Computer Vision
Assignment 10

Alireza Moradi

December 1, 2020

1 Image Segmentation

Some of the practical applications of image segmentation are:

- Content-based image retrieval
- Machine vision
- Medical imaging, including volume rendered images from computed tomography and magnetic resonance imaging.
  - Locate tumors and other pathologies
  - Measure tissue volumes
  - Diagnosis, study of anatomical structure
  - Surgery planning
  - Virtual surgery simulation
  - Intra-surgery navigation
- Traffic control systems
  - Video surveillance
  - Video object co-segmentation and action localization

Content-based image retrieval, also known as query by image content (QBIC) and content-based visual information retrieval (CBVIR), is the application of computer vision techniques to the image retrieval problem, that is, the problem of searching for digital images in large databases. Content-based image retrieval is opposed to traditional concept-based approaches.

"Content-based" means that the search analyzes the contents of the image rather than the metadata such as keywords, tags, or descriptions associated with the image. The term "content" in this context might refer to colors, shapes, textures, or any other information that can be derived from the image itself. CBIR is desirable because searches that rely purely on metadata are dependent on annotation quality and completeness.
Source

2 Erode & Dilate

I used `Border Reflect` to achieve these results.

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	1
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0
0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0

Table 1: Dilate

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
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

Table 2: Erode

3 Implementations

3.1 Detect Shape Color

`or` First I smoothed the image and applied a threshold to get a binary image. `cv2.findContours` returns the set of outlines and then I used `imutil` to get the required tuples of each contour. Then I iterate over contours and compute image moments for the contour region.

In computer vision and image processing, image moments are often used to characterize the shape of an object in an image. These moments capture basic statistical properties of the shape, including the area of the object, the centroid (i.e., the center (x, y)-coordinates of the object), orientation, along with other desirable properties.

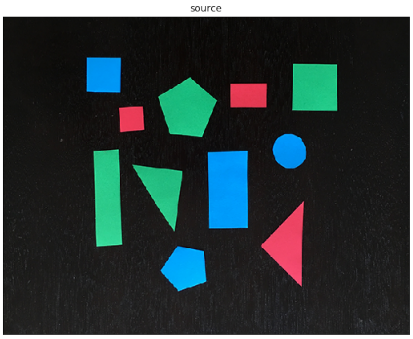
In order to perform shape detection, I used contour approximation. As the name suggests, contour approximation is an algorithm for reducing the number of points in a curve with a reduced set of points.

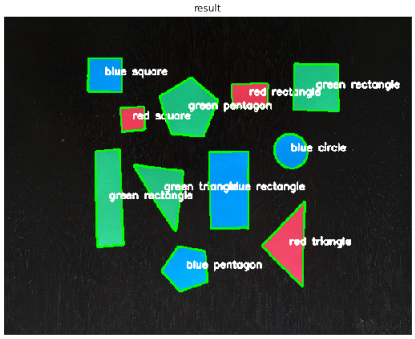
In order to perform contour approximation, I first computed the perimeter of the contour, followed by constructing the actual contour approximation.

A contour consists of a list of vertices. I checked the number of entries in this list to determine the shape of an object.

In order to actually label and tag regions of an image as containing a certain color, I computed the Euclidean distance between a dataset of known colors (i.e., the lab array) and the averages of a particular image region. Unlike HSV and RGB color spaces, the Euclidean distance between L*a*b* colors has actual perceptual meaning

The known color that minimizes the Euclidean distance will be chosen as the color identification.






```
import imutils
from scipy.spatial import distance as dist
from collections import OrderedDict

class ShapeDetector:
    def __init__(self):
        pass
    def detect(self, c):
        shape = "unidentified"
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.04 * peri, True)
        if len(approx) == 3:
            shape = "triangle"
        elif len(approx) == 4:
```

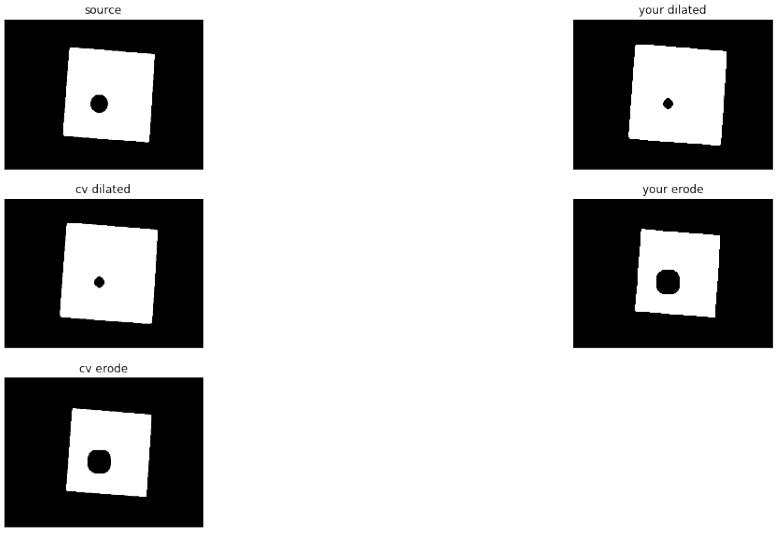
```
            (x, y, w, h) = cv2.boundingRect(approx)
            ar = w / float(h)
            shape = "square" if ar >= 0.95 and ar <= 1.05 else "rectangle"
        elif len(approx) == 5:
            shape = "pentagon"
        else:
            shape = "circle"
        return shape

class ColorLabeler:
    def __init__(self):
        colors = OrderedDict({
            "red": (255, 0, 0),
            "green": (0, 255, 0),
            "blue": (0, 0, 255)})

        self.lab = np.zeros((len(colors), 1, 3), dtype="uint8")
        self.colorNames = []

        for (i, (name, rgb)) in enumerate(colors.items()):

            self.lab[i] = rgb
            self.colorNames.append(name)

        self.lab = cv2.cvtColor(self.lab, cv2.COLOR_RGB2LAB)

    def label(self, image, c):
        mask = np.zeros(image.shape[:2], dtype="uint8")
        cv2.drawContours(mask, [c], -1, 255, -1)
        mask = cv2.erode(mask, None, iterations=2)
        mean = cv2.mean(image, mask=mask)[:3]

        minDist = (np.inf, None)

        for (i, row) in enumerate(self.lab):

            d = dist.euclidean(row[0], mean)

            if d < minDist[0]:
                minDist = (d, i)

        return self.colorNames[minDist[1]]
```

```
def detect_shape_color(image):
    '''
    Detects shapes and their color in the input image.

```

```
    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The result image.
    '''

    result = image.copy()

    #Write your code here
    blurred = cv2.GaussianBlur(image, (5, 5), 0)
    gray = cv2.cvtColor(blurred, cv2.COLOR_BGR2GRAY)
    lab = cv2.cvtColor(blurred, cv2.COLOR_BGR2LAB)
    thresh = cv2.threshold(gray, 60, 255, cv2.THRESH_BINARY)[1]

    centers = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    centers = imutils.grab_contours(centers)
    sd = ShapeDetector()
    cl = ColorLabeler()
    for c in centers[1:]:
        M = cv2.moments(c)

        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])

        color = cl.label(lab, c)
        shape = sd.detect(c)

        cv2.drawContours(image, [c], -1, (0, 255, 0), 2)
        cv2.putText(image, f"{color} {shape}", (cX, cY), cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, (255, 255, 255), 2)

    return image
```

3.2 Erode & Dilate

To dilate, I convolved the structural element with the image and used a `logical and` to find the final result.




```python
def your_dilate(image, structuring_element):
    '''
    Applies your dilation.

    Parameters:
        image (numpy.ndarray): The input image.
        structuring_element (numpy.ndarray): The structuring element must be square.

    Returns:
        numpy.ndarray: The result image.
    '''

    result = image.copy()

    res = cv2.filter2D(result, -1, structuring_element)
    result = np.logical_and(image,res)

    return res
```

To erode, because there should be a subset of structural element in that slice of the image, the covolution of image with element should result in a pixel with the value of at least the total number of ones in the element.

```python
def your_erode(image, structuring_element):
    '''
    Applies your erosion.

```

```
    Parameters:
        image (numpy.ndarray): The input image.
        structuring_element (numpy.ndarray): The structuring element must be square.

    Returns:
        numpy.ndarray: The result image.
    '''

    result = image.copy()
    s = np.sum(structuring_element)

    res = cv2.filter2D(np.array(result,dtype=np.float32)/255, -1,
                    structuring_element,borderType=cv2.BORDER_CONSTANT)
    result[np.where(res >= s)] = 255
    result[np.where(res < s)] = 0
    return result
```

```
def cv_dilate(image, structuring_element):
    '''
    Applies OpenCV dilation.

    Parameters:
        image (numpy.ndarray): The input image.
        structuring_element (numpy.ndarray): The structuring element must be square.

    Returns:
        numpy.ndarray: The result image.
    '''

    result = image.copy()

    result = cv2.dilate(image,structuring_element)

    return result
```

```
def cv_erode(image, structuring_element):
    '''
    Applys OpenCV erosion.

    Parameters:
        image (numpy.ndarray): The input image.
        structuring_element (numpy.ndarray): The structuring element must be square.

    Returns:
        numpy.ndarray: The result image.
    '''
```

```
12
13      result = image.copy()
14
15      result = cv2.erode(image,structuring_element)
16
17      return result
```