Computer Vision Project Report
Leaves Segmentation

Szymon Ślusarz

Faculty of Mathematics and Information Science, Warsaw University of Technology

December 22, 2022

1 Introduction

This project's goal is to acquire knowledge about techniques frequently used in image processing. This time we will be focusing on image segmentation, which is a process of partitioning digital image into segments (e.g. regions or objects). We use image segmentation to obtain images that are easier to analyse or process.

During this project, we will work on public dataset called "Plant Village", which contains images of leaves with various diseases, as well as their representation in grayscale colormap and their segmented versions. In my case, I will be working on grape leaves suffering from black rot disease.

The project is divided into two subtasks. In the first one we have to take segmented images from the mentioned dataset, and create ground truth binary masks out of them, mainly using thresholding. Second subtask is about finding segmentation masks to each of the color leaves images, at the end we have to compare them with previously generated ground truth masks and calculate dice coefficient and Jaccard index to check their accuracy.

Before proceeding to the main part of the report, I inspected all segmented leaves from the dataset and removed ones that were very poorly segmented, to avoid downgrading accuracy due to dataset errors.

Figure 1: Examples of removed segmentations

2 Description of solution

2.1 Preparing ground truth binary labels

Figure 2: Segmented leaf from datset

Having an image of the segmented leaf (Fig. 2) imported to our python script, I had to perform a few steps to obtain the best possible ground truth masks.

- First of all, I had to convert segmented image to grayscale color map in order to perform binary thresholding on it. I did it using the following command:

```
# changing color space of segmented image to grayscale
gray = cv2.cvtColor(seg, cv2.COLOR_RGB2GRAY)
```

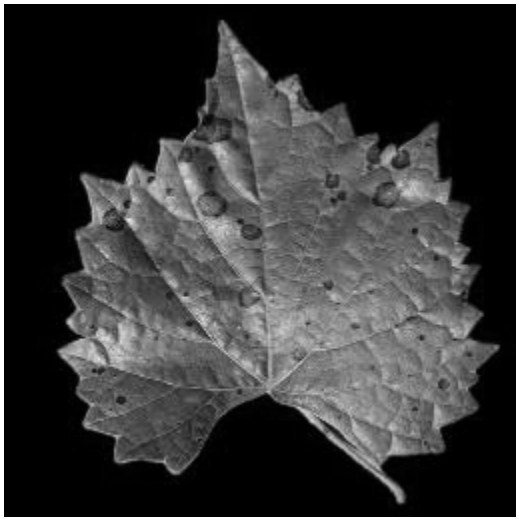

Figure 3: Segmented leaf in grayscale

- After that, I opened histogram of figure 3 in GIMP software, to select proper threshold to create a mask. I checked that the ideal threshold is around 16, so I used this value in the following command and obtained such mask (Fig. 4)

```
# binary thresholding
T, thresh = cv2.threshold(gray, 16, 255, cv2.THRESH_BINARY)
```

Figure 4: Leaf mask after thresholding

- As we can see the mask is still not perfect as it has noises within the contours of the leaf. That is because some pixels on the leaf (Fig. 3) are very dark, and the threshold considers them as a background. In order to get rid of them I did closing morphological transformation on the mask to fill (close) the noises, which resulted in the ground truth mask (Fig. 5)

```
# removing noice inside leaf's contours
mask_gt = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel=np.ones((5,
    5), dtype=np.uint8))
```

Figure 5: Ground truth leaf mask

2.2 Main segmentation

In this section I will describe my main segmentation algorithm on the same leaf as in the previous section (Fig. 6), but I will use its base, not segmented color version, and prepare a mask from it.

Figure 6: Leaf image from dataset

First of all to start segmentation process of the above image, I decided to change its colorspace to HSV (Hue, Saturation, Value) (Fig. 7), as it will be much easier to distinguish color ranges, background and all unnecessary shades. I did it using the following command.

```
# changing color space of image to hsv
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)))
```

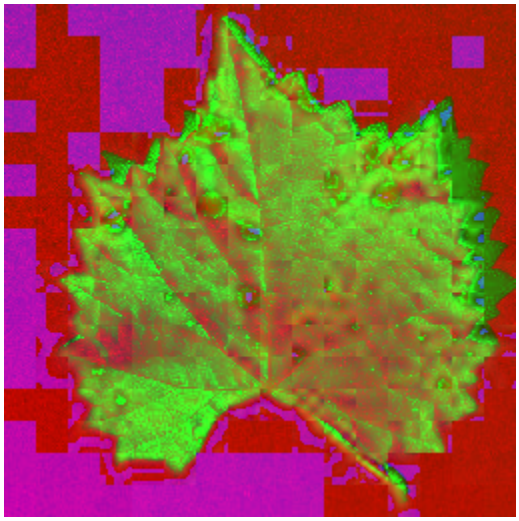

Figure 7: Leaf image in HSV colormap

After converting a color space to HSV, I proceeded to the most challenging part of the task, which was to create a proper color masks. Personally, I decided to use two color masks, first one with shades of green and second one with shades of brown. I chose such colors because every image consists of greenish leaf with brownish spots. To obtain these masks I had to find green and brown color ranges. I did by testing the ranges partly intuitively, and partly using GIMP app.

```
# setting brown and green hsv color ranges
lower_brown = np.array([0, 95, 86])
upper_brown = np.array([31, 255, 200])

```

```
lower_green = np.array([25, 37, 33])
upper_green = np.array([120, 255, 255])
```

Having them, I could create two masks shown on (Fig. 8) with:

```
# creating two masks using the above color ranges
green_mask = cv2.inRange(hsv, lower_green, upper_green)

brown_mask = cv2.inRange(hsv, lower_brown, upper_brown)
```

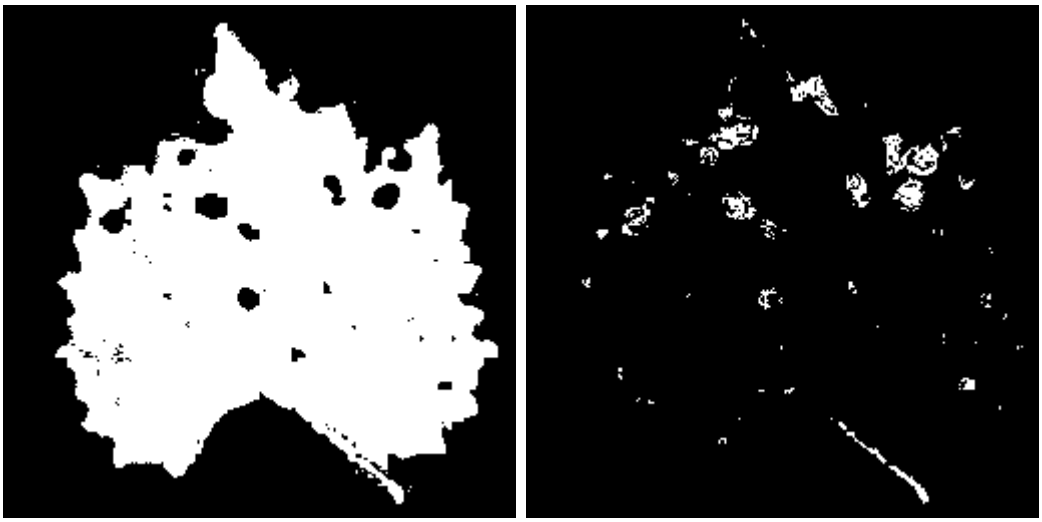

Figure 8: Green and brown color masks

As it is visible on the above images is that brown mask takes also some shades of yellow color. It was caused by the slight extension of the brown color range in order to obtain better results. Next step of the algorithm was to connect two color masks into one general mask (Fig. 9).

```
# connecting two masks into one
mask = cv2.bitwise_or(green_mask, brown_mask)
```

Figure 9: Connected color masks

On the above mask we can see that there are still some spots that were not covered by brown mask, but that is some kind of a compromise, because by avoiding some shades of

brown, I omitted noises resulting from the brownish background. However, I managed to fill the spots within the leaf edges using closing morphological transformations connected with slight Gaussian blurring and obtain the final mask (Fig. 10). The reason behind using two closing transformations and no opening, is that majority of masks I created, do not contain background noises, so I decided to focus on cleaning up the region within the area of the leaf.

```
# removing noise and empty spots with morphology and blur
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel=np.ones((5, 5),
    dtype=np.uint8))
mask = cv2.medianBlur(mask, 7)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel=np.ones((5, 5),
    dtype=np.uint8))
```

Figure 10: Final segmentation mask generated by the algorithm

2.3 Possible improvements

As we should know from the lectures, there is no one 'correct' segmentation for an image, and it is clearly visible that my segmentation mask (Fig. 10) is not perfect. The edges of the leaf are not as sharp as on the original image, due to usage of morphological transformations and blurring. But that was necessary to fill the spots not covered by the created color masks. The solution to this seems to be self-evident - modify color ranges. But as I mentioned previously, such modifications would require from us to use different operations to remove noises from the background, which might also lead to not sharp enough edges.

Another idea is to create masks for more colors, however that also might cause some problems while segmenting leaf with a bit different shades of masking colors, but overall this solution could give higher similarity coefficients.

Summarising, all of these approaches will give us high accuracy scores, but it is not possible to create one algorithm that will be 100% accurate for the whole dataset. Certainly each solution has its strengths and problems you have to deal with.

3 Results

3.1 Calculating mean results

The last part of the task was to present obtained results, using two different similarity coefficients:

- Dice coefficient

$$Dice = \frac{2|A \cap B|}{|A| + |B|}$$

  Mean dice coefficient for the whole dataset was **97.7%**

- Intersection over Union (Jaccard index)

$$IoU = \frac{target \cap prediction}{target \cup prediction}$$

  Mean IoU for the whole dataset was **95.5%**

As we can see from the above results, my algorithm turned out to be very accurate.

```
# calculating dice coefficient and IoU
intersect = np.sum(cv2.bitwise_and(mask, mask_gt))
sum1 = np.sum(mask)
sum2 = np.sum(mask_gt)

dice = (2 * intersect) / (sum1 + sum2)
iou = intersect / np.sum(cv2.bitwise_or(mask, mask_gt))
```

3.2 Worst cases

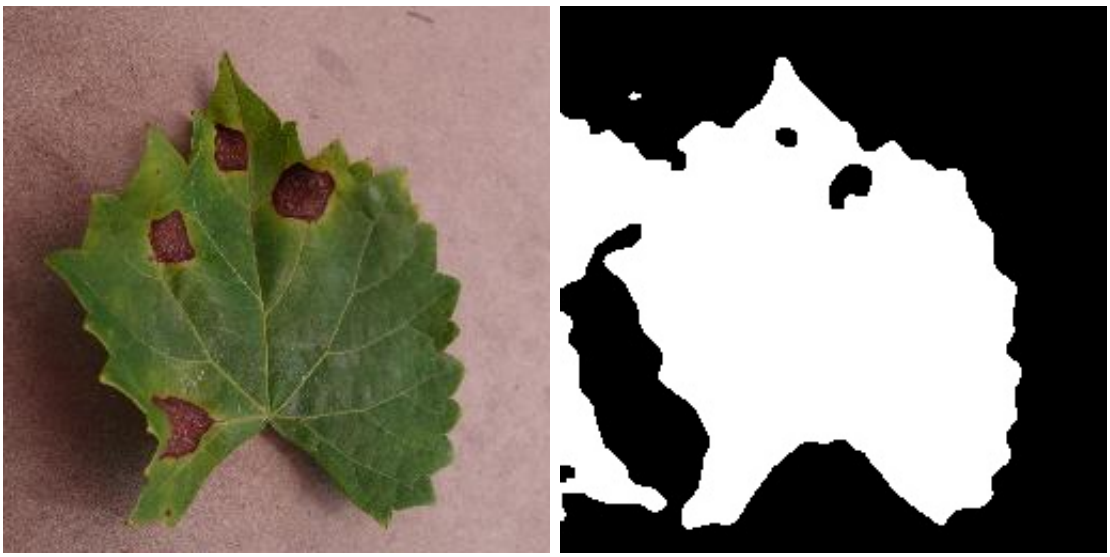

Figure 11: Dice ≈ **89%**, IoU ≈ **80,1%**

Figure 12: Dice ≈ **91%**, IoU ≈ **803,5%**

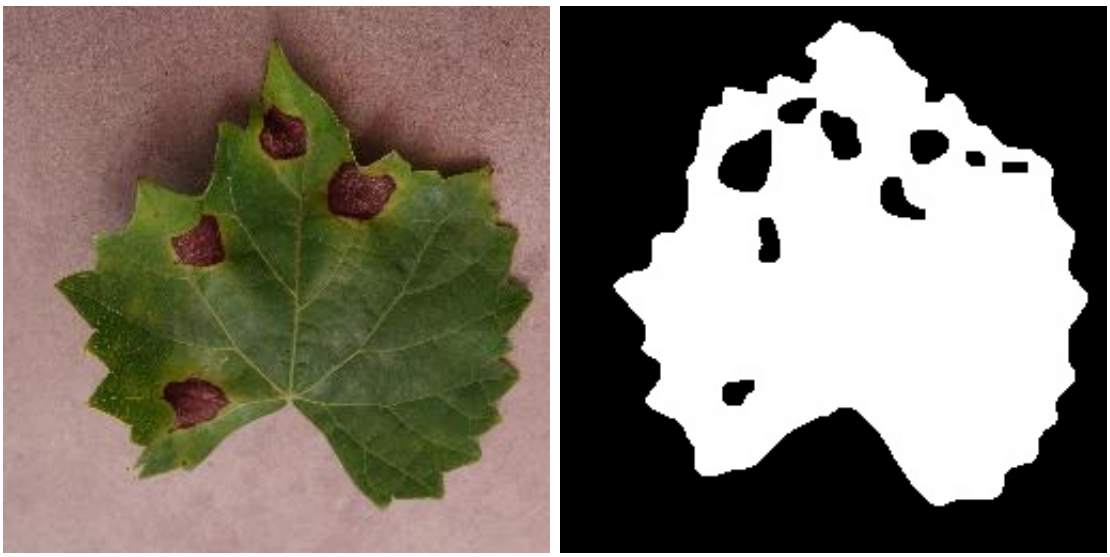

Figure 13: Dice ≈ **91%**, IoU ≈ **83,5%**

Figure 14: Dice ≈ **92%**, IoU ≈ **85,2%**

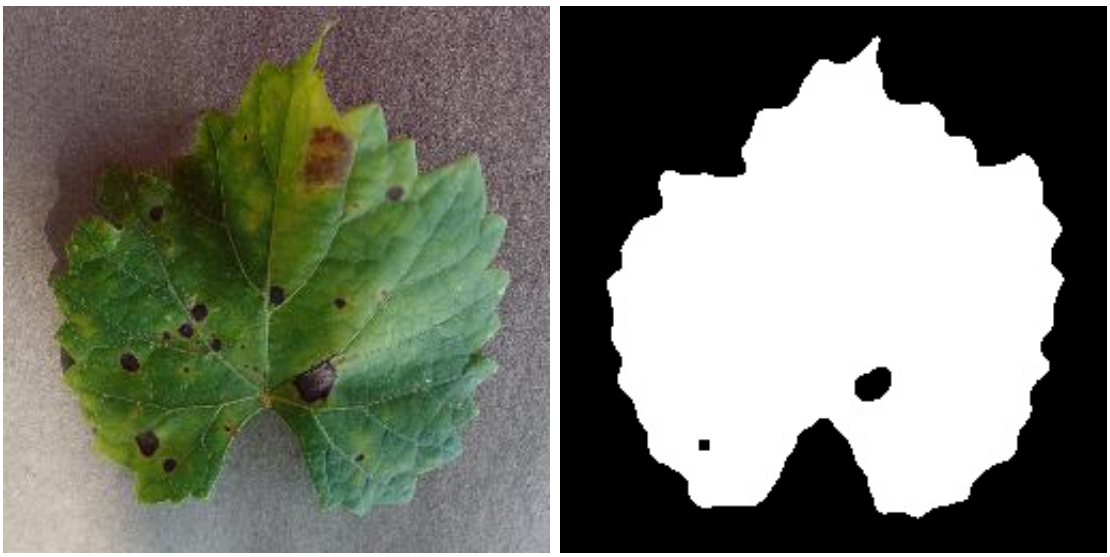

Figure 15: Dice ≈ **92%**, IoU ≈ **85,3%**

From the worst leaves examples I can conclude that my algorithm had problems with very dark shades of brown, also on some images it wrongly detected brownish part of a leaf's shade as a part of a leaf.

3.3 Best cases

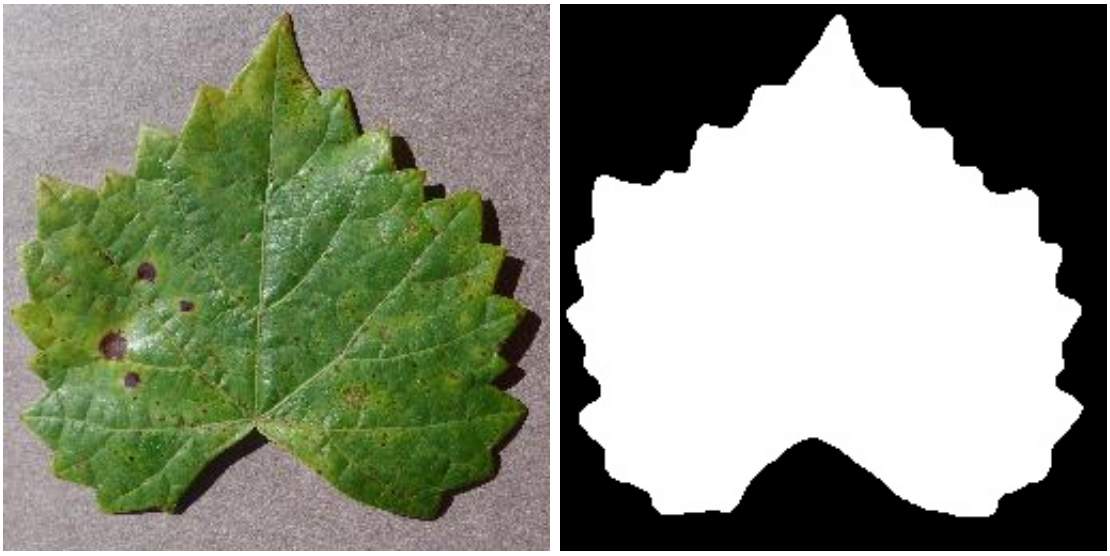

Figure 16: Dice ≈ **99,1%**, IoU ≈ **98,2%**

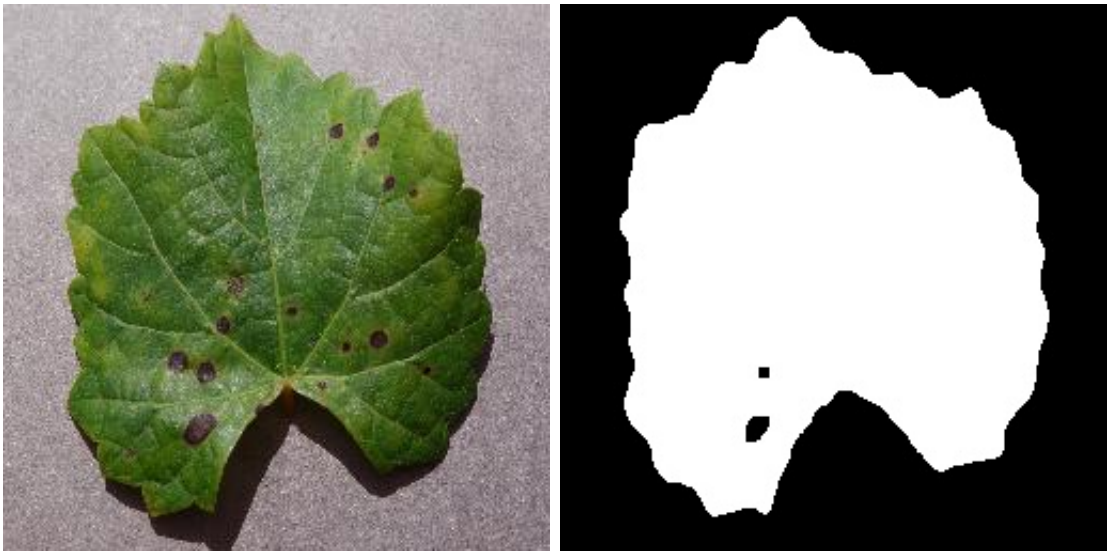

Figure 17: Dice ≈ **99,1%**, IoU ≈ **98,2%**

Figure 18: Dice ≈ **99,1%**, IoU ≈ **98,3%**

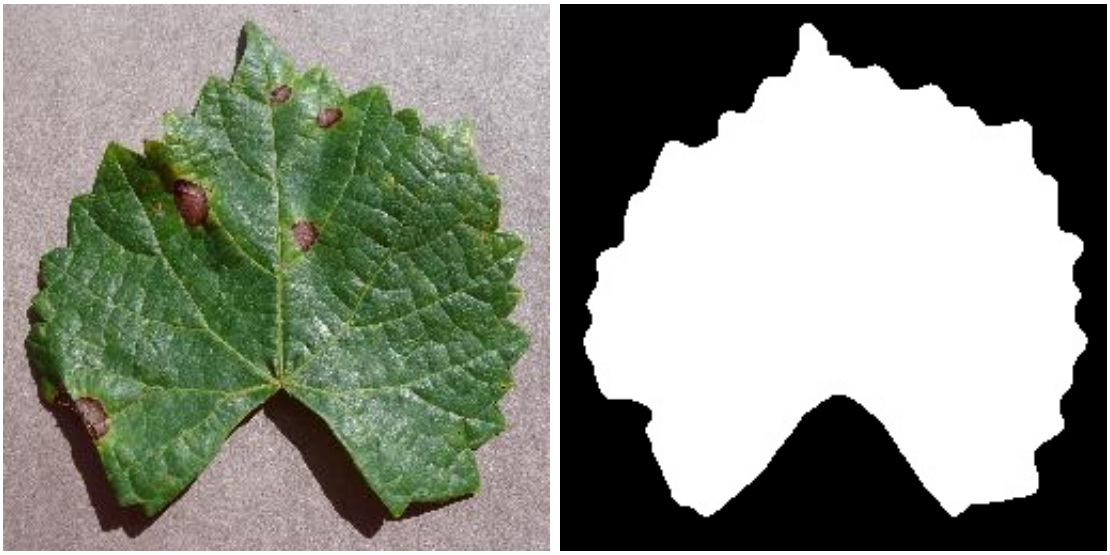

Figure 19: Dice ≈ **99,1%**, IoU ≈ **98,3%**

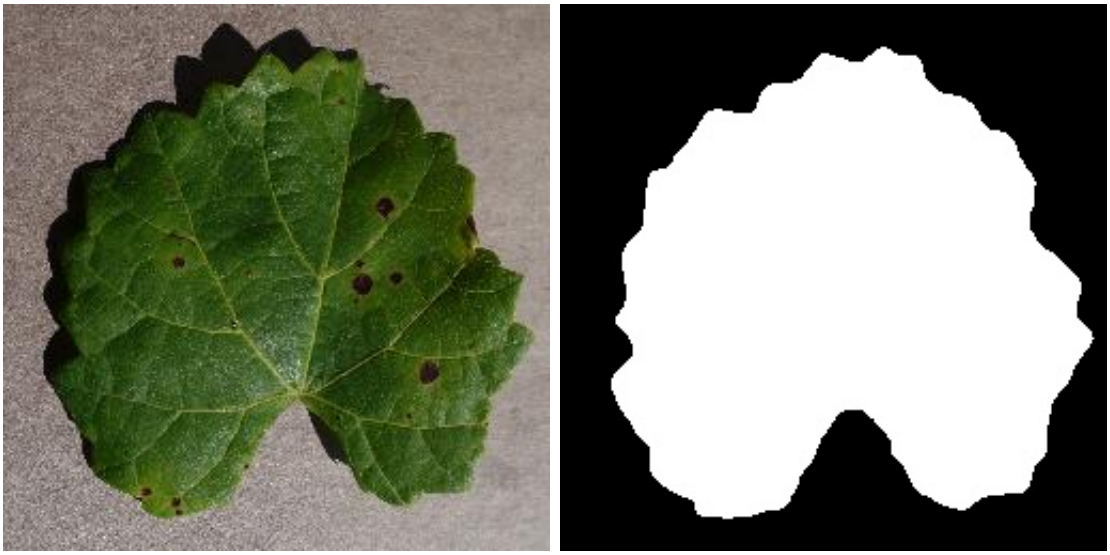

Figure 20: Dice ≈ **99,2%**, IoU ≈ **98,4%**

We can see, and what coefficients tell us, is that these five leaves were segmented extremely accurately. On some of them we can still notice small issues, but we can treat them as acceptable.

4 Conclusions

Summarising, my algorithm, despite its few minor flaws with color ranges, turned out to be impressively accurate comparing to the earlier prepared ground truth masks. During this project I learned a lot about the whole process of image segmentation, and why it is so challenging process for computer scientists. I encountered an unexpected number of issues, but luckily I managed to deal with the most of them.