Clustering Dynamic Textures with the Hierarchical EM Algorithm

Antoni B. Chan
Dept. of Computer Science
City University of Hong Kong

Emanuele Coviello Gert. R. G. Lanckriet
Dept. of Electrical and Computer Engineering
University of California, San Diego

Abstract

The dynamic texture (DT) is a probabilistic generative model, defined over space and time, that represents a video as the output of a linear dynamical system (LDS). The DT model has been applied to a wide variety of computer vision problems, such as motion segmentation, motion classification, and video registration. In this paper, we derive a new algorithm for clustering DT models that is based on the hierarchical EM algorithm. The proposed clustering algorithm is capable of both clustering DTs and learning novel *DT cluster centers that are representative of the cluster members, in a manner that is consistent with the underlying generative probabilistic model of the DT. We then demonstrate the efficacy of the clustering algorithm on several applications in motion analysis, including hierarchical motion clustering, semantic motion annotation, and bag-of-systems codebook generation.*

1. Introduction

Modeling motion as a spatio-temporal texture has shown promise in a wide variety of computer vision problems, which have otherwise proven challenging for traditional motion representations, such as optical flow [1, 2]. In particular, the *dynamic texture model*, proposed in [3], has demonstrated a surprising ability to abstract a wide variety of complex global patterns of motion and appearance into a *simple* spatio-temporal model. The dynamic texture (DT) is a probabilistic generative model, defined over space and time, that represents a video as the output of a linear dynamical system (LDS). The model includes a hidden-state process, which encodes the motion of the video over time, and an observation variable that determines the appearance of each video frame, conditioned on the current hidden-state. Both the hidden-state vector and the observation vector are representative of the entire image, enabling a holistic characterization of the motion for the entire sequence. The DT model has been applied to a wide variety of computer vision problems, including video texture synthesis [3], video registration [4, 5], motion and video texture segmentation [6, 7, 8], human activity recognition [9], and motion classification [10, 11, 12, 13, 14]. These successes illustrate both the modeling capabilities of the DT representation, and the robustness of the underlying probabilistic framework.

In this paper, we address the problem of clustering dynamic texture models, *i.e.*, clustering linear dynamical systems. Given a set of DTs (*e.g.*, each learned from a small video cube extracted from a large set of videos), the goal is to group similar DTs into K clusters, while also learning a representative DT "center" that can sufficiently summarize each group. This is analogous to standard K-means clustering, except that the datapoints are dynamic textures, instead of real vectors. A robust DT clustering algorithm has several potential applications in video analysis, including: 1) hierarchical clustering of motion; 2) video indexing for fast video retrieval; 3) DT codebook generation for the bag-of-systems motion representation; 4) semantic video annotation via weakly-supervised learning. Finally, DT clustering can also serve as an effective method for learning DTs from a large dataset of video via hierarchical modeling.

The parameters of the LDS lie on a non-Euclidean space (non-linear manifold), and hence cannot be clustered directly with the K-means algorithm, which operates on real vectors in Euclidean space. One solution, proposed in [13], first embeds the DTs into a Euclidean space using non-linear dimensionality reduction (NLDR) with an appropriate distance between DTs (*e.g.*, the Martin distance), and then performs K-means on the low-dimensional space to obtain the clustering. While this performs the task of grouping the DTs into similar clusters, [13] is not able to generate *novel* DTs as cluster centers, due to the pre-image and out-of-sample limitations of kernelized NLDR techniques. In this case, the DT whose low-dimensional embedding is closest to the low-dimensional cluster center is selected as the representative DT center. These limitations could be addressed by clustering the DTs parameters directly on the non-linear manifold, *e.g.*, using intrinsic mean-shift [15] or LLE [16]. However, these methods require analytic expressions for the log and exponential map on the manifold, which are difficult to compute for the DT parameters.

An alternative to clustering with respect to the manifold structure is to directly cluster the probability distributions of the DTs. One method for clustering probability distributions, in particular, Gaussians, is the hierarchical expectation-maximization (HEM) algorithm, proposed in [17]. The HEM algorithm of [17] takes a Gaussian mix-

ture model (GMM) with K_b mixture components and reduces it to another GMM with K_r components ($K_r < K_b$), where each of the new Gaussian components represents a group of the original Gaussians (*i.e.*, forming a cluster of Gaussians). The HEM algorithm for GMMs has been employed in [18] to build GMM hierarchies for efficient image indexing, and in [19] to estimate GMMs from large image datasets for semantic annotation.

In this paper, we derive an HEM algorithm for *clustering dynamic textures* through their probability distributions. The resulting algorithm is capable of both clustering DTs and learning *novel* DT cluster centers that are representative of the cluster members, in a manner that is consistent with the underlying generative probabilistic model of the DT. We then demonstrate the efficacy of the clustering algorithm on several computer vision problems: 1) hierarchical motion clustering; 2) semantic motion annotation using weakly-supervised learning; and 3) codebook generation for the bag-of-systems motion representation. The remainder of the paper is organized as follows. In Section 2, we review the dynamic texture model, and in Section 3, we derive the HEM algorithm for dynamic textures. Finally, Section 4 concludes with a discussion on applications and experimental evaluations.

2. Dynamic texture models

A dynamic texture [3] (DT) is a generative model for both the appearance and the dynamics of video sequences. The model consists of a random process containing an *observation variable* y_t, which encodes the appearance component (vectorized video frame at time t), and a *hidden state variable* x_t, which encodes the dynamics (evolution of the video over time). The appearance component is drawn at each time instant, conditionally on the current hidden state. The state and observation variables are related through the *linear dynamical system* (LDS) defined by

$$x_t = Ax_{t-1} + v_t, \quad (1)$$

$$y_t = Cx_t + w_t + \bar{y}, \quad (2)$$

where $x_t \in \mathbb{R}^n$ and $y_t \in \mathbb{R}^m$ are real vectors (typically $n \ll m$). The matrix $A \in \mathbb{R}^{n\times n}$ is a *state transition matrix*, which encodes the dynamics or evolution of the hidden state variable (*i.e.*, the motion of the video), and the matrix $C \in \mathbb{R}^{m\times n}$ is an *observation matrix*, which encodes the appearance component of the video sequence. The vector $\bar{y} \in \mathbb{R}^n$ is the mean of the dynamic texture (i.e. the mean video frame). v_t is a *driving noise process*, and is zero-mean Gaussian distributed, *i.e.*, $v_t \sim \mathcal{N}(0, Q)$, where $Q \in \mathbb{R}^{n\times n}$ is a covariance matrix. w_t is the *observation noise* and is also zero-mean Gaussian, *i.e.*, $w_t \sim \mathcal{N}(0, R)$, where $R \in \mathbb{R}^{m\times m}$ is a covariance matrix (typically, i.i.d. noise is assumed, or $R = rI_m$). Finally, the *initial condition* is specified as $x_1 \sim \mathcal{N}(\mu, S)$, where $\mu \in \mathbb{R}^n$ is the mean of the initial state, and $S \in \mathbb{R}^{n\times n}$ is the covariance. The dynamic texture is specified by the parameters $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$, which can be learned using maximum-likelihood (e.g. expectation-maximization [20]), or a suboptimal, but computationally efficient, method [3].

While a dynamic texture models a time-series as a single sample from a linear dynamical system, the dynamic texture mixture (DTM), proposed in [8], models multiple time-series as samples from a set of K dynamic textures. The DTM model introduces an assignment random variable $z \sim \text{multinomial}(\pi_1, \cdots, \pi_K)$, which selects the parameters of one of the K dynamic texture components for generating a video observation. Each mixture component is parameterized by $\Theta_z = \{A_z, C_z, Q_z, R_z, \mu_z, S_z, \bar{y}_z\}$, and the DTM model is parameterized by $\Theta = \{\pi_z, \Theta_z\}_{z=1}^K$. Given a set of video samples, the maximum-likelihood parameters of the DTM can be estimated with recourse to the expectation-maximization (EM) algorithm [8].

3. The HEM algorithm for dynamic textures

The hierarchical expectation-maximization (HEM) algorithm was proposed in [17] to reduce a Gaussian mixture model (GMM) with a large number of components into a representative GMM with fewer components. In this section we derive the HEM algorithm when the mixture components are *dynamic textures*.

3.1. Formulation

Formally, let $\Theta^{(b)} = \{\pi_i^{(b)}, \Theta_i^{(b)}\}_{i=1}^{K^{(b)}}$ denote the base DT mixture model with $K^{(b)}$ components. The likelihood of the observed random variable $y_{1:\tau} \sim \Theta^{(b)}$ is given by

$$p(y_{1:\tau}|\Theta^{(b)}) = \sum_{i=1}^{K^{(b)}} \pi_i^{(b)} p(y_{1:\tau}|z^{(b)} = i, \Theta^{(b)}), \quad (3)$$

where $y_{1:\tau}$ is the video, τ is the video length, and $z \sim \text{multinomial}(\pi_1^{(b)}, \cdots \pi_{K^{(b)}}^{(b)})$ is the hidden variable that indexes the mixture components. $p(y_{1:\tau}|z^{(b)} = i, \Theta^{(b)})$ is the likelihood of the video $y_{1:\tau}$ under the ith DT mixture component, and $\pi_i^{(b)}$ is the prior weight for the ith component. The goal is to find a reduced DT mixture model, $\Theta^{(r)}$, which represents (3) using fewer mixture components. The likelihood of the observed video random variable $y_{1:\tau} \sim \Theta^{(r)}$ is

$$p(y_{1:\tau}|\Theta^{(r)}) = \sum_{j=1}^{K^{(r)}} \pi_j^{(r)} p(y_{1:\tau}|z^{(r)} = j, \Theta^{(r)}), \quad (4)$$

where $K^{(r)}$ is the number of DT components in the reduced model ($K^{(r)} < K^{(b)}$), and $z^{(r)} \sim \text{multinomial}(\pi_1^{(r)}, \cdots, \pi_{K^{(r)}}^{(r)})$ is the hidden variable for indexing components in $\Theta^{(r)}$. Note that we will always use

i and j to index the components of the base model $\Theta^{(b)}$ and the reduced model $\Theta^{(r)}$, respectively. We will also use the short-hand $\Theta_i^{(b)}$ and $\Theta_j^{(r)}$ to denote the ith component of $\Theta^{(b)}$ and the jth component of $\Theta^{(r)}$, respectively. For example, we denote $p(y_{1:\tau}|z^{(b)}=i,\Theta^{(b)}) = p(y_{1:\tau}|\Theta_i^{(b)})$.

3.2. Parameter estimation

To obtain the reduced model, HEM [17] considers a set of N virtual observations drawn from the base model $\Theta^{(b)}$, such that $N_i = N\pi_i^{(b)}$ samples are drawn from the ith component. The DT, however, has both observable Y and hidden state X variables (which are distinct from the hidden assignments of the overall mixture). To adapt HEM to models with hidden state variables, the most straightforward approach is to draw virtual samples from both X and Y according to their joint distribution. However, when computing the parameters of a new reduced DT model, there is no guarantee that the virtual hidden states from the base models live in the same basis (equivalent DT can be formed by scaling, rotating, or permuting A, C, and X). This basis mismatch will cause problems when estimating parameters from the virtual samples of the hidden states. The key insight is that, in order to remove ambiguity caused by multiple equivalent internal state representations, we must only generate virtual samples from the observable Y, while treating the hidden states X as missing information in HEM.

We denote the set of N_i virtual video samples for the ith component as $Y_i = \{y_{1:\tau}^{(i,m)}\}_{m=1}^{N_i}$, where $y_{1:\tau}^{(i,m)} \sim \Theta_i^{(b)}$ is a single video sample and τ is the length of the virtual video (a parameter we can choose). The entire set of N samples is denoted as $Y = \{Y_i\}_{i=1}^{K^{(b)}}$. To obtain a consistent hierarchical clustering, we also assume that all the samples in a set Y_i are eventually assigned to the same reduced component $\Theta_j^{(r)}$. The parameters of the reduced model can then be computed using maximum likelihood estimation with the virtual video samples,

$$\Theta^{(r)^*} = \underset{\Theta^{(r)}}{\operatorname{argmax}} \log p(Y|\Theta^{(r)}) \tag{5}$$

$$= \underset{\Theta^{(r)}}{\operatorname{argmax}} \log \prod_{i=1}^{K^{(b)}} \sum_{j=1}^{K^{(r)}} \pi_j^{(r)} \int p(Y_i, X_i|\Theta_j^{(r)}) dX_i \tag{6}$$

where $X_i = \{x_{1:\tau}^{(i,m)}\}$ are the hidden-state variables corresponding to Y_i. (6) requires marginalizing over hidden states, and hence it can be solved using the EM algorithm [21], which is an iterative optimization method that alternates between estimating the hidden variables with the current parameters, and computing new parameters given the estimated hidden variables (the "complete data"), given by

$$\text{E-Step: } \mathcal{Q}(\Theta^{(r)}, \hat{\Theta}^{(r)}) = \mathbb{E}_{X,Z|Y,\hat{\Theta}^{(r)}}[\log p(X,Y,Z|\Theta^{(r)})],$$

$$\text{M-Step: } \Theta^{(r)^*} = \underset{\Theta^{(r)}}{\operatorname{argmax}}\, \mathcal{Q}(\Theta^{(r)}, \hat{\Theta}^{(r)}),$$

where $\hat{\Theta}^{(r)}$ is the current estimate of the parameters, $p(X,Y,Z|\Theta^{(r)})$ is the "complete-data" likelihood, and $\mathbb{E}_{X,Z|Y,\hat{\Theta}^{(r)}}$ is the conditional expectation with respect to the current model parameters.

As is common with the EM formulation, we introduce a hidden assignment variable $\mathbf{z}_{i,j}$, which is an indicator variable for when the video sample set Y_i is assigned to the jth component of $\Theta^{(r)}$, *i.e.*, when $z_i^{(r)} = j$. The complete-data log-likelihood is then

$$\log p(X,Y,Z|\Theta^{(r)}) = \sum_{i=1}^{K^{(b)}} \sum_{j=1}^{K^{(r)}} \mathbf{z}_{i,j} \log \pi_j^{(r)} + \mathbf{z}_{i,j} \log p(Y_i, X_i|\Theta_j^{(r)}). \tag{7}$$

The $\mathcal{Q}$ function is obtained by taking the conditional expectation with respect to $\{X,Z\}$ (see [22] for derivation),

$$\mathcal{Q}(\Theta^{(r)}, \hat{\Theta}^{(r)}) = \sum_{i=1}^{K^{(b)}} \sum_{j=1}^{K^{(r)}} \hat{\mathbf{z}}_{i,j} \log \pi_j^{(r)} + \hat{\mathbf{z}}_{i,j} N_i \mathbb{E}_{y|\Theta_i^{(b)}} \left[\mathbb{E}_{x|y,\hat{\Theta}_j^{(r)}} [\log p(y_{1:\tau}, x_{1:\tau}|\Theta_j^{(r)})] \right], \tag{8}$$

where the probability of assigning the ith base component to the jth reduced component is

$$\hat{\mathbf{z}}_{i,j} = \frac{\pi_j^{(r)} \exp\left(N_i \mathbb{E}_{\Theta_i^{(b)}} [\log p(y_{1:\tau}|\hat{\Theta}_j^{(r)})] \right)}{\sum_{j'=1}^{K^{(r)}} \pi_{j'}^{(r)} \exp\left(N_i \mathbb{E}_{\Theta_i^{(b)}} [\log p(y_{1:\tau}|\hat{\Theta}_{j'}^{(r)})] \right)}. \tag{9}$$

Note that the $\mathcal{Q}$ function in (8) is very similar to that of the EM algorithm for DTM [8]. In HEM, each base DT $\Theta_i^{(b)}$ takes role of a "data-point" in standard EM, where an additional expectation w.r.t. $\Theta_i^{(b)}$ averages over the possible value of the "data-point", yielding the double expectation $\mathbb{E}_{y|\Theta_i^{(b)}}[\mathbb{E}_{x|y,\hat{\Theta}_j^{(r)}}[\cdot]]$. These expectations of the hidden state, conditioned on each component $\Theta_i^{(b)}$, are computed through a common DT model $\hat{\Theta}_j^{(r)}$. Hence, the potential problem with mismatches between the hidden state bases of $\Theta^{(b)}$ is avoided. We next derive the E- and M-steps.

3.3. E-step

Substituting the DT component likelihoods into (8) yields a $\mathcal{Q}$ function for HEM-DTM, which requires the following summary statistics (see [22] for derivation):

$$\begin{aligned}
\hat{N}_j &= \textstyle\sum_i \hat{\mathbf{z}}_{i,j}, & \Phi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{P}_{t,t|j}^{(i)},\\
\hat{M}_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j}, & \Psi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=2}^{\tau} \hat{P}_{t,t-1|j}^{(i)},\\
\xi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \hat{x}_{1|j}^{(i)}, & \varphi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=2}^{\tau} \hat{P}_{t,t|j}^{(i)},\\
\eta_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \hat{P}_{1,1|j}^{(i)}, & \phi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=2}^{\tau} \hat{P}_{t-1,t-1|j}^{(i)},\\
\gamma_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{u}_t^{(i)}, & \Lambda_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{U}_{t|j}^{(i)},\\
\beta_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{x}_{t|j}^{(i)}, & \Gamma_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{W}_{t|j}^{(i)},
\end{aligned}$$

with $\hat{\mathbf{w}}_{i,j} = \hat{\mathbf{z}}_{i,j} N_i = \hat{\mathbf{z}}_{i,j} \pi_i^{(b)} N$. These terms are the aggregates of the individual expectations,

$$\begin{aligned}
\hat{x}_{t|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(b)}} \left[\mathbb{E}_{x|y,\hat{\Theta}_j^{(r)}}[x_t] \right], \\
\hat{P}_{t,t|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(b)}} \left[\mathbb{E}_{x|y,\hat{\Theta}_j^{(r)}}[x_t x_t^T] \right], \\
\hat{P}_{t,t-1|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(b)}} \left[\mathbb{E}_{x|y,\hat{\Theta}_j^{(r)}}[x_t x_{t-1}^T] \right], \\
\hat{W}_{t|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(b)}} \left[(y_t - \bar{y}_j) \mathbb{E}_{x|y,\hat{\Theta}_j^{(r)}}[x_t]^T \right], \\
\hat{U}_{t|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(b)}} \left[(y_t - \bar{y}_j)(y_t - \bar{y}_j)^T \right], \\
\hat{u}_t^{(i)} &= \mathbb{E}_{y|\Theta_i^{(b)}} \left[y_t \right],
\end{aligned} \tag{10}$$

where $\hat{\Theta}_j^{(r)}$ is the current parameter estimate for the jth component of the reduced model. Here, the inner expectation $\mathbb{E}_{x|y,\hat{\Theta}_j^{(r)}}[x_t]$ is the conditional state estimator of the Kalman smoothing filter [20, 23], when given an observation y. Hence, $\hat{x}_{t|j}^{(i)}$ is the output of the state estimator from a Kalman smoothing filter for $\hat{\Theta}_j^{(r)}$, when the observation y is generated with a different model $\Theta_i^{(b)}$. This is also known as "suboptimal filter analysis" or "sensitivity analysis" [23], where the goal is to analyze filter performance when an optimal filter, according to some source distribution, is run on a different source distribution. The expectations in (10), along with expected log-likelihood $\mathbb{E}_{y|\Theta_i^{(b)}}[\log p(y_{1:\tau}|\hat{\Theta}_j^{(r)})]$ in (9), can be computed using sensitivity analysis on the Kalman forward and smoothing filters (see Appendix for details).

3.4. M-step

In the M-step, the parameters $\Theta^{(r)}$ are updated by maximizing the $\mathcal{Q}$ function, yielding the parameter updates for each DT component $\Theta_j^{(r)}$ (derivation in [22]),

$$\begin{aligned}
&C_j^* = \Gamma_j \Phi_j^{-1}, \quad && R_j^* = \tfrac{1}{\tau \hat{M}_j}(\Lambda_j - C_j^* \Gamma_j), \\
&A_j^* = \Psi_j \phi_j^{-1}, \quad && Q_j^* = \tfrac{1}{(\tau-1)\hat{M}_j}(\varphi_j - A_j^* \Psi_j^T), \\
&\mu_j^* = \tfrac{1}{\hat{M}_j} \xi_j, && S_j^* = \tfrac{1}{\hat{M}_j} \eta_j - \mu_j^* (\mu_j^*)^T, \\
&\pi_j^* = \tfrac{\hat{M}_j}{K^{(b)}}, && \bar{y}_j^* = \tfrac{1}{\tau \hat{M}_j}(\gamma_j - C_j^* \beta_j).
\end{aligned} \tag{11}$$

4. Applications and Experiments

In this section, we discuss several novel applications of HEM-DTM to video and motion analysis, including hierarchical motion clustering, semantic motion annotation, and DT codebook generation for the bag-of-systems video representation. These applications exploit several desirable properties of HEM to obtain promising results. First, given a set of input DTs, HEM estimates a novel set of DTs that represents the input in a manner that is consistent with the underlying generative probabilistic models, by maximizing the log-likelihood of "virtual" samples generated from the input DTs. As a result, the clusters formed by HEM are also consistent with the probabilistic framework. Second, HEM can estimate models on large datasets, by breaking the learning problem into smaller pieces. In particular, intermediate models are learned on small non-overlapping portions of a large dataset, and the final model is estimated by running HEM on the intermediate models. Because HEM is based on maximum-likelihood principles, the resulting model is equivalent to performing maximum-likelihood estimation on the full dataset. However, the computer memory requirements are significantly less, since we no longer have to store the entire dataset during parameter estimation. In addition, the intermediate models are estimated independently of each other, so the task can be easily parallelized. In the remainder of the section, we present three applications of HEM-DTM to video and motion analysis.

4.1. Hierarchical clustering of video textures

We first consider hierarchical motion clustering of video textures, by successively clustering DT with the HEM algorithm. Given a set of K_1 video textures, spatio-temporal cubes are extracted from the video and a DT is learned for each video texture. This forms the first level of the hierarchy (the video-level DT). The next level in the hierarchy is formed by clustering the DTs from the previous level into K_2 groups with the HEM algorithm ($K_2 < K_1$). The DT cluster centers are selected as the representative models at this level, and the process is continued with each level in the hierarchy learned from the preceding level. The result is a tree representation of the video dataset, with similar textures grouped together in the hierarchy. Note that this type of hierarchy could not be built in a straightforward manner using the EM algorithm on the original spatio-temporal cubes. While it is possible to learn several DTMs with successively smaller values of K, there is no guarantee that the resulting mixtures, or the cluster memberships of the video patches, will form a tree.

4.1.1 Experimental setup

We illustrate hierarchical motion clustering on the video texture dataset from [8]. This dataset is composed of 99 video sequences, each containing 2 distinct video textures (see Figure 2 for examples). There are 12 texture classes in total, ranging from water (sea, river, pond) to plants (grass and trees), to fire and steam. The first level of the hierarchy is obtained by learning a DT for each texture in each video (hence, $K_1 = 198$). Each DT is learned using [6] on 100 spatio-temporal cubes ($5 \times 5 \times 60$ pixels) sampled from the texture segment. The second level of the hierarchy is obtained by running HEM on the level-1 DT mixture to reduce the components to $K_2 = 12$. Finally, the third and fourth levels are obtained by running HEM on the previous level for $K_3 = 6$ and $K_4 = 3$ clusters, respectively.

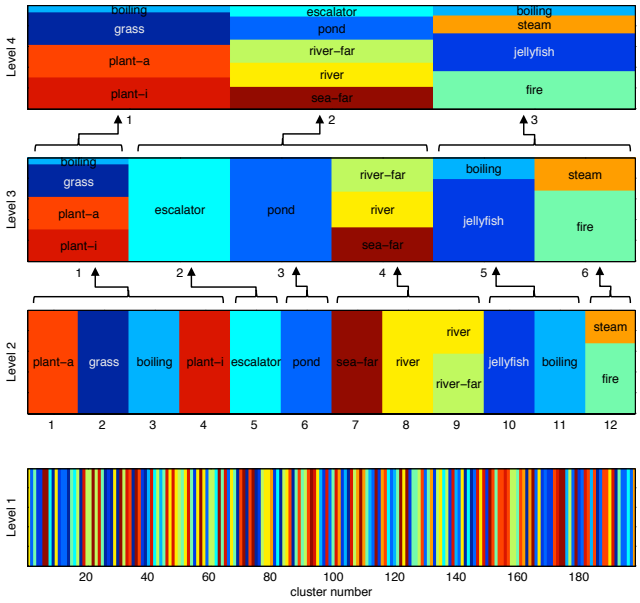

Figure 1. Hierarchical clustering of video textures: each level in the hierarchy is obtained by clustering the DT models from the preceding level. The arrows and brackets show the cluster membership from the preceding level (the groupings between Levels 1 and 2 are omitted for clarity).

4.1.2 Clustering Results

Figure 1 shows the hierarchical clustering that is obtained with HEM. The first level contains the DTs that represent each texture segment in the database. Each vertical bar represents one DT, where the color indicates the ground-truth cluster label (texture name). In the second level, the 12 DT components are shown as vertical bars, where the colors indicate the proportion of the cluster membership with a particular ground-truth cluster label. In most cases, each cluster corresponds to a single texture (*e.g.*, *grass*, *escalator*, *pond*), which illustrates that HEM is capable of clustering DTs into similar motions. The Rand index for the level-2 clustering using HEM is 0.973 (for comparison, clustering histograms-of-oriented-optical-flow using K-means yields a Rand index of 0.958). One error is seen in the HEM cluster with both the *river* and *river-far* textures, which is reasonable considering that the *river-far* texture contains both near and far perspectives of water. Moving up to the third level of the hierarchy, HEM forms two large clusters containing the plant textures (*plant-i*, *plant-a*, *grass*) and water textures (*river-far*, *river*, *sea-far*). Finally, in the fourth level, the video textures are grouped together according to broad categories: plants (*grass*, *plant-a*, *plant-i*), water (*pond*, *river-far*, *river*, *sea-far*), and rising textures (*fire*, *jellyfish*, and *steam*). These results illustrate that HEM for DT is capable of extracting meaningful clusters in a hierarchical manner.

4.2. Semantic video texture annotation

Another application of HEM is learning models for semantic image annotation. [19] treats image annotation as a semantic multi-class labeling problem (SML), where each keyword is modeled as a distribution of image features, $p(y|w_i)$ where w_i is the ith keyword and y the observed features. An image is annotated with the keywords with highest posterior probability, conditioned on the image, *i.e.*, $p(w_i|y) \propto p(y|w_i)p(w_i)$. Each annotation model $p(y|w_i)$ is learned independently by estimating a GMM on the positively-labeled images. One advantage of SML is that the models can be learned from weakly-labeled data (*i.e.*, image labels *without* segmentation data) by pooling over many images to amplify the relevant features. In this case, HEM is used to reduce the complexity of the learning problem. First, a GMM is estimated from each image using the standard EM algorithm. The annotation-level models are then estimated by applying HEM on the image-level GMMs to obtain a reduced mixture model. The burden of computation is in estimating the image-level GMMs, which can be easily executed in parallel.

We can perform *semantic motion annotation* with DTs by employing HEM in similar manner. First, each video is summarized by estimating a DTM with the EM algorithm from a dense sampling of spatio-temporal cubes. Next, a video annotation model is learned for each keyword by applying HEM to the video DTMs that are positively labeled with the keyword. Finally, a test video is annotated with the keywords with the largest posterior probability, given the spatio-temporal cubes of the test video.

4.2.1 Experimental setup

We learned DTM annotation models for the 12 video texture classes from the database of [8]. For each video in the training set, a DTM with $K = 4$ components ($n = 5$) is learned with the EM algorithm on $5 \times 5 \times 60$ spatio-temporal cubes. HEM is then applied to estimate the annotation models, which are DTMs with $K = 2$. Finally, a test video is annotated with the two keywords with largest posterior probability. The annotation models were trained on 50% of the dataset, with the remaining videos used for testing. We record the precision (P) and recall (R), along with the F-score (the harmonic mean of P and R), averaged over the keywords. The results are averaged over 10 trials.

4.2.2 Annotation Results

Table 1 presents the results for annotation using the DTM models. We compare two instantiations of the models: DTM that uses the image mean, and DTM that does not use the image mean. In the latter case, each video is normalized to have zero mean in time. The DTM without the image mean achieves a higher F-score (0.48) than the model using the mean (0.46). This discrepancy in performance is due to variations in lighting intensity in the various textures, which is negated by normalizing the video.

For comparison, we trained an image annotation model using GMMs on DCT features (GMM-DCT), as in [19],

a)

b) river-far, escalator | grass, fire | plant-a, river-far | grass, steam | sea-far, pond | river, jellyfish

c) *river-far, sea-far* | *grass, plant-a* | *plant-a, river* | *grass, plant-a* | *sea-far, river-far* | *jellyfish, river*

Figure 2. Video texture examples: a) video with 2 textures; b) ground-truth labels; c) *top 2 semantic annotations using DTM.*

Method	1 annotation			2 annotations		
	P	R	F	P	R	F
DTM (w/o mean)	**0.67**	**0.36**	**0.47**	**0.52**	**0.44**	**0.48**
DTM (w/ mean)	0.63	0.33	0.44	**0.52**	0.41	0.46
GMM-DCT (w/o mean)	0.51	0.28	0.37	0.44	0.38	0.41
GMM-DCT (w/ mean)	0.62	0.32	0.43	0.50	0.44	0.47
GMM-OF	0.60	0.34	0.44	0.47	0.41	0.44

Table 1. Semantic motion annotation results using DTM.

with $K = 2$ components. Although all the frames of the video are used to train the model, each frame is treated independently and thus the GMMs do not explicitly encode any motion. When using the image-mean, the GMM achieves a slightly lower F-score of 0.47 than the DTM (0.48). However, when the image-mean is ignored, the performance drops significantly to 0.41. This indicates that the GMM uses the intensity of the image to model the motion keywords. On the other hand, the DTM can model both the motion and appearance; when the appearance component (image-mean) is removed, the model can still distinguish the various motion classes. Looking at other motion features, learning GMMs on the optical flow of the video (GMM-OF) also yields a lower F-score of 0.44 than DTM.

Figure 2 shows the top two annotations for several example videos in the test set. In general, the system is able to annotate at least one of the video textures correctly. There are some errors, which are caused by similar motion classes (*e.g.*, *river-far* and *sea-far*).

4.3. Codebook generation for the bag-of-systems motion representation

The bag-of-systems (BoS) representation for video, where each motion codeword is a dynamic texture model (LDS), was recently proposed in [13]. To learn the DT codebook, [13] first estimates individual DTs, learned from spatio-temporal cubes extracted at spatio-temporal interest points in the video. Codewords are then generated by clustering the individual DTs using a combination of non-linear dimensionality reduction (NLDR) and K-means clustering. Due to the pre-image problem of kernelized NLDR, this clustering method is not capable of producing *novel* DT codewords, as discussed in Section 1.

HEM-DTM can be used to generate *novel* DT codewords for the bag-of-systems representation. First, for each video in the training corpus, a dense sampling of spatio-temporal cubes is extracted, and a DTM is learned with the EM algorithm [8]. Next, these DTMs are pooled together to form one large DTM, and the number of mixture components is reduced using the proposed HEM algorithm. Finally, the novel DT cluster centers are selected as the BoS codewords. Note that this method of codebook generation is able to exploit all the training data, as opposed to only a subset selected via interest-point operators as in [13]. This is made possible through the efficient hierarchical learning of the codebook model, as discussed in the previous sections.

4.3.1 Experimental setup

We use HEM to learn the BoS codebook on the database from [13]. The video-level DTMs were learned with $K = 4$ components and $n = 10$ on ~ 4000 overlapping spatio-temporal cubes with size $5 \times 5 \times 75$ pixels. A DT codebook was learned by reducing the mixture formed from all the video DTMs to $K = 8$ components with the HEM algorithm ($\tau = 20$ and $N = 1000$). Given the codebook, the BoS representation of a video is formed by counting the number of occurrences of each codeword in the video, where each spatio-temporal cube is assigned to the codeword with largest likelihood. A standard term frequency (TF) or term frequency-inverse document frequency (TF-IDF) representation can then be used to compute the histogram representation (weight vector w).

To illustrate the effectiveness of the BoS codebook learned with HEM, we perform the same classification experiments as [13], which consist of two binary problems (*water vs. fountain* and *fountain vs. waterfall*), a four-class problem, and an eight-class problem. In each experiment, 50% of the videos are used for training the model, with the remaining ones used for testing. We test the k-nearest neighbors classifier with the χ^2 and square-root distances between the TF weight vectors (denoted as TF-kNN-χ^2 and TF-kNN-S), for $k \in \{1, 3\}$. We also test the naive Bayesian classifier (NB) . The results are averaged over 20 trials.

4.3.2 Classification results

Table 2 presents the video classification results for the various classifiers using the HEM-DTM codebook, proposed here, and the NLDR codebook from [13]. "best" refers to the best accuracy among the various classifiers.

The first classification task considers the classes water and fountain, which are visually very different, and HEM-DTM and NLDR codebooks both achieve the same "best" classification rate of 100%. For the *fountain vs. waterfall* problem, which is a more challenging task because the classes are visually similar, classification with the

	HEM-DTM						NLDR [13]			
	TF-1NN-χ^2	TF-3NN-χ^2	TF-1NN-S	TF-3NN-S	NB	best	TF-1NN-χ^2	TF-3NN-χ^2	NB	best
Water vs. Fountain	97	94	99	99	100	100	97	97	100	100
Fountain vs. Waterfall	100	97	98	95	100	100	69	68	98	98
4 classes	94	89	95	92	95	95	87	89	68	89
8 classes	83	78	88	83	81	88	60	58	55	80

Table 2. Classification results using different methods for learning a bag-of-systems codebook.

HEM-DTM codebook achieves 100% accuracy. Looking at the individual classifiers, those based on the HEM-DTM codebook consistently outperform those based on NLDR. For example, the accuracy of TF-1NN-χ^2 is 100% for the HEM-DTM codebook, but drop to 69% for the NLDR codebook. Hence, the HEM-DTM codebook is a more stable representation in these first two scenarios.

The final two classification tasks consider 4-class and 8-class problems. The HEM-DTM codebook again outperforms that of NLDR (95% versus 89% on the 4-class problem, and 88% versus 80% on the 8-class problem), with each HEM-BoS classifier consistently improving over the NLDR version. Furthermore, when increasing the cardinality of the classification task to 4 and 8 classes, the accuracy of NB using the NLDR codebook drops dramatically, suggesting that the NLDR codebook violates the naive Bayes assumption that the codewords are independent of each other. In contrast, the performance using the HEM codebook does not drop as much, suggesting that the HEM codewords are independent of each other (*i.e.*, unique), and that they are spread out in the DTM space, enabling better representation of the data. Finally, we note that both the χ^2 distance and square root distance achieve high accuracy, despite the latter being discarded in [13] due to inconsistent results. These classification results indicate that HEM is an effective method for learning a codebook for the bag-of-systems representation. The improvement in performance is due to both the generation of novel DT codewords, and the ability to learn these codewords efficiently from more data, *i.e.*, from a dense sampling of spatio-temporal cubes, rather than those selected by interest point operators.

5. Conclusions

In this paper we derived a hierarchical EM algorithm that clusters DTs while learning novel DTs as representative cluster centers. Experiments using the new algorithm on several motion analysis problems, such as motion annotation and BoS codebook generation, show promising results. Future work will be directed at extending HEM to general graphical models, allowing a wide variety of generative models to be clustered or used as codewords in a bag-of-X representation.

Appendix: Computing the E-step for HEM-DTM

The expectations in (10) for each $\Theta_b = \Theta_i^{(b)}$ and $\Theta_r = \Theta_j^{(r)}$ can be computed efficiently with Algorithm 1, which we derive in [22]. First, the Kalman smoothing filter (Algorithm 2) computes the conditional expectations $\hat{x}_{t|\tau}^{(r)} = \mathbb{E}_{x|y,\Theta_r}[x_t]$, $\hat{V}_{t|\tau}^{(r)} = \mathrm{cov}_{x|y,\Theta_r}(x_t)$, and $\hat{V}_{t,t-1|\tau}^{(r)} = \mathrm{cov}_{x|y,\Theta_r}(x_t, x_{t-1})$, where $\hat{a}_{t|s}^{(r)}$ denotes the expectation at time t, conditioned on sequence $y_{1:s}$, w.r.t. Θ_r. Next, sensitivity analysis of the Kalman filter (Algorithm 3) computes the mean and variance of the one-step ahead state estimators when $y_{1:t-1} \sim \Theta_b$,

$$\hat{\mathbf{x}}_t = \mathbb{E}_{\Theta_b}\begin{bmatrix} x_t^{(b)} \\ \hat{x}_{t|t-1}^{(b)} \\ \hat{x}_{t|t-1}^{(r)} \end{bmatrix}, \hat{\mathbf{V}}_t = \mathrm{cov}_{\Theta_b}\left(\begin{bmatrix} x_t^{(b)} \\ \hat{x}_{t|t-1}^{(b)} \\ \hat{x}_{t|t-1}^{(r)} \end{bmatrix}\right). \quad (12)$$

The notation $\mathbf{V}^{[i,j]}$ refers to the (i,j) matrix in the block matrix $\mathbf{V}$, and $\mathbf{x}^{[i]}$ refers to the ith vector in the block vector $\mathbf{x}$. Finally, sensitivity analysis of the Kalman smoothing filter (Algorithm 4) computes the mean and variance of the state estimators for the full sequence $y_{1:\tau} \sim \Theta_b$,

$$\begin{aligned}
&\hat{x}_t^{(b)} = \mathbb{E}_{y|\Theta_b}\left[\hat{x}_{t|\tau}^{(r)}\right], \quad \hat{\kappa}_t^{(b)} = \mathrm{cov}_{y|\Theta_b}(y_t, \hat{x}_{t|\tau}^{(r)}) \\
&\hat{V}_t^{(b)} = \mathbb{E}_{y|\Theta_b}[\hat{V}_{t|\tau}^{(r)}], \quad \hat{V}_{t,t-1}^{(b)} = \mathbb{E}_{y|\Theta_b}[\hat{V}_{t,t-1|\tau}^{(r)}], \\
&\hat{\chi}_t^{(b)} = \mathrm{cov}_{y|\Theta_b}(\hat{x}_{t|\tau}^{(r)}), \quad \hat{\chi}_{t,t-1}^{(b)} = \mathrm{cov}_{y|\Theta_b}(\hat{x}_{t|\tau}^{(r)}, \hat{x}_{t-1|\tau}^{(r)}).
\end{aligned}$$

Algorithm 1 Expectations for HEM-DTM

1: **Input**: DT parameters Θ_b and Θ_r, length τ.
2: Run Kalman smoothing filter (Algorithm 2) on Θ_b and Θ_r.
3: Run sensitivity analysis on Θ_b and Θ_r for the Kalman filter and Kalman smoothing filter (Algorithms 3 and 4).
4: Compute E-step expectations, for $t = \{1, \cdots, \tau\}$:
$\hat{u}_t^{(b)} = C_b \hat{\mathbf{x}}_t^{[1]} + \bar{y}_b$,
$\hat{U}_t^{(b)} = C_b \hat{\mathbf{V}}_t^{[1,1]} C_b^T + R_b + (\hat{u}_t^{(b)} - \bar{y}_r)(\hat{u}_t^{(b)} - \bar{y}_r)^T$,
$\hat{P}_t^{(b)} = \hat{V}_{t|\tau}^{(r)} + \hat{\chi}_t^{(b)} + \hat{x}_t^{(b)}(\hat{x}_t^{(b)})^T$,
$\hat{P}_{t,t-1}^{(b)} = \hat{V}_{t,t-1|\tau}^{(r)} + \hat{\chi}_{t,t-1}^{(b)} + \hat{x}_t^{(b)}(\hat{x}_{t-1}^{(b)})^T$,
$\hat{W}_t^{(b)} = \hat{\kappa}_t^{(b)} + (\hat{u}_t^{(b)} - \bar{y}_r)(\hat{x}_t^{(b)})^T$.
5: Compute expected log-likelihood ℓ:
$\hat{\Lambda}_t = C_r(\hat{\mathbf{V}}_t^{[3,3]} + \hat{\mathbf{x}}_t^{[3]}(\hat{\mathbf{x}}_t^{[3]})^T)C_r^T$, $\hat{\Sigma}_t = C_r \hat{V}_{t|t-1}^{(r)} C_r^T + R_r$,
$\hat{\lambda}_t = C_b \hat{\mathbf{V}}_t^{[2,3]} C_r^T + (C_b \hat{\mathbf{x}}_t^{[1]} + \bar{y}_b - \bar{y}_r)(\hat{\mathbf{x}}_t^{[3]})^T C_r^T$,
$\ell = \sum_{t=1}^{\tau} -\frac{1}{2}\mathrm{tr}\left(\hat{\Sigma}_t^{-1}(\hat{U}_t^{(b)} - \hat{\lambda}_t - \hat{\lambda}_t^T + \hat{\Lambda}_t)\right)$
$-\frac{1}{2}\log\left|\hat{\Sigma}_t\right| - \frac{m}{2}\log(2\pi)$.
6: **Output**: $\{\hat{x}_t^{(b)}, \hat{P}_t^{(b)}, \hat{P}_{t,t-1}^{(b)}, \hat{W}_t^{(b)}, \hat{U}_t^{(b)}, \hat{u}_t^{(b)}\}, \ell$.

Acknowledgment

E.C. and G.R.G.L. wish to acknowledge support from NSF grants DMS-MSPA 0625409 and CCF-0830535.

Algorithm 2 Kalman smoothing filter

Input: DT parameters $\Theta = \{A, C, Q, R, \mu, S, \bar{y}\}$, length τ.
Initialize: $\hat{x}_{1|0} = \mu_b$, $\hat{V}_{1|0} = S_b$.
for $t = \{1, \cdots, \tau\}$ **do**
  {Kalman filter – forward recursion}

$$\begin{aligned}
&\hat{V}_{t|t-1} = A\hat{V}_{t-1|t-1}A^T + Q,\\
&K_t = \hat{V}_{t|t-1}C^T(C\hat{V}_{t|t-1}C^T + R)^{-1},\\
&\hat{V}_{t|t} = (I - K_tC)\hat{V}_{t|t-1},\\
&G_t = AK_t, \quad F_t = A - AK_tC.
\end{aligned}$$

end for
Initialize: $\hat{V}_{\tau,\tau-1|\tau} = (I - K_\tau C)A\hat{V}_{\tau-1|\tau-1}$.
for $t = \{\tau, \cdots, 2\}$ **do**
  {Kalman smoothing filter – backward recursion}

$$\begin{aligned}
&J_{t-1} = \hat{V}_{t-1|t-1}A^T(\hat{V}_{t|t-1})^{-1}, \quad H_{t-1} = A^{-1} - J_{t-1},\\
&\hat{V}_{t-1|\tau} = \hat{V}_{t-1|t-1} + J_{t-1}(\hat{V}_{t|\tau} - \hat{V}_{t|t-1})J_{t-1}^T,\\
&\hat{V}_{t-1,t-2|\tau} = \hat{V}_{t-1|t-1}J_{t-2}^T\\
&\quad + J_{t-1}(\hat{V}_{t,t-1|\tau} - A\hat{V}_{t-1|t-1})J_{t-2}^T.
\end{aligned}$$

end for
Output: $\{\hat{V}_{t|t-1}, \hat{V}_{t|t}, \hat{V}_{t|\tau}, \hat{V}_{t,t-1|\tau}, G_t, F_t, H_t\}$.

Algorithm 3 Sensitivity analysis of the Kalman filter

Input: DTs Θ_b and Θ_r, associated Kalman filters, length τ.
Initialize: $\hat{\mathbf{x}}_1 = \begin{bmatrix} \mu_b \\ \mu_b \\ \mu_r \end{bmatrix}$, $\hat{\mathbf{V}}_1 = \begin{bmatrix} S_b & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.
for $t = \{2, \cdots, \tau + 1\}$ **do**
  Form block matrices:

$$\mathbf{A}_{t-1} = \begin{bmatrix} A_b & 0 & 0 \\ G_{t-1}^{(b)}C_b & F_{t-1}^{(b)} & 0 \\ G_{t-1}^{(r)}C_b & 0 & F_{t-1}^{(r)} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} I \\ 0 \\ 0 \end{bmatrix},$$

$$\mathbf{C}_{t-1} = \begin{bmatrix} 0 \\ G_{t-1}^{(b)} \\ G_{t-1}^{(r)} \end{bmatrix}, \mathbf{D}_{t-1} = \begin{bmatrix} 0 \\ 0 \\ G_{t-1}^{(r)} \end{bmatrix}.$$

  Update means and covariances:

$$\begin{aligned}
\hat{\mathbf{x}}_t &= \mathbf{A}_{t-1}\hat{\mathbf{x}}_{t-1} + \mathbf{D}_{t-1}(\bar{y}_b - \bar{y}_r),\\
\hat{\mathbf{V}}_t &= \mathbf{A}_{t-1}\hat{\mathbf{V}}_{t-1}\mathbf{A}_{t-1}^T + \mathbf{B}Q_b\mathbf{B}^T + \mathbf{C}_{t-1}R_b\mathbf{C}_{t-1}^T.
\end{aligned}$$

end for
Output: $\{\hat{\mathbf{x}}_t, \hat{\mathbf{V}}_t\}$.

Algorithm 4 Sensitivity analysis of the Kalman smoothing filter

Input: DTs Θ_b and Θ_r, associated Kalman smoothing filter, and Kalman filter sensitivity analysis, length τ.
Initialize: $\hat{x}_\tau^{(b)} = A_r^{-1}\hat{\mathbf{x}}_{\tau+1}^{[3]}$, $\hat{\chi}_\tau^{(b)} = A_r^{-1}\hat{\mathbf{V}}_{\tau+1}^{[3,3]}A_r^{-T}$, $L_\tau = A_r^{-1}$, $M_\tau = \mathbf{0}$.
for $t = \{\tau, \cdots, 1\}$ **do**
  Compute cross-covariance:

$$\rho_t = \left(L_tF_t^{(r)}\hat{\mathbf{V}}_t^{[3,2]} + (L_tG_t^{(r)}C_b + M_t)\hat{\mathbf{V}}_t^{[1,1]}\right)C_b^T + L_tG_t^{(r)}R_b.$$

  if $t > 1$ **then**
    Compute sensitivity:

$$\begin{aligned}
&\omega_t = L_tF_t^{(r)}\hat{\mathbf{V}}_t^{[3,3]} + (L_tG_t^{(r)}C_b + M_t)\hat{\mathbf{V}}_t^{[2,3]},\\
&\hat{x}_{t-1}^{(b)} = H_{t-1}^{(r)}\hat{\mathbf{x}}_t^{[3]} + J_{t-1}^{(r)}\hat{x}_t^{(b)},\\
&\hat{\chi}_{t-1}^{(b)} = \begin{bmatrix} H_{t-1}^{(r)} & J_{t-1}^{(r)} \end{bmatrix}\begin{bmatrix} \hat{\mathbf{V}}_t^{[3,3]} & \omega_t^T \\ \omega_t & \hat{\chi}_t^{(b)} \end{bmatrix}\begin{bmatrix} (H_{t-1}^{(r)})^T \\ (J_{t-1}^{(r)})^T \end{bmatrix},\\
&\hat{\chi}_{t,t-1}^{(b)} = \omega_t(H_{t-1}^{(r)})^T + \hat{\chi}_t^{(b)}(J_{t-1}^{(r)})^T.
\end{aligned}$$

    Update matrices:

$$\begin{aligned}
L_{t-1} &= H_{t-1}^{(r)} + J_{t-1}^{(r)}L_tF_t^{(r)},\\
M_{t-1} &= J_{t-1}^{(r)}(L_tG_t^{(r)}C_b + M_t)A_b.
\end{aligned}$$

  end if
end for
Output: $\{\hat{x}_t^{(b)}, \hat{\chi}_t^{(b)}, \hat{\chi}_{t,t-1}^{(b)}, \hat{\kappa}_t^{(b)} = \rho_t^T\}$.

References

[1] B. Horn and B. Schunk, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–204, 1981.

[2] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. DARPA Image Understanding Workshop*, 1981, pp. 121–130.

[3] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto, "Dynamic textures," *Intl. J. Computer Vision*, vol. 51, no. 2, pp. 91–109, 2003.

[4] A. W. Fitzgibbon, "Stochastic rigidity: image registration for nowhere-static scenes," in *ICCV*, vol. 1, 2001, pp. 662–70.

[5] A. Ravichandran and R. Vidal, "Dynamic texture registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[6] G. Doretto, D. Cremers, P. Favaro, and S. Soatto, "Dynamic texture segmentation," in *ICCV*, vol. 2, 2003, pp. 1236–42.

[7] A. Ghoreyshi and R. Vidal, "Segmenting dynamic textures with Ising descriptors, ARX models and level sets," in *Dynamical Vision Workshop in the European Conf. on Computer Vision*, 2006.

[8] A. B. Chan and N. Vasconcelos, "Modeling, clustering, and segmenting video with mixtures of dynamic textures," *IEEE TPAMI*, vol. 30, no. 5, pp. 909–926, May 2008.

[9] R. Chaudry, A. Ravichandran, G. Hager, and R. Vidal, "Histograms of oriented optical flow and Binet-Cauchy kernels on nonlinear dynamical systems for the recognition of human actions," in *CVPR*, 2009.

[10] P. Saisan, G. Doretto, Y. Wu, and S. Soatto, "Dynamic texture recognition," in *CVPR*, vol. 2, 2001, pp. 58–63.

[11] A. B. Chan and N. Vasconcelos, "Probabilistic kernels for the classification of auto-regressive visual processes," in *CVPR*, vol. 1, 2005, pp. 846–851.

[12] ——, "Classifying video with kernel dynamic textures," in *IEEE Conf. Computer Vision and Pattern Recognition*, 2007.

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

[14] B. Ghanem and N. Ahuja, "Phase based modelling of dynamic textures," in *IEEE Intl. Conf. on Computer Vision*, 2007.

[15] H. Cetingul and R. Vidal, "Intrinsic mean shift for clustering on Stiefel and Grassmann manifolds," in *CVPR*, 2009.

[16] A. Goh and R. Vidal, "Clustering and dimensionality reduction on Riemannian manifolds," in *CVPR*, 2008.

[17] N. Vasconcelos and A. Lippman, "Learning mixture hierarchies," in *Neural Information Processing Systems*, 1998.

[18] N. Vasconcelos, "Image indexing with mixture hierarchies," in *IEEE Conf. Computer Vision and Pattern Recognition*, 2001.

[19] 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, no. 3, pp. 394–410, March 2007.

[20] R. H. Shumway and D. S. Stoffer, "An approach to time series smoothing and forecasting using the EM algorithm," *Journal of Time Series Analysis*, vol. 3, no. 4, pp. 253–264, 1982.

[21] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society B*, vol. 39, pp. 1–38, 1977.

[22] A. B. Chan, E. Coviello, and G. Lanckriet, "Derivation of the hierarchical EM algorithm for dynamic textures," City University of Hong Kong, Tech. Rep., 2010.

[23] A. Gelb, *Applied Optimal Estimation*. MIT Press, 1974.