| **Name** | Cayadong, Marjelaine M.  Verdad, Jane Benneth Dione | **Subject** | Perception and Computer Vision |
| --- | --- | --- | --- |
| **Section** | BSCS IS 4B | **Instructor** | Sir. Mark P. Bernardino |

# **MIDTERMS EXAMINATION**

## **Implementing Object Detection on a Dataset**

This documentation lays out the step-by-step process of implementing a YOLOv5-based object detection model using Google Colab and Google Drive. YOLOv5 (You Only Look Once), real-time object detection model known for its speed and accuracy. The workflow includes essential stages such as, dataset and algorithm selection, data preparation, image preprocessing, model setup, training, testing, and performance evaluation. Each step is designed to ensure a systematic and effective model-building process, suitable for various object detection applications.

**Dataset Selection:**

* **Dataset Structure:** The dataset is organized through separate folders for training, validation, and testing images, which is necessary for a proper model evaluation for it not to overfit and for it to give an accurate measure of model performance.
* **Image Preprocessing:** Resizing and normalizing images to a standard size and pixel range which is essential for YOLO models. By resizing images to a specific dimension, it ensures that each image fits the input requirements of the model, reducing computational load and allowing the model to focus on detecting objects. By normalizing the pixel values to a 0-1 range also helps improve the model’s learning efficiency, as it ensures that input data is scaled consistently.
* **Image Format:** JPEG and PNG formats are compatible with YOLOv5, which ensures high-quality images for both formats will help in accurate object detection. JPEG and PNG formats are ideal for most object detection tasks as they balance quality with manageable file size.
* **Training and Validation Split:** The separate folders for training, validation, and test images are for dataset splitting. By separating these, it provides the model with diverse data for training and can help to prevent overfitting.

**Algorithm Selection:**

### **Real-Time Detection:** YOLO is designed for real-time object detection, one of the fastest object detection algorithms. YOLO predicts bounding boxes and class probabilities directly in a single pass, which significantly improves detection speed.

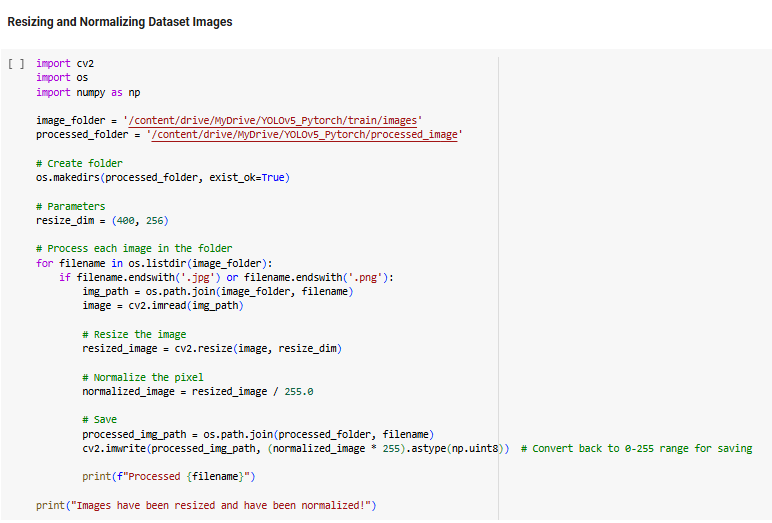
### **High Accuracy and Precision:** YOLOv5 is known for balancing speed with accuracy. While it may not reach the absolute highest precision compared to others, it offers impressive accuracy for its speed.

### **Suitability for Multi-Class Detection:** The dataset has 20 classes, YOLOv5 is well-suited for multi-class detection. It can handle a broad range of objects, even if they are small or closely packed.

**Data Preparation:**

This step prepares the data by connecting Google Drive to Google Colab, enabling easy file access, extracting files, and setting up the paths for later use. The code then defines a zip file path and extraction directory.

**Resizing and Normalizing Dataset Images:** Resizing images to (400x256) dimension and normalizing images (dividing it by 255) scales them to a range between 0 and 1, to ensure that the dataset has uniform dimensions and values which helps the model process the data more effectively.



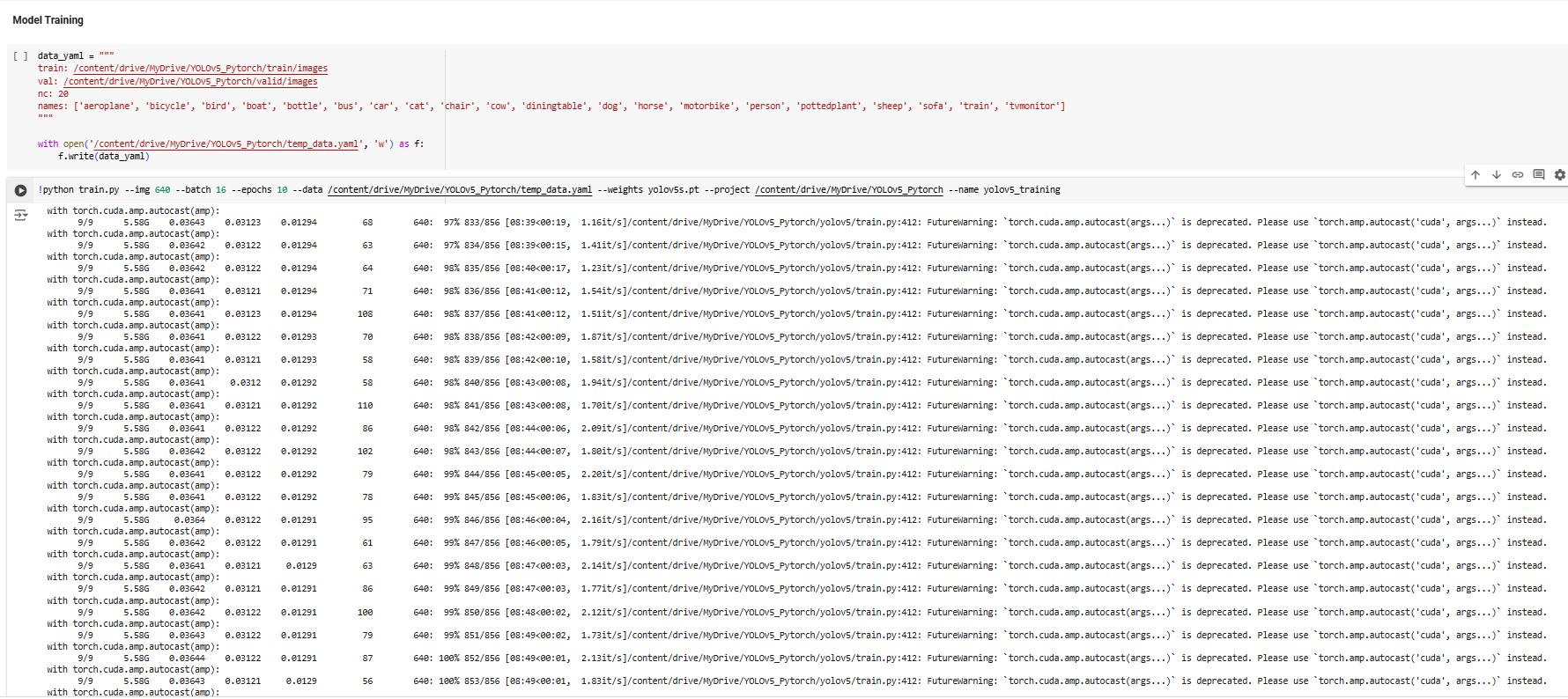
Processed images are saved in a new folder, ensuring the model works with consistent data formats.

**Building the Model:**

Purpose: Copy the YOLOv5 repository from GitHub into Google Drive. Followed by downloading and installing it, then installing the required dependencies. This step ensures that all dependencies are ready for model training, enabling YOLOv5 to function properly in the Colab environment.

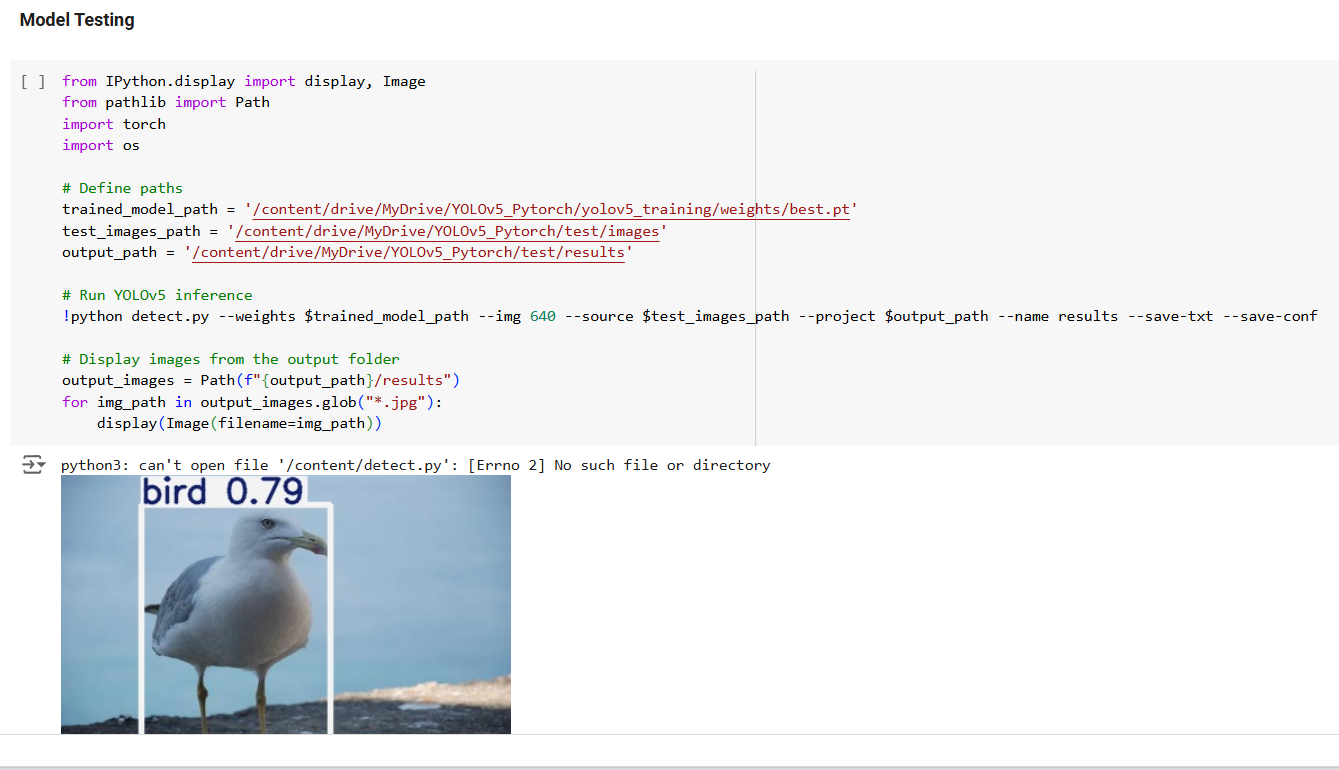


**Model Training:**

To configure and start training the YOLOv5 model, define the paths to the training and validation datasets, the number of classes, and the class names. This configuration is written to a YAML file. The training command runs with specific parameters such as image size of 640, batch size of 16, and 10 for the epochs. 

**Model Testing:**

The paths for the trained model, test images, and output directory are defined. Running this line of code performs object detection on the test images and saves the results in the specified folder. Displaying the output images allows the user to visually verify the model's performance in detecting objects in the test dataset.



**Checking Project Performance:**

This code evaluates the trained model’s performance by calculating various metrics, such as Precision, Recall, F1-score, and accuracy to provide insight into the model's effectiveness in identifying objects. Average inference speeds are calculated as well, to understand model efficiency and performance.



**Challenges:**

Some objects in the images are not detected due to the object in the image just like this:

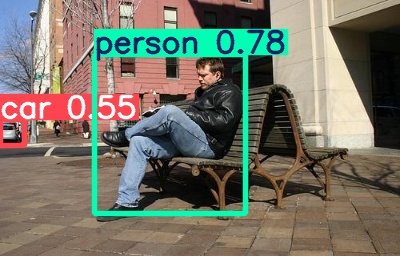


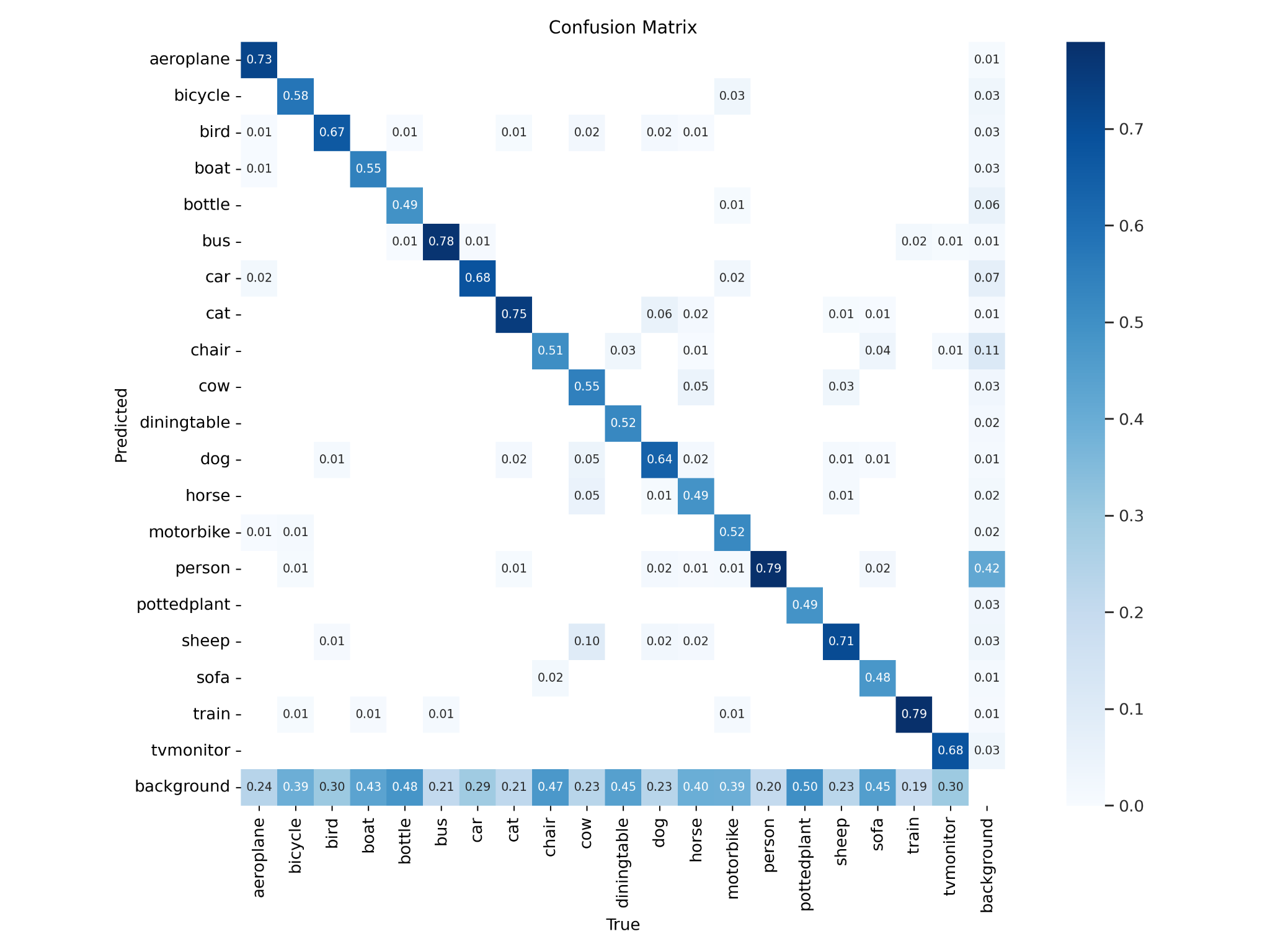
This happens because some classes do not have enough sample images and some are not visually similar to others. To avoid this, it is better to ensure having a balanced dataset among all the classes..

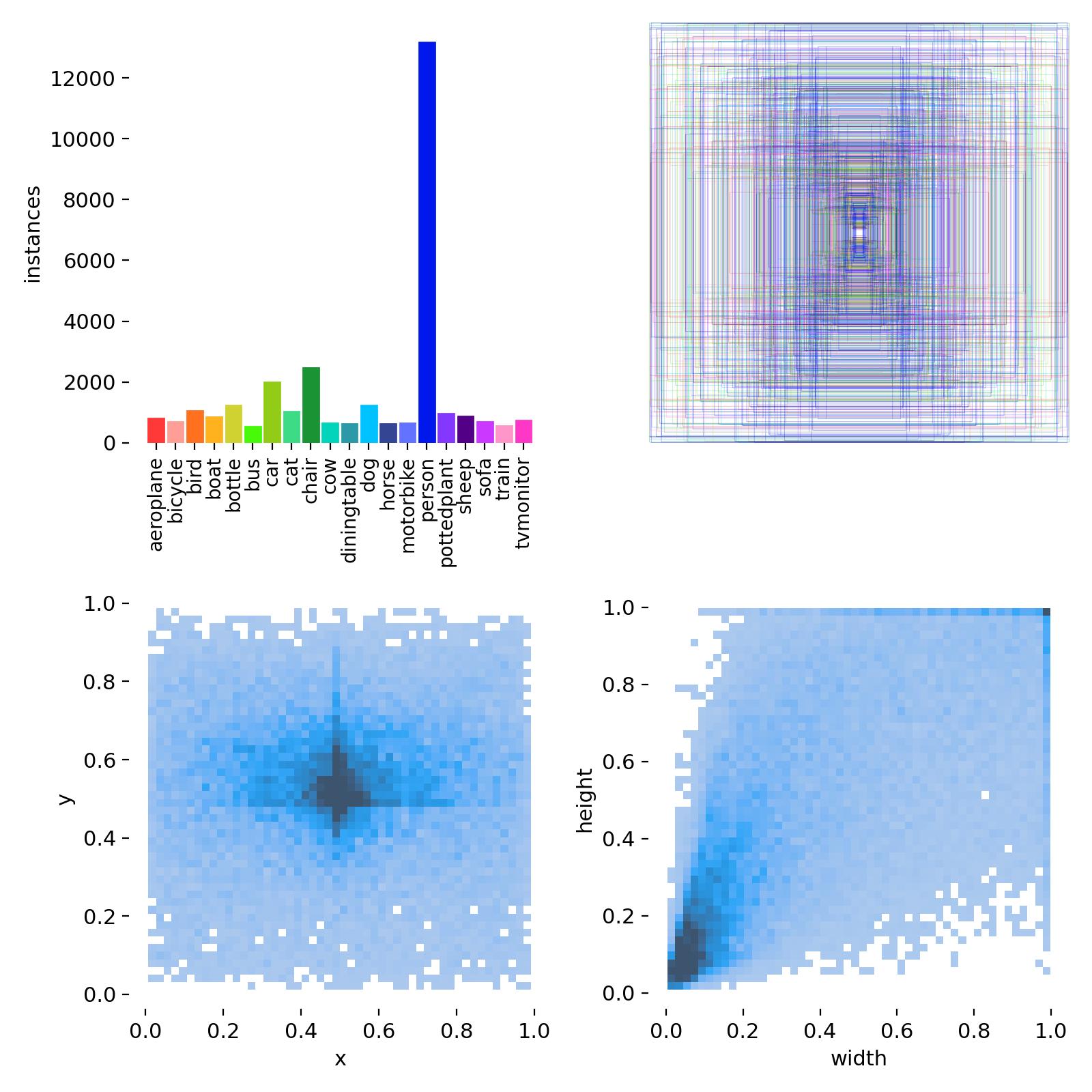
**Image Results:**

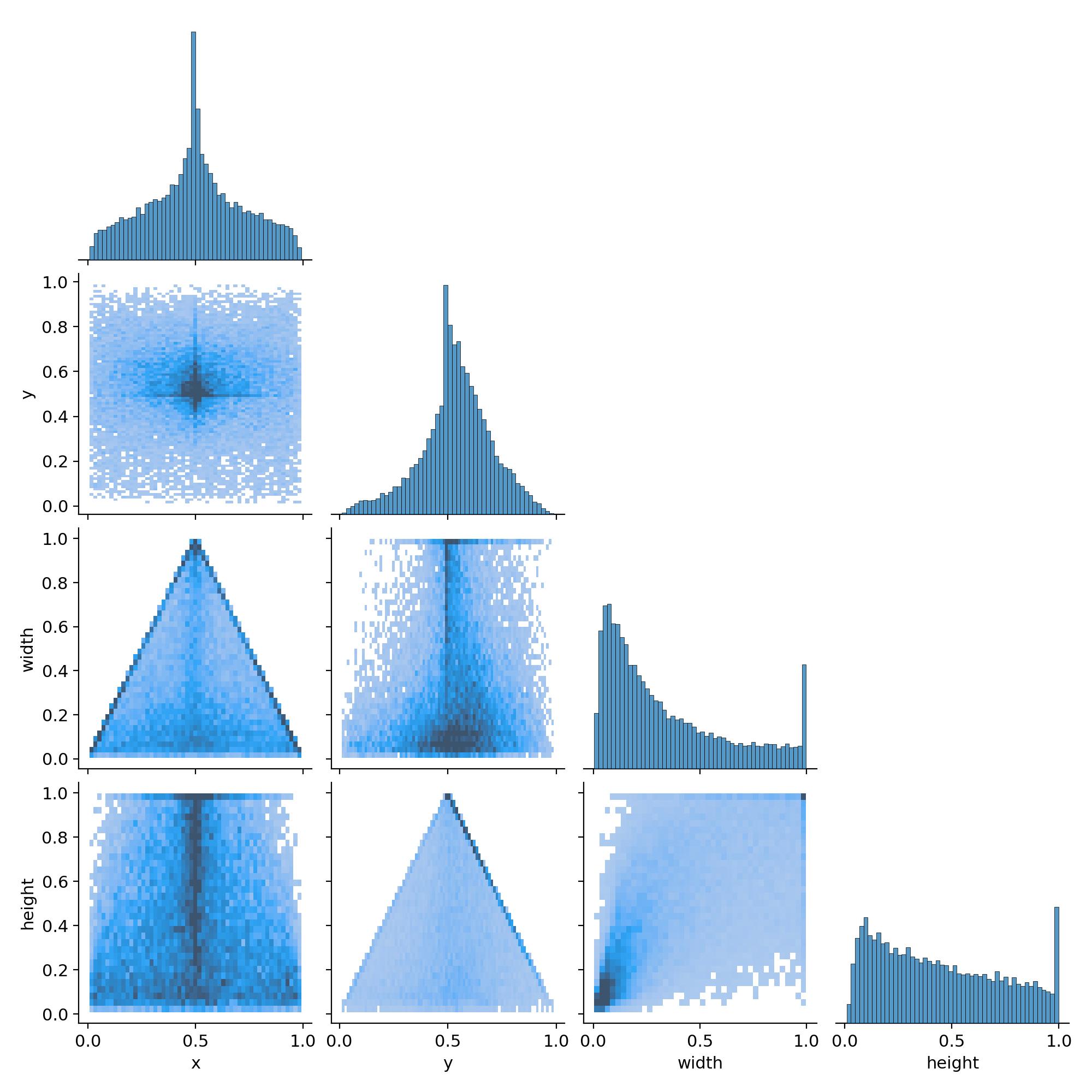


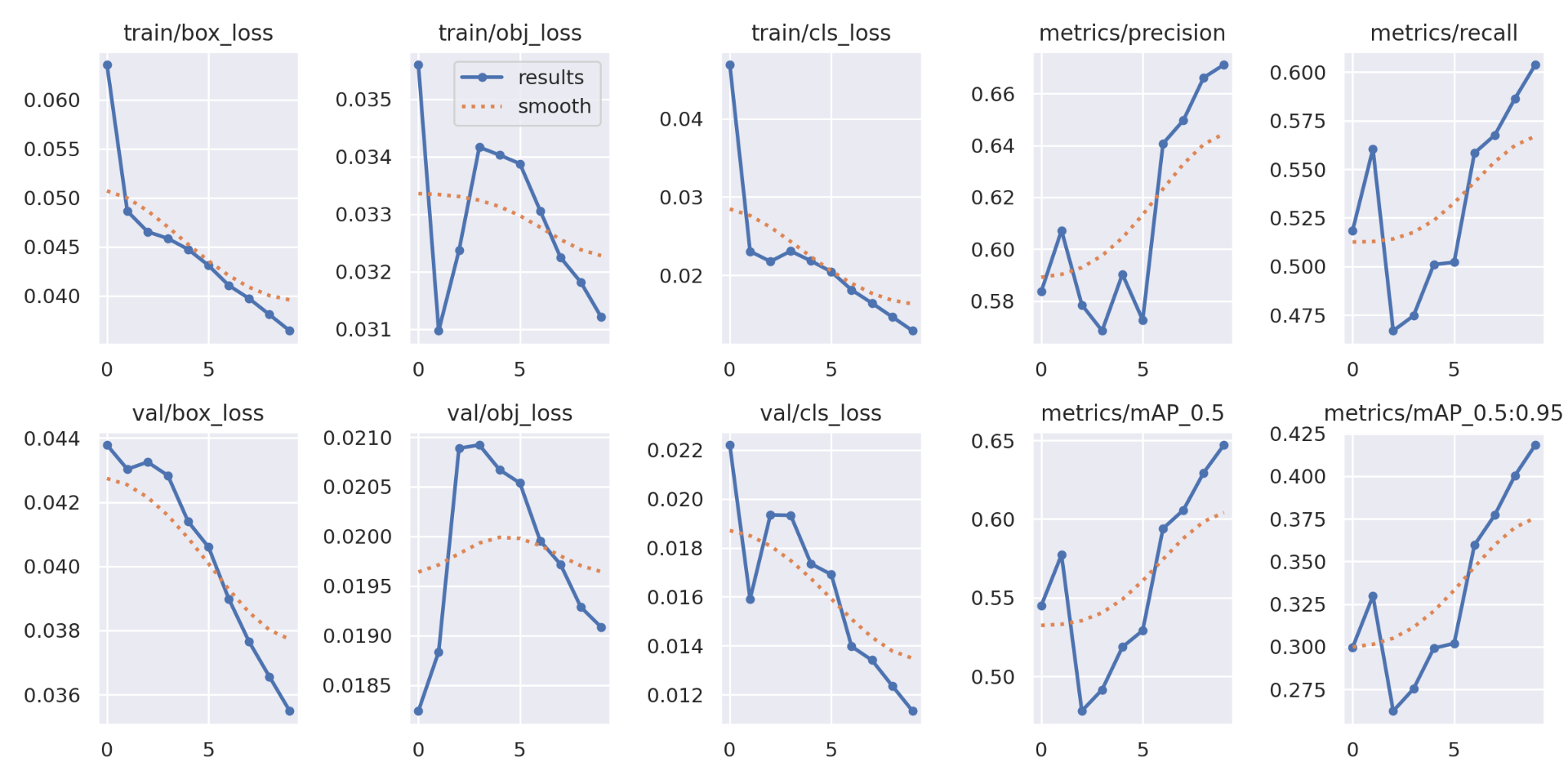


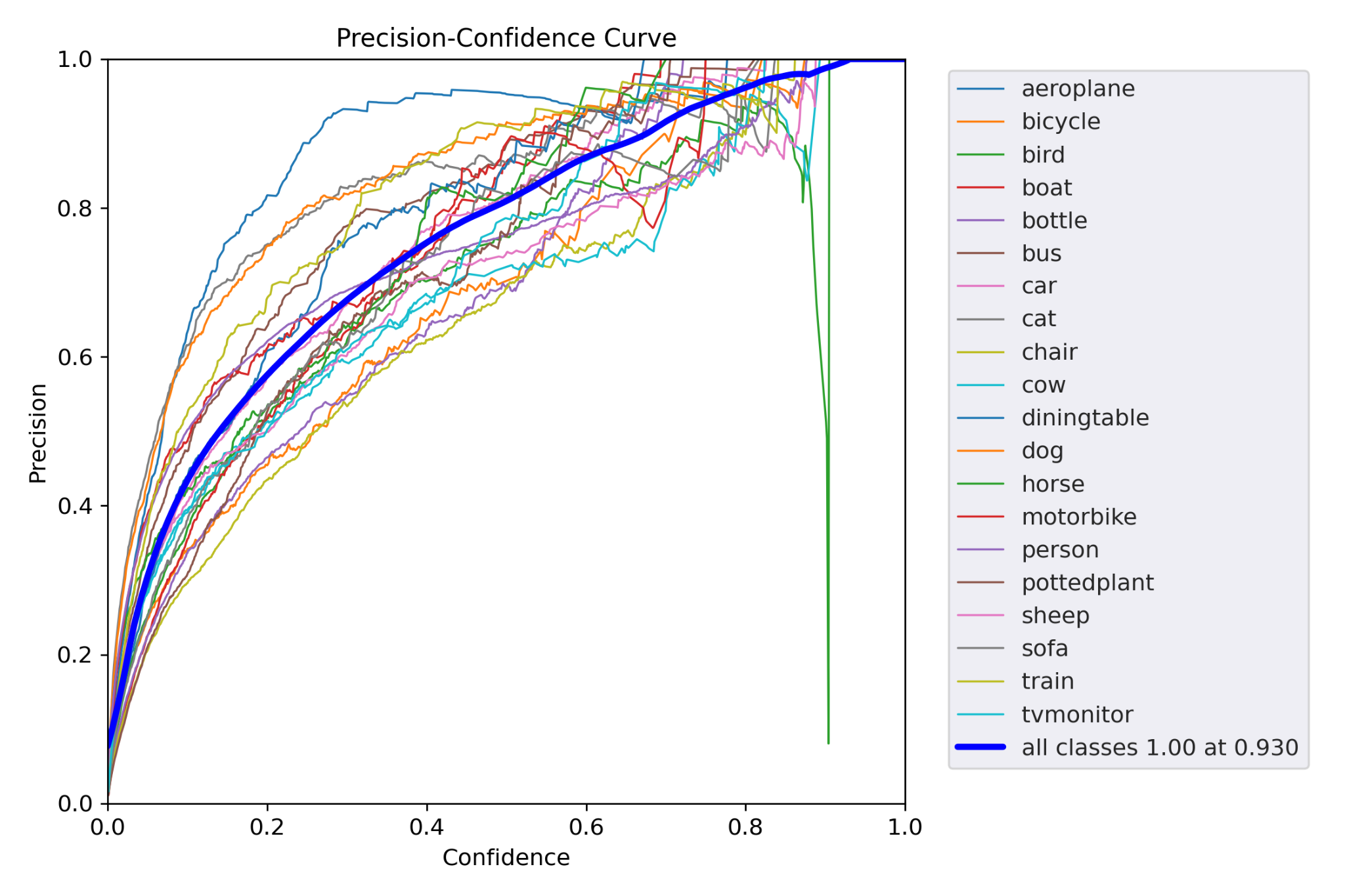


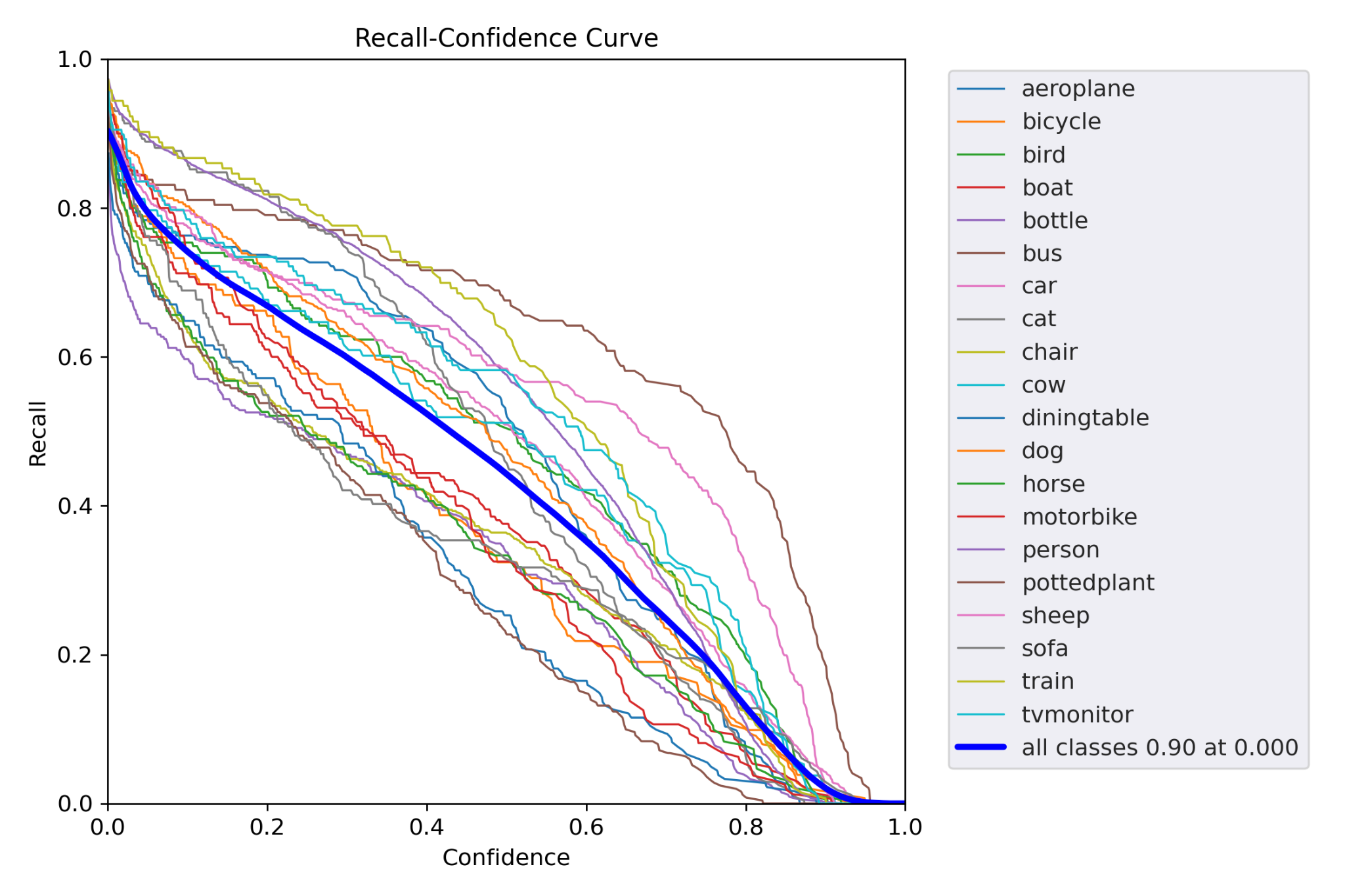
**Graph Results:**



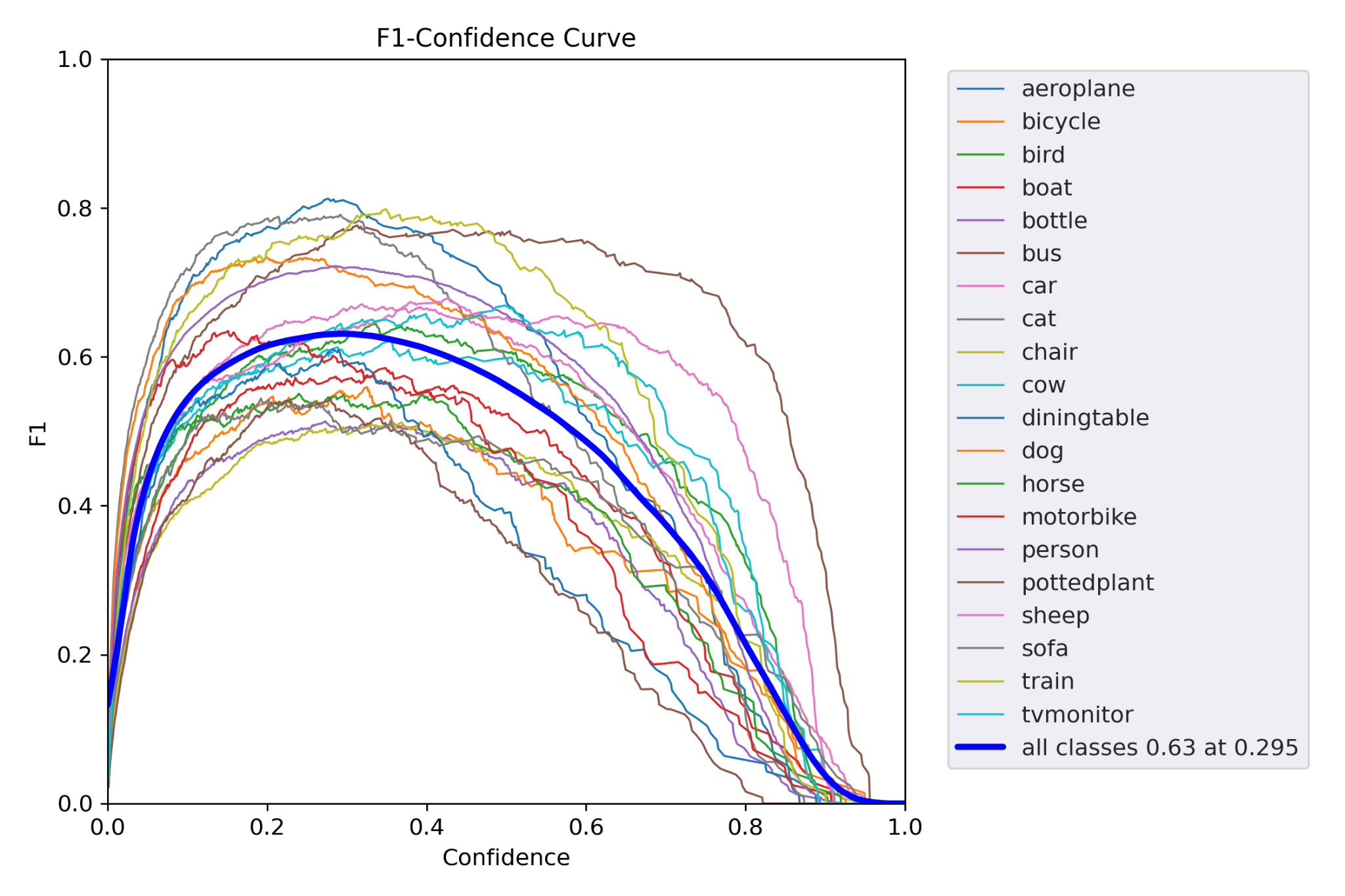








Graph for Precision Graph for Recall



Graph for F1