BuzzTech: Machine Learning at the Edge

Deploying YOLOv8 on Raspberry Pi Zero 2W for Real-Time Bee Counting at the Hive Entrance."

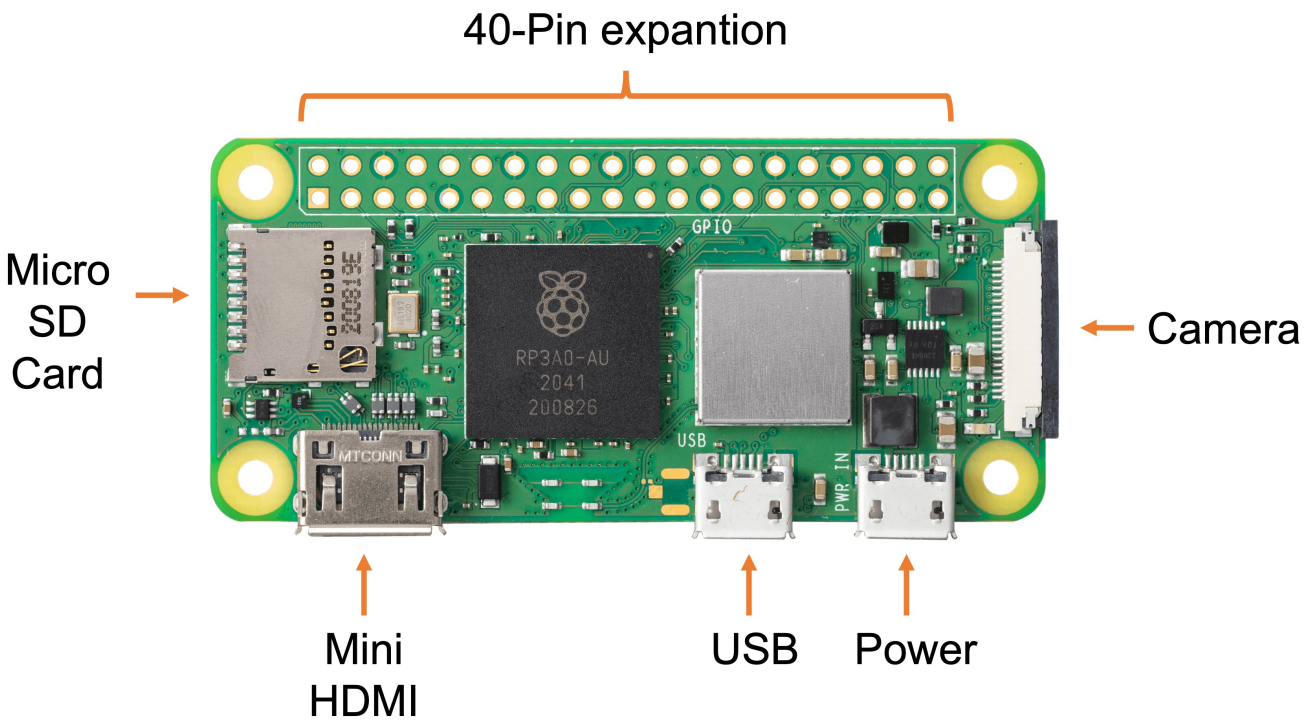
Introduction

At the Federal University of Itajuba in Brazil, with the master's student José Anderson Reis and Professor José Alberto Ferreira Filho, we are exploring a project that delves into the intersection of technology and nature. This tutorial will review our first steps and share our observations on deploying YOLOv8, a cutting-edge machine learning model, on the compact and efficient Raspberry Pi Zero 2W (*Raspi-Zero*). We aim to estimate the number of bees entering and exiting their hive—a task crucial for beekeeping and ecological studies.

Why is this important? Bee populations are vital indicators of environmental health, and their monitoring can provide essential data for ecological research and conservation efforts. However, manual counting is labor-intensive and prone to errors. By leveraging the power of embedded machine learning, or tinyML, we automate this process, enhancing accuracy and efficiency.

This tutorial will cover setting up the Raspberry Pi, integrating a camera module, optimizing and deploying YOLOv8 for real-time image processing, and analyzing the data gathered.

Hardware Setup



The Raspberry Pi Zero 2W has a 1GHz quad-core 64-bit Arm Cortex-A53 CPU and 512MB of SDRAM, making it suitable for various applications. It includes 2.4GHz 802.11 b/g/n wireless LAN and Bluetooth 4.2, Bluetooth Low Energy (BLE), and an onboard antenna for enhanced connectivity. A Mini HDMI® port supports visual outputs, and data connections can be managed through a micro USB On-The-Go (OTG) port. Storage is handled via a microSD card slot, including a CSI-2 camera connector for imaging projects. The board also features an unpopulated HAT-compatible 40-pin header footprint for expansions. It supports H.264 and MPEG-4 decoding, H.264 encoding at 1080p30, and has OpenGL ES 1.1, 2.0 graphics capabilities.

Power is supplied through a micro USB port, which offers composite video and reset pins via solder test points. The device's compact dimensions are 65mm x 30mm, making it highly versatile for embedded applications.

Power Supply

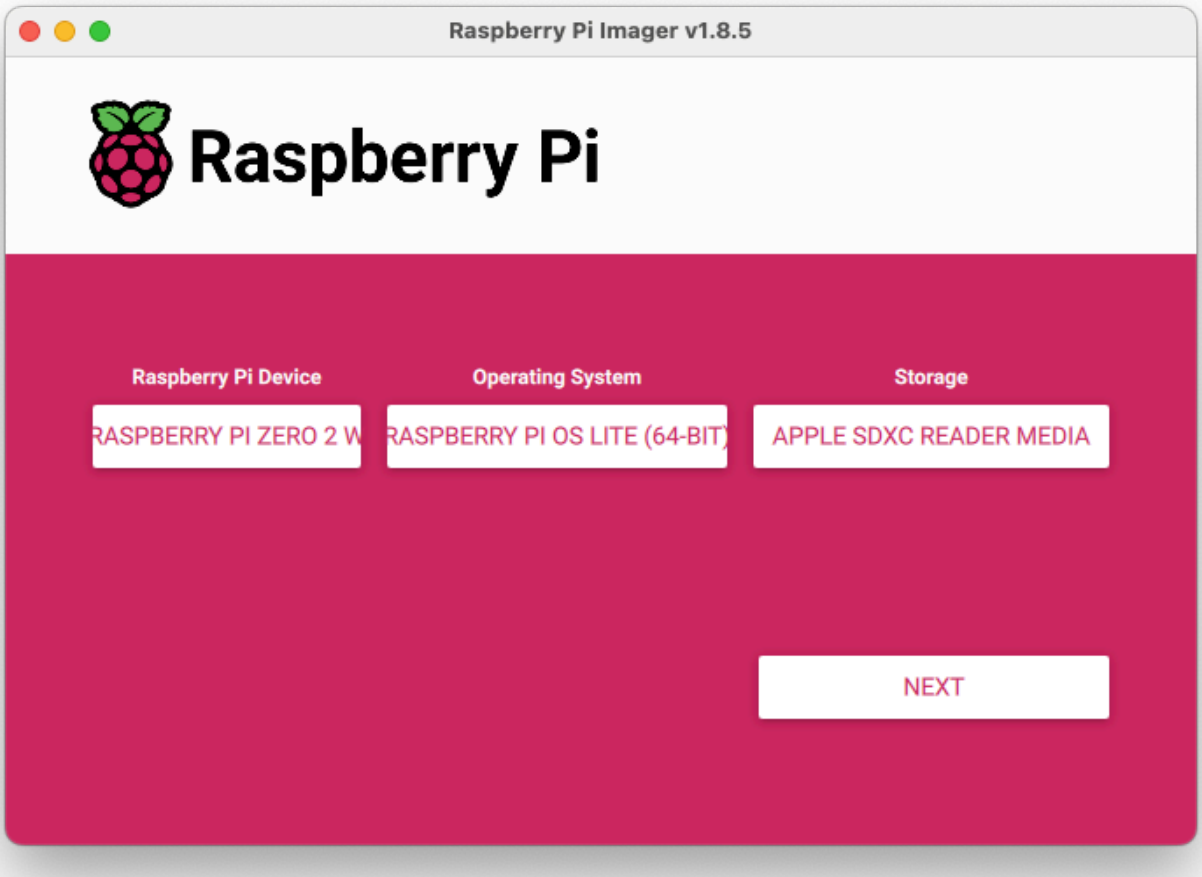
A 5v/2.5A power supply is recommended for the power station, but for our final project, a smaller solar station should be designed for use in the field.

Install the Operating System

To use your Raspberry Pi, you'll need an operating system. By default, Raspberry Pis checks for an operating system on any SD card inserted in the slot, so we should install an operating system using Raspberry Pi Imager.

Raspberry Pi Imager is a tool for downloading and writing images on macOS, Windows, and Linux. It includes many popular operating system images for Raspberry Pi. We will also use the Imager to preconfigure credentials and remote access settings.

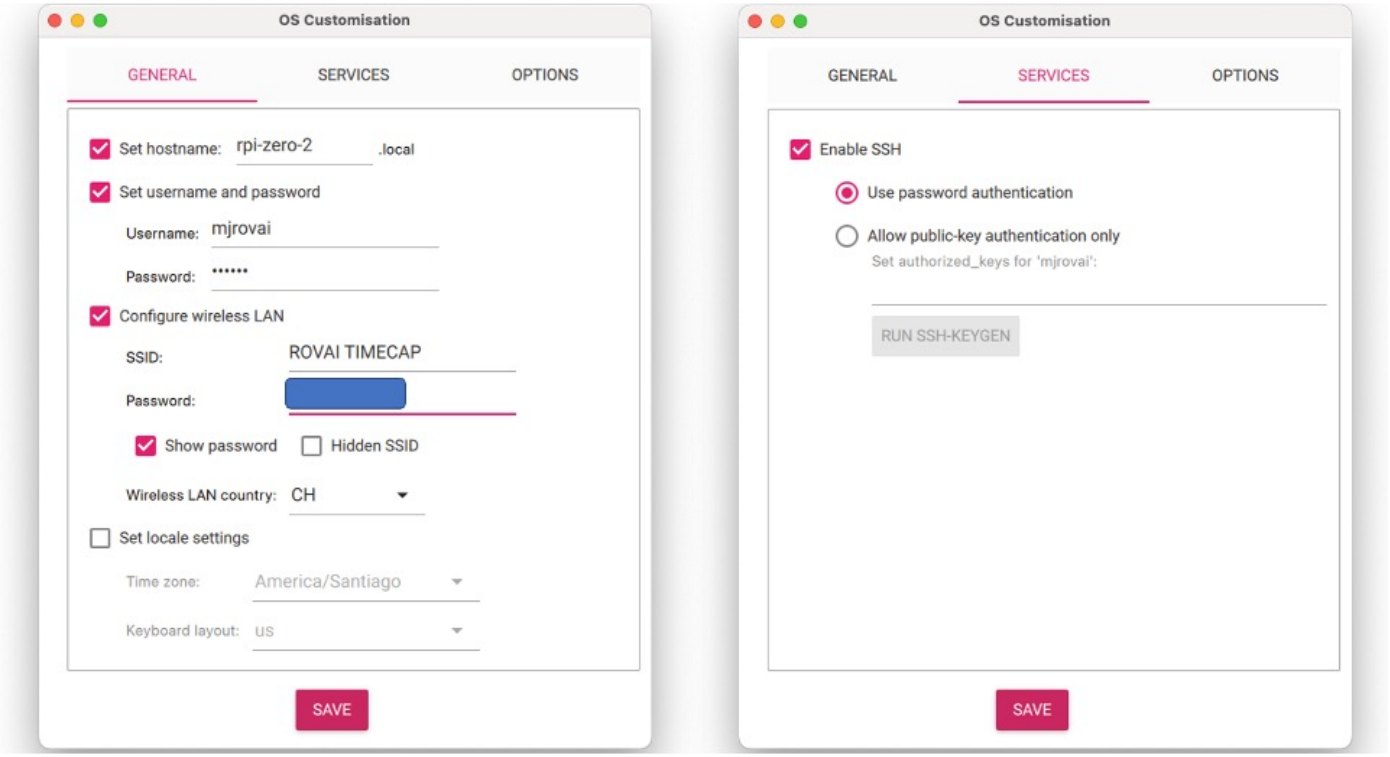
After downloading the Imager and installing it on your computer, use a new and empty SD card. Select the device (**RASPBERRY PI ZERO 2 W)**, the Operating System (**RASPBERRY PI OS LITE (64-BIT)**), and your Storage Device:



> Due to its reduced SDRAM (512MB), the recommended OS for the Rasp Zero is the 32-bit version. However, to run YOLOv8 from Ultralitics, we should use the 64-bit version. Also, the LITE version (no Desktop) is selected to reduce the amount of RAM needed for regular operation.

We should also define the options, such as the hostname, username, password, LAN configuration (on GENERAL TAB), and, more importantly, SSH Enable on the SERVICES tab.

> You can access the terminal of a Raspberry Pi remotely from another computer on the same network using the **S**ecure **SH**ell (SSH) protocol.



After burning the OS to the SD card, install it in the Raspi-zero's SD slot and plug in the 5V power source.

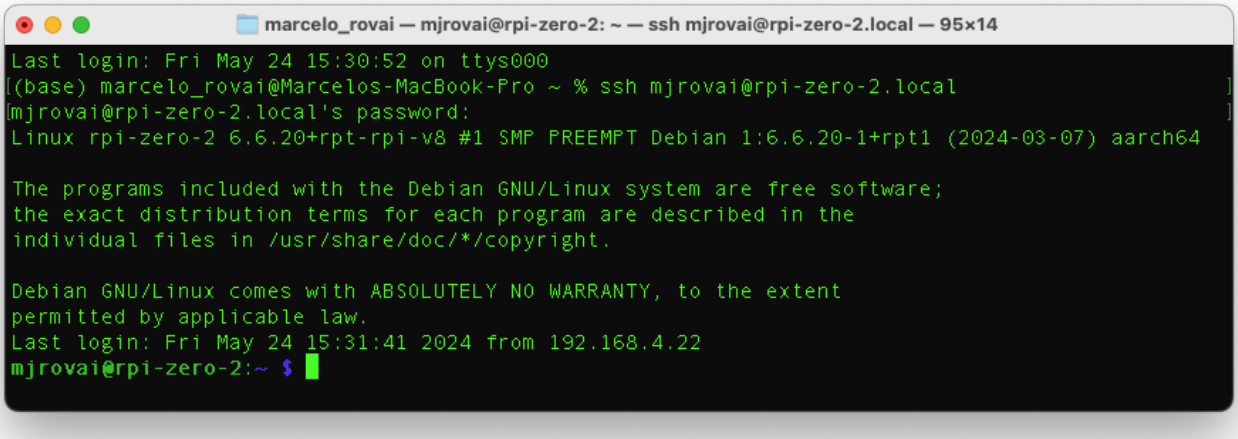
Interacting with the Rasp-Zero via SSH

The easiest way to interact with the Rasp-Zero is via SSH ("Headless"). You can use a Terminal (MAC/Linux) or PuTTy (Windows).

On terminal type:

```
ssh mjrovai@rpi-zero-2.local
```

> You should replace `mjrovai` with your *username* and `rpi-zero-2` with the *hostname* chosen during set-up.



When you see the prompt:

```
mjrovai@rpi-zero-2:~ $
```

It means that you interacting remotely with your Rasp-Zero.

It is a good practice to update the system regularly. For that, you should run:

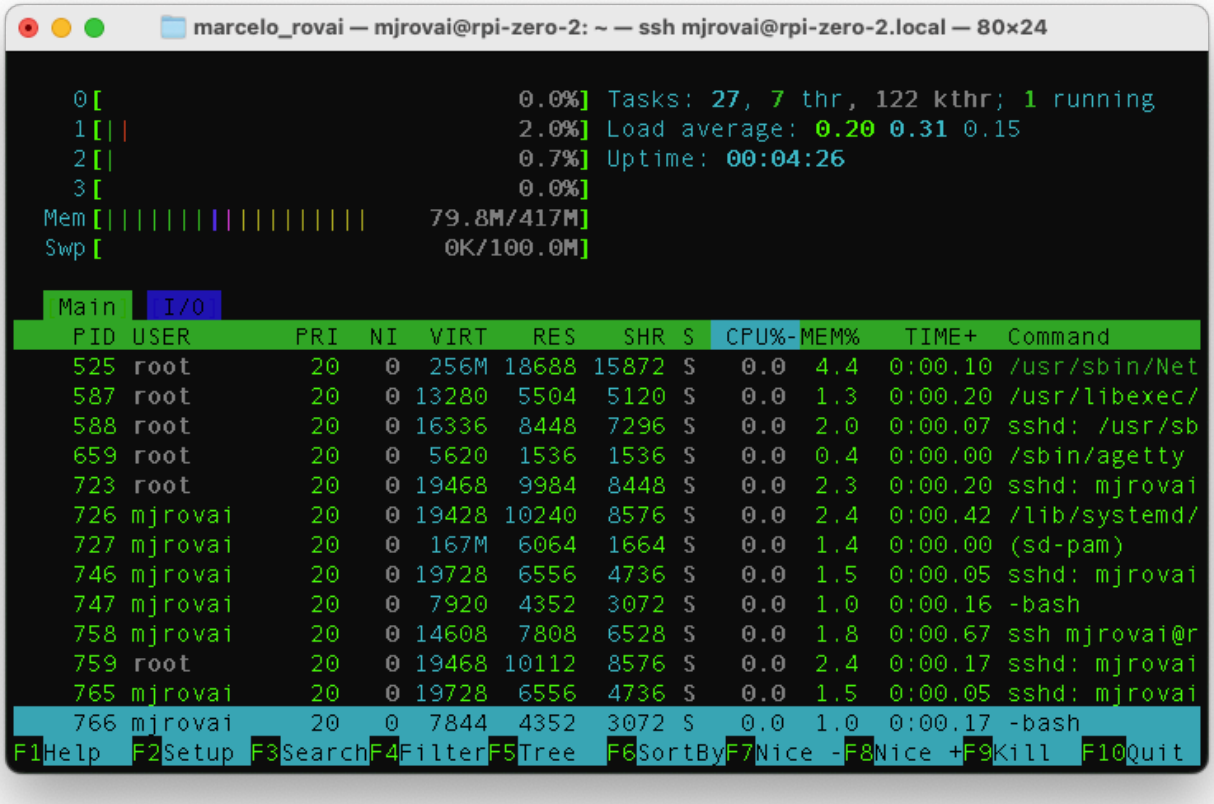
```
sudo apt-get update
```

Pip is a tool for installing external Python modules on a Raspberry Pi. However, it has not been enabled in recent OS versions. To allow it, you should run the command (only once):

```
sudo rm /usr/lib/python3.11/EXTERNALLY-MANAGED
```

Increasing SWAP Memory

Using `htop`, you can easily monitor the resources running on your device:



The first important point is that the SWAP (Swp) memory available is only 100MB, which is very small for running YOLO on the Rasp-Zero.

> Swap memory, also known as swap space, is a part of a computer's hard disk or SSD that temporarily stores inactive data from the Random Access Memory (RAM). This allows the operating system (OS) to continue running even when RAM is full, which can prevent system crashes or slowdowns.

Let's increase it to 2MB.

First, turn off swap-file:

```
sudo dphys-swapfile swapoff
```

Next, you should open and change the file `/etc/dphys-swapfile`. For that, we will use the nano:

```
sudo nano /etc/dphys-swapfile
```

Search for the **CONF_SWAPSIZE** variable (default is 100) and update it to **2000**:

```
CONF_SWAPSIZE=2000
```

And save the file.

Next, turn on the swapfile again and reboot the Rasp-zero:

```
sudo dphys-swapfile setup
sudo dphys-swapfile swapon
sudo reboot
```

When your device is rebooted (you should enter with the SSH again), you will realize that the maximum swap memory value shown on top is now something near 2GB (in my case, 1.95GB).

> To keep the *htop* running, you should open another terminal window to interact continuously with your Rasp-Zero.

To shut down the Rpi-Zero via terminal:

When you want to turn off your Raspberry Pi, simply pulling the power cord is **not** a good idea. This is because the Rasp-Zero may still be writing data to the SD card, in which case merely powering down may result in data loss or, even worse, a corrupted SD card.

For safety shut down, use the command line:

```
sudo shutdown -h now
```

> To avoid possible data loss and SD card corruption, before removing the power, you should wait a few seconds after shutdown for the Raspberry Pi's LED to stop blinking and go dark. Once the LED goes out, it's safe to power down.

Installing the Camera

There are now several Raspberry Pi camera modules. The original 5-megapixel model was released in 2013, followed by an 8-megapixel Camera Module 2, released in 2016. The latest camera model is the 12-megapixel Camera Module 3, released in 2023.

The original 5MP device is no longer available from Raspberry Pi but can be found from several alternative suppliers (Arducam OV5647).

To use the camera, let's first change the configuration.txt file:

```
sudo nano /boot/firmware/config.txt
```

At the bottom of the file, add the line:

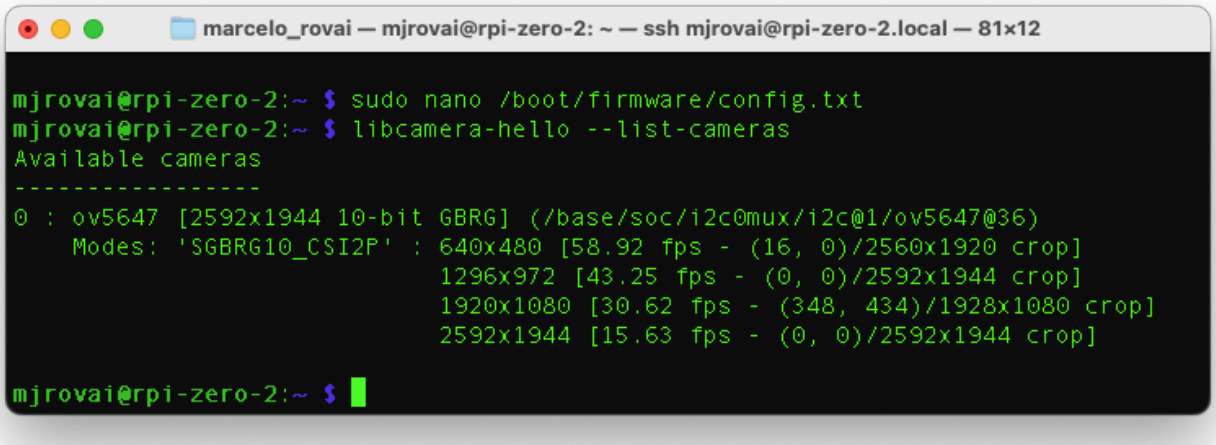
```
dtoverlay=ov5647,cam0
```

Save the file (CTRL+O [ENTER] CRTL+X) and reboot the Rasp-Zero:

```
Sudo reboot
```

After the boot, you can see if the camera is listed:

```
libcamera-hello --list-cameras
```



> libcamera is an open-source software library that supports camera systems directly from the Linux operating system on Arm processors. It minimizes proprietary code running on the Broadcom GPU.

Let's capture a jpeg image with a resolution of 640 x 480 for testing and save it to a file named test.jpg

```
rpicam-jpeg --output test.jpg --width 640 --height 480
```

if we want to see the file saved, we should use `ls -f`, which lists all current directory content in long format.

> For more command line examples for the Raspberry Pi, use this link.

Transfer Files between the Rasp-Zero and the desktop

However, it is impossible to see the image content in the Rasp-Zero (in a simple way, without unique apps), as we do not have a GUI. The easiest way to see the file content is to transfer it to our main computer. We can transfer it using a pen drive or an FTP program over the network. For the last one, let's use FileZilla FTP Client.

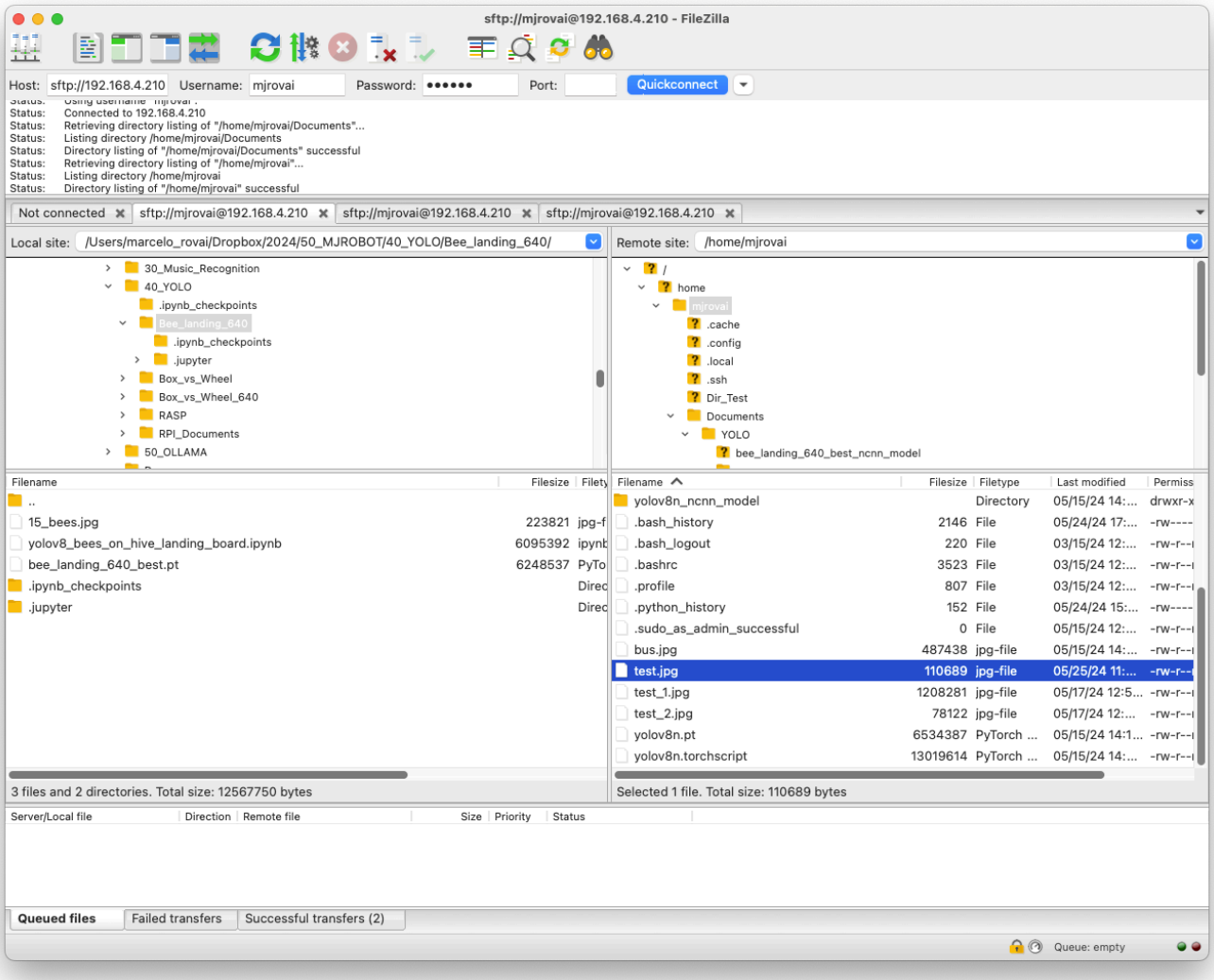
Follow the instructions and install the program for your Desktop OS.

It is essential to know what is the Rasp-Zero IP, for example, on the terminal use:

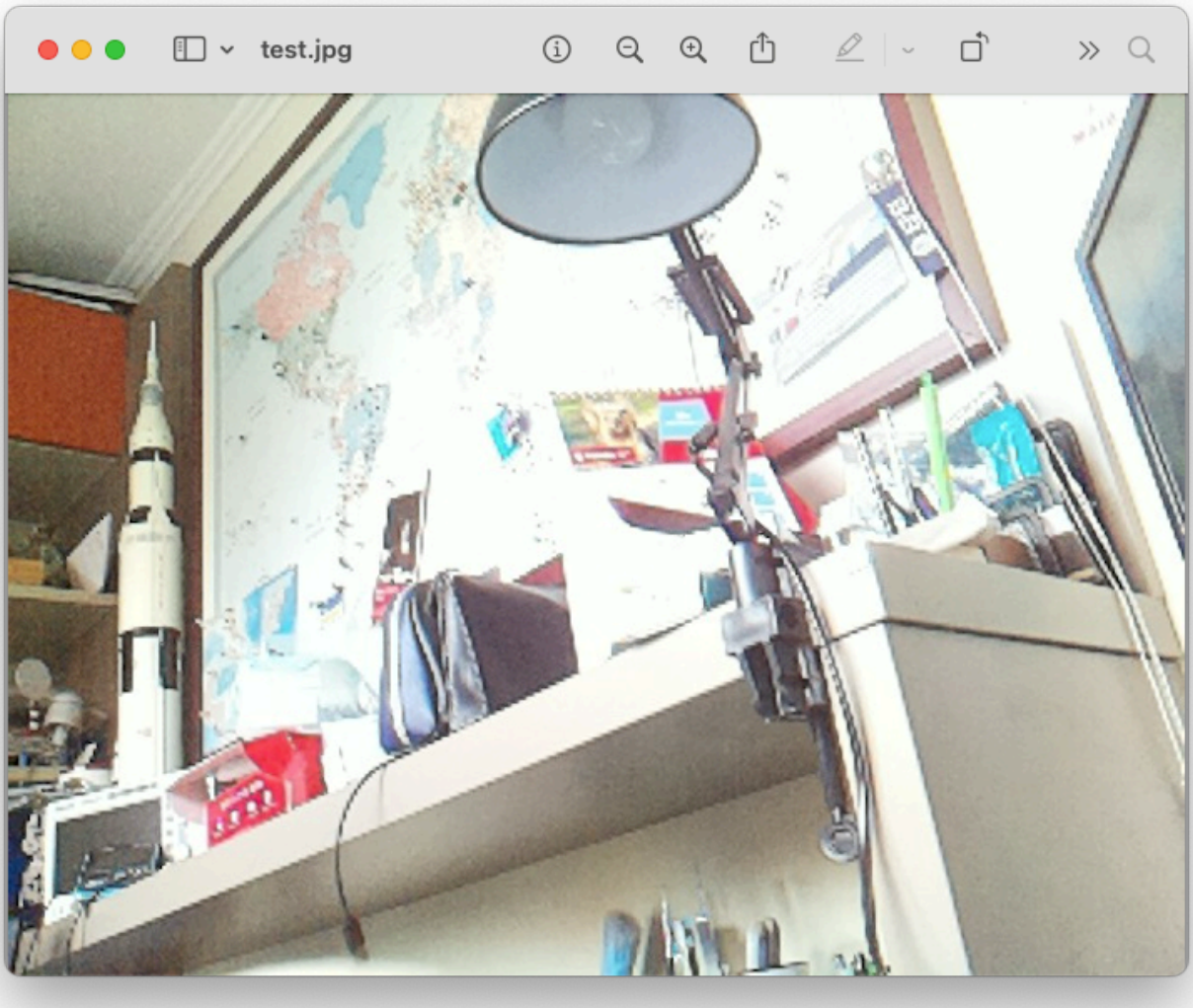
```
Ifconfig -a
```

And copy the IP address that appears with `wlan0: inet`. In my case, it is `192.168.4.210`.

Use this IP as the Host in the File: `sftp://192.168.4.210`, and enter your Rasp-Zero Username and Password. Pressing `Quickconnect` will open two separate windows, one for your desktop and another for the Rasp-Zero—the test.jpg should be in your user directory.



Please select the destination on your desktop (for example, `Downloads` or `C:/`), and press the mouse right button over the file name to `Download` it. Then, use any application on your computer to open the image.



> You can also use FileZilla to Upload files from your computer to the Rasp-Zero.

Installing and using Ultralytics YOLOv8

Ultralytics YOLOv8, is a version of the acclaimed real-time object detection and image segmentation model, YOLO. YOLOv8 is built on cutting-edge advancements in deep learning and computer vision, offering unparalleled performance in terms of speed and accuracy. Its streamlined design makes it suitable for various applications and easily adaptable to different hardware platforms, from edge devices to cloud APIs.

Let's start installing the Ultarlytics packages for local inference on the Rasp-Zero:

1. Update the packages list, install pip, and upgrade to the latest:

```
sudo apt update
sudo apt install python3-pip -y
pip install -U pip
```

2. Install the `ultralytics` pip package with optional dependencies:

```
pip install ultralytics[export]
```

3. Reboot the device:

```
sudo reboot
```

Testing the YOLO

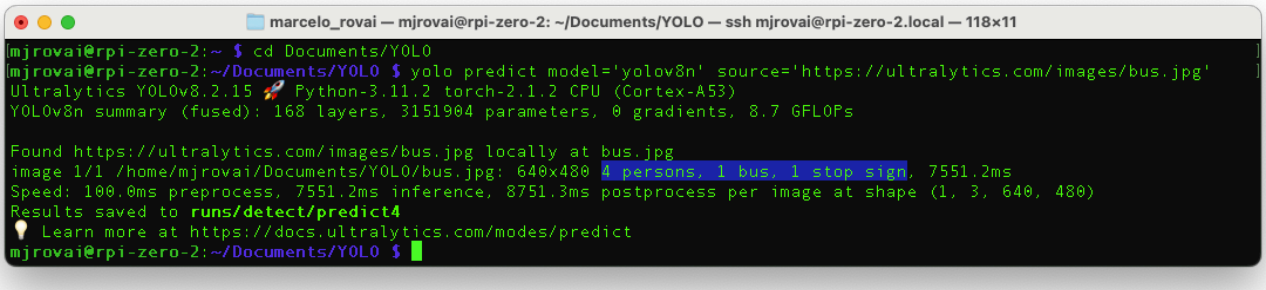
After the Rasp-Zero booting, let's create a directory for working with YOLO and change the current location to it::

```
mkdir Documents/YOLO
cd Documents/YOLO
```

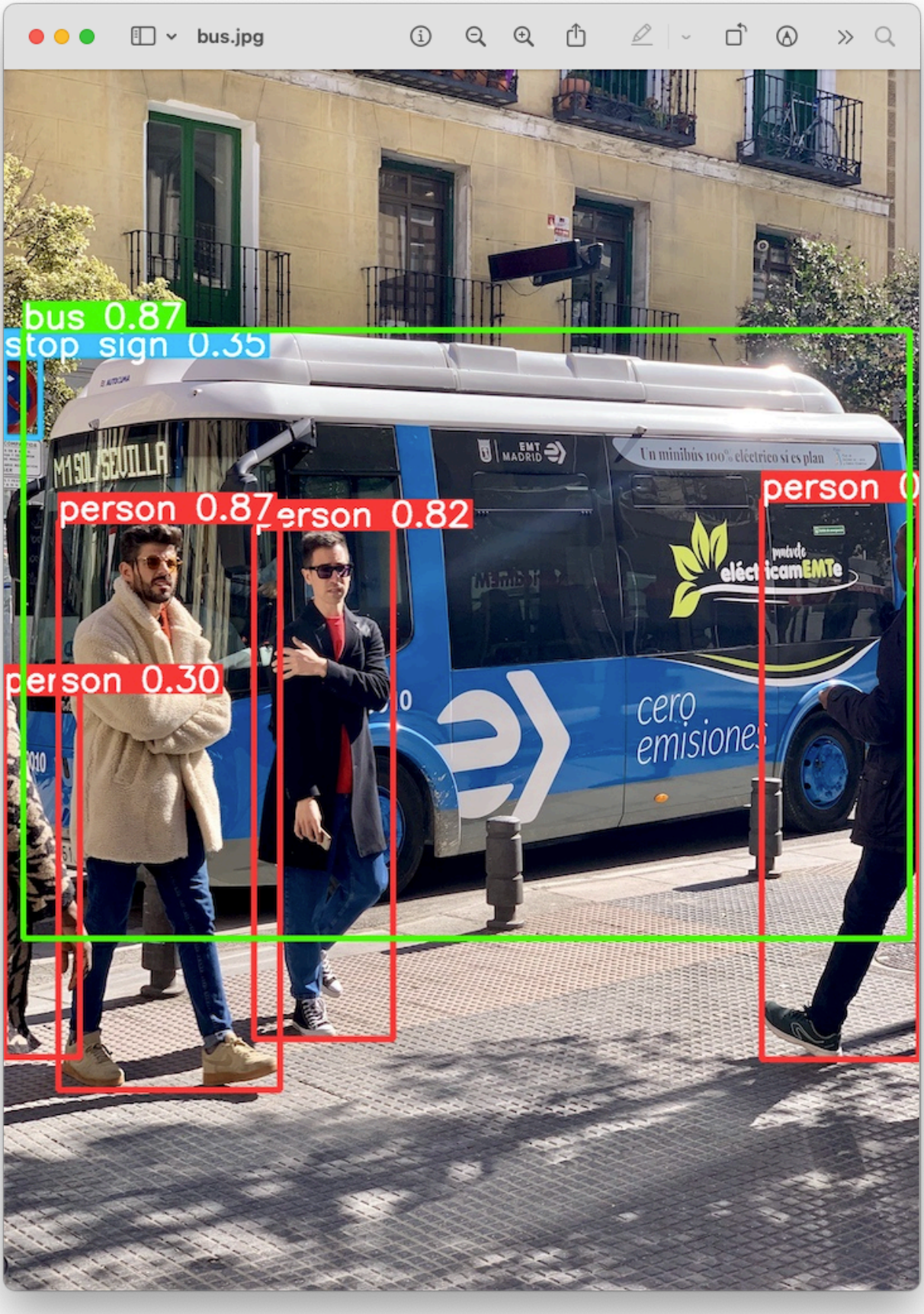
Let's run inference on an image that will be downloaded from the Ultralytics website, using the YOLOV8n model (the smallest in the family) at the Terminal (CLI):

```
yolo predict model='yolov8n' source='https://ultralytics.com/images/bus.jpg'
```

The inference result will appear in the terminal. In the image (bus.jpg), 4 `persons`, 1 `bus,` and 1 `stop signal` were detected:



Also, we got a message that `Results saved to runs/detect/predict4`. Inspecting that directory, we can see a new image saved (bus.jpg). Let's download it from the Rasp-Zero to our desktop for inspection:



So, the Ultrayitics YOLO is correctly installed on our Rasp-Zero.

Export Model to NCNN format

An issue is the high latency for this inference, 7.6 s, even with the smaller model of the family (YOLOv8n). This is a reality of deploying computer vision models on edge devices with limited computational power, such as the Rasp-Zero. One alternative is to use a format optimized for optimal performance. This ensures that even devices with limited processing power can handle advanced computer vision tasks well.

Of all the model export formats supported by Ultralytics, the NCNN is a high-performance neural network inference computing framework optimized for mobile platforms. From the beginning of the design, NCNN was deeply considerate about deployment and use on mobile phones and did not have third-party dependencies. It is cross-platform and runs faster than all known open-source frameworks (such as TFLite).

NCNN delivers the best inference performance when working with Raspberry Pi devices. NCNN is highly optimized for mobile embedded platforms (such as ARM architecture).

So, let's convert our model and rerun the inference:

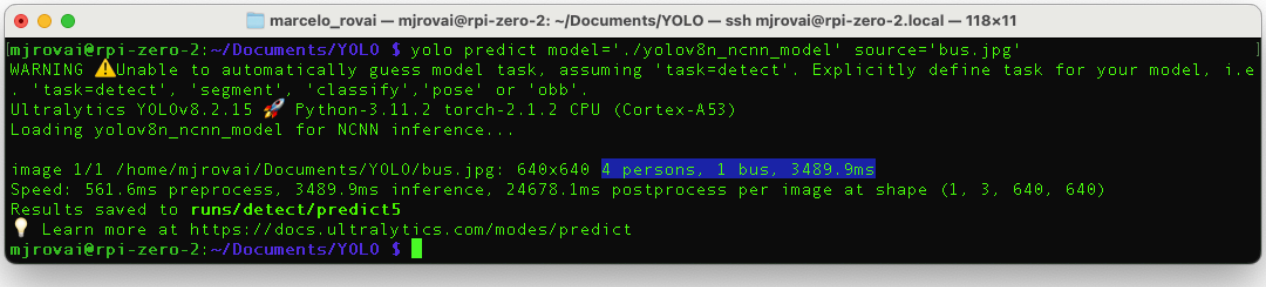
1. Export a YOLOv8n PyTorch model to NCNN format, creating: '/yolov8n_ncnn_model'

```
yolo export model=yolov8n.pt format=ncnn
```

2. Run inference with the exported model (now the source could be the bus.jpg image that was downloaded from the website to the current directory on the last inference):

```
yolo predict model='./yolov8n_ncnn_model' source='bus.jpg'
```

Now, we can see that the latency was reduced by half.



Talking about the YOLO Model

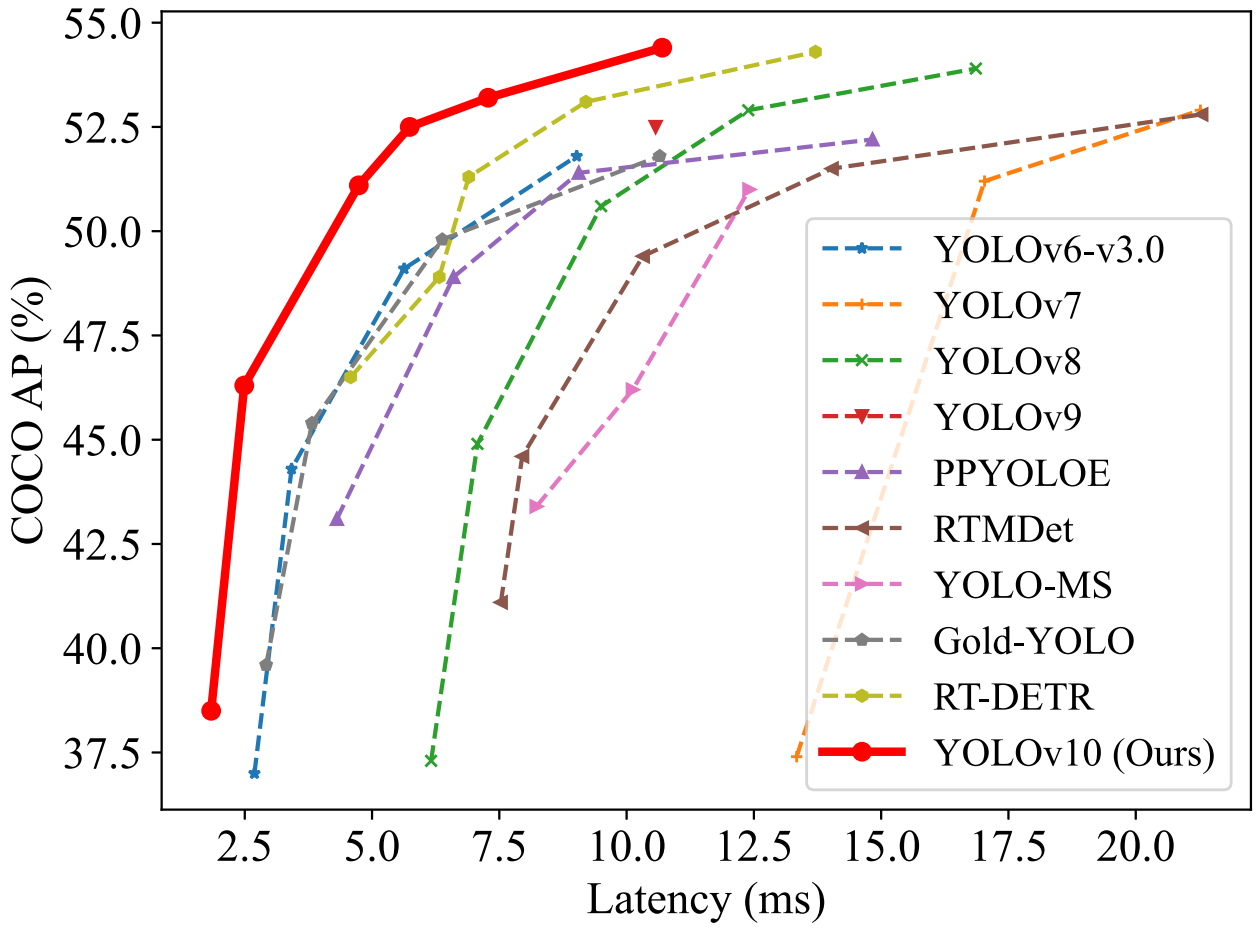
The YOLO (You Only Look Once) model is a highly efficient and widely used object detection algorithm known for its real-time processing capabilities. Unlike traditional object detection systems that repurpose classifiers or localizers to perform detection, YOLO frames the detection problem as a single regression task. This innovative approach enables YOLO to simultaneously predict multiple bounding boxes and their class probabilities from full images in one evaluation, significantly boosting its speed.

Key Features:

1. **Single Network Architecture**:

- YOLO employs a single neural network to process the entire image. This network divides the image into a grid and, for each grid cell, directly predicts bounding boxes and associated class probabilities. This end-to-end training improves speed and simplifies the model architecture.

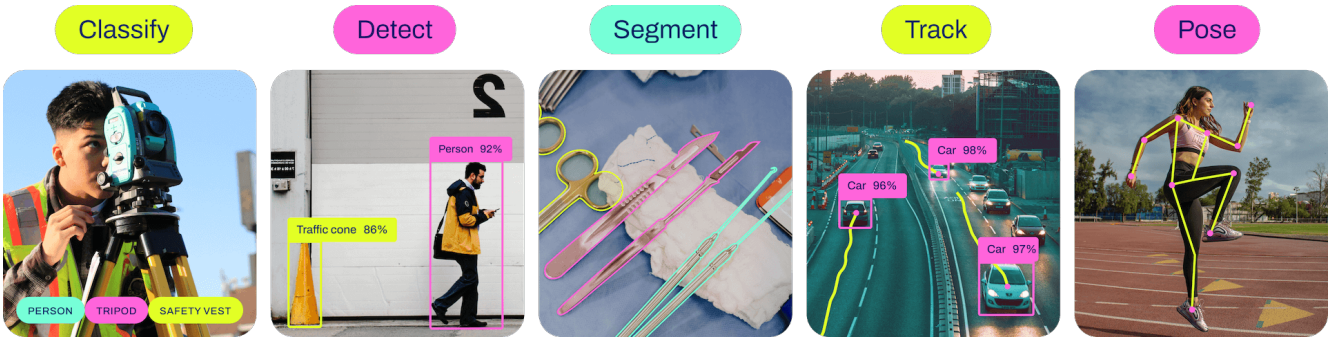
2. **Real-Time Processing**:
   - One of YOLO's standout features is its ability to perform object detection in real time. Depending on the version and hardware, YOLO can process images at high frames per second (FPS). This makes it ideal for applications requiring quick and accurate object detection, such as video surveillance, autonomous driving, and live sports analysis.
3. **Evolution of Versions**:
   - Over the years, YOLO has undergone significant improvements, from YOLOv1 to the latest YOLOv10. Each iteration has introduced enhancements in accuracy, speed, and efficiency. YOLOv8, for instance, incorporates advancements in network architecture, improved training methodologies, and better support for various hardware, ensuring a more robust performance.
   - Although YOLOv10 is the family's newest member with an encouraging performance based on its paper, it was just released (May 2024) and is not fully integrated with the Ultralitycs library. Conversely, the precision-recall curve analysis suggests that YOLOv8 generally outperforms YOLOv9, capturing a higher proportion of true positives while minimizing false positives more effectively (for more details, see this article). So, this work is based on the YOLOv8n.



4. **Accuracy and Efficiency**:

- While early versions of YOLO traded off some accuracy for speed, recent versions have made substantial strides in balancing both. The newer models are faster and more accurate, detecting small objects (such as bees) and performing well on complex datasets.

5. **Wide Range of Applications**:
   - YOLO's versatility has led to its adoption in numerous fields. It is used in traffic monitoring systems to detect and count vehicles, security applications to identify potential threats, and agricultural technology to monitor crops and livestock. Its application extends to any domain requiring efficient and accurate object detection.
6. **Community and Development**:
   - YOLO continues to evolve and is supported by a strong community of developers and researchers (being the YOLOv8 very strong). Open-source implementations and extensive documentation have made it accessible for customization and integration into various projects. Popular deep learning frameworks like Darknet, TensorFlow, and PyTorch support YOLO, further broadening its applicability.
   - Ultralitics YOLOv8 can not only Detect (our case here) but also Segment and Pose models pre-trained on the COCO dataset and YOLOv8 Classify models pre-trained on the ImageNet dataset. Track mode is available for all Detect, Segment, and Pose models.

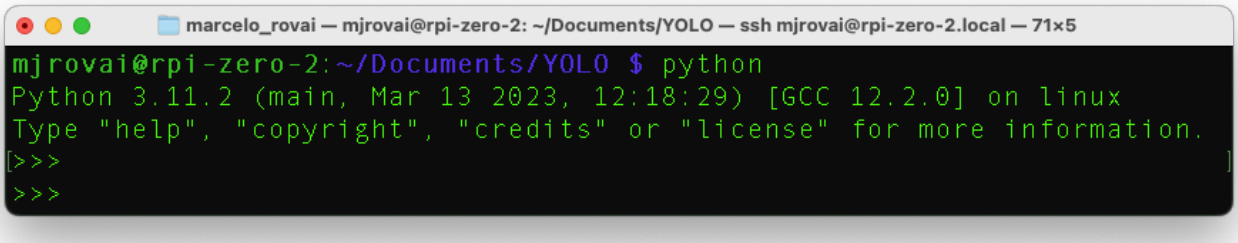


In this tutorial, we leverage the power of YOLOv8 exported to NCNN format to estimate the number of bees at a beehive entrance using a Raspberry Pi Zero 2W (Rasp-Zero) in real-time. This setup demonstrates the practicality and effectiveness of deploying advanced machine learning models on edge devices for real-time environmental monitoring.

Exploring YOLO with Python

To start, let's call the Python Interpreter so we can explore how the YOLO model works, line by line:

```
python
```

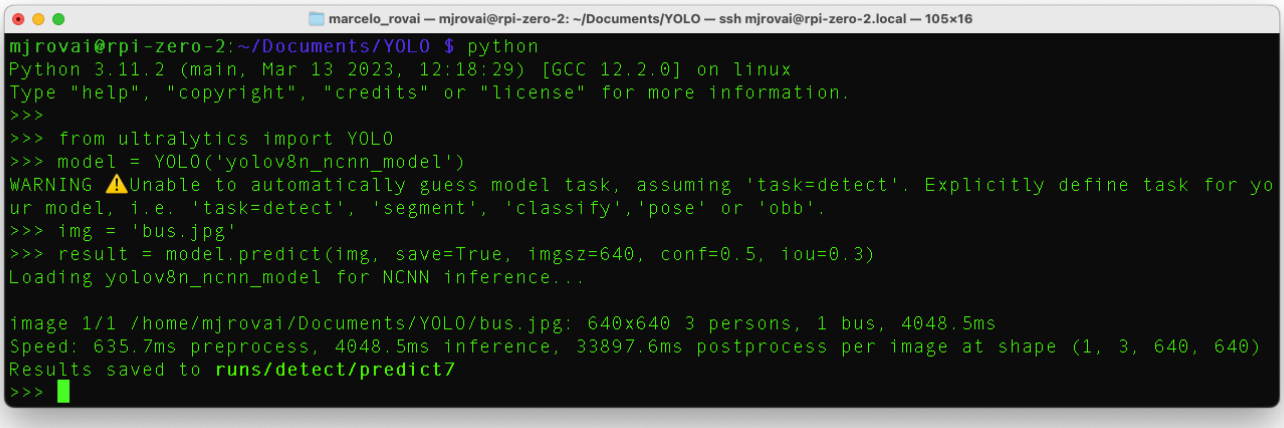


Now, we should call the YOLO library from Ultralitics and load the model:

```
from ultralytics import YOLO
model = YOLO('yolov8n_ncnn_model')
```

Next, run inference over an image (let's use again `bus.jpg`):

```
img = 'bus.jpg'
result = model.predict(img, save=True, imgsz=640, conf=0.5, iou=0.3)
```

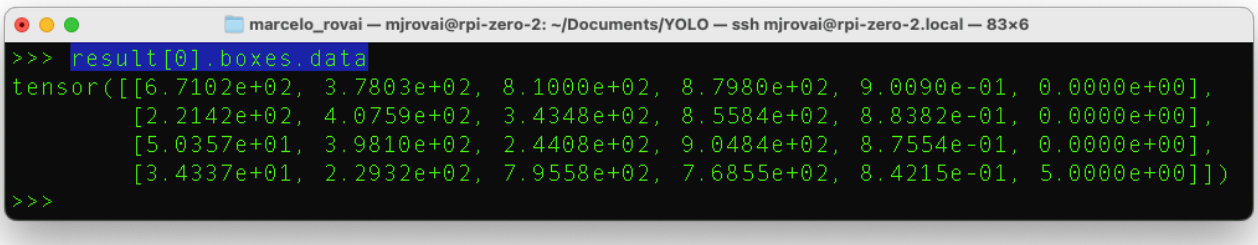


We can verify that the result is the same as the one we get running the inference at the terminal level (CLI).

```
image 1/1 /home/mjrovai/Documents/YOLO/bus.jpg: 640x640 3 persons, 1 bus, 4048.5ms
Speed: 635.7ms preprocess, 4048.5ms inference, 33897.6ms postprocess per image at shape
(1, 3, 640, 640)

Results saved to runs/detect/predict7
```

But, we are interested in analyzing the "result" content.

For example, we can see `result[0].boxes.data`, showing us the main inference result, which is a tensor shape (4, 6). Each line is one of the objects detected, being the 4 first columns, the bounding boxes coordinates, the 5th, the confidence, and the 6th, the class (in this case, `0: person` and `5: bus`):

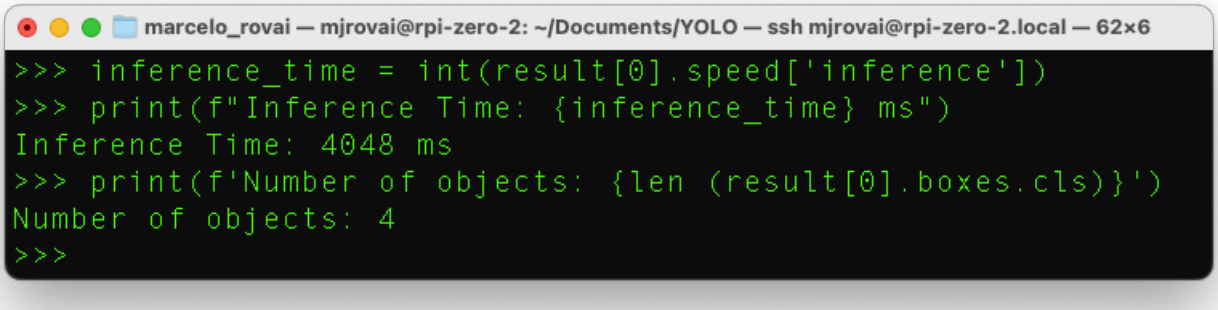


We can access several inference results separately, as the inference time, and have it printed in a better format:

```
inference_time = int(result[0].speed['inference'])
print(f"Inference Time: {inference_time} ms")
```

Or we can have the total number of objects detected:

```
print(f'Number of objects: {len (result[0].boxes.cls)}')
```



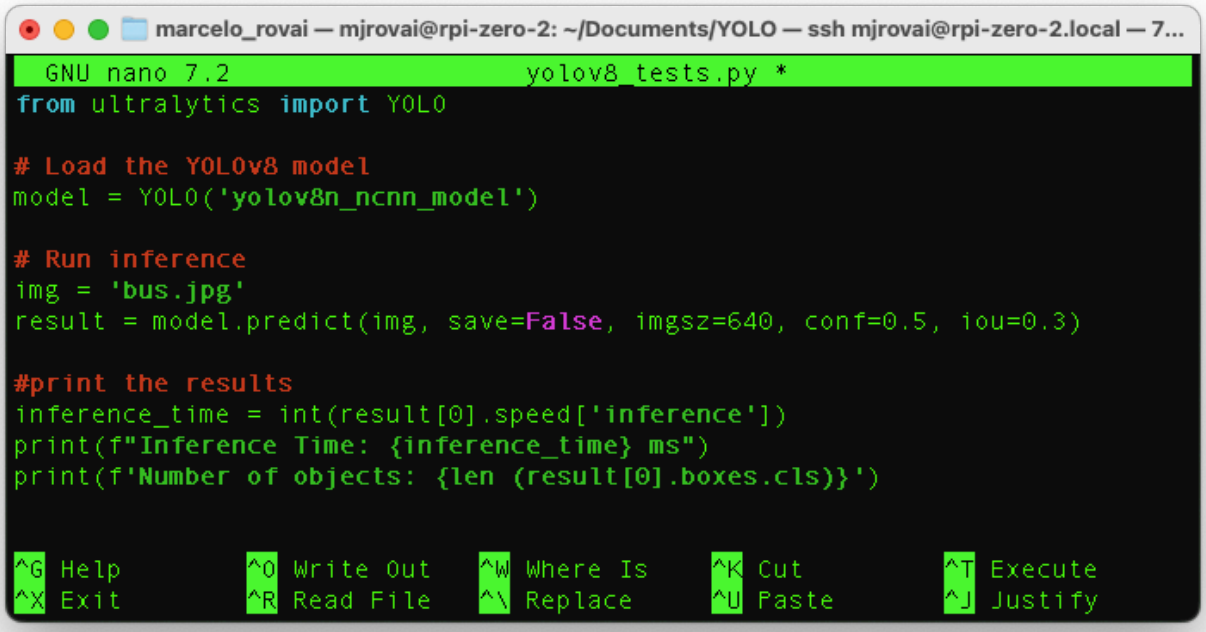
With Python, we can create a detailed output that meets our needs. In our final project, we will run a Python script at once rather than manually entering it line by line in the interpreter.

For that, let's use `nano` as our text editor. First, we should create an empty Python script named, for example, `yolov8_tests.py`:

```
nano yolov8_tests.py
```

Enter with the code lines:

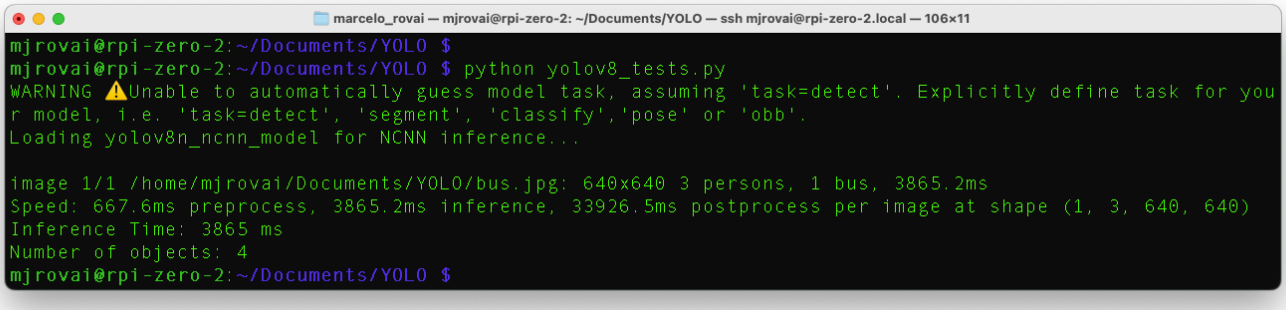
```
from ultralytics import YOLO

# Load the YOLOv8 model
model = YOLO('yolov8n_ncnn_model')

# Run inference
img = 'bus.jpg'
result = model.predict(img, save=False, imgsz=640, conf=0.5, iou=0.3)

# print the results
inference_time = int(result[0].speed['inference'])
print(f"Inference Time: {inference_time} ms")
print(f'Number of objects: {len (result[0].boxes.cls)}')
```



And enter with the commands: `[CTRL+O]` + `[ENTER]` + `[CTRL+X]` to save the Python script.

Run the script:

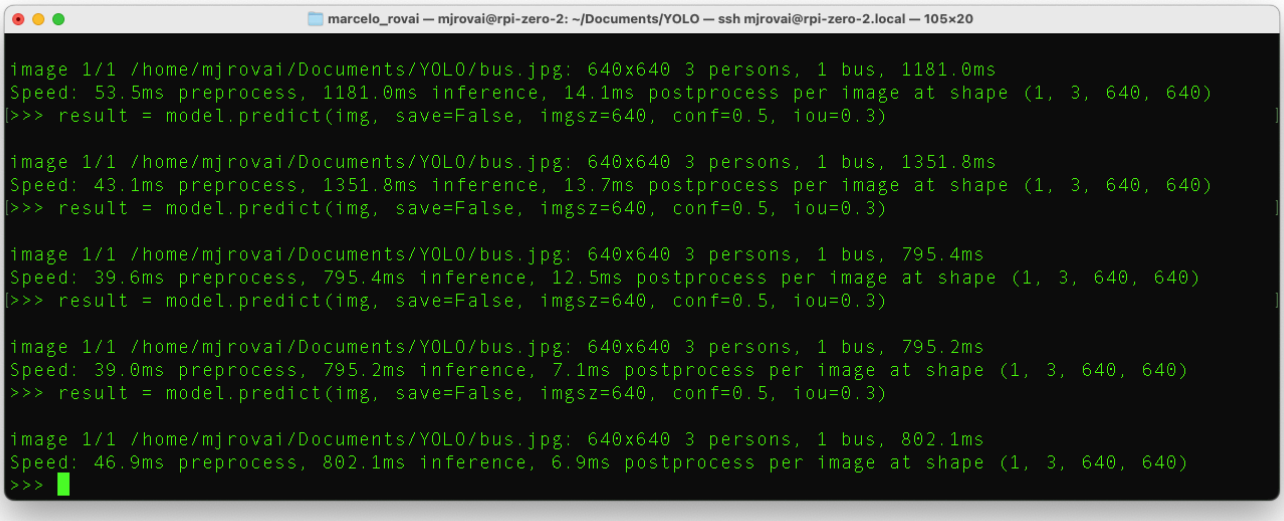
```
python yolov8_tests.py
```



We can verify again that the result is precisely the same as when we run the inference at the terminal level (CLI) and with the built-in Python interpreter.

Note about the Latency:

The process of calling the YOLO library and loading the model for inference for the first time takes a long time, but the inferences after that will be much faster. For example, the first single inference took 3 to 4 seconds, but after that, the inference time is reduced to less than 1 second.



Estimating the number of Bees

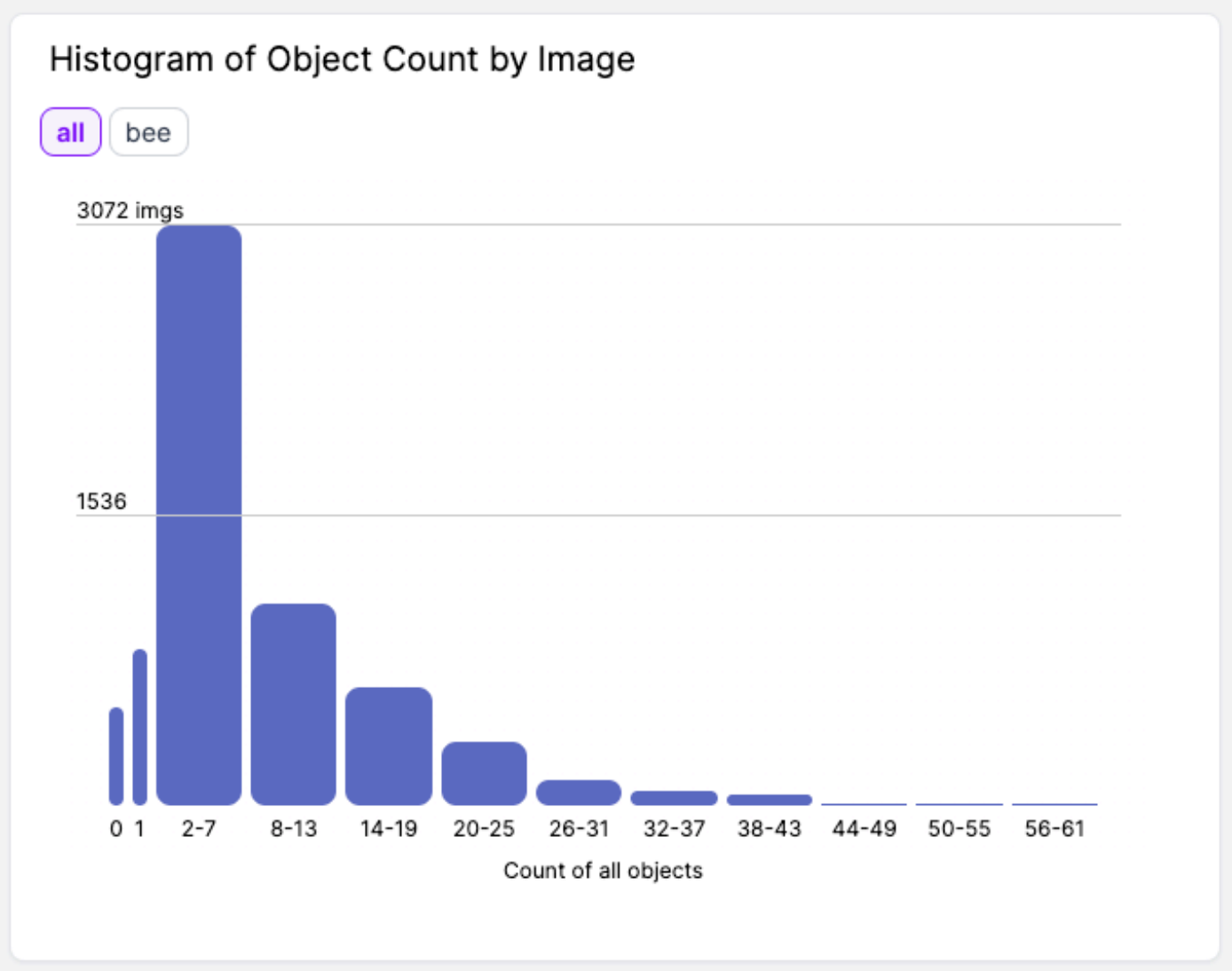
For our project at the university, we are preparing to collect a dataset of bees at the entrance of a beehive using the same camera connected to the Rasp-Zero. The images should be collected every 10 seconds. With the Arducam OV5647, the horizontal Field of View (FoV) is 53.5^{o}, which means that a camera positioned at the top of a standard Hive (46 cm) will capture all of its entrance (about 47 cm).

Dataset

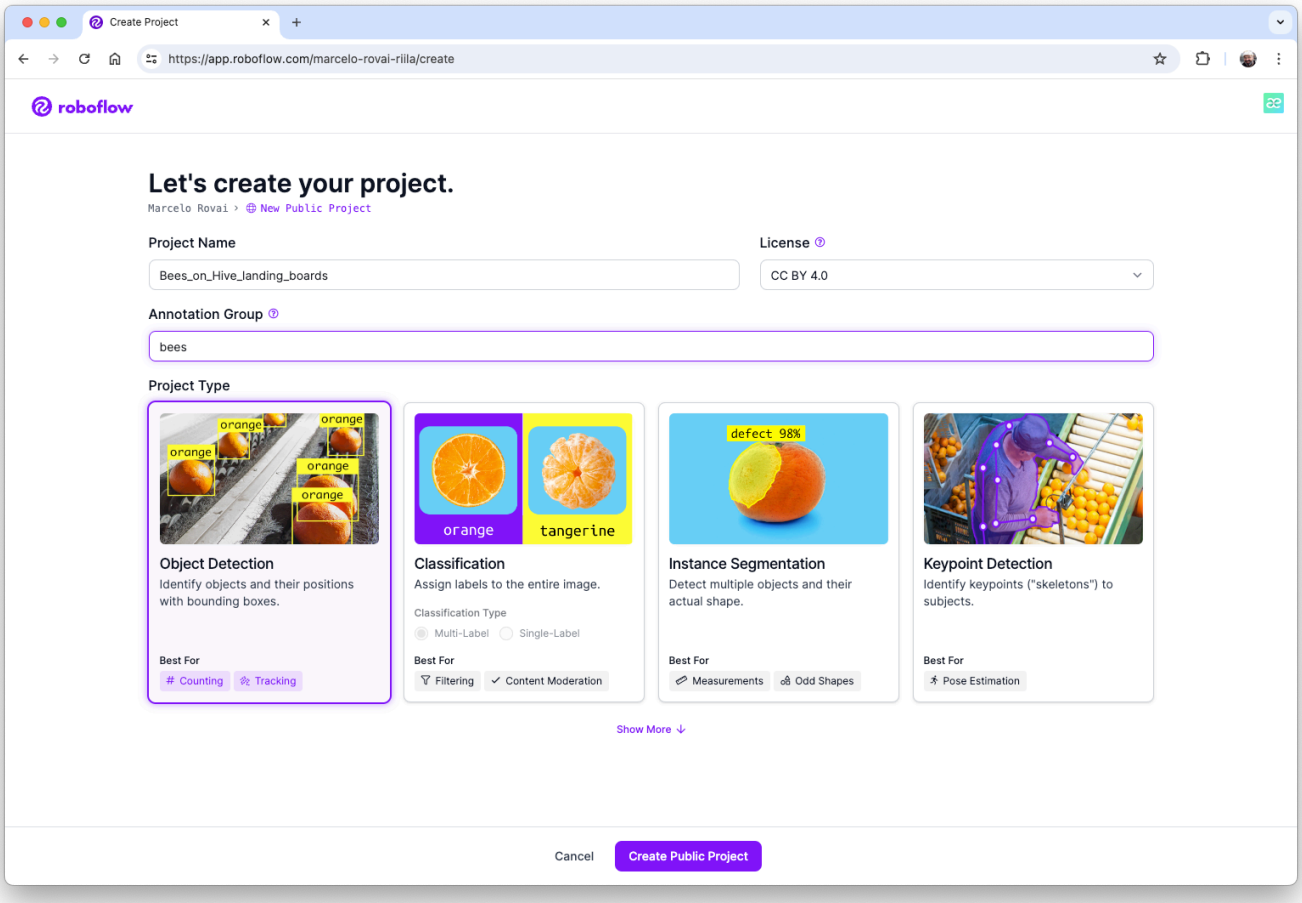
The dataset collection is the most critical phase of the project and should take several weeks or months. For this tutorial, we will use a public dataset: "Sledevic, Tomyslav (2023), "[Labeled dataset for bee detection and direction estimation on beehive landing boards," Mendeley Data, V5, doi: 10.17632/8gb9r2yhfc.5"

The original dataset has 6,762 images (1920 x 1080), and around 8% of them (518) have no bees (only background). This is very important with Object Detection, where we should keep around 10% of the dataset with only background (without any objects to be detected).

The images contain from zero to up to 61 bees:

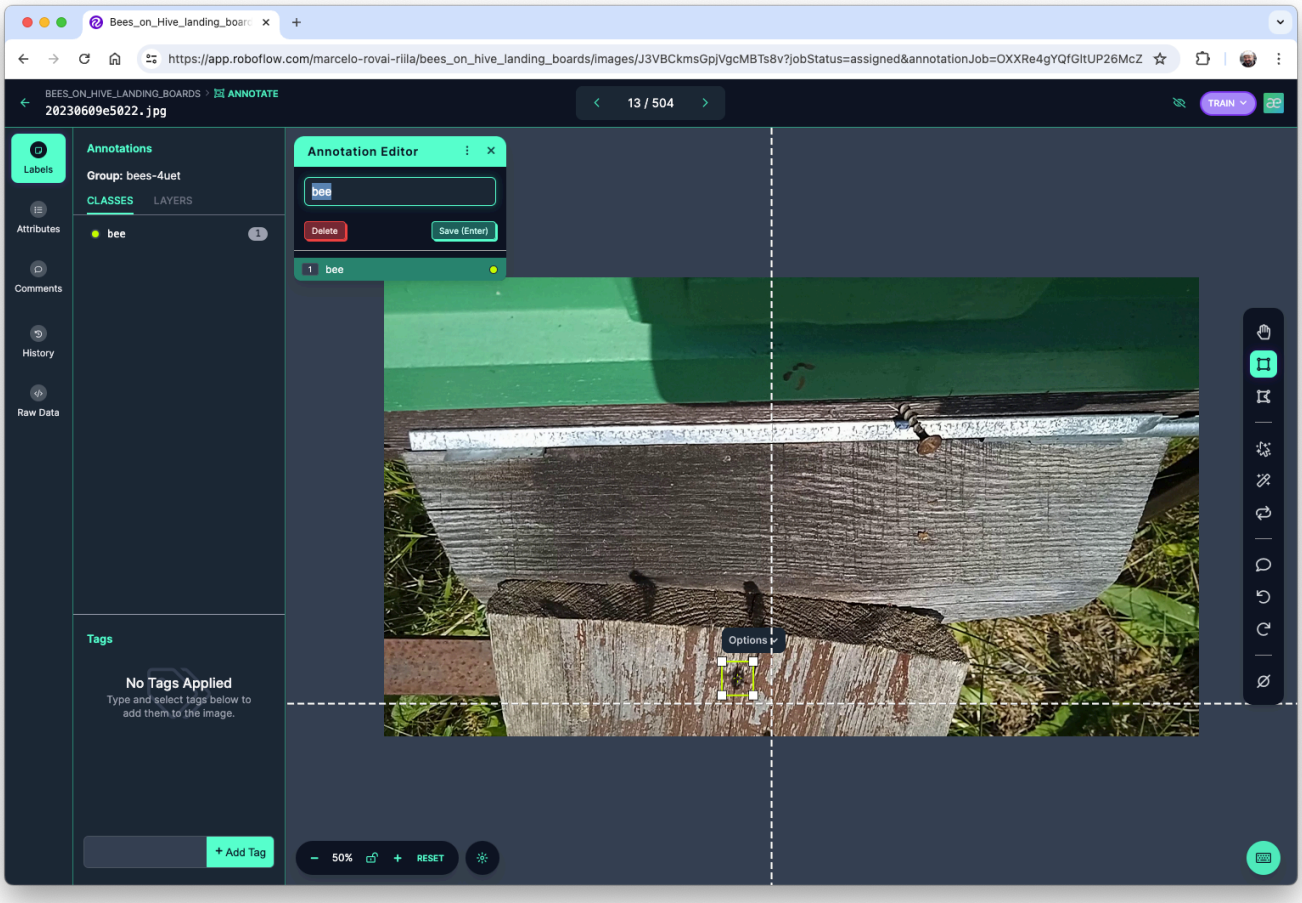


We downloaded the dataset (images and annotations) and uploaded it to Roboflow. There, you should create a free account and start a new project, for example, ("Bees_on_Hive_landing_boards"):

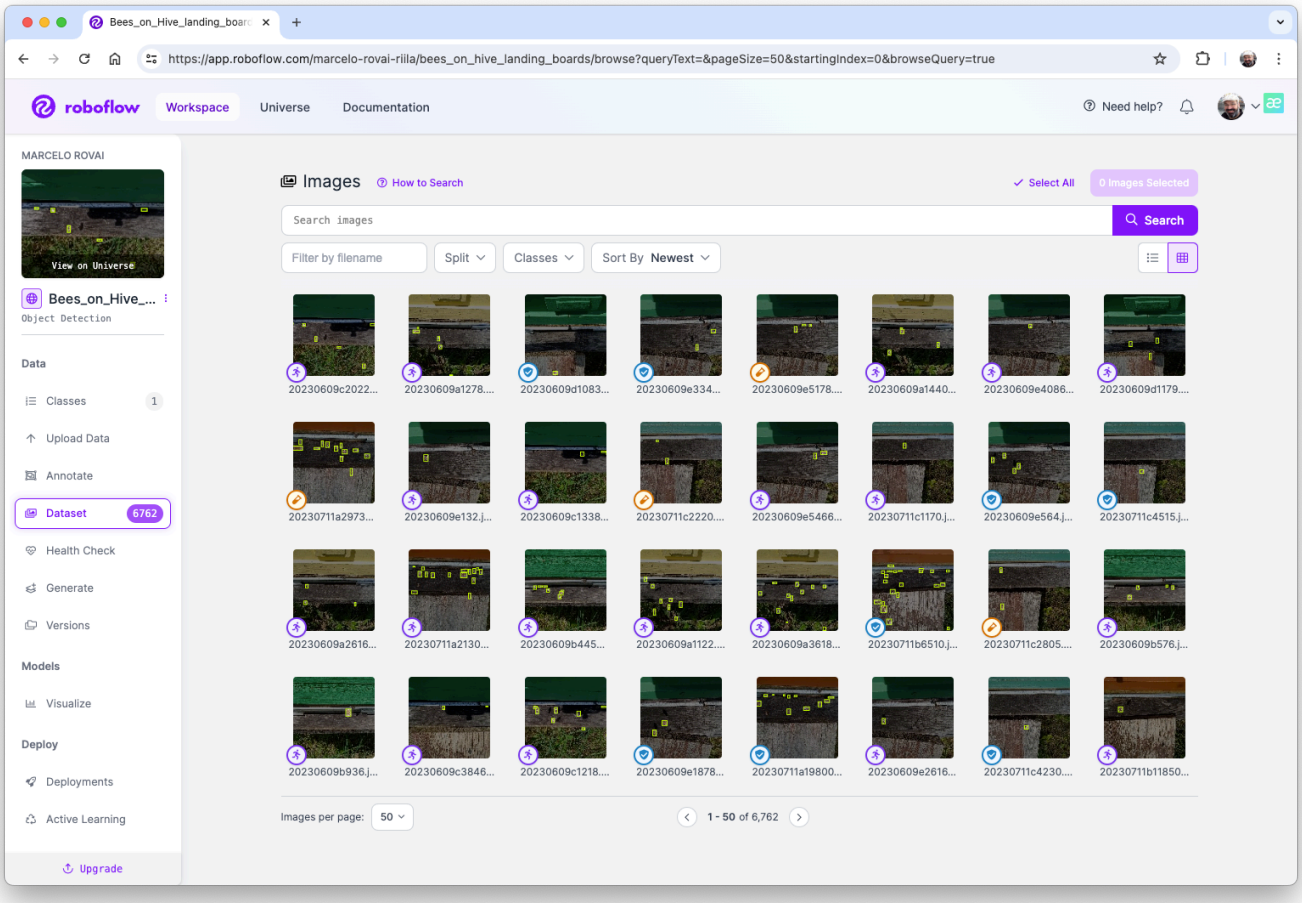


> We will not enter details about the Roboflow process once many tutorials are available.

Once the project is created and the dataset is uploaded, you should review the annotations using the "Auto-Label" Tool. Note that all images with only a background should be saved w/o any annotations. At this step, you can also add additional images.



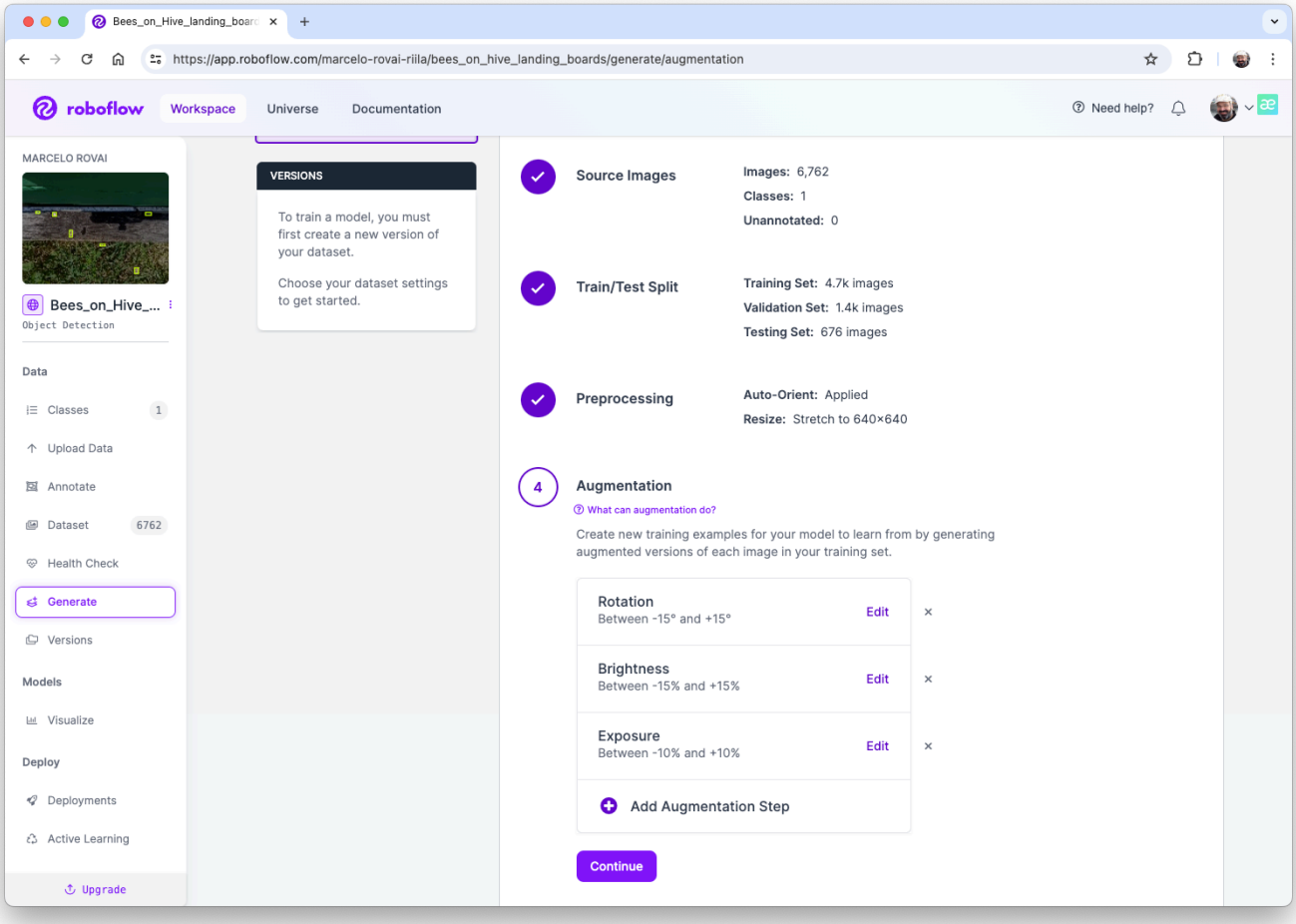
Once all images are annotated, you should split them into training, validation, and testing.



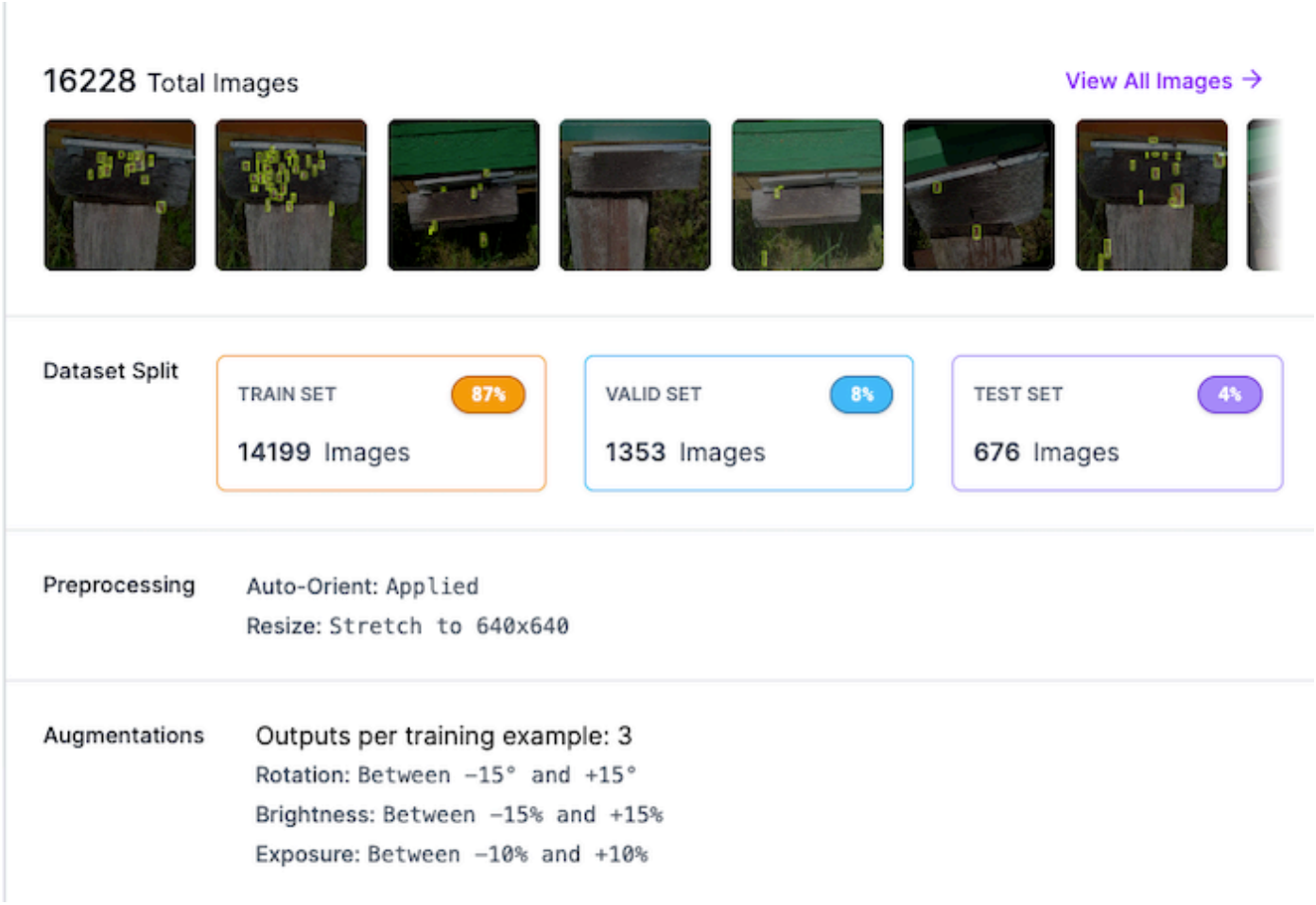
Pre-Processing

The last step with the dataset is preprocessing to generate a final version for training. The Yolov8 model can be trained with 640 x 640 pixels (RGB) images. Let's resize all images and generate augmented versions of each image (augmentation) to create new training examples from which our model can learn.

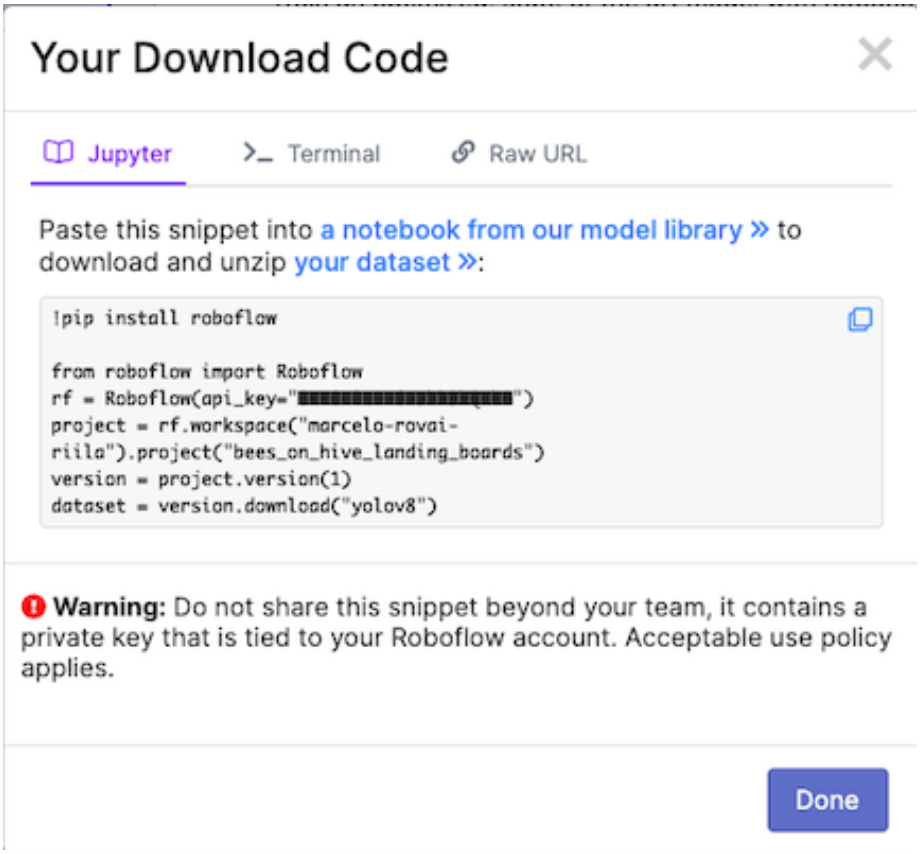
For augmentation, we will rotate the images (+/-15^{o}) and vary the brightness and exposure.



This will create a final dataset of 16,228 images.



Now, you should export the model in a YOLOv8 format. You can download a zipped version of the dataset to your desktop or get a downloaded code to be used with a Jupyter Notebook:



And that is it! We are prepared to start our training using Google Colab.

> The pre-processed dataset can be found at the Roboflow site.

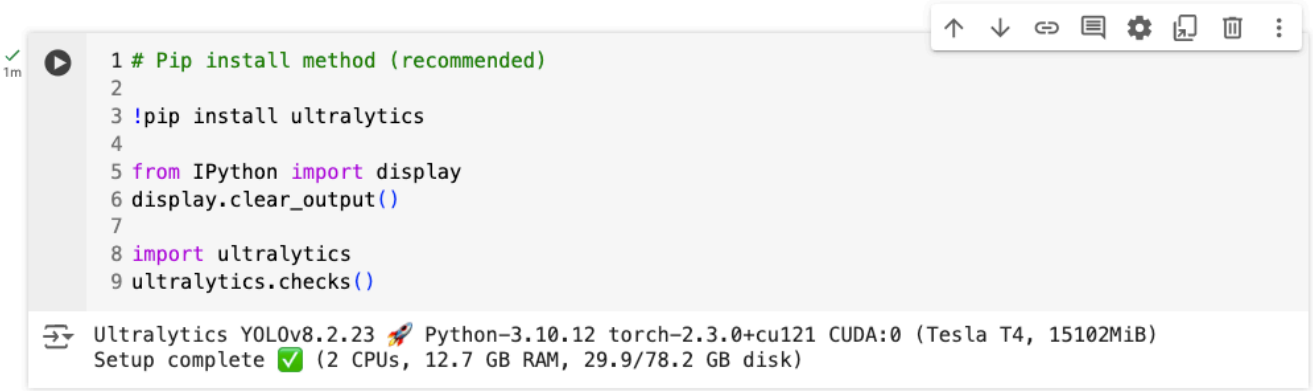
Training YOLOv8 on a Customized Dataset

For training, let's adapt one of the public examples available from Ultralitytics and run it on Google Colab:

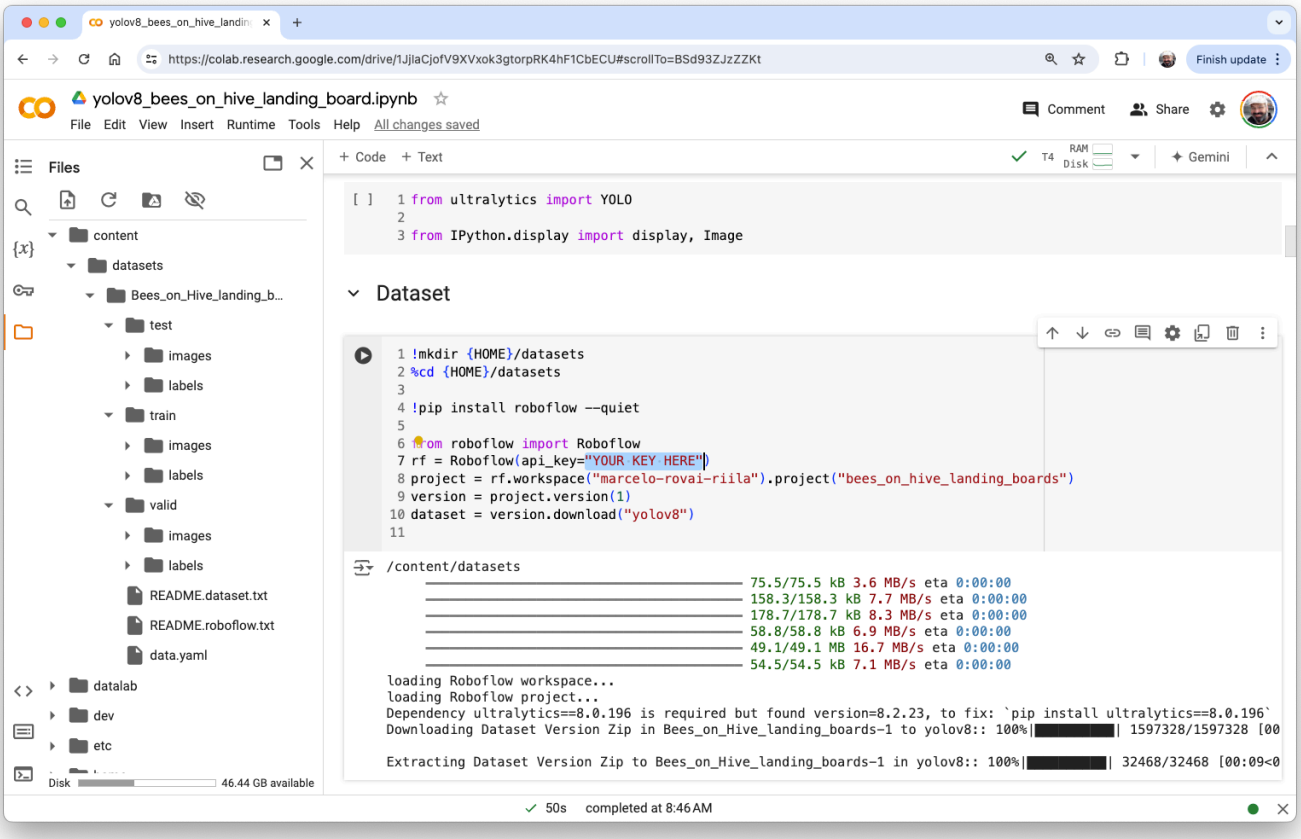
- yolov8_bees_on_hive_landing_board.ipynb 

Critical points on the Notebook:

1. Run it with GPU (the NVidia T4 is free)
2. Install Ultralytics using PIP.



3. Now, you can import the YOLO and upload your dataset to the CoLab, pasting the Download code that you get from Roboflow. Note that your dataset will be mounted under `/content/datasets/`:



4. It is important to verify and change, if needed, the file `data.yaml` with the correct path for the images:

```
names:
- bee
nc: 1
roboflow:
  license: CC BY 4.0
  project: bees_on_hive_landing_boards
  url: https://universe.roboflow.com/marcelo-rovai-
riila/bees_on_hive_landing_boards/dataset/1
  version: 1
  workspace: marcelo-rovai-riila
test: /content/datasets/Bees_on_Hive_landing_boards-1test/images
train: /content/datasets/Bees_on_Hive_landing_boards-1/train/images
val: /content/datasets/Bees_on_Hive_landing_boards-1/valid/images
```

5. Define the main hyperparameters that you want to change from default, for example:

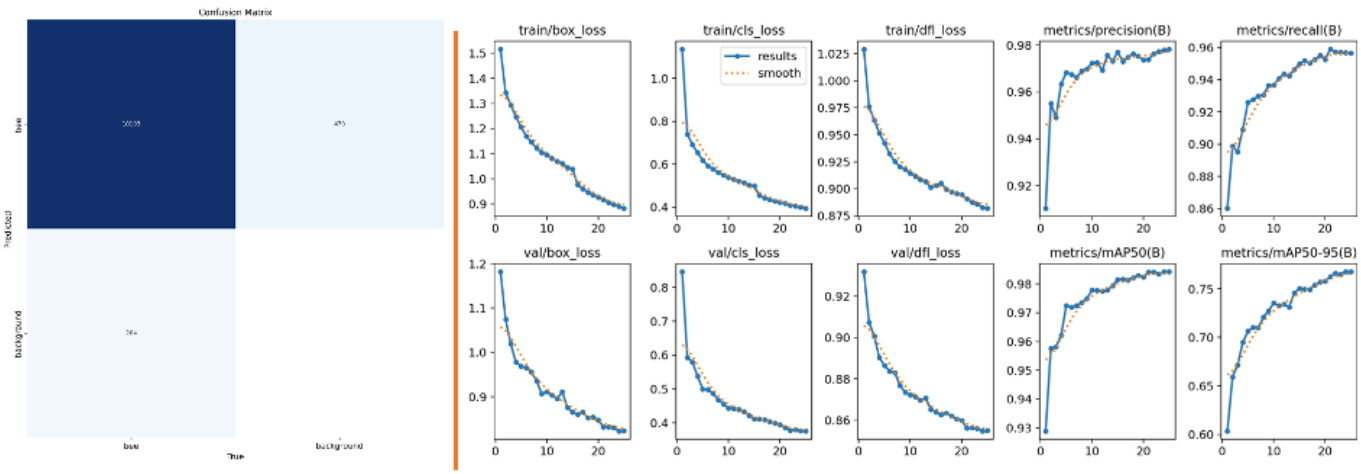
```
MODEL = 'yolov8n.pt'
IMG_SIZE = 640
EPOCHS = 25 # For a final project, you should consider at least 100 epochs
```

6. Run the training (using CLI):

```
!yolo task=detect mode=train model={MODEL} data={dataset.location}/data.yaml epochs=
{EPOCHS} imgsz={IMG_SIZE} plots=True
```

```
25 epochs completed in 2.679 hours.
Optimizer stripped from runs/detect/train3/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train3/weights/best.pt, 6.2MB

Validating runs/detect/train3/weights/best.pt...
Ultralytics YOLOv8.2.15 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 43/43 [00:33<00:00,  1.27it/s]
                   all       1353      10477      0.978      0.957      0.984      0.768
Speed: 0.3ms preprocess, 2.5ms inference, 0.0ms loss, 5.6ms postprocess per image
Results saved to runs/detect/train3
```

The model took 2.7 hours to train and has an excellent result (mAP50 of 0.984). At the end of the training, all results are saved in the folder listed, for example: `/runs/detect/train3/`. There, you can find, for example, the confusion matrix and the metrics curves per epoch.



7. Note that the trained model (`best.pt`) is saved in the folder `/runs/detect/train3/weights/`. Now, you should validade the trained model with the `valid/images`.

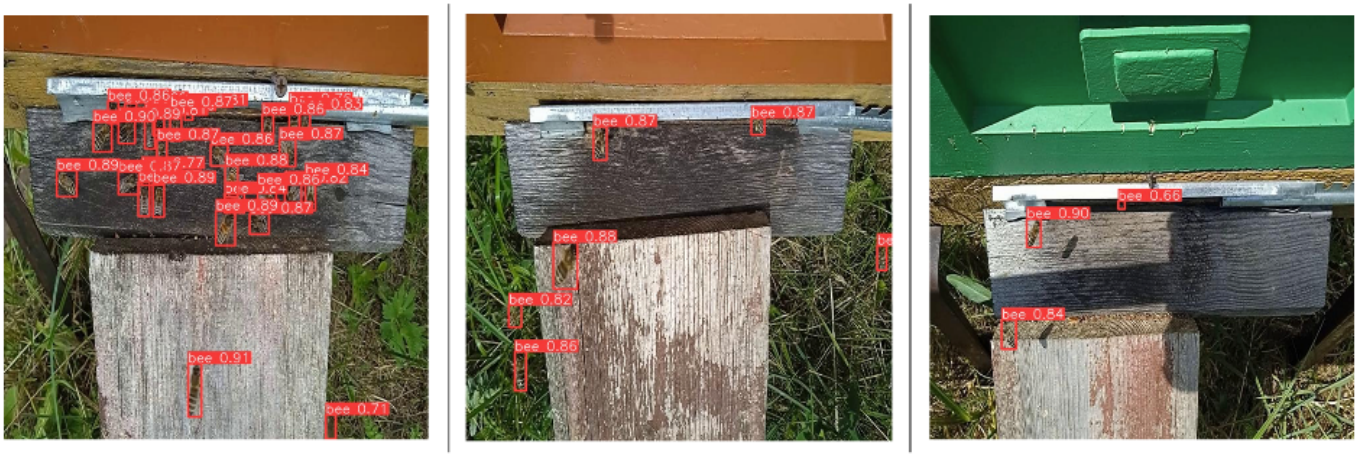
```
!yolo task=detect mode=val model={HOME}/runs/detect/train3/weights/best.pt data=
{dataset.location}/data.yaml
```

The results were similar to training.

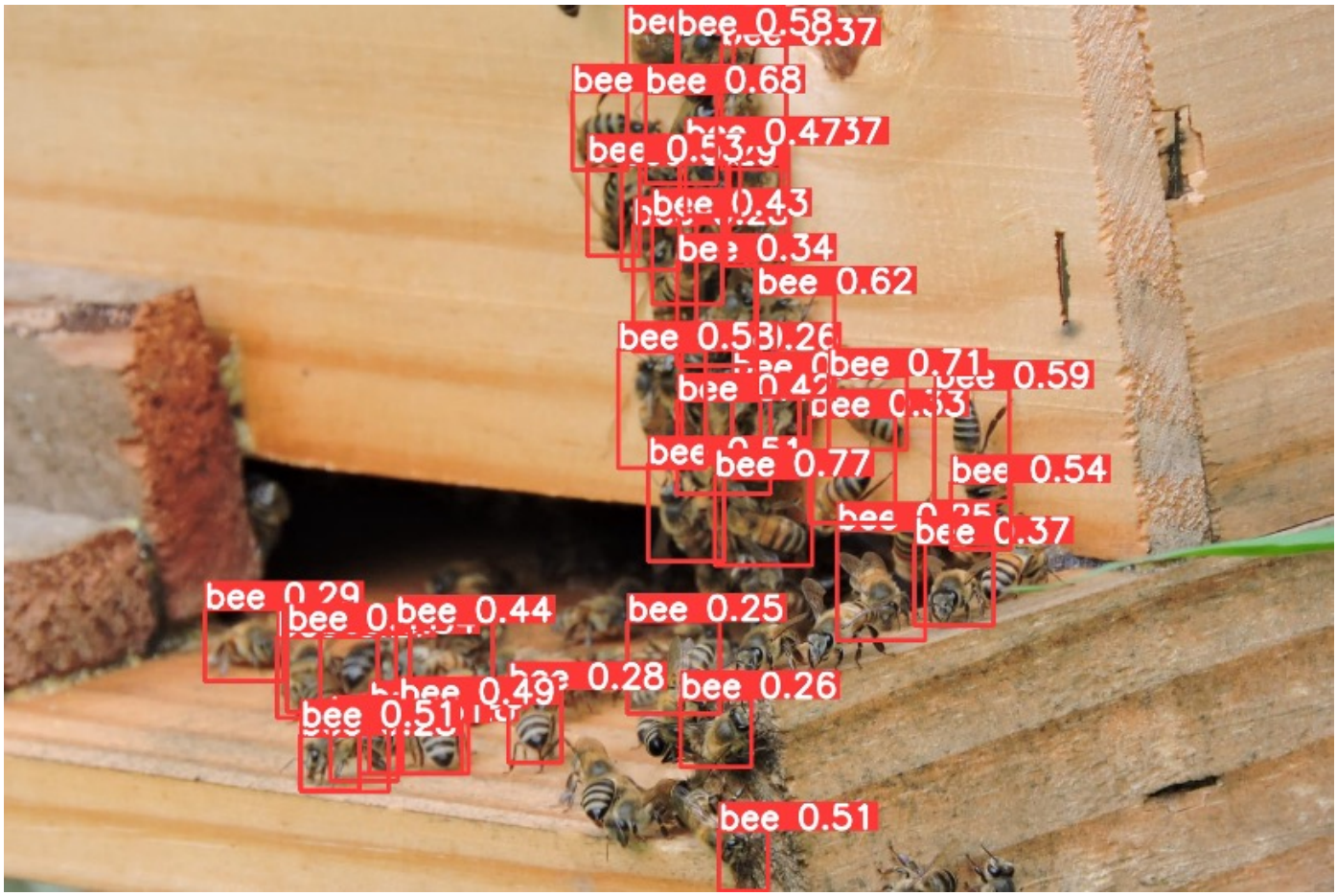
8. Now, we should perform inference on the images left aside for testing

```
!yolo task=detect mode=predict model={HOME}/runs/detect/train3/weights/best.pt conf=0.25
source={dataset.location}/test/images save=True
```

The inference results are saved in the folder `runs/detect/predict`. Let's see some of them:



We can also perform inference with a completely new and complex image from another beehive with a different background (the beehive of Professor Maurilio of our University). The results were great (but not perfect and with a lower confidence score). The model found 41 bees.



9. The last thing to do is export the train, validation, and test results for your Drive at Google. To do so, you should mount your drive.

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

   and copy the content of `/runs` folder to a folder that you should create in your Drive, for example:

   ```
   !scp -r /content/runs
   '/content/gdrive/MyDrive/10_UNIFEI/Bee_Project/YOLO/bees_on_hive_landing'
   ```

Inference with the trained model, using the Rasp-Zero

Using the FileZilla FTP, let's transfer the `best.pt` to our Rasp-Zero (before the transfer, you may change the model name, for example, `bee_landing_640_best.pt`).

The first thing to do is convert the model to an NCNN format:

```
yolo export model=bee_landing_640_best.pt format=ncnn
```

As a result, a new converted model, `bee_landing_640_best_ncnn_model` is created in the same directory.

Let's create a folder to receive some test images (under `Documents/YOLO/`:

```
mkdir test_images
```

Using the FileZilla FTP, let's transfer a few images from the test dataset to our Rasp-Zero:

Let's use the Python Interpreter:

```
python
```

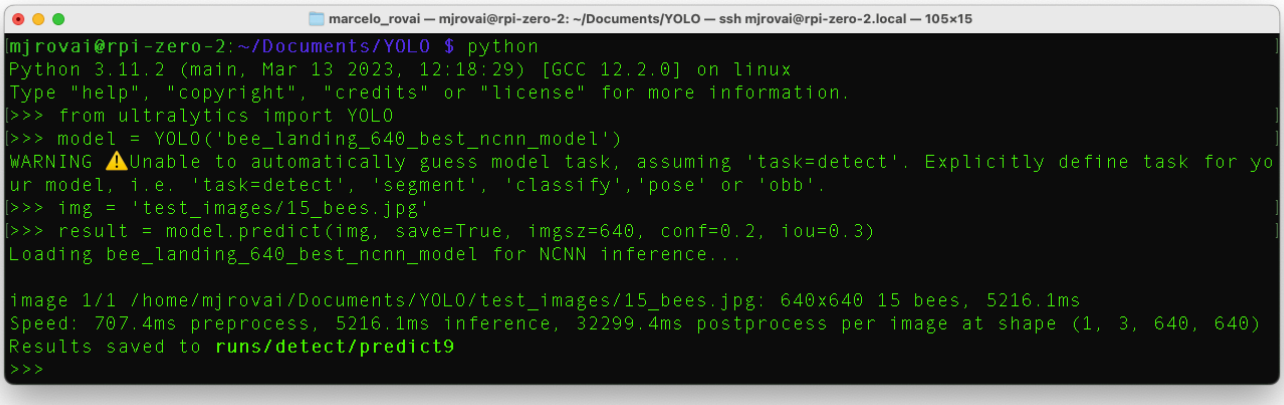
As before, we will import the YOLO library and define our converted model to detect bees:

```
from ultralytics import YOLO
model = YOLO('bee_landing_640_best_ncnn_model')
```

Now, let's define an image and call the inference (we will save the image result this time to external verification):

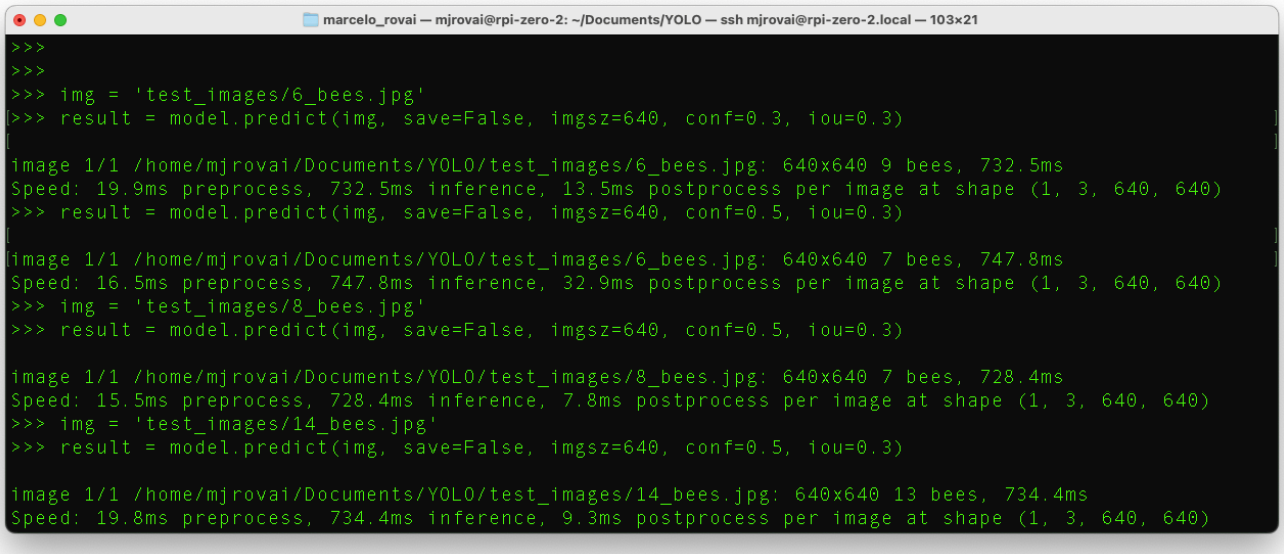
```
img = 'test_images/15_bees.jpg'
result = model.predict(img, save=True, imgsz=640, conf=0.2, iou=0.3)
```

The inference result is saved on the variable `result,` and the processed image on `runs/detect/predict9`



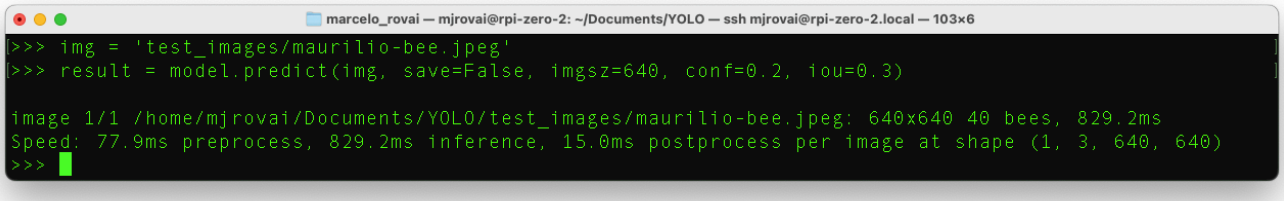
Using FileZilla FTP, we can send the inference result to our Desktop for verification:

let's go over the other images, analyzing the number of objects (bees) found:



Depending on the confidence, we can have some false positives or negatives. But in general, with a model trained based on the smaller base model of the YOLOv8 family (YOLOv8n) and also converted to NCNN, the result is pretty good, running on an Edge device such as the Rasp-Zero. Also, note that the inference latency is around 730ms.

For example, by running the inference on `Maurilio-bee.jpeg`, we can find `40 bees`. During the test phase on Colab, 41 bees were found (we only missed one here.)



Considerations about the Post-Processing

Our final project should be very simple in terms of code. We will use the camera to capture an image every 10 seconds. As we did in the previous section, the captured image should be the input for the trained and converted model. We should get the number of bees for each image and save it in a database (for example, timestamp: number of bees).

We can do it with a single Python script or use a Linux system timer, like `cron`, to periodically capture images every 10 seconds and have a separate Python script to process these images as they are saved. This method can be particularly efficient in managing system resources and can be more robust against potential delays in image processing.

Setting Up the Image Capture with `cron`

First, we should set up a `cron` job to use the `rpicam-jpeg` command to capture an image every 10 seconds.

1. **Edit the `crontab`**:

- Open the terminal and type `crontab -e` to edit the cron jobs.
- `cron` normally doesn't support sub-minute intervals directly, so we should use a workaround like a loop or watch for file changes.

2. **Create a Bash Script (`capture.sh`)**:
   - **Image Capture**: This bash script captures images every 10 seconds using `rpicam-jpeg`, a command that is part of the `raspijpeg` tool. This command lets us control the camera and capture JPEG images directly from the command line. This is especially useful because we are looking for a lightweight and straightforward method to capture images without the need for additional libraries like `Picamera` or external software. The script also saves the captured image with a timestamp.

```
#!/bin/bash
# Script to capture an image every 10 seconds

while true
do
  DATE=$(date +"%Y-%m-%d_%H%M%S")
  rpicam-jpeg --output test_images/$DATE.jpg --width 640 --height 640
  sleep 10
done
```

   - We should make the script executable with `chmod +x capture.sh`.
   - The script must start at boot or use a `@reboot` entry in `cron` to start it automatically.

Setting Up the Python Script for Inference

Image Processing: The Python script continuously monitors the designated directory for new images, processes each new image using the YOLOv8 model, updates the database with the count of detected bees, and optionally deletes the image to conserve disk space.

Database Updates: The results, along with the timestamps, are saved in an SQLite database. For that, a simple option is to use sqlite3.

In short, we need to write a script that continuously monitors the directory for new images, processes them using a YOLO model, and then saves the results to a SQLite database. Here's how we can create and make the script executable:

```
#!/usr/bin/env python3
import os
import time
import sqlite3
from datetime import datetime
from ultralytics import YOLO

# Constants and paths
IMAGES_DIR = 'test_images/'
MODEL_PATH = 'bee_landing_640_best_ncnn_model'
DB_PATH = 'bee_count.db'
```

```
def setup_database():
    """ Establishes a database connection and creates the table if it doesn't exist. """
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS bee_counts
        (timestamp TEXT, count INTEGER)
    ''')
    conn.commit()
    return conn

def process_image(image_path, model, conn):
    """ Processes an image to detect objects and logs the count to the database. """
    result = model.predict(image_path, save=False, imgsz=640, conf=0.2, iou=0.3,
verbose=False)
    num_bees = len(result[0].boxes.cls)
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    cursor = conn.cursor()
    cursor.execute("INSERT INTO bee_counts (timestamp, count) VALUES (?, ?)", (timestamp,
num_bees))
    conn.commit()
    print(f'Processed {image_path}: Number of bees detected = {num_bees}')

def monitor_directory(model, conn):
    """ Monitors the directory for new images and processes them as they appear. """
    processed_files = set()
    while True:
        try:
            files = set(os.listdir(IMAGES_DIR))
            new_files = files - processed_files
            for file in new_files:
                if file.endswith('.jpg'):
                    full_path = os.path.join(IMAGES_DIR, file)
                    process_image(full_path, model, conn)
                    processed_files.add(file)
            time.sleep(1)  # Check every second
        except KeyboardInterrupt:
            print("Stopping...")
            break

def main():
    conn = setup_database()
    model = YOLO(MODEL_PATH)
    monitor_directory(model, conn)
    conn.close()

if __name__ == "__main__":
    main()
```

The python script must be executable, for that:

1. **Save the script**: For example, as `process_images.py`.
2. **Change file permissions** to make it executable:

   ```
   chmod +x process_images.py
   ```

3. **Run the script** directly from the command line:

   ```
   ./process_images.py
   ```

We should consider keeping the script running even after closing the terminal; for that, we can use `nohup` or `screen`:

```
nohup ./process_images.py &
```

or

```
screen -S bee_monitor
./process_images.py
```

Note that we are capturing images with their own timestamp and then log a separate timestamp for when the inference results are saved to the database. This approach can be beneficial for the following reasons:

1. **Accuracy in Data Logging**:
   - **Capture Timestamp**: The timestamp associated with each image capture represents the exact moment the image was taken. This is crucial for applications where precise timing of events (like bee activity) is important for analysis.
   - **Inference Timestamp**: This timestamp indicates when the image was processed and the results were recorded in the database. This can differ from the capture time due to processing delays or if the image processing is batched or queued.
2. **Performance Monitoring**:
   - Having separate timestamps allows us to monitor the performance and efficiency of your image processing pipeline. We can measure the delay between image capture and result logging, which helps optimize the system for real-time processing needs.
3. **Troubleshooting and Audit**:
   - Separate timestamps provide a better audit trail and troubleshooting data. If there are issues with the image processing or data recording, having distinct timestamps can help isolate whether delays or problems occurred during capture, processing, or logging.

Script For Reading the SQLite Database

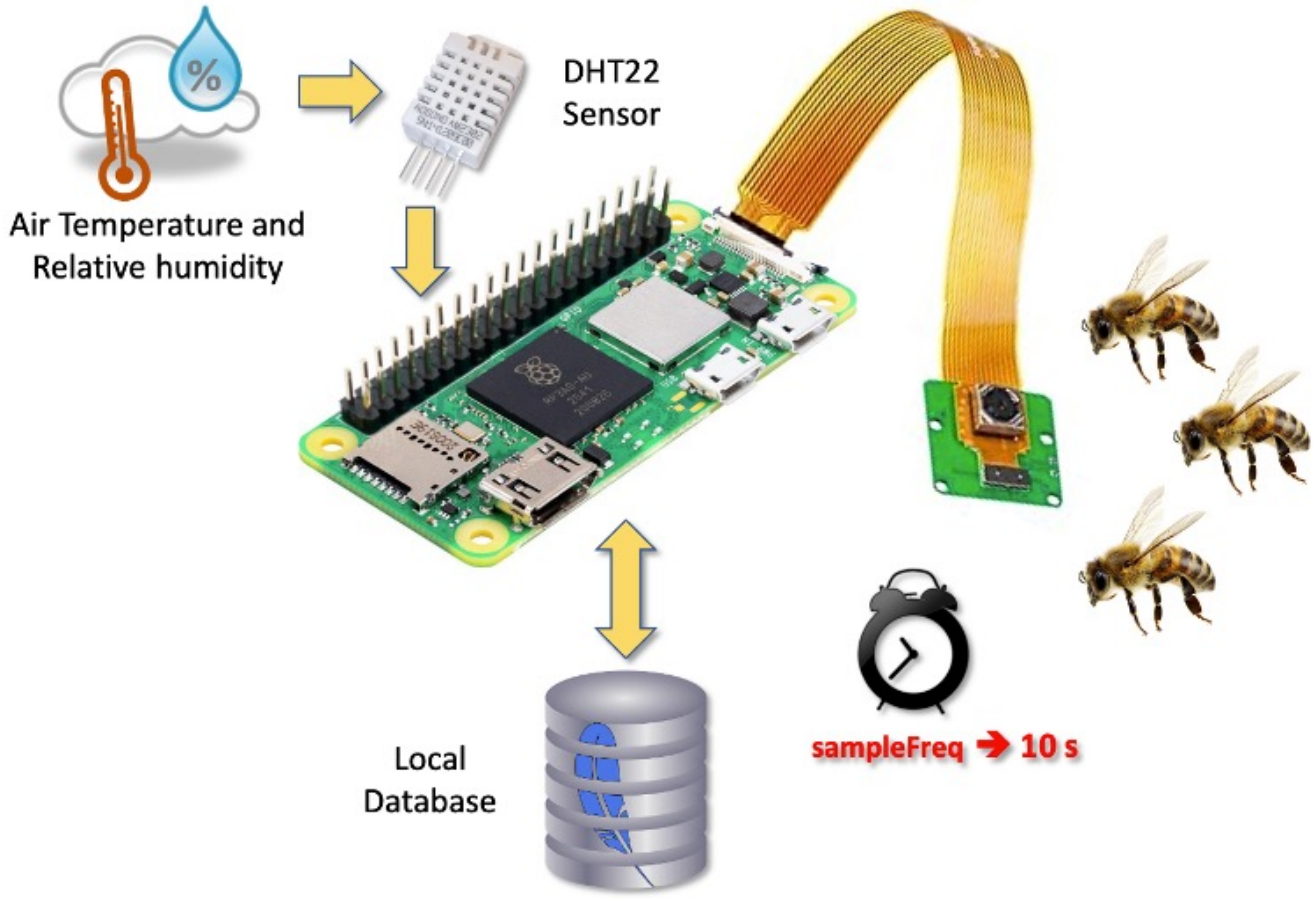
Here is an example of a code to retrieve the data from the database:

```
#!/usr/bin/env python3
import sqlite3
```

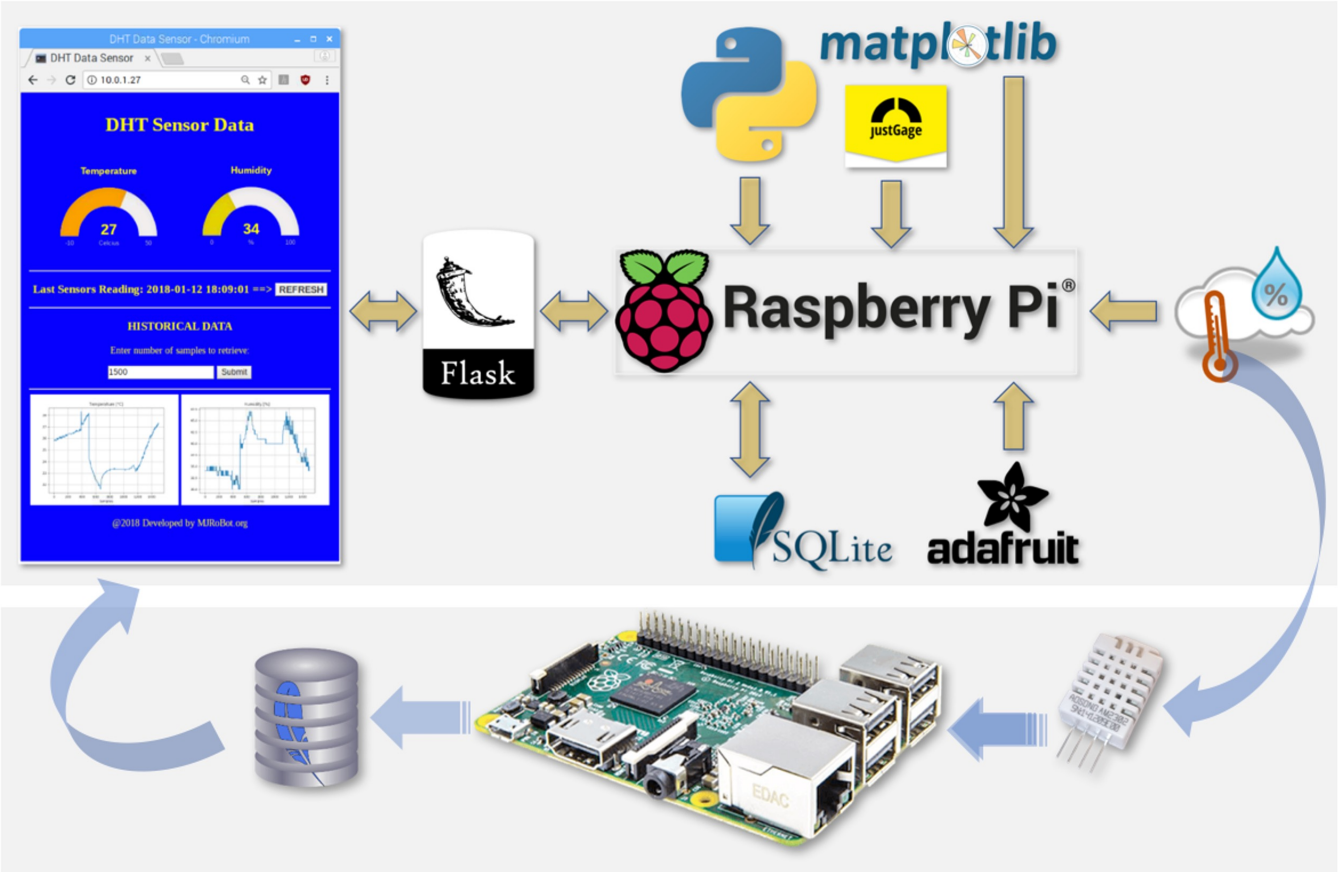
```
def main():
    db_path = 'bee_count.db'
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    query = "SELECT * FROM bee_counts"
    cursor.execute(query)
    data = cursor.fetchall()
    for row in data:
        print(f"Timestamp: {row[0]}, Number of bees: {row[1]}")
    conn.close()

if __name__ == "__main__":
    main()
```

Adding Environment data

Besides bee counting, environmental data, such as temperature and humidity, are essential for monitoring the bee-have health. Using a Rasp-Zero, it is straightforward to add a digital sensor such as the DHT-22 to get this data.



Environmental data will be part of our final project. If you want to know more about connecting sensors to a Raspberry Pi and, even more, how to save the data to a local database and send it to the web, follow this tutorial: [From Data to Graph: A Web Journey With Flask and SQLite].



Conclusion

In this tutorial, we have thoroughly explored integrating the YOLOv8 model with a Raspberry Pi Zero 2W to address the practical and pressing task of counting (or better, "estimating") bees at a beehive entrance. Our project underscores the robust capability of embedding advanced machine learning technologies within compact edge computing devices, highlighting their potential impact on environmental monitoring and ecological studies.

This tutorial provides a step-by-step guide to the practical deployment of the YOLOv8 model. We demonstrate a tangible example of a real-world application by optimizing it for edge computing in terms of efficiency and processing speed (using NCNN format). This not only serves as a functional solution but also as an instructional tool for similar projects.

The technical insights and methodologies shared in this tutorial are the basis for the complete work to be developed at our university in the future. We envision further development, such as integrating additional environmental sensing capabilities and refining the model's accuracy and processing efficiency. Implementing alternative energy solutions like the proposed solar power setup will expand the project's sustainability and applicability in remote or underserved locations.

> The Dataset paper, Notebooks, and PDF version are in the Project repository.

On the TinyML4D website, you can find lots of educational materials on TinyML. They are all free and open-source for educational uses—we ask that if you use the material, please cite it! TinyML4D is an initiative to make TinyML education available to everyone globally.