**Experiment 1.2**

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

**Software Required:** Matlab, Google Colab

**Objective:** The primary objective of this project is to create a program that successfully implements feature matching techniques for image classification. This program will systematically compare and match distinctive features extracted from images, enabling accurate and reliable image classification. By utilizing advanced feature matching algorithms, the project aims to enhance the precision and robustness of image classification processes, thereby contributing to the advancement of computer vision applications. The program's objective is to empower researchers, developers, and practitioners with a powerful tool for improving the efficiency and effectiveness of image classification tasks through the utilization of feature matching techniques.

**Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

query\_image = cv2.imread('/content/WhatsApp Image 2023-08-29 at 12.30.23.jpeg', cv2.IMREAD\_GRAYSCALE)

target\_image = cv2.imread('/content/ytyg.jpeg', cv2.IMREAD\_GRAYSCALE)

feature\_extractors = {

'SIFT': cv2.SIFT\_create(),

'ORB': cv2.ORB\_create(),

'AKAZE': cv2.AKAZE\_create(),

'BRISK': cv2.BRISK\_create()

}

matchers = {

'BFMatcher': cv2.BFMatcher(),

'FlannBasedMatcher': cv2.FlannBasedMatcher()

}

query\_kp, query\_des = feature\_extractors['SIFT'].detectAndCompute(query\_image, None)

target\_kp, target\_des = feature\_extractors['SIFT'].detectAndCompute(target\_image, None)

results = {}

# Loop through feature extractors and matchers

for extractor\_name, extractor in feature\_extractors.items():

for matcher\_name, matcher in matchers.items():

# Skip AKAZE with FlannBasedMatcher due to compatibility issues

if extractor\_name == 'AKAZE' and matcher\_name == 'FlannBasedMatcher':

continue

# Compute matches

if matcher\_name == 'BFMatcher':

matches = matcher.knnMatch(query\_des, target\_des, k=2)

else:

matches = matcher.knnMatch(np.float32(query\_des), np.float32(target\_des), k=2)

# Apply ratio test

good\_matches = []

for m, n in matches:

if m.distance < 0.75 \* n.distance:

good\_matches.append(m)

# Store match count in results

match\_count = len(good\_matches)

results[(extractor\_name, matcher\_name)] = match\_count

# Print the results

for (extractor\_name, matcher\_name), match\_count in results.items():

print(f"{extractor\_name} + {matcher\_name}: {match\_count} matches")

# Print the results

for (extractor\_name, matcher\_name), match\_count in results.items():

print(f"{extractor\_name} + {matcher\_name}: {match\_count} matches")

# Plot the matches for the best combination

best\_combination = max(results, key=results.get)

best\_extractor = feature\_extractors[best\_combination[0]]

best\_matcher = matchers[best\_combination[1]]

if best\_combination[1] == 'BFMatcher':

matches = best\_matcher.knnMatch(query\_des, target\_des, k=2)

else:

matches = best\_matcher.knnMatch(np.float32(query\_des), np.float32(target\_des), k=2)

good\_matches = []

for m, n in matches:

if m.distance < 0.75 \* n.distance:

good\_matches.append(m)

result\_image = cv2.drawMatches(query\_image, query\_kp, target\_image, target\_kp, good\_matches, None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

plt.figure(figsize=(10, 8))

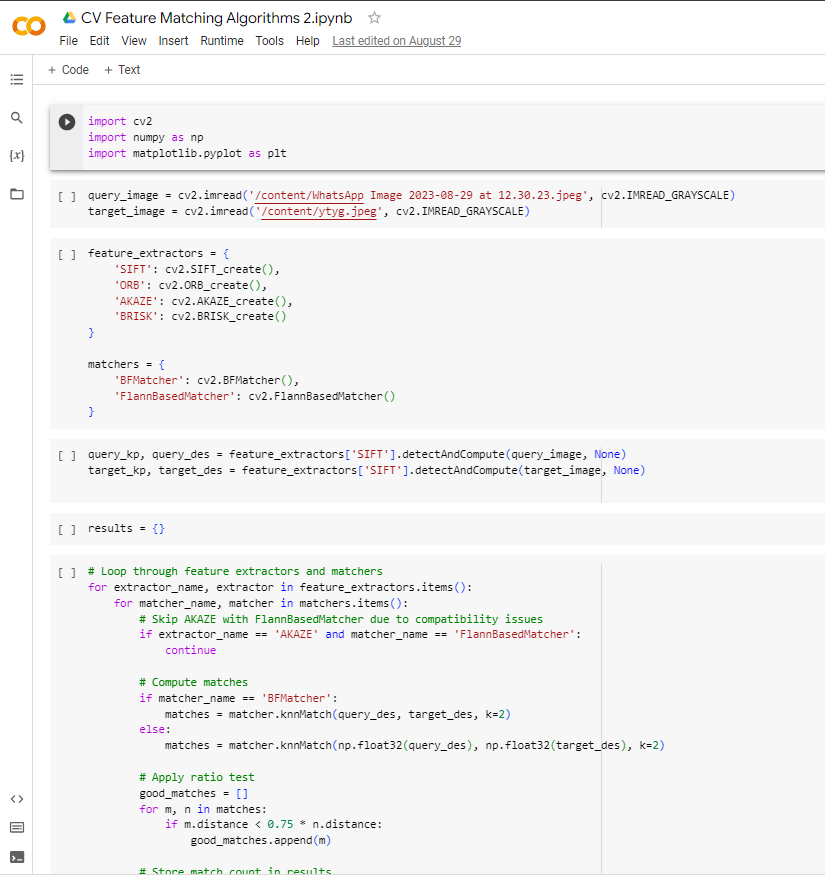
plt.imshow(result\_image)

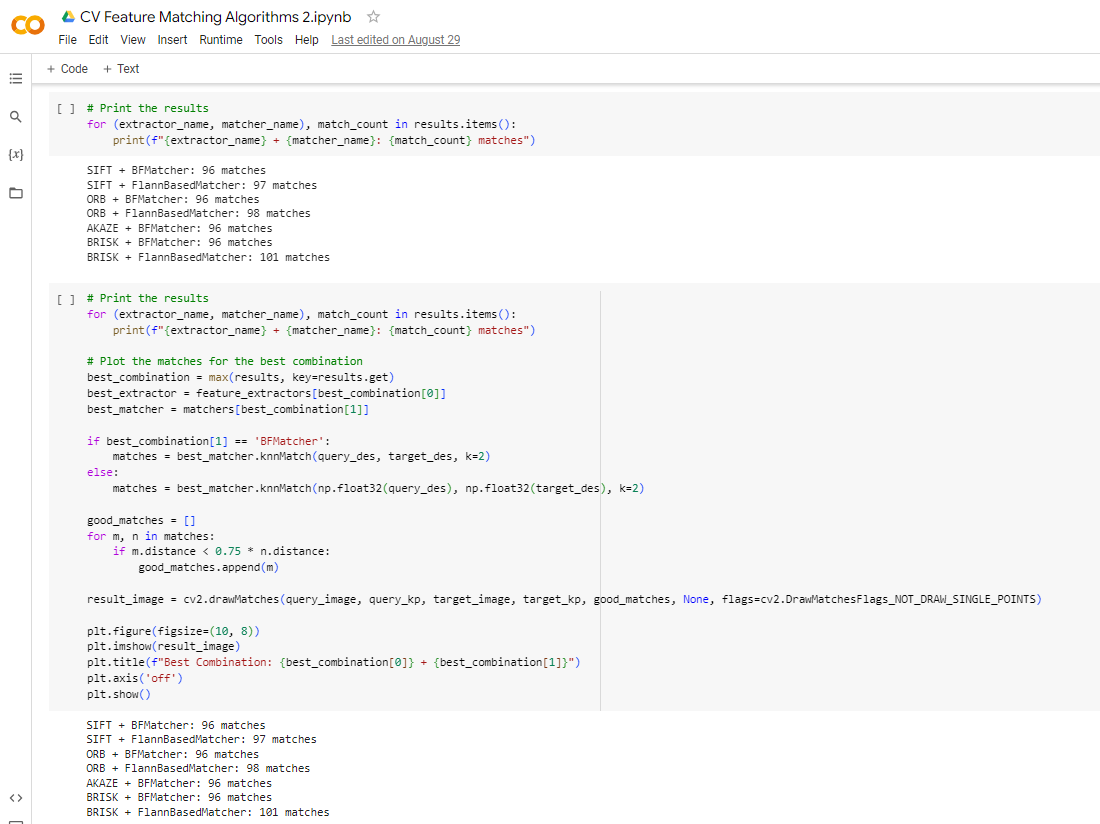
plt.title(f"Best Combination: {best\_combination[0]} + {best\_combination[1]}")

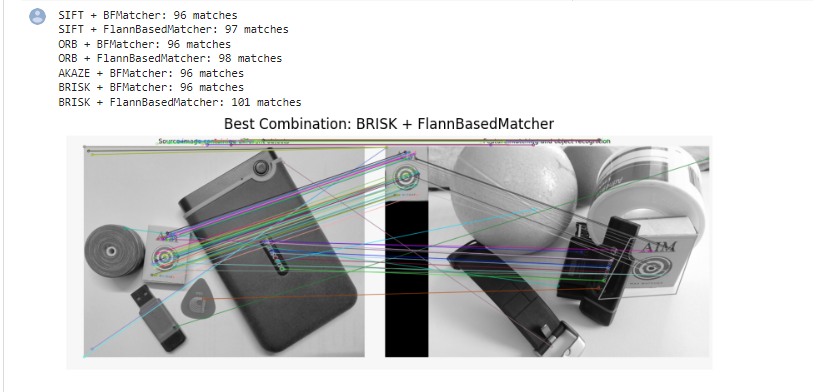
plt.axis('off')

plt.show()

**Implementation:**

****

****

****