Introduction to image processing and computer vision

Project I

Nails Segmentation and Labelling

Supervisor: Rafal Jozwiak

Author

M Hasibur Rahman

295459

December 29, 2020

Contents

I Introduction

I.I Nail Segmentation

Several methods have been used in image segmentation to detect nails which particularly plays a big role in medicine. The surface growth status of nails can reflect the physiological condition of human body. Diseases can be predicted and prevented by by particularly looking at the lunula and nail plate parts of the nail. The proportion of lunula to the nail plate reflects the human health status. Hence, setting these parts of nails as reference frames and finding a image processing method to segment such parts for the betterment of humans shows the importance of image processing in general.

Even though this paper does not deal with segmentation of particular parts of the nail like lunula, it proposes an image processing method to segment nails from given images despite the noise of the image, background, light fluctuation or color of skin. To be able to identify the nail would lay the groundwork to further segmentations such as, specific parts of nail segmentation as mentioned before and its importance.

We have been given a set of 52 different images which are to evaluated. The goal is to output the given images such that, the nails are only visible seperated from the background, binary mask and assessment according to intersection over Jaccard index and Dice coefficient. Furthermore a mean results for whole data set and per individual image is to be done.

II Algorithm Description

II.I Binary segmentation

A mask is created for each image. The first step of creating the mask is try to removal of the background and nearby elements that are not are not in particular range of shades of nail colour with the use of hsv of color palette. The range used is general nail color range.

Then, the mask is cleared up a bit from the background noise which may appear, since the color range on which the mask was created includes gray and white. After that, a median blur is used to fill holes in masks and to clean the background from noise [1].

Code Snippet 1:

```
import cv2 as cv
import numpy as np
# from matplotlib import pyplot as plt

# using a sample image from data set
img = cv.imread('images/4c48acb6-e402-11e8-97db-0242ac1c0002.jpg')
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)

# minRange for min skin color Rnage
# maxRange for maximum skin color Range
lower = np.array([10, 10, 60], dtype = "uint8")
upper = np.array([20, 150, 255], dtype = "uint8")

mask = cv.inRange(hsv, lower,upper)

kernel = np.ones((3, 3), np.uint8)
#blur , optionally use opening
mask = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
mask = cv.medianBlur(mask, 5)


cv.imshow("mask", mask)
cv.imshow("originalimg", img)
cv.waitKey(0)
```

Since the nail color range is similar to the skin color range we also get the mask of the hands as-well.

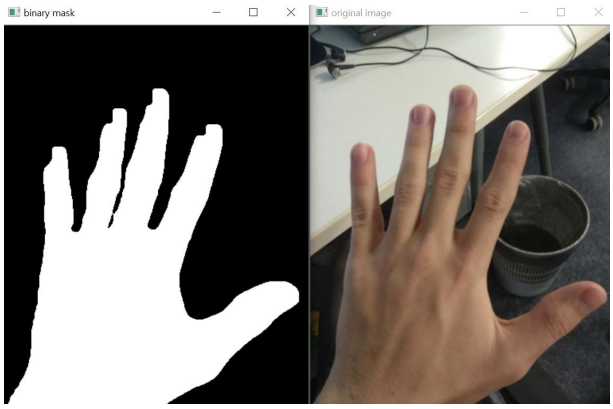

Figure 1: Result of binary segmentation on specific range

Also, since in the data set we have painted nails with different colors, setting a specific range for masking will not be the best way to go about this. Hence we need to find another way to segment the image.

II.II Canny edge detection

With many failed attempts using opencv functions such as edge detection, Canny(), contours etc. and similar methods. I resorted to use sickit-image [2] which provides a collection of algorithms for image processing. I used their canny edge detector function and its functionality is described below:

Theorem 1. Function: canny()

The Canny filter (sigma - as used in code snippet 2) is a multi-stage edge detector. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients.The Gaussian reduces the effect of noise present in the image. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude.

The Canny has three adjustable parameters: the width of the Gaussian (the noisier the image, the greater the width), and the low and high threshold for the hysteresis thresholding. [3]

I used binary_fill_holes() after the canny() to fill some edges which perfectly form a shape without any gaps or issues. After using the canny function I converted the implemented skimage to opencv (using function img_as_ubyte()) to show image. I was not able to get rid of the unwanted edges and their binary_fill_holes(), thats why, the results below include the edges detected and the masks. That is, we get a non filtered partially segmented results.

Code Snippet 2:

```
import cv2 as cv
import numpy as np
from skimage import io
from skimage.util import img_as_ubyte
from skimage.feature import canny
from scipy import ndimage as ndi
from skimage import feature
from matplotlib import pyplot as plt

#taking a sample image from data set
img = cv.imread('images/1eecab90-1a92-43a7-b952-0204384e1fae.jpg')
segmented = cv.imread('labels/1eecab90-1a92-43a7-b952-0204384e1fae.jpg')
imgray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
#using canny() from skimage with sigma value = 1
edges = feature.canny(imgray, sigma=1)
fill = ndi.binary_fill_holes(edges)
#convert image from skimage to opencv
masked = img_as_ubyte(fill)

cv.imshow("cv", masked)
cv.waitKey(0)
```

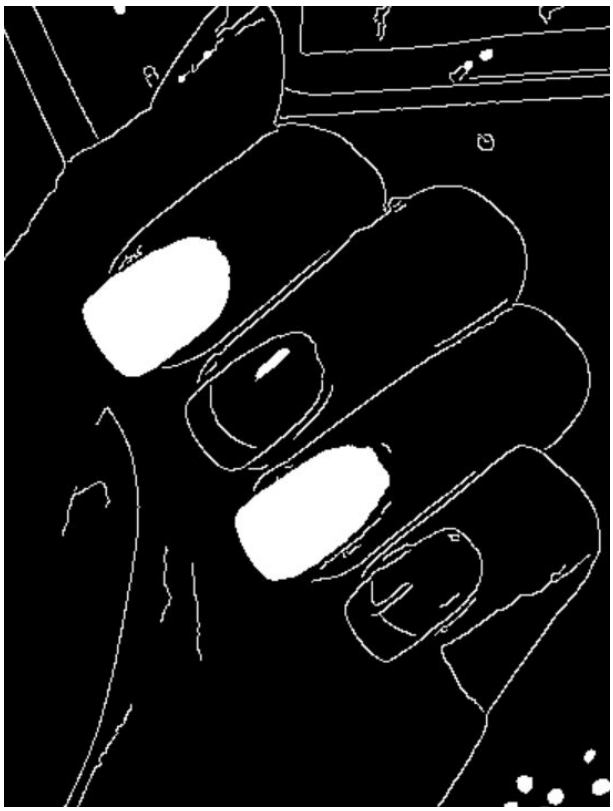

Figure 2: Result of above code snippet

III Results

The code used to get accuracy is shown below where Jaccard index was calculated for each image.
Code snippiet 3:

```
import cv2 as cv
import numpy as np
from skimage import io
from skimage.util import img_as_ubyte
from skimage.feature import canny
from scipy import ndimage as ndi
from skimage import feature
from matplotlib import pyplot as plt

#image to evaluated and set one by one
img = cv.imread('images/da236f3a-8c82-4c64-9a7d-9b950fd8b47e.jpg')
imgray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

#the image given by teacher for the prediction and will be used for
    jaccard formula
segimg = cv.imread('labels/da236f3a-8c82-4c64-9a7d-9b950fd8b47e.jpg')
#convert to BGR2GRAY so that we can compare the masked and predicted result
seggray = cv.cvtColor(segimg, cv.COLOR_BGR2GRAY)

#using canny() from skimage with sigma value = 1
edges = feature.canny(imgray, sigma=1)
fill = ndi.binary_fill_holes(edges)
#convert image from skimage to opencv
masked = img_as_ubyte(fill)



#jaccard index (iou) - formula gotten from project description
iou = np.sum(np.logical_and(masked, seggray)) /
    np.sum(np.logical_or(masked, seggray))
print(iou*100)
#plt.imshow(fill)
#plt.show()

cv.imshow("ourresult", masked)
cv.imshow("prediction",seggray)
cv.waitKey(0)
```

After running the above code for all the image individually this is the jaccard index value I get for the 52 images in data set:
{ 40.97, 19.65, 0.475, 1.88, 0.986, 0.61, 1.38, 1.34, 0.463, 1.28, 1.45, 2.62, 12.85,
0.9, 2.77, 12.35, 4.40, 2.87, 2.28, 42.47, 8.02, 1.83, 1.30, 18.8, 10.8, 26.9, 2.76,
6.06, 1.05, 8.32, 2.84, 5.12, 27.7, 2.84, 2.63, 8.7, 11.45, 1.72, 6.40, 2.29, 3.13,
13, 4.34, 7.9, 1.31, 7, 12, 3.42, 2.84, 3.46, 48.42, 0.17 }

The mean result of the whole data set for jaccard is about **8.04%** if no mistakes were done while collecting result for each image individually.

III.I Good results (somewhat), where iou $> 20\%$

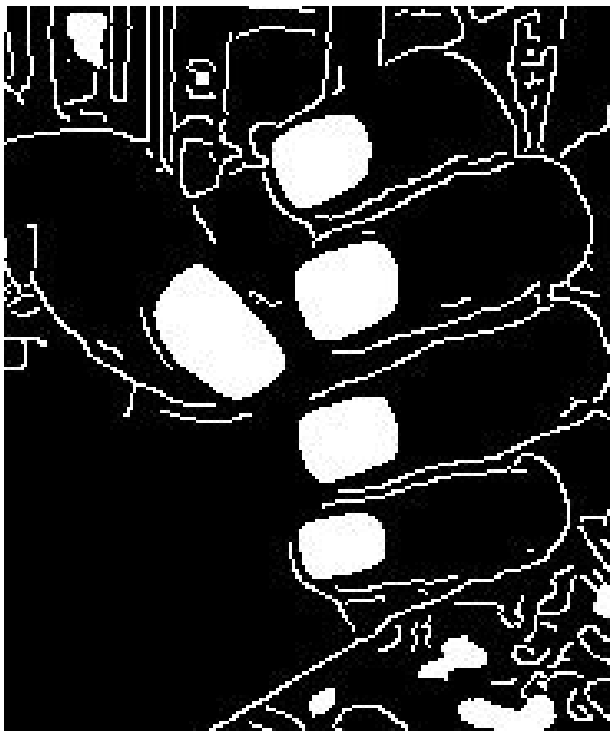

Figure 3: iou is about 42.47

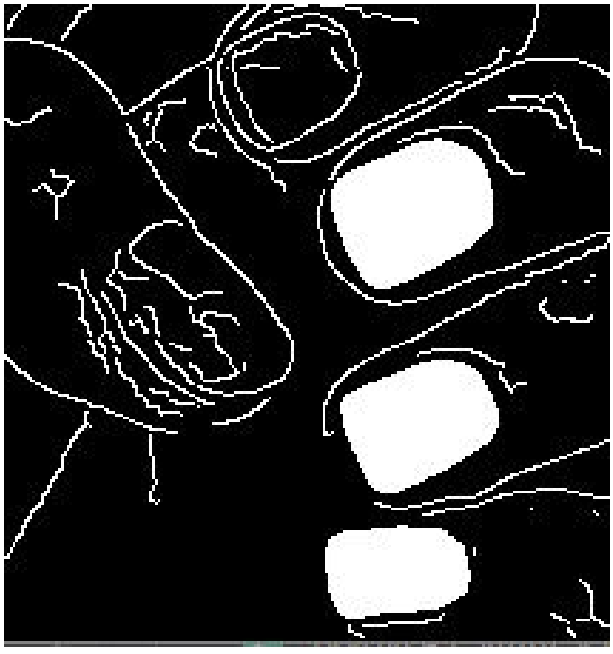

Figure 4: iou is about 48.42

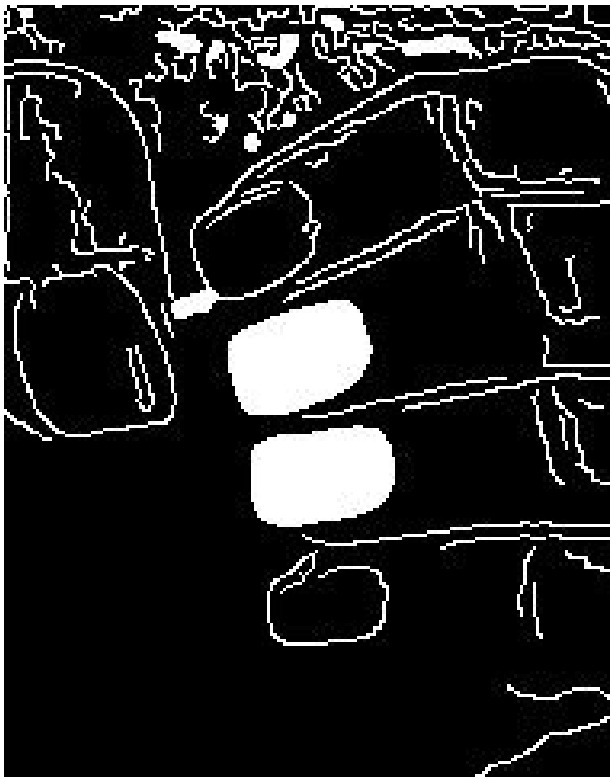

Figure 5: iou is about 26.9

III.II Bad results

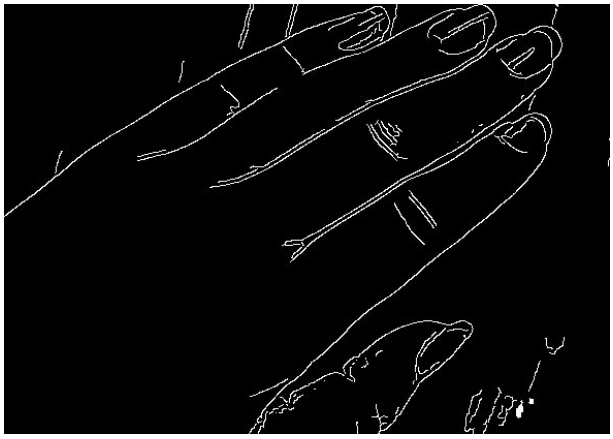

Figure 6: iou is about 1.31

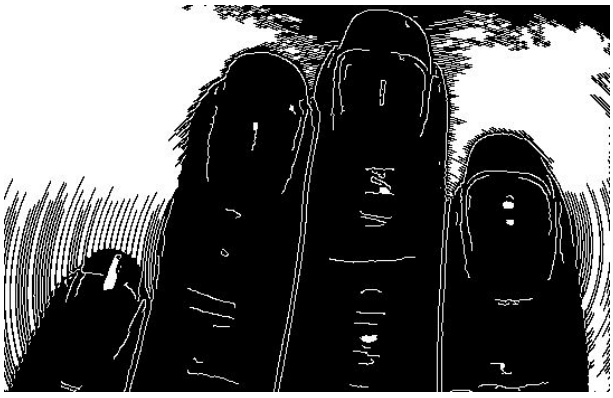

Figure 7: iou is about 7.00

Figure 8: iou is about 0.9

For more results you can see the folder - resultperimage that was attached during submissions.

IV Conclusion and Comments

Despite some usage of opencv and scikit-image I did not manage to still get a good accuracy rate. The average jaccard index i.e. iou value I got is approximately **8.04%**. The cause of this bad result is maybe because I was solely reliant on the sigma value of the canny() function. And so probably I was not able to find the perfect sigma value due to the different images in data sets giving different amount of edges and thus affecting the hysteresis threshold. Also, because I was not able to remove the edges. A somewhat fair iou values was obtained usually if the image in data set focused mostly on nails and had less noises in the background.

Note: I apologize again for using scikit-images (if it is not allowed) but I really wanted to see some result and try the task and not have an empty report.

V Source code

```
#The following code was written by M Hasibur Rahman, 295459
import cv2 as cv
import numpy as np
from skimage import io
from skimage.util import img_as_ubyte
from skimage.feature import canny
from scipy import ndimage as ndi
from skimage import feature
from matplotlib import pyplot as plt

#part for showing binary segmentation and using ranges
# using a sample image from data set
img = cv.imread('images/4c48acb6-e402-11e8-97db-0242ac1c0002.jpg')
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)

# minRange for min skin color Rnage
# maxRange for maximum skin color Range
lower = np.array([10, 10, 60], dtype = "uint8")
upper = np.array([20, 150, 255], dtype = "uint8")

mask = cv.inRange(hsv, lower,upper)

kernel = np.ones((3, 3), np.uint8)
#blur , optionally use opening
mask = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
mask = cv.medianBlur(mask, 5)


cv.imshow("mask", mask)
cv.imshow("originalimg", img)

#-------------------------------------------------------------
#part for canny edge detection and calculating iou values
#image to be evaluated and set one by one
img = cv.imread('images/1eecab90-1a92-43a7-b952-0204384e1fae.jpg')
imgray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

```
#the corresponding image given by teacher for the prediction in labels
    folder and will be used for jaccard formula
segimg = cv.imread('labels/1eecab90-1a92-43a7-b952-0204384e1fae.jpg')
#convert to hsv so that we can compare the masked and predicted result
seggray = cv.cvtColor(segimg, cv.COLOR_BGR2GRAY)

#using canny() from skimage with sigma value = 1
edges = feature.canny(imgray, sigma=1)
fill = ndi.binary_fill_holes(edges)
#convert image from skimage to opencv
masked = img_as_ubyte(fill)



#jaccard index (iou) - formula gotten from project description
iou = np.sum(np.logical_and(masked, seggray)) /
    np.sum(np.logical_or(masked, seggray))
print(iou*100)
#plt.imshow(fill)
#plt.show()

cv.imshow("ourresult", masked)
cv.imshow("prediction",seggray)
cv.waitKey(0)
```

Bibliography

[1] *opencv basic thresholding*. 2020. URL: https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html (page 3).
[2] *sci-kit image*. 2020. URL: https://scikit-image.org/ (page 4).
[3] *sci-kit image canny edge detection*. 2020. URL: https://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html#sphx-glr-auto-examples-edges-plot-canny-py (page 4).