Quantization of Convolutional Neural Networks: A Practical Approach

Dwith Chenna

IEEE Senior Member

***Abstract*- Deep neural networks play a critical role in the remarkable developments in the field of computer vision. Convolutional Neural Networks (CNN), widely used in computer vision tasks, require substantial computation and memory resources, making it challenging for these models to run on resource-constrained devices. Quantization is an efficient way to reduce the compute and memory footprint of these models, making it possible to run them on edge devices. Many techniques have been studied to allow practical applications of quantization CNN models on mobile or edge devices. The practical implementation and adoption of these quantization techniques is heavily limited by approaches available in tools i.e. TFlite/Pytorch and underlying hardware. However, possible degradation in performance makes it challenging to achieve comparable performance to the original float-point model. In this paper, we will review different aspects of quantization, including assumptions, best practices, tools, and recipes to get the best results from quantization.**

***Keywords*: Quantization, Convolutional Neural Networks.**

1. Introduction

Convolutional Neural Networks (CNN) have evolved in model complexity and computation to achieve the best possible accuracy. This limits them from deployment on resource constrained hardwares like mobile, AR/VR, drones or any other edge devices. Successful deployment of CNN model on these hardware requires small model size, low compute and high power efficiency. Many research works focused on developing efficient model architectures, model compression and computational efficiency of CNNs while trying to maintain acceptable or minimal accuracy loss. MobileNet[1], SqueezeNet[2], ShuffleNet[3] and DenseNet[4] design efficient network architectures that try to use compute and memory efficient operations. Similarly quantizing weights/activations of a CNN from 32 bit float-point into lower bit precision, led to approaches like Binary Neural Networks[5], XNOR-net[6], Ternary weight networks[7] and many more [8-13]. These quantization methods result in degradation of model accuracy due to quantization error, it is often critical to balance latency vs accuracy tradeoff. The lower than 8 bit quantization schemes show significant accuracy degradation and may not generalize well for different models. Quantization approaches that use 8 bit integers have been the popular choice due to its efficient tradeoff between latency vs accuracy, many hardware platforms have native support of 8 bit integer arithmetic which makes it portable on many platforms. This has led to many popular open source tools i.e. TFlite, Pytorch and AIMET[27] supporting 8 bit quantization for weights and activations.

In this paper we will be focusing on the 8 bit integer quantization approach and various approaches available to achieve comparable accuracy to the float-point model. The rest of the paper is organized as follows Section 2 describes an overview of the two quantization schemes for representation of float-point distribution. Section 3 presents techniques to quantize CNN models and available tools for the implementation. Section 4 discusses the best Network Architecture choices to enable inference optimization. Sections 5-7 gives details of integer quantization, evaluation and analysis. Sections 8-9 describe alternative approaches for improving quantized accuracy and best practices.

2. Quantization Scheme

In this section we describe the mathematical definition of the quantization scheme which allows efficient implementation of integer arithmetic operations on the quantized values. The mapping for real numbers r to quantized integers q as equation (1), where S and Z are scale and zero point quantization parameters. For 8 bit quantization, q is quantized 8 bit integer, scale is usually a float-point value that is represented using fixed point representation[], zero-point is of the same type as quantized value. Special constraint on the zero-point to ensure that real zero values are quantized without any error. The quantized values can be mapped to real values as equation (2)

$$q = round\ (r/S + Z) \qquad (1)$$

$$r = S(q - Z) \qquad (2)$$

In case of float-point distribution, the quantization parameters can be S and Z can be calculated based on the distribution

$$S = (r\,max - r\,min)/(q\,max - q\,min) \qquad (3)$$

$$Z = round\,(q\,max - r\,max/S) \qquad (4)$$

These quantization schemes are broadly classified into two types i) Symmetric quantization and ii) Asymmetric quantization, depending on the nature of the float-point distribution to be quantized.

2.1 Symmetric Quantization

This implementation assumes the quantization is symmetric around zero as shown in Fig. 1, which allows the quantization parameter zero-point Z = 0. This allows for efficient implementation of integer arithmetic by eliminating the handling of zero-point.

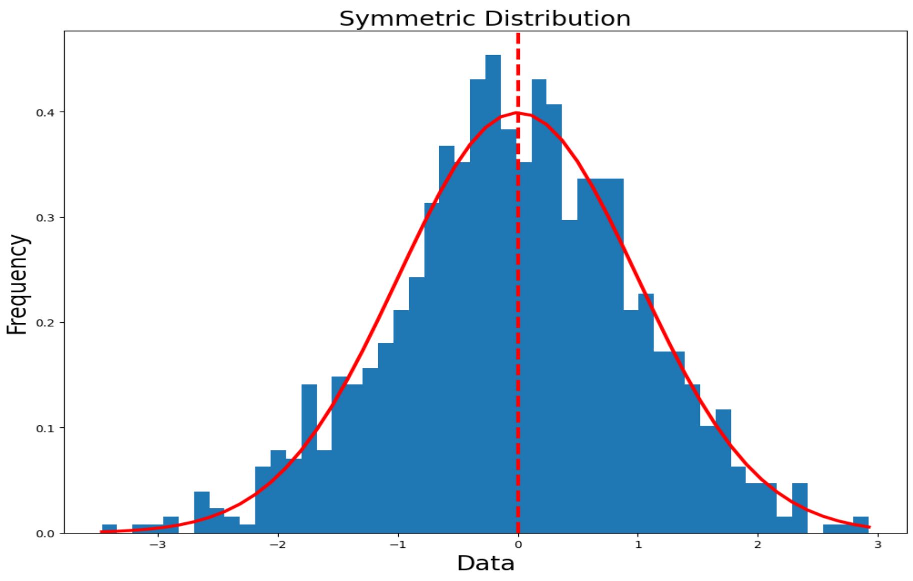

Fig 1. Symmetric distribution with zero point = 0

2.2 Asymmetric Quantization

Asymmetric quantization allows efficient utilization of bit width by account for the distribution for scale and shift. This allows trade off computational complexity to achieve better reduction of quantization error.

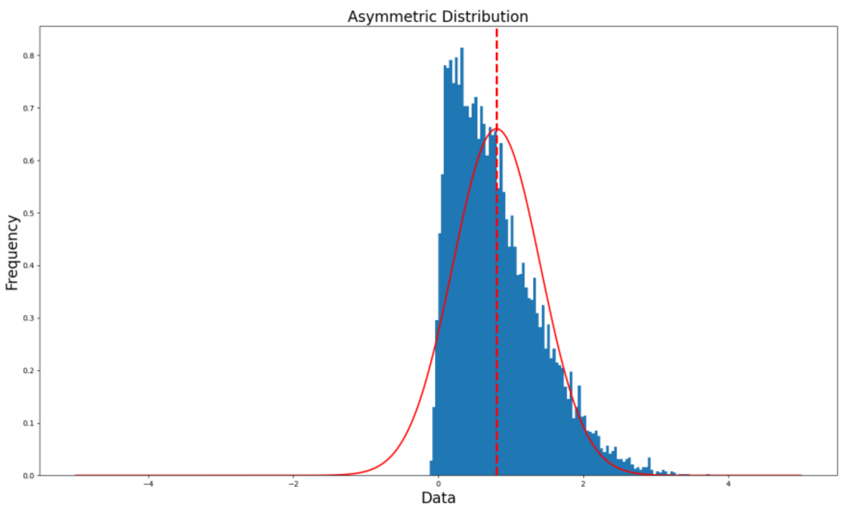

Fig 2. Asymmetric distribution with non-zero zero point

The choice of the quantization scheme Symmetric vs Asymmetric depends on the nature of the distribution, needing to trade off between accuracy and performance benefits. Fig 3. Shows how asymmetric quantization can be used to handle shifts in the data distribution, while symmetric distribution might lead to excessive quantization noise due to it inability to quantization resolution effectively

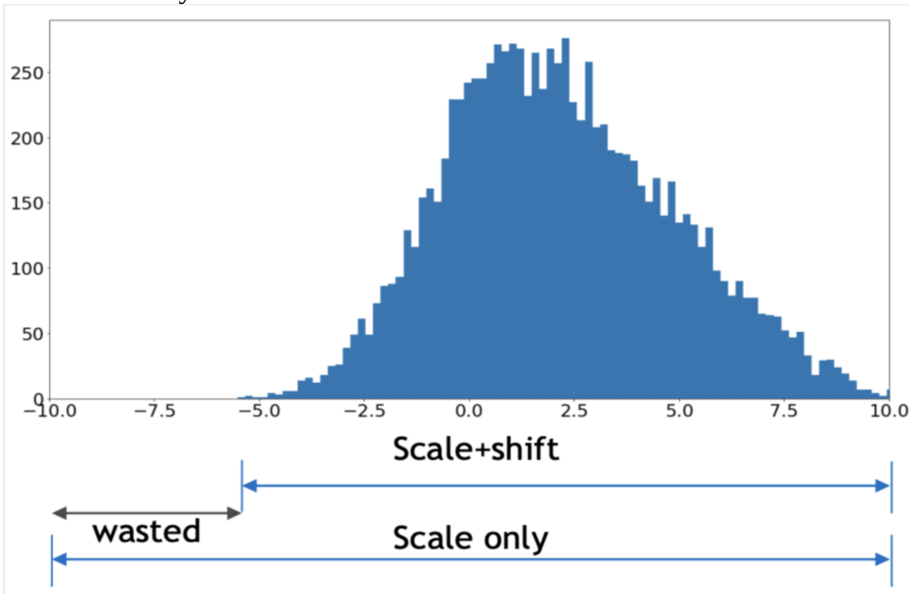

Fig 3. Shows scale and shift effects affected by symmetric vs Asymmetric quantization

3. Convolutional Neural Networks (CNN) quantization

In the case of CNN, which is trained using float-point arithmetic, we need to quantize weights and activations to efficiently store and compute them on embedded devices. Tensors are fundamental building blocks to represent multidimensional data for weight and activation of the network. In the next section we will discuss different design choices for quantizing these tensors

3.1 Quantization Granularity

When trying to quantize the float-point tensor there are various levels of quantization granularity you can choose to best suit the distribution. The most popular approach is to use i) per tensor, where the quantization parameters are computed for the entire tensor and ii) per channel quantization, where the quantization parameters are calculated per channel. The weights of the CNNs are usually symmetric [14], as shown by the empirical data and each layer is convolved with many different convolutional filters, which can have different ranges making it suitable for symmetric per channel quantization. Activation quantization needed to handle a wider variety of activation distribution is usually not centered around zero, due to activations like ReLU, which makes it suitable for asymmetric per tensor quantization. Batch normalization helps regularize the activation distribution, making them more amenable to quantization.

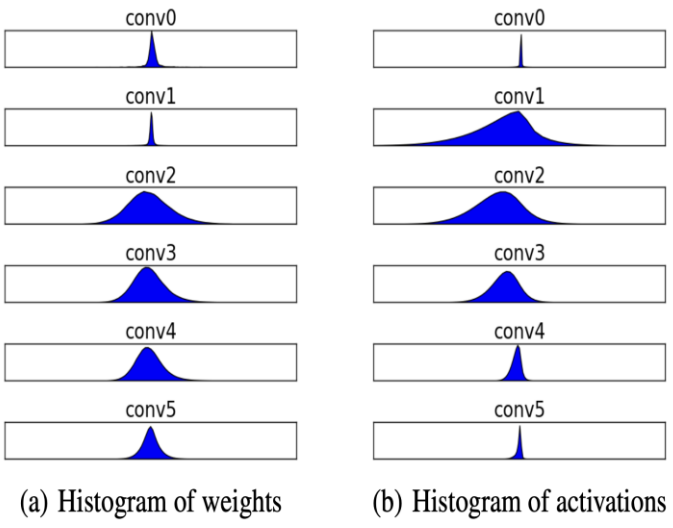

Fig 4. Histogram distribution of weights and activations [14]

3.2 Types of Quantization

The quantization techniques used for CNN are broadly classified as i) Post Training Quantization (PTQ) and ii) Quantization Aware Training (QAT). CNN parameters need to be adjusted to maintain the accuracy performance after quantization. The process of retraining the model to account for quantization is called Quantization Aware Training (QAT) or without retraining through Post Training Quantization (PTQ). A high level comparison of two approaches[] is shown in Fig 5.

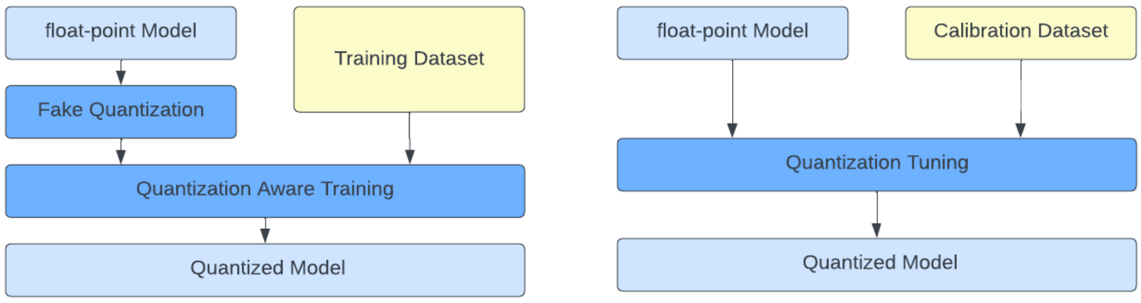

Fig 5. Comparison of Quantization Aware Training (QAT) on the left and Post Training Quantization (PTQ) on the right

3.3 Post Training Quantization (PTQ)

PTQ is a simple yet efficient technique that uses a fully trained float-point model along with a small calibration dataset for quantization. Unlike weights, quantization of activation can be difficult as the distribution is unknown, we use calibration dataset to collect the statistics and compute quantization parameters for the network. We will explore different challenges in achieving better accuracy through PTQ in Sections 5-7.

3.4 Quantization Aware Training (QAT)

Quantization Aware Training (QAT) tries to emulate the quantization of activation in order to allow the network to learn and achieve improvement performance for quantized results. This is achieved through introducing fake quantization nodes as discussed in Section 8. This approach has shown to recover the lost accuracy due to quantization, however the unavailability of training data and compute resources to train the model should be limiting the practical implementation of this approach.

3.5 Quantization Tools

The ultimate choice of quantization scheme depends on the available tools. TFlite and Pytorch quantizations are the most popular tools used for quantization of CNNs, both support 8 bit integer quantization. TFlite uses symmetric per channel quantization for weights and Asymmetrics quantization for activation, with optional support for 16 bit integer activation. Pytorch also supports 8 bit integer quantization as an experimental feature, allowing the user to choose the quantization scheme and granularity. In this study, we will be using TFlite quantization due to its ease of use and quantization analysis tools. These tools also allow mixed precision quantization and lawyer wise quantization error analysis tools. Before diving into the details, we will be discussing the effects of architecture choices on the overall quantization performance.

4. Network Architecture

Network Architecture Search (NAS) research was focused on development of model architecture, which are primarily trying to reduce the overall compute and size of the model. These models prioritize using depthwise convolutions to reduce the compute by order of magnitude. NAS based models like MobileNet/EfficientNet are deeper and leaner, making them suitable for cache based systems. Unlike most of the models trained for accuracy, which need to be bulked up to achieve higher accuracy, these models are easier targets for quantization due to redundancy within the network. However, models trained for efficiency like Mobilenet/EfficientNet are relatively difficult to quantize and tend to have significant degradation in accuracy. NAS can be optimized to select quantization aware networks and are better suited for the hardware. Overall efficiency of the network architecture depends on the quantization scheme, optimization tools and underlying hardware. NAS models are effective at finding architectures that are best suited for quantization and quantization aware network architecture to improve the robustness of the model. Quantization is expected to improve the inference latency, these improvements from quantization are hardware dependent with multiple factors such as on device memory, bandwidth, compute capacity affecting the quantization speed up. Hence, hardware-aware quantization will achieve the best overall performance.

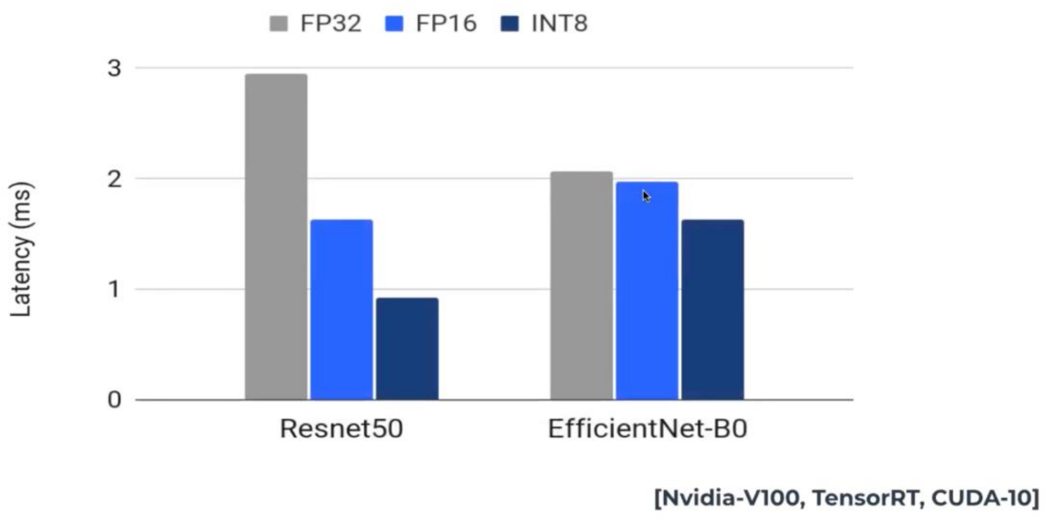

Fig 6. Comparison of Resnet50 and EfficientNet-B0 quantized results on Nvidia-V100 [25]

Activation functions need a special mention here, ReLU activation out performs other activation functions ReLU6, Sigmoid to name a few for precision requirement and quantization support. The effectiveness of different action functions for quantization of the network, based on the quantization noise using SQNR was used as a metric[]. ReLU showed to have much better results compared to ReLU6, even though ReLU6 reduces the quantization range, the signal power is reduced by clipping activations which affects the signal distribution with distortion making it less quantization friendly. Sigmoid activations are usually implemented using LUTs tend to have higher quantization error and significant performance limitations for hardware implementation.

5. **Post Training Quantization**

Post Training Quantization (PTQ) is the most popular approach for quantization as it is not resource intensive and only needs a small representative dataset to achieve acceptable results. PTQ is applied on pre-trained models, especially for cases where data is unavailable (privacy reasons) or limited availability of the data. Recent studies have shown that PTQ without any calibration data is possible through synthetic data[16] or by using statistical knowledge of the network architecture[17]. Since it does not need any retraining PTQ quantization is not resource intensive but may result in significant accuracy degradation[18]. PTQ can be applied where the data is limited or unlabeled, this often comes at the cost of lower accuracy especially for low-precision quantization.

5.1 Layer Fusion

Before quantization most of the inference optimization enables layer fusion for convolution + batch normalization + ReLU activations. It is advisable to enable this fusion before quantization, as it reduces the number of potential quantization error sources. Models with batch normalization usually have it as a separate layer, it is possible to fold their operations into previous convolution layer weights and biases for inference efficiency, also referred to as batchnorm constant folding in equations 5-7.

$$BatchNorm(w) = \gamma\left(\frac{w-\mu}{\sqrt{\sigma^2+\epsilon}}\right) + b \qquad (5)$$

$$w_{fold} = \frac{\gamma w}{\sqrt{\sigma^2+\epsilon}} \qquad (6)$$

$$b_{fold} = b - \gamma\left(\frac{\mu}{\sqrt{\sigma^2+\epsilon}}\right) \qquad (7)$$

Where γ is the batch normalization’s scale parameter, σ is the estimate of the variance of convolution results across the batch, and ε is just a small constant for numerical stability. After folding, the batch-normalized convolutional layer reduces to the simple convolutional layer depicted with the folded weights w_{fold} and the corresponding folded biases. Similarly activation functions ReLU/ReLU6 are also integrated to reduce data movement and improve efficiency of inference.

5.2 Calibration Dataset

Calibration dataset plays a critical role in determining the quantization parameters used for PTQ, these can be randomly selected unlabeled data representative of actual distribution expected during inference time. Other techniques like SelectQ[], base these selection on the statistics of the activations distribution for optimal and consistent performance with small dataset as small as 10-100 images. The upper limit on deciding the calibration dataset size is to ensure it is diverse enough to represent the actual inference inputs in order to avoid introducing bias error into the model. Research

[], shows that calibration dataset as big as few 100~1000 shuffled images generalize the classification accuracy of a network trained in ImageNet dataset which contains ~1M images.

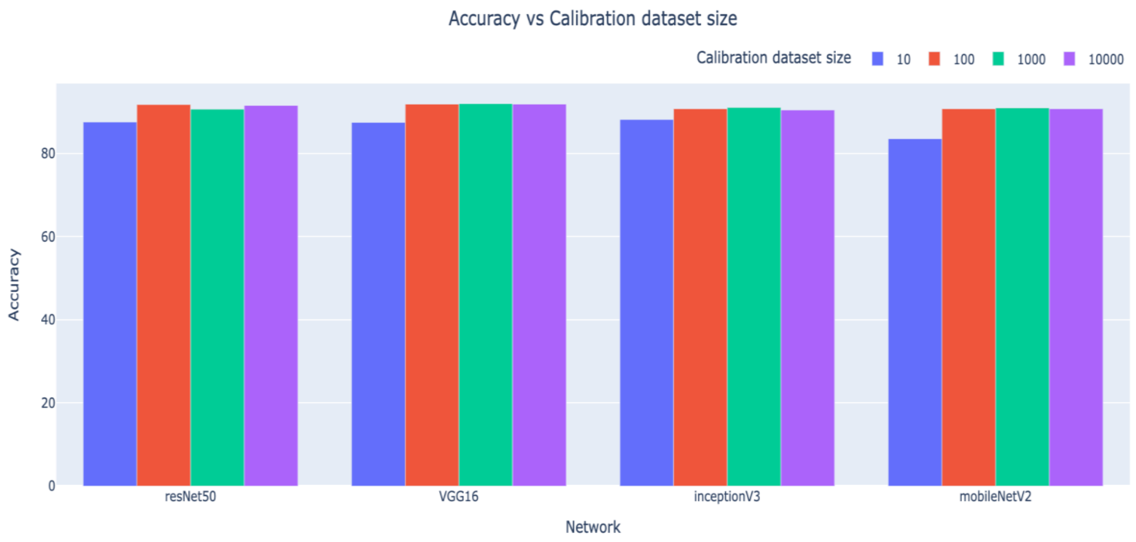

Fig 7. Effects of calibration dataset size on quantized model accuracy

6. Quantization Evaluation

Accuracy of the quantized model depends heavily on the network. The quantization scheme used for the model depends on the network architecture. Fig. is an overview of how different quantization schemes work with popular network architectures [15]. Bulkier models ResNet50 seems to be amenable to similar quantization schemes that provide both performance and accuracy, however they inherently have a lot more compute. The more compute efficient model Mobilenet needs more complex quantization schemes that allow for lower quantization error at the cost of performance. So, it is critical to choose an architecture that is the best trade off between quantization accuracy and overall performance.

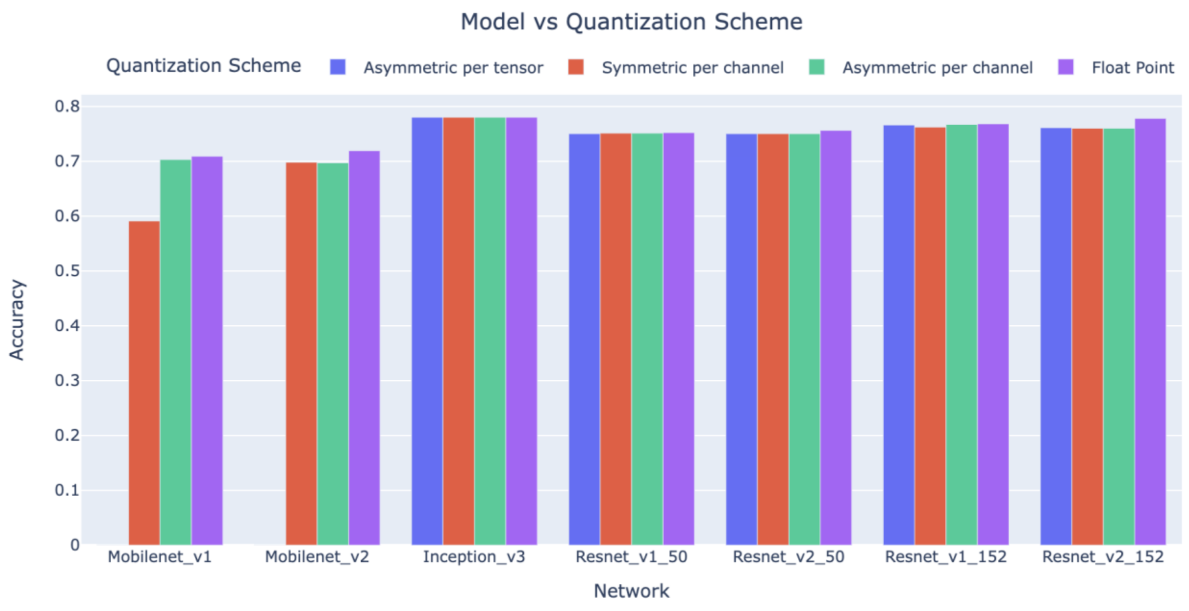

Fig 8. Evaluation of quantization model accuracy for different quantization schemes

Network	Asymmetric, per-layer	Symmetric , per-channel	Asymmetric, per-channel	Activation Only	Floating Point
Mobilenet-v1_1_224	0.001	0.591	0.703	0.708	0.709
Mobilenet-v2_1_224	0.001	0.698	0.697	0.7	0.719
Nasnet-Mobile	0.722	0.721	0.74	0.74	0.74
Mobilenet-v2_1.4_224	0.004	0.74	0.74	0.742	0.749
Inception-v3	0.78	0.78	0.78	0.78	0.78
Resnet-v1_50	0.75	0.751	0.751	0.751	0.752
Resnet-v2_50	0.75	0.75	0.75	0.75	0.756
Resnet-v1_152	0.766	0.762	0.767	0.761	0.768
Resnet-v2_152	0.761	0.76	0.76	0.76	0.778

Table 1. Evaluation of quantization model accuracy for different quantization schemes [14]

7. Quantization Analysis

Quantization of float-point models into 8-bit fixed point models, introduces quantization noise in the weights and activation, which leads to degradation in model performance. Common pitfalls include passing quantized input/output to the fully quantized model. This performance degradation might lead to minor degradation in accuracy or complete failure in model prediction.
As illustrated in the Fig, even a small change in the weights can lead to significant increase in loss.

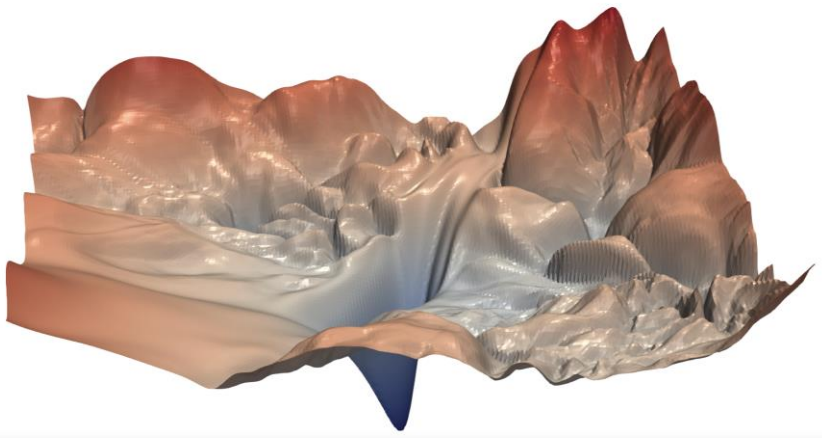

Fig 9. The loss surfaces of ResNet-56 with/without skip connections by Hao Li et al [26]

As shown in Fig 10. various sources of quantization error in a quantized implementation of convolution operation, that can lead to degradation in accuracy of the model. The bit precision used for different values depends on precision requirement i.e. bias is quantized to 32bit as the networks are more sensitive to quantization error in bias which can contribute directly to the output. These constant bias values don't contribute much to the overall size of the network. Similarly the accumulators used for storing these results can use 32 bit accumulators to avoid any potential overall or clipping. In the following section we discuss a few tools to debug these error, which vary in scope from network level to single operation level.

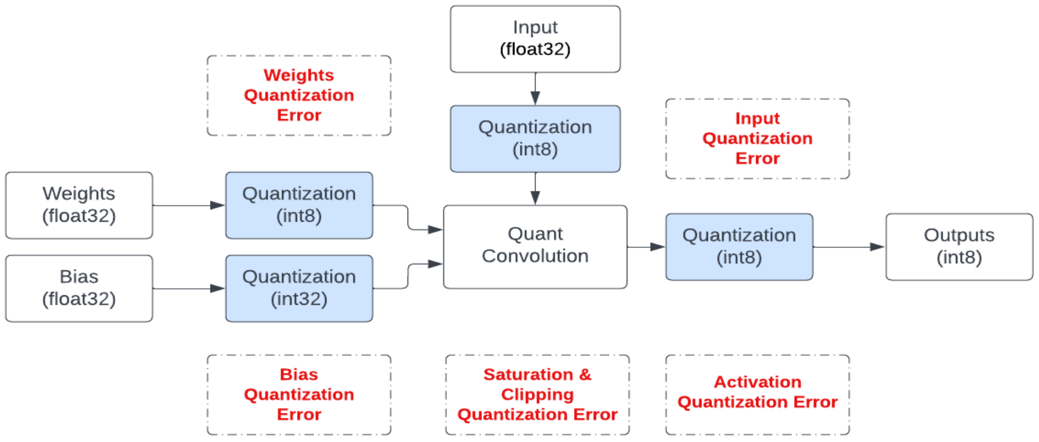

Fig 10. Shows various sources of quantization error during fixed-point convolution operations.

7.1 Quantization Error Analysis

In section, we will try to investigate when quantization fails through quantization error analysis, when trying to quantize a model we should ensure that all the layers have quantization support or atleast have float-point fall back if needed. In many cases the quantization error can be attributed to few problematic layers, many popular tools for quantization TFlite/Pytorch provide tools for layerwise quantization error analysis that enable users to identify problematic layers and fix them as shown in Fig 10.

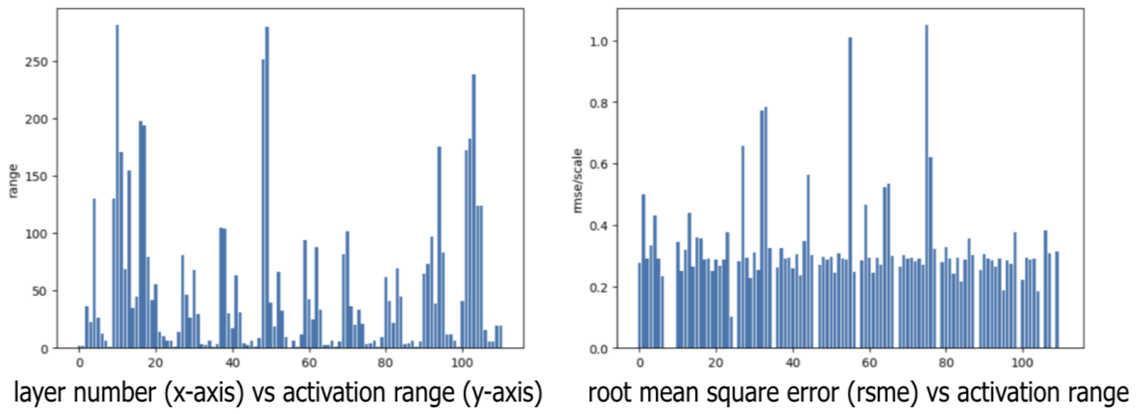

Fig 10. Analysis of layerwise root mean square error (rsme) for float-point vs quantized activations using tflite

7.2 Weight Equalization

Large difference in weight values for different output channels: more quantization error. Asymmetric/Symmetric per channel quantization helps handle better diversity on the weight distribution. Weight equalization[17] is an effective way to reduce the variance of weight distribution across channels to make them more robust to quantization. This is evident from the fact that many modern compute efficient networks like MobileNet/EfficientNet need per tensor quantization[], which means the same quantization parameters cannot be used to represent such high variation in weights across channels. An intuitive approach to improve the model quantization performance to rescale the weights for each output channel such that their ranges are more similar. This allows us to increase the range of values that are too low and reduce the range of values that are too high, thus reducing the quantization error. This can be achieved by exploiting scale equivariance across layers with point-wise linear activation functions. This allows us to change the weight distribution, without affecting the output of the model [].

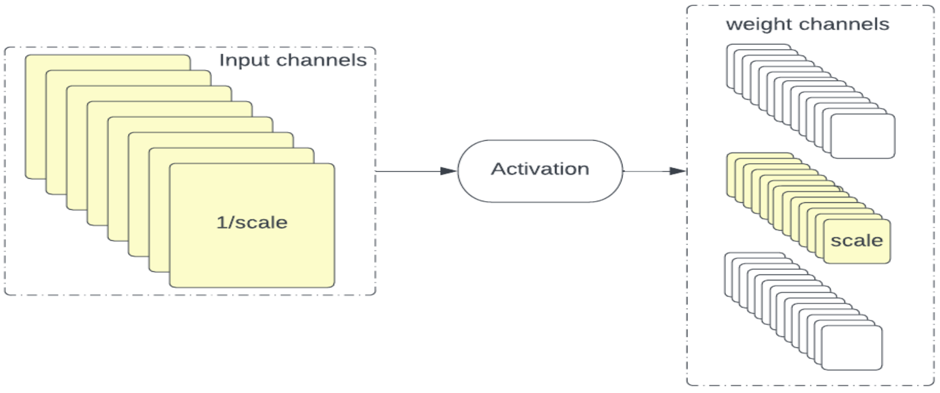

Fig 11. Illustration of scaling channels for cross-layer weight equalization

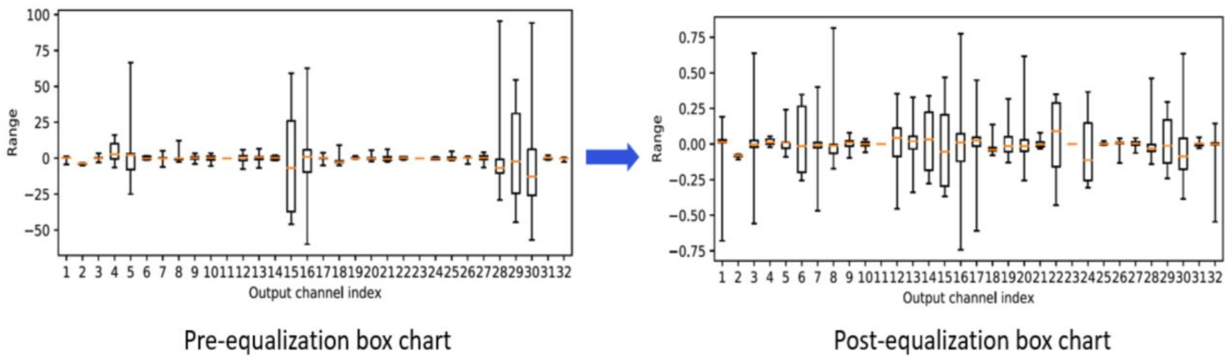

Fig 12. Visualization showing pre/post-equalization weights [27]

7.3 Mixed Precision

Among many ways to mitigate the problem with quantization error, it might seem a straightforward approach to use large bit width by increasing the precision to 16-bit or FP16 from 8-bit, could help recover some of the lost accuracy. Selective quantization allows mixed precision inference using different 8-bit/16-bit integers or FP16/FP32 for debugging. Various studies show how different bit width for quantizing weights/activations effects [], the overall performance of the networks.

7.4 Visualization

Visualization of the weights/activations allows us to estimate their distribution effects of quantization. These could be powerful tools to give user intuition into the effects of quantization on the network and possible artifacts of quantization error.

7.5 Min/Max Tuning

Quantization parameters are determined based on the min/max ranges of the distribution as discussed in Section 2. Min/Max tuning is used to eliminate outliers of weights/activation that cause all other weights/activations to be less precise. These ranges are determined differently for weights/activations. Weights are fixed for a trained model, this enables substantial opportunities for efficient quantization. Activation distribution depends on the input to the network, to estimate these ranges we can either use the range stored during training or through a calibration dataset for quantization. We will briefly discuss different methods used to estimate the activation ranges for quantization. Activation quantization parameter selection is challenging as discussed earlier, even with a representative calibration dataset. The range of the activation helps decide the clipping range from actual distribution, and helps choose trade off between range and quantization error. There are many approached to select the optimal

Mean/Standard Deviation

The mean μ and standard deviation σ of the activations are used to decide the min/max clipping. This is best suited for a distribution that can be approximated to gaussian and centered around the mean.

$min/max = \mu$ +/- 3σ

Histogram

The distribution histogram based methods used percentile % to select the range to be quantized. It is usually large enough to be representative of the distribution and avoid outliers that can significantly affect the quantization accuracy.

Set the range to a percentile of the distribution of absolute values seen during calibration [21]. For example, 99% calibration would clip 1% of the largest magnitude values. This is an efficient way to eliminate outliers that could adversely affect the quantization accuracy.

Moving Average
The default implementation of tensorflow [28], uses moving average to update the min/max values for each iteration of the calibration dataset.

Entropy
Min/Max values selected to minimize the entropy of the distribution. It uses KL divergence to minimize information loss between the original floating-point values and values that could be represented by the quantized format. This is the default method used by TensorRT []. It is particularly useful when not all values to be quantized in the tensor are not equally important.

8. **Quantization Aware Training**

Quantization changes to the trained model may move the model away from point convergence as discussed in quantization error analysis, which was trained in float-point precision as discussed in Section 5.1. QAT will address this issue by converging the model to a different and potentially better point for the quantized model. QAT tries to emulate quantized inference during training, while the actual training still happens in float-point. The quantization is modeled during training by inserting fake quantization nodes on both weights and activations. Back propagation is performed on float-point, as accumulating the gradients in quantized precision can result in diminishing gradients or high error especially in low-precision [22]. QAT has been shown to work despite the approximation for quantization and back propagation, closing the gap between quantized accuracy results to less than 1% of float-point in many models for classification. The QAT is computationally expensive as it needs retraining the model for several epochs to recover accuracy especially for low bit precision. This is a worthwhile investment retraining when possible for applications where accuracy is of utmost importance. It also allows for simpler quantization schemes which have more efficient implementations. During QAT, it is also possible to collect statistical data for quantization parameter selection, which can further improve PTQ accuracy.

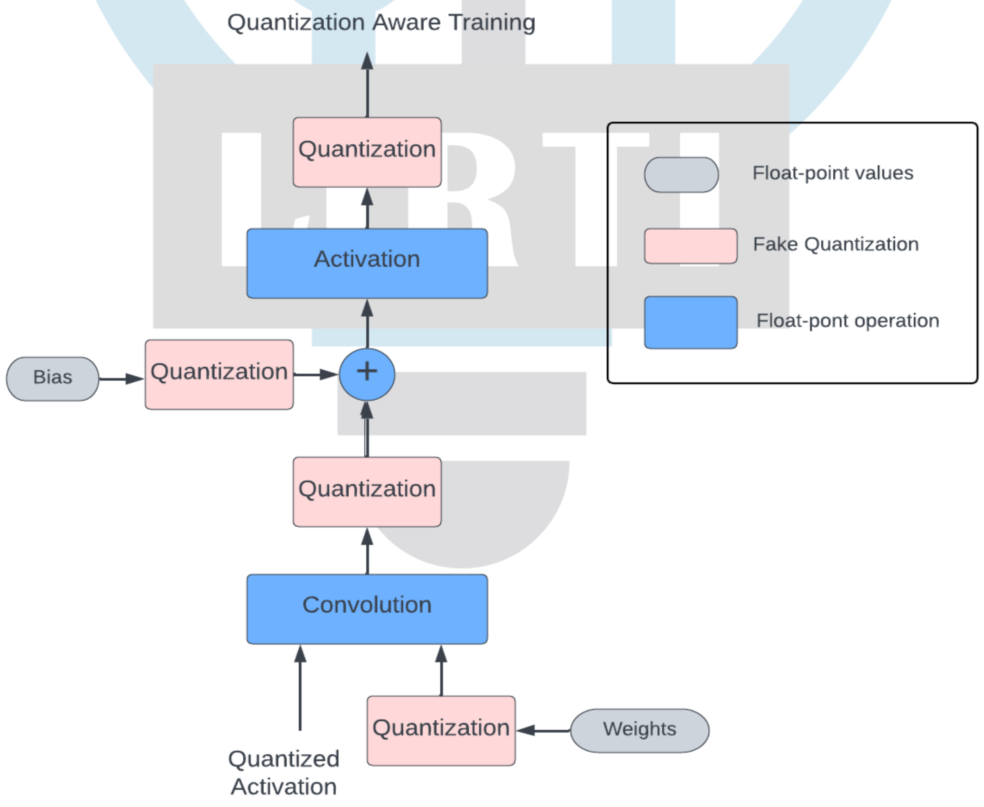

Fig x. Shows simulation of quantization during QAT

9. Best Practices

In this section we will try to summarize best practices to have efficient quantized model implementation on device

Model Selection

Efficiency of the quantization is a function of the model size and its architecture. Large models are more tolerant of quantization error. NAS for efficient architecture for quantization and hardware aware model optimization provide the best performance on device.

Quantization tools

Availability of quantization tools limits the quantization schemes and analysis tools for model quantization. TFlite and Pytorch quantization tools allow for different quantization schemes, analysis tools and QAT, to improve and fix quantized model accuracy.

Calibration Dataset

Quantization is essentially a fine tuning process, where the calibration dataset needs to be representative of the inference time inputs. The statistical data from around ~(100-1000) samples, seem to be enough to model the distribution of a large dataset like ImageNet. The upper limit on the size of the dataset is based on adding diversity to avoid introducing any bias into the quantized network.

Quantization Evaluation

The overall performance of the network is dependent on many factors related to network architecture and quantization. It is always advised to evaluate the quantized accuracy of the model using these different quantization schemes before diving deep into the quantization error analysis and debugging.

Quantization Analysis

There are various sources of quantization error that affect the overall performance of the network. In case of catastrophic failures, it is usually helpful to identify potentially problematic layers through mixed precision inference. Once identified the lost accuracy could potentially be recovered through techniques like weight equalization and min/max tuning.

Quantization Aware Training

Fine tuning an already trained model for quantization accuracy helps recover the lost quantization accuracy within < 1% of the float point model. This process can be significantly expensive in terms of computer and resource as you need to retrain the model for multiple epochs. It is worthwhile in applications where quantized accuracy is of utmost importance.

REFERENCES:

[1] A.G.Howard,M.Zhu,B.Chen,D.Kalenichenko,W.Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision appli- cations. *CoRR*, abs/1704.04861, 2017.
[2] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
[3] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.
[4] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Wein- berger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recogni- tion (CVPR)*, July 2017.
[5] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.
[6] M.Rastegari,V.Ordonez,J.Redmon,andA.Farhadi.Xnor- net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.
[7] F. Li, B. Zhang, and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.
[8] S. Han, H. Mao, and W. J. Dally. Deep compression: Com- pressing deep neural network with pruning, trained quantiza- tion and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.
[9] C. Leng, H. Li, S. Zhu, and R. Jin. Extremely low bit neural network: Squeeze the last bit out with admm. *arXiv preprint arXiv:1707.09870*, 2017.
[10] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462*, 2017.
[11] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremen- tal network quantization: Towards lossless cnns with low- precision weights. *arXiv preprint arXiv:1702.03044*, 2017.
[12] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.
[13] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.
[14] D. Lin, S. Talathi, and S. Annapureddy. Fixed point quantization of deep convolutional networks. In the

International Conference on Machine Learning, pages 2849–2858, 2016.
[15] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342, 2018.
[16] Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. (2020). Zeroq: A novel zero shot quantization framework.
[17] Nagel, M., van Baalen, M., Blankevoort, T., and Welling, M. (2019). Data-free quantization through weight equalization and bias correction.
[18] Banner, R., Nahshan, Y., Hoffer, E., and Soudry, D. (2019). Post-training 4-bit quantization of convolution networks for rapid-deployment.
[19] Szymon Migacz. Nvidia 8-bit inference width tensorrt. In GPU Technology Conference, 2017.
[20] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on cpus. In Deep Learning and Unsupervised Feature Learning Workshop, NIPS, 2011.
[21] Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V Arthur, Izzet B Yildiz, and Dharmendra S Modha. Discovering low-precision networks close to full-precision networks for efficient embedded inference. arXiv preprint arXiv:1809.04191, 2018.
[22] Matthieu Courbariaux, Yoshua Bengio, and JeanPierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. In Advances in neural information processing systems, pages 3123–3131, 2015.
[23] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. IEEE transactions on neural networks and learning systems, 29(11):5784–5789, 2018.
[24] Zhang, Zhao; Gao, Yangcheng; Fan, Jicong; Zhao, Zhongqiu; Yang, Yi; Yan, Shuicheng (2022): SelectQ: Calibration Data Selection for Post-Training Quantization. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.21456291.v2
[25] https://deci.ai/resources/achieve-fp32-accuracy-int8-inference-speed/
[26] H Li, Z Xu, G Taylor, C Studer, T Goldstein - Advances in neural information processing systems, 2018
[27] S. Siddegowda, M. Fournarakis, M. Nagel, T. Blankevoort, C. Patel, and A. Khobare. Neural network quantization with ai model efficiency toolkit (aimet). arXiv preprint arXiv:2201.08442, 2022.
[28] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In CVPR, 2018

IJRTI