A Scalable and Accurate Descriptor for Dynamic Textures using Bag of System Trees

Adeel Mumtaz, Emanuele Coviello, Gert. R. G. Lanckriet, Antoni B. Chan

Abstract—The bag-of-systems (BoS) representation is a descriptor of motion in a video, where dynamic texture (DT) codewords represent the typical motion patterns in spatio-temporal patches extracted from the video. The efficacy of the BoS descriptor depends on the richness of the codebook, which depends on the number of codewords in the codebook. However, for even modest sized codebooks, mapping videos onto the codebook results in a heavy computational load. In this paper we propose the BoS Tree, which constructs a bottom-up hierarchy of codewords that enables efficient mapping of videos to the BoS codebook. By leveraging the tree structure to efficiently index the codewords, the BoS Tree allows for fast look-ups in the codebook and enables the practical use of larger, richer codebooks. We demonstrate the effectiveness of BoS Trees on classification of four video datasets, as well as on annotation of a video dataset and a music dataset. Finally, we show that, although the fast look-ups of BoS Tree result in different descriptors than BoS for the same video, the overall distance (and kernel) matrices are highly correlated resulting in similar classification performance.

Index Terms—Dynamic Textures, Bag of Systems, Video Annotation, Music Annotation, Dynamic Texture Recognition, Efficient Indexing, Large Codebooks.

1 Introduction

The bag-of-systems (BoS) representation [1], a *high-level* descriptor of motion in a video, has seen promising results in video texture classification [2, 3, 4]. The BoS representation of videos is analogous to the bag-of-words representation of text documents, where documents are represented by counting the occurrences of each word, or the bag-of-visual-words representation of images, where images are represented by counting the occurrences of visual codewords in the image. Specifically, in the BoS framework the codebook is formed by generative time-series models (in particular, linear dynamical *systems* or dynamic textures [5]) instead of words, each of them compactly characterizing typical textures and dynamics patterns of pixels or low-level features in a spatio-temporal patch. Hence, each video is represented by a BoS histogram with respect to the codebook, by assigning individual spatio-temporal patches to the most likely codeword, and then counting the frequency with which each codeword is selected. An advantage of the BoS approach is that it decouples modeling content from modeling classes. As a consequence, a codebook of sophisticated generative models can be robustly compiled from a large collection of videos, while simpler models, based on standard text mining algorithms, are used to capture statistical regularities in the BoS histograms representing the subsets of videos associated to each individual class.

The BoS representation was originally proposed for video texture classification [1], where the dynamic texture (DT) codewords quantize prototypical patterns in spatio-temporal cubes corresponding to interest points in videos, and was proven superior to standard methods based on modeling each video with a single DT model [6]. The BoS representation is not limited to video, but is also applicable as a descriptor to any type of time-series data. In particular, the BoS framework has also proven highly promising in automatic *music* annotation and retrieval [7], registering significant improvements with respect to current state of the art systems.

In practice, the efficacy of the BoS descriptor (or any bag-of-words representation) depends on the richness of the codebook, i.e., the ability to effectively quantize the feature space, which directly depends on both the method of learning codewords from training data, and the number of codewords in the codebook. For the former, the learning of good codewords is addressed in [2] by using a hierarchical EM algorithm. For the latter, increasing the number of codewords also increases the computational cost of mapping a video onto the codebook; indeed, the computational complexity is linear in the number of codewords. For the standard bag-of-visual-words, increasing the number of codewords is typically not a problem, since the simple L2-distance function is used to identify the visual codeword closest to an image patch. On the other hand, for the BoS in [2], finding the closest codewords to a video patch requires calculating the likelihood of a video patch under each DT codeword using the Kalman filter. For even modest sized codebooks, this results in a heavy computational load. For example, the BoS codebooks of [2] are limited to only 8 codewords and [3] uses a maximum of 64 codewords.

In order to handle an extremely large bag-of-visual-words codebook (e.g., 10^6 codewords), Nister and Stewenius [8] proposed a tree-structured vector quantizer (TSVQ) for Euclidean vectors, which creates a hierarchical quantization of the feature space for efficient indexing of the vector codewords. Inspired by this idea, to address the computational challenges of the BoS representation, in this paper we propose the *BoS Tree*,

- *A. Mumtaz and A. B. Chan (corresponding author) are with the Department of Computer Science, City University of Hong Kong. E-mail: adeelmumtaz@gmail.com, abchan@cityu.edu.hk.*
- *E. Coviello and G. R. G. Lanckriet are with the Department of Electrical and Computer Engineering, University of California, San Diego. E-mail: emanuetre@gmail.com, gert@ece.ucsd.edu.*

which combines the expressiveness of a large BoS codebook with the efficiency of a small BoS codebook. Our proposed approach constructs a bottom-up hierarchy of codewords, and then leverages the tree structure to efficiently index the codewords by choosing only the most-likely branches when traversing the tree. In this way, *the proposed BoS Tree allows for fast look-ups on the codebook and consequently enables the practical use of a larger BoS codebook*. The novelty of our approach is that we extend tree-structured search to a codebook consisting of time-series models (i.e., DT models), instead of to a VQ codebook of Euclidean vectors. Although a VQ codebook could be extended to video patches, e.g., by concatenating all frames into a single vector, the resulting image codewords would not handle spatio-temporal variations well, and would be extremely high dimensional.

The contributions of this paper are three-fold. First, we propose the BoS Tree for fast-indexing of large BoS codebooks. Second, we experiment with the BoS Tree on a variety of applications, including video annotation, music annotation and retrieval, and video texture classification, and demonstrate that BoS Tree reduces the computational cost by at least one order of magnitude versus a standard large codebook, while achieving similar performance. Finally, we perform a detailed analysis of the BoS Tree projections and present evidence for the obtained results by comparing BoS Tree and BoS histograms.

The remainder of this paper is organized as follows. We discuss related work in Section 2. In Section 3, we review the BoS representation and DT model. Next we propose the BoS Tree in Section 4. After that in Section 5, we present three applications of the BoS Tree with experimental evaluations. Finally in Section 6, we present a detailed analysis of the BoS Tree descriptor as compared to that of the BoS.

2 Related Work

Most state of the art methods for dynamic texture recognition are based on either DT models [9, 10, 11, 1] or aggregations of local descriptors [12, 13]. At first [9, 10] represented each video as a DT model and then performed classification by either nearest neighbors or support vector machine (SVM), by adopting an appropriate distance (dissimilarity) measure between dynamic textures, *e.g.*., Martin distance [9] or Kullback-Leibler divergence [10]. The resulting descriptors suffers the drawback of only modeling a particular viewpoint of each texture because they work on the *holistic or global appearance* of the video, i.e., the video frame as a whole. In order to address this issue, subsequent methods addressed the problems of translation-invariance and view-point variation: [11] proposed a new distance (dissimilarity) measure between DTs, which is based only on the spectrum or cepstrum of the hidden-state process and ignores the appearance component of the model; [1] proposes a *bag-of-systems* (BoS) representation in which spatio-temporal patches are extracted from videos using interest-point operators, and then assigned to DT codewords. Experimental results in [1] show that the patch-based framework of the BoS is more adaptive to changes in viewpoint, as compared to approaches based on modeling the holistic appearance of a video with a DT [9, 10]. The BoS Tree studied in this paper is an efficient method for computing a BoS descriptor when the number of codewords is large.

The bag-of-features "cousin" of our BoS Tree is the tree-structured vector quantizer (TSVQ) [14], which creates a hierarchical quantization of a feature space, and was proposed by Nister *et al.* for efficiently indexing a large vocabulary of image codewords [8], and by Grauman *et al.* to define the bins of multi-resolution histograms [15]. The difference between existing methods and our proposed approach is that the BoS Tree extends tree-structured search to a codebook formed by time-series models (i.e., DT models), which are well suited to handle spatio-temporal variations in videos. In practice, this allows for an efficient deployment of a BoS codebook with a large number of codewords.

Efficient indexing of codewords is also related to fast approximate nearest-neighbor (NN) search. Typical approaches to fast NN for real-vectors also exploit a tree data structure, e.g., KD-trees and metric ball trees, and use branch and bound methods to traverse the tree to find the nearest neighbor [16, 17]. Cayton [18] generalizes metric ball trees to Bregman divergences, enabling fast NN search of histograms using the KL divergence. Alternatively, approximate search can be performed efficiently using randomized trees [19], by building a forest of KD trees [20] and randomly selecting splitting dimensions with high variance, or using locality-sensitive hashing (LSH) [21], which maps similar items into the same hashing bucket with high probability.

The BoS Tree proposed here is similar to the Bregman-ball tree in [18], in that both use KL divergence-based clustering to hierarchically construct a tree. The main differences are that our BoS Tree is based on probability distributions (in fact random processes) with hidden states, while [18] is limited to only exponential family distributions, and that our nearest-neighbor search is based on data likelihood, not KL divergence. In addition, we use a simple forward search to traverse the tree, whereas [18] uses a more complicated branch and bound method. Experimentally, we found that the forward search was both efficient and produced satisfactory BoS descriptors. The approximate branch and bound method proposed in [22] is applicable to a BoS Tree but is limited to NN search based on KL divergence. Finally, our BoS Tree is also related to fast image retrieval work by [23], where each image is modeled as a Gaussian distribution and a retrieval tree is constructed using the HEM algorithm for Gaussian models. In contrast to [23], our work retrieves time-series models (linear dynamical systems) instead of Gaussians, and calculates similarities using log-likelihood scores.

In contrast to using DT models, several approaches perform DT recognition using aggregations of local descriptors. [13] uses a combination of local binary pattern (LBP) histograms extracted from three orthogonal planes in space-time (XY, XT, YT), while [12] uses distributions of local space-time oriented structures. Although these two local descriptors are more robust to viewpoint variations, compared to holistic appearance models, they are not capable of capturing longer-term motion dynamics of the texture process, as a consequence of the aggregation operation. Similarly [24] performs only spatial binning, where descriptors are extracted from a set of

spatiotemporal oriented filters over different spatial windows of the video and then concatenated. Although the spatial binning captures location-dependent texture patterns, it is not able to well represent long-term temporal dynamics present in the texture processes.

Finally, with respect to our previous work, the BoS Tree was originally proposed in [25]. In contrast to [25], this paper presents a more complete analysis and significantly more experimental results, including: 1) a new experiment on semantic motion annotation using a video dataset of real scenes; 2) a new experiment on dynamic texture recognition using a large natural scene data set from [24], with comparisons to other state-of-the-art methods; 3) an insightful comparison of the descriptors produced by the BoS Tree and those produced by the standard BoS approach.

3 The BoS Representation

Analogous to the bag-of-words representation for text documents, the bag-of-systems (BoS) descriptor [1] represents videos with respect to a vocabulary, where generative time-series models, specifically linear dynamical *systems* or dynamic textures, are used in lieu of words.

3.1 The dynamic texture model

In general, the content of a video is represented by a set of P time series of low-level feature vectors $\mathcal{Y} = \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(P)}\}$, which correspond to spatio-temporal cubes sampled from the video, where P depends on the size of the video and the granularity of the sampling process. Each time series $\mathbf{y}^{(p)} = [y_1^{(p)}, \cdots, y_\tau^{(p)}]$ is composed of τ vectorized image patches extracted from consecutive frames. In the BoS representation, the codebook discretizes the space of time-series using a set of dynamic texture codewords.

The *dynamic texture* (DT) model [5] represents time series data by assuming that it is generated by a doubly embedded stochastic process, in which a lower dimensional hidden Gauss-Markov process $x_t \in \mathbb{R}^n$ encodes the temporal evolution of the observation process $y_t \in \mathbb{R}^m$. Specifically, the DT model is described by a *linear dynamical system* (LDS),

$$x_t = Ax_{t-1} + v_t, \quad (1)$$
$$y_t = Cx_t + w_t + \bar{y}, \quad (2)$$

and is specified by the parameters $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$, where the state transition matrix $A \in \mathbb{R}^{n\times n}$ encodes the dynamics of the hidden state variable (e.g., the evolution), the observation matrix $C \in \mathbb{R}^{m\times n}$ encodes the basis functions for representing the sequence, $v_t \sim \mathcal{N}(0, Q)$ and $w_t \sim \mathcal{N}(0, R)$ are respectively the driving and observation noises, $\bar{y} \in \mathbb{R}^n$ is the mean feature vector, and $\mathcal{N}(\mu, S)$ specifies the initial condition.

3.2 Learning the codebook

The BoS codebook $\mathcal{C}$ is learned from a training set $\mathcal{X}_c$, *i.e.*, a collection of representative videos. A two-stage procedure is typically used, where first each video is summarized with a set of DTs, followed by clustering of the video DTs to obtain the codewords.

In [1], spatio-temporal (ST) interest point operators are used to extract interesting motion patches, and DT parameters $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$ are estimated for each patch. The video DTs are then embedded into a Euclidean space via non-linear dimensionality reduction in tandem with the Martin distance [26, 9]. The embedded DTs are clustered in the Euclidean space using the K-means clustering algorithm. Finally, to represent each cluster, the learned DTs, which map the nearest to the cluster centers in the embedding, are selected as the codewords.

An alternative approach, presented in [2, 3], is based on the probabilistic framework of the DT. For each video, spatio-temporal patches are extracted using dense sampling, and a dynamic texture mixture (DTM) is learned for each video using the EM algorithm [27]. The video DTs are then directly clustered using the hierarchical EM algorithm, producing novel DT cluster centers that are used as the codewords.

Finally, a third approach [4] defines a distance between DTs as the minimal distance after aligning the two DTs in their equivalence space. The distance is then used with a generalized K-means algorithm to cluster the video patch DTs, forming a novel DT codewords.

While these three approaches effectively produce small-sized codebooks (both [1, 2] use 8 codewords, [3] uses 64, and [4] uses 56), they are only applicable to small and simple datasets, *e.g.*, UCLA 8-class [1]. Indeed, they are not rich enough to produce accurate classifications when applied to larger or more challenging datasets, as demonstrated in the experiments in Section 5. A final approach [28] forms a large codebook by directly selecting each DT from the video-level DTMs as a codeword. This forms a very large (and hence rich) codebook (400 codewords), but has significant computational cost when mapping to the codebook.

3.3 Projection to the codebook

Given a codebook $\mathcal{C}$, a video $\mathcal{Y}$ is represented by a BoS histogram $\mathbf{h}_\mathcal{Y} \in \mathbb{R}^{|\mathcal{C}|}$ that records how often each codeword appears in that video. To build the BoS histogram, we extract a dense sampling of spatio-temporal cubes from $\mathcal{Y}$. Each cube $\mathbf{y}^{(p)}$ is compared to each codeword $\Theta_i \in \mathcal{C}$ by using the likelihood of the codeword generating the cube, $p(\mathbf{y}^{(p)}|\Theta_i)$, which can be efficiently computed with the "innovations" form using the Kalman filter [29]. Defining a quantization threshold $k \in \{1, \dots, |\mathcal{C}|\}$, each cube is then assigned to the k most likely codewords, and the BoS histogram for $\mathcal{Y}$ is finally built by counting the frequency with which each codeword is selected. Specifically, the weight of codeword Θ_i is calculated with

$$\mathbf{h}_\mathcal{Y}[i] = \frac{1}{|\mathcal{Y}|} \sum_{\mathbf{y}^{(p)}\in\mathcal{Y}} \frac{1}{k} \mathbb{1}[i \in \operatorname*{argmax}_j^k p(\mathbf{y}^{(p)}|\Theta_j)], \quad (3)$$

where $\operatorname{argmax}_j^k$ returns the indices of the codewords with the k-largest likelihoods, and $\mathbb{1}[\cdot]$ is the indicator function.

When the quantization threshold k is equal to 1, then (3) reduces to the typical notion of the *term frequency* (TF) representation. The effect of $k > 1$ is to counteract quantization errors that can occur when a time series is approximated equally well by multiple codewords.

An alternative to the standard TF representation is the *term frequency-inverse document frequency* (TF-IDF) representation, which takes into account the statistics of the training set by assigning more weight to codewords that appear less frequently in the collection, and down-weighting codewords that are more common. Specifically, given the BoS histogram $\mathbf{h}_\mathcal{Y}$, the corresponding TF-IDF representation is obtained with the following mapping:

$$\hat{\mathbf{h}}_\mathcal{Y}[i] = \frac{1}{\alpha}\mathbf{h}_\mathcal{Y}[i] \cdot IDF[i], \quad \text{for } i = 1, \dots, |\mathcal{C}|, \tag{4}$$

where α normalizes the histogram, and the IDF factor is computed as

$$IDF[i] = \log \frac{|\mathcal{X}_c|}{|\{\mathcal{Y} \in \mathcal{X}_c : h_\mathcal{Y}[i] > 0\}|}. \tag{5}$$

The denominator in (5) is the number of training videos that exhibit at least one codeword Θ_i. Hence if all videos contain the codeword Θ_i, then the IDF factor will be 0 and the codeword is uninformative.

Mapping a video $\mathcal{Y}$ to its BoS histogram $\mathbf{h}_\mathcal{Y}$ requires a total of $|\mathcal{Y}||\mathcal{C}|$ likelihood computations, *i.e.*, each spatio-temporal cube $\mathbf{y}^{(p)} \in \mathcal{Y}$ is compared to each codeword $\Theta_i \in \mathcal{C}$. When both $|\mathcal{Y}|$ and $|\mathcal{C}|$ are large, projecting one video on the codebook is computationally demanding, especially when using large video patches, which makes individual likelihood comparisons slow. Therefore, the deployment of a large codebook is impractical due to the associated long delays. However, representing the variety of visual information typical of large and diverse video collections requires a rich, large codebook. In the next section we propose the BoS Tree which, by organizing codewords in a bottom-up hierarchy, reduces the number of computations necessary to index a large collection of codewords.

4 The BoS Trees

In this section we propose the *BoS Tree*, which consists of a bottom-up hierarchy of codewords learned from a corpus of representative videos. The bottom level of the tree is formed by a large collection of codewords. A tree structure is then formed by repeatedly using the HEM algorithm to cluster the codewords at one level, and using the novel cluster centers as codewords at the new level. Branches are formed between the codewords at a given level and their cluster centers at the next higher level.

When mapping a new video onto the codebook, each video patch is first mapped onto the codewords forming the top-level of the BoS Tree. Next, the video patch is *propagated* down the tree, by identifying branches with the most-promising codewords (i.e., with largest likelihood). Selecting the most-likely branches reduces the number of likelihood computations, while also preserving the descriptor quality, since portions of the tree that are not explored are not likely to be codewords for that patch. In this way, the BoS Tree efficiently indexes codewords while preserving the quality of the BoS descriptor, and hence enables the deployment of larger codebooks in practical applications.

In this section, we first discuss the HEM algorithm for clustering dynamic textures, followed by the algorithms used for forming and using the BoS Tree.

4.1 The HEM algorithm

Given a collection of DTs, the HEM algorithm for DTMs (HEM-DTM) [2, 3] partitions them into K clusters of DTs that are "similar" in terms of their probability distributions, while also learning a *novel* DT to represent each cluster. This is similar to K-means clustering, with the difference that the data points are DTs instead of Euclidean vectors.

Specifically, the HEM-DTM takes as input a DTM with $K^{(b)}$ components and reduces it to a new DTM with fewer components $K^{(r)} < K^{(b)}$. Given the input DTM $\Theta^{(b)} = \{\Theta_i^{(b)}, \pi_i^{(b)}\}_{i=1}^{K^{(b)}}$, the likelihood of a spatio-temporal cube $\mathbf{y}$ is given by

$$p(\mathbf{y}|\Theta^{(b)}) = \sum_{i=1}^{K^{(b)}} \pi_i^{(b)} p(\mathbf{y}|z^{(b)} = i, \Theta_i^{(b)}), \tag{6}$$

where $z^{(b)} \sim \text{multinomial}(\pi_1^{(b)}, \cdots, \pi_{K^{(b)}}^{(b)})$ is the hidden variable that indexes the mixture components. $p(\mathbf{y}|z^{(b)} = i, \Theta_i^{(b)})$ is the likelihood of $\mathbf{y}$ under the i^{th} mixture component, and $\pi_i^{(b)}$ is the prior weight for the i^{th} component.

The goal is to find a reduced model $\Theta^{(r)}$, which represents (6) using fewer mixture components. The likelihood of the spatio-temporal cube $\mathbf{y}$ given the reduced mixture $\Theta^{(r)} = \{\Theta_j^{(r)}, \pi_j^{(r)}\}_{j=1}^{K^{(r)}}$ is given by

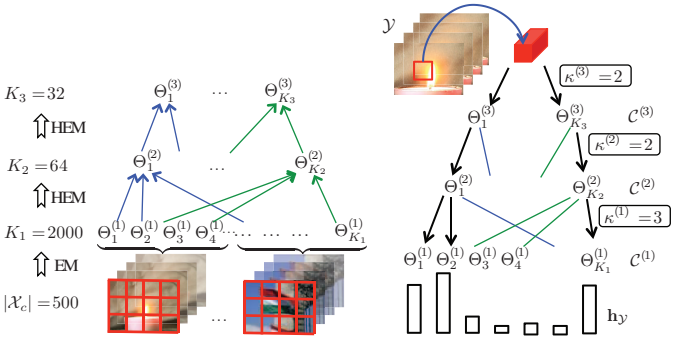
$$p(\mathbf{y}|\Theta^{(r)}) = \sum_{j=1}^{K^{(r)}} \pi_j^{(r)} p(\mathbf{y}|z^{(r)} = j, \Theta_j^{(r)}), \tag{7}$$

where $z^{(r)} \sim \text{multinomial}(\pi_1^{(r)}, \cdots, \pi_{K^{(r)}}^{(r)})$ is the hidden variable for indexing components in $\Theta^{(r)}$.

The HEM-DTM algorithm estimates (7) from (6) by maximizing the likelihood of N *virtual* spatio-temporal cubes $Y = \{Y^i\}_{i=1}^{K^{(b)}}$ generated accordingly to $\Theta^{(b)}$, where Y^i is a set of $N_i = \pi_i^{(b)} N$ samples drawn from $\Theta_i^{(b)}$. In order to produce a consistent clustering of the input DTs, the HEM algorithm assigns the whole sample set Y^i to a single component of the reduced model. Assuming that the size of the virtual sample is appropriately large, the law of large number allows the virtual samples to be replaced with an expectation with respect to the input DTs. A complete description of HEM-DTM appears in [2, 3], while here we note that the output of the HEM algorithm is: 1) a clustering of the original $K^{(b)}$ components into $K^{(r)}$ groups, where the cluster membership is encoded by the assignments $\hat{z}_{i,j} = p(z^{(r)} = j|z^{(b)} = i)$, and 2) novel cluster centers represented by the individual mixture components of (7), *i.e.*, $\{\Theta_j^{(r)}\}_{j=1}^{K^{(r)}}$.

4.2 Building a BoS Tree

A BoS Tree is built from a collection of representative videos $\mathcal{X}_c$ with an unsupervised clustering process based on the HEM-DTM algorithm (Figure 1a). The bottom level of the tree $\mathcal{C}^{(1)} = \{\Theta_i^{(1)}\}_{i=1}^{K_1}$ consists of a large codebook compiled by pooling together the DT codewords extracted from individual videos in $\mathcal{X}_c$ (as in [28]). For each video, a DTM with K_v components is learned using the EM algorithm [27], and the DTMs from all videos are pooled to form $K_1 = K_v|\mathcal{X}_c|$ codewords. Next, starting from the bottom level, a BoS



(a) Building a BoS Tree. (b) Indexing with a BoS Tree.

Fig. 1: (a) A BoS Tree is built from a collection of videos $\mathcal{X}_c$ by forming a hierarchy of codewords. (b) The tree structure of the BoS Tree enables efficient indexing of codewords.

Tree of L levels is built recursively using the HEM-DTM algorithm $L-1$ times. Each new level of the BoS Tree, *i.e.*, $\mathcal{C}^{(\ell+1)} = \{\Theta_j^{(\ell+1)}\}_{j=1}^{K_{\ell+1}}$, is formed by clustering the K_ℓ DTs at the previous level ℓ into $K_{\ell+1} < K_\ell$ groups using the HEM-DTM algorithm. In particular, the input mixture is given by the DT codewords at level ℓ with uniform weight, *i.e.*, $\Theta^{(b)} = \{\Theta_i^{(\ell)}, \frac{1}{K_\ell}\}_{i=1}^{K_\ell}$, and the novel DT cluster centers learned by the HEM-DTM algorithm are used as codewords at the new level, *i.e.*, $\mathcal{C}^{(\ell+1)} = \{\Theta_j^{(r)}\}_{j=1}^{K_{\ell+1}}$.

The branches between contiguous levels in the BoS Tree are instantiated as dictated by the assignment variables $\hat{z}_{i,j}$ of the HEM-DTM algorithm, which is a function of the Kullback-Leibler (KL) divergence between DTs at each level. In particular, to connect level $\ell+1$ to level ℓ, we define the set of branches for each codeword $j \in [1\, K_{\ell+1}]$ from level $\ell+1$ as

$$\mathcal{B}_j^{(\ell+1)} = \{i \in [1\, K_\ell] | j = \operatorname*{argmin}_h \mathrm{KL}(\Theta_i^{(\ell)} || \Theta_h^{(\ell+1)})\}. \quad (8)$$

This is effectively the set of input DTs (at level ℓ) that are assigned to cluster j when constructing level $\ell+1$ of the BoS Tree. Finally, the BoS Tree $\mathcal{T}$ is the collection of codewords at each level and their corresponding branch sets, *i.e.*, $\mathcal{T} = \{\mathcal{C}^{(1)}, \cdots, \mathcal{C}^{(L)}, \mathcal{B}^{(2)}, \cdots, \mathcal{B}^{(L)}\}$.

It is worth mentioning here that BoS Tree is built in a bottom-up manner, unlike tree-structured vector quantizers [8], which use a top-down approach to cluster the entire collection of feature vectors from the training videos. The reasons are two-fold. First, the top-down approach requires running the EM algorithm [27] on the full training set (densely sampled patches from all training videos) to learn DTMs at each level of the tree. Although the training videos are split between branches of the tree, all videos are still being processed by EM at each level. In contrast, for the bottom-up approach, the training videos are only used for EM learning on the bottom-level of the tree. The subsequent higher-levels are obtained by directly clustering the DT codewords from the lower-levels. Hence, the bottom-up approach is more scalable, especially when E-step inference is computationally intensive as in the case for DTMs.[1] Second, the bottom-up strategy computes a "video-based" codebook with codewords that are spread out evenly among the training videos, i.e., each video contributes a few representative codewords. Recent work [30] shows that a "video-based" codebook obtains better annotation/retrieval performance than a "collection-based" codebook, where codewords are learned from the whole training set at once. [30] suggests that the "video-based" codebook has better discrimination ability since it contains more codewords in the high-density regions of the feature space.

4.3 Fast codewords indexing with BoS Trees

The BoS Tree $\mathcal{T}$ allows for quick look-ups in the large codebook $\mathcal{C}^{(1)}$, which forms the bottom level of the tree, by leveraging the hierarchical structure to index the codewords efficiently (Figure 1b). To map a video $\mathcal{Y}$ to its BoS histogram $\mathbf{h}_\mathcal{Y} \in \mathbb{R}^{K_1}$, we extract a dense sampling of spatio-temporal cubes and propagate each cube down only the more promising paths of the BoS Tree. In particular, each cube $\mathbf{y}$ is initially compared to the codewords at the top level of the BoS Tree (*i.e.*, level L), and assigned to the $\kappa^{(L)}$ most likely ones,

$$\mathcal{J}^{(L)} = \operatorname*{argmax}_{j \in [1\, K_L]}^{\kappa^{(L)}} p(\mathbf{y}|\Theta_j^{(L)}). \quad (9)$$

$\mathcal{J}^{(L)}$ is the set codewords indices at level L that will be explored, and the parameter $\kappa^{(L)}$ controls how many branches are explored at level L. Next, the cube $\mathbf{y}$ is propagated down to the successive level following the branches that depart from the codewords selected at the current level,

$$\mathcal{J}^{(\ell)} = \operatorname*{argmax}_{i \in \bigcup_{j \in \mathcal{J}^{(\ell+1)}} \mathcal{B}_j^{(\ell+1)}}^{\kappa^{(\ell)}} p(\mathbf{y}|\Theta_i^{(\ell)}), \quad (10)$$

for $\ell = L-1, L-2, \ldots, 2, 1$. At the bottom level of the BoS Tree (*i.e.*, $\ell = 1$), the number of occurrences of each codeword is registered, and TF or TF-IDF histograms are then computed.

Setting the quantization thresholds $[\kappa^{(1)}, \ldots, \kappa^{(L)}]$ to values larger than 1 counteracts the effect of quantization errors and improves the accuracy of the BoS (in comparison to the full codebook computation), but increases the number of likelihood computations.

An alternative to selecting a *fixed* number of codewords at one level, is to select a *variable* number of codewords based on the uncertainty of the BoS quantization. This is implemented by defining the operator

$$\Omega(\mathcal{J}, \mathfrak{T}) = \{j \in \mathcal{J} | p(\mathbf{y}|\Theta_j) \geq \mathfrak{T} \max_{h \in J} p(\mathbf{y}|\Theta_h)\}$$

which selects all codewords whose likelihood is within a threshold $\mathfrak{T}$ from the largest, and replacing (9) and (10) with

$$\mathcal{J}^{(L)} = \Omega([1\, K_L], \mathfrak{T}^{(L)}) \quad (11)$$

$$\mathcal{J}^{(\ell)} = \Omega(\cup_{j \in \mathcal{J}^{(\ell+1)}} \mathcal{B}_j^{(\ell+1)}, \mathfrak{T}^{(\ell)}). \quad (12)$$

1. Formally, consider a two-level tree with K_1 and K_2 codewords at the bottom and top levels (branching factor of $B = K_1/K_2$), which is built from N training patches. For the top-down approach, the data is clustered into K_2 groups, and then each group is clustered into B sub-groups, resulting in $O(N(B+K_2))$ E-step inference operations. For the bottom-up approach, patches from each video are grouped into K_v clusters, and then the resulting K_1 codewords are grouped into K_2 clusters, which yields $O(NK_v + K_1K_2)$ E-step operations. Under the reasonable assumption that $K_1 \ll N$, the bottom-up approach will be more efficient when $K_v < B + K_2$. In our experiments, a typical setting is $K_v = 4$, $K_1 = 640$, $K_2 = 64$, $B = 10$, and $N = 72000$.

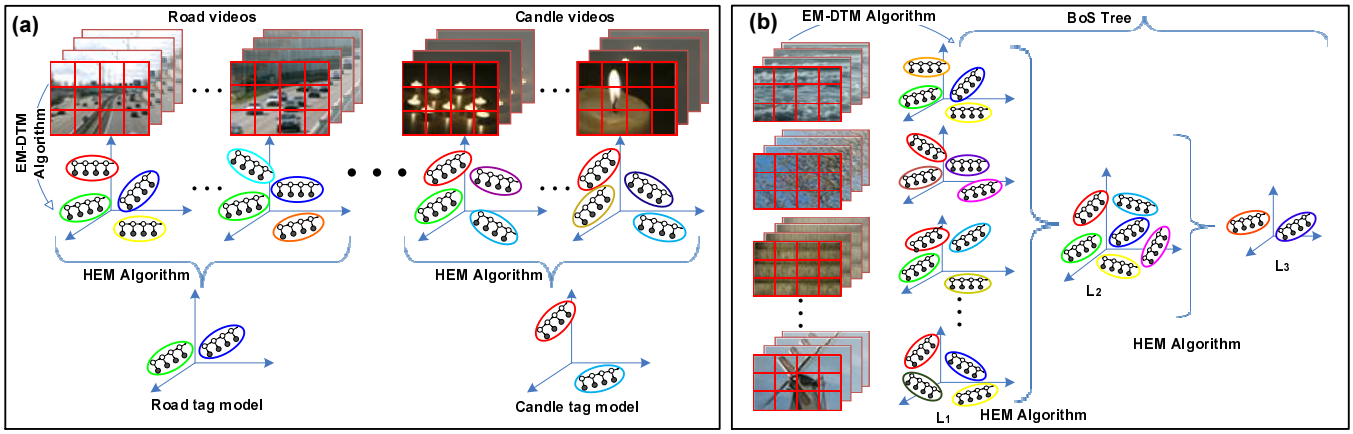


Fig. 2: Comparison of SML-DTM framework and BoS Tree framework: (a) Learning DT annotation models using HEM-DTM algorithm. (b) Building a BoS Tree with 3 levels.

The BoS Tree reduces the number of likelihood computations necessary to map a video to its codebook representation. Assuming that a BoS Tree has K top-level codewords, L levels, and B branches on average per codeword, the average number of likelihood computations required for the BoS Tree look-up is $(K + B(L-1))$, which is much less than the $K \cdot B^{L-1}$ computations required for directly indexing the bottom-level of the tree. Therefore, the BoS Tree enables the use of large and rich codebooks while still maintaining an acceptable look-up time. As the portions of the BoS Tree that are not explored are the ones that are not likely to provide appropriate codewords for a given video (in both the likelihood sense for a tested codeword, and KL-divergence sense for the children of that codeword), there is not expected to be a big loss in performance with respect to linear indexing of a large codebook. We demonstrate this experimentally in Section 5.

5 Applications and Experiments

In this section we present an empirical evaluation of the BoS Tree. We first present an application of the BoS Tree to semantic motion annotation of videos and to dynamic scene recognition. Then, to demonstrate the applicability of the BoS Tree framework to time series data other than video, we present an experiment on automatic music annotation.

5.1 Implementation notes

In the following experiments, the EM algorithm for DTMs (EM-DTM) [27] is first used to learn *video-level* DTMs from overlapping video patches (spatio-temporal cubes) extracted from the video. These individual mixture components of the video-level DTMs form the bottom level of the BoS Tree as described in Section 4.2. We initialize the EM-DTM algorithm using an iterative "component splitting" procedure, as described in [27], where EM is run repeatedly with an increasing number of mixture components. Specifically, we start by estimating a DTM with $K = 1$ components by running EM-DTM to convergence.[2] Next, we select a DT component, and duplicate it to form two components (this is the "splitting"), followed by slightly perturbing the DT parameters. This new DTM with $K = 2$ components serves as the initialization for EM-DTM, which is again run until convergence. The process is repeated until the desired number of components is reached. We use a growing schedule of $K = \{1, 2, 4, 8, 16\}$, and perturb the observation matrix C when creating new DT components.

We use a similar procedure for initializing the HEM-DTM algorithm [3], which is used to build successive levels in the BoS Tree. We set the number of virtual samples of the HEM-DTM algorithm to $N = 1000$ and temporal length $\tau = 20$. The state-space dimension is set to $n = 10$. Finally, we use an isotropic covariance matrix for the observation noise of the DT, *i.e.*, $R = rI$.

5.2 Types of BoS descriptors

In our experiments we test the following types of BoS descriptors, which vary in their size and indexing method:

- **BoS Tree:** A BoS Tree (BoST) codebook is formed from all videos in the training set, as described in Section 4.2 and illustrated in Figure 2b. The 1st (bottom) level of the BoS Tree is used as the codebook. A video is mapped to a BoS histogram using the fast tree-based indexing described in Section 4.3.
- **large BoS:** The *bottom* level codebook of the BoS Tree is used as the codebook. Given a video, the BoS histogram is calculated using *direct indexing* of the codewords. The expressive power of the large BoS is the same as the BoS Tree; the main difference is in the indexing method.
- **reduced BoS:** A BoS with a reduced number of codewords is generated by applying the HEM algorithm to the large BoS as in [2, 3]. This is equivalent to using the DTs at the 2nd-level of the BoS Tree as the codewords. The BoS histogram is calculated using direct indexing.

The BoS histograms are then converted to TF or TF-IDF descriptors. Note that the reduced and large BoS use direct-indexing, and hence they can be regarded as instances of [2, 3].

5.3 Semantic video texture annotation

In the first experiment, we perform video annotation on the DynTex data set [31] using BoS Tree, and compare results with supervised multi-class labeling with DTM models (SML-DTM) [3]. We use the BoS Tree to map videos to their histogram representations over the codebook, and successively fit a classifier to make prediction based on these histograms.

In our earlier SML-DTM work [3], each tag was modeled with a DTM, which was learned by applying the HEM-DTM

2. The single component is initialized using the least-squares method [5].

algorithm directly to the video-level DTMs associated with that tag, as shown in Figure 2a. A video is then annotated by computing the likelihood of its video patches under the tag DTMs, and then selecting the k most likely tags. Note that the SML-DTM framework in [3] represents each video as a set of patches, whereas the BoS Tree framework uses a higher level encoding (i.e., the BoS histogram).

5.3.1 BoS tree video annotation framework

In this section we present a framework for video annotation using BoS Trees. First, all videos are represented using TF-IDF descriptors, which are obtained using the BoS Tree, and a support vector machine (SVM) classifier [32] is trained for each tag model.[3] In order to account for the simplicial structure of the BoS histograms, we used a kernel SVM with the Bhattacharyya kernel (see Table 3), since this kernel produced the best results in previous work on BoS classification [3]. In practice, the Bhattacharyya kernel is taking the square root of the histogram entries. This has the effect of projecting the points on a simplex to the shell of a sphere, where they are more likely to be linearly separable. SVM parameters were determined using 10-fold cross-validation on the training set.

To annotate a test video, first its BoS Tree representation is computed, and then the probability of the video being the positive class is calculated for each binary tag classifier. The probabilities from the binary classifiers are concatenated and normalized, producing a semantic multinomial for annotation. For implementation of binary classifiers and probability estimates, we used the LibSVM package [33].

5.3.2 Dataset

We performed video annotation experiments on the DynTex database [31], which is a collection of 385 video sequences of everyday surroundings with ground-truth annotations. We follow the benchmark protocol established in [3], which comprises of the 35 most frequent tags (337 sequences). These tags are grouped into two major categories: *process* tags, which are mainly based on the appearance and describes the physical texture process (*e.g.*, flag, road, and windmill); *structural* tags, which describe only the motion characteristics and can have a wide range of appearances (*e.g.*, waving and turbulent). Videos are labeled with variable number of tags with an average of 2.34 tags per video. This benchmark consists of 5 trials, where videos are randomly split into 50% training and 50% test sets.[4] Examples from the dataset are presented in the Appendix.

5.3.3 Experimental setup

Following the settings in [3], videos are first truncated to 50 frames, converted to grayscale, and downsampled by 3 times using bicubic interpolation. Overlapping spatio-temporal cubes of size $7 \times 7 \times 20$ (step of $4 \times 4 \times 10$) are then extracted from these videos. Video-level DTMs are learned with $K_v = 4$ components, giving on average of 670 codeword in the large codebook at the bottom level $\mathcal{C}^{(1)}$ of the tree. A two-level BoS tree is formed by reducing the large codebook of size 670 to 16 codewords only.

3. In particular, for a given tag, a 1-vs-all binary classifier is trained with positive examples corresponding to videos with that tag, and negative examples as the remaining videos.

4. Details of the benchmark and more experimental results can be found at: http://visal.cs.cityu.edu.hk/research/hemdtm/

TABLE 1: Annotation results on the DynTex dataset.

	Avg. Precision	Avg. Recall	Avg. F-Measure	X-speedup
SML-DTM [3]	0.420	0.507	0.397	1.0
BoSTree (K=670)	0.467	0.484	0.430	3.8
large BoS (K=670)	0.468	0.510	0.442	0.41
reduced BoS (K=16)	0.133	0.157	0.124	17.5

Following the procedure in [34] we measured annotation performance by computing precision, recall and F-score for each individual tag, as well as their averages across all tags. Precision for a tag is the probability that the model correctly uses the tag while annotating a video and recall for a tag is the probability that the model annotates a video that should have been annotated with the tag. Precision, recall and F-score measure for a tag w are defined as:

$$P = \frac{|W_C|}{|W_A|}, R = \frac{|W_C|}{|W_T|}, F = 2((P)^{-1} + (R)^{-1})^{-1}, \quad (13)$$

where $|W_T|$ is the number of sequences that have tag w in the ground truth, $|W_A|$ is the number of times the annotation system uses w when automatically tagging a video, and $|W_C|$ is the number of times w is correctly used. When a tag is never selected for annotation, the corresponding precision (that otherwise would be undefined) is set to the tag prior from the training set, which equals the performance of a random classifier.

We compared the annotation performance of the BoS Tree (BoST) representation with DTM tag models (SML-DTM) [3]. In addition to BoS Trees, we also consider several alternative methods for BoS histograms: a *large BoS* of size $K = 670$ and a *reduced BoS* of size $K = 16$. All results are averaged over 5 trials in the benchmark.

5.3.4 Annotation results

Table 1 presents an overall comparison between different annotation methods based on average precision, recall, and F-measure for annotation with top 3 tags. Annotation using BoST outperforms SML-DTM with an average F-score of 0.430 versus 0.397. Figure 3 plots the PR curve and F-score for all 35 annotation levels for BoST and SML-DTM, and shows that the BoST annotation dominates SML-DTM at almost all levels of recall. Table 2 breaks down the performances of SML-DTM and BoS Trees on individual tags, as well as averaged over *process* and *structural* categories. Once again BoS Trees dominating in most of the individual tags with an

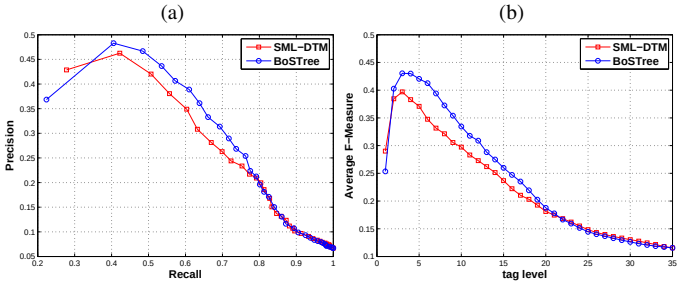


Fig. 3: (a) Average precision/recall plot; (b) F-measure plot, showing all annotation levels, using BoSTree and SML-DTM on DynTex.

TABLE 2: Per-tag performance of BoSTree and SML-DTM annotation on DynTex. Average number of training videos available for each tag are in parenthesis.

		Precision		Recall		F-Measure	
		SML-DTM	BoST	SML-DTM	BoST	SML-DTM	BoST
process tags	*anemone(9.8)*	0.291	0.451	0.751	0.678	0.410	**0.519**
	aquarium(1.8)	0.140	0.333	0.167	0.400	0.150	**0.317**
	basin(9.8)	0.150	0.610	0.353	0.192	0.191	**0.208**
	boiling(3.6)	0.235	0.533	1.000	0.580	0.372	**0.514**
	candle(4.4)	0.482	0.588	1.000	0.960	0.631	**0.709**
	escalator(2.8)	0.733	0.467	0.450	0.517	**0.513**	0.474
	field(3.6)	0.486	0.645	0.312	0.593	0.350	**0.593**
	flag(8.4)	0.226	0.486	0.575	0.529	0.308	**0.501**
	foam(3.8)	0.362	0.267	0.667	0.333	**0.431**	0.293
	fountain(29.4)	0.437	0.514	0.501	0.745	0.463	**0.607**
	laundry(3)	0.138	0.707	0.800	0.840	0.232	**0.687**
	mobile(1.6)	0.800	0.600	0.217	0.167	**0.340**	0.260
	net(1.4)	0.700	0.400	0.400	0.267	**0.480**	0.300
	plant(14.2)	0.261	0.275	0.781	0.557	**0.390**	0.366
	pond(4)	0.273	0.400	0.720	0.213	**0.367**	0.274
	rain(2)	1.000	0.600	0.733	0.467	**0.800**	0.500
	river(8)	0.287	0.282	0.456	0.206	**0.351**	0.230
	road(1.8)	1.000	0.900	0.600	0.733	0.700	**0.753**
	sea(7)	0.539	0.750	0.896	0.876	0.661	**0.799**
	server(1.4)	1.000	0.800	0.700	0.900	**0.800**	0.800
	shower(1.6)	0.000	0.000	0.000	0.000	0.000	0.000
	sky(2)	0.000	0.000	0.000	0.000	0.000	0.000
	source(6)	0.323	0.750	0.350	0.350	0.316	**0.436**
	stream(12.6)	0.335	0.268	0.249	0.198	**0.249**	0.185
	toilet(1.6)	0.600	0.400	0.600	0.300	**0.527**	0.333
	tree(20.8)	0.503	0.459	0.827	0.721	**0.624**	0.554
	waterfall(10.2)	0.119	0.347	0.187	0.152	0.133	**0.183**
	windmill(4.2)	0.179	0.279	0.400	0.400	0.217	**0.258**
structural tags	*dmotion(45.8)*	0.468	0.409	0.335	0.779	0.385	**0.532**
	dmotions(21.8)	0.212	0.298	0.269	0.295	0.227	**0.291**
	interinsic(8)	0.491	0.643	0.725	0.684	0.560	**0.627**
	oscillating(49)	0.681	0.700	0.722	0.893	0.692	**0.783**
	random(4.8)	0.383	0.267	0.251	0.069	**0.229**	0.100
	turbulent(46.2)	0.498	0.543	0.429	0.735	0.456	**0.620**
	waving(38.2)	0.374	0.367	0.313	0.621	0.339	**0.461**
	Process	**0.414**	**0.468**	**0.525**	**0.460**	**0.393**	***0.416***
	Structural	**0.444**	**0.461**	**0.435**	**0.582**	**0.412**	***0.488***

average *process* category F-score of 0.416 versus 0.393 and average *structural* category F-score of 0.488 versus 0.412. On the individual process tags, BoST typically outperforms SML-DTM when there are more training examples for the tag. Likewise, BoST outperforms SML-DTM on almost all the structural tags, since these tags are groups of process tags and hence more training data is available. By construction, tags with fewer training examples have fewer codewords in the codebook. Because there are fewer dedicated codewords for these tags, it may be more difficult for the tag model to overcome noise in the descriptor.

Table 1 also reports the speedup (in terms of average number of likelihood calculations relative to SML-DTM) used for annotating a video. BoST requires almost 4 times less likelihood calculations than SML-DTM. These results suggest that BoS Trees is a quicker and more accurate method for computing the descriptor for video annotation. Next we compare BoST and the large BoS with direct indexing. For the same-sized large codebook, BoS and BoST obtain similar average precision, but BoS has better average recall. However, this improved recall comes at the expense of using more than 9 times more likelihood computations. Finally, BoS with a small codebook performs significantly worse than BoS/BoST with the large codebook. These results demonstrate that a large codebook can significantly improve accuracy, while using BoS Tree reduces the computational requirements with an acceptable loss in recall.

5.4 Dynamic texture recognition

We validate the BoS codebook trees on the task of dynamic texture recognition, while comparing with existing state-of-the-art methods [24, 35, 36, 37, 38, 10, 39, 40, 41].

5.4.1 Datasets

We validate our proposed BoS Tree on four datasets, following the protocols established by their respective papers and comparing to their published results. Example frames from these datasets are presented in the Appendix.

YUPENN dynamic scenes: [24] introduces a new dataset that emphasizes scene specific temporal information over short time durations due to objects and surfaces rather than camera-induced ones, which are predominant in the Maryland data set [38]. YUPENN consists of fourteen dynamic scene categories, each containing 30 color videos. Representative images of each class are shown in Figure 4. The average dimensions of the videos are $250 \times 370 \times 145$ (H x W x L). This dataset is very challenging and consists of videos obtained from various sources such as footage captured by the authors, YouTube, BBC Motion Gallery and Getty Images. Due to the diversity of video sources, the videos contain significant differences in image resolution, frame rate, scene appearance, scale, illumination conditions, and camera viewpoint.

UCLA-39: The UCLA-39 dataset [40] contains 312 gray-scale videos representing 39 *spatially stationary* classes, which were selected from the original 50 UCLA texture classes [6].

Fig. 4: Examples from YUPENN dynamic scenes data set.

TABLE 3: Distances and kernels used for classification.

square-root distance	(SR)	$d_s(h_1,h_2) = \arccos(\sum_k \sqrt{h_{1k}h_{2k}})$
χ^2-distance	(CS)	$d_{\chi^2}(h_1,h_2) = \frac{1}{2}\sum_k \frac{\lvert h_{1k}-h_{2k}\rvert}{h_{1k}+h_{2k}}$
χ^2 kernel	(CSK)	$k_{\chi^2}(h_1,h_2) = 1 - \sum_k \frac{(h_{1k}-h_{2k})^2}{\frac{1}{2}(h_{1k}+h_{2k})}$
Intersection kernel	(HIK)	$k_I(h_1,h_2) = \sum_k \min(h_{1k},h_{2k})$
Bhattacharyya kernel	(BCK)	$k_B(h_1,h_2) = \sum_k \sqrt{h_{1k}h_{2k}}$

Each video is cropped into a right portion and a left portion (each 48×48), with one used for training and the other for testing. Classification of UCLA-39 is the most challenging variant of the UCLA-based datasets (e.g., [6, 1, 39]), and tests the translation invariance of the feature descriptor, since the training video patch is visually quite different from the testing patch.

UCLA-8: [1] groups related classes from the original 50 UCLA texture classes into 9 super-classes, where each super-class contains different viewpoints of the same texture process. In [1], experiments are conducted on 8 of these classes. The original *uncropped* videos are used.

DynTex-35: The DynTex-35 dataset [41] is a collection of videos from 35 texture classes from everyday surroundings. Originally, the data consisted of a single video of size $192 \times 240 \times 50$ per class. As in [41], each video is split into 10 non-overlapping sub-videos (each having different spatial and temporal dimensions).

5.4.2 Experiment setup

For the YUPENN, each video is truncated to 150 frames, converted to grayscale, and downsampled such that the largest spatial dimension is 128 (while keeping the same aspect ratio). Overlapping spatio-temporal cubes of size $7 \times 7 \times 20$ (step: $5 \times 5 \times 15$) are then extracted from the YUPENN videos. For UCLA-39 and UCLA-8, overlapping spatio-temporal cubes with size $5 \times 5 \times 75$ (step: $2 \times 2 \times 75$) pixels are extracted densely from the grayscale video. For DynTex35, the videos are converted to grayscale, and overlapping spatio-temporal cubes with size $7 \times 7 \times 50$ (step: $5 \times 5 \times 30$) pixels are extracted. We retain only video cubes with a minimum total variance of 5 for YUPENN dynamic scenes dataset, and 1 for UCLA-39, UCLA-8 and DynTex35, hence discarding cubes that do not contain significant motion.

For each cross-validation split in our datasets, the BoS Tree was learned from the training set only. For UCLA-39, UCLA-8, DynTex35, a DTM with $K_v = 4$ components is learned for each video from its spatio-temporal cubes. For YUPENN, the video-level DTMs used only $K_v = 2$ due to the large size of the training set. The DTs from all videos are collected to form the DT codewords, *i.e.*, $K_1 = K_v|\mathcal{X}_c|$, where $|\mathcal{X}_c|$ is the size of the training set. The BoS Tree is then formed by successively applying the HEM-DTM algorithm, as described in the previous section. For YUPENN and UCLA-8, we build a three level tree, using $K_2 = 64$, $K_3 = 16$ and $K_2 = 16$, $K_3 = 8$ respectively. For UCLA-39 and DynTex35 we tested different trees for $L \in \{2, 3, 4\}$ levels, using $K_2 = 64$, $K_3 = 32$ and $K_4 = 16$. We used $\kappa^{(\ell)} = \kappa = 1$ or $\mathfrak{T}^{(\ell)} = \mathfrak{T} = 0.995$ for traversing the BoS Trees.

For video classification, we first map all the videos to their BoS histograms using the learned BoS Tree, and then represent the visual content of each video as TF and TFIDF vectors. We then use a k-nearest neighbor (k-NN) classifier or support vector machine (SVM) for the video classification task. In order to account for the *simplicial* structure of BoS histograms, we build our k-NN classifier in terms χ^2-distance (CS) or square root-distance (SR) (Table 3), which are appropriate distance metrics for histograms. Similarly, for SVM we use the chi-squared kernel (CSK), Bhattacharyya kernel (BCK), or histogram intersection kernel (HIK), as in Table 3. The LibSVM software package [33] was used for the SVM, with all parameters selected using a 10-fold cross-validation on the training set.

In addition to BoS Trees, we also consider several alternative methods for BoS histograms: *large BoS* (838 codewords for YUPENN, 184 for UCLA-8, 624 for UCLA-39, and 630 for DynTex-35); and a *reduced BoS* of size $K \in \{8, 16, 32, 64\}$. Note that the reduced codebook with $K = 64$ corresponds to the BoS from [3].

We also consider a standard bag-of-visual-words (BoW) representation for vectorized video patches, which is computed using tree-structured vector quantization (TSVQ) [8]. PCA is applied to the vectorized video patches to reduce the dimension to 100.[5] The TSVQ is trained from the same set of (vectorized) video patches as the BOS Tree. We set the branching factor (*BF*) for different levels of the VQ tree such that the number of codewords at each level matches closely with the BoS Tree.[6] For video classification using TSVQ, we first map all the videos to BoW descriptors, and then apply the same set of classifiers used with BoS. To save space, we only report the best TSVQ result among the various combinations of distance/kernel and classifier.

For each experiment we registered average classification accuracy. For experiments involving UCLA and DynTex datasets we also counted the average number of likelihood computations (per video) executed at test time to produce the BoS histograms, from which we computed the speed up with respect to the large BoS codebook (X-Speedup). A small number of likelihood computations results in faster look-ups in the codebook and a larger speedup. We used the same leave-one-video-out protocol for YUPENN as in [24], resulting in a total of 420 trials. Results are averaged over 20 random trials on UCLA-8 (50% training and 50% test videos) as in [1], 2 trials on UCLA-39 (training on the right sub-video, and testing on the left, and vice versa) as in [40], and leave-one-sub-video-out classification for DynTex-35, with one sub-video from all classes used for testing and the remainder for training [41].

5.4.3 Video classification results

Table 4 summarises the classification results on YUPENN, and compares to TSVQ [8] and various spatial, temporal, and spatiotemporal methods reported in [24]. BoS Tree achieves the best overall results with mean recognition rate of 85%, which is 16% higher than the best performing TSVQ (using

5. Not applying PCA obtained similar results, but with slower speed.

6. For YUPENN, we build a three-level tree with *BF* $\in \{16, 4, 13\}$ resulting in $\{16, 64, 832\}$ codewords at each level. For UCLA-8, we build a two-level tree with *BF* $\in \{16, 12\}$ resulting in $\{16, 192\}$ codewords at each level. For UCLA-39 and DynTex-35, we build 2-, 3- and 4-level trees with *BF* $\in \{64, 10\}, \{32, 2, 10\}, \{16, 2, 2, 10\}$ resulting in $\{64, 640\}$, $\{32, 64, 640\}$ and $\{16, 32, 64, 640\}$ codewords respectively at each level.

Scene classes	Color [35]	GIST [36]	HOF [37]	Chaos [38]	SOE [24]	TSVQ [8]	BoST
beach	50	**90**	37	27	87	63	83
c. street	47	50	83	17	83	70	**90**
elevator	83	53	93	40	67	73	**100**
f. fire	47	50	67	50	83	80	**100**
fountain	13	40	30	7	47	37	**67**
highway	30	47	33	17	77	73	**87**
l. storm	83	57	47	37	90	80	**100**
ocean	73	93	60	43	**100**	80	90
railway	43	50	83	3	**87**	73	80
r. river	57	63	37	3	**93**	73	80
sky	30	90	83	33	90	77	**93**
snowing	53	20	57	10	33	77	**83**
waterfall	30	33	60	10	43	53	**67**
w. farm	57	47	33	17	57	57	**77**
Avg. (%)	50	56	59	20	74	69	**85**

TABLE 4: Comparison of BoS Tree classification rates with TSVQ [8] and various spatial and temporal methods reported in [24].

3-level tree with TF and CSK-SVM) and 11% higher than the previous state-of-the-art results, based on spatiotemporal oriented energy (SOE), reported in [24]. Looking at the individual scene classes, BoS Tree obtains the highest accuracy on 10 out of 14 classes. Table 5 shows the confusion matrices for BoS Tree and SOE from [24]. SOE exhibits the most confusion between the snowing, waterfall, fountain, and elevator classes, which have similar directional components. In contrast, BoST mainly confuses classes with similar water textures, e.g., fountain vs. waterfall, and river vs. beach vs. ocean. Note that SOE is based on color features and spatial context information (through spatial binning), whereas our BoS tree uses only grayscale and does not encode spatial context. Hence, our results suggest that modeling the temporal dynamics of the video (e.g., with DTs) can improve the recognition of these scenes, even without accounting for spatial configurations or color information.

Table 6 reports classification results on UCLA-8, UCLA-39 and DynTex-35 using various classifiers and techniques to build BoS histograms. Each row refers to the combination of a specific classifier with TF or TF-IDF representation, while columns correspond to different techniques to map videos to BoS histograms (large BoS, reduced BoS, and BoST).

Several observations can be made from the results on UCLA-39, which is the most challenging dataset. First, using a codebook with a larger number of codewords substantially increases the classification performance, e.g., with accuracy increasing from 41.35% for 16 codewords to 81.73% for 624 codewords, using TF-IDF and HIK-SVM. However, the computational cost also increases substantially by a factor of 39 times (from 7377 likelihood computations per video, about 5 seconds on a standard desktop PC, to 287,690 or about 182 seconds). Second, using BoST leads to the highest accuracy while requiring only a fraction of the likelihood computations necessary when directly indexing a large codebook. For example with TF-IDF and HIK-SVM, using a 2-level codebook improves accuracy to 82.37%, while also decreasing the average number of likelihood computations by a factor of 8 (36,393 computations or 23 seconds). For other classifiers, the accuracy is on par, or decreases slightly, compared to the large CB. These results demonstrate that the BoST efficiently and effectively indexes codewords, and hence allows practical use of a large and rich codebook. Third, although they use about the same number of likelihood operations, BoST significantly outperform the reduced codebooks generated with HEM-DTM in terms of classification accuracy. While the former leverages the hierarchical structure of codewords to access a large collection of codewords, the latter only reduces the size of the codebook which does not result in a BoS rich enough to produce highly accurate classification. Lastly, using the BoST with $L = 4$ and setting the traversing threshold in (9) and (10) to $\mathfrak{T} = 0.995$ leads to the best performance. By executing a limited number of additional likelihood computations (only 30% more than BoST with $L = 4$), the threshold method is able to explore the sub-trees of similar codewords when the patch has near equal preference to both.

Looking at UCLA-8, BoST achieves best overall accuracy of 97.28% using TF-IDF and CSK-SVM, which is equal to the performance of large BoS with direct indexing. However, BoST reduces the computations by a factor of 6.6, compared to the large BoS. Similar observations can be drawn on DynTex-35, although the differences in classification accuracy are less substantial due to the easiness of the classification task. Figure 5 shows the speed vs. performance graphs using BoS codebooks of various sizes and BoS Trees of varying heights for UCLA-39 and DynTex-35. Note that BoST achieves similar

(a)

Actual \ *Classified*	Sky	Beach	Ocean	Street	Railway	R. River	Highway	Snowing	Waterfall	Fountain	L.Storm	F. Fire	W. Farm	Elevator
Sky	**28**	1			1									
Beach		**25**	1			4								
Ocean		2	**27**			1								
Street				**27**	1		2							
Railway	2			3	**24**							1		
R. River			5	1		**24**								
Highway				3			**26**					1		
Snowing				1				**25**	1	2		1		
Waterfall						1	1		**20**	8				
Fountain		1							8	**20**		1		
L.Storm											**30**			
F. Fire												**30**		
W. Farm		1		1				1	1	1		1	**23**	1
Elevator														**30**

(b)

Actual \ *Classified*	Sky	Beach	Ocean	Street	Railway	R. River	Highway	Snowing	Waterfall	Fountain	L.Storm	F. Fire	W. Farm	Elevator
Sky	**27**		1		1	1								
Beach		**26**	3			1								
Ocean			**30**											
Street				**25**		3			1			1		
Railway		1			**26**	2						1		
R. River		1				**28**	1							
Highway		1		1		3	**23**		2					
Snowing				1		6		**10**	4	2		3	1	3
Waterfall						2		2	**13**	7		3	2	1
Fountain								3	9	**14**		4		
L.Storm											**27**	3		
F. Fire								2	1		2	**25**		
W. Farm								3	3	1	1	5	**17**	
Elevator								2	3	4		1		**20**

TABLE 5: Confusion matrix for YUPENN data set using (a) a three-level BoS tree and (b) SOE ($4 \times 4 \times 1$). Bold shows the number of correct classifications for each scene category.

TABLE 6: Video classification results on UCLA-8, UCLA-39 and DynTex-35 using a large codebook, reduced codebooks, and BoS trees. Each row reports the average classification accuracy of a different classifier/kernel combination. The final three rows report the speedup relative to large BoS to build the BoS histograms at test time, and the TSVQ and reference results (Ref).

Method			UCLA-8				UCLA-39								DynTex-35						
			large BoS	reduced BoS		BoST $\kappa=1$	large BoS	reduced BoS			BoST $\kappa=1$			$\Upsilon=0.995$	large BoS	reduced BoS			BoST $\kappa=1$		
			$\lvert\mathcal{C}\rvert=184$	K = 16	K = 8	L = 2	$\lvert\mathcal{C}\rvert=624$	K = 64	K = 32	K = 16	L = 2	L = 3	L = 4	L = 4	$\lvert\mathcal{C}\rvert=630$	K = 64	K = 32	K = 16	L = 2	L = 3	L = 4
T	N	CS	95.98	94.89	91.20	95.98	46.79	46.79	42.95	33.97	43.59	46.47	42.31	42.63	92.86	94.86	92.86	90.86	91.14	92.00	90.57
T	N	SR	96.74	96.30	92.28	96.09	62.82	52.88	42.31	34.94	60.90	58.01	55.13	58.33	98.00	98.57	97.71	94.57	98.29	98.00	98.29
F	S	CSK	97.17	95.33	86.41	97.07	62.82	52.88	44.87	33.33	60.26	58.97	57.05	59.29	98.29	97.14	96.29	89.43	98.00	98.29	98.86
	V	HIK	75.11	83.04	75.54	81.96	78.53	57.69	48.40	39.10	78.53	78.53	73.72	78.85	96.86	96.29	91.71	86.29	97.71	97.71	97.14
	M	BCK	81.09	85.43	70.87	89.13	71.79	52.88	45.19	38.78	71.15	71.15	69.55	72.12	96.57	94.57	92.57	84.29	96.29	96.86	96.57
T	N	CS	95.98	94.89	92.07	95.98	46.15	45.83	42.95	34.62	43.59	46.15	41.99	42.31	92.29	94.86	92.86	91.14	90.57	92.00	90.86
F	N	SR	96.74	95.98	92.83	96.41	65.38	56.09	45.19	36.22	61.22	58.97	56.41	60.58	98.00	98.29	97.43	94.29	98.00	98.00	98.00
I	S	CSK	97.28	94.57	83.91	97.28	61.22	53.53	45.51	37.50	61.86	59.94	59.62	60.90	98.29	97.14	96.29	89.43	97.71	97.43	98.00
D	V	HIK	74.13	81.85	69.78	81.85	81.73	58.01	48.40	41.35	82.37	82.37	79.81	83.33	97.14	96.57	92.29	85.43	97.43	97.43	97.71
F	M	BCK	80.43	84.02	67.39	89.24	74.36	55.13	51.92	40.06	73.72	74.04	72.76	75.32	96.57	95.14	91.71	82.57	96.57	96.29	96.29
Best			97.28	96.30	92.83	97.28	81.73	58.01	51.92	41.35	82.37	82.37	79.81	83.33	98.29	98.57	97.71	94.57	98.29	98.29	98.86
X-Speedup			1	11.50	23	6.66	1	9.75	19.50	39	7.91	12.46	17.39	12.74	1	9.84	19.69	39.37	8.31	13.56	19.47
Ref			80 [1], 52.27 [9], 84 [42]				42.3 [39], 20 [40], 15 [10]								97.14 [41]						
TSVQ [8]			96.63				57.69								96.57						

accuracy to the direct-indexed large BoS, while reducing the computation by almost an order of magnitude. By organizing codewords in a hierarchical structure, the BoS Tree efficiently budgets the likelihood computations while indexing a larger and more descriptive codebook.

Finally, BoST performance improves on the current state-of-the-art reported in the literature [1, 9, 42, 39, 40, 10, 41] on the three textures datasets (last row of Table 6). On UCLA-8, the accuracy has improved from 52.27% [9] or 84% [42] to 97.28% for BoST. On UCLA-39, the accuracy has improved from 20% [40] or 42.3% [39] to 82.37% for BoST. In contrast to [40], which is based solely on motion dynamics, and [39], which models local appearance and instantaneous motion, the BoS representation is able to leverage both the local appearance (for translation invariance) and motion dynamics of the video to improve the overall accuracy.

Table 6 also shows the results for the best performing TSVQ, which are the 2, 3 and 4 level TSVQ for UCLA-8, UCLA-39 and DynTex-35, respectively. Similar to other methods TSVQ also performs poorly on UCLA-39 (accuracy of 57.69%) compared with BoST. On UCLA-8 and DynTex-35, TSVQ has better accuracy (96%), but still worse than BoST. The reason that TSVQ can achieve relatively high accuracy on UCLA-8 and DynTex-35 is because the training and test videos have very similar appearances, whereas for UCLA-39 the appearances are more varied.

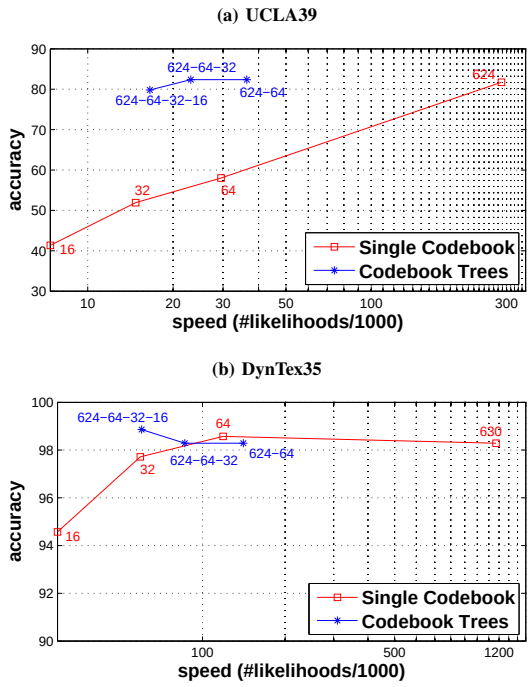


Fig. 5: Speed/accuracy tradeoff for BoS and BoS Trees.

Finally, we compare the top-down and bottom-up approaches for building a BoS Tree. On the UCLA39 dataset, we built a two-level top-down BoST with $\{640, 64\}$ codewords at each level, which has a similar size to the bottom-up BoST with $\{624, 64\}$ codewords in Table 6. Interestingly, the top-down BoST can only achieve accuracy of 68.27%, compared to 82.37% of the bottom-up version. Also building the BoST with the top-down approach is almost 6 times more expensive than the bottom-up approach. The top-down BoST requires 2993 total minutes (over all CPUs) versus 510 minutes for the bottom-up BoST. Hence, these results suggest that the bottom-up approach is a more efficient and more robust method for building the BoS Trees as compared to the top-down approach.

5.5 Music annotation

In this section, we show the applicability of BoS Trees on an additional type of time-series data, i.e., musical signals.

5.5.1 Dataset

We perform automatic music annotation on the CAL500 dataset (details in [43] and references therein), which is a collection of 502 popular Western song, and provides binary annotations with respect to a vocabulary of musically relevant tags (annotations or labels), *e.g.*, rock, guitar, romantic. In our experiments we follow the same protocol as [28] and consider the 97 tags associated to at least 30 songs in CAL500 (11 genre, 14 instrumentation, 25 acoustic quality, 6 vocal characteristics, 35 mood and 6 usage tags).

5.5.2 Experiment setup

The acoustic content of a songs is represented by a time-series of 34-bin Mel-frequency spectral features (see [43]), extracted over half-overlapping windows of 92 ms of audio signal. A dense sampling of audio-fragments (analogous to spatio-temporal cubes in videos) is then formed by collecting sequences of $\tau = 125$ consecutive feature vectors (corresponding to approximately 6 seconds of audio), with 80% overlap.

TABLE 7: Music annotation results on CAL500, using a large codebook, reduced codebooks, and BoS Trees. The last column reports the speedup relative to large CB to build the BoS histograms at test time.

		Retrieval			Annotation			
		MAP	AROC	P@10	P	R	F	X-Speedup
large CB	K = 1604	0.454	0.723	0.460	0.406	0.244	0.270	0.97
reduced CB	K = 128	0.403	0.668	0.402	0.342	0.209	0.227	12.18
	K = 64	0.381	0.649	0.378	0.315	0.192	0.204	24.37
	K = 32	0.368	0.634	0.368	0.298	0.180	0.191	48.74
BoS Tree	L = 2	0.445	0.712	0.451	0.398	0.24	0.261	8.00
	L = 3	0.443	0.712	0.448	0.393	0.235	0.258	11.20
	L = 4	0.439	0.711	0.448	0.394	0.232	0.255	13.89
SML-DTM [43]		0.446	0.708	0.460	0.446	0.217	0.264	1

A BoS Tree is learned for each cross-validation split from only the training data. The first level of the BoS Tree is formed by estimating a DTM with $K_s = 4$ components from each training song, and then pooling all the DT components together. BoS Trees for $L \in \{2, 3, 4\}$ levels are tested, with $K_2 = 128$, $K_3 = 64$ and $K_4 = 32$. We use $\kappa^{(1)} = 5$ and $\kappa^{(\ell)} = 2$ for $\ell > 1$. We used the TF-IDF representation.

We cast music annotation as a multi-class multi-label classification task. In particular, given a training set of audio-fragments and their annotations, for each tag we use logistic regression (LR) to learn a linear classifier with a probabilistic interpretation in tandem with the histogram intersection kernel. Given a BoS descriptor corresponding to a new song, the output of the LR classifiers is normalized to a semantic multinomial, *i.e.*, a vector of tag posterior probabilities. We use the LibLinear software package [44] for the LR classifier, with all parameters selected using 4-fold cross validation on the training set.

On the test set, a novel test song is annotated with the 10 most likely tags, corresponding to the peaks in its semantic multinomial. Retrieval given a one tag query involves rank ordering all songs with respect to the corresponding entry in their semantic multinomials. Performance is measured with the same protocol as in [28]: for annotation, per-tag precision (P), recall (R) and F-score (F), averaged over all tags; and for retrieval, mean average precision (MAP), area under the operating characteristic curve (AROC), and precision at the first 10 retrieved objects (P@10), averaged over all one-tag queries. In addition, we register the average (per song) number of likelihood computations executed at test time. All reported metrics are result of 5-fold cross validation, where each song appears in the test set exactly once. We compare our BoS Tree to recent results in music annotation based on a large BoS codebook [28], and supervised multi-class labeling with DTM models (SML-DTM) [43].

5.5.3 Music annotation results

In Table 7 we report annotation and retrieval performance on the CAL500 dataset. We first note that the BoS Trees lead to near optimal performance with respect to the direct-indexed large codebook (implemented with $k = 5$ as in [28]) and SML-DTM, but require an order of magnitude less likelihood computations at test time. In particular, in our experiments, the delay associated to likelihood computations was 8 to 14 seconds per song for the BoS Trees (depending on L), and 2 minutes for direct-indexed large codebook and for SML-DTM. Second, as with video annotation and classification, increasing the codebook size improves the accuracy of music annotation and retrieval, by increasing the richness of the codebook. Again, this justifies the efficacy of large BoS codebooks and our proposed BoS Tree for efficient indexing.

6 Analysis of the BoS Tree Descriptor

Experimental results presented in the previous sections show that the BoS Tree (BoST) can achieve similar classification performance to direct-indexed BoS framework, but with one order of magnitude reduced computational cost. In this section, we perform a detailed analysis to investigate how well the indexing tree can recover the true BoS descriptor, and its effect on the distance (kernel) matrix. Here we focus our analysis on the large BoS (624 codewords) and BoST (two level 624-64) extracted from UCLA-39, although similar results were obtained for other datasets. In the experiments in Section 5.4, the BoS and BoST obtain similar average classification accuracy (81.73% and 82.37%) using the histogram intersection kernel. We next present an analysis moving from a high-level perspective (e.g., similarity in distance matrices and nearest-neighbor rankings) to low-level details (e.g., differences in codeword assignment for individual patches).

6.1 Comparing distance matrices and NN ranks

As the NN (SVM) classifiers depend on distance (kernels), we first examine the similarities between the distance matrices produced using either BoS or BoST descriptors. Figure 6(a) and 6(b) show the intersection distance[7] matrices for UCLA-39 using the BoS descriptors and the BoST descriptors, respectively. The structure of the two distance matrices are quite similar, and can be further visualized with the scatter plot in Figure 7a. Quantitatively, the two distance matrices are highly correlated, with a Pearson correlation coefficient of $\rho = 0.982$. Therefore, the kernel matrices are also highly correlated, and hence similar classification performance (about 80%) is achieved in both BoS and BoST experiments.

Similar distance matrices imply that the nearest neighbor structure between videos for BoS and BoST should also be similar. For each video, the list of the top-K nearest neighbors were calculated using BoS and BoST. The two lists are then compared by calculating their set intersection. Figure 8 plots the set intersection values for different levels of K and averaged over all the videos. The 1st nearest neighbor is the same 74% of the time, whereas there is 87% overlap for the top-3 nearest neighbors. The high values of intersection for small K indicate that the *local* nearest neighbor structure is preserved well. For larger K, the average intersection value steadily increases, indicating an improvement in overlapping and preservation of the global structure. From these results we conclude that, at the high-level, both BoS and BoST descriptors generate similar kernel/distance matrices, and thus are very similar for the purpose of classification.

6.2 Comparing descriptors

In this section we investigate more closely the relation between the BoS and BoST histograms. Note, that similar structure

7. The intersection distance is defined as $1 - k_I(h_1, h_2)$.

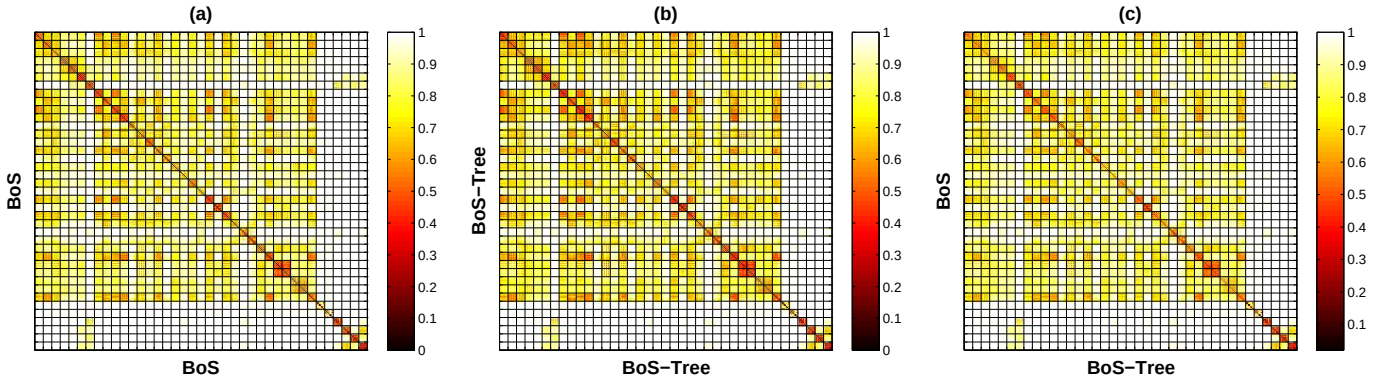


Fig. 6: Intersection distance matrix between (a) BoS histograms, (b) BoST histograms, (c) BoS and BoST histograms, for all videos in UCLA-39. Videos are grouped by class, delineated by black lines.

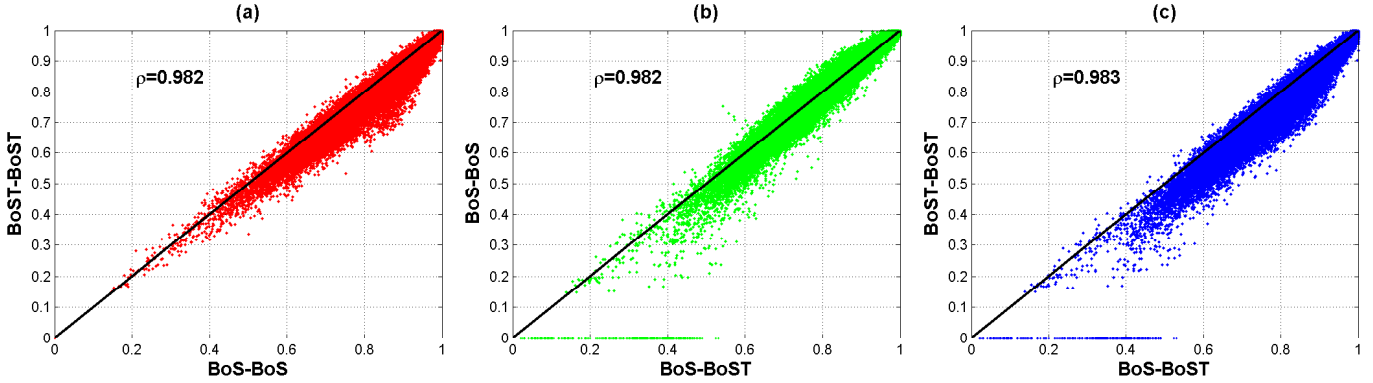


Fig. 7: Scatter plots between the distance matrices from Figure 6.

in the kernel/distance matrices (observed in the previous section) does not necessarily imply similarity of BoS and BoST histograms. We next compare the BoS and BoST histograms extracted from the same video. Figure 9 plots the distance between BoS and BoST histograms for each video. The average intersection distance between BoS and BoST is about 0.30, which indicates that the two descriptors extracted from the same video are similar, but not exactly the same. Figure 6c shows the full distance matrix between BoS and BoST histograms using the intersection distance (Figure 9 is the reordered diagonal of this matrix). The distance matrix looks quite similar to the distance matrices for BoS or BoST in Figures 6a and 6b. The scatter plot between the BoS-BoS distance matrix and BoS-BoST distance matrix appears in Figure 7b, and indicates high correlation between the matrices ($\rho = 0.982$). Similar results are obtained when comparing the BoST-BoST distance matrix and that of BoS-BoST (Figure 7c). This high correlation suggests that, although the BoS and BoST descriptors for a particular video are not exactly the same, the distances between videos are still preserved well.

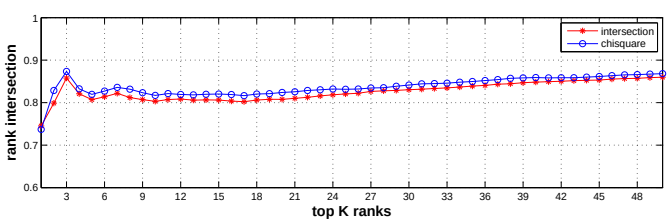


Fig. 8: Average set intersection between top-K nearest neighbors using BoS or BoST histograms, for all videos in UCLA39.

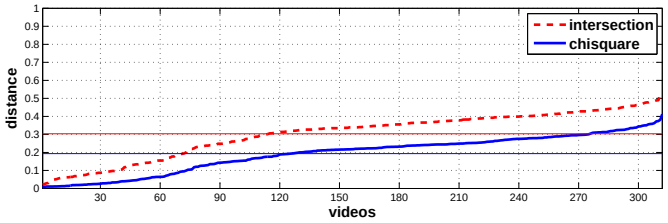


Fig. 9: Distances between BoS and BoST histograms for each video in UCLA-39. Videos are sorted according to the distance. The horizontal line is the average distance.

6.3 Differences in codeword assignments

Finally, to understand how the BoS and BoST descriptors vary, we next examine the differences in how patches are assigned to codewords for BoS and BoST. For each video, the number of patches that are assigned to the *different* codewords in BoS and BoST is counted, and the percentage of disagreement is calculated for each video,

$$\text{diff} = \frac{\#\text{ of disagreements}}{\#\text{ of patches}}. \quad (14)$$

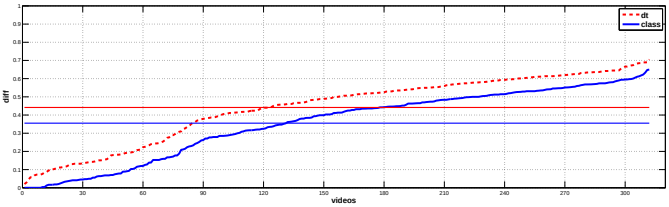


Fig. 10: Percentage of patches that are assigned to different codewords in BoS and BoST histograms for UCLA-39. Videos are sorted by increasing assignment of patches.

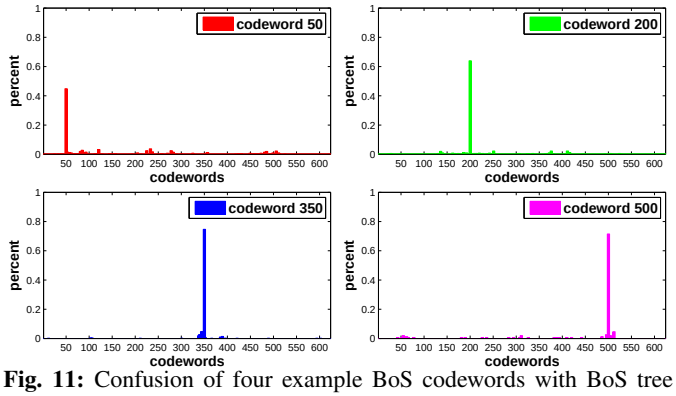


Fig. 11: Confusion of four example BoS codewords with BoS tree codewords.

Figure 10 shows the differences in codeword assignment for videos in UCLA-39. On average, 44.2% of the patches are assigned to different codewords in BoS and BoST.

To evaluate how this affects classification, we next allow a patch to be assigned to codewords from videos in the same class (16 codewords). Figure 10 also plots the percentage of disagreement of patches assigned to codewords of different classes. At the class-level, 35.5% of patches are assigned to codewords of videos in another class. Hence, for the patches that are assigned differently, about 8.7% are assigned to codewords belonging to videos in the same class.

To examine the misassignment of patches, we compute the confusion matrix between BoS and BoST codewords where the (i, j) entry is obtained by counting all patches assigned to the i^{th} codeword in the BoS codebook and the j^{th} codeword in the BoST codebook. Figure 11 shows four typical example rows of the confusion matrix, representing confusion between a particular BoS codeword (actual) and all the BoST codewords (assigned). The peak in each plot indicates the percentage of correct assignments for the codeword (on average 55.8%), which is surrounded by assignments to within-class codewords (on average 8.7%). The remaining assignments (on average 35.5%) are spread uniformly at random to the out-of-class codewords.[8]

In summary, from the analysis in this section, we can conclude that the BoS Tree can successfully generate a descriptor that is equivalent to the original BoS descriptor, for the purposes of classification. At the low-level, the BoS Tree introduces uniform random noise to the descriptor. However, this does not affect the structure of the kernel/distance matrices. Therefore, the BoST descriptor can be used in place of BoS in order to decrease the computational requirements, while maintaining similar levels of classification accuracy.

7 Conclusions

In this paper we have proposed the BoS Tree, which efficiently indexes DT codewords of a BoS representation using a hierarchical indexing structure. The BoS Tree enables the practical use of larger and richer collections of codewords in the BoS representation. We demonstrated the efficacy of the BoS Tree on video classification of four datasets, as well as on annotation of a music and video dataset. In particular, the BoS Tree achieves similar accuracy to the direct-indexed large BoS, while reducing the computation by almost an order of magnitude. Finally, we showed that, although the BoS Tree and BoS descriptors are different for the same video, the overall kernel (distance) matrices are highly correlated thus leading to similar classification performance. In particular, the BoS Tree adds uniform random noise to the descriptor, which does not significantly affect the structure of the kernel matrix. Finally, extending [22] to perform nearest-neighbor search based on log-likelihoods (rather than KL divergence) or adapting approximate search using randomized trees [19] to time-series (using DTs) would be interesting future work.

8. To show that the confusion among out-of-class codewords is close to a uniform distribution, we first aggregated the out-of-class confusions in all codewords to form an overall distribution of confusion. The entropy of this distribution was 8.9, which is close to the maximum possible value of 9.3 for a uniform distribution.

Acknowledgments

The authors thank R. Péteri for the DynTex dataset, and G. Doretto for the UCLA dataset. A.M. and A.B.C. were supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (CityU 110610 & CityU 110513). E.C., A.B.C. and G.R.G.L. acknowledge support from Google, Inc. EC and GRGL acknowledge support from Qualcomm, Inc., Yahoo! Inc., KETI under the PHTM program, and NSF Grants CCF-0830535 and IIS-1054960. G.R.G.L. acknowledges support from the Alfred P. Sloan Foundation. This research was supported in part by the UCSD FWGrid Project, NSF Research Infrastructure Grant Number EIA-0303622. This research was conducted in part using the resources of the HPCCC, HKBU, which receives funding from Research Grant Council, University Grant Committee of the HKSAR and HKBU.

References

[1] A. Ravichandran, R. Chaudhry, and R. Vidal, "View-invariant dynamic texture recognition using a bag of dynamical systems," in *CVPR*, 2009.
[2] A. B. Chan, E. Coviello, and G. R. G. Lanckriet, "Clustering dynamic textures with the hierarchical em algorithm," in *CVPR*, 2010.
[3] A. Mumtaz, E. Coviello, G. Lanckriet, and A. Chan, "Clustering dynamic textures with the hierarchical em algorithm for modeling video," *IEEE TPAMI*, vol. 35(7), pp. 1606–1621, 2013.
[4] B. Afsari, R. Chaudhry, A. Ravichandran, and R. Vidal, "Group action induced distances for averaging and clustering linear dynamical systems with applications to the analysis of dynamic scenes," in *CVPR*, 2012, pp. 2208–15.
[5] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto, "Dynamic textures," *Intl. J. Computer Vision*, 2003.
[6] P. Saisan, G. Doretto, Y. Wu, and S. Soatto, "Dynamic texture recognition," in *CVPR*. IEEE, 2001.
[7] K. Ellis, E. Coviello, A. Chan, and G. Lanckriet, "A bag of systems representation for music auto-tagging," *IEEE TASLP*, vol. PP, no. 99, pp. 1–1, 2013.
[8] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *CVPR*, 2006.
[9] P. Saisan, G. Doretto, Y. Wu, and S. Soatto, "Dynamic texture recognition," in *CVPR*, vol. 2, 2001, pp. 58–63.
[10] A. B. Chan and N. Vasconcelos, "Probabilistic kernels for the classification of auto-regressive visual processes," in *CVPR*, 2005.
[11] F. Woolfe and A. Fitzgibbon, "Shift-invariant dynamic texture recognition," in *ECCV*, 2006.
[12] K. G. Derpanis and R. P. Wildes, "Dynamic texture recognition based on distributions of spacetime oriented structure," in *CVPR*, 2010.
[13] G. Zhao and M. Pietikainen, "Dynamic texture recognition using local binary patterns with an application to facial expressions," *IEEE TPAMI*, 2007.
[14] A. Gersho and R. Gray, *Vector quantization and signal compression*. Springer Netherlands, 1992, vol. 159.

[15] K. Grauman and T. Darrell, "Approximate correspondences in high dimensions," *NIPS*, 2007.
[16] J. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, pp. 509–517, 1975.
[17] J. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Information Processing Letters*, 1991.
[18] L. Cayton, "Fast nearest neighbor retrieval for bregman divergences," in *ICML*, 2008.
[19] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *CVPR*, 2007.
[20] V. Lepetit, P. Lagger, and P. Fua, "Randomized trees for real-time keypoint recognition," in *CVPR*, 2005, pp. 775–781.
[21] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327494
[22] E. Coviello, A. Mumtaz, A. Chan, and G. Lanckriet, "That was fast! speeding up nn search of high dimensional distributions." in *ICML*, 2013, pp. 468–476.
[23] N. Vasconcelos, "Image indexing with mixture hierarchies," in *CVPR*, 2001.
[24] K. Derpanis, M. Lecce, K. Daniilidis, and R. Wildes, "Dynamic scene understanding: The role of orientation features in space and time in scene classification," in *CVPR*, 2012.
[25] E. Coviello, A. Mumtaz, A. Chan, and G. Lanckriet, "Growing a bag of systems tree for fast and accurate classification," in *CVPR*, june 2012, pp. 1979 –1986.
[26] R. Martin, "A metric for arma processes," *Signal Processing, IEEE Transactions on*, vol. 48, no. 4, pp. 1164–1170, Apr 2000.
[27] A. B. Chan and N. Vasconcelos, "Modeling, clustering, and segmenting video with mixtures of dynamic textures," *IEEE TPAMI*, 2008.
[28] K. Ellis, E. Coviello, and G. R. G. Lanckriet, "Semantic annotation and retrieval of music using a bag of systems representation," in *Proc. ISMIR*, 2011.
[29] R. H. Shumway and D. S. Stoffer, "An approach to time series smoothing and forecasting using the EM algorithm," *Journal of Time Series Analysis*, 1982.
[30] K. Ellis, E. Coviello, A. Chan, and G. Lanckriet, "A bag of systems representation for music auto-tagging," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 21, no. 12, pp. 2554–2569, Dec 2013.
[31] R. Péteri, S. Fazekas, and M. J. Huiskes, "DynTex: A comprehensive database of dynamic textures," *Pattern Recognition Letters*, vol. 31, no. 12, pp. 1627–32, 2010.
[32] V. N. Vapnik, *The nature of statistical learning theory*. Springer-Verlag, 1995.
[33] C. Chang and C. Lin, "Libsvm: a library for support vector machines," *ACM TIST*, 2011.
[34] G. Carneiro, A. B. Chan, P. J. Moreno, and N. Vasconcelos, "Supervised learning of semantic classes for image annotation and retrieval," *IEEE TPAMI*, vol. 29(3), pp. 394–410, 2007.
[35] S. Grossberg and T. R. Huang, "Artscene: A neural system for natural scene classification," *Journal of Vision*, vol. 9, no. 4, pp. 1–19, 2009, article 6.
[36] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, pp. 145–175, 2001.
[37] M. Marszalek, I. Laptev, and C. Schmid, "Actions in context," in *CVPR*, 2009, pp. 2929–2936.
[38] N. Shroff, P. Turaga, and R. Chellappa, "Moving vistas: Exploiting motion for describing scenes," in *CVPR*, 2010.
[39] K. Derpanis and R. Wildes, "Dynamic texture recognition based on distributions of spacetime oriented structure," in *CVPR*, 2010.
[40] F. Woolfe and A. Fitzgibbon, "Shift-invariant dynamic texture recognition," *Computer Vision–ECCV*, 2006.
[41] G. Zhao and M. Pietikainen, "Dynamic texture recognition using local binary patterns with an application to facial expressions," *IEEE TPAMI*, vol. 29, no. 6, pp. 915 –928, june 2007.
[42] A. Ravichandran, R. Chaudhry, and R. Vidal, "Categorizing dynamic textures using a bag of dynamical systems," *IEEE TPAMI*, vol. PP, no. 99, p. 1, 2012.
[43] E. Coviello, A. Chan, and G. Lanckriet, "Time Series Models for Semantic Music Annotation," *IEEE TASLP*, 2011.
[44] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin, "Liblinear: A library for large linear classification," *JMLR*, 2008.

Adeel Mumtaz received the B.S. degree in computer science from Pakistan Institute of Engineering and Applied Sciences and the M.S. degree in computer system engineering from Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Pakistan, in 2004 and 2006, respectively. He is currently working toward the PhD degree in Computer Science at the City University of Hong Kong. He is currently with the Video, Image, and Sound Analysis Laboratory , Department of Computer Science, CityU. His research interests include Computer Vision, Machine Learning and Pattern recognition.

Emanuele Coviello received the "Laurea Triennale" degree in information engineering and the "Laurea Specialistica" degree in telecommunication engineering from the Università degli Studi di Padova, Padova, Italy, in 2006 and 2008, respectively, and the Ph.D. degree in electrical and computer engineering, from the University of California, San Diego (UCSD), in 2014. Dr. Coviello received the "Premio Guglielmo Marconi Junior 2009" award, from the Guglielmo Marconi Foundation (Italy), and won the "2010 Yahoo! Key Scientific Challenge Program" sponsored by Yahoo!. His main interest is machine learning applied to content based information retrieval and multimedia data modeling, to analyze, understand and organize video, audio and music content. Dr. Coviello currently serves as founder and CEO of Keevio, Inc.

Gert Lanckriet received the M.S. degree in electrical engineering from the Katholieke Universiteit Leuven, Leuven, Belgium, in 2000 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 2001 and 2005, respectively. In 2005, he joined the Department of Electrical and Computer Engineering, University of California, San Diego, where he heads the Computer Audition Laboratory. His research focuses on the interplay of convex optimization, machine learning, and signal processing, with applications in computer audition and music information retrieval. Prof. Lanckriet was awarded the SIAM Optimization Prize in 2008 and is the recipient of a Hellman Fellowship, an IBM Faculty Award, an NSF CAREER Award and an Alfred P. Sloan Foundation Research Fellowship. In 2011, MIT Technology Review named him one of the 35 top young technology innovators in the world (TR35).

Antoni B. Chan received the B.S. and M.Eng. degrees in electrical engineering from Cornell University, Ithaca, NY, in 2000 and 2001, respectively, and the Ph.D. degree in electrical and computer engineering from the University of California, San Diego (UCSD), San Diego, in 2008. From 2001 to 2003, he was a Visiting Scientist with the Vision and Image Analysis Laboratory, Cornell University, Ithaca, NY, and in 2009, he was a Postdoctoral Researcher with the Statistical Visual Computing Laboratory, UCSD. In 2009, he joined the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong, as an Assistant Professor. His research interests include computer vision, machine learning, pattern recognition, and music analysis. Dr. Chan was the recipient of an NSF IGERT Fellowship from 2006 to 2008, and an Early Career Award in 2012 from the Research Grants Council of the Hong Kong SAR, China.