Name :- Pratham Ingawale

Roll no:- 25

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 and organizing images based on their visual content and semantic relationships, rather than relying solely on text-based queries. In this program, we implemented a simplified image retrieval system using Python and OpenCV. Here's the theoretical background:

Feature Extraction:

Feature extraction is a critical step in image retrieval. In our program, we used the Oriented FAST and Rotated BRIEF (ORB) algorithm to extract key points and descriptors from images.

ORB is a robust and efficient feature extraction algorithm that can describe local image characteristics.

Similarity Measurement:

To determine the similarity between images, we used the Hamming distance for comparing binary ORB descriptors. The Hamming distance counts the number of differing bits between two binary strings.

In a more advanced system, other similarity measures like Euclidean distance, cosine similarity, or deep neural network-based methods can be used.

Searching for Similar Images:

Given a query image, the program computes the ORB descriptors for the query image and compares them with the descriptors of images in the dataset.

The number of matches between the descriptors of the query image and each dataset image is used to score the similarity.

Images are then ranked by their scores, with the most similar images having the highest scores.

Displaying Images with Scores:

To visualize the retrieval results, we displayed the top-ranked images in a grid.

We ensured that all images in the grid had a consistent size by resizing them to a common dimension.

We added the retrieval scores to each image using the putText function from OpenCV.

User Interaction:

The program allows the user to interact with the results, and images are displayed together in a single window.

The user can use the left and right arrow keys to navigate through the grid of images and scores.

Customization:

The program is customizable. You can adjust the number of columns in the grid (grid_width) and the dimensions of the resized images (image_width and image_height) to suit your preferences and the dataset's characteristics.

This program provides a basic framework for image retrieval and searching, and more advanced systems can be built upon this foundation by incorporating deep learning-based feature extraction and more sophisticated retrieval algorithms

Code:

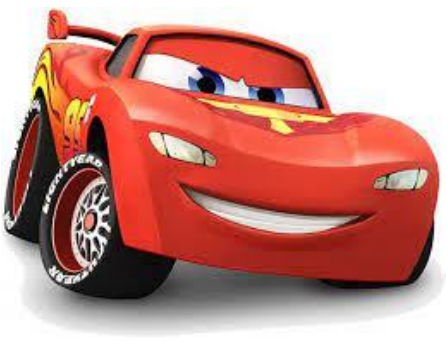
```
import cv2

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
```

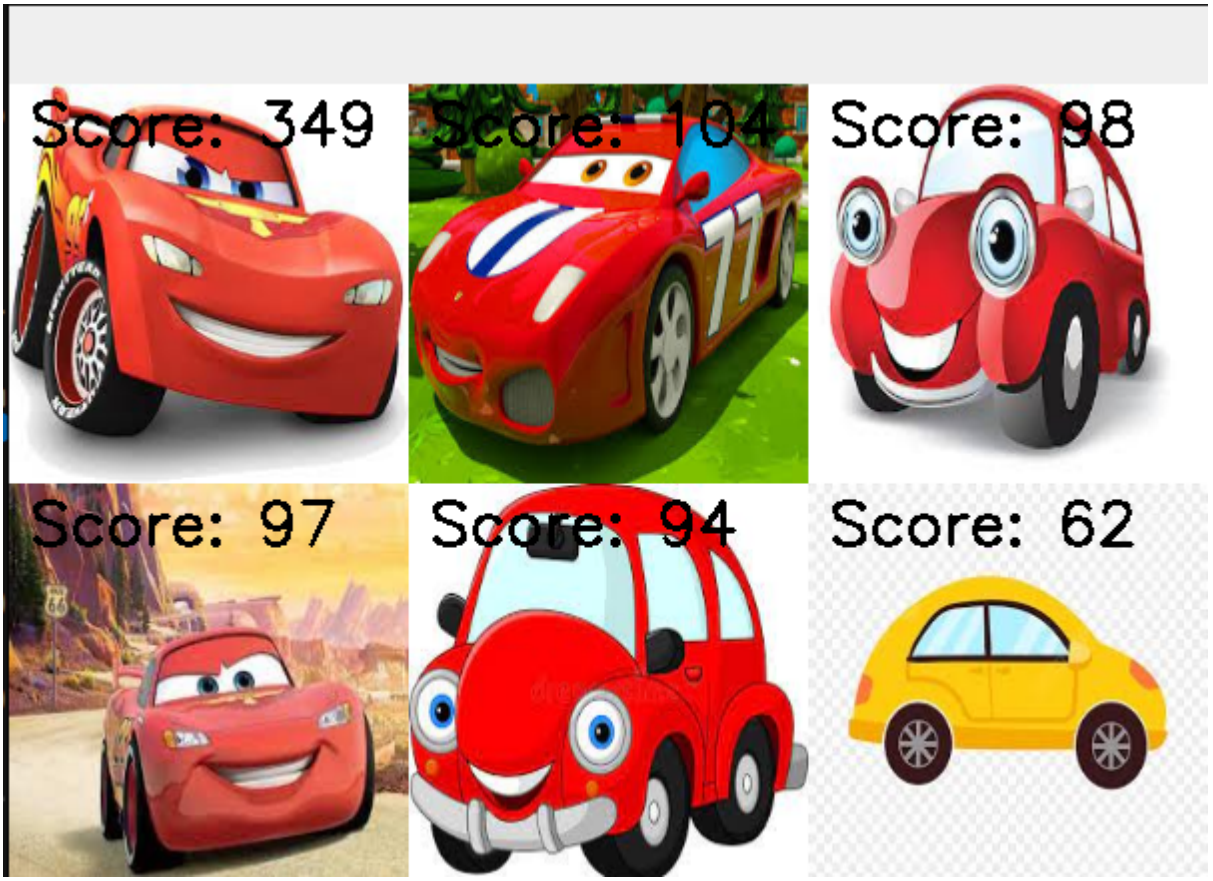
```
canvas = np.zeros((grid_height * image_height, grid_width *
image_width, 3), dtype="uint8")


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 EXP, we developed a Python program for image retrieval and searching. We used the ORB (Oriented FAST and Rotated BRIEF) feature extraction method and a basic score-based retrieval algorithm to find and display similar images in a grid. By resizing images to a common size and overlaying scores, we created a visual representation of the retrieved images with their respective similarity scores. This program provides a simple but effective way to search for and visualize images related to a query image from a given dataset.