Experiment 1.2

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

Software Required: Matlab, Google Colab

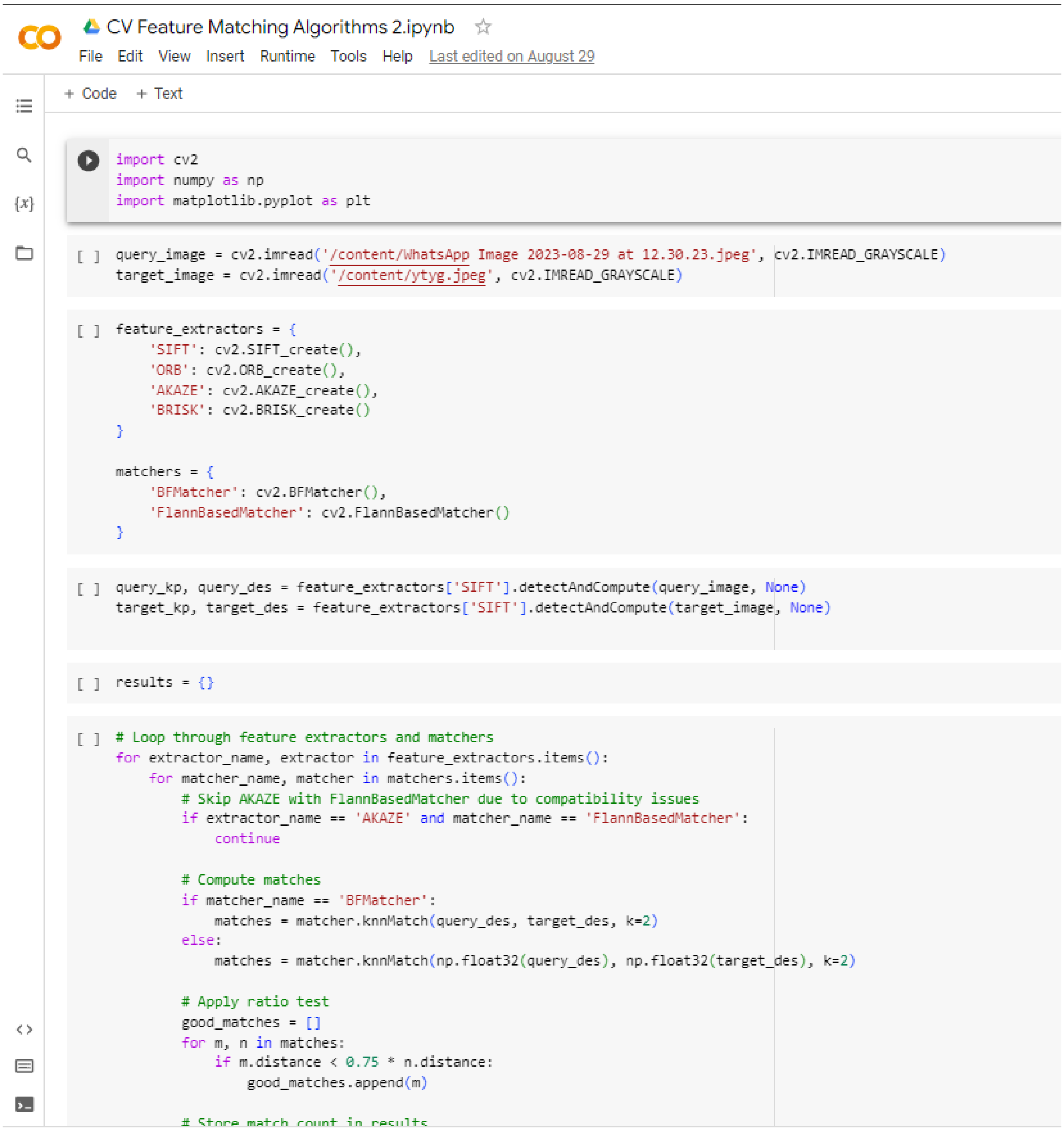
Objective: The primary objective of this project is to create a program that successfully implements feature matching techniques for image classification. This program will systematically compare and match distinctive features extracted from images, enabling accurate and reliable image classification. 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:


CV Feature Matching Algorithms 2.ipynb
File Edit View Insert Runtime Tools Help Last edited on August 29
+ Code + Text

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

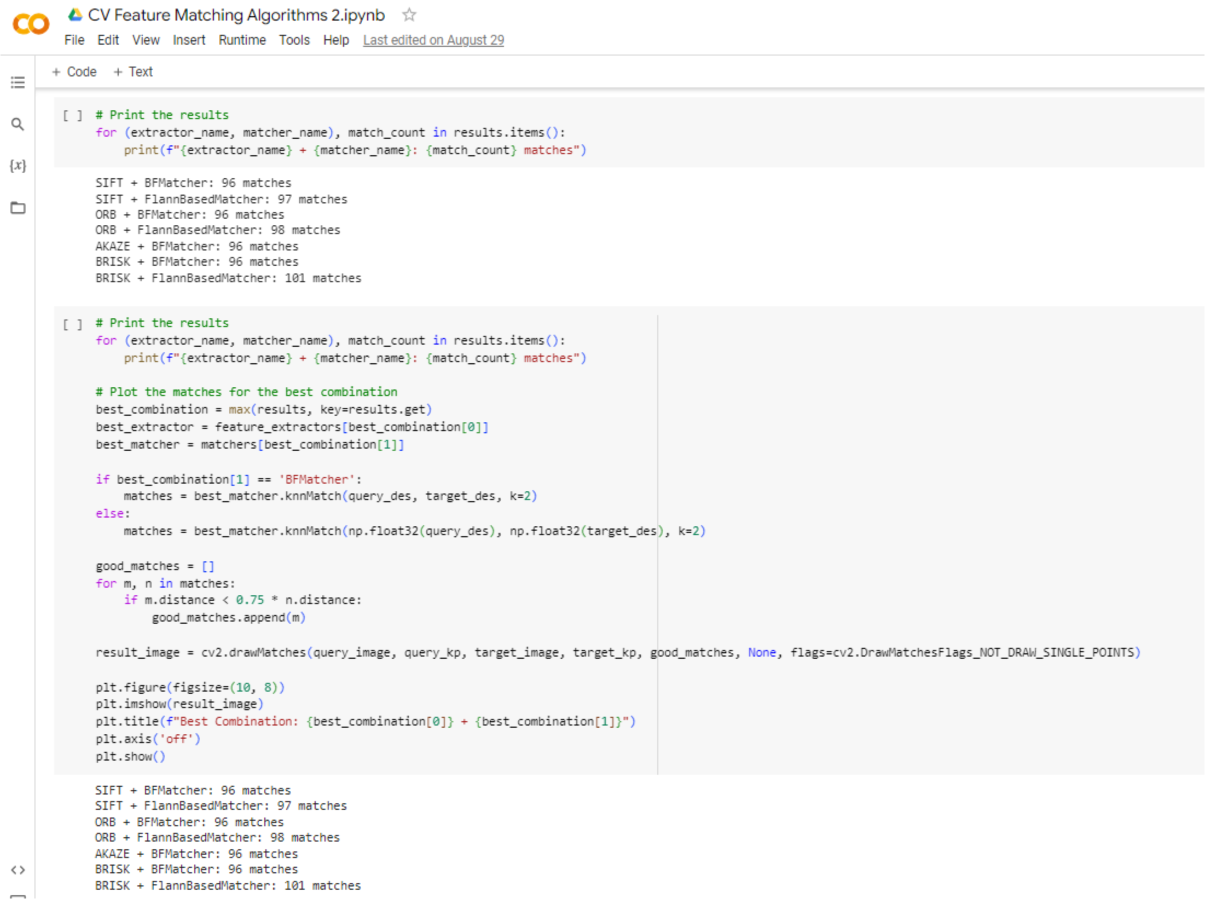
```
query_image = cv2.imread('/content/WhatsApp Image 2023-08-29 at 12.30.23.jpeg', cv2.IMREAD_GRAYSCALE)
target_image = cv2.imread('/content/ytyg.jpeg', cv2.IMREAD_GRAYSCALE)
```

```
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
```



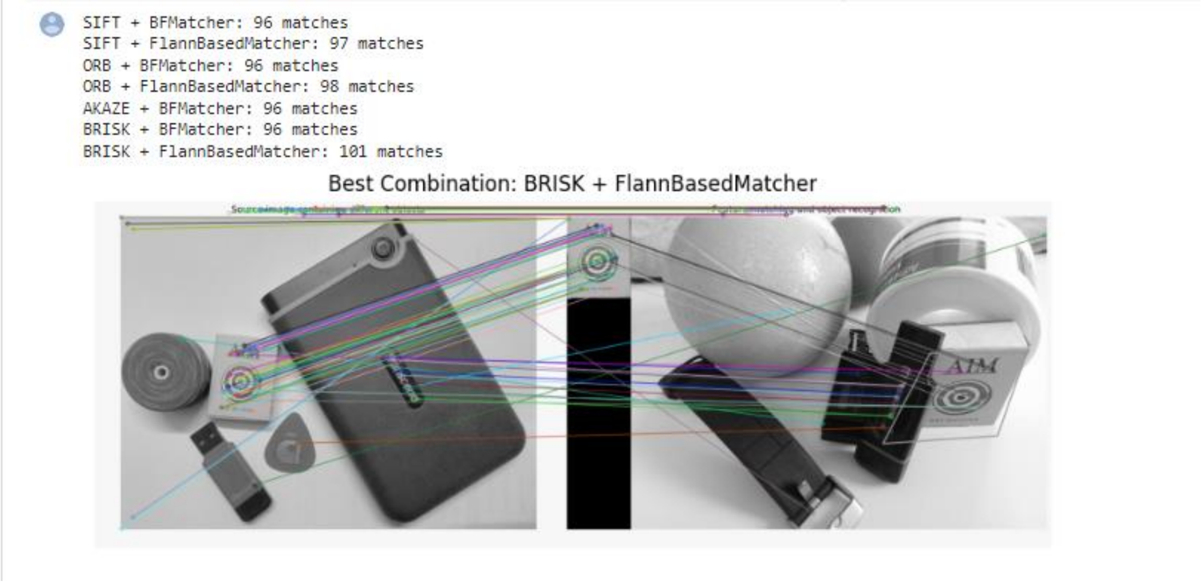
CO
CV Feature Matching Algorithms 2.ipynb
File Edit View Insert Runtime Tools Help Last edited on August 29
+ Code + Text

```
[ ] # Print the results
    for (extractor_name, matcher_name), match_count in results.items():
        print(f"{extractor_name} + {matcher_name}: {match_count} matches")
```

```
SIFT + BFMatcher: 96 matches
SIFT + FlannBasedMatcher: 97 matches
ORB + BFMatcher: 96 matches
ORB + FlannBasedMatcher: 98 matches
AKAZE + BFMatcher: 96 matches
BRISK + BFMatcher: 96 matches
BRISK + FlannBasedMatcher: 101 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()
```

```
SIFT + BFMatcher: 96 matches
SIFT + FlannBasedMatcher: 97 matches
ORB + BFMatcher: 96 matches
ORB + FlannBasedMatcher: 98 matches
AKAZE + BFMatcher: 96 matches
BRISK + BFMatcher: 96 matches
BRISK + FlannBasedMatcher: 101 matches
```



```
SIFT + BFMatcher: 96 matches
SIFT + FlannBasedMatcher: 97 matches
ORB + BFMatcher: 96 matches
ORB + FlannBasedMatcher: 98 matches
AKAZE + BFMatcher: 96 matches
BRISK + BFMatcher: 96 matches
BRISK + FlannBasedMatcher: 101 matches
```

Best Combination: BRISK + FlannBasedMatcher