Content-Based Image Retrieval Using Locality Sensitive Hashing

Data Structures and Algorithms

Fall 2025

Date: 2026-01-27

Abstract
This report describes the implementation of a content-based image retrieval (CBIR) system using a custom vector database backed by SQLite, exact k-nearest neighbor (k-NN) search, and Locality Sensitive Hashing (LSH) for approximate search. Image embeddings are generated with a pre-trained ResNet18 model and stored as 512-dimensional vectors. The system provides CRUD operations over vectors and a graphical user interface for interactive search. We compare exact and approximate retrieval in terms of query time and recall.

1. Introduction
Content-based image retrieval represents images as numerical vectors (embeddings) and retrieves similar images by comparing these vectors. In this project, embeddings are generated with ResNet18 and stored in a vector database. Similarity search is implemented both exactly (brute-force k-NN) and approximately (LSH).

2. Dataset and Embeddings
We use the Caltech-101 dataset and precomputed ResNet18 embeddings stored on disk as NumPy arrays. Each image is represented by a 512-dimensional vector. These arrays are used only for the initial bootstrap of the vector database; afterward, all operations use the SQLite database.

3. System Architecture
The system consists of the following components: (1) a vector database implemented in Python with SQLite persistence, (2) exact k-NN search over in-memory vectors, (3) an LSH index for approximate search, and (4) a Streamlit GUI that supports image upload, search, and visualization.

4. Vector Storage and CRUD
Vectors and metadata are stored in a SQLite database (data/vector_store.sqlite). The table schema includes: id (UUID), vector (float32 BLOB), labels (JSON list), and image_path. The VectorDB layer loads all rows into memory at startup and supports Create, Read, Update, and Delete operations. SQLite provides durable persistence across runs.

5. Embedding Pipeline

For new user uploads, images are preprocessed using the default ResNet18 transforms and fed through a pre-trained ResNet18 network with the classification head removed. The output 512-D vector is L2-normalized before storage. This ensures cosine similarity corresponds to dot product.

6. Exact k-NN Search

Exact search computes similarity between the query vector and all vectors in the database. We support cosine similarity (dot product for normalized vectors), Euclidean distance, and Manhattan distance. The brute-force approach is O(N·d) per query.

7. Approximate Search with LSH

Locality Sensitive Hashing is used to reduce the search space by hashing similar vectors into the same buckets. We implement random hyperplane LSH for cosine similarity. At query time, we retrieve candidate vectors from the matching buckets and re-rank them by cosine similarity. This yields sub-linear expected query time at the cost of reduced recall.

8. Graphical User Interface

The Streamlit GUI enables users to upload images, insert them into the database, and search for similar images using either exact k-NN or LSH. It also provides a 2D scatter plot of embeddings using PCA for visualization.

9. Benchmarking Methodology

We benchmarked exact k-NN (cosine and Euclidean) and LSH (cosine rerank) on 20 random queries from the dataset with k=10. Each method reports average query time. LSH additionally reports recall@k against the cosine ground truth.

10. Results

Method	Metric	k	Avg Time (s)	Recall@k	LSH Params
knn	cosine	10	0.000400		
knn	euclidean	10	0.011088		
lsh	cosine_rer	10	0.000156	0.29	planes=12

ank , tables=2

11. Analysis and Discussion

Exact k-NN with cosine similarity is faster than Euclidean distance because it avoids per-vector square root operations and reduces to a dot product for normalized vectors. LSH provides faster queries by limiting comparisons to a candidate set, but recall is lower. Increasing the number of hash planes or tables can improve recall at the cost of more computation.

12. Time Complexity

Brute-force k-NN: O(N·d) per query, where N is the number of vectors and d is the dimension. LSH: O(n_planes·d·n_tables) to hash plus candidate rerank, with expected sub-linear behavior in practice depending on hash selectivity.

13. Limitations

LSH recall depends on parameter tuning and may miss true nearest neighbors. The current system loads all vectors into memory at startup, which may be heavy for very large datasets. The GUI scatter plot uses PCA and does not preserve all high-dimensional relationships.

14. Conclusion

We implemented a complete CBIR pipeline using ResNet18 embeddings, SQLite-backed vector storage, exact k-NN search, and LSH approximate search. The GUI allows interactive querying and visualization. Results confirm the expected trade-off between accuracy and query time for LSH.

References

Leskovec, Rajaraman, Ullman. Mining of Massive Datasets, Chapter 3.

He et al. Deep Residual Learning for Image Recognition (ResNet).