Report on the car detection task

Viktor Rakhmatulin

July 16, 2024

Abstract

In this project, we developed a system to detect cars with high localization accuracy. The model meets the client's requirements with a confidence threshold of 0.9 and an Intersection over Union (IoU) of at least 0.8. We used a pre-trained object detection model to evaluate its performance on a test dataset of 100 car images. The model achieved a mean Average Precision (mAP@0.5) of 0.827 and mAP@0.5:0.95 of 0.676. For confidence $\geq$ 0.9 and IoU $\geq$ 0.8, the model achieved a precision of 0.9272, recall of 0.6222, and F1 score of 0.7447. The mean inference speed of 7.3 ms per image at a resolution of 640x640 was achieved on an Nvidia GeForce RTX 2070 SUPER Mobile. For deployment, the model was exported in ONNX and TorchScript formats. Inference was configured in Docker using a bash script that specifies the build and run commands for the image.

1 Summary

The detection task was conducted using a pre-trained model. Section 1.1 describes the implementation features and technical choices made for the task. Section **??** presents the achieved results, and the final section discusses possible improvements. Deployment ability is also evaluated based on the image size (smaller is better) and the use of standardized formats like ONNX or TensorRT.

1.1 Preliminary Considerations

Detecting cars is a widely-used task in various fields. Among the competing solutions, Detectron2 from Facebook offers detection models trained as of 2019 [5], and Hugging Face [1] hosts state-of-the-art transformer-based solutions known for their high accuracy. The YOLO series of models continues to evolve in popularity due to its balance between performance and computational efficiency.

For our prototype solution, we decided to implement a YOLO model due to its ease of use and the availability of pre-trained models suited for our task. Transformer architectures, while accurate, have significant computational and memory demands, making them less practical for some applications. The Ultralytics team provides a comprehensive API for new models, facilitating faster development. Additionally, YOLOv10 outperforms RT-DETR in both box mAP and inference time performance [3]. However, it is important to note that YOLO's GPL-3.0 License requires any changes to the source code and obtained weights to be made publicly available [4].

We utilized a fine-tuned network for our task. Specifically, we found a fine-tuned YOLOv5 model trained on a few relevant classes (Car, Motorcycle, Truck, Bus, Bicycle). The fine-tuned model performed better for the specific classes of interest due to its specialized training. Other fine-tuned models, such as those available on Kaggle [7], could also be used.

Several open-source datasets containing cars are available, including the Waymo Open Dataset [6] and the KITTI dataset, both of which provide images of cars with annotated bounding boxes. Generally, larger datasets improve the generalizability of the network. For simplicity in evaluation, we used the dataset provided with the fine-tuned network to simulate the evaluation of the network under custom conditions.

1.2 Results and Discussion

We achieved the following results: mAP50: 0.827, mAP50-95: 0.676, Speed: 0.2ms pre-process, 6.1ms inference, 1.0ms NMS per image at shape (32, 3, 640, 640) For confidence $\geq$ 0.9 and IoU $\geq$ 0.8:

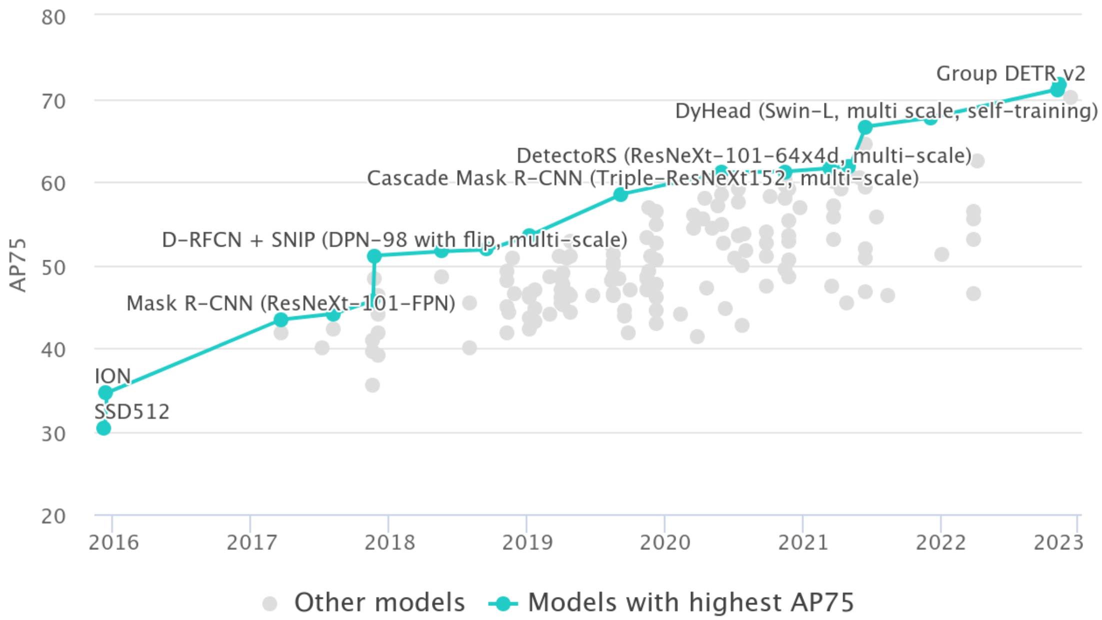

Figure 1: This figure demonstrates the AP75 of state-of-the-art models trained on results from the COCO dataset [2].

- Precision: 0.9272
- Recall: 0.6222
- F1 Score: 0.7447

YOLO calculates scores for maximum F1 confidence threshold by default:

- Precision: 0.826
- Recall: 0.872

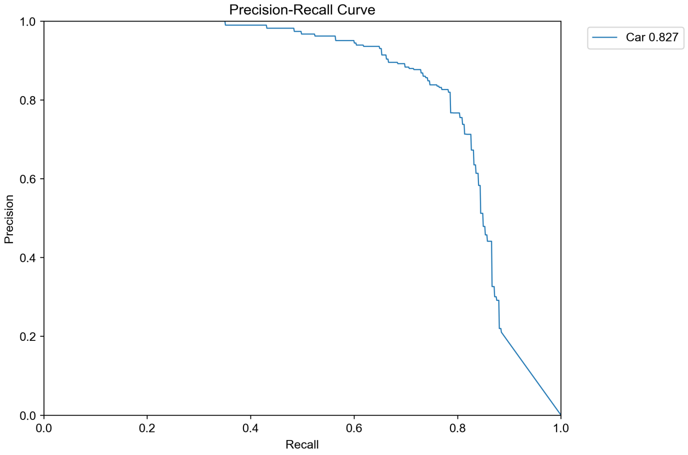

Figure 2: Precision-Recall Curve for cars only

Ways to improve detection quality, excluding fine-tuning of the original model:

- **Data Augmentation Techniques:** Implement techniques such as horizontal flipping, multi-scale inference, or rotation during training to increase diversity in the training data, and then average the predictions during inference to obtain more robust results.

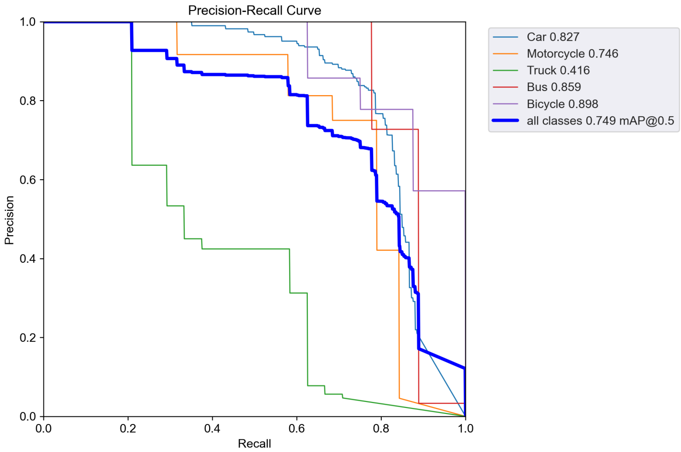

Figure 3: Precision-Recall Curve for all classes

- **Ensemble Methods:** Combine predictions from multiple diverse deep neural network models (DNNs) to leverage complementary strengths and enhance overall detection performance.
- **Explore Advanced Pre-trained Models:** Investigate more sophisticated pre-trained models that are fine-tuned specifically for object detection tasks, which may offer superior performance compared to the baseline model.

Ways to improve deployment quality:

- **Include Only Runtime-Related Components in Docker:** This approach minimizes the memory footprint, reduces computational overhead, and speeds up installation by eliminating unnecessary components.
- **Switch to Faster Compiled Languages Like C++:** Using compiled languages such as C++ for critical parts of the code can significantly enhance performance compared to interpreted languages.
- **Optimize Model for Inference:** Utilize tools like TensorRT or ONNX Runtime to optimize the model for faster inference, which can include layer fusion, precision calibration, and hardware-specific optimizations.
- **Monitoring and Logging:** Integrate robust monitoring and logging to track performance metrics, identify bottlenecks, and facilitate troubleshooting.

2 Limitations

There are several limitations to this project:

- **Dataset Size and Diversity:** The test dataset consists of only 100 car images, which may not be representative of real-world scenarios with diverse backgrounds, lighting conditions, and occlusions.
- **License Constraints:** The GPL-3.0 License of the YOLO model requires making available any changes in the source code and obtained weights, which might be restrictive for certain commercial applications.
- **Model Generalization:** While the model performs well on the test dataset, its ability to generalize to unseen data or different domains (e.g., different types of vehicles or environments) is not thoroughly evaluated.
- **Real-Time Performance:** Although the model demonstrates good inference speed, achieving real-time performance under varying conditions and with larger input sizes might be challenging.

References

[1] https://huggingface.co/

[2] Papers with code website https://paperswithcode.com/sota/object-detection-on-coco?metric=AP75

[3] YOLOv10 Performance, https://arxiv.org/abs/2405.14458

[4] YOLO License Information, https://github.com/ultralytics/ultralytics/issues/5691

[5] Detectron2 Model Zoo, https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

[6] Waymo Open Dataset, https://waymo.com/open/

[7] Kaggle KITTI Object Detection, https://www.kaggle.com/code/shreydan/kitti-object-detection-yolov8n/notebook#Model