CS 5330 - Pattern Recognition and Computer Vision

Project 2: Content-based Image Retrieval

Name: Joseph Defendre, Sourav Das

Date: February 2026

Course: CS 5330 Pattern Recognition and Computer Vision

---

1. Project Description

This project implements a content-based image retrieval (CBIR) system that matches a query image to a database by comparing visual features. The system supports baseline patch matching, color histograms, multi-region histograms, texture histograms (Sobel magnitude), deep network embeddings, and a custom feature configuration for a chosen category. Each method produces a feature vector for the target and database images, computes a distance metric, and returns the top-N closest matches.

---

2. Required Results

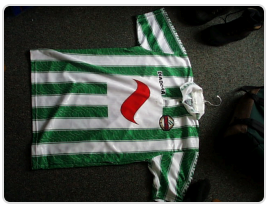
Task 1 — Baseline Matching

Implemented a 7×7 center-patch feature and ranked images with SSD for the required query.

Query: pic.1016.jpg **Top matches:**

pic.1016.jpg

pic.0986.jpg



pic.0641.jpg

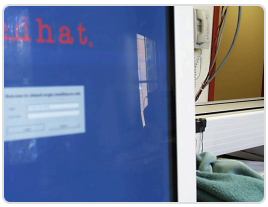
pic.0547.jpg

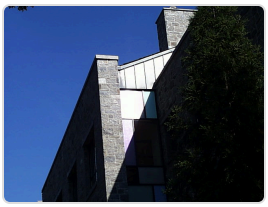
Task 2 — Histogram Matching

Computed a whole-image RG chromaticity histogram (16×16 bins), normalized counts, and used histogram intersection for matching.

Query: pic.0164.jpg **Top matches:**


pic.0164.jpg


pic.0080.jpg


pic.1032.jpg


pic.0461.jpg

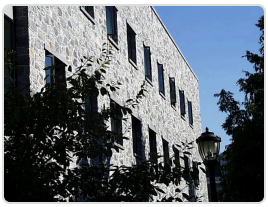
Task 3 — Multi-histogram Matching

Using two or more color histograms (e.g., different regions) and combine their distances with a custom weighting scheme. Built top/bottom RGB histograms (8 bins per channel) and combined histogram-intersection distances with equal weights.

Query: pic.0274.jpg **Top matches:**


pic.0274.jpg


pic.0273.jpg


pic.1031.jpg


pic.0409.jpg

Task 4 — Texture + Color

Combining a whole-image color histogram with a texture histogram and design a distance metric that balances both. Used an RGB color histogram plus a Sobel magnitude histogram for texture, then compared with histogram intersection using equal weights.

Query: pic.0535.jpg **Top matches:**


pic.0535.jpg


pic.0004.jpg


pic.0001.jpg


pic.0356.jpg

Comparison vs Task 2/3: The texture+color method shifts matches toward images with similar edge structure (buildings/geometry) rather than only color similarity, compared to the histogram-only methods.

Task 5 — Deep Network Embeddings

Using deep network embeddings (e.g., ResNet18 global average pooling output) as features and match with a suitable distance metric. Loaded precomputed ResNet18 embeddings from CSV and used cosine distance to rank results for the required queries.

Query: pic.0893.jpg **Top matches:**


pic.0893.jpg

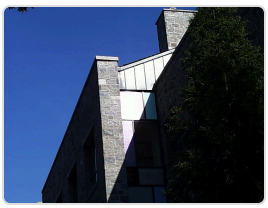

pic.0897.jpg


pic.0136.jpg


pic.0146.jpg

Query: pic.0164.jpg **Top matches:**


pic.0164.jpg


pic.1032.jpg


pic.0213.jpg


pic.0690.jpg

Comparison vs classic features: DNN matches tend to preserve semantic content and layout even when overall color distributions differ, while classic histograms bias toward color similarity.

Task 6 — Classic vs DNN Comparison

We ran histogram RG and DNN embeddings on the specified images (pic.1072.jpg, pic.0948.jpg, pic.0734.jpg) and summarized how color-biased vs semantic matches differ.

Queries: pic.1072.jpg, pic.0948.jpg, pic.0734.jpg

pic.1072.jpg — Histogram RG returns color-similar scenes; DNN returns images with similar scene structure even with different palettes.

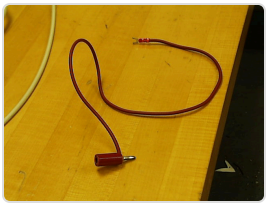
pic.0948.jpg — Histogram RG emphasizes color tone; DNN yields closer scene/subject context.

pic.0734.jpg — DNN results show near-neighbor scene continuity, while histogram RG mixes in visually similar colors from different contexts.

Task 7 — Custom Design (Sunset-Oriented Feature)

Designing a custom feature for a category of our choice (optionally including DNN features, but not exclusively) and report best and least similar matches. We implemented a sunset-oriented 3-region RGB histogram with higher weight on the horizon/sky transition and used histogram intersection to retrieve top-5 and least-5 matches.

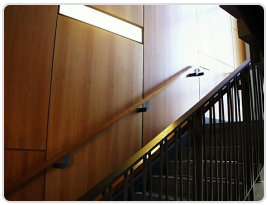
Query: pic.0048.jpg — Top 5 matches:



pic.0048.jpg

pic.0552.jpg

pic.0533.jpg



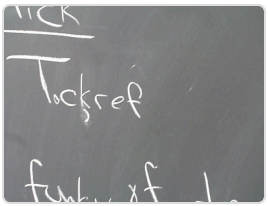
pic.1003.jpg

pic.1059.jpg

Query: pic.0048.jpg — Least similar:

pic.0511.jpg

pic.0558.jpg



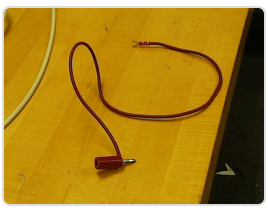
pic.0228.jpg

pic.0890.jpg

pic.0689.jpg

Query: pic.0552.jpg — Top 5 matches:

pic.0552.jpg



pic.0048.jpg

pic.0324.jpg

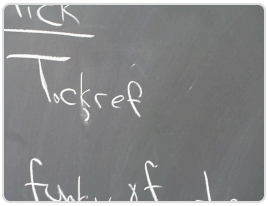
pic.0197.jpg

pic.0958.jpg

Query: pic.0552.jpg — Least similar:

pic.0511.jpg

pic.0558.jpg



pic.0228.jpg

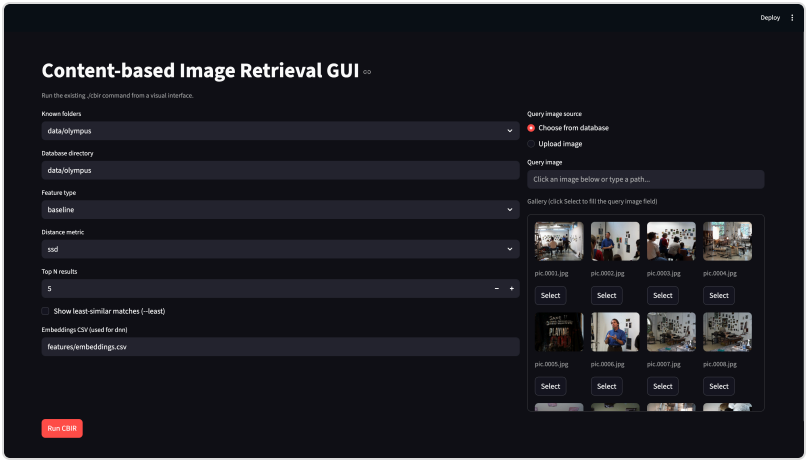
pic.0890.jpg

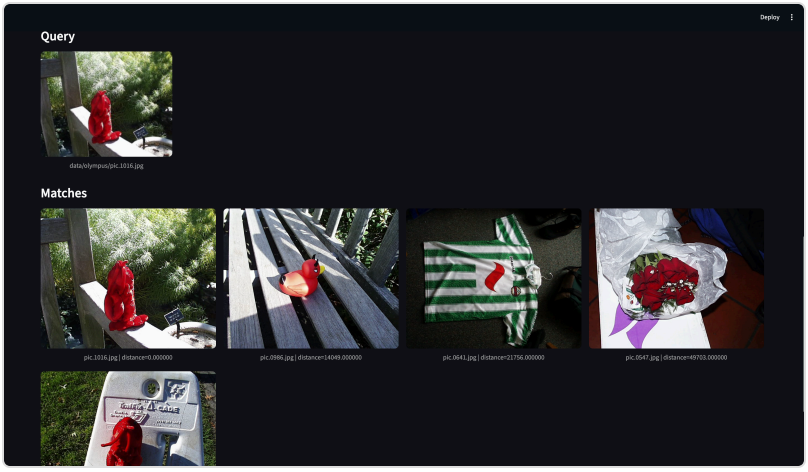
pic.0046.jpg

---

3. Extensions

We built a lightweight Streamlit front-end to run CBIR visually. The GUI lets users pick a query image and database folder, choose the feature type and distance metric, set `N`, optionally show least-similar results, and display thumbnail grids of the ranked images. It wraps the compiled C++ `cbir` binary and supports DNN embeddings by accepting a CSV path.





---

4. Reflection

Building multiple feature extractors clarified how different representations capture different aspects of visual similarity. The classic histograms were fast and intuitive but biased toward color distributions, while adding texture helped match structural cues. The DNN embeddings frequently produced more semantically coherent matches, especially when color alone was ambiguous. The custom sunset feature benefited from emphasizing top-to-bottom regions, which aligned with the sky-to-ground gradient often present in sunsets.

This project also surfaced the practical challenges that don't show up in a clean algorithm description. Getting C++ feature code, file I/O, and normalization to agree across different pipelines took more time than expected, and small mistakes (like inconsistent image resizing or channel order) produced results that were confusing until we tracked them down. We also had to balance speed with quality: some features were quick to compute but brittle, while richer features demanded more careful tuning and debugging. It was a good reminder that "working" code isn't the same as "trustworthy" code.

On the learning side, we came away with a deeper intuition for how representation choices shape retrieval behavior. We learned to read failure cases as signals about the feature, not just as bad luck from the dataset, and to iterate with curiosity rather than frustration. Building a simple GUI also made the system feel more tangible and helped us test ideas faster. Overall, the biggest takeaway was that CBIR is as much about disciplined engineering and thoughtful evaluation as it is about the underlying math.

---

5. Acknowledgments

- **Claude (Anthropic):** Used as a generative AI assistant for help with code implementation, debugging, and report writing.
- **Streamlit:** https://docs.streamlit.io/ - Used to build the lightweight GUI for running CBIR queries and visualizing ranked results.
- **OpenCV Documentation:** https://docs.opencv.org/ - Reference for image processing functions, video capture, and Haar cascades.
- **ResNet18 (PyTorch Model Zoo):** https://pytorch.org/vision/stable/models/generated/torchvision.models.resnet18.html - Pretrained embeddings referenced for the deep feature baseline and comparison.

- **CS 5330 Course Materials:** Lecture notes and project specifications provided guidance on distance metric implementation.

---

6. Time Travel Days Used

0 days