

# AI Based High Performance Authentication System for Industries

**K Indra Gandhi, Jawahar PK, G. Kannan**

*Department of ECE, B.S.Abdur Rahman Crescent Institute of Science & Technology, India.*

*Email: [indra@crescent.education](mailto:indra@crescent.education)*

Face recognition technology has gained significant popularity in a computer vision applications, such as surveillance, object tracking, human identification, and self-driving cars. Its non-intrusive nature makes it particularly desirable for security purposes. This paper emphasizes the development of a real-time video-based facial recognition system, employing Artificial Intelligence (AI) to facilitate swift authentication. The main aim is to enhance the processing time of face recognition while maintaining a high level of accuracy. To achieve this objective, two separate algorithms are utilized for face recognition: the Histogram of Oriented Gradients (HoG) with Support Vector Machine (SVM) on the CPU, and the Convolutional Neural Network (CNN) with CUDA on the GPU. Experimental findings demonstrate that the CNN algorithm achieves an accuracy rate of 91.69%, while the HOG+SVM algorithm achieves 62.91% accuracy in real-time face recognition. These findings carry practical implications for the Nokia industry in real-time video-based employee authentication, thus promoting the adoption of secure and efficient face recognition technology in diverse domains.

**Keywords:** Computer vision, Face recognition, HOG, CNN, CUDA.

## 1. Introduction

Facial recognition systems are software applications designed to verify or identify individuals from digital images, finding applications in identity authentication, security, and access control. The technology has gained significant interest over the past decade due to its potential as a user-friendly and minimally invasive method of human identification [1]. While humans can easily recognize faces, automating this process presents challenges. Facial recognition algorithms must contend with variations in lighting, facial expressions, and perspectives, making it a complex task for modern computer systems.

Deep learning techniques, such as convolutional neural networks (CNNs), have demonstrated remarkable success in various computer vision tasks, including image classification, segmentation, and object detection. These methods mitigate preprocessing challenges and enhance evaluation techniques. Real-world applications of facial recognition systems include automatic tagging on social media platforms and face unlock features in mobile applications. Other algorithms, like the histogram of oriented gradients (HOG), have also been employed

for facial recognition. This technique quantifies occurrences of gradient orientations in localized image regions, computed on a dense grid of uniformly spaced cells [6].

Ishita et al. [2] aimed to develop a cost-effective security surveillance system using Raspberry Pi kits, incorporating traditional face detection methods such as Haar detection and PCA. They suggested using RF I-cards instead of passwords for enhanced security in specific areas. The study utilized Haar cascade and Eigenface approaches for adaptive face detection. Jie Wang et al. [3] investigated Convolutional Neural Networks (CNNs), emphasizing the importance of convolutional and downsampling layers to reduce memory requirements. Ciresan et al. [4] achieved near-human performance using artificial neural network architectures with receptive fields in convolutional layers. Wazwaz et al. [5] implemented a network with a microcomputer and camera for human face analysis on a Raspberry Pi, utilizing algorithms for face detection and recognition. Savath Saypadith et al. [7] proposed a CNN-based framework for face detection, tracking, and recognition on the NVIDIA Jetson TX2 board, demonstrating real-time recognition of multiple faces. Effective facial recognition systems today require substantial and diverse datasets to achieve accuracy. They must adeptly manage facial data variations, accurately identify features, and operate in real-time, necessitating significant RAM resources. Research indicates that integrating GPUs into these systems significantly enhances processing speed, enabling real-time analysis and backend processing.

To achieve faster facial recognition while ensuring accuracy and efficiency, selecting algorithms that are both fast and accurate is crucial. GPUs play a pivotal role in accelerating the recognition process. This paper aims to compare the efficiency of the HOG + linear SVM and CNN algorithms for facial recognition, providing a comparative analysis to identify the most efficient technique. Our objective is to leverage available resources to optimize facial recognition systems for efficiency and accuracy. The study evaluates the performance of facial recognition systems using the HOG+SVM algorithm on the CPU and the Convolutional Neural Network (CNN) algorithm on the GPU with CUDA, analyzing real-time video data to assess accuracy and speed.

## **2. Methodology:**

The proposed system consists of three main phases: capturing facial images, encoding facial datasets, and conducting real-time face recognition.

### **Phase 1: Capturing Face Images**

The process begins by launching the program and initiating video capture. In this step, the system actively detects a single face within the frame and proceeds to capture images of the same face from various angles. It actively organizes these diverse pictures of the same individual systematically and stores them within a dedicated folder, with each folder corresponding to a specific person's name. This process repeats for all individuals who require recognition. Once completed, the system actively has a collection of facial data neatly labeled in folders according to each person's identity. Finally, consolidate all these labeled facial datasets into a central dataset folder, ready for use in face encoding purposes. This phase is responsible for capturing images of faces in real-time. It involves the following steps:

1. **Importing Libraries:** The code begins by importing libraries necessary for image processing, video capture, and argument parsing. These libraries include OpenCV (cv2), imutils, time, and argparse.
2. **Argument Parsing:** It constructs an argument parser to accept command-line arguments. Specifically, it requires the path to the Haar Cascade classifier file for face detection and the directory where captured face images will be saved.
3. **Loading the Cascade Classifier:** It loads OpenCV's Haar Cascade Classifier for face detection using the provided cascade file path.
4. **Initializing the Video Stream:** The code initializes a video stream using the video stream class from the imutils video module. This prepares the camera sensor for use and allows it to warm up for 2 seconds.
5. **Frame Capture and Processing:** It enters a loop that continuously captures frames from the video stream. For each frame, the following actions are performed:
  - a. A copy of the original frame is made for potential saving.
  - b. The frame is resized to a width of 400 pixels to speed up face detection.
  - c. Faces are detected in the grayscale version of the frame using the loaded Cascade Classifier.
  - d. Bounding boxes are drawn around detected faces.
  - e. The processed frame is displayed on the screen.

Each image is named sequentially, starting from "00001.png" and incrementing with each capture.

## Phase 2: Encoding Face Datasets

This phase is responsible for encoding the face datasets. It involves the following steps:

1. **Importing Libraries:** The code imports necessary libraries, including imutils, paths for listing image files, face\_recognition for face-related operations, and pickle for serialization.
2. **Argument Parsing:** It constructs an argument parser to accept command-line arguments. The required arguments are the path to the directory containing the face images and the path to the file where the facial encodings will be serialized.
3. **Listing Image Paths:** It collects the file paths of all images in the specified dataset directory.
4. **Initializing Lists:** The code initializes two lists: known Encodings to store facial encodings and known Names to store the corresponding names of individuals.
5. **Looping Over Image Paths:** For each image in the dataset directory, the following actions are performed:
  - The person's name is extracted from the image path.
  - The image is loaded, converted to RGB format, and stored in the image variable.

- Face locations (bounding boxes) in the image are detected.
- Facial encodings for each detected face are computed. The encodings and corresponding names are added to the respective lists. The images are trained using HOG+SVM and CNN algorithm.

6. **Serialization:** After processing all images, the code serializes the collected facial encodings and names into a binary file using the Pickle module. This serialized data can be later used for recognition.

### Phase 3: Real-Time Face Recognition

The process commences by initiating the video capture process. It actively converts the live stream of image data into individual image frames, then actively detects faces in real-time within these frames. Subsequently, it actively compares the extracted facial features with those previously labeled. If a match is found between the features and the labeled faces, it proceeds to actively print the name of the recognized person. However, if recognition fails, it actively prints "Unknown." This process enables real-time face recognition with dynamic feedback.

This phase performs real-time face recognition using the previously encoded dataset. It involves the following steps:

1. **Importing Libraries:** The code imports the necessary libraries, including `imutils.video` for video streaming, `face_recognition` for face recognition, and `OpenCV (cv2)` for image processing.
2. **Argument Parsing:** It constructs an argument parser to accept command-line arguments. The required arguments are the path to the Haar Cascade classifier file for face detection and the path to the serialized facial encodings database.
3. **Loading Encodings and Detector:** It loads the known facial encodings and names from the serialized database file. Additionally, it loads the Haar Cascade classifier for face detection.
4. **Initializing Video Stream:** The code initializes a video stream using `VideoStream` and allows the camera to warm up for 2 seconds.
5. **Frame Processing and Recognition:** The code enters a loop where it continuously captures frames from the video stream and processes them. For each frame, it performs the following steps:
  - Converts the frame to grayscale (for face detection) and RGB format (for face recognition).
  - Detects faces in the grayscale frame using the Haar Cascade classifier.
  - Reorders the detected bounding box coordinates to the (top, right, bottom, left) format expected by `face_recognition`.
  - Computes facial encodings for each detected face.
  - Matches the computed encodings against the known encodings obtained from the dataset.
  - Determines the recognized person's name based on the matching results.

- 6. Display and Exit: The code displays the processed frame with bounding boxes and recognized names.
- 7. FPS Calculation: The Frames Per Second (FPS) is calculated and displayed at the end of execution to give an idea of the processing speed.

3. Proposed Framework

A visual representation of the proposed framework is shown in Figure 1. As an initial step, face detection is performed, and a dataset of human faces is created, which is then labeled. Next, the faces are encoded using the desired method, which can either be HOG+SVM or CNN.

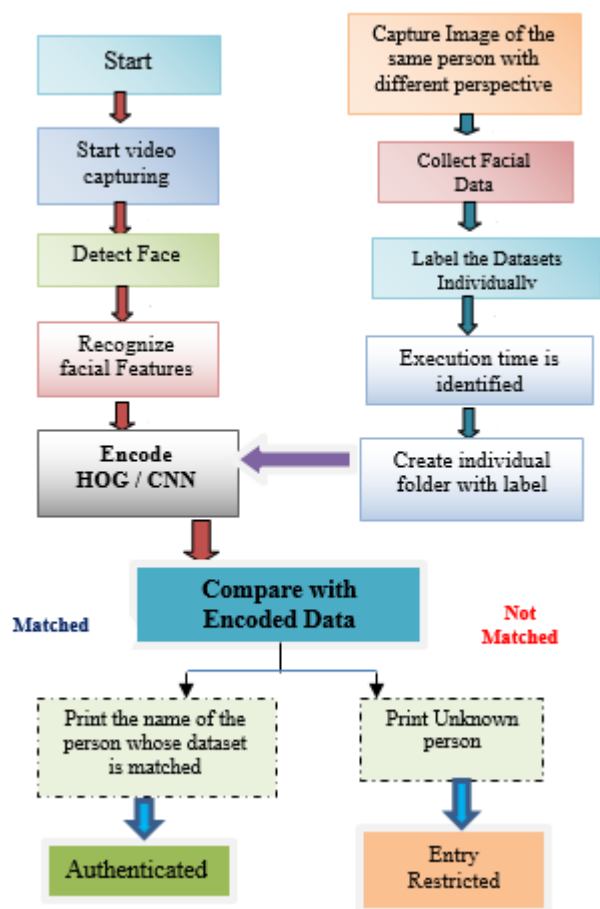


Figure1 Proposed framework

As the results of both algorithms need to be compared, the images are trained with both methods and the outcomes are analyzed. Finally, during the recognition phase, the encoded face is matched with the detected face by the system.

### Building Dataset Using Haar Cascade Classifier

Facial detection plays a crucial role in authentication systems. This process involves the utilization of Haar cascades. Initially, the input image is loaded using the built-in function `cv2.imread`, where the image path serves as the input parameter. The image is subsequently converted to grayscale and displayed [9, 8, 10]. Additionally, the Haar cascade classifier is loaded. In Figure 2, the Haar-like feature is illustrated, encompassing both edge and line characteristics. The white bar in the grayscale image represents pixels near the light source.

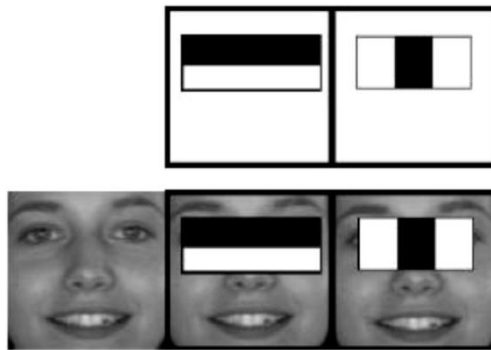


Figure 2. Haar Feature

Haar-like features are rectangular filters applied to an image to identify specific patterns or features. The Haar value (HV) is computed by subtracting the sum of pixel values within a white rectangle (SPW) from the sum within a black rectangle (SPB), as shown in Equation 1.

$$HV = SPW - SPB \quad (1)$$

The flow of a Haar cascade classifier can be summarized as follows:

- **Image Capture:** The process begins with the camera capturing an image.
- **Grayscale Conversion:** The acquired image is converted into grayscale, simplifying the subsequent analysis.
- **Face Detection:** The cascade classifier scans the grayscale image to identify the presence of a face.
- **Eye Detection:** If a face is detected, the classifier then proceeds to search for both eyes within the recognized face region.
- **Eye Verification:** The system checks whether both eyes are successfully identified.
- **Face Normalization:** If both eyes are found, the system normalizes the size and orientation of the face image for consistent processing.

The captured images are stored in a folder, and the name of the folder corresponds to the name of the person in the captured images. This way, "n" number of datasets can be created.

### Encode Dataset

After creating the datasets, the next step is to encode (train) them. The encoding can be done

using either HOG+SVM or CNN.

Histogram of Oriented Gradients (HOG)

In the HOG concept, the approach is centered on the grouping of pixels into small cells rather than using the individual gradient direction of each pixel in an image. Within each cell, gradient directions are calculated and grouped into several orientation bins. The gradient magnitudes are summed for each bin, with greater weight assigned to stronger gradients, thereby reducing the impact of small random orientations resulting from noise. A picture of the dominant orientation in that cell is generated by means of this histogram. This process is applied to all cells, resulting in a representation of the image's structure [12,13,14]. The HOG features maintain the distinct representation of an object while allowing for some variations in shape. An experimental image version enhanced with HOG features is depicted in Figure 3. the dlib is used to estimate the location of 68 coordinates (x, y) that map the facial points on a person's face in the figure 4.

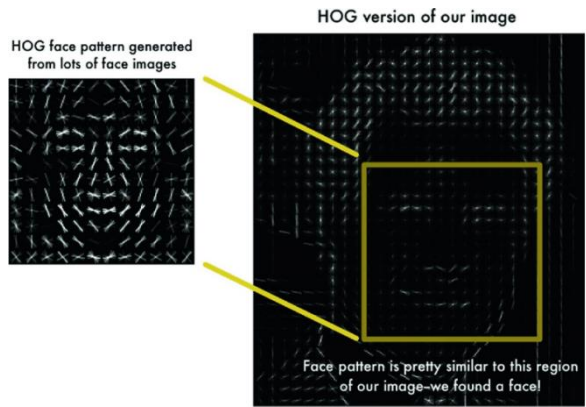


Figure 3. Experimental version of an image with HOG



Figure 4. Coordinates of a human face

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning model equipped with a corresponding learning algorithm, designed for the analysis of data in tasks such as classification and



regression. [15,16]Capable of addressing both linear and non-linear problems, SVM proves effective in a variety of practical scenarios. The algorithm constructs a line or hyperplane to delineate data into distinct classes. SVM is employed in image training by extracting features from images and assigning appropriate labels[28].

### Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a specialized form of multilayer perceptrons, tailored with regulated connectivity patterns. Unlike traditional fully connected networks, where every neuron in one layer is linked to all neurons in the subsequent layer, CNNs exhibit a more structured architecture. They typically consist of an input layer, an output layer, and multiple hidden layers, predominantly comprising convolutional layers. These convolutional layers perform operations such as dot products or element-wise multiplications. Each convolutional layer is usually followed by a Rectified Linear Unit (ReLU) activation function and may include additional layers like pooling, fully connected, and normalization layers. These layers collectively process the input data, making them aptly named "hidden layers" due to the transformations applied by their operations [18, 19].

CNNs are renowned for their capability to handle complex computations across numerous layers. However, the extensive dataset processing can lead to prolonged computation times compared to methods like Histogram of Oriented Gradients (HOG). To mitigate this, leveraging GPU acceleration can significantly enhance processing speed during both training and inference phases [20, 29, 30].

### GPU Programming

CUDA (Compute Unified Device Architecture) is an API and parallel computing platform developed by NVIDIA for utilizing their CUDA-enabled GPUs for general-purpose computing, known as General Purpose computing on Graphics Processing Units (GPGPU). It provides developers and engineers direct access to the GPU's virtual instruction set and parallel computational elements, enabling the execution of compute kernels [21, 22]. GPUs, specialized processors designed for real-time, high-resolution 3D graphics and compute-intensive tasks, are leveraged by CUDA to efficiently meet computational demands. Beyond traditional graphical applications, CUDA has found extensive application in accelerating non-graphical tasks across diverse fields like computational biology and cryptography. These applications often experience significant performance enhancements, sometimes achieving orders of magnitude improvement in processing speed [23, 24,31].

### Recognition

All the prerequisites for face recognition have been completed. Moving on to the recognition phase, when the camera is activated, faces are detected. After detecting a face, a comparison is made with the encoded data[25,27].HOG matches the extracted and encoded features, whereas CNN matches the values of all the layers in the image. This theoretically explains why CNN has higher accuracy since it matches with many layers of the image, whereas HOG only matches with the image features. Then, a boundary box is created around the face and labeled with the matched name.



4. Experiment Setup

Hardware required

In this paper, the HOG algorithm was implemented on an Intel Core i5 with 4GB of RAM, while the CNN algorithm was programmed on an NVIDIA GeForce GTX 1050 Ti, which has approximately 768 CUDA cores. The graphics clock has a speed of 1290 MHz and the processor clock has a speed of 1392 MHz .

Software required

The operating system used is Ubuntu 18.04.4 LTS, which is a result of contributions by thousands of developers who aimed to create an ideal developer environment. To work with the GPU, CUDA Toolkit 10.2 is utilized[20,21]. Python language is employed in this work, as it provides rich set of libraries for face recognition system such as OpenCV, Dlib, and TensorFlow.

5. Results and Discussions

Different images of the authenticated person are captured with various lighting conditions, angles, and poses. Figure 5 , shows the working of building a face dataset.

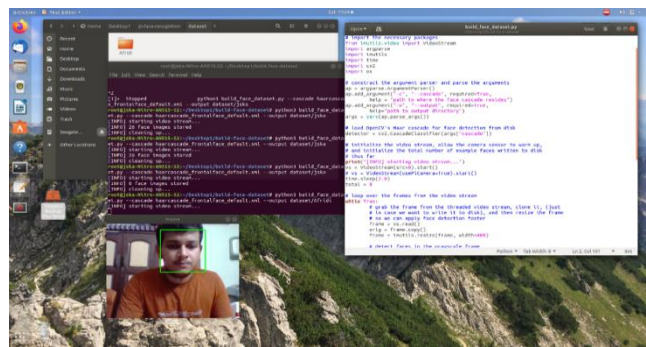


Figure 5 Building a face dataset.

Images of the authenticated individual are captured under various lighting conditions, angles, and postures. Figure 6 illustrates a sample dataset of these captured images.



Figure 6 Sample data set of authenticated person

The labeled face database is then saved with the person's name as shown in figure 7.

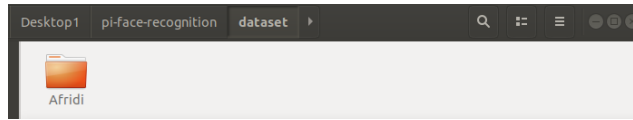


Figure 7 Dataset created and labeled

Once the face database is created, it is essential to encode the known faces. Encoding can be performed using either HOG or CNN. The encoding of the recorded dataset using the HOG method is depicted in Figure 8, and the process takes approximately 4.93 seconds to complete.

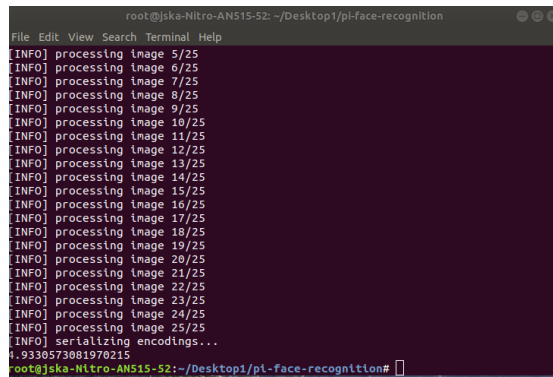


Figure 8 Encoding the data set with HOG

The figure 9 demonstrates the encoding of the recorded dataset using the CNN method. This encoding process requires approximately 91.60 seconds to complete.

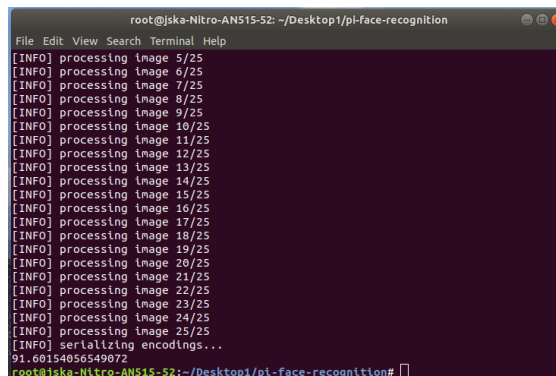


Figure 9 Encoding the data set with CNN

After encoding (training) the dataset using HOG or CNN, the faces can be recognized. HOG with CPU can be used if the number of datasets is low, whereas CNN with GPU can be used if the number of datasets is substantial. Following encoding, face recognition is performed. For analytical purposes, encoding is done using both HOG and CNN methods.

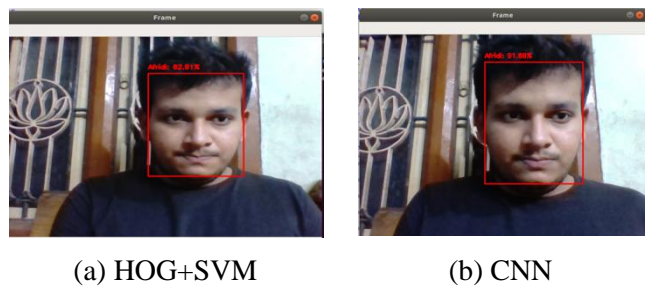


Figure 10 Face recognition in real time video

Figure 10 shows the recognized face in a real video. The recognized face is displayed in a rectangular box with its accuracy and labeled name. Figure 3a represents HOG-based face recognition on CPU with an accuracy of 62.91%. Figure 3b represents CNN-based face recognition on GPU with an accuracy of 91.69%.

The proposed face recognition system can also be programmed to detect unknown faces and label them as "Unknown." This helps in identifying unknown people on the premises and can be used to prevent unauthorized access.



Figure 11 Identification of unknown and known face in real time video

From the captured video, unknown faces are detected and checked against a recorded dataset of known individuals. If there is no match, they are labeled as "Unknown." The system can also detect multiple faces simultaneously, as shown in Figure 11. It can process multiple faces in real time and display their labeled names. Figure 11 demonstrates that the proposed system is capable of identifying multiple faces and distinguishing between known and unknown faces simultaneously in real time.

## 6. Performance Evaluation

### Performance Metrics

The performance metrics of the proposed model are Accuracy, Precision, Sensitivity and F1 score. Table 1 represents the training and test dataset and Table 2 represents performance evaluation of HoG and CNN.

Table 1 Training and Test datasets

Algorithm	Total no of images in dataset	Training to Test Ratio	No. of Training Images	No. of Test Images
CNN	2500	70:30	1750	750
	2500	80:20	2000	500
HoG+SVM	2500	70:30	1750	750
	2500	80:20	2000	500

Table 2 Performance evaluation of HoG and CNN

Algorithm	Training to Test Ratio	Accuracy (%)	Precision (%)	Sensitivity (%)	F1-Score (%)
CNN	70:30	91.253	95.461	96.596	96.027
	80:20	92.133	94.513	93.683	97.583
HoG	70:30	90.569	94.897	95.389	95.627
	80:20	92.056	93.458	92.789	95.787

### Execution Time Analysis

In this proposed work, face recognition is processed using the HOG algorithm in the CPU and the CNN algorithm in the GPU architecture.

Table 3 Execution Time comparison of HOG and CNN

No. of Input Images in dataset	HOG+SVM		CNN	
	Execution Time (s)	Encoding Size (KB)	Execution Time (s)	Encoding Size (KB)
10	1.9	10.9	36.7	11
25	4.42	22.7	90.8	23.3
100	18	92.1	361	96.5
250	44.39	231.4	901	242.5
500	87.93	463.9	2117	486.0

To analyze execution time, datasets of different sizes are fed to both HOG and CNN methods. Table 3 represents the execution time comparison between HOG and CNN algorithms with varying dataset sizes. The encoding size of HOG is relatively constant, while the encoding size of CNN increases with the number of input images. This is because HOG only needs to store the features that it has extracted from the data, while CNN needs to store the entire network architecture, as well as the weights of the network.

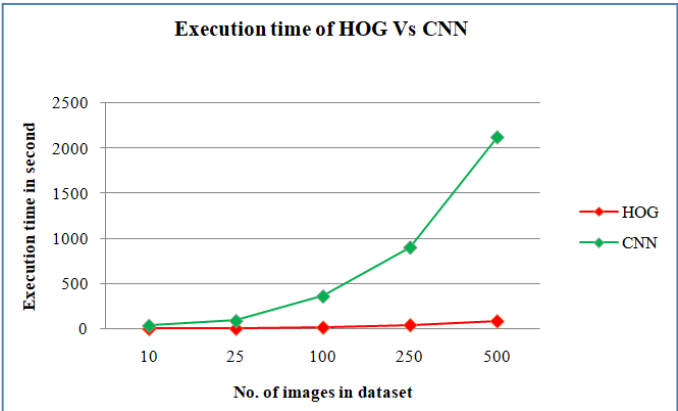


Figure 12 Comparison of HOG and CNN for different dataset size

Table 4 Comparison between CNN and HOG performance for different images







Sample image	CNN	HOG+SVM
1		
2		
3		



Figure 12 depicts a comparison of execution time between HOG and CNN for various dataset sizes. The execution time of HOG is significantly faster than the execution time of CNN for small datasets. However, the execution time of CNN starts to approach the execution time of HOG as the number of input images increases. This is because CNN is a more powerful algorithm that can learn more complex features from the data, and it therefore takes longer to process each image. In the context of face recognition systems, HOG is often used as a pre-processing step to extract features from images. These features can then be used by a more complex algorithm, such as CNN, to perform face recognition.

#### Accuracy analysis

To determine the accuracy of the given method, the detected and recognized faces are compared with existing datasets, and the percentage of matching is calculated. For HOG, the extracted features are matched with the encoded features, and the percentage of feature match is recorded.

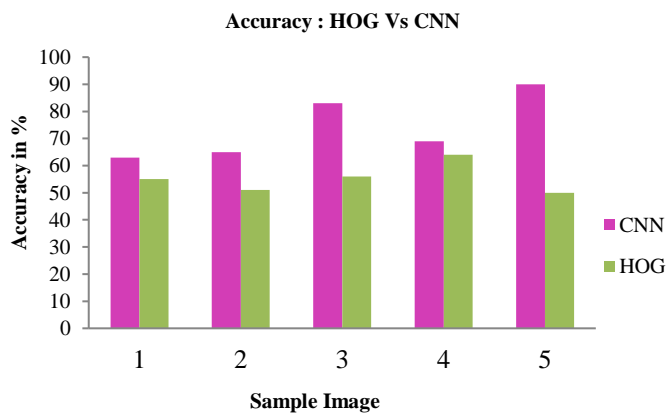


Figure 13 Performance evaluation of HOG and CNN with respect to Accuracy



In CNN, as it takes layers of the image, the values of the layers are compared with the extracted values, and the percentage of accuracy is recorded. Five sample images were considered, and the accuracy was measured for both HOG and CNN in different architectures. Table 2 presents a performance comparison between CNN and HOG for various images. Figure 13 represent the performance evaluation of face recognition through HOG and CNN algorithms, using CPU and GPU, respectively. From Table 1, it is observed that CNN takes more execution time than HOG since CNN extracts more features from the face. In Figure 5, it is observed that CNN provides higher accuracy than HOG. Based on the results, it can be concluded that CNN has higher accuracy in detecting and recognizing real-time video. The results of this table suggest that HOG is a better choice for small datasets, while CNN is a better choice for large datasets. However, the choice of algorithm also depends on the specific application. For example, if speed is critical, then HOG may be a better choice, even for large datasets.

## 7. Conclusion

This paper explored two distinct artificial intelligence approaches, HOG and CNN, for real-time video-based face recognition. The analysis involved implementing these algorithms on different hardware platforms to assess their performance. The experimental results demonstrated that CNN outperforms HOG in terms of accuracy. This is attributed to CNN's ability to efficiently handle large datasets through parallel execution on GPU architecture, resulting in improved encoding and accuracy. However, it is important to consider that GPU architecture tends to be more expensive compared to CPU architecture. For cost-sensitive image processing applications with smaller datasets, HOG remains a preferred choice. Conversely, in scenarios requiring hard real-time image processing and larger datasets, CNN proves to be more effective. This high-performance solution will be particularly useful for industrial authentication systems, ensuring secure and efficient access control. Overall, the selection between HOG and CNN depends on the specific requirements of the application, balancing factors such as cost, dataset size, and real-time processing needs.

## References

1. Viola P, Jones M. Robust real-time face detection. *Int J Comput Vis.* 2004;57(2):137-54.
2. Gupta I, Patil V, Kadam C, Dumbre S. Face detection and recognition using Raspberry Pi. In: 2016 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE); 2016 Dec 19-21; Pune, India. IEEE; 2016. p. 83-6. doi: 10.1109/WIECON-ECE.2016.8009092.
3. Wang J, Li Z. Research on face recognition based on CNN. In: 2nd International Symposium on Resource Exploration and Environmental Science; 2018 Apr 12-13; Wuhan, China. IOP Conf Ser Earth Environ Sci. 2018;170:1-5. doi: 10.1088/1755-1315/170/3/032110.
4. Ciresan D, Meier U, Schmidhuber J. Multi-column deep neural networks for image classification. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2012 Jun 16-21; Providence, RI. IEEE; 2012. p. 3642-9.
5. Wazwaz AA, Herbawi AO, Teeti MJ, Hmeed SY. Raspberry Pi and computers-based face detection and recognition system. In: 2018 4th International Conference on Computer and Technology Applications (ICCTA); 2018 May 3-5; Istanbul, Turkey. IEEE; 2018. p. 171-4.



- doi: 10.1109/CATA.2018.8398677.
6. Lawrence S, Giles CL, Tsoi AC, Back AD. Face recognition: A convolutional neural network approach. *IEEE Trans Neural Netw.* 1997;8(1):98-113.
  7. Saypadith S, Aramvith S. Real-time multiple face recognition using deep learning on embedded GPU system. In: *APSIPA Annual Summit and Conference*; 2018 Nov 12-15; Honolulu, HI. APSIPA; 2018. p. 1318-24.
  8. Kumar A, Kaur A, Kumar M. Face detection techniques: A review. *Artif Intell Rev.* 2018;50(3):331-56. doi: 10.1007/s10462-018-9650-2.
  9. Shivashankar J, Bhutekar K, Manjaramkar AK. Parallel face detection and recognition on GPU. *Int J Comput Sci Inf Technol.* 2014;5:2013-8.
  10. Bradski G, Kaehler A. *Learning OpenCV: Computer vision with the OpenCV library.* Sebastopol (CA): O'Reilly Media; 2008.
  11. NVIDIA. GeForce GTX 1050 Ti specifications [Internet]. 2017 [cited 2024 Aug 27]. Available from: <https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-1050-ti/specifications>.
  12. Canonical Ltd. Ubuntu desktop features [Internet]. 2024 [cited 2024 Aug 27]. Available from: <https://ubuntu.com/desktop/developers>.
  13. Khan M, Chakraborty S, Astya R, Khepra S. Face detection and recognition using OpenCV. In: *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*; 2019 Jan 18-19; Greater Noida, India. IEEE; 2019. p. 116-9. doi: 10.1109/ICCCIS48478.2019.8974493.
  14. Herout A, Josth R, Juranek R, Havel J, Hradis M, Zemcik P. Real-time object detection on CUDA. *J Real-Time Image Proc.* 2011;6:159-70.
  15. Prasad PS, Pathak R, Gunjan VK, Ramana Rao HV. Deep learning-based representation for face recognition. In: *Proceedings of the International Conference on Computational and Cyberphysical Engineering (ICCCE)*; 2019 Mar 14-16; Chennai, India. *Lecture Notes in Electrical Engineering*, vol. 370. Springer; 2019. p. 309-18. doi: 10.1007/978-981-13-8752-8\_33.
  16. Pei Z, Xu H, Zhang Y, Guo M, Yang YH. Face recognition via deep learning using data augmentation based on orthogonal experiments. *Electronics.* 2019;8(5):566. doi: 10.3390/electronics8050566.
  17. Ahonen T, Hadid A, Pietikainen M. Face recognition with local binary patterns. In: *Proceedings of the European Conference on Computer Vision*; 2004 May 11-14; Prague, Czech Republic. Springer; 2004. p. 469-81.
  18. Taigman Y, Yang M, Ranzato M, Wolf L. DeepFace: Closing the gap to human-level performance in face verification. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*; 2014 Jun 23-28; Columbus, OH. IEEE; 2014. p. 1701-8.
  19. Viola P, Jones M. Rapid object detection using a boosted cascade of simple features. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*; 2017 Jul 21-23; Providence, RI. IEEE; 2017. p. I-511.
  20. Ranjan R, Patel VM, Chellappa R. HyperFace: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Trans Pattern Anal Mach Intell.* 2019;41(1):121-35.
  21. Liu Z, Luo P, Wang X, Tang X. Deep learning face attributes in the wild. In: *Proceedings of the IEEE International Conference on Computer Vision*; 2015 Dec 13-16; Santiago, Chile. IEEE; 2015. p. 3730-8.
  22. Bansal A, Nanduri A, Castillo CD. Deep learning face representations for access control. In: *Proceedings of the IEEE International Joint Conference on Biometrics*; 2017 Oct 1-4; Denver, CO. IEEE; 2017. p. 1-8.
  23. Li C, Luo J, Zhang J, Xie X, Li W, Zhang L. Face recognition using deep multi-pose representations. *IEEE Trans Image Process.* 2021;30:576-88.

24. Parkhi OM, Vedaldi A, Zisserman A, Jawahar CV. Deep face recognition. In: Proceedings of the British Machine Vision Conference; 2015 Sep 7-10; Swansea, UK. BMVA Press; 2015. p. 41.1-41.12.
25. Gudla SS, Sastry NMA. A survey on face recognition methods. In: Proceedings of the International Conference on Advances in Computing and Data Sciences; 2021 Apr 9-10; Dehradun, India. Springer; 2021. p. 182-91.
26. Zhu X, Ramanan D, Fowlkes CC. Face detection, pose estimation, and landmark localization in the wild. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2012 Jun 16-21; Providence, RI. IEEE; 2012. p. 2879-86.
27. Gao S, Zhao M, Wang X, Zhang G. Face recognition based on deep learning: An overview. *Int J Comput Vis*. 2022;130(2):466-88.
28. Li H, Luo J, Li S, Zhou Z, Fu H. Recent advances in deep learning for face recognition: A comprehensive survey. *Neurocomputing*. 2022;503:214-35.
29. Shen T, Huang X, Zeng Z, Li X, Zhang G. An overview of face recognition: Techniques, datasets, and applications. *ACM Trans Multimedia Comput Commun Appl*. 2022;18(1):1-27.
30. Mishra DK, Kumar D. Face recognition system using artificial intelligence: Comparison of classifiers. In: 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC); 2023 Aug 4-6; Coimbatore, India. IEEE; 2023. p. 1527-32. doi: 10.1109/ICESC57686.2023.10193514.
31. Indra Gandhi K, Kannan G, Jawahar PK. Real-time enhanced efficient thread level parallelism scheme for performance improvement in heterogeneous edge computing. *Multidiscip Sci J*. 2024;9:2024145-2024145.