QUANTIZATION – LƯỢNG TỬ HÓA

Trần Quang Minh

Hà nội, 22/04/2024

MỤC LỤC

REFERENCE

[1]. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference (2017)*
https://arxiv.org/pdf/1712.05877.pdf

[2]. *(Github) Gemmlowp*
https://github.com/google/gemmlowp/tree/master

[3]. *(PyTorch) Quantization*
https://pytorch.org/docs/stable/quantization.html

[4]. *A Survey of Quantization Methods for Efficient Neural Network Inference (2021)*
https://arxiv.org/pdf/2103.13630.pdf

[5]. *(Viblo) Quantization với Pytorch*
https://viblo.asia/p/quantization-voi-pytorch-phan-1-3Q75wv6GlWb
https://viblo.asia/p/quantization-voi-pytorch-phan-2-Qbq5QBBwKD8

IMPLEMENT

[6]. /data/minhtq22/content-detection/yolov83-human/quantize_infer_model/

[7]. /data/minhtq22/content-tracking/mot-human-tf2/quantize_infer_model/

PHẦN 1: KHÁI QUÁT CHUNG VỀ CÁC THUẬT TOÁN QUANTIZATION

Lượng tử hóa mô hình (Quantization) là một kĩ thuật nén mô hình (Compression) nhằm giúp giảm dung lượng lưu trữ mô hình, tăng thời gian inference với sự đánh đổi là giảm độ chính xác đi một mức đủ nhỏ để chấp nhận được.

Tùy mục đích bài toán (như giảm dung lượng lưu trữ, hay tăng thời gian chạy realtime, hay chạy tương thích phần cứng cụ thể, ...), có nhiều kiểu thuật toán Quantization. Có thể chia các thuật toán đó theo các tiêu chí như sau:

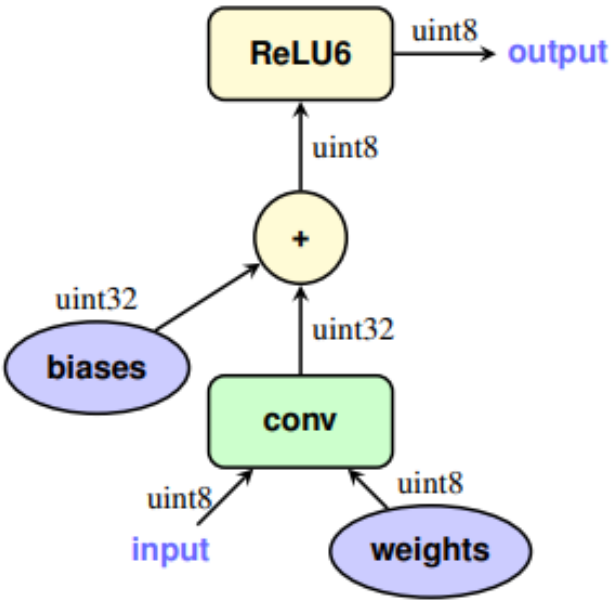
Theo cách inference:

- ***Dynamic Quantization*** : Trọng số được quantize trước, các kích hoạt được quantize trong lúc inference – như vậy mỗi dữ liệu sẽ được linh động một dải riêng ở mỗi bước quantize kích hoạt.
- ***Static Quantization*** : Trọng số và kích hoạt đều được quantize trước khi inference với những bộ số scale chung cho tất cả dữ liệu. Để tính được bộ số này thuật toán sử dụng 1 tập dữ liệu nhỏ đầu vào để calib range cho các output sau mỗi layer, các range này của tập calib được xem như đại diện cho tất cả dữ liệu. Như vậy so với Dynamic thì độ chính xác có thể thấp hơn một chút nhưng đánh đổi lại sự nhanh nhẹn, giảm thời gian inference vì không phải tính lại range cho output sau mỗi layer/kích hoạt trong model.
- ***Simulated Quantization*** : Phương pháp này là mô phỏng quantize – tức là chỉ quantize trọng số còn inference thì trên số float-point. Như vậy nếu xét về độ chính xác thì cao hơn 2 loại trên (vì không phải làm tròn các output) nhưng thực ra chỉ giúp nén mô hình nhỏ dung lượng lại và sau mỗi layer phải dequantize trọng số từ int/low-precision sang fp32 để inference, không giúp giảm thời gian tính toán.

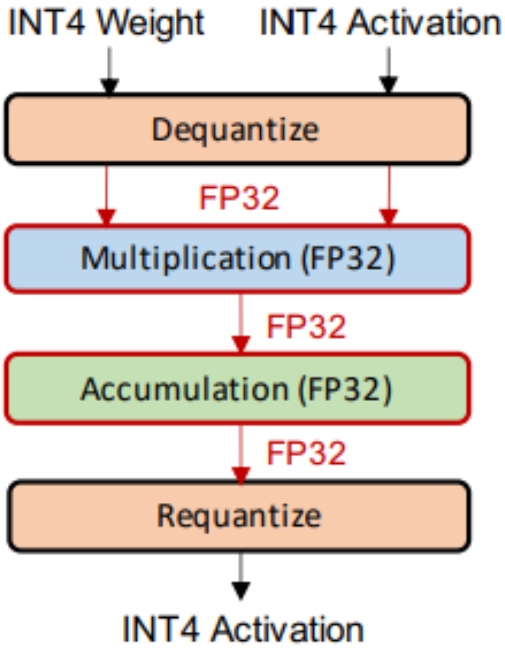
Theo sự yêu cầu tinh chỉnh bằng training:

- ***Post-training Quantization (PTQ)*** : Quantize sau training truyền thống.
- ***Quantization Aware Training (QAT)*** : Quantize trong lúc training (lượng tử hóa đào tạo nhận thức) để tinh chỉnh mô hình và quantize luân phiên nhau, giúp giảm sai khác của model trước và sau khi quantize lần cuối. Vì thế QAT cải thiện độ chính xác so với PTQ.

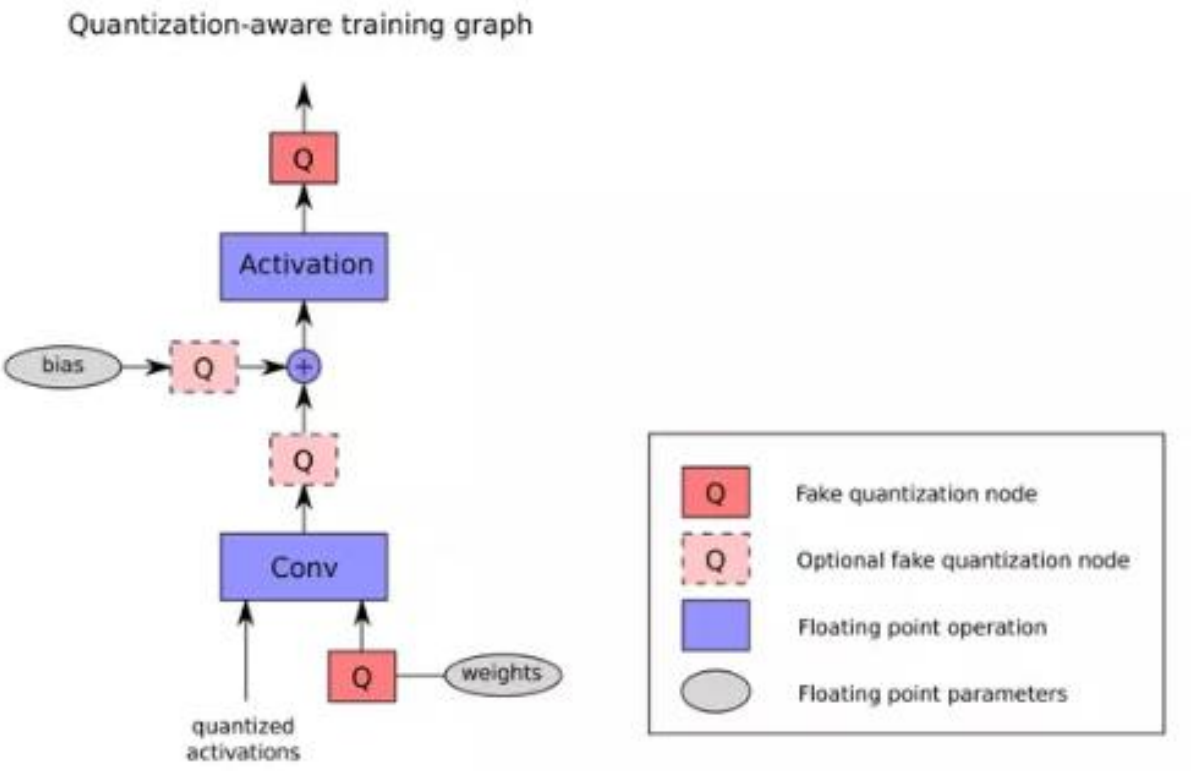
Trong core ***Tmoby*** chạy demo ***Yolov8 – human tracking*** và ***SiamRPN – single object tracking*** trên ZCU-102, kỹ thuật quantize được sử dụng là *Static Quantization*.



Hình 1. Integer-only inference (dynamic/static quantization)



Hình 2. Simulated quantization. Hình trên ví dụ cho trường hợp quantize 4 bit



Hình 3. Sơ đồ Quantiza Aware Training. Lớp fake-quantization được chèn vào để giả quantize – đưa weight từ fp32 về vẫn là fp32 tuy nhiên được điều chỉnh đến những giá trị đại diện cho giá trị ở dải int, sẽ nói kĩ hơn ở phần QAT.

PHẦN 2: STATIC QUANTIZATION

Phần này trình bày về cách quantize để lấy các trọng số quantize và các fix-point. Các thuật toán trong phần này dựa trên lý thuyết về Static Quantization cơ bản và dựa trên việc đáp ứng cho core Tmoby đang dùng. Các đáp ứng cho core Tmoby cần lưu ý trong phần này đó là ***uint8-quantization***, ***layer-wise-quantization***.

Một số quy ước đặt biến trong các phép toán bên dưới như *S* (scale, fp32), M_0 (int32), *n* (shift, int), *Z* (zero-point, uint8), x_q (data quantize, uint8), *x* (data original, fp32), phép toán lấy Scale, Multiplier và Shift, ... , tham khảo **[1]**.

$$S = \frac{max_{fp32} - min_{fp32}}{255} \rightarrow (M_0\,, n)$$

$$M_0, n \coloneqq np.frexp(S)$$
$$M_0, n \coloneqq round(M_0 * 2^{31}), n$$

2.1. Convolution, Linear layers

Phần này nói chung cho các layer Convolution, Pointwise, Depthwise, Dense (Linear). Phép × trong công thức đại diện cho phép nhân ma trận với layer Dense/Linear, hoặc phép tích chập với bộ cộng tích lũy trong các layer Convolution, PW, DW.

2.1.1. Lấy các fix-point

$$y = x_1 \times x_2 + bias$$

$$\rightarrow S_o(y_q - Z_o) = S_1(x_{1q} - Z_1) \times S_2(x_{2q} - Z_2) + S_1 S_2 bias_q$$

$$\rightarrow y_q = Z_o + \frac{S_1 S_2}{S_o}\left[(x_{1q} - Z_1) \times (x_{2q} - Z_2) + bias_q\right]$$

Đặt $\frac{S_1 S_2}{S_o} = M$. Lấy các thông số Multiplier và Shift:

$$M \rightarrow (M_0\,, n)$$

Các Zero-point, Multiplier và Shift ở trên là các fix-point cần thiết.

2.1.2. Inference

Inputs : $(inp_q\,, Z_i)$

Weights : $(w\,, Z_w\,, bias_q)$

$$y \coloneqq (\sum) round\left(\frac{(inp_q - Z_i)(w_q - Z_w) * M_0}{2^{31}}\right)$$

($(\sum)$ có nghĩa lấy tổng các số sau làm tròn trong trường hợp đang tính bộ cộng tích lũy).

$$y \coloneqq round\left(\frac{y}{2^{-n}}\right) + Z_o$$

$$out \coloneqq ReLU(255)(y)$$

Vậy out là output sau layer add trên.

2.2. Add layers

2.2.1. Lấy các fix-point

$$y = x_1 + x_2$$

$$\rightarrow S_o(y_q - Z_o) = S_1(x_{1q} - Z_1) + S_2(x_{2q} - Z_2)$$

Không mất tính tổng quát, giả sử $S_2 < 2S_1$, tức là $\frac{S_2}{2S_1} < 1$

$$\rightarrow \frac{S_o}{2S_1}(y_q - Z_o) = \frac{1}{2}(x_{1q} - Z_1) + \frac{S_2}{2S_1}(x_{2q} - Z_2)$$

Đặt $\frac{S_o}{2S_1} = M_o$, $\frac{1}{2} = M_1$, $\frac{S_2}{2S_1} = M_2$. Lấy các thông số Multiplier và Shift:

$$M_o \rightarrow (M_{o0}, n_o)$$
$$M_1 \rightarrow (M_{10}, n_1)$$
$$M_2 \rightarrow (M_{20}, n_2)$$

Ở phần này có thêm 1 fix-point nữa là Left Shift n_{left}, trong quantize 8-bit mặc định $n_{left} = 20$.

Như vậy các Zero-point, Multiplier, Shift và Left-shift là các fix-point đưa vào chương trình.

2.2.2. Inference

Inputs : $(x_{1q}, Z_1, M_{10}, n_1), (x_{2q}, Z_2, M_{10}, n_2)$

Các bước trong thuật toán quantize-infer add mô tả như sau:

$$\begin{cases} inp1 \coloneqq round\left[\dfrac{\left(x_{1q} - Z_1\right) * M_{10}}{2^{31-n_{left}}}\right] \\ inp2 \coloneqq round\left[\dfrac{\left(x_{2q} - Z_2\right) * M_{20}}{2^{31-n_{left}}}\right] \end{cases}$$

$$\begin{cases} inp1 \coloneqq round\left(\dfrac{inp1}{2^{-n_1}}\right) \\ inp2 \coloneqq round\left(\dfrac{inp2}{2^{-n_2}}\right) \end{cases}$$

$$y \coloneqq inp1 + inp2$$

$$y \coloneqq round\left(\frac{y * M_{o0}}{2^{31}}\right)$$

$$y \coloneqq round\left(\frac{y}{2^{n_{left}-n_o}}\right) + Z_o$$

$$out \coloneqq ReLU(255)(y)$$

Vậy out là output sau layer add trên.

2.3. Concatenate layers

2.3.1. Lấy các fix-point

Đặt range(x) = (min(x), max(x)), rõ ràng đối với Concatenate layer:

$$range(out)$$
$$= (\min([range(x)[0] for\ x\ in\ inputs]), \max([range(x)[1]\ for\ x\ in\ inputs])$$

nên range(out) lớn nhất trong tất cả range input và output.

$$y = concat(x_1, x_2, \dots, x_k)$$

$$\rightarrow S_o\left(y_q - Z_o\right) = concat\left(S_1\left(x_{1q} - Z_1\right), S_2\left(x_{2q} - Z_2\right), \dots, S_k\left(x_{kq} - Z_k\right)\right)$$

$$\rightarrow y_q - Z_o = concat\left(\frac{S_1}{S_o}\left(x_{1q} - Z_1\right), \frac{S_2}{S_o}\left(x_{2q} - Z_2\right), \dots, \frac{S_k}{S_o}\left(x_{kq} - Z_k\right)\right)$$

Đặt $\frac{S_1}{S_o} = M_1, \frac{S_2}{S_o} = M_2, \dots, \frac{S_k}{S_o} = M_k$. Lấy các thông số Multiplier và Shift:

$$M_1 \rightarrow (M_{10}, n_1)$$

$$M_2 \rightarrow (M_{20}\,,\, n_2)$$
$$\dots$$
$$M_k \rightarrow (M_{k0}\,,\, n_k)$$

Các Zero-point, Multiplier và Shift ở trên là các fix-point cần thiết.

2.3.2. Inference

Inputs : $(x_{1q}\,, Z_1\,, M_{10}\,, n_1)\,, (x_{2q}\,, Z_2\,, M_{10}\,, n_2)\,, \dots, (x_{kq}\,, Z_k\,, M_{k0}\,, n_k)$

Các bước trong thuật toán quantize-infer add mô tả như sau:

$$\begin{cases} inp1 \coloneqq round\left[\dfrac{(x_{1q} - Z_1) * M_{10}}{2^{31-n_1}}\right] \\ inp2 \coloneqq round\left[\dfrac{(x_{2q} - Z_2) * M_{20}}{2^{31-n_2}}\right] \\ \dots \\ inpk \coloneqq round\left[\dfrac{(x_{kq} - Z_k) * M_{k0}}{2^{31-n_k}}\right] \end{cases}$$

$$y \coloneqq concat([inp1\,, inp2\,, \dots, inpk])$$

$$y \coloneqq round\left(\frac{y * M_{o0}}{2^{31}}\right)$$

$$y \coloneqq round\left(\frac{y}{2^{-n_o}}\right) + Z_o$$

$$out \coloneqq ReLU(255)(y)$$

Vậy out là output sau layer concat trên.

2.4. Convolution-Batchnorm fuse

Batchnorm layer (BN) thường xuất hiện trong model khi training và thường đi thành module Conv-BN-ReLU. Tuy nhiên, khi inference thì có thể hợp nhất BN với lớp thuộc họ Convolution ngay phía trên nó để tính, bởi thực chất khi inference thì BN đóng vai trò chỉ là scale và kéo offset lại cho output sau Conv (BN thực hiện 1 biến đổi tuyến tính). Cũng nhờ việc fuse vào lớp Conv, ta cũng sẽ không cần thiết kế thuật toán quantize cho BN vì nó không còn tồn tại như là layer riêng biệt trong model nữa. Chỉ cần quantize layer Conv mới thay cho cả module Conv-BN (hoặc Conv-BN-ReLU). Tham khảo **[1].**

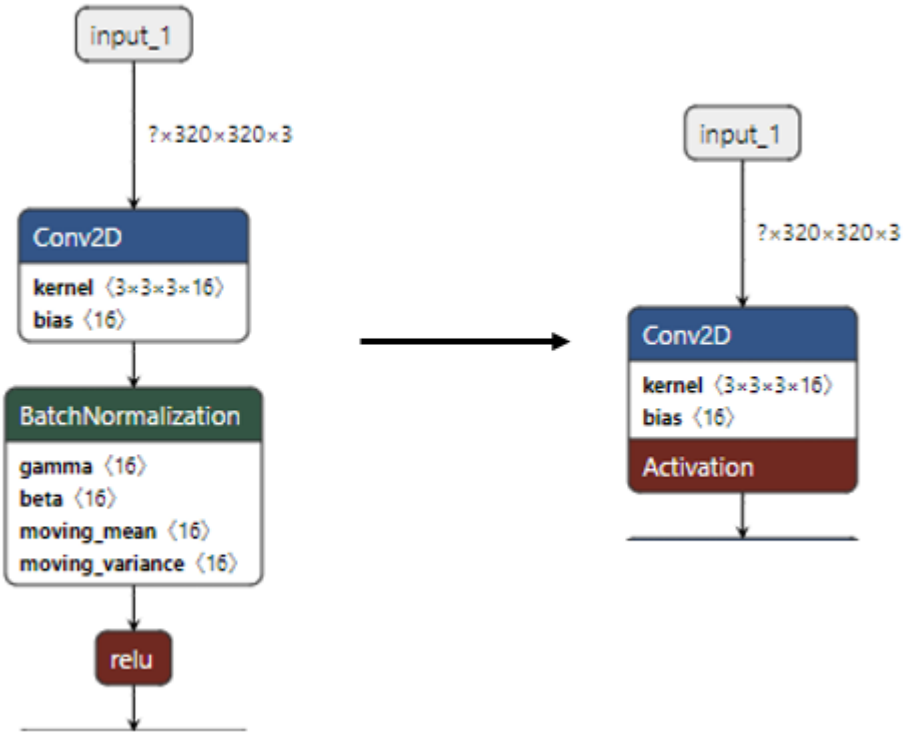
Đối với Conv-BN fuse.

Công thức thay đổi weights trong Conv thành weights fuse:

$$w_{\text{fold}} := \frac{\gamma w}{\sqrt{EMA(\sigma_B^2) + \varepsilon}}.$$

Code: ***/data/minhtq22/content-detection/yolov83-human/quantize_infer_model/save_quantize_light8n.py***, function ***fused_bn_to_cv***

Đối với Conv-BN-ReLU fuse. Đầu tiên BN cũng sẽ được fuse vào Conv như trường hợp trước được layer Conv mới, sau đó ReLU coi như là activation cho layer Conv mới này.



Hình 4. Fuse Conv-BN-ReLU to Conv

PHẦN 3: POST TRAINING QUANTIZATION VÀ QUANTIZATION AWARE TRAINING

3.1. Post-training Quantization (PTQ)

Sau khi train một model theo cách truyền thống, nếu model có BN thì sẽ đến bước fuse model theo các module (Conv-BN → Conv) hoặc (Conv-BN-ReLU → Conv). Sau đó thực hiện quantize lần lượt cho các layer trong model với gợi ý các bước chung như sau:

- Tìm range fp32 của input/output layer. Range này thì cần có 1 tập dữ liệu nhỏ đưa vào để calib đại diện cho toàn tập dữ liệu. Tập dữ liệu calib này thường từ 100 đến 1000 mẫu, tùy cách sử dụng. Tất nhiên tập càng lớn thì kì vọng calib càng tốt.

- Tính Zero-point và Scale cho mỗi input/output layer đó. Cặp (Z, S) này sẽ được sử dụng đại diện cho cả layer đó, cho mục đích quantize bản thân layer đó hoặc có thể là đem đi quantize cho layer phía sau nào đó nếu có connect đến nhau. Chú ý input của 1 layer cũng là output của layer nối phía trước nó để tránh tính toán lặp nhiều lần.

- Quantize mỗi layer theo thuật toán quantize trình bày trong phần 2.

3.2. Quantization Aware Training (QAT)

QAT cũng bao gồm 1 bước quantize cuối là PTQ, đó là sau khi model đã hoàn thành bước training-quantize đồng thời.

Trong bước training-quantize đồng thời: model cũng train như việc train bình thường, tuy nhiên có chút khác biệt đó là sau mỗi step train (có thể là sau mỗi epoch hoặc mỗi batch trong mỗi epoch) thì model sẽ chèn 1 lớp tính toán phụ gọi là ***fake-quantization***. Lớp này không quantize model về lower-bit (int, unit, fp16) mà model vẫn biểu diễn bởi float-point như 1 model float bình thường. Nhưng mỗi giá trị trọng số/bias sẽ biểu diễn đại diện cho giá trị low-bit (int, unit, fp16) mà đáng lẽ nó nhận được bằng cách quantize thông thường. Vậy nên lớp này gọi là fake-quantization để vừa làm model vừa có thể tham gia train tiếp với biểu diễn float, vừa không phải làm tròn lại khi quantize thật. Bản thân fake-quantization đã làm tròn mỗi trọng số về 1 giá trị gần nhất và hợp lý nhất nào đó tiện cho quantize thật và sau đó model sẽ tính lại loss và tiếp tục train để cải thiện sự mất độ chính xác gây ra bởi quantize.

Công thức giả quantize cho 1 số float như sau (**[1]**):
Biết range đã được calib là đoạn $[a, b]$, x là float-point value, fq là giá trị fake quantize (float-point), target quantize int8:

$$clamp(x) := \min(\max(x, a), b)$$

$$s \coloneqq \frac{b-a}{255}$$

$$fq \coloneqq \left[\frac{clamp(x)-a}{s}\right].s + a$$

Code tham khảo: */data/minhtq22/content-quantize/mobilenet-minimalistic-4bit/quantize_aware_training.py*

3.3. Thư viện Quanntization tham khảo

PyToch quantization, ***Tensorflow Lite***

Các thư viện lớn support quantize có thể không thể dùng flexible như là tự implement nhưng trong các trường hợp kiến trúc model đơn giản (1inp – 1out, các kích hoạt đơn giản, …) hay các kiến trúc nổi tiếng phổ thông mà tool có support, việc sử dụng hợp lý tool cũng có thể cho kết quả PTQ, QAT tốt hơn là so với việc tự implement từ đầu.