Classifying Video with Kernel Dynamic Textures

Antoni B. Chan and Nuno Vasconcelos
Department of Electrical and Computer Engineering
University of California, San Diego
abchan@ucsd.edu, nuno@ece.ucsd.edu

Abstract

The dynamic texture is a stochastic video model that treats the video as a sample from a linear dynamical system. The simple model has been shown to be surprisingly useful in domains such as video synthesis, video segmentation, and video classification. However, one major disadvantage of the dynamic texture is that it can only model video where the motion is smooth, i.e. video textures where the pixel values change smoothly. In this work, we propose an extension of the dynamic texture to address this issue. Instead of learning a linear observation function with PCA, we learn a non-linear observation function using kernel-PCA. The resulting kernel dynamic texture *is capable of modeling a wider range of video motion, such as chaotic motion (e.g. turbulent water) or camera motion (e.g. panning). We derive the necessary steps to compute the Martin distance between kernel dynamic textures, and then validate the new model through classification experiments on video containing camera motion.*

1. Introduction

The dynamic texture [1] is a generative stochastic model of video that treats the video as a sample from a linear dynamical system. Although simple, the model has been shown to be surprisingly useful in domains such as video synthesis [1, 2], video classification [3, 4, 5], and video segmentation [6, 7, 8, 2]. Despite these numerous successes, one major disadvantage of the dynamic texture is that it can only model video where the motion is smooth, i.e. video textures where the pixel values change smoothly. This limitation stems from the linear assumptions of the model: specifically, 1) the linear state-transition function, which models the evolution of the hidden state-space variables over time; and 2) the linear observation function, which maps the state-space variables into observations. As a result, the dynamic texture cannot model more complex motion, such as chaotic motion (e.g. turbulent water) or camera motion (e.g. panning, zooming, and rotations).

To some extent, the smoothness limitation of the dynamic texture has been addressed in the literature by modifying the linear assumptions of the dynamic texture model. For example, [9] keeps the linear observation function, while modeling the state-transitions with a closed-loop dynamic system. In contrast, [10, 11] utilize a non-linear observation function, modeled as a mixture of linear subspaces, while keeping the standard linear state-transitions. Similarly in [12], different views of a video texture are represented by a non-linear observation function that models the video texture manifold from different camera viewpoints. Finally, [7] treats the observation function as a piece-wise linear function that changes over time, but is not a generative model.

In this paper, we improve the modeling capability of the dynamic texture by using a non-linear observation function, while maintaining the linear state transitions. In particular, instead of using PCA to learn a linear observation function, as with the standard dynamic texture, we use *kernel* PCA to learn a non-linear observation function. The resulting *kernel dynamic texture* is capable of modeling a wider range of video motion. The contributions of this paper are three-fold. First, we introduce the kernel dynamic texture and describe a simple algorithm for learning the parameters. Second, we show how to compute the Martin distance between kernel dynamic textures, and hence introduce a similarity measure for the new model. Third, we build a video classifier based on the kernel dynamic texture and the Martin distance, and evaluate the efficacy of the model through a classification experiment on video containing camera motion. We begin the paper with a brief review of kernel PCA, followed by each of the three contributions listed above.

2. Kernel PCA

Kernel PCA [13] is the kernelized version of standard PCA [14]. With standard PCA, the data is projected onto the linear subspace (linear principal components) that best captures the variability of the data. In contrast, kernel PCA (KPCA) projects the data onto non-linear functions in the input-space. These non-linear principal components are de-

fined by the kernel function, but are never explicitly computed. An alternative interpretation is that kernel PCA first applies a non-linear feature transformation to the data, and then performs standard PCA in the feature-space.

Given a training data set of N points $Y = [y_1, \ldots, y_N]$ with $y_i \in \mathbb{R}^m$ and a kernel function $k(y_1, y_2)$ with associated feature transformation $\phi(y)$, i.e. $k(y_1, y_2) = \langle \phi(y_1), \phi(y_2) \rangle$, the c-th kernel principal component in the feature-space has the form [13] (assuming the data has zero-mean in the feature-space):

$$v_c = \sum_{i=1}^{N} \alpha_{i,c} \phi(y_i) \tag{1}$$

The KPCA weight vector $\alpha_c = [\alpha_{1,c}, \ldots, \alpha_{N,c}]^T$ is given by $\alpha_c = \frac{1}{\sqrt{\lambda_c}} v_c$, where λ_c and v_c are the c-th largest eigenvalue and eigenvector of the kernel matrix K, which has entries $[K]_{i,j} = k(y_i, y_j)$. Finally, the KPCA coefficients $X = [x_1, \cdots, x_N]$ of the training set Y are given by $X = \alpha^T K$, where $\alpha = [\alpha_1, \cdots, \alpha_n]$ is the KPCA weight matrix, and n is the number of principal components.

Several methods can be used to reconstruct the input vector from the KPCA coefficients, e.g. minimum-norm reconstruction [15, 16], or constrained-distance reconstruction [17]. Finally, in the general case, the KPCA equations can be extended to center the data in the feature-space if it is not already zero-mean (see [18] for details).

3. Kernel Dynamic Textures

In this section, we introduce the kernel dynamic texture. We begin by briefly reviewing the standard dynamic texture, followed by its extension to the kernel dynamic texture.

3.1. Dynamic texture

A dynamic texture [1] is a generative model for video, which treats the video as a sample from a linear dynamical system. The model, shown in Figure 1, separates the visual component and the underlying dynamics into two stochastic processes. The dynamics of the video are represented as a time-evolving state process $x_t \in \mathbb{R}^n$, and the appearance of the frame $y_t \in \mathbb{R}^m$ is a linear function of the current state vector with some observation noise. Formally, the system equations are

$$\begin{cases} x_t = Ax_{t-1} + v_t \\ y_t = Cx_t + w_t \end{cases} \tag{2}$$

where $A \in \mathbb{R}^{n \times n}$ is the state-transition matrix, $C \in \mathbb{R}^{m \times n}$ is the observation matrix, and $x_0 \in \mathbb{R}^n$ is the initial condition. The state and observation noise are given by $v_t \sim_{iid} \mathcal{N}(0, Q,)$ and $w_t \sim_{iid} \mathcal{N}(0, rI_m)$, respectively.

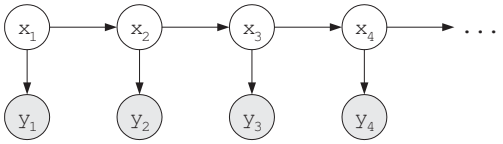


Figure 1. Graphical model of the dynamic texture.

Algorithm 1 Learning a kernel dynamic texture

Input: Video sequence $[y_1, \ldots, y_N]$, state space dimension n, kernel function $k(y_1, y_2)$.
Compute the mean: $\bar{y} = \frac{1}{N} \sum_{i=t}^{N} y_t$.
Subtract the mean: $y_t \leftarrow y_t - \bar{y}, \ \forall t$.
Compute the (centered) kernel matrix $[K]_{i,j} = k(y_i, y_j)$
Compute KPCA weights α from K.
$[\hat{x}_1 \cdots \hat{x_N}] = \alpha^T K$
$\hat{A} = [\hat{x}_2 \cdots \hat{x}_N][\hat{x}_1 \cdots \hat{x}_{N-1}]^\dagger$
$\hat{v}_t = \hat{x}_t - \hat{A}\hat{x}_{t-1}, \ \forall t$
$\hat{Q} = \frac{1}{N-1} \sum_{t=1}^{N-1} \hat{v}_t \hat{v}_t^T$
$\hat{y}_t = C(\hat{x}_t), \ \forall t$, (e.g. minimum-norm reconstruction).
$\hat{r} = \frac{1}{mN} \sum_{t=1}^{N} \|y_t - \hat{y}_t\|^2$
Output: $\alpha, \hat{A}, \hat{Q}, \hat{r}, \bar{y}$

When the parameters of the model are learned using the method of [1], the columns of C are the principal components of the video frames (in time), and the state vector is a set of PCA coefficients for the video frame, which evolve according to a Gauss-Markov process.

3.2. Kernel Dynamic Textures

Consider the extension of the standard dynamic texture where the observation matrix C is replaced by a non-linear function $C(x_t)$ of the current state x_t,

$$\begin{cases} x_t = Ax_{t-1} + v_t \\ y_t = C(x_t) + w_t \end{cases} \tag{3}$$

In general, learning the non-linear observation function can be difficult since the state variables are unknown. As an alternative, the inverse of the observation function, i.e. the function $D(y) : \mathbb{R}^m \rightarrow \mathbb{R}^n$ that maps observations to the state-space, can be learned with kernel PCA. The estimates of the state variables are then the KPCA coefficients, and the state-space parameters can be estimated with the least-squares method of [1]. The learning algorithm is summarized in Algorithm 1. We call a non-linear dynamic system, learned in this manner, a *kernel dynamic texture* because it uses kernel PCA to learn the state-space variables, rather than PCA as with the standard dynamic texture. Indeed when the kernel function is the linear kernel, the learning algorithm reduces to that of [1].

The kernel dynamic texture has two interpretations: 1) kernel PCA learns the non-linear observation function

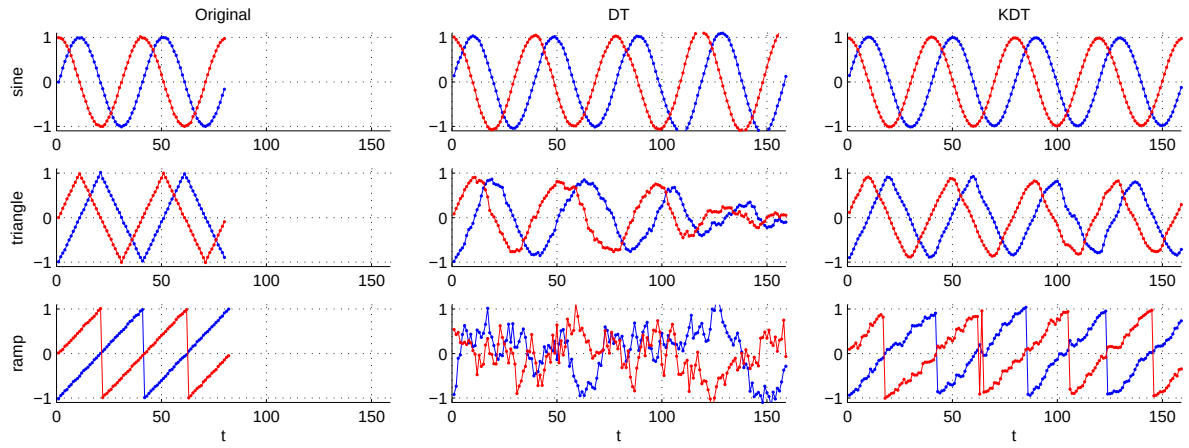


Figure 2. Synthesis examples: (left) The original time-series (sine wave, triangle wave, or periodic ramp); and a random sample generated from: (middle) the dynamic texture; and (right) the kernel dynamic texture, learned from the signal. The two dimensions of the signal are shown in different colors.

$C(x)$, which contains non-linear principal components; or 2) kernel PCA first transforms the data with the feature-transformation $\phi(y)$ induced by the kernel function, and then a standard dynamic texture is learned in the feature-space. This feature-space interpretation will prove useful in Section 4, where we compute the Martin distance between kernel dynamic textures.

3.3. Synthetic examples

In this section we show the expressive power of the kernel dynamic texture on some simple synthetic time-series. Figure 2 (left) shows three two-dimensional time-series: 1) a sine wave, 2) a triangle wave, and 3) a periodic ramp wave. Each time-series has length 80, and contains two periods of the waveform. The two-dimensional time-series was projected linearly into a 24-dimensional space, and Gaussian i.i.d. noise ($\sigma = 0.01$) was added. Note that the triangle wave and periodic ramp are not smooth signals in the sense that the former has a discontinuity in the first derivative, while the latter has a jump-discontinuity in the signal.

A dynamic texture and a kernel dynamic texture[1] were learned from the 24-dimensional time-series, with state-space dimension $n = 8$. Next, a random sample of length 160 was generated from the two models, and the 24-dimensional signal was linearly projected back into two dimensions for visualization. Figure 2 shows the synthesis results for each time-series. The kernel dynamic texture is able to model all three time-series well, including the more difficult triangle and ramp waves. This is in contrast to the dynamic texture, which can only represent the sine wave. For the triangle wave, the dynamic texture fails to capture the sharp peaks of the triangles, and the signal begins to degrade after $t = 80$. The dynamic texture does not capture any discernible signal from the ramp wave.

The results from these simple experiments indicate that the kernel dynamic texture is better at modeling arbitrary time-series. In particular, the kernel dynamic texture can model both discontinuities in the first derivative of the signal (e.g. the triangle wave), and jump discontinuities in the signal (e.g. the periodic ramp). These types of discontinuities occur frequently in video textures containing chaotic motion (e.g. turbulent water), or in texture undergoing camera motion (e.g. panning across a sharp edge). While the application of the kernel dynamic texture to video synthesis is certainly interesting and a direction of future work, in the remainder of this paper we will focus only on utilizing the model for classification of video textures.

3.4. Other related works

The kernel dynamic texture is related to non-linear dynamical systems, where both the state transitions and the observation functions are non-linear functions. In [19], the EM algorithm and the extended Kalman filter are used to learn the parameters of a non-linear dynamical system. In [20], the nonlinear mappings are modeled as multi-layer perceptron networks, and the system is learned using a Bayesian ensemble method. These methods can be computationally intensive because of the many degrees of freedom associated with both non-linear state-transitions and non-linear observation functions. In contrast, the kernel dynamic texture is a model where the state-transition is linear and the observation function is non-linear.

The kernel dynamic texture also has connections to dimensionality reduction; several manifold-embedding algorithms (e.g. ISOMAP, LLE) can be cast as kernel PCA with kernels specific to each algorithm [21]. Finally, the kernel dynamic texture is similar to [22, 23], which learns appear-

[1] The centered RBF kernel with width computed from (11) was used.

ance manifolds of video that are constrained by a Gauss-Markov state-space with *known* parameters A and Q.

4. Martin distance for kernel dynamic textures

Previous work [3] in video classification used the Martin distance as the similarity distance between dynamic texture models. The Martin distance [24] is based on the principal angles between the subspaces of the extended observability matrices of the two textures [25]. Formally, let $\Theta_a = \{C_a, A_a\}$ and $\Theta_b = \{C_b, A_b\}$ be the parameters of two dynamic textures. The Martin distance is defined as

$$d^2(\Theta_a, \Theta_b) = -\log \prod_{i=1}^{n} \cos^2 \theta_i \tag{4}$$

where θ_i is the i-th principal angle between the extended observability matrices $\mathcal{O}_a$ and $\mathcal{O}_b$, defined as $\mathcal{O}_a = \begin{bmatrix} C_a^T & A_a^T C_a^T & \cdots & (A_a^T)^n C_a^T & \cdots \end{bmatrix}^T$, and similarly for $\mathcal{O}_b$. It is shown in [25] that the principal angles can be computed by solving the following generalized eigenvalue problem:

$$\begin{bmatrix} 0 & \mathcal{O}_{ab} \\ (\mathcal{O}_{ab})^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} \mathcal{O}_{aa} & 0 \\ 0 & \mathcal{O}_{bb} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5}$$

subject to $x^T \mathcal{O}_{aa} x = 1$ and $y^T \mathcal{O}_{bb} y = 1$, where

$$\mathcal{O}_{ab} = (\mathcal{O}_a)^T \mathcal{O}_b = \sum_{t=0}^{\infty} (A_a^t)^T C_a^T C_b A_b^t \tag{6}$$

and similarly for $\mathcal{O}_{aa}$ and $\mathcal{O}_{bb}$. The first n largest eigenvalues are the cosines of the principal angles, and hence

$$d^2(\Theta_a, \Theta_b) = -2 \sum_{i=1}^{n} \log \lambda_i \tag{7}$$

The Martin distance for kernel dynamic textures can be computed by using the interpretation that the kernel dynamic texture is a standard dynamic texture learned in the feature-space of the kernel. Hence, in the matrix $F = C_a^T C_b$, the inner-products between the principal components can be replaced with the inner-products between the kernel principal components in the feature-space. However, this can only be done when the two kernels induce the same inner-product in the same feature-space.

Consider two data sets $\{y_i^a\}_{i=1}^{N_a}$ and $\{y_i^b\}_{i=1}^{N_b}$, and two kernel functions k_a and k_b with feature transformations $\phi(y)$ and $\psi(y)$, i.e. $k_a(y_1, y_2) = \langle \phi(y_1), \phi(y_2) \rangle$ and $k_b(y_1, y_2) = \langle \psi(y_1), \psi(y_2) \rangle$, which share the same inner-product and feature-spaces. Running KPCA on each of the data-sets with their kernels yields the KPCA weight matrices α and β, respectively. The c-th and d-th KPCA components in each of the feature-spaces are given by,

$$u_c = \sum_{i=1}^{N_a} \alpha_{i,c} \phi(y_i^a), \qquad v_d = \sum_{i=1}^{N_b} \beta_{i,d} \psi(y_i^b). \tag{8}$$

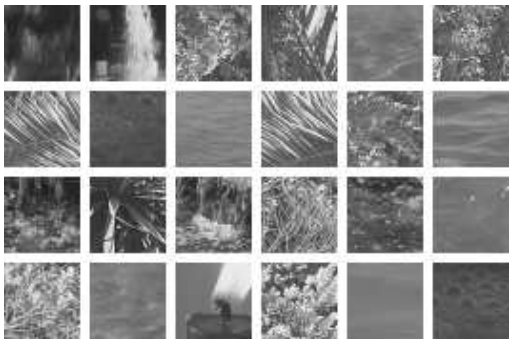


Figure 3. Examples from the UCLA-pan video texture database.

The inner-product between these two KPCA components is

$$\langle u_c, v_d \rangle = \left\langle \sum_{i=1}^{N_a} \alpha_{i,c} \phi(y_i^a), \sum_{i=1}^{N_b} \beta_{i,d} \psi(y_i^b) \right\rangle \tag{9}$$

$$= \alpha_c^T G \beta_d \tag{10}$$

where G is the matrix with entries $[G]_{i,j} = g(y_i^a, y_j^b)$, and $g(y_1, y_2) = \langle \phi(y_1), \psi(y_2) \rangle$. The function g is the inner-product in the feature-space between the two data-points, transformed by two *different* functions, $\phi(y)$ and $\psi(y)$. For two Gaussian kernels with bandwidth parameters σ_a^2 and σ_b^2, it can be shown that $g(y_1, y_2) = \exp(-\frac{1}{2} \| \frac{1}{\sigma_a} y_1 - \frac{1}{\sigma_b} y_2 \|^2)$ (see [18] for details). Finally, the inner product matrix between all the KPCA components is $F = \alpha^T G \beta$.

5. Experimental evaluation

In this section we evaluate the efficacy of the kernel dynamic texture for classification of video textures undergoing camera motion.

5.1. Databases

The UCLA dynamic texture database [3, 4] contains 50 classes of various video textures, including boiling water, fountains, fire, waterfalls, and plants and flowers swaying in the wind. Each class contains four grayscale sequences with 75 frames of 160×110 pixels. Each sequence was clipped to a 48×48 window that contained the representative motion.

A second database containing panning video textures was built from the original UCLA video textures. Each video texture was generated by panning a 40×40 window across the original UCLA video. Four pans (two left and two right) were generated for each video sequence, resulting in a database of 800 panning textures, which we call the UCLA-pan database. The motion in this database is composed of both video textures and camera panning, hence the dynamic texture is not expected to perform well on it. Examples of the UCLA-pan database appear in Figure 3, and video montages of both databases are available from [18].

5.2. Experimental setup

A kernel dynamic texture was learned for each video in the database using Algorithm 1 and a centered Gaussian kernel with bandwidth parameter σ^2, estimated for each video as

$$\sigma^2 = \frac{1}{2}\text{median}\{\|y_i - y_j\|^2\}_{i,j=1,\ldots,N} \quad (11)$$

Both nearest neighbor (NN) and SVM classifiers [26] were trained using the Martin distance for the kernel dynamic texture. The SVM used an RBF-kernel based on the Martin distance, $k_{md}(\Theta_a, \Theta_b) = e^{-\frac{1}{2\sigma^2}d^2(\Theta_1,\Theta_2)}$. A one-versus-all scheme was used to learn the multi-class SVM problem, and the C and γ parameters were selected using three-fold cross-validation over the training set. We used the `libsvm` package [27] to train and test the SVM.

For comparison, a NN classifier using the Martin distance on the standard dynamic texture [3] was trained, along with a corresponding SVM classifier. NN and SVM classifiers using the image-space KL-divergence between dynamic textures [4] were also trained. Finally, experimental results were averaged over four trials, where in each trial the databases were split differently with 75% of data for training and cross-validation, and 25% of the data for testing.

5.3. Results

Figures 4 (a) and (c) show the NN classifier performance versus n, the number of principal components (or the dimension of the state space). While the NN classifier based on the kernel dynamic texture and Martin distance (KDT-MD) performs similarly to the dynamic texture (DT-MD) on the UCLA database, KDT-MD outperforms DT-MD for all values of n on the UCLA-pan database. The best accuracy increases from 84.3% to $89.8.\%$ on the UCLA-pan database when using KDT-MD instead of DT-MD, while only increasing from 89.0% to 89.5% on the UCLA database. This indicates that the panning motion in the UCLA-pan database is not well modeled by the dynamic texture, whereas the kernel dynamic texture has better success. The performance of the SVM classifiers is shown in Figures 4 (b) and (d). The dynamic texture and kernel dynamic texture perform similarly on the UCLA database, with both improving over their corresponding NN classifier. However, the KDT-MD SVM outperforms the DT-MD SVM on the UCLA-pan database (accuracies of 94.3% and 92.8%, respectively).

When looking at the performance of the KL-based classifiers, we note that for the UCLA databases the mean-image of the video is highly discriminative for classification. This can be seen in Figures 4 (a), where the accuracy of the KL-divergence NN classifier is plotted for dynamic textures learned from the normal data (DT-KL) and from zero-mean data (DT-KL0). The best performance for DT-KL occurs when $n = 0$, i.e. the video is simply modeled as the mean image with some i.i.d. Gaussian noise. On the other hand, when the mean is ignored in DT-KL0, the performance of the classifier drops dramatically (from 94% to 15% accuracy for $n = 0$). Hence, much of the discriminative power of the KL-based classifier comes from the similarity of image means, not from video motion. Because the Martin distance does not use the image means, we present the classification results for DT-KL0 to facilitate a fair comparison between the classifiers. The DT-KL0 NN classifier performed worse than both KDT-MD and DT-MD, as seen in Figures 4 (a) and (c). The SVM trained on DT-KL0 improved the performance over the DT-KL0 NN classifier, but is still inferior to the KDT-MD SVM classifiers. A summary of the results on the UCLA and UCLA-pan databases is given in Table 1.

Database	KDT-MD	DT-MD	DT-KL0
UCLA NN	$\mathbf{0.895}$ (20)	0.890 (15)	0.365 (2)
UCLA SVM	$\mathbf{0.975}$ (20)	0.965 (15)	0.725 (2)
UCLA-pan NN	$\mathbf{0.898}$ (30)	0.843 (30)	0.816 (5)
UCLA-pan SVM	$\mathbf{0.943}$ (30)	0.928 (25)	0.920 (5)

Table 1. Classification results for the UCLA and UCLA-pan databases. (n) is the number of principal components.

Finally, Figure 5 shows the distance matrix for DT-MD and KDT-MD for three classes from UCLA-pan: two classes of water falling from a ledge, and one class of boiling water (see Figure 5 (right) for examples). The DT-MD performs poorly on many of these sequences because the water motion is chaotic, i.e. there are many discontinuities in the pixel values. On the other hand, KDT-MD models the discontinuities, and hence can distinguish between the different types of chaotic water motion.

Acknowledgments

The authors thank Benjamin Recht for helpful discussions, and Gianfranco Doretto and Stefano Soatto for the database from [3]. This work was partially funded by NSF award IIS-0534985 and NSF IGERT award DGE-0333451.

References

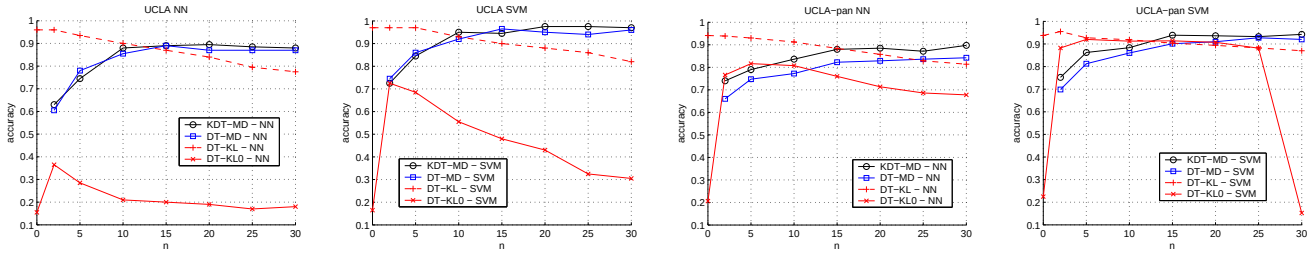
[1] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto, "Dynamic textures," *Intl. J. Computer Vision*, vol. 51, no. 2, pp. 91–109, 2003.

[2] A. B. Chan and N. Vasconcelos, "Layered dynamic textures," in *Neural Information Processing Systems 18*, 2006, pp. 203–10.

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

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

[5] S. V. N. Vishwanathan, A. J. Smola, and R. Vidal, "Binet-cauchy kernels on dynamical systems and its application to the analysis of dynamic scenes," *IJCV*, vol. 73, no. 1, pp. 95–119, 2007.



(a) (b) (c) (d)

Figure 4. Classification results on the UCLA database using: (a) nearest neighbors and (b) SVM classifiers; and results on the UCLA-pan database using: (c) nearest neighbors and (d) SVM classifiers. Classifier accuracy is plotted versus the number of principal components n.

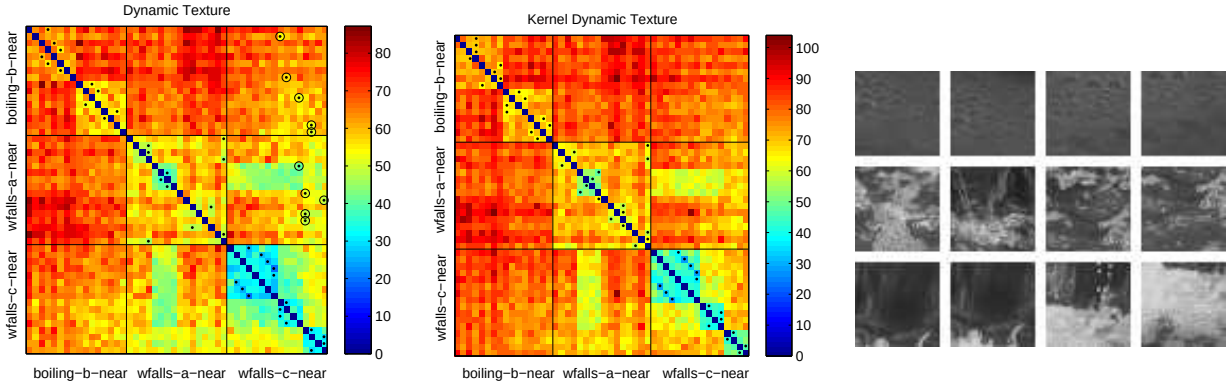


Figure 5. Misclassification of chaotic water: the Martin distance matrices for three water classes using (left) dynamic textures, and (middle) kernel dynamic textures. Nearest neighbors in each row are indicated by a black dot, and the misclassifications are circled. (right) Four examples from each of the three water classes.

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

[7] R. Vidal and A. Ravichandran, "Optical flow estimation & segmentation of multiple moving dynamic textures," in *CVPR*, 2005.

[8] A. B. Chan and N. Vasconcelos, "Mixtures of dynamic textures," in *IEEE Intl. Conf. Computer Vision*, vol. 1, 2005, pp. 641–7.

[9] L. Yuan, F. Wen, C. Liu, and H.-Y. Shum, "Synthesizing dynamic textures with closed-loop linear dynamic systems," in *Euro. Conf. Computer Vision*, 2004, pp. 603–616.

[10] C.-B. Liu, R.-S. Lin, N. Ahuja, and M.-H. Yang, "Dynamic texture synthesis as nonlinear manifold learning and traversing," in *British Machine Vision Conf.*, vol. 2, 2006, pp. 859–868.

[11] C.-B. Liu, R.-S. Lin, and N. Ahuja, "Modeling dynamic textures using subspace mixtures," in *ICME*, 2005, pp. 1378–81.

[12] G. Doretto and S. Soatto, "Towards plenoptic dynamic textures," in *3rd Intl. Work. Texture Analysis and Synthesis*, 2003, pp. 25–30.

[13] B. Schölkopf, A. Smola, and K. R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[14] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. John Wiley and Sons, 2001.

[15] B. Schölkopf, S. Mika, A. Smola, G. Rätsch, and K. R. Müller, "Kernel pca pattern reconstruction via approximate pre-images," in *ICANN, Perspectives in Neural Computing*, 1998, pp. 147–52.

[16] S. Mika, B. Schölkopf, A. Smola, K. R. Müller, M. Scholz, and G. Rätsch, "Kernel PCA and de-noising in feature spaces," in *Neural Information Processing Systems*, vol. 11, 1999, pp. 536–52.

[17] J.-Y. Kwok and I.-H. Tsang, "The pre-image problem in kernel methods," *IEEE Trans. Neural Networks*, vol. 15, no. 6, 2004.

[18] A. B. Chan and N. Vasconcelos, "Supplemental material for 'classifying video with kernel dynamic textures'," Statistical Visual Computing Lab, Tech. Rep. SVCL-TR-2007-03, 2007, `http://www.svcl.ucsd.edu`.

[19] Z. Ghahramani and S. T. Roweis, "Learning nonlinear dynamical systems using an EM algorithm," in *NIPS*, 1998.

[20] H. Valpola and J. Karhunen, "An unsupervised ensemble learning method for nonlinear dynamic state-space models," *Neural Computation*, vol. 14, no. 11, pp. 2647–92, 2002.

[21] J. Ham, D. D. Lee, S. Mika, and B. Schölkopf, "A kernel view of the dimensionality reduction of manifolds," in *ICML*, 2004.

[22] A. Rahimi, B. Recht, and T. Darrell, "Learning appearance manifolds from video," in *IEEE CVPR*, vol. 1, 2005, pp. 868–75.

[23] A. Rahimi and B. Recht, "Estimating observation functions in dynamical systems using unsupervised regression," in *NIPS*, 2006.

[24] R. J. Martin, "A metric for ARMA processes," *IEEE Transactions on Signal Processing*, vol. 48, no. 4, pp. 1164–70, April 2000.

[25] K. D. Cock and B. D. Moor, "Subspace angles between linear stochastic models," in *IEEE Conf. Decision and Control*, 2000.

[26] V. N. Vapnik, *The nature of statistical learning theory*. Springer-Verlag, 1995.

[27] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, `http://www.csie.ntu.edu.tw/~cjlin`.