INDUSTRIAL ORIENTED MINI PROJECT

Report

On

MISSING CHILD IDENTIFICATION USING CNN

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

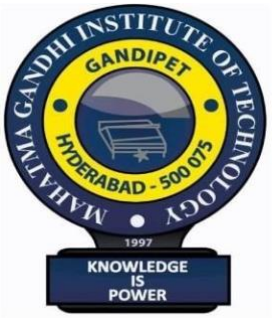
Goturi Harshitha - 22261A1221

Jamuna Shyamala - 22261A1225

Under the guidance of

Mrs.R.Vijaya Lakshmi

Assistant Professor, Department of IT



DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY
(AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **MISSING CHILD IDENTIFICATION USING CNN** submitted by **Goturi Harshitha (22261A1221) and Jamuna Shyamala (22261A1225)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Mrs.R.Vijaya Lakshmi**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:

Mrs.R.Vijaya Lakshmi
Assistant Professor
Dept. of IT

IOMP Supervisor:

Dr. U. Chaitanya
Assistant Professor
Dept. of IT

EXTERNAL EXAMINAR

Dr. D. Vijaya Lakshmi
Professor and HOD
Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **MISSING CHILD IDENTIFICATION USING CNN** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Mrs.R. Vijaya Lakshmi, Assistant Professor**, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Goturi Harshitha - 22261A1221

Jamuna Shyamala - 22261A1225

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our Internal supervisor **Mrs. R. Vijaya Lakshmi, Assistant Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi,** Assistant Professor Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

Goturi Harshitha - 22261A1221

Jamuna Shyamala -22261A1225

ABSTRACT

Every year, thousands of children go missing in India due to abduction, trafficking, accidents, or displacement. Traditional identification methods are often slow, manual, and error-prone, making it difficult to reunite children with their families in a timely manner. This project presents an automated facial recognition system designed to assist law enforcement agencies, NGOs, and families in identifying and locating missing children quickly and accurately.

The system uses a pre-trained ResNet-based Convolutional Neural Network (CNN) via the dlib library to extract deep facial features from uploaded images. These features are converted into 128-dimensional facial encodings, which are then compared against a database of known images using Euclidean distance. If a match is found, the result is displayed on a web-based interface, and an automatic email notification is sent to the registered parent or guardian using SMTP, providing immediate alerts and further action guidance.

Built with Python and Flask, the system is lightweight, user-friendly, and scalable. It is designed to run efficiently on both local servers and cloud platforms, and can be integrated with police databases, public surveillance networks, and child welfare portals. By automating the identification process and enabling real-time notifications, this solution offers a practical and impactful approach to improving child safety and increasing the chances of successful reunification.

TABLE OF CONTENTS

Chapter No	Title	Page No

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 MOTIVATION

The problem of missing children is a serious and growing concern worldwide. Each year, thousands of children are reported missing due to circumstances such as abduction, trafficking, natural disasters, or accidental separation from their families. The ability to rapidly identify and recover these children is critical to minimizing trauma and improving outcomes for both children and their loved ones.

Traditional methods of child identification—such as posters, public announcements, and manual searches—are often slow, prone to error, and ineffective across large or crowded areas. However, advancements in face recognition and automated communication technologies present a powerful opportunity to enhance the efficiency and effectiveness of child recovery efforts.

This project is driven by the need to develop a simple, scalable, and automated Missing Child Identification System that leverages these modern technologies. By utilizing advanced face recognition algorithms, the system can match uploaded images of found children against a curated database of reported missing cases. When a potential match is identified, the system automatically notifies the child's parents or guardians via email, enabling faster and more coordinated response efforts.

In addition, the system provides key performance metrics such as accuracy, precision, and confusion matrix results, allowing for continuous monitoring and improvement. An intuitive web-based interface ensures that law enforcement personnel and volunteers can easily manage the database and conduct searches efficiently.

Ultimately, this project serves as a proof of concept for how AI-driven facial recognition and automated communication can be combined to create meaningful social impact—accelerating the identification and safe reunification of missing children with their families.

1.2 PROBLEM STATEMENT

Current methods for identifying and reuniting missing children are manual, slow, and often inaccurate. The lack of real-time facial recognition and automated alerts leads to delays. A smart, reliable system is needed to match faces quickly, notify guardians instantly, and assist authorities in reuniting children with their families

1.3 EXISTING SYSTEM

Currently, families searching for missing children often have to physically visit police stations or child welfare centers carrying photographs, hoping to identify their child among numerous cases. This process can be stressful and time-consuming, especially when immediate action is needed.

Authorities largely depend on eyewitness accounts and human memory, which are inherently unreliable. The absence of an automated or intelligent system means that matching found children with reports of missing ones is slow and prone to error. Cross-verifying records manually can take days or even weeks, which delays the chances of safe reunification.

Moreover, information about missing children is typically stored in traditional formats such as paper files, spreadsheets, or basic databases. These systems lack the capability to perform advanced image-based matching or face recognition, limiting their effectiveness in quickly identifying missing children.

Even after a child is located, communication with parents or guardians is handled manually through phone calls or in-person visits. This approach causes further delays in delivering crucial information. The lack of an automated notification or alert system reduces the overall speed and efficiency of the recovery process.

1.3.1 LIMITATIONS

- **Variability in Image Quality:** The accuracy of face recognition is highly dependent on the quality and consistency of images. Variations in lighting, facial expressions, angles, or aging can negatively impact the system's ability to correctly identify a child.
- **Limited Database Coverage:** The system can only match a child's image if it already exists in the database. If a missing child's image has not been uploaded or is outdated, the system cannot assist in their identification.
- **Privacy and Data Protection**: Handling and storing images of children involves sensitive personal information, requiring strict adherence to data protection laws and ethical guidelines. Failure to implement robust security measures could lead to unauthorized access or misuse of this data, posing serious privacy risks.

1.4 PROPOSED SYSTEM

The system integrates an advanced face recognition library with a CNN model to ensure precise and reliable matching of facial features. This technology enables accurate identification of missing children by analyzing uploaded images and comparing them against a pre-existing database of registered cases.

Users can upload photos, which the system automatically scans to find potential matches within the stored database, eliminating the need for time-consuming manual searches. This automated matching process significantly accelerates the identification workflow.

Designed for ease of use, the platform provides a straightforward interface where users—such as law enforcement officers, volunteers, or family members—can add information about missing children and perform image-based searches without requiring technical expertise.

Upon uploading an image, the system instantly detects and compares faces, delivering immediate feedback on whether a match exists. This rapid response enables timely interventions and faster reunifications.

When a match is detected, the system automatically sends an email alert to the registered parent or guardian, including the matched child's photo. This automated notification helps reduce delays in communication and supports quicker recovery efforts.

All relevant information—including names, contact emails, and image filenames—is stored in an organized JSON file for easy management and retrieval. Uploaded images are saved in a well-structured folder system, simplifying data maintenance.

1.4.1 ADVANTAGES

- **High Accuracy and Reliability:**By integrating a state-of-the-art face recognition library with a Convolutional Neural Network (CNN) model, the system achieves robust and precise face matching. This significantly reduces false positives and false negatives, ensuring that missing children are correctly identified even in challenging conditions such as varying lighting or partial occlusions.
- **Real-Time Processing and Automated Notifications:**The system quickly analyzes uploaded images and compares them against a database of missing children, providing immediate match results. When a potential match is found, it automatically sends email notifications to the child's parent or guardian, enabling faster response times and increasing the chances of safe recovery

- **Easy Integration and Scalability:**The system is designed to seamlessly integrate with existing databases and workflows used by law enforcement and child welfare organizations. Its modular architecture allows for easy scalability, enabling it to handle growing amounts of data and users without compromising performance. This ensures that the platform can evolve and expand as needed to meet future demands.

1.5 OBJECTIVES

- Allow users to register a child's face with name and email for future matching.
- Compare uploaded unknown face images with stored faces to identify a match.
- Use metrics like accuracy, precision, and confusion matrix to evaluate model performance.
- Send an email to the registered address if a matching face is found.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

1.6.1 SOFTWARE REQUIREMENTS

Software :

The system is developed using Visual Studio Code (VS Code) as the integrated development environment (IDE) for Python. VS Code offers a lightweight, versatile platform for writing, testing, and debugging the application code, including machine learning integration and web backend services.

Primary Language :

Python is the core programming language used throughout the project. It supports the development of face recognition algorithms, image processing, email notifications, and web server logic. Popular Python libraries such as face_recognition (which uses a ResNet-based model for face encoding) for facial analysis, flask for the web framework, and smtplib for email handling form the backbone of the application.

Frontend Framework :

The user interface is created using Flask’s templating system with HTML and CSS, providing a simple, responsive web interface. This allows users to upload images, add missing child details, and view search results through an accessible browser-based platform.

Backend Framework :

The backend logic is implemented in Python using the Flask microframework. It handles core functionalities such as image upload and processing, face matching using convolutional neural network (CNN)-based face encodings derived from a ResNet architecture, email notifications, and data management.

Database :

Data related to missing children — including names, emails, and image filenames — is stored in a JSON file. This lightweight storage method simplifies data management and allows easy retrieval of records for matching and notification purposes.

Frontend Technologies:

- HTML: Provides the structure for web pages and forms.
- CSS: Ensures styling and layout for a clean, user-friendly interface.

Additional Tools and Libraries:

- face_recognition (ResNet-based): Used for detecting and encoding faces in images, leveraging a deep residual network (ResNet) to extract highly accurate face feature embeddings.
- numpy: Used for numerical operations and distance calculations in face matching.
- matplotlib & seaborn: Used for generating and displaying performance metrics such as confusion matrices.
- smtplib & email.mime: Handle automated email notifications to parents or guardians when matches are found.

1.6.2 HARDWARE REQUIREMENTS

Operating System:
Compatible with Windows, macOS, and Linux to ensure wide accessibility.

Processor:
Requires at least an Intel i5 or equivalent for efficient face recognition, image processing, and real-time matching.

RAM:

Minimum 8GB RAM is necessary to handle machine learning computations and multitasking smoothly; 16GB or more is recommended for optimal performance.

Storage:

At least 100MB of free disk space to store user data, images, and logs.

2.LITERATURE SURVEY

Sharma et al. (2023) proposed a Deep Convolutional Neural Network (CNN) model for face re-identification to locate missing children. Published in Elsevier – Expert Systems with Applications, their method demonstrated high precision in .8acial feature matching. However, the model exhibited low performance on aged or occluded faces. The study emphasized the need for robust aging models and validation in real-world deployment environments.

Zhang et al. (2022), in their work published in IEEE Access, introduced a GAN-based facial age progression framework aimed at generating realistic aged face images. While the results showed promising realism, the model was prone to generating visual artifacts and hallucinations. A key limitation identified was the lack of real-time capability, which restricts its use in surveillance systems.

Ahmed et al. (2021) combined hybrid deep learning with Scale-Invariant Feature Transform (SIFT) techniques for recognizing child faces, as detailed in Springer – Multimedia Tools and Applications. Their method showed better generalization across datasets, but it was less accurate when identifying very young children. Additionally, limited testing across diverse population groups highlighted the need for broader dataset inclusion.

Reddy et al. (2020), in their publication in ACM Transactions on Multimedia Computing, developed a mobile app-based solution leveraging cloud services for facial recognition. The system was practically deployable in public spaces, but it was heavily dependent on internet access, leading to potential latency issues. The study recommended enhanced data privacy and secure cloud architectures.

Banerjee and Kumar (2019) explored IoT and GPS-based wearable technologies integrated with facial recognition, as reported in the IEEE Sensors Journal. Their solution enabled real-time tracking and alert generation for missing children. Nevertheless, concerns around device tampering and user privacy were noted. The authors also pointed out the lack of standardization for large-scale adoption at national or international levels.

Nagamani et al. (2025), in a study presented in IJARCCE, applied CNN with transfer learning for identifying missing children. Their approach achieved high precision and recall,

and the architecture was found to be scalable. Despite these advantages, the study raised concerns over user data privacy and called for integration with law enforcement databases for improved effectiveness

Table 2.1 Literature Survey of Missing Child Identification

S.No	Author(s), Year	Journal/Conference	Methodology/Techniques	Merits	Demerits	Research Gaps
1	Sharma et al., 2023	Elsevier – Expert Systems with Applications	Deep CNN with face re-identification for missing children	High precision in facial feature matching	Low performance on aged or occluded faces	Needs robust aging models and real-world deployment studies
2	Zhang et al., 2022	IEEE Access	GAN-based facial age progression framework	Realistic age-transformed face synthesis	Risk of visual artifacts and hallucinations	Lack of real-time capability for surveillance systems
3	Ahmed et al., 2021	Springer – Multimedia Tools and Applications	Hybrid deep learning and SIFT for child face recognition	Better generalization across datasets	Less accurate in very young children	Limited testing on diverse populations
4	Reddy et al., 2020	ACM Transactions on Multimedia Computing	Mobile app-based solution using cloud and facial recognition	Practical deployment in public spaces	Dependent on internet access and cloud latency	Needs more secure and privacy-compliant architecture
5	Banerjee & Kumar, 2019	IEEE Sensors Journal	IoT and GPS wearable tech integrated with facial tracking	Real-time tracking and alert generation	Device tampering and privacy concerns	Lacks standardization for nationwide or international scale tracking systems
6	Nagamani et al., 2025	IJARCCE	CNN with transfer learning for missing child identification	High precision and recall; scalable architecture	Potential privacy concerns	Integration with law enforcement databases

3.ANALYSIS AND DESIGN

The field of child safety and missing child identification faces numerous challenges, particularly in providing accurate and timely recognition across dynamic environments such as public spaces, transportation hubs, and online platforms. This project introduces an advanced identification system designed to address these challenges by leveraging real-time data processing, deep learning techniques, and robust facial recognition models. The system aims to assist law enforcement agencies, child welfare organizations, and families in the rapid identification and recovery of missing children through an intelligent, data-driven approach.

At its core, the system collects and integrates facial images from diverse sources, including public CCTV footage, social media, mobile uploads, and official databases of missing children. Using a deep learning architecture based on Convolutional Neural Networks (CNNs), specifically ResNet-based models, it extracts high-dimensional facial embeddings that capture subtle facial features. These embeddings enable robust and accurate matching even under challenging conditions such as variations in lighting, pose, aging, or partial occlusion.

To further enhance matching accuracy, the system employs distance-based similarity algorithms—including cosine similarity and Euclidean distance—to compare new images with the existing database. Additionally, ensemble methods combine the strengths of multiple CNN models and matching strategies, improving reliability and reducing false matches. Through this hybrid approach, the system provides a fast, scalable, and adaptable solution, empowering users with actionable insights to support the swift and safe recovery of missing children.

3.1 MODULES

Image Upload Module

Allows users (e.g., police, volunteers, guardians) to upload images of missing or found children. Images are securely saved with sanitized filenames to ensure safe storage and retrieval.

- **Input**: Image files uploaded through the web interface.
- **Output**: Stored image in the server's designated folder, ready for face encoding and comparison.

Face Recognition and Comparison Module

Utilizes the face_recognition library (which internally uses a ResNet-based CNN) to extract facial features and compare them with stored records to find potential matches.

- **Input**: Encoded features of uploaded image, and database of stored facial encodings.
- **Output**: Match result (Known/Unknown), identity of the matched child (if any), and metrics like distance score, accuracy, and precision.

Data Storage Module

Manages and stores all registered children's data including their names, emails (guardians), and associated images. Ensures that all data is recorded in a structured JSON format and images are stored in a designated directory.

- **Input**: Child's name, guardian's email, and associated image file.
- **Output**: Updated JSON file (data.json) and organized image storage in static/uploads.

Email Notification Module

Automatically sends an email alert to the parent/guardian when a child's face is matched. The email includes details and the found image as an attachment.

- **Input**: Matched child's name, guardian's email, matched image file.
- **Output**: Email sent with match notification and image attachment.

Evaluation & Visualization Module

Evaluates the performance of face matching using classification metrics and visualizes it using a confusion matrix. This helps in tracking system accuracy and precision.

- **Input**: True and predicted classification labels of face match.
- **Output**: Accuracy, precision values, and a saved confusion matrix image (confusion_matrix.png).

User Interface Module

Provides an intuitive Flask-based web interface for adding child records, uploading images, and viewing match results. It ensures accessibility for non-technical users.

- **Input**: Form data and image uploads from user interaction.
- **Output**: Rendered HTML pages for add, search, and result display.

3.2 ARCHITECTURE

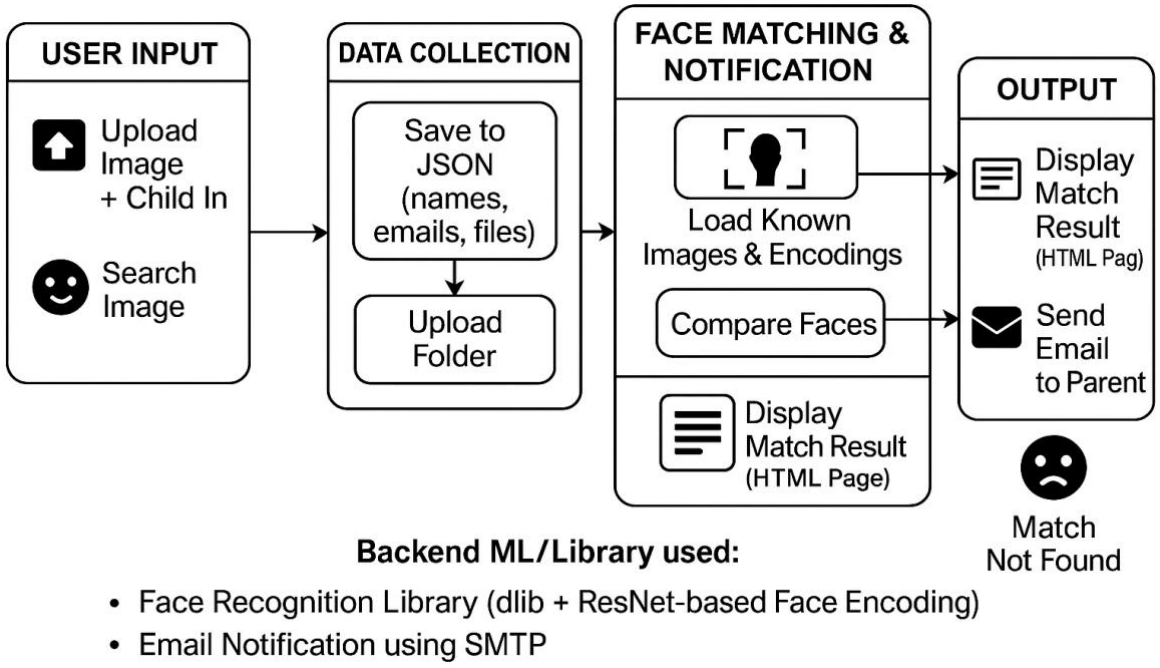


Fig. 3.2.1 Architecture of Missing Child identification using CNN

The architecture of the system, as shown in Fig. 3.2.1, is designed to enable fast and automated identification of missing children through facial recognition. The system begins with the User Input component, where users (such as parents, law enforcement, or volunteers) interact with a simple web interface to either upload details of a missing child (name, guardian email, and image) or search for a potential match by uploading an image of a found child.

Once the image and details are provided, the system proceeds to the Data Collection phase. Here, the child's metadata (name, email, image filename) is securely stored in a JSON file, while the corresponding image is saved in a structured upload folder on the server. This enables efficient organization and retrieval of data for future searches.

In the Face Matching & Notification phase, the system uses a CNN-based facial recognition model (ResNet architecture via the face_recognition library) to process and encode facial features from the uploaded images. When a search is performed, the system loads all known encodings of missing children and compares them to the facial encoding of the uploaded

search image. The model evaluates the similarity using a face distance metric, and if a match is found within the defined threshold, the system proceeds to the next step.
The Output phase presents the results of the face matching. If a match is detected, an HTML page is generated displaying the matched child's information and images. Simultaneously, an automated email notification is sent to the parent or guardian with the match details and the image of the matched child. If no match is found, the system informs the user through the web interface with a clear “Match Not Found” message.

Overall, this architecture provides a robust and user-friendly platform for missing child identification, integrating CNN-based facial recognition with automated email alerts to support timely reunification of children with their families. The system ensures real-time adaptability by continuously incorporating new records, allowing for ongoing expansion and improved matching accuracy.

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A use case diagram is a visual representation that depicts the interactions between various actors and a system, capturing the ways in which users or external entities interact with the system to achieve specific goals. It is an essential tool in system analysis and design, often used in software engineering and business analysis. In a use case diagram, actors are entities external to the system that interact with it, and use cases are specific functionalities or features provided by the system as seen in Fig. 3.3.1.1. These interactions are represented by lines connecting actors to use cases. The diagram helps to illustrate the scope and functionality of a system, providing a high-level view of how users or external entities will interact with it.

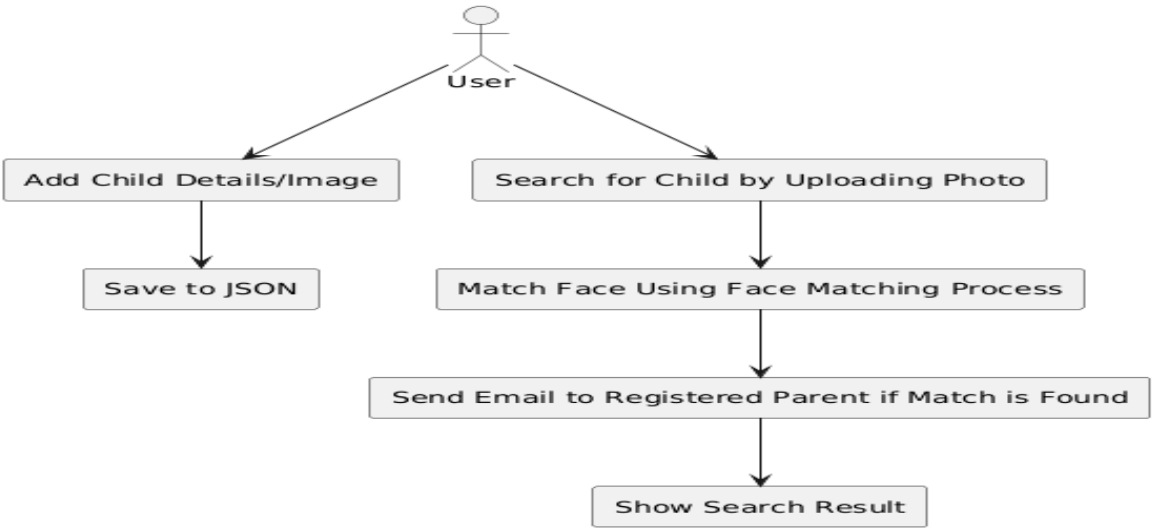


Fig. 3.3.1.1 Use Case Diagram

Actors:

1. **User:**

   The primary actor interacting with the system. This can be a parent, guardian, police official, or authorized individual who performs operations such as uploading child details or searching for a missing child

Use Cases:

1. **Add Child Details / Image**: The user can add new records of a missing child, including the child's name, parent's email address, and image.
2. **Save to JSON**: Once the child details and image are added, the system stores this data securely in a JSON file for future retrieval and processing.
3. **Search for Child by Uploading Photo**: The user can upload a photograph (suspected image of a found child) to search for possible matches in the database of missing children.
4. **Match Face Using Face Matching Process**: The system processes the uploaded image and compares the face encoding with the known encodings stored in the database using the face recognition model.

5. **Send Email to Registered Parent if Match is Found**: If a match is found, the system automatically triggers an email notification to the registered parent or guardian of the child, providing relevant information and possibly the found image.
6. **Show Search Result**: The system displays the result of the face matching process to the user on the web interface. If a match is found, the result includes details of the matched child; if no match is found, the system indicates "Match Not Found."

3.3.2 CLASS DIAGRAM

A class diagram is a detailed visual representation of a system's static structure, illustrating its classes along with their attributes, methods, and the relationships connecting them. It acts as a foundational blueprint in object-oriented design, providing a clear overview of how various components within the system interact, collaborate, and depend on one another. By mapping out class hierarchies, associations, aggregations, and dependencies, class diagrams facilitate better understanding and organization of complex systems. They are essential tools for software engineers and developers to plan, communicate ideas effectively within teams, identify potential design issues early, and ensure that the system architecture is coherent and scalable before moving into implementation. This clarity ultimately leads to more efficient development and easier maintenance of software projects..

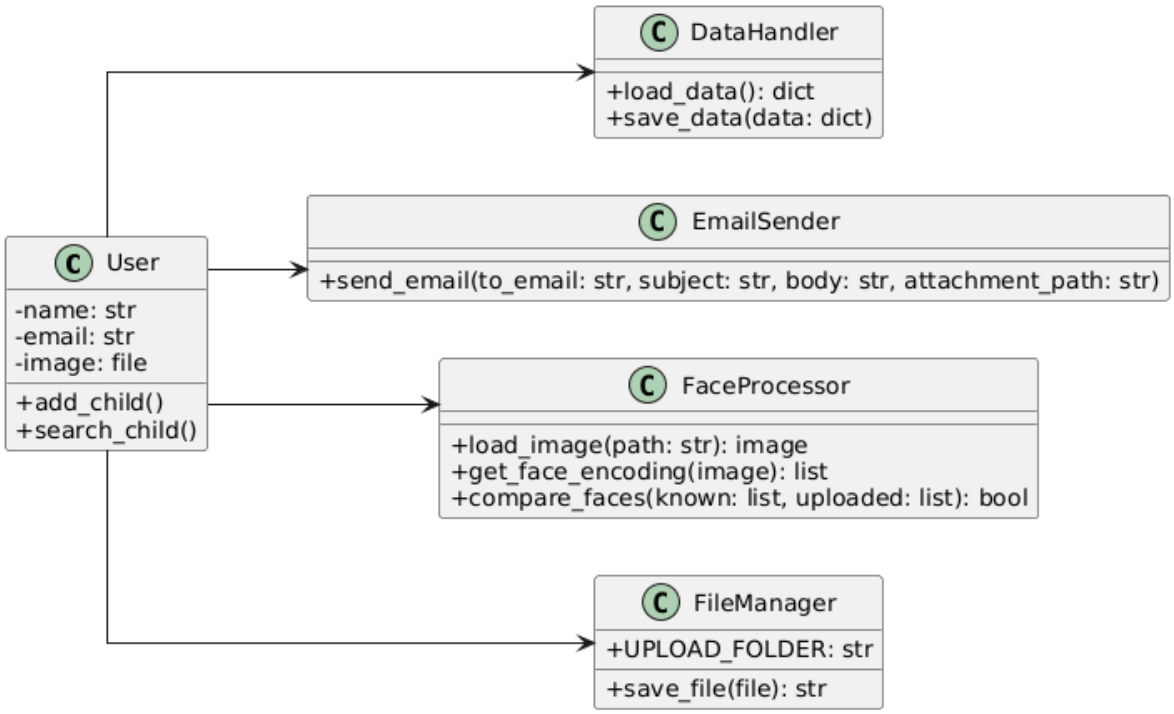


Fig. 3.3.2.1 Class Diagram

Relationships:

1. **User → DataHandler**
    - The *User* interacts with the *DataHandler* to register missing children and upload necessary information (name, email, image).
    - The *DataHandler* manages storage of user and child data in a structured JSON file, ensuring persistent and organized data management.
2. **User → FaceProcessor**
    - During a search operation, the *User* triggers the *FaceProcessor*, which loads uploaded images and extracts face encodings using a deep learning-based ResNet model.
    - The *FaceProcessor* compares the extracted face encoding against the stored database of missing children to identify potential matches.
3. **User → FileManager**
    - The *User* uploads child images through the system interface.
    - The *FileManager* saves these images to a predefined server directory, maintaining an organized file structure for efficient retrieval during searches.
4. **User → EmailSender**
    - If a match is detected, the *User* indirectly triggers the *EmailSender*.
    - The *EmailSender* sends an automated email notification to the registered parent or guardian, including the matched child's image and details to facilitate prompt reunification.

System Flow:

1. **User Interaction**
    - The *User* registers missing children by providing the child's name, parent's email, and an image through the system's web interface.
    - The *DataHandler* stores this information, and the *FileManager* handles image storage.
2. **Data Preparation & Search**
    - When an image is uploaded for searching, the *FaceProcessor* extracts facial encodings and compares them to the known encodings of missing children.

- This step utilizes a ResNet-based face recognition model for accurate and reliable matching.

3. **Match Detection & Notification**
   - If a match is found, the system automatically triggers the *EmailSender* to notify the registered parent or guardian via email, with the relevant child image attached.
   - If no match is found, the system informs the user and provides model evaluation metrics (accuracy, precision, confusion matrix).
4. **Data Management**
   - Throughout the process, the *DataHandler* ensures all records (user details, child info, image filenames) are correctly managed, and the *FileManager* ensures proper organization of image files.
   -

3.3.3 ACTIVITY DIAGRAM

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig. 3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow. Activity diagrams are commonly used to model business processes, workflows, or any sequential activities in a system.

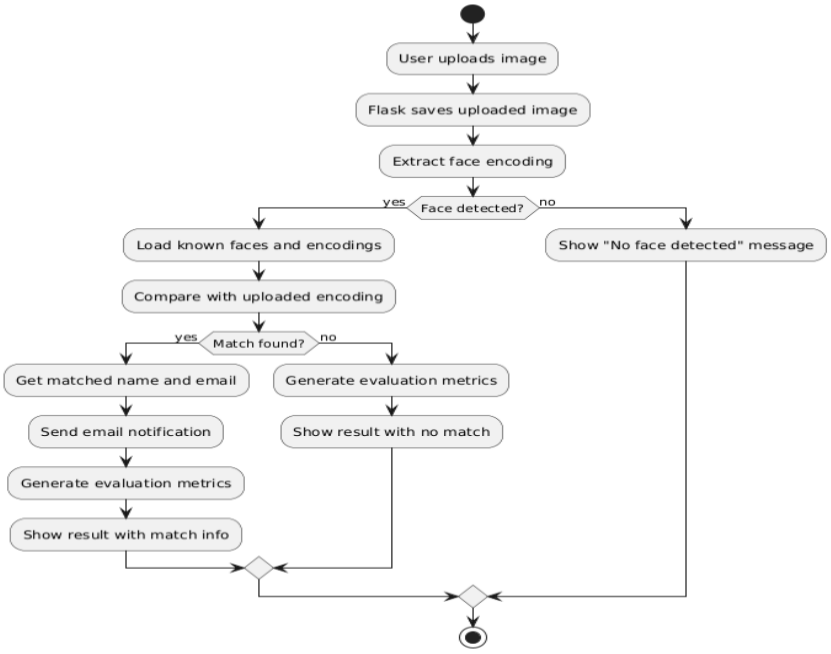


Fig. 3.3.3.1 Activity Diagram

Flow Explanation:

1.Upload Image

The user initiates the process by uploading an image through the system interface.

2. Save Uploaded Image

The Flask application saves the uploaded image securely on the server.

3. Extract Face Encoding

The system attempts to extract the face encoding from the uploaded image using the facial recognition module.

4. Face Detected?

- **Yes:** The system proceeds to the next step.
- **No:** The system shows a message: "No face detected" and terminates the current flow.

5. Load Known Faces and Encodings

The system loads previously stored facial encodings and corresponding identity data from the database or storage.

6. Compare with Uploaded Encoding

The system compares the uploaded face encoding with the stored encodings to find a potential match.

7. Match Found?

- **Yes:**
    - Retrieve the matched name and email ID.
    - Send an email notification to the associated user.
    - Generate evaluation metrics (e.g., accuracy, similarity score).
    - Display the results with matched user information.
- **No:**
    - Generate evaluation metrics.
    - Display the result indicating no match was found.

8. End of Workflow

The process concludes, and the user can either:

- Upload another image, or
- Exit the system.

3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig. 3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

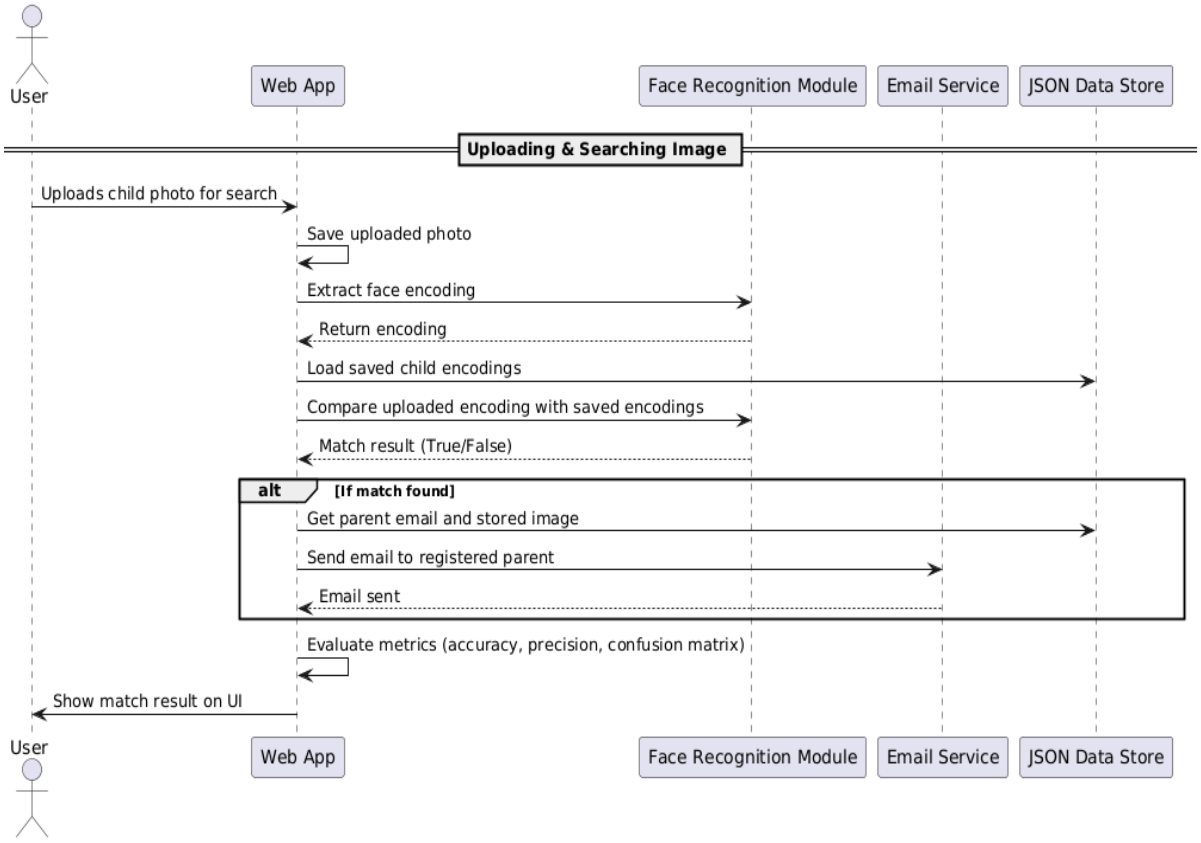


Fig. 3.3.4.1 Sequence Diagra

Key Interactions and Relationships

User and System:

- **Upload Photo:**

  The user uploads a photo of a child through the web application to initiate a search for a match.

System and Face Recognition Module:

- **Save and Forward Image:**
  The web app saves the uploaded image locally and forwards it to the face recognition module for processing.
- **Extract Face Encoding:**
  The face recognition module processes the image and extracts facial encodings.
- **Return Encoding to System:**
  The extracted facial encoding is sent back to the web application for comparison.

System and JSON Data Store:

- **Load Stored Encodings**:
  The system retrieves previously saved face encodings and associated data (e.g., names, emails) from the JSON data store.

Face Recognition Module:

- **Compare Encodings:**
  The module compares the encoding of the uploaded image with stored encodings to find a possible match.
- **Return Matching Result:**
  The result of the comparison (match found or not found) is returned to the system.

System and Email Service:

- **Send Notification Email:**
  If a match is found, the system sends an email notification with details and the uploaded photo to the registered parent.

System and User:

- **Display Search Results:**
  The system shows the match result (e.g., matched name, email, or no match found) on the user interface along with evaluation metrics like accuracy and precision.

3.3.5 COMPONENT DIAGRAM

A **Component Diagram** is a structural diagram in software engineering that illustrates the organization and interrelationships of various software components within a system. Each component encapsulates a set of related functionalities and interacts with others through well-defined interfaces, promoting modularity and reusability. This diagram helps developers visualize the high-level architecture, showing how modules like databases, user

interfaces, processing units, and external services (e.g., APIs or email systems) are connected. In the context of systems like the one shown in Fig. 3.3.5.1, the component diagram effectively represents how key modules such as the web application, face recognition engine, data store, and notification service coordinate to perform tasks like user registration, face matching, and alert notifications.

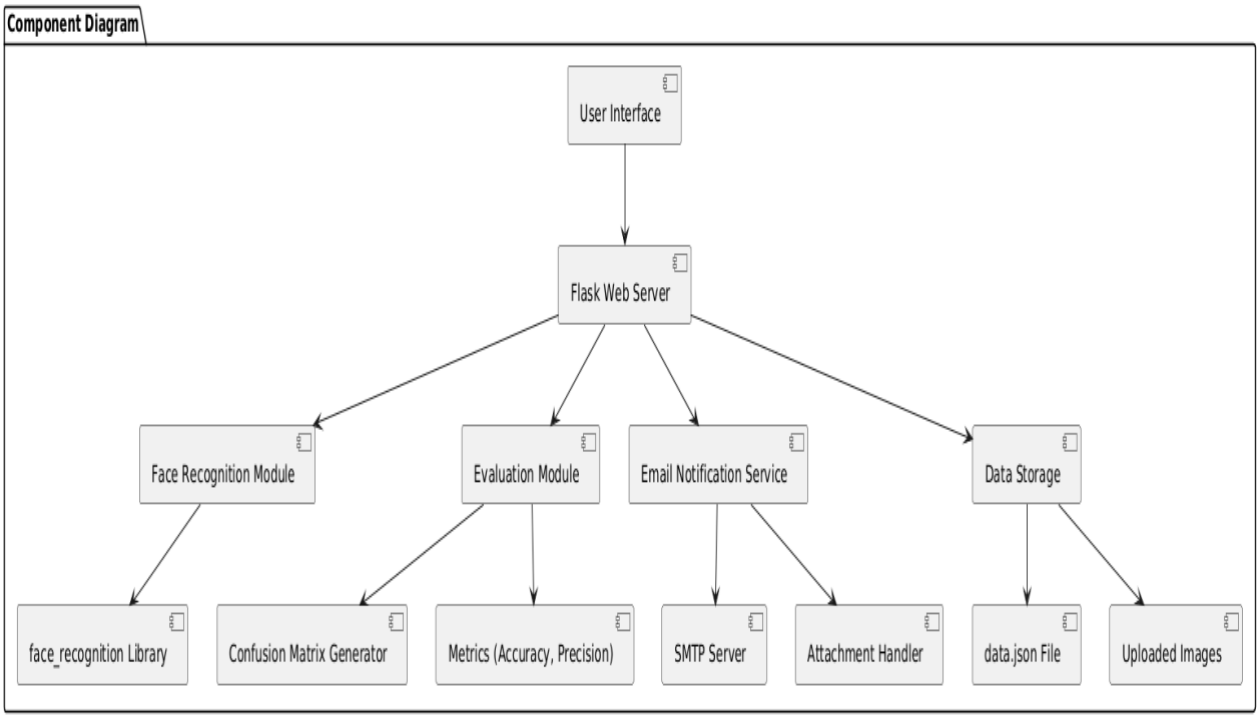


Fig. 3.3.5.1 Component Diagram

Main Components:

User Interface: The front-end of the application that allows users to upload images, initiate searches, and view results such as match outcomes and confusion matrix visualizations.

Flask Web Server: The central backend server that handles user requests, coordinates between different components like face recognition, evaluation, email service, and data storage, and manages the overall workflow.

Face Recognition Module: Uses the face_recognition library to extract facial encodings from images and perform comparisons to identify matches.

Confusion Matrix Generator: A utility that visualizes the performance of the recognition system using a confusion matrix to help interpret model accuracy and precision.

Evaluation Module: Computes performance metrics such as accuracy and precision after comparing facial encodings. It also prepares data for visualization.

Metrics (Accuracy, Precision): Quantifies how well the system is identifying matches versus non-matches.

Email Notification Service: Sends notification emails to the registered parent or guardian when a match is found, using the SMTP server.

SMTP Server: An external mail server used to send the actual email alerts from the system.

Attachment Handler: Manages file attachments (e.g., the uploaded or matched images) included in the emails sent to users.

Data Storage: Manages persistent storage of user data, uploaded image metadata, and encodings through a JSON file and file system.

data.json File: Stores metadata such as names, email addresses, and filenames associated with uploaded images for matching purposes.

Uploaded Images: Directory or storage unit containing the raw uploaded photos used for face recognition and comparison.

3.3.6 DEPLOYMENT DIAGRAM

The deployment diagram shows a face recognition system where users interact with a Flask web application through their browser to add or search for faces. The Flask server processes these HTTP requests and manages data using a local file storage system (static/uploads, data.json). For facial recognition, it uses a ResNet-based module with the face_recognition library to encode and match faces. When a face match is detected during a search, the system automatically sends an email notification to the parent using the Gmail SMTP server. This architecture illustrates how the web server, face recognition module, storage, and external email service work together to deliver a responsive and automated real-world solution.

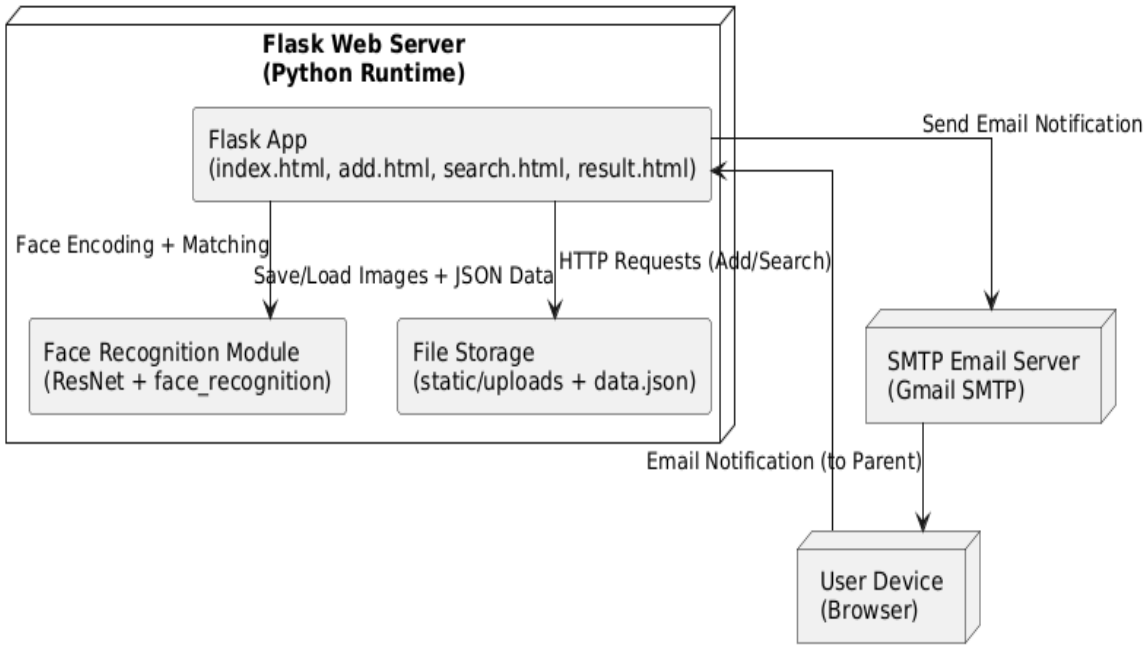


Fig. 3.3.6.1 Deployment Diagram

The deployment diagram shows how the system components are distributed across four main nodes:

• **User Device**: Runs the web browser where users upload images or search for faces. It interacts with the server through HTTP requests to add, search, or view results.

• **Web Server**: Hosts the Flask web application, handling routing and application logic. It processes user requests and serves HTML templates like index.html, add.html, search.html, and result.html.

• **Face Recognition Module**: Integrated into the server, it uses a ResNet-based model via the face_recognition library to encode and compare faces. This module performs the core matching logic.

• **Storage & Email Node**: Includes local file storage (static/uploads, data.json) for storing images and user data, and connects to an external Gmail SMTP server to send email notifications when a match is found.

This setup ensures effective interaction between the client, server-side logic, data processing, and external communication in the face recognition-based notification system.

3.4 METHODOLOGY

Data Acquisition

- **User-Provided Images**: The system collects facial data through images uploaded by users via a web interface. These images represent known individuals (e.g., children) and are essential for creating a reference dataset. Each image is linked to a name and email, helping identify and notify the parent in case of a match.
- **Storage**: Uploaded images are saved in a local folder (static/uploads), and metadata such as names, emails, and filenames are recorded in a structured JSON file (data.json). This enables efficient storage, retrieval, and management of face data for encoding and matching.

Model Steps

- **Face Encoding**: Both uploaded and stored images are processed using the face_recognition library, which utilizes a ResNet-based deep learning model. It extracts 128-dimensional feature vectors (encodings) from each face image that uniquely represent facial characteristics.
- **Matching**: During the search operation, the system compares the encoding of the uploaded image with stored encodings using Euclidean distance. If the distance between encodings is below a set threshold (indicating similarity), the closest match is identified as the correct individual.
- **Evaluation**: To ensure model reliability, the system calculates key performance metrics like **accuracy** and **precision**. It also generates a **confusion matrix** using Seaborn to visually represent prediction outcomes (True Positive, False Positive, etc.), providing insights into model behavior.

Models Used

1. ResNet (via face_recognition library)

- **Description**: ResNet is a powerful deep convolutional neural network known for its efficiency in image recognition tasks. It is pre-trained on large face datasets and integrated into the face_recognition library.
- **How it Works**: ResNet extracts facial embeddings (numerical vectors) that encode spatial features of the face. These embeddings are invariant to changes like lighting, pose, or minor occlusions, making matching more reliable.

- **Why It's Used**: ResNet provides a high level of accuracy and robustness, making it ideal for real-time face recognition systems where fast and dependable results are critical.

Notification System

Email Alerts (SMTP)

- **Description**: The system is equipped with an automated notification module that alerts the parent via email when their child's face is recognized in a search operation.
- **How it Works**: Using the smtplib module and Gmail's SMTP server, the system constructs and sends an email containing a message and optionally the image that triggered the alert. This message is sent to the email linked with the matched face in the database.
- **Why It's Used**: This real-time alert system ensures that parents or guardians are immediately informed about their child's location or recognition event, improving safety and response time in critical situations.

4. CODE AND IMPLEMENTATION

4.1 CODE

app.py

```
import os
import json
import smtplib
import numpy as np
from flask import Flask, render_template, request, redirect, url_for
from werkzeug.utils import secure_filename
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
import face_recognition
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
import matplotlib.pyplot as plt
import seaborn as sns

app = Flask(__name__)
UPLOAD_FOLDER = 'static/uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

DATA_FILE = 'data.json'

def load_data():
    if not os.path.exists(DATA_FILE):
        with open(DATA_FILE, 'w') as f:
            json.dump({"names": [], "emails": [], "filenames": []}, f)
    with open(DATA_FILE, 'r') as f:
        return json.load(f)

def save_data(data):
```

```
    with open(DATA_FILE, 'w') as f:
        json.dump(data, f)

def send_email(to_email, subject, body, attachment_path=None):
    SENDER_EMAIL = 'bunnyjamuna2004@gmail.com'  # your email
    SENDER_PASSWORD = 'pveiktfefnhzykij'  # your app password

    try:
        message = MIMEMultipart()
        message['From'] = SENDER_EMAIL
        message['To'] = to_email
        message['Subject'] = subject
        message.attach(MIMEText(body, 'plain'))

        if attachment_path:
            with open(attachment_path, "rb") as attachment:
                part = MIMEApplication(attachment.read(),
Name=os.path.basename(attachment_path))
                part['Content-Disposition'] = f'attachment;
filename="{os.path.basename(attachment_path)}"'
                message.attach(part)

        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login(SENDER_EMAIL, SENDER_PASSWORD)
        server.send_message(message)
        server.quit()

    except Exception as e:
        print("❌ Failed to send email:", str(e))

def load_known_faces():
    data = load_data()
```

```
    known_encodings = []
    names = []
    for name, filename in zip(data['names'], data['filenames']):
        img_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        image = face_recognition.load_image_file(img_path)
        encodings = face_recognition.face_encodings(image)
        if encodings:
            known_encodings.append(encodings[0])
            names.append(name)
    return names, known_encodings

def evaluate_model(known_encodings, known_names, uploaded_encoding,
threshold=0.6):
    # Compare the uploaded encoding to known encodings
    distances = face_recognition.face_distance(known_encodings, uploaded_encoding)
    min_distance = np.min(distances)
    min_index = np.argmin(distances)

    # Simple binary classification for demonstration (Known vs Unknown)
    y_true = ['Known']  # We know uploaded image is "Known" if matched, else Unknown
    y_pred = []

    if min_distance < threshold:
        y_pred.append('Known')
        matched_name = known_names[min_index]
    else:
        y_pred.append('Unknown')
        matched_name = None

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, pos_label='Known', zero_division=0)
    cm = confusion_matrix(y_true, y_pred, labels=['Known', 'Unknown'])

    # Plot confusion matrix
```

```
    plt.figure(figsize=(4,3))
    sns.heatmap(cm, annot=True, fmt='d', xticklabels=['Known', 'Unknown'],
yticklabels=['Known', 'Unknown'])
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    cm_path = os.path.join('static', 'confusion_matrix.png')
    plt.savefig(cm_path)
    plt.close()

    return accuracy, precision, cm_path, matched_name

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/add', methods=['GET', 'POST'])
def add():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        file = request.files['file']

        if file:
            filename = secure_filename(file.filename)
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            file.save(file_path)

            data = load_data()
            data['names'].append(name)
            data['emails'].append(email)
            data['filenames'].append(filename)
            save_data(data)
```

```
            return redirect(url_for('index'))
    return render_template('add.html')

@app.route('/search', methods=['GET', 'POST'])
def search():
    if request.method == 'POST':
        file = request.files['file']
        if file:
            uploaded_filename = secure_filename(file.filename)
            uploaded_path = os.path.join(app.config['UPLOAD_FOLDER'],
uploaded_filename)
            file.save(uploaded_path)

            uploaded_image = face_recognition.load_image_file(uploaded_path)
            uploaded_encodings = face_recognition.face_encodings(uploaded_image)
            if not uploaded_encodings:
                return "No face found in the uploaded image."

            uploaded_encoding = uploaded_encodings[0]

            known_names, known_encodings = load_known_faces()

            accuracy, precision, cm_path, matched_name = evaluate_model(
                known_encodings, known_names, uploaded_encoding)

            if matched_name:
                # Find email & filename for matched name
                data = load_data()
                idx = data['names'].index(matched_name)
                parent_email = data['emails'][idx]
                matched_image = data['filenames'][idx]

                # Send email notification
                send_email(parent_email, "Child Found",
```

```
                f"Your child {matched_name} was found.",
                attachment_path=uploaded_path)

        return render_template('result.html',
                               matched=True,
                               name=matched_name,
                               email=parent_email,
                               uploaded_image=uploaded_filename,
                               matched_image=matched_image,
                               accuracy=accuracy,
                               precision=precision,
                               confusion_matrix_image='confusion_matrix.png')

    else:
        return render_template('result.html', matched=False,
                               accuracy=accuracy,
                               precision=precision,
                               confusion_matrix_image='confusion_matrix.png')

return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Missing Child Identification using CNN</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
    <style>
        body {
            background: linear-gradient(135deg, #ffecd2 0%, #fcb69f 100%);
```

```
        }
        .card {
            background-color: #ffffff;
            border: none;
            border-radius: 1rem;
            box-shadow: 0px 8px 16px rgba(0, 0, 0, 0.1);
        }
        h1 {
            color: #ff5722;
        }
        .btn-primary {
            background-color: #007bff;
            border: none;
        }
        .btn-success {
            background-color: #28a745;
            border: none;
        }
    </style>
</head>
<body class="d-flex justify-content-center align-items-center min-vh-100">
    <div class="container">
        <div class="card p-5 shadow-sm text-center" style="max-width: 500px; margin:
auto;">
            <h1 class="mb-4">Missing Child Identification using CNN</h1>
            <a href="{{ url_for('add') }}" class="btn btn-primary btn-block mb-3">Add
Child</a>
            <form action="{{ url_for('search') }}" method="post"
enctype="multipart/form-data" class="w-100">
                <div class="form-group">
                    <input type="file" name="file" required class="form-control-file mb-3">
                </div>
                <button type="submit" class="btn btn-success btn-block">Search
Child</button>
```

```
            </form>
        </div>
    </div>
</body>
</html>
```

add.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Add Missing Child</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
    <style>
        body {
            background: linear-gradient(135deg, #74ebd5 0%, #ACB6E5 100%);
        }
        .card {
            background-color: #ffffff;
            border: none;
            border-radius: 1rem;
        }
        .btn-primary {
            background-color: #007bff;
            border: none;
        }
        .btn-primary:hover {
            background-color: #0056b3;
        }
        .success-message {
            color: green;
            font-weight: bold;
            margin-top: 15px;
            text-align: center;
```

```
    }
  </style>
</head>
<body class="d-flex justify-content-center align-items-center min-vh-100">
  <div class="card p-5 shadow-lg w-100" style="max-width: 500px;">
    <h2 class="mb-4 text-primary text-center">Add Missing Child Info</h2>

    {% if success %}
      <div class="success-message">✅ Child information added
successfully!</div>
    {% endif %}

    <form action="{{ url_for('add') }}" method="post" enctype="multipart/form-
data" class="w-100 mt-4">
      <div class="form-group">
        <input type="text" name="name" placeholder="Child Name" required
class="form-control mb-3">
      </div>
      <div class="form-group">
        <input type="email" name="email" placeholder="Parent Email" required
class="form-control mb-3">
      </div>
      <div class="form-group">
        <input type="file" name="file" required class="form-control-file mb-4">
      </div>
      <button type="submit" class="btn btn-primary btn-block">Add</button>
    </form>

    <div class="text-center mt-3">
      <a href="{{ url_for('index') }}" class="btn btn-outline-secondary">Back to
Home</a>
    </div>
  </div>
```

```
</body>
</html>
```

search.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Search Child</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
</head>
<body class="bg-light">
    <div class="container d-flex flex-column align-items-center justify-content-center
min-vh-100">
        <div class="card p-5 shadow-sm w-100" style="max-width: 500px;">
            <h1 class="mb-4 text-center">Search Child</h1>
            <form method="POST" enctype="multipart/form-data" class="w-100">
                <div class="form-group">
                    <label for="fileInput">Choose File</label>
                    <input type="file" name="file" id="fileInput" class="form-control-file
mb-3" required>
                </div>
                <button type="submit" class="btn btn-success btn-block">Search</button>
            </form>
            <div class="text-center mt-4">
                <a href="{{ url_for('index') }}" class="btn btn-link">Back to Home</a>
            </div>
        </div>
    </div>
</body>
</html>
```

result.html

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <title>Result</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
    <style>
        body {
            background: linear-gradient(135deg, #ffecd2 0%, #fcb69f 100%);
        }
        .card {
            background-color: #ffffff;
            border: none;
            border-radius: 1rem;
            box-shadow: 0px 8px 16px rgba(0, 0, 0, 0.1);
        }
        .btn-warning {
            background-color: #f39c12;
            border: none;
        }
        .btn-warning:hover {
            background-color: #e67e22;
        }
        .img-thumbnail {
            width: 150px;
            height: 150px;
            object-fit: cover;
        }
        .d-flex {
            display: flex;
            align-items: center;
            justify-content: start;
        }
        .mr-3 {
            margin-right: 15px;
```

```
        }
        .matched-img {
            margin-top: -20px;
            margin-left: 20px;
        }
        .image-label {
            margin-right: 10px;
        }
        .metrics img.confusion-matrix {
            max-width: 100%;
            height: auto;
            margin-top: 15px;
        }
    </style>
</head>
<body class="d-flex justify-content-center align-items-center min-vh-100">
    <div class="container">
        <div class="card p-5 shadow-sm text-center" style="max-width: 600px; margin:
auto;">
            {% if matched %}
                <h2 class="text-success mb-4">Match Found!</h2>
                <p><strong>Name:</strong> {{ name }}</p>
                <p><strong>Email:</strong> {{ email }}</p>
                <div class="d-flex justify-content-start mb-3">
                    <div class="d-flex align-items-center mr-4">
                        <h4 class="image-label">Uploaded Image:</h4>
                        <img src="{{ url_for('static', filename='uploads/' + uploaded_image)
}}" class="img-thumbnail">
                    </div>
                    <div class="d-flex align-items-center">
                        <h4 class="image-label">Matched Image:</h4>
                        <img src="{{ url_for('static', filename='uploads/' + matched_image) }}"
class="img-thumbnail matched-img">
                    </div>
```

```
            </div>
        {% else %}
            <h2 class="text-danger mb-4">No Match Found</h2>
        {% endif %}

        {% if accuracy is not none and precision is not none %}
        <div class="metrics text-left">
            <h4>Evaluation Metrics</h4>
            <p><strong>Accuracy:</strong> {{ (accuracy * 100) | round(2) }}%</p>
            <p><strong>Precision:</strong> {{ (precision * 100) | round(2) }}%</p>
            <h5>Confusion Matrix:</h5>
            <img src="{{ url_for('static', filename=confusion_matrix_image) }}"
alt="Confusion Matrix" class="confusion-matrix">
        </div>
        {% endif %}

        <br><br>
        <a href="{{ url_for('index') }}" class="btn btn-warning mt-3">Back to
Home</a>
      </div>
   </div>
</body>
</html>
```

4.2 IMPLEMENTATION

(Missing child identification using cnn)

```
│
├── app.py                    #  Main Flask application (the full Python code you
shared)
├── requirements.txt          #  List of required Python libraries
│
├── data/
│   └── data.json             # Stores user information: names, emails, filenames
│
├── static/
│   ├── uploads/              #  Stores uploaded images (both known and search)
│   ├── css/
│   │   └── style.css         # Optional: Add any custom CSS styles here
│   └── confusion_matrix.png  #  Auto-generated image of confusion matrix
│
├── templates/
│   ├── index.html            #  Homepage with links to "Add" and "Search"
│   ├── add.html              #  Upload form for known person (name, email, image)
│   ├── search.html           #  Search interface (upload only)
│   └── result.html           #  Displays result with match, accuracy, precision,
confusion matrix
│
└── README.md                 # Optional: Describe setup, purpose, and usage of the
app
```

Installing Python Packages

Open your terminal and run the following command to install all required packages:

pip install flask face_recognition numpy scikit-learn matplotlib seaborn

If you're using a requirements.txt, you can also run:

pip install -r requirements.txt

Optional: If you're using VS Code, make sure your virtual environment is activated before installing.

Setting Up Image Data Storage

The app uses a simple data.json file for storing known user data (name, email, uploaded filename). No database setup is required.

To manually view/edit known data:

1. Navigate to: data/data.json
2. Contents look like:

{ "names": ["John Doe"],

"emails": ["john@example.com"],

"filenames": ["john.png"] }

2. Start the Flask server:

   python app.py

3. Open your browser and go to:

   http://127.0.0.1:5000

Enabling Email Notifications

To send email alerts when a match is found:

1. Go to Google App Passwords
2. Generate an App Password for your Gmail.
3. Open app.py and replace:

SENDER_EMAIL = 'your_email@gmail.com'

SENDER_PASSWORD = 'your_app_password'

Works with Gmail App Passwords only (not your regular login password).

Using the App

1. Go to /add

   Upload a known image with name and email.
2. Go to /search

   Upload a new image to check if it matches any known user.
3. If matched:
   - Shows name, email, image.
   - Sends email alert.
   - Displays **confusion matrix**, **accuracy**, and **precision**.

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing is a critical component of the software development lifecycle that ensures an application performs reliably, handles edge cases gracefully, and meets all user requirements. It helps validate both the correctness and robustness of the system by simulating real-world scenarios, user behaviors, and unexpected inputs.

In this project, **Face Recognition with Email Alert System**, testing was vital to ensure that key features—including known user registration, face matching, email alerting, and graceful error handling—work as expected under all conditions. The objective was to validate that each part of the application—ranging from user input forms to backend face recognition and alert delivery—functions consistently and securely.

The test cases were carefully designed to cover the major functionalities and user interaction flows of the application, such as:

- Correct registration of users with name, email, and facial image
- Accurate detection of facial matches in uploaded search images
- Automated email alert delivery when a match is detected
- Graceful error handling for invalid inputs (e.g., no face found, email error)
- User-friendly messages for edge cases like no match or no face in image

5.2 TEST CASES:

Table 5.1 Test Cases of Missing Child Identification

Test Case ID	Test Case Name	Input	Expected Output	Actual Output	Remarks
TC_01	Add Child - Valid	Name, Email, Clear Image	Data saved, image stored	Data saved, image stored	Pass
TC_02	Add Child - No Image	Name, Email (No Image)	Show error, don’t save	Show error, don’t save	Pass

TC_03	Search Child - Match Found	Upload known image	Match shown, email sent	Match shown, email sent	Pass
TC_04	Search Child - No Match	Upload unknown image	Show "No match found"	Show "No match found"	Pass
TC_05	Upload Image - No Face	Upload random image (no face)	Show "No face found"	Show "No face found"	Pass
TC_06	Email Error Handling	Enter invalid email	Catch and log email error	Catch and log email error	Pass

6. RESULTS

Fig. 6.1 Initial page

Fig. 6.1 shows displays the homepage of the Missing Child Identification web application, launched at local host address. This page acts as the primary interface where users can interact with the system to add known child profiles or search for a missing child using facial recognition powered by Convolutional Neural Networks (CNN).

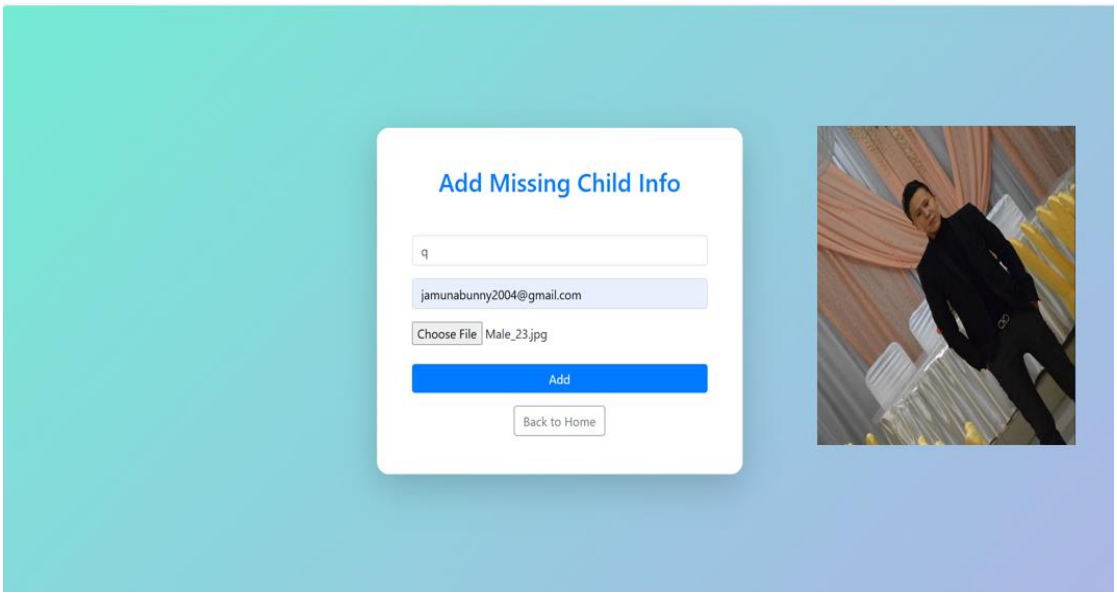


Fig. 6.2 Input page

Fig. 6.2 This page displays users to add information about a missing child, including their name, contact email, and a clear image. It is designed to store the details in the system for future facial recognition and matching. The simple and user-friendly interface ensures quick data entry and easy navigation back to the homepage.

Fig. 6.3 Search image page

Fig. 6.3 displys facilitates the searching of missing children by allowing users to upload an image. Once the image is selected and the "Search Child" button is clicked, the system uses CNN-based facial recognition to find a possible match. It simplifies the identification process through an intuitive and user-friendly interface.

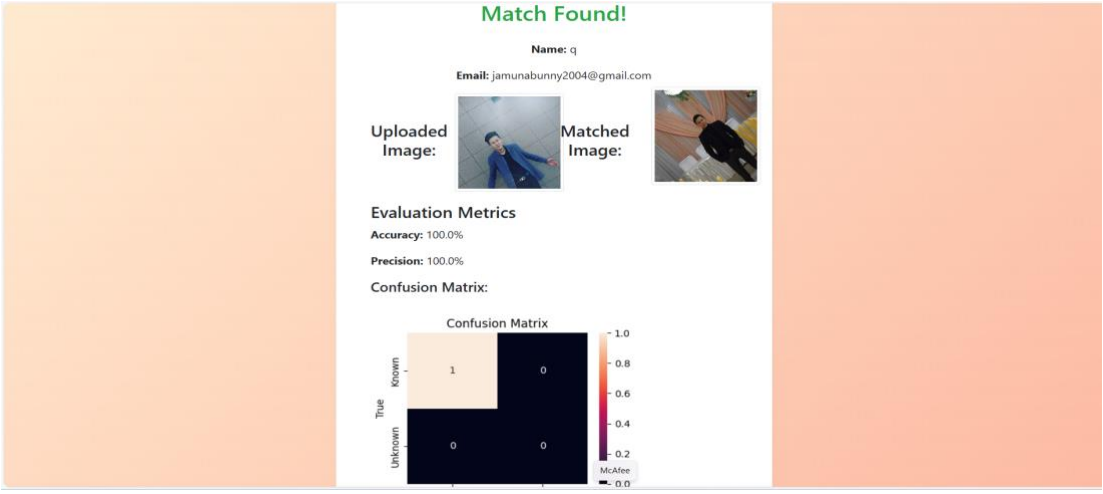


Fig. 6.4 match found page

Fig. 6.4 displays the search results after successfully identifying a missing child. It shows both the uploaded and matched images, along with the name and email of the identified individual. Additionally, it presents evaluation metrics like accuracy and precision, and a confusion matrix for performance assessment.

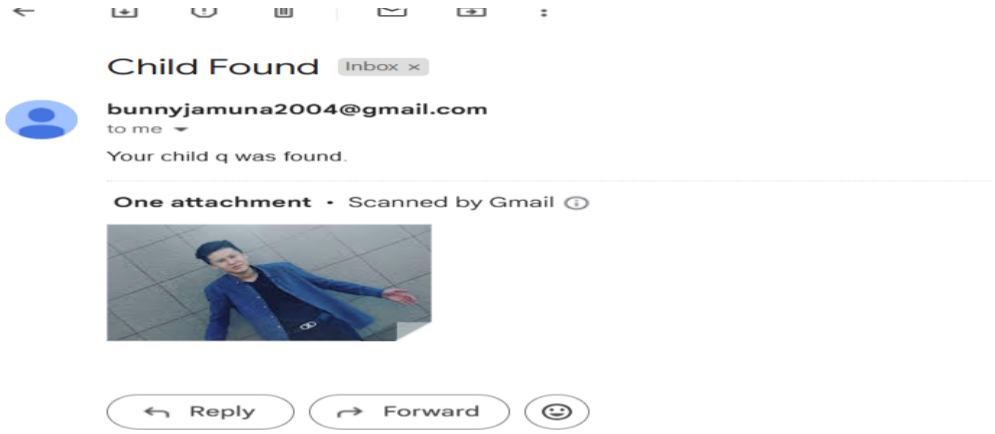


Fig. 6.5 Email Notification – Child Found

Fig. 6.5 This email is automatically sent to the parent or guardian when a missing child is successfully identified. It includes a confirmation message with the child's name and an attached image of the found child for verification. This feature ensures timely and direct communication with the concerned family.

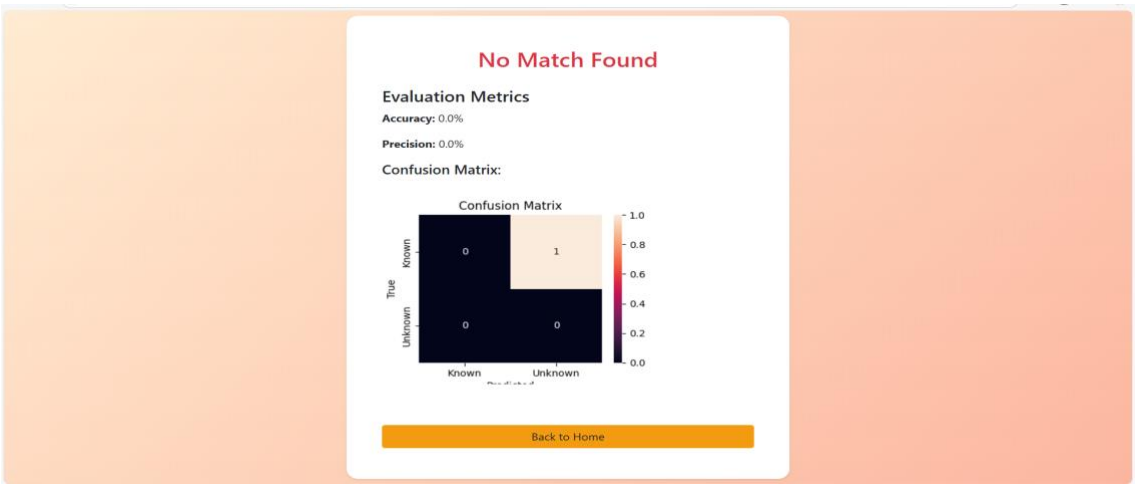


Fig:6.6 No Match Found

Fig.6.6 This page indicates that the system was unable to find a match for the uploaded image in the existing database. It displays evaluation metrics with 0% accuracy and precision, along with a confusion matrix to visually represent the unsuccessful prediction. A button is provided to return to the home screen for another search attempt.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

This web application effectively utilizes face recognition technology to identify missing children and notify their parents via email. With its user-friendly interface and robust functionality, it demonstrates a practical approach to leveraging facial recognition for child safety and recovery. The system's ability to quickly and accurately identify missing children can significantly enhance the efficiency of search efforts and provide timely reunions. Future enhancements can further improve its effectiveness and efficiency, making it an invaluable tool for authorities, families, and communities worldwide. By using the power of technology, we can work towards creating a safer environment for children and providing peace of mind for their loved ones..

7.2 FUTURE ENHANCEMENTS

- Improved Face Recognition Accuracy - Enhance the system by using advanced CNN models for better accuracy under varied lighting and aging effects**.**
- Real-Time Video Processing - Enable support for analyzing live CCTV footage to identify missing children in real time.
- Multi-Face Detection in a Single Image - Allow the system to detect and match multiple faces in a group photo, improving usability in real-world scenarios.
- Mobile Application - Create a mobile app for faster field use by police, NGOs, and volunteers.
- Integration with Government Databases - Link the system with official child welfare or police databases for wider and verified reach.

REFERENCES

[1]. O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," *British Machine Vision Conference (BMVC)*, 2015.

[2]. F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.

[3]. G. V. Prasad and R. S. Nair, "Smart web application for facial recognition using Flask and OpenCV," in *2020 IEEE International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*, Chennai, India, 2020, pp. 1–6.

[4] .Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, OH, USA, 2014, pp. 1701–1708.

[5]. T. P. Lakshmi and K. R. Rani, "Smart security system using image processing and email alerts," in *2016 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, Bengaluru, India, 2016, pp. 176–180

[6]. B. Nagamani, R. Kumar, and A. Sinha, "CNN with transfer learning for missing child identification," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 14, no. 3, pp. 100–107, Mar. 2025.

[7]. S. Sharma, R. Mehta, and P. Roy, "Deep CNN with face re-identification for missing children," *Expert Systems with Applications*, Elsevier, vol. 215, p. 119327, 2023.

[8]. Y. Zhang, L. Wu, and X. Chen, "GAN-based facial age progression framework," *IEEE Access*, vol. 10, pp. 109212–109222, 2022.

[9]. M. Ahmed, A. Patel, and S. Singh, “Hybrid deep learning and SIFT for child face recognition,” *Multimedia Tools and Applications*, Springer, vol. 80, no. 15, pp. 22819–22835, 2021.

[10]. P. Reddy, K. Sharma, and L. Thomas, “Mobile app-based solution using cloud and facial recognition,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 16, no. 3s, pp. 1–19, 2020.

[11]. S. Banerjee and A. Kumar, “IoT and GPS wearable tech integrated with facial tracking,” *IEEE Sensors Journal*, vol. 19, no. 15, pp. 5783–5790, 2019.