Building Models of Animals from Video

Deva Ramanan, D.A. Forsyth, Kobus Barnard

***Abstract*— This paper argues that tracking, object detection, and model-building are all similar activities. We describe a fully automatic system that builds 2D articulated models known as *pictorial structures* from videos of animals. The learned model can be used to detect the animal in the original video – in this sense, the system can be viewed as generalized tracker (one that is capable of modeling objects while tracking them). The learned model can be matched to a visual library; here, the system can be viewed as a video recognition algorithm. The learned model can also be used to detect the animal in novel images – in this case, the system can be seen as a method for learning models for object recognition. We find that we can significantly improve the pictorial structures by augmenting them with a discriminative texture model learned from a texture library. We develop a novel texture descriptor that outperforms the state-of-the-art for animal textures. We demonstrate the entire system on real video sequences of three different animals. We show that we can automatically track and identify the given animal. We use the learned models to recognize animals from two datasets; images taken by professional photographers from the Corel collection, and assorted images from the web returned by Google. We demonstrate quite good performance on both datasets. Comparing our results with simple baselines, we show that for the Google set, we can detect, localize, and recover part articulations from a collection demonstrably hard for object recognition.**

***Index Terms*— tracking, video analysis, object recognition, texture, shape**

I. Introduction

We argue that tracking, detection, and model building are all similar activities. Suppose we are given a video sequence and told a single unknown deformable animal is present. If we could automatically build a visual model of the animal, we could perform a number of helpful tasks (see Figure 1). We could use the model to find the animal in the original video (so that we can **track** it). By looking up the visual model in a library, we can also **identify** the animal video. Finally, we can use the model to **detect** the animal in other images.

One contribution of this work is the introduction of a system that learns 2D deformable models from video. Deformable models are attractive because they can encode part articulations; for example, it is valuable know where the head, neck, body and legs of a giraffe are. Deformable models have a long-standing history in the vision community beginning with pictorial structures [1] and deformable templates [2], and also include the more recent active appearance models [3] and constellation models [4].

D. Ramanan is with the Toyota Technological Institute, Chicago, IL 60637. D.A. Forsyth is with the Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL 61801. K. Barnard is with the Computer Science Department, University of Arizona, Tuscon, AZ, 85721.

E-mail: ramanan@tti-c.org, daf@cs.uiuc.edu, kobus@cs.arizona.edu.

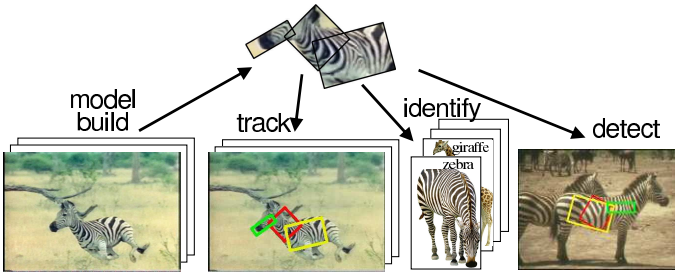

Fig. 1. An overview of this paper. Given a video sequence with a single animal, we cluster candidate segments (Section V) to build a visual model of the animal (Section VI). We then use the model to track the animal in the original video (Section VII), identify the animal (Section IX), and detect the animal in new images (Section X).

II. Related Work: Building Models for Object Detection

Learning articulated models from images is difficult because we must solve the correspondence problem. To build a good shape model for a giraffe, we first need to know where the head and legs are in each image, and so we need good part detectors. But to learn good part detectors, we need a shape model that describes what image regions correspond with each other. This interdependency makes this problem hard.

One option is to ignore the articulation, and just build a "bag of features" model that lacks any explicit spatial component [5, 6]. Such models maybe useful for detection, but appear difficult to apply for localization and articulated pose recovery.

Given that one wants to build models with explicit kinematics, one can solve the correspondence problem by hand-labeling part locations in training images. This has been the most common approach for building articulated models [2, 3, 7–9]. This approach will not scale when building systems designed to recognize hundreds or thousands of objects.

Unsupervised learning of deformable models dates back at least to Weber *et al.* and Fergus *et al.* [10, 11]. The learning was done in an Expectation Maximization (EM) framework, where labels for parts were the hidden variables marginalized out. Since EM is an algorithm susceptible to local maxima, care is required in collecting a set of training images with relatively little clutter in the background.

We demonstrate that video addresses this correspondence problem; motion constraints help determine what image regions move where [12]. We develop an automatic method for building articulated models (pictorial structures, in our case) by searching for possible animal limbs that look consistent over time and that move smoothly from frame to frame. Pawan *et al.* [13] describe an algorithm for refining models built from video using EM. Such an approach is sensitive to initialization, and the system described here can be seen as a method for

initialization.

However, a pictorial structure learned from a particular video is overly tuned to the specific animal captured in the video. We want to use the model to find all instances of that animal (from other images and videos). To do this, we augment the model with a *discriminative* texture model built from a texture library. Our model is capable of recognizing giraffe textures (which are notoriously difficult because they have structure at two spatial scales — see [5, 6] and Fig.15). We compare our model with various texture descriptors in Sec.VIII and show that it outperforms the current state-of-the-art for animal recognition. Texture descriptors that assume a known segmentation (or that an image consists of a single texture) tend to perform poorly on real images; our model is trained and tested on natural textures observed in images of real scenes.

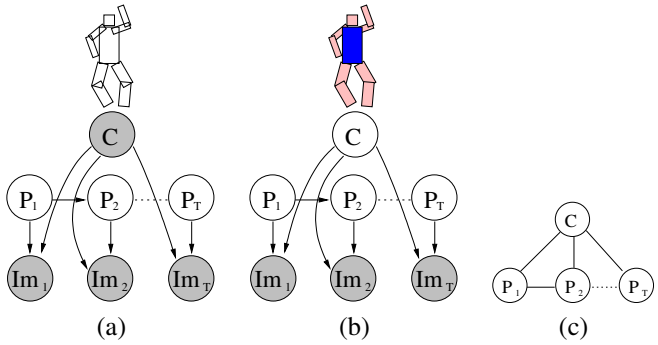

Fig. 2. In (**a**), we display the graphical model corresponding to model based tracking. Object position P_t follows a Markovian model, while image likelihoods are defined by a model template C. Since C is given *a priori*, the template must be invariant to clothing; a common approach is to use an edge template. By treating C as a random variable (**b**), we *build* a template specific to the particular person in a video as we track his/her position P_t. We show an undirected model in (**c**) that is equivalent to (**b**).

III. Related Work: Tracking

It will be useful to describe our algorithm in terms of an articulated tracker. Most of the relevant work has focused on human kinematic tracking. Space does not allow a full review, so we refer the reader to [14]. Most tracking approaches can be loosely cast in the framework of hidden Markov models (HMM), where the object pose (X_t) is the hidden variable to be estimated, and the observations are images (Im_t) of a video sequence. Standard Markov assumptions allow us to decompose the joint into

$$\Pr(X_{1:T}, Im_{1:T}) = \prod_t \Pr(X_t|X_{t-1}) \Pr(Im_t|X_t),$$

where we use the shorthand $X_{1:T} = \{X_1, \ldots, X_T\}$. Tracking corresponds to inference on this probability model; typically one searches for the maximum a posteriori (MAP) sequence of poses given an image sequence.

We will assume the dynamic model $\Pr(X_t|X_{t-1})$ is weak because articulated objects (such as animals and people) can move in unpredictable ways. The likelihood model $Pr(Im_t|X_t)$ usually involves evaluating (a possibly deforming) template at different image locations. Most of the literature has focused on mechanisms of inference, which are typically variants of dynamic programming [8, 15, 16], kalman filtering [17–19] or particle filtering [20–22]. One uses the pose in the current frame and a dynamic model to predict the next pose; these predictions are then refined using image data. In particular, particle filtering uses multiple predictions — obtained by running samples of the prior through a model of the dynamics — which are reweighted by comparing them with the local image data (the likelihood).

A pre-process (such as background subtraction) oftentimes identifies the image regions of interest, and those are the observations fed into a probabilistic model. In the radar tracking literature, this pre-process is known as data-association [23]. We argue that the dominant difficulty in making any video tracking algorithm succeed lies in data-association – identifying those image regions that consist of the object to be tracked.

Particle filters use the dynamic model $\Pr(X_t|X_{t-1})$ to perform data association; (multiple) motion predictions tell us where to look in an image. This approach, though computationally efficient, is susceptible to drift when tracking with a weak dynamic model in a cluttered background. The alternative is to do data association using the likelihood model $Pr(Im_t|X_t)$, but this requires a template that produces a low likelihood for background regions. Indirect methods of doing this include background subtraction, but again, this is only valid in restricted situations. Alternatively, one could build a template that directly identifies images regions of interest.

Most template-based trackers use an edge template, using such metrics as chamfer distance or correlation to evaluate the likelihood for a given pose. For simplicity, let us assume object state consists solely of pose P_t, the template is represented by an image patch C, and the likelihood is measured by SSD (though we will look at alternate encodings of appearance). In this case, we can write the likelihood as:

$$\Pr(Im_t|P_t, C) \propto \exp^{-||Im_t(P_t)-C)||^2} \quad (1)$$

where we have ignored constants for simplicity. If we condition on C (assume the appearance template is given), our model reduces to a standard HMM (see Figure 2). Algorithms that track people by template matching follow this approach [8, 22, 24–26]. Since these templates are built a priori, they are detuned because they must generalize across all possible people wearing all possible clothes. Such templates must necessarily be based on generic features (such as edges) and so are easily confused by background clutter. This makes them poor at data association.

By treating the template C as an unknown quantity, we want to *build* a template tuned to a specific object in a video. A template that encodes the red color of a person's shirt *can* perform data association since it can quickly ignore those pixels which are not red. Under this model, the focus on tracking becomes not so much identifying where an object is, but learning what it looks like.

This view of *tracking as model-building* dates back at least to the layered sprite model of Jojic and Frey [27], and the morphable models of Brand [28] and Torresani *et al.* [29]. These approaches also build a model of an object

while simultaneously tracking it. The work of [27] does not deal with articulated objects, although extensions are proposed in [13]. Morphable models can track deforming objects in 3D, but current implementations require manual initialization and sequences where point features can be reliably tracked.

IV. Our approach

To construct an algorithm that builds models from videos, we look to the object detection community for inspiration. Many authors have developed algorithms that build object models from image collections [5, 6, 11, 30, 31]. Say we want to use one of these algorithms to learn a model of a zebra. We assemble a set of positive example images containing zebras, and a set of negative example images not containing zebras. This form of input is often called *semi-supervised* data because we are labeling which images contain a zebra, but for a given zebra image, we do not label which image regions are zebra and which are background. The task of the learning algorithm is to "finish" the partial labeling; learn a zebra model that labels zebra image regions. Most algorithms do this by variants of clustering or EM; basically one looks for image regions that consistently appear in the positive set, but not in the negative set. Presumably the algorithm will find striped image patches in the positive set, and so learn a corresponding zebra texture model.

We can apply the same learning algorithms to frames from a video sequence of a zebra. We treat the frames as images from the positive set. Unfortunately, we do not have a negative set with which to compare. But we have an alternate source of information; smoothness of motion. We know that a zebra region must appear consistently in most frames of a zebra video, *and* that those zebra regions must move smoothly from one frame to the next. In essence, we can use the temporal coherence in a video sequence to provide supervisory signals.

Assume we are given a video sequence of single animal. This paper presents an algorithm that automatically builds a visual model of the animal. Section V describes a clustering method that constructs rough spatio-temporal tracks of body segments over a sequence. In Section VI, we use the tracks to learn a pictorial structure [1, 7].

Once we learn the model, there are several neat applications. We use it to find the animal in the original video (so that we can **track** it better; Section VII). By looking up the visual model in a library, we can also **identify** the animal (Section IX). Finally, we can use the model to **detect** the animal in other images (Section X).

We significantly improve the quality of the visual model by augmenting it with a animal texture model learned from a library of textures. Examining various texture descriptors, we find they do not characterize animal textures well. We develop a novel texture representation in Section VIII that outperforms the state-of-the-art.

V. Building a spatio-temporal track

Say we are given a video with a single animal, and we want to build a spatio-temporal track of how its body parts deform over time. If we assume the animal is made up of body segments, we can:

1) *Detect* candidate segments with a detuned segment detector
2) *Cluster* the resulting segments to identify body segments that look similar across time
3) *Prune* segments that move too fast in some frames.

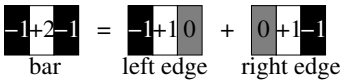

Fig. 3. One can create a rectangle detector by convolving an image with a bar template and keeping locally maximal responses. A standard bar template can be written as the summation of a left and right edge template. The resulting detector suffers from many false positives, since *either* a strong left or right edge will trigger a detection. A better strategy is to require *both* edges to be strong; such a response can be created by computing the minimum of the edge responses as opposed to the summation.

A. Detecting Segments

We model segments as cylinders that project to rectangles in an image. One might construct a rectangle detector using a Haar-like template of a light bar flanked by a dark background (Figure 3). To ensure a zero DC response, one would weight values in white by 2 and values in black by -1. To use the template as a detector, one convolves it with an image and defines locally maximal responses above a threshold as detections. This convolution can be performed efficiently using integral images [32]. We observe that a bar template can be decomposed into a left and right edge template $f_{bar} = f_{left} + f_{right}$. By the linearity of convolution (denoted *), we can write the response as

$$im * f_{bar} = im * f_{left} + im * f_{right}.$$

In practice, using this template results in many false positives since either a single left or right edge triggers the detector. We found taking a *minimum* of a left and right edge detector resulted in response function that (when non-maximum suppressed) produced more reliable detections

$$\min(im * f_{left}, im * f_{right})$$

With judicious bookkeeping, we can use the same edge templates to find dark bars on light backgrounds. We explicitly searched over 15 template orientations and 25 scales (5 lengths crossed with 5 widths).

It turns out to be hard to build accurate low-level segment detectors. Figure 4-(a) shows three frames from a video of a zebra in which the detectors often fire on the animal body, but also fire on clutter in the background. We would like to pick out the true animal body parts from the set of candidate detections. Unfortunately, we do not know what the animal segments should look like (since we are not told a zebra is present). But we know that animal segments should be *consistent* in appearance over time; if the head is striped in the first frame, it should be striped in the final frame. We find collections of segments that look similar to each other by *clustering* the entire set of detected segments.

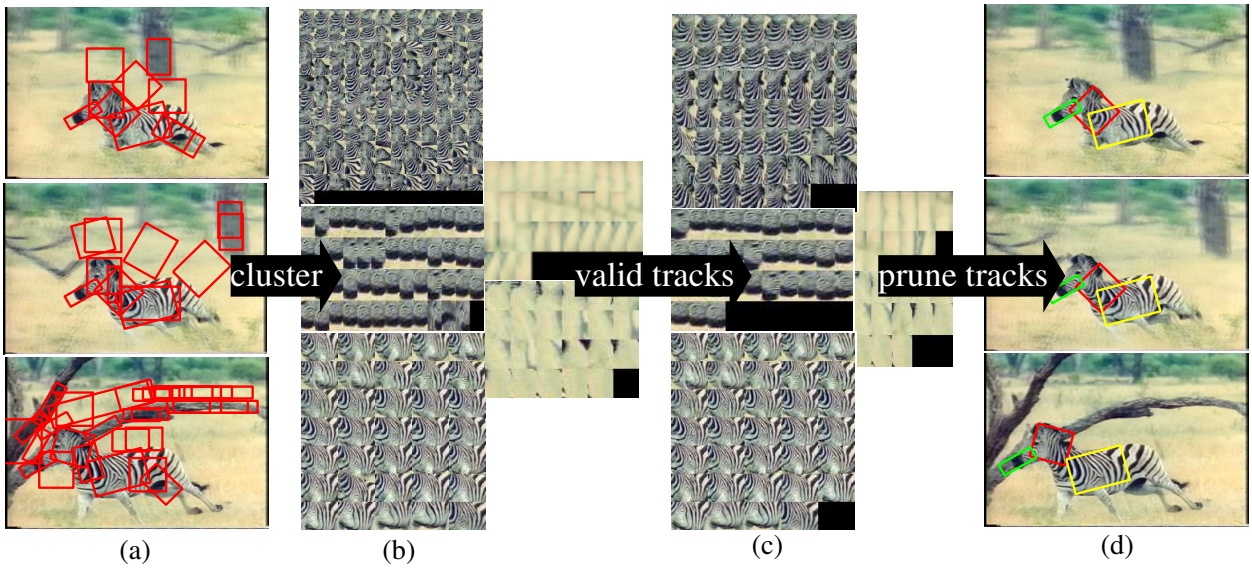

Fig. 4. Obtaining spatio-temporal tracks by clustering. We first search for candidate segments using local detectors (we show 3 sample frames in (a)). We cluster the image patches together in (b). From each cluster we extract a valid sequence obeying our motion model in (c). We prune away the short sequences to retain the final spatio-temporal tracks in (d).

B. Clustering Segments

Since we do not know the number of segments in our model (or for that matter, the number of segment-like things in the background), we do not know the number of clusters *a priori*. Hence, clustering segments with parametric methods like Gaussian mixture models or k-means is difficult. We opted for the mean shift procedure [33], a non-parametric density estimation technique.

We create a feature vector for each candidate segment, consisting of a normalized color histogram in the Lab color space, appended with shape information (in our case, simply the length and width of the candidate patch). Note that this feature vector is to be used for *clustering*, for which it is sufficient. The representation of appearance is not limited to this feature vector.

The color histogram is represented with projections onto the L, a, and b axis, using 10 bins for each projection. Hence our feature vector is $10 + 10 + 10 + 2 = 32$ dimensional. We scale the histogram and scale dimensions so as to obtain a meaningful $\mathcal{L}_2$ distance for this space. Further cues — for example, image texture — might be added by extending the feature vector, but appear unnecessary for clustering.

Identifying segments with a coherent appearance across time involves finding points in this feature space that are (a) close and (b) from different frames. This is difficult to do; we drop requirement (b), which can be imposed on clusters *post hoc*, and concentrate on (a). The mean shift procedure is an iterative scheme where we find the mean position of all feature points within a hypersphere of radius h, recenter the hypersphere around the new mean, and repeat until convergence. We initialize this procedure at each original feature point, and regard the resulting points of convergence as cluster centers. For example, for the zebra sequence in Figure 4, starting from each original segment patch yields five points of convergence (denoted by the centers of the five clusters in (b)).

Sometimes, illumination changes will force a single animal part to appear in two or more clusters. As a post-processing step we greedily merge clusters which contain members within h of each other (starting with the two closest clusters). We account for over-merging of clusters by extracting multiple valid sequences from each cluster during step (c) (for each cluster during the third step in Figure 4, explained further in the following section, we keep extracting sequences of sufficient length until none are left). Hence for a single arm appearance cluster, we might discover two valid tracks of a left and right arm.

C. Enforcing a motion model

As Figure 4 indicates, not every coherent patch is associated with a moving figure. The second column of clusters in 4-(b) are background regions. However, at this point cluster elements are neither constrained to move with bounded velocity nor required to form a sequence — there might be several elements from the same frame.

We now find the most likely sequence of candidates for each cluster that obeys the velocity constraints. By fitting an appearance model to each cluster (typically a Gaussian, with mean at the cluster mean and standard deviation computed from the cluster), we can formulate this optimization as a straightforward dynamic programming problem. Let P_t be the position of a segment in the t^{th} frame. We assume these have a Markovian behavior; i.e. $Pr(P_t|P_{1:t-1}) = Pr(P_t|P_{t-1})$. The reward for a given candidate is its likelihood under the Gaussian appearance model, and the temporal rewards are '0' for links violating our velocity bounds and '1' otherwise. We add a dummy candidate to each frame to represent a "no match" state with a fixed charge. By applying dynamic programming, we obtain a sequence of segments, at most one per frame, where the segments are within a fixed velocity

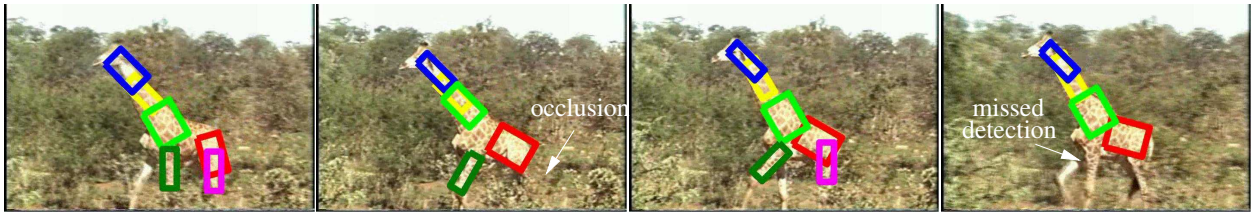

Fig. 5. We show spatio-temporal tracks obtained from a video of a giraffe. The number of parts and the spatio-temporal tracks were obtained automatically using the algorithm from Figure 4. Note that because we allow "no-match" states in our tracks, we can recover from occlusions, but also suffer from missed detections. We later correct the missed detections by re-tracking using a model learned from the spatio-temporal tracks (Figure 13).

bound of one another and where *all* lie close to the cluster center in appearance.

As Figure 4-(c) demonstrates, this results in a somewhat smaller set of segments associated with each cluster, particularly the second column of background clusters. The camera in our zebra video is panning with the animal, so the background is constantly changing. Since there is no single background patch that stays in view for the duration of sequence, the background clusters in Figure 4-(b) are made of patches from different parts of the background. Temporal links between disparate patches violate our velocity model. Hence, the temporal smoothness enforced by the dynamic programming tends to reduce the size of the background clusters.

We now discard those tracks which are not long enough. In Figure 4-(c), this prunes away the second two clusters. Note we could impose other tests of validity beyond the length of a track; we might require that a segment move at some point, and so we would prune away a track which is completely still. Alternatively, if we are given two different videos of the same animal, we might prune away those clusters which do not appear in both.

The segments belonging to the remaining three clusters are shown in Figure 4-(d). Our algorithm "discovers" the number of parts (three, in this case) automatically. The initial number of clusters is given by the mean shift algorithm, while the subsequent dynamic progamming and pruning throws away the bad clusters. Each of the remaining clusters (which we interpret as a body part) tends to look coherent over time, moves smoothly over time, and is consistently detected in many frames. For example, our algorithm automatically learns a six-part model for a giraffe (Figure 5). We now can learn a visual model from the spatio-temporal tracks (Section VI). But first, it is useful to cast our clustering procedure in light of our constant appearance model developed in Section III.

D. Approximate Inference

The segment finding procedure for the discussed above is, in fact, an approximate inference procedure for the graphical model shown in Figure 6. Recall our original blob model from Figure 2 (shown again in Figure 6-(a)); this model captured the fact that we want to track a segment while building a model of its appearance. The algorithm described in this section is a loopy inference procedure for our blob model (see also [34–36]). We pass max product messages asynchronously, and visualize our message schedule with the embedded trees shown in Figure 6.

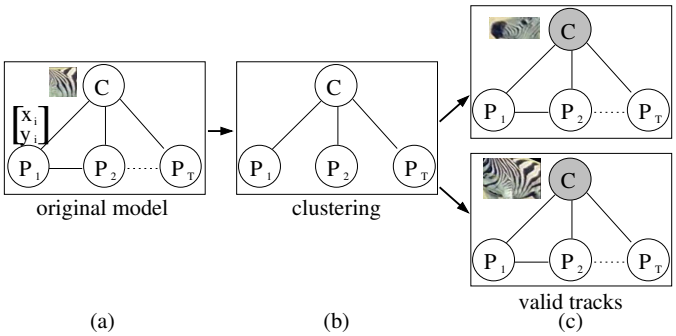

Fig. 6. Approximate inference on our blob tracking model from Figure 2. The original model (**a**) encodes the fact that we want to track a segment while simultaneously building a model of its appearance. An alternative interpretation is that we want to cluster, or learn a coherent appearance, while simultaneously enforcing that all the patches from a cluster obey a motion model. We can do the latter (approximately) by dropping the motion constraint. We naively cluster, looking for collections of coherent segments (**b**). In this case, we find multiple coherent appearances (corresponding to the zebra head and body). We instantiate the model multiple times, for each cluster. Given the learned appearance, we do dynamic programming to extract a sequence of valid tracks where all the segments look similar to the learned model (**c**).

In the first subtree, we want to infer a posterior over C given the image patches from a sequence. We show in [14] that the mean shift clustering procedure finds modes in the posterior of C. We interpret each mode, or cluster, as a *unique* segment. We instantiate multiple copies of the model Figure 6-(b), one for each cluster. We can partly justify this procedure by our aggressive post-clustering merging of clusters; any left-over clusters which remain separate are likely to be different segments, and not multiple appearance modes of a single segment.

We now can treat C as an observed quantity, for each instantiation of Figure 6-(c). Inferring P_t from such a model is straightforward; this is just our dynamic programming solution to find the most likely sequence of candidates given a known appearance. Note our initial claim of segment configurations P_t being Markovian is only true when we condition on C. Finally, we disregard those instantiations we deem invalid (i.e., not existing for enough frames).

VI. Learning a Pictorial Structure

We use the spatio-temporal tracks (obtained by clustering) to build a visual model called a *pictorial structure*. A pictorial structure model is a parts-based model of an object consisting of two terms; a geometric term that relates the spatial arrange-

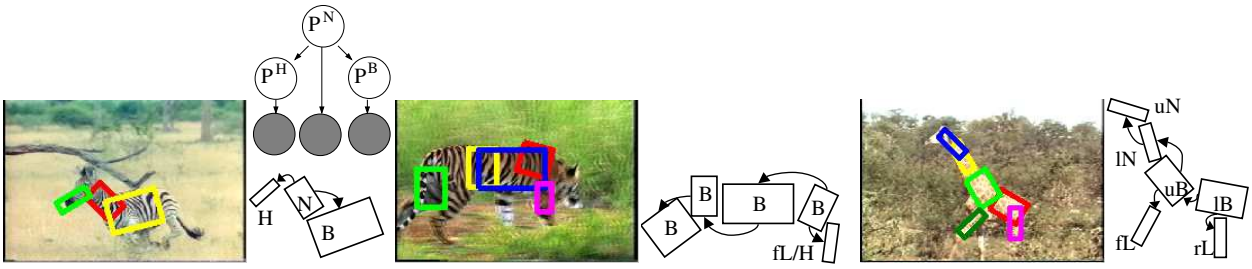

Fig. 7. We show the pictorial structures learned from videos of a zebra **left**, tiger **center**, and giraffe **right**. We visualize the zebra pictorial structure as a graphical model (**above on left**). This model is parameterized by probability distributions capturing geometric arrangement of parts $\Pr(P^i|P^j)$ and local part appearances $\Pr(Im(P^i)|P^i)$ (the vertical arrows into the shaded nodes). These distributions and the tree structure of the graph are automatically learned from the video. We manually attach a semantic description to each limb as Head, upper/lower Neck, upper/lower Body, or front/rear Leg. Labeling the tiger model is tricky; many limbs swim around the animal Body, and one flips between the Head and front Leg. We use these labels to help evaluate localization performance in our results; they are *not* part of the shape model. Obtaining a set of canonical labels appears difficult.

ment of parts, and an appearance term that describes the local appearance of each part [1, 7, 8].

$$\Pr(P^1 \dots P^N|Im) \propto \prod_{(i,j)\in E} \Pr(P^i|P^j) \prod_{i=1}^{N} \Pr(Im(P^i)|P^i). \quad (2)$$

$\Pr(P^i|P^j)$ are geometric terms that capture the spatial arrangement of part i with respect to part j, and $\Pr(Im(P^i)|P^i)$ captures the local appearance of the image at part i. Here, we extend part configuration P^i to include both position (x, y) and orientation θ. The position of each non-root segment P^i is represented with respect to the coordinate system of its parent P^j. E is a set of edges that capture the dependency structure of the model. A model is fully specified when the edge structure and the probability distributions along each edge are known. If E is a tree, one can efficiently match these models to an image using Dynamic Programming (DP). Felzenszwalb and Huttenlocher [7] describe efficient DP-based techniques for computing the MAP estimate and for sampling from the posterior in Equation 2. One can also use the (unnormalized) posterior as an animal detector by only accepting those maximal configurations above a threshold.

Learning by maximum likelihood: Standard methods learn pictorial structures by maximum likelihood estimation (MLE) given images where parts are labeled [7, 37]. Theoretically, one could learn pictorial structures from unlabeled data using EM, where labels are hidden variables marginalized over. This approach is taken in [11], where the learned models are called *constellation models*. To avoid local maxima issues with EM, those models must be learned from uncluttered images and with finely tuned part detectors. In our case, we learn pictorial structures by direct MLE without any manual intervention; we use the coherence in a video to provide the labeling. Note we do not need precise labels, but rather correspondence between parts over time – this is provided by the cluster membership.

Learning E: Typical methods for learning the spatial structure E will not work in our case; we briefly describe the approach from [7, 8, 12] here. Consider a fully connected bi-directional graph where each vertex represents a segment P^h, P^n, and P^b (the precise segment labels are not needed so long as their correspondence between frames is known). Directed edges in this graph are weighted by the entropy of $\Pr(P^i|P^j)$.

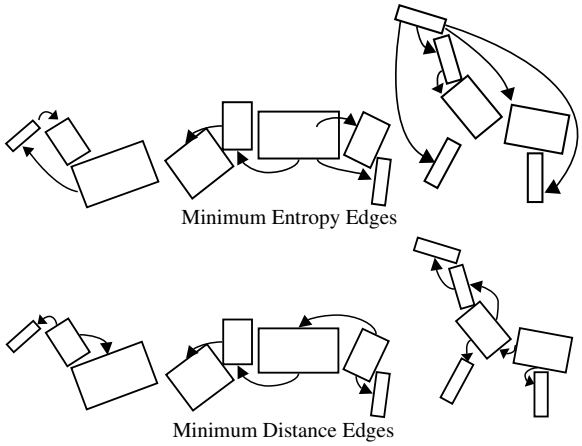

Fig. 8. On the **top**, we show the edge structure E learned by a minimum entropy spanning tree (this are the edges that maximize the likelihood of the observed spatio-temporal tracks). This can create links between two far away segments (the giraffe's head and leg) because of noisy entropy estimates. A better strategy is to link the parts that tend to lie near each other (**bottom**).

To learn the tree structure that maximizes the likelihood of the observations, [7, 8, 12] finds the minimum-entropy spanning tree. This tends to result in poor models. Often two far away limbs will be directly linked in the learned spatial model. This is because the position P^i of the detected limbs are quite noisy (due to the detuned limb detector), in turn producing noisy entropy estimates (see Figure 8). To enforce the natural prior that two limbs that tend to appear near each other should be spatially linked together, we replace the entropy term with the mean distance between those two limbs (and then compute the minimum spanning tree). We root this tree at the most "stable" limb (the limb detected most often in the original video). This produces the tree spatial models in Figure 8.

$\mathbf{Pr(P^i|P^j)}$: We fit the geometric terms $\Pr(P^i|P^j)$ by standard Gaussian MLE assuming diagonal covariance matrices. For example, assume one has a collection of zebra images where *H*ead, *N*eck, and *B*ody segments are labeled. The sample mean and standard deviation of head positions relative to the neck would define the MLE estimate of $\Pr(P^H|P^N)$. In our case, we use the cluster membership to provide the labels. Because we are using gaussian potentials to represent articulated joints, we expect the variance in the (x, y) dimension to be

small (since body parts must stay connected) and the variance in θ to be large (since body parts can rotate to a large degree).

$\mathbf{Pr(Im(P^i))}$: One could also fit the appearance terms $\Pr(Im(P^i))$ by MLE; however, we found this yields a poor detector since we have ignored the background. We learn a **discriminative** part appearance model by learning a animal texture classifier from the video. We use the spatio-temporal tracks to segment the video into animal (foreground) and background pixels. We fit a 5-component Gaussian mixture model in RGB space for the foreground/background, as in [38].[1] We use our pictorial structure to find an animal in a image using the procedure in Figure 10. Given an image, we first use our texture model to label the animal pixels. We evaluate the part likelihood $\Pr(Im(P^i))$ by convolving the label mask with rectangle templates looking for light bars on dark backgrounds (as in Section V-A); we want parts to lie on animal pixels and not the background. An alternative would have been to learn a separate texture model for each animal limb (this is done for people in [39]); we found this was not necessary for animals with homogeneous texture.

Note our RGB-based texture classifier is specific to the video it is trained on. Hence they are limited in their use; we can use them to find animals in the original videos (so that we can track them, as in Section VII). However, if we want to find animals in new images, we need to build more sophisticated texture models (Section VIII).

VII. Tracking by finding

Given the learned pictorial structure, we can use it to track the animal in the original video. For each frame, we find the best-matching body pose (by dynamic programming), or obtain a distribution over body poses (by sampling); see Figure 10. Because we are tracking by detection, our tracker is quite robust. It can recover from errors and occlusion because it automatically re-initializes itself in every frame. We show results for three sequences depicting different moving animals in Figures 11, 12, and 13.[2] The tracks were not hand initialized, and the same program was used in each case. The program automatically learns a pictorial structure and then identifies instances in each frame. In each sequence, the animal's body deforms considerably, the zebra because it is moving very fast, the giraffe and the tiger because giraffes and tigers deform a lot when they move. Nonetheless, the program is able to build an appearance model that is clearly sufficient to capture the essence of the moving animal, but lacks some details. In particular, legs are narrow, fast, and hard to detect; consequently, the learned pictorial structures fails to model them. Furthermore, the temporal correspondences for the segments — which are indicated by colored outlines in the figures — are largely correct. Finally, the tracker has been able to identify the main pool of pixels corresponding to the animal in each frame.

Following [40], we evaluate our tracker using detection rates (Figure 9) obtained from the original video. Our algorithm builds a representation of each animal as a collection of parts. We define a correct localization to occur when the majority of pixels covered by an estimated part have the correct semantic label from Figure 7. The quality of the track using the pictorial structure is quite good, and is much better than the original spatio-temporal track. This suggests that tracking is easier with a better model. One can envision iterating this procedure (in a manner quite similar to EM) by refitting a pictorial structure model to the newly tracked segments, and then tracking given the pictorial structure model. We performed only one iteration.

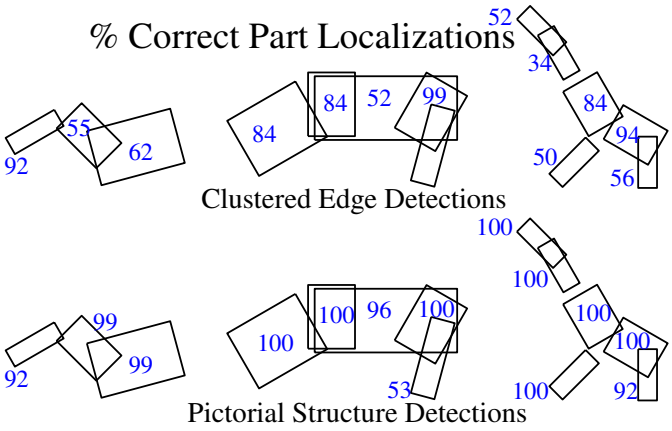

Fig. 9. We evaluate our trackers using detection rates for the original videos. Our algorithm builds representations of animals as a collection of parts. We overlay the percentage of frames where parts are correctly localized. We define a part to be correctly localized when it overlaps a pixel region with the correct semantic label from Figure 7. On the **top**, we show results from the original spatio temporal tracks obtained by clustering edge detections. After learning a pictorial structure from the tracks, we use the model to re-detect the animal. This results in significantly better performance, as shown on the **bottom**.

Our tracker is successful largely because of the quality of the foreground masks produced by the learned animal classifiers (Figure 10). These classifiers do not generalize well to novel images (with say, different illumination conditions); we build robust animal texture classifiers in Section VIII.

VIII. Building a texture model

To both identify a pictorial structure (Section IX) and detect it in new images (Section X), we need a good animal texture descriptor. We want a descriptor capable of producing foreground masks like those of Figure 10, but for novel images. Specifically, it must be able to **segment** out an animal from cluttered backgrounds (typically of foliage). Descriptors developed for standard vision datasets (such as CUReT [41]) may not be appropriate for segmentation since they classify entire images of homogeneous texture. Giraffe textures in particular are notorious for being difficult to capture (e.g. [5, 6]; see Figure 15).

A. Texture library

To evaluate descriptors on small image patches, we create a texture library. We use the Hemera Photo-Object[42] database of image clip art; these annotated images have associated foreground masks. We use all the images in the "animals" category, throwing away those animals with less than 3 example images.

[1] Technically speaking, we learn generative models for the foreground and background. They are "discriminative" in the sense they are used to classify pixels.

[2] Videos are available online at the first author's webpage.

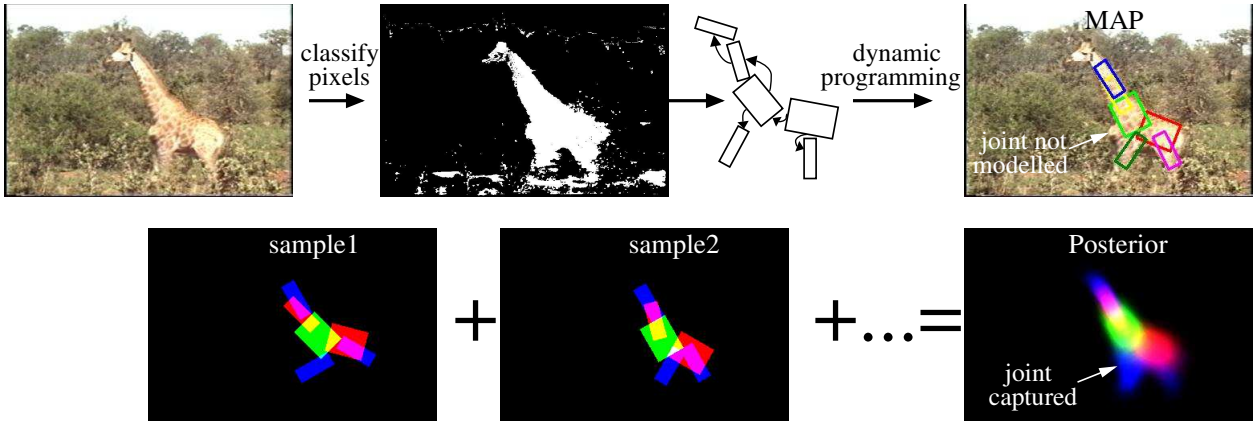

Fig. 10. Detecting a pictorial structure $\Pr(P^{head}, P^{neck}, \ldots | Im)$. Given the image on the **left**, we first classify foreground pixels using a color model learned from the spatio-temporal tracks. We use dynamic programming to find an arrangement of limbs that lie on foreground pixels and that look like the shape prior; this yields the MAP estimate on the **right**. We also can use a foreground mask and shape prior to generate sample body poses from the posterior $\Pr(P^{head}, P^{neck}, \ldots | Im)$ (using the method of [7]). We superimpose the samples to yield the final posterior map on the **bottom**. The posterior models the front leg joint better than the MAP estimate; this suggests we can use uncertainty in how we match a model to compensate for its inadequacies.

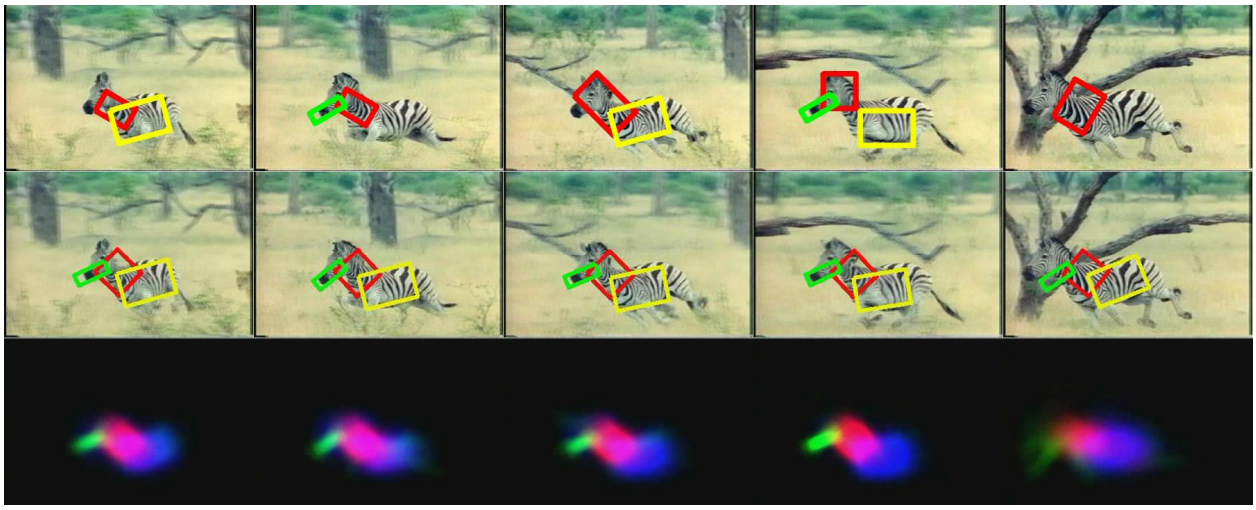

Fig. 11. Tracking results for the zebra video. In the **top** row, we show the spatio-temporal tracks obtained by clustering together segments that obeyed our motion model. Correspondence over time (denoted by the colors) are given by cluster membership. Given those segments, we learn the zebra pictorial structure model shown in Figure 7. Given the learned model, we can re-track by computing MAP estimates for each frame in the video (**middle**). We can also visualize the entire posterior using the sampling method from Figure 10 (**bottom**). Note that the tracks tend to get significantly better as we build an improved visual model of the zebra; we quantify this in Figure 9.

This leaves us with about 500 images spanning 38 animals. We assemble a texture library by randomly sampling 1500 17X17 patches from each animal class (Figure 14).

B. Descriptor Evaluation

We use the library to compare 3 different patch descriptors; histograms of textons [43], intensity-normalized patch pixel values [44], and the SIFT descriptor [45]. **Textons** are quantized filter bank outputs that capture small scale phenomena (such as t-junctions, corners, bars, etc.). They are typically binned into a histogram over some spatial neighborhood. The SIFT patch descriptor is a 128 dimensional descriptor of gradients binned together according to their orientation and location; it is designed to be robust to small changes in pixel intensity and position. Note we use the raw descriptor on a 17X17 patch, without normalizing for scale or dominant orientation (as in[45]).

We evaluate our texture model by 3-fold cross-validation. We tried a variety of classifiers, such as K-way logistic regression, SVMs, and K-Nearest Neighbors (NN). This classification problem is difficult because of the large number of classes (almost 40); training an all-pairs SVM classifier took exorbitantly long. When training a SVM on 2 animal classes, we did not observe any sparsity. This suggests that the decision boundary is curvy (and so we need all sample points as support vectors). K-NN performed the best, with K = 1. Hence we evaluate patch descriptors using 1-NN classification (again using 3-fold cross-validation) in Table I.

There are three conclusions we can draw: (1) Classifying small patches seems much harder than classifying entire images of homogeneous texture. Our results are worse than those reported for texture databases like CUReT [6, 43, 44]. (2) There is a large variance in performance depending on the animal class. Discriminating between elephants and rhinoceros

Fig. 12. Tracking results for the tiger video, using the same conventions as Figure 11. Note the tracks tend to get significantly better as we build an improved visual model (we quantify this in Figure 9).

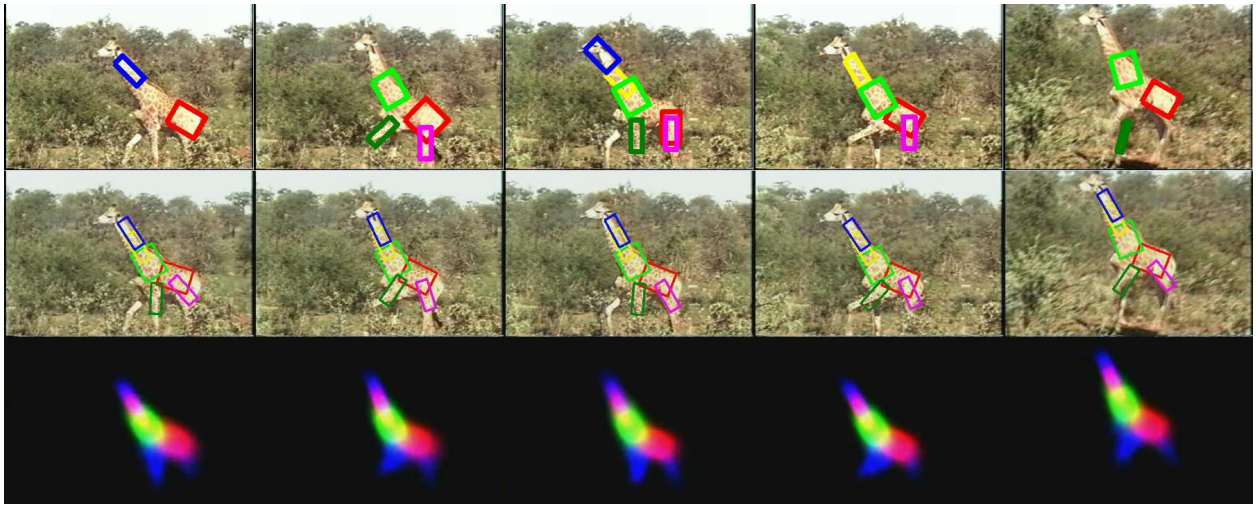

Fig. 13. Tracking results for the giraffe video, using the same conventions as Figure 11. Note the tracks tend to get significantly better as we build an improved visual model (we quantify this in Figure 9).

is hard because of their similar hides, but highly textured animals such as zebras, tigers, and giraffes stand out. Finally, (3) SIFT seems much better suited for detecting animal textures from small patches. We look at the ability of SIFT to segment out animals from real images in Section VIII-C.

C. Why are giraffes hard?

Segmenting giraffes present particular difficulties for texton based descriptions (e.g. [5, 6]; Figure 15). The texture is characterized by phenomena at two scales (long thin stripes that lie in between big blobs). If we calculate textons over a large scale, we miss the thin stripes. If we calculate textons on a small scale, the long scale spatial structure of the textons defines the big blobs (Figure 15). This spatial structure is lost when we construct a histogram of local neighborhoods from the texton map. This suggests that we should not think of a giraffe texture as an unordered collection of textons, but rather simply a patch, or a collection of spatially ordered pixels. A robust patch descriptor such as SIFT is a natural choice (other descriptors such as [46] may also prove useful).

Our results are surprising because SIFT was not designed to represent texture (as noted in [45]); however we find it can represent texture given we store enough examples. The drawback to our nearest neighbor approach is time required to classify a new patch; we must compare it against 1500 prototypes from 38 classes. Obtaining a simpler parametric representation of animal texture remains future work.

We now can use our texture models to identify the animal in a video (Section IX) and detect the animal in new images (Section X).

IX. Identifying the animal

We use our patch-based texture model from Section VIII to identify the animal in a video. We assume that the animal in a given video is one of the 38 animals in Hemera. We scale the Hemera images and video clips to be similar sizes,

Fig. 14. Our library of animal textures built from Hemera. We show a subset of 100 17X17 patches for each of our 38 animals; we mark the giraffe, tiger, and zebra rows in yellow. Our recognition task requires texture classification *and segmentation* (we need to separate the animal from its background). This means we need to evaluate textures on a local image patch. We use this library to evaluate patch descriptors in Table I.

Comparing texture descriptors for detecting animal patches

Descriptor	All	Zebras	Tigers	Giraffe
Patches	8.2	8.93	5.56	5.97
Textons	11.1	31.3	12.7	12.5
SIFT	13.6	40.0	19.1	21.9

TABLE I. We count how often we can correctly identify an animal based on texture from a single patch from Figure 14. We report percentage of correct detections in cross validation experiments for a 1-NN classifier using 1500 prototypes per class. For the full (38 class) multi-class problem 'All', we perform quite poorly. Many animal classes (such as elephants and rhinoceroses) are hard to discriminate using texture alone. When scoring correct detections solely on zebra, tiger, and giraffe test patches, we do much better, indicating those animals have distinctive texture. Looking at various patch representations (normalized patch pixel values, histograms of textons, and a SIFT descriptor), we find SIFT performs the best. We adopt it as our texture descriptor, and examine its behavior further in Figure 15.

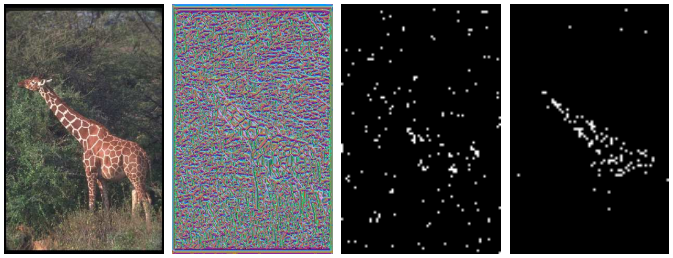

Fig. 15. Given a query image **left**, we replace each 17X17 patch with its closest match from a patch in our texture library. This means we need a good animal texture descriptor; one that captures the long thin stripes that lie within big blobs typical in a giraffe. Standard approaches use histograms of textons (quantized filter bank outputs) [5, 6, 43, 44]. We show a texton map on the **middle left**, where each color maps to an individual texton. The big blobs that distinguish the giraffe from the background are only apparent from the *long-scale spatial arrangement* of textons. Looking at histograms of textons over small neighborhoods looses this spatial arrangement. Hence classifying giraffe patches based on texton histograms is a poor approach, as seen in the **middle right** (and as acknowledged by [5, 6]). Rather, if we classify patches using a descriptor capturing spatial arrangement of pixels (e.g. SIFT), we are better at detecting giraffe patches (**right**).

and assume the animals are present similar scales. We use a two-part matching procedure, shown in Figure 16.

Texture cue: We match the texture models built from *Hemera to the video*. We use the animal tracks (Section VII) to segment the video into animal/non-animal pixels. We extract the set of all 17X17 animal patches from the video, and classify each as one of the 38 animals. We do 1-NN classification on each patch, finding the closest match from our library of animal textures (by matching SIFT descriptors). We can obtain a texture posterior for animal labels given a video by counting the number of times the classifier voted for the i^{th} animal class. Looking at Figure 17, we see that matching solely based on texture is not enough to get the correct animal label; the giraffe video matches best with a 'leopard' texture.

Shape cue: We add a shape cue by matching the shape model built from the *video to Hemera.* For each image in the Hemera collection, we use dynamic programming to find a configuration of limbs that occupies the foreground mask *and* that is arranged according to the shape prior learned from the video. We show 4 matches for our giraffe shape model in Figure 18; note the model matches quite well to giraffe images in Hemera. For each animal class, we take the best shape match score obtained over all images in that class. We normalize the scores to obtain a shape posterior over animal labels in Figure 17. Using shape, we label the giraffe video as 'giraffe', but both the zebra and tiger video are mislabeled.

We compute a final posterior by adding the (log) texture and shape posteriors (weighting shape by $\frac{1}{2}$) in the bottom row of Figure 17. Selecting the best class, we identify the correct animal label for each of our videos.

Alternatives: One might attempt to the label the videos by *directly* matching the pictorial structure models built from the videos (Section VI) to Hemera. We found that the shape prior performs well, but the simplistic RGB models for part appearance produce poor matches in Hemera. Video is a good domain to learn shape (because motion constraints establish correspondence) but a poor domain to learn appearance (because we see only a single instance). It is hard to build a good giraffe texture model by looking at a single giraffe. On the other hand, image collections are convenient for learning appearance (because we see many instances) but not shape (because correspondence is unknown). Our matching procedure builds a shape and texture model separately, using the domain that is well-suited for each.

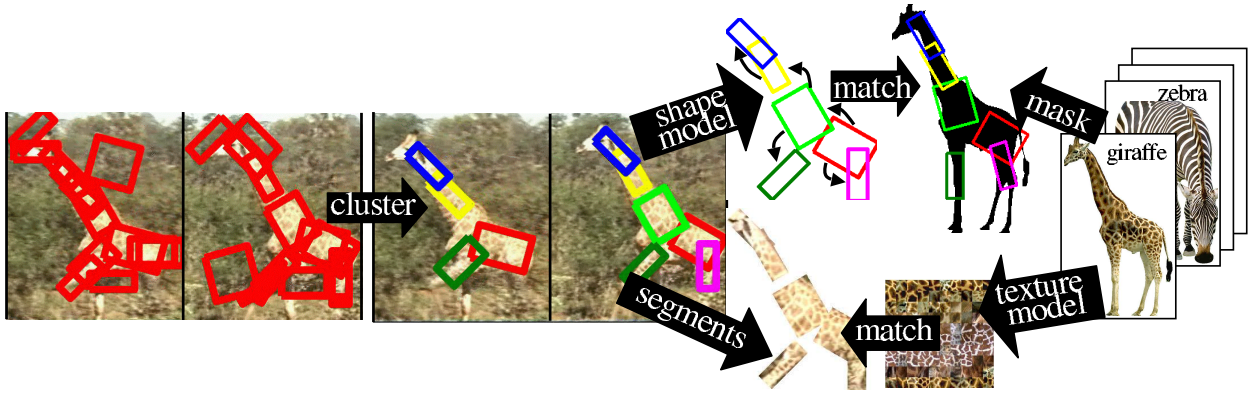

Fig. 16. We identify animals in videos by matching to labeled image collections. Given an animal video (**left**), we obtain spatio-temporal tracks of limbs by clustering (Section V). We use the tracks to learn a spatial model (Section VI) and segment the video into animal/non-animal pixels (Section VII). On the **right**, we build a texture model for various animals from the Hemera collection of labeled and segmented images. We link our models by matching the shape model built from video to the foreground mask of the Hemera images and matching the texture model built from Hemera to the segmented video (Section IX). This automatic matching *identifies* the animal in the video. We use the combined shape and texture model for recognition in Figure 20.

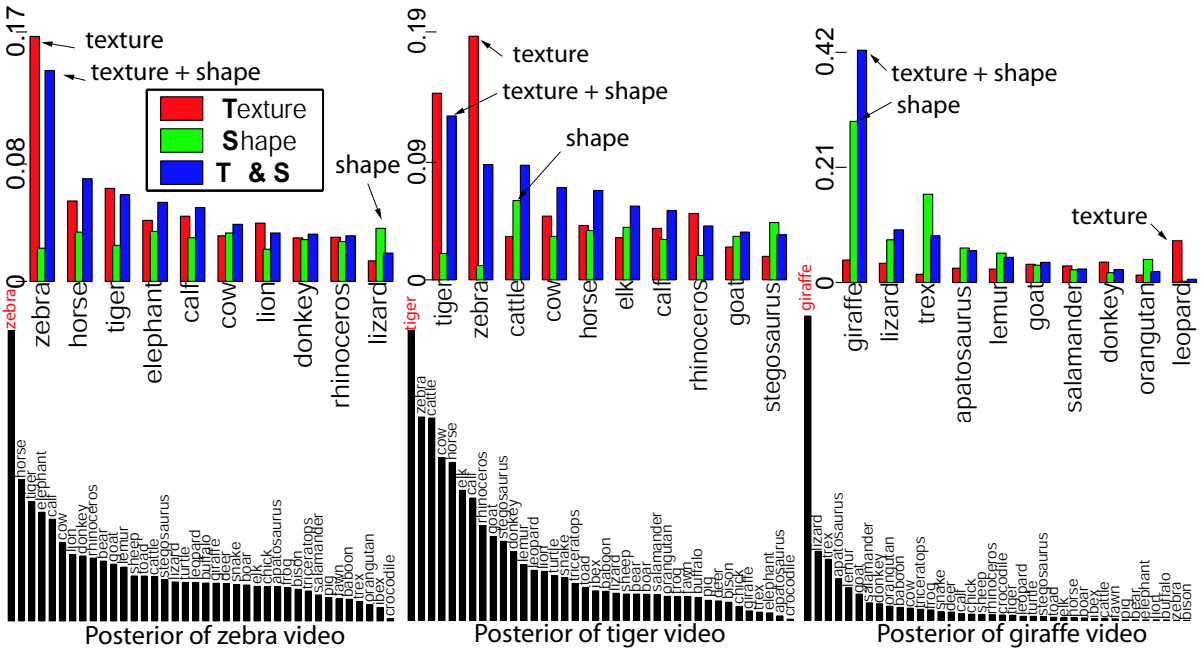

Fig. 17. *We identify the animals in our videos by linking the shape models built from video to the texture models built from the labeled Hemera image collection. We show posteriors of animal class labels given the zebra (***left***), tiger (***middle***), and giraffe (***right***) videos. In the* **top row***, we show posteriors of the ten best labels based on a texture cue, shape cue, and the combination of the two. We mark the MAP class estimate for each cue. Matching texture models built from Hemera to the segmented videos, we mislabel the giraffe video as 'leopard'. By matching shape models built from videos to Hemera images, we match the giraffe correctly, but incorrectly label the zebra and tiger videos. Combining the two cues, we match all the videos to the correct animal label. We show posteriors for the final combined cue over the entire set of labels in the* **bottom row***. Note the graphs are not scaled equally.*

X. Finding Animals in New Images

We use our patch-based texture model from Section VIII and the correspondence obtained from Section IX to build a system for finding animals in new images. We follow the approach outlined in Figure 20.

Given a query image, we first obtain a "foreground" mask using the texture library built in Section VIII. We replace each 17X17 image patch with the closest match from our library, using a SIFT descriptor. We append the Hemera animal texture library with a 'background' texture class of 20000 patches extracted from random Corel images (not in our test pool and not containing animals). We then construct a binary label image with a '1' if a patch was replaced with a given animal patch. We interpret this binary image as a foreground mask for that animal label, and use DP to find rectangles in the foreground arranged according to the shape model learned from video (Section VI). For the 'zebra','tiger', and 'giraffe' animal labels, we know the correct shape model to use because we have *automatically* linked them (Section IX). Hence our final animal detection system is completely automatic.

In practice, it is too expensive to classify every patch in a query image. Fortunately, the SIFT descriptor is designed to be somewhat translation invariant; off-by-one pixel errors should not affect it. This suggests we sample patches from the image,

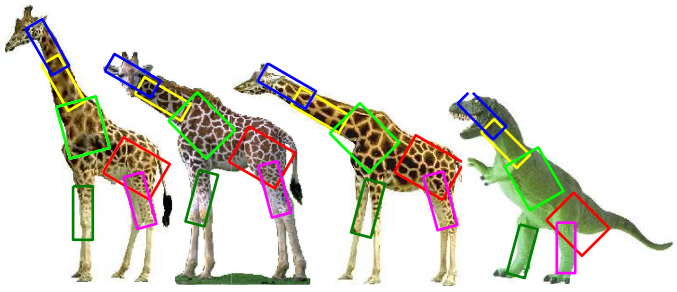

Fig. 18. The top 4 matches (the top match on the **left**) in the Hemera collection for the shape model learned from the giraffe video. Note our shape model captures the articulated variation in pose, resulting in accurate detections and reasonable false positives.

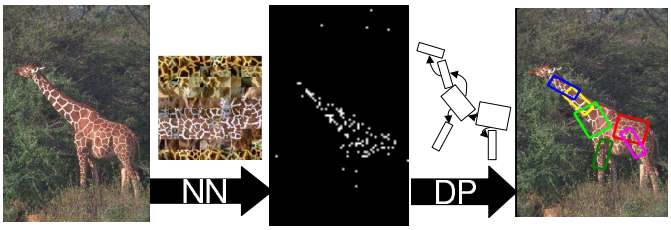

Fig. 19. Our model recognition algorithm, as described in Section X. Assume we wish to detect/localize a giraffe in a query image (**left**). We replace each image patch with its closest match from our library of Hemera animal and background textures (NN or nearest neighbor classification). We construct a binary label image with '1's for those patches replaced with a giraffe patch (**center**). We use dynamic programming (DP) to find a configuration of limbs that are likely under the shape model (learned from the video) *and* that lie on top of giraffe pixels in the label image (constructed from the image texture library). We show MAP limb configurations on the **right**.

and match them to our texture library; we match 5000 patches per image, which takes about 2 minutes in our implementation. Speeding up the matching using approximate nearest neighbor techniques [47] or building a parametric texture model may allow us to classify more patches from an image.

A. Evaluation

We tested our models on two datasets; images from the Corel collection and various animal images returned from Google. We scale images to be roughly the same dimension as our video clips. Our Corel set contained 304 images; 50 zebras, 120 tigers, 34 giraffes, and 100 random images from Corel. Note these random images are *different* from the set used to learn a background patch library. The second collection of 1418 images was constructed by assembling a random subset of animal images returned by Google. It contains 315 zebras, 70 tigers, 472 giraffes, and 561 images of other animals ('leopard', 'koala', 'beaver', 'cow', 'deer', 'elephant', 'monkey','antelope', 'parrot', and 'polar bear').

Detection. We show precision-recall (PR) curves in Figure 19. For the **S**hape detector, we build an animal detector using *only* the video and not Hemera. We build a crude texture library using positive and negative patches inside and outside the spatio-temporal tracks. Given a new image, we construct a binary label image by replacing patches with their closest match from this limited texture library. We then use DP to find the MAP configuration of limbs from the binary label image. For the **T**exture detector, we build a detector using *only* Hemera and not the video. We compute a binary label image using the entire patch library (Hemera animal patches plus background patches). Our final detector is a threshold on the sum of animal pixels (as in [5, 6]). For the **S & T** detector, we construct a binary label image using the entire patch library, and then use DP to find the MAP limb configuration. We compare our detectors with 2 baselines; a 1-NN classifier trained on color histograms and random guessing. We tried a variety of other classifiers as baselines (such as logistic regression and SVMs) but 1-NN performed the best.

Difficulty of datasets. Recognition is still relatively poorly understood, meaning that reports of absurdly high recognition rates can usually be ascribed to simplicity of the test set. Careful experimentation requires determining how difficult a dataset is; to do so, one should assess how simple baselines perform on that dataset [48–51]. This is often informative: for example, it is known that variations in reported performance between different face recognition algorithms are almost entirely explained by variations in the performance of the baseline on the dataset [48]. In almost all cases, our shape and texture animal models outperform the baselines of random guessing and color histogram classification. The one notable exception is our tiger detector on the Corel data, for which a color histogram outperforms all our methods. This can be ascribed to the insufficiently well known fact that Corel backgrounds are strongly correlated with Corel foregrounds (so that a Corel CD number can be predicted using simple color histogram features [51]). In the Google set, our baselines do worse, but our detectors do better. This *negative* correlation seems to stem from the fact that Google images have varied backgrounds, unlike Corel (see Figure 22 versus Figure 23). Such backgrounds hurts our global histogram baseline but may help our animal detector (since the animal might be easier to segment). Comparing to detection results reported in [5, 6], we obtain *better performance on a demonstrably harder dataset*.

Importance of shape. In almost all cases, adding shape greatly improves detection accuracy. An exception is detecting tigers in the Google set (Figure 19). We believe this is the case because of severe changes in scale; many tiger pictures are head shots, for which our shape model is not a good match (this also confuses our texture model, resulting in the lower overall performance). However, for low recall rates, shape is still useful in yielding high precision. The top few matches for the tiger detector will be tigers only if we use shape as a cue. Our results for shape are particularly impressive given the quality of our texture detector baseline. It has been shown that feature matching with SIFT features [50] produces quite good performance on established object recognition datasets [11]. Such a scheme is equivalent to our texture baseline, which we demonstrate is outperformed by our shape and texture detector.

Location and kinematic recovery. Looking at the best matches to our detectors (Figure 22 and Figure 23), we see that we reliably localize the detected animal and quite often we recover the correct configuration of limbs. We quantify this by manually evaluating the recovered configurations in Table

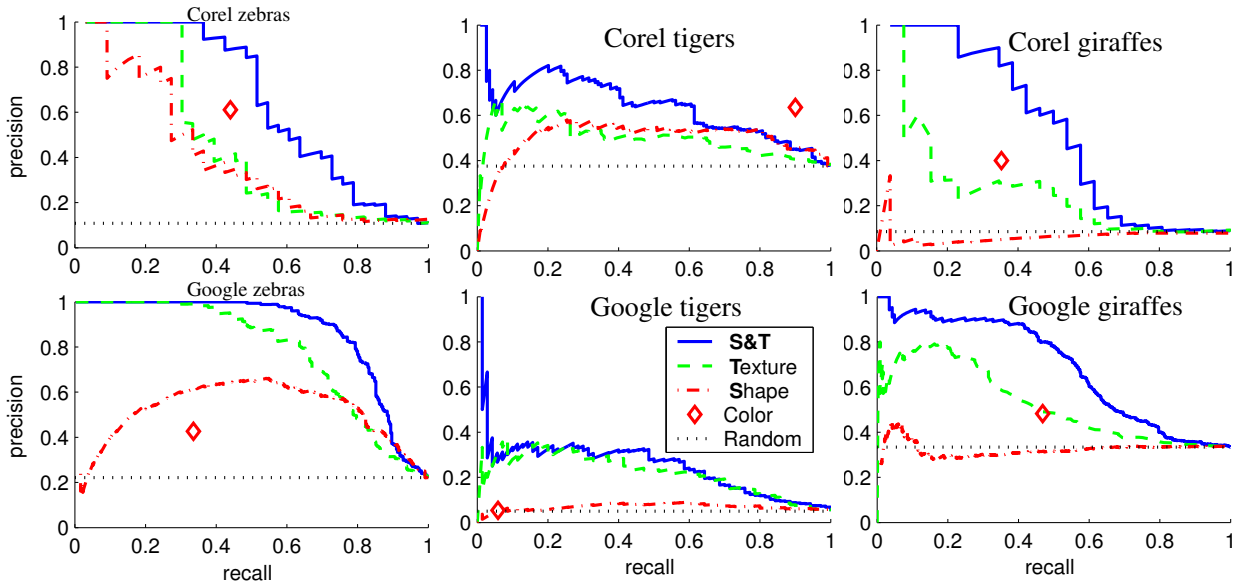

Fig. 20. Precision recall curves for zebra, tiger, and giraffe detectors run on a set of 304 Corel images (**top**) and 1418 images returned by Google (**bottom**). The '**S**hape' detectors are built using shape models and crude texture models learned from the video. The '**T**exture' detectors are built using texture models trained on the image collection. The **S & T** detectors use texture models from the image collection and shape models from the video (where the linking was automatic, as described in Section IX). We compare with 2 baselines; a 1-NN classifier trained on color histograms and random guessing. For the tiger detector run on Corel, the color histogram does quite well, suggesting we should look at the Corel dataset with suspicion. We show that, in general, shape improves detection performance. Comparing our zebra and giraffe detection results to [5, 6], we show *better performance on a demonstrably harder dataset*.

Percentage of correct localizations

Dataset	Zebra	Tiger	Giraffe
Corel	84.9	92.0	76.9
Google	94.0	94.0	68.0

(a)

Percentage of correctly estimated kinematics

Dataset	Zebra	Tiger	Giraffe
Corel	24.2	28.0	38.4
Google	30.0	34.0	46.0

(b)

TABLE II. Results for localization (**a**) and kinematic recovery (**b**). We define a correct localization to occur when a majority of the pixels within the estimated limbs are true animal pixels (we have a greater than 50% chance of hitting the animal if we shoot at the estimated limbs). We also show the percentage of animal images where the correct kinematics are recovered. By hand, we mark a configuration to be correct if a majority of the estimated limbs overlap a pixel region matching the semantic labeling from Figure 7. The kinematic results for the giraffe are impressive given the large number of different semantic labels; correct configurations tend to align the upper neck, the lower neck, the upper body, the lower body, the front leg, and the rear leg. Our animal detector localizes the animal quite well and often recovers reasonable configurations.

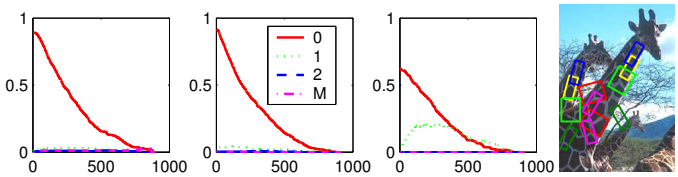

Fig. 21. Counting results for the zebra (**left**), tiger (**middle left**), and giraffe (**middle right**) models. We plot results for Corel. We show fraction of images with 'i' animals that were correctly classified as a function of our detector threshold (where $i \in \{0, 1, 2, many\}$ and many is 3 or more). We see that 20% percent of tiger images can be correctly classified as having 1 tiger. However, since animals often appear in herds and overlap, counting them in general is a difficult problem. We show an example of a difficult image on the **right**. Depending upon how one scores partial occlusions and multiple scales, there could be two to four giraffes present. Counting appears to be a quite difficult object recognition task [52].

II. We define a correct localization to occur when a majority of the pixels covered by the estimated limbs are animal pixels (if we shoot at the estimated limbs, we'll most likely hit the animal). We define a kinematic recovery as correct when a majority of the limbs overlap a pixel region with the correct semantic label from Figure 7. The pose results for the giraffe are impressive given the large number of different semantic labels; correct configurations tend to align the upper neck, the lower neck, the upper body, the lower body, the front leg, and the rear leg. In general, we correctly localize the animal, and often we recover a reasonable estimate of its configuration, although we suffer from scale issues (see Figure 23).

Counting. We detect multiple instances of the same animal in a single image by finding the MAP animal configuration, masking away those pixels covered by the estimated limbs, and repeating. We are able to successfully classify 20% of the tiger pictures from our Corel set that contain one tiger as having one tiger. In general, we do quite poor at counting because animals often occur together in herds; this confuses our greedy counting procedure, which would work better on well separated animals in an image. Counting remains a challenging problem for object recognition; relatively few systems have demonstrated results [52].

Another important application of accurate localization is the ability to apply *mutual exclusion*. Since our tiger detector often becomes confused by zebras, we would expect much better performance if upon finding a zebra with our zebra detector, we masked away those pixels before applying the tiger detector. This strategy will only work with reasonably accurate localization.

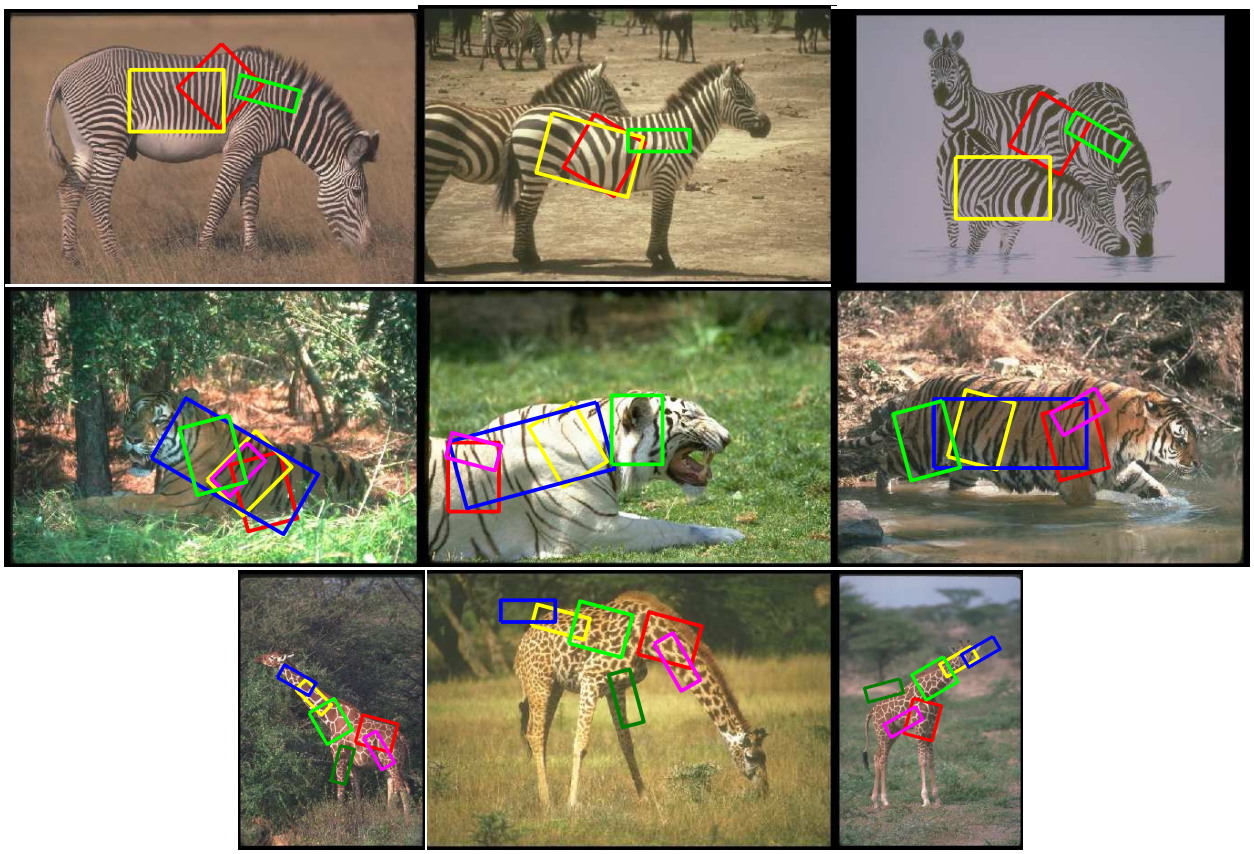

Fig. 22. Results for our zebra (**top row**), tiger (**middle row**), and giraffe (**bottom row**) models using shape and texture on a test pool of 304 Corel animal images. We show the top scoring detections for each detector. Even though this dataset is relatively easy for detection (by evidence of good baseline performance), we can still evaluate localization and kinematic recovery results. We localize the animal quite well, and often recover reasonable kinematic estimates (though sometimes we have trouble determining which direction an animal is facing).

XI. Discussion

One contribution of this work is a novel (but simple) representation of texture; rather than using a histogram of textons, we represent texture with a patch of pixels. We demonstrate that this representation outperforms the state-of-the-art for our task of detecting animals.

Limitations: Pictorial structure models seem to be aspect-dependent; if we learn a model from a video of giraffe walking sideways, we may not be able to use that model to find a giraffe walking towards a camera. One line of attack might be a mixture of pictorial structures, where each encodes a single aspect. Our clustering method of building spatio-temporal tracks also seems limited to videos with single animals and relatively little background clutter (the same restrictions imposed on other unsupervised model-building algorithms [10, 11]). Possible methods of removing such restrictions would be to use a stronger model of the background (e.g., background subtraction techniques) and to use a spatial prior (perhaps of 4-legged animals) in the clustering procedure (as in [53]).

Broadly speaking, we introduce (and rigorously evaluate) an unsupervised system for learning articulated models using video. Video is useful because both motion and appearance consistency are strong cues for learning. Such cues allow us to learn fairly complex pictorial structures with internal kinematics. These models allow us to *automatically* track a deforming animal in a video and identify the animal from an image library. One would also hope to use the models to find animals in new images. This turns out to be hard because of a fundamental limitation of video; only a single object instance is observed, and so the learned appearance is too specific. We show a useful strategy of combining models learned from video and image collections (where multiple instances are observed). These learned models appear promising for recognition tasks beyond detection, such as localization, kinematic recovery, and (possibly) counting.

References

[1] M. A. Fischler and R. A. Elschlager, "The representation and matching of pictorial structures," *IEEE Transactions on Computer*, vol. 1, no. 22, pp. 67–92, January 1973.

[2] U. Grenander, Y. Chow, and D. Keenan, *Hands: a pattern theoretic study of biological shapes*. Springer-Verlag, 1991.

[3] T. Cootes, G. Edwards, and C. Taylor, "Active appearance models," in *European Conference on Computer Vision*, 1998.

[4] M. Burl, M.Weber, and P. Perona, "A probabilistic approach to object recognition using local photometry and global geometry," in *ECCV*, 1998, pp. 628–641.

[5] C. Schmid, "Constructing models for content-based image retrieval," in *Proc CVPR*, 2001.

[6] S. Lazebnik, C. Schmid, and J. Ponce, "Affine-invariant local descriptors and neighborhood statistics for texture recognition," in *ICCV*, 2003.

[7] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *Int. J. Computer Vision*, vol. 61, no. 1, January 2005.

[8] S. Ioffe and D. A. Forsyth, "Human tracking with mixtures of trees," in *ICCV*, 2001.

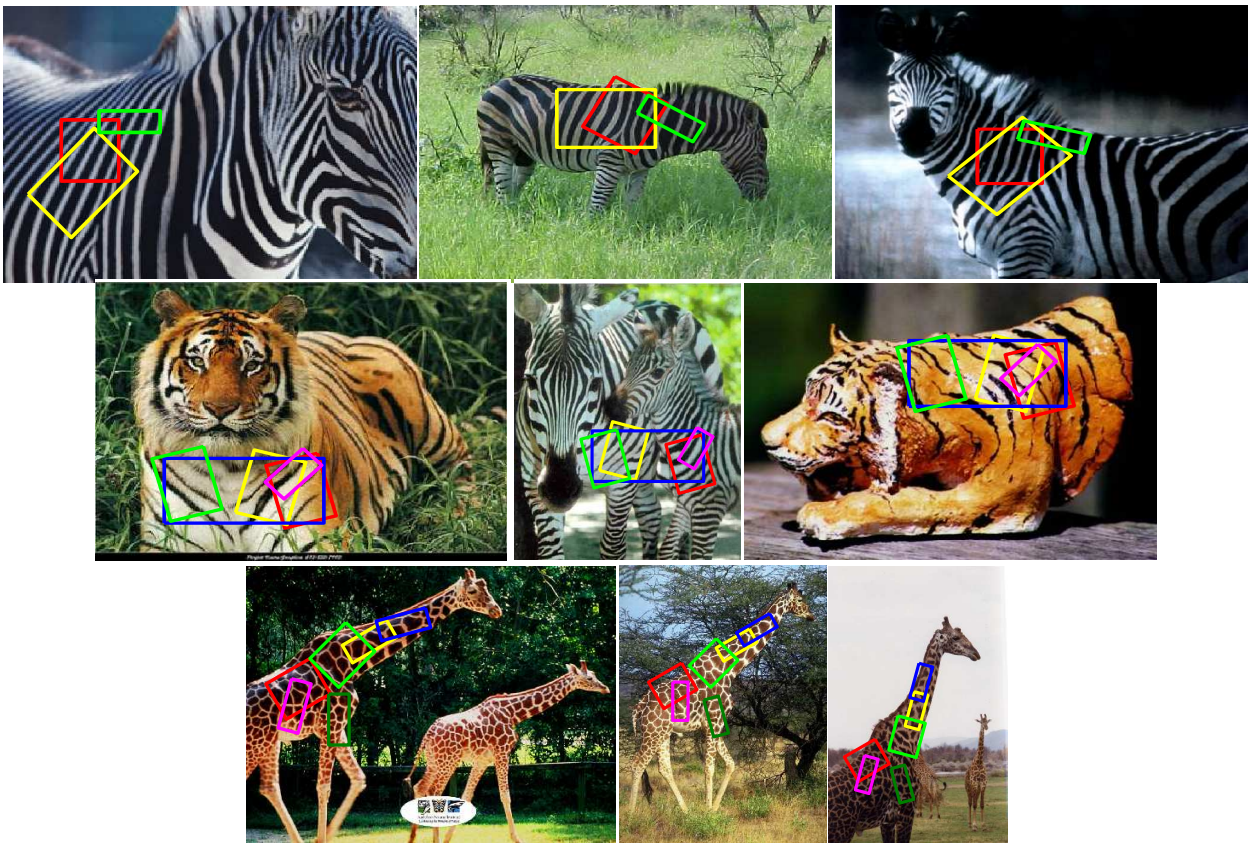

Fig. 23. Results for our zebra (**top row**), tiger (**middle row**), and giraffe (**bottom row**) models using shape and texture on a test pool of 1418 animal images obtained from Google. We show the top scoring detections for each detector. Our tiger model mistakenly fires on a Google zebra due to the similar texture. The quasi-correct zebra configurations suggest our shape model might perform better if we searched over scale. The giraffe configurations tend to be quite good. The Google results are impressive given the poor performance of our baselines; we are detecting, localizing, and often recovering reasonable pose estimates for objects in a dataset *demonstrably hard for object recognition*.

[9] T. Leung, M. Burl, and P. Perona, "Finding faces in cluttered scenes using random labelled graph matching," in *Int. Conf. on Computer Vision*, 1995.

[10] M. Weber, M. Welling, and P. Perona, "Unsupervised learning of models for recognition," in *ECCV (1)*, 2000, pp. 18–32. [Online]. Available: citeseer.nj.nec.com/weber00unsupervised.html

[11] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," in *CVPR*, 2003.

[12] D. Ramanan and D. A. Forsyth, "Using temporal coherence to build models of animals," in *ICCV*, 2003.

[13] M. Kumar, P. Torr, and A. Zisserman, "Learning layered pictorial structures from video," in *Indian Conference on Vision, Graphics and Image Processing*, 2004.

[14] D. Ramanan, "Tracking people and recognizing their activities," Ph.D. dissertation, U.C. Berkeley, 2005.

[15] D. Hogg, "Model based vision: a program to see a walking person," *Image and Vision Computing*, vol. 1, no. 1, pp. 5–20, 1983.

[16] J. O'Rourke and N. Badler, "Model-based image analysis of human motion using constraint propagation," *IEEE T. Pattern Analysis and Machine Intelligence*, vol. 2, pp. 522–546, 1980.

[17] C. Bregler and J. Malik, "Tracking people with twists and exponential maps," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 1998, pp. 8–15.

[18] D. Gavrila and L. Davis, "3d model-based tracking of humans in action: a multi-view approach," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 1996, pp. 73–80.

[19] K. Rohr, "Incremental recognition of pedestrians from image sequences," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 1993, pp. 9–13.

[20] H. Sidenbladh, M. J. Black, and D. J. Fleet, "Stochastic tracking of 3d human figures using 2d image motion," in *European Conference on Computer Vision*, 2000.

[21] A. Blake and M. Isard, "Condensation - conditional density propagation for visual tracking," *Int. J. Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.

[22] K. Toyama and A. Blake, "Probabilistic tracking with exemplars in a metric space," *Int. J. Computer Vision*, vol. 48, no. 1, pp. 9–19, 2002.

[23] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.

[24] G. Mori and J. Malik, "Estimating human body configurations using shape context matching," in *ECCV*, 2002.

[25] J. Sullivan and S. Carlsson, "Recognizing and tracking human action," in *European Conference on Computer Vision*, 2002.

[26] D. M. Gavrila, "Pedestrian detection from a moving vehicle," in *European Conference on Computer Vision*, 2000, pp. 37–49.

[27] N. Jojic and B. Frey, "Learning flexible sprites in video layers," in *CVPR*, 2001.

[28] M. Brand, "Morphable 3d models from video," in *CVPR*, 2001.

[29] L. Torresani, D. Yang, G. Alexander, and C. Bregler, "Tracking and modeling non-rigid objects with rank constraints," in *CVPR*, 2001.

[30] M. Weber, M. Welling, and P. Perona, "Unsupervised learning of models for recognition," in *ECCV (1)*, 2000, pp. 18–32.

[31] P. Duygulu, K. Barnard, N. de Freitas, and D. Forsyth, "Object recognition as machine translation," in *Proc. European Conference on Computer Vision*, 2002, pp. IV: 97–112.

[32] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *CVPR*, 2001.

[33] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE T. Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[34] D. Ramanan and D. A. Forsyth, "Finding and tracking people from the bottom up," in *Proc CVPR*, 2003.

[35] J. Coughlan and S. Ferreira, "Finding deformable shapes using loopy belief propogation," in *Proc ECCV*, 2002.

[36] M.Wainwright, T. Jaakola, and A.Willsky, "Tree-based reparameterization for approximate inference on loopy graphs," in *NIPS*, 2001.

[37] S. Ioffe and D. Forsyth, "Finding people by sampling," in *Int. Conf. on Computer Vision*, 1999, pp. 1092–1097.
[38] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut - interactive foreground extraction using iterated graph cuts," *Proc. ACM Siggraph*, 2004.
[39] D. Ramanan, D. Forsyth, and A. Zisserman, "Strike a pose: Tracking people by finding stylized poses," in *CVPR*, June 2005.
[40] Y. Song, X. Feng, and P. Perona, "Towards detection of human motion," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2000, pp. 810–17.
[41] K. Dana, S. Nayar, B. van Ginneken, and J. Koenderink, "Reflectance and texture of real-world surfaces," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 1997, pp. 151–157.
[42] *Hemera Photo Objects*, Hemera Technologies, Inc, http://www.hemera.com.
[43] T. Leung and J. Malik, "Representing and recognizing the visual appearance of materials using three-dimensional textons," *Int. J. Computer Vision*, vol. 43, no. 1, pp. 29–44, 2001.
[44] M. Varma and A. Zisserman, "Texture classification: Are filter banks necessary?" in *CVPR*, 2003.
[45] D. Lowe, "Object recognition from local scale-invariant features," in *ICCV*, 1999.
[46] A. Berg and J. Malik, "Geometric blur for template matching," in *CVPR*, 2001.
[47] P. Indyk and R. Motwani, "Approximate nearest neighbor - towards removing the curse of dimensionality," in *30th Symposium on Theory of Computing*, 1998.
[48] P. Phillips and E. Newton, "Meta-analysis of face recognition algorithms," in *Proceeedings of the Int. Conf. on Automatic Face and Gesture Recognition*, 2002.
[49] M. E. Nilsback and B. Caputo, "Cue integration through discriminative accumulation," in *CVPR*, 2004.
[50] G. Dorko and C. Schmid, "Object class recognition using discriminative local features," iEEE PAMI, under preparation.
[51] O. Chapelle, P. Haffner, and V. Vapnik, "Support vector machines for histogram-based image classification," *IEEE Neural Networks*, vol. 10, no. 5, pp. 1055–1064, 1999.
[52] S. Ioffe and D. Forsyth, "Probabilistic methods for finding people," *Int. J. Computer Vision*, 2001.
[53] D. Ramanan and D. A. Forsyth, "Finding and tracking people from the bottom up," in *CVPR*, 2003.