MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Pedestrian Detection Using Boosted Features over Many Frames

Michael Jones, Daniel Snow

TR2008-027 July 2008

Abstract

A scanning window type pedestrian detector is presented that uses both appearance and motion information to find walking people in surveillance video. We extend the work of Viola, Jones and Snow [18] to use many more frames as input to the detector thus allowing a much more detailed analysis of motion. The resulting detector is about an order of magnitude more accurate than the detector of Viola, Jones and Snow. It is also computationally efficient, processing frames at the rate of 5 Hz on a 3 GHz Pentium processor. The detectorś accuracy and speed make it practical for real applications.

CVPR 2008

Pedestrian Detection using Boosted Features over Many Frames

Michael J. Jones Daniel Snow
Mitsubishi Electric Research Labs
201 Broadway, Cambridge, MA 02139
{mjones,snow}@merl.com

Abstract

A scanning window type pedestrian detector is presented that uses both appearance and motion information to find walking people in surveillance video. We extend the work of Viola, Jones and Snow [18] to use many more frames as input to the detector thus allowing a much more detailed analysis of motion. The resulting detector is about an order of magnitude more accurate than the detector of Viola, Jones and Snow. It is also computationally efficient, processing frames at the rate of 5 Hz on a 3 GHz Pentium processor. The detector's accuracy and speed make it practical for real applications.

1. Introduction

$\bar{W}_J^{N+1}$

This paper extends the work of Viola, Jones and Snow [18] on detecting pedestrians in video using both appearance and motion information. The goal is to develop an algorithm with enough speed and accuracy to be useful for detecting pedestrians in real outdoor surveillance scenarios. In typical outdoor surveillance scenarios, the camera is static and the pedestrians are often very small (around 20 pixels tall). Although the pedestrian detector described in [18] achieved state-of-the-art accuracy with a detection rate of about 90% and around 1 in 100,000 false positives per image window, this false positive rate is still too high to be truly useful for real applications. The goal here is to reduce the false positive rate by at least an order of magnitude while maintaining the high detection rate.

The key to achieving this is to use more video frames as the input to the pedestrian detector. Instead of the minimal two frames used in [18], we use ten frames which allows much more motion information to be analyzed. The other major difference is that we divide the detector into eight separate direction-specific detectors (north, northeast, east, southeast, etc). This is analogous to the state-of-the-art in multi-view face detection in which different face poses are ultimately detected by separate detectors [10, 9, 12].

The basic framework is to use the AdaBoost learning algorithm to select a set of features that separates pedestrians from non-pedestrians for each direction class. The input to the classifier is small image windows from ten consecutive frames of a video sequence. An image window is simply a patch or rectangular area of an image. The feature set consists of Haar-like features that act either on a single frame, on the difference between two frames or on the shifted difference between two frames. The first type of feature discriminates based on the appearance of pedestrians and the second and third types discriminate based on the motion. The features will be described in more detail later. In addition we use a soft cascade [1] (instead of a cascade) to make the resulting detector very efficient to compute. Soft cascades were chosen over standard cascades because the speed versus accuracy trade-off can be more easily optimized with soft cascades.

2. Related Work

There have been many papers in the area of person or pedestrian detection. Gavrila [7] provides a good survey of the field. We concentrate here on the papers most relevant to our work. Many papers have focused on the problem of detecting fairly large pedestrians (in terms of number of pixels) in static images. These include the work of Dalal and Triggs [3], Zhu et al. [19], Tuzel et al. [16], Leibe et al. [11], Papageorgiou et al. [14] and Mohan et al. [13]. In all of these the people must be around 100 pixels tall to be detected. Even with such relatively high resolution, the best reported system [16] has an accuracy of about 90% detection rate with about 1 in 10,000 false positives per window on the difficult INRIA test set.

Dalal, Triggs and Schmid [4] extend the previous work of Dalal and Triggs [3] to handle video from moving cameras. They use an optical flow based approach along with histograms of oriented gradients to detect large pedestrians from moving cameras. Various other researchers have also looked at pedestrian detection from video sequences. Gavrila et al. [8] present a vision-based pedestrian detection system designed for use in a moving car. The system integrates various techniques for finding pedestrians including the use of stereo cameras. Although their system is practical (or very nearly so) for use in automobiles, it is not directly applicable to a single camera surveillance scenario. Like us, Efros et al. [6] were also interested in people that are around 30 pixels high in video sequences, but their focus was on analyzing the actions of such people and not in detecting them in the first place. An even earlier algorithm that worked on low resolution pedestrians in video was Cutler and Davis [2]. They used periodic motion to detect walking people and other objects in video. Their system analyzed much longer video sequences than ours in order to notice periodicity. Our approach has the advantage of requiring shorter video sequences (less than one period) to detect a pedestrian. Also, Cutler and Davis's system was never tested for the purpose of finding detection rates and false positive rates on a large test set so it is not known how accurate their system is on the detection task in real scenarios.

The approach taken in this paper is distinguished from previous work by various attributes. It analyzes a moderate number of frames at once (10 frames), it detects very small pedestrians (as few as 20 pixels high), has a much lower false positive rate than static detectors (about 1 in a million false positives per window) and is fast (about 5 Hz for 360x240 pixel images on a P4 3 GHz machine).

3. Input Representation

The input representation used in this work is one of its main distinguising aspects. Ten consecutive frames from a 15 fps camera (or every other frame from a 30 fps camera) are used as input to the pedestrian detector. This allows a modest amount of motion (about one walking step depending on walking speed) to be analyzed by the classifier. For the low resolution pedestrians we are trying to detect, the motion information is more informative than appearance and so using more frames can potentially lead to much greater accuracy. We have chosen a base input window size of 24 pixels wide by 26 pixels high. The window has to be large enough to allow a pedestrian to be visible within it both in frame 1 and in frame 10. The typical pedestrian height in our training examples is 20 pixels. An *example* (used for training a classifier) consists of 10 image windows from 10 consecutive frames. Each window is in exactly the same position in all 10 frames. For some typical positive examples in the east, south and northeast directions see figure 1.

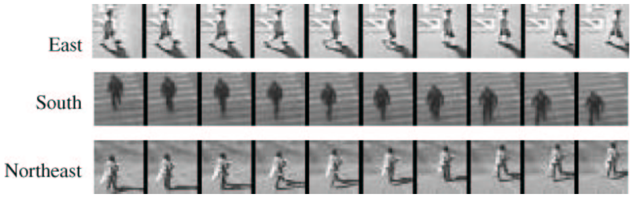

Figure 1. Positive examples for three different directions of pedestrian motion. Each row of 10 windows comprises one example.

At this resolution, a pedestrian is roughly a person shaped blob usually moving in an approximately straight line with a walking motion in the leg region and a mostly translational motion in the torso region (arms are usually not very visible). We are not using enough frames to judge periodicity (a la Cutler and Davis [2]), but leg motion is nevertheless distinctive. Very few if any non-pedestrian windows in a video meet this description. Using ten frames and the simple features introduced in the next section allows this type of description to be learned. This representation also implies that it will be very advantageous to split the detector into different classes based on direction of movement. The idea of using different detectors for different directions of motion is analogous to the state-of-the-art in face detection in which different views of faces are handled by different detectors [10, 9, 12]. Splitting the detector into multiple detectors for each of the various walking directions raises the question of what range of direction angles should each direction-specific detector handle. We did some experiments with east moving pedestrians and found that 45° could be covered reasonably well with a single detector. This means 8 different detectors are required to cover the full 360° range of directions. We did not fully explore the trade-off between number of direction-specific detectors and accuracy, and so the use of 8 detectors may not be optimal. Finally, in this work we combined the 8 detectors in the simplest possible way, by running all detectors on every image window and merging any overlapping detections. More sophisticated techniques could lead to further improvements.

Each detector is trained at the base scale (24x26 pixels). To detect pedestrians at larger scales we use an image pyramid and scan over each scale of the pyramid independently. More details will be given on training and scanning later.

The difficulty in using 10 frames instead of one or two is designing a feature set that can take advantage of the extra information without growing exponentially and becoming computationally too large to handle.

4. Features

Like the face detection work of Viola and Jones [17], our pedestrian detection procedure classifies gray scale image windows based on the values of simple features. The features are selected from a large pool of possible features using Real AdaBoost [15]. The learning procedure is described in the next section. First we will describe the various types of features that make up the pool of possible features.

A feature in this work is a simple binary classifier. It

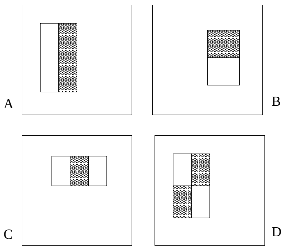

Figure 2. Example Haar-like filters shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Haar-like filters with two rectangles are shown in (A) and (B). Figure (C) shows one with three rectangles filter, and (D) with four rectangles.

consists of a *filter* that takes image windows from n consecutive frames as input, a threshold and a positive and negative vote. Thus, a feature, h_i, is defined as

$$h_i(x_1, x_2, \ldots, x_n) = \begin{cases} \alpha & \text{if } f_i(x_1, x_2, \ldots, x_n) > \theta_i \\ \beta & \text{otherwise} \end{cases} \tag{1}$$

where x_j is an image window from frame j and x_k is the same image window from frame k, $f_i()$ is a filter, θ_i is a threshold, and α and β are real valued true and false votes respectively. In the experiments reported here, $n = 10$.

There are three basic types of filters. The first type is an appearance filter. It consists of a Haar-like filter that acts on a window from a single input frame. The Haar-like filters used in this work are shown in figure 2. As in [17], these filters simply take the sum of pixel brightness values within the white rectangles and subtract the sum of brightness values within the dark rectangles. In addition, an absolute value version of each appearance filter is also included in the filter set. This is important for pedestrian detection because it allows, for example, a darkly clothed person on a light background to give the same response as a lightly clothed person on a dark background. Thus, appearance filters have one of the following two forms:

$$f_i(x_1, x_2, \ldots, x_n) = b(x_i) \tag{2}$$

or

$$f_i(x_1, x_2, \ldots, x_n) = |b(x_i)| \tag{3}$$

where $b(x_i)$ is one of the Haar-like filters of the types shown in figure 2 computed on image window x_i.

The second type of filter is a difference filter which computes the absolute value of the difference between a Haar-like filter acting on image windows in two different input frames. This allows various patterns of motion to be detected. In this case, the Haar-like filters include the types in figure 2 as well as ones with a single rectangle acting on each frame. Difference filters have the following form:

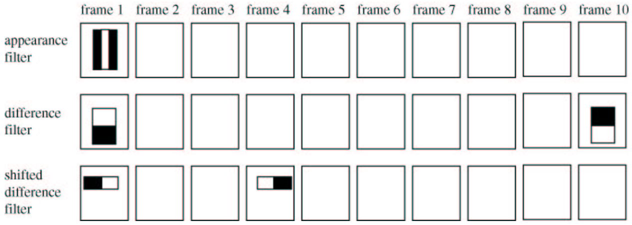

Figure 3. Three different types of filters are shown. The first row illustrates one particular appearance filter. Such filters act on a single window, which could be any of the 10. The second row shows a particular difference filter. Such filters consist of one filter acting on one window and the negative of that filter acting on another window. The third rows shows a particular shifted difference filter. Such filters are similar to difference filters except the second filter is a shifted negative version of the first.

$$f_i(x_1, x_2, \ldots, x_n) = |b(x_i) - b(x_j)|. \tag{4}$$

The third type of filter is a shifted difference filter which computes the absolute value of the difference between a Haar-like filter acting on a window in one input frame and a shifted version of the same filter acting on the same window in a second input frame. This allows motions in a particular direction (the direction of the shift) to be detected. Shifted difference filters have the following form:

$$f_i(x_1, x_2, \ldots, x_n) = |b(x_i) - b_s(x_j)| \tag{5}$$

where $b_s(x_j)$ represents the Haar-like filter $b(x_j)$ shifted by s pixels up, down, left or right. In practice, we trained using only filters with shifts of 1 or 2 pixels.

Although a filter takes windows from n frames as input, any particular filter only uses 1 or 2 of those windows to compute its value. The three basic types of filters are illustrated in figure 3. The main reason for using these types of simple features is that they are very fast to compute. The integral image representation [17] can be used to make computation of such filters extremely efficient.

If all possible filters of the types described above were generated, the filter set would be extremely large which would make training time and memory requirements impractical. To solve this problem, we use the simple approach of subsampling the total filter set to yield a managable size of a few tens of thousands of filters. In practice we have found that increasing the size of the filter set beyond this only leads to small improvements in accuracy.

The main difference between these filters and those in Viola, Jones and Snow [18] are that these can take advantage of the larger number of frames in the input. Also, in our case the filters are shifted in the case of shifted difference filters instead of having filters act on shifted difference images as in [18]. Shifting the filters is computationally much more efficient especially when windows from 10 frames are used as input.

5. Learning Framework

The learning framework we use is almost the same as in [18] except that instead of building a cascade of classifiers we use a soft cascade [1]. A soft cascade is a single strong classifier (linear combination of weak classifiers) in which evaluation of the classifier can terminate early if the cumulative sum of weak classifier outputs falls below one of the rejection thresholds associated with each weak classifier. The advantage of this approach over a cascade is that it is easier to trade off between speed and accuracy by simply changing the rejection thresholds. The rejection thresholds are set after AdaBoost training and can be adjusted later depending on speed and accuracy requirements.

The AdaBoost learning algorithm [15] is used to both select a series of features (weak classifiers) from the pool of possible features and to set the parameters for each feature. The parameters of a feature are θ, α and β as described in section 4. The result of AdaBoost training is a strong classifier of the form

$$C(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \sum_{i=1}^{N} h_i(x_1, x_2, \ldots, x_n) \geq T \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

To turn this into a soft cascade, N rejection thresholds are selected. After the value of each feature is added to the sum in equation 6, the sum is compared against that feature's rejection threshold. If the sum is less than the rejection threshold, computation halts and the strong classifier returns 0, "not pedestrian". Otherwise computation continues. The rejection thresholds, r_i are determined after AdaBoost training based on desired speed and accuracy tradeoffs. Please see Bourdev and Brandt [1] for an algorithm for computing rejection thresholds as well as more details on soft cascades.

The training procedure requires difficult negative examples to learn a detector with high accuracy. This problem arises from the fact that the set of negative examples is so astronomically large that a reasonable sample for training cannot possibly be representative. In practice it is easy to gather an extremely large set of negative examples since a single image contains hundreds of thousands of image windows most of which are negative examples. We assume that the learning procedure has such a large set of training examples which we will call the full training set. We will call the manageable subset of the full training set that is used for training at any one time the working set. In [1], a bootstrapping method is used to add a few difficult negative examples to the training set during every boosting round. We use a related but different approach to incoporating difficult training examples. In our training procedure, AdaBoost is interrupted occasionally so that a new working set can be computed that contains difficult training examples. The new working set is found by sampling from the full training set using a distribution that puts more weight on difficult examples. An example is considered difficult if it has small or negative margin with respect to the current strong classifier. The margin of example $\mathbf{x}_i$ is defined as $y_i \sum_k h_k(\mathbf{x}_i)$. where $y_i \in \{+1, -1\}$ is the ground truth label of example $\mathbf{x}_i$ and h_k is the kth feature. We call this process of sampling from the full training set according to the distribution of margins *resampling*. Because pedestrians are extremely rare compared to non-pedestrians, the resampling is done separately for positive and negative examples to yield an equal number of each.

Thus, to train a strong classifier with resampling, an initial working set is gathered (for example by resampling using a uniform distribution) and AdaBoost is used to select weak classifiers until the error on the current training set falls below some threshold. At that point, resampling is done to pick a new more difficult working set and AdaBoost training is resumed. This process of resampling and then AdaBoost training is repeated until the desired number of features are learned or the desired accuracy is achieved on a validation set.

6. Experiments

Using the framework described above, we trained eight different pedestrian detectors for the following directions of motion: north, south, east, west, northeast, southeast, northwest, southwest. In practice, only five of the detectors were actually trained. The other three were computed by "flipping" other detectors. Flipping a detector means that the mirror image of each filter is computed by reflecting it over the central vertical axis of the image window. Using this idea, the east pedestrian detector is used to derive a west pedestrian detector, the southeast detector is used to derive the southwest detector, and the northeast detector is used to derive the northwest detector. This works because a person walking west is a mirror image of a person walking east.

The training data consisted of frames from 25 video sequences of various lengths with a total of 78,858 frames. The training sequences came from shooting short (a few minutes long) video sequences using a consumer mini-DV camcorder in various outdoor locations. Each sequence was of an outdoor scene in which the camera was relatively high (from about 1 to 8 stories high). Bounding boxes around each pedestrian for every frame were marked by hand.

From this raw data, an initial set of positive and negative training examples (consisting of 10 image windows each) were extracted for each direction. The direction of motion was estimated by computing the angle between the center of

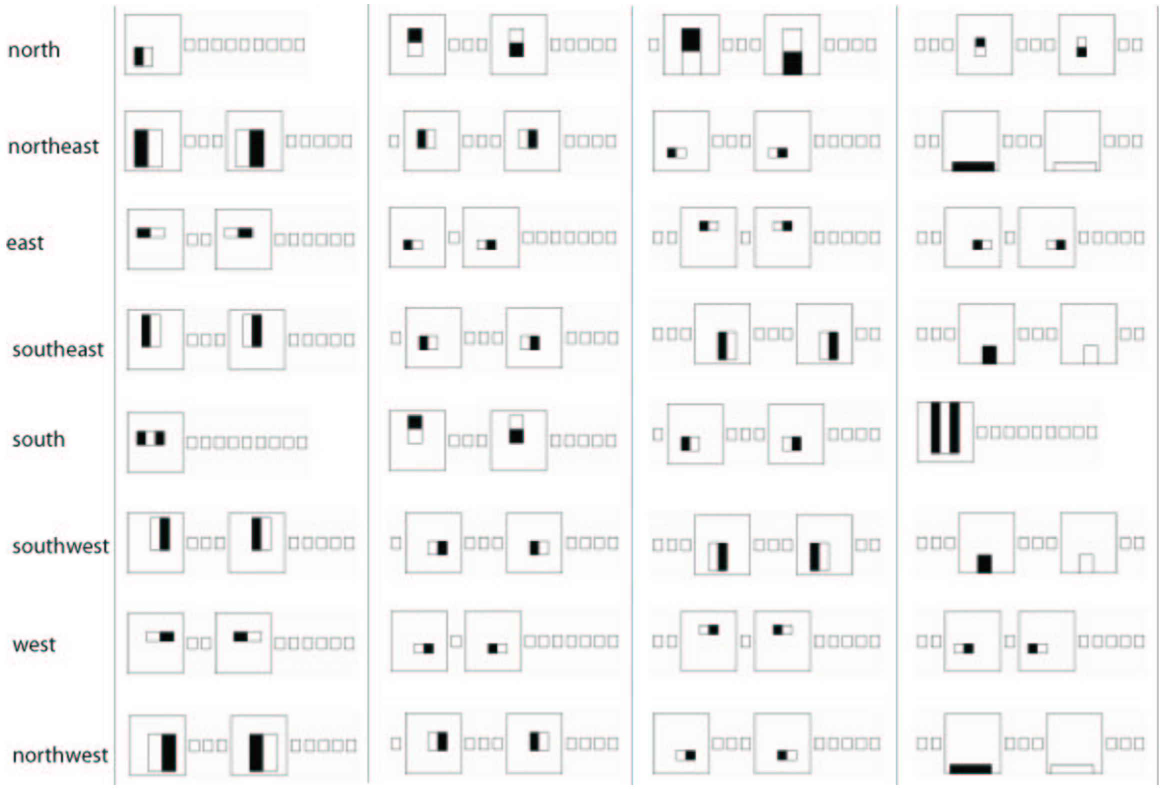

Figure 4. The first four filters selected by AdaBoost for each direction-specific detector. Most of the windows in each sequence of 10 input windows are represented by small rectangles since the selected filter does not use them. The one or two windows used by the filter are drawn larger with the Haar-like filter illustrated within it. It would be difficult to distinguish shifted difference filters from non-shifted difference filters, however, none of the first four filters is shifted for any detector. See the text for a discussion of the selected filters.

the pedestrian box in frame 1 and in frame 10. The amount of motion had to be at least 0.5 pixels per frame. This set a minimum on the speed of the pedestrian used for training. Each pedestrian bounding box was scaled to be 24x26 pixels (width × height). This yielded examples such as in figure 1. For negative examples, image windows from 10 consecutive frames that did not significantly overlap with pedestrian boxes were selected (and scaled to 24x26 pixels). The initial training set consisted of 2500 positive examples and 2500 negative examples.

For training, about 40,000 filters were randomly selected from the much greater set of possible filters of the three basic types shown in figure 3. All of the labeled video frames from all 25 sequences were used for the negative resampling set. Real AdaBoost learning with resampling was run for 400 iterations to yield strong classifiers with 400 features for each of the 5 directions trained. As mentioned above, the three classifiers for the remaining three directions were computed by flipping the appropriate trained classifier.

To evaluate a detector, it must be scanned across all positions, scales and frames of a video sequence. The details of the scanning process are as follows. First, an image pyramid is created for each frame. We used a scale factor of 0.75 between frames of the pyramid. For a set of 10 consecutive frames, the detector is evaluated in each position of each level of the image pyramid while shifting the detection window with a 1 pixel step size. For a particular position, the ten 24x26 pixel image windows are given as input to the soft cascade detector which outputs a 0 or 1 to indicate whether a pedestrian was detected in that position. If two or more detections overlap significantly then their bounding boxes are merged by averaging their top left x and y coordinates and their heights and widths. For a 360 x 240 pixel image, our scanning process evaluates 176,920 windows.

Each of the 8 detectors is evaluated in sequence on each window in the image and the union of all merged bounding boxes are returned as detections. One could undoubtedly use more efficient technique for combining the eight detectors (such as following the recent work in multi-view face detection), but this improvement has not been pursued here.

7. Results

A test set consisting of 21 video sequences was collected in similar but different scenarios to the training sequences.

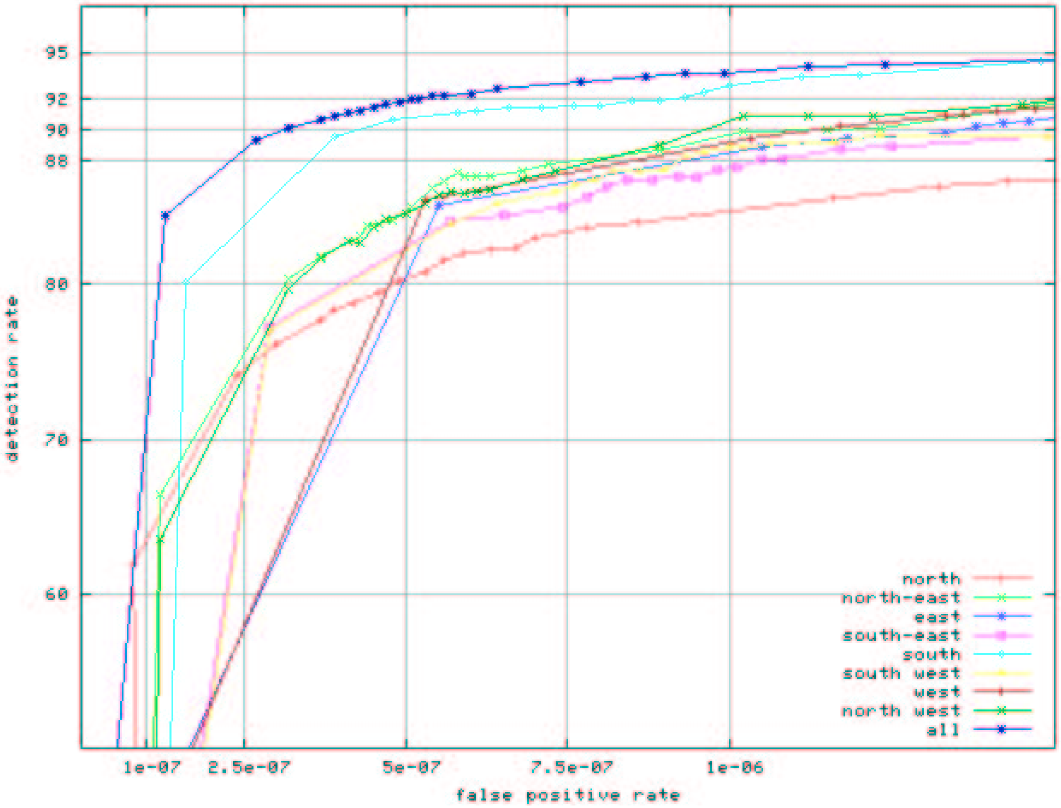

Figure 5. ROC curve for all 8 direction-specific detectors plus the combined detector. The combined detector achieves a detection rate of 93% with a 1 in a million false positive rate per window.

All but one of these sequences were collected with the same camera as the training set. The final test sequence we took from the PETS 2001 dataset [5]. We doubled the size of the testing set by flipping (mirror image) all frames of each video. This means that for each pedestrian walking east, for example, the flipped video contains pedestrians walking west. The total number of frames (including the flipped frames) was 83,152 and the total number of tracks was 450.

The detection rate for a detector is computed over all pedestrians in all frames independently. This means that the total number of pedestrians to be detected in the test set is the sum off all pedestrians in all frames. No tracking information is used. The same pedestrian in frame 1 and frame 2 is counted as two instances to be detected. The pedestrian in frame 1 will be detected while looking at the appropriate window in frames 1 through 10 while the pedestrian in frame 2 will be detected while looking at the appropriate window in frames 2 through 11. For each direction-specific detector, the positive examples only include pedestrians walking in the appropriate direction.

A receiver operating characteristic (ROC) curve which plots detection rate versus false positive rate was computed for each direction specific pedestrian detector on the test set and for the combined detector (consisting of the merged results of all 8 detectors). These ROC curves are shown in figure 5. The results show that each direction-specific detector achieves a detection rate from 93% to 84% with a false positive rate of 10^{-6}. The combined detector achieves 93% detection rate with about a 10^{-6} false positive rate. This is about an order of magnitude improvement over Viola, Jones and Snow [18]. Since a single 360 x 240 pixel frame has 176,920 windows in it, this false positive rate is equivalent to .177 false positives per frame.

How can the combined detector do better than the individual detectors? The most common false positive for a direction-specific detector is a detection on a pedestrian or part of a pedestrian moving in a different direction from the direction it was trained on. Such detections are false detections for the direction-specific detectors. However, for the combined detector good detections on pedestrians moving in any direction are true positives. Furthermore, some of the partial detections (detections that fall on just a part of the pedestrian) are merged with good detections and essentially get absorbed by the detection merging process.

A note about creating ROC curves. To generate the points of the ROC curve, the rejection thresholds for the detector must be adjusted to trade-off between correct detections and false positives. This is done by the following equation:

$$r_i = r_i - \mid r_i \mid * f$$

When f is less than 0, the rejection thresholds are increased (which decreases false positives and decreases the detection rate). When f is greater than 0, the rejection thresholds are decreased (which increases false positives and increases the detection rate). There are other reasonable methods for adjusting the rejection thresholds to generate the ROC curve, but we have not explored these here.

It is interesting to look at the first four features learned for each direction-specific pedestrian detector. These are shown in figure 4. Most of the detectors picked only motion-based filters among the top four filters. Only the north and the south detectors chose appearance-based filters instead of motion-based filters among the top four filters learned. This is probably due to the fact that when a person walks directly toward or away from the camera, the amount of motion is much smaller than when a person walks perpendicular to the camera view. Thus motion is not as strong of a cue for the north and south directions. It is also interesting to note that none of the first four filters of any detector use the ninth or tenth frame. Although the ninth and tenth frames do get used in later features, this does suggest that retraining with only seven or eight frames may result in very little loss in accuracy. Also, none of the first few filters chosen compare the first frame to the second which confirms our suspicion that the minimal amount of motion information used in [18] was far from optimal. Across all 8 detectors, 25% of the filters selected by AdaBoost training were appearance type filters, 56% were difference type filters and 19% were shifted difference type filters.

The test set included very small pedestrians as well as moving trees, bushes, cars, baby strollers, etc. These video sequences are intended to represent typical outdoor surveillance scenarios. We are not able to use standard test sets for comparison since the publicly available test sets are either static images or in the case of Dalal et al. [4], have a moving camera for which our method does not currently apply. Our method is undoubtedly much more accurate than the recent work based on static images. The typical pedestrians in our test set are too small to rely solely on the appearance information that static methods must rely on.

The running time to scan all 8 detectors over all scales of a 360×240 (using scale factor 0.75) is about 0.2 seconds or 5 frames per second on a Pentium 4 2.8 GHz computer (not including the time to load an image from disk or from a camera). Since the speed of the detector is directly related to the average number of features computed per window it is interesting to examine this number. On average the number of features computed per window for a single direction-specific detector is a little less than 4. Totaling over all eight detectors, the average number of features computed per window is 31.2. Some example detections are shown in figures 6 through 9 (only the first frame of the 10 frame sequence is shown).

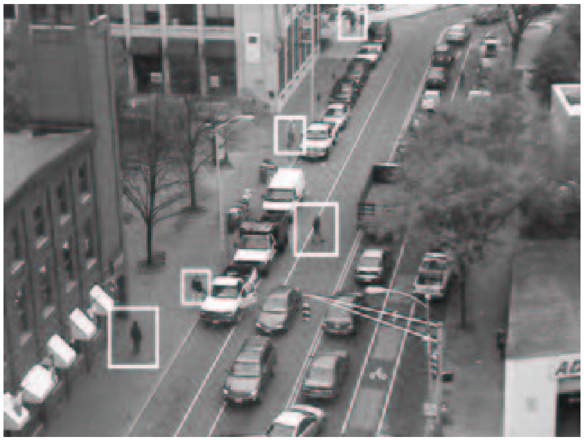

Figure 6. Scene with many moving cars

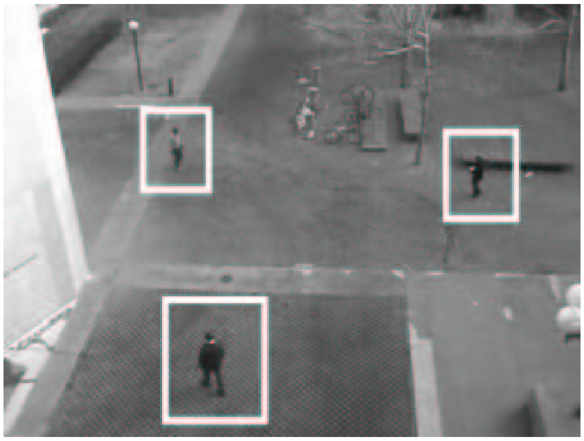

Figure 7. Pedestrians walking in various directions

8. Conclusions

We have presented an extension to the pedestrian detector of Viola, Jones and Snow [18] that uses many frames in a scanning window style detector. This extension allows much more sophisticated motion analysis than was possible in [18]. In addition, the detector is split into eight direction-specific detectors analogously to the way multi-view face detection is typically handled. The resulting detector is able to detect pedestrians in typical surveillance scenarios with a detection rate of about 93% with a false positive rate per window of 1 in a million which is an order of magnitude improvement over [18].

Most other work in human/pedestrian detection either

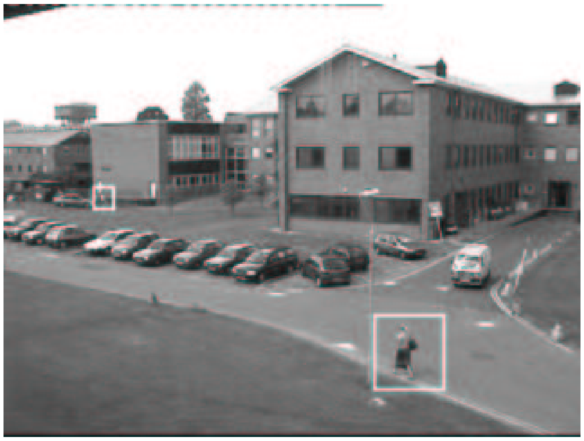

Figure 8. PETS 2001 test sequence

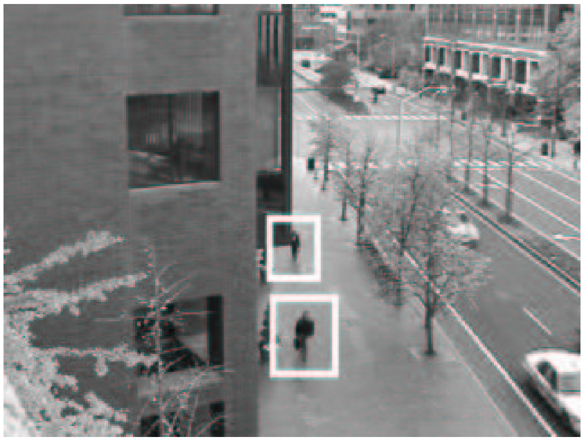

Figure 9. Scene with moving trees and cars

looks only at a single image or builds on top of a general tracker for video. Our approach does neither of these. It builds a scanning window type detector that acts directly on the pixels from a window in 10 consecutive frames of video. The improvement in accuracy we achieved shows the importance of motion information for detecting pedestrians in surveillance video in which the pedestrians are relatively low resolution.

References

[1] L. Bourdev and J. Brandt. Robust object detection via soft cascade. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages II:236–243, 2005. 1, 4

[2] R. Cutler and L. Davis. Robust real-time periodic motion detection: Analysis and applications. *IEEE Patt. Anal. Mach. Intell.*, 22:781–796, 2000. 2

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005. 1

[4] N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In *European Conference on Computer Vision*, pages 428–441, 2006. 1, 7

[5] P. . dataset. http://www.cvg.cs.rdg.ac.uk/pets2001/ pets2001-dataset.html. 6

[6] A. Efros, A. Berg, G. Mori, and J. Malik. Recognizing action at a distance. In *International Conference on Computer Vision*, pages 726–733, 2003. 2

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

[8] D. Gavrila, J. Giebel, and S. Munder. Vision-based pedestrian detection: The protector system. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2004. 1

[9] C. Huang, H. Ai, Y. Li, and S. Lao. High-performance rotation invariant multiview face detection. *IEEE Patt. Anal. Mach. Intell.*, 29(4):671–686, 2007. 1, 2

[10] M. Jones and P. Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab Technical Report TR-2003-96*, 2003. 1, 2

[11] B. Leibe, E. Seemann, and B. Schiele. Pedestrian detection in crowded scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 878–885, 2005. 1

[12] S. Li and Z. Zhang. Floatboost learning and statistical face detection, 2004. 1, 2

[13] A. Mohan, C. Papageorgiou, and T. Poggio. Example-based object detection in images by components. *IEEE Patt. Anal. Mach. Intell.*, 23:349–361, 2001. 1

[14] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *International Conference on Computer Vision*, 1998. 1

[15] R. Schapire and Y. Singer. Improving boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3), 1999. 2, 4

[16] O. Tuzel, F. Porikli, and P. Meer. Human detection via classification on riemannian manifolds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007. 1

[17] P. Viola and M. Jones. Robust real-time face detection. *Int. J. Computer Vision*, 57:137–154, 2004. 2, 3

[18] P. Viola, M. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. *Int. J. Computer Vision*, 63(2):153–161, 2005. 1, 3, 4, 6, 7

[19] Q. Zhu, M. Yeh, K. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1491–1498, 2006. 1