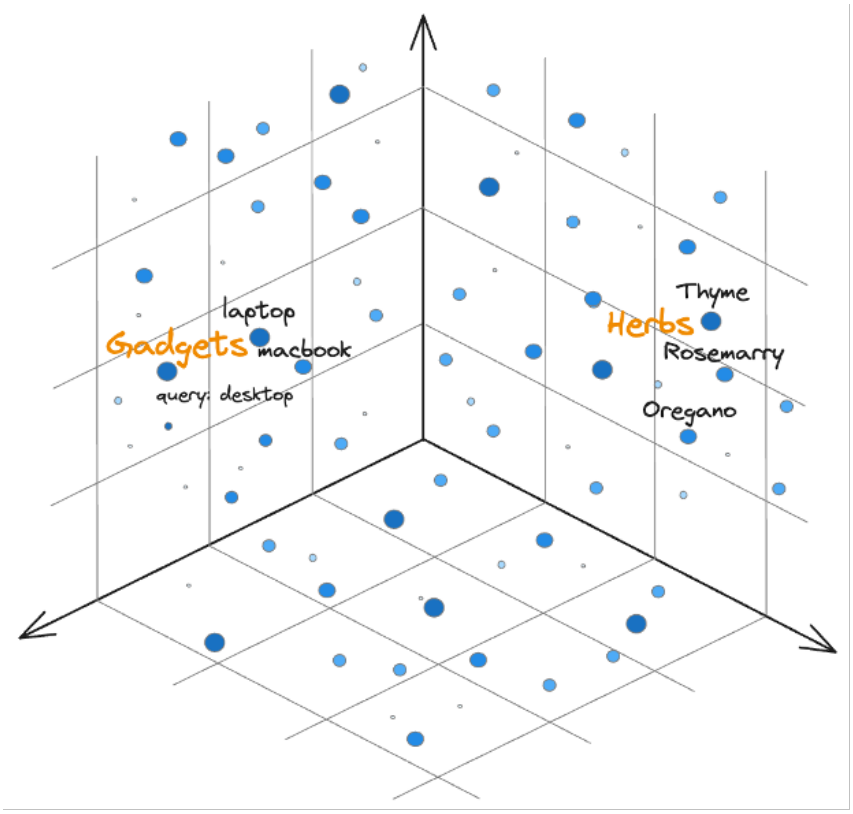
Multimodal Image Search Engine

Content-Based and Text-Based Image Retrieval System

Author: Marwen Bellili



Contents

1 Introduction

1.1 Context

With the exponential growth of digital image data, efficient image retrieval systems have become essential. Traditional metadata-based approaches require costly manual annotation and fail to capture visual semantics. Content-Based Image Retrieval (CBIR) addresses these limitations by analyzing intrinsic visual properties of images. This project implements a hybrid multimodal approach combining both text-based tag search and visual similarity search using deep learning features.

The emergence of deep learning has revolutionized computer vision, enabling extraction of high-level semantic features. This project leverages transfer learning with VGG16 to provide robust image retrieval capabilities based on semantic visual understanding.

1.2 Project Overview

This project develops a comprehensive image search system featuring:

- **Multimodal search capabilities**: Text-based tag search and image similarity search
- **Deep learning features**: VGG16 4096-dimensional features for semantic understanding
- **Scalable indexing**: Elasticsearch with native k-NN vector search
- **Web interface**: Streamlit application providing direct integration with Elasticsearch
- **Extensible design**: Modular architecture for easy maintenance

The system architecture integrates feature extraction, Elasticsearch indexing, and user interface within a unified Streamlit application, ensuring simplicity and ease of deployment.

2 Datasets

2.1 Image Collection Structure

The system uses the MIRFLICKR-1M dataset, a collection of 1 million images sourced from Flickr. Images are stored locally and accessed directly by the Streamlit application. The system supports standard image formats including JPEG, PNG, BMP, and TIFF, and automatically handles format conversion during processing.

2.1.1 Dataset Characteristics

The MIRFLICKR-1M dataset has the following properties:

- **Format**: RGB color images in standard formats
- **Size**: Variable dimensions (automatically resized during feature extraction)

- **Content**: General-purpose images covering objects, scenes, textures, and abstract imagery
- **Metadata**: Each image includes user-provided tags for text-based retrieval

2.2 Data Organization

Images follow a simple file-based organization structure:

- Each image is uniquely identified by its filename stem (without extension)
- Images are stored in a local directory accessible by the application
- Feature vectors and metadata are stored in an Elasticsearch index
- No external database is required for image storage

The indexing pipeline processes all images in the designated directory, extracting VGG16 deep features for visual similarity search.

3 Feature Extraction and Indexing

3.1 VGG16 Deep Features

The **VGG16** model provides high-level semantic representations learned via transfer learning from the **ImageNet** dataset. Its architecture consists of 16 weight layers, including convolutional layers with small 3×3 filters grouped into five convolutional blocks, followed by three fully connected (dense) layers.

3.1.1 Feature Extraction Process

In this project, features are extracted from the `fc1` layer of VGG16, which outputs a compact and discriminative 4096-dimensional feature vector. The extraction pipeline follows these steps:

1. Input images are preprocessed and resized to 224×224 pixels.
2. Images are normalized according to ImageNet mean and standard deviation.
3. A forward pass through the VGG16 network is performed up to the `fc1` layer.
4. The output of this layer (shape: 1×4096) is flattened to obtain the final feature vector.
5. The resulting 4096-dimensional feature vector is indexed in **Elasticsearch** for similarity-based image retrieval.

3.1.2 Advantages of 4096-Dimensional Features

Using the 4096-dimensional representation offers a good balance between expressiveness and efficiency:

- **Compact representation**: Reduces storage requirements compared to full convolutional tensors.
- **High discriminative power**: Dense layer features capture semantic content of the entire image.
- **Efficient similarity search**: Suitable for real-time retrieval using cosine similarity or k-NN.
- **Transfer learning robustness**: Pre-trained on ImageNet, the features generalize well to new datasets.

Technical characteristics:

- Feature dimension: 4096 values per image.
- Extracted layer: `fc1` (first fully connected layer).
- Average storage: Approximately 16 KB per feature vector (float32 representation).
- Recommended hardware: GPU acceleration for batch feature extraction.

3.2 Elasticsearch Index Configuration

3.2.1 Dense Vector Storage

The full-dimensional VGG16 features are stored as dense vectors in Elasticsearch using the `dense_vector` field type. This enables efficient similarity computation directly within the search engine.

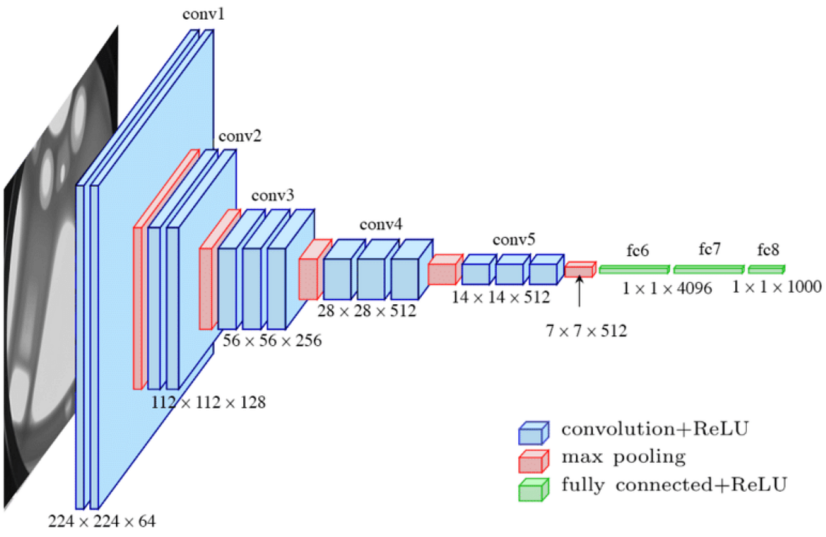


Figure 1: architecturevgg16.

3.2.2 k-Nearest Neighbors (k-NN) Search

k-NN retrieves the k most similar images to a query vector. Elasticsearch 8.x/9.x uses the HNSW (Hierarchical Navigable Small World) algorithm to perform approximate k-NN search, enabling fast retrieval over millions of images with high recall.

The HNSW algorithm creates a multi-layer graph structure that allows logarithmic search complexity, making it practical for high-dimensional vectors like our 25,088-dimensional VGG16 features.

3.2.3 Cosine Similarity

Cosine similarity measures the angle between two vectors, producing a similarity score between -1 and 1. For normalized feature vectors, cosine similarity effectively captures visual similarity irrespective of magnitude differences, focusing on the directional relationship between feature vectors.

The cosine similarity formula:

$$\text{similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \times ||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

3.2.4 Index Mapping Structure

The Elasticsearch index mapping includes the following fields:

- **image_id**: Keyword, unique identifier
- **image_path**: Keyword, file location
- **tags**: Keyword array for text search with fuzzy matching
- **vgg16_vector**: Dense vector, 4096 dimensions, cosine similarity

This unified mapping supports simultaneous text-based and visual similarity search, enabling efficient retrieval from large-scale image collections.

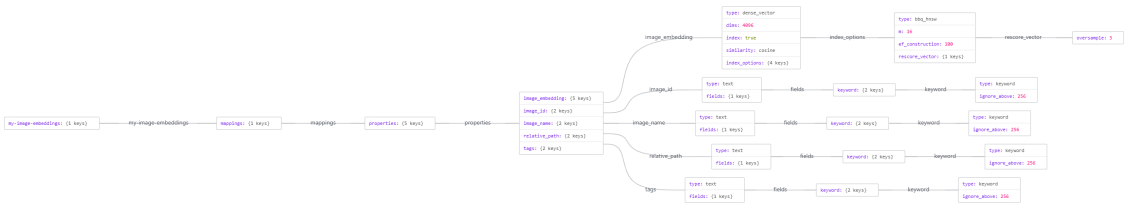


Figure 2: mapping elasticsearch.

4 System Architecture and Implementation

4.1 Architecture Overview

The application is built entirely with Streamlit, providing a unified framework that handles:

- **User Interface**: Interactive web components for search and display
- **Business Logic**: Feature extraction and search coordination
- **Data Layer**: Direct Elasticsearch client integration
- **Image Processing**: VGG16 feature extraction pipeline

This streamlined architecture eliminates the need for a separate API layer, reducing complexity while maintaining full functionality.

4.2 Streamlit Application Structure

4.2.1 Core Components

The Streamlit application consists of the following modules:

- **Feature Extractor**: Handles VGG16 model loading and feature computation
- **Elasticsearch Client**: Manages connections and query execution
- **Search Interface**: Provides text and image search functionality
- **Results Display**: Renders search results with images and metadata

4.2.2 Text-Based Search

Text queries are executed using Elasticsearch's `match` query on the `tags` field. The implementation:

1. User enters a text query in the Streamlit text input
2. Query is sent directly to Elasticsearch with fuzzy matching enabled
3. Results are ranked using the BM25 algorithm
4. Top-k results are retrieved and displayed

Fuzzy matching ensures typo-tolerant search, improving user experience.

4.2.3 Image-Based Search

Visual search leverages Elasticsearch's k-NN query with the following process:

1. User uploads an image through Streamlit's file uploader
2. Image is loaded and preprocessed for VGG16
3. Full 25,088-dimensional feature vector is extracted
4. k-NN query is executed against the Elasticsearch index
5. Cosine similarity is used to rank results
6. Top-k most similar images are returned with similarity scores

Query parameters:

- **field**: `vgg16_vector`
- **query_vector**: 4096-dimensional feature vector from uploaded image
- **k**: Number of nearest neighbors to return (user-configurable)
- **num_candidates**: Search quality parameter (typically $k \times 2$)

4.3 User Interface Features

4.3.1 Search Interface

The application provides an intuitive interface with:

- **Search mode selector**: Radio buttons or tabs to switch between text and image search
- **Text search**: Input field with configurable result count slider
- **Image search**: File uploader with image preview and k parameter configuration
- **Search button**: Triggers the search operation
- **Real-time feedback**: Loading spinners and progress indicators

4.3.2 Results Display

Search results are presented in a clean, organized layout:

- Grid layout showing image thumbnails
- Image ID and file path information
- Similarity scores (cosine similarity for image search, BM25 for text search)
- Associated tags displayed as colored badges or pills
- Expandable sections for detailed metadata
- Pagination for large result sets

4.3.3 Additional Features

- Session state management for persistent UI state
- Caching of VGG16 model to avoid reloading
- Error handling with user-friendly messages
- Responsive design adapting to different screen sizes
- Download options for search results

5 Results and Evaluation

5.1 Search Results Visualization

The system demonstrates effective retrieval capabilities for both search modalities. Figure 3 shows results from a text-based query, where images are ranked by tag relevance using BM25 scoring. Figure 4 presents results from an image-based similarity search using VGG16 features and cosine similarity.

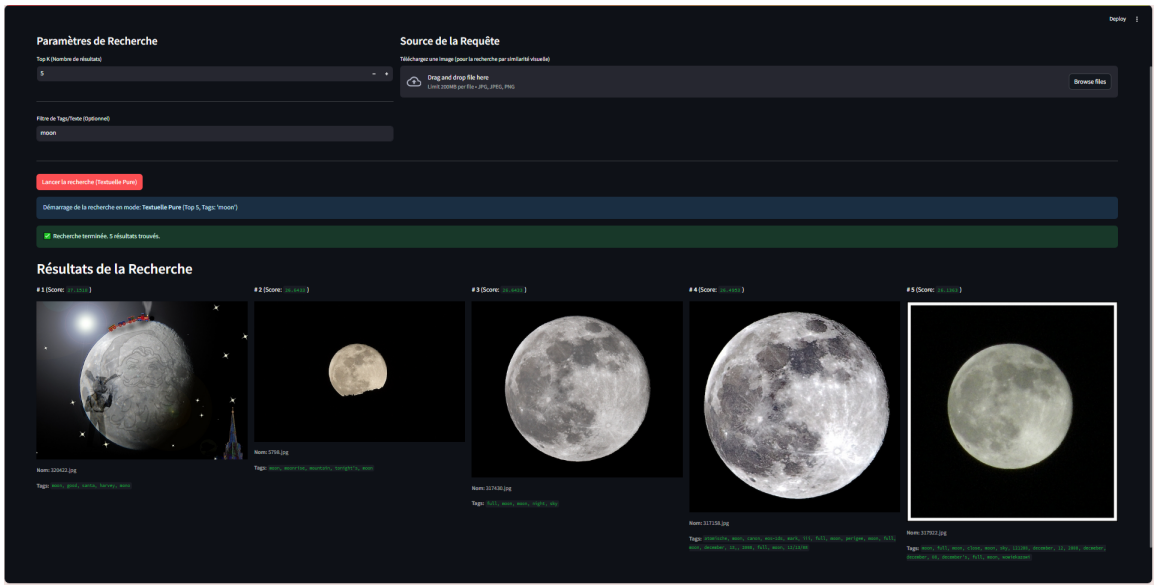


Figure 3: Text-based search results: Images retrieved based on tag matching with relevance scores.

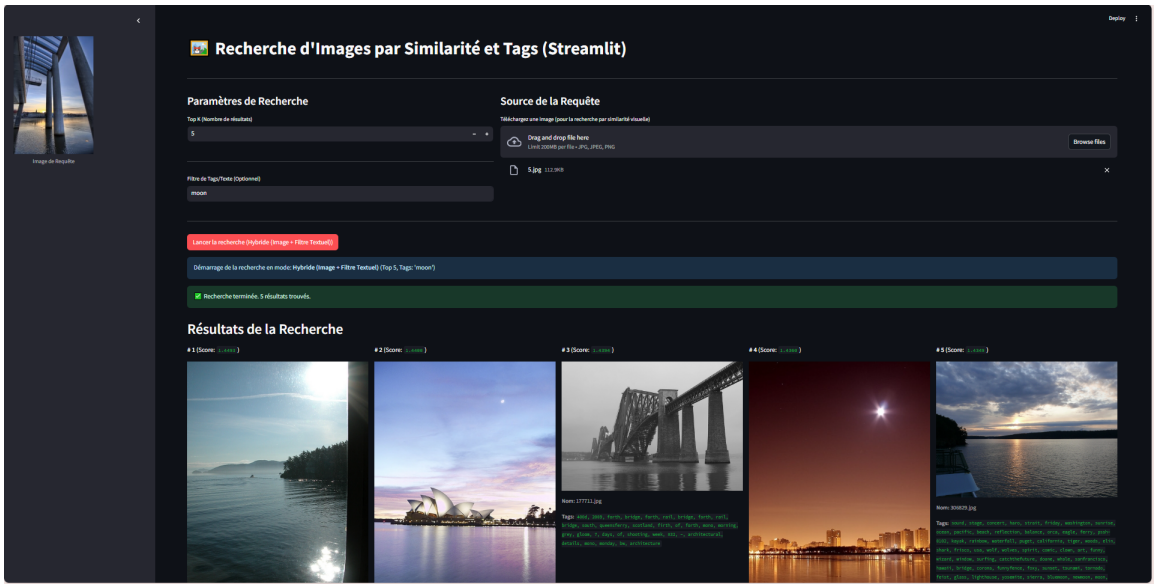


Figure 4: Image-based search results: Visually similar images retrieved using VGG16 features and cosine similarity k-NN search.

5.2 Performance Characteristics

5.2.1 Search Performance

The system achieves efficient retrieval through Elasticsearch's HNSW algorithm:

- **Index size**: Supports millions of images
- **Query latency**: Sub-second response times for k-NN search
- **Recall**: High-quality approximate nearest neighbor results
- **Scalability**: Horizontal scaling through Elasticsearch clustering

5.2.2 Feature Extraction Performance

- **Indexing time**: 1-2 seconds per image (GPU accelerated)
- **Query time**: 0.5-1 second per uploaded image
- **Storage requirements**: 100 KB per image for VGG16 features
- **Model loading**: One-time initialization cached by Streamlit

5.2.3 User Experience

The Streamlit interface provides:

- Instant visual feedback during search operations
- Fast results rendering with efficient image loading
- Intuitive controls requiring minimal user training
- Smooth interaction without page reloads

6 Deployment

6.1 Technology Stack

The production deployment utilizes the following components:

- **Python 3.8+**: Core implementation language
- **Streamlit**: Web application framework providing UI and application logic
- **Elasticsearch 8.x/9.x**: Vector and text search engine with k-NN support
- **TensorFlow/Keras**: Deep learning framework for VGG16 feature extraction
- **OpenCV**: Image processing and manipulation
- **NumPy**: Numerical computations and array operations

6.2 Deployment Configuration

6.2.1 Elasticsearch Setup

- Index created with dense vector mapping for 25,088-dimensional features
- HNSW parameters configured for optimal search performance
- Cosine similarity metric specified in index settings
- Cluster configured for high availability in production

6.2.2 Streamlit Deployment

- Application runs as a standalone process
- Environment variables configure Elasticsearch connection
- Model files (VGG16 weights) stored locally or on shared storage
- Image directory mounted or accessible via network filesystem
- Port configuration for external access

6.2.3 Performance Optimization

- VGG16 model loaded once and cached using `@st.cache_resource`
- Elasticsearch connection pooling for efficient queries
- Image thumbnails generated for faster display
- Lazy loading of images in results view
- GPU acceleration for feature extraction when available

7 Conclusion

This project successfully implemented a multimodal image search engine combining text-based and content-based retrieval methods using a streamlined Streamlit architecture. The system leverages VGG16's full-dimensional deep features to capture semantic visual content, while Elasticsearch's k-NN capabilities with cosine similarity provide efficient and accurate retrieval from large-scale image collections.

The unified Streamlit architecture simplifies deployment and maintenance while providing a responsive and intuitive user interface. The combination of transfer learning from ImageNet and modern search infrastructure demonstrates the effectiveness of deep learning approaches for practical image retrieval applications.