```
!git clone https://github.com/ultralytics/yolov5  # clone
%cd yolov5
%pip install -qr requirements.txt  # install

import torch
from yolov5 import utils
display = utils.notebook_init()  # checks
```

```
YOLOv5 🚀 v6.1-11-g63ddb6f torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 42.1/78.2 GB disk)
```

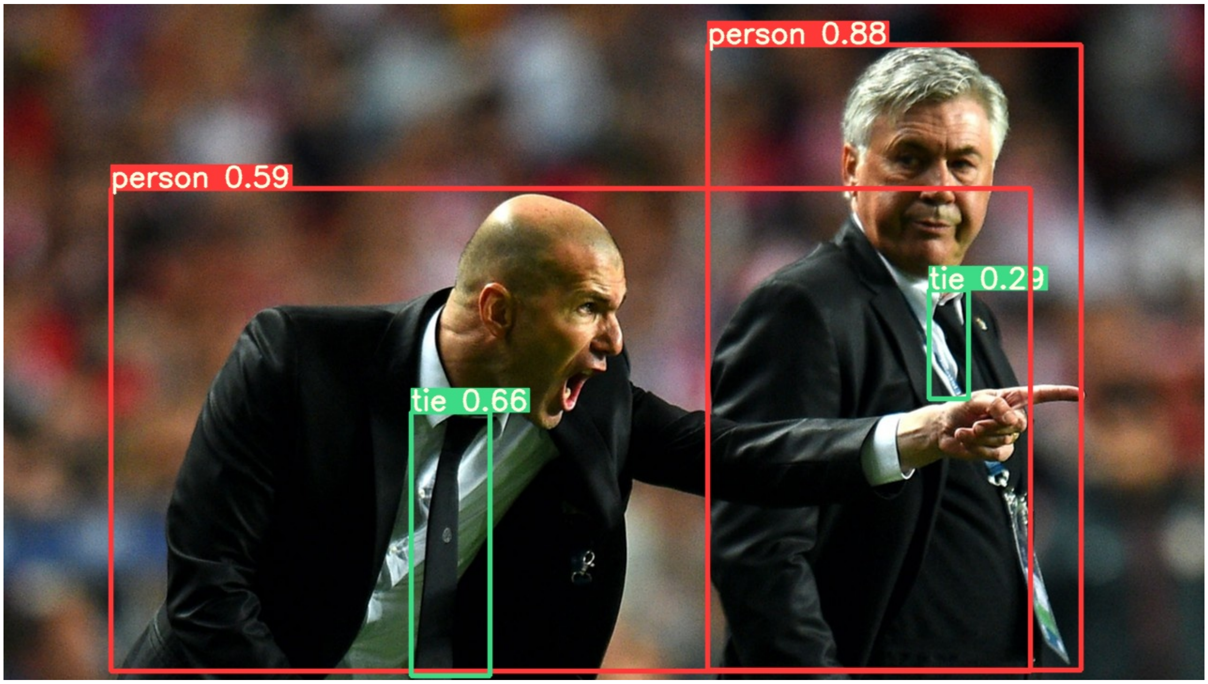
```
!unzip -q /content/Task_2.zip -d ../
```

1. Inference

`detect.py` runs YOLOv5 inference on a variety of sources, downloading models automatically from the [latest YOLOv5 release](https://github.com/ultralytics/yolov5/releases), and saving results to `runs/detect`. Example inference sources are:

```
python detect.py --source 0  # webcam
                          img.jpg  # image
                          vid.mp4  # video
                          path/  # directory
                          path/*.jpg  # glob
                          'https://youtu.be/Zgi9g1ksQHc'  # YouTube
                          'rtsp://example.com/media.mp4'  # RTSP, RTMP, HTTP stream
```

```
!python detect.py --weights runs/train/exp/weights/best.pt --img 640 --conf 0.7 --source /con
#display.Image(filename='runs/detect/exp/zidane.jpg', width=600)
```



2. Validate

Validate a model's accuracy on COCO val or test-dev datasets. Models are downloaded automatically from the latest YOLOv5 release. To show results by class use the `--verbose` flag. Note that `pycocotools` metrics may be ~1% better than the equivalent repo metrics, as is visible below, due to slight differences in mAP computation.

COCO val

Download COCO val 2017 dataset (1GB - 5000 images), and test model accuracy.

```
# Download COCO val
torch.hub.download_url_to_file('https://ultralytics.com/assets/coco2017val.zip', 'tmp.zip')
!unzip -q tmp.zip -d ../datasets && rm tmp.zip
```

```
# Run YOLOv5x on COCO val
!python val.py --weights yolov5x.pt --data coco.yaml --img 640 --iou 0.65 --half
```

```
val: data=/content/yolov5/data/coco.yaml, weights=['yolov5x.pt'], batch_size=32, imgsz=6
YOLOv5 🚀 v6.0-48-g84a8099 torch 1.10.0+cu102 CUDA:0 (Tesla V100-SXM2-16GB, 16160MiB)

Downloading https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5x.pt to y
100% 166M/166M [00:03<00:00, 54.1MB/s]
```

```
Fusing layers...
Model Summary: 444 layers, 86705005 parameters, 0 gradients
val: Scanning '../datasets/coco/val2017' images and labels...4952 found, 48 missing, 0 e
val: New cache created: ../datasets/coco/val2017.cache
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
                 all       5000      36335      0.729       0.63      0.683      0.496
Speed: 0.1ms pre-process, 4.9ms inference, 1.9ms NMS per image at shape (32, 3, 640, 640

Evaluating pycocotools mAP... saving runs/val/exp/yolov5x_predictions.json...
loading annotations into memory...
Done (t=0.46s)
creating index...
index created!
Loading and preparing results...
DONE (t=5.15s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=90.39s).
Accumulating evaluation results...
DONE (t=14.54s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.507
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.689
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.552
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.345
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.559
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.652
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.381
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.630
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.682
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.526
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.732
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.829
Results saved to runs/val/exp
```

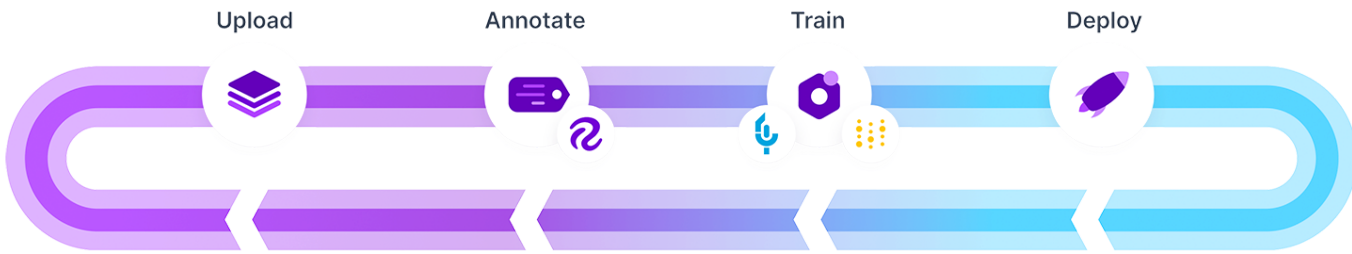
COCO test

Download [COCO test2017](https://github.com/ultralytics/yolov5/blob/master/data/scripts/get_coco.sh) dataset (7GB - 40,000 images), to test model accuracy on test-dev set (**20,000 images, no labels**). Results are saved to a `*.json` file which should be **zipped** and submitted to the evaluation server at https://competitions.codalab.org/competitions/20794.

```
# Download COCO test-dev2017
torch.hub.download_url_to_file('https://ultralytics.com/assets/coco2017labels.zip', 'tmp.zip'
!unzip -q tmp.zip -d ../datasets && rm tmp.zip
!f="test2017.zip" && curl http://images.cocodataset.org/zips/$f -o $f && unzip -q $f -d ../da
```

```
# Run YOLOv5x on COCO test
!python val.py --weights yolov5x.pt --data coco.yaml --img 640 --iou 0.65 --half --task test
```

3. Train



Close the active learning loop by sampling images from your inference conditions with the `roboflow` pip package

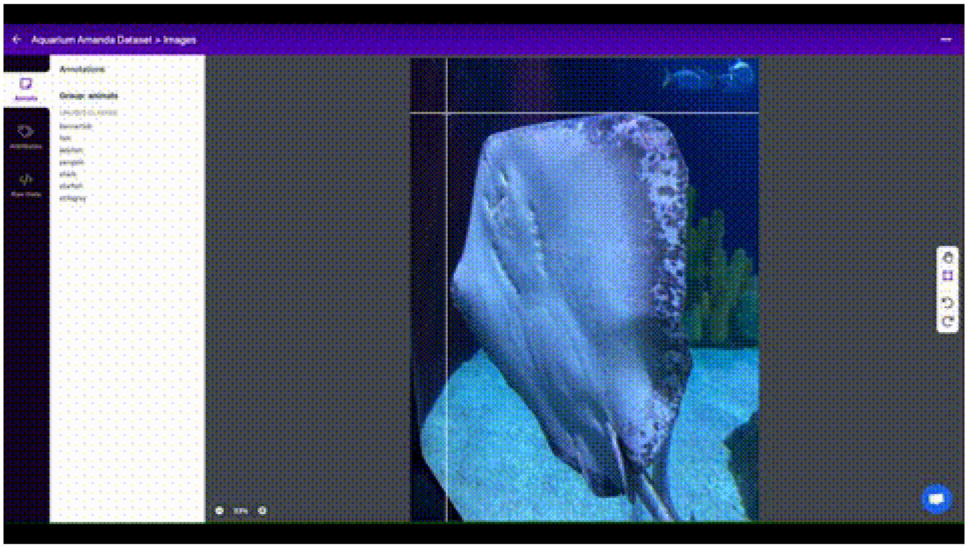
Train a YOLOv5s model on the [COCO128]() dataset with `--data coco128.yaml`, starting from pretrained `--weights yolov5s.pt`, or from randomly initialized `--weights '' --cfg yolov5s.yaml`.

- **Pretrained [Models]()** are downloaded automatically from the [latest YOLOv5 release]()
- **[Datasets]()** available for autodownload include: [COCO](), [COCO128](), [VOC](), [Argoverse](), [VisDrone](), [GlobalWheat](), [xView](), [Objects365](), [SKU-110K]().
- **Training Results** are saved to `runs/train/` with incrementing run directories, i.e. `runs/train/exp2`, `runs/train/exp3` etc.

Train on Custom Data with Roboflow 🌟 NEW

[Roboflow]() enables you to easily **organize, label, and prepare** a high quality dataset with your own custom data. Roboflow also makes it easy to establish an active learning pipeline, collaborate with your team on dataset improvement, and integrate directly into your model building workflow with the `roboflow` pip package.

- Custom Training Example: https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/
- Custom Training Notebook: Open in Colab



Label images lightning fast (including with model-assisted labeling)

```
# Tensorboard  (optional)
%load_ext tensorboard
%tensorboard --logdir runs/train
```

```
# Weights & Biases  (optional)
%pip install -q wandb
import wandb
wandb.login()
```

```
# Train YOLOv5s on COCO128 for 200 epochs
!python train.py --img 640 --batch 5 --epochs 200 --data custom_dataset.yaml.txt --weights yo
```

```
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Ari
train: weights=yolov5s.pt, cfg=, data=custom_dataset.yaml.txt, hyp=data/hyps/hyp.scra
github: up to date with https://github.com/ultralytics/yolov5 ✅
YOLOv5 🚀 v6.1-11-g63ddb6f torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epoc
Weights & Biases: run 'pip install wandb' to automatically track and visualize YOLOv5
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6
Downloading https://github.com/ultralytics/yolov5/releases/download/v6.1/yolov5s.pt t
100% 14.1M/14.1M [00:00<00:00, 111MB/s]

Overriding model.yaml nc=80 with nc=7

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6
  1                -1  1     18560  models.common.Conv                      [32, 64,
  2                -1  1     18816  models.common.C3                        [64, 64,
  3                -1  1     73984  models.common.Conv                      [64, 128,
  4                -1  2    115712  models.common.C3                        [128, 128
  5                -1  1    295424  models.common.Conv                      [128, 256
  6                -1  3    625152  models.common.C3                        [256, 256
```

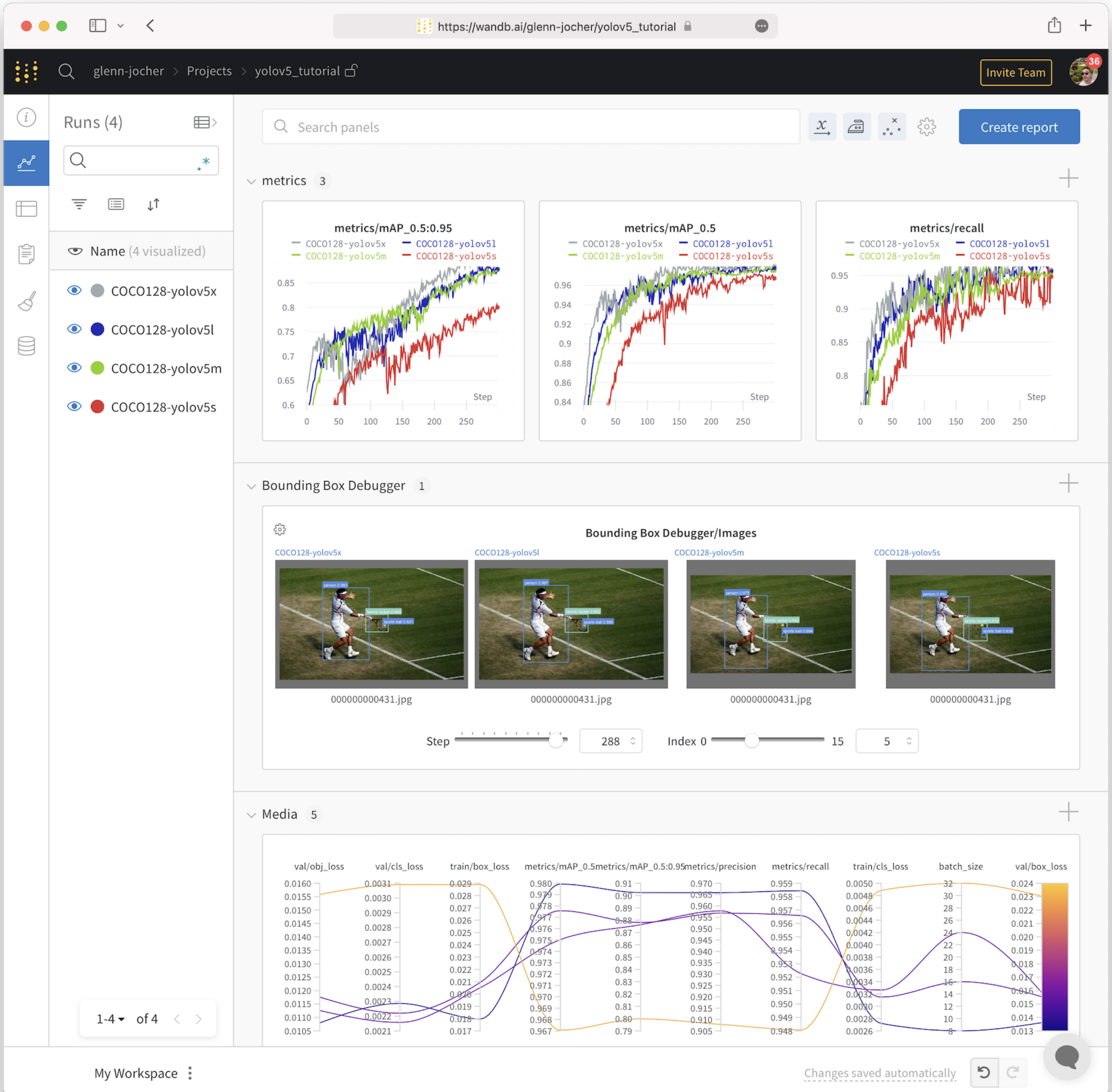
```
  7                -1  1   1180672  models.common.Conv                      [256, 512
  8                -1  1   1182720  models.common.C3                        [512, 512
  9                -1  1    656896  models.common.SPPF                      [512, 512
 10                -1  1    131584  models.common.Conv                      [512, 256
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2,
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256
 14                -1  1     33024  models.common.Conv                      [256, 128
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2,
 16           [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     90880  models.common.C3                        [256, 128
 18                -1  1    147712  models.common.Conv                      [128, 128
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  1    296448  models.common.C3                        [256, 256
 21                -1  1    590336  models.common.Conv                      [256, 256
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  1   1182720  models.common.C3                        [512, 512
 24      [17, 20, 23]  1     32364  models.yolo.Detect                      [7, [[10,
Model Summary: 270 layers, 7038508 parameters, 7038508 gradients, 15.9 GFLOPs

Transferred 343/349 items from yolov5s.pt
Scaled weight_decay = 0.0005078125
optimizer: SGD with parameter groups 57 weight (no decay), 60 weight, 60 bias
albumentations: version 1.0.3 required by YOLOv5, but version 0.1.12 is currently ins
train: Scanning '/content/Task_2/labels/Train' images and labels...34 found, 1 missin
train: New cache created: /content/Task_2/labels/Train.cache
train: Caching images (0.0GB ram): 100% 35/35 [00:00<00:00, 439.21it/s]
val: Scanning '/content/Task_2/labels/Val' images and labels...21 found, 0 missing, 0
val: New cache created: /content/Task_2/labels/Val.cache
val: Caching images (0.0GB ram): 100% 21/21 [00:00<00:00, 191.74it/s]
Plotting labels to runs/train/exp/labels.jpg...

AutoAnchor: 1.68 anchors/target, 0.971 Best Possible Recall (BPR). Anchors are a poor
AutoAnchor: Running kmeans for 9 anchors on 34 points...
AutoAnchor: Evolving anchors with Genetic Algorithm: fitness = 0.9455: 100% 1000/1000
```

4. Visualize

Weights & Biases Logging 🌟 NEW

Weights & Biases (W&B) is now integrated with YOLOv5 for real-time visualization and cloud logging of training runs. This allows for better run comparison and introspection, as well improved visibility and collaboration for teams. To enable W&B `pip install wandb`, and then train normally (you will be guided through setup on first use).

During training you will see live updates at https://wandb.ai/home, and you can create and share detailed Reports of your results. For more information see the YOLOv5 Weights & Biases Tutorial.



Local Logging

All results are logged by default to `runs/train`, with a new experiment directory created for each new training as `runs/train/exp2`, `runs/train/exp3`, etc. View train and val jpgs to see mosaics, labels, predictions and augmentation effects. Note an Ultralytics **Mosaic Dataloader** is used for training (shown below), which combines 4 images into 1 mosaic during training.

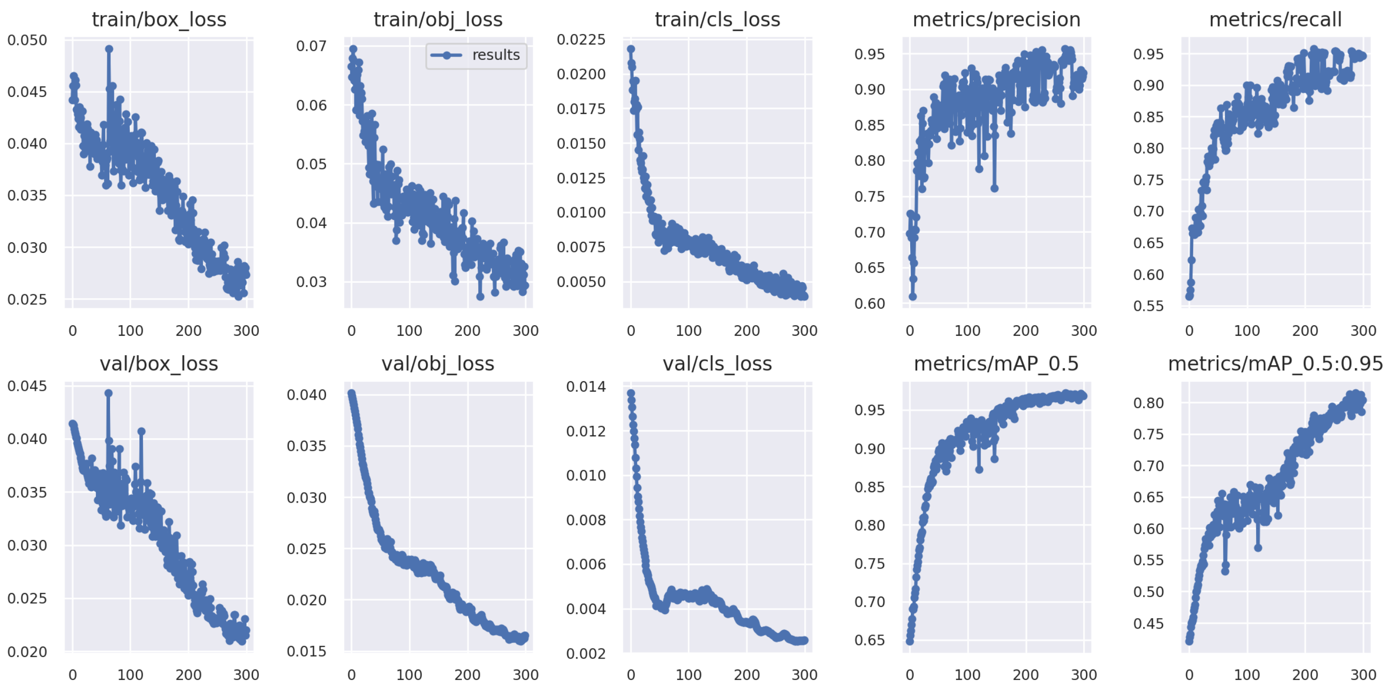
`train_batch0.jpg` shows train batch 0 mosaics and labels

`test_batch0_labels.jpg` shows val batch 0 labels

`test_batch0_pred.jpg` shows val batch 0 *predictions*

Training results are automatically logged to [Tensorboard] and [CSV] as `results.csv`, which is plotted as `results.png` (below) after training completes. You can also plot any `results.csv` file manually:

```
from utils.plots import plot_results
plot_results('path/to/results.csv')  # plot 'results.csv' as 'results.png'
```



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 and Kaggle** notebooks with free GPU: Open in Colab Open in Kaggle
- **Google Cloud** Deep Learning VM. See GCP Quickstart Guide
- **Amazon** Deep Learning AMI. See AWS Quickstart Guide
- **Docker Image**. See Docker Quickstart Guide docker pulls 205k

Status

CI CPU testing passing

If this badge is green, all YOLOv5 GitHub Actions Continuous Integration (CI) tests are currently passing. CI tests verify correct operation of YOLOv5 training (train.py), testing (val.py), inference (detect.py) and export (export.py) on MacOS, Windows, and Ubuntu every 24 hours and on every commit.

Appendix

Optional extras below. Unit tests validate repo functionality and should be run on any PRs submitted.

```
# Reproduce
for x in 'yolov5n', 'yolov5s', 'yolov5m', 'yolov5l', 'yolov5x':
  !python val.py --weights {x}.pt --data coco.yaml --img 640 --task speed  # speed
  !python val.py --weights {x}.pt --data coco.yaml --img 640 --conf 0.001 --iou 0.65  # mAP
```

```
# PyTorch Hub
import torch

# Model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

# Images
dir = 'https://ultralytics.com/images/'
imgs = [dir + f for f in ('zidane.jpg', 'bus.jpg')]  # batch of images

# Inference
results = model(imgs)
results.print()  # or .show(), .save()
```

```
# CI Checks
%%shell
export PYTHONPATH="$PWD"  # to run *.py. files in subdirectories
rm -rf runs  # remove runs/
for m in yolov5n; do  # models
  python train.py --img 64 --batch 32 --weights $m.pt --epochs 1 --device 0  # train pretrain
  python train.py --img 64 --batch 32 --weights '' --cfg $m.yaml --epochs 1 --device 0  # tra
  for d in 0 cpu; do  # devices
    python val.py --weights $m.pt --device $d # val official
    python val.py --weights runs/train/exp/weights/best.pt --device $d # val custom
    python detect.py --weights $m.pt --device $d  # detect official
    python detect.py --weights runs/train/exp/weights/best.pt --device $d  # detect custom
  done
  python hubconf.py  # hub
  python models/yolo.py --cfg $m.yaml  # build PyTorch model
  python models/tf.py --weights $m.pt  # build TensorFlow model
  python export.py --img 64 --batch 1 --weights $m.pt --include torchscript onnx  # export
done
```

```
# Profile
from utils.torch_utils import profile
```

```
m1 = lambda x: x * torch.sigmoid(x)
m2 = torch.nn.SiLU()
results = profile(input=torch.randn(16, 3, 640, 640), ops=[m1, m2], n=100)


# Evolve
!python train.py --img 640 --batch 64 --epochs 100 --data coco128.yaml --weights yolov5s.pt -
!d=runs/train/evolve && cp evolve.* $d && zip -r evolve.zip $d && gsutil mv evolve.zip gs://b


# VOC
for b, m in zip([64, 64, 32, 16], ['yolov5s', 'yolov5m', 'yolov5l', 'yolov5x']):  # zip(batch
  !python train.py --batch {b} --weights {m}.pt --data VOC.yaml --epochs 50 --cache --img 512


# TensorRT
# https://docs.nvidia.com/deeplearning/tensorrt/install-guide/index.html#installing-pip
!pip install -U nvidia-tensorrt --index-url https://pypi.ngc.nvidia.com  # install
!python export.py --weights yolov5s.pt --include engine --imgsz 640 640 --device 0  # export
!python detect.py --weights yolov5s.engine --imgsz 640 640 --device 0  # inference
```