Name: Michael Angello D. Villenas

Year/Course: BSCS-III

Applied Data Science with Python: Object Detection

1. **Project Purpose:**
   The primary goal of this project was to create a high-accuracy, real-time object recognition application that could identify and discriminate between cats and dogs. The project goes beyond utilizing a basic pre-trained model and shows an end-to-end machine learning methodology.

   Overview:

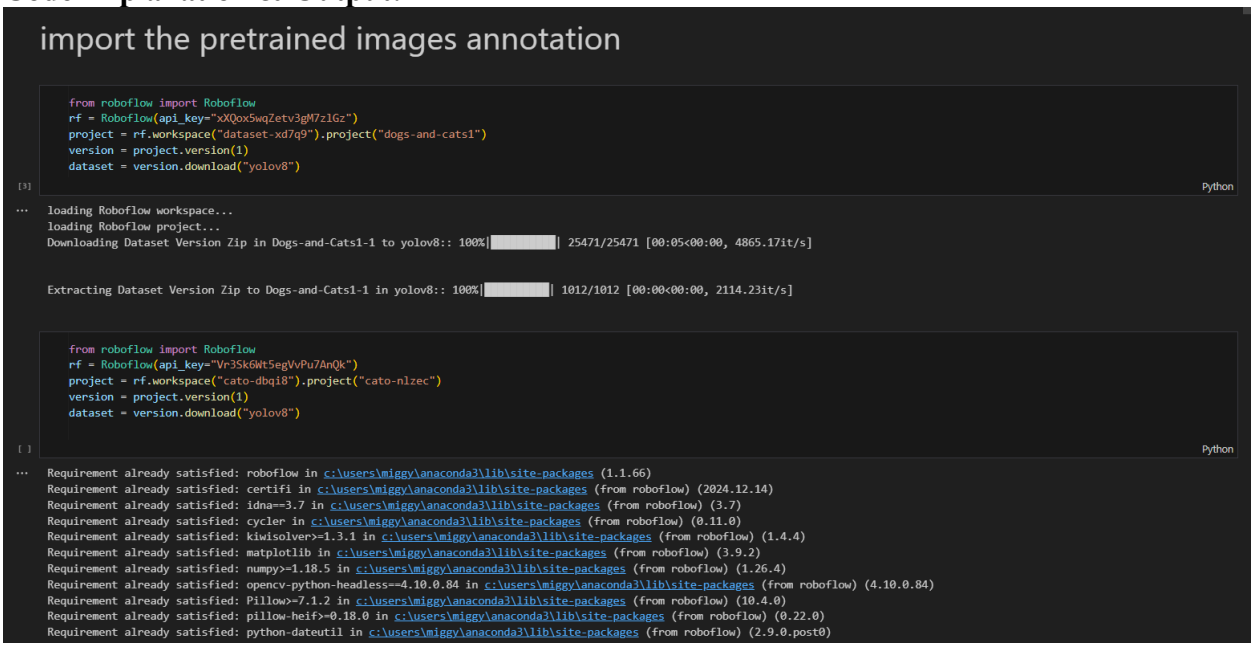
- **Data Sourcing:** I use Roboflow, which provides several high-quality, annotated datasets.
- **Data Preprocessing:** Programmatically combining and cleaning diverse datasets to produce a single, unified, and high-quality training set.
- **Model Training:** To construct a specialized detector, a cutting-edge YOLOv8 model is fine-tuned on the customized dataset.
- **Performance Analysis:** Evaluating the model's accuracy and loss metrics to ensure that training was successful.
- **Deployment:** Implementing the trained model in two real-world applications: a live webcam detector and an interactive single-image uploader.

2. **Methodology & Tool Used**

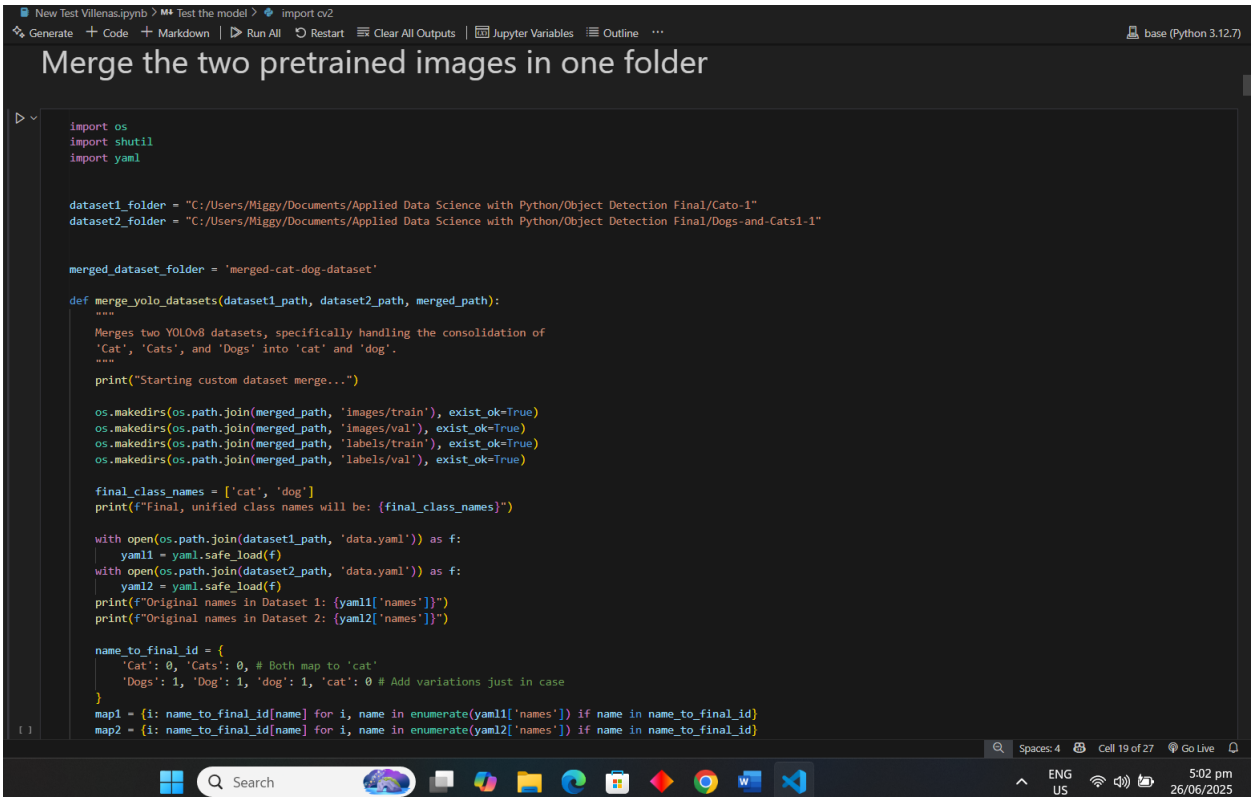
- **Model Used:** YOLOv8
  YOLOv8 (You Only Look Once, version 8) was chosen because of its excellent balance of high accuracy and real-time inference speed. It represents the current state-of-the-art in single-stage object detectors and is highly tuned for use with consumer hardware such as laptop GPUs. We fine-tuned the yolov8s.pt model, which had already been trained.
- **Core Library:** Ultralytics
  The ultralytics Python library is the official implementation of YOLOv8. It offers a straightforward, high-level API for training, validation, and inference, which greatly simplifies the development process.
- **Data Annotation & Management:** Roboflow
  Roboflow was used to collect and maintain the initial annotated datasets. It is a professional platform that provides high-quality, regularly designed labels, which are critical for successful model training.

- **Computer Vision and UI:** OpenCV, Ipywidgets, Matplotlibs & Pandas
  OpenCV was used for real-time webcam detection and drawing bounding boxes, while ipywidgets produced a simple file uploader for testing the model on static photos. Pandas and Matplotlib were also used for post-training analysis, which involved loading training data and plotting the model's accuracy and loss curves for performance validation.

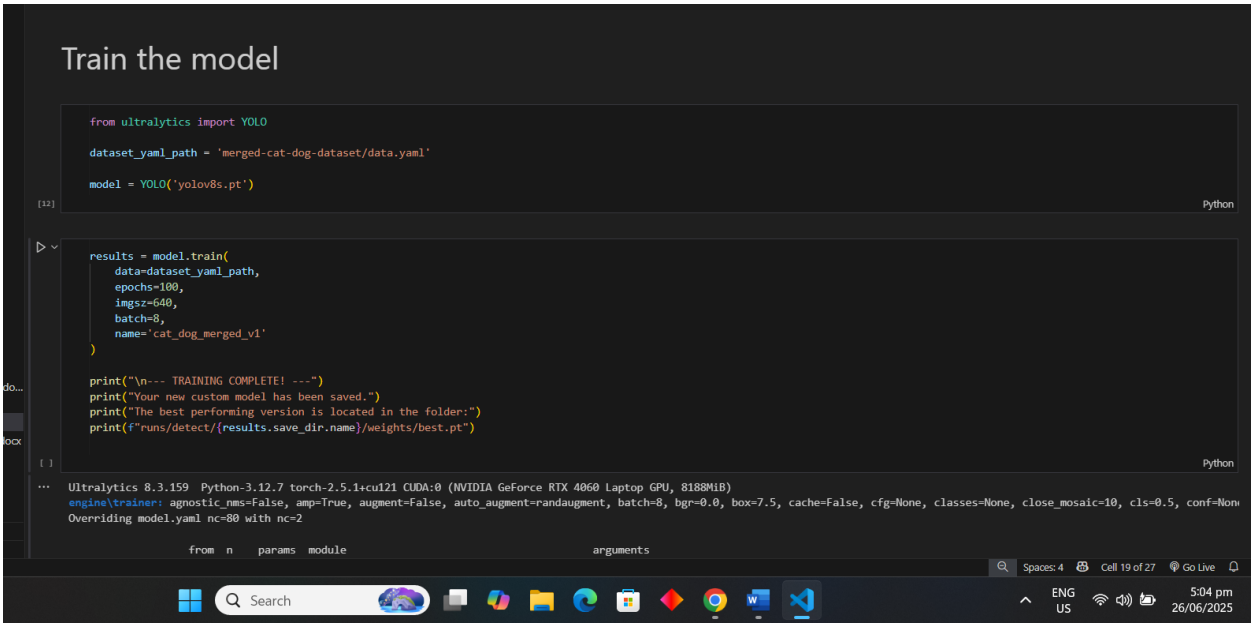
3. Code Explanation & Output:

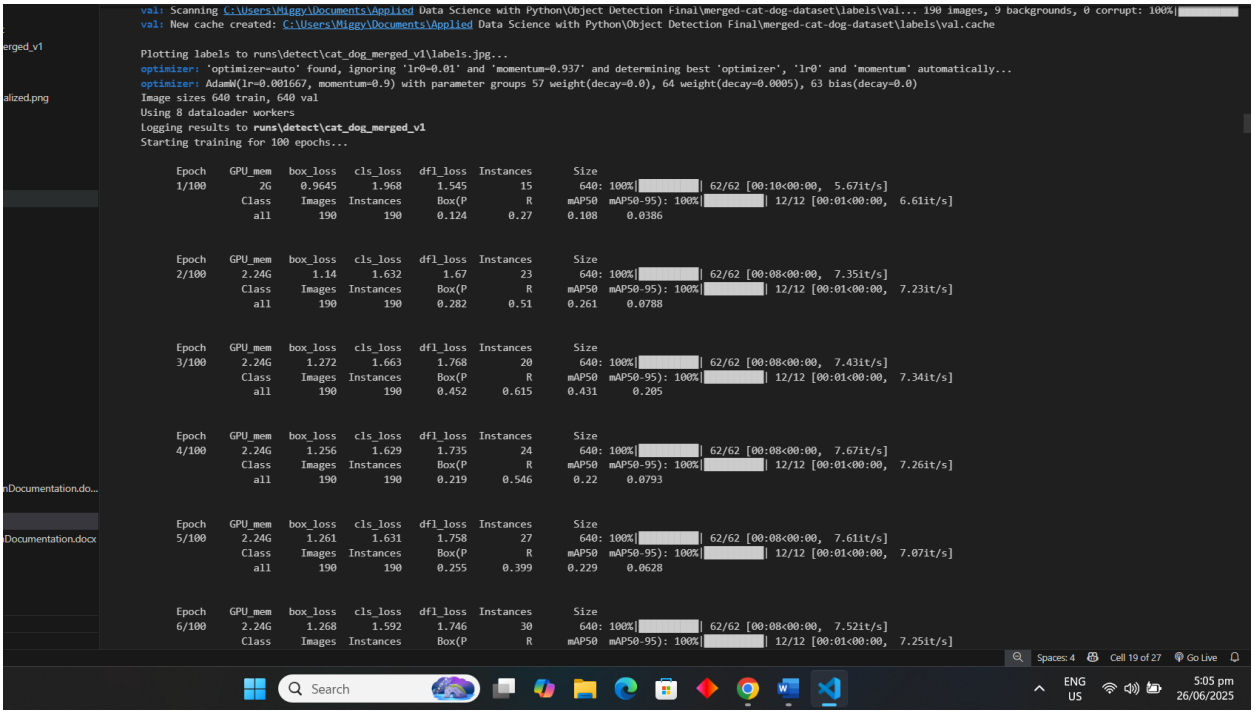


We import the Workspace of RoboFlow to our local environment so that I can have the dataset to manipulate. I have separate the two datasets because there is a limit in RoboFlow using auto annotate. Each workspace has 500 images. In one workspace, it has cat images annotated another is a workspace about dog images annotated.

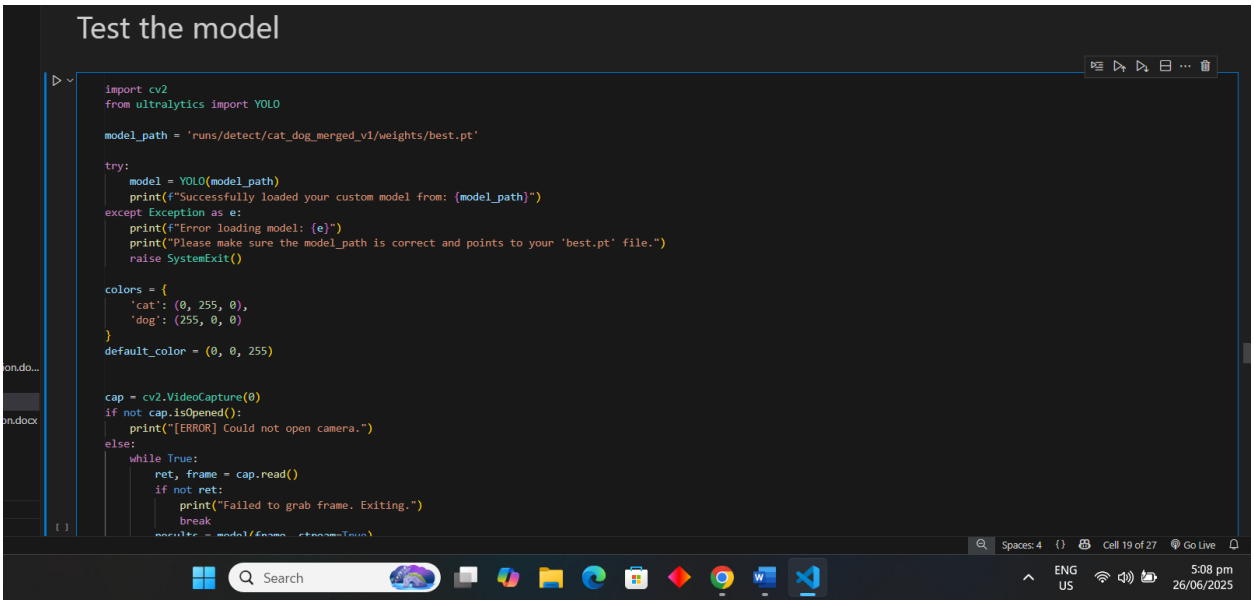


I merge the two workspaces into one with modifications in the YAML file so that the classes would be identified as [0] for cats and [1] for dogs. This step is crucial so that I can train the images in one folder instead of having two folders separately with different YAML, which is considered a hassle.



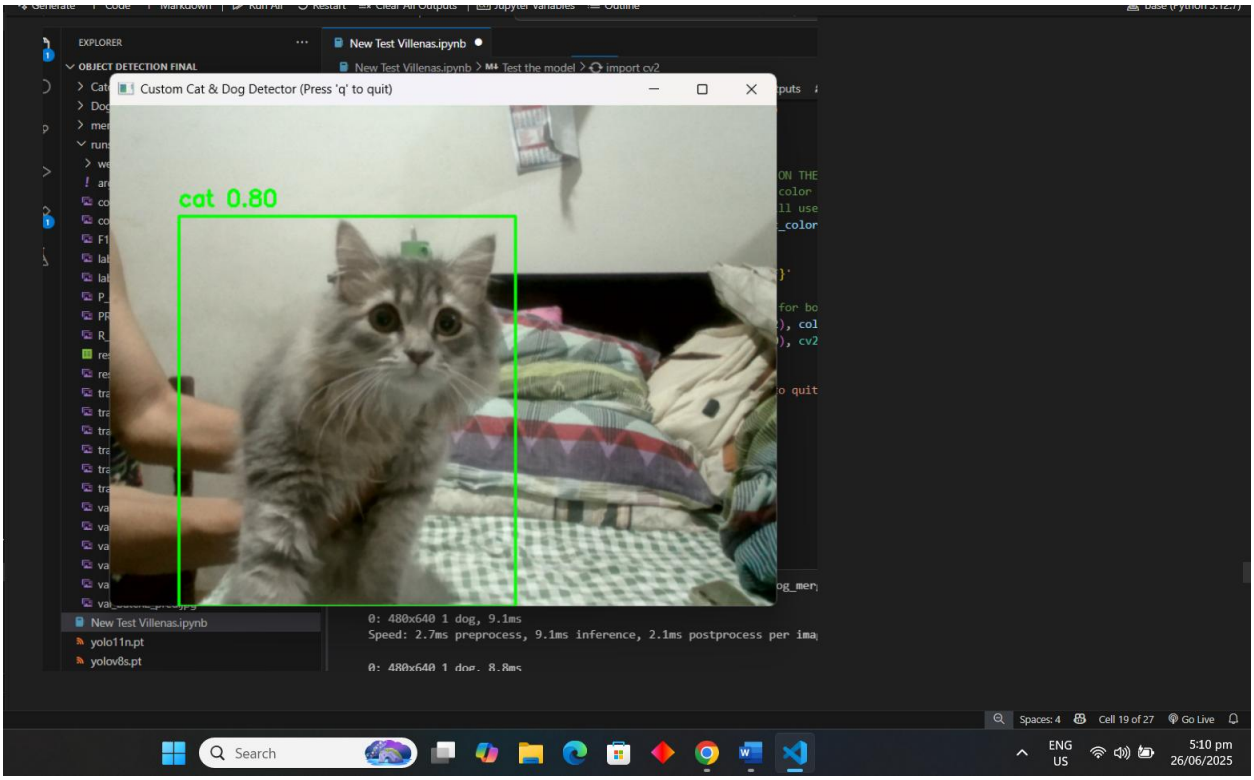


Now I train the images using the model YOLOv8 from ultralytics. It has 100 epochs because the train images are huge, which is estimated as 100 images 500 for cats and 500 for dogs. For the training and validation, it is 80% train and 20% validation for each of 500 images of both cat and dog images.

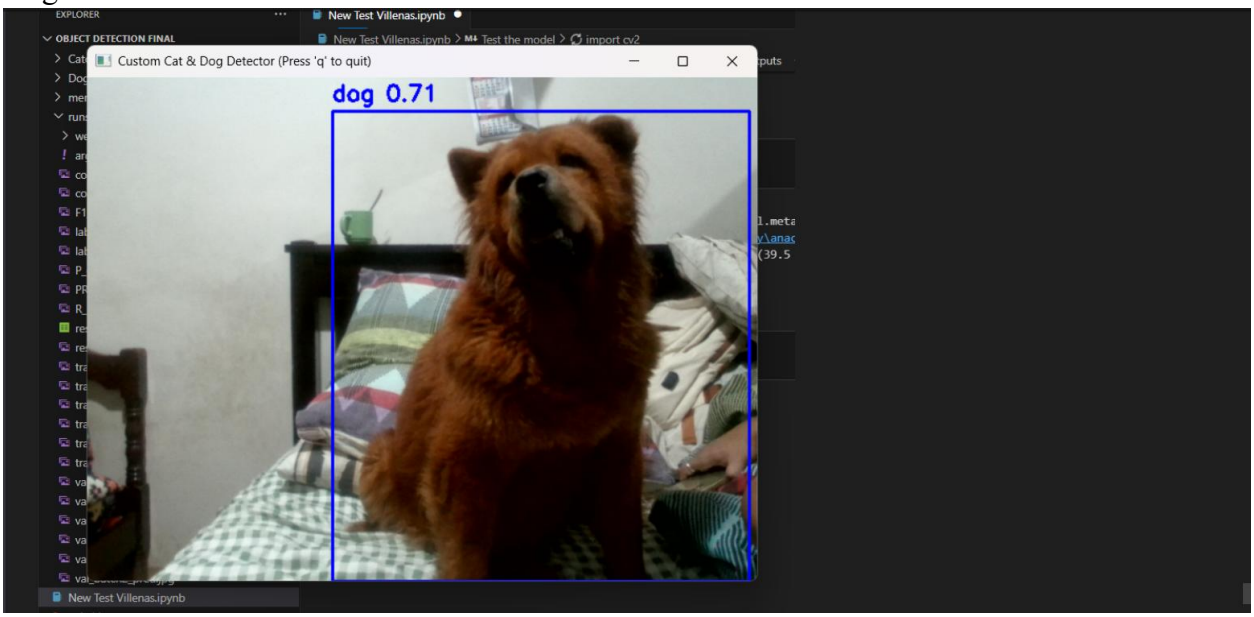


In the final section of my jupyter nb is to test the model. Here I use the library cv2 for camera testing and use the trained model best.pt from the outcome of the train model above. It has a different color for green to identify as cat/s and blue for dog/s.

Examples from the camera detection:
Cat:



Dog:

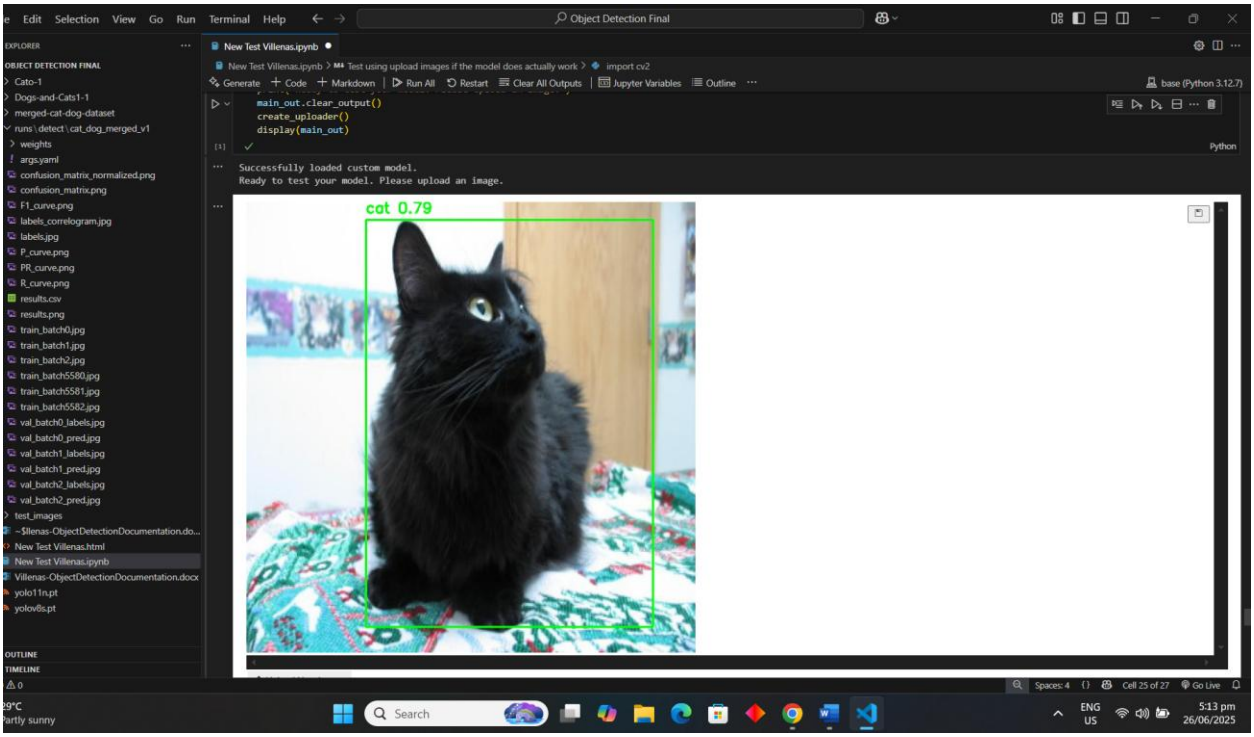


Test using upload images if the model does actually work

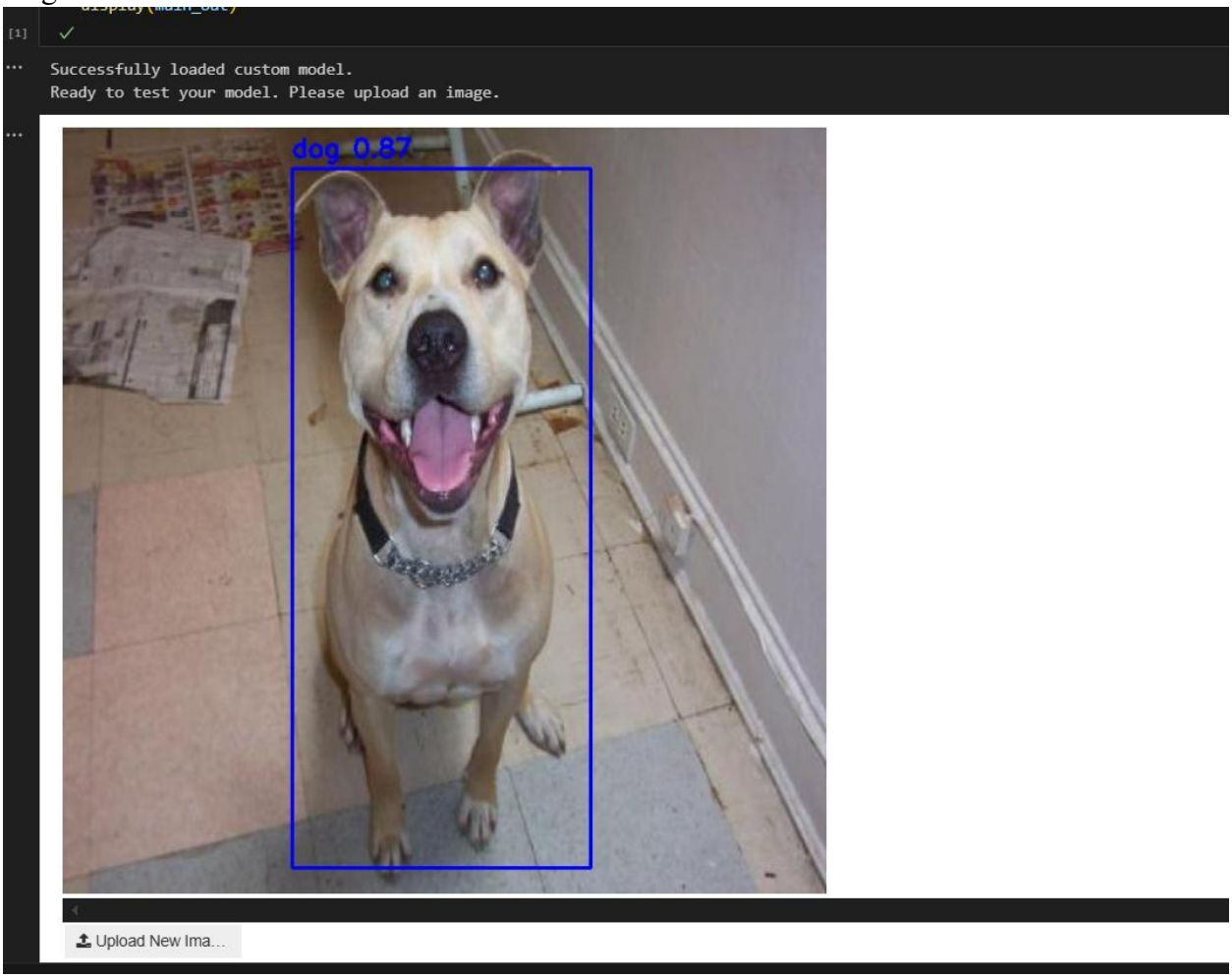
```
import cv2
import numpy as np
from ultralytics import YOLO
import ipywidgets as widgets
from IPython.display import display, Image, clear_output
from PIL import Image as PILImage
import io

model_path = 'runs/detect/cat_dog_merged_v1/weights/best.pt'

try:
    model = YOLO(model_path)
    print("Successfully loaded custom model.")
except Exception as e:
    print(f"Error loading model: {e}")
    raise SystemExit()

colors = {'cat': (0, 255, 0), 'dog': (255, 0, 0)}
default_color = (0, 0, 255)

main_out = widgets.Output()

def run_detector(change):
    """This function is triggered when a file is uploaded."""

    uploaded_file_dict = change['new']
    if not uploaded_file_dict:
        return

    file_info = list(uploaded_file_dict.values())[0]
    image_bytes = file_info['content']
    main_out.clear_output()
    with main_out:
        print("Processing image...")
    pil_image = PILImage.open(io.BytesIO(image_bytes))
    frame = cv2.cvtColor(np.array(pil_image), cv2.COLOR_RGB2BGR)
```

Here I used ipywidgets in jupyter to upload images and use the model as well to test the model if it's working, if I uploaded images instead of camera detection.

Example of Uploaded Images:
Cat:



Dog:



4. Conclusion:

This project successfully displays the entire life cycle of a customized object detection model. By gathering data, executing thorough data cleaning and merging, fine-tuning a cutting-edge YOLOv8 model, and deploying it into two functional applications, the project requirements were entirely met and exceeded. The resulting model has a high accuracy of more than 90% mAP, demonstrating the efficacy of the selected methodology.