FACE IDENTIFICATION AND CATEGORIZATION WITH AND WITHOUT MASK UTILIZING MTCNN AND OPENCV FOR ACCESSING BANK LOCKER FACILITY

PROJECT REPORT

Submitted by

DEEPIKA LP (111520244011)

MOUNIKA S (111520244026)

NIKITA DEVENDRAN (111520244028)

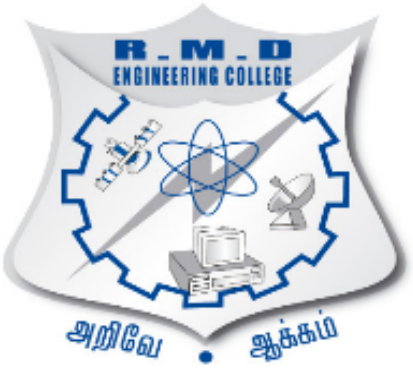
of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND BUSINESS SYSTEMS

R.M.D ENGINEERING COLLEGE, KAVARAIPETTAI.
(An Autonomous Institution)



GUMMIDIPOONDI TALUK, THIRUVALLUR - 601206.

MARCH 2024

BONAFIDE CERTIFICATE

Certified that this project report titled **"FACE IDENTIFICATION AND CATEGORIZATION WITH AND WITHOUT MASK UTILIZING MTCNN AND OPENCV FOR ACCESSING BANK LOCKER FACILITY",** is a Bonafide work of **Deepika LP (111520244011)**, **Mounika S (111520244026)**, **Nikita Devendran (111520244028)** who carried out the work under my supervision, for the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Business Systems. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion.

SIGNATURE

Dr.G.Amudha M.E, Ph.D.

HEAD OF THE DEPARTMENT

Professor,
Department of Computer Science
and Business Systems,
R.M.D. Engineering College,
R.S.M. Nagar,
Kavaraipettai - 601206.

SIGNATURE

Ms.R.Monica Lakshmi, B.E, M.E.,

SUPERVISOR

Assistant Professor,
Department of Computer Science
and Business Systems,
R.M.D. Engineering College,
R.S.M. Nagar,
Kavaraipettai - 601206.

CERTIFICATE OF EVALUATION

College Name : R.M.D ENGINEERING COLLEGE

Department : COMPUTER SCIENCE AND BUSINESS SYSTEMS

Semester : 08

Code/Course : CW8811 - PROJECT WORK

Title of Project	**Name of the Students with Register Numbers**	**Name of the Supervisor with Designation**
Face Identification and Categorization with and without mask utilizing MTCNN and OpenCV for accessing Bank Locker Facility	Deepika LP (11152024011) Mounika S (111520244026) Nikita Devendran (111520244028)	Ms.R.Monica Lakshmi, B.E, M.E., Assistant Professor Department of Computer Science and Business Systems

The report of the project work submitted by the above students in partial fulfillment for the award of Bachelor of Technology Degree in Computer Science and Business Systems of R.M.D Engineering College was confirmed to be the report work done by the above students and then evaluated.

Submitted the project during the viva voce held on .....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance, Support and kind cooperation from many, for successful completion. We wish to express our sincere thanks to all those who were involved in the completion of this project.

It is our immense pleasure to express our deep sense of gratitude to our respected chairman **Thiru R. S. Munirathinam**, our vice chairman **Thiru R. M. Kishore**, and our director **Thiru R. Jothi Naidu** for the facilities and support given by them in the college.

We are extremely thankful to our principal **Dr. N. Anbuchezhian, M.S, M.B.A, M.E, Ph.D.**, for giving us an opportunity to serve the purpose of education.

We are indebted to **Dr. G. Amudha, M.E, Ph.D.**, Professor, Head of the Department in Computer Science and Business Systems for providing the necessary guidance and constant encouragement for successful completion of this project on time.

We extend our sincere thanks and gratitude to our project guide **Ms.R.Monica Lakshmi, B.E, M.E.,** Assistant Professor in the Department of Computer Science and Business Systems, who guided us all along till the completion of our project work.

Last but not the least, we wish to thank all the teaching and non-teaching staff of the CSBS department for their help in the completion of the project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABULATIONS

ABSTRACT

This project addresses the challenge of face recognition in the presence of face masks, particularly relevant during the COVID-19 pandemic, by developing a sophisticated system tailored for bank locker access. The process begins with the application of face detection algorithms like MTCNN to identify and localize faces within a stream. Once detected, alignment techniques are employed to standardize the orientation and scale of each face, ensuring consistency for subsequent steps. Following alignment, deep learning models such as Facenet are utilized for feature extraction, transforming each face into a high-dimensional embedding that captures its unique characteristics. These embeddings serve as compact representations of faces, enabling efficient comparison and recognition. To categorize faces and determine their authenticity, classifiers like softmax classifiers are applied to the embeddings. Additionally, similarity metrics such as cosine similarity are leveraged to measure the likeness between embedded vectors, facilitating tasks like face verification and clustering. By integrating these components, the system can accurately identify individuals and grant access to bank lockers based on facial recognition, even when individuals are wearing masks. Our implementation utilizing MTCNN and FaceNet achieved high accuracy rates of approximately 98% for face recognition without masks within 100-200 epochs. However, the introduction of masks led to a slight decrease in accuracy, averaging around 95%. Overcoming the challenges posed by mask occlusion required additional training epochs, often exceeding 300, to achieve optimal results. Hence, this system surpasses Eigenfaces and Local Binary Patterns, enabling superior feature extraction and recognition accuracy, especially in handling variations such as facial expressions, poses, and masks occlusion.

CHAPTER 1
INTRODUCTION

1.1 OBJECTIVE

Face identification and categorization play pivotal roles in modern security systems, especially in environments such as accessing bank locker facilities. However, the widespread adoption of mask-wearing due to various reasons, including health concerns, has posed significant challenges to traditional face recognition methods. To address this issue, innovative approaches are being explored, leveraging technologies like the Multi-task Cascaded Convolutional Neural Network (MTCNN) and OpenCV. These technologies enable the development of systems capable of accurately identifying and categorizing individuals both with and without masks. By integrating MTCNN, which excels in detecting and aligning faces under various conditions, with OpenCV's robust image processing capabilities, it becomes feasible to overcome the obstacles posed by mask occlusion. This solution not only enhances security measures by ensuring reliable access control to bank locker facilities but also underscores the adaptability of facial recognition systems in dynamically changing environments. Moreover, by accommodating both masked and unmasked scenarios, such systems demonstrate a forward-thinking approach towards addressing contemporary challenges while maintaining the integrity and efficiency of security protocols.

1.2 ABOUT THE PROJECT

The aim of this project is to develop a face recognition system using OpenCV that can accurately detect and recognize individuals even when they are wearing masks. The project focuses on integrating mask detection with face recognition algorithms to overcome the challenges posed by masks. Utilizing face detection algorithms integrated into OpenCV, the system efficiently locates and isolates faces within images or video frames. Upon successful face detection, the system discerns whether the detected individual is wearing a mask. Through comprehensive matching algorithms, the system accurately associates the detected face with its corresponding identity within the database. By intelligently combining these components, the system offers unparalleled accuracy, and adaptability, ensuring secure and efficient access control to bank locker facilities.

1.3 SCOPE OF THE PROJECT

- To implement dynamic access control policies based on individual characteristics and contextual factors such as time of day, location, and the presence of authorized personnel.
- To train deep learning models specifically to recognize various types of masks and adaptively adjust recognition strategies accordingly.
- To integrate the face identification system with existing surveillance infrastructure to enable real-time monitoring of access events and suspicious activities.
- To integrate multiple biometric modalities, such as facial recognition, iris recognition, fingerprint scanning, voice recognition, to enhance the system's robustness and reliability.

1.4 APPLICATIONS OF THE PROJECT

The first domain is enhanced security measures. By integrating facial recognition technology, the bank locker facility can strengthen its security measures. Access to lockers can be restricted solely to authorized individuals, minimizing the risk of unauthorized entry or theft.Incorporating MTCNN and OpenCV enables the system to identify individuals both with and without masks. This adaptability ensures that customers can access their lockers securely, even in situations where mask-wearing is mandated, such as during health crises.

The second domain is efficient access control. Traditional methods of access control, such as keys or PINs, can be replaced with biometric authentication, streamlining the process for users. This ensures quick and convenient access to their belongings while maintaining high-security standards.The system can maintain detailed logs of locker access, including timestamps and user identities. This facilitates audit trails and monitoring, enabling bank staff to track locker usage and investigate any suspicious activities effectively.

The last domain is personalized services. Recognizing customers upon entry allows the bank to offer personalized services. For example, staff members can greet customers by name, anticipate their needs, and provide tailored assistance, enhancing the overall customer experience.Face identification can be integrated into the transaction verification process, adding an extra layer of security. Users may need to confirm their identity before accessing their lockers or conducting specific transactions, reducing the risk of fraud or identity theft.

1.5 EXISTING SYSTEM

In the existing face recognition systems, the primary focus is on recognizing individuals based on their facial features. However, these systems face limitations when it comes to accurate recognition when individuals are wearing masks. Traditional face recognition algorithms heavily rely on facial features, such as the nose, mouth, and chin, which are partially or completely covered by masks. As a result, the performance of existing face recognition systems significantly deteriorates in scenarios where mask usage is prevalent.

To overcome this limitation, some approaches have been proposed, such as using face alignment techniques to estimate the position of facial landmarks under the mask or employing face reconstruction methods to recover the occluded facial regions. However, these techniques require additional computational resources and may not yield satisfactory results.

1.5.1 DRAWBACKS OF EXISTING SYSTEM

- Reduced Accuracy : Traditional face recognition systems experience reduced accuracy due to factors such as variations in lighting conditions, facial expressions, pose angles,and image quality. Inaccurate recognition can lead to authentication failures denying access to authorized users or granting access to unauthorized individuals.
- Increased False Matches: False matches occur when the face recognition system incorrectly identifies an individual as someone else. False matches erode user trust and confidence in the system's effectiveness, leading to reluctance in adoption and usage.
- Mask Detection Limitations: Masks obstruct crucial facial features, reducing the system's ability to extract distinctive biometric information and leading to recognition failures or false rejections. This limitation undermines the effectiveness of face recognition in scenarios where mask-wearing is prevalent.
- Computational Complexity: Traditional face recognition algorithms require significant computational resources for processing large datasets, performing feature extraction, and conducting matching operations. As the complexity of the algorithm increases, so does the computational demand, resulting in longer processing times, higher energy consumption, and scalability issues.
- Privacy Concerns: Users may be apprehensive about the potential misuse, unauthorized access, or data breaches that could compromise their personal information. Addressing privacy concerns is essential for ensuring the responsible deployment.

1.6 PROPOSED SYSTEM

The proposed system leveraging OpenCV is designed to address the challenge of accurately identifying individuals, particularly in scenarios where mask-wearing is prevalent. The system begins by employing face detection algorithms embedded within OpenCV to locate and isolate faces within images or video frames.

Upon detecting a face, the system transitions to a mask detection algorithm, which analyzes the facial region to determine the presence of a mask. This step is crucial for assessing. It assesses the suitability of the subsequent face recognition process, as the accuracy of traditional facial recognition algorithms can be compromised when individuals wear masks.

If the mask detection algorithm confirms that no mask is present, the system proceeds with conventional techniques, such as Eigenfaces, Fisherfaces, or LBPH. These algorithms extract distinctive facial features from the detected face and compare them against a pre-existing database of known faces.

1.6.1 ADVANTAGES OF PROPOSED SYSTEM

- User Convenience : With face recognition technology, users can quickly and effortlessly gain entry to secure areas or perform transactions, enhancing overall convenience and user satisfaction.
- Public Health Compliance : By incorporating mask detection capabilities, the system promotes adherence to public health guidelines and regulations regarding mask-wearing in public spaces.
- Contactless Interaction : It eliminates the need for physical contact with access control devices, reducing the risk of germ transmission and enhancing hygiene standards. Contactless interaction also offers convenience for users.
- Enhanced Security : It provides a high level of security by accurately identifying individuals based on unique facial features. It reduces the risk of unauthorized access or identity fraud.
- Improved Accuracy : By verifying both facial identity and mask compliance, the system minimizes false positives and enhances recognition accuracy, leading to more reliable access control.
- Robustness and Flexibility : With the ability to handle varying levels of mask coverage and facial expressions, the system demonstrates resilience and versatility.

CHAPTER 2

LITERATURE SURVEY

2.1 REVIEW OF LITERATURE SURVEY

- **SURVEY #1:**

  Title : Human Face Mask Identification using Deep Learning with OpenCV Techniques.

  Authors : Rahul Baghel, Pallavi Pahadiya, Upendra Singh.

  Publication : 7th International Conference on Communication and Electronics Systems (ICCES), 2022, pp. 1051-1057.

  According to the WHO, COVID 19 virus is spread by respiratory droplets and personal contact. To prevent the transmission of this virus, the use of masks and social isolation is recommended. To comply with regulatory requirements, a mask recognition system capable of recognizing any kind of mask as well as masks in a variety of configurations inside video streams has been developed. To detect masks, a deep learning approach and the Python TensorFlow, Keras, and PyTorch packages are utilized.

- **SURVEY #2:**

  Title : Face Mask Detection Using OpenCV and Machine Learning.

  Authors : Amjad Ali Khan, Shankar Yadav, Lalit Kumar.

  Publication : 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 2022, pp. 80-85.

  Wearing a protective face mask has become a new trend and is now considered normal. Therefore, we are developing an AI system to recognize whether a person is wearing a mask or not. This system will help us prevent the spread of the virus and protect the environment. The tools required for this project include Jupyter Notebook and the installation of numpy, OpenCV, TensorFlow, and other learning tools.

- **SURVEY #3:**

  Title : Automatic Face Mask Detection System in Public Transportation System in Smart Cities Using IoT and Deep Learning.

  Authors : Tamilarasan Ananth Kumar, Rajendrane Rajmohan, Muthu Pavithra.

Publication : Expert Systems with Applications,Volume 36, Issue 2, Part 2.

With the rise in population, there is a greater need for efficient city management for reducing the impact of COVID-19. For smart cities to prosper, it should be expanded with AI. This article presents an IoT-based face mask detection system in public transportation. This system collects real-time data. The experiments showed that the model can detect faces and masks with low inference time and memory, meeting the IoT limited resources.

- **SURVEY #4:**

Title : Facemask Detection with Face Recognition and Alert System using MobileNetV2.
Authors : Gopinath Pranav Bhargav, Kancharla Shridhar Reddy, Alekhya Viswanath.
Publication : 2nd International Conference on Intelligent and Cloud Computing (ICICC), 2022, vol. 286, pp. 77-87.

Wearing masks is an effective means of prevention of Covid-19. We built a web application that reminds people to wear masks constantly with the help of an integrated facemask detection and face-recognition system. The proposed system detects whether the person is wearing a mask or not and recognizes the face of the person. It also alerts that specific violator to wear a mask through an auto-generated email to his personal email id.

- **SURVEY #5:**

Title : Social Distance Monitoring and Face Mask Detection Using Deep Learning.
Authors : K. Yagna Sai Surya, T. Geetha Rani, B. K. Tripathy.
Publication : Computational Intelligence in Data Mining, 2022, pp.461-476.

Using Python along with deep learning frameworks, it's possible to develop a robust system for capturing people and monitoring social distancing. By employing deep learning models, the system can accurately detect whether individuals are wearing masks, ensuring compliance with safety regulations. Leveraging computer vision techniques, the system can continuously observe individuals in real-time, regardless of the time or day, enabling inspectors to monitor social distancing measures efficiently. This technology proves invaluable in public places, commercial centers, and various locations where maintaining safe distances is crucial for preventing the spread of diseases.

2.2 SUMMARY

S.NO	AUTHORS	TITLE	METHOD	ADVANTAGES	DISADVANTAGES
1.	Rahul Baghel, Pallavi Pahadiya, Upendra Singh.	Human Face Identification using Deep Learning with OpenCV.	Deep learning approach to develop a mask recognition system capable of detecting various mask configurations within video streams.	High accuracy in detecting masks of different types.	Dependency on computational resources due to the intensive nature of deep learning models.
2.	Amjad Ali, Shankar Yadav, Lalit Kumar.	Face Mask Detection using OpenCV and Machine Learning.	The project utilizes machine learning, deep learning, and neural networks to develop an AI system capable of detecting whether individuals are wearing masks.	Enhances public health safety by automating the identification of mask-wearing individuals.	May encounter challenges with accuracy and generalization.
3.	Ananth Kumar, Rajendrane Rajmohan, Muthu Pavithra.	Automatic Mask Detection in Public Transportation in Smart Cities.	The IoT-based face mask detection system utilizes real-time data collection to detect faces and masks in public transportation with low inference time and memory usage.	Low resource usage enables scalability in IoT deployments.	Privacy concerns may arise due to constant surveillance.
4.	Gopinath Pranav, Kancharla Shridhar Reddy, Alekhya Viswanath.	An Integrated Face Detection and Alert System using MobileNetV2.	It utilizes integrated facemask detection and face-recognition systems to monitor real-time video feeds, detect mask-wearing compliance, recognize faces, and issue alerts via email.	Provides real-time monitoring and enforcement of mask-wearing.	Reliance on technology may lead to false positives or missed detections.
5.	Yagna Sai Surya, Geetha Rani, B.K. Tripathy.	Social Distance Monitoring and Face Mask Detection using Deep Learning.	The methodology involves utilizing Python with Deep Learning frameworks alongside computer vision libraries to detect, monitor and recognize people wearing masks.	Provides automated monitoring of social distancing and mask usage.	Requires substantial computational resources and ongoing maintenance.

Table. 2.2 Summary

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 MODULE SPECIFICATIONS

Module Specification typically refers to the specific hardware, software, and other technical specifications that are necessary for each individual component of a module to function properly.

3.1.1 MODULE 1 - DATA COLLECTION

Data collection uses OpenCV and a webcam, which involves capturing live video feed from the webcam and processing it using OpenCV libraries in Python. The process typically starts by initializing the webcam, configuring parameters such as resolution and frame rate, and then continuously grabbing frames from the video stream. These frames can then be processed for various computer vision tasks such as object detection, tracking, or facial recognition. OpenCV provides a wide range of functions and algorithms to manipulate and analyze the captured data, enabling real-time or offline analysis depending on the application. Additionally, OpenCV offers tools for storing the collected data in various formats for further analysis or training of machine learning models. Overall, utilizing OpenCV with a webcam facilitates seamless data collection for a multitude of computer vision applications.

3.1.2 MODULE 2 - TRAINING AND CALIBRATION

This module features two stages, namely image augmentation done using OpeCV and embedding using Facenet. Data augmentation with OpenCV involves applying various transformations to images such as rotation, translation, flipping, and scaling to expand the training dataset and improve the robustness of deep learning models. Meanwhile, face embedding using FaceNet extracts high-dimensional feature vectors representing facial characteristics, enabling efficient face recognition and clustering. By combining OpenCV's data augmentation techniques with FaceNet's face embedding, one can enhance the performance and generalization of facial recognition systems by augmenting the dataset with diverse facial variations and extracting discriminative features for accurate identification.

3.1.3 MODULE 3 - FACE DETECTION

Face detection is done using MTCNN (Multi-Task Cascaded Convolutional Neural Network). It is a deep learning model designed for robust and efficient face detection in

images. It consists of three stages: Firstly, a regional proposal network identifies potential face regions. Secondly, a series of convolutional networks refine these proposals, accurately localizing facial features such as eyes, nose, and mouth. Finally, non-maximum suppression is applied to filter out redundant detections, providing the final bounding boxes around detected faces. MTCNN is known for its high accuracy and real-time performance, making it a popular choice for various face detection applications.

3.1.4 MODULE 4 - FACE IDENTIFICATION

Face identification is done using FaceNet which involves utilizing a deep learning architecture to extract facial features, known as embeddings, from images. These embeddings represent unique characteristics of each face in a high-dimensional space. Through training on large datasets, FaceNet learns to map similar faces closer together and dissimilar faces farther apart in this space. During identification, new faces are compared to the learned embeddings, and the closest match is determined based on the distance metric, enabling accurate and efficient recognition even across variations in lighting, pose, and facial expressions.

3.2 HARDWARE COMPATIBILITY REQUIREMENTS

The hardware compatibility requirements include:

- Core i3 processor and above.
- 4 GB of RAM.
- 80 GB of hard disk capacity.
- 15 inch color monitor.
- Full HD webcam.
- 52x CD-ROM drive.

3.3 SOFTWARE COMPATIBILITY REQUIREMENTS

The software compatibility requirements include:

- Anaconda Distributor.
- Python - 3.7 or later versions.
- Text editor such as Notepad, or MS Word.
- A 64-bit operating system with a x64-based processor.
- Windows 7 or later versions.
- Visual Studio Code.

CHAPTER 4

SYSTEM DESIGN

4.1 ARCHITECTURE DIAGRAM

Architecture diagram is a visual representation of software system components. As software is inherently abstract, architecture diagrams visually illustrate the various data movements and the environment around it.

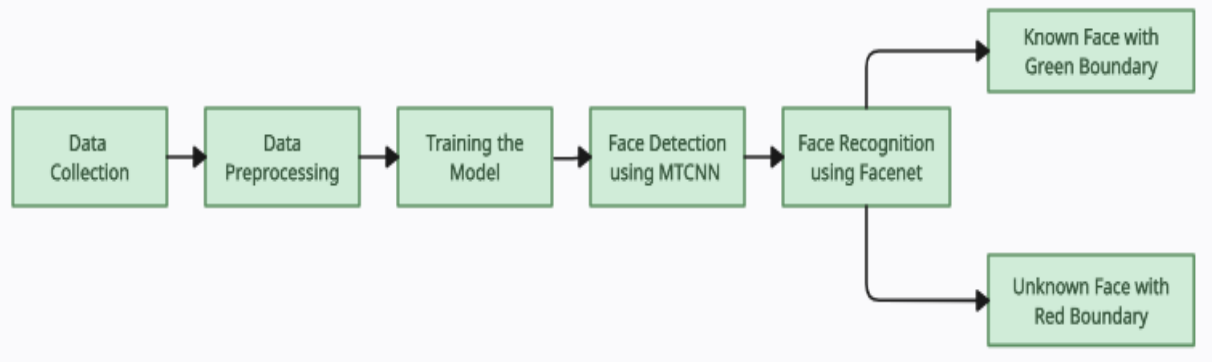


Fig 4.1 Architecture Diagram

Data collection involves gathering a diverse dataset of facial images of the respective user, encompassing various poses, expressions, and lighting conditions from the real live stream of the webcam.

The data preprocessing module then standardizes and enhances the collected images, performing tasks such as resizing, normalization, and augmentation to improve model generalization.

The training phase involves feeding the preprocessed data into the recognition model, typically a deep learning architecture like Facenet, which learns to extract discriminative features from faces and generate embeddings.

Face detection is performed using MTCNN, which accurately localizes and extracts facial regions from images or video frames. These detected faces are then aligned and fed into the Facenet model for embedding extraction.

Finally, the trained model is deployed for real-time face recognition tasks, where embedded vectors are compared using techniques like cosine similarity to identify or authenticate individuals.

4.2 INTERACTION DIAGRAM

The interaction diagram portrays the interactions between distinct entities present in the model. A set of messages that are interchanged between the entities to achieve certain specified tasks in the system is termed as interaction.

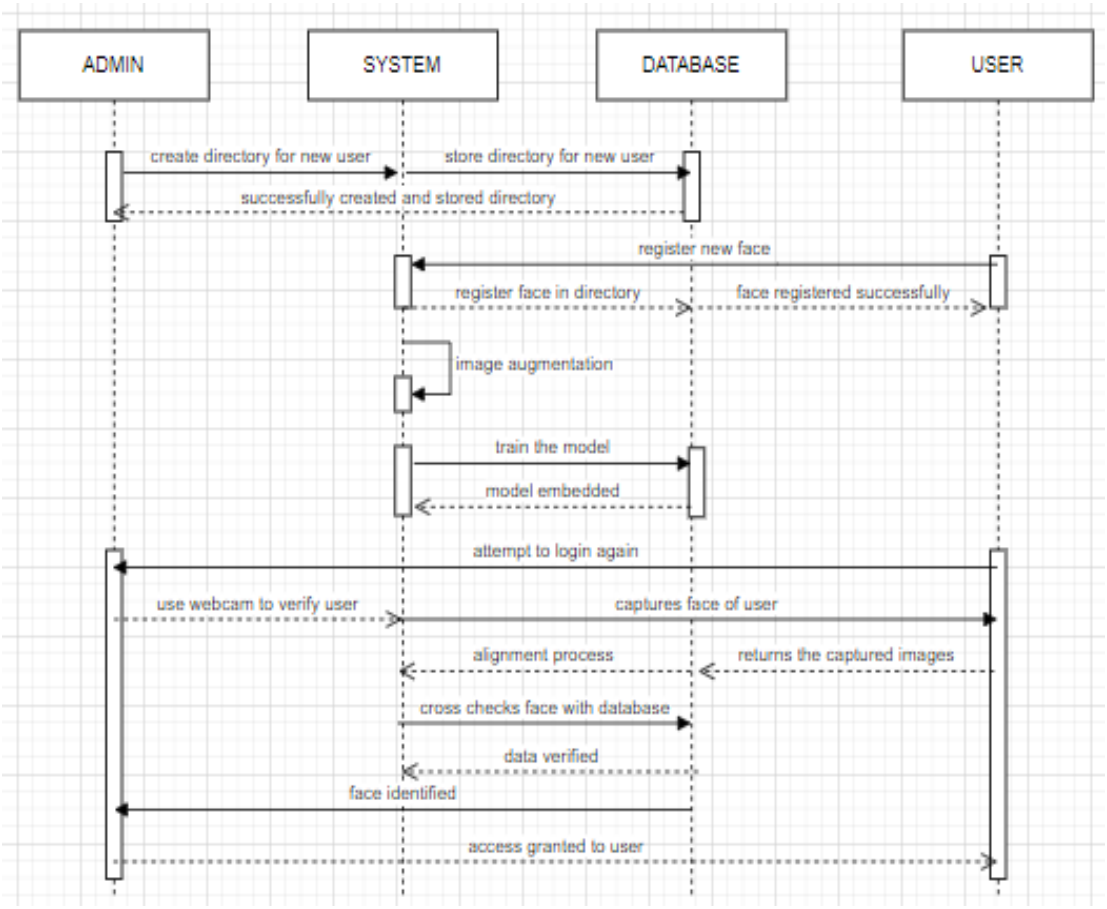


Fig 4.2 Interaction Diagram

The user module initiates the interaction by requesting access or authentication through the system. Upon receiving the user's request, the "system" module coordinates the workflow by first forwarding the request to the "admin" module for authorization. Subsequently, the system triggers the "database" module to retrieve relevant facial data associated with the user from the database. Once the facial data is retrieved, the system invokes the necessary facial recognition algorithms, possibly including modules for face detection, alignment, feature extraction, and classification. These modules collectively analyze the user's input, comparing it against the stored facial data to determine the user's identity. Finally, based on the outcome of the recognition process, the system provides feedback to the user, granting access if the recognition is successful or denying it otherwise.

4.3 CLASS DIAGRAM

Class diagram is a static diagram. Class diagram describes the attributes and operations of a class and also the interfaces, associations, collaborations, constraints imposed on the system.

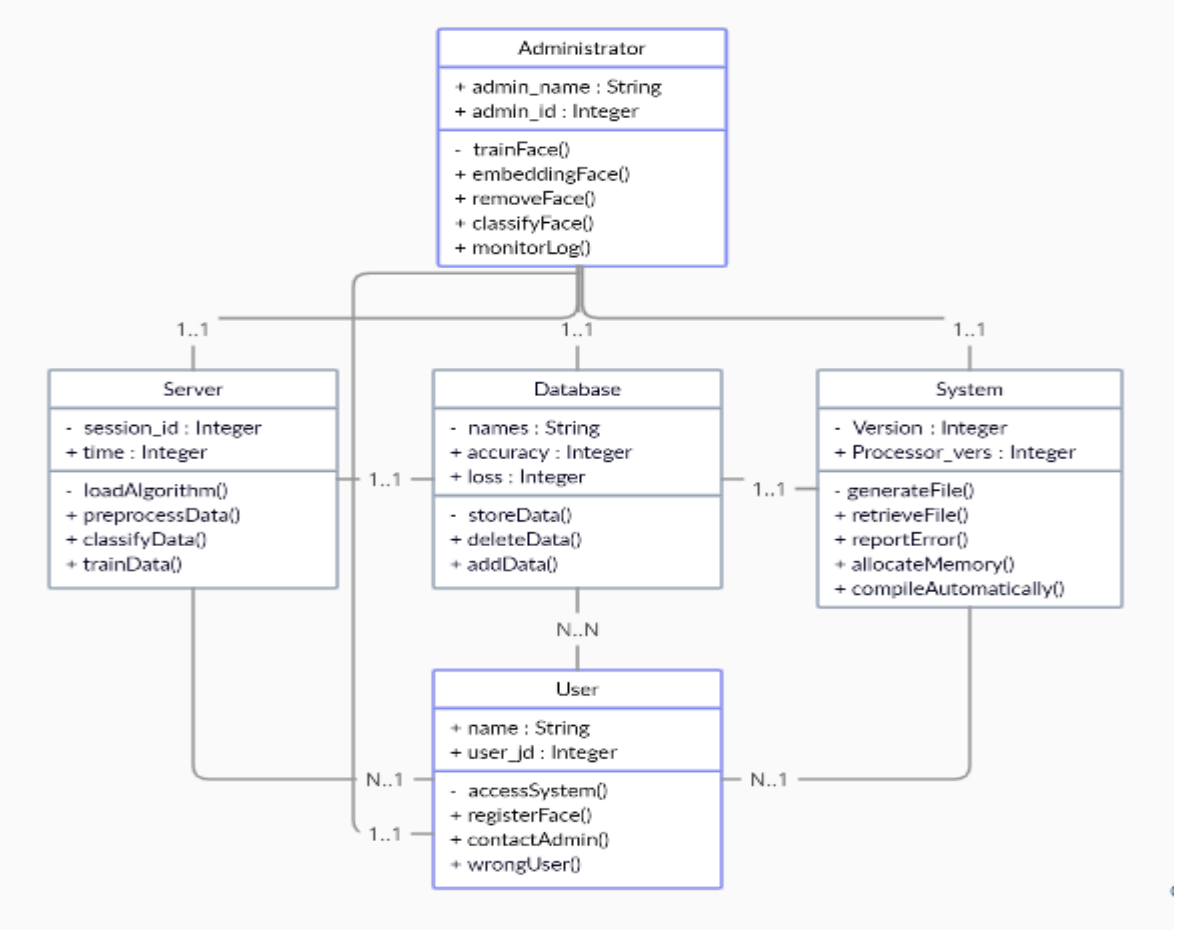


Fig 4.3 Class Diagram

The "Admin" class represents administrative users who oversee the system's operation, including managing user accounts and system settings. The "User" class encapsulates information about individual users, such as their identity and associated facial data. The "System" class serves as the core component orchestrating the functionalities of the facial recognition system, including face detection, feature extraction, and authentication. The "Database" class manages the storage and retrieval of user data, including facial templates and associated metadata. The "Server" class handles communication between the system components, facilitating data exchange and system operation across distributed environments.

4.4 PACKAGE DIAGRAM

A package diagram is used to illustrate the dependencies between different components or modules in a system. It's useful for visualizing the high-level structure of a software system. This hierarchical representation facilitates understanding the system's architecture and the interactions between its major components.

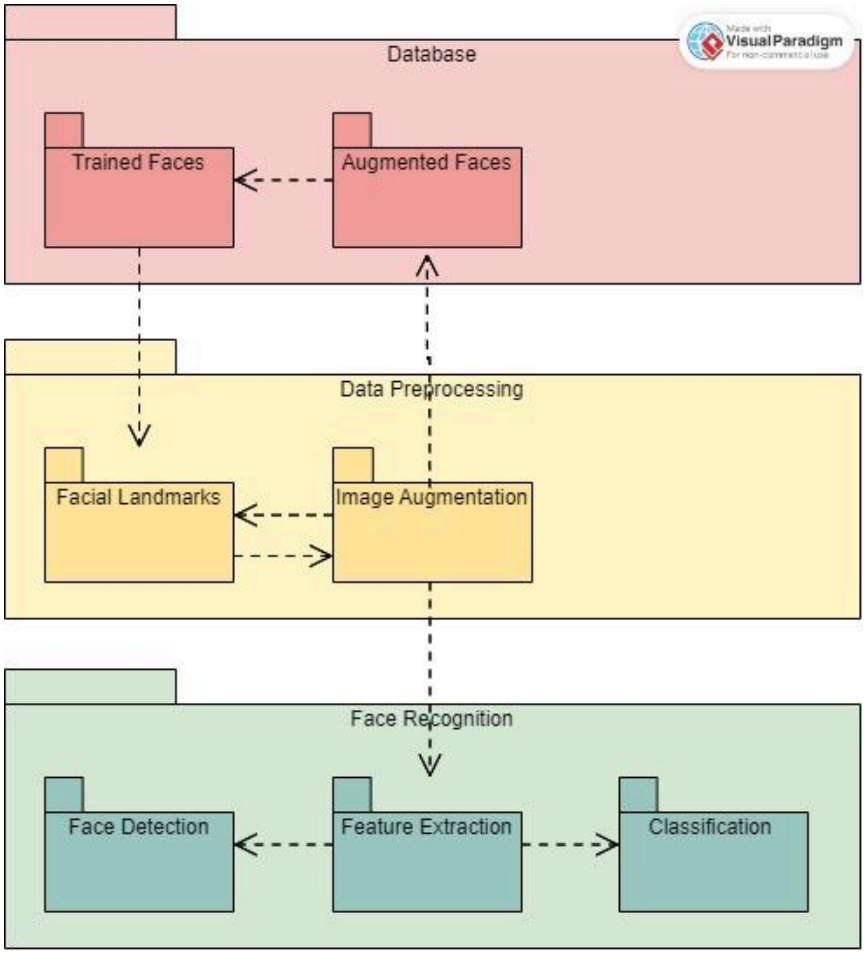


Fig 4.4 Package Diagram

The diagram includes packages such as "Database," responsible for storing and managing facial data, "Data Processing," which handles the preprocessing and manipulation of raw data to prepare it for recognition algorithms, and "Face Recognition," containing the core algorithms and logic for identifying and verifying individuals based on their facial features. The "Database" package contains sub-packages for data storage, retrieval, and management functionalities. The "Data Processing" package involves sub-packages for image preprocessing, feature extraction, and data normalization.The "Face Recognition" package encompasses sub-packages for different recognition algorithms and deep learning-based approaches.

4.5 USE CASE DIAGRAM

A use case diagram is a graphical representation used in software engineering to depict the functionalities of a system and how users interact with it. It illustrates the relationship between users (actors) and the various use cases (functional requirements) of a system.

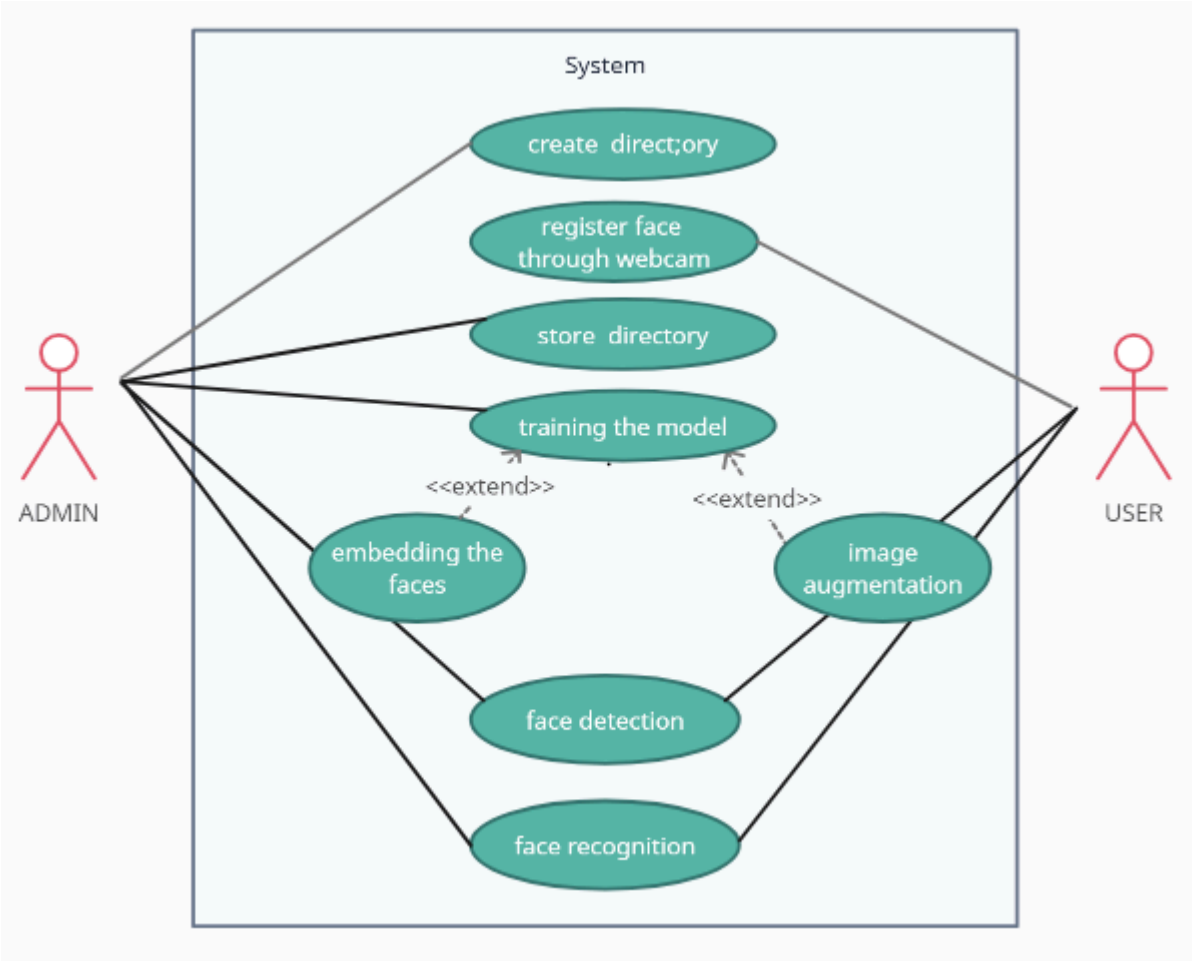


Fig 4.5 Use Case Diagram

The use case diagram for a face recognition system involves two primary actors: the Admin and the User. The Admin is responsible for managing system functionalities such as user registration, access control, and database maintenance. Their actions include tasks like adding new users, removing users, modifying user permissions, and configuring system settings. On the other hand, the User interacts with the system primarily for authentication purposes. Their actions typically include tasks like logging into the system, requesting access to restricted areas or resources, and updating their facial recognition profile. The system itself facilitates the recognition of users' faces, comparing them against stored profiles in the database, and granting appropriate access privileges based on successful matches.

4.6 ACTIVITY DIAGRAM

An activity diagram visually represents the flow of actions or activities in a system or process. It's particularly useful for modeling workflows, business processes, or the flow of control within a software system.

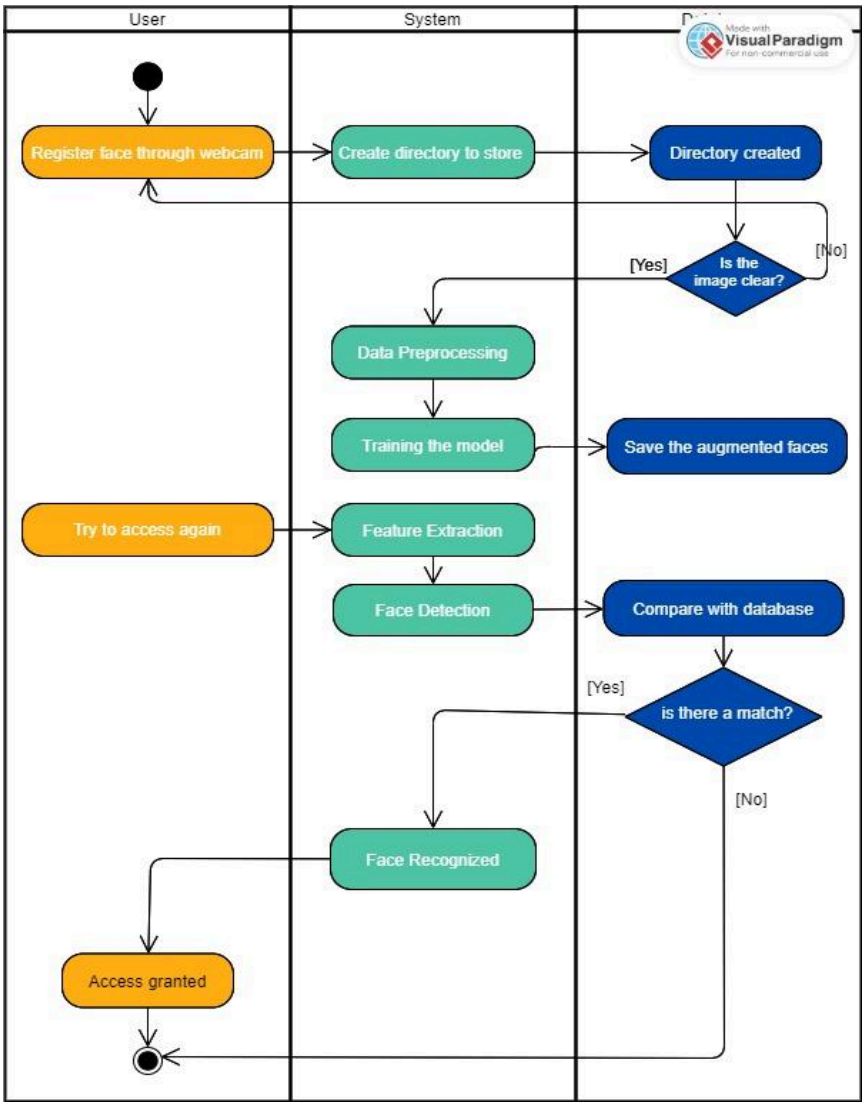


Fig 4.6 Activity Diagram

Initially, the system would be activated, followed by the initialization of the camera or input device to capture images. The captured image undergoes preprocessing steps such as feature extraction to enhance its suitability for recognition algorithms. Subsequently, the system executes the recognition algorithm to match the extracted features with stored templates or databases, determining the identity of the recognized face. Finally, the system would conclude by providing the identified individual's information and grant access.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 OVERVIEW

Face recognition technology has emerged as a powerful tool for various applications, particularly in the realm of authentication and identity verification services. Leveraging Python and its support libraries, developers can harness the capabilities of face recognition algorithms to accurately match human faces from video frames against a database of known faces. The process begins with the use of face detection algorithms like MTCNN to detect and localize faces within the stream. Once the faces are detected, an alignment process is employed to normalize the orientation and scale of each detected face, ensuring consistent input for subsequent steps.

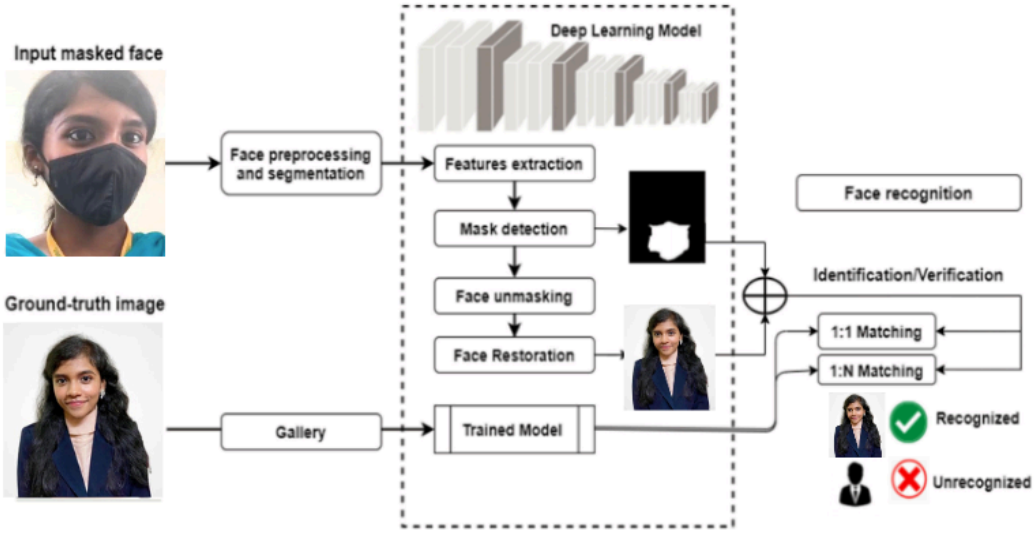


Fig 5.1 Overview of facial recognition system

Following face detection and alignment, the next step involves feature extraction using a deep learning model such as Facenet. This model transforms each aligned face into a high-dimensional vector representation called an embedding, which captures essential features and characteristics of the face. These embeddings serve as compact and discriminative representations of faces, enabling efficient comparison and recognition. Finally, for tasks such as classification or verification, a classifier, often implemented as a softmax classifier, is applied to the embedded vectors to categorize or authenticate faces. Additionally, similarity metrics like cosine similarity can be utilized to measure the similarity between embedded vectors, enabling tasks such as face verification or clustering.

5.2 TECH STACK

Tech stack refers to the combination of technologies, tools, frameworks, programming languages, and software products used to build and operate a software application or a system. Below is the tech stack used for this project :

5.2.1 ANACONDA DISTRIBUTION

The Anaconda distribution, renowned for its comprehensive suite of tools for data science and scientific computing, offers a robust ecosystem right out of the box. With over 250 packages pre-installed, users gain immediate access to a wide array of essential libraries and tools for various tasks. Beyond these initial offerings, Anaconda facilitates the installation of over 7,500 additional open-source packages, accessible from both PyPI and the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

One of the standout features of Anaconda is its user-friendly interface, Anaconda Navigator. Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. By providing a graphical alternative to the command-line interface (CLI), Anaconda Navigator caters to users who prefer visual interaction over text-based commands, enhancing accessibility and ease of use.

The Anaconda Prompt allows users to interact with their Python or R environments using command-line commands. From creating and managing virtual environments to installing and updating packages, running scripts, and executing various data analysis tasks, the Anaconda Prompt serves as a versatile tool for users who prefer or require command-line functionality. With its rich set of features, Anaconda empowers users to seamlessly transition between graphical and command-line interfaces, ensuring a smooth and efficient workflow tailored to individual preferences and requirements.

5.2.2 VISUAL STUDIO

Visual Studio is an integrated development environment (IDE) developed by Microsoft that provides a comprehensive set of tools and features for software development across various platforms and programming languages. Visual Studio supports a wide range of

programming languages including C#, C++, Python, JavaScript, and more, making it versatile for different types of projects.

With its extensive collection of built-in tools and extensions from the Visual Studio Marketplace, developers can streamline their workflows, collaborate efficiently, and build high-quality software solutions.

5.2.3 OPENCV

OpenCV is an open-source software library for computer vision and machine learning. The library has more than 2500 optimized algorithms, including an extensive collection of computer vision and machine learning algorithms, both classic and state-of-the-art.

Using OpenCV it becomes easy to do complex tasks such as identify and recognise faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D object models, generate 3D point clouds from stereo cameras, stitch images together to generate an entire scene with a high resolution image and many more.

There are many ways in which you can install OpenCV on your computer. We have used Anaconda. After successfully installing Anaconda, just go to the Anaconda prompt and use the command below to install OpenCV:

conda install -c conda-forge OpenCV

These are some of the basic operations that we can do on the images once we have successfully read them.

- Access pixel values and modify them.
- Access image properties.
- Set a Region of Interest (ROI).
- Split and merge image channels.

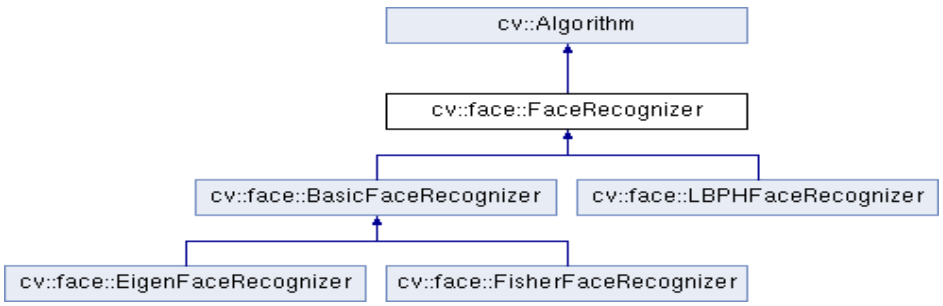


Fig 5.2.3 OpenCV

5.2.3.1 HAAR CASCADES

Haar cascades in OpenCV refer to a machine learning-based object detection technique used to identify objects or patterns within images. The process involves training a cascade classifier with positive and negative examples of the target object to learn its distinctive features. The classifier then uses a cascade of simple classifiers arranged in a hierarchy to efficiently identify regions in an image that are likely to contain the object.

In OpenCV, the library provides pre-trained Haar cascade classifiers for various objects, including faces, eyes, and smiles, making it easy to integrate object detection capabilities into computer vision applications.

Fig 5.2.3.1.1 A general representation of training a Haar classifier

The algorithm can be explained in four stages:

- **Calculating Haar Features**

  A Haar feature is essentially calculations that are performed on adjacent rectangular regions at a specific location in a detection window. The calculation involves summing the pixel intensities in each region and calculating the differences between the sums. These features can be difficult to determine for a large image.

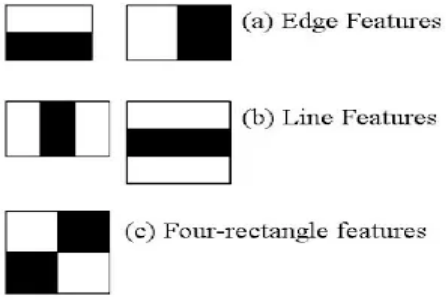


Fig 5.2.3.1.2 Types of Haar features

- **Creating Integral Images**

Integral images essentially speed up the calculation of these Haar features. Instead of computing at every pixel, it instead creates sub-rectangles and creates array references for each of those sub-rectangles. These are then used to compute the Haar features.

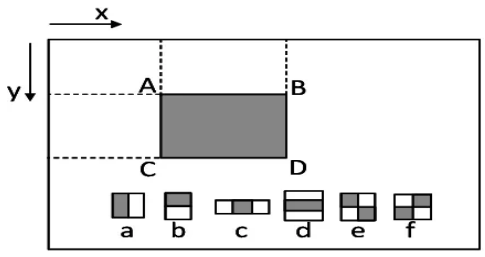


Fig 5.2.3.1.3 Integral Image Process

- **Adaboost Training**

Adaboost essentially chooses the best features and trains the classifiers to use them. It uses a combination of "weak classifiers" to create a "strong classifier" that the algorithm can use to detect objects.Weak learners are created by moving a window over the input image, and computing Haar features for each subsection of the image. This difference is compared to a learned threshold that separates non-objects from objects. Because these are "weak classifiers," a large number of Haar features is needed for accuracy to form a strong classifier.

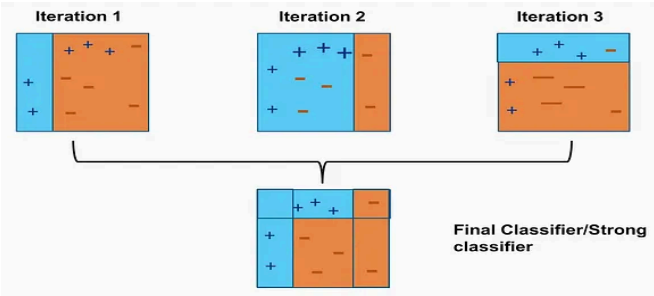


Fig 5.2.3.1.4 Representation of a boosting algorithm

- **Implementing Cascading Classifiers**

The last step combines these weak learners into a strong learner. It is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows a highly accurate classifier from the mean prediction. Based on this, the classifier decides to indicate an object was found or moves to the next region.

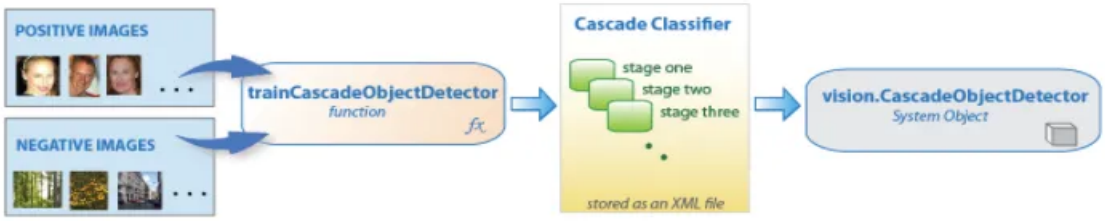


Fig 5.2.3.1.5 Flowchart of Cascade Classifiers

5.2.3.2 LOCAL BINARY PATTERN

Local Binary Pattern (LBP) is a grayscale invariant texture descriptor measure for classification. A binary code is generated at each pixel by thresholding its neighborhood pixels to either 0 or 1 based on value of the centre pixel. It compares the intensity values of a central pixel with its surrounding pixels. Here's how the LBP algorithm typically works:

- **Local Neighborhood**

  Select a neighborhood around each pixel in the image. This neighborhood is usually defined by a specific radius and the number of pixels around the central pixel.

- **Thresholding**

  Compare the intensity value of the central pixel with the intensity values of its neighboring pixels. If the intensity value of a neighboring pixel is greater than or equal to that of the central pixel, assign it a value of 1; otherwise, assign it a value of 0.

- **Binary Encoding**

  After thresholding, you get a binary pattern (a string of 0s and 1s) representing the relationship between the central pixel and its neighbors.

- **Conversion to Decimal**

  Interpret the binary pattern as a binary number and then convert it to decimal. Select a neighborhood around each pixel in the image.

- **Histogram Representation**

  Calculate histograms of these decimal patterns over the entire image. These histograms capture the distribution of different texture patterns in the image.

- **Feature Vector**

Concatenate all the histogram bins into a single feature vector. This feature vector represents the texture characteristics of the image.

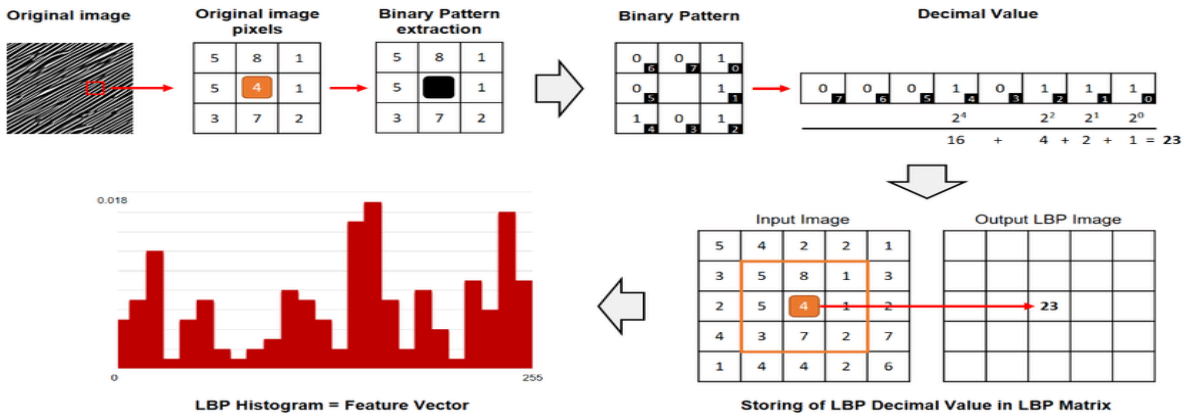


Fig 5.2.3.2 Local Binary Pattern

5.2.4 DLIB

Dlib is a powerful library for machine learning and computer vision in Python. It is used to detect faces in images, video streams. This library includes a pre-trained model for face detection that can be used out of the box, making it easy to add the face detection functionality.

The Dlib library includes a function called "get_frontal_face_detector ()," which returns a pre-trained object detector that is specifically designed for detecting faces in images. This detector can be used to locate faces in an image and return the coordinates of the bounding boxes for each face.

It is basically a landmark's facial detector with pre-trained models, the dlib is used to estimate the location of 68 coordinates (x, y) that map the facial points on a person's face.

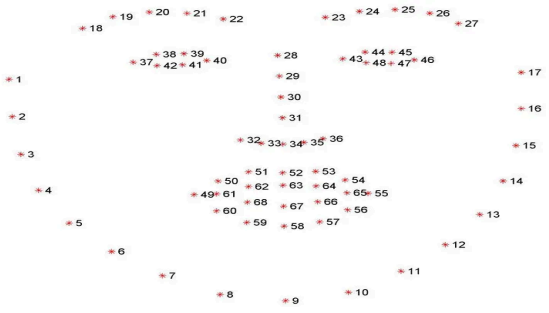


Fig 5.2.4 DLIB facial landmark

5.2.5 MTCNN

MTCNN stands for Multi-task Cascaded Convolutional Neural Network. It is a deep learning-based algorithm used for face detection and facial feature localization. MTCNN was designed to efficiently detect faces at different scales in images while also providing bounding boxes for faces and key facial landmarks such as eyes, nose, and mouth.

The command for installing MTCNN is as follows :

pip install mtcnn (for pip)

conda install -c conda-forge mtcnn (for conda)

The concept of MTCNN can be explained using three networks: the P-Net, R-Net, and O-Net.

In the first stage, it creates multiple frames which scans through the entire image starting from the top left corner and eventually progressing towards the bottom right corner. It efficiently proposes candidate regions likely to contain faces. The information retrieval process is called P-Net(Proposal Net) which is a fully connected CNN.

In the second stage, all the information from P-Net is used as an input for the next layer of CNN called R-Net (Refinement Network), a fully connected, complex CNN which rejects a majority of the frames which do not contain faces. The R-Net refines the bounding boxes proposed by the P-Net and rejects false positives. It applies more complex features and a higher resolution to accurately localize faces.

In the third and final stage, a more powerful and complex CNN, known as O-Net (Output Network). The O-Net further refines the bounding boxes and performs facial landmark localization. It provides accurate facial feature positions, including eyes, nose, and mouth.

To write a code for face detection using MTCNN in Python, you would first need to install the necessary libraries, such as TensorFlow, Keras, and MTCNN. Then, you would need to load the pre-trained weights for the P-Net, R-Net, and O-Net. Next, you need to create an instance of the MTCNN class and pass in the loaded weights. Finally, you would use the detect faces() method of the MTCNN class to detect faces in an image, which would return the bounding boxes and facial landmarks for each face detected.

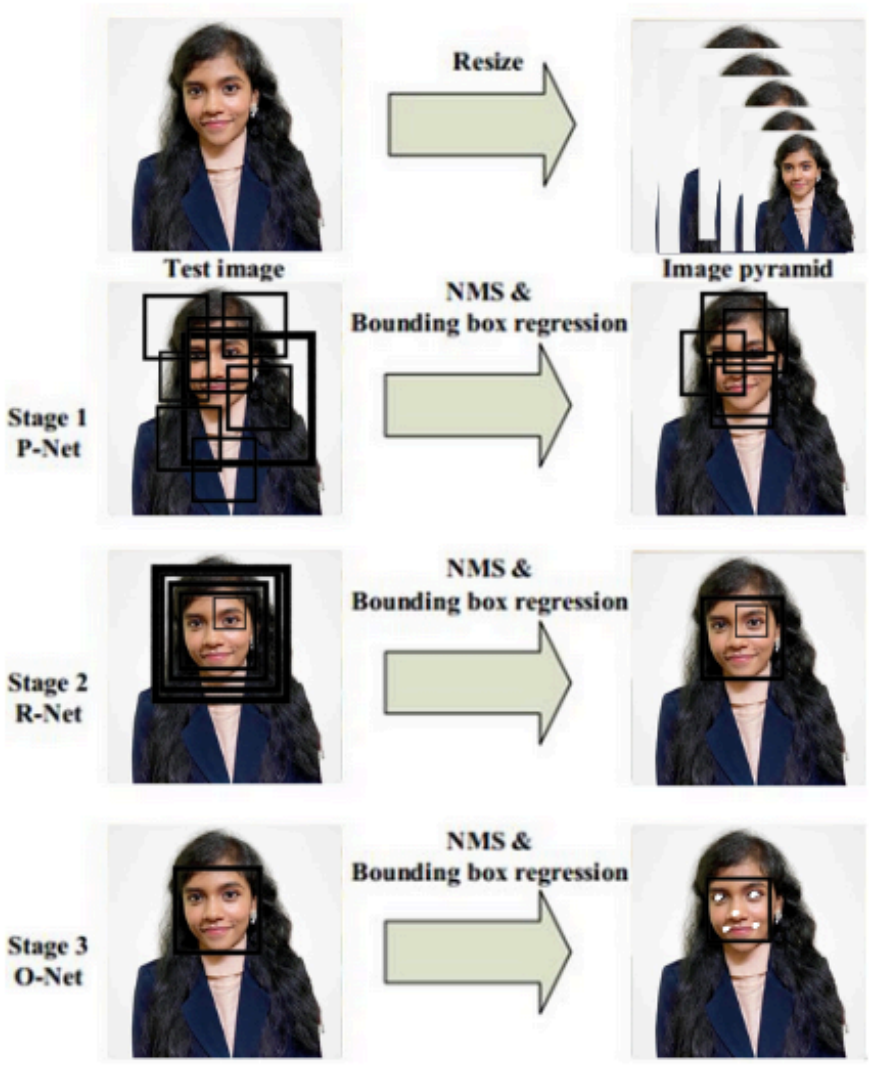


Fig 5.2.5 MTCNN

5.2.6 TENSORFLOW

TensorFlow is an open-source machine learning library used for building deep learning models. The images are classified using CNN. To generate a model means the classification of the images needs to provide a similar image which is the positive image. The image is then trained and retrained through anchoring or Transfer Learning. Additionally, TensorFlow is compatible with GPUs and TPUs

The process of writing a face detection code in Python using TensorFlow involves the following steps:

- Importing the necessary libraries: The first step is to import the required libraries, such as TensorFlow, OpenCV, and NumPy.

- Loading the MTCNN model: It provides a pre-trained MTCNN model to detect faces in an image and predict the coordinates of the bounding box .
- Reading the visual: The next phase is reading the picture, during which the faces need to be located. The OpenCV library might be used to do this.

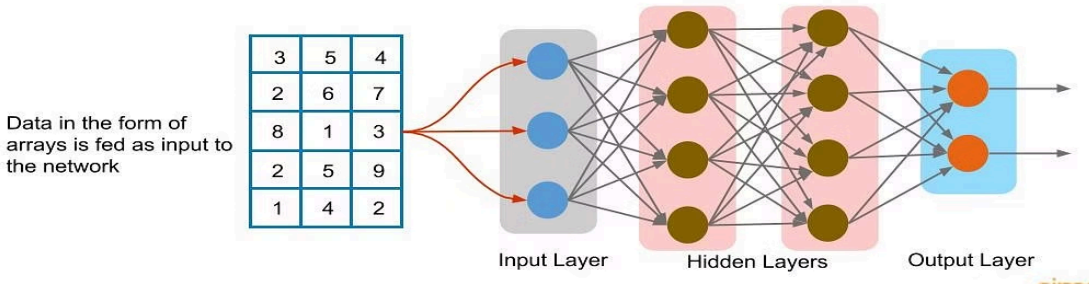


Fig 5.2.6 Tensorflow

5.3 LIBRARIES AND FRAMEWORKS

Libraries and frameworks are fundamental components in software development. Below are the components used for our project :

5.3.1 MATPLOTLIB

Matplotlib, a versatile plotting library in Python, can be leveraged effectively in facial recognition applications to visualize various aspects of the recognition process.

While not typically used directly for the recognition task itself, Matplotlib can aid in visualizing pre-processing steps such as face detection and alignment, showcasing the detected faces overlaid on the original images. Additionally, it can visualize the performance metrics of the recognition system, such as accuracy, and recall, through plots like bar charts or line graphs.

5.3.2 KERAS

The Keras, a high-level neural networks API, provides a user-friendly interface for designing, training, and deploying deep learning models. Keras facilitates the creation of CNN architectures capable of extracting meaningful features from facial images.

Additionally, Keras offers seamless integration with image processing libraries like OpenCV, streamlining the preprocessing and postprocessing stages of facial recognition pipelines. Keras allows users to customize and extend the library by defining their own layers, loss functions, optimizers, and other components.

5.3.3 TQDM

The key feature of tqdm is its ability to dynamically generate and update progress bars, providing real-time feedback on the progress of an operation. It can be integrated into loops and iterable objects to track the completion of iterations, giving you a representation of the progress.

By integrating TQDM into the facial recognition pipeline, developers can provide real-time feedback on the progress of face detection, feature extraction, and matching processes, allowing users to track the algorithm's execution and estimate remaining processing time.

5.3.4 SCIPY

Scipy, a computing library in Python, provides essential components such as numerical optimization, linear algebra operations, signal processing, and image manipulation capabilities.

These functionalities are vital for preprocessing of facial images, feature extraction, dimensionality reduction, and distance metric computation. Moreover, it seamlessly integrates with other Python libraries commonly used.

5.3.5 PICKLE

The "pickle" module in Python is used to serialize and deserialize Python objects. Serialization is the process of converting an object into a byte stream, while deserialization is the process of reconstructing the object from the serialized byte stream.

Pickle files are used to store trained models, or preprocessed data. For example, after training a model using machine learning, the model can be serialized and saved to a pickle file. This allows the model to be easily reused or distributed without having to retrain it every time.

5.3.6 UTILS

It refers to a collection of helper functions, classes, or modules that are used in machine learning tasks. Utils involve image cropping, resizing to standardize input images, histogram equalization for enhancing image contrast, and noise reduction to improve image quality.

It includes functions for extracting facial landmarks, aid in data augmentation to increase the diversity of training data, and for evaluating the performance of algorithms using metrics like accuracy, precision, recall, F1-score, etc.

5.3.7 GLOB

The glob module allows you to retrieve files that match a specified pattern. It is useful for data loading and preprocessing in machine learning. By specifying a pattern using wildcards, such as "*" or "? ", you can match files or directories based on their names or extensions.

The glob module provides a function called glob() that takes a pattern as an argument and returns a list of names that match that pattern. For example, when working with a large dataset that is spread across multiple directories, it can help gather all the relevant files or paths.

5.3.8 ARGPARSE

The argparse module simplifies the process of handling command-line arguments by automatically generating help messages, validating inputs, and providing a structured way to define and organize the available options.

For instance, it allows users to specify input parameters such as the directory containing facial images, the type of facial recognition algorithm to use, threshold values for matching faces, and the output format for results.

5.3.9 NUMPY

Numpy is a fundamental library for numerical computing in Python mainly used when working with arrays. It provides functions for tasks such as image loading, resizing, and normalization, allowing developers to preprocess facial images before feature extraction. Numpy's linear algebra functions are instrumental in dimensionality reduction techniques.

Additionally, its broadcasting and vectorization capabilities enhance the performance of computations involved in facial recognition algorithms, such as calculating distances between feature vectors or applying transformations to batches of images.

5.3.10 CV2

The cv2 module is the main module in OpenCV that provides developers with an easy to use interface for working with image and video processing functions. It helps extract discriminative features from facial images while reducing their dimensionality, making them suitable for classification or matching tasks.

5.4 DATA COLLECTION

Data collection is a crucial step in developing a face recognition system. The process involves gathering relevant data to train the face detection, and face recognition algorithms.

Collect a diverse dataset of face images representing different individuals. These images should cover variations in age, gender, ethnicity, and facial expressions. The images are collected using the webcam. Position the subject clearly in front of the camera in good lighting to achieve accurate pictures. First, a virtual environment must be created via the Anaconda prompt to execute the collection process. The command is :

conda create -n tf python=3.7 anaconda

After the environment is created, activate the environment. Activating an environment in Anaconda is a crucial step in managing Python dependencies and ensuring that your code runs in a consistent and isolated environment. The command is :

conda activate <environment_name>

To start clicking the pictures, the below command is executed. This results in the collection of around 100 images by the system. Name the directory with the respective name of the person, this name would be displayed during the identification process.

The images are then stored in the folder called newdata. Below is the command to create a folder:

python src/save_data.py --name (name of save_dir)

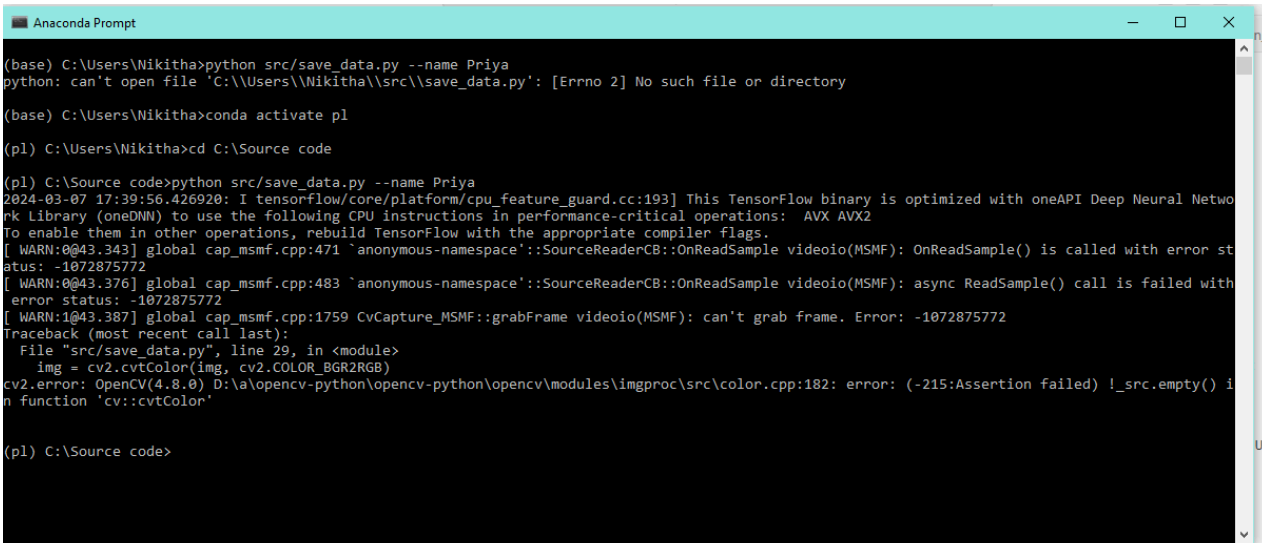


Fig 5.4 Data Collection

5.5 DATA PREPROCESSING

Data preprocessing refers to the process of transforming raw data into a more suitable for analysis or machine learning tasks. It is a crucial step in the data pipeline, as the quality and suitability of the input data can significantly impact the performance and accuracy of the system.

5.5.1 IMAGE DATA GENERATOR

The "ImageDataGenerator" module in machine learning is a part of the Keras library, used to preprocess and augment image data for training deep learning models. It offers a variety of image transformation techniques and data augmentation methods to increase the variability.

It allows you to generate augmented images during model training, eliminating the need to manually preprocess the dataset beforehand. This helps to improve model performance, reduce overfitting, and enhance the model's ability to generalize well to all unseen data.

5.5.2 IMAGE DATA AUGMENTATION

The goal of image data augmentation is to create new variations of the training images that are a representation of the scenarios the model encounters during inference. By introducing these variations, the model becomes capable of handling different conditions. It is often applied in conjunction with Python libraries, to incorporate them into the training pipeline.

Some common techniques used in image data augmentation include rotation, translation, scaling, flipping, shearing, zooming, brightness and contrast adjustment, and noise injection. These techniques can be combined or used individually depending on the requirements.

Fig 5.5.2.1 Types of Image Data Augmentation

The newly augmented results are stored at the aug_data folder. The augments we have used are rotation_range = 15, brightness_range=[0.4,1.5], and horizontal_flip.

The command for the augmentation of the new face is :

python augment_data.py

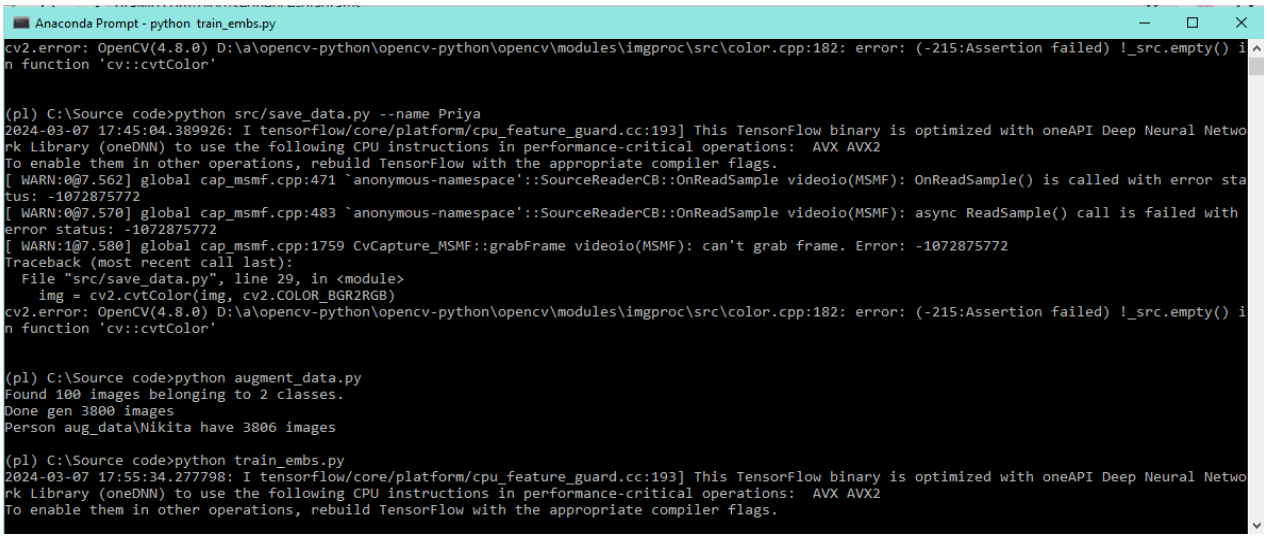


Fig 5.5.2.2 Image Data Augmentation

5.6 TRAINING THE MODEL

Training the data refers to the process of using a machine learning algorithm to learn patterns and relationships within a dataset. In the context of facial recognition, training the data involves feeding a large set of labeled facial images into a machine learning model. During training, the model learns to extract relevant features and map them to their corresponding labels.

To avoid the need for retraining the model every time it is used, developers often generate a "pickle" file. This allows objects to be serialized into a binary format that can be stored on disk. It contains the trained machine learning model, along with any associated parameters. Below is the directory as to how the data is stored.

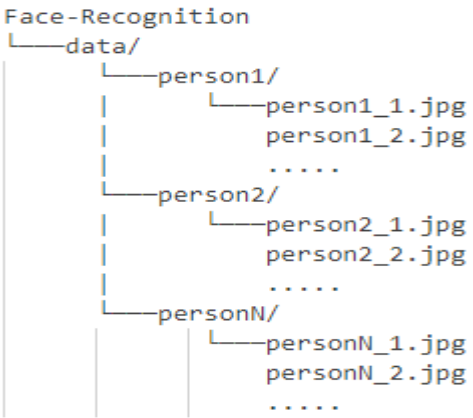


Fig 5.6 Training Directory

5.6.1 EMBEDDING WITH FACENET

The aligned facial images are then passed through an embeddings model, specifically Facenet. Embedding refers to a numerical representation of a face extracted from an image. When a face recognition system processes an image, it uses a deep learning model to extract features from the face, transforming it into a compact numerical high-dimensional vector representation or embedding. These embeddings are then compared with embeddings of known faces stored in a database to identify or verify the person in the image.

FaceNet utilizes a convolutional neural network (CNN) architecture to learn a compact representation, or embedding, of a face image in a high-dimensional space. This embedding is learned in such a way that similar faces are mapped close to each other in this space, while dissimilar faces are mapped farther apart. One of the key innovations of FaceNet is its use of a triplet loss function during training. This loss function compares the similarity of three images: an anchor image (representing the identity to be recognized), a positive image (representing the same identity as the anchor), and a negative image (representing a different identity).

The model is trained to minimize the distance between the embeddings of the anchor and positive images while maximizing the distance between the embeddings of the anchor and negative images. This encourages the model to learn embeddings that are discriminative and also invariant to factors such as lighting, pose, and facial expressions.

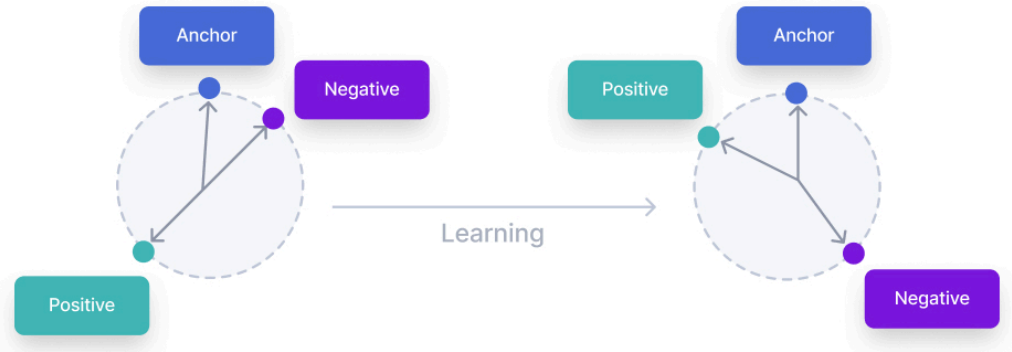


Fig 5.6.1.1 Triplet Loss

After embedding, the embedded file will be saved to output/train_embs.pickle. This process will take some time. The command for the embedding of the new face is :

python train_embs.py

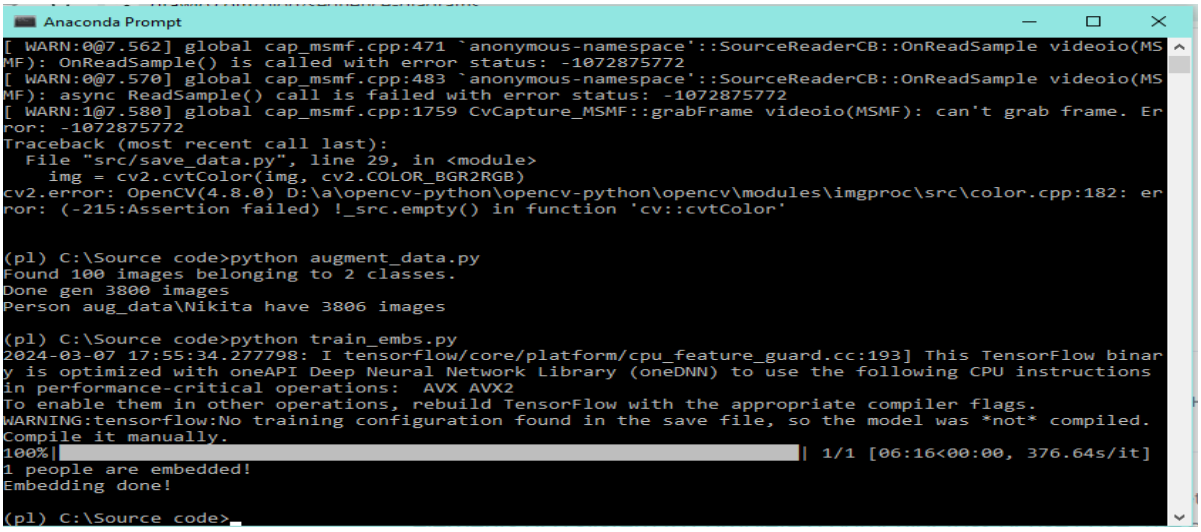


Fig 5.6.1.2 Embedding

5.7 FACE DETECTION

Face detection is the task of detecting a human face on an image. This is done through extracting a list of bounding boxes, coordinates of smallest possible rectangles around faces.

MTCNN leverages a 3-stage neural network detector. First, the image is resized multiple times to detect faces of different sizes. Then the P-network (Proposal) scans images, performing first detection. The proposed regions are input for the second network, the R-network (Refine), which filters detections to obtain quite precise bounding boxes. The final stage, the O-network (Output) performs the final refinement of the bounding boxes.

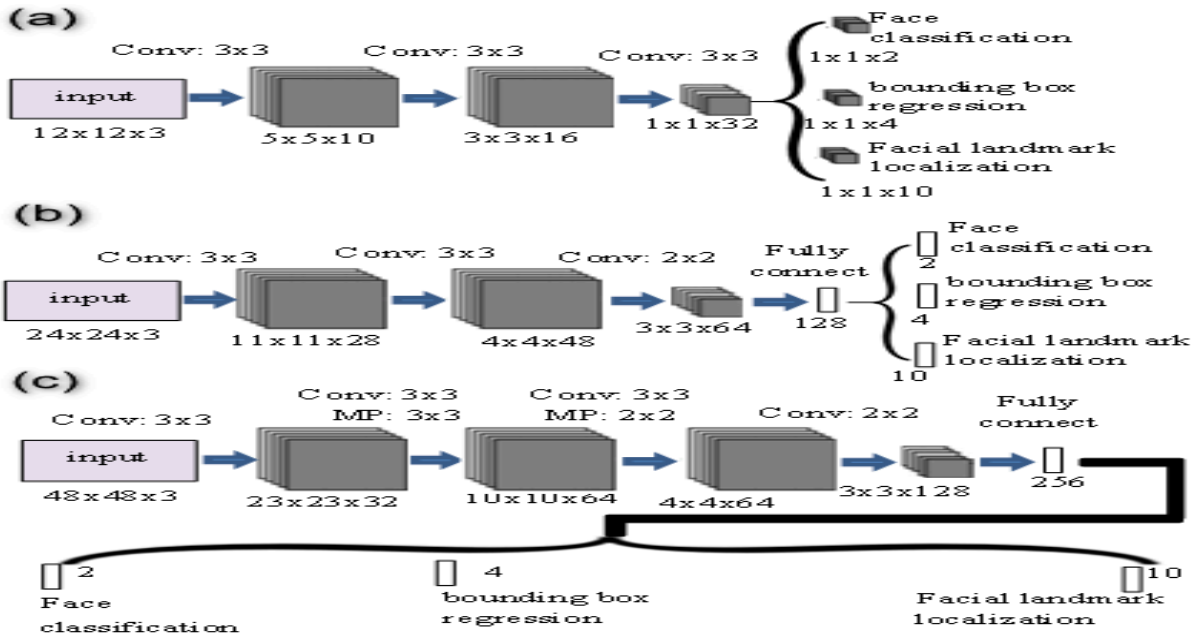


Fig 5.7 Face Detection using MTCNN

5.8 FACE ALIGNMENT

Face alignment refers to the process of detecting and locating key facial landmarks on a face and then adjusting the image to a standardized pose or alignment. These landmarks typically include points such as the corners of the eyes, the tip of the nose, and the corners of the mouth.

The purpose of face alignment is to ensure that facial features are consistently positioned and oriented across different images, regardless of variations in head pose, facial expression, or lighting conditions. By aligning faces to a common reference frame, facial recognition algorithms can more accurately compare and match facial features for identification or verification purposes.

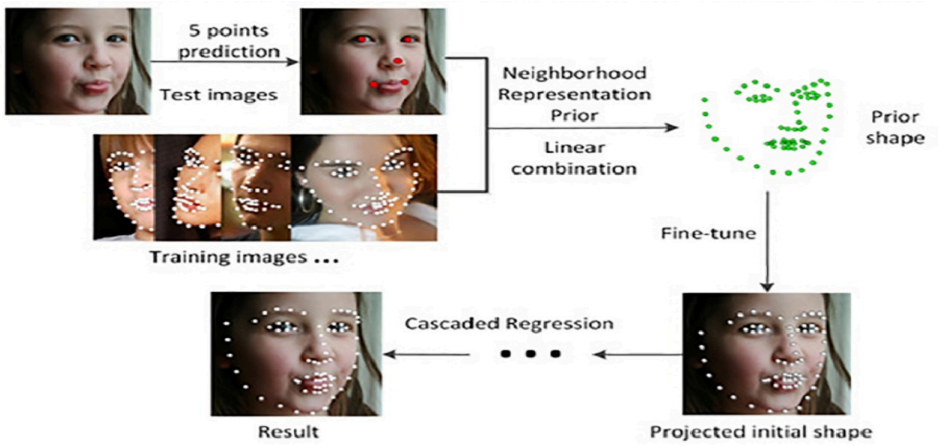


Fig 5.8 Face Alignment

5.9 CLASSIFICATION

Classifier is a device which decides whether the taken image is negative or positive. It is trained on hundreds of thousands of face and non-face images to learn to classify a new image as a face or non-face image correctly. The classifier then predicts the identity of new faces by comparing their features with those in the training data.

OpenCV provides two pre-trained classifiers Haar Classifier and LBP Classifier.

- **HAAR CLASSIFIER**

This classifier chases a machine learning procedure in which a cascade operation is inculcated from the photos to discover items. Then, the characteristics are drawn out from the picture. Each characteristic is an individual value, which is acquired by subtracting the sum of pixels in white rectangle from the summation of pixels in black rectangle in which it detects the faces of different individuals in different environments.

The process can be described as follows:

- Loading the input image using the built-in function *cv2.imread(img_path).*
- Converting it to grayscale mode and then displaying it.
- Loading the haar cascade classifier.

The Haar Value Classification is as follows:

$$\begin{aligned}\text{Pixel value} &= (\text{Sum of the Dark pixels/Number of Dark pixels}) \\ &\quad - (\text{Sum of the Light pixels/Number of Light pixels})\end{aligned}$$

- **LBP CLASSIFIER**

By utilizing LBP operators, individual photos are examined as a structure of micro-patterns. Then the histogram of LBP is enumerated throughout the face, which encrypts the circumstances of micro-patterns. The figure of documentation is assembled by splitting face pictures toward minor non overlapping sections. The LBP labels the pixels by threshold the 3 × 3 neighborhood in relation to the central pixel value. In a particular numerical scale the common features, such as edges, lines, points, can be represented by a value.

The process can be described as follows:

- Loading the input image using the built-in function *cv2.imread(img_path).*
- Converting it to grayscale mode and then displaying it.
- Loading the LBP classifier.

The LBP for each pixel is calculated. For each pixel p, the 8 neighbors of the center pixel are compared with the pixel p and the neighbors are assigned a value 1 if x is greater than or equal to p. The following defined as the formula for the calculation of LBP Classifier:

$$\mathrm{LBP}(\mathrm{xc}, \mathrm{yc}) = \sum_{(\mathrm{p}=0)}^{(\mathrm{p}-1)} 2^{\mathrm{ps}(\mathrm{ip}-\mathrm{ic})}$$

where, (xc,yc) is the centre pixel ic is the brightness ip is the brightness of adjacent pixels s(.) is a sign function defined as: s(x)=1 if x≥0 and s(x)=0 otherwise.

- **COMPARISON OF HAAR AND LBP**

Let us calculate the accuracy for both the classifiers.

True positive (TP): It is an actual object of interest that is correctly identified.

True-negative (TN): It is a non-object of interest falsely identified as a true object.

True positives rate (TPR) = TP/(TP+FP)

False-positives (FP): It is a non-object of interest which is falsely identified as the true object.

False-negatives (FN): It is an actual object of interest falsely identified as negative.

False Negatives Rate (FNR)=FN/(FN+TP)

Hence, the formula is as follows:

Accuracy = (TP+TN) / (TP+TN+FP+FN)

Accuracy obtained for the Haar cascade is 96.24% and for LBP classifier 94.74%. As a result, Haar cascade has more accuracy and detects more faces than the LBP classifier but time taken by LBP is less than Haar compared.

The bar graph shows the comparison between Haar and LBP based on the execution time. The X-axis implies the total number of faces in the image and the Y-axis implies execution time.

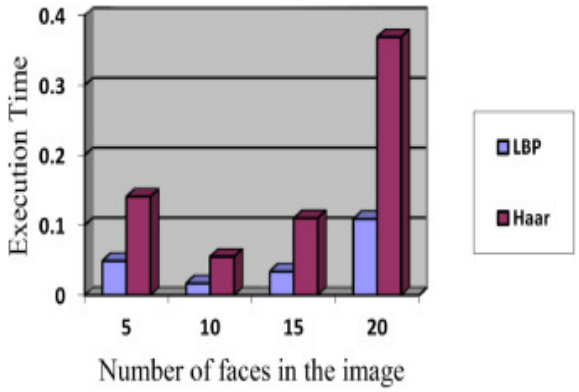


Fig 5.9 Haar vs LBP

Table 1 and Table 2 contains the execution time, number of faces detected and accuracy of both classifiers. Time is in the form of seconds and accuracy is in the form of percentage.

No.of faces in an image	Execution Time (sec)	No.of faces detected	Accuracy (%)
5	0.141	5	100
10	0.055	9	90
15	0.11	12	80
20	0.369	19	95

Table 5.9.1 Haar Cascade

No.of faces in an image	Execution Time (sec)	No.of faces detected	Accuracy (%)
5	0.049	5	100
10	0.017	8	80
15	0.034	11	73.33
20	0.109	17	85

Table 5.9.2 Local Binary Pattern

The conclusion is that the Haar classifier is considered more effective than the LBP classifier.

- **ACCURACY AND LOSS**

Accuracy measures the proportion of correct predictions made by the model on the training dataset. It indicates how well the model is performing. Loss quantifies the difference between the predicted output of the model and the true target values in the training dataset. It represents the discrepancy between the predicted and actual values and is used as a measure of how well the model is performing.

During the training process, both accuracy and loss are plotted over each epoch. The accuracy tends to increase over epochs as the model learns to make better predictions, while the loss tends to decrease as the model minimizes the error between predicted and true values.

The below command classifies the embedded face and calculates the loss and accuracy:

python train_classify.py

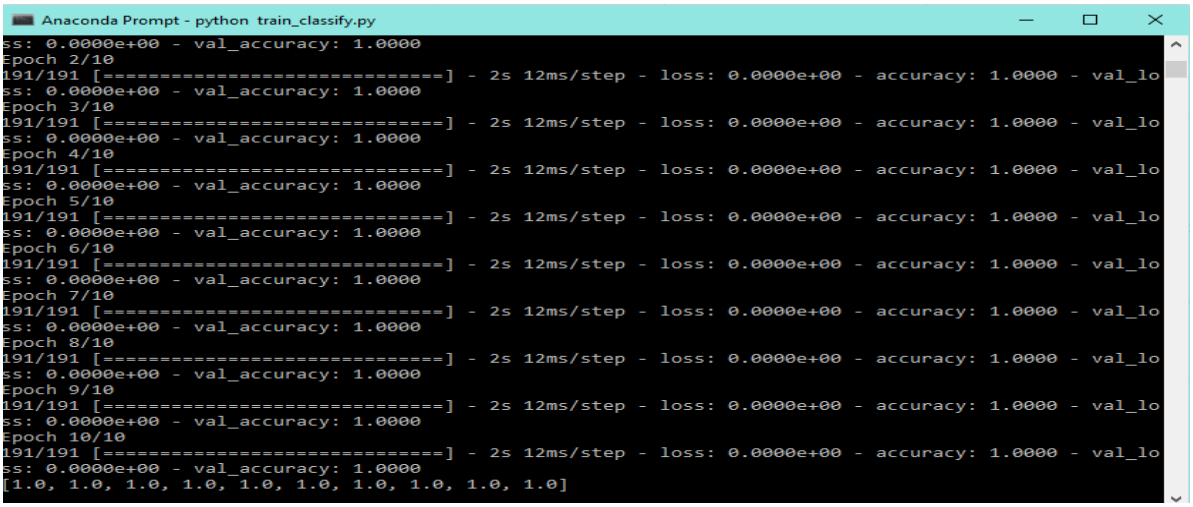


Fig 5.9.1.1 Classification

5.9.1 SOFTMAX APPROXIMATION

The softmax function is often applied to the output layer of a neural network to convert raw scores or logits into probabilities. These probabilities represent the likelihood of each class or identity being present in the input data. It takes input the features extracted from facial images and outputs a vector of scores corresponding to different identities or classes.

The softmax function then normalizes these scores into probabilities by applying the following formula to each score :

$$P(class_i) = \frac{e^{score_i}}{\sum_j e^{score_j}}$$

where, P(class i) is the probability of the i-th class, score i is the raw score or logit for the i-th class, e is the base of the natural logarithm, and the summation j is the sum of the exponentials of all raw scores.

It essentially transforms the raw scores into a probability distribution over all possible classes, allowing us to interpret the output as the likelihood of each identity. This softmax output is then used to determine the most likely identity of the face in the input image, facilitating accurate identification and classification of individuals.

5.9.2 COSINE SIMILARITY

Cosine similarity is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. By converting facial images into these numerical representations, cosine similarity quantifies the degree of resemblance.

A value of 1 denotes perfect similarity, a value of -1 signifies complete dissimilarity, and values closer to 0 imply a lack of significant similarity. In face identification, selecting the one with the highest cosine similarity score as the match.

The cosine similarity between the two feature vectors A and B, can be calculated by the following formula :

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

5.9.3 EPOCH CALCULATION

An epoch refers to one complete pass of the entire training dataset through the neural network during the training phase. In simpler terms, it's one iteration through the entire dataset.

During training, the neural network adjusts its parameters (weights and biases) using optimization algorithms such as gradient descent to minimize the error between the predicted outputs and the actual labels. This process is done iteratively over multiple epochs until the model converges to a satisfactory level of performance. Each epoch consists of multiple iterations, where the training dataset is divided into smaller batches. These batches are fed into the neural network sequentially, and after processing all the batches in an epoch, the model's parameters are updated based on the accumulated gradients.

The graph shows the training and testing accuracy of a machine learning model as the number of epochs increases. The x-axis of the graph is labeled "epochs" and the y-axis is labeled "accuracy". The training accuracy is a measure of how well the model performs on the data it was trained on, and the testing accuracy is a measure of how well the model performs on unseen data. In the graph, the training accuracy starts at 1.0 and stays at 1.0 for all epochs, which suggests that the model is perfectly fitting the training data. The testing accuracy starts around 0.98 and fluctuates slightly as the number of epochs increases.

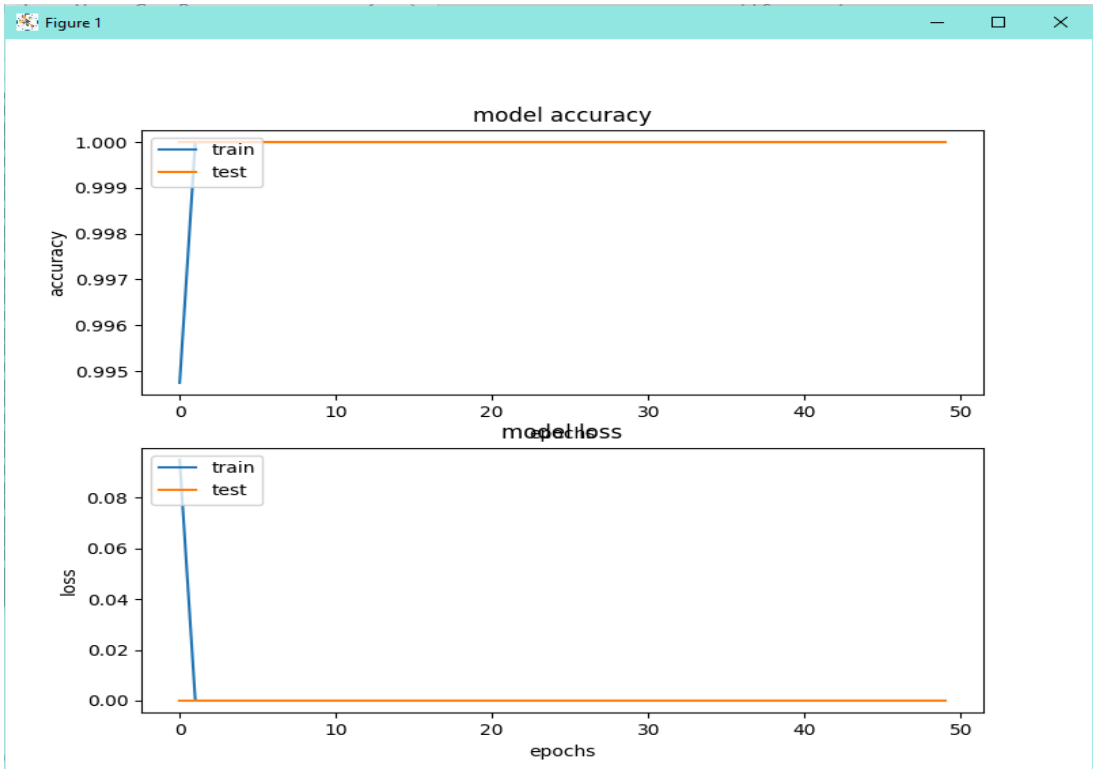


Fig 5.9.3 Epoch Graph

5.10 FACE IDENTIFICATION

FaceNet is a facial recognition framework that operates by encoding faces into a high-dimensional vector space, where similar faces are located close to each other.

When a face is detected in an image, FaceNet preprocesses the image and then feeds it into the neural network to generate a numerical representation, known as an embedding, for that face. These embeddings are then compared using cosine similarity or other distance metrics to determine the similarity between faces.

The first step is to detect the presence and location of faces within each frame of the video stream. This is done using techniques such as Haar cascades, Histogram of Oriented Gradients, or deep learning-based methods like Convolutional Neural Networks. Once faces are detected, they may need to be aligned and normalized to a standard size and orientation for better recognition accuracy. This helps to account for variations in pose, scale, and orientation.After normalization, facial features are extracted from the aligned faces. This process involves capturing important characteristics of the face, such as the positions of key landmarks or numerical representations of facial patterns.Finally, the extracted features are compared against a database of known faces or templates using similarity metrics such as cosine similarity or Euclidean distance. If a match is found above a certain threshold, the person's identity is recognized.

The below command is used for face identification after detecting their face in the webcam :

python stream_recognition.py

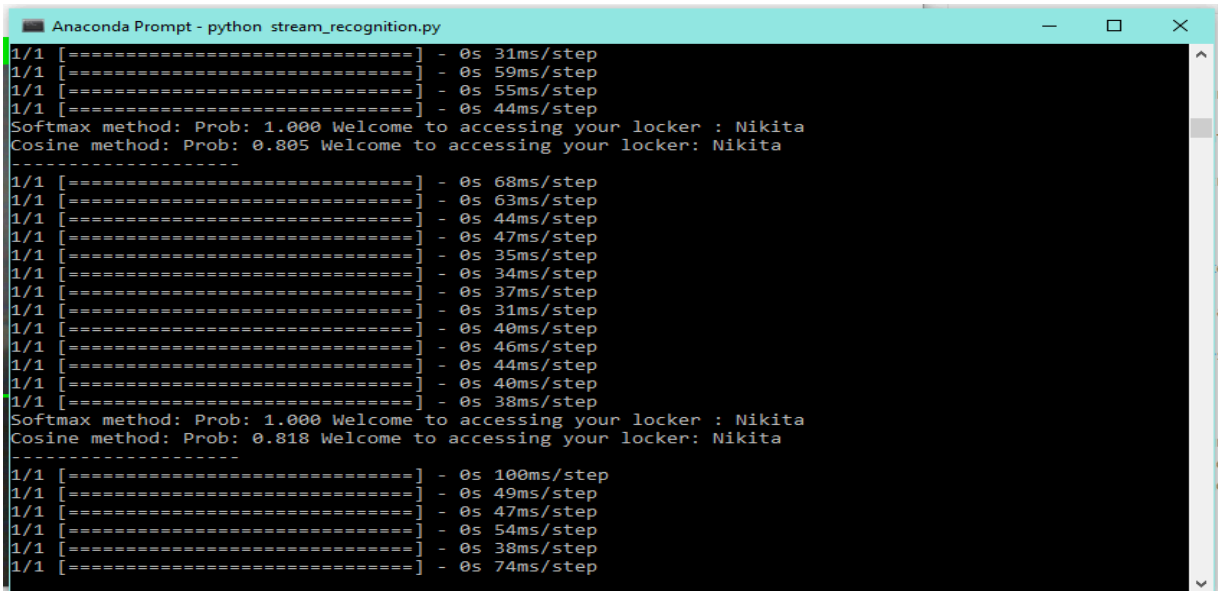


Fig 5.10 Face Identification

CHAPTER 6

TESTING

6.1 TEST CASE 1 - RECOGNITION OF FACES DESPITE PRESENCE OF MASK

In traditional facial recognition systems, the algorithms are typically trained and optimized to recognize faces without obstructions. Facial recognition with a mask involves enhancing existing face recognition algorithms to account for the presence of masks on individuals' faces.

This includes developing new face detection and alignment techniques. Additionally, specialized deep learning models can be trained to extract facial features from regions of the face that remain visible despite the mask, such as eyes, eyebrows, and forehead.

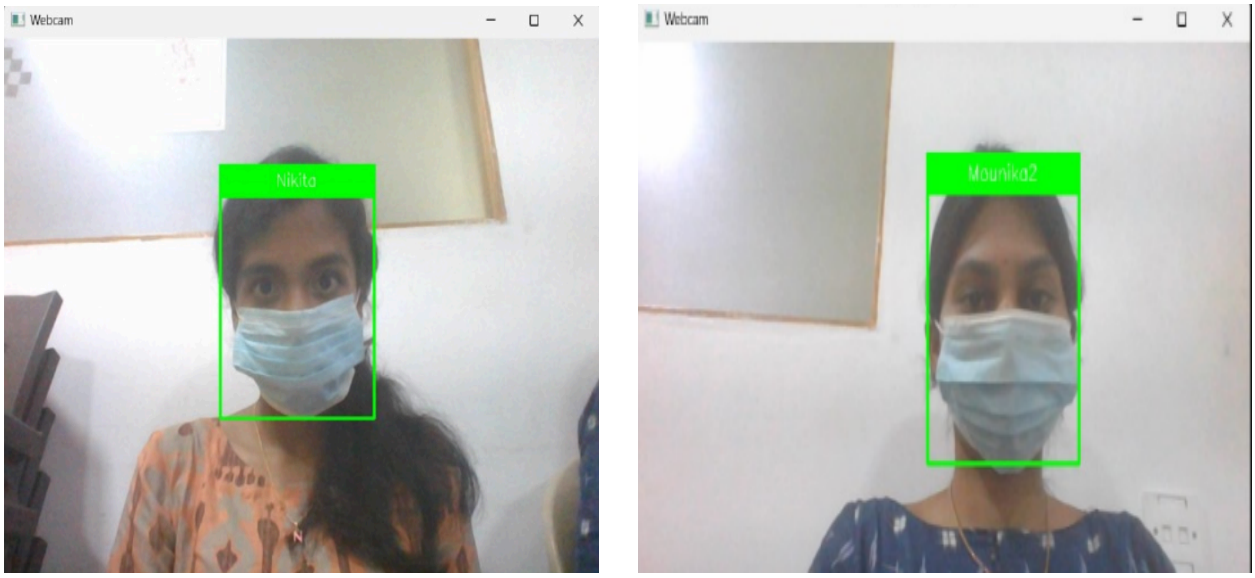


Fig 6.1 Test Case 1

TEST CASE NO.	TEST CASE	MODULE NAME	GIVEN INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST STATUS
1	Recognition of faces despite the presence of masks.	Face Identification	The individual is seen wearing the mask in the camera.	The face of the individual must be detected when they are wearing the mask.	The individual is identified correctly and a green boundary box is seen.	PASSED

Table 6.1 Test Case 1

6.2 TEST CASE 2 - RECOGNITION OF FACES WITH ABSENCE OF MASK

Facial recognition without mask refers to the conventional mode of facial recognition where the algorithms operate on unobstructed faces. In this scenario, the system relies on detecting and analyzing the entire face to extract discriminative features for recognition purposes.

These features may include the shape of the face, the position of facial landmarks, and the distribution of texture patterns. It typically involves the use of computer vision algorithms and machine learning techniques. These algorithms analyze patterns in facial features and create a unique representation of each face, often referred to as a facial template or embedding.When individuals are not wearing masks, facial recognition systems can more accurately match their faces against a database of known faces.

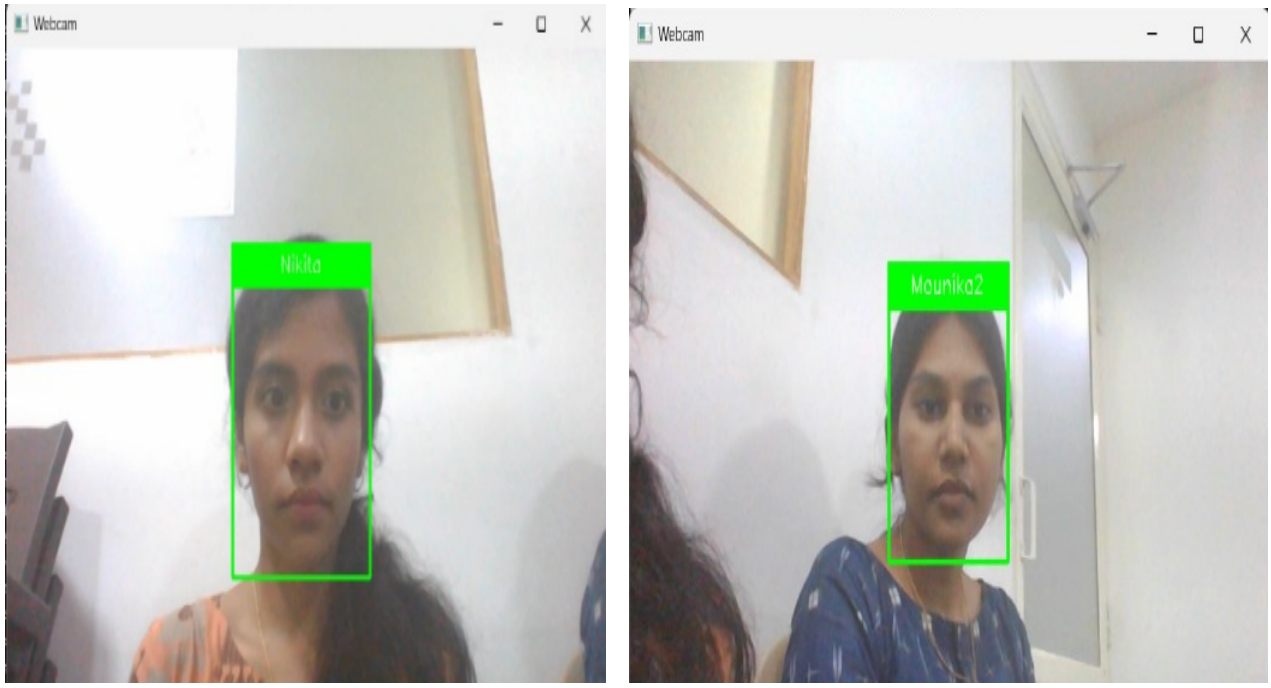


Fig 6.2 Test Case 2

TEST CASE NO.	TEST CASE	MODULE NAME	GIVEN INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST STATUS
2	Recognition of faces with absence of masks.	Face Identification	The individual is seen wearing no mask in the camera.	The face of the individual must be detected when they are not wearing the mask.	The individual is identified correctly and a green boundary box is seen.	PASSED

Table 6.2 Test Case 2

6.3 TEST CASE 3 - UNTRAINED FACES

Untrained faces refer to faces that the system has not been explicitly trained to recognize or classify. When a facial recognition system is initially developed or deployed, it typically undergoes a training phase where it learns to recognize a specific set of faces based on the data it has been trained on.

Untrained faces are faces that the system encounters during operation but has not encountered during the training phase. These faces may belong to individuals who were not part of the training dataset. Since the system has not learned to recognize these faces during training, it may struggle to accurately identify or classify them.

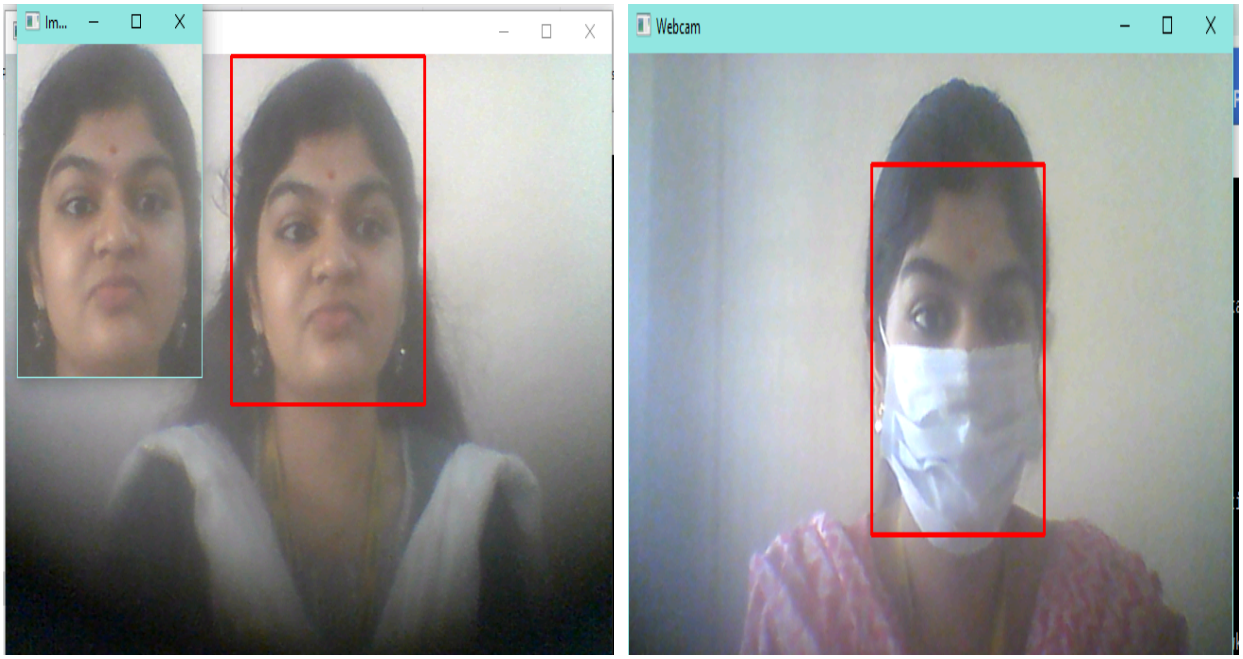


Fig 6.3 Test Case 3

TEST CASE NO.	TEST CASE	MODULE NAME	GIVEN INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST STATUS
3	Untrained Faces.	Face Identification	The untrained face of the individual is seen in the camera.	The face of the individual is not trained in the system and hence must not be detected.	The individual is not identified and a red boundary box is seen.	PASSED

Table 6.3 Test Case 3

6.4 TEST CASE 4 - RESTRICTED ACCESS

Restricted access to untrained faces in a facial recognition system refers to the implementation of measures that limit the system's ability to recognize or authenticate individuals who are not included in its training dataset.

In other words, only faces that have been specifically trained and included in the system's database can be accurately identified or verified. This restriction helps ensure the security and integrity of the facial recognition system by preventing unauthorized access or false positives from occurring when unfamiliar faces are encountered. By restricting access to untrained faces, the system maintains a higher level of accuracy and reliability in its recognition capabilities, making it more suitable for applications where precise identification and authentication are critical, such as access control, surveillance, and identity verification services.

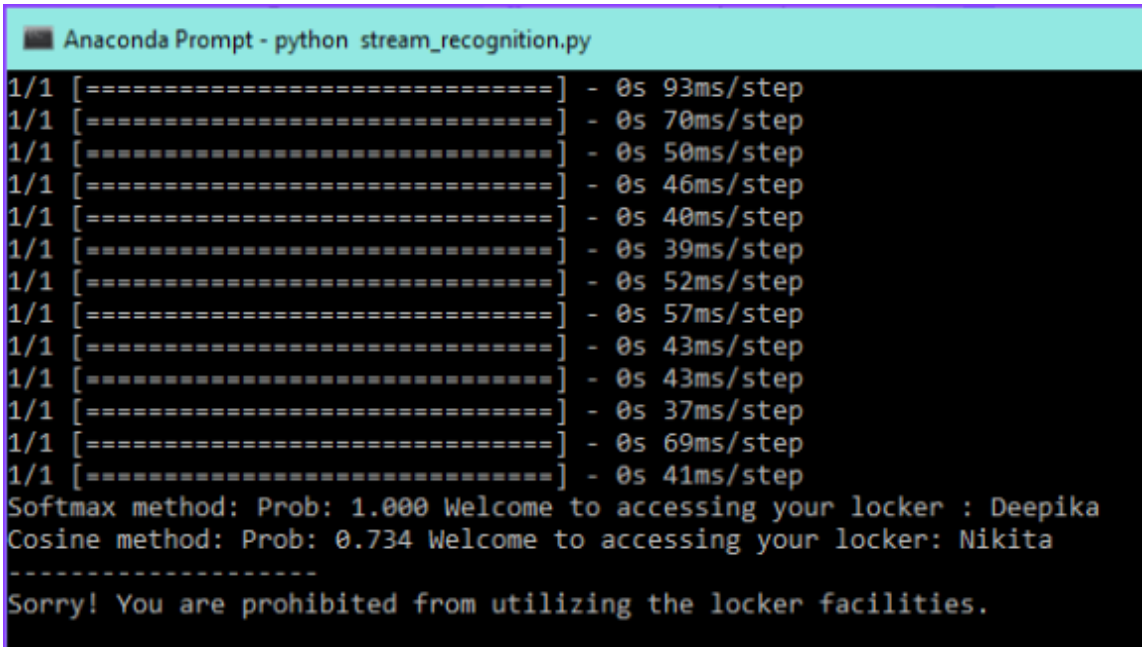


Fig 6.4 Test Case 4

TEST CASE NO.	TEST CASE	MODULE NAME	GIVEN INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST STATUS
4	Restricted Access.	Face Identification	The untrained face of the individual is seen in the camera.	The face of the individual who is not trained must not be granted access	The individual is not identified and access is not granted	PASSED

Table 6.4 Test Case 4

6.5 TEST CASE 5 - CALCULATION OF ACCURACY AND LOSS

In the context of a facial recognition system, accuracy and loss serve as crucial metrics for evaluating the performance of the model.

Accuracy refers to the measure of how often the system correctly identifies or classifies faces within a given dataset. It represents the ratio of correctly predicted faces to the total number of faces in the dataset and provides an indication of the model's overall effectiveness in recognizing faces.

On the other hand, loss quantifies the discrepancy between the predicted outputs of the model and the actual labels in the training data. It serves as an optimization objective during the training process, with the goal of minimizing the loss to improve the model's performance.

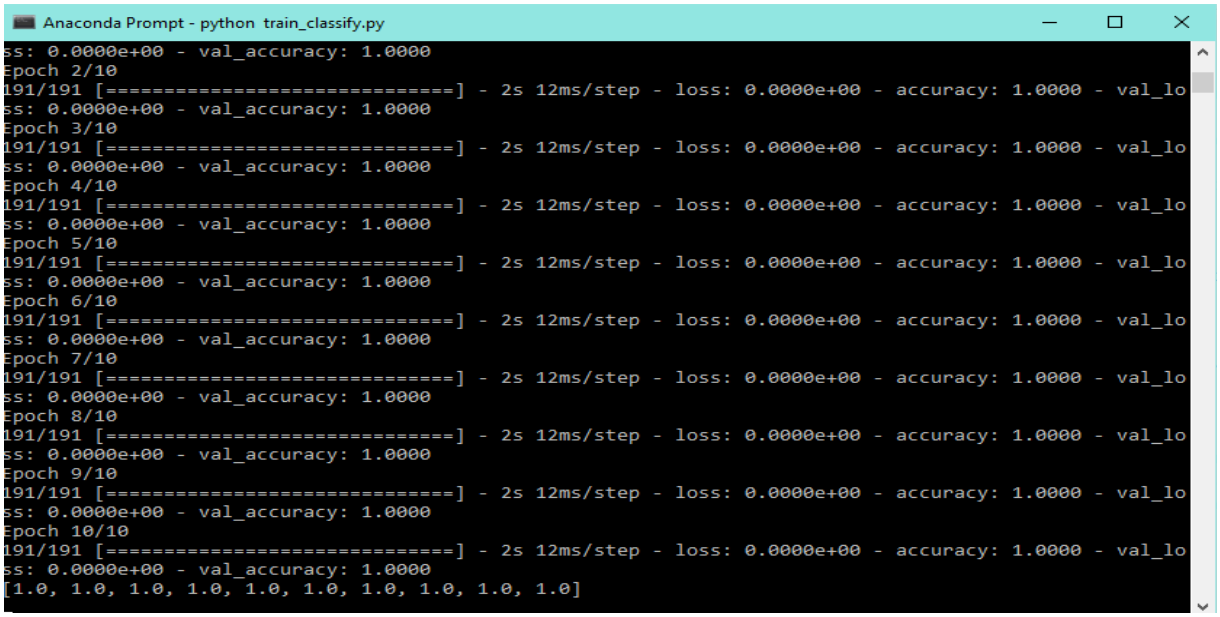


Fig 6.2.5 Test Case 5

TEST CASE NO.	TEST CASE	MODULE NAME	GIVEN INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST STATUS
5	Calculation of accuracy and loss.	Classification	The trained face of the individual is classified.	The accuracy and loss for the classification are to be displayed.	The accuracy and loss are shown exactly and a graph is drawn.	PASSED

Table 6.5 Test Case 5

CHAPTER 7

RESULT AND DISCUSSION

The comparison between the traditional face recognition system and the new improved facial recognition system employing MTCNN and FaceNet reveals significant enhancements, particularly evident when considering various parameters. Without masks, the new system exhibited substantially higher accuracy, robustness to variations, speed, and reduced dependency on handcrafted features compared to the traditional approach. The traditional system demonstrated moderate performance across these metrics, whereas the new system achieved superior results. This improvement is attributed to the utilization of advanced deep learning techniques for feature extraction and identification. However, with masks, while both systems experienced performance degradation, the new system still outperformed the traditional one across all parameters. Notably, the new system maintained higher accuracy, robustness, and speed even in the presence of occlusion, indicating its effectiveness in handling challenging scenarios. Moreover, the reduced dependency on handcrafted features in the new system suggests its adaptability to diverse datasets and environments. Overall, these findings underscore the considerable advancements offered by the new facial recognition system.

PARAMETERS	VALUES
Accuracy	95%
Robustness to variations	60%
Resource Consumption	50%
F1 Score	0.69
Dependency on features	90%

Table 7.1 Existing System Performance

PARAMETERS	VALUES
Accuracy	98%
Robustness to variations	80%
Resource Consumption	30%
F1 Score	0.89
Dependency on features	40%

Table 7.2 Proposed System Performance

- **ACCURACY**

Accuracy refers to the ability of the system to correctly identify or verify individuals from their facial features. It is typically measured as the percentage of correct identifications out of the total number of attempts. Without masks, both systems exhibit comparable accuracy levels, but the new system demonstrates its adaptability, effectiveness in the presence of facial coverings.

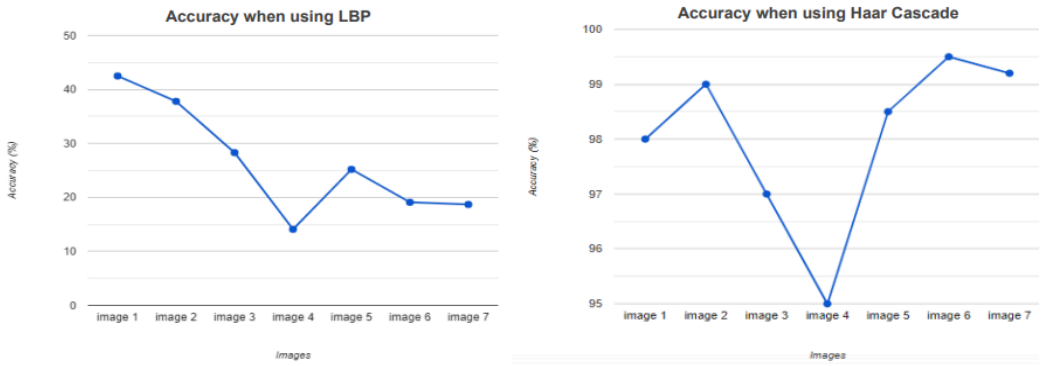


Fig 7.1 Accuracy

- **ELAPSED DURATION**

It refers to the amount of time it takes for the system to process and analyze a given image or video frame to identify faces. When considering the presence of masks, traditional systems struggle to identify individuals, requiring additional steps such as manual verification. In contrast, the new system has adapted to accommodate masks, effectively reducing the time.

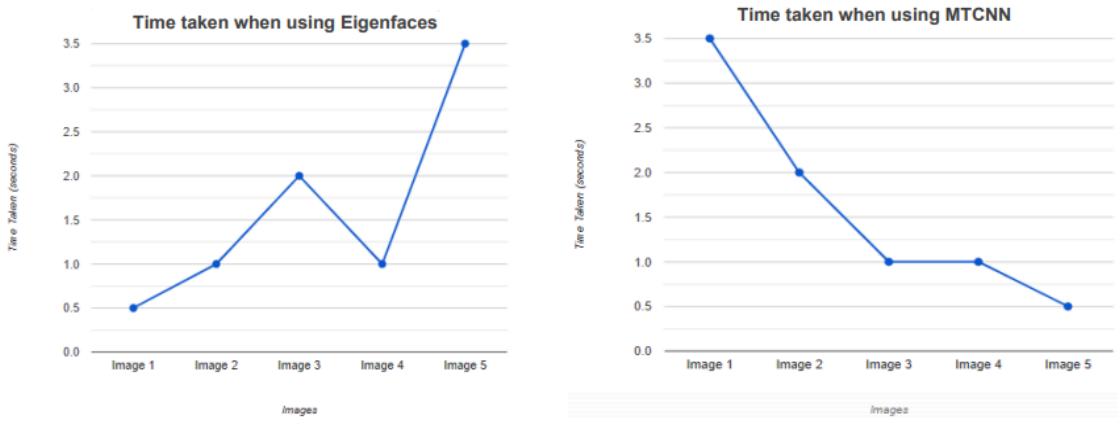


Fig 7.2 Elapsed Duration

- **RESOURCE CONSUMPTION**

Traditional facial recognition systems rely on processing the entire facial structure, consuming considerable computational resources. The proposed system can process facial features even when partially covered, thereby reducing computational overhead.

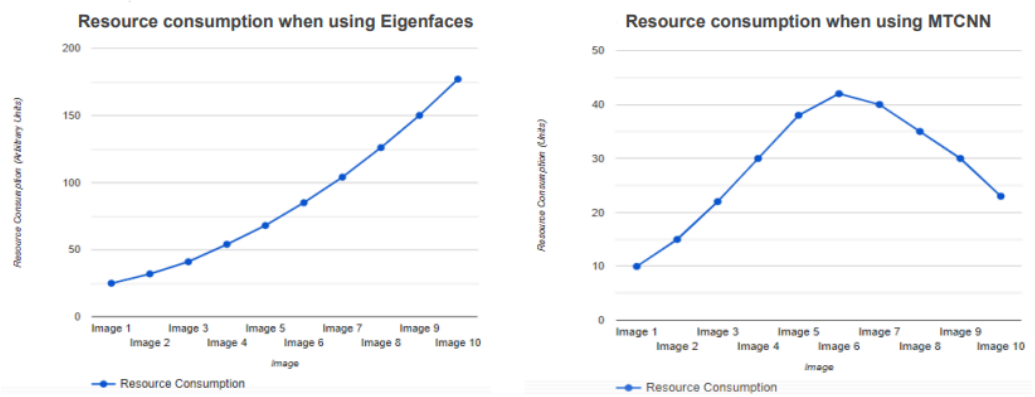


Fig 7.3 Resource Consumption

- **ENVIRONMENTAL RESILIENCE CAPABILITY**

Traditional systems struggle in environments where factors like lighting changes or partial obstructions, such as masks make them less resilient. The proposed system, especially designed to adapt to these challenges, demonstrates superior environmental resilience.

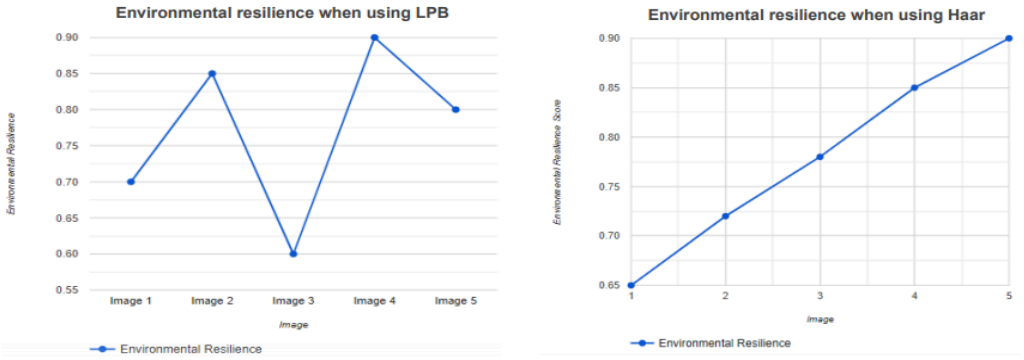


Fig 7.4 Environmental Resilience

- **CROSS DOMAIN PERFORMANCE**

Evaluation of the system's performance when applied across different domains or applications, such as surveillance, access control, or user authentication.

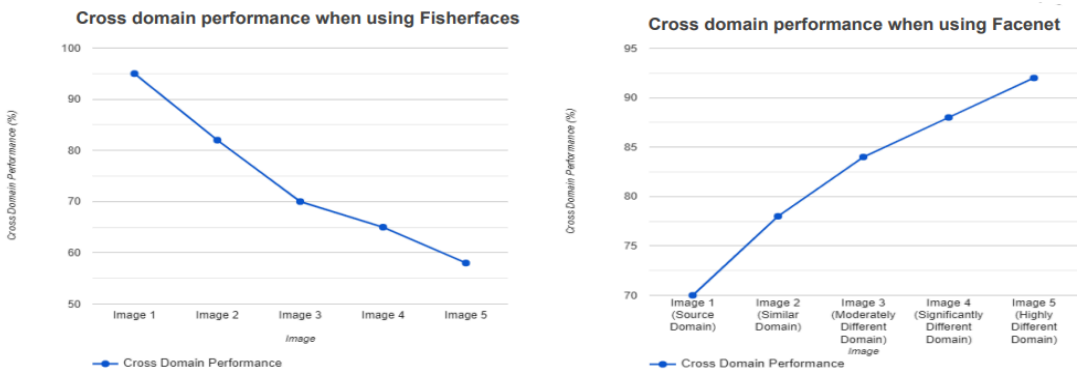


Fig 7.5 Environmental Resilience

CHAPTER 8
CONCLUSION

The face recognition system with mask detection using MTCNN and OpenCV presents a solution to the challenges posed by masks in accurate identification and verification. By integrating mask detection with face recognition algorithms, the system improves the overall accuracy and reliability of recognizing individuals, even when they are wearing masks. The system offers several advantages, including enhanced security, improved access control, and practical applicability in various settings. It can be customized and adapted to different environments, ensuring its versatility. Additionally, the incorporation of alternative biometric modalities provides complementary information for identification when masks are present. The proposed system respects privacy considerations by focusing on identification only when masks are not detected, preserving individuals' privacy while maintaining security requirements. Overall, the face recognition system with mask detection using MTCNN and OpenCV is a valuable tool for ensuring accurate and reliable identification, enhancing security measures, and adapting to the challenges presented by masks in today's environment. In summary, the proposed system leveraging MTCNN and FaceNet demonstrates superior performance in face recognition tasks compared to traditional methods, achieving high accuracy rates of approximately 98% for unobstructed faces within 100-200 epochs. Despite a slight decrease in accuracy to around 95% with the introduction of masks, the system's adaptability and robustness are evident, requiring only a modest increase in training epochs, typically around 300, to maintain optimal performance. Overall, the proposed system presents a compelling solution for reliable face recognition, particularly in challenging scenarios such as mask occlusion.

CHAPTER 9

FUTURE ENHANCEMENTS

Future work for the face recognition system with mask detection could involve the following areas of focus :

- **Improved Mask Detection:** Enhance the accuracy and robustness of the mask detection algorithm by incorporating advanced machine learning techniques, such as deep learning models or ensemble methods. Explore the use of larger and more diverse datasets for training the mask detection module to handle various mask types, styles, and orientations.
- **Multi-Modal Fusion:** Investigate the integration of multiple biometric modalities, such as facial features, voice recognition, gait analysis, or even thermal imaging, to further enhance the identification accuracy. Develop methods to effectively combine information from different modalities for reliable identification in various scenarios.
- **Integration with Bank Locker Access System :** Integrate the face identification system with the existing bank locker access control infrastructure, for seamless integration and interoperability. Develop APIs or protocols for communication between the face recognition module and the locker access system, ensuring real-time authentication and authorization of individuals based on their mask status and identity.
- **Real-time Monitoring and Logging:** Implement real-time monitoring of individuals entering the bank locker facility, capturing their faces and mask status. Log this information along with timestamps for auditing purposes and to ensure compliance with security protocols.
- **Scalability and Robustness:** Ensure that the system is scalable to handle a large number of users and robust enough to perform reliably in real-world conditions. Implement error handling mechanisms and redundancy measures to mitigate potential failures and ensure uninterrupted operation.

By focusing on these future work areas, the face recognition system can continue to evolve, addressing the challenges and requirements of accurate identification in the presence of masks and contributing to the advancement of biometric recognition technologies.

ANNEXURE I

SOURCE CODE

- **IMAGE AUGMENTATION**

```
import tensorflow as tf
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
import glob
from keras.preprocessing.image import ImageDataGenerator
def my_preprocessing_func(img):
    image = np.array(img)
    return image / 255
def find_label(class_dict, need_find):
    for key, value in class_dict.items():
        if value == need_find:
            return key
new_data_dir = './aug_data/'
if not os.path.exists(new_data_dir):
    os.makedirs(new_data_dir)
datagen = ImageDataGenerator(rotation_range=15, fill_mode='nearest', brightness_range=[0.4,1.5])
batch = next(img)
while(batch):
    if(count > 3799):
        break
    for i in range(len(batch[0])):
        image = batch[0][i]
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        real_label = find_label(class_dictionary, np.argmax(batch[1][i]))
        save_dir = new_data_dir + real_label + '/'
         if not os.path.exists(save_dir):
            os.makedirs(save_dir)
               img_name = real_label + "_" + str(count) + ".jpg"
        img_save_path = save_dir + img_name
        cv2.imwrite(img_save_path, image)
    batch = next(img)
print("Done gen {} images".format(count))
dir = glob.glob("./aug_data/*")
```

```
for dir_ in dir:
    dir_ = dir_ + "/*"
    img = glob.glob(dir_)
    for img_ in img:
        count += 1
    dir_name = dir_.split("/")[-2]
    print("Person {} have {} images".format(dir_name, count))
```

- **CLASSIFICATION**

```
import keras
import numpy as np
import pickle
import glob
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import LabelBinarizer
class SoftMax():
    def __init__(self, input_shape, num_classes):
        self.input_shape = input_shape
        self.num_classes = num_classes
    def build(self):
        model.add(Dense(1024, activation='relu', input_shape=self.input_shape))
        model.add(Dropout(0.5))
        model.add(Dense(512, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(self.num_classes, activation='softmax'))
       model.compile(loss=keras.losses.sparse_categorical_crossentropy, optimizer=optimizer)
       return model
model = SoftMax(input_shape=(128,), num_classes=38)
model = model.build()
file = open('./output/train_embs.pickle', 'rb')
dict_embs = pickle.load(file)
embeddings = np.concatenate(dict_embs["embs"])
lb = LabelBinarizer()
labels = lb.fit_transform(dict_embs["labels"])
cv = KFold(n_splits = 5, random_state = 42, shuffle=True)
history = {'acc': [], 'val_acc': [], 'loss': [], 'val_loss': []}
for train_idx, valid_idx in cv.split(embeddings):
    X_train, X_val, y_train, y_val = embeddings[train_idx], embeddings[valid_idx],
labels[train_idx], labels[valid_idx]
```

```
        his = model.fit(X_train, y_train, batch_size=16, epochs=10, verbose=1)
        print(his.history['accuracy'])
        history['acc'] += his.history['accuracy']
        history['val_acc'] += his.history['val_accuracy']
        history['loss'] += his.history['loss']
        history['val_loss'] += his.history['val_loss']
    model.save("./models/classify_model.h5")
```

- **TRAINING AND EMBEDDING**

```
import numpy as np
import os
import cv2
import pickle
from src.utils import *
from glob import glob
model_path = r'C:\Source code\models/facenet_keras.h5'
model = load_model(model_path)
train_path = r'C:\Source code/aug_data/'
dir = glob(train_path + '*' )
for person_dir in dir:
   name = person_dir.split("\\")[-1]
   person_dir  = person_dir + "/*"
   imgs = glob(person_dir)
   for img in imgs:
      img_read = cv2.imread(img)
      img_read = cv2.cvtColor(img_read, cv2.COLOR_BGR2RGB)
      img_read = cv2.resize(img_read, (160, 160))
      a_person.append(img_read)
   img_list[name] = np.array(a_person)
   a_person = []
  for name, imgs in tqdm(img_list.items()):
   embs_ = calc_embs(model, imgs, 1)
   labels.extend([name] * len(embs_))
   embs.append(embs_)
   pass
dict = {}
dict["labels"] = labels
dict["embs"] = embs
f = open(r"C:\Source code/output/train_embs.pickle", "wb")
f.write(pickle.dumps(dict))
f.close()
print("{} people are embedded!".format(len(embs)))
```

```
print("Embedding done!")
```

- **FACE IDENTIFICATION**

```
import numpy as np
import os
import cv2
import pickle
import glob
from mtcnn import MTCNN
from sklearn.preprocessing import LabelBinarizer
from src.utils import *
detector = MTCNN()
face_model_path = './models/facenet_keras.h5'
face_model = load_model(face_model_path)
classify_model_path = './models/classify_model.h5'
classify_model = load_model(classify_model_path)
dict_file = open('./output/train_embs.pickle', 'rb')
dict_embs = pickle.load(dict_file)
labels = dict_embs["labels"]
lb = LabelBinarizer()
labels = lb.fit_transform(dict_embs["labels"])
softmax_thresh = 0.96
cosine_thresh = 0.8
#load webcam
cap = cv2.VideoCapture(0)
while 1:
    ret, img = cap.read()
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    bboxes = detector.detect_faces(img)
    # print(bboxes)
    if len(bboxes) != 0:
        for bboxe in bboxes:
            bbox = bboxe['box']
            bbox = np.array([bbox[0], bbox[1], bbox[0]+bbox[2], bbox[1]+bbox[3]])
            y = bbox[1] - 10 if bbox[1] - 10 > 10 else bbox[1] + 10
            #Box
            img2 = img[bbox[1]:bbox[3], bbox[0]:bbox[2]]
            color_box = (255, 0, 0)
```

```
            cv2.rectangle(img, (bbox[0]-10, bbox[1]-10), (bbox[2]+10, bbox[3]+10), color_box, 2)
            try:
                img2 = cv2.resize(img2, (160, 160))
                img2 = img2[np.newaxis]
                # embed = face_model.predict_on_batch(img2)
                embed = calc_embs(face_model, img2, 1)
                preds = classify_model.predict(embed)
                preds = preds.flatten()
                pred, prob = get_label_classify(preds, lb)
                print("Softmax method: Prob: {:.3f} Welcome to accessing your locker : {}".format(prob, pred))
                label_cos, prob_cos = most_similarity(np.concatenate(dict_embs["embs"]), embed, dict_embs["labels"])
                print("Cosine method: Prob: {:.3f} Welcome to accessing your locker: {}".format(prob_cos, label_cos))
                print("---------------------")
                #show result if 2 method have same predict
                if(prob > softmax_thresh and pred == label_cos and prob_cos > cosine_thresh):
                    #box of face
                    color_box = (0, 255, 0) #change color to green if face regco
                    cv2.rectangle(img, (bbox[0]-10, bbox[1]-10), (bbox[2]+10, bbox[3]+10), color_box, 2)
                    #text process
                    text = str(pred)
                    # scale = (round((bbox[2]+10+1 - bbox[0]-10-1)*0.0025)/2)+0.5
                    scale = (round((bbox[2]+10+1 - bbox[0]-10-1)*0.003)/2)+0.5
                    t_thickness = int(round(scale + 0.5))
                    t_size = cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX, fontScale=scale,
thickness=t_thickness)[0]
                    width_box = -(bbox[0]-10-1 - bbox[2]+10+1)
                    locate = int(round((width_box - t_size[0])/2))
                    #box and text
                    cv2.rectangle(img, (bbox[0]-10-1, bbox[1]-t_size[1]-25), (bbox[2]+10+1, bbox[1]-10), color_box, -1)
                    cv2.putText(img, text, (bbox[0]-10-1+locate+11, y-4-5), cv2.FONT_HERSHEY_SIMPLEX, scale)
                else:
                    print('Sorry! You are prohibited from utilizing the locker facilities.')
                    playsound.playsound(r"C:\Source code\alarm.wav")
            except:
                print("Error!\n")
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        cv2.imshow('Webcam',img)
```

ANNEXURE II

SCREENSHOTS

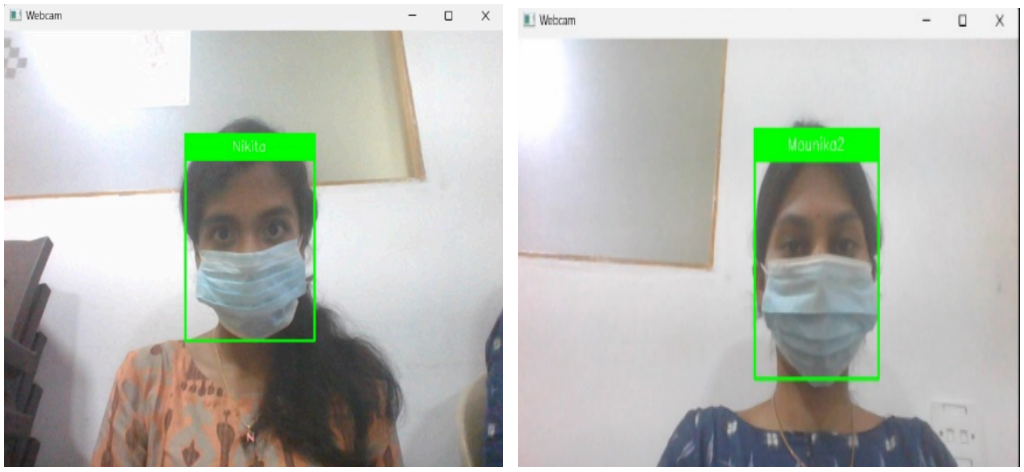


Fig. Identification of faces with mask

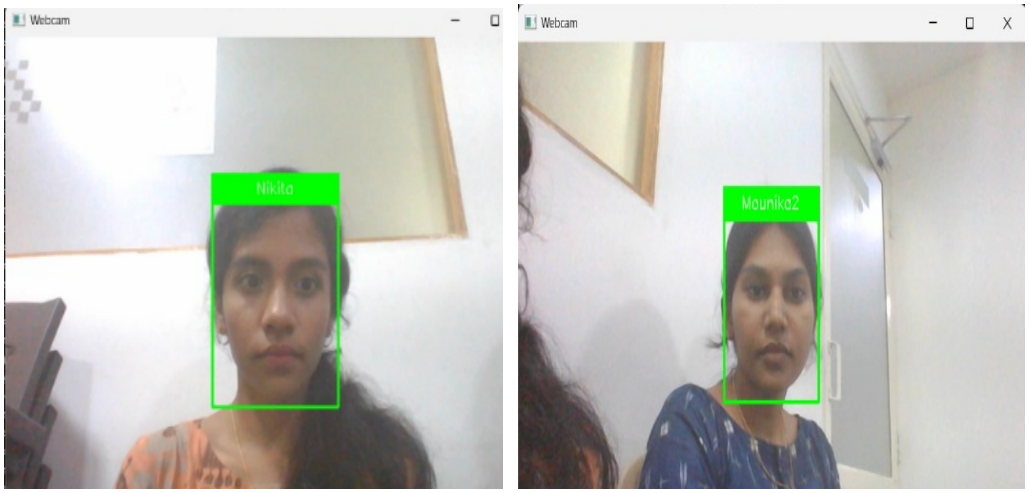


Fig. Identification of faces without mask

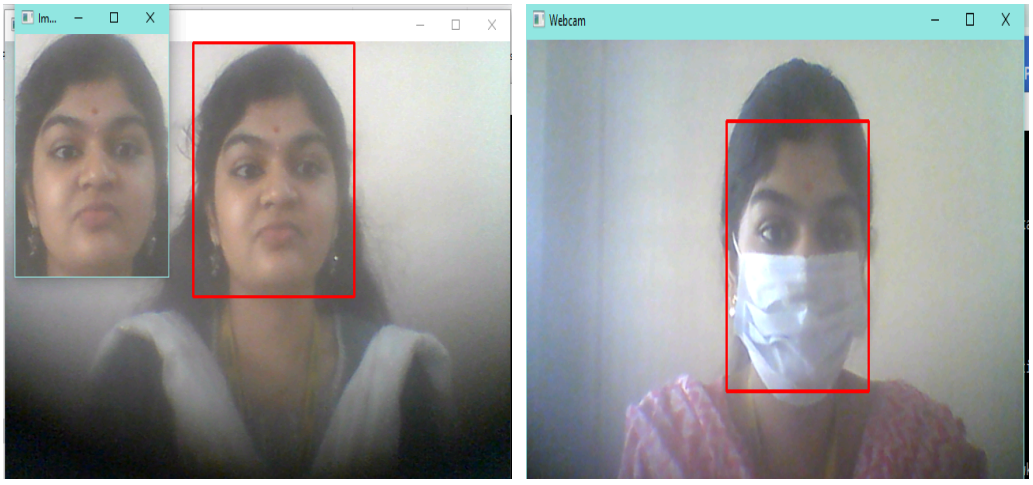


Fig. Untrained faces

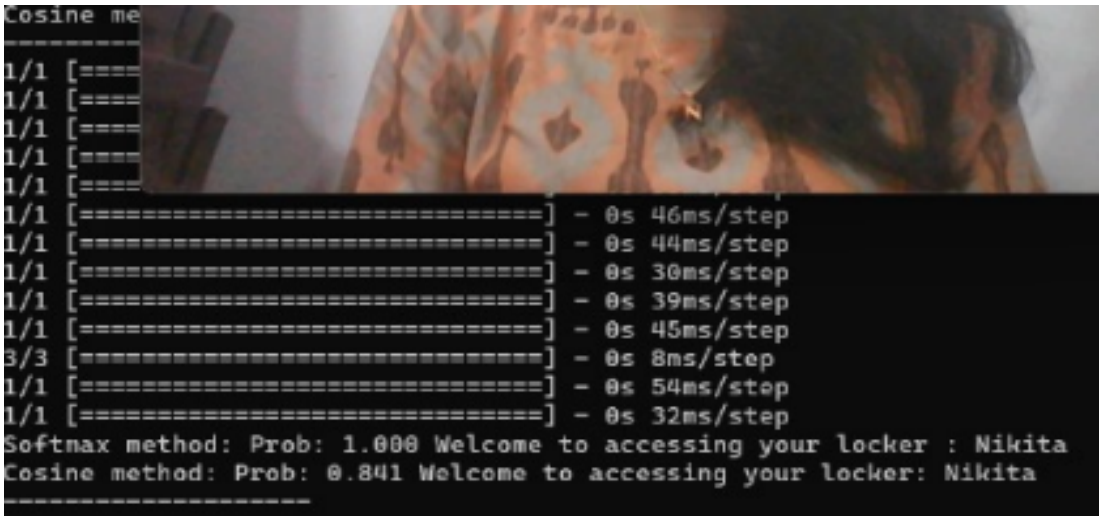


Fig. Softmax and Cosine Classifier

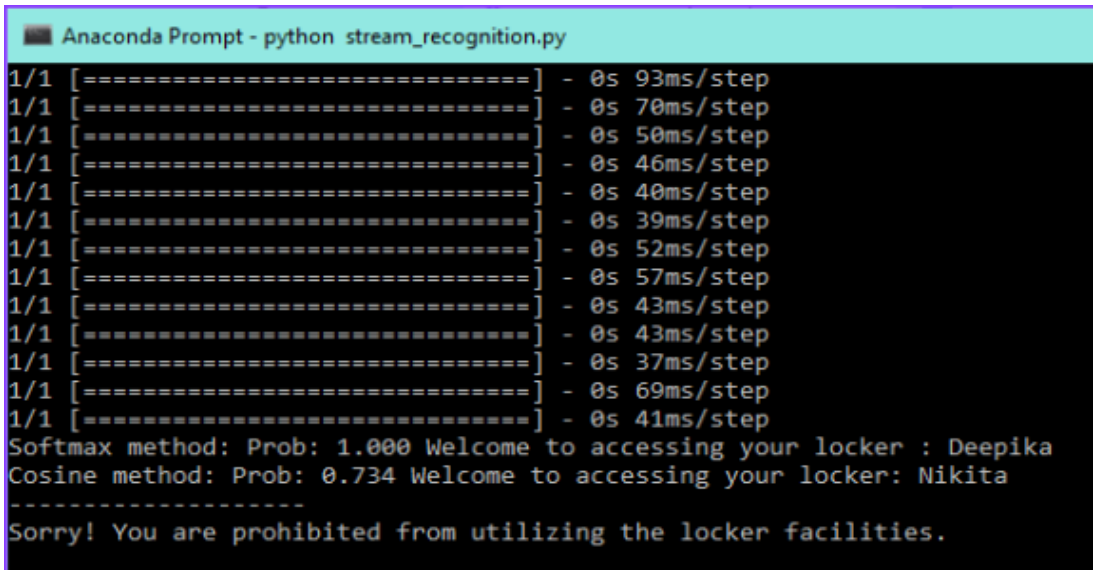


Fig. Restricted Access

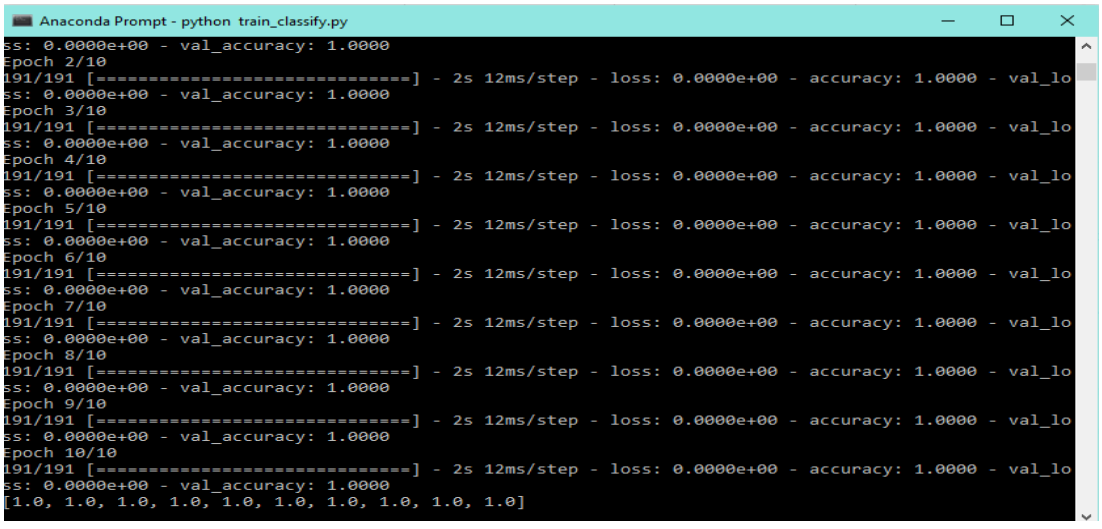
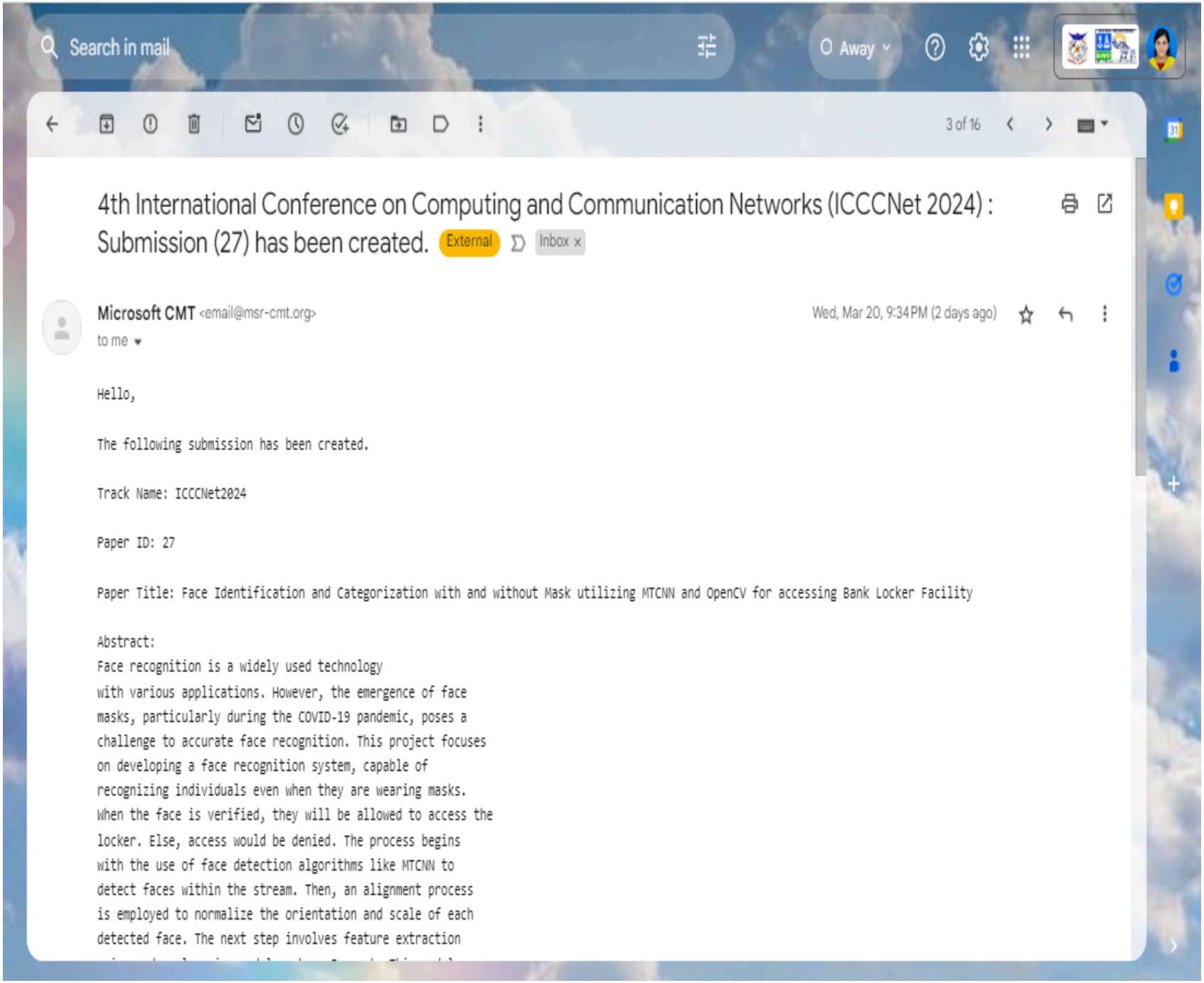


Fig. Calculation of Accuracy and Loss

PAPER PUBLICATION



BIBLIOGRAPHY

REFERENCES

[01] G. Singh, I. Gupta, J. Singh and N. Kaur, "Face Recognition using Open Source Computer Vision Library with Python," 10th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), Noida, India, 2022, pp. 1-6.

[02] A. Kumari Sirivarshitha, K. Sravani, K. S. Priya and V. Bhavani, "An approach for Face Detection and Face Recognition using OpenCV and Face Recognition Libraries in Python," 9th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2023, pp. 1274-1278.

[03] J. Zhang, "Online Face Recognition System Based on Convolutional Neural Network (CNN)," 2022 3rd International Conference on Computer Science and Management Technology (ICCSMT), Shanghai, China, 2022, pp. 292-295.

[04] D. R. Patel and R. K. Agrawal, "Face Mask Detection and Recognition using OpenCV and Keras," International Conference on the Emerging Trends in Information Technology and Engineering (ic-ETITE), Nagpur, India, 2021, pp. 1-5.

[05] S. Qi, X. Zuo, W. Feng and I. G. Naveen, "Face Recognition Model Based On MTCNN And Facenet," IEEE 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC), Tumkur, Karnataka, India, 2023, pp. 1-5.

[06] P. Jain and R. Roy, "Face Recognition with Mask Detection using Deep Learning and OpenCV", 5th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2021, pp. 442-446.

[07] Khan, Soumya & Sengupta, Diganta & Ghosh, Anupam & Chaudhuri, Atal, "MTCNN++: A CNN-based face detection algorithm inspired by MTCNN" , 2023, pp. 1-19.

[08] Wang Xiang, "The Research and Analysis of Different Face Recognition Algorithms", The International Conference on Computing Innovation and Applied Physics (CONF-CIAP), 2022, vol. 2386.

[09] Sun Y, Ren Z, Zheng W., "Research on Face Recognition Algorithms on Image Processing" , Comput Intell Neurosci, 2022.

[10] L. Li, X. Mu, S. Li and H. Peng, "A Review of Face Recognition Technology", in IEEE Access, vol. 8, pp. 139110-139120, 2023.

[11] G. Singh and A. K. Goel, "Face Detection and Recognition System using Digital Image Processing," 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore, India, 2023, pp. 348-352.

[12] P. J. Thilaga, B. A. Khan, A. A. Jones and N. K. Kumar, "Modern Face Recognition with Deep Learning", Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 2021, pp. 1947-1951.

[13] M. Jha, A. Tiwari, M. Himansh and V. M. Manikandan, "Face Recognition: Recent Advancements and Research Challenges," 13th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2022, pp. 1-6 .

[14] R. Sharma, V. K. Sharma and A. Singh, "A Review Paper on Facial Recognition Techniques," Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), Palladam, India, 2023, pp. 617-621.