Parallel Image Burst Analysis in CUDA

Kandiraju Sai Ashritha
Computer Engineering
National Institute of Technology
Karnataka, Surathkal
14CO121
Email: ashritha1615@gmail.com

Prajwala TM
Computer Engineering
National Institute of Technology
Karnataka, Surathkal
14CO133
Email: prajwala.tm@gmail.com

I. Problem Statement

An application that rapidly performs parallel image analysis on a image burst - several photographs captured in quick succession.

II. Objectives

1) Understanding the Viola Jones face detection algorithm
2) Analysis of any additional optimizations to the serial algorithm
3) Implementation of the serial algorithm in C++
4) Understanding the bottlenecks and challenges associated with the serial implementation
5) Analysis of parallelizable components of the Viola Jones algorithm
6) Implementation of the parallel version in CUDA
7) Developing an algorithm for determination of eye state(open/close)
8) Implementation of the serial eye state detection algorithm in C++
9) Understanding the bottlenecks and challenges associated with the serial implementation of eye state algorithm if any
10) Analysis of parallelizable components of eye state detection algorithm
11) Implementation of the parallel version of application in CUDA

III. Project Timeline

1) **March 1st-5th** Understanding the Viola Jones algorithm and setting up the working environment
2) **March 6th-11th** Implementation of the serial version of the algorithm in C++
3) **March 12th** Submission of Mid-progress report
4) **March 13th-18th** Analysis and implementation of basic parallel version of Viola Jones algorithm in CUDA
5) **March 19th-25th** Modifications and optimisations to the basic CUDA version
6) **March 26th-April 1st** Developing a naive serial algorithm for eye state detection and understanding bottlenecks if any
7) **April 2nd-5th** Implementation of parallel version of eye state detection algorithm in CUDA
8) **April 6th-11th** Work on further optimisations and test the application

IV. Work Distribution

1) Collectively understand the serial version of the Viola-Jones algorithm
2) Collectively analyse optimisations to the algorithm if any
3) Divide the implementation of the serial Viola Jones algorithm
4) Collectively analyse the bottlenecks associated with serial implementation
5) Collectively develop algorithm for eye state detection
6) Divide the implementation of the eye state algorithm individually
7) Testing the different components of the application separately

V. Viola Jones Algorithm Overview

Below is a diagram showing a brief overview of the algorithm implemented for face detection and recognition in the input image burst:

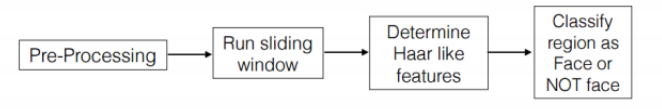

Fig. 1: Brief overview of Viola Jones Algorithm

A. Pre-processing

The image pre-processing consists of converting the image to Gray-scale, down-sampling the image for Face Detection.

B. Run sliding window

A sliding window of size 25×25 is used to traverse the image pixel by pixel and runs certain filters on the image area covered by it. These filters, known as Haar filters capture certain features used for facial detection.

C. Haar filters

The Haar filters used by the sliding window reveal horizontal and vertical features in the image. A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image. The features in Fig.11 capture features near the eyes, nose bridge apart from others.

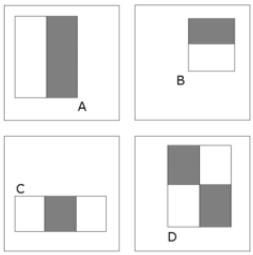

Fig. 2: Few Haar filters

D. Cascade classifier

The Haar-like features ted, compuare compiled into a single large classifier, which is applied to each sub-window. At each step, the algorithm classifies a sub-window using a specific feature. If the sub-window meets the criteria then the algorithm proceeds with applying other features, otherwise, the sub-window is rejected. Intuitively, at each stage, the algorithm compares the values against tested classifiers and asks Does this feature resemble a facial feature? Earlier stages are easier whereas later stages extremely computationally intensive and more difficult to meet the criteria.

VI. Sequential Approach

1) The Viola-Jones face detection algorithm has been implemented using C++. A burst of images is input to the application and the faces in each of the images in the burst are detected and marked with rectangular bounding boxes. The number of faces detected in each of the algorithms are also returned.
2) The part of the image marked by the rectangular bounding boxes can be alone used as input to the eye-state detection algorithm that will be implemented in the future for the detection of the best image in the burst.
3) A standard Haar filter consisting of a large number of cascade classifications has been used. This could be optimized further by using Adaboost algorithm to use only the relevant set of features and reduce the hierarchy.
4) An analysis of our CPU code showed that a large percentage of overall time was spent on computing sum of pixel intensities for each sub window.

VII. Results

The results of the sequential implementation of the algorithm are depicted below.

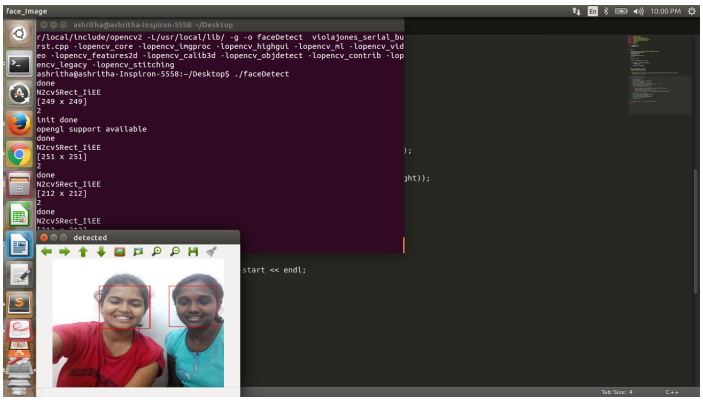

Fig. 3: Result-1

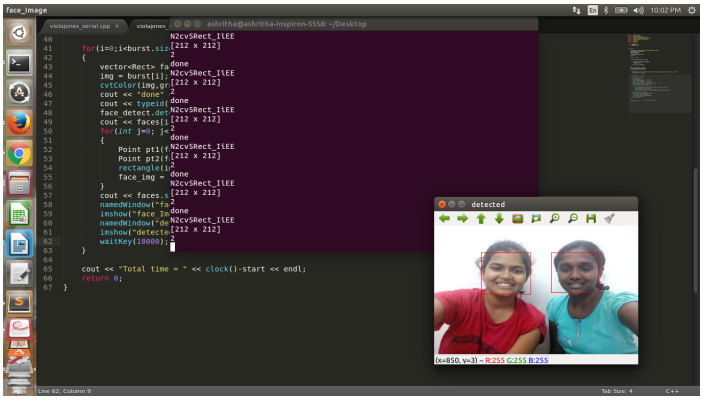

Fig. 4: Result-2

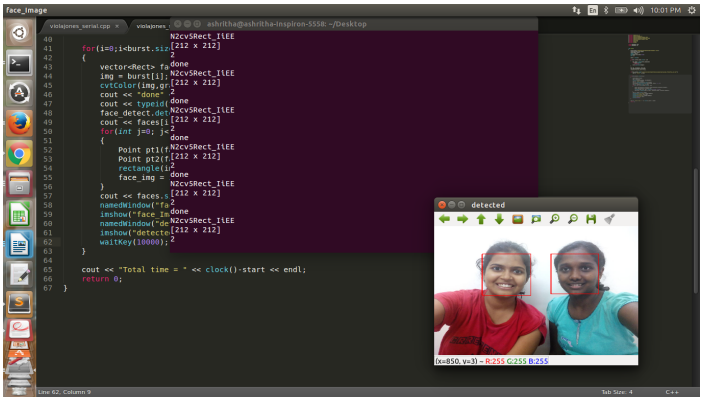

Fig. 5: Result-3

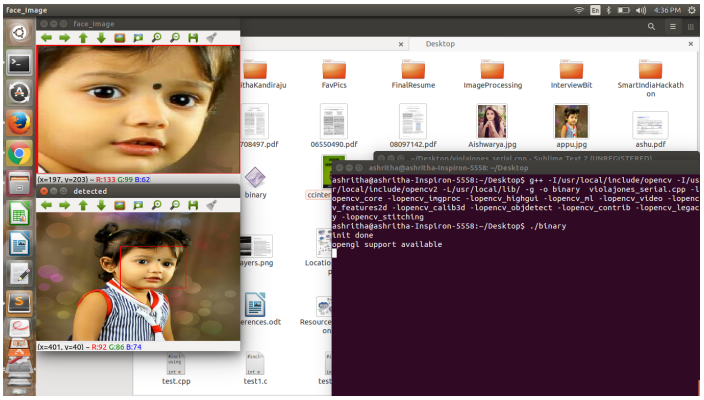

Fig. 6: Result-4

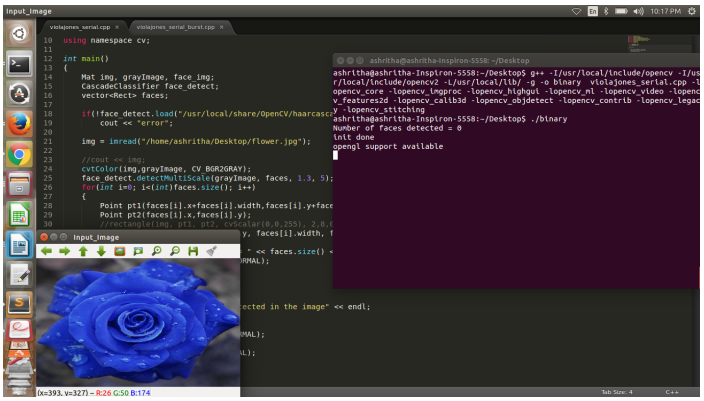

Fig. 7: Result-5

Fig.3 shows an image where faces with fully-closed eyes have been detected. Fig.4 shows faces with partially-closed eyes detected, Fig.5 depicts the image in which eyes are open and the people are smiling. Fig.6 shows an image input to the algorithm and the output depicting the cropped part of the face which will be input to the eye-state detection algorithm to be implemented. The final Fig.7 shows a non-face image input, and the algorithm displays that there is no face detected in the input. The next part of the work discards such sub-windows and considers only the windows consisting of facial features.

VIII. Parallel version of Viola Jones

The Viola Jones algorithm[1] essentially has three important concepts tied to it - Integral image calculation, Adaboost classifier training algorithm[2] and cascade classifier. In the implementation, a trained classifier network which includes different HAAR classifier features obtained as a result of training on thousands of images has been used. Therefore, the main focus of implementation was on identifying the different portions of the algorithm that can be parallelized and leverage the execution with abundant GPU resources efficiently. Three main phases of face detection were attempted to be parallelized - Nearest neighbour calculation, integral image computation and scanning window stage along with classifying the output from each classifying stage. Several key concepts, principles and insights of the implementation have been inspired from previous implementations of face detection on GPGPUs, FPGA done here [3,4,5].

IX. Work flow of Viola Jones algorithm

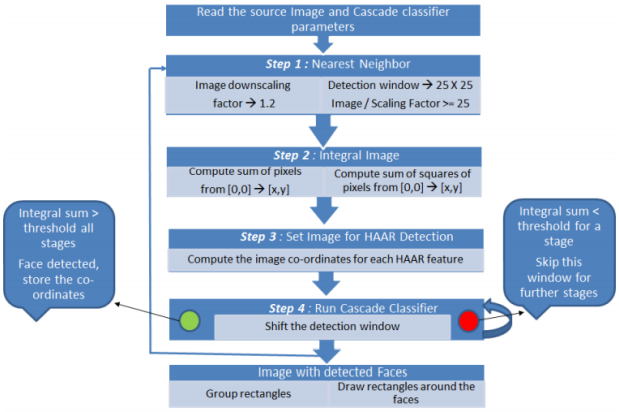

Fig. 8: Face detection algorithm implementation flow

Fig. 8 shows the overall flow of the algorithm. First the input image and cascade classifier parameters are read. Then the downscaling of the image is performed and integral image of the downscaled image is computed. Next the downscaled image co-ordinates required for the HAAR features is determined. These are relative positions of each HAAR rectangle boundaries in a 25×25 window. This scanning window is processed through each stage of the cascade classifier. For a given stage if the integral sum is less than the threshold for that stage, then the remaining stages are skipped for this window (this implies that no face has been detected). If the integral sum exceeds the threshold limit of a stage, in all the stages, then it is qualified as a face. In that case, the window position and image scale are stored. Once window is processed as above, it shifted by 1 pixel and then passed through step 4 for face detection. This process is continued until the total image is scanned at that particular scale. Then the whole process is repeated by downscaling the image in step 1 through face detection in step 4. The input image runs through this series of steps from 1 through 4 until its downscaled to the size of detection window. Finally, all the window positions and the scale at which a face has been detected are obtained. The faces are indicated in the final image with a rectangle bounding box.

A. *Nearest Neighbour*

The detection window has been fixed to a size 25×25. But the face in the image can be of any scale. To make the face detection scale invariant, we have used pyramid of images as shown in Fig. 9 that contain downscaled versions of the original image.

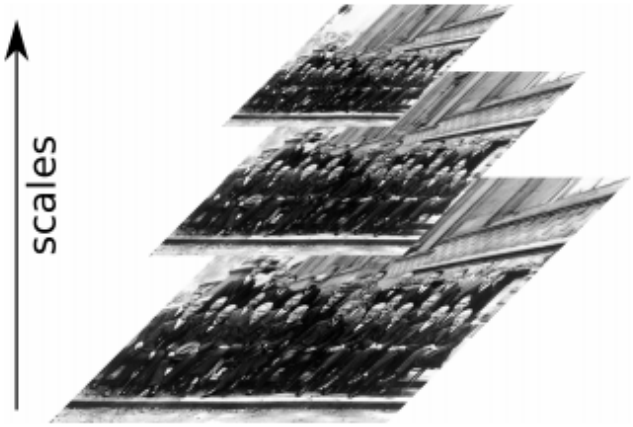

Fig. 9: Image pyramid for scale-invariant face detection

Thus, downscaling the image to make the face detection independent of the detection window size is an important step that is achieved by the nearest neighbour function. Every time the images width and height are downscaled by a factor of 1.2 until one of them reaches the detection windows width or height of 25 pixels. A downscaling example for a scale factor of 2 can be seen in Fig. 10.

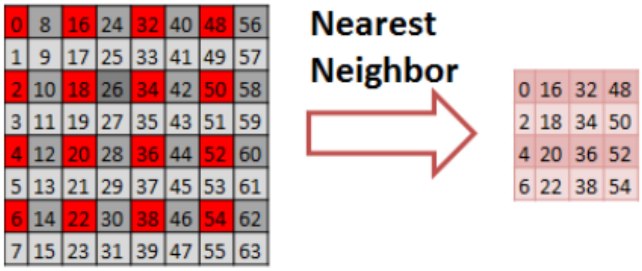

Fig. 10: Image downscaling using Nearest neighbour

Parallelization Scope : Each pixel in the downscaled image can be calculated with the help of scale factor, offset, width and height of the source image. As this operation is independent of the other pixels we map pixel positions to threads and the computation is thus parallelized.

B. *Integral Image*

The Viola Jones classifier method is based on Haar-like features. The features consist of white and black rectangles as shown in Fig. 11. These features can be thought of as pixel intensity evaluation sets. For each feature, we subtract black regions pixel value sum from white regions pixel value sum. If this sum is greater than a certain threshold, it is decided that the region has this feature. This is the characteristic value of a feature. Haar features are used for face detection in the viola jones algorithm.

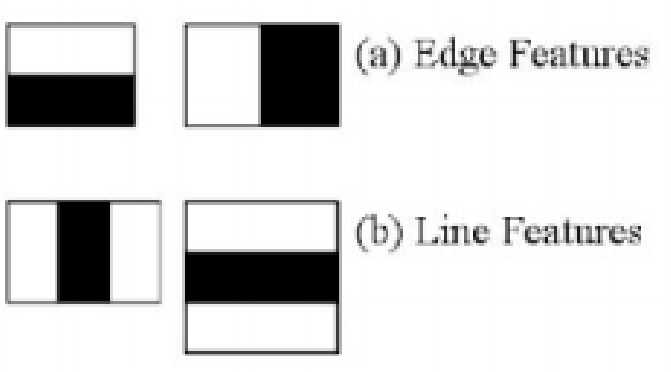

Fig. 11: Haar feature rectangles

The calculation of characteristic value of a feature is an important step in face detection. It has to be calculated for every possible pixel region in the given image. In order to efficiently determine the value, integral image of the given image is used. For any given pixel (x, y), the integral value is the sum of all the pixels above and to the left of (x, y), inclusive. Therefore sum of any given region with corner integral pixels A, B, C, D is Sum = D + A - B - C as shown in Fig. 12.

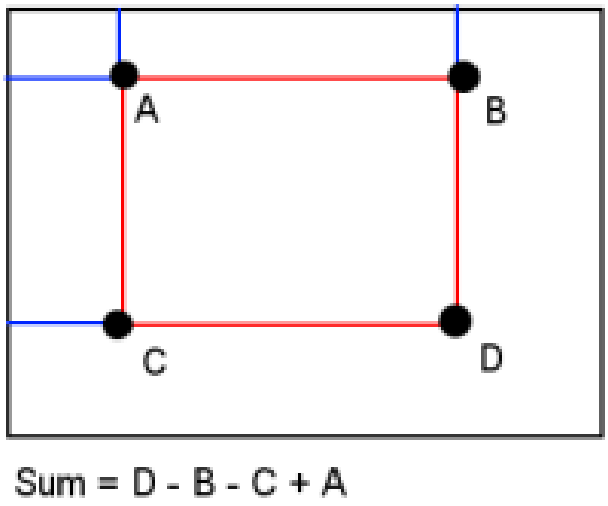

Fig. 12: Sum of a sub-window using Integral image

The importance of Integral image is that it reduces the number of computations. As shown in Fig. 13 to obtain the sum of 2×3 sub-window, integral image involves just 2 additions and 2 subtractions instead of 6 additions.

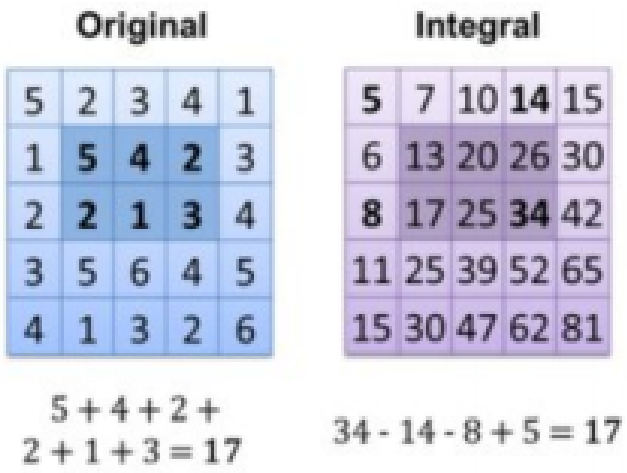

Fig. 13: Integral image calculation reduces computations

For each X and Y in the downscaled image, Integral Image computes the sum of all the pixels above and to the left of (X, Y). Integral image calculation can be split into two separate operations as RowScan(RS) and columnScan(CS) of the image, where, RowScan(RS) is inclusive prefix scan along the row for all the rows and ColumnScan(CS) is inclusive prefix scan along the column for all the columns. RowScan and ColumnScan operations are shown in Fig. 14.

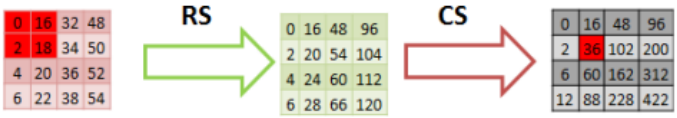

Fig. 14: Image integral sum calculation using RS and CS operations

Parallelization Scope : In RowScan, prefix scan of each row is independent of the other rows in the image and same is the case with columns in the ColumnScan operation. Block level parallelism has been leveraged here. Prefix scan for a row/column though by itself seems to be a sequential operation, it has been parallelized using Hillis-Steele Scan algorithm.

4 Kernels have been used to accomplish the parallel implementation of the algorithm. Kernel 1 downscales the image and computes parallel row scan. Kernel 2 implements matrix transpose. Kernel 3 calculates row scan. Kernel 4 calculates matrix transpose. The nearest neighbor and rowscan functionality is accomplished in a single kernel to avoid storing the downscaled image to global memory between kernel launches. Therefore, each downsampled row of the source image is stored in the shared memory and the syncthreads() function is called before proceeding to RowScan on that respective row. The kernel 1 combined operation can be seen in Fig. 15.

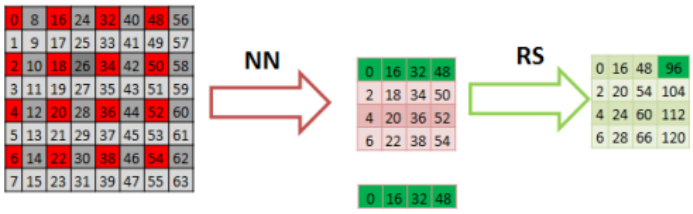

Fig. 15: Nearest neighbour and row scan in parallel

The image matrix obtained after kernel 1 is transposed, a rowscan is performed on it and finally its transposed back to obtain the integral image sum. The combined operations of kernels 2,3 and 4 can be seen in Fig. 16.

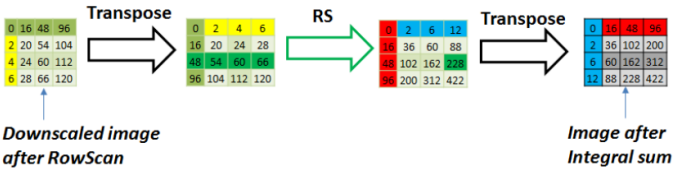

Fig. 16: Column scan kernel breakdown

C. Scan Window Processing

Each Haar classifier has up to 3 rectangles and has a fixed size of 25×25. Several weak Haar Classifiers are combined in weighted sum to form Strong Classifiers. The algorithm consists of 25 stages of strong classifiers with a total of 2913 Haar classifiers. For a given image, we consider a scan window of size 25×25 starting from the top left corner. The scan window is identified by (x,y) coordinates of its starting pixel. Each scan window is then passed through the stages of strong classifiers to detect if they contain the haar features. Each Haar classifier of Stage 'x' contains up to three rectangles. The rectangle is defined by (x,y) coordinates of its starting pixel, its width and height. This data is useful for calculating the four corners of the region in scan window corresponding to this rectangle. Then, the integral image is used to calculate the sum of the pixels of the rectangles region. Each Haar rectangle is also associated with a weight that is used to calculate the weighted sum of the scan window for Haar classifier. The accumulated sum is determined for all the Haar classifiers in the stage 'x'. If the sum is greater than a threshold[x], the scan window is considered to pass the stage 'x'. The scan window that passes through all the stages is considered to be a face. The next scan window is chosen by incrementing the pixel number. The process is repeated for all the scan windows of the image.

X. Results

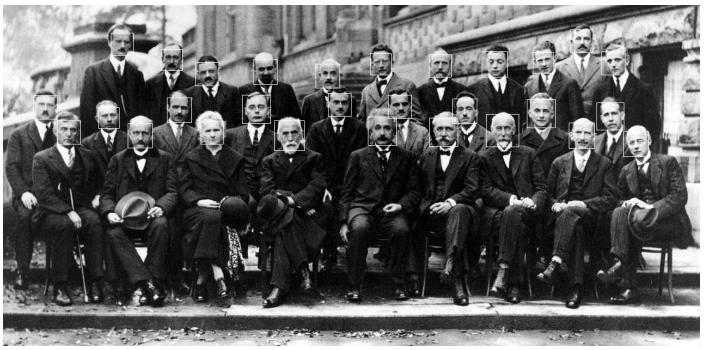

Fig. 17: Output

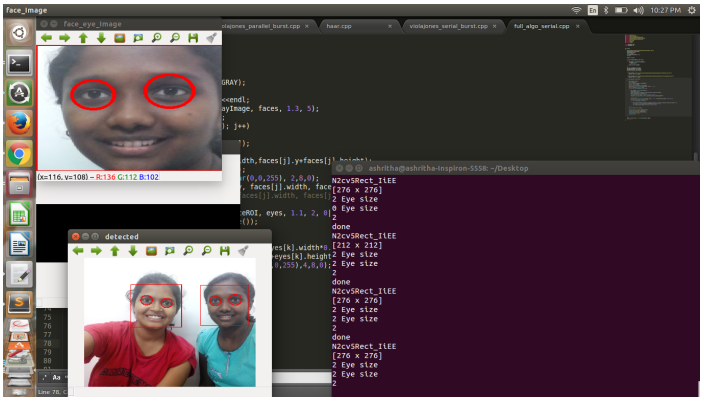

Fig. 18: Eyes and Faces detected

Fig. 17 shows the output obtained by the parallel viola jones algorithm. Fig. 18 shows the eyes and faces detected using the complete algorithm that includes eye state detection also.

XI. List of Objectives Achieved

- Understanding Viola Jones algorithm used for frontal face detection.
- Analysis of the serial algorithm.
- Implementation of the serial algorithm in C++.
- The bottlenecks in the serial implementation of the viola jones algorithm have been identified.
- Parallelizable components of the algorithm have been analyzed.
- Implementation of the parallel version using CUDA C++.
- Implementation of the eye detection algorithm using opencv eye cascade classifier.
- Implementation of the face and eye detection algorithm serial version.
- Implementation of the face and eye detection algorithm in parallel using OpenMp.

XII. References

1) P. Viola and M. Jones, Rapid object detection using a boosted cascade of simple features, in Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1. IEEE, 2001, pp. I511.
2) Y. Freund, R. Schapire, and N. Abe, A short introduction to boosting, Journal-Japanese Society For Artificial Intelligence, vol. 14, no. 771-780, p. 1612, 1999.
3) J. Kong and Y. Deng, Gpu accelerated face detection, in Intelligent Control and Information Processing (ICICIP), 2010 International Conference on. IEEE, 2010, pp. 584588.
4) J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, Fpga-based face detection system using haar classifiers, in Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays. ACM, 2009, pp. 103112.
5) L.-c. Sun, S.-b. Zhang, X.-t. Cheng, and M. Zhang, Acceleration algorithm for cuda-based face detection, in Signal Processing, Communication and Computing (ICSPCC), 2013 IEEE International Conference on. IEEE, 2013, pp. 15.