EMERGING TECH CONFERENCE – Edge Intelligence

Volume 03, 2024, Page 46 – 52

Proceedings of Emerging Tech Conference:
Edge Intelligence 2024

Edge deep learning for low capabilities devices

Fotis Filippou[1], Fotis Foukalas[1] and Theodoros Tsiftsis[1]
[1] *Department of Informatics and Telecommunications, Lamia, University of Thessaly*
fotisfilippoy@gmail.com, ffoukalas@uth.gr, tsiftsis@uth.gr

Abstract

Nowadays, Machine Learning (ML) is being used to construct many applications in domains such us object detection, image classification, speech to text etc. Deep Neural Networks (DNNs) are the core of Machine Learning as they offer remarkable accuracy and performance across various tasks. Despite their powerful capabilities, DNNs often require substantial computational resources, which can be challenging to manage, especially when deploying them on edge devices. So, these models have to be optimized before being deployed to these devices. Optimizing a model means making it smaller and more efficient without losing too much performance. Even though techniques like pruning reduce the number of parameters, the goal is to keep accuracy and speed as close as possible to the original. We are going to present a hybrid solution combining two techniques, pruning and quantization. Pruning is the process of eliminating inessential weights and connections in order to reduce the model size. Once the unnecessary parameters are removed, the model is quantized by converting the weights of the remaining parameters from 32 floating point precision to half. We verify and validate the performance of this hybrid approach using the COCO dataset (contains 80 classes) and the pre-trained YOLOv8 model. At the final stage, the hybrid model is deployed on an edge device, the NVIDIA Jetson Nano (4GB).

1 Introduction

In the last decade, Machine Learning has been the core for our everyday life. DNNs have a big impact on the performance in many applications such as computer vision, natural language processing (NLP), speech recognition etc. (1) (2). Despite of this enormous success of DNNs there are several problems that are being considered about the deployment of such models in computation-constrained environments (mostly on edge devices). These problems are usually created by the large model size and the high computational cost which requires in most cases an edge device with a “strong” GPU in order to avoid these problems (2). So, it is very crucial to compress these neural networks in order to reduce their size and computational demands, making them more suitable for deployment in edge environments.

Accomplishing these compression goals, requires some techniques that are being used more and more in the last years and expound a big interest in the ML community. Based on their properties, these techniques are split to four categories (3):

1 pruning and quantization
2 low-rank factorization
3 transferred convolutional filters

4 knowledge distillation

Parameter pruning and quantization techniques focus on identifying and eliminating redundant or non-essential model parameters. Low-rank factorization methods leverage matrix and tensor decomposition to approximate the most informative parameters in deep neural networks (DNNs). Transferred or compact convolutional filter approaches design specialized convolutional filters to reduce the parameter space, thereby conserving storage and computational resources. Knowledge distillation methods involve training a smaller, more efficient neural network to replicate the output of a larger model. (3)

In this paper, we are planning to examine a hybrid solution for the YOLO model using a hybrid approach that includes pruning and quantization. This is done in order to reduce the model size and number of parameters without compromising the model performance (4). A similar work has been done from the authors of paper (4) where they have applied this hybrid approach in 3 DNNs (ResNet-56, ResNet-110 and GoogLeNet), that are used for image classification. On an average, the authors have achieved to reduce the number of flops and parameters by 40% and simultaneously loosing not more than 2% in accuracy of the model. The difference in our work is that we use this hybrid approach on YOLO model which is not an image classification model but an object detection model.

This paper is organized into 4 sections. The first one is the introduction, that is already described. The section 2 focuses on the previous works on model compression. Section 3 presents the whole pipeline (Figure 1) that has been implemented, started from training till deploying. Last, Section 4 includes the conclusions of the method that has been followed and future work.

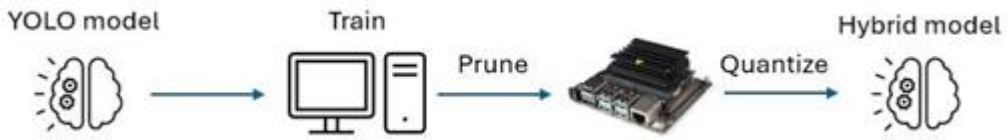

Figure 1: Pipeline of the hybrid technique

2 Related Work

The authors of paper (5) have investigated of complex DL models on optimizing their functionality on an embedded device, particularly on the NVIDIA Jetson Nano. They have measured the performance of image classification and video action detection models based on the inference speed and their experimental results have shown that the optimized models exhibit on average 16.11% speed improvement over their original edition. More specifically, they convert a PyTorch model into a TensorRT engine within 2 steps. Initially, the PyTorch model is being exported into ONNX (Open Neural Network Exchange) format. Next, from the ONNX file, the TensorRT engine is built using the TensorRT Python API. This involves several stages, including creating a network definition, importing the model through the ONNX parser, and building the TensorRT engine with a builder. Moreover, the inference process with the TensorRT engine file includes 6 steps: (5)

1. creation of an inference execution context
2. memory allocation for input and output on the CUDA device
3. input data is transferred from the host into the input memory allocated on the CUDA device
4. TensorRT engine performs inference using the asynchronous execute API
5. the output is transferred back into the host memory

6. the stream used for data transfers and inference execution is synchronized to ensure the completion of all operations

On another paper (6), the authors have focused on Deep Learning Model Optimization (DLMO). They present a usage strategy of DLMO based on the performance evaluation through light convolution, quantization, pruning techniques and knowledge distillation, known to be excellent in reducing memory size and operation delay with a minimal accuracy drop and through experiments on image classification they derive possible and optimal strategies to apply deep learning into edge devices. More specifically, they explore strategies for utilizing DLMO by evaluating its performance across various combinations of Lightweight Convolution, Quantization, and Pruning techniques. They use the Canadian Institute For Advanced Research 10 (CIFAR10) and CIFAR100 datasets for the whole research. Also, they use VGGNet and ResNet as baseline networks and for comparison the lightweight MobileNet v1, v2 and v3 networks. So, in different use cases they evaluate the performances of several quantization techniques, which comprise Quantization Aware Training (QAT) and subtypes of Post Training Quantization (PTQ), i.e., Baseline Quantization (BLQ), Full Integer Quantization (FIQ) and Float 16 Quantization (F16). For the pruning technique, the performance improvement is analyzed by applying the training method to the basic Convolution Neural Network (CNN) and lightweight CNN technologies. By the finish of the whole process and the necessary tests, they conclude that quantization was the most effective in compressing the model size, but it introduced significant delays in data conversion. Also, the pruning technique was excellent in all aspects of model compression, accuracy loss minimization and delay minimization. Additionally, it is recommended to train the deep learning models using knowledge distillation, as this approach can enhance accuracy without causing any additional increases in latency or model size. Last, by classifying Artificial Internet of Things services according to three performance factors (accuracy, size, and delay), they identified the optimal combinations of DLMO techniques for different scenarios.

3 Pipeline

Every day, we watch more and more applications that are using DNNs to have a big impact on our life. Besides the use of DNNs in large computer systems, DNNs are used on many low performance edge devices such as smartphones. So, these models have to be compressed in order to be capable of running is such devices. The model that we have focused on for this paper is the YOLO and more specifically the version 8 of it. YOLOv8's architecture is an evolution of previous YOLO models, utilizing a convolutional neural network divided into two main parts: the backbone and the head. The backbone is based on a modified version of the CSPDarknet53 architecture, consisting of 53 convolutional layers enhanced with cross-stage partial connections. The head comprises multiple convolutional layers followed by fully connected layers responsible for predicting bounding boxes, objectness scores, and class probabilities. Notably, YOLOv8 integrates a self-attention mechanism in the head of the network and a feature pyramid network for multi-scaled object detection, enabling it to focus on various parts of an image and detect objects of different sizes and scales (7).

We adopt the Ultralytics framework (8) in order to download YOLOv8 and make the necessary actions in order to compress it. It’s also important to refer that YOLOv8 has 5 editions that differ on accuracy (mAP50-95) and speed. mAP50-95 is the average of the mean average precision calculated at varying intersection over union (IoU) thresholds, ranging from 0.50 to 0.95 and it's a measure of the model's accuracy considering only the "easy" detections (8). For the whole pipeline we are going to use the COCO dataset which contains 80 pre-trained classes. We aim to test all of the above models with this dataset of how the perform before and after compression on an edge device, the Nvidia Jetson Nano (4GB). First of

all, we have to train our model by loading each yolov8 pre-trained model and using the COCO dataset as parameter for the data. Also, we use 300 epochs for training for best results. Moreover, we apply parameters such as patience in order to prevent overfitting. The fact that training process requires high computational cost makes us clear that we should utilize a powerful GPU for the whole process in order to avoid extended training time.

3.1. Pruning

After the training process is completed, we have a ready to deploy model. So, after moving it to Jetson Nano we observe that the speed of inference is too slow and necessitates the use of model compression techniques. So, we will try to make the model faster and appropriate for the Jetson Nano by following a hybrid method that combines both pruning and quantization.

Model pruning refers to the act of removing unimportant parameters from a deep learning neural network model to reduce the model size and enable more efficient model inference. Generally, only the weights of the parameters are pruned, leaving the biases untouched (9). So, after identifying and eliminating unnecessary parameters, we take a lighter model that requires less computational power. It is particularly useful for deploying models on devices with limited resources such as Nvidia Jetson Nano. We can take more information about pruning and understand it better from Figure 2.

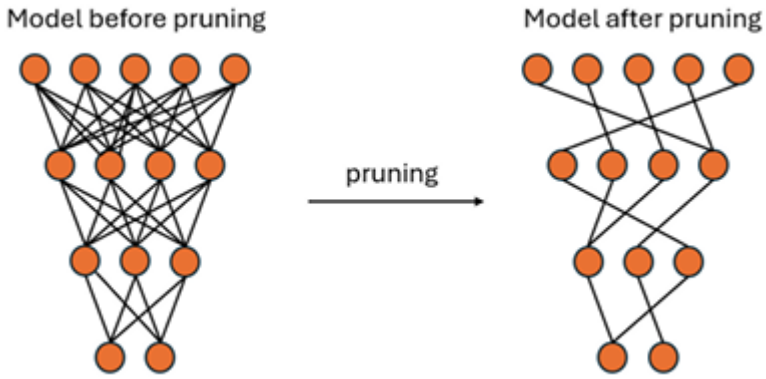

Figure 2: Pruning Process

We have created a custom function in order to prune our model. By this function, we can prune our model by an amount p (0<p

1 For each layer of the model, we check its type and if the layer is Conv2d we continue to step 3

2 Compute the L1-norm (unstructured) of the weight matrix of the layer (W) by the following equation:

$$||W_1|| = \sum_{i,j}|w_{i,j}| \qquad Equation\ 1$$

3 Sort the elements of W by their absolute values

4 Set the smallest weights to zero, based on the computed mask

5 Remove any applied masks to finalize the pruning process

6 Return the pruned model

3.2. Quantization

The next step of our hybrid technique, after pruning is done, is to quantize the pruned model. Quantization converts the model's weights and activations from high precision (like 32-bit floats) to lower precision (like 8-bit integers). By reducing the model size, it speeds up inference and loses a small percent in accuracy (8). We can see an example of Quantization in below figure.

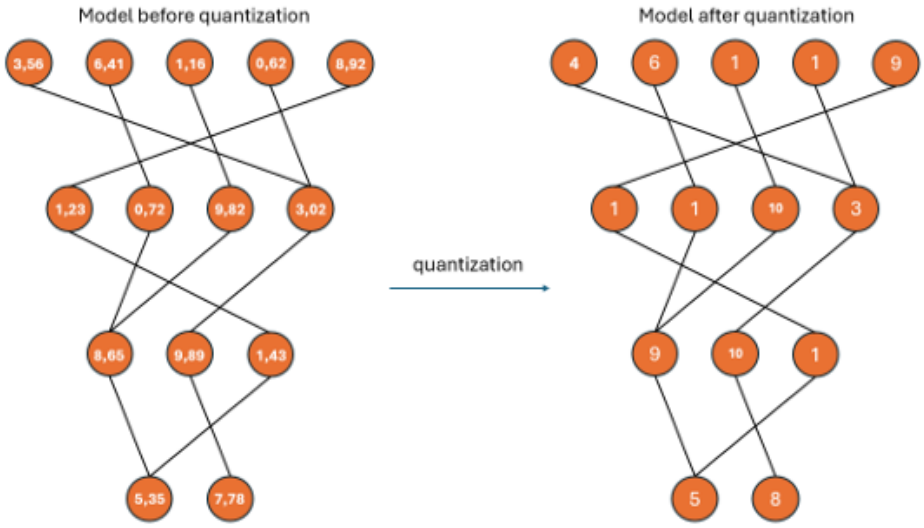

Figure 3: Quantization process

In order to implement quantization, we move our pruned model from the computer where it was trained and pruned to the Nvidia Jetson Nano. This is necessary because quantization needs to be performed on the edge device due to framework incompatibilities between different devices. The framework we use is TensorRT, which is specifically optimized for Nvidia devices like the Jetson Nano. TensorRT provides high-performance inference by leveraging the GPU capabilities of the Jetson Nano, and it supports various optimizations including precision calibration and layer fusion During the quantization process, the model is first converted to an ONNX file. Next, TensorRT generates a .engine file, which is the quantized model ready for deployment. There are several parameters that we can change on quantization like converting FP32 to FP16 or to INT8 which leads to a reduce on model size, lower power consumption and fast inference (8). In the figures we can see the results of mAP50-95 on figure 4 and of inference speed per frame (ms) on figure 5 for yolov8n, s and m versions before and after implementing the hybrid technique (5% pruned and quantized from FP32 to FP16) for Nvidia Jetson Nano.

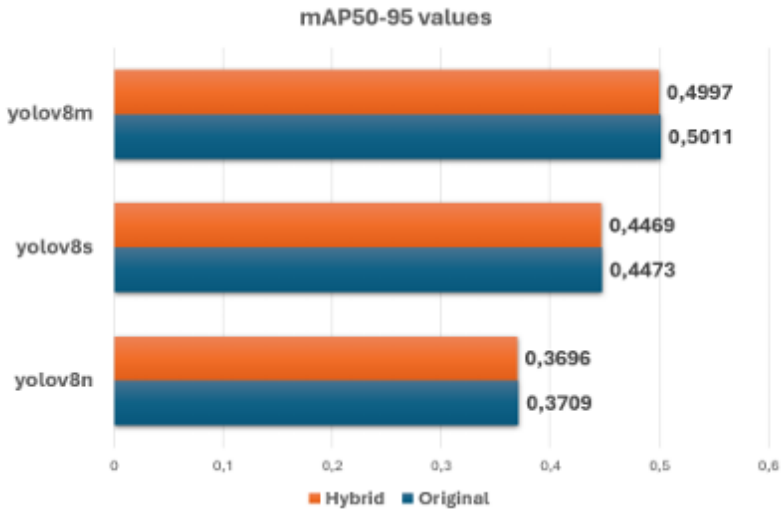

Figure 4: mAP50-95 for different version of YOLOv8 models on Jetson Nano

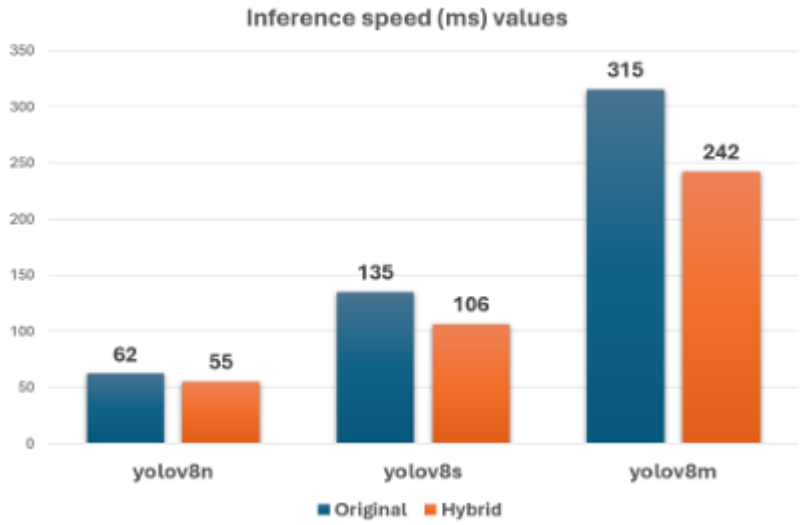

Figure 5: Inference speed (ms) for different version of YOLOv8 models on Jetson Nano

4 Conclusions and Future Work

By following this hybrid technique for different versions of YOLOv8 we can extract many conclusions based on results from figures 4 and 5. First, we can see a drop in mAP50-95 value by less than 0.5% which is normal to happen, due to pruning and quantization. Moreover, we observe that the inference speed is about 1.2X faster than original and by this fact the hybrid model can be easily deployed to edge devices. We should take in mind that we have converted FP32 to FP16(half) during pruning process whereas if we convert FP32 to INT8 we will see a bigger drop in mAP50-95 and a faster inference speed. Also, we have made all the tests for YOLOv8 models with COCO dataset but the whole pipeline can be applied to any PyTorch model that uses a pre-trained version of YOLO in order to be trained. This generalization is crucial

for deploying models in diverse real-world scenarios where the ability to adapt to different data and tasks is essential. As edge computing becomes increasingly prevalent, the importance of optimizing models for deployment in such environments cannot be overstated. The hybrid technique discussed here provides a robust framework for doing so, ensuring that models remain both accurate and efficient. As AI continues to expand into new domains and applications, techniques like these will be essential for making advanced models accessible and practical in a wide range of real-world scenarios.

For future work, advanced pruning strategies, like structured pruning or those guided by neural architecture search (NAS), could be investigated to remove redundant parameters more effectively, resulting in more compact models without significant accuracy loss. Also, another area for future research is dynamic quantization, where the model could adjust its precision based on the input complexity or available computational resources, thereby optimizing speed and efficiency. Expanding the evaluation of this technique across diverse datasets, especially those relevant to specific application domains like medical imaging or autonomous vehicles, will be crucial in assessing its generalizability.

Acknowledgment

This work has been funded by the Greek funded project "Connected smart cities for Greece 2.0 program" with code TAEDR-0536642.

References

[1] Kulkarni, Uday and Hosamani, Abhishek S and Masur, Abhishek S and Hegde, Shashank and Vernekar, Ganesh R and Siri Chandana, K. A Survey on Quantization Methods for Optimization of Deep Neural Networks. 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS). 2022, pp. 827-834.

[2] Peng, Hanyu and Wu, Jiaxiang and Zhang, Zhiwei and Chen, Shifeng and Zhang, Hai-Tao. Deep Network Quantization via Error Compensation. IEEE Transactions on Neural Networks and Learning Systems. 2022, pp. 4960-4970.

[3] Zhang, Yu Cheng and Duo Wang and Pan Zhou and Tao. A Survey of Model Compression and Acceleration for Deep Neural Networks. IEEE Signal Processing Magazine. 2020.

[4] Kulkarni, Narayan and Singh, Nidhi and Joshi, Yamini and Hasabi, Nikhil and Meena, S M and Kulkarni, Uday and Gurlahosur, Sunil V. Hybrid Optimization for DNN Model Compression and Inference Acceleration. 2022 2nd International Conference on Intelligent Technologies (CONIT). 2022, pp. 1-8.

[5] Tushar Prasanna Swaminathan, Christopher Silver, Thangarajah Akilan. Benchmarking Deep Learning Models on NVIDIA Jetson Nano for Real-Time Systems: An Empirical Investigation. June 26, 2024.

[6] Lee, Hyungkeuk and Lee, NamKyung and Lee, Sungjin. A Method of Deep Learning Model Optimization for Image Classification on Edge Device. Sensors. 2022.

[7] Modelbit. [Online] https://www.modelbit.com/model-hub/yolo-v8-model-guide.

[8] Qiu, Glenn Jocher and Ayush Chaurasia and Jing. Ultralytics YOLOv8. [Online] 2023. https://github.com/ultralytics/ultralytics.

[9] Neo, Marcus. Datature. [Online] February 29, 2024. https://www.datature.io/blog/a-comprehensive guide-to-neural-network-model-pruning.