Detecting Pedestrians

Neil Alldrin
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093
nalldrin@cs.ucsd.edu

Abstract

Viola, Jones, and Snow recently implemented a pedestrian detection system that incorporates both appearance and motion in real-time. Simple sum-of-pixel filters are boosted into a robust pedestrian classifier. Detection is then achieved by thesholding a linear combination of these simple filters. The simplicity of the filters, along with some implementation tricks, enables the system to run in real-time. Motion information is incorporated by taking differences between successive frames in time.

This paper is a reimplementation of their system, with the purpose of evaluating the merits and pitfalls of their approach. I also discuss some issues that were inadequately explained in their paper, such as how to train features used in the cascade, and provide a performance comparison.

1 Introduction

The pedestrian detection system of Viola, Jones, and Snow [7] combines appearance and *motion* into their earlier work on object detection [6]. Their system successively applies simple features to windows in each frame. A pedestrian is "detected" if a linear combinations of feature responses exceeds a predetermined threshold. Each feature is a simple sum-of-pixels filter with an associated weight and threshold. For performance reasons, the features are applied sequentially in a cascade so that only regions of an image likely to contain a pedestrian are examined by later features. Since pedestrians tend to be sparse, only a few features need be evaluated for most image regions. The features, weights, and thresholds used during detection are learned with a variant of AdaBoost.

Motion is captured through difference images, which are simply differences between successive framesin time. Shifted versions of these difference images capture local translational motion information. Perhaps the biggest advantage of difference images is that they requires much less computation than other techniques for motion capture such as optical flow.

This paper is a reimplementation of the pedestrian detection system of Viola, Jones, and Snow. An overview of how the system works and is trained is presented, followed by a comparison to the original detection system.

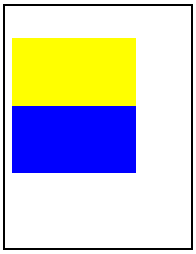

Figure 1: A sum-of-pixel filter. The response of the filter is the sum of the intensities of pixels within the light colored rectangle subtracted by the sum of the intensities of pixels in the dark colored rectangle.

2 Related Work

Using motion for recognition is far from a new idea. Recent work has been done on periodicity analysis [1] [4]. These systems, however, examine motion over long periods of time whereas the system presented in this paper only uses local motion information. Perhaps more related is work by Efros et al. that uses optical flow measurements to recognize human action [3]. While optical flow possibly provides more motion information than difference images, it is prohibitively expensive to compute and the representational gap between optical flow and difference images is unclear.

3 Data Structures

A variety of data stuctures are required for efficient implementation of the pedestrian detection system. Integral images enable fast sum-of-pixel filter evaluation, difference images capture local motion in a video sequence, and scaling effects are dealt with using pyramids of difference images.

3.1 Sum-of-pixel Filters

The detection system needs filters that fulfill two requirements: (1) they must be capable of distinguishing pedestrians from non-pedestrians with better than random probability and (2) their responses need to be extremely fast to compute. Sum-of-pixel filters do both.

See figure 1. A sum-of-pixel filter is composed of a set of oriented rectangles; the response is the sum of the intensities in the rectangles. Let I be an image and (x_1, y_1) and (x_2, y_2) be the upper left and lower right corners of a rectangle within the image. Then the response r of the rectangle is

$$r = \sum_{x=x1}^{x_2} \sum_{y=y1}^{y_2} I(x,y). \tag{1}$$

Each rectangle has an orientation o_i. Suppose a filter has rectangles with responses $r_1, r_2, ..., r_n$ and orientations $o_1, o_2, ..., o_n$, where $o_i \in \{-1, 1\}$. Then the response of the filter is

$$r_{filter} = \sum_{i=1}^{n} o_i r_i. \tag{2}$$

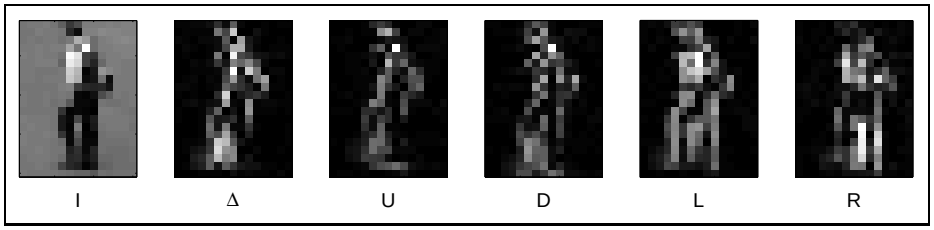

Figure 2: A sample difference image.

3.2 Integral Images

Integral images fascilitate fast computation of sum-of-pixel filters by enabling the sum of a rectangle to be computed in four memory operations. Each pixel in the integral image II is the sum of the pixels above and to the left of the corresponding pixel in the original image I. So, $II(x,y) = \sum_{x'=1}^{x} \sum_{y'=1}^{y} I(x',y')$. The sum of intensities in a rectangular region $\{(x_1,y_1),(x_2,y_2)\}$ is

$$r = II(x_1,y_1) - II(x_1,y_2) - II(x_2,y_1) + II(x_2,y_2).$$

Hence, the response of a rectangle is reduced from $M \times N$ operations to just four operations, where M and N are the width and height of the rectangle respectively. Dynamic programming could also be used to provide similar speedup, but makes implementation more complicated.

3.3 Motion

Motion is captured in difference images, which are just absolute differences between pairs of images in time. Five kinds of difference images are used in the detection system:

$$\begin{aligned} \Delta &= abs(I_t - I_{t+1}) \\ U &= abs(I_t - I_{t+1}\uparrow) \\ D &= abs(I_t - I_{t+1}\downarrow) \\ L &= abs(I_t - I_{t+1}\leftarrow) \\ R &= abs(I_t - I_{t+1}\rightarrow) \end{aligned}$$

where I_t is the image at time t and $\{\uparrow\downarrow\leftarrow\rightarrow\}$ are operations that shift an image one pixel up, down, left, and right respectively. Figure 2 shows a difference image used while training the classifier.

To account for scaling effects, a hierarchy of difference images at various scales is produced. These scaled difference images are arranged in levels in a pyramid structure. Each level l is constructed by first scaling the original input image by a scale factor s_l then constructing the difference images for that level,

$$\begin{aligned} \Delta^l &= abs(I_t - I_{t+1}) \\ U^l &= abs(I_t^l - I_{t+1}^l\uparrow) \\ D^l &= abs(I_t^l - I_{t+1}^l\downarrow) \\ L^l &= abs(I_t^l - I_{t+1}^l\leftarrow) \\ R^l &= abs(I_t^l - I_{t+1}^l\rightarrow) \end{aligned}$$

where I^l is I scaled by $s_l = 0.8^{l-1}$. This process ensures that each level of difference images are shifted in a scale-invariant way. Scaling stops when I^l is less than 20×15 pixels.

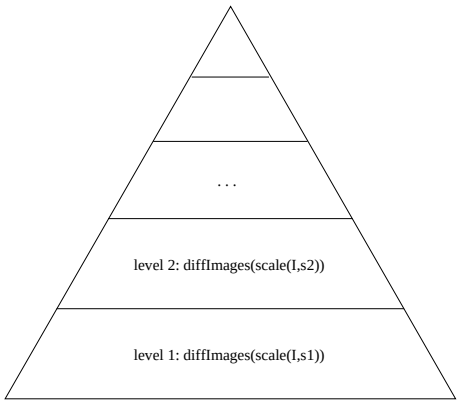

Figure 3: The pyramid datastructure. Each level in the pyramid contains six images $\{I^l, \Delta^l, U^l, D^l, L^l, \text{and } R^l\}$ and an associated scale factor s.

4 The Detection Algorithm

The pedestrian detection algorithm works by running a classifier on every 20×15 window in a pyramid. The classifier is a set of weak classifiers arranged in a cascade as shown in figure 4. The cascade improves computational efficiency by quickly eliminating windows which are unlikely to contain a pedestrian. Only windows that have similar characteristics to a pedestrian make it to the end of the cascade.

Each classifier in the cascade is a thresholded sum of weighted features and each feature is a thresholded sum-of-pixels filter. Let F be a feature with corresponding filter f and threshold th. Then the output of the filter is,

$$o(F) = \begin{cases} 1 & \text{if } response(f) \geq th \\ 0 & \text{otherwise} \end{cases}$$

where $response(f)$ is the response of f as defined in equation 2. The output of a classifier in the cascade is then

$$o(classifier) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i o(F_i) \geq th_{classifier} \\ 0 & \text{otherwise} \end{cases}$$

where $th_{classifier}$ is the classifier threshold and $\{F_1, F_2, ..., F_n\}$ are the features in the classifier with outputs $\{o(F_1), o(F_2), ..., o(F_n)\}$ and weights $\{w_1, w_2, ..., w_n\}$.

Four types of filters are used in the system: appearance filters, which operate on the original image I, and three types of motion filters, which operate on $\{\Delta, U, D, L, R\}$. The appearance filters are the same as used in Viola and Jones' earlier work [6]. In all, there are over 500,000 possible filters that operate in a 20×15 window across $\{I, \Delta, U, D, L, R\}$.

Four basic kinds of appearance filters are employed with two, three, four, and six rectangles respectively. Figure 5 shows some examples of appearance filters. The set of appearance filters includes all scales and orientations of the base appearance filters that fit inside a 20×15 window, over 100,000 filters in total.

The first type of motion filter compares sums of absolute differences between Δ and one of $\{U, D, L, R\}$,

$$f_a = r_a(\Delta) - r_a(S)$$

where $S \in \{U, D, L, R\}$ and $r_a()$ is a single box rectangular sum in the detection window.

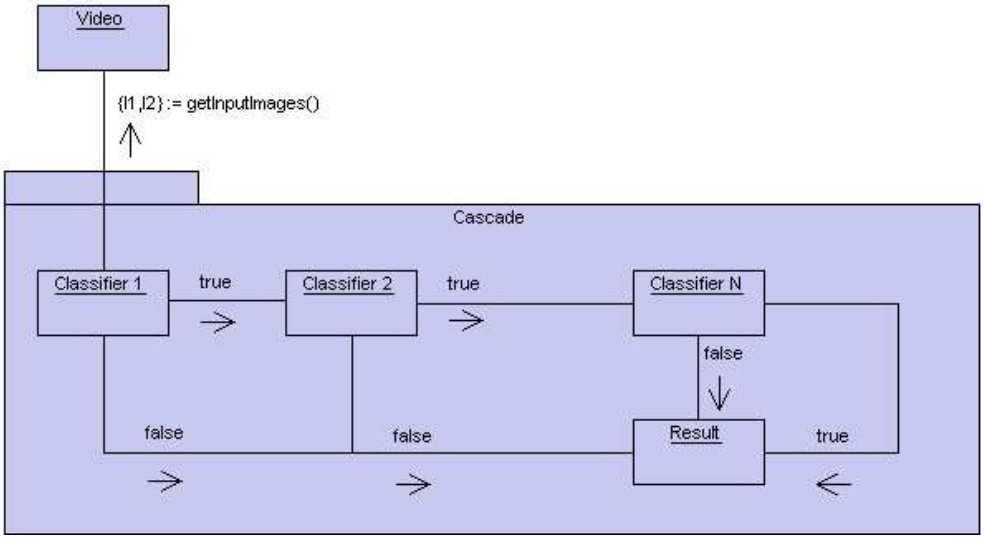

Figure 4: The cascade architecture. The input image is passed through a sequence of classifiers. For an image to be classified as a pedestrian it must pass through all classifiers. Classification stops immediately if any of the stages fail.

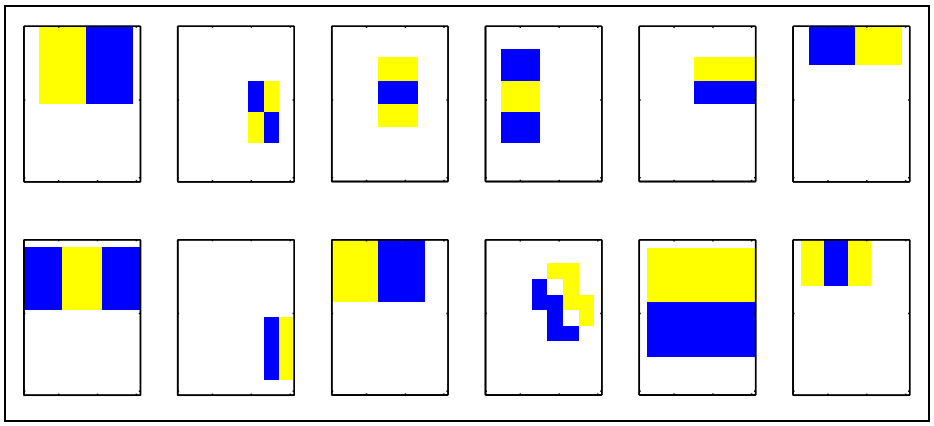

Figure 5: Sample appearance filters.

trainClassifiers(X,Y,$filterSet$) {
// $X = \{x_1, x_2, ..., x_n\}$, a set of example windows
// $Y = \{y_1, y_2, ..., y_n\}$ where $y_i \in \{0, 1\}$, labels for the windows

$m =$ number of negative examples
$l =$ number of positive examples
$W = \{w_1, w_2, ..., w_n\}$, where $w_i = \begin{cases} \frac{1}{2m} & \text{if } y_i = 0 \\ \frac{1}{2l} & \text{if } y_i = 0 \end{cases}$
For $t = 1, ..., T$

1. Normalize the weights, $w_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$
2. For each filter $f_j \in filterSet$, train a feature h_j by determining the optimal threshold for the filter.
3. Assign errors $\varepsilon_j = \sum_{i=1}^{n} w_i |h_j(x_i) - y_i|$
4. Select the classifier h_t with the lowest error ε_t
5. Update the weights, $w_i = w_i \beta^{e_i}$ where $e_i = 1$ if example x_i is correctly classified otherwise $e_i = 0$ and $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$

The final strong classifier is $h(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$
where $\alpha_t = \log(\frac{1}{\beta_t})$

}

Figure 6: Pseudo-code for the AdaBoost algorithm used to train the pedestrian classifiers.

The second type of motion filter is just the appearance filters applied to difference images,

$$f_b = \phi_b(S)$$

where $\phi_b()$ is one of the rectangle filters used for appearance.

The third type of motion filter measures the magnitude of motion in one of the difference images,

$$f_c = r_c(S)$$

where $r_c()$ is a single box rectangluar sum in the detection window.

5 Training

The classifiers used for detection are trained with a variant AdaBoost. Training involves selecting a subset of features, assigning weights to those features, and assigning a threshold to each classifier. This is achieved by treating features as weak classifiers and boosting them into a strong classifier.

Figure 6 shows the AdaBoost algorithm [5] [6]. While the full set of over 500,000 filters could be used for training, in practice it is sufficient to use a randomly sampled subset of the filters (50,000 were used in this paper). These filters are drawn from the appearance and motion filters described in section 4.

One point glossed over by Viola, Jones, and Snow is how to determine the optimal thresholds for a feature during training. Since AdaBoost does not consider all examples equally (it assigns weights to each example), a simple mean of the filter responses does not suffice. The solution used in this paper is based on univariate quadratic discriminant analysis

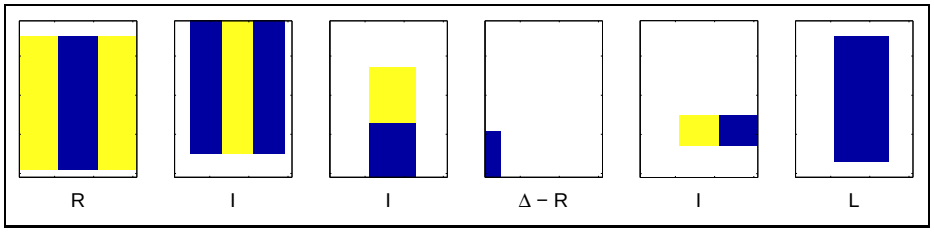

Figure 7: The first six features learned by AdaBoost and their associated image(s). It appears to be honing in primarily on vertical bars.

[2]. Univariate quadratic discriminant analysis assumes the filter responses for the positive and negative examples form Gaussian distributions and assigns weighted means and variances to these sets. From the weighted means and variances, a threshold can be chosen that approximately minimizes the error.

The thresholds chosen in this implementation are a simplification of this technique that assigns the feature threshold to the average of the weighted means of the negative and positive examples. Let $f(x)$ be the response of a filter to example x and $X_{neg} = \{x_1, x_2, ..., x_m\}$ be the set of negative examples with associated weights $W_{neg} = \{w_1, w_2, ..., w_i\}$. Then the weighted mean of the negative examples for filter f is $\mu_n = \frac{\sum_i w_i f(x_i)}{\sum_i w_i}$. If μ_p is similarly defined for the positive examples, then the threshold assigned to f is $\frac{1}{2}(\mu_n + \mu_p)$.

The cascade is formed by first training one large classifier (40 filters were in this paper) and then inserting thresholds at intermediate points within the classifier. Intermediate thresholds are chosen so that a significant portion of the negative examples are rejected while still passing most of the postive examples. These thresholds are manually selected, although this could be automated with a holdout set and target detection and false positive rates (as is done by Viola, Jones, and Snow).

6 Implementation

The detection system presented in this paper is written entirely in Matlab. Even after heavy optimization, detection still takes roughly three seconds per 384×288 frame on a 3.0GHz Xeon. To contrast, Viola, Jones, and Snow's implementation processed four frames per second on a roughly equivalent system. While they do not specify their implementation details, it seems likely they used a lower level language such as C or C++, which would explain the performance differential.

7 Results

Video sequences used for both training and testing were taken from the PETS2001 database (datasets 1,2 and 3). The classifier was trained on windows of successive frames extracted from the videos. For negative examples, windows were uniformly randomly selected across frames, window locations, and window scales. Windows containing a pedestrian were manually removed. Positive examples were collected by manually tracking pedestrians through video sequences. The bounding box was adjusted as necessary to keep the pedestrian properly scaled and in the center of the window. Approximately 4000 negative examples and 6000 positive examples (taken from six different pedestrians) were used during training.

The classifier was trained using these examples and 50,000 randomly selected filters from the total set of 500,000. Forty of these features were selected by AdaBoost along with cor-

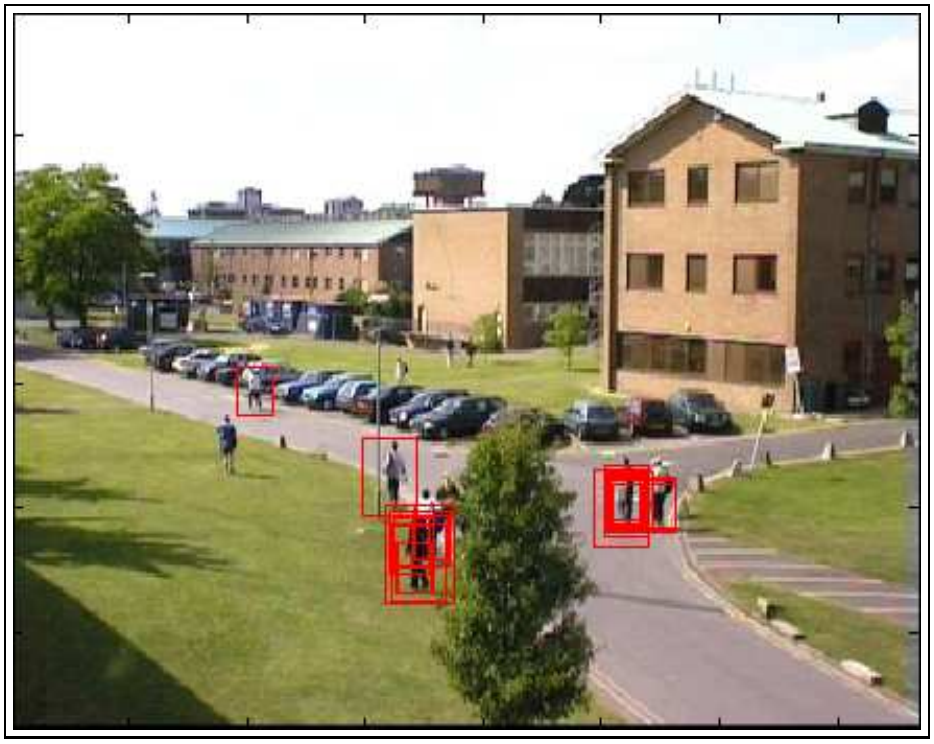

Figure 8: Results from the detection algorithm on the PETS2001 test dataset. Notice there is one false negative and no false positives in this particular image.

responding feature weights and a threshold. Cross-validation indicated an average 98.8 percent detection rate and a 1.2 percent false positive rate for unmodified classifiers returned by AdaBoost. Considering the number of windows per image this is an unacceptable result, but the inaccuracy most likely reflects problems with the training set since it was manually constructed. Figure 7 shows the first few features selected by AdaBoost.

One problem with the classifier returned by AdaBoost is that the threshold is much too low. This is because the prior probability for a pedestrian is much lower than the prior probability for a non-pedestrian, but the AdaBoost algorithm assumes they have equal priors. Attempts were made to adjust the threshold automatically based on a holdout set, but not enough negative examples were present to accurately do this. Instead, the threshold was manually adjusted until the false positive rate was qualitatively low enough. The resulting threshold yielded zero false positives, but also had a surprisingly low detection rate of about 9 percent on the holdout set. While the holdout set had a dismal detection rate, the performance on actual images was much better because many windows overlap each pedestrian, giving the classifier multiple chances to detect a given pedestrian. The thresholds for the cascade were manually chosen so that as many negative examples were rejected as possible while still allowing almost all positive examples though.

Figure 8 shows detected pedestrians in a frame from the PETS2001 test dataset. Notice there are no false positives and only one false negative. While this is slightly better than average, it is indicative of the classification performance of the detector on the PETS2001 dataset.

8 Comparison to Viola, Jones, and Snow

While the majority of my implementation is the same as that described by Viola, Jones, and Snow, there are a few areas of divergence. For example, while learning the optimal threshold for a feature, the orientation should also be learned, otherwise the best an inverted feature could do is classify all examples the same. The orientation was omitted for two reasons: (1) calculating the orientation for every feature response adds computational complexity, (2) the set of filters being trained on contains both orientations for all filters. Another difference between the two implementations is that my training set of examples is inferior. Viola, Jones, and Snow collected 16000 positive examples and about 20 million negative examples, compared to my 6000 positive and 5000 negative examples. Their examples are also drawn from a variety of sources, whereas mine are all taken from the PETS2001 training sets. Another difference is that I manually select thresholds where they automate this process to some degree.

9 Future Work

The boosted sum-of-pixel feature technique introduced by Viola and Jones has many potential uses. One such use would be to introduce it into a particle filter context where the sum-of-pixel classifier could be used to estimate a likelihood. This would enable an extremely simple parameterization of pixel coordinates, scale, and velocity. This would also increase the robustness of the Viola and Jones algorithm and make it faster since full image searches would no longer be necessary. The simplicity of the parameterization also would have a huge benefit over more complex contour based parameterizations. Another possible application of this algorithm would be for behavior classification. For this to work, however, longer-term motion analysis might be necessary necessary. Perhaps looking at N successive frames instead of 2 would improve classification performance.

10 Conclusions

The pedestrian detection algorithm introduced by Viola, Jones, and Snow is unique in that it incorporates both motion and appearance information in near real-time. My implementation was able to do a relatively good job despite training with an inadequate training set and requiring manually specified thresholds. However, detection rates were definitely below what some other techniques are capable of achieving. Again, the key advantage is speed. With a detection rate of three seconds per frame in an interpreted programming language, the algorithm is indeed very fast for what it accomplishes.

References

[1] CUTLER, R., AND DAVIS, L. S. Robust real-time periodic motion detection, analysis, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8 (2000), 781–796.

[2] DUDA, R. O., HART, P. E., AND STORK, D. G. *Pattern Classification*. John Wiley and Sons Inc., 2001.

[3] EFROS, A. A., BERG, A. C., MORI, G., AND MALIK, J. Recognizing action at a distance. pp. 726–733.

[4] HARITAOGLU, I., HARWOOD, D., AND DAVID, L. S. W4: Real-time surveillance of people and their activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8 (2000), 809–830.

[5] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *The Elements of Statistical Learning*. Springer, 2001. HAS t 01:1 1.Ex.

[6] VIOLA, P., AND JONES, M. Rapid object detection using a boosted cascade of simple features. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition* (2001).

[7] VIOLA, P., JONES, M. J., AND SNOW, D. Detecting pedestrians using patterns of motion and appearance. In *iccv03* (Nice, France, 2003), pp. 734–741.