Train YOLO Models in Google Colab

Author: Evan Juras, EJ Technology Consultants

Last updated: January 3, 2025

GitHub: Train and Deploy YOLO Models

Introduction

This notebook uses Ultralytics to train YOLO11, YOLOv8, or YOLOv5 object detection models with a custom dataset. At the end of this Colab, you'll have a custom YOLO model that you can run on your PC, phone, or edge device like the Raspberry Pi.

Custom YOLO candy detection model in action!

I created a YouTube video that walks through this guide step by step. I recommend following along with the video while working through this notebook.

Click here to go to the video!

Important note: This notebook will be continuously updated to make sure it works with newer versions of Ultralytics and YOLO. If you see any differences between the YouTube video and this notebook, always follow the notebook!

Working in Colab

Colab provides a virtual machine in your browser complete with a Linux OS, filesystem, Python environment, and best of all, a free GPU. We'll install PyTorch and Ultralytics in this environment and use it to train our model. Simply click the Play button on sections of code in this notebook to execute them on the virtual machine.

Navigation

To navigate this notebook, use the table of contents in the left sidebar to jump from section to section.

Verify NVIDIA GPU Availability

Make sure you're using a GPU-equipped machine by going to "Runtime" -> "Change runtime type" in the top menu bar, and then selecting one of the GPU options in the Hardware accelerator section. Click Play on the following code block to verify that the NVIDIA GPU is present and ready for training.

```
!nvidia-smi
```

```
Thu Aug 28 16:43:11 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
```

```
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   37C    P8              9W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

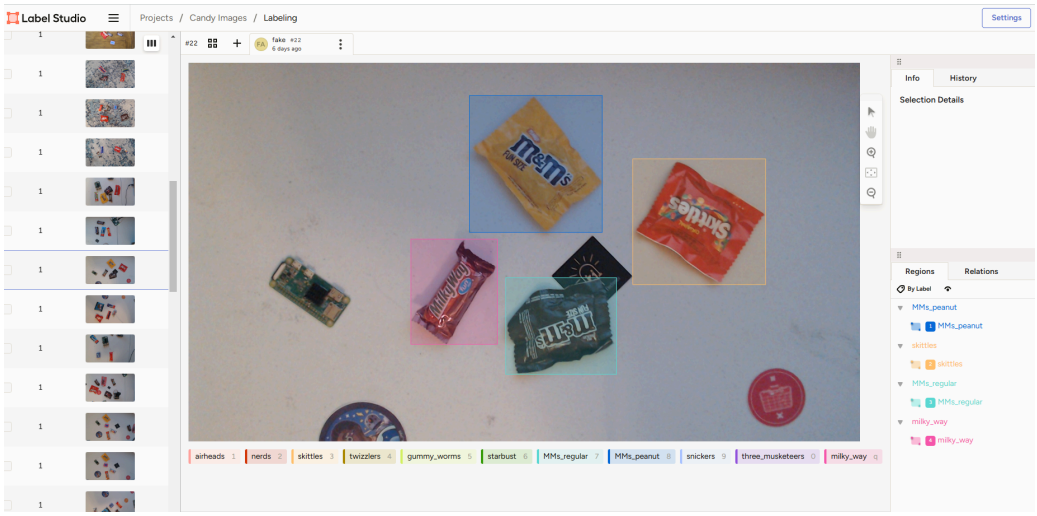
1. Gather and Label Training Images

Before we start training, we need to gather and label images that will be used for training the object detection model. A good starting point for a proof-of-concept model is 200 images. The training images should have random objects in the image along with the desired objects, and should have a variety of backgrounds and lighting conditions.

There are a couple options for gathering images:

- Build a custom dataset by taking your own pictures of the objects and labeling them (this typically results in the best performance)
- Find a pre-made dataset from sources like Roboflow Universe, Kaggle, or Google Images V7

If you want to build your own dataset, there are several tools available for labeling images. One good option is Label Studio, a free and open-source labeling tool that has a simple workflow while providing capabilities for more advanced features. My YouTube video that walks through this notebook (link to be added soon) shows how to label images with Label Studio.

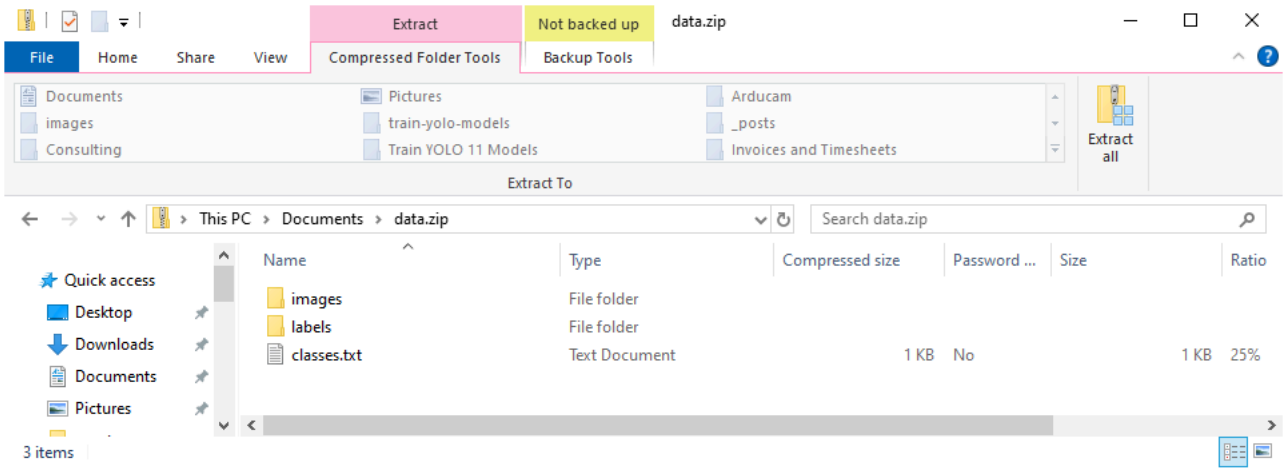


Example of a candy image labeled with Label Studio.

If you used Label Studio to label and export the images, they'll be exported in a `project.zip` file that contains the following:

- An `images` folder containing the images
- A `labels` folder containing the labels in YOLO annotation format
- A `classes.txt` labelmap file that contains all the classes
- A `notes.json` file that contains info specific to Label Studio (this file can be ignored)

If you obtained your dataset from another source (like Roboflow Universe) or used another tool to label your dataset, make sure the files are organized in the same folder structure.



Organize your data in the folders shown here. See my Candy Detection Dataset for an example.

Once you've got your dataset built, put into the file structure shown above, and zipped into `data.zip`, you're ready to move on to the next step.

2. Upload Image Dataset and Prepare Training Data

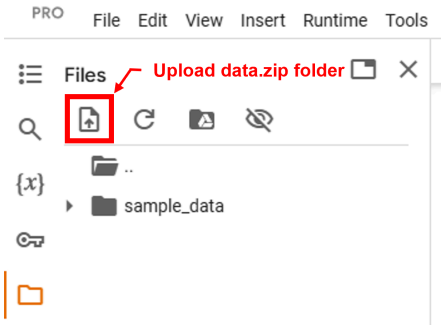
Next, we'll upload our dataset and prepare it for training with YOLO. We'll split the dataset into train and validation folders, and we'll automatically generate the configuration file for training the model.

2.1 Upload images

First, we need to upload the dataset to Colab. Here are a few options for moving the `data.zip` folder into this Colab instance.

Option 1. Upload through Google Colab

Upload the `data.zip` file to the Google Colab instance by clicking the "Files" icon on the left hand side of the browser, and then the "Upload to session storage" icon. Select the zip folder to upload it.



Option 2. Copy from Google Drive

You can also upload your images to your personal Google Drive, mount the drive on this Colab session, and copy them over to the Colab filesystem. This option works well if you want to upload the images beforehand so you don't have to wait for them to upload each time you restart this Colab. If you have more than 50MB worth of images, I recommend using this option.

First, upload the `data.zip` file to your Google Drive, and make note of the folder you uploaded them to. Replace `MyDrive/path/to/data.zip` with the path to your zip file. (For example, I uploaded the zip file to folder called "candy-dataset1", so I would use `MyDrive/candy-dataset1/data.zip` for the path). Then, run the following block of code to mount your Google Drive to this Colab session and copy the folder to this filesystem.

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

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

```
!cp /content/gdrive/MyDrive/path/to/data.zip /content
```

Option 3. Use my candy detection or coin detection dataset

If you just want to test the process on a pre-made dataset, you can use one of my datasets:

- Candy image dataset, which contains 162 pictures of popular candies (Skittles, Snickers, etc)
- Coin image dataset, which contains 750 pictures of US coins (pennies, dimes, nickels, and quarters)

Download one of the datasets by running the following code block. I'll use the candy detection dataset as the example for the rest of the notebook.

```
# To use my one of pre-made dataset instead of your own custom dataset, download it here (control which dataset is downloaded by comment
!wget -O /content/data.zip https://s3.us-west-1.amazonaws.com/evanjuras.com/resources/candy_data_06JAN25.zip # Candy dataset
#!wget -O /content/data.zip https://s3.us-west-1.amazonaws.com/evanjuras.com/resources/YOLO_coin_data_12DEC30.zip # Coin dataset
```

2.2 Split images into train and validation folders

At this point, whether you used Option 1, 2, or 3, you should be able to click the folder icon on the left and see your `data.zip` file in the list of files. Next, we'll unzip `data.zip` and create some folders to hold the images. Run the following code block to unzip the data.

```
# Unzip images to a custom data folder
!unzip -q /content/dataset_tikus_variety.zip -d /content/dataset
```

Ultralytics requires a particular folder structure to store training data for models. Ultralytics requires a particular folder structure to store training data for models. The root folder is named “data”. Inside, there are two main folders:

- **Train**: These are the actual images used to train the model. In one epoch of training, every image in the train set is passed into the neural network. The training algorithm adjusts the network weights to fit the data in the images.
- **Validation**: These images are used to check the model's performance at the end of each training epoch.

In each of these folders is a “images” folder and a “labels” folder, which hold the image files and annotation files respectively.

I wrote a Python script that will automatically create the required folder structure and randomly move 90% of dataset to the "train" folder and 10% to the "validation" folder. Run the following code block to download and execute the scrpt.

```
!wget -O /content/train_val_split.py https://raw.githubusercontent.com/EdjeElectronics/Train-and-Deploy-YOLO-Models/refs/heads/main/uti

# TO DO: Improve robustness of train_val_split.py script so it can handle nested data folders, etc
!python train_val_split.py --datapath="/content/custom_data" --train_pct=0.9
```

3. Install Requirements (Ultralytics)

Next, we'll install the Ultralytics library in this Google Colab instance. This Python library will be used to train the YOLO model.

```
!pip install ultralytics
```

```
Collecting ultralytics
  Downloading ultralytics-8.3.189-py3-none-any.whl.metadata (37 kB)
Requirement already satisfied: numpy>=1.23.0 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (2.0.2)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (3.10.0)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (4.12.0.88)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (11.3.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (6.0.2)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (2.32.4)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (1.16.1)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (2.8.0+cu126)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.12/dist-packages (from ultralytics) (0.23.0+cu126)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.12/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: polars in /usr/local/lib/python3.12/dist-packages (from ultralytics) (1.25.2)
Collecting ultralytics-thop>=2.0.0 (from ultralytics)
  Downloading ultralytics_thop-2.0.16-py3-none-any.whl.metadata (14 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.3.0->ultralytics) (
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.3.0->ultralytics) (0.12
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.3.0->ultralytics)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.3.0->ultralytics)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.3.0->ultralytics) (2
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.3.0->ultralytics) (
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.3.0->ultralytic
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.23.0->ultraly
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.23.0->ultralytics) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.23.0->ultralytics)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.23.0->ultralytics)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytics) (3.19.1)
```

```
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytic
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytics) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytics) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytics) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytics) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytics) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultr
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ul
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultr
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultraly
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultraly
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralyt
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultral
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultra
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultra
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultral
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytics
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytic
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultra
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultraly
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch>=1.8.0->ultralytics) (3.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib>=3.3.0->
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch>=1.8.0->ul
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>=1.8.0->ultralytics)
Downloading ultralytics-8.3.189-py3-none-any.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 62.5 MB/s eta 0:00:00
Downloading ultralytics_thop-2.0.16-py3-none-any.whl (28 kB)
Installing collected packages: ultralytics-thop, ultralytics
Successfully installed ultralytics-8.3.189 ultralytics-thop-2.0.16
```

4. Configure Training

There's one last step before we can run training: we need to create the Ultralytics training configuration YAML file. This file specifies the location of your train and validation data, and it also defines the model's classes. An example configuration file model is available here.

Run the code block below to automatically generate a `data.yaml` configuration file. Make sure you have a labelmap file located at `custom_data/classes.txt`. If you used Label Studio or one of my pre-made datasets, it should already be present. If you assembled the dataset another way, you may have to manually create the `classes.txt` file (see here for an example of how it's formatted).

```
# Python function to automatically create data.yaml config file
# 1. Reads "classes.txt" file to get list of class names
# 2. Creates data dictionary with correct paths to folders, number of classes, and names of classes
# 3. Writes data in YAML format to data.yaml

import yaml
import os

# def create_data_yaml(path_to_classes_txt, path_to_data_yaml):

#   # Read class.txt to get class names
#   if not os.path.exists(path_to_classes_txt):
#     print(f'classes.txt file not found! Please create a classes.txt labelmap and move it to {path_to_classes_txt}')
#     return
#   with open(path_to_classes_txt, 'r') as f:
#     classes = []
#     for line in f.readlines():
#       if len(line.strip()) == 0: continue
#       classes.append(line.strip())
#   number_of_classes = len(classes)

#   # Create data dictionary
#   data = {
#       'path': '/content/data',
#       'train': 'train/images',
#       'val': 'validation/images',
#       'nc': number_of_classes,
#       'names': classes
#   }

#   # Write data to YAML file
#   with open(path_to_data_yaml, 'w') as f:
#     yaml.dump(data, f, sort_keys=False)
#   print(f'Created config file at {path_to_data_yaml}')

#   return

# Define path to classes.txt and run function
# path_to_classes_txt = '/content/custom_data/classes.txt'
# path_to_data_yaml = '/content/data.yaml'
```

```
# create_data_yaml(path_to_classes_txt, path_to_data_yaml)

print('\nFile contents:\n')
!cat /content/dataset/data.yaml
```

```
File contents:

train: ../train/images
val: ../valid/images
test: ../test/images

nc: 3
names: ['Tikus', 'pendeteksi tikus rumah - v2 2023-08-28 4-13am', 'tikus']

roboflow:
  workspace: tikus-5t8ne
  project: tikus-custom-variety-grmnt
  version: 1
  license: CC BY 4.0
  url: https://universe.roboflow.com/tikus-5t8ne/tikus-custom-variety-grmnt/dataset/1
```

5. Train Model

5.1 Training Parameters

Now that the data is organized and the config file is created, we're ready to start training! First, there are a few important parameters to decide on. Visit my article on Training YOLO Models Locally to learn more about these parameters and how to choose them.

Model architecture & size (`model`):

There are several YOLO11 models sizes available to train, including `yolo11n.pt` , `yolo11s.pt` , `yolo11m.pt` , `yolo11l.pt` , and `yolo11xl.pt` . Larger models run slower but have higher accuracy, while smaller models run faster but have lower accuracy. I made a brief YouTube video that compares performance of different YOLO models on a Raspberry Pi 5 and a laptop with a RTX 4050 GPU, check it out here to get a sense of their speed accuracy. If you aren't sure which model size to use, `yolo11s.pt` is a good starting point.

You can also train YOLOv8 or YOLOv5 models by substituting `yolo11` for `yolov8` or `yolov5` .

Number of epochs (`epochs`)

In machine learning, one “epoch” is one single pass through the full training dataset. Setting the number of epochs dictates how long the model will train for. The best amount of epochs to use depends on the size of the dataset and the model architecture. If your dataset has less than 200 images, a good starting point is 60 epochs. If your dataset has more than 200 images, a good starting point is 40 epochs.

Resolution (`imgsz`)

Resolution has a large impact on the speed and accuracy of the model: a lower resolution model will have higher speed but less accuracy. YOLO models are typically trained and inferenced at a 640x640 resolution. However, if you want your model to run faster or know you will be working with low-resolution images, try using a lower resolution like 480x480.

5.2 Run Training!

Run the following code block to begin training. If you want to use a different model, number of epochs, or resolution, change `model` , `epochs` , or `imgsz` .

```
!yolo detect train data=/content/dataset/data.yaml model=yolo11s.pt epochs=60 imgsz=640
```

```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.ultralytics.c
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11s.pt to 'yolo11s.pt': 100% ━━━━━━━━━━━━ 18.4/18.4
Ultralytics 8.3.189 🚀 Python-3.12.11 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=16, bgr=0.0, box=7.5, cache=False, cf
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf': 100% ━━━━━━━━━━━━ 755.1/755.1KB 19.
Overriding model.yaml nc=80 with nc=3

                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     26080  ultralytics.nn.modules.block.C3k2            [64, 128, 1, False, 0.25]
  3                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
  4                  -1  1    103360  ultralytics.nn.modules.block.C3k2            [128, 256, 1, False, 0.25]
  5                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
  6                  -1  1    346112  ultralytics.nn.modules.block.C3k2            [256, 256, 1, True]
  7                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                  -1  1   1380352  ultralytics.nn.modules.block.C3k2            [512, 512, 1, True]
```

```
 9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
10                  -1  1    990976  ultralytics.nn.modules.block.C2PSA           [512, 512, 1]
11                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
12             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
13                  -1  1    443776  ultralytics.nn.modules.block.C3k2            [768, 256, 1, False]
14                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
15             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
16                  -1  1    127680  ultralytics.nn.modules.block.C3k2            [512, 128, 1, False]
17                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
18            [-1, 13]  1         0  ultralytics.nn.modules.conv.Concat           [1]
19                  -1  1    345472  ultralytics.nn.modules.block.C3k2            [384, 256, 1, False]
20                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
21            [-1, 10]  1         0  ultralytics.nn.modules.conv.Concat           [1]
22                  -1  1   1511424  ultralytics.nn.modules.block.C3k2            [768, 512, 1, True]
23        [16, 19, 22]  1    820569  ultralytics.nn.modules.head.Detect           [3, [128, 256, 512]]
YOLO11s summary: 181 layers, 9,428,953 parameters, 9,428,937 gradients, 21.6 GFLOPs

Transferred 493/499 items from pretrained weights
Freezing layer 'model.23.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks...
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n.pt to 'yolo11n.pt': 100% ━━━━━━━━━━━━ 5.4/5.4MB
AMP: checks passed ✅
train: Fast image access ✅ (ping: 0.0±0.0 ms, read: 1887.4±1061.1 MB/s, size: 313.0 KB)
train: Scanning /content/dataset/train/labels... 1025 images, 0 backgrounds, 0 corrupt: 100% ━━━━━━━━━━━━ 1025/1025 2377.3it/s 0.4
train: New cache created: /content/dataset/train/labels.cache
WARNING ⚠️ Box and segment counts should be equal, but got len(segments) = 2, len(boxes) = 1602. To resolve this only boxes will b
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_average',
val: Fast image access ✅ (ping: 0.0±0.0 ms, read: 436.1±399.0 MB/s, size: 32.6 KB)
val: Scanning /content/dataset/valid/labels... 292 images, 0 backgrounds, 0 corrupt: 100% ━━━━━━━━━━━━ 292/292 1110.6it/s 0.3s
val: New cache created: /content/dataset/valid/labels.cache
Plotting labels to runs/detect/train/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum'
optimizer: AdamW(lr=0.001429, momentum=0.9) with parameter groups 81 weight(decay=0.0), 88 weight(decay=0.0005), 87 bias(decay=0.0
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/detect/train
Starting training for 60 epochs...
```

The training algorithm will parse the images in the training and validation directories and then start training the model. At the end of each training epoch, the program runs the model on the validation dataset and reports the resulting mAP, precision, and recall. As training continues, the mAP should generally increase with each epoch. Training will end once it goes through the number of epochs specified by `epochs`.

> **NOTE:** Make sure to allow training to run to completion, because an optimizer runs at the end of training that strips out unneeded layers from the model.

The best trained model weights will be saved in `content/runs/detect/train/weights/best.pt`. Additional information about training is saved in the `content/runs/detect/train` folder, including a `results.png` file that shows how loss, precision, recall, and mAP progressed over each epoch.

6. Test Model

The model has been trained; now it's time to test it! The commands below run the model on the images in the validation folder and then display the results for the first 10 images. This is a good way to confirm your model is working as expected. Click Play on the blocks below to see how your model performs.

```
!yolo detect predict model=runs/detect/train/weights/best.pt source=dataset/valid/images save=True
```

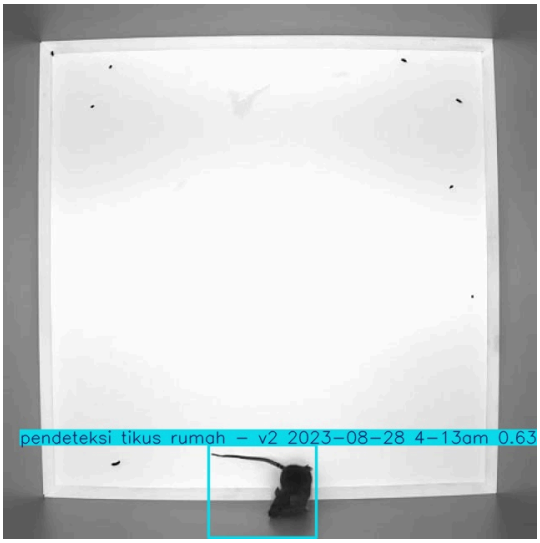
```
Ultralytics 8.3.189 🚀 Python-3.12.11 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
YOLO11s summary (fused): 100 layers, 9,413,961 parameters, 0 gradients, 21.3 GFLOPs

image 1/292 /content/dataset/valid/images/101_jpg.rf.cf0c705313e558d7bd34930a2234c425.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 2/292 /content/dataset/valid/images/101_jpg.rf.e3eb0cafba6e7d09a319015de4cb815d.jpg: 480x640 3 tikuss, 89.8ms
image 3/292 /content/dataset/valid/images/109_jpg.rf.e6326089f5cbe37f877ce0300d5febc0.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 4/292 /content/dataset/valid/images/10_jpg.rf.6337786100f38eb800d63c3288bb326c.jpg: 640x640 4 pendeteksi tikus rumah - v2 20
image 5/292 /content/dataset/valid/images/110_jpg.rf.1bb40f27c2d5db3f453c8654f58eddb4.jpg: 640x640 2 pendeteksi tikus rumah - v2 2
image 6/292 /content/dataset/valid/images/112_jpg.rf.233993af914b01cf2ccd1efd4224db31.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 7/292 /content/dataset/valid/images/116_jpg.rf.15cc1cbd6f127beb1b9ba7e4fe69a860.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 8/292 /content/dataset/valid/images/11_jpg.rf.3cc26490a72a007efe777d110c1e0b64.jpg: 640x640 3 pendeteksi tikus rumah - v2 20
image 9/292 /content/dataset/valid/images/11_png_jpg.rf.4ef7f1f5893302e5ab23eb99051058e6.jpg: 640x640 1 pendeteksi tikus rumah - v
image 10/292 /content/dataset/valid/images/120_jpg.rf.3045d8a94bb8cb67ba1db24e85df2638.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 11/292 /content/dataset/valid/images/121_jpg.rf.988c759cb44532632a4a388fb08887f9.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 12/292 /content/dataset/valid/images/122_jpg.rf.7e2a9d7ed588899013994e8ad243e753.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 13/292 /content/dataset/valid/images/123_jpg.rf.32795783bfb0919025ede5a350a9b93f.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 14/292 /content/dataset/valid/images/123_jpg.rf.6ecd96b673831a1ee8add0a052c6edc9.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 15/292 /content/dataset/valid/images/125_jpg.rf.14ace17f059e109034c88e035455ffeb.jpg: 480x640 3 tikuss, 11.5ms
image 16/292 /content/dataset/valid/images/127_jpg.rf.b32bcee8f42cb2aba8af01abf713b02f.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 17/292 /content/dataset/valid/images/129_jpg.rf.1c719c2da73784ec0282a7f94fe53b5f.jpg: 480x640 3 tikuss, 12.8ms
image 18/292 /content/dataset/valid/images/12_jpg.rf.1c68c0cd815a2b4107951f352e93ec76.jpg: 640x640 3 pendeteksi tikus rumah - v2 2
```

```
image 19/292 /content/dataset/valid/images/130_jpg.rf.3260ec0fee2d1d8f68bf285873390a4d.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 20/292 /content/dataset/valid/images/131_jpg.rf.57a11d1d3adb609f42603d578fd48191.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 21/292 /content/dataset/valid/images/132_jpg.rf.03157e609a419e7efbf9bb90eb08e374.jpg: 640x640 3 pendeteksi tikus rumah - v2
image 22/292 /content/dataset/valid/images/132_jpg.rf.c42d33a085c8b1ffbc387a55b4da6ad8.jpg: 640x640 2 pendeteksi tikus rumah - v2
image 23/292 /content/dataset/valid/images/133_jpg.rf.cec4f6bf7909edb61068fe72a02fe101.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 24/292 /content/dataset/valid/images/13_jpg.rf.9d134b81e83be1b9270fbad8ff29b3ba.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 25/292 /content/dataset/valid/images/13_jpg.rf.f5d382cc0f0656c6fd6c4945809560c9.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 26/292 /content/dataset/valid/images/13_png_jpg.rf.773febadf57034468db1537801b35cd6.jpg: 640x640 (no detections), 10.9ms
image 27/292 /content/dataset/valid/images/141_jpg.rf.d08e1540945602960d7f82d09741117b.jpg: 640x640 7 pendeteksi tikus rumah - v2
image 28/292 /content/dataset/valid/images/141_jpg.rf.e32abb061a1972d73d5b1e5bef25496a.jpg: 640x640 7 pendeteksi tikus rumah - v2
image 29/292 /content/dataset/valid/images/144_jpg.rf.48a7687896b664355e96ea24c3788ddc.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 30/292 /content/dataset/valid/images/144_jpg.rf.6e38b8ddd84f9befb2d2f75815e0c9ef.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 31/292 /content/dataset/valid/images/148_jpg.rf.c6d77b0531ef747e74fb6cd5f7b229d9.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 32/292 /content/dataset/valid/images/14_jpg.rf.0c6cfd74a984b5895ffd8ef6d6e9c8dc.jpg: 640x640 2 pendeteksi tikus rumah - v2 2
image 33/292 /content/dataset/valid/images/14_jpg.rf.d8b1d82a5856a887d472e677eeab9bca.jpg: 640x640 4 pendeteksi tikus rumah - v2 2
image 34/292 /content/dataset/valid/images/14_jpg.rf.db0576fa462f73e1e17e54a253b42e90.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 35/292 /content/dataset/valid/images/159_jpg.rf.6388417543e77b45f1f4eae35183c7d2.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 36/292 /content/dataset/valid/images/159_jpg.rf.9891fd2b1c19ba0ab90cafccf7b3fae1.jpg: 480x640 3 tikuss, 9.1ms
image 37/292 /content/dataset/valid/images/15_jpg.rf.a4700a1458985365bcfba6d7fd3a9973.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 38/292 /content/dataset/valid/images/162_jpg.rf.056710fb83c517fcbf31bad5ec4309d9.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 39/292 /content/dataset/valid/images/163_jpg.rf.1e3de6c3a893048f542af1fde7918049.jpg: 480x640 3 tikuss, 8.4ms
image 40/292 /content/dataset/valid/images/165_jpg.rf.434144d4421821b28ba9235f6dc9d2de.jpg: 480x640 3 tikuss, 7.8ms
image 41/292 /content/dataset/valid/images/1678376942378-Screenshot_20230309-224648_jpg.rf.25565243b9f5bfa08367852ef1831954.jpg: 3
image 42/292 /content/dataset/valid/images/1678376942378-Screenshot_20230309-224648_jpg.rf.f4846833e009e11c37efb9d1dfecc534.jpg: 3
image 43/292 /content/dataset/valid/images/169_jpg.rf.472cc3699f1c2085b86491cda919a945.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 44/292 /content/dataset/valid/images/16_jpg.rf.e30063b599f3ed1e72242c965774f0bb.jpg: 640x640 1 pendeteksi tikus rumah - v2 2
image 45/292 /content/dataset/valid/images/171_jpg.rf.b032e6516dfd6e31448842abfa828a8a.jpg: 480x640 2 tikuss, 9.8ms
image 46/292 /content/dataset/valid/images/171_jpg.rf.b10ae75898b62f853e3a3cbfe8e6a9f0.jpg: 640x640 2 pendeteksi tikus rumah - v2
image 47/292 /content/dataset/valid/images/171_jpg.rf.c75a1e028220b1cb1b900d2e6d0d92d2.jpg: 640x640 1 pendeteksi tikus rumah - v2
image 48/292 /content/dataset/valid/images/1753231932794_jpg.rf.668b88d5732af32861697d743c1deb09.jpg: 640x480 1 Tikus, 51.2ms
image 49/292 /content/dataset/valid/images/1753231932876_jpg.rf.a114747e2badaeac0a1d42d0dfc3b8b2.jpg: 640x480 1 Tikus, 8.0ms
image 50/292 /content/dataset/valid/images/1753231933075_jpg.rf.261f69244dbc5b6173e908482f7f7c19.jpg: 640x480 1 Tikus, 1 pendeteks
image 51/292 /content/dataset/valid/images/1753231933163_jpg.rf.cb2403cd2d58ae00d30128c6ae1360f6.jpg: 480x640 1 Tikus, 8.4ms
image 52/292 /content/dataset/valid/images/1753231933453_jpg.rf.81553df4006dd06c740d84567255b7a4.jpg: 640x480 1 Tikus, 8.4ms
image 53/292 /content/dataset/valid/images/1753231933491_jpg.rf.768c26bbd6d78f7c1a54a5964da800fe.jpg: 640x480 (no detections), 9.9
image 54/292 /content/dataset/valid/images/1753231933685_jpg.rf.79da806f2e916be63dbf4fd29fbf32ab.jpg: 640x480 1 Tikus, 12.4ms
```

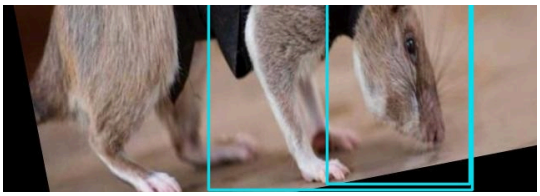
```
import glob
from IPython.display import Image, display
for image_path in glob.glob(f'/content/runs/detect/predict/*.jpg')[:10]:
  display(Image(filename=image_path, height=400))
  print('\n')
```

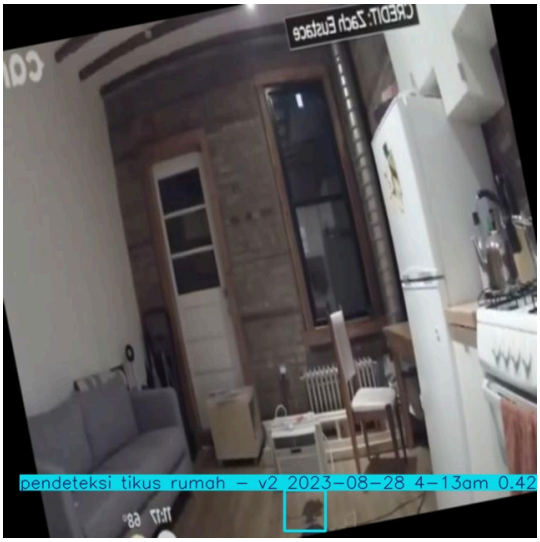

tikus 0.90

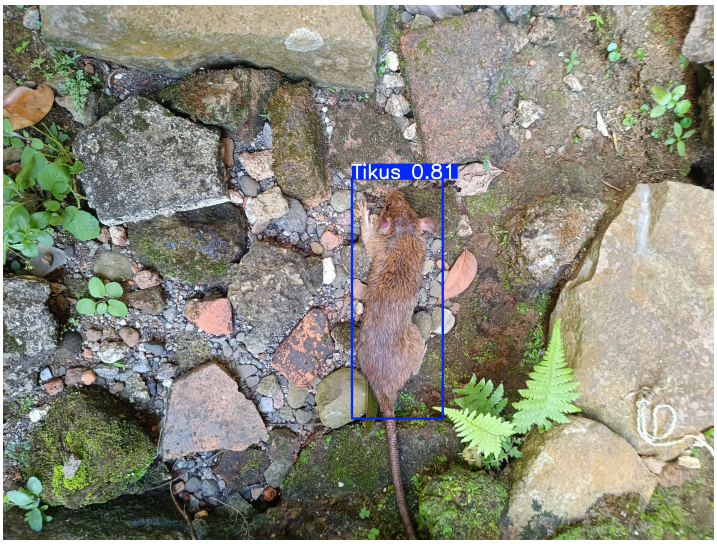

pendeteksi tikus rumah – v2 2023-08-28 4-13am 0.82


pendeteksi tikus rumah – v2 2023-08-28 4-13am 0.63


pendeteksi tikus rumah – v2 2023-08-28 4-13am 0.50

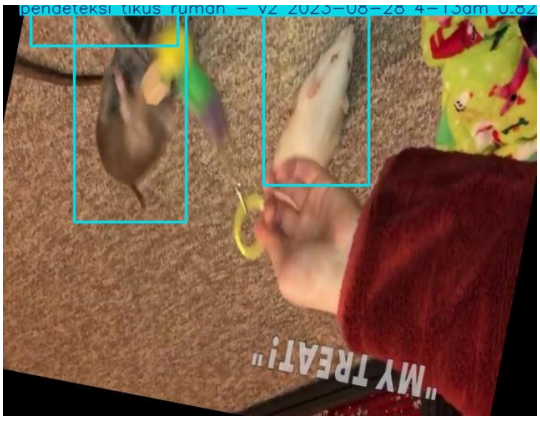



pendeteksi tikus rumah – v2 2023-08-28 4-13am 0.42


Tikus 0.81


Tikus 0.90


pendeteksi tikus rumah – v2 2023-08-28 4-13am 0.82


"MY TREAT!"


RACUN PEMBASMI TIKUS
AMPUH !!
猫王
1号
NO.1
PROMO
10PCS


Rat Training
Special Guest
Esther Minic-Rosenthal

The model should draw a box around each object of interest in each image. If it isn't doing a good job of detecting objects, here are a few tips:

1. Double-check your dataset to make sure there are no labeling errors or conflicting examples.
2. Increase the number of epochs used for training.
3. Use a larger model size (e.g. `yolo11l.pt`).
4. Add more images to the training dataset. See my dataset video for tips on how to capture good training images and improve accuracy.

You can also run the model on video files or other images images by uploading them to this notebook and using the above `!yolo detect predict` command, where `source` points to the location of the video file, image, or folder of images. The results will be saved in `runs/detect/predict`.

Drawing boxes on images is great, but it isn't very useful in itself. It's also not very helpful to just run this models inside a Colab notebook: it's easier if we can just run it on a local computer. Continue to the next section to see how to download your newly trained model and run it on a local device.

7. Deploy Model

Now that your custom model has been trained, it's ready to be downloaded and deployed in an application! YOLO models can run on a wide variety of hardware, including PCs, embedded systems, and phones. Ultralytics makes it easy to convert the YOLO models to various formats (`tflite`, `onnx`, etc.) and deploy them in a variety of environments.

This section shows how to download the model and provides links to instructions for deploying it on your PC and edge devices like the Raspberry Pi.

7.1 Download YOLO Model

First, zip and download the trained model by running the code blocks below.

The code creates a folder named `my_model`, moves the model weights into it, and renames them from `best.pt` to `my_model.pt`. It also adds the training results in case you want to reference them later. It then zips the folder as `my_model.zip`.

```
# Create "my_model" folder to store model weights and train results
!mkdir /content/my_model
!cp /content/runs/detect/train/weights/best.pt /content/my_model/my_model.pt
!cp -r /content/runs/detect/train /content/my_model

# Zip into "my_model.zip"
%cd my_model
!zip /content/my_model.zip my_model.pt
!zip -r /content/my_model.zip train
%cd /content
```

```
/content/my_model
  adding: my_model.pt (deflated 8%)
  adding: train/ (stored 0%)
  adding: train/train_batch3250.jpg (deflated 9%)
  adding: train/val_batch2_labels.jpg (deflated 10%)
  adding: train/weights/ (stored 0%)
  adding: train/weights/best.pt (deflated 8%)
  adding: train/weights/last.pt (deflated 8%)
  adding: train/labels.jpg (deflated 29%)
  adding: train/train_batch3252.jpg (deflated 8%)
  adding: train/train_batch0.jpg (deflated 5%)
  adding: train/val_batch0_pred.jpg (deflated 13%)
  adding: train/BoxR_curve.png (deflated 12%)
  adding: train/confusion_matrix_normalized.png (deflated 29%)
  adding: train/results.csv (deflated 61%)
  adding: train/confusion_matrix.png (deflated 31%)
  adding: train/train_batch3251.jpg (deflated 9%)
  adding: train/BoxP_curve.png (deflated 14%)
  adding: train/val_batch2_pred.jpg (deflated 9%)
  adding: train/BoxPR_curve.png (deflated 17%)
  adding: train/train_batch2.jpg (deflated 4%)
  adding: train/BoxF1_curve.png (deflated 11%)
  adding: train/args.yaml (deflated 53%)
  adding: train/val_batch1_pred.jpg (deflated 7%)
  adding: train/val_batch1_labels.jpg (deflated 7%)
  adding: train/results.png (deflated 7%)
  adding: train/val_batch0_labels.jpg (deflated 13%)
  adding: train/train_batch1.jpg (deflated 4%)
/content
```

```
# This takes forever for some reason, you can also just download the model from the sidebar
from google.colab import files
```