Counting bees with the LABRADOR board

Last month, we from the **TinyML4D** group had the opportunity to visit InovaUSP at Professor Marcelo Zuffo's invitation. In that opportunity, we could learn about the Labrador board, a 100% Brazilian Single Board Computer designed for the Internet of Things.

The Labrador is part of the Caninos Loucos Program.

The Bee Counting Project

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 explores the intersection of technology and nature. The project aims to estimate the number of bees entering and exiting their hive—a task crucial for beekeeping and ecological studies. So far, we have deployed a YOLOv8 model on the Raspberry Pi Zero 2W (*Raspi-Zero*), but on this project, we want to share our first findings on using the **Labrador board** for the task.

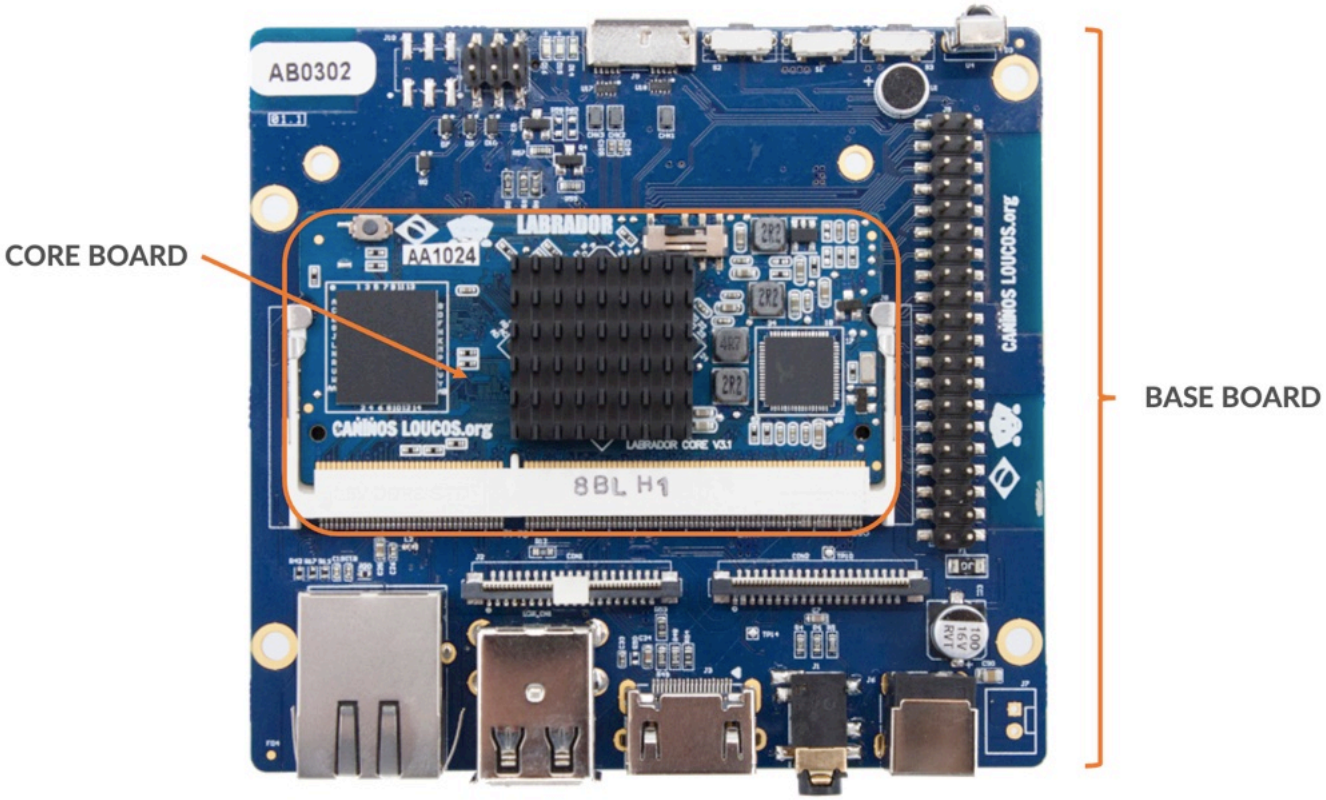
Why is this important? Bee populations are vital indicators of environmental health, and their monitoring can provide essential data for ecological research and conservation efforts. However, manual counting is labor-intensive and prone to errors. By leveraging the power of embedded machine learning,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

This tutorial will cover setting up the Labrador, optimizing and deploying YOLOv8 for real-time image processing, and analyzing the data gathered.

The Labrador Board

Specifications

Labrador is a single-board computer designed for the Internet of Things. It consists of 2 boards: the Labrador Core Board provides all the processing power and memory of a modern computer, and the Labrador Base Board expands the options for peripherals and communications, offering a variety of connectors. This most updated version comes with Bluetooth 5.0 and is ready for LoRaWAN connectivity.



CORE BOARD V3.0

CPU	64-bit quad-Core ARM Cortex A53 1,3GHz CPU (ARM v8 instruction set)
GPU	ARM Mali450 MP6 (4PP + 2GP). Supports: OpenGL-ES 1.1 e 2.0, OpenVG 1.1, EGL 1.5
Memory	2 GB LPDDR3 SDRAM 16GB eMMC
Operating System	Debian 11 Linux Kernel 4.19
PMU	ATC2306C – Power management and audio subsystem
Video	2160p@30fps ou 1080p@60fps supports video encoding (including H264, H263, MPEG-4)
Expansion	204 pins DDR3 SODIMM connector (male)
Button	ADFU
Dimensions	67.6 x 31.0 mm
Weight	10.1g

BASE BOARD M V2.0

Storage	MicroSD Card Slot SD/SDHC/SDXC
Ethernet	10/100Mbps (RJ45)
Wireless	Wi-Fi 802.11 b/g/n 2.4GHz Bluetooth LE 5.0 LoRaWAN™* 1 x infrared receptor (38kHz)*
USB	2 x USB2.0 HOST (type A) 1 x USB3.0 OTG (micro-B)*
Display	1 x HDMI 1.4 (type A), up to 1920×1080@60Hz 1 x LVDS-DSI para LCDs, up to 1920×1080@60Hz 1 x CVBS PAL/NTSC (PJ342 3,5mm)*
Audio	HDMI output* Analog stereo output (PJ342 3,5mm) I2S input/output Embedded microphone
Camera	1 x 8 bits parallel interface *
LED	1 x on/off (red) 1 x programmable (green) 1 x programmable (blue)
Buttons	1 x on/off 1 x reboot 1x programmable*
Power	5~12V @ 3W (internal diameter 2,1mm, external 5,5mm, positive center).
Expansion	204-pins SODIMM connector (female) 40-pin header: 28 GPIOs (compatible with Raspberry Pi / supports UART, I2C, SPI*, PWM and I2S) ADC input
Debug	UART
Dimensions	92,7 x 88.0 mm
Weight	59,4g

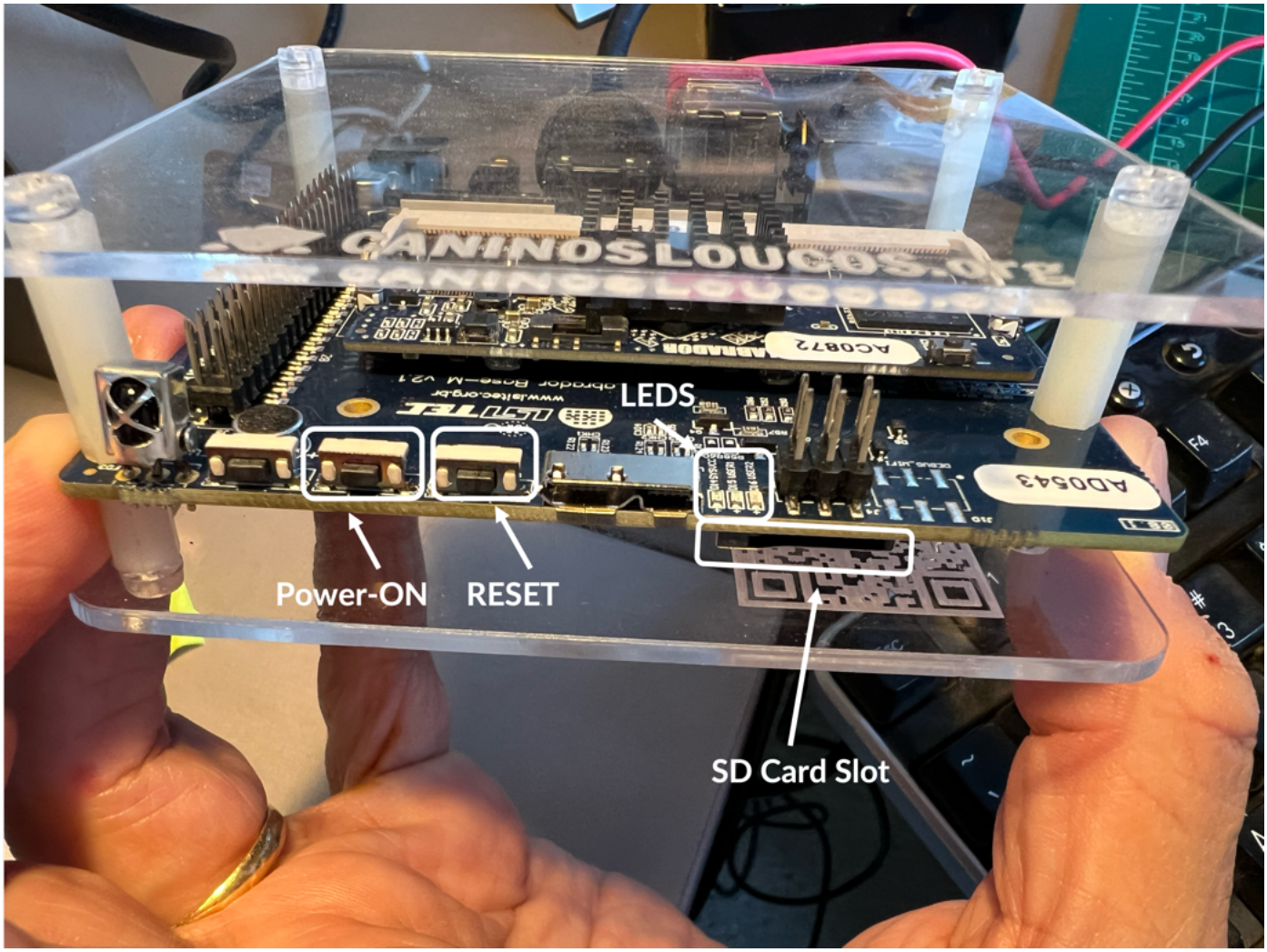
Power Supply

We are using 12v/2A to power the board, but for our final project, a smaller solar station should be designed for use in the field.

Installing the Operating System

The Labrador board (in our case, core V3.1) came with a pre-installed Linux version (4.19.98). You can test it with this installed Linux version (Debian 10) from a fresh board.

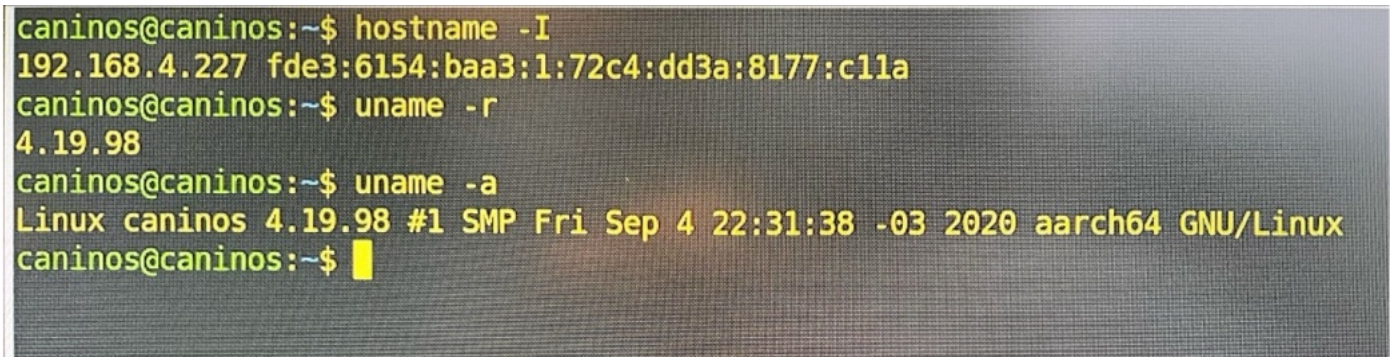
- Connect a USB keyboard/mouse and a HDMI monitor.
- Power on the Labrador.
- Press the **Base Board Power button** for a few seconds.



- The green/blue LED starts to flash.
- Enter with Debian credentials:

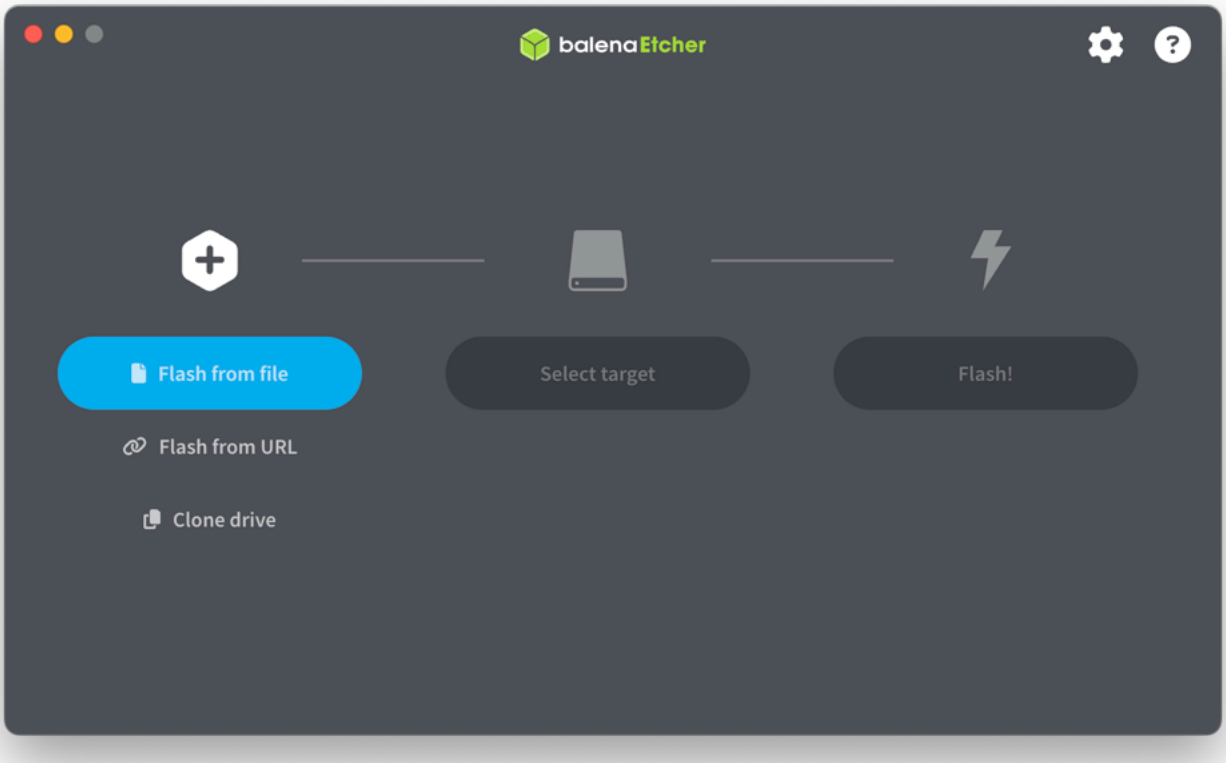
```
Username: caninos
Password: caninos
```

- Connect with the internet
- Open Terminal: [**ctrl]+[alt]+[T]**
- Check the version and verify the IP

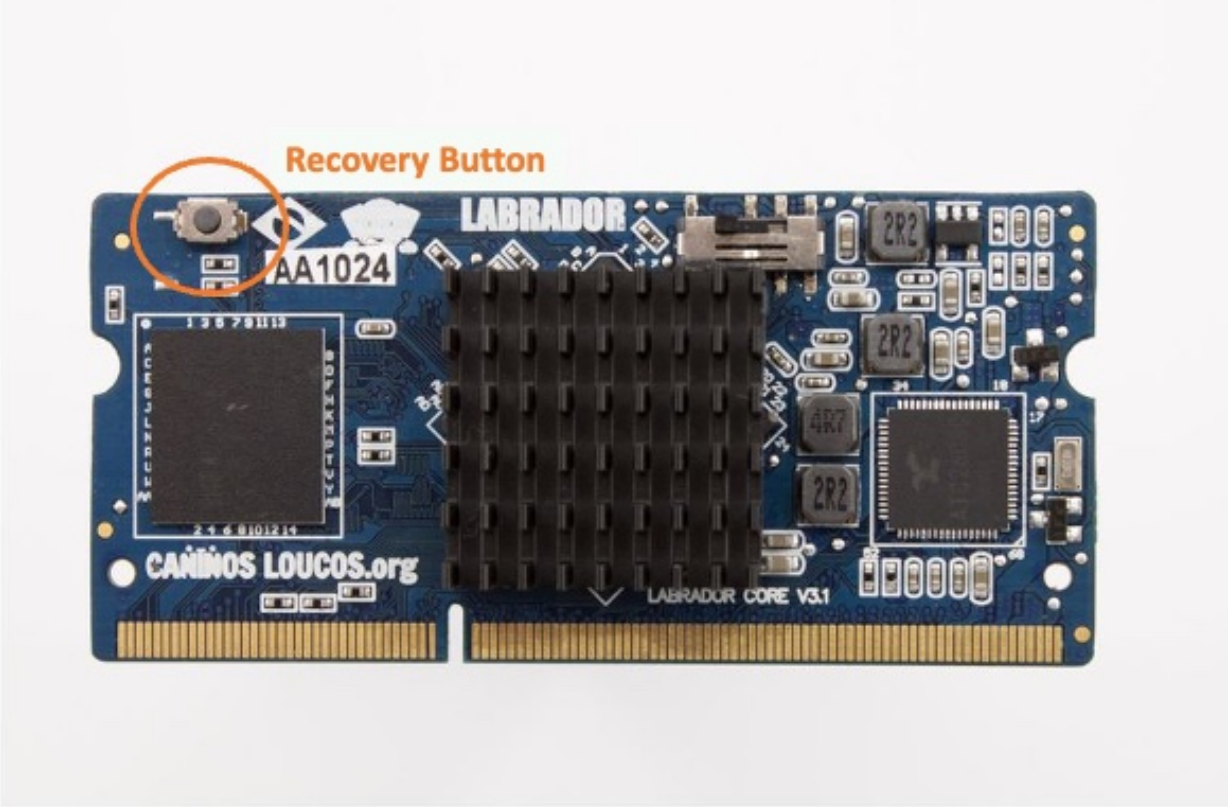


For our project, we should install the most updated 64-bit image available (**v0.10 - Jun 12, 2024**). The Kernel version was updated to 6.1.76, a new Bootloader (v2.4) was provided, and the Linux was upgraded to Debian 12, among other improvements.

For burning the image on an SD card, we can use, for example, the Etcher Balena, a tool for writing images on macOS, Windows, and Linux.



- Insert an **SD card** (at least 32GB) in your computer and **Flash** the image downloaded from Caninos Loucos website.
- Remove the Labrador from the power supply, and insert the burned SD card in the Labrador Base Board SD card slot reader.
- Press and hold the Core board recovery button (in the upper left corner of the Coreboard, next to the serial number label),



- Connect the power supply, wait about 3 seconds or until the logo appears on the screen, and release the recovery button.
- The system will boot from the card. To check, on the terminal, use the command:

  ```
  lsblk
  ```

  The device in `mmcblk0p1` (SD card) must be mounted in the system root, and the device `mmcblk2` (Labrador's internal memory) may or may not be mounted.

One of the significant advantages of the Labrador is its internal 16GB eMMC memory. So, let's install the OS on it:

- To install the system, open the terminal and run the command:

  ```
  cd install
  sudo ./install.sh
  ```

> If Labrador already has a system previously installed in its internal memory, we will need to confirm so that the update process can continue.

- Wait for the process to finish, remove the SD card, and restart the Labrador.

Let's check the installation:

```
uname -a
```

Linux labrador 6.1.76 #1 SMP Wed Jun 19 16:59:33 -03 2024 aarch64 GNU/Linux

The remote access server and utilities (SSH)

The Secure SHell (SSH) is a **secure** way to connect over the Internet. A free version of SSH called OpenSSH is available as `openssh-client` and `openssh-server` packages in Debian.

- Install the openSSH server:

```
sudo apt update
sudo apt install openssh-server
```

- Get the IP address:

```
hostname -I
```

In my case, 192.168.4.227

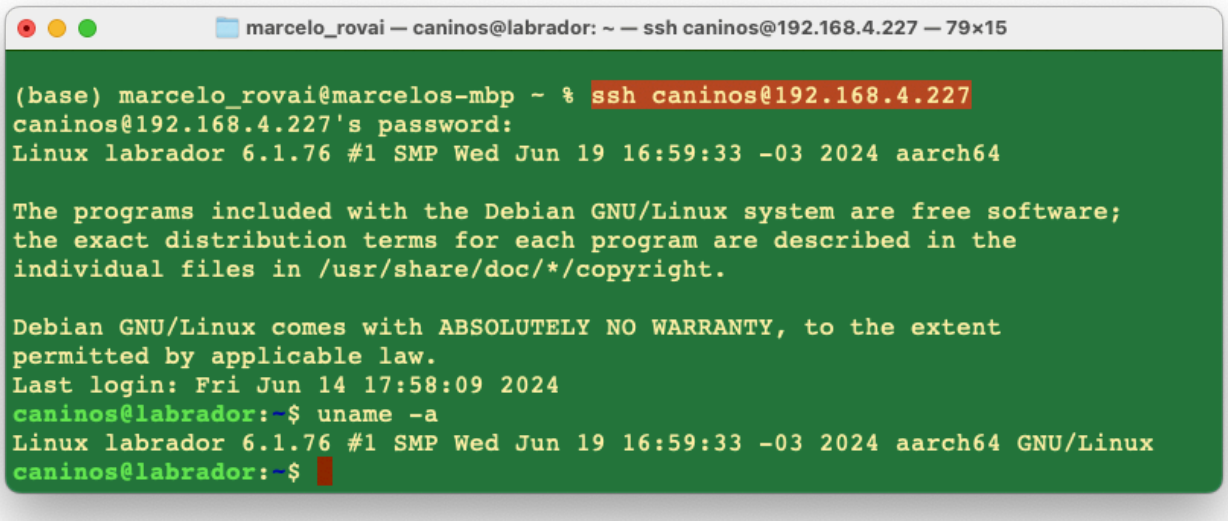
Now, it is possible to access the Labrador remotely for another computer. For example, in the terminal of my Mac, I should use:

```
ssh caninos@192.168.4.227
```

When you see the prompt:

```
caninos@labrador:~ $
```

It means that you interacting remotely with the Labrador.



> You can use PuTTy on Windows.

Notes

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 the Labrador. However, it has not been enabled in recent OS versions. To allow it, you should run the command:

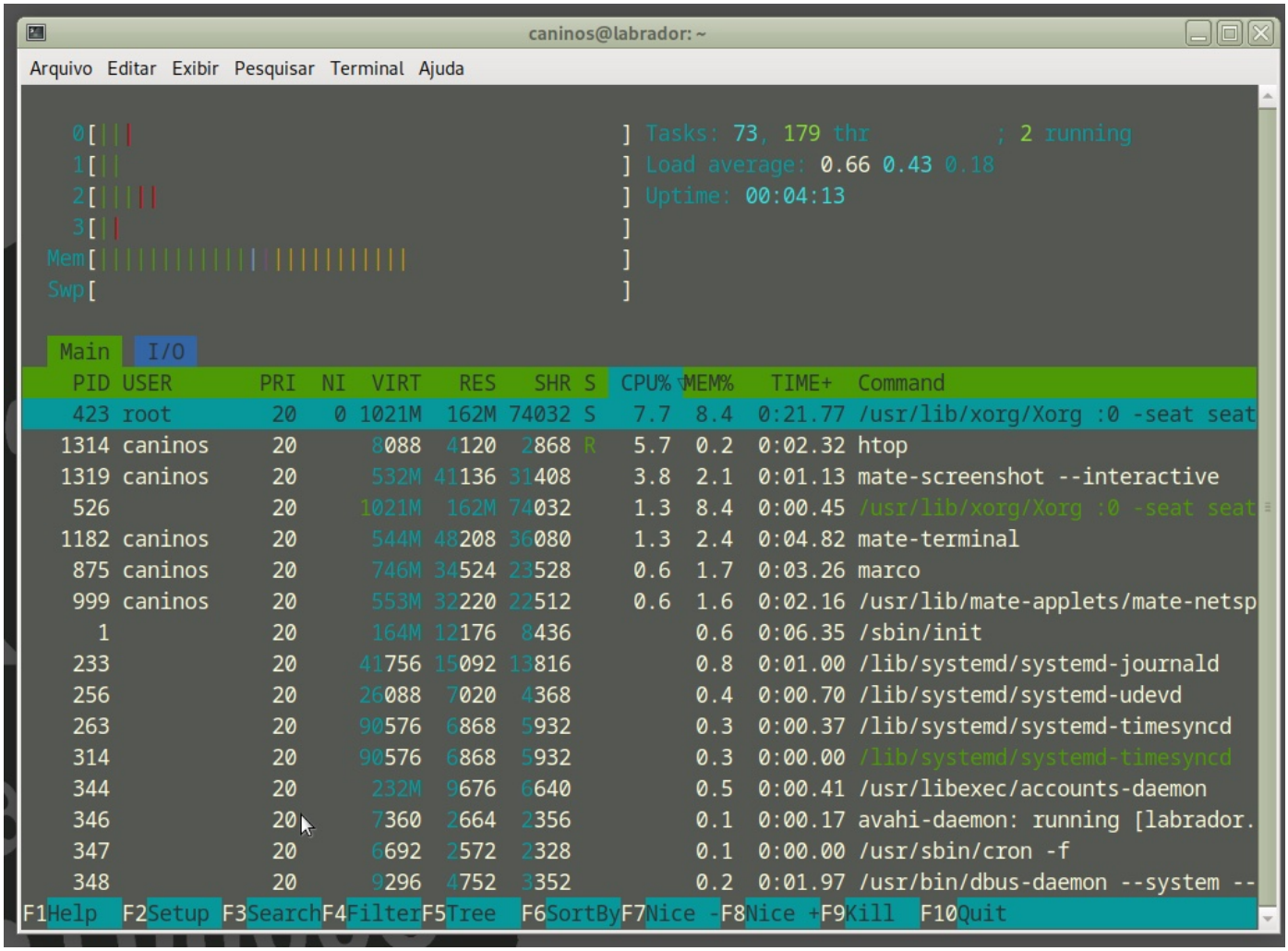
```
sudo rm /usr/lib/python3.11/EXTERNALLY-MANAGED
```

Also, when using SSH, it is possible to turn off the Labrador, using:

```
sudo shutdown -h now
```

To verify the resources of Labrador, install htop:

```
sudo apt install htop
htop
```



Transfer Files between the Labrador and a computer

The easiest way to see the Labrador 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.]

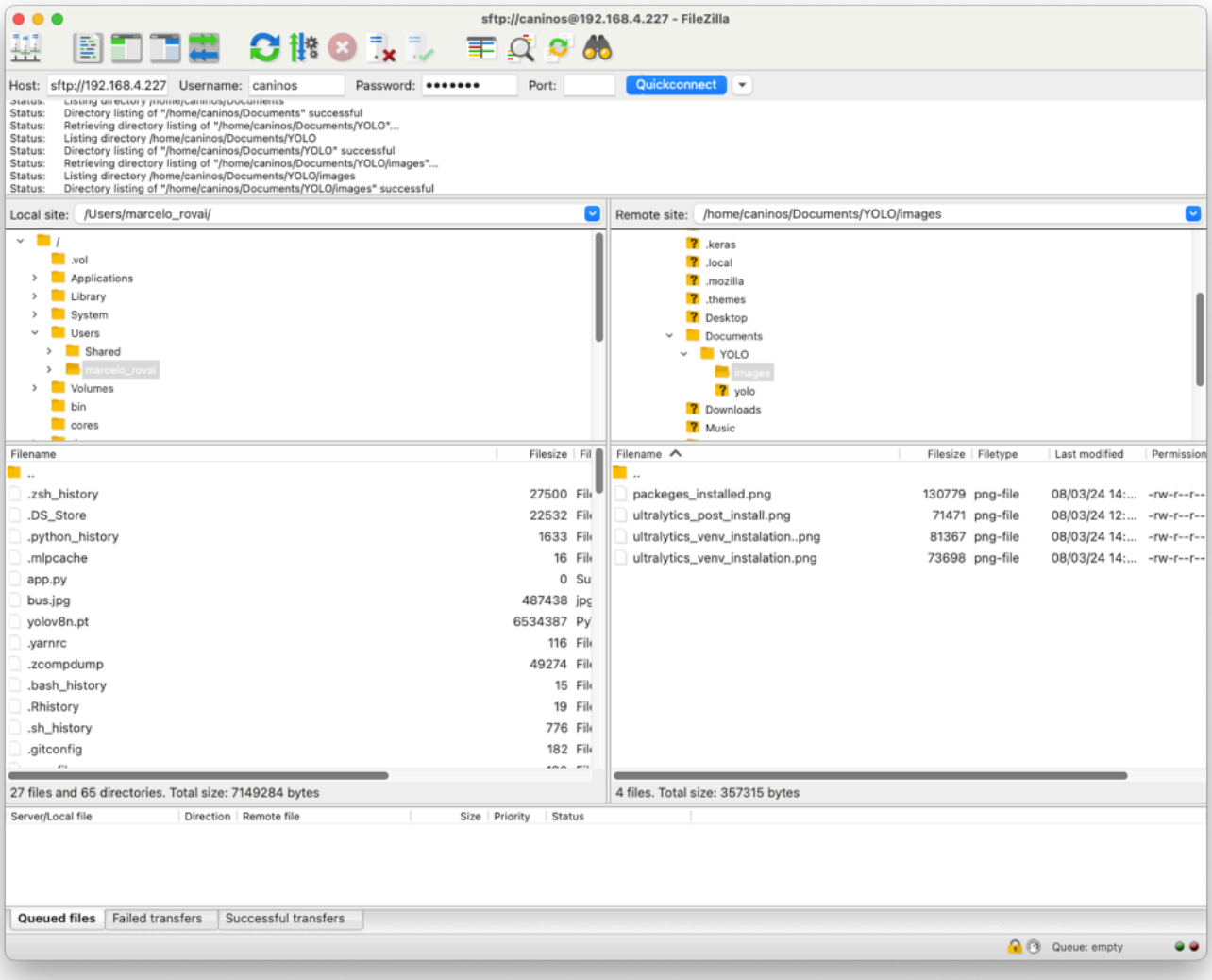
Follow the instructions and install the program for your Desktop OS.

It is essential to know what is the Labrador IP, for example, on the terminal use:

```
Ifconfig -a
```

And copy the IP address that appears with `wlan0: inet`. In my case, it is `192.168.4.227`.

Use this IP as the Host in the File: `sftp://192.168.4.227`, and enter the Labrador Username (caninos) and Password (caninos). Pressing `Quickconnect` will open two separate windows, one for your desktop and another for the Labrador.



The Ultralytics YOLOv8

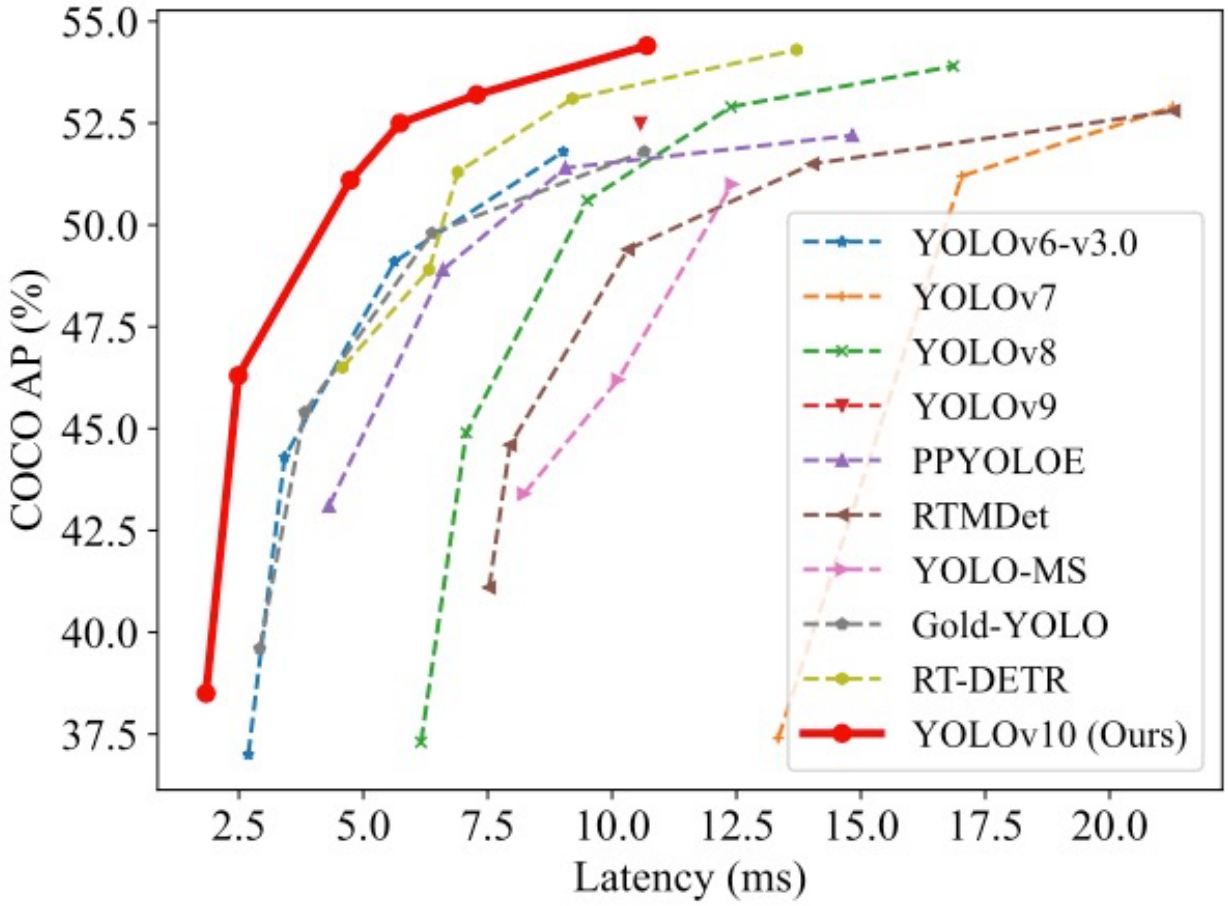
Ultralytics YOLOv8, is a version of the acclaimed real-time object detection and image segmentation model, YOLO. YOLOv8 is built on cutting-edge advancements in deep learning and computer vision, offering unparalleled performance in terms of speed and accuracy. Its streamlined design makes it suitable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

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 project, we leverage the power of YOLOv8 exported to TFLite format to estimate the number of bees at a beehive entrance using the Labrador in real-time. This demonstrates the practicality and effectiveness of deploying advanced machine learning models on edge devices for real-time environmental monitoring.

Installation

Before starting the installation, let's create a directory for working with YOLO on a Python environment (yolo) and change the current location to it:

```
sudo apt install python3.11-venv
mkdir Documents/YOLO
cd Documents/YOLO
python3 -m venv yolo
source yolo/bin/activate
```

Let's start installing the Ultralytics packages for local inference on the Labrador:

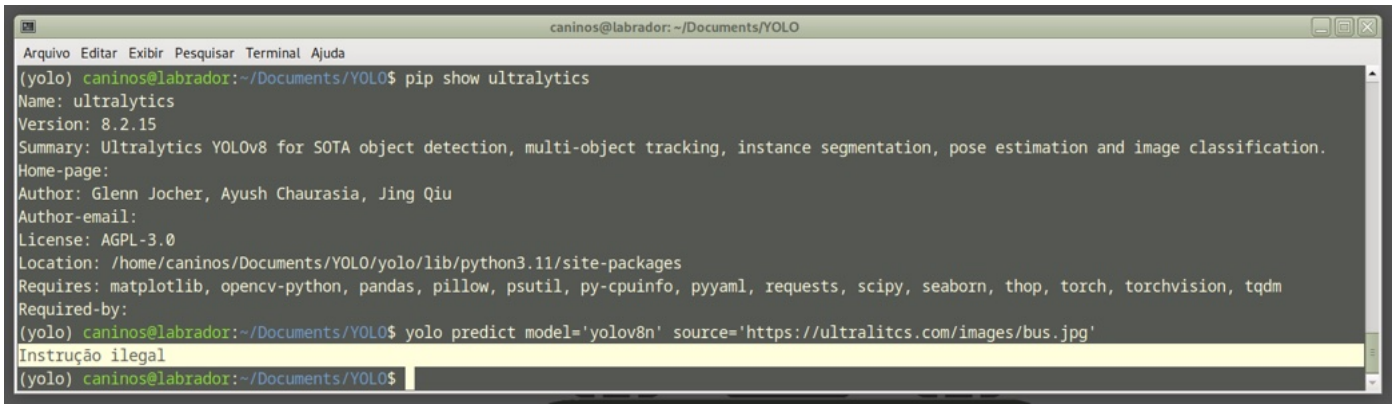
1. Update the packages list, install pip, and upgrade to the latest:

```
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==8.2.15
```

Debugging Process



The default installation does not work. The problem is with the "libarm_compute" library, which is part of the PyTorch installation. So, it is necessary to uninstall everything related to Pytorch and install one of its previous versions.

```
pip uninstall torch
pip uninstall torchvision
pip uninstall torchaudio

pip install torch==2.3.1 torchvision==0.18.1 torchaudio==2.3.1 --index-url
https://download.pytorch.org/whl/cpu
```

Reboot the Labrador:

```
sudo reboot
```

Testing the 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):

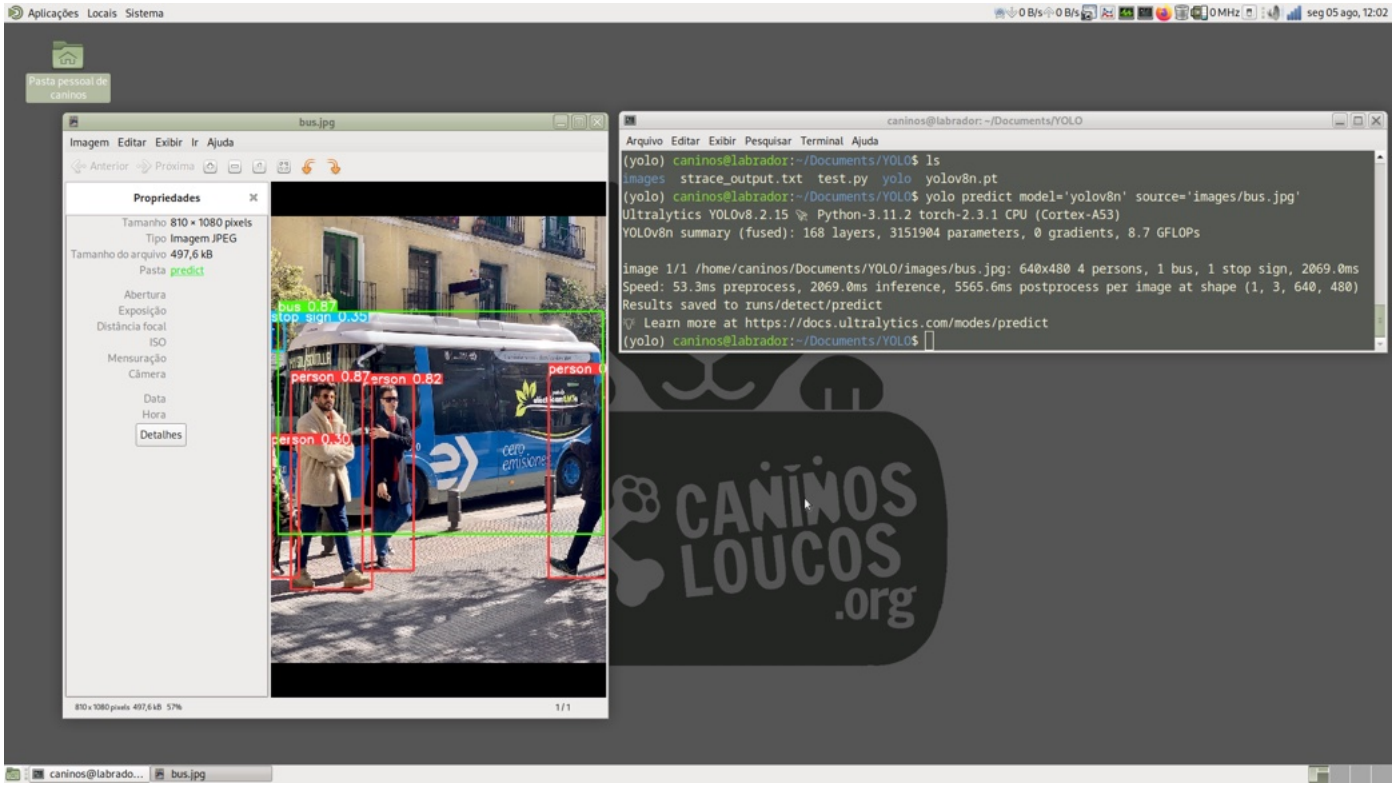
```
yolo predict model='yolov8n' source='https://ultralytics.com/images/bus.jpg'
```

Or you can use Filezilla to upload an image to the Labrador:

```
yolo predict model='yolov8n' source='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/predict.` Inspecting that directory, we can see a new image saved (bus.jpg).

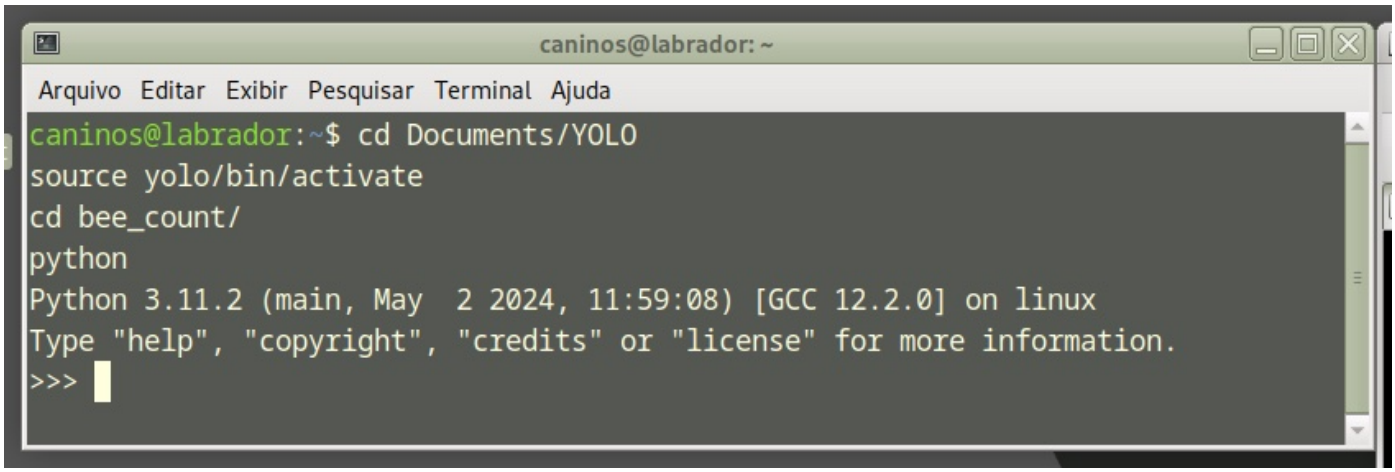


So, the Ultrayitics YOLO is correctly installed on our Labrador.

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
```

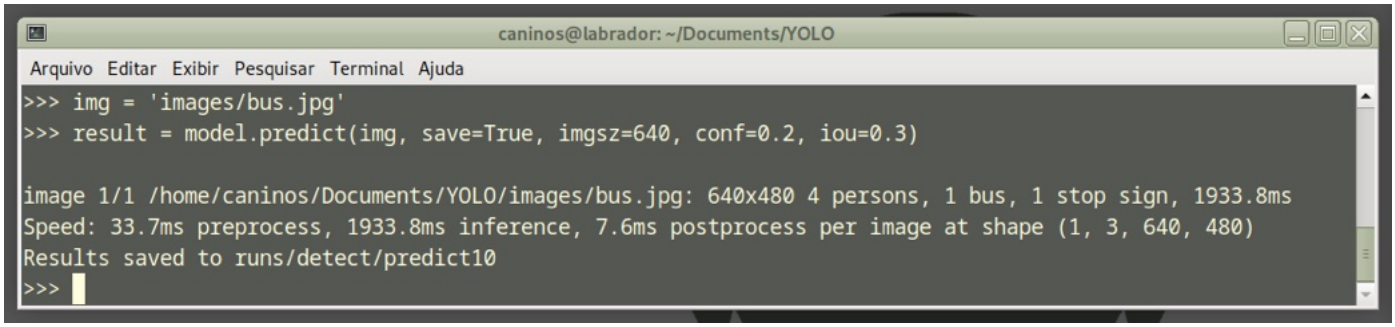


Now, we should call the YOLO library from Ultralitics and load the model:

```
import ultralytics
from ultralytics import YOLO
model = YOLO('yolov8n')
```

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

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

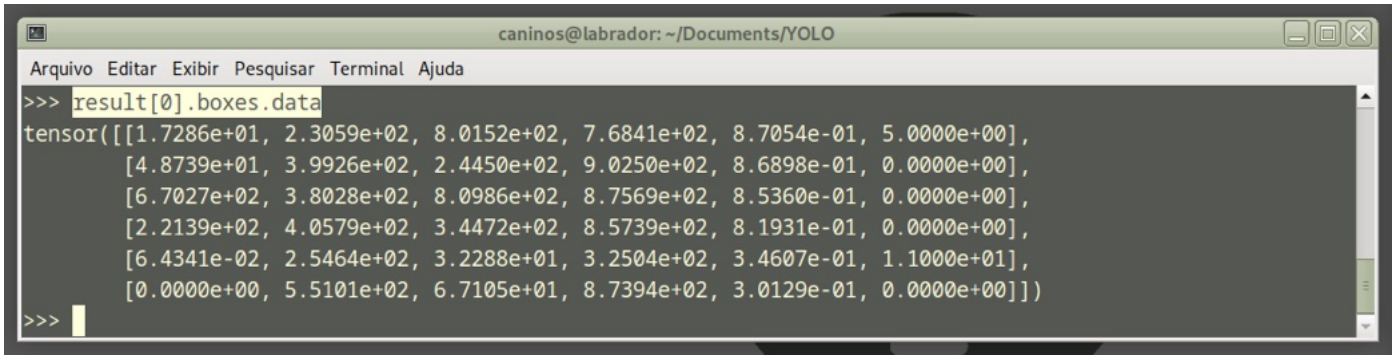


We can verify that the result is the same as the one we get running the inference at the terminal level (CLI).

```
image 1/1 /home/caninos/Documents/YOLO/images/bus.jpg: 640x640 4 persons, 1 bus, 1 stop
sign 1933.8ms
Speed: 33.7ms preprocess, 1933.8ms inference, 7.6ms postprocess per image at shape (1, 3,
640, 480)

Results saved to runs/detect/predict10
```

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 (6, 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`, `11: stop signal,` and `5: bus`):

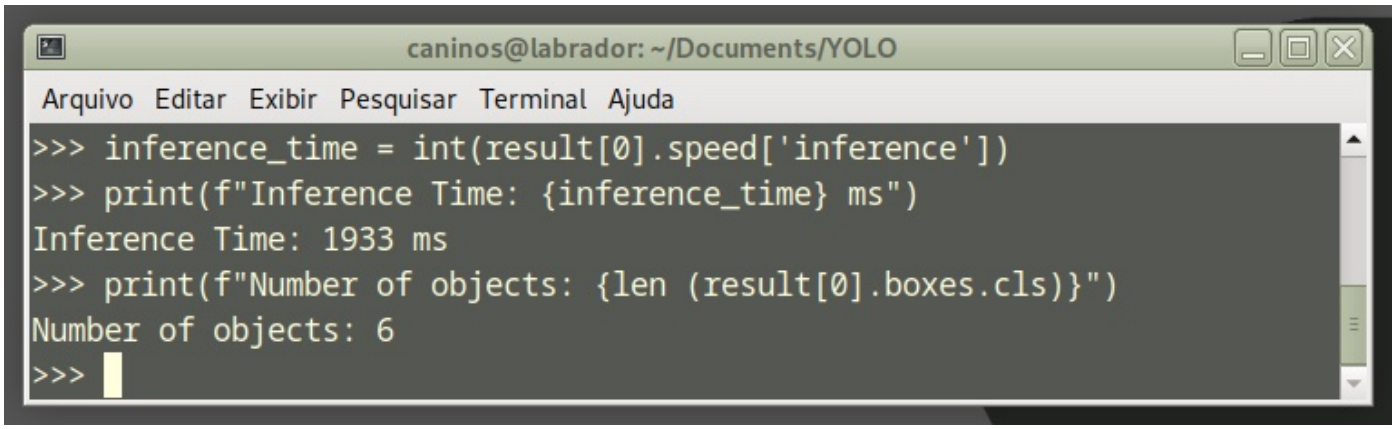


We can access several inference results separately, as the inference time, and have it printed in a better format:

```
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)}')
```



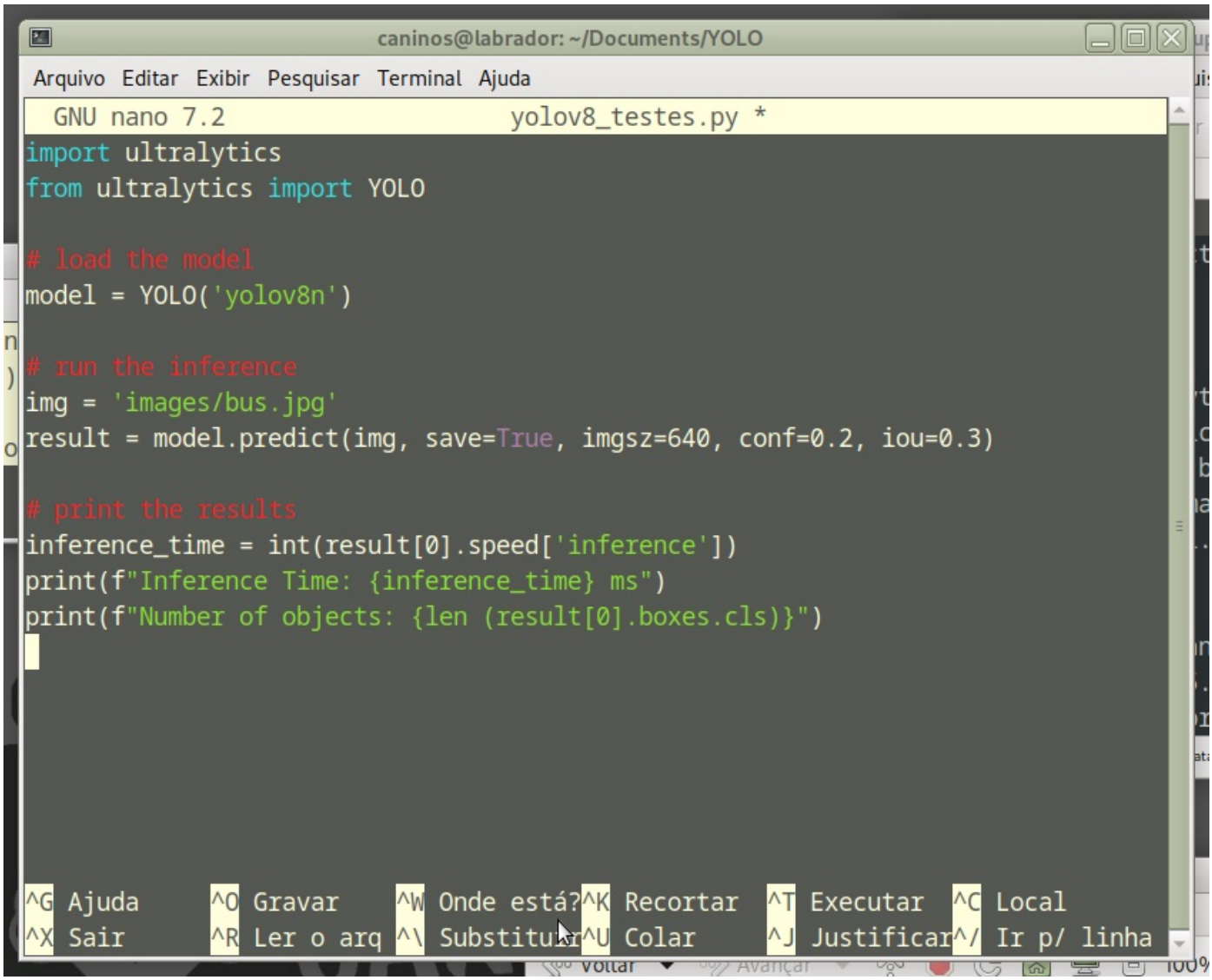
With Python, we can create a detailed output that meets our needs. In our final project, we will run a Python script at once rather than manually entering it line by line in the interpreter.

For that, let's use `nano` as our text editor. First, we should create an empty Python script named, for example, `yolov8_tests.py` (Let's do it using SSH for change):

```
nano yolov8_tests.py
```

Enter the code lines:

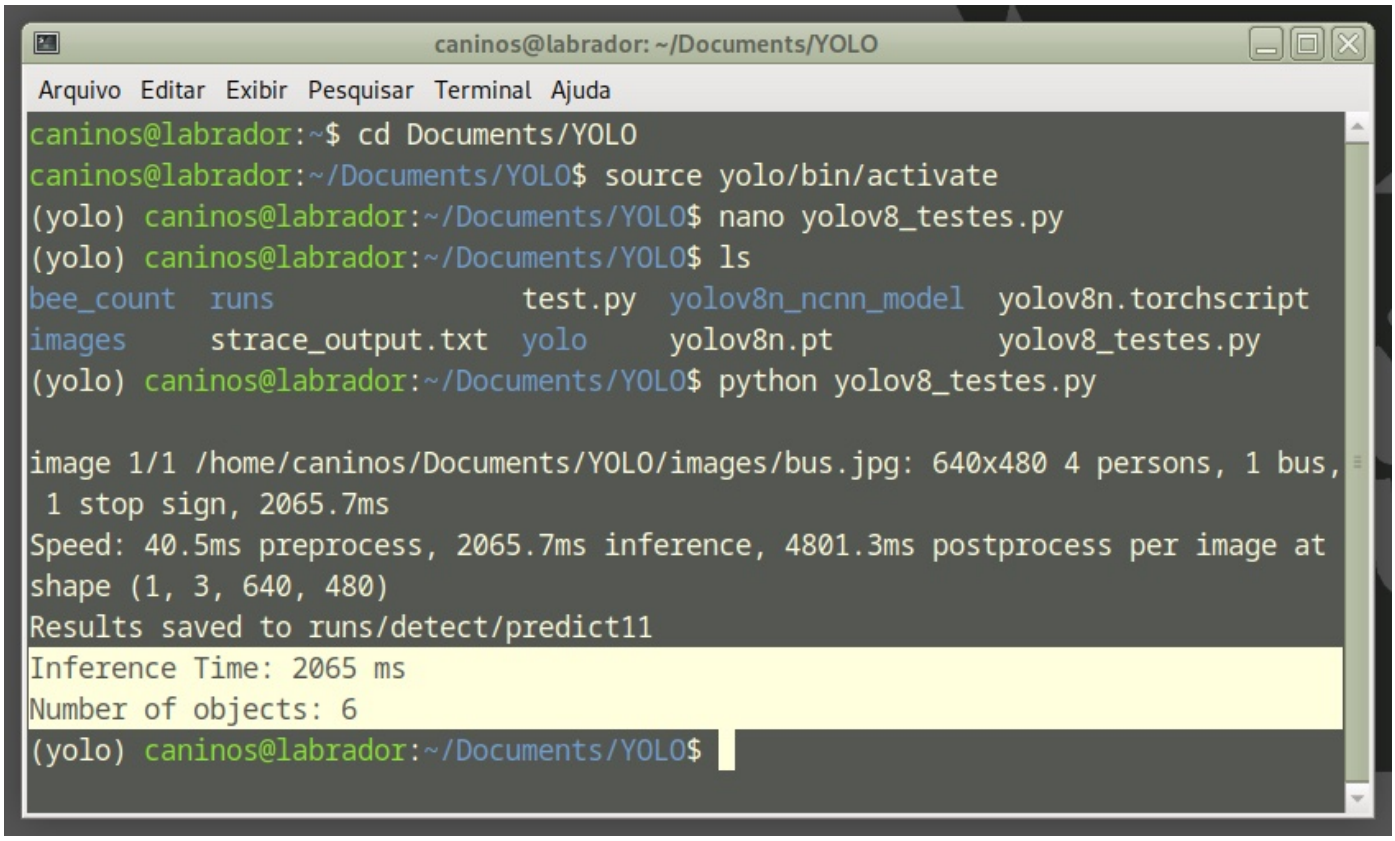
```
import ultralytics
from ultralytics import YOLO

# load the model
model = YOLO('yolov8n')

# run inference
img = '/images/bus.jpg'
result = model.predict(img, save=False, imgsz=640, conf=0.2, 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: 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.

Estimating the number of Bees

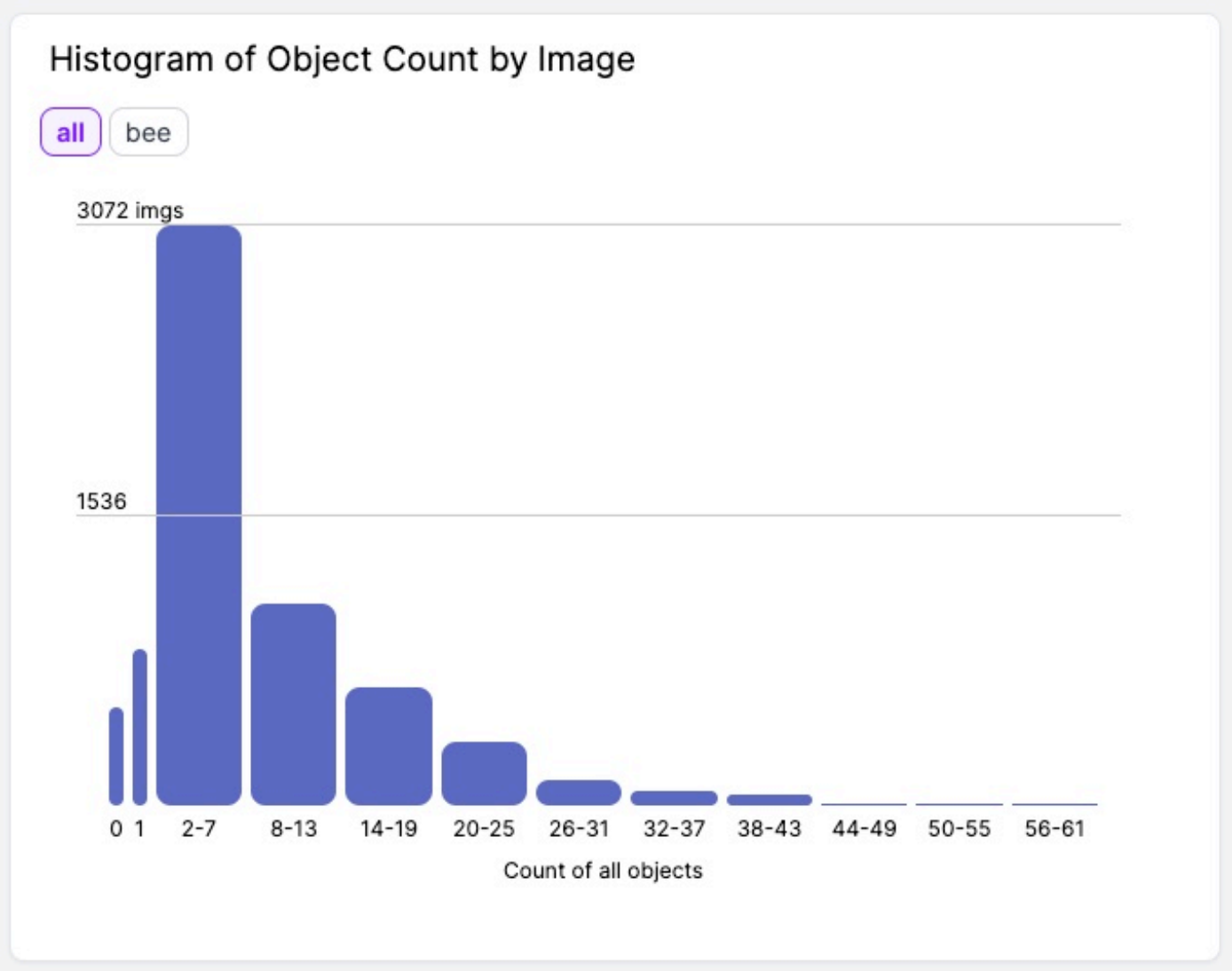
For our university project, we are preparing to collect a dataset of bees at the entrance of a beehive using a camera installed on the Labrador. The images should be collected every **10 seconds**. With a standard USB Cam, the horizontal Field of View (FoV) is 53.5o, which means that a camera positioned at the top of a standard Hive (46 cm) will capture all of its entrances (about 47 cm).

Dataset

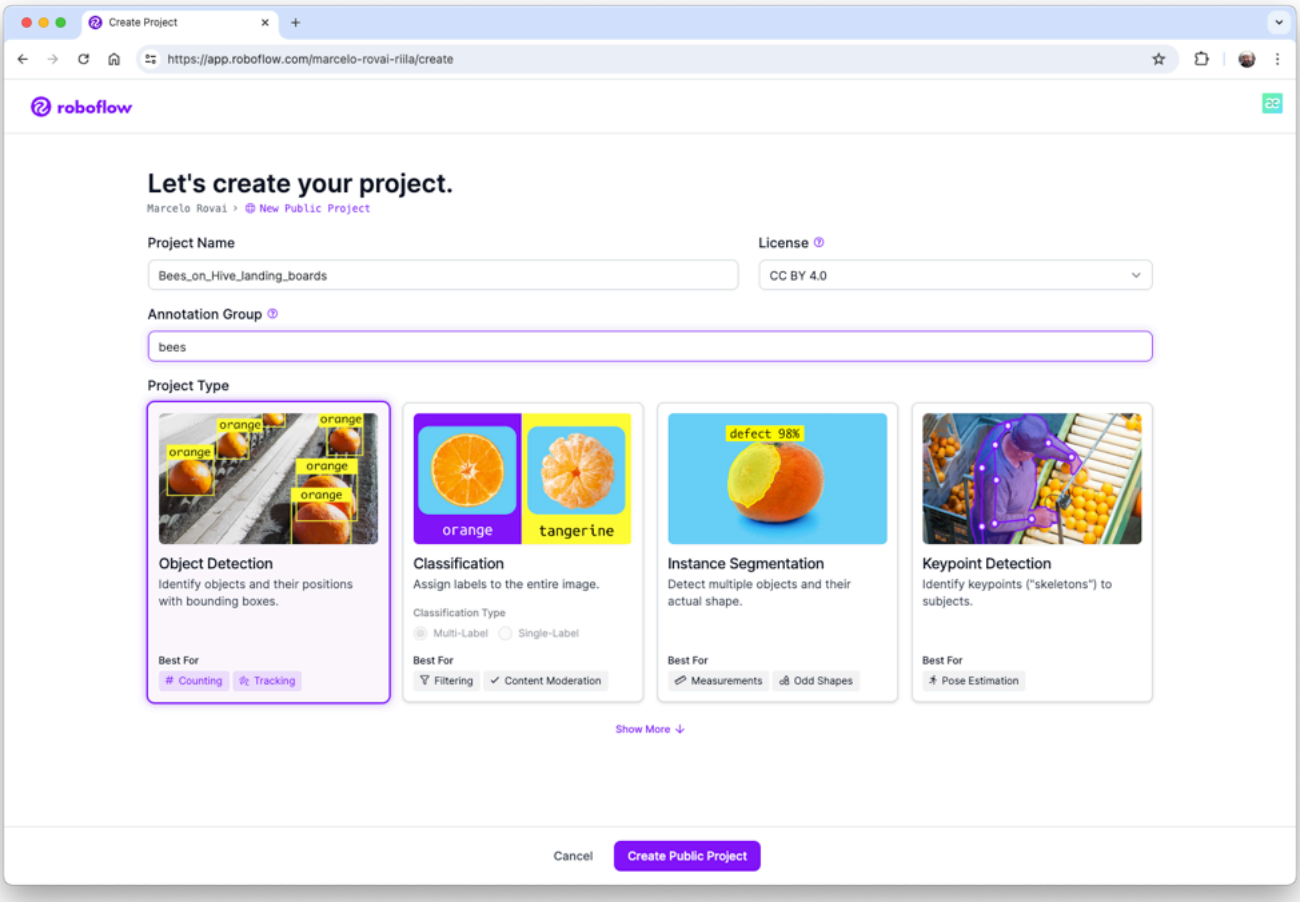
The dataset collection is the most critical phase of the project and should take several weeks or months. For this project, we will use a public dataset: "Sledevic, Tomyslav (2023), “[Labeled dataset for bee detection and direction estimation on beehive landing boards,” Mendeley Data, V5, doi: 10.17632/8gb9r2yhfc.5"

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:

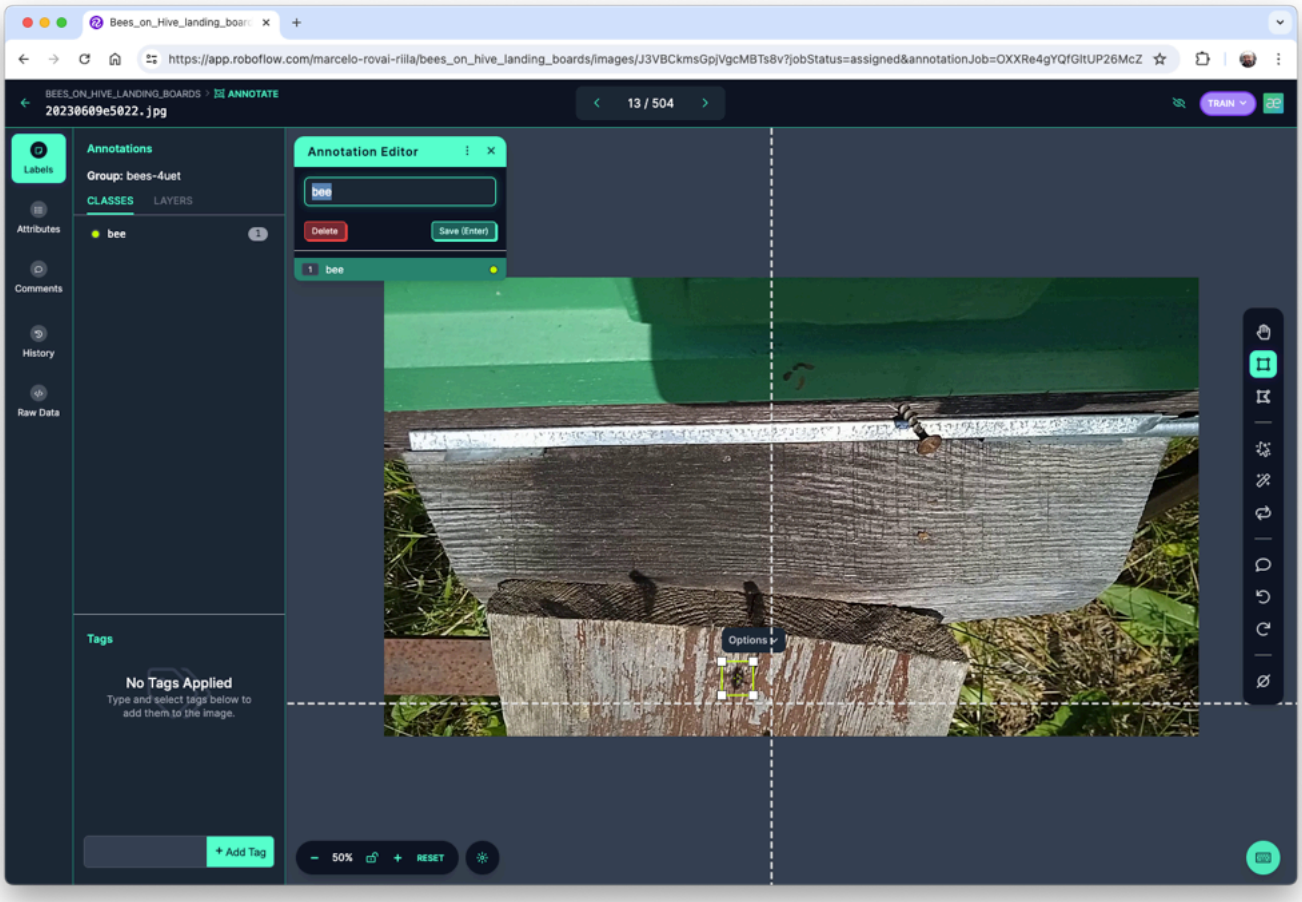


We downloaded the dataset (images and annotations) and uploaded it to Roboflow. There, you should create a free account and start a new project, for example, ("Bees_on_Hive_landing_boards"):

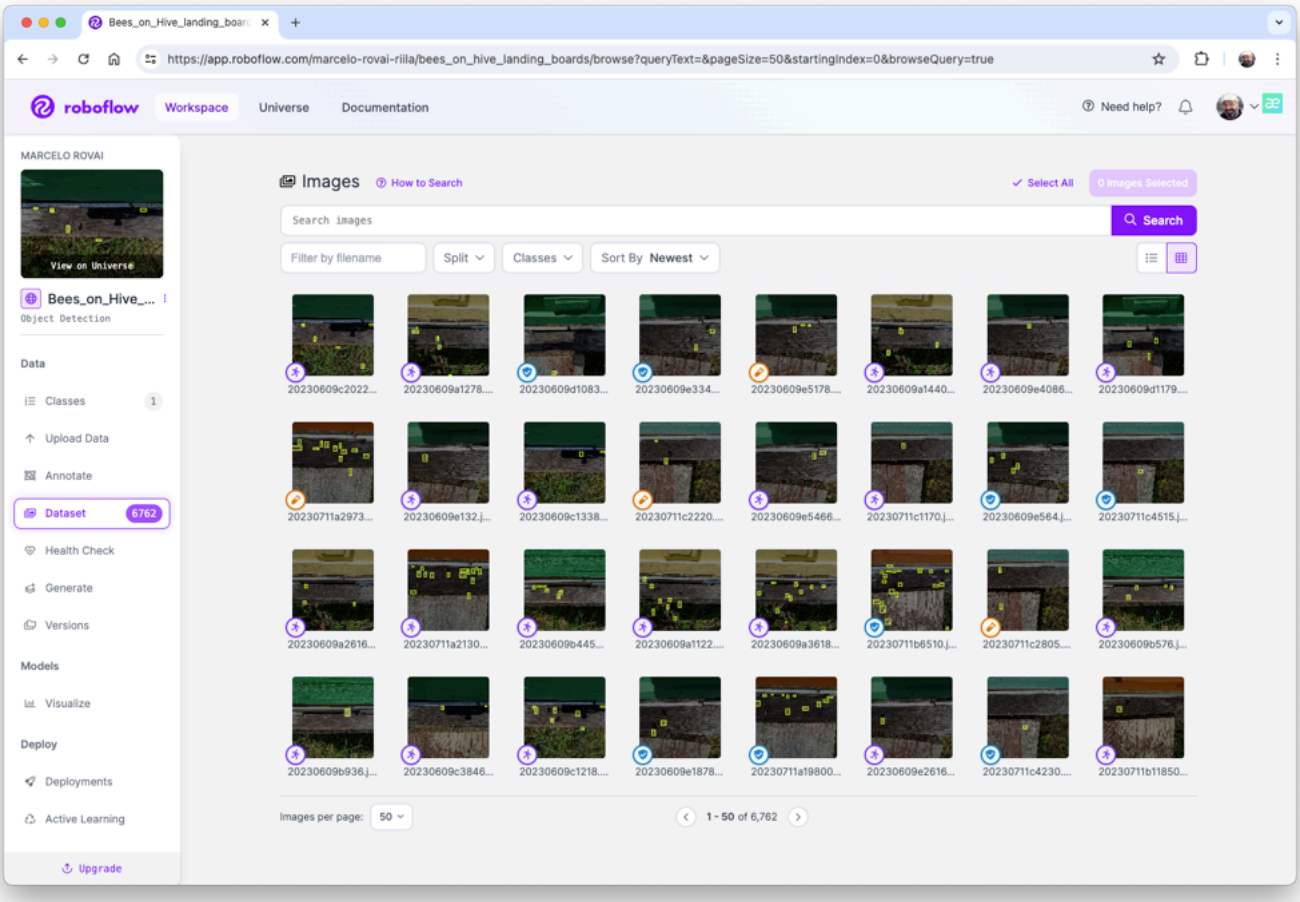


> We will not enter details about the Roboflow process once many tutorials are available.

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.



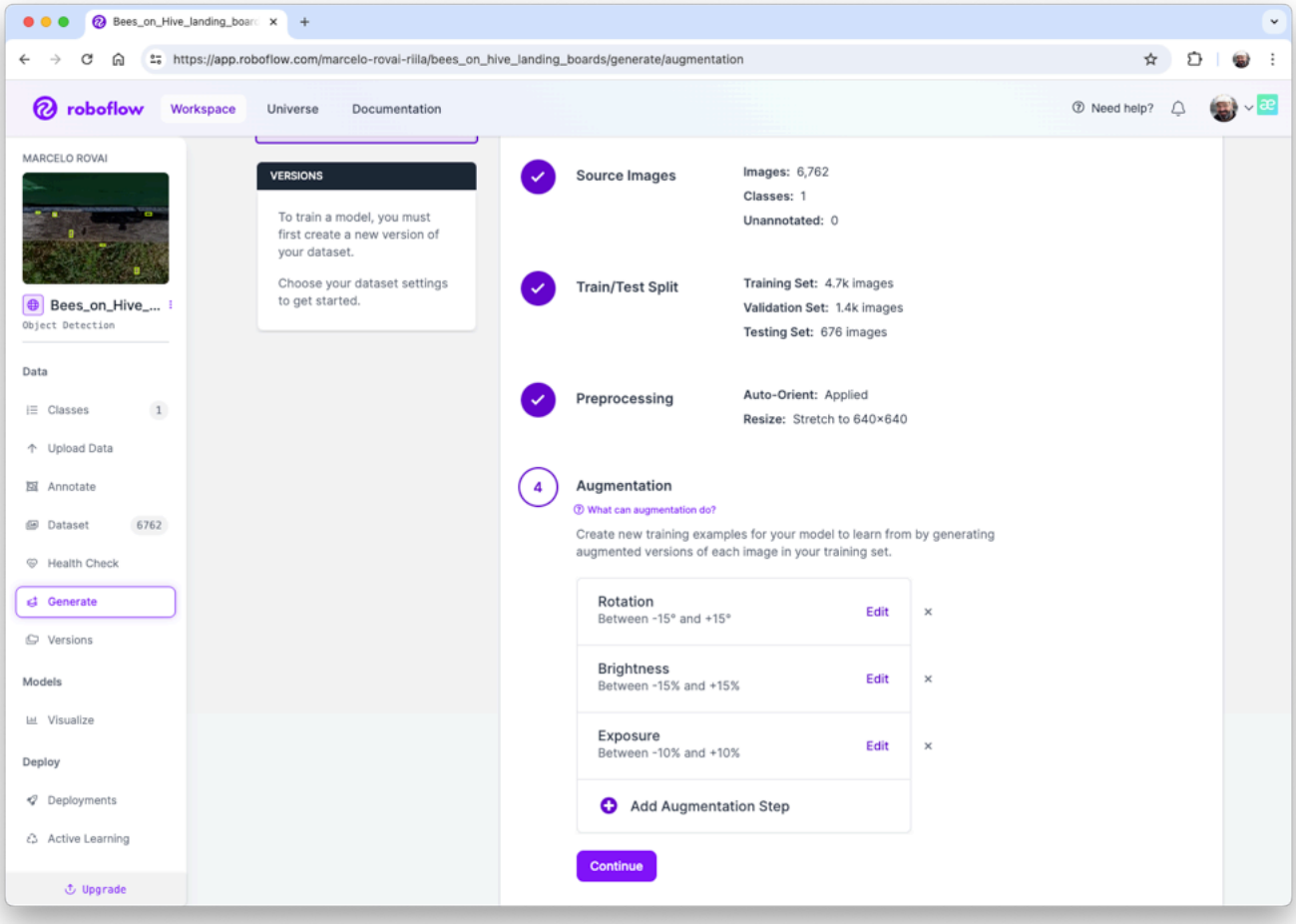
Once all images are annotated, you should split them into training, validation, and testing.



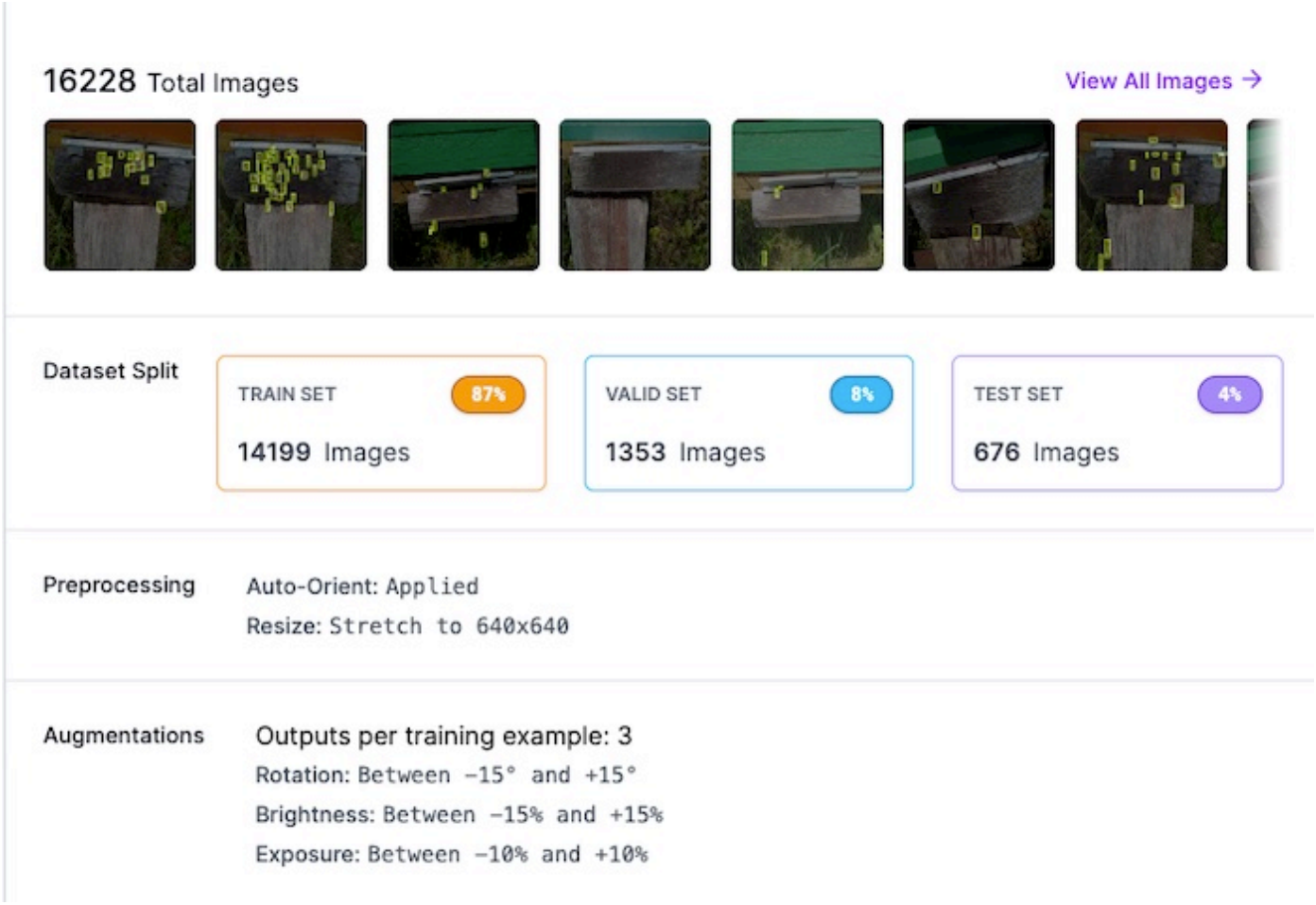
Pre-Processing

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.

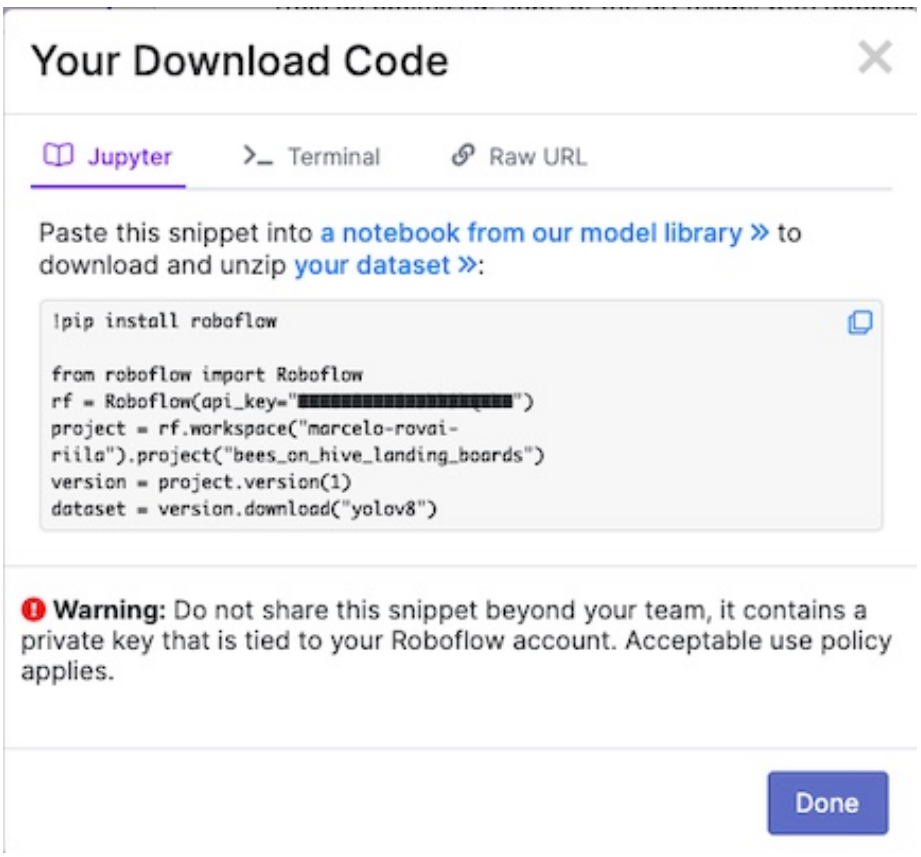
For augmentation, we will rotate the images (+/-15^{o}) and vary the brightness and exposure.



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

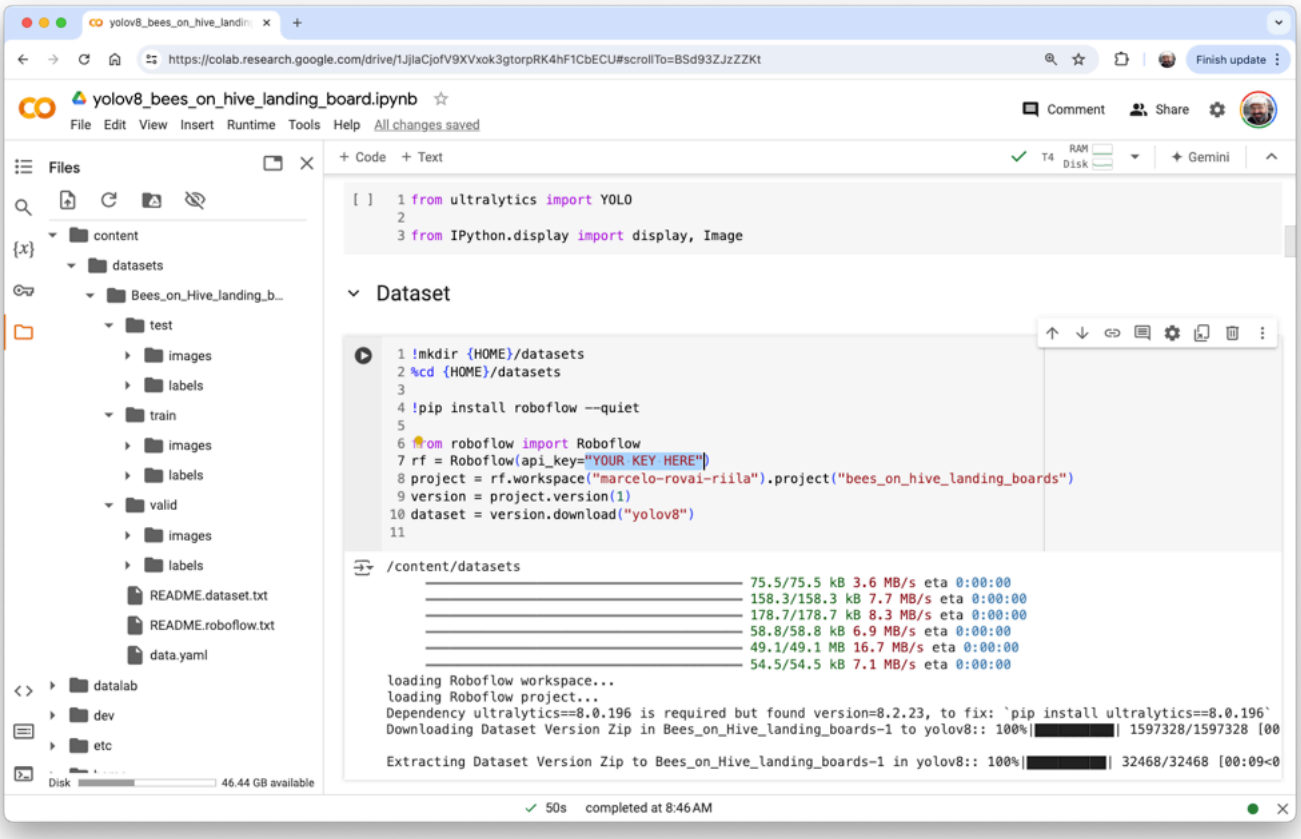
For training, let's adapt one of the public examples available from Ultralitytics and run it on Google Colab:

- 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 essential 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:

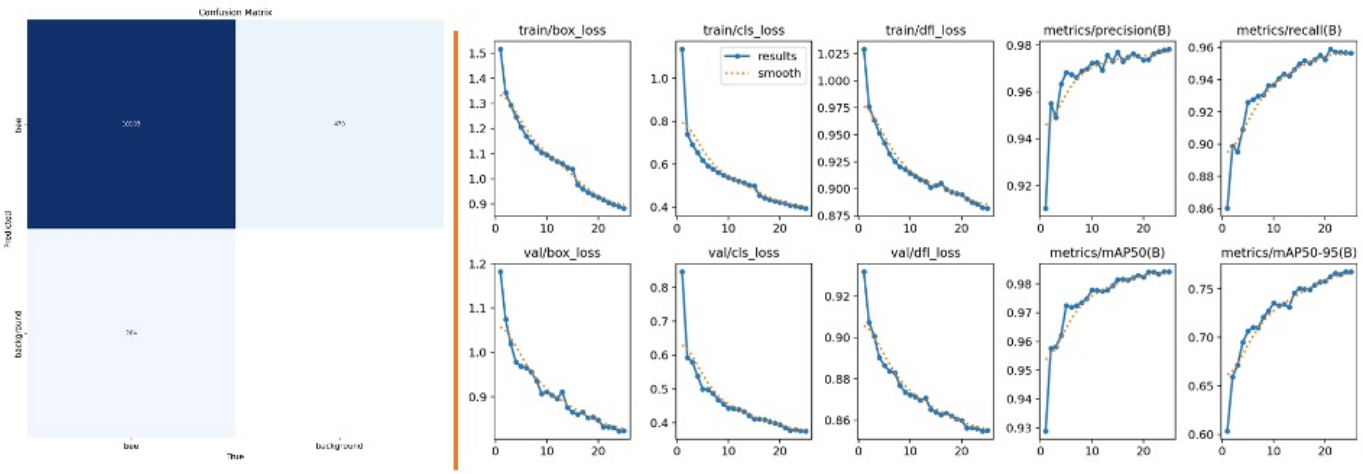
```
MODEL = 'yolov8n.pt'
IMG_SIZE = 640
EPOCHS = 25 # For a final project, you should consider at least 100 epochs
```

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. The trained model (`best.pt`) is saved in the folder `/runs/detect/train3/weights/`. Now, you should validate 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
```

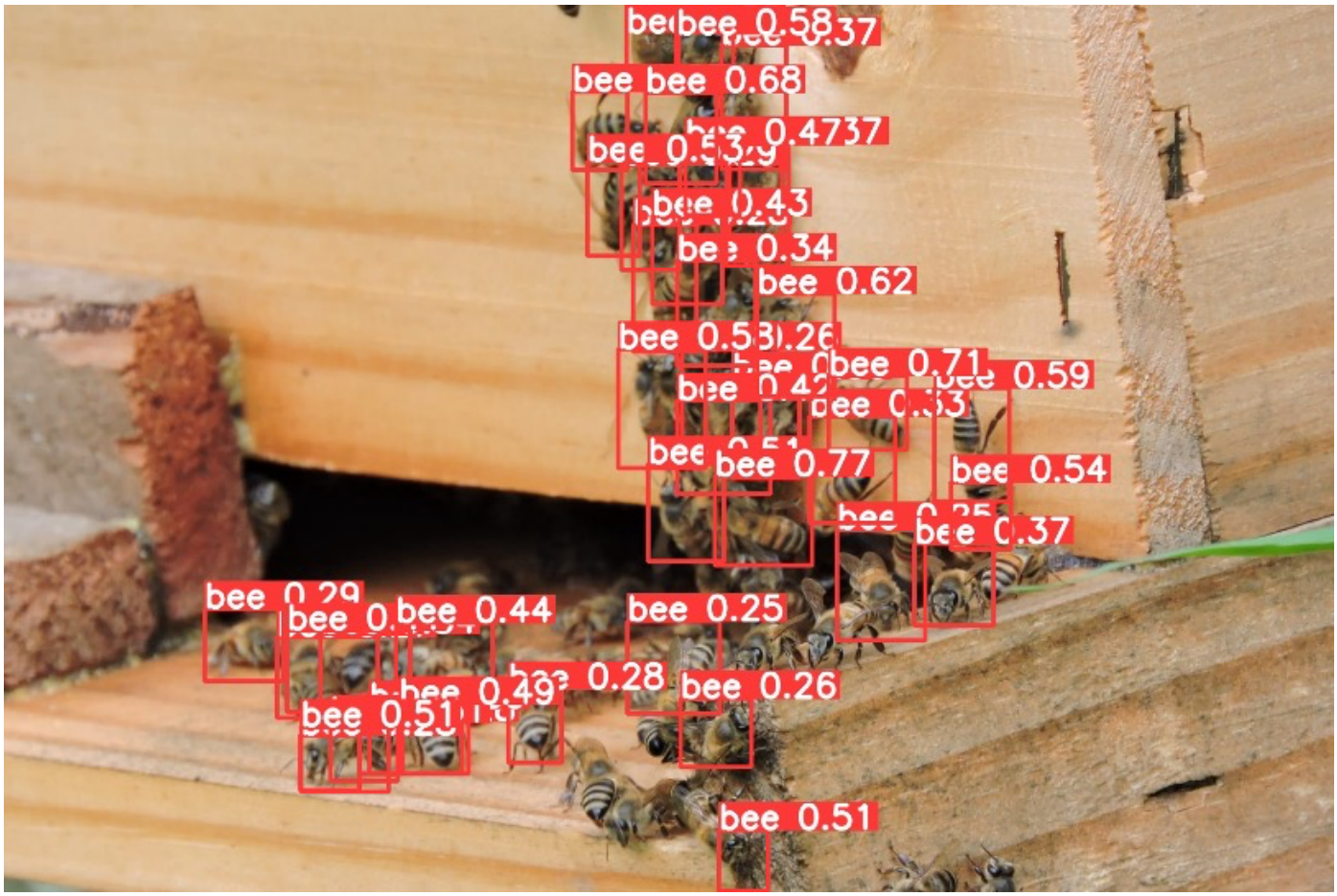
   The results were similar to training.

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 could have been better and with a lower confidence score). The model found 41 bees.



9. Let's export the train, validation, and test results for a Drive at Google. To do so, we should mount our drive.

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

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

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

Export a Trained model to TFLite format

Deploying computer vision models on edge or embedded devices requires a format that can ensure seamless performance. The TensorFlow Lite or TFLite export format allows us to optimize Ultralytics YOLOv8 models for tasks like object detection and image classification in edge device-based applications.

Implementing with Embedded Linux: To accelerate inference times when running inferences on the Labrador, we can use an exported TFLite model.

When we trained the model, the conversion to TFLite was only available for TensorFlow 2.13.1. So, we should install it.

```
!pip uninstall tensorflow -y
!pip install  tensorflow==2.13.1
```

And verify if the installation was OK:

```
import tensorflow as tf
print(tf.__version__)
```

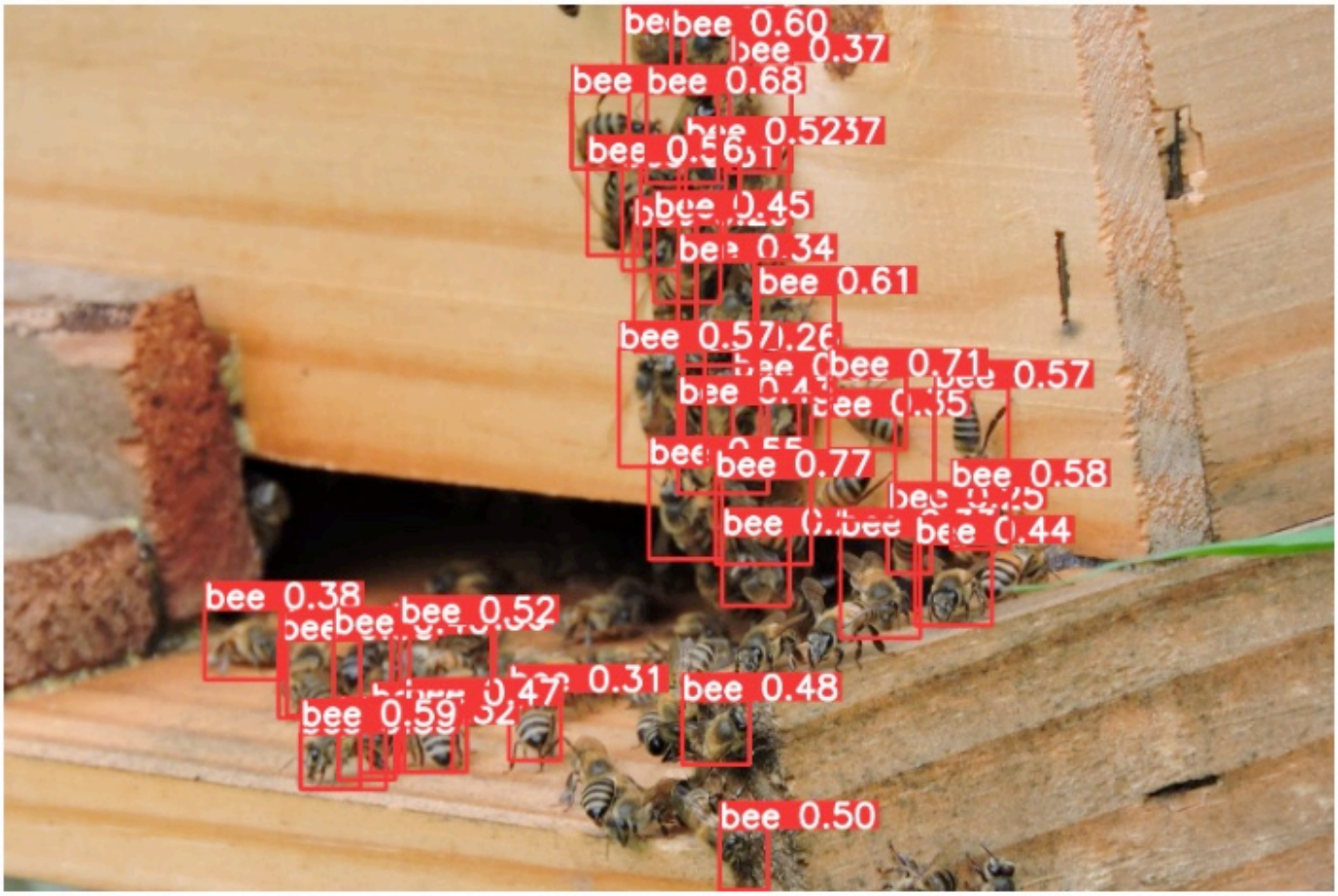
And run the command to create a float32 and float16.tflite models.

```
!yolo export model={HOME}/runs/detect/train3/weights/best.pt format=tflite
```

And let's us test the converted float16.tflite model with the last image:

```
!yolo predict model={HOME}/runs/detect/train3/weights/best_saved_model/best_float16.tflite
source=maurilio-bee.jpeg
```

```
from IPython.display import display, Image
display(Image(filename='runs/detect/predict5/maurilio-bee.jpeg', width=640))
```



The result is very similar.

Save the converted models to be used with the Labrador.

Inference with the trained model, using the Labrador

Using the FileZilla FTP, let's transfer the models to our Labrador (before the transfer, we should change the model name, for example, `bee_landing_640_float16.tflite`).

Let's create a new directory (`bee_count`)

```
mkdir test_images
cd bee_count
```

and a folder to receive some test images (under `Documents/YOLO/bee_count`):

```
mkdir test_images
```

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

Let's use the Python Interpreter:

```
python
```

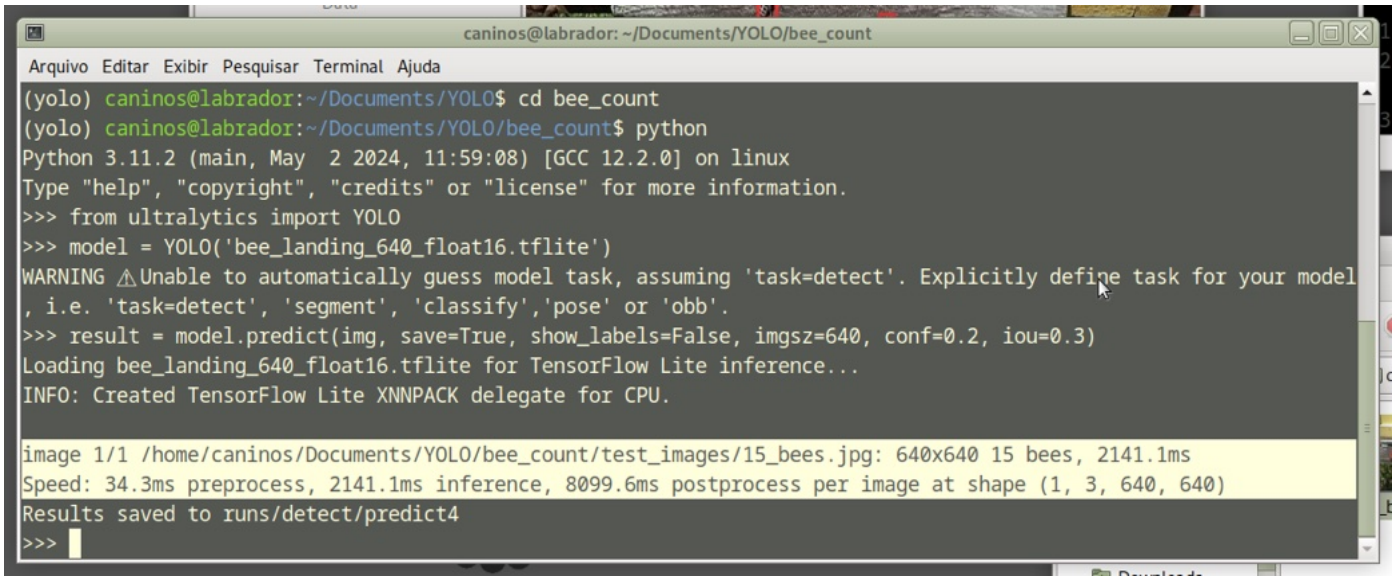
As before, we will import the YOLO library and define our converted model to detect bees:

```
from ultralytics import YOLO
model = YOLO('bee_landing_640_float16.tflite')
```

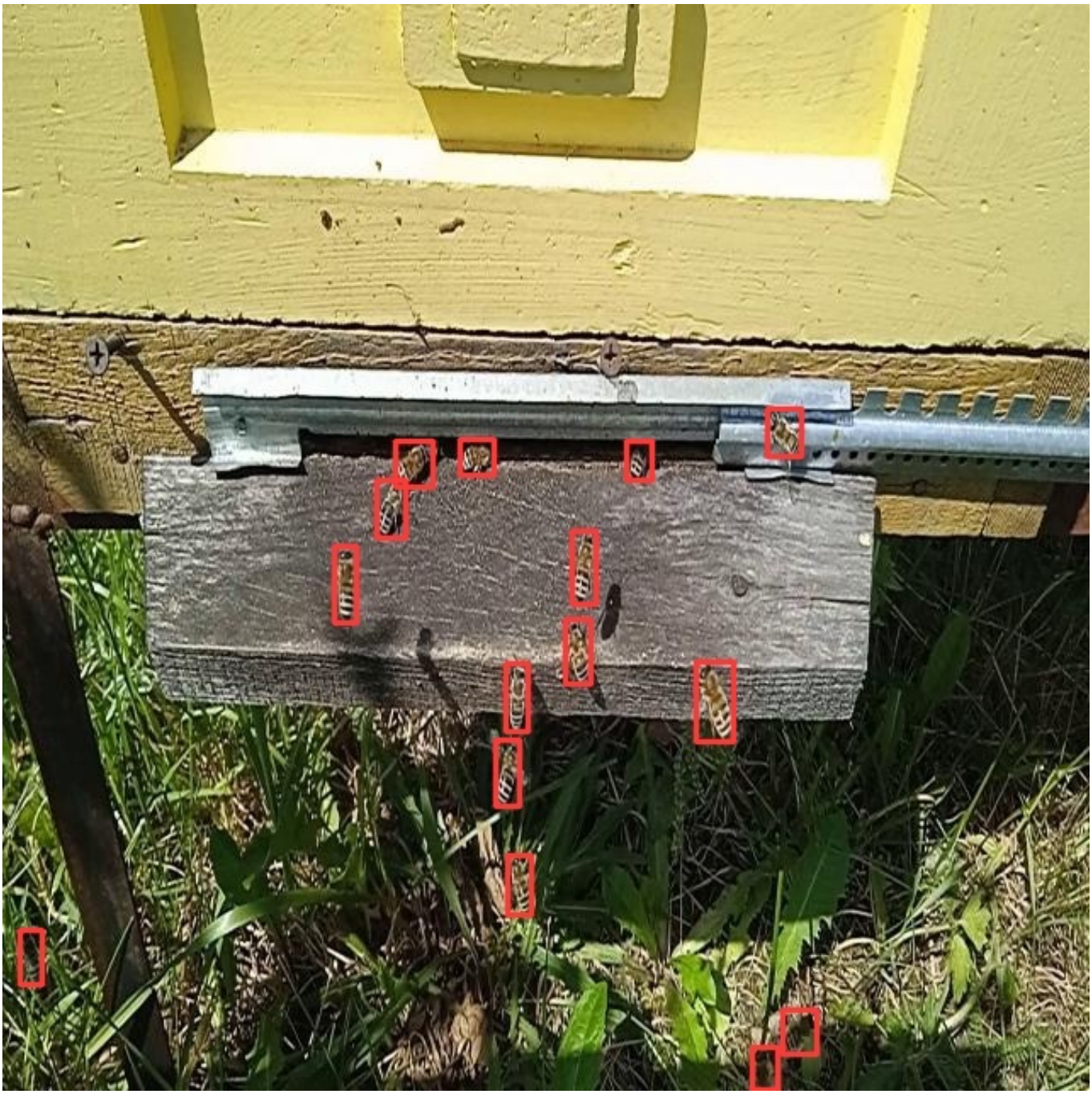
Now, let's define an image (15_bees.jpg), which we know has 15 bees on it, and call the inference (we will save the image w/o labels for external verification):

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

The inference result is saved on the variable `result,` and the processed image on `runs/detect/predict4`



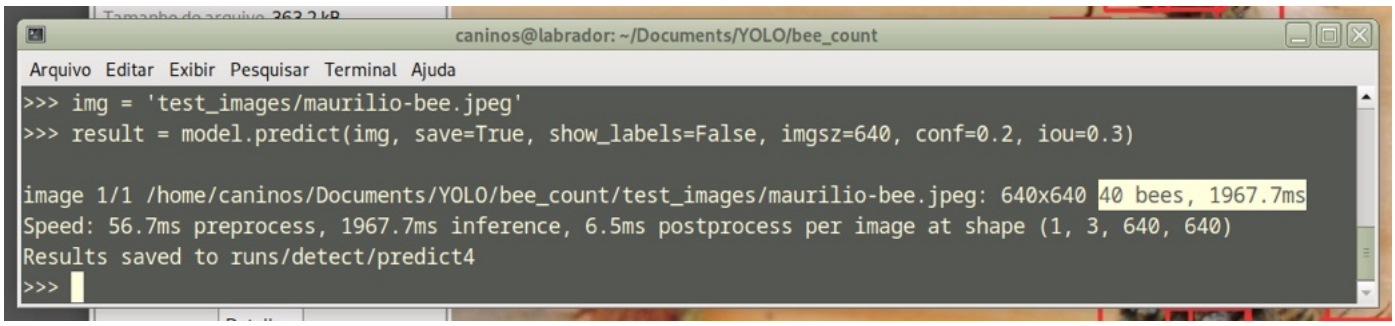
Using FileZilla FTP, we can send the inference result to our Desktop for verification:

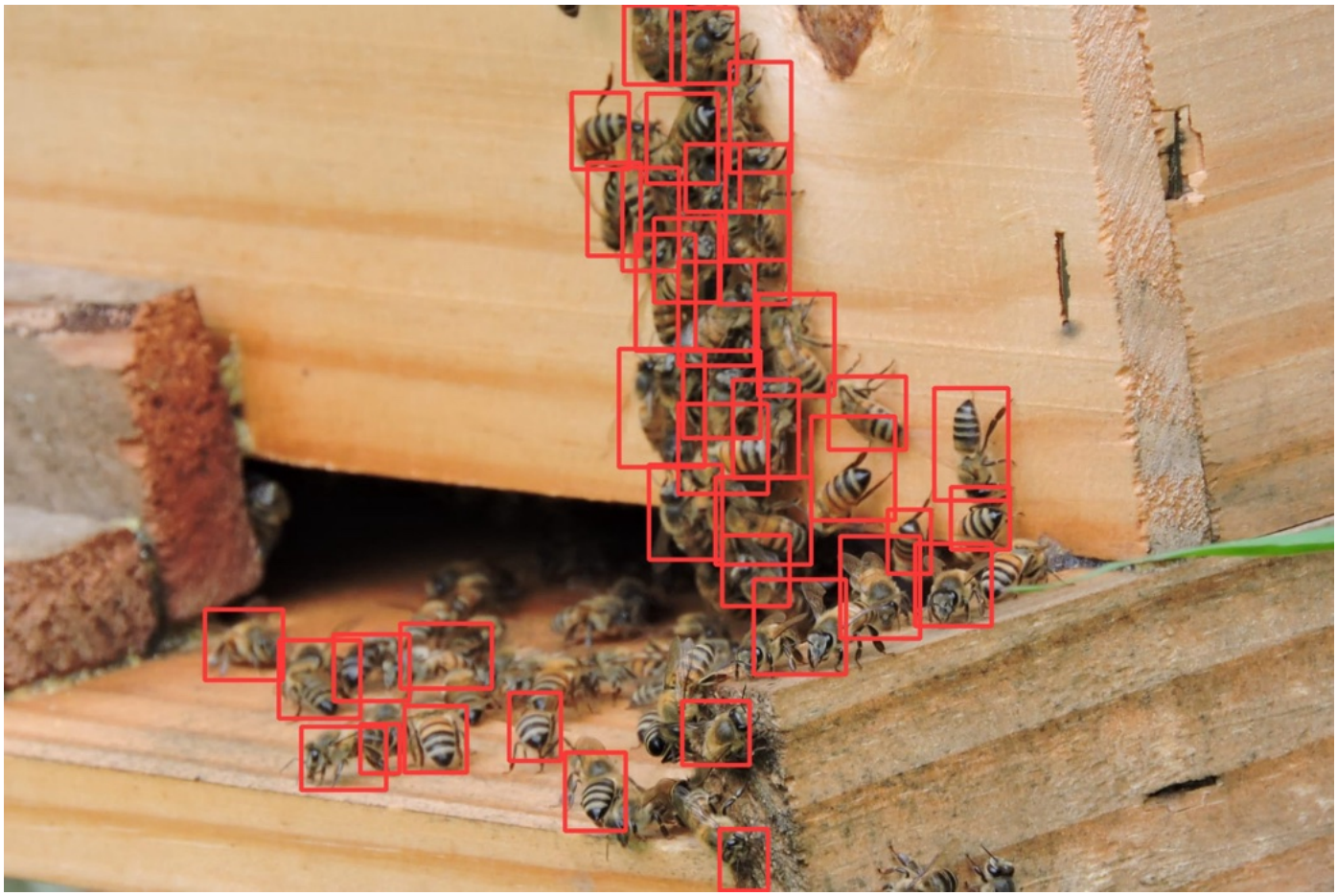


We can go over the other images, analyzing the number of objects (bees) found:

Depending on the confidence, we can have some false positives or negatives. However, with a model trained based on the smaller base model of the YOLOv8 family (YOLOv8n) and converted to TFLite, the result is pretty good running on an Edge device such as the Labrador. Also, note that the inference latency is around 2s.

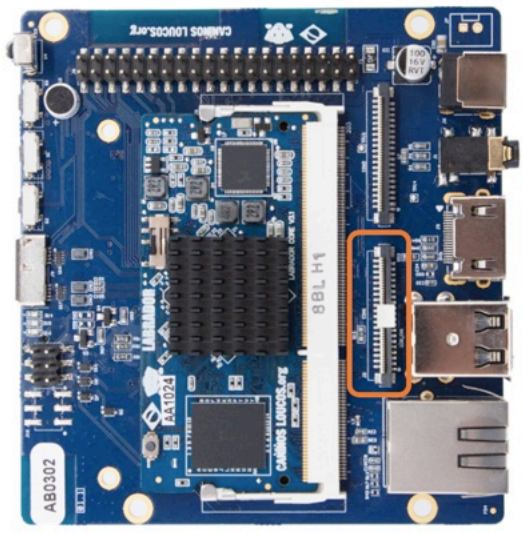
For example, by running the inference on `maurilio-bee.jpeg` (an image from a completely different bee hive), we can find `40 bees`. During the test phase on Colab, 41 bees were found (we only missed one here.)

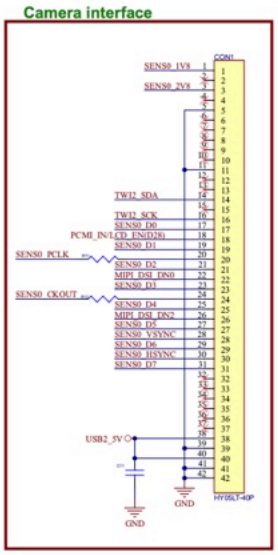




Considerations about the next steps

The next step will be installing and testing a camera in a real scenario. Two possible solutions are using a standard USB WebCam or a camera like the OV5640, which is Banana Pi compatible with the CSI connector.





Camera Banana Pi M1/M1 + OV5640

Another area to be investigated is the latency. With TFlite, the smaller Yolov8 model (nano), the inference takes around 2 seconds. For the Bee-Counting project, this is OK, but other possibilities should be investigated, such as exporting the Yolo model to an NCNN format.

Conclusion

In this project, we have thoroughly explored integrating the YOLOv8 model with the Labrador board 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.

We also provided 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 TFLite format). This not only serves as a functional solution but also as an instructional tool for similar projects.

The Labrador proved a viable edge device for use in the field in this project. One great advantage of the Labrador over the Raspberry Pi is that it has only internal memory, which avoids the famous death by SD card that happens with RPis running for multiple days in a row.

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.

TINYML4D