VISUAL RECOGNITION ASSIGNMENT 1

Name
Subbhashit Mukherjee

Roll Number
MT2023065

CONTENT

Draw a bounding box around the foreground object

Our first task is to create bounding boxes around the foreground object. We will be making bounding boxes around objects using a python library called OpenCV. OpenCV is a huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

We will be doing the following process to make boxes around objects:

1. **Image Loading**:

   The original color image is loaded using the cv2.imread function.

2. **Grayscale Conversion**:

   The loaded image is converted to grayscale using cv2.cvtColor to simplify subsequent processing steps. This results in a 2D grayscale image.

3. **Gaussian Blur**:

   Gaussian blur is applied to the grayscale image using cv2.GaussianBlur. This step helps reduce noise and smooth out the image, which is beneficial for subsequent edge detection.

4. **Canny Edge Detection**:

   Canny edge detection is performed on the blurred image using cv2.Canny. This algorithm identifies edges based on intensity gradients, producing a binary image highlighting significant edges.

5. **Contour Extraction**:

   Contours are extracted from the binary edge image using cv2.findContours. The external retrieval mode (cv2.RETR_EXTERNAL) is used to find only the outer contours, and the simple approximation method (cv2.CHAIN_APPROX_SIMPLE) is employed.

6. **Bounding Box Drawing**:

   Bounding boxes are drawn around the detected contours on a copy of the original image using cv2.drawContours. Each bounding box is represented by the coordinates of its top-left and bottom-right corners.

7. **Visualization**:

   The original color image, grayscale image, blurred image, Canny edges, and the image with drawn bounding boxes are visualized using Matplotlib.

Convert to gray-scale and detect edges using Canny edge detection method

Our second task is to convert the image to grayscale and detect edges in it using canny edge detection method. We used the following steps to perform this task:

1. **Image Loading**:

   The code loads an image using cv2.imread from the specified file path.The loaded image is in BGR format.

2. **Grayscale Conversion**:

   The loaded color image is converted to grayscale using cv2.cvtColor with the cv2.COLOR_BGR2GRAY conversion code. Grayscale conversion reduces the image to a single channel, representing pixel intensities.

3. **Gaussian Blur**:

   Gaussian blur is applied to the grayscale image using cv2.GaussianBlur with a (5,5) kernel and a standard deviation of 0. Blurring helps in reducing noise and creating a smoother image.

4. **Canny Edge Detection**:

   Canny edge detection is performed on the blurred image using cv2.Canny with threshold values of 30 and 105.

Use K-means for segmentation on the RGB image, and display the segmented region

Our final task is touse kmeans for segmentation and display the segmented region. We used the following steps to perform this task:

1. **Image Loading and Color Conversion**:

   The code loads an image using cv2.imread from the specified file path .The loaded image is then converted from BGR to RGB color space using cv2.cvtColor.

2. **Vectorization of Image**:

   The image is vectorized by reshaping it into a one-dimensional array, where each element represents a pixel in RGB space.

3. **K-Means Clustering**:

   K-Means clustering is applied to the vectorized data using cv2.kmeans.The number of clusters (K) is set to 3 initially, and the algorithm is run for a maximum of 10 attempts.

4. **Convergence Criteria**:

The convergence criteria for K-Means is defined using cv2.TERM_CRITERIA_EPS and cv2.TERM_CRITERIA_MAX_ITER.This ensures that the algorithm stops when either the specified number of iterations (10) is reached or the specified epsilon (1.0) is achieved.

5. **Cluster Centers and Labels**:

The resulting cluster centers and labels are obtained after K-Means clustering.

6. **Assigning Colors to Pixels**:

The colors of the original image are replaced with the colors of the cluster centers based on the labels assigned by K-Means.

7. **Reshaping Result to Image Format**:

The result is reshaped to the original image format for further processing.

Results

Contour Accuracy:

In the first task involving contour detection, the obtained contours exhibited some limitations in terms of accuracy. The contours were influenced by factors such as image noise, resolution, and the choice of contour approximation methods. Efforts were made to enhance accuracy by adjusting parameters within the cv2.findContours function, focusing on retrieval modes and approximation methods. Additionally, preprocessing steps, including smoothing and thresholding, were explored to improve the quality of detected edges.

Canny Edge Detection:

Canny edge detection, while a powerful method, faced challenges in capturing all relevant edges within the image. Experimentation with various threshold values was conducted to fine-tune the edge detection process. In situations where images presented noise or variations in lighting, Gaussian blur was applied prior to Canny edge detection. However, it was observed that achieving optimal results remained a delicate balance between capturing weak edges and excluding noise.

K-Means Segmentation:

K-Means clustering was employed for image segmentation, aiming to group similar regions together. The choice of the number of clusters (K) was pivotal in achieving effective segmentation. Initial attempts with a modest number of clusters (e.g., 3) provided a baseline understanding. However, notable improvements were observed as the number of clusters was increased. An optimal segmentation outcome was achieved when the number of clusters was set to 7, showcasing the algorithm's ability to delineate diverse regions within the image.