Here is your **Detectron2 Crash Course**.

Unlike YOLO (which is designed for "Download & Run"), **Detectron2** is designed for "Research & Customization." It is built by Facebook AI Research (FAIR) and is extremely modular.

This course assumes you are using **Google Colab** (highly recommended because installing Detectron2 locally on Windows is painful).

---

1. The Philosophy (YOLO vs. Detectron2)

- **YOLO:** "Here is a command line tool. Give me a folder of images, I give you a model."
- **Detectron2:** "Here is a Lego set. Assemble your own Backbone, Neck, and Head using a Python Config file."

2. Installation (The "Hard" Part)

Detectron2 depends heavily on strict PyTorch and CUDA versions. Run this in Colab to install it without tears.

```python
# 1. Install Dependencies
!python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'
# (Note: This takes ~2-3 minutes in Colab)
```

---

3. Hello World (Inference)

Let's run a pre-trained **Mask R-CNN** (Instance Segmentation) on a random image.

Key Concept: The `CfgNode` (Config).

In Detectron2, you don't pass arguments to functions. You create a `cfg` object, modify it, and pass *that* around.

```python
import cv2
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger() # Standard D2 setup

# Import core modules
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2 import model_zoo
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog

# 1. Get an Image
!wget http://images.cocodataset.org/val2017/000000439715.jpg -O input.jpg
im = cv2.imread("./input.jpg")

# 2. Setup Config (The "Brain")
cfg = get_cfg()
# Load architecture (Mask R-CNN with ResNet-50)
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
# Load pre-trained weights
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # set threshold for this model

# 3. Predict
predictor = DefaultPredictor(cfg)
outputs = predictor(im)

# 4. Visualize
# MetadataCatalog holds info like class names ("person", "horse")
v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))

# Show
from google.colab.patches import cv2_imshow
cv2_imshow(out.get_image()[:, :, ::-1])
```

4. Custom Training (The "Weird" Part)

This is where people get stuck. Detectron2 **does not** read folders like `train/` and `val/` automatically. You must **Register** your dataset.

Step A: Data Format

Detectron2 expects a specific JSON format (list of dicts):

```python
# What Detectron2 wants to see:
dataset_dicts = [
    {
        "file_name": "my_image.jpg",
        "height": 720,
        "width": 1280,
        "image_id": 1,
        "annotations": [
            {
                "bbox": [x, y, w, h],
                "bbox_mode": BoxMode.XYWH_ABS,
                "category_id": 0, # Class ID
                # "segmentation": [[poly_points...]] # Only if doing segmentation
            }
        ]
    }
]
```

Step B: The Catalog

You must tell Detectron2: "Hey, the string `'my_dataset_train'` refers to *this* function that returns the list above."

```python
from detectron2.data import DatasetCatalog, MetadataCatalog

# 1. Register the dataset
# 'get_my_dicts' is a function YOU write that returns the list above
DatasetCatalog.register("my_balloon_train", lambda: get_my_dicts("balloon/train"))
MetadataCatalog.get("my_balloon_train").set(thing_classes=["balloon"])

# 2. Setup Training Config
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("my_balloon_train",) # Point to your registered name
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  # Learning Rate
cfg.SOLVER.MAX_ITER = 300     # Epochs (sort of)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1  # We only have 1 class (balloon)

# 3. Train
from detectron2.engine import DefaultTrainer
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

---

5. Cheat Sheet: YOLO vs. Detectron2 Terms

YOLO Term	Detectron2 Equivalent	Description
`data.yaml`	`DatasetCatalog`	How the model finds your data.
`yolo train ...`	`DefaultTrainer(cfg).train()`	The training loop.
`model.predict()`	`DefaultPredictor(cfg)(img)`	Running the model.
`runs/detect/exp`	`output/`	Where logs and weights are saved.
`best.pt`	`model_final.pth`	The saved model weights.

Summary

- **Use Detectron2** if you need: Segmentation masks, Keypoints, or strict control over the architecture layers.
- **Don't use it** if you just want simple Bounding Box detection (YOLO is 10x easier).

For your **Parking Project**:

Sticking to **YOLO** was the right call. Writing a custom Detectron2 dataloader just to draw boxes around cars would have been overkill.