Clayton Haley
December 14, 2020

Peanut Pod Research Report

This semester, I have had the opportunity to emerge into the subject of Deep Learning. I have learned various topics within the subject area. Now, I have the chance to share my progress and document my results for the Peanut Pod Research. There is still so much more that could be done with the project. Tuning the training model, for example, making changes to parameter values and the training data itself may affect the results of the training model, ultimately affecting the inference results.

Step 1: CVAT Annotations

I will start from the beginning of the process: creating the data sets (annotations). For each frame, it was necessary to adjust the brightness, contrast, and saturation in order to draw accurate polygons. If these settings are not adjusted, it will be difficult to see the pods, and the model might not be as accurate as we hope. There are two classes of peanut pods: Partial (Red), Complete (Green). I did not look into the classifications themselves, meaning I made assumptions on how to define each individual pod. For example, if one appeared to be fully grown (having both nuts in the shell), then it was marked as Complete. On the other hand, if the pod looked like it had only one nut, then it was identified as Partial. I believe that it was best to attempt to identify the two different classes so we can ultimately determine how many pods of each class are in each frame, then drawing an estimate for the total amount of peanut pods for the entire video clip.

Training

category	#instances	category	#instances
P	208	C	200
total	408		

Test

category	#instances	category	#instances
P	140	C	105
total	245		

Clayton Haley
December 14, 2020

Below, I have an example for each annotation (peanut pod) class:

Partial

Complete

Step 2: Train

The first step in training the model is setting the correct parameters. There are specific parameters of the training model that greatly impact the results of the model.

```
from detectron2.engine import DefaultTrainer

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("pods_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00125
cfg.SOLVER.MAX_ITER = 300
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 2

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

cfg.DATALOADER.NUM_WORKERS
Generally speaking, having one or two additional workers along with the main process increases the running time.

cfg.SOLVER.IMS_PER_BATCH
This attribute describes the number of images per iteration. In this model, there are 2 images for each.

cfg.SOLVER.BASE_LR
The learning rate is how quickly the model can arrive at the best accuracy. This number is between 0.0 and 1.0.

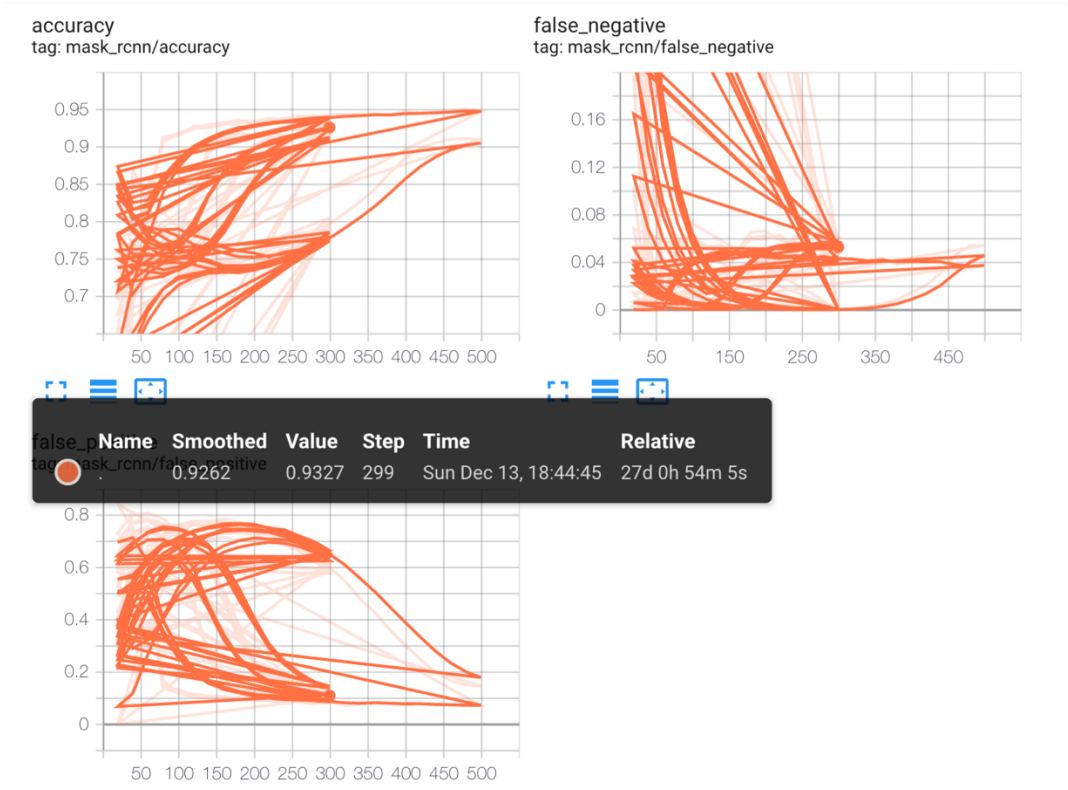
Clayton Haley
December 14, 2020

cfg.SOLVER.MAX_ITER:
The maximum number of iterations for training. 300 iterations are reasonable for this small dataset. Larger datasets require longer training.

cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE
The number of elements selected for each image. This can also be referred to the number of ROIs (regions of interest). The value of 128 is faster and better for this small dataset.

cfg.MODEL.ROI_HEADS.NUM_CLASSES
The number of classes. For this model we have 2: Partial, Complete.

Below are the obtained Mask R-CNN results. The graphs for each metric appear to be very confusing and abnormal. Although the accuracy is high at 93%, we still want to see a smoother and more visually pleasing graph. There are a few ways that we can improve these results. One option is to divide the images in the training set into chunks. If the image has a lot of empty space without annotations, it may be harder to detect the pods. Smaller images with larger annotations will help in achieving smoother and higher accuracy results. In regard to splitting the data into chunks, we could also try to crop out each individual annotation by their bounding boxes. This will make for much smaller data images and will most likely lead to better results. To obtain a more accurate training model, we can also add more annotations. This is will help the model learn from more instances, which will allow it to be more accurate in testing.



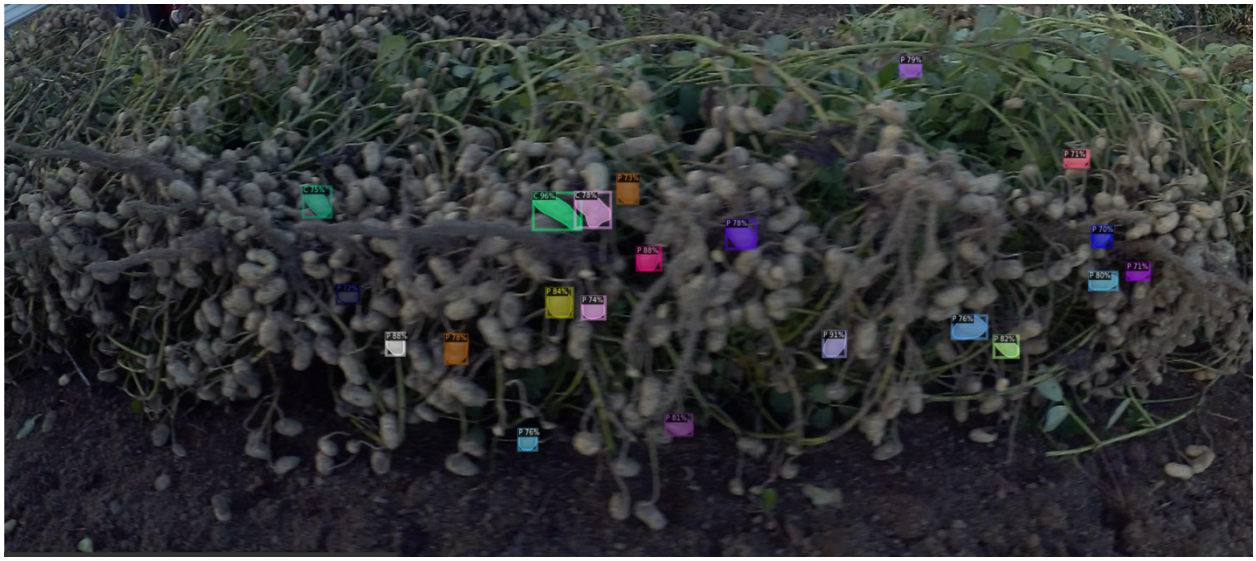
Clayton Haley
December 14, 2020

Step 3: Test

The next and last step in the object detection process is to test our model on a new data set. The code below displays the necessary attributes we need to include for testing. "SCORE_THRESH_TEST" is used to suppress the bounding boxes that have IoU (Intersection over Union) values less than the given value. In this case, I used 0.7.

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
predictor = DefaultPredictor(cfg)
```

The image below shows the results for its specific instances. We can see that the model was successful in obtaining results for each instance in the image, but the accuracy results vary across the image. This is just an example. There are many other images in the data set that may only have 1-3 successfully detected objects. Some might have zero. This issue relates back to the training process. It shows how important training the model is. In order to prevent issues in testing, we must strive for better training and testing results by implementing chunking or adding more annotations.



Clayton Haley
December 14, 2020

I will introduce the inference results for testing. AP is identified as “mean average precision”. AP50 means that the AP is computed at a single IoU of 0.50. The results for bbox refer to the bounding boxes that surround the annotations. Additionally, the results for segm refer to the segmentations (instances) that are in the bounding boxes. The testing model gives evaluation results for both, as well as the results for each category (P and C).

Evaluation results for bbox:

AP	AP50	AP75	APs	APm	APl
20.590	28.496	24.943	0.000	22.352	nan

Per-category bbox AP:

category	AP	category	AP
P	24.489	C	16.692

Evaluation results for segm:

AP	AP50	AP75	APs	APm	APl
21.403	29.759	26.493	0.000	21.459	nan

Per-category segm AP:

category	AP	category	AP
P	24.693	C	18.112

Unfortunately, the results show very low AP values across the board, but we cannot allow this to be discouraging. It must be motivating to learn how to improve these results. As I stated before, we can use data augmentation methods to improve the performance of this model. I am encouraged to dive deeper into the subject in order to gain the knowledge and skills to achieve the highest accuracy possible. It has been a great experience working on this project, and it does not have to end here. The goal from here on out is to increase the number of annotations, as well as finding a way to crop out each instance so it is easier for the model to learn faster and to be the most accurate.