
Open In Colab
(https://colab.research.google.com/github/ultralytics/yolov5/blob/master/tutorial.ipynb)

Setup

Clone repo, install dependencies, %cd into ./yolov5 folder and check GPU.

In [ ]:
```
!git clone https://github.com/ultralytics/yolov5  # clone repo
!pip install -qr yolov5/requirements.txt  # install dependencies (ignore errors)
%cd yolov5

import torch
from IPython.display import Image, clear_output  # to display images
from utils.google_utils import gdrive_download  # to download models/datasets

clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_ava
ilable() else 'CPU'))
```

```
Setup complete. Using torch 1.5.1+cu101 _CudaDeviceProperties(name='Tesla T4', major=7, minor=5, total_memory=15079
MB, multi_processor_count=40)
```

1. Inference

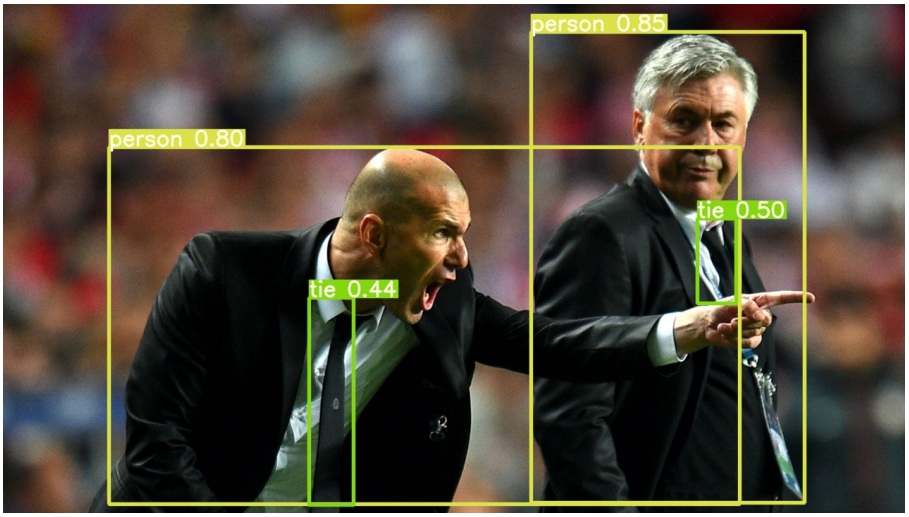
Run inference with a pretrained checkpoint on contents of /inference/images folder. Models are auto-downloaded from Google Drive (https://drive.google.com/open?id=1Drs_Aiu7xx6S-ix95f9kNsA6ueKRpN2J).

In [ ]:
```
!python detect.py --weights yolov5s.pt --img 416 --conf 0.4 --source inference/images/
Image(filename='inference/output/zidane.jpg', width=600)
```

```
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', fourcc='mp4v', half=False, img_si
ze=416, iou_thres=0.5, output='inference/output', save_txt=False, source='./inference/images/', view_img=False, weights='y
olov5s.pt')
Using CUDA device0 _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', total_memory=16280MB)

image 1/2 inference/images/bus.jpg: 416x352 3 persons, 1 buss, Done. (0.009s)
image 2/2 inference/images/zidane.jpg: 288x416 2 persons, 2 ties, Done. (0.009s)
Results saved to /content/yolov5/inference/output
Done. (0.100s)
```

Out[ ]:



Inference can be run on a variety of sources: images, videos, directories, webcams, rtsp and http streams as shown in the example below.

In [ ]:
```
# Example syntax (do not run cell)
!python detect.py --source file.jpg  # image
                  file.mp4  # video
                  dir/  # directory
                  0  # webcam
                  'rtsp://170.93.143.139/rtplive/470011e600ef003a004ee33696235daa' # rtsp
                  'http://112.50.243.8/PLTV/88888888/224/3221225900/1.m3u8'  # http
```

2. Test

Test a model on COCO val or test-dev dataset to determine trained accuracy. Models are auto-downloaded from Google Drive (https://drive.google.com/open?id=1Drs_Aiu7xx6S-ix95f9kNsA6ueKRpN2J). To show results by class use the `--verbose` flag. Note that `pycocotools` metrics may be 1-2% better than the equivalent repo metrics, as is visible below, due to slight differences in mAP computation.

2.1 val2017

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

In [ ]:
```
# Download COCO val2017
gdrive_download('1Y6Kou6kEB0ZEMCCpJSKStCor4KAReE43','coco2017val.zip')  # val2017 dataset
!mv ./coco ../  # move folder alongside /yolov5
```

```
Downloading https://drive.google.com/uc?export=download&id=1Y6Kou6kEB0ZEMCCpJSKStCor4KAReE43 as coco2017val.zip... unzipping... Done (11.2s)
```

In [ ]:
```
# Run YOLOv5x on COCO val2017
!python test.py --weights yolov5x.pt --data coco.yaml --img 672
```

```
Namespace(augment=False, batch_size=32, conf_thres=0.001, data='./data/coco.yaml', device='', img_size=672, iou_thres=0.65, merge=False, save_json=True, single_cls=False, task='val', verbose=False, weights=['yolov5x.pt'])
Using CUDA device0 _CudaDeviceProperties(name='Tesla T4', total_memory=15079MB)

Fusing layers... Model Summary: 284 layers, 8.89222e+07 parameters, 8.89222e+07 gradients
Scanning labels ../coco/labels/val2017.cache (4952 found, 0 missing, 48 empty, 0 duplicate, for 5000 images): 100% 5000/5000 [00:00<00:00, 22899.17it/s]
           Class     Images    Targets          P          R      mAP@.5  mAP@.5:.95: 100% 157/157 [02:38<00:00,  1.01s/it]
             all      5e+03   3.63e+04      0.426      0.746       0.66      0.469
Speed: 22.3/1.7/24.0 ms inference/NMS/total per 672x672 image at batch-size 32

COCO mAP with pycocotools... saving detections_val2017__results.json...
loading annotations into memory...
Done (t=0.41s)
creating index...
index created!
Loading and preparing results...
DONE (t=4.39s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=76.56s).
Accumulating evaluation results...
DONE (t=11.02s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.484
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.668
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.528
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.311
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.534
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.628
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.371
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.609
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.662
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.501
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.714
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.807
```

2.2 test-dev2017

Download COCO test2017 dataset, 7GB, 40,000 images, to test model accuracy on test-dev set, 20,000 images. Results are saved to a `*.json` file which can be submitted to the evaluation server at https://competitions.codalab.org/competitions/20794 (https://competitions.codalab.org/competitions/20794).

In [ ]:
```
# Download COCO test-dev2017
gdrive_download('1cXZR_ckHki6nddOmcysCuuJFM--T-Q6L','coco2017labels.zip')  # annotations
!f="test2017.zip" && curl http://images.cocodataset.org/zips/$f -o $f && unzip -q $f && rm $f  # 7GB,  41k images
!mv ./test2017 ./coco/images && mv ./coco ../  # move images into /coco and move /coco alongside /yolov5
```

In [ ]:
```
# Run YOLOv5s on COCO test-dev2017 with argument --task test
!python test.py --weights yolov5s.pt --data ./data/coco.yaml --task test
```

3. Train

Download https://www.kaggle.com/ultralytics/coco128 (https://www.kaggle.com/ultralytics/coco128), a small 128-image tutorial dataset, start tensorboard and train YOLOv5s from a pretrained checkpoint for 3 epochs (actual training is much longer, around **300-1000 epochs**, depending on your dataset).

In [ ]:
```
# Download coco128
gdrive_download('1n_oKgR81BJtqk75b00eAjdv03qVCQn2f','coco128.zip')  # coco128 dataset
!mv ./coco128 ../  # move folder alongside /yolov5
```

```
Downloading https://drive.google.com/uc?export=download&id=1n_oKgR81BJtqk75b00eAjdv03qVCQn2f as coco128.zip... u
nzipping... Done (5.3s)
```

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 (https://drive.google.com/open?id=1Drs_Aiu7xx6S-ix95f9kNsA6ueKRpN2J).

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

In [ ]:
```
# Start tensorboard (optional)
%load_ext tensorboard
%tensorboard --logdir runs
```

In [ ]:
```
# Train YOLOv5s on coco128 for 3 epochs
!python train.py --img 640 --batch 16 --epochs 3 --data coco128.yaml --cfg yolov5s.yaml --weights yolov5s.pt --nosave --cache
```

```
Namespace(batch_size=16, bucket='', cache_images=True, cfg='./models/yolov5s.yaml', data='./data/coco128.yaml', device='', epochs=3, evolve=False, hyp='', img_size=[640], multi_scale=False, name='', noautoanchor=False, nosave=True, notest=False, rect=False, resume=False, single_cls=False, weights='yolov5s.pt')
Using CUDA device0 _CudaDeviceProperties(name='Tesla T4', total_memory=15079MB)

2020-07-11 20:37:09.422496: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.1
Start Tensorboard with "tensorboard --logdir=runs", view at http://localhost:6006/
Hyperparameters {'optimizer': 'SGD', 'lr0': 0.01, 'momentum': 0.937, 'weight_decay': 0.0005, 'giou': 0.05, 'cls': 0.58, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.014, 'hsv_s': 0.68, 'hsv_v': 0.36, 'degrees': 0.0, 'translate': 0.0, 'scale': 0.5, 'shear': 0.0}

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Focus                     [3, 32, 3]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     19904  models.common.BottleneckCSP             [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  1    161152  models.common.BottleneckCSP             [128, 128, 3]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  1    641792  models.common.BottleneckCSP             [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1    656896  models.common.SPP                       [512, 512, [5, 9, 13]]
  9                -1  1   1248768  models.common.BottleneckCSP             [512, 512, 1, False]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    378624  models.common.BottleneckCSP             [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16           [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     95104  models.common.BottleneckCSP             [256, 128, 1, False]
 18                -1  1     32895  torch.nn.modules.conv.Conv2d            [128, 255, 1, 1]
 19                -2  1    147712  models.common.Conv                      [128, 128, 3, 2]
 20          [-1, 14]  1         0  models.common.Concat                    [1]
 21                -1  1    313088  models.common.BottleneckCSP             [256, 256, 1, False]
 22                -1  1     65535  torch.nn.modules.conv.Conv2d            [256, 255, 1, 1]
 23                -2  1    590336  models.common.Conv                      [256, 256, 3, 2]
 24          [-1, 10]  1         0  models.common.Concat                    [1]
 25                -1  1   1248768  models.common.BottleneckCSP             [512, 512, 1, False]
 26                -1  1    130815  torch.nn.modules.conv.Conv2d            [512, 255, 1, 1]
 27    [-1, 22, 18]  1         0  models.yolo.Detect                      [80, [[116, 90, 156, 198, 373, 326], [30, 61, 62, 45, 59, 119], [10, 13, 16, 30, 33, 23]]]
Model Summary: 191 layers, 7.46816e+06 parameters, 7.46816e+06 gradients

Optimizer groups: 62 .bias, 70 conv.weight, 59 other
Scanning labels ../coco128/labels/train2017.cache (126 found, 0 missing, 2 empty, 0 duplicate, for 128 images): 100% 128/128 [00:00<00:00, 20484.22it/s]
Caching images (0.1GB): 100% 128/128 [00:00<00:00, 156.07it/s]
Scanning labels ../coco128/labels/train2017.cache (126 found, 0 missing, 2 empty, 0 duplicate, for 128 images): 100% 128/128 [00:00<00:00, 22082.55it/s]
Caching images (0.1GB): 100% 128/128 [00:00<00:00, 152.91it/s]

Analyzing anchors... Best Possible Recall (BPR) = 0.9935
Image sizes 640 train, 640 test
Using 2 dataloader workers
Starting training for 3 epochs...

     Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
      0/2     6.84G   0.04376   0.06831     0.02    0.1321       225       640: 100% 8/8 [00:09<00:00,  1.22s/it]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 8/8 [00:09<00:00,  1.24s/it]
                 all         128         929       0.34       0.762        0.69       0.446

     Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
      1/2     6.06G   0.04333   0.08225   0.02207    0.1476       182       640: 100% 8/8 [00:03<00:00,  2.17it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 8/8 [00:02<00:00,  3.28it/s]
                 all         128         929       0.342       0.755       0.687       0.447

     Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
      2/2     6.06G    0.0444   0.07251   0.01855    0.1355       216       640: 100% 8/8 [00:03<00:00,  2.15it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 8/8 [00:02<00:00,  3.46it/s]
                 all         128         929       0.354       0.759       0.689        0.45
Optimizer stripped from runs/exp0/weights/last.pt, 15.2MB
3 epochs completed in 0.009 hours.
```

4. 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 (https://arxiv.org/abs/2004.10934)](https://arxiv.org/abs/2004.10934).

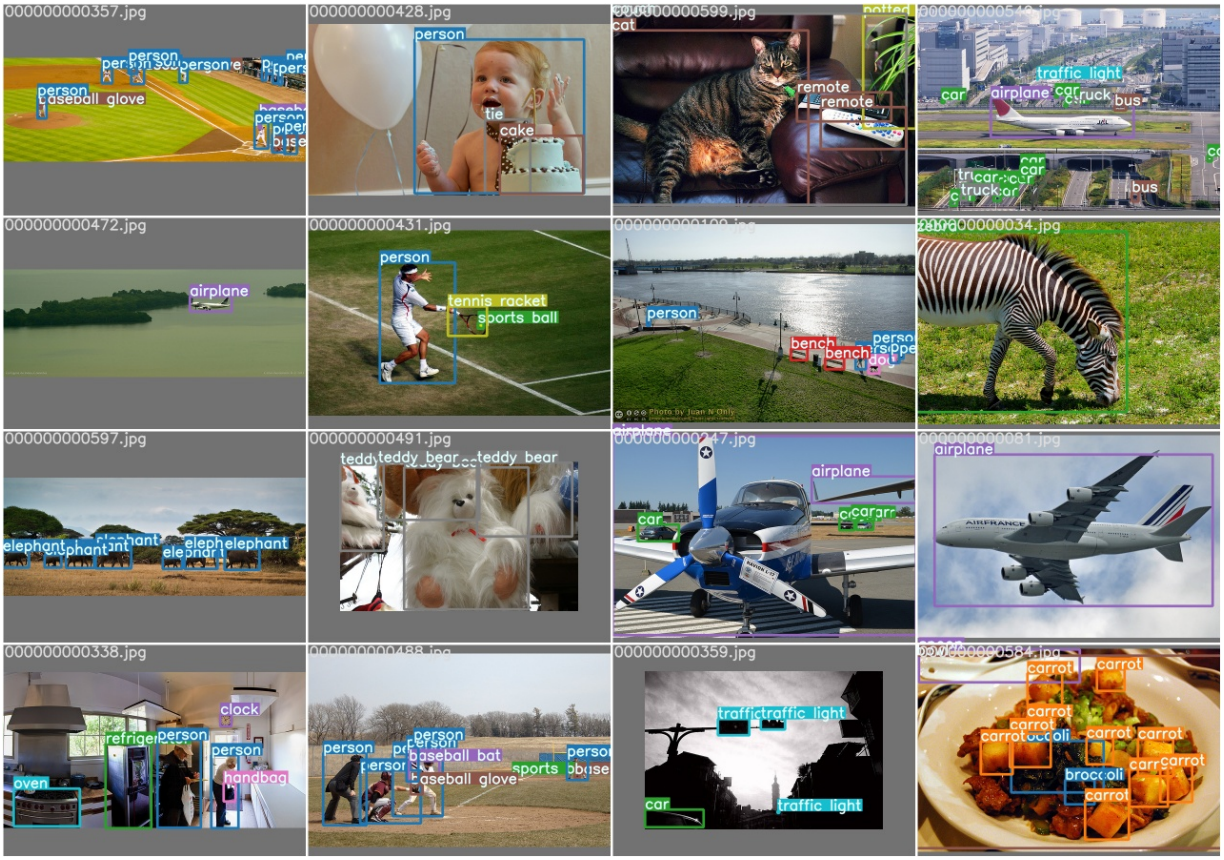
In [ ]:
```
Image(filename='runs/exp0/train_batch1.jpg', width=900)  # view augmented training mosaics
```

Out[ ]:

View `test_batch0_gt.jpg` to see test batch 0 *ground truth* labels.

In [ ]:
```
Image(filename='runs/exp0/test_batch0_gt.jpg', width=900)  # view test image labels
```

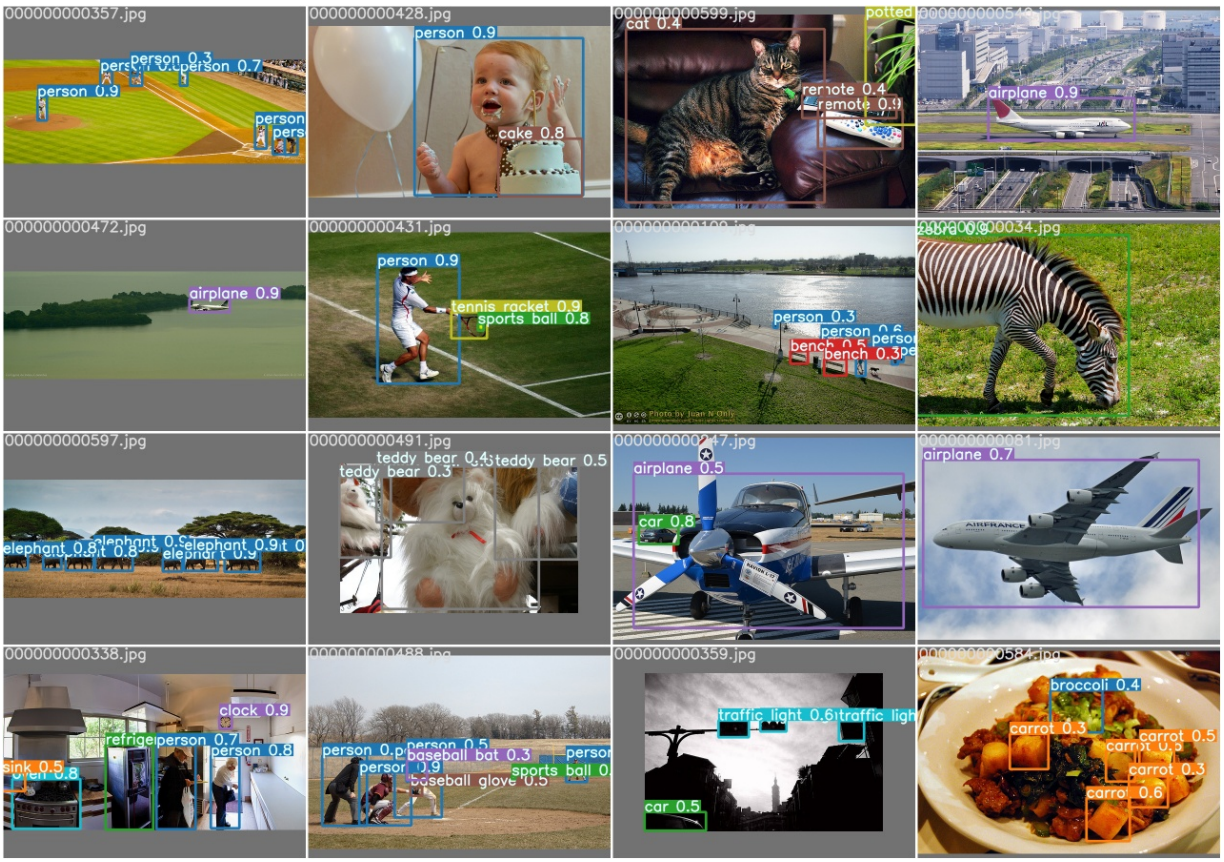
Out[ ]:



View test_batch0_pred.jpg to see test batch 0 *predictions*.

In [ ]:
```
Image(filename='runs/exp0/test_batch0_pred.jpg', width=900)  # view test image predictions
```

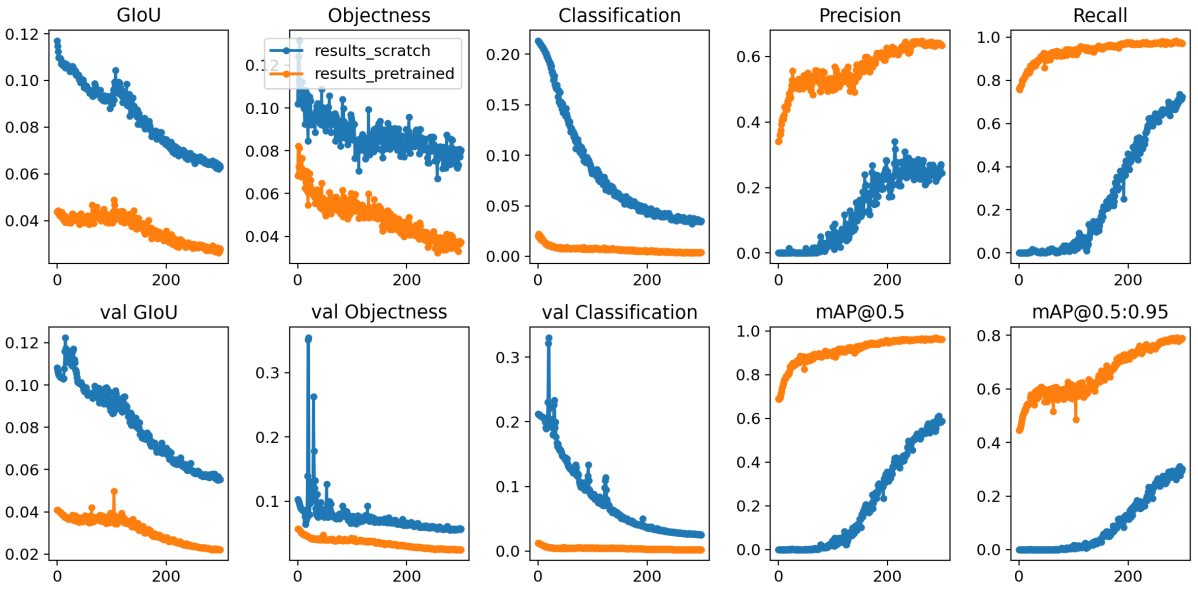
Out[ ]:



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).

In [ ]:
```
from utils.utils import plot_results; plot_results()  # plot results.txt files as results.png
```

Out[ ]:



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: 
- **Kaggle Notebook** with free GPU: https://www.kaggle.com/ultralytics/yolov5 (https://www.kaggle.com/ultralytics/yolov5)
- **Google Cloud** Deep Learning VM. See GCP Quickstart Guide (https://github.com/ultralytics/yolov5/wiki/GCP-Quickstart)
- **Docker Image** https://hub.docker.com/r/ultralytics/yolov5 (https://hub.docker.com/r/ultralytics/yolov5). See Docker Quickstart Guide (https://github.com/ultralytics/yolov5/wiki/Docker-Quickstart)

Appendix

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

In [ ]:
```
# Re-clone repo
%cd ..
!rm -rf yolov5 && git clone https://github.com/ultralytics/yolov5
%cd yolov5
```

In [ ]:
```
# Test GCP ckpt
%%shell
for x in best*
do
  gsutil cp gs://*/*/*/$x.pt .
  python test.py --weights $x.pt --data coco.yaml --img 672
done
```

In [ ]:

```
# YOLOv5 unit tests
%%shell
cd .. && rm -rf yolov5 && git clone https://github.com/ultralytics/yolov5 && cd yolov5
export PYTHONPATH="$PWD" # to run *.py. files in subdirectories
pip install -qr requirements.txt onnx
python3 -c "from utils.google_utils import *; gdrive_download('1n_oKgR81BJtqk75b00eAjdv03qVCQn2f', 'coco128.zip')" &
& mv ./coco128 ../
for x in yolov5s #yolov5m yolov5l yolov5x # models
do
  python train.py --weights $x.pt --cfg $x.yaml --epochs 4 --img 320 --device 0  # train
  for di in 0 cpu # inference devices
  do
    python detect.py --weights $x.pt --device $di  # detect official
    python detect.py --weights runs/exp0/weights/last.pt --device $di  # detect custom
    python test.py --weights $x.pt --device $di # test official
    python test.py --weights runs/exp0/weights/last.pt --device $di # test custom
  done
  python models/yolo.py --cfg $x.yaml # inspect
  python models/export.py --weights $x.pt --img 640 --batch 1 # export
done
```