Kecilin Assignment: YOLO v8

Misael Jordan Enrico

7 March 2024

1 Methodology

The instruction was to first create a YOLO v8 detection model that detects trucks. After that is done, the model is then modified so that it detects ambulance as well.

Given the instruction, my immediate thought was to start fine-tuning a pre-trained YOLO v8 model made by Ultralytics because truck is one among the classes this model is trained on. This model has already has its own truck feature extractor. Therefore, from 23 modules (similar to a group of layers), only the last module's weights are unfrozen. Freezing layers means that the gradients are not recorded and the weights are unchanged during training. What I'm hoping here is that the previous layers are able to produce feature maps associated to trucks and therefore we only need to train the last layer output layer.

For training, I collected 6022 images from COCO dataset accessible from RoboFlow. Initially, the dataset contains not only just truck but other 79 item classes, which sums up to 129311 images. I created a script that filters out all images that don't contain trucks. Unfortunately, I was only able to perform limited data augmentations due the the limited time given. The augmentation operations performed are rotation, reshape, color shift, and mozaic transformation done using Ultralytics library.

Moving on, the model is then modified to also be able to detect ambulance. Trucks have intersecting properties with ambulance. Therefore, I'd argue that feature extractor for truck composed of earlier layers can also detect features of ambulance. For this reason, used the truck detector model as the model base and start fine-tuning from here. The difference is that, unlike the previous training, I unfreeze two layers instead of just one. The reasoning behind this is that tuning one layer might not be enough to predict ambulance given the previous truck feature extractor composed of the previous layers.

In this occasion, I fine-tuned the model further using a combination of the truck dataset from the previous training session along with ambulance image dataset obtained RoboFlow consisting of 580 annotated images. Only limited augmentation methods are applied such as rotation, reshape, color shift, and mozaic transformation.

2 Answer to Questions

2.a How Truck Detector Model Accuracy is Measured

There are two types of accuracy for object detection problem; class accuracy and bounding-box accuracy.

I consider class accuracy to be similar to the accuracy of classification problem. Therefore, metrics such as precision, recall, and F1-score is suitable to quantify class accuracy. Plots such as confusion matrix, F1 curve, and PR curve might give us additional insights as well.

For bounding-box accuracy, some of the often used metrics are mean average precision 50 (mAP-50) and mean average precision 50 to 95 (mAP-50-95). The metric mAP basically calculates the proportion of the predicted bounding box that intersects with the actual bounding box with respect to the total area covered by both bounding box. It ranges between 0 and 1. Additionally, mAP50 counts how many predicted boxes actually have mAP higher than 50%. Finally, mAP-50-95 measures the average mAP between 50 and 95 with an increment value of 5 (This value can be modified).

2.b How Ambulance/Truck Detector Model Accuracy is Measured

The mAP-50 and mAP-50-95 are a good metrics to measure accuracy. However, I'd focus more on class-oriented accuracy such as precision, recall, and F1-score since this has become multi-class detection problem. This is important especially because truck can be really similar to ambulance and misclassification is more prevalent.

2.c How to Measure Time Taken for Inference

1. Record the time it takes to predict, say 100000 images. Then divide the time with 100000 to get inference time per image. Selecting a high number is good to avoid measuring overhead time.

2. Generally, we want inference time to be equal or less than 16.66 ms, which corresponds to 60 frames per second or higher. I think it's fast enough for average streaming processing.

This method also measures time taken for data movement from host (CPU) to device (GPU). To get true GPU runtime, I usually use Nsight compute to profile CUDA program. I don't know if Pytorch has comparable profiler.

2.d How to Increase Inference Time

Here are some strategies I could think of to reduce inference time, listed in ascending order of aggressiveness.

1. Use better accelerator. In general, run on GPU that has higher throughput and memory bandwidth.

2. Use a shallower network. Accuracy might be affected.

3. Port the model into TensorRT version. This framework speeds up inference by combining layers and other architecture optimizations.

4. Quantization. Using simpler weights representation might speed up operations.