

[Open in app](#)[Get started](#)

tds

Published in Towards Data Science

You have **2** free member-only stories left this month.

[Sign up for Medium and get an extra one](#)



Maryam Bahrami

[Follow](#)Jul 27, 2021 · 3 min read ★ · [Listen](#)[Save](#)

Train MaskRCNN on custom dataset with Detectron2 in 4 steps

The simplest way to apply object detection on custom dataset



Photo by: Author

Detectron2 is the latest Python library for object detection released by the AI Facebook researchers team. The main advantage of it over Torchvision is that you can



[Open in app](#)[Get started](#)

training MaskRCNN on a custom dataset using Detectron2, so you can see how easy it is in a minute.

First step: Make annotations ready

The annotations must be in the following COCO format, which is a bit different from COCO format introduced [here](#). For each image, include information about the image path, width, height, id and annotations.

Note: If you already have the dataset in the [COCO format](#), you can skip this step and go to the next step. The only difference is, you should register your data with

```
register_coco_instances() instead of the register().
```

Keep in mind that the `filename` should be an image path and `image_id` must be unique among the images of the dataset. `segmentation` is a polygon with `n` points, `(x_i, y_i)`. For bounding boxes, you have multiple choices `[x_0, y_1, x_1, y_1]`, `[x_0, y_0, width, height]` or any other representation presented [here](#). However, the `bbox_mode` should be consistent with `bbox` representation.

For example, I set the `bbox_mode` to `XYWH_ABS` that is `[x_0, y_0, width, height]`.

```
1 dataset = [{'file_name': '../first_image.jpg',
2             'image_id': 125361,
3             'height': 1300,
4             'width': 800,
5             'annotations': [
6                 {'iscrowd': 0,
7                  'segmentation': [[x_0, y_0, x_1, y_1, ..., x_n, y_n]],
8                  'bbox': [x_0, y_0, width, height],
9                  'bbox_mode': BoxMode.XYWH_ABS,
10                 'category_id': 0 },
11                {'iscrowd': 0,
12                 'segmentation': [[x_0, y_0, x_1, y_1, ..., x_n, y_n]],
13                 'bbox': [x_0, y_0, width, height],
14                 'bbox_mode': BoxMode.XYWH_ABS,
15                 'category_id': 1 } ]},
16            {'file_name': '../second_image.jpg',
17             'image_id': 1425361,
```




[Open in app](#)
[Get started](#)

```

23         'bbox': [x_0, y_0, width, height],
24         'bbox_mode': BoxMode.XYWH_ABS,
25         'category_id': 0 },
26     {'iscrowd': 0,
27      'segmentation': [[x_0, y_0, x_1, y_1, ..., x_n, y_n]],
28      'bbox': [x_0, y_0, width, height],
29      'bbox_mode': BoxMode.XYWH_ABS,
30      'category_id': 3 ]}], ...]

```

template of Detectron2 dataset

Second step: Load the data

Assume your dataset is already in the above format and is saved locally as `.json`. To load the data, we should register the dataset in Detectron2 dataset catalog, for this we need a data loader function:

```

1  import json
2  from detectron2.data import MetadataCatalog, DatasetCatalog
3
4
5  def load_data(t="train"):
6      if t == "train":
7          with open("../train.json", 'r') as file:
8              train = json.load(file)
9          return train
10     elif t == "val":
11         with open("../val.json", 'r') as file:
12             val = json.load(file)
13         return val
14
15
16     for d in ["train", "val"]:
17         DatasetCatalog.register(d, lambda d=d: load_data(d))
18         MetadataCatalog.get(d).set(thing_classes=["Dog", "Cat", "Mouse"])
19     metadata = MetadataCatalog.get("train")

```

detectron2 load.py hosted with ❤ by GitHub

[view raw](#)

Load dataset

Third step: Customize configurations




[Open in app](#)
[Get started](#)

We can get configuration files from `detectron2.model_zoo` . In addition, we can use pretrained model by loading the weight from `model_zoo` as well. Besides, we can set the other configurations, as I did in the following with respect to my desire model.

- By default mask is off. To train a MaskRCNN turn it on: `MODEL.MASK_ON = True`
- The backbone network is by default `build_resnet_backbone` , but the pretrained model uses ResnetFPN. I prefer to keep the default and use `resnet34` instead of `resnet101` to reduce the complexity of the model; `MODEL.BACKBONE.NAME = "build_resnet_backbone"` and `cfg.MODEL.RESNETS.DEPTH = 34` .
- I reduce the size of all images equally by setting `cfg.INPUT.MIN_SIZE_TRAIN = (800,)`
- Since I know I have few objects per image, I reduce the pre-process and post-process NMS in the region proposal network.
- You can also set the number of images per batch based on your GPU device `cfg.SOLVER.IMS_PER_BATCH = 4` .
- Set the dataset and output directory locally.

```

1  from detectron2.config import get_cfg
2  from detectron2 import model_zoo
3
4
5  def custom_config(num_classes):
6      cfg = get_cfg()
7
8      # get configuration from model_zoo
9      cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN
10     cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_
11
12     # Model
13     cfg.MODEL.MASK_ON = True
14     cfg.MODEL.ROI_HEADS.NUM_CLASSES = num_classes
15     cfg.MODEL.BACKBONE.NAME = "build_resnet_backbone"
16     cfg.MODEL.RESNETS.DEPTH = 34
17
18

```




[Open in app](#)
[Get started](#)

```

23     cfg.SOLVER.gamma = 0.5
24     cfg.SOLVER.IMS_PER_BATCH = 4
25
26     # Test
27     cfg.TEST.DETECTIONS_PER_IMAGE = 20
28
29     # INPUT
30     cfg.INPUT.MIN_SIZE_TRAIN = (800,)
31
32     # DATASETS
33     cfg.DATASETS.TEST = ('val',)
34     cfg.DATASETS.TRAIN = ('train',)
35
36     # DATASETS
37     cfg.OUTPUT_DIR = "your local path"
38
39     return cfg

```

Last Step: Train

Now, the training is simple. You can do it in a few lines of code. Create the list of labels is `thing_classes`. Here, I have created three labels: person, dog, and cat.

```

1  from detectron2.data import MetadataCatalog, DatasetCatalog
2  from detectron2.engine import DefaultTrainer
3
4
5  if __name__ == '__main__':
6      for d in ["train", "val"]:
7          DatasetCatalog.register(d, lambda d=d: load_data(d))
8          MetadataCatalog.get(d).set(thing_classes=["Person", "Dog", "Cat"])
9      metadata = MetadataCatalog.get("train")
10
11     cfg = custom_config()
12
13     trainer = DefaultTrainer(cfg)
14     trainer.resume_or_load(resume=False)
15     trainer.train()

```

maskrcnn_train.py hosted with ❤ by GitHub

[view raw](#)



[Open in app](#)
[Get started](#)

Visualization is handy for presenting the object detection results, and it also helps in evaluation.

The final model has been saved in the output directory. We can load the weights from the final model, read the images from the test set one by one, run the predictor, and save the output image with the masks and bounding boxes locally.

The test set should contains the `file_name` , which is the image path and the `image_id` . For example,

```
test_data = [{ 'file_name': '.../image_1.jpg',
               'image_id': 10},
             { 'file_name': '.../image_2.jpg',
               'image_id': 20}
            ]
```

`visualization()` takes the custom configuration, metadata, and the test set. It takes the final model and adds the bounding boxes and the masks to the image. Finally, it creates a directory in the output directory and saves the results there.

```
1  from detectron2.engine import DefaultPredictor
2  from detectron2.utils.visualizer import Visualizer, ColorMode
3  import matplotlib.pyplot as plt
4  import cv2
5  import os
6
7
8  def visualization(metadata, cfg, test_set):
9      cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
10     cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.8
11     predictor = DefaultPredictor(cfg)
12     for d in test_set:
13         im = cv2.imread(d["file_name"])
14         outputs = predictor(
15             im)
16         v = Visualizer(im[:, :, ::-1],
17                       metadata=metadata,
```



[Open in app](#)[Get started](#)

visualization step

Conclusion

Detectron2 makes object detection simple and quick. In this tutorial, I explained how you can run MaskRCNN in few steps. Moreover, Detectron2 created a beginners tutorial for object detection [here](#).

References

[1] [Detectron2 Github repository](#)

[2] [Detectron2 Tutorial](#)

Join Medium to read great tutorials and stroies!

I write Machine Learning, Deep Learning and Data Science tutorials. Upgrade to read more.

sciencenotes.medium.com



Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

[Get this newsletter](#)

