CSE 463
Homework 2 Report

Sena Özbelen
1901042601
4 June 2024

The Selected Objects

- calculator : 0
- camera : 1
- cell phone : 2
- cereal box : 3
- dry battery : 4
- flashlight : 5
- garlic : 6
- hand towel : 7
- instant noodles : 8
- marker : 9

They have a mapping as shown above to compute the confusion matrix.

The Selected Detectors

- **SIFT (Scale-Invariant Feature Transform):** SIFT meticulously analyzes the image at different scales, searching for stable keypoints and their orientations.
- **KAZE (Keys and Affine-Invariant Regions):** It prioritizes speed while maintaining good accuracy for scale and rotation.
- **BRISK (Binary Robust Invariant Scalable Keypoints):** BRISK is all about speed and efficiency. It uses a simpler approach to find keypoints and creates compact descriptions using binary codes. The binary descriptors might not capture as much detail as the others.

The Details of the Code

- Step 1 : Load data

It retrieves the image paths from the dataset directory. It puts all the image paths in a list.

- Step 2 : Split the dataset

It splits the dataset as training and test by using train_test_split function from sklearn.

- Step 3 : Create detectors and the BFMatcher

- Step 4 : Define the main functions for the object detection

There are two main functions to run this object detection algorithm.

```python
def detect_and_compute(images, detector):
    keypoints = []
    descriptors = []
    count = 0
    for image in images:
        img = cv2.imread(image, cv2.IMREAD_GRAYSCALE)
        kp, des = detector.detectAndCompute(img, None)
        keypoints.append(kp)
        if des is not None:
            descriptors.append(des)
        else:
            descriptors.append(np.empty((0, 64)))  # Append empty array if descriptor is None
            count += 1

    return keypoints, descriptors, count
```

detect_and_compute : It gets each image and feeds the detector's detectAndCompute function to get the key points and descriptors. This is used for all 3 detectors. For some images, there might not be found any key points. For these ones, empty descriptions are added not to break the order of the images. Counter info is used to show the number of None descriptors.

```python
def match_and_predict(test_descriptors, train_descriptors, matcher):
    predictions = []
    for des_test in test_descriptors:
        best_matches = []
        for des_train in train_descriptors:

            if des_test.size == 0 or des_train.size == 0:  # Skip if descriptors are empty
                best_matches.append(0)
                continue

            matches = matcher.knnMatch(des_test, des_train, k=2) # Use matcher

            # Apply ratio test
            good_matches = []
            for m_n in matches:
                if len(m_n) == 2:  # Ensure that there are two matches
                    m, n = m_n
                    if m.distance < 0.75 * n.distance:
                        good_matches.append(m)

            best_matches.append(len(good_matches)) # Add the number of good matches

        best_match_index = np.argmax(best_matches) # Get the index of the best match
        predictions.append(best_match_index)
    return predictions
```

match_and_predict: This iterates over each test image in order to predict its class by using descriptors. It uses BFMatcher and for each training image, it gets the matches between the test image and the training image and then applies ratio test to get the good matches. I used the one available in **openCV's page**. Then, for each training image, it adds the total number of good matches to the best matches list. Once it iterates over each training image, it gets the index of the best match which has the most good matches. This is added to the list of predictions. This is done for each test image.

- Step 5 : Get the keypoints and descriptions and check the number of None descriptors

```
print(len(train_images))
print(len(test_images))
```

✓ 0.0s

```
5966
663
```

```
print("Number of None occurrences in descriptors_sift_train:", count1)
print("Number of None occurrences in descriptors_sift_test:", count2)
print("Number of None occurrences in descriptors_kaze_train:", count3)
print("Number of None occurrences in descriptors_kaze_test:", count4)
print("Number of None occurrences in descriptors_brisk_train:", count5)
print("Number of None occurrences in descriptors_brisk_test:", count6)
```

✓ 0.0s

```
Number of None occurrences in descriptors_sift_train: 0
Number of None occurrences in descriptors_sift_test: 0
Number of None occurrences in descriptors_kaze_train: 10
Number of None occurrences in descriptors_kaze_test: 0
Number of None occurrences in descriptors_brisk_train: 1478
Number of None occurrences in descriptors_brisk_test: 159
```

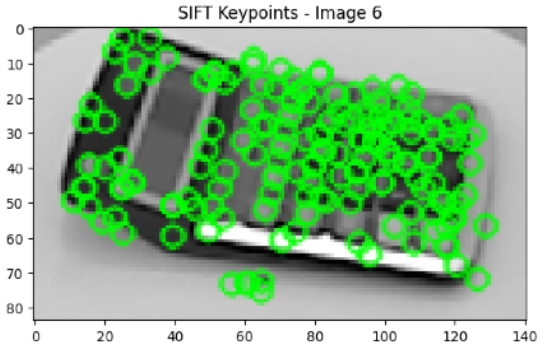
It uses detect_and_compute and then it shows the number of None descriptors. BRISK has a lot of None descriptors so this will affect the accuracy of object detection algorithm.

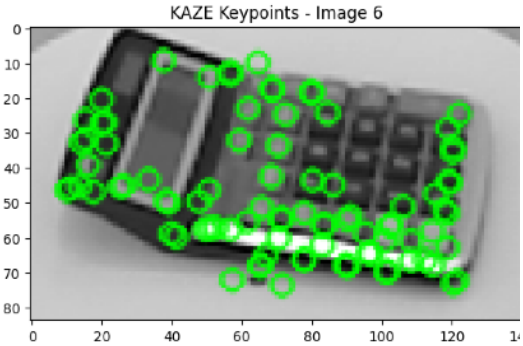
- Step 6 : Extract the labels of the images and map them into numerical labels

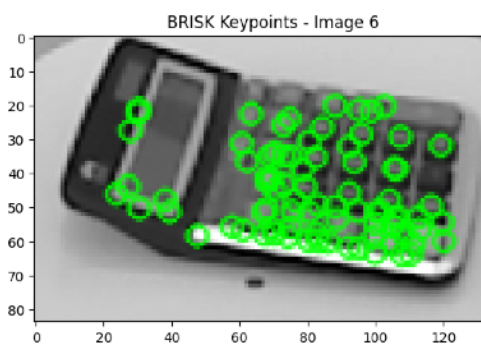
Numerical conversion is necessary for confusion matrix computation. Therefore, I used the mapping that I already mentioned in the selected objects part.

- Step 7 : Get a sample including each class and visualize the key points

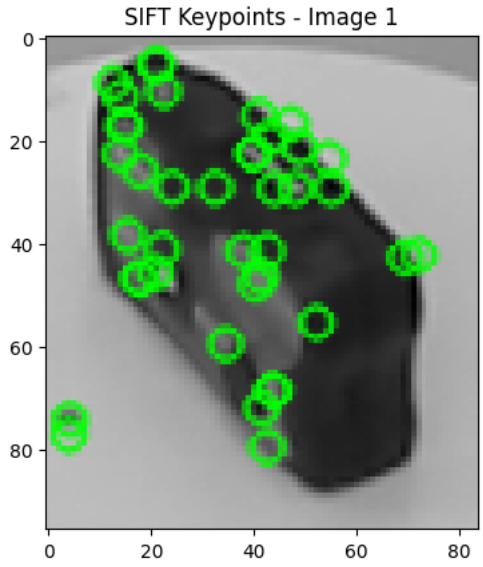
For calculator, SIFT has the most key points. KAZE couldn't find some details such as some keys. BRISK couldn't find that many points for the screen part.

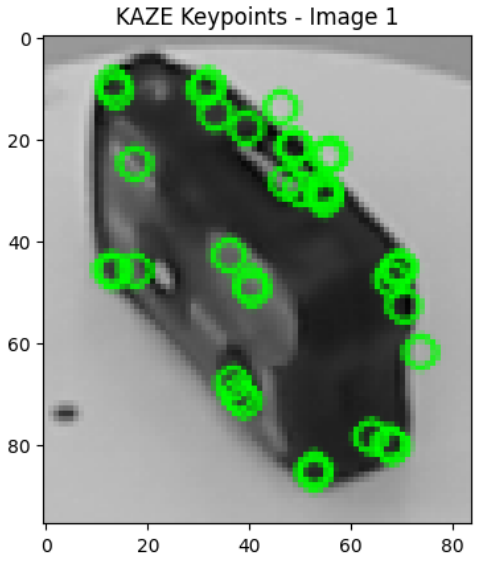


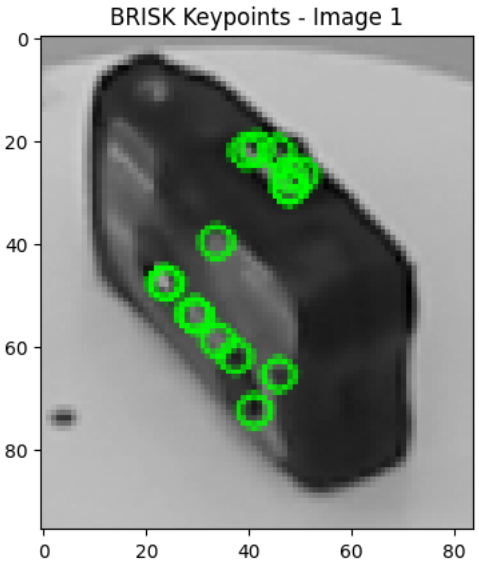




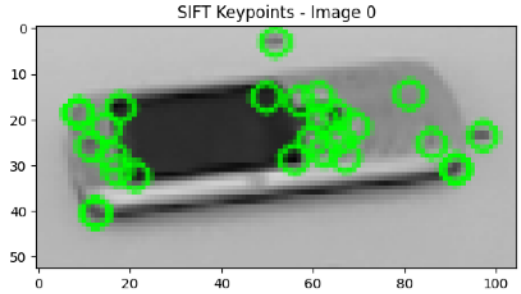
For camera, SIFT and KAZE have similar points. BRISK couldn't find that many points. It focuses on the upper small part and some small parts in the front only.

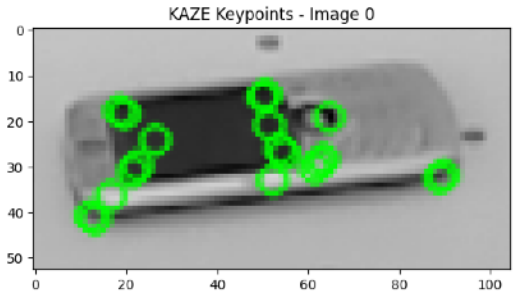


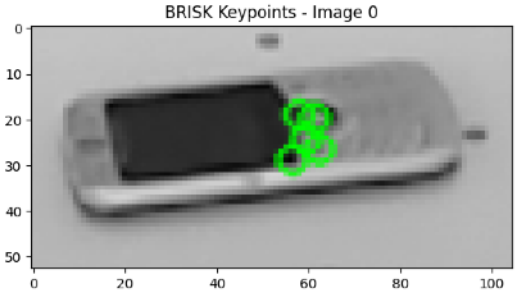




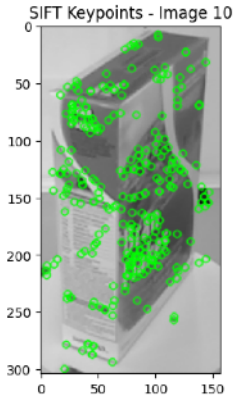
For cellphone, SIFT and KAZE focus on the main button and the upper part of the screen. BRISK only focuses on the main button.

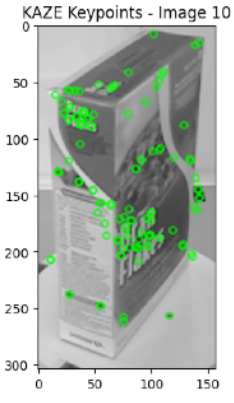


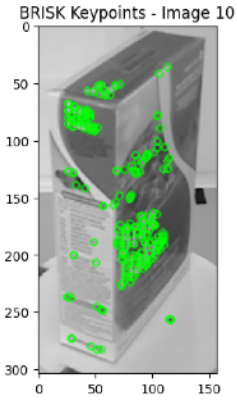




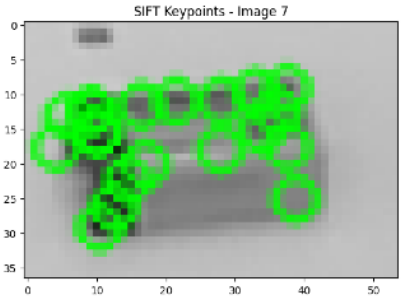
For cereal box, SIFT finds many points from the box and from the background as well. KAZE and BRISK focus on the specific symbols and big text.

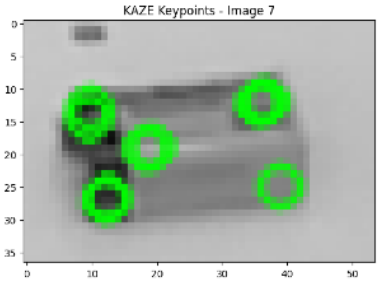


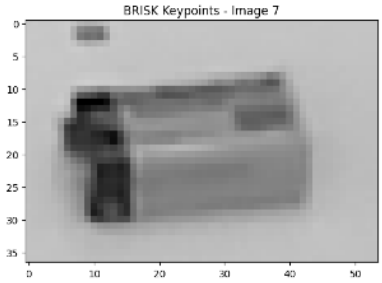




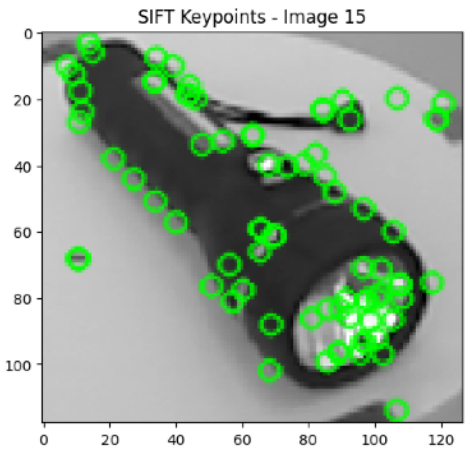
For dry battery, SIFT again finds many points on the battery. KAZE only finds a few specific and distinct points. BRISK fails for dry battery.

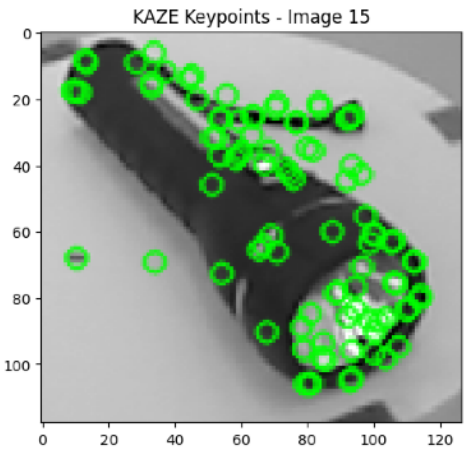


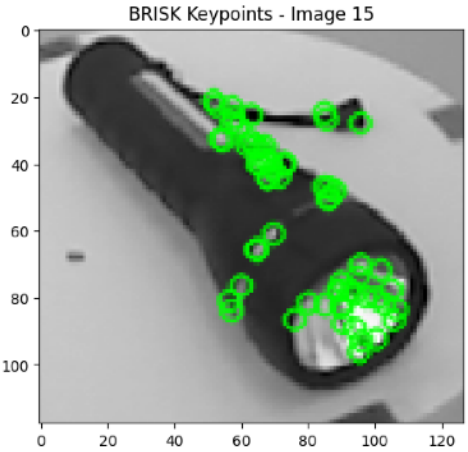




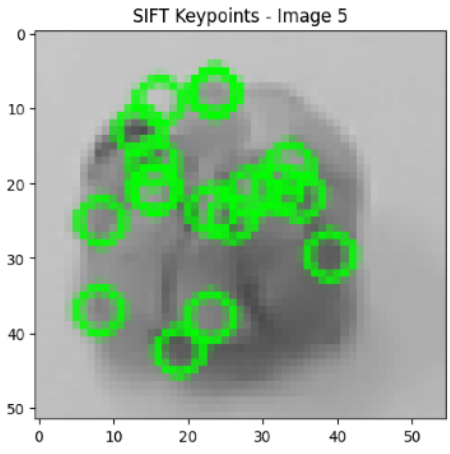
For flashlight, SIFT and KAZE find most edges. KAZE also finds the string very well. BRISK only finds the button part and the bulb part.

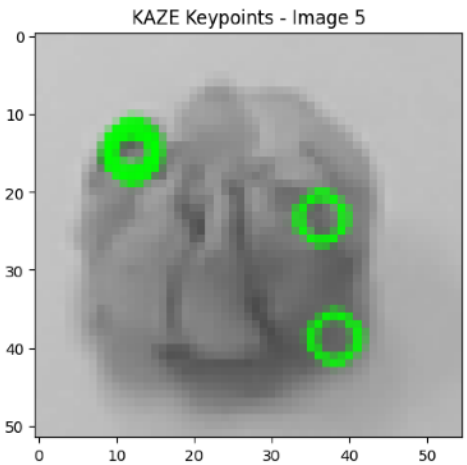


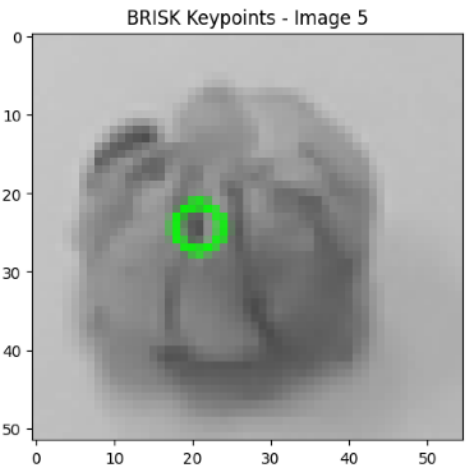




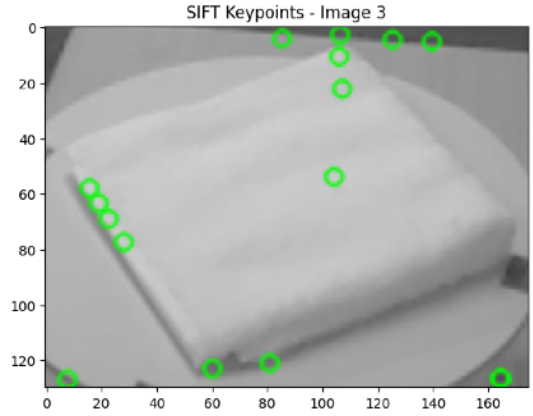
For garlic, SIFT finds a lot of points from the upper part of the garlic and some edges. KAZE and BRISK only find a few edge-like points.

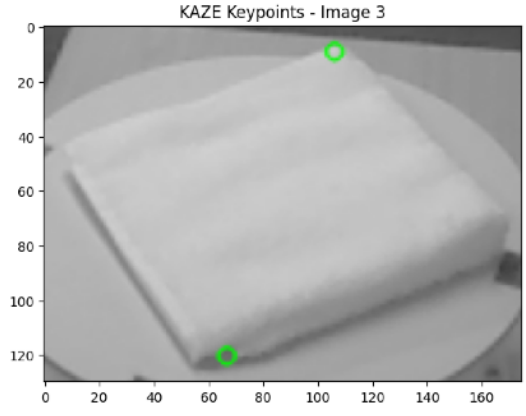


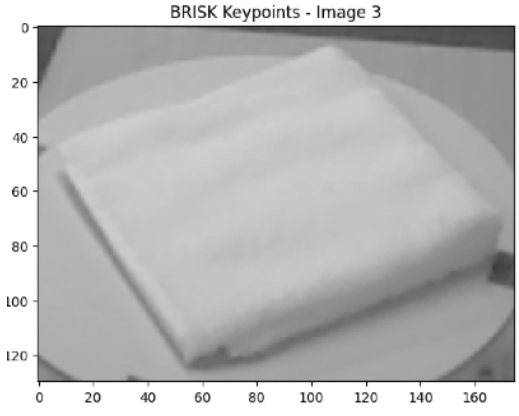




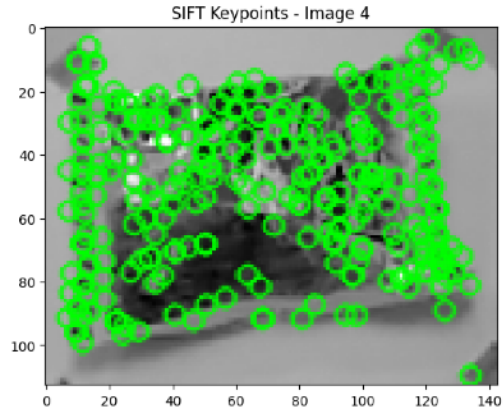
For hand towel, SIFT only finds some edges of the towel and some points in the background. KAZE finds two corner of the towel and BRISK fails.

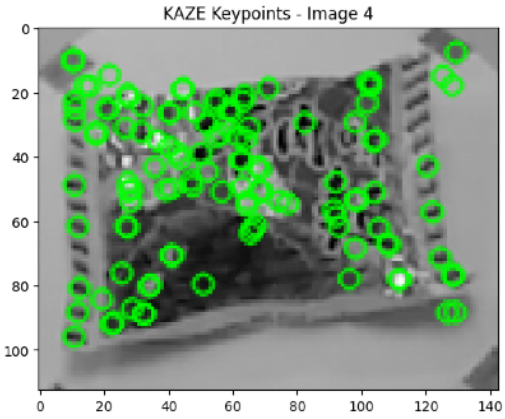


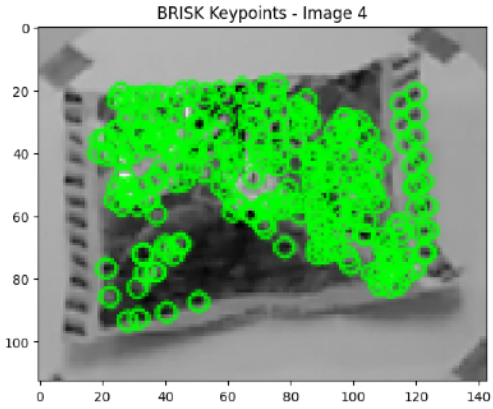




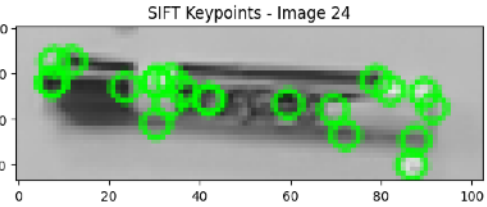
For instant noodles, SIFT again finds a lot of point from the whole surface of the noodle package. KAZE and BRISK find the text on the package and the some short lines located at the side edges.

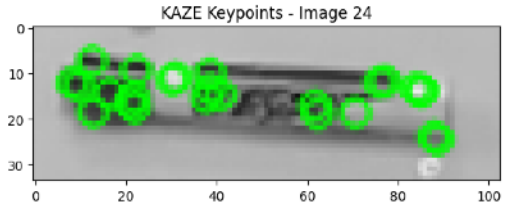


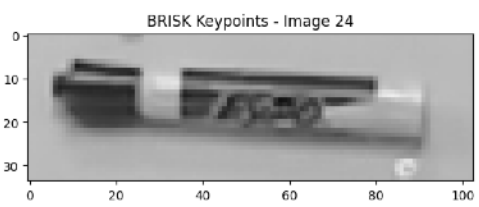




For marker, SIFT and KAZE find a lot of points such as text, lid, edges but BRISK fails.





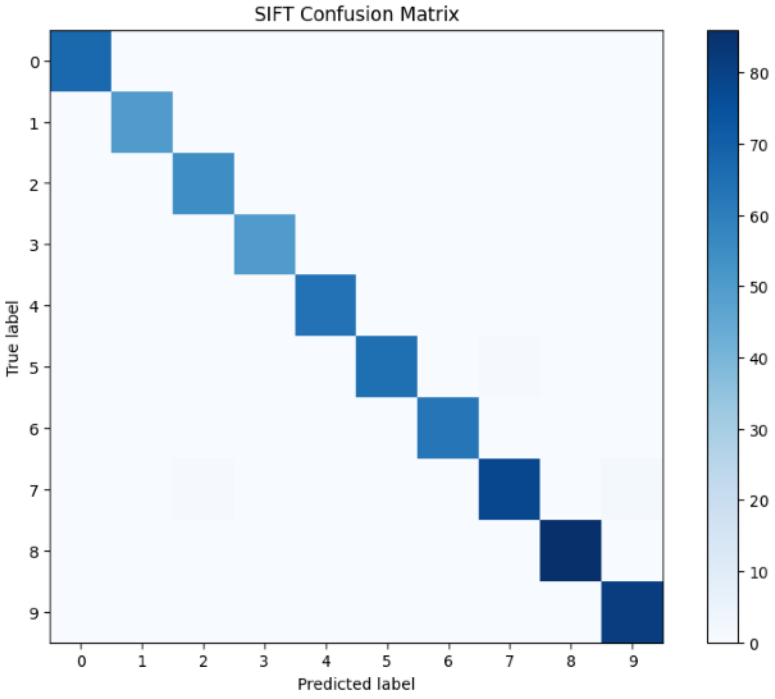


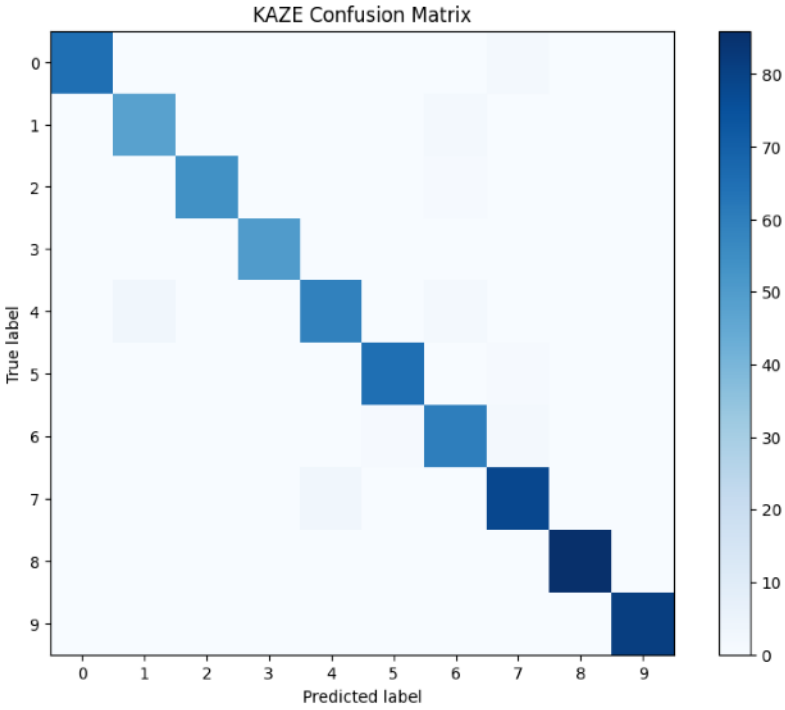
-Step 8 : Get the prediction for each test image

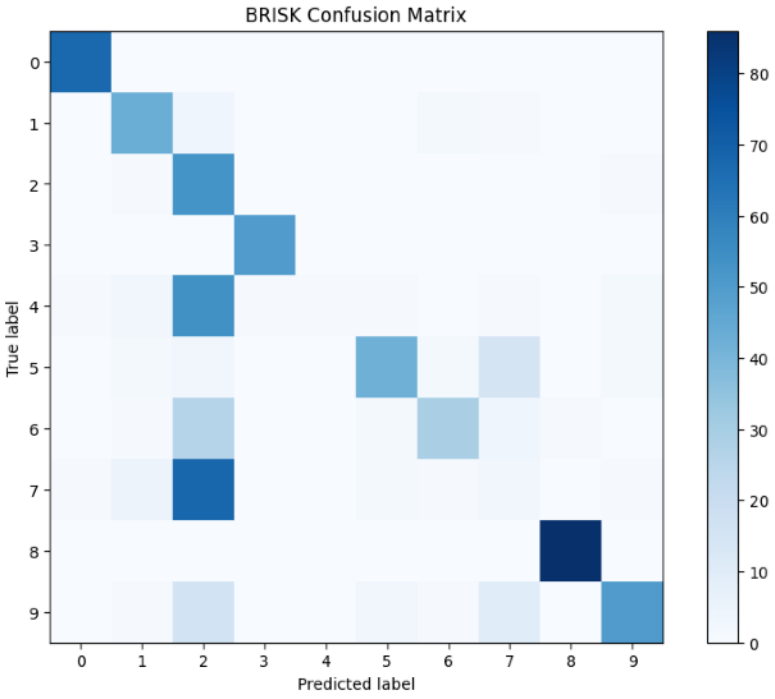
It gets the predictions by using match_and_predict. These predictions are the index of the training images. We need the labels of these images.

- Step 9 : Compute confusion matrices

It computes the confusion matrices by sending the real class label and the label of the training image which is selected as the best match.







Since SIFT is the most accurate detector, it has the best result. KAZE has some flaws but it still has a great result. Whereas, BRISK fails at class 4 and 7 which are dry battery and hand towel because it didn't find any key points for these objects as we can see from the visualization part. The class 6, garlic has some problems as well because of the same reason. However, it has a good result for the class 9, marker even though it couldn't find the key points.