1. **ABSTRACT**

This project focuses on developing a real-time face detection and recognition system using OpenCV and the Face-Recognition library. The system is designed to identify individuals from live camera feeds by comparing detected faces against a pre-existing database of known faces. Utilizing image processing techniques, the system first detects faces within the video stream and then encodes these faces into unique numerical representations, allowing for quick and accurate matching.The implementation leverages deep learning models for face encoding and recognition, ensuring robust performance even under varying lighting and angle conditions. The project aims to provide a scalable, cost-effective solution that can be easily integrated into applications like security systems, attendance tracking, and identity verification. Continuous learning capabilities, through dynamic database updates and error handling, allow the system to improve over time, adapting to new faces and enhancing recognition accuracy.

1. **INTRODUCTION**

# FACE DETECTION AND RECOGNITION SYSTEM

Face detection and recognition systems have become increasingly relevant in modern applications, ranging from security surveillance to user authentication in smart devices. This project focuses on building a face detection and recognition system utilizing open-source libraries such as OpenCV and Face-Recognition, which process live camera feeds in real-time to detect and recognize human faces.

The core functionality of a face recognition system involves two key processes: detecting a face in an image or video stream and recognizing or identifying the person based on stored data. In recent years, advancements in machine learning, particularly deep learning, have significantly improved the performance of these systems, making them both faster and more reliable.

This face detection and recognition system can be deployed in various real-world applications, including security systems, attendance monitoring, identity verification, and access control. Its real-time processing capabilities allow for efficient recognition without the need for high-end hardware or expensive equipment. The project aims to demonstrate the integration of computer vision techniques with accessible consumer-grade hardware and open-source software to achieve a reliable and scalable solution.

This project implements a real-time face detection and recognition system designed for practical applications such as surveillance, attendance monitoring, and access control. It leverages open-source libraries such as OpenCV for detecting faces and specialized face recognition tools to identify individuals from a database of known faces. The system uses image encoding to match new face detections with existing records, efficiently identifying and labeling the recognized individuals.

# OVERVIEW OF FACE DETECTION AND RECOGNITION

The human face is one of the most unique identifiers and plays a critical role in human recognition systems utilize this inherent uniqueness to automate identification processes across various applications. In recent years, these systems have become increasingly prevalent in areas such as security, surveillance, and identity verification.

Face Detection is the first stage of the recognition process, where the system identifies and isolates faces from images or video streams. This is done using specialized algorithms that can accurately locate facial features, even in complex environments or low-light conditions. Once a face is detected, the system extracts the region of interest, enabling further processing.

Face recognition takes this process a step further by comparing the detected face against a database of pre-recorded faces. The system uses encoding techniques to generate unique facial "signatures" or embeddings, which are stored for future comparisons. When a face is detected, it is encoded and matched against the existing database. If a match is found, the system identifies the person and labels the face accordingly.

This project presents a real-time face detection and recognition system that uses open-source tools like OpenCV and Face-Recognition libraries. The system is designed to capture live video streams, detect faces, and recognize known individuals by comparing the detected faces with those stored in the system’s database. Upon successful recognition, the person’s name is displayed alongside their face in the video feed, providing a fast and efficient solution for real-time applications.

By leveraging accessible hardware and open-source software, this system offers a cost-effective and scalable solution suitable for various environments. Whether used for enhancing security, managing attendance, or or verifying identities, this project demonstrates the practical implementation of face detection and recognition technology.

* 1. **APPLICATION**
     + **Surveillance and Monitoring:** This system is highly effective in enhancing security in public and private spaces. By detecting and recognizing faces in real-time, it helps security personnel monitor and track individuals in high-risk areas like airports, shopping malls, and other public spaces. The system can immediately alert security officials to the presence of blacklisted or suspicious individuals, reducing the risk of unauthorized access or criminal activity.
     + **Access Control:** Face recognition system offer a secure and convenient alternative to traditional access control methods. This system can be integrated into secure entry points, allowing authorized personnel to gain access based on facial recognition. It eliminates the need for keycards or PIN codes, reducing the risk of unauthorized entry and ensuring a seamless and hands-free experience.
     + **Automated Attendance:** In workplaces and educational institutions, this system can automate the process of attendance management. By capturing the faces of employees or students, the system records attendance without the need for manual intervention. This not only reduces administrative workload but also ensures greater accuracy in attendance tracking.

# OBJECTIVE

* Provide secure identification.
* Automate the Identification process.
* Facilitate Cost-Effective Deployment.
* Scalable and Flexible System.
* Improve Usability and Efficiency.

# SYSTEM STUDY

* 1. **EXISTING SYSTEM**

Traditional face recognition system typically rely on static image databases or manual identification, where individuals are identified by matching their face images with existing records. These systems are often used in security and surveillance but come with several limitations. For example, many existing systems lack the capability for **real-time processing**; they involve post-capture analysis where images or video feeds are processed after the fact. Additionally, these systems usually depend on **proprietary software** and hardware, making them expensive to deploy and maintain. Many of these solutions are not easily customizable or scalable and require **high-performance servers** and specialized surveillance equipment. Furthermore, updating the face database often requires manual interventions, and integration with other systems can be cumbersome.

# PROPOSED SYSTEM

The proposed system introduces a more **cost-effective** and **real-time** solution for face recognition by leveraging **open-source** libraries such as OpenCV and SimpleFacerec. Unlike traditional systems, this solution uses **live camera feeds** and performs real-time face detection and recognition, automatically identifying known faces without delay. It encodes face images from a folder and can easily update the face database with new faces. The system is designed to run on **standard consumer hardware** like laptops or desktops with basic webcams, eliminating the need for expensive surveillance cameras or dedicated servers. This makes it highly scalable and accessible for smaller projects or individual developers. Furthermore, the use of Python and open-source tools makes the system easy to implement, maintain, and customize for various applications like security, attendance systems, or identity verification. The **lightweight design** ensures that it can be deployed in a variety of real-time applications without significant hardware investments.

# SYSTEM CONFRIGURATION

* 1. **HARDWARE REQUIREMENT**

Processor type : Intel i5 Core.Speed : 2.50GHZ.

RAM : 16 GB.

Hard disk : 200 GB.

Camera : Built-in laptop camera.

# SOFTWARE REQUIREMENT

O/S : Windows11.

Language : Python.

1. **SYSTEM DESIGN**
   1. **PROPOSED WORKFLOW**

**Data Collection**

**Image pre-processing**

**Face Detection**

**Face Encoding**

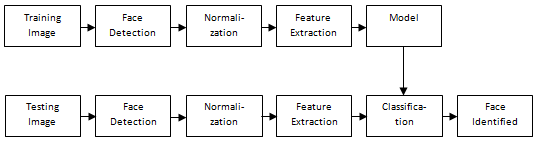
**Face Matching and Recognition**

**Real-Time Display**

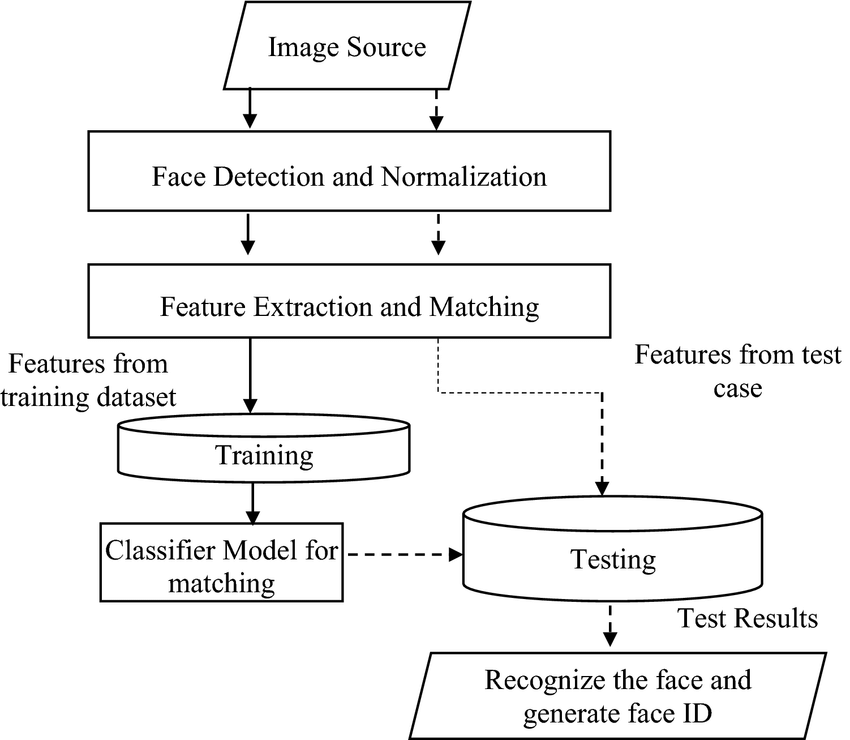
**Database Update**

**Error Handling and Retraining**

**Analysis and Output**

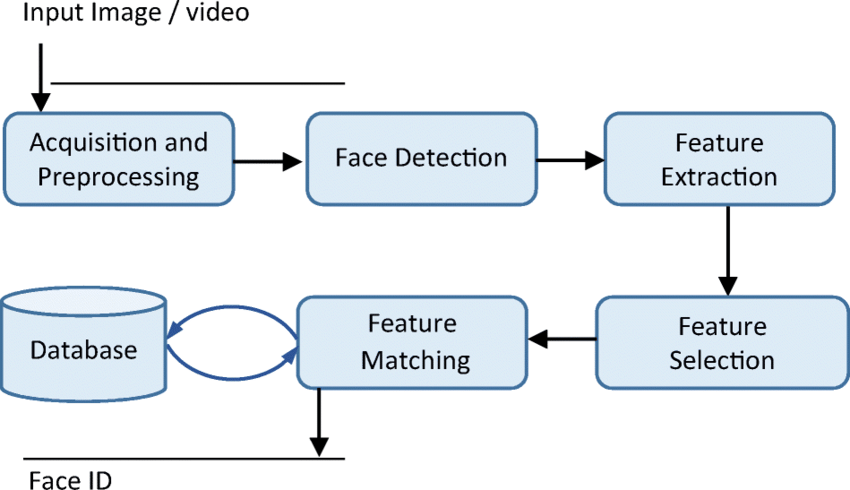


* 1. **PROCESS OF DETECTING FACE AND RECOGNIZING**



**5.3 BLOCK DIAGRAM OF PROPOSED HUMAN FACE RECOGNITION**

**SYSTEM**



**5.4 FLOW CHART OF DETECTING FACE AND RECOGNIZING**

**6. SYSTEM SPECTIFICATION**

* 1. **SOFTWARE DESCRIPTION: PYTHON**

Python was created by Guido van Rossum, and released in 1991.Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

**6.2 SOFTWARE DESCRIPTION:**

1. **FREE AND OPEN SOURCE:**

Python Is Developed Under an OSi-approved License. Python is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

# EASY TO CODE:

Python is a high level programming language. Python is very easy to learn the language as compared to other languages like C, C#, JavaScript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

# EASY TO READ:

Learning Python is quite simple. As was already established, Python’s syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

1. **OBJECT-ORIENTED LANGUAGE:**

One of the key features of python is object oriented programming language. Python supports object-oriented language and concepts of classes, object encapsulation,etc.

# GUI PROGRAMMING SUPPORT:

Graphical User interfaces can be made using a module such as PvQt5,PyQt4, wxPython.PyQt5 is the most popular option for creating graphical apps with Python.

# HIGH-LEVEL LANGUAGE:

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

# LARGE COMMUNITY SUPPORT:

Python has gained popularity over the years. Our questions are constantly answered by the enormous StackOverflow community. These websites have already provided answers to many questions about Python, so Python users can consult them as needed.

# EASY TO DEBUG:

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program’s issues once you understand how to interpret Python’s error traces. Simply by glancing at the code, you can determine what it is designed to perform.

# PYTHON IS A PORTABLE LANGUAGE:

Python language is also a portable language. For example, if we have Python code for Windows and if Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called byte code.

# LARGE STANDARD LIBRARY

Python has a large standard library that provides a rich set of modules and functions so you do not want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

# PYTHON IS AN INTEGRATED LANGUAGE:

Python is also an Integrated language because we can easily integrate Python with other languages like C, C++, etc.

# INTERPRETED LANGUAGE:

There are many libraries present in Python such as regular expressions, unit-testweb browsers, etc.

# DYNAMICALLY TYPED LANGUAGE

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don’t need to specify the type of variable.

# FRONTEND AND BACKEND DEVELOPMENT

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like JavaScript. Backend is the strong forte of Python it’s extensively used for this work cause of its frameworks like Djangoand Flask.

# ALLOCATING MEMORY DYNAMICALLY

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

# 6.3 WHY PYTHON?

Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick
* Python can be treated in a procedural way, an object-oriented way or a functional way.

.

# 6.4 FEASIBILITY STUDY

The feasibility study is carried out to test whether the proposed system is worth being implemented. The proposed system will be selected if it is best enough in meeting the performance requirements. The feasibility carried out mainly in three sections namely.

* Economic Feasibility.
* Technical Feasibility.
* Behavioural Feasibility.

# ECONOMIC FEASIBILITY:

Economic analysis is the most frequently used method for evaluating effectiveness of the proposed system. More commonly known as costbenefit analysis. This procedure determines the benefits and saving that are expected from the system of the proposed system. The hardware in system department if sufficient for system development.

# TECHNICAL FEASIBILITY:

This study centre around the system’s department hardware, software and to what extend it can support the proposed system department is having the required hardware and software there is no question of increasing the cost of implementing the proposed system. The criteria, the proposed systemis technically feasible and the proposed system can be developed with the existing facility.

# BEHAVIOURAL FEASIBILITY:

People are inherently resistant to change and need sufficient amount of training, which would result in lot of expenditure for the organization. The proposed system can generate reports with day-to-day information immediately at the user’s request, instead of getting a report, which doesn’t contain much detail.

# 6.5 CONVOLUTIONAL NETURAL NETWORK:

Convolutional Neural Network (CNN) are the backbone of the Face Detection and Recognition system, significantly enhancing its ability to accurately identify faces in real-time. CNNs are specifically designed for processing visual data, making them ideal for tasks like facial recognition due to their proficiency in extracting and analyzing features from images. In the initial phase, CNNs work by automatically scanning input images to identify key facial features such as the eyes, nose, mouth, and overall face structure. These features are extracted through a series of convolutional layers that detect patterns in the pixel data. Each layer in the CNN focuses on different aspects of the image, gradually capturing more complex features as the data passes through deeper layers. The end result of this process is a feature map, which provides a detailed representation of the face.

Once the facial features are identified, CNNs convert these into a compact numerical format known as face embeddings. This transformation is crucial because it allows the system to represent each face as a unique vector of numbers, encapsulating the distinctive characteristics of the individual. These embeddings are then used to compare and match faces by calculating the similarity between the vector representations. This approach makes face recognition efficient and scalable, as it can handle large datasets of known faces. The CNN-based face encoding is robust enough to handle variations caused by different lighting conditions, facial angles, expressions, or even partial occlusions, thereby increasing the accuracy and reliability of the system in diverse real-world scenarios.

In the recognition phase, the encoded face vectors generated by the CNN are compared with the known face encodings stored in a database. The system uses distance metrics like Euclidean distance to measure how similar the new face encoding is to the existing ones in the database.

* **Input Layer**-It takes in the raw pixel value of input image.
* **Convolutional Layer**- It is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel to generate a feature map Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.
* **Activation Layer**- It produces a single output based on the weighted sum of inputs.
* **Pooling Layer**-Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling (also called sub sampling or down sampling) reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:
  + **Max Pooling**– taking the largest element in the feature map
  + **Average Pooling**-taking the average of elements in the feature map
  + **Sum Pooling**–taking the sum of all elements in the feature map
* **Fully Connected Layer**-The layer we call as FClayer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network. The feature map matrix will be converted as column vector (x1, x2, x3 …). With the fully connected layers, we combined these features together to create a model. For classifying input image into various classes based on training set.
* **Dropout Layer**-It prevents nodes in a network from co-adapting to each other.

# ADVANTAGES:

1. It is considered as the best ml technique for image classification due to high accuracy.
2. Image pre-processing required is much less compared to other algorithms.
3. It is used overfeed forward neural networks as it can be trained better incase of complex images to have higher accuracies.
4. It reduces images to a form which is easier to process without losing features which are critical for a good prediction by applying relevant filters and reusability of weights
5. It can automatically learn to perform any task just by going through the training data

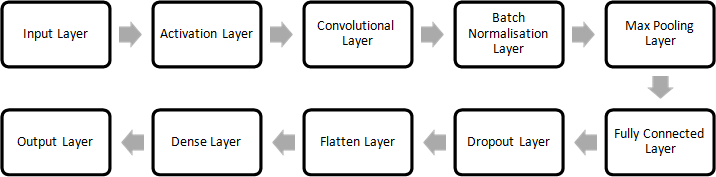
i.e. there no need for prior knowledge

1. There is no need for specialized hand-crafted image features like that incase of SVM, Random Forest etc.

# DISADVANTAGES:

1. It requires a large training data.
2. It requires appropriate model.
3. It is time consuming.
4. It is a tedious and exhaustive procedure.
5. While convolutional network shave already existed for a longtime, their success was limited due to the size of the considered network.

**Solution** – Transfer learning for in adequate data which will replace the last fully connected layer with pre-trained ConvNet with new fully connected layer.



**A DIAGRAM OF A MODEL TRAINED FROM SCRATCH USING ARCHITECTURE**

1. **IMPLEMENTATION**
   1. **MODULES :**
2. Data Collection
3. Image pre-processing
4. Face Detection
5. Face Encoding
6. Face Matching and Recognition
7. Real-Time Display
8. Database Management
9. Error Handling and Retraining
10. Analysis and Output

# MODULES DESCRIPTION :

## Data Collection :

Before any analysis can take place, images must be pre-processed to enhance their quality and ensure they are This module is responsible for gathering input data necessary for the face detection and recognition process. It can source data from live camera feeds or pre-existing databases of known faces. This module captures real-time video streams from connected cameras, allowing for continuous monitoring and face detection, and retrieves static images from local storage, which can include previously stored images of individuals for the recognition process. It ensures that the data collected is in a suitable format for further processing, maintaining quality and consistency.

## Image pre-processing :

Before any analysis can take place, images must be pre-processed to enhance their quality and ensure they are suitable for detection algorithms. This module prepares the input images for analysis by resizing images to a standard resolution to maintain uniformity and improve processing speed, normalizing pixel values to adjust for variations in lighting and contrast, which enhances the effectiveness of detection algorithms, and converting images to grayscale if necessary, simplifying the data and reducing computational load while preserving essential facial features.

## Face Detection :

The heart of the face detection and recognition system, this module employs sophisticated algorithms algorithms to identify and isolate faces from images or video streams. It utilizes methods such as Haar cascades or deep learning-based methods to accurately locate faces. This module processes images to extract regions of interest (ROIs) where faces are detected, marking their locations with bounding boxes. It works in real-time to ensure quick detection and is robust enough to handle various conditions, including different lighting and angles.

## Face Encoding :

The pixel values of the images are normalized to be between 0 and 1, which is a common preprocessing step for neural networks.After detecting faces, this module converts them into unique numerical representations, known as embeddings. These embeddings capture the essential features of the face for later comparison. It employs deep learning techniques, particularly Convolutional Neural Networks (CNNs), to encode facial features into a compact vector format. Each face is transformed into a fixed-length array of numbers, allowing allowing for efficient comparison and matching against a database of known faces. This module ensures that similar faces yield similar embeddings, facilitating accurate recognition.

## Face Matching and Recognition :

This module is crucial for identifying individuals based on the encoded facial data. It compares the newly generated embeddings with those stored in the database to determine if a match exists. It uses distance metrics (like Euclidean distance) to assess how closely the newly encoded face matches existing embeddings.

## Real-Time Display :

This module is responsible for providing visual feedback on the detection and recognition process, ensuring that results are displayed immediately to users. It overlays bounding boxes around detected faces in the video feed, enhancing user understanding of what the system recognizes, and displays names or labels next to recognized faces, facilitating immediate identification for security personnel or users. The display updates continuously to reflect ongoing recognition efforts.

## Database Management :

Efficient database management is vital for storing and updating face encodings and associated identities. This module oversees the organization and retrieval of data, maintaining a database of known face encodings that allows for quick access during the recognition process. It adds new face encodings when a new individual is recognized, ensuring that the database remains current and accurate. This module supports efficient data retrieval and management as the system scales.

## Error Handling and Retraining :

This module plays a significant role in improving the accuracy and reliability of the system by managing errors and refining the model based on real-world usage.

## Analysis and Output :

Finally, this module compiles the results of the detection and recognition processes, generating actionable insights from the gathered data..

# SYSTEM TESTING AND MAINTANENCE

* 1. **SYSTEM TESTING**

To perform system testing for Face Recognition project using Python and a Convolutional Neural Network (CNN), you can follow these steps. A system testing aim to ensure that your application works as expected and meets the desired functionality and performance criteria:

# MODEL VALIDATION :

* + Include cross-validation techniques to assess the model's performance on different subsets of data.
  + Use a confusion matrix to analyze the types of errors the model makes, providing insights into areas that require improvement.

# ERROR HANDLING IMPROVEMENTS :

* + Test the system's ability to handle corrupted image files, unsupported formats, or unusual input.
  + Implement more robust error logging to capture issues that occur during testing for easier troubleshooting.

# SCALABILITY TESTING :

* + Simulate larger datasets to test the system's performance under increased load, ensuring it scales well.
  + Evaluate how the system handles simultaneous face recognition requests from multiple users.

# REAL – WORLD TESTING :

* + Test the system with images taken under different lighting conditions, angles, and varying face orientations to ensure robust face detection.
  + Include tests with images of individuals wearing glasses, hats, or masks to check the system's adaptability.

# PERFORMANCE METRICS :

* + Measure metrics like precision, recall, and F1-score to get a comprehensive view of the model's accuracy beyond just a single score.
  + Conduct speed tests to determine how quickly the system processes and identifies faces in real-time.

# SECURITY ENHANCEMENTS :

* + Conduct penetration testing to ensure that the system is protected against vulnerabilities like spoofing or unauthorized access.
  + Implement tests to verify that facial data and recognition results are stored securely, adhering to data privacy standards.

# USABILITY TESTING :

* + Ensure that the system is user-friendly by conducting usability testing with actual end-users to gather feedback on the interface and overall user experience.

# INTEGRATION TESTING :

* + If the system will be integrated with other software or hardware components (e.g., security systems, databases), perform integration testing to ensure seamless communication and data exchange between components.

# REGRESSION TESTING :

* + If any updates or bug fixes are made, conduct regression testing to ensure new changes do not negatively impact the existing features.

# CONCLUSION :

In conclusion, the Face Detection and Recognition system developed using OpenCV and the Face-Recognition library provides a practical, real-time solution for identifying individuals in live video streams. Leveraging deep learning techniques and open-source tools, this project demonstrates a scalable approach suitable for applications like surveillance, attendance monitoring, and access control. By using accessible hardware and implementing continuous learning through error handling and retraining, the system evolves to meet new challenges. This project lays a solid foundation for future enhancements in security and authentication technologies.

# REFERENCES :

# BOOK REFERENCE

# [1] Bishop, C. M. (2006). "Pattern Recognition and Machine Learning." Springer.

# [2] Bradski, G., & Kaehler, A. (2008). "Learning OpenCV: Computer Vision with the OpenCV Library." O'Reilly Media.

# [3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). "Deep Learning." MIT Press.

# [4] Raschka, S., & Mirjalili, V. (2019). "Python Machine Learning" (3rd Edition). Packt Publishing.

# [5] Szeliski, R. (2010). "Computer Vision: Algorithms and Applications." Springer.

# WEB REFERENCE

**[1]** Deep Learning for Face Recognition by Towards Data Science’

<https://towardsdatascience.com/deep-learning-for-face-recognition-a3e229b2c3c2>

**[2]** Google AI Blog on Facial Recognition

<https://ai.googleblog.com/>

**[3]** IBM Watson Visual Recognition

<https://www.ibm.com/watson/services/visual-recognition/>

**[4]** Medium - Face Recognition with Deep Learning in Python

<https://medium.com/tag/face-recognition>

**[5]** Microsoft Azure Face API Documentation

<https://docs.microsoft.com/en-us/azure/cognitive-services/face/>

# 

# 11. APPENDIX

**11.1 SAMPLE CODE :**

**MAIN.PY**

import cv2

from simple\_facerec import SimpleFacerec

#Encode faces from a folder

sfr=SimpleFacerec()

sfr.load\_encoding\_images("C:/project/images")

#Load Camera

cap = cv2.VideoCapture(0)

while True:

    ret,frame=cap.read()

    #Detect Faces

    face\_locations, face\_names=sfr.detect\_known\_faces(frame)

    for face\_loc ,name in zip(face\_locations,face\_names):

        y1,x2,y2,x1=face\_loc[0],face\_loc[1],face\_loc[2],face\_loc[3]

        cv2.putText(frame, name , (x1, y1 - 10), cv2.FONT\_HERSHEY\_DUPLEX, 1,(0,0,200), 2)

        cv2.rectangle(frame, (x1,y1) , (x2,y2) , (0,0,200) , 2)

    cv2.imshow("Frame", frame)

    key=cv2.waitKey(1) & 0xff

    #Break the loop when the 'q' key is pressed

    if key == ord('q'):

        break

cap.release()

cv2.destroyAllWindows()

# SIMPLEFACEREC.PY

import face\_recognition

import cv2

import os

import glob

import numpy as np

class SimpleFacerec:

    def \_\_init\_\_(self):

        self.known\_face\_encodings = []

        self.known\_face\_names = []

        # Resize frame for a faster speed

        self.frame\_resizing = 0.25

    def load\_encoding\_images(self, images\_path):

        # Load Images

        images\_path = glob.glob(os.path.join(images\_path, "\*.\*"))

        print("{} encoding images found.".format(len(images\_path)))

        # Store image encoding and names

        for img\_path in images\_path:

            img = cv2.imread(img\_path)

            rgb\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

            # Get the filename only from the initial file path.

            basename = os.path.basename(img\_path)

            (filename, ext) = os.path.splitext(basename)

            # Get encoding

            img\_encoding = face\_recognition.face\_encodings(rgb\_img)[0]

            # Store file name and file encoding

            self.known\_face\_encodings.append(img\_encoding)

            self.known\_face\_names.append(filename)

        print("Encoding images loaded")

    def detect\_known\_faces(self, frame):

        small\_frame = cv2.resize(frame, (0, 0), fx=self.frame\_resizing, fy=self.frame\_resizing)

        # Find all the faces and face encodings in the current frame of video

        # Convert the image from BGR color (which OpenCV uses) to RGB color (which face\_recognition uses)

        rgb\_small\_frame = cv2.cvtColor(small\_frame, cv2.COLOR\_BGR2RGB)

        face\_locations = face\_recognition.face\_locations(rgb\_small\_frame)

        face\_encodings = face\_recognition.face\_encodings(rgb\_small\_frame, face\_locations)

        face\_names = []

        for face\_encoding in face\_encodings:

            # See if the face is a match for the known face(s)

            matches = face\_recognition.compare\_faces(self.known\_face\_encodings, face\_encoding)

            name = "Unknown"

            # # If a match was found in known\_face\_encodings, just use the first one.

            # if True in matches:

            #     first\_match\_index = matches.index(True)

            #     name = known\_face\_names[first\_match\_index]

            face\_distances = face\_recognition.face\_distance(self.known\_face\_encodings, face\_encoding)

            best\_match\_index = np.argmin(face\_distances)

            if matches[best\_match\_index]:

                name = self.known\_face\_names[best\_match\_index]

            face\_names.append(name)

        # Convert to numpy array to adjust coordinates with frame resizing quickly

        face\_locations = np.array(face\_locations)

        face\_locations = face\_locations / self.frame\_resizing

        return face\_locations.astype(int), face\_name

* 1. **OUTPUT :**

