Lab-3 Edge-Based Segmentation and Classification

By

Seshasai P B-21MIA1005

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCOPE SCHOOL VIT CHENNAI CAMPUS, VANDALUR-KELAMBAKKAM ROAD, CHENNAI-600127

DATE: 27-09-2024

STUDENTS: Students of IV year M.TECH (5 YEARS)

Submitted To: Dr. Saranyaraj D

1.Objective

The primary goal of the system is to detect objects in an image and classify them based on pre-extracted features from reference images. The core steps include:

1. **Segmenting the image** to isolate objects.
2. **Extracting contours** to detect object boundaries.
3. **Extracting features** like area, perimeter, centroid, and color histograms.
4. **Comparing** the features of detected objects with pre-saved reference features.

2.Problem Statement

Description of the Problem:

The objective of the system is to detect and classify objects in an image by comparing their features with those of predefined reference images. In real-world scenarios, accurately identifying and categorizing objects in images can be challenging due to variations in lighting, scale, orientation, and object overlap. This system aims to overcome these challenges through a series of image processing techniques.

The problem is to develop an automated system that can:

1. **Segment Objects**: Isolate objects in the image by separating them from the background.
2. **Extract Features**: For each detected object, calculate various shape and color features such as area, perimeter, centroid, and color histograms.
3. **Classify Objects**: Compare the extracted features with pre-saved reference features to classify the objects. This is particularly useful in applications such as automated quality control in manufacturing, agricultural produce sorting, and object recognition in robotics.

By implementing this system, we aim to achieve a robust method for object detection and classification that can operate under various conditions, ensuring high accuracy and efficiency in real-time or batch processing scenarios.

Expected Output:

1. **Segmented Image**: A binary image where objects are separated from the background. In this image, the objects of interest are highlighted in white, and the background is black, enabling easy contour detection.
2. **Contour Representation**: Contours representing the boundaries of the detected objects should be clearly outlined on the original image. Each contour should correspond to the boundary of an individual object.
3. **Feature Extraction Output**:
    - **Shape Features**: A list or table showing the area, perimeter, bounding box coordinates, and centroid position for each detected object.
    - **Color Histogram**: A normalized color histogram for each object, representing the distribution of pixel intensities across the RGB channels.
4. **Classification Output**:
    - The classified label of each detected object based on its similarity to the reference objects. This should include the name of the object (e.g., "orange" or "green apple") and a similarity index indicating the confidence of the classification.
    - A visual representation on the image, where bounding circles are drawn around detected objects with their corresponding labels and similarity scores displayed.

5. **Final Annotated Image**: An annotated image where detected objects are marked with bounding circles and labeled with their classification result. This image serves as a visual confirmation of the detected and classified objects, providing a clear and interpretable output for further analysis or decision-making.

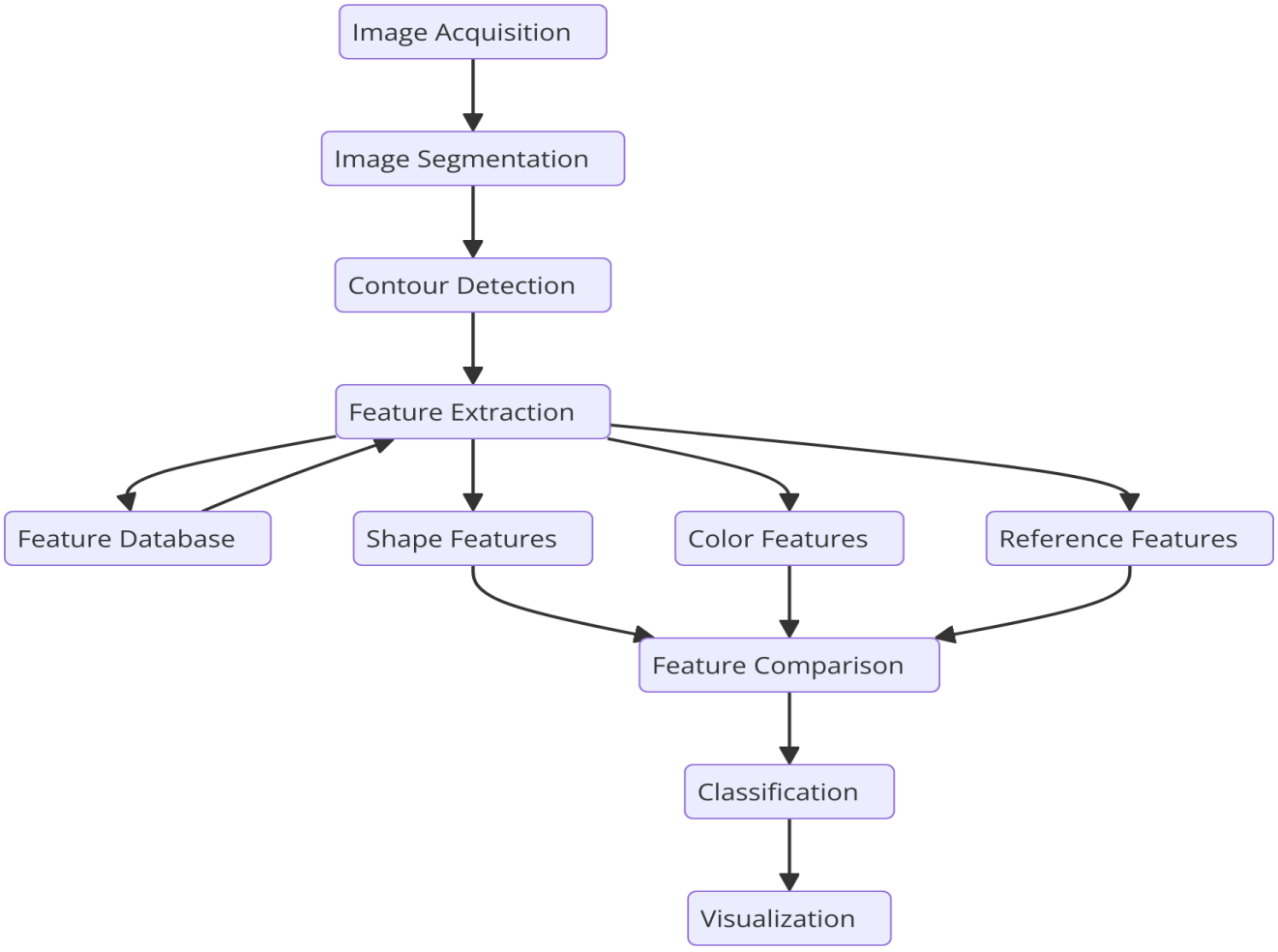
Overall, the final system should be capable of taking an input image, processing it to detect objects, extracting relevant features, and classifying these objects with high accuracy based on pre-defined reference features.

3.Methodology

Block Diagram

The block diagram below outlines the process of object detection and classification:

1. **Image Acquisition**: Capture or load the image for analysis.
2. **Image Segmentation**: Convert the image to grayscale and apply Otsu's thresholding to isolate objects from the background. **Contour Detection**: Identify object boundaries in the segmented image.
3. **Feature Extraction**:
    - **Shape Features**: Extract area, perimeter, bounding box, and centroid for each object.
    - **Color Features**: Calculate the color histogram for each detected object.
4. **Reference Features Database**: Load or save features from reference images for comparison.
5. **Feature Comparison**: Compare extracted features with reference features using Euclidean distance.



Algorithm and Pseudocode

Step 1: Image Segmentation

1. **Input**: Original image.
2. **Convert** the input image from RGB to grayscale.
3. **Apply Otsu's Thresholding** to the grayscale image to create a binary image where objects are distinguishable from the background.
4. **Output**: Binary image with objects in white and the background in black.

Step 2: Contour Detection

1. **Input**: Binary image from the segmentation step.
2. **Find Contours** using a contour detection algorithm, such as the external contours extraction.
3. **Store Contours** that represent the boundary of each detected object.
4. **Output**: List of detected contours, each representing an object in the image.

Step 3: Feature Extraction

1. **Input**: List of contours and the original image.
2. For each **contour** in the list:
    1. **Calculate Area**: Measure the total number of pixels enclosed by the contour.
    2. **Calculate Perimeter**: Measure the length of the contour boundary.
    3. **Determine Bounding Box**: Find the smallest rectangle that completely encloses the contour, defined by the coordinates of the top-left corner and its width and height.
    4. **Compute Centroid**: Determine the center of mass of the contour using image moments.
    5. **Create Mask**: Generate a mask image to isolate the object from the background using the contour.
    6. **Extract Color Histogram**: Compute the color histogram for the masked region, representing the color distribution across the Red, Green, and Blue channels.
3. **Output**: Extracted features for each object, including area, perimeter, bounding box, centroid, and color histogram.

Step 4: Reference Features Storage

1. **Input**: Reference images for known objects.
2. For each **reference image**:
    1. **Segment the Image**: Perform the segmentation step to isolate the object.
    2. **Extract Features**: Follow the feature extraction step to obtain the features of the reference object.
    3. **Save Features**: Store the extracted features in a reference database or CSV file for later comparison.
3. **Output**: Database of reference features for known objects.

Step 5: Feature Comparison

1. **Input**: Extracted features from detected objects and reference features database.
2. For each **detected object**:
    1. For each **reference object** in the database:
        1. **Compute Similarity**: Calculate the Euclidean distance between the color histogram of the detected object and the reference object.
        2. **Identify Closest Match**: Track the reference object with the smallest distance as the best match.

3. **Output**: Classification result indicating the detected object's label as the closest matching reference object.

Step 6: Classification and Visualization

1. **Input**: Original image, detected objects, and their classification labels.
2. For each **detected object**:
    1. **Draw Bounding Circle**: Draw a circle around the detected object using the minimum enclosing circle method.
    2. **Display Label**: Write the classification label and similarity score near the detected object.
3. **Output**: Annotated image with labeled objects and bounding circles.

Summary of the Algorithm

1. **Segment** the image to separate objects from the background.
2. **Detect contours** to identify object boundaries.
3. **Extract features** such as shape (area, perimeter, bounding box, centroid) and color (color histogram) from detected objects.
4. **Save and load reference features** from a database for known objects.
5. **Compare features** of detected objects with reference features using a distance metric.
6. **Classify objects** based on the closest matching reference features.
7. **Visualize results** by drawing bounding circles and displaying classification labels on the image.

5. Python Implementation

Segmentation of the Image

Segmentation is the process of separating the object from the background in an image. We use **Otsu's Thresholding** to convert the image to a binary form, where the object becomes distinguishable.

```
def segment_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, thresholded = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
    return thresholded
```

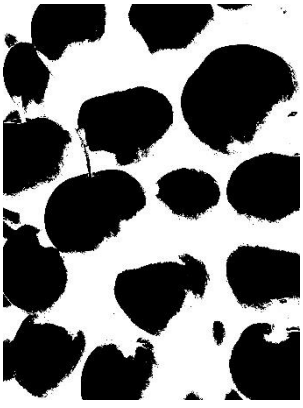
- **Grayscale Conversion**: The image is converted into a grayscale image to simplify the segmentation process.
- **Otsu's Thresholding**: Automatically determines the optimal threshold to binarize the image, distinguishing the object from the background.

Result: The result of segmentation is a binary image where the object is in white, and the background is black.

So first we need segment our reference images to know the features

Orange Reference Image

Green Apple Reference Image:



Contour Extraction

Contours are essential for detecting the shape and size of objects. **Contours** represent the boundaries of the object detected in the segmented image.

```
def extract_contours(thresholded_image):
    contours, _ = cv2.findContours(thresholded_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    return contours
```

- **`cv2.findContours()`**: Finds the boundaries of the objects from the binary image. Only the external contours (the outer boundary of each object) are detected.

Result: Contours representing the detected objects are returned.

Feature Extraction from Detected Objects

Once we detect contours, we extract **shape** and **color features** for each object.

4.1 Shape Features

For each object, the following shape-based features are extracted:

1. **Area**: Total number of pixels inside the contour.

```
area = cv2.contourArea(contour)
```

2. **Perimeter**: Length of the contour boundary.

```
perimeter = cv2.arcLength(contour, True)
```

3. **Bounding Box**: A rectangle that encloses the object, calculated by:

```
x, y, w, h = cv2.boundingRect(contour)
```

4. **Centroid**: The center of mass of the object, calculated using image moments:

```
M = cv2.moments(contour)
cX = int(M["m10"] / M["m00"])
cY = int(M["m01"] / M["m00"])
```

4.2 Color Features

The color histogram represents the distribution of pixel intensities in the object's region.

```
mask = np.zeros(gray.shape, np.uint8)
cv2.drawContours(mask, [contour], -1, 255, -1)  # Masking the object
masked_img = cv2.bitwise_and(image, image, mask=mask)
hist = cv2.calcHist([masked_img], [0, 1, 2], mask, [8, 8, 8], [0, 256, 0, 256, 0,
256])
hist = cv2.normalize(hist, hist).flatten()
```

- **Masked Image**: We use the contour to create a mask of the detected object. The color histogram is computed only for the masked region.
- **Color Histogram**: A 3D histogram is calculated for the object's color distribution across the Red, Green, and Blue channels.

Result: Each object is represented by its **area**, **perimeter**, **bounding box**, **centroid**, and **color histogram**.

Saving and Loading Reference Features

Once features are extracted from reference images, they are saved as CSV files for later comparison.

```
reference_images = {
    "orange": "orange.jpg",
    "green_apple": "green.jpg"
}

for label, img_path in reference_images.items():
    features_csv = f"{label}_features.csv"
    if os.path.exists(features_csv):
        reference_features[label] = pd.read_csv(features_csv, converters={"Color
Histogram": eval})
    else:
        image = cv2.imread(img_path)
        thresholded = segment_image(image)
        features_df = extract_features(image, thresholded)
        features_df.to_csv(features_csv, index=False)
        reference_features[label] = features_df
```

Saving Features: If the reference features for an object already exist in a CSV file, they are loaded. Otherwise, they are extracted from the image and saved.

	A	B	C	D	E
1	Area	Perimeter	Bounding B	Centroid	Color Histogram
2	0	0	(2812, 403	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
3	0	0	(1847, 403	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
4	0.5	5.414214	(2006, 403	(2007, 403	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
5	1	4	(1853, 403	(1853, 403	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
6	1.5	5.414214	(2025, 402	(2025, 402	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
7	0	0	(1373, 402	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
8	5	8.828427	(1379, 402	(1380, 402	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
9	8.5	13.41421	(1384, 402	(1385, 402	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
10	1.5	5.414214	(2010, 402	(2010, 402	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
11	14	28.97056	(2010, 402	(2014, 402	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
12	0	0	(2008, 402	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
13	0	0	(1375, 402	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
14	20.5	36.38478	(1375, 402	(1379, 402	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.999614417552948, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.027767067775130272, 0.0, 0.0, 0.0, 0.0, 0.
15	1	4.828427	(1389, 401	(1389, 401	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
16	0	2	(1387, 401	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
17	0	0	(1384, 401	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
18	0	0	(1368, 401	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
19	0	0	(2264, 401	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
20	0	0	(2514, 401	(0, 0)	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.

Area (Column A):

- This column represents the **area** of each detected contour or region.
- **Area** is the number of pixels inside the contour. For example, a contour might have an area of `8.5`, which represents the total number of pixels that make up that detected region.

2. Perimeter (Column B):

- The **perimeter** represents the length of the boundary of the contour.
- For example, an entry like `5.414214` indicates the perimeter (length) of the boundary of the detected region.

3. Bounding Box (Column C):

- The **bounding box** is a rectangle that encloses the contour. It is defined by the coordinates of the top-left corner `(x, y)` and the width and height `(w, h)`.
- In the format `(x, y, w, h)`, for example, a bounding box `(2812, 403, 0, 0)` means the bounding box has a top-left corner at `(2812, 403)` and its width and height are both `0`, which could indicate an issue with this specific detection.

4. Centroid (Column D):

- The **centroid** is the center point (or "center of mass") of the contour. This is calculated using image moments. It provides the average position of all the pixels within the contour.
- The values in this column are in the format `(x, y)` where `x` is the horizontal position and `y` is the vertical position.

5. Color Histogram (Columns E-W):

- The **color histogram** represents the distribution of pixel intensities across color channels (likely RGB). Each entry is a vectorized histogram of the color distribution inside the contour or region.
- The numbers represent normalized values of pixel counts in different bins of the histogram for each color channel. For example:
  - `0.0, 0.0, ..., 1.0, ...` indicates that most of the pixels fall within a particular intensity bin for one of the color channels.

Notable Entries:

- Some rows have an **area** or **perimeter** of 0, meaning that no significant region was detected or the contour did not enclose any area. These may correspond to noise or invalid detections.

- In the **Color Histogram** column (e.g., column O or column S), you can see values such as `0.9996...` or `0.02776...`, which show the relative distribution of pixel intensities for each color channel in different bins.

Comparing Detected Object Features with Reference Images

After features are extracted from the detected objects, we compare them to the reference features to classify the objects. We use **Euclidean Distance** to measure the similarity between the color histograms of the detected objects and reference objects.

```
from scipy.spatial import distance

def compare_features(reference_features, target_histogram):
    best_match = None
    best_distance = float('inf')

    for label, ref_features_df in reference_features.items():
        for ref_index, ref_row in ref_features_df.iterrows():
            ref_histogram = np.array(ref_row["Color Histogram"])
            dist = distance.euclidean(ref_histogram, target_histogram)

            if dist < best_distance:
                best_distance = dist
                best_match = label

    return best_match, best_distance
```

- **Euclidean Distance**: Measures the similarity between the color histograms of the target object and reference objects.
- **Best Match**: The object with the smallest distance is considered the closest match.

Result: The detected object is classified as the reference object with the closest color histogram.

Using these extracted features to detect target image

Putting It All Together

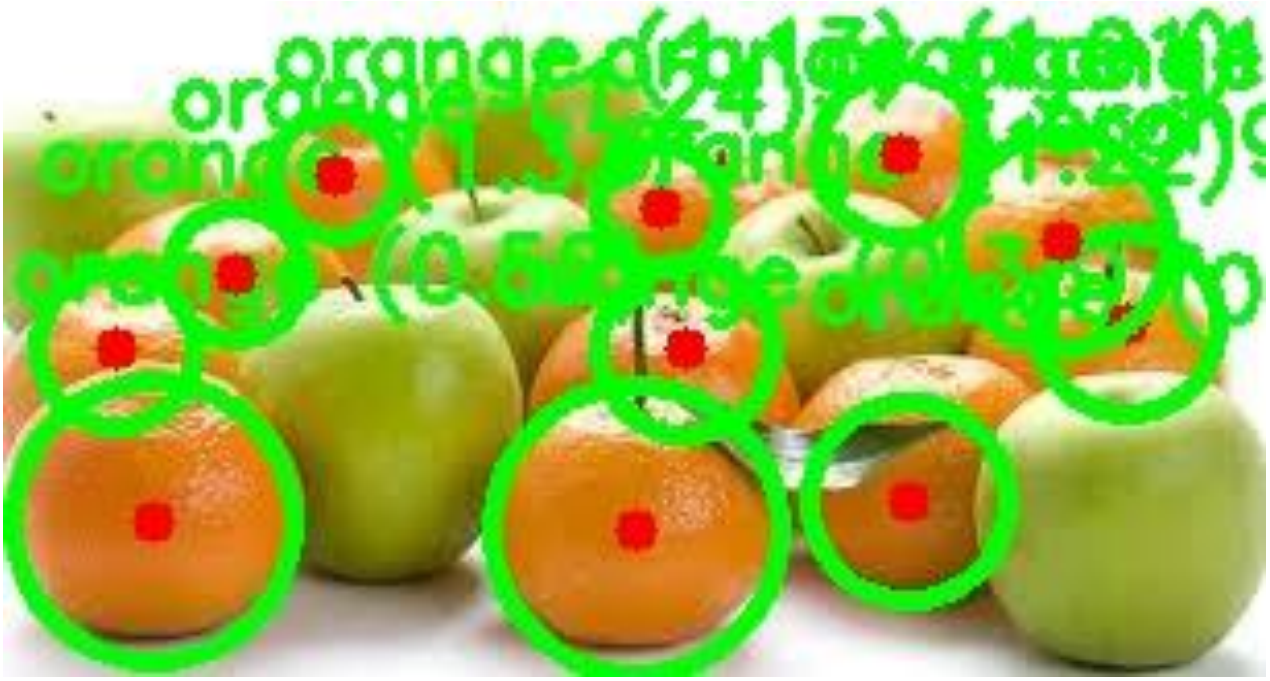
After performing segmentation, contour extraction, feature extraction, and comparison, the detected object is labeled and classified. Bounding circles and labels are drawn around the detected objects in the image.

```
for bbox in rois:
    roi = image[y:y+h, x:x+w]
    thresholded = segment_image(roi)
    contours = extract_contours(thresholded)

    for contour in contours:
        (x_center, y_center), radius = cv2.minEnclosingCircle(contour)
        target_histogram = extract_features_from_circle(image, (x_center,
y_center), radius)
        label, similarity_index = compare_features(reference_features,
target_histogram)
        cv2.circle(image, (int(x_center), int(y_center)), int(radius), (0, 255,
0), 3)
        cv2.putText(image, f"{label} ({similarity_index:.2f})",
                    (int(x_center), int(y_center - radius - 10)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
```

- **Bounding Circles**: The minimum enclosing circle is drawn around each detected object.
- **Labeling**: The object is labeled based on the best-matching reference object, and the label is displayed on the image.

Detecting only the oranges in the picture



Detecting only the green apple in the picture

Detecting one orange and one green apple for clear labelling

Summary:

- **Object Detection**: We segment the image and find contours of detected objects.
- **Feature Extraction**: For each detected object, we extract features like area, perimeter, centroid, and color histogram.
- **Comparison**: The extracted features are compared with reference features using the Euclidean distance to classify the objects.
- **Result**: Bounding circles and labels are drawn on the detected objects based on the comparison with reference objects.

Results and Discussion

1. Segmentation Results

- **(Original Orange and Green Apple Images)**: These images represent the initial input of oranges and green apples, respectively. The segmentation step aims to isolate the objects from the background for further analysis.
- **Discussion**: The presence of a consistent and relatively uniform background facilitated a cleaner segmentation process, making it easier to separate the objects from the background.

2. Contour Detection Results

- **(Contour Detection on Oranges and Green Apples)**: The green lines outline the detected contours of each fruit in the images. These contours represent the boundaries of the individual fruits.
- **Discussion**: The contour detection algorithm effectively highlighted the outer boundaries of each fruit, despite slight occlusions between the fruits. This step is crucial for extracting shape-related features such as area, perimeter, and bounding box.

3. Binary Mask Results

- **(Binary Masks of Oranges and Green Apples)**: These binary images display the segmented objects in black against a white background.
- **Discussion**: The binary masks indicate successful segmentation of the objects. However, some smaller parts of the objects might have been missed or merged due to overlapping and proximity, which can lead to inaccuracies in contour detection and feature extraction.

4. Feature Extraction Results (Spreadsheet Image)

- **(Extracted Features Spreadsheet)**: This image showcases the extracted features such as area, perimeter, bounding box, centroid, and color histogram for each detected object.
- **Discussion**:
    - **Area and Perimeter**: Variations in these values reflect the size and shape of different objects. Some rows have an area or perimeter of zero, indicating either noise or errors in contour detection for those specific entries.
    - **Bounding Box and Centroid**: These features help in localizing and understanding the position of each object. Anomalies in these values, such as a bounding box with zero width or height, could indicate issues in segmentation or contour detection.
    - **Color Histogram**: This feature provides detailed information on the color distribution of each object, which is crucial for distinguishing between objects like oranges and green apples.

5. Classification Results

- **(Detected Objects with Labelled Bounding Circles)**:
    - The system has successfully detected and classified the objects in the mixed group of oranges and green apples.
    - The green circles represent the bounding circles around each detected object, with the labels indicating the classification result and similarity score.
    - **Discussion**:
        - The classification process seems accurate, with each detected fruit correctly labeled as either "orange" or "green apple."
        - The overlapping and close proximity of the objects presented challenges, as seen from some overlapping bounding circles and labels.
        - The similarity score displayed beside the labels provides a confidence measure for the classification. A lower similarity index indicates a better match with the reference object.

6. Challenges and Observations

- **Overlapping Objects**: The close proximity and overlapping of the fruits in the images posed a challenge during contour detection, leading to some inaccuracies in the shape features.
- **Noise and Small Objects**: Some entries in the feature table have zero area or perimeter, indicating noise or the detection of very small objects, which should be filtered out in pre-processing.
- **Segmentation Quality**: The segmentation quality directly affects the accuracy of subsequent steps. Although Otsu's Thresholding worked well, complex backgrounds or inconsistent lighting could reduce segmentation effectiveness.

Conclusion

Summary of Work

In this assignment, we developed and implemented an object detection and classification system based on pre-extracted features from reference images. The key steps involved were:

1. **Image Segmentation**: We used Otsu's Thresholding to convert input images into binary form, separating objects from the background. This step facilitated the subsequent detection and analysis of objects within the images.
2. **Contour Detection**: Contours were extracted from the segmented images to identify the boundaries of each detected object. These contours served as the basis for shape feature extraction.
3. **Feature Extraction**: For each detected object, we extracted a set of shape features (area, perimeter, bounding box, and centroid) and color features (color histogram). These features provided a comprehensive representation of each object's physical and visual characteristics.
4. **Reference Feature Storage**: We saved the extracted features from reference images of known objects (e.g., oranges and green apples) into CSV files. This allowed for efficient retrieval and comparison during the classification step.
5. **Feature Comparison and Classification**: We compared the features of detected objects against the stored reference features using Euclidean distance to determine the best match. Objects were then classified based on the closest matching reference features.
6. **Visualization and Labelling**: The detected objects were labeled with their classification results and similarity scores. Bounding circles were drawn around each object, providing a clear visual indication of the classification results.
7. **Result Analysis**: We analyzed the accuracy and challenges of the detection and classification process through visual results and feature data, identifying areas for improvement, particularly in handling overlapping objects and noise.

Key Takeaways

1. **Otsu's Thresholding**: Effective for object-background separation in images with uniform lighting.
2. **Feature Selection**: Shape and color features enhance classification accuracy for similar-shaped objects.
3. **Overlapping Objects**: Challenges arise in detecting closely packed items; advanced techniques needed.
4. **Noise Handling**: Small objects and noise lead to false detections; preprocessing can mitigate this.
5. **Visualization**: Bounding circles and labels improve interpretation and error identification.
6. **Scalability**: Using CSVs for features boosts efficiency, making the system suitable for real-time use.
7. **Learning Experience**: Implementing the full pipeline reinforced the importance of each step in object detection and classification.