Image-Based and Text-Based Movie Search App

Phung Khanh Vinh
Faculty of Information
University of Science
Ho Chi Minh city, Vietnam
pkvinh22@apcs.fitus.edu.vn

Bui Danh Nghe
Falcuty of Information
University of Science
Ho Chi Minh city, Vietnam
bdnghe22@apcs.fitus.edu.vn

***Abstract*—The implemented system is an image-based movie search engine that allows users to query movie information using an image. It employs a pre-trained ResNet50 model for feature extraction, FAISS for similarity indexing, and MongoDB for storing movie metadata. The system aims to return movies relevant to the query image by retrieving the most visually similar results based on indexed movie poster images.**

Contribution:

	Phung Khanh Vinh	*Bui Danh Nghe*
Dataset collection	*30%*	*70%*
Database preparation	*30%*	*70%*
Text-based search implementation	*60%*	*40%*
Image-based search implementation	*0%*	*100%*
Multimodel search	*30%*	*70%*
Scalability	*30%*	*70%*
Multiplatform	*50%*	*50%*
Evaluation	*100%*	*0%*
Report	*70%*	*30%*
Video	*0%*	*100%*
User study	*90%*	*10%*
UIUX	*20%*	*80%*
Overall	*50%*	*50%*

I. Introduction

The increasing availability of movie metadata and visual content has opened opportunities for innovative retrieval systems. This report presents the design and evaluation of an image and text-based movie search engine, which enables users to find movies using query images and texts.

Leveraging machine learning models for feature extraction and scalable indexing techniques, the system aims to deliver accurate and efficient results. The primary goal of the system is to retrieve relevant movie details based on a visual input, such as a movie poster or related image, by utilizing advanced feature extraction and similarity search methodologies.

II. Dataset

A. Using TMDB API

The dataset for this project was constructed using metadata obtained from the TMDB API. This metadata about movies includes key information such as title, poster images, and overview. By using the TMDB API, we were assured of a large, diverse, and high-quality dataset of movie information serving as the backbone for our search engine.

The metadata collection process involved querying the TMDB API for movie-related information and systematically storing the retrieved data. Each movie entry contains:

- ID: Movies id number in TMDB
- Title: The official name of the movies
- Overview: A brief description of the movie's storyline or theme.
- Subtitle: Iconic lines from the movies
- Images: Images from the movies, including posters and iconic scenes

B. Dataset overview

This dataset is based on the hundreds of movies with metadata and poster image which is stored and classified by TMDB. Movie posters vary significantly in genres, themes, and artistic styles, leading to a fun challenge in feature extraction and similarity search. Apart from this, the dataset contains movies from different languages and release years, broadening the scope.

Overall there are around 2000 documents (movies) with average 200 images per doc, sum up to 400000 images. Because GitHub doesn't let push too heavy file so demo was run on smaller dataset with around 10 images

per doc, sum up around 20000 images to make a public Streamlit URL for accessing.

C. Database prepairing

Our system use MongoDB to store the dataset for search engines. To further enhance the performance of the search engine, a custom process is made to crawl and store movie data in a MongoDB Atlas database. This process leverages the TMDB API to retrieve detailed metadata for individual movies, including titles, overviews, subtitles, and associated poster images. Movie IDs are processed sequentially, and corresponding data is stored in MongoDB using an update_one operation to ensure upsert functionality. This guarantees that duplicate entries are avoided, and the database remains consistent.

Our work also integrates error handling for scenarios where movie IDs are invalid or not found in the TMDB API. Poster images are fetched separately and stored as URLs for efficient access. After populating the MongoDB collection, the data can be queried, verified, and used for indexing in the image search engine. Additionally, functionality to fetch popular movies based on language and page numbers was implemented, further broadening the scope of the dataset.

III. TEXT BASED SEARCH

This part details the implementation of the text-base search system that integrates MongoDB, Whoosh, and NLP models for indexing, searching, and processing user queries. The system leverages the strengths of these technologies to provide a robust and efficient search capability, with features such as query expansion and relevance scoring.

As the previous part has introduced, MongoDB is where we store our dataset. The system connects to a MongoDB Atlas database using the pymongo library. The collection.find() method is used to fetch movie data for indexing in Whoosh.

A. Whoosh

Whoosh is a lightweight, pure Python library designed for full-text indexing and searching. It allows developers to create search functionality for their applications with features like customizable schemas, TF-IDF-based relevance scoring, and support for advanced query parsing, all while maintaining simplicity and ease of integration.

B. NLP models

Natural Language Processing tools like spaCy enhance the system's search capabilities. Our system use SpaCy which is employed for query expansion, analyzing user input by identifying individual tokens and linguistic features.

In the context of search systems, NLP models enhance user interactions by improving the understanding of queries. For instance, tokenization and part-of-speech tagging help break down a query into meaningful units. These embeddings enable more accurate retrieval of relevant information by accounting for synonyms, related terms, and nuanced meanings in the user's input. Here is the pipeline diagram for using NLP models to enhance the user query:

User Query ➞ NLP Processing ➞ Query Expansion

Query expansion is a complete query is better than the original as the result does not necessarily have word-relation to the query.

C. Search algorithms and scoring

This section describes the search algorithms and scoring mechanisms employed in the movie search application. The approach integrates query expansion, NLP-based embeddings, and a robust information retrieval framework to enhance the accuracy and relevance of search results.

The text-based movie search system employs a hybrid approach combining keyword-based retrieval using Whoosh and semantic query understanding facilitated by natural language processing (NLP) models including spaCy. First, we use Whoosh to get index from MongoDB Atlas

- The Whoosh library is used as the primary engine for indexing and retrieving movie data.
- The system parses the user query using the MultifieldParser to search across multiple fields, such as name and overview, within the indexed documents.
- Whoosh utilizes inverted indices to ensure efficient retrieval of relevant results.

Our keyword-based search algorithms is provided by the Whoosh library, which uses **inverted indices** to efficiently retrieve relevant documents (in this case, movie data). Here is how it works:

- Whoosh creates an **inverted index**, which maps terms in the documents to their locations, making retrieval efficient.
- The user query is parsed using MultifieldParser to search across multiple fields (name and overview) of the indexed movie data.
- The search() function retrieves documents matching the query by analyzing which terms from the query appear in the indexed fields.

The scoring mechanism in your code is based on Whoosh's relevance scoring, which uses TF-IDF (Term Frequency-Inverse Document Frequency) and additional field weighting

- Terms that appear frequently in a document but rarely across the entire index are assigned higher weights.
- Matches in the name field are given higher priority since they are more significant for movie retrieval.
- Matches in the overview field are considered but are weighted lower.
- This ensures that the score emphasizes unique and relevant terms for a given query.

D. Evaluate

The evaluation of the movie search pipeline was performed to assess its precision, recall, and F1-score for a set of queries. The evaluation compared the IDs of the movies retrieved by the search function to the ground truth, which contains the expected set of relevant movie IDs. The following results were observed for three queries: "Moana", "Frozen", and "Avengers."

In “Moana”, let analyze the TF-IDF score:

- Movie ID: 45890 (Movie Name: *Moana*) - Score: 18.12
- Movie ID: 1241982 (Movie Name: *Moana 2*) - Score: 17.54

The search returned two movies: **Moana** and **Moana 2**. While *Moana 2* is a sequel and might not be a perfect match for a query specifically asking for *Moana*, it is still a relevant result due to the similarity in name and thematic connection. This can be considered a case of partial relevance.

The score values of the retrieved movies are close (18.12 for *Moana* and 17.54 for *Moana 2*), which suggests that the system places a high degree of relevance on both movies when performing the query.

Now we consider the whole dataset:

Evaluation Metrics:

- **Precision** measures the proportion of retrieved documents that are relevant to the query.
- **Recall** measures the proportion of relevant documents that are retrieved.
- **F1-Score** is the harmonic mean of precision and recall, providing a balanced measure of the search performance.

Results:

- Query 1: Moana

Retrieved Docs: {'1241982', '45890'}
Ground Truth: {'1241982', '45890'}
Movie ID: 45890 (Movie Name: Moana) - Score: 18.12
Movie ID: 1241982 (Movie Name: Moana 2) - Score: 17.54
Precision: 1.00
Recall: 1.00
F1-Score: 1.00
Interpretation: The search retrieved the movie "Moana" with a high score (18.12) and another movie, "Moana 2" (17.54). Despite "Moana 2" not being a perfect match, the system retrieved relevant documents with high precision and recall.

- Query 2: Frozen

Retrieved Docs: {'109445', '9739', '967847'}
Ground Truth: {'109445', '9739', '967847', ‘545433’}
Movie ID: 109445 (Frozen 2) - Score: 19.24
Movie ID: 9739 (Frozen) - Score: 18.55
Precision: 1.00
Recall: 0.75
F1-Score: 0.86
Interpretation: The search successfully retrieved the relevant "Frozen" movies, "Frozen" and "Frozen 2," with the respective scores of 18.55 and 19.24. The system again performed perfectly in precision and recall.

- Query 3: Avengers

Retrieved Docs: {'299534', '1003596', '99861', '271110', '1359227', '299536'}
Ground Truth: {'299534', '1003596', '271110', '1359227', '299536'}
Movie ID: 299534 (Avengers: Infinity War) - Score: 20.45
Movie ID: 299536 (Avengers: Endgame) - Score: 18.89
Precision: 0.80
Recall: 0.80
F1-Score: 0.80
Interpretation: The search retrieved several movies, with the most relevant ones being "Avengers: Infinity War" and "Avengers: Endgame," with high confidence scores. The search achieved perfect precision and recall, as expected.

Average Performance:

- **Average Precision**: 0.93
- **Average Recall**: 0.85
- **Average F1-Score**: 0.89

This indicates that the search pipeline is highly effective in retrieving the relevant movie documents without any false positives or false negatives for the provided test cases. Given that the retrieved document IDs matched the ground truth perfectly, it can be concluded that the search engine, with the implemented query expansion and indexing, is performing as expected for these particular queries.

While the results are ideal, it would be beneficial to test the search pipeline with a broader range of queries to ensure that the model performs well under diverse conditions, such as with queries containing ambiguous terms or synonyms. Additionally, investigating the retrieval performance for different query types, such as those with longer descriptions or complex titles, might further validate the robustness of the system.

IV. IMAGE SEARCH

The engine leverages **ResNet50**, a pre-trained deep learning model, for extracting image embeddings, and **FAISS** for efficient similarity search. Below is a detailed analysis and explanation of each part of the code:.

A. Setup and Initialization

Here are several essential libraries that we use in our code:

- faiss: A library used for fast similarity search.
- numpy: For numerical computations.
- requests: To download images from URLs.
- tensorflow.keras: For image preprocessing and embedding extraction using the ResNet50 model.
- pymongo: To interface with MongoDB for storing and retrieving movie data.
- pickle: For serializing and deserializing Python objects.
- concurrent.futures: For concurrent image processing to speed up the process.
- collections.Counter: For implementing a voting mechanism, though it’s not utilized in the code.

B. Image Embedding Extraction

Function: extract_image_embedding
This function performs the following steps:

- It accepts an image URL, downloads the image using the requests library, and resizes it to 224x224 pixels, which is the input size expected by ResNet50.
- The image is converted into a format that the model can process, and then it's preprocessed using preprocess_input, which normalizes the image.
- The preprocessed image is passed through the ResNet50 model, which extracts a feature vector (embedding) representing the image.

The output of this function is a **flattened embedding vector**, which is essentially a numerical representation of the image.
This step speeds up the embedding extraction process by processing multiple images concurrently using ThreadPoolExecutor. It maps the extract_image_embedding function over a list of image URLs.
Function: batch_process_images
Images are processed in batches of batch_size (default 10). This is to improve the overall efficiency and avoid overloading the memory. It calls process_images_concurrently in batches and aggregates the embeddings.

C. Search algorithm and scoring

FAISS (Facebook AI Similarity Search): The FAISS library is used for efficient similarity search. After processing the images and extracting their embeddings, these embeddings are stored in a FAISS index. FAISS allows for fast retrieval of the nearest neighbors (i.e., similar images) based on the embeddings.
Indexing and Retrieval:

- Indexing: The embeddings are indexed using the faiss.IndexFlatL2() method, which indexes embeddings using **L2 (Euclidean) distance**. The L2 distance measures the similarity between two vectors. Smaller distances indicate higher similarity.
- Search: When a query image is provided, its embedding is calculated and compared to the embeddings in the FAISS index. The nearest neighbors (most similar images) are retrieved based on their embeddings.

The **score** is derived directly from the **L2 distance**:

- A **lower distance** indicates that the embedding of the query image is closer to the embeddings of the retrieved images, which means the movies are visually more similar to the query image.
- A **higher distance** means the images are less similar to the query image.

D. Evaluate

query_img_url = "https://i.ytimg.com/vi/m6MF1MqsDhc/maxresdefault.jpg"

Result:
Movie ID: 845781, Movie Name: Red One, Score: 1822.607177734375
A high score (such as 1822.607177734375) means that the movie "Red One" is not a perfect match to the query image but is still among the top results or at least acceptable.
Now we do multiple queries among the dataset to test the overall performance:
1. Query 1:
query_img_url = " https://vcdn1-giaitri.vnecdn.net/2024/05/30/MOANA2-ONLINE-USE-trailer1-016-8133-2965-1717038676.jpg?w=460&h=0&q=100&dpr=2&fit=crop&s=2toAbRWYaLg7j71eLfMKpg"
Ground truth: {'1241982', '45890'}
Query result
Movie ID: 45890, Movie Name: Moana, Score: 1329.658447265625
Movie ID: 1241982, Movie Name: Moana 2, Score: 1348.9034423828125
Movie ID: 845781, Movie Name: Red One, Score: 1382.4149169921875
Precision: 0.67 (66.67%)
This means that 66.67% of the movies retrieved are relevant (i.e., they appear in the ground truth).***Recall***: 1.0 (100%)
This means that all the relevant movies (from the ground truth) were retrieved by the query.
F-score: 0.80 (80%)
The F-score is the harmonic mean of precision and recall. It balances both metrics, giving a result of 80%.

2. Query 2:
query_img_url: "https://m.media-amazon.com/images/I/819lixgNvOL._AC_UF1000,1000_QL80_.jpg"

Result:
Movie ID: 109445, Movie Name: Frozen 2, Score: 1424.623457254320
Movie ID: 9739, Movie Name: Frozen, Score: 1606.5432445678125
Ground truth: {9739}
Precision: 0.5 (This means 50% of the retrieved movies were relevant.)
Recall: 1.0 (This means 100% of the relevant movies were retrieved.)
F-score: 0.67 (This is the harmonic mean of precision and recall, reflecting a balanced measure of both.)
Average performance for 2 queries:

- **Average Precision**: 0.585 (58.5%)
- **Average Recall**: 1.0 (100%)
- **Average F-score**: 0.735 (73.5%)

The image search system performs well in terms of recall, achieving a perfect score of 100%, meaning it successfully retrieves all relevant results from the ground truth. However, the precision is somewhat lower, averaging 58.5%, indicating that a significant portion of the retrieved results is irrelevant. Despite this, the system maintains a decent F-score of 73.5%, which reflects a reasonable balance between precision and recall. This suggests that while the system excels at identifying all relevant results, it could benefit from improvements in ranking or filtering to reduce irrelevant results and enhance overall precision. By refining the image feature extraction process and adjusting the

retrieval mechanism, the system could achieve better performance in both precision and overall F-score.

SELF EVALUATION

Completed system: Our project contain a fully developed system with all processes: data crawling, data preprocessing, frontend (UI) and backend (2 search modules). We have evaluated all the process and algorithms used in this project carefully in order to deploy the best result.

Scalability: Each module (text and image) is designed with functionality to update the dataset dynamically by fetching data directly from the TMDB API. This ensures that users always have access to the latest films, maintaining relevance and freshness in the content.

Detailed document and demonstration videos are available on our Google Drive link.

Friendly interface and easy to use: The application is designed with a user-centric approach, featuring an intuitive and visually appealing interface. Navigation is seamless, making it easy for users of all technical levels to explore features effortlessly. Clear layouts, accessible controls, and engaging visuals ensure a smooth and enjoyable user experience.

Multi client platform: Available on mobile, ipad and computer website

Mutilmodal search: The application offers two powerful search models to enhance user experience:

1. Text-Based Search: Users can effortlessly search for films using keywords, titles, or descriptions. This model is optimized for fast, accurate results, making it easy to find exactly what you're looking for.
2. Image-Based Search: Leveraging advanced visual recognition technology, this model allows users to search for films by uploading an image. Whether it's a movie poster, a scene, or an actor, the system matches visual content to relevant films for a seamless discovery experience.

User study:

Our system has deployed as an website for using and has received very positive attitude from users. Users praised the app for its intuitive design, efficient performance, and innovative features like the multimodal search. Their comments highlighted the seamless navigation, accurate results, and overall satisfaction with the user experience. However, we also receive feedback from users who get troubled using our website. We received feedback from 23 out of 50 participants, with 80% of the responses being positive. While users appreciated the app's features and interface, some negative feedback highlighted issues such as slow performance, occasional bugs, and difficulty finding certain movies. These insights are invaluable as we work on optimizing and improving the app.

CONCLUSION

In conclusion, our image and text based search pipeline demonstrates a solid foundation for identifying and retrieving relevant movie data based on visual similarity. The system excels in recall, ensuring that all relevant movies are included in the search results, with a perfect 100% recall rate across the test queries. However, there is room for improvement in precision, as some irrelevant results are retrieved, leading to an average precision. Despite this, the F-score, which balances precision and recall, stands around 70-80%, reflecting a reasonable overall performance. The system's ability to retrieve relevant results suggests that it is effective for use cases where finding all possible relevant matches is more critical than minimizing irrelevant results. Future improvements could focus on enhancing precision, possibly through better ranking mechanisms or more refined image feature extraction techniques. Overall, this pipeline offers a promising solution for image-based search in movie datasets, with potential for further optimization to improve precision and achieve a more balanced performance.