Gall Bladder Segmentation from Ultrasound Image

Oct-Nov 2020

Assignment1(COL780)

By-

Rohan Yuttham

2017ME10605

(me1170605@iitd.ac.in)

Contents

1.Introduction

Given a set of ultrasound image data, this report shall focus on image processing techniques to extract a gall bladder from the input image. Note that no machine learning techniques shall be implemented to obtain the result. We shall only be using OpenCV functions to segment the gall bladder.

The output should be a binary mask, i.e., a grayscale image with 0 where gall bladder is not present as shown below in Figure 1.

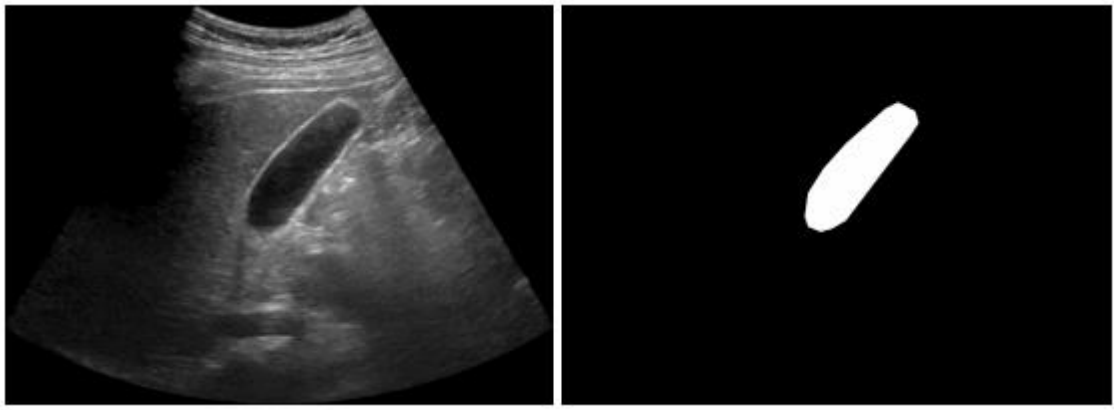

Figure 1: A sample input and output

The report shall discuss on the methods tried and used to accomplish the required task. Finally the obtained masks are compared with a test set of 10 images.

2.Methods Used

Image Pre-Processing:

Before we proceed, we want our input image to be in a manner such that its features are distinguished and can be easily extracted using OpenCV tools.

1.Smoothing Images

We will apply two filters on the initial image. This is an application of 2D convolution using a kernel.

We begin by using medianBlur and is followed by GaussianBlur. Median filtering computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This method helps in removing the *Salt and Pepper noise* from image.

In Gaussian filtering, instead of a box filter consisting of equal filter coeeficients, a Gaussian kernel is used. The height and width of kernel is specified. It helps in removing the Gaussian noise from image

```
img = cv2.imread(img_path, 0)
blur = cv2.medianBlur(img,5)
blur = cv2.GaussianBlur(img,(5,5),0)
```

Figure 2 image after applying median blur

2. Histograms

As part of the pre-processing, I had started by using only cv2.equalizeHist but after applying further processing, it turned out that the gall bladder was not completely distinguishable from the background. Some portions were merging in background. Then, this was replaced by CLAHE (Contrast Limited Adaptive Histogram Equalization). Although this resulted in making the earlier non distinguishable images to work but the images which had turned out to be good using Global Histogram Equalization turned out to be giving results which were not clear.

Finally, both of above *Histogram Equalization* methods are implemented in the image.

Basically, first histogram equalization we just saw, considers the global contrast of the image. In many cases, it is not a good idea. Due to over brightness many features are lost. So, to solve this problem, *adaptive histogram equalization* is used. In this method, image is divided into small blocks and each of these blocks are then equalized as usual. This also uses *contrast limiting* to avoid any amplification of noise.

```
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8)) #first CLAHE is done
    hist_eq = clahe.apply(blur)
    hist_eq = cv2.equalizeHist(hist_eq) #Global Histogram Equalization
```

Figure 3 After applying Hist Equalization

Observe that although the original look of the image seems to go away, but the feature we want to extract i.e., gall bladder is now distinguishable from other parts of image.

Feature Segmentation

Now, we are ready to extract this feature from pre-processed image.

1.Image Thresholding

In this assignment *Otsu's Binarization* is used. In this method, a threshold value is automatically calculated from image histogram. If a pixel value is greater than this threshold value, it is assigned one value(white), else it is assigned 0(black).

```
ret,th = cv2.threshold(hist_eq,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

Figure 4 Image after Thresholding

2. Morphological Transformation

A morphological transform called *Opening* is applied in the binarized image. This is a result of two other morphological transforms applied one after the other namely *Dilation* followed by *Erosion*. Dilation increases the foreground object and erosion erodes away the boundaries of foreground object. This is followed by inverting the image using *bitwise_not.*

```
kernel = np.ones((3,3),np.uint8)
closing = cv2.morphologyEx(th,cv2.MORPH_CLOSE,kernel, iterations = 2)
closing = cv2.bitwise_not(closing) # Inverting the obtained image
```

Figure 5 After application of Closing

3. Floodfill

This OpenCV function simply fills the image with one color till an edge is found. To be sure that entire boundary areas are covered, floodfill is applied on all four corners.

```
    height, width = img.shape
    cv2.floodFill(closing,None,(0,0),0)
    cv2.floodFill(closing,None,(0,height-1),0)
    cv2.floodFill(closing,None,(width-1,height-1),0)
    cv2.floodFill(closing,None,(width-1,0),0)
```

Figure 6 After applying floodfill on corners

4. Removal of white noise

We will see the application of *Connected Components* in this section. The idea is to get the information of all the connected components, which in this case are the white spaces. And, finally the white space with largest area is retained and everything else is discarded as white noise. See the following piece of code for better understanding.

```
    nlabels, labels, stats, centroids = cv2.connectedComponentsWithStats(binary_map, None,
None, None, 8, cv2.CV_32S)
    areas1 = np.asarray(stats[1:,cv2.CC_STAT_AREA])# Areas of each white space is obtained

    final_result = np.zeros((labels.shape), np.uint8) # A mask is created
    for i in range(0, nlabels - 1):
        if areas1[i] ==np.max(areas1):   #keep only the largest area
            final_result[labels == i + 1] = 255
```

Figure 7 Final segmented image

3.Conclusion

Finally, the gallbladder is extracted from the ultrasound image using tools of image processing. However, the obtained segment is not perfectly as the shape of gall bladder. Using a test set of 10 images, an accuracy of 78% is obtained.