GTU Department of Computer Engineering
CSE 463 - Fall 2024
Homework 2 Report

Emirkan Burak Yılmaz
1901042659

1 Contents

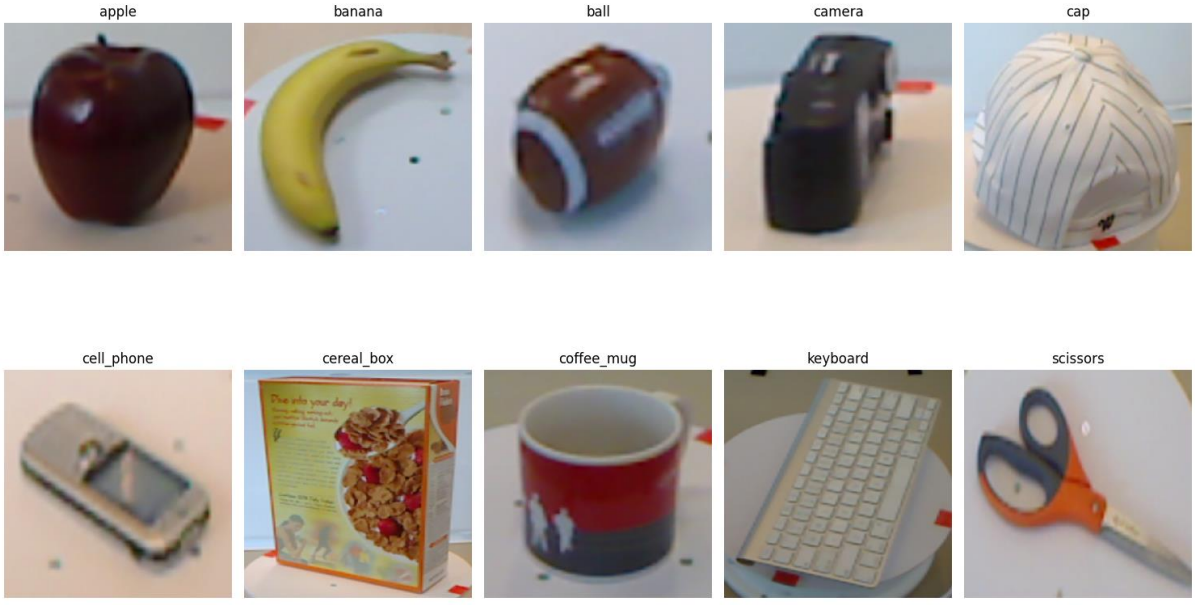
2 Introduction

This project involves implementing a simple object recognition algorithm based on feature detectors and descriptors using the RGB parts of the RGB-D Object Dataset from the University of Washington. The steps outlined in this project include selecting a subset of objects, using different feature detectors and descriptors, and evaluating the performance of the recognition algorithm.

3 Dataset

The dataset used is the RGB-D Object Dataset, which includes 300 objects across 51 categories. For this experiment, objects from 10 different categories were selected. A total of 200 images per object were randomly subsampled, ensuring a variety of angles for each object. The chosen objects are:

- Apple
- Banana
- Ball
- Camera
- Cap
- Cell Phone
- Cereal Box
- Coffee mug
- keyboard
- Scissors



4 Feature Detector and Descriptor

Three different feature detectors and descriptors were used from the OpenCV library:

- SIFT (Scale-Invariant Feature Transform): SIFT is a widely used algorithm known for its robustness to changes in scale, rotation, and illumination. It detects key points in images and generates descriptors that encode information about the local image regions surrounding these key points. SIFT descriptors are invariant to image scaling and rotation, making them suitable for object recognition tasks where objects may appear in different scales and orientations.
- ORB (Oriented FAST and Rotated BRIEF): ORB combines two keypoint detection algorithms: FAST (Features from Accelerated Segment Test) for keypoint detection and BRIEF (Binary Robust Independent Elementary Features) for feature description. ORB is computationally efficient and offers good performance in terms of speed and accuracy. It is particularly useful in real-time applications and scenarios where computational resources are limited.
- BRISK (Binary Robust Invariant Scalable Keypoints): BRISK uses octagonal sampling for keypoint detection and binary tests for descriptors. It prioritizes computational efficiency while maintaining robustness to scale, rotation, and viewpoint changes.

FAST and BRIEF alone were not used since they don't provide feature descriptors on their own in OpenCV library.

5 Object Recognition Algorithm

The object recognition algorithm involves four primary steps:

1. Feature Extraction: This step involves extracting features from images using a selected method, such as SIFT. These features include keypoints and descriptors that describe local aspects of the image.
2. Visual Vocabulary Creation: Here, the extracted features are grouped using k-means clustering to form a visual vocabulary, commonly referred to as a Bag of Words (BoW). Each cluster of features represents a visual word.
3. Feature Transformation: The images are then transformed into histograms of visual words. Each histogram bin corresponds to a visual word, with the bin count indicating the number of features that match this visual word.
4. Classification: Support Vector Machine (SVM) with a linear kernel is trained on these histogram representations. The SVM classifies new data based on the learned patterns.

```python
class FeatureExtractor(BaseEstimator, TransformerMixin):
    def __init__(self, n_clusters=100, method='SIFT'):
        self.n_clusters = n_clusters
        self.method = method
        self.kmeans = KMeans(n_clusters=self.n_clusters)
        self.scaler = StandardScaler()
        self.detector = self.init_detector()

    def init_detector(self):
        if self.method == 'SIFT':
            return cv2.SIFT_create()
        elif self.method == 'ORB':
            return cv2.ORB_create()
        elif self.method == 'BRISK':
            return cv2.BRISK_create()
        else:
            raise ValueError(f"Unsupported method: {self.method}")

    def fit(self, X, y=None):
        features = self.extract_features(X)
        self.kmeans.fit(features)
        return self

    def transform(self, X):
        # Transform the images into histograms of visual words
        histograms = self.create_histograms(X)
        # Standardize the histograms
        return self.scaler.fit_transform(histograms)
    def extract_features(self, images):
        descriptors_list = []
        for img in images:
            if self.method == 'SIFT' or self.method == 'ORB' or self.method == 'BRISK':
                keypoints, descriptors = self.detector.detectAndCompute(img, None)
                if descriptors is not None:
                    descriptors_list.extend(descriptors)
            else:
                raise ValueError(f"Unsupported method: {self.method}")
        return np.array(descriptors_list)

    def create_histograms(self, images):
        histograms = []
        for img in images:
            if self.method == 'SIFT' or self.method == 'ORB' or self.method == 'BRISK':
                keypoints, descriptors = self.detector.detectAndCompute(img, None)
            else:
                raise ValueError(f"Unsupported method: {self.method}")
            if descriptors is not None:
                # Predict the cluster for each descriptor and create a histogram
                histogram = np.zeros(self.n_clusters)
                cluster_result = self.kmeans.predict(descriptors)
                for i in cluster_result:
                    histogram[i] += 1
                histograms.append(histogram)
            else:
                histograms.append(np.zeros(self.n_clusters))
        return np.array(histograms)
```

Figure 1: Custom feature extractor class for transforming features to histograms.

The Bag of Words (BoW) model treats image features as discrete words in a visual vocabulary. After clustering, each descriptor is assigned to the nearest cluster center, transforming the continuous descriptors into discrete visual words. Each image is represented by a histogram summarizing the distribution of these words, creating a fixed-size vector that is ideal for machine learning applications.

The choice of SVMs for classification is due to their efficiency in high-dimensional spaces, typical of image data. SVMs are particularly effective with small to medium-sized datasets and are robust against overfitting, making them suitable for this application. Unlike deep learning approaches that require large datasets, SVMs provide excellent performance with smaller data sets and clear separation margins.

```
# Train and evaluate the classifier for each feature extraction method
methods = ['SIFT', 'ORB', 'BRISK']

# Loop through each feature extraction method
for method in methods:
    # Create a pipeline with FeatureExtractor and SVM classifier
    # FeatureExtractor: extracts features from images using the specified method
    # SVC: SVM classifier with a linear kernel
    pipeline = make_pipeline(FeatureExtractor(method=method), SVC(kernel='linear'))

    # Fit the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Predict the labels for the test data
    y_pred = pipeline.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy using {method}: {accuracy:.2f}")

    display_confusion_matrix(y_test, y_pred, selected_categories)
```

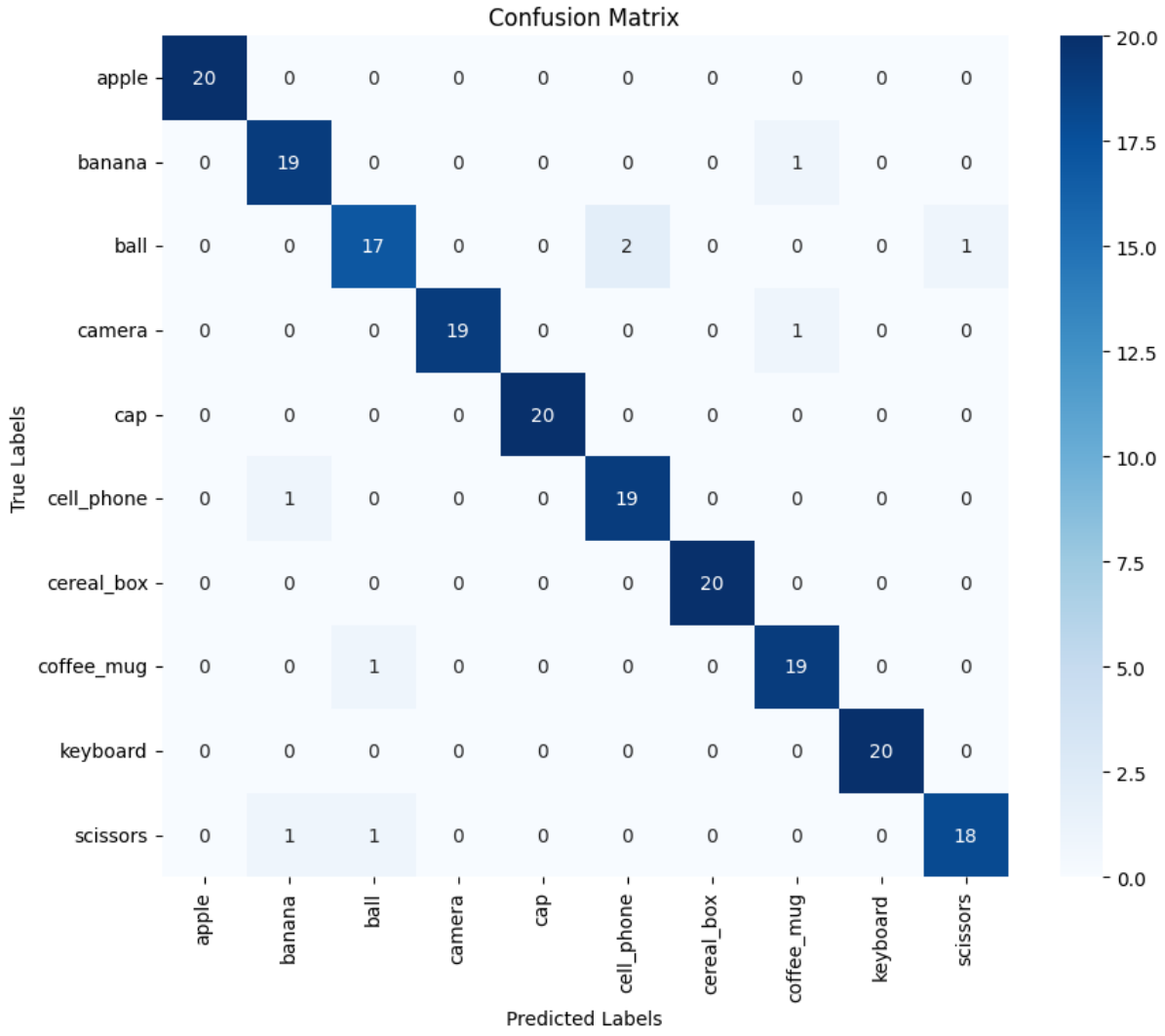
Figure 2: Feature extraction and training pipeline.

In summary, this approach involves clustering similar features, with each image described by a histogram representing these clusters. This histogram serves as a compact, fixed-size feature vector ideal for machine learning. The choice of SVMs leverages their effectiveness in handling high-dimensional data and their robust performance with smaller datasets, making them perfectly suited for classifying these feature vectors.

6 Results

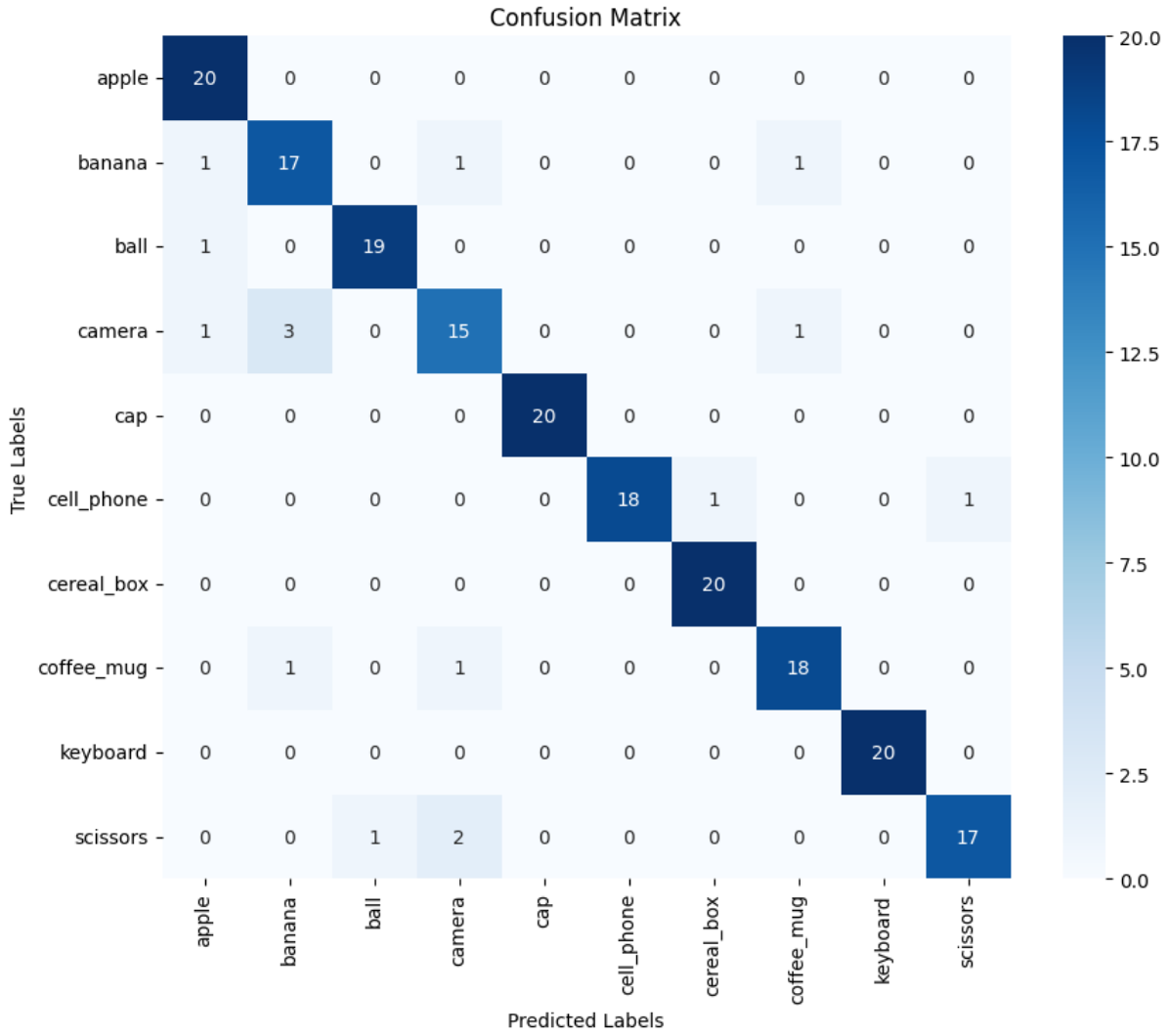
- SIFT

Accuracy: 0.95



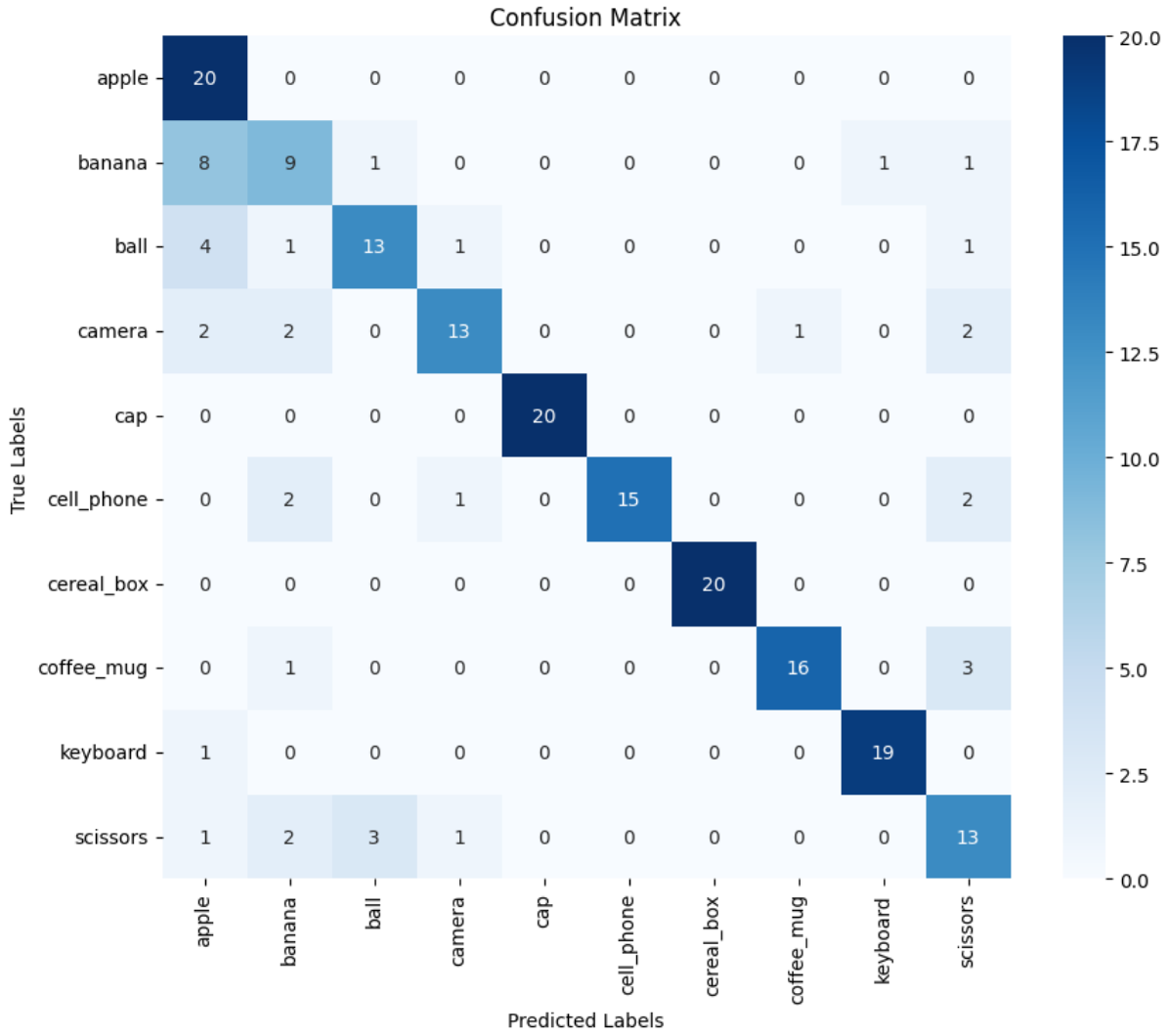
- ORB

Accuracy: 0.92



- BRISK

Accuracy: 0.79



7 Discussion

The results show that SIFT provided the highest accuracy in object recognition among the three methods. This is expected as SIFT is known for its robustness to scale and rotation changes. ORB and BRISK, being faster, provided slightly lower accuracy but are still good alternatives considering their computational efficiency.

Reasons for Performance Differences

- SIFT: More robust to scale and rotation changes, leading to higher accuracy.
- ORB: Faster but may miss some features, leading to slightly lower accuracy.
- BRISK: Provides a good balance between speed and accuracy but is less robust compared to SIFT.

Reasons for Failures

- Occlusion: Some objects might be partially occluded in the images, leading to fewer detectable features.
- Lighting Variations: Differences in lighting conditions can affect feature detection and descriptor matching.

8 Conclusion

The results demonstrate that SIFT is the most effective feature detector and descriptor for this object recognition task, followed by ORB and then BRISK. SIFT's detailed and robust descriptors contribute to its high accuracy, making it ideal for tasks where precision is critical. ORB offers a good balance between performance and computational efficiency, making it suitable for real-time applications. BRISK, while efficient, may not provide the level of detail required for high-accuracy object recognition. These findings highlight the trade-offs between accuracy and computational efficiency when choosing feature detectors and descriptors for object recognition. Future work could involve integrating more advanced techniques such as deep learning-based feature extraction to further improve recognition performance.