16623 Advanced Computer Vision Apps Final Project Report
Pedestrian Detection based on iOS Devices

Xingchen Yu
Robotics Institute, school of Computer Science,
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
xingche1@andrew.cmu.edu

Yuzhang Wu
Robotics Institute, school of Computer Science,
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
yuzhang2@andrew.cmu.edu

Abstract

This paper addresses the implementation and methods of pedestrian detection we used, couple of experiments based on these methodologies, and optimization of results based on iOS devices. Pedestrian detection is a key problem in computer vision, with several applications that have the potential to positively impact quality of life and pedestrian detection was developed rapidly during last decades. HAAR-like, HOG and LBP feature are three important features for object detection. INRIA and CVC are two of outstanding pedestrian databases which used in many famous experiments of pedestrian detection. We utilize these three features and these databases to train our pedestrian classifiers and analyze images and videos under OpenCV and Armadillo. We evaluate results using FPS(frame per second). Our experiments and implementations show that performance is improved a lot by using GPU and image preprocessing. In particular, detection is disappointing at low resolutions and for partially occluded pedestrians.

Keywords—pedestrian detection; computer vision; HAAR; HOG; LBP; classifier; OpenCV; Armadillo; FPS; GPU;

1. Introduction

Pedestrian detection based on iOS devices is defined as: check pedestrians in the real-time video frames, if there is, give location information. It is the first step of vehicle auxiliary driving, intelligent video surveillance and human behavior analysis. In recent years, pedestrian detection is also used in the emerging fields, such as aerial images, victim rescue. Pedestrians have both rigid and flexible object characteristics, and the appearance is easy to be influenced by wearing, scale, shielding, attitude and angle of view, which makes it difficult and hot spot to develop pedestrian detection [1].

1.1. Challenges

a. Wide variety of articulated poses
b. Variable appearance/clothing
c. Complex backgrounds
d. Unconstrained illumination
e. Occlusions, different scales

1.2. Approaches

There are many computer vision approaches for detecting pedestrians. We evaluate detectors using Haar-like, Hog and LBP features. Under the environment of OpenCV, we need to collect positive and negative images, train cascade classifier Adaboost training method, detect and localize upright people in video frames.

1.3. Optimazation

Without any optimized algorithms, we may detect multiple bounding boxes surrounding the pedestrian in the video frames. Non-maximum suppression is about to reduce this number to one bounding box, we can improve the accuracy of detection by move the overlapped detection boxes.

Apart from accuracy problems, when implement the algorithm and classifiers on iOS platforms, there are still some issues to overcome due to the limitation of computing. Traditional method of using CPU and OpenCV often cost a lot and the computing speed is not fast. So we try to implement our method in GPU and Armadillo which could accelerate the computing speed and cost lest since the hardware base is not the same.

2. Background

Pedestrian detection has been a central topic in computer vision research, spanning more than 20 years of research [6, 15]. A wide variety of methods have been applied to pedestrian detection over the years, with continued improvement in performance [4, 5, 6, 7, 9, 10, 12, 13, 18]. Some methods focus on improving the base features used [6, 9, 10, 15], whereas others focus on the learning algorithms [13, 18], or other techniques such as incorporating Deformable Parts Models [13, 14] or using context [13, 14, 16, 17]. Dollar et al. [12, 14] developed a benchmark and evaluation toolbox that has been instrumental in tracking progress in the field. Benenson et al. [6] have recently proposed a comparative paper that evaluates performance of various features and methods on

pedestrian detection. Viola and Jones proposed a cascade-of-classifiers approach [11], which has been widely used for real-time applications. The method has been extended by employing different types of features and techniques [13], but fundamentally the concept of the cascade, with early rejection of majority of test examples, has been widely utilized to achieve real-time performance. Perhaps the most popular feature used for pedestrian detection (and several other image-based detection tasks) is the HOG feature developed by Dalal and Triggs [10]. Although not real-time, about 1 FPS, this work has been instrumental to the development of faster and more accurate features for pedestrian detection, which are used in the top performing methods in combination with SVM or Decision forests [5, 12, 13]. Deformable Parts Models [7] have shown success on the pedestrian detection task [8, 9]. Deep learningbased techniques have also been applied to pedestrian detection and have led to improve-ments in accuracy [3, 15]. These approaches are still slow, ranging from over a second per image [15] to several minutes[17]. The faster approaches do not apply deep nets to the raw pixel input so their accuracy is reduced.

Improving the speed of pedestrian detection has also been an active area. Benenson et al. proposed a method reaching speeds of 100 to 135 FPS [4] for detection in a 480x640 image, albeit with significantly lower accuracy. Other researchers have focused specifically on speeding up Deep Neural Networks [9, 11, 13], but with no real-time solutions yet.

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. There are many convenient video processing classes. CvVideoCamera is easy to use for returning video stream from the camera, and programmer can implement image processing tasks on each frame. CascadeClassifier can do detection for many facial features. CoreImage is a pixel-accurate, non-destructive image processing technology in Mac OS and IOS. It assembles the code for this instruction pipeline with a just-in-time compiler, which is executed by either the CPU or graphics card's GPU, whichever can perform the calculation faster.

3. Approach.

3.1. Datasets

We take hundreds of real-world pictures in Pittsburgh and cut the pedestrians off to the size 40 * 98. With the positive samples of three pedestrian datasets, we collect 8450 positive samples and more than 10 thousands negative samples and convert all of them to gray scale images for training.

A. INRIA pedestrian dataset

The first is the well-established INRIA pedestrian database, containing 3506 64×128 images of humans cropped from a varied set of personal photos. It contains various views with a relatively large range of poses. Fig. 1 shows some samples. The people are usually standing, but appear in any orientation and against a wide variety of background image including crowds. Many are bystanders taken from the image backgrounds, so there is no particular bias on their pose.

Figure 1： Positive samples, which are 64×128 images of humans cropped from a varied set of personal photos.

B. CVC01 and CVC02 pedestrian dataset

CVC01 and CVC02 consists of 3054 images of humans. The imagery has been recorded in urban scenarios around Barcelona (Spain), using a Bumblebee color stereo camera with resolution 640x480 pixels and 6mm focal length. The annotated pedestrians are in the range from 0 to 50 m from the camera, which corresponds to a smallest pedestrian of 12x24 pixels.

C. MIT CBCL Pedestrian Dataset

CBCL dataset of people was generated from color images and video sequences taken in Boston and Cambridge in a variety of seasons using several different digital cameras and video recorders. Each image was extracted from raw data and was scaled to the size 64x128 and aligned so that the person's body was in the center of the image.

3.2. Feature Extraction Methods

A. Haar-like feature

Haar-like features are the difference of the sum of pixels of areas inside the rectangle, which can be at any position and scale within the original image.

Haar-like features are divided into three categories: edge features, linear features, central features and diagonal features, which are composed of feature templates. Fig. 2 shows these three categories. The feature template has two rectangles：white and black. Haar eigenvalues reflect the gray level of the image. Such as facial features can described by a simple rectangle features, such as: The color of eyes is darker than the color of the cheek, the color of both sides of the nose is darker than the color of nose, the

mouth is darker than the surrounding color. But the rectangular feature is sensitive to some simple graphics, such as edges and line segments, so it can only describe the structure of a specific trend (horizontal, vertical, diagonal).

One of the commonly used techniques is the integral image. Integral image is a matrix of the same size as the original image, and the value of each element is the sum of all the pixels in the upper left corner of the image.

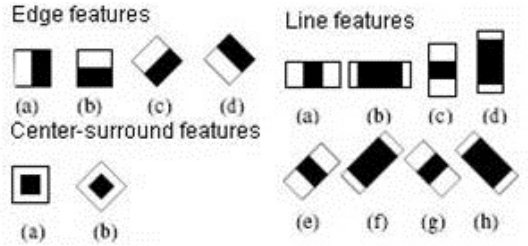

Figure 2: Three categories of Haar-like feature.

B. HOG feature

The full name of HOG is histogram of oriented gradient which is a feature descriptor used for target detection. This technique counts the number of times of the image local orientation gradient. The method is similar to the edge direction histogram and scale-invariant feature transform, but the difference is that the hog is based on the consistent space of the density matrix to improve the accuracy. Navneet Dalal and Bill Triggs proposed HOG in 2005 CVPR for the first time, in order to use it in static image or video of pedestrian detection[2].

The core idea of HOG is that the shape of the detected object can be described by the intensity gradient or the edge direction. The whole image is divided into small connected regions(cells), each cell generating an orientation gradient histogram or the edge direction of pixel in cell. The combination of these histograms can express descriptors. In order to improve the accuracy, the local histogram can be standardized by calculating the intensity of a large area (block) in the image as measure. And use this value (measure) to normalize all cells in this block. Fig. 3 shows HOG in which image gradients are used to generate descriptors.

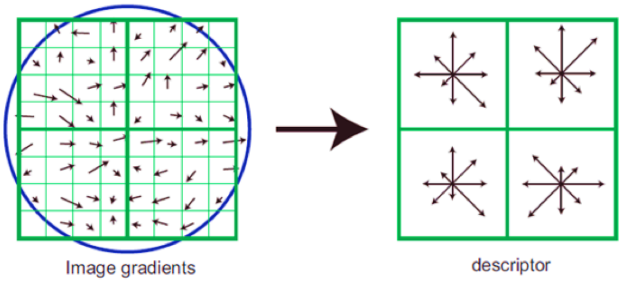

Figure 3: Using image gradients to generate descriptors.

C. LBP feature

The full name of LBP is local binary patterns which is a type of visual descriptor used for classification. And it has since been found to be a powerful feature for texture classification.

In order to get LBP features, the whole image is divided into cells, within each cell, for each pixel, compare the pixel to each of its neighbors. Binary number '0' and '1' are used to represent the value of neighbors greater than the pixel or not, like Fig. 4, and conbine all the binary numbers in a specfic order to generate a n-digit binary number. Fig. 5 shows the process to generate the new binary number. This number is used to represent the LBP value of this pixel. Then, compute the histogram over the cell and concatenate all of them together [3,4,5].

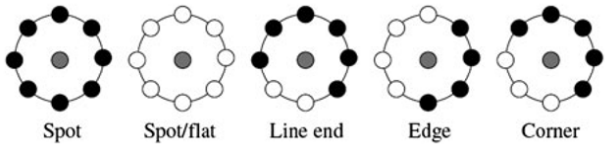

Figure 4: 0 or 1 representation of neighbor pixels.

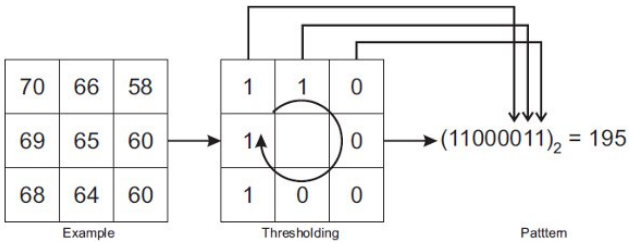

Figure 5: process of generating the new binary number.

3.3. Original Experiments

We use OpenCV to train cascade classifiers. Classification method AdaBoost is to create a strong classifier by the conjuncture of many weak classifiers (hit rate barely better than 50%). AdaBoost works by choosing a base algorithm (e.g. decision trees) and iteratively improving it by accounting for the incorrectly classified examples in the training set.

Based on three features, we tested the dependency of the classifiers on the number of training samples and the number of stages. We did several experiments and got their results. And we used miss rate, false positive per image and frame per second to express the detection results. We also compared our results with the results in the paper [1]. In the paper, they evaluated different methods to detect pedestrians. The average analysis time per image is 300ms.

The number of stages in our tests is 20. We use 8450 positive images and 10000 negative images for every test and we take advantage of 8000 positive images and 5000 negative images per stage.

At first, we use haar-like features. Then, we used HOG

features, and we use LBP features at last.

TABLE I
EXPERIMENTS

Test	Feature	Total Pos Num	Pos Num per Stage	Total Neg Num	Neg Num per Stage
1	Haar-like	8450	8000	10598	5000
2	HOG	8450	8000	10598	5000
3	LBP	8450	8000	10598	5000
Test	Training Time	Stage	FPS		
1	35 hours	20	8.5		
2	42 hours	20	7.5		
3	34 hours	20	18		

3.4. Optimization

A. Non-maximum Suppression

We find in the original experiments, in each frame, we detect multiple bounding boxes surrounding the pedestrian in the image, but non-maximum suppression is about to reduce this number to one bounding box, like Fig. 6.

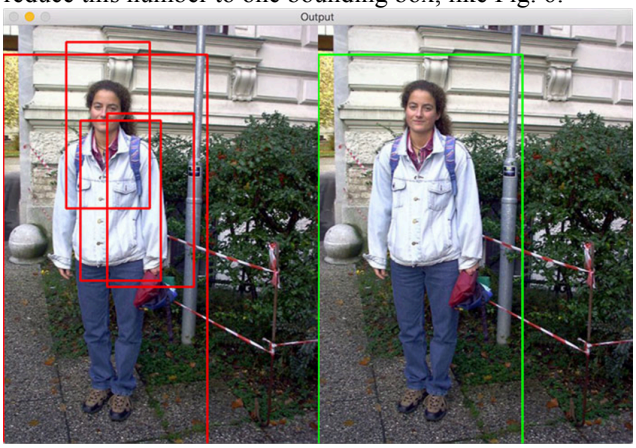

Figure 6: Multiple bounding boxes surrounding the pedestrian and use non-maximum suppression to move.

In our project, non-maximum suppression is used to ignore redundant, overlapping bounding boxes. Normally, we set the threshold as 0.3 which means if the overlap area is more than 30% of the original box area, we move it.

At first, our set of bounding boxes are in the form of (startX, startY, endX, endY) and we compute the area of each of the bounding boxes. Then, we compute the overlap ratios and determine which bounding boxes we can ignore. If the overlap ratio is greater than the threshold, then we know that the two bounding boxes sufficiently overlap and we can thus suppress the current bounding box.

Even for images that contain multiple objects, non-maximum suppression is able to ignore the smaller overlapping bounding boxes and return only the larger ones. However, non-maximum suppression could make FPS slower because it needs some calculations of overlap.

B. pre-image-processing, Armadillo and GPU

First, what we can do to is to add some image processing part into our project. Since if we use some filters and morphological operations, the number of noises on the image would decrease, which will cost less in the later detection. However, these image processing would definitely cost some consumption but compared to de decrease of latter detection, the pre-operation is worth.

What's more, we also improved our project using Armadillo library and taking use of GPU to accelerate the speed of computing.

Armadillo is a linear algebra software library for the C++ programming language. It aims to provide efficient and streamlined base calculations, while at the same time having a straightforward and easy-to-use interface. Its intended target users are scientists and engineers. It supports integer, floating point (single and double precision), complex numbers, and a subset of trigonometric and statistics functions.

Various matrix decompositions are provided through optional integration with Linear Algebra PACKage (LAPACK) and Automatically Tuned Linear Algebra Software (ATLAS) libraries.

High-performance LAPACK replacement libraries such as Math Kernel Library (MKL) and AMD Core Math Library (ACML) can also be used. The library employs a delayed-evaluation approach (during compile time) to combine several operations into one and reduce (or eliminate) the need for temporaries. Where applicable, the order of operations is optimized. Delayed evaluation and optimization are achieved through template metaprogramming.

A graphics processing unit (GPU), occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics and image processing, and their highly parallel structure makes them more efficient than general-purpose CPUs for algorithms where the processing of large blocks of data is done in parallel. In a personal computer, a GPU can be present on a video card, or it can be embedded on the motherboard or—in certain CPUs—on the CPU die.

The CPU (central processing unit) has often been called the brains of the PC. But increasingly, that brain is being enhanced by another part of the PC – the GPU (graphics processing unit), which is its soul. All PCs have chips that render the display images to monitors. But not all these chips are created equal.

4. Results

4.1. Result of Original Experiments

Figure 7: Haar-like features without optimization, 8.5 FPS

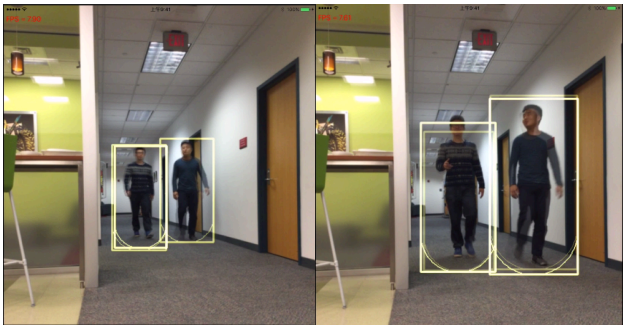

Figure 8: HOG features without optimization, 7.5 FPS

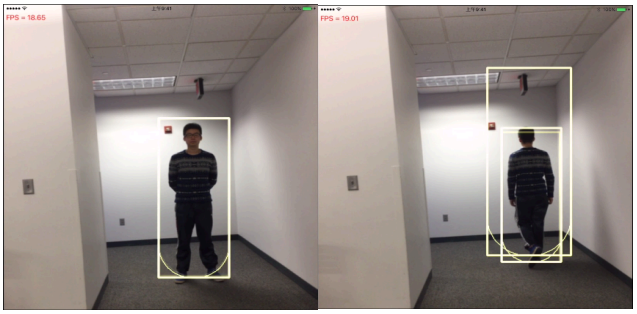

Figure 9: LBP features without optimization, 18 FPS

4.2. Result of Optimization in Accuracy Experiments

Figure 10: Haar-like features with non-maximum supression algorithm, 7.5 FPS

Figure 11: HOG features with non-maximum supression algorithm, 6.5 FPS

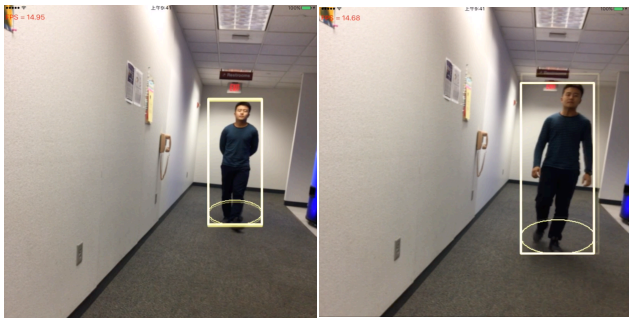

Figure 12: LBP features with non-maximum supression algorithm, 14 FPS

4.3. Result of Optimization in Speed Experiments

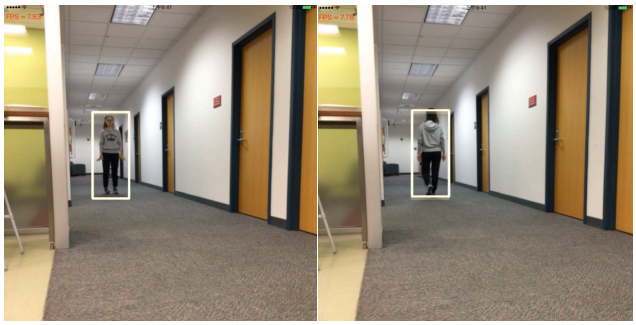

Figure 13: Haar-like features with non-maximum supression algorithm, GPU and Armadillo, 8 FPS

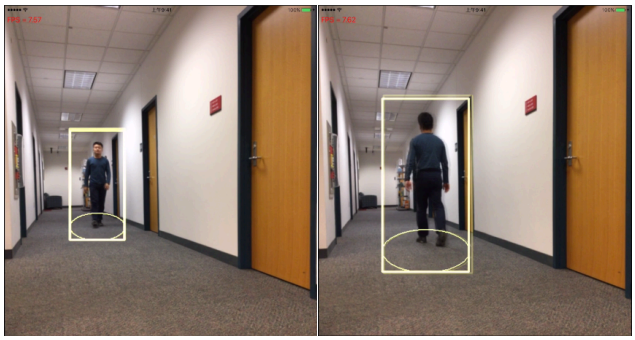

Figure 14: HOG features with non-maximum supression algorithm, GPU and Armadillo, 7.5 FPS

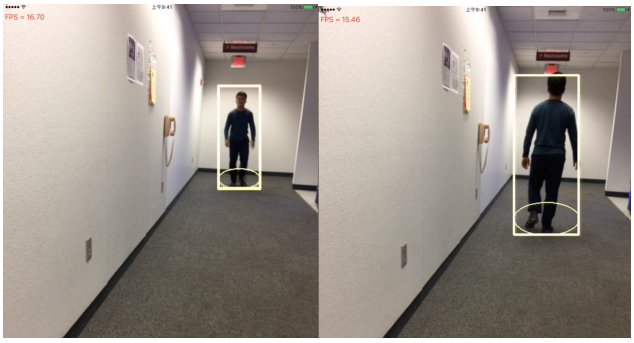

Figure 15: LBP features with non-maximum supression algorithm, GPU and Armadillo, 17 FPS

4.4. Comparison

TABLE II
SPEED COMPARISION AMONG THREE FEATURES IN UNDER DIFFERENT CIRCUMSTANCE

Test	Haar-like (fps)	HOG(fps)	LBP(fps)
1	8.5	7.5	18
2	7.5	6.5	14
3	8	7.5	17

1. No optimization
2. Non-maximum supression optimization
3. Speed optimization using GPU and Armadillo

4.5. Conclusion

The initial experiment with original classifier, we cannot get neither accurate nor fast results. After implementing the non-maximum suppression algorithm, although the results become better, the performance in speed again decreased. Using what we learned from class about how to accelerate video based application on iPad including using GPU and Armadillo library, our results' performance can be robust both in accuracy and speed.

5. List of work

Equal work was performed by both project members.

6. GitHub Page

https://github.com/wuyuzhang/CMU_16623CVAPP_Project

Clone copy:

https://github.com/wuyuzhang/CMU_16623CVAPP_Project.git

Since the check point, the focus has mostly been consistent as planned, but with increasing emphasis on optimization and speed ups of the algorithm rather than exploratory processes of the proof of concept. We have implemented project with optimization on both improving accuracy using a noise-eliminating algorithm and accelerating speed with GPU and Armadillo library. What we did also include real-time experiment to check the results.

7. References

[1] Dollar P, Wojek C, Schiele B, et al. Pedestrian Detection: An Evaluation Of The State Of The Art[J]. Pattern Analysis & Machine Intelligence IEEE Transactions on, 2012, 34(4):743-761.

[2] Dalal N, Triggs B. Histograms of oriented gradients for human detection[J]. Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, 1(12):886-893.

[3] T. Ojala, M. Pietikäinen, and D. Harwood (1994). Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. Proceedings of the 12th IAPR International Conference on Pattern Recognition (ICPR 1994), vol. 1, pp. 582 - 585.

[4] T. Ojala, M. Pietikäinen, and D. Harwood (1996). A Comparative Study of Texture Measures with Classification Based on Feature Distributions. Pattern Recognition, vol. 29, pp. 51-59.

[5] C. Silva, T. Bouwmans, C. Frelicot,. An eXtended Center-Symmetric Local Binary Pattern for Background Modeling and Subtraction in Videos. VISAPP 2015, Berlin, Germany, March 2015.

[6] R. Benenson, M. Matthias, T. Tuytrlaars, and L. Van Gool. Seeking the strongest rigiddetector. CVPR, 2013.

[7] R. Benenson, M. Omran, J. Hosang, and B. Schiele. Ten years of pedestrian detection,what have we learned? 2nd Workshop on Road scene understanding and Autonomousdriving, ECCV, 2014.

[8] G. Chen, Y. Ding, J. Xiao, and T. Han. Detection evolution with multi-order contextualco-occurrence. CVPR, 2013.

[9] E. Coelingh, A. Eidehall, and M. Bengtsson. Collision warning with full auto brakeand pedestrian detection - a practical example of automatic emergency braking. 13th\International IEEE Conference on Intelligent Transportation Systems (ITSC), 2010.

[10] A. Costea and S. Nedevschi. Word channel based multiscale pedestrian detection withoutimage resizing and using only one classifier. CVPR14, 2014.

[11] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. CVPR,2005.

[12] Y. Ding and J. Xiao. Contextual boost for pedestrian detection. CVPR, 2012.

[13] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark.CVPR, 2009.

[14] P. Dollar, R. Appel, and P Perona. Crosstalk cascades for frame-rate pedestrian detector.ECCV, 2010.

[15] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation ofthe state of the art. PAMI, 2012.

[16] M. Enzweiler and D. M. Gavrila. Monocular pedestrian detection: Survey and experiments. IEEE Trans. on Pattern Analysis and Machine Intelligence (T-PAMI), 2009.

[17] A. Ess, B. Leibe, K. Schindler, , and L. van Gool. A mobile vision system for robust multi-person tracking. CVPR, 2008.
[18] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. PAMI, 2010.