Algorithms and Representations for Visual Recognition

Subhransu Maji

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-53
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-53.html

May 1, 2012

Algorithms and Representations for Visual Recognition

by

Subhransu Maji

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science
and the Designated Emphasis
in
Communication, Computation, and Statistics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jitendra Malik, Chair
Professor Trevor Darrell
Professor Bruno Olshausen

Fall 2011

Algorithms and Representations for Visual Recognition

Abstract

Algorithms and Representations for Visual Recognition

by

Subhransu Maji

Doctor of Philosophy in Computer Science
and the Designated Emphasis
in
Communication, Computation, and Statistics

University of California, Berkeley

Professor Jitendra Malik, Chair

We address various issues in learning and representation of visual object categories. A key component of many state of the art object detection and image recognition systems, is the image classifier. We first show that a large number of classifiers used in computer vision that are based on comparison of histograms of low level features, are "additive", and propose algorithms that enable training and evaluation of additive classifiers that offer better tradeoffs between accuracy, runtime memory and time complexity than previous algorithms. Our analysis speeds up the training and evaluation of several state of the art object detection, and image classification methods by several orders of magnitude.

Many successful object detection algorithms localize an object by simply evaluating a classifier at multiple locations and scales in an image, and finding peaks in the classifier response. In this setting, the overall speed of the detector can be improved not only by improving the efficiency of the classifier, which we addressed earlier, but also by efficient search, which we address next. We develop a discriminative voting algorithm based on Hough transform, which cuts down the complexity of this search.

In the last part of the thesis, we propose a representation for fine scale category recognition such as, action and pose of people in images, which is aided by more supervision. Leveraging on "crowdsourcing", we collect annotations of various kinds – keypoints, segmentations, attribute labels, pose, etc., for several tens of thousands of objects. The problem of comparing two instances visually can then be replaced by a simpler problem of comparing their annotations. The similarity function over the annotations provides us a flexible notion of correspondence between instances of a visual category, which we use to learn appearance models relevant to the task. We apply this framework to build a system for action recognition, that captures salient pose, appearance and interactions with objects, of people performing various actions in static images.

To mom, dad and sister.

Contents

Acknowledgments

Graduate school has been an incredible journey. There are many people I am indebted to for this wonderful experience. Thanks Jitendra Malik, for being a great advisor, an inspiration and for all the excitement about research. Thanks Michael Jordan, Dan Klein, Jitendra Malik and Richard Karp for all the wonderful courses. Thanks Trevor Darrell for a little bit of machine learning and practical advise. Thanks Ruzena Bajcsy for the big picture and for being so kind.

Thanks to all the members of the computer vision group without which the research would have been all but boring. Thanks Alex Berg and Michael Maire for showing the way. Thanks Patrik Sundberg for the all the energy and enthusiasm about everything. Thanks Chunhui Gu, Lubomir Bourdev, Pablo Arbelaez, Jon Barron, Alle Yang, Thomas Brox and Cees Snoek for all the discussions, debates and code. Thanks Bharath Hariharan, Georgia Gkioxari and Saurabh Gupta for all the new excitement.

Berkeley would not have been such a great place were it not for the outdoors, food, ambience and all the people to share that with. Thanks Gayane for everything. Thanks Blaine Nelson, Ajith Warrier and everyone else for all the volleyball outside SODA. Thanks Rhishikesh Limaye for all the biking and hiking around Berkeley. Thanks Siddharth, Narayanan, Sushmit, Sudeep, Pallavi, Vivek, Pannag, Nandini and all my friends at Berkeley for the countless times and things. Thanks to all my “wingmates” for the wonderful times in India, and everywhere else we met.

Finally all this would not have been possible without the support of my family - my parents who have made innumerable sacrifices though out their lives for me, my sister for all her support and enthusiasm, my grandparents for all the love and care. Thank you very much. To everyone else who has helped me on my journey, I am very grateful.

Chapter 1

Introduction

This generation of computer vision researchers are facing a new problem – there is simply too much training data, and many of our algorithms do not scale to the sizes of datasets one could collect. This has come about in the last decade due to a variety of reasons. Proliferation of cheap sensors, combined with the growth of public repositories of images, such as Flickr and Picasa, has left billions of images at the disposal of computer vision researchers. Often these come with user generated tags, or can be associated with search terms with the help of search engines like Google, Yahoo!, MSN, etc., providing training data for millions of visual categories at an unprecedented scale. Add to that the emergence of economical "crowdsourcing" services like Amazon Mechanical Turk, which enable scalable and cheap collection of vast amounts of highly accurate supervised data. A result of all this is that large datasets like ImageNet, containing millions of images for hundreds of thousands of categories, and PASCAL VOC datasets, containing few thousand objects of dozens of categories, are becoming the norm for benchmarking computer vision algorithms. We are forced to think about representations which generalize across categories and enable sub-ordinate categorization and learning algorithms which are efficient during training and testing.

In this work we address some of the challenges in learning and representation when dealing with large datasets, where algorithms that are super-linear are too expensive. Linear classifiers have been popular in this setting because of their efficiency during training and testing, but are often inferior in terms of accuracy when compared to their non-linear counterparts. We show that a class of widely used classifiers in computer vision based on non-linear kernel Support Vector Machines (SVMs), are actually quite efficient. These class of kernels called "additive" kernels, often arise when comparing images based on histograms of their low-level features. Our analysis shows that these classifiers have the same run time memory and time complexity as linear SVMs during both training and testing, saving many orders of magnitude over standard implementations, making them practical for large scale classification or even real-time detection tasks.

Crowdsourcing has become a practical way of collecting large amounts of annotated data for various computer vision tasks with the emergence of efficient market places for completing ”micro-tasks” performed by humans, like Amazon Mechanical Turk (AMT). In the second part of my thesis, we propose a method to “bootstrap” hard computer vision problems by aiding the learning algorithms with supervision. We build rich representations and a learning framework which enable image understanding at multiple levels – categories, sub-categories, attributes, segmentation, pose, etc. We demonstrate how this representation can be used for the challenging task of estimating the pose and actions of people in images.

1.1 Outline

Chapters 2, 3 and 4 address various bottlenecks in building an object detector. We focus on variants of sliding window object detectors, which include many of the current state of the art detection systems. These detectors are based on an image classifier being evaluated by varying the location, scale and aspect ratio of the classification window in an image, hence the name. In Chapter 2, we provide an efficient algorithm which speeds up the evaluation of many non-linear kernel SVM based classifiers by various orders of magnitude making them practical for detection tasks. We then show in Chapter 3, that these classifiers can also be trained efficiently, motivated by the analysis in the previous chapter. In Chapter 4, we address the complexity of search over pose using a variant of Hough transformation and propose a discriminative spatial feature selection algorithm to improve the overall accuracy and efficiency of detection. These methods have been widely adopted and have become essential ingredients of various state of the art detection and classification algorithms.

Moving beyond detecting rigid objects such as faces and pedestrians, in Chapter 5, we show how with appropriate supervision and learning algorithms, one can build rich category models. Arguably, the fundamental problem in building visual category models is the notion of correspondence between instances. We bootstrap this problem by annotating instances with various attributes – keypoints, 3D pose, segmentation mask, action labels, etc., and replacing the problem of visual correspondence by a simpler problem of comparing their annotations. This provides us a flexible notion of matching which can then be used to learn task specific appearance models. Our approach based on a novel part based representation called “poselets” can be used not only for detection, but also to infer the segmentation, pose, action and other attributes of people in images, which is a highly visually diverse category. Finally, in Chapter 6, we describe our experience in collecting over $250,000$ of the above annotations on Amazon Mechanical Turk which has enabled this line of research.

Chapter 2

Evaluation of Additive Kernel SVMs

Consider sliding window detection, one of the leading approaches for detecting objects in images like faces [78, 112], pedestrians [78, 21, 33] and cars [80]. In this approach, first, a classifier is trained to recognize an object at a fixed “pose” - for example, as shown in Figure 2.1, one may train a classifier to classify 64×96 pixel pedestrians which are all centered and scaled to the same size, from background. In order to detect pedestrians at arbitrary location and scale in an image, the classifier is evaluated by varying the location and scale of the classification window. Finally, detections are obtained by finding peaks of the classification score over scales and locations, a step commonly referred to as non-maximum suppression. Although this approach is simple – the classifier does not have to deal with invariance – a key drawback of this approach is computational complexity. On typical images these classifiers can be evaluated several tens of thousands of times. One may also want to search over aspect ratios, viewpoints, etc., compounding the problem. Therefore efficient classifiers are crucial for effective detectors.

Discriminative classifiers based on Support Vector Machines (SVMs) and variants of boosted decision trees are two of the leading techniques used in vision tasks ranging from object detection [78, 112, 21, 33], multi-category object recognition in Caltech-101 [46, 61], to texture discrimination [123]. Classifiers based on boosted decision trees such as [112], have faster classification speed, but are significantly slower to train. Furthermore, the complexity of training can grow exponentially with the number of classes [107]. On the other hand, given the right feature space, SVMs can be more efficient during training. Part of the appeal of SVMs is that, non-linear decision boundaries can be learnt using the “kernel trick” [94]. However, the run-time complexity of a non-linear SVM classifier can be significantly higher than a linear SVM. Thus, linear kernel SVMs have become popular for real-time applications as they enjoy both faster training and faster classification, with significantly less memory requirements than non-linear kernels.

Although linear SVMs are popular for efficiency reasons, several non-linear kernels are used in computer vision as they provide better accuracy. Some of the most popular ones are based on comparing histograms of low level features like color and texture computed over the image and using a kernel derived from histogram intersection or chi squared distance to train a SVM classifier. In order to evaluate the classification function, a test histogram is compared to a histogram for

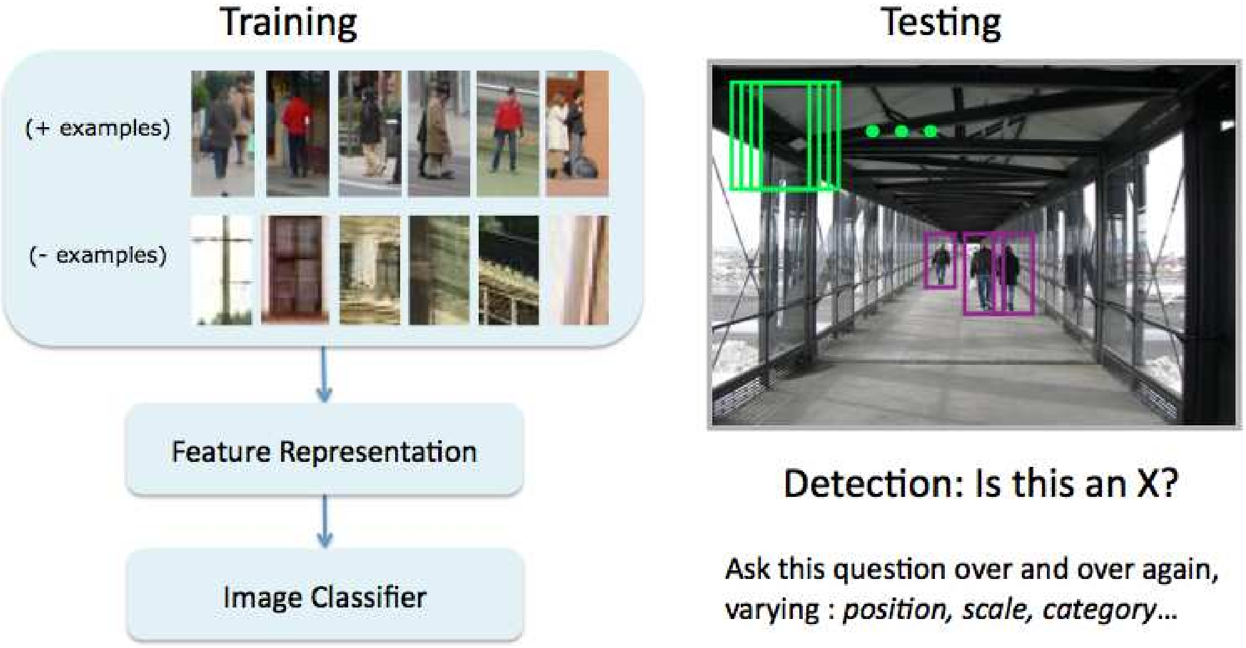


Figure 2.1: A typical "sliding window" detection pipeline.

each of the support vectors. The number of support vectors can often be a significant fraction of the training data, so this step is computationally very expensive as the test time scales linearly with the number of support vectors. This paper presents and analyzes a technique to greatly speed up that process for histogram comparison functions that are *additive* - where the comparison is a linear combination of functions of each coordinate of the histogram. **In particular we show it is possible to evaluate the classifier approximately in time independent of the number of support vectors – similar to that of a linear SVM.**

This more efficient approach makes SVMs with additive kernels – used in many of the current most successful object detection/recognition algorithms – efficient enough to apply much more broadly, even possibly to real-time applications. The class of kernels includes the pyramid matching or *intersection kernels* used in Grauman & Darell [46] ; and Lazebnik, Schmid & Ponce [61], and the chi squared kernel used by Varma & Ray [109]; and Chum & Zisserman [18], which together represent some of the best results in image and object recognition on the Caltech [30] and PASCAL VOC [28] datasets.

Although the results in this paper apply to any additive kernel, we begin by analyzing the *histogram intersection* kernel, $K_{\min}(h_a, h_b) = \sum_i \min\left(h_a(i), h_b(i)\right)$, that is often used as a measurement of similarity between histograms h_a and h_b. Because it is positive definite [104] for non-negative features and conditionally positive definite for arbitrary features [65], it can be used as a kernel for discriminative classification using SVMs. Recently, intersection kernel SVMs (henceforth referred to as IKSVMs), have become popular with the introduction of pyramid match kernel [46] and spatial pyramid match kernel [61] for object detection and image classification. Unfortunately this success typically comes at great computational expense compared to simpler linear SVMs, because non-linear kernels require memory and computation linearly proportional to the number of support vectors for classification.

In this chapter we show the following:

- SVMs using the histogram intersection kernel can be *exactly* evaluated exponentially faster than the straight forward implementation used in the previous state of the art, as has been previously shown in [51], and independently in our own work in [66] (Section 2.2).
- A generalization allows *arbitrary* additive kernel SVMs to be evaluated with the same "big O" computational cost, as linear SVMs (Section 2.3), as well as significantly reducing the memory overhead, making them practical for detection and real time applications.
- We show that *additive* kernels arise naturally in many computer vision applications (Section 2.4), and are already being used in many state of the art recognition systems.
- Additive kernels, such as histogram intersection are sufficiently general, i.e., the corresponding kernel SVM classifier, can represent arbitrary additive classifiers. The difference between additive kernels can be analyzed mainly in terms of the implied regularization for a particular kernel. This helps us to understand both the potential benefit and the inherent limitations of *any* additive classifier, in addition to shedding some light on the trade-offs between choices of additive kernels for SVMs (Section 2.5).
- Our approach can be computationally more efficient compared to some of the recently proposed methods and the previous state of the art in kernel classifier evaluation (Section 2.6).
- Combining these efficient additive classifiers with a novel descriptor provides an improvement over the state of the art linear classifiers for pedestrian detection, as well for many other datasets (Section 4.4).
- These techniques can be applied generally to settings where evaluation of weighted additive kernels is required, including kernel PCA, kernel LDA, and kernelized regression and kernelized k-means. (Section 2.8).

2.1 Support Vector Machines

We begin with a review of support vector machines for classification. Given labeled training data of the form $\{(y_i, \mathbf{x}_i)\}_{i=1}^{N}$, with $y_i \in \{-1, +1\}$, $\mathbf{x}_i \in \mathbb{R}^n$, we use a C-SVM formulation [20]. For the linear case, the algorithm finds a hyperplane which best separates the data by minimizing :

$$\tau(\mathbf{w}, \xi) = \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=i}^{N} \xi_i \tag{2.1}$$

subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$, where $C > 0$, is the tradeoff between regularization and constraint violation. For a kernel on data points, $K(\mathbf{x}, \mathbf{z}) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, that is the inner product, $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z})$, in an unrealized, possibly high dimensional, feature space, one can obtain the same by maximizing the dual formulation :

$$W(\alpha) \;=\; \sum_{i=i}^{N} \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.2}$$

$$\textit{subject to:} \qquad 0 \leq \alpha_i \leq C \text{ and } \sum \alpha_i y_i = 0 \tag{2.3}$$

The decision function is $sign\left(h(\mathbf{x})\right)$, where:

$$h(\mathbf{x}) = \sum_{l=1}^{m} \alpha_l y_l k(\mathbf{x}, \mathbf{x}_l) + b \tag{2.4}$$

Notice that the dual formulation only requires access to the kernel function and not the features $\Phi(.)$, allowing one to solve the formulation in very high dimensional feature spaces efficiently – also called the *kernel trick*. For clarity, in a slight abuse of notation, the features, $\mathbf{x}_l : l \in \{1, 2, \ldots, m\}$, will be referred to as support vectors. Thus in general, m kernel computations are needed to classify a point with a kernelized SVM and all m support vector must be stored. Assuming these kernels can be computed in $\mathcal{O}(n)$ time, the overall complexity of the classifier is $\mathcal{O}(mn)$. For linear kernels we can do better because, $k(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z}$, so $h(\mathbf{x})$ can be written as $h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, where $\mathbf{w} = \sum_{l=1}^{m} \alpha_l y_l \mathbf{x}_l$. As a result, classifying with a linear SVM only requires $\mathcal{O}(n)$ operations, and $\mathcal{O}(n)$ memory.

2.2 Fast Exact IKSVMs

We motivate our discussion using the histogram intersection or the $\min$ kernel. Often similarity between images is obtained by comparing their distribution over low level features like edge orientations, pixel color values, codebook entries, etc. These distributions could be represented as histograms and a similarity measure like the histogram intersection can be used. The histogram intersection kernel is known to be positive definite [104] for histogram based features and hence can be used with the standard SVM machinery. This representation is popular for the "bag-of-words" approaches which have led to state of the art results in many object detection and classification tasks.

We first show that it is possible to speed up classification for intersection kernel SVMs (IKSVMs). This analysis was first presented in [51] and later independently in our own work [66]. For histogram based feature vectors $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n_+$, the intersection kernel $K_{\min}(\mathbf{x}, \mathbf{z})$ is defined as:

$$K_{\min}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} \min\left(x_i, z_i\right) \tag{2.5}$$

and classification is based on evaluating:

$$\begin{aligned} h(\mathbf{z}) &= \sum_{l=1}^{m} \alpha_l y_l K_{\min}(\mathbf{z}, \mathbf{x}_l) + b && (2.6) \\ &= \sum_{l=1}^{m} \alpha_l y_l \left(\sum_{i=1}^{n} \min\left(z_i, x_{l,i}\right) \right) + b && (2.7) \end{aligned}$$

The non linearity of $\min$ prevents us from "collapsing" the weight vector in a similar manner for linear kernels. Thus the complexity of evaluating $h(\mathbf{x})$ in the standard way is $\mathcal{O}(mn)$. The key property of intersection kernels is that we can exchange the summations in equation 2.7 to obtain:

$$\begin{aligned} h(\mathbf{z}) &= \sum_{l=1}^{m} \alpha_l y_l \left(\sum_{i=1}^{n} \min\left(z_i, x_{l,i}\right) \right) + b && (2.8) \\ &= \sum_{i=1}^{n} \left(\sum_{l=1}^{m} \alpha_l y_l \min\left(z_i, x_{l,i}\right) \right) + b && (2.9) \\ &= \sum_{i=1}^{n} h_i(z_i) + b && (2.10) \end{aligned}$$

Thus the overall function $h(\cdot)$ can be rewritten as the sum of one dimensional functions $h_i(\cdot)$, where :

$$h_i(s) = \sum_{l=1}^{m} \alpha_l y_l \min\left(s, x_{l,i}\right) \qquad (2.11)$$

The complexity of computing each $h_i(s)$ in the naive way is still $\mathcal{O}(m)$ with an overall complexity of computing $h(\mathbf{x})$ still $\mathcal{O}(mn)$. We now show how to compute each h_i in $\mathcal{O}(\log m)$ time.

Consider the functions $h_i(s)$ for a fixed value of i. Let $\bar{x}_{l,i}$ denote the sorted values of $x_{l,i}$ in increasing order with corresponding α's and labels as $\bar{\alpha}_l$ and $\bar{y}_l$. If $s < \bar{x}_{1,i}$ then $h_i(s) = s \sum_l \bar{\alpha}_l = 0$, since $\sum_l \bar{\alpha}_l = 0$. Otherwise let r be the largest integer such that $\bar{x}_{r,i} \leq s$. Then we have,

$$\begin{aligned} h_i(s) &= \sum_{l=1}^{m} \bar{\alpha}_l \bar{y}_l \min\left(s, \bar{x}_{l,i}\right) && (2.12) \\ &= \sum_{1 \leq l \leq r} \bar{\alpha}_l \bar{y}_l \bar{x}_{l,i} + s \sum_{r < l \leq m} \bar{\alpha}_l \bar{y}_l && (2.13) \\ &= A_i(r) + s B_i(r) && (2.14) \end{aligned}$$

Where we have defined,

$$\begin{aligned} A_i(r) &= \sum_{1 \leq l \leq r} \bar{\alpha}_l \bar{y}_l \bar{x}_{l,i}, && (2.15) \\ B_i(r) &= \sum_{r < l \leq m} \bar{\alpha}_l \bar{y}_l && (2.16) \end{aligned}$$

Equation 2.14 shows that h_i is piecewise linear. Furthermore h_i is continuous because:

$$\begin{aligned} h_i(\bar{x}_{r+1}) &= A_i(r) + \bar{x}_{r+1} B_i(r) \\ &= A_i(r+1) + \bar{x}_{r+1} B_i(r+1). \end{aligned}$$

Notice that the functions A_i and B_i are independent of the input data and depend only on the support vectors and α. Thus, if we precompute them, then $h_i(s)$ can be computed by first finding r, the position of s in the sorted list $\bar{x}_{l,i}$ using binary search and linearly interpolating between $h_i(\bar{x}_r)$ and $h_i(\bar{x}_{r+1})$. This requires storing the $\bar{x}_l$ as well as the $h_i(\bar{x}_l)$ or twice the storage of the standard implementation. **Thus the runtime complexity of computing $h(\mathbf{x})$ is $\mathcal{O}(n \log m)$ as opposed to $\mathcal{O}(nm)$, a speed up of $\mathcal{O}(m/\log m)$.** This can be significant if the number of support vectors is large.

2.3 Approximate Additive Kernel SVMs

It is possible to compute approximate versions of the classifier even faster. Traditional function approximation quickly breaks down as the number of dimension increase. However for the intersection kernel SVMs we have shown that the final classifier can be represented as a sum of one dimensional functions. As long as the kernel is "additive", i.e., the overall kernel $K(\mathbf{x}, \mathbf{y})$ can be written as,

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} K_i(x_i, y_i) \tag{2.17}$$

the resulting kernel SVM classifier is also additive, i.e., $h(\mathbf{s})$ can be written as,

$$h(\mathbf{s}) = \sum_{i=1}^{n} h_i(s_i) + b \tag{2.18}$$

where,

$$h_i(s_i) = \sum_{l=1}^{m} \alpha_l y_l K_i(s_i, x_{l,i}) \tag{2.19}$$

and $x_{l,i}$ denotes the i'th dimension of the l'th support vector.

This decomposition allows us to approximate the final classifier by approximating each dimension independently. The simplest of these is a piecewise polynomial approximation in which we represent the function in each dimension as a piecewise polynomial function using b sections, each of degree k. This requires $b \times (k+1)$ floating points per dimension. Classification requires table lookup followed by the evaluation of a k degree polynomial, which requires $2(k+1)$ floating point operations using Euler's method. Two special cases are the piecewise constant and piecewise linear approximations corresponding to degree $k = 0$ and $k = 1$ respectively. In our experiments we restrict ourselves to these cases, as one can approximate any function arbitrary well. The final classifier corresponds to a lookup table of size $m \times (b+1)$. **The overall complexity of the classifier then is $\mathcal{O}\left(2(k+1)n\right)$ – essentially the same as that of a linear SVM classifier**.

In our MATLAB/C++ implementation the speed of the piecewise linear approximations is about $5.5\times$ slower than the linear classifier. The more expensive table lookups can be avoided by rewriting the piecewise linear interpolation as a dot product of a dense vector of function values and a sparse vector indicating the bin indices, for e.g. see [65] or [82]. These implementations are essentially as fast as the linear classification method, especially, when there are a large number of classes and the encoding time can be amortized over the number of classes.

Although these one dimensional functions can be precomputed once for each classifier – this could become a bottleneck if the number of classes are large, or if the classifier needs to be updated often for example during training. To approximate these one dimensional functions using a piecewise linear approximation, one has to sample these functions at a fixed set of points. The complexity of evaluating these one dimensional functions $h_i(s_i) = \sum_{l=1}^{m} \alpha_l y_l K_i(s_i, x_{l,i})$ at b locations is $\mathcal{O}(bm)$. When b is large, i.e. $b >> \log m$, for the intersection kernel one can sample these functions faster using the exact IKSVM evaluation presented in Section 2.2 in $\mathcal{O}((m+b)\log m)$ time, making it the approach of choice for certain applications.

2.4 Additive Kernels in Computer Vision

We identify several naturally arising additive kernels in computer vision applications, though we note that variants of these kernels also arise in natural language processing, such as text classification, etc. There are two important classes of additive kernels used in the computer vision, which we describe next.

2.4.1 Comparing Histograms

Often similarity between images is obtained by comparing their distribution over low level features like edge orientations, pixel color values, codebook entries, textures, etc. These distributions are typically represented as histograms and a similarity measure like the histogram intersection or the negative χ^2 or l_2 distance is used. Both the histogram intersection kernel [104], and the χ^2 kernels are known to be positive definite for histogram based features and hence can be used with the standard SVM machinery. See [9, 77] for a proof that the histogram intersection kernel and its variants are positive definite and [4] for a proof for the χ^2 kernel.

The histogram intersection kernel, $K_{\min}$, and the χ^2 kernel, K_{χ^2}, for normalized histograms are defined as follows:

$$K_{\min}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} \min\left(x_i, z_i\right), K_{\chi^2}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} \frac{2x_i z_i}{x_i + z_i} \tag{2.20}$$

Figure 2.2 visualizes these additive kernels. We also note that the intersection kernel is conditionally positive definite for all features and hence can be used with SVMs even when the features are not histograms, i.e. they need not be positive or normalized. For proof see paper [65]. A special case worth mentioning is the generalized histogram intersection kernel [9] defined by :

$$K(\mathbf{x},\mathbf{z}) = \sum_{i=1}^{n} \min\left(|x_i|^{\beta}, |z_i|^{\beta}\right) \tag{2.21}$$

This is known to be positive definite for all $\beta > 0$. Chappelle et al. [17] observe that this remapping of the histogram bin values by $x \rightarrow x^{\beta}$, improves the performance of linear kernel SVMs to become comparable to RBF kernels on an image classification task over the Corel Stock Photo Collection. Simply square-rooting the features with linear kernel, which is also called the Bhattacharyya kernel, has also shown to provide significant improvements, when used with “bag-of-words” style features, for various image classification and detection tasks [110, 82]. This representation also arises in text classification setting where the histograms represent counts of words in a document.

2.4.2 Approximate Correspondences

Another class of additive kernels are based on the matching sets of features between images. Two popular variants are the *pyramid match* and the *spatial pyramid match* kernels. We describe each of them briefly.

Pyramid Match Kernel. Introduced by Grauman and Darell [46, 44], who proposed a way to measure similarity between sets of features using partial correspondences between the elements in the sets. The similarity measure reduces to a weighted histogram intersection of features computed in a multi-resolution histogram pyramid, hence the name. This approach builds on Indyk and Thaper’s [54] approximation to matching costs using l_1 embeddings. An attractive feature of this method is that the matching has linear time complexity in the feature dimension, and naturally forms a Mercer kernel which enables it to be used with discriminative learning frameworks like kernel SVMs. This kernel has been used in various vision tasks like content-based-image-retrieval, pose estimation, unsupervised category discovery [45] and image classification. This kernel is additive because the overall kernel is simply a weighted histogram intersection.

Spatial Pyramid Match Kernel. Lazebnik, Schmid and Ponce [61] introduced a similarity based on approximate global geometric correspondence of local features of images. Instead of a global histogram of features one creates histograms of features over increasingly fine sub-regions of the image in a “spatial pyramid” representation. Like the pyramid match kernel, the spatial matching is now approximated by the weighted histogram intersection of the multi-resolution spatial pyramid. This remarkably simple and computationally efficient extension of an orderless bag-of-features has proved to be extremely useful, and has become a standard baseline for various tasks which require image to image similarity like object detection, image classification, pose estimation,

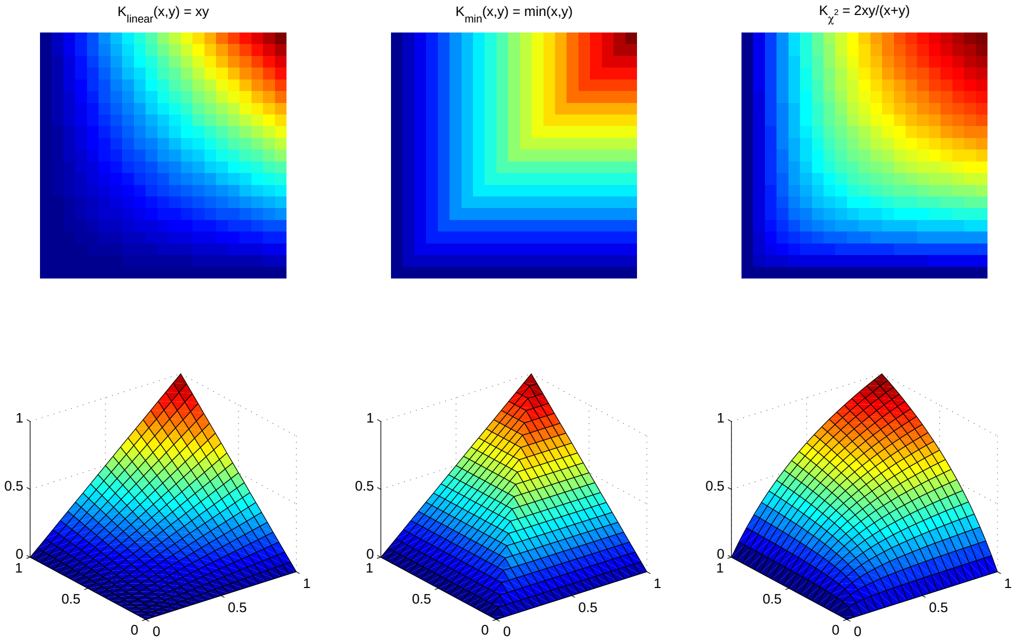


Figure 2.2: Visualization of linear, intersection and χ^2 kernels for one dimensional features. One can see that the χ^2 kernel is a smoother version of the intersection kernel and is twice differentiable on the interior.

action recognition, etc. Many state of the art object detection and image classification results on PASCAL Visual Object Challenge [28], ImageNet [55] and TRECVID [102] challenge are based on variants of the kernel where the underlying features change. Nevertheless this kernel is also additive as the overall kernel is once again a weighted histogram intersection of the underlying features.

2.5 Learning Additive Classifiers

Additive classifiers are based on functions of the form:

$$f(\mathbf{x}) = \sum_{i=1} f_i(x_i) \tag{2.22}$$

i.e., the overall function f is a sum of one dimensional functions. Additive functions were popularized by Hastie and Tibshirani [49], for fitting statistics of data. Linear classifiers are the simplest additive classifiers where each $f_i(x_i) = w_i x_i$. By allowing arbitrary f_i, additive models can provide better fits to the training data than linear models. Our key insight in Section 2.3, was to

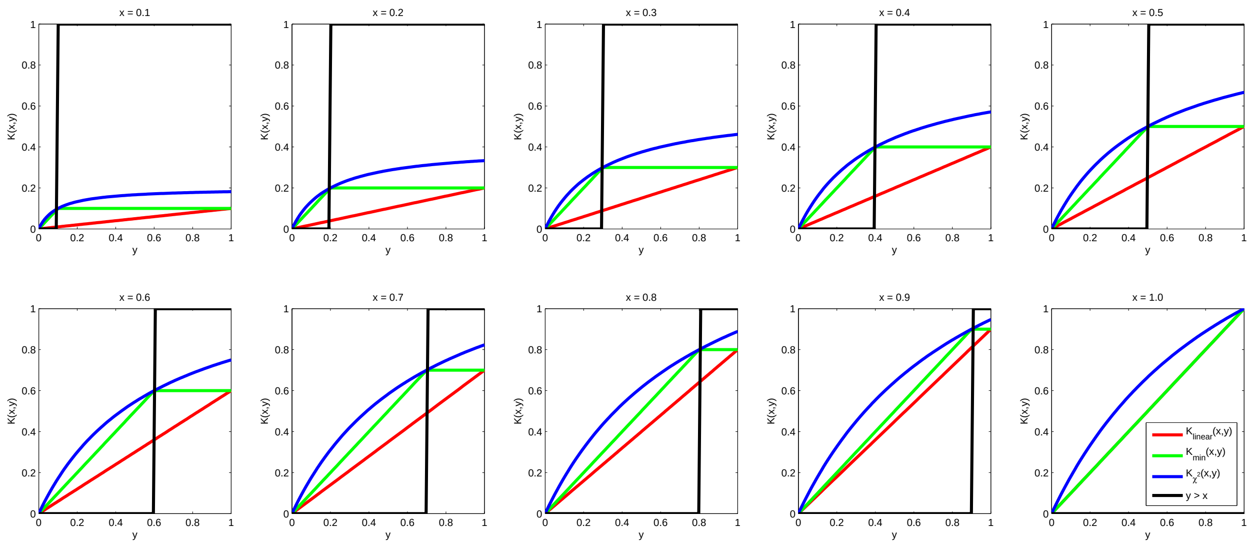


Figure 2.3: Visualization of the basis functions of linear, intersection, χ^2 kernels and decision stumps for one dimensional features.

observe that if the kernel K is additive, then the learned SVM classifier is also additive. Thus the standard SVM training machinery provides an efficient way to train additive classifiers compared to the traditional *backfitting* algorithm [40]. Additive classifiers also arise in boosting when the weak-learners are functions of one dimension, for example, decision stumps, $(x_i > c)$. Hence the standard AdaBoost algorithm [93], is yet another way of training additive classifiers.

We now show that the additive classifiers based on histogram intersection kernel are general, i.e., can represent any additive function on the input features as a linear combination of intersection kernel of the features as shown by the next theorem.

Theorem 2.5.1. *Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be points in $\mathbb{R}^d \geq 0$ and $f(\mathbf{x}_i) = f_1(x_{i,1}) + f_2(x_{i,2}) + \ldots + f_d(x_{i,d})$, be an additive function, where $x_{i,j}$ denotes the value of j'th dimension of the i'th point. Then there exists $\alpha_1, \alpha_2, \ldots, \alpha_n$ such that $f(\mathbf{x}_i) = \sum_j \alpha_j K_{\min}(\mathbf{x}_i, \mathbf{x}_j)$, $\forall i = 1, 2\ldots, n$.*

Proof. We prove this by showing that there exists a weight vector $\mathbf{w}$, in the Reproducing Kernel Hilbert Space of the intersection kernel, $K_{\min}$, such that $\mathbf{w} \cdot \phi(\mathbf{x}_i) = f(\mathbf{x}_i)$. First we show that there is a weight vector w_k, for each f_k, such that $w_k \cdot \phi(x_{j,k}) = f_k(x_{j,k})$. This follows immediately from the fact that the gram matrix G^k, consisting of entries $G^k_{ij} = \min(x_{i,k}, x_{j,k})$ is full rank for unique $x_{i,k}$, and the system of equations, $\alpha G^k = f^k$, has a solution (if the values are not unique, one can remove the repeated entries). Since the overall function is additive, we can obtain the weight vector $\mathbf{w}$ with the required property by stacking the weight vectors, w_k, from each dimension. Thus by representer theorem, there exists α such that $\mathbf{w} \cdot \phi(\mathbf{x}_i) = \sum_j \alpha_j K_{\min}(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i)$. □

Note that the α is shared across dimensions and this proof may be applied to any additive

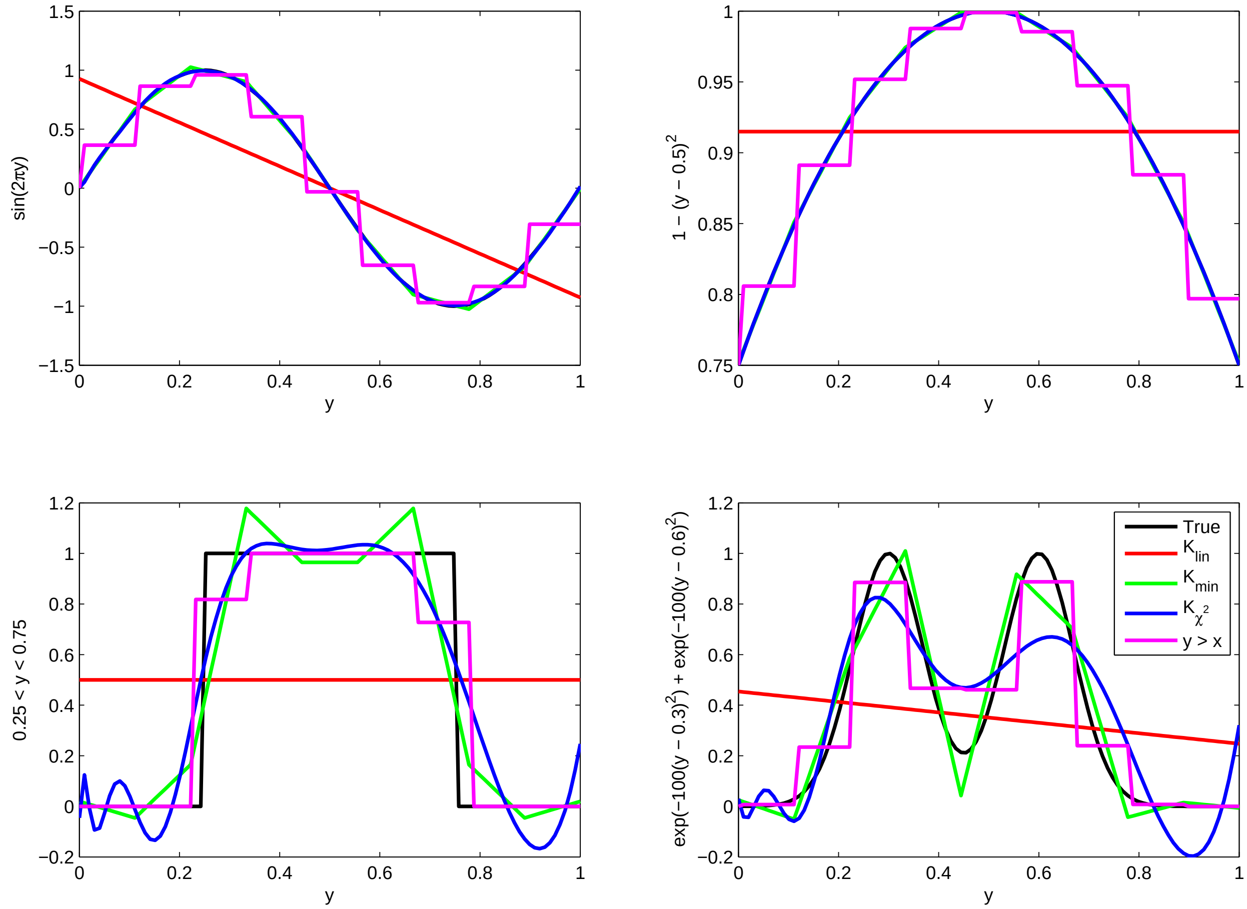


Figure 2.4: Approximations of various one dimensional functions by linear, intersection, χ^2 kernels and decision stumps as the basis functions.

kernel which satisfies the property that the kernel in each dimension is full rank, for example the χ^2 kernel.

Thus the SVM classifier represents the overall function as a linear combination of kernel functions in each dimension. The one dimensional functions $K_i(s_i, x_{l,i})$ for a fixed value of $x_{l,i}$ can be thought of as a basis function for each dimension of the classifier. Figure 2.3 shows these basis functions for the intersection and χ^2 kernels. Figure 2.4 shows several one dimensional functions approximated by a linear combination of 10 basis functions centered at $0.1, 0.2, \ldots 1.0$ and a constant. The linear combination coefficients were found using linear least-squares regression. The decision stumps $(x_i > c)$, gives us a piecewise constant approximation while the histogram intersection gives a piecewise linear approximation and χ^2 kernel gives smoother polynomial like approximation. Compared to linear case, kernels like the intersection, χ^2 kernel and decision stumps are able to approximate these functions much better.

2.6 Previous Work

There are several approaches for speeding up classification using kernel SVM classifiers, which we discuss briefly next.

Approximate Kernel SVMs For the histogram intersection kernel, Herbster [51] first proposed the fast evaluation algorithm we presented in Section 2.2. In our earlier work [66] we independently proposed the same method for exact classification along with the approximate method described in Section 2.3, which is more general and applies to arbitrary additive kernels. Recently, Rahimi and Recht [83], propose embeddings that approximate shift-invariant kernels, i.e., $K(\mathbf{x}, \mathbf{y}) = f(|\mathbf{x} - \mathbf{y}|)$, using a feature map Φ, such that $K(\mathbf{x}, \mathbf{y}) \sim \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$. Based on this analysis and our own work [66, 65], Vedaldi and Zisserman [110] propose embeddings which approximate a class of additive kernels that are “homogeneous”. This allows one to use the explicit form of the classifier $f(\mathbf{x}) = w \cdot \Phi(\mathbf{x})$, instead of the kernelized version, which can be more efficient in some settings. We discuss some of these methods in the next chapter.

However during classification for additive kernels, the piecewise linear approximation we proposed can be much faster. To see this observe that the piecewise linear approximation can be written as a dot product of a weight vector corresponding to the values of the function sampled at uniformly spaced points, with a sparse vector corresponding to the projection of the data on to a uniformly spaced linear B-Spline basis centered at these points (also see [65]). In this representation, evaluating the classifier requires only two multiplications and one addition per dimension, which can be much smaller compared to the approximate embeddings of [110].

Another line of approach applicable to Gaussian kernels is the work of Yang et al. [118] who use the fast Gauss transform to build efficient classifiers – however this is applicable when the feature dimension is very small, typically less than ten.

Reduced Set Methods. For general kernels, a class of methods, known as “reduced set methods”, approximate the classifier by constructing representations using a small subset of data points, typically much smaller than the number of support vectors. These set of points can be the set of input points themselves as in the work of [15, 79], where the most representative support vectors are kept as a post processing step. Instead of having a single approximation, one can have a series of approximations with more and more points to obtain a cascade of classifiers, an idea which has been used in [89] to build fast face detectors. Another class of methods build classifiers by having a regularizer in the optimization function which encourages sparseness, (eg. l_1-*norm* on the alphas) or pick support vectors in a greedy manner till a stopping criteria is met [57]. These methods may be able to reduce the number of support vectors by a order of magnitude, but are still significantly slower than a linear SVM. Often this come at the expense of classification accuracy. Thus, these approaches are not competitive when the kernel is additive compared to our approach.

Coarse to Fine methods. The coarse to fine approach for speeding up the classification is popular in many realtime vision applications. Simpler features and classifiers are used to reject easy examples quickly in a cascade. This idea has been applied to face [50, 10] and pedestrian detection [116] to achieve an order of magnitude speedup in the overall detection time. Methods like branch and bound [58], context [52], bottom-up regions [92], Hough transformation [69, 62], etc., improve efficiency by reducing the number of classifier evaluations. This paper improves the efficiency of the underlying discriminative classifier, allowing more powerful classifiers to be evaluated exponentially faster – in practice up to several thousand times faster than naive implementations and entirely complementary to the techniques mentioned for reducing the number of classifier evaluations.

2.7 Experimental Results

Since its introduction, our ideas for efficiently computing weighted combination for additive kernels has been applied to many applications like image-classification on Caltech-101 [31], PASCAL Visual Object Challenge [28], handwritten digits [68], video retrieval (TRECVID [102]), near-duplicate image detection [99], pedestrian detection frameworks combining static image features and optical flow [114], efficient classifiers for training large scale data [65, 110, 115], etc. We summarize some of these applications in Section 2.8.

We present experiments on several image classification and detection datasets and compare the performance of linear, intersection as well as a non-linear kernel, such as radial basis, or polynomial kernel. We also report the speedup obtained by the piecewise linear approximation compared to the naive method of evaluating the classifier. Table 2.1 contains a summary of our results. The piecewise linear approximations are as accurate as the exact additive classifier classifier with about 100 pieces on all datasets. **On various datasets the intersection kernel SVM is significantly better than the linear SVM and often comparable to rbf-kernel SVM, while offering up to three orders of magnitude speedup**. The details of each dataset and the features are presented below.

2.7.1 Toy Example : Learning a circle

We illustrate the additive kernel approximation using a toy example. The data is generated by sampling points from a two dimensional Gaussian and all points within a certain radius of the center belong to one class and the points outside belong to the other class as seen in Figure 2.5 (top-left).

A linear classifier works poorly in this case as no two dimensional line can separate the points well. However, the intersection kernel SVM is able to achieve an accuracy of 99.10% on this data. This is because it is able to approximate the circle which is an additive function ($x^2 + y^2 \leq r$), using two one-dimensional curves, x^2 and y^2. Figure 2.5 shows the learned classifier represented with varying number of bins using a piecewise linear approximation as well as the classification

Dataset	Linear SVM	IKSVM	Kernel SVM	Kernel Type
Toy Dataset (Raw features)				
Accuracy	51.9%	99.10%	99.50%	$rbf, \gamma = 0.5$
		22×		
MNIST Digits (OV-SPHOG)				
Error Rate	1.44%	0.77%	0.56%	$poly, d = 5$
		1200×		
USPS Digits (Raw Pixels)				
Error Rate	11.3%	8.7%	4.0%	$poly, d = 3$
		24×		
USPS Digits (OV-SPHOG)				
Error Rate	3.4%	3.4%	3.2%	$poly, d = 5$
		26×		
INRIA Pedestrian (SPHOG)				
Recall at 2 FPPI	43.12%	86.59%	-	
		2594×		
DC Pedestrians (SPHOG)				
Accuracy	72.19 ± 4.40%	89.03 ± 1.39%	88.13 ± 1.43%	$rbf, \gamma = 175$
		2253×		
Caltech 101 (SPHOG)				
Accuracy (15 examples)	38.79 ± 0.94%	50.10 ± 0.65%	44.27 ± 1.45%	$rbf, \gamma = 250$
		37×		
Accuracy (30 examples)	44.33 ± 1.33%	56.59 ± 0.77%	50.13 ± 1.19%	$rbf, \gamma = 250$
		62×		
UIUC Cars (SPHOG)				
Precision at EER	89.8%	98.5%	93.0%	$rbf, \gamma = 2.0$
		65×		

Table 2.1: Summary of our results. We show the performance using a linear, intersection and non-linear kernel as well as the speedup obtained by a piecewise linear approximation of the intersection kernel classifier on each dataset. The rbf kernel is defined as $K(\mathbf{x}, \mathbf{y}) = \exp\left(-\gamma(\mathbf{x} - \mathbf{y})^2\right)$ and the $poly$ kernel of degree d, is defined as $K(\mathbf{x}, \mathbf{y}) = (1 + \gamma(\mathbf{x} \cdot \mathbf{y}))^d$. All the kernel hyper-parameters were set using cross-validation.

accuracy as a function of the number of approximation bins. The accuracy saturates with 10 bins. On more realistic datasets the number of bins required for a good approximation depends on of smoothness of the underlying function, but empirically 100 bins were sufficient for a good approximation in all our experiments.

2.7.2 MNIST and USPS Digits

The MNIST dataset[1] was introduced by Yann LeCun and Corinna Cortes and contains $60,000$ examples of digits $0-9$ for training and $10,000$ examples for testing. As before we construct features based on histograms over oriented responses computed by convolving the image with a Gaussian derivative filter with $\sigma = 2$ and bin the response in 12 orientations. The images in this dataset are 28×28 pixels and we collect histograms over blocks of sizes 28×28, 14×14 , 7×7 and 4×4 pixels. We also found that adding overlapping blocks which overlap by half the block size improves performance at the expense of increasing the feature vector dimension by a factor of about four. This is similar in spirit of the overlapping blocks in the HOG descriptor in the pedestrian detector of [21]. These features with an IKSVM classifier achieves an error rate of 0.79%, compared to an error rate of 1.44% using linear and 0.56% using polynomial kernel. Similar features and IKSVM achieves an error rate of 3.4% on the much harder USPS dataset. We refer the readers to [68], for a complete set of experiments for the task of handwritten digit classification. Figure 2.6 shows the errors made by our digit recognition system on the MNIST dataset.

A key advantage is that the resulting IKSVM classifier is very fast. The estimated number of multiply-add operations required by the linear SVM is about 40K while the intersection kernel requires about 125K operations including the time to compute the features. This is significantly less than about 14 million operations required by a polynomial kernel SVM reported in the work of [23]. The reduced set methods [16](1.0% error) requires approximately 650K operations, while the neural network methods like LeNet5 (0.9% error) requires 350K and the boosted LeNet4 (0.7% error) requires 450K operations. For a small cost for computing features we are able to achieve competitive performance while at the same time are faster at both training and test time.

2.7.3 INRIA Pedestrians

The INRIA pedestrian dataset [21] was introduced as an alternate to the existing pedestrian datasets (eg. MIT Pedestrian Dataset) and is significantly harder because of wide variety of articulated poses, variable appearance/clothing, illumination changes and complex backgrounds. Linear kernel SVMs with Histograms of Oriented Gradients (HOG) features achieve high accuracy and speed on this dataset [21]. We use the multi-scale HOG features introduced in [66] and train a intersection kernel SVM on these features. The single scale HOG used in the original paper [21] when used with IKSVM provides small improvements over the linear kernel, similar to those observed

[1] http://yann.lecun.com/exdb/mnist/

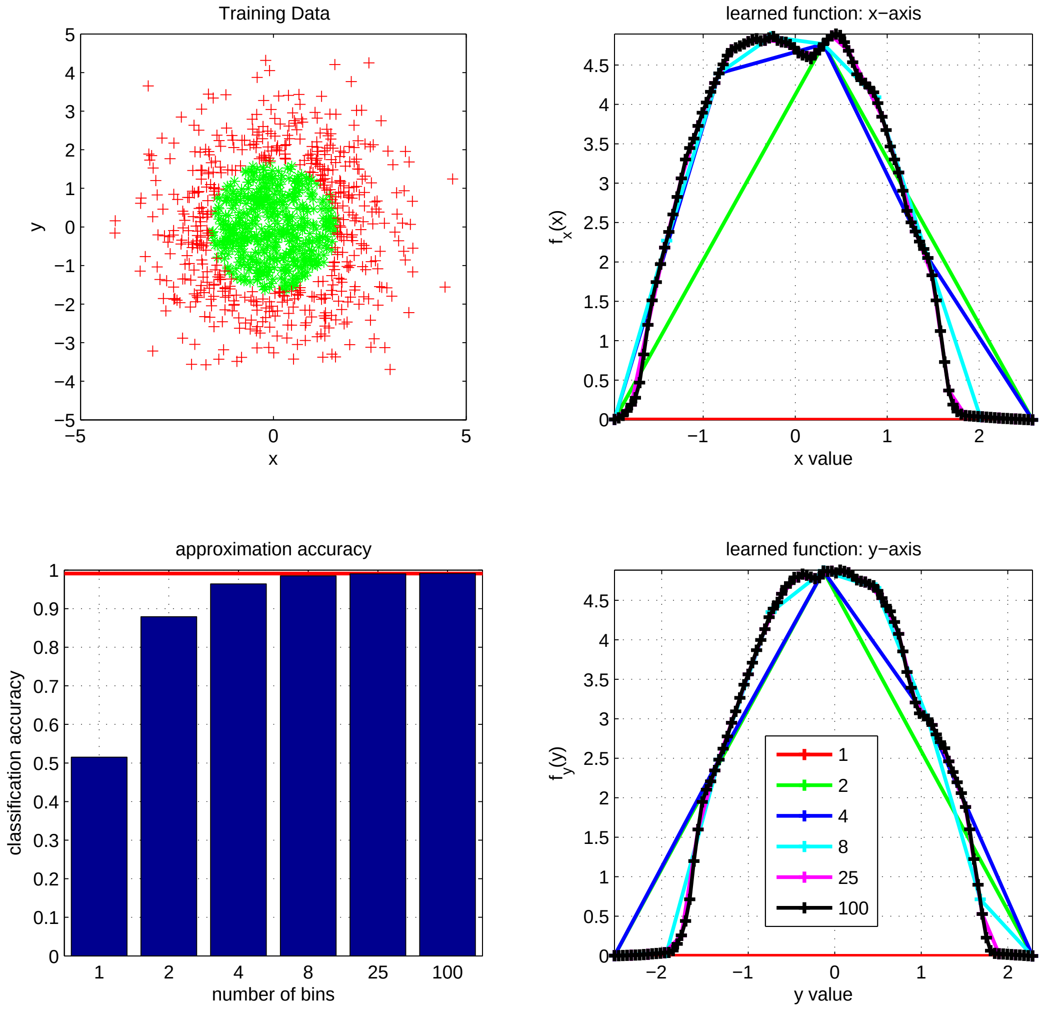


Figure 2.5: Toy example. (Top Left) The training data (Bottom Left) Accuracy of the learned classifier approximated by a piecewise linear function of varying number of bins. (Top Right and Top Left) Learned functions on x and y dimensions respectively as well as the piecewise linear approximations using a varying number of bins.

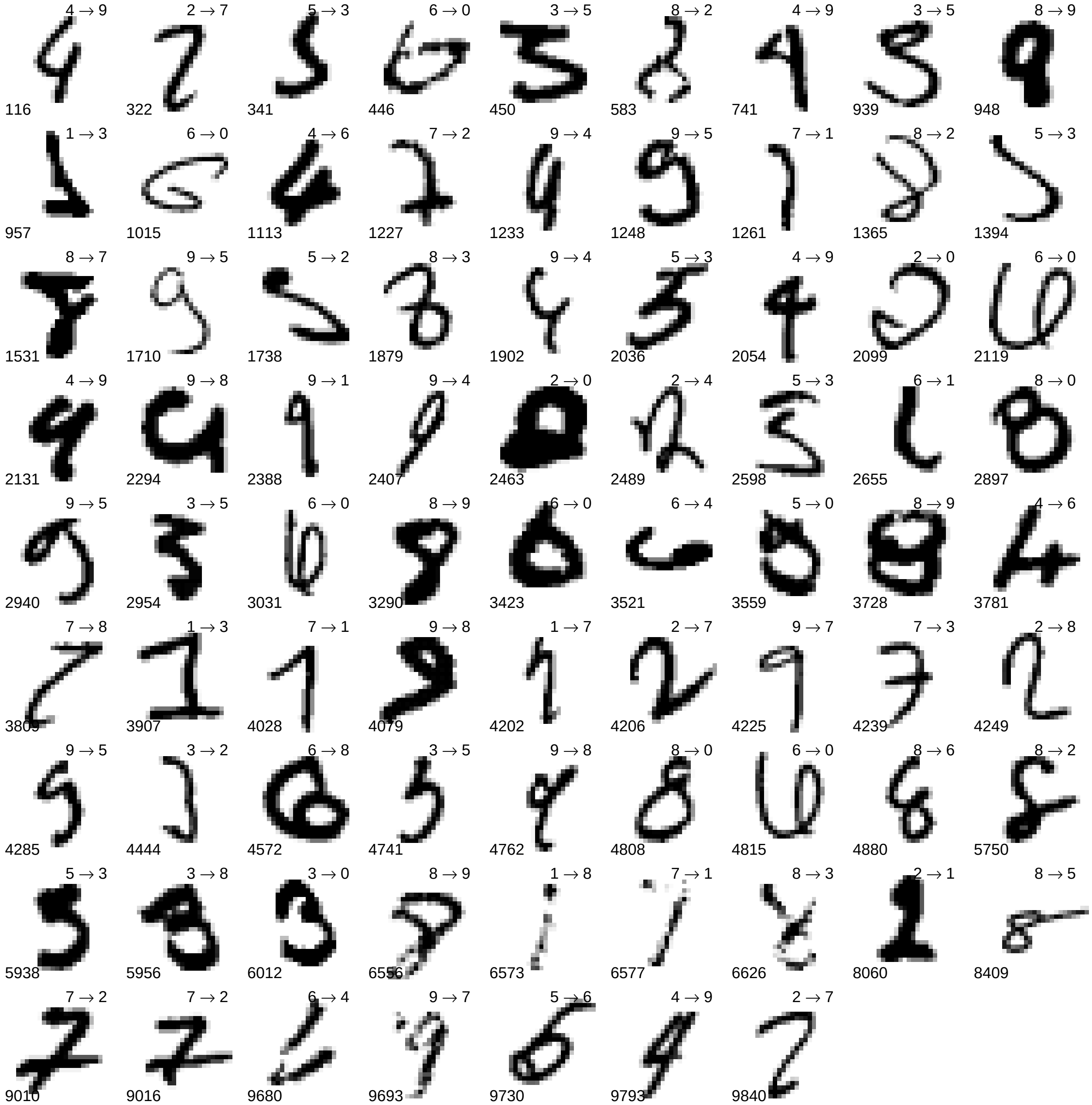


Figure 2.6: All the errors made by the classifier on the MNIST dataset. On each image $a \to b$ means that the digit a was misclassified as b.

Classification Method	**Detection Rate (2 FPPI)**	**Speedup**
Linear SVM	43.12 %	-
IKSVM (binary search)	86.59 %	$\mathbf{473}\times$
IKSVM (piecewise linear)	86.59 %	$\mathbf{2594}\times$
IKSVM (piecewise constant)	86.59 %	$\mathbf{3098}\times$
Dalal & Triggs [21]	79.63 %	-
Dalal & Triggs [21]*	82.51 %	-

Table 2.2: Detection rate at 2 FPPI on INRIA person dataset. The last run of [21] is obtained by running the detector using a finer “scaleratio” of 1.05 between successive layers of the image pyramid, instead of the default 1.1.

by using the rbf-kernel. We also found that the HOG with l_1-normalization of the gradient based features works better with the intersection kernel. The multi-scale HOG however outperforms l_1-normalized HOG. Results are shown in Table 2.2 using 100 bin approximation. Figure 2.7 shows sample detections on this dataset.

2.7.4 Daimler Chrysler Pedestrians

We use the Daimler Chrysler pedestrian benchmark dataset, created by Munder and Gavrila [74]. The dataset is split into five disjoint sets, three for training and two for testing. Each training set has 5000 positive and negative examples each, while each test set has 4900 positive and negative examples each. We report results by training on two out of three training sets at a time and testing on each of the test sets to obtain six train-test splits. Due to small size of the images (18×36), we only compute the multi-level features with only three levels ($L = 3$) of pyramid with cell sizes 18×18, 6×6 and 3×3 at levels 1, 2 and 3 respectively. The block normalization is done with a cell size of $w_n \times h_n = 18 \times 18$. The features at level l are weighted by a factor $c_l = 1/4^{(L-l)}$ to obtain a 656 dimensional vector, which is used to train an IKSVM classifier.

The classification results using the exact methods and approximations are shown in Table 2.3. Our results are comparable to the best results for this task [74]. The IKSVM classifier is comparable in accuracy to the rbf-kernel SVM, and significantly better than the linear SVM. The speedups obtained for this task are significant due to large number of support vectors in each classifier. The piecewise linear with 30 bins is about **2000**$\times$ faster and requires **200**$\times$ less memory, with no loss in classification accuracy. The piecewise constant approximation on the other hand requires about 100 bins for similar accuracies and is even faster.

Our unoptimized MATLAB implementation for computing the features takes about about $17ms$ per image and the time for classification ($0.02ms$) is negligible compared to this. Compared to the $250ms$ required by the cascaded SVM based classifiers of [74], our pipeline is $15\times$ faster. Figure 2.8 shows some of the errors made by our classifier.

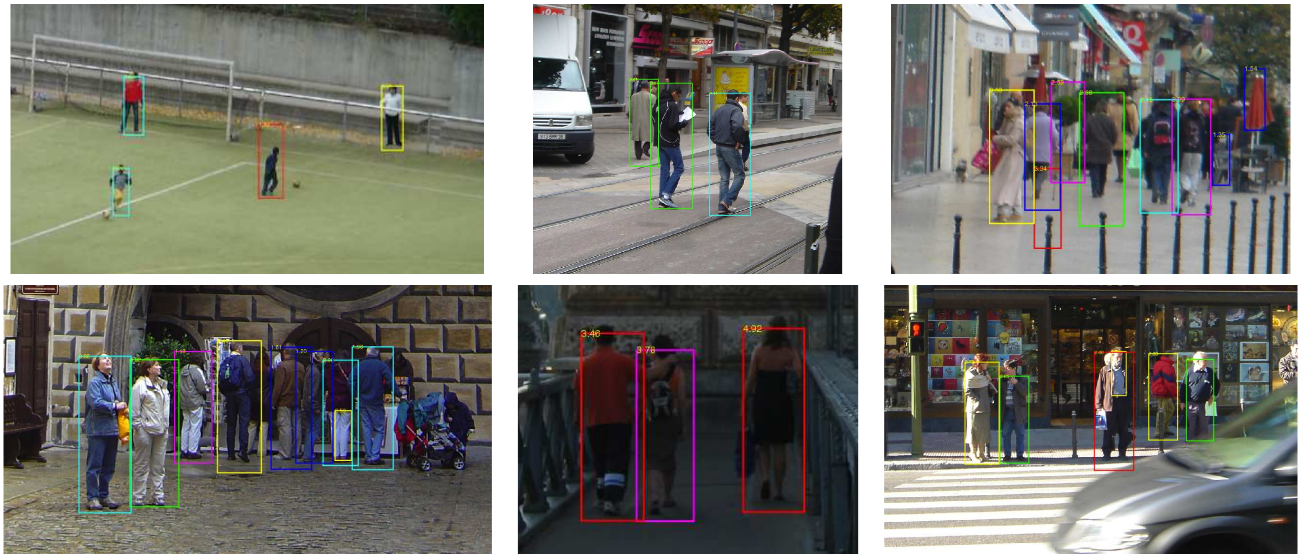


Figure 2.7: Sample pedestrian detections on the INRIA person dataset using spHOG + IKSVM classifier.

Classification Method	Accuracy(%)	Speedup
Linear SVM	72.19 ± 4.40	-
IKSVM (binary search)	89.06 ± 1.42	$\mathbf{485}\times$
IKSVM (piecewise linear)	89.03 ± 1.39	$\mathbf{2253}\times$
IKSVM (piecewise constant)	88.83 ± 1.39	$\mathbf{3100}\times$
RBF-SVM	88.85 ± 1.13	–

Table 2.3: Accuracy on Daimler Crysler Pedestrians dataset for various methods.

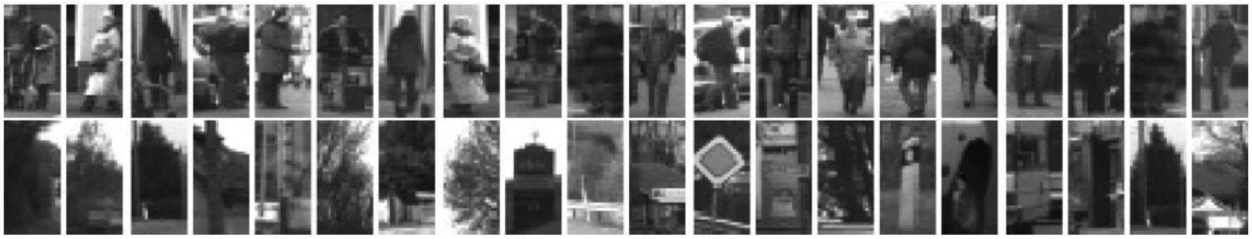


Figure 2.8: (Top Row) False negatives and (Bottom Row) false positives of the classifier on the Daimler-Chrysler pedestrian dataset.

	15 examples, 115.2 ± 15 SVs		30 examples, 185.0 ± 26 SVs	
Classification Method	**Accuracy(%)**	**Speedup**	**Accuracy(%)**	**Speedup**
Linear SVM	38.79 ± 0.94	-	44.33 ± 1.33	-
IKSVM (binary search)	50.15 ± 0.61	**11**×	56.49 ± 0.78	**17**×
IKSVM (piecewise linear)	50.10 ± 0.65	**37**×	56.59 ± 0.77	**62**×
IKSVM (piecewise constant)	49.83 ± 0.62	**45**×	56.11 ± 0.94	**76**×

Table 2.4: Classification accuracy of various methods on Caltech-101 dataset using 15 and 30 training examples per category. The piecewise linear classifiers are up to 60× faster without loss in accuracy over the exact method.

2.7.5 Caltech 101

Our next set of experiments are on Caltech-101 [31]. The aim here is to show that existing methods can be made significantly faster, even when the number of support vectors in each classifier is small. We use the framework of [61] and use our own implementation of their ”weak features” and achieve an accuracy of 56.49% (compared to their 54%), with 30 training and test examples per class and one-vs-all classifiers based on IKSVM. The performance of a linear SVM using the same features is about 44.33%, while that of a rbf kernel is 50.13%. The IKSVM classifiers on average have 185 support vectors and a piecewise linear approximation with 60 bins is 62× faster and the piecewise constant approximation is 76× faster than a standard implementation, with no loss in accuracy (see Table 2.4).

It is interesting to note the performance of one-vs-one classifiers as they are faster to train. With 15 training and 50 test examples per category, one-vs-one classifiers give an accuracy of 47.43 ± 0.37 for intersection, compared to 39.58 ± 0.78 for linear kernel, with 5-fold cross validation. Increasing with number of training examples to 30, improves the performance to 53.80 ± 2.43 for intersection kernel compared to 45.66 ± 2.63 for linear kernel.

2.7.6 UIUC Cars

This dataset was collected at UIUC [1] and contains images of side views of cars. The training set consists of 550 car and 500 non-car images. We test our methods on the single scale image test set which contains 170 images with 200 cars. The images are of different sizes themselves but contain cars of approximately the same scale as in the training images. Results are shown in Table 4.7. Once again the IKSVM classifier outperforms both the linear and the rbf kernel SVM and is comparable to the state of the art. Figure 2.9 shows some of the detections and mis-detections on this dataset.

Classification Method	**Performance(%)**	**Speedup**
Linear SVM	89.8	-
IKSVM (binary search)	98.5	**23**×
IKSVM (piecewise linear)	98.5	**65**×
IKSVM (piecewise constant)	98.5	**83**×
RBF-SVM	93.0	-
Agarwal & Roth [1]	79.0	-
Garg *et al.* [41]	88.0	-
Fregus *et al.* [35]	88.5	-
ISM [62]	97.5	-
Mutch & Lowe [76]	99.6	-
Lampert *et al.* [58]	98.5	-

Table 2.5: Performance at Equal Error Rate on UIUC cars dataset.

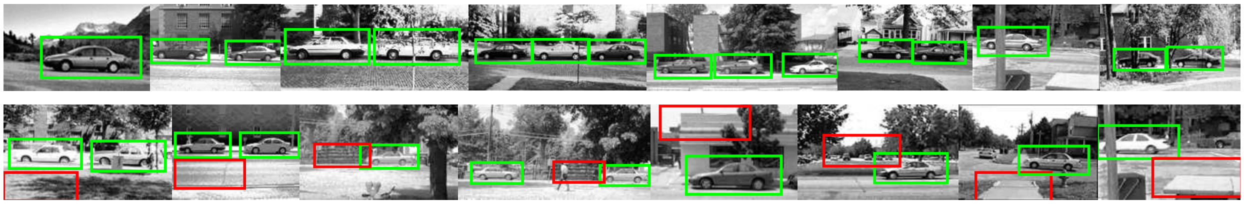


Figure 2.9: Example detections (green) and mis-detections (red) on UIUC cars dataset.

2.8 Conclusion

In this paper we showed that a class of non linear kernels called *additive* kernels lead to SVM classifiers which can be approximately evaluated very efficiently. Additive kernels are strictly more general than linear kernels and often lead to significant improvements and our technique brings down the memory and time complexity of classification using additive kernels to only a small constant multiple of that of a linear SVM. Additive kernels are widely used in computer vision and our technique has found wide spread applications in many classification/detection tasks.

In addition our technique has lead to efficient training algorithms for additive classifiers which we discuss in the next chapter, and has sped up many applications involving histogram based comparison, like multiple kernel learning based detectors [111] and kernel methods like k-means [117], PCA/LDA/regression, etc.

Chapter 3

Training of Additive Classifiers

Too much training data can make learning a bottleneck. Quite suddenly this is becoming a real danger for computer vision research. Efficient marketplaces for small increments of human labeling effort such as Mechanical Turk [2, 103] are making possible huge collections of images labeled and verified by real people at a rate of multiple images per penny as exemplified by `image-net.org` [55] a repository of millions of image examples of the wordnet [32] hierarchy. This complements a range of dataset collection efforts from semi-automatic [6, 63, 95, 19, 85] with $10,000$-$600,000+$ images to fully manual but unpaid [91] with $50,000+$ labeled objects to more traditional datasets [30, 47]. All of which means that thousands to millions of training examples may become the norm for object recognition.

In a sense this is already the case for training object detectors. It is inexpensive to collect many positive images of, say, pedestrians and images of non-pedestrians. Training for a high quality detector typically proceeds in rounds of training a detector and then evaluating the detector on datasets to identify additional false positives to use for future training rounds. When detectors are run using a sliding window at multiple scales in a large image there can easily be $100,000$ or more potential negative training examples per image.

This large amount of data dictates the algorithms that are used. Approximate nearest neighbor techniques in (relatively) high dimensional feature spaces that require no training but may learn parameters for hashing [108, 54, 97, 56] have been applied to image classification. Even these are too slow for detection where boosted decision trees and linear classifiers are the default [21, 112]. Contrast this with the most accurate systems for object recognition in settings where efficiency is less critical, usually obtained using kernelized support vector machines (SVMs) that must compare a test image (or region) to each support vector [8, 109, 61, 46, 20].

In the earlier chapter, we pointed out that many of these SVMs were based on additive kernels and had a classification function with an additive form that could be efficiently approximated and evaluated nearly as fast as a linear classifier. We did not however address the problem of efficiently training classifiers, relying on standard training for kernelized SVM classifiers and then fitting their fast additive classifier to match the SVM classifier.

Our main contribution is to show that classifiers based on additive models can be trained directly in a max margin framework extremely efficiently, and achieve approximately the same accuracy as first training an SVM and then fitting an additive classifiers to the resulting decision function as was done in [66] while taking as little as **1%** of the training time. We achieve this remarkable speedup by a special encoding of the learning problem that allows us to take advantage of (our own modification of) recent techniques for training linear classifiers [98, 29].

The result is something of a “free lunch” (or at least very inexpensive) for computer vision researchers because the combination of our fast training techniques with fast evaluation make training and testing an additive classifier only a small (constant factor) slower than training a linear classifier. At the same time the additive classifiers produce error rates that are almost always significantly lower than those for a linear classifier on computer vision data.

In addition our optimization method is derived from PEGASOS [98] and uses stochastic subgradient descent which allows us to present an “on-line” version of our approach streaming data from out of core and a very efficient interactive training approach for classifiers used in detection.

We report experimental results quantifying training time and accuracy on on image classification tasks and pedestrian detection, including detection results better than the best previous on the INRIA dataset.

3.1 Background

In this paper we train parametric additive classifiers directly, but some of our choices for representation and embedding are motivated by considering SVMs both with linear and non-linear but additive kernels.

The history of the features and kernels used for pedestrian detection and image classification is quite complex we round up the most closely related work. Embeddings that allow sub-linear search for similar distributions of features with respect to the Earth Mover’s distance were introduced by Thaper & Indyk [54] and later combined with the intersection kernel (aka min-kernel) by Grauman et al. [46] to train accurate image classifiers. Lazebnik et al. [61] refined the embedding by using multiple levels for spatial bins, but not for other dimensions of features. One level of Lazebnik’s simple features are very similar to the Histogram of Oriented Gradient (HOG) feature from Dalal and Triggs [21] which was carefully developed to work well with a linear kernel for pedestrian detection and has also been used as the basis of a structured prediction approach to pedestrian detection [33]. The mutli-scale features used by Maji et al. [66] fall between those of [61] and [21].

Earlier Viola & Jones [112] developed their very successful boosting algorithm for training a cascade for face detection. Boosting may seem unrelated to the kernel discussion above, but recent work demonstrates random approximations to boosting using linear SVM training as an intermediate procedure [84]. Furthermore if the weak learners are additive so is a boosted function. Using a random selection of weak learners and our approach may be an effective alternative to [84].

Additive models are well known in the machine learning community [49, 26], and efficient evaluation for non-parametric kernelized SVMs with additive kernels is addressed exactly in [51, 66] and approximately in [66] which motivates much of our analysis of the encodings we use to transform our problem to a form suitable for efficient optimization.

The data encoding strategy used in Sparse Network of Winnows(SNoW) [119], can be seen as a special case of our own and we include experiments that verify advantages of the SNoW encoding strategy over linear in some cases, while showing analytically and empirically that our approach provides significantly better performance.

Our own optimization procedure generalizes that of the very impressive PEGASOS [98] and in comparison experiments we use LIBLINEAR based on [29]. Both represent amazing progress in training efficiency. While the stochastic nature of PEGASOS may be reminiscent of neural network approaches, differences are the max margin formulation, and one key to its efficiency, the renormalization at each step based on the regularization parameter. This is what moves its convergence from $\frac{1}{\lambda^2}$ of other stochastic gradient descent methods to $\frac{1}{\lambda}$.

3.2 Overview

We are interested in learning classifiers based on additive models. The decision functions are $sign(f(x))$ where

$$f(\mathbf{x}) = \sum_i f_i(x_i) \tag{3.1}$$

We call f_i the i^{th} "coordinate function", it operates on the i^{th} coordinate of $\mathbf{x}$. Although additive models are often non-parametric, here we are specifically interested in parametric coordinate functions that can be learned efficiently. For labeled training data $\{(\mathbf{x}^k, y^k)\}_{k=1\ldots n}$ with the labels $y^k \in \{-1, +1\}$ and the data $\mathbf{x}^k \in \mathbb{R}^d$ learning involves finding the f that minimizes a cost function measuring both the training error or loss ℓ and a regularization penalty R

$$f^* = argmin_f R(f) + \frac{1}{n}\sum_k \ell(y^k, f(\mathbf{x}^k)) \tag{3.2}$$

In the rest of the paper we will use the hinge loss $\ell(y^k, f(\mathbf{x}^k)) = \max(0, 1 - y^k f(\mathbf{x}^k))$ motivated by the generalization advantages of large margins, and by the interpretation of f as the decision function of an SVM with additive kernel.

We explore representations that transform Equation 3.2 into an efficiently solvable optimization. If $\mathbf{w}$ is a vector of parameters specifying the parametric function f^w then we want to encode $\mathbf{w}$ as $\hat{\mathbf{w}}$ and a data point $\mathbf{x}$ as $\hat{\mathbf{x}}$ so that $f^w(\mathbf{x}) \approx \hat{\mathbf{w}}'\hat{\mathbf{x}}$ where we emphasize that this may be approximate. After encoding we can write the optimization in Equation 3.2 as

$$f^{\mathbf{w}^*} = argmin_{f^{\mathbf{w}}} R(\hat{\mathbf{w}}) + \frac{1}{n}\sum_k \max(0, 1 - y^k(\hat{\mathbf{w}}'\hat{\mathbf{x}}^k)) \tag{3.3}$$

If each coordinate function $f_i(x_i) = w_i x_i$ and $R(\mathbf{w}) = \mathbf{w}'\mathbf{w}$ then $\hat{\mathbf{w}} = \mathbf{w}$ and $\hat{\mathbf{x}} = \mathbf{x}$ and this is simply a linear support vector machine (without bias). More generally we can use this formulation if the f_i are a linear combination of a finite number of basis functions. In our experiments f_i are piecewise linear with uniformly spaced breaks. This choice is motivated by simplicity and the analysis in the earlier chapter, showing that decision functions with that form could effectively approximate the decision function of SVMs using the min kernel while being very efficient to evaluate, but we emphasize that other spline functions can be used easily. The choice of representations can be thought of as a change of basis or regularization. We discuss this in Section 3.6.

Depending on the form of the regularization function R we can use different approaches for optimization. For instance when $R(\hat{\mathbf{w}}) = \lambda\hat{\mathbf{w}}'\hat{\mathbf{w}}$ we can use an “off the shelf” SVM package on the encoded data $\{(\hat{\mathbf{x}}^k, y^k)\}$. On the other hand, for the “full” version of our approach – motivated by regularization for kernelized SVMs – we present a modified version of the the PEGASOS [98] stochastic sub-gradient descent (with careful normalization) linear SVM solver that can handle $R(\hat{\mathbf{w}}) = \mathbf{w}'\mathbf{H}\mathbf{w}$ for positive definite $\mathbf{H}$. Only the case of $\mathbf{H} = \mathbf{I}$, the identity, is addressed by [98].

Section 3.3 goes into options for piecewise linear encoding in detail, including analysis of representation error and the implications for choices of regularization R. Then Section 3.4 presents our modified version of PEGASOS. In Section 3.6, we connect our learning algorithm to the additive model learning literature. In particular we adapt a *penalized spline* formulation due to Eilers and Marx [26], to train additive classifiers efficiently. We show interesting connections between B-Spline basis and histogram intersection kernel and show that for a particular choice of regularization and degree of the B-Splines, our proposed learning algorithm closely approximates the histogram intersection kernel SVM.

3.3 Encoding

We will consider the encoding process described in Section 3.2 as an approximation to the embedding implied by a specific additive kernel, the $\min$ or histogram intersection kernel, K_{min}, also known as the intersection kernel or min kernel:

$$K_{\min}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} \min\left(x(i), z(i)\right) \tag{3.4}$$

The Reproducing Kernel Hilbert Space (RKHS) of the $\min$ kernel is universal with respect to additive functions, i.e. any additive function on the input features can be expressed as dot product of a weight vector and the features in the RKHS. This is analogous to the fact that weighted sum of (possibly infinite) decision stumps can universally express any one dimensional function. This coupled with the fact the $\min$ kernel is conditionally positive definite (CPD) for real valued $\mathbf{x}$ allows one to use $\min$ kernels to learn general additive models on real valued features. CPD kernels are a set of kernels which satisfy a weaker set of conditions than positive definite kernels, but can be easily modified to yield a PD kernel. For the $\min$ kernel this corresponds to adding a large positive constant. Other additive kernels like the $-\chi^2$ induce RKHS which can also express arbitrary functions, but we often find that these differences in the set of basis functions tend to not matter on large training datasets, which is our focus.

First we show explicitly that any SVM $h(\mathbf{x})$ with support vectors $\{\mathbf{v}^k\}$ and additive kernel $K(\mathbf{x}, \mathbf{y}) = \sum_i K_i(x_i, y_i)$ is additive:

$$\begin{aligned} h(\mathbf{x}) &= \sum_j \alpha_j K(\mathbf{x}, \mathbf{v}^j) + b && (3.5) \\ &= \sum_j \alpha_j \sum_i K_i(x_i, v_i^j) + b && (3.6) \\ &= \sum_i \sum_j \alpha_j K_i(x_i, v_i^j) + b && (3.7) \\ &= \sum_i f_i(x_i) + b && (3.8) \end{aligned}$$

where $f_i(x_i) = \sum_j \alpha_j K_i(x_i, v_i^j)$. For the histogram intersection kernel K_i is simply $\min$. It is sufficient to consider encoding for each dimension separately (as the h is additive) so consider encoding two coordinate values x and y both in $[0, 1]$ for simplicity. In this case the goal of an encoding for the two is that $\min(x, y) = \hat{x}\hat{y}$

One straight forward encoding is to choose a fixed discretization scale and represent the features in the "unary". Let N denote the number discrete levels and $U(n), n \in Z$ denote the unary representation of the number n, i.e. $U(3) = 1, 1, 1, 0, 0, 0$, $U(6) = 1, 1, 1, 1, 1, 1$, etc, and $R(.)$ denote the rounding function, then we define our first feature encoding:

$$\phi_1(x) = \sqrt{\frac{1}{N}} U\left(R\left(Nx\right)\right) \tag{3.9}$$

Intuitively this encoding discretizes the feature into a fixed set of levels and represents each feature using the unary representation. The kernel can then be defined by

$$\begin{aligned} \min(x, y) &\approx \; < \phi_1(x), \phi_1(y) > \\ &= \; < \sqrt{\frac{1}{N}} U\left(R\left(Nx\right)\right), \sqrt{\frac{1}{N}} U\left(R\left(Ny\right)\right) > \\ &= \; \frac{1}{N} < U\left(R\left(Nx\right)\right), U\left(R\left(Ny\right)\right) > \end{aligned}$$

An alternate representation is to use an encoding which instead of rounding to the nearest bin, keeps more detailed information about the values. We define the alternate representation $U'(r)$ for any real number $r >= 0$ as the unary representation, but replacing the first zero in the unary representation of $U(\lfloor r \rfloor)$ by $\alpha(r) = r - \lfloor r \rfloor$. As an example $U'(3.5) = 1, 1, 1, 0.5, 0, 0$

$$\phi_2(x) = \sqrt{\frac{1}{N}} U'\left(Nx\right) \tag{3.10}$$

The dot product then becomes:

$$\begin{aligned} \min(x,y) &\approx\ <\phi_2(x),\phi_2(y)> \\ &= \frac{1}{N} < U'(Nx), U'(Ny) > \end{aligned}$$

We consider the approximation quality for both these linear encodings in the next section.

3.3.1 Approximation Quality

We will present the worst case and average approximation errors for both these embeddings. In both cases :

$$E^{\max}_{\phi}(x,y) = |\min(x,y) - <\phi(x),\phi(y)>| < \frac{1}{N} \tag{3.11}$$

However we can be more precise about these errors for each of the kernels. The $\min$ operation is symmetric so we need only consider the case when $x \le y$ and $\min(x,y) = x$.

ϕ_1 : Since $x \le y$ we have $R(Nx) \le R(Ny)$. So $< U(R(Nx)), U(R(Ny)) >= R(N(x))$. Therefore the max approximation error

$$E^{\max}_{\phi_1}(x,y) = \max|x - R(Nx)/N| = \frac{1}{2N} \tag{3.12}$$

ϕ_2 : Since $x \le y$, there can only be two cases:

1. $\lfloor Nx \rfloor < \lfloor Ny \rfloor$: In which case the embedding is exact because:

$$\begin{aligned} <\phi_2(x),\phi_2(y)> &= \ < U'(Nx), U'(Ny) > \\ &= \frac{1}{N}(\lfloor Nx \rfloor + \alpha(Nx)) \\ &= \frac{1}{N}(\lfloor Nx \rfloor + Nx - \lfloor Nx \rfloor) \\ &= x \\ &= \min(x,y) \end{aligned}$$

2. $\lfloor Nx \rfloor = \lfloor Ny \rfloor = Nm$: Denote $\alpha(Nx)$ by a and $\alpha(Ny)$ by b. Then we have $\min(x,y) = x = m + \frac{a}{N}$, and

$$\begin{aligned} <\phi_2(x),\phi_2(y)> &= \ < U'(Nx), U'(Ny) > \\ &= \frac{1}{N}(\lfloor Nx \rfloor + ab) \\ &= \frac{1}{N}(Nm + ab) \\ &= m + \frac{1}{N}ab \\ &= \min(x,y) + \frac{1}{N}(ab - a) \end{aligned}$$

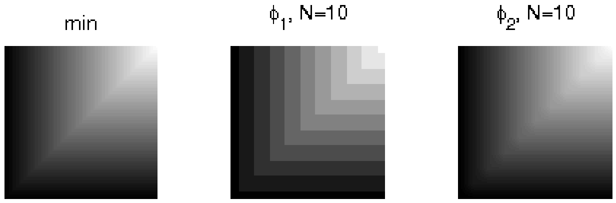


Figure 3.1: From left to right $\min(x, y)$, $\phi_1(x)\phi_1(y)$ and $\phi_2(x)\phi_2(y)$ with $N = 10$. Note that the ϕ_2 encoding is very close to $\min$.

First notice that in this the dot product is always an underestimate of $\min$ value as $ab - a \leq 0$ with $a, b \in [0, 1]$. Also the max approximation error is :

$$E^{\max}_{\phi_2}(x, y) = \max_{a \leq b, a, b \in [0,1]} \frac{1}{N}|ab - a| = \frac{1}{4N}$$

If we assume that x, y are distributed uniformly in $[0, 1] \times [0, 1]$, then we can also compute the expected error, $E^{avg}_{\phi_1} = \frac{1}{4N}$, while $E^{avg}_{\phi_2} = \frac{1}{12N^2}$. This shows that the encoding error decreases with the number of bins and the ϕ_2 encoding is twice as accurate as the ϕ_1 encoding in terms of max error, and significantly better if we care about the average error under a uniform distribution. Figure shows the kernel function for $\min$, ϕ_1 and ϕ_2 for $N = 10$.

3.3.2 Sparse Version of Encoding and Regularization

We saw that the $\min$ kernel can be approximated to within ϵ using $O(1/\epsilon)$ bins for ϕ_1 and $O(1/\sqrt{\epsilon})$ bins for ϕ_2. Hypothetically we could train a linear SVM on those encodings which would be an approximation the the SVM on the original data using an intersection kernel. However these representations are dense, and training a linear SVM on such dense representations become infeasible as the number of dimensions become large. Instead we propose a sparse representation for each of the embeddings given by:

$$\phi^s_2(x) = \frac{1}{\sqrt{N}}(i : 1 - a, i + 1 : a) \tag{3.13}$$

(a vector of all zeros except $\frac{1}{\sqrt{N}}(1-a)$ at position i and $\frac{1}{\sqrt{N}}a$ at position $i+1$) where $a = \alpha(Nx)$ as defined earlier, and $i = \lceil Nx \rceil$ and features are represented by $index : value$ pairs. The transform for ϕ^s_1 is the same except both $1 - a$ and a are rounded to 0 or 1, resulting in an encoding similar to that of SNoW [119] where they train a linear SVM on these sparse features. The SNoW encoding however does not preserve the underlying $\min$ based similarity measure. We now propose an encoding for $\mathbf{w}$ (as in Equation 3.3) that is compatible with using $\phi^s_{\{1,2\}}$ to encode $\mathbf{x}$.

If $\mathbf{w} \in \mathbb{R}^N$ is a weight vector (for instance found by fitting an SVM) on encoded data $\phi_2(x) \in \mathbb{R}^N$ and $\mathbf{w}^s \in \mathbb{R}^{N+1}$ a weight vector on the same data encoded as $\phi^s_2(x) \in \mathbb{R}^{N+1}$. We want $\mathbf{w}$ such that $\mathbf{w} \cdot \phi_2(x) = \mathbf{w}^s \cdot \phi_2^s(x)$. The required relationship is

$$w^s(i+1) = w^s(i) + w(i), \text{and } w^s(1) = 0 \tag{3.14}$$

An important point is how to compute the regularization penalty on $\mathbf{w}^s$. Again if we were hypothetically training a linear SVM on the dense ϕ_2 encodings of the data the regularization penalty would be $\mathbf{w}'\mathbf{w}$.

The corresponding regularization penalty for $\mathbf{w}^s$ is then:

$$\mathbf{w}'\mathbf{w} = \sum_{i=1}^{N} w(i)^2 = \sum_{i=1}^{N} (w^s(i+1) - w^s(i))^2 \tag{3.15}$$

This can expressed as $\mathbf{w}^{s'}\mathbf{H}\mathbf{w}^s$ where $\mathbf{H}$ is tridiagonal, with the form:

$$\mathbf{H} = \begin{pmatrix} 1 & -1 & 0 & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \cdots & & & \\ & & -1 & 2 & -1 & \\ & & 0 & -1 & 1 & \end{pmatrix}$$

So far our discussion has dealt with only a single coordinate. All encodings are done on a coordinate by coordinate basis and appended. For instance if we use $N = 20$ divisions per coordinate and have 100 dimensional features $x \in \mathbb{R}^{100}$ then $\phi_2^s(x) \in \mathbb{R}^{2000}$ but has at most 2×100 non-zero entries. Similarly $\mathbf{H}$ is 2000×2000 and all zeros except for 100 blocks as described above along the diagonal. In what follows we only use the sparse encodings, so ϕ_1 will mean ϕ_1^s and ϕ_2 will mean ϕ_2^s.

3.4 Optimization

Once encoding the data is done and we have chosen a regularization penalty of the form $R(\hat{\mathbf{w}}) = \hat{\mathbf{w}}'\mathbf{H}\hat{\mathbf{w}}$ as described above we need to find parameters $\hat{\mathbf{w}}^*$ that minimize the cost function c,

$$c(\hat{w}) = \frac{\lambda}{2}\hat{\mathbf{w}}'\mathbf{H}\hat{\mathbf{w}} + \frac{1}{n}\sum_{k=1\ldots n} \max(0, 1 - y^k(\hat{\mathbf{w}}'\hat{\mathbf{x}}^k)) \tag{3.16}$$

where λ is the regularization vs loss tradeoff. When $\mathbf{H}$ is the identity this is simply optimization to fit a linear SVM. In that case a standard linear SVM solver can be used although ideally one that can efficiently utilize a sparse representation for $\hat{\mathbf{x}}$ such as [29, 98].

For our regularization motivated by the min kernel, $\mathbf{H}$ is the tri-diagonal matrix described in Section 3.3.2. And we use our modified version of the PEGASOS algorithm for fitting linear SVMs [98]. The original analysis of PEGASOS depends on two aspects of the objective function

c – first that c be strongly convex which is true in our case as long as $\mathbf{H}$ is positive definite[1], and second that the optimum $\hat{\mathbf{w}}^*$ has norm $\hat{\mathbf{w}}^{*'}\hat{\mathbf{w}}^* \leq \frac{1}{\lambda}$, in our case $\hat{\mathbf{w}}^{*'}\mathbf{H}\hat{\mathbf{w}}^* \leq \frac{1}{\lambda}$.

Next we show our modification of PEGASOS in its entirety in Algorithm 1. Note that if $\mathbf{H}$ is replaced with the identity matrix then this is exactly the PEGASOS algorithm. When we use the tri-diagonal $\mathbf{H}$ and either encoding, ϕ_2 or ϕ_2 for $\hat{x}$ as described in Sec. 3.3.2 we call the algorithm "piecewise linear sub-gradient descent" (PWLSGD).

Algorithm 1 Our modification of PEGASOS (PWLSGD)

```
Require: $S, T, \lambda > 0$ and $k > 0$
  initialize $\hat{\mathbf{w}}_1$ randomly, such that $\hat{\mathbf{w}}_1'\mathbf{H}\hat{\mathbf{w}}_1 \leq \frac{1}{\lambda}$
  for $t = 1$ to $T$ do
    Choose $A_t \subset S$, where $|A_t| = l$
    Set $A_t^+ = \{(\hat{\mathbf{x}}, y) \in A_t : y \langle \hat{\mathbf{w}}, \hat{\mathbf{x}} \rangle < 1\}$
    Set $\eta_t = \frac{1}{\lambda t}$
    Set $\hat{\mathbf{w}}_{t+\frac{1}{2}} = \hat{\mathbf{w}}_t - \eta_t \left( \lambda \hat{\mathbf{w}}_t \mathbf{H} + \frac{1}{l} \sum_{(\mathbf{x},y) \in A_t^+} y\hat{\mathbf{x}} \right)$
    Set $\hat{\mathbf{w}}_{t+1} = \min \left( 1, \frac{1/\sqrt{\lambda}}{\hat{\mathbf{w}}_{t+\frac{1}{2}}{}' \mathbf{H} \hat{\mathbf{w}}_{t+\frac{1}{2}}} \right) \hat{\mathbf{w}}_{t+\frac{1}{2}}$
  end for
```

Here $S = \{(\mathbf{x}^k, y^k)\}_{k=1\ldots n}$ is all of the training data, A_t is a random subset of k chosen for the t^{th} iteration, and A_t^+ is the subset of these which violate the margin constraint using estimate of weight vector $\hat{\mathbf{w}}_t$ in step t. From [98] the error $|c(\hat{\mathbf{w}}_t) - c(\hat{\mathbf{w}}^*)|_\infty \leq \epsilon$ after $\tilde{O}(\frac{1}{\delta\epsilon\lambda})$ steps with probability $1 - \delta$ when $k = 1$ and after $\tilde{O}(\frac{1}{\epsilon\lambda})$ steps when $k = n$. Intermediate values of k fall between these bounds. In practice the convergence depends on the number of margin violations – basically the difficulty of the classification problem.

We mention briefly some differences in computational complexity from the original PEGASOS. Our variation requires computing $\mathbf{H}'\hat{\mathbf{w}}_\mathbf{t}$ and $\hat{\mathbf{w}}_\mathbf{t}'\mathbf{H}\hat{\mathbf{w}}_t$ for each update. For tridiagonal $\mathbf{H}$ this costs roughly 3 times the computation for $\mathbf{H} = \mathbf{I}$, hence the small multiple in computation time. It is possible that more efficient implementations than our current one, using loop unrolling and other techniques, might be able to hide some of this added complexity. In addition for the particular encodings ϕ_1 and ϕ_2 the encoding pattern in our data $\hat{x}$ is known and fixed (exactly one or exactly two coefficients can be non-zero in each coordinate block) so we can avoid using linked lists for representing the data.

On-line and Interactive Learning: One significant benefit of basing optimization on a stochastic sub-gradient descent method such as PEGASOS is that we can perform learning in stages. For

[1] Our tri-diagonal $\mathbf{H}$ is not positive definite. Adding a small constant (e.g. 0.01) to the first diagonal entry in each coordinate block the diagonal makes it positive definite without effecting the accuracy on experiments. Except for small $k <= 3$ using the original semidefinite $\mathbf{H}$ has no effect on the convergence rate.

instance in the process of training a detector it is evaluated on many images, false positives (or missed detection) are added to a new training set. After evaluating several hundred or thousand of these a new classifier is trained [21]. We can actually update the classifier by running additional steps of stochastic gradient descent using the new data from each image. As long as the distribution of images is randomized the convergence estimates are very similar. This approach also avoids the memory bottle-neck reported by [21].

The above assumes a priori labeled data, but this does not need to be the case. A human could mark false positives and missed detections in each successive image in an interactive setting. Running a few iterations of training per image can be done faster than humans can label.

3.5 Experimental Results

We present training time and testing accuracy numbers for each of the proposed methods. We have a choice of encoding: identity, ϕ_1, and ϕ_2, and a choice of learning algorithm: linear ($\mathbf{w}'\mathbf{w}$ regularization) using an off the shelf SVM solver, or a piecewise linear classifier ($\mathbf{w}'\mathbf{H}\mathbf{w}$ regularization) using our PWLSGD algorithm on the encoded features. We present results on Caltech-101, Daimler-Crysler dataset and INRIA pedestrian dataset and show that all encodings outperform linear classifiers on the non-encoded features by significant amount, and that the encoding and training can be done in a small time compared to training a kernel SVM.

3.5.1 Caltech-101

Our first set of experiments are on the Caltech-101 dataset [30]. We use this dataset to show that the accuracy using spatial pyramid match kernel introduced in [61] can be matched using our embeddings. For each category we select either 15 or 30 random examples for training and test it random set of at most 50 training examples as some categories have fewer than 50 remaining for test. We report numbers by averaging the class accuracy for 101 categories using 5-fold cross validation. All the parameters for the models are obtained using by optimizing the performance on a fixed set of 15 training and 15 test examples per category and we use the same parameters for both 15 and 30 training images. We use our own implementation of the “weak features” introduced in [61] and achieve an accuracy of 50.15% and 56.49% , with 15 and 30 training examples per class and one-vs-all SVM classifiers based on the spatial pyramid match kernel. This kernel reduces to a $\min$ (or intersection) kernel on histograms of oriented gradients obtained from each level of a spatial pyramid, concatenated together after suitable weighting. Table 3.1 shows the cumulative training time and accuracies of various methods on this dataset. Linear SVMs are the fastest but also perform the worst. The ϕ_2 encoding with our piecewise linear training algorithm achieves accuracy similar to the intersection kernel SVM at lower training times. Even a linear SVM trained on the ϕ_2 encoded features offers a good accuracy improvement over a linear SVM trained on the raw features at the cost of a small increase in training time. The accuracy using snow encoding (ϕ_1) is quite worse possibly because of quantization.

		15 examples		30 examples	
Encoding	Training Algorithm	Training Time(s)	Accuracy(%)	Training Time(s)	Accuracy(%)
identity	LIBLINEAR	**18.57 (0.87)**	41.19 (0.94)	**40.49 (0.80)**	46.15 (1.33)
identity	LIBSVM (int kernel)	844.13 (2.10)	**50.15 (0.61)**	2686.87 (4.30)	**56.49 (0.78)**
snow=ϕ_1	LIBLINEAR	45.22 (1.17)	46.02 (0.58)	89.68 (0.93)	51.64 (1.02)
ϕ_2	LIBLINEAR	42.31 (1.43)	48.70 (0.61)	101.97 (1.09)	54.79 (1.24)
ϕ_2	PWLSGD	238.98 (2.49)	**49.89 (0.45)**	291.30 (1.98)	**55.35 (0.72)**

Table 3.1: Cumulative training time in seconds *(stdev)* and mean class accuracy *(stdev)* for various encodings and algorithms on Caltech 101 dataset using 5 fold cross validation.

Encoding	Training Algorithm	Training Time(s)	Accuracy(%)
identity	LIBLINEAR	**1.89 (00.10)**	72.98 (4.44)
identity	LIBSVM (int. kernel)	363.10 (27.85)	**89.05 (1.42)**
snow=ϕ_1	LIBLINEAR	2.98 (00.33)	85.71 (1.43)
ϕ_2	LIBLINEAR	1.86 (00.04)	88.80 (1.62)
ϕ_2	PWLSGD	3.18 (00.01)	**89.25 (1.58)**

Table 3.2: Training time in seconds (stdev) and accuracy (stdev) of various algorithms on the Daimler Chrysler Pedestrian dataset. Each training set has $20,000$ features of 656 dimensions and it takes about $1.84(0.006)$ seconds to encode them.

3.5.2 Daimler Chrysler Pedestrian Dataset

Our second set of experiments are on the Daimler Chrysler pedestrian benchmark dataset, created by Munder and Gavrila [74]. The dataset is split into five disjoint sets, three for training and two for testing. Each training set has 5000 positive and negative examples each, while each test set has 4900 positive and negative examples each. We report the training times and accuracies by training on two out of three training sets at a time and testing on each of the test sets. We use the same spatial pyramid of histograms of oriented gradients features as before. Once again we optimize the parameters on one split and keep the parameters fixed for all the remaining runs. Table 3.2, shows the performance of various algorithms on this dataset. Once again the ϕ_2 encoded features with the piecewise linear training obtains accuracy similar to the intersection kernel SVM at requiring only about 1% of its training time.

3.5.3 INRIA Pedestrians

We present further results on the INRIA pedestrian dataset using two slightly different features. This is the largest dataset we experiment on training on up to about $50,000$ features of about 4000

Encoding	Training Algorithm	Training Time (HOG)	Training Time (spHOG)
identity	LIBLINEAR	40s	20.12s
identity	LIBSVM (lin. lernel)	$>$180 min	140 min
identity	LIBSVM (int. kernel)	$>$180 min	148 min
snow=ϕ_1	LIBLINEAR	35.52s	121.81s
ϕ_2	LIBLINEAR	22.45s	26.76s
ϕ_2	PWLSGD	99.85s	76.12s

Table 3.3: **(HOG)** $47,327$ features of 3780 dimension. Encoding Time $87.22s$. Dalal and Triggs use a modified SVMLIGHT which is faster than LIBSVM, but still takes several minutes to train, slower than our PWLSGD on ϕ_2 encoding which produces both better classification using either HOG or spHOG (below) and better detection (Fig. 3.2 using spHOG). **(spHOG)** : Training 39K features of 2268 dimension using PWLSGD on the ϕ_2 encoding takes only about 1% of the time taken to train a kernel SVM, and performs as well for classification (see below).

dimensions. We describe how we collect our training/test sets below.

Hard Training Data (HOG). We use the Dalal and Triggs implementation and collect all the “hard” training examples after the first round of training of a linear SVM. The dataset consists of $47,327$ features of 3780 dimension each, which is the largest dataset we test our algorithms on. We report accuracies by testing on random split of 20% of the dataset consisting of 20% of each of positive examples.

Hard Training Data (spHOG). We use the spatial pyramid HOG (spHOG) from Maji et al. [66] to train a SVM classifier. The primary goal was to see if we could approximate the classifier learned by the expensive SVM learning framework using our fast approximation. There are about 39K features of 2268 dimension. We set aside 10% of the data for cross validation optimization of the hyperparameters.

Figure 3.2, shows the classification accuracy of various methods and features on this dataset. Linear SVM on the HOG features performs quite well, and the intersection kernel SVM offers a slight improvement in accuracy. On the spHOG features the performance of the linear SVM is quite poor and there is a significant improvement in accuracy obtained by using the intersection kernel. In both these datasets the performance is closely matched by the ϕ_2 encoding with the piecewise linear training method. Table 3.3 shows the training times taken by the various training algorithms. Training a kernel SVM classifier on the entire dataset using LIBSVM can take several hours, while our technique takes less than two minutes.

Final detection actually showing our performance on detection in the INRIA data are in Figure 3.2. In order to produce these, the classifier was run on a sliding window and non max suppression to the results was applied according to the same procedures described by Dalal and Triggs.

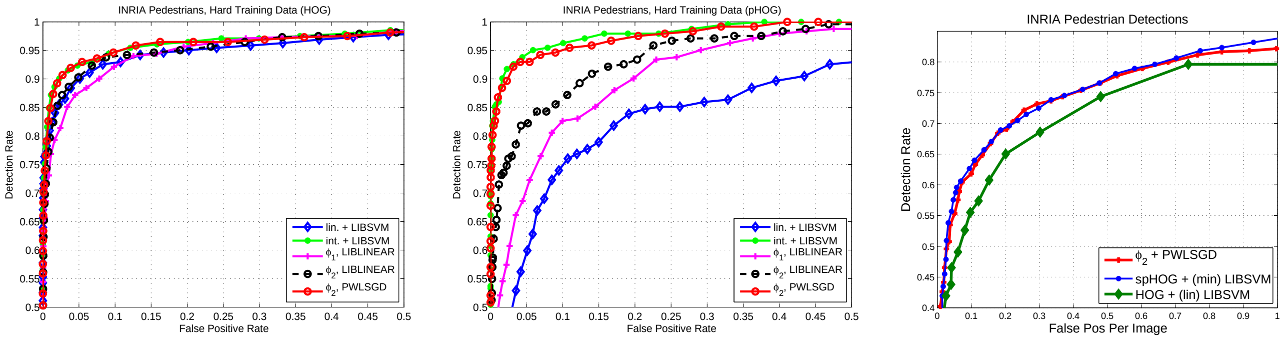


Figure 3.2: Cross Validation Plots on INRIA Pedestrian Hard Training Data. (left) HOG, (middle) spHOG features. Linear SVM works very well with HOG features as these features were designed with linear SVM in mind. We still observe a slight boost in performance. However with the pHOG features the linear SVM does very poorly and the intersection kernel does the best. The performance is closely matched by our ϕ_2 embedding with PWLSGD algorithm. (right) **Detection** Plots on the INRIA benchmark. We compare our detector with the Dalal and Triggs, HOG + (lin.) LIBSVM. All the detectors are run at a stride of 8×8 pixels, and scaleratio of $2^{1/8}$. The correct detection criteria is ratio of bounding box intersection to union above 50%.

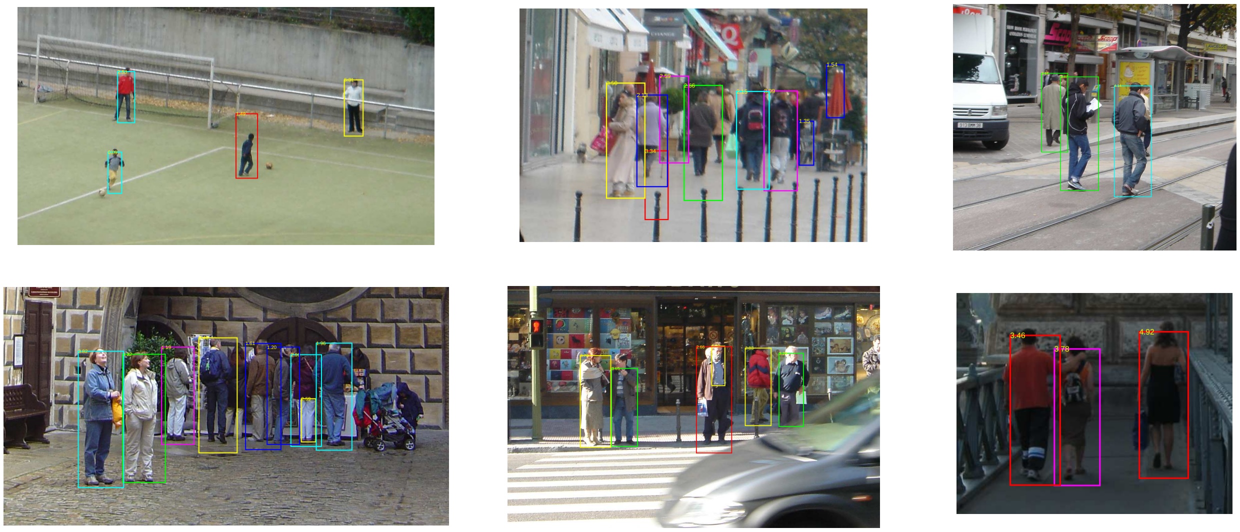


Figure 3.3: Sample detections on the INRIA Pedestrian dataset using ϕ_2 + PWLSGD algorithm.

3.6 Additive Modeling using Spline Embeddings

Eilers and Marx [26] proposed a practical modeling approach for GAMs. The idea is based on the representing the functions in each dimension using a relatively large number of uniformly spaced B-Spline basis. The smoothness of these functions is ensured by penalizing the first or second order differences between the adjacent spline coefficients. Let $\boldsymbol{\phi}(x^k)$ denote the vector with entries $\phi_i(x^k)$, the projection of x^k on to the ith basis function. The P-Spline optimization problem for the classification setting with the hinge loss function consists of minimizing $c(\mathbf{w})$:

$$c(w) = \frac{\lambda}{2}\mathbf{w}'D_d'D_d\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}'\boldsymbol{\phi}(x^k)\right)\right) \tag{3.17}$$

The matrix D_d constructs the dth order differences of $\boldsymbol{\alpha}$:

$$D_d\boldsymbol{\alpha} = \Delta^d\boldsymbol{\alpha} \tag{3.18}$$

The first difference of $\boldsymbol{\alpha}$, $\Delta^1\boldsymbol{\alpha}$ is a vector of elements $\alpha_i - \alpha_{i+1}$. Higher order difference matrices can be computed by repeating the differencing. For a n dimensional basis, the difference matrix D_1 is a $(n-1)\times n$ matrix with $d_{i,i} = 1$, $d_{i,i+1} = -1$ and zero everywhere else. The matrices D_1 and D_2 are as follows:

$$D_1 = \begin{pmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & \dots & & \\ & & & 1 & -1 \end{pmatrix}; D_2 = \begin{pmatrix} 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \dots & & & \\ & & & 1 & -2 & 1 \end{pmatrix}$$

To enable a reduction to the linear case we propose a slightly different difference matrix D_1. We let D_1 be a $n \times n$ matrix with $d_{i,i} = 1, d_{i,i-1} = -1$. This is same as the first order difference matrix proposed by Eilers and Marx, with one more row added on top. The resulting difference matrices D_1 and $D_2 = D_1^2$ are both $n \times n$ matrices:

$$D_1 = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \dots & & \\ & & -1 & 1 & \\ & & & -1 & 1 \end{pmatrix}; D_2 = \begin{pmatrix} 1 & & & & & \\ -2 & 1 & & & & \\ & 1 & -2 & 1 & & \\ & & 1 & -2 & 1 & \\ & & & \dots & & \\ & & & 1 & -2 & 1 \end{pmatrix}$$

The first row in D_1 has the effect of penalizing the norm on the first coefficient of the spline basis, which plays the role of regularization in the linear setting (e.g. ridge regression, linear SVMs, etc). Alternatively one can think of this as an additional basis at left most point with its coefficient set to zero. *The key advantage is that the matrix D_1 is invertible and has a particularly simple form which allows us to linearize the whole system.* We will also show in Section 3.6.1 that the derived embeddings also approximate the learning problem of kernel SVM classifier using the $\min$ kernel ($K_{\min}$) for a particular choice of spline basis.

$$K_{\min}(\mathbf{x}, \mathbf{y}) = \sum_i \min(x_i, y_i) \tag{3.19}$$

Given the choice of the regularization matrix D_d which is invertible, one can linearize the whole system by re-parametrizing $\mathbf{w}$ by $D_d^{-1}\mathbf{w}$, which results in :

$$c(w) = \frac{\lambda}{2}\mathbf{w}'\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}'D_d^{'-1}\boldsymbol{\phi}(x^k)\right)\right) \tag{3.20}$$

Therefore the whole classifier is linear on the features $\boldsymbol{\phi}^d(x^k) = D_d^{'-1}\boldsymbol{\phi}(x^k)$, i.e. the optimization problem is equivalent to

$$c(w) = \frac{\lambda}{2}\mathbf{w}'\mathbf{w} + \frac{1}{n}\sum_k \max\left(0, 1 - y^k\left(\mathbf{w}'\boldsymbol{\phi}^d(x^k)\right)\right) \tag{3.21}$$

The inverse matrices $D_1^{'-1}$ and $D_2^{'-1}$ are both upper triangular matrices. The matrix $D_1^{'-1}$ has entries $d_{i,j} = 1, j \geq i$ and $D_2^{'-1} = D_1^{'-2}$ has entries $d_{i,j} = j - i + 1, j \geq i$ and look like:

$$D_1^{'-1} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 & 1 \\ & 1 & 1 & \ldots & 1 & 1 \\ & & 1 & \ldots & 1 & 1 \\ & & \ldots & \ldots & \ldots & \\ & & & & 1 & 1 \\ & & & & & 1 \end{pmatrix}; D_2^{'-1} = \begin{pmatrix} 1 & 2 & 3 & \ldots & n-1 & n \\ & 1 & 2 & \ldots & n-2 & n-1 \\ & & 1 & \ldots & n-3 & n-2 \\ & & \ldots & \ldots & \ldots & \\ & & & & 1 & 2 \\ & & & & & 1 \end{pmatrix}$$

We refer the readers to [27] for an excellent review of additive modeling using splines. Figure 3.4 shows the $\boldsymbol{\phi}^d$ for various choices of the regularization degree $d = 0, 1, 2$ and B Splines basis, linear, quadratic and cubic.

3.6.1 Additive Kernel Reproducing Kernel Hilbert Space & Spline Embeddings

We begin by showing the close resemblance of the spline embeddings to the $\min$ kernel. To see this, let the features in $[0, 1)$ be represented with $N + 1$ uniformly spaced linear spline basis centered at $0, \frac{1}{N}, \frac{2}{N}, \ldots, 1$. Let $r = \lfloor Nx \rfloor$ and let $\alpha = Nx - r$. Then the features $\boldsymbol{\phi}(x)$ is given by $\phi_r(x) = 1 - \alpha, \phi_{r+1}(x) = \alpha$ and the features $\boldsymbol{\phi}^1(x)$ for D_1 matrix is given by $\phi_i^1(x) = 1$, if $i \leq r$ and $\phi_r^1(x) = \alpha$. It can be seen that these features closely approximates the $\min$ kernel, i.e.

$$\frac{1}{N}\boldsymbol{\phi}^1(x)'\boldsymbol{\phi}^1(y) \approx \min(x, y) + 1 \tag{3.22}$$

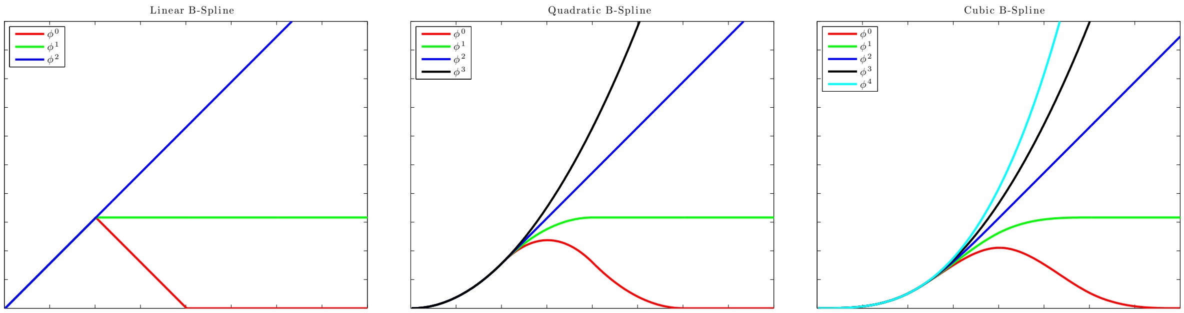


Figure 3.4: Local basis functions for linear (left), quadratic (middle) and cubic (right) for various regularizations degrees d. In each figure $\boldsymbol{\phi}^d$ refers to the dense features $D_d^{'-1}\boldsymbol{\phi}$. When $d = 0$, the function shown in the local basis of B-Splines. When $d = r + 1$, where r is degree of the B-Spline basis, then $\boldsymbol{\phi}^d$ are truncated polynomials basis, $(x - \tau_i)_+^r$ (see Section 3.6.1).

The features $\boldsymbol{\phi}^1(x) = D_1^{'-1}\boldsymbol{\phi}(x)$ constructs a unary like representation where the number of ones equals the position of the bin of x. One can verify that for a B-spline basis of degree r ($r = 1, 2, 3$), the following holds:

$$\frac{1}{N}\boldsymbol{\phi}^1(x)'\boldsymbol{\phi}^1(y) = \min(x, y) + \frac{r+1}{2}, \text{if}|x - y| \geq \frac{r}{N} \tag{3.23}$$

Define K_d^r the kernel corresponding to a B-Spline basis of degree r and regularization matrix D_d as follows:

$$K_d^r(x, y) = \frac{1}{N}\boldsymbol{\phi}^d(x)'\boldsymbol{\phi}^d(y) - \frac{r+1}{2} = \frac{1}{N}\boldsymbol{\phi}(x)'D_d^{-1}D_d^{'-1}\boldsymbol{\phi}(y) - \frac{r+1}{2} \tag{3.24}$$

Figure 3.5 shows K_r^1 for $r = 1, 2, 3$ corresponding to a linear, quadratic and cubic B-Spline basis. In a recent paper, Maji and Berg [65], propose to use linear spline basis and a D_1 regularization, to train approximate intersection kernel SVMs, which in turn approximate arbitrary additive classifiers. Our features can be seen as a generalization to this work which allows arbitrary spline basis and regularizations.

B-Splines are closely related to the truncated polynomial kernel [113, 81] which consists of uniformly spaced knots $\tau_1, \ldots, \tau_n$ and truncated polynomial features:

$$\phi_i(x) = (x - \tau_i)_+^p \tag{3.25}$$

However these features are not as numerically stable as B-Spline basis (see [27] for an experimental comparison). Truncated polynomials of degree k corresponds to a B-Spline basis of degree k and D_{k+1} regularization, i.e, same as K_k^{k+1} kernel, when the knots are uniformly spaced. This is because B-Splines are derived from truncated polynomial basis by repeated application of the difference matrix D_1[22]. As noted by the authors in [27], one of the advantages of the P-Spline formulation is that is decouples the order of regularization and B Spline basis. Typically D_1 regularization provides sufficient smoothness in our experiments [64].

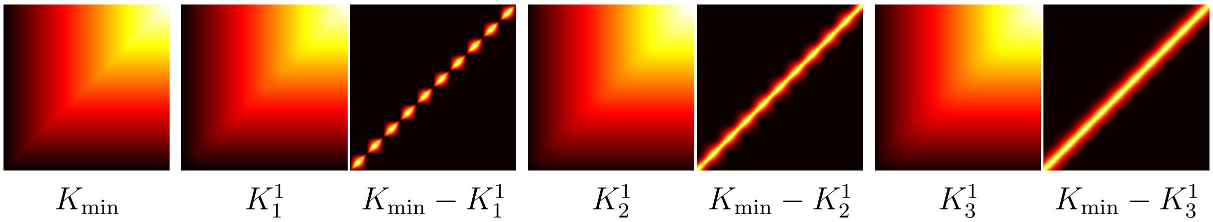


Figure 3.5: **Spline Kernels**. $K_{\min}(x, y), x, y \in [0, 1]$ along with K_r^1 for $r = 1, 2, 3$ corresponding to linear, quadratic and cubic B-Spline basis. Using uniformly spaced basis separated by 0.1, these kernels closely approximate the $\min$ kernel as seen in the difference image. The approximation is exact when $|x - y| > 0.1r$.

3.7 Conclusion

We have shown how to train additive classifiers motivated by our analysis in our earlier chapter, very efficiently – within a small multiple of the time required by the very fastest linear SVM training algorithms, shown both theoretically and in experiments. Our resulting additive classifiers consistently perform better than linear classifiers on vision tasks. In particular we can train our piece-wise linear additive classifier for pedestrian detection (based on spHOG features) which produces better results than than Dalal & Trigg's linear detector (based on HOG) in only 76.13 seconds, more than $100\times$ faster than the standard training. The proposed algorithm, as we show is similar to a P-Spline formulation and can also be used to derive learning algorithms for training classifiers in the max-margin and hinge loss framework.

3.8 Appendix

Min Kernel is Conditionally Positive Definite

A kernel $(x, y) \in A \times A \mapsto k(x, y) \in \mathbb{R}$ is said to be conditionally positive definite if it is symmetric (i.e., $k(x, y) = k(y, x)$) and

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0 \tag{3.26}$$

where $n \geq 1$, $x_1, x_2, \ldots, x_n$ are points in A and $(c_1, c_2, \ldots, c_n)$ is a vector in $\mathbb{R}^n$ such that

$$\sum_{i=1}^{n} c_i = 0; \tag{3.27}$$

Proof: Clearly $k(x, y) = \min(x, y)$ is symmetric, and we know that $\min(x, y)$ is positive definite [77] when $x, y \in \mathbb{R}^+$. Let $t = \min_i x_i$. Then we can write the sum as :

$$\begin{aligned}
& \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \left(\min(x_i - t, x_j - t) + t\right) && (3.28)\\
= \; & \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \min(x_i - t, x_j - t) + t \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j && (3.29)\\
= \; & \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \min(x'_i, x'_j) + t\left(\sum_{i=1}^{n} c_i\right)^2 && (3.30)\\
= \; & \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \min(x'_i, x'_j) \geq 0 \text{ , as } \forall i, x'_i \geq 0 && (3.31)
\end{aligned}$$

Chapter 4

Hough Transforms for Object Detection

A major bottleneck for a sliding window detectors we described in the first chapter, is the number of locations one has to look in an image to find the object. Even with the fastest classifier, we are limited in terms of speed because of the complexity of the search over pose. Various techniques have been proposed in the literature which try to alleviate the complexity issue, including looking at salient regions, coarse to fine search, branch-and-bound [58], etc.

The Hough transform [24] provides another way of dealing with the complexity issue and has been used for various pose estimation problems including shape detection [3]. The idea is to let easy to detect local parts vote for possible transformations of the object like translation, scale and aspect variation – one can use the peaks of the voting space for importance sampling of windows for further evaluation. Of particular interest is the implicit shape model [62] which is a formulation of the Hough transform where local parts probabilistically vote for locations of the objects. Combined with verification step, this approach has been used successfully for detection of objects like cars and pedestrians by [62]. In this setting, any technique that causes the voting space to better reflect the presence of the object has a direct impact on the speed and accuracy of this two stage classifier.

In this chapter we cast the Hough transform in a discriminative framework where each local part casts a weighted vote for the possible locations of the object center. We show that the weights can be learned in a max-margin framework which directly optimizes the classification performance. Compared to other approaches, including a naive-bayes weighing scheme based on how “representative” the local part is, the discriminative training framework leads to a significant reduction in the false positive rate on various datasets while simultaneously learning the important parts of the object. We call our approach max-margin hough transform or M^{2}HT.

On the ETHZ shape dataset [37] the M^{2}HT detector has a detection rate of 60.9% at 1.0 false positive per image compared to 52.4% using uniform weights and 54.2% using naive-bayes weights. On UIUC car dataset [1] the M^{2}HT detector has half the false positive per image rate at 90% recall compared to the Hough detector based on both uniform and naive-bayes weights. The performance of M^{2}HT is also better than both on the INRIA horse dataset [39]. The voting step is fast and scales well with respect to pose variations. We combine this with a verification step using a standard SVM classifier, which then finds the location of the objects by doing a local search around the proposed regions. Our two stage classifier achieves a detection rate of 91.9% at 0.3 false positive per image (FPPI) on the ETHZ shape dataset, a significant improvement over the state of the art, while running the verification step on two orders of magnitude fewer windows than in a sliding window approach. On UIUC cars we obtain a performance of 97.5% at equal error rate, while having to run the verification step on only 10 windows per image. On INRIA horse dataset [37] the overall detector has a recall of 85.27% at 1.0 FPPI, almost the same as sliding window detector while having to consider up to two orders of magnitude fewer windows per image.

The rest of the chapter is structured as follows. We present an overview of the probabilistic Hough transform in Section 4.1. In Section 4.2, we cast the voting process in a discriminative framework and outline the max-margin formation of the problem. The overall detection strategy is described in Section 4.3. In section 4.4 we present our experiments on various datasets. Conclusions and directions of future work are presented in Section 4.5.

4.1 Probabilistic Hough Transform

Let f_i denote the feature observed at a location l_i, which could be based on the properties of the local patch around l_i. Let $S(\mathcal{O}, \mathbf{x})$ denote the score of object $\mathcal{O}$ at a location $\mathbf{x}$. Here $\mathbf{x}$ denotes pose related properties like position, scale, aspect ratio, etc. Let C_i denotes the i'th codebook entry of the vector quantized space of features f. The implicit shape model [62] framework obtains the overall score $S(\mathcal{O}, \mathbf{x})$ by adding up the individual probabilities $p(\mathcal{O}, \mathbf{x}, f_j, l_j)$ over all observations, i.e.,

$$\begin{aligned} S(\mathcal{O}, \mathbf{x}) &= \sum_j p(\mathcal{O}, \mathbf{x}, f_j, l_j) \\ &= \sum_j p(f_j, l_j) p(\mathcal{O}, \mathbf{x} | f_j, l_j) \end{aligned}$$

Assuming a uniform prior over features and locations and marginalizing over the codebook entries we get,

$$\begin{aligned} S(\mathcal{O}, \mathbf{x}) &\propto \sum_j p(\mathcal{O}, \mathbf{x} | f_j, l_j) \\ &= \sum_{i,j} p(C_i | f_j, l_j) p(\mathcal{O}, \mathbf{x} | C_i, f_j, l_j) \end{aligned}$$

One can simplify this further using that fact that $p(C_i | f_j, l_j) = p(C_i | f_j)$ because the codebook entries are matched based on appearance only and the distribution $p(\mathcal{O}, \mathbf{x} | C_i, l_j, f_j)$ depends only on the matched codebook entry C_i and l_j.

$$\begin{aligned} S(\mathcal{O}, \mathbf{x}) &\propto \sum_{i,j} p(C_i|f_j) p(\mathcal{O}, \mathbf{x}|C_i, l_j) \\ &= \sum_{i,j} p(C_i|f_j) p(\mathbf{x}|\mathcal{O}, C_i, l_j) p(\mathcal{O}|C_i, l_j) \end{aligned}$$

The first term is the likelihood that the codebook entry C_i generated the feature f. We base this on the distance of the codebook entry to the feature as follows:

$$p(C_i|f) = \begin{cases} \frac{1}{Z}\exp(-\gamma d(C_i, f)) & \text{if } d(C_i, f) \leq t \\ 0 & \text{otherwise} \end{cases}$$

Where Z is a constant to make $p(C_i|f)$ a probability distribution and γ, t are positive constants. The second term is the probabilistic Hough vote for the location of the object, which can be estimated during training time by observing the distribution of the locations of the codebook activations relative to the object center. In our experiments we maintain a binned estimate of $p(\mathbf{x}|\mathcal{O}, C_i, l_j)$ by discretizing the space of relative locations of the object. The third term is the weight of the codebook entry emphasizing how confident we are that the codebook entry C_i at location l_j matches the object as opposed to background. Assuming that the probability $p(\mathcal{O}|C_i, l)$ is independent of the location (location invariance) we have a simple way of estimating this using both positive and negative examples as follows :

$$\begin{aligned} p(\mathcal{O}|C_i, l) &= p(\mathcal{O}|C_i) \\ &\propto \frac{p(C_i|\mathcal{O})}{p(C_i)} \end{aligned}$$

Here, $p(C_i|\mathcal{O})$ is the relative frequency of the codebook entry C_i on the object features , while $P(C_i)$ is the relative frequency on both negative and positive training images. We refer to this as naive-bayes weights, as the weight is set independently for each codebook entry.

4.2 Max-Margin Hough Transform

The overall procedure in the previous section can be seen as a weighted vote for object locations over all codebook entries C_i. In this section we will show how to learn these weights w_i in a discriminative manner which directly optimizes the classification performance. The key idea is to observe that the score of the $S(\mathcal{O}, x)$ is a linear function of $p(\mathcal{O}|C_i)$ (making the similar location invariance assumption that $p(\mathcal{O}|C_i, l) = p(\mathcal{O}|C_i)$). One can see this readily from the following :

$$\begin{aligned} S(\mathcal{O}, x) &\propto \sum_{i,j} p(x|\mathcal{O}, C_i, l_j) p(C_i|f_j) p(\mathcal{O}|C_i, l_j) \\ &= \sum_{i,j} p(x|\mathcal{O}, C_i, l_j) p(C_i|f_j) p(\mathcal{O}|C_i) \\ &= \sum_{i} p(\mathcal{O}|C_i) \sum_{j} p(x|\mathcal{O}, C_i, l_j) p(C_i|f_j) \\ &= \sum_{i} w_i \times z_i \\ &= \mathbf{w}^T \mathbf{z} \end{aligned}$$

Where z_i is given by the following equation:

$$z_i = \sum_{j} p(\mathbf{x}|\mathcal{O}, C_i, l_j) p(C_i|f_j) \tag{4.1}$$

For a given object location and identity, the summation over j is a constant and is only a function of the observed features, locations and the estimated distribution over the centers for the codebook entry C_i. Thus suggests a discriminative training algorithm that finds weights that maximize the score S on correct object location over incorrect ones. Unlike the earlier method of estimating w_i based just on codebook activations, we have the ability to additionally use the conditional distribution of the object centers to learn the weights. We formalize our training algorithm in the next section as well as present experiments to validate our approach.

4.2.1 Discriminative Training

Given a set of training examples, a set of positive object locations and negative ones $\{(y_i, l_i)\}_{i=1}^N$, where $y_i \in \{+1, -1\}$ is the label and l_i is the location of the i'th training instance. Typically we pick the negative instances by finding the peaks in the voting space on negative training images. The first stage corresponds to the feature computation which computes the contribution z_k of each codebook center C_k to the score of the object location. This is done by carrying forward the voting process and adding up the votes for l_j from each feature f_j according to the Equation 4.1. Let $\mathbf{a}_i = [z_1 z_2 \ldots z_K]$, denote the vector of these coefficients. Thus the score assigned by the model to the instance i is $\mathbf{w}^T \mathbf{a}_i$. Weights are learned by maximizing this score on correct locations of the object over incorrect ones. In order to to be robust to outliers and avoid over-fitting, we propose a max-margin formulation of the problem leading to the following optimization problem,

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{T}\xi_i \\ s.t.: \quad & y_i(\mathbf{w}^T\mathbf{a}_i + b) \geq 1 - \xi_i, \xi_i \geq 0, \forall i = 1, 2, \ldots, N \\ & \mathbf{w} \succeq 0 \end{aligned}$$

The optimization is similar to optimization problem of a linear Support Vector Machine [20], with the additional positivity constraint on the weights. We use a standard off the shelf optimization package called CVX [43] for carrying out the optimization.

4.3 Overall Detection Strategy

The overall detector works in two stages, first the M^{2}HT detector is run on the image and a small set of regions most likely to contain the object of interest is found. Next a verification classifier based on a SVM finds the true location and score of the object by doing a local search around each region by sampling nearby scales and translations. Instead of densely sampling windows all over the image, the Hough step lets us concentrate around the regions most likely to contain the object and at the same time allowing us to implicitly sample a wider set of transforms including aspect ratio. We briefly describe the details of both the steps in the next two sections.

4.3.1 M^{2}HT Detector

Weights are learned on codebooks generated using k-means clustering of Geometric Blur (GB) features [5] sampled uniformly along the edges in an image. We choose four orientation directions and the outer radius of the GB feature typically as 20% of the object height, giving us a good tradeoff between the repeatability and discriminativeness of the features. On the positive set of training images, the relative locations of the center of the object is recorded during training time and a binned approximation of this distribution is maintained. We perform a second iteration over the positive and negative images and compute the contribution of each cluster center to the score of the true location of the object. Negative examples are obtained by first running the hough detector on negative images and finding the peaks in the voting space above a threshold. Once again the contributions of the cluster centers to score of these negative locations are computed. These contributions, which are the features (a_i) in the previous section, are then used to learn the weights of the codebook entries using our max-margin formulation.

4.3.2 Verification Classifier

We train a SVM classifier using the pyramid match kernel [46, 61] on histograms of oriented gradients as features. Gradients are computed using response to $[-1\ 0\ 1]$ and $[-1\ 0\ 1]^T$ filters and histograms in 9 orientations are computed. The image is divided into grids of increasing resolutions for 4 levels, and histograms from each level are weighted according to the equation $w_l = 2^{l-1}$, $l = 1$ being the coarsest scale, and concatenated together to from one long feature vector. A SVM is trained by using histogram intersection as the kernel on these features. We refer to this as the IKSVM classifier. On all datasets training is done by scaling the positive instances of the category to the median aspect ratio and a number of windows sampled from negative training images serve as negative examples. To detect an instance of an object in the sliding window mode the classifier is run at various location and scales by keeping the aspect ratio of the image fixed. Search over aspect ratio adds another factor to the run time, so we do not do it. A simpler baseline would have been to use a linear kernel, but others[37] have noticed that on ETHZ shape dataset, linear SVM does not give full recall. We use the speedup method for IKSVM classification proposed in [66] which makes the runtime of the classifier essentially equivalent to a linear SVM.

4.4 Experimental Results

In all our experiments we would like to verify two things: **(1)** The M^{2}HT detector should have a better performance compared to Houghtransform detector using uniform weights or naive-bayes weights. Quantitatively this means a lower false positive rate for the same recall. **(2)** The performance of the two stage M^{2}HT + IKSVM detector should be comparable to the IKSVM detector in the sliding window mode, while having to evaluate the IKSVM detector on a significantly fewer locations. Additionally, if the Houghtransform votes for pose parameters we would like to see that the two stage detector is robust to these pose changes. Finally the overall approach should compare favorably to other approaches in the literature both in terms of accuracy and space-time complexity. To validate our claims, we present our experiments on the ETHZ shape, UIUC cars and INRIA horses dataset.

4.4.1 ETHZ Shape Dataset

The first dataset we report our results on is the ETHZ Shape Dataset. It contains 255 test images and featuring five shape-based classes (apple logos, bottles, giraffes, mugs, and swans). For training we use half the positive examples and an equal number of negative examples equally distributed among the rest of the categories. All the remaining examples are used for testing. We use the same training and test splits used by authors of [37] for a fair comparison.

M^{2}HT Detector Training: For the hough training step all ground truth bounding boxes of a particular category are scaled to a height of 96 pixels, while keeping the aspect ratio fixed. A separate codebook is learned for each category using k-means clustering with $k = 400$ centers. For categories like mugs and giraffes the aspect ratio varies widely so we train the hough detector to vote for both the center and aspect ratio of the object. We maintain a binned approximation of distribution of the location of the center with bin width=10px, bin height=10px and aspect width=0.1.

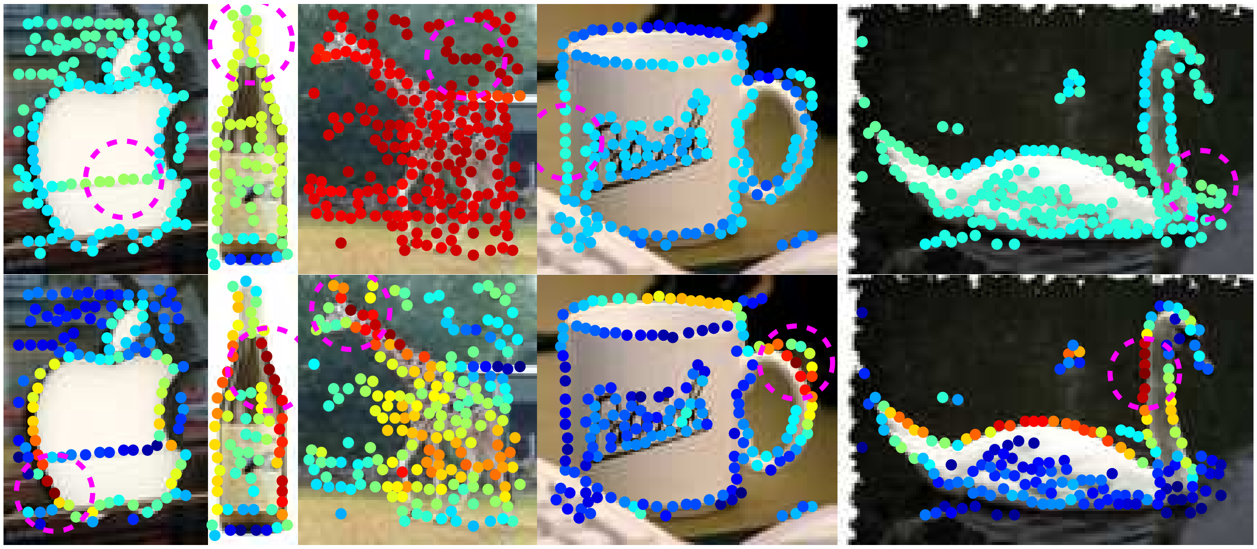


Figure 4.1: Learned weights on various categories of the ETHZ shape dataset. **(top row)** naive-bayes weights, **(bottom row)** M^{2}HT weights. In each image, the most important part is circled with radius equal to the outer radius of the gb descriptor. Notice how the salient parts like the handles on mugs, the neck and leg region of the giraffe are assigned high weights, while the background clutter is ignored. The naive-bayes weights are strongly affected by rare structures in the background. For each category the colors represent the strength of the weights (red is higher) and are on the same scale for both naive-bayes and M^{2}HT.

We then run the max-margin training procedure described in Section 4.3.1 to learn the weights for the codebook entries. Figure 4.1, shows the learned weights for various categories. The learning framework assigns high weights to parts of the object which are both characteristic and are good predictors of the object location, while simultaneously ignoring the background clutter in images. Notice that we only use the ground truth bounding box for training, which includes a significant amount of background clutter for categories like giraffes and swans. The naive-bayes weights are strongly affected by rarely occurring structures in the background.

M^{2}HT Detector Results: Table 4.4 shows the detection rate at 1.0 FPPI for various weights. The M^{2}HT detector alone has a detection rate of 60.9% at 1.0 false positive per image (FPPI) compared to 54.2% using naive bayes and 52.4% using uniform weights. In our experiments, increasing the number of codebook entries improves the performance of the max-margin weights, but due to the small number of training examples (as low as 16 for swans), the conditional distribution of the center over both scale and aspect ratio cannot be reliably computed, so we do not train with more cluster centers. At about 30 windows per image we have almost full recall for using any of the hough detectors, and the performance of the overall detector is similar with all the methods. This is at least two orders of magnitude less than the number of windows considered by a sliding

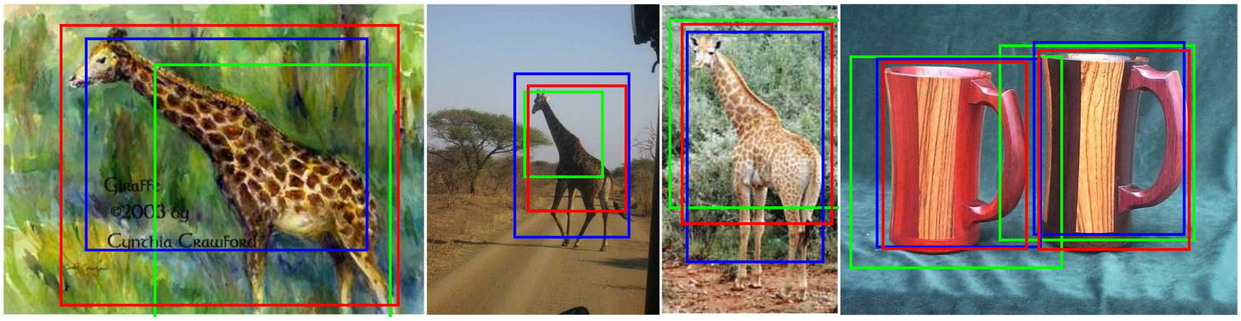


Figure 4.2: Sample detections where the bounding box predicted by the M^{2}HT+IKSVM detector (blue) is closer to the groundtruth (red) than the IKSVM detector when used in the sliding window mode (green).

window detector

Overall Detector Results : Figure 4.5 compares the results of the IKSVM detector used in the sliding window mode at a fixed aspect ratio v.s. the M^{2}HT + IKSVM detector. Performance are reported using the PASCAL criterion, i.e., a detection is counted as correct if the intersection over union of the detection rectangle and ground truth rectangle is greater than 0.5. The IKSVM baseline is quite good and achieves a detection rate of 87.7% (0.3 FPPI) and 88.48% (0.4 FPPI). Sampling the nearby scales and locations around the regions proposed by the hough transform leads to an improved detection rate of 91.9%(0.3 FPPI) and 93.2% (0.4 FPPI). Including the windows of the local search is still at least two orders of magnitude less than a sliding window detector for a similar dense sampling. Additionally we implicitly sample over aspect ratios because the hough detector proposes regions of various aspect ratios. This leads to a significant improvement over the baseline IKSVM detector for the giraffe and mugs category, where the aspect ratio varies widely. Figure 4.2 show some images where the bounding box of the two stage classifier fits the object better than that of the sliding window classifier. Figure 4.3 shows examples of detections and missed detections for various categories. Our results are significant improvement over previous best results 61.4% of KAS [37] and 67.1% of TPS-RPM [38] at 0.3 FPPI as shown in Figure 4.4. The results of TPS-RPM are not directly comparable as the authors report numbers using a five-fold cross validation, but still is better considering that the average variation in accuracy over trials is about 9% as observed by the authors.

4.4.2 UIUC Cars

This database was collected at UIUC [1] and contains images of side views of cars. The training set consists of 550 car and 500 non-car images. We test our methods on the single scale image test set which contains 170 images with 200 cars. The images are of different sizes themselves but contain cars of approximately the same scale as in the training images.

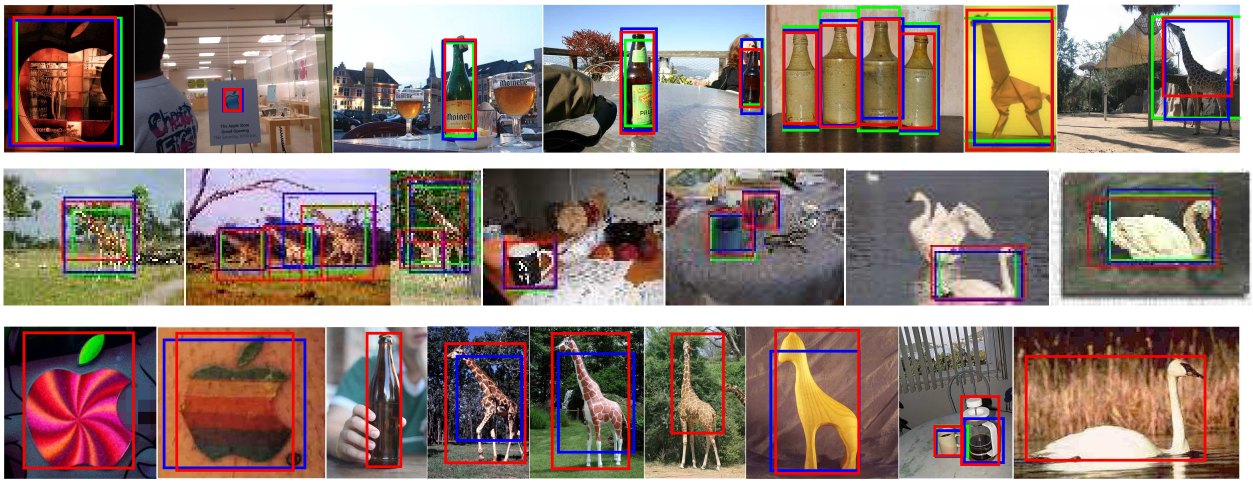
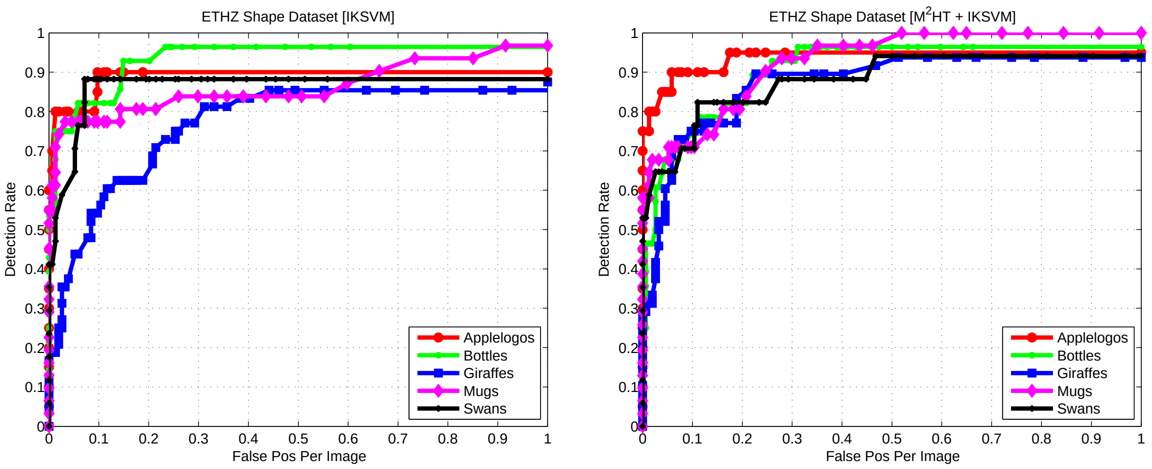


Figure 4.3: **(Rows 1-2)** Example detections on the ETHZ shape dataset using M^{2}HT+IKSVM detector (blue), IKSVM detector used in sliding window mode (green) overlaid with groundtruth (red). **(Row 3)** Example images where at least one of the two detectors fails.

	Hough Detector (1.0 FPPI)			**Overall Detector (0.3/0.4 FPPI)**			
Category	uniform	nbayes	max-m	IKSVM	M^{2}HT + IKSVM	KAS	TPS-RPM*
Applelogos	70.0	70.0	85.0	90.0/90.0	95.0/95.0	50.0/60.0	77.7/83.2
Bottles	62.5	71.4	67.0	96.4/96.4	92.9/96.4	92.9/92.9	79.9/81.6
Giraffes	47.1	47.1	55.0	79.1/83.3	89.6/89.6	49.0/51.1	40.0/44.5
Mugs	35.5	35.5	55.0	83.9/83.9	93.6/96.7	67.8/77.4	75.1/80.0
Swans	47.1	47.1	42.5	88.2/88.2	88.2/88.2	47.1/52.4	63.2/70.5
Average	52.4	54.2	60.9	87.5/88.4	91.9/93.2	61.4/66.9	67.1/71.9

Figure 4.4: Performance of various algorithms. All the results are reported using the PASCAL criterion (Intersection/Union ≥ 0.5). The hough detector alone has a detection rate of 60.9% at 1 false positive per image (FPPI) an improvement of 6.7% over the naive bayes weights and 8.5% over uniform weights. The IKSVM classifier when used in sliding window mode has a average detection rate of 87.5% at 0.3 FPPI. By combining with the hough detector, the performance improves to 91.9% at 0.3 FPPI. There are significant improvements in the giraffe and mugs category, which have high variation in aspect ratio. This is a significant improvement over previous best results 61.4% of KAS [37] and 67.1% of TPS-RPM [38] at 0.3 FPPI. *The results of TPS-RPM are not directly comparable as the authors report numbers using a* 5*-fold cross validation.**



(a) (b)

Figure 4.5: Detection plots using the IKSVM detector and M^{2}HT + IKSVM on ETHZ shape dataset. All results are reported using the PASCAL criterion. **(a)** The IKSVM baseline achieves a detection rate of 87.7% (0.3 FPPI) and 88.48% (0.4 FPPI). **(b)** Sampling densely on the regions proposed by the M^{2}HT detector leads to an improved detection rate of 91.9%(0.3 FPPI) and 93.2% (0.4 FPPI).

M^{2}HT Detector Training: Similar to the ETHZ dataset we compute GB features on both the positive and negative windows by uniformly sampling points on the edges and learn a codebook using k-means with $k = 100$. For every cluster the conditional distribution of the center of the object is maintained as binned approximation with a bin width=4 and bin height=4. This is a fairly dense sampling given that the training images are 100×40, so we spatially smooth the bins to avoid any artifacts. A second loop over the training images is done to compute the features for the max-margin training. Figure 4.6 shows the learned weights on this dataset using max-margin training and naive bayes. Notice how the learning framework emphasizes the regions near the bottom of the car, which are both repeatable and good predictors of the object center.

M^{2}HT Detector Results: Figure 4.8 plots the recall as a function of the false positive per image on for various learning schemes. At 90% recall the M^{2}HT detector has about half as many false positives per image than the hough detector using uniform weights and naive bayes weights. Considering only the top 10 windows per image and running the IKSVM verification step leads to performance of 97.5% at equal error rate an improvement of 1.74% over IKSVM detector using the sliding window detector alone, while having to consider $10\times$ fewer regions per image. The increased precision is because the IKSVM classifier densely samples windows near the most likely locations of the object, while being able to discard a large fraction of the regions in the image not containing an object. Our method compares favorably to other methods in the literature as shown

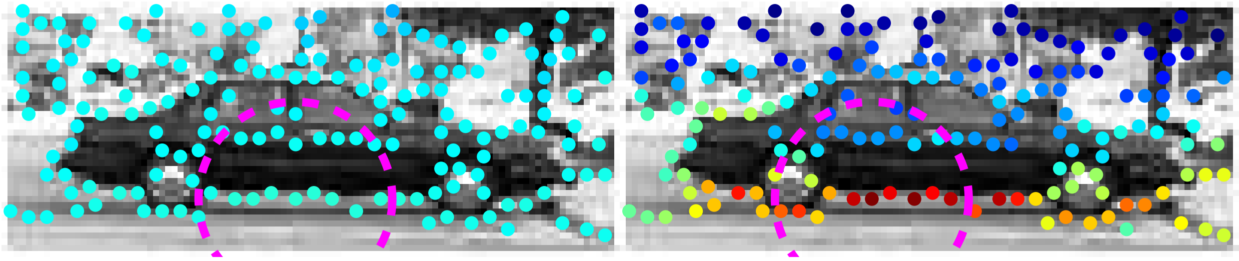


Figure 4.6: Learned weights on UIUC car dataset : **(left)** naive-bayes weights **(right)** M^{2}HT weights (red is higher). In each image the most important part is circled with radius equal to the outer radius of the gb descriptor. Notice how the features at the bottom of the car and near the wheels are emphasized which are both repeatable and good predictors of the object center. Both weights are on the same scale.

in Figure 4.7.

4.4.3 INRIA Horses

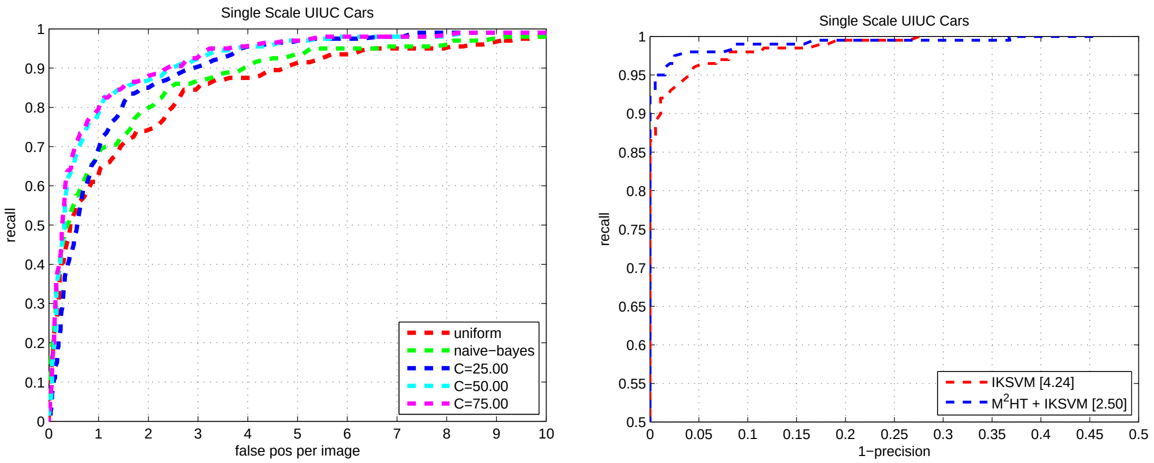
The INRIA horse dataset collected by Jurie and Ferrari, consists of 170 images with one or more side-views of horses and 170 images without horses. Horses appear at several scales, and against cluttered backgrounds. We use the same training and test split of [37] consisting of 50 positive and 50 negative examples for training and the rest for testing.

M^{2}HT Detector Training: We learn a codebook with k-means with $k = 500$ and learn weights for each cluster center. Figure 4.10 shows the weights learned for various features using the max-margin training and naive-bayes scheme. The IKSVM classifier is trained by scaling all the ground truth bounding boxes to the median aspect ratio of all horses in this dataset.

M^{2}HT/Overall Detector Results: Figure 4.11 shows the performance of the M^{2}HT detector and the overall detector. The M^{2}HT detector outperforms both the naive-bayes and the uniform weights. The overall performance of the M^{2}HT + IKSVM detector is same as the IKSVM detector while having to consider only 25 windows per image, which is up to two orders of magnitude fewer than the sliding window classifier. At 1.0 false positive per image we have a detection rate of 85.27% for M^{2}HT + IKSVM and 86% for IKSVM compared to previously published results of 80.77%[37] and 73.75% [38]. The results of [38] are however not directly comparable as the authors report results using 5-fold cross validation.

Method	Performance
IKSVM	95.76 %
M^{2}HT + IKSVM	97.5 %
Agarwal & Roth [1]	79 %
Garg et al. [41]	88 %
Fregus et al. [35]	88.5 %
ISM [62]	97.5 %
Mutch & Lowe [76]	99.6 %
Lampert et al. [58]	98.5 %

Figure 4.7: Performance at Equal Error Rate on UIUC Single Scale Cars for various methods. The M^{2}HT + IKSVM detector has an improvement of 1.74% over the IKSVM baseline in the sliding window mode, while having to consider only 10 windows per image. Our method compares favorably to other methods in the literature.



(a) (b)

Figure 4.8: (a) Detection plots on UIUC car dataset for various values of the learning parameter C using the max-margin hough training. At 90% recall the false positive rate is only about half compared to both uniform weights and naive bayes weights. (b) Combining with the verification step using the IKSVM classifier. Only the top 10 windows per image are considered, which is about $10\times$ fewer than the number of windows considered by a sliding window detector. By sampling around the regions proposed by the hough detector there is an improvement of 1.74% over the sliding window detector.

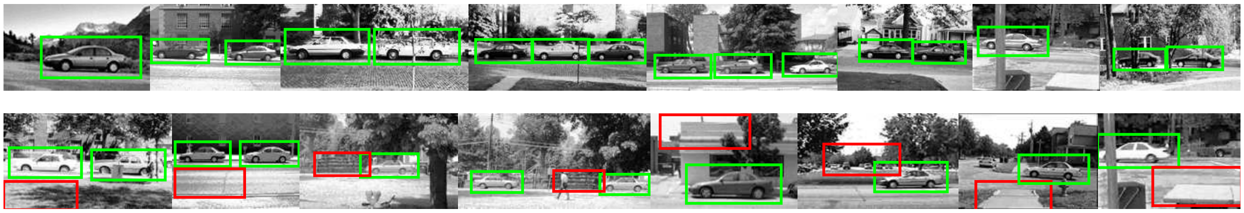


Figure 4.9: Example detections using the M^{2}HT + IKSVM detector on UIUC cars dataset. Correct detections are shown in green and incorrect detections are show in red.

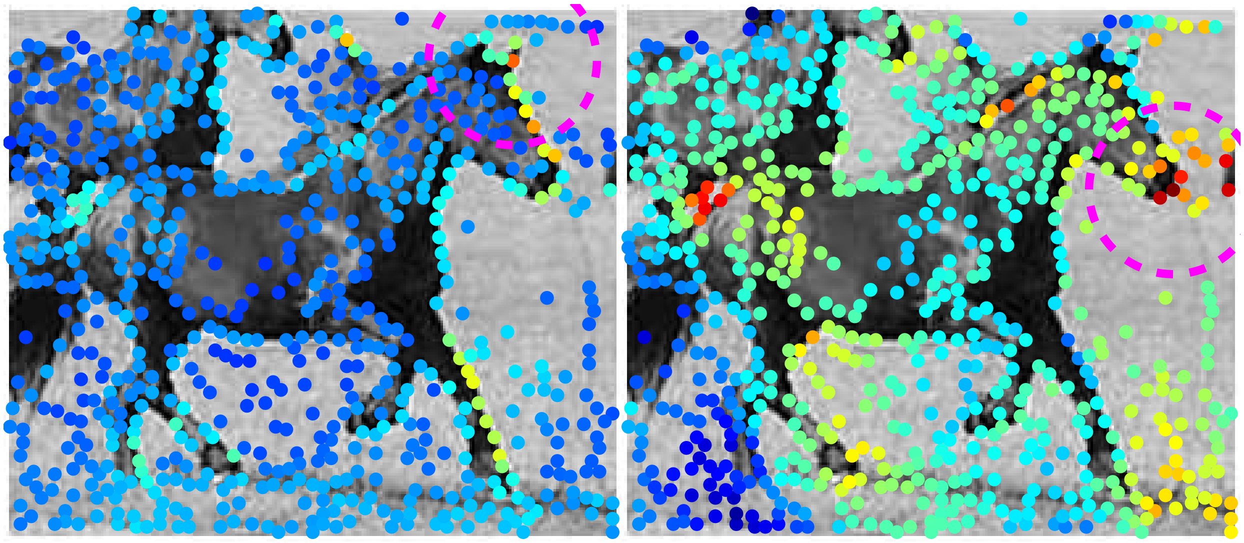


Figure 4.10: Learned weights on INRIA horses dataset. **(left)** naive bayes weights, **(right)** M^{2}HT weights (red is higher). In each image, the most important part is circled with radius equal to the outer radius of the gb descriptor.

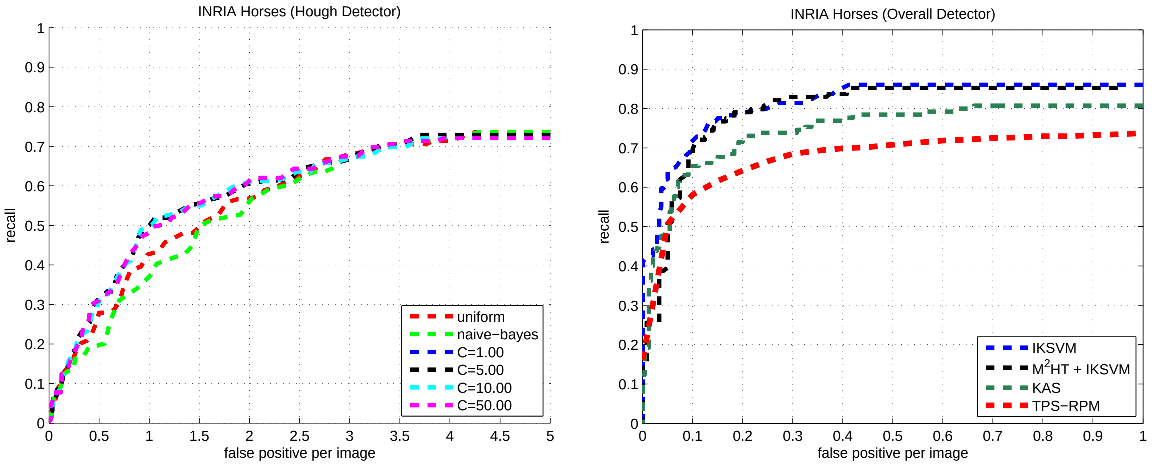


Figure 4.11: Detection plots on INRIA horse dataset using the PASCAL criterion. **(left)** Comparison of M^{2}HT detector for various choices of the learning parameter C vs. uniform weights and naive-bayes weights. The M^{2}HT detector consistently outperforms both. **(right)** Overall detections results using IKSVM and two stage M^{2}HT + IKSVM. Performance of M^{2}HT + IKSVM is similar to IKSVM while having to consider only 25 windows per image on average, which is up to two order of magnitude fewer than in sliding window approach. At 1.0 false positive per image we have a detection rate of 85.27% for M^{2}HT + IKSVM and 86% for IKSVM compared to previously published results of 80.77% (KAS) [37] and 73.75%(TPS-RPM) [38] *(Note that the results of TPS-RPM are not directly comparable as the authors report numbers using* 5*-fold cross validation.)*

4.5 Conclusion

We cast Hough transform in discriminative framework to learn weights on local parts that participate in a voting based detection scheme. The learned weights outperform both the “uniform” and “naive-bayes” weights on various datasets. Our max-margin problem formulation is convex and can be optimized using off the shelf optimization packages such as CVX.

In addition we show that the two stage M^{2}HT + IKSVM detector has better runtime complexity than a sliding window detector and at the same time is more robust to pose variations. Our approach leads to the state the art results on ETHZ shape dataset and competitive results on the UIUC car and INRIA horse dataset.

Although in this chapter we focus on local parts that are generic, i.e., obtained by k-means clustering of local patches, this need not be the case. We introduce more semantic parts in the next chapter based on a novel representation called “poselets”, which are trained discriminatively using additional annotations. These parts can be used for person detection [13], as well as predicting the location of various joints [12] using the Hough voting framework. For these problems, the proposed max-margin formulation has led to improvements in accuracy.

Chapter 5

Supervised Models for Object Recognition

Describing visual categories is at the heart of most recognition systems. The notion of categories itself has been the topic of debate among psychologists for centuries and there are many competitive theories – exemplar based models [71], prototypes [90] and the classical one based on a list of attributes which dates back to Aristotle (see [75], for a pleasant overview). For the task of visual categorization one has to deal with the additional variation because of the non-invertible camera projection (or the eye). Computer vision techniques that are based on variants of template matching have proved to the most robust approach to identify visual categories, but the question of what should one train a template of still remains.

In the object detection setting one has seen tremendous progress in detecting a rigid object from a fixed viewpoint, for example frontal faces [112], or pedestrians [21], but the general problem of detecting a category such as person is still far from being solved. For example, the current best method for person detection on the challenging PASCAL VOC dataset achieves an mean average precision of only about 50%. Part of the problem is that there is a significant variance in appearance not only because of changing viewpoint, but also because people appear in variety of clothing, are occluded and interact with objects around them in a variety of poses.

One may try to build templates each of the cases we wish to detect, but there are two problems with this approach, in the spirit of the exemplar model [71]. First, these templates have to be learned which subjects them to the usual bias-variance tradeoffs – more data is good, simple models are better. Second, during test one is reduced to a nearest neighbor like search, the complexity of which is high. Thus the exemplar model of visual recognition is at a disadvantage both from computational and model estimation point of view, though has recently been tried somewhat successfully recently for detection in [70].

The second approach is to build a representation which is of sufficiently high dimension that a classifier can learn the highly non-linear model of the appearance variation – more along the lines of a single prototype view [90]. Popularly known as multiple kernel learning [109], it has seen reasonable success in many object detection [111] and image classification benchmarks. In this setting it typical to have features of several thousand dimensions to learn a classifier for a visual category. This approach is extremely attractive – one does not have to deal with the invariances,

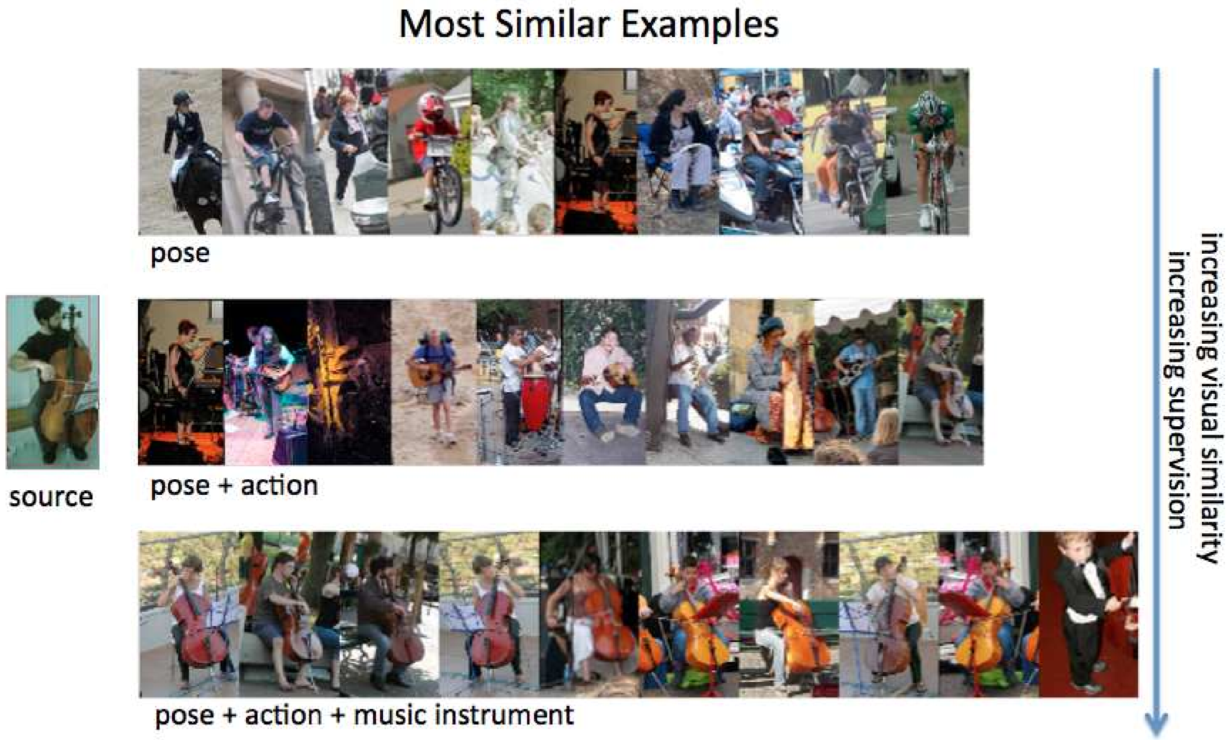


Figure 5.1: Most similar examples of the source image using (Top Row) pose only, (Middle Row) pose and action label, (Bottom Row) pose, action and subcategory label. From top to bottom the visual similarity increases at the expense of more supervision.

or describe the properties of the class as long as the feature space is sufficiently rich, and there is enough training data. However, where it fails is in providing interpretability of the learned model, making it harder to engineer richer models of categories. For example, if one were to now ask for the pose, or camera viewpoint or the kind of clothing of an instance, it would be hard to infer that from the classification function.

5.1 Supervised Learning of Categories

It appears that a middle ground between the prototype theory and exemplar theory of categories might be a good choice. Instead of having one prototype, we could have a few, much smaller than the number of instances. This may not suffer from the computational complexity issues of the exemplar model. At the same time because one of the prototypes could be thought of as an "average" of some number instances, one could benefit because the learning algorithm has more examples to look at. *But what should the prototypes be?*

The key question of finding the set of prototypes then becomes one of visual correspondence.

Any instance of a visual category can be thought to be in correspondence with at least one of the prototypes. As good as it may sound, the idea of visual correspondence is hard, subjective and task dependent. Categories form a hierarchy - at one level one may describe an object as a car, but at a finer level one could say the car is a Beetle or a Mini, or a red Mini at an even finer level.

Given the subjective and task dependent nature of problem, we aim to bootstrap the visual correspondence problem with additional supervision. We annotate a set of keypoints on all the objects within the category which provides us the coarsest level of correspondence. Prototypes can be thought of as representation of instances which share a common configuration of keypoints. To have a flexible, task dependent notion, we collect attribute labels which could correspond to say, the kind of clothing, action labels, or subcategories. These enable one to refine the notion of correspondence using both the keypoint correspondence and the attribute label.

As an example, Figure 5.1 top-row, shows examples of people sharing the same pose as the source image. It might be hard to learn an appearance model for all these instances because of the variety in the action type, interaction with objects, etc., which change the appearance. Figure 5.1 middle-row, shows examples that share the same pose and action, namely, playing instrument. The examples are more visually coherent but not as much as the the bottom row, since there are many kinds of musical instruments. Learning an appearance template here might be easiest in the last case, at the expense of more supervision and potentially more templates or prototyptes to recognize the category.

Collecting such annotation can be time consuming, but this has become increasingly efficient and cost effective due to “crowdsourcing”. Services like Amazon Mechanical Turk, which matches workers to micro tasks, makes it easy to collect large amounts of annotations relatively quickly. In the next chapter I’ll describe how we collected annotations of various kinds ranging from keypoints, segmentation masks, 3D pose to attribute labels.

The idea of using keypoints for aligning examples using 3D keypoint annotations was proposed by Bourdev and Malik [12] and subsequently modified to deal with simpler 2D keypoint annotations by Bourdev, Maji and Malik, in [13], leading to the state of the art person detector on the PASCAL VOC challenge at the time of publication. Subsequently, models of segmentation [14], pose estimation, action recognition [67] and attribute recognition [11], have been built on top of that framework. In this chapter we focus on the task of pose estimation and action recognition of people from static images.

5.2 Pose and Action Recognition from Still Images

We can say a fair amount about the people depicted in Figure 5.2 – the orientations of their heads, torsos and other body parts with respect to the camera, whether they are sitting, standing, running or riding horses, their interactions with particular objects, etc. And clearly we can do it from single image, video is helpful but not essential, and we do not need to see the whole person to make these inferences.

A classical way to approach the problem of action recognition in still images is to recover the

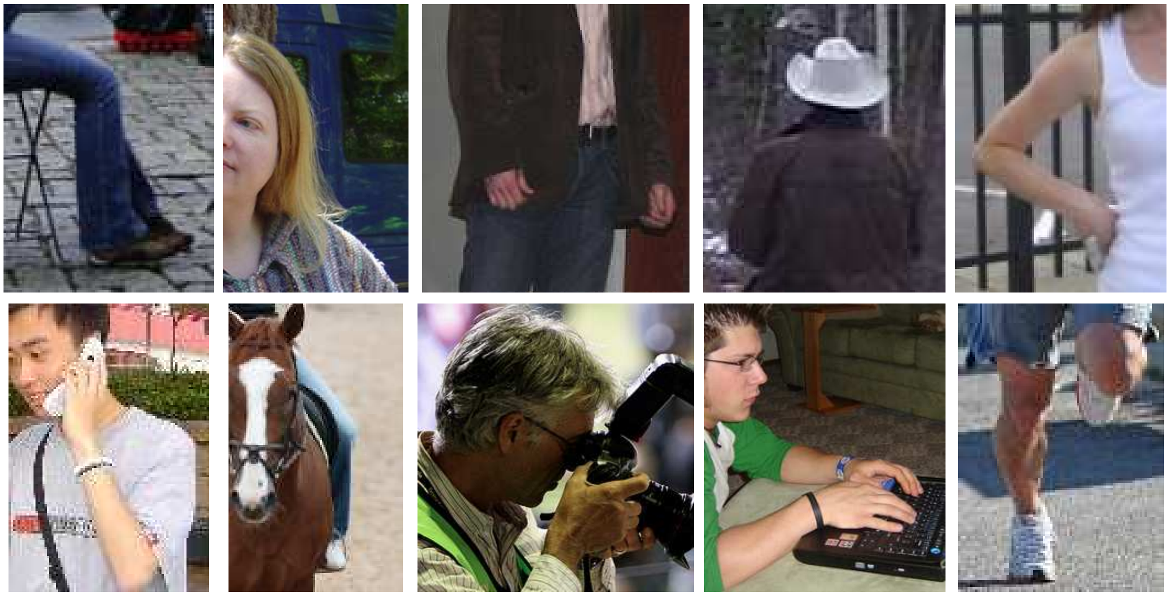


Figure 5.2: Pose and action is revealed from all these patches.

underlying stick figure [34, 86]. This could be parameterized by the positions of various joints, or equivalently various body parts. In computer graphics this approach has been a resounding success in the form of various techniques for “motion capture”. By placing appropriate markers on joints, and using multiple cameras or range sensing devices, the entire kinematic structure of the human body can be detected, localized and tracked over time [101]. But when all we have is a single image of a person, or a part of a person, not necessarily at high resolution, in a variety of clothing, the task is much harder. Research on pictorial structures [34, 86] and other techniques [88] for constructing consistent assemblies of body parts has made considerable progress, but this is very far from being a solved problem.

We take the position that recovering the precise geometric locations of various body parts is trying to solve a harder intermediate problem than necessary for our purposes. We advocate instead the use of a representation, the “poselet activation vector”, which implicitly represents the configuration of the underlying stick figure, and inferences such as head and torso pose, action classification, can be made directly from the poselet activation vector.

We can motivate this by a simpler example. Consider the problem of inferring the pose of a face with respect to camera. One way of doing it is as an explicit 2D to 3D geometric problem by finding the locations of the midpoints of the eyes, nose etc, and solve for the pose. Alternatively one can consider the outputs of various face detectors - one tuned to frontal faces, another to three-quarter view faces, another to faces in profile. The responses of these detectors provide a distributed representation of the pose of the face, and one can use an “activation vector” of these responses as the input to a regression engine to estimate pose. In biological vision, strategies such as these are common place. Color is represented by a response vector corresponding to three cone types, line orientation by the responses of various simple cells in V1, and indeed neurons have been found in macaque inferotemporal cortex which show differential responses to faces at different orientations, suggesting a distributed representation there as well.

In order to generalize this strategy to the human body, we must deal with its articulated nature. Different parts can be in different configurations, and occlusion can result in only some parts being visible. In addition one needs to deal with the variation in aspect due to changes in camera direction. Poselets, introduced by Bourdev and Malik [12] and further developed in Bourdev et al. [13] for person detection and segmentation provide a natural framework.

We show that the poselet activation vector, which represents the degree to with each poselet is present in the image of a person, provides a distributed representation of pose and appearance. We use it to estimate the 3D orientation of the head and torso of people in the challenging PASCAL VOC 2010 person detection dataset [28]. This dataset is significantly hard where the current state of the art methods achieve detection performance only about 50%. Our approach achieves an error of 26.3° across views for the head yaw and matches the "human error rate" when the person is front facing.

Action recognition from still images can benefit from this representation as well. Motion and other temporal cues which have been used for generic action recognition from videos [96, 100, 42, 25], are missing in still images which makes it a difficult problem. In this setting the pose and appearance of the person provides valuable cues for inferring the action. For example as seen in Figure 5.3, certain actions like walking and running are associated with specific poses while people riding bikes and horses have both a distinctive pose and appearance.

Actions often involve interactions with other objects and one can model these interactions to disambiguate actions [121]. In addition context based on actions of other agents in the scene can provide valuable cues as well [59]. For example, certain activities like marathon events or musicians playing in a concert, are group activities and it is likely that everyone in the scene is performing the same action.

The rest of the paper is structured as follows: we begin with a review of work in the area of action recognition and pose estimation in Section 5.3. In Section 5.4, we describe how we construct the poselet activation vector for a given person in an image. We present experiments on 3D pose estimation of people in the PASCAL VOC 2010 people detection dataset in Section 5.5. Finally we report results on the recently introduced PASCAL VOC 2010 action classification dataset in Section 5.6 and conclude in Section 5.7.

5.3 Previous Work

The current work draws from the literature of two active areas in the computer vision – pose estimation and action recognition. We briefly describe some without any hope of doing justice to either of the areas.

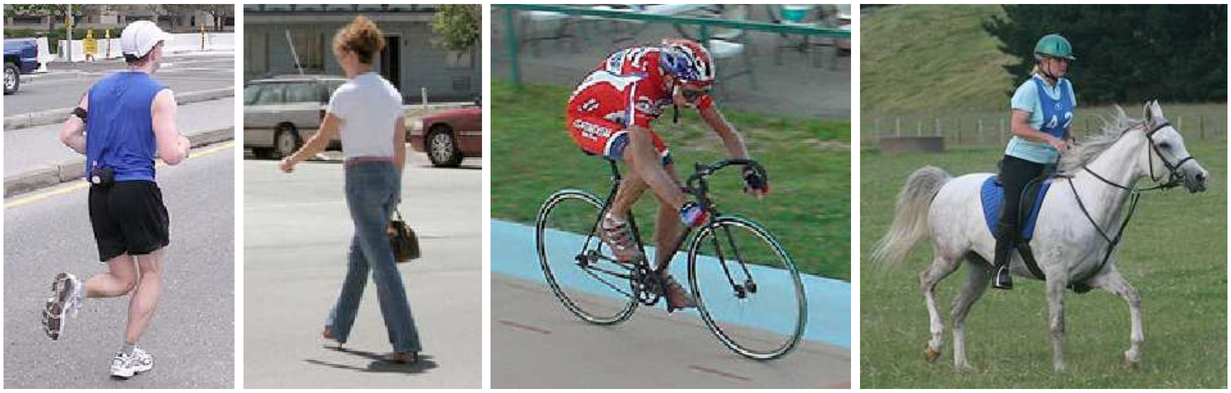


Figure 5.3: Pose and appearance variation across actions.

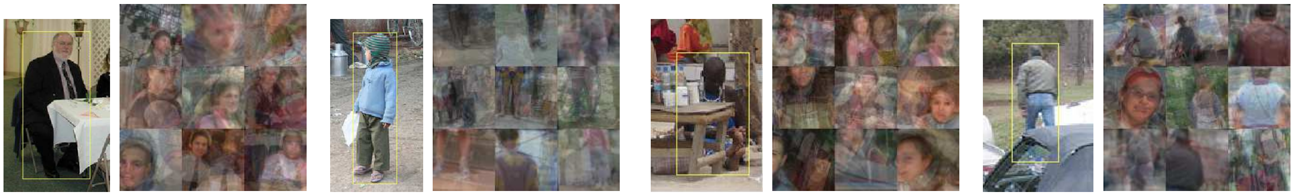


Figure 5.4: **Our distributed representation of pose using poselets.** Each image is shown with the top 9 active poselets consistent with the person in the image (shown by their average training examples). Occlusion, variations in clothing, clutter, lack of resolution in images makes the pose estimation a hard problem and our representation is robust to these.

Human pose estimation from still images. Pictorial structures based algorithms like that of [34, 88, 86, 36] deal with the articulated nature of humans by finding body parts like limbs and torsos and constructing the overall pose using the prior knowledge of human body structure. Though completely general, these methods suffer when the parts are hard to detect in images. Another class of methods work by assuming that the humans appear in backgrounds which are easy to remove, and in such cases the contour carries enough information about the pose. This includes the shape-context based matching of silhouettes in the work of [73], the work of [97] where approximate nearest neighbor techniques are used to estimate the pose using a large dataset of annotated images.

A common drawback of all these approaches is that they treat the task of pose estimation and detection separately. Pictorial structure based models often assume a rough localization of the person and fail when there is significant occlusion or clutter. In such a two-stage pipeline it would be helpful if the detector provides a rough estimate of the pose to guide the next step. We also believe that the detection algorithms need to have a crude treatment of pose in them. This is reflected by the fact that some of the best people detectors on the PASCAL VOC challenge namely the detector of Felzenszwalb et al. [33] and Bourdev et al. [13] are part based detectors which have some treatment of pose.

Action Recognition from video. Actions in this setting are described by some representation of its spatio-temporal signature. This includes the work of Blank et al. [7] and Shechtman and Irani [42], who model actions as space-time volumes and classification is based on similarity of these volumes. Schuldt et al. [96] and Laptev [60] generalize the notion of interest points from images to space-time volumes and use it to represent actions. Actions as motion templates has been explored in the work of Efros et al. [25], where actions are described as series of templates of optical flow. Other methods like [87, 122] are based on representations on the 2D motion tracks of a set of features over time.

Action recognition from still images. Humans have a remarkable ability to infer actions from a still image as shown in Figure 5.2. In this setting it is natural to build representations on top the output of a pose estimation algorithm. Due to the drawbacks of the current pose estimation algorithms, several approaches build pose representations that are more robust – Ikizler and Pinar [53] represent pose using a “histogram of oriented rectangle” feature which is the probability distribution of the part locations and orientations estimated using part detectors. Thurau and Hlavac [106] represent pose as a histogram of pose primitives. These methods inherit most if not all of the problems of pose estimation.

The closest in spirit to our approach is the work of Yang et al. [120], who also use a representation based on poselets to infer actions. Pose represented as a configuration of body part locations is expressed as a latent variable which is used for action recognition. Training and inference in the model amount to reasoning over these latent poses which are themselves inferred using a tree like prior over body parts and poselet detections. Unlike their approach we don’t have an explicit representation of the pose and use the “poselet activation vector” itself as a distributed representation. In addition, our poselets encode information from multiple scales and are not restricted to parts like legs and arms. In our experiments we found that such an over-complete representation greatly improves the robustness of the system. We show that linear classifiers trained on top of the poselet activation vector can be used for both 3D pose estimation of people in the challenging PASCAL VOC 2010 dataset and static image action recognition demonstrating the effectiveness of our representation.

5.4 Poselet Activation Vector

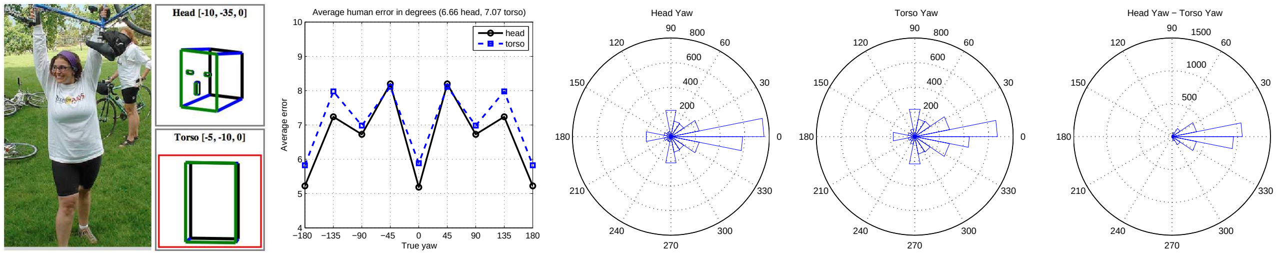
Our framework is built on top of poselets [12, 13] which are body part detectors trained from annotated data of joint locations of people in images. The annotations are used to find patches similar in pose space to a given configuration of joints. A poselet is a SVM classifier trained to recognize such patches. Along with the appearance model one can also obtain the distributions of these joints and person bounding boxes conditioned on each poselet from the annotations. Figure 5.11 shows some example poselets.

Given the bounding box of a person in an image, our representation, called the poselet activation vector, consists of poselets that are consistent with the bounding box. The vector has an entry for each poselet type which reflects the degree to which the poselet type is active in that person. This provides a distributed representation of the high dimensional non-linear pose space of humans as shown in Figure 5.4. Notice that the pose and appearance information is encoded at multiple scales. For example, we could have a part which indicates just the head or just the torso or the full pedestrian. We use this representation for both action recognition and 3D pose estimation from still images.

5.5 3D Pose Estimation from Still Images

First we quantitatively evaluate the power of the poselet activation vector representation for estimating pose. Our task is to estimate the 3D pose of the head and torso given the bounding box of the person in the image. Current approaches for pose estimation based on variants of pictorial structures are quite ill suited for this task as they do not distinguish between a front facing and back facing person. Some techniques can estimate the 3D pose of the head by first detecting fiducial points and fitting it to a 3D model of the head, or by regressing the pose from the responses of face detectors trained to detect faces at different orientations [72]. These methods are not applicable when the face itself is occluded or when the image is at too low a resolution for a face detector, a common occurrence in our dataset.

The pose/aspect of the person in encoded at multiple scales and often one can roughly guess the 3D pose of the person from various parts of the person as seen in Figure 5.2 and our representation based on poselets are an effective way to use this information. Our results show that we are able to estimate the pose quite well for both profile and back facing persons.



(a) AMT interface (b) Human Error (c) Head Yaw (d) Torso Yaw (e) Head-Torso Yaw

Figure 5.5: (a) Interface for annotating the 3D pose on Amazon Mechanical Turk. (b) Human error rate across view for estimating the pose of the head and torso. (c, d, e) Distribution of the yaw of head, torso and torso relative to the head, on our 3D pose dataset.

A Dataset of 3D Pose Annotations. Since we wanted to study the problem of pose estimation in a challenging setting, we collected images of people from the *validation* subset of PASCAL VOC 2010 dataset not marked as difficult. We asked the users on Amazon Mechanical Turk [2], to estimate the rotations around X,Y and Z of the head and torso by adjusting the pose of two gauge figures as seen in Figure 6.7(a). We manually verified the results and threw away the images where there was high disagreement between the annotators. These typically turned out to be images of low resolution or severe occlusion.

Our dataset has very few examples where the rotation along X and Z axes is high, as is typical of consumer photographs, hence we removed images which have rotations along X and Z $> 30°$ and focus on estimating the rotation around Y (Yaw) only. In the end we have 1620 people annotations that along with their reflections result in 3240 examples. The distribution of the yaw across the dataset is shown in Figure 6.7(c, d, e).

Figure 6.7(b) shows the human error in estimating the yaw across views of the head and torso. This is measured as the average of standard deviation of the annotations on a single image in the view range. The error is small for people in canonical views, i.e. when the person is facing front, back, left or right, whereas it is high when the person in facing somewhere in between. Overall the annotators are fairly consistent with one another with a median error of $6.66°$ for the head and $7.07°$ for the torso across views.

Experiments. Similar to [13], we train 1200 poselets on the PASCAL train 2010 + H3D trainval dataset. Instead of all poselets having the same aspect ratio, we used four aspect ratios: 96×64, 64×64, 64×96 and 128×64 and trained 300 poselets of each. In addition we fit a model of bounding box prediction for each poselet. We construct the poselet activation vector by considering all poselet detections whose predicted bounding box overlaps the bounding box of the person, defined by the intersection over union > 0.20 and adding up the detection scores for each poselet type. We use this 1200 dimensional vector to estimate the pose of the person.

We estimate the pose of the head and torso separately. We discretize the yaw $\in [-180°, 180°]$

into 8 discrete bins and train one-vs-all linear classifiers for predicting the discrete label. The angle is obtained by parabolic interpolation using the highest predicted bin and its two adjacent neighbors. We optimize our parameters on one split of the data and report results using 10 fold cross validation. We split the training and test set equally ensuring both the image and its reflection are both either in the training or the test set.

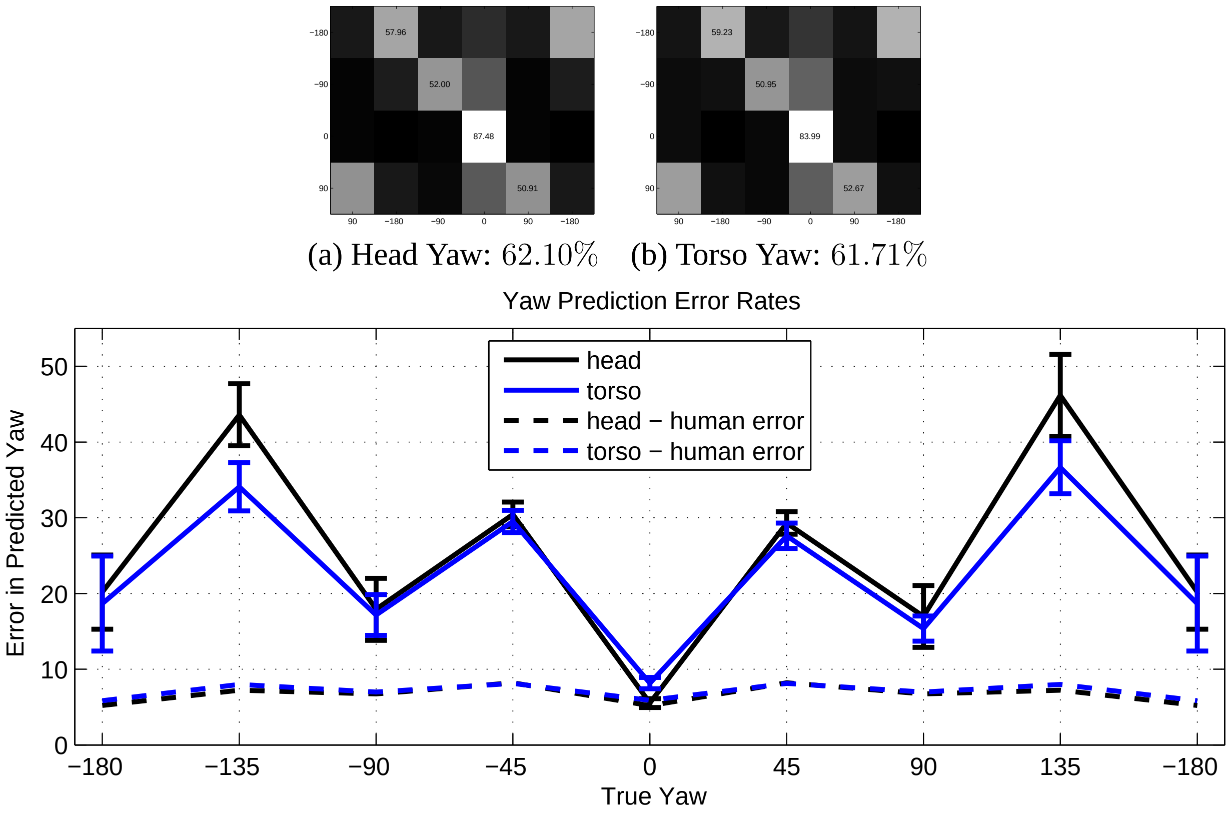
Figure 5.6(a, b) show the confusion matrix for the task of predicting the discrete view, one of front, left, right and back, for the head and torso. The average diagonal accuracy is 62.1% for the head and 61.71% for the torso. The median errors in predicting the real valued view are shown in Figure 5.6(c). We report results by averaging the error for predicting view across 8 discrete views. Since the dataset is biased towards frontal views, this error metric gives us a better idea of the accuracy of the method. Across all views the error is about $26.3°$ and $23.4°$ for the head and torso respectively, while across the front views, i.e. yaw $\in [-90°, 90°]$, the error is lower: $20.0°$ and $19.6°$ respectively. In particular, the error when the person is facing front, i.e. yaw $\in [-22.5°, 22.5°]$ matches the human error rate. Our method is able to recognize the pose of back facing people, i.e. yaw $\in [157.5°, -157.5°]$, a $45°$ range around the back facing view, with an error of about $20°$ error for the head and torso. Approaches based on face detection would fail but our representation benefits from information at multiple scales like the overall shape of the person, as shown in Figure 5.7. The error is smaller when the person is facing exactly left, right, front and back while it is higher when the person is facing somewhere in between, qualitatively similar to humans.

At roughly $25°$ error across views, our method is significantly better than the baseline error of $90°$ for the method that always predicts the view as frontal (It gets $0°$ error for frontal view, but $180°$ error for back view). Figure 5.8 shows some example images in our dataset with the estimated pose. We believe this is a good result on this difficult dataset demonstrating the effectiveness of our representation for coarse 3D pose estimation.

5.6 Static Action Classification

In this section we present our method for action classification and report results on the newly introduced PASCAL VOC 2010 action classification benchmark. The input is a set of bounding boxes on images and the task is to score each of these with respect to nine action categories namely : *phoning, playinginstrument, reading, ridingbike, ridinghorse, running, takingphoto, usingcomputer* and *walking*. Figure 5.3 shows examples from various action categories.

Action specific poselets. There are 608 training examples for all the action categories. To train poselet models we first annotate each person with 2D joint locations on Amazon Mechanical Turk. Five independent annotators were asked to annotate every image and the results were averaged with some outlier rejection. Similar to the approach of [13] we randomly sample windows of various aspect ratios and use the joint locations to find training examples each poselet.



(a) Head Yaw: 62.10% (b) Torso Yaw: 61.71%

(c) Error in predicting yaw across views

Figure 5.6: (a, b) Average confusion matrix over 10-fold cross validation, for predicting four views *left*, *right*, *front* and *back*. The mean diagonal accuracy is 62.10% and 61.71% for predicting the head and the torso respectively. (c) Error in predicting the yaw averaged over 8 discrete views using 10-fold cross validation. Across all views the error is about $26.3°$ and $23.4°$ for the head and torso respectively, while across the front views, i.e. yaw $\in [-90°, 90°]$, the error is lower $20.0°, 19.6°$. In particular the error when the person is facing front, i.e. yaw $\in [-22.5°, 22.5°]$ matches the human error rate.

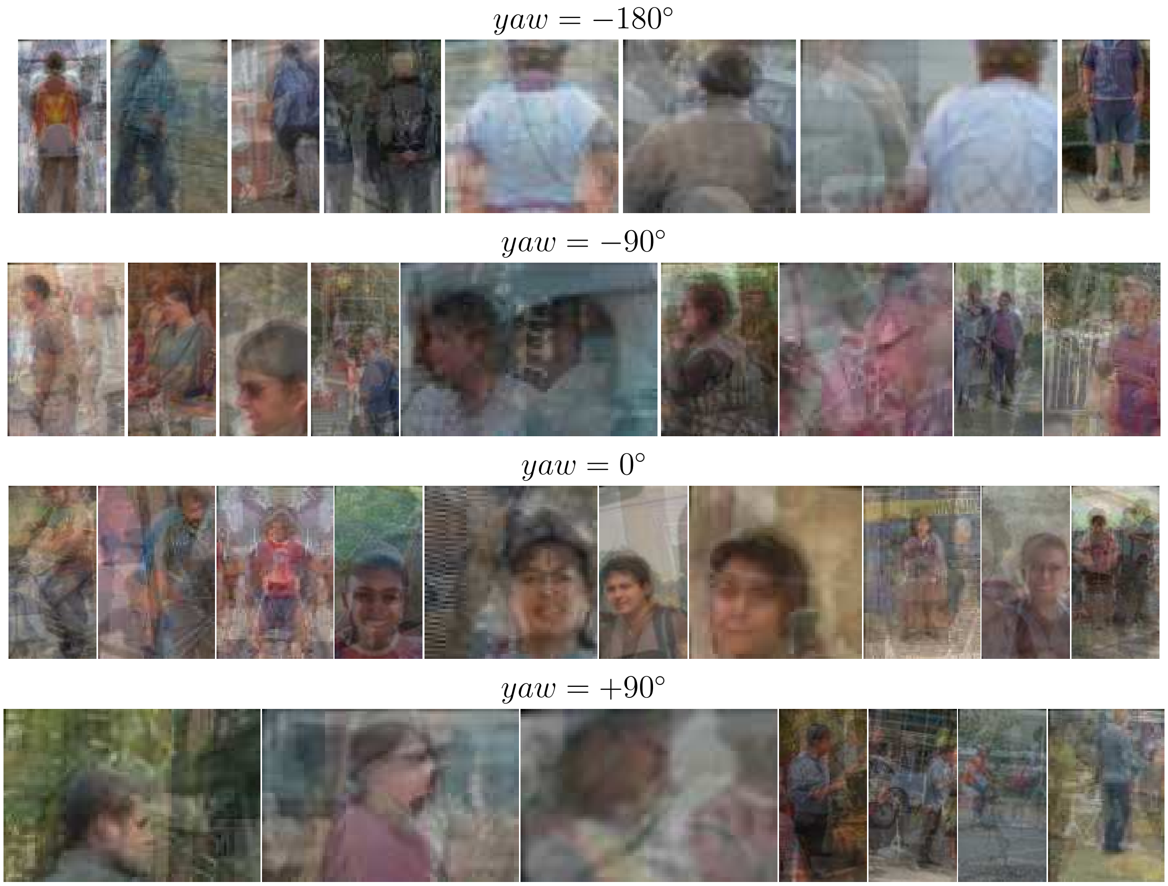


Figure 5.7: **Poselets with the highest weights for discrete view classification of the head.** Note that information from multiple scales is used to infer the view. When the person is back-facing, i.e. yaw $= -180°$, poselets corresponding to pedestrians and upper-body are selected where as for the frontal view face poselets are selected.

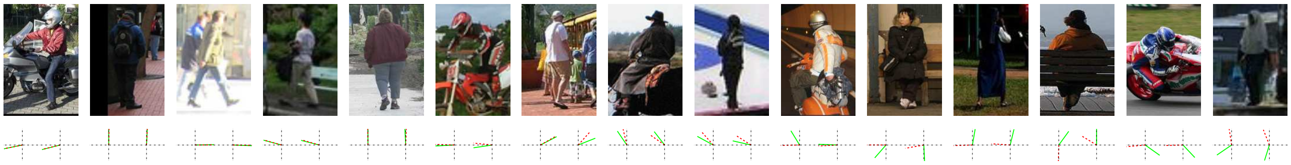


Figure 5.8: Left to right are examples images in our 3D pose dataset of increasing prediction error. Under each image the plot shows the true yaw for the head (left) and torso (right) in green and the predicted yaw in red. We are able to estimate the pose even when the face, limbs and other body parts are hard to detect.

Figure 5.9 shows that pose alone cannot distinguish between actions and the appearance information is complimentary. For example we would like to learn that people riding bikes and horses often wear helmets, runners often wear shorts, or that people taking pictures have their faces occluded by a camera. *To model this, we learn action specific appearance by restricting the training examples of a poselet to belong to the same action category*.

Many poselets like a "face" poselet may not discriminate between actions. *The idea illustrated in Figure 5.10, is windows that capture salient pose specific to certain actions are likely to be useful for action discrimination*. We measure "discriminativeness" by the number of within class examples of the "seed" windows in the top $k = 50$ nearest examples for the poselet. The idea is that if a pose is discriminative then there will be many examples of that poselet from within the same class. Combined with the earlier step this gives us a way to select poselets which detect salient pose and appearance for actions as shown in Algorithm 2. Appearance models are based on HOG [21] and linear SVM. We learn 1200 action specific poselets. Figure 5.11 shows representative poselets from four action categories.

Algorithm 2 Action specific poselet selection.

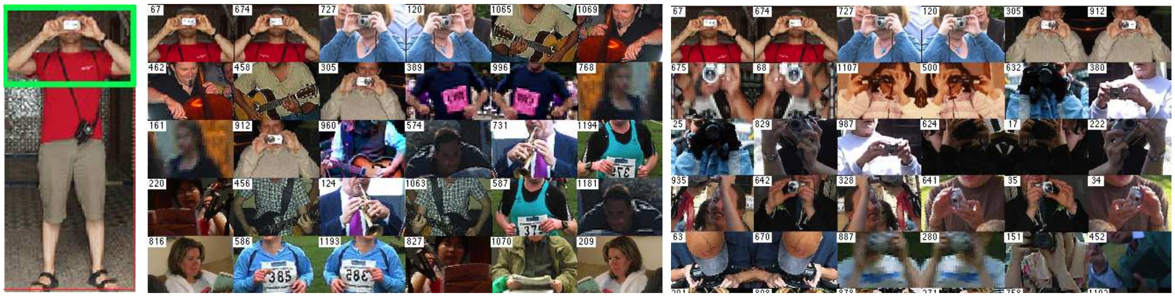
```
Require: 2D keypoint/action labels on training images.
1: for $i = 1$ to $n$ do
2:     Pick a random seed window and find the nearest examples in configuration space based on
       the algorithm of [13].
3:     Compute the number of within class examples in the $k = 50$ nearest examples.
4: end for
5: Select the top $m$ seed windows which have the highest number within class examples.
6: For each selected window, restrict the training examples to within the class and learn an
   appearance model based on HOG and linear SVM.
```

Remarks:

- *Steps* $1 - 5$ *learn action specific pose, while step* 6 *learns action specific appearance.*
- *We ensure diversity by running steps* $1 - 6$ *in parallel. We set* $m = 60, n = 600$ *across* 20 *nodes to learn* 1200 *poselets.*

Poselet Activation Vector. The action poselets are run in a scanning window manner and we collect poselet detections whose predicted bounds overlap the given person bounds, defined by the intersection over union of the area > 0.15. The i'th entry of the poselet activation vector is the sum of the scores of all such detections of poselet type i.

Spatial Model of Object Interaction. Interaction with other objects often provides useful cues for disambiguating actions [121]. For example, images of people riding horses have the person and the horse in certain spatial configurations. We model the interaction with four object categories : *horse, motorbike, bicycle* and *tvmonitor*. We learn a mixture model of the relative spatial



seed all examples within class examples

Figure 5.9: The middle image shows the nearest examples matching the seed using the pose alone, while the image on right shows the top examples within the *takingphoto* category. This allows us to learn appearance and pose specific to that action.

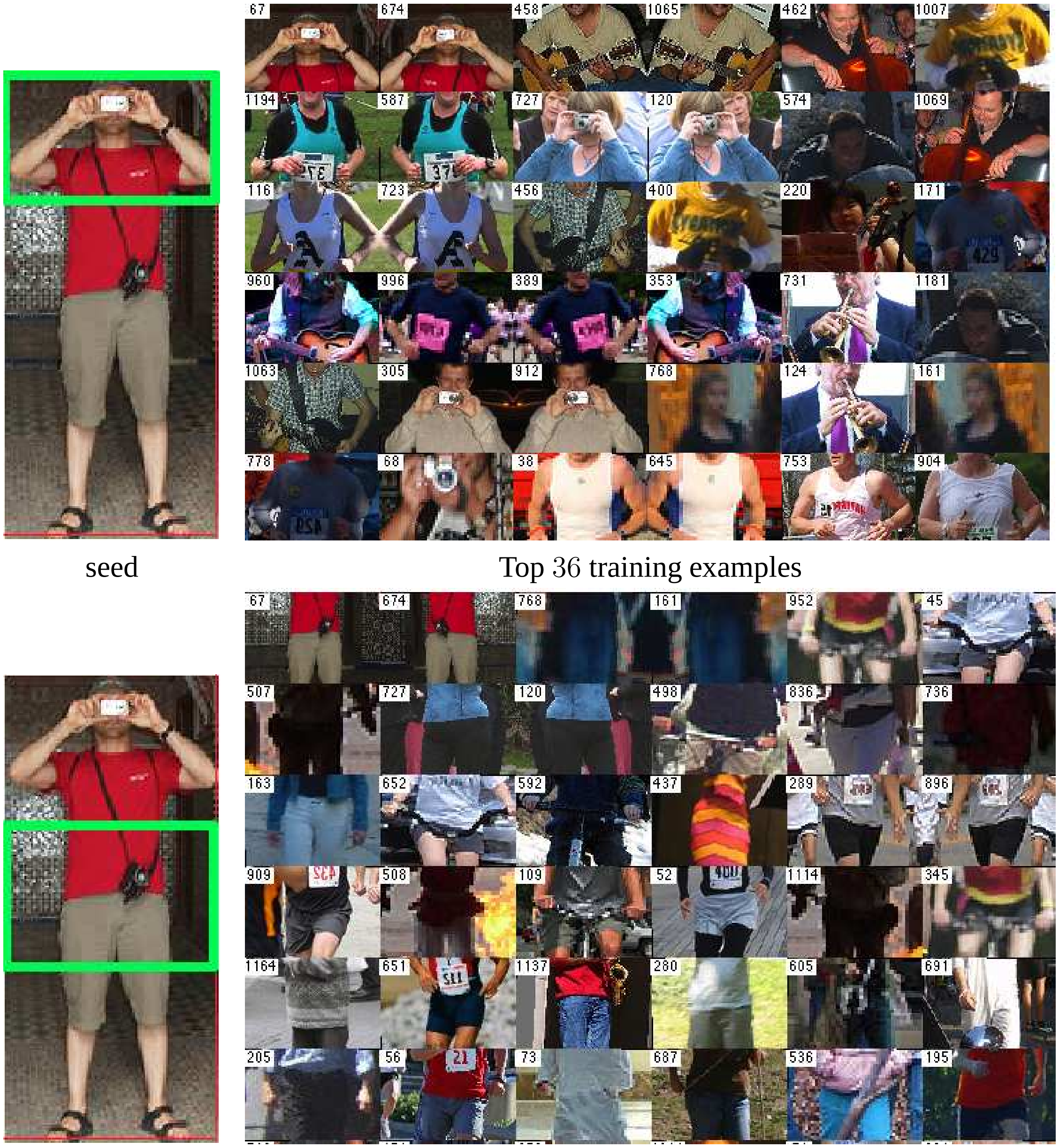
location between the person bounding box and the object bounding box in the image as shown in Figure 5.12. For detecting these objects we use the detector based on poselets trained on these object categories presented in the PASCAL VOC 2010 object detection challenge. For each object type we fit a two component mixture model of the predicted bounding box to model the various aspects of the person and objects.

Given the object detections we find all the objects whose predicted person bounds overlap the bounds of the given person > 0.3. Similar to the poselet activation vector we construct an ”object activation vector” by taking the highest score of the detection for each object type among these.

Action context. Often the action of a person can be inferred based on what others are doing in the image. This is particularly true for actions like *playinginstrument* and *running* which are group activities. Our action context for each person is a 9 dimensional vector with an entry for each action type whose value is the highest score of the action prediction among all the other people in the image. Overall the second stage classifier is a separate linear SVM for each action type trained on 10 features: self score for that action and 9 for action context.

Experiments. Table 5.1 shows the performance of various features on the test and validation set. All the parameters described were set using a 10-fold cross validation on the trainval subset of the images.

The *poselet activation vector* alone achieves a performance of 59.8 on the validation subset of images and does quite well in distinguishing classes like *ridinghorse*, *running*, *walking* and *phoning*. Adding the object model boosts the performance of categories like *ridingbike* and *usingcomputer* significantly, improving the average AP to 65.3. These classes either have the widely varying object types and poselets are unable to capture the appearance variation. Modeling the spatial interaction explicitly also helps for classifying *usingcomputer* class as the interaction is often outside the bounding box of the person. Finally the context based re scoring improves the



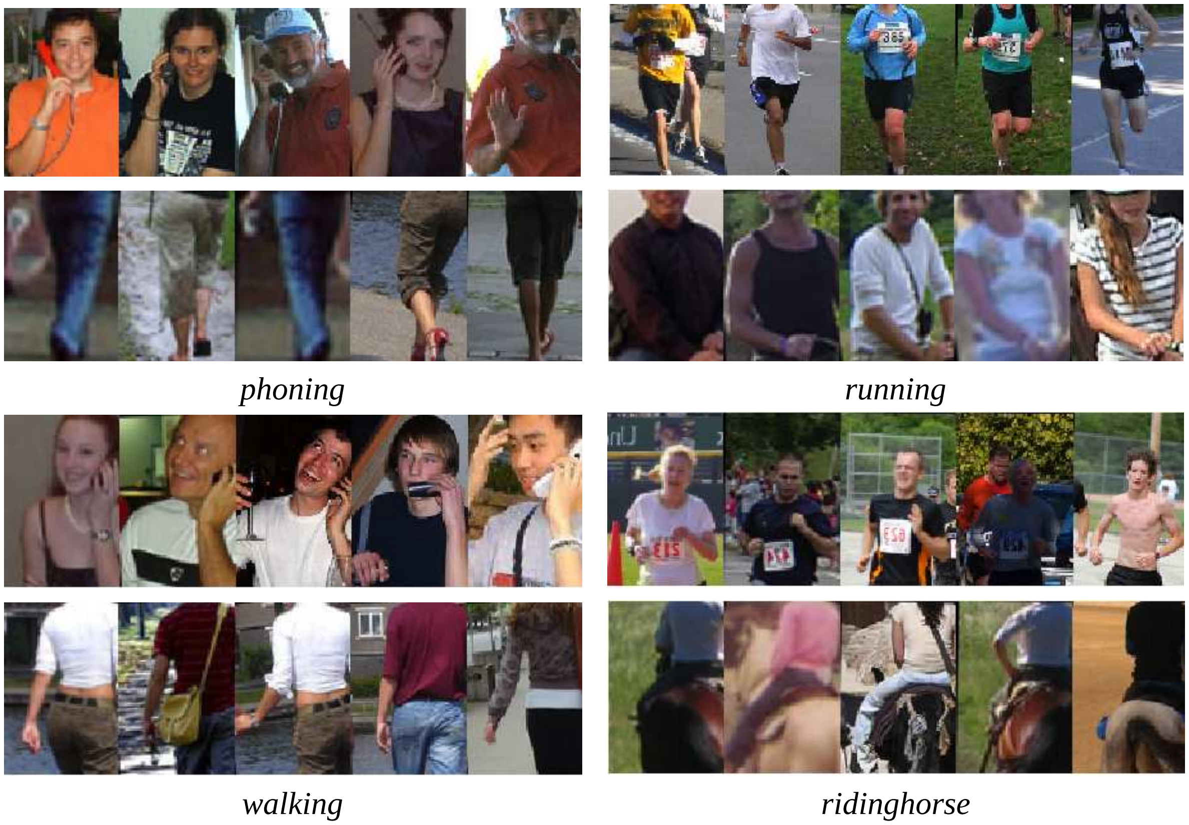
seed

Top 36 training examples

seed

Top 36 training examples

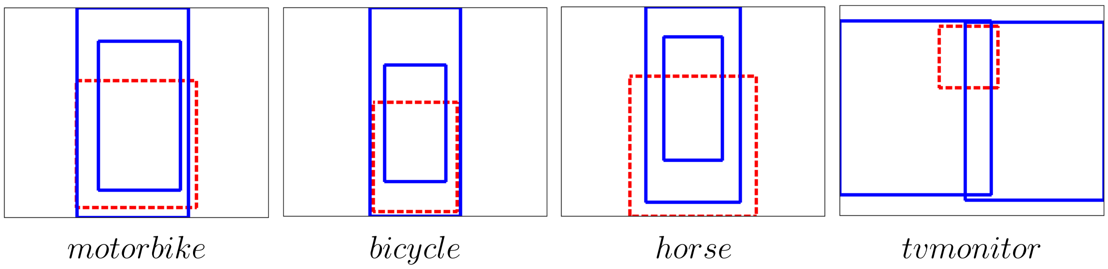
Figure 5.10: The top row shows a seed window that captures a salient pose for the *takingphoto* category. The 36 nearest examples in configuration space for the top seed window has 7 examples from the *takingphoto* category while the bottom seed has only 2.



phoning *running*

walking *ridinghorse*

Figure 5.11: Example poselets shown by their top 5 training examples for various action categories. These capture both the pose and appearance variation across the action categories.



$motorbike$ $bicycle$ $horse$ $tvmonitor$

Figure 5.12: Spatial model of the object person interaction. Each image shows the modes of the bounding boxes of the person (blue) relative to the bounding box of the object (red). For *horse*, *motorbike* and *bicycle* category the two modes capture front and side views of the object while for the *tvmonitor* it captures the fact that TV monitors are often at the left or right corner of the person bounding box.

	Validation			Test
category	PAV	w/ OAV	w/ C	w/ C
phoning	63.3	62.0	62.0	49.6
playinginstrument	44.2	44.4	45.6	43.2
reading	37.4	44.4	44.3	27.7
ridingbike	62.0	84.7	85.5	83.7
ridinghorse	91.1	97.7	97.5	89.4
running	82.4	84.1	86.0	85.6
takingphoto	21.1	22.9	24.6	31.0
usingcomputer	54.2	64.9	64.3	59.1
walking	82.0	83.6	80.8	67.9
average	59.8	65.3	65.6	59.7

Table 5.1: Average precision on the action validation and test set using various features. PAV is the performance using just the *poselet activation vector*. Column w/OAV shows the performance by including the *object activation vector* as features and column w/C shows the performance by including action context. The object features help in the *ridingbike, ridinghorse* and *usingcomputer* categories, while the context improves the performance on *playinginstrument* and *running* categories. Our methods achieves an average AP of 59.7 on the test set which is comparable to the winning techniques in PASCAL VOC 2010.

performance of *playinginstrument* and *running* class as these are often group activities.

Figure 5.13 shows the confusion matrix of our classifier. Some high confusion pairs are $\{$*reading, takingphoto*$\}$ $\rightarrow$ *playinginstrument* and *running* $\rightarrow$ *walking*. Figure 5.14 shows misclassified examples for several pairs of categories. Overall our method achieves an AP of 65.6 on the validation and 59.7 on the test set which is comparable to the winning techniques in PASCAL VOC 2010 challenge, for example, 60.1 for "INRIA_SPM_HT" and 60.3 for "CVC_BASE". We refer the readers to the challenge website[1] for details and results of other entries.

5.7 Conclusion

We demonstrate the effectiveness of the poselet activation vector on the challenging tasks of 3D pose estimation of people and static action recognition. Contrary to the traditional way of representing pose which is based on the locations of joints in images, we use the poselet activation vector to capture the inherent ambiguity of the pose and aspect in a multi-scale manner. This is well suited for estimating the 3D pose of persons as well as actions from static images. In the future we would like to investigate this representation for localizing body parts by combining top

[1] `http://pascallin.ecs.soton.ac.uk/challenges/VOC`

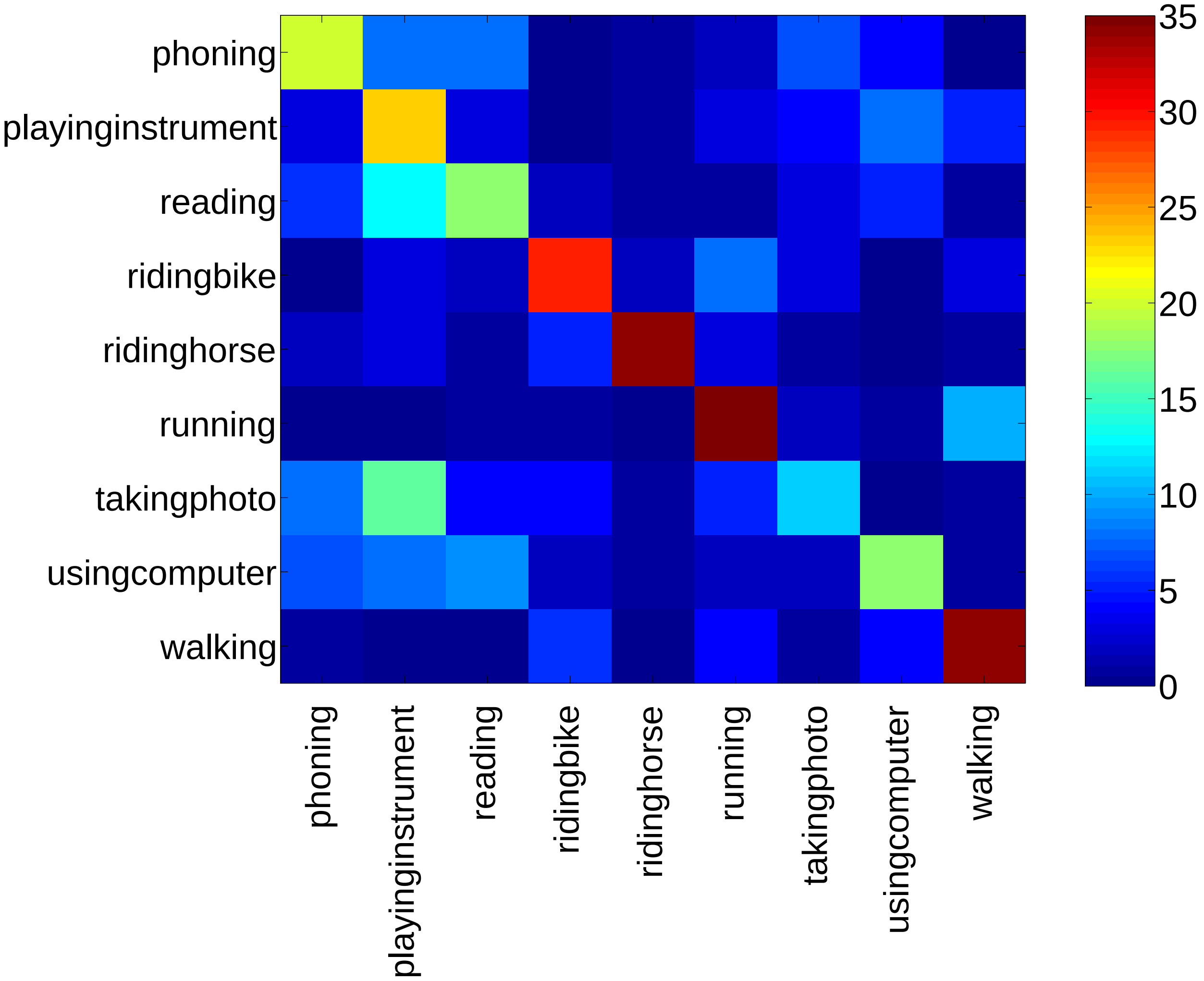


Figure 5.13: Confusion matrix for our action classifier. Each row shows the distribution of the true labels of the top 50 ranked examples for each action category on the validation subset of the images. Some high confusion pairs are $\{$*reading*, *takingphoto*$\}$ → *playinginstrument* and *running* → *walking*.

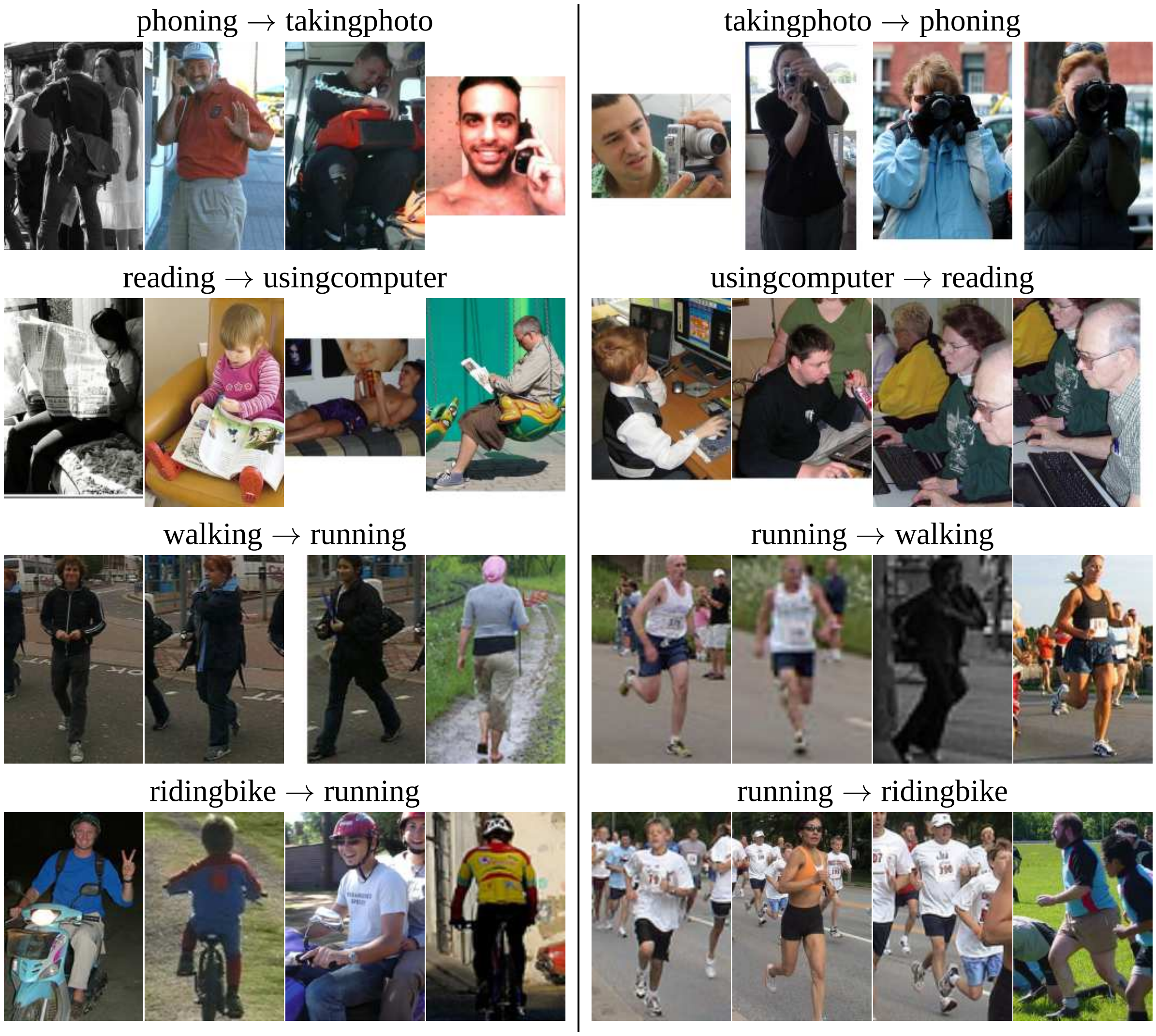


Figure 5.14: **Pairwise confusions between several classes on the PASCAL 2010 validation set**. Each A → B shows the top 4 images of class A ranked by classifier of class B. Confusion is often caused when the person has similar pose or failures of the object detector.

down pose estimates with bottom-up priors and exploit pose-to-pose constraints between people and objects to estimate pose better.

Most of the other high performing methods on the PASCAL VOC 2010 action classification task use low-level features based on color and texture together with a SVM classifier, without any explicit treatment of pose. We believe that such methods benefit from the fact that one is provided with accurate bounding boxes of the person in the image. This is quite unrealistic in an automatic system where one has to estimate the bounds using a noisy object detector. We on the other hand use the bounding box information quite loosely by considering all poselet detections that overlap sufficiently with the bounding box. In addition, the poselet activation vector provides a compact representation of the pose and action, unlike the high dimensional features typical of “bag-of-words” style approaches.

The annotations and code for estimating the yaw of the head and torso in images, as well as the keypoint annotations and code for static image action classification can be downloaded from the author’s website.

Chapter 6

Crowdsourcing for Computer Vision

We describe our experience of collecting roughly $250,000$ image annotations on Amazon Mechanical Turk (AMT) [2]. The annotations we collected range from location of keypoints and figure ground masks of various object categories, 3D pose estimates of head and torsos of people in images and attributes like gender, type of hair and clothing, etc. We describe the setup and strategies we adopted to automatically approve and reject the annotations. Such automation is necessary for large scale annotations since the task of verification can itself be tedious and time consuming. These annotations were used to train algorithms for detection/segmentation [13], semantic boundary extraction [48], pose estimation/action recognition [67] and attribute recognition of people in images [11], some of which we described in the previous chapter.

Collecting annotations in a cost effective manner has become possible due to emergence of efficient marketplaces like AMT. Services like AMT serve as marketplace, where "workers" complete Human Intelligence Tasks (HITs), as illustrated in Figure 6.1. The large pool of available workers enables completion of large scale visual annotation tasks in a time and cost effective manner. There are three ingredients for constructing a HIT (Human Intelligence Task) which "workers" on AMT can complete :

1. **User Interface.** This is the front-end which enables the user to do the task inside their web browsers. Some of our tasks required users to draw the boundaries or mark the locations of various keypoints of objects. All our GUIs (Graphical User Interfaces) were written in Java/JavaScript + HTML.

2. **Instructions.** Contains the task description, with examples of completed task as well as GUI usage instructions.

3. **Verification.** A method to approve/reject the HITs. This becomes important for large scale annotations as this step also has to be done automatically. One can have a task done by multiple workers followed by outlier rejection or a secondary HIT to verify the results to automatically select the right answers. We adopt the first approach for all our tasks.

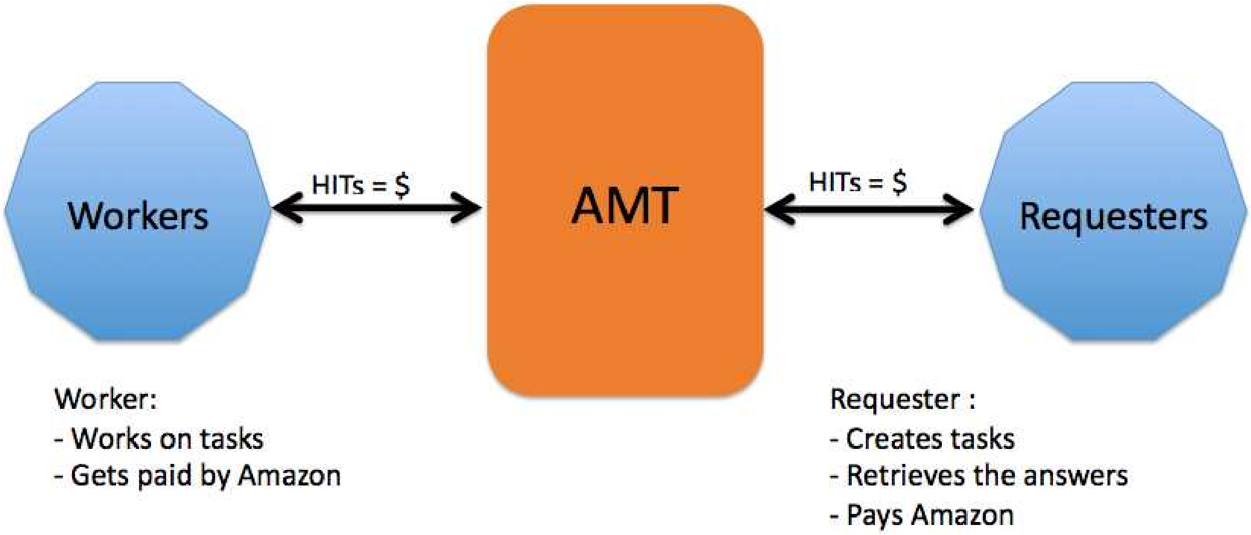


Figure 6.1: Amazon mechanical turk serves as a marketplace for workers to complete "Human Intelligence Tasks" (HITs).

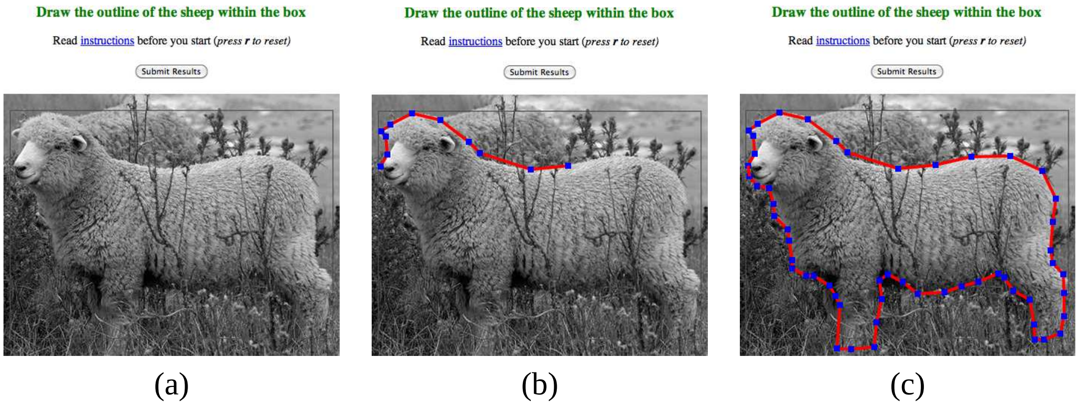
An interesting aspect of collecting annotations on AMT is that we can measure the inherent hardness of these tasks. Many of these tasks don't require specific training and human performance of even a casual annotator is quite good. The agreement between various workers on a given problem can provide us a sense of the hardness of the task and an upper bound on the performance one might expect from an automatic system. In the 3D pose estimation problem, we see that the humans are not perfect, with an average error of 6° across views.

We describe the three ingredients, i.e. the interface, instructions and verification method for the each of the tasks we set up on AMT in the next few sections.

6.1 Figure-ground Masks of Objects

We collect figure ground masks for various object categories. We focus on the categories and images from the PASCAL VOC 2010 dataset [28]. The dataset has $23,374$ objects in the training/validation set from 20 categories. The statistics of the dataset are show in the Table 6.1.

Interface & Instructions. Our interface was as simple polygon outline tool which allows the user to draw a closed polygon and then move the vertices around to adjust the polygon. The advantage of this interface is that it is quite simple and intuitive to use. On the other hand, it only allows the user to draw one closed polygon which does not work well for objects with holes. An alternate interface was one which allows the user to paint the pixels belonging to the figure. This interface is too time consuming if done at the pixel level and too inaccurate on the boundaries if done at a "superpixel" (or a coarser quantization) level.



Draw the outline of the sheep within the box

Read instructions before you start (*press r to reset*)

Figure 6.2: The user interface for annotating the outer boundary of the objects. (a) The user sees this inside the Amazon Mechanical Turk environment. (b) Partial annotation by the user. (c) The user closes the polygon and edits the boundary if needed, and clicks the submit button.

Figure 6.2 shows our interface for marking the outer boundaries of the objects. We provide instructions and sample segmentations, to describe the task to the user. Below that we display the image to be segmented. To avoid confusion when there are multiple overlapping objects in the image, we draw a bounding box to indicate which object we are interested in.

The interface is written in Javascript + HTML5. It uses the "canvas" tag [105] which is currently supported in the latest Internet Explorer, Firefox and Safari browsers. We did not have any users complaining that the interface was not working properly for them. This switch was partly motivated by the difficulties we had in porting our keypoint labeling tool (next section) written in JAVA to various browsers. At the time of writing the "canvas" tag was only partly supported on IE, in particular they had no support for displaying text. We would like to port the keypoint labeling tool to Javascript + HTML5 in the future once text is supported by them [1].

Verification. We collect 5 independent annotations per object. For approving the HITs automatically, we compute the pairwise overlap between the masks of an object, and find the one which overlaps the maximum with everyone else. We consider all masks which overlap with this mask greater than a threshold as correct. The threshold is chosen manually based on how flexible the object category is. For example for rigid objects like "bottles" and "tv-monitors" the we choose a threshold of 0.75 while for less rigid objects like cats and dogs we choose a lower threshold of 0.65 or even lower. In general the quality of segmentations submitted by the users are pretty good and only about 10% of the submitted hits were rejected. Figure 6.3 shows the distribution of submit times for the "aeroplane" and "cat" categories. Figure 6.4 shows some of the submitted results by the workers. Figure 6.5 shows some outliers which are rejected automatically by our algorithm.

[1] canvas text is now supported on most browsers, and the keypoint tool has been ported to HTML5

Category	Number of Objects	Reward (cents)	Submit Time (seconds)
Aeroplane	738	2	77/59
Bicycle	614	2	87/69
Bird	971	1	72/57
Boat	687	1	47/36
Bottle	1014	1	47/36
Bus	498	1	55/41
Car	1774	1	55/42
Cat	1132	1	70/57
Chair	1890	1	60/44
Cow	464	2	70/58
Diningtable	468	1	50/36
Dog	1416	1	70/57
Horse	621	2	95/77
Motorbike	611	2	80/65
Person	7296	1	55/43
Pottedplant	821	1	65/50
Sheep	701	2	67/50
Sofa	451	1	65/51
Train	524	1	59/46
Tvmonitor	683	1	32/25
Total	23374		

Table 6.1: Statistics of PASCAL VOC 2010 trainval set. For each image we collected 5 independent annotations. We paid them either 1 or 2 cents (US currency), based on the how complex we thought the boundaries of the class were, as shown in the “Reward” column. This is more or less also reflected in the mean/median submit time of the HITs shown in the last column.

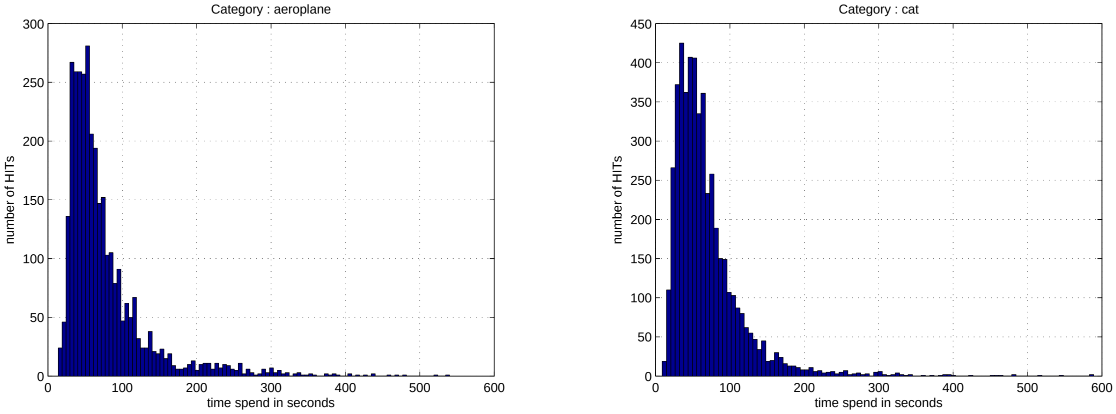


Figure 6.3: Histogram of submit times for the “aeroplane” (left) and “cat” (right) categories.

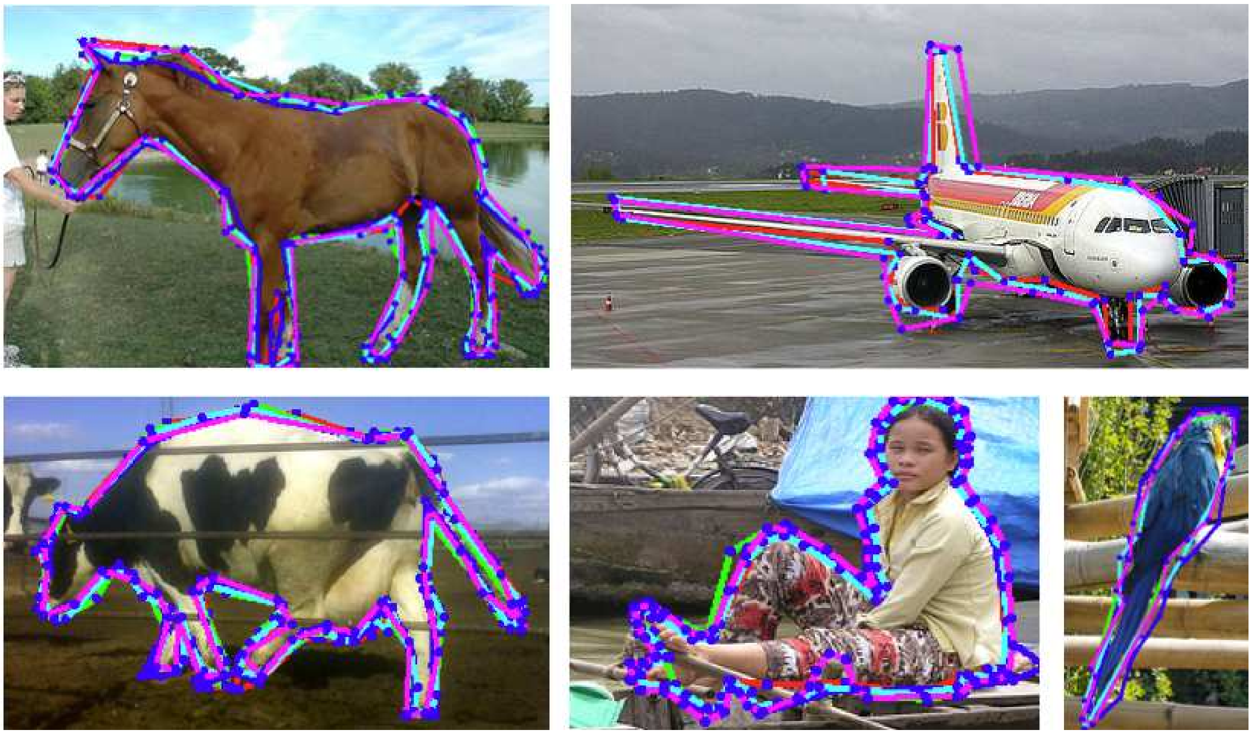


Figure 6.4: Example results submitted by workers.

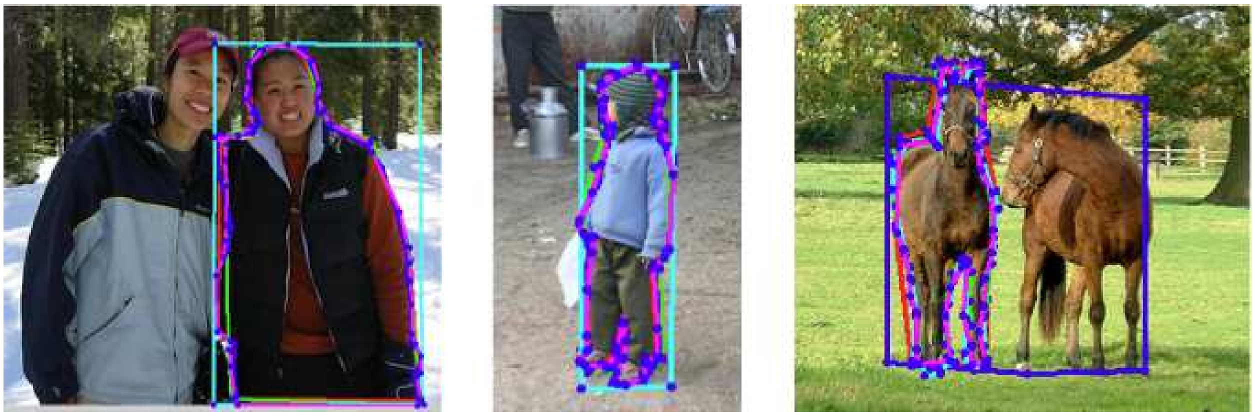


Figure 6.5: Outliers in the submitted boundaries by the workers on several images. These are automatically rejected as they do not have high overlap with the best answer.

6.2 Keypoint Annotation of Objects

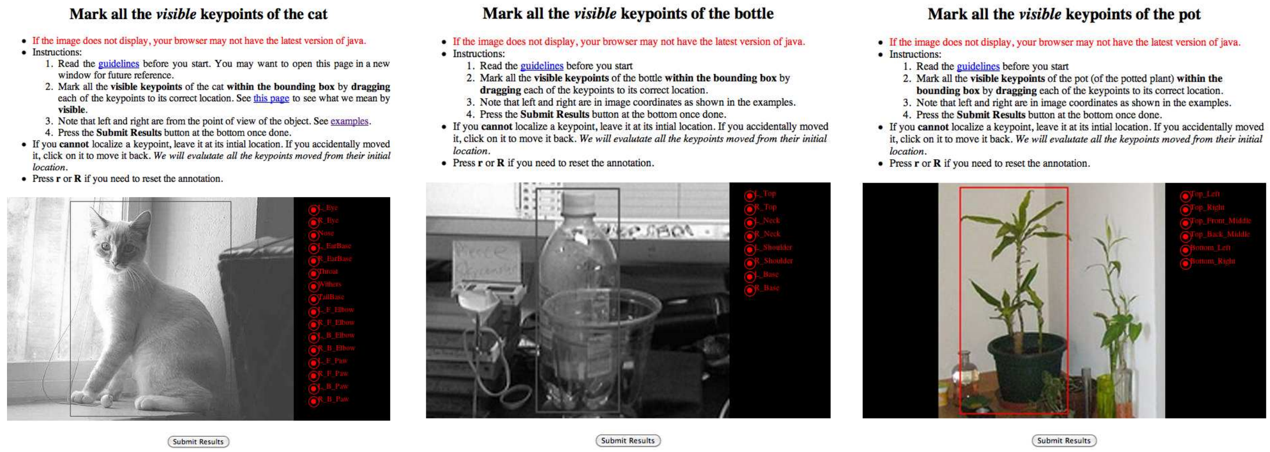
Our goal here was to mark the locations of various keypoints of objects in images. The first challenge is deciding which keypoints to use. This is fairly straightforward for animal categories, where one can base them on anatomical parts, but becomes non trivial for categories such as a chair, a boat and an airplane, whose examples have large structural variations. For example, there are chairs with four legs or one stem and a wide base. Some chairs have armrests, and others don't. Military airplanes look very different from commercial ones, and sail boats have little in common with cruise ships. Our approach was to split the categories into a few sub-categories and each of which has its own set of keypoints. This allows us to train separate "poselets", which we described in the previous chapter with more supervision. For example one could train poselets for pointed front of a military airplane, the round tip of a commercial airliner and the propeller blades of a propeller plane.

The second challenge is that some categories do not have a principal orientation, which makes it difficult to assign keypoints in the reference frame of the object. For example, it is clear what the front left leg is in the case of a horse, but what is the front left leg of a table? Other categories have round parts and thus have no extrema points, such as the base of a bottle or a potted plant. Our solution in these cases is to introduce view-dependent keypoints. For example, we have a keypoint for the bottom left corner of a bottle, and we define the front left leg of a table based on the current camera view. The number of keypoints and the sub-categories are shown in Table 6.2.

Interface & Instructions. Figure 6.6 shows the interface we have for annotating the keypoints. Each user is shown an image within a bounding box and a list of keypoints. The user drags and drops these to their locations in the image. The user is instructed not to mark the points which are not visible due to occlusion, truncation etc. If a user accidentally moves a point then he/she can click on it to move it back to its initial position. Once the user is done he/she can press submit.

Class	# Keypoints	# Subcategories
Aeroplane	16	3
Bicycle	11	1
Bird	12	2
Boat	11	2
Bottle	8	1
Bus	8	1
Car	14	1
Cat	16	1
Chair	10	1
Cow	16	1
Dining table	8	1
Dog	16	1
Horse	19	1
Motorbike	10	1
Person	20	1
Potted plant	6	1
Sheep	16	1
Sofa	12	1
Train	7	1
TV monitor	8	1

Table 6.2: Class-specific variations in the keypoint annotations. **#Keypoints** is the number of keypoints and **#Subcategories** is the number of subcategories.



Mark all the *visible* keypoints of the cat

- If the image does not display, your browser may not have the latest version of java.
- Instructions:
  1. Read the guidelines before you start. You may want to open this page in a new window for future reference.
  2. Mark all the **visible keypoints** of the cat **within the bounding box** by **dragging** each of the keypoints to its correct location. See this page to see what we mean by **visible**.
  3. Note that left and right are from the point of view of the object. See examples.
  4. Press the **Submit Results** button at the bottom once done.
- If you **cannot** localize a keypoint, leave it at its intial location. If you accidentally moved it, click on it to move it back. *We will evalutate all the keypoints moved from their initial location.*
- Press **r** or **R** if you need to reset the annotation.

Mark all the *visible* keypoints of the bottle

- If the image does not display, your browser may not have the latest version of java.
- Instructions:
  1. Read the guidelines before you start
  2. Mark all the **visible keypoints** of the bottle **within the bounding box** by **dragging** each of the keypoints to its correct location.
  3. Note that left and right are in image coordinates as shown in the examples.
  4. Press the **Submit Results** button at the bottom once done.
- If you **cannot** localize a keypoint, leave it at its intial location. If you accidentally moved it, click on it to move it back. *We will evalutate all the keypoints moved from their initial location.*
- Press **r** or **R** if you need to reset the annotation.

Mark all the *visible* keypoints of the pot

- If the image does not display, your browser may not have the latest version of java.
- Instructions:
  1. Read the guidelines before you start
  2. Mark all the **visible keypoints** of the pot (of the potted plant) **within the bounding box** by **dragging** each of the keypoints to its correct location.
  3. Note that left and right are in image coordinates as shown in the examples.
  4. Press the **Submit Results** button at the bottom once done.
- If you **cannot** localize a keypoint, leave it at its intial location. If you accidentally moved it, click on it to move it back. *We will evalutate all the keypoints moved from their initial location.*
- Press **r** or **R** if you need to reset the annotation.

$(a)cat$ $(b)bottle$ $(c)pottedplant$

Figure 6.6: The user interface for annotating the keypoints of the objects. An image inside the bounding box is shown along with the list of keypoints on the right. The user can move the points and place them on their locations in the image or leave them at untouched if the point is not visible.

Verification. Each object was annotated by 5 independent users. We assume that a keypoint is visible if at least 2 annotators have marked its location. To determine the location of each keypoint, we find the closest pair of annotations and average all the annotations which are within a certain radius of them. We also get an estimate of the variance of keypoints and optionally can fix points which have large variance.

6.3 3D Pose of Humans

We construct a dataset of people annotated with the 3D pose of the head and torso. One may try to estimate the 3D pose from the 2D keypoints, but this is nontrivial because of occlusions, truncations and variations of head/torso sizes across people. Hence we asked users to estimate the rotations around X, Y and Z directly. Our research goal was to study the task of human pose estimation in a challenging setting, hence we collect images of people from the *validation* subset of PASCAL VOC 2010 dataset, but remove the person annotations which are marked difficult or truncated.

Interface & Instructions. The interface in Figure 6.7(a), shows an image on the left and two gauge figures corresponding to the head and the torso on the right. They are asked to adjust the pose of the gauge figures to match the 3D pose of the shown person in the image. Different keys rotate the figures along predefined axis. Other possible ways to manipulate such objects in 3D

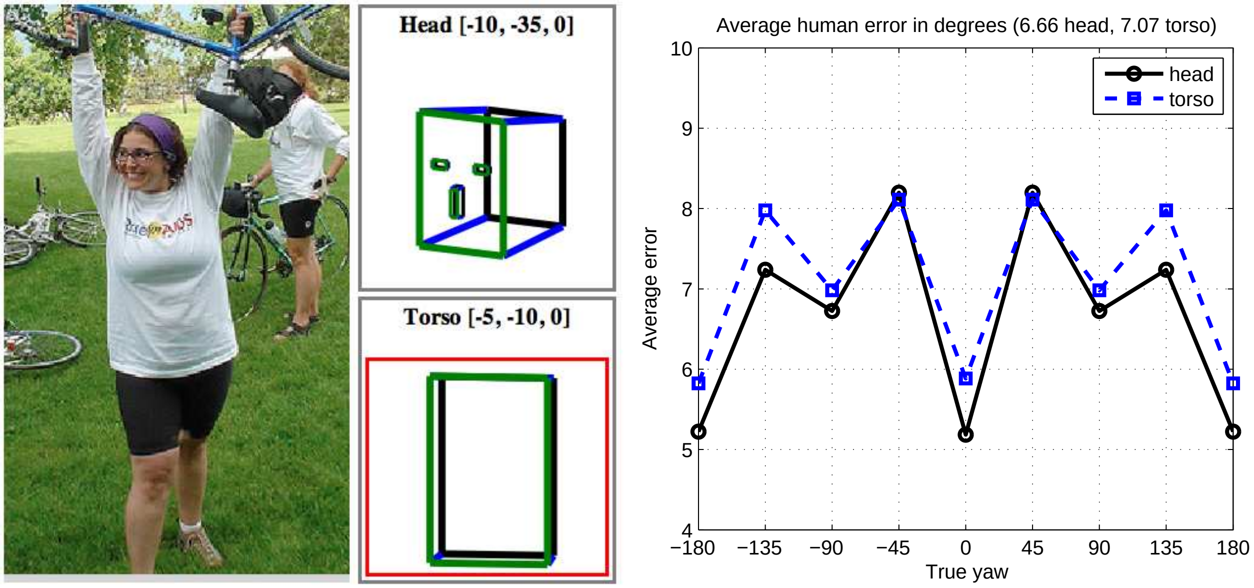


Figure 6.7: (a) The user interface for 3D pose annotation on AMT. (b) Human error rate across views.

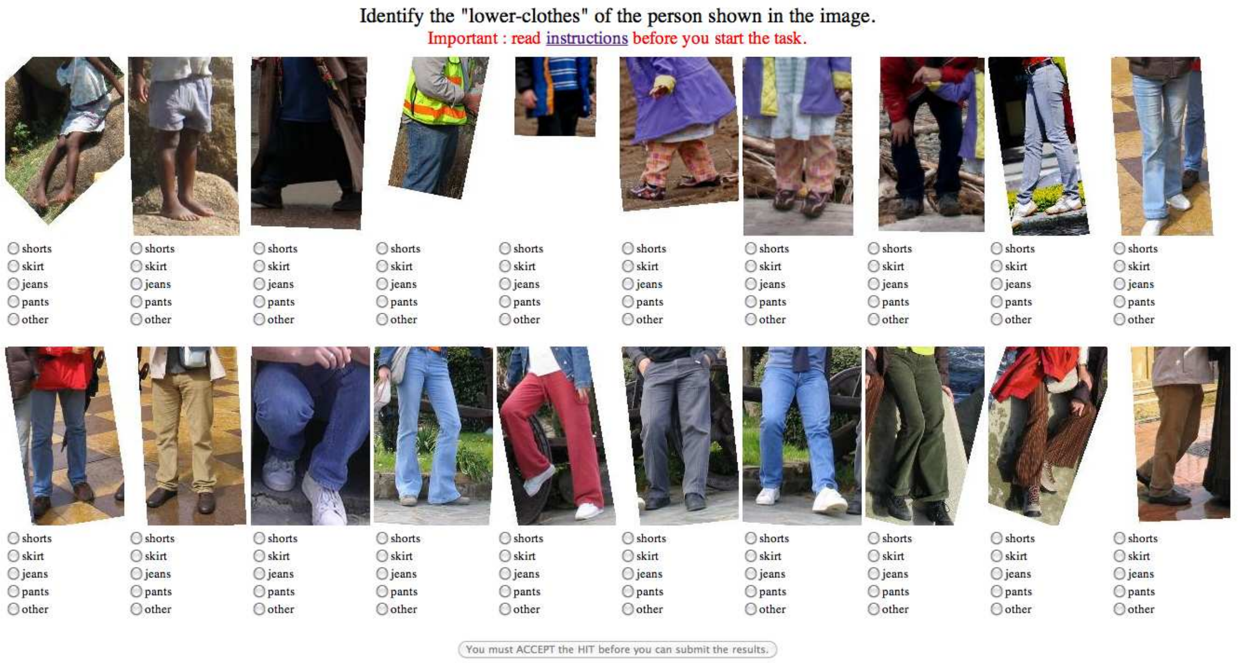
are well known to the design and graphics community, but our interface was easiest to implement using Javascript.

Verification. Each person was annotated by 4 different annotators for outlier rejection and estimate of variance. We manually verified the results and discarded the images, where there was high disagreement between the annotators. These typically turned out to be images of low resolution or severely occluded ones. Our dataset has very few examples where the rotation along X and Z axes is high which is natural in consumer photographs of people. We collected a total of 1620 people annotations.

Figure 6.7(b) shows the human error in estimating the yaw across views of the head and torso. This is measured as the average of standard deviation of the annotations on a single image in the view range. The error is small when the person is facing front, back, left or right whereas it is high when the person in facing somewhere in between. Overall the annotators are fairly consistent with one another with a median error of $6.66°$ for the head and $7.07°$ for the torso across views.

6.4 Attributes of People

About 9000 images of people were taken from the H3D and PASCAL VOC 2010 dataset for which we collect several attributes.



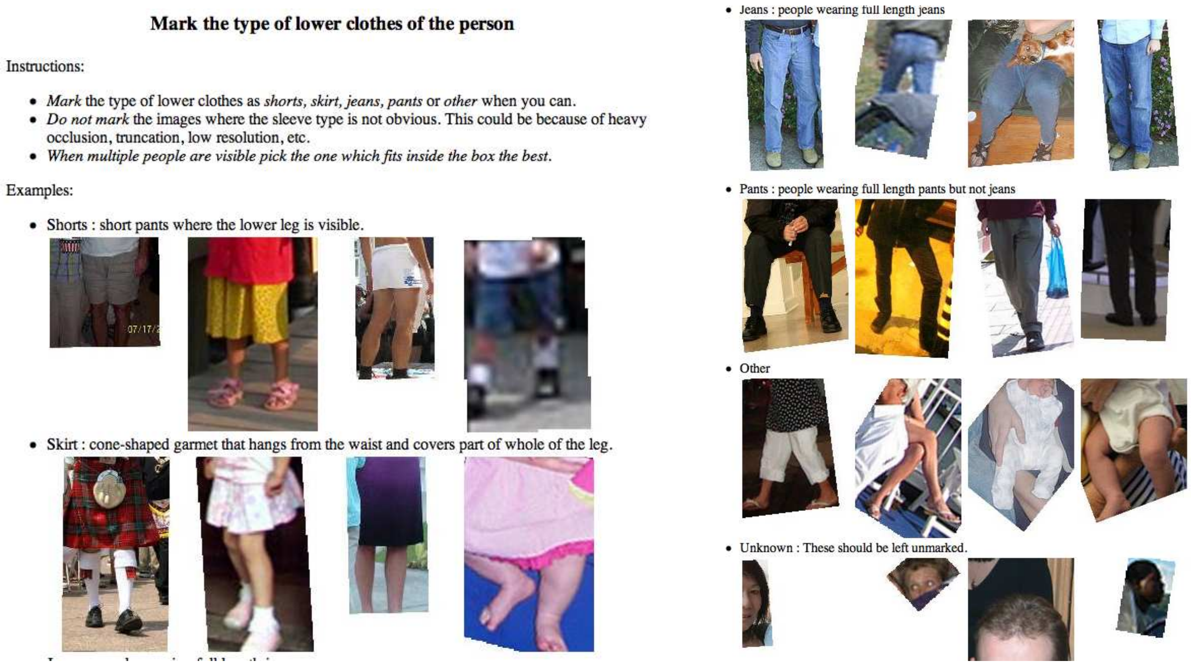
Identify the "lower-clothes" of the person shown in the image.

Important : read instructions before you start the task.

Figure 6.8: User interface for marking the ”lower-clothes” attribute.

Interface & Instructions. Each person is shown a set of images and asked to mark the attribute for each image (Figure 6.8). If the attribute is not clear because of occlusion, truncation, etc, the user was asked not to mark any option. The users were also shown examples for each attribute kind, as shown in Figure 6.9. Table 6.3 shows the list of attributes we annotated. Instead of showing the entire person we only show the region of interest, for example, upper bodies for hair-type and gender and lower bodies for lower-clothes. We are able to do this using the keypoint annotations we obtained earlier on the same dataset. This makes it much more easier for the users to annotate them and there were many more images which were marked with some attribute compared to an earlier run we did using the entire person shown as the same sized images. We typically paid the workers 1 cent for marking 16 attributes per HIT.

Verification. We collected labels for all attributes on all annotations by five independent annotators. A label was considered as ground truth if at least 4 of the 5 annotators agreed on the value of the label. We discarded 501 annotations in which less than two attributes were specified as ground truths which left us with 8035 images. We paid the workers who got at least half the marked annotations right.



Mark the type of lower clothes of the person

Instructions:

- *Mark* the type of lower clothes as *shorts, skirt, jeans, pants* or *other* when you can.
- *Do not mark* the images where the sleeve type is not obvious. This could be because of heavy occlusion, truncation, low resolution, etc.
- *When multiple people are visible pick the one which fits inside the box the best.*

Examples:

- Shorts : short pants where the lower leg is visible.
- Skirt : cone-shaped garmet that hangs from the waist and covers part of whole of the leg.
- Jeans : people wearing full length jeans
- Pants : people wearing full length pants but not jeans
- Other
- Unknown : These should be left unmarked.

Figure 6.9: Instructions for marking the ”lower-clothes” attribute. Examples of each choice of ”lower-clothes”, such as jeans, shorts etc are shown to help the annotator identify these. The user is also shown examples which may be left unmarked.

Attribute	Choices
gender	male, female
race	white, black, asian, indian
age	baby(0-2), child(2-10), adult, old(65+)
hair-type	long, short, no-hair
glasses	regular, sunglasses, no-glasses
shoes	barefoot, sneaker/shoe, sandal
sleeve-type	short-sleeve, long-sleeve, no-sleeve
upper-clothes	t-shirt, shirt, noclothes, bikini, tanktop, bikerwear, other
headgear	cap/hat, full-helmet, half-helmet, other, none
lower-clothes	shorts, skirt, jeans, pants, other
hair-color	black, blonde, white, no-hair, other

Table 6.3: List of attributes we annotated on Amazon Mechanical Turk.

6.5 Conclusion

Services like AMT have enabled us to collect large amounts of annotations in a cost effective manner. Although there are many advantages there are some disadvantages too. The quality of the annotations is a subject of concern sometimes and is a function of the difficulty of the task, amount of time spend by the “workers” which in turn is a function of the “reward”, etc. Sometimes the task is ill posed, for example, how does one label wings of airplanes with multiple wings, leading to inconsistencies in annotations. For classes where the structure varies a lot, creating detailed instructions for various special cases or curating the annotations manually as a post processing step can take the same order of time as manually annotating the images, defeating the advantages of AMT. In such cases one may want to adopt other strategies, for example, have a small set of trained, but highly paid workers, or subject workers to a carefully designed qualification test.

Bibliography

[1] Shivani Agarwal and Dan Roth. “Learning a sparse representation for object detection”. In: *European Conference on Computer Vision (ECCV)*. 2002.

[2] *Amazon Mechanical Turk*. `http://www.mturk.com`.

[3] D. H. Ballard. “Generalizing the Hough transform to detect arbitrary shapes”. In: *Pattern Recognition* 13 (1981).

[4] Serge Belongie et al. “Spectral Partitioning with Indefinite Kernels Using the Nystrom Extension”. In: *European Conference on Computer Vision (ECCV)*. 2002.

[5] Alexander C. Berg and Jitendra Malik. “Geometric Blur for Template Matching”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2001.

[6] T. L. Berg et al. “Names and faces in the news”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2004.

[7] Moshe Blank et al. “Actions as Space-Time Shapes”. In: *International Conference on Computer Vision (ICCV)*. 2005.

[8] A. Bosch, A. Zisserman, and X. Munoz. “Representing shape with a spatial pyramid kernel”. In: *International Conference on Image and Video Retrieval (ICIVR)*. 2007.

[9] S. Boughorbel, J.-P. Tarel, and N. Boujemaa. “Generalized Histogram Intersection Kernel for Image Recognition”. In: *IEEE Conference on Image Processing (ICIP)*. 2005.

[10] Lubomir Bourdev and Jonathan Brandt. “Robust Object Detection via Soft Cascade”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2005.

[11] Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. “Describing People: Poselet-Based Attribute Classification”. In: *International Conference on Computer Vision (ICCV)*. 2011.

[12] Lubomir Bourdev and Jitendra Malik. “Poselets: Body Part Detectors Trained Using 3D Human Pose Annotations”. In: *International Conference on Computer Vision (ICCV)*. 2009.

[13] Lubomir Bourdev et al. “Detecting People Using Mutually Consistent Poselet Activations”. In: *European Conference on Computer Vision (ECCV)*. 2010.

[14] Thomas Brox et al. “Object Segmentation by Alignment of Poselet Activations to Image Contours”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2011.

[15] Christopher J. C. Burges. “Simplified Support Vector Decision Rules”. In: *International Conference on Machine Learning (ICML)*. 1996.

[16] C.J.C. Burges and B. Schölkopf. “Improving the accuracy and speed of support vector machines”. In: *Neural Information Processing Systems (NIPS)*. 1997.

[17] O. Chapelle, P. Haffner, and V.N. Vapnik. “Support vector machines for histogram-based image classification”. In: *Neural Networks, IEEE Transactions on* 10.5 (1999), pp. 1055–1064.

[18] Ondej Chum and Andrew Zisserman. “Presented at PASCAL Visual Recognition Challenge Workshop”. In: 2007.

[19] B. Collins et al. “Towards scalable dataset construction: An active learning approach”. In: *European Conference on Computer Vision (ECCV)*. 2008.

[20] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (1995), pp. 273–297.

[21] N. Dalal and B. Triggs. “Histograms of oriented gradients for human detection”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2005.

[22] C. De Boor. *A practical guide to splines*. Springer Verlag, 2001.

[23] Dennis Decoste and Bernhard Schölkopf. “Training Invariant Support Vector Machines”. In: *Machine Learning* 46.1-3 (2002), pp. 161–190. ISSN: 0885-6125.

[24] Richard O. Duda and Peter E. Hart. “Use of the Hough transformation to detect lines and curves in pictures”. In: *Commun. ACM* 15.1 (1972), pp. 11–15. ISSN: 0001-0782. DOI: `http://doi.acm.org/10.1145/361237.361242`.

[25] Alexei A. Efros et al. “Recognizing Action at a Distance”. In: *International Conference on Computer Vision (ICCV)*. 2003.

[26] P.H.C. Eilers and B.D. Marx. “Generalized linear additive smooth structures”. In: *Journal of Computational and Graphical Statistics* 11.4 (2002), pp. 758–783.

[27] P.H.C. Eilers and B.D. Marx. “Splines, knots, and penalties”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* (2005).

[28] M. Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision (IJCV)* 88.2 (2010), pp. 303–338.

[29] R.E. Fan et al. “LIBLINEAR: A library for large linear classification”. In: *Journal of Machine Learning Research (JMLR)* 9 (2008), pp. 1871–1874.

[30] L. Fei-Fei, R. Fergus, and P. Perona. “Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2004.

[31] Li Fei-Fei, Rob Fergus, and Pietro Perona. "One-Shot Learning of Object Categories". In: *IEEE Transaction of Pattern Analysis and Machine Intelligence (PAMI)* 28.4 (2006).

[32] C. Fellbaum. *WordNet: An electronic lexical database*. MIT Press, 1998.

[33] P. Felzenszwalb, D. McAllester, and D. Ramanan. "A discriminatively trained, multiscale, deformable part model". In: *Computer Vision and Pattern Recognition (CVPR)*. 2008.

[34] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. "Pictorial Structures for Object Recognition". In: *International Journal of Computer Vision (IJCV)* 61 (1 2005), pp. 55–79. ISSN: 0920-5691. DOI: `10.1023/B:VISI.0000042934.15159.49`. URL: `http://portal.acm.or`

[35] R. Fergus, P. Perona, and A. Zisserman. "Object class recognition by unsupervised scale-invariant learning". In: *Computer Vision and Pattern Recognition (CVPR)*. 2003.

[36] V. Ferrari, M. Marin-Jimenez, and A. Zisserman. "Progressive Search Space Reduction for Human Pose Estimation". In: *Computer Vision and Pattern Recognition (CVPR)*. 2008.

[37] V. Ferrari et al. "Groups of Adjacent Contour Segments for Object Detection". In: *IEEE Transaction of Pattern Analysis and Machine Intelligence (PAMI)* 30.1 (2008), pp. 36–51. ISSN: 0162-8828. DOI: `http://dx.doi.org/10.1109/TPAMI.2007.1144`.

[38] Vittorio Ferrari, Frederic Jurie, and Cordelia Schmid. "Accurate Object Detection with Deformable Shape Models Learnt from Images". In: *Computer Vision and Pattern Recognition (CVPR)*. 2007.

[39] Vittorio Ferrari, Tinne Tuytelaars, and Luc Van Gool. "Object Detection by Contour Segment Networks". In: *European Conference on Computer Vision (ECCV)*. 2006.

[40] J.H. Friedman and W. Stuetzle. "Projection pursuit regression". In: *Journal of the American statistical Association* (1981), pp. 817–823.

[41] Ashutosh Garg, Shivani Agarwal, and Thomas S. Huang. "Fusion of Global and Local Information for Object Detection". In: *International Conference on Pattern Recognition (ICPR)*. 2002.

[42] Lena Gorelick et al. "Actions as Space-Time Shapes". In: *IEEE Transaction of Pattern Analysis and Machine Intelligence (PAMI)* 29 (2007).

[43] M. Grant and S. Boyd. "CVX: Matlab software for disciplined convex programming (web page and software). http://stanford.edu/$\sim$ boyd/cvx". In: (2008).

[44] K. Grauman and T. Darrell. "The pyramid match kernel: Efficient learning with sets of features". In: *Journal of Machine Learning Research (JMLR)* 8 (2007), pp. 725–760.

[45] K. Grauman and T. Darrell. "Unsupervised Learning of Categories from Sets of Partially Matching Image Features". In: *Computer Vision and Pattern Recognition (CVPR)*. 2006.

[46] Kristen Grauman and Trevor Darrell. "The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features". In: *Computer Vision and Pattern Recognition (CVPR)*. 2005.

[47] G. Griffin, AD. Holub, and P Perona. “The Caltech-256”. In: *Caltech Technical Report*. 2006.

[48] B. Hariharan et al. “Semantic Contours from Inverse Detectors”. In: *International Conference on Computer Vision (ICCV)*. 2011.

[49] Trevor Hastie and Robert Tibshirani. *Generalized Additive Models*. Chapman & Hall/CRC, 1990.

[50] B Heisele et al. “Hierarchical classification and feature reduction for fast face detection with support vector machines”. In: *Pattern Recognition* 36 (September 2003).

[51] M. Herbster. “Learning additive models online with fast evaluating kernels”. In: *Computational Learning Theory*. Springer. 2001, pp. 444–460.

[52] Derek Hoiem, Alexei Efros, and Martial Hebert. “Putting Objects in Perspective”. In: *International Journal of Computer Vision* 80.1 (2008), pp. 3–15–15. ISSN: 0920-5691. DOI: 10.1007/s11263-008-0137-5. URL: http://dx.doi.org/10.1007/s11263-008-0137

[53] Nazli Ikizler and Pinar Duygulu. “Histogram of oriented rectangles: A new pose descriptor for human action recognition”. In: *Image Vision Computation* 27 (10 2009). ISSN: 0262-8856.

[54] Piotr Indyk and Nitin Thaper. “Fast Image Retrieval via Embeddings”. In: *3rd International Workshop on Statistical and Computational Theories of Vision*. 2003.

[55] W. J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2009. URL: http://www.image-net.org.

[56] P. Jain, B. Kulis, and K. Grauman. “Fast image search for learned metrics”. In: 2008.

[57] S. Sathiya Keerthi, Olivier Chapelle, and Dennis DeCoste. “Building Support Vector Machines with Reduced Classifier Complexity”. In: *Journal of Machine Learning Research (JMLR)* 7 (2006), pp. 1493–1515. ISSN: 1533-7928.

[58] C. H. Lampert, M. B. Blaschko, and T. Hofmann. “Beyond sliding windows: Object localization by efficient subwindow search”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2008.

[59] Tian Lan et al. “Beyond Actions: Discriminative Models for Contextual Group Activities”. In: *Neural Information Processing Systems (NIPS)*. 2010.

[60] Ivan Laptev. “On Space-Time Interest Points”. In: *International Journal of Computer Vision (IJCV)* 64 (2-3 2005).

[61] S. Lazebnik, C. Schmid, and J. Ponce. “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2006.

[62] Bastian Leibe, Ales Leonardis, and Bernt Schiele. “Combined object categorization and segmentation with an implicit shape model”. In: *In ECCV workshop on statistical learning in computer vision*. 2004, pp. 17–32.

[63] L.-J. Li, G. Wang, and L. Fei-Fei. “OPTIMOL: automatic Object Picture collecTion via Incremental MOdel Learning”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2007.

[64] Subhransu Maji. *Linearized Additive Classifiers*. 2011. eprint: `arXiv:1110.0879`.

[65] Subhransu Maji and Alexander C. Berg. “Max Margin Additive Classifiers for Detection”. In: *International Conference on Computer Vision (ICCV)*. 2009.

[66] Subhransu Maji, Alexander C. Berg, and Jitendra Malik. “Classification using intersection kernel support vector machines is efficient”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2008.

[67] Subhransu Maji, Lubomir Bourdev, and Jitendra Malik. “Action Recognition from a Distributed Representation of Pose and Appearance”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2011.

[68] Subhransu Maji and Jitendra Malik. *Fast and Accurate Digit Classification*. Tech. rep. UCB/EECS-2009-159. EECS Department, University of California, Berkeley, 2009. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-159.html`.

[69] Subhransu Maji and Jitendra Malik. “Object detection using a max-margin Hough transform”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2009.

[70] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. “Ensemble of Exemplar-SVMs for Object Detection and Beyond”. In: *International Conference on Computer Vision (ICCV)*. 2011.

[71] D.L. Medin and M.M. Schaffer. “Context theory of classification learning.” In: *Psychological review* 85.3 (1978), p. 207.

[72] K. Mikolajczyk, R. Choudhury, and C. Schmid. “Face detection in a video sequence–a temporal approach”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2001.

[73] G. Mori and J. Malik. “Recovering 3d Human Body Configurations Using Shape Contexts”. In: *IEEE Transaction of Pattern Analysis and Machine Intelligence (PAMI)* 28.7 (2006).

[74] S. Munder and D. M. Gavrila. “An Experimental Study on Pedestrian Classification”. In: *IEEE Transaction of Pattern Analysis and Machine Intelligence (PAMI)* 28.11 (2006).

[75] Gregory L. Murphy. *The Big Book of Concepts (Bradford Books)*. The MIT Press, 2002.

[76] Jim Mutch and David G. Lowe. “Multiclass Object Recognition with Sparse, Localized Features”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2006.

[77] F. Odone, A. Barla, and A. Verri. “Building kernels from binary strings for image matching”. In: *IEEE Transactions on Image Processing* 14.2 (2005), pp. 169–180.

[78] Edgar Osuna, Robert Freund, and Federico Girosi. "Training Support Vector Machines: an Application to Face Detection". In: 1997.

[79] Edgar E. Osuna and Federico Girosi. "Reducing the run-time complexity in support vector machines". In: *Advances in kernel methods: support vector learning* (1999), pp. 271–283.

[80] Constantine Papageorgiou and Tomaso Poggio. "A Trainable System for Object Detection". In: *International Journal of Computer Vision (IJCV)* 38.1 (2000), pp. 15–33. ISSN: 0920-5691.

[81] N.D. Pearce and M.P. Wand. "Penalized splines and reproducing kernel methods". In: *The american statistician* 60.3 (2006), pp. 233–240.

[82] F. Perronnin, J. S andnchez, and Yan Liu. "Large-scale image categorization with explicit data embedding". In: *Computer Vision and Pattern Recognition (CVPR)*. 2010.

[83] A. Rahimi and B. Recht. "Random features for large-scale kernel machines". In: *Neural Information Processing Systems (NIPS)*. 2007.

[84] Ali Rahimi and Benjamin Recht. "Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning". In: *Neural Information Processing Systems (NIPS)*. 2009.

[85] D. Ramanan, S. Baker, and S Kakade. "Leveraging Archival Video for Building Face Datasets". In: *International Conference on Computer Vision (ICCV)*. 2007.

[86] D. Ramanan and D.A. Forsyth. "Finding and tracking people from the bottom up". In: *cvpr*. 2003.

[87] Cen Rao, Alper Yilmaz, and Mubarak Shah. "View-Invariant Representation and Recognition of Actions". In: *International Journal of Computer Vision (IJCV)* 50 (2 2002). ISSN: 0920-5691.

[88] Xiaofeng Ren, A.C. Berg, and J. Malik. "Recovering human body configurations using pairwise constraints between parts". In: *International Conference on Computer Vision (ICCV)*. 2005.

[89] Sami Romdhani et al. "Computationally Efficient Face Detection". In: *International Conference on Computer Vision (ICCV)*. 2001.

[90] E. Rosch et al. "Basic objects in natural categories". In: *Cognitive psychology* 8.3 (1976), pp. 382–439.

[91] B.C. Russell et al. "LabelMe: a database and web-based tool for image annotation". In: *International Journal of Computer Vision (IJCV)* 77.1 (2008), pp. 157–173.

[92] K.E.A. Van de Sande et al. "Segmentation as Selective Search for Object Recognition". In: *International Conference on Computer Vision (ICCV)*. 2011.

[93] R.E. Schapire. "A brief introduction to boosting". In: *International Joint Conference on Artificial Intelligence*. 1999.

[94] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001. ISBN: 0262194759.

[95] F. Schroff, A. Criminisi, and A Zisserman. “Harvesting Image Databases from the Web”. In: *International Conference on Computer Vision (ICCV)*. 2007.

[96] C. Schuldt, I. Laptev, and B. Caputo. “Recognizing human actions: a local SVM approach”. In: *International Conference on Pattern Recognition (ICPR)*. 2004.

[97] G. Shakhnarovich, P. Viola, and T. Darrell. “Fast pose estimation with parameter-sensitive hashing”. In: *International Conference on Computer Vision (ICCV)*. 2003.

[98] S. Shalev-Shwartz, Y. Singer, and N. Srebro. “Pegasos: Primal estimated sub-gradient solver for svm”. In: *International Conference on Machine Learning (ICML)*. 2007.

[99] Lifeng Shang et al. “Real-time Large Scale Near-duplicate Web Video Retrieval”. In: *ACM Multimedia*. 2010.

[100] Eli Shechtman and Michal Irani. “Space-Time Behavior Based Correlation OR How to tell if two underlying motion fields are similar without computing them?” In: *IEEE Transaction of Pattern Analysis and Machine Intelligence (PAMI)* 29 (2007), pp. 2045–2056.

[101] Leonid Sigal, Alexandru Balan, and Michael Black. “HumanEva: Synchronized Video and Motion Capture Dataset and Baseline Algorithm for Evaluation of Articulated HumanMotion”. In: *International Journal of Computer Vision (IJCV)* 87 (1 2010). 10.1007/s11263-009-0273-6, pp. 4–27. ISSN: 0920-5691. URL: `http://dx.doi.org/10.1007/s11263-009-02`

[102] Alan F. Smeaton, Paul Over, and Wessel Kraaij. “Evaluation campaigns and TRECVID”. In: *MIR ’06: Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval*. 2006, pp. 321–330. ISBN: 1-59593-495-2.

[103] A. Sorokin and D. A. Forsyth. “Utility data annotation with Amazon Mechanical Turk”. In: *1st Internet Vision Workshop*. 2008.

[104] Michael J. Swain and Dana H. Ballard. “Color indexing”. In: *International Journal of Computer Vision (IJCV)* 7.1 (1991). ISSN: 0920-5691.

[105] *The Canvas Element*. `http://www.w3.org/TR/html5/the-canvas-element.html`.

[106] C. Thurau and V. Hlavac. “Pose primitive based human action recognition in videos or still images”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2008.

[107] A. Torralba, K. Murphy, and W. Freeman. “Sharing features: efficient boosting procedures for multiclass object detection”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2004.

[108] Antonio Torralba, Rob Fergus, and Yair Weiss. “Small Codes and Large Image Databases for Recognition”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2008.

[109] M. Varma and D. Ray. “Learning The Discriminative Power-Invariance Trade-Off”. In: *International Conference on Computer Vision (ICCV)*. 2007.

[110] A. Vedaldi and A. Zisserman. “Efficient Additive Kernels via Explicit Feature Maps”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2010.

[111] A. Vedaldi et al. “Multiple Kernels for Object Detection”. In: *International Conference on Computer Vision (ICCV)*. 2009.

[112] Paul Viola and Michael J. Jones. “Robust Real-Time Face Detection”. In: *International Journal of Computer Vision (IJCV)* 57.2 (2004), pp. 137–154.

[113] G. Wahba. *Spline models for observational data*. Vol. 59. Society for Industrial Mathematics, 1990.

[114] Stefan Walk et al. “New Features and Insights for Pedestrian Detection”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2010.

[115] Gang Wang, Derek Hoiem, and David Forsyth. “Learning Image Similarity from Flickr Groups Using Stochastic Intersection Kernel Machines”. In: *International Conference on Computer Vision (ICCV)*. 2009.

[116] Gregory J. Zelinsky Wei Zhang and Dimitris Samaras. “Real-time accurate object detection using multiple resolutions”. In: *International Conference on Computer Vision (ICCV)*. 2007.

[117] Jianxin Wu and James M. Rehg. “Beyond the Euclidean distance: Creating effective visual codebooks using the histogram intersection kernel”. In: *International Conference on Computer Vision (ICCV)*. 2009.

[118] Changjiang Yang et al. “Improved Fast Gauss Transform and Efficient Kernel Density Estimation”. In: *International Conference on Computer Vision (ICCV)*. 2003.

[119] Ming-Hsuan Yang, Dan Roth, and Narendra Ahuja. “A Tale of Two Classifiers: SNoW vs. SVM in Visual Recognition”. In: *European Conference on Computer Vision (ECCV)*. 2002.

[120] Weilong Yang, Yang Wang, and G. Mori. “Recognizing human actions from still images with latent poses”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2010.

[121] Bangpeng Yao and Li Fei-Fei. “Modeling Mutual Context of Object and Human Pose in Human-Object Interaction Activities”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2010.

[122] Alper Yilmaz and Mubarak Shah. “Actions sketch: a novel action representation”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2005.

[123] J. Zhang et al. “Local Features and Kernels for Classification of Texture and Object Categories: A Comprehensive Study”. In: *International Journal of Computer Vision (IJCV)* (2007).