

Introduction to image processing and computer vision

Plant Segmentation and Labeling

Laboratory Project I

Patryk Walczak

January 8, 2020

Contents

1	Introduction	1
1.1	Project Description	2
2	Masks and Bounding Boxes	3
2.1	Masks - algorithm description	3
2.2	Bounding Boxes - algorithm description	6
2.3	Saving the results	6
2.4	Comparision	7
3	Segmented Leaves	10
3.1	Label - algorithm description	10
3.2	Saving the results	11
3.3	Comparision	12
4	Source Code	14
4.1	mask.py	14
4.2	boundingBoxes.py	16
4.3	labeling.py	17
4.4	saveToFile.py	19
4.5	mask.py	20
4.6	labelComapison.py	23
4.7	test.py	26
5	References	28

1 Introduction

Main task is about finding the best mask for each plant in the dataset (900 images of plants) using image segmentation, the process of splitting the digital image into several objects (segments). More precisely, separate the plant from the background and optionally make the bounding boxes. Then the second part of the project is about dividing the whole plant on distinct leaves, for the better view make each mask of the leaf in a different colour. All results are saved and compared with the patterns.

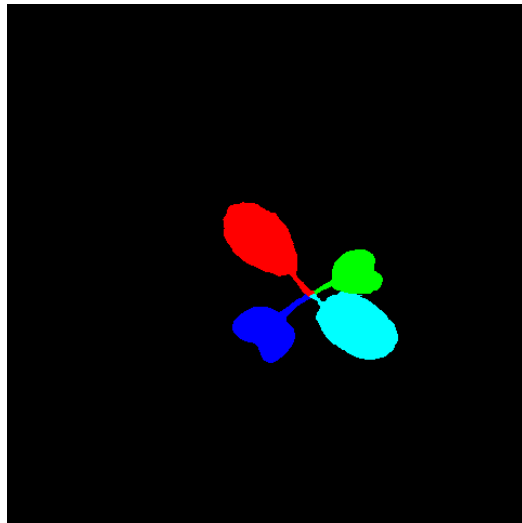
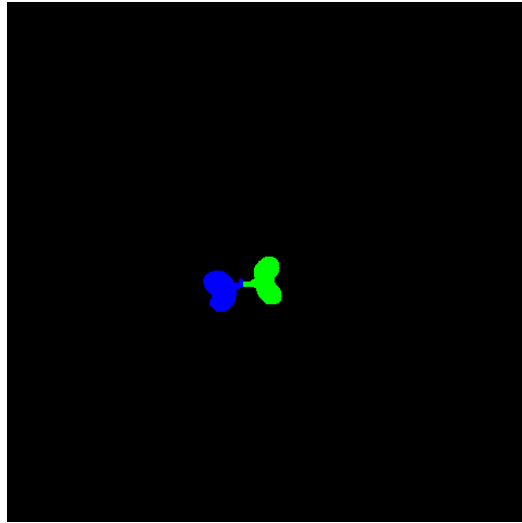




Figure 1: Some of the images and their patterns

1.1 Project Description

The dataset contains images of 5 plants made by 3 cameras every 4 hours by 10 days. For the 900 images I prepared several separated files with the algorithms for finding masks, labels, bounding boxes, making comparisons, and for saving the result to the files.

List of files :

- mask.py - finding the best mask for each image
- boundingBoxes.py - using the mask to create the bounding box
- labeling.py - create the separate mask for each leaf
- saveToFile.py - save masks, labels and bounding boxes to files
- maskComparison.py - compare my masks with the expectations
- labelComapison.py - compare my labels with the expectations

2 Masks and Bounding Boxes



Figure 2: Example of my results

2.1 Masks - algorithm description

Images contain plant on the white background, which makes the finding mask easier but some of them contain some other green objects also (e.g. bottle), what could be a problem.



My algorithm as input takes the path to the image and the debug flag.

```
1 def mask(path, test):
```

Firstly my algorithm reads the image with colours, and remove generally all colours except green. For finding the best values for lower and upper array I prepared test.py, where in real-time I could change the values and observe how it affects the mask.

Part of the code. Whole file is in Source Code section.

```
1  hsv = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
2
3  # get info from track bar and apply to result
4  hM = cv2.getTrackbarPos('h M','result')
5  hL = cv2.getTrackbarPos('h L','result')
6  sM = cv2.getTrackbarPos('s M','result')
7  sL = cv2.getTrackbarPos('s L','result')
8  vM = cv2.getTrackbarPos('v M','result')
9  vL = cv2.getTrackbarPos('v L','result')
10 # b1 = cv2.getTrackbarPos('blur1','result')
11 # b2 = cv2.getTrackbarPos('blur2','result')
12
13 # Normal masking algorithm
14 lower = np.array([hL,sL,vL])
15 upper = np.array([hM,sM,vM])
16
17 res = cv2.inRange(hsv,lower, upper)
```

```
1  hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
2  lower_green = np.array([0, 0, 0],np.uint8)
3  upper_green = np.array([179, 255, 165],np.uint8)
4  mask = cv2.inRange(hsv, lower_green , upper_green)
```

Only the plant should be taken into consideration, so my algorithm takes the most centre contour. But if the contour is too big it means that the algorithm takes a plant and some other object. So additional removing of the green shades is needed.

```
1  # finding contur of the biggest area
2  ret,thresh = cv2.threshold(mask,127,255,0)
3  contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,\
4  cv2.CHAIN_APPROX_SIMPLE)
5  c = sorted(contours, key = cv2.contourArea, reverse = True)[0]
6
7  # if the biggest contour has the area > 40000 then remove more
   precisely
8  # all colors exept green
9  if cv2.contourArea(c) > 40000:
10 segmented = cv2.bitwise_and(image , image , mask=mask)
11 hsv = cv2.cvtColor(segmented , cv2.COLOR_BGR2HSV)
12 lower_green = np.array([27, 29, 0],np.uint8)
13 upper_green = np.array([179, 255, 165],np.uint8)
14 mask = cv2.inRange(hsv, lower_green , upper_green)
15 if test :
16 cv2.imshow("warunek", mask)
17 cv2.waitKey()
18 ret,thresh = cv2.threshold(mask,127,255,0)
19 contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE, cv2.
   CHAIN_APPROX_SIMPLE)
20 # set minArea = 5000
21 minArea = 5000
22 else :
23 # set minArea = 1000
24 minArea = 1000
```

The variable *minArea* is specified for each case and it defines the lower bound of the contour's size.

Then take the most centered biggest contour with the area bigger than *minArea*, and save the mask.

```
1 # finding the generall shape of plant as the most center contour
2 # of the area > minArea
3 closest=1000
4 for c in contours :
5     if cv2.contourArea(c)>minArea:
6         M = cv2.moments(c)
7         cX = int(M["m10"] / M["m00"])
8         cY = int(M["m01"] / M["m00"])
9         dist=abs(width/2-cX)+abs(height/2-cY)
10        if closest > dist:
11            closest=dist
12        cnt=c
13
14 # take the generall mask of the plant
15 mask = cv2.drawContours(np.zeros((height ,width ,3), np.uint8 ), [
16     cnt], 0, (255,255,255), cv2.FILLED)
17 mask = cv2.cvtColor(mask ,cv2.COLOR_BGR2GRAY)
```

After that, the algorithm takes the segmented mask to operate only on the plant part.

```
1 # take only part of the mask from the image
2 segmented = cv2.bitwise_and(image , image , mask=mask)
```

And it applies more precisely removing of the rest green shades. If the step had been used before choosing the plant contour would have been harder because the plant would have been splited in many parts.

```
1 # more precisely remove all colors exept green
2 # only from the segmented part of the image
3 hsv = cv2.cvtColor(segmented , cv2.COLOR_BGR2HSV)
4 lower_brown = np.array([37, 31, 28],np.uint8)
5 upper_brown = np.array([86, 255, 134],np.uint8)
6 mask_seg = cv2.inRange(hsv, lower_brown , upper_brown)
```

At the end it applies erosion and dilatation, the number of iterations depends on the area of the biggest contour.

```
1 # perform a series of erosions and dilations depends on
2 # the max contour area
3 kernel22 = cv2.getStructuringElement(cv2.MORPH_RECT, (1,1))
4 mask_seg = cv2.morphologyEx(mask_seg, cv2.MORPH_OPEN , kernel22)
5 ret,thresh = cv2.threshold(mask_seg,127,255,0)
6 contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
7 # take the biggest contur
8 maxContour = sorted(contours, key = cv2.contourArea, reverse = True)
9 [0]
10 # set size of the contur devided by 25000
11 size = int(cv2.contourArea(maxContour)//25000)
12 # perform erosion and dilatation with size iteration
```

```

12 mask_seg = cv2.erode(mask_seg, None, iterations = size)
13 mask_seg = cv2.dilate(mask_seg, None, iterations = size)

```

Finally, the algorithm returns the mask.

```

1 # and return the mask of the plant
2 return mask_seg

```

2.2 Bounding Boxes - algorithm description



The algorithm for boxes is very short. It takes the mask and the image, for the mask, prepares the smallest rectangle containing the whole mask and applies the rectangle to the image.

```

1 def bounding_boxes(p):
2     mask = main(p, False)
3     image = cv2.imread(p, cv2.IMREAD_COLOR)
4     x, y, w, h = cv2.boundingRect(mask)
5     cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
6     return image

```

2.3 Saving the results

For saving I prepared separate file, creating directories if those don't exist, and save the result to them.

All result masks are in the directory "my_mask", and all result bounding boxes are in "my_bounding_boxes" directory.

```

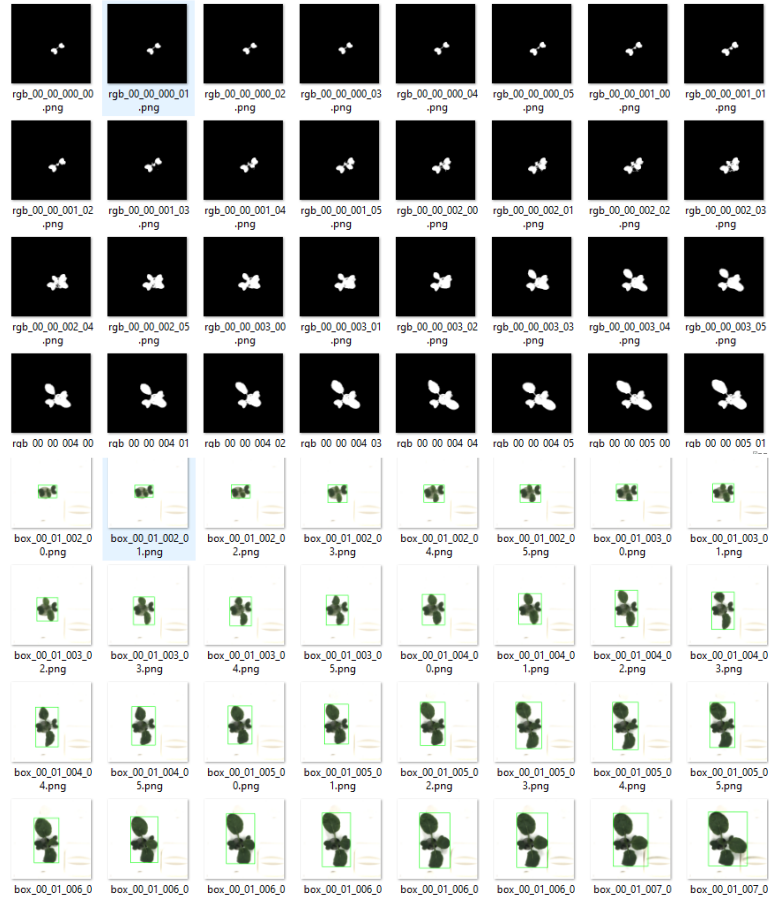
1 directoryMask = 'my_masks'
2 directoryBoxes = 'my_bounding_boxes'
3
4 # save maske
5 def saveMasks(name):
6     image = mask(name, False)
7     name = os.path.basename(name)

```

```

8 cv2.imwrite(pathToWrite+directoryMask+'/' +name, image)
9
10 #save label
11 def saveLabels(name):
12     image = sum(label(name, False))
13     name = os.path.basename(name)
14     cv2.imwrite(pathToWrite+directoryLabel+'/' +name.replace('rgb', 'label
    '), image)

```



2.4 Comparision

I prepared also a new file "maskComparison.py" for checking the final result. Obligatory was including Intersection over Union metric and Dice coefficient. There are also 2 additional comparisons : Jaccard Index and Structural Similarity Index.

All comparison results are in the directory "ComparisonMasks", in separate files. Each plant has its result, at the end of each comparison result file there comes its summary. Images are read in the following way:


```

1 # read the pattern
2 def pattern(path):
3     image = cv2.imread(pathToPatterns+path)
4     image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
5     ret,th1 = cv2.threshold(image,0,255,cv2.THRESH_BINARY)
6     # return binary mask
7     return th1
8
9 # read my mask
10 def my_mask(path):
11     image = cv2.imread(pathToMyMasks+path.replace('label','rgb'))
12     image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
13     return image

```

And the comparison functions look like:

```

1 # IoU comparison
2 def IoU_compare(my_mask,pat):
3     img1 = np.asarray(pat).astype(np.bool)
4     img2 = np.asarray(my_mask).astype(np.bool)
5     num = np.sum(np.logical_and(img1,img2))
6     den = np.sum(np.logical_or(img1,img2))
7     return num/den
8
9 # dice coefficient coparison
10 def dice_coeff_compare(my_mask, pat):
11     img1 = np.asarray(pat).astype(np.bool)
12     img2 = np.asarray(my_mask).astype(np.bool)
13     img_intersection = np.bitwise_and(img1, img2)
14     image_sum = img1.sum() + img2.sum()
15     if image_sum == 0:
16         return 0
17     return 2. * img_intersection.sum() / image_sum
18
19 # ssim coparison
20 def ssim_compare(my_mask, pat):
21     (score, diff) = compare_ssim(my_mask, pat, full=True)
22     diff = (diff * 255).astype("uint8")
23     return score
24
25 # jaccard comaprison by function
26 def jaccard_compare(my_mask, pat):
27     score = jaccard_similarity_score(my_mask.flatten(), pat.flatten())
28     return score

```

Results are also printed at the end of the program.

```

---Jaccard Index---
average 99.24
minimal value 96.61  rgb_00_04_009_03.png
maximal value 99.98  rgb_00_03_000_04.png
--- IoU metric ----
average 88.64
minimal value 72.73  rgb_01_04_003_03.png
maximal value 98.28  rgb_02_04_009_05.png
--- Dice coefficient ----
average 93.85
minimal value 84.21  rgb_01_04_003_03.png
maximal value 99.13  rgb_02_04_009_05.png
---Structural Similarity Index---
average 97.68
minimal value 91.67  rgb_00_04_009_03.png
maximal value 99.89  rgb_00_03_000_04.png

```

```

rgb_00_00_009_00.png 94.69827986982799
rgb_00_00_009_01.png 93.77848754034002
rgb_00_00_009_02.png 94.11177342733517
rgb_00_00_009_03.png 94.14398946976851
rgb_00_00_009_04.png 93.45685848177509
rgb_00_00_009_05.png 94.43584100850381
rgb_00_01_000_00.png 91.64870689655173
rgb_00_01_000_01.png 92.67382174521698
rgb_00_01_000_02.png 93.78827646544183
rgb_00_01_000_03.png 93.96164830681354
rgb_00_01_000_04.png 94.09056024558711
rgb_00_01_000_05.png 94.41878367975366
rgb_00_01_001_00.png 94.88945270025371
rgb_00_01_001_01.png 93.89237372069991
rgb_00_01_001_02.png 94.94672754946728
rgb_00_01_001_03.png 94.11923388184516
rgb_00_01_001_04.png 94.38887533544768
rgb_00_01_001_05.png 93.65573378022503
rgb_00_01_002_00.png 93.41983317886933

```

The results are not perfect because in some images the shadow of the plant has the same colour as the part of a leaf. In others, flowerpot is also green, and it is impossible to remove the pot without erasing the plant. However almost all images are around 90%, and the worst score is still over 72%.

3 Segmented Leaves

The second part of the project is about segmentating the plant on the leaves.

3.1 Label - algorithm description

The whole described here algorithm is in "labeling.py" file.

Firstly, algorithm takes the result of the previous part, the mask.

```
1 my_mask = mask(p,False)
```

Then it finds the biggest contour and computes the area of the contour.

```
1 # find the biggest contour
2 ret,thresh = cv2.threshold(my_mask,127,255,0)
3 contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.
    CHAIN_APPROX_SIMPLE)
4 maxContour = sorted(contours, key = cv2.contourArea, reverse = True)
    [0]
5 maxContourArea = cv2.contourArea(maxContour)
```

After that program prepares the variables for next functions. The values depends on the area of the max contour. One of the cases is below.

```
1 #cases of erosion and dilatation depends on the area
2 if maxContourArea < 7500 :
3     size = int(cv2.contourArea(maxContour)//10000) + 5
4     erod = size
5     dilat = 3 * size
6     minArea = 0
7 else:
```

Then erosion is performed.

```
1 # perform erosion
2 mask_erode = cv2.erode(my_mask, None, iterations = erod)
```

Once again it takes all contours, sorts them and for every one makes the mask, dilatation and segmentation with my_mask. Save all of them in list l.

```
1 contours = sorted(contours, key = cv2.contourArea, reverse=True)
2 for cnt in contours:
3     if cv2.contourArea(cnt)>minArea:
4         # make the mask of the contour and dilate it
5         m = cv2.drawContours(np.zeros((height ,width ,3), np.uint8 ), [cnt],
            0, colors[iter%(len(colors)+1)], cv2.FILLED)
6         m = cv2.dilate(m, None, iterations = dilat)
7         # segmented with my_mask
8         m = cv2.bitwise_and(m, m, mask=my_mask)
9         iter=iter+1
10        # add to list
11        l.append(m)
```

At the end algorithm removes intersections of the smaller and the bigger mask from the bigger one.

```
1 # for every mask in list
2 for i in range(0, len(l)-1):
3 #from bigger mask remove the intersection with the smaller
4 l[i] = removeBitwiseMask(l[i], l[i+1])
```

Finally, it returns the list of the masks.

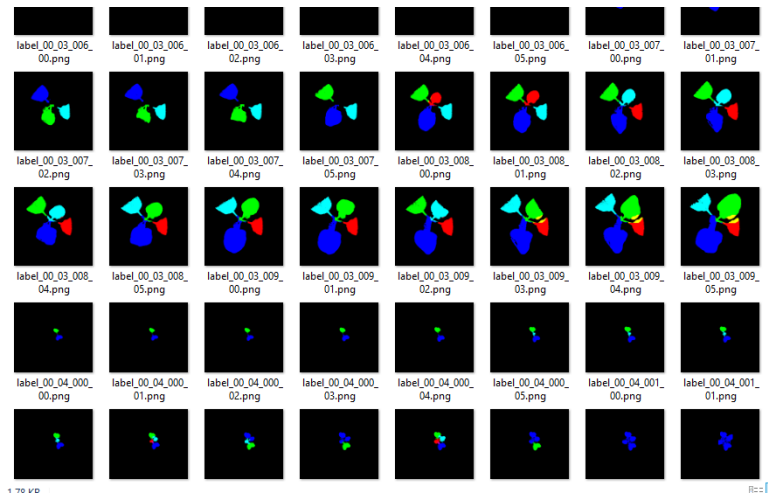
```
1 #return the list of mask
2 return l
```

3.2 Saving the results

Saving labels is in the same file "saveToFile.py" as in the case of masks. Before saving, the function has to sum all elements of the list returned by *label* function to save them as one file - as in the patterns.

The saving function is as follows:

```
1 #save label
2 def saveLabels(name):
3 image = sum(label(name, False))
4 name = os.path.basename(name)
5 cv2.imwrite(pathToWrite+directoryLabel+'/' +name.replace('rgb', 'label'
    '), image)
```



3.3 Comparision

For comparison labels, I prepared "labelComapison.py". But in the case only the two required comparisons are useful - Intersection over Union metric and Dice coefficient. The two remaining tests are out of point and return score above 90%, what is false result

All comparison results are in the directory "ComparisionLabels", in separate files. Single one consists the data for each for each leaf, for each plant and mean for the whole dataset. But in label case, the comparison is divided on every colour in the pattern. Each colour mask is separately saved in the dictionary.

```
1 # read the pattern
2 def pattern(path):
3     image = cv2.imread(pathToPatterns+path)
4     image = cv2.cvtColor(image, cv2.IMREAD_COLOR)
5     d = dict()
6     for c in colors:
7         arr = np.array(c)
8         res = cv2.inRange(image, arr, arr)
9         if np.sum(res) > 0:
10            segmented = cv2.bitwise_and(image, image, mask=res)
11            d[c] = segmented
12    return d
13
14 # read my mask
15 def my_label(path, keys):
16     image = cv2.imread(pathToMyLabels+path)
17     image = cv2.cvtColor(image, cv2.IMREAD_COLOR)
18     d = dict()
19     for c in keys:
20         arr = np.array(c)
21         res = cv2.inRange(image, arr, arr)
22         segmented = cv2.bitwise_and(image, image, mask=res)
23         d[c] = segmented
24    return d
```

Comparison functions are the same as in mask comparison

```
1 # IoU comparison
2 def IoU_compare(my_mask, pat):
3     img1 = np.asarray(pat).astype(np.bool)
4     img2 = np.asarray(my_mask).astype(np.bool)
5     num = np.sum(np.bitwise_and(img1, img2))
6     den = np.sum(np.bitwise_or(img1, img2))
7     return num/den
8
9 # dice coefficient coparision
10 def dice_coeff_compare(my_mask, pat):
11     img1 = np.asarray(pat).astype(np.bool)
12     img2 = np.asarray(my_mask).astype(np.bool)
13     img_intersection = np.logical_and(img1, img2)
14     image_sum = img1.sum() + img2.sum()
15     if image_sum == 0:
16         return 0
17     return 2. * img_intersection.sum() / image_sum
```

The result for the whole plant is the mean from the sum of all leaves divided by the number of leaves.

```

1 for key in pat.keys():
2 dice[str(name)+str(key)] = dice_coeff_compare(my_l[key], pat[key])
3 iou[str(name)+str(key)] = IoU_compare(my_l[key], pat[key])
4 d += dice[str(name)+str(key)]
5 u += iou[str(name)+str(key)]
6 file_dice.write(str(name)+" "+str(key)+" "+str(dice[str(name)+str(
    key)]*100)+"\n")
7 file_iou.write(str(name)+" "+str(key)+" "+str(iou[str(name)+str(key)
    ]*100)+"\n")
8 dice[str(name)] = d/len(pat)
9 iou[str(name)] = u/len(pat)
10 file_dice.write(str(name)+" "+str(dice[str(name)]*100)+"\n")
11 file_iou.write(str(name)+" "+str(iou[str(name)]*100)+"\n")

```

All results are saved in two files in "ComparisionLabels" directory. At the end of each file are summary containing : average, minimal and maximal score.

```

--- IoU metric ---
average 12.57
minimal value 0.0 label_00_00_001_00.png(255, 255, 0)
maximal value 96.77 label_00_03_000_04.png(255, 0, 0)
--- Dice coefficient ---
average 14.69
minimal value 0.0 label_00_00_001_00.png(255, 255, 0)
maximal value 98.36 label_00_03_000_04.png(255, 0, 0)

label_00_00_000_00.png (255, 0, 0) 87.46713409290096
label_00_00_000_00.png (0, 255, 0) 85.64867967853043
label_00_00_000_00.png 86.5579068857157
label_00_00_000_01.png (255, 0, 0) 86.24382207578253
label_00_00_000_01.png (0, 255, 0) 85.36324786324786
label_00_00_000_01.png 85.8035349695152
label_00_00_000_02.png (255, 0, 0) 84.34382194934767
label_00_00_000_02.png (0, 255, 0) 86.38253638253637
label_00_00_000_02.png 85.36317916594203
label_00_00_000_03.png (255, 0, 0) 86.56603773584905
label_00_00_000_03.png (0, 255, 0) 87.44897959183675
label_00_00_000_03.png 87.0075086638429
label_00_00_000_04.png (255, 0, 0) 88.34178131788559
label_00_00_000_04.png (0, 255, 0) 86.08445297504798
label_00_00_000_04.png 87.21311714646679
label_00_00_000_05.png (255, 0, 0) 86.71662125340599
label_00_00_000_05.png (0, 255, 0) 87.40875912408758
label_00_00_000_05.png 87.0626901887468
label_00_00_001_00.png (255, 0, 0) 89.48425987943737
label_00_00_001_00.png (0, 255, 0) 83.1146106736658
label_00_00_001_00.png (255, 255, 0) 0.0
label_00_00_001_00.png 57.53295685103439
label_00_00_001_01.png (255, 0, 0) 90.83056478405315

```

The results are far from perfect. The best scores are close to the results of mask comparison but the average shows that the algorithm is still not working well. The reason for that is probably the fact that mask is close to the pattern but the differences are significant. Bad mask of the plant implicates the bad masks for each leaf.

Moreover, the choosing of the colour was a big problem. Because the oldest one should have a blue colour. In the algorithm the biggest one is blue. And the comparison firstly divides the images by colours, so a big part of the older plants return in the comparison

the result close to 0. The oldest leaf in some photos is under the younger one so its visible area is no longer the biggest. Furthermore, the algorithm considers only one image which implies the problem, with finding the oldest one, hard to solve because nothing on the images marks the age of leaves.

But taking into consideration only the part with finding the leaves on the image the algorithm does quite well. The result of such a comparison might be close to 40-50%. This is still not perfect, but much closer to that.

4 Source Code

4.1 mask.py

```
1 # import the necessary packages
2 import numpy as np
3 import cv2
4 import random
5 import os
6 from math import sqrt as sqrt
7
8 def mask(path, test):
9     # load the image
10    image = cv2.imread(path,cv2.IMREAD_COLOR)
11    height, width, channels = image.shape
12    if test :
13        cv2.imshow("Image", image)
14        cv2.waitKey()
15
16    # remove generally all colors exept shades of green
17    hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
18    lower_green = np.array([0, 0, 0],np.uint8)
19    upper_green = np.array([179, 255, 165],np.uint8)
20    mask = cv2.inRange(hsv, lower_green , upper_green)
21    if test :
22        cv2.imshow("Image", mask)
23        cv2.waitKey()
24
25    # finding contur of the biggest area
26    ret,thresh = cv2.threshold(mask,127,255,0)
27    contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
28    c = sorted(contours, key = cv2.contourArea, reverse = True)[0]
29
30    # if the biggest contur has the area > 40000 then remove more
    precisely all colors except green
31    if cv2.contourArea(c) > 40000:
32        segmented = cv2.bitwise_and(image , image , mask=mask)
33        hsv = cv2.cvtColor(segmented , cv2.COLOR_BGR2HSV)
34        lower_green = np.array([27, 29, 0],np.uint8)
35        upper_green = np.array([179, 255, 165],np.uint8)
36        mask = cv2.inRange(hsv, lower_green , upper_green)
37        if test :
```

```

38         cv2.imshow("warunek", mask)
39         cv2.waitKey()
40         ret,thresh = cv2.threshold(mask,127,255,0)
41         contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
42         # set minArea = 5000
43         minArea = 5000
44     else :
45         # set minArea = 1000
46         minArea = 1000
47
48     # finding the generall shape of plant as the most center contour
49     # of the area > minArea
50     closest=1000
51     for c in contours :
52         if cv2.contourArea(c)>minArea:
53             M = cv2.moments(c)
54             cX = int(M["m10"] / M["m00"])
55             cY = int(M["m01"] / M["m00"])
56             dist=abs(width/2-cX)+abs(height/2-cY)
57             if closest > dist:
58                 closest=dist
59                 cnt=c
60
61     # take the general mask of the plant
62     mask = cv2.drawContours(np.zeros((height ,width ,3), np.uint8 ),
[cnt], 0, (255,255,255), cv2.FILLED)
63     mask = cv2.cvtColor(mask ,cv2.COLOR_BGR2GRAY)
64     if test :
65         cv2.imshow("Image", mask)
66         cv2.waitKey()
67
68     # take only part of the mask from the image
69     segmented = cv2.bitwise_and(image , image , mask=mask)
70     if test :
71         cv2.imshow("seg", segmented)
72         cv2.waitKey()
73
74     # more precisely remove all colors exept green
75     # only from the segmented part of the image
76     hsv = cv2.cvtColor(segmented , cv2.COLOR_BGR2HSV)
77     lower_brown = np.array([37, 31, 28],np.uint8)
78     upper_brown = np.array([86, 255, 134],np.uint8)
79     mask_seg = cv2.inRange(hsv, lower_brown , upper_brown)
80     if test :
81         cv2.imshow("Mask Seg", mask_seg)
82         cv2.waitKey()
83
84     # perform a series of erosions and dilations depends on the max
contour area
85     kernel22 = cv2.getStructuringElement(cv2.MORPH_RECT, (1,1))
86     mask_seg = cv2.morphologyEx(mask_seg, cv2.MORPH_OPEN , kernel22)
87     ret,thresh = cv2.threshold(mask_seg,127,255,0)
88     contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.
CHAIN_APPROX_SIMPLE)

```



```

89     # take the biggest contour
90     maxContour = sorted(contours, key = cv2.contourArea, reverse =
True)[0]
91     # set size of the contour devided by 25000
92     size = int(cv2.contourArea(maxContour)//25000)
93     # perform erosion and dilatation with size iteration
94     mask_seg = cv2.erode(mask_seg, None, iterations = size)
95     mask_seg = cv2.dilate(mask_seg, None, iterations = size)
96     if test :
97         cv2.imshow("Mask Closed", mask_seg)
98         cv2.waitKey()
99     # and return the mask of the plant
100     return mask_seg
101
102 # Debuging
103 # image_path = './multi_plant/rgb_00_01_000_04.png'
104 # cv2.imshow("Result",mask(image_path, True))
105 # cv2.waitKey()

```

4.2 boundingBoxes.py

```

1 import glob
2 import numpy as np
3 import cv2
4 import os
5 from main_code import main
6
7 pathToRead = './multi_plant'
8 pathToWrite = './'
9 directoryName = 'my_bounding_boxes'
10 directory = os.path.dirname(pathToWrite+directoryName)
11
12 def bounding_boxes(p):
13     mask = main(p,False)
14     image = cv2.imread(p,cv2.IMREAD_COLOR)
15     x,y,w,h = cv2.boundingRect(mask)
16     cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
17     return image
18
19 def save_image(name, image):
20     name = os.path.basename(name)
21     cv2.imwrite(pathToWrite+directoryName+'/' +name, image)
22
23 files = [f for f in glob.glob(pathToRead + "**/*.png", recursive=
True)]
24 if not os.path.exists(directory):
25     os.makedirs(directory)
26 i=0
27 try :
28     for f in files:
29         image = bounding_boxes(f)
30         save_image(f, image)
31         print(i/9)
32         i+=1
33         k = cv2.waitKey(5) & 0xFF

```

```

34         if k == 27:
35             break
36 except Exception as e:
37     print(e)
38     print(f)

```

4.3 labeling.py

```

1 import numpy as np
2 import cv2
3 from mask import mask
4
5 image_path = './multi_plant/rgb_00_03_000_00.png'
6
7 colors = [(255,0,0),(0,255,0),(255,255,0),(0,0,255),(255,0,255),
8           ,(0,255,255),(128,128,128),(255,128,128),(128,255,128),
9           ,(128,128,255)]
10
11 # additional fucntion for overlapping mask
12 # remove the are intersection of mask1 and mask2 from mask1
13 def removeBitwiseMask(mask1, mask2):
14     m2 = cv2.cvtColor(mask2,cv2.COLOR_BGR2GRAY)
15     ret,th2 = cv2.threshold(m2,0,255,cv2.THRESH_BINARY)
16     mask_inv = cv2.bitwise_not(th2)
17     mask1 = cv2.bitwise_and(mask1 , mask1 , mask=mask_inv)
18     return mask1
19
20 # the main function
21 # finding the mask for leaves
22 def label(p,test):
23     # read mask from mask fucntion
24     my_mask = mask(p,False)
25     if test :
26         cv2.imshow("Mask", my_mask)
27     height, width = my_mask.shape
28
29     # find the biggest contour
30     ret,thresh = cv2.threshold(my_mask,127,255,0)
31     contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
32     maxContour = sorted(contours, key = cv2.contourArea, reverse = True)[0]
33     maxContourArea = cv2.contourArea(maxContour)
34
35     #cases of erosion and dilatalion depends on the area
36     if maxContourArea < 7500 :
37         size = int(cv2.contourArea(maxContour)//10000) + 5
38         erod = size
39         dilat = 3 * size
40         minArea = 0
41     else:
42         if maxContourArea < 15000 :
43             size = int(cv2.contourArea(maxContour)//5000) + 13
44             erod = size
45             dilat = 2 * size

```

```

44         minArea = 0
45     else:
46         if maxContourArea < 35000:
47             size = int(cv2.contourArea(maxContour)//5000) + 10
48             erod = size
49             dilat = 2.2 * size
50             minArea = 500
51         else:
52             size = int(cv2.contourArea(maxContour)//10000) + 3
53             erod = size
54             dilat = 4.1 * size
55             minArea = 0
56     dilat = int(dilat)
57
58     # pefrome erosion
59     mask_erode = cv2.erode(my_mask, None, iterations = erod)
60     if test :
61         cv2.imshow("Mask Closed", mask_erode)
62         cv2.waitKey()
63
64     # take all contour in the mask_erode
65     ret,thresh = cv2.threshold(mask_erode,127,255,0)
66     contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.
CHAIN_APPROX_SIMPLE)
67     l = list()
68     iter=0
69
70     # sort the contours
71     contours = sorted(contours, key = cv2.contourArea, reverse=True)
72     for cnt in contours:
73         if cv2.contourArea(cnt)>minArea:
74             # make the mask of the contour and dilate it
75             m = cv2.drawContours(np.zeros((height ,width ,3), np.
uint8 ), [cnt], 0, colors[iter%(len(colors)+1)], cv2.FILLED)
76             m = cv2.dilate(m, None, iterations = dilat)
77             # segmented with my_mask
78             m = cv2.bitwise_and(m, m, mask=my_mask)
79             iter=iter+1
80             # add to list
81             l.append(m)
82
83     # for every mask in list
84     for i in range(0,len(l)-1):
85         #from bigger mask remove the intersection with the smaller
86         l[i] = removeBitwiseMask(l[i], l[i+1])
87
88     if test :
89         cv2.imshow("Sum(l)", sum(l))
90         cv2.waitKey()
91     #return the list of mask
92     return l
93
94 # cv2.imshow("Result",sum(label(image_path, True)))
95 # cv2.waitKey()

```

4.4 saveToFile.py

```
1 # import the necessary packages
2 import glob
3 import numpy as np
4 import cv2
5 import os
6 from mask import mask
7 from labeling import label
8
9 # paths and directories
10 pathToRead = './multi_plant'
11 pathToWrite = './'
12 directoryMask = 'my_masks'
13 directoryLabel = 'my_labels'
14 directoryBoxes = 'my_bounding_boxes'
15
16 # save mask
17 def saveMasks(name):
18     image = mask(name, False)
19     name = os.path.basename(name)
20     cv2.imwrite(pathToWrite+directoryMask+'/'+name, image)
21
22 #save label
23 def saveLabels(name):
24     image = sum(label(name, False))
25     name = os.path.basename(name)
26     cv2.imwrite(pathToWrite+directoryLabel+'/'+name.replace('rgb', 'label'), image)
27
28 #save bounding box
29 def saveBoxes(name):
30     mask = mask(name, False)
31     image = cv2.imread(name, cv2.IMREAD_COLOR)
32     x,y,w,h = cv2.boundingRect(mask)
33     cv2.rectangle(image, (x,y), (x+w,y+h), (0,255,0), 2)
34     name = os.path.basename(name)
35     cv2.imwrite(pathToWrite+directoryBoxes+'/'+name.replace('rgb', 'box'), image)
36
37 # read all files from the pathToRead
38 files = [f for f in glob.glob(pathToRead + "**/*.png", recursive=True)]
39
40 #if destination directory doesn't exist then create
41 if not os.path.exists(directoryMask):
42     os.makedirs(directoryMask)
43 if not os.path.exists(directoryLabel):
44     os.makedirs(directoryLabel)
45 if not os.path.exists(directoryBoxes):
46     os.makedirs(directoryBoxes)
47 # counter of progress
48 i=0
49 # try to catch exceptions
50 try :
```

```

51     # for every file in reading directory
52     for f in files:
53         # saveMasks(f)
54         # saveBoxes(f)
55         saveLabels(f)
56         # print the progress
57         print(i/9)
58         i+=1
59 # if there is any exception print the error and the path which rise
    the exception
60 except Exception as e:
61     print(e)
62     print(f)

```

4.5 mask.py

```

1  # import the necessary packages
2  import numpy as np
3  import cv2
4  import glob
5  import os
6  from skimage.measure import compare_ssim
7  from sklearn.metrics import jaccard_similarity_score
8
9  # paths and directories
10 pathToPatterns = './multi_label/'
11 pathToMyMasks = './my_masks/'
12 pathToResults = './ComparisionMasks/'
13
14 # read the pattern
15 def pattern(path):
16     image = cv2.imread(pathToPatterns+path)
17     image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
18     ret,th1 = cv2.threshold(image,0,255,cv2.THRESH_BINARY)
19     # return binary mask
20     return th1
21
22 # read my mask
23 def my_mask(path):
24     image = cv2.imread(pathToMyMasks+path.replace('label','rgb'))
25     image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
26     return image
27
28 # ssim coparison
29 def ssim_compare(my_mask, pat):
30     (score, diff) = compare_ssim(my_mask, pat, full=True)
31     diff = (diff * 255).astype("uint8")
32     return score
33
34 # jaccard comaprison by function
35 def jaccard_compare(my_mask, pat):
36     score = jaccard_similarity_score(my_mask.flatten(), pat.flatten())
37     return score
38

```

```

39 # IoU comparison
40 def IoU_compare(my_mask, pat):
41     img1 = np.asarray(pat).astype(np.bool)
42     img2 = np.asarray(my_mask).astype(np.bool)
43     num = np.sum(np.logical_and(img1, img2))
44     den = np.sum(np.logical_or(img1, img2))
45     return num/den
46
47 # dice coefficient comparison
48 def dice_coeff_compare(my_mask, pat):
49     img1 = np.asarray(pat).astype(np.bool)
50     img2 = np.asarray(my_mask).astype(np.bool)
51     img_intersection = np.bitwise_and(img1, img2)
52     image_sum = img1.sum() + img2.sum()
53     if image_sum == 0:
54         return 0
55     return 2. * img_intersection.sum() / image_sum
56
57 # read all files
58 files = [f for f in glob.glob(pathToPatterns + "**/*.png", recursive
    =True)]
59 # prepare the empty dictionaries
60 dice = dict()
61 ssim = dict()
62 jacc = dict()
63 iou = dict()
64
65
66 if not os.path.exists(pathToResults):
67     os.makedirs(pathToResults)
68
69 completeName = os.path.join(pathToResults, "Dice.txt")
70 file_dice = open(completeName, 'w')
71 completeName = os.path.join(pathToResults, "Ssim.txt")
72 file_ssim = open(completeName, 'w')
73 completeName = os.path.join(pathToResults, "Jacc.txt")
74 file_jacc = open(completeName, 'w')
75 completeName = os.path.join(pathToResults, "IoU.txt")
76 file_iou = open(completeName, 'w')
77
78 # set counter of progress
79 i=0
80 # try to catch exceptions
81 try :
82     # for every file in reading directory
83     for f in files:
84         name = os.path.basename(f)
85         pat = pattern(name)
86         my_m = my_mask(name)
87         name = name.replace('label', 'rgb')
88         ssim[name] = ssim_compare(my_m, pat)
89         jacc[name] = jaccard_compare(my_m, pat)
90         dice[name] = dice_coeff_compare(my_m, pat)
91         iou[name] = IoU_compare(my_m, pat)
92     # save to files

```

```

93     file_dice.write(str(name)+" "+str(dice[name]*100)+"\n")
94     file_jacc.write(str(name)+" "+str(jacc[name]*100)+"\n")
95     file_ssim.write(str(name)+" "+str(ssim[name]*100)+"\n")
96     file_iou.write(str(name)+" "+str(iou[name]*100)+"\n")
97     # print the progress
98     print(round(i/9,0))
99     i+=1
100
101     # compute the results for Jaccard Index
102     key_max = max(jacc.keys(), key=(lambda k: jacc[k]))
103     key_min = min(jacc.keys(), key=(lambda k: jacc[k]))
104     avg_value = np.array([jacc[key] for key in jacc]).mean()
105
106     # print the result for Jaccard Index
107     print("---Jaccard Index---")
108     print("average ", round(avg_value*100,2))
109     print("minimal value ", round(jacc[key_min]*100,2)," ",key_min)
110     print("maximal value ", round(jacc[key_max]*100,2)," ",key_max)
111
112     file_jacc.write("---Jaccard Index---\n")
113     file_jacc.write("average "+str(round(avg_value*100,2))+"\n")
114     file_jacc.write("minimal value "+str(round(jacc[key_min]*100,2))
115 + " "+str(key_min)+"\n")
116     file_jacc.write("maximal value "+str(round(jacc[key_max]*100,2))
117 + " "+str(key_max)+"\n")
118
119     # compute the results for IoU metric
120     key_max = max(iou.keys(), key=(lambda k: iou[k]))
121     key_min = min(iou.keys(), key=(lambda k: iou[k]))
122     avg_value = np.array([iou[key] for key in iou]).mean()
123
124     # print the result for IoU metric
125     print("--- IoU metric ---")
126     print("average ", round(avg_value*100,2))
127     print("minimal value ", round(iou[key_min]*100,2)," ",key_min)
128     print("maximal value ", round(iou[key_max]*100,2)," ",key_max)
129
130     file_iou.write("--- IoU metric ---\n")
131     file_iou.write("average "+str(round(avg_value*100,2))+"\n")
132     file_iou.write("minimal value "+str(round(iou[key_min]*100,2))+
133 +str(key_min)+"\n")
134     file_iou.write("maximal value "+str(round(iou[key_max]*100,2))+
135 +str(key_max)+"\n")
136
137     # compute the results for Dice coefficient
138     key_max = max(dice.keys(), key=(lambda k: dice[k]))
139     key_min = min(dice.keys(), key=(lambda k: dice[k]))
140     avg_value = np.array([dice[key] for key in dice]).mean()
141
142     # print the result for Dice coefficient
143     print("--- Dice coefficient ---")
144     print("average ", round(avg_value*100,2))
145     print("minimal value ", round(dice[key_min]*100,2)," ",key_min)
146     print("maximal value ", round(dice[key_max]*100,2)," ",key_max)

```

```

144     file_dice.write("--- Dice coefficient ----\n")
145     file_dice.write("average "+str(round(avg_value*100,2))+"\n")
146     file_dice.write("minimal value "+str(round(dice[key_min]*100,2))
147 + " "+str(key_min)+"\n")
148     file_dice.write("maximal value "+str(round(dice[key_max]*100,2))
149 + " "+str(key_max)+"\n")
150
151     # compute the results for Structural Similarity Index
152     key_max = max(ssim.keys(), key=(lambda k: ssim[k]))
153     key_min = min(ssim.keys(), key=(lambda k: ssim[k]))
154     avg_value = np.array([ssim[key] for key in ssim]).mean()
155
156     # print the result for Structural Similarity Index
157     print("---Structural Similarity Index---")
158     print("average ", round(avg_value*100,2))
159     print("minimal value ", round(ssim[key_min]*100,2)," ",key_min)
160     print("maximal value ", round(ssim[key_max]*100,2)," ",key_max)
161
162     file_ssim.write("---Structural Similarity Index---\n")
163     file_ssim.write("average "+str(round(avg_value*100,2))+"\n")
164     file_ssim.write("minimal value "+str(round(ssim[key_min]*100,2))
165 + " "+str(key_min)+"\n")
166     file_ssim.write("maximal value "+str(round(ssim[key_max]*100,2))
167 + " "+str(key_max)+"\n")
168
169     file_dice.close()
170     file_jacc.close()
171     file_ssim.close()
172     file_iou.close()
173
174 # if there is any exception print the error and the path which rise
175 the exception
176 except Exception as e:
177     print(e)
178     print(f)

```

4.6 labelComapison.py

```

1  # import the necessary packages
2  import numpy as np
3  import cv2
4  import glob
5  import os
6  from skimage.measure import compare_ssim
7  from sklearn.metrics import jaccard_similarity_score
8  from mask import mask
9
10 # paths and directories
11 pathToPatterns = './multi_label/'
12 pathToMyLabels = './my_labels/'
13 pathToResults = './ComparisionLabels/'
14
15 colors = [(255,0,0),(0,255,0),(255,255,0),(0,0,255),(255,0,255),
16           ,(0,255,255),(128,128,128),(128,0,0)]

```



```

17 # read the pattern
18 def pattern(path):
19     image = cv2.imread(pathToPatterns+path)
20     image = cv2.cvtColor(image, cv2.IMREAD_COLOR)
21     d = dict()
22     for c in colors:
23         arr = np.array(c)
24         res = cv2.inRange(image, arr, arr)
25         if np.sum(res) > 0:
26             segmented = cv2.bitwise_and(image, image, mask=res)
27             d[c] = segmented
28     return d
29
30 # read my mask
31 def my_label(path, keys):
32     image = cv2.imread(pathToMyLabels+path)
33     image = cv2.cvtColor(image, cv2.IMREAD_COLOR)
34     d = dict()
35     for c in keys:
36         arr = np.array(c)
37         res = cv2.inRange(image, arr, arr)
38         segmented = cv2.bitwise_and(image, image, mask=res)
39         d[c] = segmented
40     return d
41
42 # IoU comparison
43 def IoU_compare(my_mask, pat):
44     img1 = np.asarray(pat).astype(np.bool)
45     img2 = np.asarray(my_mask).astype(np.bool)
46     num = np.sum(np.bitwise_and(img1, img2))
47     den = np.sum(np.bitwise_or(img1, img2))
48     return num/den
49
50 # dice coefficient coparison
51 def dice_coeff_compare(my_mask, pat):
52     img1 = np.asarray(pat).astype(np.bool)
53     img2 = np.asarray(my_mask).astype(np.bool)
54     img_intersection = np.logical_and(img1, img2)
55     image_sum = img1.sum() + img2.sum()
56     if image_sum == 0:
57         return 0
58     return 2. * img_intersection.sum() / image_sum
59
60 # read all files
61 files = [f for f in glob.glob(pathToPatterns + "**/*.png", recursive
    =True)]
62 # prepare the empty dictionaries
63 dice = dict()
64 ssim = dict()
65 jacc = dict()
66 iou = dict()
67
68 if not os.path.exists(pathToResults):
69     os.makedirs(pathToResults)
70

```

```

71 completeName = os.path.join(pathToResults, "Dice.txt")
72 file_dice = open(completeName, 'w')
73 completeName = os.path.join(pathToResults, "IoU.txt")
74 file_iou = open(completeName, 'w')
75
76 # set counter of progress
77 i=0
78 # try to catch exceptions
79 try :
80     # for every file in reading directory
81     for f in files:
82         name = os.path.basename(f)
83         pat = pattern(name)
84         my_l = my_label(name, pat.keys())
85         s=0
86         j=0
87         d=0
88         u=0
89         for key in pat.keys():
90             dice[str(name)+str(key)] = dice_coeff_compare(my_l[key],
91 pat[key])
92             iou[str(name)+str(key)] = IoU_compare(my_l[key], pat[key])
93
94             d += dice[str(name)+str(key)]
95             u += iou[str(name)+str(key)]
96             file_dice.write(str(name)+" "+str(key)+" "+str(dice[str(
97 name)+str(key)]*100)+"\n")
98             file_iou.write(str(name)+" "+str(key)+" "+str(iou[str(
99 name)+str(key)]*100)+"\n")
100         dice[str(name)] = d/len(pat)
101         iou[str(name)] = u/len(pat)
102         file_dice.write(str(name)+" "+str(dice[str(name)]*100)+"\n")
103         file_iou.write(str(name)+" "+str(iou[str(name)]*100)+"\n")
104     # print the progress
105     print(round(i/9,0))
106     i+=1
107
108
109 # compute the results for IoU metric
110 key_max = max(iou.keys(), key=(lambda k: iou[k]))
111 key_min = min(iou.keys(), key=(lambda k: iou[k]))
112 avg_value = np.array([iou[key] for key in iou]).mean()
113
114 # print the result for IoU metric
115 print("--- IoU metric ----")
116 print("average ", round(avg_value*100,2))
117 print("minimal value ", round(iou[key_min]*100,2)," ",key_min)
118 print("maximal value ", round(iou[key_max]*100,2)," ",key_max)
119
120 file_iou.write("--- IoU metric ----\n")
121 file_iou.write("average "+str(round(avg_value*100,2))+"\n")
122 file_iou.write("minimal value "+str(round(iou[key_min]*100,2))+
123 "+str(key_min)+"\n")
124 file_iou.write("maximal value "+str(round(iou[key_max]*100,2))+
125 "+str(key_max)+"\n")

```

```

120 # compute the results for Dice coefficient
121 key_max = max(dice.keys(), key=(lambda k: dice[k]))
122 key_min = min(dice.keys(), key=(lambda k: dice[k]))
123 avg_value = np.array([dice[key] for key in dice]).mean()
124
125 # print the result for Dice coefficient
126 print("--- Dice coefficient ----")
127 print("average ", round(avg_value*100,2))
128 print("minimal value ", round(dice[key_min]*100,2)," ",key_min)
129 print("maximal value ", round(dice[key_max]*100,2)," ",key_max)
130
131 file_dice.write("--- Dice coefficient ----\n")
132 file_dice.write("average "+str(round(avg_value*100,2))+"\n")
133 file_dice.write("minimal value "+str(round(dice[key_min]*100,2))
134 +" "+str(key_min)+"\n")
135 file_dice.write("maximal value "+str(round(dice[key_max]*100,2))
136 +" "+str(key_max)+"\n")
137
138 # if there is any exception print the error and the path which rise
139 # the exception
140 except Exception as e:
141     print(e)
142     print(f)

```

4.7 test.py

```

1 import cv2
2 import numpy as np
3 from main_code import main
4
5 #cap = cv2.VideoCapture(0)
6
7 def nothing(x):
8     pass
9 # Creating a window for later use
10 cv2.namedWindow('result')
11
12 # Starting with 100's to prevent error while masking
13 h,s,v = 100,100,100
14
15 # image = cv2.imread('./multi_plant/rgb_01_02_008_00.png',cv2.
16     IMREAD_COLOR)
17 # height, width, channels = image.shape
18
19 # # remove all colors except of particular shades of green
20 # hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
21 # lower_green = np.array([31, 28, 51],np.uint8) #[30, 22, 22] 40 55
22     40
23 # upper_green = np.array([73, 104, 152],np.uint8) #[85, 235, 195]
24     120 130 110
25 # mask = cv2.inRange(hsv, lower_green , upper_green)
26
27 image = main('./multi_plant/rgb_00_00_009_02.png',False)
28

```

```

27 Lower=[0, 0, 0]
28 Upper=[179, 255, 165]
29
30
31
32 # Creating track bar
33 cv2.createTrackbar('h M', 'result',Upper[0],179,nothing)
34 cv2.createTrackbar('h L', 'result',Lower[0],179,nothing)
35 cv2.createTrackbar('s M', 'result',Upper[1],255,nothing)
36 cv2.createTrackbar('s L', 'result',Lower[1],255,nothing)
37 cv2.createTrackbar('v M', 'result',Upper[2],255,nothing)
38 cv2.createTrackbar('v L', 'result',Lower[2],255,nothing)
39 # cv2.createTrackbar('blur1', 'result',1,20,nothing)
40 # cv2.createTrackbar('blur2', 'result',1,20,nothing)
41
42
43 while(1):
44
45     #_, frame = cap.read()
46
47     #converting to HSV
48     hsv = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
49
50     # get info from track bar and apply to result
51     hM = cv2.getTrackbarPos('h M','result')
52     hL = cv2.getTrackbarPos('h L','result')
53     sM = cv2.getTrackbarPos('s M','result')
54     sL = cv2.getTrackbarPos('s L','result')
55     vM = cv2.getTrackbarPos('v M','result')
56     vL = cv2.getTrackbarPos('v L','result')
57     # b1 = cv2.getTrackbarPos('blur1','result')
58     # b2 = cv2.getTrackbarPos('blur2','result')
59
60     # Normal masking algorithm
61     lower = np.array([hL,sL,vL])
62     upper = np.array([hM,sM,vM])
63
64     res = cv2.inRange(hsv,lower, upper)
65     # kernel123 = cv2.getStructuringElement(cv2.MORPH_RECT, (b1,b2))
66     # res = cv2.morphologyEx(res , cv2.MORPH_OPEN , kernel123)
67     #res = cv2.medianBlur(res , b)
68     result = cv2.bitwise_and(image,image,mask = res)
69
70     cv2.imshow('result',result)
71
72     k = cv2.waitKey(5) & 0xFF
73     if k == 27:
74         print([hL,sL,vL])
75         print([hM,sM,vM])
76         break
77
78 #cap.release()
79
80 cv2.destroyAllWindows()

```

5 References

1. https://www.overleaf.com/learn/latex/Code_listing
2. https://en.wikipedia.org/wiki/Image_segmentation
3. <https://pythonprogramming.net/color-filter-python-opencv-tutorial/>
4. https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html
5. https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rects_circles/bounding_rects_circles.html