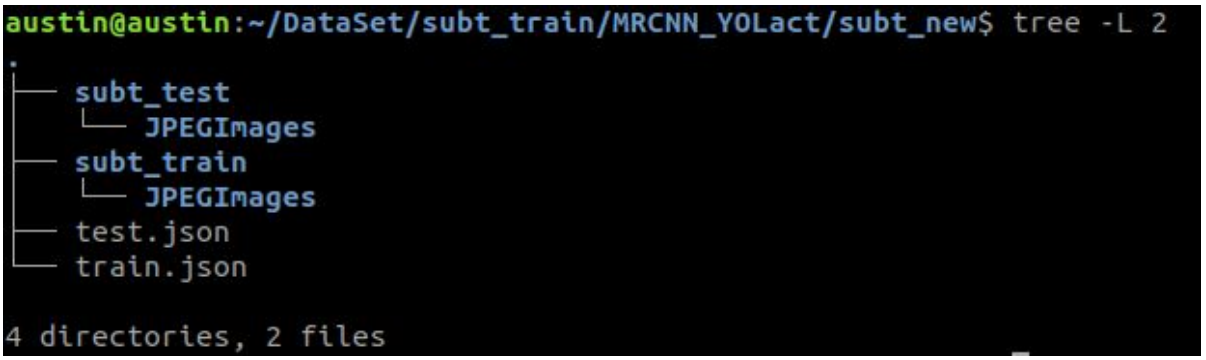
This tutorial is used to teach how to create your own datasets and training model.

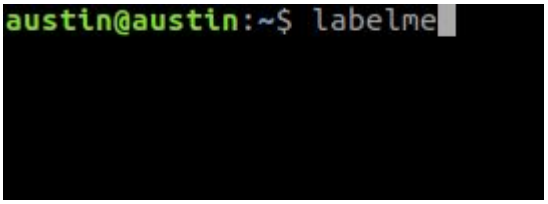
1. MaskRCNN

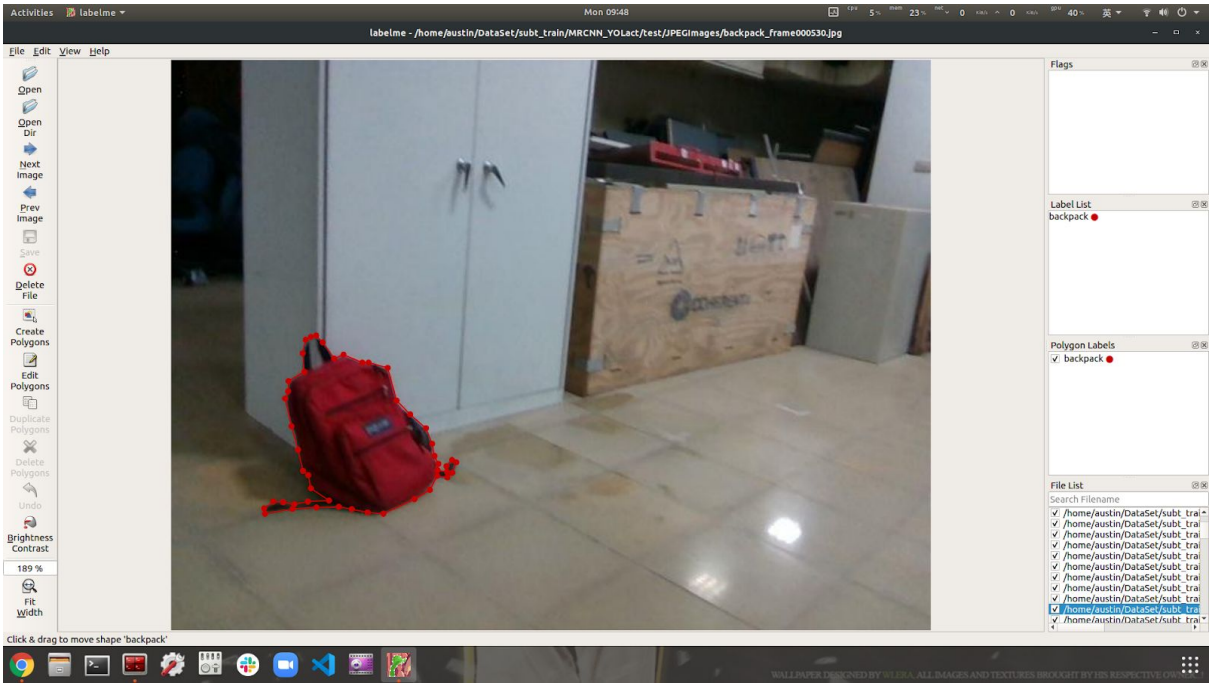
This model use a form of datasets called "coco-datasets" , it's structure is looked like this:



The folder "subt_train" and "subt_test" contain the RGB image folder "JPEGImages" for training and testing , train.json and test.json are the json file that have the information of each RGB image , for example , the location of the mask.

To create coco-datasets , first you need to collect the RGB image of all classes you need , and use the tool called "labelme" to label the object , just type "labelme" at the terminal and click "Open Dir" to open your image folder which you want to label :





After lableing all image and saving , your image folder will contain json file belonging to each image. Before creating the "coco-datases" you need to create a txt file that list the class :

Then we can create the "coco-datases" by using this command :
"python3 labelme2coco.py --labels "class name txt file dir" "image folder dir" "generating folder dir"

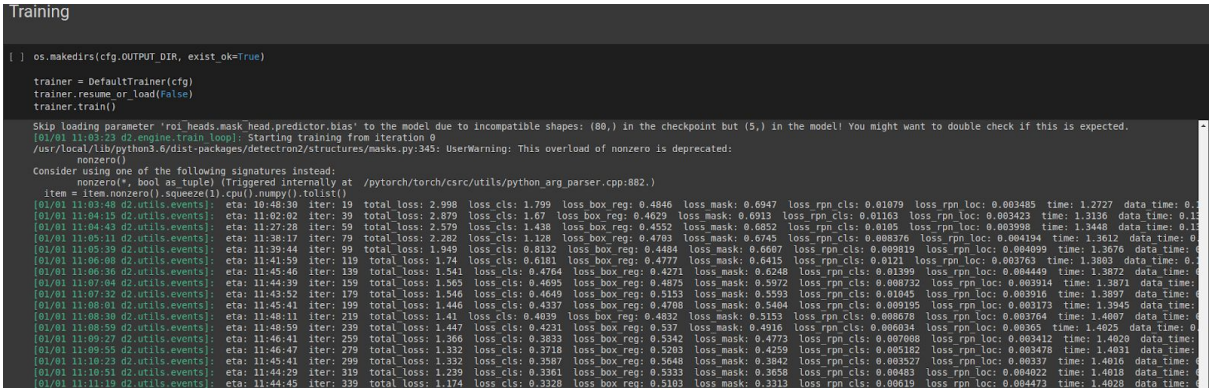
```
austin@austin:~$ python3 labelme2coco.py --labels "class name txt file dir" "image folder dir" "generating folder dir"
```

And then will get a json file like "train.json" or "test.json" and a folder that contain RGB image like "subt_train" or subt_test".
Do above for training data and testing data.

After creating datasets , you can use the our colab notebook to train MaskRCNN.
Change the direction of the datasets to your own :

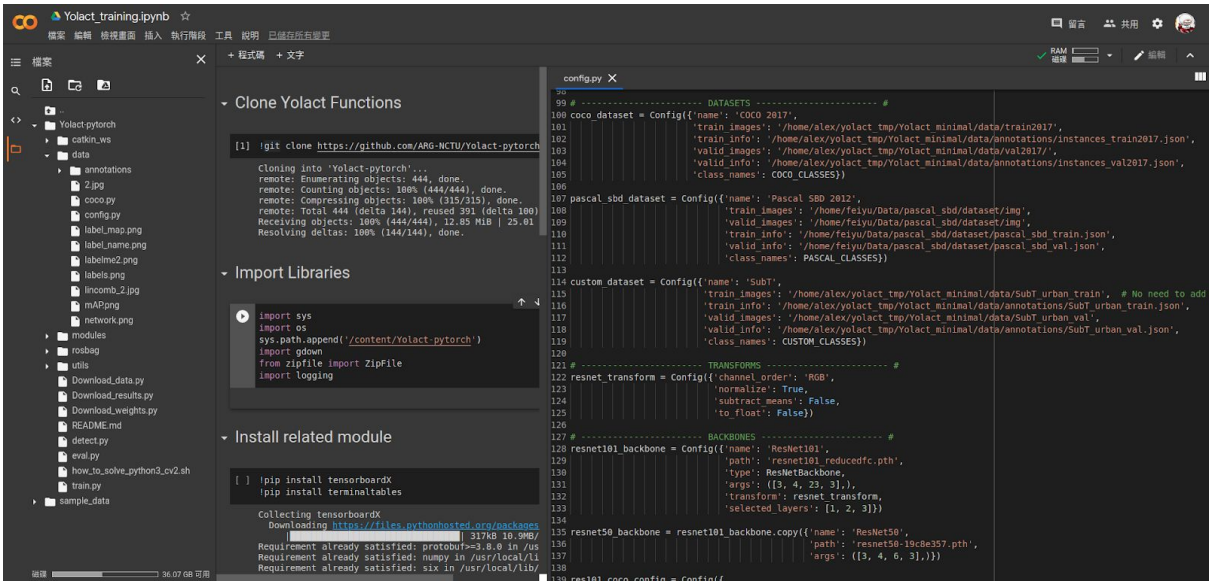
```
[ ]  from detectron2.data.datasets import register_coco_instances

     register_coco_instances('subt_train', {},
                             '/content/subt-urban-coco-dataset/SubT_urban_train.json',
                            '/content/subt-urban-coco-dataset/SubT_urban_train')
     register_coco_instances('subt_val', {},
                             '/content/subt-urban-coco-dataset/SubT_urban_val.json',
                            '/content/subt-urban-coco-dataset/SubT_urban_val')
```

Run the notebook and it will start training :

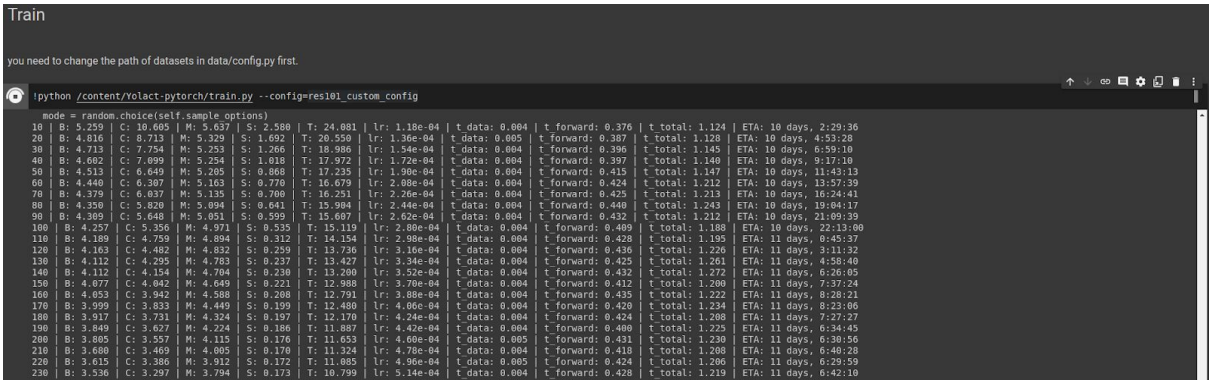


2. Yolact

This model also using the "coco-datasets" , you can use our colab notebook to train Yolact.
After cloning the Yolact from github , click the file icon at the left side and click "Yolact-pytorch" -> "data" -> "config.py" , change the file direction at "custom_dataset = ..." to your own.



Run the notebook and it will start training :



3. FCN

The datasets of the FCN model need two data type "RGB image" and "Mask image" , it's structure is looked like this :

```
austin@austin:~/DataSet/subt_train/FCN_train$ tree -L 2
.
├── test
│   ├── images
│   └── masks
└── train
    ├── images
    └── masks

6 directories, 0 files
```

RGB image (left) and Mask image (right) :

To create mask of image , After using lableme to generate json file for each image , make a folder that only contain these json file , and run the "genejson.sh" (you need to change "dir" and "path" to your own) to create mask , it will bw generate at the json file folder .

You can use our colab notebook to train FCN.
Change the direction of the datasets to your own :

Define path, directory trainning environment

```
[ ]  # get data
     FullPath = os.getcwd()
     data_dir  = os.path.join(FullPath + "/data/FCN_train")
     if not os.path.exists(data_dir):
         print("Data not found!")
```

Run the notebook and it will start training :

```
train()

/usr/local/lib/python3.6/dist-packages/torch/optim/lr_scheduler.py:136: UserWarning: Detected call of `lr_scheduler.step()` before `optimizer.step()`. In PyTorch 1.1.0 and later, you should call them in the opposi
  "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarning)
epoch1, iter0, loss: 0.705998957157135
epoch1, iter1, loss: 0.6945726871490479
epoch1, iter2, loss: 0.6786231398582458
epoch1, iter3, loss: 0.6761897206306458
epoch1, iter4, loss: 0.6707481741905212
epoch1, iter5, loss: 0.6647651195526123
epoch1, iter6, loss: 0.6649534106254578
epoch1, iter7, loss: 0.663619875907898
epoch1, iter8, loss: 0.6579977869987488
epoch1, iter9, loss: 0.6676369309425354
epoch1, iter10, loss: 0.6683955192565918
epoch1, iter11, loss: 0.6660330295562744
epoch1, iter12, loss: 0.6625357866287231
epoch1, iter13, loss: 0.6609653830528259
epoch1, iter14, loss: 0.6623077392578125
epoch1, iter15, loss: 0.6570571064949036
epoch1, iter16, loss: 0.6546377539634705
epoch1, iter17, loss: 0.6635231971740723
```