A Dynamic Balance Quantization Method for YOLOv3

Yang Hua*, Lixin Yu, Xiao Meng, Zhiyong Qin and Boxu Chen
Beijing Microelectronic Technology Institute, Beijing, 100076, China
*E-mail: huay@mxtronics.com

Abstract. This paper describes a quantization method for pre-trained deep CNN models, and has achieved very good results in YOLOv3. The weights are quantized to int8 and the biases are quantized to int16, by contrasting the float inference result, the mAP loss is less than 0.5%. While running neural nets on hardware, 8-bit fixed-point quantization is the key to efficient inference. However, it is a very difficult task for running a very deep network on 8-bit hardware, often resulting in a significant decrease in accuracy or spending a lot of time on retraining the network. Our method uses dynamic fixed-point quantization and adds a small amount of bit-shifts to balance the accuracy of each layer in a YOLO network. At the same time, it further shows that this method can also be extended to other computer vision architectures and tasks, such as semantic segmentation and image classification.

1. Introduction
In the CNN era, researchers design deeper and deeper neural networks. While convolutional neural networks have the advantages of local connection, weight sharing, dimensionality reduction, they have been widely used in scientific research and industrial projects. These computer vision based neural networks provide humans with various conveniences, such as face recognition and autonomous driving.

In the past several years, convolution has been designed into various modules to help CNN achieve good accuracy. However, high accuracy usually relies on deep and wide architectures, which makes it challenging to deploy models to embedded devices. In applications, a variety of neural networks are deployed on the terminal to run to meet actual usage requirements. Due to the relatively limited computing power of the terminal, it is necessary to compress the pre-trained model to reduce the amount of calculation required during inference.

Many quantization methods have not been evaluated on a reasonable CNN architecture. The fully connected layers have the most parameters, and recent CNN structures have shown that removing these FC layers can also obtain high accuracy. For example, FC layers are not used in Inception and ResNet. Quantizing a model which is already efficient at trading off latency with accuracy is a more meaningful challenge.

YOLO stands for You Only Look Once. It's an object detector that uses features learned by a deep convolutional neural network to detect an object. YOLOv3 is a fully convolutional network using 75 convolutional layers, making it hard to run on hardware. In this paper, we use the official weight file for quantization.
Many quantization methods work well with YOLOv3-tiny, but no acceptable quantization method has been achieved for a fully YOLOv3 model.

2. Related work
Prior approaches have proposed many methods to trade off latency with accuracy. Jacob provides a quantization scheme that quantizes both weights and activations into 8-bit integers, while only quantizing a few parameters (bias vectors) into 32-bit integers [1]. This method relies on fine-tuning to minimize the accuracy loss caused by quantization on real model. Nagel proposed a data-free quantization method that does not rely on re-training, and accuracy can also be improved by optimizing plastic inference.[2]

Krishnamoorthi introduced a per-channel quantization scheme[3] in which the weight of the convolutional weight tensor is quantized for each output channel. Although it is effective to use bit-shifts in custom hardware, due to the correlation between the ratio of each channel and the offset value, there is a serious uncertainty in the output shift of a single-layer, and results in complex hardware.

Other methods improving quantized accuracy need to consider architecture changes or re-training, and these fine-tuning methods may even more complex than quantization methods[4,5,6,7]. At the same time, the quantization noise leads to additional hyper-parameters for optimization during the re-training process, which makes their practical application more complicated[8].

3. Our approach

3.1. Quantization bit width
For a complete tensor, we can specify a simple quantizer, which is defined as per-layer. To improve the accuracy, we can use different quantizer for each convolution kernel of the tensor. For example, a 4-dimensional tensor is a set of 3-dimensional convolution kernels. Each member in the set is responsible for generating a feature output. Per-channel quantization scheme has different scales and zeros for each convolution kernel, but it will complicate the inner product calculations in convolution operations and matrix multiplication operations. After performing per-layer int8 quantization on YOLOv3, the mAP loss is 5.54%, and there is almost no accuracy loss (<0.1%) while using per-channel quantization.
When two integer number matrices are multiplied, int8 multiplication and accumulation operations require 32 bits to store, that is, the actual working bit width during quantization inference needs 4 times the quantization bit width.

After using the 32-bit adder to get the result, the following 3 steps are required:
① Intercept int32 of the output value to int8;
② Update the output displacement scale;
③ Apply the activation function to the int8 result.

YOLOv3 uses successive 1x1 and 3x3 convolutional layers, the weights basically fitted the normal distribution. In the experiment, we found that using a 32-bit working bit width, when the quantization bit width is less than 18, the overflow generated is less than 3%, results the final accuracy impact is also less than 0.1%.

![Overflow rates and mAP loss of per-channel quantization with 16-bit bias and 32-bit working bit.](image)

Quantizing a 32-bit floating number to a n-bit fixed-point number, an irreversible deviation caused by rounding is bound to occur[9], and the biased error of the output changes the distribution of the input data of the next layer.

\[ \Delta = Q(W) - W \] \hspace{1cm} (1)

The expected output error due to quantization error is:

\[ E(\Delta) = E(\hat{y}) - E(y) \] \hspace{1cm} (2)

After outputting the results of each layer, subtracting this error can effectively improve the accuracy of the results. The output result of a certain layer can just estimate the mean value of the input of the next layer. When using the test set optimization API, the \( E(\Delta_l) \) of each layer can be counted, regaining about 0.3%~0.5% accuracy in YOLOv3.

### 3.2. Per-group scheme

In a pre-trained model, the data variance between the output channels of some layers is very large. In YOLOv3 DBLs use Add or Concat outputs as inputs. Using per-layer quantization scheme makes all channels with smaller values directly set to 0, resulting in a decrease with accuracy. Per-channel quantization scheme can solve this problem, each channel has its own independent scaling factor and offset value, but it will cause a lot of additional hardware overhead in hardware implementation.
For layers with large output data variance, especially the first several layers of the network, more bits are needed for quantization, and more hardware units are needed to be designed, by increasing the amount of computations needed.

![Figure 3. Per-channel quantized weights of the first Conv layer in pre-trained YOLOv3.](image)

In the boxplot, this tensor exhibits strong differences between channel range.

In these layers, the channels can be grouped to use scaling factors, it makes full use of the quantization range. After the multiplication completed, each tensor will be divided by the group scale then accumulated. So the output result remains unchanged. This scheme can greatly improve the accuracy of the output results of this layer, but requires additional hardware design. By analyzing the distribution of each layer floating results, box plots help to group the weights of each layer.

![Figure 4. Illustration of a fixed-point hardware with bit-shifts.](image)

In YOLOv3 convolution output results, a detection value greater than the certain threshold is a target detection hit. At this time, int16 results is required.

### 3.3. Block optimization

The activation function used in YOLOv3 is LeakyReLU, which satisfies \( f(s \times x) = s \times f(x) \), and all other piecewise linear activation functions are applicable [10]. Two adjacent convolution layers in the YOLO network:

\[
\begin{align*}
    x_1 &= f(W^{(1)} x_0 + b^{(1)}) \\
    x_2 &= f(W^{(2)} R(x_1) + b^{(2)})
\end{align*}
\]

Let:
We can get the new equation:

\[ x_2 = f \left( W^{(2)} x_1 + b^{(2)} \right) = f \left( W^{(2)} f( W^{(1)} x_0 + b^{(1)}) + b^{(2)} \right) \]  

(5)

Where S is a diagonal matrix, and the values on the diagonal correspond to the scaling factors used by each channel. When quantizing the network, the parameters need to be scaled. The large difference in scaling coefficients of different channels will lead to huge quantization errors. The scaling characteristics of LeakyReLU can be used to adjust the scaling coefficients of different channels between adjacent CBLs. Layers with multiple inputs or multiple outputs cannot use this scheme, which in YOLOv3 are Add and Concat layers.

In YOLOv3, there are two CBLs inside each res unit can be optimized, and six CBLs can be optimized before three regressors.

The quantization process of a YOLOv3 model is as follows:

1. Transform the pre-trained network to the ONNX model;
2. Complete the pre-processing of the test picture collection;
3. Select the number of digits that need to be quantified, and select part of the test pictures for network inference analysis;
4. Iterative optimization of scale among CBLs within the network unit;
5. Separate optimization for the larger error layer;
6. Select the test picture set for pure integer reasoning;
7. After processing the inference result, compare the mAP of the network before and after quantization;
8. Test whether the hardware simulation results are consistent.

### 4. Experiments

After the above-mentioned multiple levels of optimization, when using 8bit quantization, the map drops by 0.46%. This quantization method has very good scalability, and maps well on a fixed-point hardware.

| Mode               | mAP (%) |
|--------------------|---------|
| Float32            | 82.77   |
| Per-layer-Int8     | 77.23   |
| Per-channel-Int8   | 82.17   |
| DBQ-Int6           | 66.84   |
| DBQ-Int7           | 76.58   |
| DBQ-Int8           | 82.31   |
| DBQ-Int9           | 82.64   |
| DBQ-Int10          | 83.23   |
| DBQ-Int11          | 83.19   |
| DBQ-Int12          | 82.55   |
Table 2. Quantization PSNR achieved on different CNNs with last Conv layer results.

| Model        | PSNR  |
|--------------|-------|
| YOLOv3-416   | 44.39 |
| YOLOv3-tiny  | 51.22 |
| SSD-300      | 61.84 |
| ResNet-34    | 54.31 |

5. Conclusions
This paper presents a dynamic balance quantization method that can be applied to many networks based on computer vision convolutional design. This method does not require fine-tuning and does not rely on training set. When a deep learning model is deployed on integer-arithmetic-only hardware, the data and network security can be guaranteed.

The experiment gives the result of mAP loss quantizing an official pre-trained YOLOv3 model. Using 8-bit fixed point arithmetic can achieve accuracy close to floating point inference. We recommend converting the models designed with different frameworks to ONNX model before using this quantitative method. We hope that this method will be helpful for both practical applications and future research on quantizing deep learning models.

References
[1] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, (IEEE Computer Society), pp. 2704–2713.
[2] Nagel, M., Baalen, M.V., Blankevoort, T., and Welling, M. (2019). Data-free quantization through weight equalization and bias correction. In Proceedings of the IEEE International Conference on Computer Vision, (Institute of Electrical and Electronics Engineers Inc.), pp. 1325–1334.
[3] Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. https://arxiv.org/abs/1806.08342
[4] Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. In 32nd International Conference on Machine Learning, ICML 2015, (International Machine Learning Society (IMLS)), pp. 1737–1746.
[5] Gysel, P., Pimentel, J., Motamedi, M., and Ghiassi, S. (2018). Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks. IEEE Transactions on Neural Networks and Learning Systems 29, 5784–5789.
[6] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, (IEEE Computer Society), pp. 2704–2713.
[7] Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. (2017). Incremental network quantization: Towards lossless cnns with low-precision weights. In 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, (International Conference on Learning Representations, ICLR), p.
[8] Louizos, C., Reisser, M., Blankevoort, T., Gavves, E., and Welling, M. (2019). Relaxed quantization for discretized neural networks. In 7th International Conference on Learning Representations, ICLR 2019, (International Conference on Learning Representations, ICLR), p.
[9] Finkelstein, A., Almog, U., & Grobman, M. (2019). Fighting quantization bias with bias.
[10] Eldad, M., Alexander, F., Uri, A., and Mark, G. (2019). Same, same but different: Recovering neural network quantization error through weight factorization. In 36th International Conference on Machine Learning, ICML 2019, (International Machine Learning Society (IMLS)), pp. 7886–7896

https://arxiv.org/abs/1906.03193