Detectron2 Beginner's Tutorial

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

You can make a copy of this tutorial by "File -> Open in playground mode" and make changes there. **DO NOT** request access to this tutorial.

Install detectron2

```
!python -m pip install pyyaml==5.1
import sys, os, distutils.core
# Note: This is a faster way to install detectron2 in Colab, but it does not include all f
# See https://detectron2.readthedocs.io/tutorials/install.html for full installation instr
!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'))

# Properly install detectron2. (Please do not install twice in both ways)
# !python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'
```

```
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) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
torch:  1.11 ; cuda:  cu113
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 pre-trained detectron2 model

We first download an image from the COCO dataset:

```
!wget http://images.cocodataset.org/val2017/000000439715.jpg -q -O input.jpg
im = cv2.imread("./input.jpg")
cv2_imshow(im)
```

Then, we create a detectron2 config and a detectron2 `DefaultPredictor` to run inference on this image.

```
cfg = get_cfg()
# add project-specific config (e.g., TensorMask) here if you're not running a model in det
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FP
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # set threshold for this model
# Find a model from detectron2's model zoo. You can use the https://dl.fbaipublicfiles...
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50
predictor = DefaultPredictor(cfg)
outputs = predictor(im)
```

```
model_final_f10217.pkl: 178MB [00:03, 48.0MB/s]
/usr/local/lib/python3.7/dist-packages/torch/functional.py:568: UserWarning: torch.m
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
```

```
# look at the outputs. See https://detectron2.readthedocs.io/tutorials/models.html#model-o
print(outputs["instances"].pred_classes)
print(outputs["instances"].pred_boxes)
```

```
tensor([17,  0,  0,  0,  0,  0,  0,  0, 25,  0, 25, 25,  0,  0, 24],
       device='cuda:0')
Boxes(tensor([[126.6035, 244.8977, 459.8292, 480.0000],
        [251.1083, 157.8127, 338.9731, 413.6379],
        [114.8496, 268.6864, 148.2352, 398.8111],
        [  0.8217, 281.0327,  78.6072, 478.4210],
        [ 49.3954, 274.1229,  80.1545, 342.9808],
        [561.2248, 271.5816, 596.2755, 385.2552],
        [385.9072, 270.3125, 413.7130, 304.0397],
        [515.9295, 278.3744, 562.2792, 389.3803],
        [335.2409, 251.9167, 414.7491, 275.9375],
        [350.9300, 269.2060, 386.0984, 297.9081],
        [331.6292, 230.9996, 393.2759, 257.2009],
        [510.7349, 263.2656, 570.9865, 295.9194],
        [409.0841, 271.8646, 460.5582, 356.8722],
        [506.8766, 283.3257, 529.9403, 324.0392],
        [594.5663, 283.4820, 609.0577, 311.4124]], device='cuda:0'))
```

```
# We can use `Visualizer` to draw the predictions on the image.
v = Visualizer(im[:, :, ::-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])
```

Train on a custom dataset

In this section, we show how to train an existing detectron2 model on a custom dataset in a new format.

We use [the balloon segmentation dataset](#) which only has one class: balloon. We'll train a balloon segmentation model from an existing model pre-trained on COCO dataset, available in detectron2's model zoo.

Note that COCO dataset does not have the "balloon" category. We'll be able to recognize this new class in a few minutes.

Prepare the dataset

```
# download, decompress the data
!wget https://github.com/matterport/Mask_RCNN/releases/download/v2.1/balloon_dataset.zip
!unzip balloon_dataset.zip > /dev/null
```

```
--2021-09-02 02:36:36--  https://github.com/matterport/Mask_RCNN/releases/download/v
Resolving github.com (github.com)... 140.82.121.3
Connecting to github.com (github.com)|140.82.121.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-releases.githubusercontent.com/107595270/737339e2-2b83-11e8
--2021-09-02 02:36:36--  https://github-releases.githubusercontent.com/107595270/737
Resolving github-releases.githubusercontent.com (github-releases.githubusercontent.c
Connecting to github-releases.githubusercontent.com (github-releases.githubuserconte
HTTP request sent, awaiting response... 200 OK
Length: 38741381 (37M) [application/octet-stream]
```

```
Saving to: ‘balloon_dataset.zip’

balloon_dataset.zip 100%[===================>]  36.95M  52.6MB/s    in 0.7s

2021-09-02 02:36:37 (52.6 MB/s) - ‘balloon_dataset.zip’ saved [38741381/38741381]
```

Register the balloon dataset to detectron2, following the [detectron2 custom dataset tutorial](). Here, the dataset is in its custom format, therefore we write a function to parse it and prepare it into detectron2's standard format. User should write such a function when using a dataset in custom format. See the tutorial for more details.

```
# if your dataset is in COCO format, this cell can be replaced by the following three line
# from detectron2.data.datasets import register_coco_instances
# register_coco_instances("my_dataset_train", {}, "json_annotation_train.json", "path/to/i
# register_coco_instances("my_dataset_val", {}, "json_annotation_val.json", "path/to/image

from detectron2.structures import BoxMode

def get_balloon_dicts(img_dir):
    json_file = os.path.join(img_dir, "via_region_data.json")
    with open(json_file) as f:
        imgs_anns = json.load(f)

    dataset_dicts = []
    for idx, v in enumerate(imgs_anns.values()):
        record = {}

        filename = os.path.join(img_dir, v["filename"])
        height, width = cv2.imread(filename).shape[:2]

        record["file_name"] = filename
        record["image_id"] = idx
        record["height"] = height
        record["width"] = width

        annos = v["regions"]
        objs = []
        for _, anno in annos.items():
            assert not anno["region_attributes"]
            anno = anno["shape_attributes"]
            px = anno["all_points_x"]
            py = anno["all_points_y"]
            poly = [(x + 0.5, y + 0.5) for x, y in zip(px, py)]
            poly = [p for x in poly for p in x]

            obj = {
                "bbox": [np.min(px), np.min(py), np.max(px), np.max(py)],
                "bbox_mode": BoxMode.XYXY_ABS,
                "segmentation": [poly],
                "category_id": 0,
```

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

for d in ["train", "val"]:
    DatasetCatalog.register("balloon_" + d, lambda d=d: get_balloon_dicts("balloon/" + d))
    MetadataCatalog.get("balloon_" + d).set(thing_classes=["balloon"])
balloon_metadata = MetadataCatalog.get("balloon_train")
```

To verify the dataset is in correct format, let's visualize the annotations of randomly selected samples in the training set:

```
dataset_dicts = get_balloon_dicts("balloon/train")
for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=balloon_metadata, scale=0.5)
    out = visualizer.draw_dataset_dict(d)
    cv2_imshow(out.get_image()[:, :, ::-1])
```


balloon
balloon
balloon
balloon
balloon
balloon
balloon
balloon
balloon
balloon

Train!

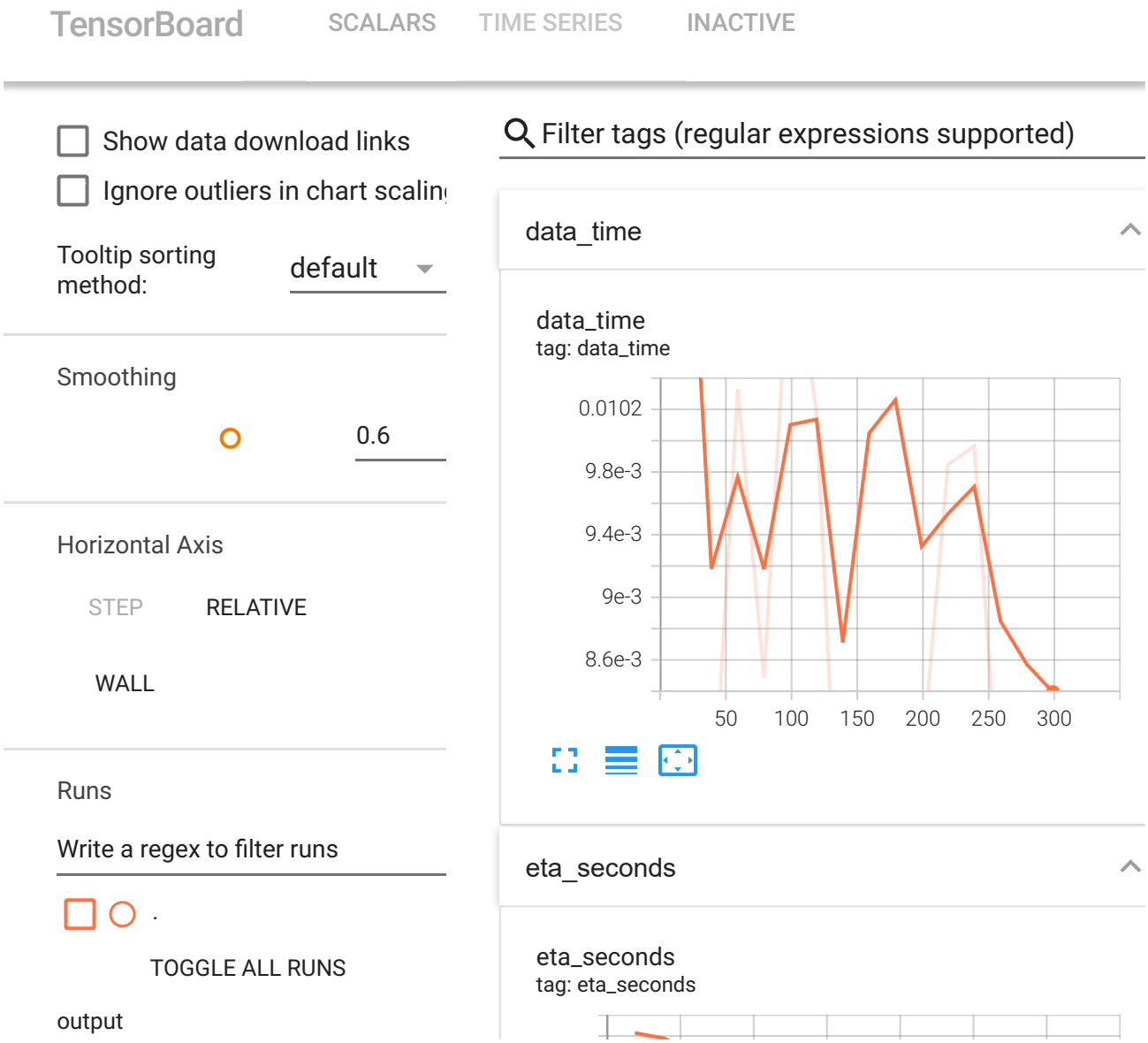
Now, let's fine-tune a COCO-pretrained R50-FPN Mask R-CNN model on the balloon dataset. It takes ~2 minutes to train 300 iterations on a P100 GPU.

```
from detectron2.engine import DefaultTrainer

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FP
cfg.DATASETS.TRAIN = ("balloon_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50
cfg.SOLVER.IMS_PER_BATCH = 2  # This is the real "batch size" commonly known to deep learn
cfg.SOLVER.BASE_LR = 0.00025  # pick a good LR
cfg.SOLVER.MAX_ITER = 300    # 300 iterations seems good enough for this toy dataset; you
cfg.SOLVER.STEPS = []        # do not decay learning rate
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128   # The "RoIHead batch size". 128 is faster
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1  # only has one class (ballon). (see https://detectron
# NOTE: this config means the number of classes, but a few popular unofficial tutorials in

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

```
              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)
    )
    (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)
    )
  )
)
(res3): Sequential(
  (0): BottleneckBlock(
    (shortcut): Conv2d(
      256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
```

```
# Look at training curves in tensorboard:
%load_ext tensorboard
%tensorboard --logdir output
```



Inference & evaluation using the trained model

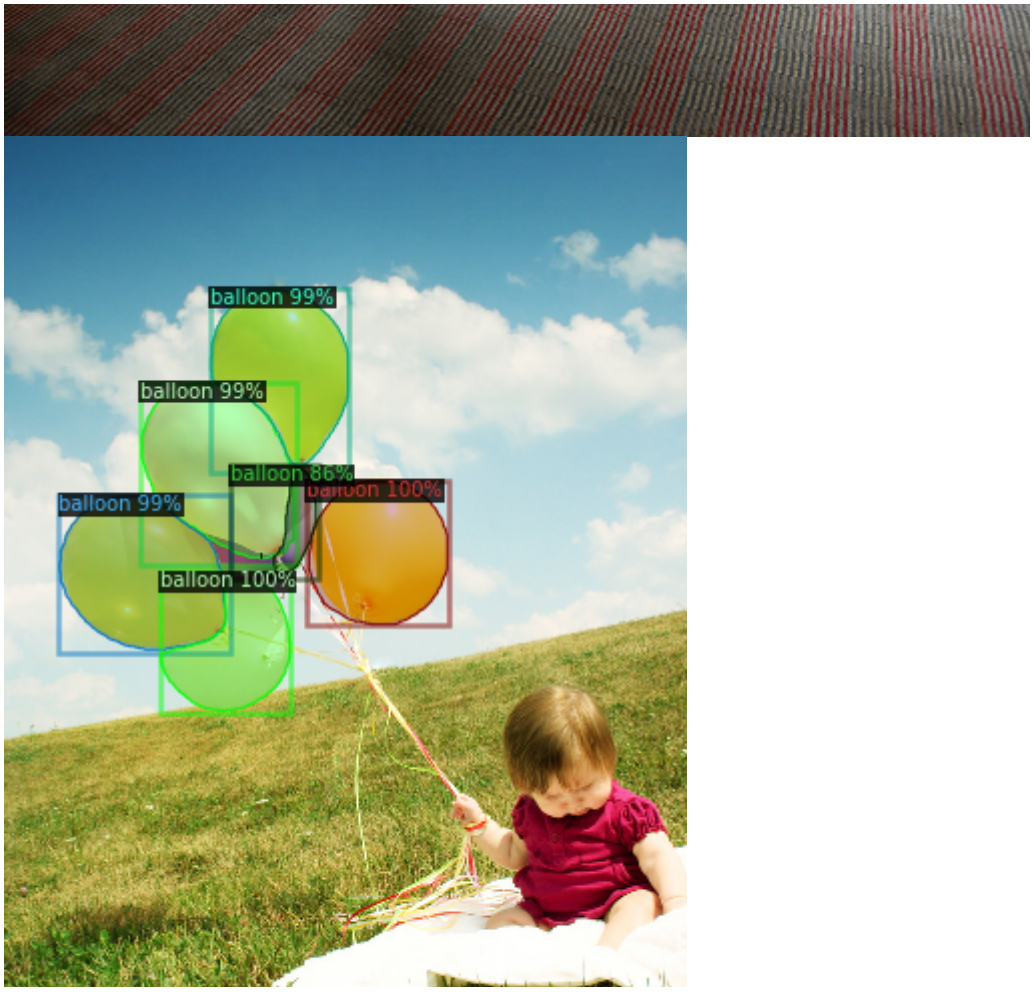
Now, let's run inference with the trained model on the balloon validation dataset. First, let's create a predictor using the model we just trained:

```
# Inference should use the config with parameters that are used in training
# cfg now already contains everything we've set previously. We changed it a little bit for
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")  # path to the model w
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set a custom testing threshold
predictor = DefaultPredictor(cfg)
```

Then, we randomly select several samples to visualize the prediction results.

```
from detectron2.utils.visualizer import ColorMode
dataset_dicts = get_balloon_dicts("balloon/val")
for d in random.sample(dataset_dicts, 3):
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)  # format is documented at https://detectron2.readthedocs.io/t
```

```
v = Visualizer(im[:, :, ::-1],
               metadata=balloon_metadata,
               scale=0.5,
               instance_mode=ColorMode.IMAGE_BW   # remove the colors of unsegmented p
)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
```



We can also evaluate its performance using AP metric implemented in COCO API. This gives an AP of ~70. Not bad!

```
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader
evaluator = COCOEvaluator("balloon_val", output_dir="./output")
val_loader = build_detection_test_loader(cfg, "balloon_val")
print(inference_on_dataset(predictor.model, val_loader, evaluator))
# another equivalent way to evaluate the model is to use `trainer.test`
```

```
[09/02 02:45:12 d2.data.common]: Serialized dataset takes 0.04 MiB
[09/02 02:45:12 d2.evaluation.evaluator]: Start inference on 13 batches
[09/02 02:45:16 d2.evaluation.evaluator]: Inference done 11/13. Dataloading: 0.001
[09/02 02:45:17 d2.evaluation.evaluator]: Total inference time: 0:00:02.871534 (0.
[09/02 02:45:17 d2.evaluation.evaluator]: Total inference pure compute time: 0:00:
[09/02 02:45:17 d2.evaluation.coco_evaluation]: Preparing results for COCO format
[09/02 02:45:17 d2.evaluation.coco_evaluation]: Saving results to ./output/coco_ir
[09/02 02:45:17 d2.evaluation.coco_evaluation]: Evaluating predictions with unoffi
Loading and preparing results...
DONE (t=0.00s)
creating index...
index created!
[09/02 02:45:17 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[09/02 02:45:17 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in
[09/02 02:45:17 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[09/02 02:45:17 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished i
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.730
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.838
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.809
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.526
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.903
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.240
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.746
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.746
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.565
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.923
[09/02 02:45:17 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 72.969 | 83.789 | 80.927 | 0.000 | 52.647 | 90.309 |
Loading and preparing results...
DONE (t=0.00s)
creating index...
index created!
[09/02 02:45:17 d2.evaluation.fast_eval_api]: Evaluate annotation type *segm*
[09/02 02:45:17 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in
[09/02 02:45:17 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[09/02 02:45:17 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished i
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.755
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.809
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.809
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.522
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.958
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.252
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.772
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.772
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.559
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.970
[09/02 02:45:17 d2.evaluation.coco_evaluation]: Evaluation results for segm:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl   |
|:------:|:------:|:------:|:-----:|:------:|:------:|
| 75.497 | 80.927 | 80.927 | 0.000 | 52.240 | 95.806 |
OrderedDict([('bbox', {'AP': 72.96887963199184, 'AP50': 83.78937893789379, 'AP75':
```

Other types of builtin models

We showcase simple demos of other types of models below:

```
# Inference with a keypoint detection model
cfg = get_cfg()   # get a fresh new config
cfg.merge_from_file(model_zoo.get_config_file("COCO-Keypoints/keypoint_rcnn_R_50_FPN_3x.ya
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7  # set threshold for this model
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Keypoints/keypoint_rcnn_R_50_FPN_3x
predictor = DefaultPredictor(cfg)
outputs = predictor(im)
v = Visualizer(im[:,:,::-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])
```

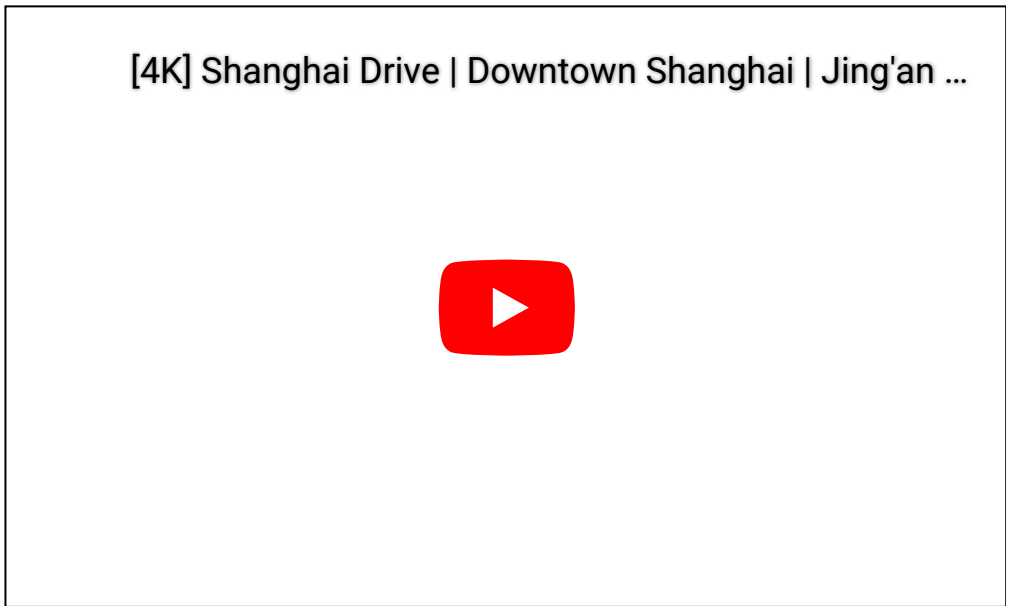
```
WARNING [10/15 06:33:32 d2.config.compat]: Config './detectron2_repo/configs/COCO-Ke
```


```
# Inference with a panoptic segmentation model
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-PanopticSegmentation/panoptic_fpn_R_10
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-PanopticSegmentation/panoptic_fpn_R
predictor = DefaultPredictor(cfg)
panoptic_seg, segments_info = predictor(im)["panoptic_seg"]
v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_panoptic_seg_predictions(panoptic_seg.to("cpu"), segments_info)
cv2_imshow(out.get_image()[:, :, ::-1])
```

```
WARNING [10/15 06:33:37 d2.config.compat]: Config './detectron2_repo/configs/COCO-Pa
```

Run panoptic segmentation on a video

```
# This is the video we're going to process
from IPython.display import YouTubeVideo, display
video = YouTubeVideo("ll8TgCZ0plk", width=500)
display(video)
```




```
# Install dependencies, download the video, and crop 5 seconds for processing
!pip install youtube-dl
!youtube-dl https://www.youtube.com/watch?v=ll8TgCZ0plk -f 22 -o video.mp4
!ffmpeg -i video.mp4 -t 00:00:06 -c:v copy video-clip.mp4
```