Real-time Pedestrian Detection in Urban Scenarios

VARGA Robert, VESA Andreea Valeria, JEONG Pangyu, NEDEVSCHI Sergiu
Technical University of Cluj Napoca
{robert.varga, pangyu.jeong, sergiu.nedevschi}@cs.utcluj.ro, andreeavaleriavesa@gmail.com
Telephone: (800) 555–1212

***Abstract*—A real-time pedestrian detection system is presented that runs at 24 fps on standard VGA resolution input images (640x480px) using only CPU processing. The detection algorithm uses a variable sized sliding window and intelligent simplifications such as a sparse scale space and fast candidate selection to obtain the desired speed. Details are provided about the initial version of the system ported on a mobile device. We also present a new labeled pedestrian dataset that was captured from a moving car that is suitable for training and testing pedestrian detection methods in urban scenarios.**

Keywords*—*Pedestrian detection; object recognition; region of interest selection; mobile devices

I. Introduction

As we move closer and closer to a fully autonomous car with collision detection, lane departure warning, adaptive cruise control, autonomous emergency braking, pedestrian detection seems to be the next big wave in offering advanced modern car safety features. Early detection could help reduce the number of accidents by heightening the awareness of the driver in time. In recent years there has been a dramatic increase in the number of people using a smartphone or a tablet as part of their everyday life, confirming the demand for having a fast and accurate detection system integrated with mobile devices. With such an increase and a technology that is constantly evolving, bringing faster and faster processors on the market, the challenge for building a pedestrian detection system on a mobile device proves to be a very good subject open for research. Even though well-established methods running on CPU and specialized hardware exist there is still a long way from fast and accurate detection on portable devices.

II. Related work

This section provides references to important contributions for pedestrian detection. For a comprehensive review on the subject the reader is advised to consult reviews and surveys such as [1], [2], [3].

It is essential for pedestrian detectors to rely on features that help discriminate objects from the background. Histogram of Oriented Gradients[4] is a well established feature for pedestrian detection. Other relevant features are: dense SIFT[5], [6], Histogram of Flow[7], Color self similarity[8], Co-occurrence Histogram of Oriented Gradients[9], Shapelet [10], Local Binary Patterns [11]. Optimizing calculation with integral images has enabled fast calculation of HOG and similar features [12], [13], [14]. Contour based features can also provide useful information [15], [16]. Another crucial ingredient is a fast classifier. In this field boosted ensemble classifiers [17], [18], [19], [20], [21], [22] dominate along with fast SVMs such as linear SVM or Histogram Intersection Kernel SVM [23].

Most top performing modern detection methods rely on integral features and perform detection using sliding windows [1], [20], [24], [25]. Lightning fast methods do exist but they rely on either specialized hardware (GPU[26], [27], [22] or FPGA [28]) or on stereo information (depth) [21], [29]. Another alternative would be a part-based detector [30].

Most of the implementations using ARM architecture are targeted at the embedded market. A couple of such embedded systems for automotive applications that posses an integrated pedestrian detection capability are: Mobileye Pedestrian Collision Warning (PCW) system, Toyota Pedestrian-Avoidance Steer Assist or Volvos Pedestrian Detection With Full Auto Brake system. However, there is no extensive work so far regarding a standalone Android application capable of performing real time pedestrian detection. Our aim is to offer a starting point in building such a solution.

III. Proposed approach

The detection algorithm for the current system follows our previous work from [31]. Here, we only provide a short overview and state the differences. The main focus of this paper is to show how to achieve high-speed detection while maintaining detection performance as much as possible.

One of the key ideas is to detect pedestrians with only a reduced number of heights. Usually detecting different heights entails resizing the image multiple times (constructing the image pyramid). This means that the number of scales is in one-to-one correspondence to the number of pedestrian heights. Our approach is opposed to the mainstream idea of using a dense scale space for image pyramid construction. Each image from the image pyramid is a rescaled version of the input image and has and associated detector of a given height. Typical detectors construct an image pyramid with up to 50-55 scales. The heights chosen here are based on statistical data acquired from the training dataset. We apply k-means clustering to obtain the 6 representative cluster centers for pedestrian bounding box heights. The number 6 was chosen because the detection rate was acceptable. A future analysis about how the detection performance is influenced by the number of heights will be performed. The difference compared to other sparse scale space methods (i.e. methods that limit the number pedestrian heights to consider) is that we do not have detection windows that are of the form $a2^n$, instead we select the representative centers based on the training data. Since detection performance degrades significantly at lower scales we may omit smaller window heights for practical applications.

The proposed detection method uses a sliding window approach with the before mentioned 6 fixed window heights and a constant aspect ratio. Considering an exhaustive search at every position, scale and aspect ratio is not feasible and also not necessary. In this work we opt for a candidate generation that accepts only candidate rectangles that have their center in the horizontal middle stripe. This selection can be motivated by studying the spatial distribution of detection window centers from the training dataset. This distribution has been observed on other datasets such as the Caltech Pedestrians [1].

Each sliding window height corresponds to a pedestrian at a specific scale and has an associated classifier. The classifiers are trained separately for each scale. Our aim is to totally eliminate image resizing and other operations on features. This is a key difference compared to other methods: Dollar et al. adjust the features based on scale in [20], while Benenson et al. [21] adjust the classifier. The detection window is moved to every valid position that is dictated by a region of interest selection method. Features are then calculated as sums of rectangular subregions for each candidate window and classification is performed. The sums of different image channels over a rectangular area can be calculated efficiently with integral images. The underlying image channels are Luv color-space, gradient magnitude and oriented gradient histogram with 6 bins. For feature extraction we rely on a module provided by Dollar [32].

Key elements of this approach that help achieve high-speed and reliable detection are:

- No image resizing
- Smart and fast region of interest selection (candidate generation)
- Fast integral channel features calculation
- A cascade of boosted decision trees for classification
- Reduced number of pedestrian heights
- Custom implementation of all processing modules and code parallelization

IV. Implementation details

The reduced execution time of the algorithm is due to the fact that almost all processing steps have been specifically rewritten for the detection task. Our implementation is in C++, compiled with Visual Studio 2010 compiler with OpenMP multithreading features enabled. Other settings include: fast code optimization enabled, fast floating point model, omit frame pointers. OpenCV 2.4.5 is the chosen library for image processing functions.

The workstation used to test our system has the following parameters: Intel Core i7 CPU, 3.5 GHz, 4 cores, 8 logical processors, 16 GB RAM. Most of the relevant operations are parallelized to use the processing power of the CPU efficiently. Speed measurements are provided in the Experimental Results section VI.

A. Detection algorithm

The detection algorithm follows the well established pipeline format having the following steps: preprocessing (resizing - not employed in this case, padding - when the image width is not divisible by 4, smoothing), region of interest selection (or candidate generation), feature channels extraction (Luv conversion, gradient computation, histogram bin aggregation), feature aggregation (rectangular region sums) only on the candidate regions, classification/prediction and non-maximum suppression.

An important aspect of the preprocessing step is how to treat images that do not have width divisible by 4. Since many operations run only if divisibility is ensured we pad the images instead of cropping or resizing. This also helps to linearize the image in the memory. Note, that during the training phase many cropped small images are fed as input to the feature extractor. These result from clipping out only the pedestrian bounding box. This is why it is important to treat irregular sized images carefully. We perform no image resizing, although for larger input images this could be included to reduce the search space. Image smoothing is moved to the feature extraction phase and gaussian bluring is replaced by a faster triangular filtering as in [13].

Region of interest selection provides the candidate rectangles from which the features are extracted and classified. It is essential to restrict the number of these candidates to reduce the workload of the following modules from the pipeline. For this step we have three main options. The first option is to use all possible bounding boxes with a given stride, fixed aspect ratio and height restricted to a set of values. The second option is to admit only the rectangles whose centers lie in the central horizontal stripe of the image. This is a heuristic that is easy to implement and it is deduced from the measurements from the pedestrian dataset (see section V). The third option is to select candidate regions based on the edges in the image (as in [31]). Here we opt for the second approach because it is sufficient and assures high coverage. The selected pedestrian heights are: 60, 92, 136, 212, 340 pixels. The stride is set to 4 or 8 pixels, the fixed aspect ratio is 0.43 (width over height).

For feature extraction we rely on the Integral Channel Features module provided by Dollar [32]. This was adapted from Matlab+mex to our C++ implementation that uses classes from the OpenCV 2.4.5 library. Feature extraction is fast because of clever usage of integral images, parallel computing of the channels and SSE instructions. The present module has the standard configuration of channels: luv, gradient magnitude and 6 gradient orientation bins. Parameters have been set to: 5000 random features with an area of at least 25; shrinking factor of 4, triangular smoothing which is equivalent to a Gaussian blur with a sigma of 1. See [13] for more details about the parameters.

Classification is performed with an ensemble of 5000 weak learners. Both the training and predicting have been reimplemented to optimize prediction speed. Discrete boosting is applied with two level decision trees. This option is motivated by [20], where the authors show that the boosting method does not have a large impact on the detection rate and that 2 level decision trees are the best for this task. The splitting criterion for the decision tree is the local training error, i.e. the best split is the one that minimizes the training error. Rejection thresholds for the cascade classifier are obtained via the direct backward pruning method [33] on the training set. In most cases it is preferred to obtain the thresholds on a validation set rather than the training set. When this validation is not performed the rejection thresholds should be lowered in order to prevent the quick rejection of unseen positive examples. A simple recalculation of the thresholds can be performed in order to obtain rejection thresholds for any end threshold (see [33]).

At detection time all rectangles obtaining a classification score higher than a given threshold θ) are retained for non-maximum suppression. For every two overlapping rectangles we retain only the one with the higher score. The overlap can be determined in multiple ways. Here, we use the formula: $o_{min} = \frac{R_1 \cap R_2}{min(R_1, R_2)}$, where R_1 and R_2 are the areas of the two rectangles, and the numerator contains the area of their intersection. This eliminates smaller bounding boxes from inside larger ones because in this case the overlap is high due to the min function from the numerator. For the same reason it more aggressive than the usual alternative: the PASCAL VOC-type overlap measure $o = \frac{R_1 \cap R_2}{R_1 \cup R_2}$. This is why lower thresholds are suitable for this method. Another alternative useful for large number of detection windows is to perform a greedy elimination. The threshold for considering overlap is set to 0.4.

B. Training procedure

At the training phase we repeat the same operations at each scale to generate classifier models. We first process the positive examples. Each bounding box of a person is cropped from the original image and resized with bilinear interpolation to the size of the current detection window (e.g. 60x26px). Adjustments are mare to center to the bounding box and to preserve the original aspect ratio (resizing factor is the same along the width and the height). To increase the diversity we also process the horizontally mirrored image. The 5000 features are calculated and saved for later with a positive label. These features are randomly generated rectangles from inside the detection window area. To obtain negative samples we select 5000-7000 random crops from the list of images not containing any pedestrians. A uniform sampling is performed, i.e. if there are 5000 images, then one random rectangle is chosen from each, if there are 20000 images (a typical case), then one random rectangle is chosen from every 4th. For every random window we calculate the features and save them for later with a negative label. Once all examples are processed the file containing the features can be fed as input to train the classifier.

Next, we perform bootstrapping. Call the initial model as model-x-0, where x stands for the height and 0 stands for no bootstrapping. By applying the classifier on negative images we obtain at first many false positives. At each stage we retain 7000 of these false positives and append their feature vectors to the training file. After retraining the classifier model-x-1 is obtained. This process is repeated 2-3 times until there are only a few false positives or no change is observed. Limiting the number of examples is essential to keep the training set balanced and also helps reduce redundancy. We have observed that the classifier for the smallest scale produces false positives even after 4 rounds of bootstrapping. This is a clear signal indicating the failure of the classifier to learn a good model from the training data.

TABLE I: Relevant parameters of the detection algorithm

Parameter	Description	Value
N	number of weak learners	1000
M	number of features	5000
d	stride (grid step)	8
ρ	aspect ratio	0.43
h	pedestrian heights	{60, 92, 136, 212, 340}
T_o	overlap threshold for NMS	0.4
B	additional bootstrap samples at each stage	7000

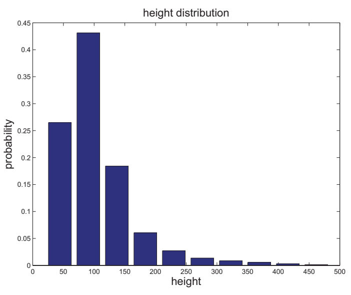
C. Implementation for Android mobile devices

In this section we present some details about how the presented approach can be implemented on a mobile device such as a tabled or smartphone. Having the algorithm already implemented in C++, developing an Android application seemed a very good solution for us since we could easily integrate the existing native code with Java code by making use of Java Native Interface (JNI) framework. The algorithm was ported on an Android mobile device having the following characteristics: NVIDIA Tegra 3 T30L chipset, Quad-core 1.2 GHz ARM Cortex-A9 CPU, NEON instruction set support. We have used JNI calls in order to capture the frames in a Java environment and send them for a faster processing in a native C++ environment. For a better performance we have used OpenCV 2.4.5 for Tegra 3 library accelerated for ARM NEON architectures and we took benefit from the multi-core processor by parallelizing the code with Qualcomm MARE Parallel Computing Library. We took advantage of the **pfor_each** functionality provided by MARE in order to substitute the **#pragma omp** parallel functionality provided by OpenMP used in our PC version. For handling the reading and writing of the training files we have used the AssetManager class provided by Java. This allows us to compress the files and perform one time writing in the internal memory of the portable device upon installation of the application and extract the data whenever we need during running the algorithm.

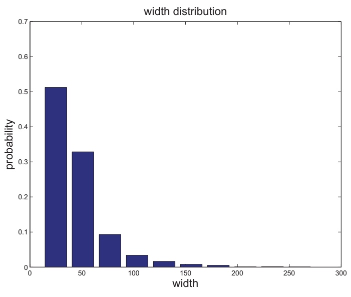
V. CLUJ PEDESTRIANS DATASET

A labeled pedestrian dataset was gathered for the purpose of training and evaluation (see Figure 3). This set was captured using a smartphone placed on the windshield of a vehicle driving through the city. The purpose of this new dataset is to reproduce as closely as possible the real situation where the detection method would be applied. The dataset is available in both video (mp4) and image (png) format. Total video length is around 15 minutes. The video framerate is of 30fps. All images have 640x480px resolution (landscape). The total number of images is 27666. The set is broken into independent sequences based on separate recordings and contains mostly frames with pedestrians. We provide the pedestrian bounding boxes for each image in the dataset in a simple text file format. Note, that some pedestrians are unlabeled. This is the case only for very crowded scenes and for persons that are far away and thus appear to be very small. There can be up to 10 pedestrians in a single frame.

The pedestrian bounding boxes are divided into a non-overlapping training set and test set. The initial training set

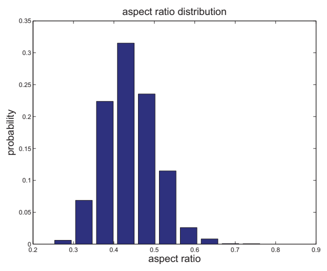


(a) height distribution

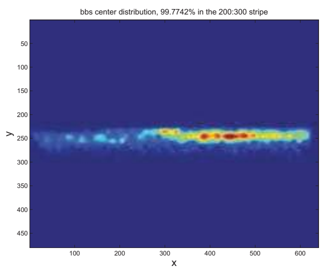


(b) width distribution

Fig. 1: Bounding box dimension statistics for the Cluj-pedestrians dataset



(a) aspectratios



(b) width distribution

Fig. 2: Bounding box aspect ratio and center frequency for Cluj Pedestrians

Fig. 3: Sample images from the Cluj Pedestrians dataset

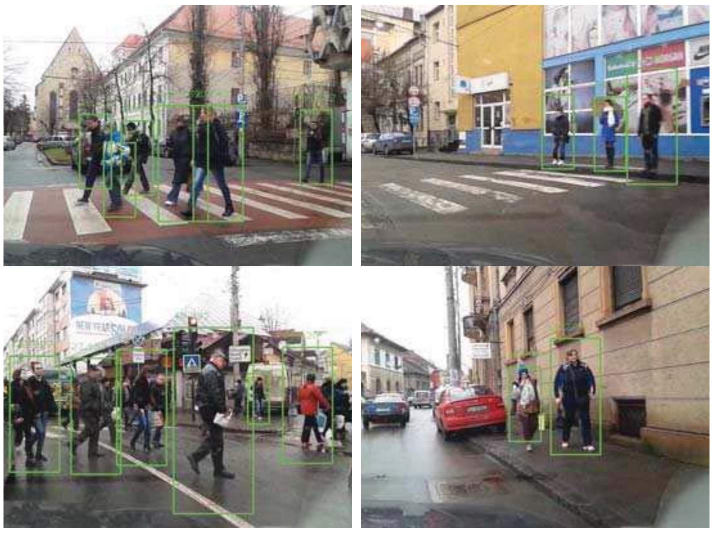


Fig. 4: Sample detections from Cluj Pedestrians dataset

contains 3205 bounding boxes with pedestrians and 219 negative examples (images with no persons) obtained by labeling every 10th frame. In order to enrich the dataset we have interpolated the detections and have generated an interpolated training set consisting of 27318 pedestrian rectangles. Due to the reduced number of negative examples it is recommended to augment the negative training set (images not containing any pedestrians) from another source (e.g. INRIA, Caltech or general images). For our training procedures we have augmented the examples from this dataset with a portion of the images (around 4500) from the Caltech Pedestrian Dataset [34] that do not contain any pedestrians.

We provide some interesting and relevant statistics for the dataset. The distribution of the heights of the bounding boxes is given in Figure 1. Approximately 88% of the bounding boxes are smaller then 140px in height, while the predominant height is 90px. The width of the bounding boxes follows an exponential distribution shown in Figure 1. The most likely aspect ratio is 0.43, see Figure 2. An important statistic about the frequency of the centers of the bounding boxes reveals that only a negligible part (0.3%) of them are positioned outside the 200-300 horizontal stripe (see Figure 2). Also, due to the placement of the recording device and because the pavement is on the right side, pedestrians occur more frequently on the right side.

VI. Experimental results

To evaluate our detection method we perform tests on two datasets: the well established INRIA benchmark and the newly acquired Cluj Pedestrians dataset. We directly compare our method to FPDW [1] [32] because it is based on it and also the source code and the detection module is available. The evaluation entails generating predictions with a low accepting threshold and generating the DET curve (detection error trade-off) by accepting detections at different thresholds. A predicted bounding box is correct if it overlaps with a ground truth box by at least 20%. The reason for the lower overlap threshold is to take into consideration the possible scale differences. The overlap is the usual Pascal VOC fraction: $\frac{R_1 \cap R_2}{R_1 \cup R_2}$, where the numerator is the intersection and the denominator is the union of the two rectangles in question. A point on the DET curve represents a fixed threshold for classification and corresponds to a miss rate and a false positives per image. This operation point can be chosen depending on the application and its requirements. In order to directly compare methods the area under the DET curve up to the one false positives per image line is calculated. A lower area means better detection performance.

First, we report on the results on the INRIA dataset. For this dataset we select the following 6 bounding box heights: {124, 204, 288, 388, 516, 664}, which were estimated as cluster

[1]http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html

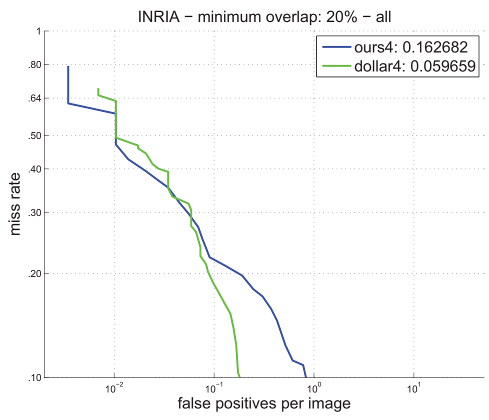


Fig. 5: DET curve on the INRIA Pedestrian dataset

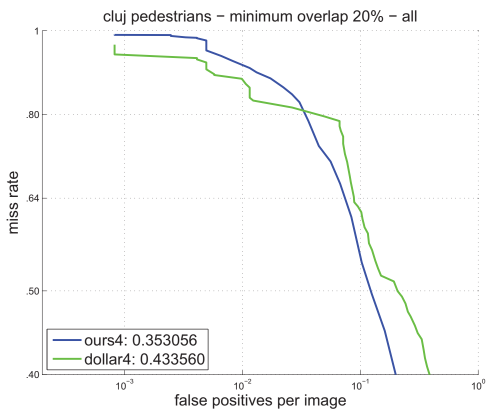


Fig. 6: DET curve on the Cluj Pedestrians dataset

centers from the training data. The DET curve is presented in Figure 5, where we compare our method with FPDW. Even though the behavior for our algorithm is similar up the 0.1 fppi line the overall performance of the FPDW is better - 0.059 compared to 0.16 area under the DET curve. This can be motivated by the fact that we use only a limited number of scales to perform detections and other simplifications.

We also evaluate both methods on the Cluj Pedestrians dataset (see Figure 4 for sample detections). For this dataset we select the following 5 bounding box heights: {60, 92, 136, 212, 340}. Note that the pedestrian heights for the INRIA dataset have larger values compared to those for the Cluj Pedestrians dataset. This dataset is much harder because it contains crowded scenes and small pedestrians. The bounding boxes for FPDW were generated using the code provided by the authors. The same evaluation code was used as in the case of the INRIA dataset. The DET curve is presented in Figure 6. In this case our method performs better at almost every point of the DET curve and in overall - 0.35 in comparison with 0.43 area under the DET curve. This can be explained by a more similar training set to the test set for our method. We have performed a grid search for optimal parameters. The parameters that can be changed after the classifier models are generated are: the stride (spatial grid step size); the overlap threshold for non-maximum suppression; the number of pedestrian heights to consider. We provide some representative results in Table II. Minimal areas under the DET curve are obtained at stride 4 and at overlap threshold of 0.4 (the criterion of overlap is o_{min}). The number of bootstrap rounds is set to 4, but for larger scales the process converges at rounds 2-3 and the classifier does not generate any false positives on the training set.

To evaluate the execution time of each part of the system we measure at least 1000 times the speed of each module. Table III summarizes the measurements that we have obtained. The bottleneck of the pipeline is the feature extraction module where heavy optimization has been performed, even so it is the slowest part. A further reduction of the number of the candidates could help drastically improve the speed. The number of candidates is around 2000-3000 depending on the scales. The estimated execution time (disregarding the visualization part) is 23.67 frames per second (around 42 milliseconds per frame). We compare the speed gain with our previous version

TABLE II: Area under the DET curve for different parameters on the Cluj Pedestrians dataset

area	**stride = 8**	**stride = 4**	**stride = 3**
overlap = 0.40	0.361	0.353	0.367
overlap = 0.65	0.373	0.367	0.375
overlap = 0.70	0.384	0.372	0.379

TABLE III: Execution time of different modules

Step	**PC [ms]**	**Android [ms]**	**Android + MARE [ms]**
RoI selection	0.131	0.29	0.29
Feature Extraction	38.3	1491.29	739.53
Classification	3.38	230.60	78.73
NMS	0.096	0.13	0.13

from [31]: 180 milliseconds on an image with a resolution of 370x480px would result in approximately 311ms running time on a 640x480px image, thus the speed gain is: $\frac{311}{42} = 7.4$. Time measurements on Android for a 640x480px resolution indicate a 1.21 FPS average frame rate (around 823 milliseconds per frame). This is the parallelized Android version which has an overall speed gain compared to the single threaded version of: $\frac{1781.89}{823.0} = 2.16$ ($\frac{1491.29}{739.53} = 2.01$ - feature extraction; $\frac{230.60}{78.73} = 2.92$ - classification).

VII. Conclusion

In this paper we have presented a real-time PC implementation of a pedestrian detection algorithm able to run on Android mobile devices. The key idea is to use the sliding window detection approach on a restricted zone of the image. We employ only 4-6 scales for detection which reduces the execution time drastically. Even with this sparse scale space the detections are acceptable. The scales are chosen based on statistical information gathered from the training set. For each scale we train a separate classifier. By evaluating our method on two datasets we have shown that the method is competitive.

Future work will focus on generating candidate regions based on methods that rely on segmentation (such as Objectness and Selective Search). Although there are many existing approaches with high recall none of them can provide object locations within milliseconds which is a necessity for both fast

PC and Android applications. We will also study the effect of a sparse scale space on the detection accuracy and how many scales are really necessary for detection. Further optimizations will be performed on the Android version especially in the feature extraction part.

Acknowledgment

This research was funded by the CoMoSeF project, code PN II PCCA PN-II-PT-PCCA-2011-3.2-0742 from 20.09.2012 (2012-2016).

References

[1] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 34, no. 4, pp. 743–761, 2012.

[2] M. Enzweiler and D. M. Gavrila, "Monocular pedestrian detection: Survey and experiments," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2179–2195, Dec. 2009.

[3] D. Gerónimo, A. M. López, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 32, no. 7, pp. 1239–1258, 2010.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR*, 2005, pp. I: 886–893.

[5] D. G. Lowe, "Object recognition from local scale-invariant features," in *ICCV*, 1999, pp. 1150–1157.

[6] B. Fulkerson, A. Vedaldi, and S. Soatto, "Localizing objects with smart dictionaries," in *ECCV*, 2008, pp. I: 179–192.

[7] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *ECCV*, 2006, pp. II: 428–441.

[8] S. Walk, N. Majer, K. Schindler, and B. Schiele, "New features and insights for pedestrian detection," in *CVPR*. IEEE, 2010, pp. 1030–1037.

[9] T. Watanabe, S. Ito, and K. Yokoi, "Co-occurrence histograms of oriented gradients for pedestrian detection," in *PSIVT*, 2009, pp. 37–47.

[10] P. Sabzmeydani and G. Mori, "Detecting pedestrians by learning shapelet features," in *CVPR*, 2007, pp. 1–8.

[11] Y. Cao, S. Pranata, and H. Nishimura, "Local binary pattern features for pedestrian detection at night/dark environment," in *ICIP*, B. Macq and P. Schelkens, Eds. IEEE, 2011, pp. 2053–2056.

[12] Q. A. Zhu, M. C. Yeh, K. T. Cheng, and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," in *CVPR*, 2006, pp. II: 1491–1498.

[13] P. Dollar, Z. W. Tu, P. Perona, and S. Belongie, "Integral channel features," in *BMVC*, 2009.

[14] F. M. Porikli, "Integral histogram: A fast way to extract histograms in cartesian spaces," in *CVPR*, 2005, pp. I: 829–836.

[15] D. M. Gavrila and S. Munder, "Multi-cue pedestrian detection and tracking from a moving vehicle," *International Journal of Computer Vision*, vol. 73, no. 1, pp. 41–59, Jun. 2007.

[16] J. J. Lim, C. L. Zitnick, and P. Dollár, "Sketch tokens: A learned mid-level representation for contour and object detection," in *CVPR*. IEEE, 2013, pp. 3158–3165.

[17] J. Friedman, T. Hastie, and R. Tibshirani, "Special invited paper. additive logistic regression: A statistical view of boosting," *The Annals of Statistics*, vol. 28, no. 2, pp. 337–374, 2000.

[18] L. Bourdev and J. Brandt, "Robust object detection via soft cascade," in *CVPR*, 2005, pp. II: 236–243.

[19] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," *International Journal of Computer Vision*, vol. 63, no. 2, pp. 153–161, Jul. 2005.

[20] P. Dollár, S. Belongie, and P. Perona, "The fastest pedestrian detector in the west," in *BMVC*, F. Labrosse, R. Zwiggelaar, Y. Liu, and B. Tiddeman, Eds. British Machine Vision Association, 2010, pp. 1–11.

[21] R. Benenson, M. Mathias, R. Timofte, and L. J. V. Gool, "Pedestrian detection at 100 frames per second," in *CVPR*. IEEE, 2012, pp. 2903–2910.

[22] R. Benenson, M. Mathias, T. Tuytelaars, and L. J. V. Gool, "Seeking the strongest rigid detector," in *CVPR*. IEEE, 2013, pp. 3666–3673.

[23] S. Maji, A. C. Berg, and J. Malik, "Classification using intersection kernel support vector machines is efficient," in *CVPR*, 2008, pp. 1–8.

[24] P. Dollár, R. Appel, and W. Kienzle, "Crosstalk cascades for frame-rate pedestrian detection," in *ECCV*, 2012.

[25] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *PAMI*, 2014.

[26] T. Machida and T. Naito, "GPU & CPU cooperative accelerated pedestrian and vehicle detection," in *ICCV Workshops*. IEEE, 2011, pp. 506–513.

[27] C. Wojek, G. Dorkó, A. Schulz, and B. Schiele, "Sliding-windows for rapid object class localization: A parallel technique," in *Pattern Recognition (DAGM)*, May 2008.

[28] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "FPGA-based real-time pedestrian detection on high-resolution images," in *CVPR Workshops*. IEEE, 2013, pp. 629–635.

[29] A. Ess, K. Schindler, B. Leibe, and L. van Gool, "Robust multi-person tracking from moving platforms," in *Logic and Probability for Scene Interpretation*, ser. Dagstuhl Seminar Proceedings, A. G. Cohn, D. C. Hogg, R. Möller, and B. Neumann, Eds., no. 08091. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

[30] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 32, no. 9, pp. 1627–1645, 2010.

[31] R. Varga and S. Nedevschi, "Gradient-based region of interest selection for faster pedestrian detection," in *Intelligent Computer Communication and Processing*. IEEE, 2013, pp. 1–2.

[32] P. Dollár, "Piotr's Image and Video Matlab Toolbox (PMT)," http://vision.ucsd.edu/∼pdollar/toolbox/doc/index.html.

[33] P. A. Viola, J. C. Platt, and C. Zhang, "Multiple instance boosting for object detection," in *NIPS*, 2005.

[34] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *CVPR*, 2009, pp. 304–311.