ENHANCEMENT OF FAST PEDESTRIAN DETECTION USING HOG FEATURES

[1]AHMAD BAGHDADI, [2]SUHAIMI ABD LATIF

[1,2]Electrical and Computer Engineering Department, IIUM
E-mail: ahmad.baghdadi@live.iium.edu.my, suhaimie@iium.edu.my

Abstract- Histograms of Oriented Gradients (HOG) suffers low processing speed, due to complex and redundant computations. In this paper, a method to improve the speed of detection while maintaining the accuracy has been proposed. This is implemented without using any special purpose hardware such as GPU. The speed has been substantially increased through the calculations of trilinear interpolation within each block of cells, utilizing the sub-cell method. The redundant calculation over two levels has been avoided: within each block and within the detection window by utilizing the concept of reusing the calculated features. Results showed a significant speed up by more than eleven times, with almost the same detection accuracy of the original algorithm.

Keywords- Pedestrian Detection, HOG, Fast Method

I. INTRODUCTION

Pedestrian detection plays a very important role in many advanced applications related to the field of image processing and computer vision, such as human-machine interaction, intelligent surveillance and security, machine vision for mobile robotics, and especially in automotive safety like driver assistance systems. Two main capabilities are required for any pedestrian detection system, the speed of detection, and the accuracy/quality of detection. Usually the trade-off between these two capabilities makes the problem of pedestrian detection a quite challenging task. That is, more accurate detection typically uses more complex algorithms, which consumes more time and affects the speed of detection. Other factors such as the variety of pedestrians' scales, positions and locations, appearance and occlusion conditions, also increase the difficulty of the detection as well.

Achieving a very good accuracy requires strong and distinct features, and that's why Histograms of Oriented Gradients (HOG) features are commonly used to represent information about shapes and appearances in the image. Moreover, HOG features provide good resistance to the changes in the appearance (illumination and shadowing), shape and location of the pedestrians.

However, achieving a fast pedestrian detection using this algorithm is quite difficult due to the computational complexity. In this paper, we illustrate where the complexity of this algorithm is, then we propose a method to improve the speed of detection while maintaining the accuracy as much as possible, and without using any special purpose hardware such as GPU. The paper is organized as follows. Section II introduces some of the related work. Section III describes the HOG algorithm, and Section IV describes the proposed method. The experimental results will be presented in Section V. Finally, Section VI concludes this paper.

II. RELATED WORK

Remarkable improvements have been introduced to pedestrian detection recently since the contribution of Dalal and Triggs in when they used HOG features. According to the reviews made by Dollar et al. in, and Benenson et al. in, the performances of state-of-the-art pedestrian detectors have been evaluated across several datasets, and reported that HOG-based methods are still preferable in this domain. This is because of two reasons: first, HOG feature outperforms any other individual feature, while using HOG with additional features can improve the detection accuracy; second, HOG – compared to other methods – provides middle level of performance between computational time and accuracy.

Combining HOG with other features such as Haar-like can enrich the image descriptor to improve the detection performance, but with an increase in the running time. To improve the speed, Zhu et al. used cascade-of-rejectors concept with AdaBoost to select a small group of features from the traditional HOG features. However, this method is giving lower accuracy performance than the original HOG algorithm. In Y. Pang et al. used sub-cell based trilinear interpolation during the calculation of HOG features in order to reduce the time consumption. However, redundant calculations still exist in the detection window in overlapped cells of the blocks.

A parts-based approach is used by Enzweiler et al. in , where pedestrian shapes have been decomposed into parts such as head, torso, and legs. This method has a very promising performance, especially in increased occlusion conditions and moving body parts. However, handling occlusion conditions is not an objective in this paper.

To speed up the input image scaling, Dollar et al. in proposed a method to use features calculated at a single scale to approximate features at nearby scales.

This method can provide a near real-time performance with detection rate of ~6 fps on 640 x 480 images. In order to achieve high detection accuracy in real-time performance, Jianxin et al. in proposed C^4, which is a real-time object detection framework that applied to detect pedestrians, faces, and cars. Their proposal is based on a computational method that does not clearly need to generate feature vectors. A new concept has been proposed by Benenson et al. in for achieving high speed pedestrian detection (100 frames per second), it involves moving the computation of resizing the images from test time into training time, and reducing the detection search space using prior information.
Other researchers in have utilized the parallel computing power of Graphics Processing Unit (GPU) for accelerating the calculation of the HOG features to achieve real-time detection. However, using GPU is beyond the scope of this paper. The aim of this paper is to find a way to simplify and reduce the amount of calculations during the extraction of HOG features, in order to achieve fast pedestrian detection with a good accuracy level.

III. HOG ALGORITHM

The main idea behind the strength of HOG features is to focus on the contrast between the contours of pedestrians against the background. These features are considered to be very discriminative, since all pedestrians have almost similar contours. HOG algorithm is based on evaluating well-normalized local histograms of image gradient orientations over a dense grid of spatial blocks. The extraction of HOG features for an image is done through the following steps:
1. Image normalization using Gamma compression.
2. Computation of the first order image gradients.
3. Accumulation of the gradient orientation information locally for all pixels in each cell to form the orientation histogram representation.
4. Contrast normalization within the overlapped blocks of cells. This will form the HOG descriptors.
5. Collection of the HOG descriptors from all blocks in the detection window into a combined feature vector to be used in the window classifier.

The default parameters defined for HOG features calculation are:

- Detection window size in pixels (64×128)
- Cell size in pixels (8×8)
- Number of cells in a block (2×2)
- Block stride in pixels (8×8)
- Number of orientation bins per each cell (N= 9)
- Block size in pixels (16×16)
- Number of cells per detection window (8×16=128)
- Number of blocks per detection window (7×15=105)
- Number of features per block (4×9=36)
- Number of features per detection window (36×105=3780)

Now the steps of calculation of HOG features in details would be as follows:

1. The image normalization is done globally and optionally, once for each image, using Gamma correction (power law) for each color channel. The simplest way to do that is to replace each color channel by its square root:

$$I'(x,y) = \sqrt{I(x,y)} \quad (1)$$

2. The calculation of the first order gradients is done for each pixel in the Image $I'(x,y)$ in both x and y directions using the simple centered derivative mask [-1 0 1] which gives the best performance for pedestrian detection regarding the miss rate at 10−4 FPPW. This can be expressed as the following:

$$Gx = [-1\ 0\ 1] \times I'(x,y) \quad (2)$$

$$Gy = [-1\ 0\ 1]' \times I'(x,y) \quad (3)$$

The gradients (Gx, Gy) are then combined to calculate the magnitude (G) and the orientation (θ) of the gradient as follows:

$$G = \sqrt{Gx^2 + Gy^2} \quad (4)$$

$$\theta = \tan^{-1}(Gx/Gy) \quad (5)$$

3. After dividing the detection window into cells of pixels, we calculate the orientation histogram for each cell depending on the weighted vote of each pixel in the cell (the vote is a function of the gradient magnitude (G)). In this step we distribute the weighted votes into orientation bins separated equally by 20° over the range 0° - 180°, which gives N = 9 orientation bins in the histogram. Since we group a number of cells to form a block, the weighted votes of the pixels are down-weighted by a Gaussian spatial window first (centered at the middle of the block), then after that in order to reduce aliasing, a 3D-linear (trilinear) interpolation is performed (in both orientation and position) to distribute each vote into neighbor orientation bins. Triliniear interpolation which is applied in this step improves detection and allows more accurate results, since it makes the weighted vote of the pixels in each cell not only affect its orientation histogram, but also contribute in the three neighbor cells in the same block. Therefore, the weighted vote will be distributed over the 8 surrounding bins (two nearest neighboring bins for each cell in the block). Two variables would be used to express the position of the cell in the block, and another variable for the orientation.Assume that (w) is the weight of a pixel (p) in a cell, let (x,y) define the position of the pixel, (x1,x2,y1,y2) define the four different combinations of the affected cells, (z) defines the orientation, (z1,z2) define the two nearest orientation bin centers, (xc,yc) define the cell center, (bx,by) define the cell bandwidth, (bz) defines the inter-bin distance for each cell (20°). Now we can calculate the orientation histogram at each pixel as follows:

$$h(x1,y1,z1) \leftarrow h(x1,y1,z1) + w \times \left(1 - \frac{x-xc}{bx}\right) \times \left(1 - \frac{y-yc}{by}\right) \times \left(1 - \frac{z-z1}{bz}\right) \quad (6)$$

$$h(x1,y1,z2) \leftarrow h(x1,y1,z2) + w \times \left(1 - \frac{x-xc}{bx}\right) \times \left(1 - \frac{y-yc}{by}\right) \times \left(\frac{z-z1}{bz}\right)$$

$$h(x1,y2,z1) \leftarrow h(x1,y2,z1) + w \times \left(1-\frac{x-xc}{bx}\right) \times \left(\frac{y-yc}{by}\right) \times \left(1-\frac{z-z1}{bz}\right) \quad (7)$$

$$h(x1,y2,z2) \leftarrow h(x1,y2,z2) + w \times \left(1-\frac{x-xc}{bx}\right) \times \left(\frac{y-yc}{by}\right) \times \left(\frac{z-z1}{bz}\right) \quad (8)$$

$$h(x2,y1,z1) \leftarrow h(x2,y1,z1) + w \times \left(\frac{x-xc}{bx}\right) \times \left(1-\frac{y-yc}{by}\right) \times \left(1-\frac{z-z1}{bz}\right) \quad (9)$$

$$h(x2,y1,z2) \leftarrow h(x2,y1,z2) + w \times \left(\frac{x-xc}{bx}\right) \times \left(1-\frac{y-yc}{by}\right) \times \left(\frac{z-z1}{bz}\right) \quad (10)$$

$$h(x2,y2,z1) \leftarrow h(x2,y2,z1) + w \times \left(\frac{x-xc}{bx}\right) \times \left(\frac{y-yc}{by}\right) \times \left(1-\frac{z-z1}{bz}\right) \quad (11)$$

$$h(x2,y2,z2) \leftarrow h(x2,y2,z2) + w \times \left(\frac{x-xc}{bx}\right) \times \left(\frac{y-yc}{by}\right) \times \left(\frac{z-z1}{bz}\right) \quad (12)$$

(13)

Fig. 1 shows the parameters of the trilinear equations mentioned previously for a block in the detection window.

4. Blocks are typically overlapped in the detection window, and each block contains multiple cell histograms from other blocks, since each cell is considered to be a part of several blocks. In order to enhance the detection performance, block based contrast normalization is done using L-2 norm as follows:

$$L-2\,\text{norm}: v \leftarrow v/\sqrt{\|v_k\|^2 + \varepsilon^2} \quad (14)$$

Where (v) represents the non-normalized vector that contains the entire histogram elements in a single block, ($\|v_k\|$) its k-norm, and (ε) a small constant to avoid division by zero. The normalized vector now represents the HOG descriptor for the block.

5. Finally, we collect the HOG descriptors for all blocks within the detection window in order to assemble the final descriptor.

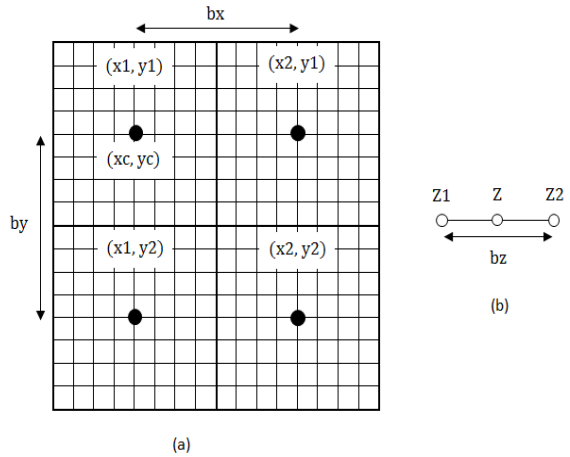


Figure 1. Parameters of the trilinear equations: (a) Position coefficients, (b) Orientation coefficients

According to the previous overview, HOG complexity is coming from two issues; First: it is obvious that the computational cost of the trilinear interpolation is very high, due to the equations required to distribute the weight of each pixel over 4 cells. Second: Block overlapping improves the detection performance, but in the same timeit will produce redundant calculations that increase the computational cost as well. Since the block stride is equal to 8 pixels (one cell) in the default detection window, this will produce 50% overlapping with 4-fold coverage.

Without any padded pixels at the border of the detection window, there will be 4 cells (one at each corner) without any overlapping, 40 cells overlapped two times, and 84 cells overlapped four times. Therefore, the total number of cells will be (4 ×1) + (40 ×2) + (84 ×4) =420 cells, instead of 128 cells. That is, redundant calculations are naturally involved in this algorithm, as shown in as shown in Fig. 2.

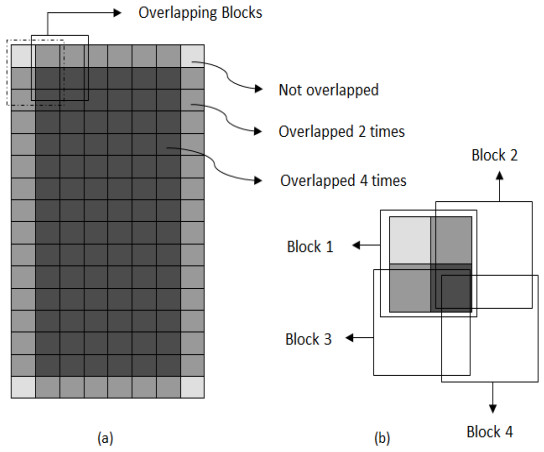


Figure 2. Overlapping in the detection window: (a) Cells in the detection window, (b) Overlapped blocks

IV. PROPOSED METHOD

In the proposed method, we applied the same values of default parameters as mentioned previously, along with linear SVM classifier as in. The improvements achieved by utilizing two techniques:

The sub-cell method and the concept of reusing the calculated features. In the first technique, we adapt the sub-cell method used in to reduce the computational cost inside each block of cells, and this is done through speeding up the calculation of trilinear interpolation.

Each cell is divided into 4 sub-cells, as shown in Fig. 3. The idea is to define 3 regions within the cell according to their locations from the neighboring cells.

This will reduce the equations required for interpolation. The regions are defined as the following:

- The first region is defined in the near corner of the cell, and any pixel in this cell will contribute only to its cell histogram.

- The second region is defined at the border of neighboring cells, and any pixel there will contribute to its cell histogram in addition to the direct neighboring cell histogram.

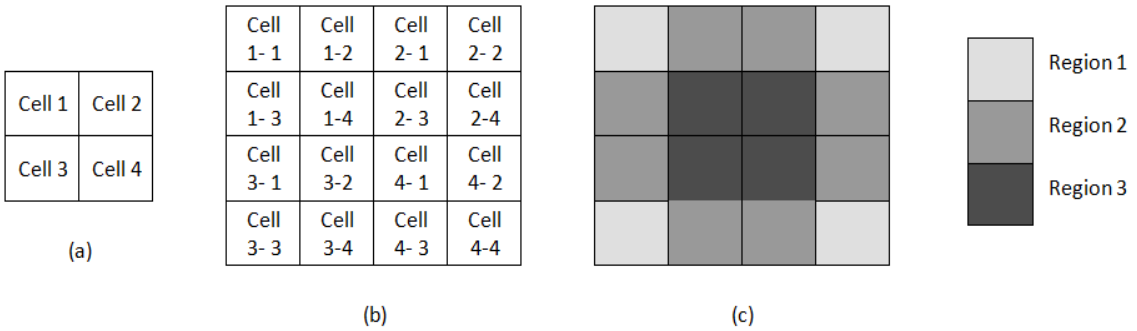


Figure 3. Sub-cell method: (a) Normal block, (b) Sub-cells within a block, (c) Three regions within a block

- The last region is defined at the far corner of the cell which is near to the center of the block, and therefore any pixel there will contribute into all cells histograms in the block.

The total number of sub-cells is 16 according to Fig. 3. To calculate any histogram in the block we need the contribution of 16 sub-cells using the original HOG algorithm. Instead of that, we only need 9 sub-cells using the sub-cell method. Thus, a total reduction of 5 sub-cell (80 pixels) unimportant contributions per histogram is achieved. Using the sub-cell method does not harm the accuracy of detection, since the accuracy is related to the number of features per detection window, which is still 3780 and didn't change. In the second technique, we utilize the concept of reusing the calculated features. Two levels are defined for this purpose: block level and detection window level. The first level is done within each block, since all cells are identical and have the same bandwidth and center. This will make the position coefficients of trilinear equations for the pixel inside the cell doesn't change from a cell to another, the only change will be the weight and the orientation of the pixel. Thus we can reuse the same values of coefficients calculated previously for different pixels that have the same positions (within a cell) in different cells. Upon the fact that blocks are defined from the top left corner and slides into bottom right corner of the detection window with a stride equal to one cell horizontally and vertically, most of these blocks are overlapped in the detection window.

This will introduce redundant calculations that increase the time consuming of the whole process. For that reason, the second level of reuse is done within the detection window, since all calculations of trilinear interpolation are repeated among blocks in the same way. The proposed method is developed to calculate features for each block and then reuse these features for the overlapped part in the next block without repeating the calculations again. For example, the calculated features form cell-2 and cell-4 in block-1, will be reused for cell-1 and cell-3 in the block-2, the same thing for cell-3 and cell-4 in block-1 will be reused for cell-1 and cell-2 in block-3, and so on.

V. EXPERIMENTAL RESULTS

To evaluate the performance of our proposed method, we use the INRIA person dataset which is originally used in. The training dataset includes 1208 normalized

positive images which contain pedestrians with an average height of 96 pixels, and 12180 negative images (sampled from 1218 images). The testing dataset includes 288 positive and 453 negative images. Experiments are done on a moderate PC machine (CPU: Intel Core i5 M520 @ 2.4GHz, RAM: 4GB, OS: Ubuntu Linux 14.10).

Using the sub-cell method reduced the computational time up to 43% due to the reduction of sub-cell contribution in each histogram from 16 to 9 sub-cells only. Furthermore, utilizing the concept of reusing the calculated features provided a speed up in the detection process up to 47%. Results show that a 640 x 480 image requires about 4.3 seconds to be processed using the original HOG algorithm (INRIA Object Detection and Localization Toolkit - OLT) by Dalal and Triggs in . On the other hand, our implementation using OpenCV library on the same PC machine reduced the run-time to about 490 milliseconds.

Therefore, our implementation gives a speed up more than eleven times from the original HOG algorithm. Table 1 illustrates the run-time comparison between our implementation and the original HOG algorithm for different image scales and detection windows.

Table 1. Run-time comparison

Image scale	1.05	1.10	1.15	1.20
Number of tested windows	31201	17020	12375	10039
Run-time (ms)				
Original HOG	4998	2875	2011	1663
Our implementation	439	239	173	141

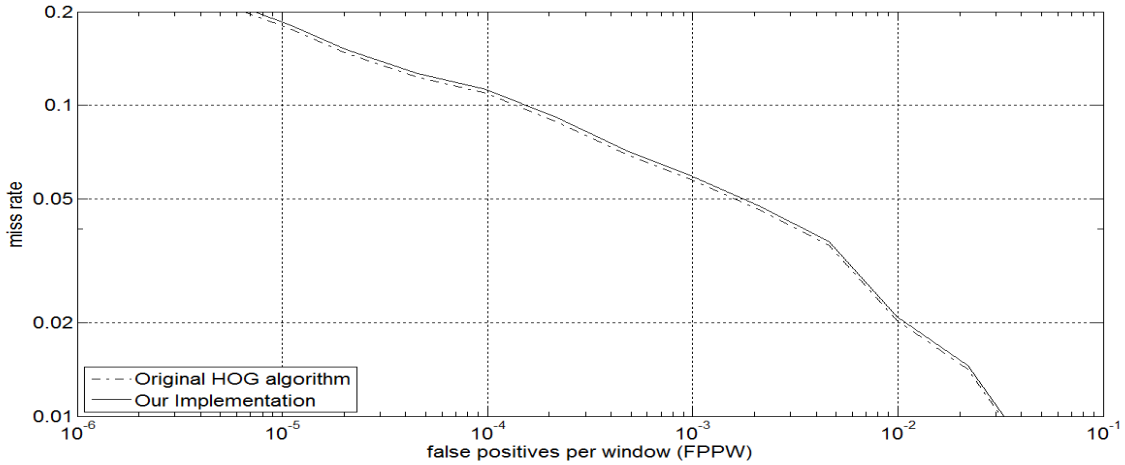


Figure 4. DET curves comparison

In terms of accuracy, we use the Detection Error Tradeoff (DET) curves to measure the amount of true detections against the amount of false positives, as used in [1]. These curves plot miss rate versus False Positives per Window (FPPW) on a log-log scale, with a reference point of 10-4 FPPW for results. Fig. 4 shows almost identical performance between the original HOG algorithm and the proposed method on the INRIA person dataset.

CONCLUSION

In this paper, we have proposed a fast HOG method for pedestrian detection. The method is based on utilizing two techniques: The sub-cell method and the concept of reusing the calculated features. The enhancement is done by speeding up the calculations of trilinear interpolation, and reducing the redundant calculations over two levels: within each block and within the detection window. The results on a 640 x 480 image showed a speed up in the processing time reached 11 times compared to the original HOG algorithm, with almost the same accuracy. In the future, we intend to introduce more enhancements to reduce image scaling operations, using an empirical approximation technique, in order to achieve a real-time detection performance.

REFERENCES

[1] N. Dalal, B. Triggs, “Histograms of oriented gradients for human detection,” IEEE Proc. on (CVPR), vol. 1, pp. 886-893, June 2005.

[2] P. Dollar, C. Wojek, B. Schiele, P. Perona, “Pedestrian Detection: An Evaluation of the State of the Art,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 34, no. 4, pp. 743-761, April 2012.

[3] S. Banerji, A. Sinha, and C. Liu, “HaarHOG: Improving the HOG Descriptor for Image Classification,” IEEE Proc. on (SMC), vol. 1, pp. 4276-4281, Oct. 2013.

[4] Y. Wei, Q. Tian, and T. Guo, “An Improved Pedestrian Detection Algorithm Integrating Haar-Like Features and HOG Descriptors,” Advances in Mechanical Engineering, vol. 2013, Article ID 546206, 8 pages, 2013.

[5] Q. Zhu, M-C. Yeh, K-T Cheng, S. Avidan, “Fast Human Detection Using a Cascade of Histograms of Oriented Gradients,” IEEE Proc. on (CVPR), vol. 2, pp. 1491-1498, 2006.

[6] V. Prisacariu, I. Reid, “fastHOG - a real-time GPU implementation of HOG,” Technical Report No. 2310/09, Department of Engineering Science, Oxford University, July 2009.

[7] A. Gepperth, M.G Ortiz, and B. Heisele, “Real-time pedestrian detection and pose classification on a GPU,” IEEE Proc. on (ITSC) vol. 1, pp. 348-353, Oct. 2013.

[8] Y. Pang, Y. Yuan, X. Li, and J. Pan, “Efficient HOG human detection,” Signal Processing, vol. 91, no. 4, pp. 773-781, April 2011.

[9] R. Benenson, M. Omran, J. Hosang, and B. Schiele. "Ten Years of Pedestrian Detection, What Have We Learned?," arXiv preprint arXiv:1411-4304, 2014.

[10] M. Enzweiler, A. Eigenstetter, B. Schiele, and D.M. Gavrila, "Multi-cue pedestrian classification with partial occlusion handling," IEEE Proc. on (CVPR), pp. 990-997, June 2010.

[11] P. Dollar, B. Schiele, P. Perona, “The Fastest Pedestrian Detector in the West,” Proceedings of the British Machine Vision Conference (BMVA), pages 68.1-68.11, Sept. 2010.

[12] J. Wu, N. Liu, C. Geyer, and J. M. Rehg, "C4: A Real-Time Object Detection Framework," IEEE Trans. on Image Processing, vol. 22, no. 10, pp. 4096-4107, Oct. 2013.

[13] R. Benenson, M. Mathias, R. Timofte, and L Van Gool, "Pedestrian detection at 100 frames per second," IEEE Proc. on (CVPR), pp. 2903-2910, June 2012.

★★★