NAIL SEGMENTATION OF HAND IMAGES

Mateo Lara

Introduction to Image Processing and Computer Vision

Rafał Józwiak

Warsaw University of Technology

Warsaw

12-29-2020

Table of Contents

I.Problem Definition

For a data set consisting of 52 hand images, we are looking to segment the images extracting the nails and getting rid of any background and the hand itself. Image segmentation is used to locate objects and boundaries; for this project, we are using classical segmentation techniques for automatic segmentation, so the segmentation does not require human intervention. Among classic methods that can be applied are Pixel-based segmentation, edge-based segmentation, region-based methods, and model-based segmentation. Several assumptions are made base on the given dataset:

- The image contains a hand and nails
- The hand is the main element of the picture
- The color of the skin is within a certain range
- The nail and the background have a somewhat different color than the hand

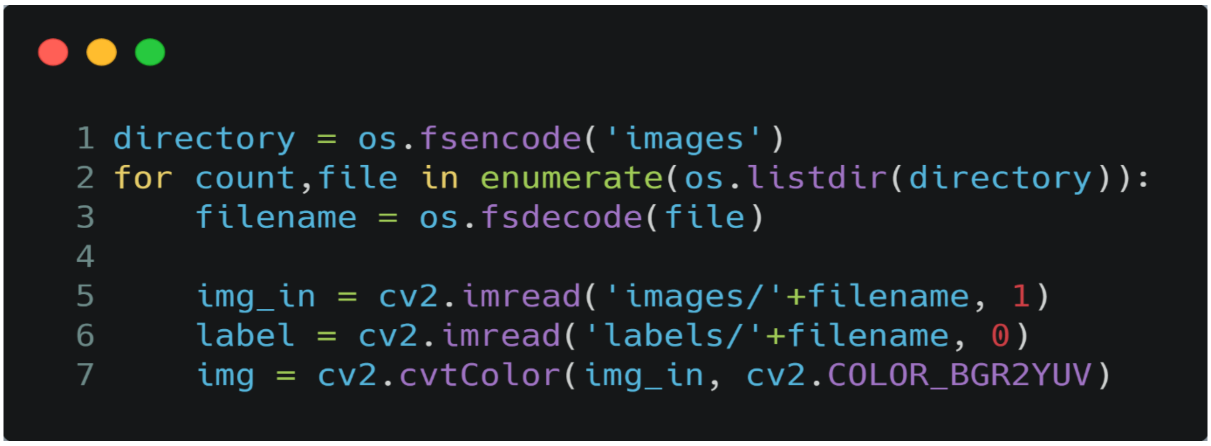
The result of the segmentation should be a binary image, in which the nails are highlighted with white color and the rest of the image with black color. The dataset contains labels with the images already segmented, so to measure how well the segmentation algorithms works, it is necessary to perform a comparison between the given labels and the results, using assessment accordingly to intersection over union metric and Dice coefficient.

II.Literature review

As previously discuss we will be using Automatic setting for the segmentation of nails, accordingly to Russ [1] all automatic methods make some assumption about the nature of the image, and to choose the proper algorithm it is important to know as mush as possible about the nature of the image. Most of the classic automatic methods use the histogram information for the segmentation algorithm but some also use the location information for the pixels. One common technique for segmentation is thresholding, this method selects certain areas of the image base on a criterion and reject the ones that does not satisfy these criteria, creating a binary image. For a one channel bimodal image the problem became trivial since we know that the region of interest is one of two peaks in the histogram, if we know a little about the nature of the image, we can select the correct peak to make the segmentation. The problem became more interesting when we have a multiband image, in that case it is common to make some color space transformation so that we can better represent the changes and discard the dimensions that does not give meaningful information. Regardless of the kind of image we have, the thresholding method takes or try to find a principal component, something that distinguish the element to segment, and base on that generates a binary image. Now since pixel-base images represents an approximation of the continuous real scene being represented, and since thresholding classifies each pixel as either part of the foreground or the background, only a finite level of accuracy can be achieved [1]. That is why some edge detection and region boundaries can be useful to segmentate the image, in this sense knowing the shape of the object can be of use and we could separate the object of interest finding its shape after applying boundary lines or contour lines. Other classical segmentation methods that can be apply together with the already discuss methods are segmentation by region growing and by region splitting and merging, region growing is a procedure that groups pixels or subregions into larger regions based on predefined criteria for growth. Region splitting and merging subdivide an image initially into a set of disjoint regions and then merge and/or split the regions to satisfy the conditions of segmentation [2]. Another possibility is to use clustering algorithms like K-means or mean shift to find regions with some degree of similarity then some additional information is necessary to select the cluster(s) that represents the segment of interest.

III.Description of solution

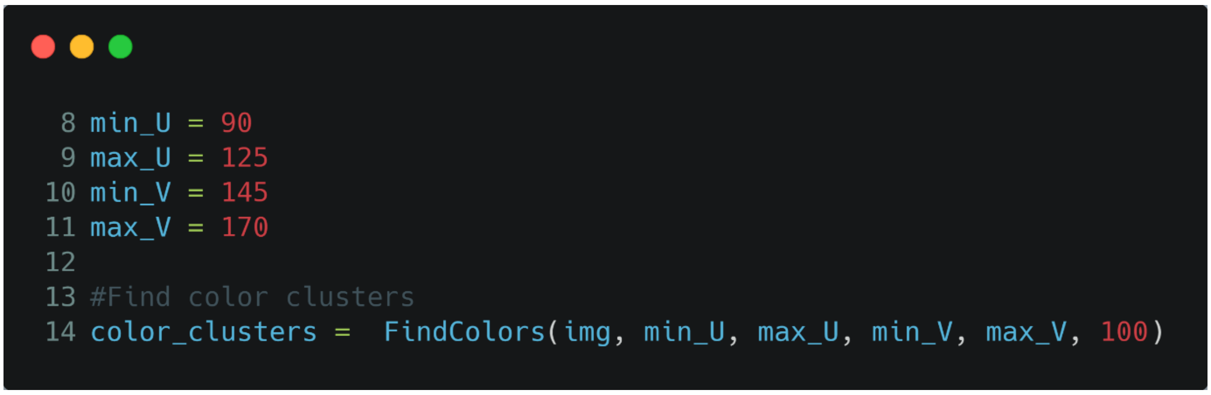
First, we have our main for-loop that allow us to take one by one the images and labels in our dataset, notice that in line 7 [Figure 1] we make a space color transformation from BGR to YUV, the reason for the transformation is going to be evident later.



```
1 directory = os.fsencode('images')
2 for count,file in enumerate(os.listdir(directory)):
3     filename = os.fsdecode(file)
4 
5     img_in = cv2.imread('images/'+filename, 1)
6     label = cv2.imread('labels/'+filename, 0)
7     img = cv2.cvtColor(img_in, cv2.COLOR_BGR2YUV)
```

Figure 1

Now we stablish a minimum and maximum value for the blue projection U and the red projection V of our transform YUV image, see line 8 to 11 [Figure 2], we intentionally exclude the luma component Y since we are more interested in the chrominance components of the image rather than the brightness, that is because we do not make any assumption about the lighting conditions of the image, but we will assume that the skin content of the image has certain range. The ranges were stablishing base on [3] and by looking at the dataset and realizing that there are no dark tones of skin present. After defining the color range, the function “FindColors” is called taking as arguments the image, the color ranges, and a kernel size defining the morphological closing inside the function, see line 14 [Figure 2]. The function was created so that, given a YUV image and a maximum and minimum value for the two chrominance components U and V it extracts the mask that represent the pixels in the image satisfying the two chrominance components boundaries. In consequence the function returns a black and white mask representing the regions satisfying the two chrominance components boundaries.



```
min_U = 90
max_U = 125
min_V = 145
max_V = 170

#Find color clusters
color_clusters =  FindColors(img, min_U, max_U, min_V, max_V, 100)
```

Figure 2

An example of the operation can be observed in [Figure 3] where the image to the left is the original image and the image to the right is the binary mask representing the hand.

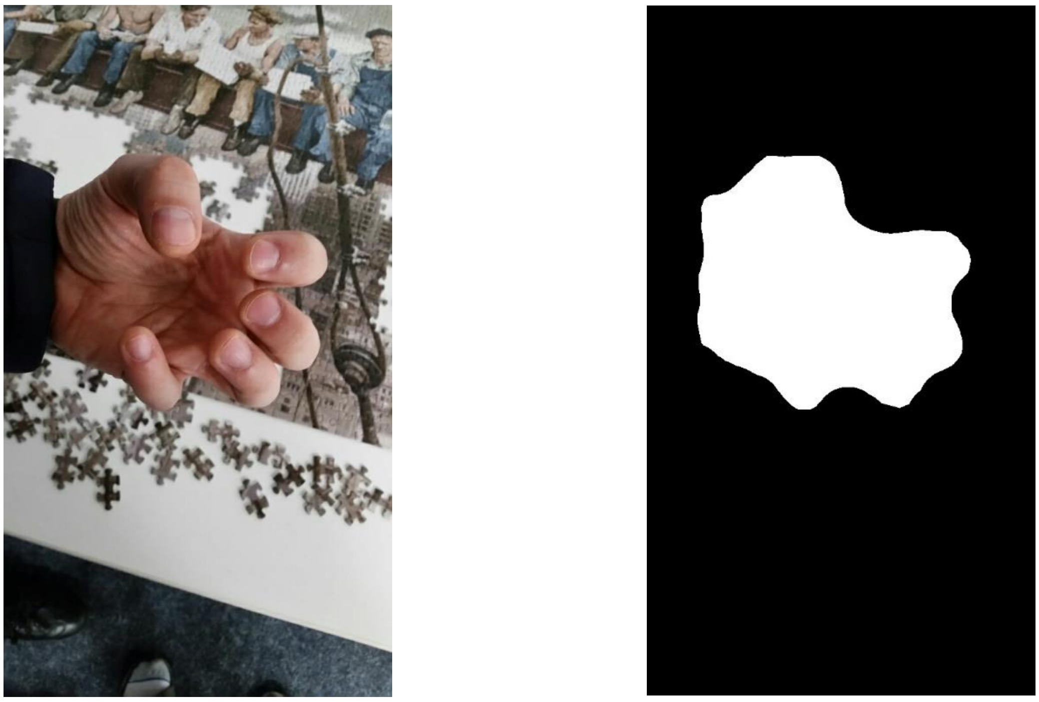


Figure 3

It is important to note that within the “FindColors” function, thresholding and binary operation techniques are applied so that we can select the image that is within the defined boundaries; there is also an area check operation and a morphological closure, as there are circumstances where the hand has been captured but the nails are missing, morphological closure can recover the presence of the nails in the image (see the source code for more information on the function created). There are obvious limitations to this technique. The first is that the skin color must be within a certain range defined by the bounds of the “FindColors” function. A clear example of the limitations of this technique can be seen in [Figure 4], where the hand in the left image appears bluish and some pixels in the background have some skin color, causing the binary mask to represent the unwanted background rather than the hand.

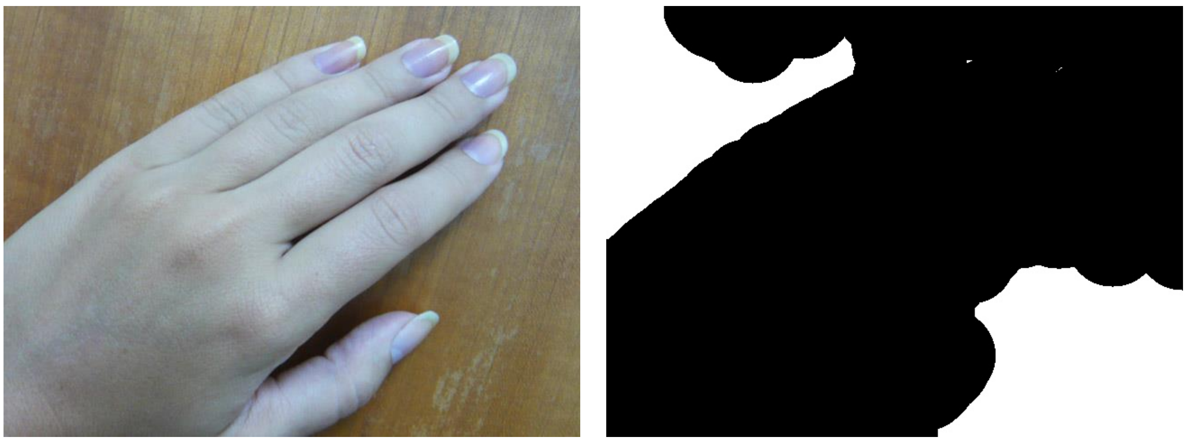


Figure 4

Fortunately, there are few cases where the hand is not within the range because the color range was chosen to match the data set. A second and more general problem is background noise, there are unwanted background pixels that may have a color that falls within the defined range, this situation worsens after the aggressive morphological operation applied to the binary image to restore the possibly missing nails. The previous example [Figure 4] is the worst-case scenario where the hand is not detected but the background is, a more realistic and common scenario is when the hand and some background is selected, see middle image [Figure 5].

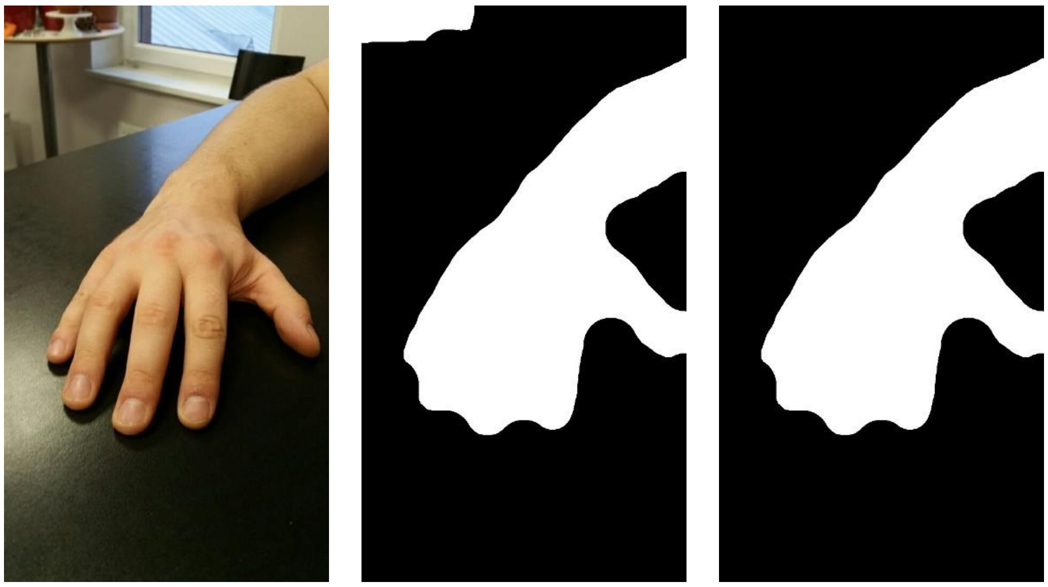
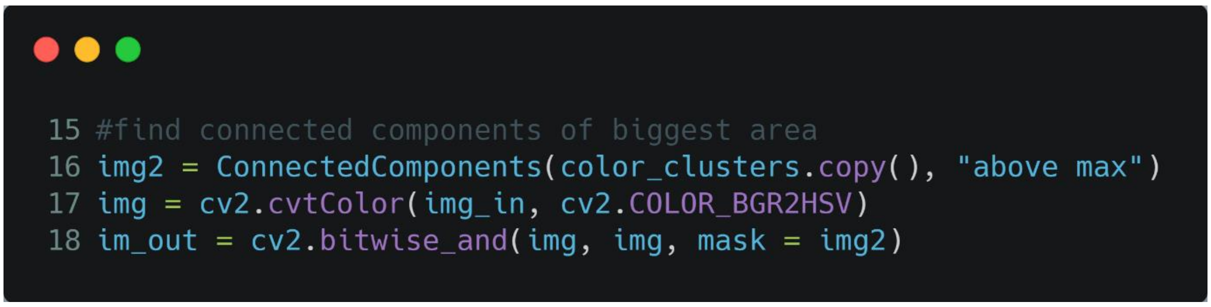


Figure 5

To solve this problem and get rid of the unwanted background, we create a function called “ConnectedComponents” that takes the binary mask generated by “FindColors” and a status string that defines which connected components should stay. This function returns a new mask indicating the connected components that match the given criteria, see line 16 [Figure 6]. Internally, this function groups the connected components and if the status "above max" is given, all connected components except the one with the largest area are removed (see source code for more information about the created function). The rightmost image in [Figure 5] shows an example where the unwanted background component was removed, leaving mainly the hand and the nails.

Although in most cases this technique can segment the hand, there are cases where the background cannot be distinguished from the hand, leaving a mask that takes up the entire image. There is also the possibility that there are two or more disconnected hands in the image. In this case, the algorithm takes only one hand (there is no such case in the dataset).



```python
#find connected components of biggest area
img2 = ConnectedComponents(color_clusters.copy(), "above max")
img = cv2.cvtColor(img_in, cv2.COLOR_BGR2HSV)
im_out = cv2.bitwise_and(img, img, mask = img2)
```

Figure 6

After obtaining a mask that mainly represents the hand and nails, we use an AND operator between the mask and the source image in the HSV color space, as a result we obtain a hand image, ideally with non-background, in the HSV color space [Figure 7].

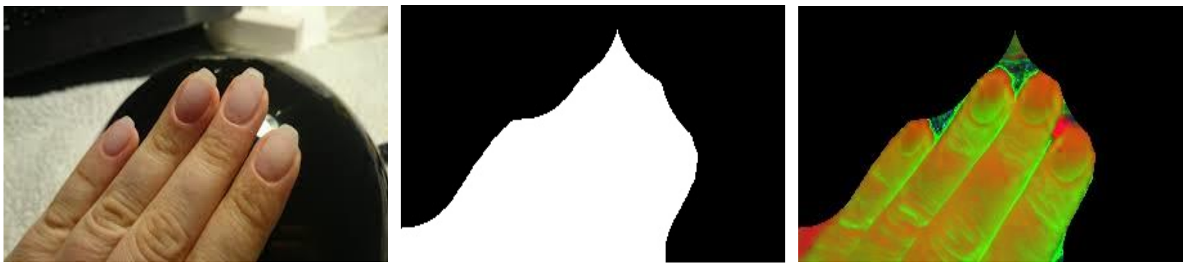
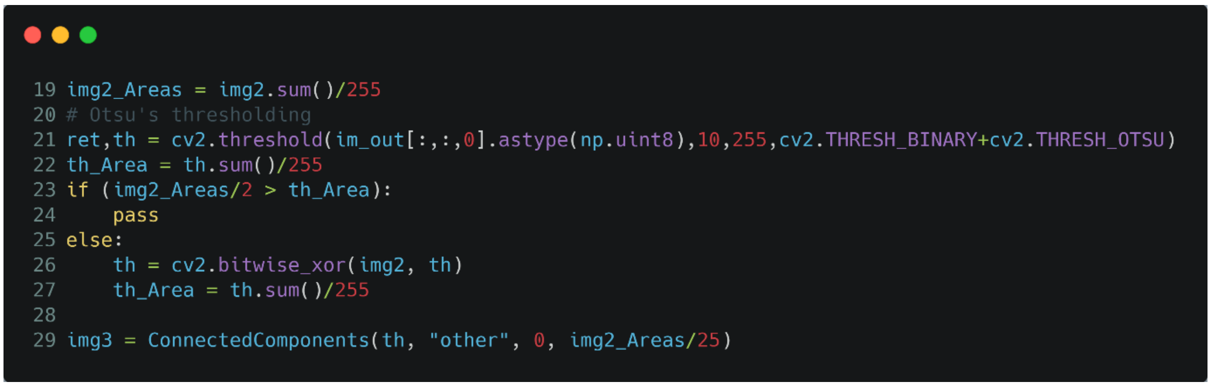


Figure 7

To complete the segmentation process and extract the nails from the hand, we use Otsu's binarization as an adaptive thresholding technique to determine an optimal global threshold from the image histogram [4]. We use the hue H channel of the HSV hand images and ignore the other two channels for the purpose of segmentation since we assume that the hand and the nails have different color, in this way the segmentation process is easily and quickly computed using one channel. In line 21 [Figure 8] we get the segmented component, but we are not sure if the component obtained is the correct one, we assume that one component is the hand, and the other is the nails. We also assume that the area of the nails is smaller than the area of the hand, so we do a validation to check if we got the correct component based on the area information of the hand and the segment element. If the area is too large to be the nails, we use an XOR operation to extract the alternative component. Finally, we use the created function "ConnectedComponents", which provides some boundary criteria for the size of the components, using some assumptions about the possible nail area based on the hand area.



```python
img2_Areas = img2.sum()/255
# Otsu's thresholding
ret,th = cv2.threshold(im_out[:,:,0].astype(np.uint8),10,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
th_Area = th.sum()/255
if (img2_Areas/2 > th_Area):
    pass
else:
    th = cv2.bitwise_xor(img2, th)
    th_Area = th.sum()/255

img3 = ConnectedComponents(th, "other", 0, img2_Areas/25)
```

Figure 8

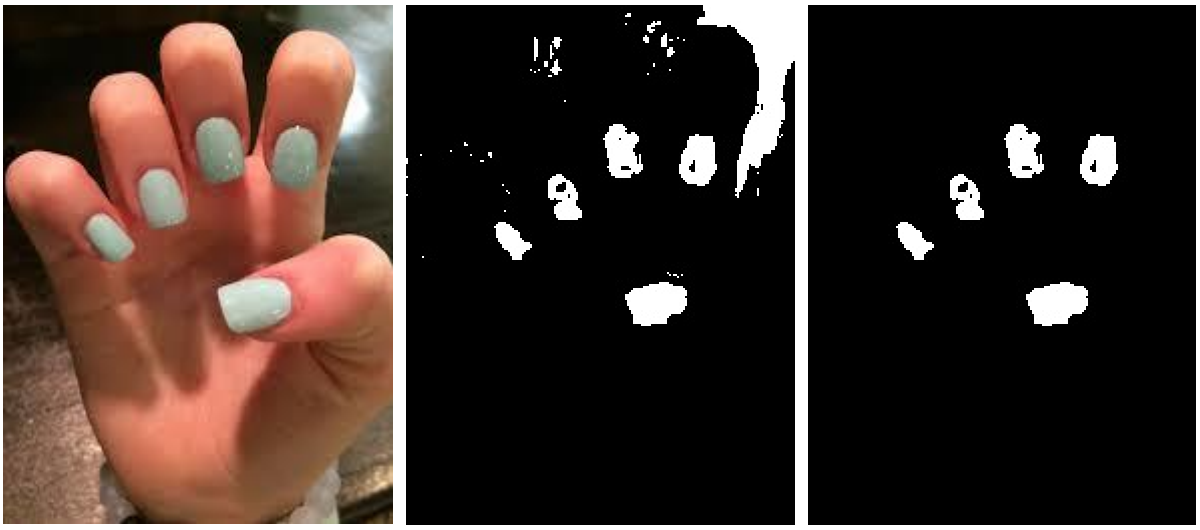


Figure 9

In the [Figure 9] we can see some results; the middle image contains the mask generated by thresholding, as expected it contains some background noise which we then reduce using our “ConnectedComponents” function; the result of this operation can be seen in the right part of the figure. It is important to mention that this way of segmenting the nails is far from perfect and there is a lot of room for improvement, especially in situations where the color of the nails is close to the color of the skin, or when the nails have different colors from each other, or when the previous stages of segmentation allow some background noise that can be mistaken for nails.

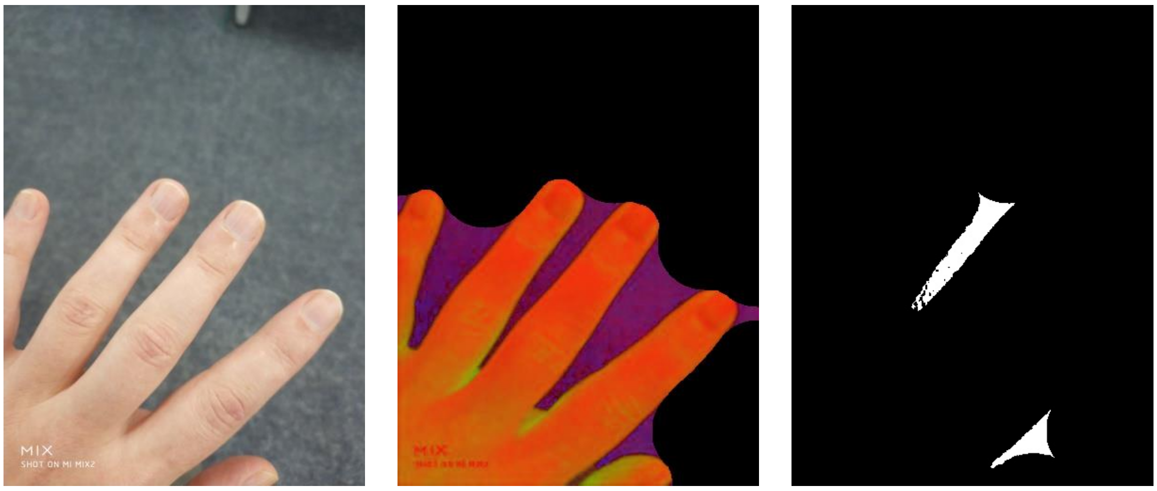
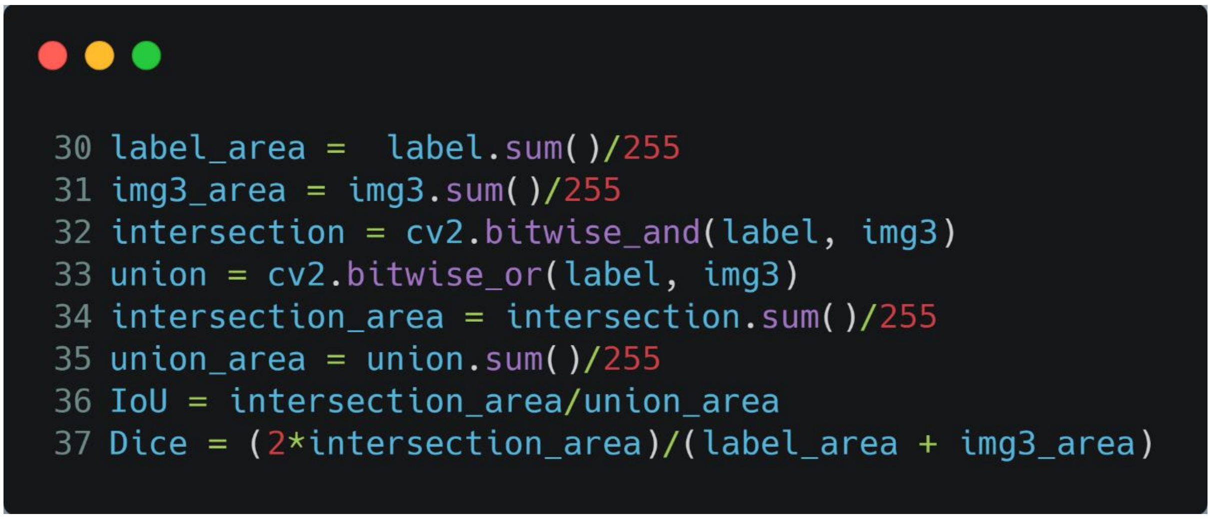


Figure 10

An example of poor segmentation can be seen in [Figure 10]. In this image, a good portion of the background is present, and the nails have a color that blends with the skin color.

IV. Results

The results per individual image of assessment accordingly to intersection over union metric (IoU) and Dice coefficient are presented per individual image and as a mean result for the whole data set [Table 1]. Notice that the results (img3 in code) are compared with the given labels that are assumed to be correct [Figure 11].



```python
label_area =  label.sum()/255
img3_area = img3.sum()/255
intersection = cv2.bitwise_and(label, img3)
union = cv2.bitwise_or(label, img3)
intersection_area = intersection.sum()/255
union_area = union.sum()/255
IoU = intersection_area/union_area
Dice = (2*intersection_area)/(label_area + img3_area)
```

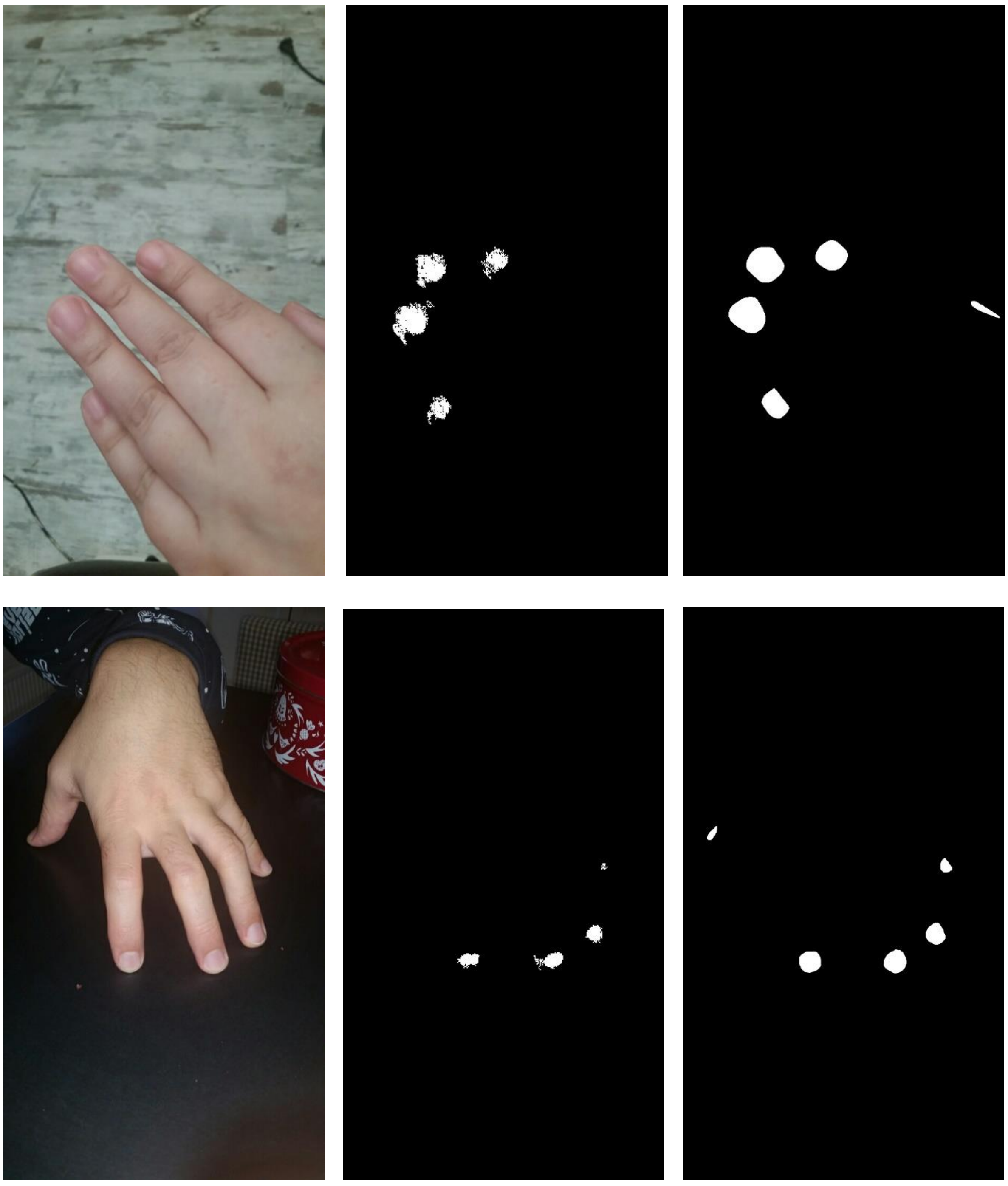
Figure 11

Image	**IoU**	**Dice**
09aefeec-e05f-11e8-87a6-0242ac1c0002.jpg	0.15	0.262
1eecab90-1a92-43a7-b952-0204384e1fae.jpg	0.049	0.094
2C29D473-CCB4-458C-926B-99D0042161E6.jpg	0.595	0.746
2c376c66-9823-4874-869e-1e7f5c54ec7b.jpg	0	0
34404E67-4BB8-432E-863E-C1BEF5EB37E0.jpg	0.02	0.039
3493127D-7B19-4E50-94AE-2401BD2A91C8.jpg	0.541	0.702
41d83dbb-7c39-4d91-979e-eec5ff71b265.jpg	0.198	0.331
4252e46c-e40f-4543-91ab-031917d46c5c.jpg	0.328	0.494
4c46b91a-e402-11e8-97db-0242ac1c0002.jpg	0	0
4c472e18-e402-11e8-97db-0242ac1c0002.jpg	0	0
4c478f3e-e402-11e8-97db-0242ac1c0002.jpg	0	0
4c47ee66-e402-11e8-97db-0242ac1c0002.jpg	0	0
4c484f8c-e402-11e8-97db-0242ac1c0002.jpg	0	0
4c48acb6-e402-11e8-97db-0242ac1c0002.jpg	0.337	0.504
4c490eae-e402-11e8-97db-0242ac1c0002.jpg	0.019	0.037
4c49b502-e402-11e8-97db-0242ac1c0002.jpg	0.006	0.012

4c4a0dd6-e402-11e8-97db-0242ac1c0002.jpg	0.038	0.074
4c4a6402-e402-11e8-97db-0242ac1c0002.jpg	0.043	0.083
54108996-6DA8-48F9-93DF-7ABB92F64E03.jpg	0	0
5CBCC5AC-B638-4DFA-AA7B-2464FADFF2F7.jpg	0.024	0.046
5fad3947-76d7-4352-9329-4a92f898dd59.jpg	0.13	0.231
7e9f5818-4425-4d8a-808a-4673d96fa250.jpg	0.022	0.044
865a1e90-7ad2-4ceb-b2a1-50b07875c5c7.jpg	0.386	0.557
869CDA2E-8251-4880-89D6-9409CBC416F3.jpg	0.849	0.918
917FB1CC-E132-497F-A463-46C61A84BB40.jpg	0	0
964c11f9-fbe3-420e-8674-9bb356cb5d3f.jpg	0.476	0.645
9ABD7CB9-7617-4A66-8892-B335FDB0E89F.jpg	0	0
a3a73edd-1483-4413-addb-9a7264b5d853.jpg	0.104	0.188
ABD5AB5C-E7CA-4420-A983-2161029ECC9E.jpg	0.167	0.287
af5626a5-feb4-4b2a-8c57-152e443ea3dc.jpg	0.12	0.215
b25b81b1-d562-4891-ae4f-83a82a2b64b5.jpg	0	0
bf93c2e2-7b5f-4108-ae85-4ef68564d418.jpg	0.107	0.194
C03527E3-ADED-423E-9F11-D797EA35E286.jpg	0	0
cc25f453-04ca-4c24-92bc-f26c5a94e78c.jpg	0	0
d60153c0-db67-11e8-9658-0242ac1c0002.jpg	0.305	0.468
d6072ec6-db67-11e8-9658-0242ac1c0002.jpg	0	0
d60a5f06-db67-11e8-9658-0242ac1c0002.jpg	0.027	0.052
d60f3ecc-db67-11e8-9658-0242ac1c0002.jpg	0.001	0.001
d61a32a0-db67-11e8-9658-0242ac1c0002.jpg	0.619	0.764
d61d6de4-db67-11e8-9658-0242ac1c0002.jpg	0.262	0.415
d62b0cd8-db67-11e8-9658-0242ac1c0002.jpg	0.219	0.359
d6303a64-db67-11e8-9658-0242ac1c0002.jpg	0	0
d6321726-db67-11e8-9658-0242ac1c0002.jpg	0.048	0.091
d633f320-db67-11e8-9658-0242ac1c0002.jpg	0.134	0.236
d6358550-db67-11e8-9658-0242ac1c0002.jpg	0.135	0.237
d63890ec-db67-11e8-9658-0242ac1c0002.jpg	0.006	0.013
d639e532-db67-11e8-9658-0242ac1c0002.jpg	0.007	0.013
d97db2d2-18ff-456d-8d85-23bfb7109aef.jpg	0	0
da236f3a-8c82-4c64-9a7d-9b950fd8b47e.jpg	0.101	0.184
edff1324-0b02-4471-b449-14cea27f0094.jpg	0	0
F6F9B3E6-FA7B-4DAC-B08C-1AD19BC43A76.jpg	0.403	0.574
feb2c029-b89c-4ce5-b208-db2114516a40.jpg	0.511	0.676
Average	0.144	0.207

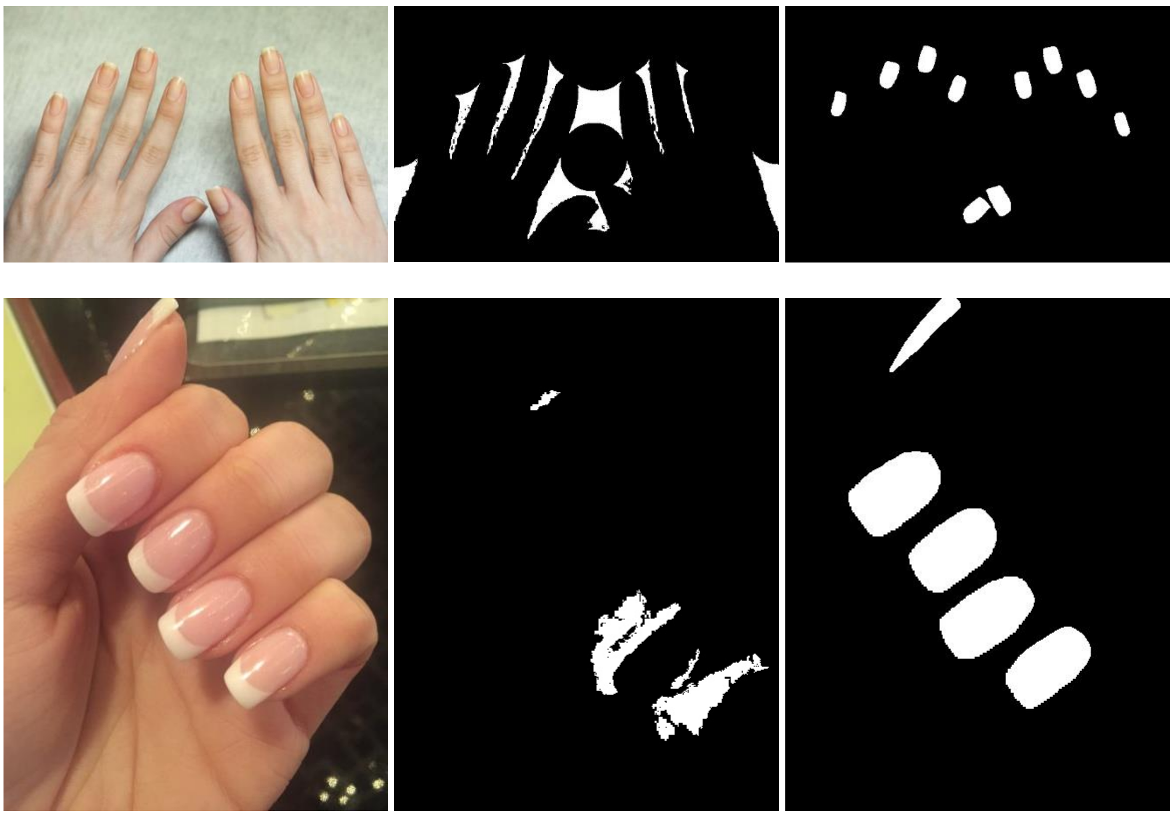
Table 1

- Good segmentation example:



Input image Segmentation Label

- Bad segmentation example:



Input image Segmentation Label

V.Comments and Conclusions

If we accept an IoU of 0.3 to be considered well classified, then there are a total of 11 images that are well classified, which means that 21% of the images are well classified, while the remaining 80% are not. The fact that the classification is poor may be related to the fact that some of the initial assumptions about the dataset were not true. The assumption that the nail and the background have a slightly different color than the hand is particularly questionable, especially since most of the images that were not classified correctly do not satisfy this property. We assume in our algorithm that the color information is relevant enough to classify correctly in all circumstances, but this was not the case. Although not covered in this paper, edge-based segmentation, clustering, and region-growing techniques have been applied with little success; texture analysis may be applied to improve segmentation results, but further investigation is needed.

VI.References

[1] Ross, L. and Russ, J.C., 2011. The image processing handbook. Segmentation and Thresholding p.395

[2] Rafael C. Gonzalez and Richard E. Woods. 2006. Digital Image Processing (3rd Edition). Prentice-Hall, Inc., USA. Image Segmentation p.699

[3] Kolkur, S., Kalbande, D., Shimpi, P., Bapat, C. and Jatakia, J., 2017. Human skin detection using RGB, HSV and YCbCr color models. arXiv preprint arXiv:1708.02694.

[4] Docs.opencv.org. 2020. Opencv: Opencv Modules. [online] Available at: <https://docs.opencv.org/master/index.html> [Accessed 26 December 2020].