CourseName: Computer Vision Lab **Course Code:** CSP-422

Experiment: 1.3

Aim: Write a program to assess various feature matching algorithms for object recognition.

Software Required: Any Python IDE (e.g.: PyCharm, Jupyter Notebook, Google Colab)

Technique used: Feature matching is a fundamental task in computer vision where you identify and match key features or points in two or more images to establish correspondences, which can be used for various tasks like image stitching, object recognition, and tracking. Feature matching algorithms are crucial for object recognition tasks as they establish correspondences between key-points or descriptors extracted from images. Some commonly used feature matching algorithms for object recognition are:

Brute-Force Matching: Brute force matching is a straightforward and exhaustive method for finding matches between elements or patterns in a dataset. It involves comparing each element in one dataset with every element in another dataset to identify similarities or matches. While conceptually simple, brute force matching can be computationally intensive and may not be efficient for large datasets, making it less practical for certain applications.

Lowe's Ratio Test: In Lowe's Ratio Test, each keypoint in the first image is paired with multiple keypoints from the second image. We select the two best matches for each keypoint based on their distance measurements. To ensure the reliability of these matches, Lowe's test checks that the distances between these two best matches are significantly different. If the distances are not distinct enough, the keypoint is discarded and excluded from further processing to improve the accuracy of feature matching.

Fast Library for Approximate Nearest Neighbors (FLANN): FLANN, or the Fast Library for Approximate Nearest Neighbors, is an open-source software library designed for efficient and approximate nearest neighbor search in high-dimensional spaces, commonly used in computer vision and machine learning applications.

Nearest Neighbor with Ratio Test: It is a technique used in object feature matching. For each feature point in an image, it identifies the closest match in a second image. However, it doesn't stop there; it also looks at the second-closest match. If the ratio of the distance to the closest match compared to the distance to the second-closest match is below a certain threshold, the match is considered reliable and retained. Otherwise, it's discarded. This method helps filter out ambiguous matches, improving the accuracy of object recognition and tracking.

Name: Abhinav Verma **UID:** 20BCS9258

Steps:

1. Import Necessary libraries.
2. Load the reference image and the query image.
3. Convert the both images to grayscale.
4. Extract features from both images using a chosen feature extraction technique (e.g., SIFT - Create SIFT object and detect the keypoints using SIFT Detector, compute the descriptors for both images).
5. Apply feature matching algorithms (e.g., Brute-Force Matcher, Lowe's Ratio Test, FLANN) to match the extracted features.
   - 5.1. **Brute-Force Matcher**:
     a. Create feature matcher for Brute-Force Matcher.
     b. Match descriptors of both images.
     c. Sort matches by distance and draw the first 50 good matches and show the images.
   - 5.2. **Lowe's Ratio Test**:
     a. Create a FLANN based matcher.
     b. Matching descriptor vectors with a FLANN based matcher.
     c. Filter out NoneType matches (no matches) and extract the distances.
     d. Sort the matches based on their distances and define a ratio threshold = 0.8.
     e. Filter good matches based on the ratio test and draw the first 50 good matches and show the images.
   - 5.3. **Nearest Neighbor with Ratio Test**:
     a. Create a Nearest Neighbors model with k=1, using L2 distance metric.
     b. Fit the model to the descriptors of the second image.
     c. Create an empty list to store nearest neighbor distances.
     d. Loop through each descriptor in descriptors_1. Use the Nearest Neighbors model to find the nearest neighbor of the current descriptor. Extract the index of the nearest neighbor and append it to the list.
     e. Draw the first 50 matches and display the matched image.

CourseName: Computer Vision Lab **Course Code:** CSP-422

Implementation:

```
import cv2
from google.colab.patches import cv2_imshow
 # reading the images
img1 = cv2.imread('/content/drive/MyDrive/Colab Notebooks/Images/Shah Rukh Khan.webp')
img2 = cv2.imread('/content/drive/MyDrive/Colab Notebooks/Images/SRK.png')

# converting image to grayscale
img1=cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2=cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# creating sift object
sift=cv2.xfeatures2d.SIFT_create()
# Detect the keypoints using SIFT Detector, compute the descriptors
# detect sift feature in both images
keypoints_1,descriptors_1=sift.detectAndCompute(img1, None)
keypoints_2,descriptors_2=sift.detectAndCompute(img2, None)
# drawing the keypoints on the image to visualize them
img1_with_keypoints = cv2.drawKeypoints(img1, keypoints, img1)

# Display the image with keypoints
cv2_imshow(img1_with_keypoints)
```

Output (Image with Keypoints):

Fig.1: Image with Keypoints

Method 1: Brute Force Method

```
# create feature matcher
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck= True)

# match descriptors of both images
matches = bf.match(descriptors_1,descriptors_2)

# sort matches by distance
matches=sorted(matches,key=lambda x:x.distance)

# draw first 50 matches
matched_image = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2,
matches[:50], img2, flags=2)

# show the image
cv2_imshow(matched_image)
```

Output (Feature Matching with Brute Force Method):

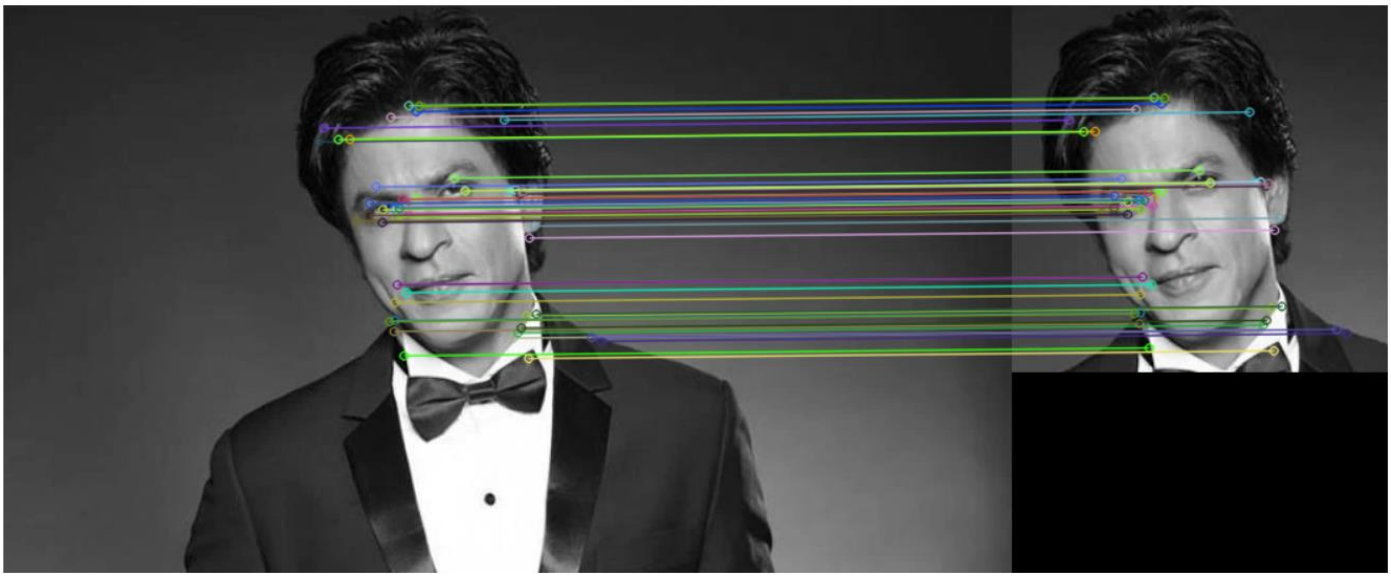

Fig.2: Feature Matching with Brute Force Method

Method 2: Lowe's Ratio Method

```
# Importing Libraries
import cv2

# Matching descriptor vectors with a FLANN based matcher

# Create a FLANN based matcher
matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)

# Compute K-nearest neighbor matches
knn_matches = matcher.knnMatch(descriptors_1, descriptors_2, 2)


# Filter out NoneType matches (no matches) and extract the distances
knn_matches = [match for match in knn_matches if match[0] is not None and
match[1] is not None]

# Sort the matches based on their distances
knn_matches = sorted(knn_matches, key=lambda x: x[0].distance)

# Define a ratio threshold
ratio_thresh = 0.8

# Filter good matches based on the ratio test
good_matches = []
for m, n in knn_matches:
    if m.distance < ratio_thresh * n.distance:
        good_matches.append(m)

# Draw the first 50 good matches
matched_image = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2,
good_matches[:50], img2, flags=2)

# Show the image
cv2_imshow(matched_image)
```

Output (Feature Matching with Lowe's Ratio Method):

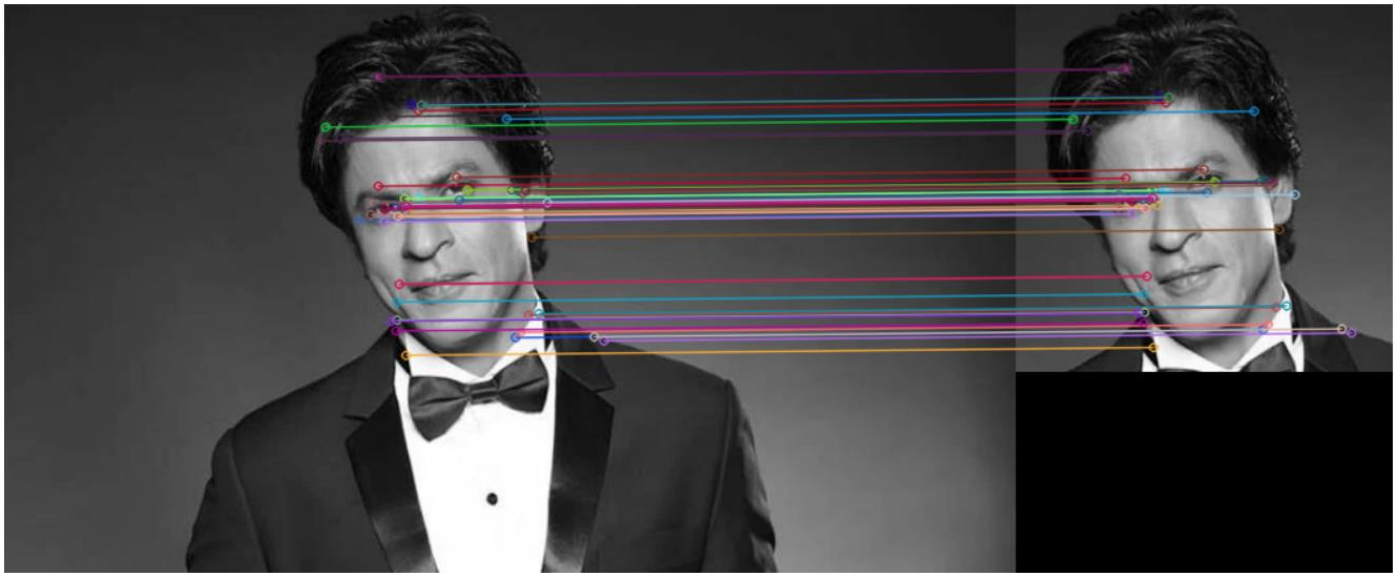

Fig.3: Feature Matching with Lowe's Ratio Method

Method 3: Nearest Neighbor with Ratio Test

```
# Import necessary libraries
from sklearn.neighbors import NearestNeighbors

import numpy as np

# Create a Nearest Neighbors model with k=1 (find the nearest neighbor),
using L2 distance metric
nn_model = NearestNeighbors(n_neighbors=1, algorithm='auto', metric='l2')

# Fit the model to the descriptors of the second image (descriptors_2)
nn_model.fit(descriptors_2)

# Create an empty list to store nearest neighbor distances
nearest_distances = []

# Loop through each descriptor in descriptors_1
for descriptor in descriptors_1:
```

```
    # Use the Nearest Neighbors model to find the nearest neighbor of the
current descriptor
    _, nearest_indices = nn_model.kneighbors(descriptor.reshape(1, -1),
n_neighbors=1)

    # Extract the index of the nearest neighbor and append it to the list
    nearest_distance = nearest_indices[0][0]
    nearest_distances.append(nearest_distance)

# Draw the first 50 matches
matched_image = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2,
matches[:50], img2, flags=2)

# Display the matched image
cv2_imshow(matched_image)
```

Output (Feature Matching with Nearest Neighbor with Ratio Test):

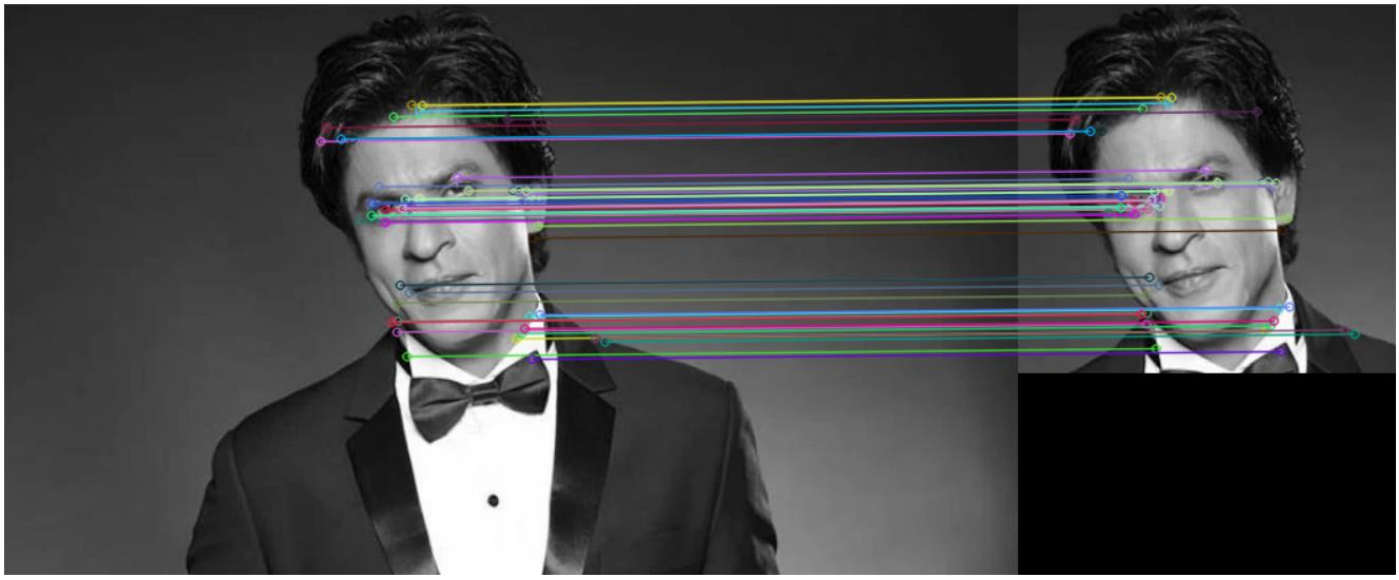

Fig.4: Feature Matching with Nearest Neighbor with Ratio Test