

Parallel Computing Project

Face Detection On Photos using OpenCV

- Submitted by :

Kunal
B.Tech
PDPM-IIITDM Jabalpur
2019085@iiitdmj.ac.in
tulsidasanikunal@gmail.com

INTRODUCTION

Over the last decade automatic face detection has attracted the attention of researchers as a necessary preparation step before face recognition, facial feature extraction or other advanced human/computer interaction systems. The value of a face detection system is determined first by the detection accuracy and the false alarm rate it scores, and second, by the computational cost it incurs to process an image i.e., the cost to detect and localize all the faces that may exist in the image. Speed and efficiency became quite important from the moment of the first commercial applications that need face detection.

Following the need for a very fast face detection system, an approach based on a serial cascade of Haar-based classifiers of increasing complexity was proposed. In this the simple ones are charged to reject quickly image areas quite distant from a face, while the more complex classifiers are used to make the final decision in the remaining and more difficult to handle image areas. While the key feature is good detection accuracy rates.

The parallel programming model that we will use to parallelize the face detection system is OpenMP, which is the standard parallel programming model for the shared memory architecture. Most vendors participate in the evolution of OpenMP and provide custom implementations. OpenMP allows for incremental and scalable parallelization of existing code, providing thus a fast way of parallelizing the face detection system with minimal changes to its data structures and algorithm. The parallelization of the face detection system demonstrates significant and cost-effective improvements in the processing time of a single image.

SERIAL APPROACH

The proposed approach will use Input images from user to detect human faces in the image and draw a rectangle around it.

The architecture has different phases like Image Input, Face detection data loading , face detection, Rectangle drawing.

A. Image Input

All the images in which faces are to be detected is to be kept in Input folder. The code when executed will take all the images in Input folder as inputs and detect faces.

B. Load Face detection data

OpenCV provides a Face detection dataset which needs to be loaded as a cascade classifier class which can be passed to the function to detect faces.

C. Face Detection

From the above data which is passed to face detection algorithm, predefined in openCV all the faces are detected and the point location of the faces are stored in a vector of type Rect.

D. Rectangle Drawing

From the above face detection algo, the stored points are taken as input and rectangles are drawn around every face detected.

PARALLEL APPROACH

The proposed approach will distribute Input images from user to every thread in order to detect human faces in the image and draw a rectangle around it.

The architecture has different phases like Image Input, Thread Initialization & Work Distribution, Face detection data loading , face detection, Rectangle drawing.

A. Image Input

All the images in which faces are to be detected is to be kept in Input folder. The code when executed will take all the images in Input folder as inputs and detect faces.

B. Threads Initialization & Work Distribution

OpenMP is used to Initialize threads and work is distributed manually to each thread.

All the below steps are performed by each thread separately.

```
//Set number of threads as 8
omp_set_num_threads(8);
```

```
#pragma omp parallel default(shared) private(tid, numt, i)
{
    //Declare a object of CascadeClassifier class which will load the data for face detection, which is already present in openCV
    CascadeClassifier FD;

    //Load the data file
    //if not loaded the function will exit
    if(!FD.load("haarcascade_frontalface_default.xml")){
        cout<<"No load file\n";
        exit(0);
    }

    int from, to;

    tid = omp_get_thread_num();
    numt = omp_get_num_threads();

    from = (N/numt)*tid;
    to = (N/numt)*(tid+1) - 1;

    if(tid == numt -1){
        to = N-1;
    }
}
```

C. Load Face detection data

OpenCV provides a Face detection dataset which needs to be loaded as a cascade classifier class which can be passed to the function to detect faces.

D. Face Detection

From the above data which is passed to face detection algorithm, predefined in openCV all the faces are detected and the point location of the faces are stored in a vector of type Rect.

E. Rectangle Drawing

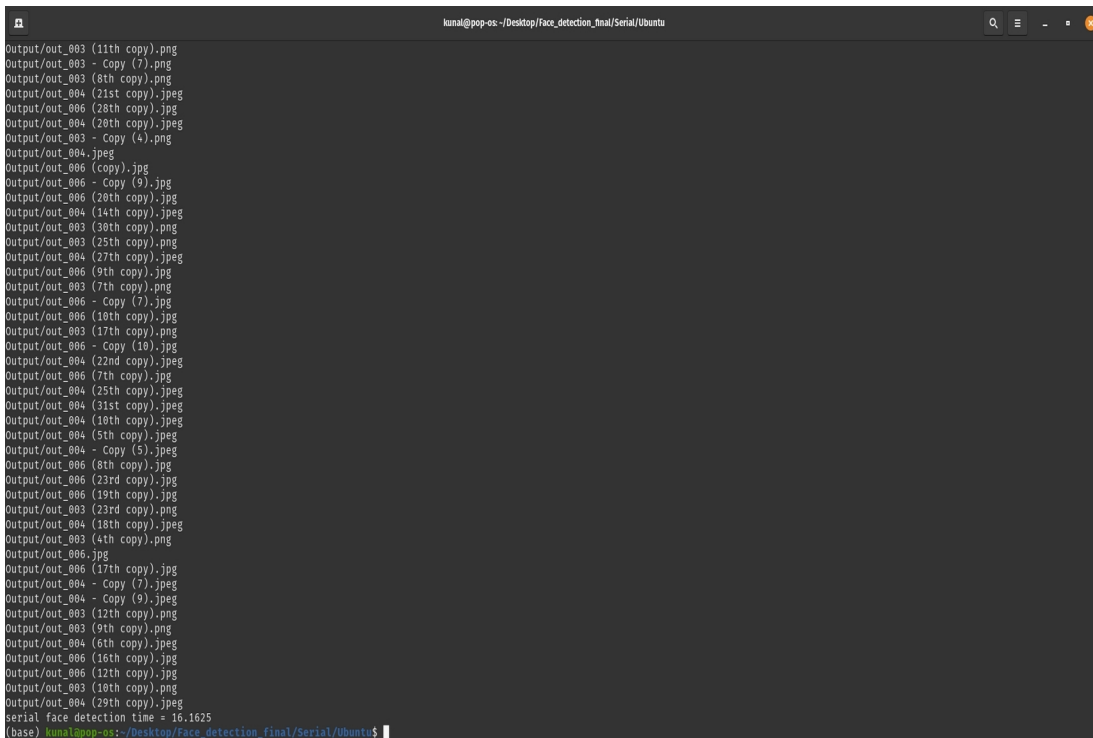
From the above face detection algo, the stored points are taken as input and rectangles are drawn around every face detected.

SERIAL IMPLEMENTATION RESULTS

Almost all the faces were detected in images, with an accuracy of above 95%.

Serial Results in Ubuntu-

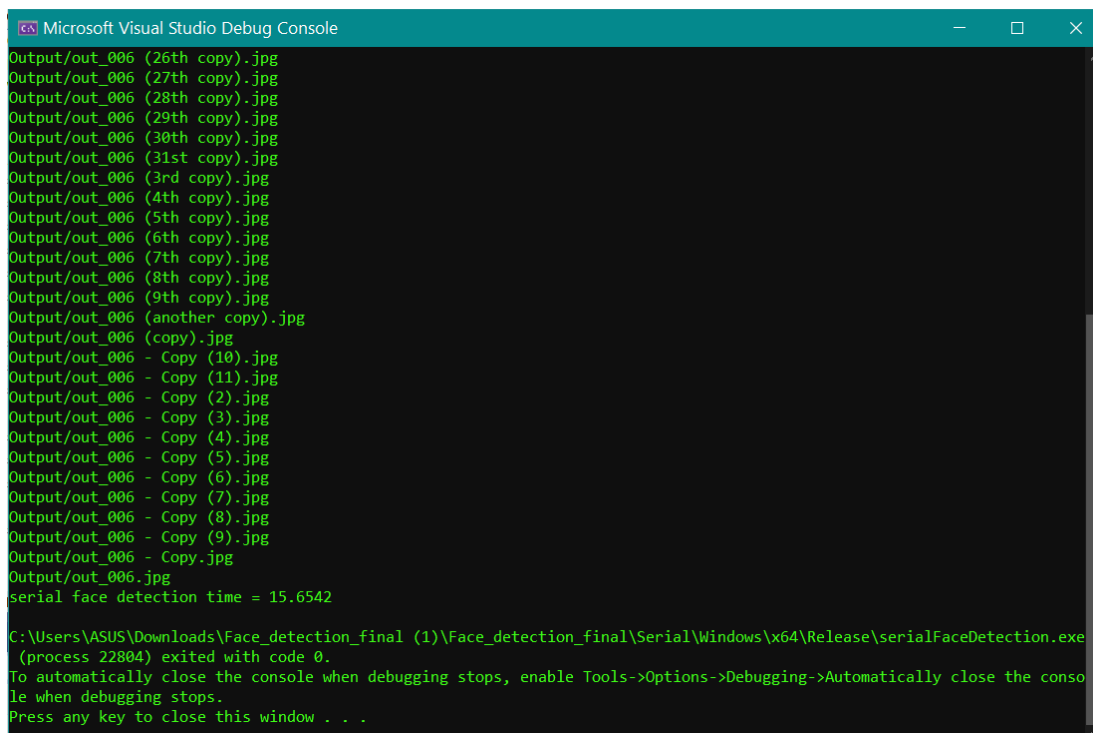
128 Pictures were processed in 16.1625 seconds.



```
kunal@pop-os: ~/Desktop/Face_detection_final/Serial/Ubuntu
Output/out_003 (11th copy).png
Output/out_003 - Copy (7).png
Output/out_003 (8th copy).png
Output/out_004 (21st copy).jpeg
Output/out_006 (28th copy).jpg
Output/out_004 (20th copy).jpeg
Output/out_003 - Copy (4).png
Output/out_004.jpeg
Output/out_006 (copy).jpg
Output/out_006 - Copy (9).jpg
Output/out_006 (20th copy).jpg
Output/out_004 (14th copy).jpeg
Output/out_003 (30th copy).png
Output/out_003 (25th copy).png
Output/out_004 (27th copy).jpeg
Output/out_006 (9th copy).jpg
Output/out_003 (7th copy).png
Output/out_006 - Copy (7).jpg
Output/out_006 (10th copy).jpg
Output/out_003 (17th copy).png
Output/out_006 - Copy (10).jpg
Output/out_004 (22nd copy).jpeg
Output/out_006 (7th copy).jpg
Output/out_004 (25th copy).jpeg
Output/out_004 (31st copy).jpeg
Output/out_004 (10th copy).jpeg
Output/out_004 (5th copy).jpeg
Output/out_004 - Copy (5).jpeg
Output/out_006 (8th copy).jpg
Output/out_006 (23rd copy).jpg
Output/out_006 (19th copy).jpg
Output/out_003 (23rd copy).png
Output/out_004 (18th copy).jpeg
Output/out_003 (4th copy).png
Output/out_006.jpg
Output/out_006 (17th copy).jpg
Output/out_004 - Copy (7).jpeg
Output/out_004 - Copy (9).jpeg
Output/out_003 (12th copy).png
Output/out_003 (9th copy).png
Output/out_004 (8th copy).jpeg
Output/out_006 (16th copy).jpg
Output/out_006 (12th copy).jpg
Output/out_003 (10th copy).png
Output/out_004 (29th copy).jpeg
serial face detection time = 16.1625
(base) kunal@pop-os:~/Desktop/Face_detection_final/Serial/Ubuntu$
```

Serial results in Windows-

128 Pictures were processed in 15.6542 seconds.



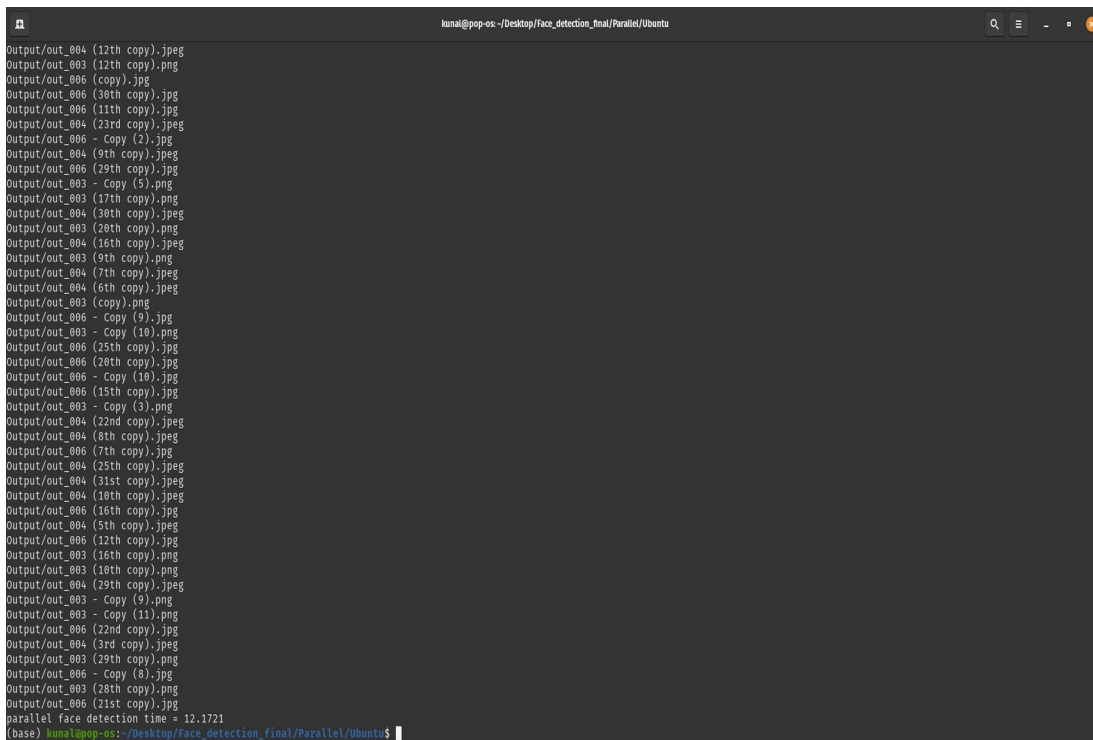
```
Microsoft Visual Studio Debug Console
Output/out_006 (26th copy).jpg
Output/out_006 (27th copy).jpg
Output/out_006 (28th copy).jpg
Output/out_006 (29th copy).jpg
Output/out_006 (30th copy).jpg
Output/out_006 (31st copy).jpg
Output/out_006 (3rd copy).jpg
Output/out_006 (4th copy).jpg
Output/out_006 (5th copy).jpg
Output/out_006 (6th copy).jpg
Output/out_006 (7th copy).jpg
Output/out_006 (8th copy).jpg
Output/out_006 (9th copy).jpg
Output/out_006 (another copy).jpg
Output/out_006 (copy).jpg
Output/out_006 - Copy (10).jpg
Output/out_006 - Copy (11).jpg
Output/out_006 - Copy (2).jpg
Output/out_006 - Copy (3).jpg
Output/out_006 - Copy (4).jpg
Output/out_006 - Copy (5).jpg
Output/out_006 - Copy (6).jpg
Output/out_006 - Copy (7).jpg
Output/out_006 - Copy (8).jpg
Output/out_006 - Copy (9).jpg
Output/out_006 - Copy.jpg
Output/out_006.jpg
serial face detection time = 15.6542
C:\Users\ASUS\Downloads\Face_detection_final (1)\Face_detection_final\Serial\Windows\x64\Release\serialFaceDetection.exe
(process 22804) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

PARALLEL IMPLEMENTATION RESULTS

Almost all the faces were detected in images, with an accuracy of above 95%.

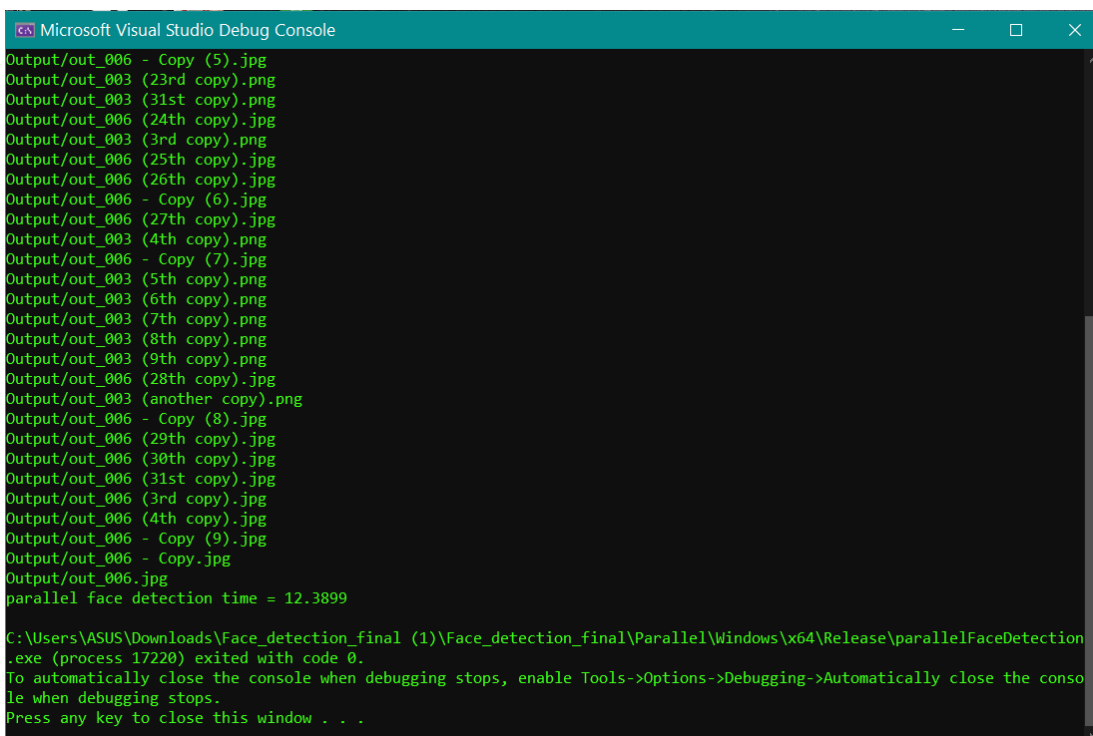
Parallel results in Ubuntu-

128 Pictures were processed in 12.1721 seconds.

A terminal window titled 'kunal@pop-os: ~/Desktop/Face_detection_final/Parallel/Ubuntu' displays a list of 128 output files, including various copies of images from 'out_003' and 'out_006'. At the bottom, it shows 'parallel face detection time = 12.1721' and the shell prompt '(base) kunal@pop-os:~/Desktop/Face_detection_final/Parallel/Ubuntu\$'.

Parallel results in Windows-

128 Pictures were processed in 12.3899 seconds.

A Visual Studio Debug Console window shows a list of 128 output files, including various copies of images from 'out_003' and 'out_006'. It reports 'parallel face detection time = 12.3899' and shows the execution path 'C:\Users\ASUS\Downloads\Face_detection_final (1)\Face_detection_final\Parallel\Windows\x64\Release\parallelFaceDetection.exe (process 17220) exited with code 0.' followed by instructions to close the console.

RESULT COMPARISON

Both in parallel as well as serial code, faces were detected with the same accuracy.

	Windows (in Sec)	Ubuntu (in Sec)
Serial Execution Time	15.65	16.16
Parallel Execution Time	12.39	12.17
Difference in Time	3.26	3.99
Percentage Improved	20.83 %	24.69 %

In Windows the face detection was 20.83% faster and in Ubuntu it was 24.69 % faster.