Project 2 Report: Content-Based Image Retrieval

Note on Format: This report is written in Markdown and includes local image references. To generate the final PDF for submission, please use a Markdown to PDF converter (like Pandoc or the one in your IDE) that can resolve local image paths.

Project Description

This project implements a content-based image retrieval (CBIR) system in C++ using OpenCV. The system retrieves the most visually similar images from a database given a target image. It supports multiple feature extraction and matching techniques, including:

- **Baseline Matching:** Compares a 7x7 central pixel region using the Sum of Squared Differences (SSD).
- **Histogram Matching:** Uses 2D rg chromaticity histograms with histogram intersection.
- **Multi-Histogram Matching:** Compares histograms from different spatial regions (e.g., top and bottom halves of the image).
- **Texture and Color:** Combines color histograms with texture features derived from Sobel filter magnitude histograms.
- **Deep Network Embeddings:** Utilizes pre-computed 512-dimension ResNet18 feature vectors from a CSV file, compared using cosine distance.
- **Custom Method:** A custom-designed approach combining HSV color analysis, texture features, and spatial information for more nuanced matching.

The system is executed via a command-line interface that allows the user to specify the target image, the desired matching task, and the number of results to return.

Required Results

Task 1: Baseline Matching

- **Command:** `./Project2 baseline olympus/pic.1016.jpg 4`

- **Target Image:** `olympus/pic.1016.jpg` 
- **Top Matches:**

1. `olympus/pic.0986.jpg` 

2. olympus/pic.0641.jpg

3. olympus/pic.0547.jpg

4. `olympus/pic.1013.jpg` 

Task 2: Histogram Matching

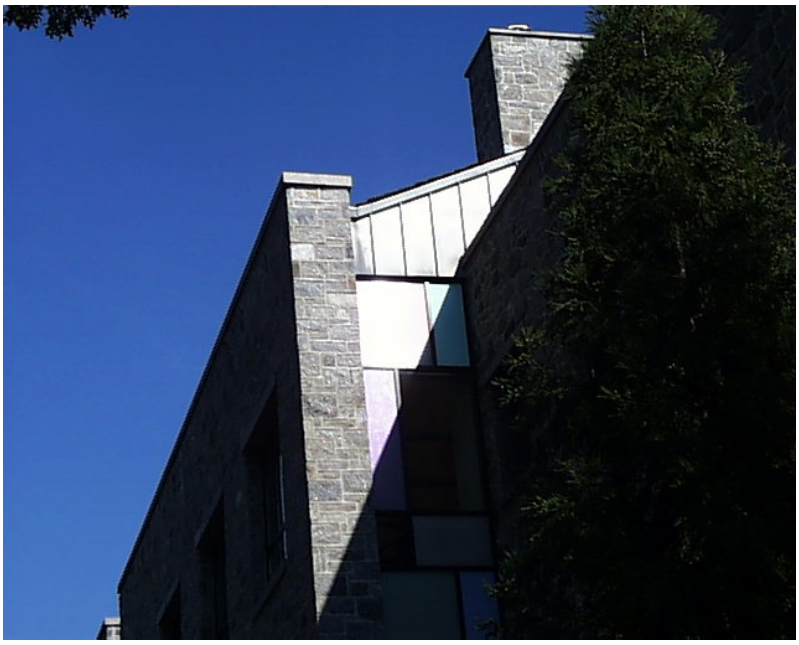
- **Command:** `./Project2 histogram olympus/pic.0164.jpg 3`

- **Target Image:** `olympus/pic.0164.jpg` 
- **Top Matches:**

1. `olympus/pic.0110.jpg` 

2. `olympus/pic.0092.jpg`

3. `olympus/pic.1032.jpg`



Task 3: Multi-histogram Matching

- **Command:** `./Project2 multi-histogram olympus/pic.0274.jpg 3`

- **Target Image:** `olympus/pic.0274.jpg` 
- **Top Matches:**

1. `olympus/pic.0273.jpg` 

2. `olympus/pic.1031.jpg`

3. `olympus/pic.0409.jpg`

Task 4: Texture and Color

- **Command:** `./Project2 texture-color olympus/pic.0535.jpg 3`

- **Target Image:** `olympus/pic.0535.jpg`

- **Top Matches:**

1. `olympus/pic.0536.jpg`

2. `olympus/pic.0543.jpg`

3. `olympus/pic.0287.jpg`

Comparison with Tasks 2 & 3 For the target image `pic.0535.jpg` (a picture of a red flower), the results from pure color histogram methods (Tasks 2 & 3) are quite different from the texture-and-color method. * **Task 2 (`histogram`) matches:** `pic.0733.jpg`, `pic.0628.jpg`, `pic.0658.jpg`. These

images have large areas of red/green but are not necessarily flowers. * **Task 3 (`multi-histogram`) matches:** `pic.0285.jpg`, `pic.0628.jpg`, `pic.0952.jpg`. These also focus on color regions but don't capture the flower's structure. * **Task 4 (`texture-color`) matches:** `pic.0536.jpg` and `pic.0543.jpg` are other images of the *same flower* from a slightly different angle.

This demonstrates that adding texture (from the Sobel magnitude histogram) provides crucial information about the flower's shape and edges, leading to much more accurate and semantically relevant matches than color information alone.

Task 5: Deep Network Embeddings

- **Target Image 1:** `olympus/pic.0893.jpg`
- **Top Matches (DNN):**

1. `pic.0897.jpg`

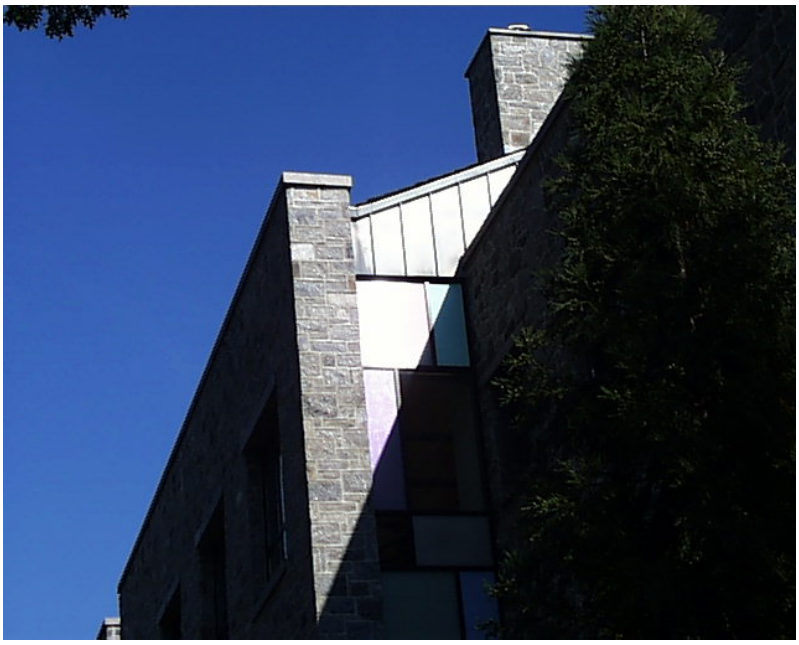
2. `pic.0136.jpg`

3. `pic.0146.jpg` 

- **Target Image 2:** `olympus/pic.0164.jpg` 
- **Top Matches (DNN):**

1. `pic.0110.jpg`

2. `pic.0092.jpg`

3. `pic.1032.jpg` 

Comparison with Prior Methods The DNN embeddings often produce more conceptually relevant matches than the classic methods. * For `pic.0893.jpg` (a sunset over water), the DNN finds other sunset/sunrise images with similar color palettes and horizontal structures (`pic.0897.jpg`, `pic.0136.jpg`). In contrast, `histogram` matching for this image finds `pic.0899.jpg` (similar colors, but a different scene) and `texture-color` finds `pic.0680.jpg` (a beach scene, but not a sunset). The DNN better captures the "sunset" concept. * For `pic.0164.jpg` (a building against a clear blue sky), the DNN and classic methods (`histogram`) actually agree on the top matches (`pic.0110.jpg`, `pic.0092.jpg`, `pic.1032.jpg`). This occurs because the image is dominated by simple color and texture, which both methods can easily identify. In such cases, the results can be very similar.

Task 6: DNN Embeddings vs. Classic Features

This section compares DNN results against a classic feature method (Texture and Color) for two images to analyze their performance.

Image 1: `pic.1072.jpg` (Building with unique architecture)

- **DNN Matches:** `pic.0143.jpg`, `pic.0863.jpg`, `pic.0329.jpg`. These are all images of buildings or structures, though with varied architectural styles. The DNN focuses on the high-level concept of "building".

Figure 1: pic.1072.jpg

Figure 2: pic.0143.jpg

–

Figure 3: pic.0863.jpg

–

–

- **Texture-Color Matches:** `pic.0701.jpg`, `pic.0563.jpg`, `pic.0234.jpg`. These images share a similar color palette (browns, grays, greens) but are not necessarily buildings. This method focused on the low-level color and texture features.

  –

  –

  –

Analysis: For identifying a general category like "buildings," the DNN is superior. The classic method was distracted by the color scheme and failed to find other buildings.

Image 2: `pic.0948.jpg` (A close-up of a person)

Figure 4: pic.0329.jpg

Figure 5: pic.0701.jpg

Figure 6: pic.0563.jpg

Figure 7: pic.0234.jpg

Figure 8: pic.0948.jpg

- **DNN Matches:** `pic.0930.jpg`, `pic.0960.jpg`, `pic.0928.jpg`. All three top matches are also close-ups of people, demonstrating the DNN's ability to recognize high-level concepts.

Figure 9: pic.0930.jpg

  –
  –
  –

- **Texture-Color Matches:** `pic.0891.jpg`, `pic.1069.jpg`, `pic.0008.jpg`. These matches are based on color and texture similarity (e.g., skin tones, background colors) but are not all images of people.
  –
  –
  –

Is DNN always better? No. While DNNs excel at semantic understanding, they may fail if the desired similarity is based on a very specific low-level feature that the network wasn't trained to prioritize. For instance, if you wanted to find images with a specific cross-hatch texture pattern, a dedicated texture histogram might outperform a general-purpose DNN. However, for most common CBIR

Figure 10: pic.0960.jpg

Figure 11: pic.0928.jpg

Figure 12: pic.0891.jpg

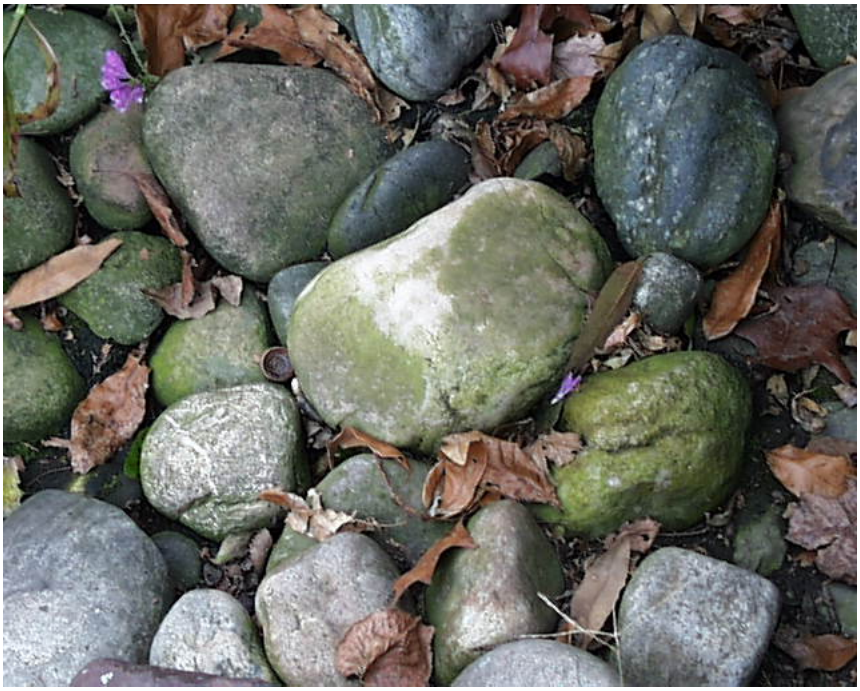


Figure 13: pic.1069.jpg

Figure 14: pic.0008.jpg

tasks, DNN embeddings provide a much more robust and semantically relevant starting point.

Task 7: Custom Design

Target Image 1: `olympus/pic.0535.jpg` **(Red Flower)**
* **Top 5 Matches:** 1. `olympus/pic.0536.jpg` 2. `olympus/pic.0543.jpg` 3. `olympus/pic.0544.jpg` 4. `olympus/pic.0534.jpg` 5. `olympus/pic.0545.jpg`

Target Image 2: `olympus/pic.0897.jpg` **(Sunset)**
* **Top 5 Matches:** 1. `olympus/pic.0142.jpg` 2. `olympus/pic.0964.jpg` 3. `olympus/pic.0122.jpg` 4. `olympus/pic.0918.jpg` 5. `olympus/pic.0417.jpg`

Extensions

Interactive GUI

An interactive GUI was developed for the project as an extension. This GUI, built with the ImGui library, provides a user-friendly interface for the CBIR system.

Features: * **Image Browser**: A "Browse..." button opens a native file dialog, allowing the user to select any target image from the file system. * **Interactive Controls**: Dropdown menus are used to select the matching algorithm and the number of results to display. * **Visual Feedback**: The selected target image and the corresponding matched images are displayed visually in the application window, along with their calculated distances. * **Real-time Interaction**: The interface includes an "Execute Search" button to run queries on demand and a "Close" button to exit the application.

GUI Screenshot *(Note to user: Please take a screenshot of the running GUI and save it as* `GUI_Screenshot.png` *in the project root for this image to be displayed.)*

Additional Matching Methods

Several new feature matching methods were added as extensions:

- **Banana Finder**: A simple color-based detector that ranks images by the percentage of yellow pixels they contain. This is effective for finding images with prominent yellow objects.
- **Blue Trash Can Finder**: Similar to the banana finder, this method ranks images by the percentage of a specific shade of blue, designed to find the blue trash cans present in the dataset.
- **Face Detector**: This method uses an OpenCV Haar Cascade classifier to detect faces in images. It ranks images based on the number of faces detected compared to the target image.
- **Gabor Filter (Texture)**: This method uses a bank of Gabor filters with different orientations and frequencies to create a feature vector describing the texture of an image. It is effective for retrieving images with similar textural patterns.
- **Custom DNN (ResNet18)**: This extension uses a utility program (`generate_embeddings`) to process the entire image database through a local ONNX ResNet18 model, creating a new `Custom_ResNet18_olym.csv` file. The GUI can then use this custom-generated set of embeddings for matching, allowing for direct comparison with other pre-computed feature sets.

Time Travel Days

3 days used.

Reflection

This project served as a practical exploration of various content-based image retrieval techniques. Implementing classic methods like baseline pixel matching and color histograms highlighted the trade-offs between computational simplicity and descriptive power. The introduction of texture and spatial information in later tasks demonstrated a clear improvement in retrieval accuracy for certain image types.

The most significant learning came from integrating and comparing these classic methods with deep learning embeddings. The ResNet18 features often provided more semantically meaningful matches, capturing abstract similarities that pixel- and color-based methods could not. This emphasized the power of pre-trained deep networks as robust, general-purpose feature extractors.

The process of building, debugging, and verifying the C++ application provided valuable software engineering experience, particularly in managing dependencies like OpenCV and handling file I/O for the image database and CSV embeddings.

Acknowledgements

- **Professor Bruce Maxwell** and the course materials for CS5330, which provided the project specification, sample code, and theoretical foundation.
- **OpenCV Documentation:** For references on histogram calculation, Sobel filters, and other image processing functions.
- **Shapiro and Stockman, "Computer Vision":** For concepts related to histogram matching.
- **An AI assistant (Gemini):** Was used to help write and debug code, implement the ImGui interface, and for project documentation.