# Tracking-Learning-Detection

Bryan Anenberg
Stanford University
anenberg@stanford.edu

Michela Meister
Stanford University
mmeister@stanford.edu

## 1. Introduction

Online tracking of an object in video is a difficult task because the objects appearance can change a lot across frames. For example, the object might undergo changes in illumination, rotation, or occlusion. A successful system must learn to adapt to these changes in order to continue tracking the object.

One way to perform tracking is tracking by detection, which uses a detector and an appearance model to find the object in each frame. The detector keeps both positive samples of the object and negative samples of the background so that it can tell the difference between the object and its surroundings. Its important that positive and negative examples are chosen carefully, because error in the appearance model can cause significant drift in tracking. Babenko et al suggest a method for tracking by detection via online multiple instance learning. Instead of labeling individual patches as positive or negative, multiple instance learning uses bags of patches sampled from around the object. These bags are then passed to an online learning algorithm which decides the probability that each individual patch contains the object. This allows for more flexibility in the sampling of positive and negative patches [1]. Hare et al sidesteps this process of sample labeling entirely by building a structured output support vector machine to predict the change in the objects appearance from one frame to another directly from the output of the tracker [5].

Multi-object tracking poses many other problems. For example, objects may occlude each other, or the path of one object may intersect with the path of another. Berclaz et al propose a graph-based solution to this problem, where each node of the graph represents a specific point in time and space and is connected via directed edges to the nodes directly preceding or following it in time. Thus one path from the source to the sink of the graph represents one trajectory through space and time. Furthermore, a maximum of one object can inhabit an individual spot at any one time, and so the maximum flow on each edge is 1. This system can be solved with linear programming to find the most probable set of object trajectories [2]. Pirsiavash et al improve upon this work by constructing a greedy, successive shortest paths algorithm to greatly reduce the running time of this graph-based solution [7].

## 2. Algorithm

Kalal et al propose a system that combines tracking and detection to follow an object. They use a Lucas-Kanade tracker that uses optical flow to follow the object between frames. The tracker works well if the object does not move much between consecutive frames but can fail if the object moves too quickly or goes out of frame. The detector searches for the object in every frame by comparing individual patches to the learned object model. These comparisons are done using fern and nearest neighbor classifiers. After each iteration, the tracker and detector return estimates of the objects location, and the integrator compares the confidences of these two estimates in order to choose a final bounding box. If the integrator has high confidence in this final bounding box, it also updates the object model by sampling a set of positive patches around the objects estimated location and a set of negative patches from the background of the image. This learning step helps the object model adapt to changes in the objects appearance from frame to frame [6].

## 3. Implementation Details

### 3.1. Learning and Detection

In theory the detector considers all potential bounding boxes in a frame. However, in practice it is too computationally expensive to compute the detector score on every bounding box. We reduce the number of frames under consideration by implementing two assumptions: that the tracked object stays approximately the same size between frames, and that the object moves a small amount between frames. We only consider bounding boxes whose width and height are within $10\%$ of the size of the previous bounding box. eqn. We also only consider bounding boxes whose centers are within $2 \cdot$ (mean of the previous bounding boxes height and width).
If after filtering the number of bounding boxes by the above two strategies there exists greater than $2000$ candidate

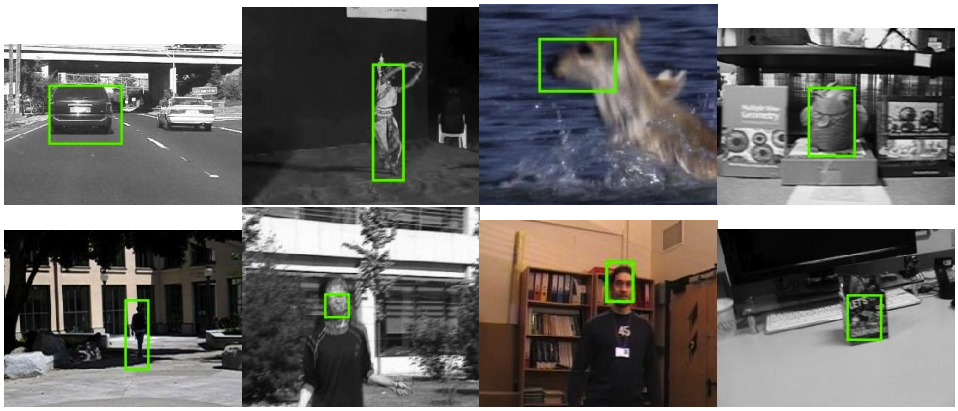

Figure 1: Frames from each from video the TLD system tracked. In order: Car4, Dancer2, Deer, Fish, Human8, Jumping, Man, and Vase. (Bolt2 is displayed in Figure 3)

bounding boxes, then we randomly sample 2000 of them.

The 20 bounding boxes with the largest classifier score (either fern or SVM) are chosen to compute the nearest neighbor confidence.

The detection model learns by updating its collection of positive and negative examples. Each example is a feature space representation of a patch. In this implementation, the model only stores 1000 positive and negative examples. For each update we randomly select positive and negative examples in the collection to replace with new positive and negative examples. However, we always retain the first 300 positive examples since we observe that the TLD tracker performs well tracking the object in the initial frames of the video.

### 3.2. Positive and Negative Sampling

The authors of the TLD [6] select 10 bounding boxes on the scanning grid which are closest to the current bounding box and then generate 10 warped version by geometric transformation (shift $\pm$ 1%, scale change $\pm$ 1%, in-plane-rotation $\pm 5^o$) and then add Gaussian noise ($\sigma = 5$). In our implementation, we also select the 10 bounding boxes closest to the current bounding box. However, since we assume that the first frame contains an accurate bounding box of the object, we generate 20 warped version instead of 10. The warped version are obtained by geometric transform geometric transformation (shift $\pm 2\%$, scale change $\pm 2\%$, in-plane-rotation $\pm 20^o$) and then Gaussian noise ($\sigma = 5$). For subsequent frames we only generate 10 warped versions with up to $10^o$ in-plane-rotation.

As in the paper, we sample 100 negative bounding boxes that overlap with the detected bounding box for the current frame by less than 20%.

### 3.3. Features Used

For each of the experiments we re-sized the bounding box patch to $25 \times 25$. For the baseline implementation, we centered the pixel intensity mean at 0 and unrolled the image pixels into a 625 dimensional feature vector. For later experiments we used the Histogram of Oriented Gradients (HOG) feature descriptor.

### 3.4. Classifier

The experiments use either the ensemble of fern classifier or the Support Vector Machine (SVM) with linear, quadratic, or radial basis function (rbf) kernel.

#### 3.4.1 Fern Ensemble

The standard TLD system was implemented with an ensemble of fern classifiers. A single fern is a hashing scheme to assign a vector to one of $2^S$ buckets, where $S$ is the fern dimension [3]. In particular, a fern computes an S-digit binary code for a vector by applying a series of $S$ binary comparisons of pixel intensities at random indices. The S-digit number is interpreted as the bucket index in the range $[0, 2^S - 1]$. Using the fern hashing technique we create histograms over the positive and negative examples in the training data set. At test time, we compute the fern hash $f(i) \in \{0, 2^S - 1\}$ which assigns a test vector to the $f(i)$ bucket for the $i$-th fern in the ensemble. The probability a test vector is labeled as positive is $\frac{1}{L'} \sum_{i=1}^{L} P(i)$ for $P(i) = \frac{p}{p+n}$ where $p$ is the number of counts in bucket $f(i)$ for the positive histogram for the $i$-th fern (similarly with $n$ and the negative histogram). $L'$ is the number of ferns whose $f(i)$ bucket in both the positive and negative histogram is not empty.

### 3.5. Integrator

As expressed in the paper [6], the integrator combines the bounding box of the tracker and the bounding boxes of the detector into a single bounding box output per frame. If neither the tracker nor the detector outputs a bounding box prediction, the integrator marks the object as missing. If the tracker doesn't output a bounding box, then the integrator defaults to the detector's prediction. The detector predicts $10$ bounding boxes with associated confidences. Each bounding box's confidence is output by the nearest neighbor classifier using the equation:

$$Conf = \frac{1-\max(nNCC)}{2-\max(nNCC)-\max(pNCC)}$$

where $nCC$ is the set of normalized correlation coefficients calculated between the provided bounding box's feature representation and the negative examples saved in the detection model. Similarly, $pNCC$ are the normalized correlation coefficients calculated with the positive examples saved in the detection model.

Rather than selecting a bounding box from this set of bounding boxes, we cluster the bounding boxes by spatial distance and confidence. We select the bounding box centroid from the clustered bounding boxes that has the largest confidence score and has greater than $4$ bounding boxes assigned to it. If no bounding box meets this criterion, we default to the bounding box centroid with the largest confidence.

If both the tracker and the detector outputs a bounding box prediction, then we compare the confidence of the tracker's bounding box with the clustered detector bounding boxes. In general if the integrator decides that the detector's predicted bounding box is a better choice than the tracker's prediction, then the TLD system should not update its internal model since this predicted bounding box for the current frame could be unreliable. For instance, if the detector could assign a high confidence to a bounding box spatially far away from the tracked object. If the integrator favors this prediction, then the TLD system could update its internal model with false negative and positive examples.

If the most confident centroid also has the most bounding boxes assigned to it and its confidence is greater than the confidence of the tracker's prediction, then we set it as the predicted bounding box for the frame. However, if this bounding box overlaps with the tracker's predicted bounding box by more than $60\%$, then we flag the frame as valid: TLD system should commence learning from this reliable frame. If the most confident centroid didn't have the most bounding boxes assigned to it, then the integrator adjusts its predicted bounding box by averaging the bounding boxes of the nearby detections.

| File | 10 | 50 | 100 |
|---|---|---|---|
| Bolt2 | 0.032 | 0.0293 | 0.041 |
| Car4 | 0.618 | 0.654 | 0.675 |
| Dancer2 | 0.717 | 0.684 | 0.734 |
| Deer | 0.607 | 0.544 | 0.618 |
| Fish | 0.324 | 0.643 | 0.726 |
| Human8 | 0.070 | 0.084 | 0.102 |
| Jumping | 0.652 | 0.629 | 0.677 |
| Man | 0.822 | 0.823 | 0.804 |
| Vase | 0.377 | 0.368 | 0.508 |

Table 1: Average overlap for 10, 50, and 100 fern ensembles. (13 fern dimensions)

## 4. Results

### 4.1. Fern classifier experiments

The baseline implementation of TLD uses an ensemble of random fern classifiers with the mean-subtracted pixel intensity features. As an experiment, we vary the number of fern dimensions and the number of ferns in the ensemble.

Table 1 reports the impact on the average overlap as we vary the number of ferns in the ensemble between $10$ and $100$ while fixing the fern dimensions at $13$. Increasing the number of ferns in the ensemble generally improves the performance. Each fern selects $13$ pairs of pixels to compare. By increasing the number ferns in the ensemble, the classifier becomes more robust to a poor random initialization of binary pairs. The probability predicted by each fern is averaged over the ensemble.

Table 2 illustrates that increasing the fern dimension (the number of binary comparisons) from $6$ to $13$ increases the average overlap for $7$ of the $9$ videos. Increasing the fern dimension could make the classifier less sensitive to small variations in the image that effects a subset of the binary fern comparisons.

Figure 2 displays the best configuration of settings (13-dim, 100-ensemble) when using the fern classifier.

### 4.2. SVM classifier experiments

As a second experiment, we decided to replace the fern classifier with an SVM. Table 3 displays the performance of the TLD system with the SVM classifier applying different choices of kernel. It is not clear by comparing the average overlap that a specific choice of kernel is optimal. For example, the RBF kernel perform significantly better on the Human8 video, but the linear kernel the best for the Vase video.

### 4.3. Histogram of Oriented Gradient features

As a third experiment, we represent each image patch Histogram of Oriented Gradient (HOG) features . HOG

| File | 6 | 13 |
|---|---|---|
| Bolt2 | 0.029 | 0.041 |
| Car4 | 0.612 | 0.675 |
| Dancer2 | 0.721 | 0.734 |
| Deer | 0.589 | 0.618 |
| Fish | 0.760 | 0.726 |
| Human8 | 0.0916 | 0.102 |
| Jumping | 0.571 | 0.677 |
| Man | 0.880 | 0.804 |
| Vase | 0.453 | 0.508 |

Table 2: Average overlap for ferns of 6 and 13 dimensions with ensemble size 100.

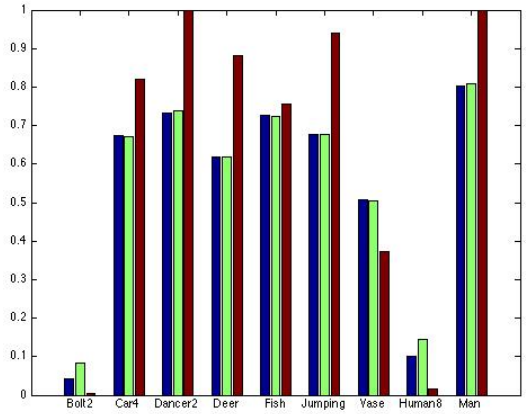


Figure 2: fern dimensions: 13, number of ferns: 100
The three evaluation metrics are displayed in the following order. Blue: average overlap, Green: AUC (area under the curve), Red: Mean Average Precision.

| File | Linear | Quadratic | RBF |
|---|---|---|---|
| Bolt2 | 0.068 | 0.079 | 0.081 |
| Car4 | 0.638 | 0.667 | 0.634 |
| Dancer2 | 0.777 | 0.772 | 0.677 |
| Deer | 0.593 | 0.607 | 0.673 |
| Fish | 0.528 | 0.366 | 0.492 |
| Human8 | 0.101 | 0.110 | 0.438 |
| Jumping | 0.144 | 0.727 | 0.573 |
| Man | 0.742 | 0.801 | 0.741 |
| Vase | 0.471 | 0.382 | 0.309 |

Table 3: Average overlap for Linear SVM, Quadratic SVM and RBF SVM.

features were originally introduced in the context of human detection [4]. HOG feature are capture the spatial distribution of intensity gradients and is invariant to geometric and color transformations. Thus, we expect HOG features to boost the tracking performance for videos featuring humans. Table 4 presents the average overlap performance of the TLD system using HOG features with SVM or fern classifiers. Although performance varies for different classifiers, we notice significant improvements in average overlap for videos with humans. For example, the linear SVM (Figure 4) and fern classifiers perform astonishingly well on the Bolt2 video (Figure 3 achieving average overlap of 54.5% for the SVM (38.1% for fern), AUC of 54.8% (39.5% for fern), and MAP of 61.2% (16.2% for fern). Previously when using mean-subtracted intensity pixels as features, the TLD system would incorrectly track the starting block of the runner, rather than the runner himself. The background pixels were granted the same amount of significance as the foreground runner pixels. The HOG descriptor takes advantage of the sharp intensity gradients between the homogeneous polyurethane track and the runner. The improvement for the Human8 detection illustrates that the HOG descriptor enables the tracker to follow the man as he walks from the sun through the shade. The gradients that outline the shape of the man are preserved despite the dramatic change in pixel intensity. In general the performance improves across all configurations for human tracking videos such as Dancer2, Human8, Jumping, and Man.

In some cases the performance did not improve. For example, the fern classifier tracks the Vase better using mean-subtracted intensity pixels as feature rather than HOG. This results could be the product of camera rotation in the video. HOG features are not invariant to rotation.

Figure 3: Using HOG features yields terrific tracking performance in the Bolt2 video.

| File | Linear | Quadratic | RBF | Fern |
|---|---|---|---|---|
| Bolt2 | 0.545 | 0.231 | 0.081 | 0.381 |
| Car4 | 0.611 | 0.657 | 0.634 | 0.678 |
| Dancer2 | 0.668 | 0.703 | 0.677 | 0.761 |
| Deer | 0.696 | 0.690 | 0.673 | 0.703 |
| Fish | 0.763 | 0.776 | 0.492 | 0.699 |
| Human8 | 0.224 | 0.766 | 0.438 | 0.282 |
| Jumping | 0.617 | 0.538 | 0.741 | 0.575 |
| Man | 0.689 | 0.763 | 0.309 | 0.690 |
| Vase | 0.344 | 0.383 | 0.573 | 0.251 |

Table 4: Average overlap for Linear SVM, Quadratic SVM, RBF SVM, and fern classifier (13 dim, 100 ferns) using HOG features.

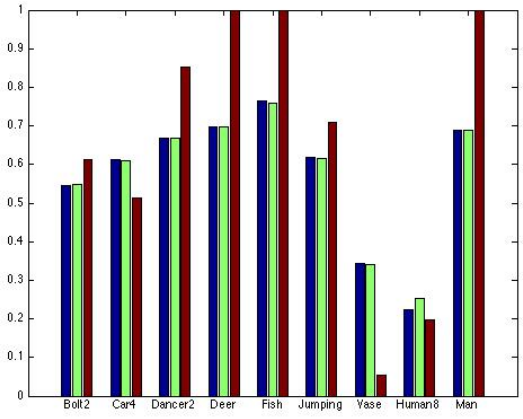


Figure 4: Linear SVM classifier + HOG pixels.

## References


[1] Boris Babenko, Ming hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. 2009.

[2] Jrme Berclaz, Franois Fleuret, Engin Tretken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. 2011.

[3] David Capel. Random forests and ferns. `http://vision.cse.psu.edu/seminars/talks/2009/random_tff/ForestsAndFernsTalk.pdf`.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893 vol. 1, June 2005.

[5] Sam Hare, Amir Saffari, and Philip H. S. Torr. Struck: Structured output tracking with kernels. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *ICCV*, pages 263–270. IEEE, 2011.

[6] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1409–1422, July 2012.

[7] Hamed Pirsiavash, Deva Ramanan, and Charless C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

# Appendices

The following are additional histograms to visualize the performance across all three evaluation metrics. Notice that in each of the following bar charts displays three evaluations measures in the following order. Blue: average overlap, Green: AUC (area under the curve), Red: Mean Average Precision.

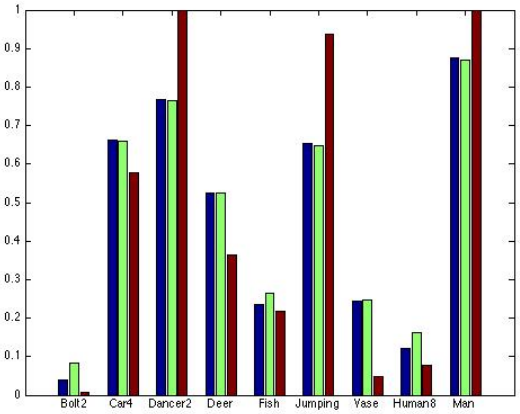


Figure 5: fern dimensions: 6, number of ferns: 20, mean-subtracted pixel features

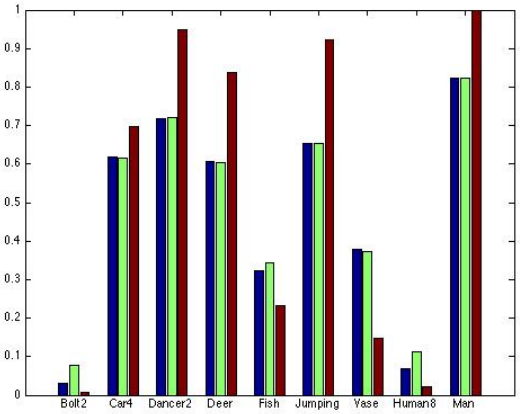


Figure 6: fern dimensions: 13, number of ferns: 10, mean-subtracted pixel features

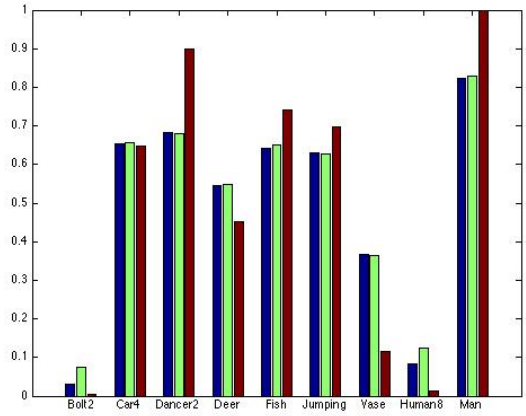


Figure 7: fern dimensions 13, number of ferns: 50, mean-subtracted pixel features

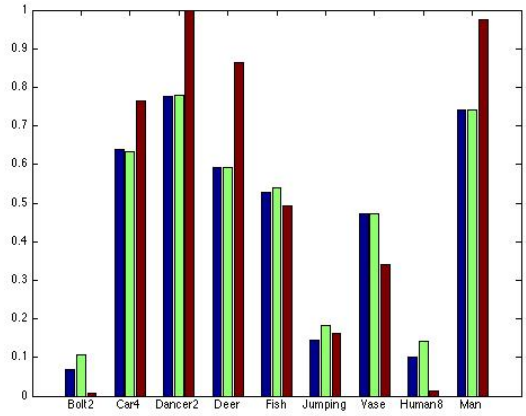


Figure 8: Linear kernel SVM classifier, mean-subtracted pixel features

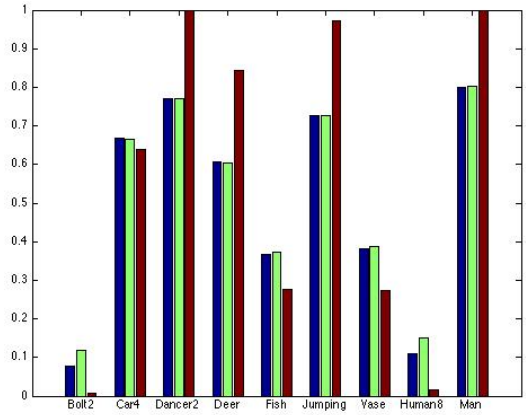


Figure 9: Quadratic kernel SVM classifier, mean-subtracted pixel features

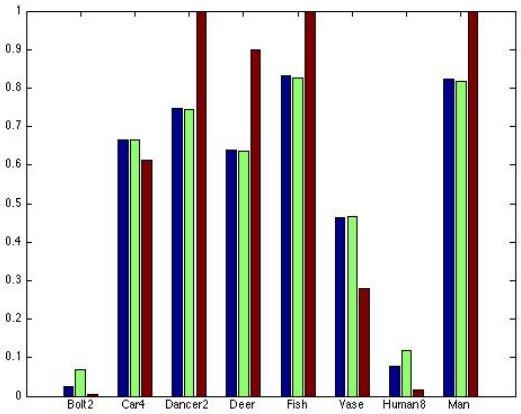


Figure 10: RBF (radial basis function) kernel, mean-subtracted pixel features

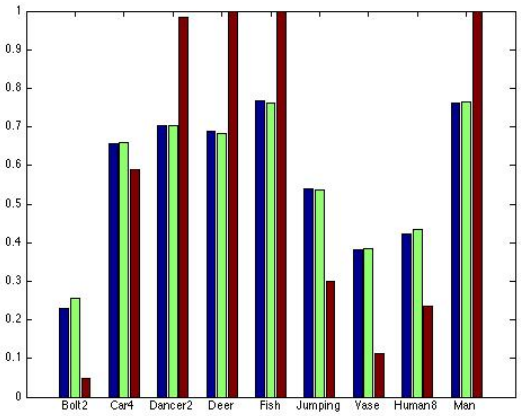


Figure 11: Quadratic kernel SVM, HOG features

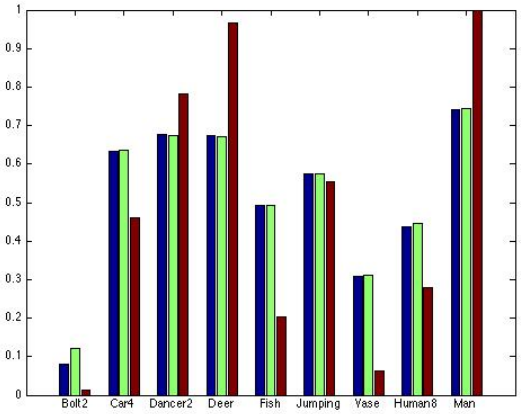


Figure 12: RBF kernel SVM, HOG features