Parallel Face Detection

By
Aarush Bhat -19BCE0564
Jyotir Aditya- 19BCE0846

Abstract

Face detection using OpenCV with haar-like cascade classifier is a very common and faster method to implement. But for massive faces in one frame and long videos, it is very time consuming and slow processing. In this project, we are representing a method for parallel processing for multiple face detection and recognition in images with many faces, long videos, videos with many faces, and real time where images can be obtained from live video and compare the time with single processing.

Introduction

Face detection is one of the popular and most common topics of discussion. Face detection is used to take better photos by camera or videos. Face detection is used as preprocessing for face recognition of someone. Face detection is also used for face unlock on devices.

In this project, we are going to implement the multiprocessing in python and make the detection faster. We are going to compare the speed or execution time by implementing single processing and multiprocessing in the cases of image, video and live stream.

Summary of Work

In our work, we have taken three different scenarios of input data i.e., image, video and live stream, and have tested the time of execution with respect to utilization of single core process and two core, four cores, six cores and eight cores of multicore process.

The result obtained is then compared and the conclusion is carried out on the basis of that. In the scenario of image, we have used different images with different numbers of faces. In the scenario of video, we have used different videos of different size, length, resolution and number of frames with in crowded and gathering modes. In the live scenario, we used the webcam of our laptops when I was alone and when I was with my family members.

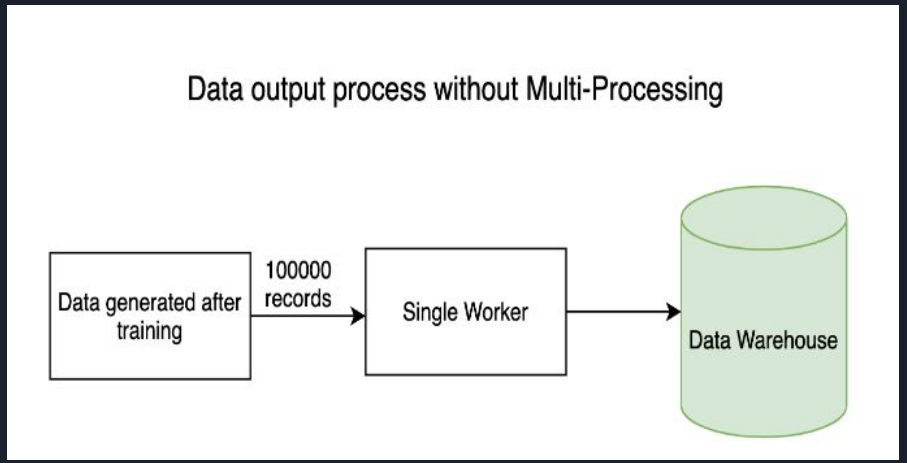
Dataset

haarcascade_frontalface_default.xml is a dataset provided by OpenCV. This xml file contains features of Faces. Since there are very large number of features, they are grouped into different stages with internalNodes and leafValues.

Link :
https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

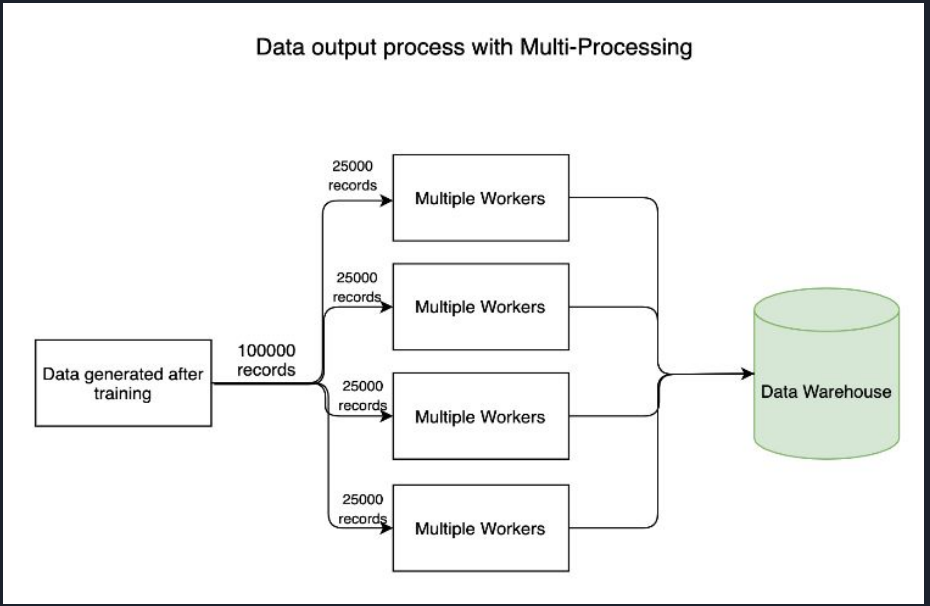
Architecture Comparison

Single core Architecture



Architecture Comparison

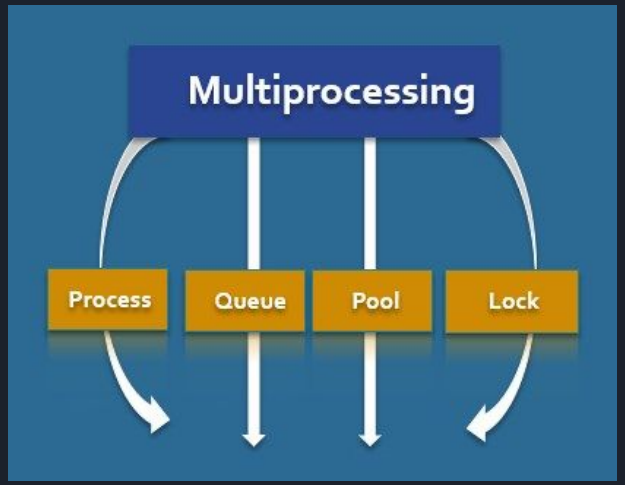
Multi - core Architecture



Python Modules

OpenCV : OpenCV is a famous and most used library in python for image processing.

Multiprocessing: This is a python library to implement multiprocessing.



Parallel Implementation

Image/ live stream :

For massive faces in an image, the process of identifying faces by drawing squares is parallelized using concurrent.Future library. Live is similar to taking images as input continuously in real-time. So, the implementation is similar to the image.

Video:

Video is a collection of images called frames and it streams to the last frame. So, parallelizing for video is done using data parallelism.

- First split the video into a collection of frames.
- Each collection is parallelized with different processors.
- Merge the collections again.
- Each collection can be multithreaded to make the process even faster.

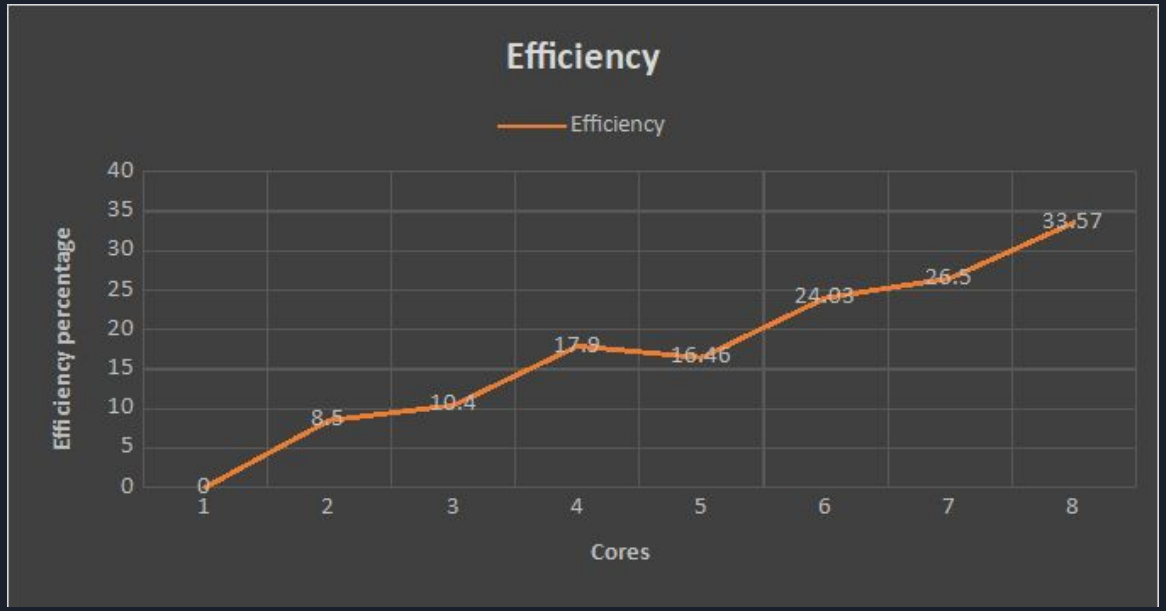
Method used for comparison

The performance of the programs is compared based on total execution time and frame per second. In the scenario of image, execution time and no of faces per second are two comparison parameters.

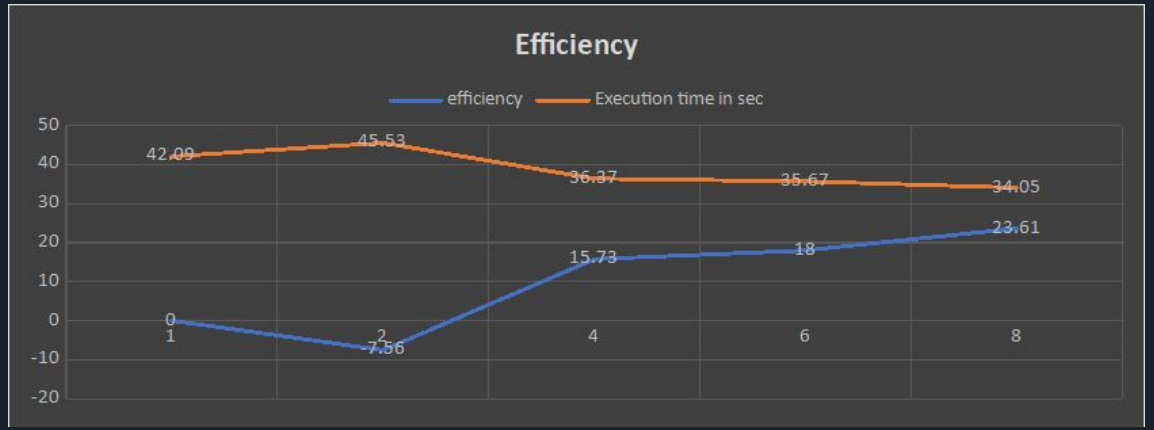
In the scenario of video, total execution time and frame per second are two comparison parameters. In the case of live streams, the total execution time is invalid so only frame per seconds.

Since there is no total frame in the live scenario, we got execution time t of 10 frames and then calculated fps by fps = (10/t).

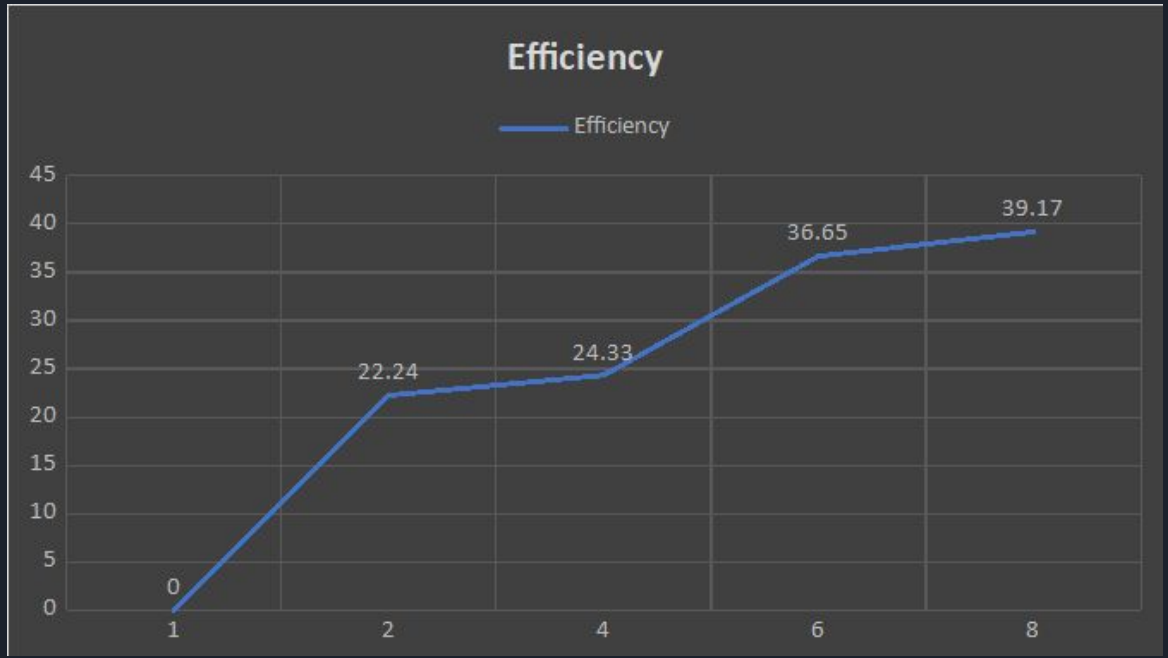
CPU CORE	Average execution time	Efficiency as compared to single core
1	1.91	0
2	1.76	8.5
3	1.73	10.40
4	1.62	17.9
5	1.64	16.56
6	1.54	24.03
7	1.51	26.5
8	1.43	33.57



Video 162 frames, Duration = 5 sec, Width = 2560, Height = 1440		
CPU CORE	Average execution time	Efficiency as compared to single core
1	42.09 3.84fps	0
2	45.53 s 3.56 fps	-7.56
4	36.37 s 4.45 fps	15.73
6	35.67 sec 4.54 fps	18.00
8	34.05 sec 4.76 fps	23.61



Video 3180 frames duration = 1:46, Width = 1280, Height = 720		
CPU CORE	Average execution time	Efficiency as compared to single core
1	177.65 s 17.9 fps	0
2	145.33 s 21.88 fps	22.24
4	142.89 s 22.25 fps	15.7322.33
6	130 s 24.46 fps	36.65
8	127.65s 24.46 fps	39.17



Conclusion

The time taken by multi core to detect faces is less as compared to that by single core to do the same. The pooling in multiprocessing and concurrency of python distributes the data across the cores and hence the time reduces. The efficiency of the algorithms increases with increase in number of faces in image, number of faces in live stream, number of frames in video, the length of the video.