Enhanced Deployment of YOLO V8 on Nvidia Jetson Nano through Model Compression Techniques

Urmil Jagad
School Of Engineering and Applied Science
Ahmedabad University
Ahmedabad, India
urmil.j@ahduni.edu.in

Harsh Loriya
School Of Engineering and Applied Science
Ahmedabad University
Ahmedabad, India
harsh.l2@ahduni.edu.in

Rahul Patel
School Of Engineering and Applied Science
Ahmedabad University
Ahmedabad, India
rahul.p2@ahduni.edu.in

***Abstract*— This report describes the first steps in the project "Enhanced Deployment of YOLO V8 on Nvidia Jetson Nano through Model Compression Techniques". The first month was spent gaining a solid understanding of the key technologies involved. This involved looking into YOLOv8, its architecture, the Nvidia Jetson Nano platform, and various model compression approaches. To obtain real experience with model compression, post-training quantization was applied to a sample model on Google Colab and Kaggle with the help of Pytorch tutorials. Although the project is still in its early phases and has not yet been deployed on the Jetson Nano.**

Keywords—DL, YOLOv8, Model Compression Techniques, Quantization

I. INTRODUCTION

The rapid growth of computer vision technologies has increased the requirement for efficient object detection systems. YOLO (You Only Look Once) is distinguished as a real-time object detection system by its outstanding speed and accuracy. In this research, we explore the complexities of YOLOv8, the most recent version of the YOLO architecture, with the goal of improving its deployment on the Nvidia Jetson Nano, a popular edge computing platform. The Nvidia Jetson Nano has significant potential in deploying deep learning models at the edge, enabling real-time inference in resource-constrained environments.

Furthermore, model compression techniques play an important role in optimizing the deployment of deep learning models to edge devices. Model compression techniques make efficient inference possible by reducing model size and computational complexity while maintaining accuracy. Throughout the past month, we have focused on understanding YOLOv8's architecture, learning about the potential of Nvidia Jetson Nano, and experimenting with various model compression techniques.

In particular, we explored and tried to implement post-training quantization to some extent. Post-training quantization is a popular model compression technique. Post-training quantization reduces numerical representation precision by quantifying the model's weights and activations, resulting in reduced model sizes and faster inference times. PyTorch tutorials were used to construct post-training quantization, with experiments conducted on Google Colab and Kaggle platforms.

Although the deployment phase on Nvidia Jetson Nano has yet to begin, the groundwork established in understanding YOLOv8, Nvidia Jetson Nano, and model compression techniques serves as a solid foundation for the project's succeeding stages. In the following months, we hope to smoothly integrate these components to provide an optimised deployment of YOLOv8 on Nvidia Jetson Nano, allowing for real-time object recognition in edge computing environments.

II. UNDERSTANDING OF YOLO V8

Ultralytics has released YOLOv8, the most recent version of YOLO. As a cutting-edge, state-of-the-art (SOTA) model, YOLOv8 builds on prior versions' success by offering new features and upgrades that boost performance, flexibility, and efficiency. YOLOv8 covers a wide range of visual AI tasks, including as detection, segmentation, posture estimation, tracking, and classification. This versatility enables users to apply YOLOv8's capabilities to a wide range of applications and areas. UNDERSTANDING OF MODEL COMPRESSION TECHNIQUES

Model compression reduces the size of the neural network while maintaining the accuracy. The resulting models are memory and energy-efficient.

A. Weight Quantization

Quantization refers to techniques for carrying out computations and storing weight/tensors with lower bitwidths than floating point precision. High speed vectorized operations on a variety of hardware platforms and a more condensed model representation are possible with it. The INT8 quantization feature of PyTorch allows for a 4x reduction in model size and memory bandwidth needs. Being limited to the forward pass for quantized operators, this method is mainly used to accelerate inference.

Post-training static quantization involves converting weights from float to int and feeding data through the network. It computes distributions of different activations by inserting observer modules at different points. These distributions are used to determine quantization at inference time. A simple technique is to divide the entire range of activations into 256 levels, but more sophisticated methods are supported. This step allows for passing quantized values between operations, resulting in a significant speed-up by converting them to floats and back to ints.

III. METHODOLOGY

We started by understanding the problem. Our method was to apply the model compression techniques on a trained Yolov8 model. So, we started by training the publicly available version of the Yolov8. After understanding the model and the techniques of the weight quantization, we decided to use coco dataset for the re-training part. We have

trained the Yolov8 model on the coco dataset and have a trained version of the same.

IV. Results

We have a trained Yolov8 model on Coco dataset. And we are trying to implement the weight quantization on the trained model. Currently we are working towards the coding part of the weight quantization.

- Statistics

Precision	0.61124
Recall	0.90002
Speed(ms)	116.058
Accuracy approx.	72.797%
Model Size (in bytes)	12944872

V. Conclusion

The first month of the project "Enhanced Deployment of YOLO V8 on Nvidia Jetson Nano through Model Compression Techniques" was dedicated to laying a solid foundation. A solid understanding of YOLOv8, the Jetson Nano platform, and numerous model compression strategies has been obtained. Additionally, hands-on practice with post-training quantization has been tried to implement through PyTorch tutorial. But the implementation was not successful, yet it gave a good knowledge about it.

While deployment on the Jetson Nano has not yet began, the insights gained during this initial phase will be useful in the coming steps. Future work will focus on adapting the chosen compression technique(s) to the YOLOv8 model, implementing the compressed model on the Jetson Nano, and assessing its performance.

VI. Future work

While previous attempts for post-training quantization using PyTorch tutorial were unsuccessful, they did provide useful insights into the actual implementation of model compression techniques. Moving forward, we will review post-training quantization and investigate other approaches appropriate for the YOLOv8 model. We will also look into model pruning, a complementary technique for reducing model size by removing unnecessary connections. We hope to determine the best way for efficiently deploying YOLOv8 on the Jetson Nano platform by developing and comparing the performance of various compression strategies.

References

[1] *Convert and Optimize YOLOv8 with OpenVINO™ — OpenVINO™ documentation*. (n.d.). https://docs.openvino.ai/2023.3/notebooks/122-yolov8-quantization-with-accuracy-control-with-output.html#:~:text=%2C%20log%3DFalse

[2] LoriyaHarsh. (n.d.-a). *GitHub - LoriyaHarsh/CV_2024_3_Group_12*. GitHub. https://github.com/LoriyaHarsh/CV_2024_3_Group_12

[3] Krishnamoorthi, R. (2018, June 21). *Quantizing deep convolutional networks for efficient inference: A whitepaper*. arXiv.org. https://arxiv.org/abs/1806.08342

[4] Ultralytics. (2024, March 13). *Home*. Ultralytics YOLOv8 Docs. https://docs.ultralytics.com/

[5] *Quantization — PyTorch 2.2 documentation*. (n.d.). https://pytorch.org/docs/stable/quantization.html

[6] Pokhrel, S. (n.d.). *4 popular model compression techniques explained*. https://xailient.com/blog/4-popular-model-compression-techniques-explained/

[7] (beta) Static Quantization with Eager Mode in PyTorch — PyTorch Tutorials 2.2.1+cu121 documentation. (n.d.). https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html