How to Train YOLOv5 on Custom Objects

This tutorial is based on the YOLOv5 repository by Ultralytics. This notebook shows training on **your own custom objects**. Many thanks to Ultralytics for putting this repository together - we hope that in combination with clean data management tools at Roboflow, this technologoy will become easily accessible to any developer wishing to use computer vision in their projects.

Accompanying Blog Post

We recommend that you follow along in this notebook while reading the blog post on how to train YOLOv5, concurrently.

Steps Covered in this Tutorial

In this tutorial, we will walk through the steps required to train YOLOv5 on your custom objects. We use a public blood cell detection dataset, which is open source and free to use. You can also use this notebook on your own data.

To train our detector we take the following steps:

- Install YOLOv5 dependencies
- Download custom YOLOv5 object detection data
- Write our YOLOv5 Training configuration
- Run YOLOv5 training
- Evaluate YOLOv5 performance
- Visualize YOLOv5 training data
- Run YOLOv5 inference on test images
- Export saved YOLOv5 weights for future inference

About

Roboflow enables teams to deploy custom computer vision models quickly and accurately. Convert data from to annotation format, assess dataset health, preprocess, augment, and more. It's free for your first 1000 source images.

Looking for a vision model available via API without hassle? Try Roboflow Train.

Install Dependencies

(Remember to choose GPU in Runtime if not already selected. Runtime --> Change Runtime Type --> Hardware accelerator --> GPU)

```
# clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
!git reset --hard fbe67e465375231474a2ad80a4389efc77ecff99
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 15002, done.
remote: Total 15002 (delta 0), reused 0 (delta 0), pack-reused 15002
Receiving objects: 100% (15002/15002), 14.08 MiB | 16.27 MiB/s, done.
Resolving deltas: 100% (10285/10285), done.
/content/yolov5/yolov5
HEAD is now at fbe67e4 Fix `OMP_NUM_THREADS=1` for macOS (#8624)
```

```
# install dependencies as necessary
!pip install -qr requirements.txt  # install dependencies (ignore errors)
```

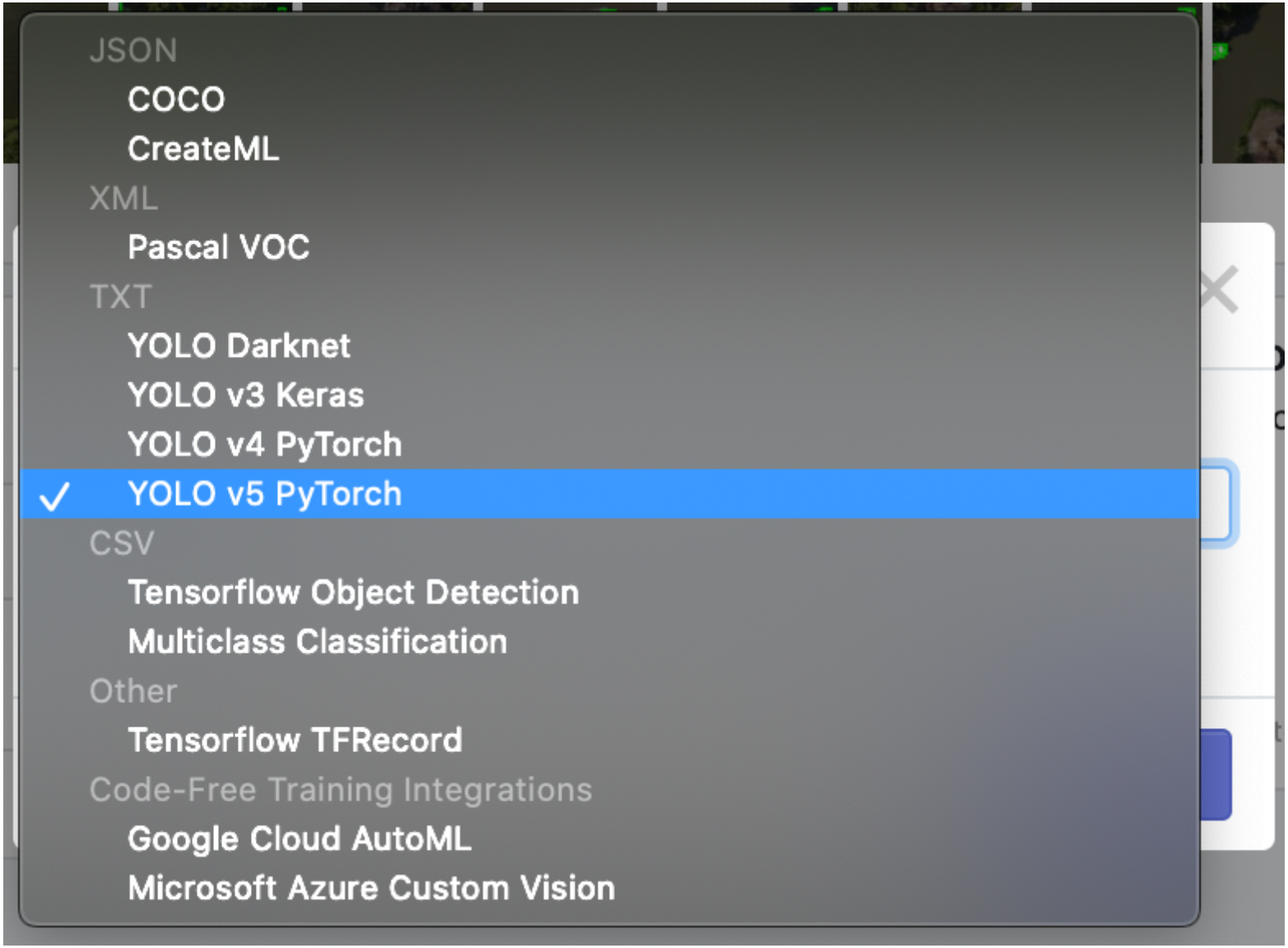
```
import torch

from IPython.display import Image, clear_output  # to display images
from utils.downloads import attempt_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_available() else
```

```
Setup complete. Using torch 1.13.1+cu116 _CudaDeviceProperties(name='Tesla T4', major=7, minor=5, total_memory=15109MB, multi_p
```

Download Correctly Formatted Custom Dataset

We'll download our dataset from Roboflow. Use the "**YOLOv5 PyTorch**" export format. Note that the Ultralytics implementation calls for a YAML file defining where your training and test data is. The Roboflow export also writes this format for us.

To get your data into Roboflow, follow the Getting Started Guide.




```
#follow the link below to get your download code from from Roboflow
!pip install -q roboflow
from roboflow import Roboflow
rf = Roboflow(model_format="yolov5", notebook="roboflow-yolov5")
```

```
    upload and label your dataset, and get an API KEY here: https://app.roboflow.com/?model=yolov5&ref=roboflow-yolov5
```

```
%cd /content/yolov5
#after following the link above, recieve python code with these fields filled in
#from roboflow import Roboflow
#rf = Roboflow(api_key="YOUR API KEY HERE")
#project = rf.workspace().project("YOUR PROJECT")
#dataset = project.version("YOUR VERSION").download("yolov5")
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="lPqxyXvJ0FK2pYNy2vtA")
project = rf.workspace().project("hook-detection")
dataset = project.version(1).download("yolov5")

# !curl -L "https://app.roboflow.com/ds/mcVn2RLlK3?key=6FHrSRydT5" &gt; roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

```
    /content/yolov5
    loading Roboflow workspace...
    loading Roboflow project...
    Downloading Dataset Version Zip in Hook-Detection-1 to yolov5pytorch: 100% [34604891 / 34604891] bytes
    Extracting Dataset Version Zip to Hook-Detection-1 in yolov5pytorch:: 100%|██████████| 1796/1796 [00:00<00:00, 3725.40it/s]
```

```
# this is the YAML file Roboflow wrote for us that we're loading into this notebook with our data
%cat {dataset.location}/data.yaml
```

```
    names:
    - hook
    nc: 1
    roboflow:
      license: CC BY 4.0
      project: hook-detection
      url: https://universe.roboflow.com/project/hook-detection/dataset/1
      version: 1
      workspace: project
    test: ../test/images
```

```
train: Hook-Detection-1/train/images
val: Hook-Detection-1/valid/images
```

Define Model Configuration and Architecture

We will write a yaml script that defines the parameters for our model like the number of classes, anchors, and each layer.

You do not need to edit these cells, but you may.

```
# define number of classes based on YAML
import yaml
with open(dataset.location + "/data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])


#this is the model configuration we will use for our tutorial
%cat /content/yolov5/models/yolov5s.yaml

     # YOLOv5 🚀 by Ultralytics, GPL-3.0 license

     # Parameters
     nc: 80  # number of classes
     depth_multiple: 0.33  # model depth multiple
     width_multiple: 0.50  # layer channel multiple
     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 v6.0 backbone
     backbone:
       # [from, number, module, args]
       [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
        [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
        [-1, 3, C3, [128]],
        [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
```

```
        [-1, 6, C3, [256]],
        [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
        [-1, 9, C3, [512]],
        [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
        [-1, 3, C3, [1024]],
        [-1, 1, SPPF, [1024, 5]],  # 9
       ]

     # YOLOv5 v6.0 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, C3, [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, C3, [256, False]],  # 17 (P3/8-small)

        [-1, 1, Conv, [256, 3, 2]],
        [[-1, 14], 1, Concat, [1]],  # cat head P4
        [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

        [-1, 1, Conv, [512, 3, 2]],
        [[-1, 10], 1, Concat, [1]],  # cat head P5
        [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

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


#customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))
```

```
%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# parameters
nc: {num_classes}  # 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)
  ]
```

Train Custom YOLOv5 Detector

Next, we'll fire off training!

Here, we are able to pass a number of arguments:

- **img:** define input image size
- **batch:** determine batch size
- **epochs:** define the number of training epochs. (Note: often, 3000+ are common here!)
- **data:** set the path to our yaml file
- **cfg:** specify our model configuration
- **weights:** specify a custom path to weights. (Note: you can download weights from the Ultralytics Google Drive folder)
- **name:** result names
- **nosave:** only save the final checkpoint
- **cache:** cache images for faster training

```
# train yolov5s on custom data for 100 epochs
# time its performance
```

```
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 100 --data {dataset.location}/data.yaml --cfg ./models/custom_yolov5s.yaml --weights '
```

```
/content/yolov5
train: weights=, cfg=./models/custom_yolov5s.yaml, data=/content/yolov5/Hook-Detection-1/data.yaml, hyp=data/hyps/hyp.scratch
github: ⚠️ YOLOv5 is out of date by 436 commits. Use `git pull` or `git clone https://github.com/ultralytics/yolov5` to updat
YOLOv5 🚀 v6.1-306-gfbe67e4 Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15110MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias
Weights & Biases: run 'pip install wandb' to automatically track and visualize YOLOv5 🚀 runs (RECOMMENDED)
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 111MB/s]

                 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  3    161152  models.common.BottleneckCSP             [128, 128, 3]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    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    147712  models.common.Conv                      [128, 128, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  1    313088  models.common.BottleneckCSP             [256, 256, 1, False]
 21                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  1   1248768  models.common.BottleneckCSP             [512, 512, 1, False]
```

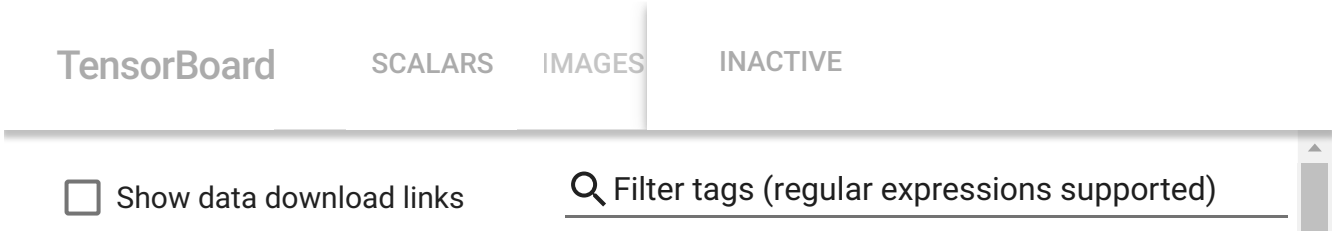
```
 24      [17, 20, 23]  1     16182  models.yolo.Detect                      [1, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 5
custom_YOLOv5s summary: 283 layers, 7255094 parameters, 7255094 gradients, 16.9 GFLOPs

AMP: checks passed ✅
Scaled weight_decay = 0.0005
optimizer: SGD with parameter groups 59 weight (no decay), 70 weight, 62 bias
albumentations: Blur(always_apply=False, p=0.01, blur_limit=(3, 7)), MedianBlur(always_apply=False, p=0.01, blur_limit=(3, 7)
train: Scanning '/content/yolov5/Hook-Detection-1/train/labels' images and labels...772 found, 0 missing, 0 empty, 0 corrupt:
train: New cache created: /content/yolov5/Hook-Detection-1/train/labels.cache
train: Caching images (0.4GB ram): 100% 772/772 [00:01<00:00, 389.00it/s]
val: Scanning '/content/yolov5/Hook-Detection-1/valid/labels' images and labels...80 found, 0 missing, 0 empty, 0 corrupt: 10
val: New cache created: /content/yolov5/Hook-Detection-1/valid/labels.cache
val: Caching images (0.0GB ram): 100% 80/80 [00:00<00:00, 164.08it/s]
Plotting labels to runs/train/yolov5s_results/labels.jpg...

AutoAnchor: 1.85 anchors/target, 0.756 Best Possible Recall (BPR). Anchors are a poor fit to dataset ⚠️, attempting to improv
AutoAnchor: WARNING: Extremely small objects found: 2455 of 12566 labels are < 3 pixels in size
AutoAnchor: Running kmeans for 9 anchors on 12566 points...
AutoAnchor: Evolving anchors with Genetic Algorithm: fitness = 0.7161: 100% 1000/1000 [00:02<00:00, 385.79it/s]
AutoAnchor: thr=0.25: 0.9998 best possible recall, 4.96 anchors past thr
```

Evaluate Custom YOLOv5 Detector Performance

Training losses and performance metrics are saved to Tensorboard and also to a logfile defined above with the **--name** flag when we train. In our case, we named this . (If given no name, it defaults to .) The results file is plotted as a png after training completes. `yolov5s_results results.txt`

Note from Glenn: Partially completed files can be plotted with . `results.txt from utils.utils import plot_results; plot_results()`

```
# Start tensorboard
# Launch after you have started training
# logs save in the folder "runs"
%load_ext tensorboard
%tensorboard --logdir runs
```




```
# we can also output some older school graphs if the tensor board isn't working for whatever reason...
from utils.plots import plot_results  # plot results.txt as results.png
Image(filename='/content/yolov5/runs/train/yolov5s_results/results.png', width=1000)  # view results.png
```

Curious? Visualize Our Training Data with Labels

After training starts, view images to see training images, labels and augmentation effects. `train*.jpg`

Note a mosaic dataloader is used for training (shown below), a new dataloading concept developed by Glenn Jocher and first featured in YOLOv4.

0 50 100 0 50 100 0 50 100 0 50 100 0 50 100

```
# first, display our ground truth data
print("GROUND TRUTH TRAINING DATA:")
Image(filename='/content/yolov5/Hook-Detection-1/test/images/hook_216_jpg.rf.9867286df9298389ca232bda9becf703.jpg', width=900)
```

```
GROUND TRUTH TRAINING DATA:
```

```
# print out an augmented training example
print("GROUND TRUTH AUGMENTED TRAINING DATA:")
Image(filename='/content/yolov5/Hook-Detection-1/test/images/hook_216_jpg.rf.9867286df9298389ca232bda9becf703.jpg', width=900)
```

GROUND TRUTH AUGMENTED TRAINING DATA:

Run Inference With Trained Weights

Run inference with a pretrained checkpoint on contents of folder downloaded from Roboflow. `test/images`

```
# trained weights are saved by default in our weights folder
%ls runs/
```

```
train/
```

```
%ls runs/train/yolov5s_results/weights
```

```
best.pt  last.pt
```

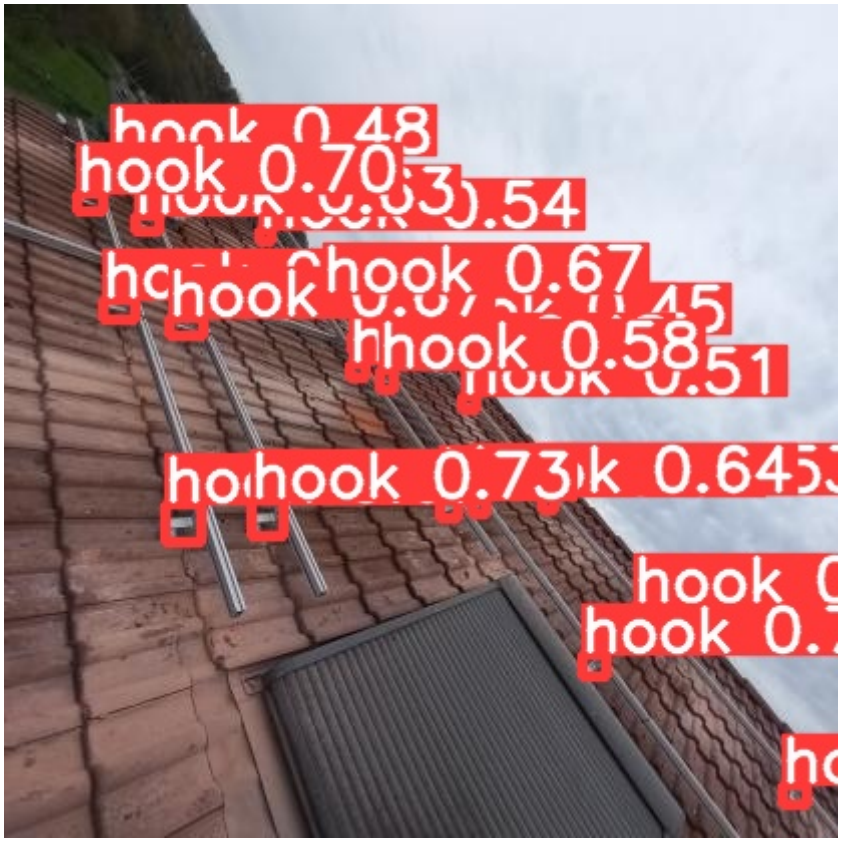
```
# when we ran this, we saw .007 second inference time. That is 140 FPS on a TESLA P100!
# use the best weights!
%cd /content/yolov5/
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.4 --source /content/yolov5/Hook-Detection-1
```

```
/content/yolov5
detect: weights=['runs/train/yolov5s_results/weights/best.pt'], source=/content/yolov5/Hook-Detection-1/test/images, data=data/
YOLOv5 🚀 v6.1-306-gfbe67e4 Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15110MiB)

Fusing layers...
custom_YOLOv5s summary: 232 layers, 7246518 parameters, 0 gradients, 16.7 GFLOPs
image 1/40 /content/yolov5/Hook-Detection-1/test/images/hook_104_jpg.rf.b93c319f6419b167b4ce35ce9065e837.jpg: 416x416 9 hooks,
image 2/40 /content/yolov5/Hook-Detection-1/test/images/hook_129_jpg.rf.d415a0aba08a0e151d588246c6bc5746.jpg: 416x416 18 hooks,
image 3/40 /content/yolov5/Hook-Detection-1/test/images/hook_133_jpg.rf.aa6db85548b794fcb00104441329bafd.jpg: 416x416 11 hooks,
image 4/40 /content/yolov5/Hook-Detection-1/test/images/hook_135_jpg.rf.85641759e645c0c38c41d366feea9417.jpg: 416x416 2 hooks,
image 5/40 /content/yolov5/Hook-Detection-1/test/images/hook_145_jpg.rf.480b7825fee3a92c61271eba3b14c6c8.jpg: 416x416 17 hooks,
image 6/40 /content/yolov5/Hook-Detection-1/test/images/hook_156_jpg.rf.db82b41dd1ec5b91e363410c4f3a7dc5.jpg: 416x416 6 hooks,
image 7/40 /content/yolov5/Hook-Detection-1/test/images/hook_165_jpg.rf.abf883b3afd392ea318f8c8b2c7a276d.jpg: 416x416 1 hook, D
```

```
image 8/40 /content/yolov5/Hook-Detection-1/test/images/hook_189_jpg.rf.f61fc6f46933604868062b546d9bf14e.jpg: 416x416 17 hooks,
image 9/40 /content/yolov5/Hook-Detection-1/test/images/hook_208_jpg.rf.ed0b410ac675a464c2b43921c194e3d2.jpg: 416x416 8 hooks,
image 10/40 /content/yolov5/Hook-Detection-1/test/images/hook_210_jpg.rf.23e4237bfcdcdc73b55d6763cca31931.jpg: 416x416 2 hooks,
image 11/40 /content/yolov5/Hook-Detection-1/test/images/hook_216_jpg.rf.9867286df9298389ca232bda9becf703.jpg: 416x416 4 hooks,
image 12/40 /content/yolov5/Hook-Detection-1/test/images/hook_222_jpg.rf.2e60bd8d0c3b3ba96ad1d7414e89f236.jpg: 416x416 12 hooks
image 13/40 /content/yolov5/Hook-Detection-1/test/images/hook_233_jpg.rf.f45349082f57b4a21b5897277a4cec64.jpg: 416x416 2 hooks,
image 14/40 /content/yolov5/Hook-Detection-1/test/images/hook_234_jpg.rf.40e324c975b83b3357eb882305377805.jpg: 416x416 2 hooks,
image 15/40 /content/yolov5/Hook-Detection-1/test/images/hook_23_jpg.rf.04fed1454dc025c73234ed4ef11410e0.jpg: 416x416 Done. (0.
image 16/40 /content/yolov5/Hook-Detection-1/test/images/hook_257_jpg.rf.79927fa29972bc720d1a5e1cf7077bcd.jpg: 416x416 17 hooks
image 17/40 /content/yolov5/Hook-Detection-1/test/images/hook_258_jpg.rf.3ed7329d5f23ec636753e6344516e67d.jpg: 416x416 3 hooks,
image 18/40 /content/yolov5/Hook-Detection-1/test/images/hook_25_jpg.rf.d857eb990e0d681d4caaad40806fdc00.jpg: 416x416 Done. (0.
image 19/40 /content/yolov5/Hook-Detection-1/test/images/hook_266_jpg.rf.0ed1b7f285e2ba9c70fd47743a54901f.jpg: 416x416 24 hooks
image 20/40 /content/yolov5/Hook-Detection-1/test/images/hook_278_jpg.rf.3899609a22ef0e39202008d1082b1d47.jpg: 416x416 10 hooks
image 21/40 /content/yolov5/Hook-Detection-1/test/images/hook_293_jpg.rf.199da2a0fc03f748138ba24b85c224d4.jpg: 416x416 9 hooks,
image 22/40 /content/yolov5/Hook-Detection-1/test/images/hook_299_jpg.rf.3de30b7d0fcd663f002ee6077db6184d.jpg: 416x416 4 hooks,
image 23/40 /content/yolov5/Hook-Detection-1/test/images/hook_304_jpg.rf.cf52e8ba759f43d284248c6465966ebb.jpg: 416x416 2 hooks,
image 24/40 /content/yolov5/Hook-Detection-1/test/images/hook_309_jpg.rf.d88cab258b8a9e08de693f693235c9ac.jpg: 416x416 15 hooks
image 25/40 /content/yolov5/Hook-Detection-1/test/images/hook_311_jpg.rf.23c9559b116dd0af5d20be6bffd20d2f.jpg: 416x416 11 hooks
image 26/40 /content/yolov5/Hook-Detection-1/test/images/hook_344_jpg.rf.a8bf24501e33962b7fcfa273a9c5ce61.jpg: 416x416 20 hooks
image 27/40 /content/yolov5/Hook-Detection-1/test/images/hook_345_jpg.rf.643824883990b7907a90f1da569cb360.jpg: 416x416 14 hooks
image 28/40 /content/yolov5/Hook-Detection-1/test/images/hook_353_jpg.rf.65eddf3854e1afb4b45131b8a6171f89.jpg: 416x416 9 hooks,
image 29/40 /content/yolov5/Hook-Detection-1/test/images/hook_363_jpg.rf.6a5efeb96ec25ded09ba760011f305c4.jpg: 416x416 2 hooks,
image 30/40 /content/yolov5/Hook-Detection-1/test/images/hook_376_jpg.rf.0cf20f5351a1187b6f4bd0a9e2104fdd.jpg: 416x416 27 hooks
image 31/40 /content/yolov5/Hook-Detection-1/test/images/hook_383_jpg.rf.fce43b10eb6a9c62a6799a0614798157.jpg: 416x416 2 hooks,
image 32/40 /content/yolov5/Hook-Detection-1/test/images/hook_393_jpg.rf.ce5e49048306f3c4e058f3d3175c4d90.jpg: 416x416 26 hooks
image 33/40 /content/yolov5/Hook-Detection-1/test/images/hook_396_jpg.rf.89998d0b9decaea6ec318289c3a64570.jpg: 416x416 20 hooks
image 34/40 /content/yolov5/Hook-Detection-1/test/images/hook_53_jpg.rf.0f4dbe66e9bc1f19829cbab8693200d3.jpg: 416x416 Done. (0.
image 35/40 /content/yolov5/Hook-Detection-1/test/images/hook_55_jpg.rf.6a13bd170afd899703e67c8421e71a52.jpg: 416x416 2 hooks,
image 36/40 /content/yolov5/Hook-Detection-1/test/images/hook_68_jpg.rf.f2b51ceedb4076b82b99f6010e080194.jpg: 416x416 7 hooks,
image 37/40 /content/yolov5/Hook-Detection-1/test/images/hook_78_jpg.rf.c6a2736627db4c2ccadc99e94b450a5f.jpg: 416x416 1 hook, D
image 38/40 /content/yolov5/Hook-Detection-1/test/images/hook_88_jpg.rf.6b25dafdae99523f343b7dee951d9e6a.jpg: 416x416 13 hooks,
image 39/40 /content/yolov5/Hook-Detection-1/test/images/hook_89_jpg.rf.87dc112fb612ca7a93de1cb4c33902d3.jpg: 416x416 7 hooks,
image 40/40 /content/yolov5/Hook-Detection-1/test/images/hook_96_jpg.rf.4bb178bda490945bd7b148cebc13e4c1.jpg: 416x416 9 hooks,
Speed: 0.3ms pre-process, 10.4ms inference, 0.9ms NMS per image at shape (1, 3, 416, 416)
Results saved to runs/detect/exp3
```

```
#display inference on ALL test images
#this looks much better with longer training above
```

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

for imageName in glob.glob('/content/yolov5/runs/detect/exp3/*.jpg'): #assuming JPG
    display(Image(filename=imageName))
    print("\n")
```


hook 0.70
hook 0.67
hook 0.58
hook 0.73

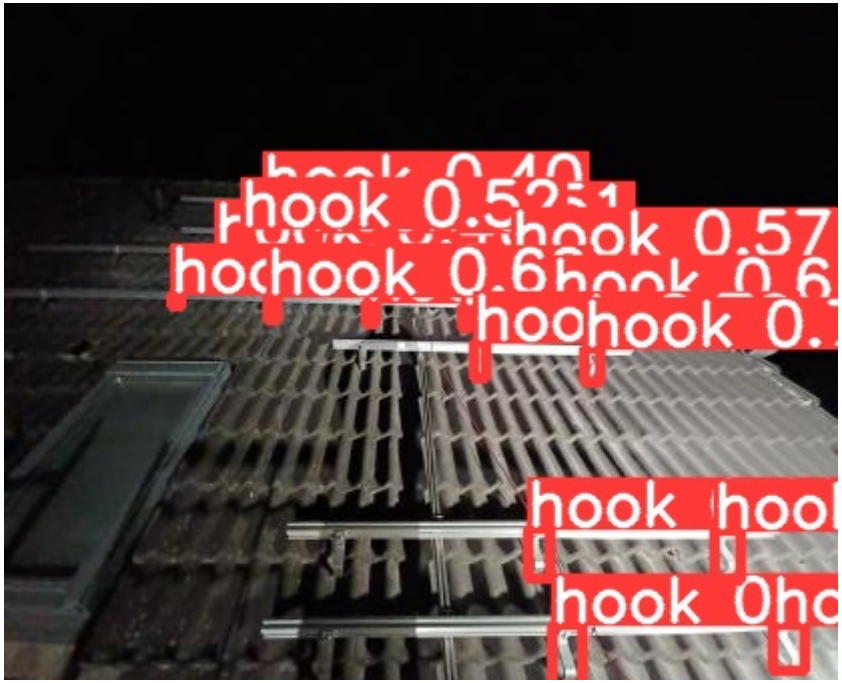

hook 0.74
hook 0.80
hook 0.77
hook 0.79
hook 0.51


hook 0.43
hook 0.43

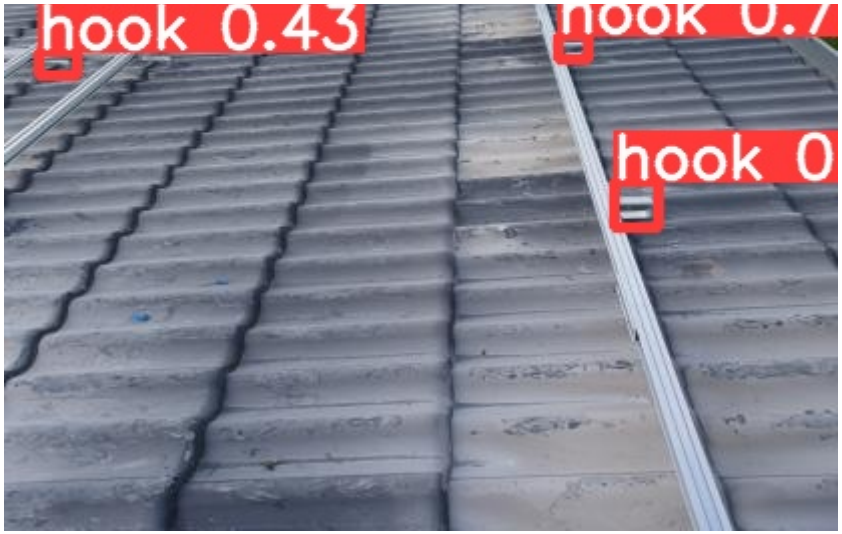

hook 0.68
hook


hook 0.71
0.62
hook 0.73

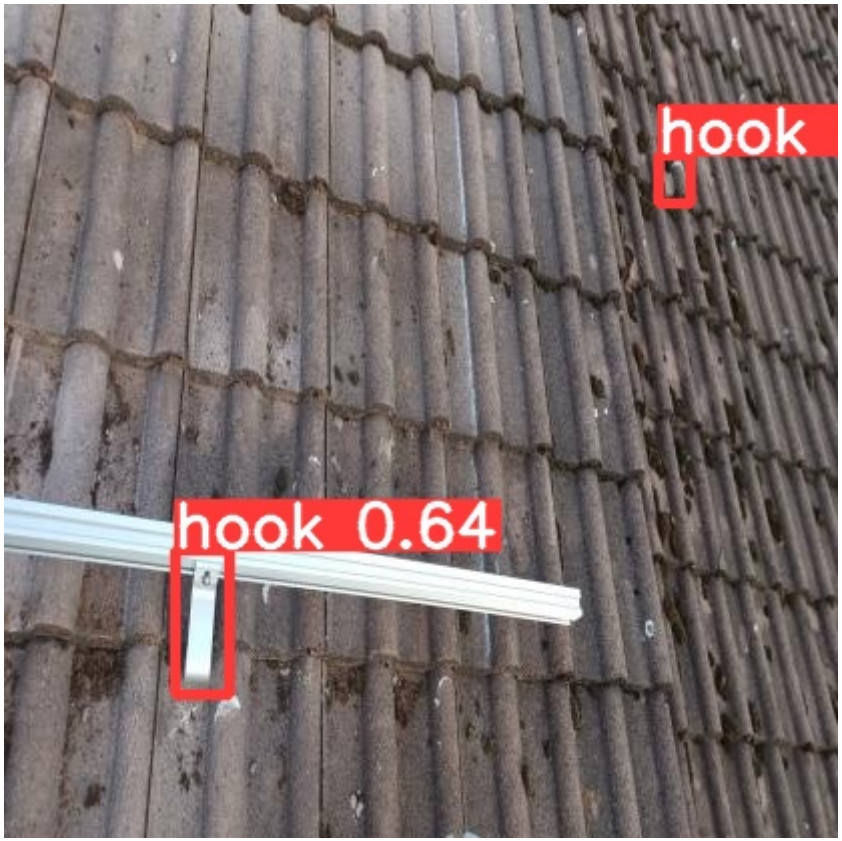

hook 0.69
hook 0.77
hook 0.74
hook 0.79
hook 0.82


hook 0.57


hook 0.76
hook 0.47


hook 0.43
hook 0


hook 0.41
hook 0.68
hook 0.75
hook 0.73


hook
hook 0.64

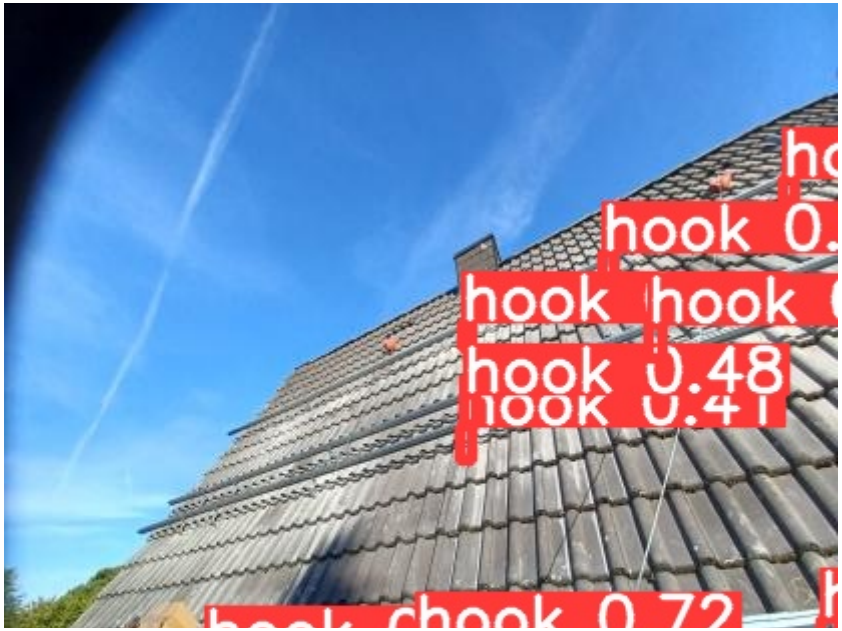

hook
hook 0.75
hook 0.75
hook 0.81


hook 0.82


hook 0.48
hook 0.65
hook 0.71


hook 0.42
hook 0.52


hook 0.78
hook 0.83


hook 0.48


hook 0.51
hook 0.43


hoolh


hook 0.71
hook 0.44


hook 0.66
hook 0.61
hook 0.56


hook 0.56

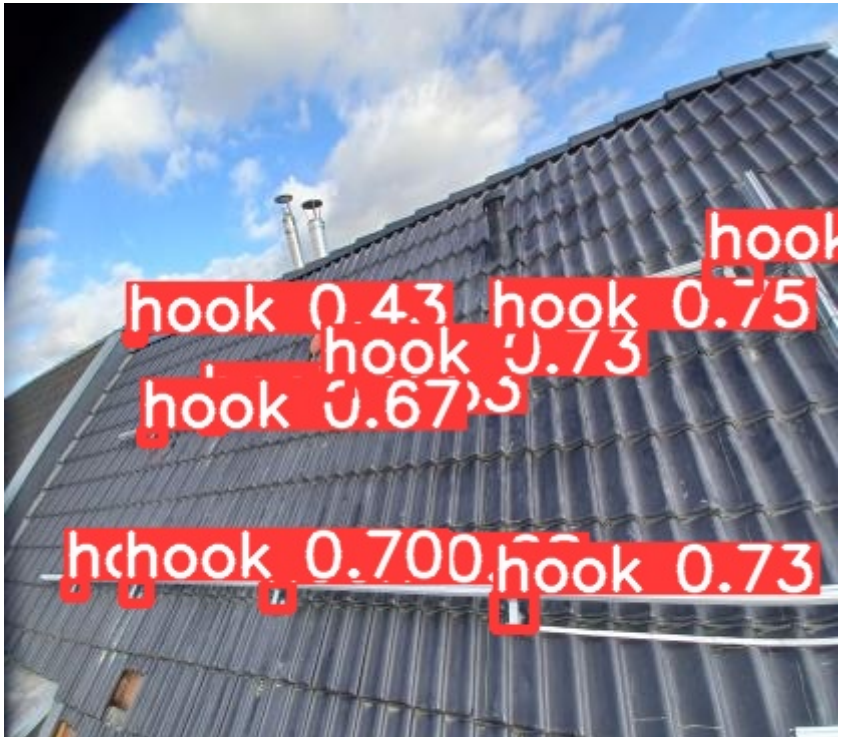

hook 0.57
hook 0.47
hook 0.63
hook 0.44


hook 0.41
hook 0.hook
hook 0.77
hook 0.80
hook 0


hook hook 0.67 ook 0.61
hook 0.72

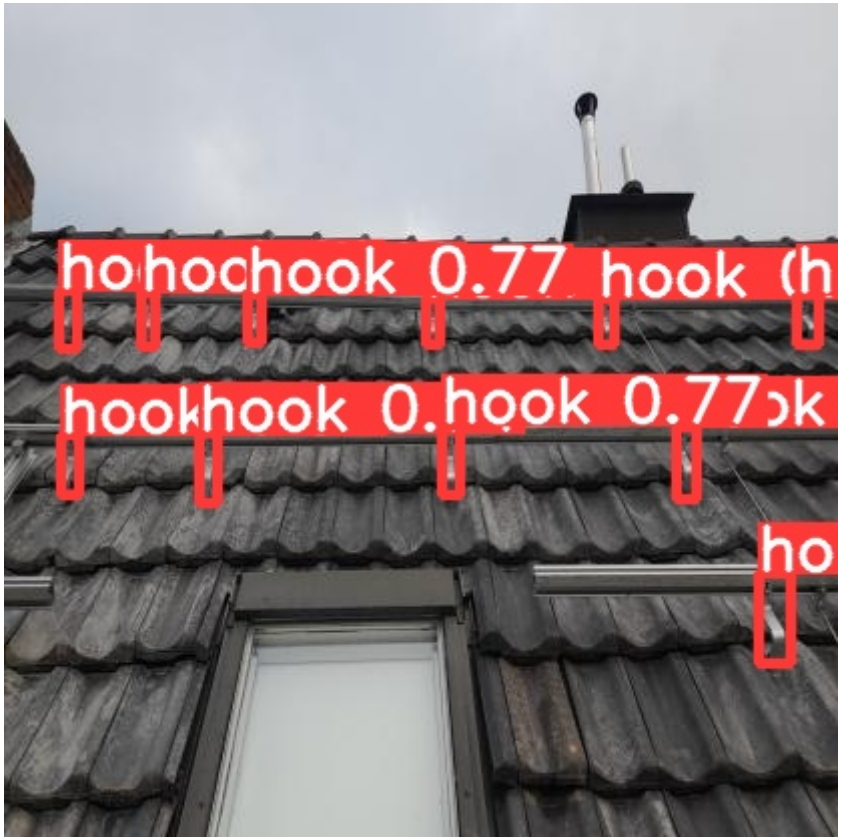

hook 0.61

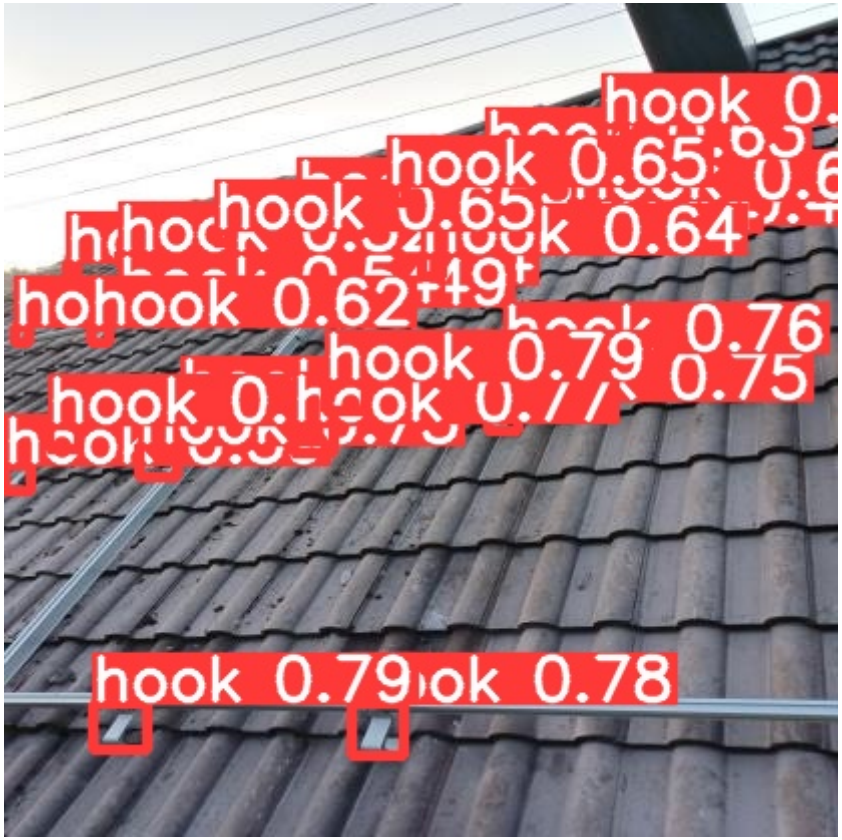

hook 0.76


hook 0.75
hook 0.73
hook 0.73


hook 0.59
hook 0.66
hook 0.66
hook 0.81
hook 0.79
hook 0.72


hook 0.56


hook 0.77
hook 0.77


hook 0.
hook 0.65
hook 0.64
hook 0.62
hook 0.76
hook 0.79
0.75
hook 0.79
ok 0.78

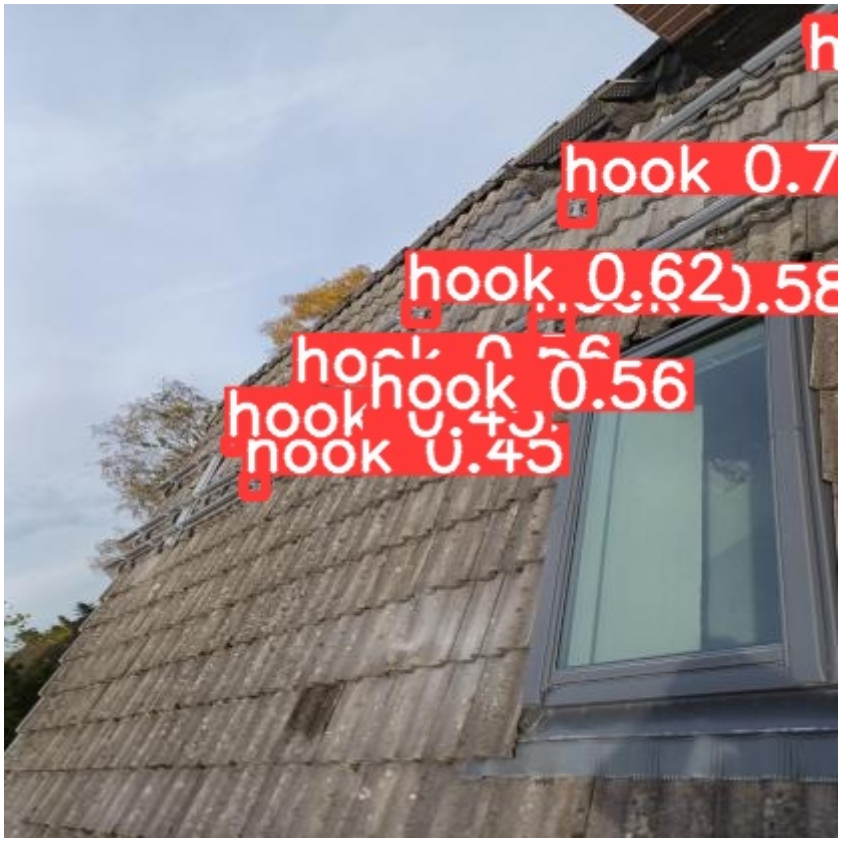

hook 0.77

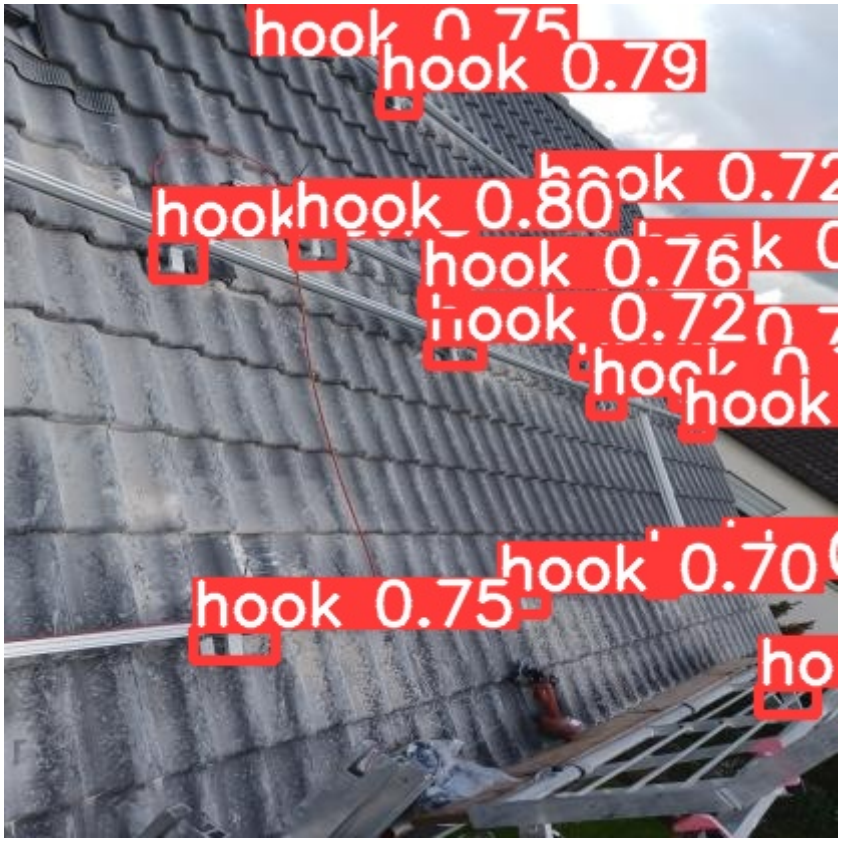

hook 0.49
hook
hook


hook 0.75
ok 0
hook 0.63
hook 0.72


hook 0
hook 0.78

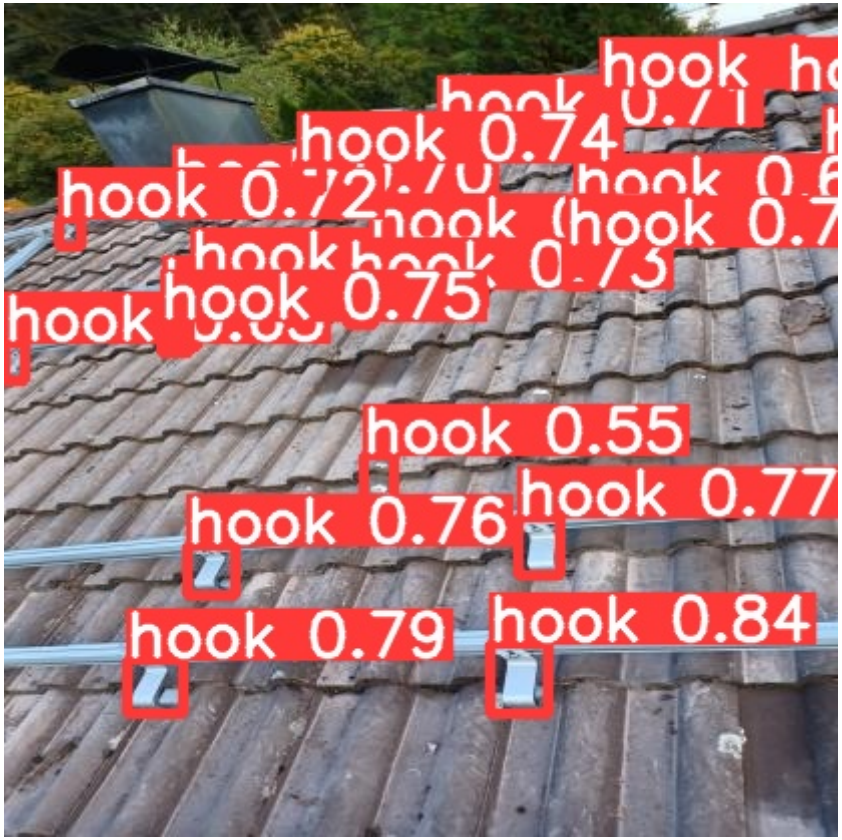

hook 0.54


hook 0.7
hook 0.62
hook 0.56
hook 0.45


hook 0.79
hook 0.80
hook 0.76
hook 0.75


hook 0.70
hook 0.65


hook 0.74


hook 0.74
hook 0.75
hook 0.55
hook 0.76
hook 0.77
hook 0.79
hook 0.84

Export Trained Weights for Future Inference

Now that you have trained your custom detector, you can export the trained weights you have made here for inference on your device elsewhere

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
%cp /content/yolov5/runs/train/yolov5s_results/weights/best.pt /content/gdrive/My\ Drive
```

Congrats!

Hope you enjoyed this!

--Team [Roboflow]