Optimizing Image Classifiers with Advanced LLM Quantization Techniques

Austin Brown
Duke University
Durham, NC 27708
arb153@duke.edu

Bharat Krishnan
Duke University
Durham, NC 27708
bpk12@duke.edu

Kushagra Ghosh
Duke University
Durham, NC 27708
kg260@duke.edu

Miles Yang
Duke University
Durham, NC 27708
mjy9@duke.edu

Abstract

This project explores the applicability of modern large language model (LLM) quantization techniques on image classification models. Quantization of these image classifiers is aimed at improving throughput and memory efficiency without incurring large performance trade-offs, and the specific use of LLM quantization techniques is aimed at analyzing the parallelism between transformers and convolutional neural networks (CNNs) and its practical capabilities. Quantization of image classifiers such as CNNs is important in practical applications such as the task of traffic and road sign detection in self-driving cars. As part of the several quantization methods implemented, this approach entails reimplementing state-of-the-art quantization techniques originally designed for pre-trained transformers (LLMs) so that they could be adapted to a convolutional neural network. By adapting these LLM quantization techniques and their underlying theoretical principles to image classifiers, this project can contribute novel insights into the impact and applicability of advanced quantization strategies on smaller models such as ResNet-18.

1 Introduction

This project aims to bridge the gap between quantization methods designed for large-scale transformer architectures and their effectiveness in smaller convolutional neural networks. Three post-training quantization methods were implemented as well as one quantization-aware training method to analyze the practicality of implementing LLM quantization methods in convolutional neural networks. The image classification task in this setting is the classification of different traffic and road signs, and the model is a ResNet-18 image classification model.

The LLM quantization techniques analyzed in this project include AWQ, SmoothQuant, and GPTQ. These served as the use case where the capability and computational overhead of vision-language models or large language models are unnecessary for certain image classification tasks. However, this project examines the applicability of theoretical insights from different modern LLM quantization techniques.

This efficiency is particularly useful in scenarios where fast inference is of extreme importance, such as in self-driving cars. Self-driving cars must detect different road signs to inform next decisions the car should make on the road. The car uses advanced computer vision methods to quickly and accurately recognize these objects in real time, and these quantization techniques can optimize classification by reducing model size and memory-to-GPU transfer costs.

Preprint. Under review.

1.1 Motivation for Adapting LLM Quantization

The mathematical operations underlying attention mechanisms and convolution share fundamental similarities in how they process information. Both operations can be viewed as computing similarity scores between inputs - convolution measures similarity between a kernel and local regions of the input, while attention computes similarity between queries and keys. This parallel becomes even more apparent when considering that convolution operations can be broken down into multiple 1×1 convolutions followed by shift and summation operations, similar to how attention uses projections of queries, keys, and values.

Given these architectural similarities, we hypothesized that quantization techniques developed for one architecture might be adaptable to the other. While attention mechanisms and convolution operations process information differently - attention allowing for dynamic, content-dependent interactions across the entire input while convolution uses fixed receptive fields and shared parameters - both face similar challenges when it comes to quantization, particularly in managing outliers that can dominate the quantization scale and lead to precision loss. This motivated our investigation into whether techniques originally designed for quantizing attention mechanisms could be effectively adapted for convolutional neural networks, despite their different scales and architectural characteristics.

2 Related Works

2.1 Quantization-Aware Training (QAT)

Quantization-Aware Training (QAT) integrates quantization effects into the training loop, enabling the model to learn weight representations robust to discretization. Unlike post-training quantization, which applies quantization after training, QAT introduces quantization during forward passes, allowing gradient-based optimization to adapt weights and activations to the quantized representation.

For weights $\mathbf{W} \in \mathbb{R}^d$ and activations $\mathbf{A} \in \mathbb{R}^d$, quantized to a lower bit-width, a quantization function $Q(\cdot)$ maps floating-point values to discrete levels. For uniform quantization with 2^b levels, using scale $\alpha > 0$ and zero-point $z \in \mathbb{Z}$, quantization and dequantization are defined as $Q(x) = \text{clamp}\Big(\left(\frac{x}{\alpha} + z\right), q_{\min}, q_{\max} \Big), \quad x_{\text{dequant}} = \alpha(Q(x) - z)$. During training, QAT applies $Q(x)$ in the forward pass but uses a straight-through estimator (STE) for the backward pass, approximating the gradient as $\frac{\partial L}{\partial x} \approx \frac{\partial L}{\partial Q(x)}$. This approach co-optimizes model and quantization parameters, adapting them to minimize quantization noise. As previous works [4][5] demonstrate, QAT can achieve near-floating-point accuracy even at lower bit-widths, making it a powerful technique for efficient model deployment.

2.2 Post-Training Quantization: Optimal Brain Compression Framework

The GPTQ (Post-Training Quantization) approach follows several ideas from the Optimal Brain Quanization method (Frantar et al., 2022) which quantizes each row independently; within each row, the method quantizes the "easier" weight first (weight with least additional error after quantization) and then adjust all remaining non-quantized weights to compensate for the loss in precision. The OBQ algorithm quantizes the weights that are least sensitive to the quantization error first and follows a greedy approach. However, the GPTQ algorithm quantizes weights in the same order for all rows of a matrix, which saves computation because the Hessian inverse calculation and update has to be done only once for each column instead of every time for each greedy update.

3 Methodology

3.1 Baseline: Full-Precision ResNet-18 with Transfer Learning

Training the baseline ResNet-18 model from scratch using only the available road sign images proved insufficient for the model to learn a generalized representation due to the limited size of the dataset. A transfer learning approach was adopted to address this, by leveraging a ResNet-18 model pre-trained on the ImageNet dataset. ImageNet, with its 21,841 classes and approximately 12 million training

samples, provides a robust foundation for feature extraction, making it an ideal source for initializing model weights.

Specifically, the pre-trained ResNet-18 model was used as a fixed feature extractor, with all weights fine-tuned for the target task of road sign classification. The model's output layer was replaced with a 29-node classifying linear layer to match the 29 classes in the dataset. During fine-tuning, the optimizer was configured to update all weights in the network, including the pre-trained weights from ImageNet. This approach, supported by theoretical insights from recent literature and lectures on transfer learning, provided a better initialization point than random weight assignment, improving model convergence and performance on the road sign classification task.

3.2 QAT: Quantization-Aware Training with fixed-point linear quantization of ResNet-18 with dynamic scaling

In this work, Quantization-Aware Training (QAT) was implemented to enhance the deployment efficiency of a ResNet-18 model fine-tuned for traffic sign classification. The process entailed customizing a QAT pipeline that integrates fixed-point linear quantization with dynamic scaling, enabling the model to adapt to quantization noise during training and maintain near-floating-point accuracy at deployment.

Model Design and Preparation A transfer-learning approach was employed by fine-tuning a pre-trained ResNet-18 model initialized with ImageNet weights. The model's fully connected layer was replaced to align with the classification task's 29 target classes. QAT was introduced by creating a custom quantized model class, `FineTunedResNet18Quantized`, which incorporated the following steps:

- **Layer Fusion:** Key layers, including convolution, batch normalization, and ReLU, were fused to improve inference efficiency and reduce quantization error. This fusion process was applied to the initial convolutional layer (`conv1`) and all layers within the model's residual blocks.
- **Quantization Configuration:** The model was configured with PyTorch's default QAT quantization backend (`fbgemm`) to leverage hardware acceleration during inference.

Training Pipeline The QAT pipeline consisted of fine-tuning the quantized ResNet-18 model using the following approach:

- **Criterion and Optimizer:** Cross-entropy loss was employed for classification, optimized with stochastic gradient descent (SGD) to ensure robust convergence.
- **Dynamic Scaling:** During training, weights and activations were quantized in the forward pass to simulate inference conditions. A straight-through estimator (STE) approximated gradients for quantized parameters during the backward pass, enabling the model to learn quantization-aware weight distributions.
- **Training Process:** The model was trained for 10 epochs with a batch size of 32. The training loop alternated between updating model parameters and validating performance to track progress and optimize hyperparameters dynamically.

Quantized Model Conversion Post-training, the model was converted into a quantized format for deployment. PyTorch's quantization conversion process was applied to transform the model's weights and activations into fixed-point representations. The resulting quantized model was saved to disk as `resnet18_finetuned_qat.pt`.

3.3 GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers, Applied to ResNet-18

GPTQ is a one-shot post training quantization method that uses "approximate second-order information" to make efficient and accurate predictions. The reference paper for the approach claims to have the first post-training method to quantize language models (hundreds of billions of parameters) to 3-4 bits a component while maintaining high accuracy. GPTQ performs layer-wise compression

quantization and minimizes the squared error of the full precision layer output $\mathbf{WX}$ with the quantized layer output $\hat{\mathbf{W}}\mathbf{X}$, where $\mathbf{X}$ is the layer input and $\mathbf{W}$ is the layer weights. This objective can be expressed by $f(\hat{W}) = \arg\min_{\hat{W}} \|\mathbf{WX} - \hat{\mathbf{W}}\mathbf{X}\|_2^2$. The GPTQ approach uses a Hessian matrix which can be obtained from the layer-wise quantization objective function:

$$\nabla_{\hat{W}} f(\hat{W}) = 2\hat{\mathbf{W}}\mathbf{X}\mathbf{X}^\top - 2\mathbf{W}\mathbf{X}\mathbf{X}^\top \implies H = \nabla^2_{\hat{W}} f(\hat{\mathbf{W}}) = 2\mathbf{X}\mathbf{X}^\top$$

The Hessian is used in the implementation of the GPTQ algorithm to examine the rate of change for the quantization error when updating the layer weights. Specifically, the inverse Hessian is used when updating the weights since it provides information about the error sensitivity, or the sensitivity of the weights to changes and how it affects the quantization error. The higher the inverse Hessian value is for a particular weight, the less sensitive the quantization error is to changes in the weight.

The reimplemented algorithm follows the procedure of first calculating the Cholesky decomposition of Hessian inverse. Then, for each batch i of columns in a layer, and for each column j in a batch of size B, the algorithm quantizes the weights in the column ($\mathbf{Q}_{:,j} \leftarrow \text{quant}(\mathbf{W}_{:,j})$), calculates the quantization error ($\mathbf{E}_{:,j-i} \leftarrow (\mathbf{W}_{:,j} - \mathbf{Q}_{:,j}) / \left[\mathbf{H}^{-1}\right]_{jj}$), and updates the remaining unquantized weights in the batch accordingly ($\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$). After processing the batch of columns, the algorithm updates the remaining unquantized weights in the layer based on the batch's errors before quantizing the next batch of columns in the layer ($\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$).

3.4 AWQ: Activation-Aware Weight Quantization for On-Device LLM Compression and Acceleration, Applied to ResNet-18

Activation-aware weight quantization (AWQ) is a post-training quantization technique developed for LLM quantization. This technique focuses on reducing quantization loss resulting from the decreased granularity of higher magnitude salient weights during regular quantization techniques, a prominent issue in LLMs due to the generally larger weight distributions on each layer, making salient weights disproportionately impactful in LLMs. Although convolutional neural networks generally may not have the issue of dependence on salient weights, given the parallelism between transformers and convolution, this application explores the practicality of applying this LLM principle to a convolutional neural network through AWQ.

Implementation There are 2 main processes done in this AWQ implementation: AWQ-specific pre-processing, and quantization.

1. **AWQ-specific pre-processing:** Intentional weight scaling preserves the saliency of higher-magnitude weights.
   (a) **Channel-wise scaling:** Weights are inversely scaled channel-wise by the activation magnitude in order to normalize the contribution of each channel.

   $$s = s_X^\alpha, \quad \alpha^* = \arg\min L(s_X^\alpha)$$

   In these equations, s is the channel-wise scaling factor, s_X^α is the channel-wise average magnitude of activations, and $L(S) = ||Q(W \cdot \text{diag}(s))(\text{diag}(s) - 1 \cdot X) - WX||$ measures the output difference after quantization. Activations are used instead of weights because they provide context-specific information about the importance of weights during inference. Activations are calibrated on a subset of the dataset, and the activation magnitude reflects the contribution of each weight to the output, making it a better indicator of which weights are salient for the model's performance. A grid search was performed to evaluate the optimal hyperparameter, which will be detailed further below.

   (b) **Upscaling of salient weights:** Salient weights are scaled up to prepare for quantization loss.

   $$Q(w \cdot s)(\frac{x}{s}) = \Delta' \cdot \text{Round}(\frac{w \cdot s}{\Delta'}) \cdot \frac{x}{s}$$

   Here, s is the optimal scaling factor for salient weights and is applied to the top 1% magnitude weights, which are defined as the salient weights. A grid search was

performed to evaluate the optimal hyperparameter, which will be detailed further below.

2. **Quantization:** All weights are quantized to the same precision.

$$Q(w) = \Delta \cdot \text{Round}(\frac{w}{\Delta}), \quad \Delta = \frac{\max(|w|)}{2^{N-1}}$$

This is the general quantization equation, where $Q(w)$ are the final quantized weights, Round$(\cdot)$ is the rounding function, N is the number of quantization bits, and Δ is the quantization scaling factor.

Results Figure 1 (Supplementary Materials) graph depicts the weight distribution of a CNN layer before and after AWQ quantization. Post-quantization, the majority of weights are concentrated into discrete bins (quantization levels), demonstrating the effectiveness of AWQ in compressing the weight representation. However, high-magnitude weights, identified as salient, are scaled prior to quantization, preserving their impact on model performance. This scaling is evident in the graph's extended upper and lower bounds compared to unscaled quantization, indicating that the outliers (salient weights) are maintained with greater precision while the remaining weights are tightly grouped into quantized buckets.

As mentioned before, a grid search was performed to find the optimal hyperparameter arguments used in the AWQ-specific pre-processing stage, effectively minimizing the quantization loss. Figure 2 (Supplementary Materials) below depict the grid search results, showing a peak with $\alpha = 0.2, s = 1.5$ showing potential for higher accuracy with perhaps a more complex AWQ implementation with mixed-precision weight storage or the more complex hyperparameter search.

3.5 SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models, applied to ResNet-18

SmoothQuant addresses a fundamental challenge in quantizing neural networks by managing activation outliers that make standard 8-bit quantization difficult. While model weights are relatively easy to quantize, activations often contain extreme outlier values that dominate the quantization scale, leading to poor precision for the majority of values. The technique introduces a mathematically equivalent transformation that "smooths out" these activation outliers by redistributing the quantization difficulty between weights and activations through a channel-wise scaling operation.

The core mechanism involves dividing input activations by a smoothing factor while multiplying corresponding weights by the same factor to maintain mathematical equivalence. This transformation enables effective 8-bit quantization for both weights and activations (W8A8) while preserving model accuracy without any retraining. The smoothing factor is calculated using a migration strength parameter α that controls how much quantization difficulty is transferred from activations to weights, with α=0.5 typically providing a good balance for most models.

In its adaptation for convolutional neural networks (CNNs), SmoothQuant's principles were applied by scaling kernel values with a smoothing factor to manage activation outliers in each convolutional layer. While the original technique was designed for large language models (LLMs), where systematic outliers emerge beyond 6.7B parameters, CNN adaptations faced different challenges due to more localized and feature-dependent outlier patterns. Despite these architectural differences, the principle of redistributing quantization difficulty through mathematically equivalent transformations proved effective.

The implementation enabled efficient INT8 quantization for both weights and activations in our compute-intensive convolution operations. By performing the smoothing transformation offline and fusing it into the model parameters, hardware efficiency was preserved without incurring additional runtime overhead. This approach leveraged hardware-accelerated INT8 matrix multiplication while preserving model accuracy.

One limitation of this adaptation is that the underlying outlier patterns differ fundamentally between CNNs and LLMs. In CNNs, outliers are closely tied to the hierarchical feature extraction process, whereas LLMs exhibit systematic outliers across attention layers. As a result, while SmoothQuant's mathematical framework delivered significant performance benefits in this context, the quantization challenges addressed in CNNs may be qualitatively distinct from those in the original LLM focus.

4 Evaluation

The ResNet-18 image classification model was quantized to perform the road sign detection task using the Traffic and Road Signs Image Dataset, an open-source dataset obtained from Roboflow. The dataset comprises 29 classes representing different traffic road signs and includes 10,000 samples (416x416 resolution). A train-validation-test split of 70:15:15 was applied.

The accuracies, PyTorch model file sizes, and CPU inference times were evaluated on the test split of the dataset. Table 1 summarizes the performance of the various quantization approaches.

Table 1: Model Performance Comparisons

Model Type	Accuracy (%)	Size (MB)	CPU Time (s)
Baseline (Full-Precision)	99.85	44.1	2.143
QAT	99.63	11.4	2.056
GPTQ	63.2	11.4	2.014
AWQ	75.5	11.4	1.964
SmoothQuant	99.87	42.67	0.512

5 Conclusion

The study results reveal a clear tradeoff between model size and throughput across different quantization techniques. AWQ, GPTQ, and QAT effectively reduced model size, achieving approximately a fourfold reduction in storage requirements. However, these methods did not yield a corresponding improvement in model throughput, as CPU inference time remained largely unaffected. In contrast, SmoothQuant significantly reduced model throughput but failed to achieve a reduction in model size, highlighting the challenge of simultaneously optimizing both metrics without compromising performance.

Accuracy outcomes varied considerably among the methods. AWQ and GPTQ demonstrated significant performance degradation, indicating limited suitability for convolutional neural networks (CNNs). This may be due to the design of these methods, particularly the scaling of high-magnitude weights, which may not be well-suited to the narrower weight distributions typically found in CNNs. Conversely, QAT and SmoothQuant maintained high accuracy, suggesting that these methods are more appropriate for CNN quantization.

These findings emphasize the inherent tradeoffs in model compression techniques. Achieving simultaneous reductions in both model size and throughput without sacrificing accuracy likely requires more sophisticated and tailored quantization strategies. The results indicate that while QAT and SmoothQuant provide promising solutions for CNN quantization, AWQ and GPTQ may require further refinement to be effectively applied in this domain.

References

[1] Frantar et al. (2023). “GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers.” In: ICLR International Conference on Learning Representations (ICLR) 2023.

[2] Lin et al., (2023). "AWQ: Activation-aware Weight Quantization for LLM" In: MLSys 2024

[3] Xiao et al., (2023). "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models" In: International Conference on Learning Representations 2023.

[4] Sze, Vivienne, et al. (2017) "Efficient processing of deep neural networks: A tutorial and survey." In: Proceedings of the IEEE105.12 : 2295-2329.

[5] Jacob, Benoit, et al. (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 (CVPR), pp. 2704-2713.

[6] Chaudhry (2024). "Traffic and Road Signs Dataset." In: Roboflow Universe. Roboflow.

6 Supplementary Materials

6.1 AWQ

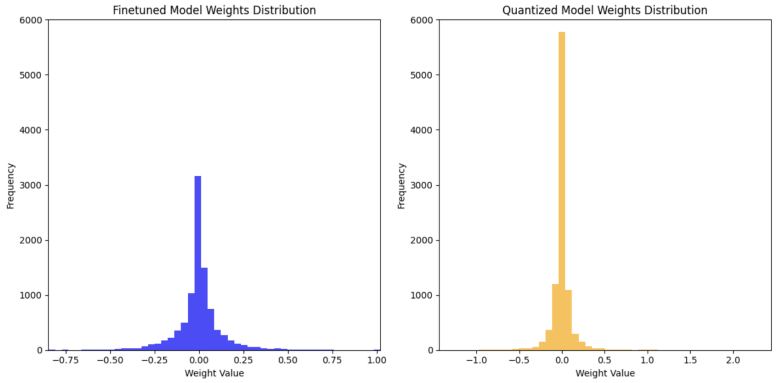

Figure 1: Example of layer weight distribution change.

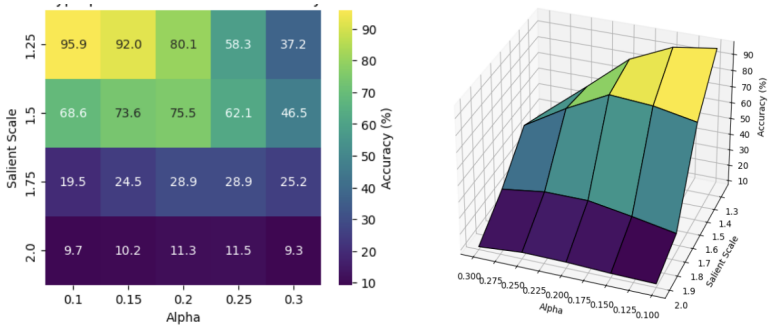

Figure 2: Grid search of hyperparameters for AWQ scaling.

6.2 GitHub Repository

The code that supports the project can be found in the following Github Repository with an open-source LICENSE: https://github.com/kushagraghosh/quantization