

# AI VIET NAM - Project

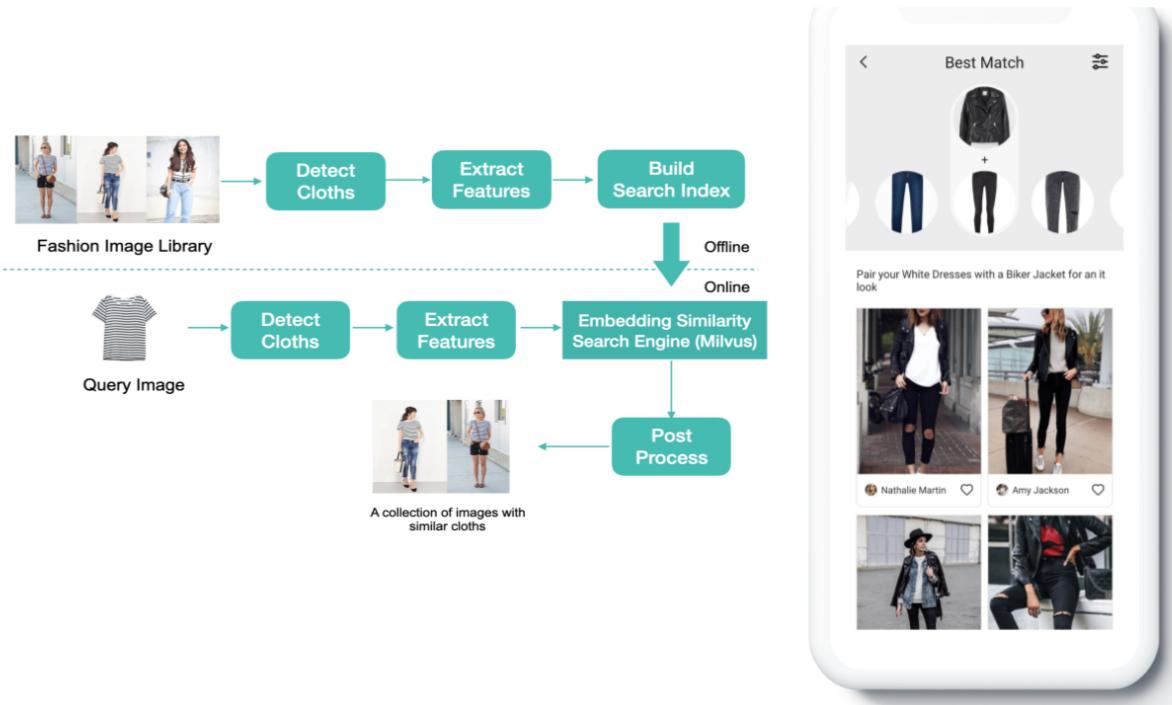
## Truy vấn ảnh dùng các toán cơ bản

Tho-Anh-Khoa Nguyen, Dang-Nha Nguyen

Ngày 3 tháng 8 năm 2024

### 1 Giới thiệu

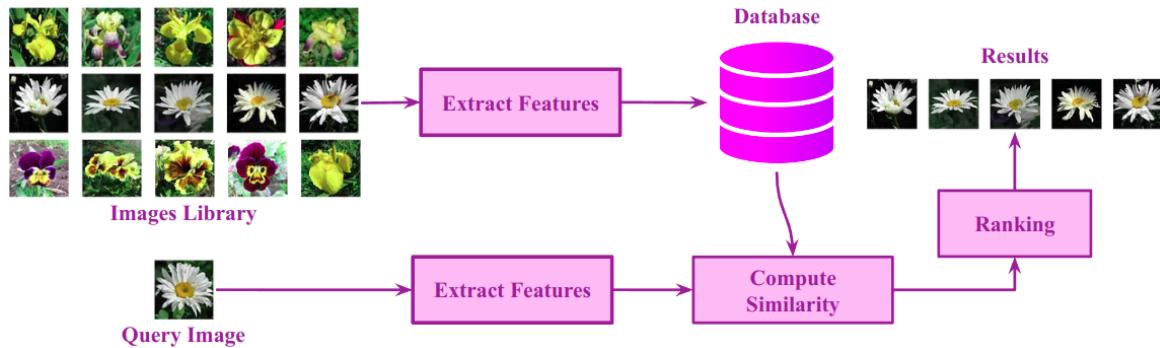
Truy vấn hình ảnh (Images Retrieval) là một bài toán thuộc lĩnh vực Truy vấn thông tin (Information Retrieval). Trong đó, nhiệm vụ của ta là xây dựng một chương trình trả về các hình ảnh (Images) có liên quan đến hình ảnh truy vấn đầu vào(Query) và các hình ảnh được lấy từ một bộ dữ liệu hình ảnh cho trước, hiện nay có một số ứng dụng truy vấn ảnh như: Google Search Image, chức năng tìm kiếm sản phẩm bằng hình ảnh trên Shopee, Lazada, Tiki, ...



Hình 1: Ứng dụng của Image Retrieval.

#### 1.1 Yêu cầu của bài toán

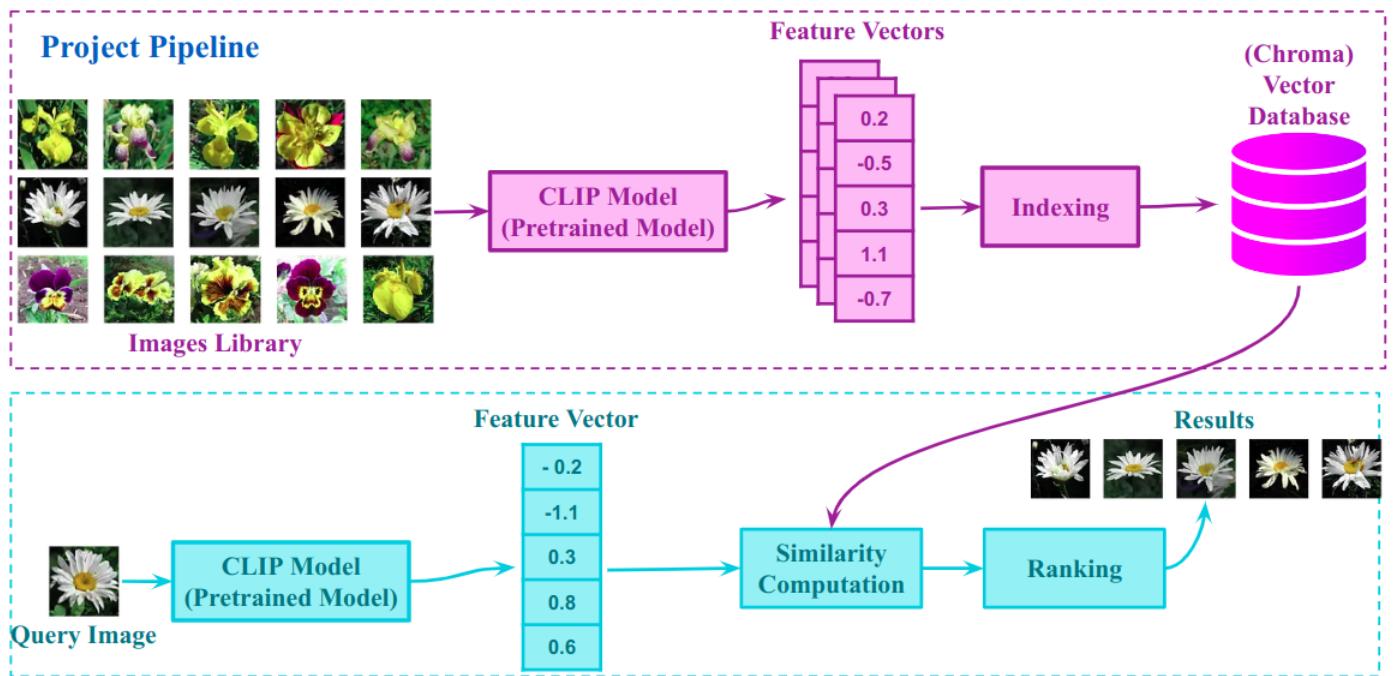
Xây dựng một chương trình mà khi đưa vào chương trình đó một hình ảnh, nó sẽ hiển thị nhiều hình ảnh tương tự. Có rất nhiều cách thiết kế hệ thống truy vấn hình ảnh khác nhau, tuy nhiên về mặt tổng quát sẽ có pipeline như sau:



Hình 2: Pipeline tổng quan của một hệ thống Images Retrieval.

Dựa vào hình trên, có thể phát biểu Input/Output của một hệ thống truy vấn văn bản bao gồm:

- **Input:** Hình ảnh truy vấn Query Image và bộ dữ liệu Images Library.
- **Output:** Danh sách các hình ảnh có sự tương tự đến hình ảnh truy vấn.



Hình 3: Image retrieval pipeline của project

Trong dự án này, chúng ta sẽ xây dựng một hệ thống truy xuất hình ảnh bằng cách sử dụng mô hình deep learning đã được huấn luyện trước (CLIP) để trích xuất đặc trưng của ảnh và thu được các vector đặc trưng. Sau đó, chúng ta sẽ sử dụng vector database để index, lưu trữ và truy xuất các ảnh tương tự với ảnh yêu cầu thông qua các thuật toán đo độ tương đồng.

## 2 Xây dựng chương trình

Dự án này sẽ giới thiệu các phương pháp từ cơ bản đến nâng cao để xây dựng một hệ thống truy vấn ảnh. Chúng ta sẽ phát triển hệ thống này trên một tập dữ liệu cụ thể. Tiếp theo, dự án sẽ tiến hành thu thập và xử lý dữ liệu để tạo ra hệ thống truy vấn ảnh được cá nhân hóa cho từng người dùng (phần đọc thêm (optional)). Các mục tiêu chính của dự án bao gồm:

- Xây dựng chương trình truy vấn ảnh cơ bản.
- Phát triển chương trình truy vấn ảnh nâng cao với CLIP model và vector database.
- (Optional) Thu thập và xử lý dữ liệu nhằm mục đích xây dựng chương trình truy vấn ảnh cá nhân hóa.

### 2.1 Chương Trình Truy Vấn Ảnh Cơ Bản:

Mục này tập trung vào việc thiết kế và triển khai một hệ thống truy vấn ảnh đơn giản, giúp người dùng có thể tìm kiếm và truy xuất hình ảnh tương tự. Để có thể tìm được các hình ảnh có liên quan đến hình ảnh truy vấn, ta có thể sử dụng các độ được trình bày ở các mục tiếp theo để đo sự tương đồng giữa hai ảnh.

Các bạn sẽ được cung cấp một tập data trong đó đường dẫn data/train là nơi chứa data sẽ trả về kết quả truy vấn, còn data/test là nơi chứa ảnh sẽ được đem đi truy vấn

Đầu tiên chúng ta sẽ import một số thư viện cần thiết. Để đọc ảnh chúng ta sử dụng thư viện PIL, để xử lí ma trận chúng ta sử dụng numpy, để thao tác với thư mục, file chúng ta sử dụng thư viện os, sử dụng matplotlib để hiển thị kết quả.

```
1 import os
2 import numpy as np
3 from PIL import Image
4 import matplotlib.pyplot as plt
```

Tiếp theo chúng ta sẽ lấy danh sách các class của ảnh trong data mà ta có

```
1 ROOT = 'data'
2 CLASS_NAME = sorted(list(os.listdir(f'{ROOT}/train')))
```

Sau đó để thực hiện tính toán trên các hình ảnh, chúng ta sẽ đọc ảnh, resize về kích thước chung (thì mới áp dụng được các phép đo) và chuyển đổi nó về dạng numpy:

```
1 def read_image_from_path(path, size):
2     im = Image.open(path).convert('RGB').resize(size)
3     return np.array(im)
4
5 def folder_to_images(folder, size):
6     list_dir = [folder + '/' + name for name in os.listdir(folder)]
7     images_np = np.zeros(shape=(len(list_dir), *size, 3))
8     images_path = []
9     for i, path in enumerate(list_dir):
10         images_np[i] = read_image_from_path(path, size)
11         images_path.append(path)
12     images_path = np.array(images_path)
13     return images_np, images_path
```

### 2.1.1 Truy vấn hình ảnh với độ đo L1

Chúng ta xây dựng một hàm *absolute\_difference()* tính độ tương đồng giữa các hình ảnh. Trong ví dụ này chúng ta sẽ sử dụng hàm L1. Hàm L1 có công thức tính như sau:

$$L1(\vec{a}, \vec{b}) = \sum_{i=1}^N |a_i - b_i|$$

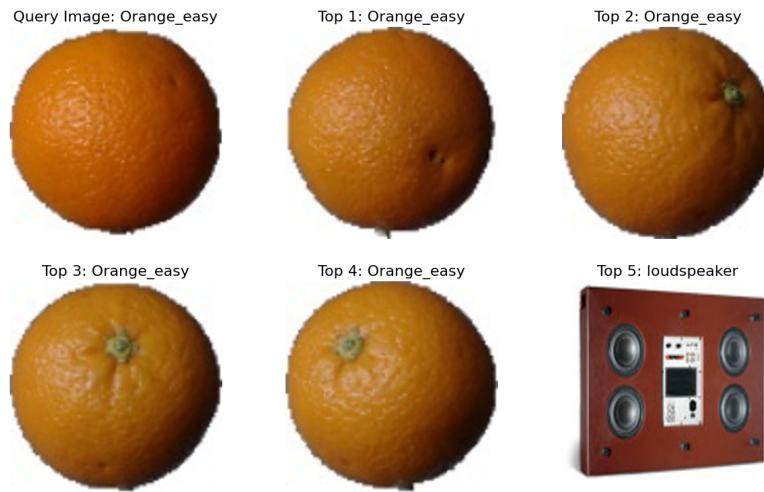
```
1 def absolute_difference(query, data):
2     axis_batch_size = tuple(range(1, len(data.shape)))
3     return np.sum(np.abs(data - query), axis=axis_batch_size)
```

Đến đây chúng ta sẽ thực hiện tính toán để tính độ tương đồng giữa ảnh input và các hình ảnh trong bộ dữ liệu. Chúng ta sẽ tạo hàm *get\_l1\_score()*, hàm này sẽ trả về ảnh *query* và *ls\_path\_score* chứa danh sách hình ảnh và giá trị độ tương đồng với từng ảnh.

```
1 def get_l1_score(root_img_path, query_path, size):
2     query = read_image_from_path(query_path, size)
3     ls_path_score = []
4     for folder in os.listdir(root_img_path):
5         if folder in CLASS_NAME:
6             path = root_img_path + folder
7             images_np, images_path = folder_to_images(path, size) # mang numpy nhieu anh,
8             paths
9             rates = absolute_difference(query, images_np)
10            ls_path_score.extend(list(zip(images_path, rates)))
11    return query, ls_path_score
```

Đoạn code này thực hiện quá trình truy xuất hình ảnh bằng cách so sánh một hình ảnh truy vấn với các hình ảnh trong tập huấn luyện dựa trên điểm L1. Đầu tiên, các hình ảnh được thay đổi cùng kích thước. Tiếp theo hệ thống sẽ so sánh ảnh truy vấn với các hình ảnh trong thư mục huấn luyện để tính điểm L1. Sau đó, kết quả truy vấn được trả về là danh sách các đường dẫn chứa hình ảnh và điểm số tính theo L1. Cuối cùng 5 kết quả tốt nhất sẽ được hiển thị cùng với ảnh truy vấn

```
1 root_img_path = f"{ROOT}/train/"
2 query_path = f"{ROOT}/test/Orange_easy/0_100.jpg"
3 size = (448, 448)
4 query, ls_path_score = get_l1_score(root_img_path, query_path, size)
5 plot_results(query_path, ls_path_score, reverse=False)
```

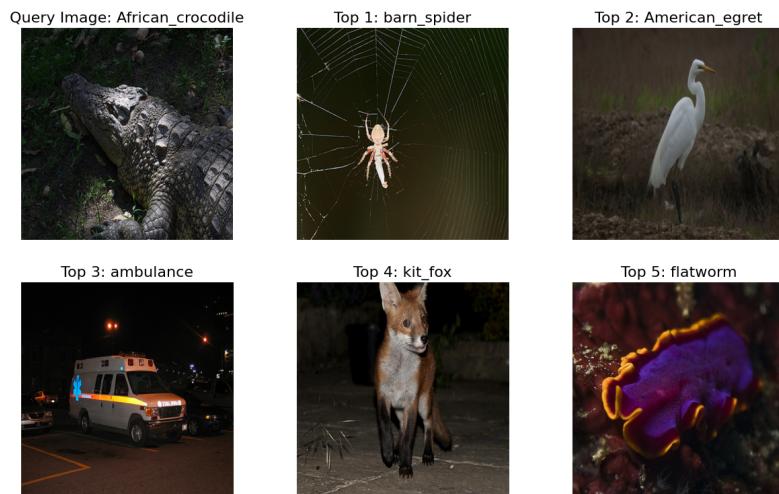


Hình 4: Image retrieval ảnh đơn giản với L1

```

1 root_img_path = f"{ROOT}/train/"
2 query_path = f"{ROOT}/test/African_crocodile/n01697457_18534.JPG"
3 size = (448, 448)
4 query, ls_path_score = get_l1_score(root_img_path, query_path, size)
5 plot_results(query_path, ls_path_score, reverse=False)

```



Hình 5: Image retrieval ảnh đơn phức tạp L1

Tiếp theo chúng ta sẽ truy vấn hình ảnh với 3 độ đo L2, Cosine Similarity, Correlation Coefficient. Chúng ta sẽ thực hiện code tương tự như độ đo L1.

### 2.1.2 Truy vấn hình ảnh với độ đo L2

$$L2(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^N (a_i - b_i)^2}$$

Chúng ta tạo hàm tính độ tương đồng L2 mean\_square\_difference()

```

1 def mean_square_difference(query, data):
2     axis_batch_size = tuple(range(1, len(data.shape)))
3     return np.mean((data - query)**2, axis=axis_batch_size)

```

Chúng ta tạo hàm get\_l2\_score(), hàm này tương tự hàm get\_l1\_score(), chỉ cần thay đổi tại biến rates.

```

1 def get_l2_score(root_img_path, query_path, size):
2     query = read_image_from_path(query_path, size)
3     ls_path_score = []
4     for folder in os.listdir(root_img_path):
5         if folder in CLASS_NAME:
6             path = root_img_path + folder
7             images_np, images_path = folder_to_images(path, size) # mang numpy nhieu anh,
8             paths
9             rates = mean_square_difference(query, images_np)
10            ls_path_score.extend(list(zip(images_path, rates)))
11    return query, ls_path_score

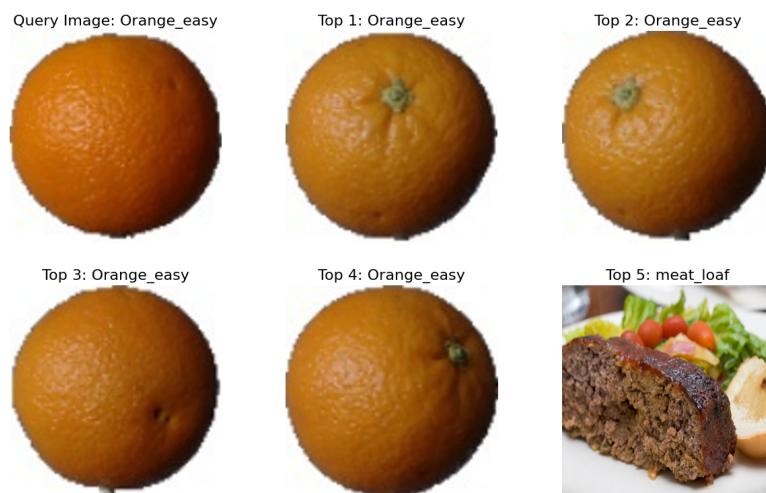
```

Vì với độ đo L2 này thì giá trị càng nhỏ sẽ càng giống nhau. Nên để hiển thị kết quả chúng ta sử dụng hàm plot\_results() như phép đo L1.

```

1 root_img_path = f"{ROOT}/train/"
2 query_path = f"{ROOT}/test/Orange_easy/0_100.jpg"
3 size = (448, 448)
4 query, ls_path_score = get_l2_score(root_img_path, query_path, size)
5 plot_results(query_path, ls_path_score, reverse=False)

```



Hình 6: Image retrieval ảnh đơn giản với L2

```

1 root_img_path = f"{ROOT}/train/"
2 query_path = f"{ROOT}/test/African_crocodile/n01697457_18534.JPG"
3 size = (448, 448)
4 query, ls_path_score = get_l2_score(root_img_path, query_path, size)
5 plot_results(query_path, ls_path_score, reverse=False)

```



Hình 7: Image retrieval ảnh đơn phức tạp L2

### 2.1.3 Truy vấn hình ảnh với độ đo Cosine Similarity

$$\text{cosine\_similarity}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_{i=1}^N a_i b_i}{\sqrt{\sum_{i=1}^N a_i^2} \sqrt{\sum_{i=1}^N b_i^2}}$$

Chúng ta tạo hàm tính độ tương đồng cosine\_similarity()

```

1 def cosine_similarity(query, data):
2     axis_batch_size = tuple(range(1, len(data.shape)))
3     query_norm = np.sqrt(np.sum(query**2))
4     data_norm = np.sqrt(np.sum(data**2, axis=axis_batch_size))
5     return np.sum(data * query, axis=axis_batch_size) / (query_norm*data_norm + np.finfo(float).eps)

```

Chúng ta tạo hàm get\_cosine\_similarity(), hàm này tương tự hàm get\_l1\_score().

```

1 def get_cosine_similarity_score(root_img_path, query_path, size):
2     query = read_image_from_path(query_path, size)
3     ls_path_score = []
4     for folder in os.listdir(root_img_path):
5         if folder in CLASS_NAME:
6             path = root_img_path + folder
7             images_np, images_path = folder_to_images(path, size) # mang numpy nhieu anh,
8             paths
9             rates = cosine_similarity(query, images_np)
10            ls_path_score.extend(list(zip(images_path, rates)))
11    return query, ls_path_score

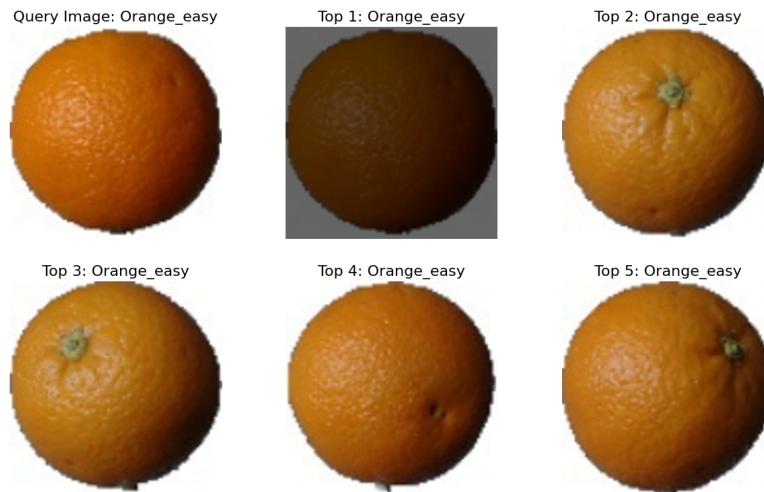
```

Để hiển thị kết quả chúng ta sử dụng hàm plot\_results(), tuy nhiên ở hàm này chúng ta sẽ sắp xếp giá trị giảm dần từ lớn đến nhỏ vì với độ đo này thì giá trị càng lớn sẽ càng giống nhau, cho nên chúng ta sử dụng reverse = True.

```

1 root_img_path = f"{ROOT}/train/"
2 query_path = f"{ROOT}/test/Orange_easy/0_100.jpg"
3 size = (448, 448)
4 query, ls_path_score = get_cosine_similarity_score(root_img_path, query_path, size)
5 plot_results(query_path, ls_path_score, reverse=True)

```



Hình 8: Image retrieval ảnh đơn giản với Cosine Similarity

```

1 root_img_path = f"{ROOT}/train/"
2 query_path = f"{ROOT}/test/African_crocodile/n01697457_18534.JPG"
3 size = (448, 448)
4 query, ls_path_score = get_cosine_similarity_score(root_img_path, query_path, size)
5 plot_results(query_path, ls_path_score, reverse=True)

```



Hình 9: Image retrieval ảnh đơn phức tạp Cosine Similarity

#### 2.1.4 Truy vấn hình ảnh với độ đo Correlation Coefficient

$$r = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} = \frac{\sum (x_i - \mu_X)(y_i - \mu_Y)}{\sqrt{\sum (x_i - \mu_X)^2} \sqrt{\sum (y_i - \mu_Y)^2}}$$

Chúng ta tạo hàm tính độ tương đồng correlation\_coefficient()

```

1 def correlation_coefficient(query, data):
2     axis_batch_size = tuple(range(1, len(data.shape)))
3     query_mean = query - np.mean(query)
4     data_mean = data - np.mean(data, axis=axis_batch_size, keepdims=True)

```

```

5     query_norm = np.sqrt(np.sum(query_mean**2))
6     data_norm = np.sqrt(np.sum(data_mean**2, axis=axis_batch_size))
7
8     return np.sum(data_mean * query_mean, axis=axis_batch_size) / (query_norm*data_norm + np.
9         finfo(float).eps)

```

Chúng ta tạo hàm get\_correlation\_coefficient\_score(), hàm này tương tự hàm get\_l1\_score()

```

1 def get_correlation_coefficient_score(root_img_path, query_path, size):
2     query = read_image_from_path(query_path, size)
3     ls_path_score = []
4     for folder in os.listdir(root_img_path):
5         if folder in CLASS_NAME:
6             path = root_img_path + folder
7             images_np, images_path = folder_to_images(path, size) # mang numpy nhieu anh,
8                 paths
9             rates = correlation_coefficient(query, images_np)
10            ls_path_score.extend(list(zip(images_path, rates)))
11
12    return query, ls_path_score

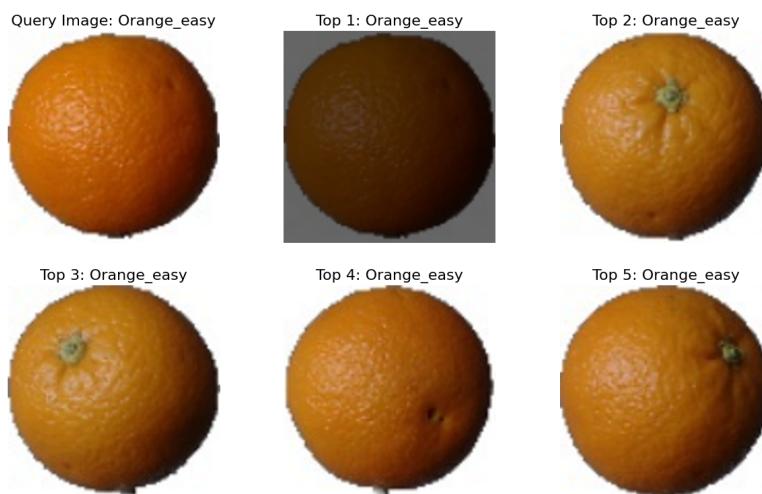
```

Để hiển thị kết quả chúng ta sử dụng hàm plot\_results(), tuy nhiên ở hàm này chúng ta sẽ sắp xếp giá trị giảm dần từ lớn đến nhỏ vì với độ đo này thì giá trị càng lớn sẽ càng giống nhau, cho nên chúng ta sử dụng reverse = True.

```

1 root_img_path = f"{ROOT}/train/"
2 query_path = f"{ROOT}/test/Orange_easy/0_100.jpg"
3 size = (448, 448)
4 query, ls_path_score = get_correlation_coefficient_score(root_img_path, query_path, size)
5 plot_results(query_path, ls_path_score, reverse=True)

```



Hình 10: Image retrieval ảnh đơn giản với Correlation Coefficient

```

1 root_img_path =f"{ROOT}/train/"
2 query_path = f"{ROOT}/test/African_crocodile/n01697457_18534.JPG"
3 size = (448, 448)
4 query, ls_path_score = get_correlation_coefficient_score(root_img_path, query_path, size)
5 plot_results(query_path, ls_path_score, reverse=True)

```



Hình 11: Image retrieval ảnh đơn phức tạp Correlation Coefficient

## 2.2 Nâng Cao Chương Trình Truy Vấn Ảnh

Phần này nhằm phát triển các tính năng nâng cao cho hệ thống truy vấn ảnh, bằng cách sử dụng deep learning model trích xuất feature vector cho các ảnh để tăng cường khả năng truy xuất hình ảnh chính xác hơn.

Image retrieval sử dụng pretrained deep learning model là một quá trình mà trong đó, các mô hình học sâu đã được đào tạo trước được sử dụng để tìm kiếm và lấy ra các hình ảnh liên quan từ một tập dữ liệu lớn dựa trên nội dung hình ảnh. Các mô hình này thường được huấn luyện trên các tập dữ liệu khổng lồ với hàng triệu hình ảnh để học các đặc điểm quan trọng từ ảnh, cho phép chúng hiểu và biểu diễn nội dung ảnh một cách hiệu quả.

Trong quá trình tìm kiếm ảnh, một hình ảnh truy vấn được đưa vào mô hình, mô hình sẽ tính toán đặc trưng của hình ảnh truy vấn và so sánh chúng với các đặc trưng đã được tính toán trước của những hình ảnh được lưu trữ trên hệ thống. Sự tương đồng giữa các đặc trưng này được sử dụng để xác định các hình ảnh có liên quan nhất, và kết quả là những hình ảnh tương tự nhất với hình ảnh truy vấn được trả về cho người dùng. Những mô hình này có khả năng phân tích và nhận diện các đặc tính phức tạp của ảnh như kết cấu, hình dạng, và màu sắc, do đó chúng rất hiệu quả trong việc tìm kiếm và lấy lại hình ảnh dựa trên nội dung.

### 2.2.1 Truy vấn hình ảnh với pretrained deep learning model

Để bắt đầu quá trình truy vấn hình ảnh sử dụng pretrained deep learning model, trước tiên, chúng ta cần cài đặt hai thư viện quan trọng là chromadb và open-clip-torch. Thư viện chromadb hỗ trợ việc quản lý và truy xuất dữ liệu hình ảnh hiệu quả (chúng ta cũng sử dụng thêm với mục đích tạo vector database), và chromadb có thể dùng open-clip-torch để cung cấp khả năng sử dụng mô hình CLIP đã được đào tạo sẵn, đây là một công cụ mạnh mẽ để phân tích nội dung hình ảnh thông qua học sâu.

```

1 pip install chromadb
2 pip install open-clip-torch

```

Import các thư viện cần thiết

```

1 import os
2 import numpy as np
3 from PIL import Image
4 import matplotlib.pyplot as plt

```

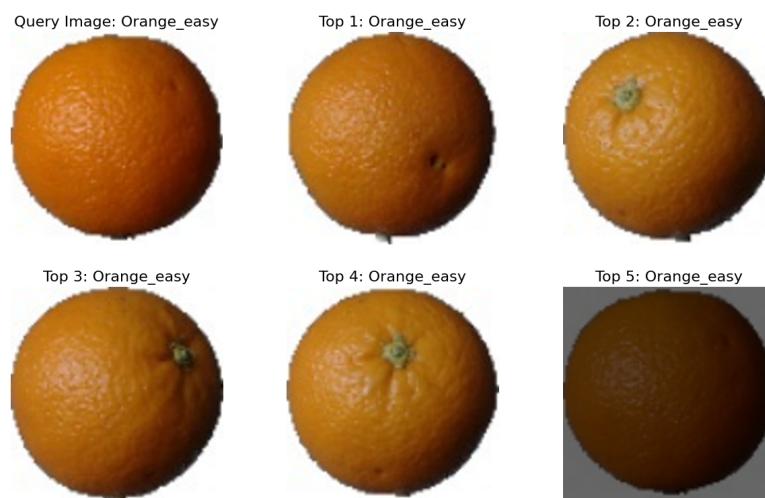
```
5 from chromadb.utils.embedding_functions import OpenCLIPEmbeddingFunction
```

Quá trình thực hiện truy vấn hình ảnh sử dụng mô hình CLIP trong bối cảnh này tương tự như các bước ở phần cơ bản trước, nhưng chúng ta sẽ nâng cấp bằng cách thêm một hàm để trích xuất vector đặc trưng cho mỗi hình ảnh. Mô hình CLIP sẽ được sử dụng để biến đổi hình ảnh thành các vector đặc trưng đại diện cho nội dung và ngữ cảnh của hình ảnh đó. Sau đó, việc so sánh các hình ảnh không được thực hiện trực tiếp trên ảnh gốc mà là thông qua việc tính sự tương đồng giữa các vector này. Đoạn code bên dưới khởi tạo một hàm để trích xuất vector đặc trưng từ một hình sử dụng mô hình CLIP. Tiếp theo, hàm get\_single\_image\_embedding nhận một hình ảnh làm đầu vào và sử dụng phương thức \_encode\_image của OpenCLIPEmbeddingFunction để trích xuất ảnh thành một vector đặc trưng.

```
1 embedding_function = OpenCLIPEmbeddingFunction()
2
3 def get_single_image_embedding(image):
4     embedding = embedding_function._encode_image(image=image)
5     return np.array(embedding)
```

**Truy vấn embedding vector với độ đo L1** hàm get\_l1\_score được nâng cấp lên bằng cách sử dụng CLIP model để trích xuất vector đặc trưng.

```
1 def get_l1_score(root_img_path, query_path, size):
2     query = read_image_from_path(query_path, size)
3     query_embedding = get_single_image_embedding(query)
4     ls_path_score = []
5     for folder in os.listdir(root_img_path):
6         if folder in CLASS_NAME:
7             path = root_img_path + folder
8             images_np, images_path = folder_to_images(path, size) # mang numpy nhieu anh,
9                 paths
10            embedding_list = []
11            for idx_img in range(images_np.shape[0]):
12                embedding = get_single_image_embedding(images_np[idx_img].astype(np.uint8))
13                embedding_list.append(embedding)
14            rates = absolute_difference(query_embedding, np.stack(embedding_list))
15            ls_path_score.extend(list(zip(images_path, rates)))
16    return query, ls_path_score
```



Hình 12: Image retrieval ảnh đơn giản với model CLIP và L1



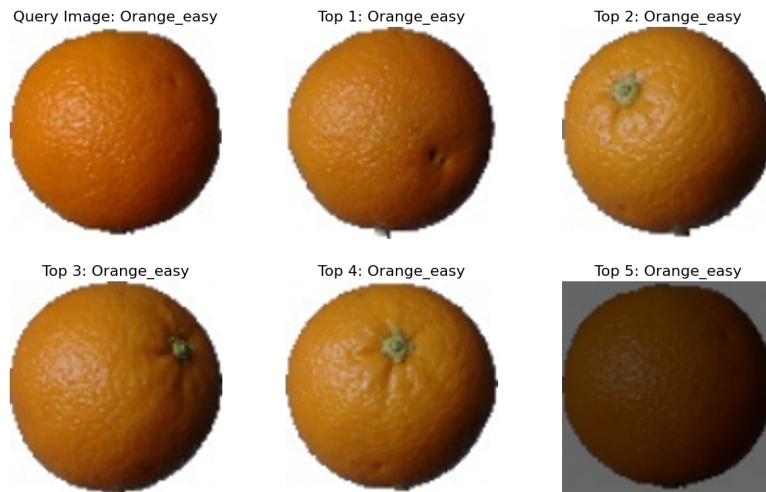
Hình 13: Image retrieval ảnh phức tạp với model Clip và L1

**Truy vấn embedding vector với độ đo L2** hàm get\_l2\_score được nâng cấp lên bằng cách sử dụng CLIP model để trích xuất vector đặc trưng.

```

1 def get_l2_score(root_img_path, query_path, size):
2     query = read_image_from_path(query_path, size)
3     query_embedding = get_single_image_embedding(query)
4     ls_path_score = []
5     for folder in os.listdir(root_img_path):
6         if folder in CLASS_NAME:
7             path = root_img_path + folder
8             images_np, images_path = folder_to_images(path, size) # mang numpy nhieu anh,
9                 paths
10            embedding_list = []
11            for idx_img in range(images_np.shape[0]):
12                embedding = get_single_image_embedding(images_np[idx_img].astype(np.uint8))
13                embedding_list.append(embedding)
14            rates = mean_square_difference(query_embedding, np.stack(embedding_list))
15            ls_path_score.extend(list(zip(images_path, rates)))
16    return query, ls_path_score

```



Hình 14: Image retrieval ảnh đơn giản với model CLIP và L2



Hình 15: Image retrieval ảnh phức tạp với model Clip và L2

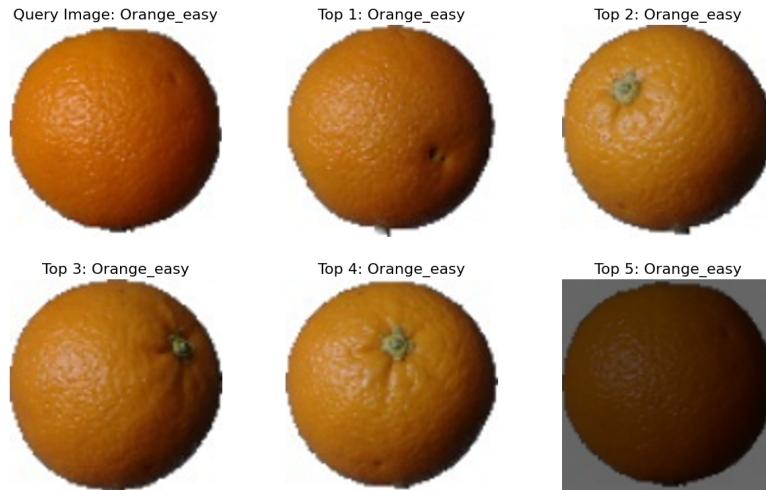
**Truy vấn embedding vector với độ đo Cosine Similarity** hàm `get_cosine_similarity_score` được nâng cấp lên bằng cách sử dụng CLIP model để trích xuất vector đặc trưng.

```

1 def get_cosine_similarity_score(root_img_path, query_path, size):
2     query = read_image_from_path(query_path, size)
3     query_embedding = get_single_image_embedding(query)
4     ls_path_score = []
5     for folder in os.listdir(root_img_path):
6         if folder in CLASS_NAME:
7             path = root_img_path + folder
8             images_np, images_path = folder_to_images(path, size) # mang numpy nhieu anh,
9                 paths
10            embedding_list = []
11            for idx_img in range(images_np.shape[0]):
12                embedding = get_single_image_embedding(images_np[idx_img].astype(np.uint8))
13                embedding_list.append(embedding)
14            rates = cosine_similarity(query_embedding, np.stack(embedding_list))
15            ls_path_score.extend(list(zip(images_path, rates)))

```

```
15     return query, ls_path_score
```



Hình 16: Image retrieval ảnh đơn giản với model CLIP và Cosine Similarity



Hình 17: Image retrieval ảnh phức tạp với model Clip và Cosine Similarity

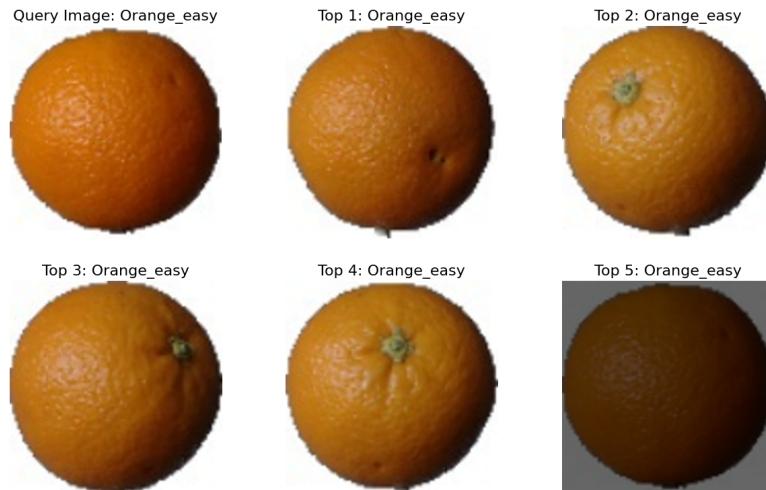
**Truy vấn embedding vector với độ đo Correlation Coefficient** hàm `get_correlation_coefficient_score` được nâng cấp lên bằng cách sử dụng CLIP model để trích xuất vector đặc trưng.

```
1 def get_correlation_coefficient_score(root_img_path, query_path, size):
2     query = read_image_from_path(query_path, size)
3     query_embedding = get_single_image_embedding(query)
4     ls_path_score = []
5     for folder in os.listdir(root_img_path):
6         if folder in CLASS_NAME:
7             path = root_img_path + folder
8             images_np, images_path = folder_to_images(path, size) # mang numpy nhieu anh,
9             paths
10            embedding_list = []
11            for idx_img in range(images_np.shape[0]):
12                embedding = get_single_image_embedding(images_np[idx_img].astype(np.uint8))
```

```

12     embedding_list.append(embedding)
13     rates = correlation_coefficient(query_embedding, np.stack(embedding_list))
14     ls_path_score.extend(list(zip(images_path, rates)))
15
return query, ls_path_score

```



Hình 18: Image retrieval ảnh đơn giản với model CLIP và Correlation Coefficient



Hình 19: Image retrieval ảnh phức tạp với model Clip và Correlation Coefficient

## 2.2.2 Tối ưu hoá quá trình truy vấn hình ảnh sử dụng mô hình CLIP và cơ sở dữ liệu vector

Chúng ta sẽ tiếp tục phát triển phương pháp ở trên, vì mỗi lần truy vấn đều cần phải sử dụng lại mô hình CLIP, phương pháp này sẽ sử dụng một cơ sở dữ liệu vector (vector database) để quản lý các embedding vector, giúp quá trình truy vấn được tối ưu hơn.

Bước đầu tiên là trích xuất vector đặc trưng từ các ảnh và lưu trữ chúng vào cơ sở dữ liệu. Đầu tiên, ta cần lấy danh sách đường dẫn của các ảnh mà ta muốn trích xuất vector. Đoạn code dưới đây mô tả quá trình trích xuất đường dẫn của các ảnh từ một thư mục cho trước. Đầu tiên, chúng ta sẽ liệt kê các thư mục con dựa trên tên của các class (CLASS\_NAME). Sau đó, liệt kê tất cả các ảnh trong mỗi

thư mục con và lưu trữ đường dẫn của từng ảnh vào một danh sách.

```

1 def get_files_path(path):
2     files_path = []
3     for label in CLASS_NAME:
4         label_path = path + "/" + label
5         filenames = os.listdir(label_path)
6         for filename in filenames:
7             filepath = label_path + '/' + filename
8             files_path.append(filepath)
9     return files_path
10
11 data_path = f'{ROOT}/train'
12 files_path = get_files_path(path=data_path)

```

**Truy vấn ảnh với L2 Collection** Trong ChromaDB, "collection" là một khái niệm quan trọng, dùng để tổ chức và quản lý dữ liệu. Một collection trong ChromaDB có thể được hiểu như là một tập hợp các vector hoặc tài liệu được chỉ mục và lưu trữ cùng nhau dựa trên một số tiêu chí hoặc đặc điểm chung. Nó tương tự như concept của "table" trong cơ sở dữ liệu quan hệ hoặc "collection" trong MongoDB. Đoạn code sau đây định nghĩa hàm add\_embedding, một hàm giúp trích xuất và lưu trữ các vector đặc trưng của ảnh vào một collection đã được tạo.

```

1 def add_embedding(collection, files_path):
2     ids = []
3     embeddings = []
4     for id_filepath, filepath in tqdm(enumerate(files_path)):
5         ids.append(f'id_{id_filepath}')
6         image = Image.open(filepath)
7         embedding = get_single_image_embedding(image=image)
8         embeddings.append(embedding)
9     collection.add(
10         embeddings=embeddings,
11         ids=ids
12     )

```

Tiếp theo chúng ta khởi tạo một client cho cơ sở dữ liệu Chroma và tạo một collection mới với cấu hình sử dụng L2 để so sánh các embedding vector. Sau đó, gọi hàm add\_embedding để thêm các vector đặc trưng của ảnh vào collection này, qua đó tạo điều kiện thuận lợi cho việc truy vấn nhanh chóng và hiệu quả.

```

1 # Create a Chroma Client
2 chroma_client = chromadb.Client()
3 # Create a collection
4 l2_collection = chroma_client.get_or_create_collection(name="l2_collection",
5                                                       metadata={HNSW_SPACE: "l2"})
6 add_embedding(collection=l2_collection, files_path=files_path)

```

Hàm search được định nghĩa để thực hiện truy xuất các ảnh dựa trên embedding của ảnh truy vấn. Hàm này nhận đường dẫn của ảnh truy vấn, loại collection và số lượng kết quả trả về mong muốn, sau đó trả về danh sách các kết quả phù hợp.

```

1 def search(image_path, collection, n_results):
2     query_image = Image.open(image_path)
3     query_embedding = get_single_image_embedding(query_image)
4     results = collection.query(
5         query_embeddings=[query_embedding],

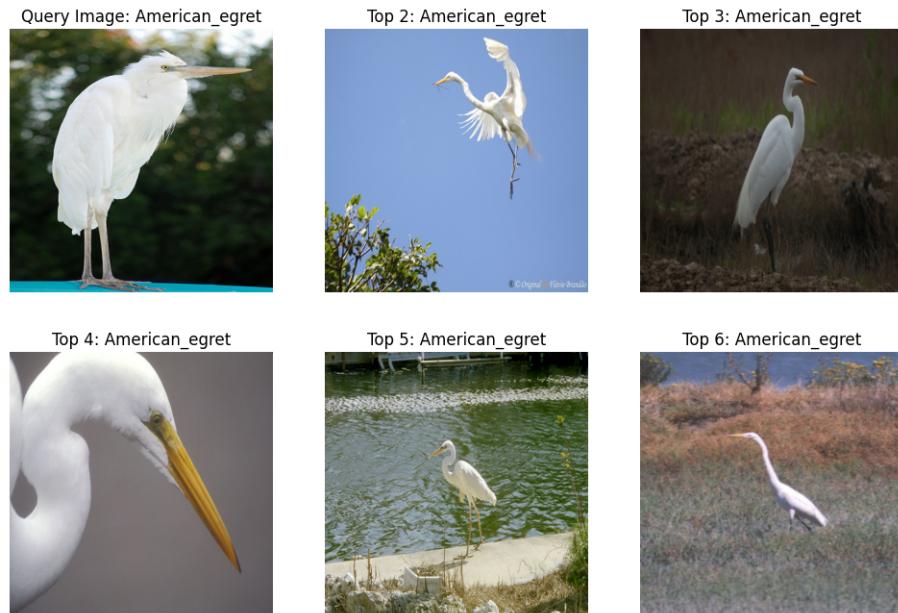
```

```

6     n_results=n_results # how many results to return
7
8     return results

1 test_path = f'{ROOT}/test'
2 test_files_path = get_files_path(path=test_path)
3 test_path = test_files_path[1]
4 l2_results = search(image_path=test_path, collection=l2_collection, n_results=5)
5 plot_results(image_path=test_path, files_path=files_path, results=l2_results)

```



Hình 20: Image retrieval dùng vector database với L2

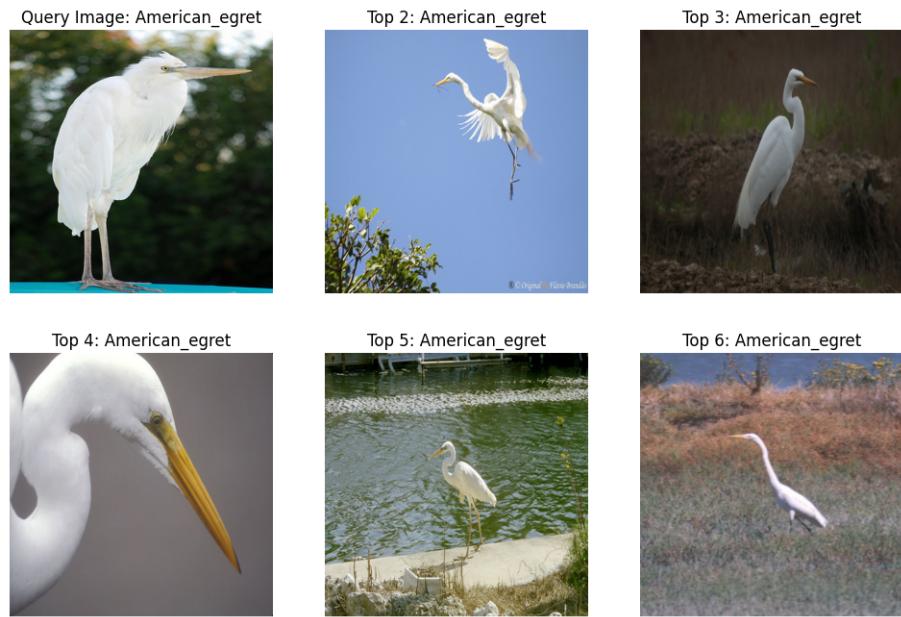
### Truy vấn ảnh với Cosine Similarity Collection

Tương tự như với L2 collection, đoạn code này khởi tạo một collection mới dựa trên khoảng cách cosine.

```

1 # Create a collection
2 cosine_collection = chroma_client.get_or_create_collection(name="Cosine_collection",
3                                                               metadata={HNSW_SPACE: "cosine"})
4 add_embedding(collection=cosine_collection, files_path=files_path)

```



Hình 21: Image retrieval dùng vector database với similarity score

### 3 Trắc Nghiệm

1. CLIP chủ yếu được sử dụng để làm gì trong bối cảnh tìm kiếm sự tương đồng giữa các hình ảnh với hình ảnh?
  - (a) Tăng cường chất lượng của hình ảnh
  - (b) Tạo chú thích hình ảnh
  - (c) Tìm những hình ảnh giống nhau dựa trên nội dung ngữ nghĩa
  - (d) Nén hình ảnh
2. Cho hai vector  $\mathbf{a} = [1, 2, 3]$  và  $\mathbf{b} = [4, 5, 6]$ , tính độ tương tự L1 (norm 1) giữa chúng.
  - (a) 9
  - (b) 12
  - (c) 15
  - (d) 18
3. Một trường hợp sử dụng phổ biến cho tìm kiếm sự tương đồng hình ảnh trong thương mại điện tử là gì?
  - (a) Dự đoán giá cổ phiếu
  - (b) Gợi ý sản phẩm dựa trên sự tương đồng hình ảnh
  - (c) Tính chi phí vận chuyển
  - (d) Tạo mô tả sản phẩm
4. Cho hai vector  $\mathbf{a} = [1, 2, 3]$  và  $\mathbf{b} = [4, 5, 6]$ , tính độ tương tự L2 (norm 2) giữa chúng.
  - (a)  $\sqrt{27}$

- (b)  $\sqrt{36}$
- (c)  $\sqrt{50}$
- (d)  $\sqrt{77}$

5. **Sự kết hợp của CLIP và ChromaDB làm tăng cường quá trình truy xuất hình ảnh như thế nào?**

- (a) Giảm kích thước vật lý của hình ảnh
- (b) Cho phép tìm kiếm dựa trên ngữ nghĩa thông qua so sánh vector
- (c) Dịch nội dung hình ảnh ra các ngôn ngữ khác nhau
- (d) Mã hóa dữ liệu hình ảnh

6. **Cho hai vector  $a = [1, 0, 1]$  và  $b = [1, 1, 0]$ , tính độ tương tự Cosine giữa chúng.**

- (a) 0
- (b)  $\frac{1}{2}$
- (c) 1
- (d)  $\frac{2}{3}$

7. **Trong bối cảnh nào, khoảng cách L1 thường được ưu tiên sử dụng để so sánh hình ảnh?**

- (a) Khi cần đo lường sự tương phản trực tiếp giữa các điểm ảnh
- (b) Khi các vector đặc trưng có độ dài không đồng đều
- (c) Khi các vector đặc trưng là đa chiều và phức tạp
- (d) Tất cả đều đúng

8. **Cho hai vector  $a = [2, 4]$  và  $b = [1, 2]$ , tính hệ số tương quan Pearson giữa chúng.**

- (a) 0.5
- (b) 1
- (c) -1
- (d) 0

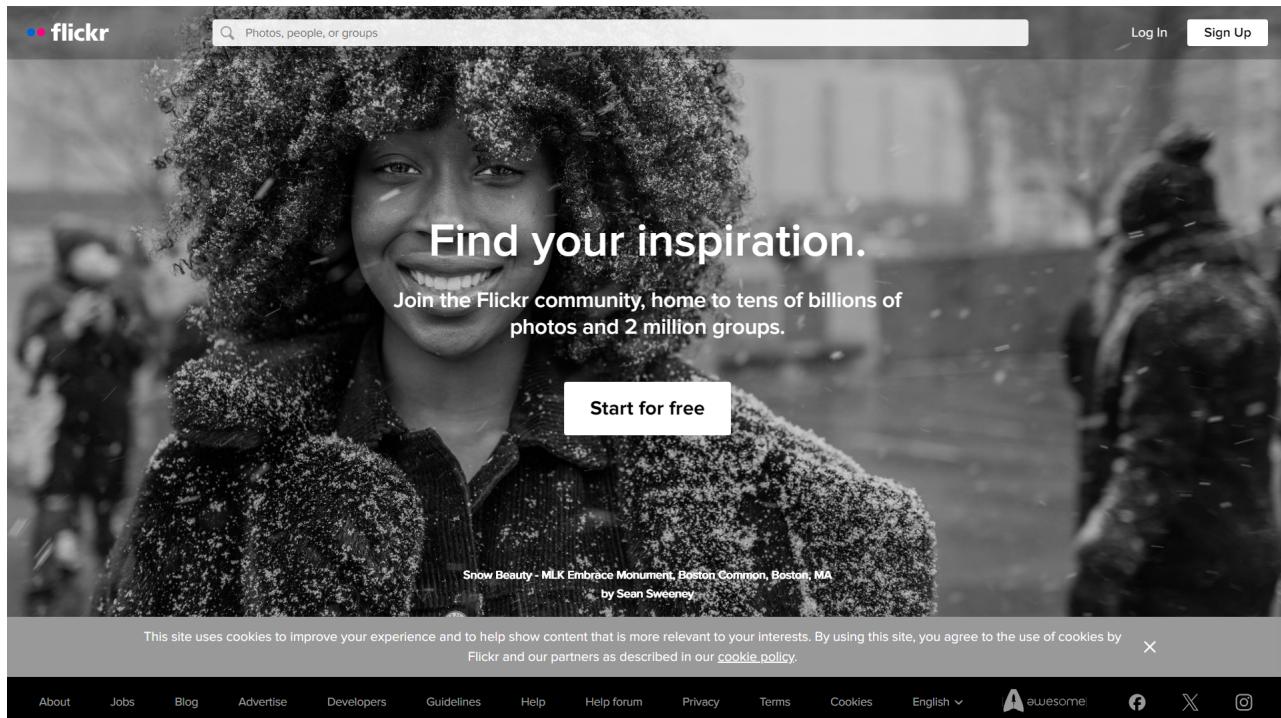
9. **Ưu điểm chính của độ tương đồng Cosine so với khoảng cách L1 trong việc so sánh hình ảnh là gì?**

- (a) Độ chính xác cao hơn khi xử lý các vector có giá trị âm
- (b) Không phụ thuộc vào độ dài của vector, chỉ quan tâm đến hướng của chúng
- (c) Tính toán nhanh hơn do ít dữ liệu hơn cần xử lý
- (d) Khả năng chống nhiễu tốt hơn trong dữ liệu

## 4 (Đọc thêm) Thu Thập và Xử Lý Dữ Liệu Để Xây Dựng Chương Trình Truy Vấn Ảnh Cá Nhân Hóa

Phần này tập trung vào việc thu thập và xử lý dữ liệu nhằm mục đích tạo ra một hệ thống truy vấn ảnh được cá nhân hóa, phù hợp với nhu cầu và sở thích riêng của từng người dùng.

Thu thập dữ liệu là tác vụ đầu tiên chúng ta cần làm để có thể xây dựng nền bộ dữ liệu ảnh dành cho bài toán Image Retrieval. Có khá nhiều cách để thu thập dữ liệu dạng ảnh như chụp bằng thiết bị ghi hình, tải từ các website "*Image Search Engine*", phổ biến như: Google Image Search, TinyEye, Bing Image Search, Yahoo Image Search, Yandex Image Search, Pinterest, Visual Search, Openverse, Flickr, Shutterstock, Pexels, ... Đối với dự án này, chúng ta sẽ thực hiện thu thập ảnh từ trang web flickr.com - một trang web tìm kiếm ảnh chất lượng và miễn phí.



Hình 22: Trang chủ flickr.com

Trước hết, chúng ta sẽ cài đặt, import các thư viện cần thiết cho dự án này:

- **BeautifulSoup**: Dùng để phân tích cú pháp HTML của trang web flickr.com.
- **Urllib**: Xử lý các vấn đề liên quan tới URL.
- **Selenium**: Dùng để tương tác với các HTML element của website flickr.com
- **Concurrent.futures**: Là Mô-đun dùng để xử lý đa luồng trong Python.
- **os**: Làm việc với đường dẫn ảnh, tổ chức folder.

```

1 !pip install tqdm
2
3 !apt-get update
4 !apt-get install -y wget

```

```

5 !pip install selenium
6 !apt-get install -y chromium-browser
7 !apt-get install -y chromium-chromedriver
8
9 from selenium import webdriver
10 from selenium.webdriver.common.by import By
11 from selenium.webdriver.support.ui import WebDriverWait
12 from selenium.webdriver.support import expected_conditions as EC
13
14 from bs4 import BeautifulSoup # For parsing HTML content
15 from urllib.parse import urljoin, urlparse # For handling URLs
16 import urllib.request # For making HTTP requests
17 import time # For handling time-related operations
18 import os # For interacting with the operating system (relate to dir, folder, file)
19 from tqdm import tqdm # For displaying progress bars (visualize progress)
20 import concurrent.futures # For multi-threading
21 import json # For writing to a text file
22 from PIL import Image # For handling images

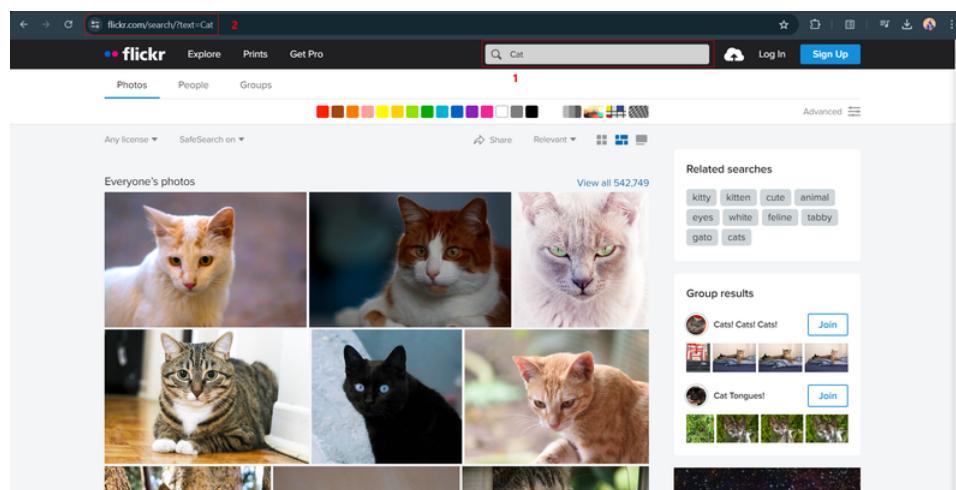
```

#### 4.0.1 Thu thập dữ liệu - Crawl URL từ Website

Mục tiêu:

- Hiểu về bộ cục của trang web flickr.com.
- Hiểu cách hoạt động cơ bản của website.
- Thiết kế class dùng để Crawl url từ website dựa vào các đặc điểm của html.
- Xử lý đa luồng khi crawl url từ website.

Trước tiên chúng ta sẽ quan sát bộ cục của trang web này trước khi thực hiện đi sâu phân tích về cấu trúc HTML của nó.



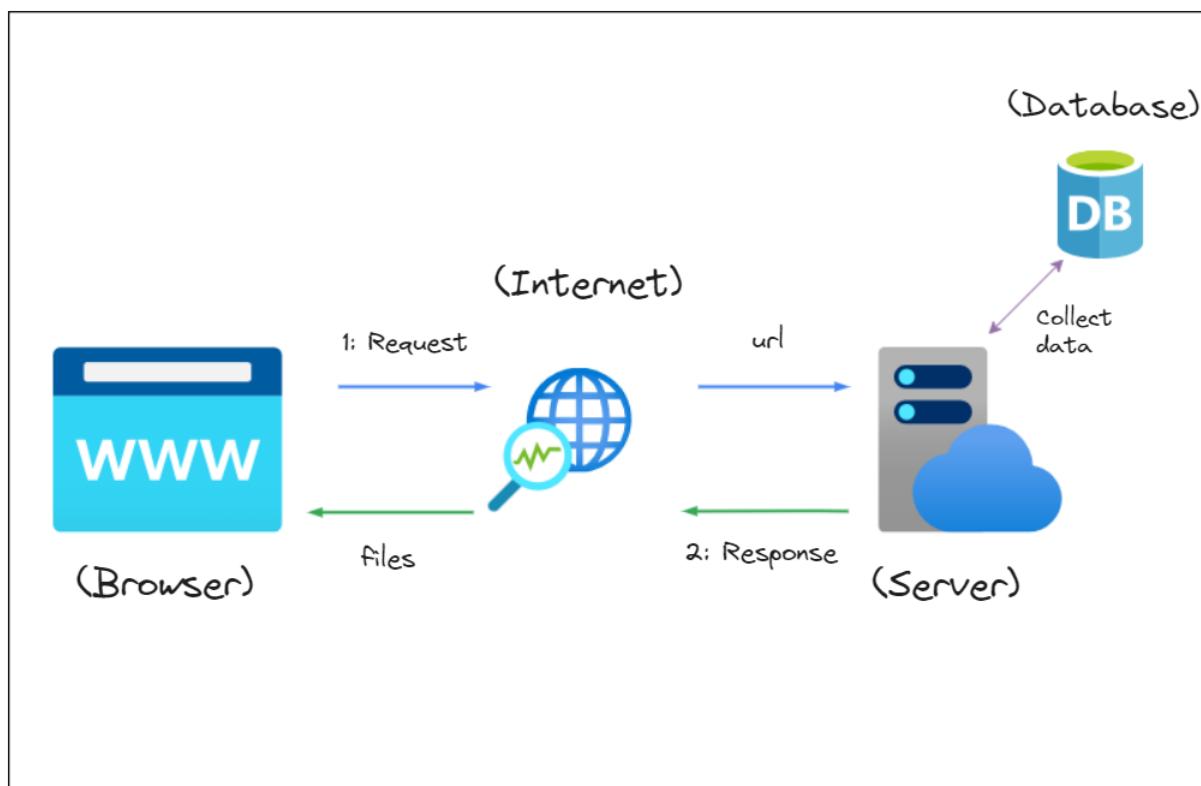
Hình 23: Bộ cục trang tìm kiếm

Có thể thấy flickr là trang web có bộ cục khá đơn giản, khi chúng ta nhập một từ khóa tìm kiếm vào ô search, bên dưới sẽ ngay lập tức hiển thị các hình ảnh chính xác. Và ở đây có một điểm quan trọng nhất cần lưu ý là đường link URL (2) của trang web. Dễ dàng nhận thấy được thay vì tìm kiếm bằng ô

search, chúng ta hoàn toàn có thể thay đổi từ khóa "Cat" trong đường dẫn bằng từ khóa mong muốn. Từ đây, chúng ta có thể liên hệ tìm kiếm với trang web flickr thông qua đường link:

1 [https://www.flickr.com/search/?text={search\\_term}](https://www.flickr.com/search/?text={search_term})

Từ đường dẫn URL, chúng ta có thể lấy được toàn bộ nội dung mà trang web hiển thị:



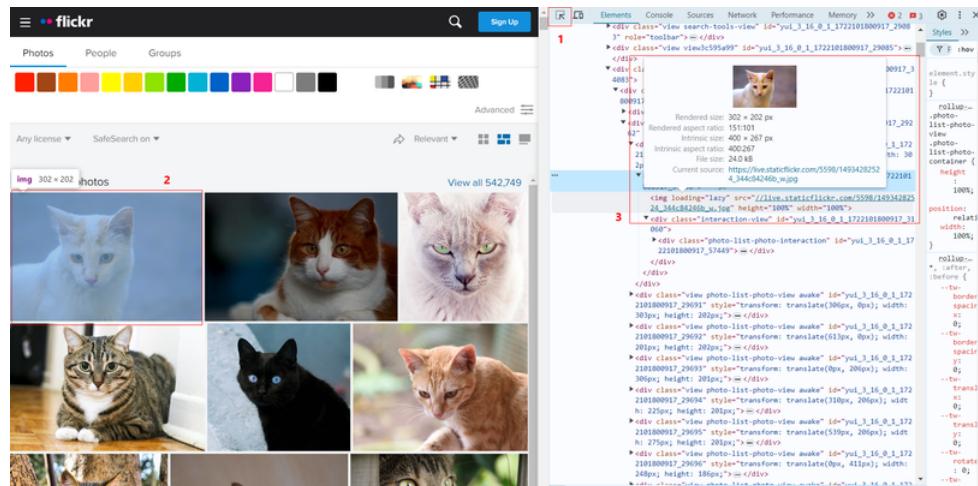
Hình 24: Cách hoạt động cơ bản của website

Khi chúng ta gõ một đường dẫn, hay click vào URL nào đó, browser (như Cốc cốc, chrome, brave) sẽ gửi một yêu cầu tới server, server sẽ đọc url, xử lý yêu cầu, truy xuất xuống database lấy những thông tin cần thiết, sau đó tổng hợp tất cả các file gửi trả lời cho browser. Trong các file đó, dĩ nhiên có file định dạng HTML, chính là file chứa cấu trúc của toàn bộ trang web. Sau khi browser nhận được phản hồi từ server, sẽ hiển thị tất cả thông tin dưới dạng một trang web, người dùng có thể dễ dàng tương tác, gửi các request khác tới server thông qua browser.

Chúng ta vừa có thể tìm kiếm nội dung bằng cách thay đổi search\_term trong đường link, vừa có thể lấy được nội dung (cấu trúc HTML) của trang đó thông qua việc gửi request tới server, như vậy, chúng ta chỉ cần xử lý thông tin ở trong file HTML nhận lại từ server để lấy ra thông tin về ảnh cần thu thập.

Để phân tích cấu trúc trang web thông qua HTML:

- **Bước 1:** Nhấn F12 để bật công cụ dev tool của browser.
- **Bước 2:** Click vào vị trí số 1 ở ảnh bên dưới, sau đó trở tới 1 bức ảnh bất kì (2).
- **Bước 3:** Quan sát cấu trúc HTML chứa đường link của bức ảnh (3).



Hình 25: Tìm hiểu cấu trúc trang web

( Các bức ảnh trên các trang web thường được lưu trữ trên các server riêng biệt. Khi bạn truy cập một trang web, trang web đó sẽ sử dụng URL của bức ảnh để gửi yêu cầu (request) đến các server chứa bức ảnh. Server sau đó sẽ gửi bức ảnh trở lại để hiển thị trên trang web. Nói cách khác, trang web chỉ lưu trữ đường dẫn đến bức ảnh, còn bức ảnh thực tế được lưu trữ và phục vụ từ server khác.)

Một cách khác để nhìn thấy rõ ràng hơn cấu trúc html của trang web, bấm tổ hợp phím Ctrl + U. Và như quan sát, tất cả các đường dẫn hình ảnh đều nằm trong khôi mã gồm thẻ <img/> như bên dưới:

```

</div>
</summary></div></section></div></nav></div></aside></div></div>



```

Hình 26: Mã nguồn html của trang web

Trước tiên, thực hiện khai báo lớp ImageScraper nhận vào các tham số như đường dẫn, số lượng ảnh tối đa cần tải mỗi lớp, số luồng hoạt động.

```

1 class UrlScraper:
2     # Constructor
3     def __init__(self, url_template, max_images=50, max_workers=4):
4         self.url_template = url_template # Link crawl
5         self.max_images = max_images # Max images
6         self.max_workers = max_workers # Thread

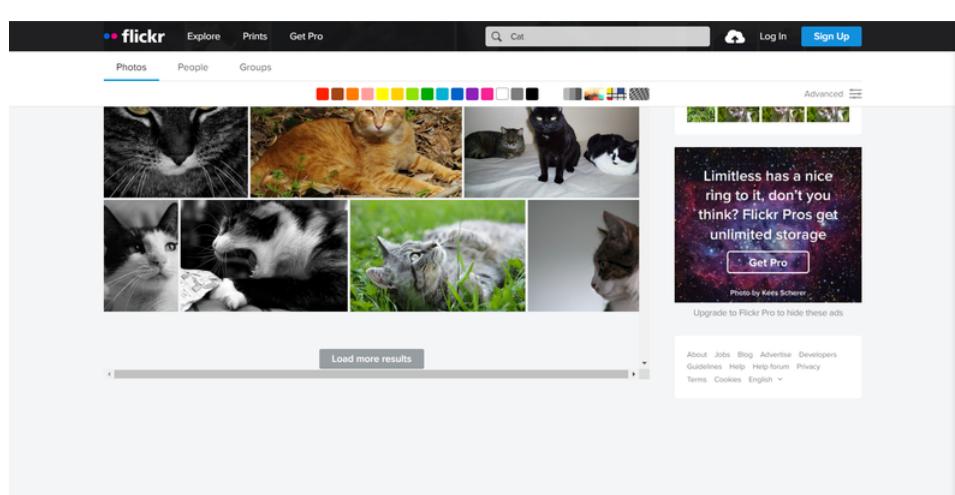
```

```

7     self.setup_environment() # Call for set up environment
8
9     # Set up environment for selenium
10    def setup_environment(self):
11        os.environ['PATH'] += ':/usr/lib/chromium-browser/'
12        os.environ['PATH'] += ':/usr/lib/chromium-browser/chromedriver/'

```

Ta quan sát được rằng, thẻ chứa url của bức ảnh là thẻ <img/> với thuộc tính loading="lazy". Từ đó, ta có thể sử dụng thư viện BeautifulSoup để bắt lấy tất cả các thẻ chứa url ảnh, Bên cạnh đó, Flickr là trang web hiển thị ảnh theo kiểu "lazy loading", tức nó sẽ không hiển thị hết tất cả ảnh một lần hay phân trang, mà nó sẽ hiển thị thêm ảnh nếu người dùng tiếp tục lướt xuống dưới. Đồng thời ở một mức nào đó, trang web sẽ yêu cầu người dùng bấm vào button "Load more results" để hiển thị thêm ảnh.



Hình 27: Tính chất lazy loading giúp trang web tối ưu hiệu năng hiển thị ảnh

Năm được đặc tính này, chúng ta sẽ sử dụng selenium để scroll trang web xuống dưới để nhận thêm nhiều nội dung HTML, đồng thời click vào button "Load more results" để lấy thêm nhiều link ảnh cho tới khi đủ thì dừng lại.

```

1 def get_url_images(self, term):
2     """
3         Crawl the urls of images by term
4
5     Parameters:
6         term (str): The name of animal, plant, scenery, furniture
7
8     Returns:
9         urls (list): List of urls of images
10    """
11
12    # Initialize Chrome driver
13    options = webdriver.ChromeOptions()
14    options.add_argument('--headless')
15    options.add_argument('--no-sandbox')
16    options.add_argument('--disable-dev-shm-usage')
17    driver = webdriver.Chrome(options=options)
18
19    url = self.url_template.format(search_term=term)

```

```
20     driver.get(url)
21
22     # Start crawl urls of image like brute force - the same mechanism with this but add
23     # some feature
24     urls = []
25     more_content_available = True
26
27     pbar = tqdm(total=self.max_images, desc=f"Fetching images for {term}") # Set up for
28     # visualize progress
29
30     while len(urls) < self.max_images and more_content_available:
31         soup = BeautifulSoup(driver.page_source, "html.parser")
32         img_tags = soup.find_all("img")
33
34         for img in img_tags:
35             if len(urls) >= self.max_images:
36                 break
37             if 'src' in img.attrs:
38                 href = img.attrs['src']
39                 img_path = urljoin(url, href)
40                 img_path = img_path.replace("_m.jpg", "_b.jpg").replace("_n.jpg", "_b.jpg")
41                 .replace("_w.jpg", "_b.jpg")
42                 if img_path == "https://combo.staticflickr.com/ap/build/images/getty/
43                   IStock_corporate_logo.svg":
44                     continue
45                 urls.append(img_path)
46                 pbar.update(1)
47
48     try:
49         load_more_button = WebDriverWait(driver, 10).until(
50             EC.element_to_be_clickable((By.XPATH, '//button[@id="'
51             'yui_3_16_0_1_1721642285931_28620"]')))
52     )
53     load_more_button.click()
54     time.sleep(2)
55
56 except:
57     driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
58     time.sleep(2)
59
60     new_soup = BeautifulSoup(driver.page_source, "html.parser")
61     new_img_tags = new_soup.find_all("img", loading="lazy")
62     if len(new_img_tags) == len(img_tags):
63         more_content_available = False
64         img_tags = new_img_tags
65
66     pbar.close()
67     driver.quit()
68     return urls
```

Đồng thời, sẽ rất lâu nếu chúng ta đợi một class tải xong url mới bắt đầu class số hai, vì thế ở đây khi crawl url, ta sẽ áp dụng xử lý đa luồng multi-threading để phân luồng tải, thực hiện tải url cho nhiều class cùng một lần và thực hiện liên tiếp, từ đó tốc độ crawl tăng lên đáng kể, giúp chúng ta tối ưu thời gian xong việc thu thập dữ liệu.

```

1 def scrape_urls(self, categories):
2     """
3         Call get_url_images method to get all urls of any object in categories\
4
5     Parameter:
6         categories (dictionary): the dict of all object we need to collect image with format
7             categories{"name_object": [value1, value2, ...]}
8
9     Returns:
10        all_urls (dictionary): Dictionary of urls of images
11        """
12        all_urls = {category: {} for category in categories}
13
14        # Handle multi-threading for efficient installation
15        with concurrent.futures.ThreadPoolExecutor(max_workers=self.max_workers) as executor:
16            future_to_term = {executor.submit(self.get_url_images, term): (category, term)
17                               for category, terms in categories.items() for term in terms}
18
19            for future in tqdm(concurrent.futures.as_completed(future_to_term), total=len(
20                future_to_term), desc="Overall Progress"):
21                category, term = future_to_term[future]
22                try:
23                    urls = future.result()
24                    all_urls[category][term] = urls
25                    print(f"\nNumber of images retrieved for {term}: {len(urls)}")
26                except Exception as exc:
27                    print(f"\n{term} generated an exception: {exc}")
28
29        return all_urls

```

Sau khi đã hoàn thành tải, chúng ta sẽ lưu lại các url thành file định dạng json, thể hiện phân cấp giống như dictionary các class cần tải (category):

```

1 def save_to_file(self, data, filename):
2     """
3         Save the data to a JSON file.
4
5     Parameters:
6         data (dict): The data to be saved.
7         filename (str): The name of the JSON file.
8
9     Returns:
10        None
11        """
12        with open(filename, 'w') as file:
13            json.dump(data, file, indent=4)
14            print(f"Data saved to {filename}")

```

Sau khi đã khai báo xong, thực hiện khởi tạo đối tượng từ class, đưa vào các tham số cần thiết. Gọi đến hàm `scrape_images` để tải tất cả các đường dẫn cần thiết về.

```

1 categories = {
2     "animal": ["Monkey", "Elephant", "cows", "Cat", "Dog", "bear", "fox", "Civet", "Pangolins"
3                 , "Rabbit", "Bats", "Whale", "Cock", "Owl", "flamingo", "Lizard", "Turtle", "Snake", "
4                 Frog", "Fish", "shrimp", "Crab", "Snail", "Coral", "Jellyfish", "Butterfly", "Flies",
5                 "Mosquito", "Ants", "Cockroaches", "Spider", "scorpion", "tiger", "bird", "horse", "
6                 pig", "Alligator", "Alpaca", "Anteater", "donkey", "Bee", "Buffalo", "Camel", "
7                 "]

```

```
3     "Caterpillar", "Cheetah", "Chicken", "Dragonfly", "Duck", "panda", "Giraffe"],  
4     "plant": ["Bamboo", "Apple", "Apricot", "Banana", "Bean", "Wildflower", "Flower", "  
5         Mushroom", "Weed", "Fern", "Reed", "Shrub", "Moss", "Grass", "Palmtree", "Corn", "  
6         Tulip", "Rose", "Clove", "Dogwood", "Durian", "Ferns", "Fig", "Flax", "Frangipani", "  
7         Lantana", "Hibiscus", "Bougainvillea", "Pea", "OrchidTree", "RangoonCreeper", "  
8         Jackfruit", "Cottonplant", "Corneliantree", "Coffeeplant", "Coconut", "wheat", "  
9         watermelon", "radish", "carrot"],  
10    "furniture": ["bed", "cabinet", "chair", "chests", "clock", "desks", "table", "Piano", "  
11        Bookcase", "Umbrella", "Clothes", "cart", "sofa", "ball", "spoon", "Bowl", "fridge", "  
12        pan", "book"],  
13    "scenery": ["Cliff", "Bay", "Coast", "Mountains", "Forests", "Waterbodies", "Lake", "  
14        desert", "farmland", "river", "hedges", "plain", "sky", "cave", "cloud", "flowergarden",  
15        "glacier", "grassland", "horizon", "lighthouse", "plateau", "savannah", "valley", "  
16        volcano", "waterfall"]  
17 }  
18  
19 urltopic = {"flickr": "https://www.flickr.com/search/?text={search_term}"}  
20 scraper = UrlScraper(url_template=urltopic["flickr"], max_images=20, max_workers=5)  
21 image_urls = scraper.scrape_urls(categories)  
22 scraper.save_to_file(image_urls, 'image_urls.json')
```

#### 4.0.2 Thu thập dữ liệu - Crawl ảnh từ URL

Mục tiêu:

- Đọc nội dung từ file json (file kết quả của bước đầu tiên)
- Tải ảnh theo các thư mục phân cấp
- Xử lý đa luồng và polite delay

Sau khi thực hiện crawl url của ảnh thành công, chúng ta sẽ nhận được kết quả là một file chứa các đường dẫn image\_urls.json như sau:

```
[{"animal": {"Cat": [{"https://live.staticflickr.com/5598/14934282524_344c84246b_b.jpg", "https://live.staticflickr.com/7697/17026317426_bb3acf19fb_b.jpg", "https://live.staticflickr.com/8750/16386660144_a6c4026657_b.jpg", "https://live.staticflickr.com/7073/7190755946_ea97e85765_b.jpg", "https://live.staticflickr.com/3940/15504684310_f555c88915_b.jpg", "https://live.staticflickr.com/7313/9775005856_9b5e0ebe16_b.jpg", "https://live.staticflickr.com/1729/41676479745_ae6d27ee9d_b.jpg", "https://live.staticflickr.com/1701/24811748270_3102fc52fb_b.jpg", "https://live.staticflickr.com/4733/27257168879_464200ea90_b.jpg", "https://live.staticflickr.com/8208/8216315457_28762c496d_b.jpg", "https://live.staticflickr.com/6100/6303228181_59371c29dc_b.jpg", "https://live.staticflickr.com/280/31389231292_e2444d0260_b.jpg", "https://live.staticflickr.com/1261/5110834170_0797f39278_b.jpg", "https://live.staticflickr.com/4150/5061790223_b6ca46a9b0_b.jpg", "https://live.staticflickr.com/4308/35910819741_f3a2f38b4a_b.jpg", "https://live.staticflickr.com/5757/30033063091_7705ba4380_b.jpg", "https://live.staticflickr.com/2947/32960031673_ed659a2198_b.jpg", "https://live.staticflickr.com/5141/5616147572_197d15f94d_b.jpg", "https://live.staticflickr.com/5345/17733589900_4b7055de52_b.jpg", "https://live.staticflickr.com/3107/2321136879_60075fbc4e_b.jpg"}, {"Dog": [{"https://live.staticflickr.com/7127/7012277475_7e126fd8b6_b.jpg", "https://live.staticflickr.com/4026/4489119695_87144ba60b_b.jpg", "https://live.staticflickr.com/65535/36216273621_3287933a7c_b.jpg", "..."}]}]
```

Hình 28: Cấu trúc lưu trữ các đường dẫn ảnh của file json

Trước tiên, khai báo lớp ImageDownloader với các tham số đầu vào như sau:

- json\_file: đường dẫn tới file json
  - download\_dir: folder chứa các ảnh sẽ download
  - max\_worker: số luồng xử lý khi tải ảnh
  - delay: thời gian nghỉ giữa các lần request
  - filename: lưu trữ đường dẫn của tất cả các ảnh

```
1 class ImageDownloader:
2     def __init__(self, json_file, download_dir='Dataset', max_workers=4, delay=1):
3         self.json_file = json_file # file containing URLs of images in JSON format
4         self.download_dir = download_dir # Folder name for storing images
5         self.max_workers = max_workers # Number of threads
6         self.delay = delay # Polite delay: when we send request too much to the server for
7             # downloading images without polite delay, it will crash or prevent your IP
8         self.filename = set() # To store filename directories
9         self.setup_directory() # Set up the folder structure
10
11     def setup_directory(self):
12         if not os.path.exists(self.download_dir):
13             os.makedirs(self.download_dir)
```

Khai báo hàm đọc file json, hàm kiểm tra xem url có đúng chuẩn không trước khi thực hiện request để tránh lỗi:

```

1 def read_json(self):
2     """
3         Read the JSON file and return the data.
4
5     Returns:
6     data (dict): The data read from the JSON file.
7     """
8     with open(self.json_file, 'r') as file:
9         data = json.load(file)
10    return data
11
12 def is_valid_url(self, url):
13     """
14         Check if the URL is valid.
15
16     Parameters:
17     url (str): The URL to be checked.
18
19     Returns:
20     bool: True if the URL is valid, False otherwise.
21     """
22     try:
23         with urllib.request.urlopen(url) as response:
24             if response.status == 200 and 'image' in response.info().get_content_type():
25                 return True
26             except Exception:
27                 return False

```

Thực hiện khai báo hàm tải một ảnh, ngoài việc request tới server để tải ảnh đó thông qua url, chúng ta còn cần đặt nó vào đúng thư mục và đặt tên file cho bức ảnh khi tải về. Tất cả các thao tác đó được thực hiện bằng các built-in function trong thư viện os.

```

1 def download_image(self, url, category, term, pbar):
2     """
3         Download the image from the given URL.
4
5     Parameters:
6     url (str): The URL of the image to be downloaded.
7     category (str): The category of the image.
8     term (str): The term or keyword associated with the image.
9     pbar (tqdm): The progress bar object.
10
11    Returns:
12    str: A message indicating the status of the download.
13    """
14    if not self.is_valid_url(url):
15        pbar.update(1)
16        return f"Invalid URL: {url}"
17
18    category_dir = os.path.join(self.download_dir, category)
19    if not os.path.exists(category_dir):
20        os.makedirs(category_dir)
21
22    term_dir = os.path.join(category_dir, term)
23    if not os.path.exists(term_dir):

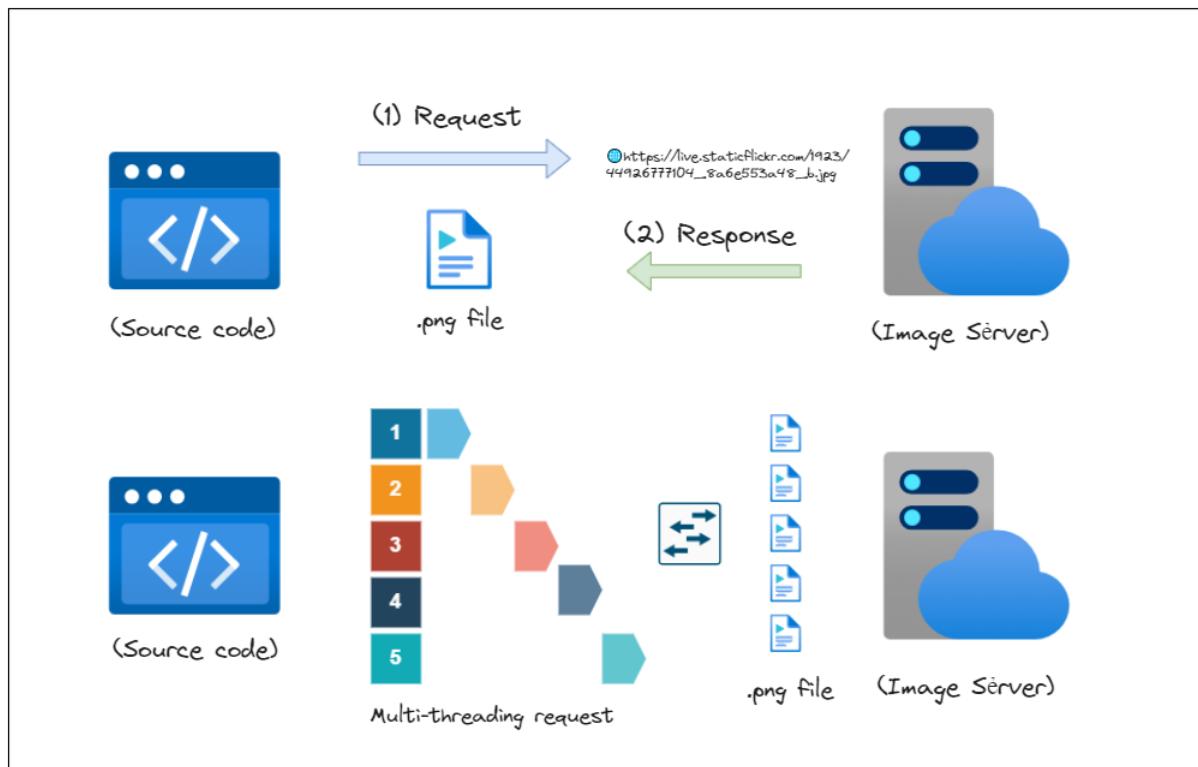
```

```
24     os.makedirs(term_dir)
25
26     filename = os.path.join(term_dir, os.path.basename(urlparse(url).path))
27
28     self.filename.add(filename) # Record the filename directory
29
30     try:
31         urllib.request.urlretrieve(url, filename)
32         pbar.update(1)
33         return f"Downloaded: {url}"
34     except Exception as e:
35         pbar.update(1)
36         return f"Failed to download {url}: {str(e)}"
```

Khi đã có hàm tải một ảnh, ta sẽ khai báo một hàm thực hiện tải nhiều ảnh bằng cách gọi lại hàm download\_image. Tuy nhiên, đối với việc tải ảnh chúng ta còn cần cân nhắc tới một số khía cạnh các như hiệu suất, request lịch sự tới server.

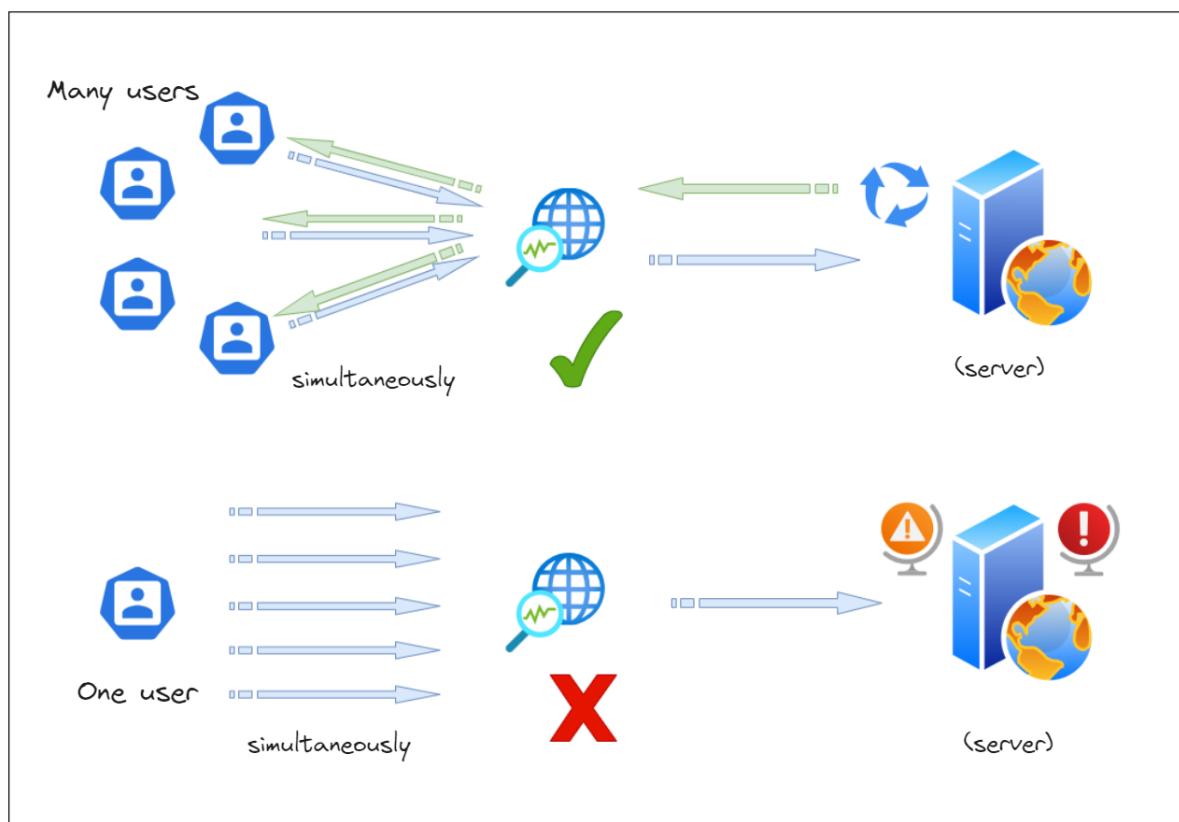
```
1  def download_images(self):
2      """
3          Download images from the JSON file.
4
5      Returns:
6      None
7      """
8
9      data = self.read_json()
10     download_tasks = []
11
12     total_images = sum(len(urls) for terms in data.values() for urls in terms.values())
13     with tqdm(total=total_images, desc="Downloading images") as pbar:
14         with concurrent.futures.ThreadPoolExecutor(max_workers=self.max_workers) as
15             executor:
16                 for category, terms in data.items():
17                     for term, urls in terms.items():
18                         for url in urls:
19                             download_tasks.append(executor.submit(self.download_image, url,
20                                         category, term, pbar))
21                         time.sleep(self.delay) # Polite delay
22
23             for future in concurrent.futures.as_completed(download_tasks):
24                 print(future.result())
25
26     self.export_filename()
27
28     def export_filename(self):
29         """
30             Export the filename directories to a text file.
31
32             Returns:
33             None
34             """
35
36             with open('filename.txt', 'w') as file:
37                 for filename in sorted(self.filename):
38                     file.write(f"{filename}\n")
```

Dầu tiên, chúng ta cần hiểu rõ về cơ chế đa luồng. Ở phần trước, chúng ta cũng đã áp dụng nó để tải các url ảnh. Thông thường, sau khi chúng ta request cho 1 url, chúng ta sẽ đợi phản hồi từ server, sau đó khi nhận được ảnh, chúng ta sẽ tiếp tục request tiếp theo. Việc này khiến cho quá trình tải rất chậm chạp và tốn rất nhiều thời gian. Trong khi đó nếu áp dụng multi-threading, chúng ta sẽ liên tục gửi các request tới server, mỗi lần gửi sẽ là n request liên tục nhau (không đồng thời) rồi tiếp tục quay lại thực hiện vòng lặp cho tới khi nhận đủ hình ảnh. Như vậy thời gian tải hình ảnh sẽ giảm đáng kể



Hình 29: Xử lí Multi-threading trong quá trình gửi request tới server

Việc gửi request tới server một cách ồ ạt tuy tốt cho thời gian của chúng ta thế nhưng lại là một hành động xấu đối với server-side. Đối với một server lớn, hàng ngày có cả triệu request từ nhiều user khác nhau, nó vẫn có thể xử lí một cách bình thường nhờ hệ thống điều phối hiệu quả. Tuy nhiên điểm khác biệt giữa request từ hàng triệu user trong một thời gian ngắn và cả ngàn request từ 1 user trong một thời gian ngắn là địa chỉ IP. Địa chỉ IP đại diện cho 1 client truy cập tới server, nếu chúng ta gửi liên tiếp request tới server, nó sẽ tự động phát hiện đây là hành vi bất thường và ngay lập tức chặn IP của chúng ta khiến quá trình tải lặp lục dừng lại. Trường hợp tệ hơn đối với các server nhỏ không có hệ thống tốt, server sẽ bị crash và sập. Vì vậy, chúng ta cần lịch sự trong việc request tới server thông qua polite delay, giữa các lần request, hãy giãn thời gian ra tầm 0.5 đến 1s để khiến cho server không bị chịu tải cao, đồng thời server không phát hiện bất thường và chặn IP của chúng ta.



Hình 30: Lỗi request liên tục tới server từ một địa chỉ IP

Khi đã khai báo xong tất cả các phương thức trong class ImageDownloader, thực hiện tạo đối tượng rồi gọi tới hàm download\_images để thực hiện tải ảnh.

```

1 downloader = ImageDownloader(json_file='image_urls.json', download_dir='Dataset', max_workers
2   =4, delay=1)
3 downloader.download_images()
4 downloader.export_filename()
```

#### 4.0.3 Xử lý dữ liệu - Làm sạch bộ dữ liệu

Thực hiện làm sạch bộ dữ liệu (xóa ảnh kích thước quá nhỏ, chuyển ảnh về 3 channel RGB, kiểm tra đúng định dạng file ảnh). Sau đó nén file và lưu vào drive.

```

1 from google.colab import drive
2 drive.mount('/content/drive/')
```

```

1 def check_and_preprocess_images(image_dir):
2     """
3     Check and preprocess images in the specified directory.
4
5     Parameters:
6     image_dir (str): The directory containing the images to be checked and preprocessed.
7
8     Returns:
```

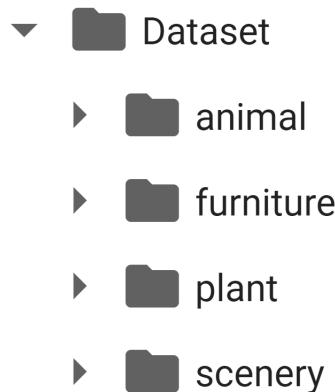
```

9     None
10    """
11
12    for root, _, files in os.walk(image_dir):
13        for file in files:
14            file_path = os.path.join(root, file)
15            try:
16                with Image.open(file_path) as img:
17                    # Check if image is smaller than 50x50 pixels
18                    if img.size[0] < 50 or img.size[1] < 50:
19                        os.remove(file_path)
20                        print(f"Deleted {file_path}: Image too small ({img.size[0]}x{img.size[1]})")
21                        continue
22
23                    # Convert non-RGB images to RGB
24                    if img.mode != 'RGB':
25                        img = img.convert('RGB')
26                        img.save(file_path)
27                        print(f"Converted {file_path} to RGB")
28
29            except Exception as e:
30                # If file is not an image, delete it
31                os.remove(file_path)
32                print(f"Deleted {file_path}: Not an image or corrupted file ({str(e)})")
33
34 check_and_preprocess_images('Dataset')

```

```
1 !zip -r /content/drive/MyDrive/Clean_Dataset.zip Dataset
```

Sau khi tải về, unzip ta sẽ được folder chứa data như sau:



Hình 31: Cấu trúc thư mục chưa dataset đã làm sạch

#### 4.0.4 Xử lí dữ liệu - Tổ chức cấu trúc folder

Để thuận tiện hơn cho việc thực hiện bài toán Image Retrieval, chúng ta sẽ gom tất cả các class lại, sau đó chia ra 2 tập train (19 ảnh mỗi class) và test (1 ảnh mỗi class).

Tải clean\_dataset về colab:

```
1 !gdown --id 1--6fe48D9ydnTpLV1GKKqJ0pqp0XB3z_
```

Unzip file clean\_dataset.zip:

```
1 !unzip Clean_Dataset
```

Thực hiện phân chia, tổ chức lại thư mục như đã đề cập ở trên:

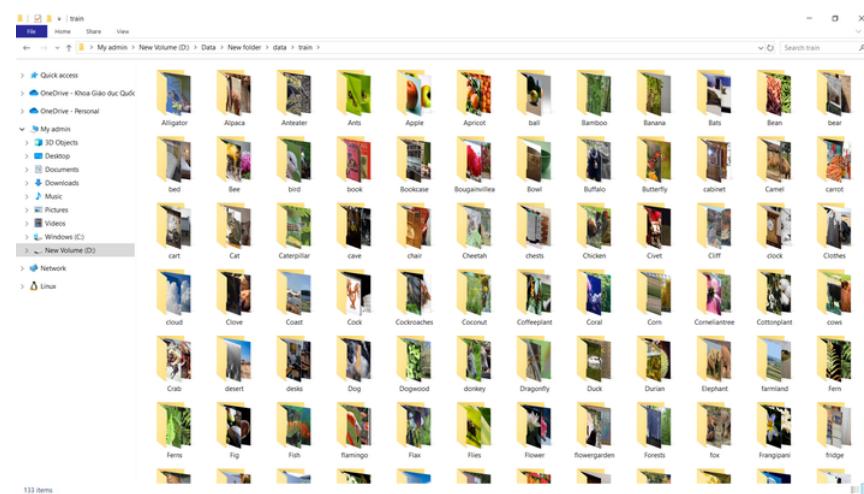
```
1 import os
2 import shutil
3 from collections import defaultdict
4
5 # Define the source and target directories
6 source_dir = "Dataset"
7 train_dir = "data/train"
8 test_dir = "data/test"
9
10 # Create the target directories if they don't exist
11 os.makedirs(train_dir, exist_ok=True)
12 os.makedirs(test_dir, exist_ok=True)
13
14 # Initialize a dictionary to hold file paths for each class
15 class_files = defaultdict(list)
16
17 # Read the file paths from the text file
18 with open('filename.txt', 'r') as file:
19     lines = file.readlines()
20     for line in lines:
21         line = line.strip()
22         if line:
23             # Extract the class name from the path
24             parts = line.split('/')
25             class_name = parts[2] # Structure Dataset/category/class/image.jpg
26             class_files[class_name].append(line)
27 class_files
28
29 # Move images to the train and test directories
30 for class_name, files in class_files.items():
31     # Create the train and test directories for the class
32     train_class_dir = os.path.join(train_dir, class_name)
33     test_class_dir = os.path.join(test_dir, class_name)
34     os.makedirs(train_class_dir, exist_ok=True)
35     os.makedirs(test_class_dir, exist_ok=True)
36
37     # Move 19 images to train and 1 image to test
38     for i, file_path in enumerate(files):
39         if i == 0:
40             shutil.copy(file_path, test_class_dir)
41         elif i < 20:
42             shutil.copy(file_path, train_class_dir)
43
44 print("Dataset organization complete!")
```

Chú ý filename.txt là file chứa đường dẫn của tất cả các ảnh đã được download ở phần trước. File name.txt sẽ được output ra sau khi khởi tạo đối tượng thuộc lớp ImageDownloader và gọi tới hàm export\_filename().

Nếu muốn tải folder data về lưu trữ, chúng ta có thể zip folder data về drive:

```
1 !zip -r /content/drive/MyDrive/data.zip data
```

Kết quả cuối cùng chúng ta được bộ dữ liệu ảnh với nhiều class như sau:



Hình 32: Dataset chứa tất cả các class được khai báo trong category