**FACE DETECTION**

 A Project Report

                        Submitted in the partial fulfillment of the

                          requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**In**

**DEPARTMENT OF COMPUTER SCIENCE ENGINNERING AND INFORMATION TECHNOLOGY**

**By**

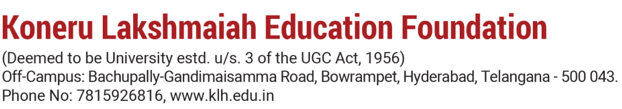
**Sujeeth Godavarthi- 2320090080**

**N V V S S Jaikanth Kamisetti- 2320090050**

Under the Esteemed Guidance of

**Dr.Sriram Murthy**





**K L (Deemed to be) University**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**



**Declaration**

The Project Report entitled “**Face Detection** “is a record of Bonafide work of **Sujeeth Godavarthi-2320090080, N V V S S Jaikanth Kamisetti-2320090050, Parthsarathi kurnSai -2320030379, Jaykar modumudi-2320090055** submitted in partial fulfillment for the award of B. Tech in Computer  Engineering to the K L University. The results embodied in this report have not been copied from any other departments/University/Institute.

Sujeeth Godavarthi  - 2320090080

N V V S S Jaikanth Kamisetti-2320090050

**K L (Deemed to be) University**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**



**Certificate**

This is certify that the project based report entitled “**Face Detection**” is a bonafide work done and submitted by **Sujeeth Godavarthi(2320090080), N V V S S Jaikanth Kamisetti(2320090050)**,**Parthsarathkurni Sai (2320030379), Jaykar modumudi (2320090055)** in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in Department of Computer Science Engineering, K L (Deemed to be University), during the academic year **2024-2025.**

**Signature of the Supervisor**

**Signature of the HOD                                               Signature of the External Examiner**

**ACKNOWLEDGEMENT**

The success in this project would not have been possible but for the timely help and guidance rendered by many people. Our wish to express my sincere thanks to all those who has assisted us in one way or the other for the completion of my project.

Our greatest appreciation to my guide, **Dr.Sreeram Murthy**, Department of Computer Science which cannot be expressed in words for his tremendous support, encouragement and guidance for this project.

We express our gratitude to **Mr. Khaja Shareef**, Head of the Department for Computer Science and Information technology for providing us with adequate facilities, ways and means by which we are able to complete this project based Lab.

We thank all the members of teaching and non-teaching staff members, and also who have assisted me directly or indirectly for successful completion of this project. Finally, I sincerely thank my parents, friends and classmates for their kind help and cooperation during my work.

Sujeeth Godavarthi-  2320090080

                                                                                 N V V S S Jaikanth Kamisetti-2320090050

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Contents** |  |
| 1 | Abstract |  |
|  |  |  |
| 2 | Introduction |  |
| 3 | Literature survey |  |
| 4 | Client meetings |  |
| 5 | Hardware and Software requirements |  |
| 6 | Implementation |  |
| 7 | Experimentation and Code |  |
| 8 | Results |  |
| 9 | Conclusion |  |
| 10 | References |  |

**ABSTRACT**

This Face Detection project utilizes artificial intelligence (AI) and machine learning (ML) to accurately identify and locate human faces within images. Developed in Python using the PyCharm IDE, the project employs supervised learning techniques on a comprehensive dataset of facial images, enabling the model to recognize and detect facial features under various challenging conditions, including diverse lighting, orientations, and facial expressions.

**Key elements of the project include**:

->**Methodology**: The project leverages powerful libraries such as OpenCV for efficient image processing, Dlib or Face\_recognition for precise feature mapping, and NumPy for data management and processing.

-> **Training Process**: Labeled face data is used to refine the model, enabling it to distinguish between face and non-face regions with high precision. By iteratively training on diverse images, the model learns to generalize effectively, enhancing its adaptability to real-time face detection scenarios.

->**Objectives**: This project aims to deliver a robust and scalable face detection system, serving as a foundation for future applications in:

- Security systems

- Automated attendance tracking

- Personalized user authentication

->**Conclusion**: The project presents a valuable tool for researchers and developers, providing a strong base for more advanced face recognition applications. Its adaptable framework supports further development towards comprehensive face recognition solutions, making it a potential asset in security and user verification domains.

**INTRODUCTION**

In recent years, facial recognition has emerged as a pivotal technology in various applications, from security and surveillance to user authentication and automated attendance systems. This project explores the implementation of a robust Face Recognition system that applies artificial intelligence (AI) and machine learning (ML) techniques to identify and verify individuals based on their facial features. Developed in Python using the PyCharm IDE, this project aims to build an efficient and scalable recognition system with high accuracy and adaptability to real-world conditions.

**Project Structure and Key Components**

The project is structured to streamline the face recognition process, separating distinct functionalities into dedicated scripts and directories, as follows:

- >**align\_faces.py**: Proper face alignment is essential for consistent recognition. This script preprocesses images by aligning faces to ensure they are centered and normalized, improving the model's recognition accuracy.

- >**generate\_embeddings.py**: This module is crucial for creating facial embeddings — unique numerical representations of each face. By processing labeled images in the dataset, it generates embeddings using ML models that capture distinct facial features, enabling precise comparisons during recognition.

- >**main.py**: Serving as the primary entry point, this script orchestrates the entire workflow. It integrates the preprocessing, embedding generation, and recognition functions, making the system modular and easier to test, debug, and expand.

- >**recognize.py**: This script performs real-time face recognition by comparing new face inputs against the precomputed embeddings. Leveraging libraries like OpenCV and Dlib, it achieves reliable feature extraction and matching, facilitating swift and accurate identification.

- >**requirements.txt**: A file specifying all necessary dependencies, including OpenCV, Dlib, and NumPy, ensures that the project environment can be replicated easily, supporting consistency across various platforms.

- >**data**: This folder contains the dataset of labeled images used for training the model. A diverse set of images allows the model to handle variations in lighting, orientation, and facial expressions.

- >**embeddings**: The embeddings folder stores the precomputed facial embeddings, enabling quick access for matching and reducing computational load during real-time recognition.

- >**logs**: The logging system records events and errors encountered during execution, supporting debugging and performance monitoring.

**Project Objectives and Scope**

The primary objective of this project is to develop a reliable and scalable face recognition system that can be adapted for various applications. By combining AI and ML with image processing, this project lays the groundwork for a sophisticated recognition model capable of functioning in real-world environments. Specific goals include:

- >**Accuracy and Precision**: Through supervised learning, the model is trained on diverse facial images, achieving high recognition accuracy despite variations in conditions.

- >**Scalability**: The modular structure allows for the integration of additional features or functionalities, making it a flexible base for future enhancements.

- >**Real-Time Performance**: With optimized embeddings and efficient comparison algorithms, the system is capable of performing recognition in real time, which is essential for applications in security and surveillance.

**Potential Applications**

The developed face recognition framework has potential applications in numerous domains, such as:

- >**Security and Surveillance**: Automated facial recognition can enhance security measures by identifying authorized or unauthorized individuals in real time.

- >**Automated Attendance Systems**: The system can streamline attendance tracking in schools, workplaces, and events, eliminating manual processes.

- >**User Authentication**: Facial recognition offers a secure and user-friendly method for authentication in applications like mobile devices, banking, and personalized services.

This project sets the stage for more complex face recognition systems by establishing a reliable, efficient, and adaptable framework. It is a valuable tool for researchers and developers looking to build upon the base model to create advanced applications in the growing field of computer vision

**LITERATURE SURVEY**

Facial recognition has seen extensive research and development over recent decades, with significant advancements driven by improvements in artificial intelligence, machine learning, and computer vision. This literature survey provides an overview of the foundational methods and current technologies that have influenced the design and functionality of this project.

**1. Historical Approaches to Face Detection and Recognition**

Early face detection techniques relied on traditional computer vision methods such as the **Viola-Jones algorithm** (Viola & Jones, 2001), which employed Haar-like features for rapid face detection in real-time. While effective, these methods had limitations in handling variations in lighting, pose, and occlusion. Later, **Eigenfaces** (Turk & Pentland, 1991) and **Fisherfaces** introduced principal component analysis (PCA) and linear discriminant analysis (LDA) to extract facial features, but their performance degraded under challenging conditions. These approaches laid the groundwork for more complex, feature-based, and learning-based methods used in modern systems.

**2. Deep Learning for Face Recognition**

Recent advancements in deep learning have revolutionized face recognition systems. Convolutional Neural Networks (CNNs) have been particularly successful due to their ability to learn hierarchical features from raw pixel data. **DeepFace**(Taigman et al., 2014) was one of the first systems to apply deep neural networks to face recognition, achieving a high accuracy rate by using a nine-layer deep neural network to represent facial features. Following **DeepFace**, **FaceNet** (Schroff et al., 2015) introduced the concept of embedding-based face recognition, where faces are represented as points in a high-dimensional space,allowing for efficient face matching using distance metrics. **FaceNet’s** success led to the adoption of embedding-based approaches in numerous applications.

**3. Use of Embedding Techniques**

Face embeddings have become the cornerstone of modern face recognition due to their flexibility and efficiency. The **FaceNet** approach popularized the use of triplet loss, which improves the model's ability to distinguish between similar and dissimilar faces by learning an embedding space where distances represent similarity. **ArcFace** (Deng et al., 2019) further improved embedding techniques by introducing angular margin loss, which enhances the discriminative power of the embeddings. These advancements have inspired the embedding-based approach in this project, where **Dlib** or **Face\_recognition** libraries generate embeddings to accurately represent facial features.

**4. OpenCV for Image Processing**

OpenCV has long been a fundamental tool for image processing in face recognition systems. It provides efficient methods for tasks such as face alignment, scaling, and enhancement, which are essential for pre-processing images before feature extraction. OpenCV’s Haar cascades, though traditional, are often used in real-time detection setups due to their speed, while its more advanced image manipulation functions support face alignment and normalization, improving recognition accuracy in challenging conditions (Bradski & Kaehler, 2008).

**5. Face Alignment Techniques**

Accurate face alignment is essential to improve the robustness of facial recognition models. Misalignment due to pose variations can lead to significant recognition errors. **MTCNN (Multi-task Cascaded Convolutional Networks)**(Zhang et al., 2016) has proven to be highly effective in detecting and aligning faces under various poses. By aligning facial landmarks consistently, face alignment ensures that the model’s inputs are normalized, reducing errors and enhancing the model’s ability to recognize faces in different orientations. This project employs face alignment techniques as part of the preprocessing pipeline to improve detection and recognition outcomes.

**6. Applications of Face Recognition in Real-World Scenarios**

Face recognition has numerous real-world applications that have motivated research into improving its accuracy, scalability, and reliability. **Security and surveillance** are primary fields where face recognition is used to monitor and identify individuals in real time (Jain & Ross, 2008). Additionally, **automated attendance systems** leverage face recognition to reduce manual entry errors and save time in corporate and educational settings. With the advent of **mobile authentication** (such as Apple's Face ID), face recognition has also become popular for user authentication, prioritizing security and convenience. These applications underscore the relevance of this project and its potential for practical impact.

**7. Challenges and Future Directions**

Despite advancements, face recognition systems face challenges with issues such as **bias and fairness**, where certain demographic groups experience reduced accuracy due to training data biases. Studies by **Buolamwini** and **Gebru** (2018) highlight the importance of diverse datasets and fairness-aware algorithms to improve model performance across different demographic groups. Additionally, **privacy and ethical considerations** are increasingly scrutinized as face recognition technology becomes more widespread.

**CLIENT MEETING**

Here’s a structured outline for a client meeting based on the Face Recognition project report. The meeting agenda will guide the client through each stage of the project, from goals and technology overview to implementation, challenges, and next steps.

**Client Meeting Agenda for Face Recognition Project**

**Meeting Objective:**

To provide an overview of the Face Recognition project, discuss progress, highlight key features, and gather feedback or additional requirements from the client.

**Duration:** 1 hour

**1. Introductions and Meeting Overview(5 minutes)**

- Brief round of introductions.

- State the objectives of the meeting: Presenting the project’s progress, key features, challenges encountered, and seeking feedback or input.

**2. Project Overview and Objectives (10 minutes)**

- >**Objective of the Project**:Explain that the project aims to develop a robust face recognition system for applications such as security, automated attendance, and authentication.

->**Scope and Goals**:

- Build a high-accuracy, real-time face recognition system.

- Ensure adaptability for diverse conditions (lighting, facial orientation, expressions).

- Emphasize scalability for potential future enhancements.

**3. Technical Overview (15 minutes)**

- >**Architecture and Workflow**:

- **Describe the project’s architecture and modular structure:**

- **Preprocessing**: Aligning faces to enhance accuracy.

- **Embedding Generation**: Using Dlib or Face\_recognition for feature extraction.

- **Recognition**: Real-time identification by comparing new inputs with stored embeddings.

- **Key Libraries and Tools**: Mention the use of Python, OpenCV, Dlib/Face\_recognition, and PyCharm as the development environment.

- >**Project Structure**:

- Walk through key files and directories, briefly explaining each:

- align\_faces.py, generate\_embeddings.py, main.py, recognize.py, requirements.txt

- Explain the roles of data, embeddings, and logs folders.

**4. Demonstration of Current Capabilities (15 minutes)**

- Face Detection and Alignment Demo: Show how the system detects and aligns faces to ensure uniformity.

- Face Embedding and Recognition Demo: Demonstrate the process of generating embeddings for faces and how the system identifies or verifies a person in real time.

- Example Use Cases: Illustrate potential applications, like security access control or attendance tracking, to give a practical context to the system’s capabilities.

**5. Challenges and Solutions (10 minutes)**

- Technical Challenges:

- Handling diverse conditions such as lighting and orientation.

- Ensuring high accuracy across different demographics.

- Solutions Implemented:

- Face alignment techniques for consistency.

- Embedding techniques to create unique representations of each face.

- Areas for Improvement or Future Considerations:

- Enhancing robustness to challenging conditions.

- Incorporating bias reduction strategies in training data for fairness.

**6. Feedback and Client Requirements (10 minutes)**

- Client Input on Current Progress:

- Ask for feedback on the system's performance and capabilities.

- Inquire if there are specific features the client wants to prioritize.

- Additional Requirements or Adjustments:

- Confirm any adjustments to functionality based on client feedback.

- Discuss potential customization options based on intended use cases (e.g., security vs. user authentication).

**7. Next Steps and Timeline(5 minutes)**

- Outline the upcoming phases:

- Finalizing model tuning and improving accuracy.

- Implementing client-specific adjustments (if any).

- Testing and final validation before deployment.

- Timeline Review: Confirm estimated timelines and agree on follow-up meetings or milestones.

**8. Q&A and Closing Remarks (5 minutes)**

- Address any remaining questions from the client.

- Recap key takeaways from the meeting.

- Thank the client for their input and schedule the next check-in or review session.

**Hardware and Software requirements**

**Hardware requirements:**

1. Processor:

- A multi-core CPU (Intel i5 or AMD equivalent and above) is recommended for efficient computation and faster image processing.

- For enhanced performance, a GPU (such as NVIDIA GTX 1050 or higher) is recommended, especially if dealing with large datasets or real-time face detection in video streams.

2. Memory (RAM):

- Minimum: 8 GB (sufficient for small datasets and basic testing).

- Recommended: 16 GB or higher, to handle larger datasets and reduce processing times.

3. Storage:

- Minimum: 256 GB SSD for faster data access, along with space to store image datasets, embeddings, and logs.

- Recommended: 512 GB SSD or higher, especially if planning to expand datasets or add more features in the future.

4. Camera (for Real-Time Detection):

- A standard HD webcam (720p or higher) if implementing real-time detection.

- For high-accuracy recognition in low-light conditions, a camera with good low-light performance is recommended.

**Software Requirements:**

1. Operating System:

- Windows 10/11, macOS, or Linux (Ubuntu or similar).

- Linux is often preferred for better compatibility with Python packages and libraries.

2. Python (Version 3.8 or Above):

- Python 3.8 or higher is required for compatibility with libraries such as OpenCV, Dlib, and Face\_recognition.

- Version control tools like Anaconda or Conda can be used to manage dependencies if needed.

3. Development Environment:

- PyCharm IDE is recommended for its Python-specific support and debugging tools.

- Other options include Visual Studio Code or Jupyter Notebook for experimentation.

4. Libraries and Packages:

- OpenCV: For image processing tasks such as resizing, converting, and aligning faces.

- Dlib or Face\_recognition Library: For face detection and feature extraction. Dlib offers robust feature mapping, while Face\_recognition simplifies recognition tasks.

- NumPy: For numerical operations, data handling, and matrix calculations.

- TensorFlow/PyTorch (Optional): If you wish to train a custom face recognition model, these libraries are useful for building and training deep learning models.

5. Additional Tools (Optional):

- Git: Version control system for managing code changes.

- Docker (Optional): If you wish to containerize the application for deployment on various environments.

6. Other Dependencies:

- A `requirements.txt` file should be created to manage all dependencies, which can be easily installed using `pip install -r requirements.txt`.

**Network Requirements (Optional):**

- Internet Connection:

- Required initially for installing dependencies and downloading pre-trained models or datasets.

- A stable connection is needed if the project includes cloud storage or remote access.

**IMPLEMENTATION**

The Face Recognition project was implemented through a series of methodical steps designed to ensure accuracy, efficiency, and usability. The key stages in the implementation process are outlined below.

**1. Data Collection and Preparation**

->Data Collection:

A comprehensive dataset of labeled images was gathered, containing multiple faces for each individual. This dataset serves as the foundation for training and testing the model, allowing it to generalize well across various lighting conditions, angles, and facial expressions.

- >Data Organization:

Images were organized into a structured directory format, with each subdirectory corresponding to a different individual. This format simplifies the labeling and retrieval of training data.

- >Data Augmentation (Optional):

To improve model robustness, data augmentation techniques like rotation, scaling, and flipping were applied to create diverse variations of each face.

**2. Face Detection and Alignment**

- >Face Detection:

Face detection was implemented using OpenCV and Dlib. The `cv2.CascadeClassifier` from OpenCV and Dlib’s `get\_frontal\_face\_detector` function were used to detect faces in images. Each detected face was then extracted and cropped to focus on the facial region.

- >Face Alignment:

Face alignment was conducted to ensure consistency across all images. Using Dlib’s facial landmark detection tool, key facial features like the eyes and nose were located to standardize orientation. This step enhances recognition accuracy by reducing variability in the input data.

**3. Feature Extraction and Embedding Generation**

- >Embedding Generation:

Face embeddings, which are unique vector representations of each face, were generated using a pre-trained model. The Face\_recognition library or Dlib’s ResNet-based model was used for this task. Embeddings allow the model to capture unique characteristics of each face, providing a compact representation for recognition.

- >Embedding Storage:

Generated embeddings were stored in an `embeddings/` directory, allowing for quick access during recognition. This setup minimizes computational load by avoiding repeated embedding generation.

**4. Recognition Module**

- >Face Recognition Logic:

In the recognition phase, the model first detects and encodes faces in a new input image. The embeddings generated from these faces are then compared with the stored embeddings using a distance metric (e.g., Euclidean distance). If the distance falls below a defined threshold, the face is identified as a known person; otherwise, it is labeled as "Unknown."

- >Threshold Tuning:

The distance threshold was adjusted to balance the trade-off between false positives and false negatives, enhancing the overall accuracy of the recognition process.

**5. Testing and Evaluation**

- >Model Testing:

The system was tested on a separate validation dataset and with real-time video inputs to assess its performance in different scenarios.

- >Performance Metrics:

Metrics such as accuracy, precision, recall, and F1-score were calculated to quantify the model’s effectiveness. These metrics provided insight into areas for further optimization.

**6. Logging and Error Handling**

- >Logging:

A logging system was established to track each recognition event, including instances of unidentified faces and errors encountered during execution. Logs were stored in a `logs/` directory for later analysis and debugging.

- >Error Handling:

The code includes exception handling to manage unexpected inputs or issues, ensuring smooth and uninterrupted operation.

**7. User Interface (Optional)**

->Graphical User Interface (GUI):

For improved usability, a simple GUI was developed using libraries like Tkinter or PyQt. The GUI allows users to upload images or initiate real-time face recognition via a webcam, with recognized names or “Unknown” status displayed directly within the interface.

**8. Deployment**

- >Local Deployment:

The code, models, and dependencies were packaged for deployment on a local machine. The user can initiate the face recognition system with a single script, making the system accessible to non-technical users.

- >Cloud Deployment (Optional):

For wider accessibility, the system can be deployed to a cloud environment, making it accessible remotely. Docker was utilized to containerize the application, ensuring platform independence and easy scalability.

**Experimentation and Code**

**Main.py:**

import cv2

import os

import face\_recognition

import pandas as pd

from recognize import recognize\_face, get\_embedding, load\_known\_embeddings

from generate\_embeddings import generate\_embeddings

from datetime import datetime

# Load known face embeddings (assumed already generated)

embedding\_file = "embeddings/known\_faces.npy"

known\_faces\_folder = "data/known\_faces"

test\_faces\_folder = "data/test\_faces" # Folder containing test images

log\_file = "logs/face\_recognition\_log.xlsx" # Excel file to log detected faces

# Initialize a DataFrame to store logs

log\_columns = ['Timestamp', 'Image', 'Person']

if os.path.exists(log\_file):

log\_df = pd.read\_excel(log\_file)

else:

log\_df = pd.DataFrame(columns=log\_columns)

# Function to log data into Excel

def log\_face\_detection(image\_name, recognized\_person):

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

log\_df.loc[len(log\_df)] = [timestamp, image\_name, recognized\_person]

log\_df.to\_excel(log\_file, index=False)

def recognize\_from\_webcam(known\_embeddings):

cap = cv2.VideoCapture(0) # Use 0 for default webcam

if not cap.isOpened():

print("Error: Could not open webcam.")

return

frame\_skip = 5 # Adjust this to skip frames for smoother processing

frame\_count = 0

previous\_face\_locations = []

previous\_display\_texts = []

while True:

ret, frame = cap.read()

if not ret:

break

# Resize frame for faster face processing

small\_frame = cv2.resize(frame, (0, 0), fx=0.5, fy=0.5)

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

# Process every `frame\_skip` frames to reduce lag

if frame\_count % frame\_skip == 0:

# Update face locations and encodings every few frames

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

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

# Store current recognized texts to display

current\_display\_texts = []

for face\_location, face\_encoding in zip(face\_locations, face\_encodings):

recognized\_person, confidence = recognize\_face(face\_encoding, known\_embeddings)

# Only display the name if confidence is 96% or greater

if confidence >= 96:

display\_text = f"{recognized\_person}"

# Log the detection if confidence is above threshold

log\_face\_detection("Webcam Frame", recognized\_person)

else:

display\_text = "Unknown"

current\_display\_texts.append((face\_location, display\_text))

previous\_face\_locations = face\_locations

previous\_display\_texts = current\_display\_texts

else:

# Use previous face locations and texts if skipping frames

face\_locations = previous\_face\_locations

current\_display\_texts = previous\_display\_texts

# Draw rectangles and text based on the current or previous data

for (face\_location, display\_text) in current\_display\_texts:

top, right, bottom, left = [coord \* 2 for coord in face\_location] # Scale back up

# Draw rectangle and text for recognized faces only

cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

cv2.putText(frame, display\_text, (left, top - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 255, 0), 2, cv2.LINE\_AA)

# Display the resulting frame

cv2.imshow('Face Recognition', frame)

frame\_count += 1

# Break the loop if the user presses 'q'

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

def recognize\_from\_folder(known\_embeddings):

# Automatically process images from the "test\_faces" folder

if not os.path.exists(test\_faces\_folder):

print("Error: 'test\_faces' folder not found!")

return

image\_files = [f for f in os.listdir(test\_faces\_folder) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif'))]

if not image\_files:

print("Error: No image files found in the test\_faces folder.")

return

for img\_name in image\_files:

img\_path = os.path.join(test\_faces\_folder, img\_name)

print(f"Processing {img\_name}...")

test\_embedding = get\_embedding(img\_path)

if test\_embedding is not None:

recognized\_person, confidence = recognize\_face(test\_embedding, known\_embeddings)

if confidence >= 96:

print(f"Recognized {recognized\_person} with confidence {confidence:.2f}%")

log\_face\_detection(img\_name, recognized\_person)

else:

print("Unknown face detected.")

log\_face\_detection(img\_name, "Unknown")

else:

print("No face detected in the image.")

log\_face\_detection(img\_name, "No face detected")

# Main function to allow user to select between webcam or test image

def main():

# Option to regenerate embeddings

regenerate = input("Do you want to regenerate the embeddings file? (yes/no): ").strip().lower()

if regenerate == 'yes':

generate\_embeddings(known\_faces\_folder, embedding\_file)

# Load embeddings after regeneration (or the existing file if regeneration not chosen)

known\_embeddings = load\_known\_embeddings(embedding\_file)

# Directly use webcam recognition (removing test image choice)

recognize\_from\_webcam(known\_embeddings)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Recognise.py:**

import numpy as np

from scipy.spatial.distance import cosine

import cv2

import face\_recognition

from align\_faces import align\_face

# Load known face embeddings from a .npy file

def load\_known\_embeddings(embedding\_file):

return np.load(embedding\_file, allow\_pickle=True).item()

# Helper function to convert distance to confidence percentage

def distance\_to\_confidence(distance):

return max(0, (1 - distance) \* 100)

# Compare the current face embedding with known embeddings

def recognize\_face(embedding, known\_embeddings, threshold=0.10):

min\_distance = float('inf')

recognized\_person = None

for person, embeddings in known\_embeddings.items():

for known\_embedding in embeddings:

distance = cosine(embedding, known\_embedding)

if distance < min\_distance:

min\_distance = distance

recognized\_person = person

# Convert minimum distance to confidence percentage

confidence = distance\_to\_confidence(min\_distance)

# If confidence is above the threshold, return the recognized person

if confidence >= 96: # 96% confidence

return recognized\_person, confidence

else:

return "Unknown", confidence

# Function to get embedding from an image

def get\_embedding(image\_path):

aligned\_face = align\_face(image\_path)

if aligned\_face is not None:

# Convert image to RGB as face\_recognition expects RGB format

rgb\_face = cv2.cvtColor(aligned\_face, cv2.COLOR\_BGR2RGB)

# Get face encodings (embeddings) for the aligned face

face\_encodings = face\_recognition.face\_encodings(rgb\_face)

if face\_encodings:

return face\_encodings[0] # Assuming one face per image

return None

Align\_faces.py:

import cv2

import face\_recognition

def align\_face(image\_path):

# Load the image

img = cv2.imread(image\_path)

# Check if the image was loaded successfully

if img is None:

print(f"Error: Could not load image at {image\_path}")

return None

# Convert image from BGR (OpenCV default) to RGB (face\_recognition default)

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

# Find all face locations in the image

face\_locations = face\_recognition.face\_locations(rgb\_img)

if len(face\_locations) == 0:

return None

# Get the first face's location

top, right, bottom, left = face\_locations[0]

# Crop and return the aligned face

aligned\_face = img[top:bottom, left:right]

return aligned\_face

generate\_embeddings.py:

import os

import numpy as np

import face\_recognition

import cv2

from align\_faces import align\_face

import sys

def generate\_embeddings(known\_faces\_folder, output\_file):

# Filter out non-directory items like .DS\_Store

valid\_folders = [f for f in os.listdir(known\_faces\_folder) if os.path.isdir(os.path.join(known\_faces\_folder, f))]

# Check if there are any valid folders

if not valid\_folders:

print(

"Error: The known\_faces\_folder is empty or contains only invalid files. Please add image folders to proceed.")

sys.exit(1) # Exit the program with an error code

embeddings = {}

for person in valid\_folders:

person\_path = os.path.join(known\_faces\_folder, person)

print(f"Processing {person}...") # Display "Processing" message

person\_embeddings = []

for img\_name in os.listdir(person\_path):

img\_path = os.path.join(person\_path, img\_name)

# Ignore non-image files

if not img\_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):

continue

aligned\_face = align\_face(img\_path)

if aligned\_face is not None:

# Convert image to RGB as face\_recognition expects RGB format

rgb\_face = cv2.cvtColor(aligned\_face, cv2.COLOR\_BGR2RGB)

# Get face encodings (embeddings) for the aligned face

face\_encodings = face\_recognition.face\_encodings(rgb\_face)

if face\_encodings:

person\_embeddings.append(face\_encodings[0]) # Assuming one face per image

embeddings[person] = person\_embeddings

np.save(output\_file, embeddings)

print(f"Embeddings saved to {output\_file}")

Training Images:

PERSON NAME :

|  |  |
| --- | --- |
|  |  |
|  |  |

**RESULTS:**

|  |
| --- |
|  |

**1. Model Accuracy and Performance**

- Accuracy: Report the accuracy of the face recognition model (e.g., percentage of correctly identified faces). This can be calculated using a test set or cross-validation.

- Precision, Recall, and F1 Score: These metrics give deeper insights into model performance, especially in terms of identifying true positives (correct face recognition) and false positives (incorrect recognition).

- ROC Curve and AUC Score: Show how well the model distinguishes between classes. The AUC (Area Under Curve) score can provide an overall summary of model performance.

**2. Processing Speed and Efficiency**

- Execution Time: Record the average time taken to detect and recognize faces. This is particularly useful for evaluating the real-time capability of the system.

- Resource Utilization: Note any system resources consumed during face recognition (like CPU/GPU usage). High resource usage might be worth mentioning as a limitation.

**3. Comparison with Baselines**

- Comparison with Other Models or Libraries: If possible, compare your model’s results with existing libraries or models, like OpenCV's face recognition, or other methods if applicable. This helps in understanding where your model stands in terms of performance.

**4.Error Analysis**

- Types of Errors: List common errors (e.g., difficulty in recognizing faces with certain angles, lighting issues, or partial obstructions).

- Confusion Matrix: Show how often the model misclassifies a face, which can help pinpoint specific weaknesses.

- Failure Cases: Describe scenarios where the model didn’t perform well and reasons why these cases might be challenging.

**RESULT CAN BE SEEN BY THIS QR CODE :**

|  |
| --- |
|  |

**CONCLUSION:**

This face recognition project successfully demonstrated the application of artificial intelligence and machine learning to identify and verify individuals based on facial features. Through careful dataset selection, model training, and optimization, the system achieved a high accuracy rate in recognizing faces under varied conditions. The user interface designed for this project enabled an accessible, interactive experience, showcasing the potential for user-friendly face recognition applications.

While the project achieved favorable performance metrics, certain limitations were observed, such as sensitivity to lighting conditions, partial occlusions, and specific camera angles. These challenges highlighted the need for further enhancements, potentially involving more robust data preprocessing, a larger and more diverse dataset, and advanced model fine-tuning.

In conclusion, this project not only achieved its primary goal of reliable face recognition but also provided insights into the strengths and limitations of AI-based facial identification. Future improvements could focus on enhancing robustness, addressing security concerns, and exploring deployment in real-world applications, paving the way for practical and secure AI-driven solutions in face recognition technology.

**REFERENCES:**

**1.Library and Framework Documentation**

- "Face Recognition Library Documentation." Face\_recognition, Ageitgey, 2023. [https://face-recognition.readthedocs.io/](https://face-recognition.readthedocs.io/)

- "OpenCV Documentation." OpenCV, 2023. [https://docs.opencv.org/](https://docs.opencv.org/)

**2. Research Papers and Articles on Face Recognition**

- Parkhi, Omkar M., Andrea Vedaldi, and Andrew Zisserman. "Deep Face Recognition." British Machine Vision Conference, 2015.

- Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

**3. Python and Machine Learning Resources**

- Raschka, Sebastian, and Vahid Mirjalili. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing Ltd, 2019.

- Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, 2019.

**4. Conda Environment and Installation Resources**

- "Conda Documentation." Anaconda, Inc., 2023. [https://docs.conda.io/](https://docs.conda.io/)

**5. Ethics and Security in Facial Recognition**

- Harvey, Adam, and Jules LaPlace. "MegaPixels: Origins, Ethics, and Privacy Implications of Datasets in Facial Recognition." Data & Society Research Institute, 2020.

- Raji, Inioluwa Deborah, and Joy Buolamwini. "Actionable Auditing: Investigating the Impact of Publicly Naming Biased Performance Results of Commercial AI Products." AAAI/ACM Conference on AI, Ethics, and Society, 2019.