How to Train YOLOv8 Object Detection on a Custom Dataset

Roboflow Blog Youtube GitHub

Ultralytics YOLOv8 is a popular version of the YOLO (You Only Look Once) object detection and image segmentation model developed by Ultralytics. The YOLOv8 model is designed to be fast, accurate, and easy to use, making it an excellent choice for a wide range of object detection and image segmentation tasks. It can be trained on large datasets and is capable of running on a variety of hardware platforms, from CPUs to GPUs.

Disclaimer

If you notice that our notebook behaves incorrectly - especially if you experience errors that prevent you from going through the tutorial - don't hesitate! Let us know and open an issue on the Roboflow Notebooks repository.

Accompanying Blog Post

We recommend that you follow along in this notebook while reading the accompanying Blog Post.

Pro Tip: Use GPU Acceleration

If you are running this notebook in Google Colab, navigate to `Edit` -> `Notebook settings` -> `Hardware accelerator`, set it to `GPU`, and then click `Save`. This will ensure your notebook uses a GPU, which will significantly speed up model training times.

Steps in this Tutorial

In this tutorial, we are going to cover:

- Before you start
- Install YOLOv8
- CLI Basics
- Inference with Pre-trained COCO Model
- Roboflow Universe
- Preparing a custom dataset
- Custom Training
- Validate Custom Model
- Inference with Custom Model

Let's begin!

Before you start

Let's make sure that we have access to GPU. We can use `nvidia-smi` command to do that. In case of any problems navigate to `Edit` -> `Notebook settings` -> `Hardware accelerator`, set it to `GPU`, and then click `Save`.

```
!nvidia-smi
```

```
Thu May  1 22:17:06 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
```

```
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   36C    P8              9W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------+-----------------------------------------------+
```

```
import os
HOME = os.getcwd()
print(HOME)
```

```
/content
```

Install YOLOv8

YOLOv8 can be installed in two ways-from the source and via pip. This is because it is the first iteration of YOLO to have an official package.

```
# Pip install method (recommended)

!pip install ultralytics==8.2.103 -q

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()
```

```
Ultralytics YOLOv8.2.103 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 41.7/112.6 GB disk)
```

```
from ultralytics import YOLO

from IPython.display import display, Image
```

CLI Basics

If you want to train, validate or run inference on models and don't need to make any modifications to the code, using YOLO command line interface is the easiest way to get started. Read more about CLI in Ultralytics YOLO Docs.

```
 yolo task=detect    mode=train    model=yolov8n.yaml      args...
           classify       predict        yolov8n-cls.yaml  args...
           segment        val            yolov8n-seg.yaml  args...
                          export         yolov8n.pt        format=onnx  args...
```

Inference with Pre-trained COCO Model

💻 CLI

`yolo mode=predict` runs YOLOv8 inference on a variety of sources, downloading models automatically from the latest YOLOv8 release, and saving results to `runs/predict`.

```
%cd {HOME}
!yolo task=detect mode=predict model=yolov8n.pt conf=0.25 source='https://media.roboflow.com/notebooks/examples/dog.jpeg' save=True
```

```
/content
Downloading https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov8n.pt to 'yolov8n.pt'...
100% 6.25M/6.25M [00:00<00:00, 78.7MB/s]
Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8n summary (fused): 168 layers, 3,151,904 parameters, 0 gradients, 8.7 GFLOPs

Downloading https://media.roboflow.com/notebooks/examples/dog.jpeg to 'dog.jpeg'...
100% 104k/104k [00:00<00:00, 76.9MB/s]
image 1/1 /content/dog.jpeg: 640x384 1 person, 1 car, 1 dog, 124.5ms
Speed: 17.1ms preprocess, 124.5ms inference, 2178.2ms postprocess per image at shape (1, 3, 640, 384)
Results saved to runs/detect/predict
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

```
%cd {HOME}
Image(filename='runs/detect/predict/dog.jpeg', height=600)
```

```
/content
```

Python SDK

The simplest way of simply using YOLOv8 directly in a Python environment.

```
model = YOLO(f'{HOME}/yolov8n.pt')
results = model.predict(source='https://media.roboflow.com/notebooks/examples/dog.jpeg', conf=0.25)
```

```
Found https://media.roboflow.com/notebooks/examples/dog.jpeg locally at dog.jpeg
image 1/1 /content/dog.jpeg: 640x384 1 person, 1 car, 1 dog, 82.0ms
Speed: 3.4ms preprocess, 82.0ms inference, 1081.0ms postprocess per image at shape (1, 3, 640, 384)
```

```
results[0].boxes.xyxy
```

```
tensor([[   0.0000,  314.4717,  625.0754, 1278.1946],
        [  55.1731,  250.0220,  648.1080, 1266.2720],
        [ 633.2291,  719.5391,  701.0538,  786.0336]], device='cuda:0')
```

```
results[0].boxes.conf
```

```
tensor([0.7271, 0.2907, 0.2846], device='cuda:0')
```

```
results[0].boxes.cls
```

```
tensor([ 0., 16.,  2.], device='cuda:0')
```

Roboflow Universe

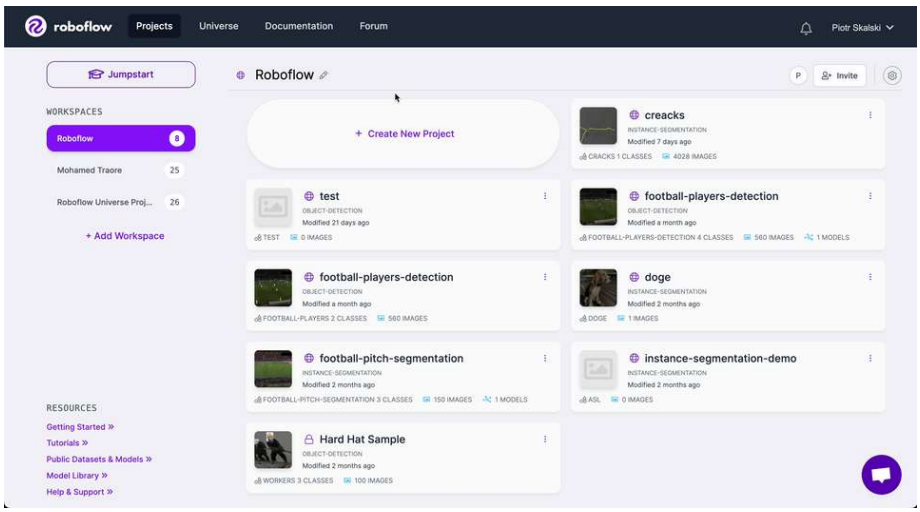
Need data for your project? Before spending time on annotating, check out Roboflow Universe, a repository of more than 110,000 open-source datasets that you can use in your projects. You'll find datasets containing everything from annotated cracks in concrete to plant images with disease annotations.

Preparing a custom dataset

Building a custom dataset can be a painful process. It might take dozens or even hundreds of hours to collect images, label them, and export them in the proper format. Fortunately, Roboflow makes this process as straightforward and fast as possible. Let me show you how!

Step 1: Creating project

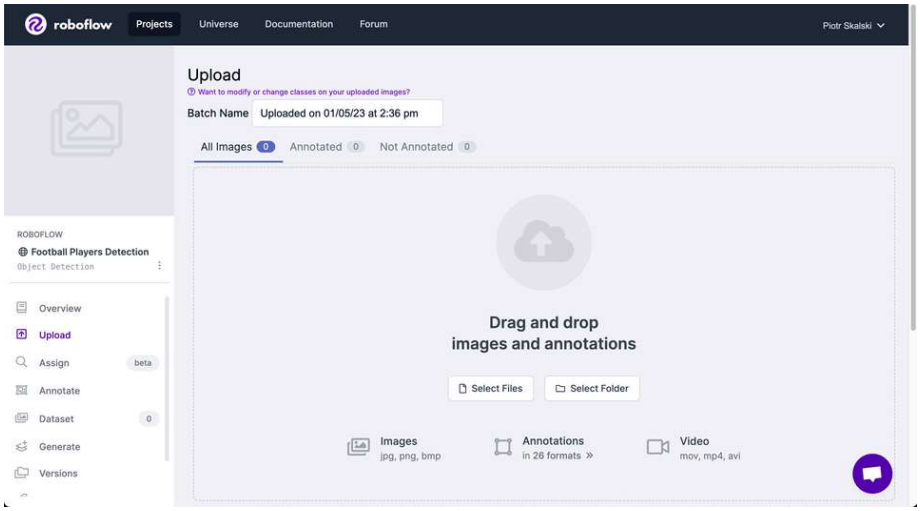
Before you start, you need to create a Roboflow account. Once you do that, you can create a new project in the Roboflow dashboard. Keep in mind to choose the right project type. In our case, Object Detection.



Step 2: Uploading images

Next, add the data to your newly created project. You can do it via API or through our web interface.

If you drag and drop a directory with a dataset in a supported format, the Roboflow dashboard will automatically read the images and annotations together.

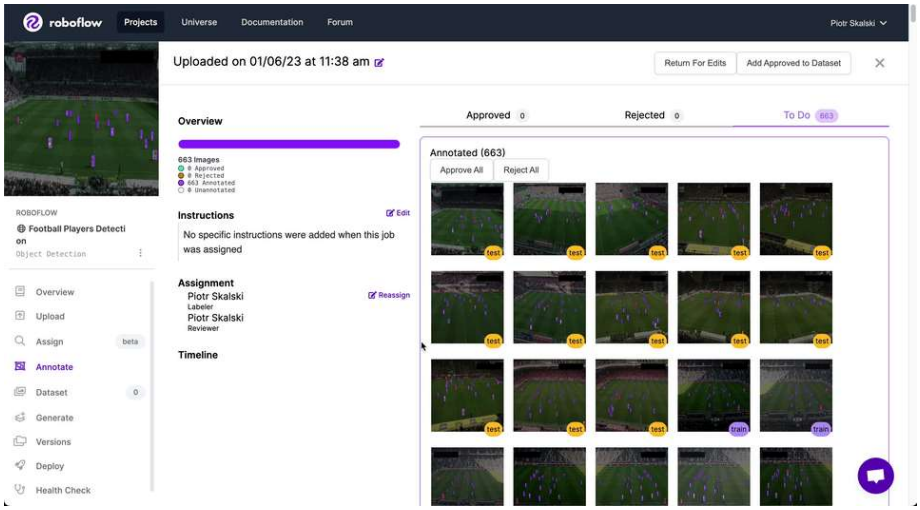


Step 3: Labeling

If you only have images, you can label them in [Roboflow Annotate](#).

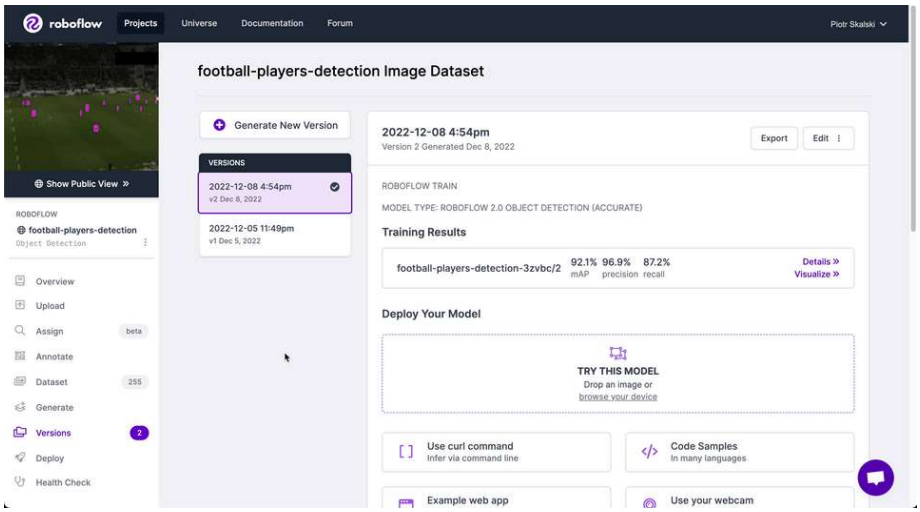
Step 4: Generate new dataset version

Now that we have our images and annotations added, we can Generate a Dataset Version. When Generating a Version, you may elect to add preprocessing and augmentations. This step is completely optional, however, it can allow you to significantly improve the robustness of your model.



Step 5: Exporting dataset

Once the dataset version is generated, we have a hosted dataset we can load directly into our notebook for easy training. Click `Export` and select the `YOLO v8` dataset format. (Formerly, we used to use `Yolov5`, as the gif shows)



🟢 Tip: The examples below work even if you use our non-custom model. However, you won't be able to deploy it to Roboflow. To do that, create a custom dataset as described below or fork (copy) one into your workspace from Universe.

```
!mkdir -p {HOME}/datasets
%cd {HOME}/datasets

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="tUGmIJPx91GrHX5tPe7B")
project = rf.workspace("final-year-design").project("object-detection-7mpdb")
version = project.version(2)
dataset = version.download("yolov8")
```

```
/content/{HOME}/datasets
Requirement already satisfied: roboflow in /usr/local/lib/python3.11/dist-packages (1.1.63)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from roboflow) (2025.4.26)
Requirement already satisfied: idna==3.7 in /usr/local/lib/python3.11/dist-packages (from roboflow) (3.7)
Requirement already satisfied: cycler in /usr/local/lib/python3.11/dist-packages (from roboflow) (0.12.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.4.8)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from roboflow) (3.10.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.26.4)
Requirement already satisfied: opencv-python-headless==4.10.0.84 in /usr/local/lib/python3.11/dist-packages (from roboflow) (4.
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.11/dist-packages (from roboflow) (11.2.1)
Requirement already satisfied: pillow-heif>=0.18.0 in /usr/local/lib/python3.11/dist-packages (from roboflow) (0.22.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.9.0.post0)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.32.3)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.17.0)
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.4.0)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.11/dist-packages (from roboflow) (4.67.1)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.11/dist-packages (from roboflow) (6.0.2)
Requirement already satisfied: requests-toolbelt in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.0.0)
Requirement already satisfied: filetype in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->roboflow) (1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->roboflow) (4.57.0
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->roboflow) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->roboflow) (3.2.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->roboflow) (3
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in Object-Detection-2 to yolov8:: 100%|██████████| 9900/9900 [00:00<00:00, 30135.21it/s]

Extracting Dataset Version Zip to Object-Detection-2 in yolov8:: 100%|██████████| 450/450 [00:00<00:00, 8388.35it/s]
```

Custom Training

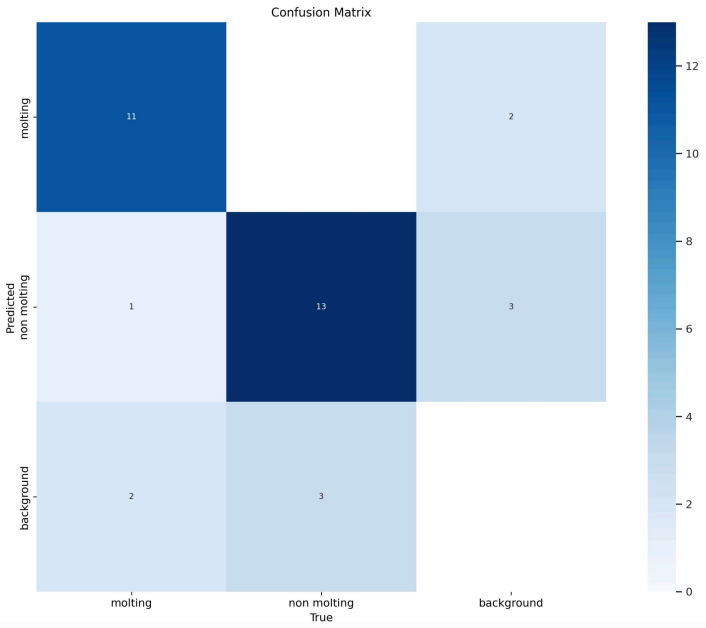
```
%cd {HOME}
```

```
%cd {HOME}

!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=25 imgsz=800 plots=True
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      11/25      5.82G      1.459      1.579      1.888         73        800: 100% 12/12 [00:05<00:00,  2.34it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  1.97it/s]
                   all         18         30      0.225      0.301      0.197       0.08

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      12/25      6.07G       1.43      1.599      1.939         48        800: 100% 12/12 [00:04<00:00,  2.42it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.75it/s]
                   all         18         30     0.0586      0.795     0.0572     0.0253

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      13/25      5.82G      1.402      1.603      1.887         42        800: 100% 12/12 [00:05<00:00,  2.32it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  2.48it/s]
                   all         18         30      0.666      0.433      0.487      0.206

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      14/25      5.82G      1.378      1.512      1.868         59        800: 100% 12/12 [00:04<00:00,  2.43it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.47it/s]
                   all         18         30      0.695      0.754      0.779      0.334

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      15/25      5.82G      1.303      1.342      1.779         54        800: 100% 12/12 [00:05<00:00,  2.34it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  2.07it/s]
                   all         18         30      0.517      0.618      0.547      0.211
Closing dataloader mosaic
/usr/local/lib/python3.11/dist-packages/ultralytics/data/augment.py:1837: UserWarning: Argument(s) 'quality_lower' are not v
  A.ImageCompression(quality_lower=75, p=0.0),
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_aver

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      16/25      6.07G      1.553      1.447      2.158         24        800: 100% 12/12 [00:06<00:00,  1.86it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  2.94it/s]
                   all         18         30      0.715      0.629      0.661      0.284

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      17/25      5.83G      1.417      1.249      2.171         26        800: 100% 12/12 [00:05<00:00,  2.28it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.24it/s]
                   all         18         30      0.668      0.571      0.563      0.231

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      18/25      5.83G      1.509      1.255      2.112         31        800: 100% 12/12 [00:05<00:00,  2.40it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.80it/s]
                   all         18         30       0.65      0.634      0.623      0.288

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      19/25      6.07G      1.387      1.244      2.136         21        800: 100% 12/12 [00:05<00:00,  2.29it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.04it/s]
                   all         18         30      0.772      0.473      0.591      0.311

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      20/25      5.83G      1.318       1.14      2.061         20        800: 100% 12/12 [00:04<00:00,  2.44it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.18it/s]
                   all         18         30      0.804      0.536      0.632      0.276

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      21/25      5.82G      1.335      1.132      2.015         20        800: 100% 12/12 [00:05<00:00,  2.29it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  3.05it/s]
```

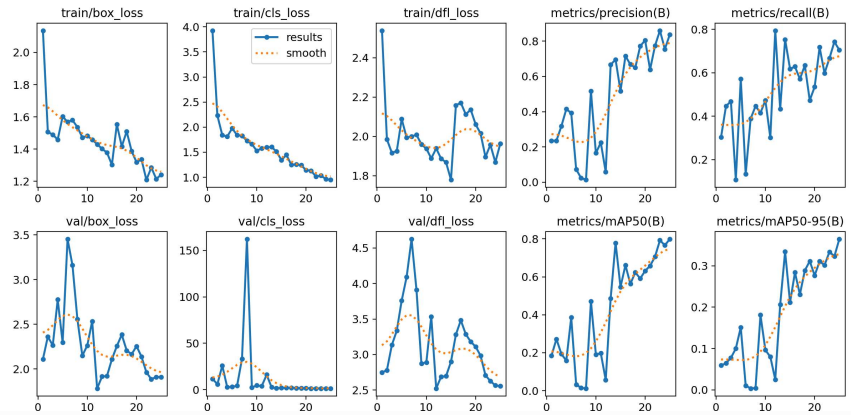
```
%cd /content/{HOME}
%ls
Image(filename=f'datasets/Object-Detection-2/runs/detect/train8/confusion_matrix.png', width=600)
```

```
/content/{HOME}
datasets/
```



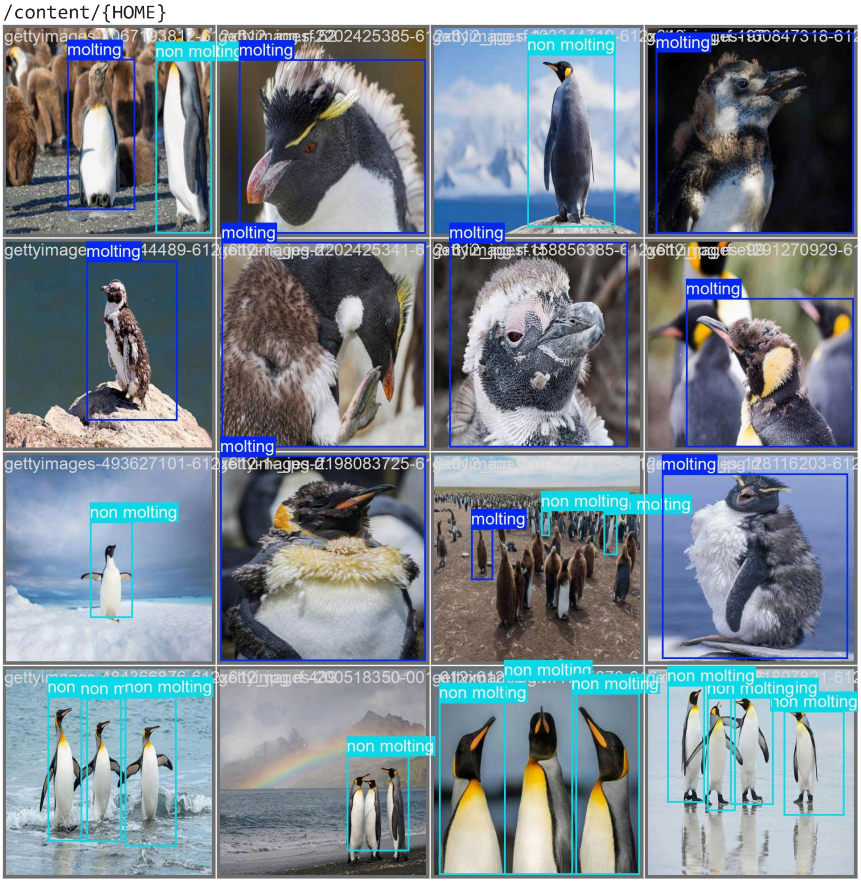

```
%cd /content/{HOME}
Image(filename=f'datasets/Object-Detection-2/runs/detect/train8/results.png', width=600)
```

```
/content/{HOME}
```




```
%cd /content/{HOME}
Image(filename=f'datasets/Object-Detection-2/runs/detect/train8/val_batch0_labels.jpg', width=600)
```

```
/content/{HOME}
```



Validate Custom Model

```
%cd /content/{HOME}
%ls
!yolo task=detect mode=val model=datasets/Object-Detection-2/runs/detect/train8/weights/best.pt data={dataset.location}/data.yaml
```

```
/content/{HOME}
datasets/
Ultralytics YOLOv8.2.103 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
Model summary (fused): 168 layers, 11,126,358 parameters, 0 gradients, 28.4 GFLOPs
val: Scanning /content/{HOME}/datasets/Object-Detection-2/valid/labels.cache... 18 images, 0 backgrounds, 0 corrupt: 100% 18/18
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:00<00:00,  2.09it/s]
                   all         18         30      0.832      0.705      0.799      0.364
               molting         12         14      0.915      0.786      0.869      0.466
           non molting          8         16      0.748      0.625      0.729      0.262
Speed: 0.5ms preprocess, 28.8ms inference, 0.0ms loss, 9.7ms postprocess per image
Results saved to runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```

Inference with Custom Model

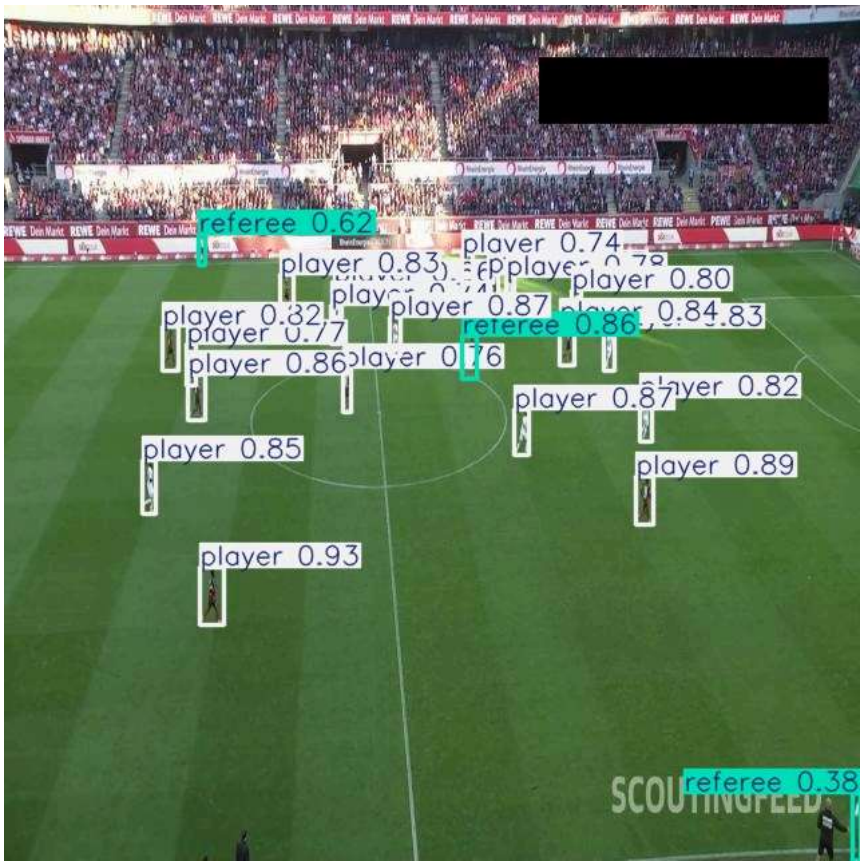
```
%cd {HOME}
!yolo task=detect mode=predict model={HOME}/runs/detect/train/weights/best.pt conf=0.25 source={dataset.location}/test/images save=
```

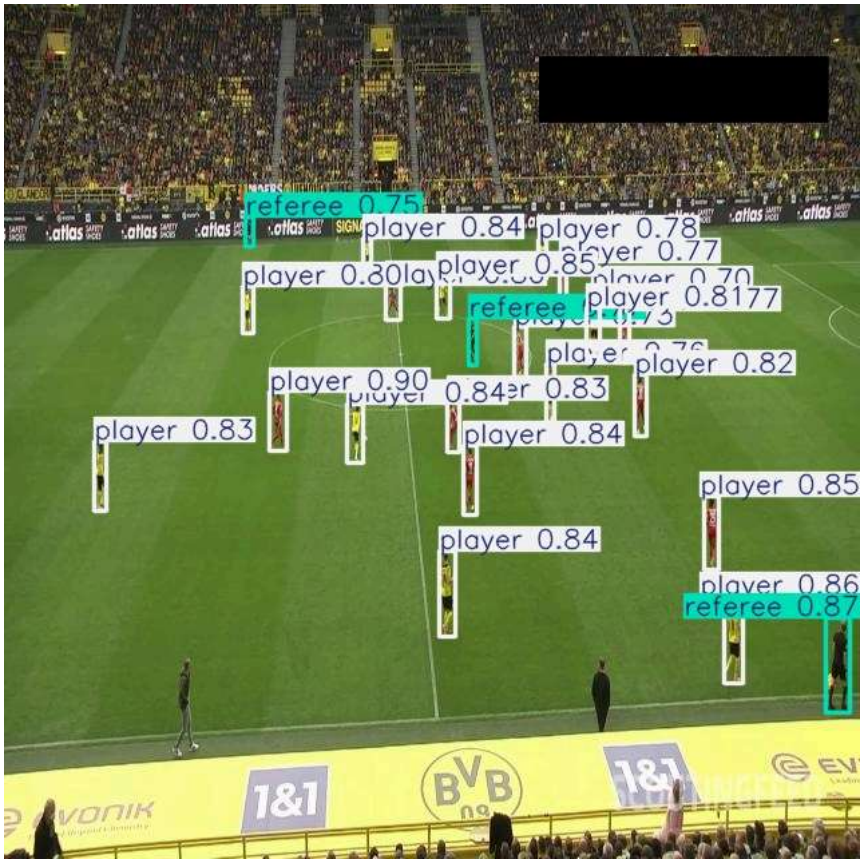
```
/content
Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11,127,132 parameters, 0 gradients, 28.4 GFLOPs

image 1/25 /content/datasets/football-players-obj-detection-2/test/images/08fd33_3_6_png.rf.d031da4f257bbf91daf9845051fb9487.jp
image 2/25 /content/datasets/football-players-obj-detection-2/test/images/08fd33_9_3_png.rf.509869506accd728341d8426de2f937a.jp
image 3/25 /content/datasets/football-players-obj-detection-2/test/images/40cd38_7_6_png.rf.57bcce20df01cee3811f2b3576481f41.jp
image 4/25 /content/datasets/football-players-obj-detection-2/test/images/42ba34_1_5_png.rf.ec272c23ac9c73ba68d85a630b6d2a22.jp
image 5/25 /content/datasets/football-players-obj-detection-2/test/images/42ba34_5_5_png.rf.3fecd509745cf069d3e57692550f5fa9.jp
image 6/25 /content/datasets/football-players-obj-detection-2/test/images/4b770a_1_4_png.rf.8d0412c27f5ac2c033a18d9e1094c815.jp
```

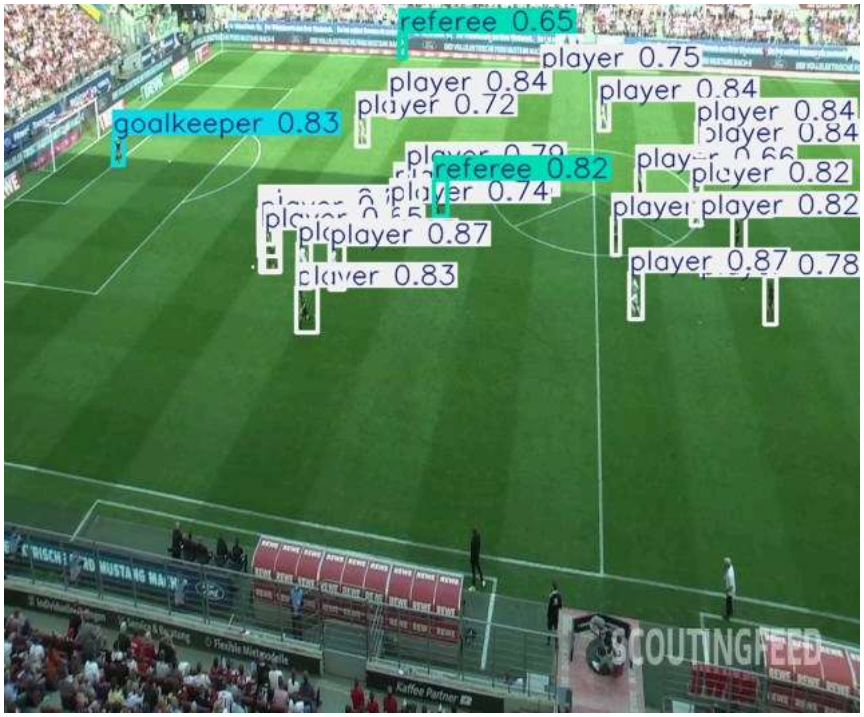
```
image 7/25 /content/datasets/football-players-obj-detection-2/test/images/4b770a_1_6_png.rf.b956639beea9d158dba2a227cd0154be.jp
image 8/25 /content/datasets/football-players-obj-detection-2/test/images/4b770a_1_8_png.rf.e88f62b8439bc31729f8f90c3a3a40ff.jp
image 9/25 /content/datasets/football-players-obj-detection-2/test/images/4b770a_3_6_png.rf.dacd11e6b0ecd7f7a23820a054aa9575.jp
image 10/25 /content/datasets/football-players-obj-detection-2/test/images/4b770a_3_9_png.rf.2784198c6be2ceb1d35e8e265def9ff9.j
image 11/25 /content/datasets/football-players-obj-detection-2/test/images/4b770a_5_1_png.rf.daf24d263b4b1c6850cda944c524b0c8.j
image 12/25 /content/datasets/football-players-obj-detection-2/test/images/4b770a_9_3_png.rf.6e86d8139dfbc9ae60a98534874eea62.j
image 13/25 /content/datasets/football-players-obj-detection-2/test/images/4b770a_9_4_png.rf.e20d272c80881d8fbce4283dd4dfd2d2.j
image 14/25 /content/datasets/football-players-obj-detection-2/test/images/538438_3_6_png.rf.30190e7f052bf4e0429476d612ad6863.j
image 15/25 /content/datasets/football-players-obj-detection-2/test/images/538438_5_2_png.rf.8c1bf652ec47dabbac95b344dfa74194.j
image 16/25 /content/datasets/football-players-obj-detection-2/test/images/573e61_1_9_png.rf.b2a094e9072b3757d1d163f751eef921.j
image 17/25 /content/datasets/football-players-obj-detection-2/test/images/573e61_9_6_png.rf.f4c0d6ef36926439ae30210ff7af268c.j
image 18/25 /content/datasets/football-players-obj-detection-2/test/images/744b27_1_10_png.rf.39e8c8fd32fdd469055cafaf083049c2.
image 19/25 /content/datasets/football-players-obj-detection-2/test/images/744b27_7_4_png.rf.98562c4750de48d41c898725a930e28b.j
image 20/25 /content/datasets/football-players-obj-detection-2/test/images/744b27_9_9_png.rf.92671749e5e36cfb9f346d74def03ad9.j
image 21/25 /content/datasets/football-players-obj-detection-2/test/images/798b45_3_10_png.rf.2f817e24b9b8fbea03ac9d2070fbd439.
image 22/25 /content/datasets/football-players-obj-detection-2/test/images/798b45_3_3_png.rf.ddd3be8d34f05ac1ead1594696f3f910.j
image 23/25 /content/datasets/football-players-obj-detection-2/test/images/798b45_7_8_png.rf.1f84905c609f776e8b19d256856245c5.j
image 24/25 /content/datasets/football-players-obj-detection-2/test/images/a9f16c_2_10_png.rf.a15d501e64f5cd76f297ed8615df02ff.
image 25/25 /content/datasets/football-players-obj-detection-2/test/images/a9f16c_2_9_png.rf.57613ad41d945e73f6cc79554a676667.j
Speed: 3.6ms preprocess, 16.3ms inference, 23.7ms postprocess per image at shape (1, 3, 800, 800)
Results saved to runs/detect/predict2
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

NOTE: Let's take a look at few results.

```
import glob
from IPython.display import Image, display

# Define the base path where the folders are located
base_path = '/content/runs/detect/'

# List all directories that start with 'predict' in the base path
subfolders = [os.path.join(base_path, d) for d in os.listdir(base_path)
              if os.path.isdir(os.path.join(base_path, d)) and d.startswith('predict')]

# Find the latest folder by modification time
latest_folder = max(subfolders, key=os.path.getmtime)

image_paths = glob.glob(f'{latest_folder}/*.jpg')[:3]

# Display each image
for image_path in image_paths:
    display(Image(filename=image_path, width=600))
    print("\n")
```


referee 0.62
player 0.74
player 0.83
player 0.80
player 0.87
player 0.82
referee 0.86
player 0.77
player 0.86
player 0.87
player 0.85
player 0.89
player 0.93
referee 0.38


referee 0.75
player 0.84
player 0.78
player 0.77
player 0.85
player 0.80
player 0.8177
referee
player 0.82
player 0.90
player 0.83
player 0.83
player 0.84
player 0.85
player 0.84
referee 0.87



Deploy model on Roboflow

Once you have finished training your YOLOv8 model, you'll have a set of trained weights ready for use. These weights will be in the `/runs/detect/train/weights/best.pt` folder of your project. You can upload your model weights to Roboflow Deploy to use your trained weights on our infinitely scalable infrastructure.

The `.deploy()` function in the Roboflow pip package now supports uploading YOLOv8 weights.

To upload model weights, add the following code to the "Inference with Custom Model" section in the aforementioned notebook:

```
project.version(dataset.version).deploy(model_type="yolov8", model_path=f"{HOME}/runs/detect/train/")
```

```
Dependency ultralytics==8.0.196 is required but found version=8.2.103, to fix: `pip install ultralytics==8.0.196`
Would you like to continue with the wrong version of ultralytics? y/n: y
View the status of your deployment at: https://app.roboflow.com/model-examples/football-players-obj-detection/2
Share your model with the world at: https://universe.roboflow.com/model-examples/football-players-obj-detection/model/2
```

Follow the links above to check if the upload succeeded. It may take a couple of minutes until the model is visible to the `roboflow` SDK.

```
# Run inference on your model on a persistent, auto-scaling, cloud API

# Load model
model = project.version(dataset.version).model
assert model, "Model deployment is still loading"

# Choose a random test image
import os, random
test_set_loc = dataset.location + "/test/images/"
random_test_image = random.choice(os.listdir(test_set_loc))
print("running inference on " + random_test_image)

pred = model.predict(test_set_loc + random_test_image, confidence=40, overlap=30).json()
pred
```

```
running inference on 4b770a_3_9_png.rf.26fd0dc802e143501b91eddef365a94d.jpg
{'predictions': [{'x': 1227.0,
   'y': 527.5,
   'width': 50.0,
   'height': 77.0,
   'confidence': 0.9045102000236511,
```

```
    'class': 'player',
    'class_id': 2,
    'detection_id': '7bb0de78-c58d-454a-b693-c3f518f94f80',
    'image_path': '/content/datasets/football-players-obj-detection-
1/test/images/4b770a_3_9_png.rf.26fd0dc802e143501b91eddef365a94d.jpg',
    'prediction_type': 'ObjectDetectionModel'},
   {'x': 731.0,
    'y': 584.5,
    'width': 52.0,
    'height': 79.0,
    'confidence': 0.8924632668495178,
    'class': 'player',
    'class_id': 2,
    'detection_id': 'b3ea8a5a-5294-45c9-9221-ba8a6f0884b5',
    'image_path': '/content/datasets/football-players-obj-detection-
1/test/images/4b770a_3_9_png.rf.26fd0dc802e143501b91eddef365a94d.jpg',
    'prediction_type': 'ObjectDetectionModel'},
   {'x': 1397.5,
    'y': 344.5,
    'width': 31.0,
    'height': 55.0,
    'confidence': 0.8912790417671204,
    'class': 'player',
    'class_id': 2,
    'detection_id': '5ef87756-163e-442c-ad88-35f3572750d5',
    'image_path': '/content/datasets/football-players-obj-detection-
1/test/images/4b770a_3_9_png.rf.26fd0dc802e143501b91eddef365a94d.jpg',
    'prediction_type': 'ObjectDetectionModel'},
   {'x': 1447.0,
    'y': 476.5,
    'width': 32.0,
    'height': 73.0,
    'confidence': 0.8694518804550171,
    'class': 'player',
    'class_id': 2,
    'detection_id': '89a9e4d7-0b7b-4cc0-8596-cf90522a5f86',
    'image_path': '/content/datasets/football-players-obj-detection-
1/test/images/4b770a_3_9_png.rf.26fd0dc802e143501b91eddef365a94d.jpg',
    'prediction_type': 'ObjectDetectionModel'},
   {'x': 1271.0,
    'y': 583.0,
    'width': 36.0,
    'height': 82.0,
    'confidence': 0.8614903688430786,
    'class': 'player',
    'class_id': 2,
    'detection_id': '74b59e7e-03a8-477c-bb65-872b55cee6d2',
    'image_path': '/content/datasets/football-players-obj-detection-
1/test/images/4b770a_3_9_png.rf.26fd0dc802e143501b91eddef365a94d.jpg',
    'prediction_type': 'ObjectDetectionModel'},
   {'x': 332.5,
    'y': 474.0,
```


Deploy Your Model to the Edge

In addition to using the Roboflow hosted API for deployment, you can use Roboflow Inference, an open source inference solution that has powered millions of API calls in production environments. Inference works with CPU and GPU, giving you immediate access to a range of devices, from the NVIDIA Jetson to TRT-compatible devices to ARM CPU devices.

With Roboflow Inference, you can self-host and deploy your model on-device. You can deploy applications using the Inference Docker containers or the pip package.

For example, to install Inference on a device with an NVIDIA GPU, we can use:

```
docker pull roboflow/roboflow-inference-server-gpu
```

Then we can run inference via HTTP:

```
import requests

workspace_id = ""
model_id = ""
```