THOMAS MORE

IT FACTORY – APPLIED COMPUTER SCIENCE

AI Project

Detecting & Tracking Vespa Velutina (Yellow Legged Hornet)

Team: 7

Team Members: Asir Faysal Rafin, Daan Michielsen,
Sharon Maharjan

2023 - 2024

Contents

Introduction

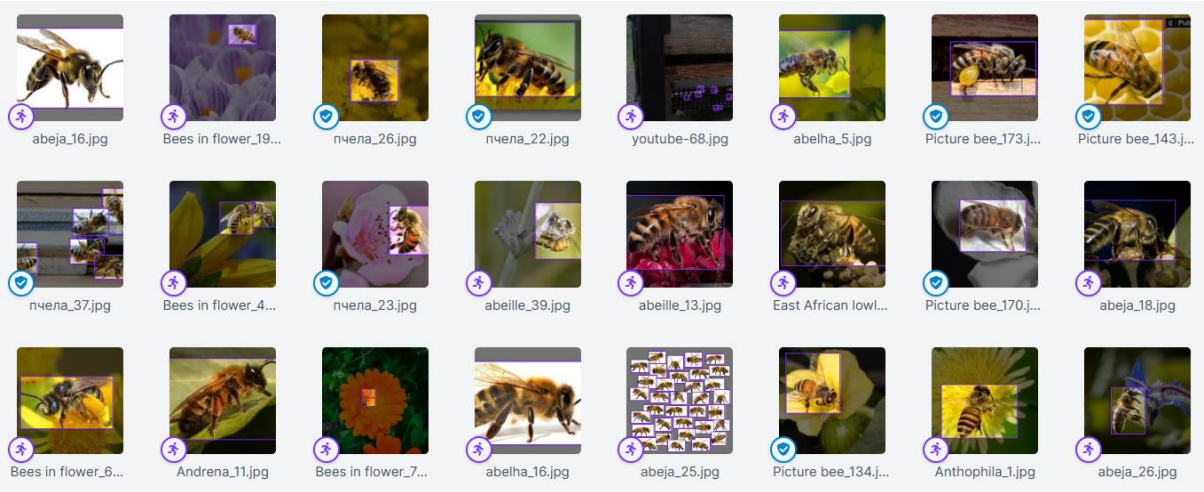
The yellow-legged hornet, also known as the Asian hornet, originates from Asia and has spread to Europe and other regions. It is a predator of honeybees. The detection and tracking of the yellow-legged hornet are important to protect and conserve the honeybees for honey production.

In this project, we aimed to use **YOLOv5** in detecting & tracking yellow-legged hornets in real time. The YOLOv5 model is a deep learning model that performs a single pass over the input image and outputs the class labels and the bounding boxes of the detected objects. When the model was ready, a Streamlit application was created where the user could upload images, videos, or use the webcam to get his inputs detected.

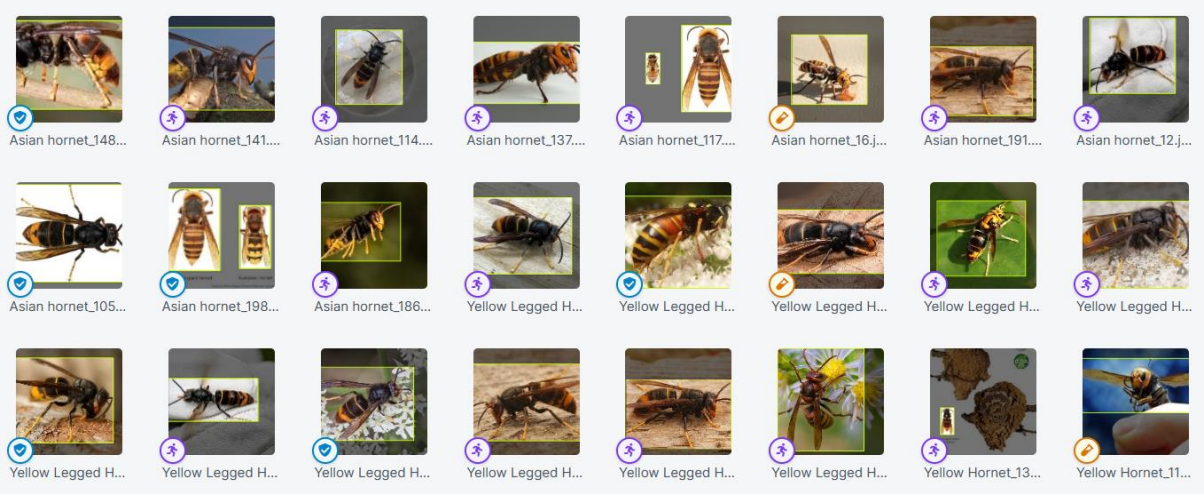
Data Collection

The main type of data to collect the data of hornets and bees is via images and videos. **Selenium** is used to scrape images from Google images. We scrape images from the Bing as well.

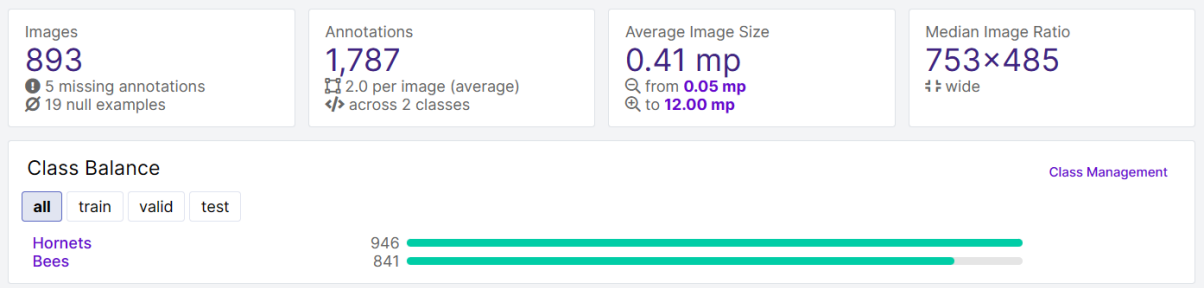
To collect the pictures of bees, we have used these search tags: ‘Abeille’, ‘Abeja’, ‘Abelha’, ‘Andrena’, ‘Andrenidae’, ‘Anthophila’, ‘Apis mellifera’, ‘Bees in flower’, ‘Colletes’, ‘East African lowland honeybee’, ‘Honey bee’, ‘Italian bee’, ‘Lebah’, ‘Picture bee’, ‘Stingless bee’, ‘пчела’, ‘نحلة’, ‘मधुमक्खी’, ‘蜜蜂’. Initially, we started scraping 200 images for each search tag, then we realized that we were getting many duplicates, so we put the limit down to 50. Still, we had many duplicate, blurry and irrelevant images left, so we had to go through with all images of bees, and manually deleted the bad ones. We have tried to collect as many different pictures as possible for bees. Below is the screenshot of some of the pictures that we have collected for bees.



Then, we started scrapping images of hornets using search tags such as, ‘Hornet’, ‘Yellow Legged Hornet’, ‘Hornet Vespa’, ‘Vespa velutina’, ‘Hoornaar’, ‘Hoornaar Vespa’, ‘Aziatische Hoornaar’, ‘Asian Hornet’, and then cleaned up manually. Below is the screenshot of some of the pictures that we have collected for hornets.



We collected more images of bees and hornets from suitable YouTube videos by using the Roboflow inbuilt YouTube video scrapper. We ended up with around 884 images and approximately an equal number of labels for each class: bees and hornets. Below is the screenshot of our dataset with more details.

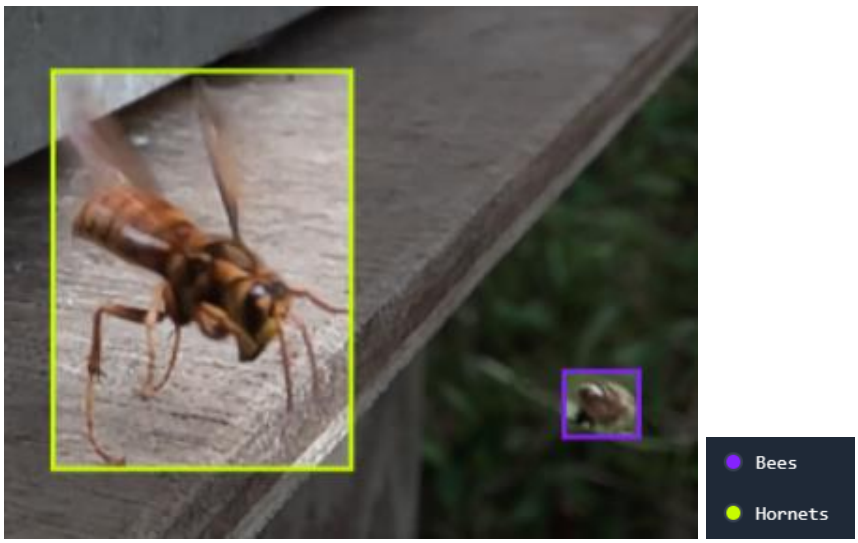


More details of our dataset can be found in Roboflow universe,
https://universe.roboflow.com/bees-and-hornets-20yku/bees-and-hornets/

Labeling

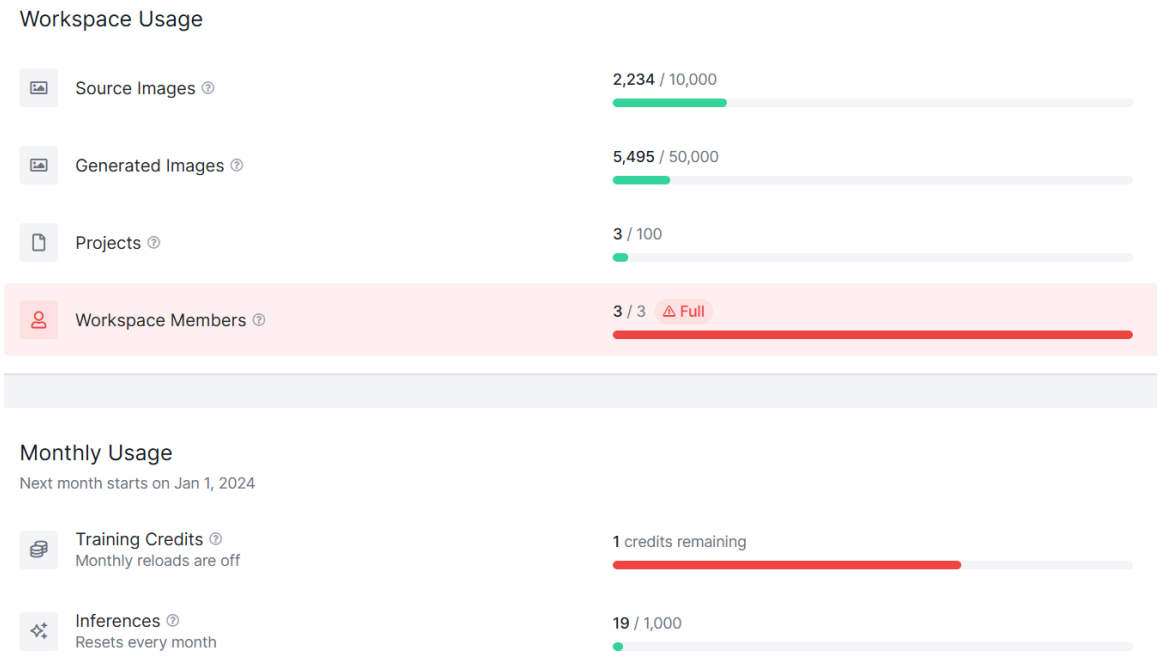
Now, it was time to start labeling. As there are many tools available online to label data, we have tried to figure out which would be best for the use case. We have tried Roboflow, Labelbox, LabelStudio. We stick with Roboflow for its ease of use, fast labeling, and fast collaboration (max 3 users for the free tier). One of the things we do not like about Roboflow is that it does not allow us to delete multiple pictures at once.

When labeling the images, we made sure of high standards. We annotated the object as closely as possible. As labeling is a manual job, we looked for any mistake multiple times, like marking the bee as a hornet. Any false labeling would seriously corrupt the model, causing bad results. Below is an example of our standard in labeling.

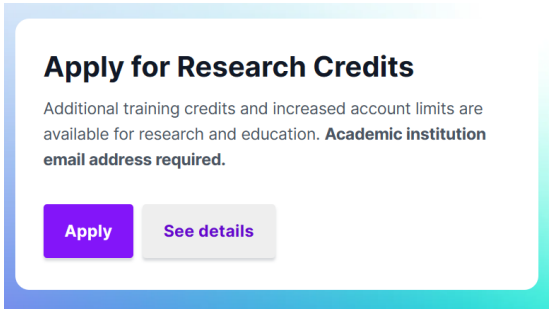


At the beginning of labeling, we prioritized bees, and we annotated all bees even if it is partially seen (like heads and tails). Then, the model was confused and tagged the hornet's head as a bee. Then, we removed all annotations that are labeled on partial body parts of bees and took only bees that were fully seen, which removed confusion for our model.

After collecting the images, we had not hit the limits of the Roboflow free tier. Below are the statistics of Roboflow.



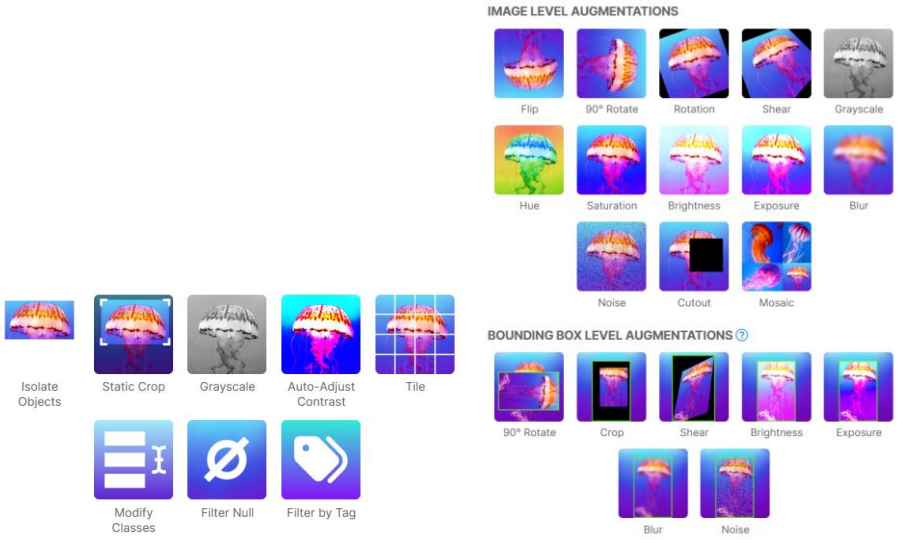
Roboflow also offers additional training limits and increased account limits for research purposes via a Google form on its website. We requested it as well recently but have not gotten a reply yet.



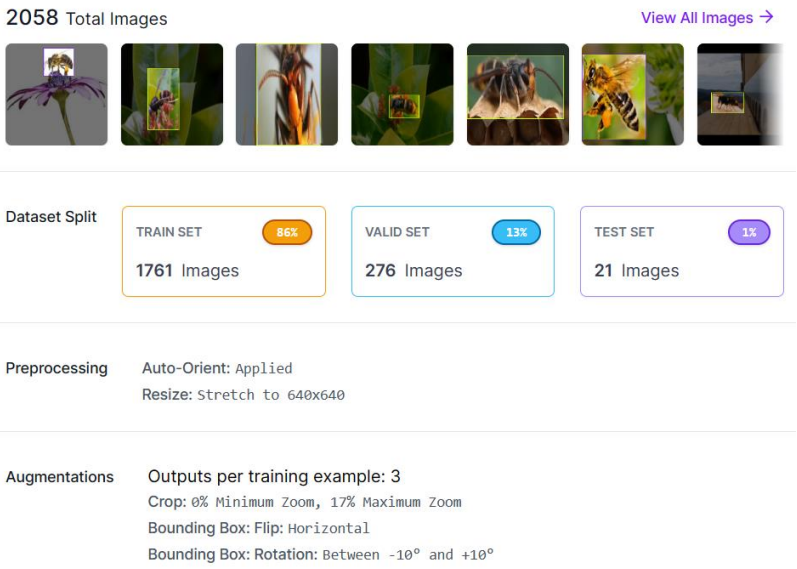
Modeling and Results

Now, it was time for modeling. In the first version of our dataset, we have separated our dataset into three separate sets, Train, Validation, and Test, with a ratio of 70:20:10, respectively. We chose **YOLOv5** over YOLOv6, YOLOv8 models for their accuracy in detecting as many objects as possible.

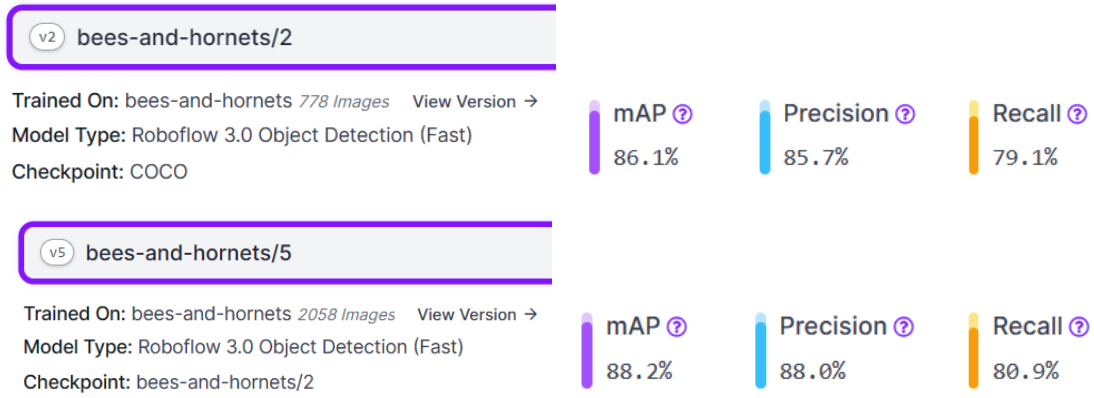
Roboflow offers preprocessing and augmentation for datasets.

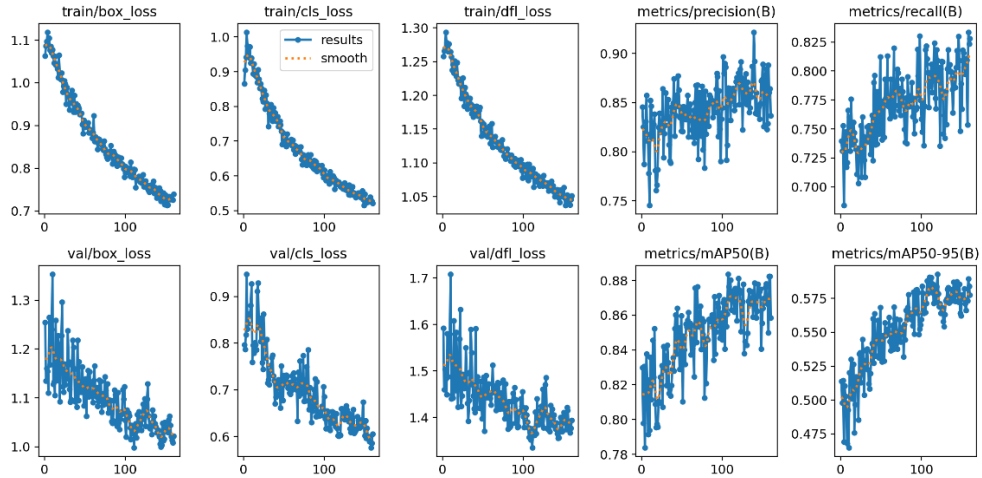


We have used Auto-Orient and Resize (stretch to 640*640) for preprocessing. We have used crop, bounding box flip, and bounding box rotation for augmentation which helped us to expand our dataset three times to 2058 images. We wanted to use most of our images for training and validation. That's why the test set was significantly reduced. For the final dataset, we had Train, Validation, and Test set, with a ratio of 86:13:1, respectively. Below is the summary of the final version of our dataset.



Roboflow offers Roboflow 3.0 Object Detection (Fast) for object detection. We had used 2 training credits out of a total of 3 credits (free). In the first training (v2), Roboflow took 300 epochs to train (and around 3 hours). Also, we did not use any augmentation for images. Also, our detection for some pictures was not good. Later, we added more pictures and cleaned up our dataset. For v5, the training started from the checkpoint of the previous training, and it took 160 epochs to train, then we had better results (mAP 88.2%).





We trained our custom YOLOv5 model in Google Colab. First, we installed the requirements, assembled our dataset, and then trained our custom YOLOv5 model. We went for 300 epochs (as we seen Roboflow going over 450 epochs in total), and it took around 2.5 hours. Code snippets are added below.

```
#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output  # to display images

print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'})")
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="TAQXNlttGMH92B2rCZpo")
project = rf.workspace("bees-and-hornets-20yku").project("bees-and-hornets")
dataset = project.version(5).download("yolov5")
```

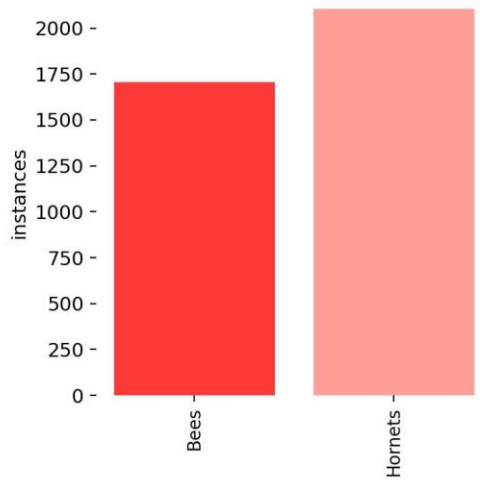
```
# set up environment
os.environ["/content/yolov5/Bees-and-Hornets-5"] = "/content/datasets"
```

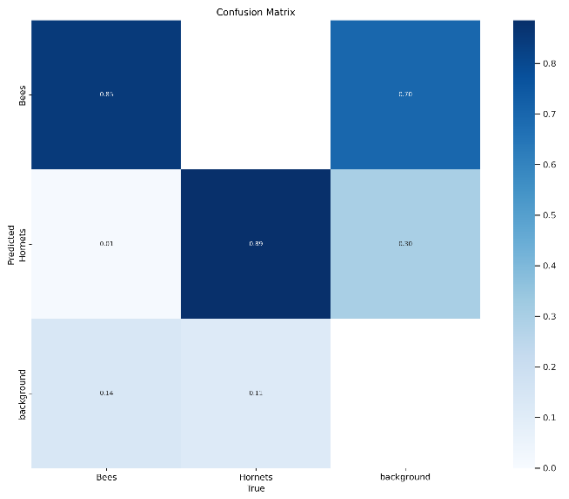
```
!python train.py --img 640 --batch 16 --epochs 300 --data {dataset.location}/data.yaml --weights yolov5s.pt --cache
```

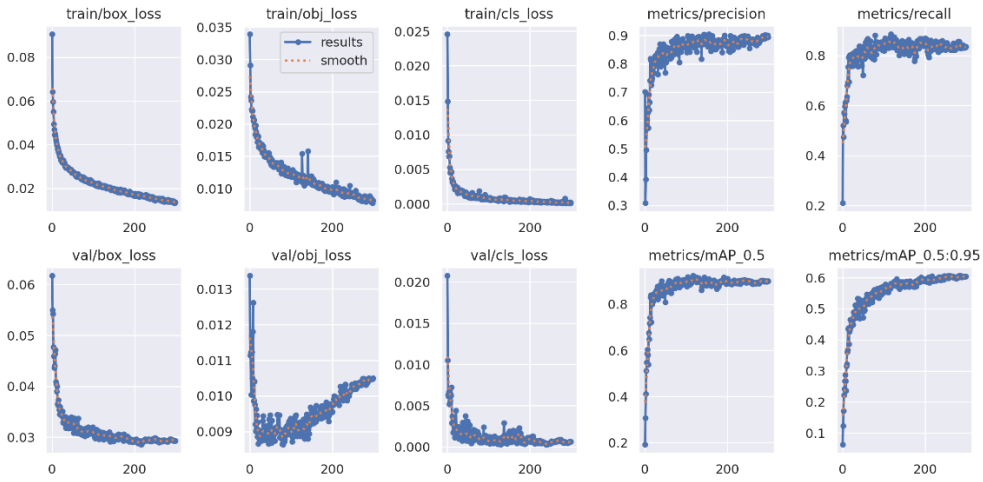
```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
    299/299      4.29G    0.01365   0.008115  0.0001461          3        640: 100% 111/111 [00:29<00:00,  3.74it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 9/9 [00:03<00:00,  2.75it/s]
                   all        276        485      0.895      0.836      0.901      0.605

300 epochs completed in 2.738 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp/weights/best.pt, 14.4MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 9/9 [00:05<00:00,  1.72it/s]
                   all        276        485      0.889      0.857      0.903      0.606
                  Bees        276        267      0.864      0.835      0.889      0.569
               Hornets        276        218      0.914      0.879      0.918      0.644
Results saved to runs/train/exp
```

P	R	mAP50
0.889	0.857	0.903

After training with YOLOv5, we have better results than Roboflow 3.0 Object Detection (Fast). That is great. Below are some metrics of our custom YOLOv5.

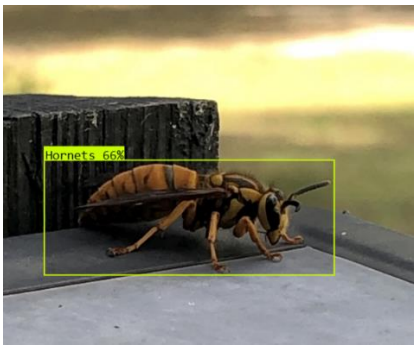


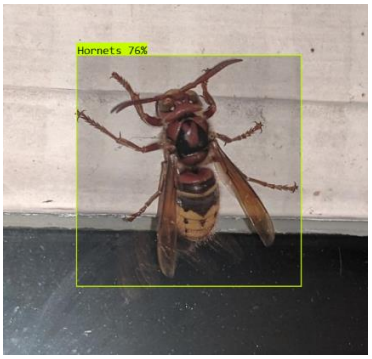




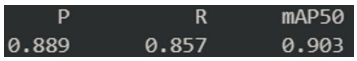
Now, it's time to compare results for two different models. To approach this, we have collected images that our models have never seen before.

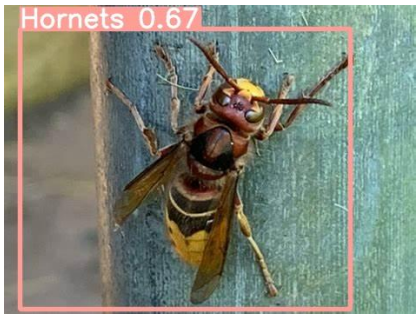
Roboflow 3.0 Object Detection (Fast)

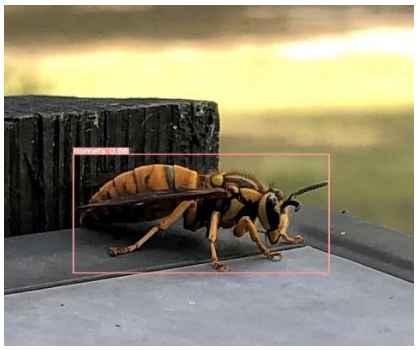


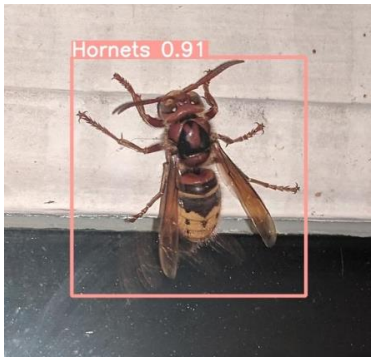


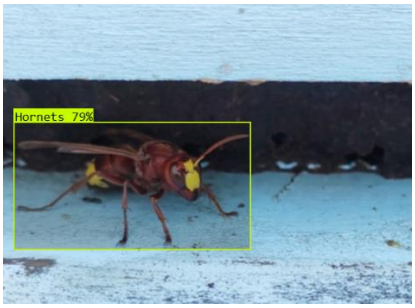
YOLOv5

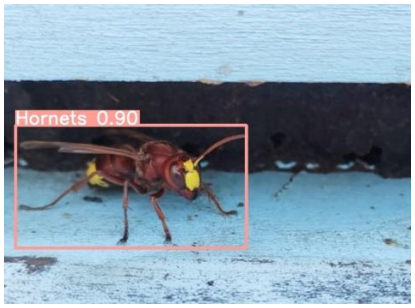


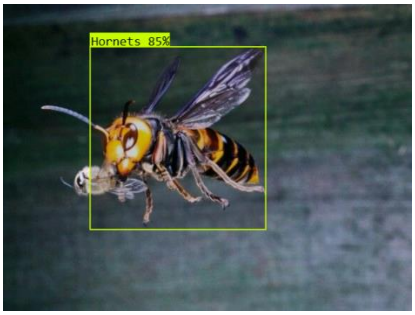












We can clearly see YOLOv5 has better confidence in most of the pictures. We have investigated more details of Roboflow 3.0 Object Detection (Fast) model, but we could not find the specific architecture or method used by the model.

Tracking

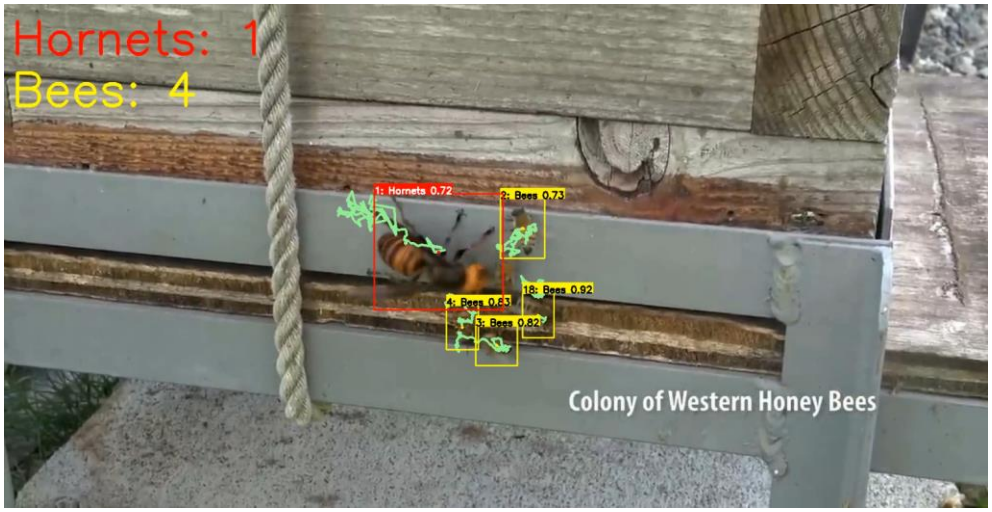
We did the object tracking in Google Colab. First, we cloned the object tracking repository, installed the requirements, and then set up YOLOv5 model for object tracking with custom weights and video input. Code snippets are added below.

```
!git clone https://github.com/RizwanMunawar/yolov5-object-tracking.git
```

```
%pip install -qr "/content/yolov5/yolov5-object-tracking/requirements.txt"
```

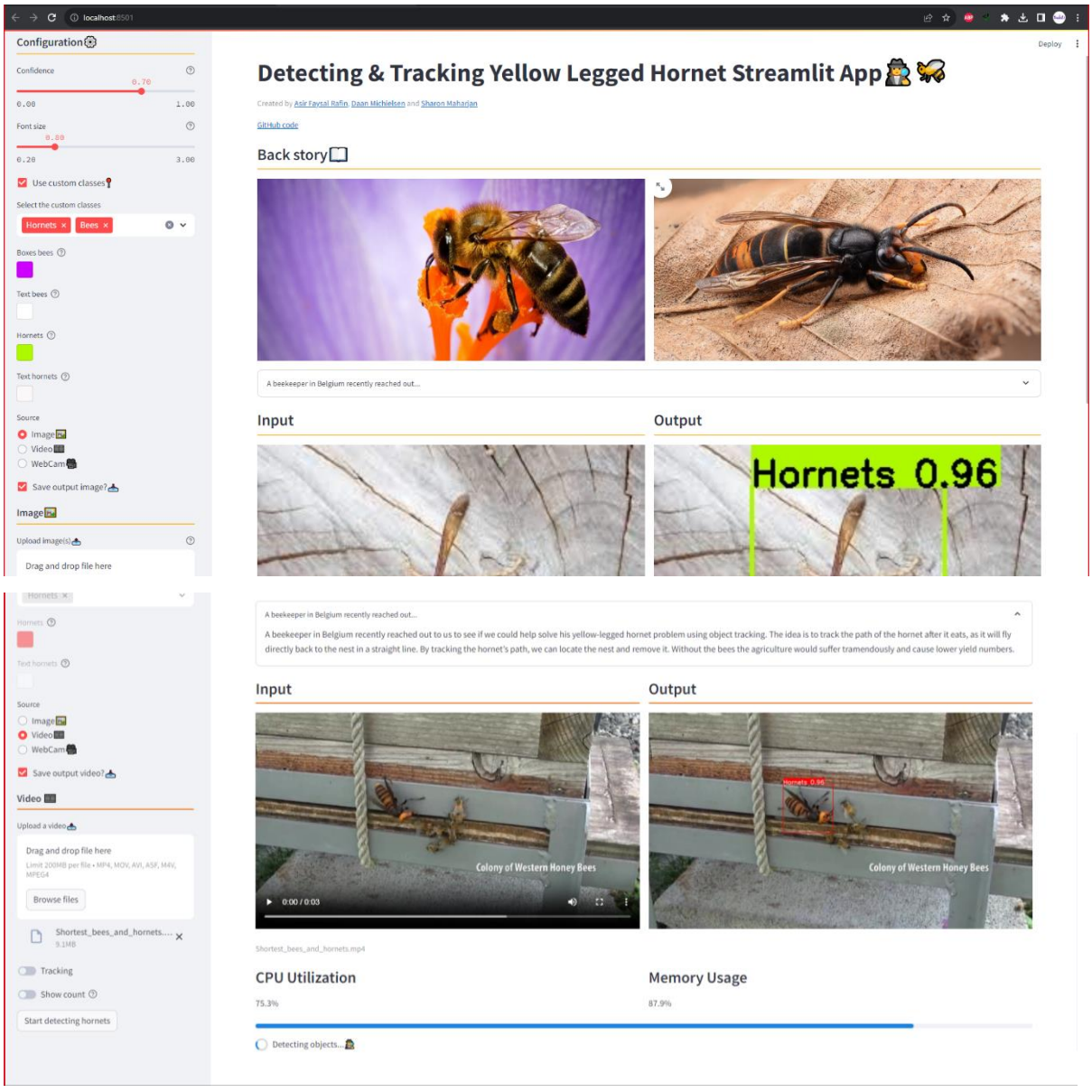
```
#for object detection + object tracking
!python /content/yolov5/yolov5-object-tracking/obj_det_and_trk.py --weights /content/best.pt --source "/content/drive/MyDrive/Test.mp4" --conf-thres 0.5
```

We succeeded in getting the output with tracking and inference.



Streamlit

We have deployed our YOLOv5 custom object detection model in a Streamlit application where users will upload an image, a video, or use webcam to detect objects. In the app, he can select and customize the classes, configure the confidence level, choose the color for bounding boxes, and generate inferences for his input. He can see results live in the browser, as well as save the output. Below is the screenshot of the app.



Problems

- Selenium: When we scrapped images from Google images, we had many duplicate images from Google. We had to put the limit down to 50 images and downloaded images with different search tags.

- We tried preprocessing images in black and white. It led to bad results; we avoided preprocessing like that later in the project.
- We had issues with our model predicting hornet's heads as bees. At the beginning of the project, we had one class to detect, which is bees. Naturally, we annotated all the bees in a photo even if their bodies were partially visible. We removed them, which cleared the confusion for the model.
- We had pushed too many images of hornets from one angle in our dataset that we took from one YouTube video. That made the model biased for hornets, as it had seen one type of image too many times. Later, we balanced it out by deleting many of those images from our database. We also scraped more images and added them to the dataset.
- We had difficulties in showing inference of class title, confidence level, and changing the color of the bounding boxes for different classes.
- We have not been able to make the tracking line go away after time. The line keeps on following the object and never goes away, except for when the object leaves the frame.
- When we deployed the model in Streamlit, we had the video output not showing. Later, we found the issue and a solution for the output not displaying. We had to use a package called moviepy instead of openCV for the creating of the result video. OpenCV does not support some codes that are needed for a browser to support the video.
- In the Streamlit app, the result video played fine, but after saving the file it was blank. But we have found the solution and now it works with ease.
- One issue that we could not fix is hosting the app in Streamlit cloud because of too many dependencies.

Conclusion

In this project, our objectives were to design and train a custom YOLOv5 model that can accurately identify and track the yellow-legged hornets in various scenarios. We have successfully developed a real-time object tracking system based on YOLOv5 deployed in Streamlit to detect and monitor the yellow-legged hornet, an invasive species that poses a threat to honeybees.

We learned by doing, how to deal with a computer vision project, realizing the importance of a good dataset, quality of labeling, and sensitivity of object detection model. We are proud of what we have achieved so far. Below, on the left, we have a picture of a bee detection with poor

results from week 1, and on the right, we have impressive results in detecting bees and hornets at the end of the project. We have learned a great deal about computer vision and are thankful to participate in the project.