Qualitative Study of Face Recognition Algorithm through Hardware-Software Acceleration

Mohammed Minhaas B. S[1], Prithvi Suresh Naidu[1], Prajwal Suresh Naidu[1], and Advaith Biligeri Jagannath[2]

[1]New York University
[2]Columbia University
[1,2]IBM Academy, Center of Excellence in Youth

***Abstract*—Face recognition algorithms have gained widespread use in recent years, but they often require significant computing power to operate effectively. One way to enhance their performance is through the use of hardware acceleration, which involves offloading certain computationally intensive tasks to specialized hardware components. This paper aims to provide a comprehensive study on the effect of software-hardware acceleration on accuracy and speed using pre-trained neural network models. The training data is stationary and consists of labeled images, which are considered to be a public benchmark. Moreover, we will highlight some of the key considerations that should be taken into account when selecting and integrating hardware acceleration into face recognition algorithms. Finally, we are able to optimize computationally time consuming functions of our implementation by using multithreading, and for the rest, we advocate the use of hardware acceleration.**

I. Introduction

Face recognition algorithms have become increasingly popular in a wide range of applications, from security and law enforcement to marketing and social media. According to Yassin Kortli chu et al in [9] these algorithms often require significant computing power to operate effectively, particularly when dealing with large amounts of data or complex recognition tasks. One promising application of face recognition algorithms is the use of them in office spaces in order to detect intruders. This application can be built by leveraging a software component and offloading computationally intensive tasks to specialized hardware components. In this research paper, we aim to provide an OpenCV-powered office space intruder detection application and study the impact of hardware acceleration on the speed of face recognition technology using a pre-trained neural network model. We will use a stationary set of labeled images as our benchmark for testing the performance of the edge-based face recognition application. Additionally, we will discuss key considerations that should be taken into account when selecting and integrating hardware acceleration into face recognition algorithms. Our research aims to provide valuable insights for researchers and practitioners working to optimize the performance of face recognition algorithms in real-world applications. To achieve these objectives, we will study the impact of multithreading and propose hardware acceleration approaches for face recognition algorithms. We will analyze the performance of these approaches in terms of speed and performance and compare them with our edge-based face recognition algorithms in our further implementation. This research has significant implications for the development of more efficient and effective state-of-the art face recognition technology that can be used in offices and restricted spaces. Overall, our study will provide a valuable contribution to the field of face recognition technology by identifying effective hardware acceleration approaches and highlighting key factors to consider when integrating hardware acceleration into face recognition algorithms.

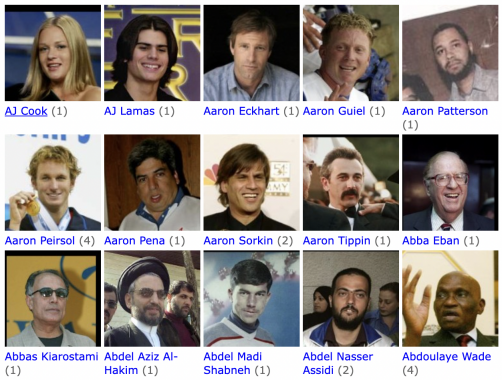

Fig. 1. Labelled Faces in the Wild

II. Dataset

The dataset we use for our model is the Living Faces in the Wild dataset (LFW) [6] which contains 13233 images of 5749 subjects with a variety of poses, lighting, expressions, age, and other parameters. Of the subjects featured in LFW, 1680 have at least two images each, while the rest have one image. In the standard LFW evaluation protocol, the verification accuracy is reported on 6,000 face pairs. Most of these images are colored, but some are in grayscale.

III. Related Work

Nowadays, there are several trustworthy and accurate computer vision algorithms available. They can be combined with embedded system technologies to produce reliable, real-time implementations at reasonable costs.

This paper [1] elaborates on the performance of face recognition systems that have recently been put into use and compares face recognition platforms and algorithms for accuracy

and processing speed. It suggests that in face detection and recognition systems, GPUs outperform other hardware accelerators, and neural network-based algorithms outperform non-neural network algorithms in terms of accuracy. We further learn that for relatively modest face databases, FPGA-based systems outperform GPU-based edge or Internet of Things platforms in terms of processing throughput while consuming less power. Therefore, it is likely best to integrate both technologies onto a single chip and focus on creating appropriate hardware-software development tools that will make it simple for users to divide algorithms between the two technologies.

According to M. P. Beham and S. M. M. Roomi [2] face recognition technologies and various phases of development related to the face recognition development cycle and method, such as soft computing-based, feature-based, and appearance-based. The study found that, with a normalized rate of 95 percent, feature-based face recognition algorithms outperformed appearance- and soft computing-based algorithms in terms of recognition accuracy.

D. Zhang, J. Li and Z. Shan talk [11] about face recognition approach using Python and Dlib, an open source library that improves face identification and detection for static images, dynamic video, and real-time camera footage, with high accuracy and efficiency. This technique has strong robustness for occlusion and may be used to perform feature calibration and face recognition using Python, using a huge set of trained face model interfaces. The results of the experiments show that the suggested method does a better job than the OpenCV methods for face detection, feature vector feature, point calibration, extraction, and comparison in pictures and videos with small deflections and good faces.

Talking about hardware acceleration, M. Turk and A. Pentland proposes [8] that there is a direct correlation between the number of cores and processing rate. When comparing the fixed hardware accelerators, GPU and multi-core CPU, GPUs perform better than CPUs in terms of performing tasks efficiently, due to the multiple number of cores and ability to provide fine-grain parallelism [3].

Compared to edge-based GPUs, cloud-based GPUs offer a significantly higher processing rate [7]. However, issues including slow reaction times, bandwidth restrictions, and data privacy are associated with cloud computing. The use of cloud-based GPUs for time-sensitive applications is therefore not recommended.

Findings show [4] that while using scaled photos speeds up the face detection stage, accuracy is sacrificed. We discovered that the degree of the face detector's complexity determines the speedup brought about by employing resized photos and GPUs. As a result, processing times for advanced detectors have greatly increased. Using the DSFD detector on images that have been shrunk to 25 percent of their original size in CPUs and 50 percent of their original size in GPUs produces the optimal speed-accuracy tradeoff. The CPU i9-8950HK and GPU RTX 2060 also produced the best results.

FPGAs are also quite appealing in terms of power and execution speed. They cannot, however, fully replace multi-core CPUs and GPUs. It appears that the best course of action is a heterogeneous system-on-chip that combines these three technologies. For instance, it is simpler to program GPUs for complex operations when power consumption is not a concern, and an FPGA core can speed up certain processes that require a lot of datapath.

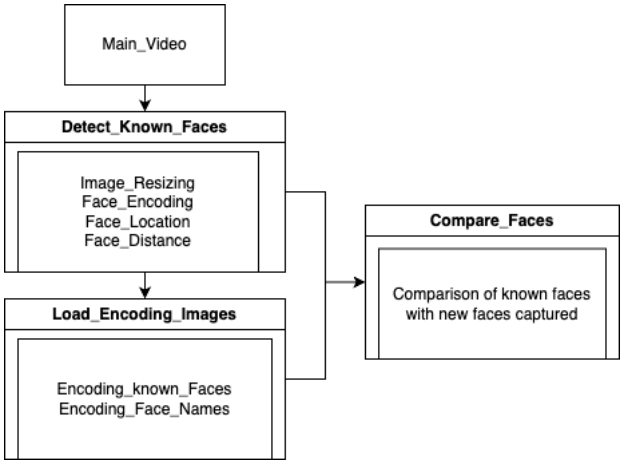

Fig. 2. Methodology Workflow Chart

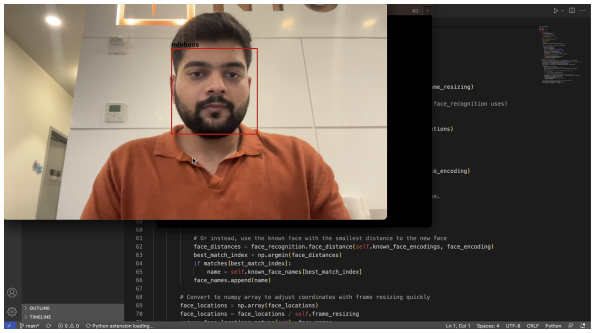

Fig. 3. Face Recognition Application

IV. Methodology

A. Face recognition

We use transfer learning methods from *OpenCV*, a Python library, to implement face recognition in our application. We also make use of the *Dlib library* which is used to construct our face embeddings used for the actual recognition process. In our implementation, the first task that we perform is detecting faces in the video stream. We try to capture the exact coordinates or location of the face on the screen using *face_locations*, so we extract this face for further processing.

In the next step, we crop the face from the image and extract face features from it using face embedding vectors. On training the network, the model learns to output similar feature vectors using *face_encodings* which returns the 128-dimension face encoding for each face in the image that looks similar to the known embeddings we are aware of.

Given that we have the face embeddings for known images stored in our database, we used this to recognize new faces in

real time. Therefore, we first compute the face embedding for the images in the frame using the same network we used earlier for training and then compare this embedding with the rest of the embeddings that we have using *face_compare_faces*.

B. Application Profiling

A profile is a set of statistics that describes how often and for how long various parts of the program are executed. We profiled this application using cProfile, a library that provides deterministic profiling of Python programs to identify potential performance bottlenecks. Later, we use these statistics by converting them into a function call graph using the gprof2dot library. Using these results, we understand where we can apply hardware acceleration to speed up our application.

V. Results

Our edge based face recognition application executes [1] in approximately 7 seconds on an Apple M2 processor, which is a general purpose ARM based system on a chip (SoC), this runtime is considered to be extremely slow, as the purpose of our application is to be real-time. From the result [2] of our application profiling, we refer to Figure 5, which is a zoomed in version of Figure 4, and Table 1 to see that there are two main bottlenecks *load_encoding_images()* and *cv2.VideoCapture()*, the former is a user defined function in our application, which upon further inspection in the graph reveals that *face_encoding()* is the method being called which is causing the most performance degradation. We try to remove this overhead by using the Python language's multi-threading functionality and creating a pool with 4 processes. This effectively bypasses the Global Interpreter Lock (GIL) to use multiple processes to encode images in parallel. Following this, as observed in Table 2, we see that we were able to reduce the total time required for face_embeddings was reduced from 0.034 / function call to 0.008 / function call which is a time reduction of $\approx 46\%$.

For the overhead caused by *cv2.VideoCapture*, we propose to use a purpose-built I/O hardware accelerator, which will be discussed in the future works section.

ncalls	totaltime / call	method
7	0.720	face_encodings (load_encoding_images())
1	0.830	cv2.VideoCapture
1	0.202	list_comprehension

TABLE I
CODE PROFILING DATA BEFORE MULTITHREADING

ncalls	totaltime / call	method
7	0.388	face_encodings (load_encoding_images())
1	0.819	cv2.VideoCapture
1	0.230	list_comprehension

TABLE II
CODE PROFILING DATA AFTER MULTITHREADING

[1]One execution is defined as an average time taken for three inferences.
[2]All results are reported as an average of three execution cycles.

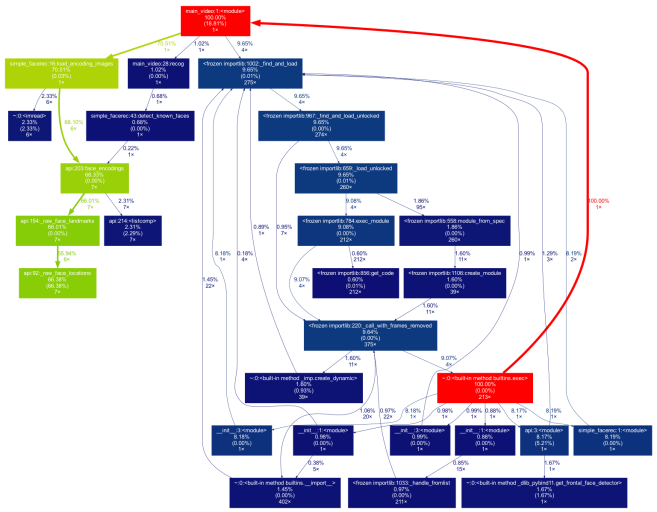

Fig. 4. Code profiling digraph to identify performance bottlenecks

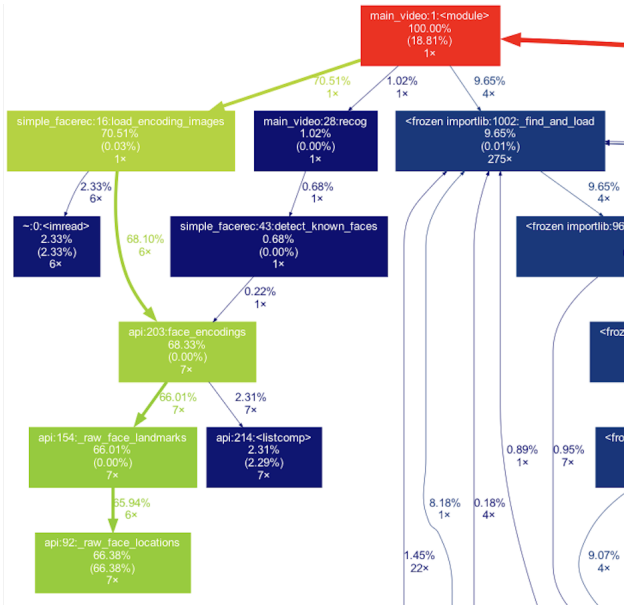

Fig. 5. Zoomed in version of code profiling digraph for the bottleneck functions

VI. Future Work

Although efficient algorithms can speed up existing face recognition algorithms, building more efficient hardware can significantly improve the performance of face recognition systems. According to the research carried out by Chen et al in [10] the authors explore the use of FPGAs for face recognition. The researchers demonstrate that using FPGAs can significantly improve the performance of face recognition systems, particularly in terms of speed and power consumption. They also discuss the benefits of using aging mechanisms in face recognition algorithms to improve the program's performance over time. FPGAs may perform computations more efficiently than traditional CPUs, resulting in lower power consumption; this is especially relevant in battery-powered devices like

smartphones and wearable electronics. Another technique that has shown promise in improving the performance of face recognition algorithms is the use of denoising and contrast enhancement. In a study published in the Journal of Electrical and Computer Engineering, the authors in [5] explore the use of denoising and contrast enhancement techniques to improve the performance of a face recognition system in low-light conditions. The researchers demonstrate that by introducing these techniques into the algorithm, the system's performance is significantly improved, particularly in challenging conditions like low-light environments. The slow runtime of a face recognition system can be a significant drawback, particularly in real-world applications where fast processing is essential. To address this issue, one approach is to design and implement a camera on a chipset by integrating both the camera hardware and face recognition algorithms into a single system. This can significantly improve the performance and speed of the system, as the hardware and software are designed to work together seamlessly. The chipset could also be designed to perform face recognition algorithms more efficiently, using specialized hardware like FPGAs or ASICs. One example of a camera chipset optimized for face recognition is the Snapdragon 845 Mobile Platform from Qualcomm. This chipset includes a Spectra 280 Image Signal Processor, which is designed to capture high-quality images in a range of lighting conditions. The chipset also includes a Hexagon 685 DSP, which can perform complex image processing tasks like face recognition more efficiently than a traditional CPU. Overall, designing and implementing a camera on a chipset can significantly improve the speed and performance of a face recognition system. This can enable faster and more accurate face recognition in real-world applications like security systems, mobile devices, and more.

VII. Conclusion

In conclusion, this paper presents a comprehensive study on the impact of software-hardware acceleration on the performance of face recognition algorithm. The research focuses on utilizing a pre-trained neural network model and a stationary set of labeled images as a benchmark for testing the performance of a face recognition algorithm, to identify bottleneck functions with high latency through application profiling and using software optimizations such as multi-threading for parallel processing to enhance the performance of these functions, which often require significant computing power.

Additionally, techniques such as denoising, contrast enhancement, and camera chipset optimization were proposed as future avenues for improving face recognition performance.

These advancements aim to enhance the real-time performance and accuracy of face recognition technology in various applications, including security systems and mobile devices.

VIII. Acknowledgement

We would like to express our sincere gratitude to IBM Academy sponsored Center of Excellence in Youth (CEY) program who supported and contributed to the completion of this research paper. Your invaluable guidance, assistance, and encouragement have been instrumental in our journey.

References

[1] A. Baobaid, M. Meribout, V. K. Tiwari, and J. P. Pena, "Hardware accelerators for real-time face recognition: A survey," 2022.

[2] M. P. Beham and S. M. M. Roomi, "A review of face recognition methods," 2013.

[3] S. Bhutekar and A. Manjaramkar, "Parallel face detection and recognition on gpu," 2014.

[4] D. Chaves, E. Fidalgo, E. Alegre, R. Rodriguez, F. Martino, and G. Azzopardi, "Assessment and estimation of face detection performance based on deep learning for forensic applications," in *Sensors (Basel), vol. 20, no. 16, p. 4491, Aug. 2020. doi: 10.3390/s20164491*, 2020.

[5] . S.-A. K. . Garg, N., "A novel approach for facial recognition using denoising and contrast enhancement in low light environment," in *Journal of Electrical and Computer Engineering, 2019, 1-7. doi: 10.1155/2019/3230256*, 2019.

[6] http://vis www.cs.umass.edu/lfw/, "Lfw dataset."

[7] A. Koubaa, A. Ammar, A. Kanhouch, and Y. AlHabashi, "Cloud versus edge deployment strategies of real-time face recognition inference," in *IEEE Transactions on Network Science and Engineering, vol. 9, no. 1, pp. 143-160, Jan. 2022.*, 2022.

[8] N. Muslim and S. Islam, "Face recognition in the edge cloud," 2017.

[9] M. Turan, "Facial recognition systems: A survey," in *Journal of Ambient Intelligence and Humanized Computing, vol. 11, pp. 2917-2931, 2020. doi: 10.1007/s12652-020-02723-4*, 2020.

[10] Y.-C. X. Y.-S. T. N. K. Ying-Hao, Yu., "Efficient face recognition using fpga and semantic features for security controls," in *34th International Symposium on Automation and Robotics in Construction doi: 10.22260/ISARC2017/0141*, 2017.

[11] D. Zhang, J. Li, and Z. Shan, "Implementation of dlib deep learning face recognition technology," in *2020 International Conference on Robots*
Intelligent System (ICRIS), Sanya, China, 2020, pp. 88-91, doi:10.1109/ICRIS52159.2020.00030., 2020.