ultralytics / yolov5

Code Issues 251 Pull requests 8 Actions Wiki Security Insi

Train Custom Data

Jump to bottom

Glenn Jocher edited this page 2 days ago · 22 revisions

This guide explains how to train your own **custom dataset** with YOLOv5.

Before You Start

Clone this repo, download tutorial dataset, and install requirements.txt dependencies, including **Python>=3.8** and **PyTorch>=1.6**.

```
git clone https://github.com/ultralytics/yolov5  # clone repo
curl -L -o tmp.zip https://github.com/ultralytics/yolov5/releases/download/v1.0/coco128.zip &
cd yolov5
pip install -qr requirements.txt  # install dependencies
```

Train On Custom Data

1. Create Dataset.yaml

data/coco128.yaml is a small tutorial dataset composed of the first 128 images in COCO train2017. These same 128 images are used for both training and validation in this example. `coco128.yaml` defines 1) a path to a directory of training images (or path to a *.txt file with a list of training images), 2) the same for our validation images, 3) the number of classes, 4) a list of class names:

```
# download command/URL (optional)
download: https://github.com/ultralytics/yolov5/releases/download/v1.0/coco128.zip

# train and val data as 1) directory: path/images/, 2) file: path/images.txt, or 3) list: [pa
train: ../coco128/images/train2017/
val: ../coco128/images/train2017/

# number of classes
```

```
nc: 80

# class names
names: ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat'
        'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse',
        'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'su
        'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skatet
        'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'ba
        'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'cha
        'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', '
        'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock'
        'teddy bear', 'hair drier', 'toothbrush']
```

2. Create Labels

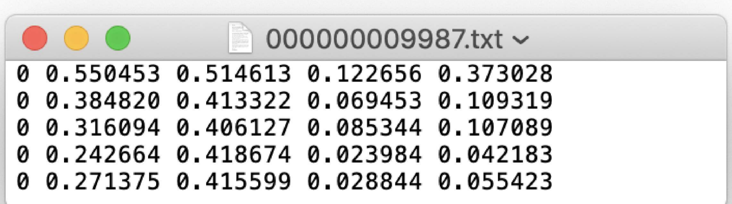
After using a tool like Labelbox, CVAT or makesense.ai to label your images, export your labels to **YOLO format**, with one `*.txt` file per image (if no objects in image, no `*.txt` file is required). The `*.txt` file specifications are:

- One row per object
- Each row is `class x_center y_center width height` format.
- Box coordinates must be in **normalized xywh** format (from 0 - 1). If your boxes are in pixels, divide `x_center` and `width` by image width, and `y_center` and `height` by image height.
- Class numbers are zero-indexed (start from 0).

Each image's label file should be locatable by simply replacing `/images/*.jpg` with `/labels/*.txt` in its pathname. An example image and label pair would be:

```
dataset/images/train2017/000000109622.jpg  # image
dataset/labels/train2017/000000109622.txt  # label
```

An example label file with 5 persons (all class `0`):

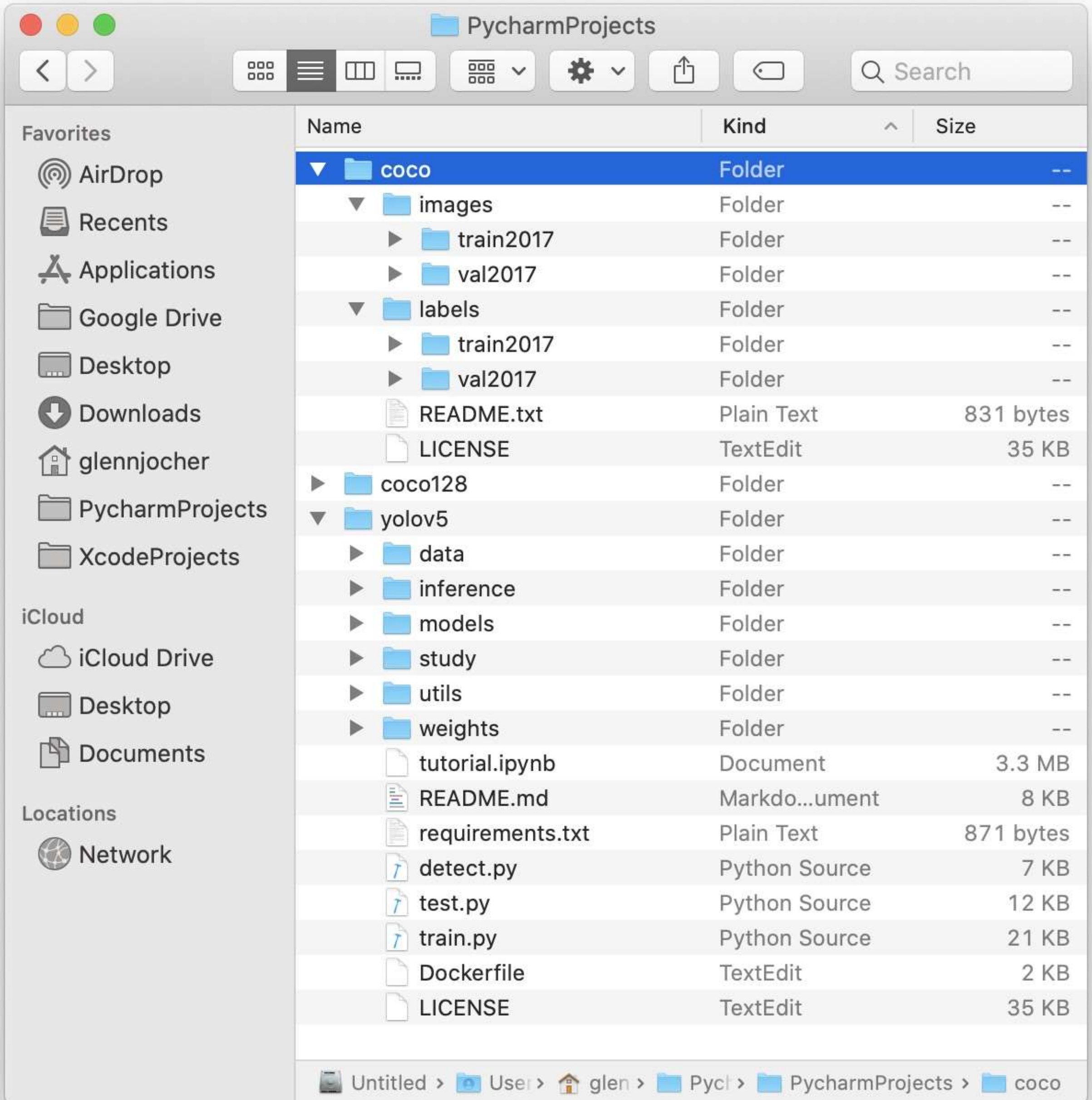


000000009987.txt

```
0 0.550453 0.514613 0.122656 0.373028
0 0.384820 0.413322 0.069453 0.109319
0 0.316094 0.406127 0.085344 0.107089
0 0.242664 0.418674 0.023984 0.042183
0 0.271375 0.415599 0.028844 0.055423
```

3. Organize Directories

Organize your train and val images and labels according to the example below. Note `/coco128` should be **next to** the `/yolov5` directory. Make sure `coco128/labels` folder is next to `coco128/images` folder.



4. Select a Model

Select a model from the `./models` folder. Here we select yolov5s.yaml, the smallest and fastest model available. See our README table for a full comparison of all models. Once you have selected a model, if you are not training COCO, **update** the `nc: 80` parameter in your yaml file to match the number of classes in your dataset from step **1**.

```
# parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
```

```
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, BottleneckCSP, [512, False]],  # 20  (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]],  # 23  (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

5. Train

Train a YOLOv5s model on coco128 by specifying model config file `--cfg models/yolo5s.yaml`, and dataset config file `--data data/coco128.yaml`. Start training from pretrained `--weights yolov5s.pt`, or from randomly initialized `--weights ''`. Pretrained weights are auto-downloaded from Google Drive.

All training results are saved to `runs/exp0` for the first experiment, then `runs/exp1`, `runs/exp2` etc. for subsequent experiments.

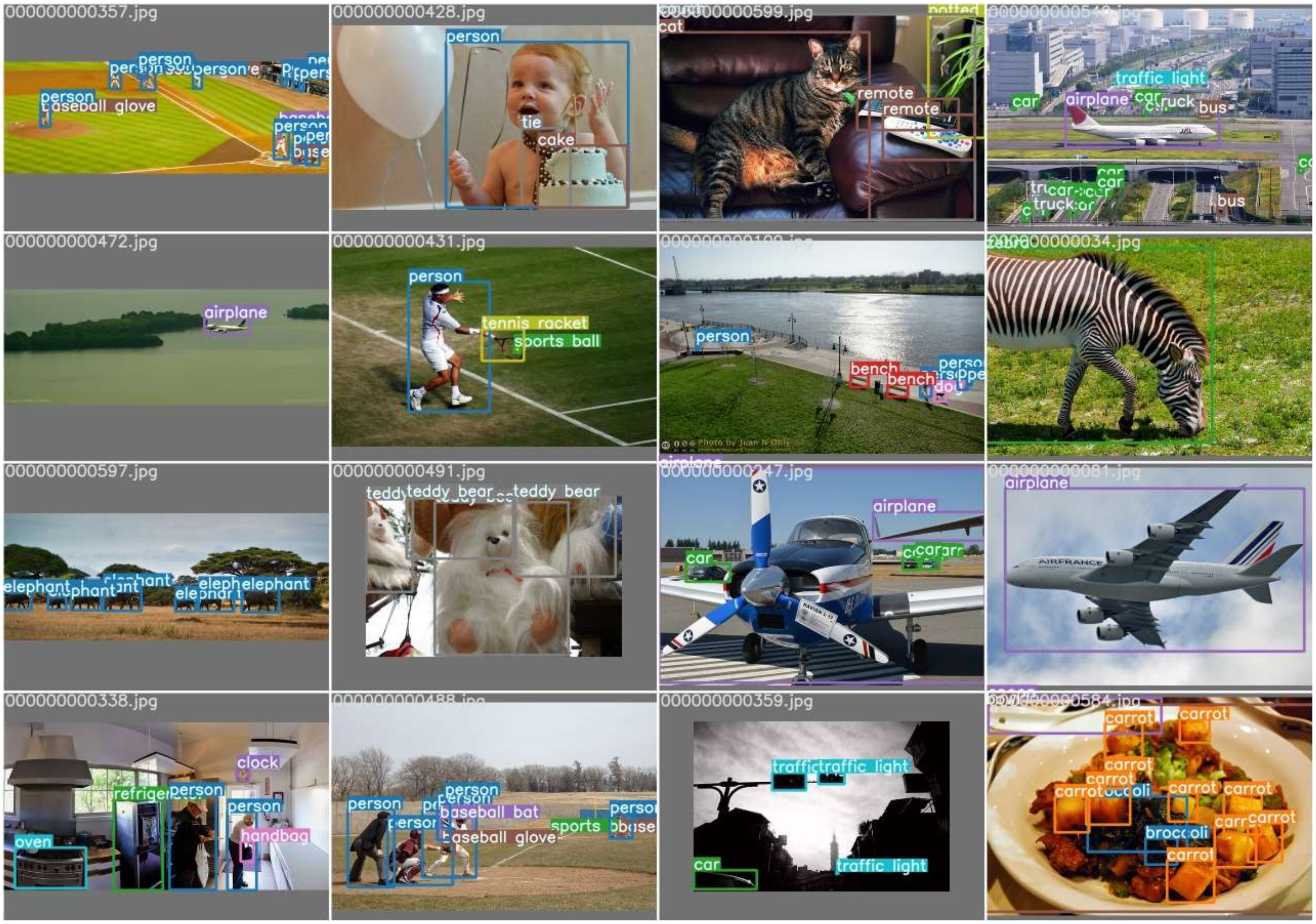
```
# Train YOLOv5s on coco128 for 5 epochs
$ python train.py --img 640 --batch 16 --epochs 5 --data ./data/coco128.yaml --cfg ./models/y
```

For training command outputs and further details please see the training section of Google Colab Notebook. Open in Colab

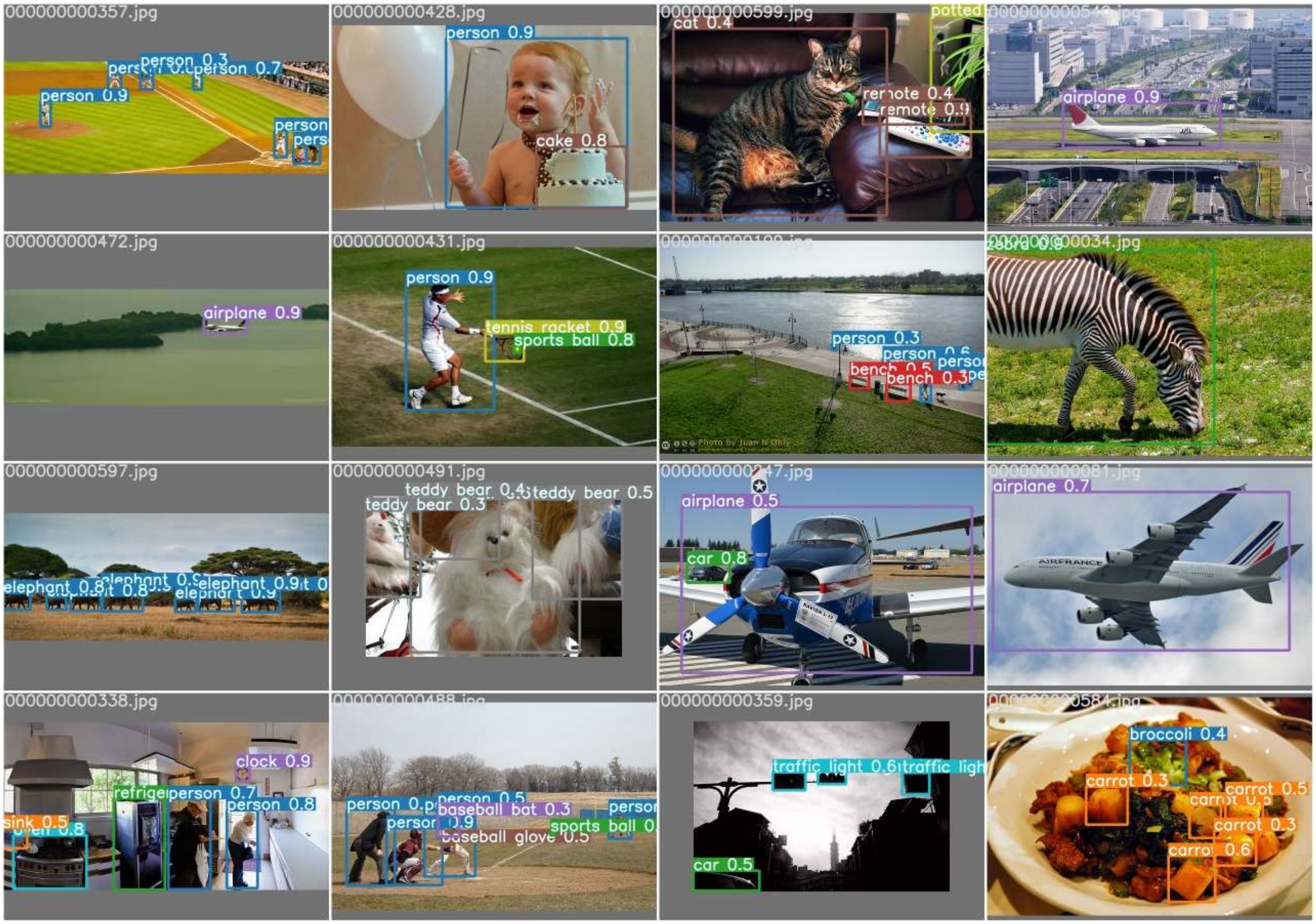
6. Visualize

View `runs/exp0/train*.jpg` images to see training images, labels and augmentation effects. A **Mosaic Dataloader** is used for training (shown below), a new concept developed by Ultralytics and first featured in YOLOv4. If your labels are not correct in these images then you have incorrectly labelled your data, and should revisit **2. Create Labels**.

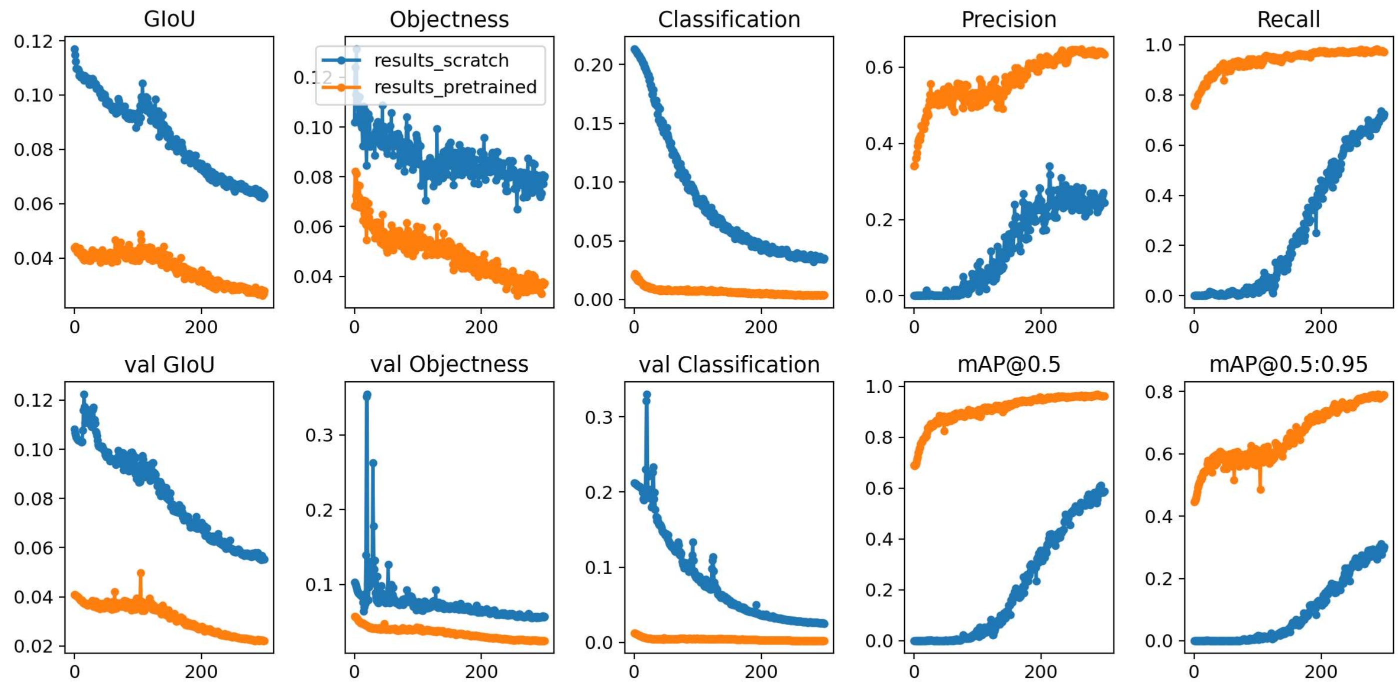
After the first epoch is complete, view `test_batch0_gt.jpg` to see test batch 0 `ground truth` labels:



And view `test_batch0_pred.jpg` to see test batch 0 *predictions*:



Training losses and performance metrics are saved to Tensorboard and also to a `runs/exp0/results.txt` logfile. `results.txt` is plotted as `results.png` after training completes. Partially completed `results.txt` files can be plotted with `from utils.utils import plot_results; plot_results()`. Here we show YOLOv5s trained on coco128 to 300 epochs, starting from scratch (blue), and from pretrained `yolov5s.pt` (orange).



Environments

YOLOv5 may be run in any of the following up-to-date verified environments (with all dependencies including CUDA/CUDNN, Python and PyTorch preinstalled):

- **Google Colab Notebook** with free GPU: Open in Colab
- **Kaggle Notebook** with free GPU: https://www.kaggle.com/ultralytics/yolov5
- **Google Cloud** Deep Learning VM. See GCP Quickstart Guide
- **Docker Image** https://hub.docker.com/r/ultralytics/yolov5. See Docker Quickstart Guide
  docker pulls 3.1k

Ultralytics LLC 2020
www.ultralytics.com

Manufacturing Services Agreement
Privacy Policy

▸ Pages 4


OLOv5
只看一次v5
keyboard 98.5
car 99.5
car 83.0
aptop 99.6
6 items

Clone this wiki locally

```
https://github.com/ultralytics/yolov5.wiki.git
```