

## Video epitomes

Vincent Cheung<sup>1</sup>, Brendan J. Frey<sup>1</sup>, Nebojsa Jojic<sup>2</sup>

<sup>1</sup>Electrical and Computer Engineering, University of Toronto, Toronto, ON, M5S 3G4, Canada

<sup>2</sup>Machine Learning and Applied Statistics, Microsoft Research, Redmond, WA, 98052, USA

### Abstract

Recently, “epitomes” were introduced as patch-based probability models that are learned by compiling together a large number of examples of patches from input images. In this paper, we describe how epitomes can be used to model video data and we describe significant computational speedups that can be incorporated into the epitome inference and learning algorithm. In the case of videos, epitomes are estimated so as to model most of the small space-time cubes from the input data. Then, the epitome can be used for various modeling and reconstruction tasks, of which we show results for video super-resolution, video interpolation, and object removal. Besides computational efficiency, an interesting advantage of the epitome as a representation is that it can be reliably estimated even from videos with large amounts of missing data. We illustrate this ability on the task of reconstructing the dropped frames in video broadcast using only the degraded video.

### 1. Introduction

The technique of using small image patches to account for high-order statistics in image and video data continues to grow in popularity in the vision community. One of the first uses of patches was to find corresponding points in neighboring video frames for computing optical flow. Instead of simply matching patches, Jepson and Black introduced probabilistic patch models that account for outliers [5]. Soon after that, Wang and Adelson showed that patches could be used for efficient video compression [9]. In related work, “textons” use image patches within a structured mathematical framework, to account for texture using a patch-based representation [11]. In these cases, patches were used primarily for analyzing image data.

More recently, patches have been used successfully for synthesizing images and videos. Patches from one part of an image have been stitched together to synthesize new images with similar texture, or to in-paint texture into an interior region [3, 2]. This approach has also been used to fill in missing or occluded regions of video data [10]. Libraries

of patches derived from high-resolution images have been used for super-resolution, both in static images and video sequences [4, 1]. Patch-based image models have been used for the purpose of “texture transfer”, also known as “unsupervised image analogies” [8].

To jointly analyze and synthesize data, patch-based probability models are introduced in [6]. These models, called “epitomes”, compile patches drawn from input images into a condensed image model that represents many of the high-order statistics in the input image. An advantage of learning a patch-based model from the input data is that the model can be used for a variety of inference tasks, such as image/texture classification, in-painting, super-resolution, texture transfer, and image segmentation [6]. Epitomes have found uses in other application areas, including speech and molecular biology.

In this paper, we address two problems: (1) How 2D image epitomes can be extended through time to form 3D space-time epitomes; (2) How epitomes can be compiled and applied in a computationally efficient manner and in particular, in a way that is significantly more efficient than using a library of patches. After describing a general framework for learning video epitomes, we demonstrate its use in several interesting applications.

### 2. Video epitomes

Fig. 1 outlines the procedure used to learn a video epitome. Viewing the input video as a 3D space-time volume, a large number of 3D training patches are drawn from the video. The learning algorithm is used to compile these patches into an “epitome” – a video that is smaller in space and time, but contains many of the spatial and temporal patterns in the input video. We derive the epitome learning algorithm by specifying a generative model, which explains how the input video can be generated from the epitome (in the opposite direction shown in the figure). The advantage of specifying a generative model is that the model is more adaptive than a non-generative technique and it can be used as a sub-component in other systems. Here, we introduce a 3D model of video similar to the 2D epitome model described in [6].

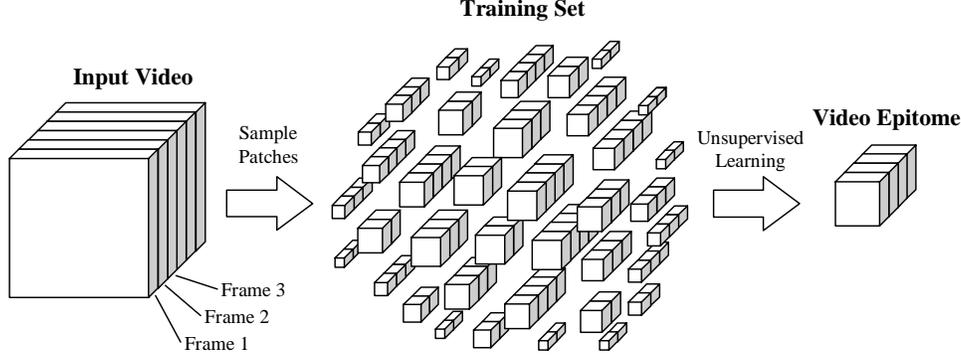


Figure 1. Learning the epitome of a video.

We treat a video sequence as a 3D array  $\mathbf{v}_{x,y,t}$  of real-valued pixel measurements (R,G, and B color channels in this paper), with  $x \in: \{1, \dots, X_v\}$ ,  $y \in: \{1, \dots, Y_v\}$ ,  $t \in: \{1, \dots, T_v\}$ . The epitome  $e$  models the video using a set of probability distributions arranged on a grid of size  $X_e \times Y_e \times T_e$ , considerably smaller than the video, *i.e.*,  $X_e Y_e T_e < X_v Y_v T_v$ . We view the epitome  $e_{x,y,t}$  as a 3D array of probability distributions. A particular pixel value  $\mathbf{v}$  can be evaluated under any of the probability distributions in  $e$ . For example, for the epitome coordinates  $x_e, y_e, t_e$ , the probability density at the pixel value stored in the entry  $x_v, y_v, t_v$  of the video is  $e_{x_e, y_e, t_e}(\mathbf{v}_{x_v, y_v, t_v})$ . Since the pixel measurements are continuous in nature, it is necessary to parameterize each of the epitome distributions. In our experiments, we use a single parametric form, a three-dimensional Gaussian distribution parameterized by a different mean and diagonal covariance matrix for each entry,

$$e_{x,y,t}(\cdot) = \mathcal{N}(\cdot; \mu_{x,y,t}, \phi_{x,y,t}), \quad (1)$$

where  $\mu_{x,y,t}$  is the mean and  $\phi_{x,y,t}$  is the covariance matrix (*e.g.*, for RGB values). The diagonal covariance matrix decouples color channel computations. Because of this, we will treat the measurements  $\mathbf{v}_{x,y,t}$  as scalar in the following derivations, which the reader can use to derive the full color model.

The epitome models the video by modeling 3D patches sampled from the video. These patches can have any shape, but to keep notation simple, we will assume each patch has linear, axis-aligned boundaries, and we think of each patch as a “cube”. Each patch is defined in terms of the ordered set of pixel coordinates in the patch,  $\mathcal{S}$ . For example, a  $10 \times 10 \times 5$  video patch starting at position 8,9 in frame 7 of the video is described by the set  $\mathcal{S} = \{8, \dots, 17\} \times \{9, \dots, 18\} \times \{7, \dots, 11\}$ . We assume the coordinates in  $\mathcal{S}$  are ordered, so that  $\mathcal{S}(k)$  refers to the  $k$ th coordinate in  $\mathcal{S}$ , *e.g.*,  $\mathcal{S}(1) = (8, 9, 7)$  in the above example.

While  $\mathbf{v}$  denotes the observed pixel values at all coordinates in the video,  $\mathbf{v}_{\mathcal{S}}$  denotes the observed pixel values

in a small video cube at coordinates  $\mathcal{S}$ , and  $\mathbf{c}_{\mathcal{S}}$  denotes the pixel values in the same cube, as predicted by the epitome. As described below, the goal of the learning algorithm is to make the predicted video cubes similar to the observed video cubes, *i.e.*,  $\mathbf{c}_{\mathcal{S}} \approx \mathbf{v}_{\mathcal{S}}$ . The cube  $\mathbf{c}_{\mathcal{S}}$  is modeled using a set of distributions in the epitome. We use  $e_{\mathcal{T}}$  to denote the set of distributions from the epitome  $e$  at coordinates  $\mathcal{T}$ . Assuming that the cube corresponding to  $\mathcal{T}$  is equal in size to cube corresponding to  $\mathcal{S}$ , we can define a one-to-one coordinate correspondence so that the set ordering is preserved. This allows us to use the notation  $e_{\mathcal{T}}(\mathbf{c}_{\mathcal{S}})$  for the probability density evaluated using distributions at coordinates  $\mathcal{T}$  and predicted values  $\mathbf{c}_{\mathcal{S}}$ :

$$p(\mathbf{c}_{\mathcal{S}} | \mathcal{T}_{\mathcal{S}}) = e_{\mathcal{T}_{\mathcal{S}}}(\mathbf{c}_{\mathcal{S}}) = \prod_{k=1}^{|\mathcal{T}|} e_{\mathcal{T}_{\mathcal{S}}(k)}(\mathbf{c}_{\mathcal{S}(k)}), \quad (2)$$

where the notation  $\mathcal{T}_{\mathcal{S}}$  indicates the assumed responsibility of  $\mathcal{T}$  for patch  $\mathcal{S}$ . The above equation assumes independence of pixels given the responsible set of epitome distributions, with the idea of capturing correlations simply through mapping sets of measurements to the overlapping sets of distributions.

The above equation describes the probability model for an individual cube. To obtain a probability model for the entire input video,  $\mathbf{v}$ , we need to address the issue of how to account for overlapping cubes. If two sets of coordinates  $\mathcal{S}$  and  $\mathcal{S}'$  overlap, then the corresponding cubes  $\mathbf{c}_{\mathcal{S}}$  and  $\mathbf{c}_{\mathcal{S}'}$  are used to predict overlapping cubes in the input video  $\mathbf{v}$ , and so they should make similar predictions for overlapping pixels. The most obvious approach to accounting for overlap is to include a constraint that the predictions are identical in regions of overlap. However, this approach requires the introduction of a partition function, making learning significantly more difficult. We take a different, novel, approach [6], where we treat the cubes  $\mathbf{c}_{\mathcal{S}}$  as independent even if they share coordinates, but enforce the agreement in the overlapping regions during inference, as described in the next section.

We now define a generative model of video sequences that is suitable for all applications described in the experimental section. The first step in the generative process consists of generating a predicted cube  $\mathbf{c}_S$  for every possible coordinate set  $S$  in the input video. This is accomplished by first randomly choosing a patch  $\mathcal{T}_S$  from the epitome using a uniform distribution, and then generating  $\mathbf{c}_S$  using the distribution  $e_{\mathcal{T}_S}(\mathbf{c}_S)$  defined by (2). Then, for each pixel coordinate  $x, y, t$  in the video, all overlapping cubes are combined to make a single prediction for the observed pixel  $\mathbf{v}_{x,y,t}$  at that coordinate. During free energy minimization, we will force the predictions to agree, so the exact form of the combination function is not important. Here, we assume that to generate a consistent video pixel  $\mathbf{v}_{x,y,t}$ , the contributions from all overlapping video cubes  $\{S : (x, y, t) \in S\}$  are averaged, and Gaussian noise with variance  $\sigma_{x,y,t}^2$  is added to all three channels:

$$\begin{aligned} p(\mathbf{v}_{x,y,t} | \{\mathbf{c}_S : (x, y, t) \in S\}) &= \\ \mathcal{N}(\mathbf{v}_{x,y,t}; \frac{\sum_S \sum_k [S(k) = (x, y, t)] \mathbf{c}_{S,k}}{\sum_S \sum_k [S(k) = (x, y, t)]}, \sigma_{x,y,t}^2) & \\ p(\mathbf{v} | \{\mathbf{c}_S\}) = \prod_{x,y,t} p(\mathbf{v}_{x,y,t} | \{\mathbf{c}_S : (x, y, t) \in S\}), & \quad (3) \end{aligned}$$

where  $[ \ ]$  is Iverson's indicator function, *i.e.*,  $[true] = 1$ ,  $[false] = 0$ . We use the notation  $\mathbf{c}_{S,k}$  for the  $k$ -th pixel in  $\mathbf{c}_S$  to emphasize that the video cubes  $\mathbf{c}_S$  are treated as independent, and so a pixel in the video cube  $\mathbf{c}_S$  is not uniquely defined by the global coordinate  $S(k)$ , and is instead, potentially different from the pixels in other cubes that overlap  $S(k)$ . However, as described by the above equations, for each coordinate  $(x, y, t)$ , a single final pixel  $\mathbf{v}_{x,y,t}$  is generated by adding noise to the average of all video cubes overlapping with coordinate  $(x, y, t)$ .

Usually, a subset of the input video patches provides accurate sufficient statistics for learning an epitome. In practice, we randomly sample the set  $\{S\}$  of patches to be used for learning (*e.g.*, from the set of all video cubes of a certain size). The joint distribution over all variables can be written  $p(\mathbf{v}, \{\mathbf{c}_S, \mathcal{T}_S\}) = p(\mathbf{v} | \{\mathbf{c}_S\}) \prod_S p(\mathbf{c}_S | \mathcal{T}_S) p(\mathcal{T}_S)$ . We often assume  $p(\mathcal{T}_S)$  is uniform for the sake of simplicity.

### 3. Learning video epitomes

Learning entails minimizing a free energy cost function:

$$F = \sum_S \sum_{\mathcal{T}_S} \int_{\mathbf{c}_S} q(\{\mathcal{T}_S, \mathbf{c}_S\}) \log \frac{q(\{\mathcal{T}_S, \mathbf{c}_S\})}{p(\mathbf{v}, \{\mathbf{c}_S, \mathcal{T}_S\})}, \quad (4)$$

where  $q(\{\mathcal{T}_S, \mathbf{c}_S\})$  is the auxiliary joint probability distribution over the set of epitome patches  $\{\mathcal{T}_S\}$  and the set of video cubes  $\{\mathbf{c}_S\}$  for all coordinate patches  $\{S\}$ . It turns out that the free energy [7] bounds the log-likelihood

of the input video,  $F \geq -\log p(\mathbf{v})$ , where  $p(\mathbf{v}) = \sum_{\{\mathcal{T}_S\}} \int_{\{\mathbf{c}_S\}} p(\mathbf{v}, \{\mathbf{c}_S, \mathcal{T}_S\})$ . So, by optimizing  $F$ , we can maximize  $p(\mathbf{v})$ . In fact, the closer  $q$  is to the true posterior  $p(\{\mathcal{T}_S, \mathbf{c}_S\} | \mathbf{v})$ , the tighter the bound, so  $q$  is an approximation to the posterior distribution. By choosing an appropriate form for  $q$ , we can achieve two goals: Obtain an efficient inference and learning algorithm by decoupling variables [7]; Constrain inference and learning to the space where overlapping cubes agree, as proposed in [6]. We choose the  $q$ -distribution as follows:

$$q(\{\mathcal{T}_S, \mathbf{c}_S\}) = \prod_S q(\mathcal{T}_S) q(\mathbf{c}_S). \quad (5)$$

$q(\mathcal{T}_S)$  is a discrete distribution over all possible patch locations in the epitome shaped as  $S$ . To enforce the overlap constraint,  $q(\mathbf{c}_S)$  is a product of Dirac functions:

$$q(\mathbf{c}_S) = \prod_k \delta(\mathbf{c}_{S,k} - \nu_{S(k)}). \quad (6)$$

As before,  $\mathbf{c}_{S,k}$  denotes the pixel  $S(k)$  in cube  $\mathbf{c}_S$ .  $\nu_{S(k)}$  denotes the variational parameter for the pixel  $S(k) = (x, y, t)$ , which is *shared* by all patches  $S$  that contain the coordinate  $(x, y, t)$ , thus constraining the patches to agree in overlapping pixels.

The free energy obtained using the above  $q$ -distribution leads to a tractable iterative learning algorithm. Setting to zero the derivatives of  $F$  w.r.t. the posterior cube colors  $\nu_{x,y,t}$ , we obtain the following update rule:

$$\nu_{x,y,t} \leftarrow \frac{\frac{\nu_{x,y,t}}{\sigma_{x,y,t}^2} + \sum_{S,k:S(k)=(x,y,t)} \sum_{\mathcal{T}_S} q(\mathcal{T}_S) \frac{\mu_{\mathcal{T}_S(k)}}{\phi_{\mathcal{T}_S(k)}}}{\frac{1}{\sigma_{x,y,t}^2} + \sum_{S,k:S(k)=(x,y,t)} \sum_{\mathcal{T}_S} q(\mathcal{T}_S) \frac{1}{\phi_{\mathcal{T}_S(k)}}}. \quad (7)$$

This update sets the ‘‘hidden video’’  $\nu$  to a weighted combination of the input video and the top-down prediction for the video, as given by the current epitome. The weights are the inverse noise variances for the video and the epitome. Setting to zero the derivatives of  $F$  w.r.t. the posterior epitome responsibilities  $q(\mathcal{T}_S)$ , we obtain

$$q(\mathcal{T}_S) \leftarrow \frac{p(\mathcal{T}_S) e_{\mathcal{T}_S}(\nu_S)}{\sum_{\mathcal{T}} p(\mathcal{T}) e_{\mathcal{T}}(\nu_S)}. \quad (8)$$

This update is similar to computing the responsibilities of components in a mixture of Gaussians. For each cube  $\nu_S$ , the distribution over its position in the epitome is proportional to the probability it was generated from each position. To perform inference on video  $\mathbf{v}$  when the epitome is given, these two updates can be iterated until convergence.

If the epitome is to be learned from the video  $\mathbf{v}$ , then the two equations above are iterated in combination with the following two updates, obtained by setting to zero the

derivatives of  $F$  w.r.t. the epitome parameters  $\mu_{x,y,t}$  and  $\phi_{x,y,t}$ :

$$\mu_{x_e, y_e, t_e} \leftarrow \frac{\sum_{\mathcal{T}, k: \mathcal{T}(k)=(x_e, y_e, t_e)} \sum_{\mathcal{S}} q(\mathcal{T}_S = \mathcal{T}) \nu_{\mathcal{S}(k)}}{\sum_{\mathcal{T}, k: \mathcal{T}(k)=(x_e, y_e, t_e)} \sum_{\mathcal{S}} q(\mathcal{T}_S = \mathcal{T})} \quad (9)$$

$$\phi_{x_e, y_e, t_e} \leftarrow \frac{\sum_{\mathcal{T}, k: \mathcal{T}(k)=(x_e, y_e, t_e)} \sum_{\mathcal{S}} q(\mathcal{T}_S = \mathcal{T}) (\nu_{\mathcal{S}(k)} - \mu_{x_e, y_e, t_e})^2}{\sum_{\mathcal{T}, k: \mathcal{T}(k)=(x_e, y_e, t_e)} \sum_{\mathcal{S}} q(\mathcal{T}_S = \mathcal{T})}. \quad (10)$$

The first update sets the mean pixel value in the epitome to the average value of all pixel values from video cubes  $\nu_{\mathcal{S}}$ , weighted by the probability that the cube is aligned with the pixel in the epitome,  $q(\mathcal{T}_S = \mathcal{T})$ . The second update is similar, except it accounts for the variance, not the mean value. All inference algorithms in this paper are based on these four equations. In the next section, we explain how these equations define different video processing applications. Then, in Section 5, we derive an efficient algorithm that implements these equations.

## 4. Epitomic video processing

First, we note that a very high variance parameter  $\sigma_{x,y,t}^2$  effectively severs all the video cubes  $\mathbf{c}_{\mathcal{S}}$  from the observation  $\mathbf{v}_{x,y,t}$ . According to the generative model equation (3), an excessive level of noise  $\sigma_{x,y,t}^2$  will make the generated value  $\mathbf{v}_{x,y,t}$  dominated by Gaussian noise and thus independent from the epitome generated video cubes. Because of this, the inferred color  $\nu_{x,y,t}$  shared for the coordinate  $(x, y, t)$  in all video cubes is dominated by the epitome prediction, as seen in (7). When  $\sigma_{x,y,t}^2$  is very low, on the other hand, (7) is dominated by the observation  $\mathbf{v}_{x,y,t}$ . This general observation leads to several video reconstruction applications of the inference algorithm described in the previous section, including filling-in missing data, obstruction removal, video interpolation, denoising, and super-resolution.

In a large area of high variance  $\sigma^2$ , iterating (7) and (8) will fill the inferred values  $\nu$  with the epitome generated cubes that tend to agree in the overlapping regions. To better understand this, consider randomly initialized set of  $\nu$  values. The inference step (8) for the cube  $\mathbf{c}_{\mathcal{S}}$  takes the current guess at  $\nu_{\mathcal{S}}$  and evaluates how likely each epitome cube  $e_{\mathcal{T}}$  is to generate  $\nu_{\mathcal{S}}$ . Since the posterior epitome responsibilities share the  $\nu$  values, the (probabilistically) chosen epitome cubes tend to have a higher level of agreement in the overlaps than if a set of random epitome cubes were selected as responsible for the video cubes. Now, applying (7) will replace the initialized values  $\nu$  with the average votes from the epitome cubes deemed likely to have been generated according to  $q(\mathcal{T}_S)$ . Since the likely video cubes have a moderate level of agreement, the generated  $\nu$  texture

will now be more consistent with the epitome texture. Iterating these two steps will lead to the solution where  $\mathcal{T}_S$  are chosen so as to “quilt” a random, but consistent texture from the epitome. Note also, that in these steps, each new estimate for a  $\nu_{x,y,t}$  is a weighted average of *all* epitome means  $\mu_{x_e, y_e, t_e}$  weighted by both the inverse epitome variances  $\phi_{x_e, y_e, t_e}$  and the probabilities of all possible cube mappings that lead to mapping epitome entry  $(x_e, y_e, t_e)$  to video entry  $(x, y, t)$ . After several iterations, however, the  $q(\mathcal{T}_S)$  become fairly peaked and consistent.<sup>1</sup>

If the area of high variance  $\sigma^2$  is surrounded by the area of low variance, then the quilting steps described above will be automatically conditioned on satisfying the agreement of the  $\nu$  values in the areas of low variance. Thus, we can perform video repair operations by setting  $\sigma^2$  values to be high in the areas of video that are either missing, or considered bad, or that we simply want to replace with the texture from the epitome.

An interesting additional property of the algorithm in the previous section is that the epitome learning can be performed *jointly* with the repair operations, even when some of the input video pixels are known to be unreliable. If the unreliable (or unwanted) pixels are given high noise parameters  $\sigma^2$ , then iterating all four of the equations (7-10) will construct the epitome only from the reliable pixels. (Only the reliable pixels propagate into  $\nu$  values, which in turn define the epitome statistics in each step (10)).

Next, we define several video reconstruction tasks by using a given  $X \times Y \times T$  video  $\mathbf{f}_{x,y,t}$  to form the input  $\mathbf{v}_{x,y,t}$ ,  $\sigma_{x,y,t}^2$  to the epitome inference algorithm, with all parameters initialized in an uninformative way. We consider the resulting set of parameters  $\nu$  a video reconstruction. In comparison to “texture quilting” techniques used in [3] for example, epitome offers both better generalization and potentially greater computational efficiency. The cost of learning an epitome is proportional to the product of the data size and the epitome size, while the cost of the quilting operations is proportional to the product of the library size and the reconstruction size. If the epitome is much smaller than the training data, then the total cost of using the epitome as the library becomes lower than the cost of using the entire training video as the library. Furthermore, epitomes can be used as a set of pointers to the original training data, so that upon the computation of the posterior  $Q(\mathcal{T}_S)$ , the training data cubes corresponding to the several best matching

<sup>1</sup>For example, if the responsible epitome cube for the video cube at  $\{1, \dots, 5\} \times \{1, \dots, 5\} \times \{1, \dots, 5\}$  is at epitome coordinates  $\{11, \dots, 15\} \times \{11, \dots, 15\} \times \{11, \dots, 15\}$ , then the video cube at  $\{2, \dots, 6\} \times \{3, \dots, 7\} \times \{4, \dots, 8\}$  will tend to be mapped to epitome coordinates  $\{12, \dots, 16\} \times \{13, \dots, 17\} \times \{14, \dots, 18\}$ , so that for the instance, the video coordinate  $(3, 4, 5)$  maps to the epitome coordinate  $(13, 14, 15)$  in both mappings. When the inference procedure does not result in such translational consistency, then it usually results in appearance consistency, i.e., each pixel tends to map to similar distributions  $e_{x,y,t}$  in all mappings.

epitome cubes  $\mathcal{T}_S$  can be searched over to find potentially even better candidates for quilting. This offers the advantage of using all the original training cubes in quilting while still using the computational benefit and regularization of the epitome representation.

#### 4.1. Denoising

If the noise level  $\sigma^2$  in the given video  $\mathbf{f}$  is known and constant, then we set  $\mathbf{v} = \mathbf{f}$  and  $\sigma_{x,y,t}^2 = \sigma^2$  and iterate (7-10), starting with non-informative initialization. This procedure effectively performs denoising by averaging similar pieces of the video. The video epitome serves as a nexus of self-similarity estimation, and iteration is necessary to estimate the epitome jointly with the denoising operation. The size of the epitome is critical in this operation. If the epitome is assumed to be very large, then, the denoising operation may still copy a lot of noisy pixels into the epitome. On the other hand, a small epitome will be forced to be more liberal in the definition of similarity, due to the lack of resources to capture the diversity. Averaging many patches into a small epitome may lead to excessive blurring.

If the noise level  $\sigma^2$  is not known, but is assumed to be *uniform* in the video, it can be estimated from the data by adding the following step to the algorithm

$$\sigma^2 = \frac{1}{XYT} (\nu_{x,y,t} - \mathbf{v}_{x,y,t})^2, \quad (11)$$

which follows from setting to zero the derivative of  $F$  with respect to  $\sigma^2$ .

#### 4.2. Super-resolution

Suppose that we want to increase the video resolution of the given video  $\mathbf{f}_{x,y,t}$ . To guide us in this task, let us assume that we have a high resolution video  $\mathbf{h}_{x,y,t}$  of a similar scene or scenes. We can then iterate (7-10) using  $\mathbf{v} = \mathbf{h}$  and small noise levels  $\sigma_{x,y,t}^2 = \epsilon$ . If  $\mathbf{h}$  is assumed to be somewhat noisy, we can increase  $\sigma_{x,y,t}^2$  or learn it as described in the previous subsection. The resulting high resolution epitome  $e$  can now be used to iterate only (7) and (8) on the input video defined as:

$$\begin{aligned} \mathbf{v}_{x,y,t} &= \mathbf{f}_{x/n,y/n,t} \\ \sigma_{x,y,t}^2 &= \epsilon + \sigma^2 [\text{mod}(x,n) > 0][\text{mod}(y,n) > 0], \end{aligned} \quad (12)$$

where  $\epsilon$  is a small number and  $\sigma^2$  is large, and  $[\ ]$  is an indicator function. In other words,  $\mathbf{v}$  is a  $nX \times nY \times T$  video created by nearest-neighbor resolution enhancement of  $\mathbf{f}$ , but the map  $\sigma_{x,y,t}^2$  labels as unreliable (noisy) the pixels which do not have coordinate divisible by  $n$ . Iterating (7) and (8) will replace the unreliable pixels with the best matching texture of  $e$  learned from  $\mathbf{h}$ .

The reader can appreciate that a similar approach can be utilized to fill in the missing “in-between” pixels even if the missing pixels are not uniformly inserted as above.

#### 4.3. Object removal and general missing data reconstruction

We now consider the general case of missing value reconstruction. Define the set of coordinates  $\mathcal{G}$  for which the measurements in  $\mathbf{v}$  are considered good, and the set of coordinates  $\mathcal{M}$  for which the measurements are missing, unreliable, or need to be replaced for any reason. When learning the epitome, the set of training patches  $\{\mathcal{S}\}$  is chosen so that none of the patches overlap with missing data. Then, by setting variances  $\sigma_{\mathcal{G}}^2 = \epsilon$  to be small and  $\sigma_{\mathcal{M}}^2 = \sigma^2$  to be large, the algorithm given by iterating (7), (8), (9), and (10), will produce the reconstruction  $\nu_{\mathcal{M}}$ , quilted from the epitome which captures the patterns in  $\mathbf{v}_{\mathcal{G}}$ . For example,  $\mathcal{M}$  could mark the spatio-temporal segment which has an unwanted object.

#### 4.4. Video interpolation

Video interpolation is a special case of the missing data problem and is also similar to that of super-resolution. Here, however, we describe a different version of the problem, one in which a similar video of higher temporal resolution is *not* given.

Formally, we can set up the problem as follows. For a sequence of frames  $\mathbf{f}_{x,y,u}$  we know that some frames are missing, so that the given sequence  $u = 1, \dots, U$  corresponds to the true frames  $t(u)$ . For instance, for a video interpolation task, we would have that  $t = \{1, 3, 5, 7, \dots\}$ . On the other hand, when a video is broadcast over the Internet, some clients will experience dropped frames in random shorter or longer bursts. Then,  $t$  follow a pattern like  $t = \{1, 2, 3, 4, 5, 15, 16, 17, 18, 19, 20, 21, 29, 30, 31, 32, \dots\}$ . For video interpolation tasks, we can define the input to the epitome algorithm as:

$$\begin{aligned} \mathbf{v}_{x,y,t(u)} &= \mathbf{f}_{x,y,u} \\ \sigma_{x,y,t(u)}^2 &= \epsilon, \quad \sigma_{x,y,t \neq t(u)}^2 = \sigma^2, \end{aligned} \quad (13)$$

with  $\epsilon$  and  $\sigma^2$  being a small and a large variance respectively. Then equations (7-10) are iterated to jointly learn the epitome and fill in the missing frames in the reconstruction  $\nu$ . When the pattern in  $t$  is uniform, as in the case of video interpolation, there is a danger that the epitome will be updated in a similar pattern, for example by decoupling the use of odd and even frames. It is thus generally useful to constrain the epitome learning so that this does not happen. A useful constraint is that the neighboring distributions  $e_{x,y,t}$  and  $e_{x,y,t+1}$  are similar. In our experiments we simply used potential functions  $\Psi_{x,y,t}$  connecting neighboring frames in the epitome as

$$\Psi_{x,y,t} = \max_{x_n, y_n, t_n \in N(x,y,t)} e^{-(\mu_{x,y,t} - \mu_{x_n, y_n, t_n})^2 / \psi}, \quad (14)$$

where  $N(x, y, t)$  denotes a neighborhood of  $x, y, t$  outside the frame  $\mathbf{t}$ , e.g.,  $N(x, y, t) = \{x - \delta x, \dots, x + \delta x\} \times \{y - \delta y, \dots, y + \delta y\} \times \{t - 1\} \cup \{x - \delta x, \dots, x + \delta x\} \times \{y - \delta y, \dots, y + \delta y\} \times \{t + 1\}$ . These potential functions ensure that the neighboring frames are deformed versions of one another, and define the prior

$$p(e) = \frac{1}{Z} \prod_{x_e, y_e, t_e} \Psi_{x_e, y_e, t_e}. \quad (15)$$

By including this prior and re-evaluating the derivative of the resulting free energy<sup>2</sup> with respect to the mean  $\mu_{x, y, t}$  we can see that (9) changes to

$$\mu_{x_e, y_e, t_e} = \frac{\mu_{\hat{x}_{en}, \hat{y}_{en}, \hat{t}_{en}} / \psi + \hat{\mu}_{x_e, y_e, t_e}^\nu / \phi_{x_e, y_e, t_e}}{1/\psi + 1/\phi_{x_e, y_e, t_e}}, \quad (16)$$

where  $\hat{\mu}_{x_e, y_e, t_e}^\nu$  denotes the estimate from (9), and  $(\hat{x}_{en}, \hat{y}_{en}, \hat{t}_{en}) = \arg \max_{(x_{en}, y_{en}, t_{en}) \in N(x_e, y_e, t_e)} \Psi_{x, y, t}$ .

To perform video interpolation we iterate (7), (8), (16), and (10).

Temporal interpolation and super-resolution can be combined in a straight forward manner to derive a super-resolved smooth video from a low-resolution video if it has enough pseudo-repeating structure and the sensor has a short spatio-temporal averaging field.

## 5. The shifted cumulative sum method

Careful study of the epitome learning rules reveals that many operations are repeated multiple times. For example, a single pixel  $\mathbf{v}_{x, y, t}$  in the video gets evaluated under every epitome distribution  $e_{x_e, y_e, t_e}$  several times in different combinations  $\mathcal{S}, \mathcal{T}$  in which  $(x, y, t) \in \mathcal{S}$  and  $(x_e, y_e, t_e) \in \mathcal{T}$ . Since the pixels are treated as independent, then for each patch of coordinates  $\mathcal{S}$ , a product of individual distributions is evaluated, or in the log domain, a sum of elements  $\log e_{x_e, y_e, t_e}(\nu_{x, y, t})$  is computed. The first observation to make here is that since the epitome algorithm requires that many different overlapping patches are evaluated in this way, it becomes computationally more efficient to compute  $\log e_{x_e, y_e, t_e}(\nu_{x, y, t})$  for all combinations of  $(x_e, y_e, t_e)$  and  $(x, y, t)$  and then sum the appropriate elements for each  $\mathcal{S}, \mathcal{T}$  combination, than to apply (8) directly.

Furthermore, overlapping patches even share many of the summations of the  $\log e_{x_e, y_e, t_e}(\nu_{x, y, t})$  terms. In fact, if the overlapping patches  $\mathcal{S}$  used in epitome operations are *all* possible cube patches of a given size (or a set of sizes), then these shared summations can be exploited in a systematic way. In particular, consider a *shifted* cumulative sum matrix

$$C(x_s, y_s, t_s) = \sum_{x < x_s, y < y_s, t < t_s} \log e_{\hat{x}_e, \hat{y}_e, \hat{t}_e}(\nu_{x, y, t}),$$

<sup>2</sup>We ignore the normalization constant  $Z$

where  $\hat{x}_e = (x_e + x_{es}) \bmod X_e$  and similarly for  $\hat{y}_e$  and  $\hat{t}_e$ , which is computed for a particular offset  $x_{es}, y_{es}, t_{es}$ . For the cube patch  $\mathcal{T}$  starting at  $\mathcal{T}(1) = (x_{es}, y_{es}, t_{es})$ , we can compute at once the epitome likelihoods for all equally shaped patches  $\mathcal{S}$  as a linear combination of up to eight shifted  $C$  matrices.<sup>3</sup> This means that for all  $XYT$  input cube patches  $\mathcal{S}$  of a certain size, and all  $X_e, Y_e, T_e$ , the computation of all epitome patch likelihoods is computed in  $o(XYT X_e Y_e T_e)$  time regardless of the patch size  $|\mathcal{S}|$ , thus allowing us to work with multiple size patches in our experiments, from very small patches capturing fine detail to very large patches that help strengthen the long-range correlation in epitome without increasing the computational complexity.

## 6. Experiments

**Video Super Resolution.** An optical zoom in modern cameras allows the user to trade the field of view for the level of captured detail. The user often desires to capture both the large context of the scene and the detail in it, by first capturing a wide angle shot followed by a large zoom (as shown in the set-up video on the web page) and slow scene scanning at the constant zoom level (omitted in the set-up video for brevity). We can use the approach described in Section 4.2 to super resolve the original low resolution wide-angle shot  $\mathbf{f}$ , using the epitome learnt on  $\mathbf{h}$ , the high-resolution “scene scanning” video captured at the higher zoom level.

In our example, the wide shot captures a large plant moving in the wind. Later, the camera zooms in and scans the plant creating the shot used for training the high resolution epitome. As described in Section 4.2, the high resolution textural and shape features of the plant, as well as the motion patterns, are then used to compute a super-resolved version of the wide shot. Note that the content of the wide shot is similar but far from identical to the content of the high resolution training data. Fig. 2 shows a single frame of the super-resolution result compared to bi-cubic interpolation, and the web page also shows a larger super-resolution sequence of the plant.

**Reconstructing dropped frames from a video broadcast.** Real-time streaming videos are becoming increasingly popular on the web. In streaming video, often video frames are dropped because of a lack of bandwidth. Client-side recovery of these missing frames using only the successfully received frames would greatly increase the experience with streaming videos.

Fig. 3 shows a few frames from a video sequence where the dropped frames effect was simulated and the missing

<sup>3</sup>For example, for  $\mathcal{T} = \{2\} \times \{5, \dots, 10\} \times \{10, \dots, 20\}$ , if we set  $(x_{es}, y_{es}, t_{es}) = (2, 5, 10)$ , the resulting shifted cumulative matrix  $C$  leads to  $\log e_{\mathcal{T}}(\nu_{[17] \times [15, 20] \times [50, 60]}) = C(17, 20, 60) - C(17, 15, 60) - C(17, 20, 50) + C(17, 15, 50)$ .

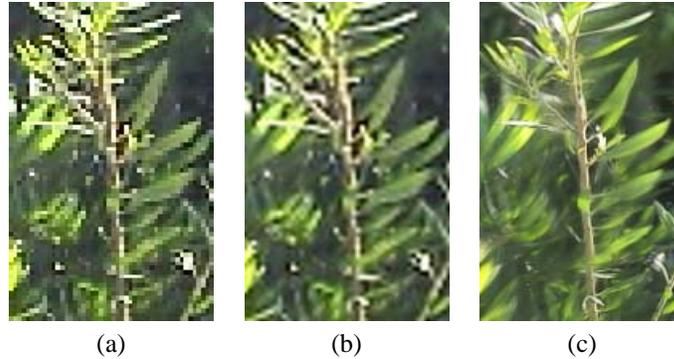


Figure 2. Video super resolution. (a) A single frame from a low resolution  $48 \times 76 \times 32$  sequence. (b) The  $128 \times 200$  bicubic interpolation of the frame. (c) The  $128 \times 200$  video epitome super-resolution result. A  $125 \times 175 \times 8$  video epitome was learnt from a  $150 \times 200 \times 17$  high resolution sequence captured when the camera physically zoomed into the scene. Super-resolution was performed by reconstructing the low resolution sequence with the epitome containing the features from the high resolution sequence. Patches of size  $75 \times 75 \times 4$ ,  $50 \times 50 \times 3$ , and  $25 \times 25 \times 2$  were used during learning and reconstruction. This video, along with a larger  $360 \times 240 \times 32$  super-resolution result are available at <http://www.psi.toronto.edu/~vincent/videoepitome.html>.

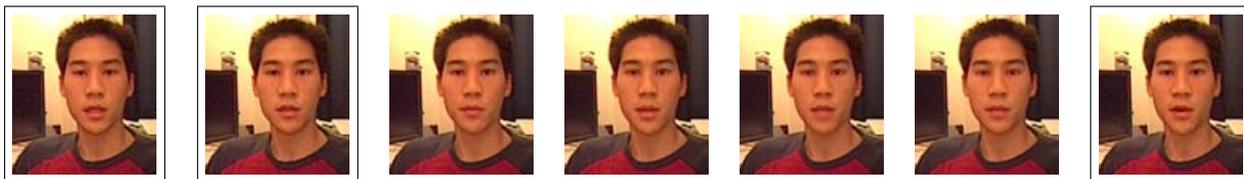


Figure 3. Dropped-frames experiment. This experiment deals with the problem of reconstructing missing frames when receiving streaming video over the Internet. The frames enclosed in boxes have been successfully received, while the other frames were dropped during transmission and have been reconstructed with the use of a video epitome. The arrival time of the frames in this  $90 \times 100 \times 39$  sequence was modeled as a discrete time Bernoulli process whereby the inter-arrival time of the frames was governed by a geometric distribution. Frames were expected to arrive in each time slot and any unfilled time slots between arrived frames were considered to be dropped frames. The mean number of missing frames between arrived frames was one. A  $90 \times 100 \times 13$  video epitome was learnt only from the non-dropped frames using patches of size  $63 \times 70 \times 6$ , and  $36 \times 40 \times 5$ , and  $30 \times 30 \times 4$ . This video sequence is available at <http://www.psi.toronto.edu/~vincent/videoepitome.html>.

frames were then reconstructed using the video epitome as described in Section 4.4. The video epitome is able to consolidate the various disconnected frames and assemble a comprehensible set of motion patterns in the video sequence. In the video on the web page, we show the received frames on the left (with freezes during the frame drops) and on the right, the video reconstruction using only the video on the left as the epitome input. Note that due to the large number of missing frames, most original motion patterns have never been seen by the epitome algorithm in their uncorrupted form anywhere in the received broadcast. Exemplar-based approaches, while able to fill in some missing frames provided that similar motions appear elsewhere in the video sequence, cannot handle this situation.

**Video in-painting.** Video epitomes contain the video's basic structural and motion characteristics, which are useful for in-painting applications. The goal of video in-painting is to fill in missing portions of a video, which can arise with damaged films or occluding objects. Video in-painting can

be performed with the video epitome as described in Section 4.3.

Fig. 4 and its corresponding video on the web show the results of in-painting on a video of a girl walking. The results here are similar to those of [10]. Part way through the video, the girl is occluded by a fire hydrant. Removing the fire hydrant can be formulated as a video reconstruction problem by considering these pixels as missing, learning the video epitome only on the observed pixel values, and performing epitome inference while setting the variances,  $\sigma^2$  to be high in the area occupied by the hydrant. The video epitome is able to compress the basic walking motion into several frames and transfers this motion pattern into the missing pixels.

**Denosing.** In Section 4.3, we discussed the general case of filling in arbitrary missing data in video given only the corrupted video. To illustrate the potential power of the video epitome, a highly corrupted video was created in which each of the RGB color channels of each pixel were

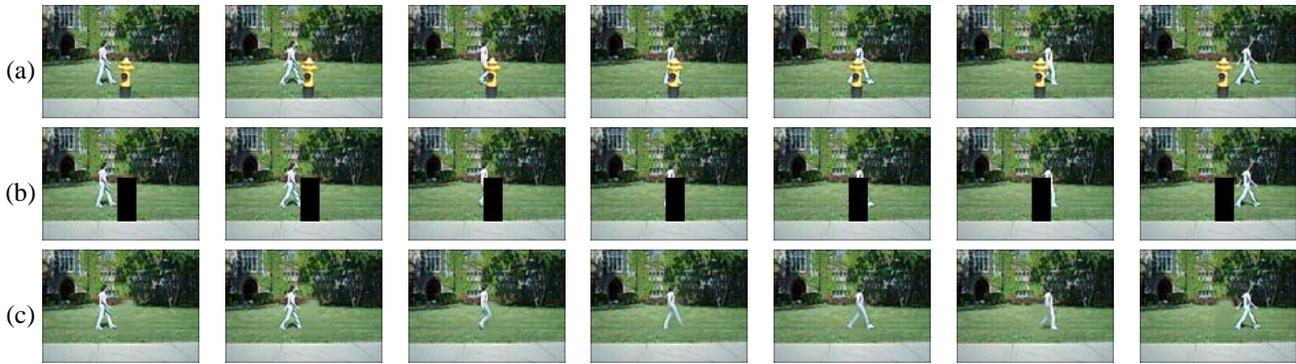


Figure 4. Video in-painting experiment using the video epitome. (a) Several frames from a 114x83x53 video. (b) Removal of the fire hydrant from the video by considering these pixels as missing. (c) In-painted frames using a 30x30x15 epitome with 20x20x5, 15x15x4, 10x10x3, and 5x5x2 patches. Video available at <http://www.psi.toronto.edu/~vincent/videoepitome.html>.

missing with 50% probability. The known bad channels for each pixel were marked by high noise variances  $\sigma^2$ . The video was then given to the epitome learning algorithm and then reconstructed by iterating (7-10). Despite high levels of corruption, the repetitive motion in the video helped the epitome learn a clean representation and reconstruct the video  $\nu$ . This video is available on the web page.

## 7. Conclusion

An epitome represents many of the high-order statistics in array data using a more compact array, and is amenable to faster searching and better generalization, compared to patch libraries. Here, we developed an efficient algorithm for learning video epitomes and performing various inference tasks. We illustrated some of the applications of video epitomes, including video enhancement and editing tasks. Videos demonstrating a variety of applications are available at <http://www.psi.toronto.edu>.

In comparison to reconstruction techniques based on a library of known data patches, an advantage of epitomes is that they can be trained directly on corrupted or degraded data, as long as the data is repetitive. Video tends to be highly redundant and is thus quite well-suited to analysis using epitomes. Because epitomes provide a representation that retains the natural flow of the input data, and offer significant computational and statistical advantages over patch libraries, we believe they will find many uses in data analysis and computer vision.

## Acknowledgements

We thank P. Anandan, A. Blake, M. Irani, A. Kannan, E. Simoncelli, and S.-C. Zhu for helpful discussions. V. Cheung was financially supported by an NSERC Canada Graduate Scholarship and B. Frey was financially supported by a Premier's Research Excellence Award.

## References

- [1] C. Bishop, A. Blake, and B. Marthi. Super-resolution enhancement of video. In *Proc. Artificial Intelligence and Statistics*, 2003.
- [2] A. Criminisi, P. Pérez, and K. Toyama. Object removal by exemplar-based inpainting. In *Proc. Conf. on Computer Vision and Pattern Recognition*, pages 721–728, 2003.
- [3] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proc. SIGGRAPH*, pages 341–346, 2001.
- [4] W. T. Freeman, T. R. Jones, and E. C. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, pages 56–65, 2002.
- [5] A. Jepson and M. Black. Mixture models for optical flow computation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 760–761, 1993.
- [6] N. Jovic, B. J. Frey, and A. Kannan. Epitomic analysis of appearance and shape. In *Proc. IEEE Intern. Conf. on Computer Vision*, 2003.
- [7] R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368, 1998.
- [8] R. Rosales, K. Achan, and B. J. Frey. Unsupervised image translation. In *Proc. IEEE Intern. Conf. on Computer Vision*, 2003.
- [9] J. Y. A. Wang, E. H. Adelson, and U. Y. Desai. Applying mid-level vision techniques for video data compression and manipulation. In *Proc. SPIE on Digital Video Compression on Personal Computers: Algorithms and Technologies*, pages 116–127, 1994.
- [10] Y. Wexler, E. Shechtman, and M. Irani. Space-time video completion. In *Proc. IEEE Comp. Vision and Pattern Recognition*, pages 120–127, 2004.
- [11] S. C. Zhu, C. Guo, Y. N. Wu, and Y. Wang. What are tex-tons? In *Proc. of the 7th European Conf. on Computer Vision-Part IV*, pages 793–807, 2002.