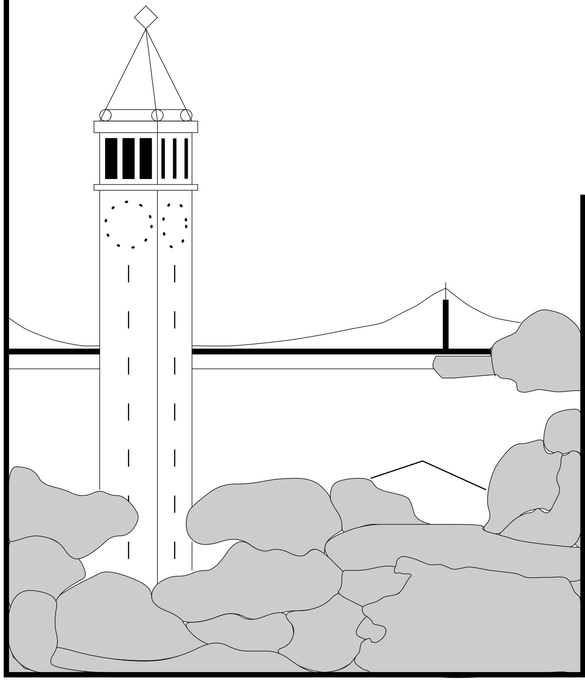
Detecting, Localizing and Recovering Kinematics of Textured Animals

Deva Ramanan[1] *and D. A. Forsyth*[1] *and Kobus Barnard*[2]
[1]*University of California, Berkeley – Berkeley, CA 94720*
[2]*University of Arizona – Tuscon, AZ 85721*
{*ramanan,daf*}@cs.berkeley.edu, kobus@cs.arizona.edu



Report No. UCB/CSD-4-1361

November 2004

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Detecting, Localizing and Recovering Kinematics of Textured Animals

Deva Ramanan[1] and D. A. Forsyth[1] and Kobus Barnard[2]
[1]University of California, Berkeley – Berkeley, CA 94720
[2]University of Arizona – Tuscon, AZ 85721
{ramanan,daf}@cs.berkeley.edu, kobus@cs.arizona.edu

November 2004

Abstract

We develop and demonstrate an object recognition system capable of accurately detecting, localizing, and recovering the kinematic configuration of textured animals in real images. We build a deformation model of shape automatically from videos of animals and an appearance model of texture from a labeled collection of animal images, and combine the two models automatically. We develop a simple texture descriptor that outperforms the state of the art. We test our animal models on 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 recognize objects from a collection demonstrably hard for object recognition.

1 Introduction

Learning shape is difficult, particularly for deformable objects and objects undergoing changes in aspect. One option is to encode each deformed state (or aspect) with its own individual template; this is usually impractical. The alternative is to require encodings to share parts with explicit kinematics. The latter is attractive because parts tend to have useful semantics; for example, it is valuable know where the head, neck, body and legs of a giraffe are. Learning such a model from images is difficult because we must solve the correspondence problem. To build a good shape model, 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.

A recent paper demonstrates that **video** addresses this correspondence problem; motion constraints help determine what image regions move where [13]. Kinematic models are built by searching for possible animal limbs that look consistent over time and that move smoothly from frame to frame; the resulting models can detect animals. However, the spatial model is rough, and there are no results for localization or kinematic recovery. In this paper, we show how to build a significantly improved spatial model (Sec. 2.2).

A further difficulty with the work described in [13] is that the model of limb appearance is overly tuned to the specific animal captured in the video. In this paper, we describe an effective, *discriminative* texture model, built from a collection of segmented images (we use a set of labeled animal pictures from the Hemera Photo-Object database [15]). Our model is capable of recognizing giraffe textures (which are notoriously difficult because they have structure at two spatial scales — see [14, 7] and Fig.5). We compare our model with various texture descriptors in Sec.4 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.

We now have kinematic models of unknown animals, built using video, and texture models of known animals using the Hemera collection. We show that, by matching the models to one another, we can combine them automatically to produce a fused model. The name of the modeled object is known (because it is obtained from Hemera); the model has a powerful, discriminative texture representation (obtained as above); and it encodes kinematic variations in a form that respects limb semantics (obtained using video).

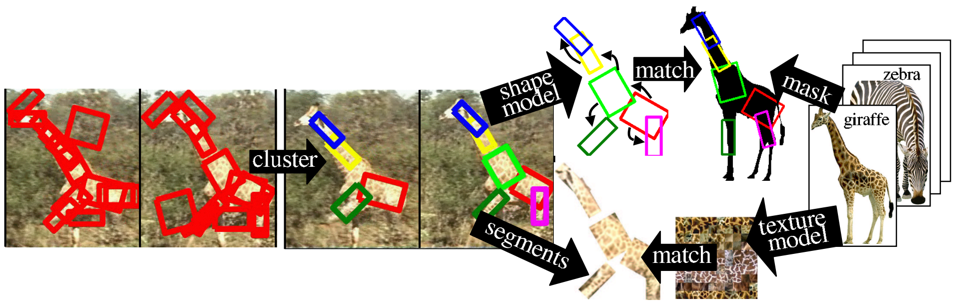


Figure 1: *Our model-building algorithm. Given an animal video (***left***), we find candidate limbs in each frame. We cluster the associated image patches to find sets of limbs that are coherent in appearance over time and that move with bounded velocity (Sec.2.1). The clustered limbs segment the video into animal/non-animal pixels and allow us to learn a spatial model (Sec. 2.2). 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 (Sec.4.1). This automatic matching* identifies *the animal in the video. We use the combined shape and texture model in Fig. 2.*

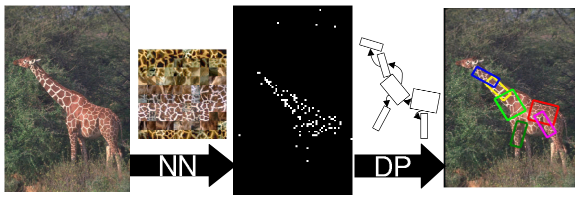


Figure 2: *Our model recognition algorithm, as described in Sec.4.2. Assume we wish to detect/localize a giraffe in a query image (***left***). We replace each image patch with its closest match from our library of Hemera animal and background textures (NN or nearest neighbor classification). We construct 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***.*

First, we demonstrate that such a model can detect animals. Detection is known to be a relatively easy task, and detection results are often inflated by backgrounds correlated with the object. Here, we use baselines to obtain datasets that are demonstrably harder than those currently in use, and to demonstrate that our model cannot be using background cues to detect objects. Second, we demonstrate that our model can localize animals — if one were to shoot at the model, there would be a high probability of hitting the relevant animal. Finally, we show that our model reports kinematics — estimated animal parts lie on image regions with the correct (manually annotated) semantic label.

2 Building shape models from video

We use the algorithm of [13] (briefly described in Sec.2.1) to recover animal segments from video. We use an improved spatial model explained in Sec.2.2.

2.1 Finding and linking limbs

Assume we are given a video sequence, and told a single unknown animal is present. There are two powerful cues that can be used to establish what it looks like: first, the animal is, rather roughly, assembled out of limbs. Second, the limbs will look the same from frame to frame — their appearance is coherent in time.

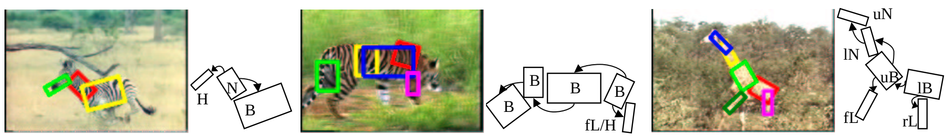


Figure 3: *We show model-building results for videos of a zebra* **left**, *tiger* **center**, *and giraffe* **right**. *We learn a tree shape model for limbs, as described in Sec. 2.2. We show directed links in our tree models with arrows. We manually attach a semantic description to each limb as Head, 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.*

To exploit the first, we run a detuned limb detector on each frame independently to generate a set of candidate limbs. The detector searches for parallel lines of contrast at a variety of positions, orientations, and scales. We expect the detector to perform poorly; it may miss animal limbs in many frames and it may fire on the background. Given the set of detected limbs, we can extract the image patch associated with each candidate and throw them in a "bag". To exploit the second cue, we cluster this bag of patches to look for collections of patches from different frames that look the same. True animal limbs look coherent over time *and* move smoothly. We enforce this constraint by searching within each cluster for the longest sequence of patches obeying a bounded-velocity motion model. We finally prune away those clusters which do not span enough frames.

We interpret each remaining cluster as one single limb in the final animal body model. Recall that the patches from each cluster point to detected candidates from the video. This means that each cluster/limb has associated with it a spatio-temporal track, where the instances all look like one another and they move with bounded velocity. This set of tracks is a track of the animal (Fig. 1) [13, 12].

2.2 Building a good spatial model

Given the tracked segments from the video, we want to build a kinematic spatial model that can find the animal in other images. We improve upon the spatial model presented in [13]. In that work, the spatio-temporal tracks generated from the clustering are interpreted as observed data, and the authors learn the tree structure that maximizes the log-likelihood of the observations [13, 4]. Consider a fully connected graphical model of limb positions, $P_{head}, P_{neck}, P_{body}$ (the precise limb labels are not needed so long as their correspondence between frames is known). Each link represents a joint distribution $\Pr(P_{limb1}, P_{limb2})$ of pairwise limb positions, to which they fit a (diagonal covariance) gaussian using the data from the tracks. The position of each limb is represented with a 3 element vector; its center (x, y) position and orientation θ. As in [4], the position of each non-root limb is represented with respect to the coordinate system of its parent.

To learn the tree structure that maximizes the log-likelihood of the observed pairwise marginals, [13] 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 of the detected limbs are quite noisy (due to the detuned limb detector), in turn producing noisy entropy estimates. To enforce the natural prior that two limbs that tend to appear near each other should be spatially linked together, we replace the pairwise 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). Once we have learned a structure using the minimum distance spanning tree, we represent the conditional distributions with gaussians (as in [13]). This produces the tree spatial models in Fig. 3. We now can write the posterior configuration for an animal given the image as

$$\Pr(P_1 \ldots P_n | Im) \propto \prod_{(i,j)\in E} \Pr(P_i|P_j) \prod_{i=1}^{n} \Pr(Im(P_i)) \tag{1}$$

where we have assumed E is the set of edges in the minimum spanning tree and the likelihood term $\Pr(Im|P_1 \ldots P_n)$ factorizes into a product of local image functions $\Pr(Im(P_i))$. Note in our current implementation we do not search over scale, a limitation we discuss further in Sec.5. We can efficiently compute the MAP estimate by fast variant of dynamic

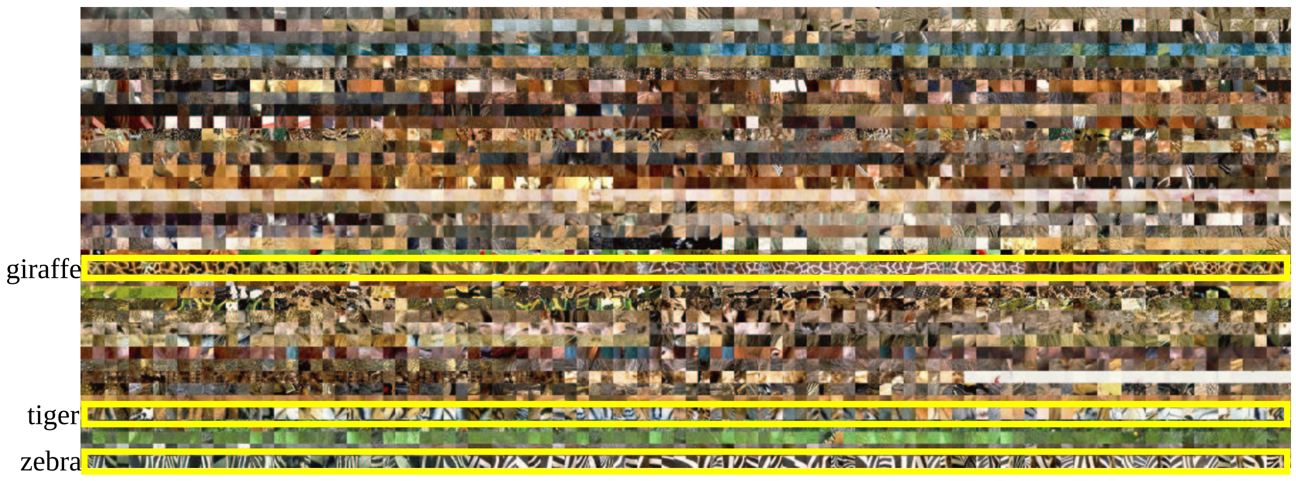


Figure 4: *Our library of animal textures built from Hemera. We show a subset of 100 17X17 patches for each of our 38 animals; we mark the giraffe, tiger, and zebra rows in yellow. Our recognition task requires texture classification* and segmentation *(we need to separate the animal from its background). This means we need to evaluate textures on a local image patch. We use this library to evaluate patch descriptors in Table 1.*

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 1: *We count how often we can correctly identify an animal based on texture from a single patch from Fig.4. 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 Fig.5.*

programming (DP) [4]. We use this posterior as an animal detector by only accepting those maximal configurations above a threshold.

The likelihood term $\Pr(Im(P_i))$ is a statistic that captures how "giraffe-like" a local image patch is. A natural statistic to use is the output of a giraffe texture classifier run on that local patch (see Fig.2). Building a good giraffe texture classifier from a single video is hard because there are not many positive or negative examples. This suggests we combine our spatial model from video with a texture model learned elsewhere.

3 Texture models from real images

We use the Hemera Photo-Object[15] database of image clip art to build a texture library; 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. This leaves us with about 500 images spanning 38 animals. We scale the Hemera images and video clips to be similar sizes, and assume the animals in the video and in Hemera occur at roughly the same scale.

To build a good animal texture model, we need a good animal texture descriptor capable of segmenting out the animal from its background (giraffes typically occur in backgrounds of fields and trees). Descriptors developed for standard vision datasets (such as CUReT [2]) may not be appropriate since they classify entire images of homogeneous texture. To evaluate descriptors on small image patches, we use the Hemera animal collection as a labeled ground-truth set (Fig.4). In Table 1, we

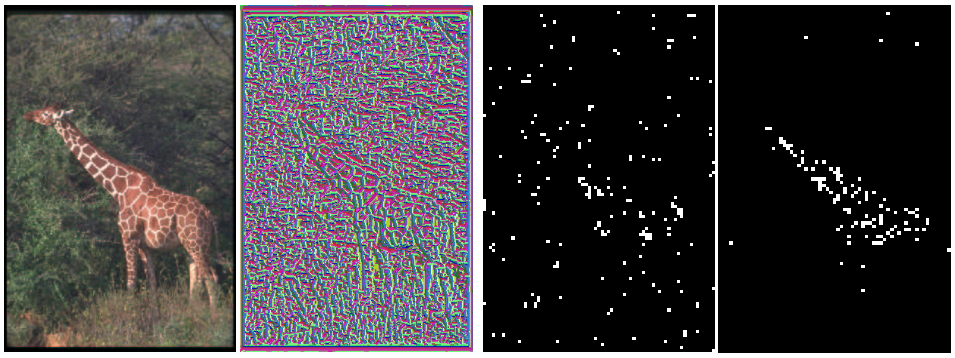


Figure 5: *Given a query image* **left***, we replace each 17X17 patch with its closest match from a patch in our texture library. This means we need a good animal texture descriptor; one that captures the long thin stripes that lie within big blobs typical in a giraffe. Standard approaches use histograms of textons (quantized filter bank outputs) [14, 7, 8, 16]. 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 [14, 7]). Rather, if we classify patches using a descriptor capturing spatial arrangement of pixels (e.g. SIFT), we are better at detecting giraffe patches (***right***).*

compare 3 different patch descriptors; histograms of textons [8], intensity-normalized patch pixel values [16], and the SIFT descriptor [9]. **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 without normalizing a patch to its dominant orientation (as in[9]). We extracted 1500 17X17 image patches from each animal class and built a 1-Nearest Neighbor (NN) classifier based on each descriptor. Using cross validation, we found the SIFT descriptor to perform the best.

Giraffes present particular difficulties for texton based descriptions (e.g. [14, 7];Fig.5). 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 (Fig.5). 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.

Our results are surprising because SIFT was not designed to represent texture (as noted in [9]); however we find it can given we store enough examples. For each of the 38 animals from our Hemera collection, we extract a set of 1500 patches of dimension 17X17, representing each patch with its SIFT descriptor. We attempted to reduce the set of our library, but saw a loss in performance in our cross-validation experiments. Obtaining a simpler parametric representation of animal texture remains future work.

4 Combining shape from video and texture from images

In order to combine the shape model learned from a video with the correct animal texture model from Hemera, we must first identify what animal is in the video.

4.1 Identifying animals from video

We assume that the animal in a given video is one of the 38 animals in Hemera. We match the texture models built from **Hemera to the video**. We segment the video into animal/non-animal pixels using the spatio-temporal tracks from Sec.2.1. 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).

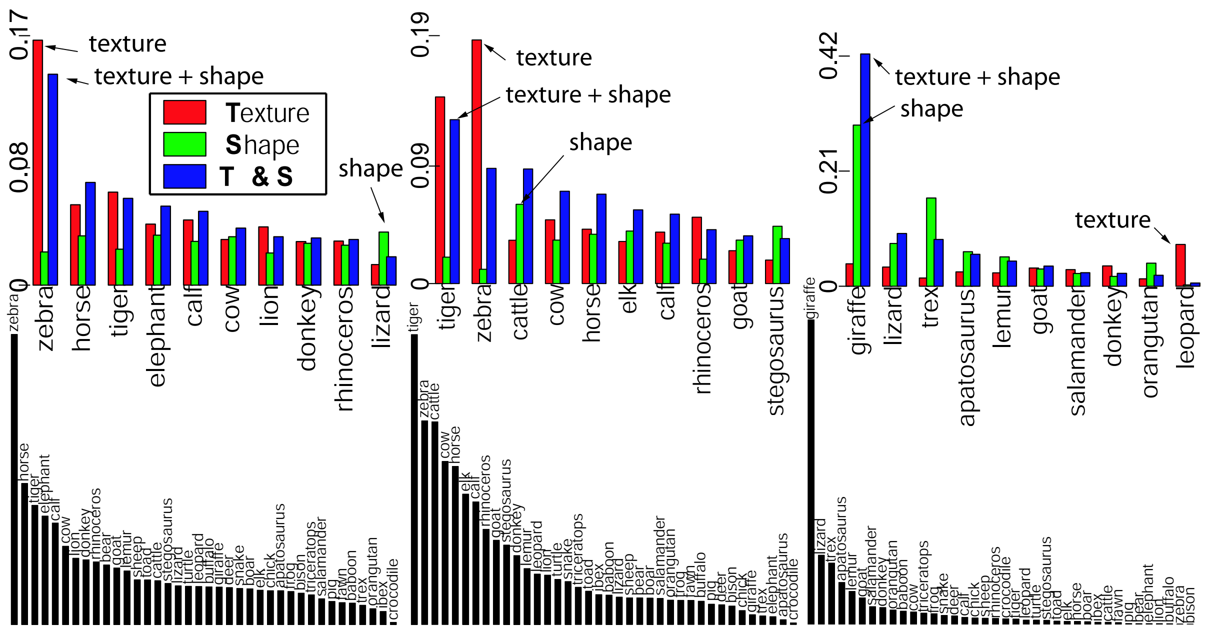


Figure 6: *We identify the animals in our videos by linking the shape models built from video to the texture models built from the labeled Hemera image collection. We show posteriors of animal class labels given the zebra (***left***), tiger (***middle***), and giraffe (***right***) videos. In the* **top 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.*

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 (Fig.6).

Matching solely based on texture is not enough to get the correct animal label; the giraffe video matches best with a 'leopard' texture. 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 DP to find a configuration of limbs that occupies the foreground mask *and* that is arranged according to the shape prior learned from the video (Fig.3). We evaluate the likelihood term $Pr(Im(P_i))$ in Eq.1 by convolving the foreground mask with oriented rectangle kernels of all '1's flanked on either side by rectangles of '-1's (we want the limbs to occupy the foreground region and not the background). We show 4 matches for our giraffe shape model in Fig.7; 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 Fig.6. 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 Fig.6. Selecting the best class, we identify the correct animal label for each of our videos.

4.2 Recognition using shape and texture

Our final recognition algorithm uses the approach outlined in Fig.2. Given a query image, we replace each 17X17 image patch with the closest match from our texture library. 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 (Eq.1). For the 'zebra','tiger', and 'giraffe' animal labels, we know the correct shape model

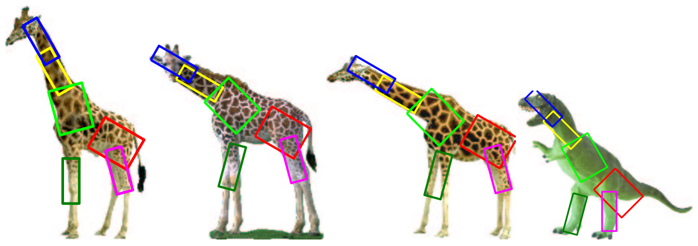


Figure 7: *The top 4 matches (the top match on the* **left***) in the Hemera collection for the shape model learned from the giraffe video. Note our shape model captures the articulated variation in pose, resulting in accurate detections and reasonable false positives.*

to use because we have automatically linked them (Sec.4.1).

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 the descriptor. This suggests we sample patches from the image, 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 [6] or building a parametric texture model may allow us to classify more patches from an image.

5 Results

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 Fig.8. 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 [14, 7]). 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 [11, 10, 3, 1]. 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 [11]. 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 [1]). Our detector performance seems to be decoupled from the baseline performance; in the Google set, our detectors do better even when the baselines do worse. This seems to be because, unlike Corel, images from Google tend to have varied backgrounds (Fig. 9), which hurts our histogram baseline but helps our animal detector. Comparing to detection results reported in [14, 7], we obtain *better performance on a demonstrably harder dataset*.

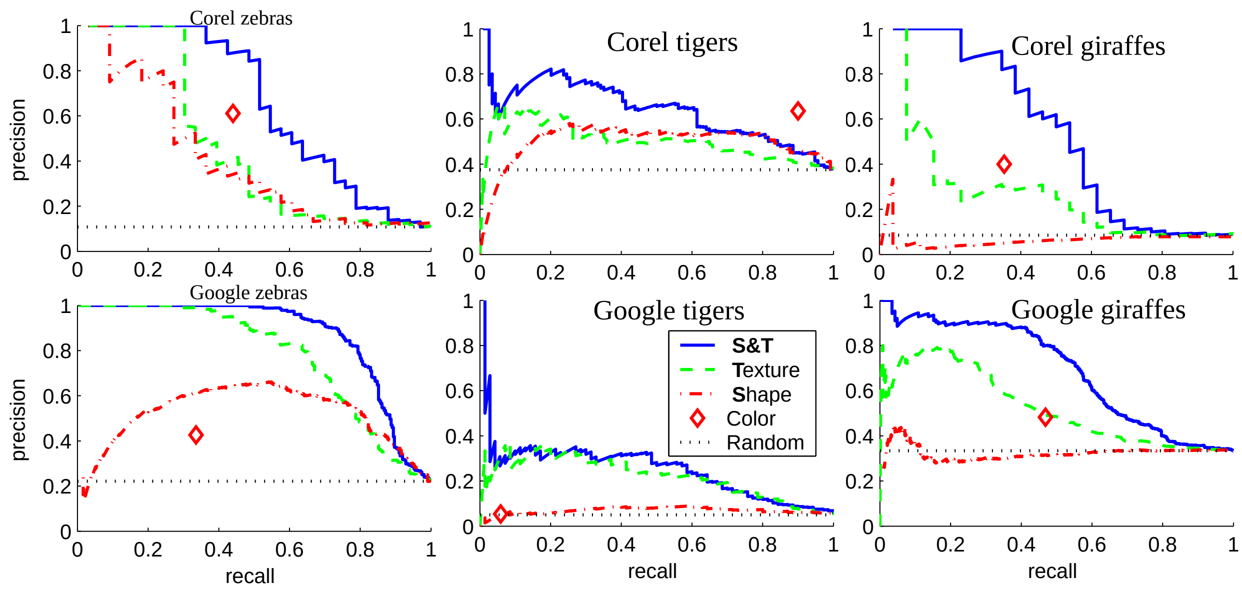


Figure 8: *Precision recall curves for zebra, tiger, and giraffe detectors run on a set of 304 Corel images (***top***) and 1418 images returned by Google (***bottom***). The 'Shape' detectors are built using shape models and crude texture models learned from the video. The "Texture' detectors are built 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 Sec. 4.1). 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 [14, 7], we show* 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 (Fig.8). 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 [3] produces quite good performance on established object recognition datasets [5]. Such a scheme is equivalent to our texture baseline, which we demonstrate is outperformed by our shape & texture detector.

Location and kinematic recovery. Looking at the best matches to our detectors (Fig.9), 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 2. 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 mark a pose as correct when a majority of the limbs overlap a pixel region with the correct semantic label from Fig.3. 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.

References

[1] O. Chapelle, P. Haffner, and V. Vapnik. Svm's for histogram-based image classification. In *IEEE Trans. on Neural Networks*, 1999.

[2] K. Dana, S. Nayar, B. van Ginneken, and J. Koenderink. Reflectance and texture of real-world surfaces. In *CVPR*, pages 151–157, 1997.

[3] G. Dorko and C. Schmid. Object class recognition under discriminative local features. IEEE PAMI, under preparation.

Percentage of correct localizations

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

Percentage of correctly estimated kinematics

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

Table 2: *Results for localization (***top***) and kinematic recovery (***bottom***). 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 Fig.3. 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.*

[4] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient matching of pictorial structures. In *CVPR*, 2000.

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

[6] P. Indyk and R. Motwani. Approximate nearest neighbor - towards removing the curse of dimensionality. In *30th Symposium on Theory of Computing*, 1998.

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

[8] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. J. Computer Vision*, 43(1):29–44, 2001.

[9] D. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[10] M. E. Nilsback and B. Caputo. Cue integration through discriminative accumulation. In *CVPR*, 2004.

[11] P. Phillips and E. Newton. Meta-analysis of face recognition algorithms. In *Proceeedings of the Int. Conf. on Automatic Face and Gesture Recognition*, 2002.

[12] D. Ramanan and D. A. Forsyth. Finding and tracking people from the bottom up. In *CVPR*, 2003.

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

[14] C. Schmid. Constructing models for content-based image retreival. In *CVPR*, 2001.

[15] H. Technologies. Hemera photo objects.

[16] M. Varma and A. Zisserman. Texture classification: Are filter banks necessary? In *CVPR*, 2003.

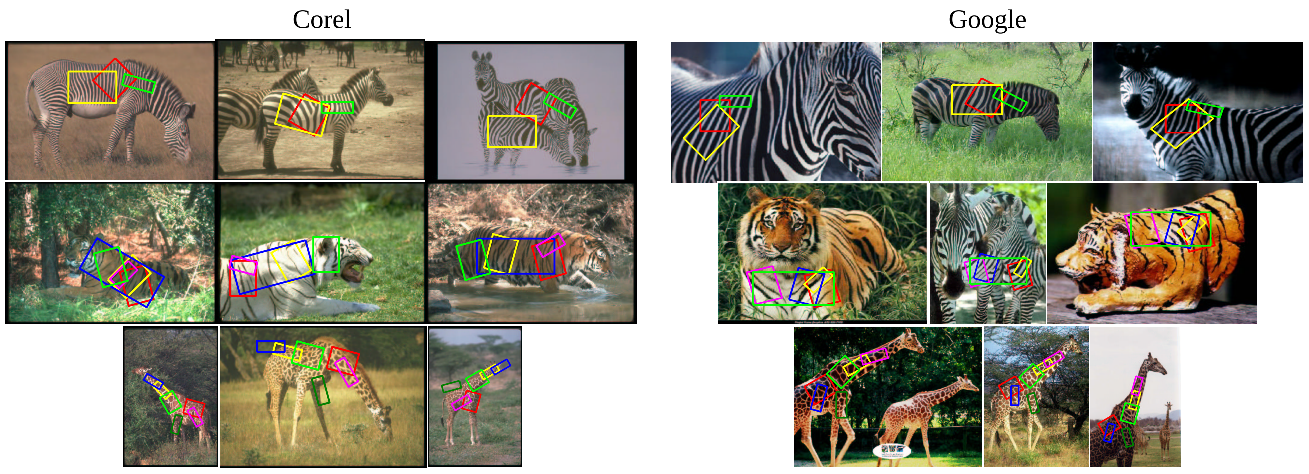


Figure 9: *Results for our zebra (***top row***), tiger (***middle row***), and giraffe (***bottom row***) models using shape and texture. We show the 3 best matches from a test pool of 304 Corel animal images and 1418 Google animal images. Our tiger model mistakenly fires on a Google zebra due to the 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.