International Journal of Database Theory and Application
Vol.8, No.6 (2015), pp.125-132
http://dx.doi.org/10.14257/ijdta.2015.8.6.11

Fast Pedestrian Detection with Adaboost Algorithm Using GPU

Chong Chao Cai[1, 2*], Jue Gao[3], Bian Minjie[1, 4], Peicheng Zhang[1, 4] and Honghao Gao[1, 3]

[1]*School of Computer Engineering and Science, Shanghai University, shanghai, China*
[2] *Faculty of Information Technology, Huzhou Vocational & Technical College, Huzhou, Zhejiang, China*
[3]*Computing Center, Shanghai University, Shanghai, China*
[4]*Shanghai Shang Da Hai Run Information System Co., Ltd, Shanghai, China*
caichongchao@163.com

Abstract

Pedestrian detection is one of the hot research problems in computer vision field. The Cascade AdaBoost System is a commonly used algorithm in this region. However, when the training datasets become larger, it is still a time consuming process to build one Adaboost classifier. In this paper we detail an implementation of the AdaBoost algorithm using the NVIDIA CUDA framework based on the haar features as feature vectors, and downscaling with integral image. The result shows that we can get nearly 6x from the standard code to with our CPU implementation to achieve a near real-time performance and ensure better classification results in misjudgment.

Keywords: *Pedestrian Detection, CUDA, Adaboost, Haar Feature, Integral Image*

1. Introduction

Pedestrian detection plays an important role in the field of computer vision and has wide applications such as video monitoring, visual surveillance, smart room, automatic driver-assistance system and other fields. During the last two decades, the pedestrian detection problem has received a great amount of interest and various representations and detection schemes have been proposed. Whether the emerging Internet giants of Google and Apple, or Ford, Daimler, they had invested a lot of manpower and resources to study in the field of automatic driving. Fast and accurate real-time pedestrian detection is the key technique.

There were extensive literatures about human detection which provided a lot of feasibility research methods. Viola *et al* [1] proposed a real-time face detection framework in 2004. The framework used AdaBoost [2] to train a chain of rejection rules that employ Haar-like wavelets and spatial-temporal differences. A Haar-like feature taking into account specific location adjacent to a rectangular area in the detection window, summarizes the difference between the pixel intensities in each region and calculate sums.

Dalal and Triggs [3] proposed the histogram of gradient (HOG) that had robust feature set and granted excellent detection results. This study got good results in human detection, and after that many works were based on HOG. Dollar *et al* [4] benchmarked sliding-window based pedestrian detectors, and their works included a Matlab toolbox besides many useful source codes and Caltech Datasets for pedestrian detection benchmark.

In this paper an implementation of the AdaBoost algorithm using the NVIDIA CUDA framework was introduced. With a Haar-like feature as feature vectors, using integral image feature for downscaling. The result shows that we can speed up the cascade detector to achieve a near real-time performance, compared to CPU, The results also present

ISSN: 2005-4270 IJDTA

several available benchmark video and show that the speed exceeds the state of art in calculation in real time.

2. Feature Selection

In the process of pedestrian detection, we need to analyze the features vectors of detection region in an image or video to determine whether it is the pedestrian. It requires multi features to build human shape. These features should be good at detecting pedestrians and non-pedestrians. In this paper, feature extraction based on Haar wavelet.

Figure 1 shows the characteristics for a specific classification are determined by the specified location, shape, within the area of interest and scale. A simple rectangular Haar-like feature can be defined as the difference of the sum of pixels of areas inside the rectangle, which can be at any position and scale within the original image. Linear feature calculation of characteristics of image pixel rectangle that covers the entire feature consists of two white stripes and black stripe.

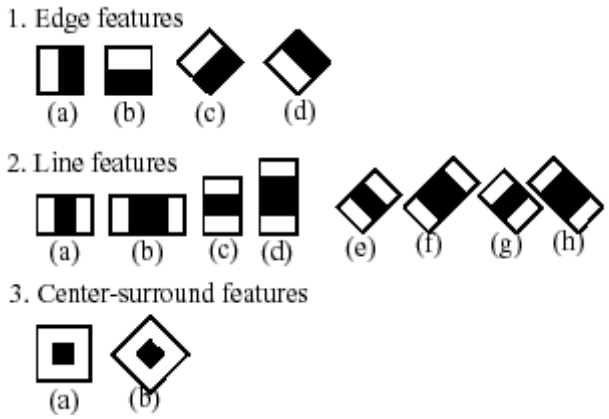


Figure 1. Haar-Like Features

Haar feature defines a group of feature detection operator. The detector is a sliding window algorithm. Whether training or testing, the problem of how to calculate feature value of the current image must be solved. In a 36×36 window arbitrary permutation generate hundreds of thousands features at least. The computation for these features is very huge, so the integral image method was used. Integral image can be defined as a matrix of the same size as the original image of a two-dimensional lookup table. Each element of the integral image contains all of the pixels in the upper-left area of the original image, which allows the calculation and sum of the rectangular area of the image. In any location or scale, using only four queries by the same amount of time to calculate the different features, greatly improved speed.

Figure 2 shows the integral image [5] in the 1 point value for the regional gray level A sum, denoted by A, in the 2 point value is A+B, in the 3 point value is A+C, in the 4 point value is A+B+C+D. Then the sum of gray level rectangular region D surrounded by 1-4 point can be expressed as A+C-B-D. As the name suggests, the value at any point (x, y) in the summed area table is just the sum of all the pixels above and to the left of (x, y)

$$I(x, y) = \sum_{\substack{x' \le x \\ y' \le y}} i(x', y')$$

Moreover, the summed region can be computed efficiently in a single pass over the image, using the fact that the value in the summed area table.

$$I(x,y)=i(x,y)+I(x-1,y)+I(x,y-1)-I(x-1,y-1)$$

The calculation of arbitrary rectangular area and can be done in constant time.

$$\sum_{\substack{A(x)<x'\le C(x)\\ A(y)<y'\le C(y)}} i(x',y')=I(C)+I(A)-I(B)-I(D)$$

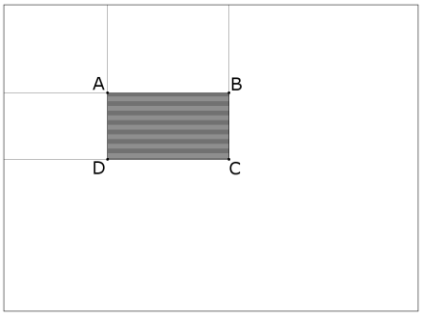


Figure 2. Calculation of Feature

3. Adaboost Algorithm

Adaboost algorithm is a huge improvement on boosting model. It has been widely used in the machine learning. The aim of algorithm is combining many weak learning classifiers in order to produce a strong learning classifier. Basic steps are shown as Table 1.

Table 1. Description of Adaboost Algorithm

Algorithm 1:AdaBoost Algorithm
1. given training sample set S, $S=\{(x_1,y_1),...,(x_i,y_i)\}$ And a predetermined number of iterations. $x_i \in X$ Is the feature vector of sample, $y_i \in \{+1,-1\}$ sample of category labels;
2. initializing the weights of training samples for $d_n^1=1/m$, m For the total sample, normalized weight;
3. by iterating to obtain the weak classifier, the number of iterations $t=1,2...T$ (T to solve a number of weak classifiers)
1) use with weight distribution d_i the training data set to learn, get the basic weak classifier, $h_t : x \to \{+1,-1\}$;
2) compute the h_t The classification error rate on the training data set $\varepsilon_t=\sum_{n=1}^{N} d_n^t I(h_t(x_i)\neq y_i)$
3) the calculation of the weak classifier h_t weighting coefficients in the final

classifier in the set $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$

4) update the sample weights for the next iteration
$d_n^{t+1} = d_n^t \exp(-\alpha_t y_i h_t(x_i)) / Z_t$ the said normalization factor $Z_t = \sum_{i=1}^{N} d_t^i \exp(-\alpha_t y_i h_t(x))$

4. combined weak classifier, $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$, get final strong classifier

$$H(x) = sign(f(x)) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

4. Achieve Adaboost Algorithm

Adaboost algorithm is very time-consuming. We introduce CUDA [6] to implement it. CUDA is an official name of GPGPU (General-purpose computing on graphics processing units) from NVIDIA Company. GPUs can be used for general treatment (not just graphics), unlike CPUs, GPUs has focused on a large number of concurrent threads which run concurrently with the slower speed of flow structure, rather than fast running a single thread.

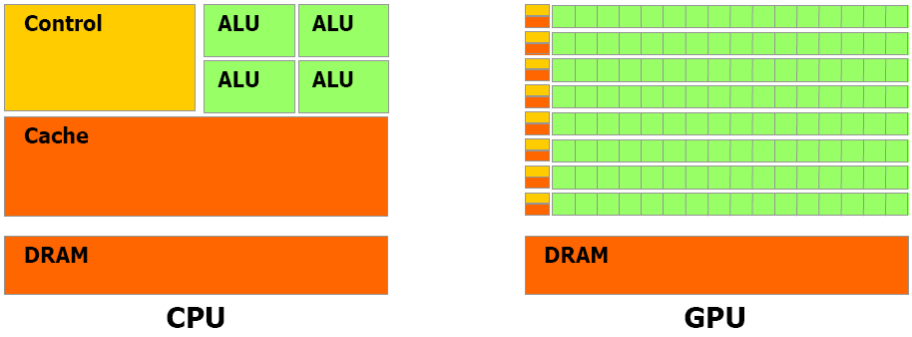


Figure 3. Structure Difference Data of CPU and GPU

Under the framework of CUDA, a program is divided into two parts: the host and device ends. The host end is executing on the CPU part, and the device end is in the display chip implementation part. Device side program is also known as the "kernel". Usually the host client program data will be copied to the graphics card memory, and then by the display chip implementation of device client program, completed by the host side program will result from the graphics memory retrieval. Under the framework of CUDA, the smallest unit of execution time display chip is thread. The whole flow of algorithm implementation using CUDA model, the following is the complete steps to achieve the detector [7].

Image acquisition and preprocessing: the image is loaded into the CPU pre-processing, using integral image to complete the Haar feature value calculation, normalized, copy the integral image data and the weight information to GPU [8].

Downscaling and feature calculation: a kernel function was used in this step. In the kernel function, each thread corresponds to a pixel. In order to downscaling and

calculate the feature value, a lot of disorder operation on the sample data were needed, the integral image data were stored in the Texture Memory.

Cascade weak classifier computation phase [9]: this phase includes random memory read and write operation. The operation is not very suitable for CUDA model, here we have to know about several CUDA model type, Global Memory, Texture Memory and Shared Memory. Texture Memory is a read-only memory for data quantity of random data access is larger or non alignment access, also has good acceleration effect. The speed of Shared Memory has fast i/o operation. In the process of our experiment for different data with different storage methods.

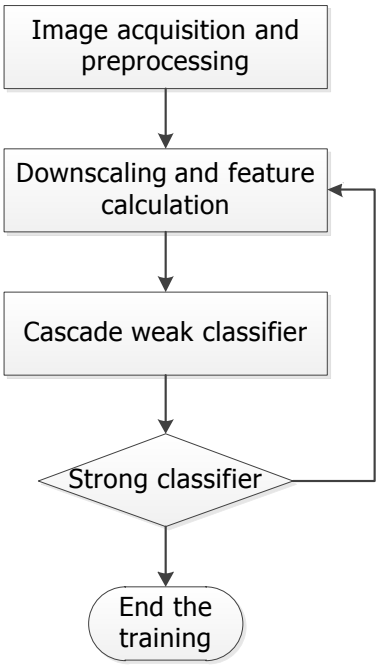


Figure 4. The Steps of Framework

5. Experiment and Result

INRIA is a database of pedestrian detection which is the most commonly used. It contains a variety of light environment. The training set has 2416 negative samples and 1218 positive samples. The testing set has 1126 positive number and 453 negative samples. All samples are normalized for size. In order to make up for the lack of training samples, the mirror image transforms processing to sample.

28320 characteristics were selected. Table 2 shows the time when training a weak classifier with training samples in different conditions. The number of samples is 5240. The time of training classifier is 22.2s in GPU platform, but when with CPU it's 33.3s. When the training sample number is 28320, the optimal acceleration training the whole classifier on the platform of GPU phase on CPU platform reaches more than 5.4 times.

Table 2. The Time for Calculate Final Classifier with Adaboost Algorithm

Sample Num	CPU(s)	GPU(s)	Speedup
5240	33.3	22.2	1.5
7460	56.3	25.6	2.2
9800	74.3	28.6	2.6
12400	107.1	30.6	3.5

16800	128.8	32.2	4.0
23040	147.4	34.3	4.3
28320	208.4	38.6	5.4

The test results show that CUDA model is adopted to train the optimal weak classifiers to form a strong final classification relative in real-time pedestrian detection than with the CPU, the performance is improved by nearly 5 times.

Accuracy: the image size for our experiment is 320×240. When detecting an image, we need testing them one by one from different scales and different positions. For the different scales detection, this choice of scaling of detector instead of scaling of the image itself. In any scale, characteristics can be obtained by the same situation. In the training, the window size is 16×32, every time 1.25 times has best detection.

The Caltech Pedestrian Dataset consists of approximately 10 hours of 640x480 30Hz video taken from a vehicle driving through regular traffic in an urban environment. In this work we use our method to detect pedestrian on this datasets.

The result shows in some videos the method gets the good results and less error detection. In some cases the method has error detection that detect the other things to be pedestrian. But the accuracy rate of this method is very high on real human.

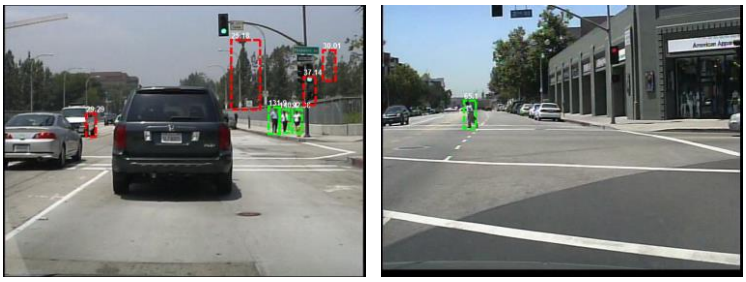


Figure 5. The Steps Results of Pedestrian Detection

In the process of experiment, there are some false positives and undetected images. Here we compare the detection results. In the GPU treatment and CPU treatment under the condition of AdaBoost algorithms, on the platform of GPU classifier, in the training process that speed is increased at the same time, keep the good generalization ability and strong robustness.

6. Conclusions

Firstly, a Haar-like feature was employed as the feature vector and integral image for downscaling. Adaboost algorithm was used to build the strong classifier, when the datasets become larger, the detection become time-consuming under the CPU architecture. CUDA framework was introduced to improve the speed of detector. The experimental results show when the sample size of 19×19, the number of samples for training was 28320, the accelerated ratio up to 5.4 times, efficient improved the speed of classification. By contrast the effect of training result with CPU and GPU. Under the classifier based on parallel Adaboost, we can see the training speed is increased under the CUDA framework, at the same time keep a better classifier effect. In the future work we focus on the load balance and data partition under CUDA framework to improve the detection.

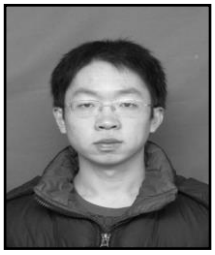
Acknowledgements

This paper is supported by Foundation of Science and Technology Commission of Shanghai Municipality under Grant No. 14590500500, Natural Science Foundation of Shanghai under Grant No. 15ZR1415200, and Young University Teachers Training Plan of Shanghai Municipality under Grant No. ZZSD13008. We gratefully acknowledge and thank those who provided comments and suggestions. The anonymous reviewers and the editor of this paper are also acknowledged for their constructive comments and suggestions. I'd like to express my sincere thanks to all those who have lent me hands in the course of my writing this paper.

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", Proceedings of the 2001 IEEE Computer Society Conference on. IEEE, **(2001)**, vol. 1, pp. 511-518.
[2] Y. Freund, R. Schapire and N. J. Abe, "A short introduction to boosting", Journal-Japanese Society for Artificial Intelligence, vol. 14, **(1999)**, pp.771-780.
[3] N. Dalal and B.Triggs, "Histograms of oriented gradients for human detection, Computer Vision and Pattern Recognition", IEEE Computer Society Conference on. IEEE, no. 1, **(2005)**, pp.886-893.
[4] P. Dollár, Z. Tu and P. Perona, "Integral Channel Features", British Machine Vision Conference, vol. 2, **(2009)**, pp.5-16.
[5] E. J. Tapia, "A note on the computation of high-dimensional integral images", Pattern Recognition Letters, vol. 32, no. 2, **(2011)**, pp.197-201.
[6] M. Garland, S. Le Grand and J. J. Nickolls, "Parallel computing experiences with CUDA", IEEE micro, no. 4, **(2008)**, pp. 13-27.
[7] B. Bilgic, B. K. P. Horn and I. Masaki, "Fast human detection with cascaded ensembles on the GPU", Intelligent Vehicles Symposium (IV), IEEE, **(2010)**, pp.325-332.
[8] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection", Image Proceedings, 2002 International Conference on. IEEE, vol. 1, **(2002)**, pp.900-903.
[9] J. Friedman, T. Hastie and R. J. Tibshirani, "Additive logistic regression: a statistical view of boosting", the annals of statistics, vol. 28, no. 2, **(2000)**, pp. 337-407.

Author



Chongchao Cai, is a PhD candidate in graduate students in School of Computer Engineering and Science, Shanghai University. His research interests include computer vision, machine learning, Pedestrian Detection.