Name: Seshasai.PB Reg\_N0: 21MIA1005 Image and Video Analytics

Object Detection and Labelling Script

# Step-by-Step Breakdown

1. **Image Segmentation (segment\_image function) Function:**

def segment\_image(image):

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

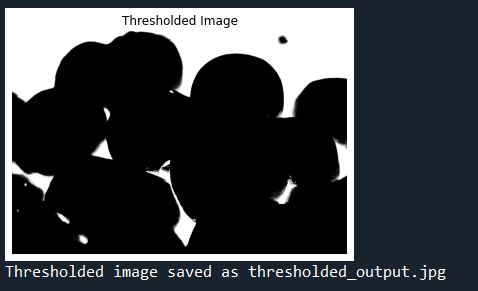
\_, thresholded = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV

+ cv2.THRESH\_OTSU)

return thresholded

## Description:

* + **Input:** A color image (BGR format).
  + **Process:** Converts the image to grayscale and applies Otsu's thresholding to segment the image into foreground and background.
  + **Output:** A binary (thresholded) image where the objects of interest are white, and the background is black.



# Feature Extraction (extract\_features function) Function:

def extract\_features(image, thresholded):

contours, \_ = cv2.findContours(thresholded, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

features = {

"Area": [],

"Perimeter": [],

"Bounding Box": [],

"Centroid": [], "Color Histogram": []

}

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

for contour in contours: # Shape Features

area = cv2.contourArea(contour) perimeter = cv2.arcLength(contour, True) x, y, w, h = cv2.boundingRect(contour) M = cv2.moments(contour)

if M["m00"] != 0:

cX = int(M["m10"] / M["m00"])

cY = int(M["m01"] / M["m00"])

else:

cX, cY = 0, 0

# Color Features

mask = np.zeros(gray.shape, np.uint8) cv2.drawContours(mask, [contour], -1, 255, -1) 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()

# Store features features["Area"].append(area) features["Perimeter"].append(perimeter) features["Bounding Box"].append((x, y, w, h)) features["Centroid"].append((cX, cY))

features["Color Histogram"].append(hist.tolist()) # Save as a list to be CSV-compatible

return pd.DataFrame(features)

## Description:

* **Input:** Original image and a thresholded image.
* **Process:** Detects contours in the thresholded image and extracts various features for each detected object:
  + **Shape Features:** Area, perimeter, bounding box, and centroid.
  + **Color Features:** A 3D color histogram in the RGB color space.
* **Output:** A pandas DataFrame containing all the extracted features.



# Creating Reference Features (create\_reference\_features function) Function:

def create\_reference\_features(): reference\_images = {

"orange": "orange.jpg", "green\_apple": "green.jpg"

}

reference\_features = {}

for label, img\_path in reference\_images.items(): features\_csv = f"{label}\_features.csv"

if os.path.exists(features\_csv):

# Load features if already saved print(f"Loading saved features for {label} from

{features\_csv}")

reference\_features[label] = pd.read\_csv(features\_csv, converters={"Color Histogram": eval})

else:

# Extract and save features

print(f"Extracting and saving features for {label}") image = cv2.imread(img\_path)

if image is None:

print(f"Error: Image {img\_path} not found.") continue

thresholded = segment\_image(image)

features\_df = extract\_features(image, thresholded)

features\_df.to\_csv(features\_csv, index=False) reference\_features[label] = features\_df

return reference\_features

## Description:

* **Purpose:** Extracts features from reference images (e.g., oranges and green apples) and saves them for future use.
* **Input:** A dictionary mapping labels (e.g., "orange", "green\_apple") to image file paths.

## Process:

* + For each reference image, check if the features have already been extracted and saved to a CSV file.
  + If not, segment the image, extract the features, and save them to a CSV file.
* **Output:** A dictionary where each key is a label, and the corresponding value is a DataFrame of features for that label.

## Reference Images

Orange



Green apple



# Extracting Features from a Bounding Box (extract\_features\_from\_bbox function)

**Function:**

def extract\_features\_from\_bbox(image, bbox): x, y, w, h = bbox

cropped\_image = image[y:y+h, x:x+w]

mask = np.ones(cropped\_image.shape[:2], dtype="uint8") \* 255 hist = cv2.calcHist([cropped\_image], [0, 1, 2], mask, [8, 8, 8],

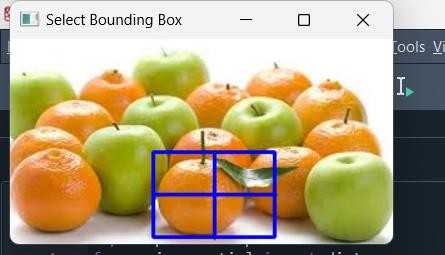
[0, 256, 0, 256, 0, 256])

hist = cv2.normalize(hist, hist).flatten()

return hist

## Description:

* + **Input:** Original image and a bounding box (x, y, w, h).
  + **Process:** Crops the image according to the bounding box and extracts a normalized color histogram from this region.
  + **Output:** The extracted color histogram as a flattened array.



# Comparing Features for Object Detection (compare\_features function) Function:

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

## Description:

* + **Input:** A dictionary of reference features and the histogram of the target object.
  + **Process:** Compares the target histogram against each reference histogram using Euclidean distance.
  + **Output:** The label of the best matching reference object and the corresponding similarity index (Euclidean distance).





# Detecting and Labeling Objects (detect\_and\_label\_bbox function) Function:

def detect\_and\_label\_bbox(image, bbox, reference\_features): target\_histogram = extract\_features\_from\_bbox(image, bbox) label, similarity\_index = compare\_features(reference\_features,

target\_histogram)

x, y, w, h = bbox

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2) label\_text = f"{label} ({similarity\_index:.2f})" cv2.putText(image, label\_text, (x, y - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (0, 255, 0), 2)

return image, similarity\_index

## Description:

* + **Input:** Original image, bounding box, and reference features.
  + **Process:** Extracts the histogram from the bounding box, compares it with reference features, and labels the object with the best match and similarity index.
  + **Output:** The labeled image and the similarity index.