DETECTING PEDESTRIANS IN STILL IMAGES USING LEARNED SHAPE FEATURES

by

Payam Sabzmeydani

B.Sc., Sharif University of Technology, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2006

APPROVAL

Name: Payam Sabzmeydani

Degree: Master of Science

Title of thesis: Detecting Pedestrians in Still Images Using Learned Shape Features

Examining Committee: Dr. Ze-Nian Li
Professor, Computing Science
Simon Fraser University
Chair

Dr. Greg Mori,
Assistant Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Richard Vaughan,
Assistant Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Anoop Sarkar,
Assistant Professor, Computing Science
Simon Fraser University
Examiner

Date Approved: Dec. 7, 2006

Abstract

The problem of detecting pedestrians in images has received much attention from the computer vision community because of its variety of applications. This problem can be considered as a two-class classification problem by labeling windows cropped from the images as pedestrians or non-pedestrians. We present two novel methods for detecting pedestrians in still images. The first method uses coarse shape cues, and is based on a likelihood ratio test. Likelihoods for shape descriptors on pedestrian and non-pedestrian images are obtained using kernel density estimation. In the second approach, we introduce a new method for learning local discriminative features from training examples, and use them for object classification. This method uses two folds of the AdaBoost classifier, first for feature creation and second to train the final classifier. The quantitative results show that the performance of this method is better than the state of the art pedestrian detector.

Acknowledgments

I am mostly grateful to Dr. Greg Mori who has been, and will always be, more than just a supervisor to me. His constant support has been the biggest encouragement through my studies.

I would like to thank my dear parents for their unconditional love and endless support, from near or far, in every day of my life.

And last, but definitely not the least, a very special thanks to my beloved finace, Maryam, for her constant love and invaluable support in every step I take.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Problem of Pedestrian Detection

Finding and recognizing objects in digital images is one of the main computer vision problems. The terms *Object detection* and *Object recognition* are broadly used in the computer vision literature, and sometimes interchangeably. In this thesis, we define them as follows: *Object recognition* is the task of finding a specific object in an image (e.g. face of a particular person, or a specific book). *Object detection* is finding occurrences of a known object class in an image (e.g. faces, books). In this thesis, we present two new methods for object detection in still images, with experimental results on the pedestrian detection application.

The object detection problem, in general, is a much harder problem than object recognition. For object recognition, the target object is fixed and extracting any feature from that object can be helpful during the recognition task. But in object detection, because of the intra-class differences between the members of the object class, we cannot use the feature cues that are common in object recognition (e.g. color cue). Instead, for the detection task, features that are representative of the intra-class similarities as well as those that are very different between different classes should be used. Extraction of these features is not a trivial task and different machine learning techniques are used to achieve this goal.

Beside the usual difficulties of detecting objects, pedestrians in particular are one of the hardest classes because of their wide range of appearances. Different clothing, different body shapes from person to person, non-rigidity and high dimensionality of human body poses are some of the main factors that make the problem of finding pedestrians a harder problem than detection of many other objects.

In our work, like most of the other object detection systems, we assume we have only two classes: the object class (pedestrians) and the non-object class (background). The goal is to train a classifier that given an image, it can correctly predict its class; object class or non-object class.

In this work, we present two new approaches for the pedestrian detection problem, each employing different training and testing approaches. The performance of our second detector is better than the current state of the art pedestrian detectors, therefore the main focus of this thesis will be on that method.

1.2 Pedestrian Detection Applications

Robust detection of pedestrians in images is important for many applications, such as surveillance systems, robotics, intelligent vehicles, and image search and retrieval. Some of these applications, such as intelligent vehicles and robotics, are very sensitive to noise and erroneous detections can lead them into disastrous accidents. Therefore to apply automatic detection systems in these applications we need to have very high accuracy detection systems, that can minimize the number of detection misses and false alarms at the same time. Another important factor other than the accuracy, is the running time of the detection system. Almost all the named applications need a real-time detection system. In the past few years, new detection approaches with the help of computer hardware improvements, have achieved this goal and there is a growing use in object detection systems in real life applications.

Currently, most surveillance systems rely on constant human supervision which is both costly and error-prone due to operator fatigue. By having a reliable pedestrian detector, less crowded surveillance scenes can be monitored semi-automatically, with less need of human supervision and reduced amount of error. A surveillance system can provide us video data, which contains more information than just a still image. But still by having a better still image pedestrian detector embedded into a video pedestrian detector, we can improve the results of the detection system.

Another useful application can be image and video search and retrieval. You might want to search for some parts of a video or some images from a database, by just entering some text queries related to existing objects in them. By having reliable object detection systems, an image database can be annotated and indexed using that systems beforehand.

Such system can be useful on wide variety of systems like search on the rapidly growing video content on the web or summarizing the surveillance video contents.

Pedestrian detection systems are also incorporated into automatic driver-assistance systems in intelligent vehicles. If the car itself can detect other objects in the road, especially pedestrians, it can monitor the road and alarm the driver about dangerous conditions, or in serious cases it can take action by itself to prevent the accident.

1.3 Approach

As mentioned before, pedestrians are one of the most challenging categories for object detection. But because of the wide range of applications that exist for pedestrian detection, many researchers have contributed to the problem.

To find pedestrians in an image we use a window-scanning method. We train a fixed size image classifier (i.e. 64×128 pixels in our experiments), that can classify images of that size as a pedestrian or a non-pedestrian. To search an image bigger than the size of this detection window, we scan the image exhaustively in all possible locations and scales. We crop different parts of the image with the classifier size and test them with the classifier. This way we can find all the pedestrians in the image with different sizes not smaller that our detection window. Figure 1.1 shows examples of pedestrian and nonpedestrian images from our training set that are cropped to our classifier size.

In any detection system, we need to train the detector first, and to do so, we need two sets of training examples, one for each object class (i.e. pedestrian class and non-pedestrian class). Using these training examples one can try to estimate the distribution of each class in an arbitrary feature space. Another approach is to build a discriminative classifier of the two classes in the feature space instead of estimating the whole distribution. In this thesis we will present two methods, each applying one of these approaches to solve the problem of pedestrian detection. We will refer the first method as the *Kernel Density Estimation* method and the second one as *AdaBoost* method. Both methods use the same low level signals (features) to build their models on them. These low level signals are the gradients of image intensity in different directions. Figure 1.2 shows some examples of pedestrian and non-pedestrian images and their extracted gradients in two different directions. We will describe these two methods in detail in Chapter 3 and Chapter 4.

Figure 1.1: Examples of pedestrian and non–pedestrian images.

1.3.1 First Method: Kernel Density Estimation

In the first method, we build statistical likelihood distributions for each pixel. To estimate class distributions, we use kernel density estimation, using geometric blur descriptors (GBD) [4] as the feature space in which the kernels are defined. An overview of the algorithm is as follows:

In the training phase, we take every training image belonging to both classes, and we extract the low level signals that we use for our geometric blur descriptors. These low level signals are simple gradients in four different directions. Using the gradient signals, we compute a geometric blur descriptor at each pixel for every image. Considering each pixel location separately, we have a number of GBDs for that pixel (each belonging to one training image) for both classes. By using each of these descriptors as one kernel in the descriptor space, we will create Kernel Density Estimation of our GB Descriptors for both classes at all the pixels.

In the test (detection) phase, for each test image, we will compute the per pixel geometric blur descriptors the same way we did for training images. At this point by using the kernel density estimates for both classes, we compute the likelihood of each pixel belonging to each of the classes (person and non-person). By using a coarse likelihood ratio test, which includes the likelihood of all the pixels, the detector will decide which class the image belongs to.

Figure 1.2: Examples of pedestrian and non-pedestrian images from our training set and their intensity gradients in x and y directions.

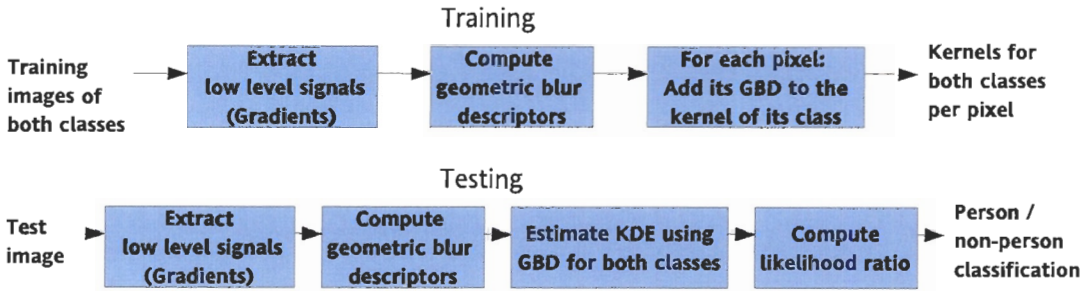


Figure 1.3: An overview of our Kernel Density Estimation detector. First graph is the training process and the second one shows the test process.

1.3.2 Second Method: AdaBoost

Our second method for pedestrian detection uses AdaBoost to model the boundary of the two classes. The training part of this approach is consisted of three layers. An overview of these layers is as follows:

1. *Low-level Features; Weak Classifiers:* The input to this layer is raw training images. We extract the gradients responses of each image in different directions, and compute simple sums of these responses around each pixel. These sums are the low level features and will be used as the weak classifiers of the next step.

2. *Mid-level Descriptors; Semi-strong Classifiers:* We use a boosting algorithm, AdaBoost [9], to select subsets of the weak classifiers from the first layer to construct some better classifiers in some local regions of the detection window. These boosted classifiers will act as our mid-level descriptors. We run the AdaBoost for some arbitrary small windows in the image, using as its input, only the weak classifiers falling in that particular window. The output of this layer are some local discriminative features in the detection window. Figure 1.4 shows a high level illustration of one mid-level feature inside the detection window. This feature contains some low-level features that are usually different in the two pedestrian and non-pedestrian classes.

3. *Final Classifier:* The mid-level feature descriptors from previous layer can only act in a local neighborhood in the image and therefore their overall classification power

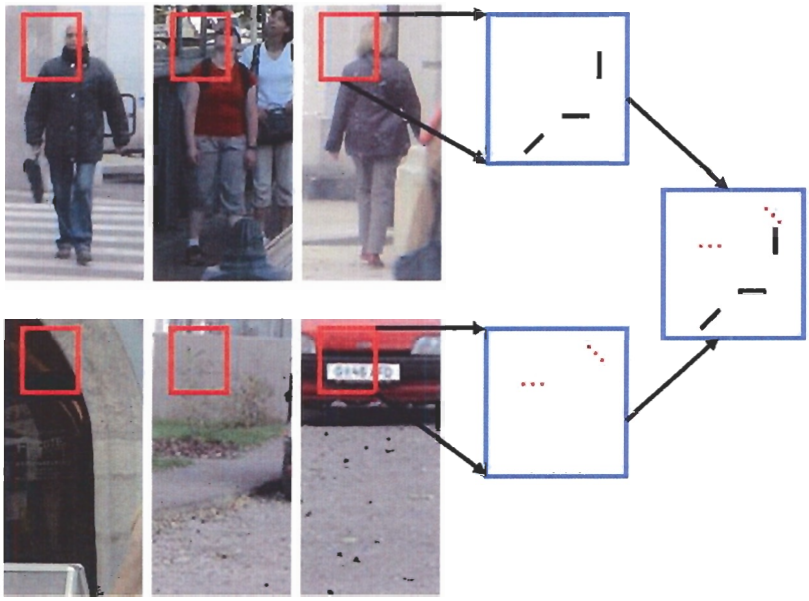


Figure 1.4: Illustration of mid-level features. A mid-level feature is a set of some low-level edge features that are common in one object class but not in the other class.

is still far below an accepted point. By merging them together we can combine the information from different parts of the image. In order to archive this goal, we use AdaBoost for the second time to combine those information and train our final classifier. This time we use our mid-level descriptors as its input, and the algorithm will choose the best subset among them that can separate the two classes as much as possible.

Figure 1.5: An overview of the training phase of our AdaBoost detector

The main contribution of this thesis is the introduction of the trainable mid-level descriptors. These descriptors are low-dimensional, highly discriminative, and easily scalable. We will discuss these attributes in more detail and justify them in chapter 4.

In the detection phase, We only need to access a subset of low-level features in each window. This subset contains those low-level features that form the mid-level features selected by the final classifier. Therefore the detection process is very fast and can be applied in real-time. Figure 1.5 and Figure 1.6 illustrate the described training and testing procedures.

Figure 1.6: An overview of the testing part of our AdaBoost detector

The main difference between this approach and previous approaches to this problem, is the existence of the middle layer. Most other pedestrian detectors use a set of fixed and pre-defined descriptors and use different machine learning algorithms to train a classifier using those descriptors. For example Dalal and Triggs [6], use a set of pre-defined set of histograms of oriented gradients (HoG) or Wu and Nevatia [51] use a set of fixed edgelet features as their descriptors. These sorts of fixed features can cause two kinds of problems: First, the defined feature set cannot guarantee that it includes all the possible discriminative features for the two classes [51]. Second, feature set might contain too many information-less features that adds too much unwanted noise to the final classifier, and therefore decrease the detection rate [6].

AdaBoost, is an adaptive boosting algorithm that is widely used in machine learning applications and more specifically for object detection problems. Viola and Jones [47] use a cascade version of AdaBoost for fast face detection using Haar-like wavelet features as the weak classifiers, and in later work Viola et al. [48] use the same AdaBoost configuration for pedestrian detection. Wu and Nevatia [51] use a nested AdaBoost for pedestrian detection with edgelet features as its input.

Chapter 2

Previous Work

The problem of object detection and particularly pedestrian detection has received much attention from the computer vision community. The attempt for analyzing and modeling humans and their motion from the video and image data goes back to more than two decades with the early works of O'Rourke and Badler [31] and Hogg [17]. Another old approach is the work by Rohr [36], where he finds and tracks pedestrians using a model-based approach. He represents the human body by a volume model and match the contours of these models with edges in the test image to extract different body parts. Various employed methods can be categorized in different ways such as the underlying appearance measures, higher level features used to bundle the raw measures, and the classifier they adopt.

There are a couple of good surveys about human tracking and motion analysis, for example Gavrila's survey on visual analysis of human movement [12] and Moeslund and Granum's survey about vision-based human motion capture [25]. There are also some surveys on related object detecting systems (e.g. Hjelmas and Low survey for face detection [16]), that explain different approaches for those applications. Unfortunately there is no survey that solely focus on detecting pedestrians in static images. This chapter presents a brief review of different approaches for pedestrian detection and categorizes them based on their similarities and differences.

The problem of pedestrian detection is one of the challenging problems in object detection. Variability in clothing, pose, and lighting, the presence of background clutter, and the small number of pixels with which pedestrian detection must be performed, make this object detection task a difficult one. Because of these difficulties a wide variety of methods has been used by different researchers during the last decades to solve this problem.

These different methods, in general, have many similarities and differences. In this chapter we will categorize these methods from different point of views and point out some of their advantages and disadvantages.

2.1 Object Representation

For any object detection task, we need to represent the object in a feature space using the available visual cues. The two main cues of information for detecting objects are their color and edge-formation (texture can be considered as a higher level combination of these cues). In some object recognition methods color is being used as the main feature cue [46]. But in most object detection problems, such as pedestrian detection, because of the color differences between different objects in the same object class, this cue cannot be used in a straightforward way. As a result, edges are considered as the best feature cue for object detection and are used widely in a variety of different detectors. Edge information can be captured and processed in many different ways in different object detection applications. It can be extracted using simple gradient filters, difference of Gaussian filters, wavelet filters, or even background subtraction methods. The edge information then might be processed in different ways to form some descriptors and templates such as histograms (e.g. HoG [6] and SIFT [22]), sets of edge samples (e.g. geometric blur [5] and edgelet features [51]), or overall body formation (e.g. distance transforms [14]).

To capture the shape of a human (or any other object), one can focus on local features and use them for detecting the whole body, or can use a global feature that captures the whole body shape at once. In the following section we will briefly describe some of the approaches that use global or local features to detect pedestrians.

2.1.1 Global Features

Methods that use global features for detection try to make a model of the objects shape as one unit. One popular kind of these features is the silhouette of the object. By using background subtraction, one can extract the silhouette of the objects and use that as the object's representation. Using background subtraction has its own problems, and limitations. It is not feasible to use background subtraction for still images, and we need to have video data. Also the video data should be captured from a stationary camera to make the background be the same in every frames.

The extracted foreground information is mostly used in two ways; some only capture the silhouette boundaries as the object feature, but others use all the edge information inside this silhouette (foreground) region and use that as their global cue. In both cases edge is the feature that is being considered, one is the edge between the foreground (object) and the background, and the other the edges inside the objects. To distinguish them during this chapter, we will call the former *silhouette* and the latter *edge map*.

One way to formalize edge information is to convert them into a distance transform, which is a common way of matching two edge maps. Gavrila and Philomin [14] design their object detection method based on distance transforms and template matching. Their method uses a template hierarchy to capture the variety of object shapes. They use chamfer matching (introduced by Barrow et al. [3] for image matching) to compare the test image edge map with the hierarchy templates to find the best match in an efficient way. In a later work Gavrila et al. [13] integrate an extended version of their previous work with some additional modules (such as trajectory estimation) into an actual vehicle.

Hausdorff distance was first used as a measure for image comparison by Huttenlocher et al. [20]. Felzenszwalb [7] uses Hausdorff distance for a human detection system. He trains a single model by making a probability model from Hausdorff distances of edge maps of the training examples.

Zhao and Nevatia [53, 54] use human shape models to interpret the foreground in a Bayesian framework by employing Markov chain Monte Carlo techniques. Their features are the foreground silhouettes computed by using background subtraction, and their Bayesian model has the information of various aspects including human shape, human height, camera model. They use the silhouettes of the moving objects as their detection features. Relying on background subtraction for object detection has its own advantages and disadvantages. The main advantage is its simplicity and the small amount of computation needed. But on the other hand, it will limit the problem to use of temporal (video) data which contains the motion information necessary for discriminating between foreground and background. Another constraint for such methods, is that the video data should be captured from an stationary camera to enforce a fixed background in the whole video. It is also hard to apply such methods in dynamic environments where objects other than the target object (pedestrians) move. Distinguishing different objects' silhouettes is not an easy task because of the little amount of information the silhouette captures from the object. Silhouette information only contains the boundary between the object and its background and no

information about the inside or outside of this boundary.

Leibe et al. [21] use silhouettes as a global feature on top of some local features. Their detector is based on local features, but as a post-processing step, they use global silhouette information to segment the target person and reason about occluded people as well. They use chamfer matching to match their hypotheses with the model silhouette boundaries that they have. We will describe their approach in more detail in section 2.1.2.

One parallel research trend with the problem of pedestrian detection, is the problem of human 3D pose estimation. The low-level features that are being used in such approaches are very similar to those used for detection systems. The reason is that there is always the need to detect the human before extracting its 3D pose. We will shortly discuss some of the features that are used in such approaches. Sminchisescu and Triggs [44, 45] use a global edge map combined with the motion boundary information for 3D body tracking. Agarwal and Triggs [1] capture the human 3D pose by using silhouettes as their features. They encode the silhouette shape using shape contexts descriptors. For finding the pose from these descriptors they use relevance vector machine (RVM) regressors. Rosales and Sclaroff [38] use global silhouettes of humans and extract low-level visual features from it. They find the best 3D pose match using a maximum likelihood criterion on possible solutions. As a follow up to their works, Rosales et al. [39] estimate the 3D body pose by using multiple views of a person. Again, they use the same silhouette based features combined with specialized mapping architecture(SMA) procedure in their approach. Mori and Malik [27] use the global edge information to estimate the 3D human body configuration. They first obtain a set of sample points over the boundary of the joints and extract the shape contexts descriptors at those points. For pose estimation, they find best matches of these exemplar points to those from the test image to find 2D joint positions. Then these 2D positions are used to construct an estimate of the 3D body configuration.

The main problem with global features is that they are not robust with respect to articulation, occlusion, and pose variations. They might also miss some parts of edge information, depending on the approach they are being used. For example if one is only using the silhouette boundary information, he is discarding the edge information of inside and outside of the silhouette, that might be important for the perfect detection.

2.1.2 Local Features

Because of the problems that were mentioned in section 2.1.1 about the global features, most systems that apply these features, use them in very controlled environments, or as a feature for a validation step. Local low-level features are used in a wider range of applications for object detection because of their ability to handle many cases (i.e. articulation, and occlusion) that global features are not. One big advantage of using local features is their capability to handle flexible objects that can change their shape (e.g. pedestrians), and handle small occlusions. When small bits of shape information are used, we can handle local changes better because the object model is not a one-unit model anymore. One of the main difficulties in using local features, is defining them. Different approaches apply different local features for pedestrian detections.

Oren et al. [30] have developed one of the first pedestrian detection techniques that uses an supervised method to train a detector. Their detection technique is based on the wavelet templates and defines the shape of an object in terms of a subset of the wavelet coefficients of the image. They try both template matching and SVM as their classifiers. Papageorgiou and Poggio [33] extend the same framework and improve the results and the running time.

Schneiderman and Kanade [42] present a method that represents the statistical distribution of object and non-object classes using a product of histograms. The features that they use to make the histograms are wavelet coefficients of different sizes and scales. They test their method to detect faces and cars. They improve their approach in a later work [43] by introducing localized parts, subsets of their features, into their detector. They train weights of the features inside each part using AdaBoost.

Viola and Jones [47] use Haar wavelet features as the weak classifiers. They use AdaBoost to combine many of these weak classifiers together to create a strong classifier. Then they create a cascade of these strong classifiers to make their system faster by rejecting many of the possible faces in the early layers of the cascade. They use their system to detect faces. Their method is much faster than many other approaches as a results of using integral images for their wavelet computations and also their cascade algorithm. In a later work, Viola et al. [48] use their system for pedestrian detection. They enhance their system by adding some temporal Haar-like features to it to capture pedestrian movements as well as its shapes. Okuma et al. [29] adopt the same work and used it for a human detector. They introduce a particle filter tracker that uses the detections of an AdaBoost detector to

initialize or correct the tracks.

Agarwal and Roth [2] make their object detector based on a sparse, part-based representation of objects. They build a vocabulary of parts that can be used to represent objects in the target class. They make the vocabulary by detecting interest points on the object classes and storing the intensity image around the points in a clustered database. They classify objects by finding correspondences between test images and stored parts in their database.

Mohan, Papageorgiou and Poggio [26] detect pedestrians as a combination of parts. Their system is structured with four distinct example-based detectors that are trained to separately find the four components of the human body: the head, legs, left arm, and right arm. After detecting these separate parts, they use a second example-based classifier to combines the results of them to classify a pattern as either a person or a non-person. For each part, an individual Support Vector Machine (SVM) classifier is trained using Haar wavelet features.

Lowe [22] introduces a method for extracting distinctive invariant features from images. He uses these features to perform reliable matching between different views of an object to perform object recognition. These Scale Invariant Feature Transforms (also known as SIFT features) are robust to scale and rotation. Because of the distinctive nature of these features a single feature can be matched against a large database of features from many objects. For the recognition task, he uses a nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and verify the object through least-squares solution for consistent pose parameters. These features are used in many generic object recognition tasks, including pedestrian detection [21].

Mikolajczyk, Schmid, and Zisserman [24] also use a part based detector to detect humans. They model humans as assemblies of seven parts. These parts are represented SIFT-like orientation base features which captures the spatial layout of the parts' appearance. Feature selection and the part detectors are learned using AdaBoost.

Dalal and Triggs [6] use histograms of oriented gradient descriptors (HoG) as their features and train a pedestrian detector using SVM classifier. They compare a variety formations of their HoG features, and SVM settings to tune their detector. They also analyze the effect of different image and feature normalizations, on the accuracy of their detector. Figure 2.1 illustrates the features they use for their detector.

Berg and Malik [4] introduce a new feature descriptor called geometric blur. Geometric

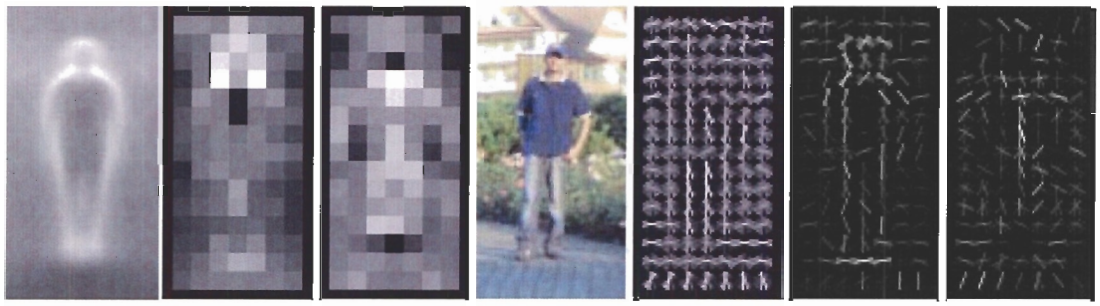


Figure 2.1: Dalal and Triggs' feature representation [6]. (a) The average gradient image. (b) Maximum positive SVM weight for different blocks. (c) Likewise for negative SVM weights. (d) A test image. (e) It's computed HOG descriptor. (f,g) HoG descriptors weighted by respectively the positive and the negative SVM weights. (©2005 IEEE, by permission)

blur descriptor is consisted of blurs of a signal around an interest point with a varying blur standard deviation relative to the distance from the point. They use these features for object detection. In a later work Berg, Berg and Malik [5] use the same features for object class recognition. They find descriptor correspondences between images and minimize a cost function over their geometric distortion. In one of the approaches proposed in this thesis we will use these geometric blur descriptors for pedestrian detection.

Wu and Nevatia [51] detect pedestrians with inter-occlusions. They introduce and use edgelet features, which are short line or curve segments with known direction. Then they train three part detectors (head-shoulder, torso, and legs) with these features, using a variation of AdaBoost [19]. They combine the responses of these part detectors to form a joint likelihood model that can detect inter-occluded pedestrians by using a MAP estimation. Figure 2.2 illustrates the definition of edgelet features.

Leibe, Seeman and Schiele [21] use local and global cues to detect pedestrians in crowded scenes. They start with local feature point detection and store histograms of gradients features, used in [22], in a database. Then they use clustering algorithms for matching feature point descriptors gathered during training with the descriptors seen in the test images. A global top-down verification process using silhouette boundary and chamfer matching is applied at the end. Mikolajczyk, Leibe, and Schiele [23] also refine the same approach for an object class recognition task. They also evaluate different scale invariant

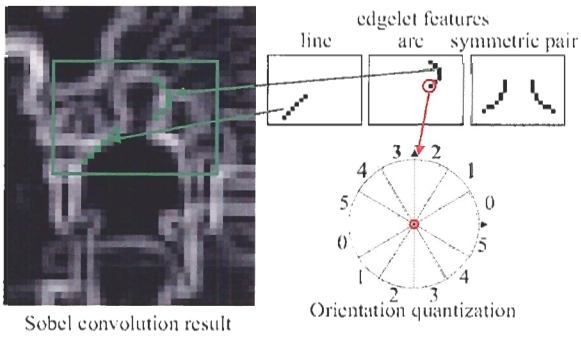


Figure 2.2: Edgelet features used by Wu and Nevatia [51] (©2005 IEEE, by permission)

region detectors and descriptors.

Lately, Munder and Gavrila [28] studied the problem of pedestrian classification with different features and classifiers. They compared PCA coefficients (global feature), Haar wavelets (nonadaptive local feature), and local receptive fields (adaptive feature) as their features and showed that local adaptive features such as local receptive fields can do a better job in representing pedestrians. They also compared different classifiers; support vector machines, neural networks, AdaBoost, and k-nearest neighbor classifiers, and showed that SVMs and AdaBoost classifiers outperform the other classifiers tested.

Some of the approaches that were described (e.g. [22] and [21]), rely on extracting distinctive feature points from the object and match them in detection time. Some other (e.g. [47], [6] and [51]) make a huge feature set, and use machine learning algorithms, such as AdaBoost and SVM, to choose a useful combination of them. Other approaches (e.g. [42]) use simpler features and use a coarse representation of the features on the whole image. All these approaches are pre-defining the features that they want to use. This can affect the classification results because of missing information that were not captured with the defined features. In the second approach proposed in this thesis in chapter 4, we use AdaBoost to train local features that are good for the classification of our two object classes (pedestrian and background) and use these features throughout the rest of approach.

2.2 Classification

Classification method is another part that differs in different pedestrian detectors. There are some factors that have influence in choosing the classifier, such as selected feature descriptors, required type of output, running time, and accuracy.

If the selected features are global features, nearest neighbor methods are mostly a good choice for the classifier. Gavrila et al. [14] use this approach by matching the distance transform of the test image with template image in a hierarchical way to find the nearest neighbor.

Some applications (e.g. segmentation) might need per pixel likelihoods of the target object after the detection phase. Density estimation approaches can provide these kinds of outputs by estimating statistical distribution of features for both classes at every pixel. For this purpose the statistical distribution of each class is estimated using maximum a posteriori estimation (MAP) over the feature domain. Schneiderman and Kanade [42] use the same approach to detect faces and cars, and Schiele and Crowley [41] use it for object recognition.

Boosting, and particularly AdaBoost, is another popular group of classification algorithms. AdaBoost's success is because it does not enforce any structure on features it needs as the input. It combines any set of non-perfect classifiers to make a strong classifier out of them. Viola et al. [47, 48] use a cascade of AdaBoost classifiers for face and pedestrian detection. Wu and Nevatia [51] also use a nested AdaBoost to detect humans.

Support vector machines (SVM) are another set of classifiers that are largely used in machine learning and computer vision applications (Refer to Webb [50] and Hastie et al. [15] Books for more detail). Dalal and Triggs [6] use a linear SVM classifier to detect pedestrians. Ronfard, Schmid, and Triggs [37] build an articulated body detector by using SVM based classifiers. They find people in static frames using learned models of both the appearance of body parts (head, limbs, hands), and of the geometry of their assemblies. Their articulated body detector use a variation of dynamic programming for efficiently assembling candidate parts into pictorial structures. They train dedicated detectors learned for each body part using SVM and Relevance Vector Machines (RVM).

Most of these classification methods, classify object classes by constructing an imaginary decision boundary around them in their feature space. Nearest neighbor classifiers, support vector machines (SVM), clustering algorithms, boosting approaches, and neural networks

all fall into this category.

Freund and Schapire [11] analyze the relation between two of the most common classifiers in object detection, SVM and AdaBoost. They point out that although the goal of both classifiers is to implicitly maximize the minimum margin of training examples they have several important differences. The most relevant difference, with respect to object detection applications, is that most of the actual work involved in applying SVM or AdaBoost has to do with selecting the appropriate kernel function in the one case and weak learning algorithm in the other. As kernels and weak learning algorithms are very different, the resulting learning algorithms usually operate in very different spaces and the classifiers that they generate are extremely different.

Chapter 3

Detecting Pedestrians Using Coarse Shape Cues

In this thesis, we explore two different approaches for pedestrian detection in still images. The first approach uses coarse shape cues to make a statistical representation of object classes. The second method, is based on training the feature set from the training data, instead of pre-defining it, to capture all the possible information for the classification process. In this chapter we will present the first approach. The common aspect of these two detectors is their underlying low-level features. These low-level features, are gradients of the image captured in four different directions ($\{0°, 45°, 90°, 135°\}$).

The method that will be described in this chapter is based on a likelihood ratio test. Likelihoods for shape descriptors on pedestrian and background images are obtained using a kernel density estimation. These kernels are built on geometric blur feature space for each pixel. For an illustrated overview of the approach refer to Figure 1.3 in the Chapter 1.

The method that we use to find pedestrians in an image is a scan-based method. We train a classifier that given a fixed size window (i.e. 60×100 in our experiments) it can decide whether it is an image of a pedestrian or not. To search a complete image, we scan the image exhaustively in different locations and different scales. In other words for every possible scale, we put our detection window on all the locations and will test that window by running our classifier to decide about that part of the image. This way we can find all the pedestrians in the image with different sizes not smaller that our detection window.

3.1 Statistical Representation of Pedestrians

We operationalize the notion of coarse shape using the geometric blur descriptors [4]. This detector falls into the category of likelihood ratio-based detectors. For each pixel, we build likelihood distributions over geometric blur descriptors for pedestrian and background classes. These likelihoods are obtained using kernel density estimation. We show that the geometric blur descriptors can capture this coarse shape information and remain robust against its variation in pedestrians, and are smooth enough to facilitate the use of kernel density estimation in spite of their high dimensionality. Moreover, this method, based on per-pixel likelihood ratios, is inherently a parts-based detector. We provide qualitative evidence that it can handle situations involving partially occluded pedestrians, a situation that will be challenging for methods which build a single descriptor for an entire window.

This detection method is motivated by the work of Schneiderman and Kanade [42], who detect faces and cars by building histogram representations of wavelet feature likelihoods for object classes and a background class. The basis of our approach is different from theirs in two ways. First, our features are descriptors based on oriented edges which gather shape information over a larger spatial extent. Second, these features are high-dimensional, and we cannot use histograms to model the statistical distribution of $P(image|person)$ and $P(image|nonperson)$. Instead we compute kernel density estimates for both object (pedestrian) and non-object (background) feature likelihoods and represent class probabilities with them. Therefore $P(image|person)$ and $P(image|nonperson)$ can be modeled by a set of kernel densities, computed from person and non-person image features. The detection condition can be shown as:

$$\frac{P(image|person)}{P(image|nonperson)} > \lambda \tag{3.1}$$

Where λ is the threshold we can choose for different detection accuracies. If the ratio is greater than λ, we say that a pedestrian exists in the query window. This likelihood ratio is equivalent to maximum a posteriori estimation.

The detection condition in Equation 3.1, should originally contain the likelihood probabilities $P(person|image)$ and $P(nonperson|image)$. But because we cannot estimate these two probabilities, we use Maximum a posteriori (MAP) estimation and use the posterior probabilities $P(image|person)$ and $P(image|nonperson)$ instead. We apply the Bayes rule as follows:

$$\begin{aligned}\frac{P(person|image)}{P(nonperson|image)} &= \frac{P(image|person)P(person)/P(image)}{P(image|nonperson)P(nonperson)/P(image)} \\ &= \frac{P(image|person)}{P(image|nonperson)}\frac{P(person)}{P(nonperson)} \\ &= \alpha\frac{P(image|person)}{P(image|nonperson)} \end{aligned} \tag{3.2}$$

This way, we can estimate the distributions $P(image|person)$ and $P(image|nonperson)$ instead. As the *image* variable is still very high dimensional, we break it into smaller domain variables (features extracted from the image). In order to have a tractable model of the likelihood, we will assume that the features within an image window are independent. In our approach, for each class we build n different probability distributions for both person and nonperson classes $P(f_i|person)$, $P(f_i|nonperson)$, $(i = 1, \ldots, n)$, each one modeling the distribution of one feature f_i. Because of some further advantages we assign one feature to each pixel in the image. As we will show later, this will help us to have a person-ness measure for each pixel. We approximate the probability ratio $\frac{P(image|person)}{P(image|nonperson)}$ by:

$$\frac{P(image|person)}{P(image|nonperson)} \approx \frac{\prod_{i=1}^{n} P(f_i|person)}{\prod_{i=1}^{n} P(f_i|nonperson)} \tag{3.3}$$

In forming equation 3.3 we implicitly assume that our features are statistically independent for both classes. However, Schneiderman and Kanade [42] have shown that this independence assumption can be relaxed because our goal is classification not probabilistic modeling. As they show, we can consider a classification example based on two random variables, A and B. If we assume that A is a direct function of B, $A = f(B)$, we will have $P(A = f(B)|B) = 1$. The optimal classifier is:

$$\frac{P(A, B|object)}{P(A, B|nonobject)} = \frac{P(B|object)}{P(B|nonobject)} > \lambda \tag{3.4}$$

and if we falsely apply the independence assumption, our classifier becomes:

$$\begin{aligned}\frac{P(A, B|object)}{P(A, B|nonobject)} &= \frac{P(A|object)P(B|object)}{P(A|nonobject)P(B|nonobject)} \\ &= \left(\frac{P(B|object)}{P(B|nonobject)}\right)^2 > \gamma \end{aligned} \tag{3.5}$$

This shows that we can achieve the optimal classification, by choosing $\gamma = \lambda^2$ even though our features were not independent of each other.

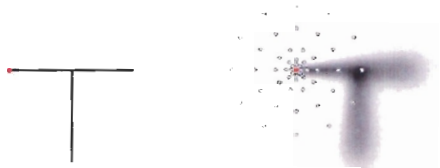
Our features f_i are formed using geometric blur descriptors. These features are vectors in $\mathcal{R}^m$, where m is the number of samples that we get from the geometric blur. For each pixel we create one feature vector f_i to represent the shape of our classes at that particular point of the image.

In the following sections we provide the details of the features used, and the estimation of the individual likelihoods.

3.1.1 Shape Features–Geometric Blurs

A variety of features based on filter responses have been employed for use in pedestrian detection. Dalal and Triggs [6] explore the use of oriented gradients, and use histograms of these gradients in their detector. Haar-like features [48] and features from SIFT-based descriptors [21] have also been used for the pedestrian detection application.

In our approach we will be modeling the feature likelihood as a collection of independent features computed at each pixel. As such, our descriptors will need to be more informative than the typical small scale filter responses typically used – they must be able to capture the shape of the pedestrian at each pixel.



a. *b.*

Figure 3.1: (a) A sparse signal S. (b) The geometric blur of S around the feature point marked in red. We only sample the geometric blur of a signal at a fixed set of points $\{s_i\}$. (©2005 IEEE, by permission [5])

To this end, in this method we use the subsampled version of the geometric blur descriptor [5] (introduced by Berg and Malik [4]). The geometric blur descriptor is a spatially varying blurred version of a chosen signal (oriented gradient responses in our work) around

a feature point. Given a signal S, the geometric blur descriptor around location x_0 is:

$$B_{x_0}(x) = S_{\alpha|x|+\beta}(x_0 - x) \tag{3.6}$$

Where $S_d = S * G_d$ is a blurred version of S, as a result of convolving it with a Gaussian kernel of standard deviation d. α and β are constants that determine the amount of blur. By sampling $B_{x_0}(x)$ at a sparse set of points $x = s_i$ (figure 3.1), we form our final feature vector as a sampled version of geometric blur descriptor. We choose our samples radially in a number of angular directions. We will show experiments with different designs of the sampling method in section 3.2.

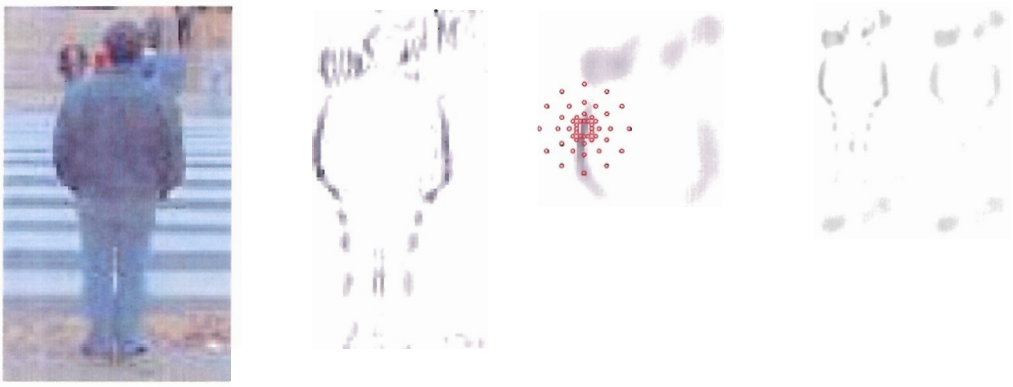
One obvious gain of using these descriptors is their robustness to geometric distortion which is one of the main problems in detecting pedestrians and other objects. In design of a geometric blur descriptor three parameters influence the resulting feature vector: input signals, blur-kernels and sub-sampling method.

To incorporate the edge information into the geometric blur descriptor, we use oriented gradient responses as the input signals of our geometric blur descriptors. We create four geometric blurs for each image using oriented gradients in four different directions. Figure 3.2 illustrates one of the signals and its geometric blur for a particular point. The final descriptor is a vector consisting of a number of samples for each of the four geometrically blurred gradient response channels. In our experiments we use 3-4 radial samples and 8-12 angular samples in our experiments resulting in a 96-192 dimensional vector.

After computing the feature vector for each location, we use L2-norm to normalize the vector. This normalization is crucial because gradient strengths vary over a wide range due to different background, clothing, and illumination changes.

3.1.2 Classification–Kernel Density Estimation

By applying the discussed geometric blur descriptors on any image, we will have a feature vector f_i for every single pixel i. The dimensionality of these vectors are very high in comparison to the number of training examples. Therefore, describing an object category in this space is not trivial and commonly used methods such as histograms cannot be applied. Instead, we use Gaussian kernel density estimators to model our object distribution in each of these feature spaces. The probability distribution $P(f_i|person)$ (and similarly



(a) (b) (c) (d)

Figure 3.2: (a) An example of a pedestrian. (b) The oriented gradient signal in one of four directions (x axis). (c) Geometric blur descriptor, and sample points at a single location. (d) Blurs for 4 sample distances. Because the sample points are fixed around the feature point, these blurs are computed once for each image

$P(f_i|nonperson)$) can be estimated by using all the feature vectors, observed at location i of the images in the training set:

$$P(f_i = x|person) \propto \frac{1}{n} \sum_{k=1}^{n} e^{\frac{-d(x,f_i^k)^2}{2h^2}} \tag{3.7}$$

where n is the number of training images of person category, f_i^k is the observed feature vector at location i in the k'th training image, and $d(\cdot,\cdot)$ is the L2 distance of two vectors in the feature space. Parameter h defines the bandwidth of samples used to compute the density estimate. As we will see in Section 3.2, tuning the parameter h is an essential task, and detector's accuracy is highly dependent to its value. For more details on kernel density estimation refer to Wand and Jones [49] and Webb [50].

Figure 3.3 illustrates an example of kernel density estimation of two classes with some sample points in an one dimensional space. Note how the kernels become smoother as we increase the bandwidth. With more samples, and in higher dimensions, these KDEs will become much more complex, and choosing the right bandwidth to represent the two classes will become a critical task.

A common concern when using kernel density estimation in high dimensional spaces (e.g. 192-dim geometric blur) is the ability to fill this space with a limited supply of training

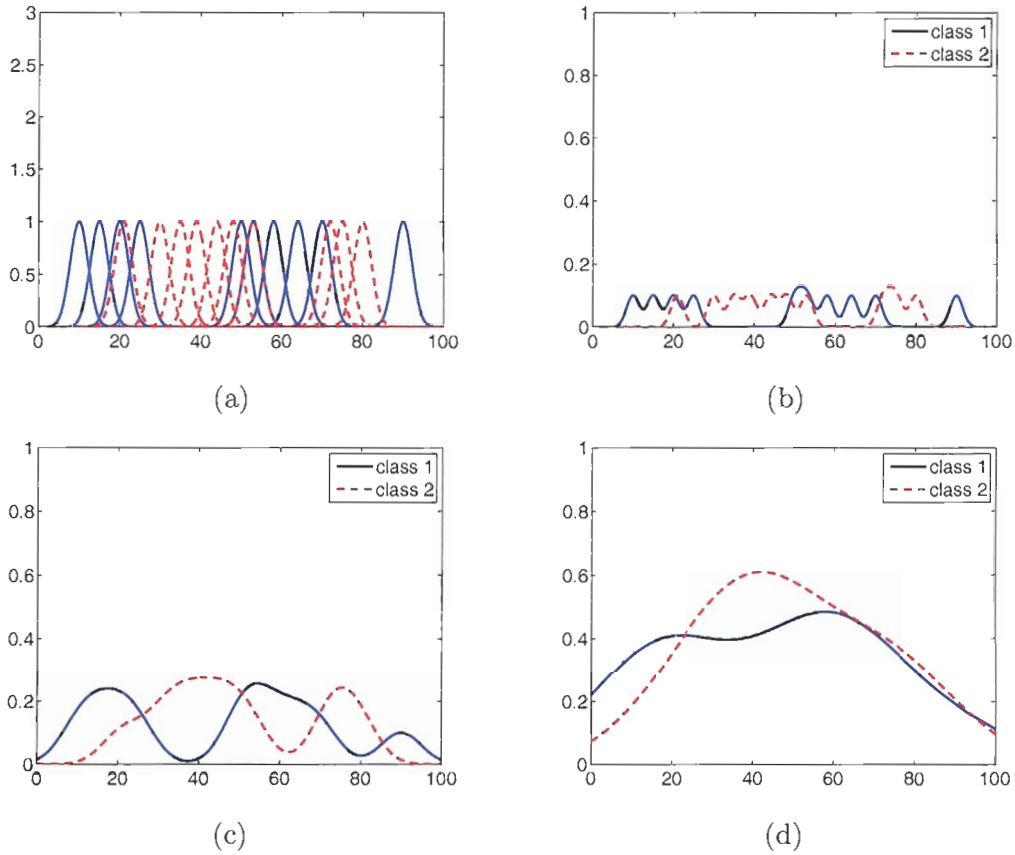


Figure 3.3: (a) Kernels of some sample points of two classes in 1D (b) Undersmoothed Kernel density estimation of the two classes (bandwidth = 5). (c) A good KDE (bandwidth = 50) (d) Oversmoothed KDE (bandwidth = 500)

data. However, the geometric blur descriptors are quite smooth, and the space of naturally occurring geometric blur descriptors does not cover the entire high dimensional space.

3.2 Experimental Results

We evaluate our method by experimenting on two different datasets. One is the MIT pedestrian dataset [34], a popular dataset for evaluation of pedestrian detection systems. Because of the near-perfect results on this dataset, we also did some experiments on a more challenging dataset, the INRIA dataset [6]. In this dataset, people are mostly in

Figure 3.4: Examples of pedestrian from INRIA (top row) and MIT (bottom row) datasets. The INRIA dataset is designed to be a hard dataset, by adding examples with more pose and background variety, and sometimes partial occlusions.

standing position, but they cover more diverse body poses and a much varying background in comparison to MIT set. Figure 3.4 show some examples of these two datasets.

In the pedestrian image sets, for the sake of consistency, we crop both datasets' images into 60×100 images of pedestrians. Our positive MIT set consists of 470 training and 450 test images. The positive INRIA set consists of 511 training and 503 test person images. For both datasets we added the right-left flips of both training and test datasets to each set. This will both increase the number of pedestrians in our dataset, and will prevent the detector from being biased in detecting people facing one direction better than the other.

In all of the experiments, for the negative training and test images, we used the person-free images of INRIA dataset. We sampled 11000, 60×100 windows from them, using 2000 for training, 7000 for testing, and 2000 for cross-validation procedure which we will describe later.

For the quantification of the results we plot miss rate versus False Positive Per Window tested (FPPW) curves on a log-log scale, consistent with Dalal and Triggs curves [6]. Miss rate is defined by $\frac{\#FN}{\#P}$ and FPPW is $\frac{\#FP}{\#N}$ where $\#FN$ and $\#FP$ are the number of false negatives and false positives respectively and $\#P$ and $\#N$ are the number of positive and negative examples used in the testing phase.

In the following parts of this section, we focus on tuning our detector parameters. First

we try to find the best geometric blur sampling method among some candidates. Then, a cross-validation technique is applied to find the best band-width for the Gaussian kernel, to be used in the kernel density estimation.

3.2.1 Different Sampling Designs

One of the factors that affects the characteristics of the geometric blur descriptors is the number of samples and their sampling locations from the blur. In our experiments, we tested three different sampling designs. During the experiments of different geometric blur designs, Gaussian kernel's bandwidth is the other parameter involved which is not tuned yet. To make the experiments comparable, we use the variance of L2-distances of training images' feature vectors of each sampling design as the Gaussian kernel bandwidth of that design.

The two parameters that we changed in the three experiments are: Number of radial samples, and number of angular samples per radius. To have the same area of influence for different designs, we fixed the closest and the furthest radii distances in them.

The first design has 48 samples per signal, which makes the feature a $192(= 4 \times 48)$ dimensional vector. There are 12 samples at each of 4 sampling distances in this design (GBD 4×12). The second design has 8 samples over 4 distances (GBD 4×8), and the third one, 8 samples over 3 distances (GBD 3×8).

The accuracy results for these designs are shown in Figure 3.5 and Figure 3.6. These figures show the performance of different designs on the two test sets INRIA and MIT. The performance of all detectors are very close and there is no obvious advantage on choosing one in either of the datasets. The only small difference, is that when the dimensionality of feature vectors decreases there is a slight gain in the performance. This can be because of the fact that when we are computing the kernel density estimates, our fixed number of training images can represent and fill the space better when we have fewer dimensions. The other advantage of using a descriptor with fewer dimensions is the speed performance. The smaller the dimensionality of our geometric blur descriptor, the less computation time we need, both for training and testing phases. Therefore we use the 3×8 design, the smallest descriptor, for the other experiments.

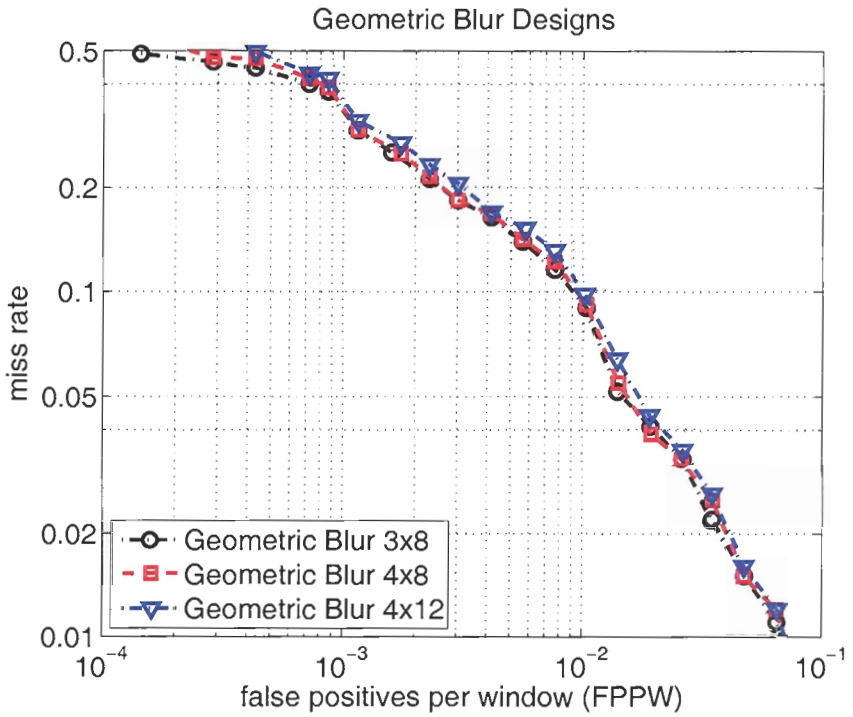


Figure 3.5: Performance of different geometric blur designs on INRIA set

3.2.2 Bandwidth Tuning

Another parameter that can influence the performance of our detector drastically, is the bandwidth that we use for the Gaussian kernel used in the kernel density estimator. To find the best bandwidth we use cross-validation. For the negative validation set, we have a separate set from our training and testing set as described in the beginning of this section. For the positive set we break our positive training set into 4 separate sets, iterating through using three as training and one as validation set (4-fold cross-validation).

The results for different bandwidth choices are shown in Figure 3.7. As we narrow the bandwidth, we get better results, up to some point where the performance does not change by narrowing the kernel. We choose that particular bandwidth as our tuned Gaussian kernel bandwidth, and we use it throughout our final experiments on the test set.

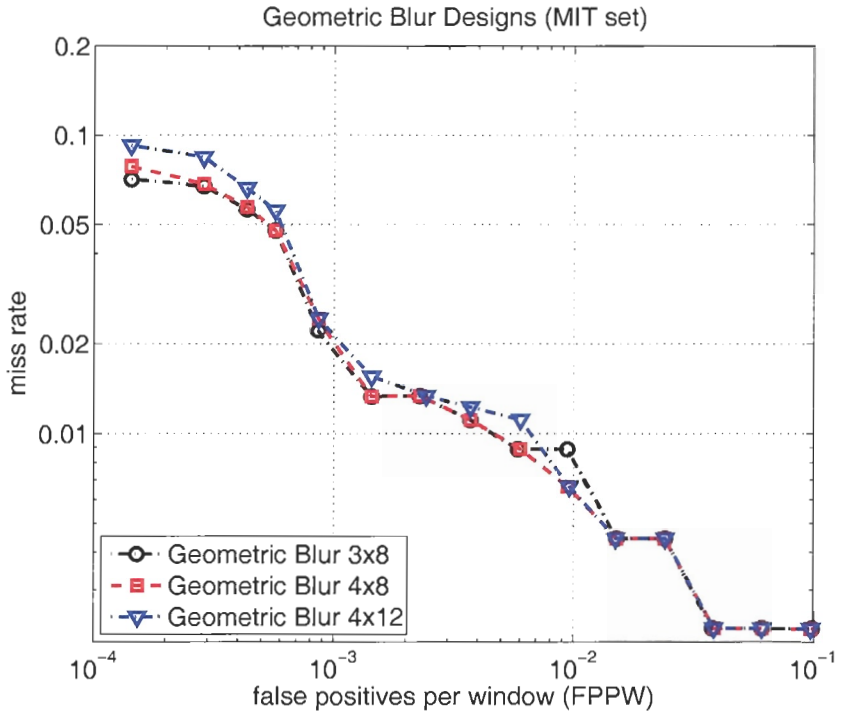


Figure 3.6: Performance of different geometric blur designs on MIT set

3.2.3 Evaluation

We have evaluated our tuned detector on both INRIA and MIT datasets. To have a base for comparison, we used the binary code provided by Dalal [6] which is one of the state of the art pedestrian detector systems. Due to some incompatibility problems we could not run their binaries on the MIT set, therefore we provide their original published figure [6] for comparison.

Figure 3.8 and Figure 3.9 show the quantitative comparisons of our detector and the base detector. On the MIT set our results are near perfect but slightly worse than HoG detectors. On INRIA set, which is a much harder set than MIT, our results are not comparable to that of Dalal and Triggs'. There are two main reasons that has led to these relatively poor results. One is the use of geometric blur features which are not appropriate to model the common shapes of each object class, and the other is the inability of kernel density estimation methods in higher dimensional data, such as our geometric blur features. We

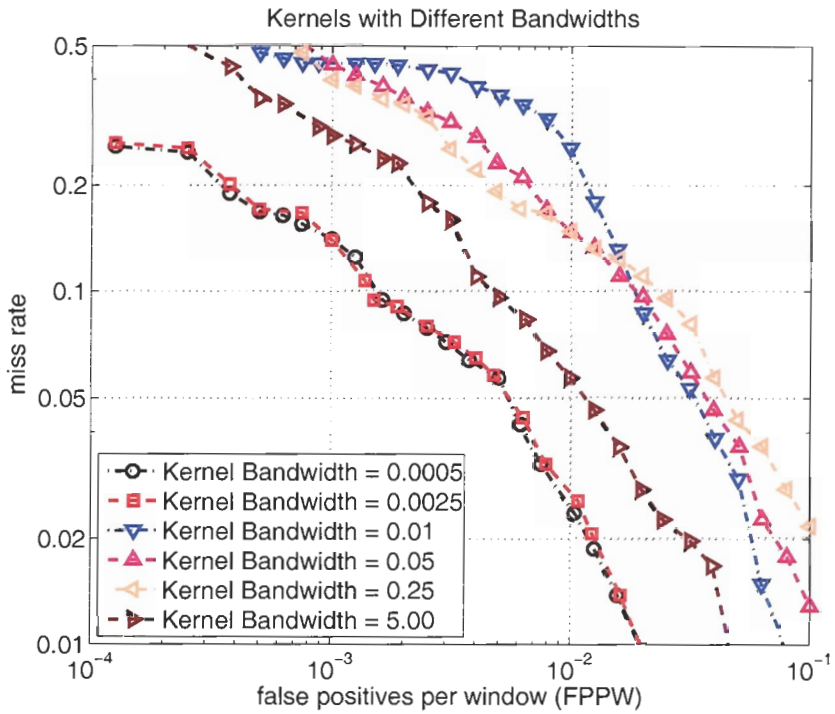


Figure 3.7: Results of cross validation test on different bandwidth parameters for the Gaussian kernel.

will further analyze these reasons in section 3.3 and also in chapter 5.

In addition to these quantitative comparisons, Figure 3.10 shows examples of per-pixel likelihood ratio results for images from our test set. In contrast with methods (e.g.[6]) which build a single feature vector for the entire window, our method obtains per-pixel likelihood ratios. In cases of partial occlusion, likelihood ratios reflect the negative influence of the occluder. We believe that this per-pixel likelihood ratio is useful for situations involving multiple pedestrians or other occlusions.

Figure 3.11 gives a qualitative assessment of the mistakes made by our detector, showing the worst false positives (most person-like backgrounds) and worst false negatives (most background-like pedestrians) from our detector's point of view.

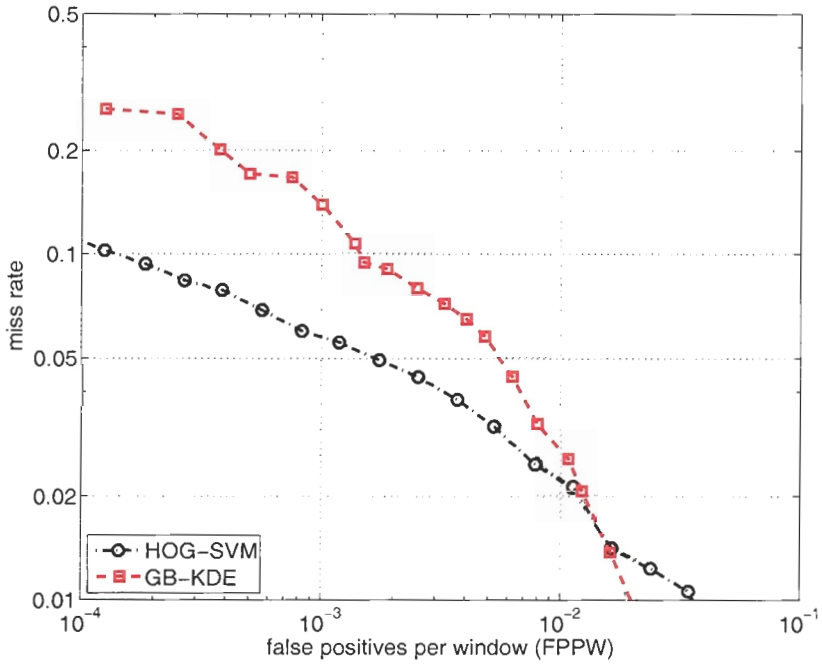
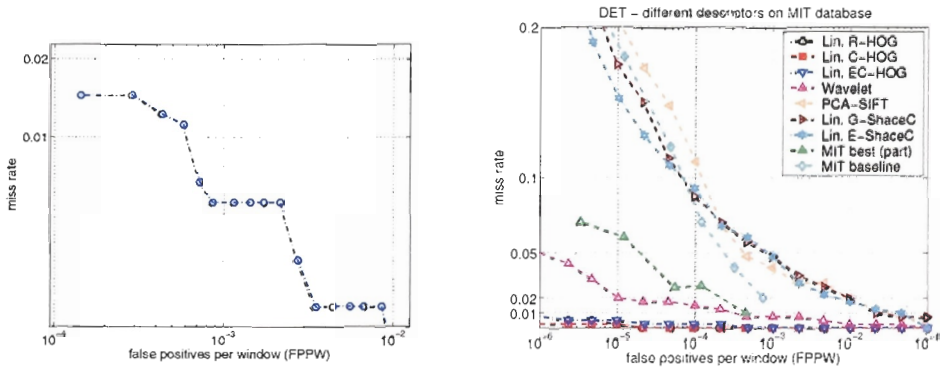


Figure 3.8: Final results of our GB-KDE based detector on the INRIA dataset compared with the results of HOG-SVM detector of Dalal and Triggs.

3.3 Analysis

This pedestrian detection approach, like every other approach, has its own advantages and disadvantages. One of the main advantages is its ability to compute the likelihood of personness of each pixel separately. This can help especially when we need to reason about occluded pedestrians. Figure 3.10 shows some examples of semi-occluded people and their likelihood response masks. These likelihoods can also be used for segmentation of the pedestrians in a bottom-up manner. One can over-segment the image using any of the common image segmentation algorithms (such as Felzenszwalb and Huttenlocher's color based segmenter [8] and Ren and Malik's classification based segmenter [35]), and use the per-pixel pedestrian likelihoods to reason about the class of each segment. This way the overall segment of the object can be extracted from its background in the detection window. If solved as a joint probability problem over all the segments together, occlusions caused by other pedestrians



(a) (b)

Figure 3.9: (a) Final results of geometric blur based detector on the MIT dataset. (b) Dalal and Triggs' [6] tests using different approaches on the MIT dataset. (©2005 IEEE, by permission)

in the crowded scenes can be resolved as well.

The main disadvantage of this method, is its model complexity and high amount of computation and memory that it needs. The model complexity is due to the high dimensionality of geometric blur descriptors. We need a huge number of training examples to estimate class distributions in a high dimensional (e.g. 192) feature space. On the other hand the more training examples added, the slower the estimation of a test example in that space would become. The need for high amount of computation and memory arises as a result of per pixel kernels that we need to keep track of for each training image. To be precise, we should store a high dimensional feature vector at each pixel for each training image. At the test time, for each pixel, we need to estimate the distance of the feature vector related to that pixel with both positive and negative kernels. There are some efficient ways of kernel density estimation [52], but because of the relatively poor results of this detector we did not explore using them to make the computations faster.

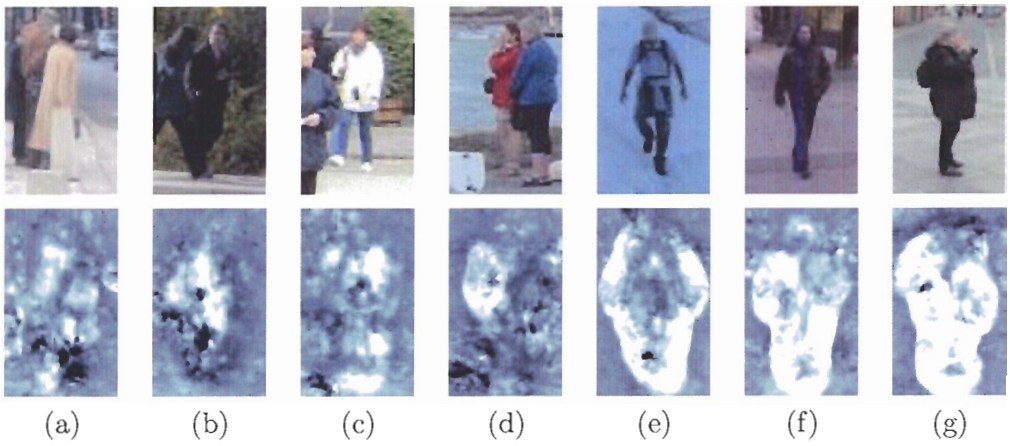


Figure 3.10: Examples of per-pixel likelihood ratios. Top row show input images, likelihood ratios below. Bright values indicate high likelihood of pedestrian, dark values indicate low values. (a-d) show instances of partial occlusion. Note that in areas of occlusion, such as the bag in (a), or leg of other person in (b), low likelihood ratios exist. (e-g) show examples of likelihood ratios for un-occluded people for comparison.

(a) (b)

Figure 3.11: Examples of errors made by our detector. (a) False negatives with lowest likelihood ratio. (b) False positives with highest likelihood ratio. False negatives typically consist of people in atypical poses, or substantial clutter. False positives usually contain strong vertical edges mimicking the torso and leg boundaries.

Chapter 4

Detecting Pedestrians Using Boosted Features

In the Chapter 3, our first approach for pedestrian detection was covered. The main problem with that approach is how complex it is. This includes computational complexity, that makes the detector slow in terms of its running time, and difficulty in modelling the distribution over GBDs. Together these lead to poor performance of the detector. In this chapter we will propose a new approach for pedestrian detection which is fast enough to be run in real time and its performance is better that the state of the art pedestrian detector [6]. This method is based on creating a specific feature set by training a set of local discriminative features using the training data. These trained features will capture more useful information, discriminative between our two object classes, than any fixed set of features. This method uses AdaBoost as its base, and rely on it for both feature creation and classification phases. For an illustrated overview of the approach refer to Figure 1.5 and Figure 1.6 in the Chapter 1.

Like the previous detector that we described (in chapter 3), this detector uses a scan-based method to search for the pedestrians in the image. We train a classifier that given a fixed size window (i.e. 64×128 in our experiments) it can classify the window as a pedestrian or a non-pedestrian. To search a complete image, we scan the image exhaustively in different locations and scales by putting the detection window in every location and every scale and classifying that window using our trained classifier. As we will discuss later, because of the structure of our detector, there is a very efficient way of doing this thorough search that

makes it possible to run this detector in real-time.

In the experiments section, we will test our detector on two different datasets and show the quantitative results. We will also discuss all the employed parameters and factors and further explain and test those parameters that affect the performance of the detector.

4.1 Local Discriminative Features via AdaBoost

A major drawback in most object detection algorithms is the fixed set of feature descriptors that they use. The problem with defining features before training the classifier is that, there could be some discriminative information that is missed by those features. By training the feature set, instead of selecting it, we can make sure that we are using all the useful information of our two object classes in the classification phase.

Viola et al. [47, 48], use AdaBoost for face and pedestrian classification, using Haar-like wavelet features. Although their features are low-level, they are treating them as their final set of features and build their final classifier directly from them. As the features are very low level and sparse, the final classifier will not capture the repetitive information of local neighborhoods completely. On the other hand, Wu and Nevatia [51] use AdaBoost with a set of hard coded mid-level features as its weak classifiers, called edgelets. These edgelets are a set of pre-defined patterns of edges in different locations. AdaBoost will make the final pedestrian classifier by using a subset of these features. The problem with this approach is that there is no guarantee that the edgelets can capture all the useful available information for classification because of their fixed nature.

In our approach, we combine the flexibility of small low-level features and the local informativeness of mid-level features by training a set of discriminative mid-level features from the low-level features. This way, we have the advantages of both approaches without suffering from their shortcomings. In our algorithm, we make a mid-level feature set by using the low-level features extracted from the training set images. Later, this feature set is used to train the final classifier. The algorithm that we use for both feature creation, and also the final classifier training, is AdaBoost.

Dalal and Triggs [6] who use HoG descriptors and SVM for their state of the art pedestrian classifier, also have a set of fixed features. Their pre-defined HoG features, which they tune all of their parameters by testing, are designed in a way to capture all the possible edge information in the image to feed it into SVM. But still because of this fixed nature

they cannot guarantee that all the useful edge information is captured in a useful way for classification. In other words, the amount of non useful information could be too much that can affect the final results.

The training part of our approach is consisted of three layers. An overview of these layers is as follows:

1. **Low-level Features as Weak Classifiers:** The input to this layer is raw training images. We extract the gradient responses of each image in four different directions, and compute the average of these responses around each pixel, and use them as our low level features. For any feature from this feature set, identified by its location and gradient direction, we can use it as a weak classifier of the two classes by setting a threshold in between its responses. Of course this classifier will not be a good one (which is apparent from its name: weak classifier), and we don't expect it to be. These weak classifiers will be used to make more sophisticated features.

2. **Mid-level Features as Semi-strong Classifiers:** For some small windows inside the detection window, we use AdaBoost to select a subset of the weak classifiers inside each window to construct better classifiers. By only using the features inside each window, we force the AdaBoost to extract as much information as possible at local neighborhoods of the image. This process will provide us several stronger local classifiers that are highly discriminative regarding our object classes. Each can be considered as a combination of edges with different orientations at different locations.

3. **Final Classifier:** The mid-level feature descriptors from previous layer can only act in a local neighborhood in the image and therefore their overall classification power is still far below an accepted point. But by merging them together we can combine the information from different parts of the image. In order to archive this goal, we use AdaBoost for the second time to combine those information and train our final classifier. This time we use our semi-strong local classifiers as its input, and the algorithm will again choose the best subset among them that can separate the two classes as much as possible.

4.1.1 An introduction to AdaBoost

Adaboost was first introduced by Freund and Schapire [10, 9], and after that it has been widely used in many of the machine learning applications, and specifically to our interest, in computer vision. The main idea behind AdaBoost (and other boosting algorithms) is to make a *strong* learning classifier by using some *weak* learning classifiers that perform just slightly better than random $P[h(x_i) = y_i] = 0.5 + \epsilon$, where $h(x)$ is a weak classifier x_i is a feature vector and y_i is its true class label.

AdaBoost's Pseudocode is given in Table 4.1. The algorithm takes as input a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where each x_i is one of the training examples' feature vector, from a high dimensional space $\mathcal{X}$. Each label y_i shows the class label for x_i and belongs to label set $\mathcal{Y} = 0, 1$ (0 for negative examples and 1 for positive examples). In the algorithm there is a weight w_i associated with each example-label tuple. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set.

For us, a *weak learner* is a function that finds the best classification of the two classes, recognized by their labels y_i, using only the j'th dimension of feature vectors x_i. We call this feature f_j. The weak learner should consider the weights associated with examples when trying to find the best classification.

Weak learner will return a *weak classifier* $h_j : \mathcal{X} \longrightarrow \mathcal{Y}$, which is consisted of a feature f_j, a threshold θ_j and a sign d_j in the inequality:

$$h_j(x) = \begin{cases} 1 & \text{if } d_j f_j < d_j \theta_j \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Threshold θ_j, is the threshold between the responses of dimension j of feature vectors. This threshold classifies the two classes by assigning each side of its value to one of the classes. The direction of this assignment is defined by $d_j \in \{-1, 1\}$. Note that θ_j can be chosen on any of the feature vector's dimensions with any value in between its responses. This is the weak learners' job to choose the $h_j(x)$ in a way that it has the minimum classification error. This error is the sum of weights of misclassified samples and is calculated by:

$$\epsilon_j = \sum_i w_i |h_j(x_i) - y_i| \tag{4.2}$$

AdaBoost calls the weak learner algorithm repeatedly in a series of rounds. One goal of

- For training examples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ with feature set x_i and labels $y_i \in 0, 1$ for non-object and object classes respectively.
- $w_{1,i} = 1/n$ for all i.
- For $t = 1, 2, \ldots, T$:
  1. For each feature j, train a weak classifier $h_j(x)$. The output of $h_j(x)$ is in the format of our labels y_i.
  2. For each hypothesis classifier h_j, calculate the error:

     $\epsilon_t = \sum_i w_i |h_j(x_i) - y_i|$

  3. Select the classifier h_t with the smallest classification error ϵ_t.
  4. Calculate new weights:

     $w_{t+1,i} = w_{t,i} e^{(-\alpha_t e_i)}$

     where $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$ and $e_i = 1$ if x_i is classified correctly and $e_i = -1$ otherwise.
  5. Normalize the weights:

     $w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}}$

- The final classifier is:

$$H(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Table 4.1: AdaBoost

the algorithm is to maintain the distribution of weights w_i over the training set. On each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set. In every iteration, the best $h_j(x)$ is added to the set of selected weak classifiers, and the weights are recalculated according to the classification results of the selected classifier:

$$w_{t+1,i} = w_{t,i} e^{(-\alpha_t |h_{t,j}(x_i) - y_i|)} \tag{4.3}$$

Where $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_j}{\epsilon_j})$, and $h_{t,j}(x)$ is the best weak classifier selected in iteration t.

The final classifier $H(x)$ is a weighted majority vote of the weak hypothesis selected in the over all T iterations:

$$H(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

4.1.2 Low-Level Features

As discussed in chapter 2, most pedestrian detection approaches capture the edge information as their lowest level features, each in their own way. Some capture it by computing image gradients [4, 6], some by computing wavelet coefficients [42], and some by applying simple rectangular filters [47] or more sophisticated features like edgelets [51]. We also use edge responses as our lowest level features. We capture this information by computing the gradient of the image intensity. We use gradient information, that are captured in four different directions, as our atomic features. The derivative mask that we use is a simple [-1, 0, 1] filter.

To reduce the influence of small spatial shifts in the detection window, we locally average the edge information in each direction. More precisely, we compute the average of gradient responses around each pixel by convolving the edge responses, for each of the directions, by a small box filter.

$$S_d(x) = |I(x) * G_d| * B \tag{4.5}$$

Where, $*$ denotes convolution, $I(x)$ is the intensity image, G_d is the gradient kernel that we use to get derivatives in direction $d \in \mathcal{D}$ (e.g. $[-1, 0, 1]$ or $[-1, 0, 1]^T$) , B is a 2-D box filter (e.g. a 5×5 matrix with all the elements $\frac{1}{25}$) used for average computation, and $S_d(x)$

is our final signal image that captures the amount of edge at every pixel in direction d. $\mathcal{D}$ is the set of possible directions that we are computing the gradients in. In our experiments we use four directions; $\mathcal{D} = \{0°, 45°, 90°, 135°\}$. We use the absolute value of the gradients (in equation 4.5) instead of their real value to compute the edge values. This is because the information we need is the orientation of the edge not its direction (a person with white clothes over a dark background should be considered the same as a person with black clothes over a light background).

At this point, for every pixel x in the image, we have the local average of edge responses in different directions d. These locally smoothed gradient responses, are considered as our low level features, and as we will see shortly, we use them as the weak classifiers of AdaBoost algorithm to build our mid-level feature set.

The information captured about the classes by each of the low level features is very little. If used as a classifier, each of these low level features $S_d(x)$, can only separate our two classes (pedestrian and background) slightly better than random classification. To make our features more meaningful and informative, we would combine them to create some local mid-level features, using AdaBoost.

4.1.3 Mid-Level Features

Suppose we have k small windows $w_i \in \mathcal{W}$, $i = 1, \ldots, k$ inside our detection window, each containing a set of neighboring pixels (the window selection process will be explained in detail in section 4.2). We build one mid-level feature for all of these windows w_i. Figure 4.1 illustrates one of these windows over some samples of the two classes and shows the actual mid-level features trained for that window.

To make mid-level local classifiers (features), for every window w_i, we collect all the features that are inside that window $\{f_d(x) : x \in w_i, d \in \mathcal{D}\}$ and use them as potential weak classifiers of an AdaBoost run. The number of low-level features n_i for each w_i is the number of pixels inside that window times the number of gradient directions $|\mathcal{D}|$.

In each iteration t of AdaBoost training process, one of the features $f_t(x) \in \{f_d(x)\}$ is chosen as the feature of the weak classifier $h_t(x)$ to be added to the final classifier. This weak classifier is in the form of:

$$h_t(x) = \begin{cases} 1 & \text{if } d_t f_t(x) < d_t \theta_t \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

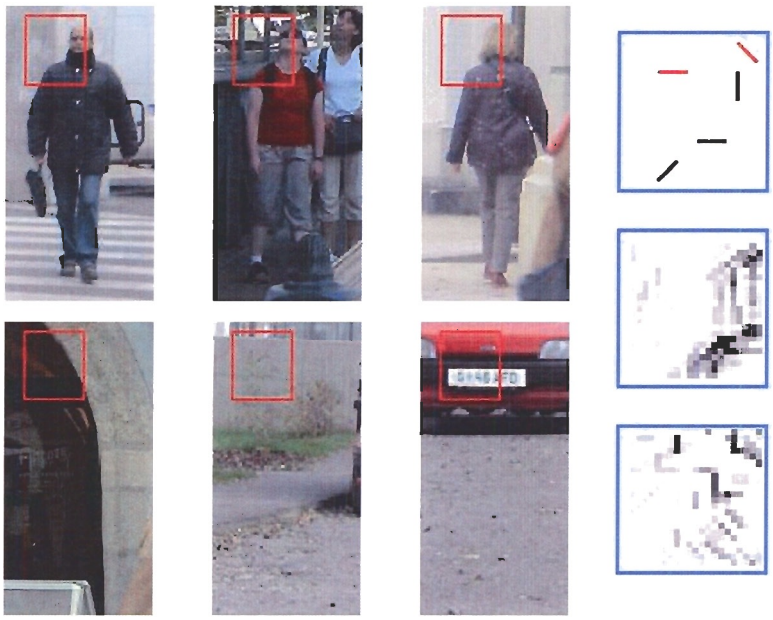


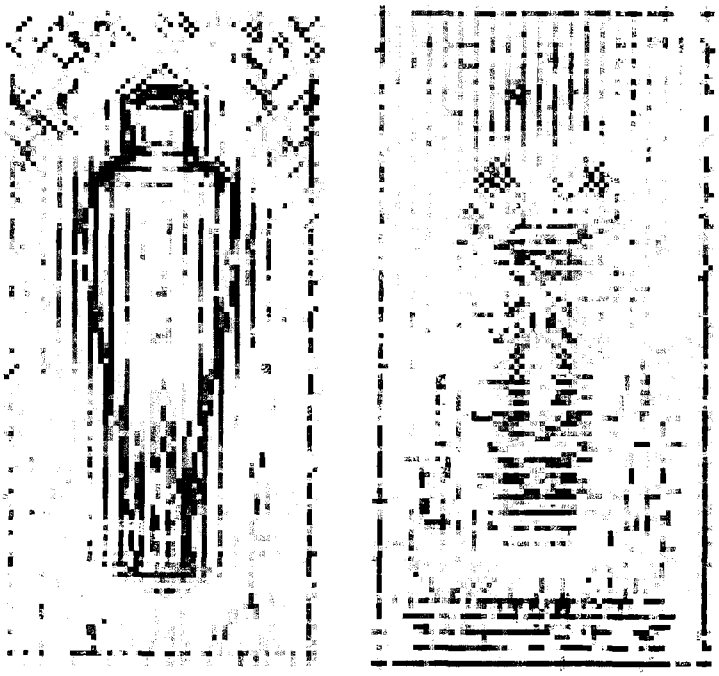
Figure 4.1: Illustration of a hypothetical mid-level feature and the actual learned mid-level features for that window. The two different feature windows separate the different features selected from each of the object classes.

Where θ_t is the threshold of the classifier and $d_t \in \{-1, 1\}$ defines the direction of the inequality. θ_t can have any value in the response range of $f_t(x)$, but we discretize it to 100 levels inside that range and use them as potential thresholds instead of searching the whole continuous range. This will reduce the running time of the training phase, and also reduces the amount of memory needed for both training and testing phases by letting us store an integer value instead of a floating point for a big number of features.

After all the T iterations of the algorithm, we get the final classifier $H_i(x)$ for window w_i. This classifier is in the form of:

$$H_i(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t^i h_t^i(x) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

Where α_t^i is the selected weight for classifier $h_t^i(x)$. For more detail about AdaBoost refer to Section 4.1.1 and Table 4.1. We train a similar classifier for every window w_i. These local classifiers are semi-strong classifiers for pedestrian classification task. They are not as



(a) (b)

Figure 4.2: Sum of weights of the low-level edge features selected for all the mid-level features across the detection window. (a) Features that belong to pedestrian class, (b) features that belong to non-pedestrian class.

weak as the low-level feature classifiers, but not strong enough to be used for pedestrian detection task separately. In the next section we show how to use these classifiers all together to classify pedestrians and non-pedestrians. Figure 4.2 shows the sum of all the low-level features selected during training of mid-level classifiers. The selected low-level features are separated in two groups according to their selected direction by AdaBoost d_t. This direction shows whether the feature discriminates pedestrian class or non-pedestrian class from the other.

Viola and Jones [47] use a fixed set of features to train their classifier using the AdaBoost. They introduce a huge set of features, assuming that they contain all the needed information for the classification task. We eliminate this assumption by training our locally discriminative features, using the training examples. By training these features over the lowest level edge features of the image, we guarantee that we have captured the necessary

information from every local neighborhood of the images. This way, we don't need to change the design of the features we want to use for different object detection tasks, we only have to retrain the mid-level features to adapt them with the new types of object and let them form in a way so they capture all the information that we need.

4.1.4 Final Classifier

Our final goal is to build a strong classifier that can classify our two classes of objects. To this end, we have created some local classifiers $H_i(x) : \mathcal{X} \longrightarrow \mathcal{Y}$. The domain of each classifier is the features inside their effective window w_i and their range is the label set $\mathcal{Y}$. But if we take a second look at the classifier form (in equation 4.7), it can be seen that the weighted sum of weak classifiers is a continuous value that its sign is determining the estimated class. Let us call this sum $s_i(x) = \sum_{t=1}^{T} \alpha_t^i h_t^i(x)$. A good characteristic about AdaBoost classifiers is that this s_i has more information than only specifying the class by its sign. The further away the value of s_i from the zero, the more certain we are about the classification. Therefore this value can be used as a confidence measure of the classification. Knowing these facts we can define our mid-level features as:

$$s_i(x) = \sum_{t=1}^{T} \alpha_t^i h_t^i(x) \tag{4.8}$$

Where $i \in \{1, 2, \ldots, k\}$ corresponding to one of the windows $w_i \in \mathcal{W}$, on which we ran AdaBoost, and $h_t^i(x)$ and α_t^i are the parameters computed for the classifier $H_i(x)$, associated with that window. Note that each $s_i(x)$ is not a classifier, instead it is a local feature that is trained to distinguish between the two classes.

Now that we have defined our new features $s_i(x)$, we can use them inside the AdaBoost to create a final strong classifier from them. Details of creating weak classifiers and re-weighting the samples is the same as previous layer. The final classifier is in the form of:

$$C(s) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t g_t(s) \geq \lambda \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Where $s \in \{s_1(x), s_2(x), \ldots, s_k(x)\}$ is a variable corresponding to one of the mid-level features, and λ is the final classifier's threshold that is originally zero in AdaBoost but we can change its value to get different detection and false positive rates. Note that this time

(a) (b)

Figure 4.3: Illustration of low level features inside the final classifier. (a) pedestrian class features, (b) non-pedestrian class features.

the weak classifiers $g_t(s)$ are applied in the new feature domain s instead of the low level features x, and therefore the final classifier looks like a combination of some weighted mid-level features. Like before, each $g_t(s)$ is consisted of one feature $s_i(x)$, a threshold θ_t, and a sign d_t.

When the final classifier is trained, one can illustrate all the low level features that are inside the selected mid-level features. Depending on the direction d_t, assigned to each of the features (classifiers), they classify one of the classes from the other: pedestrian from non–pedestrian or non–pedestrian from pedestrian. An illustration of both sets of features are shown in figure 4.3. Note how the feature set that classify pedestrian class from non–pedestrians contains the features of a person's silhouette.

One might argue that using all the low-level features together in one AdaBoost run, and training a classifier in one step could result in equivalent performance. There are two arguments against that approach. First is that inputing all of the low-level features in to

AdaBoost will make the training time very slow, if not intractable (we attempted such an experiment, and were unable to complete it). That is because we are introducing many more possible weak classifier that AdaBoost should choose from in each iteration. The second, and the more important reason is the laziness of AdaBoost (like many other machine learning algorithms). AdaBoost always extracts the minimum amount of information, from the training data, needed to predict the classes. This can lead to misclassification in the test data because of the difference in the selected features. We overcome this problem by introducing mid-level features. We extract more information at local areas of the image, before focusing on the final classification. Of course this over-harvested information contains more noisy data, but in overall that extra noisy information will enable us to handle test images better. Our approach is related to the FeatureBoost algorithm of O'Sullivan et. al [32], which deemphasizes (or removes) individual features in successive AdaBoost-like iterations. While the motives are related, our work is different in the execution. We explicitly learn features in sub-windows of the detection window, and then combine them in a subsequent classification stage.

4.2 Experimental Results

In this section we will describe the data sets we used to test our detector and the parameters we used for the implementation. We will also study the performance of our detector with different setups, and compare it to other state of the art approaches.

4.2.1 Data Sets

To evaluate our detector we only use the INRIA dataset. We do not use the MIT set for our experiments because of the lack of comparison information with other approaches due to simplicity of that set. Instead the more challenging INRIA dataset [6] gives us the opportunity to show our detectors capabilities. Our detection window size is 64×128 pixels, and we use cropped images to this size for both the training and testing purposes.

The training set of INRIA dataset is consisted of 1208 cropped pedestrian images with their left-right flips (2416 person images in total), and 1218 person free images. To train our detector, we used all the 2416 person images as our positive set, and we cropped samples from the person-free images to build our negative set.

The test set is consisted of 566 initial pedestrian images, that by adding their left-right

reflections we will have a total of 1132 person images as the positive set. For the negative set, there are 453 person-free images that we will search them exhaustively for the testing and evaluation.

To search an image exhaustively, we will crop sample windows of 64×128 pixels from the different scales of the image systematically. We build the multi-scale pyramid of the original image by resizing it, with a scale ratio of 0.8, until the width of the image is less than 64 or its height is less than 128 pixels. In each scaled image inside the pyramid, we crop images of size 64×128 with strides of 8 pixels between them. This exhaustively sampling method is used in testing phase as well as the training phase to gather harder examples for bootstrapping. For comparison compatibility with Dalal and Triggs' work, this sampling method is exactly the same as their dense sampling method.

4.2.2 Parameters and Settings

We have different sets of parameters for our experimental results. Some of the parameters are fixed during different tests. You can find a list of fixed parameters in Table 4.2. Most of these fixed parameters are related to the low-level feature extraction part.

Fixed Parameters	
Detection window width	64 pixels
Detection window height	128 pixels
Number of gradient orientations	4 $(0°, 45°, 90°, 135°)$
Gradient filter	[-1, 0, 1]
Box filter size for computing the average	5 pixels (5×5 box filter)
T (number of mid-level features in the final classifier)	1000

Table 4.2: Fixed Parameters

We further explore other parameters that are specific to our proposed approach. One of the parameters that affect the results of the detector drastically is the mid-level window selection method. As we described in section 4.1.3, there is a window set $\mathcal{W}$ that defines the area of influence of each of the mid-level features. Our experiments show that restricting $\mathcal{W}$ to just one window size decreases the power of the detector. The reason is that the information captured from a tiny window only reflects the classification capability of a small set of features which is not very high. On the other hand, a big window has many low

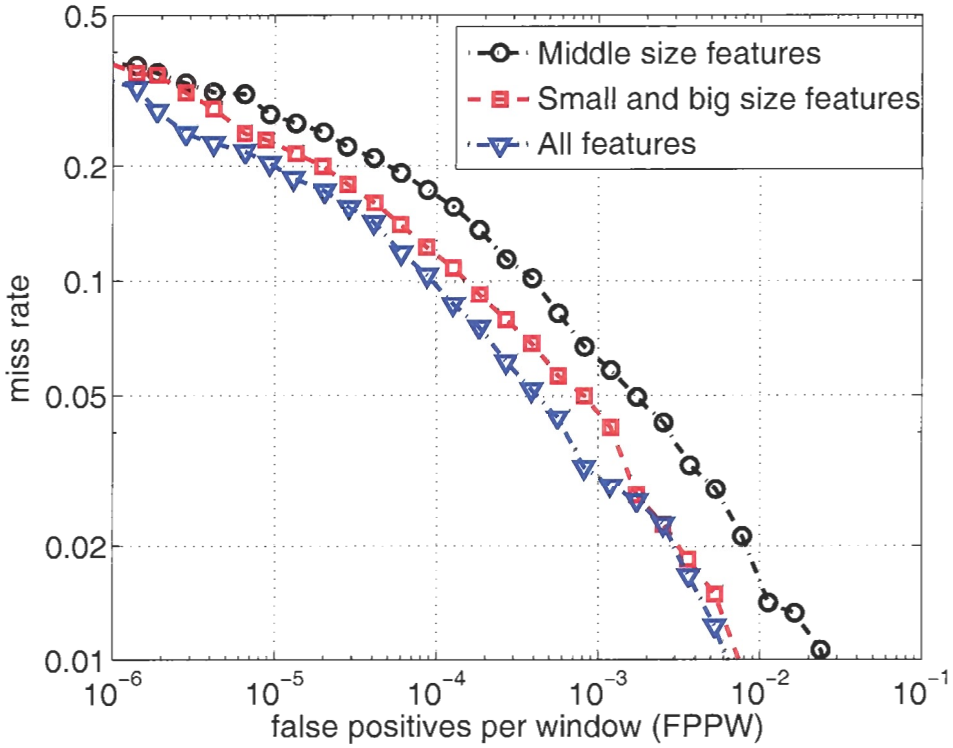


Figure 4.4: Performance of the detector using different sets of mid-level feature settings. Having small, middle, and big features at the same time performs the best.

level features inside, that the correlation between neighboring features inside it will not be reflected in the mid-level features extracted from it. We experimented our detector on three different window sets $\mathcal{W}_i$. The first set $\mathcal{W}_1$ is only consisted of some mid-size windows, the second set $\mathcal{W}_2$ contains small-size and large-size windows, and the last set $\mathcal{W}_3$ is the union of the other two sets: $\mathcal{W}_3 = \mathcal{W}_1 \cup \mathcal{W}_2$. The detailed parameters and settings used for each window size is shown in Table 4.3. In Figure 4.4 the classification results of using each of the three window sets $\mathcal{W}_i$ is shown. Note that the results of the set that contains all the three sizes of windows has the best results among the three. This shows that each features with a particular size can capture some information from the image which the other feature sizes are unable of. This observation is another support for our argument about the advantage of using mid-level features in section 4.1.4.

Different mid-level feature settings	
Small size features	
Window size	5×5 pixels
Number of low-level weak classifiers extracted	10
Stride between windows	5 pixels
Medium size features	
Feature window size	10×10 pixels
Number of low-level weak classifiers extracted	30
Stride between windows	4 pixels
Large size features	
Feature window size	15×15 pixels
Number of low-level weak classifiers extracted	35
Stride between windows	4 pixels

Table 4.3: Different mid-level feature parameters

Another experiment shows that more levels of window sizes does not improve the results of the final detector. Figure 4.5, shows the performance curves of the 3-level window size detector, and a 5-level window size detector. Surprisingly the 3-level detector performs slightly better than the other one. We wanted these features to contain new information in addition to our other 3 feature sizes to increase the performance. But as the performance does not increase it shows that these features contain redundant information.

Assigning initial weights to training examples in AdaBoost is another setting that can be modified. One can give more weights to positive examples than the negative ones, as opposed to giving equal weights to all the examples. Both methods have been used in different approaches that use AdaBoost. It can be argued that because of the domination of negative examples in the training set, by giving more weights to positive examples we can make the AdaBoost to focus on positive examples as much as negative ones. We conducted an experiment using both initial weight settings to observe its effect on the performance of the final detector. Our experiments showed that these settings will result in almost the same performance at the end, and there is no significant advantage on choosing any of the settings. This can be explained by the fading of the initial weights after a few iterations in any AdaBoost run. In each iteration of AdaBoost, all the samples are reweighted according to the classification results of the chosen weak classifier at that iteration. Therefore, as long as the initial weights are not very biased on some of the examples, after a few iterations higher

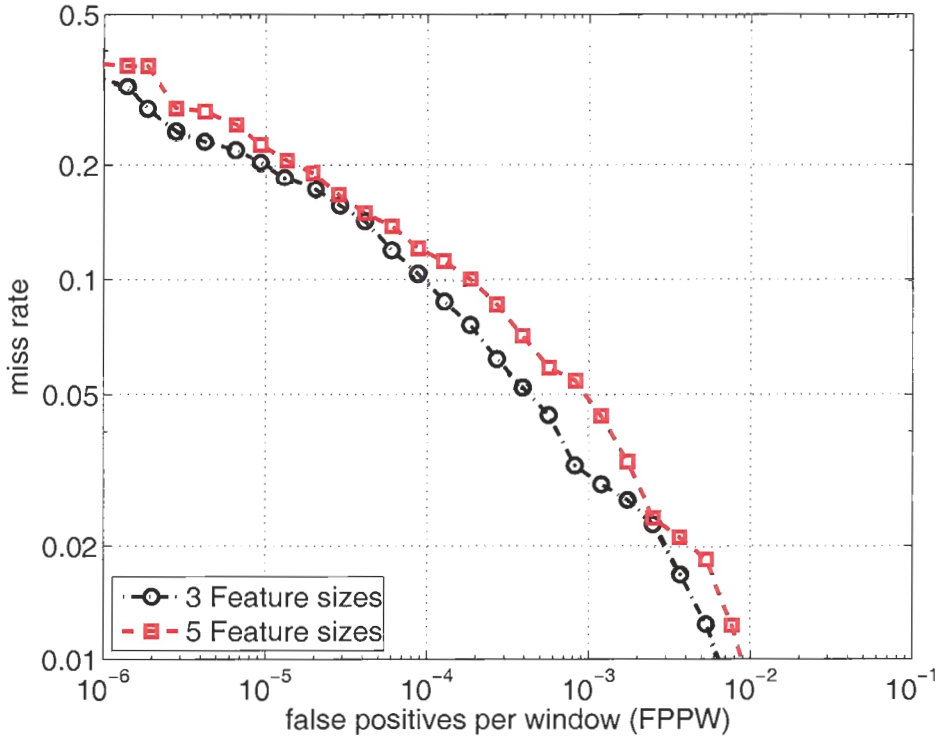


Figure 4.5: Comparison results of adding more levels of mid-level feature sizes to the detector.

weights will be assigned to harder examples which cannot be classified easily. Performance results of this experiment are shown in Figure 4.6.

Another parameter in the AdaBoost algorithm is T which is the number of weak classifiers that the final classifier contains. We investigated the influence of this parameter on the detector's performance. Figure 4.7 shows the miss rate of the detector for three fixed FPPW values for different number of mid-level features (T) selected in the final classifier. For all the three FPPW values, the performance improves by increasing the number of mid-level features in the final classifier. This implies that we are not over-fitting our final classifier by selecting more mid-level features in it.

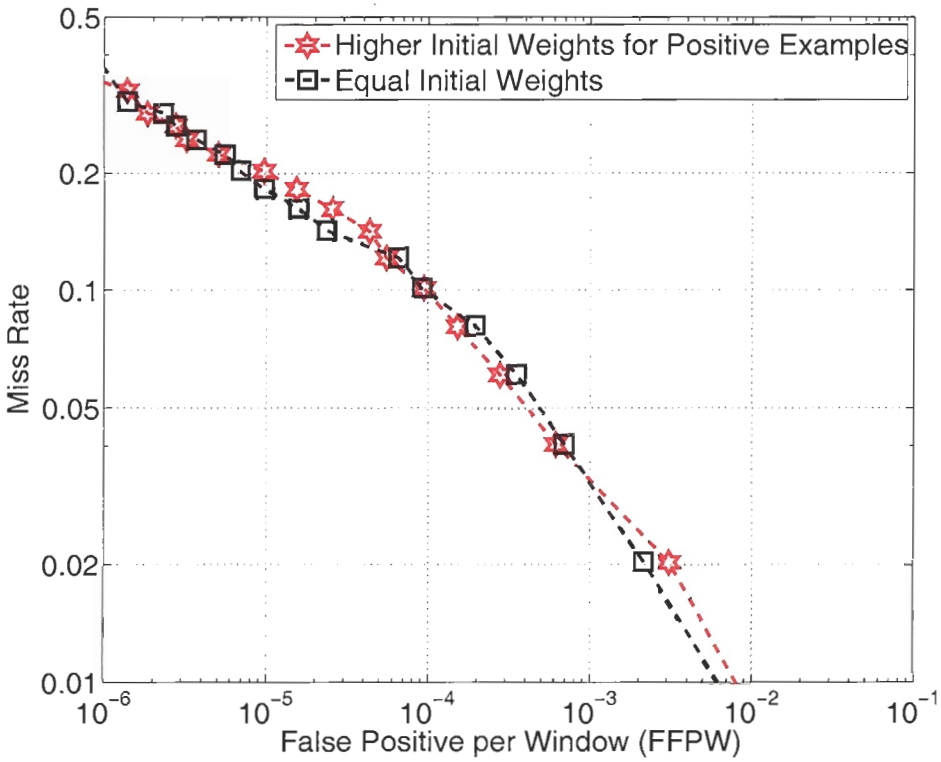


Figure 4.6: Effect of using different initial weight settings in AdaBoost. There is no significant advantage on choosing any of the settings.

4.2.3 Bootstrapping

As mentioned earlier, we use bootstrapping to retrain our classifiers. After training the mid-level features and the final classifier, we run this classifier on all of the negative training images. This way we can collect false positive negative images that are being classified as pedestrians. We use these images in addition to our original set of negative images to retrain the features and classifier. Bootstrapping will force the features and the classifier to focus more on the harder (more pedestrian-like) negative examples. There is no argument that bootstrapping will improve the classifier, but we further explored the effect of bootstrapping on different parts of our training phase. We used the $\mathcal{W}_1$ feature set introduced in section 4.2.2 for this experiment. We trained three different classifiers. First one is the

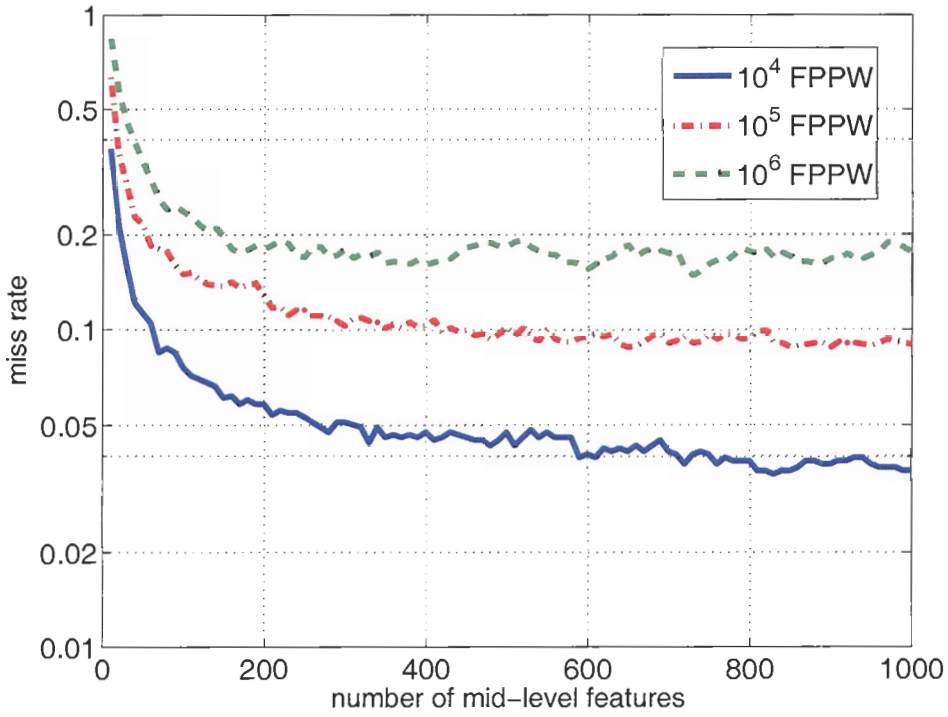


Figure 4.7: Influence of having different number of mid-level features inside the final classifier on the performance of the detector.

original trained classifier without bootstrapping. Second classifier is trained by bootstrapping only the final classifier and not retraining the mid-level features with the new negative examples. For the third classifier we retrained both mid-level features and the final classifier. Results are shown in figure 4.8, and as expected retraining both features and the final classifier will increase the performance by a factor of 10. This improvement by retraining the mid-level features is another fact, supporting the discriminativeness of mid-level features. By retraining these features, they capture more important information that are necessary for classification of pedestrian–like background images from pedestrians.

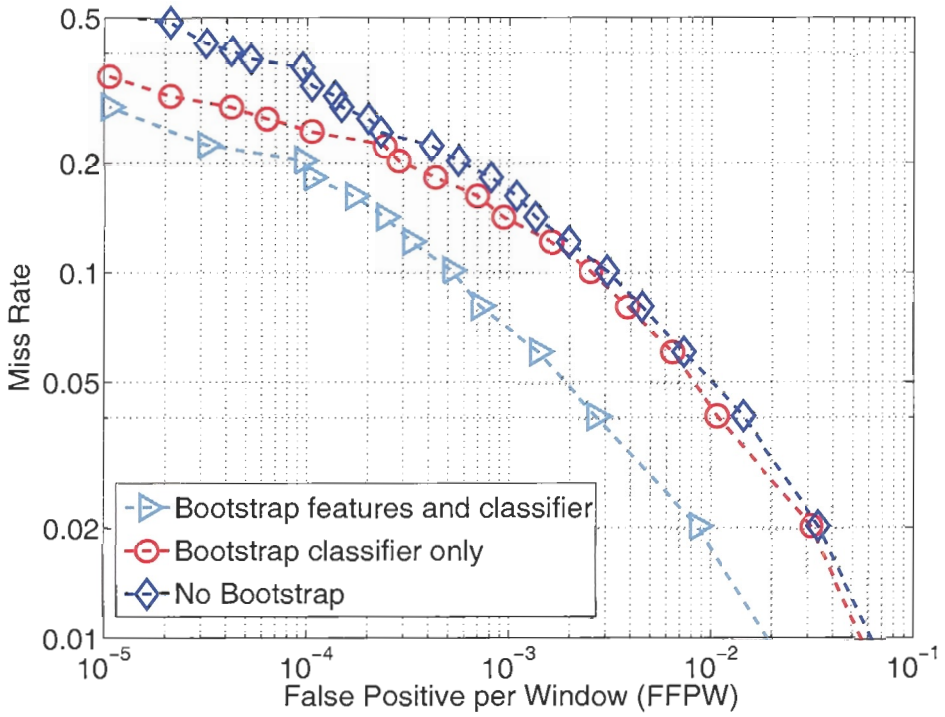


Figure 4.8: Effect of bootstrapping mid-level features and final classifier on the detector's performance.

4.2.4 Cascade

Our approach is based on classifying every possible window in the image as pedestrian or background. This can cause in long detection times for high resolution images. To overcome this problem we can employ the AdaBoost cascade method used by Viola and Jones [47] for fast face detection. A notion similar to Viola and Jones cascade is also used by Rowley et al. [40] for object detection. In Viola and Jones method, instead of making one final AdaBoost classifier using the low-level features (mid-level features in our approach), they make a cascade of classifiers. The idea for using a cascade is that the first few weak classifiers chosen by AdaBoost can classify many of the simple negative background images from pedestrians. Simpler classifiers are used in the early stages of the cascade to reject the majority of windows before more complex classifiers are called to classify harder

test windows. This way, for easier examples we do not need to compute all the mid-level features, and just by applying those in the first classifier stages we can reject most of them and continue only with harder examples. An illustration of the cascade process is shown in Figure 4.9.

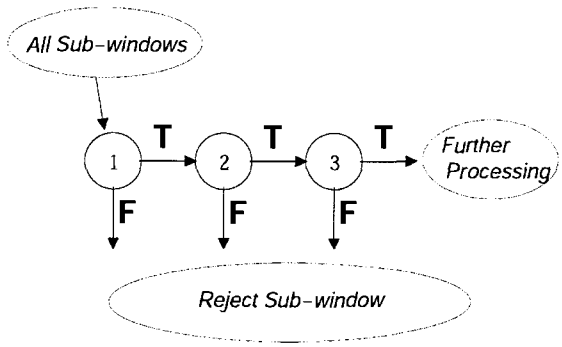


Figure 4.9: Cascade of classifiers introduced by Viola and Jones [47]. (©2001 IEEE, by permission)

By employing the cascade of classifiers method into our detector, the running time of the program dropped with a factor of 5 to 10, depending on the test images. The performance of the detector also dropped slightly. We believe the reason is the separation of more discriminative features in the early stages from the less discriminative ones in the later stages. Note that we Bootstrap the negative training set at the end of each cascade stage and choose examples that are hard for the detector to classify up to that stage. This way in final stages we might fall into the problem of over-fitting to the training set data, due to rejection of many background images in the previous stages.

4.2.5 Normalization

Normalization is another factor that is used in many detection systems to overcome the problem of different intensity and gradient responses over different images. Because of the color, illumination, and background differences from image to image, gradient responses can vary drastically. Therefore feature vector normalization is used to overcome the problem of different gradient magnitudes.

Instead of normalizing the whole image or whole gradients at the beginning, we apply our

normalization at the mid-level stage. This way we can handle local illumination changes much better. We use $L2$–norm normalization. We normalize the feature vector of every mid-level feature, by normalizing all the low-level features that fall into the window of that mid-level feature:

$$f_d = \frac{f_d}{\sqrt{(\sum_{i=1}^{k} f_d(i)^2) + \epsilon}} \tag{4.10}$$

Where f_d is the vector of low-level features inside one mid-level feature, k is the number of those low-level features, and ϵ is a small number set to 1 in our experiments. We normalize all four direction responses at once to decrease the noise level.

It appears that this normalization can make the computations very slow as we are doing it for every window. We use the notion of Integral Images, introduced by Viola and Jones [47] to overcome this problem. Integral Image is a cumulative sum of an image in 2D. At each pixel of Integral Image, you can find the sum of all the pixels in the original image at left and top of that pixel.

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{4.11}$$

Where ii is the integral image, and i is the original image.

We compute the Integral Image of a virtual image which is the square of low-level features summed up in the 4 directions.

$$ii(x, y) = \sum_{x' \leq x, y' \leq y, d \in \{1,\ldots,4\}} gi(x', y', d)^2 \tag{4.12}$$

Where gi is a 3D image of the low-level features (gradient response in the 4 directions).

Figure 4.10 compares the performance of two detectors with and without normalization. Applying this normalization method improved our results by 17% at 10^{-6} FPPW.

4.3 Analysis

The pedestrian detection approach that we introduced in this chapter has many advantages over previous pedestrian detectors. It is fast and reliable. The running time of our detector is comparable to that of Dalal and Triggs [6]. Our Matlab implementation, exhaustively

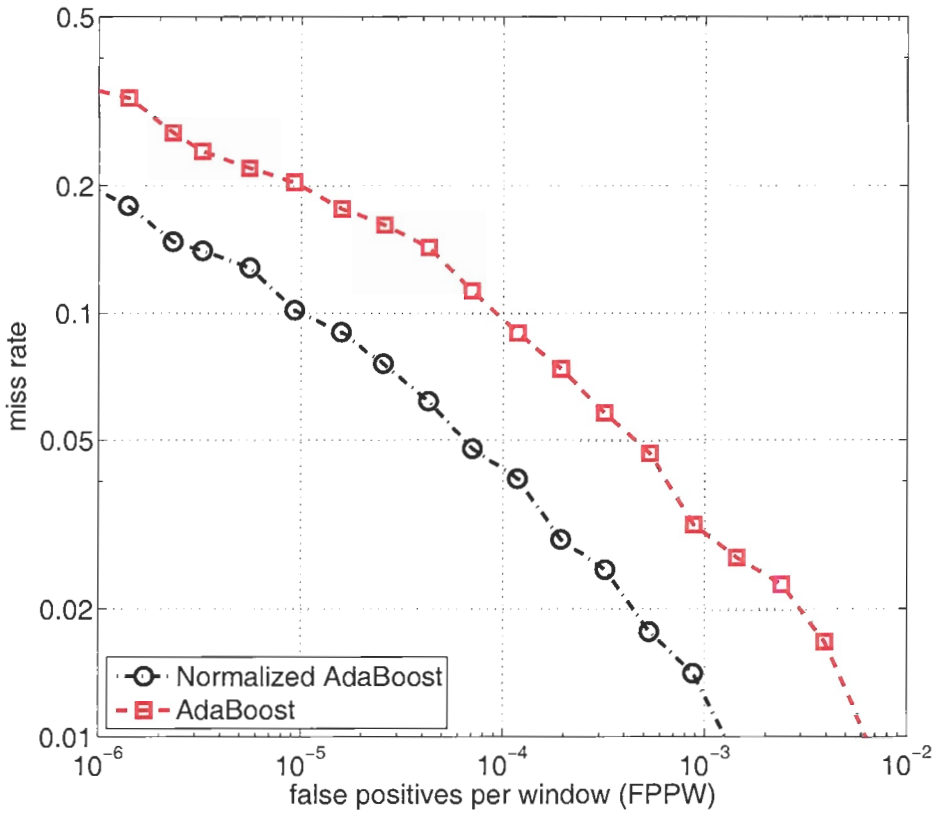


Figure 4.10: Effect of normalizing mid-level features on the detector's performance.

searches a 320×240 image in all the possible scales (about 4000 detection windows) in less than 10 seconds. By introducing the mid-level features, which is the main contribution in this work, we operationalized the notion of locally trained features. These features capture more useful information for the classification, than the fixed features, widely used for detection systems.

The performance of our final detector on INRIA dataset is shown in figure 4.11. We compare our performance results with that of Dalal and Triggs' HOG detector [6], which is the state of the art pedestrian detector. Note the higher performance of our detector by a factor of 10 at false-positive levels of as low as 10^{-6}.

Figure 4.12 shows the detection results of our detector on some sample images. The

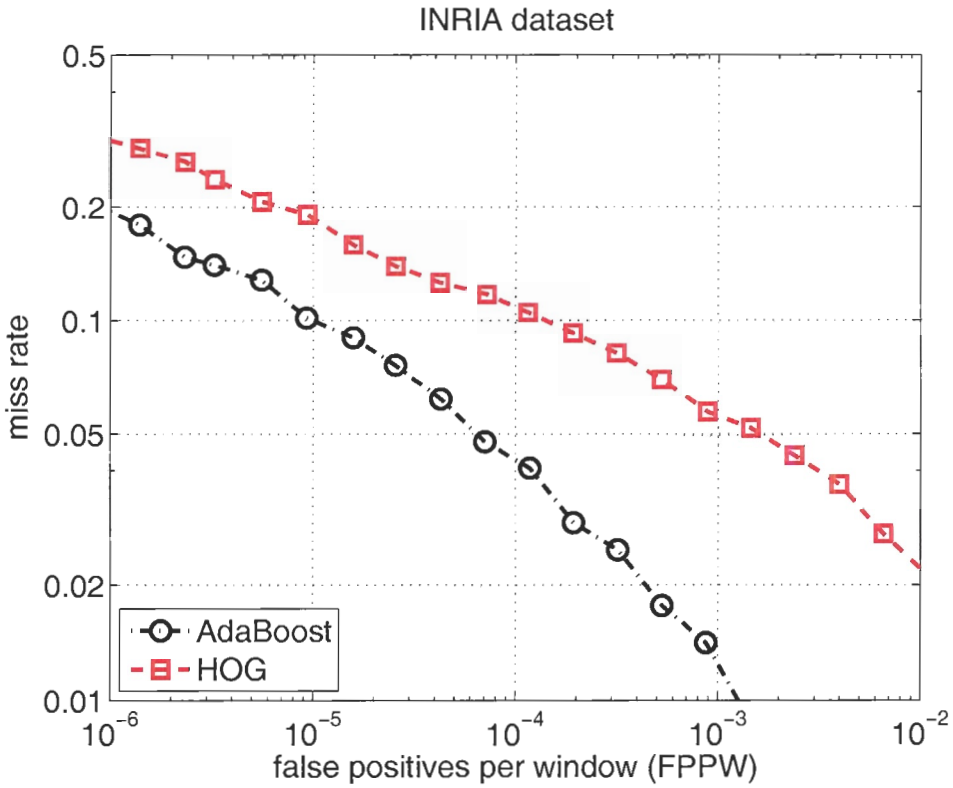


Figure 4.11: Overall Performance of our AdaBoost detector compared to Dalal and Triggs' HOG detector.

threshold used for these results is set to the one that has 10^{-5} false positive rate according to our experiments. Note the multiple detections in easy images and misses and false positives in harder ones.

Figure 4.12: Results of running the detector on some of the test images. The multiple detections are because of multi-scale search, and one pedestrian might be detected in different scales. Some non-pedestrians are also falsely detected as pedestrians (false positives) in some of the images.

Chapter 5

Conclusions

We introduced two new methods for the problem of pedestrian detection, with different auxiliary goals. One aimed to find per-pixel likelihoods while detecting pedestrians, so that those likelihoods can be used for further tasks after the detection. The focus of the second method is solely on discriminating between pedestrians and non-pedestrians, accurately and fast. The main contribution of this work is the introduction of trainable features that are used for this detector. In this chapter, we give an overall comparison of these two methods and discuss their possible extensions.

5.1 Comparison and Discussion

Our coarse shape feature detector that uses geometric blurs as the shape descriptors and kernel density estimates for class distribution estimation, does not perform adequately. The poor performance of this detector can be the result of two mistakes. First is the use of fixed, high dimensional feature vectors to describe the local shape; geometric blurs. Geometric blurs are highly descriptive and discriminative descriptors, and they can be used to match feature points very accurately. The discriminativeness of these features is one main source of our problems. To interpret the shape of an object, globally or at a pixel, we need features that can be blurred over all the training examples. But geometric blur features will differ between every two examples, and this problem is magnified when we have a non-rigid object like pedestrians. Because of this problem, training examples will fall into different places inside the geometric blur feature space. Kernel density estimation could help us to model these sparse set of features and estimate the overall distribution, but high dimensionality of

the feature space and lack of enough training examples to fill that space, made us unable to have accurate estimates.

Our second approach, that uses AdaBoost to train features and classifiers, tries to overcome the problems that led to failure of the first approach. We needed some features that are trainable and not fixed, and also their dimensionalities are as low as possible. To do so, we introduced the trainable mid-level features, that describe the shape of the object locally. These features are very discriminative regarding the two object classes, unlike the geometric blur features that are generally discriminative. The mid-level features are very low dimensional because they only contain the dimensions that have useful information for the classification task. Another change that we made in the design of this detector comparing to the previous one, was the use of AdaBoost to create our final classifier instead of estimating object class likelihoods at each pixel. This way, we only use a subset of features that can describe our objects. It will prevent repetition of information and decreases the amount of noise. As a result, the performance of the second detector increased substantially and is even better than the performance of the state of the art pedestrian detectors. Figure 5.1 illustrates the performance of our two detectors and the state of the art HOG detector [6].

As mentioned, the main contribution of this thesis is the introduction of trainable mid-level features. As a direct result of trainability of these features, beside the low-dimensionality and discriminativeness, is their scalability. These features can be trained for any object class and there is no need to change the design of the features depending on the classes that we want to train our detector for. We only need to re-train the mid-level features and the final classifier for the new object class.

The only advantage of our kernel density estimate detector over the AdaBoost detector, is the final per-pixel likelihood results that it provides. Those results can be used for other applications other than only detection, such as segmentation. The AdaBoost detector, cannot provide such output because of the sparsity of its selected features.

5.2 Future Work

Each of our detectors can be extended in different ways. We mostly focus on the second one, AdaBoost detector, because of its promising results.

The KDE-based detector can be extended by embedding a segmentation algorithm in it.

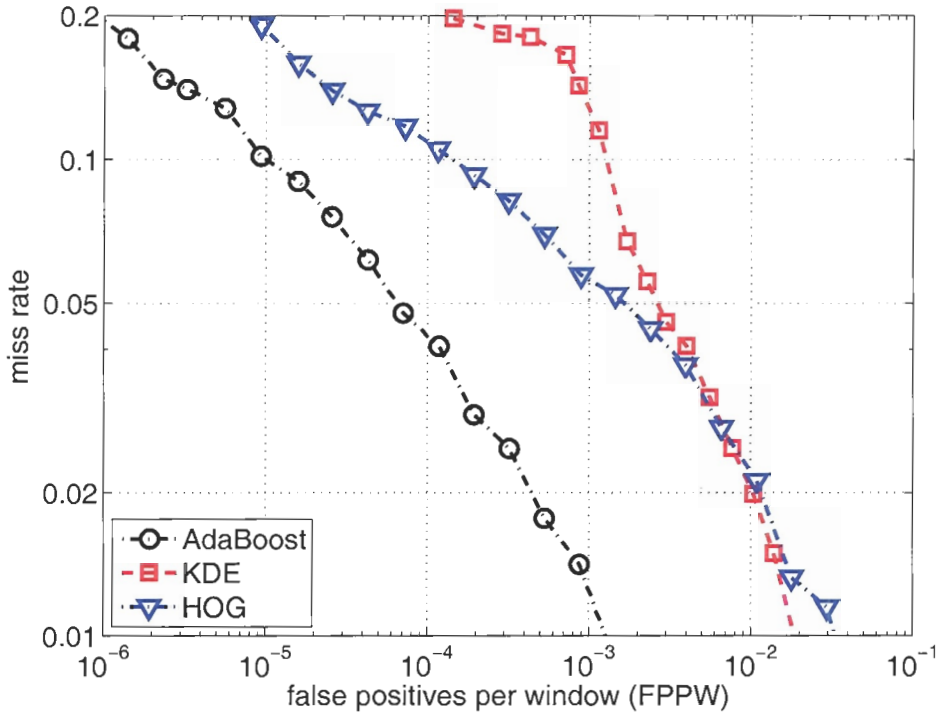


Figure 5.1: Comparison results of our two detectors and HOG detector.

We conducted some preliminary tests using color segmentation to over segment the image and do the detection and segmentation of the pedestrian from background at the same time. We observed that segmentation during detection can improve the results of both detection and segmentation task, but as those experiments are out of scope of the focus of this thesis we did not further explored the experiments.

Our second method, the AdaBoost-based detector, can be extended in different ways, due to its flexibility and performance:

- We can easily use the same structure of the detector for detecting other objects than only pedestrians. Without changing many parameters, we can train the mid-level features and the final classifier for another object category.

- AdaBoost is not only a two-class classification algorithm. It can be extended easily to classify more than two classes. As AdaBoost is the main underlying algorithm for

both feature creation and classifier training tasks of our approach, extending our work to multi-class classifications is a plausible task.

- We showed that adding an extra layer of features, by training them using the AdaBoost, can improve the results of object detection task. An interesting further investigation of this approach is to use a pyramid hierarchy of mid-level features instead of having only one layer. That way we might be able to describe different object parts in the hierarchy as well as improving the detection performance.

- Hoiem et al. [18] use the context and perspective to search for objects the way humans do. They show that with probabilistic estimation of surfaces and world coordinates, one can put objects into perspective and model the scale and location variance in the image. This way not everywhere and every scale in the image can contain our target objects. By incorporating this method with our detector, not only we can have a much faster running time because of the reduced number of possible pedestrian windows, but also the false positive rate can decrease substantially.

Bibliography

[1] A. Agarwal and B. Triggs. 3d human pose from silhouettes by relevance vector regression. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 02, pages 882–888, 2004.

[2] S. Agarwal and D. Roth. Learning a sparse representation for object detection. In *Proc. 7th Europ. Conf. Computer Vision*, volume 4, pages 113–130, 2002.

[3] H.G. Barrow, J.M. Tenenbaum, R.C. Bolles, and Wolf H.C. Parametric correspondence and chamfer matching: Two new techniques for image matching. *Proc. 5th Int. Joint Conf. Artificial Intelligence*, pages 659–663, 1977.

[4] A. Berg and J. Malik. Geometric blur for template matching. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, pages 607–614, 2001.

[5] A.C. Berg, T.L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, pages 26–33, 2005.

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, 2005.

[7] P.F. Felzenszwalb. Learning Models for Object Recognition. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, pages 56–62, 2001.

[8] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[9] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, 148:156, 1996.

[10] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[11] Y. Freund and R.E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

[12] D.M. Gavrila. Visual analysis of human movement: A survey. *Computer Vision and Image Understanding*, 73(1):82–98, 1999.

[13] D.M. Gavrila, J. Giebel, S. Munder, D.C. Res, and G. Ulm. Vision-based pedestrian detection: the PROTECTOR system. *Intelligent Vehicles Symposium, 2004 IEEE*, pages 13–18, 2004.

[14] D.M. Gavrila and V. Philomin. Real-time object detection for smart vehicles. In *Proc. 7th Int. Conf. Computer Vision*, volume 1, 1999.

[15] T. Hastie, R. Tibshirani, and J.H. Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2001.

[16] E. Hjelmas and B.K. Low. Face detection: A survey. *Computer Vision and Image Understanding*, 83(3):236–274, 2001.

[17] D. Hogg. Model-based vision: a program to see a walking person. *Image and Vision Computing*, 1(1):5–20, 1983.

[18] D. Hoiem, A.A. Efros, and M. Hebert. Putting objects in perspective. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 2, pages 2137–2144, 2006.

[19] C. Huang, H. Al, B. Wu, and S. Lao. Boosting nested cascade detector for multi-view face detection. *Pattern Recognition, 2004. Proceedings of the 17th International Conference on*, 2, 2004.

[20] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[21] B. Leibe, E. Seemann, and B. Schiele. Pedestrian detection in crowded scenes. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 1, 2005.

[22] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[23] K. Mikolajczyk, B. Leibe, and B. Schiele. Local Features for Object Class Recognition. In *Proc. 10th Int. Conf. Computer Vision*, volume 2, 2005.

[24] K. Mikolajczyk, C. Schmid, and A. Zisserman. Human detection based on a probabilistic assembly of robust part detectors. In *Proc. 8th Europ. Conf. Computer Vision*, volume 1, pages 69–81, 2004.

[25] T.B. Moeslund and E. Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding*, 81(3):231–268, 2001.

[26] A. Mohan, C. Papageorgiou, and T. Poggio. Example-based object detection in images by components. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(4):349–361, 2001.

[27] G. Mori and J. Malik. Estimating human body configurations using shape context matching. In *Proc. 7th Europ. Conf. Computer Vision*, volume 3, pages 666–680. Springer, 2002.

[28] S. Munder and D.M. Gavrila. An experimental study on pedestrian classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1863–1868, 2006.

[29] K. Okuma, A. Taleghani, N. de Freitas, J.J. Little, and D.G. Lowe. A boosted particle filter: Multitarget detection and tracking. In *Proc. 8th Europ. Conf. Computer Vision*, volume 1, pages 28–39, 2004.

[30] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, pages 193–199, 1997.

[31] J. O'Rourke and N.I. Badler. Model-Based Image Analysis of Human Motion Using Constraint Propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(6):522–536, 1980.

[32] J. O'Sullivan, J. Langford, R. Caruana, and A. Blum. FeatureBoost: A Meta-Learning Algorithm that Improves Model Robustness. *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 703–710, 2000.

[33] C. Papageorgiou and T. Poggio. Trainable pedestrian detection. In *Proc. 7th Int. Conf. Computer Vision*, volume 4, 1999.

[34] C. Papageorgiou and T. Poggio. A Trainable System for Object Detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.

[35] X. Ren and J. Malik. Learning a Classification Model for Segmentation. In *Proc. 9th Int. Conf. Computer Vision*, pages 10–17, 2003.

[36] K. Rohr. Incremental recognition of pedestrians from image sequences. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, pages 8–13, 1993.

[37] R. Ronfard, C. Schmid, and B. Triggs. Learning to parse pictures of people. In *Proc. 7th Europ. Conf. Computer Vision*, 2002.

[38] R. Rosales and S. Sclaroff. Inferring body pose without tracking body parts. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 02, page 2721, 2000.

[39] R. Rosales, M. Siddiqui, J. Alon, and S. Sclaroff. Estimating 3D Body Pose using Uncalibrated Cameras. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 1, pages 821–827, 2001.

[40] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[41] B. Schiele and J.L. Crowley. Recognition without Correspondence using Multidimensional Receptive Field Histograms. *International Journal of Computer Vision*, 36(1):31–50, 2000.

[42] H. Schneiderman and T. Kanade. A statistical method for 3d object detection applied to faces and cars. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 1, pages 746–751, 2000.

[43] H. Schneiderman and T. Kanade. Object Detection Using the Statistics of Parts. *International Journal of Computer Vision*, 56(3):151–177, 2004.

[44] C. Sminchisescu and B. Triggs. Covariance scaled sampling for monocular 3D body tracking. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 1, pages 447–454, 2001.

[45] C. Sminchisescu and B. Triggs. Estimating Articulated Human Motion with Covariance Scaled Sampling. *The International Journal of Robotics Research*, 22(6):371–391, 2003.

[46] M.J. Swain and D.H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[47] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, 2001.

[48] P. Viola, M. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. In *Proc. 9th Int. Conf. Computer Vision*, pages 734–741, 2003.

[49] M.P. Wand and M.C. Jones. *Kernel Smoothing.* Chapman & Hall/CRC, 1995.

[50] A. Webb. *Statistical Pattern Recognition.* Hodder Arnold, 1999.

[51] B. Wu and R. Nevatia. Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In *Proc. 10th Int. Conf. Computer Vision*, 2005.

[52] C. Yang, R. Duraiswami, N.A. Gumerov, and L. Davis. Improved fast gauss transform and efficient kernel density estimation. In *Proc. 9th Int. Conf. Computer Vision*, pages 664–671, 2003.

[53] T. Zhao and R. Nevatia. Bayesian human segmentation in crowded situations. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 2, pages 459–466, 2003.

[54] T. Zhao and R. Nevatia. Tracking multiple humans in crowded environment. In *Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition.*, volume 2, 2004.