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
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 16575, done.
remote: Counting objects: 100% (53/53), done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 16575 (delta 28), reused 37 (delta 18), pack-reused 16522
Receiving objects: 100% (16575/16575), 15.03 MiB | 25.96 MiB/s, done.
Resolving deltas: 100% (11387/11387), done.
/content/yolov5
```

```
# 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 'CPU'))
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.3/207.3 kB 2.1 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 23.9 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 755.2/755.2 kB 50.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 7.7 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source
imageio 2.31.6 requires pillow<10.1.0,>=8.3.2, but you have pillow 10.3.0 which is incompatible.
Setup complete. Using torch 2.2.1+cu121 _CudaDeviceProperties(name='Tesla T4', major=7, minor=5, total_memory=15102MB, multi_processor_c
```

Step 6: Download a Dataset

Add your Roboflow API key below to download the default money counting dataset. Alternatively, use the code provided by the Roboflow dashboard in the above step to load a custom dataset.

```
!pip install -q roboflow
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 74.6/74.6 kB 1.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 158.3/158.3 kB 3.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.7/178.7 kB 12.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 6.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.1/49.1 MB 13.1 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.5/54.5 kB 6.5 MB/s eta 0:00:00
```

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="nzRKDl9IvGTHb70vn9f4")
project = rf.workspace("logic-gate-dlan3").project("logic-gates-2")
version = project.version(1)
dataset = version.download("yolov5")
```

```
Requirement already satisfied: roboflow in /usr/local/lib/python3.10/dist-packages (1.1.28)
Requirement already satisfied: certifi==2023.7.22 in /usr/local/lib/python3.10/dist-packages (from roboflow) (2023.7.22)
Requirement already satisfied: chardet==4.0.0 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.0.0)
Requirement already satisfied: cycler==0.10.0 in /usr/local/lib/python3.10/dist-packages (from roboflow) (0.10.0)
Requirement already satisfied: idna==2.10 in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.10)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.4.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from roboflow) (3.7.1)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.25.2)
Requirement already satisfied: opencv-python-headless==4.8.0.74 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.8.0.74)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from roboflow) (10.3.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.8.2)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.0.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.31.0)
```

```
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.16.0)
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.0.7)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.66.2)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from roboflow) (6.0.1)
Requirement already satisfied: requests-toolbelt in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.0.0)
Requirement already satisfied: python-magic in /usr/local/lib/python3.10/dist-packages (from roboflow) (0.4.27)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (1.2.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (4.51.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (3.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->roboflow) (3.3.2)
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in logic-gates-2-1 to yolov5pytorch:: 100%|██████████| 29932/29932 [00:00<00:00, 67456.50it/s]

Extracting Dataset Version Zip to logic-gates-2-1 in yolov5pytorch:: 100%|██████████| 2436/2436 [00:00<00:00, 8782.34it/s]
```

```
%pwd
```

```
'/content/yolov5'
```

```
dataset.location
```

```
'/content/yolov5/logic-gates-2-1'
```

```
%cd /content/yolov5
```

```
/content/yolov5
```

```
dataset.location
```

```
'/content/yolov5/logic-gates-2-1'
```

```
# 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:
- '0'
- '1'
- '2'
- '3'
- '4'
- '5'
- '6'
nc: 7
roboflow:
  license: CC BY 4.0
  project: logic-gates-2
  url: https://universe.roboflow.com/logic-gate-dlan3/logic-gates-2/dataset/1
  version: 1
  workspace: logic-gate-dlan3
test: ../test/images
train: logic-gates-2-1/train/images
val: logic-gates-2-1/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'])
```

```
num_classes
```

```
'7'
```

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

```
# YOLOv5 🚀 by Ultralytics, AGPL-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 200 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 50 --data {dataset.location}/data.yaml --cfg ./models/custom_yolov5s.yaml --weights 'yolov5s.
```

```
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:04<00:00,  1.64it/s]
                   all        243       1082      0.676       0.71      0.654      0.471

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      43/49      1.98G    0.02622    0.02856    0.02513        126        416: 100% 53/53 [00:09<00:00,  5.36it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:04<00:00,  1.81it/s]
                   all        243       1082      0.709      0.673       0.67      0.483

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      44/49      1.98G     0.0262    0.02771    0.02472        105        416: 100% 53/53 [00:10<00:00,  4.86it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:05<00:00,  1.49it/s]
                   all        243       1082      0.714      0.646      0.674      0.483

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      45/49      1.98G    0.02608    0.02818    0.02457        120        416: 100% 53/53 [00:10<00:00,  5.23it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:05<00:00,  1.59it/s]
                   all        243       1082      0.707      0.695       0.66      0.476

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      46/49      1.98G    0.02543     0.0284    0.02396        122        416: 100% 53/53 [00:09<00:00,  5.63it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:05<00:00,  1.60it/s]
                   all        243       1082      0.729      0.705      0.672      0.478

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      47/49      1.98G    0.02546    0.02842    0.02315        114        416: 100% 53/53 [00:09<00:00,  5.66it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:04<00:00,  1.62it/s]
                   all        243       1082      0.711       0.69      0.672      0.488

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      48/49      1.98G    0.02586    0.02739    0.02456         90        416: 100% 53/53 [00:10<00:00,  5.29it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:04<00:00,  1.90it/s]
                   all        243       1082      0.742      0.694      0.673      0.491

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      49/49      1.98G    0.02524     0.0282    0.02347         91        416: 100% 53/53 [00:10<00:00,  5.26it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:04<00:00,  1.76it/s]
                   all        243       1082      0.713      0.706      0.674      0.491

50 epochs completed in 0.216 hours.
Optimizer stripped from runs/train/yolov5s_results/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/yolov5s_results/weights/best.pt, 14.8MB

Validating runs/train/yolov5s_results/weights/best.pt...
Fusing layers...
custom_YOLOv5s summary: 182 layers, 7262700 parameters, 0 gradients
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:09<00:00,  1.15s/it]
                   all        243       1082      0.742      0.694      0.673      0.491
                     0        243        334      0.897      0.961      0.955      0.701
                     1        243        163      0.748      0.975      0.952       0.73
                     2        243        117      0.831      0.545      0.785      0.584
                     3        243        180      0.914      0.961      0.974      0.663
                     4        243        166      0.535      0.819      0.634      0.466
                     5        243         18          1          0      0.115     0.0824
                     6        243        104       0.27      0.596      0.293      0.212
Results saved to runs/train/yolov5s_results
CPU times: user 9.83 s, sys: 1.18 s, total: 11 s
Wall time: 13min 46s
```

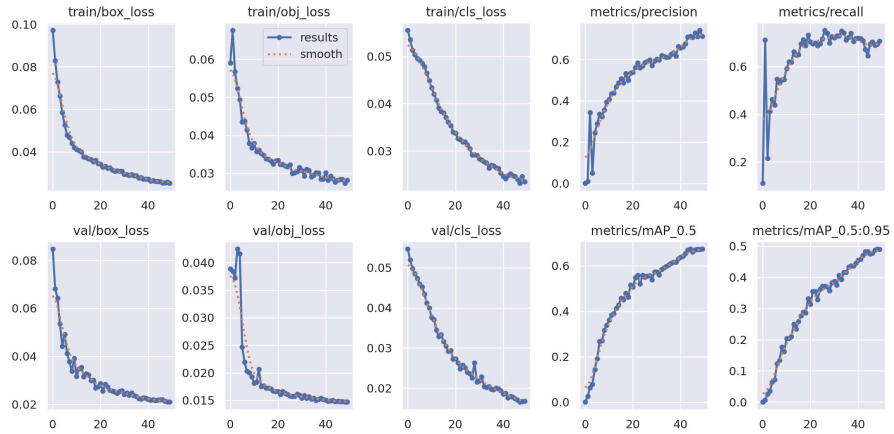
Evaluate Custom YOLOv5 Detector Performance

You can view the training graphs associated with a training job in the `/content/yolov5/runs/train/yolov5s_results/results.png` folder.

Training losses and performance metrics are also saved to Tensorboard and also to a logfile defined above with the **--name** flag when we train. In our case, we named this `yolov5s_results`.

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

```
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
```

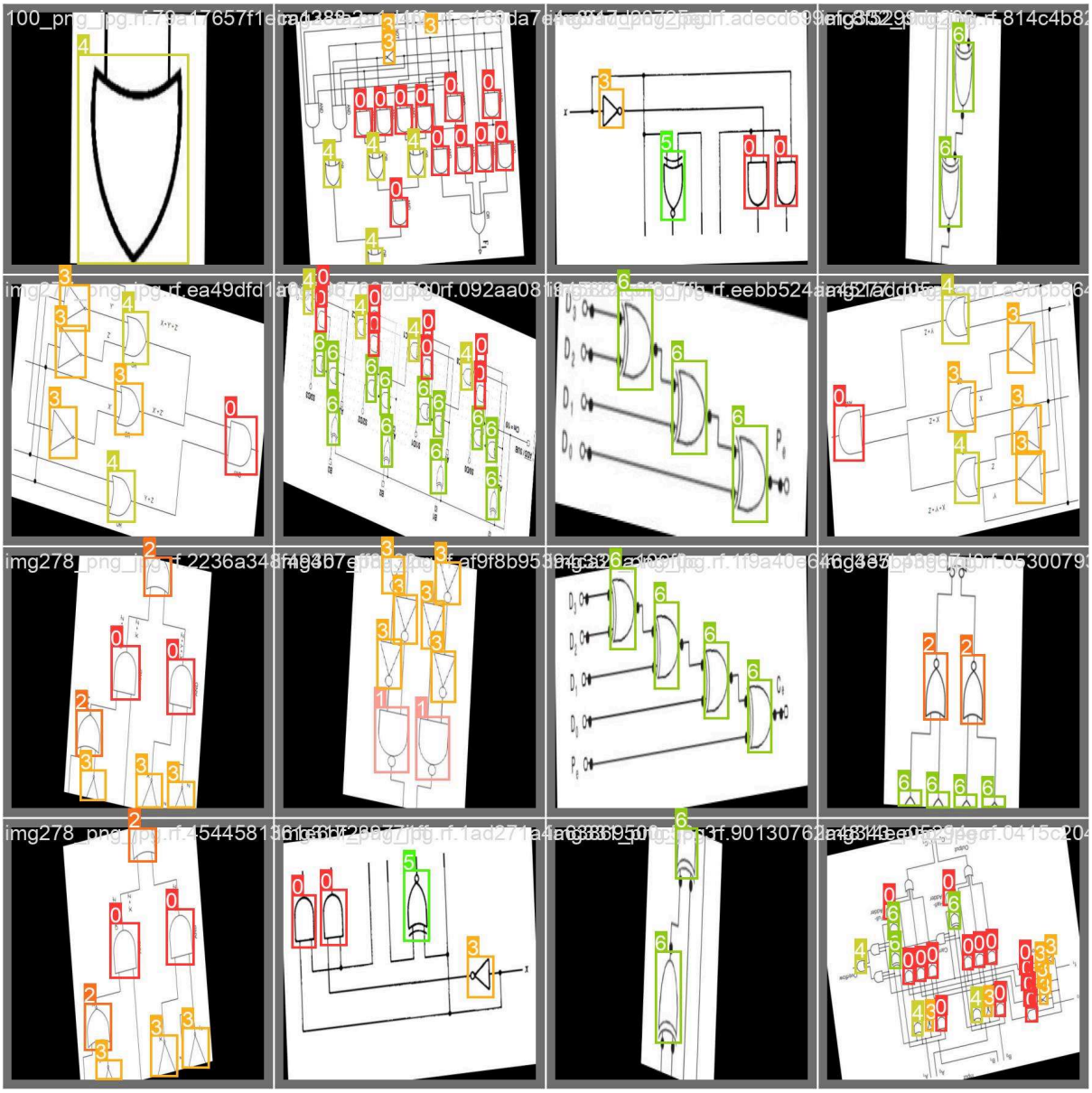


Curious? Visualize Our Training Data with Labels

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

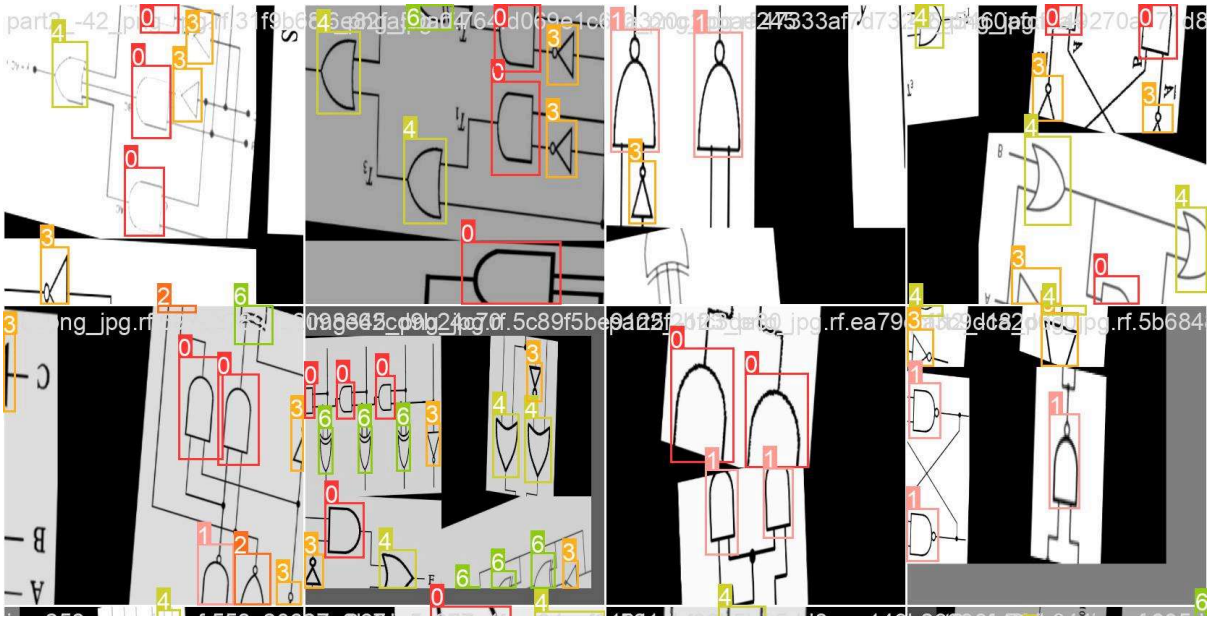
Note a mosaic dataloader is used for training (shown below), a new dataloading concept developed by Glenn Jocher and first featured in [YOLOv4](#).

```
Image(filename='/content/yolov5/runs/train/yolov5s_results/val_batch0_labels.jpg', width=900)
```



```
# print out an augmented training example
print("GROUND TRUTH AUGMENTED TRAINING DATA:")
Image(filename='/content/yolov5/runs/train/yolov5s_results/train_batch0.jpg', width=900)
```

GROUND TRUTH AUGMENTED TRAINING DATA:



Run Inference With Trained Weights

Next, we can run inference with a pretrained checkpoint on all images in the `test/images` folder to understand how our model performs on our test set.

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

```
train/
```

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

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

In the snippet below, replace `Cash-Counter-10` with the name of the folder in which your dataset is stored.

```
%cd /content/yolov5/
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.4 --source /content/and_nor_logic.jpg
```

```
/content/yolov5
detect: weights=['runs/train/yolov5s_results/weights/best.pt'], source=/content/and_nor_logic.jpg, data=data/coco128.yaml, imgsz=[416, 4
YOLOv5 🚀 v7.0-306-gb599ae42 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
custom_YOLOv5s summary: 182 layers, 7262700 parameters, 0 gradients
image 1/1 /content/and_nor_logic.jpg: 192x416 2 0s, 1 4, 1 6, 106.4ms
Speed: 1.1ms pre-process, 106.4ms inference, 510.9ms NMS per image at shape (1, 3, 416, 416)
```