Quantization Support Comparison for Tensorflow and Pytorch

Abhi Lad - al4363
MS Computer Science
Columbia University
New York, USA
al4363@columbia.edu

***Abstract*—In the present work, we conduct an empirical comparison of quantization support in two prominent deep learning frameworks, TensorFlow and PyTorch, utilizing the ResNet18 architecture trained on the CIFAR-10 dataset. Specifically, we assess three model variations: the baseline model, a model subjected to post-training quantization, and a model trained with quantization-aware training. Instead of manual quantization, we exploit the in-built tools provided by each framework, ensuring a fair comparison and facilitating reproducibility. Our evaluation metrics encompass model accuracy, memory consumption, computational time, and inference latency per sample. Our comparative study elucidates intriguing observations concerning the distinct performance characteristics and trade-offs associated with each model variation in the context of the respective frameworks. This research promises to guide machine learning practitioners and researchers in making informed decisions when implementing and optimizing models, particularly when considering the deployment of these models in resource-constrained environments.**

***Index Terms*—quantization, optimization, pytorch, tensorflow**

I. Introduction

Deep Learning models have shown impressive performance across various domains, including image recognition [1], natural language processing [2], and more. However, their deployment in resource-constrained environments such as mobile devices and embedded systems is often challenging due to their substantial computational and memory requirement [3]. Model quantization, which refers to the process of reducing the precision of the numbers that a neural network uses to represent the weights, has emerged as an effective technique to compress models without significant loss in performance [4].

Two popular deep learning frameworks, TensorFlow [5] and PyTorch [6], provide tools for quantization. However, there is a lack of comprehensive studies comparing the quantization support provided by these two frameworks. In this paper, we aim to fill this gap. We start with an extremely popular multimodal model Transformer [11] and Flicker8k [12] image captioning dataset and iteratively remove components which are not supported by profiling and quantization methods. We finally utilize ResNet18 [7], a widely-used model architecture for image recognition tasks, trained on the CIFAR-10 dataset [8]. The overall workflow of our project is presented in Fig. 1. We examine three variations of the model: the baseline model, a model subjected to post-training quantization, and a model trained with quantization-aware training.

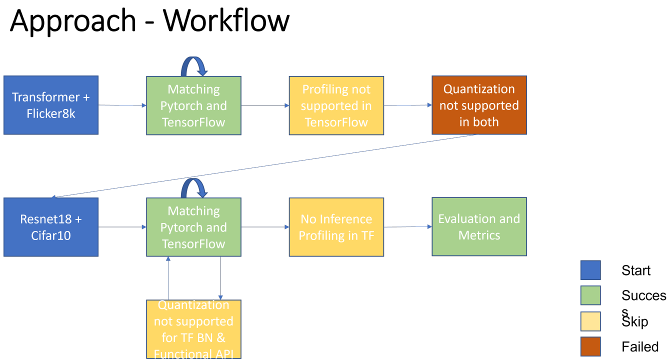

Approach - Workflow
Transformer + Flicker8k
Matching Pytorch and TensorFlow
Profiling not supported in TensorFlow
Quantization not supported in both
Resnet18 + Cifar10
Matching Pytorch and TensorFlow
No Inference Profiling in TF
Evaluation and Metrics
Quantization not supported for TF BN & Functional API
Start
Succes s
Skip
Failed

Fig. 1. The overall workflow of the project.

Our comparison is unique in that we utilize the tools provided by each framework for quantization rather than manual quantization, which ensures a more robust and fair comparison. We evaluate the models on several metrics including model accuracy, memory consumption, computational time, and inference latency per sample.

The insights from our study provide valuable guidance for practitioners and researchers looking to implement and optimize models for deployment in resource-constrained environments. Additionally, our work might motivate further research into the development of more efficient and effective quantization techniques.

II. Models and Dataset Description

A. Models

Transformer-based image captioning models serve as an excellent case for analyzing quantization in deep learning models due to their intricate design and the diverse range of operations they perform. The Transformer architecture, as proposed by [11], has revolutionized many areas of deep learning and has been pivotal in pushing the state-of-the-art in tasks such as image captioning [13]. These models exhibit a complex interplay of multi-head attention mechanisms, position-wise feed-forward networks, and normalization layers, which represent a broad spectrum of computational operations found in various other deep learning models.

Quantization [15] is a technique often used for model compression and efficient inference. It involves reducing the numerical precision of the model's parameters, which can lead to significant reductions in model size and computational requirements [14]. This makes it especially relevant for deploying models on resource-constrained devices. However, quantization can also introduce approximation errors, and understanding its impact on complex models like transformers is crucial. The diverse nature of operations in transformer models, from linear transformations to attention computations, makes them an ideal candidate for studying the effects of quantization.

However, after countless trials and failures due to framework limitations, in this study, we leverage the ResNet18 architecture, a popular choice for image classification tasks due to its efficiency and strong performance. ResNet18 belongs to the family of Residual Networks (ResNets), which introduced the novel concept of 'skip connections' or 'shortcuts' to address the issue of vanishing gradients in deep neural networks. The ResNet18 model comprises 18 layers, including convolutional and fully connected layers, and is known for its capability to extract intricate patterns and features from the input data.

For each framework (TensorFlow and PyTorch), we consider three variants of the ResNet18 model: the baseline model, the post-training quantized model, and the quantization-aware trained model. The baseline model is a standard ResNet18 model with no modifications. The post-training quantized model is the baseline model subjected to quantization after training, reducing the precision of the model weights. Finally, the quantization-aware trained model is trained with knowledge of the quantization process, allowing it to better adapt to the reduced precision during training.

B. Datasets

Flickr8k is an excellent multimodal dataset for analyzing quantization in deep learning models due to its diverse nature and combination of both images and textual descriptions. The dataset consists of 8,000 images, each accompanied by five different human-generated captions. This unique combination of visual and textual data allows researchers to explore and evaluate quantization techniques on models designed for a variety of tasks, such as image captioning, object detection, and sentiment analysis. The rich and varied content of the images in the Flickr8k dataset, which includes a wide range of objects, scenes, and activities, ensures that models trained on this dataset learn to recognize and describe complex patterns. This complexity makes it an ideal benchmark for assessing the impact of quantization on model performance, as it can reveal the trade-offs between model size, computational efficiency, and accuracy.

Moreover, the presence of multiple captions for each image encourages the development of models that can generate diverse and accurate descriptions, which can then be further analyzed in the context of quantization. By investigating the effects of quantization on models trained with the Flickr8k dataset, researchers can gain valuable insights into the robustness and generalization capabilities of quantized models in real-world applications and understand how to better optimize these models for efficient deployment on resource-constrained devices.

However, due to framework support issues, we train and evaluate our models on the CIFAR-10 dataset [8], a well-known benchmark dataset in the field of machine learning. CIFAR-10 consists of 60,000 32x32 color images, split into 50,000 training images and 10,000 test images. The images belong to 10 different classes, each represented by 6,000 images. Classes include objects such as airplanes, dogs, cats, trucks, and more.

The dataset provides a good balance of complexity and computational feasibility for our comparative study. Its small image size allows for faster training and evaluation of the models, while the diversity in the dataset ensures a robust test of the model's performance. The relatively small size of the dataset also allows us to study the effects of quantization more closely, as the impact of model size reduction and computational speedup will be more pronounced.

III. Training and Profiling Methodology

A. Training

We begin our experiments by training the baseline ResNet18 model in both PyTorch and TensorFlow using the CIFAR-10 dataset. The model is trained for 50 epochs using a standard training process, where we optimize the model parameters to minimize the cross-entropy loss between the model's predictions and the true labels. We use SGD optimizer and the learning rate is set at 0.1 and is decayed at specific milestones during the training process. A batch size of 128 is used for training, and data augmentation techniques such as random cropping and horizontal flipping are applied to the training data for model robustness. We remove the BatchNormalization layers from both Pytorch and TensorFlow implementations due to quantization support issues. We ensure that both the implementations are near identical in training specs, while also maintaining Sequential dataflow to account for quantization aware training.

After training the baseline model, we then proceed to quantize the model in two ways: post-training quantization and quantization-aware training. In post-training quantization, the model is quantized after the training process is complete, effectively reducing the precision of the model's parameters. This step involves calibrating the quantized model over training data. Pytorch has support for local device quantization model which converts the baseline model into int8 quantized model which we can access as normal Pytorch model. TensorFlow converts the quantized models into TF-lite format which represents bytes and thus cannot be accessed as normal models. In quantization-aware training, the model is made aware of the upcoming quantization during training, allowing it to adapt to the reduced precision. In Pytorch, this is done by fusing the layers and then adding quantization stubs on input and output layer of the model. While in TensorFlow, we annotate the entire model as quantization aware, and the training routine internally handles the QAT calls.

B. Profiling

To evaluate and compare the performance of the model variants, we conduct a detailed profiling of each model, focusing on three main metrics: model accuracy, time and memory consumption, and inference latency. The overview of the profiling flow is presented in Fig. 2.

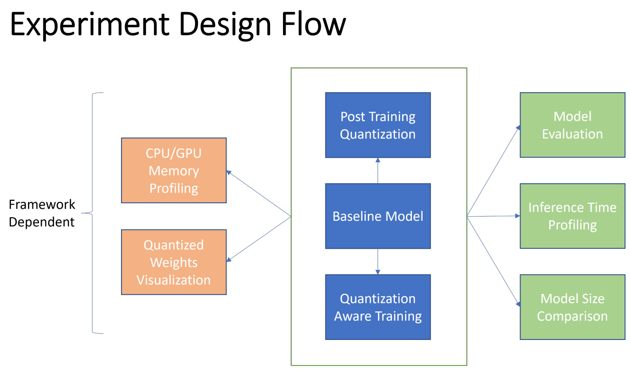


Fig. 2. The experiment and profiling overview of the models.

Model Accuracy: We measure the accuracy of each model variant on the test portion of the CIFAR-10 dataset. The accuracy provides a direct measure of how well the model performs its task, giving us a sense of the impact of quantization on model performance.

Time and Memory Consumption: We log the CPU and GPU time and memory consumption during both the training and evaluation phases. This allows us to understand the computational efficiency of each model variant and the impact of quantization on resource utilization. However, this step is framework dependent as TensorFlow does not have proper Profiling support for evaluation.

Inference Latency: We measure the average time taken to process a single sample during evaluation, also known as inference latency. This metric is crucial in real-world applications where speed is often as important as accuracy. This metric is mainly used in quantized models to quantify the performance gains for edge hardware.

For the profiling, we leverage the built-in profiling tools provided by PyTorch and TensorFlow. For PyTorch, we use the torch.profiler module, and for TensorFlow, we use the TensorBoard Profiler. Both tools allow us to measure time and memory consumption on both CPU and GPU, providing detailed insights into the computational behavior of the models.

By comparing these metrics across the three model variants in both PyTorch and TensorFlow, we aim to provide a comprehensive comparison of the two frameworks' quantization support and the implications of quantization for practical deep learning applications.

IV. Performance Tuning Methodology

The objective of performance tuning in our experiment is to optimize the computation efficiency of the ResNet18 models in both TensorFlow and PyTorch frameworks. The strategies we adopt for performance tuning are designed to minimize computational overhead and maximize the speed of model inference while maintaining a high level of model accuracy. Below, we discuss the performance tuning methodology employed for our study.

Hardware and Software Optimization: We ensure that the hardware and software configurations are optimized for deep learning tasks. For example, we use GPU acceleration for training and inference whenever available. We ensure that the CUDA toolkit and cuDNN library are updated to the latest versions compatible with our TensorFlow and PyTorch installations. Initially we also planned to use TPUs, but the support for TPU in Pytorch is not adequate and neither Pytorch nor TensorFlow has necessary support for TPU based inference of quantized models.

Data Pipeline Optimization: As we explored the time and memory profile of the models in the initial runs, we identified that most of the time was spent on CPU intensive tasks and was the reason for bottlenecking the model performance. So we tune the workers and pin_memory parameters in Pytorch dataloader and the prefetch and tf.Autotune functionality in TensorFlow to optimize data pipeline bottlenecks. However, even after this tuning, around 70% time was still consumed on the CPU operations which could not be reduced further in the given time frame.

Model Optimization: Post-training quantization and quantization-aware training are the primary model optimization strategies that we utilize. Both these techniques have the potential to greatly reduce the model size and speed up inference, with some trade-off in model accuracy.

In post-training quantization, the weights of the already trained model are quantized from floating-point to lower precision, such as int8. This significantly reduces the model size and accelerates model inference, particularly on hardware that supports lower precision arithmetic. Pytorch provides the option to fuse layers before quantization to reduce number of layers and quantization error propagation. TensoFlow directly converts the model into TF-lite format (bytes) and does not provide any control over layer collapse.

Quantization-aware training goes a step further by simulating the effects of quantization during training. This allows the model to learn to compensate for the loss in precision, usually resulting in a quantized model with higher accuracy compared to post-training quantization. Pytorch requires manual addition of quantization stubs at input and output of the model for QAT, however TensorFlow just requires us to annotate the model using a builtin function. We can annotate whole model or specific layers with corresponding accuracy-inference time tradeoffs. We train the quantization aware models for 5 epochs. While the baseline models achieved saturation during the training, we observe that while QAT routine, both the models tend to improve.

V. Experimental Results

In this section, we present the experimental results of our study. The performance of baseline, post-training quantization, and quantization-aware training models in both TensorFlow and PyTorch frameworks was evaluated. We provide comprehensive analysis based on model accuracy, model size, training and inference time, memory consumption, and inference latency.

Model Accuracy: The accuracy of the models was evaluated on the CIFAR10 test set. The results are present in Fig. 3. Due to initialization variations, Pytorch and TensorFlow had wildly different accuracy scores. However, it was observed that the trend of accuracy scores for baseline, post training quantization (PTQ) and quantization aware training (QAT) models were similar across both the frameworks. The models subjected to post-training quantization exhibited a slight drop in accuracy, attributed to the loss in precision due to quantization. However, the models trained with quantization-aware training managed to achieve higher accuracy than their respective baseline models, validating the effectiveness of this technique in compensating for quantization losses. This is quite surprising since the baseline models reached saturation during their training.

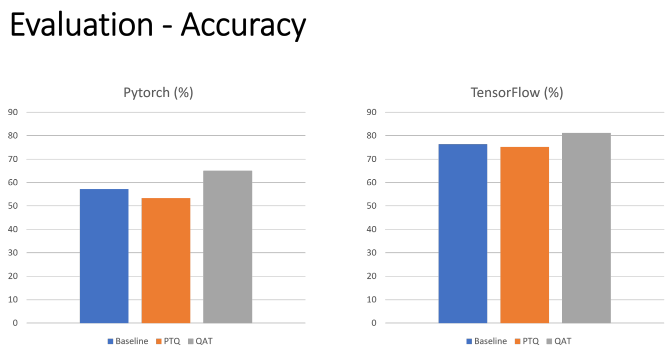


Fig. 3. The overall workflow of the project.

Model Size: Post-training quantization and quantization-aware training significantly reduced the model size in both the frameworks. The comparison of model size before and after quantization are presented in Table I This can be crucial in scenarios where storage space is a concern, such as deploying models on edge devices. While we tried to make models consistent across both the frameworks, it can be noted that the size differs based on the parameters and internal storage implementations.

Training Time: The time taken for training and calibration was also logged for the models in both frameworks. The quantized models generally exhibited faster inference times due to the reduced precision computations. Table II represents the time required to train the quantized models. As it can be seen, due to addition of quantization stubs and monitoring agents, the training times increase significantly for both the frameworks, with Pytorch being 95.3% and TensorFlow being 217.8% higher than their baseline training times.

TABLE I
Model size comparison of quantized models

Model Size	Baseline (MB)	Quantized (MB)	% reduction in model size
Pytorch	42.62	10.76	74.7
TensorFlow	42.61	11.31	73.4

TABLE II
Training time comparison for quantized models

Training Time	Baseline (s)	QAT (s)	% increase in time
Pytorch	16.80	32.82	95.3
TensorFlow	18.04	57.34	217.8

The calibration times for post training quantization are presented in Table III. As it can be seen, TensorFlow requires 1900s while Pytorch only requires 40s to calibrate the quantized model on the training data.

Device Time Consumption: The device specific time consumption during training and inference was profiled only for Pytorch, as TensorFlow does not support inference profiling. For 5 input samples, the baseline model on CPU takes 33ms out of which 70% of time is spent on conv layers. For quantized model, it takes 13ms out of which only 27% is spent on conv while 34% is spent on quantization agents. For GPU bound baseline model, we observe that out of 19ms spent, 1.5ms are spent on GPU while 17.5ms are spent on CPU. From these observations, it is clear that our models are very CPU bound. But still the Quantized models exhibited reduced time footprints, which could prove beneficial in resource-constrained environments.

Inference Latency: Lastly, the inference latency per sample was evaluated. The observations are presented in Fig. 4. It was observed that the quantized models, especially those trained with quantization-aware training, exhibited lower latencies, indicating their potential for real-time applications. However, it should be noted that TensorFlow does not allow quantized models to run natively on the devices. So, we need to use interpreter which adds significant overhead on inference time which needs to be factored.

In the end, we provide a comprehensive list of framework support issues related to general profiling and model optimization through quantization in Table IV.

VI. Conclusion

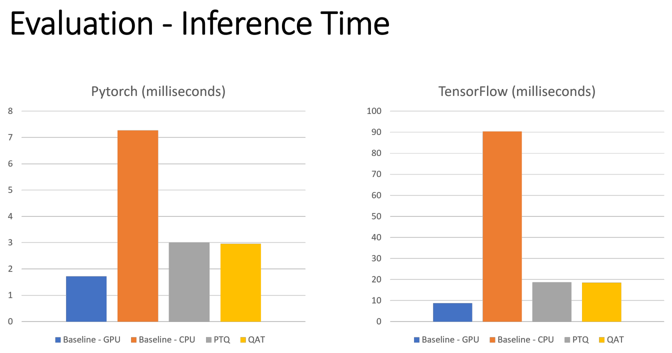
In this project, we experiment with the current quantization toolkit available with Pytorch and TensorFlow. Based

TABLE III
Calibration time comparison for post training quantization

	PTQ Calibration Time (s)
Pytorch	∼40
TensorFlow	∼1900

TABLE IV
MAJOR ISSUES DISCOVERED DURING THE PROJECT

Functionality	**Pytorch**	**Tensorflow**
Hardware Support	- No general TPU support - No quantization support on GPU	- Convoluted implementation for GPU/TPU
Profiler		- Profiler has conflict with TF model layers - No profiler support for inference
Quantization	- Manual layer fusion expected - Manual quantization agent setup - Only supported on CPU - Automated quantization in beta - Embedding type layers not supported	- Requires special layer annotation for unsupported layers (no documentation) - Auto quantization requires special interpreter - QAT has significant overhead (3x Pytorch) - Embedding type layers not supported
Analysis	- Quantization stubs do not allow further probing	- TF lite models are byte type, thus preventing access to model layers



Evaluation - Inference Time

Fig. 4. Inference latency comparison for quantized models.

on the quantitative metrics, it can be inferred that both the frameworks perform with marginal difference on quantization task. However, both the frameworks have very limited support and requires many modifications/cutbacks to work, as stated in the approach and evaluation sections.We have compiled a comprehensive list of issues related to quantization and framework feature conflicts and we believe it can serve as a roadmap for opensource community to work and improve ML optimization toolkits. Our study provides valuable insights into the performance of TensorFlow and PyTorch when handling quantized models. Our results can guide practitioners in making an informed choice of framework and quantization strategy based on their specific requirements.

ACKNOWLEDGMENT

I would like to thank Dr. Maghraoui and Dr. Dube to provide me with necessary tools and opportunity to work and explore the current research gaps in High Performance Machine Learning systems.

REFERENCES

[1] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems.

[2] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[3] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[4] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Adam, H. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[5] Abadi, M., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

[6] Paszke, A., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems.

[7] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition.

[8] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition.

[10] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, ”Attention is All You Need,” 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2017.

[12] M. J. Huiskes and M. S. Lew, ”The MIR Flickr Retrieval Evaluation,” in Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval (MIR’08), Vancouver, British Columbia, Canada, 2008.

[13] Cornia, M., Baraldi, L., Serra, G., & Cucchiara, R. ”Meshed-memory transformer for image captioning.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.

[14] Han, S., Pool, J., Tran, J., & Dally, W. ”Learning both weights and connections for efficient neural network.” Advances in neural information processing systems. 2015.

[15] Zhu, A., et al. ”Benchmarking and analyzing deep neural network training.” IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2018.