A Project Report

On

Image Segmentation of Assembly Prototype

BY

SHAURYA VIJAYVARGIYA

2018A8PS0079H

Under the supervision of

TATHAGATA RAY

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

CS F376: DESIGN PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(OCTOBER 2020)

ACKNOWLEDGMENTS

This project wouldn't be possible without BITS Pilani and I am very grateful to them for granting me this opportunity.

I would like to express my thanks to the people who have helped me most throughout the project. I deeply appreciate my mentor and faculty in-charge Dr. Tathagata Ray for his guided support.

I am also thankful to my colleagues and friends who have helped me in making great progress throughout the project. I wish to thank my parents for their personal support and helping me continue all the way.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Image Segmentation of Assembly Prototype**" submitted by Mr. Shaurya Vijayvargiya (ID No. 2018A8PS0079H) in partial fulfillment of the requirements of the course CS F376, Design Project Course, embodies the work done by him under my supervision and guidance.

Date: 19-10-2020 **(TATHAGATA RAY)**

BITS- Pilani, Hyderabad Campus

CONTENTS

Title page….……1

Acknowledgements……..2

Certificate….……....3

1. Overview……..5

2. Detectron2……...…….……....6

3. Structure of Dataset...……...8

4. Training in Google Colab.……..10

5. Results…….....11

Scope for Development……...15

References…… …..15

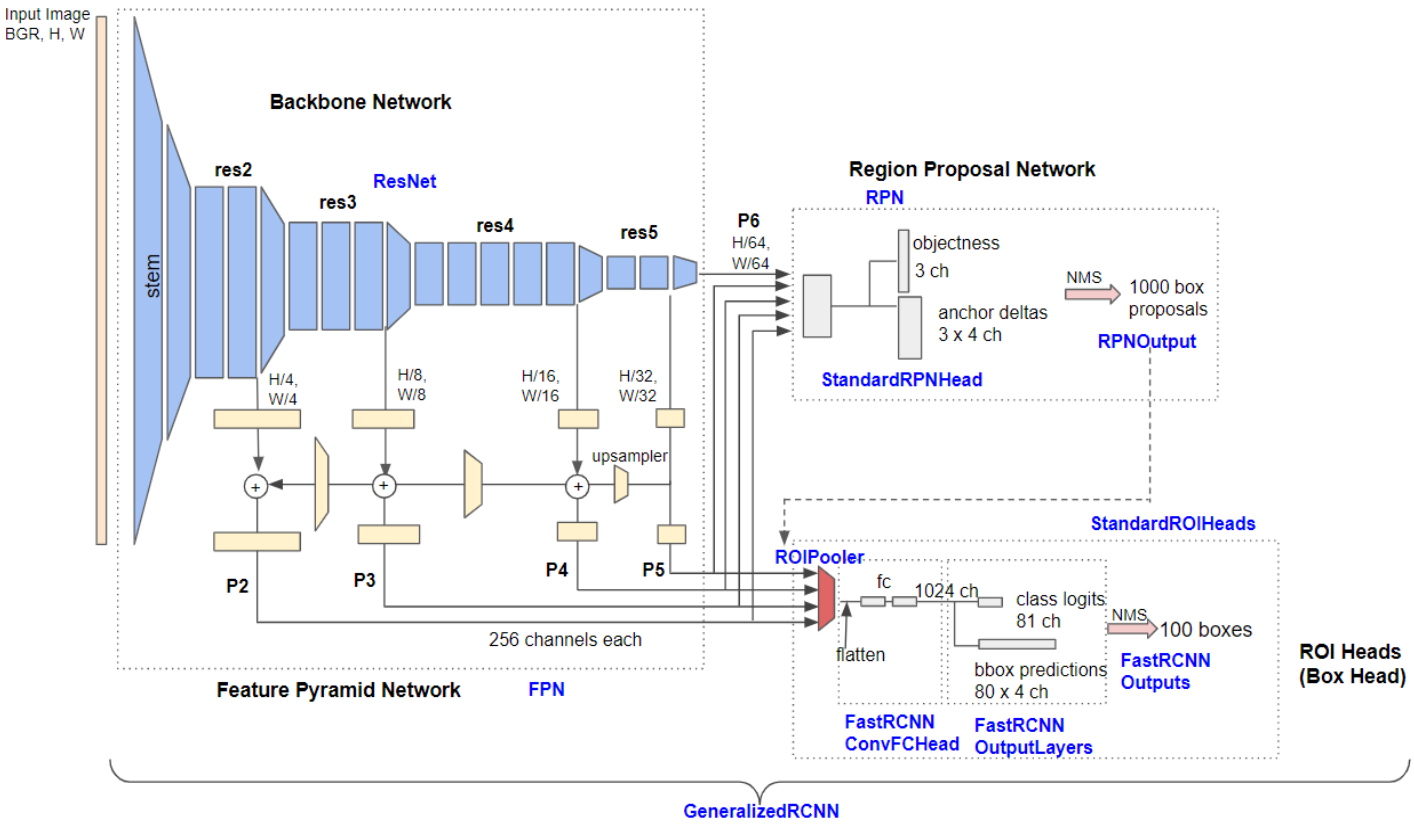
Overview

Image Segmentation is the process of partitioning an image into multiple segments. The main goal of segmentation is to have a better understanding of the image and where in the image the desired objects are. It locates objects and their boundaries by assigning a label to each pixel denoting which category it belongs to. It has applications in various fields and is one of the most interesting and challenging problems in Computer Vision.

This project does Image Segmentation on a Custom COCO dataset designed synthetically using foreground and background images. For training, Detectron2 is used. The model trains on a dataset of 10000 images and then predicts the instances on a few test images with accuracies as high as 95% in some cases.

Detectron2

The main backbone of this project is Detectron2. Detectron2 is a next-gen open source object detection system from Facebook AI Research. It is capable of working on state of the art models and produce astounding results such as bounding box detection, instance and semantic segmentation, and person key-point detection. It is powered by PyTorch deep learning framework. It has a very complex architecture and looks like the following image.



There are three main components in the image above –

a) **Backbone Network: It extracts feature maps from the input image at different scales. This network uses Resnets which makes use of residual networks. One of the main problems Resnets solve is vanishing gradient. For deep neural networks, the gradients from where the loss functions is calculated easily shrinks to zero after several applications of chain rule. This results on the weights never updating its values and hence no learning happens. Resnets use skip connections because of which gradients can directly flow through the skip connections backwards from later layers to initial filters.**

b) **Region Proposal Network: Detects object regions from the multi-scale features. 1000 box proposals (by default) with confidence scores are obtained.**

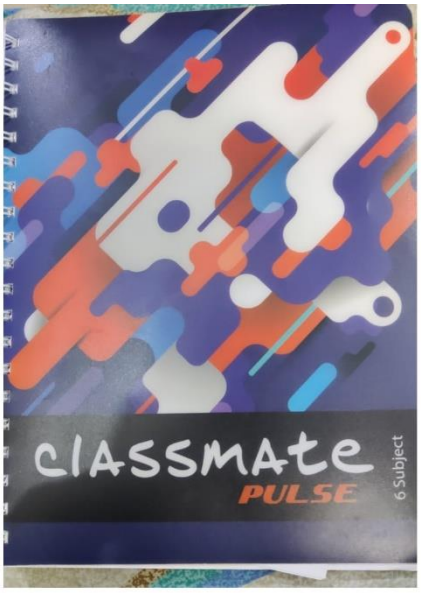
c) **Box Heads: Crops and warps feature maps using proposal boxes into multiple fixed-size features, and obtains fine-tuned box locations and classification results via fully-connected layers. Finally 100 boxes (by default) in maximum are filtered out using non-maximum suppression (NMS). The box head is one of the sub-classes of ROI Heads.**

Using this complex architecture, one can produce impressive results such as the following –

Structure of Dataset

The dataset is in the well known COCO dataset format. It is designed synthetically using foreground and background images and then randomly placing the foreground images on the background images to obtained the generated dataset. An example shown below explains the process better.

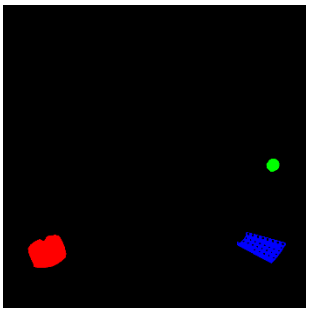
Background Image + Foregroud Images = Generated Image + Mask



(Tyre)

(piece1)

(bolts)



Dataset contains the training data which has 10000 of such generated images and masks and the validation set contains 300 pairs of such images. The categories (labels) are as follows –

1. **Parts**
    i. **Piece1**
    ii. **Piece2**
    iii. **Rod**
    iv. **Tyre**
2. **Screws**
    i. **Bolts**
    ii. **Nuts**

There are 2 supercategories and each has some subcategories.

The COCO annotations json file is generated which is fed into Detectron2 for training the generated dataset. This is the format of the annotations file

```
{
    "info": {...},
    "licenses": [...],
    "images": [...],
    "annotations": [...],
    "categories": [...], <-- Not in Captions annotations
    "segment_info": [...] <-- Only in Panoptic annotations
}
```

The masks of each image is used for the *annotations* section to store which category does each pixel belong to.

The *categories* section stores the categories of objects available in the dataset as shown above.

***Images* section contains the path of each image with their dimensions and unique id.**

Each image is of size 1024x1024 because the detectron2 algo requires images to be divisible by 2 atleast 6 times.

Training in Google Colab

Google colab is used for training the project because it provides powerful GPUs for computing the instances which is a very heavy task.

Walkthrough –

1. **Registering Dataset: Detectron2 requires one to register the dataset first in order to start training. The train and validation set are registered as shown below using the coco annotions file for each set.**

```
from detectron2.data.datasets import register_coco_instances

register_coco_instances("mechanics_train", {}, "/content/drive/My Drive/CS DOP/mechanics_dataset_2/train/coco_instances.json", "/content/drive/My Drive/CS DOP/mechanics_dataset_2/train/images")
register_coco_instances("mechanics_val", {}, "/content/drive/My Drive/CS DOP/mechanics_dataset_2/val/coco_instances.json", "/content/drive/My Drive/CS DOP/mechanics_dataset_2/val/images")
```

2. **Training the Model: The model is trained using the mechanics training dataset. Pretrained weights are used. In this case weights of Mask_RCNN's FPN trained on the ImageNet dataset are used. Number of classes of ROI Heads is set to 6 because there are 6 categories in the dataset.**

```
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
import os

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("mechanics_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 = 0.00025
cfg.SOLVER.MAX_ITER = 2000
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 6

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
```

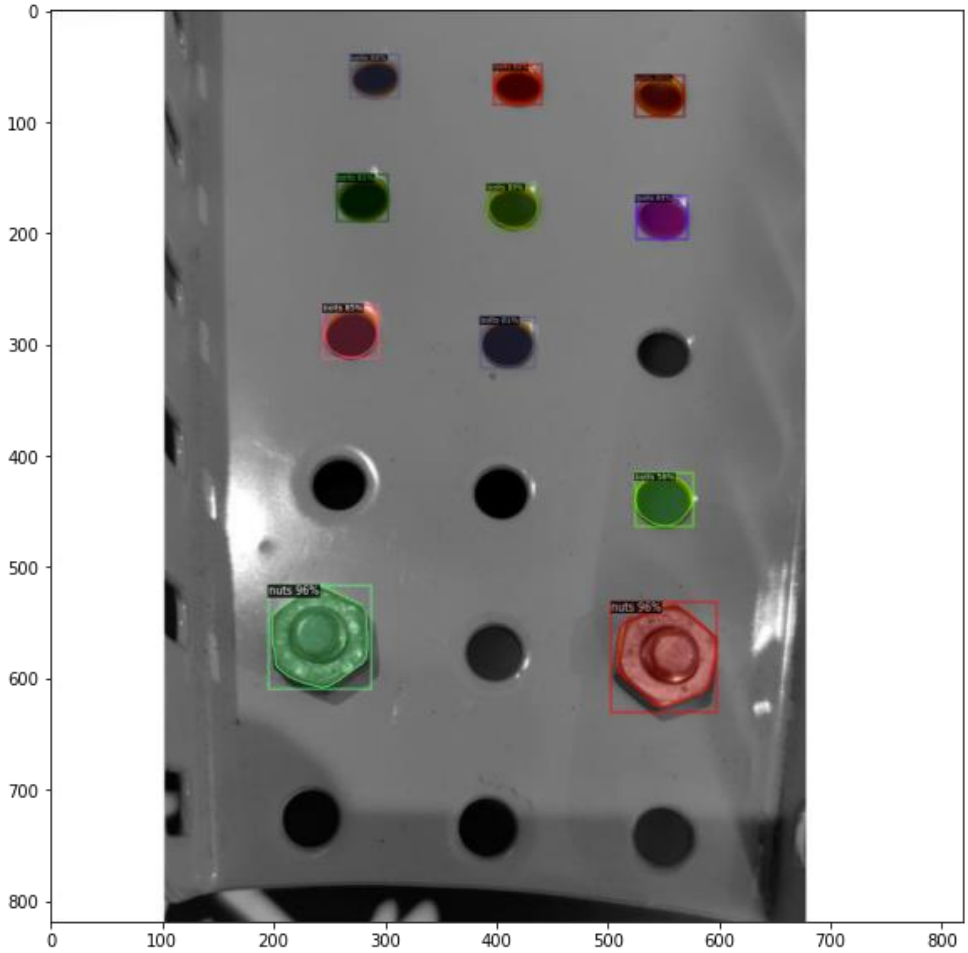
```
trainer.resume_or_load(resume=False)
trainer.train()
```

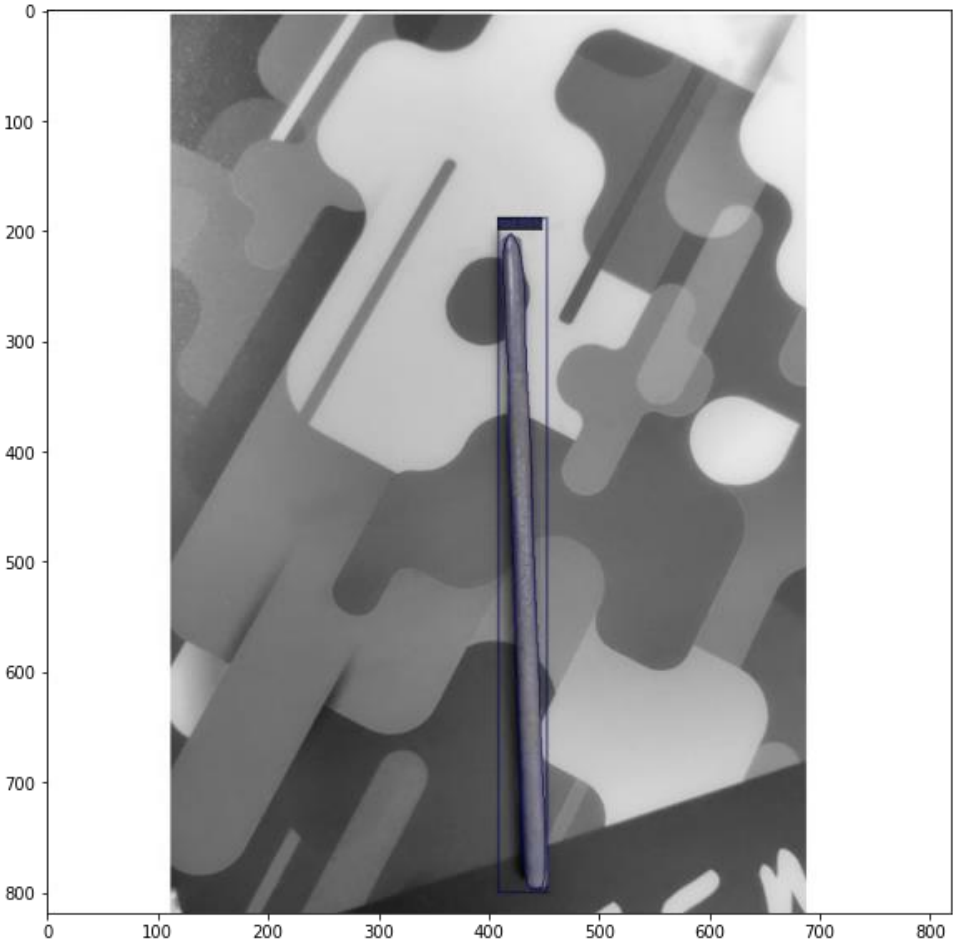
Once the training is done the final training time and speed are shown:

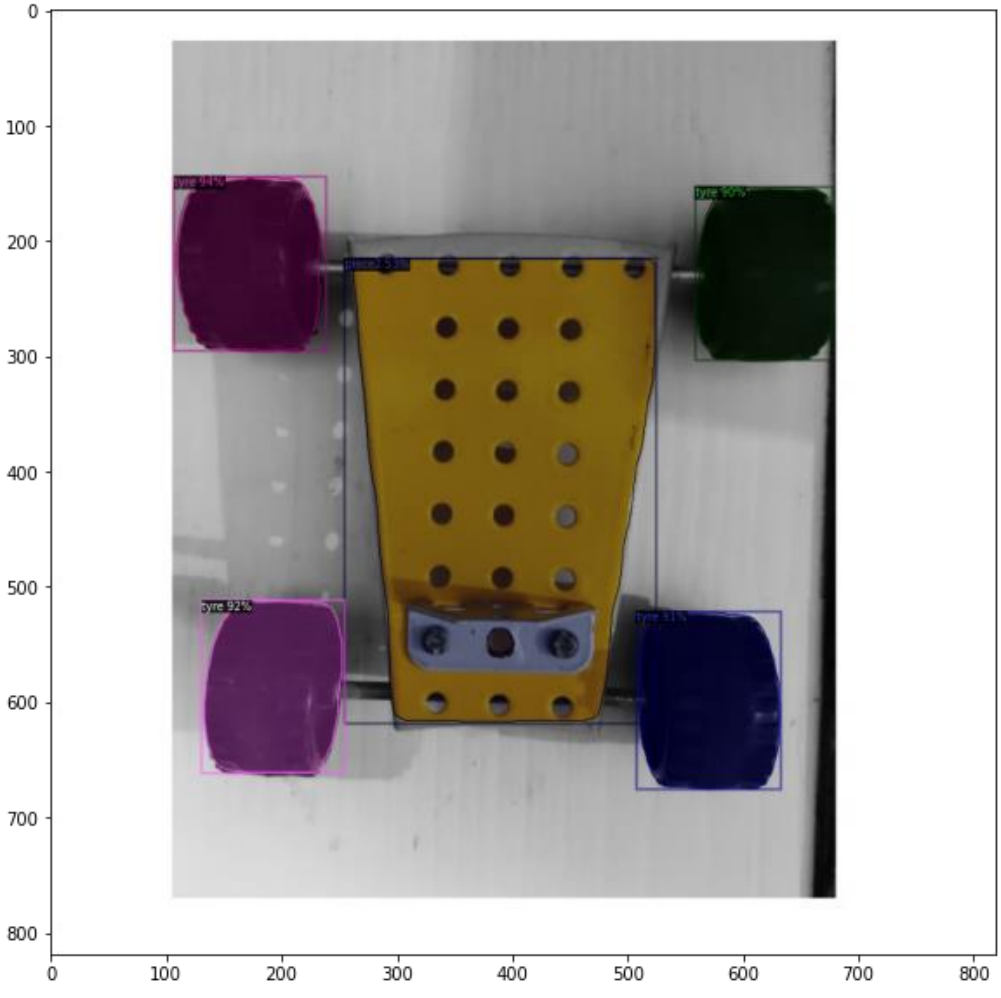
```
[10/13 16:54:36 d2.engine.hooks]: Overall training speed: 1997 iterations
in 0:37:06 (1.1147 s / it)
[10/13 16:54:36 d2.engine.hooks]: Total training time: 0:37:09 (0:00:03
on hooks)
```

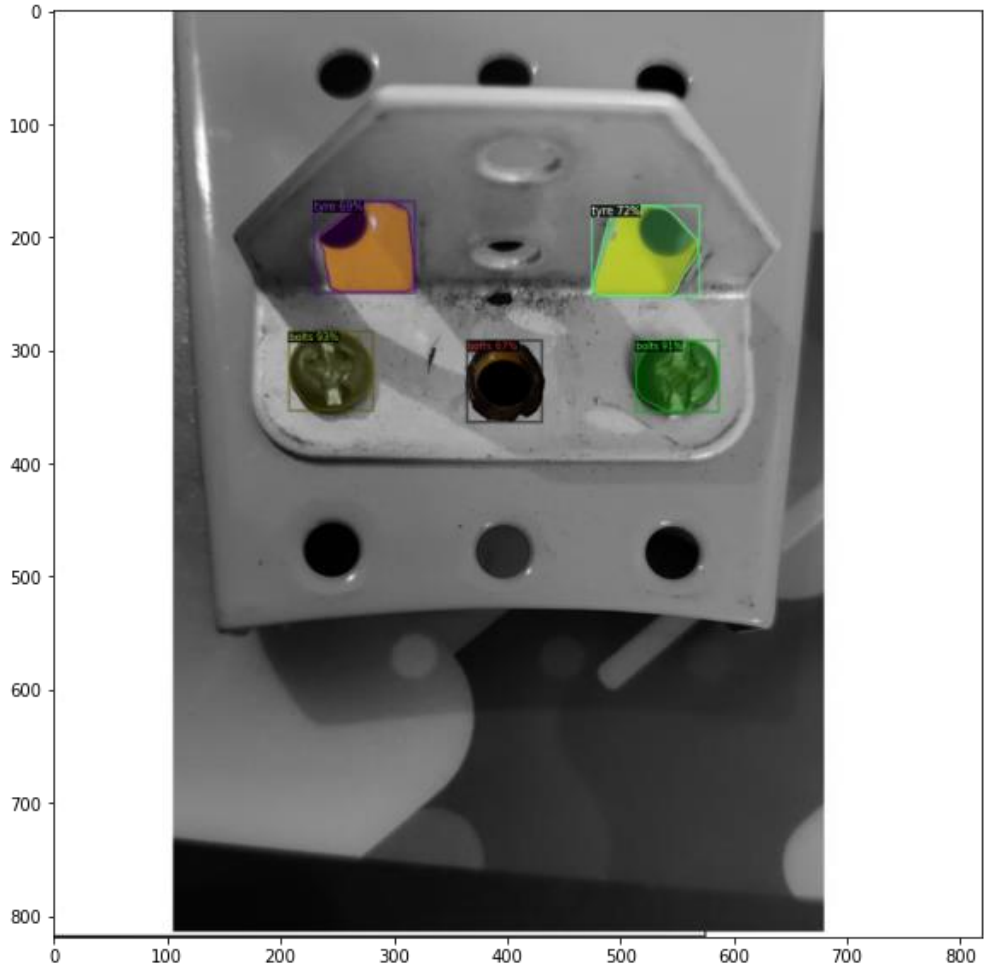
Results

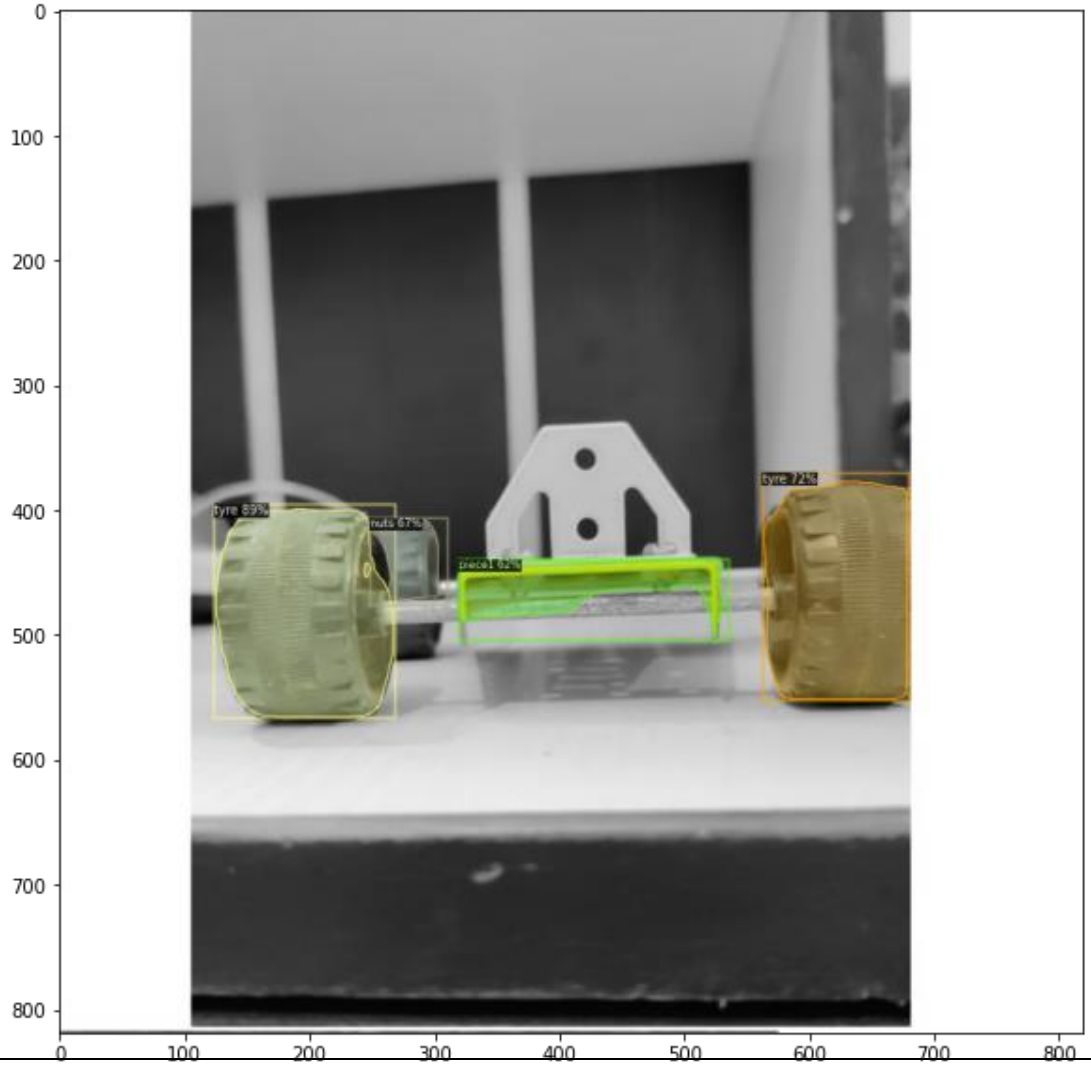
The following images are obtained after using the trained model for inference on the test dataset –

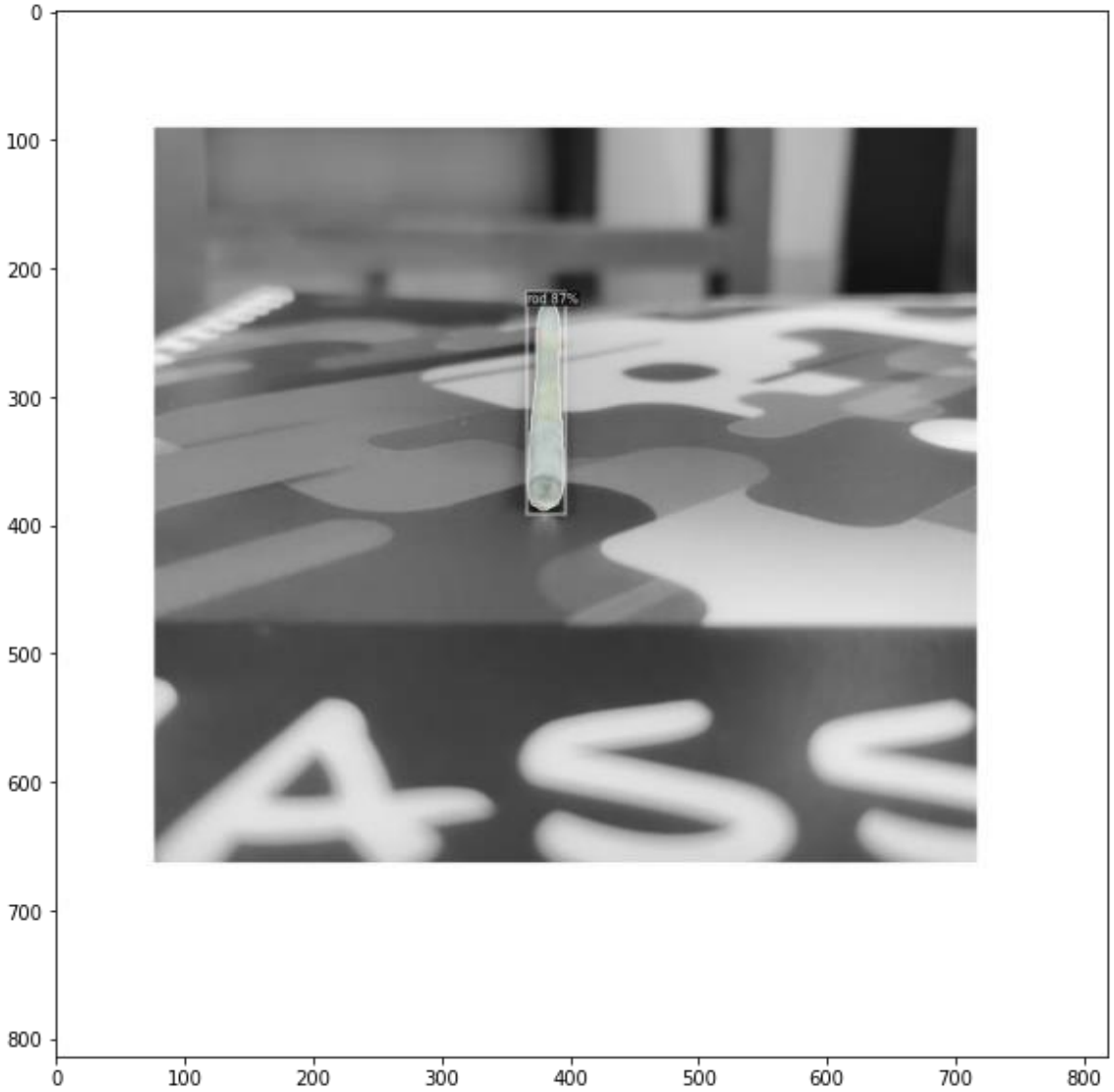


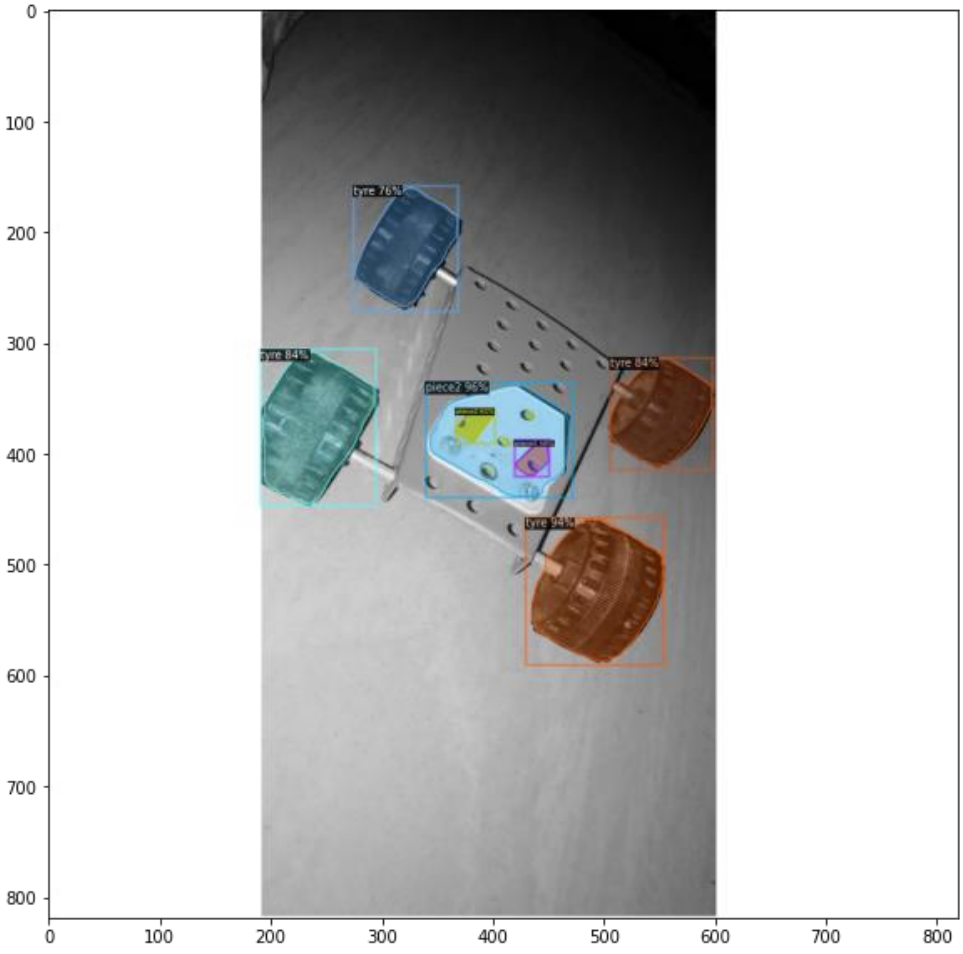

0
100
200
300
400
500
600
700
800
0
100
200
300
400
500
600
700
800


0
100
200
300
400
500
600
700
800
0
100
200
300
400
500
600
700
800


0
100
200
300
400
500
600
700
800
0
100
200
300
400
500
600
700
800


0
100
200
300
400
500
600
700
800
0
100
200
300
400
500
600
700
800


rod 87%


tyre 76%
tyre 84%
piece2 99%
tyre 84%
tyre 94%

Further Scope of Development

- **Developing better models which detect partially visible objects due to occlusion by other objects or truncation by the image boundary.**
- **Look into layering effect (how segmentation will work when layering is done)**
- **Cross checking detected object instances for edges using edge detection algorithms, finding exact dimensions using the respective CAD models.**

References

- https://github.com/akTwelve/cocosynth
- https://github.com/facebookresearch/detectron2
- https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd