המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

Cucumber plant parts Detection and Segmentation

Technion - Israel institute of technology

CS 234329 - Project in image processing and analysis

15 May 2019

Abstract

This Project tackles the task of segmentation and indexation on cucumber plant parts. The approach is to implement an end to end workflow to partially annotated and relatively small datasets.
Mask RCNN paper from facebook research has proven very efficient in this task even on small datasets. Therefore the focus is maximizing the effectiveness of a problematic dataset on different levels. This reports discusses partially annotated datasets, and inconsistent annotation styles.
The conclusion tends towards a trade-off between investing in the dataset quality and using augmentation acrobatics to quickly use the existing data. There is no doubt that consistent fully annotated real pictures will get the most out of the network.

Authors: Simon Lousky, Asher Yartsev, Or Shemesh
Supervisors: Alon Zvirin, Yaron Honen, Dima Kuznichov

Table of Contents

Introduction

A wide number of geometrical and AI based models are aimed to detect and segment objects. In the farming industry this task is particularly interesting and can have massive impact on automatisation and therefore have environmental and economical values. Previously used models for this task in the lab were patch based classification, and encoder decoder to separate different parts of a plant. These techniques yielded valuable results on tomato plants but did not perform so well on other available datasets like the cucumber dataset. It was hypothesized that Mask-RCNN should tackle the task easily enough.Hence, our objective was to use a previously developed model that generated synthetic data to train on the cucumber dataset, and try to get the best results possible. Since the objective was getting the best results, and after understanding better how Mask-RCNN works, we wondered about training the model on a regular ‘coco’ style dataset. This expanded our work to be the creation of a technical workflow and environment to easily train on generated data as well as on real datasets.

System Description

Mask R-CNN in a nutshell

Easily put, this network resembles a tree where the nodes are DNN components, each with a simple and specific task.
Let’s describe the relationship between those components:
The first block is a CNN (call it backbone network) which should be some state of the art feature extractor like ResNet (resnet-50 in our case).
The second component would be the RPN (Region Proposal Network) Which shares convolutional layers with the backbone using a FPN (Feature Pyramid Network). FPN improves region proposal and is more accurate than using only the last feature map layer of the backbone.
The third node is a FCN (Fully Connected) that performs both regression on the bounding boxes and classification inside the regions.
The last one is the masking CNN part which is basically a binary classifier that indicates for each pixel in the region whether it is part of an object or not.
More Mask R-CNN implementation details are explained in a later chapter.

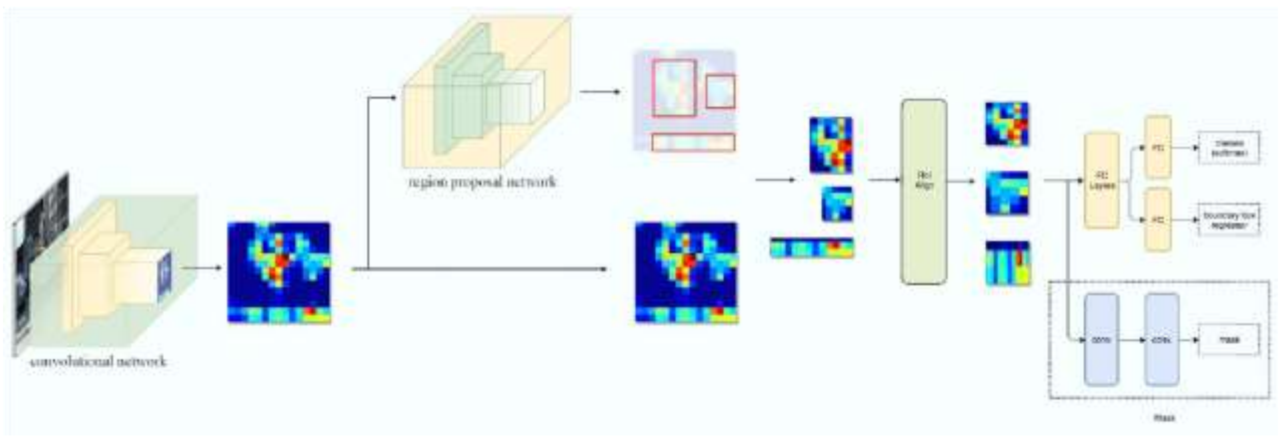


Figure 1 - Schematic of Mask R-CNN architecture

Our dataset

The dataset was 4 sets of 6000 x 4000 pictures, accompanied by a .json file in the COCO format[1] .Our goal was to successfully detect four different classes, preferably doing it in a single forward pass on a picture. All four classes were four different parts of a cucumber plant:

1. Fruits (aka cucumber) - 101 Pictures

[1] See http://cocodataset.org/#format-data

2. Leaves - 140 pictures
3. Flowers - 131 pictures
4. Stems - 481 pictures

Each class having its own dataset, they differed both in quality and quantity.
Training this Network assumes full supervision of all objects, Therefore not having one combined dataset is the main issue, given the need to inference through a single network for all our classes. On the other hand hundreds of picture (one would argue that tens of pictures) is a reasonable amount for a pre-trained Mask RCNN model, both in terms of numbers and diversity.

Dataset Inconsistencies

Three major types of inconsistencies made our dataset challenging.

Occlusion management

The first is occlusion management. In some cases the masks contain occluded parts, and in others the occlusion is omitted. For example in figure 2 , the stem that hides the blue left cucumber is annotated as part of it, but the cucumber in the middle of the picture is separated by a stem. Moreover it is worth noticing that COCO dataset is consistent regarding the occlusion matter. See figure 2 - The man’s hands are hiding the surfboard but are not marked as part of it.

Multiple polygons for single object

The second anomaly is annotations of different parts of the same object as different objects. See the middle pink cucumber in figure 2, the yellow color indicates that it is annotated as a new fruit. In this case as well the COCO annotation style permits multiple polygons for a single object. See figure 2 - The car in the background is made of two separate polygons, or the three parts of the surfboard that are annotated as the same object.

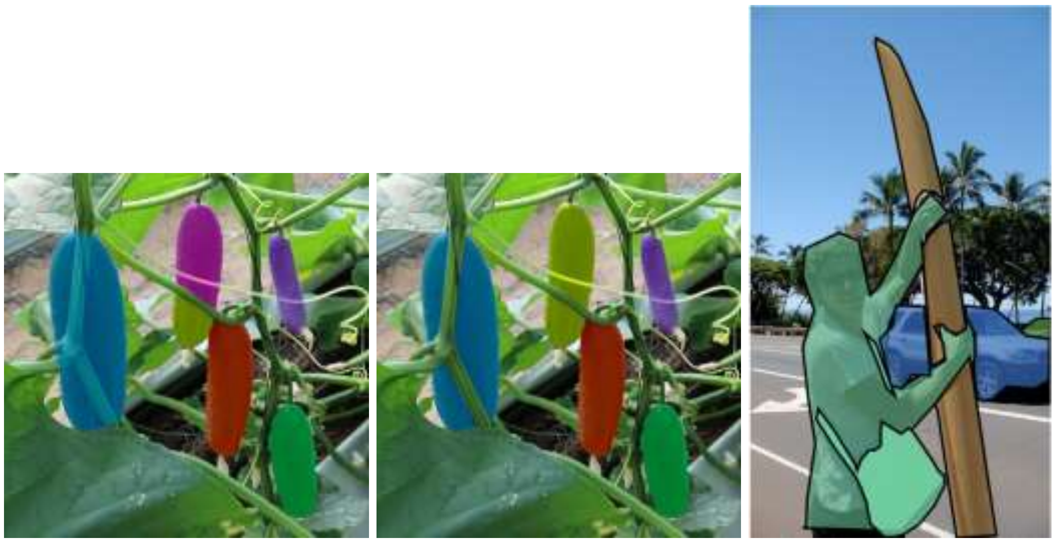


Figure 2 - Annotation of occluded objects.

Left image shows the typical anomalies found in the original dataset. Middle Image is the desired annotation for such image.
The Right image is an example from MS-COCO (image id 246120).

Under supervised datasets

The third inconsistency type is low percentage of annotated objects per image in our dataset. Like previously said, the network assumes full supervision over the objects, therefore having relevant but unannotated objects, is a serious pitfall in using the picture as ground truth. Let’s assume we wish to detect all mature foreground leaves. In figure 3, you can see only one leaf is annotated out of many leaves that are clearly visible, compared to a non perfect but yet significantly better annotated image on the right (figure 4), where 13 apples out of 16 are annotated.

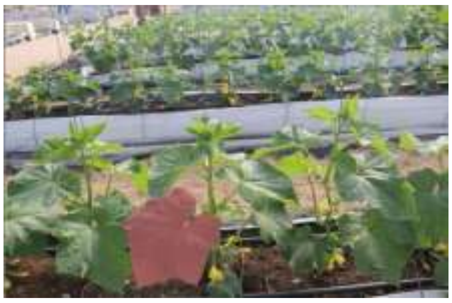
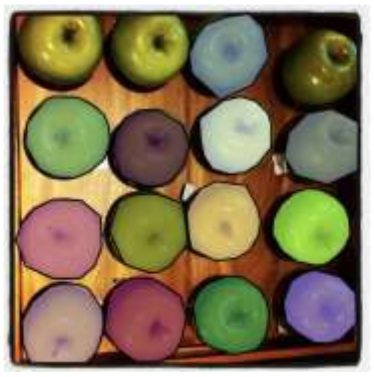
	

Figure 3 - Annotation of frequent objects images. Left annotated image from our dataset compared	Figure 4 - Image from MS-COCO (image id 239565)

Towards data enhancement

First steps

The direction we were thrown at, regarding the dataset issue, was synthetic data creation. Based on the work previously made by the lab, we were about to train the model with pictures of randomly dispersed objects on neutral backgrounds, generated at training time (figure 5 and figure 6). This technique seemed promising and we were intrigued by the question whether this approach would yield results as good as real, fully annotated datasets. This question lead to some work we discuss later on.

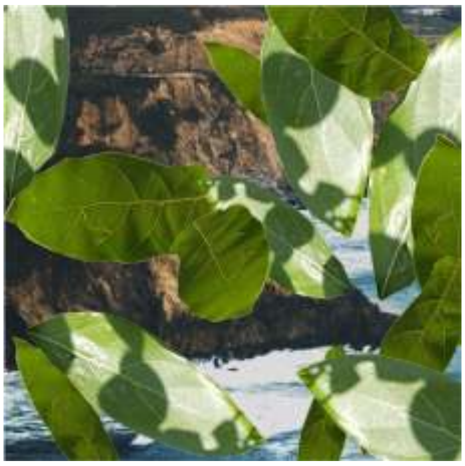
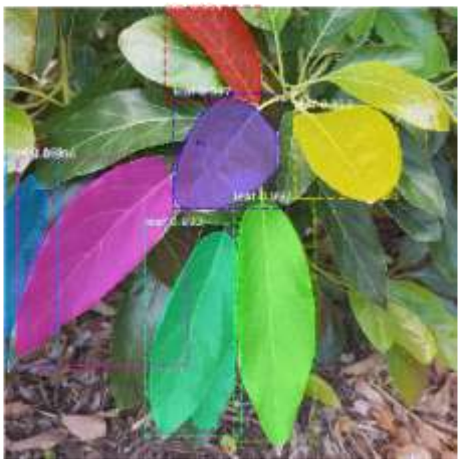
	

Figure 5 - Synthetic image	Figure 6 - Inference after synthetic training

Repairing the dataset

Looking at the final goal that was at stake, we were determined in any way possible to make the best out of what we got. We knew we're going to want to play around with the data, change resolutions, augment the pictures, maybe even fix some of the pitfalls in the dataset. For this reason we searched for an open source tool that could help us visualize and manipulate the dataset. The best option we found was called *COCO-annotator*[2], it was intuitive, easy enough to configure and bring up locally. This tool gave us the ability to manipulate, add annotations and preview the dataset immediately, but we needed some additional features we later developed:

1. Merger Tool - This tool was intended to connect few polygons that the original dataset was considering multiple objects. See figure 2
2. Stylus and Ipad related features - In order to get the most out of the tool, we thought adapting it to a stylus enabled touch device would make the experience of annotating and correcting a bad dataset much easier. The obvious additions were:
    a. Zoom in out
    b. Move around
    c. Mouse events changed touch compatible events
3. Performance improvement - The original repository had a background task draining all the resources of the server and never saving the work done, we fixed this issue as well.

[2] Original COCO-annotator by jsbroks - https://github.com/jsbroks/coco-annotator

Altered version for our project - https://github.com/simonlousky/cucu-annotator

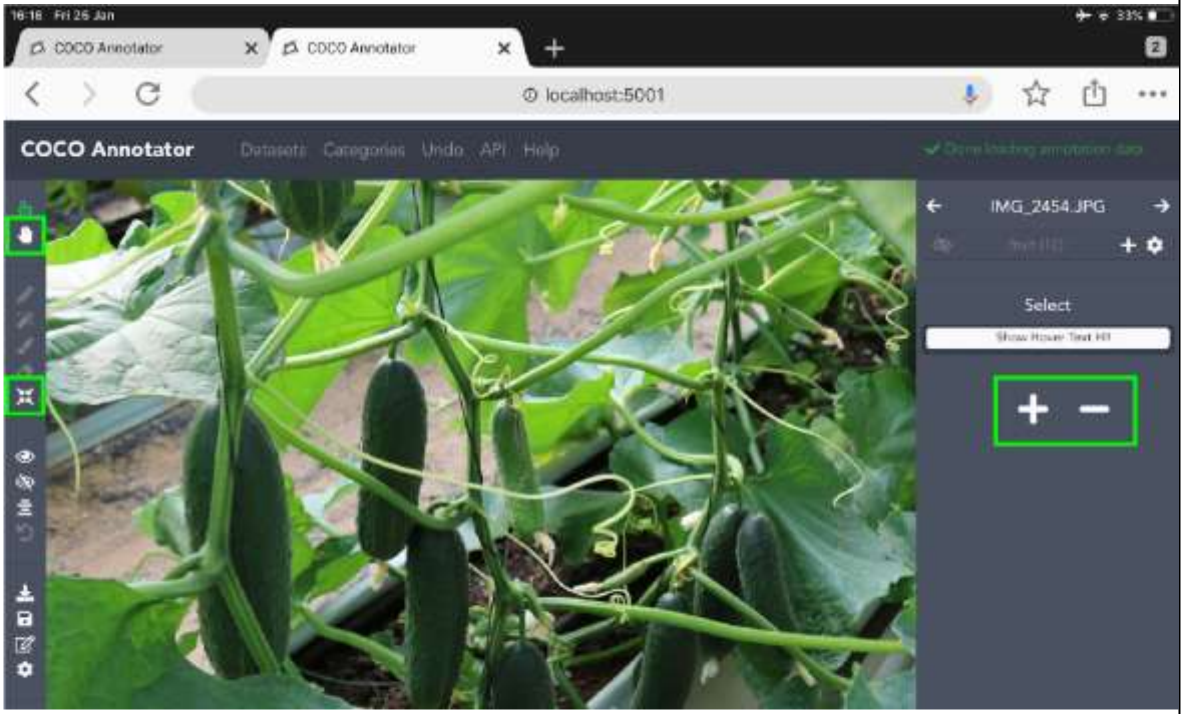


Figure 7- Our altered version of COCO Annotator

Having the final goal in mind, we used the newly improved coco-annotator and did some work on the original dataset.
One additional thing to note is that in any step we took regarding manipulation of the dataset, including data augmentation we later talk about, we insisted on having a way back to the original format of the data set. That is - A simple directory with a bunch of picture and a single .json file with all the annotations.

Synthetic datasets

The first step in image generation was singling out objects out of the pictures, using the segmentation coordinates found in the .json file. This was done using a semi-smart script that could find any object, cut around its edges, normalize its size, and apply transparency to its background. In order to get minimal manual work on the resulting object bank, the script Iterates over all the segmentations in the annotations file, filtered out objects with irrelevant size or bad width/height ratio, resized all of them and saved them in a new directory, one for each class.

Object filtering

As stated before, objects were filtered using simple logic. One parameter was the size of the object compared to the picture, we fixed a minimum size of 10% of the pictures's height. The Second parameter was the minimal height:width aspect ratio of an object. Fruits ratio was fixed at 3:1, leaves at 1:1, stems at 4:1. Another matter to address was occluded objects which could yield partial objects. For this we only singled out objects made of one polygon. Using this simple filtering algorithm reduced the manual work by an order of magnitude.

Although you could arguably use the direct output of this script to train the network and get good results, we decided to manually filter the data even more carefully.

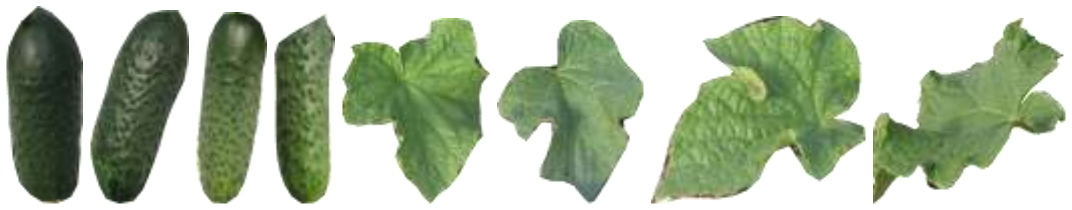


Figure 8 - Sample of cut and filtered objects

Enhanced blending

One might argue that incorporated objects in a synthetic pictures can become easy targets for a deep neural network. The sharp edges of the cut object definitely break the pattern of any background and therefore the network might focus on this feature. In an attempt to imitate real life images, we blurred the mask contour to make the edges of the RGB picture blend with the background more realistically (See figure 9) . This theory is later put to the test.

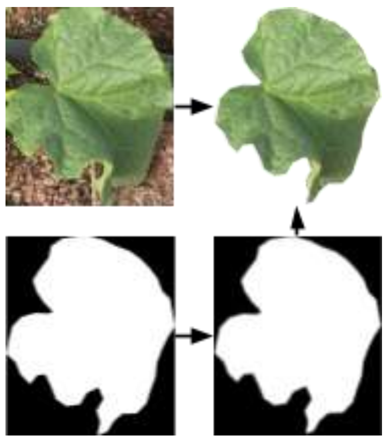


Figure 9- Blurred mask makes the RGB content fade to the background RGB content, without blurring the RGB content itself.

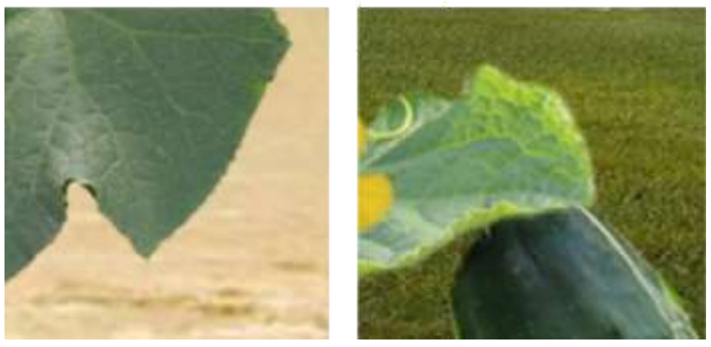


Figure 10- on the left a sharp edge, on the right a leaf edge with enhanced blending applied

Object scenery

One other aspect of generating an image is the composition of the pictures. How many leaves are there, how are they dispersed, how big they are etc…
These hyper parameters are all concentrated in a configuration file and the tests were performed with the following ones:

- `MIN_SCALE_OBJ = 0.1:` The minimum size of an object is 10% of the picture's height or width.

- `RPN_ANCHOR_SCALES:` The maximum size of an object is 30% of the picture's height or width.
- `MIN_GENERATED_OBJECT = 5:` The minimum number of objects in picture is 5.
- `MAX_GENERATED_OBJECT = 20:` The maximum number of objects in picture is 20.
- `OBJECT_IOU_THRESHOLD = 0.05:` The maximum intersection between object is 5% of the picture's size

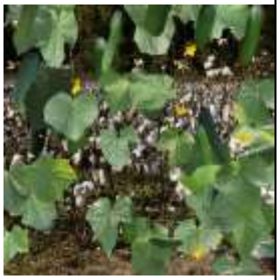
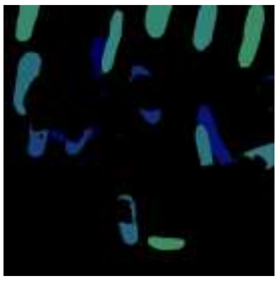
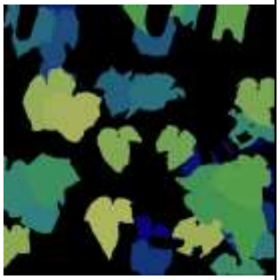
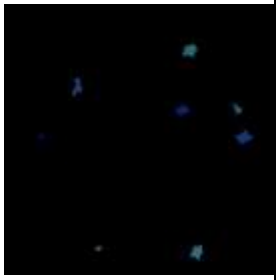
 Final image	 cucumber masks	 Leaf masks	 Flower masks

Figure 11 - Sample image from a mixed synthetic dataset

Real datasets

In spite that the original work was about training the network using the generated pictures technique, we found ourselves interested in comparing the results with a fully supervised training set of original pictures. This dataset would also need to be augmented in the DNN fashion way. Here we describe the treatments made offline (not in during training time) to improve the dataset.

We decided to enlarge our dataset by using a stochastic augmentation method, that can sample altered pictures from any dataset, applying directional flips, skews, brightness and contrast shifts by some degree of randomness. To do so, we used the Python module *Augmentor*[3]. In this process we tried to preserve the quality of the dataset by erasing nonsense annotations that rather looked like artifacts on the altered pictures. Objects smaller than 150 pixels in a 1536 x 1024 image and annotations of objects with extreme aspect ratios were removed. Pictures left with no annotations were removed as well.

[3] Original library - https://github.com/mdbloice/Augmentor

Altered version for our purposes - https://github.com/simonlousky/alteredAugmentor

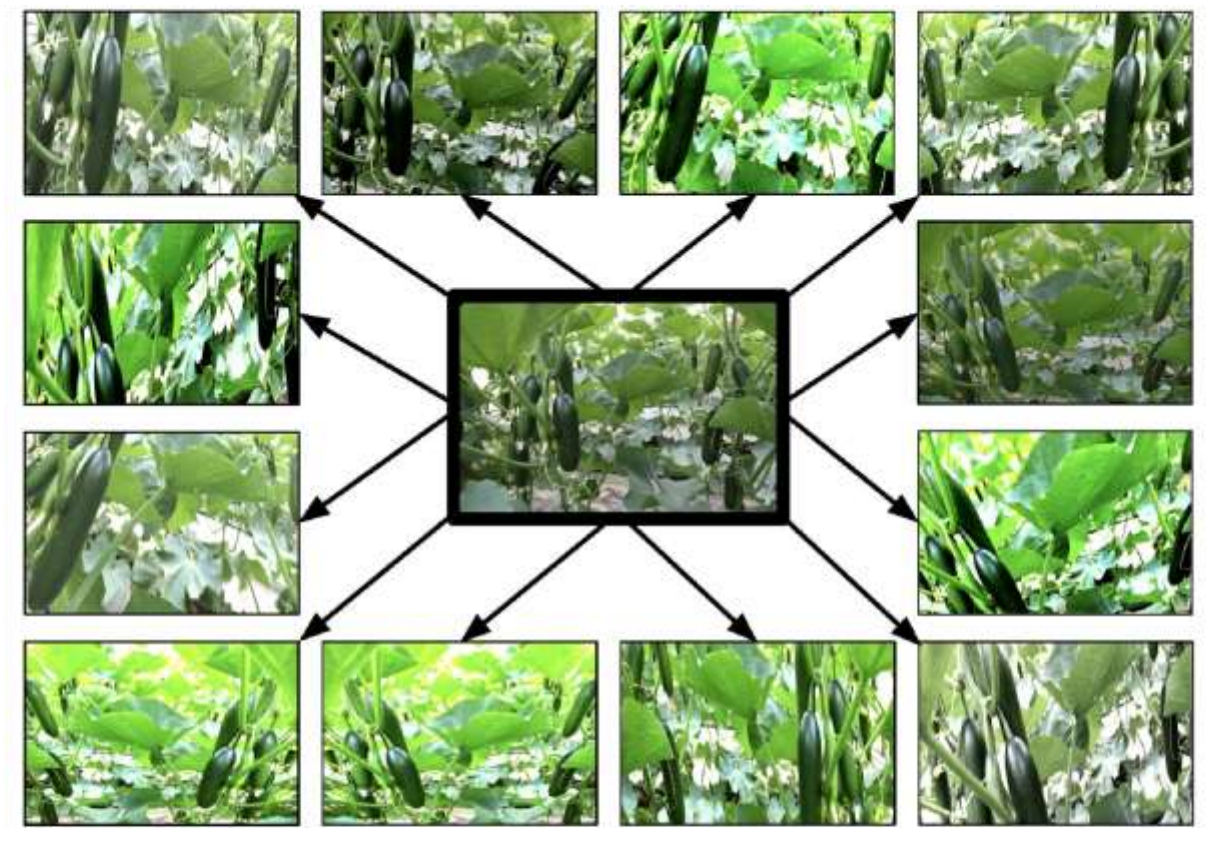


Figure 12 - Example of new augmented images generated from the same original image (IMG_2718.jpg) in the center.

Mask R-CNN Train and Implementation aspects

Mask R-CNN on Tensorboard

We invested some effort to redesign the original *matterport/Mask_RCNN* implementation to take advantage of Tensorboard's visual power, since the original code wasn't written with this design in head.

That way we, and others following us, will be able to analyze the net structure with confidence.

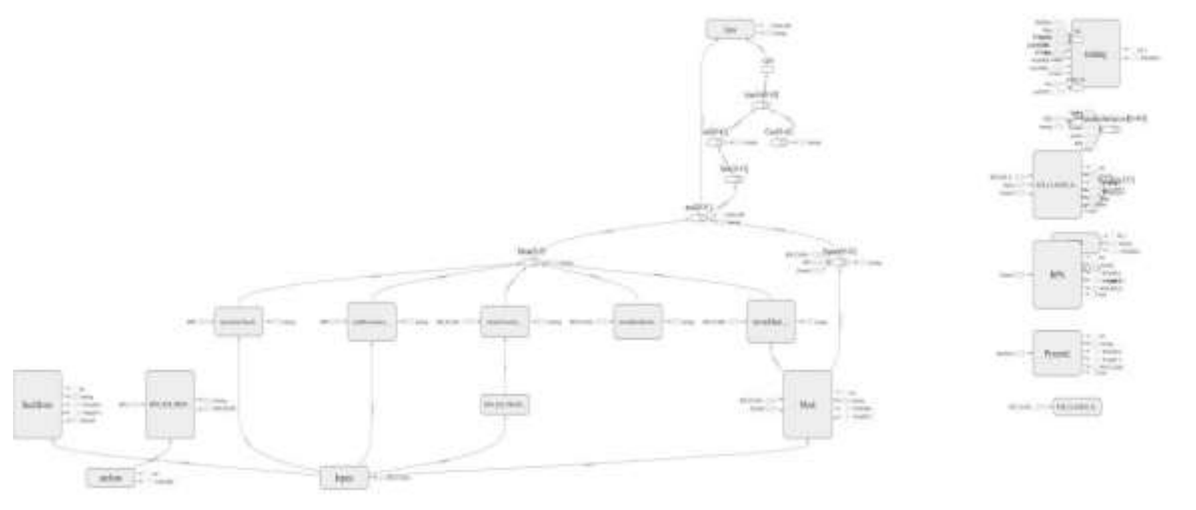


Figure 13 - Graph generated by Tensorboard after editing the model.
Full size png can be found in project repository on GitHub.

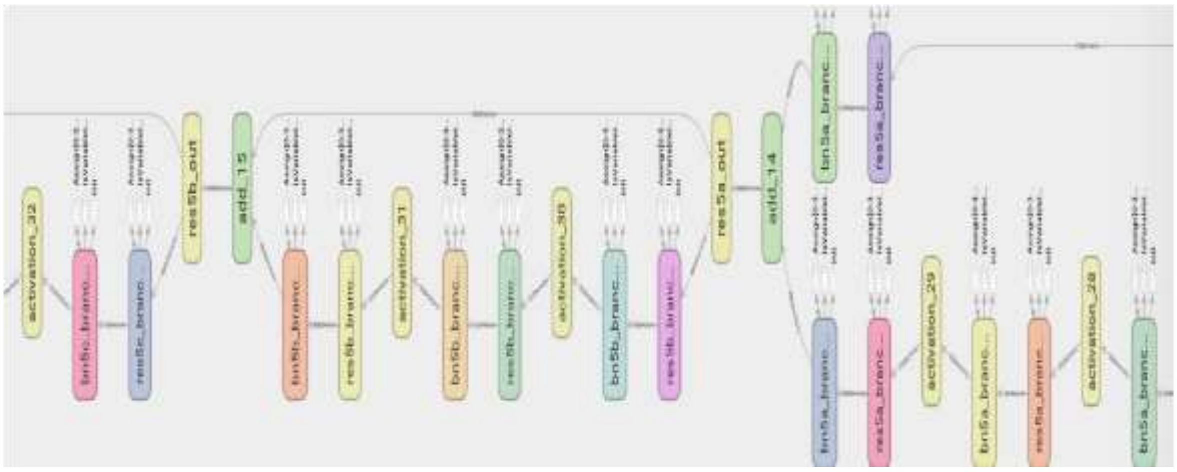


Figure 14 - A very small part of the default generated graph which make the net impossible to track and debug when needed

Mask R-CNN flowchart

During the first steps and last steps of our project we realized that fully understanding the architecture of Mask R-CNN is helping us focus on hyperparameters better. Unfortunately no satisfactory documentation was found, and TensorFlow showed way too much graphic “noise” that clouded the big picture. So with patience and full correspondence to the implementation we leaned on, and formulated a flowChart (figure 16) to help us and others in the future.

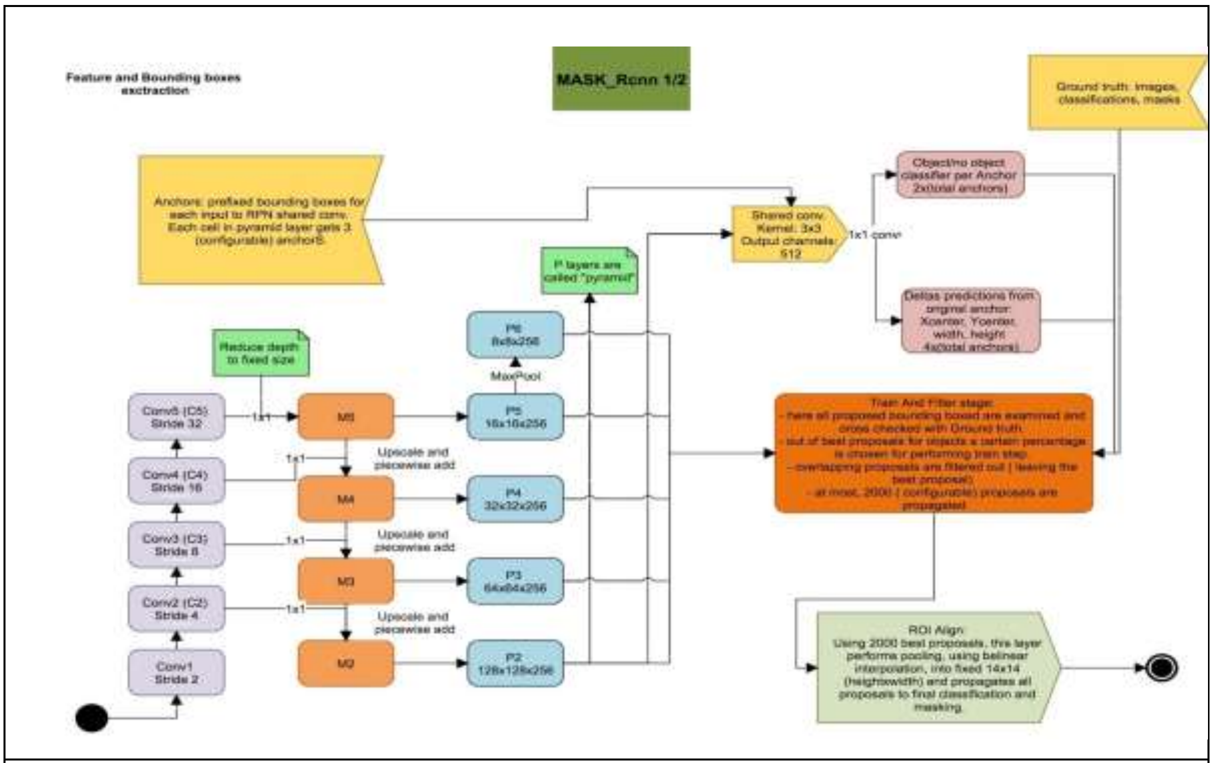

Feature and Bounding boxes exctraction
MASK_Rcnn 1/2
Ground truth: images, classifications, masks
Anchors: prefixed bounding boxes for each input to RPN shared conv. Each cell in pyramid layer gets 3 (configurable) anchorS.
Object/no object classifier per Anchor 2x(total anchors)
Shared conv. Kernel: 3x3 Output channels: 512
1x1 conv
P layers are called "pyramid"
Deltas predictions from original anchor: Xcenter, Ycenter, width, height 4x(total anchors)
P6 8x8x256
MaxPool
Reduce depth to fixed size
Conv5 (C5) Stride 32
1x1
M5
P5 16x16x256
Train And Filter stage: - here all proposed bounding boxed are examined and cross checked with Ground truth. - out of best proposals for objects a certain percentage is chosen for performing train step. - overlapping proposals are filtered out (leaving the best proposal) - at most, 2000 (configurable) proposals are propagated
Conv4 (C4) Stride 16
1x1
Upscale and piecewise add
M4
P4 32x32x256
Conv3 (C3) Stride 8
1x1
Upscale and piecewise add
M3
P3 64x64x256
Conv2 (C2) Stride 4
1x1
Upscale and piecewise add
M2
P2 128x128x256
ROI Align: Using 2000 best proposals, this layer performs pooling, using belinear interpolation, into fixed 14x14 (heightxwidth) and propagates all proposals to final classification and masking.
Conv1 Stride 2

Figure 15

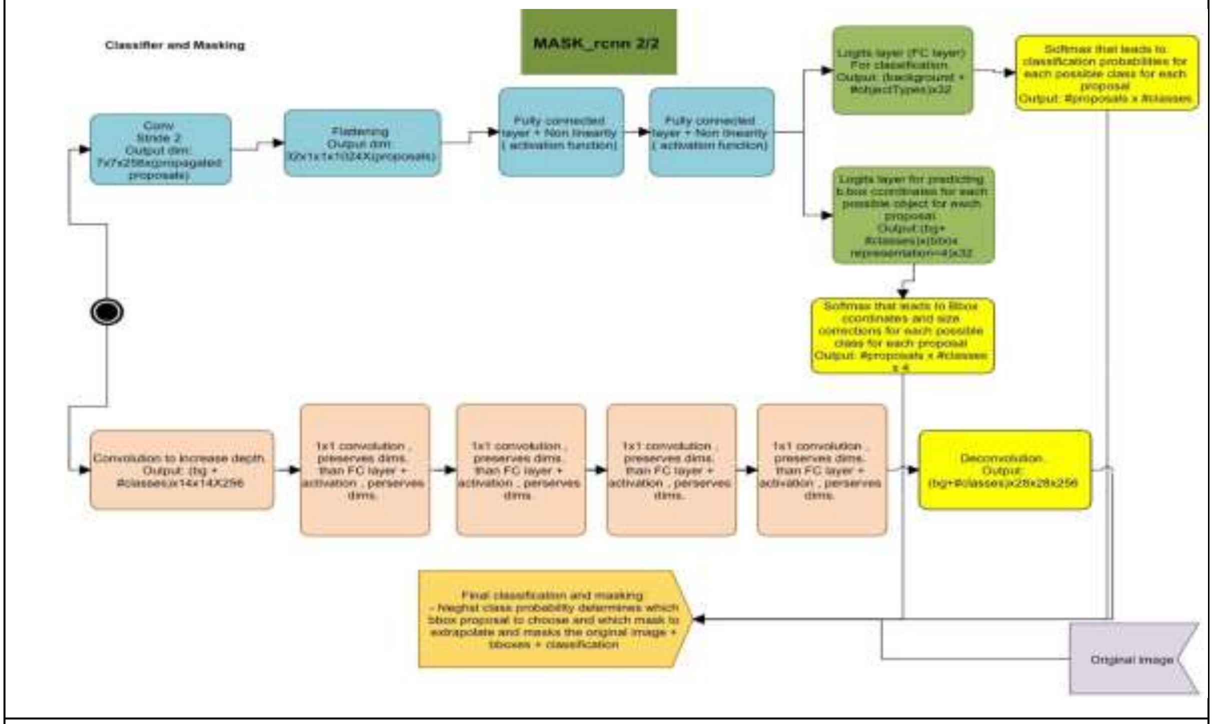
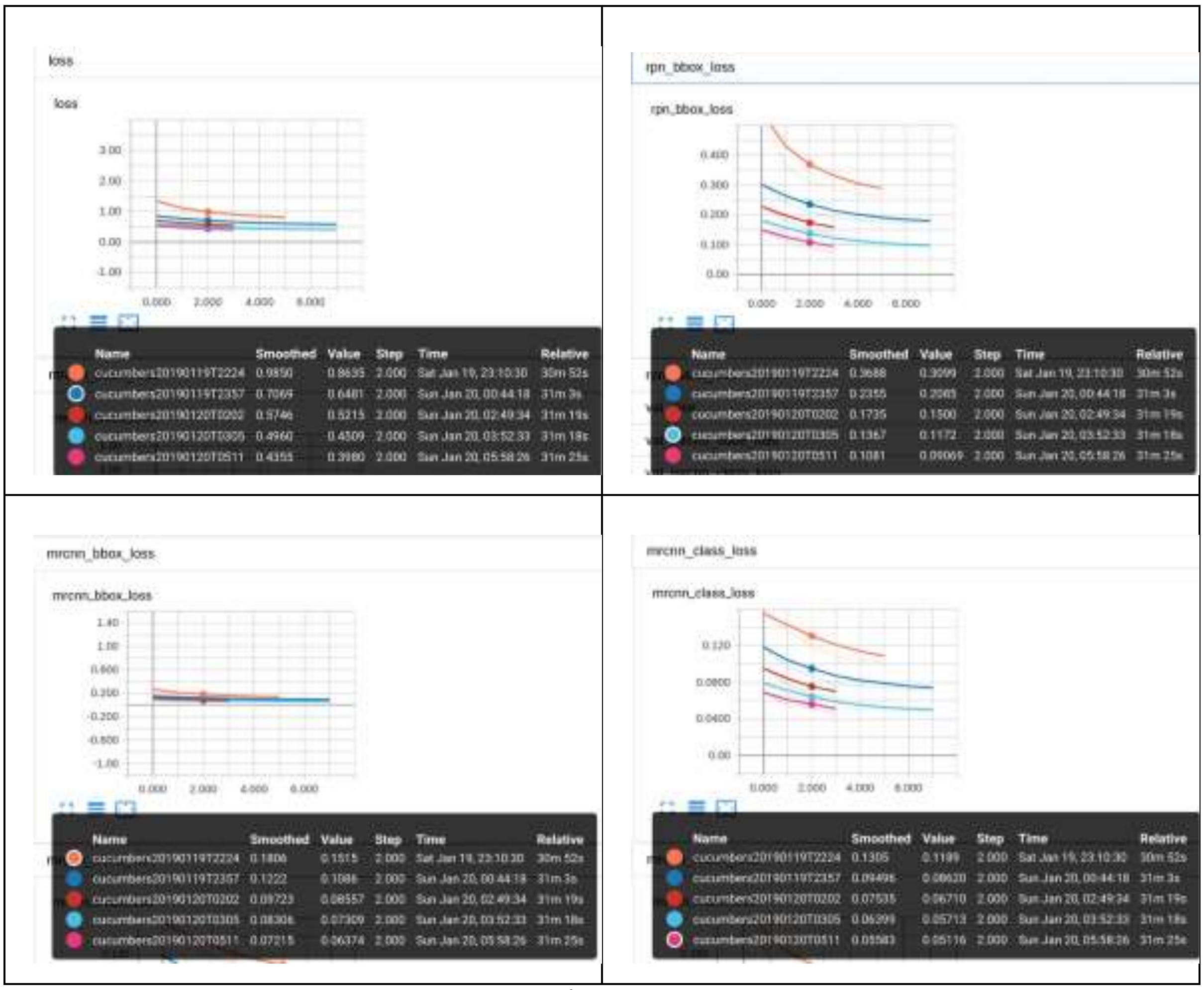

Classifier and Masking
MASK_rcnn 2/2
Logits layer (FC layer) For classification Output: (background + #objectTypes)x32
Softmax that leads to classification probabilities for each possible class for each proposal Output: #proposals x #classes
Conv Stride 2 Output dim: 7x7x256x(propagated proposals)
Flattening Output dim: (0x1x1x1024X(proposals)
Fully connected layer + Non linearity (activation function)
Fully connected layer + Non linearity (activation function)
Logits layer for predicting b.box coordinates for each possible object for each proposal. Output:(bg+ #classes)x(bbox representation=4)x32
Softmax that leads to Bbox coordinates and size corrections for each possible class for each proposal Output: #proposals x #classes x 4
Convolution to increase depth. Output: (bg + #classes)x14x14X256
1x1 convolution , preserves dims. than FC layer + activation , perserves dims.
1x1 convolution , preserves dims. than FC layer + activation , perserves dims.
1x1 convolution , preserves dims. than FC layer + activation , perserves dims.
1x1 convolution , preserves dims. than FC layer + activation , perserves dims.
Deconvolution. Output: (bg+#classes)x28x28x256
Final classification and masking - Heighst class probability determines which bbox proposal to choose and which mask to extrapolate and masks the original image + bboxes + classification
Original image

Figure 16

Monitoring through callbacks

The implementation we worked with, used embedded TensorFlow callbacks, this way we could graphically monitor each Loss function. (Figure 17)



	Name	Smoothed	Value	Step	Time	Relative
●	cucumbers20190119T2224	0.9850	0.8635	2.000	Sat Jan 19, 23:10:30	30m 52s
●	cucumbers20190119T2357	0.7069	0.6481	2.000	Sun Jan 20, 00:44:18	31m 3s
●	cucumbers20190120T0202	0.5746	0.5215	2.000	Sun Jan 20, 02:49:34	31m 19s
●	cucumbers20190120T0305	0.4960	0.4509	2.000	Sun Jan 20, 03:52:33	31m 18s
●	cucumbers20190120T0511	0.4355	0.3980	2.000	Sun Jan 20, 05:58:26	31m 25s

	Name	Smoothed	Value	Step	Time	Relative
●	cucumbers20190119T2224	0.3688	0.3099	2.000	Sat Jan 19, 23:10:30	30m 52s
●	cucumbers20190119T2357	0.2355	0.2085	2.000	Sun Jan 20, 00:44:18	31m 3s
●	cucumbers20190120T0202	0.1735	0.1500	2.000	Sun Jan 20, 02:49:34	31m 19s
●	cucumbers20190120T0305	0.1367	0.1172	2.000	Sun Jan 20, 03:52:33	31m 18s
●	cucumbers20190120T0511	0.1081	0.09069	2.000	Sun Jan 20, 05:58:26	31m 25s

	Name	Smoothed	Value	Step	Time	Relative
●	cucumbers20190119T2224	0.1806	0.1515	2.000	Sat Jan 19, 23:10:30	30m 52s
●	cucumbers20190119T2357	0.1222	0.1086	2.000	Sun Jan 20, 00:44:18	31m 3s
●	cucumbers20190120T0202	0.09723	0.08557	2.000	Sun Jan 20, 02:49:34	31m 19s
●	cucumbers20190120T0305	0.08306	0.07309	2.000	Sun Jan 20, 03:52:33	31m 18s
●	cucumbers20190120T0511	0.07215	0.06374	2.000	Sun Jan 20, 05:58:26	31m 25s

	Name	Smoothed	Value	Step	Time	Relative
●	cucumbers20190119T2224	0.1305	0.1199	2.000	Sat Jan 19, 23:10:30	30m 52s
●	cucumbers20190119T2357	0.09496	0.08620	2.000	Sun Jan 20, 00:44:18	31m 3s
●	cucumbers20190120T0202	0.07535	0.06710	2.000	Sun Jan 20, 02:49:34	31m 19s
●	cucumbers20190120T0305	0.06399	0.05713	2.000	Sun Jan 20, 03:52:33	31m 18s
●	cucumbers20190120T0511	0.05583	0.05116	2.000	Sun Jan 20, 05:58:26	31m 25s

Figure 17

The graphs were taken from small experimental training procedures, in which we just tested an innovative training technique on generated data. Every few epochs we generated totally new data, gradually incrementing the number and scale of generated objects. We were guided by the intuition that a good practice to avoid overfit is generating more complicated dataset every few epochs. In other words, each time the model is performing well on the dataset we challenge it harder.

In our research process, during first training phazes, we noticed that many times our training session reached a plateau in terms of minimizing the Loss.

We put efforts to support more TensorFlow Callbacks allowing visualization of weight distribution in each layer during training.

Unfortunately, results can only be shown with the initialization weights, and their evolution over time could not be monitored.

We eventually found out that due to poor implementation design pursuing this thread would be too time consuming.

Hyper-Parameters Analysis

We believe it’s vital to elaborate on optional configurations that affect this network. This section expresses the vast domain of possible twicks and fine-tunings.

RPN architecture

- `RPN_ANCHOR_SCALES:` We put some thoughts in the size we want our anchors to be, since a better set of anchor sizes, one that statistically fits most of the objects, should lead to a slightly better convergence of *rpn_bbox_loss*.
- `RPN_ANCHOR_RATIOS:` Ratios are sometimes also a consideration to improve bbox_loss, for example, training on stems should require high and thin anchors. For some implementational reasons, adding ratios instead of replacing them leads to dimension mismatches. Pursuing this thread could be interesting since thin objects and thin parts in general seem challenging according to our experience and to some related topics on the internet.
- `RPN_NMS_THRESHOLD:` Intersection percent over which two ROIs are considered one single object and only one is kept. We can't determine exactly what is better: Values closer to 1 filter out best matches, therefore converge faster at the beginning, or smaller values that slow down convergence at the beginning but might produce better results since the learning curve is slower.
- `RPN_TRAIN_ANCHORS_PER_IMAGE:` The final number of ROIs to consider on a single forward pass. The ROIs are randomly selected from the result of the RPN. It's important to realize that this HyperParameter is coupled with `RPN_NMS_THRESHOLD` since, the higher the threshold, the more diverse ROI proposal is. High NMS will be more likely to choose dispersed anchors to train on. This should represent better the underlying distribution of objects in the scene.

Masking network

- `USE_MINI_MASK:` This boolean is for faster performance at the cost of masks

  accuracy.

  Empirically we can state that turning on this variable had a significantly bad effect on our masks quality. We always prefered to turn it off.

General

- `LEARNING_RATE:` We explored few approaches:
  - Extra small learning rate (micro scale)
  - A big learning rate(~0.1)
  - Incremental approach where in each epoch we reduce learning rate. We've

  found out that high learning rates lead to poor train results. Extra small learning rate had no significant benefit over a decaying learning rate that starts from 0.001. Therefore, we decided to move on with our incremental approach.

Workflow

Here we describe the flow of actions from getting a problematic dataset to training and testing the model.

Data Viability

1. Quickly inspect the json to check it is formatted correctly. In our case some of the files had the wrong order of segmentation values. We used a python script[4] to correct the data.
2. Visually check the sanity of the data by opening it in our *COCO-Annotator*[5]*.*
3. **Correct the data if needed**. If your dataset is small, and you find a significant pitfall, the right tools will get you through it very fast, and save you time later trying to bypass those.

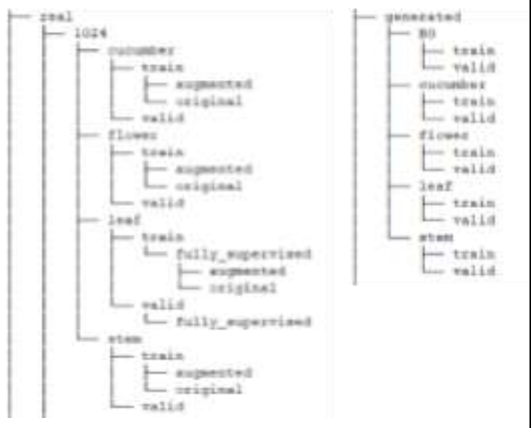
Data Organization

1. Split the dataset to train and validation sets, we made a script for this as well.
2. Resize the dataset pictures if needed. We could not train on our 4000x6000 original pictures therefore we resized all of them to 1536x1024 and 768x512 using the *altered Augmentor*[6].
3. Extract the single objects for synthetic image generation. Keep track of object created from the training set and those created from the validation set. This step as well is assisted by a script. This step is followed by an optional manual filtering step.
4. Keep the dataset organized in a practical fashion. We used the layout you can see in figure 18 to organize the datasets. Take in consideration that the data generating class we created assumes this layout in order to properly work.

Data Augmentation

1. For real supervised dataset, use the altered augmentor to sample a dataset big enough. We sampled 2000 pictures from every original dataset.
2. For generated dataset, tune the object scenery parameters according to your understanding. The parameters are depicted in the chapter about synthetic datasets.

[4] The script is available in the altered COCO-Annotator repository
[5] Instructions on how to do that is found in the repository
[6] Instructions on how to use the Augmentor scripts can be found in the repository



```
├── real
│   ├── 1024
│   │   ├── cucumber
│   │   │   ├── train
│   │   │   │   ├── augmented
│   │   │   │   └── original
│   │   │   └── valid
│   │   ├── flower
│   │   │   ├── train
│   │   │   │   ├── augmented
│   │   │   │   └── original
│   │   │   └── valid
│   │   ├── leaf
│   │   │   ├── train
│   │   │   │   └── fully_supervised
│   │   │   │       ├── augmented
│   │   │   │       └── original
│   │   │   └── valid
│   │   │       └── fully_supervised
│   │   └── stem
│   │       ├── train
│   │       │   ├── augmented
│   │       │   └── original
│   │       └── valid
├── generated
│   ├── BG
│   │   ├── train
│   │   └── valid
│   ├── cucumber
│   │   ├── train
│   │   └── valid
│   ├── flower
│   │   ├── train
│   │   └── valid
│   ├── leaf
│   │   ├── train
│   │   └── valid
│   └── stem
│       ├── train
│       └── valid
```

Figure 18 - Generated layout has no resolution layer since all objects and backgrounds are compatible with every synthetic image resolution.

Training

1. Read the repository readme and just start your training!
2. Have fun!

Experiments and Results

Metrics

IoU

Intersection over union, a method to quantify the percent overlap between target mask and prediction output.

Plot overlap matrix

Show number of predictions vs actual instances.
Show prediction confidence.
Show IoU for each prediction .

Precision recall graph

sorted Precision* as function of recall

Precision - Number of true positives over total classifications $\frac{True\ Positive}{True\ Positive + False\ Positive}$

Recall - Number of true positives over total true instances $\frac{True\ Positive}{True\ Positive + False\ Negative}$

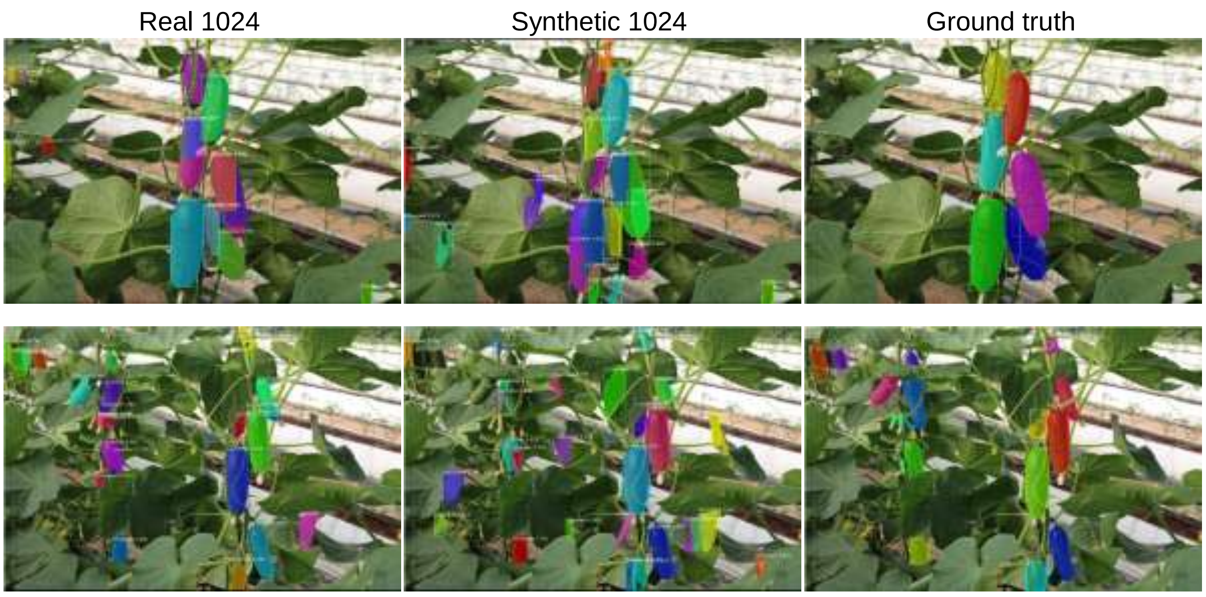
Fruit only - real vs synthetic

Definition of class - foreground mature fruits, occlusion omitted.

Mean AP - IoU=50						
Type	Resolution	Train set	Original	4000	lean	
Synthetic	512	Fruit only	0.38	0.07	0.36	0.06
	1024	Fruit only	0.42	0.09		
	1024	Mixed	0.38	0.42		
Real	512	Fruit only (2000)	0.53	0.51	**0.65**	**0.59**
	1024	Fruit only (2000)	0.70	0.56		
	1024	Fruit only (10,000)	0.71	0.70		

Mean AP - IoU=20						
Type	Resolution	Train set	Original	4000	lean	
Synthetic	1024	Fruit only	0.44	0.09	0.37	0.06
	1024	Mixed	0.38	0.44		
Real	1024	Fruit only (2000)	0.73	0.56	**0.725**	**0.63**
	1024	Fruit only (10,000)	0.72	0.70		

Side note about synthetic data and high resolution testing: Earlier models trained with much bigger object sizes reacted very much better on the high resolution pictures. Nevertheless the real dataset training set makes the model much more robust to resolution change.

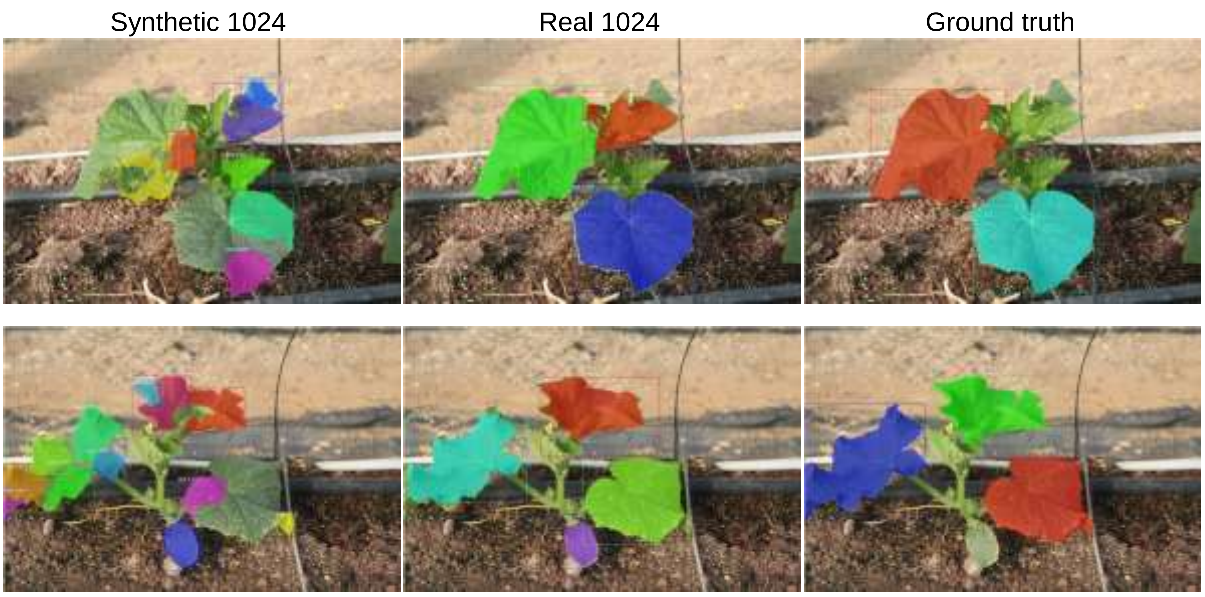


Leaves only - real vs synthetic

Definition of class - foreground mature leaves, occlusion omitted.
The original dataset is only partially supervised making training on a real dataset problematic. We could just ignore the fact that it is partially supervised but the problem would then be testing the results. Testing is tricky because the test set have to be be fully supervised in order to give a correct precision and recall grades.
For this purpose we used the COCO-annotator to supervise a very small dataset (8 training pictures, and 2 validation pictures).

Mean AP - IoU=50						
Type	**Resolution**	**Train set**	**Original**	**4000**	**lean**	
Synthetic	**512**	**Leaf only**	1.0	0.0	0.33	0.0
	1024	**Leaf only**	0.0	0.0		
	1024	**Mixed**	1.0	0.0		
Real	**512**	**Leaf only (2000)**	1.0	0.0	**1.0**	**0.38**
	1024	**Leaf only (2000)**	1.0	0.77		

Mean AP - IoU=20						
Type	**Resolution**	**Train set**	**Original**	**4000**	**lean**	
Synthetic	**1024**	**Leaf only**	0.48	0.0	0.74	0.07
	1024	**Mixed**	1.0	0.14		
Real	**1024**	**Leaf only (2000)**	1.0	0.77	**1.0**	**0.77**



The results are not very informative because of the small size of the validation set. Anyway the bias in favor of real datasets stays the same.

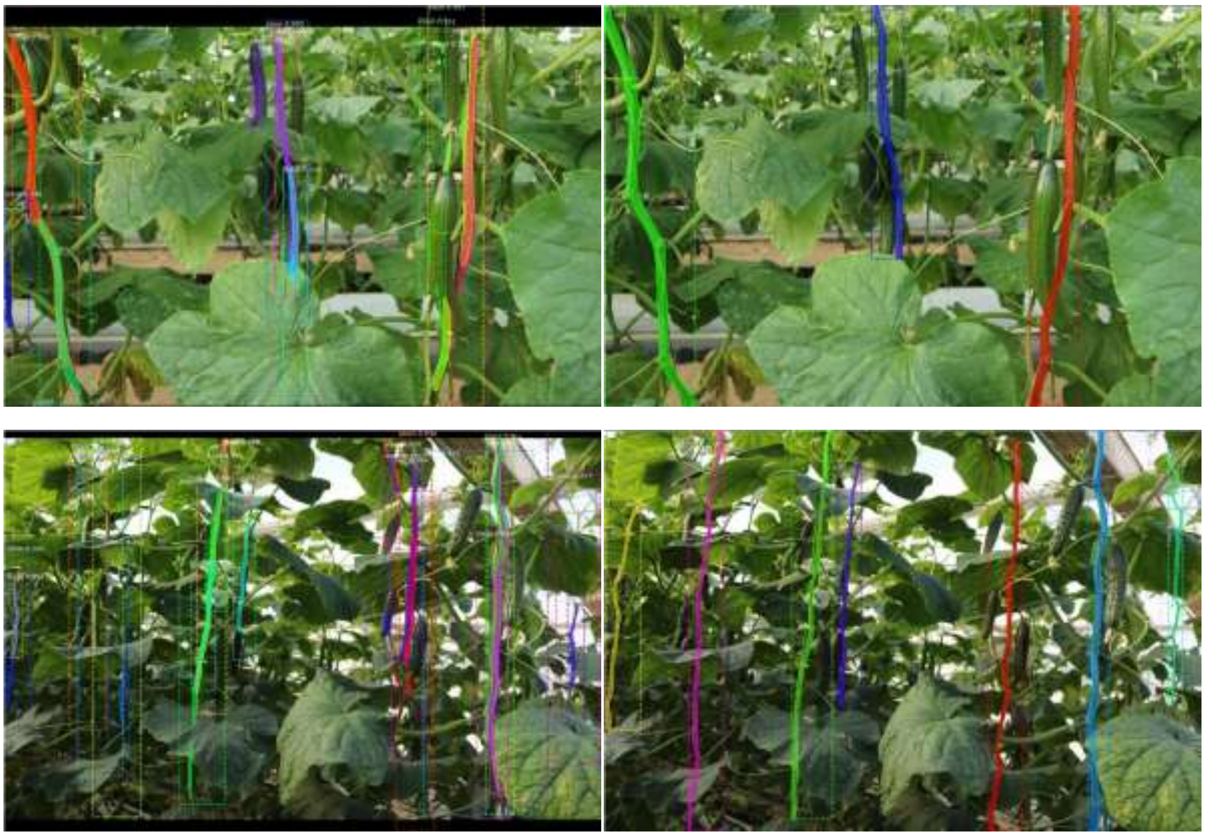
Stems only - real vs synthetic

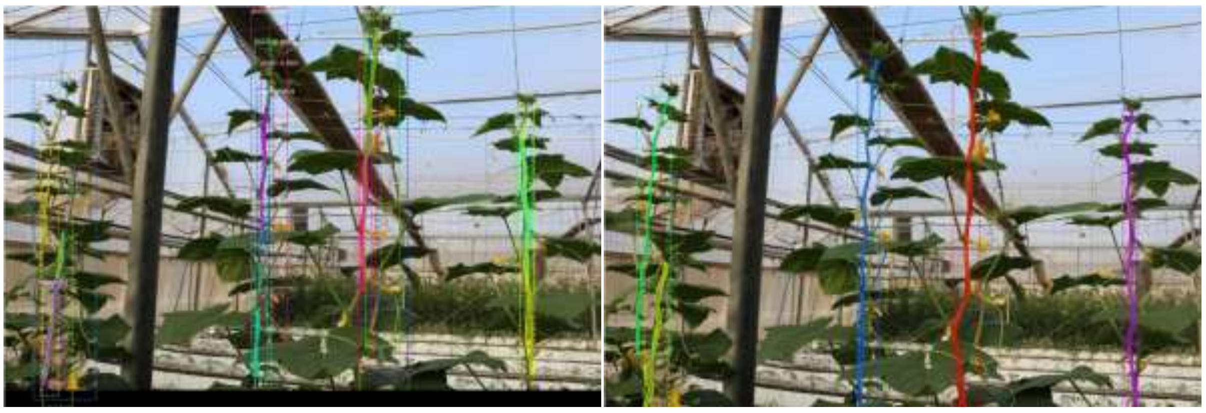
Definition of class - Foreground main stem, occluded location included.

Mean AP - IoU=50						
Type	Resolution	Train set	Original	4000	lean	
Synthetic	512	Stem only	0.0	0.01	0.01	0.01
	1024	Stem only	0.03	0.01		
	1024	Mixed	0.0	0.0		
Real	512	Stem only (2000)	0.05	0.01	**0.08**	**0.045**
	1024	Stem only (2000)	0.11	0.08		

Mean AP - IoU=20				
Type	Resolution	Train set	Original	4000
Synthetic	1024	Stem only	0.24	0.22
	1024	Mixed	0.01	0.002
Real	1024	Stem only (2000)	0.62	0.54

Real 1024 Ground truth





The model trained on the real dataset is not particularly wrong with its decisions, but the limited aspect ratio seems to be responsible for splitting the stems. Split detections are not counted as true positives thus the very small precision.

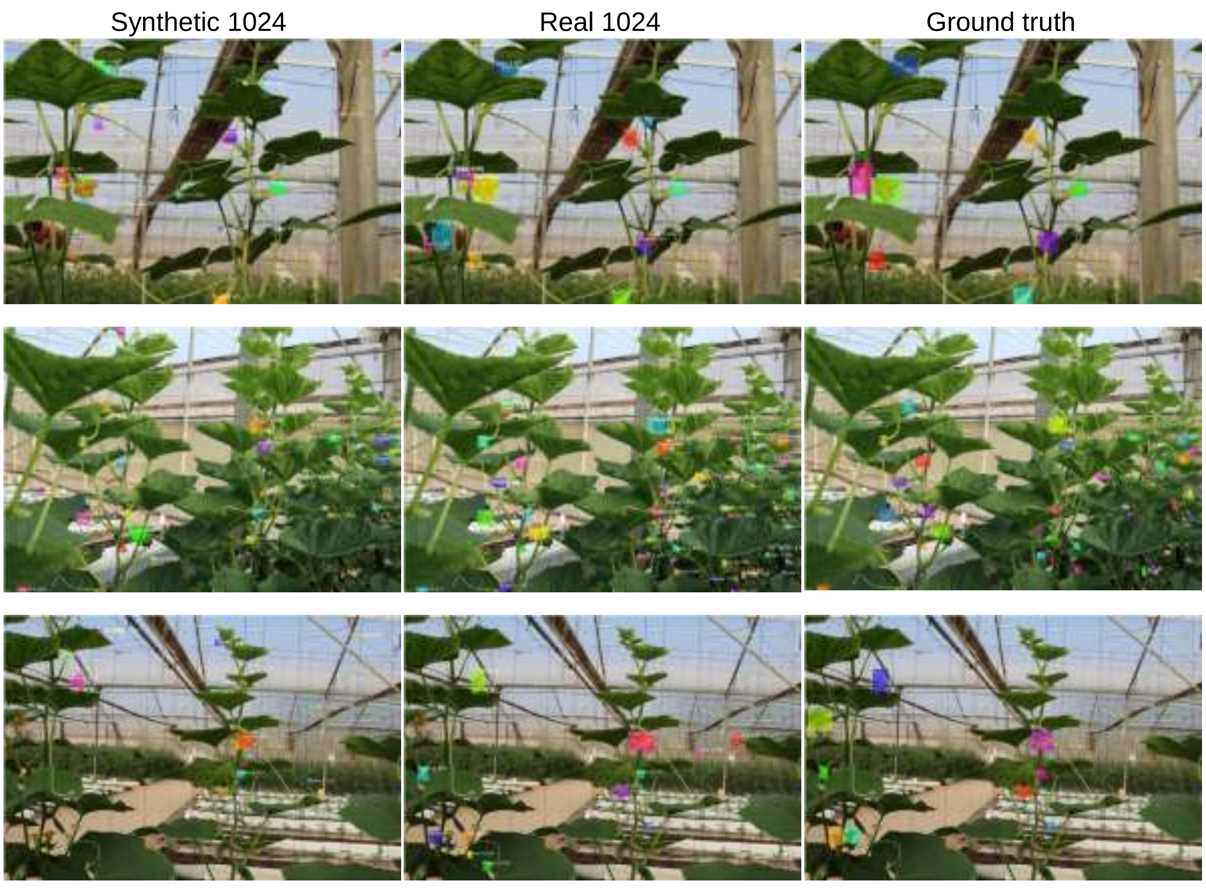
On the other hand the model trained on synthetic data does not perform well at all on stems.

Flowers only - real vs synthetic

Definition of class - Foreground main stem, occluded location included.

Mean AP - IoU=50						
Type	**Resolution**	**Train set**	**Original**	**4000**	**lean**	
Synthetic	**512**	**Flower only**	0.22	0.06	0.17	0.05
	1024	**Flower only**	0.20	0.07		
	1024	**Mixed**	0.09	0.04		
Real	**512**	**Flower only (2000)**	0.43	0.30	**0.45**	**0.29**
	1024	**Flower only (2000)**	0.47	0.28		

Mean AP - IoU=20						
Type	**Resolution**	**Train set**	**Original**	**4000**	**lean**	
Synthetic	**1024**	**Flower only**	0.30	0.12	0.23	0.14
	1024	**Mixed**	0.16	0.15		
Real	**1024**	**Flower only (2000)**	0.64	0.56	0.64	0.56



Enhanced blending vs sharp blending

Mean AP - IoU=50					
	Cucumber only	**Stem only**	**Mixed on flower**	**Mixed on cucumber**	**Mean**
Erode and Blur	0.42	0.03	0.07	0.29	**0.2025**
No blending	0.41	0.001	0.01	0.22	0.16025

Mean AP - IoU=20					
	Cucumber only	**Stem only**	**Mixed on flower**	**Mixed on cucumber**	**Mean**
Erode and Blur	0.45	0.24	0.14	0.30	**0.2825**
No blending	0.43	0.12	0.02	0.23	0.2

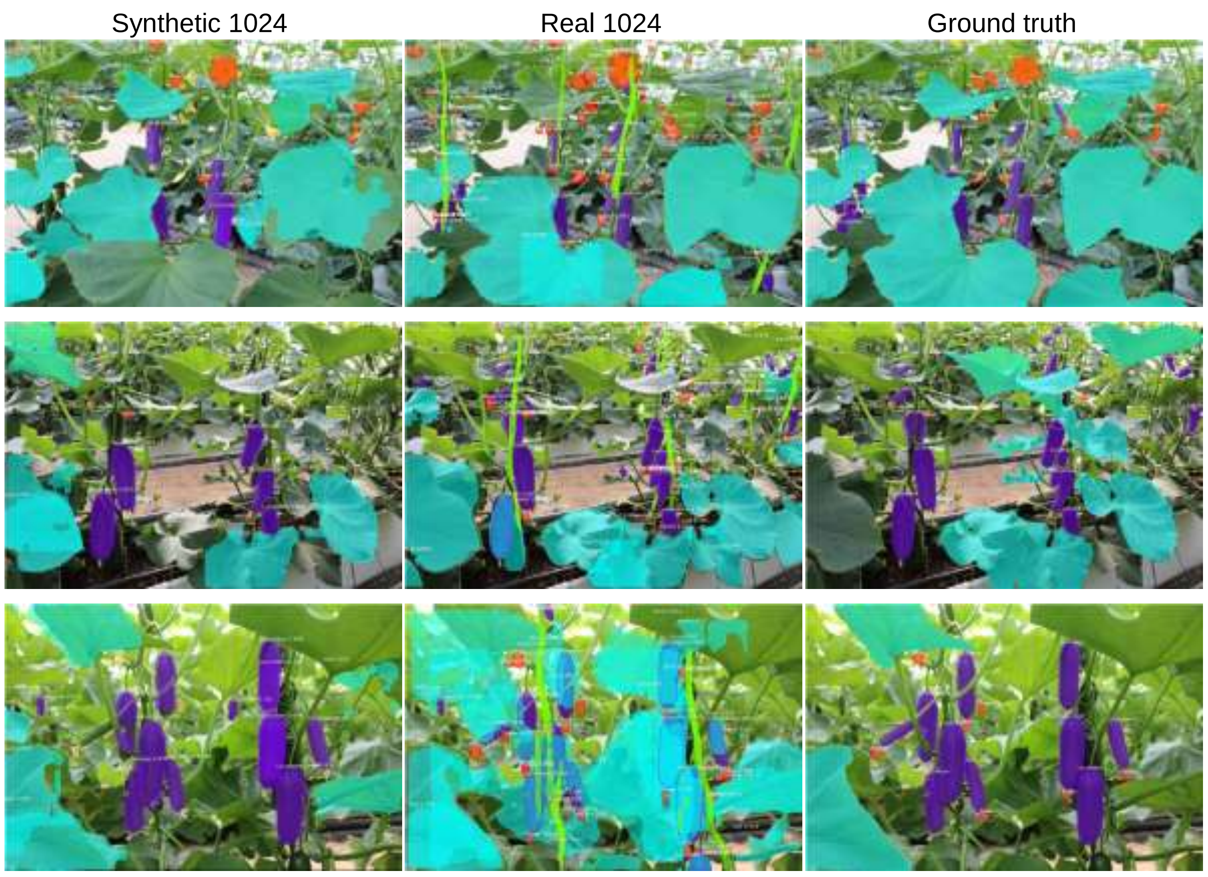
Blur and erosion properties clearly influence the result, and should be optimized.

Final results - Team real and team synthetic on a test set

To produce these final results on the real data team we inferenced a validation set through the best model of each class, and then combined the results to a single picture, colorized by class.

Mean AP - IoU=50			
Type	**Resolution**	**Train set**	**Original**
Synthetic	**1024**	**Mixed**	0.29
Real	**1024**	**Single models combined**	**0.42**

Mean AP - IoU=20			
Type	**Resolution**	**Train set**	**Original**
Synthetic	**1024**	**Mixed**	0.33
Real	**1024**	**Single models combined**	**0.55**



It is clearly notable that the models react very well to singled out objects, and has a hard time with crowds. Moreover leaves are more easily recognized when shot from above, as they appear in the training set.
Overall the models seem to mostly do their job.

Notes on the Results

- Due to hardware limitations, inference on 6000x4000 pictures need to apply cropping. Mask-RCNN has to apply an image patching approach, that causes big objects to partially appear in each patch, thus reducing IoU. This causes Mean-AP to decrease, since many maskings don't cross the 50% IoU threshold.

- The results suggest that mixed-object models perform worse than single-object models. By analyzing our research containers and observing generated dataset for the model, we understood that our generator generates up to 20 objects, no matter what they are. Thus the richer the object distribution got, the less actual objects from each type appeared.
- The only viable results are the results on the fruits and the stems.
  Leaves - Only a tiny test set is fully supervised therefore do not represent real life results.
  Flowers - Real dataset is trained on the box style flowers and performs perfectly, but the synthetic models use 3 manually cut flowers therefore this is not a fair fight.
- Objects scenery may have a big influence on the results. The results just point out that this may be a difficult task.

Conclusions

- Mask R-CNN is very well fitted for fruit and flower detection. Detecting leaves and stems in a dense or crowded environment of a lot of the same makes the detections very challenging. Moreover it is much more difficult to clearly define the objects we are trying to segment and the ones we are not (as explained in the dataset section).
- Training on generated data does not make sense when you have the tools to fully supervise a small training set. The real dataset used to train the leaf model in the final results is made of only 8 pictures (traditionally augmented to 2000 picture). It could be interesting to try to beat the real datasets with synthetic data, by changing the blending properties, the object scenery and maybe add additional tricks. Nevertheless the results are clear on this point - **You are better off fully annotating a dozen of pictures than spending hours in creating the perfect synthetic data.**

Further Work and Suggestions

- Smart synthetic plants - spreading leaves and cucumbers around a stem.
- Transfer learning of masks between classes (e.g for flowers masking).
- Real time inference - run real time inferences while keeping track of object indexation. Keep track of evolution for each leaf or cucumber as well.
- LSTM or any RNN concept to improve object detection (flower turns to cucumber, stems grows up, etc).
- Use a 3D engine platform to generate data. Create infinite angles, curvature and lighting of leaves, using a single leaf texture.
- Background generative models. Generate backgrounds instead of using a finite bank of backgrounds. Possibly this would better mimic real supervised pictures.

- Investigate the loss of the model for very thin objects and thin parts, and try to propose a better performing mask detector for them.
- Further improve and adopt the *COCO-annotator* for easy, fast and cheap annotations. Put effort in tablet aspects and usage.
- Further improve the *altered Augmentor*. Add HSV color alteration, maybe incorporate the png-to-json script inside for seamless augmentation.
- Localize intersecting pictures in datasets and combine their annotations.

References

Mask R-CNN paper - https://arxiv.org/abs/1703.06870

Faster R-CNN paper - https://arxiv.org/abs/1506.01497

Matterport Mask R-CNN - https://github.com/matterport/Mask_RCNN Dima

repository of Mask R-CNN -

Original COCO-annotator - https://github.com/jsbroks/coco-annotato r

Original Augmentor - https://github.com/mdbloice/Augmentor

COCO dataset - http://cocodataset.org/#home

COCO annotations from scratch - http://www.immersivelimit.com/tutorials/create-coco-annotations-from-scratch Building a custom mask r-cnn - https://towardsdatascience.com/building-a-custom-mask-rcnn-model-with-tensorflow-objectdetection-952f5b0c7ab4