Scalable Feature Extraction for Visual Surveillance

Antonio C. Nazare Jr., Renato Ferreira, William Robson Schwartz

Department of Computer Science, Universidade Federal de Minas Gerais,
Av. Antônio Carlos, 6627, Pampulha, Belo Horizonte, Brazil
{antonio.nazare,renato,william}@dcc.ufmg.br

Abstract. The availability of surveillance cameras placed in public locations has increased vastly in the last years, providing a safe environment to people at the cost of huge amount of visual data collected. Such data are mostly processed manually, a task which is labor intensive and prone to errors. Therefore, automatic approaches must be employed to enable the processing of the data, so that human operators only need to reason about selected portions. Aiming at solving problems in the domain of visual surveillance, computer vision techniques have been applied successfully for several years. However, they are rarely tackled in a scalable manner. With that in mind, in this paper we tackle the feature extraction problem, one of the most expensive and necessary tasks in computer vision, by proposing a scheme to allow scalable feature extraction that uses the full power of the multi-core systems.

Keywords: Visual surveillance, scalable feature extraction, Smart Surveillance Framework.

1 Introduction

Visual surveillance in public areas have been widely employed in large cities in the recent years, providing a safer environment. To be able to cover important parts of a city properly a large number of cameras must be deployed and monitored by human operators. However, such task requires focused attention for extended periods of time, rendering it unsuitable for humans [10]. Therefore, the employment of automatic techniques to detect and understand relevant activities taking place in the scene is essential to being able to process continuously increasing amounts of data.

In visual surveillance, a sequence of problems have to be solved before one is able to analyze activities being performed by humans in a video [1]. Among them are feature extraction [11], background subtraction [16], pedestrian detection [6], tracking [22] and re-identification [7], face recognition [23], gesture recognition [13] and action recognition [17].

In this work we propose a scheme for executing feature extraction so that it can leverage the processing power of parallel systems. As feature extraction is central in many of the processing steps, our mechanism also includes a caching

mechanism which allows the reuse of previously computed features. Our approach is implemented as the *Feature Extraction Server* (FES), developed on top of the publicly available *Smart Surveillance Framework* (SSF) [14], which allows researchers to implement their solutions to surveillance problems as a sequence of processing modules that communicate through a shared memory.

The proposed FES provides a number of advantages regarding the usage of the available computational resources (e.g., multiple cores) and the implementation point of view. 1) the user can implement his/her own feature extraction methods that the FES will distribute the processing according to the computational power at hand or according to the parameter setting chosen by the user; 2) it allows users to develop novel feature descriptors and evaluate them easily on problems such as detection and recognition; and 3) this centralize approach based on a server to extract features allows the caching of features vectors so that several modules might share the same vectors for different purposes.

Experimental results demonstrate the improvements achieved when the feature extraction is parallelized to use the multiple core processors. In addition, a vast improvement can be achieved by caching the feature descriptors. Finally, we also show that for some feature extraction methods it is not worth caching the descriptors due to time consumed with the cache indexing.

2 Related Work

This section presents an overview of traditional local features for surveillance based on computer vision and optimized techniques for their extraction. It also describes, briefly, some surveillance systems developed by the research community and the approaches they employ to the problem of extracting features, comparing them with the approach proposed in this paper.

There are many papers on features extraction methods. Li and Allinson [11] present a comprehensive review of several types of features, focusing on local region detectors and local descriptors. Among the several known feature extraction methods, we can mention few relevant methods. (*i*) *Scale Invariant Feature Transformation (SIFT)* [12] - a local image region is divided into a grid (*i.e.* 4×4 and a gradient orientation histogram is computed for each cell of the grid; (*ii*) *Histogram of Oriented Gradients (HOG)* [3] - a histogram of location and orientation of image gradients is constructed and used as feature vector; (*iii*) *Gray-Level Co-occurrence Matrix (GLCM)* [9] - the occurrence of pairs of pixel intensities is tabulated in a matrix, from which statistical measures are computed and used as feature descriptors.

Researchers have also devoted their studies to optimize the feature extraction. One of the early works was proposed by Viola and Jones [20], the integral image, an intermediate representation that allows faster computation of rectangle features. Dollar *et al.* [5] proposed linear and non-linear transformations to compute multiple registered image channels, called Integral Channel Feature. Authors employed these descriptors into their ChnFtrs detector achieving state-of-the-art results in pedestrian detection. Based on their previous work on Integral Channel

Feature, Dollar *et al.* [4] proposed a feature extraction that exploits the interpolation of features in different image scales, significantly reducing the cost and producing faster detectors when coupled with cascade classifiers.

In the recent years, many surveillance systems were designed and developed. Most works in the literature describe systems specialized in a certain functions. For instance, Xia *et al.* [21] focus on wide-area traffic monitoring for highway roads and Odobez *et al.* [15] designed a metro station monitoring system.

Differently from the specialized systems, the framework proposed in [14] is classified as a general purpose system since its allows different configurations to solve specific surveillance tasks. Others known general purpose systems include the Knight [18], a commercial product which is a fully automated with multiple surveillance cameras and monitoring system that detects, categorizes and tracks moving objects; the IBM Smart Surveillance System (S3) [19], which is among the most advanced surveillance systems nowadays, providing various capabilities as automatic monitoring of a scene and manage the surveillance data.

Feature extraction is critical for surveillance systems since several algorithms require feature descriptors as input. However, most feature extraction algorithms are highly time consuming and not suitable for real time applications. Thus, an aspect that differentiates the SSF from all the listed systems is the Feature Extraction Server (FES), which allows the feature extraction to be performed using the entire computational power available in the system to maximize the performance (one can use all available CPU cores). In the others systems mentioned earlier, the feature extraction process receives no special treatment, essential to large scale surveillance systems.

3 Feature Extraction Server

As pointed out earlier, feature extraction is required to solve several problems in surveillance and since such problems have to be solved simultaneously to make inferences regarding the suspicious activities captured by the cameras, it must be efficient. However, even though local feature extraction methods have been proposed [5, 20], it is still a time consuming task, which is further aggravate by the presence of multiple cameras. For that, we present the *Feature Extraction Server* (FES), a runtime framework which allows leveraging of modern parallel architectures for aiming to increase the performance of such methods.

According to [14] and as illustrated in Figure 1, the SSF is divided into parts: modules and kernel. While the former allows the users to develop their computer vision applications, the latter is responsible for the communication among modules and provides the Feature Extraction Server (FES), described in this work, and the Complex Query Server (CQS), responsible for providing querying capabilities for the user (e.g., retrieve video frames containing people wearing a given color of clothing). This work focuses on the FES by describing its features and evaluating the gain provided by its employment.

The FES relies on an asynchronous approach to receive requests, process them and return feature vectors to modules to maximize the occupancy of the

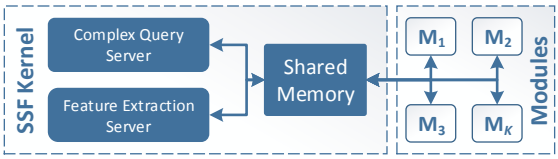


Fig. 1: Architecture of the Smart Surveillance Framework as proposed in [14].

processing units available. Once a request is sent to the FES, it does not block the processing being executed in the module, which can continue working while the request is been processed by the FES. For instance, the module might be processing the feature vectors already extracted while others are being extracted. Therefore, not all features vectors need to be stored in memory before processing, preventing from high memory consumption. In fact, the maximum amount of allocated memory can be set to avoid the process from using the virtual memory.

Figure 2 illustrates the main components of the feature server: *request control*, *extraction method* and *feature extraction memory*. Using the FES, a feature extraction request is performed as follows. First, a module sends extraction requests by passing the image regions from which the features will be extracted by a given method. Such requests are sent to a queue in the *request control*, which allows the module to make all requests for an image and continue its processing while the features are extracted. Then, the request control selects the extraction method chosen by the module and forward the requests to the *extraction method*, which process them using N instances, but first checks in the *feature extraction memory* the availability of memory, if there is not memory available, the extraction waits until some memory has been released. Finally, once the feature extraction is completed, the feature vector is pushed to the output queue and is ready to be retrieved by the requesting module.

The *request control* is responsible for screening the requests made by the modules. It is composed of an input queue and is aware of the feature extraction methods available. Once a request enters the queue, the request control forwards it to the correct feature extraction method. The request control is useful in the sense that the feature extraction becomes centralized, such that two modules requiring the same feature extraction method will use the same instance of the extraction method, which will allow the usage of cached features if two modules request feature extraction for the same image region.

The *extraction method* controls the feature extraction for a specific feature descriptor, such as HOG, GLCM and others [11]. When the extraction method receives a request, it first verifies in the cache if the same request had been made before and the feature descriptors are available, if so, return them, otherwise it checks in the *feature extraction memory* whether there is memory available in the feature extraction memory. This procedure allows the FES to set a limit of memory that can be used for the feature extraction process, otherwise the entire memory available in the machine could be consumed quickly compromising the execution. If there is no memory available, the extraction method is blocked until some memory is released, otherwise, it sends the request to one of its instances to perform the actual feature extraction for an image region. The descriptors

extracted by the instance are stored in the cache. Experiments show that the usage of cache reduces greatly the computational cost for feature extraction.

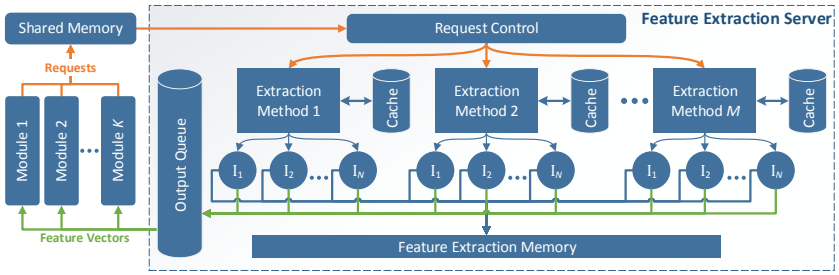


Fig. 2: Feature Extraction Server (FES) and its interface with a module.

4 Experimental Results

This section describes the experiments conducted to demonstrate the performance of the Feature Extraction Server (FES). For that were implemented and executed three traditional methods of features extraction: pixel intensity, histogram of oriented gradients (HOG) and gray-level co-occurrence matrix (GLCM). Even though there are many feature extraction methods, we have chosen these three methods because they present different computational cost and memory consumption, allowing us to evaluate different aspects of the FES.

The experiments consist in extracting feature descriptors of an image with a resolution of 640×480 pixels, using the aforementioned methods. To represent a realistic scenario, we employ the sliding window algorithm [8], widely used in object detection, to sample the image regions from which the feature descriptors are extracted. This algorithm works by exhaustively scanning an input image to generate a set of coordinates of several detection windows in multiple scales. For this work, we follow the block setup used in [3], in which each detection window is split into 105 blocks and we set the stride and scales parameters to generate a total of $48,495$ detection windows per image. We evaluate the FES regarding two aspects: the performance of parallel feature extraction and improvements obtained by the cache memory in the feature extraction.

All experiments are performed using the following machine: a Intel® Xeon™ 2.30GHz processor with 6 cores (12 cores on Hyper Threading mode and 32GB of main memory.

Number of Instances. To demonstrate the performance of parallelism provided by FES, we conducted experiments based on the number of instances used in the extraction. Each experiment consisted in the execution of a method repeated ten times and varying the number of instances from 1 to 12.

As shown in Table 1, one can observe an improvement in the computational performance, proportional to the number of instances used in the FES, which demonstrates the advantage of its usage in multi-core environments. The GLCM method showed a proportional reduction in run time on all experiments, while

Inst.	GLCM	HOG	Intensity	Inst.	GLCM	HOG	Intensity
1	341.26 ± 1.14	17.71 ± 0.12	11.66 ± 0.08	7	58.62 ± 0.22	5.62 ± 0.02	2.98 ± 0.12
2	177.93 ± 1.01	9.67 ± 0.19	5.96 ± 0.07	8	56.04 ± 0.21	5.73 ± 0.01	2.99 ± 0.14
3	120.68 ± 0.65	7.20 ± 0.12	4.12 ± 0.07	9	54.38 ± 1.21	5.88 ± 0.02	2.97 ± 0.11
4	91.82 ± 0.46	5.85 ± 0.05	3.23 ± 0.06	10	51.15 ± 0.15	6.02 ± 0.03	2.94 ± 0.13
5	74.37 ± 2.81	5.37 ± 0.07	3.06 ± 0.14	11	49.02 ± 0.08	6.25 ± 0.04	2.95 ± 0.09
6	61.64 ± 0.13	5.38 ± 0.05	3.03 ± 0.14	12	47.23 ± 0.13	6.55 ± 0.04	2.91 ± 0.02

Table 1: Computation time obtained for the feature extraction as a function of the number of extraction instances (in seconds).

for the other two methods, HOG and intensity, it was only observed up to four instances. In the HOG case, there is a slightly increase in the run time starting from six instances. This is because the computational complexity of HOG and intensity is smaller, hence there is an overhead caused by the FES, starting at four instances. This behavior can be explained by the Amdahl's Law [2]. This law states that a fraction of sequential operations, even in small numbers, can significantly limit the speedup achieved by a multi-core computer.

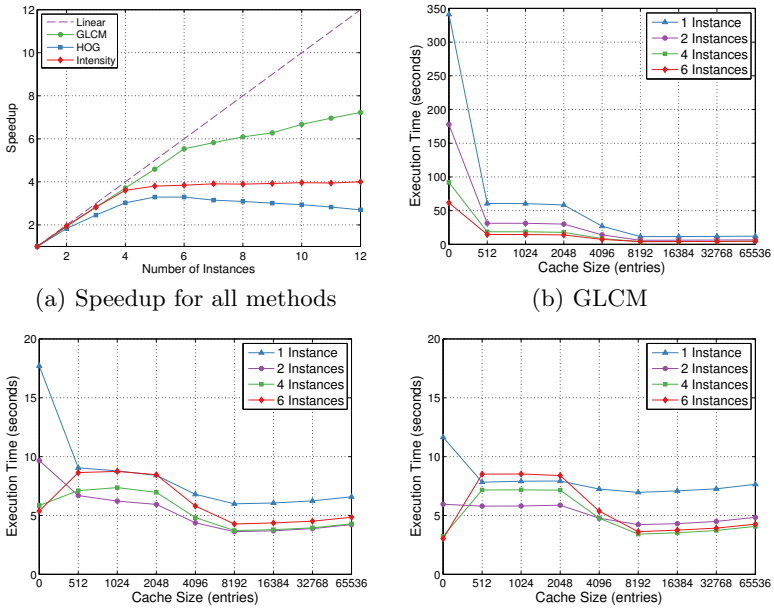
Figure 3(a) shows the speedup obtained from the values of Table 1. The GLCM's speedup has a linear growth up to six instances, which demonstrates the FES scalability for computationally expensive methods. Starting from seven instances, the speedup begins to decay due to the machine we used for our evaluation, which had only 6 physical cores as previously reported. For the HOG and intensity methods, the speedup presented a linear growth up to only four instances due to the overhead present in the FES that is more evident when the method is not very computational expensive.

Cache Size. This set of experiments aim at showing the performance gain obtained when the cache memory is used for the feature extraction method and when its size is increased. We developed experiments where each extraction method is individually executed for a different cache with at most C entries, where $C \in \{0, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536\}$ by varying the number of instances in 1, 2, 4, and 6. Each test was executed ten times. The results are shown in Figures 3(b), 3(c) and 3(d).

Figure 3(b) shows a significant reduction in time for the GLCM method without using cache (near 80% for a cache of size 512), for every number of instances. The improvement is also observed for 1024, 2048, 4096, and 8192 cache size. However, starting from 16834 entries, the runtime does not decrease. This is because the number of extracted features is not enough to fill the entire cache, compromising the spatial locality of the memory.

The cache utilization also significantly contributed to the performance of HOG and intensity. However, this contribution is only observed when the HOG is performed in one or two instances, and in the case of intensity, the improvement is observed only for the execution of the method with a single instance, although two instances yields only a slightly reduction in the run time.

Unlike the previous results, experiments with 4 and 6 instances for HOG and intensity increased the run time due to the overhead caused by competition for access to the cache, since the low computational cost of the methods yields the instances to quickly compute the features and consequently making them wait



(a) Speedup for all methods (b) GLCM

(c) HOG (d) Intensity

Fig. 3: (a) Speedup achieved as a function of the number of instances. The dashed line indicates the ideal linear speedup; (b,c,d) Computation time with the addition of cache memory with multiple sizes (maximum number of entries).

to have write access to the cache. One may also notice a small increase in run time for cache values above 8192, which we believe is also caused by poor spatial locality of the memory.

5 Conclusions

In this paper we presented the Feature Extraction Server (FES) able of receiving feature extraction requests, process them and return feature vectors to modules to maximize the occupancy of the processing units available. Results demonstrated that we could achieve almost linear speedup, provided that the method is sufficiently costly and that enabling cache decreases by 80% the runtime.

Acknowledgments

The authors would like to thank the Brazilian National Research Council – CNPq (Grant #487529/2013-8) and the Minas Gerais Research Foundation - FAPEMIG (Grant APQ-01806-13).

References

1. Aggarwal, J., Ryoo, M.: Human Activity Analysis: A Review. ACM Comput. Surv. 43(3), 1–43 (2011)

2. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18-20, 1967, spring joint computer conference. pp. 483–485. AFIPS '67 (Spring) (1967)
3. Dalal, N., Triggs, B.: Histograms of Oriented Gradients for Human Detection. In: IEEE CVPR. pp. 886–893 (2005)
4. Dollár, P., Belongie, S., Perona, P.: The Fastest Pedestrian Detector in the West. In: BMVC (2010)
5. Dollár, P., Tu, Z., Perona, P., Belongie, S.: Integral Channel Features. In: BMVC (2009)
6. Dollar, P., Wojek, C., Schiele, B., Perona, P.: Pedestrian Detection: An Evaluation of the State of the Art. IEEE TPAMI 34(4), 743–761 (2012)
7. Doretto, G., Sebastian, T., Tu, P., Rittscher, J.: Appearance-based Person Reidentification in Camera Networks: Problem Overview and Current Approaches. Journal of Ambient Intelligence and Humanized Computing 2, 127–151 (2011)
8. Forsyth, D.A., Ponce, J.: Computer Vision: A Modern Approach. Prentice Hall Professional Technical Reference (2011)
9. Haralick, R., Shanmugam, K., Dinstein, I.: Texture Features for Image Classification. IEEE Transactions on Systems, Man, and Cybernetics 3(6) (1973)
10. Keval, H.: CCTV Control Room Collaboration and Communication: Does it Work? In: HCTW. pp. 1–4 (2006)
11. Li, J., Allinson, N.M.: A Comprehensive review of Current Local Features for Computer Vision. Elsevier Neurocomputing 71(10-12), 1771–1787 (2008)
12. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. IJCV 60, 91–110 (2004)
13. Mitra, S., Acharya, T.: Gesture Recognition: A Survey. IEEE TSMC : Part C 37(3), 311–324 (2007)
14. Nazare, A.C., dos Santos, C.E., Ferreira, R., Schwartz, W.R.: Smart Surveillance Framework: A Versatile Tool for Video Analysis. In: IEEE Winter Conference on Applications of Computer Vision (2014)
15. Odobez, J.M., Carincotte, C., Emonet, R., Jouneau, E., Zaidenberg, S., Ravera, B., Bremond, F., Grifoni, A.: Unsupervised Activity Analysis and Monitoring Algorithms for Effective Surveillance Systems. In: ECCV (2012)
16. Piccardi, M.: Background subtraction techniques: a review. In: IEEE SMC, Part C. vol. 4, pp. 3099–3104 (2004)
17. Poppe, R.: A survey on vision-based human action recognition. Elsevier IVC 28(6), 976–990 (2010)
18. Shah, M., Javed, O., Shafique, K.: Automated Visual Surveillance in Realistic Scenarios. IEEE Multimedia 14(1), 30–39 (2007)
19. Tian, Y.l., Brown, L., Hampapur, A., Lu, M., Senior, A., Shu, C.f.: IBM smart surveillance system (S3): event based video surveillance system with an open and extensible framework. In: IEEE AVSS (2008)
20. Viola, P., Jones, M.: Rapid Object Detection using a Boosted Cascade of Simple Features. In: IEEE CVPR (2001)
21. Xia, J., Rao, W., Huang, W., Lu, Z.: Automatic multi-vehicle tracking using video cameras: An improved CAMShift approach. KSCE Journal of Civil Engineering 17(6), 1462–1470 (Aug 2013)
22. Yilmaz, A., Javed, O., Shah, M.: Object Tracking: A Survey. ACM Computing Surveys 38(4) (2006)
23. Zhang, X., Gao, Y.: Face Recognition Across Pose: A Review. PR 42(11), 2876–2896 (2009)