DLS Assignment 3

Team 4: Ilia Mistiurin, Nazgul Salikhova, Milyausha Shamsutdionova

July 2024

1 The Problem Formulation

This project aims to develop a deep learning-based search system to identify original photos from fragmented images.

Clarifying Questions:

- How should we represent the dataset that will allow to search the original photo by its fragment?
- Fragments are cropped parts of 'original' images that are uploaded into the system?
- Is the found photo for a fragment supposed to be an original photo with a fragment or a photo with a similar fragment?
- Should we use photos with high resolution to create components for a database?
- How large should the dataset be that we are working with (number of images)?
- How will the system be evaluated and tested to ensure its accuracy and reliability?

Use Cases and Business Goal

1. Detection of plagiarism in images.
2. Content verification in social media.

Requirements

- High accuracy in matching fragments to originals.
- Low latency in search results.
- Scalable solution to handle large datasets.

Scope and Personalization

- Features needed: Image preparation for a database, similarity searching.
- Scale: Hundreds of images with their components.

Performance

- Prediction latency (ideal scenario): ≤ 5 seconds.
- Scale of prediction: Ability to handle real-time queries.

Constraints

- The finding of the best way to make a potential fragments (dataset) from original ones to enchance the efficiency of searching can be challenging.
- Limited computational resources.

Data

- Sources: Public image dataset of a specific domain (for object detection).
- Availability: Open Images dataset (class = "Flower"), custom dataset for evaluation.

Assumptions

- The images are of sufficient resolution to make a qualitative potential fragments.
- The matching algorithm primarily rely on visual features without the need for additional metadata.
- The original photo of a fragment locates in database.
- The size of query fragment is fixed or standartized before searching.
- No private images are used.

ML Problem Translation

ML objective: Match image fragments to original photos or similar ones.
ML I/O: Input - image fragment; Output - matching original photo/s.
ML category: Image retrieval/matching.
Do we need ML: For projects requiring high accuracy, scalability, and robustness, especially with large datasets and varying image conditions, machine learning would generally be preferred.

Trade-offs

- No need of high computational resources (currently with limited-sized database).
- The choice of embedder model may affect on quality of searching.

2 Metrics (Online and Offline)

- The offline classification metrics like Precision, Recall, and F1 Score will be key metrics for our project. These metrics assess will say how well the system correctly identifies matching images.
- Considering metrics related to computational cost, latency, and operational efficiency will be useful either. These metrics will ensure that the system performs efficiently in terms of resource utilization.
- Among online metrics the user feedback(like rate) will help us understand how well the search works and will indicates user satisfaction and engagement based on likes or ratings given to matched images.

3 Architectural Components (MVP Logic)

High level architecture and main components

Representation learning:

- Transform input data into representations (embeddings) - similar images that are close in their embedding space
- Input image cropped via sliding window and Object Detection model for creating smaller images(components) for creating a database.
- Use distance between embeddings as a similarity measure between images
- Use HNSW or Approximate K-Nearest Neighbours Search (depends on performance)

Modular architecture design

- Image preprocessing: Sliding window technique, Object detection model: ultralytics/yolov8-oiv7
- Embedder: nomic-ai/nomic-embed-vision-v1.5
- Database: Qdrant (+ possibly FAISS for indexing)

Figure 1 illustrates the modular architecture design components mentioned above.

4 Data Collection and Preparation

Data Needs

- Target variable: Matching original photos
- Big actors: Users, image fragments, original photos
- Type and volume: Images, medium volume (100-500 pcs)

Data Sources

- Public dataset: Open Images v7 (class = "Flower")
- Custom dataset for an evaluation

Data Storage

- Structured and unstructured data handling
- Image storage - database (embedded image parts), hard drive (images itself)

ML Data types

- Structured: numerical
- Unstructured: images

Labelling

Since our project focuses on identifying original photos from fragmented images, the labelling process is streamlined but still critical to the model's performance.

Labelling Methods

- Natural labels: In our context, these are inherent in the images themselves from Open Images dataset. The original photos are labeled as containing flowers, and fragments are created from these photos.
- Programmatic labeling: We generate approximately 50,000 image instances using a sliding window technique. This involves systematically cropping sections of the original images to create fragments.
- ML utilizing labeling: For new images we use the YOLOv8 model pretrained on the Open Images dataset to ensure that these fragments contain flowers, providing implicit labels.

Data Generation Pipeline

1. Data collection and ingestion
2. Cropping by object detection boxes and using sliding window technique

5 Feature Engineering

Feature Selection

- Define actors: Users, images
- Actor-specific features: searching photo by its fragment
- Extra features: uploading the photo into database

Feature Representation

- Embeddings
- Scaling and normalization for numerical features

Preprocessing Features

- Images: cropping

6 Models Usage

In our project, we leveraged state-of-the-art (SOTA) pretrained models to achieve our objectives without the need for additional training. Here's a detailed breakdown of our approach:

Models Selection

- **YOLOv8:** Selected for its superior object detection capabilities, particularly effective for identifying and localizing flowers within images.
- **Nomic-Embed-Vision-v1.5:** Chosen for its high performance in generating image embeddings, crucial for our similarity search and image retrieval tasks.

Usage

- YOLOv8:
  - Type: Convolutional Neural Network (CNN)
  - Usage: Utilized to create a dataset of flower fragments by detecting and cropping regions containing flowers from larger images.
- Nomic-Embed-Vision-v1.5:
  - Type: Embedding model
  - Usage: Employed to generate embeddings for both original images and their corresponding fragments, facilitating efficient similarity searches.

7 Prediction Service

The Prediction Service within our architecture is responsible for handling online searching with low latency. This service ensures that predictions are processed and returned as requests arrive, allowing for real-time interaction with the system.

Key aspects of the Prediction Service include leveraging optimized algorithms and infrastructure to minimize latency, thus providing users with quick responses to their queries. This capability is critical for our image search system, where users expect near-instantaneous results when searching for matching images based on fragment inputs.

Prediction pipeline

Embedding generation service:
image - preprocessing - embedding gen (ML model) - image embedding

HNSW that is built in the Qdrant (currently, but possible use of Approximate KNN like FAISS later if needed).

- Retrieve the most similar images from embedding space
- For query q search only k most similar vectors
- Quite fast implementation on the CPU
- No ability to shrink embedding space without recreating the vector space

Indexing pipeline

- Indexing service: indexes images by their embeddings (possibly FAISS)
- Keep the table updated for new images
- increases memory usage - use optimization (vector / product quantization)

8 Online Testing and Model Deployment

- Deployment of the application using Docker could be possible.
- Testing via user feedback on the search accuracy.

9 Scaling, Monitoring, and Updates

- Qdrant could be easily scaled using clusters.
- Object Detection model improvement via higher compute resources or other case-suited models.

10 Development of UML - diagram

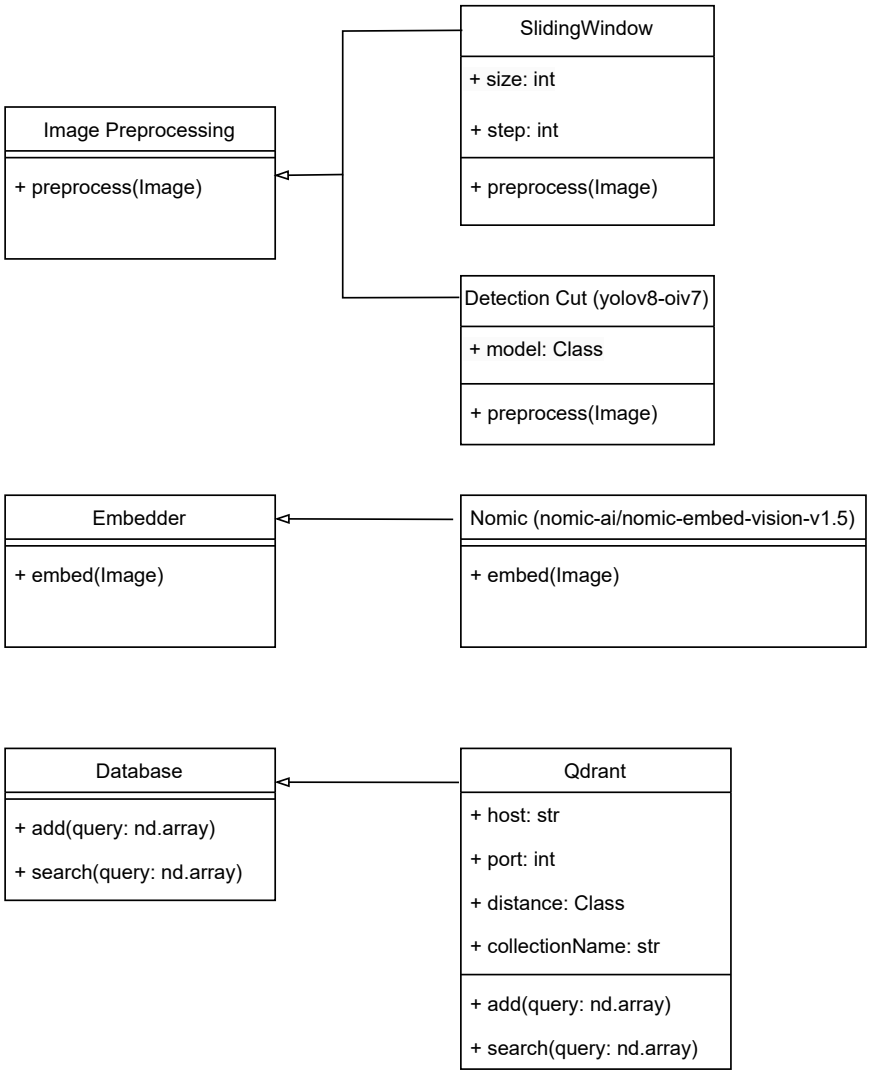

Figure 1: UML - diagram.