

DẠI HỌC QUỐC GIA, THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH



CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT (CO2003)

BÀI TẬP LỚN

**Deduplicate image based
on feature hashing techniques**

GVHD: TS. Lê Thành Sách

SV thực hiện: Huỳnh Gia Bảo – 2410233
Nguyễn Gia An – 2410021
Nguyễn Hữu Phước – 2412806
Lại Trần Trí – 2413631

TP Hồ Chí Minh, Tháng 11 Năm 2025

Source code: <https://github.com/baohuynh12056/HashDuplicationImage>



Mục lục

1 Giới thiệu	1
1.1 Tính cấp thiết của đề tài	1
1.2 Mục tiêu nghiên cứu	1
1.3 Cấu trúc báo cáo	2
2 Cơ sở lý thuyết	3
2.1 Giải thuật hash	3
2.2 Ứng dụng	3
2.3 Bảng băm	4
2.4 Bloom Filter	4
2.5 Local Sensitive Hash	6
2.6 Random Projection for Locality Sensitive Hashing	6
2.7 SimHash	8
2.8 MinHash	9
2.9 Hàm băm MurmurHash3	11
2.10 Phương pháp TF-IDF	12
3 Xây dựng hệ thống phát hiện và phân loại ảnh trùng	14
3.1 Kiến trúc tổng thể của hệ thống	14
3.2 Tiền xử lý ảnh	15
3.3 Trích xuất đặc trưng	15
3.4 Hashing và ảnh xạ đặc trưng	16
3.5 Tối ưu hóa ngưỡng tự động	21
3.6 Cơ chế tính điểm chất lượng ảnh	23
4 Thực nghiệm và đánh giá kết quả	25
4.1 Giới thiệu sơ về dataset	25
4.2 Đánh giá độ chính xác	25
4.3 Đánh giá thời gian xử lý	27
4.4 Đánh giá chỉ số F1-Score	28
4.5 Tổng hợp và kết luận	29
5 Demo	30
5.1 Ý tưởng	30
5.2 Deduplication	30
5.3 Visualization	30



6 Hạn chế và hướng phát triển	32
6.1 Hạn chế của hệ thống hiện tại	32
6.2 Hướng phát triển	33
7 Lời Kết	35
Tài liệu tham khảo	36

1 Giới thiệu

1.1 Tính cấp thiết của đề tài

Trong kỷ nguyên số hiện nay, Internet và các nền tảng mạng xã hội phát triển với tốc độ chóng mặt, kéo theo sự bùng nổ khổng lồ về lượng dữ liệu hình ảnh được tạo ra mỗi ngày. Tuy nhiên, đi kèm với sự phát triển này là những thách thức lớn về việc quản lý và xử lý dữ liệu thừa:

- **Vấn đề trong huấn luyện trí tuệ nhân tạo (AI):** Các bộ dữ liệu (dataset) dùng để huấn luyện các mô hình Deep Learning/Machine Learning thường được thu thập tự động từ Internet (crawling). Dữ liệu thô này chứa rất nhiều "rác" và các hình ảnh trùng lặp. Việc để sót các dữ liệu trùng lặp không chỉ làm tăng thời gian huấn luyện vô ích mà còn gây ra hiện tượng *data leakage* (rò rỉ dữ liệu) giữa tập train và test, hoặc khiến mô hình bị lệch (bias), ảnh hưởng nghiêm trọng đến độ chính xác thực tế.
- **Lãng phí tài nguyên hệ thống:** Đối với các nền tảng lưu trữ và chia sẻ ảnh, việc người dùng tải lên nhiều lần cùng một bức ảnh hoặc các bức ảnh gần giống nhau gây lãng phí lớn về không gian lưu trữ (storage) và băng thông mạng (bandwidth). Chi phí duy trì hạ tầng cho lượng dữ liệu dư thừa này là rất lớn.
- **Trải nghiệm người dùng và Spam:** Trên các mạng xã hội, việc xuất hiện tràn lan các hình ảnh spam giống hệt nhau hoặc các nội dung bị re-up liên tục làm giảm chất lượng trải nghiệm của người dùng, gây loãng thông tin.

Xuất phát từ thực tế trên, nhu cầu về một giải pháp tự động để phát hiện và xử lý ảnh trùng lặp là vô cùng cấp thiết. Hệ thống phát hiện ảnh trùng (Image Deduplication) ra đời không chỉ giúp tiết kiệm chi phí lưu trữ, tối ưu hóa băng thông mà còn là bước tiền xử lý (preprocessing) quan trọng để làm sạch dữ liệu đầu vào cho các bài toán thị giác máy tính cao cấp hơn.

1.2 Mục tiêu nghiên cứu

Dựa trên những vấn đề đã đặt ra, đề tài tập trung xây dựng một hệ thống hoàn chỉnh với các mục tiêu cụ thể sau:

1. **Phát hiện ảnh trùng lặp:** Xây dựng thuật toán có khả năng phát hiện các ảnh trùng lặp hoàn toàn hoặc gần giống nhau (near-duplicate) về mặt nội dung, bất chấp sự thay đổi nhỏ về kích thước, định dạng hay nhiễu.



2. **Phân loại và gom nhóm:** Tự động gom nhóm các ảnh tương đồng vào các thư mục riêng biệt một cách hiệu quả để dễ dàng quản lý.
3. **Tối ưu hóa lựa chọn ảnh:** Cài đặt cơ chế tự động chọn ra hình ảnh có chất lượng tốt nhất (độ phân giải cao nhất, rõ nét nhất) trong mỗi nhóm ảnh tương đồng để giữ lại, hỗ trợ việc loại bỏ các bản sao kém chất lượng.
4. **Đánh giá hiệu năng thuật toán:** Thực hiện so sánh hiệu năng và độ chính xác của nhiều thuật toán băm khác nhau để tìm ra phương pháp tối ưu nhất.
5. **Xây dựng ứng dụng thực tế:** Dóng gói hệ thống thành một website hoặc ứng dụng demo, cho phép người dùng tải ảnh lên và trực quan hóa kết quả phát hiện trùng lặp.

1.3 Cấu trúc báo cáo

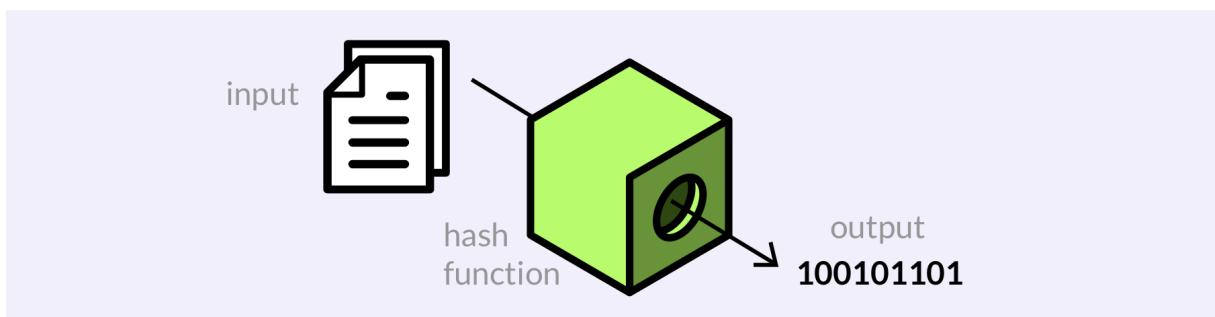
Báo cáo được trình bày theo trình tự logic từ lý thuyết đến thực nghiệm, bao gồm 8 phần chính:

- **Chương 1 - Giới thiệu:** Trình bày tính cấp thiết của đề tài, mục tiêu nghiên cứu và cấu trúc tổng quan của báo cáo.
- **Chương 2 - Cơ sở lý thuyết:** Giới thiệu các kiến thức nền tảng về thuật toán băm.
- **Chương 3 - Xây dựng hệ thống phát hiện và phân loại ảnh trùng:** Mô tả chi tiết kiến trúc hệ thống, quy trình tiền xử lý ảnh, trích xuất đặc trưng, kỹ thuật Hashing, tối ưu hóa ngưỡng tự động và cơ chế tính điểm chất lượng ảnh.
- **Chương 4 - Thực nghiệm và đánh giá kết quả:** Giới thiệu về dataset. Phân tích chi tiết kết quả đạt được về độ chính xác, thời gian xử lý và so sánh hiệu quả giữa các phương pháp khác nhau.
- **Chương 5 - Demo:** Trình bày ý tưởng, quy trình deduplication và visualization trên ứng dụng thực tế.
- **Chương 6 - Hạn chế và hướng phát triển:** Nhận diện các điểm hạn chế của hệ thống hiện tại và đề xuất hướng cải thiện trong tương lai.

2 Cơ sở lý thuyết

2.1 Giải thuật hash

Hàm băm (hash function) là giải thuật nhằm sinh ra các giá trị băm tương ứng với mỗi khối dữ liệu (có thể là một chuỗi kí tự, một đối tượng, một hình ảnh,...). Giá trị băm đóng vai trò như một khóa (key) để phân biệt các khối dữ liệu, tuy nhiên, người ta chấp nhận hiện tượng trùng khóa hay còn gọi là đụng độ (collision) và cố gắng cải thiện các giải thuật để giảm thiểu sự đụng độ đó. Hàm băm thường được dùng trong bảng băm nhằm giảm chi phí tính toán khi tìm một khối dữ liệu trong một tập hợp (nhờ việc so sánh các giá trị băm nhanh hơn việc so sánh những khối dữ liệu lớn).



Hình 1: Hash function

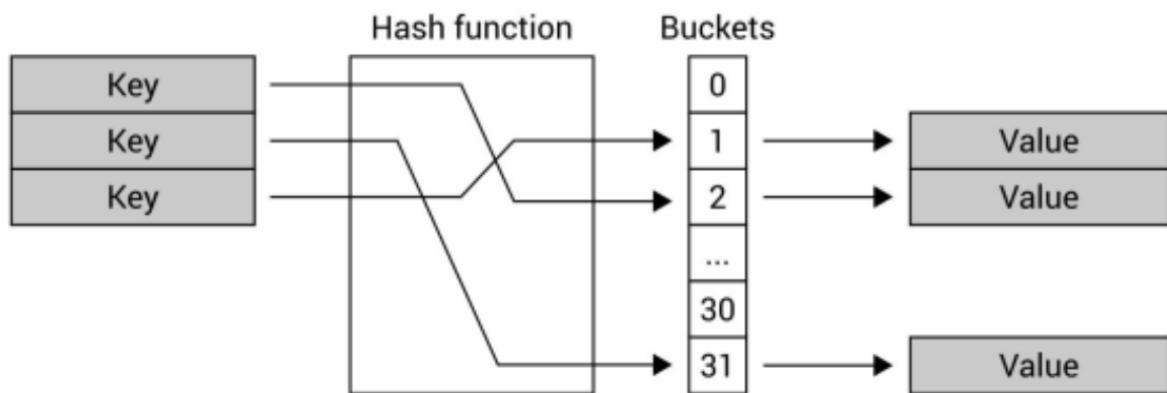
2.2 Ứng dụng

Các hàm băm được ứng dụng trong nhiều lĩnh vực, chúng thường được thiết kế phù hợp với từng ứng dụng. Ví dụ, các hàm băm mật mã học giả thiết sự tồn tại của một đối phương - người có thể cố tình tìm các dữ liệu vào với cùng một giá trị băm. Một hàm băm tốt là một phép biến đổi "một chiều", nghĩa là không có một phương pháp thực tiễn để tính toán được dữ liệu vào nào đó tương ứng với giá trị băm mong muốn, khi đó việc giả mạo sẽ rất khó khăn. Một hàm một chiều mật mã học điển hình không có tính chất hàm đơn ánh và tạo nên một hàm băm hiệu quả; một hàm trapdoor mật mã học điển hình là một hàm đơn ánh và tạo nên một hàm ngẫu nhiên hiệu quả ...

Các hàm băm còn được ứng dụng trong việc nhận dạng âm thanh, chẳng hạn như xác định xem một file MP3 có khớp với một file trong danh sách một loạt các file khác hay không.

2.3 Bảng băm

Bảng băm (hash table, còn được gọi là hash map) là một kiểu dữ liệu có vai trò rất quan trọng trong thực tế, thường được dùng như một từ điển: mỗi phần tử trong bảng băm là một cặp (khóa, giá trị). Nếu so sánh với mảng, khóa được xem như là chỉ số của mảng, còn giá trị giống như giá trị mà ta lưu tại chỉ số tương ứng. Bảng băm không như các loại từ điển thông thường - ta có thể tìm được giá trị thông qua khóa của nó.



Hình 2: Hash table

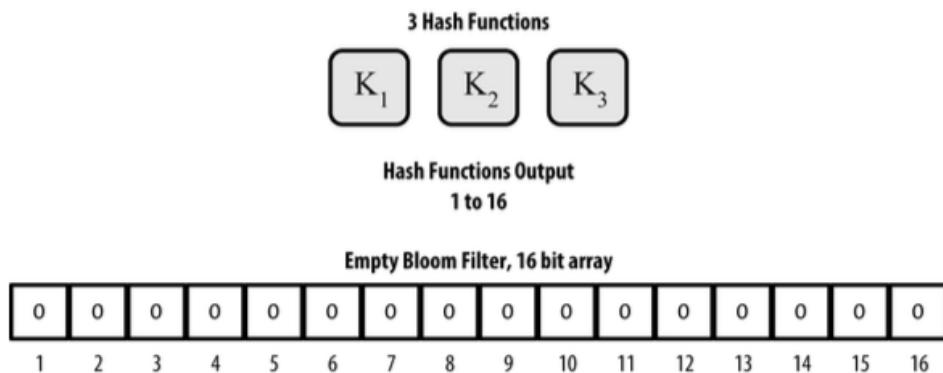
Một trong những yêu cầu quan trọng nhất của bảng băm là khả năng tra cứu nhanh một giá trị (value) dựa trên khóa (key) của nó. Yêu cầu đặt ra là nó cần có độ phức tạp hằng số $O(1)$.

2.4 Bloom Filter

Bloom Filter là một cấu trúc dữ liệu xác suất được Burton Howard Bloom giới thiệu năm 1970 [1], nhằm kiểm tra xem một phần tử có thuộc tập hợp hay không mà không cần lưu trữ toàn bộ các phần tử. Bloom Filter cho phép xác định chắc chắn rằng phần tử không thuộc tập, nhưng khi kết quả là “thuộc”, thì chỉ đúng với **một xác suất nhất định** (tồn tại khả năng dương tính giả – *false positive*).

Nhờ đặc tính tiết kiệm bộ nhớ và tốc độ truy vấn nhanh, Bloom Filter được ứng dụng rộng rãi trong các hệ thống kiểm tra tồn tại phần tử, như cơ sở dữ liệu phân tán, bộ nhớ đệm, phát hiện URL trùng, hay loại bỏ trùng lặp trong tập dữ liệu lớn.

Một Bloom Filter bao gồm hai thành phần chính. Mảng bit có kích thước m bit, ban đầu tất cả được gán giá trị 0. Bộ hàm băm gồm k hàm băm độc lập: $h_1(x), h_2(x), \dots, h_k(x)$. Mỗi hàm băm ánh xạ một phần tử x vào một vị trí trong mảng bit: $h_i(x) \in [0, m - 1]$.



Hình 3: Cấu trúc Bloom Filter ($m = 16, k = 3$)

Khi thêm phần tử x vào Bloom Filter, ta tính k giá trị băm của x và đặt các bit tương ứng trong mảng thành 1. Khi kiểm tra một phần tử y , nếu tất cả các bit tại vị trí băm đều bằng 1 thì y có thể thuộc tập (có khả năng dương tính giả), nếu bất kỳ bit nào bằng 0 thì y chắc chắn không thuộc tập.

Dưới đây là mô tả chi tiết bằng mã giả cho các thao tác cơ bản của Bloom Filter:

Algorithm 1 Bloom Filter

```

1: Khởi tạo: Mảng bit  $B[0 \dots m - 1] \leftarrow 0$ 
2: Đầu vào: Số hàm băm  $k$ , tập hàm băm  $h_1, h_2, \dots, h_k$ 
3: function INSERT( $x$ )
4:   for  $i = 1$  to  $k$  do
5:      $index \leftarrow h_i(x)$ 
6:      $B[index] \leftarrow 1$ 
7:   end for
8: end function
9: function QUERY( $y$ )
10:  for  $i = 1$  to  $k$  do
11:    if  $B[h_i(y)] == 0$  then
12:      return False
13:    end if
14:  end for
15:  return True
16: end function

```

2.5 Local Sensitive Hash

Local Sensitive Hash là một kỹ thuật băm nhạy cảm với độ tương tự được Andrei Broder giới thiệu vào năm 1997 [2], nhằm mục đích tìm kiếm xấp xỉ trong không gian dữ liệu có kích thước lớn. Thay vì yêu cầu so khớp chính xác, LSH cho phép phát hiện các phần tử “gần giống nhau” với xác suất cao — nghĩa là hai phần tử càng tương tự thì khả năng chúng có cùng giá trị băm càng lớn.

LSH được ứng dụng rộng rãi trong các lĩnh vực như truy xuất ảnh, phát hiện trùng văn bản, tìm kiếm tài liệu tương tự, và trong các hệ thống gợi ý. Cơ chế hoạt động giúp giảm đáng kể độ phức tạp khi xử lý dữ liệu lớn, đặc biệt khi so sánh trực tiếp từng cặp phần tử là quá tốn kém.

Ý tưởng chính của LSH là xây dựng một tập các hàm băm “nhạy cảm” với độ tương tự. Giả sử có hai đối tượng x và y , với độ tương tự được đo bằng một hàm $sim(x, y)$. Một hàm băm h được gọi là *local sensitive* nếu xác suất để $h(x) = h(y)$ tăng theo độ tương tự $sim(x, y)$. Cụ thể:

$$P[h(x) = h(y)] = f(sim(x, y))$$

với f là một hàm đơn điệu tăng.

Để tăng độ tin cậy, nhiều hàm băm được nhóm lại thành *buckets* — giúp đảm bảo rằng các phần tử tương tự nhau có khả năng cao rơi vào cùng nhóm. Khi cần tìm kiếm lân cận của một phần tử, chỉ các phần tử trong cùng hoặc lân cận bucket mới được xét, từ đó giảm đáng kể số lượng phép so sánh.

2.6 Random Projection for Locality Sensitive Hashing

Random Projection for Locality Sensitive Hashing là một phương pháp băm nhạy cảm với độ tương tự dựa trên phép chiếu ngẫu nhiên trong không gian vector. Kỹ thuật này thường được sử dụng để đo mức độ tương tự giữa các điểm dữ liệu trong không gian có số chiều lớn, đặc biệt trong các bài toán tìm kiếm láng giềng gần đúng.

Ý tưởng chính của phương pháp này là: hai vector đầu vào x và y càng gần nhau trong không gian (theo khoảng cách cosine hoặc Euclidean), thì xác suất để chúng có cùng giá trị băm càng cao. Do đó, ta có thể so sánh các giá trị băm của chúng bằng **Hamming distance** (khoảng cách Hamming) — tức là số bit khác nhau giữa hai vector băm.

Trong phương pháp này, ta tạo ra một tập các vector ngẫu nhiên r_1, r_2, \dots, r_k trong cùng không gian với dữ liệu. Mỗi hàm băm $h_i(x)$ được định nghĩa dựa trên dấu của phép chiếu:

$$h_i(x) = \begin{cases} 1, & \text{nếu } (r_i \cdot x) \geq 0 \\ 0, & \text{nếu } (r_i \cdot x) < 0 \end{cases}$$

Với cách này, mỗi phần tử x được biểu diễn bởi một vector băm nhị phân độ dài k :

$$H(x) = [h_1(x), h_2(x), \dots, h_k(x)]$$

Khoảng cách giữa hai vector băm $H(x)$ và $H(y)$ được tính bằng khoảng cách Hamming:

$$d_H(H(x), H(y)) = \sum_{i=1}^k [h_i(x) \neq h_i(y)]$$

Giá trị d_H càng nhỏ thì x và y càng tương tự nhau.

Dưới đây là mô tả mã giả cho quy trình chèn và truy vấn trong Random Projection LSH:

Algorithm 2 Random Projection for Locality Sensitive Hashing

```
1: Khởi tạo: Sinh  $k$  vector ngẫu nhiên  $r_1, r_2, \dots, r_k$  trong  $\mathbb{R}^d$ 
2: Đầu vào: Tập dữ liệu  $\{x_1, x_2, \dots, x_n\}$ , số bảng băm  $L$ 
3: function HASH( $x$ )
4:   for  $i = 1$  to  $k$  do
5:      $h_i(x) \leftarrow 1$  nếu  $(r_i \cdot x) \geq 0$ , ngược lại 0
6:   end for
7:   return  $H(x) = [h_1(x), h_2(x), \dots, h_k(x)]$ 
8: end function
9: function INSERT( $x$ )
10:    $hash\_key \leftarrow Hash(x)$ 
11:   Lưu  $x$  vào bucket có khóa  $hash\_key$ 
12: end function
13: function QUERY( $q$ )
14:    $H_q \leftarrow Hash(q)$ 
15:    $Candidates \leftarrow$  các phần tử trong bucket có khóa gần  $H_q$  (theo Hamming distance
nhỏ nhất)
16:   return các phần tử trong  $Candidates$  có độ tương tự cao nhất
17: end function
```

Phương pháp Random Projection LSH mang lại hiệu quả cao khi xử lý dữ liệu có hàng triệu phần tử. Thay vì so sánh từng cặp điểm, ta chỉ cần so khớp các khóa băm nhị phân — giúp giảm đáng kể độ phức tạp tính toán. Ngoài ra, việc sử dụng khoảng cách Hamming cho phép tính toán nhanh chóng bằng các phép XOR hoặc bitwise operation trong phần cứng.

2.7 SimHash

SimHash là một biến thể của Locality Sensitive Hashing được Google giới thiệu để phát hiện văn bản hoặc tài liệu tương tự trong các tập dữ liệu rất lớn [3, 4]. Phương pháp này đặc biệt hiệu quả trong bài toán phát hiện trùng lặp nội dung web nhờ khả năng ánh xạ tài liệu thành một mã băm nhị phân ngắn, trong khi vẫn bảo toàn tương đối mức độ tương tự giữa chúng.

Ý tưởng chính của SimHash là sử dụng các vector trọng số đặc trưng để sinh ra một vector dấu (*sign vector*) biểu diễn toàn bộ tài liệu. Giả sử mỗi tài liệu D được biểu diễn bởi tập đặc trưng $\{f_1, f_2, \dots, f_n\}$ cùng trọng số tương ứng w_i . Ta sinh ngẫu nhiên các vector băm $r_1, r_2, \dots, r_k \in \mathbb{R}^d$. Với mỗi bit i của vector băm, ta cộng/trừ trọng số tùy theo dấu của phép chiếu:

$$v_i = \sum_{j=1}^n w_j \cdot \text{sign}(h_i(f_j))$$

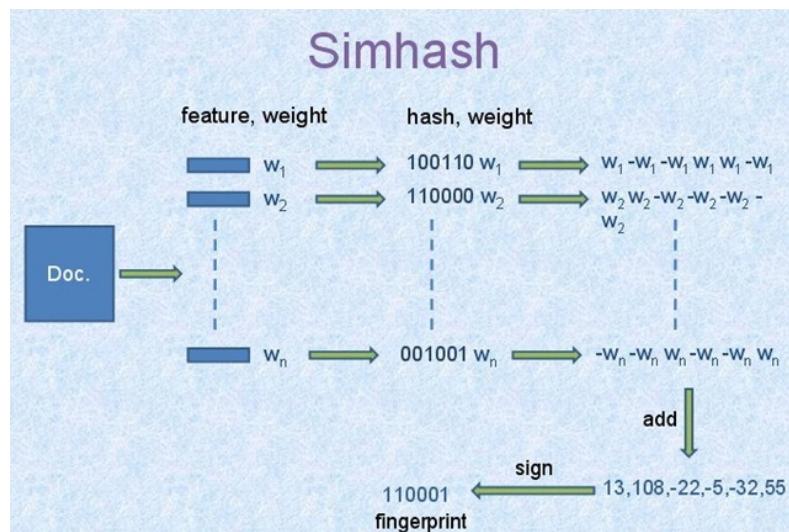
Trong đó:

$$\text{sign}(h_i(f_j)) = \begin{cases} +1, & \text{nếu } (r_i \cdot f_j) \geq 0 \\ -1, & \text{nếu } (r_i \cdot f_j) < 0 \end{cases}$$

Sau khi tính xong v_i cho tất cả các bit, ta tạo vector băm nhị phân:

$$H(D)_i = \begin{cases} 1, & \text{nếu } v_i \geq 0 \\ 0, & \text{nếu } v_i < 0 \end{cases}$$

Kết quả là mã SimHash $H(D)$ có độ dài k bit.



Hình 4: Quy trình sinh SimHash từ các đặc trưng của tài liệu.

Mức độ tương tự giữa hai tài liệu D_1 và D_2 được đo bằng khoảng cách Hamming giữa hai mã SimHash:

$$sim(D_1, D_2) = 1 - \frac{d_H(H(D_1), H(D_2))}{k}$$

Giá trị sim càng cao chứng tỏ hai tài liệu càng giống nhau.

Mã giả cho thuật toán SimHash được trình bày như sau:

Algorithm 3 SimHash

```
1: Đầu vào: Tập đặc trưng  $\{f_1, f_2, \dots, f_n\}$  và trọng số  $w_i$ 
2: Khởi tạo: Vector  $V[1 \dots k] \leftarrow 0$ 
3: for mỗi đặc trưng  $f_i$  do
4:    $h \leftarrow Hash(f_i)$                                 ▷ Tạo vector băm nhị phân độ dài k
5:   for  $j = 1$  to  $k$  do
6:     if  $h[j] = 1$  then
7:        $V[j] \leftarrow V[j] + w_i$ 
8:     else
9:        $V[j] \leftarrow V[j] - w_i$ 
10:    end if
11:   end for
12: end for
13: for  $j = 1$  to  $k$  do
14:    $H[j] \leftarrow 1$  nếu  $V[j] \geq 0$ , ngược lại 0
15: end for
16: return  $H$ 
```

SimHash đặc biệt phù hợp với các ứng dụng tìm kiếm và phát hiện trùng lặp trong hệ thống dữ liệu văn bản quy mô lớn, như Google Search hay công cụ thu thập dữ liệu web.

2.8 MinHash

MinHash là một kỹ thuật băm nhạy cảm với độ tương tự được đề xuất bởi Andrei Broder (1997) [2], nhằm ước lượng nhanh **độ tương tự Jaccard** giữa hai tập hợp lớn. Phương pháp này được sử dụng rộng rãi trong các hệ thống phát hiện trùng tài liệu, phân cụm văn bản, và tìm kiếm tương tự dựa trên tập từ khóa.

Với hai tập A và B , độ tương tự Jaccard được định nghĩa là:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



Tuy nhiên, việc tính trực tiếp $J(A, B)$ khi kích thước tập lớn là rất tốn kém. MinHash giải quyết vấn đề này bằng cách định nghĩa một hàm băm đặc biệt sao cho:

$$P[h(A) = h(B)] = J(A, B)$$

Nghĩa là xác suất để hai tập có cùng giá trị băm bằng chính độ tương tự Jaccard của chúng.

Giả sử có tập phần tử $U = \{e_1, e_2, \dots, e_n\}$ và một hàm băm ngẫu nhiên $h : U \rightarrow \mathbb{N}$. Với mỗi tập $S \subseteq U$, giá trị MinHash của S được định nghĩa là phần tử có giá trị băm nhỏ nhất:

$$\text{MinHash}(S) = \min_{e \in S} h(e)$$

Để giảm sai số, ta sử dụng k hàm băm khác nhau và biểu diễn mỗi tập bằng vector:

$$H(S) = [h_1^{\min}(S), h_2^{\min}(S), \dots, h_k^{\min}(S)]$$

Mức độ tương tự giữa hai tập A và B được tính bằng tỉ lệ các giá trị băm trùng nhau:

$$\text{sim}(A, B) = \frac{1}{k} \sum_{i=1}^k [h_i^{\min}(A) = h_i^{\min}(B)]$$

Giá trị này xấp xỉ độ tương tự Jaccard thực tế.

Mã giả cho thuật toán MinHash như sau:

Algorithm 4 MinHash

```
1: Đầu vào: Tập  $S$ , số hàm băm  $k$ 
2: Khởi tạo:  $Sig[1 \dots k] \leftarrow +\infty$ 
3: for mỗi phần tử  $e \in S$  do
4:   for  $i = 1$  to  $k$  do
5:      $hash\_val \leftarrow h_i(e)$ 
6:     if  $hash\_val < Sig[i]$  then
7:        $Sig[i] \leftarrow hash\_val$ 
8:     end if
9:   end for
10: end for
11: return  $Sig$ 
```

Nhờ cơ chế đơn giản và hiệu quả, MinHash là nền tảng cho nhiều phương pháp LSH khác, đặc biệt trong xử lý văn bản và phân tích dữ liệu rời rạc.



2.9 Hàm băm MurmurHash3

MurmurHash3 là một hàm băm phi mật mã được phát triển bởi Austin Appleby vào năm 2008 [5]. Hàm này được thiết kế nhằm tối ưu tốc độ và hiệu năng trong các ứng dụng cần phân phối giá trị băm đều, nhưng không yêu cầu tính bảo mật cao như trong mã hoá.

Điểm nổi bật của MurmurHash3 nằm ở cơ chế trộn bit hiệu quả. Quá trình băm gồm nhiều phép toán trên bit như **XOR**, **shift** và **multiply** với các hằng số lớn, giúp tạo ra phân phối ngẫu nhiên tốt giữa các giá trị đầu ra. Điều này đảm bảo rằng hai đầu vào tương tự nhau nhưng khác biệt dù chỉ một bit vẫn cho ra kết quả băm rất khác biệt (tính phân tán cao).

MurmurHash3 có nhiều biến thể, phổ biến nhất là:

- **MurmurHash3_x86_32**: cho kết quả 32-bit, thường dùng trong ứng dụng nhẹ.
- **MurmurHash3_x64_128**: cho kết quả 128-bit, phù hợp với xử lý dữ liệu lớn.

Công thức tổng quát cho một vòng lặp xử lý dữ liệu của MurmurHash3 (32-bit) có thể biểu diễn như sau:

$$k = k \times c_1; \quad k = (k \ll 15) \text{ hoặc } (k \gg 17); \quad k = k \times c_2;$$

$$h = h \oplus k; \quad h = (h \ll 13) \text{ hoặc } (h \gg 19); \quad h = h \times 5 + 0xe6546b64$$

trong đó c_1, c_2 là các hằng số ngẫu nhiên được chọn nhằm đảm bảo tính phân phối đều của giá trị băm.

Ưu điểm chính của MurmurHash3:

- **Tốc độ cao**: hoạt động hiệu quả trên CPU hiện đại nhờ chỉ sử dụng phép toán bitwise và nhân.
- **Phân phối đều**: giảm thiểu va chạm giữa các giá trị băm.
- **Dễ cài đặt**: mã nguồn ngắn gọn, không phụ thuộc nền tảng.

Nhờ các ưu điểm này, MurmurHash3 thường được sử dụng trong các hệ thống như Apache Hadoop, Cassandra, và Elasticsearch để phân phối khóa dữ liệu, cũng như trong việc sinh chỉ số băm cho Bloom Filter hoặc SimHash. Trong bài báo cáo này, MurmurHash3 sẽ được dùng để tính các vector băm của tập đặc trưng trong kỹ thuật SimHash.



2.10 Phương pháp TF-IDF

Hans Peter Luhn (1958) được coi là “cha đẻ của lĩnh vực Information Retrieval” và là tác giả của bài báo “*The Automatic Creation of Literature Abstracts – 1958*” [6]. Phương pháp TF-IDF (Term Frequency – Inverse Document Frequency) ra đời dựa trên ý tưởng đánh giá tầm quan trọng của một thành phần (từ khóa, đặc trưng) trong một tập hợp dữ liệu thông qua tần suất xuất hiện của nó, đồng thời giảm trọng số của các thành phần phổ biến nhưng ít mang tính phân biệt.

Cốt lõi của TF-IDF là sự kết hợp giữa hai yếu tố: *tần suất xuất hiện cục bộ* (TF) và *tần suất nghịch đảo toàn cục* (IDF). Ý tưởng này có thể được mở rộng để áp dụng không chỉ cho văn bản, mà còn cho các dạng dữ liệu khác như đặc trưng ảnh, âm thanh hoặc tín hiệu, nhằm đánh giá mức độ “quan trọng” của từng đặc trưng trong tập mẫu huấn luyện.

Trong phạm vi đề tài này, kỹ thuật TF-IDF được sử dụng như **một biến thể để tính trọng số cho các vector đặc trưng hình ảnh**, thay vì cho các từ trong văn bản. Mỗi đặc trưng trích xuất từ ảnh (ví dụ: đặc trưng màu sắc, kết cấu, hoặc mô tả cục bộ) được xem như một “từ khóa”, và tập ảnh tương ứng được xem như “tài liệu”. Theo đó:

- **TF (Term Frequency)** biểu thị mức độ xuất hiện hoặc cường độ của đặc trưng trong một ảnh cụ thể.
- **IDF (Inverse Document Frequency)** biểu thị mức độ phân biệt của đặc trưng đó trong toàn bộ tập ảnh — những đặc trưng phổ biến trong nhiều ảnh sẽ có trọng số thấp, trong khi đặc trưng chỉ xuất hiện ở một nhóm ảnh nhỏ (mang tính phân biệt cao) sẽ có trọng số lớn hơn.

Giá trị trọng số của một đặc trưng được tính theo công thức tương tự:

$$w_i = tf_i \times \log \left(\frac{N}{n_i} \right)$$

Trong đó:

- w_i : Trọng số của đặc trưng thứ i trong ảnh.
- tf_i : Mức độ xuất hiện (hoặc cường độ) của đặc trưng i trong ảnh hiện tại.
- N : Tổng số ảnh trong tập dữ liệu.
- n_i : Số lượng ảnh có chứa đặc trưng i .



Việc gán trọng số này giúp mô hình phân biệt rõ hơn giữa các đặc trưng “chung” (xuất hiện trong hầu hết ảnh) và các đặc trưng “riêng” (chỉ có trong một số nhóm ảnh cụ thể). Sau khi chuẩn hóa, các trọng số TF-IDF được kết hợp để tạo thành vector đặc trưng có trọng số cho từng ảnh.

Vector đặc trưng có trọng số này sẽ được sử dụng cho thuật toán **SimHash** trong khuôn khổ bài báo cáo này. Nhờ đó, SimHash không chỉ xem xét sự hiện diện của đặc trưng, mà còn phản ánh được mức độ quan trọng tương đối của chúng trong không gian ảnh. Cách tiếp cận này giúp giảm nhiễu từ các đặc trưng phổ biến, tăng độ chính xác trong việc sinh mã băm và so sánh độ tương đồng giữa các ảnh.

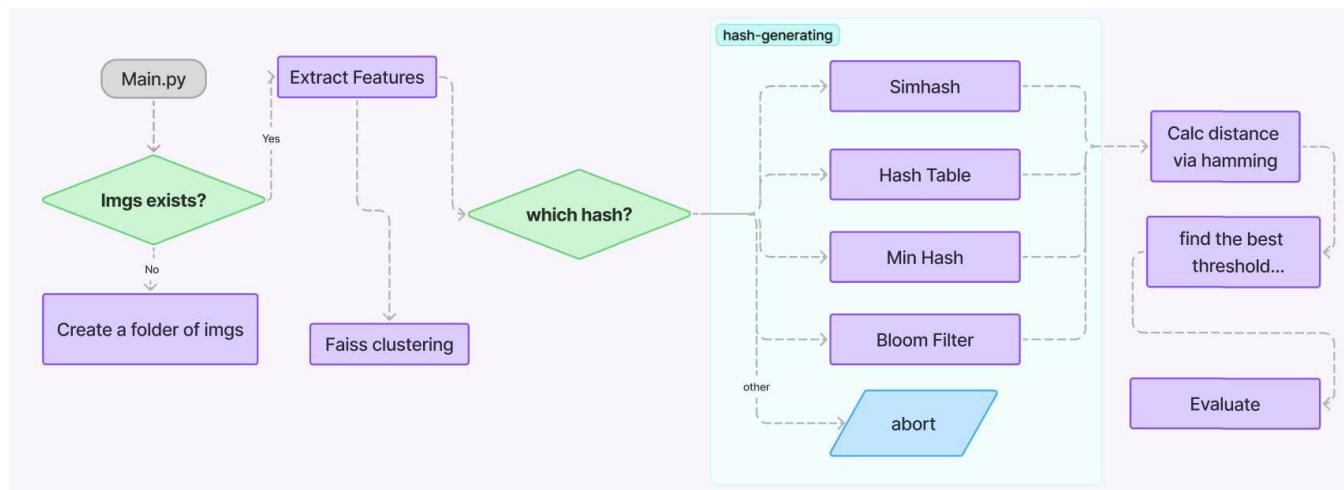
3 Xây dựng hệ thống phát hiện và phân loại ảnh trùng

3.1 Kiến trúc tổng thể của hệ thống

Hệ thống phát hiện và phân loại ảnh trùng nhau được thiết kế gồm bốn bước chính như sau:

- Tiền xử lý ảnh (Preprocessing):** Ảnh đầu vào được chuẩn hoá kích thước và phân phối điểm ảnh nhằm đảm bảo tính đồng nhất trước khi đưa vào mô hình học sâu.
- Trích xuất đặc trưng bằng ResNet-50 [7]:** Mô hình ResNet-50 được sử dụng ở dạng đã loại bỏ lớp Fully Connected cuối cùng (FC layer). Từ đó, mỗi ảnh được biểu diễn dưới dạng một vector đặc trưng có kích thước 2048 chiều.
- Băm đặc trưng:** Các vector đặc trưng được ánh xạ thành các mã băm (biểu diễn bằng các dãy bits).
- Nhóm và phát hiện ảnh trùng nhau:** Các ảnh sẽ được so sánh dựa trên khoảng cách Hamming giữa các mã băm tương ứng. Khi khoảng cách nhỏ hơn một ngưỡng đặt trước, hệ thống xem hai ảnh là trùng hoặc tương tự.

Cuối cùng, hiệu quả hệ thống được đánh giá bằng các chỉ số như ...



Hình 5: Quy trình nhóm ảnh và đánh giá.

3.2 Tiền xử lý ảnh

Trong giai đoạn tiền xử lý, mỗi ảnh đầu vào được chuẩn hoá về định dạng và kích thước để đảm bảo tính đồng nhất trước khi trích xuất đặc trưng. Các bước cụ thể như sau:

- **Đọc và chuyển đổi ảnh:** Mỗi tệp ảnh được đọc dưới dạng RGB. Đối với các ảnh không có đủ kênh màu, hệ thống chuyển sang dạng xám (grayscale) sau đó chuyển ngược lại về RGB nhằm đảm bảo đầu vào luôn có 3 kênh.
- **Chuẩn hóa kích thước:** Mọi ảnh được resize về kích thước 224×224 , tương thích với đầu vào của mô hình ResNet50.
- **Chuẩn hóa giá trị điểm ảnh theo ImageNet:** Mọi ảnh được chuẩn hoá theo bộ thống kê chuẩn của ImageNet:
$$\text{mean} = (0.485, 0.456, 0.406), \quad \text{std} = (0.229, 0.224, 0.225)$$

Nhằm đưa dữ liệu vào cùng phân phối mà ResNet đã được huấn luyện trước.

Bên cạnh việc chuẩn hoá, hệ thống còn áp dụng các phép biến đổi tăng cường (data augmentation) trên GPU thông qua thư viện Kornia. Tập các ảnh augment này được gom lại thành một batch và xử lý đồng thời trên GPU, sau đó dùng để tính vector đặc trưng trung bình cho từng ảnh.

3.3 Trích xuất đặc trưng

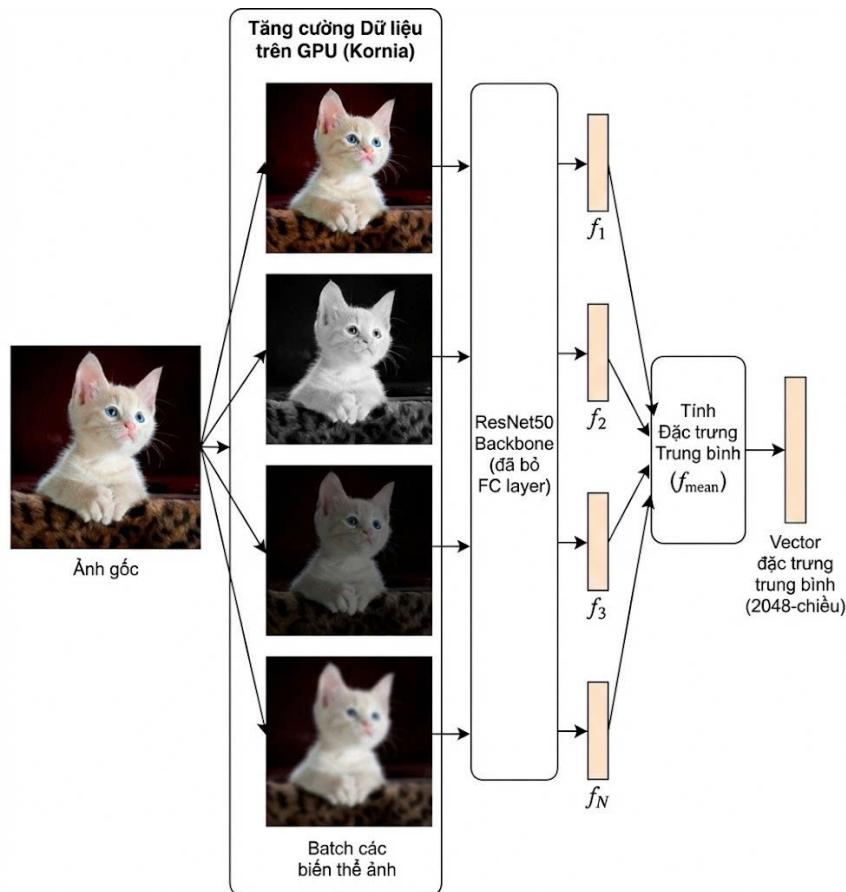
Việc trích xuất đặc trưng được thực hiện bằng mô hình ResNet50 đã được huấn luyện trước trên ImageNet. Để lấy đặc trưng mức sâu, lớp Fully Connected cuối cùng được loại bỏ, giữ lại phần backbone của mạng:

$$\text{ResNet50}_{\text{backbone}} = \text{ResNet50}_{\text{pretrained}} \setminus \text{FC layer}$$

Đầu ra của mạng sau khi loại bỏ lớp FC là một tensor kích thước 2048-chiều, biểu diễn đặc trưng của ảnh.

- Mỗi ảnh sau khi áp dụng N phép biến đổi augment được đưa qua backbone ResNet50.
- Từ mỗi biến thể, ta thu được một vector đặc trưng $f_i \in \mathbb{R}^{2048}$.
- Đặc trưng cuối cùng của ảnh được tính bằng trung bình:

$$f_{\text{mean}} = \frac{1}{N} \sum_{i=1}^N f_i$$



Hình 6: Tăng cường ảnh bằng thư viện Kornia

Phương pháp này giúp đặc trưng trở nên ổn định hơn trước các biến đổi hình học và nhiễu, từ đó cải thiện độ bền vững của hệ thống trong bài toán phát hiện ảnh trùng hoặc tương tự nhau.

Các vector đặc trưng cuối cùng được lưu lại dưới dạng mảng NumPy để phục vụ cho giai đoạn băm và phân cụm trong các bước tiếp theo.

3.4 Hashing và ánh xạ đặc trưng

Ở phần cơ sở lý thuyết của bài báo cáo đã trình bày nguyên lý của các phương pháp băm dành cho dữ liệu vectơ: Locality-Sensitive Hashing, BloomFilter, Min-Hash và SimHash. Trong phần này, bài báo cáo tập trung mô tả cách hệ thống hiện thực hóa các kỹ thuật trên bằng C++ và tích hợp vào Python thông qua Pybind11.

Sau khi trích xuất đặc trưng từ ResNet (mục 3.3), mỗi vectơ đặc trưng có kích thước $d = 2048$. Để tăng tốc độ truy vấn ảnh tương tự, hệ thống sử dụng các hàm băm nhạy tương đồng nhằm ánh xạ vectơ gốc sang không gian bit ngắn hơn nhưng vẫn bảo toàn quan hệ giống nhau.

3.4.1 Sinh siêu phẳng ngẫu nhiên (Random Hyperplanes)

Các thuật toán LSH dạng “random hyperplane hashing” đều bắt đầu từ việc sinh ra một tập các siêu phẳng ngẫu nhiên trong không gian \mathbb{R}^d . Mỗi siêu phẳng được biểu diễn bằng một vectơ pháp tuyến $\mathbf{w}_i \sim \mathcal{N}(0, 1)^d$. Trong hiện thực C++, các siêu phẳng được chuẩn hoá:

$$\mathbf{w}_i \leftarrow \frac{\mathbf{w}_i}{\|\mathbf{w}_i\|}$$

Điều này đảm bảo tính ổn định số và giảm sai lệch khi dấu của tích vô hướng được sử dụng để tạo bit băm.

3.4.2 HashTable–LSH

Lớp HashTable dùng toàn bộ P siêu phẳng để tạo một chuỗi bit duy nhất:

$$\text{hash}(\mathbf{x}) = \sum_{i=0}^{P-1} [(\mathbf{x} \cdot \mathbf{w}_i > 0) \ll i]$$

Đây là phiên bản LSH cổ điển sử dụng random hyperplane hashing. Mỗi ảnh được ảnh xạ thành một chuỗi bits có độ dài P bits tùy vào cấu hình được chọn.

3.4.3 BloomFilter–LSH

Lớp BloomFilter được xử lý như một biến thể đề phù hợp với bài toán đề ra được lấy ý tưởng từ các bài báo [8, 9, 10], tương tự với Lớp HashTable sinh ra chuỗi P nhưng tách thành k nhóm, mỗi nhóm gồm $\frac{P}{k}$ siêu phẳng (với P là tổng số siêu phẳng). Với một vectơ đặc trưng \mathbf{x} , hàm băm được tính:

$$b_i = \begin{cases} 1 & \text{nếu } \mathbf{x} \cdot \mathbf{w}_i > 0 \\ 0 & \text{ngược lại} \end{cases}$$

Mỗi nhóm trả về một số chuỗi bits có độ dài là $\frac{P}{k}$. Kết quả cuối cùng là vectơ kích thước k :

$$\text{hash}(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_k(\mathbf{x}))$$

Thiết kế này giúp tăng độ chính xác của việc nhóm các ảnh trùng. Do việc so sánh các ảnh bằng khoảng cách Hamming nên việc nhóm ảnh bằng BloomFilter đảm bảo các ảnh được coi là ảnh trùng có ít nhất $\frac{P}{k}$ bits liên tiếp giống nhau. Từ đó cho ra kết quả tốt hơn so với lớp HashTable

3.4.4 MinHash dựa trên giá trị tối thiểu theo siêu phẳng

Hiện thực MinHash trong hệ thống có điều chỉnh so với MinHash truyền thống. Với mỗi chiều j , hệ thống xét:

$$m_j = \min_{p \in \{1 \dots P\}} (x_j \cdot w_{p,j})$$

Sau đó chuẩn hoá theo một trong hai cách (tùy chọn đầu vào):

- ngưỡng theo trung vị: $m_j \geq \text{median}_j$ thì $b_j = 1$ và ngược lại
- hoặc chuẩn hoá Z-score: $z_j = (m_j - \mu_j)/\sigma_j > 0$ thì $b_j = 1$ và ngược lại

Kết quả là một chữ ký bit có độ dài bằng số chiều d :

$$\text{signature}(\mathbf{x}) \in \{0, 1\}^d$$

3.4.5 SimHash dựa trên MurmurHash128 và trọng số TF-IDF

Lớp SimHash được xây dựng dựa trên phương pháp băm nhạy cảm với độ tương tự, kế thừa kỹ thuật áp dụng cho truy vấn ảnh quy mô lớn từ nghiên cứu của Guo và cộng sự [11].

- sử dụng MurmurHash3_x64_128 để băm từng chiều x_j thành ϕ_j
- áp dụng trọng số TF-IDF:

$$w_j = x_j \cdot \text{idf}_j$$

Tổng hợp theo công thức:

$$V_k = \sum_{j=1}^d \phi_{j,k} \cdot w_j$$

Bit thứ k của SimHash là:

$$h_k = \begin{cases} 1 & V_k \geq 0 \\ 0 & V_k < 0 \end{cases}$$

Kết quả cuối cùng là một giá trị băm 64 hoặc 128 bit (tùy chọn đầu vào).

3.4.6 Tích hợp Python thông qua Pybind11

Toàn bộ các mô-đun băm (BloomFilter, HashTable, MinHash, SimHash) được đóng gói thành thư viện Python bằng Pybind11. Điều này cho phép pipeline tiền

xử lý đặc trưng viết bằng Python (ResNet50, Kornia, NumPy) có thể gọi trực tiếp các thuật toán băm tốc độ cao viết bằng C++.

Ví dụ mô-đun binding:

```
PYBIND11_MODULE(hashcore, m) {
    py::class_<BloomFilter>(m, "BloomFilter")
        .def(py::init<size_t,size_t,size_t>())
        .def("hashFunction", &BloomFilter::hashFunction);

    py::class_<HashTable>(m, "HashTable")
        .def(py::init<size_t,size_t>())
        .def("hashFunction", &HashTable::hashFunction);

    py::class_<MinHash>(m, "MinHash")
        .def(py::init<size_t,size_t>())
        .def("computeSignatures", &MinHash::computeSignatures)
        .def("hashFunction", &MinHash::hashFunction);

    py::class_<SimHash>(m, "SimHash")
        .def(py::init<size_t>())
        .def("IDF", &SimHash::IDF)
        .def("hashFunction", &SimHash::hashFunction);
}
```

Nhờ đó, pipeline Python có thể xử lý ảnh trên GPU, còn phần băm tương đồng được xử lý bằng C++.

3.4.7 Gom cụm dựa trên mã băm

Đối với các phương pháp (MinHash, SimHash, LSH), việc gom cụm được thực hiện dựa trên cấu trúc bảng băm và khoảng cách Hamming. Quy trình bao gồm hai bước:

1. **Phân loại vào Bucket:** Tất cả các ảnh có cùng giá trị mã băm (hash signature) sẽ được đưa vào cùng một danh sách (bucket). Ánh xạ này được thực hiện thông qua cấu trúc dữ liệu từ điển (dictionary):

$$\text{Bucket}(h) = \{\text{img}_i \mid \text{hash}(\text{img}_i) = h\}$$

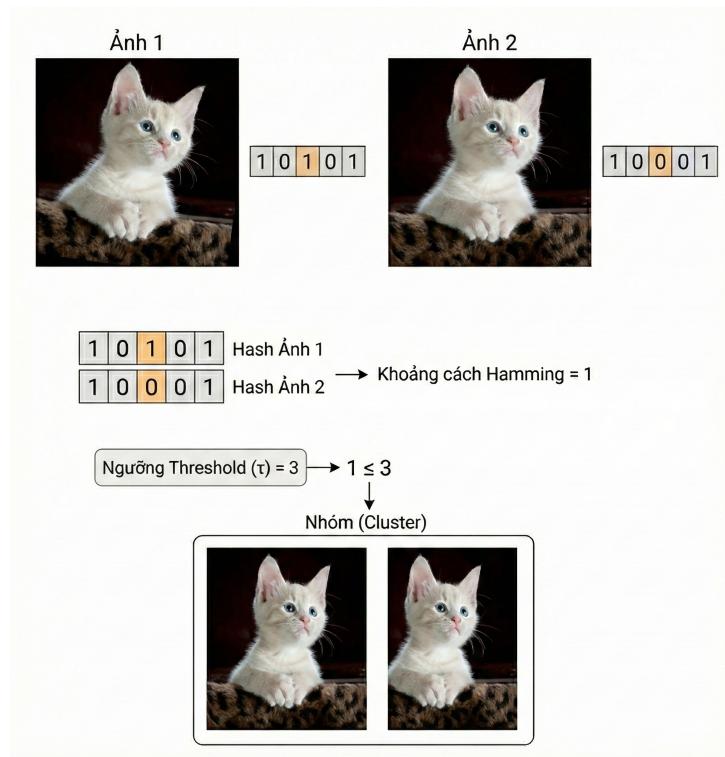
2. **Gộp nhóm dựa trên bán kính Hamming:** Do nhiều trong ảnh hoặc các

bien đổi nhỏ, hai ảnh tương tự nhau có thể lệch nhau vài bit. Hệ thống sử dụng thuật toán gộp nhóm (merging buckets) như sau:

- Duyệt qua các bucket hiện có.
- Tính khoảng cách Hamming giữa các mã băm đại diện của các bucket:

$$D_H(h_1, h_2) = \text{popcount}(h_1 \oplus h_2)$$

- Nếu $D_H \leq \tau$ (với τ là ngưỡng sai số cho phép, ví dụ $\tau = 2$), hai bucket sẽ được gộp lại thành một cụm duy nhất.



Hình 7: So sánh khoảng cách hamming giữa các ảnh

Riêng đối với **BloomFilter**, do đặc thù sử dụng k hàm băm, tiêu chí gom cụm được điều chỉnh dựa trên tổng khoảng cách Hamming của các thành phần con, đảm bảo độ chính xác cao hơn khi xử lý dữ liệu phân tán.

3.4.8 Gom cụm dựa trên đồ thị và FAISS

Để tạo cơ sở so sánh hoặc khi cần độ chính xác tối đa mà không quan tâm nhiều đến tốc độ, hệ thống tích hợp thư viện FAISS để tìm kiếm lồng giềng gần nhất trên không gian vector thực $d = 2048$.

Quy trình gom cụm được mô hình hóa dưới dạng bài toán tìm các thành phần liên thông trên đồ thị vô hướng:

- **Dựng đồ thị:** Mỗi ảnh là một đỉnh V_i . Cạnh E_{ij} tồn tại nếu độ tương đồng Cosine giữa hai vector đặc trưng lớn hơn ngưỡng θ (ví dụ $\theta = 0.9$):

$$\cos(\mathbf{f}_i, \mathbf{f}_j) = \frac{\mathbf{f}_i \cdot \mathbf{f}_j}{\|\mathbf{f}_i\| \|\mathbf{f}_j\|} \geq \theta$$

- **Duyệt đồ thị:** Hệ thống sử dụng thuật toán Tìm kiếm theo chiều rộng (BFS) để gom tất cả các đỉnh có đường đi tới nhau vào cùng một cụm. Điều này cho phép phát hiện các chuỗi ảnh trùng nhau ví dụ: A trùng B, B trùng C \rightarrow A, B, C cùng một nhóm.

3.5 Tối ưu hóa ngưỡng tự động

Trong quá trình triển khai hệ thống thành ứng dụng web thực tế, nhóm nghiên cứu nhận thấy việc sử dụng một ngưỡng cố định (fixed threshold) cho khoảng cách Hamming không mang lại hiệu quả đồng nhất trên mọi tập dữ liệu. Các bộ ảnh khác nhau có phân phối độ tương đồng khác nhau, dẫn đến việc ngưỡng cố định có thể gây ra hiện tượng:

- **Under-clustering:** Ngưỡng quá thấp khiến các ảnh biến đổi nhẹ không được gom vào nhóm.
- **Over-clustering:** Ngưỡng quá cao khiến các ảnh khác nhau bị gom nhầm thành một nhóm.

Để giải quyết vấn đề này mà không yêu cầu người dùng cuối phải có kiến thức chuyên môn để tự chỉnh tham số, nhóm đã phát triển thuật toán **Phân tích Histogram khoảng cách (Distance Histogram Analysis)** để tự động tìm ngưỡng tối ưu.

3.5.1 Cơ sở lý thuyết: Phân phối hai đỉnh

Giả thuyết đặt ra là tập hợp các khoảng cách giữa các cặp ảnh trong một dataset thường tuân theo phân phối hai đỉnh:

1. **Đỉnh thứ nhất (Duplicates Peak):** Tập trung ở giá trị khoảng cách thấp, đại diện cho các cặp ảnh trùng hoặc rất giống nhau.
2. **Đỉnh thứ hai (Non-duplicates Peak):** Tập trung ở giá trị cao, đại diện cho phần lớn các cặp ảnh không liên quan đến nhau.

Ngưỡng tối ưu (T_{opt}) chính là điểm cực tiểu cục bộ (thung lũng) nằm giữa hai đỉnh này.



3.5.2 Thuật toán tìm kiếm thung lũng (Valley Detection Algorithm)

Quy trình tìm ngưỡng tự động được thực hiện qua 4 bước, mô phỏng lại logic của đoạn mã nguồn đã cài đặt:

Bước 1: Tính toán Histogram khoảng cách

Hệ thống tính khoảng cách Hamming $d(h_i, h_j)$ cho các cặp mã băm (có thể sử dụng lấy mẫu ngẫu nhiên nếu tập dữ liệu quá lớn) và lập biểu đồ tần suất $H(x)$.

Bước 2: Làm mượt bằng bộ lọc Gaussian (Gaussian Smoothing)

Do biểu đồ thực tế thường chứa nhiễu (răng cửa), việc tìm cực tiểu trực tiếp sẽ không chính xác. Ta áp dụng phép tích chập với nhân Gaussian để làm mượt:

$$H_{\text{smooth}}(x) = H(x) * G_\sigma(x)$$

Trong đó σ (sigma) là hệ số làm mượt (thực nghiệm chọn $\sigma = 1.5$). Điều này giúp loại bỏ các biến động nhỏ cục bộ mà vẫn giữ lại hình dáng tổng thể của phân phối.

Bước 3: Xác định cực tiểu cục bộ

Hệ thống tìm các điểm x thoả mãn điều kiện cực tiểu:

$$\frac{\partial H_{\text{smooth}}}{\partial x} = 0 \quad \text{và} \quad \frac{\partial^2 H_{\text{smooth}}}{\partial x^2} > 0$$

Trong mã nguồn, việc này được thực hiện bằng hàm `argrelextrema` so sánh giá trị với các láng giềng bậc 2.

Bước 4: Chọn ngưỡng hợp lệ

Ngưỡng tối ưu T_{opt} được chọn là cực tiểu đầu tiên xuất hiện sau đỉnh thứ nhất.

$$T_{opt} = \min\{x \in \text{LocalMinima} \mid x > \text{SafetyMargin}\}$$

Trong trường hợp phân phối không rõ ràng (không tìm thấy thung lũng), hệ thống sẽ fallback về ngưỡng an toàn dựa trên phân vị (percentile 10%).

3.6 Cơ chế tính điểm chất lượng ảnh

Trong mỗi cụm ảnh (cluster) bao gồm nhiều bản sao và biến thể, việc xác định bức ảnh tốt nhất để đề xuất người dùng giữ lại là một bài toán tối ưu hóa đa mục tiêu. Thay vì dựa trên cảm tính, hệ thống xây dựng một mô hình đánh giá định lượng dựa trên 7 chỉ số thị giác cốt lõi, đảm bảo bức ảnh được chọn không chỉ có độ phân giải cao nhất mà còn đạt chuẩn về thẩm mỹ và độ nguyên vẹn.

3.6.1 Thiết kế hàm mục tiêu

Chất lượng tổng thể (Q_{total}) của một bức ảnh được mô hình hóa dưới dạng tổng trọng số của các thành phần chất lượng thành phần:

$$Q_{total} = \sum_{i=1}^n w_i \cdot S_i$$

Trong đó:

- S_i : Điểm số chuẩn hóa (0-100) của chỉ số thị giác thứ i .
- w_i : Trọng số tầm quan trọng của chỉ số i ($\sum w_i = 1$).

Hệ thống trọng số (w_i) không được gán ngẫu nhiên mà dựa trên nghiên cứu về nhận thức thị giác người (Human Visual System - HVS), ưu tiên độ sắc nét và độ phân giải - hai yếu tố ảnh hưởng trực tiếp nhất đến trải nghiệm xem.

3.6.2 Các chỉ số đánh giá thành phần

a. Độ sắc nét (Sharpness - $w = 0.3$)

Đây là yếu tố quan trọng nhất. Một bức ảnh dù độ phân giải cao nhưng bị mờ nhòe do rung tay hoặc lẩy nét vẫn bị coi là kém chất lượng.

- *Phương pháp*: Sử dụng toán tử Laplace (Laplacian Variance) để đo mức độ biến thiên của cường độ sáng tại các cạnh.
- *Ý nghĩa*: Phương sai càng cao chứng tỏ ảnh càng nhiều chi tiết và cạnh sắc nét.

b. Độ phân giải (Resolution - $w = 0.2$)

Do lường lượng thông tin chi tiết mà ảnh chứa đựng.

- *Công thức*: Dựa trên tổng số điểm ảnh (MegaPixels - MP). Điểm bão hòa được đặt ở mức 12MP (chuẩn phổ biến của smartphone hiện nay), vượt qua ngưỡng này điểm số sẽ tăng không đáng kể theo quy luật hiệu suất giảm dần.

c. Độ giàu màu sắc (Color Richness - $w = 0.15$)

Dùng để phân biệt ảnh màu sống động với ảnh đen trắng hoặc ảnh bị mất màu (washed-out).

- *Cơ sở:* Đo độ lệch chuẩn giữa các kênh màu (R, G, B). Ảnh xám có độ lệch giữa các kênh xấp xỉ 0, trong khi ảnh có màu sắc phong phú sẽ có độ biến thiên cao giữa các kênh này.

d. Độ phơi sáng (Exposure - $w = 0.15$)

Dánh giá mức độ cân bằng ánh sáng, mục tiêu là phạt nặng các ảnh quá tối (underexposed) hoặc quá cháy sáng (overexposed).

- *Mô hình:* Tính toán khoảng cách từ độ sáng trung bình của ảnh tới giá trị lý tưởng (≈ 127 trong không gian màu 8-bit).

e. Độ tương phản (Contrast - $w = 0.1$)

Đo lường dải động (dynamic range) của ảnh, thể hiện sự khác biệt giữa vùng sáng nhất và tối nhất.

- *Phương pháp:* Dựa trên độ lệch chuẩn của histogram độ sáng. Ảnh có độ tương phản tốt giúp chủ thể tách bạch rõ ràng, không bị bệt màu vào nền.

f. Độ nguyên vẹn cạnh (Edge Integrity - $w = 0.05$)

Một chỉ số đặc biệt được thiết kế để phát hiện các ảnh đã bị xoay hoặc cắt xén (crop) cẩu thả, để lại các vùng đen vô nghĩa (padding) ở 4 góc hoặc viền ảnh.

- *Cơ chế:* Kiểm tra giá trị pixel trung bình tại 4 góc ảnh và các cạnh biên. Nếu phát hiện vùng đen tuyệt đối hoặc vùng màu đồng nhất bất thường, điểm số sẽ bị trừ nặng.

g. Độ nhiễu (Noise Level - $w = 0.05$)

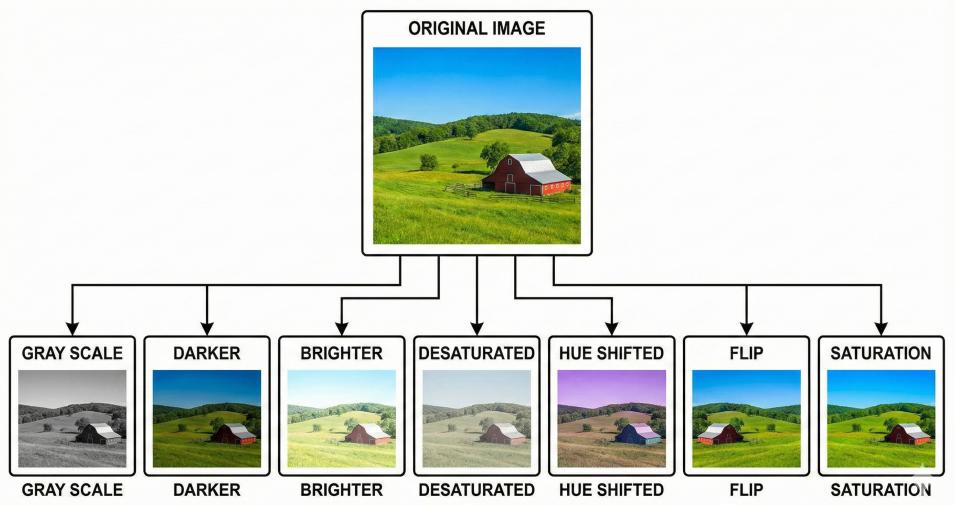
Dánh giá mức độ hạt (grain) hoặc nhiễu kỹ thuật số, thường gặp khi chụp thiếu sáng buộc cảm biến phải đẩy ISO lên cao.

- *Phương pháp:* Ước lượng mức độ nhiễu qua các vùng phẳng của ảnh (ít chi tiết). Ảnh càng mịn màng ở các vùng này thì điểm chất lượng càng cao.

4 Thực nghiệm và đánh giá kết quả

4.1 Giới thiệu sơ về dataset

Datasets dùng để đánh giá là các tập ảnh với nhiều biến thể được tạo ra từ ảnh gốc như: ảnh xám, ảnh làm tối, ảnh làm sáng hơn, ảnh làm mờ, ảnh xoay, ảnh có nhiễu, ...



Hình 8: Ví dụ một vài biến thể của ảnh trong các datasets

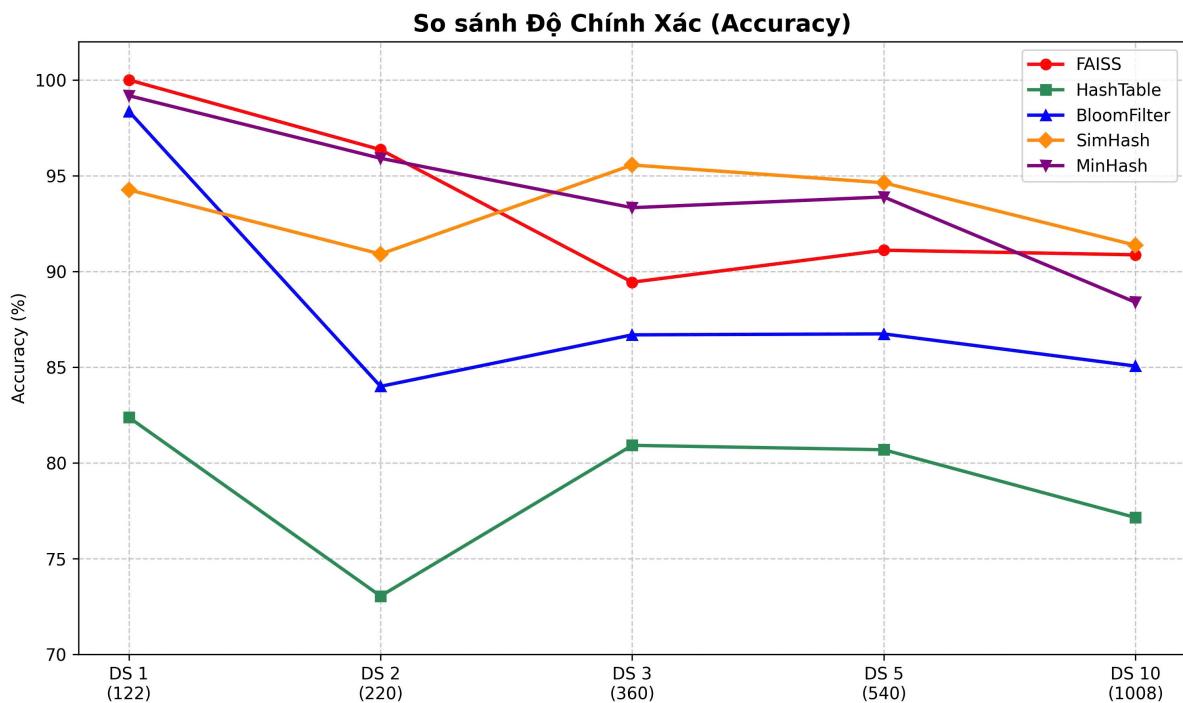
Lưu ý: Quy cách đặt tên file tuân theo định dạng `{class}_{id}.{ext}` để hệ thống có thể tự động đánh giá chính xác lớp của ảnh.

4.2 Đánh giá độ chính xác

Độ chính xác được tính dựa trên tỷ lệ số ảnh được gom nhóm đúng vào lớp chủ sở hữu so với tổng số ảnh.

Phương pháp	DS1 (122)	DS2 (220)	DS3 (360)	DS5 (540)	DS10 (1008)
FAISS	100.00	96.36	89.44	91.11	90.87
HashTable	82.38	73.05	80.92	80.69	77.15
BloomFilter	98.36	84.00	86.69	86.74	85.06
SimHash	94.26	90.91	95.56	94.63	91.37
MinHash	99.18	95.91	93.33	93.89	88.39

Bảng 1: Bảng so sánh Accuracy (%) giữa các phương pháp



Hình 9: Biểu đồ thể hiện sự thay đổi Accuracy theo kích thước dữ liệu

Nhận xét:

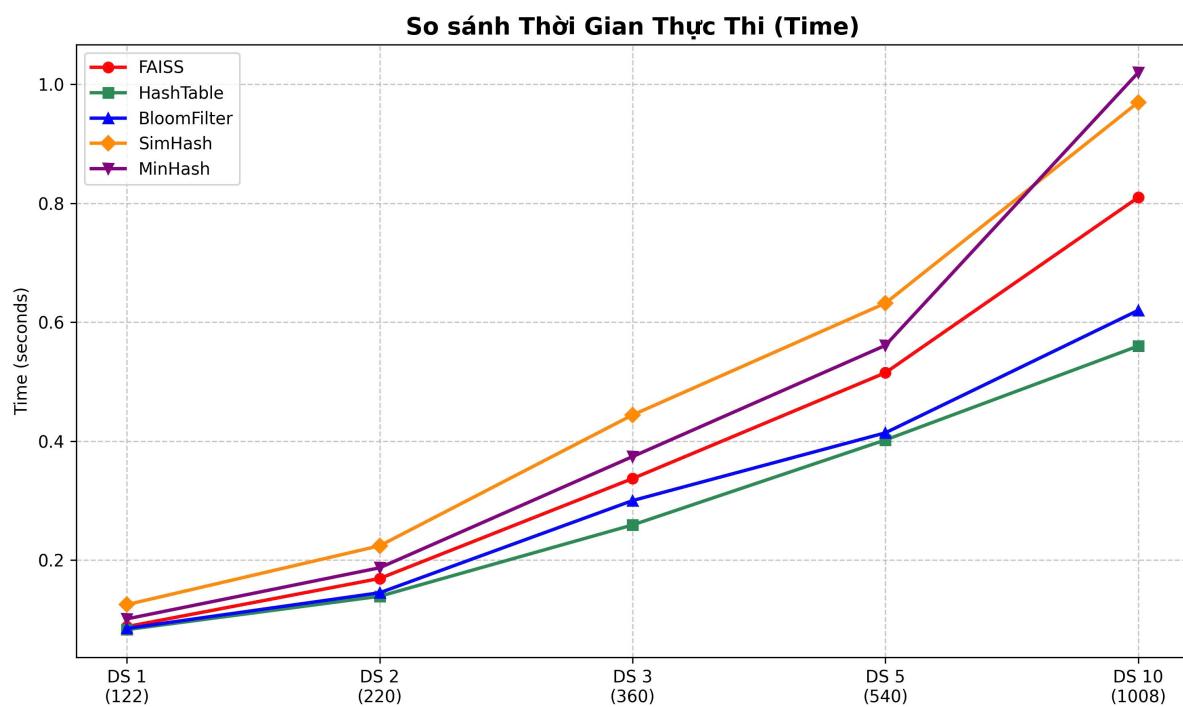
- **FAISS** đạt độ chính xác tuyệt đối (100%) trên tập dữ liệu nhỏ nhưng có xu hướng giảm nhẹ khi số lượng ảnh tăng lên, tuy nhiên vẫn duy trì ở mức cao ($>90\%$).
- **SimHash** thể hiện sự ổn định tốt nhất trong các thuật toán băm, duy trì accuracy trên 90% ở mọi kích thước tập dữ liệu và đạt đỉnh cao nhất ở tập DS 3 (95.56%).
- **HashTable** cho kết quả thấp nhất, dao động không ổn định (từ 73% đến 82%), cho thấy hạn chế của phương pháp băm truyền thống với các biến thể ảnh.
- **MinHash** và **BloomFilter** hoạt động rất tốt với tập dữ liệu nhỏ nhưng hiệu suất giảm rõ rệt khi xử lý tập dữ liệu lớn nhất (DS 10).

4.3 Đánh giá thời gian xử lý

Thời gian xử lý bao gồm thời gian trích xuất đặc trưng, băm và gom nhóm dữ liệu.

Phương pháp	DS1 (122)	DS2 (220)	DS3 (360)	DS5 (540)	DS10 (1008)
FAISS	0.088	0.169	0.337	0.515	0.81
HashTable	0.083	0.139	0.259	0.402	0.56
BloomFilter	0.085	0.145	0.300	0.414	0.62
SimHash	0.125	0.224	0.444	0.632	0.97
MinHash	0.101	0.187	0.374	0.561	1.02

Bảng 2: Bảng so sánh thời gian xử lý (giây)



Hình 10: Biểu đồ so sánh tốc độ xử lý giữa các thuật toán

Nhận xét:

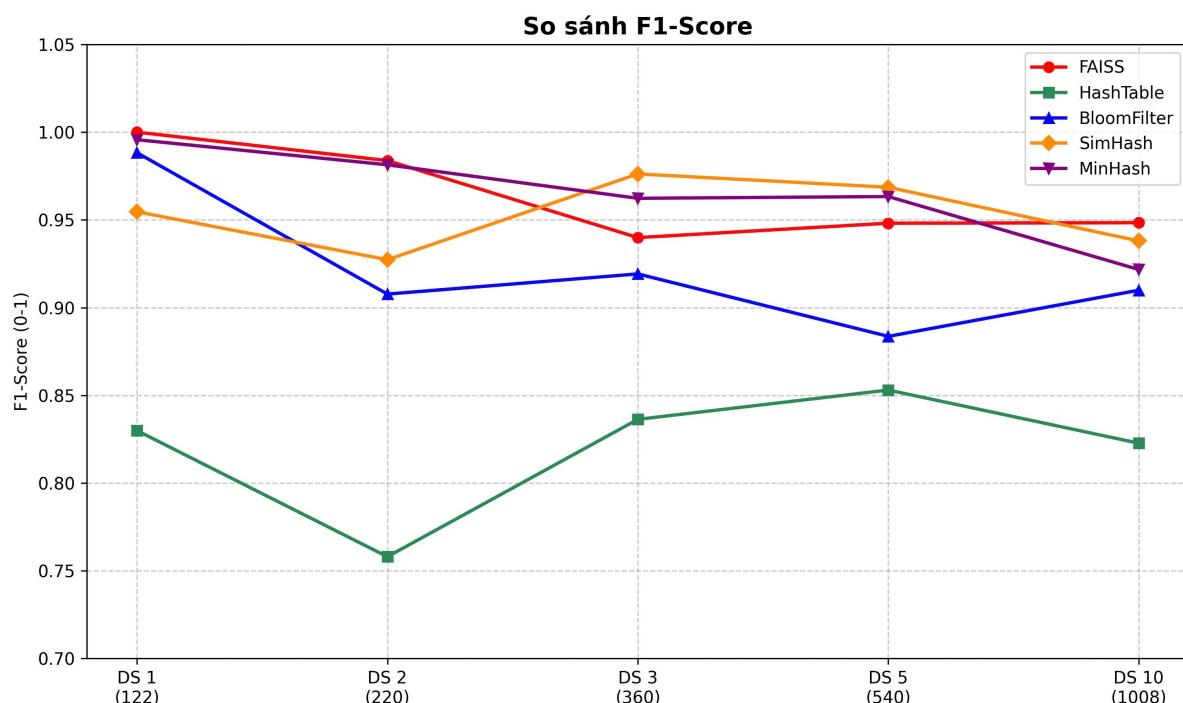
- HashTable là phương pháp có tốc độ nhanh nhất trên mọi tập dữ liệu (chỉ mất 0.56s cho hơn 1000 ảnh) nhờ độ phức tạp tính toán thấp.
- FAISS và BloomFilter có hiệu năng tốc độ rất tốt, đứng thứ hai sau HashTable. Đặc biệt FAISS dù xử lý vector phức tạp nhưng vẫn tối ưu thời gian rất tốt.
- MinHash và SimHash là hai phương pháp chậm nhất do khối lượng tính toán lớn hơn (tạo chữ ký min-wise hoặc tính cosine similarity), thời gian tăng tuyến tính rõ rệt theo số lượng ảnh.

4.4 Đánh giá chỉ số F1-Score

F1-Score là trung bình điều hòa giữa Precision và Recall, phản ánh độ tin cậy tổng thể của mô hình.

Phương pháp	DS1 (122)	DS2 (220)	DS3 (360)	DS5 (540)	DS10 (1008)
FAISS	1.0000	0.9838	0.9399	0.9481	0.9484
HashTable	0.8299	0.7580	0.8363	0.8530	0.8227
BloomFilter	0.9883	0.9077	0.9192	0.8836	0.9099
SimHash	0.9546	0.9273	0.9762	0.9686	0.9380
MinHash	0.9957	0.9814	0.9623	0.9633	0.9217

Bảng 3: Bảng so sánh F1-Score



Hình 11: Biểu đồ so sánh chất lượng phân loại (F1-Score)

Nhận xét:

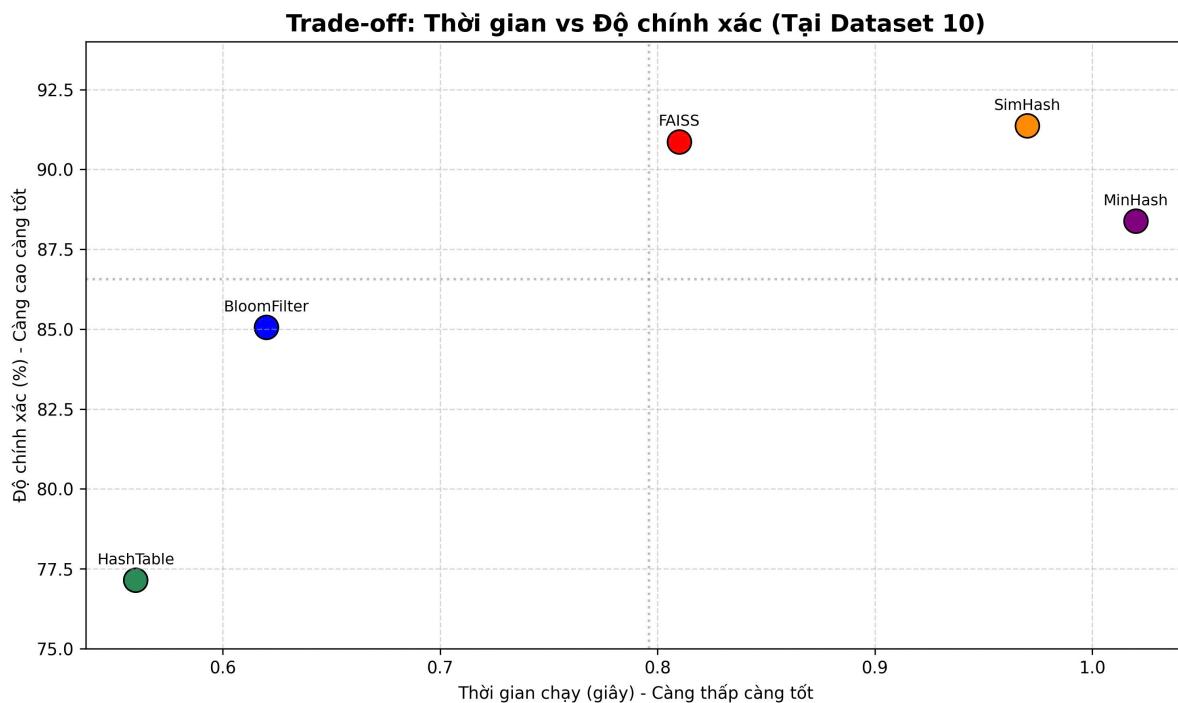
- **FAISS** và **SimHash** cho thấy chất lượng phân loại vượt trội với F1-Score luôn duy trì ở mức rất cao (> 0.93) ngay cả khi dữ liệu lớn.
- **MinHash** mặc dù có Accuracy giảm ở tập lớn, nhưng F1-Score vẫn giữ ở mức chấp nhận được (0.92), chứng tỏ thuật toán vẫn gom nhóm khá tốt các đặc trưng quan trọng.
- **HashTable** tiếp tục cho thấy điểm yếu về chất lượng phân loại với F1-Score thấp nhất (0.82).

4.5 Tổng hợp và kết luận

Dưới đây là bảng tổng hợp kết quả trên tập dữ liệu lớn nhất (DS10 - 1008 ảnh) để có cái nhìn toàn diện:

Phương pháp	Accuracy (%)	Time (s)	F1-Score
FAISS	90.87	0.81	0.9484
HashTable	77.15	0.56	0.8227
BloomFilter	85.06	0.62	0.9099
SimHash	91.37	0.97	0.9380
MinHash	88.39	1.02	0.9217

Bảng 4: Tổng hợp kết quả thực nghiệm trên tập DS10



Hình 12: Biểu đồ Trade-off giữa thời gian và độ chính xác (DS10)

Kết luận:

- Nếu ưu tiên **độ chính xác**: SimHash và FAISS là lựa chọn hàng đầu. FAISS cân bằng tốt hơn một chút về tốc độ.
- Nếu ưu tiên **tốc độ xử lý**: HashTable là nhanh nhất, nhưng phải đánh đổi bằng độ chính xác thấp. BloomFilter là phương án trung庸 hợp lý.
- Với bài toán phát hiện ảnh trùng lặp trong thực tế, FAISS hoặc SimHash được khuyến nghị sử dụng nhờ sự ổn định của chỉ số F1-Score và Accuracy.

5 Demo

5.1 Ý tưởng

Chúng ta đang sống trong kỷ nguyên của công nghệ số. Trong một chuyến đi chơi, ta có thể chụp hàng trăm bức ảnh, nhưng chỉ chọn ra được vài tấm ưng ý. Phần còn lại – những tấm bị rung, chụp liên tiếp, hay chỉ là bản sao – nằm im lìm trong ổ cứng, tạo nên một sự lãng phí khổng lồ.

Vấn đề lớn nhất không phải là *lưu trữ*, mà là *dọn dẹp*. Nhìn vào một thư mục chứa danh sách hàng ngàn tên file như `IMG_001.jpg`, `IMG_002.jpg`... là một trải nghiệm vô cùng tồi tệ. Người dùng không biết bắt đầu từ đâu, không thấy được bức tranh toàn cảnh, sợ xóa nhầm, chưa kể đến việc tốn rất nhiều thời gian.

Ý tưởng chủ đạo

Chúng tôi không muốn tạo ra đơn thuần một công cụ quản lý file. Chúng tôi muốn xây dựng một "**Universe Map**". Thay vì bắt người dùng đọc những danh sách, con số khô khan, chúng tôi cho họ **nhìn thấy** cấu trúc của sự bừa bộn đó, và dọn dẹp nó như đang chơi một trò chơi.

5.2 Deduplication

Vấn đề: Máy tính vốn dĩ hoạt động rất "máy móc". Với máy tính, một bức ảnh gốc và một bức ảnh bị chỉnh sáng nhẹ là hai tệp tin hoàn toàn khác nhau. Nhưng với mắt người, chúng là một. Nếu phần mềm chỉ tìm các file giống hệt nhau, nó sẽ bỏ sót hơn 90% các ảnh rác thực sự (near-duplicate).

Giải pháp Chúng tôi dạy cho hệ thống cách "so sánh" như con người. Thay vì so sánh từng điểm ảnh, hệ thống tập trung vào **đặc trưng** của ảnh đó.

- Nó hiểu rằng khi bạn chụp liên tiếp 10 tấm, dù góc máy lệch đi một chút, đó vẫn là cùng một khoảnh khắc.
- Nó tự động gom tất cả các bức ảnh có **đặc trưng** gần giống nhau lại thành một **nhóm (cluster)**.

→ *Kết quả: Người dùng không cần soi từng cặp ảnh. Họ chỉ cần duyệt qua các nhóm (cluster) và quyết định giữ lại bức ảnh tốt nhất.*

5.3 Visualization

Vấn đề:

Nếu hệ thống chỉ đơn thuần thông báo: "Bạn có 1000 bức ảnh trong đó có 900 bản sao", đối với người dùng, đó chỉ là những con số vô hồn. Họ biết mình có nhiều ảnh, nhưng họ không thể hình dung được **cấu trúc** thật sự cũng như sự tương đồng giữa các nhóm.

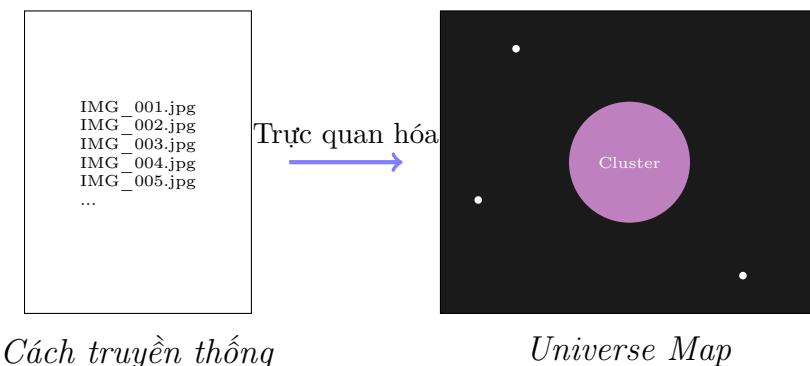
Thách thức lớn nhất đặt ra là: *Làm sao để nhồi nhét mỗi quan hệ phúc tạp của hàng vạn tấm ảnh vào một màn hình laptop chật hẹp mà người dùng vẫn hiểu được?*

Các phương pháp truyền thống đều thất bại trong việc mang lại cái nhìn tổng quan:

- **Danh sách** Hàng dọc các tên file IMG_0467.JPG, IMG_0592.JPG, ..., người dùng không nhìn thấy sự tương đồng hay khác biệt giữa các ảnh. Quan trọng hơn là không biết các ảnh này giống hay khác nhau **nếu thế nào**.
- **Biểu đồ 2D**: Khi cố gắng ép hàng nghìn điểm dữ liệu lên một mặt phẳng, chúng sẽ đè lên nhau gây ra hiện tượng chồng lấn (occlusion). Người dùng không phân biệt được đâu là một nhóm ảnh lớn, đâu là những điểm ảnh rời rạc.

→ Người dùng cần một "**vũ trụ**" thực sự để định vị và kiểm soát dữ liệu của mình, chứ không chỉ là một bảng báo cáo thống kê.

Giải pháp: Dó là lúc ý tưởng xây dựng "**Universe Map**" ra đời:



Trong không gian này, chúng tôi quy định một quy luật đơn giản: "**các ảnh càng giống nhau sẽ càng gần nhau**".

- **Ánh sáng** Dựa trên kết quả phân tích, hệ thống tự động gán các màu sắc rực rỡ (tím, xanh neon, vàng...) cho các nhóm ảnh trùng lặp.
- **Tinh vân** Thay vì hiển thị ảnh thumbnail ngay lập tức gây rối mắt, mỗi bức ảnh được đại diện bằng một đốm sáng (particle). Khi người dùng đưa chuột vào, thông tin chi tiết mới hiện ra. Điều này giúp giao diện luôn sạch sẽ dù tải hàng nghìn dữ liệu.

6 Hạn chế và hướng phát triển

Dựa trên kết quả thực nghiệm và quá trình phân tích hệ thống, nhóm nghiên cứu đã nhận diện được những hạn chế mang tính cốt lõi về mặt thuật toán và dữ liệu, đồng thời đề xuất các hướng đi cụ thể để nâng cấp hệ thống từ mô hình học thuật sang ứng dụng công nghiệp.

6.1 Hạn chế của hệ thống hiện tại

6.1.1 Độ chêch trong đánh giá thuật toán

Kết quả thực nghiệm cho thấy **FAISS** đạt độ chính xác (Accuracy) và F1-Score rất cao, thậm chí vượt trội hơn so với các phương pháp Hashing. Tuy nhiên, nhóm nghiên cứu nhận định rằng kết quả này chịu ảnh hưởng lớn từ cấu trúc của tập dữ liệu thử nghiệm:

- **Vấn đề Top-K:** Các tập dataset hiện tại được cấu trúc sao cho mỗi lớp (class) chỉ chứa trung bình khoảng hơn 10 ảnh trùng lặp. Mặc định FAISS được cấu hình tìm kiếm $TopK = 10$. Sự trùng hợp ngẫu nhiên giữa số lượng ảnh trong một cụm thực tế và tham số K tìm kiếm đã giúp FAISS đạt điểm số tối ưu.
- **Khả năng mở rộng (Scalability):** Trong bài toán thực tế, số lượng ảnh trùng trong một nhóm có thể lên tới hàng chục, hàng trăm hoặc biến thiên không xác định. Khi đó, FAISS với K cố định sẽ bỏ sót đáng kể các ảnh trùng lặp nằm ngoài top K , hoặc phải tăng K lên rất lớn gây lãng phí tài nguyên. Ngược lại, các thuật toán Hashing (SimHash, MinHash) với cơ chế bucket có khả năng gom nhóm không giới hạn số lượng phần tử, về lý thuyết sẽ hoạt động hiệu quả hơn trên các tập dữ liệu quy mô lớn (Large-scale datasets).

6.1.2 Hạn chế về tài nguyên tính toán và Batch Size

Mô hình trích xuất đặc trưng (ResNet50) hiện tại được cấu hình với *batch_size* nhỏ để tương thích với các dòng GPU tầm trung và phổ thông.

- Điều này tạo ra nút thắt cổ chai về hiệu năng khi xử lý lượng dữ liệu lớn, khiến tốc độ xử lý chưa đạt mức thời gian thực (real-time).
- Tuy nhiên, với vai trò là một công cụ lọc ảnh rác và làm sạch dữ liệu (Data Cleaning Tool) trong môi trường học thuật, tốc độ hiện tại vẫn ở mức chấp nhận được và đảm bảo tính ổn định.

6.1.3 Tính chủ quan của thuật toán tìm ngưỡng (Thresholding)

Thuật toán *Valley Detection* (tìm thung lũng) để tự động xác định ngưỡng cắt (threshold) hiện được xây dựng dựa trên các nhận định chủ quan và quan sát thực nghiệm trên số lượng dataset hạn chế.

- Giả định về phân phối "hai đỉnh" (bimodal distribution) của biểu đồ khoảng cách Hamming có thể không chính xác đối với các tập dữ liệu có đặc thù khác (ví dụ: tập dữ liệu quá rời rạc hoặc quá dày đặc).
- Do thiếu tài nguyên và thời gian để thu thập các mẫu dữ liệu đa dạng hơn, thuật toán hiện tại chưa đủ độ tin cậy để triển khai cho các bài toán thương mại hóa đòi hỏi độ chính xác tuyệt đối.

6.2 Hướng phát triển

Để khắc phục các hạn chế trên và hướng tới việc đóng gói sản phẩm thương mại cũng như mở rộng quy mô xử lý, nhóm đề xuất lộ trình phát triển gồm ba mũi nhọn chính:

6.2.1 Kiến trúc lai (Hybrid Pipeline) cho bài toán công nghiệp

Thay vì lựa chọn cực đoan giữa Hashing hoặc FAISS, giải pháp tối ưu cho hệ thống thực tế (Industrial Scale) là kết hợp cả hai theo mô hình phễu lọc (Coarse-to-Fine):

1. **Bước 1 - Sàng lọc thô:** Sử dụng các kỹ thuật Hashing (như SimHash hoặc MinHash) làm bước trung gian để xử lý nhanh toàn bộ dữ liệu. Bước này giúp giảm không gian tìm kiếm và loại bỏ phần lớn lượng ảnh trùng lặp cơ bản với chi phí tính toán thấp.
2. **Bước 2 - Tinh chỉnh:** Kết quả từ bước Hashing sẽ được đưa vào FAISS hoặc so sánh Cosine trực tiếp để lọc lại lần cuối. Điều này giúp loại bỏ các trường hợp mất đặc trưng do chuẩn hóa mã băm và đảm bảo độ chính xác cao nhất cho kết quả đầu ra.

6.2.2 Tối ưu hóa tài nguyên và Batch Size

Để nâng cao hiệu suất:

- Tiến hành benchmark và tinh chỉnh (tuning) tham số *batch_size* tối ưu cho các dòng GPU cao cấp hơn.
- Sử dụng các kỹ thuật lượng tử hóa mô hình (Model Quantization) hoặc chuyển đổi sang định dạng ONNX/TensorRT để tăng tốc độ suy luận (Inference) của ResNet mà không làm giảm chất lượng vector đặc trưng.

6.2.3 Nghiên cứu sâu về phân phối khoảng cách

Nếu có đủ thời gian và tài nguyên dữ liệu lớn hơn trong tương lai:

- Nhóm sẽ thực hiện phân tích thống kê sâu hơn trên biểu đồ phân phối khoảng cách Hamming (Hamming Distance Histograms) của nhiều loại dataset khác nhau.
- Mục tiêu là xây dựng một mô hình xác định ngưỡng thích nghi (Adaptive Thresholding) dựa trên học máy (Machine Learning) thay vì các luật heuristic cố định, từ đó đảm bảo tính ổn định của hệ thống khi triển khai thương mại hóa.

6.2.4 Nâng cấp lên Deep Hashing và Tìm kiếm ngữ nghĩa

Đây là hướng đi khác mang tính đột phá nhằm giải quyết triệt để bài toán về tốc độ và ngữ nghĩa ảnh mà phương pháp truyền thống đang gặp phải:

- **Chuyển đổi sang Deep Hashing:** Phương pháp hiện tại tách rời hai bước (Trích xuất đặc trưng ResNet → Băm) khiến thông tin bị mất mát đáng kể khi nén vector số thực (float) thành chuỗi bits. Giải pháp là sử dụng các mô hình *Deep Hashing* (như HashNet, DSH) có khả năng học "End-to-End" trực tiếp từ pixel ảnh ra mã nhị phân, giúp bảo toàn tối đa thông tin ngữ nghĩa trong mã băm.
- **Giải quyết nút tắc tốc độ ResNet:** Hiện tại, thời gian trích xuất đặc trưng của ResNet chiếm phần lớn độ trễ của hệ thống. Deep Hashing cho phép xây dựng các mạng CNN nhẹ hơn, được tối ưu hóa chuyên biệt để sinh ra mã nhị phân, giúp giảm thiểu khối lượng tính toán và tăng tốc độ xử lý gấp nhiều lần.
- **Mở rộng bài toán Semantic Search (Tương tự Google Lens):** Nhờ khả năng học ngữ nghĩa sâu, hệ thống có thể nâng cấp từ việc "tìm ảnh trùng lặp" (duplicate detection) sang "tìm ảnh tương đồng ngữ nghĩa" (semantic similarity). Ví dụ: Nhận diện cùng một sự vật ở các góc chụp, ánh sáng khác nhau. Đây là nền tảng để xây dựng các công cụ tìm kiếm quy mô cực lớn (hàng nghìn tỷ ảnh) với tốc độ phản hồi siêu tốc, tiệm cận với khả năng của Google Lens.



7 Lời Kết

Trước tiên, chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy **Lê Thành Sách**, người đã tận tâm giảng dạy, hướng dẫn và truyền đạt những kiến thức giá trị trong suốt quá trình học tập và thực hiện bài tập lớn này. Những bài giảng của thầy không chỉ giúp chúng em nắm vững kiến thức chuyên môn mà còn rèn luyện tư duy logic, kỹ năng nghiên cứu khoa học, từ đó hoàn thành bài tập với kết quả tốt nhất có thể.

Chúng em cũng xin gửi lời cảm ơn chân thành đến các thầy cô trong **Khoa Khoa Học Và Kỹ Thuật Máy Tính** đã luôn tạo điều kiện thuận lợi, xây dựng nền tảng kiến thức vững chắc và khơi dậy niềm đam mê học tập trong mỗi sinh viên.

Bên cạnh đó, chúng em xin trân trọng cảm ơn sự hợp tác và nỗ lực của các thành viên trong nhóm, những người đã làm việc chăm chỉ để hoàn thành bài tập lớn trong một khoảng thời gian ngắn. Đồng thời, chúng em cũng ghi nhận và biết ơn sự hỗ trợ, góp ý quý báu từ các bạn cùng lớp, cũng như từ những tài liệu tham khảo đã cung cấp những thông tin hữu ích cho quá trình thực hiện bài tập lớn này.

Dù đã cố gắng hết sức, nhưng bài tập lớn này vẫn khó tránh khỏi những hạn chế. Chúng em rất mong nhận được sự góp ý từ thầy và các bạn để có thể tiếp tục cải tiến và hoàn thiện hơn trong tương lai.

Trân trọng cảm ơn!

Nhóm 4 - Lớp TN01

Tài liệu tham khảo

- [1] B. H. Bloom. "Space/time trade-offs in hash coding with allowable errors." *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [2] A. Z. Broder. "On the resemblance and containment of documents." *Proceedings of Compression and Complexity of Sequences 1997*, IEEE, pp. 21–29, 1997.
- [3] G. S. Manku, A. Jain, and A. D. Sarma. "Detecting near-duplicates for web crawling." *Proceedings of the 16th International Conference on World Wide Web (WWW)*, ACM, pp. 141–150, 2007.
- [4] Moses S. Charikar. "Similarity Estimation Techniques from Rounding Algorithms." *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, Princeton University, 2002.
- [5] A. Appleby. "MurmurHash3: A Non-Cryptographic Hash Function." *GitHub Repository*, 2008. Available: <https://github.com/aappleby/smhasher>
- [6] H. P. Luhn. "The Automatic Creation of Literature Abstracts." *IBM Journal of Research and Development*, vol. 2, no. 2, pp. 159–165, 1958.
- [7] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." *CVPR*, 2016.
- [8] Suman K. Bera et al. "Advanced Bloom Filter Based Algorithms for Efficient Approximate Data De-Duplication in Streams." *arXiv preprint arXiv:1212.3964*, 2012.
- [9] Andrea Salvi et al. "Bloom Filters and Compact Hash Codes for Efficient and Distributed Image Retrieval." *IEEE International Symposium on Multimedia (ISM)*, pp. 515–520, 2016.
- [10] André Araujo et al. "Large-Scale Query-by-Image Video Retrieval Using Bloom Filters." *IEEE Transactions on Circuits and Systems for Video Technology* (arXiv version), 2016.
- [11] Qin-Zhen Guo et al. "Simhash for large scale image retrieval." *Applied Mechanics and Materials*, Vols. 651-653, pp. 2197–2200, 2014.