Name: Prerna Kanekar

Roll no: 27

Experiment no 10

Aim:: To Create Program to perform a retrieving Image and Searching

Objective: The fundamental need of any image retrieval model is to search and arrange the images that are in a visual semantic relationship with the query given by the user.

Most of the search engines on the Internet retrieve the images based on text-based approaches that require captions as input.

Theory:

Image retrieval is the process of searching for and organizing images in a manner that reflects their visual content. Unlike text-based image retrieval, which relies on textual metadata such as captions or keywords, content-based image retrieval (CBIR) operates on the visual features of images themselves. This section will delve into the fundamental concepts and techniques that underlie a CBIR system.

1. Feature Extraction:In CBIR, images are represented using a set of features that capture their visual characteristics. These features can be low-level, like color histograms, texture descriptors, or more advanced, such as deep learning-based feature vectors.Convolutional Neural Networks (CNNs) are commonly used for feature extraction in modern CBIR systems. Pre-trained CNN models, like VGG, ResNet, or Inception, can transform images into high-dimensional feature vectors.Feature extraction aims to create a compact and meaningful representation of the image's visual content. This representation enables efficient comparison and retrieval.

2. Similarity Metrics:

To search for similar images, a similarity metric is employed to compare the feature vectors of the query image and database images.Common similarity metrics include cosine similarity, Euclidean distance, or Jaccard similarity, depending on the nature of the features used.Cosine similarity is often preferred for feature vectors as it measures the cosine of the angle between the vectors, providing a measure of their similarity without being sensitive to vector length.

3. Query Processing:When a user submits a query image, its features are extracted using the same methodology as the database images.These query features are then compared to the features of images in the database using the chosen similarity metric.

4. Ranking and Retrieval:The result of the similarity comparison is a list of database images ranked by their similarity to the query image.The images most similar to the query appear at the top of the list, providing an ordered retrieval result.

5. Challenges:

   - Image variability: Images can have variations in scale, viewpoint, lighting, and background, making it challenging to establish robust feature representations.
   - Scalability: Handling large image databases efficiently is a significant challenge. Indexing and retrieval speed become critical in large-scale systems.
   - Semantic gap: There may be a discrepancy between low-level visual features and high-level semantic content in images, which can affect retrieval accuracy.

6. Improvements:

   - Fusion of multiple features: Using multiple feature types and combining their results can enhance retrieval performance.
   - Relevance feedback: Allowing users to provide feedback on retrieved images to refine future searches.
   - Machine learning techniques: Utilizing machine learning models to learn feature embeddings and improve ranking algorithms.

Code:

```
import cv2
import numpy as np
import os


# Define the folder containing the images
image_folder = 'image_data'


# Function to extract image features (e.g., using ORB)
def extract_features(image_path):
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
orb = cv2.ORB_create()
keypoints, descriptors = orb.detectAndCompute(image, None)
return descriptors


# Function to search for similar images
def search_similar_images(query_image_path, image_folder):
query_features = extract_features(query_image_path)
# Store image paths and their respective scores
image_scores = []
for filename in os.listdir(image_folder):
if filename.endswith(('.jpg', '.png')):
image_path = os.path.join(image_folder, filename)
features = extract_features(image_path)
if features is not None:
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(query_features, features)
score = len(matches)
image_scores.append((image_path, score))
# Sort images by their scores
image_scores.sort(key=lambda x: x[1], reverse=True)
return image_scores


# Provide the path to your query image
query_image_path = 'query.jpg'


# Search for similar images
similar_images = search_similar_images(query_image_path, image_folder)


# Determine the dimensions for the resized images in the grid
image_width, image_height = 200, 200


# Calculate the number of rows needed for the grid
grid_width = 3 # Number of columns in the grid
grid_height = (len(similar_images) // grid_width) + 1


# Create an empty canvas for the grid
canvas = np.zeros((grid_height * image_height, grid_width * image_width,
3), dtype="uint8")
```

```
for i, (image_path, score) in enumerate(similar_images):
row = i // grid_width
col = i % grid_width
# Load the image
image = cv2.imread(image_path)
# Resize the image to a common size
image = cv2.resize(image, (image_width, image_height))
# Add the score to the image
image_with_score = cv2.putText(image, f"Score: {score}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0, 0), 2)
# Place the image in the canvas
canvas[row * image_height:(row + 1) * image_height, col * image_width:(col
+ 1) * image_width] = image_with_score


cv2.imshow("Similar Images", canvas)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:
Query image:

Similarities:

Conclusion:

In this experiment, we designed a Python application for image retrieval and search. Our approach involved utilizing the ORB (Oriented FAST and Rotated BRIEF) feature extraction technique and a straightforward score-based retrieval algorithm to identify and showcase similar images in a grid format. Through the standardization of image sizes and the addition of similarity scores, we generated a visual representation of the retrieved images. This program offers a user-friendly yet efficient method to search for and visualize images related to a query image within a provided dataset.