CRISP--CONTENT-BASED-RAPID-IMAGE-SEACH-USING-PARALLEL-COMPUTING

Team members:

B.DHEERAJ REDDY 22BCE1257
K.SHANMUKH SAI 22BCE5011
KRISHNA 22BCE1678

Project Explanation: Content-Based Image Retrieval (CBIR) with Sequential and Parallel Processing

*This project implements a **Content-Based Image Retrieval (CBIR) system**, which searches for similar images based on their visual features rather than metadata (like filenames or descriptions). The system extracts features from images and compares them using the **Chi-Square Distance** metric. Additionally, it compares **sequential processing** with **parallel processing** to evaluate speedup.*

Key Components:

1. Image Preprocessing (preprocess_image(image_path))

- *Reads an image using OpenCV.*
- *Resizes it to **500x500** for consistency.*

```python
# Resize the image
if original_image is not None:
    resized_image = cv2.resize(original_image, (500, 500))
    rgb_original = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
    rgb_resized = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)
    print("\nInput vs Output:")
    display("Original Image", rgb_original, "Resized Image", rgb_resized)
```

Python

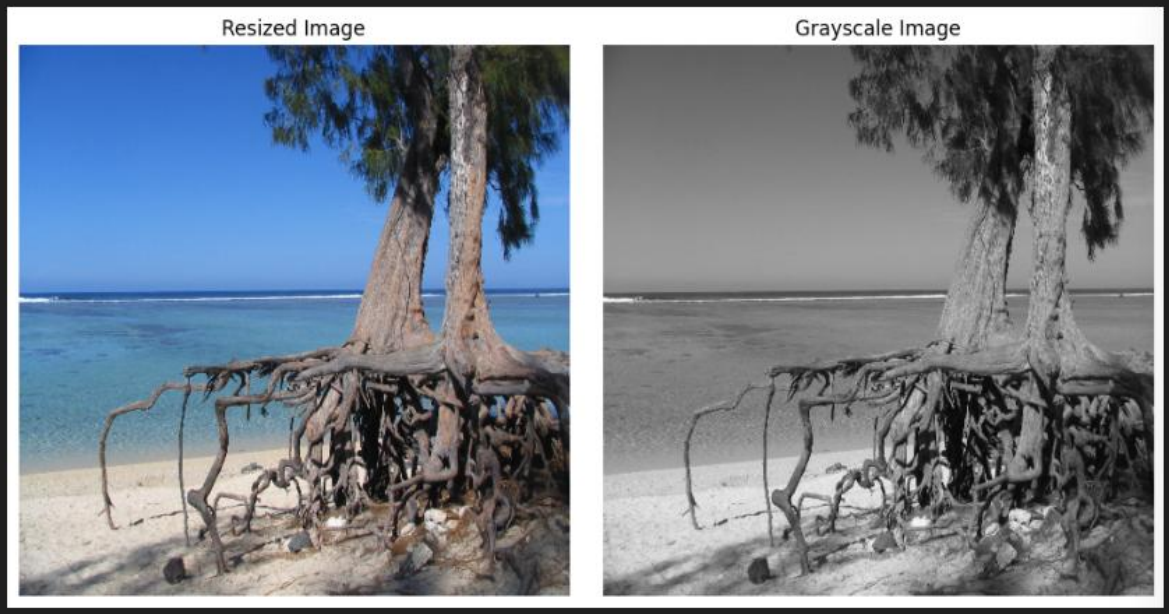
```
Input vs Output:
```

- *Converts it to **grayscale**.*

```python
#  Convert the resized image to grayscale
if 'resized_image' in locals():
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    rgb_resized = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)
    print("\nInput vs Output:")
    display("Resized Image", rgb_resized, "Grayscale Image", gray_image, cmap_out='gray')
```

Python

```
Input vs Output:
```



- *Applies **Gaussian Blur** to remove noise.*

```python
#  Apply Gaussian blur to the grayscale image
if 'gray_image' in locals():
    blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
    print("\nInput vs Output:")
    display("Grayscale Image", gray_image, "Blurred Image", blurred_image, cmap_in='gray', cmap_out='gray')
```

Python

```
Input vs Output:
```

- *Enhances edges using **Laplacian sharpening**.*

```python
# Apply Laplacian sharpening
if 'gray_image' in locals() and 'blurred_image' in locals():
    laplacian = cv2.Laplacian(blurred_image, cv2.CV_64F)
    sharpened_image = cv2.subtract(gray_image.astype(np.float64), 0.5 * laplacian)
    sharpened_image = np.clip(sharpened_image, 0, 255).astype(np.uint8)
    print("\nInput vs Output:")
    display("Grayscale Image", gray_image, "Sharpened Image", sharpened_image, cmap_in='gray', cmap_out='gray')
```

Python

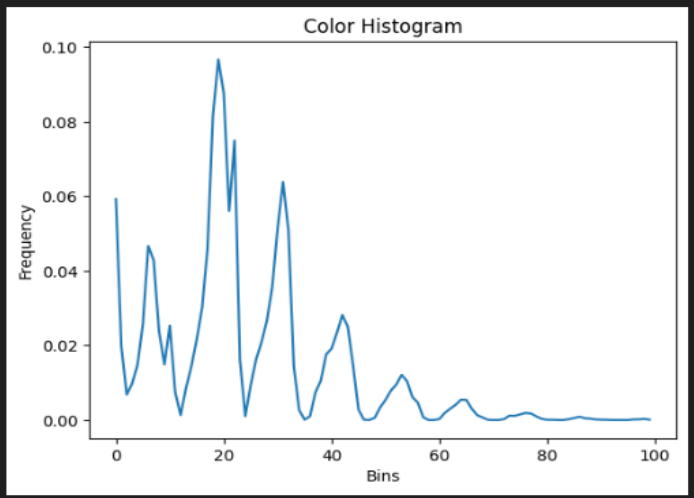
```
Input vs Output:
```

2. Feature Extraction (extract_features(image))

- *Extracts a **color histogram** (12x12x12 bins) to capture color distribution.*

```python
# Display the color histogram
if 'color_hist' in locals():
    plt.figure()
    plt.title("Color Histogram")
    plt.xlabel("Bins")
    plt.ylabel("Frequency")
    plt.plot(color_hist[:100])
    plt.show()
```

Python



```python
# Extract the color histogram (no direct image output)
if 'resized_image' in locals():
    def extract_color_histogram(image, bins=(12, 12, 12)):
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        hist = cv2.calcHist([hsv], [0, 1, 2], None, bins, [0, 180, 0, 256, 0, 256])
        cv2.normalize(hist, hist)
        return hist.flatten()

    color_hist = extract_color_histogram(resized_image)
    rgb_resized = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)
```
[70] Python

- *Computes **Hu Moments**, which help describe the shape of objects in an image.*

```python
# Extract Hu Moments
if 'gray_image' in locals():
    def extract_hu_moments(image):
        moments = cv2.moments(image)
        hu_moments = cv2.HuMoments(moments).flatten()
        return -np.sign(hu_moments) * np.log10(np.abs(hu_moments))

    hu_moments = extract_hu_moments(gray_image)
    print("\nOutput: Hu Moments:", hu_moments)
```
[72] Python

```
Output: Hu Moments: [ 2.88156247  8.43503971 10.29492196 11.05178905 21.95152247 15.34317623
 21.81954781]
```

3. Similarity Measurement (chi2(histA, histB))

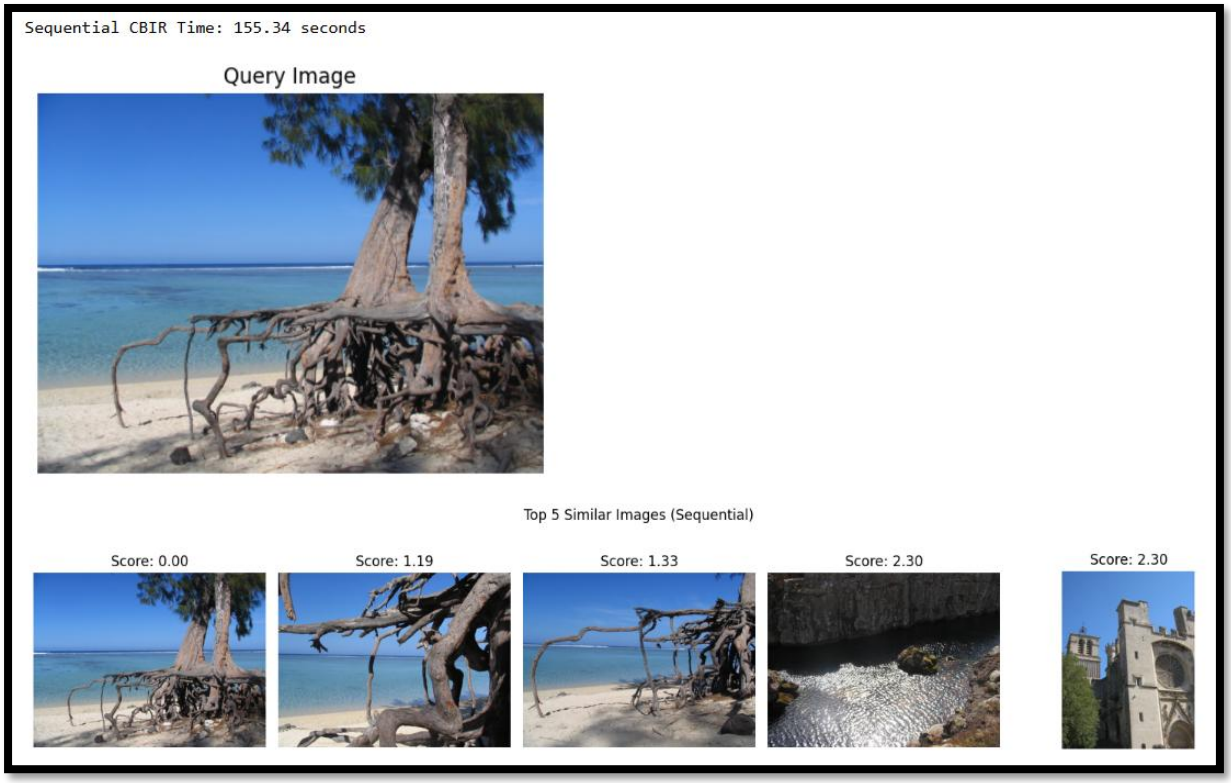
- *Uses the **Chi-Square Distance** to compare two feature vectors.*
- *A smaller distance means the images are more similar.*
- ***Lower Chi-Square score (closer to 0):** The images are **more similar**.*
- ***Higher Chi-Square score:** The images are **less similar**.*

```python
# Chi-Square Distance Function
def chi2(histA, histB, eps=1e-10):
    return 0.5 * np.sum(((histA - histB) ** 2) / (histA + histB + eps))
```
] Python

Implementation of CBIR in Two Approaches

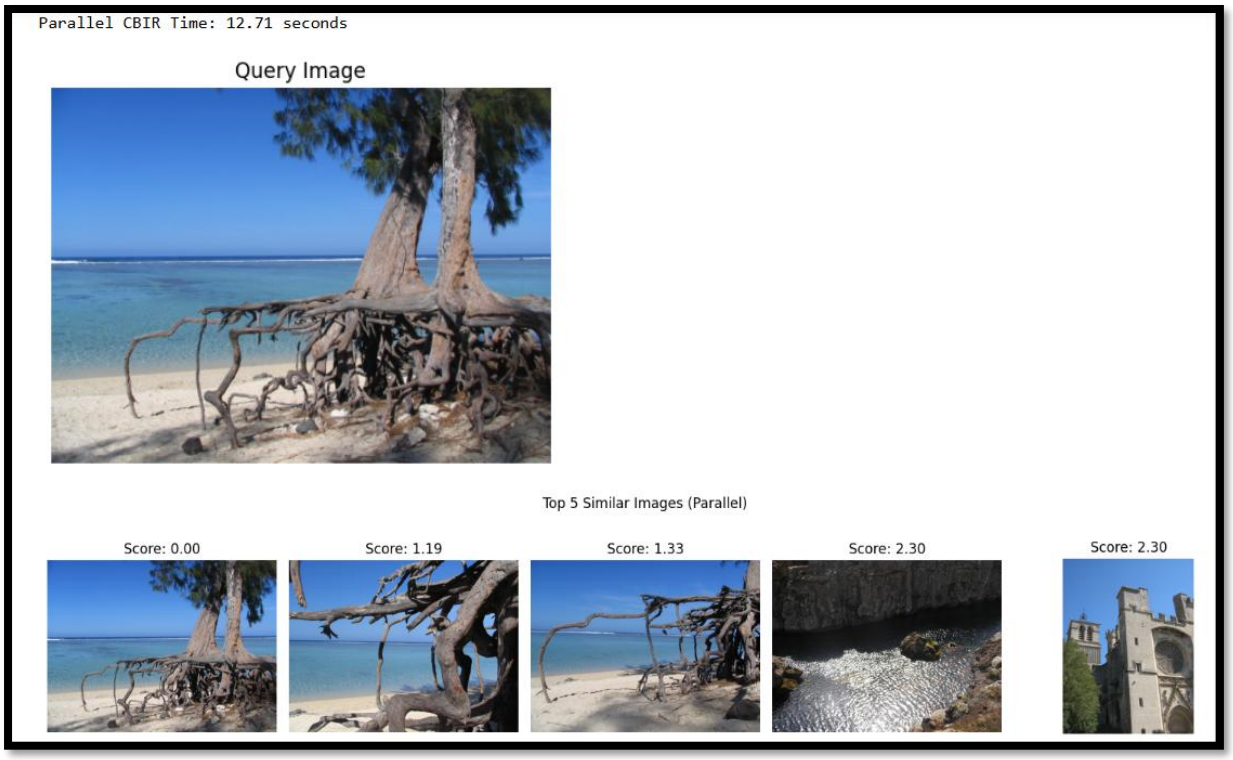
1. Sequential Processing

- *Reads all images one by one and extracts features.*
- *Computes distances between the query image and all dataset images.*
- *Sorts and returns the* ***top 5 most similar images****.*



2. Parallel Processing (Using ThreadPoolExecutor)

- *Uses **multithreading** to extract features from multiple images simultaneously.*
- *Computes similarity distances in parallel.*
- *This improves speed compared to the sequential approach.*



Performance Comparison

- *Measures the execution time for **both sequential and parallel approaches**.*
- ***Speedup** is calculated as: Speedup= Sequential Execution Time/Parallel Execution Time*
- *A higher speedup means parallel processing is more efficient.*

```python
### Speedup Calculation ###
speedup = seq_time / par_time
print(f"Speedup: {speedup:.2f}x")
```

Python

```
Speedup: 12.22x
```

Visualization

- *Displays the **query image**.*
- *Shows the **top 5 retrieved similar images** for both **sequential and parallel** approaches.*

- *Each retrieved image is displayed with its similarity score.*

Conclusion

- *The project* ***demonstrates the efficiency of parallel computing*** *in image retrieval tasks.*
- ***Parallel processing significantly reduces computation time****, making CBIR systems more scalable.*
- *This can be extended to* ***large-scale image databases*** *for applications like* ***image search engines, medical imaging, and facial recognition****.*

CODE :

https://drive.google.com/file/d/1Z0LtyceDtrMBtlarhQvuLE9TTl5fh0gq/view?usp=drive_link

DATASET:

https://www.kaggle.com/datasets/vadimshabashov/inria-holidays