Design of Parallel Architectures of Classifiers Suitable for Intensive Data Processing

Bogusław Cyganek
AGH University of Science and Technology
Department of Electronics
Krakow, Poland
cyganek@agh.edu.pl

Kazimierz Wiatr
AGH University of Science and Technology
Academic Computer Center Cyfronet
Krakow, Poland
wiatr@agh.edu.pl

***Abstract*—Processing of visual data, such as object recognition and image segmentation, is based on data classification. In this paper architectures of ensembles of classifiers are discussed which show superior accuracy in respect to a single classifier. To achieve comparable response time the parallel computer architectures need to be considered, however. In the paper we present a parallel implementation on a graphic card of an ensemble of one-class support vector machines for image segmentation. We show that the parallel architecture of the ensemble of classifiers allows both, the high accuracy and speed up factor of two orders of magnitude compared with the serial software implementation.**

Keywords-ensemble of classifiers; OC-SVM; data processing; GPU implementation; image segmentation

I. INTRODUCTION

Data processing highly relies on classification methods. The two key parameters of the classifiers are their accuracy and response time. However, a choice of a right classifier to a given task depends on a number of factors, such as a number of training data, their type as well as data dimensionality, and frequently is a matter of the skills and experience of a system designer [7][15][18]. The choice of the right classifiers is especially important in the time critical and massive data processing systems. To this group belong vision processing systems addressed in this paper. They are frequently used in such tasks as face and gesture recognition, public area surveillance, driving assistance, and many more.

In this paper implementation issues of different architectures of ensembles of classifiers employed to the tasks of image processing are discussed. Such ensembles show superior accuracy compared to a single classifier, as reported in the literature [3]-[5][7]-[11][19]. However, operation of many classifiers leads to excessive response time of such systems if implemented in serial software. A solution to this problem is parallel operation of the member classifiers which is possible and effective if their operations are independent as much as possible.

In this paper a number of parallel architectures of classifiers is discussed which allow functional and data decomposition of the classification problem and which aim at full utilization of the multi-core processors, graphic cards (GPU), as well as field programmable gate arrays (FPGA) [20]. Our discussion is exemplified with the ensemble of one-class support vector machines (OC-SVM) applied to image segmentation, which is based on our previous work [3][4]. In this paper, we present its parallel implementation on the graphic card which allows real-time operation. We also show that simple application of the data splitter and then a group of cooperating classifiers usually leads to a better performance compared to a single classifier trained with all available training data.

The paper is organized as follows: In Section II, details of the architecture of ensembles of classifiers are discussed. In Section III, details of the data splitter and base classifiers are provided. In Section IV, details of the parallel implementation are discussed. Finally, the paper ends with experimental results discussed in Section V, as well as with conclusions presented in Section VI.

II. ARCHITECTURE OF ENSEMBLES OF CLASSIFIERS

In this section, we discuss two of the most popular architectures of ensembles of classifiers: the serial and the parallel ones. Architecture of the serial cascade of classifiers is shown in Fig. 1. Input data is processed in the *pipeline* like structure. Each classifier filters out data, passing the positively classified ones to the next one in the chain, and so on.

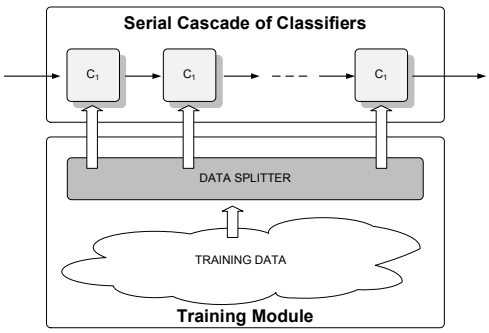

Figure 1. Architecture of the serial cascade of classifiers. Input data is processed in the pipeline structure. Each classifier filters out data, passing the positivly classified ones to the next in the chain, and so on. Training is done with the AdaBoost to amplify response on poorly classified examples.

An example of such a system is the face detection method by Viola and Jones [19]. Training is done with the

 ISBN: 978-1-61208-258-5

AdaBoost which amplifies response on poorly classified examples. Such strategy imposes data decomposition into sets of usually decreasing number of elements [9].

Data processing in a serial chain of classifiers is effective if member classifiers are able to operate in a pipeline mode. One of the requirements in this case is that each classifier in the chain consumes the same time quant for data processing. The penalty of using a cascade is a delay necessary to fill up the processing line which is proportional to the number of used classifiers. However, in practice these requirements are not easy to fulfill.

On the other hand, architecture with a parallel ensemble of classifiers is depicted in Fig. 2. In this case all classifiers are assumed to operate independently which is a big advantage considering implementation and execution time. However, all partial responses need to be synchronized and collected by the answer fusion module which outputs a final response. There are different methods of training of the member classifiers C_i, shown in Fig. 2, as is discussed in many publications [7]. Some of the most popular are data bagging and data clustering, which will be discussed in the next sections.

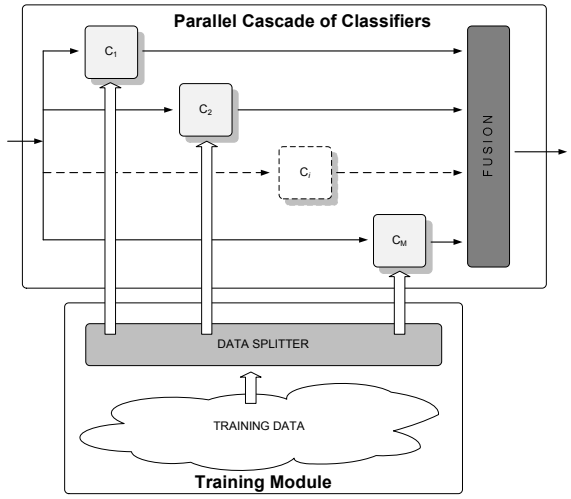

Figure 2. Parallel cascade of classifiers. Input data is split and fed to all classifiers in the ensemble. A final response is provided from the fusion module. To obtain diversity of the ensemble either different classifiers are used and/or different data partitions are used obtained from bagging or clustering. The architecture allows parallel operation of the member classifiers.

There are many examples of the parallel classifier systems organized as shown in Fig. 2. In this paper we exemplify our discussion with parallel implementations of the two of our systems. The first one employs tensor based classifiers (HOSVD) trained with different data partitions obtained with data bagging [5]. The system was tested with the problem of handwritten character recognition showing significantly better results when compared to the single HOSVD classifier. The second of the systems, discussed with more details in this paper, contains one-class SVM classifiers [16][17]. Each of them is trained with a separate partition obtained from data clustering with the k-means and the kernel k-means methods [3][4]. This type of ensemble is discussed in further sections of this paper, whereas details of the tensor based system can be accessed in [5].

III. DESIGN OF THE BUILDING BLOCKS

In this section, we discuss details of blocks of the parallel systems of ensembles of classifiers shown in Fig. 2.

A. Data Splitters

The role of a data splitter is to arrange the training process in order to obtain the best accuracy of the ensemble. The two tested methods are as follows:

- Bagging - consists of creating a number of data sets D_i from the training set D with a uniform data sampling with replacement [18]. As shown by Grandvalet, bagging reduces variance of a classifier and improves its generalization properties [6]. Each set D_i is used to train a separate member of the ensemble, which contains less data than D. Thanks to this data decomposition a better accuracy can be obtained due to a higher diversity. Also, the problem of processing massive data can be greatly reduced. It is also possible to extend the ensemble with a new classifier if new training data are available at a later time.
- Data clustering - consists in usually unsupervised partitioning of the input dataset D into typically disjoint sets D_i [7][10]. In our previous systems the k-means as well as their fuzzy and kernel versions were used. In this case a first step is the choice of data centers. Then data distances to each center are computed and the points are assigned to their nearest centers. After that, positions of the centers are recomputed to account for new members of that partition. The procedure follows until there are no changes in data partitioning [4][18]. Similarly to bagging, splitting by clustering also allows better accuracy and data decomposition useful in parallel realizations.

B. Selection of the Member Classifiers

Choice of the member classifiers depends on many factors, such as type and dimensionality of data. However, the classifiers need to be chosen in a way to assure the best accuracy and speed of operation, especially when processing massive vision data. Good results were obtained in the tested systems using the aforementioned tensor classifiers, as well as using the OC-SVMs which we address in this section. Further references are provided in [4][5].

The binary SVMs were introduced by Cortes and Vapnik [1]. Their one-class version is due to Tax and Duin [16][17], as well as to Schölkopf and Smola [13]. Training of the OC-SVM relies on computation of the parameters of the hyperplane $\mathbf{w}$ that separates data points D_i with the maximal margin from the origin, as shown in Fig. 3. The separating hyperplane is defined as follows [13]

$$\langle \mathbf{w}, \mathbf{x} \rangle - \rho = 0, \tag{1}$$

where $\langle \mathbf{w},\mathbf{x} \rangle$ is a scalar product between vector $\mathbf{w}$ and $\mathbf{x}$. Further, to allow some outliers in the training set D_i with N data points, the slack variables ξ_n are introduced. This leads to the following convex optimization problem

$$\begin{aligned} &\min_{\mathbf{w},\xi_1 \ldots \xi_N,\rho} \left[\frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu N}\sum_{n=1}^{N}\xi_n - \rho \right] \\ &\forall_{1 \le n \le N} \quad \langle \mathbf{w}, \mathbf{x}_n \rangle \ge \rho - \xi_n, \quad \xi_n \ge 0, \end{aligned} \tag{2}$$

where ν is a training parameter that controls allowable number of outliers. The above optimization problem can be solved with the Lagrange multipliers α_n, as follows

$$\begin{aligned} &\min_{\alpha_1 \ldots \alpha_N} \left[\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m \langle \mathbf{x}_n, \mathbf{x}_m \rangle \right], \text{ with} \\ &\forall_{1 \le n \le N} \quad 0 \le \alpha_n \le \frac{1}{\nu N}, \text{ and } \sum_{n=1}^{N}\alpha_n = 1. \end{aligned} \tag{3}$$

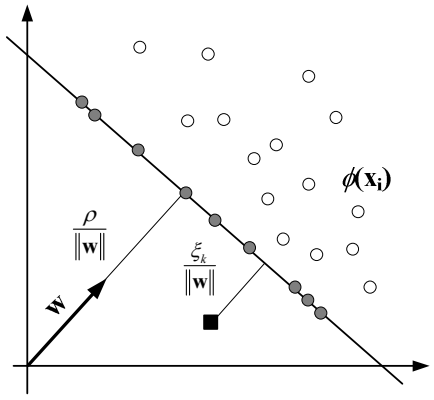

Figure 3. Hyperplane separating a single class of data in the feature space. Support vectors are on the hyperplane. An outlier (black square) is a data on the second side of the hyperplane. This one is controlled with the slack variable.

Data points for which α_n>0 lie on the hyperplane $\mathbf{w}$. These are called support vectors (SV). Solution to (3) results with a set of SV and associated scalar multipliers α_n. The hyperplane $\mathbf{w}$ can be represented as the following weighted sum of the SVs

$$\mathbf{w} = \sum_{n \in SVs} \alpha_n \mathbf{x}_n , \tag{4}$$

Taking any support vector $\mathbf{x}_m$ a distance of the hyperplane $\mathbf{w}$ to the origin can be computed as the following scalar product

$$\rho = \langle \mathbf{w}, \mathbf{x}_m \rangle = \sum_{n \in SVs} \alpha_n \langle \mathbf{x}_n, \mathbf{x}_m \rangle . \tag{5}$$

The real power of OC-SVM is their operation in the so called feature space. This is obtained replacing the inner product in (5) with the kernel function. For the latter the most frequent choice is the Gaussian kernel function, defined as follows [13][14]

$$K_G(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}, \tag{6}$$

where γ is a parameter that control a spread of the kernel. During classification of a test point $\mathbf{x}_x$ its distance to the hyperplane in the feature space is computed. This can be expressed as $K(\mathbf{w},\mathbf{x}_x)$, which if is greater than ρ indicates that a point belongs to the class. Thus, a point $\mathbf{x}_x$ is classified if the following inequality is fulfilled

$$\sum_{n \in SVs} \alpha_n K(\mathbf{x}_n, \mathbf{x}_x) \ge \underbrace{\sum_{n \in SVs} \alpha_n K(\mathbf{x}_n, \mathbf{x}_m)}_{p}. \tag{7}$$

To speed up response of the system the value on the right side of (7) can be precomputed to a scalar p.

Thanks to the relatively simple classification rule (7) its implementation on a graphic card is also not very complicated, as will be discussed in implementation section.

C. Fusion Module

Partial answers of the members of the ensemble need to be collected and used to produce a final response of the system. This is a task of the output fusion module, shown in Fig.2. Again, there are many algorithms for this task from which the following two were tested in the discussed ensembles:

- Majority voting
- Weighted majority voting

Further details are provided in the references [4][5][7][9].

IV. Implementation of the Ensemble of Classifiers

Two implementations were made in order to show properties of the proposed ensemble of classifiers. The first one is the serial software in C++ which uses the HIL library described in [2]. On the other hand, the parallel implementation of the system relies on the graphic cards with the CUDA (Compute Unified Device Architecture) environment [21]. CUDA is architecture of the parallel computing platform and programming model of the majority of the graphic cards by nVidia [21]. Two graphic cards were used in the tests: the FX3800M and the GTX280. Results of comparison of serial and parallel implementations are described in the next section.

Implementation of the CUDA kernel function *SVM_Answer_Kernel*, which carries out the OC-SVM classification, is shown in Fig. 6. This kernel is launched in parallel on the GPU devices. In the lines 8-11 of the listing in Fig. 6 an offset to the data buffer is computed to find out which part of data a kernel is assigned to. Then the *Compute_SV_DistanceTo* kernel function is invoked (shown in the line 16 of Fig. 6), which computes a distance of a test

point (a color pixel) to the hyperplane found in the training process. A training is done off line exclusively on the CPU. Found distance to the hyperplane of each of the test points is saved in the output buffer, as shown in the code lines 17-19 in Fig. 6.

Finally, the CUDA implementation of the Gaussian kernel is presented in Fig. 8. The function *exp_kernel* operates in accordance with (6). It can process data of any length which is provided by the *kDataDim* input parameter. In the case of color pixels *kDataDim*=3.

V. EXPERIMENTAL RESULTS

The ensemble of classifiers was tested in a time demanding task of human face detection from color images. For the training, manually gathered samples of pixels of human skin were used. However, the initial training set was clustered into three disjoint partitions with the k-means method, as described in Section IIIA. Then each of these partitions was used to train a separate base OC-SVM classifier. The parameters ν from (2) and γ from (6) of each of the OC-SVM were found by the grid search method. Operation of the system on the few color frames is depicted in Fig. 4.

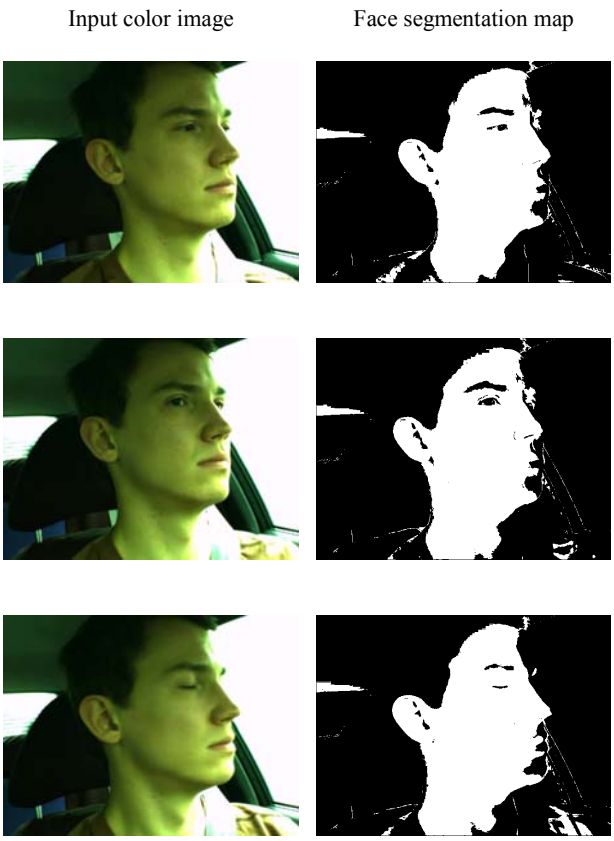

Figure 4. Examples of real-time face segmentation with the ensemble of three OC-SVM classifiers. Color images shown in the left column. Face segmentation maps shown in the right column.

Found parameters of the three base OC-SVM classifiers of the ensemble are shown in TABLE I. The parameter #SVs denotes number of support vectors and ρ is defined in (5).

TABLE I. PARAMETERS AND CONFIGURATION OF THE ENSEMBLES USED IN THE EXPERIMENTS

No.	#SVs	γ	ν	ρ
1	5	0.016279	0.0001262	0.780138
2	5	0.011379	0.0001198	0.864579
3	5	0.013299	0.0001626	0.783055

Accuracy of such ensembles in application of image segmentation reaches up to 0.85-0.90 of the *F* measure [18]. Almost in all examples an application of more than one classifier in an ensemble leads to the higher accuracy, as also discussed in [4]. However, thanks to their parallel architecture the real-time processing of HD images is easy to achieve. A plot of a speed-up ratio of the serial C++ vs. parallel CUDA implementation of the ensemble with the OC-SVM classifiers is shown in Fig. 5.

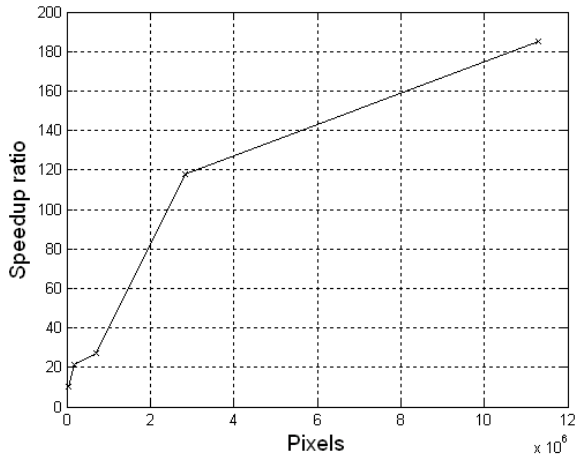

Figure 5. Speed-up ratio of the serial C++ vs. parallel CUDA implementations of the ensemble with OC-SVM classifiers.

In the above plot, we notice that for images which a large number of pixels, such as the ones with resolution 3968x2848, a speed-up ratio exceeds 180. This can be even higher on the FPGA platforms, at a cost of a much larger implementation effort, however. Also, interesting is comparison of serial implementations ported to the multi-core platforms, for example with help of the OpenMP library. In this case, a speed-up ratio of 3-5 times was achieved on the system with the Intel® quad-core processor i7 Q820 with a clock 1.73GHz and 8GB of the system RAM.

VI. CONCLUSIONS AND FUTURE WORK

In the paper different architectures of ensembles of classifiers suitable for parallel processing were discussed. It

was demonstrated that an application of an ensemble of classifiers usually leads to a higher accuracy when compared with a single classifier. In the paper an ensemble of OC-SVM classifiers was used in the problem of color image segmentation in real-time. It was shown that thanks to the highly parallel architecture, the ensemble with OC-SVM classifiers allows two-orders of magnitude speed-up ratio in the CUDA implementation when compared with the serial software version. Details of the parallel implementation with the CUDA code are also provided. It is worth noticing that the method is more general and allows classification of other than visual types of data as well.

Further research will be focused on development of new architectures with different base classifiers which are well suited for parallel implementations on different computer platforms.

ACKNOWLEDGMENT

Financial support in the years 2012-2013 of the National Center for Research and Development of the Republic of Poland, under the project SYNAT is greatly appreciated.

REFERENCES

[1] C. Cortes and V. Vapnik, Support vector machines. Machine Learning, Vol. 20, 1995, pp. 273-297.

[2] B. Cyganek and J.P. Siebert, An Introduction to 3D Computer Vision Techniques and Algorithms, Wiley, 2009.

[3] B. Cyganek and K. Wiatr, Image Contents Annotations with the Ensemble of One-Class Support Vector Machines. International Conference on Neural Computation Theory and Applications, 24-26 October, Paris, France, 2011.

[4] B. Cyganek, One-Class Support Vector Ensembles for Image Segmentation and Classification. Journal of Mathematical Imaging & Vision, Vol. 42, No. 2-3, Springer, 2012, pp. 103–117.

[5] B. Cyganek, Ensemble of Tensor Classifiers Based on the Higher-Order Singular Value Decomposition. HAIS 2012, Salamanca, Springer, Part II, LNCS 7209, 2012, pp. 578–589.

[6] Y. Grandvalet, Bagging equalizes influence. Machine Learning, Vol. 55, 2004, pp. 251-270.

[7] L.I. Kuncheva, Combining Pattern Classifiers. Methods and Algorithms. Wiley Interscience, 2005.

[8] M. Kurzyński and M. Woźniak, Combining classifiers under probabilistic models: experimental comparative analysis of methods. Expert Systems, Vol. 29, No. 4, 2012, pp. 374–393.

[9] R. Polikar, Ensemble Based Systems in Decision Making. IEEE Circuits and Systems Magazine, 2006, pp. 21-45.

[10] A. Rahman and B. Verma, Cluster-based ensemble of classifiers. Expert Systems, 2012.

[11] E. Rafajłowicz, Classifiers sensitive to external context theory and applications to video sequences. Expert Systems, Vol. 29, No. 1, 2012, pp. 84–104.

[12] J. Sanders and E. Kandrot, CUDA by Example. An Introduction To General-Purpose GPU Programming. Addison-Wesley, 2011.

[13] B. Schölkopf and A.J. Smola, Learning with Kernels, MIT Press, 2002.

[14] J. Shawe-Taylor and N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.

[15] R. Tadeusiewicz and M.R. Ogiela, New Diagnostics Perspectives Connected with a Concept of Automatic Patterns Understanding. Chapter, in book by Korbicz J., Patan K., Kowal M., Fault Diagnostics and Fault Tolerant Control, EXIT, Warsaw, ISBN 83-60434-32-1, 2007, pp. 43-49.

[16] D.M.J. Tax, One-class classificatio. PhD thesis, TU Delft University, 2001.

[17] D. Tax, R. Duin, Support Vector Data Description, Machine Learning, Vol. 54, 2004, pp.45-66.

[18] S. Theodoridis and K. Koutroumbas, Pattern Recognition, 4th ed., Academic Press, 2009.

[19] P. Viola and M. Jones, Robust real-time face detection, Proceedings of the International Conference on Computer Vision, 2001, pp. 747-755.

[20] K. Wiatr, CYFRONET supercomputers in support of modern research projects, KU KDM 2011, fourth ACC Cyfronet AGH users' conference, Zakopane, March 09–11, 2011.

[21] www.nvidia.com

```
1     __global__ void SVM_Answer_Kernel( const float * inDataBuf, unsigned char * outDataBuf,
2                                        float * outValuesBuf, const int kTotalNumOfData,
3                                        float * SV_vectors, float * alpha_vector,
4                                        const int kNumOfSVs, const float kGamma,
5                                        const float kRho, const int kDataDim  )
6     {
7          // Here we map threadIdx and BlockIdx to pixel position in the buffer
8          int x = threadIdx.x + blockIdx.x * blockDim.x;
9          int y = threadIdx.y + blockIdx.y * blockDim.y;
10
11         int offset = x + y * blockDim.x * gridDim.x;
12
13         if( offset < kTotalNumOfData )        {
14              const float * this_data_offset = inDataBuf + offset * kDataDim;
15              // since inDataBuf and SV_vectors actually are 2D structures
16              float distance = Compute_SV_DistanceTo(  this_data_offset, SV_vectors,
                                                         alpha_vector, kNumOfSVs, kGamma, kDataDim );
17              float val = distance - kRho;
18              * ( outValuesBuf + offset ) = val;
19              * ( outDataBuf + offset ) = val > 0.0 ? 0 : 255;
20         }
21    }
```

Figure 6. Implementation of the CUDA kernel for the OC-SVM classifier. The *Compute_SV_DistanceTo* function is called (shown in Fig. 7) which computes a distance of a test point (a pixel) to the hypersphere.

```
1     // SV_vectors and alpha_vector should be in the constant memory
2     __device__ float Compute_SV_DistanceTo(    const float * data_point, const float * SV_vectors,
3                                                const float * alpha_vector, const int kNumOfSVs,
4                                                const float kGamma, const int kDataDim )
5     {
6          // We assume RBF kernels ONLY!!
7          float alpha = 0.0;
8          float kernel_product = 0.0;
9
10         // Compute the second term
11         register double theSum = 0.0;
12         #pragma unroll
13         for( register int i = 0; i < kNumOfSVs; ++ i )
14         {
15    #if USE_CONST_MEMORY
16
17              kernel_product  = exp_kernel( in_SVs_vector_GPU + i * kDataDim, data_point, kGamma, kDataDim );
18              alpha           = in_alphas_vector_GPU[ i ];
19
20    #else //USE_CONST_MEMORY
21
22              kernel_product  = exp_kernel( SV_vectors + i * kDataDim, data_point, kGamma, kDataDim );
23              alpha           = alpha_vector[ i ];
24
25    #endif //USE_CONST_MEMORY
26
27              theSum     += alpha * kernel_product;
28         }
29
30         return theSum;
31    }
```

Figure 7. CUDA kernel function executed for each pixel. The function invokes the *exp_kernel* function shown in Fig. 8.

```
1     __device__ float exp_kernel( const float * x, const float * y, const float kGamma, const int kDataDim )
2     {
3          register float tmp = 0.0;
4          register float sum = 0.0;

5          #pragma unroll
6          for( register int d = 0; d < kDataDim; ++ d )
7          {
8               tmp = x[ d ] - y[ d ];
9               sum += tmp * tmp;
10         }

11         return expf( - kGamma * sum );
12    }
```

Figure 8. CUDA implementation of the exponential (Gaussian) kernel.

 ISBN: 978-1-61208-258-5