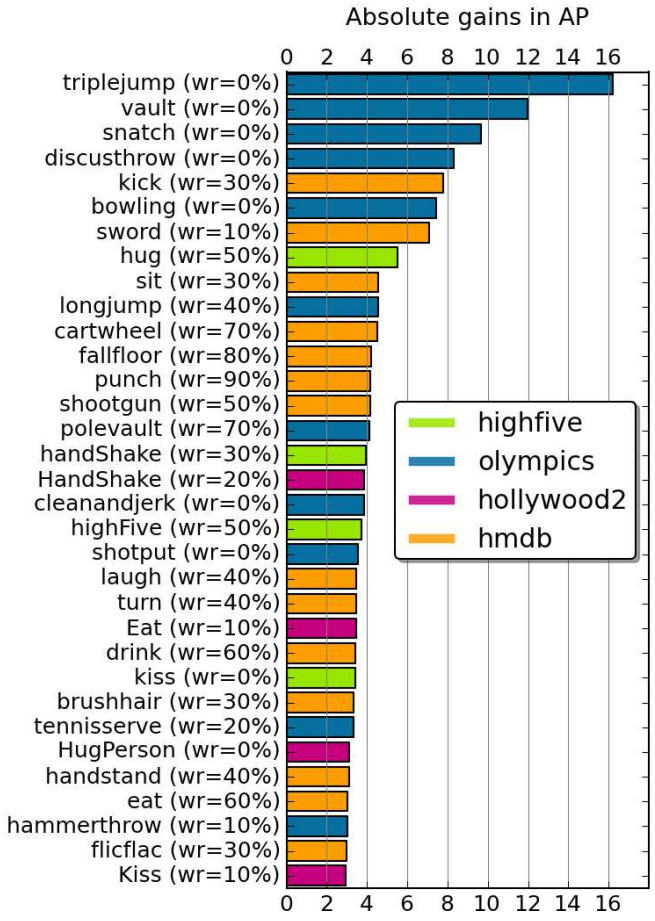

Fig. 9: *Largest per-category performance gains of our "SDT tree ATEP" method over the "BOF tracklets" baseline, where "wr" denotes the cross-validated root weight used in ATEP (cf. eq. 15).*

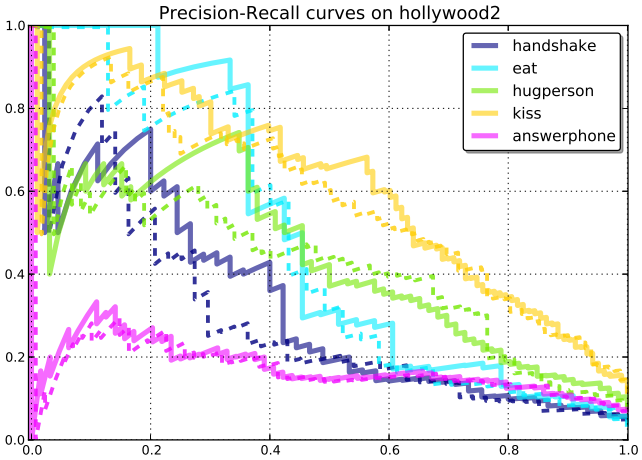

Fig. 10: *Precision - Recall curves for the four best Hollywood 2 categories and the worst one ("answerphone") with our method. Dashed lines correspond to the "BOF tracklets" baseline.*

Second, only our method yields clear and consistent performance improvements, whereas other structured baselines are in general less accurate (except for SDKM and BPKM trees with ATEP on the High Five activities). This highlights that automatically decomposing activities is challenging, and that modeling an incorrect structure may degrade performance, even below that of unstructured methods. For instance, models based on simple k-means clusters perform on average -4.1% lower than the BOF tracklets baseline. Our results, therefore, seem to indicate that our spectral divisive thresholding algorithm, coupled with its connectedness criterion, can alleviate the issue of picking a fixed set of good clusters a priori by instead proposing a hierarchical decomposition. Note also that our experiments confirm that using an approximated spectral embedding has two benefits: first it allows for an efficient and theoretically-motivated bipartitioning strategy based on thresholding one-dimensional vectors, second it yields more discriminative hierarchical clustering results ($+2.6\%$ *w.r.t.* BPKM tree + ATEP).

Third, using hierarchical relations between motion components with our ATEP kernel seems to be the main factor explaining our performance gains. Indeed, using our ATEP kernel consistently improves over its unstructured counterpart (ACP) applied on the leaves only ($+7\%$ on average for SDT), as well as on the full BOF-trees ($+6.6\%$ on average for SDT), *cf.* Figure 11 for per-class details on Olympic Sports. This confirms that our method captures useful structure information, and that the improvement does not result solely from the decomposition into parts, but first and foremost from the comparison of their relations.

Finally, we found that our hierarchical ATEP kernel on SDT BOF-trees in conjunction with an SVM performs comparably to or better than the state of the art (*cf.* Table 3), including (i) *latent part models* ($+23.0\%$ AP *w.r.t.* [Niebles et al, 2010], $+19.9\%$ ACC *w.r.t.* [Liu et al, 2011], $+18.7\%$ AP *w.r.t.* [Tang et al, 2012]), (ii) *complex graphical models* ($+7.7\%$ ACC *w.r.t.* [Brendel and Todorovic, 2011], who use video segmentation, and $+2.1\%$ ACC *w.r.t.* [Todorovic, 2012], who use additional localization annotations at training time), and (iii) interaction-specific *structured learning* ($+29.6\%$ AP *w.r.t.* [Patron-Perez et al, 2010]). Our method is outperformed on the Hollywood 2 dataset by [Vig et al, 2012], who use BOF with saliency masks, [Jiang et al, 2012a], who combine multiple features, and [Wang et al, 2013], who also combine multiple trajectory features and uses spatio-temporal pyramids. Note that our method would also directly benefit from these enhancements of the BOF model.

4.5 Cluster-tree statistics

Figure 12 presents some statistics for our trees over all datasets. We can observe that most of the trees we obtain have less than 5,000 nodes, 3,000 leaves, and a depth of 50. Moreover,

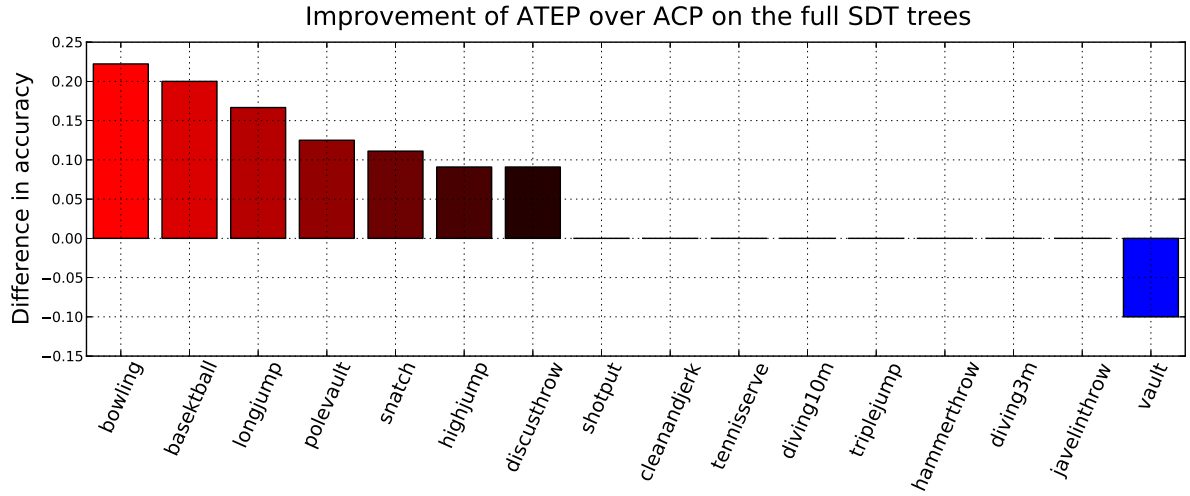

Fig. 11: *Performance improvement obtained by using the hierarchical relations of the SDT BOF-trees on the Olympic Sports dataset.*

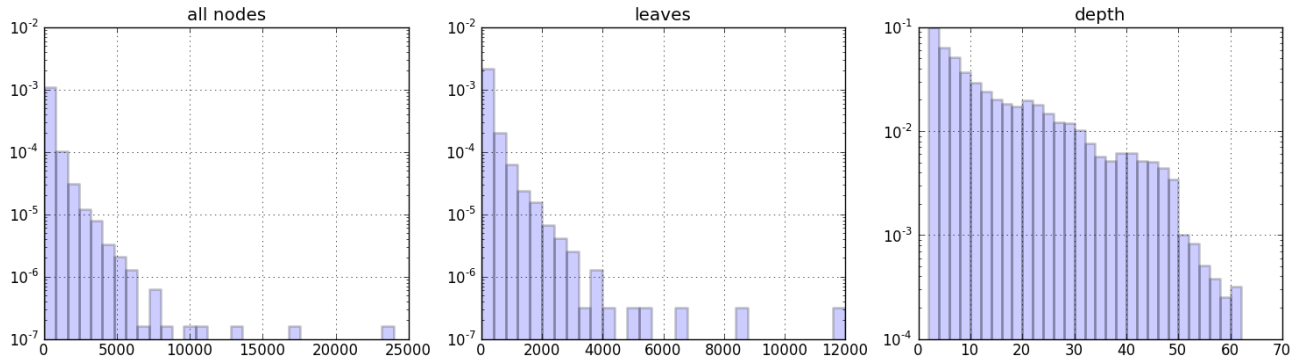

Fig. 12: *Histograms of tree sizes, number of leaves, and depths of the trees obtained with our clustering algorithm over all datasets. The median tree is of depth 9, with 33 nodes and 17 leaves, whereas the average is 13 / 324 / 162, meaning that a few trees are much larger than the others.*

many trees are small: the median number of nodes is 33 and the median depth is 9. These decompositions correspond to atomic actions, short videos, or activities with few different motions (*e.g.*, a single whole body translation when diving). Note also that a few trees are much larger and correspond to long, cluttered videos involving many different motions (*e.g.*, some fighting scenes from the Hollywood2 dataset).

Figure 13 visualizes the leaves of the trees for some videos. It shows that the leaves often correspond to noisy clusters, over-segmenting the motion of the scene. This intuitively confirms the benefit of using a hierarchical clustering approach, as finding the optimal level of granularity in the motion decomposition is difficult and task-dependent.

5 Discussion

We introduced an efficient hierarchical clustering algorithm on short duration trajectories (tracklets). Our approach relies on recursive bi-partitioning using a spatio-temporal connectedness criterion to threshold the projection of tracklets on a multi-modal spectral embedding obtained with the Nyström approximation. A video is structured as a tree of nested motion components. Each one of these data-driven parts is represented by a bag-of-features over local motion descriptors in order to form our BOF-Tree activity model. We also proposed a kernel on unordered binary trees that can accurately compare activities with a variable number of hierarchically ordered parts. This kernel uses an additive property of our motion decomposition to efficiently compare sub-trees, and leverages structure information to refine part comparisons.

Our experiments show that the combination of our clustering algorithm and our tree kernel outperforms unstructured bag-of-features baselines, video decompositions with other clustering techniques, and state of the art approaches based on latent parts, video segmentation, and structured learning. Furthermore, we observed that our method can be successfully applied to short actions as long as they have a characteristic motion decomposition structure. We observed, however, that the performance improvements with our method are more significant on complex activities involving the spatio-temporal composition of several short actions.

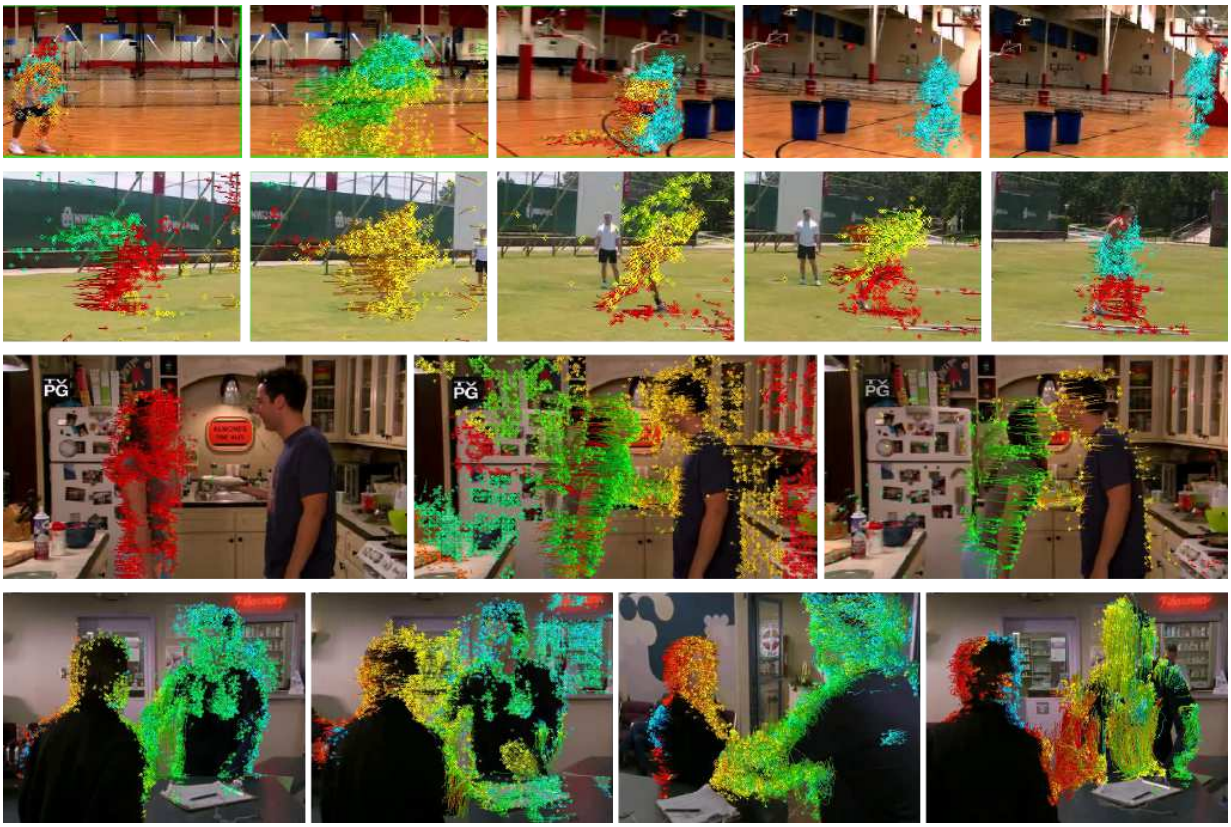

Fig. 13: *Example videos with leaf clusters from our cluster-trees.*

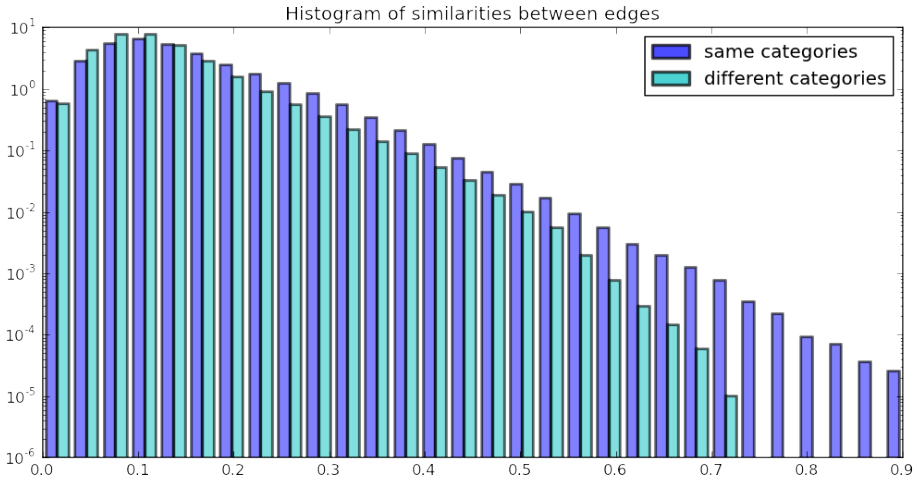

Fig. 14: *Normalized histograms of similarities between edges of different trees from all datasets, either from activities of the same category, or from different categories. Note that the y-axis is in log-scale.*

Our ATEP kernel effectively differs from a matching or alignment metric, as it leverages all pairwise edge similarities, which are generally spread over the whole trees. Figure 14 indeed shows that the most similar edges (similarity above 0.7) of trees from the same categories account for only a small fraction (approximately 0.1%) of all the intra-class edge similarities (note that the y-axis is in log-scale). Figure 14 also shows that there are more moderately similar edges (similarity above 0.2) between videos of the same categories than of different categories. These two factors suggest that most of the useful similarities are between the vast amount of moderately similar edges. Therefore, as mentioned previously, comparing trees using all the edges allows our approach to be robust to real-world conditions and imperfect decompositions.

Nevertheless, Figure 14 indicates that actions of the same category share more highly similar edges than actions of different categories. Therefore, a possible extension of our approach consists in adapting it to spatio-temporal activity localization by searching for high-scoring edges or sub-trees.

Acknowledgements This work was partially funded by the MSR/INRIA joint project, the European integrated project AXES, the PASCAL 2 Network of Excellence, the Gargantua project under program Mastodons of CNRS, the LabEx PERSYVAL-Lab (ANR-11-LABX-0025), and the ERC advanced grant ALLEGRO.

References

Bilen H, Namboodiri VP, Van Gool LJ (2011) Object and Action Classification with Latent Variables. In: BMVC
Bradski G, Kaehler A (2008) Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media
Brendel W, Todorovic S (2011) Learning spatiotemporal graphs of human activities. In: ICCV
Brox T, Malik J (2010) Object segmentation by long term analysis of point trajectories. In: ECCV
Dalal N, Triggs B, Schmid C (2006) Human Detection Using Oriented Histograms of Flow and Appearance. In: ECCV
De Castro E, Morandi C (1987) Registration of translated and rotated images using finite Fourier transforms. PAMI
Diestel R (2005) Graph theory. Springer - Verlag
Duda R, Hart P, Stork D (2001) Pattern classification
Dupé F, Brun L (2008) Hierarchical bag of paths for kernel based shape classification. SSSPR
Farnebäck G (2003) Two-frame motion estimation based on polynomial expansion. Image Analysis
Felzenszwalb PF, Girshick RB, McAllester D, Ramanan D (2010) Object detection with discriminatively trained part based models. PAMI
Foster L, Waagen A, Aijaz N, Hurley M, Luis A, Rinsky J, Satyavolu C, Way MJ, Gazis P, Srivastava A (2009) Stable and efficient gaussian process calculations. JMLR
Fowlkes C, Belongie S, Chung F, Malik J (2004) Spectral grouping using the Nystrom method. PAMI
Fradet M, Robert P, Pérez P (2009) Clustering point trajectories with various life-spans. In: CVMP
Gaidon A, Harchaoui Z, Schmid C (2011) Actom Sequence Models for Efficient Action Detection. In: CVPR
Gaidon A, Harchaoui Z, Schmid C (2012) Recognizing activities with cluster-trees of tracklets. In: BMVC
Gilbert A, Illingworth J, Bowden R (2010) Action recognition using mined hierarchical compound features. PAMI
Grundmann M, Meier F, Essa I (2008) 3D shape context and distance transform for action recognition. In: ICPR
Hastie T, Tibshirani R, Friedman J (2008) The elements of statistical learning (2nd edition). Springer
Hongeng S, Nevatia R (2003) Large-scale event detection using semi-hidden markov models. In: ICCV
Ikizler-Cinbis N, Sclaroff S (2010) Object, scene and actions: Combining multiple features for human action recognition. In: ECCV
Jiang Y, Dai Q, Xue X, Liu W, Ngo C (2012a) Trajectory-Based Modeling of Human Actions with Motion Reference Points. In: ECCV
Jiang Z, Lin Z, Davis L (2012b) Recognizing Human Actions by Learning and Matching Shape-Motion Prototype Trees. PAMI
Kliper-Gross O, Gurovich Y, Hassner T, Wolf L (2012) Motion Interchange Patterns for Action Recognition in Unconstrained Videos. In: ECCV
Kovashka A, Grauman K (2010) Learning a hierarchy of discriminative space-time neighborhood features for human action recognition. In: CVPR
Kuehne H, Jhuang H, Garrote E, Poggio T, Serre T (2011) HMDB: a large video database for human motion recognition. In: ICCV
Laptev I (2005) On space-time interest points. IJCV
Laptev I, Marszalek M, Schmid C, Rozenfeld B (2008) Learning realistic human actions from movies. In: CVPR
Laxton B, Lim J, Kriegman D (2007) Leveraging temporal, contextual and ordering constraints for recognizing complex activities in video. In: CVPR
Lezama J, Alahari K, Sivic J, Laptev I (2011) Track to the future: Spatio-temporal video segmentation with long-range motion cues. In: CVPR
Liu J, Kuipers B, Savarese S (2011) Recognizing human actions by attributes. In: CVPR
Lowe DG (2004) Distinctive image features from scale-invariant keypoints. IJCV
Maji S, Berg A, Malik J (2008) Classification Using Intersection Kernel Support Vector Machines is efficient. In: CVPR
Marszalek M, Laptev I, Schmid C (2009) Actions in context. In: CVPR
Matikainen P, Hebert M, Sukthankar R (2010) Representing pairwise spatial and temporal relations for action recognition. ECCV
Mikolajczyk K, Uemura H (2008) Action recognition with motion-appearance vocabulary forest. In: CVPR
Niebles JC, Fei-Fei L (2007) Hierarchical model of shape and appearance for human action classification. In: CVPR
Niebles JC, Chen C, Fei-Fei L (2010) Modeling Temporal Structure of Decomposable Motion Segments for Activity Classification. In: ECCV
Oliver NM, Rosario B, Pentland AP (2000) A Bayesian computer vision system for modeling human interactions. PAMI
Pablo A, Maire M, Fowlkes C, Malik J (2011) Contour detection and hierarchical image segmentation. PAMI
Patron-Perez A, Marszalek M, Zisserman A, Reid ID (2010) High Five: Recognising Human Interactions in TV Shows. In: BMVC
Prest A, Ferrari V, Schmid C (2012) Explicit modeling of human-object interactions in realistic videos. PAMI
Raptis M, Kokkinos I, Soatto S (2012) Discovering Discriminative Action Parts from Mid-Level Video Representations. In: CVPR
Reddy KK, Liu J, Shah M (2009) Incremental action recognition using feature-tree. In: CVPR
Sadanand S, Corso JJ (2012) Action Bank: A High-Level Representation of Activity in Video. In: CVPR
Sapienza M, Cuzzolin F, Torr P (2012) Learning discriminative space-time actions from weakly labelled videos. In: BMVC
Schölkopf B, Smola AJ (2002) Learning with Kernels
Schölkopf B, Smola A, Müller KR (1998) Nonlinear component analysis as a kernel eigenvalue problem. Neural computation
Sculley D (2010) Web-scale k-means clustering. In: WWW
Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge Univ Pr
Shi J, Malik J (1998) Motion segmentation and tracking using normalized cuts. In: ICCV, IEEE
Shi J, Malik J (2000) Normalized cuts and image segmentation. PAMI
Shi J, Tomasi C (1994) Good features to track. In: CVPR
Suard F, Rakotomamonjy A, Bensrhair A (2007) Kernel on bag of paths for measuring similarity of shapes. In: European Symposium on Artificial Neural Networks, pp 1–6
Szeliski R (2010) Computer vision: algorithms and applications. Springer-Verlag
Tang K, Fei-Fei L, Koller D (2012) Learning latent temporal structure for complex event detection. In: CVPR
Todorovic S (2012) Human activities as stochastic kronecker graphs. In: ECCV

Vig E, Dorr M, Cox D (2012) Space-variant descriptor sampling for action recognition based on saliency and eye movements. In: ECCV
Wang H, Kläser A, Schmid C, Cheng-Lin L (2013) Dense trajectories and motion boundary descriptors for action recognition. IJCV
Wang Y, Mori G (2011) Hidden part models for human action recognition: Probabilistic vs. max-margin. PAMI
Williams C, Seeger M (2001) Using the Nyström method to speed up kernel machines. In: NIPS
Yu G, Yuan J, Liu Z (2012) Propagative Hough Voting for Human Activity Recognition. In: ECCV