EAGLE EYE

OBJECT DETECTION

INTRODUCTION

Object detection is one of the fundamental tasks in computer vision and comes into play in a wide variety of applications ranging from self-driving vehicles to surveillance and retail automation. Particularly, YOLO (You Only Look Once) models became quite popular among the developers as they reliably perform object detection in a real-time manner. This was a pilot project of developing a flexible object detection model, which could be used to solve many real-time problems using single-shot architecture and the goal here was to create custom YOLOv8 model trained on a portion of Google Open Images (OID) dataset.

To make this work, we first had to convert OID's annotations into the format YOLO needs. This presentation walks through that process step-by-step, explaining how we transformed the dataset, set up the model, and tested its performance.

2. Dataset Preparation

Overview of the Google Open Images Dataset:

Google Open Images (OID) A massive dataset containing tens of millions of labelled images from millions of categories in various contexts and settings. OID is indeed a powerful resource but the problem is — OID format is not directly suitable to YOLO.

Therefore ,We had to convert everything into something that YOLO would like.

Mapping OID Labels to YOLO Class IDs:

We constructed a mapping from OID's string-based class labels to numerical IDs so that YOLO could actually understand the labels. This mapping enabled us to translate from each OID label (for example, “Person” or “Car”) to a number that YOLO is able to recognize.

Converting Bounding Box Coordinates to YOLO Format:

Compared to OID which describes boxes using the left, right, top and bottom edges, YOLO requires instead the box be described by its center point coordinates, width and height. Therefore, computed those values then normalized to fall in the [0, 1] space based on the dimensions of each image. By doing the conversion, this enabled YOLO to understand the bounding boxes.

Creating YOLO Annotation Files

Once the labels and coordinates were adjusted, we created a new file in the type .txt containing annotations compatible with YOLO format for each image. Each file is built in a form readable by YOLO in the training process.

3. Model Training

YOLOv8 Architecture Overview:

YOLOv8 indeed draws all good things from all previous versions, but it brings new layers and modules in an improvement made to be faster and more accurate. We built YOLOv8 from scratch rather than using a pre-trained model, fully obtaining the flexibility to modify and fine-tune the model's structure for our dataset.

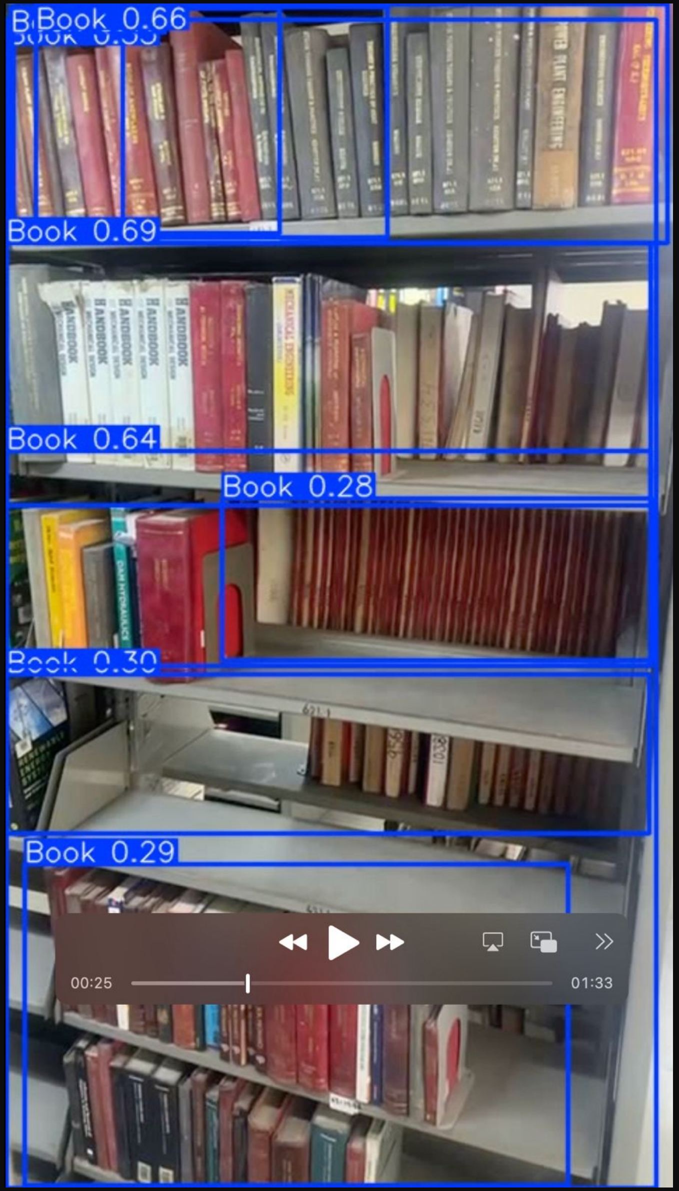
Training Process :

We divided the OID dataset into training, validation, and test sets using cross-validation for better performance.

We are using the YOLO v8 architecture, which provides robust functionality for object detection tasks such as object detection, classification, and confidence scoring. YOLO v8 is specifically designed to streamline these tasks, offering a high level of accuracy and efficiency right out of the box.

Comparison with Benchmark Models:

In exchange for the advantages of customizing the dataset and architecture to meet a particular set of requirements, our custom YOLOv8 model could deliver a similar amount of accuracy as compared to other pre-trained YOLO models



5. Discussion

Advantages of the Custom YOLOv8 Approach

Building YOLOv8 from the bottom up lets us dial the model in ways useful for real-time object detection. We now also have a smooth process to transform other datasets to YOLO, which is going to make it very easy to extend the utility of this model for any given task.

Limitations

Although OID is highly varied, some of the class labels within it were not very directly relevant to our target and diluted the capability of our model in some cases. Moreover, training a model like YOLOv8 with no training from scratch given is considerably computational expensive, and especially so for OID.

6. Conclusion

We illustrate the entire process of developing a custom YOLOv8 model, from transforming the dataset to training and testing. Using Roboflow, we converted the Open Images Dataset (OID) into a YOLOv8-compatible format, making it ready for model training. We then trained the YOLOv8 model from scratch using the yolov8n.yaml configuration file. This approach yielded an accurate object detection solution suitable for real-time applications, while also allowing for flexibility in utilizing other datasets with YOLOv8, making our model adaptable to various use cases.

7. Future Work

For now, looking forward in our plans, we will expand the project along the following lines:

Training of RCNN: In addition to YOLO, we would test out RCNN models. By training an RCNN on the very same dataset, we'll be able to compare the effectiveness of different architectures over the OID dataset and inspect potential improvements over object localization and classification.

Expanding Dataset Coverage: We shall train our model on the entire Google Open Images (OID) dataset with 9,00,000 images ; meaning all classes and samples available. We would then be adding to its already vast ability to generalize, which thus makes possible a wide variety of objects recognizable in different conditions.