Image matching
Lost and Found system

Author
Tam Yat Hin

Supervisor
Dr. Luo Ping

April 18, 2023
Abstract

Image recognition tasks on lost and found services are a tedious and costly process that requires a significant amount of manpower. However, with the advances in machine learning, specifically in image recognition, this task can be simplified through the development of a deep learning model. The proposed project aims to create an application that integrates the use of image recognition into lost and found services. The application allows lost item owners to upload an image of their item to search for a match in the database. For instance, the image matching process utilizes feature embedding to extract image features and a KNN algorithm to find similar images. This approach is expected to reduce labor costs while improving the efficiency and accuracy of the process. Therefore, it provides a low-cost, efficient, and practical solution to operators who currently do not offer lost and found services, ultimately benefiting all public transport users.
Acknowledgements

I would like to express my special thanks to my supervisor Dr. Lou Ping who gave me a valuable opportunity to work on this project to design a lost and found system. Without his advice and support, this project would not have been possible.

Secondly, I would also like to appreciate my teammates for their team spirit and enthusiasm.

Last but not least, I would like to thank you Miss Mable Choi assist my report writing.
Contents

Abstract i

Acknowledgements ii

List of Figures vi

List of Tables vii

Abbreviations & Acronyms viii

1 Overview and Background 1
  1.1 Background & motivation ........................................ 1  
      1.1.1 Frequency, Cost and Success Rate of Lost Item Retrieval 1  
      1.1.2 Improving Lost Item Retrieval by Technology .......... 1  
  1.2 Revolutionizing with Machine Learning Algorithms .......... 1  
  1.3 Designing an Application to Overcome Limitations of Image- 
      Identifying Processing ........................................ 2  
  1.4 Project Objective .............................................. 3  
  1.5 Project Contribution ............................................ 4  
  1.6 Outline of the report ........................................... 4

2 Image Identification with ML Algorithms 5
  2.1 Task definition .................................................. 5  
  2.2 Accuracy Score Calculation ..................................... 6  
  2.3 First Stage ...................................................... 7  
      2.3.1 Using k-Nearest Neighbor for Similar Image Identifi- 
          cation ...................................................... 7
2.3.2 Limitations of the KNN Model .......................... 8

2.4 Second Stage .................................................. 10
2.4.1 Improving Image Classification Accuracy with Feature Extraction ........................................... 10
2.4.2 Feature Extraction using classifier Embedding layer .............................................................. 11
2.4.3 Unsupervised learning - AutoEncoder ................. 13
2.4.4 Evaluation on Two Feature Extraction Methods ...... 15

2.5 Third Stage ......................................................... 17
2.5.1 Hyper parameter tuning - Distance formula ........ 17
2.5.2 Hyper parameter tuning - Latent size ................... 19

2.6 Summary .......................................................... 20

3 Implementation of Lost and Found Application ............. 21
3.1 Application Design .............................................. 21
3.1.1 User Role - Lost item owner ............................... 22
3.1.2 User Role - Operator Staff ................................. 23
3.1.3 Technology Stack ............................................ 24

3.2 Backend Implementation ........................................ 25
3.2.1 Database Modeling .......................................... 25
3.2.2 RESTful API .................................................. 27
3.2.3 Database Querying .......................................... 27
3.2.4 Authorization ................................................ 28
3.2.5 Caching on Image Latent .................................. 30

3.3 Application Flow ................................................. 30
3.3.1 Step 1: Operator Staff Upload a Newly Found Lost Item ......................................................... 31
3.3.2 Step 2: Lost Item Owner Submits a Request ........ 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.3</td>
<td>Step 3: Operator Staff Verify Information</td>
<td>35</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Alternative Method: Searching for Lost Items Using Tags</td>
<td>37</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Special case: Lost Item Owner Submitting a Request Before Operator Found</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>Application Features</td>
<td>41</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Featured Dashboard</td>
<td>41</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Shortcut Navigation</td>
<td>42</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Data Filtering, Sorting and Export</td>
<td>43</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Responsive Design</td>
<td>44</td>
</tr>
<tr>
<td>3.5</td>
<td>Summary</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>Conclusion</td>
<td>46</td>
</tr>
</tbody>
</table>

References | 47
List of Figures

1.1 Abstract Interaction Flow Diagram between Client and Server for Lost Item Retrieval ........................................ 3
2.1 Task Definition Flowchart ........................................ 5
2.2 Accuracy Score Calculation Sample ................................ 6
2.3 Example of how KNN works .................................... 7
2.4 Result on CIFAR10 ............................................... 9
2.5 Feature Extraction Process ...................................... 10
2.6 Extract Embedding from CNN model .......................... 11
2.7 Training Process of AutoEncoder ............................... 13
2.8 The Detail Architecture of Encoder and Decoder .............. 14
2.9 The Accuracy on Different Extractors in CIFAR10 .......... 15
2.10 The Test Sample on ResNet50 in CIFAR10 ................. 16
2.11 PCA 2D Projection on ResNet50 Latent Output in CIFAR10 17
2.12 The Training Loss with Different Latent Size .............. 19
3.1 Basic Flow for Lost Item Owners ............................... 22
3.2 Basic Flow for Operator Staff .................................. 23
3.3 Technology Stack ................................................. 24
3.4 Database Modeling ............................................... 26
3.5 The JWT Token Example ........................................ 29
3.6 Step to Create Form for Newly Found Lost Item ........... 31
3.7 Process on How Lost Item Owner Submits a Request ........ 33
3.8 Process on How Lost Item Owner Submits a Request (Continue) 34
3.9 Verification Process for Operator Staff ....................... 36
3.10 Checking Request Status by Reference Number ............ 37
3.11 Find Lost Items by Tags Description ......................... 38
3.12 Operator does not Found any Lost Items Same as the uploaded Image .................. 39
3.13 Operator Staff Select Image from Requests .................. 40
3.14 Lost Item Receive Confirmation Email .................. 41
3.15 Lost and Found Service Operator Dashboard .................. 42
3.16 Utility Bar for Filter, Sort and Export data .................. 43
3.17 Tablet and Mobile Screen for Dashboard .................. 44

List of Tables

2.1 Accuracy of Plain KNN on CIFAR10 Testing Images .......... 8
2.2 Metrics of Pre-trained CNN classifiers .................. 12
2.3 Accuracy on Different Distance Formula .................. 18
2.4 Metric on AutoEncoder with different latent size .................. 19
Abbreviations & Acronyms

List of Abbreviation

API Application Programming Interface.

CRUD Create, Read, Update, Delete.

CSRF Cross-Site Request Forgery.

JWT Json Web Tokens.

KNN K Nearest Neighbor.

MAE Mean Absolute Error.

ML Machine Learning.

ORM Object-relational mapping.

PCA Principal component analysis.

RESTful API Representational State Transfer API.

RMSE Root Mean Squared Error.
1 Overview and Background

1.1 Background & motivation

1.1.1 Frequency, Cost and Success Rate of Lost Item Retrieval

It is usual to find items left behind by other passengers on public transport and common facilities. Mass Transit Railway (MTR) found more than 7,000 lost items per month and 5,600 cases were reported but only 30 percent to 40 percent were successfully found.[1] Assuming each case requires around 5 minutes to process, it costs 262,500 HKD to deal with 84,000 cases per year according to the minimum wage of 37.5 HKD per hour.[2] This does not count cases on other lost items hot spots such as franchised buses, taxis, and airports.

1.1.2 Improving Lost Item Retrieval by Technology

In addition, verbal descriptions of lost items that are reported may be ambiguous or insufficient. As a result, the lost item owner may have to invest precious time and massive effort into finding their belongings. Therefore, many transport operators are not willing to spend resources on this inefficient service. Nevertheless, technological advancements have provided a solution for those who are still looking for their unique belongings. A comprehensive, systematic, and automatic lost and found application can reduce administrative costs, increase the success rate of found items, and allow for efficient handling of reported cases.

1.2 Revolutionizing with Machine Learning Algorithms

The use of advanced technology has significantly improved the lost and found service. One of the key components of any lost and found system is image-identifying processing, which plays a critical role in enabling computers to
recognize and analyze lost items based on information provided by users. Using a machine learning algorithm, the system extracts features from the pixels in the uploaded photo and compares them to a database of lost items, which may be represented as text or a vector. This process enables the system to identify similarities and generate matching results, which are then reported back to the user. Overall, the incorporation of image-identifying processing technology has revolutionized the lost and found service by improving accuracy, efficiency, and convenience for both users and operators.

1.3 Designing an Application to Overcome Limitations of Image-Identifying Processing

Although image-identifying processing is a crucial part of the lost and found system, it encounters practical limitations in recognizing the image accurately. Therefore, to improve the accuracy of the lost and found service, we plan to design a user-oriented application that addresses these limitations.

The application guide the user or operator to upload suitable information to the model by providing clear instructions and prompts. For instance, the application may include tools to adjust the resolution of the image to further enhance the accuracy of recognition.

Figure 1.1 shows the abstract interaction between the client-side application and the server-side application in the proposed lost and found system.
The client-side application instructs the user to take a photo and input necessary information such as the item category, date, and location. The image and information are then uploaded to the server-side application.

On the server-side, an image-identifying algorithm processes the image and compares its features with those of the lost items in the database. The algorithm then generates a report containing the most similar lost item to the user, along with the possible location where the item was found.

This abstract flow illustrates the proposed solution’s user-oriented design, which aims to simplify the lost and found process and increase the likelihood of finding lost items. By leveraging machine learning technology, the system can quickly and accurately identify lost items based on user-provided information and images.

Designing an application is essential to improve the functionality and efficiency of the lost and found service.

1.4 Project Objective

The main objective of this project is to design and develop a lost and found application that utilizes image-identifying processing to assist lost item owner
in finding their lost belongings. Also provides a comprehensive, systematic, and automatic flow to lost and found services. Additionally, the application allows for easy communication between the user and the server-side application.

1.5 Project Contribution

This free application is available for any transport operator to integrate into their lost and found services.

1.6 Outline of the report

This report is divided into four main chapters. The first chapter provides an introduction to the lost and found service, the motivation behind this project, and an overview of the report. The second chapter describes the machine learning model used in the application and how it is trained to identify lost items. The third chapter discusses the design and implementation of the lost and found application, including its architecture and functionality. The final chapter presents the conclusion of this report.
2 Image Identification with ML Algorithms

This chapter provides an overview of the machine learning model used for image identification. It also covers the process of training the model and evaluating its performance. The training data and parameters used are also discussed.

2.1 Task definition

The goal of this machine learning model is to analyze an input image and identify the most similar images from a database of images. Given a set of training images, the model learns to identify the important features of each image for comparison to other images.

Figure 2.1 illustrates the task definition of the image identification model, where the wallet image is presented as input on the left side, and the model is expected to identify the most similar images from a database represented by the eight images on the right side, as indicated by the arrow in the middle.

![Figure 2.1: Task Definition Flowchart](image)

Upload an image

Return a list of images by similarity order

This task requires the model to extract relevant features from the input image and compare them with those of images in the database to determine the best match.
2.2 Accuracy Score Calculation

To evaluate the performance of the machine learning model, we designed a similarity accuracy score calculation method. This method involved comparing the class labels of predicted similar images with the input image label, and assigning a score of one for each matching label. Figure 2.2 demonstrates this calculation method by displaying an input image labeled as ”ship” and the five similar images returned by the model, which had labels of ”ship”, ”ship”, ”airplane”, ”ship”, and ”airplane”.

![Accuracy Score Calculation Sample](image)

Figure 2.2: Accuracy Score Calculation Sample

The score for each data point was computed by counting the number of predicted similar images that had matching labels with the input image, as in the case of the first, second, and fourth images. The score was then divided by the total number of similar images, which was five in this case. The resulting score of 60% indicates that three out of the five similar images had matching labels with the input image.

To calculate the overall accuracy score of the model, we averaged the accuracy scores of all data points. This approach is aligned with the common expectation of users who use image search engines, which is to see a list of visually similar images that belong to the same class as the input image. By measuring the similarity of images based on their class labels, we can
evaluate the performance of our machine learning model.

2.3 First Stage

2.3.1 Using k-Nearest Neighbor for Similar Image Identification

The K Nearest Neighbor (KNN) algorithm is a non-parametric method that is commonly used to identify similar images. Figure 2.3 illustrates the algorithm’s concept, where the green and orange dots represent items that belong to categories A and B, respectively. Because items with the same category are closer together, we can find the k nearest neighboring points to a new data point by calculating its distance from each point in the database. This enables us to determine the top-k closer point of the new data point.

![Figure 2.3: Example of how KNN works](image)

To apply this model in our image search application, we first convert each image into a vector. During processing, we calculate the distance between each vector and the input image vector. The k nearest feature vectors are then output as the most similar images.

The KNN model offers a high degree of flexibility and versatility, as it does not require training, and the value of k can be easily adjusted as needed. This
makes it a suitable option for processing the output on demand. However, the model may face some limitations when dealing with a large set of input images.

### 2.3.2 Limitations of the KNN Model

The CIFAR10 dataset consists of 60,000 32x32 color images that are classified into 10 different classes. [3] The dataset is divided into 50,000 training images and 10,000 testing images.

Table 2.1 presents the accuracy of the KNN algorithm using the mean squared error (RMSE) metrics on 10,000 testing images. To mimic real-world scenarios, the training data used for this study consists of only 5,000 images, which is a relatively small set compared to the testing set of 10,000 images. This approach aligns with the real-world scenario where items that are not requested by users are not stored in the database for a long time. Additionally, once an item is taken away, it should not be counted.

<table>
<thead>
<tr>
<th>Model</th>
<th>k=1</th>
<th>k=3</th>
<th>k=5</th>
<th>k=7</th>
<th>k=9</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>27.01%</td>
<td>24.79%</td>
<td>23.93%</td>
<td>23.31%</td>
<td>22.82%</td>
</tr>
<tr>
<td>Random Guess</td>
<td>10.00%</td>
<td>10.00%</td>
<td>10.00%</td>
<td>10.00%</td>
<td>10.00%</td>
</tr>
</tbody>
</table>

The accuracy decreases as the value of k increases as expected. Furthermore, even with k=1, the accuracy of the model is still relatively low, suggesting that using KNN only may not be optimal. Compared to random guessing, the model still performs significantly better, indicating that it is able to capture some patterns in the data. However, there is still much room for improvement in the model’s accuracy.

In Figure 2.4, the model correctly predicts the image of the ship because the background in both images contains a white sky, red ship, and blue sea, making them similar. However, the model incorrectly predicts the image of
the plane as the ship because of the similar background. This suggests that
the model is not solely focused on recognizing the main object in an image
but is also influenced by the surrounding pixels. Consequently, the model
may yield poor results if the items do not fit the majority of pixels in an
image.

Figure 2.4: Result on CIFAR10

From these evaluations, it is apparent that not all pixels in an image con-
tain meaningful information for image classification. While the KNN model
shows some promising results, it is not sufficient to meet the requirements of
the application. It suggests that more advanced techniques, such as feature
extraction, are necessary to improve the accuracy of the model.
2.4 Second Stage

2.4.1 Improving Image Classification Accuracy with Feature Extraction

As discussed in the previous section, the fundamental KNN test results indicate that the model's accuracy is relatively low as humans do not classify images by examining every single pixel. Factors such as shape, size, color, and structure play a more significant role in determining the differences between images. Moreover, a signal image can contain millions of pixels so computation cost can become an issue for KNN. Therefore, a feature extractor is necessary before using KNN.

Figure 2.5 shows a schematic representation of a feature extraction process using a feature extractor. The input consists of multiple images that are fed into the feature extractor. The feature extractor extracts the features from each image and converts them into a latent feature vector. The latent feature vectors are then used as inputs to a K-nearest neighbor (KNN) algorithm, along with the latent features of the database images. The KNN algorithm searches for the most similar latent feature vectors to the input image's latent features and returns the corresponding database images. The output of the KNN algorithm is four images that are similar to the input image.

This process is useful in scenarios where there are large image databases, and it is not feasible to compare the input image with all images in the database. By using latent feature vectors, the KNN algorithm only needs to
compare the input image with a smaller set of latent feature vectors, which leads to a faster and more efficient retrieval process.

2.4.2 Feature Extraction using classifier Embedding layer

Extracting features using a pre-trained convolutional neural network (CNN) has become a popular method for image recognition tasks.

Figure 2.6 displays the architecture of a typical CNN, which consists of two main parts: the feature extractor and the classifier. The feature extractor comprises the early layers of the network, responsible for extracting low-level features such as edges and corners from the input image. The later layers of the network, known as the classifier, are responsible for assigning a label to the input image based on the features extracted by the feature extractor.

![Figure 2.6: Extract Embedding from CNN model](image)

We remove the last classification layer from each model and use the output of the previous layer as the feature representation for each image. By doing
so, it leverages the powerful feature extraction capabilities of these models without the need for fine-tuning or retraining.

Practically, PyTorch’s torchvision library provides some pre-trained classifiers. After analyzing the metrics of the pre-trained classifiers, we have chosen two of them for evaluation. In Table 2.2, the performance metrics of two pre-trained CNN classifiers are shown. The EfficientNet-B7 achieved higher accuracy than ResNet50, however, it also requires a larger memory to store its parameters and takes a longer time to run.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy on ImageNet</th>
<th>Params</th>
<th>GFLOPS</th>
<th>Latent Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>EfficientNet-B7</td>
<td>84.122%</td>
<td>66.3M</td>
<td>37.75</td>
<td>2,560</td>
</tr>
<tr>
<td>ResNet50</td>
<td>80.858%</td>
<td>25.6M</td>
<td>4.09</td>
<td>2,048</td>
</tr>
</tbody>
</table>

These two classifiers have a good balance between model size and computational power, making them suitable for deployment on a web server with decent performance.
2.4.3 Unsupervised learning - AutoEncoder

An AutoEncoder is a type of neural network that is trained to learn compressed representations of data. Figure 2.7 shows two parts: an encoder and a decoder. The encoder takes in an input image and compresses it into a smaller, encoded representation. This compressed representation contains the most important features of the input image. The decoder then takes this encoded representation and reconstructs the original image as closely as possible.

![Training Process of AutoEncoder](image)

During training, the network is optimized to minimize the difference between the original image and the reconstructed image, which encourages the network to learn a good compressed representation. This compressed representation can then be used as a feature extractor, by passing it as input to KNN model.

The beauty of this approach is that it is unsupervised, meaning that it does not require labeled data for training. This significantly reduces the cost and effort needed for data collection and labeling, making it a more efficient and scalable solution.

The architecture of our autoencoder is illustrated in Figure 2.8. It consists of an encoder with 3 convolution blocks and 2 linear layers. Each of the convolution blocks consists of a batch normalization layer, a convolution layer with stride 2, and a ReLU activation function. On the other hand,
the decoder performs the reverse process of the encoder. It also contains 3 convolution blocks and 2 linear layers. The batch normalization layer, transposed convolution layer with stride 2, and ReLU activation function are included in each of the convolution blocks.

![Diagram of Encoder and Decoder Architecture](image)

**Figure 2.8: The Detail Architecture of Encoder and Decoder**

Batch normalization is a technique used to recenter and rescale the input value of a neural network, making it easier for the network to learn and improve gradient propagation. The convolution layer, on the other hand, applies a kernel operation on the input image to extract features, and using stride 2 helps to downscale or upscale the image pixel. The convolutional layer plays a critical role in feature extraction. The two linear layers in the encoder and decoder allow the feature channels from the convolutional layer to exchange information and output the overall feature information.
2.4.4 Evaluation on Two Feature Extraction Methods

In this evaluation, the dataset used in the section 2.3.2 is reused. Figure 2.9 illustrates the performance of plain KNN, ResNet50, EfficientNet-B7, and the AutoEncoder as feature extractors with varying numbers of k.

![Graph](image)

Figure 2.9: The Accuracy on Different Extractors in CIFAR10

As expected, increasing the number of k decreases the accuracy. However, the classifier embeddings outperform the plain KNN and achieve an accuracy of over 80%. The EfficientNet-B7 outperforms the ResNet50 by a small margin. Although the AutoEncoder-based feature extractor performs better than the plain KNN, there is still room for improvement compared to the classifier embeddings.

In Figure 2.10, an input image is shown on top and five images are displayed at the bottom, which the ResNet50 and KNN determine as the most similar to the input image.
The model appears to capture the important features such as the shape of the deer, instead of simply determining similarity based on the background.

Figure 2.11 displays a 2D projection of the ResNet50 embeddings using Principal component analysis (PCA), where each color represents a different class. Prior to feature extraction, the data points were scattered throughout the plot. However, after applying the ResNet50 embeddings, the latent output brought the data points closer together, resulting in a more concentrated plot.
This suggests that the embedding layer is able to successfully capture the essential features of the images, and as a result, similar classes tend to cluster together in the reduced dimensional space.

Overall, the results demonstrate the effectiveness of using pre-trained classifier embeddings as feature extractors for image retrieval tasks.

2.5 Third Stage

In the previous section, it was shown that classifier embeddings are effective in extracting features. However, to further improve the accuracy, this section focuses on hyperparameter tuning to optimize the model.

2.5.1 Hyper parameter tuning - Distance formula

One of the hyperparameters that can be tuned is the choice of the distance formula used by the KNN algorithm. Both Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are common choices for distance formulas in image retrieval. In the following formulas, \( x^i \) represents each pixel on an image stored in the database, and \( y^i \) represents each pixel on an input image provided by the user.
\[ RMSE = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2} \]

\[ MAE = \sum_{i=0}^{n} |x_i - y_i| \]

Cosine Similarity = \[ \frac{X \cdot Y}{\|X\| \|Y\|} \]

\{x_1, x_2, ...x_i \in X\}, \{y_1, y_2, ...y_i \in Y\}

KNN estimates similarity based on the shortest distance between data points. Another option for distance calculation is cosine similarity, which calculates the angle between data points. The data point with the smallest angle is considered the most similar.

Table 2.3 displays the accuracy results of various distance formulas used.

<table>
<thead>
<tr>
<th>Model</th>
<th>RMSE</th>
<th>MAE</th>
<th>Cosine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain KNN</td>
<td>27.01%</td>
<td>29.04%</td>
<td>28.72%</td>
</tr>
<tr>
<td>ResNet50</td>
<td>80.71%</td>
<td>78.37%</td>
<td>80.46%</td>
</tr>
<tr>
<td>EfficientNet-B7</td>
<td>86.09%</td>
<td>84.68%</td>
<td>85.92%</td>
</tr>
<tr>
<td>AutoEncoder</td>
<td>37.5%</td>
<td>37.03%</td>
<td>48.30%</td>
</tr>
</tbody>
</table>

ResNet50 and EfficientNet-B7 outperform the others in terms of the root RMSE. On the other hand, the AutoEncoder-based feature extractor performs the best when using cosine similarity as the distance metric. Notably, the AutoEncoder-based feature extractor’s performance significantly improves when using cosine similarity compared to other distance metrics. In general, the Efficientnet-B7 model performs the best, specifically when using the RMSE distance metric.
2.5.2 Hyper parameter tuning - Latent size

To optimize the feature representation of the images, the size of the latent space can be adjusted as it can affect the amount of information contained within. To find the optimal size of the latent space for the AutoEncoder, the last layer of the encoder and the first layer of the decoder are modified. Table 2.4 presents the accuracy, loss and number of parameters of the model with different latent sizes.

<table>
<thead>
<tr>
<th>Latent Size</th>
<th>Accuracy</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>46.01%</td>
<td>8,856,393</td>
</tr>
<tr>
<td>128</td>
<td>48.30%</td>
<td>9,118,729</td>
</tr>
<tr>
<td>256</td>
<td>48.81%</td>
<td>9,643,401</td>
</tr>
<tr>
<td>512</td>
<td>47.90%</td>
<td>10,692,745</td>
</tr>
<tr>
<td>1024</td>
<td>46.86%</td>
<td>12,791,433</td>
</tr>
</tbody>
</table>

Table 2.4: Metric on AutoEncoder with different latent size

AutoEncoder with a latent size of 256 achieved the highest accuracy overall. As expected, the size of the model increases as the latent size increases. While the accuracy improves as the latent size increases from 64 to 256, it drops after adding more latent from 256 to 1024. This could suggest that 256 latent size is sufficient to contain the information of the image.

The trend of training loss with increasing iterations is depicted in Figure 2.12.

![Figure 2.12: The Training Loss with Different Latent Size](image)

Figure 2.12: The Training Loss with Different Latent Size
The latent size of 64 seems to be insufficient for reconstructing the image, as evidenced by the fact that the training loss does not decrease significantly with more iterations. Moreover, as the latent size is increased beyond 256, the training loss becomes relatively stable and does not show significant improvement.

2.6 Summary

In this chapter, the models for image matching were introduced and discussed. The image features were extracted using a feature extractor and the most similar image was computed using the K Nearest Neighbor (KNN) algorithm based on those latent features. Moreover, the classifier-embedding technique shows to achieve better results in feature extraction and the use of Root Mean Squared Error (RMSE) was recommended. In the next chapter, the implementation of the models is discussed, along with their integration into the system.
3 Implementation of Lost and Found Application

The Lost and Found application is designed to provide an efficient platform for reporting and searching lost and found items. This chapter discusses the implementation details of the application, including the technologies used, system architecture, and user interface design. Additionally, it covers the features of the application and how they were implemented.

3.1 Application Design

This section outlines the basic flow for the two main user roles in the Lost and Found application: the lost item owner and the operator staff. It describes the actions they take and how they exchange information. Additionally, it includes the technology stack used to implement this complex application.
3.1.1 User Role - Lost item owner

If a lost item owner is looking for their missing belonging, they can visit the lost and found service website and provide relevant information. Figure 3.1 illustrates the process where the user uploads an image of the lost item and the server returns a list of similar images for the user to choose from. Once the user selects the most similar image, they can provide their contact information. The server then generates a unique reference number for the user to store.

This mechanism eliminates the need for users to create an account or log in. Instead, a unique reference number is generated by the server, which the user can use to track the status of their request. Once the staff has verified and approved the information provided by the user, the status is updated accordingly. This not only provides a more convenient way for the user but also reduces the server's resource usage.
3.1.2 User Role - Operator Staff

The operator staff can upload the image, date, and venue of the found item to the server database. Figure 3.2 illustrates the basic flow for operator staff. Since the server receives the lost item request over time, the staff is also responsible for verifying the request information and either approving or rejecting the requests.

This approach not only prevents abuse of the application but also reduces the workload of the staff. Image recognition is performed by the image matching algorithm and the lost item owner. The staff only needs to verify that the information matches, such as the lost venue and date, to ensure that the item actually belongs to the owner. The verification process only requires the staff to approve or reject, which allows them to process verifications quickly.
3.1.3 Technology Stack

The Lost and Found application offers various functionalities such as database data storage, running image matching algorithms, and email notifications. To provide these functionalities to users, an application interface is necessary. Figure 3.3 illustrates the application architecture and the coding frameworks used for each component. React is used for the frontend to provide a user interface and interactive components for viewing data and performing actions. Django is used for the backend to provide an interface for database querying, running PyTorch functions, and connecting to the SMTP server to email users.

![Technology Stack Diagram]

Figure 3.3: Technology Stack

The Application Programming Interface (API) design of the application is based on the Django Rest Framework, which enables the frontend application to perform actions via AJAX calls with Create, Read, Update, Delete (CRUD) actions. The user interface design follows the principles of Material Design, which is a design developed by Google. It enables applications to have a user-friendly and visually pleasing interface, making it intuitive and easy for users to navigate.
3.2 Backend Implementation

This section discusses various aspects of the application’s backend implementation before delving into the frontend. It covers topics such as database modelling, API design, authorization system, and optimization techniques.

3.2.1 Database Modeling

In order to maintain data consistency and persistence, a relational database design was implemented. The database design is shows in Figure 3.4, which shows the five main entities of the application: Staff, Lost Item, Requests, Venue, and Tags. The lines connecting these entities represent the relationships between them.

The Staff entry in the database stores the username and hashed password for authentication purposes. The Lost Item entry contains the image of the lost item, the date it was found, and its inventory status. The inventory status helps to prevent the image matching algorithm from returning images that have already been returned to their owners. The Request entry not only contains the information about the lost item, such as the image and the found date claimed by the owner, but also the contact information for the user so that the system can send notifications. The status of the request can be pending, approved, or rejected. Finally, the Venue and Tag entries are stored separately because they are used for filtering purposes.
Each staff member can report multiple lost items and verify multiple requests, and each lost item and verification can only be done by one staff member. A lost item can only be associated with one venue, but it can be described with multiple tags. On the other hand, a request can be associated with multiple tags, but unlike a lost item, it can be associated with multiple venues since the lost item owner may have visited more than one place. Each lost item can only be matched with exactly one request.
3.2.2 RESTful API

Representational State Transfer API (RESTful API) is a type of web API that uses HTTP requests to interact with data resources using standard CRUD operations. The RESTful API follows a set of constraints and principles to provide a scalable and flexible architecture for web applications.

To implement a RESTful API, resources in the application can be mapped to endpoints. In our case, resources such as lost items, requests, tags, and venues can have corresponding endpoints, such as "/lostitem", "/request", "/tag", and "/venue". These endpoints can support HTTP methods such as "GET", "POST", "PUT", "PATCH", and "DELETE", which correspond to viewing, creating, updating entirely, updating partially, and removing data respectively. Additionally, a POST endpoint "/lostitem/similiar" is available for the image matching algorithm to return similar lost items based on an uploaded image. Another endpoint, POST "/email", allows staff to send email notifications to lost item owners.

3.2.3 Database Querying

In Django, it provides Object-relational mapping (ORM) to interact with a relational database using an object-oriented programming language. It means manipulating the database through objects and methods instead of SQL queries.

As an example, instead of constructing SQL queries like "Select id, image from lostitem" to retrieve all the data for the "lostitem" table with only the "id" and "image" columns, Django’s ORM allows us to write code like "lostitem.all().only("id", "image")", which provides a more convenient and intuitive way to interact with the database.

However, if a table attribute contains related data, such as lost items having multiple tags, and an endpoint like GET "/lostitem" needs to display both the tag "id" and "name", it can lead to N+1 queries where initial
query is followed by N additional queries to retrieve related data. This can result in inefficient database usage and reduced performance if "lostitem.all().prefetch_related()" is not used. Therefore, the ORM has a potential downside in causing performance issues and each query that involves relational data should be carefully checked.

3.2.4 Authorization

Some of the endpoints in the API are secured, such as "/lostitem" that can only be accessed by operator staff to create, update, and delete. If an unauthorized user tries to access the secure endpoint, they should be rejected and receive a "401 Unauthorized" error.

In order to verify a user’s identity, they are required to provide their username and password. The server then compares the password provided by the user to the hashed password stored in the database to authenticate the user. Storing hashed passwords instead of plain text passwords helps to avoid the risk of database passwords exposing. However, requiring the operator staff to log in for each request they send may be inconvenient. To address this, an authorization token can be used to maintain the login status of the user.

One common way to attach an authorization key to API requests is through the use of cookies. However, cookies are susceptible to Cross-Site Request Forgery (CSRF) attacks. In a CSRF attack, a user may unknowingly click on a malicious website that contains a DELETE API request to ":our app domain>/lostitem/1". Even if the malicious website does not have access to the user’s cookies, the API request still carry the authorization token, and the lositem 1 is deleted without the user’s consent. Therefore, an alternative approach is to store the authorization token in the Authorization header of the API request. This way, the token is not automatically included in requests that are initiated from malicious websites, and it provides a more
secure way to authenticate API requests.

To verify an authorization token, there are two methods. One is for the server to assign a random token to the user and store it in the database. When the server receives the user’s token, it checks if the token exists in the database. However, this approach requires an extra database query, which creates overhead. Alternatively, Json Web Tokens (JWT) provide a solution. In Figure 3.5, the encoded token is represented by the red, purple, and blue sections, which correspond to the header, payload, and verify signature, respectively. The header specifies the algorithm used by the JWT, the payload includes the username and expiration time of the token, and the verify signature is used by the server to verify the token.

To authenticate the user, the server can hash the header, payload, and secret key, and compare the resulting hash value with the verify signature in the JWT. If they match, the token is valid and the user is authorized to access secure endpoints. This process doesn’t require any database queries and can be done at runtime. Although this method does not provide the
ability to force the user to log out, using a short-lived token can help prevent token abuse. In conclude, the JWT token is used in the authorization header to authenticate the user.

3.2.5 Caching on Image Latent

As discussed in section 2.4.1, feature extraction involves extracting latent features from the images. These latent features are used to compare the similarity between images. Hence, in the database entries for lost items and requests, a latent image attribute is included to store the corresponding latent features. In Django, this attribute converts the latent tensor to a JSON format stored in an array. Consequently, during comparisons, the server does not need to open the image file but can directly convert the JSON to a tensor. The feature extractor only needs to be run once when creating or replacing an image. This approach significantly reduces the time spent on IO operations.

If the feature extraction model or the size of the latent is updated, it could potentially cause issues in the application. To avoid this, an additional attribute should be added to the database to store the version of the latent. If the image’s latent version matches the current version, it can be reused without needing to extract the feature again. Otherwise, the feature extractor should be rerun to obtain the updated feature for the image.

3.3 Application Flow

This section provides an overview of the application flow and shows the interactions between the operator staff, lost item owner, and the server. The following special cases are also discussed at the end.
3.3.1 Step 1: Operator Staff Upload a Newly Found Lost Item

Figure 3.6 illustrates the process of how operator staff can input information for a newly found lost item. In the first step, staff can visit the "create new item" page either through the dashboard or directly by using the link "/dashboard/create-item". Once the staff uploads the image, a form will be displayed for the staff to input the necessary information.

In the first component, the operator staff can input multiple tags to describe the lost item. The server provides default tags based on the image, which the staff can modify or remove. The staff can also create new tags, search for other tags, or remove suggested tags. For example, the staff can add the "logitech" tag from the pre-defined tags. Additionally, the staff can
select the venue and date where they found the item.

Once the staff submits the form, the information of the newly found lost item is stored in the database. The staff can then search for the item later by its tags, date, or venue. The lost item owner also be able to view that image.

### 3.3.2 Step 2: Lost Item Owner Submits a Request

In figure 3.8, the lost item owner first visits the index page of the application. They then select the service they need, which in this case is "Find My Lost Item". They are prompted to upload an image of the lost item. The server then returns a set of similar images based on the uploaded image. The images are displayed in a grid format, with the most similar image at the top left and the least similar image at the bottom right. The lost item owner can select an image that closely resembles their lost item. If needed, they can zoom in on the image to check for details or possible cropping.
Once the lost item owner has selected an image, they need to provide additional information such as their name, the date and venue the item was lost, and their contact details. This information is used by the staff to verify and notify the owner if their lost item is found.
If the form is submitted successfully, the lost item owner will be redirected to a page that shows the reference number and status of their request. The lost item owner can save the reference number for future use to check the status of their request. In Figure 3.8, the "Pending" status means that the operator has received the form and the request is being processed soon.

In this scenario, the image uploaded in the previous section (Section 3.3.1) is displayed. Despite the difference in perspective of the lost item, the system is still able to identify it and return the top 5 most similar items, which are all mouses.
3.3.3 Step 3: Operator Staff Verify Information

To verify the information submitted by the lost item owner, the operator staff can navigate to "dashboard/requests" to view the list of requests. The table is automatically sorted by the submission date, with the most recent requests appearing at the top. To enter the verification page, the staff can click on the information icon on the right-hand side of the table. On this page, the contact information of the requester is displayed, as well as the information provided by the lost item owner and the information recorded by the operator staff.
The requester has claimed to have lost the item on 2023-03-17 and the operator staff also found it on the same date. Additionally, the requester has also claimed to have visited three places on that date, one of which matches where the operator staff found the item. Therefore, the operator staff can approve the request by clicking on the "Approve" button below.
In figure 3.10, we can see how a lost item owner checks their status. The user first visits the index page and clicks on "Check My Report Status" this time. They are then directed to a page where they could enter the reference number they previously stored. After entering the reference number, the user is redirected to the status page shown in section 3.3.1. Now, the status has been updated to "Approved," which means the lost item owner can go to retrieve his/her lost item.

3.3.4 Alternative Method: Searching for Lost Items Using Tags

Although some lost item owners may not have an image of their lost item, they can still try to search for it using the "Search by Tag" function. This function allows the lost item owner to input one or multiple tags that describe their lost item, such as "phone", "wallet", "keys", etc. After submitting the
Figure 3.11: Find Lost Items by Tags Description

form, the server will return a list of lost items that match the tags entered by the lost item owner. The lost item owner could then browse through the list and see if any of the items listed match their lost item.

If the lost item owner finds a match, they could click on the item and provide the necessary information for verification and notification, as described in section 3.3.1.

3.3.5 Special case: Lost Item Owner Submitting a Request Before Operator Found

Another special situation arises when a user submits a request for a lost item before the actual lost item owner does. For example, the lost item owner might not find their lost item on the similar image page shown in figure 3.12. In this case, the lost item owner could click the "I Do Not Find My Item" button and provide their contact information as before.
When the operator staff uploads a new found item, the server checks if there are any pending requests for lost items with the same date and venue. If a match is found, the server runs the image matching algorithm and returns a list of similar images. The operator staff could then select the image that matches the found item.
The lost item owner receives an email notification from the system informing them that a potential match has been found for their lost item. The email contains a link to confirm if the image shown is indeed their lost item. Once the lost item owner clicks on the link and confirms that the image matches their lost item, the status of the request is changed to "Approved". This confirms that both parties have confirmed that the lost item has been found and the owner could go and collect their item.
3.4 Application Features

In addition to the primary workflow, the application also includes additional features to enhance the experience for operator staff.

3.4.1 Featured Dashboard

Every time the operator staff logs in, they are directed to the dashboard page, as shown in Figure 3.15. The dashboard page provides useful data such as the number of requests and lost items found for the current month, top 5 lost item hot spots, and the most popular tags.
These statistics provide valuable insights for the operator staff to make informed decisions and improve the efficiency of the lost and found system. For example, the requests report and lost item found report can help the operator staff identify patterns and trends in lost and found items, which can inform decisions on resources allocation such as how many staff need to be assigned to lost and found service or post a reminder on lost item hot spots.

3.4.2 Shortcut Navigation

The application has various features that provide shortcut navigation to enhance user-friendliness. For instance, the "Remaining Pending Requests" card on the dashboard page, as shown in figure 3.15, allows the operator staff to quickly access the verification page by clicking on it. Similarly, clicking on any of the popular tags in the "Popular Tags" card direct the staff to the lost item inventory page filtered by that particular tag. This feature enables the staff to quickly view items that have been labeled with specific tags.
3.4.3 Data Filtering, Sorting and Export

Figure 3.16 demonstrates various functionalities, including the ability to show/hide columns in the user interface, adjust the density or height of each column, and filter the table based on the field type. The filtering options available depend on the type of field being filtered. For instance, if the field is a date, the filter options could be before or after a specific date.

![Utility Bar for Filter, Sort and Export data](image)

Figure 3.16: Utility Bar for Filter, Sort and Export data

In case the existing functionalities are not enough for the operator staff, they can also export the data in CSV format for further data analysis. This can be done by clicking the "Export" button, as shown in Figure 3.16. The
exported file can then be imported into other data analysis software.

### 3.4.4 Responsive Design

The application’s screen design is not limited to what is shown in figure 3.15, which represents the layout on a large computer screen. The design is also optimized for display on tablets and mobile devices, as shown in figure 3.17.

![Figure 3.17: Tablet and Mobile Screen for Dashboard](image)

The web application is designed to be responsive and detect the width of the user’s screen device, providing a different user interface accordingly. This feature allows the operator staff to easily take a picture of the lost item while collecting it, without having to enter all the information at once. It also enables lost item owners to access the page using different devices without any inconvenience.
3.5 Summary

This section provides an overview of the application design, which includes user roles, technology stack, and backend implementation. It also covers the database modeling, RESTful API, database querying, authorization, and caching on image latency. Additionally, it explains the application flow, including the main flow and special cases. Finally, it discusses application features, such as the featured dashboard, shortcut navigation, data filtering, sorting and export, and responsive design.
4 Conclusion

Managing a lost and found service can be a significant expense for small and medium-sized companies. However, recent advances in technology have made it possible to develop a more cost-effective system by utilizing deep learning models such as KNN with feature extractor for image identification. By implementing a user-friendly application, this project aims to simplify the complicated process of managing lost and found items and improve the efficiency and success rate of the system. The overall result will benefit both operators who do not currently offer lost and found services and the general public.
References

