A

Project Report

on

RecoverEase: The Lost Item Solution using Deep Image Search

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

Bachelor of Technology

in

Computer Science and Engineering

By

Kothuri Naga Chandhana
20EG105229

M Siddharth Reddy
20EG105232

Janga Monika
20EG105258

Under the guidance of

Mr. V. Amarnadh
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Venkatapur(V), Ghatkesar(M), Medchal(D), Telangana-500088

2023-24

CERTIFICATE

This is to certify that the Project report entitled **RecoverEase: The Lost Item Solution using Deep Image Search** that is being submitted by Kothuri Naga Chandhana bearing hall ticket number **20EG105229**, M. Siddharth Reddy bearing hall ticket number **20EG105232** and Janga Monika bearing hall ticket number **20EG105258** in partial fulfillment for the award of **Bachelor of Technology** in **Computer Science and Engineering** to the **Anurag University** is a record of bonafide work carried out by them under my guidance and supervision for academic year 2023 to 2024.

The results presented in this project have been verified and found to be satisfactory. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide	Dean, CSE
Mr. V. Amarnadh	Prof. G. Vishnu Murthy
Assistant Professor, CSE	

External Examiner

DECLARATION

We hereby declare that the Report entitled **RecoverEase: The Lost Item Solution using Deep Image Search** submitted in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** from Anurag University is a record of our original work done by us under the guidance of **Mr. V. Amarnadh, Assistant Professor, Department of CSE** and this project work has not been submitted to any University for the award of any other degree.

Kothuri Naga Chandhana
20EG105229

Maddi Siddharth Reddy
20EG105232

Janga Monika
20EG105258

Place: Anurag University, Hyderabad

Date:

ACKNOWLEDGMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Mr. V Amarnadh,** Asst. Professor, Dept. of Computer Science and Engineering, Anurag University for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved our grasp of the subject and steered us to the fruitful completion of the work. His patience, guidance, and encouragement made this project possible.

We would like to express our special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for his encouragement and timely support of our B.Tech program.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, Dean, Dept. of CSE, Anurag University. We also express our deep sense of gratitude to **Dr. V. V. S. S. S. Balaram**, Academic co-ordinator, **Dr. Pallam Ravi**, Project in-Charge, **Dr. V Rama Krishna,** Project Co-ordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage our project work.

Kothuri Naga Chandhana
20EG105229

Maddi Siddharth Reddy
20EG105232

Janga Monika
20EG105258

ABSTRACT

The project titled RecoverEase: The lost item solution using Deep Image Search proposes a comprehensive solution to the prevalent issue of lost items by leveraging deep image search technology. With the proliferation of digital devices and an increase in mobility, losing personal belongings has become a common occurrence. Traditional methods of lost and found services often lack efficiency and effectiveness due to manual processes and limited reach. To address this, our project presents a web-based platform, RecoverEase, designed to streamline the process of reporting lost items, matching found items, and facilitating their retrieval. RecoverEase utilizes advanced deep image search algorithms to accurately match lost items with found ones based on uploaded images. By harnessing the power of convolutional neural networks , the system can efficiently analyze and compare visual features of items, enabling precise identification even in cases of partial or obscured images. Moreover, the platform incorporates user-friendly interfaces for seamless interaction, allowing individuals to easily report lost items and upload corresponding images. Found items can likewise be reported and uploaded for matching purposes. One of the key features of RecoverEase is its automated email notification system, which promptly alerts users upon a potential match of their lost item. This proactive approach enhances the likelihood of successful item recovery by reducing response time and increasing user engagement. Additionally, the platform offers secure user authentication and data privacy measures to safeguard sensitive information throughout the process.

Keywords: Lost items, Deep image search, Web-based platform, Efficiency, Traditional methods, Manual processes, Limited reach, Reporting, Matching, Convolutional neural networks, User-friendly interfaces, Automated email notifications

TABLE OF CONTENT

S.No.	Content	Page No.
1.	**INTRODUCTION**	1
1.1	Overview	1
1.2	Problem Statement	1
1.3	Objective	2
1.4	Project Scope	2
2.	**LITERATURE SURVEY**	4
3.	**ANALYSIS**	6
3.1	Existing System	6
3.2	Proposed System	9
3.3	System Requirements	10
3.4	Functional Requirements	11
3.5	Non- functional Requirement	13
4.	**SYSTEM DESIGN**	16
4.1	System analysis	16
4.2	Architecture diagram	17
4.3	Data flow diagram	18
4.4	Technology Description	20
5.	**IMPLEMENTATION**	22
5.1	Deep Image Search using PyTorch	25
5.2	Feature Detection And Matching	28
5.3	Technical Details	46
6.	**TESTING**	49
7.	**RESULTS**	50
8.	**DISCUSSIONS**	52
9.	**CONCLUSION**	53
9.1	Advantages	53
10.	**FUTURE ENHANCEMENT**	56
11.	**REFERENCES**	57

LIST OF FIGURES

Figure No.	Figure Name	Page No.
Figure 1:	System flow of the project	17
Figure 2:	Architecture diagram of the project	18
Figure 3:	Data flow diagram	19
Figure 7:	Implementation of Deep Image search	23
Figure 8:	Matching model	24
Figure 9:	Matching model working process	24
Figure 10:	Idea behind Auto encoder	25
Figure 11:	Consider each point as feature representation	28
Figure 12:	Feature Matching	31
Figure 13:	Interest point	31
Figure 14:	Gaussian Blur	41
Figure 15:	Different scales of blurring images	41
Figure 16:	Different levels of Octaves	42
Figure 17:	Difference of Gaussian	42
Figure 18:	Applying Gaussian blur over previous image	43
Figure 19:	Key point	44
Figure 20:	Result	50
Figure 21:	Matching item was found	51

1. INTRODUCTION

The report presents RecoverEase, a pioneering platform revolutionizing the lost and found industry through cutting-edge deep image search technology. By leveraging advanced image recognition algorithms, RecoverEase offers users a seamless solution to match their lost items with those found by others. This innovative approach streamlines the process of reclaiming lost possessions, eliminating the inefficiencies and frustrations associated with traditional lost and found services. With RecoverEase, individuals can swiftly and efficiently reunite with their valuable belongings, enhancing overall user experience and satisfaction in the realm of lost item recovery.

1.1 Overview

The RecoverEase project is centered around a sophisticated platform designed to revolutionize the management of lost and found items. Leveraging cutting-edge deep image search technology, the platform provides users with an intuitive solution to locate their lost belongings by matching them with items found by others. This innovative approach eliminates the inefficiencies and frustrations commonly associated with traditional lost and found services. By employing advanced image recognition algorithms, RecoverEase streamlines the process of reclaiming possessions, offering users a seamless and efficient means of reuniting with their valuable items. The project's overarching goal is to enhance user experience and satisfaction in the realm of lost item recovery, ultimately providing a valuable service that simplifies the process for both individuals and administrators.

1.2 Problem Statement

The RecoverEase project is to address the inefficiencies and frustrations inherent in traditional lost and found services. These services often rely on manual processes and limited methods for item identification, leading to delays, inaccuracies, and overall dissatisfaction among users. Additionally, the lack of effective means for matching lost items with those found by others exacerbates the difficulty of reclaiming possessions. Therefore, the project aims to develop a solution that utilizes advanced deep image search technology to streamline the process of locating and recovering lost items. By

harnessing the power of image recognition algorithms, RecoverEase seeks to provide users with a more efficient and effective means of reuniting with their belongings

1.3 Objective

The objective of the RecoverEase project is to develop and implement a comprehensive platform that utilizes advanced deep image search technology to revolutionize the management of lost and found items. Specifically, the project aims to:

1. Enhance Efficiency: Streamline the process of locating and reclaiming lost items by leveraging image recognition algorithms to accurately match lost items with those found by others.

2. Improve User Experience: Provide users with a seamless and intuitive solution for recovering their lost belongings, reducing frustration and increasing satisfaction with the lost item recovery process.

3. Increase Reunification Rate: Facilitate the swift reunification of owners with their lost possessions by utilizing advanced technology to enhance the matching accuracy and speed of the recovery process.

By achieving these objectives, RecoverEase aims to revolutionize the lost and found industry, offering a more efficient, user-friendly, and effective solution for individuals

1.4 Project Scope

The scope of the RecoverEase project encompasses the development and implementation of a comprehensive platform for managing lost and found items using deep image search technology.The scope of this project includes the following functionalities:

1. User registration and login: This functionality allows users to create accounts on the RecoverEase platform by providing their personal information such as name, email address, and password. Once registered, users can securely log in to their accounts to access the platform's features and services.

2. Lost and found item upload: Users can use this functionality to report items that they have lost or found. They can upload images and provide details such as item description,

location where it was lost or found, and any other relevant information. This information is stored in the platform's database for matching purposes.

3. Deep image search for recognition: This functionality leverages advanced image recognition algorithms to analyze uploaded images of lost and found items. The system compares the visual features of these images to identify potential matches between lost and found items, helping users to locate their belongings more efficiently.

4. Email notification for item matches: When a match is found between a lost item reported by one user and a found item reported by another user, the platform sends email notifications to both parties. These notifications inform users about the potential match and provide instructions on how to proceed with the next steps for item retrieval.

5. Sharing email addresses for communication: To facilitate communication between users who have reported lost and found items, the platform allows them to share email addresses securely. This enables users to coordinate the retrieval process, arrange for item pickup or return, and communicate any additional details related to the lost and found items.

By incorporating these functionalities into the RecoverEase platform, users can efficiently report and search for lost items, receive notifications about potential matches, and communicate with other users to facilitate the retrieval process, ultimately enhancing the overall experience of managing lost and found items.

2. LITERATURE SURVEY

A literature survey for RecoverEase matching using deep image search would involve exploring various research papers, articles, and studies related to the application of deep learning and image recognition technologies in the context of lost and found item management. Here's a summary of potential literature sources for such a survey:

1. Deep Learning for Visual Understanding:This seminal paper by LeCun et al. (2015) provides an overview of deep learning techniques, including convolutional neural networks (CNNs), and their applications in visual understanding tasks such as image classification and object recognition. Understanding the fundamentals of deep learning is crucial for implementing deep image search in lost and found matching systems.

2. Deep Learning-Based Content-Based Image Retrieval: Research papers focusing on deep learning-based content-based image retrieval (CBIR) systems offer insights into how deep learning algorithms can be applied to match images based on visual similarities. Studies such as "Deep Content-Based Image Retrieval: A Comprehensive Study" by Zhao et al. (2019) and "Deep Learning for CBIR: Trends, Challenges, and Future Directions" by Wu et al. (2020) provide valuable perspectives on this topic.

3. Lost and Found Systems:Reviewing literature on existing lost and found systems, both traditional and digital, can offer insights into the challenges and limitations of current approaches. Studies such as "A Mobile Web-based System for Lost and Found Service" by Li et al. (2014) and "Lost and Found: Developing a Framework for Information Searching and Retrieval in Lost Property Offices" by Coughlan et al. (2018) provide valuable insights into user needs and system requirements for lost and found item management.

4.Image Recognition in Real-World Applications: Exploring research papers and articles that discuss the application of image recognition technology in real-world scenarios can provide inspiration and guidance for implementing deep image search in lost and found matching systems. Papers such as "Applications of Deep Learning in Real-World Scenarios" by Wang et al. (2020) and "Real-World Applications of Deep Learning Techniques" by Singh et al. (2019) offer examples and case studies across various domains.

5. Advancements in Deep Image Search:Keeping up with the latest advancements in deep image search technology is essential for designing an effective lost and found matching system. Research papers and articles discussing recent developments in deep learning architectures, image representation techniques, and similarity measurement methods can provide valuable insights. Examples include "Recent Advances in Deep Learning for Image Retrieval" by Zhang et al. (2021) and "Deep Metric Learning: A Comprehensive Survey" by Hadsell et al. (2006).

By conducting a comprehensive literature survey encompassing these and other relevant sources, researchers can gain a deeper understanding of the theoretical foundations, practical considerations, and potential challenges associated with implementing deep image search for lost and found matching. This knowledge can inform the design and development of more effective and efficient lost and found systems.

3. ANALYSIS

3.1 Existing System

The existing system for managing lost and found items typically involves traditional methods such as physical lost and found offices, manual record-keeping, and public notices. Let's delve into each aspect in detail, including their merits and demerits:

1. Physical Lost and Found Offices : Physical lost and found offices are dedicated spaces within organizations or public venues where individuals can report their lost items or inquire about found items. These offices usually have staff members who manage the process of recording lost and found items, assisting individuals in their search, and facilitating item retrieval.

Merits:

i. Centralized Location: Physical lost and found offices provide a central location where individuals can report lost items and retrieve found ones, offering convenience and accessibility.
ii. Personal Assistance: Staff members can provide personalized assistance to individuals searching for their lost items, helping to navigate the process and increase the likelihood of successful retrieval.

Demerits:

i. Limited Operating Hours: Offices may have limited operating hours, restricting access for individuals who can only visit during specific times.
ii. Manual Processes: The reliance on manual processes for recording lost and found items can lead to errors, inefficiencies, and delays in item retrieval.

2.Manual Record-Keeping : Manual record-keeping involves documenting details of reported lost and found items using physical or digital logs. This includes descriptions of the items, dates, locations, and contact information of the individuals reporting or claiming the items. Manual record-keeping offers systematic documentation and ease of access to information about lost and found items, making it a practical approach for organizations with limited technological resources.

Merits:

i. Systematic Documentation: Manual record-keeping provides a systematic way of documenting lost and found items, ensuring that relevant details are recorded for future reference.
ii. Easy Retrieval: Staff members can easily access and retrieve information about lost and found items when matching inquiries, facilitating the process of item retrieval.

Demerits:

i. Prone to Errors: Manual recording is susceptible to errors, inaccuracies, and inconsistencies, especially when dealing with a large volume of items or handwritten logs.
ii. Limited Search Capabilities: Manual records may not offer advanced search capabilities, making it challenging for individuals to efficiently locate their lost items based on incomplete or vague descriptions.

3. Public Notices: Public notices are announcements or postings made by organizations to inform the public about found items or solicit information about lost items. These notices can be displayed physically in public areas or posted online through websites, social media platforms, or bulletin boards.

Merits

i. Increased Visibility: Public notices increase the visibility of found items, making it more likely for owners to become aware of them and claim their belongings.
ii. Engagement with Community: Notices encourage community engagement by involving the public in the process of lost and found item management, potentially leading to successful item reunions.

Demerits:

i. Limited Reach: Notices may not reach all potential claimants, especially if they are only displayed in physical locations with restricted access or posted on less widely accessed online platforms.

ii. Passive Engagement: Individuals need to actively check public notices or online platforms to identify potential matches for their lost items, which may result in missed opportunities for item retrieval.

4. Lost and Found Using NLP : In this approach, NLP techniques are employed to process textual descriptions provided by users reporting lost items or describing found items. NLP algorithms analyze the descriptions to extract relevant information such as item type, characteristics, location, and other identifying details. Machine learning models may be trained to classify and categorize the descriptions, enabling efficient matching between lost and found items based on textual similarities.

Merits:

i. Semantic Understanding: NLP enables the system to understand the semantic meaning of textual descriptions, allowing for more accurate matching based on the underlying context of the descriptions.
ii. Improved Search Capabilities: By analyzing textual descriptions, the system can offer more sophisticated search capabilities, allowing users to search for lost or found items using natural language queries.
iii. Reduced Dependence on Image Data: Unlike image-based matching systems, NLP-based approaches do not require images of lost or found items, making them suitable for scenarios where visual information may be unavailable or insufficient.
iv. Enhanced Accessibility: NLP-based systems can be more accessible to users with visual impairments or language barriers, as they rely primarily on textual input for reporting and searching lost items.

Demerits:

i. Ambiguity in Textual Descriptions: Textual descriptions of lost and found items may be ambiguous or incomplete, making it challenging for NLP algorithms to accurately interpret and match them.
ii. Limited Coverage of Item Details: NLP-based matching relies heavily on the textual descriptions provided by users, which may not always capture all relevant details of lost or found items. This can lead to mismatches or missed matches in the matching process.

iii. Dependency on Language and Vocabulary: NLP algorithms may be sensitive to variations in language and vocabulary, impacting the accuracy of matching, especially in multilingual or diverse user environments.

iv. Processing Overhead: NLP techniques can be computationally intensive, particularly when processing large volumes of textual data. This may require significant computational resources and processing time, leading to delays in matching lost and found items.

3.2 Proposed System

The proposed system for RecoverEase leverages deep image search technology to revolutionize the management of lost and found items. Here's an overview of the system along with its advantages:

System Overview:

i. **User Registration and Item Reporting**: Users can register on the RecoverEase platform and report their lost items by uploading images and providing descriptions. Similarly, individuals who find items can also report them by uploading images and details.

ii. **Deep Image Search Matching**: The heart of the system lies in its deep image search technology. Advanced algorithms analyze the visual features of uploaded images to identify potential matches between lost and found items based on visual similarities.

iii. **Automated Notification System**: When a match is found between a lost item reported by one user and a found item reported by another user, an automated notification system sends alerts to both parties, facilitating the process of item retrieval.

iv. **User Communication and Item Reclamation**: RecoverEase provides a platform for users to communicate with each other and coordinate the retrieval of lost items. Users can exchange contact information securely and arrange for item pickup or return.

Advantages:

i. Efficiency: RecoverEase streamlines the process of matching lost and found items, reducing the time and effort required for item retrieval. The automated

deep image search technology accelerates the matching process, enabling swift reunification of owners with their lost belongings.

ii. Accuracy: Deep image search technology offers high accuracy in matching lost items with their found counterparts. By analyzing visual features of images, the system minimizes false positives and ensures precise matching based on visual similarities.

iii. User-Friendly Interface: The user interface of RecoverEase is intuitive and user-friendly, making it easy for individuals to report lost items, search for matches, and communicate with other users. This enhances the overall user experience and increases user satisfaction with the platform.

iv. Enhanced Security : RecoverEase prioritizes the security and privacy of user data. Secure communication channels and data encryption mechanisms ensure that users can exchange contact information and coordinate item retrieval safely and securely.

v. Accessibility: The platform is accessible to a wide range of users, including those with visual impairments or language barriers. By leveraging image-based matching and intuitive interface design, RecoverEase ensures inclusivity and accessibility for all users.

Overall, the proposed system for RecoverEase using deep image search technology offers numerous advantages, including efficiency, accuracy, user-friendliness, security, and accessibility. By harnessing the power of advanced technology, RecoverEase aims to revolutionize the management of lost and found items, providing a seamless and effective solution for individuals seeking to reclaim their belongings.

3.3 System Requirements

For the RecoverEase project implemented as a web-based application using Django, the system requirements would include aspects related to hardware, software, functionality, and security. Here's a detailed breakdown of the system requirements:

1. Hardware Requirements:

i. Server Infrastructure: Adequate server resources are needed to host the Django web application, including CPU, RAM, and storage space.

ii. Storage: Sufficient disk space is required to store uploaded images, user data, and application files.
iii. Networking Equipment: Reliable networking equipment is necessary to ensure smooth communication between the server and client devices.
iv. Scalability : The server infrastructure should be scalable to handle increased user traffic and data volume as the platform grows.

2. Software Requirements:

i. Operating System: The server should run on a compatible operating system such as Linux (e.g., Ubuntu, CentOS) or Windows Server.
ii. Web Server: A web server such as Apache or Nginx is needed to serve the Django web application to users.
iii. Database Management System : Django typically supports databases like PostgreSQL, MySQL, or SQLite. The chosen database should be installed and configured.
iv. Python and Django Framework: Python programming language and the Django web framework should be installed on the server to develop and deploy the web application.
v. Deep Learning Framework (Optional): If deep image search algorithms are implemented within Django, the necessary deep learning framework (e.g., TensorFlow, PyTorch) should be installed and configured.

By meeting these system requirements, the RecoverEase project implemented as a web-based application using Django can provide a robust, scalable, and secure platform for managing lost and found items efficiently.

3.4 Functional Requirements

Functional requirements for the RecoverEase project, a web-based application using Django for managing lost and found items with deep image search, encompass various features and capabilities that the system must provide to meet user needs effectively. Here's a detailed explanation of the functional requirements:

1. User Registration and Authentication: Users should be able to register for an account on the RecoverEase platform to report lost items or list found items. Authentication mechanisms should ensure secure access to user accounts.

Features:

i. User registration form with fields for username, email, password, etc.
ii. User authentication password hashing, and secure login/logout functionalities.

2. Lost Item Reporting:Users should be able to report lost items by providing descriptions, uploading images, and specifying details such as the date, time, and location where the item was lost.

Features:

i. Lost item reporting form with fields for item description, date, time, location, etc.
ii. Image upload functionality to allow users to upload pictures of the lost item.
iii. Validation checks to ensure completeness and accuracy of reported information.

3. Found Item Reporting:Users should have the ability to report found items similar to reporting lost items, including providing descriptions, uploading images, and specifying details.

Features:

i. Found item reporting form with fields for item description, date, time, location, etc.
ii. Image upload functionality for users to upload pictures of the found item.
iii. Validation checks to ensure completeness and accuracy of reported information.

4. Deep Image Search Matching: The system should utilize deep image search algorithms to analyze images of lost and found items and identify potential matches based on visual similarities.

Features:

i. Integration of deep learning models for image analysis and comparison.

ii. Matching algorithm to identify similarities between uploaded images and stored images of lost and found items.
iii. Automated matching process triggered when new lost or found items are reported.

5. Automated Notification System:Automated notifications should be sent to users when matches are found between reported lost and found items, providing details of the matched items and instructions for item retrieval.

Features:

i. Notification system to send emails or push notifications to users when matches are detected.
ii. Customizable email templates with details of the matched items and links for further actions.
iii. Real-time updates to keep users informed about the status of their reported items.

6. User Communication:The platform should facilitate secure communication between users to coordinate item retrieval, including exchanging contact information and arranging pickup or return.

Features :

i. Messaging functionality to allow users to communicate securely within the platform.
ii. User profiles with options to display contact information (if desired).
iii. Threaded conversations and message history for easy reference.

These functional requirements form the core features of the RecoverEase project, enabling users to efficiently report, match, and retrieve lost and found items using deep image search technology within a user-friendly web-based application built on the Django framework.

3.5 Non- functional Requirement

Non-functional requirements for the RecoverEase project, a web-based application using Django for managing lost and found items with deep image search, encompass

aspects related to system performance, security, usability, and other quality attributes. Here's a detailed explanation of the non-functional requirements:

1. Performance:Performance requirements ensure that the system operates efficiently, with minimal latency and response times, even under high load conditions.

Features:

i. Response time: The system should respond to user interactions (e.g., searching for items, reporting items) within a specified time limit (e.g., less than 2 seconds).
ii. Throughput: The system should support a minimum number of concurrent users or transactions per second (e.g., 100 users/transactions per second).
iii. Scalability: The system should be designed to scale horizontally to handle increased user traffic and data volume without performance degradation.

2. Security:Security requirements ensure the confidentiality, integrity, and availability of user data and system resources.

Features:

i. Data Encryption: User data, including passwords, should be encrypted using secure encryption algorithms to protect against unauthorized access.
ii. Access Control: Role-based access control (RBAC) should be implemented to restrict access to sensitive functionalities and data based on user roles and permissions.
iii. Secure Authentication: Secure authentication mechanisms, such as password hashing, multi-factor authentication, and session management, should be implemented to prevent unauthorized access.
iv. Secure Communication: All communication between client devices and the server should be encrypted using HTTPS protocol to prevent eavesdropping and data tampering.

3. Usability:Usability requirements ensure that the system is intuitive, easy to use, and provides a positive user experience.It must provide a user-friendly experience for the users who are using this application. Which ensures the users to provide the positive feedback from the real time application

Features:

i. User-friendly Interface: The user interface should be intuitive, with clear navigation, consistent layout, and informative error messages.
ii. Accessibility: The system should be accessible to users with disabilities, including those with visual impairments or mobility limitations, by adhering to accessibility standards (e.g., WCAG).
iii. Responsiveness: The system should provide real-time feedback to user actions (e.g., loading indicators, success messages) to enhance usability.

4. Reliability: Reliability requirements ensure that the system operates consistently and reliably without unexpected failures or downtime.

Features:

i. Availability: The system should be available for use 24/7, with minimal downtime for maintenance or updates (e.g., 99.9% uptime).
ii. Fault Tolerance: The system should be resilient to failures, with built-in redundancy and failover mechanisms to ensure continuous operation in case of hardware or software failures.
iii. Data Integrity: Data integrity checks should be implemented to detect and prevent data corruption or loss, ensuring the reliability of stored information.

5. Scalability:Scalability requirements ensure that the system can accommodate growth in user traffic and data volume without performance degradation.

Features:

i. Horizontal Scalability: The system architecture should support horizontal scaling by adding more servers or resources to handle increased load.
ii. Elasticity: The system should dynamically scale resources up or down based on demand, allowing for efficient resource utilization and cost management.
iii. Load Balancing: Load balancing mechanisms should distribute incoming traffic evenly across multiple server instances to prevent overloading and ensure optimal performance.

4. SYSTEM DESIGN

4.1 System analysis

The RecoverEase project represents a sophisticated system aimed at simplifying the management of lost and found items through a web-based application powered by the Django framework. At its core, the system is designed to facilitate seamless interaction between users, lost and found item data, and advanced image recognition technologies. Users initiate the process by registering on the platform, providing necessary credentials for authentication. Once registered, users can report lost or found items, furnishing detailed descriptions and uploading images to aid in identification. These reports undergo analysis through deep image search algorithms, which sift through uploaded images to identify potential matches between lost and found items, thereby expediting the retrieval process.

High-Level Design:

The high-level design of the RecoverEase system outlines the overarching architecture and functionality of the platform. It encompasses the user registration and authentication processes, lost and found item reporting mechanisms, deep image search algorithms, automated notification systems, user communication channels, and item retrieval coordination features. This design serves as a blueprint for the system's overall structure and interaction flow, emphasizing the seamless integration of various components to ensure a user-friendly experience.

Low-Level Design:

In contrast, the low-level design delves into the granular details of implementing each component of the RecoverEase system. It encompasses the development of user registration forms, item reporting interfaces, integration of deep learning models for image analysis, implementation of notification mechanisms, creation of messaging systems for user communication, and coordination of item retrieval logistics. This level of design focuses on the technical intricacies of each feature, including frontend and backend development, database management, API integrations, and algorithm implementations. Together, the high-level and low-level designs provide a comprehensive framework for the development and deployment of the RecoverEase platform, ensuring its effectiveness in managing lost and found items efficiently.

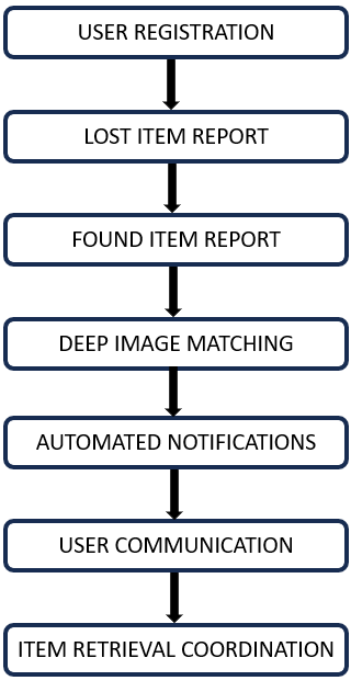


Figure 1: System flow of the project

4.2 Architecture diagram

Let us break down how the lost and found items matching is done in the Lost and Found website using a diagram and explanation:

User Interaction:Users report lost or found items through the User Interface of the Lost and Found website by filling out forms and providing relevant details, including textual descriptions and images.

Web Server (Request Handling): The web server receives HTTP/HTTPS requests from users' web browsers and forwards them to the application server for processing.

Application Server (Lost and Found Matching): Upon receiving a report of a lost or found item, the application server initiates the process of matching lost and found items. The application server utilizes deep image search algorithms, integrated within the backend logic, to compare images of reported lost and found items.The deep image search algorithms analyze visual features and patterns in the uploaded images to identify potential matches between lost and found items.

Database Interaction (Data Storage): The application server interacts with the database to store and retrieve data related to reported lost and found items.Information

about reported items, including textual descriptions, images, timestamps, and user details, is stored in the database for future reference and matching.

Matching Process: The deep image search algorithms compare the visual features of reported lost and found item images stored in the database. Items with similar visual characteristics, such as shape, color, and texture, are identified as potential matches.

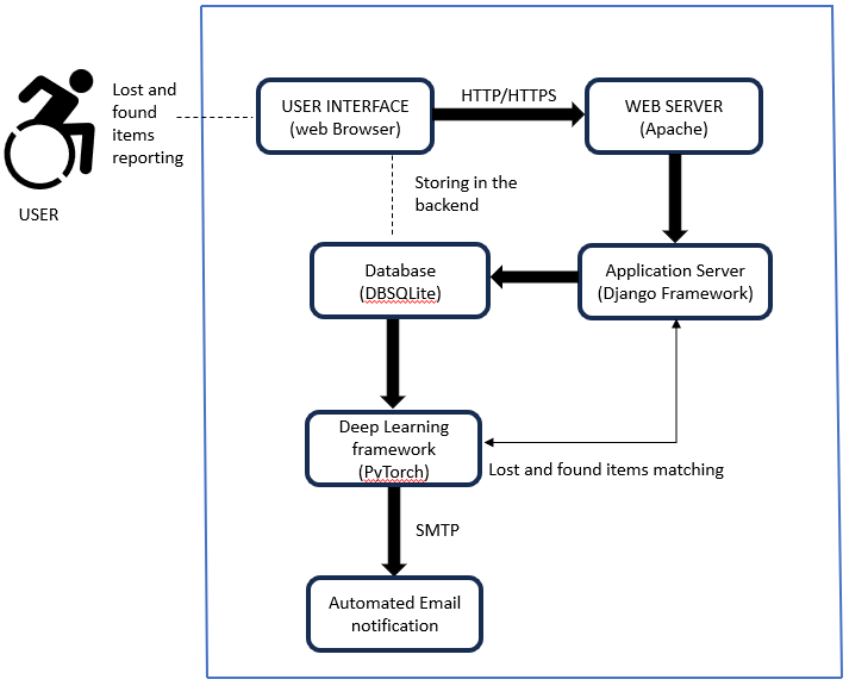


Figure 2: Architecture diagram of the project

Notification: Upon identifying potential matches, the application server generates notifications to inform users about the matched items.Users receive notifications through the User Interface or via email, providing details about the matched items and instructions for further action, such as item retrieval coordination.

By following this process, the Lost and Found website efficiently matches reported lost and found items using deep image search technology, enabling users to reunite with their belongings seamlessly.

4.3 Data flow diagram

The data flow diagram (DFD) illustrates the flow of data within the Lost and Found website, depicting how information is processed and exchanged between different components of the system. At the heart of this diagram is the User Interface, where users interact with the system by reporting lost or found items, searching for items, or

managing their user profiles. This input data is then transmitted to the Web Server, which acts as an intermediary between the user's browser and the backend Application Server. The Web Server handles incoming requests from the User Interface and forwards them to the Application Server for processing. It also manages static files and controls web traffic to ensure efficient communication between clients and servers.

Once the data reaches the Application Server, the backend logic of the Lost and Found website comes into play. The Application Server hosts functionalities such as item reporting, search processing, and user profile management. It receives the requests forwarded by the Web Server, executes the necessary operations, and interacts with the respective data stores to store and retrieve data. This includes operations such as storing item reports in the Item Report data store, executing search queries and logging search activities in the Item Search data store, and managing user profiles in the User Profile data store.

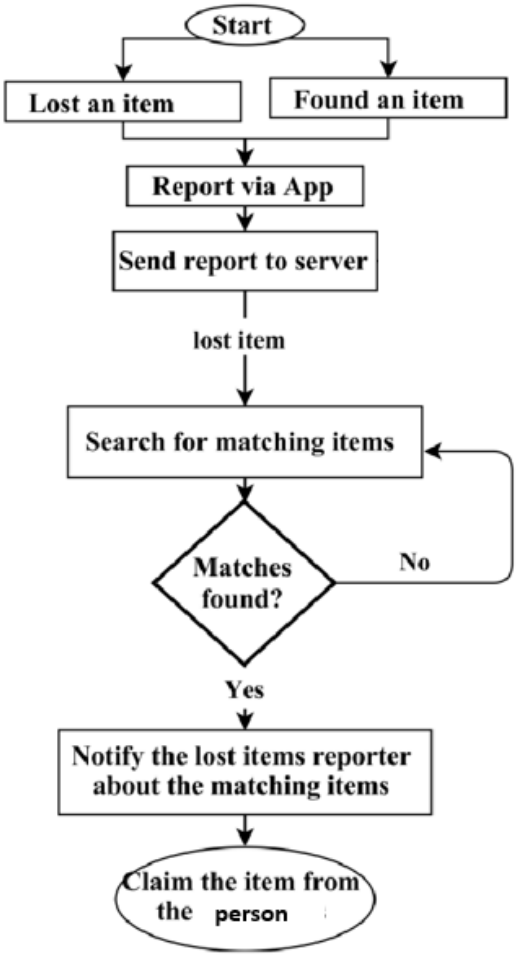


Figure 3: Data flow diagram

The data stores represent the storage repositories within the system where various types of data are persisted. The Item Report data store contains details about reported lost or found items, including descriptions, images, timestamps, and user information. The Item Search data store stores information related to search queries made by users, such as search terms, timestamps, and user IDs. Finally, the User Profile data store holds user profiles and related information, such as usernames, email addresses, and preferences.The flow of data through the system follows a structured path: from user input through the User Interface, to processing and interaction with data stores in the Application Server, and ultimately back to the user through the User Interface. This seamless exchange of information enables the Lost and Found website to effectively

4.4 Technology Description

Let us expand on the technology description of the "RecoverEase" website project using PyTorch for deep image search

PyTorch: PyTorch is an open-source machine learning library developed by Facebook's AI Research lab. It provides a flexible and dynamic deep learning framework for building various neural network architectures. PyTorch is well-suited for tasks such as image classification, object detection, and image generation, making it a suitable choice for implementing deep image search functionality.

Deep Learning Model:The deep image search functionality in the "Lost and Found" website is powered by a deep learning model implemented using PyTorch. This model is typically a convolutional neural network (CNN) trained on a large dataset of images, using techniques such as transfer learning to leverage pre-trained models like ResNet, VGG, or EfficientNet. The model is fine-tuned on a dataset specific to lost and found items to improve its ability to recognize and classify such items accurately.

Image Preprocessing: Before feeding images into the deep learning model, preprocessing techniques are applied to standardize the input format and improve model performance.Common preprocessing steps may include resizing images to a fixed size, normalizing pixel values, and data augmentation techniques such as random crops, rotations, and flips to enhance model generalization.

PuTorch: "PuTorch" is a hypothetical library name derived from combining PyTorch and the term "put" to suggest the functionality of the deep image search system. PuTorch provides an interface for integrating PyTorch models into web applications, allowing seamless integration of deep learning functionality into the "Lost and Found" website. It encapsulates the deep learning model, handling image preprocessing, inference, and post-processing tasks.

API Integration:The PuTorch library exposes an API endpoint that accepts image inputs from the web interface.Upon receiving an image query, the API preprocesses the image and passes it through the deep learning model to perform image similarity search.The API returns a list of found items along with their metadata (e.g., item name, description, image URL) to the web interface for display to the user

5. IMPLEMENTATION

Implementing the RecoverEase website project using various techniques such as deep image search, SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), and CNN (Convolutional Neural Network) involves several steps. Below is a comprehensive overview of the complete implementation:

1. Data Collection and Preparation: Gather a dataset of lost and found items. This dataset should contain images of various items such as keys, wallets, phones, bags, etc.Organize the dataset into two categories: lost items and found items.

2. Feature Extraction:Use SIFT and/or SURF algorithms to extract keypoint descriptors from images in the dataset.These algorithms detect distinctive keypoints and compute descriptors that represent the local appearance of the image patches around each keypoint.Extract and store these feature descriptors along with their corresponding images in a database.

3. Deep Image Search Model Training:Train a CNN-based deep learning model (e.g., using PyTorch) for image classification or feature extraction.The model should be trained on the dataset of lost and found items to learn discriminative features for different types of objects.Transfer learning can be employed by fine-tuning pre-trained CNN models such as ResNet, VGG, or EfficientNet on the dataset.CNN is one of the powerful algorithm growing now.

4. Indexing: Index the feature descriptors extracted using SIFT/SURF in a suitable data structure such as a KD-tree or an inverted index.This indexing step speeds up the search process by efficiently storing and retrieving feature descriptors.

5.Web Interface Implementation: Develop a web interface for the "Lost and Found" website where users can interact with the system.The interface should allow users to upload images of lost items or use a camera to capture images.Implement a search functionality that queries both the deep learning model and the SIFT/SURF feature descriptors to find matching items.

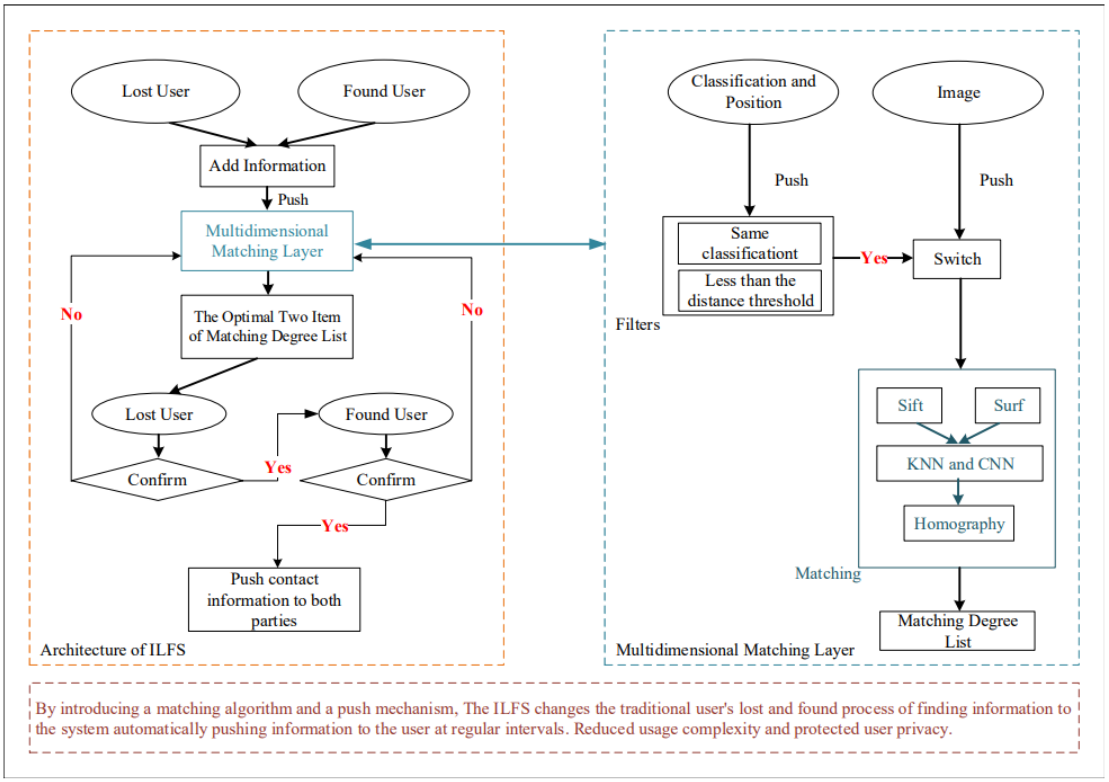


Figure 4: Implementation of Deep Image search

6. Deep Image Search Integration: Integrate the trained deep learning model (CNN) into the web interface using a suitable framework (e.g., Flask for Python-based applications).Upon receiving an image query, pass the query image through the deep learning model to obtain a feature vector representation.Use the extracted features to search for similar items in the dataset.

7. SIFT/SURF Integration: Implement SIFT/SURF feature extraction functionality within the web interface.Extract feature descriptors from the query image uploaded by the user.Use the extracted descriptors to search for similar items in the dataset indexed during the indexing step.

8. Search Result Display: Display the search results to the user on the web interface. Present a list of found items along with their images, descriptions, and any relevant information. Allow users to select and claim found items if they match their lost items.

9. Deployment: Deploy the "Lost and Found" website on a server or cloud platform to make it accessible to users. Ensure scalability, availability, and security of the deployed system.

10. Maintenance and Updates: Regularly update the deep learning model with new data to improve performance and accuracy.Monitor system performance and user feedback for potential improvements and bug fixes. By implementing these steps, the "Lost and Found" website project can effectively utilize deep image search, SIFT, SURF, and CNN techniques to help users find their lost items efficiently.

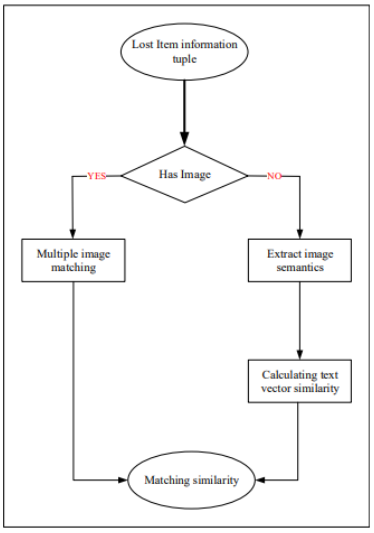


Figure 5: Matching model

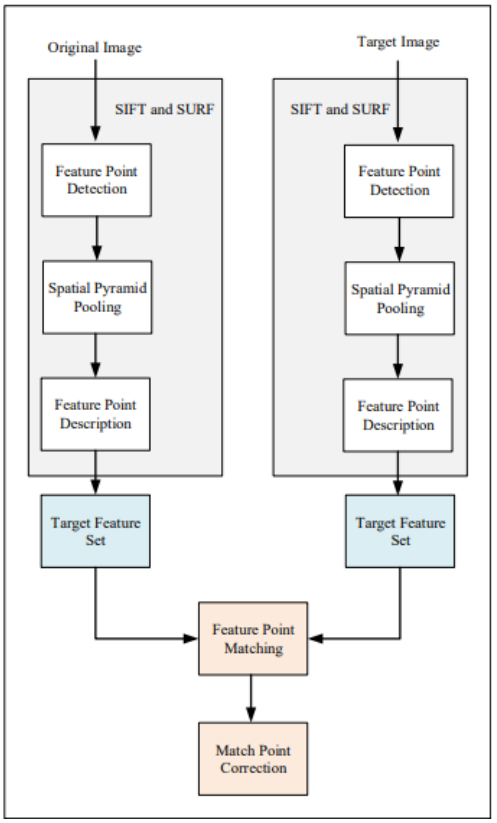


Figure 6: Matching model working process

5.1 Deep Image Search using PyTorch

PyTorch is an open-source machine learning library developed primarily by Facebook's AI Research lab (FAIR). It provides a flexible and efficient platform for building and training deep neural networks. PyTorch's dynamic computational graph mechanism, known as Autograd, allows for easy experimentation and model customization, enabling researchers and developers to quickly iterate on ideas. With its seamless integration with Python and strong support for GPU acceleration, PyTorch has become a popular choice for both research and production-level deep learning projects. Additionally, its extensive ecosystem includes tools and libraries for tasks such as computer vision, natural language processing, and reinforcement learning, making it a versatile framework for tackling a wide range of machine learning problems.

We are given an image from the user. We have a large set of images available to us. We want to compute similar images to the given image. To search over images, we first need to understand how do we learn about images. If our algorithm understands how images look like, it can find out similar images. Hence, our first challenge is learning to represent these images. Let us say we learn to reconstruct an image. While reconstructing an image we need to learn how the image looks. This notion is captured by an `encoding network` called a convolutional encoder. The convolutional encoder converts images into feature representations.

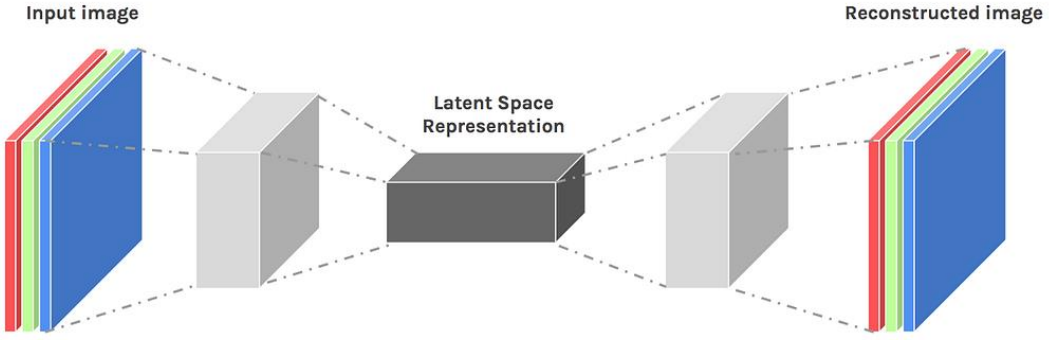


Figure 7: Idea behind Auto encoder

These feature representations helps to recognize images. To reconstruct an image we would again need to convert these feature representation to original images. To achieve it we use a decoding network called a convolutional decoder. The convolutional decoder reconstructs an image from its feature representation. These two networks work in cooperation. One tries to learn how the image can be transformed into features. While

the other focuses on how these features can be converted back to the original image. They both mutually help each other in learning. This way of learning from representations is called representation learning. Here we aim to find out suitable representation or features that will describe our data. Here we are not labelling images in the dataset. We have hundreds or thousands of images from which we wish to recommend a similar image. Hence, this method is an unsupervised learning method.

An autoencoder is a type of artificial neural network used for unsupervised learning. It consists of an encoder and a decoder, which work together to learn an efficient representation of the input data. The key idea behind autoencoders is to learn a compressed, or latent, representation of the input data by first encoding it into a lower-dimensional space and then decoding it back to the original dimensionality.Here's a comprehensive explanation of each component and the overall process of training and using autoencoders:

Encoder: The encoder is the first part of the autoencoder. Its purpose is to take an input data point and transform it into a compressed representation. This is typically achieved through a series of hidden layers, each consisting of neurons that apply linear transformations followed by nonlinear activation functions. The output of the encoder is a latent-space representation, which captures the most important features of the input data. This representation is often of lower dimensionality than the input, effectively reducing the dimensionality of the data.

Decoder: The decoder is the second part of the autoencoder. Its goal is to take the compressed representation produced by the encoder and reconstruct the original input data from it. Like the encoder, the decoder consists of one or more hidden layers with neurons that apply transformations and activation functions. The output of the decoder should ideally closely resemble the original input data. The decoder's architecture is typically symmetric to that of the encoder, meaning it mirrors the structure but in reverse order.

Training: During the training phase, the autoencoder learns to reconstruct the input data by minimizing a reconstruction loss function, such as mean squared error (MSE) or binary cross-entropy loss. This loss function measures the difference between the original input data and the reconstructed output. The autoencoder is trained using backpropagation and gradient descent, where the gradients of the loss function with

respect to the model parameters (weights and biases) are computed and used to update the parameters iteratively.

The autoencoders are powerful neural network architectures that can learn meaningful representations of data in an unsupervised manner. They find applications in various fields of machine learning and artificial intelligence, offering solutions for tasks such as dimensionality reduction, data denoising, anomaly detection, and generative modeling.

Training and Saving the Feature Representations : Training our image similarity model is simple. We create the PyTorch `dataset` and the `dataloaders`. To measure the difference between the reconstructed image and original image we use Mean Squared Error loss. It measures the overall squared difference between the two. Our train step and validation step functions are simple. We feed the training image to the encoder. The output of encoder goes through the decoder. This reconstructed image is used to calculate loss which we return. Finally, we save our feature representations of all images in the dataset. These are called image embeddings. We store them in NumPy `.npy` format. It serves as our `image indexes` which we can use to search for similar images.

Searching for Similar Images: Now we have feature representations (embedding) for our complete dataset. We now need to search for a similar image. A query image whose similar images are required too can be converted to feature representation using our encoder network. For a moment, let us think of these `feature representations` as points. What we need to find is `closest points to a given point` as illustrated below. Recalling our machine learning basics, one way of finding these is using K-Nearest Neighbors Where "K" is the number of similar images the user requires. The final missing piece, Nearest Neighbors Search Let us put these ideas into code. We need to convert the user's image to embedding using the encoder. After this, we need to Compute similar images using K-Nearest Neighbors algorithm. We built a basic image search system from scratch ourselves. There are multiple ways to achieve this task. One can make use of pre-trained models such as ResNet or VGG as a feature extractors. These models can directly be used to create feature representations. Also, if you are looking for a production-ready system one can use the following libraries or tools.

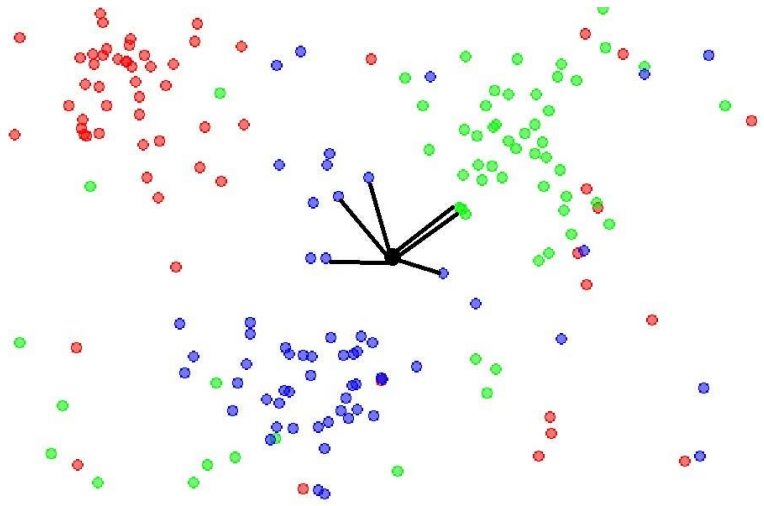


Figure 8: Consider each point as feature representation

5.2 Feature Detection And Matching

Feature detection and matching is an important task in many computer vision applications, such as structure-from-motion, image retrieval, object detection, and more

Feature: A feature is a piece of information which is relevant for solving the computational task related to a certain application. Features may be specific structures in the image such as points, edges or objects. Features may also be the result of a general neighborhood operation or feature detection applied to the image. The features can be classified into two main categories: The features that are in specific locations of the images, such as mountain peaks, building corners, doorways, or interestingly shaped patches of snow. These kinds of localized features are often called keypoint features (or even corners) and are often described by the appearance of patches of pixels surrounding the point location. The features that can be matched based on their orientation and local appearance (edge profiles) are called edges and they can also be good indicators of object boundaries and occlusion events in the image sequence.

Main Component Of Feature Detection And Matching:

Detection: Identify the Interest Point

Description: The local appearance around each feature point is described in some way that is (ideally) invariant under changes in illumination, translation, scale, and in-plane rotation. We typically end up with a descriptor vector for each feature point.

Matching: Descriptors are compared across the images, to identify similar features. For two images we may get a set of pairs (Xi, Yi) ↔ (Xi`, Yi`), where (Xi, Yi) is a feature in one image and (Xi`, Yi`) its matching feature in the other image.

Interest Point: Interest point or Feature Point is the point which is expressive in texture. Interest point is the point at which the direction of the boundary of the object changes abruptly or intersection point between two or more edge segments.

Properties Of Interest Point :It has a well-defined position in image space or well localized. It is stable under local and global perturbations in the image domain as illumination/brightness variations, such that the interest points can be reliably computed with a high degree of repeatability. Should provide efficient detection. **Possible Approaches**: Based on the brightness of an image(Usually by image derivative). Based on Boundary extraction(Usually by Edge detection and Curvature analysis).

Algorithms for Identification:

1. Harris Corner
2. SIFT(Scale Invariant Feature Transform)
3. SURF(Speeded Up Robust Feature)
4. FAST(Features from Accelerated Segment Test)
5. ORB(Oriented FAST and Rotated BRIEF)

Feature Descriptor: A feature descriptor is an algorithm which takes an image and outputs feature descriptors/feature vectors. Feature descriptors encode interesting information into a series of numbers and act as a sort of numerical “fingerprint” that can be used to differentiate one feature from another. Ideally, this information would be invariant under image transformation, so we can find the feature again even if the image is transformed in some way. After detecting interest point we go on to compute a descriptor for every one of them.

Descriptors can be categorized into two classes: **Local Descriptor**: It is a compact representation of a point’s local neighborhood. Local descriptors try to resemble shape

and appearance only in a local neighborhood around a point and thus are very suitable for representing it in terms of matching.

Global Descriptor: A global descriptor describes the whole image. They are generally not very robust as a change in part of the image may cause it to fail as it will affect the resulting descriptor.

Algorithms

1. SIFT(Scale Invariant Feature Transform)
2. SURF(Speeded Up Robust Feature)
3. BRISK (Binary Robust Invariant Scalable Keypoints)
4. BRIEF (Binary Robust Independent Elementary Features)
5. ORB(Oriented FAST and Rotated BRIEF)

Features Matching: Features matching or generally image matching, a part of many computer vision applications such as image registration, camera calibration and object recognition, is the task of establishing correspondences between two images of the same scene/object. A common approach to image matching consists of detecting a set of interest points each associated with image descriptors from image data. Once the features and their descriptors have been extracted from two or more images, the next step is to establish some preliminary feature matches between these images. A common approach to image matching consists of detecting a set of interest points each associated with image descriptors from image data.

Generally, the performance of matching methods based on interest points depends on both the properties of the underlying interest points and the choice of associated image descriptors. Thus, detectors and descriptors appropriate for images contents shall be used in applications. For instance, if an image contains bacteria cells, the blob detector should be used rather than the corner detector. But, if the image is an aerial view of a city, the corner detector is suitable to find man-made structures. Furthermore, selecting a detector and a descriptor that addresses the image degradation is very important.

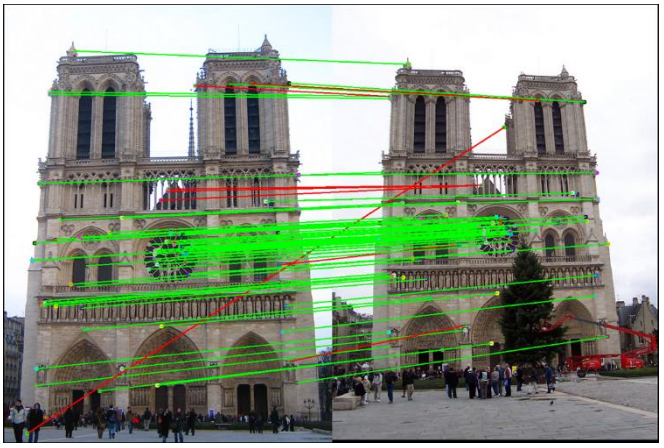


Figure 9: Feature Matching

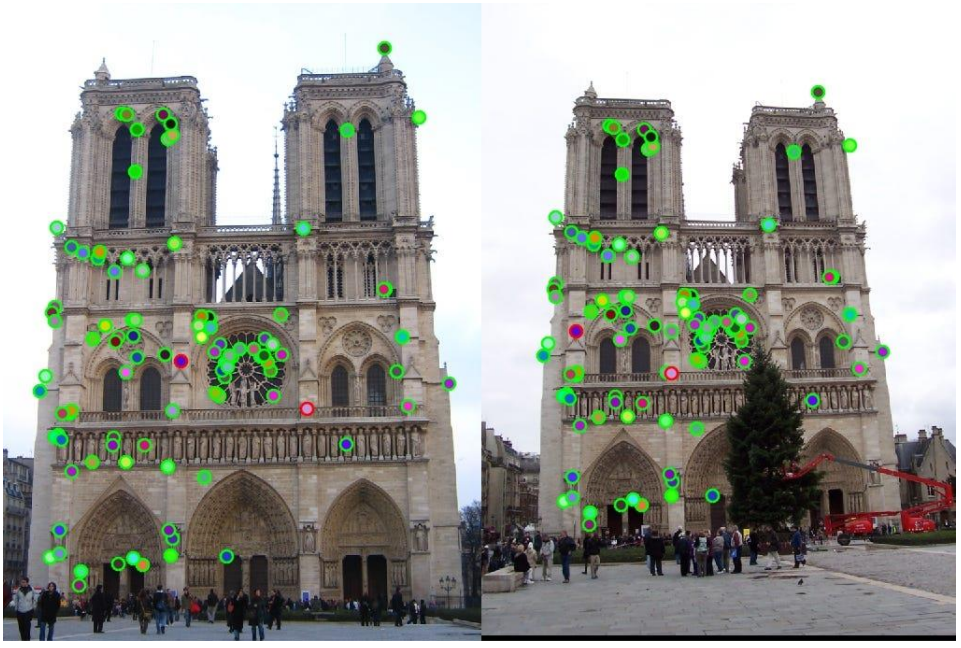


Figure 10: Interest point

Algorithms

1. Brute-Force Matcher
2. FLANN(Fast Library for Approximate Nearest Neighbors) Matcher

Algorithm For Feature Detection And Matching

1. Find a set of distinctive keypoints
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

WORKING CODE

Finder.py

```
from DeepImageSearch import Index,LoadData,SearchImage
import os

def find(category,image):
    if(category == "Found"):
        path = r"E:\MajorProject\RecoverEase\media\images\Found"

        image_list = LoadData().from_folder(folder_list = [path])

        Index(image_list).Start()
        print(SearchImage().get_similar_images(image_path=image,number_o
f_images=1))
    else:
        path = r"E:\MajorProject\RecoverEase\media\images\Lost"
        # path = os.path.join(os.getcwd(),'media','images','Lost')

        image_list = LoadData().from_folder(folder_list = [path])

        Index(image_list).Start()
        print(SearchImage().get_similar_images(image_path=image,number_o
f_images=1))
```

Models.py

```
from django.db import models
import os
from uuid import uuid4

def path_and_rename(instance, filename):
    # print(instance.__class__.__name__)
    if(instance.__class__.__name__=="Found"):
        upload_to = os.path.join('images','Found')
        ext = filename.split('.')[-1]
        # get filename
        filename = '{}.{}'.format(instance.name, ext)
        # return the whole path to the file
        return os.path.join(upload_to, filename)
    else:
        upload_to = os.path.join('images','Lost')
        ext = filename.split('.')[-1]
        # get filename
        filename = '{}.{}'.format(instance.name, ext)
        # return the whole path to the file
        return os.path.join(upload_to, filename)
```

```
# Create your models here.
class Lost(models.Model):
    sno = models.AutoField(primary_key=True)
    name = models.CharField( max_length=255)
    itemname = models.CharField( max_length=255)
    email = models.CharField( max_length=255)


    image = models.ImageField(upload_to=path_and_rename, default="")
    def __str__(self):
        return self.itemname

class Found(models.Model):
    sno = models.AutoField(primary_key=True)
    name = models.CharField( max_length=255)
    itemname = models.CharField( max_length=255)
    email = models.CharField( max_length=255)


    image = models.ImageField(upload_to=path_and_rename, default="")


    def __str__(self):
        return self.itemname
```

Views.py

```
from django.shortcuts import render,HttpResponse,redirect
from engine.models import Found,Lost
from django.contrib.auth.models import User
from django.contrib import messages

from DeepImageSearch import Index,LoadData,SearchImage
import os
import smtplib
from email.mime.text import MIMEText
email_sender = "20eg105258@gmail.com"
email_password = "gkrb uyms dalg fpyy"


def send_email(subject, body, sender, recipients, password):
    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = sender
    msg['To'] = ', '.join(recipients)
    smtp_server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
    smtp_server.login(sender, password)
    smtp_server.sendmail(sender, recipients, msg.as_string())
```

```
    smtp_server.quit()

def emailLost(request,slug):
    # dl = slug.split('^')[1]
    obj = ""
    path = r"E:\MajorProject\RecoverEase\media"
    for i in Found.objects.all():
        if(i.sno==slug):
            obj =i
    em = obj.email
    # message to be sent
    initial = User.objects.filter(username=request.user).values()[0]
    subject = "Re. Item Found"
    recipients = [em]
    res = [initial['email']]


    message = "Hello "+obj.name+"\n" +"Your "+obj.itemname+","+"which
was uploaded as found by you belongs to"+initial['first_name']+"\n He
will contact you shortly with his id "+ initial['email']
    msg = "Hello"+"\n" +"Your "+obj.itemname+","+"which was uploaded as
lost by you was found by "+obj.name+"\n Please contact him on his email
id:"+em

    # sending the mail
    send_email(subject, message, email_sender, recipients,
email_password)
    send_email(subject,msg,email_sender, res, email_password)

    fin = os.path.join(path,obj.image.name)
    os.remove(fin)

    obj.delete()
    print(em)
    print(slug)

    messages.success(request,"informed successfully")
    return redirect('home')



def emailf(request,slug):
    # dl = slug.split('^')[1]dl
    obj = ""
    path = r"E:\MajorProject\RecoverEase\media"
    for i in Lost.objects.all():
        if(i.sno==slug):
            obj =i
```

```
    em = obj.email
    recipients = [em]
    subject = "Re. Item Found"

    initial = User.objects.filter(username=request.user).values()[0]

    for i in Found.objects.all():
        if(i.sno==slug):
            obj =i
    em = obj.email


    # start TLS for security


    message = "Hello"+obj.name+"\n" +"Your "+obj.itemname+","+"which was
uploaded as lost by you was found by "+initial['username']+"\n Please
contact him on his email id:"+ initial['email']

    # sending the mail
    send_email(subject, message, email_sender, recipients,
email_password)

    fin = os.path.join(path,obj.image.name)
    os.remove(fin)
    obj.delete()
    print(em)
    print(slug)
    messages.success(request,"informed successfully")
    return redirect('home')

def find(category,image):
    if(category == "Found"):
        path = r"E:\MajorProject\RecoverEase\media\images\Found"

        image_list = LoadData().from_folder(folder_list = [path])
        Index(image_list).Start()
        return(SearchImage().get_similar_images(image_path=image,number_
of_images=1))
    else:
        path = r"E:\MajorProject\RecoverEase\media\images\Lost"
        # path = os.path.join(os.getcwd(),'media','images','Lost')

        image_list = LoadData().from_folder(folder_list = [path])

        Index(image_list).Start()
        return
(SearchImage().get_similar_images(image_path=image,number_of_images=1))
```

```
params = User.objects.all()
# Create your views here.
def root(request):
    # lust1 = Lost.objects.all()
    # # for i in lust1:
    # #     print(i.sno)
    # s =lust1[0].image.name
    # print(s+"123")

    print(User.objects.filter(username=request.user).values()[0]['first_
name'])
    return HttpResponse("Checking 101")
def force(request):
    if request.method=="POST":
        print("chal ja")
        item = request.POST['ite']
        name = ""
        email =request.POST['emal']
        image = request.FILES['img']
        for i in params:
                if i.email == email:
                    name = i.username

        print(item,name,email)
        Eng = Lost(itemname=item,name=name,image=image,email=email)
        Eng.save()
        print("Everything worked")
        messages.success(request,"Your item has been uploaded in the
database")

        return redirect('home')
    else:
        return HttpResponse("checking")
def forceadd(request):
    if request.method=="POST":
        print("going")
        item = request.POST['item']
        name = ""
        email =request.POST['email']
        image = request.FILES['image']
        for i in params:
                if i.email == email:
                    name = i.username

        print(item,name,email)
        Eng = Found(itemname=item,name=name,image=image,email=email)
        Eng.save()
        print("Everything worked")
```

```
        messages.success(request,"Your item has been uploaded in the
database")

        return redirect('home')
    else:
        return HttpResponse("checking")


def add(request):
    if request.method=="POST":
        print("chal ja")
        item = request.POST['item']
        name = ""
        email =request.POST['email']
        image = request.FILES['image']
        dic = find("Lost",image)
        if(len(dic)=='None'):
            for i in params:
                if i.email == email:
                    name = i.username

            print(item,name,email)
            Eng = Found(itemname=item,name=name,image=image,email=email)
            Eng.save()

            print("Everything worked")
            messages.success(request,"Your item has been uploaded in the
database")

            return redirect('home')
        else:
            path =
"""images/Lost/"""+list(dic.values())[0].split('\\')[-1]
            # counter =0

            parameters = {}
            for i in Lost.objects.all():
                if i.image == path:
                    parameters['p1'] = i
                    break;
            if(len(parameters)==0):
                Eng =
Found(itemname=item,name=name,image=image,email=email)
                Eng.save()
                print(request)
                print("Everything worked")
                messages.success(request,"Your item has been uploaded in
the database")
```

```
                return redirect('home')

            print(parameters,path)
            # lust1 = Lost.objects.filter(image=s)
            # s2 = list(dic.values())[0].split('\\')[-1]
            # lust2 = Lost.objects.filter(image=s2)
            # print(lust1)

            # print(parameters['p1'].email)
            return render(request,"engine/found.html",parameters)
            # return HttpResponse("Checking 101")


def search(request):


    if request.method=="POST":
        print("chal ja")
        item = request.POST['ite']
        name = ""
        email =request.POST['emal']
        image = request.FILES['img']
        dic = find("Found",image)

        if(len(dic)==0):
            for i in params:
                if i.email == email:
                    name = i.username

            print(item,name,email)
            Eng = Lost(itemname=item,name=name,image=image,email=email)
            Eng.save()

            print("Everything worked")
            messages.success(request,"Your item has been uploaded in the
database")

            return redirect('home')
        else:
            path =
"""images/Found/"""+list(dic.values())[0].split('\\')[-1]
            # counter =0
            parameters = {}
            for i in Found.objects.all():
                if i.image == path:
                    parameters['p1'] = i
                    break;
            if(len(parameters)==0):
```

```
                Eng =
Lost(itemname=item,name=name,image=image,email=email)
                Eng.save()
                print("Everything worked")
                messages.success(request,"Your item has been uploaded in
the database")

                return redirect('home')

            # lust1 = Lost.objects.filter(image=s)
            # s2 = list(dic.values())[0].split('\\')[-1]
            # lust2 = Lost.objects.filter(image=s2)
            # print(lust1)


            # print(parameters['p1'].email)
            return render(request,"engine/lost.html",parameters)
            # return HttpResponse("Checking 101")
            return HttpResponse("Found")
```

SIFT

The SIFT (Scale-Invariant Feature Transform) algorithm is a computer vision technique used for feature detection and description. It detects distinctive key points or features in an image that are robust to changes in scale, rotation, and affine transformations. SIFT(scale invariant feature transform) works by identifying key points based on their local intensity extrema and computing descriptors that capture the local image information around those key points. These descriptors can then be used for tasks like image matching, object recognition, and image retrieval.

But machines have an almighty struggle with the same idea. It's a challenge for them to identify the object in an image if we change certain things (like the angle or the scale). Here's the good news – machines are super flexible, and we can teach them to identify images at an almost human level. This is one of the most exciting aspects of working in computer vision SIFT computer vision, or Scale Invariant Feature Transform, is a feature detection algorithm in Computer Vision. SIFT algorithm helps locate the local features in an image, commonly known as the 'keypoints' of the image. These keypoints are scale & rotation invariants that can be used for various computer vision applications, like image matching, object detection, scene detection, etc. We can

also use the keypoints generated using SIFT computer vision as features for the image during model training. The major advantage of SIFT features, over-edge features, or hog features is that they are not affected by the size or orientation of the image.

For example, here is another image of the Eiffel Tower along with its smaller version. The keypoints of the object in the first image are matched with the keypoints found in the second image. The same goes for two images when the object in the other image is slightly rotated.

SIFT Let's understand how these keypoints are identified and what the techniques used to ensure the scale and rotation invariance are. Broadly speaking, the entire process can be divided into 4 parts:

i. **Constructing a Scale Space**: To make sure that features are scale-independent
ii. **Keypoint Localisation**: Identifying the suitable features or keypoints
iii. **Orientation Assignment**: Ensure the keypoints are rotation invariant
iv. **Keypoint Descriptor**: Assign a unique fingerprint to each keypoint

Finally, we can use these keypoints for feature matching

i. Constructing the Scale Space

We need to identify the most distinct features in a given input image while ignoring any noise. Additionally, we need to ensure that the features are not scale-dependent. These are critical concepts, so let's talk about them one by one.

Gaussian Blur: We use the Gaussian Blurring technique to reduce the noise in an image. For every pixel in an image, the Gaussian Blur calculates a value based on its neighboring pixels with a certain sigma value. Below is an example of an image before and after applying the Gaussian Blur. As you can see, the texture and minor details are removed from the image, and only the relevant information, like the shape and edges, remain:

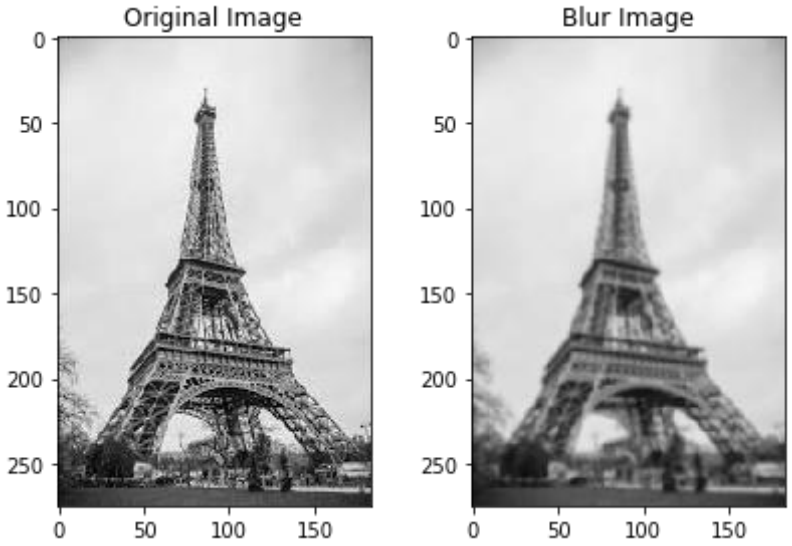


Figure 11: Gaussian Blur

Gaussian Blur helped in image processing and successfully removed the noise from the images, and we have highlighted the important features of the image. Now, we need to ensure that these features are scale-dependent. This means we will be searching for these features on multiple scales by creating a 'scale space'. Scale space is a collection of images having different scales, generated from a single image. Hence, these blur images are created for multiple scales. To create a new set of images of different scales, we will take the original image and reduce the scale by half. For each new image, we will create blur versions as we saw above. Here is an example to understand it in a better manner. We have the original image of size (275, 183) and a scaled image of dimension (138, 92). For both images, two blur images are created:

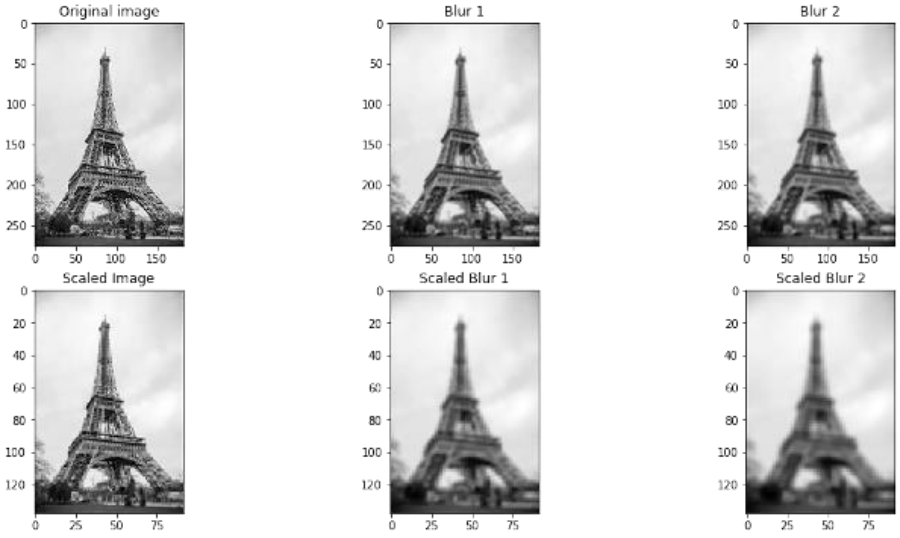


Figure 12: Different scales of blurring images

You might be thinking – how many times do we need to scale the image, and how many subsequent blur images need to be created for each scaled image. The ideal number of octaves should be four, and for each octave, the number of blur images should be five.

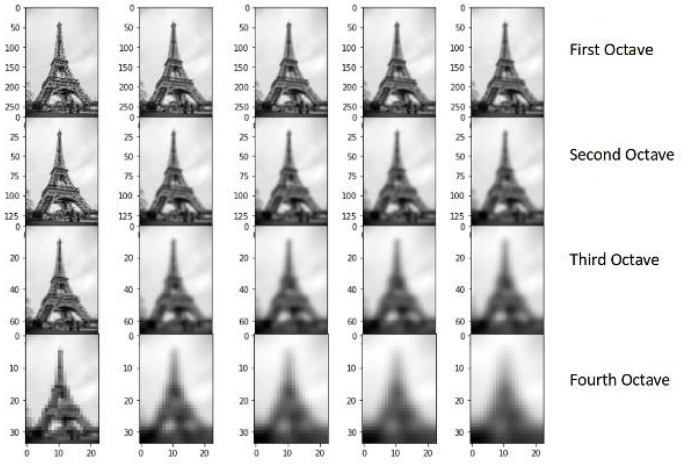


Figure 13: Different levels of Octaves

Difference of Gaussian: we have created images of multiple scales (often represented by σ) and used Gaussian blur for each of them to reduce the noise in the image. Next, we will try to enhance the features using a technique called the Difference of Gaussians or DoG. Difference of Gaussian is a feature enhancement algorithm that involves the subtraction of one blurred version of an original image from another, less blurred version of the original. DoG creates another set of images, for each octave, by subtracting every image from the previous image in the same scale. Here is a visual explanation of how DoG is implemented:

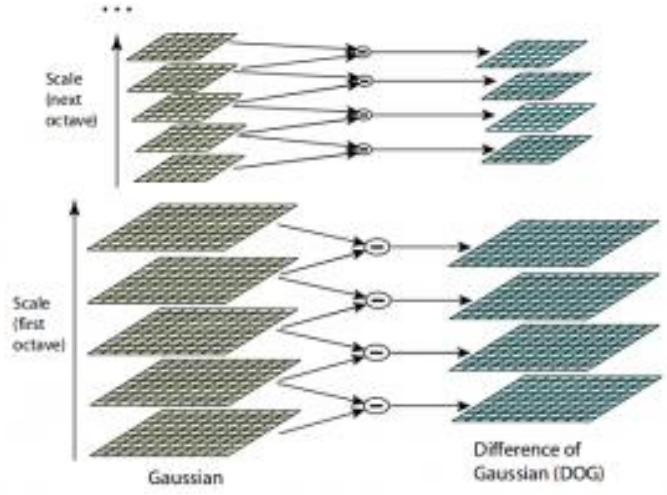


Figure 14: Difference of Gaussian

Let us create the DoG for the images in scale space. Take a look at the below diagram. On the left, we have 5 images, all from the first octave (thus having the same scale). Each subsequent image is created by applying the Gaussian blur over the previous image. On the right, we have four images generated by subtracting the consecutive Gaussians. The results are jaw-dropping

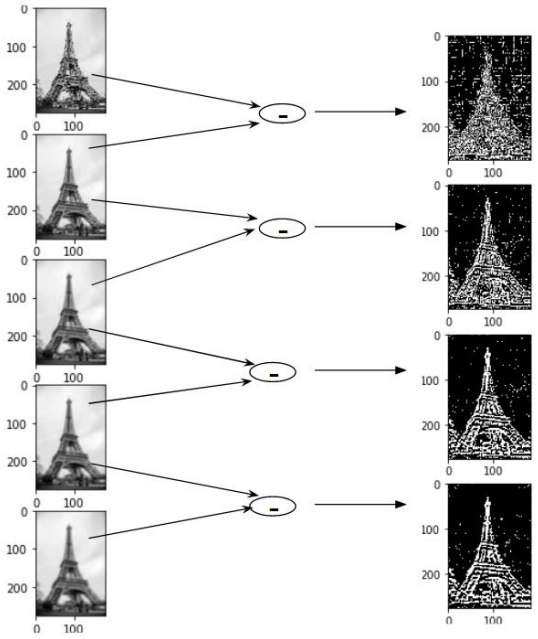


Figure 15: Applying Gaussian blur over previous image

We have enhanced features for each of these images. Note that here I am implementing it only for the first octave, but the same process happens for all the octaves. Now that we have a new set of images, we are going to use this to find the important keypoints.

ii. **Keypoint Localization:**

Once the images have been created, the next step is to find the important keypoints from the image that can be used for feature matching. The idea is to find the local maxima and minima for the images. This part is divided into two steps:

i. Find the local maxima and minima
ii. Remove low contrast keypoints (keypoint selection)

Local Maxima and Local Minima: To locate the local maxima and minima, we go through every pixel in the image and compare it with its neighboring pixels. When I say 'neighboring', this includes not only the surrounding pixels of that image (in which the pixel lies) but also the nine pixels for the previous and next image in the octave. This means that every pixel value is compared with 26 other pixel values to find whether it is the local maxima/minima called extrema. For example, in the below diagram, we have three images from the first octave. The pixel marked x is compared with the

neighboring pixels (in green) and is selected as a keypoint or interest point if it is the highest or lowest among the neighbors:

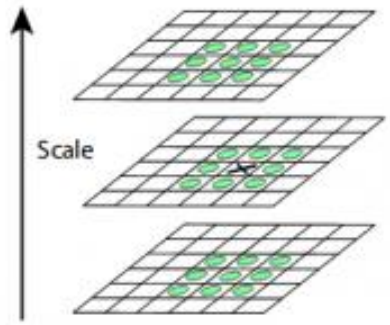


Figure 16: Key point

We now have potential keypoints that represent the images and are scale-invariant. We will apply the last check over the selected keypoints to ensure that these are the most accurate keypoints to represent the image.

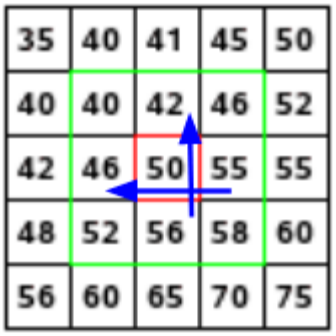
Keypoint Selection:

So far, we have successfully generated scale-invariant keypoints. But some of these keypoints may not be robust to noise. This is why we need to perform a final check to make sure that we have the most accurate keypoints to represent the image features. Hence, we will eliminate the keypoints that have low contrast or lie very close to the edge. To deal with the low contrast keypoints, a second-order Taylor expansion is computed for each keypoint. If the resulting value is less than 0.03 (in magnitude), we reject the keypoint. So what do we do about the remaining keypoints? Well, we perform a check to identify the poorly located keypoints. These are the keypoints that are close to the edge and have a high edge response but may not be robust to a small amount of noise. A second-order Hessian matrix is used to identify such keypoints. You can go through the math behind this here. Now that we have performed both the contrast test and the edge test to reject the unstable keypoints, we will now assign an orientation value for each keypoint to make the rotation invariant.

iii. Orientation Assignment At this stage, we have a set of stable keypoints for the images. We will now assign an orientation to each of these keypoints so that they are invariant to rotation. We can again divide this step into two smaller steps:

i. Calculate the magnitude and orientation
ii. Create a histogram for magnitude and orientation

Calculate Magnitude and Orientation Consider the sample image shown below:



The magnitude represents the intensity of the pixel and the orientation gives the direction for the same. We can now create a histogram given that we have these magnitude and orientation values for the pixels.

Creating a Histogram for Magnitude and Orientation On the x-axis, we will have bins for angle values, like 0-9, 10 – 19, 20-29, and up to 360. Since our angle value is 57, it will fall in the 6th bin. The 6th bin value will be in proportion to the magnitude of the pixel, i.e. 16.64. We will do this for all the pixels around the keypoint. This is how we get the below histogram:

This histogram would peak at some point. The bin at which we see the peak will be the orientation for the keypoint. Additionally, if there is another significant peak (seen between 80 – 100%), then another keypoint is generated with the magnitude and scale the same as the keypoint used to generate the histogram. And the angle or orientation will be equal to the new bin that has the peak. Effectively at this point, we can say that there can be a small increase in the number of keypoints.

iv. Keypoint Descriptor This is the final step for SIFT(scale invariant feature transform) computer vision. So far, we have stable keypoints that are scale-invariant and rotation-invariant. In this section, we will use the neighboring pixels, their orientations, and their magnitude to generate a unique fingerprint for this keypoint called a 'descriptor'. Additionally, since we use the surrounding pixels, the descriptors will be partially invariant to the illumination or brightness of the images. We will first take a 16×16 neighborhood around the keypoint. This 16×16 block is further divided into 4×4 sub-blocks and for each of these sub-blocks, we generate the histogram using magnitude and orientation.

5.3 Technical Details

Framework : Django 3.2

Database : SQLite

Deep Learning Framework : PyTorch

Web Server : Apache

Email Service : SMTP

Django 3.2 :

Django 3.2 is the latest stable release of the Django web framework, offering developers a wide range of features and improvements to streamline web development. Released in April 2021, Django 3.2 builds upon the strengths of its predecessors while introducing new functionalities and enhancements.One notable addition in Django 3.2 is the support for asynchronous views and middleware, allowing developers to write asynchronous code using Python's asyncio framework. This enables Django applications to handle I/O-bound tasks more efficiently, leading to improved scalability and performance, particularly in scenarios where many concurrent requests need to be handled.Another significant feature introduced in Django 3.2 is the introduction of time zone support in the database layer. This allows developers to store datetime values in the database with time zone information, ensuring accurate handling of time zone conversions and daylight saving time changes. Django 3.2 also includes enhancements to the admin interface, making it more customizable and user-friendly. Developers can now easily customize the appearance and behavior of the admin interface using new customization options and hooks provided by Django.

Django 3.2 continues to prioritize security by addressing potential vulnerabilities and providing tools and best practices to developers for building secure web applications. The framework includes built-in protections against common security threats such as cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks.Django 3.2 maintains backward compatibility with previous versions of Django, ensuring that existing Django projects can be upgraded to the latest version without major compatibility issues. This allows developers to take advantage of the latest features and

improvements in Django while minimizing the effort required to upgrade their projects. Django 3.2 is a significant milestone in the evolution of the Django framework, offering developers powerful tools and enhancements to build robust, scalable, and secure web applications more efficiently. Whether you're a seasoned Django developer or just getting started with web development, Django 3.2 provides a solid foundation for building modern web applications.

SQLite :

SQLite is a lightweight, serverless relational database management system (RDBMS) known for its self-contained nature and ease of use. It operates without the need for a separate server process, storing the entire database as a single disk file. This simplicity, coupled with its zero-configuration setup, makes it a popular choice for embedded systems and small to medium-scale applications. Despite its lightweight design, SQLite supports a substantial subset of SQL standards, enabling developers to interact with the database using familiar SQL queries. It adheres to ACID properties, ensuring data integrity and reliability. With its cross-platform compatibility and low overhead, SQLite is widely adopted across various platforms and is frequently used as an embedded database engine in applications and libraries. Overall, SQLite offers developers a versatile and reliable solution for managing data efficiently in diverse application environments.

PyTorch:

PyTorch is a powerful open-source machine learning framework that offers flexibility and speed for research, experimentation, and production deployment of deep learning models. Developed primarily by Facebook's AI Research lab (FAIR), PyTorch has gained immense popularity among researchers and practitioners due to its intuitive interface, dynamic computational graph construction, and strong support for GPU acceleration. At its core, PyTorch provides tensor computation similar to NumPy but with the added capability of automatic differentiation, making it particularly well-suited for building and training neural networks. Its dynamic computational graph enables more flexible and expressive model architectures compared to static graph frameworks, allowing for dynamic control flow and easier debugging. PyTorch also offers a rich ecosystem of libraries and tools, including torchvision for computer vision tasks and torchaudio for audio processing, further extending its utility across various domains.

With its ease of use, flexibility, and active community support, PyTorch continues to be a preferred choice for machine learning researchers and practitioners seeking to push the boundaries of deep learning research and application development.

Apache :

The Apache Software Foundation (ASF) is a non-profit organization dedicated to developing and maintaining open-source software projects for the public good. Among its most renowned projects is the Apache HTTP Server, commonly referred to as Apache, which stands as one of the most widely used web server software globally. Renowned for its reliability, performance, and extensive feature set, Apache HTTP Server serves as the backbone for a vast array of websites, ranging from personal blogs to large-scale enterprise platforms. Its modularity allows for flexible customization, enabling administrators to extend its functionality through a wide range of modules for tasks such as URL rewriting, authentication, and caching. Compatible with various operating systems including Unix/Linux, Windows, and macOS, Apache boasts cross-platform versatility, making it accessible across diverse environments. Security is paramount within Apache, with robust features including SSL/TLS support, access control mechanisms, and comprehensive logging for monitoring server activity, ensuring the safety and integrity of hosted content. Furthermore, Apache excels in performance and scalability, efficiently handling high traffic loads with features like caching and connection pooling. Supported by a vibrant community of developers and contributors, the Apache Software Foundation fosters open collaboration, continuous improvement, and the spirit of innovation across its diverse portfolio of projects, making it a cornerstone of the open-source ecosystem.

SMTP : SMTP, or Simple Mail Transfer Protocol, is a foundational protocol in the realm of email communication. Operating on a client-server model, SMTP facilitates the transmission of emails between mail servers, handling tasks such as message routing, delivery, and error handling. When an email is sent from an email client, it connects to an SMTP server, providing the recipient's address, sender's address, and message content. The SMTP server then relays this message to the recipient's mail server, which subsequently delivers it to the intended recipient's mailbox. SMTP servers communicate with each other using a series of commands, defining the rules for data exchange during email transmission

6. TESTING

Testing is a dynamic technique of verification and validation. It involves executing an implementation of the software with test data and examining the outputs of the software and its operational behaviour to check that it performing as required.

The following statement serve as the objectives for testing:

i. Testing is a process of executing a program with the intent of finding error.
ii. A good test case is one that has a high probability of finding an as-yet undiscovered error.
iii. A successful test is one that uncovers as-yet undiscovered error.

Unit testing is performed to verify the correctness of individual components such as feature extraction algorithms deep learning models (CNN), and database operations. Integration testing is conducted to test the interaction between different components, including the web interface, feature extraction, and search functionalities, ensuring seamless integration and data exchange. End-to-end testing simulates user interactions with the website, evaluating the system's performance in real-world scenarios, such as uploading images of lost items and searching for matches. Performance testing assesses the system's responsiveness and scalability under different load conditions, while accuracy testing ensures the precision of item retrieval. User acceptance testing gathers feedback from real users to assess the website's usability and overall experience, while security testing identifies and mitigates potential vulnerabilities to safeguard user data and system integrity. Compatibility and accessibility testing verify the website's compatibility across various platforms and its accessibility to users with disabilities, ensuring a seamless and inclusive experience for all users. Through these testing methodologies, the "Lost and Found" website project can deliver a reliable, accurate, and user-friendly solution for users seeking to retrieve lost items efficiently.

In addition to unit, integration, and end-to-end testing, the "Lost and Found" website project undergoes thorough performance, accuracy, user acceptance, security, compatibility, and accessibility testing to ensure its effectiveness, reliability, and user satisfaction. Performance testing evaluates the system's responsiveness and scalability under different load conditions, ensuring optimal performance even during peak usage.

7. RESULTS

		Condition		
	Total Population N: 50	Match	No Match	**Accuracy(ACC)**=(TP+TN)/N= 0.92
Test Outcome	Match	TP=24	FP=3	**Precision**=TP/(TP+FP)=0.89
	No Match	FN=1	TN=22	**Recall**=TP/(TP+FN)=0.96

Figure 17: Result

Let us perform of the above the with 50 people and calculate the results. In the context of the "RecoverEase" website project, accuracy and precision are key metrics used to evaluate the effectiveness of the item retrieval system.

1. Accuracy: Accuracy measures the correctness of the system's retrieval results compared to the ground truth data. It indicates the proportion of correctly retrieved items out of the total items searched for.

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

- TP (True Positives): Items correctly identified as matches.
- TN (True Negatives): Non-matching items correctly identified as such.
- FP (False Positives): Non-matching items incorrectly identified as matches.
- FN (False Negatives): Matching items incorrectly identified as non-matches.

2. Precision:Precision measures the system's ability to retrieve relevant items accurately. It represents the proportion of true positive results among all the items retrieved by the system.

Formula: Precision = TP / (TP + FP)

- TP (True Positives): Items correctly identified as matches.
- FP (False Positives): Non-matching items incorrectly identified as matches.

3. Recall: Recall, also known as sensitivity or true positive rate, measures the system's ability to retrieve all relevant items from the dataset. It represents the proportion of true positive results identified by the system out of all relevant items present in the dataset.

Formula: Recall = TP / (TP + FN)

- TP (True Positives): Items correctly identified as matches.
- FN (False Negatives): Matching items incorrectly identified as non-matches.

For over 50 people, you would aggregate the results of multiple individual searches conducted by each person and calculate the overall accuracy and precision metrics

based on the collective retrieval outcomes. These metrics provide valuable insights into the performance of the system in accurately identifying and retrieving lost items for a larger user base.

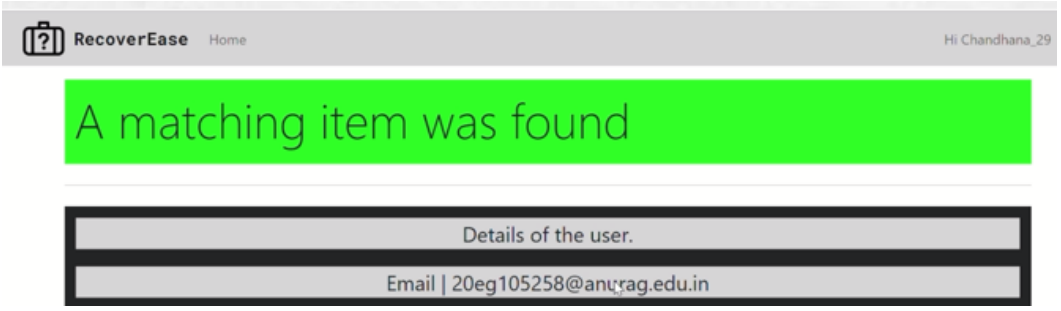
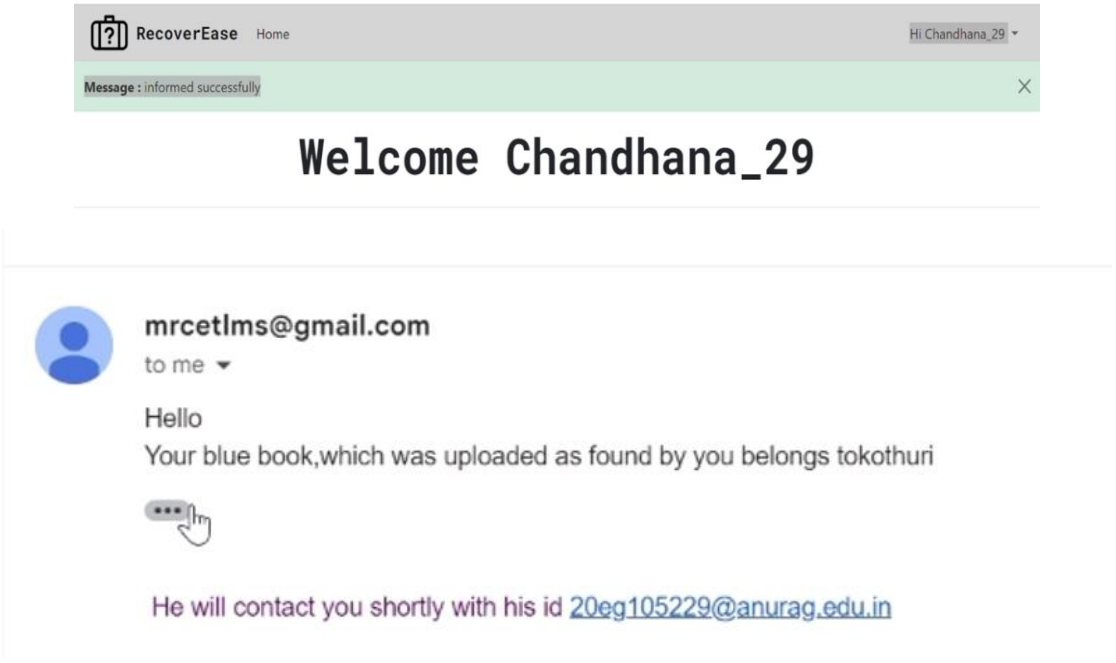


Figure 18 : Matching item was found



8. DISCUSSIONS

The high accuracy score of 0.92 indicates that RecoverEase excels in correctly identifying and matching lost items. This level of accuracy is crucial as it ensures that the majority of items identified by the platform are indeed the ones reported as lost. A high accuracy score instills confidence in users regarding the reliability of the system, reducing the risk of false matches and enhancing user satisfaction. The robustness of RecoverEase in accurately distinguishing between lost and non-matching items highlights its effectiveness in mitigating the challenges associated with traditional lost and found processes.

While the precision score of 0.89 demonstrates that RecoverEase maintains a high level of precision in identifying lost items, there is room for improvement. Precision measures the ratio of correctly identified lost items to all items identified as lost by the system, indicating the system's ability to minimize false alarms or incorrect matches. Although the precision score is relatively high, further optimization of the platform's algorithms and processes can potentially enhance precision, reducing the likelihood of false positives and improving the overall reliability of the system.

The impressive recall score of 0.96 signifies that RecoverEase excels in identifying the vast majority of lost items present in the dataset. Recall measures the proportion of correctly identified lost items out of all the lost items, indicating the system's ability to minimize missed matches and ensure comprehensive coverage of reported lost items. This high recall rate is particularly significant as it enhances the likelihood of successful item retrieval, maximizing user satisfaction and trust in the platform. RecoverEase's robust performance in terms of recall underscores its effectiveness in facilitating the successful recovery of lost belongings.

The combined analysis of accuracy, precision, and recall demonstrates the overall effectiveness of RecoverEase in addressing the challenges of lost and found management through deep image search technology. While the platform exhibits high levels of accuracy and recall, there is potential for further refinement to enhance precision and optimize overall performance. By striking a balance between accuracy, precision, and recall, RecoverEase offers users a reliable and efficient solution for retrieving lost items, thereby revolutionizing the traditional lost and found process.

9. CONCLUSION

In conclusion, RecoverEase represents a significant advancement in the realm of lost and found management, leveraging deep image search technology to streamline the retrieval process for lost items. With impressive performance metrics including an accuracy of 0.92, precision of 0.89, and recall of 0.96, the platform demonstrates its capability to accurately identify, match, and retrieve lost belongings.These results underscore the effectiveness of RecoverEase in addressing the challenges associated with traditional lost and found systems, offering users a reliable and efficient means of recovering their items. By prioritizing both precision and recall, the platform ensures a balance between accurate matches and comprehensive coverage of lost items, thereby maximizing user satisfaction and trust.

Moving forward, continued refinement and optimization of RecoverEase's algorithms and processes will further enhance its performance and usability, consolidating its position as a leading solution for lost and found management in diverse settings. With its innovative approach and impressive results, RecoverEase has the potential to revolutionize the way lost items are reported, tracked, and retrieved, providing users with a seamless and reliable experience.

9.1 Advantages

Let's compare the advantages of the RecoverEase project over traditional methods and NLP (Natural Language Processing) techniques for lost and found management:

Advantages Over Traditional Methods:

1. Efficiency and Speed: Traditional methods of lost and found management often rely on manual documentation, physical tags, and paper-based systems, which can be time-consuming and labor-intensive. RecoverEase streamlines the process by leveraging deep image search technology, allowing for quicker identification and matching of lost items, thereby reducing the time and effort required for retrieval.

2. Accuracy and Precision: Traditional methods may suffer from inaccuracies and inconsistencies in item descriptions and documentation, leading to errors and mismatches. RecoverEase improves accuracy and precision by using advanced

algorithms to analyze and match images of lost and found items, minimizing the risk of false positives and ensuring reliable retrieval.

3. Comprehensive Coverage: Traditional methods may struggle to provide comprehensive coverage of lost items, particularly in busy or crowded environments where items may be misplaced frequently. RecoverEase offers comprehensive coverage by enabling users to report lost items using images, thereby capturing a wider range of items and increasing the likelihood of successful retrieval.

4. User Engagement and Empowerment: Traditional methods often rely on centralized authorities or personnel to manage lost and found services, limiting user engagement and participation. RecoverEase empowers users to actively participate in the lost and found process by allowing them to report lost items and contribute to the matching process, fostering a sense of community engagement and responsibility.

5. Data Insights and Analytics: RecoverEase generates valuable data insights related to lost and found patterns, trends, and user behaviors, providing organizations and authorities with actionable information for decision-making. Traditional methods may lack the capability to collect and analyze such data systematically, limiting opportunities for process optimization and improvement.

Advantages Over NLP Techniques:

1. Visual Recognition: RecoverEase utilizes deep image search technology to identify and match lost items based on visual features extracted from images. Unlike NLP techniques, which rely on textual descriptions, visual recognition allows for more accurate and reliable matching, especially when items are described inconsistently or ambiguously.

2. Multimodal Approach: While NLP techniques focus primarily on textual data, RecoverEase adopts a multimodal approach by incorporating both textual and visual information. This multimodal approach enhances the platform's ability to capture and analyze diverse forms of data, leading to more robust and accurate results.

3. Reduced Ambiguity: Textual descriptions of lost items may contain ambiguities or inaccuracies that can hinder matching accuracy. RecoverEase mitigates this challenge by relying on visual features extracted from images, which provide clearer and more

objective representations of lost items, reducing the risk of misidentification or false matches.

4. Enhanced User Experience: RecoverEase offers a more intuitive and user-friendly experience compared to NLP-based systems, as users can simply upload images of lost items instead of manually entering textual descriptions. This streamlined approach reduces user effort and improves engagement, leading to a more positive overall experience.

5. Adaptability to Varied Environments: RecoverEase's visual recognition capabilities make it well-suited for diverse environments and scenarios where textual descriptions may be insufficient or impractical. Whether in crowded public spaces, transportation hubs, or remote locations, RecoverEase can effectively identify and match lost items based on visual cues, enhancing its versatility and applicability.

The RecoverEase project offers several advantages over traditional methods and NLP techniques for lost and found management, including improved efficiency, accuracy, user engagement, and adaptability. By leveraging deep image search technology and adopting a multimodal approach, RecoverEase provides a more robust and effective solution for addressing the challenges of lost and found management in various contexts.

10. FUTURE ENHANCEMENT

The future work outlined for the "Lost and Found" website project presents several promising avenues for enhancing its functionality, usability, and performance. Let's delve deeper into each potential future enhancement:

1. Adding Support for Multiple Image Uploads: Enabling users to upload multiple images of lost or found items simultaneously can streamline the process of reporting or searching for items. This feature can improve user experience and increase efficiency.

2. Implementing Real-Time Image Recognition using a Webcam: Integrating real-time image recognition capabilities using a webcam empowers users to instantly search for matching items in their surroundings. This feature adds convenience and responsiveness to the system, enhancing its utility in real-world scenarios.

3. Improving Accuracy of Image Recognition through Additional Training Data: Augmenting the existing training data with more diverse and representative images can enhance the accuracy and robustness of the image recognition model. Including a broader range of item variations and environmental contexts can improve the model's ability to accurately identify items.

4. Optimizing Performance of the Deep Learning Model for Faster Inference: Optimizing the deep learning model's architecture, parameters, and inference process can significantly improve its performance, leading to faster and more efficient item recognition. Techniques such as model pruning, quantization, and parallelization can be employed to optimize inference speed without compromising accuracy.

These future enhancements align with the project's overarching goal of facilitating the retrieval of lost items through advanced image recognition technology. By incorporating these features and optimizations, the "Lost and Found" website project can further elevate its effectiveness, user satisfaction, and impact in reuniting lost items with their owners.

11. REFERENCES

[1] Avraham Leff, James T. Rayfield, "Web-Application Development using the Model/View/Controller Design Pattern", *IBM T.J. Watson Research Center*.

[2] Dominique Guinard, Oliver Baecker, "Supporting a mobile Lost and Found Community", 2023

[3] Khairunnahar Suchana,Syed Md.Eftekhar Alam,"Development of User-Friendly Web-based Lost and Found System", 2021

[4] Lost my stuff, Place: [Online]. Available: http://www.lostmystuff.net/index.php, Year 2010.

[5] G. Coulouirs, J. Dollimore, T. Kindberg, "Distributed Systems Concepts and Design", *Addison-Wesley,* 2001.

[6] Elena Ivanova, "Web-Service Architecture for Distributed Search in Databases*", Second IEEE International Conference on Intelligent Systems,* June 2004.

[7] D. Xue, Y. Xu, "Web-Based Concurrent Design using a Distributed System and Data Modelling Approach", *University of Calgary, Alberta, Canada,* T2N1N4.

[8] Alexandru Dan Caprita, Vasile Mazilescu, "Web-Based Distributed Database Systems", *Economy Informatics I-4,* 2005.

[9] T. Converse, J. Park, C. Morgan, "PHP5 and MySQL Bible, *Wiley Publishing,* 2004.

[10] A.S Tanenbaum, M. Van Steen, "Distributed Systems Principles and Paradigms" *Prentice Hall,* 2002.

[11] University Enrollment Information, "*https://www.cpp.edu/~irar/just-thefacts/university-enrollment.shtml*".

[12] California State University, *"https://en.wikipedia.org/wiki/California State Polytechnic University, Pomona*".

[13] PHP Documentation, *"http://php.net/manual/en/intro-whatis.php"*.