SUP'COM
Higher School of Communication of Tunis

Project Report

Content Based Image Retreival

Realized by:
Asma Abidalli
Sarra Hammami
INDP3 AIM

Supervised by:
Mr.Riadh Tebourbi

2023/2024

Contents

List of Figures

1 Introduction

1.1 Context

Image retrieval (IR) systems try to solve the problem of organizing and efficiently serving digital images in large collections. With the recent explosion of digital data, especially images, and the ever growing need for quick and accurate retrieval this field is developing very quickly.To achieve an image search, two different approaches are applied: text-based image retrieval and content-based image retrieval (CBIR) [1]. In the first approach, images are queried through textual metadata (description, labels, keywords..). This technique requires that all images in a database are annotated, which is not practical for large datasets and cannot provide users with accurate search results corresponding to their exact queries. For this reason, content-based image retrieval techniques are developed. CBIR engines allow searching for images using other images as queries, returning similar images as a result.

1.2 Project overview

Our project involves the creation of an advanced hybrid image search engine, capable of utilizing two distinct search methods. This system utilizes both the Flickr photos dataset, comprising a vast collection of over 2 million images and a local dataset featuring multiple classes for diverse search queries. The core of our solution includes the establishment of two separate indexes in Elasticsearch , each designed to handle a specific search method.The developed solution combines different technologies that involve data processing, machine learning, data indexing and querying and software development.

To understand the architecture we can brake it down into two processes: Data processing (from feature extraction to indexing) and image querying. Figure 1 resumes these two processes. The user can interact with the system via a web application.

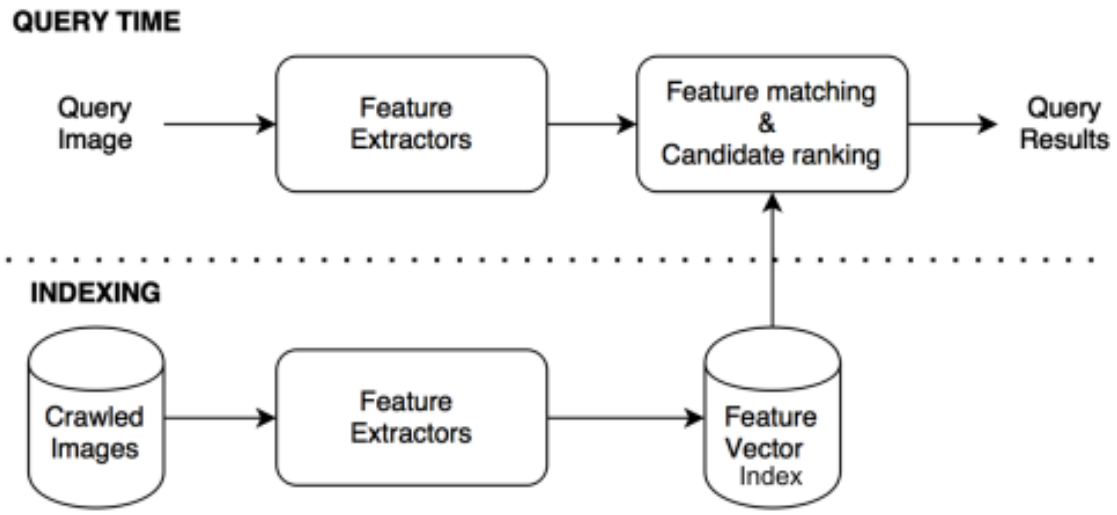

Figure 1: CBIR architecture

2 Dataset

2.1 Data description

We used 2 diffrent datasets for each search method.

- Yahoo Flickr [4] Creative dataset: The dataset includes 100 million images and videos along with associated metadata such as tags, descriptions, and user-generated annotations. Researchers and developers often use this dataset for tasks related to image recognition, object detection, and other computer vision applications due to its size and diversity.It is :
  - 16 GB of size
  - Geotagged (X, Y)
  - Keywords based
  - Accessible on the Flickr cloud

  The following figure shows an example of a simple line from the metadata csv file whci will impoted bu logstash into ES.

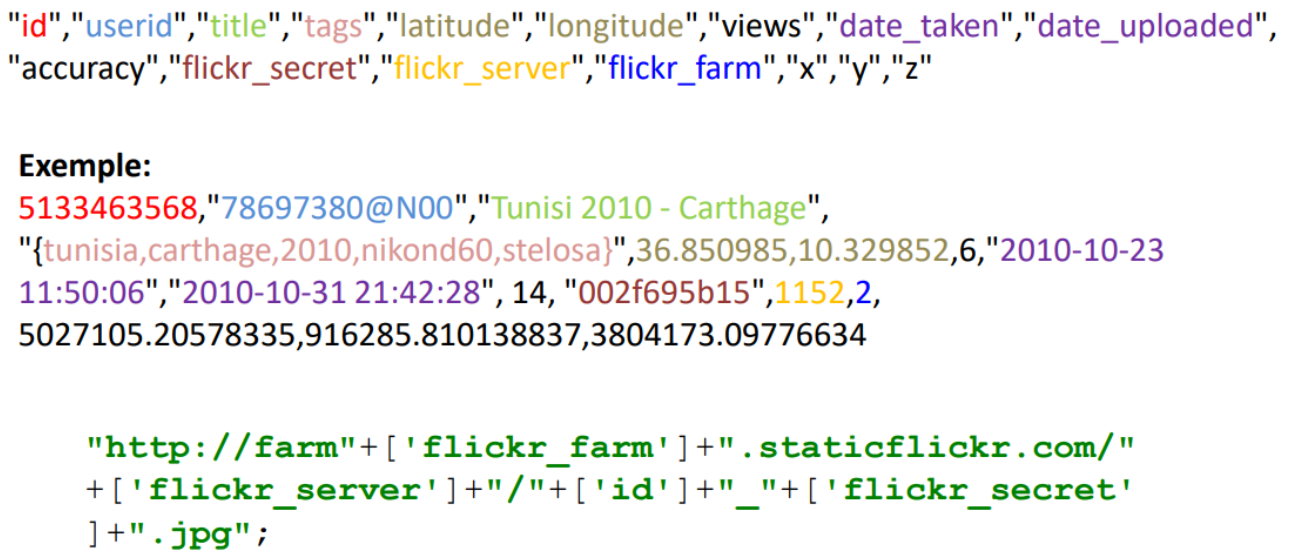

Figure 2: metadata.csv example

- idb images: a local dataset containing multiple objects classes used in searching by content similarities. The dataset contains approximately 900 images.

2.2 Data preparation

- We used the PyTorch DataLoader class ,a powerful utility used for efficiently loading and iterating over datasets in machine learning tasks. It provides an abstraction to manage large datasets and perform operations such as batching, shuffling, and parallel data loading.

We cutomized the Pytorch Dataloader class to apply different transformation on the dataset.

- These transformations include resizing the images to a consistent size using bilinear interpolation and converting them into tensors a numerical representation compatible with deep learning models (ResNet in our case) .Applied using the PyTorch transforms module. This step ensured that the images were in a format suitable for deep learning models.

```python
from PIL import Image
import pathlib

from torchvision import transforms
from torch.utils.data import Dataset
allowed_extensions = ['.jpg', '.png']
def get_transformation():
    transform = transforms.Compose(
        [
            transforms.Resize((224, 224), interpolation = transforms.InterpolationMode.BILINEAR),
            transforms.ToTensor(),
        ])

    return transform
```

Figure 3: Data preparation code

- The process of extracting features from all images is done with Pretrained RESNET50 [5] modal and imagenet weights .The result of the model is a feature vector with 2048 features.ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks.

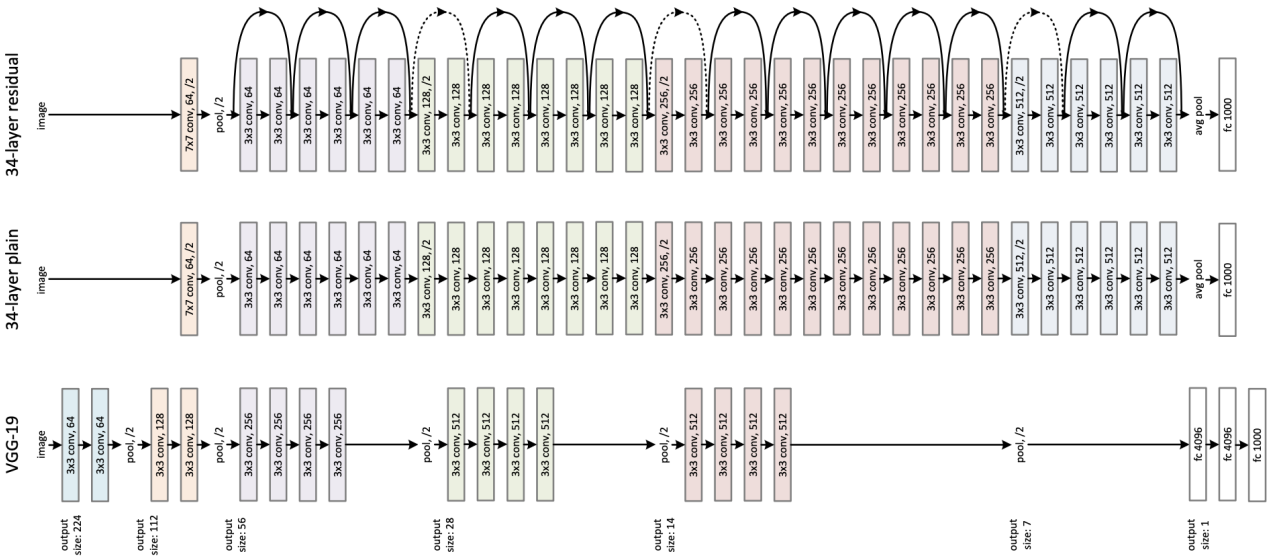

Figure 4: RESNET-50 architecture

3 Data Indexing

This project provides both content-based image retrieval and text-based image retrieval. For this, text and images need to be indexed separately. Elasticsearch , an open-source RESTful search and analytics engine, is the heart of this work. Images metadata (id,,title,tags ...) are indexed into an [2] elasticsearch index. This enables text search at very high speeds. The second step was to index the images for performing a content-based search. For that we used ES dense vectors to store feature vectors and apply

- Dense vectors refer to numerical vectors where most of the elements are non-zero or significant. These vectors are typically used in machine learning and similarity search tasks where efficient storage and retrieval of vector data are crucial.
- We used L2 norm as vector similarity metric to use in kNN search. Documents are ranked by their vector field's similarity to the query vector

3.1 Indexing process

1. First we have to index the text data , for that we used elasticsearch python client .and Logstash to push the metadata into es created index.

```
from elasticsearch import Elasticsearch, exceptions
# Initialize Elasticsearch client
es = Elasticsearch([{'host': 'localhost', 'port': 9200, 'scheme': 'http'}])
# Define your index mapping
index_mapping = {
    "mappings": {
        "properties": {
            "id": {"type": "text", "index": True},
            "userid": {"type": "text", "index": True},
           "title": {"type": "text","index" : true},
          "tags": {"type": "text","index" : true},
          "latitude": {"type": "double"},
          "longitude": {"type": "double"},
          "views": {"type": "integer"},
          "date_taken": {"type": "date","format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd"},
          "date_uploaded": {"type": "date","format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd"},
          "accuracy": {"type": "short"},
         "flickr_secret": {"type": "keyword","index" : true},
         "flickr_server": {"type": "keyword","index" : true},
         "flickr_farm": {"type": "keyword","index" : true},
         "x": {"type": "keyword","index" : true},
         "y": {"type": "keyword","index" : true},
          "z": {"type": "keyword","index" : true},
            "location": {"type": "geo_point"}
        }
    }
}
```

Figure 5: Index settings and creation

2. For indexing the second index we used elasticsearch Bulk Api . Using the Bulk API, we can send a single request containing multiple action and document pairs. Each action specifies the operation type (index, create, update, or delete) and the document data. The Bulk API processes these actions in a single batch, significantly improving the indexing performance, especially for large datasets .
   The figure Below shows the process of indexing image's feature vector along with its path.

```python
# Initialize Elasticsearch client
es = Elasticsearch([{'host': 'localhost', 'port': 9200,"scheme": "http"}])

# Initialize MyResnet model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
restnet_model = MyResnet50(device=device)

# Initialize MyDataLoader
image_root = 'D:/AIM-tbourbi/projetCBIR/training_set'
data_loader = MyDataLoader(image_root)
#print(len(data_loader))

# Prepare actions for bulk indexing
actions = []
for idx in range(len(data_loader)):
    image, image_path = data_loader[idx]
    image = image.unsqueeze(0).to(device)
    feature = restnet_model.extract_features(image)
    image_path_str = str(image_path).replace("\\", "/").strip('/')

    # Prepare document for Elasticsearch
    doc = {
        '_op_type': 'index',
        '_index': 'cbirproject',
        '_id': idx,
        '_source': {
            'image_path': image_path_str,
            'image_vector': feature
        }
    }
    actions.append(doc)

# Bulk index documents
try:
    success, _ = bulk(es, actions=actions, raise_on_error=True)
    print(f"Successfully indexed {success} documents.")
except Exception as e:
    print(f"Failed to index documents: {e}")
```

Figure 6: Index settings and creation

4 API Development

4.1 ElasticSearch client

In our project, we have used the Elasticsearch Python client to create a customized search system. This system incorporates features like fuzzy search and searches based on images and tags. To get started, we set up an Elasticsearch instance, and then we used the built-in search function by specifying the index name and the query body.

4.2 Text Based Search

With text search, we have the ability to find images in the index by conducting searches across multiple fields, including titles, and tags. This versatile search approach enhances the user's ability to discover images that are contextually relevant to their query. To illustrate this functionality, the following code snippet provides the signature of the text search function, offering a glimpse into how the search is conducted:

```
def search():
    if request.method == 'POST':
        # Check if the search query is in the form data
        if 'search_query' in request.form:
            search_term = request.form['search_query']
            # Execute Elasticsearch search query for text search
            res = es.search(
                index="flickerphotos",
                size=30,
                body={
                    "query": {
                        "multi_match": {
                            "query": search_term,
                            "fields": ["url", "title", "tags"],
                            "fuzziness": "AUTO",
                            "prefix_length": 1
                        }
                    }
                }
            )
            # Render the template with text search results
            return render_template('index.html', res=res)

    # Handle other cases (GET request, no search query, etc.)
    return render_template('index.html')
```

Figure 7: Text Based Search Function

4.3 Image Based Search

In the context of our content-based image retrieval, we have implemented a "search by image" feature, allowing users to locate images that share visual characteristics with a provided image based on its feature vector. This functionality is particularly valuable for content-based image retrieval, where the goal is to find images not by textual descriptions or tags, but by their actual content.

The central component of this feature is the 'search_similar_images' function, which accepts a

feature vector and an index name as inputs. Within this function, we construct an Elasticsearch query to assess the similarity between the feature vector of the query image and those within the index, employing the L2-norm (Euclidean distance) as the metric. Upon executing the query, the code collects and returns a list of image paths representing visually similar images to the query image.

```python
def search_similar_images(feature_vector, index_name='cbirproject'):
    # Construct Elasticsearch query to calculate Euclidean distance and sort by it
    search_body = {
        "query": {
            "script_score": {
                "query": {
                    "match_all": {}
                },
                "script": {
                    "source": "1 / (1 + l2norm(params.query_vector, 'image_vector'))",
                    "params": {
                     "query_vector": feature_vector
                    }
                }
            }
        }
    }

    # Execute the Elasticsearch search query
    #print("Elasticsearch Query Body:", search_body)
    response = es.search(index=index_name, body=search_body)

    # Extract similar images from the search results

    similar_images = []
    for hit in response['hits']['hits']:
        similar_images.append(hit['_source']['image_path'])

    return similar_images
```

Figure 8: Image Based Search Function

5 Web App Development

To enhance the user experience with the search engine, we've developed a web application using Flask [3], coupled with HTML and CSS for creating a seamless and user-friendly interface. Flask , known for its flexibility, has enabled us to rapidly design and implement a web-based frontend that allows users to interact with the content-based image retrieval system effortlessly. This web application complements our project by providing an intuitive gateway for users to access and utilize the search engine's capabilities, ensuring a smooth and enjoyable user experience.

Figure 9: Search Engine UI

6 Results

In this part, we present two examples of search results; the first one is image based research and the other one is for text.

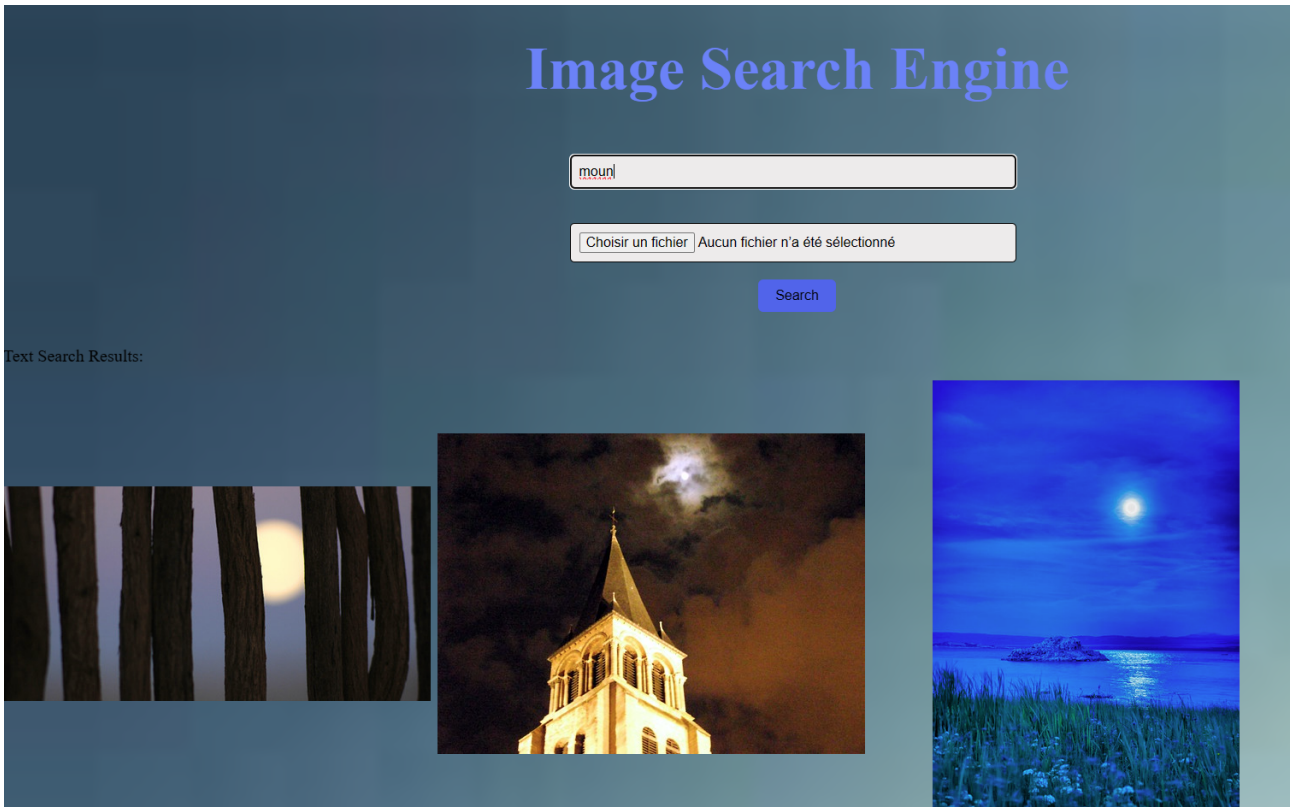

Figure 10: Image based search result

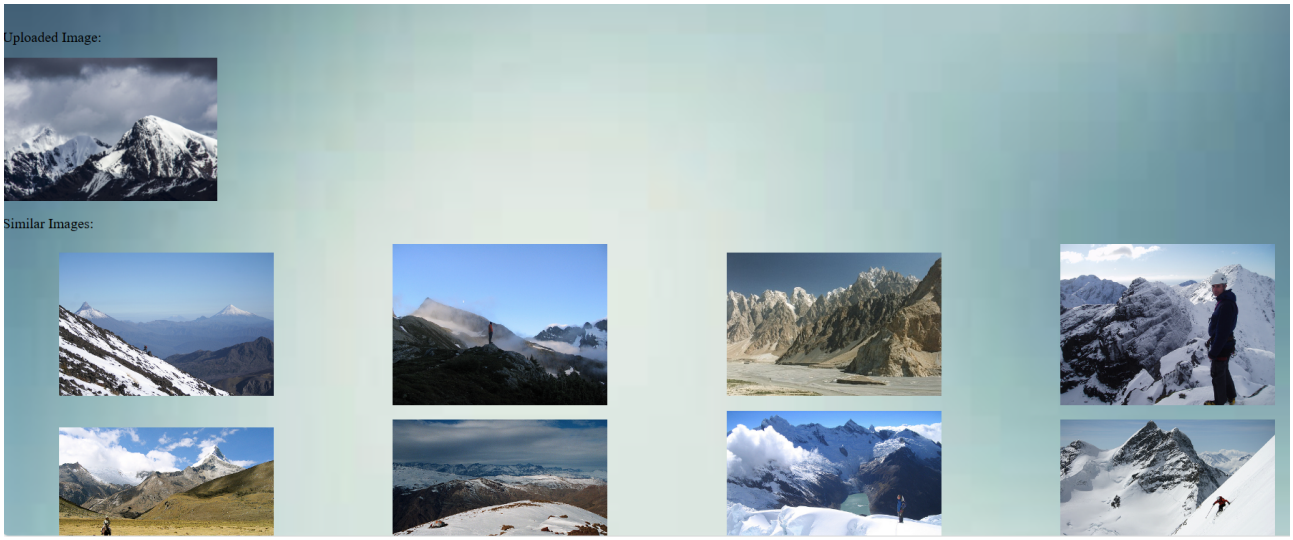

Figure 11: Text based search result

7 Conclusion

This project offers an integrated solution for image retrieval, allowing users to explore images based on visual content and text descriptions. It covers various aspects, from data preparation to a user-friendly interface, enhancing the search experience.

In this project, we have leveraged ResNet and Elasticsearch to craft a robust content-based image retrieval system. Our journey encompassed multiple stages, starting with data preparation and the extraction of image features using ResNet's deep convolutional neural network architecture. The feature vectors obtained not only accurately represent the visual content of images but also serve to enhance both storage efficiency and search response times.

We have indexed data in Elasticsearch, maintaining two separate indexes. One index contained image links and features, while the other focused on metadata. This division allowed users to perform versatile text-based searches through the Elasticsearch Search API, facilitating image discovery based on textual queries. Furthermore, the system supported image-based searches, aligning with image content. This dual-index approach enhanced the overall effectiveness of our image retrieval system, catering to both text and content-based search requirements.

References

[1] *Content based image retreival*. URL: https://www.baeldung.com/cs/cbir-tbir.

[2] *Elasticsearch*. URL: https://www.elastic.co/fr/elasticsearch/.

[3] *Flask*. URL: https://flask.palletsprojects.com/en/3.0.x/.

[4] *Flickr yahoo dataset*. URL: https://www.flickr.com/photos/tags/yahoo/.

[5] *ResNet-50*. URL: https://datagen.tech/guides/computer-vision/resnet-50/.