המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

Technion - Israel Institute of Technology

CORN PLANT SEGMENTATION

CS 234329 - Image Processing and Analysis Project

Snir Homsky and Iliya Rubinchik

Supervisors: Alon Zvirin and Yaron Honen

March 18th, 2020

Table of Contents

Abstract

Mask R-CNN is a conceptually simple, flexible, and general framework for object instance segmentation which efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance. This Project tackles the task of corn plant segmentation given a partially annotated small dataset using Mask R-CNN. We present a workflow for producing a large dataset from a small given dataset, which includes image augmentation, generation of artificial plant images, and generation of artificial images simulating real greenhouse scenes. Finally, we present results on leaf segmentation as well as whole plant segmentation, and discuss these results.

Introduction

In this project we were tasked with creating a good segmentation for corn leaves and corn plants, identifying leaf order from oldest to youngest in each plant, and finding the start and end points of each leaf. The plants that were given to us were in different stages of growth, some were young and some were older, giving them different appearance.

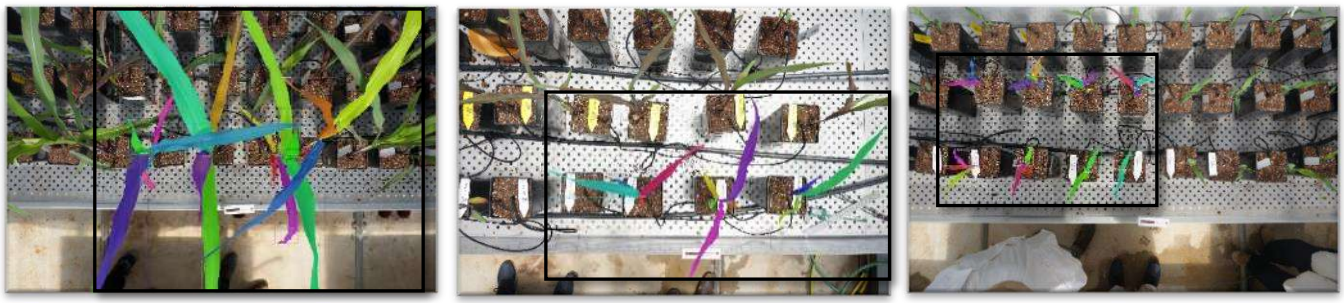
We used Mask R-CNN, a neural network known to perform well in instance segmentation. Mask R-CNN shows top results in all three tracks of the COCO suite of challenges, including instance segmentation, bounding box object detection, and person keypoint detection. Mask R-CNN outperforms all existing, single-model entries on every task, including the COCO 2016 challenge winners.

Our work consisted of generating a large set of artificial images simulating real greenhouse scenes and using it to train the neural network. We used TensorFlow - a platform for deep learning programming, Keras - a neural network library which allows to generate batches of tensor image data with real-time data augmentation, and OpenCV - a library mainly aimed at real-time computer vision which was used for image processing tasks. Additionally, we used python's multiprocessing module to speed up data processing. All code was written in Python.

We were given a small annotated dataset of 45 images of size 3264 x 4912 (Figure 1a), in each image about half of the plants (3 – 8) were annotated (Figure 1b), and in order for Mask R-CNN to work, we planned on using data augmentation to create a bigger dataset, which will be used to train Mask R-CNN. Because not all plants were annotated, we had to cut out the annotated plants and use only them to construct new images.

Figure 1

(a) Original dataset samples



(b) Ground truth annotations

From the affine and color augmentation that we tried, the best results were obtained by rotation and flipping horizontally/vertically. Overall, the best results were obtained by creating our own plants from individual leaves. However, due to the way grown plants intersect between themselves, which made it hard even for us to differentiate between the plants, we got good results on young plants, and mediocre results on grown plants.

Annotations of leaf masks (polygons of leaf contours) were fine and accurate but start/end points and leaf ordinality were inconsistent, therefore we could not train the network to recognize start and end points of leaves, nor to identify leaf ordinality.

System Description

All we needed was an NVIDIA GeForce GTX 1080 graphics card and about 32 GB of RAM, a Mask R-CNN implementation, Python and Keras. We used two PCs and two servers to run training simultaneously, implying the project was implemented both on Windows and Linux.

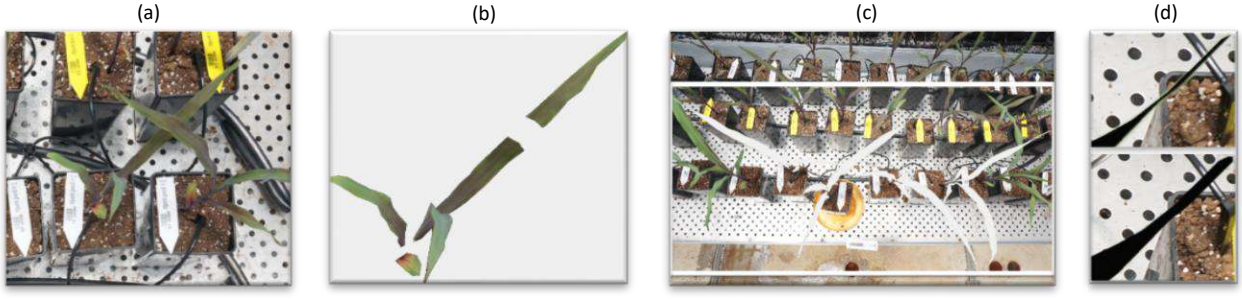
Execution Description

Several approaches were attempted for processing the annotated data, augmenting the data for creating a large training set, inspecting and improving the results. Following are the process development stages. Each step depicts a different approach.

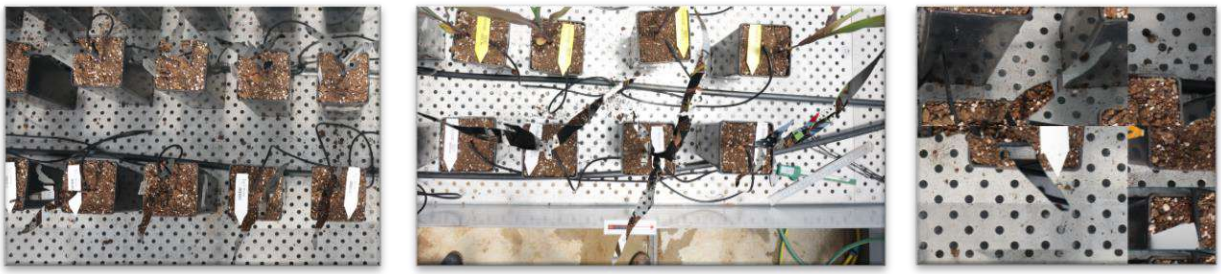
Step 1 - Cleaning up the images from plants that were left unannotated. This is necessary because otherwise Mask R-CNN will learn to differentiate between plants that were annotated and plants that were not.

1. Cutting out the annotated plants and creating an image out of each single plant with transparent background. (Figures 2a, 2b)
2. Creating backgrounds from the parts of the images that have no plants in them.
    a. Crop out the annotated plants and leave only the ROI (region of interest). (Fig. 2c)
    b. After removing the plants there are still some plant pixels left on the border of the cut, so erosion and smoothing needs to be applied to remove them. (Fig. 2d)
    c. Paste as many images with cropped out plants as necessary over each other until the holes are filled completely. (Figures 2e, 2f)
    d. Create a color histogram of the plants, to identify background parts. Each patch that had 1% colors of plants or lower was saved.
    e. Create backgrounds in various sizes from the collected patches. (Fig. 2g)
    f. Match backgrounds with the cut-out plants from section a. (Fig. 2h)

Figure 2



(e) (f) (g)



(h)

Figure 2. Figure 2b illustrates the plant from Figure 2a after cutting. Figure 2c illustrates stage 2(a). Figure 2d (top) illustrates the border of the cut with green pixels left. Figure 2d (bottom) illustrates the border of the cut after erosion and smoothing. Figures 2e and 2f illustrate the result after stage 2(c). Note that there are still leaves in Figure 2f. Figure 2g illustrates the result after stage 2(e). Figure 2h illustrates the result after stage 2(f).

Resulting from step 1, we had approximately 180 images of individual plants. In order to expand the dataset, we used data augmentation.

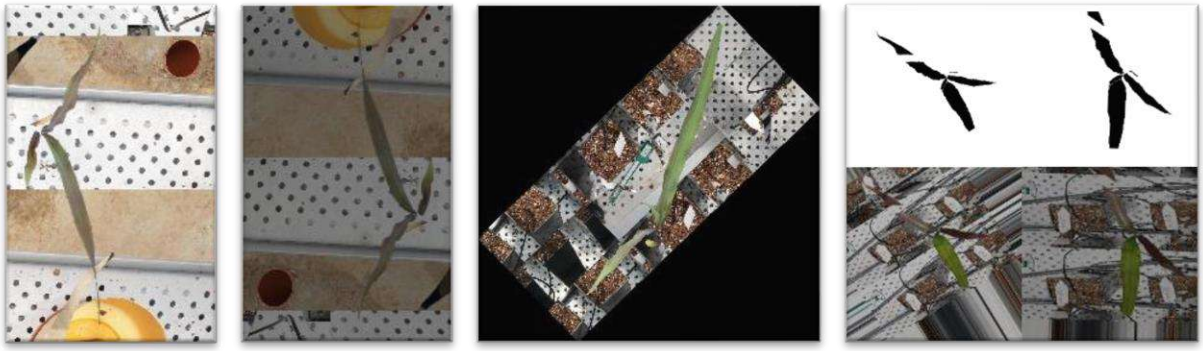
Step 2 - Custom data augmentation

1. Create a new data augmentation. (resizing, translation, rotation, flip, shear, brightness adjustment, stretch, contrast normalization, Gaussian blur, hue and saturation adjustment)
2.
    a. We used Pillow and OpenCV modules to transform the images.
    b. We converted the JSON files to mask images and applied the augmentations to both the images and the masks. Color augmentations were only applied to the images.
    c. We saved the new images and new masks as a new dataset.
3. Train Mask R-CNN on the expanded dataset.
4. Check the results using TensorBoard, and search for a data augmentation that would lower the total loss value and the validation loss value together.
5. Let Mask R-CNN segment images from the test set and see if it segmented them correctly.
6. Try to combine different data augmentations to create a bigger dataset and better results from TensorBoard.
7. In case of unsatisfactory results, return to step 1, to create another data augmentation.

We continued in this cycle of work, up until we got to a dataset that was ~3000 images in size.

Figure 3

(a) (b) (c) (d)



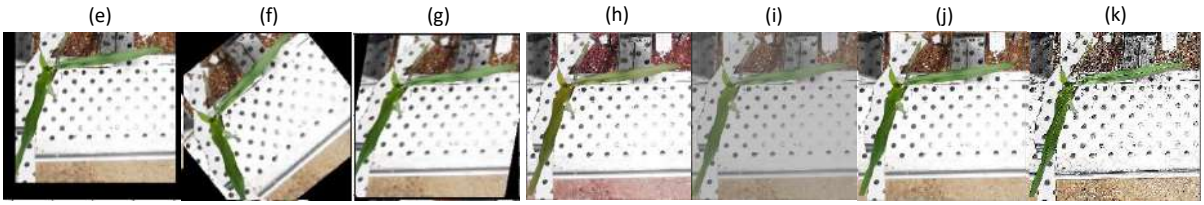


Figure 3 illustrates the types of data augmentation used. (a) Plant before augmentation. (b) Horizontal and vertical flips, brightness adjustment. (c) Rotation. (d) Plant and mask after rotation and stretching. (e) Translation. (f) Rotation. (g) Shearing. (h) Hue and saturation adjustment. (i) Contrast normalization. (j) Gaussian blur. (k) Embossing.

We approximated that we would need a dataset of at least 3000 training images to see some results. We've reached a dataset of size 3600, but the results weren't getting better and the training time was up to 30 hours, which made testing new data augmentations too slow. We had to think of a way to lower the training time.

We thought the training time could be lowered by using Keras, because it can create new augmented images while training instead of creating them beforehand and reading them from the disk while training. This proved to be slower, however made it easier for us to implement new data augmentations and keep track of our datasets. We went from creating datasets with images in a resolution of ~3000x4000 to images in a resolution of 1024x1024, which greatly reduced the running time from 30 hours to 8 hours. We also tried images in a resolution of 256 x 256 but they were too small to get us the results we wanted.

We still needed to improve the results. We realized that the generated background images do not contribute and perhaps even reduce performance and decided to replace them with random soil images. While the images in the dataset contained multiple plants, the training set we have created only had one plant in each image. So, we've created a new dataset with multiple plants in each image. Moving to multiple plants dramatically improved the results. It only required finding a way to position the plants together so that they wouldn't overlap too much. We first tried training the network to identify entire plants but got poor results (33% IOU average). Then we trained the network to identify leaves and the results improved significantly (64% IOU average). But the results were still unsatisfactory and further improvement was needed.

We then thought perhaps the 180 single plants don't have enough variation in them, so we had decided to construct our own plants.

Step 3 - Artificial plant construction

1. Crop out each leaf and save it as an image.
2. Align all leaves vertically. Because end points were inconsistently annotated, we first had to find the top and bottom points of each leaf.
    a. For each leaf in a plant, find the two points that are farthest away from each other. Due to the elongated shape of the leaves this gave us the start and end points of the leaves. All that is left is to differentiate between the two points, A and B.
    b. Choose one leaf, and
        i. Create a sum of distances from point A in the leaf to all other two points (A and B) of each of the other leaves, and save all distances from A and their corresponding points to an array DistA (e.g. [(p_1, 50), … (p_n, 40)]).

ii. Create a sum of distances from point B in the leaf to all other two points of each of the other leaves, and save all distances from B and their corresponding points to an array DistB (e.g. [(p_1, 150), … (p_n, 10)]).

c. The bottom points of the leaves are clustered together, and thus the point which is at the start of the leaf would have a smaller sum of distances. Let's say the point that had the smaller sum is A, sort DistA in ascending order, and save the first n (number of leaves) points to an array Middle_arr.

d. Find the middle point between all points in Middle_arr, this is the center of the plant.

e. Crop out the leaf and rotate it so that its bottom is in the middle of the bottom of the new image, and its top is in the middle of the top of the new image.

3. Construct new plants using the aligned leaves (Fig. 4)
   a. Randomly select the number of leaves in the plant. (4 – 8)
   b. Sort the leaves by size. This is needed for pasting the smaller leaves on top of the larger ones.
   c. For each leaf, ordered from large to small:
      i. Generate a random angle α between $-15°$ and $15°$.
      ii. If the leaf is even: paste the leaf at angle α around a fixed center point.
      iii. Else: paste the leaf at angle $\alpha + 180°$.
      iv. Continue to step i until all leaves are pasted.
4. Replace the original plants with the new plants in the previous experiment. (Fig. 5)

This gave us similar results.

Figure 4

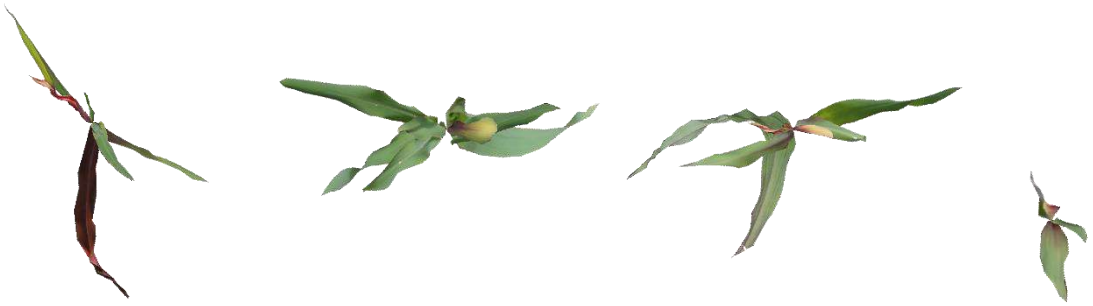


Figure 4. Examples of artificial plants.

Figure 5

Figure 5. Examples of training images from Step 3 - artificial plants on top of ground and greenhouse backgrounds.

We thought that maybe the plants should be better separated to look more like the original images, and to limit them from overlapping too much.

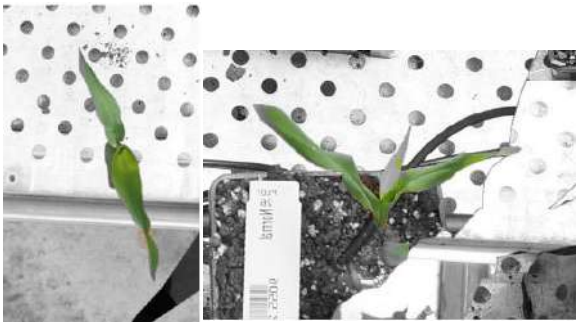
5. We created new images, with multiple plants in each image.
    a. For each image, we randomly selected if it was going to have 1, 2, or 3 rows and 1, 2, or 3 columns, So each image had 1 to 9 plants in it.
    b. Each plant was placed randomly in one of the cells in order to mimic the original images (e.g. you can't have the base of two plants in the same coordinate).

Unfortunately, this did not improve the results.

Results

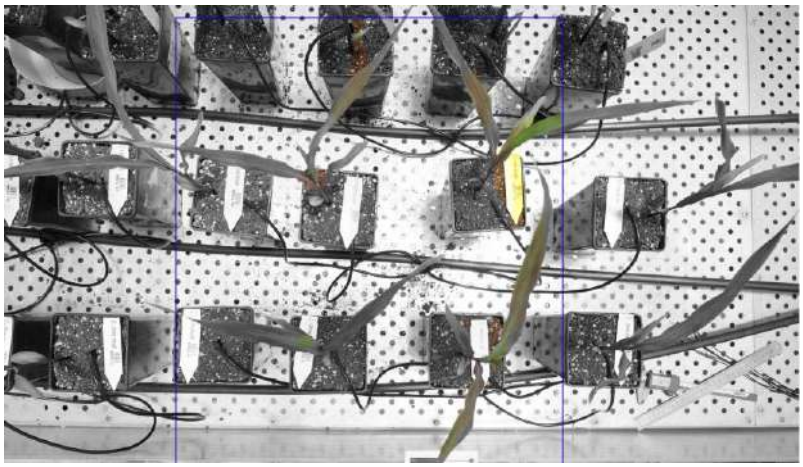
1. Custom Data Augmentation
   The results on augmented single plants weren't so bad as you can see in **figure 6.** (everything that is colored is recognized as the plant, everything that is in greyscale is recognized as background)

Figure 6



However, the results on the test images were poor, since it recognizes all plants as a single instance of one plant (each instance has a blue box around it), and moreover it didn't segment the plants well. As you can see in **Figure 7,** most plants are in black and white, meaning it recognized them as background.

Figure 7



2. Keras data augmentation, datasets of smaller images (1024x1024) and each image has one plant. Results in **Figure 8.**

Figure 8

Still we can see that Mask R-CNN recognizes all plants as one instance, but the segmentation got a bit better. So, we've implemented a simple IOU.

Table 1 presents the numeric results (explanation below):

Table 1

Type of images		affine	affine						All
			hue, invert, flip, contrast		emboss	flip	gauss	hue	
			3200 steps	6400 steps					
IOU on original images from test set (ROIs)	Original size~3000x4000	9%	9%	11%	10%	13%	10%	8%	9%
	Min(width,height)= 1024	11%	11%	8%	8%	12%	10%	9%	9%
	Max(width,height)= 1024	12%	11%	8%	11%	14%	10%	9%	7%
	50% of the original size	9%	11%	9%	10%	14%	10%	9%	8%
IOU on cropped out single plants from test set		49%	57%	49%	57%	54%	44%	53%	54%

We thought the results should be better, and that maybe training on images in a resolution of 1024x1024 and then testing on images of a much bigger size ruins the results, so we scaled the test image to a max of 1024, a min of 1024 and to half of its size. But still got the same results.

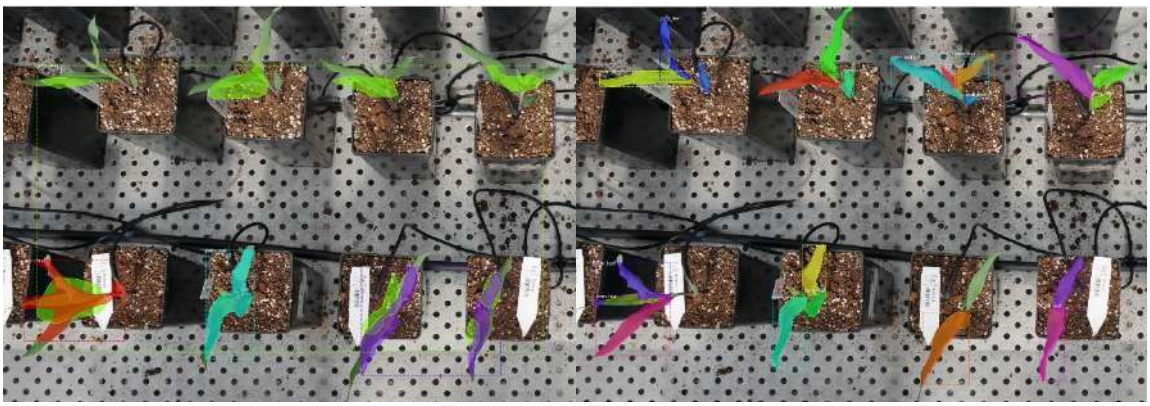
However, the IOU on images of single plants where much better, so we knew the neural network was learning. The problem was the dataset - we needed a better data augmentation.

3. Rethinking our strategy:
    a. We've decided to drop the custom backgrounds because they had cut-outs in the shape of leaves, which may have confused the neural-network, and decided to use backgrounds from the internet. (Fig 9)
    b. We've decided to create images with multiple plants in them, since in the test images all images contain multiple plants that intersect.
    c. Moving to masks per leaf instead of masks per plant.

Figure 9

Results from this strategy are visualized in **Figure 10**.

Figure 10



And numeric results are presented in **Table 2**:

Table 2

	Type of images		2-5 plants per image		4-10 plants per image	
			mask per plant	mask per leaf	mask per plant	mask per leaf
	IOU on original images from the test set (ROIs)		32%	55%	-	61%
	IOU on cropped out single plants from test set		53%	59%	-	59%
New IOU	IOU on original images from test set (ROIs)	IOU average	32%	60%	-	64%
		clashes	18%	2%	-	3%
		Incorrect	0%	23%	-	20%
		Not found	43%	40%	-	39%

New IOU - match result segments to ground truth segments

Clash – two or more result segments match the same ground truth segment (percent of all result segments)

Incorrect – a result segment which does not match any ground truth segment (percent of all result segments)

Not Found – a ground truth segment which was not matched to any result segment (percent of all ground truth segments)

4. We still didn't get good enough results. We've decided it's because we have a small variation of plants (we still use the original 180 plants). So, we've created augmented plants (Figures 4,5). But we didn't get significant improvements (Table 3). Figure 11 Visualizes the results.

Figure 11

Table 3

		4-10 plants mask per leaf
IOU on images from test (original ROIs)		56%
IOU on cropped out single plants from test:		60%
New IOU on images from test (original ROIs)	IOU average	60%
	clashes	5%
	Incorrect	16%
	Not found	39%

Conclusions

1. Using affine augmentation and color augmentation didn't really help, the only thing that did was flipping horizontally/vertically.
2. Categorizing all plants as one class when they look so different from one another was a mistake. A better approach would have been to create 2 classes - one for old plants, and a second one for young plants.
3. Creating backgrounds out of the given dataset was a waste of time, we saw no difference when using generic backgrounds.
4. Working with images above 1024x1024 was a waste of time for epochs larger than 1800 images and training session with 30 epochs. Each training session took a minimum of 20 hours.
5. Training MASK R-CNN on images with single plants resulted in bad IOU since all the grown plants intersect consistently in images from the original dataset.

   In conclusion, if we had split the dataset to a dataset of young plants and grown plants, we would have gotten much better results. In addition, images with all the plants annotated in them would have helped too, since then we could have used data augmentation on them directly.

Further Work and Suggestions

Build a better data augmentation - synthesize a plant that resembles a grown plant from the ground truth. Perhaps only leaves from grown plants should be used for that purpose. Create an image with multiple grown plants that overlap and then minimize it to 1024x1024.

Sources

1. He, Kaiming & Gkioxari, Georgia & Dollár, Piotr & Girshick, Ross. (2017). Mask R-CNN.
2. Mask R-CNN implementation and examples.
3. Balloon Instance Segmentation article with code example.
4. Tensorflow
5. OpenCV
6. Kuznichov, Dmitry & Zvirin, Alon & Honen, Yaron & Kimmel, Ron. (2019). Data Augmentation for Leaf Segmentation and Counting Tasks in Rosette Plants.