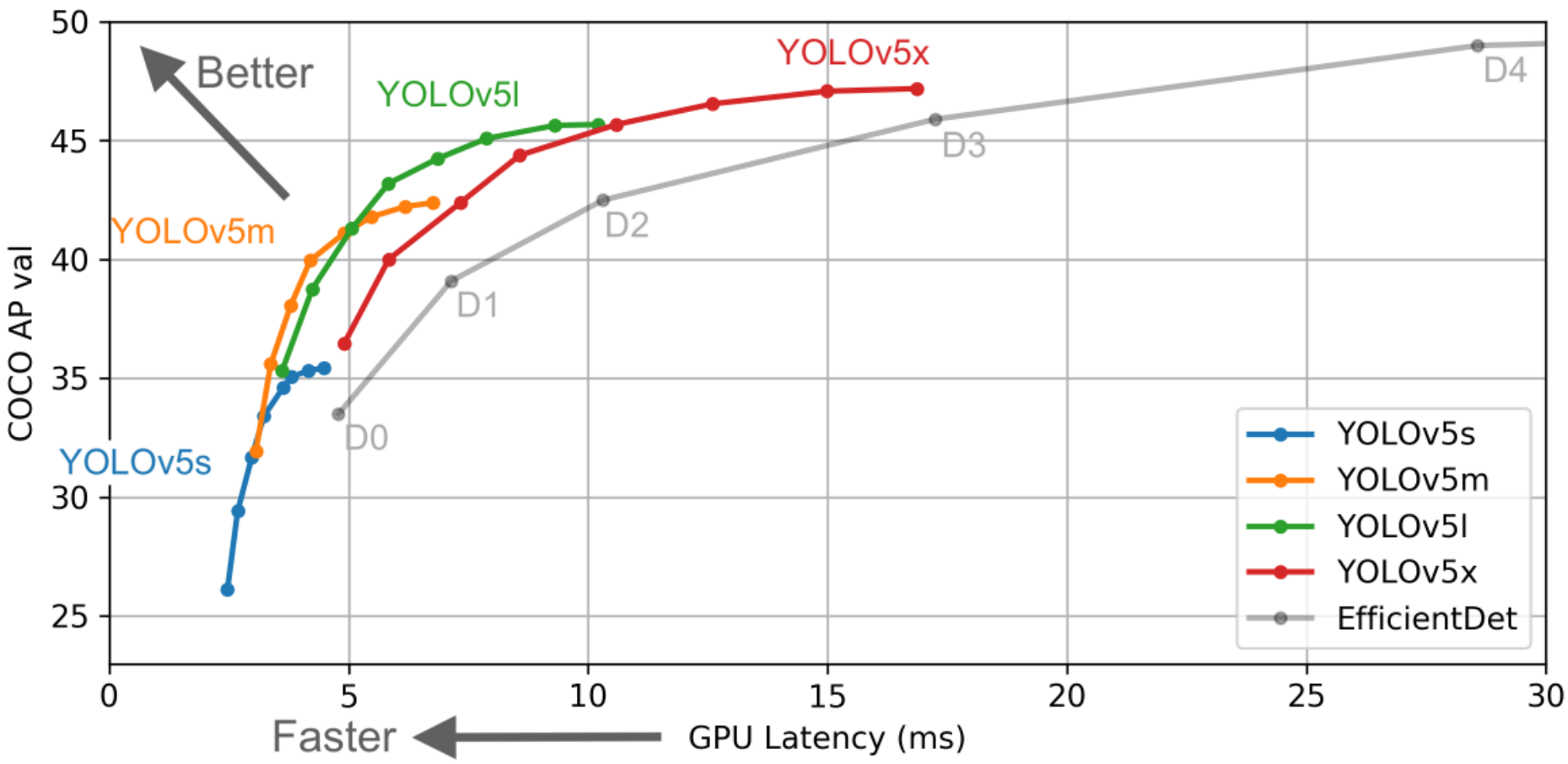
Deep Learning Computer Vision Awesome Guides

How to Create an End to End Object Detector using Yolov5

By Rahul Agarwal 04 August 2020

Ultralytics recently launched YOLOv5 amid controversy surrounding its name. For context, the first three versions of YOLO (You Only Look Once) were created by Joseph Redmon. Following this, Alexey Bochkovskiy created YOLOv4 on darknet, which boasted higher Average Precision (AP) and faster results than previous iterations.

Now, Ultralytics has released YOLOv5, with comparable AP and faster inference times than YOLOv4. This has left many asking: is a new version warranted given similar accuracy to YOLOv4? Whatever the answer may be, it's definitely a sign of how quickly the detection community is evolving.



Since they first ported YOLOv3 , Ultralytics has made it very simple to create and deploy models using Pytorch, so I was eager to try out YOLOv5. As it turns out, Ultralytics has further simplified the process, and the results speak for themselves.

In this article, we'll create a detection model using YOLOv5, from creating our dataset and annotating it to training and inferencing using their remarkable library. This post focuses on the implementation of YOLOv5, including:

- Creating a toy dataset
- Annotating the image data
- Creating the project structure
- Training YOLOv5

Creating Custom Dataset

You can forgo the first step if you have your image Dataset. Since I don't have images, I am downloading data from the Open Image Dataset(OID), which is an excellent resource for getting annotated image data that can be used for classification as well as detection. Note that we won't be using the provided annotations from OID and create our own for the sake of learning.

1. OIDv4 Download Images:

To download images from the Open Image dataset, we start by cloning the OIDv4_ToolKit and installing all requirements.

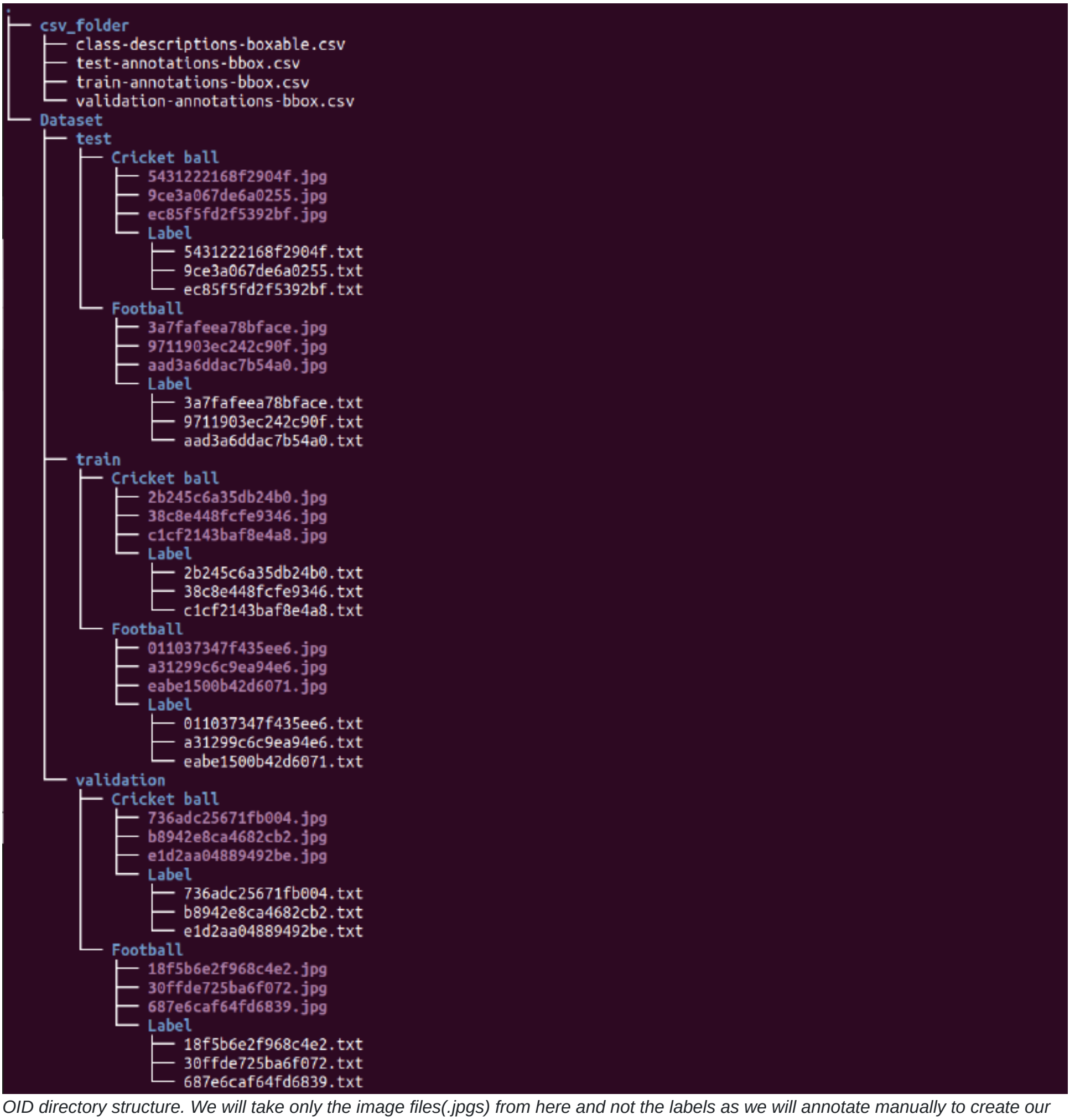
```
git clone [https://github.com/EscVM/OIDv4_ToolKit](https://github.com/EscVM/OIDv4_ToolKit)
cd [OIDv4_ToolKit](https://github.com/EscVM/OIDv4_ToolKit)
pip install -r requirements.txt
```

We can now use the main.py script within this folder to download images as well as labels for multiple classes.

Below I am downloading the data for Cricketball and Football to create our Custom Dataset. That is, we will be creating a dataset with footballs and cricket balls, and the learning task is to detect these balls.

```
python3 main.py downloader --classes Cricket_ball  Football --type_csv all -y --limit 500
```

The below command creates a directory named "OID" with the following structure:



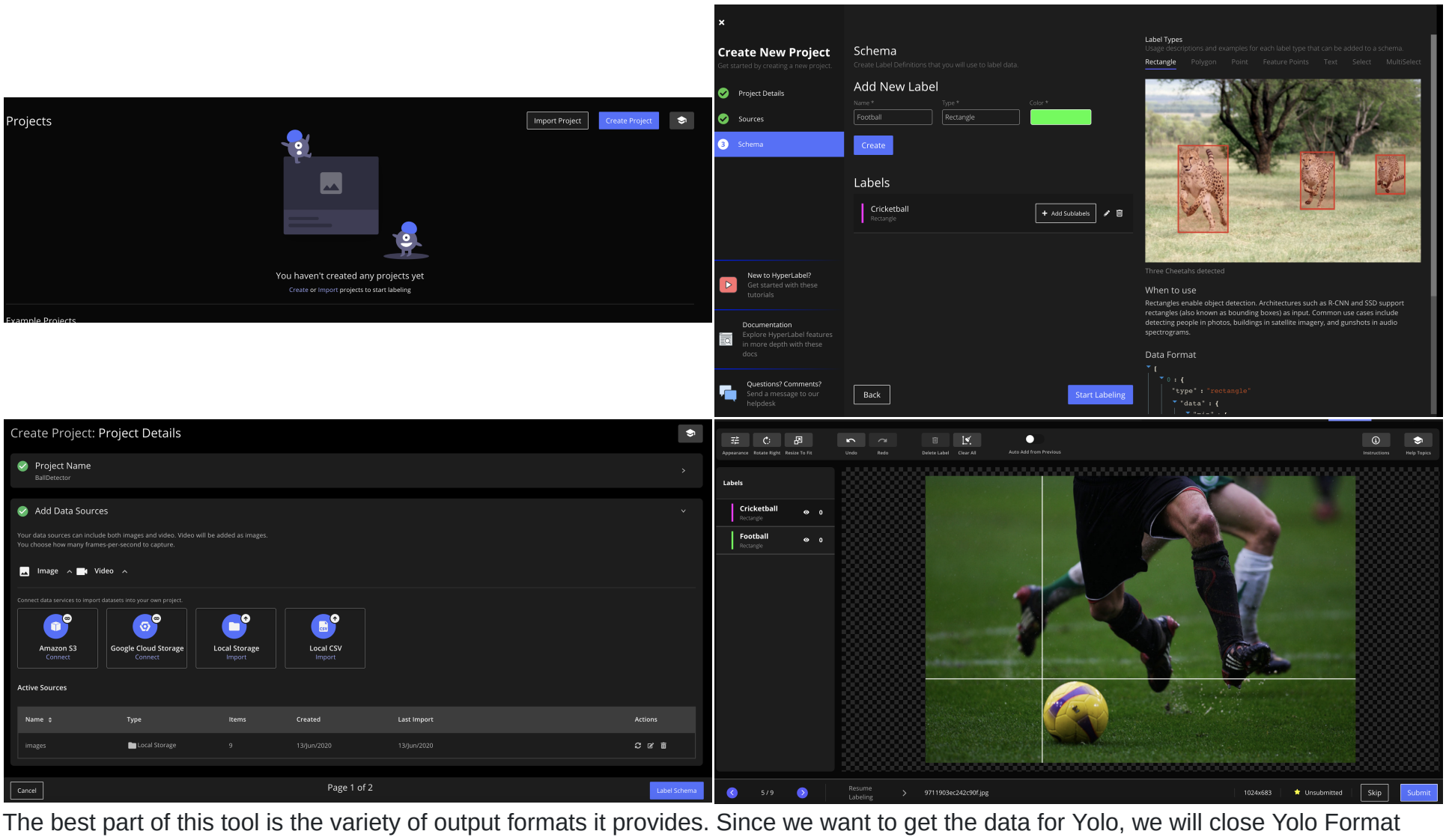
OID directory structure. We will take only the image files(.jpgs) from here and not the labels as we will annotate manually to create our Custom Dataset, though we can use them if required for a different project.

Before we continue, we will need to copy all the images in the same folder to start our labeling exercise from Scratch. You can choose to do this manually, but this can also be quickly done programmatically using recursive glob function:

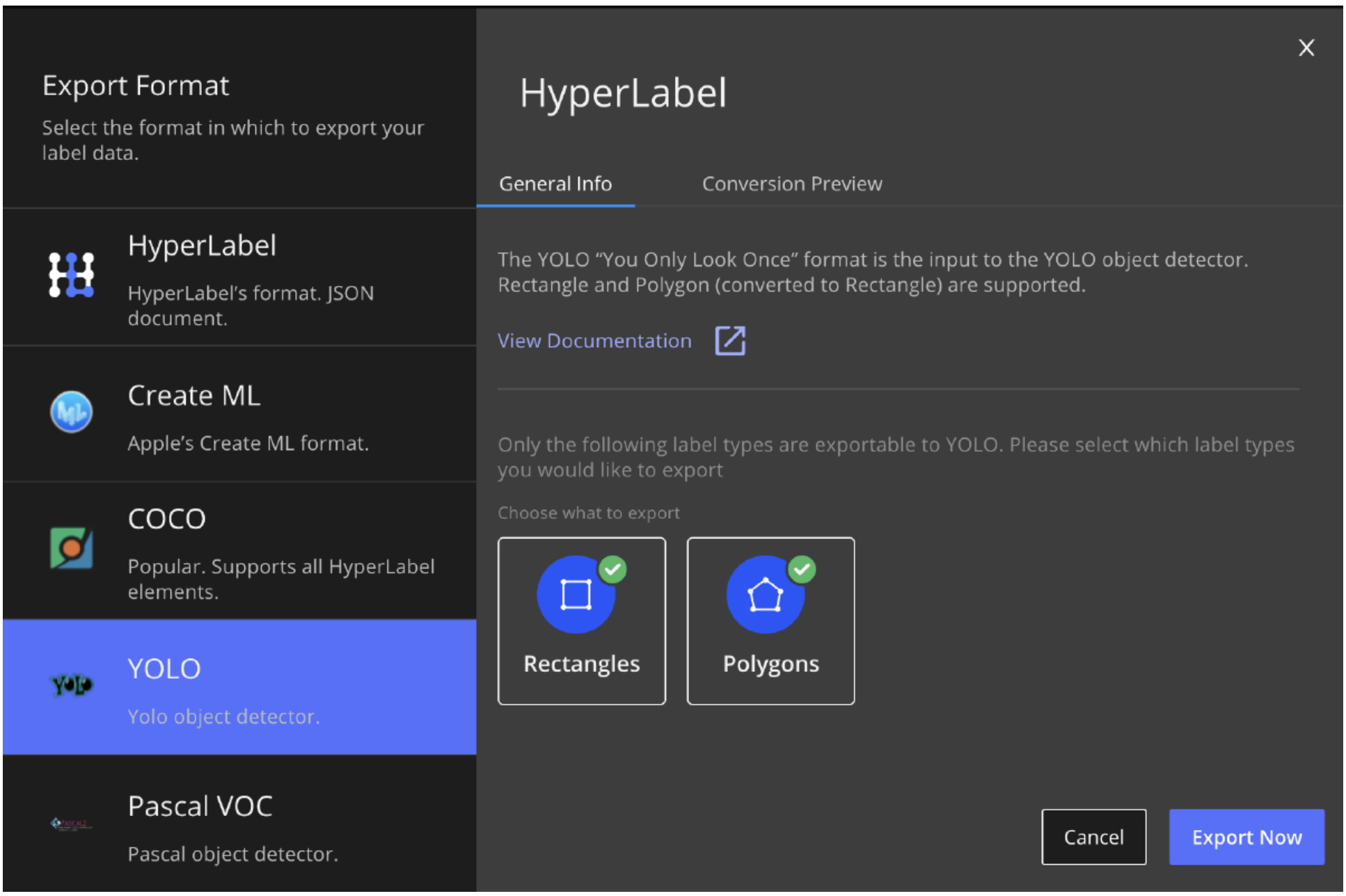
```
import os
from glob import glob

os.system("mkdir Images")
images = glob(r'OID/**/*.jpg', recursive=True)
for img in images:
    os.system(f"cp {img} Images/")
```

2. Label Images with HyperLabel

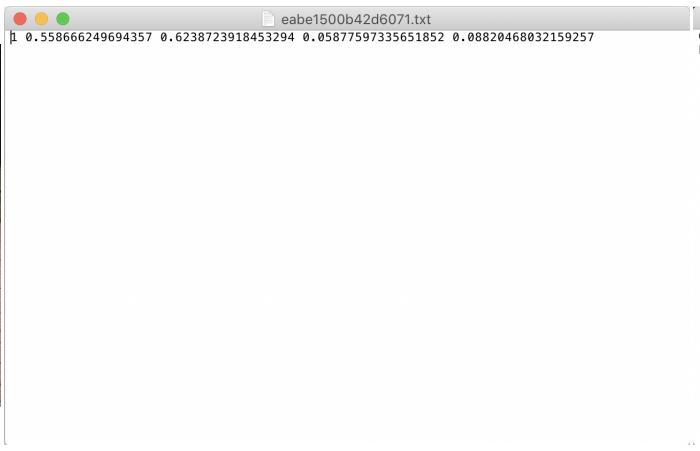
We will use a tool called Hyperlabel to label our images. In the past, I have used many tools to create annotations like labelimg, labelbox, etc. but never came across a tool so straightforward and that too open source. The only downside is that you cannot get this tool for Linux and only for Mac and Windows, but I guess that is fine for most of us.

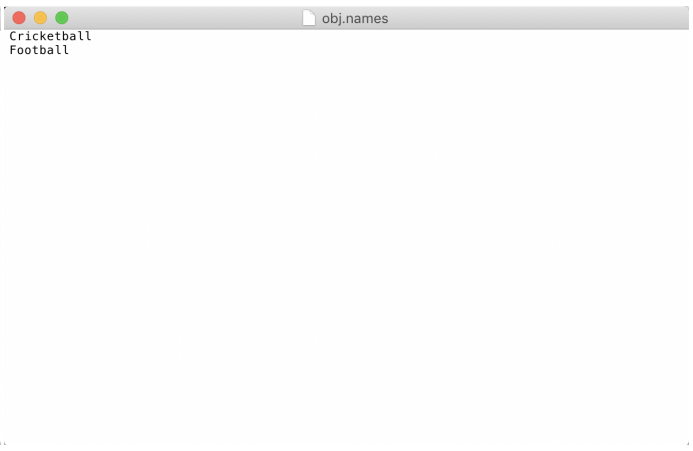


The best part of this tool is the variety of output formats it provides. Since we want to get the data for Yolo, we will close Yolo Format and export it after being done with our annotations. But you can choose to use this tool if you want to get annotations in JSON format(COCO) or XML format(Pascal VOC) too.



Exporting in Yolo format essentially creates a .txt file for each of our images, which contains the class_id, x_center, y_center, width, and the height of the image. It also creates a file named obj.names , which helps map the class_id to the class name. For example:





Notice that the coordinates are scaled from 0 to 1 in the annotation file. Also, note that the class_id is 0 for Cricketball and 1 for football as per obj.names file, which starts from 0. There are a few other files we create using this, but we won't be using them in this example.

Once we have done this, we are mostly set up with our custom dataset and would only need to rearrange some of these files for subsequent training and validation splits later when we train our model. The dataset currently will be a single folder like below containing both the images as well as annotations:

```
dataset
    - 0027773a6d54b960.jpg
    - 0027773a6d54b960.txt
    - 2bded1f9cb587843.jpg
    - 2bded1f9cb587843.txt
    --
    --
```

Setting up the project

To train our custom object detector, we will be using Yolov5 from Ultralytics. We start by cloning the repository and installing the dependencies:

```
# clone repo
git clone [https://github.com/ultralytics/yolov5](https://github.com/ultralytics/yolov5)
cd yolov5
pip install -U -r requirements.txt
```

We then start with creating our own folder named training in which we will keep our custom dataset.

```
!mkdir training
```

We start by copying our custom dataset folder in this folder and creating the train validation folders using the simple train_val_folder_split.ipynb notebook. This code below just creates some train and validation folders and populates them with images.

```
import glob, os
import random

# put your own path here
dataset_path = 'dataset'

# Percentage of images to be used for the validation set
percentage_test = 20

!mkdir data
!mkdir data/images
!mkdir data/labels
!mkdir data/images/train
!mkdir data/images/valid
!mkdir data/labels/train
!mkdir data/labels/valid

# Populate the folders
p = percentage_test/100
for pathAndFilename in glob.iglob(os.path.join(dataset_path, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))
    if random.random() <=p :
        os.system(f"cp {dataset_path}/{title}.jpg data/images/valid")
        os.system(f"cp {dataset_path}/{title}.txt data/labels/valid")
    else:
        os.system(f"cp {dataset_path}/{title}.jpg data/images/train")
        os.system(f"cp {dataset_path}/{title}.txt data/labels/train")
```

After running this, your data folder structure should look like below. It should have two directories images and labels.

We now have to add two configuration files to training folder:

1. Dataset.yaml: We create a file “dataset.yaml” that contains the path of training and validation images and also the classes.

```
# train and val datasets (image directory or *.txt file with image paths)
train: training/data/images/train/
val: training/data/images/valid/

# number of classes
nc: 2

# class names
names: ['Cricketball', 'Footbal']
```

2. Model.yaml: We can use multiple models ranging from small to large while creating our network. For example, yolov5s.yaml file in the yolov5/models directory is the small Yolo model with 7M parameters, while the yolov5x.yaml is the largest Yolo model with 96M Params. For this project, I will use the yolov5l.yaml which has 50M params. We start by copying the file from yolov5/models/yolov5l.yaml to the training folder and changing nc , which is the number of classes to 2 as per our project requirements.

```
# parameters
nc: 2  # change number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple
```

Train

At this point our training folder looks like:

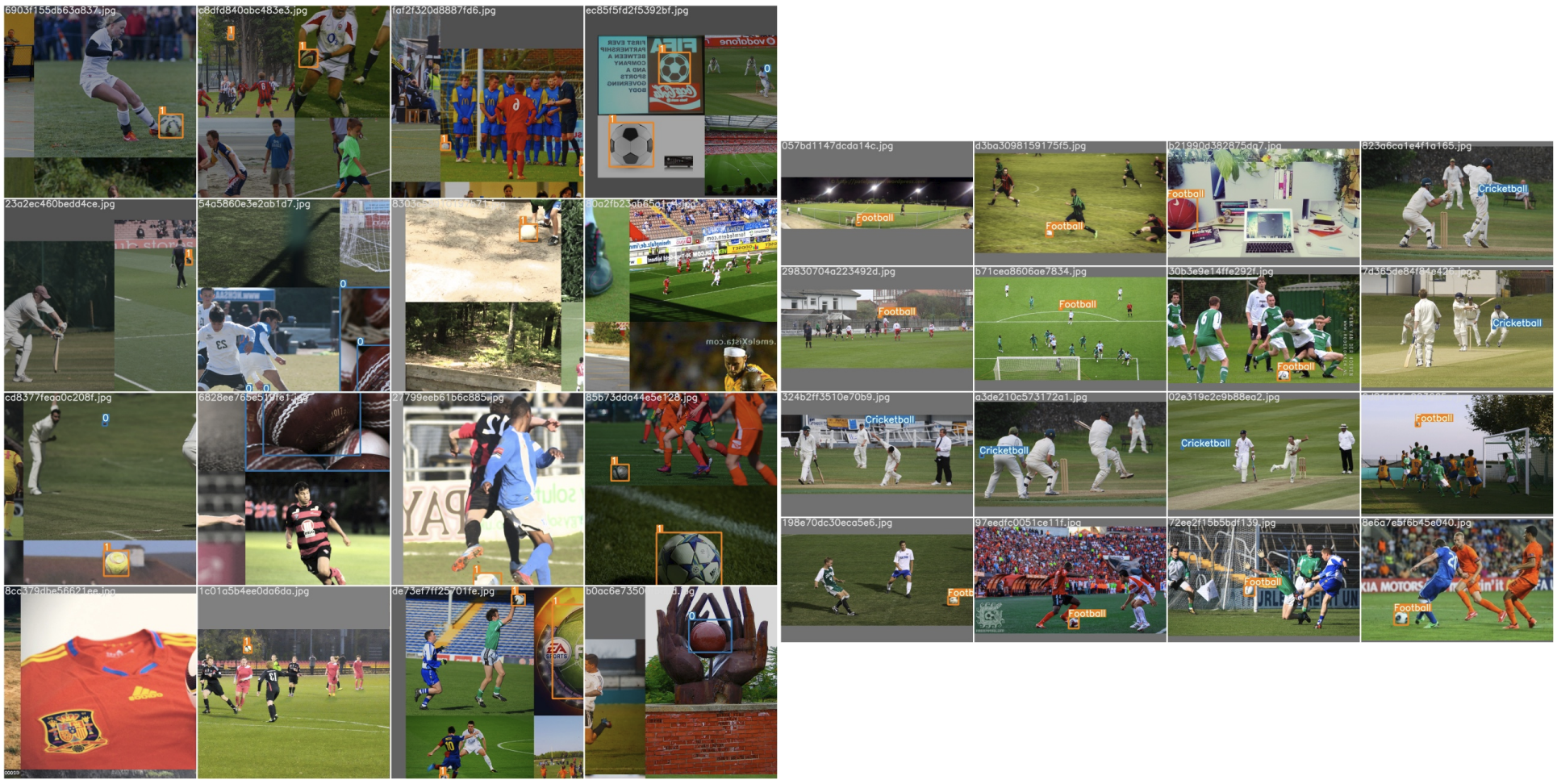
Once we are done with the above steps, we can start training our model. This is as simple as running the below command, where we provide the locations of our config files and various other params. You can check out the different other options in train.py file, but these are the ones I found noteworthy.

```
# Train yolov5l on custom dataset for 300 epochs
$ python train.py --img 640 --batch 16 --epochs 300--data training/dataset.yaml --cfg training/yolov5l.yaml --
weights ''
```

Sometimes you might get an error with PyTorch version 1.5 in that case run on a single GPU using:

```
# Train yolov5l on custom dataset for 300 epochs
$ python train.py --img 640 --batch 16 --epochs 300--data training/dataset.yaml --cfg training/yolov5l.yaml --
weights '' --device 0
```

Once you start the training, you can check whether the training has been set up by checking the automatically created filetrain_batch0.jpg , which contains the training labels for the first batch and test_batch0_gt.jpg which includes the ground truth for test images. This is how they look for me.



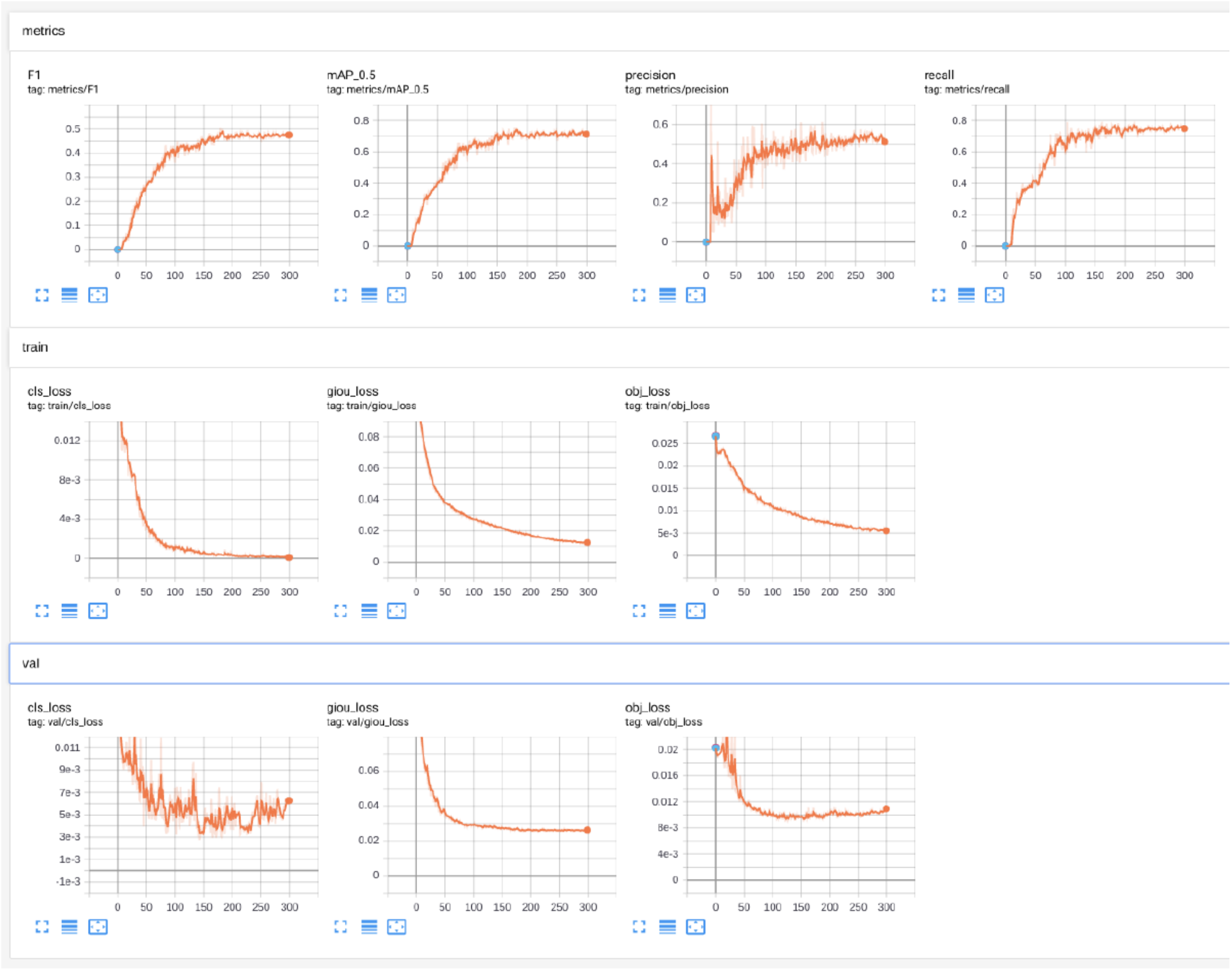
Left: train_batch0.jpg, Right: test_batch0_gt.jpg

Results

To see the results for the training at localhost:6006 in your browser using tensorboard, run this command in another terminal tab

```
tensorboard --logdir=runs
```

Here are the various validation metrics. These metrics also get saved in a file results.png at the end of the training run.



Predict

Ultralytics Yolov5 provides a lot of different ways to check the results on new data.

To detect some images you can simply put them in the folder named inference/images and run the inference using the best weights as per validation AP:

```
python detect.py --weights weights/best.pt
```

You can also detect in a video using the detect.py file:

```
python detect.py --weights weights/best.pt --source inference/videos/messi.mp4 --view-img --output
inference/output
```

Here I specify that I want to see the output using the — view-img flag, and we store the output at the location inference/output. This will create a .mp4 file in this location. It's impressive that the network can see the ball, the speed at which inference is made here, and also the mindblowing accuracy on never observed data.

You can also use the webcam as a source by specifying the –source as 0. You can check out the various other options in detect.py file.

Conclusion

In this post, I talked about how to create a Yolov5 object detection model using a Custom Dataset. I love the way Ultralytics has made it so easy to create an object detection model.

Additionally, the various ways that they have provided to see the model results make it a complete package I have seen in a long time.

If you would like to experiment with the custom dataset yourself, you can download the annotated data on Kaggle and the code at Github .

If you want to know more about various ***Object Detection techniques, motion estimation, object tracking in video, etc***., I would like to recommend this excellent course on Deep Learning in Computer Vision in the Advanced machine learning specialization . If you wish to know more about how the object detection field has evolved over the years, you can also take a look at my last post on Object detection.

Thanks for the read. I am going to be writing more beginner-friendly posts in the future too. Follow me up at Medium or Subscribe to my blog

This story was first published here

About Me

I’m a data scientist consultant and big data engineer based in Bangalore, where I am currently working with WalmartLabs .

Know More

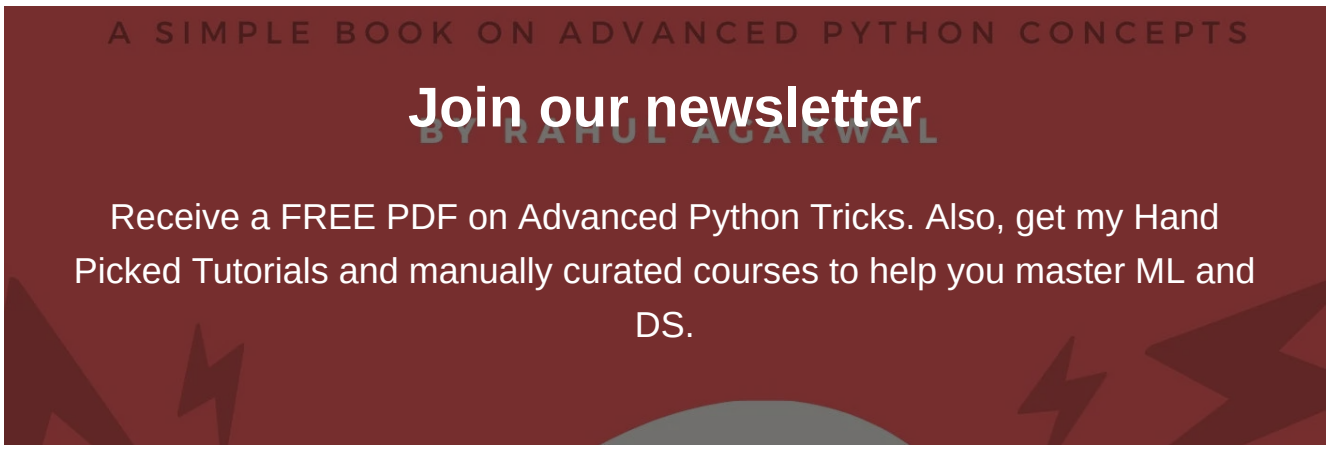
Topics

Awesome Guides
Big Data
Computer Vision
Data Science
Deep Learning
Learning Resources
Natural Language Processing
Programming

Tags

Algorithms Artificial Intelligence Dask Deployment Ec2 Generative Adversarial Networks Graphs Image Classification Instance Segmentation Interpretability Jobs Kaggle Language Modeling Machine Learning Math Multiprocessing Object Detection Opinion Pandas Production Productivity Python Pytorch Spark Sql Statistics Streamlit Text Classification Timeseries Tools Transformers Translation Visualization Xgboost

Connect With Me

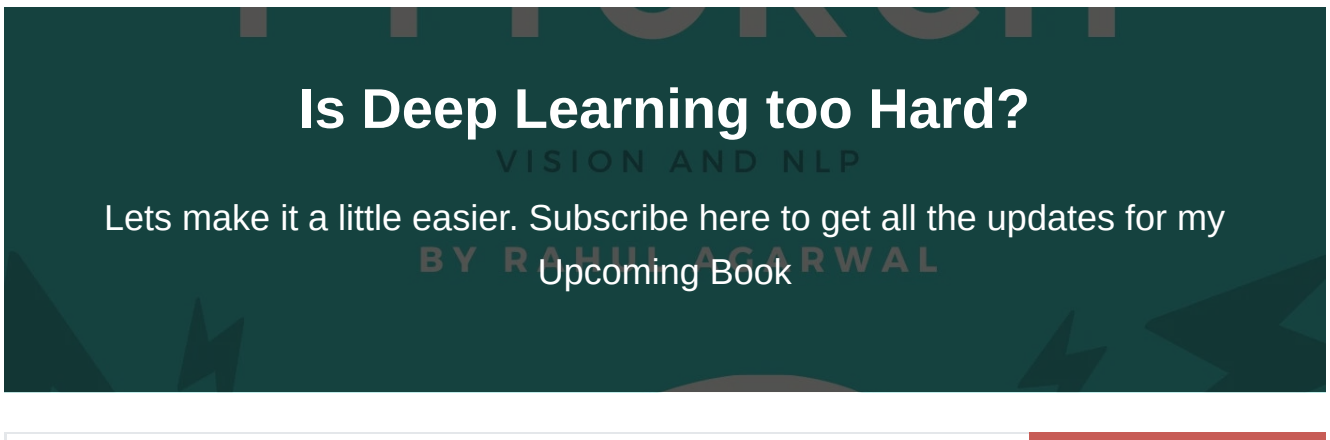


Join our newsletter

Receive a FREE PDF on Advanced Python Tricks. Also, get my Hand Picked Tutorials and manually curated courses to help you master ML and DS.

Your email address

Send it my way!



Is Deep Learning too Hard?

Lets make it a little easier. Subscribe here to get all the updates for my Upcoming Book

Your email address

Let Me Know!

MLWhiz

Contact Me

India, Bangalore

rahul@mlwhiz.com

Social Contacts

Linkedin

Medium

Twitter

Facebook

Github

Categories

Awesome Guides

Quick Links

About

Post

Big Data

Computer Vision

Data Science

Deep Learning

Learning Resources

Natural Language Processing

Programming