Detectron2 Beginner's Tutorial

Detectron2

Welcome to detectron2! This is the official colab tutorial of detectron2. Here, we will go through some basics usage of detectron2, including the following:

- Run inference on images or videos, with an existing detectron2 model
- Train a detectron2 model on a new dataset.

```
!python -m pip install pyyaml
import sys, os, distutils.core
# Note: This is a faster way to install detectron2 in Colab, but it does not include all functionalities (e.g. compiled operators).
# See https://detectron2.readthedocs.io/tutorials/install.html for full installation instructions
!git clone 'https://github.com/facebookresearch/detectron2'
dist = distutils.core.run_setup("./detectron2/setup.py")
!python -m pip install {' '.join([f"'{x}'" for x in dist.install_requires])}
sys.path.insert(0, os.path.abspath('./detectron2'))
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (6.0.1)
fatal: destination path 'detectron2' already exists and is not an empty directory.
Requirement already satisfied: Pillow>=7.1 in /usr/local/lib/python3.10/dist-packages (9.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: pycocotools>=2.0.2 in /usr/local/lib/python3.10/dist-packages (2.0.7)
Requirement already satisfied: termcolor>=1.1 in /usr/local/lib/python3.10/dist-packages (2.3.0)
Requirement already satisfied: yacs>=0.1.8 in /usr/local/lib/python3.10/dist-packages (0.1.8)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (0.9.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (2.2.1)
Requirement already satisfied: tqdm>4.29.0 in /usr/local/lib/python3.10/dist-packages (4.66.1)
Requirement already satisfied: tensorboard in /usr/local/lib/python3.10/dist-packages (2.14.1)
Requirement already satisfied: fvcore<0.1.6,>=0.1.5 in /usr/local/lib/python3.10/dist-packages (0.1.5.post20221221)
Requirement already satisfied: iopath<0.1.10,>=0.1.7 in /usr/local/lib/python3.10/dist-packages (0.1.9)
Requirement already satisfied: omegaconf<2.4,>=2.1 in /usr/local/lib/python3.10/dist-packages (2.3.0)
Requirement already satisfied: hydra-core>=1.1 in /usr/local/lib/python3.10/dist-packages (1.3.2)
Requirement already satisfied: black in /usr/local/lib/python3.10/dist-packages (23.11.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (23.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.44.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.23.5)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from yacs>=0.1.8) (6.0.1)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (1.4.0)
Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (1.59.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (2.17.3)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (3.5.1)
Requirement already satisfied: protobuf>=3.19.6 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (2.31.0)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (67.7.2)
Requirement already satisfied: six>1.9 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (1.16.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (0.7.
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard) (3.0.1)
Requirement already satisfied: portalocker in /usr/local/lib/python3.10/dist-packages (from iopath<0.1.10,>=0.1.7) (2.8.2)
Requirement already satisfied: antlr4-python3-runtime==4.9.* in /usr/local/lib/python3.10/dist-packages (from omegaconf<2.4,>=2.1) (4.9.
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from black) (8.1.7)
Requirement already satisfied: mypy-extensions>=0.4.3 in /usr/local/lib/python3.10/dist-packages (from black) (1.0.0)
Requirement already satisfied: pathspec>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from black) (0.11.2)
Requirement already satisfied: platformdirs>=2 in /usr/local/lib/python3.10/dist-packages (from black) (3.11.0)
Requirement already satisfied: tomli>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from black) (2.0.1)
Requirement already satisfied: typing-extensions>=4.0.1 in /usr/local/lib/python3.10/dist-packages (from black) (4.5.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboar
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0.5-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboar
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard) (2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard) (20
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oa
```

```
import torch, detectron2
!nvcc --version
TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
CUDA_VERSION = torch.__version__.split("+")[-1]
print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
print("detectron2:", detectron2.__version__)
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
torch:  2.1 ; cuda:  cu118
detectron2: 0.6
```

```
# Some basic setup:
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import os, json, cv2, random
from google.colab.patches import cv2_imshow

# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog
```

Run a pretrained Detectron2 model

We first download some image from the given URLs:

```
!wget http://images.cocodataset.org/val2017/000000007574.jpg -q -O input.jpg
im_input = cv2.imread("./input.jpg")
cv2_imshow(im_input)
```

```
!wget http://images.cocodataset.org/val2017/000000013923.jpg -q -O test1.jpg
im_test1 = cv2.imread("./test1.jpg")
cv2_imshow(im_test1)
```

```
!wget http://images.cocodataset.org/val2017/000000018380.jpg -q -O test2.jpg
im_test2 = cv2.imread("./test2.jpg")
cv2_imshow(im_test2)
```

We can see there are multiple objects in these images: bottles, tables, chairs, people, etc. Let us see if we can detect them all by using a pre-trained model given by Detectron2.

Let's take a look at the model output.

In inference mode, the builtin model outputs a `list[dict]`, one dict for each image. For the object detection task, the dict contain the following fields:

- "instances": Instances object with the following fields:
    - "pred_boxes": Storing N boxes, one for each detected instance.
    - "scores": a vector of N scores.
    - "pred_classes": a vector of N labels in range [0, num_categories].

For more details, please see https://detectron2.readthedocs.io/tutorials/models.html#model-output-format for specification

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST= 0.5  # set threshold for this model
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml")
predictor = DefaultPredictor(cfg)
outputs = predictor(im_input)
```

```
[11/11 05:49:59 d2.checkpoint.detection_checkpoint]: [DetectionCheckpointer] Loading from https://dl.fbaipublicfiles.com/detectron2/COCO
/usr/local/lib/python3.10/dist-packages/torch/functional.py:504: UserWarning: torch.meshgrid: in an upcoming release, it will be require
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
```

```
1 print(outputs)
```

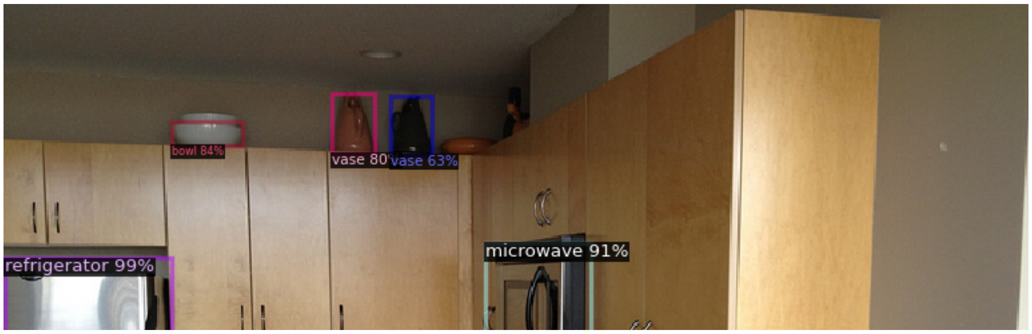
```
{'instances': Instances(num_instances=17, image_height=480, image_width=640, fields=[pred_boxes: Boxes(tensor([[  0.6764, 158.7422, 105.
        [480.6438, 298.5876, 509.7858, 378.3471],
        [293.3246, 287.8804, 344.7386, 388.2297],
        [501.2999, 306.3395, 533.4362, 387.4037],
        [300.8816, 148.8771, 366.4546, 227.1448],
        [104.8806,  73.4484, 148.7521,  88.7021],
        [205.1707,  55.5022, 231.9424,  93.1075],
        [357.5101, 394.1846, 393.9808, 425.1449],
        [ 38.0390, 355.4640, 315.9145, 442.3582],
        [357.1301, 286.4151, 387.7237, 365.4046],
        [197.1212, 267.4070, 208.2965, 292.6432],
        [241.4115,  57.8996, 267.1059,  93.9473],
        [454.1900, 310.0651, 484.8028, 360.4231],
        [243.9083, 382.6956, 275.9801, 407.7662],
        [185.0177, 355.2176, 304.0785, 409.8493],
        [441.3940, 307.2240, 470.4242, 347.5514],
        [330.0779, 267.5388, 355.4027, 303.1052]], device='cuda:0')), scores: tensor([0.9938, 0.9790, 0.9769, 0.9570, 0.9114, 0.8425, 0.
        0.6727, 0.6304, 0.6266, 0.6026, 0.6025, 0.5396, 0.5248, 0.5182],
       device='cuda:0'), pred_classes: tensor([72, 39, 39, 39, 68, 45, 75, 55, 71, 41, 39, 75, 41, 45, 71, 41, 39],
       device='cuda:0')])}
```

```
1 print(outputs["instances"].pred_classes)
2 print(outputs["instances"].pred_boxes)
```

```
tensor([72, 39, 39, 39, 68, 45, 75, 55, 71, 41, 39, 75, 41, 45, 71, 41, 39],
       device='cuda:0')
Boxes(tensor([[  0.6764, 158.7422, 105.0817, 397.2280],
        [480.6438, 298.5876, 509.7858, 378.3471],
        [293.3246, 287.8804, 344.7386, 388.2297],
        [501.2999, 306.3395, 533.4362, 387.4037],
        [300.8816, 148.8771, 366.4546, 227.1448],
        [104.8806,  73.4484, 148.7521,  88.7021],
        [205.1707,  55.5022, 231.9424,  93.1075],
        [357.5101, 394.1846, 393.9808, 425.1449],
        [ 38.0390, 355.4640, 315.9145, 442.3582],
        [357.1301, 286.4151, 387.7237, 365.4046],
        [197.1212, 267.4070, 208.2965, 292.6432],
        [241.4115,  57.8996, 267.1059,  93.9473],
        [454.1900, 310.0651, 484.8028, 360.4231],
        [243.9083, 382.6956, 275.9801, 407.7662],
        [185.0177, 355.2176, 304.0785, 409.8493],
        [441.3940, 307.2240, 470.4242, 347.5514],
        [330.0779, 267.5388, 355.4027, 303.1052]], device='cuda:0'))
```

```
1 outputs_q1q2 = {'q1': [], 'q2': []}
2 outputs_q1q2['q1'].append(outputs["instances"])
```

```
1 # We can use "Visualizer" to draw the predictions on the image
2 v = Visualizer(im_input[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
3 out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
4 cv2_imshow(out.get_image()[:, :, ::-1])
```



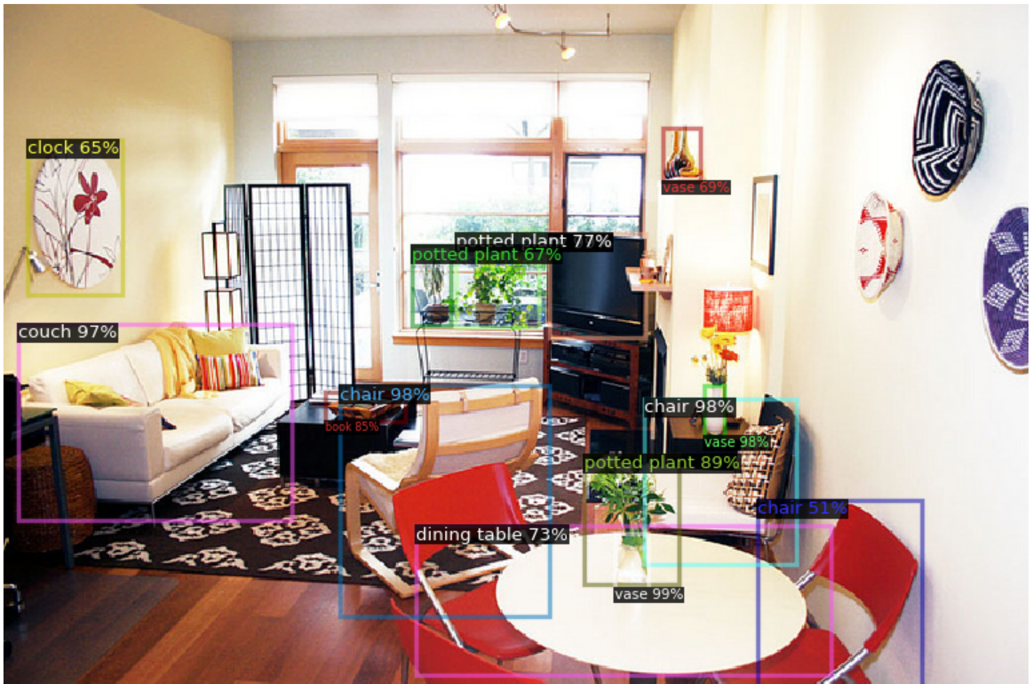
AWESOME!!! Great progress so far! We are able to detect sink, microwave, bottle and even refrigerator! At this point, we have used the pre-trained model to do the inference on the given image. There are in total 17 objects are being detected. The image is adopted from the [MS-COCO](#) dataset and there are 81 classes including person, bicycle, car, etc. You may find the id-category mapping [here](#).

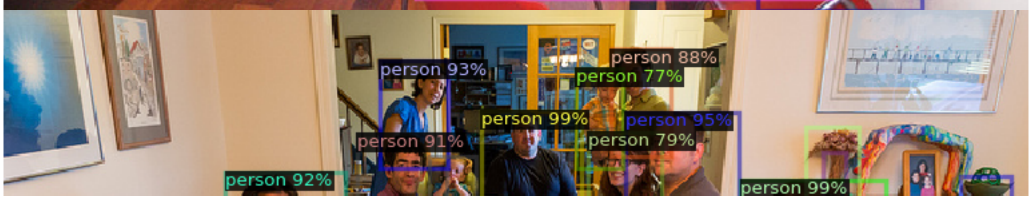
The model we just used is `COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml`. Actually, the Detectron2 provides us more than that, you may find great amouts of models for different tasks in the given [MODEL_ZOO](#). What about we try a different model to see what its output will look like?

- Q1 (5%): Object Detection. Use the same configuration `COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml`, with IoU threshold of 0.5 (`SCORE_THRESH_TEST=0.5`), to also run inference on the rest two images (test1.jpg & test2.jpg) and view the outputs with bounding boxes.
- Q2: Object Detection. Use the `COCO-Detection/faster_rcnn_R_101_FPN_3X.yaml`, which has a ResNet-101 as the backbone, with IoU threshold of 0.5 and view the outputs of all three images with bounding boxes. By looking at the outputs, can you find the difference with the one `COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml` we used in Q1? (e.g., numbers of objects, confidence scores, ...)
- Q3: Object Detection. Use the `COCO-Detection/faster_rcnn_R_101_FPN_3X.yaml` with an IoU threshold of 0.9 and view the outputs of all three images with bounding boxes.
- Q4 (5%): Instance Segmentation. The models we have tried in Q1-Q3 are the Faster R-CNN models for object detection. Here, let's try a Mask R-CNN model `COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml`, with IoU threshold of 0.5, to perform the instance segmentation and view the outputs of all three images with segmentation masks. Compare the difference of outputs between an object detection model with an instance segmentation model.

```
## Q1

outputs = predictor(im_test1)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_test1[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
outputs = predictor(im_test2)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_test2[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
```






```
# todo: Q2
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST= 0.5  # set threshold for this model
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")
predictor = DefaultPredictor(cfg)
outputs = predictor(im_test2)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_test2[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
outputs = predictor(im_test1)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_test1[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
outputs = predictor(im_input)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_input[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
```

```
[11/11 05:50:08 d2.checkpoint.detection_checkpoint]: [DetectionCheckpointer] Loading from https://dl.fbaipublicfiles.com/detectron2/COCO
```

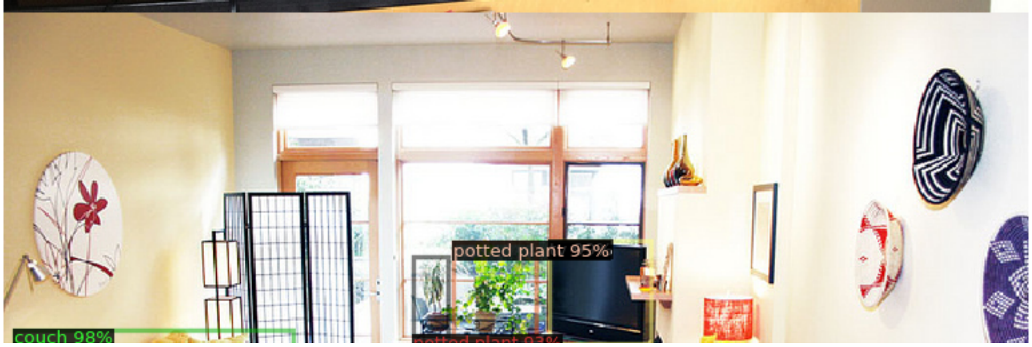
In this new 101 iteration of CNN, the bounding boxes are more "tightly packed" In addition, some of the mistakes previously made arent. For example, if you look at the im_input image, the ribbon was marked cake, but is not anymore. In the test 2 image, a fork is wrongly detected as a pizza, and is correctly determined as a fork. Such inaccuracies are reduced in this 101 iteration ofer the 50 iteration.

The confidance scores are all generally higher as well, and the number of boxes, though not by much, have reduced. For example, in the Test 1 image, the value redudced from 13 to 12.

```
# todo: Q3
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST= 0.9  # set threshold for this model
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")
predictor = DefaultPredictor(cfg)
outputs = predictor(im_input)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_input[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])

outputs = predictor(im_test1)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_test1[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
outputs = predictor(im_test2)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_test2[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
```

[11/11 05:50:14 d2.checkpoint.detection_checkpoint]: [DetectionCheckpointer] Loading from https://dl.fbaipublicfiles.com/detectron2/COCO




```
# todo: Q4
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST= 0.9  # set threshold for this model
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml")
predictor = DefaultPredictor(cfg)
outputs = predictor(im_input)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_input[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])

outputs = predictor(im_test1)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_test1[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
outputs = predictor(im_test2)
outputs_q1q2 = {'q1': [], 'q2': []}
outputs_q1q2['q1'].append(outputs["instances"])
# We can use "Visualizer" to draw the predictions on the image
v = Visualizer(im_test2[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
```


clock 94%
potted plant 93%
potted plant 94%
couch 95%
chair 95%
vase 100%
chair 96%
vase 99%


person 97%
person 99%
person 96%
person 91%
person 99%
person 100%
person 91%
person 98%
person 100%
person 100%
person 100%
person 99%
cup 93%
wine glass 91%
wine glass 98%
cup 99%
cup 98%
bowl 97%
cup 99%

This is different in that there is quite literally a mask over whatever is being detected. In addition, the accuracies are quite high as well, all usually above the 90% mark. All in all, it seems to be a better alternative.

Now let's train on the provided sportsmot dataset

please upload the provided sportsmot dataset first

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
!unzip "/content/drive/MyDrive/sportsmot.zip"
```

```
  inflating: sportsmot/JPEGImages/train_000055.jpg
  inflating: sportsmot/JPEGImages/train_000057.jpg
  inflating: sportsmot/JPEGImages/train_000058.jpg
  inflating: sportsmot/JPEGImages/train_000059.jpg
  inflating: sportsmot/JPEGImages/train_000060.jpg
  inflating: sportsmot/JPEGImages/train_000062.jpg
```

```
    inflating: sportsmot/JPEGImages/train_000126.jpg
```

```
from detectron2.structures import BoxMode

def get_dataset_dicts(data_root, txt_file):
    dataset_dicts = []
    filenames = []
    csv_path = os.path.join(data_root, txt_file)
    with open(csv_path, "r") as f:
        for line in f:
            filenames.append(line.rstrip())

    for idx, filename in enumerate(filenames):
        record = {}

        image_path = os.path.join(data_root, filename)

        im = cv2.imread(image_path)
        if im is None:
            continue
        height, width = im.shape[:2]

        record['file_name'] = image_path
        record['image_id'] = idx
        record['height'] = height
        record['width'] = width

        image_filename = os.path.basename(filename)
        image_name = os.path.splitext(image_filename)[0]
        annotation_path = os.path.join(data_root, 'labels', '{}.txt'.format(image_name))
        annotation_rows = []

        with open(annotation_path, "r") as f:
            for line in f:
                temp = line.rstrip().split(" ")
                annotation_rows.append(temp)

        objs = []
        for row in annotation_rows:
            xcentre = int(float(row[1])*width)
            ycentre = int(float(row[2])*height)
            bwidth = int(float(row[3])*width)
            bheight = int(float(row[4])*height)

            xmin = int(xcentre - bwidth/2)
            ymin = int(ycentre - bheight/2)
            xmax = xmin  + bwidth
            ymax = ymin + bheight

            obj= {
                'bbox': [xmin, ymin, xmax, ymax],
                'bbox_mode': BoxMode.XYXY_ABS,
                # alternatively, we can use bbox_mode = BoxMode.XYWH_ABS
                # 'bbox': [xmin, ymin, bwidth, bheight],
                # 'bbox_mode': BoxMode.XYWH_ABS,
                'category_id': int(row[0]),
                'iscrowd': 0
            }

            objs.append(obj)
        record['annotations'] = objs
        dataset_dicts.append(record)
    return dataset_dicts
```

```
import os
import os.path as osp
# Metadata configurations
data_root = "sportsmot"
train_txt = "sportsmot_train.txt"
test_txt = "sportsmot_test.txt"

train_data_name = "train"
test_data_name = "test"

thing_classes = ["person"]

```

```
output_dir = "./outputs"

def count_lines(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return i + 1

train_img_count = count_lines(os.path.join(data_root, train_txt))
print("There are {} samples in training data".format(train_img_count))
```

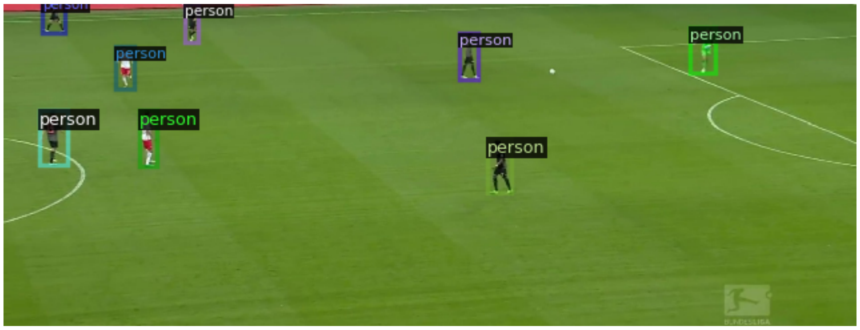
```
There are 725 samples in training data
```

```
# Register the traffic_sign_train datasets
DatasetCatalog.register(name=train_data_name,
                        func=lambda: get_dataset_dicts(data_root, train_txt))
train_metadata = MetadataCatalog.get(train_data_name).set(thing_classes=thing_classes)

# Register the traffic_sign_test datasets
DatasetCatalog.register(name=test_data_name,
                        func=lambda: get_dataset_dicts(data_root, test_txt))
test_metadata = MetadataCatalog.get(test_data_name).set(thing_classes=thing_classes)
```

```
train_data_dict = get_dataset_dicts(data_root, train_txt)

for d in random.sample(train_data_dict, 3):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=train_metadata, scale=0.5)
    out = visualizer.draw_dataset_dict(d)
    cv2_imshow(out.get_image()[:, :, ::-1])
```




```
from detectron2.engine import DefaultTrainer

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml"))
cfg.DATASETS.TRAIN = (train_data_name,)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml") # let's trainining initialize from model zo
cfg.SOLVER.IMS_PER_BATCH = 3
cfg.SOLVER.BASE_LR = 0.001  # pick a good LR
cfg.SOLVER.MAX_ITER = 300    # 300 iterations seems good enough for this toy dataset; you will need to train longer for a practical datase
cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(thing_classes)  # only has one class (traffic-sign)
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 256   # faster, and good enough for this toy dataset (default: 512)
cfg.OUTPUT_DIR = output_dir
```

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

```
[11/11 06:20:50 d2.engine.defaults]: Model:
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (top_block): LastLevelMaxPool()
    (bottom_up): ResNet(
      (stem): BasicStem(
        (conv1): Conv2d(
          3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
      )
      (res2): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv1): Conv2d(
            64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
```

```
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
```

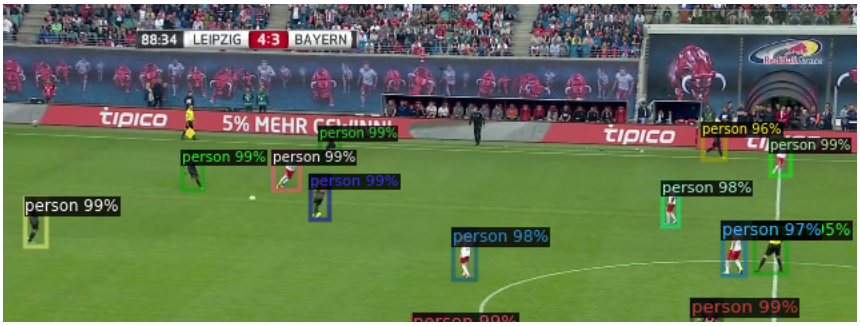
```python
# cfg alrady contains everything we've set previously. Now we changed it a little bit for inference:
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")  # path to the model we just trained
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.65
predictor = DefaultPredictor(cfg)
```

```
[11/11 06:25:05 d2.checkpoint.detection_checkpoint]: [DetectionCheckpointer] Loading from ./outputs/model_final.pth ...
```

```python
from detectron2.utils.visualizer import ColorMode

test_data_dict = get_dataset_dicts(data_root, test_txt)

for d in random.sample(test_data_dict, 3):
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)
    v = Visualizer(im[:, :, ::-1],
                   metadata=test_metadata,
                   scale=0.5,
                   )
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2_imshow(out.get_image()[:, :, ::-1])
```



Problem 2

Multi-Object Tracking

After training the detector, now we want to implement tracking on the testing video.

```
# Let's start with a detector class
class detector:
    def __init__(self,predictor):
        self.model = predictor

    def predict(self,img):
        pred = self.model(img)
        pred = [pred['instances'][i].pred_boxes.tensor.tolist()[0] for i in range(len(pred['instances']))]
        return pred

# TODO
# Initiate a detector and inference on the first test image('sportsmot/JPEGImages/test_000001.jpg') and print the bounding box prediction.
# The output format should be x1,y1,x2,y2
```

```
test_data_dict = get_dataset_dicts(data_root, test_txt)
```

```
im = cv2.imread(test_data_dict[0]['file_name'])

outputs = predictor(im)
```

```
thisDetector = detector(predictor)
```

```
thisDetector.predict(im)
```

```
[[695.1017456054688, 329.04803466796875, 727.0886840820312, 395.6009216308594],
 [198.265625, 240.1858673095703, 231.60711669921875, 291.2448425292969],
 [608.6708984375, 380.8650817871094, 638.8543090820312, 440.002685546875],
 [767.8047485351562, 455.3278503417969, 804.1311645507812, 538.384033203125],
 [1243.25048828125, 387.00177001953125, 1274.087158203125, 466.32476806640625],
 [868.5335083007812,
  257.01336669921875,
  898.5045776367188,
  311.70037841796875],
 [545.5725708007812, 209.90061950683594, 572.3731079101562, 260.3439636230469],
 [1062.9161376953125,
  292.83416748046875,
  1092.3922119140625,
  357.8119201660156],
 [348.142822265625, 453.1558532714844, 381.67755126953125, 525.9806518554688],
 [1110.0975341796875,
  293.57086181640625,
  1133.2042236328125,
  359.6961975097656],
 [1171.104736328125, 245.98287963867188, 1189.6796875, 302.8831481933594],
 [487.81304931640625,
  327.46539306640625,
  510.3650207519531,
  391.7694091796875],
 [746.527587890625, 254.35745239257812, 772.0448608398438, 310.05157470703125],
 [65.74864959716797, 274.23773193359375, 93.96112060546875, 335.4070129394531],
 [33.90310287475586, 308.1076965332031, 59.97958755493164, 378.60284423828125],
 [89.94819641113281,
  281.0174560546875,
  115.04015350341797,
  336.06024169921875],
 [754.9312133789062, 326.55084228515625, 775.4976196289062, 397.2494812011719],
```

```
 [885.1402587890625, 402.6461181640625, 910.5313110351562, 479.73583984375],
 [210.8836212158203, 286.97467041015625, 233.4933624267578, 349.0710754394531],
 [493.33306884765625,
  208.96238708496094,
  510.21954345703125,
  260.8963317871094],
 [662.82080078125, 306.70709228515625, 686.7090454101562, 372.1379089355469]]
```

Now you will implement your own tracker!

Let's start with the IoU function and tracklet class.

```python
# calculate the overlap ratio of two bounding boxes
def calculate_iou(bbox1, bbox2):

    x1_1, y1_1, x2_1, y2_1 = bbox1
    x1_2, y1_2, x2_2, y2_2 = bbox2
    x_left = max(x1_1, x1_2)
    y_top = max(y1_1, y1_2)
    x_right = min(x2_1, x2_2)
    y_bottom = min(y2_1, y2_2)

    if x_right < x_left or y_bottom < y_top:
        return 0.0

    area_bbox1 = (x2_1 - x1_1 + 1) * (y2_1 - y1_1 + 1)
    area_bbox2 = (x2_2 - x1_2 + 1) * (y2_2 - y1_2 + 1)
    intersection_area = (x_right - x_left + 1) * (y_bottom - y_top + 1)

    iou = intersection_area / float(area_bbox1 + area_bbox2 - intersection_area)

    return iou
```

```python
# base class for tracklet, you can of course add more features and try to improve the performance!
class tracklet:
    def __init__(self,tracking_ID,box):
        self.ID = tracking_ID
        self.cur_box = box
        self.alive = True

    def update(self,box):
        self.cur_box = box

    def close(self):
        self.alive = False
```

```python
from scipy.optimize import linear_sum_assignment

class IoU_Tracker:
    def __init__(self):
        self.all_tracklets = [] # this saves all the tracklets so that we can know how many tracklets we have
        self.cur_tracklets = [] # this saves tracklets from the last frame for current frame's association
        self.online_tracklets = [] # this saves the tracklets after association, so we can pass the tracking result to output

    def update(self,frame_id,detection):

        if frame_id%100 == 0:
            print(f'Running tracking || current frame {frame_id}')

        if len(self.cur_tracklets) == 0:
            for det in detection:
                new_tracklet = tracklet(len(self.all_tracklets)+1,det)
                self.cur_tracklets.append(new_tracklet)
                self.all_tracklets.append(new_tracklet)
        else:
            cost_matrix = np.zeros((len(self.cur_tracklets),len(detection)))

            # build up cost matrix, each element in cost matrix should be 1-IoU between tracklet and detection
            for row in range(len(self.cur_tracklets)):
                for col in range(len(detection)):
                    cost_matrix[row][col] = 1 - calculate_iou(self.cur_tracklets[row].cur_box,detection[col])

            row_inds,col_inds = linear_sum_assignment(cost_matrix)

            matches = min(len(row_inds),len(col_inds))
```

```

            for idx,trk in enumerate(self.cur_tracklets):
                if idx not in row_inds: # if it is not matched in the above Hungarian algorithm stage
                # TODO
                # use tracklet's close function to kill those unmatched tracklets
                  self.cur_tracklets[idx].close()
                  self.all_tracklets[idx].close()

            for idx,det in enumerate(detection):
                if idx not in col_inds: # if it is not matched in the above Hungarian algorithm stage
                # TODO
                # initiate unmatched detections as new tracklets
                  new_tracklet = tracklet(idx,det)
                  self.all_tracklets.append(new_tracklet)
                  self.cur_tracklets.append(new_tracklet)

            for idx in range(matches):
                row,col = row_inds[idx],col_inds[idx]
                if cost_matrix[row][col] == 1:
                    # TODO 1. Kill the tracklet using tracklet's close function
                    self.cur_tracklets[row].close()
                    self.all_tracklets[row].close()
                    # TODO 2. Initiate a new tracklet for the new detection
                    new_tracklet = tracklet(len(self.all_tracklets),det)
                    # TODO 3. Append new tracklet to the current tracklets and all tracklets
                    self.all_tracklets.append(new_tracklet)
                    self.cur_tracklets.append(new_tracklet)
                else:
                    self.cur_tracklets[row].update(detection[col])

        self.cur_tracklets = [trk for trk in self.cur_tracklets if trk.alive]

        return self.cur_tracklets
```

Now it's time to run tracking!

```
import glob

# TODO
# initialize your detector (the one you just trained)
thisDetector = detector(predictor)

# TODO
# initialize tracker
thisIOUTracker = IoU_Tracker()

# TODO run tracking
images = np.loadtxt('sportsmot/sportsmot_test.txt',dtype=str)
results = []

print(f'length of sequence is {len(images)}')

for frame_id, img_path in enumerate(images,1):
    img = cv2.imread('sportsmot/'+img_path)
    # TODO get detection with your model
    detection = thisDetector.predict(img)

    # TODO update tracker with detection
    result = thisIOUTracker.update(frame_id,detection)

    for track in result:
        x1,y1,x2,y2 = track.cur_box
        track_id = track.ID
        results.append(f'{int(frame_id)},{int(track_id)},{int(x1)},{int(y1)},{int(x2-x1)},{int(y2-y1)},{1},{1},{1}') # format: frame_id, t

with open('results.txt','w') as f:
    for line in results:
        f.writelines(line)
        f.writelines('\n')
```

```
length of sequence is 500
Running tracking || current frame 100
Running tracking || current frame 200
Running tracking || current frame 300
Running tracking || current frame 400
Running tracking || current frame 500
```