Experiment: 1.4

**Aim:** Write a program to analyze the impact of refining feature detection for image segmentation by matching Feature Scores.

**Software Required:** Any Python IDE (e.g.: PyCharm, Jupyter Notebook, GoogleCollab)

**Technique used:**

**FLANN - Fast Library for Approximate Nearest Neighbors**

FLANN, which stands for "Fast Library for Approximate Nearest Neighbors," is a versatile and efficient library used extensively in computer vision, machine learning, and data mining applications.

**Nearest Neighbor Search**: FLANN specializes in solving the nearest neighbor search problem, which involves finding the data point in a dataset that is most similar to a given query point. This problem is fundamental in various fields, including recommendation systems, image retrieval, and clustering

**Approximate Nearest Neighbors**: FLANN excels in finding approximate nearest neighbors quickly, making it suitable for large datasets where an exhaustive search for the exact nearest neighbor may be computationally expensive.

**Algorithm Agnostic**: FLANN offers a variety of algorithms for nearest neighbor search, allowing users to choose the most suitable one based on their specific needs. Popular algorithms include KD-Tree, Hierarchical Clustering, and Locality-Sensitive Hashing (LSH).

**Efficient Indexing**: FLANN provides indexing structures that preprocess the dataset to speed up future nearest neighbor searches. These indexing structures significantly reduce query time complexity, making FLANN ideal for real-time applications.

**Parameter Tuning**: FLANN allows users to fine-tune parameters, such as the number of trees in a KD-Tree or the number of hash tables in LSH, to optimize search performance for specific datasets and applications.

**High-Dimensional Data**: FLANN is effective even in high-dimensional spaces, where traditional methods may struggle. It copes well with the "curse of dimensionality" by employing techniques like tree splitting and hashing.

**Pseudo Code**:

1. Import necessary libraries and modules, including OpenCV and os.
2. Load the target object image from the specified path and convert it to grayscale.
3. Initialize FLANN (Fast Library for Approximate Nearest Neighbors) parameters for feature matching. These parameters define the algorithm used for matching.
4. Initialize the FLANN-based matcher for feature matching.
5. Create empty lists to store matching scores and image file paths for the dataset images.
6. Loop through the images in the dataset directory:

a. For each image file, check if it has an image file extension (e.g., .jpg, .jpeg, .png).

b. Load the current dataset image, convert it to grayscale, and preprocess if necessary.

c. Extract SIFT (Scale-Invariant Feature Transform) features from both the target object and the current dataset image.

d. Match features between the target object and the dataset image using FLANN and apply Lowe's ratio test to filter good matches.

e. Calculate the matching score based on the number of good matches.

f. Store the matching score and the image path in respective lists.

1. Sort the dataset images based on their matching scores in descending order to identify the most similar images.
2. Display the top-ranked images and their matching scores. In this code, the top 5 images are displayed.

**Implementation**:

import os

import cv2

from google.colab.patches import cv2\_imshow

from IPython.display import Image

count=0

drivedir="/content/drive/MyDrive/Colab Notebooks/Images/"

curdir=drivedir+ "dogs/"

# Get the current working directory

print(curdir)

# Get a list of all the files in the current directory

files = os.listdir(curdir)

files = sorted(files)

for file in files:

    print(file)

# Get the name of the last file in the list

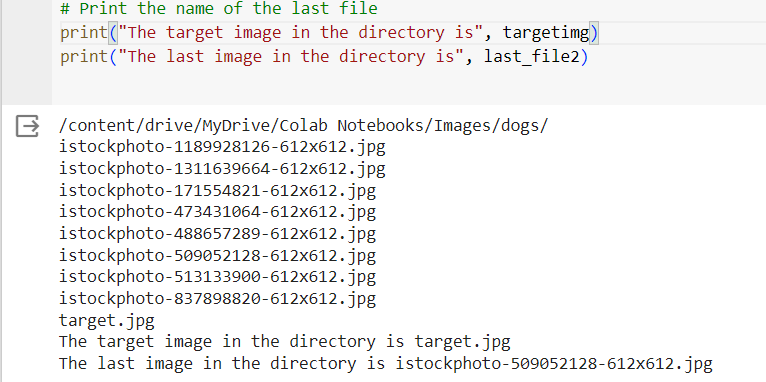
targetimg = files[-1]

last\_file2= files[-4]

# Print the name of the last file

print("The target image in the directory is", targetimg)

print("The last image in the directory is", last\_file2)



target\_image\_path = curdir+targetimg

target\_image = cv2.imread(target\_image\_path, cv2.IMREAD\_GRAYSCALE)  # Convert to grayscale

print(target\_image)

# Initialize FLANN parameters for feature matching

FLANN\_INDEX\_KDTREE = 1

index\_params = dict(algorithm=FLANN\_INDEX\_KDTREE, trees=5)

search\_params = dict(checks=50)

# Initialize the FLANN matcher

flann = cv2.FlannBasedMatcher(index\_params, search\_params)

# Initialize lists to store matching scores and image file paths

matching\_scores = []

matching\_images = []

# Loop through the dataset images

for file\_name in os.listdir(curdir):

    if file\_name.lower().endswith(('.jpg', '.jpeg', '.png')):

        image\_path = os.path.join(curdir, file\_name)

        # Skip the target image itself

        if image\_path == target\_image\_path:

            continue

        # Load and preprocess the current dataset image

        dataset\_image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  # Convert to grayscale

        # Extract features from both the target object and dataset image

        sift = cv2.SIFT\_create()

        print(target\_image)

        keypoints\_target, descriptors\_target = sift.detectAndCompute(target\_image, None)

        keypoints\_dataset, descriptors\_dataset = sift.detectAndCompute(dataset\_image, None)

        # Match features between the target object and dataset image

        matches = flann.knnMatch(descriptors\_target, descriptors\_dataset, k=2)

        # Apply Lowe's ratio test to filter good matches

        good\_matches = []

        for m, n in matches:

            if m.distance < 0.7 \* n.distance:

                good\_matches.append(m)

        # Calculate the matching score based on the number of good matches

        matching\_score = len(good\_matches)

        # Store the matching score and image path

        matching\_scores.append(matching\_score)

        matching\_images.append(image\_path)

# Sort images based on matching scores in descending order

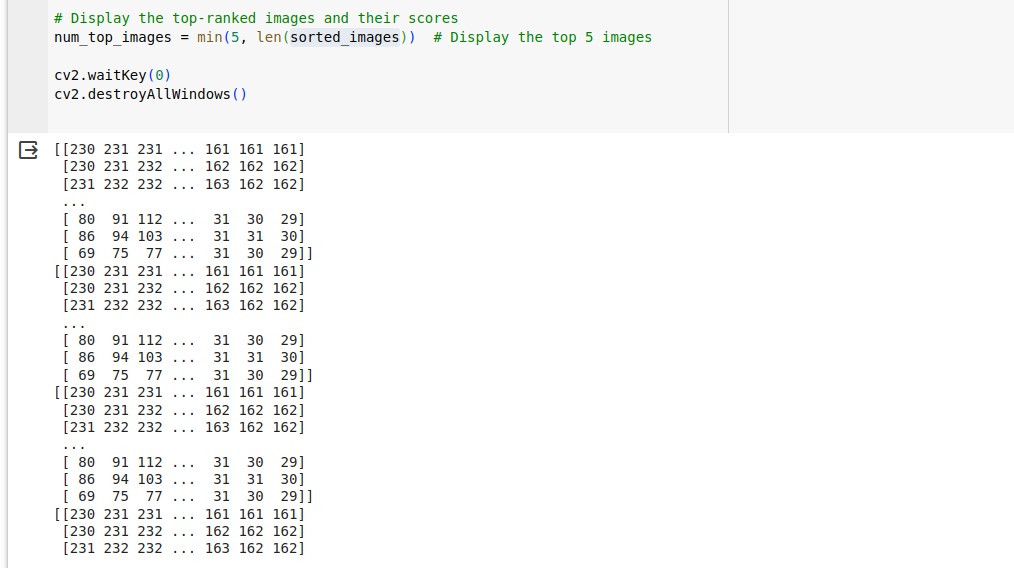
sorted\_images = [img for \_, img in sorted(zip(matching\_scores, matching\_images), reverse=True)]

# Display the top-ranked images and their scores

num\_top\_images = min(5, len(sorted\_images))  # Display the top 5 images

cv2.waitKey(0)

cv2.destroyAllWindows()



for i in range(num\_top\_images):

    print(f"Image: {sorted\_images[i]}, Matching Score: {matching\_scores[i]}")

    cv2\_imshow(cv2.imread(sorted\_images[i]))

**Output screenshot**:

