International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 04 | April - 2022 Impact Factor: 7.185 ISSN: 2582-3930

FACIAL RECOGNITION SYSTEM USING HETEROGENEOUS PROGRAMMING

Vaibhav Kovela[1], Rishika[2], Samantha[3], Chandaka Babi[4].
[1,2,3]Student, Department of Computer Science Engineering, Gitam University , Visakhapatnam.
[5]Assistant Professor, Department of Computer Science Engineering, Gitam University , Visakhapatnam.

Guide - Dr. Chandaka Babi

ABSTRACT

Face recognition technology is a biometric technology that uses a person's face traits to identify them. Face photos are collected by people, and the recognition technology processes them automatically. It starts with detection, which involves differentiating human faces from other things in the picture, and then moves on to recognizing those faces that have been discovered. In this research project, a deep learning model based on the ResNet Network is created to recognize the existence of a concerned individual's face kept in the database. The most common HOG Classifier was used to complete the detection task, as it is lighter and more accurate than other detectors currently in use. Face detection and identification is a difficult process that needs a lot of computational resources .We experimented with highly-parallel Graphics Processing Units. We have also compared the results of a typical CPU based implementation with our GPU based implementation to chalk out the efficiency and factor of speed-up. We have also developed a Graphical Interface using the Qt Platform to provide ease of usage.

1. INTRODUCTION

Face and facial landmark feature identification on photos is a crucial step in effective face recognition. Face landmark detection is essential for a variety of face analysis applications, including face verification (identification), face morphing, and face overlapping masks. Since the 1980s, rapid facial recognition has become possible. The problem was that these early implementations could only recognize faces under extremely certain circumstances: proper illumination, frontal picture, and so on. Machine learning algorithms are capable of detecting patterns in how faces are displayed on photographs, making face identification one of the most common challenges they solve. The goal of this study was to find highly variable facial patterns and detect them. These algorithms can recognise faces in nearly every situation like in poor light and occlusion.

The Proposed pipeline of the project is defined into 3 major stages :

- Pre Processing Stage
- Facial Detection and Facial Landmark Detection
- Facial Recognition

Since each stage of the pipeline differs in the computational requirements for optimized implementation of the system , we have defined the resource usage of each stage throughout the pipeline. which delivers various plans to pass on a message furtively.

2. LITERATURE SURVEY

Sharma [1] In 2009 introduced the first Graphical Processing Unit(GPU) realization of a face detection algorithm using CUDA. They reached a detection at 19 fps on a (1280 × 960) video stream, which is a good improvement in detection time. However, the accuracy was only 81% with 16 false positives on the CMU test set.

Kong and Deng [2] in 2010 proposed a GPU accelerated OpenCV implementation that achieved between 49.08 ms and 196.73ms (20,4-5,1 fps) on images from (340x240) to (1280x1024).

The proposed approach by **Devrari and Kumar [4]** in 2011 includes enhanced Haarlike features and uses SVM (Support Vector Machine) for training and classification. They achieved 3.3 fps on (2592x1900) images.

Chouchene [3] in 2015 proposed parallel implementation of face detection on Nvidia 310M GPU that could achieve 24 fps for small images (32x32) and only 11 fps for bigger images (1024x1024).
In their other work **Mutneja** and **Singh**, 2019 much more analysis was done for the algorithm and they achieve 25 fps for (480x640) images. By introducing guassian filters in the preprocessing phase to eradicate noise existance and Implementing contrast enhansement using (IMF-CLAHE).
Patidar [6] in 2020 presented an optimized parallel face detection system using CUDA on GPU and achieved 1.28 fps in the FDDB image set. Although interesting results were recorded, a lack of information about the used classifier and accuracy and false-positive and the test data set in most of the works.
Fayez[5] in 2016 proposed an image scanning framework using General Purpose Graphical Processing Unit (GPGPU) in which they have implemented the Viola-Jones algorithm. They have achieved 37fps for (1920x1080) images.
A Haar-based face detection for (1920x1080) video on Nvidia GTX 470 was proposed by **Oro [7]** in 2011 and 2012 and achieved a performance of 35 fps. (Tek and Gokmen, 2012) used 3 GPUs for the implementation and they achieved 99 fps with good detection rate even though the classifier was small.
Hefenbrock [9] presented a multi-GPU implementation. They used a desktop server containing four Tesla GPUs for the implementation and achieved 15.2 fps. However, the integral image computation was not parallelized.
Nguyen [8] in 2013 they used 5 Fermi GPUs and improved in efficiency by using a dynamic warp scheduling approach to eliminate thread divergence. They used the technique of thread pool mechanism to significantly alleviate the cost of creating, switching, and terminating threads. They reported realized 95.6 fps on (640x480) images.

3. METHODOLOGY

3.1 Histogram of Oriented Gradients:

In light of the condition, there are numerous strategies of encoding wherein picture decipherment is most well known method. Practically all advanced record organizations can be utilized for decipherment, yet the arrangements that are more appropriate are those with a serious level of overt repetitiveness. Overt repetitiveness can be characterized as the pieces of an article that give precision far more noteworthy than needed for the article's utilization and show. The excess pieces of an item are those bits that can be modified without the change being distinguished without any problem. Picture and sound documents particularly conform to this necessity, while research has likewise revealed other record organizes that can be utilized for data stowing away. It very well may be isolated primarily into four classifications:

The HOG Descriptor's working is basically divided into 5 phases

- Gradient Computation
- Orientation binning
- Descriptor blocks
- Block normalization
- Object recognition

3.1.1 Gradient Computation

This is a type of preprocessing step that is frequently omitted in reality since preprocessing is usually done according to the programmers' wishes and requirements.

3.1.2 Orientation Binning

The creation of cell histograms is the second stage in the process. Based on the values determined in the gradient computation, each pixel within the cell casts a weighted vote for an orientation-based histogram channel. The histogram channels are evenly spread over 0 to 180 degrees or 0 to 360 degrees, depending on whether the gradient is "unsigned" or "signed." The cells themselves can be rectangular or radial in shape, and the histogram channels are evenly spread over 0 to 180 degrees or 0 to 360 degrees, depending on whether the gradient is "unsigned" or "signed."

3.1.3 Descriptor Blocks

The HOG descriptor is therefore the concatenated vector of the components of all of the block areas' normalised cell histograms. These blocks frequently overlap, implying that each cell contributes to the final description many times. Rectangular R-HOG blocks and circular C-HOG blocks are the two most common block geometries.

The number of cells per block, the number of pixels per cell, and the number of channels per cell histogram are the three characteristics that define R-HOG blocks.

3.1.4 Block Normalization

Rather of normalising each histogram separately, the cells are first aggregated into blocks and then normalised using all of the blocks' histograms. The histograms of the four cells inside the block are concatenated into a vector with 36 components for this block normalisation (4 histograms x 9 bins per histogram). To normalise this vector, divide it by its magnitude.

3.1.5 Object Detection

HOG descriptors may be used to recognise objects by feeding them into a machine learning system as features. HOG descriptors, on the other hand, aren't connected to any particular machine learning technique.

3.2 Residual Networks (ResNet)

An artificial neural network called a residual neural network (ResNet) is a type of artificial neural network (ANN). Skip connections, or shortcuts, are used by residual neural networks to hop past some layers. The most common ResNet models include double- or triple-layer skips with non-linearities (ReLu) and batch normalisation in between. To learn the skip weights, an extra weight matrix might be employed. HighwayNets are the name for these models. DenseNets are models that have several parallel skips. A non-residual network can be characterised as a plain network in the context of residual neural networks.

There are two primary reasons for adding skip connections: to minimise disappearing gradients and to mitigate Degradation (accuracy saturation). When more layers are added to a sufficiently deep model, the training error increases. The weights adjust throughout training to muffle the upstream layer and magnify the previously skipped layer. Only the weights for the neighbouring layer's link are changed in the simplest instance, with no explicit weights for the upstream layer. When a single nonlinear layer is stepped over, or when the intermediate layers are all linear, this method works well. If not, an explicit weight matrix for the missed connection should be learnt.

In the early phases of training, skipping layers effectively simplifies the network by employing fewer layers. Because there are fewer layers to propagate through, the influence of disappearing gradients is reduced, which speeds up learning. As the network learns the feature, it progressively recovers the skipped levels. When all layers are extended at the conclusion of training, it stays closer to the manifold and so learns quicker. The feature space is explored further by a neural network with no remnant pieces. This makes it more susceptible to disturbances that lead it to depart off the manifold, and thus takes additional training data to recover.

This network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into residual network.

Residual Block : This design introduces the notion of a Residual Network to overcome the problem of vanishing/exploding gradients. A technology called skip connections is used in this network. The skip connection links straight to the output after skipping a few stages of training.

3.3 Parallel Computation

Parallel computing is a method of utilising numerous computer resources at the same time. To facilitate effective use of these resources, the operating system must be built correctly. When processors have several cores, a well-designed programme may make optimal use of all of them. Typically, this type of application is divided down into separate components that may be addressed in parallel.

3.3.1 Graphical Processing Unit (GPU)

GPUs were created with the purpose of processing vertices and polygons, creating 3D objects for video games, and displaying data on displays in near real-time. GPUs did not become a key accelerator for a wide range of calculations until the development of programmable shaders and high-level languages. Demanding applications may take use of the GPU's massive parallelism in this

new era of GPU computing. A GPU's hundreds of cores enable it to speed data-parallel applications while also attaining lower power budgets (when compared to a CPU). In the general purpose market, GPUs have progressed quickly, and they are now widely employed by the scientific community to speed up computations.

3.4 Experimental Setup

In order to give us with an acceptable computing platform, we chose a desktop PC with a decent GPU and a respectable frame rate. The features of this machine are described in Table 1. We take use of the NVIDIA Pascal Architecture's acceleration. This GPU contains 384 CUDA cores and 2 GB of GDDR5X RAM memory, allowing it to accelerate neural network models with tremendous computational power. With 8 GB of DDR3 RAM memory overclocked to 2.7 GHz, the Core i5 CPU has four physical cores and eight logical cores thanks to hyper-threading.

Table 1: Desktop Characteristics

CPU	Intel i5 @ 3.9 GHz
GPU	Nvidia MX250
RAM	8 GB
SSD	250 GB

4. IMPLEMENTATION

We investigated the HOG Descriptor's performance and accuracy in identifying faces and producing descriptors in the first stage of development. We used Dlib's Predefined package to implement HOG. We upscaled the image by two times during the preparation step. After that, we input the picture to the HOG Detector to acquire the Facial detections, also known as Face chips. We bundle them in a specified format before feeding them to the ResNet Network to acquire the descriptors for that face.

In a subsequent stage, we input the structure from the HOG Detector to the ResNet network, which is a Dlib Library-predefined GPU-implemented network. The descriptors are provided by the network model once we feed it the photos. We compare the picture descriptors provided with those in the database. We consider it positive if the distance between the descriptors is less than the given threshold. Figures 1 and 2 depict our pipeline in action.

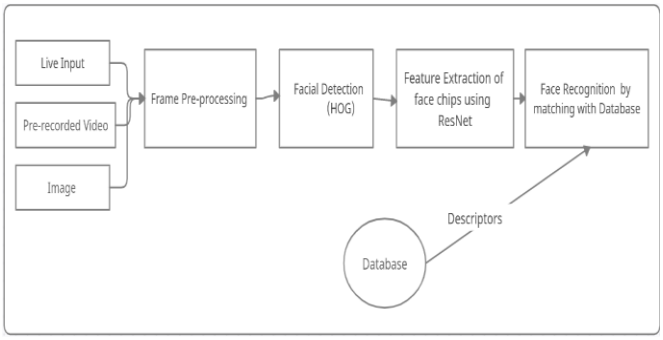

Fig. 1 : Main Recognition Pipeline

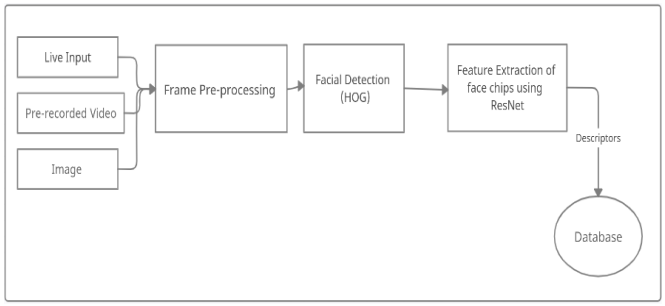

Fig. 2 : New Face Addition

5. RESULTS

The resultant system was very accurate in recognising faces and the our system could predict faces at a frame rate of 14 fps on an average whereas frame rate of the algorithm when implemented on CPU was 5 fps average. The given below are some of our output images .

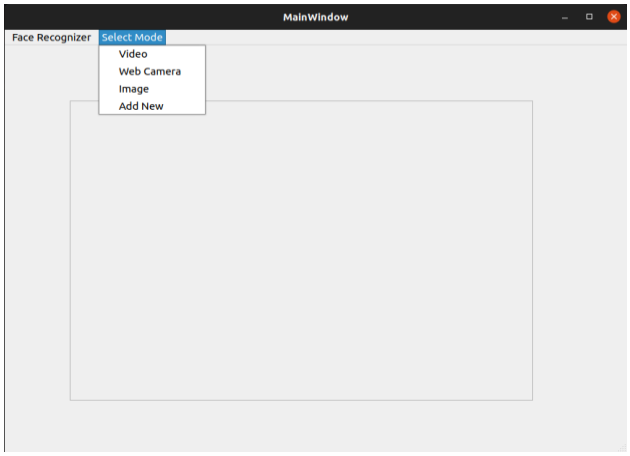

Fig. 3: Home Page with options

Fig. 4 : Results from a live camera input

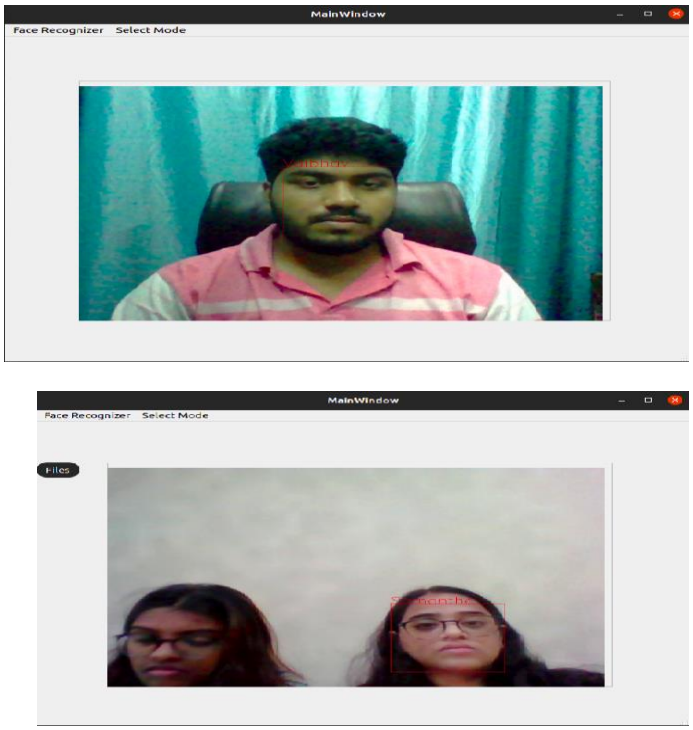

Fig. 5 : Results from an individual image

6. CONCLUSION

The major goal of this thesis was to see if it was possible to do real-time face identification in video streams. Using our parallel pipeline system, we were able to demonstrate that the framework can handle a 480p quality video at a reasonable frame rate. We may apply this framework to other applications with the objective of proving real-time face detection performance. There are a significant variety of image/video processing applications that can benefit from a comparable CPU-GPU platform using an accelerated system that can combine the performance of a GPU with the speed and flexibility of a CPU with many cores. There is a lot of space for more study into putting single stage detectors on GPU to boost calculation performance, which might eventually detect in real-time which is the final goal of this project.
Combining single and multistage detectors with a Deep Neural Network-based object detector might make this more lighter and more precise than previous efforts. Additional optimizations might be sought to increase the framework's speed.

7. REFERENCES

[1] Sharma, B., Thota, R., Vydyanathan, N., and Kale, A. (2009). Towards a robust, real-time face processing system using cuda-enabled gpus. In 2009 International Conference on High Performance Computing (HiPC), pages 368–377. IEEE.

[2] Kong, J. and Deng, Y. (2010). Gpu accelerated face detection. In 2010 International Conference on Intelligent Control and Information Processing, pages 584–588. IEEE.

[3] Chouchene, M., Sayadi, F. E., Bahri, H., Dubois, J., Miteran, J., and Atri, M. (2015). Optimized parallel implementation of face detection based on gpu component. Microprocessors and Microsystems, 39(6):393–404.

[4] Devrari, K. and Kumar, K. V. (2011). Fast face detection using graphics processor. International Journal of Computer Science and Information Technologies, 2(3):1082–1086.

[5] Fayez, M., Faheem, H., Katib, I., and Aljohani, N. R. (2016). Real-time image scanning framework using gpgpu-face detection case study. In Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), page 147. The Steering Committee of The World Congress in Computer Science, Computer . . . .

[6] Patidar, S., Singh, U., Patidar, A., Munsoori, R. A., and Patidar, J. (2020). Comparative study on face detection by gpu, cpu and opencv. Lecture Notes on Data Engineering and Communications Technologies, 44:686–696.

[7] Oro, D., Fernandez, C., Saeta, J. R., Martorell, X., and ´ Hernando, J. (2011). Real-time gpu-based face detection in hd video sequences. In 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pages 530–537. IEEE.

[8] Nguyen, T., Hefenbrock, D., Oberg, J., Kastner, R., and Baden, S. (2013). A software-based dynamic-warp scheduling approach for load-balancing the viola– jones face detection algorithm

on gpus. Journal of Parallel and Distributed Computing, 73(5):677–685.

[9] Hefenbrock, D., Oberg, J., Thanh, N. T. N., Kastner, R., and Baden, S. B. (2010). Accelerating viola-jones face detection to fpga-level using gpus. In 2010 18th IEEE Annual International Symposium on FieldProgrammable Custom Computing Machines, pages 11–18. IEEE.

[10] Bilaniuk, O., Fazl-Ersi, E., Laganiere, R., Xu, C., Laroche, D., and Moulder, C. (2014). Fast lbp face detection on low-power simd architectures. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 616–622.

[11] Li, E., Wang, B., Yang, L., Peng, Y.-t., Du, Y., Zhang, Y., and Chiu, Y.-J. (2012). Gpu and cpu cooperative accelaration for face detection on modern processors. In 2012 IEEE International Conference on Multimedia and Expo, pages 769–775. IEEE.