Appears in IEEE Conf. on Computer Vision and Pattern Recognition, Providence, 2012.

Growing a Bag of Systems Tree for Fast and Accurate Classification

Emanuele Coviello[1] Adeel Mumtaz[2] Antoni B. Chan[2] Gert R.G. Lanckriet[1]
[1]Dept. of ECE, University of California, San Diego [2] Dept. of CS, City University of Hong Kong

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 directly 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 three video datasets, as well as on annotation of a music dataset.

1. Introduction

The bag-of-system (BoS) representation [1], a *high-level* descriptor of motion in a video, has seen promising results in video texture classification [2]. 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 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 a codebook of dynamic texture (DT) models was used to 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 [3]. 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 [4], 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 codebook size 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.

To address the computational challenges of the BoS representation, in this paper we propose the *BoS Tree*, which combines the expressiveness of a large codebook with the efficiency of a small codebook. Our proposed approach constructs a bottom-up hierarchy of codewords, and then leverages the tree structure to efficiently index the code-

words 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 contributions of this paper are four-fold. First, we propose the BoS Tree for fast-indexing of large BoS codebooks. Second, we apply the BoS Tree to video classification, reducing the computational cost by at least one order of magnitude versus a standard large codebook, without loss in performance. In fact, the BoS tree *improves* on previous state-of-the-art results on two video texture datasets. Finally, we apply BoS Trees to music annotation experiments, showing suitability of the method to other types of time-series data. The remainder of this paper is organized as follows. In Section 2, we first review the BoS representation and DT model, while in Section 3 we propose the BoS Tree. Finally, we present experiments on video classification in Section 4, and on music annotation in Section 5.

2. The BoS representation

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

2.1. The DT codeword

In general, the content of a video is represented by a set of T time series of low-level feature vectors $\mathcal{Y} = \{y_{1:\tau}^{(1)}, \ldots, y_{1:\tau}^{(T)}\}$, which correspond to spatio-temporal cubes sampled from the video, where T depends on the size of the video and the granularity of the sampling process. Each time series $y_{1:\tau}^{(t)}$ is composed of τ patches extracted at 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 the driving respectively observation noises, the vector $\bar{y} \in \mathbb{R}^n$ is the mean feature vector, and $\mathcal{N}(\mu, S)$ specifies the initial condition.

2.2. Learning the codebook

The BoS codebook $\mathcal{C}$ is learned from a codebook 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 interest point operators are used to extract interesting motion patches, from which DTs are learned. The video DTs are then embedded into a Euclidean space via non-linear dimensionality reduction in tandem with the Martin distance. 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 closest to the cluster centers in the embedding, are selected as the codewords.

An alternative approach, presented in [2], 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 [6]. [2] then directly clusters the video DTs using the hierarchical EM algorithm, producing novel DT cluster centers that are used as the codewords.

While these two approaches effectively produce small-sized codebooks (both [1, 2] use 8 codewords), 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 Sections 4 and 5. A final approach [4] 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, but has significant computational cost when mapping to the codebook.

2.3. Projection to the codebook

Once a codebook is available, 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 of $\mathcal{Y}$, we extract a dense sampling of spatio-temporal cubes from $\mathcal{Y}$. Each cube $y_{1:\tau}^{(t)}$ is compared to each codeword $\Theta_i \in \mathcal{C}$ by mean of its likelihood, efficiently computed with the "innovations" form using the Kalman filter [7] . Defined a quantization threshold $k \in \{1, \ldots, |\mathcal{C}|\}$, 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

$$\mathbf{h}_\mathcal{Y}[i] = \frac{1}{|\mathcal{Y}|} \sum_{y_{1:\tau}^{(t)} \in \mathcal{Y}} \frac{1}{k} \mathbb{1}[i \in \operatorname*{argmax}_i^k P(y_{1:\tau}^{(t)} | \Theta_i)], \quad (3)$$

where argmax_i^k returns the indices of the codewords with the k-largest likelihoods. When the quantization threshold $k = 1$, 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 $\mathbf{h}_\mathcal{Y}$ for $\mathcal{Y}$, the corresponding TF-IDF representation is obtained with the following mapping:

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

where the IDF factor is computed as

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

and α normalizes the histogram.

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 $y_{1:\tau}^{(t)} \in \mathcal{Y}$ has to be 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 collections of codewords.

3. The BoS Trees

In this section we propose the *BoS Tree*, which consist 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 center 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.

3.1. The HEM algorithm

Given a collection of DTs, the HEM algorithm for DTMs (HEM-DTM) [2] 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, *i.e.*, $K^{(r)} < K^{(b)}$. Given the input DTM $\Theta^{(b)} = \{\Theta_i^{(b)}, \pi_i^{(b)}\}_{i=1}^{K^{(b)}}$, the likelihood of a spatio-temporal cube $y_{1:\tau}$ is given by

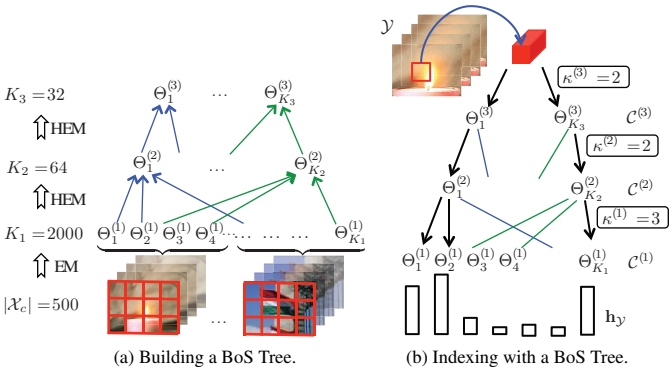
$$p(y_{1:\tau}|\Theta^{(b)}) = \sum_{i=1}^{K^{(b)}} \pi_i^{(b)} p(y_{1:\tau}|z^{(b)} = i, \Theta^{(b)}), \quad (6)$$

where $z^{(b)} \sim \text{multinomial}(\pi_1^{(b)}, \cdots \pi_{K^{(b)}}^{(b)})$ is the hidden variable that indexes the mixture components. $p(y_{1:\tau}|z = i, \Theta^{(b)})$ is the likelihood of $y_{1:\tau}$ under the i^{th} mixture component, and $\pi^{(b)}$ is the prior weight for the i^{th} component. The likelihood of the spatio-temporal cube $y_{1:\tau}$ given the reduced mixture $\Theta^{(r)} = \{\Theta_j^{(r)}, \pi_j^{(r)}\}_{j=1}^{K^{(r)}}$ is given by

$$p(y_{1:\tau}|\Theta^{(r)}) = \sum_{j=1}^{K^{(r)}} \pi_j^{(r)} p(y_{1:\tau}|z^{(r)} = j, \Theta^{(r)}), \quad (7)$$

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 $Y^i = \{y_{1:\tau}^m\}_{m=1}^{N_i}$ 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], while here we note that the output of the HEM algorithm is: (1) a clustering of the original $K^{(b)}$ components



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

Figure 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.

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.*, $\{\Theta_j^{(r)}\}_{j=1}^{K^{(r)}}$.

3.2. 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 1, left). 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 $\mathcal{X}_c$ (using the EM-DTM algorithm for DTMs [6]). Starting from the bottom level, a BoS 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 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

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

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.*, $\mathcal{T} = \{\mathcal{C}^{(1)}, \cdots, \mathcal{C}^{(L)}, B^{(2)}, \cdots, B^{(L)}\}$.

3.3. 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 1, right). 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 $y_{1:\tau}$ 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,

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

Next, the cube is propagated down to the successive level following the branches that depart from the codewords selected at the current level,

$$J^{(\ell)} = \operatorname*{argmax}_{i \in \bigcup_{j \in J^{(\ell+1)}} B_j^{(\ell+1)}}^{\kappa^{(\ell)}} p(y_{1:\tau} | \Theta_i^{(\ell)}), \quad (10)$$

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

$$\Omega(J, \mathfrak{T}) = \{j \in J | p(y_{1:\tau} | \Theta_j) \geq \mathfrak{T} \max_{h \in J} p(y_{1:\tau} | \Theta_h)\}$$

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

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

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

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 Sections 4 and 5.

3.4. Related Work

The bag-of-features "cousin" of our BoS Tree is the tree-structured vector quantizer (TSVQ) [8], 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 [9], and by Grauman *et al*. to define the bins of multi-resolution histograms [10]. The novelty of our paper is that we apply tree-structured search to a codebook formed by 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.

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 [11, 12]. Cayton [13] generalizes metric ball trees to Bregman divergences, enabling fast NN search of histograms using the KL divergence. Adapting approximate search using randomized trees [14] to time-series (using DTs) would be interesting future work.

The BoS Tree proposed here is similar to the Bregman-ball tree in [13], in that both use KL divergence-based clustering to hierarchically construct a tree. The main differences are that our BoS Tree is based on continuous probability distributions (in fact random processes), while [13] is limited to only discrete 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 [13] uses a more complicated branch-and-bound method. Experimentally, we found that the forward search was both efficient and accurate.

Finally, our BoS Tree is also related to fast image retrieval work by [15], where each image is modeled as a Gaussian distribution and a retrieval tree is constructed using the HEM algorithm for Gaussian mixture models.

4. Experiments: video classification

In this section we evaluate the proposed BoS Trees for video classification on three datasets.

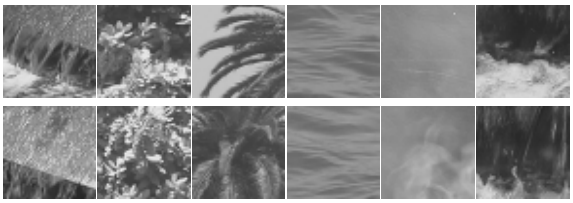


Figure 2. Example frames from UCLA-39. Right views (top) are visually different from the corresponding left views (bottom).

Figure 3. Example frames from KTH. There are 6 actions.

4.1. Datasets

We consider three datasets, the UCLA-39 dataset [16], the DynTex-35 dataset [17] and the KTH dataset [18]. The first two datasets are composed of video textures, while the third dataset consists in human actions.

UCLA-39: The UCLA-39 dataset [16] contains 312 gray-scale videos representing 39 *spatially stationary* classes, which were selected from the original 50 UCLA texture classes [3]. 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 a challenging task, and tests the translation invariance of the feature descriptor, since the training video patch (right portion) is visually quite different from the testing patch (left portion, or vice versa). While other UCLA-based datasets could be used (e.g., [3, 1, 19]), we believe that UCLA-39 is the most challenging variant and hence we adopt it in this paper. Figure 2 illustrates typical frames from UCLA-39.

DynTex-35: The DynTex-35 dataset [17] is a collection of videos from 35 different texture classes from everyday surroundings. Originally, the data consisted of a single video of size $192 \times 240 \times 50$ per class. As in [17] , each video is split into 10 non-overlapping sub-videos (each having different spatial and temporal dimensions).

KTH: The KTH dataset [18] consists of 2391 videos of six types of human actions (walking, jogging, running, boxing, hand waving and hand clapping) performed several times by 25 subjects in different scenarios (outdoors, outdoors with scale variation, outdoors with different clothes, and indoors). Each sequence is downsized to 160×120 pixels and have a length of 4 seconds in average. We follow the leave-one-out experimental protocol. Typical frames from KTH are illustrated in Figure 3. Note that, although the overall motion in KTH may not be stationary, we demonstrate that our BoS descriptor can represent local patches of motion sufficiently well (Section 4.3).

4.2. Experiment setup

From each video we extract a dense sampling of spatio-temporal cubes of size $5\times5\times75$ for UCLA-39, $7\times7\times30$ for

Table 1. 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)	$K(h_1,h_2) = 1 - \sum_k \frac{(h_{1k}-h_{2k})^2}{\frac{1}{2}(h_{1k}+h_{2k})}$
Intersection kernel	(HIK)	$K(h_1,h_2) = \sum_k \min(h_{1k},h_{2k})$
Bhattacharyya kernel	(BCK)	$K(h_1,h_2) = \sum_k \sqrt{h_{1k}h_{2k}}$

DynTex-35, and $7 \times 7 \times 10$ for KTH. We retain only video cubes with a minimum total variance of 1 for UCLA-39 and DynTex-35, and 5 for KTH.

For each cross-validation split in our datasets, the BoS Tree was learned from the training set only. For each video, a DTM with $K_v = 4$ components is learned from its spatio-temporal cubes using the EM-DTM algorithm. 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. We test 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, to represent the visual content of each video. We then use a k-nearest neighbor (k-NN) classifier or multi-class 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 1), 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 1. The LibSVM software package [20] was used for the SVM, with all parameters selected using 10-fold cross-validation on the training set.

In addition to BoS Trees, we also consider several alternative methods for BoS histograms: direct indexing of the large level-1 codebook (of sizes 624 and 630 on UCLA-39 and DynTex-35); and using a reduced codebook of size $K \in \{16,32,64\}$, obtained with the HEM-DTM algorithm as in [2]. For each experiment we registered average (per video) classification accuracy and average (per video) number of likelihood computations executed at test time to produce the BoS histograms, from which we computed the speed up with respect to the large level-1 codebook (X-Speedup). A small number of likelihood computations results in faster look-ups in the codebook, and in a larger speedup. Finally, results are averaged over 2 trials on UCLA-39 (the right sub-video used for training and the left for testing, and vice versa), 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.

4.3. Video classification results

Table 2 reports classification results on UCLA-39 and DynTex-35. 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: direct-indexed large codebook (large CB), a reduced codebook (reduced CB), and our proposed BoS Tree. Several observations can be made from the results on UCLA-39. First, increasing the codebook size (with direct indexing) 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, from requiring 7377 likelihood computations per video (about 5 seconds on a standard desktop PC) to 287,690 (about 182 seconds). Second, using BoS Trees leads to the best performance 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 to 36,393 (23 seconds), an almost 8 times reduction in computation. For other classifiers, the accuracy is on par, or decreases slightly, compared to the large CB. These results demonstrate that the BoS Trees efficiently and effectively index codewords, and hence allow practical use of a large and rich codebook. Third, although they use about the same number of likelihood operations, BoS Trees 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 rich enough to produce highly accurate classification. Lastly, using the BoS Tree with $L = 4$ and setting the transversing 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 the BoS Tree 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.

Similar conclusions can be drawn on DynTex-35, although the differences in classification accuracy are less substantial due to the easiness of the classification task. Again, we note that the BOS Tree achieves the same accuracy as the direct-indexed large CB, while reducing the computation by almost an order of magnitude.

Finally, the BoS Tree performance improves on the current state-of-the-art reported in the literature [19, 16, 21, 17] on the two textures datasets (last row of Table 2). On UCLA-39, the accuracy has improved from 20% [16] or 42.3% [19] to 82.37% for BoS Tree. In contrast to [16], which is based solely on motion dynamics, and [19], 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.

In addition, on the KTH dataset our BoS Tree ($L = 2$, $K_1 = 3040$ codewords and $K_2 = 32$) obtains a classification accuracy of 92.3% when using TF-IDF and CSK-SVM, which compares reasonably with results reported in the literature (which range from 87.3% [22] to 95.7% [23]). This shows the potential of BoS Trees as a universal motion descriptor for classification of non-texture, non-stationary videos, such as human actions. However, the comparison with state-of the art system suggests we could improve our BoS Trees with interest point operators, scale-selection, similar to those leveraged by [23] (95.7%) or [24] (93.2%). This is a topic of future work. We also note that the variety of codewords the BoS Tree can index is crucial to achieve good classification results, as a reduced codebook with $K = 32$ performed poorly with 72.8% accuracy.

5. Experiments: music annotation

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

5.1. Dataset

We perform automatic music annotation on the CAL500 dataset (details in [25] and references therein), which is a collection of 502 popular Western songs by as many different artists, 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 in [4] 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).

5.2. Experiment setup

The acoustic content of a songs is represented by computing a time-series of 34-bin Mel-frequency spectral features (see [25]), 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. 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 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 HIK kernel. Given a BoS histogram $\mathbf{h}$ 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 [26] for the LR classifier, with all parameters selected using 4-fold cross validation on the training set.

Table 3. 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	1
reduced CB	K = 128	0.403	0.668	0.402	0.342	0.209	0.227	12.55
	K = 64	0.381	0.649	0.378	0.315	0.192	0.204	25.10
	K = 32	0.368	0.634	0.368	0.298	0.180	0.191	50.20
BoS Tree	L = 2	0.445	0.712	0.451	0.398	0.24	0.261	8.24
	L = 3	0.443	0.712	0.448	0.393	0.235	0.258	11.54
	L = 4	0.439	0.711	0.448	0.394	0.232	0.255	14.31
SML-DTM [25]		0.446	0.708	0.460	0.446	0.217	0.264	1.03

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 [4]: 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 [4], and supervised multi-class labeling with DTM models (SML-DTM) [25].

5.3. Music annotation results

In Table 3 we report annotation and retrieval performance on the CAL500 dataset. Due to space constraints, we report results only for LR using TF-IDF representation and HIK kernel, but similar trends were noticed for TF representation and other kernels.

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 [4]) 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

Table 2. Video classification results on 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 two rows report the speedup relative to large CB to build the BoS histograms at test time, and reference results (Ref).

Method			UCLA-39								DynTex-35						
			large CB	reduced CB			BoS Tree $\kappa = 1$			$\mathfrak{T} = 0.995$	large CB	reduced CB			BoS Tree $\kappa = 1$		
			$\lvert\mathcal{C}\rvert = 624$	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	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	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	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	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	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	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	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	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	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	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			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	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			42.3 [19], 20 [16], 15 [21]								97.14 [17]						

direct-indexed large codebook and for SML-DTM. Second, as with video 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. Conclusions

In this paper we have proposed the BoS Tree, which efficiently indexes DT codewords of a BoS representation using a hierarchical structure. The BoS Tree enables the practical use of larger and richer collections of codewords in the BoS representation. We have proven the efficacy of the BoS Tree on three video datasets, and on a music dataset as well.

Acknowledgements

E.C., A.B.C. and G.R.G.L. acknowledge support from Google, Inc. E.C. and G.R.G.L. acknowledge support from Qualcomm, Inc, Yahoo!, Inc., the Hellman Fellowship Program, the Sloan Foundation, and NSF Grants CCF-0830535 and IIS-1054960. A.M. and A.B.C. were supported by the Research Grants Council of the Hong Kong Special Administrative Region, China [9041552 (CityU 110610)]. This research was supported in part by the UCSD FWGrid Project, NSF Research Infrastructure Grant Number EIA-0303622.

References

[1] A. Ravichandran, R. Chaudhry, and R. Vidal, "View-invariant dynamic texture recognition using a bag of dynamical systems," in *CVPR*, 2009. 1, 2, 5

[2] A. B. Chan, E. Coviello, and G. R. G. Lanckriet, "Clustering dynamic textures with the hierarchical em algorithm," in *CVPR*, 2010. 1, 2, 3, 6

[3] P. Saisan, G. Doretto, Y. Wu, and S. Soatto, "Dynamic texture recognition," in *CVPR*. IEEE, 2001. 1, 5

[4] 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. 1, 2, 7

[5] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto, "Dynamic textures," *Intl. J. Computer Vision*, 2003. 2

[6] A. B. Chan and N. Vasconcelos, "Modeling, clustering, and segmenting video with mixtures of dynamic textures," *IEEE TPAMI*, 2008. 2, 4

[7] 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. 2

[8] A. Gersho and R. Gray, *Vector quantization and signal compression*. Springer Netherlands, 1992, vol. 159. 5

[9] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *CVPR*, 2006. 5

[10] K. Grauman and T. Darrell, "Approximate correspondences in high dimensions," *NIPS*, 2007. 5

[11] J. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, pp. 509–517, 1975. 5

[12] J. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Information Processing Letters*, 1991. 5

[13] L. Cayton, "Fast nearest neighbor retrieval for bregman divergences," in *ICML*, 2008. 5

[14] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *CVPR*, 2007. 5

[15] N. Vasconcelos, "Image indexing with mixture hierarchies," in *CVPR*, 2001. 5

[16] F. Woolfe and A. Fitzgibbon, "Shift-invariant dynamic texture recognition," *Computer Vision–ECCV*, 2006. 5, 6, 8

[17] 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. 5, 6, 8

[18] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local svm approach," in *ICPR 2004*. IEEE, 2004. 5

[19] K. Derpanis and R. Wildes, "Dynamic texture recognition based on distributions of spacetime oriented structure," in *CVPR*, 2010. 5, 6, 8

[20] C. Chang and C. Lin, "Libsvm: a library for support vector machines," *ACM TIST*, 2011. 6

[21] A. B. Chan and N. Vasconcelos, "Probabilistic kernels for the classification of auto-regressive visual processes," in *CVPR*, 2005. 6, 8

[22] M. Yang, F. Lv, W. Xu, K. Yu, and Y. Gong, "Human action detection by boosting efficient motion features," in *ICCV*. IEEE, 2009. 7

[23] A. Gilbert, J. Illingworth, and R. Bowden, "Action recognition using mined hierarchical compound features," *IEEE TPAMI*, 2011. 7

[24] M. Bregonzio, S. Gong, and T. Xiang, "Recognising action as clouds of space-time interest points," in *CVPR*. IEEE, 2009. 7

[25] E. Coviello, A. Chan, and G. Lanckriet, "Time Series Models for Semantic Music Annotation," *IEEE TASLP*, 2011. 7

[26] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin, "Liblinear: A library for large linear classification," *JMLR*, 2008. 7