- Invited paper -

Analysis of a GPU implementation of Viola-Jones' Algorithm for Features Selection

Germán Lescano[1,2], Pablo Santana-Mansilla[1,2] and Rosanna Costaguta[1]
[1]*Instituto de Investigación en Informática y Sistemas de Información (IIISI)*
Facultad de Ciencias Exactas y Tecnologías (FCEyT)
Universidad Nacional de Santiago del Estero (UNSE)
Santiago del Estero, CP 4200, Argentina
{gelescano,psantana,rosanna}@unse.edu.ar
[2]*Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)*

Abstract

Faces and facial expressions recognition is an interesting topic for researchers in machine vision. Viola-Jones algorithm is the most spread algorithm for this task. Building a classification model for face recognition can take many years if the implementation of its training phase is not optimized. In this study, we analyze different implementations for the training phase. The aim was to reduce the time needed during training phase when using one computer with a cheap graphical processing unit (GPU). The execution times were analyzed and compared with previous studies. Results showed that combining C language, CUDA, etc., it is possible to reach acceptable times for training phase. Further research may involve the measurement of the performance of our approach computers with better GPU capacity and exploring a multi-GPU approach.

Keywords: Adaboost, Viola-Jones Algorithm, feature selection, CUDA.

1. Introduction

Face and facial expressions recognition is an interesting topic for researchers in machine vision [9]. An important stage in a face recognition algorithm is the building of a classification model that can discriminate faces. Building a classification model require a training phase during which a sample of images is analyzed with the aim of extracting those features that best describe a face.

Viola and Jones [11] proposed an algorithm that can detect faces in real time. It is widely used in a variety of software and hardware applications that incorporate elements of computer vision, like the face detection module in video conferencing, human-computer interaction, and digital photo cameras [6]. This algorithm can be implemented on a wide range of small low power devices, including hand-helds devices and embedded processors. However, a drawback of this algorithm is that the training phase is extremely time-consuming.

In this work, we propose and analyze implementations for the training phase of Viola-Jones algorithm. These implementations try to reduce the execution times when working on a single computer and involves the use of parallel computing, specifically CUDA architecture. CUDA is a parallel computing platform and programming model created by NVIDIA [12]. It enables dramatic increases in computing performance by harnessing the power of the GPU. In order to reduce execution times, we had focus on feature selection because this process has a notable impact on training times.

This paper is organized as follows. In section 2, the training phase of Viola-Jones algorithm is described. In section 3, details about the proposed implementation are given. In section 4, we compare our proposal with other alternatives we found in the literature. In section 5, conclusions and future research directions are presented.

2. The Training Phase

Viola and Jones [11] propose a variant of Adaboost algorithm [2] for the training phase. This algorithm selects a small number of features that best describe a face. These features are known as weak classifier. Once weak classifiers are selected, the algorithm combine them to get a strong classifier.

In Viola-Jones algorithm, features are modeled like rectangles which are subdivided in black and white regions, a reminiscent of Haar basis functions [8]. Fig. 1 shows an example of typical features. To compute a feature, the sum of pixels within the white rectangles are subtracted from the sum of pixels within the black rectangles.

Fig. 1 Haar-Like patterns frequently employed [11].

Eq (1) shows the definition of a weak classifier. In

this equation f is a feature, θ is a threshold, $p \in \{-1,1\}$ is the polarity that indicates the direction of the inequality and x is a sub-window of an image.

$$h(x,f,p,\theta) = \begin{cases} 1 \; si \; pf(x) < p\theta \\ 0 \; en \; otro \; caso \end{cases} \quad \text{(Eq. 1)}$$

Each iteration of the training phase tries to select the single rectangle feature that best separates the positive (face images) and negative examples (not face images). For each feature, the weak classifier determines the optimal threshold of classification function, such that the minimum number of examples is misclassified. Fig. 2 shows the flow chart of the algorithm proposed by Viola and Jones [11].

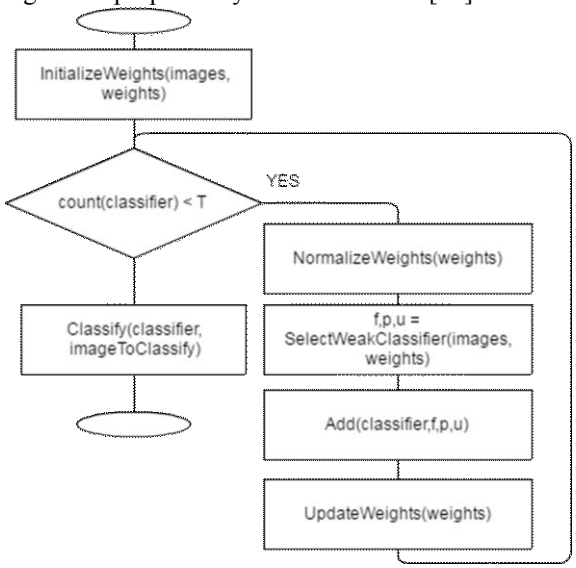

Fig. 2 Boosting algorithm of training phase.

Training phase begins with the initialization of weights related to training images. Weights of positives images are initialized with $1/2m$ and weights of negative images with $1/2l$, where m is the number of face images and l is the number of not face images. Then, the algorithm iterates to get the T weak classifiers that are required. The T number is given as a parameter. Within each iteration four processes are run. In the first process, weights are normalized with the function in Eq (2). In the second process, a weak classifier is selected. This means that this process determines the values of the feature, the polarity and the threshold that minimize the classification error defined in Eq (3). Fig. 3 shows a flowchart of the weak classifier selection process that was designed from the interpretation proposed by Morelli and Padovani [5] in their pseudo-code. In the third process, the weak classifier selected is added to the set of weak classifiers. In the last process, weights of images are updated again with the function in Eq (4).

Once the T is reached, the algorithm combines the chosen weak classifiers to get a strong classifier which output is showed in Eq (5).

$$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \quad \text{(Eq. 2)}$$

$$\varepsilon_t = min_{f,p,\theta} \sum_i w_i |h(x_i,f,p,\theta) - y \quad \text{(Eq. 3)}$$

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$
where e_i=0 if example x_i is classified correctly, e_i = 1 otherwise, and $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$ (Eq. 4)

$1 \; if \; \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T} \alpha_t$, where (Eq. 5)
$\alpha_t = log\frac{1}{\beta_t}$, or 0 otherwise

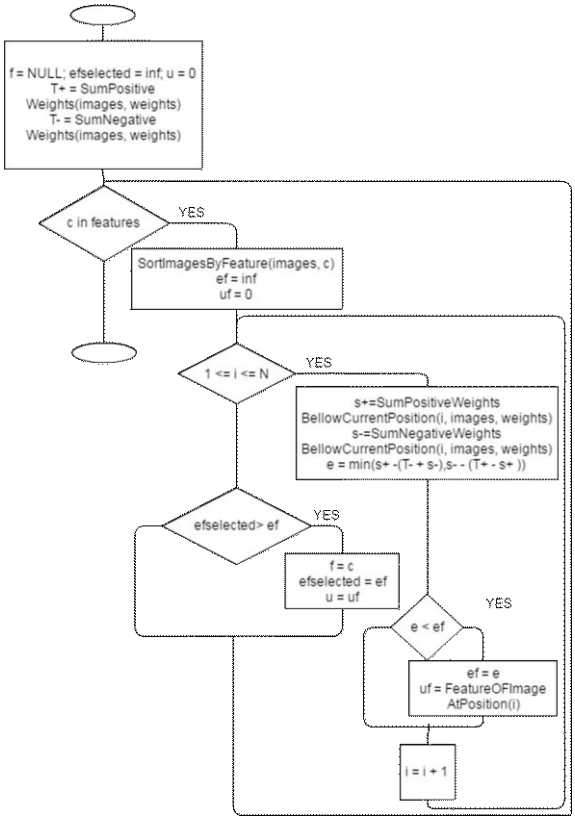

Fig. 3 Algorithm for weak classifiers selection.

In the process of selecting a weak classifier (Fig. 3), for each feature, the images into the samples are sorted by feature value in ascendant way. The Adaboost optimal threshold for that feature can then be computed in a single pass over this sorted list. On each iteration over the sorted list, four sums are evaluated for each element: the total sum of positive examples weights T+, the total sum of negative example weights T-, the sum of positive weights below the current example S+ and the sum of negative weights below the current example S-. Furthermore, an error is computed for each feature using the Eq (6).

This value represents the error that we would produce if the element were considered the threshold for the feature. Once all errors have been computed, the lowest one is selected.

$$\varepsilon = \min(S^+ + (T^- - S^-), S^- + (T^+ - S^+)) \quad \text{(Eq. 6)}$$

3. Implementing Training Phase on a GPU

Five experimental settings were proposed to implement the training phase of Viola-Jones algorithm. These implementations differ in terms of data allocation and key functions processing (data sorting for example). As four out of five implementations needed execution times that surpass 2 hours, we decided to focus on the fifth implementation. Table 1 summarizes implementation one to four and shows execution time that they needed to choose a weak classifier. More details of first to fourth implementation can be found in [14].

In this section, we describe an implementation of the training phase of Viola-Jones algorithm in order to be executed in a cheap GPU. C language was chosen to code this implementation because it enables us to make decision of programming at low-level and it is compatible with CUDA. Execution time of the implementation was evaluated throughout the processing of 5000 images of faces and 10,000 images of not faces. This quantity is similar to the experiment of Viola-Jones. The images used in our experiments were simulated from a seed of face images (165) and not face images (165).

The weak classifier selection is a function that is executed several times in the training phase and has a lot of data processing. In the flowchart of Fig. 3, we can see that the function begins computing the sum of weights corresponding to the 15,000 positive and negative images. Then the first loop iterates by 162,336 features. Each interaction of the loop gets, from the training images, the features values corresponding to the feature analyzed and sorts them. Then, the second loop is executed with the aim of computing the sum of positive weights below the current example S+ and the sum of negative weights below the current example S-.

Table 1 Summary of implementations reported in [14]

Implementation 1	**Description.** - Data are retrieved from files. - There are about 2500 millions of hard drive accesses. - Sort method: Bubble
	Execution time: 4.84 years
Implementation 2	**Description** - Information about 162336 feature of each training file are stored in a unique file. - Sort method: QuickSort - All operations are sequential.
	Execution time: 31.42 hs
Implementation 3	**Description** - Thrust for sorting task. - Reduction Thrust was used to compute operations within the second loop in Fig. 3 - Data transference between RAM and GPU memory within the second loop in Fig. 3
	Execution time: 17.08 days
Implementation 4	**Description** - Information about weights and class associate with an image are allocated in GPU memory until the first loop in Fig. 3 was true. - Harris Algorithm [3] was used to parallelize sum operations within the second loop in Fig. 3.
	Execution time: 23.43hs

In order to store information of training images we decided to use a 15,000 x 4 matrix. This matrix is allocated in memory and has the following data for each image: the identifier of each training file, the category of the image (face or not face), its weight and the category (face or not face) assigned by the classifier algorithm. In addition, we used as many files as training images in order to save the features of the images. Each file has 162,336 lines (this number corresponds to the total quantity of features that can be generated for an image of 24 x 24 pixels) and each line registers information of one feature. Specifically, each line indicates the kind of the feature (someone of the five types showed in Fig. 1), x

position of sub image, *y* position of sub image, weight of sub image, height of sub image, value of the feature (difference between white rectangles and black rectangles), and its identification.

When training process starts a 15,000 x 162,336 matrix is created. Each line in this matrix stores the sixth column of each line of a file with information of image features. This step allowed us to reduce the execution time. To be sure data were not paged by the operative system we used cudaHostAlloc instruction that allocates a buffer of page-locked host memory [7].

We used the Thrust library for sorting the matrix of image features by the column corresponding to the feature that is selected in the first loop showed in Fig. 3. Thrust is a parallel algorithms library that enhances programmer productivity while enabling performance portability between GPUs and multicore CPUs [13].

Table 2 Execution times required to get a weak classifier

Number of Streams	Execution Time (seconds)
16	0.0517
32	0.0525
64	0.0532
128	0.0534
256	0.0538
512	0.0543
1,024	0.0535
2,048	0.0532
4,096	0.0534
8,192	0.0444

Streams were used to improve the performance of the algorithm. These streams compute the operations involved in the second loop showed in Fig. 3. In each iteration of the loop, a stream loads in asynchrony way, from the memory of CPU to the memory of GPU, each row of matrix of image features. While this data is loaded, two kernels are thrown in order to compute data available in GPU memory. One kernel sorts the loaded row by the feature which is analyzed. The other kernel runs operations involved in the second loop. The results of these calculations are copied asynchronously to an array that stores, for each feature, the error that would be committed if the value in the sample for this feature were chosen as threshold value. This array is stored in pinned memory.

At the end of the execution of all the streams, the min_element function of Thrust library is used to find the smallest error in the above-mentioned array. This final step allows one to get the selected feature, the threshold value and its corresponding error.

Table 2 shows the execution times to process a feature using 15,000 training images and different numbers of streams. We employed a computer with the following hardware configuration: i7 processor of 3.4 Ghz and GPU GeForce GT 730 with 2 GB of global memory.

With this implementation and using 8,192 streams, the processing of 15,000 training image with 162,336 features will take about 2hs.

4. Discussion

A bibliographical exploration regarding to time reduction when building images classifiers with boosting algorithm allowed us to identify research works such as Huang and Shi [4], Abualkibash et al. [1] and Tsai et al. [10].

Huang and Shi [4] worked with 65,230 features and 18,676 samples. In their experiments, Huang and Shi used computers with a 1.8 Ghz processor. Abualkibash et al. [1] describe the same experiment that was made by Huang and Shi [4], yet Abualkibash et al. [1] utilized computers equipped with quad-core processors. Details about processing capacity of CPUs are not given in Abualkibash et al. [1]. Fig.4 shows a comparison of execution time obtained during fifth experimental setting (described in 3) with execution time during experiments conducted by Huang and Shi [4] and Abualkibash et al. [1] (further details are given in [14]).

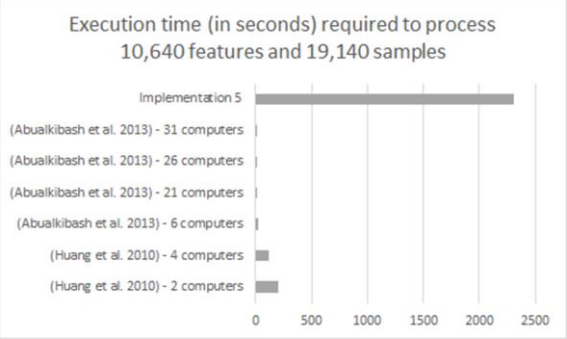

Fig. 4 Execution time during experiments conducted by Huang and Shi [4], Abualkibash et al. [1] and alternative *5*

Regarding to Tsai et al. [10], they used a GPU Nvidia Tesla K20c. This GPU has 2496 cores, 706 Mhz of memory frequency, and 5 GB of global memory. Tsai et al. [10] made three experiments: the first one with 1,119 features and 19,575 samples; the second experiment with 6,090 features and 19,161 features; and the last experiment with 10,640 features

and 19,140 samples. The results of these three experiments can be seen in Table 3.

Table 3 Comparison between execution time of alternative 5 and execution time during experiments conducted by Tsai et al. [10]

Experiments	Time to select a feature	
	Tsai et al. [10]	Alternative 5 implemented in Geforce GT 730 of 2GB memory
1,119 features 19,575 samples	0.788 sec	1.413 min (1,119 streams)
6,090 features 19,161 samples	1.157 sec	7.313 min (4,096 streams)
10,640 features 19,140 samples	1.217 sec	6.2497 min (8,192 streams)

Considering results that are showed in Fig. 4 and Table 3 one could conclude that the fifth experimental setting proposed in this paper is not a good option to reduce the time of classifiers building. Nevertheless, one should consider that the fifth alternative of solution was tested in a computer with a GPU of limited processing capacity. For this reason, it would be interesting to run the fifth solution in a computer with better hardware capacities and then compare it with the results achieved by Huang and Shi [4], Abualkibash et al. [1] and Tsai et al. [10]. For example, a GPU like the one used by Tsai et al. [10] not only has more computing capacity but also it is possible to execute more streams.

As the laboratory at the university had not have more powerful computers than the described in point 4 (i7 processor of 3.4 Ghz and GPU Geforce GT 730), it was decided to hired the computer instance g2.2xlarge from amazon.com. The instance g2.2xlarge has: 8 virtual CPU Intel Xeon E5-2670; 15 GB of RAM; and a GPU Nvidia Grid K520 with 4 GB of RAM, 797 Mhz of frequency and 1536 cores.

Table 4 shows a comparison of time needed for executing the experiments of Table 2 (the same number of features and sample files) in the hired instance and the GPU GeForce GT 730. It was expected that execution in the GPU available in g2.2xlarge instance would be faster than the execution in GeForce GT 730. Nonetheless, the results do not confirm the previous assumption. A reason might be the virtualization effect although this needs further research.

Table 4 Comparison of execution time when implementing solution 5 in GeForce GT 730 and in instance g2.2xlarge.

Experiments	Time to select a feature (Alternative 5 implemented in GeForce Gt 730)	Time to select a feature (Alternative 5 implemented in GPU of g2.2xlarge)
1,119 features 19,575 samples	1.413 min	4.1789 min
6,090 features 19,161 samples	7.313 min	22.039 min
10,640 features 19,140 samples	6.2497 min	38.472 min

5. Conclusions

This work analyzed several implementations of training process for generating a classifier with the capacity of face recognition. The aim was to reduce the time needed to train the classifier using a single computer. The focus was the process for selection of weak classifiers because this stage is the most invoked during classifiers building and it is the most demanding in terms of execution time. With the several alternatives implemented, sequential and parallel through CUDA architecture, it was possible to achieve substantial improvements.

The use of GPUs for developing and accelerating applications is a feasible alternative because here are cheap GPU in the market. In addition, there is an effort made by manufacturer of GPU for developing software that helps programmers to use GPUs, for example NVIDIA with her CUDA platform. With the use a GPU, applications can compute many data without wasting too much time.

Further research may involve the measurement of the performance of fifth experimental setting in a computer with bigger computing capacity in its GPU and exploring a multi-GPU approach. Collaterally, we believe that further research is needed for evaluate if virtualization affects the GPU performance.

References

[1] Abualkibash, M.; ElSayed, A.; Mahmood, A.: Highly scalable, parallel and distributed Adaboost algorithm using light weight threads and web services on a network of multi-core machines. International

Journal of Distributed and Parallel Systems (IJDPS), 4(3): 29-40, May 2013.

[2] Freund, Y.; Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1): 23-37. August 1997.

[3] Harris, M.: Optimizing parallel reduction in CUDA. Reporte técnico. 2007. Disponible en: http://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf

[4] Huang, Z.; Shi, X.: A distributed parallel AdaBoost algorithm for face detection. 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS). Vol 1: 147-150, Oct 2010.

[5] Morelli A., Padovani S.: Detección y Reconocimiento de Cara. Tesis de Licenciatura en Ciencias de la Computación. Universidad de Buenos Aires. 2011.

[6] Obukhov, A.: Haar Classifiers for Object Detection with CUDA. In: Wen-Mei W. Hwu (Ed.), GPU Computing Gems. 517-544. Burlington, MA 01803, USA, 2011.

[7] Sanders, J.; Kandrot, E.: CUDA C on multiple GPUs. In: CUDA by Example. An Introduction to General-Purpose GPU Programming. 213-236. Boston, MA 02116, USA, 2011.

[8] Papageorgiou, C.; Oren, M.; Poggio, T.: A general framework for object detection. International Conference on Computer Vision. 555-562, 04 January - 07 January, 1998.

[9] Taheri, S.; Patel, V.; Chellappa, R.: Component-Based Recognition of Faces and Facial Expressions. IEEE Transactions on Affective Computing, 4(4): pp. 360-371, October-December 2013.

[10] Tsai, P.; Hsu, Y.; Chiu, C; Chu, T.: Accelerating AdaBoost algorithm using GPU for multi-object recognition. 2015 IEEE International Symposium on Circuits and Systems (ISCAS), 738-741, May 2015.

[11] Viola P., Jones M.: Robust Real-Time Face Detection. International Journal of Computer Vision, 57(2): 137–154, May 2004.

[12] NVidia CUDA technology, http://www.nvidia.com/object/cuda_home_new.html

[13] Thrust, http://thrust.github.io/

[14] Lescano, G.; Santana-Mansilla, P.; Costaguta, R.: Experiences accelerating features selection in Viola-Jones algorithm. XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016), Nov 2016