Project 2 Report

Name: Priyanshu Ranka (NUID: 002305396)

Semester: Spring 2025

Professor: Prof. Bruce Maxwell

Course: Pattern Recognition and Computer Vision

1. Project Overview

This project focuses on **Content-Based Image Retrieval (CBIR)** by leveraging various **image descriptors** and **distance metrics** to identify visually similar images from a database. The tasks implemented progressively improve retrieval accuracy by utilizing different feature extraction methods and similarity measures.

Implemented Tasks:

1. Baseline Matching (7×7 Central Patch & SSD) – Uses a small central image patch as the feature vector and compares images using the *Sum of Squared Differences (SSD)*.
2. Histogram Matching – Computes *color histograms* and uses *histogram intersection* for similarity comparison.
3. Multi-Histogram Matching – Incorporates *regional histograms* for a more robust color-based similarity measure.
4. Texture and Color-Based Matching – Combines *whole-image color histograms* with *Sobel-based texture histograms* for enhanced retrieval accuracy.
5. Deep Network Embeddings (ResNet18) – Utilizes *precomputed 512-dimensional feature vectors* from a *ResNet18 deep network* trained on ImageNet and applies *SSD for similarity measurement*.
6. Comparison of Deep Learning & Classical Features – Evaluates the retrieval performance of ResNet18 embeddings against color- and texture-based methods.
7. Custom Design (CBIR for Shoes) – Develops a specialized *custom retrieval system* tailored for a specific image category.

Each successive approach refines image similarity calculations, leading to more *accurate* and *meaningful* retrieval results.

2. Task Demonstrations

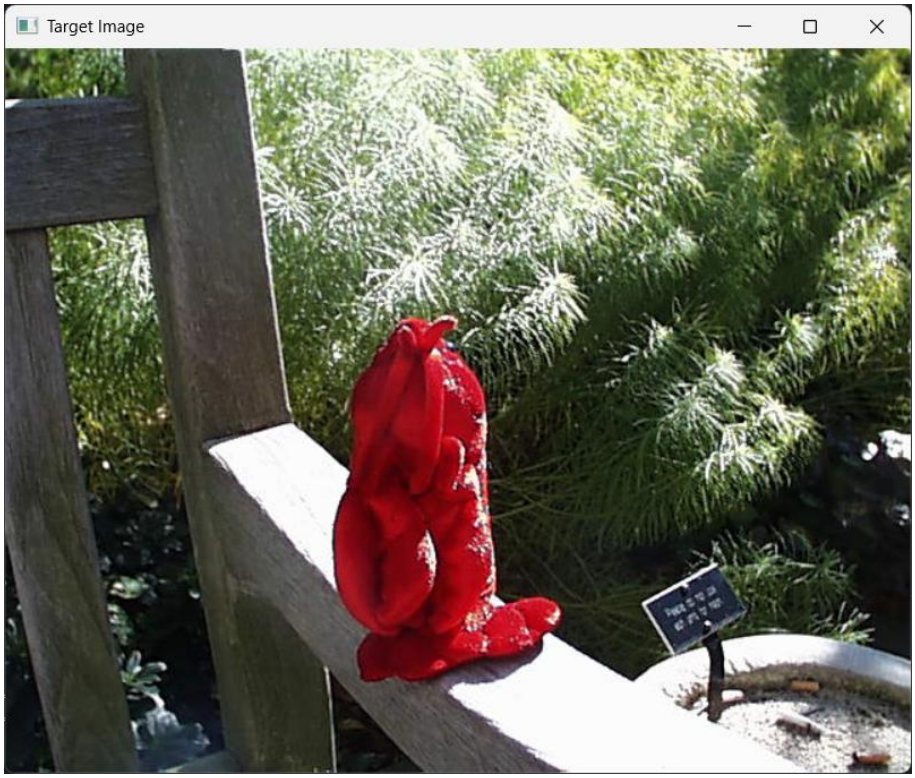
2.1 Baseline Matching

Objective: Implement a basic image retrieval system using the *7x7 square* from the center of each image as the feature vector and the *Sum of Squared Differences (SSD)* as the distance metric. This involves several steps: Extract the 7x7 patch from the center of each image. Implement the SSD calculation *without* using any OpenCV built-in functions. Ensure that comparing an image with itself results in an SSD of 0. Develop a program (either a single program or two separate programs) that reads a target image, computes its 7x7 feature vector, iterates through a directory of images, computes each image's feature vector and its SSD to the target, and outputs the top N matching images based on the SSD values. This combined task is crucial for establishing the foundation of your image retrieval system.

Approach: The steps of the approach are

1. Extract simple 7x7 pixel features from the target and database images.
2. Use SSD to measure the dissimilarity between these features.
3. Rank the database images based on their SSD to the target image.
4. Display the target image and the top N most similar images.

Results:



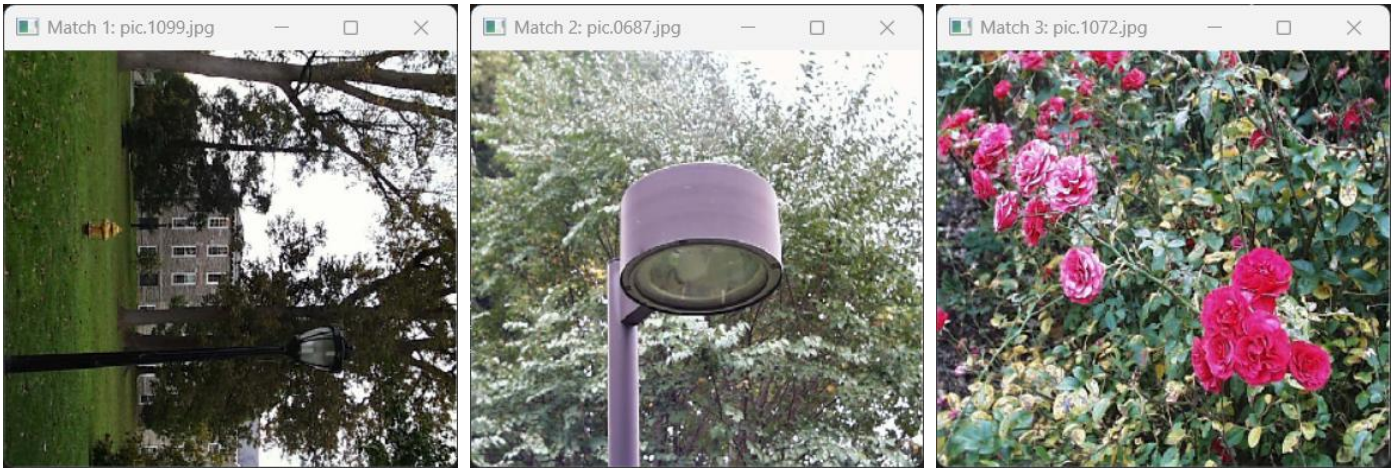


Figure 2.1 (a): (above) The target image is pic.1016.jpg. (below) The top 3 matches.

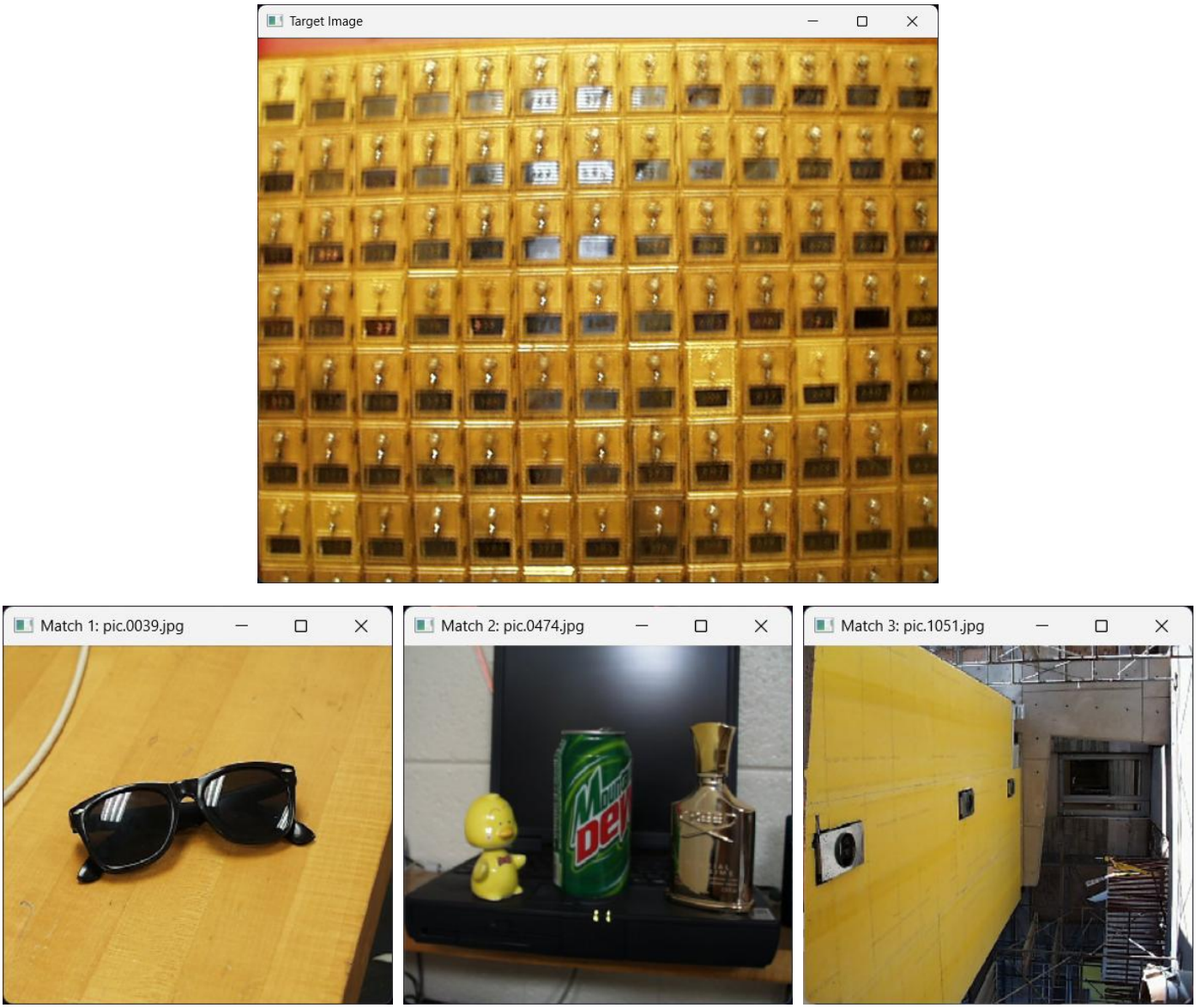


Figure 2.1 (b): The target image and its top 3 matches.

-> The above are the results for the baseline matching. As we can see, the results are not very accurate and the way matches are made doesn't make a lot of sense since it only checks for the central patch and not the whole image as it should.

2.2 Histogram Matching

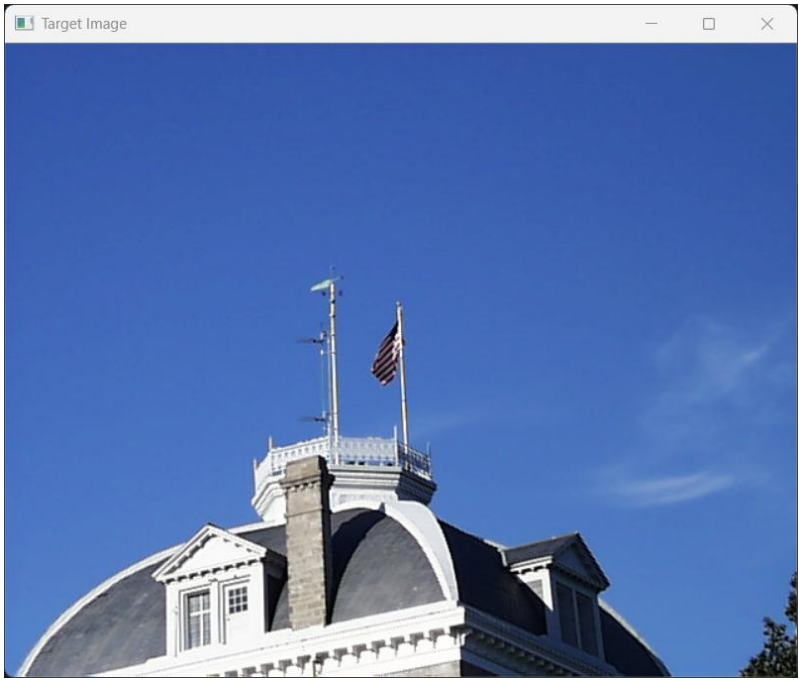
Objective: Implement image retrieval using a 2D or 3D color histogram as the feature vector and histogram intersection as the distance metric. Write your own histogram calculation and intersection code. Normalize histograms before intersection.

Approach: The *computeHistogram* function calculates a 2D color histogram using *rg chromaticity*. It converts the image to float, splits the RGB channels, calculates normalized red and green values *(r_norm, g_norm),* and then uses *calcHist* to compute the 2D histogram. The histogram is then normalized and flattened into a vector.

The *computeHistogramIntersection* function calculates the intersection between two histograms. It iterates through the bins and sums the minimum value between the corresponding bins of the two histograms.

The main function loads a target image, computes its histogram, iterates through a database directory, computes histograms for each image in the database, calculates the histogram intersection between each database image's histogram and the target image's histogram, sorts the results by similarity, and displays the top *N* matches. It uses *filesystem::directory_iterator* to traverse the database directory. It normalizes the histograms and uses the intersection value as a similarity measure (higher is more similar). It then displays the target image and the top N matched images.

Results:



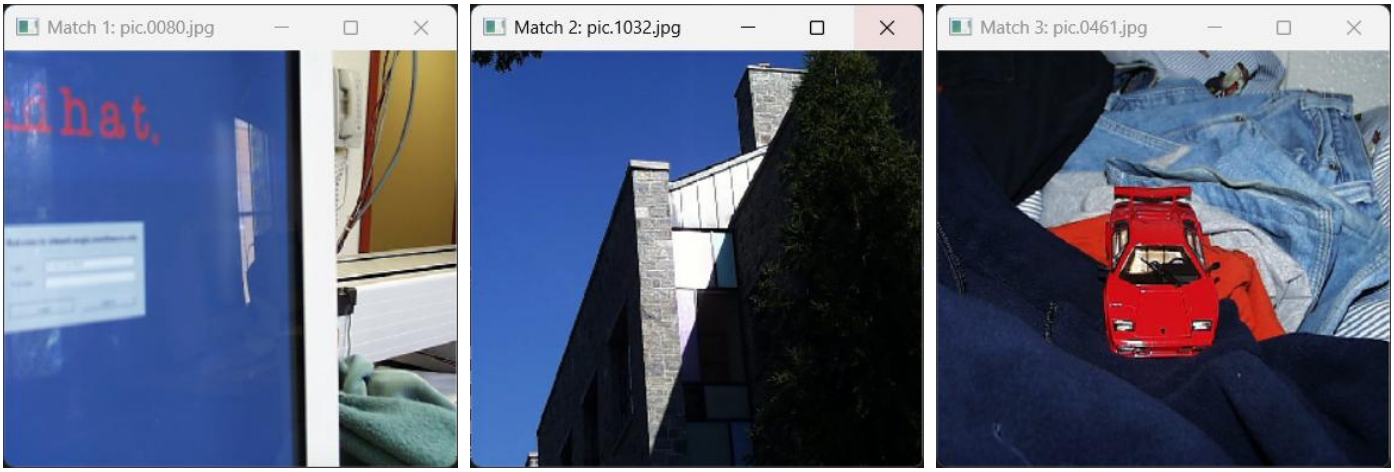


Figure 2.2 (a): (above) The Target image, pic.0164.jpg. and the top 3 matches – pic.0080, pic.1032 and pic.0461 respectively.

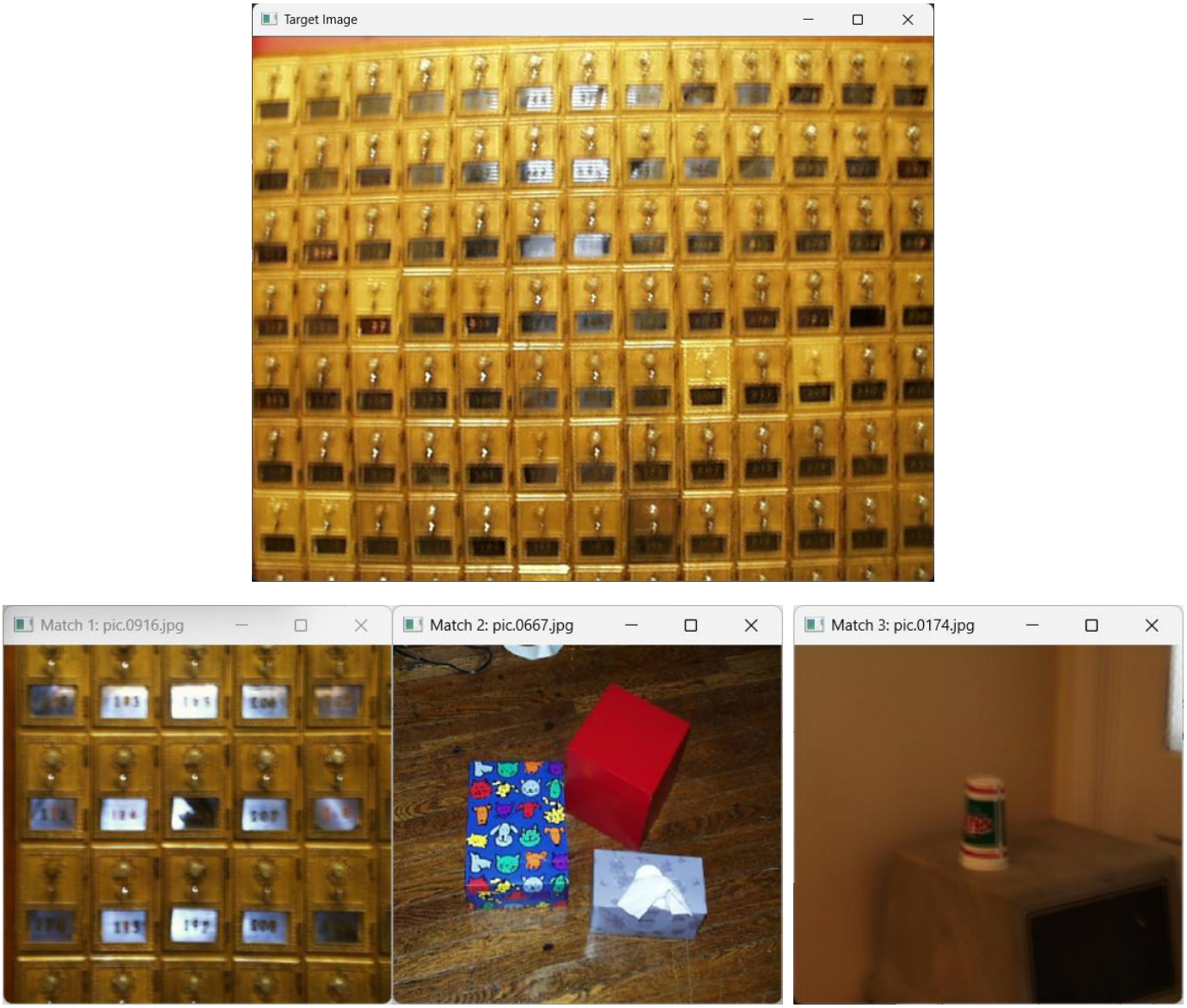


Figure 2.2 (b): (above) Target image, pic.0218.jpg and the top 3 matches.

-> The results of this method are very much better than the previous method. This is because it matches the color histograms of the images rather than just depending on the middle 7x7 section of the images.

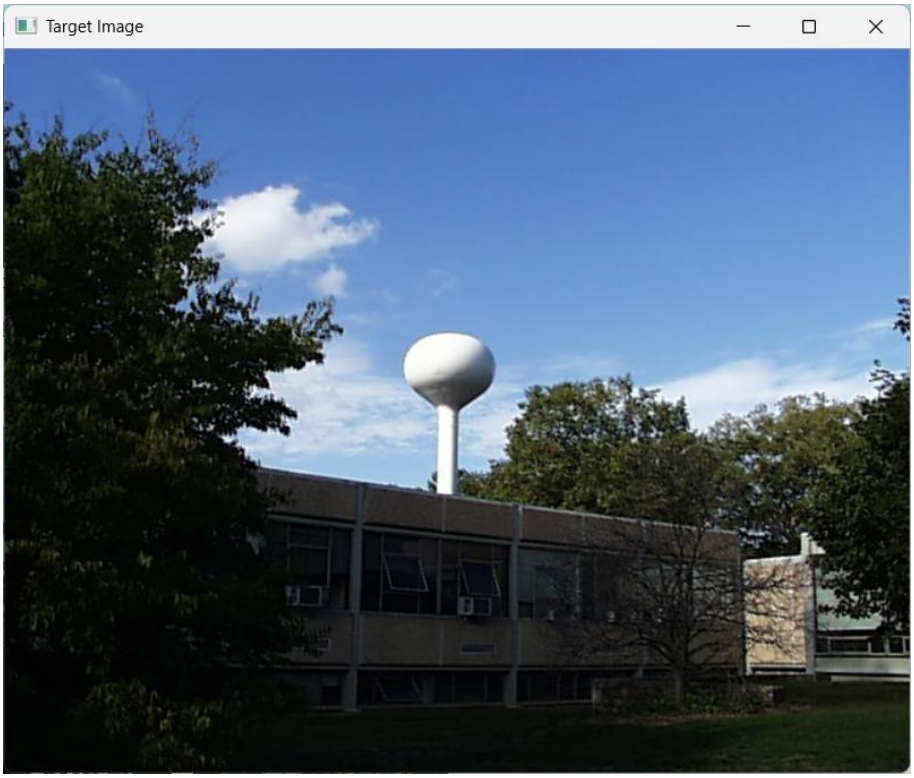
2.3 Multi-histogram Matching

Objective: Use multiple color histograms from different image regions (e.g., top/bottom) as the feature vector. Design a custom distance metric (e.g., weighted average of histogram intersections) to combine regional distances.

Approach: The approach can be broken down into these steps:

1. Regional Histogram Computation: The code divides each image into *two overlapping horizontal regions (upper 2/3 and lower 2/3)*. For each region, it calculates a 3D RGB (8bins each) color histogram using *calcHist* and normalizes the histogram. While a 3D histogram is calculated, it's treated as a 1D vector during the intersection calculation due to the way *min()* and *sum()* are used on the *cv::Mat* objects.
2. Histogram Intersection: The code calculates the intersection between corresponding region histograms of two images. The intersection is calculated by taking the element-wise minimum of the two histograms (effectively treating them as 1D vectors) and then summing these minimum values.
3. Weighted Similarity Score: A custom similarity score is computed by taking a weighted average of the histogram intersection values for the two regions. Equal weights are used in this implementation. This weighted average combines the color information from both regions into a single similarity measure.
4. *Image Retrieval:* The system loads a target image and computes its regional histograms. It then iterates through a database of images, computing regional histograms and similarity scores for each database image relative to the target image. The images are ranked based on their similarity scores, and the top N matches are displayed.

Results:



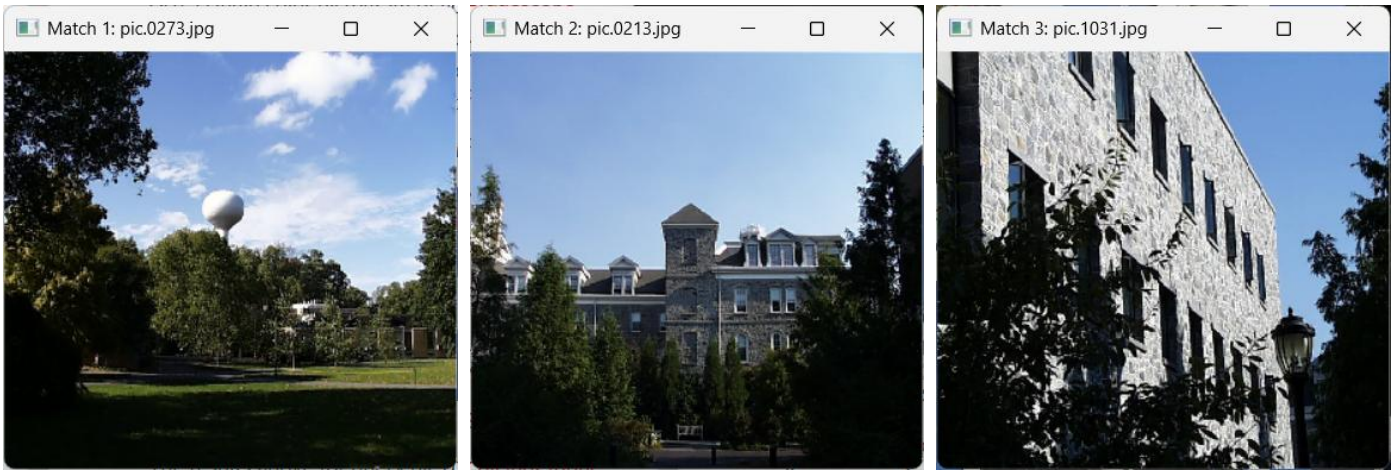
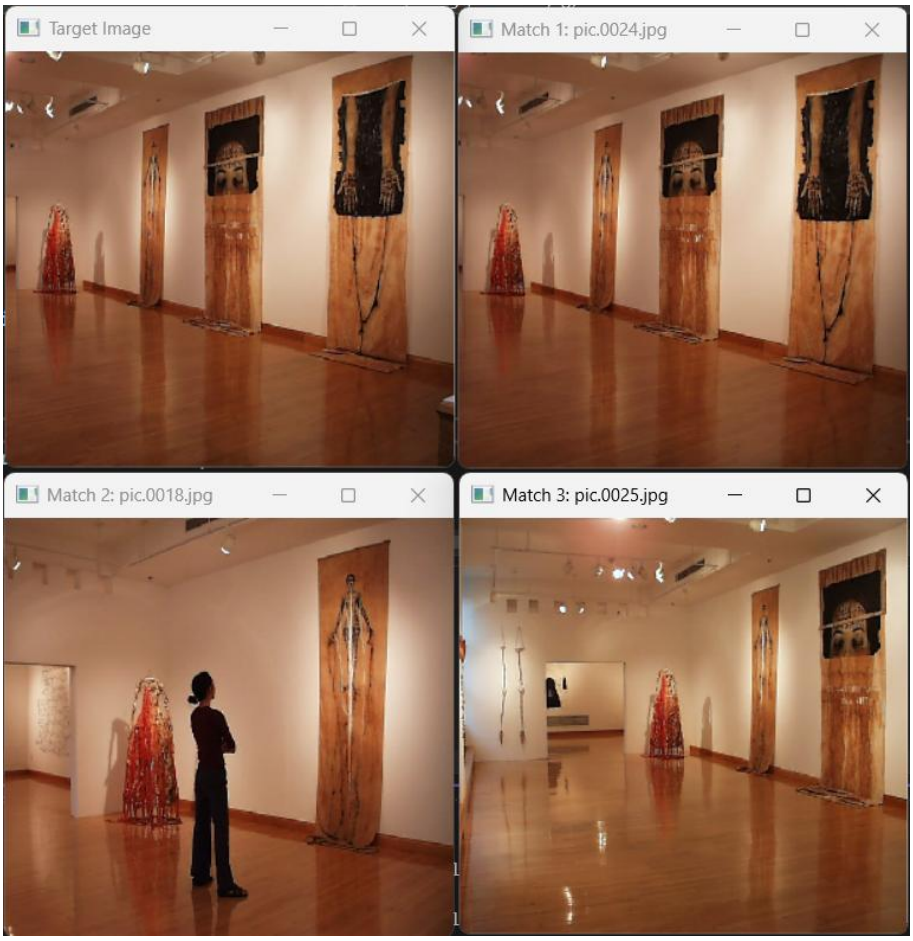


Figure 2.3 (a): (above) The target image, pic.0274.jpg and the top 3 matches, pic.0273, pic.0213, pic.1031.

Figure 2.3 (b): (below) The target image, pic0013, and the top 3 matches – pic.0024, pic.0018, pic.0025 respectively.



-> It is very clear that the result of this method surpasses the previous two. This is because instead of comparing the features from just a single histogram, this method appends the features from 2 different histogram results and then compares it. Thus, the matches made are better.

2.4 Texture and Color

Objective: Combine whole-image color and texture histograms as the feature vector. Choose a texture metric (e.g., Sobel magnitude histogram). Design a distance metric with equal weighting for color and texture.

Approach: The image information is stored in ImageData structure which has its name, color histogram values and texture histogram values. The code uses a combined color and texture-based approach for image retrieval. It calculates normalized histograms for color (HSV color space) and texture (Sobel magnitude) using *getColorHistogram(const Mat& image)* and *getTextureHistogram(const Mat& image)*. It uses the Sum of Squared Differences (SSD) as the distance metric. The use of normalized histograms is crucial for ensuring that the comparison is not biased by image size. The combination of color and texture features allows for a more comprehensive representation of the image content. The code then retrieves the top N images with the smallest distances to the target image. This is done by find TopNMatches() function and then displayImages() function is used to display the images.

Results:

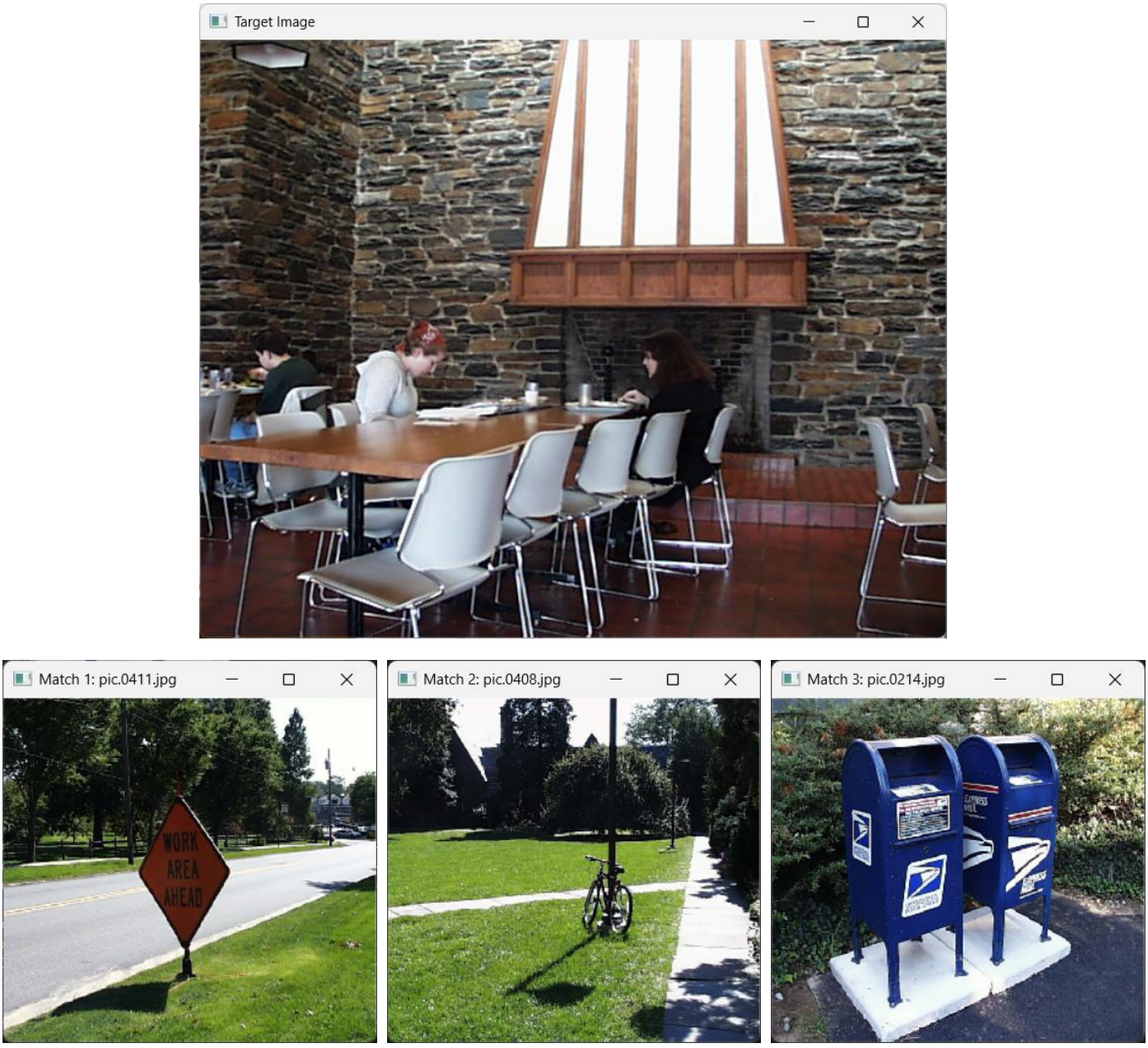


Figure 2.4(a): The target image, pic.0535, and the top 3 matches – pic.0411, pic.0408, pic.0214 respectively.

-> The results of this method are rather unclear. According to the conventions, the results should have improved but here the results are rather dull. If we take the same images using method 2 or 3, we tend to get better results as they focus better on the edge changes because histograms allow us to do that in a better way.

Comparison:

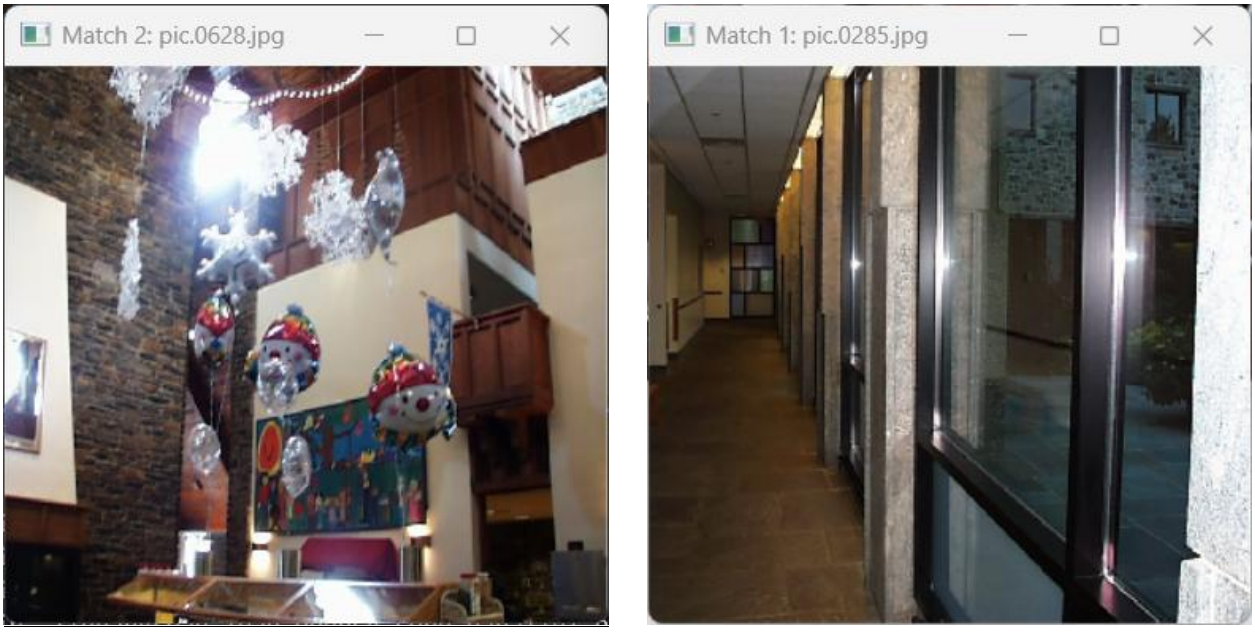


Figure 2.4(b): The above are the results of the same target image that method 4 uses.

(right) Method_2 (left) Method_3

We can see how the results of these arent as accurate as well. But in comparison to mrthod_4, they are superior. They display the edges where brown and white colors meet very well. In actuality the results of method 4 should have been better but for this example it is not the case.

2.5 Deep Network Embedding

Objective: Use provided ResNet18 embeddings from a CSV file for image retrieval. Choose a distance metric (SSD or cosine distance). Retrieve target image feature vector from the file, compare with database vectors, and output top N matches.

Approach: This approach utilizes the feature vectors provided in the in CSV file, where each row contains the Image name and 512 feature vector.

1. *Read Feature Vectors from CSV:* function is used to extract the filenames and their corresponding 512 features from ResNet18.CSV. The data structure is {filename, features} pairs.
2. *Extract target Image Features:* The feature vector for the target image is retrieved from the dataset using *getTargetFeatures() function*.
3. *Computing the SSD:* Computation of SSD, with help of *computeSSD()* function, from the target image's feature vector and all other images in the data set. The results are stored in (distance, filename) pair.
4. *Sorting and Retrieving the Top 3:* Once the list is ready, it is sorted in ascending order and the top 3 images are returned. displayImages() function is used for this task.

Results:

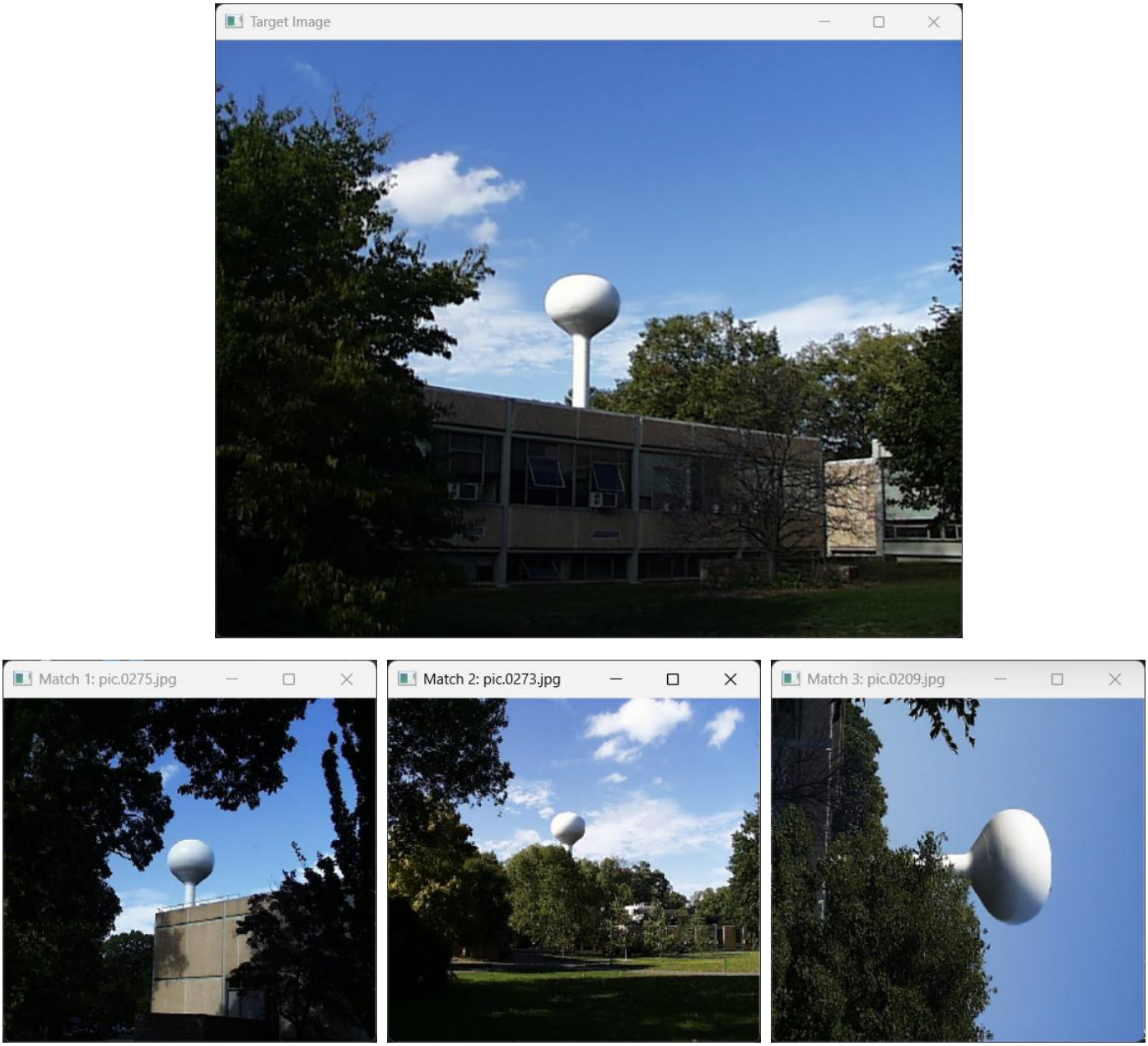


Figure 2.5 (a): The target image, pic.0274, and the top 3 matches – pic.0275, pic.0273, pic.0209 respectively.

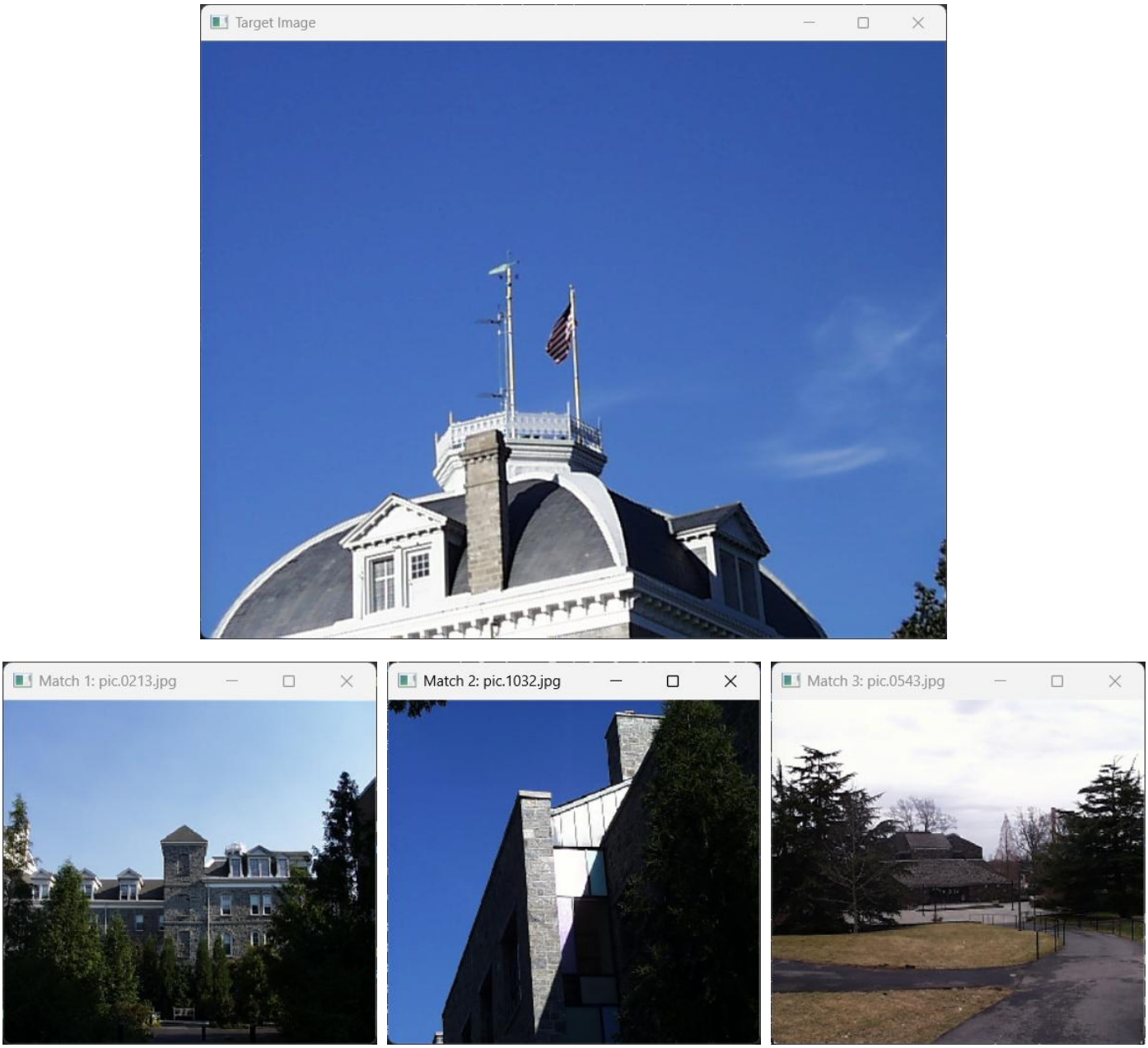


Figure 2.5 (b): The target image, pic.02164, and the top 3 matches – pic.0213, pic.1032, pic.0543 respectively.

Figure 2.5 (c): The target image, pic.0893, and the top 3 matches – pic.0897, pic.0136, pic.0885 respectively.

-> Comparing the results of this method with previous ones, this method outstands them. For the first example, Figure 2.5(a), the same target image was used in method 3. We see that one of the matched images is the same, but the rest are different. The result of this method gives more similar images than the previous one. For Figure 2.5(b) the target image is the same as used in Histogram matching. Here is also one of the matched images matches but the rest of the results are not like the target image. In comparison, the results we get from using DNN matching are far more better.

2.6 Compare DNN Embeddings and Classic Features

Objective: Compare image retrieval results using ResNet18 embeddings versus classic features (color, texture). Choose 2-3 images, visualize top matches for both methods, and analyze similarities, differences, and relative performance.

Solution: Let's compare both methods first

Classic Features:

Advantage- It is computationally efficient and is good for specific tasks, usually low level ones, like object detection.

Disadvantage- It is sensitive to variations in lightning, scale, rotation etc. It might not capture the high0level semantics information like DNN does.

Use Case- Used for image retrieval based low level visual features and object recognition in good conditions.

DNN Embeddings:

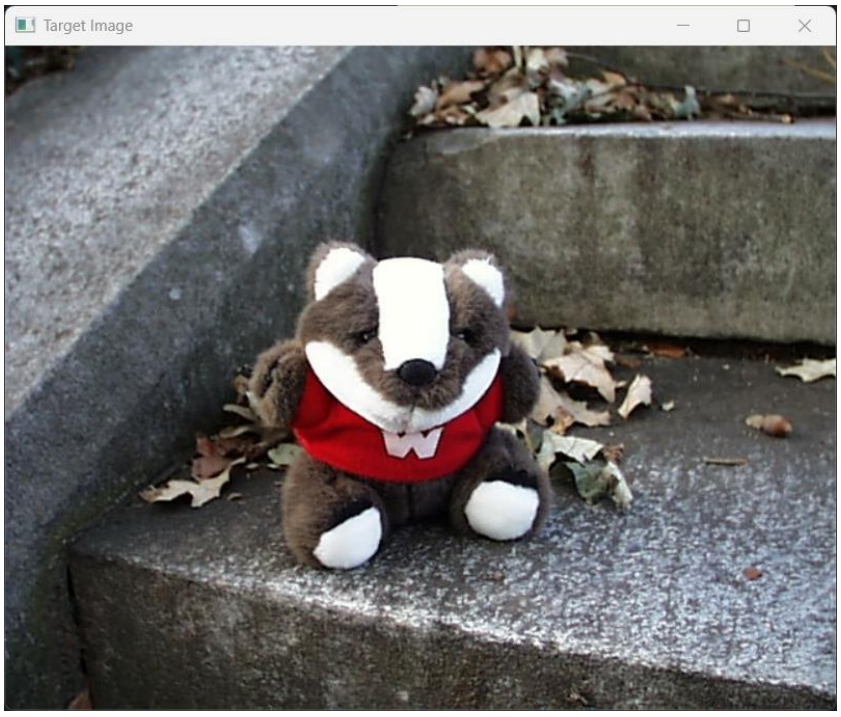
Advantage- Captures the high level visual and semantics information, robust to variations in lightning, rotations and scale variations.

Disadvantage- Computationally expensive and requires large training data sets.

Use case- Image retrieval based on similar semantics, CBIR, image classification etc.

From the above, one's advantages are others disadvantages and vice versa.

Comparison: The comparison of the results of the same target image using both these methods is as follows



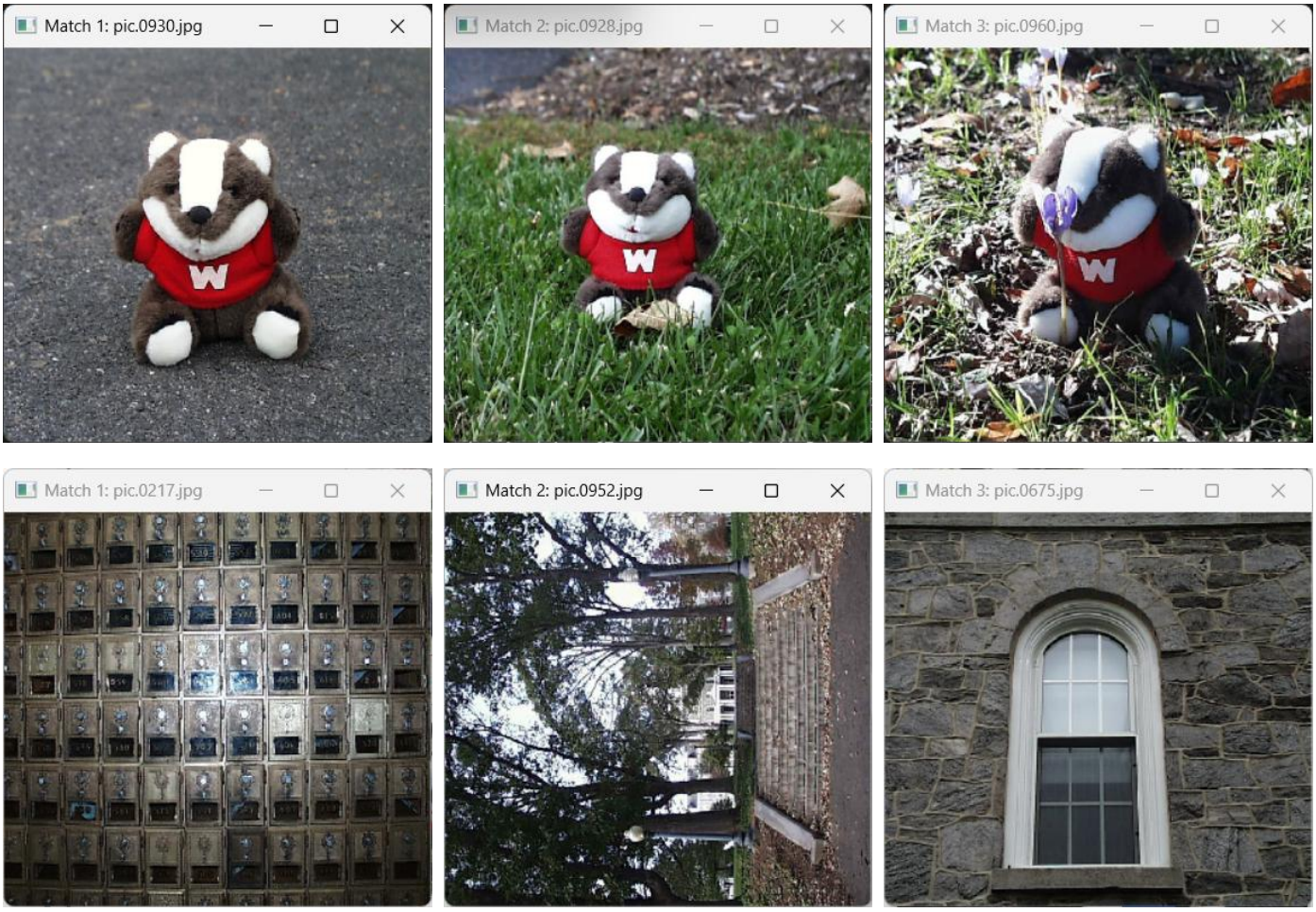


Figure 2.6: (top) Target image, pic.948. (middle) Top 3 results from DNN Matching.

(bottom) Top 3 results of Classic method (histogram matching)

2.7 Custom Design

Objective: Design a custom CBIR system for a specific image category ("Shoe" in this case). Create a feature vector combining multiple features (including, optionally, DNN embeddings, but not exclusively). Design a custom distance metric. Implement retrieval and evaluate performance.

Approach: The approach for this one can be described by the functions in it.

Image Representation: *ImageData* struct stores filename, ResNet18 features, and image data for each image.

Color Histogram: *getColorHistogram* calculates and normalizes the HSV color histogram from the image data.

CSV Reading: *readCSV* reads pre-computed ResNet18 features and filenames from a CSV file, also reading the images themselves.

Target Features: *getTargetFeatures* retrieves and combines the ResNet18 features and the color histogram for the target image.

Similarity (SSD): *computeSSD* calculates the Sum of Squared Differences between feature vectors.

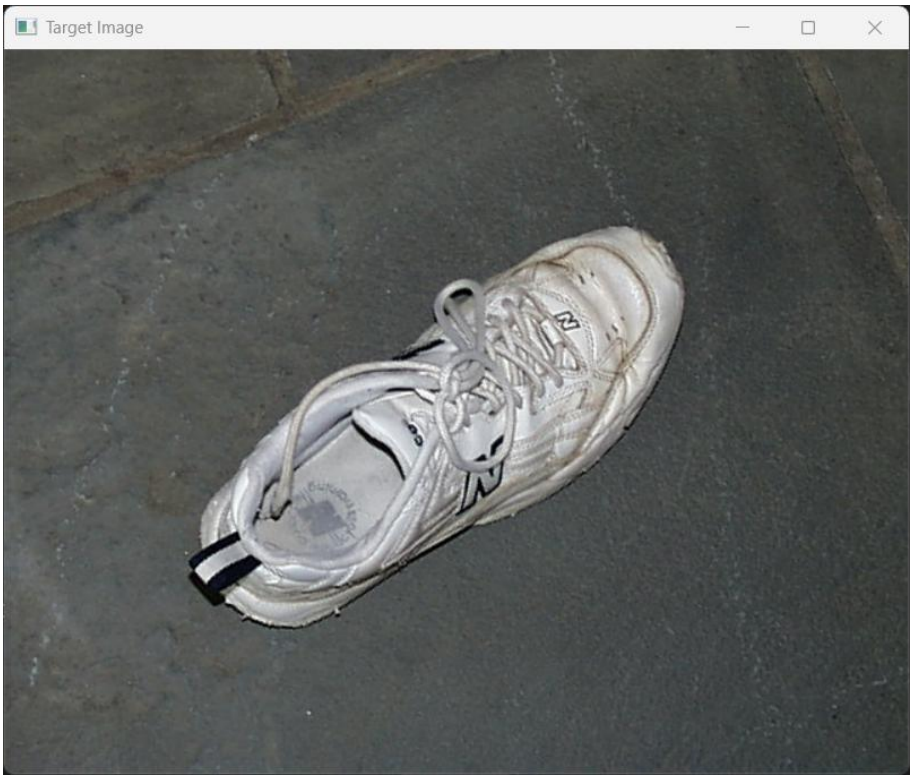
Top Matches: *findTopMatches* finds top N matches by calculating SSD between target and database images, skipping the target image itself.

Display: *displayImages* displays the target image and the top N matched images.

Main: *main* handles input, data loading, target feature retrieval, finding matches, and displaying results.

Steps:

1. main receives target image path and N.
2. readCSV loads image data and ResNet18 features from CSV.
3. getTargetFeatures combines target image's ResNet18 features and color histogram.
4. findTopMatches calculates SSD between target and database images' combined features.
5. findTopMatches sorts results by SSD.
6. displayImages shows target and top N matches.



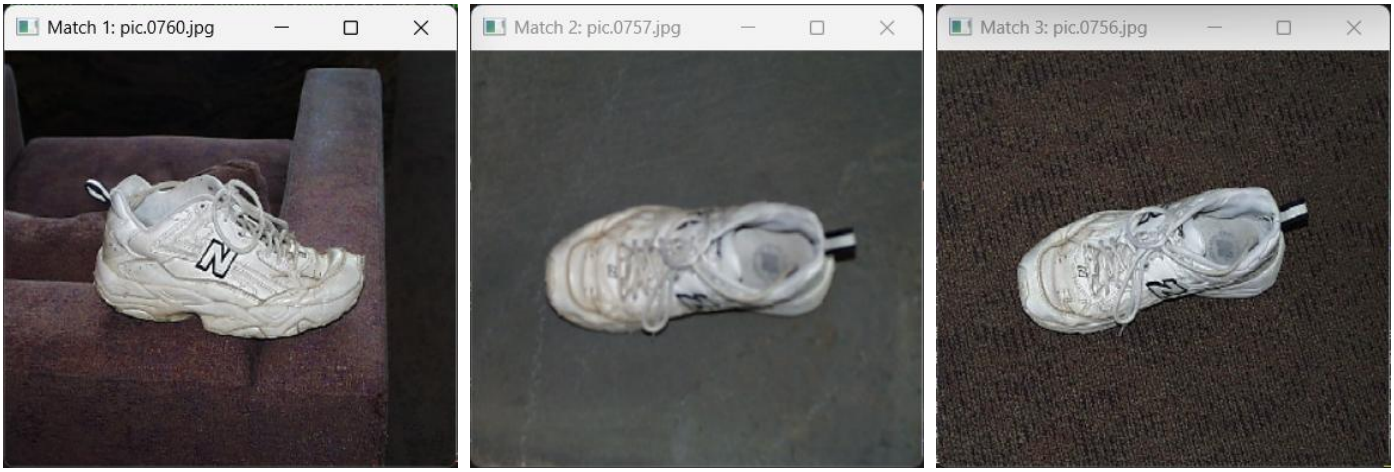


Figure 2.7: The target image, pic.0758, and the top 3 matches – pic.0760, pic.0757, pic.0756 respectively.

3. Reflections/Learnings

Task 2.1: Baseline Matching (Central Patch & SSD)

- Simple approaches like SSD on a small patch do not perform well, as they ignore global image structure.
- Limitation: Only the central region is compared, leading to inaccurate matches.

Task 2.2: Histogram Matching

- Color histograms provide a better representation of images than small patches.
- Histogram Intersection improves retrieval but struggles with texture variations.

Task 2.3: Multi-Histogram Matching

- Using multiple histograms from different regions provides a more comprehensive representation.
- It outperforms single-histogram retrieval by capturing both foreground and background color distributions.

Task 2.4: Texture and Color-Based Matching

- Combining color and texture histograms improves results, as textures capture structural information.
- Texture-based features (e.g., Sobel magnitude histogram) help distinguish images with similar colors but different textures.

Task 2.5: Deep Network Embeddings (ResNet18)

- Deep Learning-based features (ResNet18) outperform handcrafted features in retrieval.
- High-dimensional embeddings encode semantic similarity, leading to more meaningful matches.

- SSD works well, but Cosine Distance could be tested for better high-dimensional performance.

Task 2.6: Comparing ResNet18 vs. Classic Features

- ResNet18 embeddings retrieve more semantically relevant images than color/texture-based methods.
- Classical techniques (color/texture histograms) perform well for specific categories (e.g., matching images with similar color distribution), but fail in complex scenes.

Task 2.7: Custom CBIR for Shoes

- A domain-specific approach (e.g., for shoes) requires carefully designed feature vectors that may combine color, texture, and DNN embeddings.
- Selecting the right combination of features is critical for improving retrieval accuracy in category-specific CBIR systems.

4. Acknowledgements

I would like to express my gratitude to:

- **Professor Bruce Maxwell** for guidance and insightful lectures in **Pattern Recognition and Computer Vision (PRCV)**.
- **Teaching Assistants** for their support in understanding OpenCV and deep learning concepts.
- **Online Resources** (OpenCV documentation, research papers, and GitHub repositories) provided valuable insights into **image retrieval algorithms and deep network embeddings**.

This project has been an enriching experience, deepening my understanding of **image retrieval, feature extraction, and deep learning-based similarity search**.