Introduction to image processing and computer vision

Plant Segmentation and Labeling

Laboratory Project I

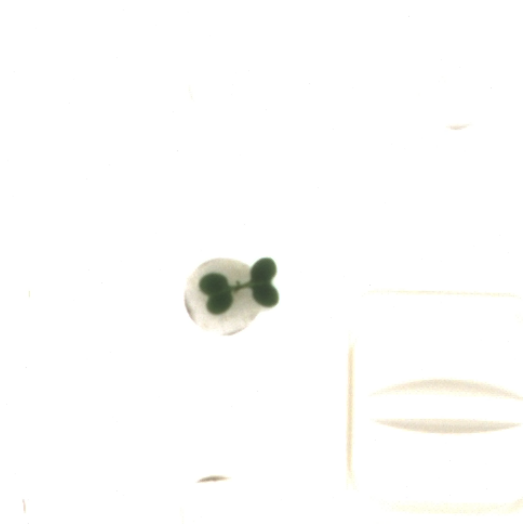
Patryk Walczak

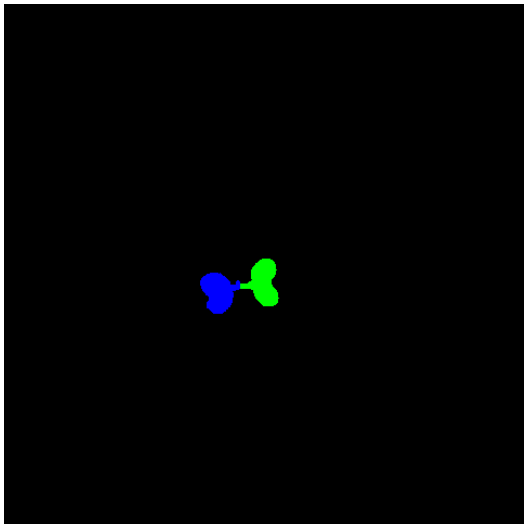
January 8, 2020

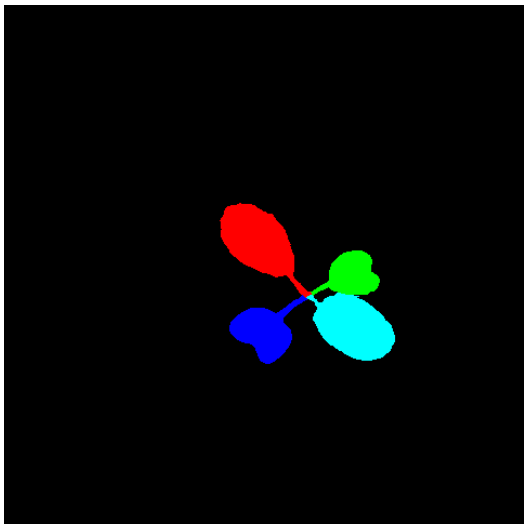
Contents

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.







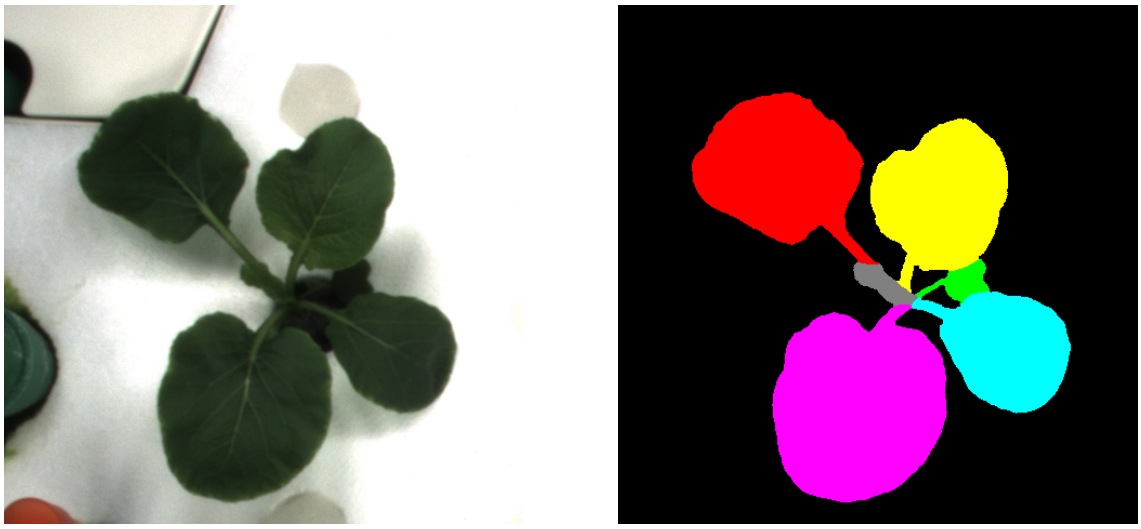


Figure 1: Some of the images and their patterns

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 :

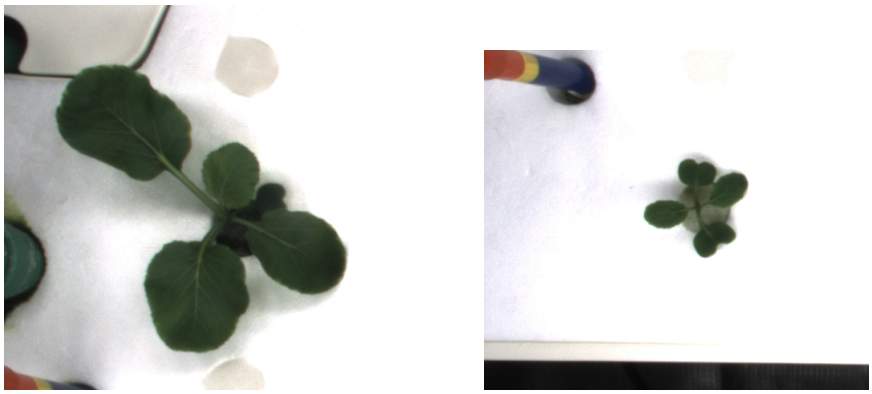
- mask.py - finding the best mask for each image
- boundingBoxes.py - using the mask to create the bounding box
- labeling.py - create the separate mask for each leaf
- saveToFile.py - save masks, labels and bounding boxes to files
- maskComparison.py - compare my masks with the expectations
- labelComapison.py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

```
    hsv = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)

    # get info from track bar and apply to result
    hM = cv2.getTrackbarPos('h M','result')
    hL = cv2.getTrackbarPos('h L','result')
    sM = cv2.getTrackbarPos('s M','result')
    sL = cv2.getTrackbarPos('s L','result')
    vM = cv2.getTrackbarPos('v M','result')
    vL = cv2.getTrackbarPos('v L','result')
    # b1 = cv2.getTrackbarPos('blur1','result')
    # b2 = cv2.getTrackbarPos('blur2','result')

    # Normal masking algorithm
    lower = np.array([hL,sL,vL])
    upper = np.array([hM,sM,vM])

    res = cv2.inRange(hsv,lower, upper)
```

```
hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
lower_green = np.array([0, 0, 0],np.uint8)
upper_green = np.array([179, 255, 165],np.uint8)
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.

```
# finding contur of the biggest area
ret,thresh = cv2.threshold(mask,127,255,0)
contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,\
cv2.CHAIN_APPROX_SIMPLE)
c = sorted(contours, key = cv2.contourArea, reverse = True)[0]

# if the biggest contour has the area > 40000 then remove more
    precisely
# all colors exept green
if cv2.contourArea(c) > 40000:
segmented = cv2.bitwise_and(image , image , mask=mask)
hsv = cv2.cvtColor(segmented , cv2.COLOR_BGR2HSV)
lower_green = np.array([27, 29, 0],np.uint8)
upper_green = np.array([179, 255, 165],np.uint8)
mask = cv2.inRange(hsv, lower_green , upper_green)
if test :
cv2.imshow("warunek", mask)
cv2.waitKey()
ret,thresh = cv2.threshold(mask,127,255,0)
contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
# set minArea = 5000
minArea = 5000
else :
# set minArea = 1000
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.

```
# finding the generall shape of plant as the most center contour
# of the area > minArea
closest=1000
for c in contours :
if cv2.contourArea(c)>minArea:
M = cv2.moments(c)
cX = int(M["m10"] / M["m00"])
cY = int(M["m01"] / M["m00"])
dist=abs(width/2-cX)+abs(height/2-cY)
if closest > dist:
closest=dist
cnt=c

# take the generall mask of the plant
mask = cv2.drawContours(np.zeros((height ,width ,3), np.uint8 ), [
    cnt], 0, (255,255,255), cv2.FILLED)
mask = cv2.cvtColor(mask ,cv2.COLOR_BGR2GRAY)
```

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

```
# take only part of the mask from the image
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.

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

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

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

Finally, the algorithm returns the mask.

```
# and return the mask of the plant
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.

```
def bounding_boxes(p):
mask = main(p,False)
image = cv2.imread(p,cv2.IMREAD_COLOR)
x,y,w,h = cv2.boundingRect(mask)
cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
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.

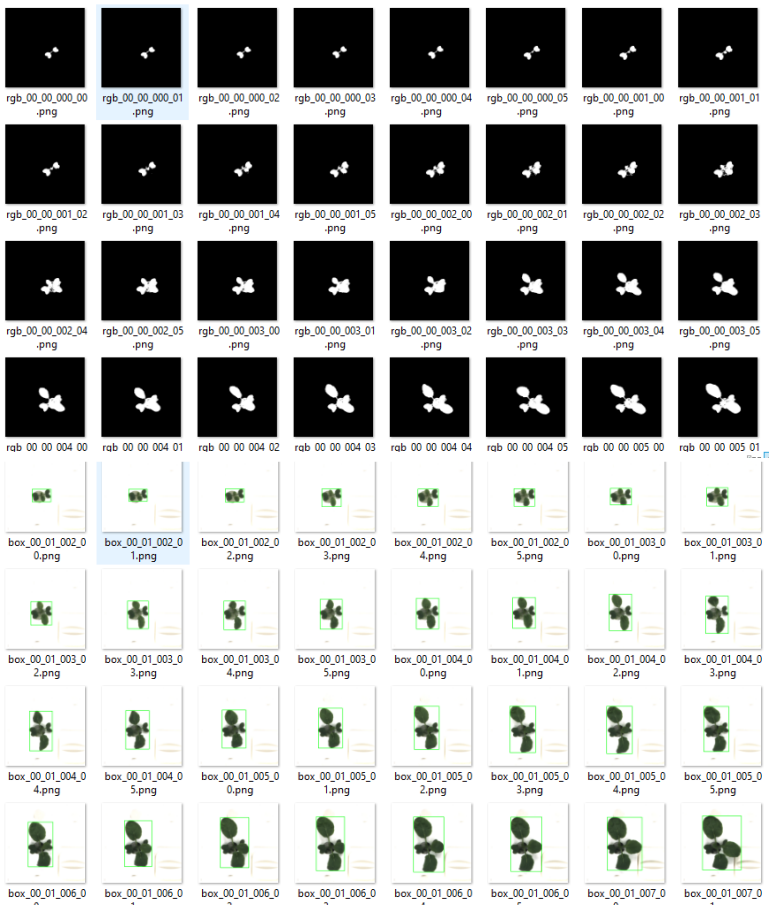
All result masks are in the directory "my_mask", and all result bounding boxes are in "my_bounding_boxes" directory.

```
directoryMask = 'my_masks'
directoryBoxes = 'my_bounding_boxes'

# save maske
def saveMasks(name):
image = mask(name, False)
name = os.path.basename(name)
```

```
cv2.imwrite(pathToWrite+directoryMask+'/'+name, image)

#save label
def saveLabels(name):
image = sum(label(name, False))
name = os.path.basename(name)
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:

```
# read the pattern
def pattern(path):
image = cv2.imread(pathToPatterns+path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret,th1 = cv2.threshold(image,0,255,cv2.THRESH_BINARY)
# return binary mask
return th1

# read my mask
def my_mask(path):
image = cv2.imread(pathToMyMasks+path.replace('label','rgb'))
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
return image
```

And the comparison functions look like:

```
# IoU comparison
def IoU_compare(my_mask,pat):
img1 = np.asarray(pat).astype(np.bool)
img2 = np.asarray(my_mask).astype(np.bool)
num = np.sum(np.logical_and(img1,img2))
den = np.sum(np.logical_or(img1,img2))
return num/den

# dice coefficient coparison
def dice_coeff_compare(my_mask, pat):
img1 = np.asarray(pat).astype(np.bool)
img2 = np.asarray(my_mask).astype(np.bool)
img_intersection = np.bitwise_and(img1, img2)
image_sum = img1.sum() + img2.sum()
if image_sum == 0:
return 0
return 2. * img_intersection.sum() / image_sum

# ssim coparison
def ssim_compare(my_mask, pat):
(score, diff) = compare_ssim(my_mask, pat, full=True)
diff = (diff * 255).astype("uint8")
return score

# jaccard comaprison by function
def jaccard_compare(my_mask, pat):
score = jaccard_similarity_score(my_mask.flatten(), pat.flatten())
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.

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

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

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

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

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

Then erosion is performed.

```
# pefrome erosion
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.

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

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

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

Finally, it returns the list of the masks.

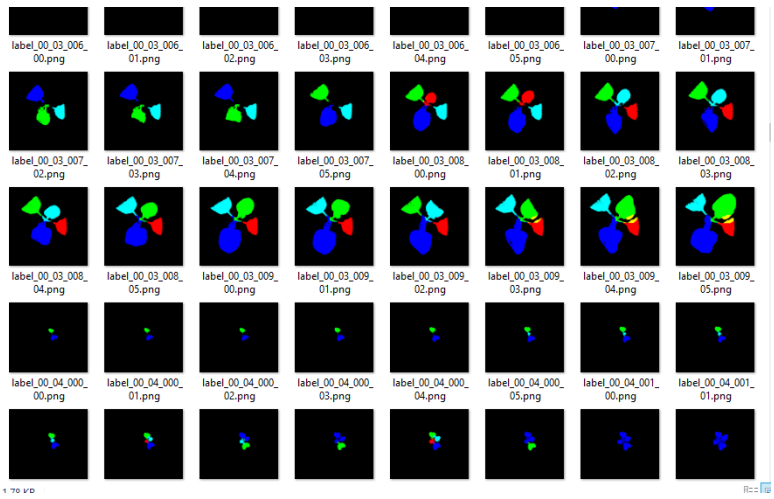
```
#return the list of mask
return l
```

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:

```
#save label
def saveLabels(name):
image = sum(label(name, False))
name = os.path.basename(name)
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.

```
# read the pattern
def pattern(path):
image = cv2.imread(pathToPatterns+path)
image = cv2.cvtColor(image, cv2.IMREAD_COLOR)
d = dict()
for c in colors:
arr = np.array(c)
res = cv2.inRange(image, arr, arr)
if np.sum(res) > 0:
segmented = cv2.bitwise_and(image , image , mask=res)
d[c] = segmented
return d

# read my mask
def my_label(path, keys):
image = cv2.imread(pathToMyLabels+path)
image = cv2.cvtColor(image, cv2.IMREAD_COLOR)
d = dict()
for c in keys:
arr = np.array(c)
res = cv2.inRange(image, arr, arr)
segmented = cv2.bitwise_and(image , image , mask=res)
d[c] = segmented
return d
```

Comparison functions are the same as in mask comparison

```
# IoU comparison
def IoU_compare(my_mask,pat):
img1 = np.asarray(pat).astype(np.bool)
img2 = np.asarray(my_mask).astype(np.bool)
num = np.sum(np.bitwise_and(img1,img2))
den = np.sum(np.bitwise_or(img1,img2))
return num/den

# dice coefficient coparision
def dice_coeff_compare(my_mask, pat):
img1 = np.asarray(pat).astype(np.bool)
img2 = np.asarray(my_mask).astype(np.bool)
img_intersection = np.logical_and(img1, img2)
image_sum = img1.sum() + img2.sum()
if image_sum == 0:
return 0
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.

```
for key in pat.keys():
dice[str(name)+str(key)] = dice_coeff_compare(my_l[key], pat[key])
iou[str(name)+str(key)] = IoU_compare(my_l[key], pat[key])
d += dice[str(name)+str(key)]
u += iou[str(name)+str(key)]
file_dice.write(str(name)+" "+str(key)+" "+str(dice[str(name)+str(
    key)]*100)+"\n")
file_iou.write(str(name)+" "+str(key)+" "+str(iou[str(name)+str(key)
    ]*100)+"\n")
dice[str(name)] = d/len(pat)
iou[str(name)] = u/len(pat)
file_dice.write(str(name)+" "+str(dice[str(name)]*100)+"\n")
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 ale 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 plats 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 prerfect, but much closer to that.

4 Source Code

4.1 mask.py

```
# import the necessary packages
import numpy as np
import cv2
import random
import os
from math import sqrt as sqrt

def mask(path, test):
    # load the image
    image = cv2.imread(path,cv2.IMREAD_COLOR)
    height, width, channels = image.shape
    if test :
        cv2.imshow("Image", image)
        cv2.waitKey()

    # remove generally all colors exept shades of green
    hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
    lower_green = np.array([0, 0, 0],np.uint8)
    upper_green = np.array([179, 255, 165],np.uint8)
    mask = cv2.inRange(hsv, lower_green , upper_green)
    if test :
        cv2.imshow("Image", mask)
        cv2.waitKey()

    # finding contur of the biggest area
    ret,thresh = cv2.threshold(mask,127,255,0)
    contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.
    CHAIN_APPROX_SIMPLE)
    c = sorted(contours, key = cv2.contourArea, reverse = True)[0]

    # if the biggest contur has the area > 40000 then remove more
    precisely all colors except green
    if cv2.contourArea(c) > 40000:
        segmented = cv2.bitwise_and(image , image , mask=mask)
        hsv = cv2.cvtColor(segmented , cv2.COLOR_BGR2HSV)
        lower_green = np.array([27, 29, 0],np.uint8)
        upper_green = np.array([179, 255, 165],np.uint8)
        mask = cv2.inRange(hsv, lower_green , upper_green)
        if test :
```

```
            cv2.imshow("warunek", mask)
            cv2.waitKey()
        ret,thresh = cv2.threshold(mask,127,255,0)
        contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,
    cv2.CHAIN_APPROX_SIMPLE)
        # set minArea = 5000
        minArea = 5000
    else :
        # set minArea = 1000
        minArea = 1000

    # finding the generall shape of plant as the most center contour
    # of the area > minArea
    closest=1000
    for c in contours :
        if cv2.contourArea(c)>minArea:
            M = cv2.moments(c)
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
            dist=abs(width/2-cX)+abs(height/2-cY)
            if closest > dist:
                closest=dist
                cnt=c

    # take the general mask of the plant
    mask = cv2.drawContours(np.zeros((height ,width ,3), np.uint8 ),
    [cnt], 0, (255,255,255), cv2.FILLED)
    mask = cv2.cvtColor(mask ,cv2.COLOR_BGR2GRAY)
    if test :
        cv2.imshow("Image", mask)
        cv2.waitKey()

    # take only part of the mask from the image
    segmented = cv2.bitwise_and(image , image , mask=mask)
    if test :
        cv2.imshow("seg", segmented)
        cv2.waitKey()

    # more precisely remove all colors exept green
    # only from the segmented part of the image
    hsv = cv2.cvtColor(segmented , cv2.COLOR_BGR2HSV)
    lower_brown = np.array([37, 31, 28],np.uint8)
    upper_brown = np.array([86, 255, 134],np.uint8)
    mask_seg = cv2.inRange(hsv, lower_brown , upper_brown)
    if test :
        cv2.imshow("Mask Seg", mask_seg)
        cv2.waitKey()

    # perform a series of erosions and dilations depends on the max
    contour area
    kernel22 = cv2.getStructuringElement(cv2.MORPH_RECT, (1,1))
    mask_seg = cv2.morphologyEx(mask_seg, cv2.MORPH_OPEN , kernel22)
    ret,thresh = cv2.threshold(mask_seg,127,255,0)
    contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.
    CHAIN_APPROX_SIMPLE)
```

```
    # take the biggest contour
    maxContour = sorted(contours, key = cv2.contourArea, reverse =
    True)[0]
    # set size of the contour devided by 25000
    size = int(cv2.contourArea(maxContour)//25000)
    # perform erosion and dilatation with size iteration
    mask_seg = cv2.erode(mask_seg, None, iterations = size)
    mask_seg = cv2.dilate(mask_seg, None, iterations = size)
    if test :
        cv2.imshow("Mask  Closed", mask_seg)
        cv2.waitKey()
    # and return the mask of the plant
    return mask_seg

# Debuging
# image_path = './multi_plant/rgb_00_01_000_04.png'
# cv2.imshow("Result",mask(image_path, True))
# cv2.waitKey()
```

4.2 boundingBoxes.py

```
import glob
import numpy as np
import cv2
import os
from main_code import main

pathToRead = './multi_plant'
pathToWrite = './'
directoryName = 'my_bounding_boxes'
directory = os.path.dirname(pathToWrite+directoryName)

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

def save_image(name, image):
    name = os.path.basename(name)
    cv2.imwrite(pathToWrite+directoryName+'/'+name, image)

files = [f for f in glob.glob(pathToRead + "**/*.png", recursive=
    True)]
if not os.path.exists(directory):
    os.makedirs(directory)
i=0
try :
    for f in files:
        image = bounding_boxes(f)
        save_image(f, image)
        print(i/9)
        i+=1
        k = cv2.waitKey(5) & 0xFF
```

```
        if k == 27:
            break
except Exception as e:
        print(e)
        print(f)
```

4.3 labeling.py

```
import numpy as np
import cv2
from mask import mask

image_path = './multi_plant/rgb_00_03_000_00.png'

colors = [(255,0,0),(0,255,0),(255,255,0),(0,0,255),(255,0,255)
    ,(0,255,255),(128,128,128),(255,128,128),(128,255,128)
    ,(128,128,255)]

# additional fucntion for overlapping mask
# remove the are intersection of mask1 and mask2 from mask1
def removeBitwiseMask(mask1, mask2):
    m2 =  cv2.cvtColor(mask2,cv2.COLOR_BGR2GRAY)
    ret,th2 = cv2.threshold(m2,0,255,cv2.THRESH_BINARY)
    mask_inv = cv2.bitwise_not(th2)
    mask1 = cv2.bitwise_and(mask1 , mask1 , mask=mask_inv)
    return mask1

# the main function
# finding the mask for leaves
def label(p,test):
    # read mask from mask fucntion
    my_mask = mask(p,False)
    if test :
        cv2.imshow("Mask", my_mask)
    height, width = my_mask.shape

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

    #cases of erosion and dilatalion depends on the area
    if maxContourArea < 7500 :
        size = int(cv2.contourArea(maxContour)//10000) + 5
        erod = size
        dilat = 3 * size
        minArea = 0
    else:
        if maxContourArea < 15000 :
            size = int(cv2.contourArea(maxContour)//5000) + 13
            erod = size
            dilat = 2 * size
```

```
            minArea = 0
        else:
            if maxContourArea < 35000:
                size = int(cv2.contourArea(maxContour)//5000) + 10
                erod = size
                dilat = 2.2 * size
                minArea = 500
            else:
                size = int(cv2.contourArea(maxContour)//10000) + 3
                erod = size
                dilat = 4.1 * size
                minArea = 0
    dilat = int(dilat)

    # pefrome erosion
    mask_erode = cv2.erode(my_mask, None, iterations = erod)
    if test :
        cv2.imshow("Mask  Closed", mask_erode)
        cv2.waitKey()

    # take all contour in the mask_erode
    ret,thresh = cv2.threshold(mask_erode,127,255,0)
    contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.
    CHAIN_APPROX_SIMPLE)
    l = list()
    iter=0

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

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

    if test :
        cv2.imshow("Sum(l)", sum(l))
        cv2.waitKey()
    #return the list of mask
    return l

# cv2.imshow("Result",sum(label(image_path, True)))
# cv2.waitKey()
```

4.4 saveToFile.py

```
# import the necessary packages
import glob
import numpy as np
import cv2
import os
from mask import mask
from labeling import label

# paths and directories
pathToRead = './multi_plant'
pathToWrite = './'
directoryMask = 'my_masks'
directoryLabel = 'my_labels'
directoryBoxes = 'my_bounding_boxes'

# save mask
def saveMasks(name):
    image = mask(name, False)
    name = os.path.basename(name)
    cv2.imwrite(pathToWrite+directoryMask+'/'+name, image)

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

#save bounding box
def saveBoxes(name):
    mask = mask(name,False)
    image = cv2.imread(name,cv2.IMREAD_COLOR)
    x,y,w,h = cv2.boundingRect(mask)
    cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
    name = os.path.basename(name)
    cv2.imwrite(pathToWrite+directoryBoxes+'/'+name.replace('rgb','
    box'), image)

# read all files from the pathToRead
files = [f for f in glob.glob(pathToRead + "**/*.png", recursive=
    True)]

#if destination directory doesn't exist then create
if not os.path.exists(directoryMask):
    os.makedirs(directoryMask)
if not os.path.exists(directoryLabel):
    os.makedirs(directoryLabel)
if not os.path.exists(directoryBoxes):
    os.makedirs(directoryBoxes)
# counter of progress
i=0
# try to catch exceptions
try :
```

```
    # for every file in reading directory
    for f in files:
        # saveMasks(f)
        # saveBoxes(f)
        saveLabels(f)
        # print the progress
        print(i/9)
        i+=1
# if there is any exception print the error and the path which rise
    the exception
except Exception as e:
        print(e)
        print(f)
```

4.5 mask.py

```
# import the necessary packages
import numpy as np
import cv2
import glob
import os
from skimage.measure import compare_ssim
from sklearn.metrics import jaccard_similarity_score

# paths and directories
pathToPatterns = './multi_label/'
pathToMyMasks = './my_masks/'
pathToResults = './ComparisionMasks/'

# read the pattern
def pattern(path):
    image = cv2.imread(pathToPatterns+path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret,th1 = cv2.threshold(image,0,255,cv2.THRESH_BINARY)
    # return binary mask
    return th1

# read my mask
def my_mask(path):
    image = cv2.imread(pathToMyMasks+path.replace('label','rgb'))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return image

# ssim coparison
def ssim_compare(my_mask, pat):
    (score, diff) = compare_ssim(my_mask, pat, full=True)
    diff = (diff * 255).astype("uint8")
    return score

# jaccard comaprison by function
def jaccard_compare(my_mask, pat):
    score = jaccard_similarity_score(my_mask.flatten(), pat.flatten
    ())
    return score

```

```
# IoU comparison
def IoU_compare(my_mask,pat):
    img1 = np.asarray(pat).astype(np.bool)
    img2 = np.asarray(my_mask).astype(np.bool)
    num = np.sum(np.logical_and(img1,img2))
    den = np.sum(np.logical_or(img1,img2))
    return num/den

# dice coefficient coparison
def dice_coeff_compare(my_mask, pat):
    img1 = np.asarray(pat).astype(np.bool)
    img2 = np.asarray(my_mask).astype(np.bool)
    img_intersection = np.bitwise_and(img1, img2)
    image_sum = img1.sum() + img2.sum()
    if image_sum == 0:
        return 0
    return 2. * img_intersection.sum() / image_sum

# read all files
files = [f for f in glob.glob(pathToPatterns + "**/*.png", recursive
   =True)]
# prepare the empty dictionaries
dice = dict()
ssim = dict()
jacc = dict()
iou = dict()


if not os.path.exists(pathToResults):
    os.makedirs(pathToResults)

completeName = os.path.join(pathToResults, "Dice.txt")
file_dice = open(completeName , 'w')
completeName = os.path.join(pathToResults, "Ssim.txt")
file_ssim = open(completeName , 'w')
completeName = os.path.join(pathToResults, "Jacc.txt")
file_jacc = open(completeName , 'w')
completeName = os.path.join(pathToResults, "IoU.txt")
file_iou = open(completeName , 'w')

# set counter of progress
i=0
# try to catch exceptions
try :
    # for every file in reading directory
    for f in files:
        name = os.path.basename(f)
        pat = pattern(name)
        my_m = my_mask(name)
        name = name.replace('label','rgb')
        ssim[name] = ssim_compare(my_m, pat)
        jacc[name] = jaccard_compare(my_m, pat)
        dice[name] = dice_coeff_compare(my_m, pat)
        iou[name] = IoU_compare(my_m, pat)
        # save to files
```

```
        file_dice.write(str(name)+" "+str(dice[name]*100)+"\n")
        file_jacc.write(str(name)+" "+str(jacc[name]*100)+"\n")
        file_ssim.write(str(name)+" "+str(ssim[name]*100)+"\n")
        file_iou.write(str(name)+" "+str(iou[name]*100)+"\n")
        # print the progress
        print(round(i/9,0))
        i+=1

    # compute the results for Jaccard Index
    key_max = max(jacc.keys(), key=(lambda k: jacc[k]))
    key_min = min(jacc.keys(), key=(lambda k: jacc[k]))
    avg_value = np.array([jacc[key] for key in jacc]).mean()

    # print the result for Jaccard Index
    print("---Jaccard Index----")
    print("average ", round(avg_value*100,2))
    print("minimal value ", round(jacc[key_min]*100,2)," ",key_min)
    print("maximal value ", round(jacc[key_max]*100,2)," ",key_max)

    file_jacc.write("---Jaccard Index----\n")
    file_jacc.write("average "+str(round(avg_value*100,2))+"\n")
    file_jacc.write("minimal value "+str(round(jacc[key_min]*100,2))
+" "+str(key_min)+"\n")
    file_jacc.write("maximal value "+str(round(jacc[key_max]*100,2))
+" "+str(key_max)+"\n")

    # compute the results for IoU metric
    key_max = max(iou.keys(), key=(lambda k: iou[k]))
    key_min = min(iou.keys(), key=(lambda k: iou[k]))
    avg_value = np.array([iou[key] for key in iou]).mean()

    # print the result for IoU metric
    print("--- IoU metric ----")
    print("average ", round(avg_value*100,2))
    print("minimal value ", round(iou[key_min]*100,2)," ",key_min)
    print("maximal value ", round(iou[key_max]*100,2)," ",key_max)

    file_iou.write("--- IoU metric ----\n")
    file_iou.write("average "+str(round(avg_value*100,2))+"\n")
    file_iou.write("minimal value "+str(round(iou[key_min]*100,2))+"
    "+str(key_min)+"\n")
    file_iou.write("maximal value "+str(round(iou[key_max]*100,2))+"
    "+str(key_max)+"\n")

    # compute the results for Dice coefficient
    key_max = max(dice.keys(), key=(lambda k: dice[k]))
    key_min = min(dice.keys(), key=(lambda k: dice[k]))
    avg_value = np.array([dice[key] for key in dice]).mean()

    # print the result for Dice coefficient
    print("--- Dice coefficient ----")
    print("average ", round(avg_value*100,2))
    print("minimal value ", round(dice[key_min]*100,2)," ",key_min)
    print("maximal value ", round(dice[key_max]*100,2)," ",key_max)

```

```
    file_dice.write("--- Dice coefficient ----\n")
    file_dice.write("average "+str(round(avg_value*100,2))+"\n")
    file_dice.write("minimal value "+str(round(dice[key_min]*100,2))
   +" "+str(key_min)+"\n")
    file_dice.write("maximal value "+str(round(dice[key_max]*100,2))
   +" "+str(key_max)+"\n")

    # compute the results for Structural Similarity Index
    key_max = max(ssim.keys(), key=(lambda k: ssim[k]))
    key_min = min(ssim.keys(), key=(lambda k: ssim[k]))
    avg_value = np.array([ssim[key] for key in ssim]).mean()

    # print the result for Structural Similarity Index
    print("---Structural Similarity Index----")
    print("average ", round(avg_value*100,2))
    print("minimal value ", round(ssim[key_min]*100,2)," ",key_min)
    print("maximal value ", round(ssim[key_max]*100,2)," ",key_max)

    file_ssim.write("---Structural Similarity Index----\n")
    file_ssim.write("average "+str(round(avg_value*100,2))+"\n")
    file_ssim.write("minimal value "+str(round(ssim[key_min]*100,2))
   +" "+str(key_min)+"\n")
    file_ssim.write("maximal value "+str(round(ssim[key_max]*100,2))
   +" "+str(key_max)+"\n")

    file_dice.close()
    file_jacc.close()
    file_ssim.close()
    file_iou.close()

# if there is any exception print the error and the path which rise
   the exception
except Exception as e:
        print(e)
        print(f)
```

4.6 labelComapison.py

```
# import the necessary packages
import numpy as np
import cv2
import glob
import os
from skimage.measure import compare_ssim
from sklearn.metrics import jaccard_similarity_score
from mask import mask

# paths and directories
pathToPatterns = './multi_label/'
pathToMyLabels = './my_labels/'
pathToResults = './ComparisionLabels/'

colors = [(255,0,0),(0,255,0),(255,255,0),(0,0,255),(255,0,255)
   ,(0,255,255),(128,128,128),(128,0,0)]

```

```
# read the pattern
def pattern(path):
    image = cv2.imread(pathToPatterns+path)
    image = cv2.cvtColor(image, cv2.IMREAD_COLOR)
    d = dict()
    for c in colors:
        arr = np.array(c)
        res = cv2.inRange(image, arr, arr)
        if np.sum(res) > 0:
            segmented = cv2.bitwise_and(image , image , mask=res)
            d[c] = segmented
    return d

# read my mask
def my_label(path, keys):
    image = cv2.imread(pathToMyLabels+path)
    image = cv2.cvtColor(image, cv2.IMREAD_COLOR)
    d = dict()
    for c in keys:
        arr = np.array(c)
        res = cv2.inRange(image, arr, arr)
        segmented = cv2.bitwise_and(image , image , mask=res)
        d[c] = segmented
    return d

# IoU comparison
def IoU_compare(my_mask,pat):
    img1 = np.asarray(pat).astype(np.bool)
    img2 = np.asarray(my_mask).astype(np.bool)
    num = np.sum(np.bitwise_and(img1,img2))
    den = np.sum(np.bitwise_or(img1,img2))
    return num/den

# dice coefficient coparison
def dice_coeff_compare(my_mask, pat):
    img1 = np.asarray(pat).astype(np.bool)
    img2 = np.asarray(my_mask).astype(np.bool)
    img_intersection = np.logical_and(img1, img2)
    image_sum = img1.sum() + img2.sum()
    if image_sum == 0:
        return 0
    return 2. * img_intersection.sum() / image_sum

# read all files
files = [f for f in glob.glob(pathToPatterns + "**/*.png", recursive
    =True)]
# prepare the empty dictionaries
dice = dict()
ssim = dict()
jacc = dict()
iou = dict()

if not os.path.exists(pathToResults):
    os.makedirs(pathToResults)

```

```
completeName = os.path.join(pathToResults, "Dice.txt")
file_dice = open(completeName , 'w')
completeName = os.path.join(pathToResults, "IoU.txt")
file_iou = open(completeName , 'w')

# set counter of progress
i=0
# try to catch exceptions
try :
    # for every file in reading directory
    for f in files:
        name = os.path.basename(f)
        pat = pattern(name)
        my_l = my_label(name, pat.keys())
        s=0
        j=0
        d=0
        u=0
        for key in pat.keys():
            dice[str(name)+str(key)] = dice_coeff_compare(my_l[key], pat[key])
            iou[str(name)+str(key)] = IoU_compare(my_l[key], pat[key])
            d += dice[str(name)+str(key)]
            u += iou[str(name)+str(key)]
            file_dice.write(str(name)+" "+str(key)+" "+str(dice[str(name)+str(key)]*100)+"\n")
            file_iou.write(str(name)+" "+str(key)+" "+str(iou[str(name)+str(key)]*100)+"\n")
        dice[str(name)] = d/len(pat)
        iou[str(name)] = u/len(pat)
        file_dice.write(str(name)+" "+str(dice[str(name)]*100)+"\n")
        file_iou.write(str(name)+" "+str(iou[str(name)]*100)+"\n")
        # print the progress
        print(round(i/9,0))
        i+=1

    # compute the results for IoU metric
    key_max = max(iou.keys(), key=(lambda k: iou[k]))
    key_min = min(iou.keys(), key=(lambda k: iou[k]))
    avg_value = np.array([iou[key] for key in iou]).mean()

    # print the result for IoU metric
    print("--- IoU metric ----")
    print("average ", round(avg_value*100,2))
    print("minimal value ", round(iou[key_min]*100,2)," ",key_min)
    print("maximal value ", round(iou[key_max]*100,2)," ",key_max)

    file_iou.write("--- IoU metric ----\n")
    file_iou.write("average "+str(round(avg_value*100,2))+"\n")
    file_iou.write("minimal value "+str(round(iou[key_min]*100,2))+" "+str(key_min)+"\n")
    file_iou.write("maximal value "+str(round(iou[key_max]*100,2))+" "+str(key_max)+"\n")

```

```
    # compute the results for Dice coefficient
    key_max = max(dice.keys(), key=(lambda k: dice[k]))
    key_min = min(dice.keys(), key=(lambda k: dice[k]))
    avg_value = np.array([dice[key] for key in dice]).mean()

    # print the result for Dice coefficient
    print("--- Dice coefficient ----")
    print("average ", round(avg_value*100,2))
    print("minimal value ", round(dice[key_min]*100,2)," ",key_min)
    print("maximal value ", round(dice[key_max]*100,2)," ",key_max)

    file_dice.write("--- Dice coefficient ----\n")
    file_dice.write("average "+str(round(avg_value*100,2))+"\n")
    file_dice.write("minimal value "+str(round(dice[key_min]*100,2))
   +" "+str(key_min)+"\n")
    file_dice.write("maximal value "+str(round(dice[key_max]*100,2))
   +" "+str(key_max)+"\n")

# if there is any exception print the error and the path which rise
   the exception
except Exception as e:
        print(e)
        print(f)
```

4.7 test.py

```
import cv2
import numpy as np
from main_code import main

#cap = cv2.VideoCapture(0)

def nothing(x):
    pass
# Creating a window for later use
cv2.namedWindow('result')

# Starting with 100's to prevent error while masking
h,s,v = 100,100,100

# image = cv2.imread('./multi_plant/rgb_01_02_008_00.png',cv2.
   IMREAD_COLOR)
# height, width, channels = image.shape


# # remove all colors except of particular shades of green
# hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
# lower_green = np.array([31, 28, 51],np.uint8) #[30, 22, 22]  40 55
    40
# upper_green = np.array([73, 104, 152],np.uint8) #[85, 235, 195]
   120 130 110
# mask = cv2.inRange(hsv, lower_green , upper_green)

image = main('./multi_plant/rgb_00_00_009_02.png',False)

```

```
Lower=[0, 0, 0]
Upper=[179, 255, 165]



# Creating track bar
cv2.createTrackbar('h M', 'result',Upper[0],179,nothing)
cv2.createTrackbar('h L', 'result',Lower[0],179,nothing)
cv2.createTrackbar('s M', 'result',Upper[1],255,nothing)
cv2.createTrackbar('s L', 'result',Lower[1],255,nothing)
cv2.createTrackbar('v M', 'result',Upper[2],255,nothing)
cv2.createTrackbar('v L', 'result',Lower[2],255,nothing)
# cv2.createTrackbar('blur1', 'result',1,20,nothing)
# cv2.createTrackbar('blur2', 'result',1,20,nothing)


while(1):

    #_, frame = cap.read()

    #converting to HSV
    hsv = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)

    # get info from track bar and apply to result
    hM = cv2.getTrackbarPos('h M','result')
    hL = cv2.getTrackbarPos('h L','result')
    sM = cv2.getTrackbarPos('s M','result')
    sL = cv2.getTrackbarPos('s L','result')
    vM = cv2.getTrackbarPos('v M','result')
    vL = cv2.getTrackbarPos('v L','result')
    # b1 = cv2.getTrackbarPos('blur1','result')
    # b2 = cv2.getTrackbarPos('blur2','result')

    # Normal masking algorithm
    lower = np.array([hL,sL,vL])
    upper = np.array([hM,sM,vM])

    res = cv2.inRange(hsv,lower, upper)
    # kernel123 = cv2.getStructuringElement(cv2.MORPH_RECT, (b1,b2))
    # res = cv2.morphologyEx(res , cv2.MORPH_OPEN , kernel123)
    #res = cv2.medianBlur(res , b)
    result = cv2.bitwise_and(image,image,mask = res)

    cv2.imshow('result',result)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        print([hL,sL,vL])
        print([hM,sM,vM])
        break

#cap.release()

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