安谋科技
arm CHINA

周易 AIPU 量化算法

白皮书

文档编号：ACN-AWP-1002A
版本 1.0

周易 AIPU 量化算法白皮书

发布信息

版本历史

发行号	日期	保密性	变更
A	10/11/2021	非保密	初始开发版本

版权声明

目录

1 关于本文档

本文档旨在介绍周易 AIPU 量化算法。本文档提供了周易 AIPU 模型量化的背景知识、量化方案概述、算子介绍、量化实例教程、以及常见问题解答（FAQ）。

1.1 参考文档

序号	文档编号	标题
-	-	-

1.2 术语和缩写

本文档使用以下术语和缩写。

术语	含义
AI	人工智能
AIPU	人工智能处理单元
API	应用程序接口
CNN	卷积神经网络
NN	神经网络
NMS	非极大值抑制

2 背景知识

2.1 模型加速常用方案简介

2.1.1 更少的参数——模型裁剪

随着数据规模和模型复杂度的不断递增，典型的深度神经网络模型参数规模通常会在几十 M 甚至几百 M 的量级，加速模型前向运算最为直接有效的手段之一即是裁剪模型以减少参数量。比如，早期的 CNN 模型参数量主要集中在全连接层，在实际部署前，通常会对相关的全连接层参数做低秩分解，在允许模型效果微损的条件下，还会进一步对卷积层做分解 [1]。

同时，知识蒸馏技术 [2] [3]的发展成熟，使得模型裁剪带来的效果损失问题，已经能够获得很好的解决，因此在模型部署前，趋向于会对模型的 filters 直接做大幅度的削减以获得一个原始模型的瘦长型“克隆”小模型，然后用知识蒸馏的方式不断训练降低小模型的效果损失，可实现同等算法效果条件下成倍的速度提升。

另外，自动化模型结构设计技术 [4]的发展，使得针对特定运算平台的专用模型 [5] [6]参数能够大幅缩小，提供了更具性价比的基准模型。

2.1.2 更多的零值——模型稀疏化

影响模型运算性能的因素除了参数的多少外，参数本身的分布也是一个重要因素，对于 AI 模型而言，稀疏化 [7]不仅有利于参数的存储和传输加载，配合专门的软硬件模块设计，更能直接加速计算。不过与模型裁剪一样，要保证模型精度的微损，模型稀疏化也需要相应的训练微调。

在训练阶段，通过对模型参数施加适当的正则化约束，可以很容易地获得含大量零值的模型。模型的稀疏化程度，除了反应在零值的整体占比外，也反应在零值分布的规律性方面（即结构化稀疏和非结构化稀疏），一般而言，结构化稀疏对于训练微调有着更高的要求，但是对于底层的软硬件加速会更加友好。目前结构化稀疏技术的普及程度仍较为偏低，使得模型稀疏化更多还是应用在优化模型的存储和传输加载上。

2.1.3 更少的算子——计算图优化

AI 模型的计算过程可以用计算图表示（图中节点表示算子，边表示算子间的数据流），在最终的实际计算前，可以对相应的计算图做大量的优化，如常量折叠，算子合并，公共子表达式提取等。主流的 AI 框架都提供了相应的计算图优化模块或者 API 可以供直接调用，如 tensorflow grappler[8], onnxruntime[9], pytorch glow[10]等，此外还有一些独立第三方算法库也支持对模型的计算图优化 [11]。

---

[1] Accelerating Very Deep Convolutional Networks for Classification and Detection. https://arxiv.org/pdf/1505.06798.pdf

[2] Distilling the Knowledge in a Neural Network. https://arxiv.org/pdf/1503.02531.pdf

[3] MEAL V2: Boosting Vanilla ResNet-50 to 80%+ Top-1 Accuracy on ImageNet without Tricks. https://arxiv.org/pdf/2009.08453.pdf

[4] Learning Transferable Architectures for Scalable Image Recognition. https://arxiv.org/pdf/1707.07012.pdf

[5] Searching for MobileNetV3. https://arxiv.org/pdf/1905.02244.pdf

[6] MnasNet: Platform-Aware Neural Architecture Search for Mobile. https://arxiv.org/pdf/1807.11626.pdf

[7] Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. https://arxiv.org/pdf/2102.00554.pdf

[8] https://www.tensorflow.org/guide/graph_optimization

与模型裁剪和稀疏化不同，通常计算图优化会确保计算的等效性，从而严格保证了模型效果的不变，无需额外的训练调优。

2.1.4 更少的位宽——模型量化

用更少的位宽表示模型参数和中间运算变量，可实现模型存储、传输和运算的整体优化，同时由于整数运算的复杂度远低于浮点数运算，相关的硬件方案实现可以更具性价比。模型量化技术 [12] [13]在过去几年中快速发展成熟，目前已几乎成为模型前向运算的首要标配加速方案。

模型量化用于模型前向运算加速时，通常除了把浮点型的模型参数映射为中低比特（16 比特或 8 比特）的整型参数外，还会对一些复杂的数学函数运算做简化处理（比如通过查找表的方式）。由于 AI 模型的参数冗余性，量化后的模型效果损失微小，通过量化训练甚至可以恢复和超过原始模型的效果。模型量化技术对于模型本身的网络结构依赖度低，可以直接应用于绝大多数的 AI 模型。

2.1.5 更好的实现——算子优化

特定硬件平台上的模型运算性能还取决于具体的算子实现优化，有些优化规则可以是通用的（比如 Winograd 算法 [14]），有些规则可以是自动化的 [15]，有些规则则是深度绑定硬件特性的指令集优化或者汇编优化。

[9] https://onnxruntime.ai/docs/resources/graph-optimizations.html

[10] https://github.com/pytorch/glow/blob/master/docs/Optimizations.md

[11] https://github.com/jiazhihao/TASO

[12] Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. https://arxiv.org/pdf/1712.05877.pdf

[13] Quantizing deep convolutional networks for efficient inference: A whitepaper. https://arxiv.org/pdf/1806.08342.pdf

[14] Fast Algorithms for Convolution Neural Networks. https://arxiv.org/pdf/1509.09308.pdf

[15] TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. https://arxiv.org/pdf/1802.04799.pdf

2.2 模型量化原理概述

模型量化本质上是将模型运算（通常只涉及前向运算）涉及的浮点计算转换成低比特定点计算。显然，模型量化其实是一种近似算法，理论上必然带来模型效果的损失。

为简化计算，模型参数从浮点域映射到整数域通常使用线性变换，即线性量化[16]，一般所说的模型量化均是指代线性量化算法。令x_f表示原始的浮点值，x_q表示量化后的整数值，s_x表示缩放系数，zp_x表示零点值，n表示量化的位宽，则线性量化的计算公式（1）可表示如下：

$$\begin{aligned} x_q &= round\left(\underbrace{\left(x_f - min_{x_f}\right)\frac{2^n-1}{max_{x_f} - min_{x_f}}}_{s_x}\right) \\ &\approx round\left(s_x x_f - \underbrace{round\left(min_{x_f} s_x\right)}_{zp_x}\right) \\ &= round\left(s_x x_f - zp_x\right) \end{aligned} \tag{1}$$

需要注意的是，公式中的zp_x会被强制为整数，这对于整数域的零值 padding 来说是绝对必须的，同时$max_{x_f} == min_{x_f}$时会强制$s_x = 1.0$以应对除零问题。此外，公式中的x_q用了无符号数表示，实际应用中完全可以替换成有符号数的表示，只需要再减去2^{n-1}即可。另外，有些框架（如 tensorflow lite[17]）或者文献中对于线性量化的公式会写作：$x_f' = \left(x_q' - zp_x'\right) * s_x'$，容易发现这两种写法其实是等效的，相当于$s_x' = 1/s_x, zp_x' = -zp_x$。实际工程应用时，对于公式中得到的$x_q$还会按照量化后的整型类型做对应的 clamp 操作以保证其数值范围不越界。

另外，由公式（1）很容易得到对应的反量化公式为：$x_f'' = \left(x_q + zp_x\right)/s_x$，而$x_f''$和$x_f$并不严格相等（即反量化并不能完全恢复量化时丢失的信息）。

2.2.1 量化训练 VS 训练后量化

根据量化算法是否发生在训练阶段可以分为量化训练（quantization aware training, 也叫做在线量化）和训练后量化（post-training quantization, 也叫做离线量化）。

在量化训练过程中，通过引入伪量化操作（将浮点量化到定点，再反量化成浮点），前向时在每层参数的输入后和激活的输出前进行伪量化，以使模型获得量化损失，继而控制每层的参数和激活输出在一定范围内，且分布更均匀，从而降低实际量化后的效果损失。需要注意的是量化训练过程中，伪量化操作并不参与模型的反向传播过程。

顾名思义，训练后量化则是只在模型完成训练后，一次性地将相关的模型参数做量化计算，以获得最终的部署模型。

相较而言，量化训练能够获得更低的效果损失，TensorFlow 和 PyTorch 等主流的训练框架都已原生支持量化训练。但是在某些特殊情况下（如部署方无法获得合适的训练数据），训练后量化仍然是合适甚至唯一的选项。

[16] Range-Based Linear Quantization. https://intellabs.github.io/distiller/algo_quantization.html#quantization-algorithms

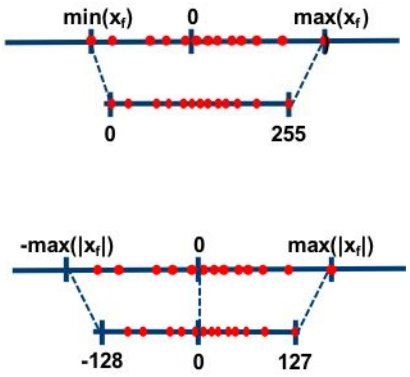
[17] https://www.tensorflow.org/lite/performance/quantization_spec

2.2.2 非对称量化 VS 对称量化

由公式（1）可知，量化后的零点值zp_x在数值上不保证一定为 0，此即为非对称量化。而在某些情形下，为了进一步简化计算（非对称量化模式下，硬件需要额外的逻辑去处理zp_x），需要保证零点值zp_x在数值上严格为 0，此即为对称量化。具体而言，对称量化的计算公式可以表示为：$x_q = round(s_x x_f)$，其中s_x的计算方式根据量化后x_q的取值范围不同而略有差异，详细情况如下：

	Full Range	**Restricted Range**
Quantized Range	$[-2^{n-1}, 2^{n-1}-1]$	$[-(2^{n-1}-1), 2^{n-1}-1]$
8-bit example	[-128, 127]	[-127, 127]
Scale Factor	$s_x = \frac{(2^n-1)/2}{max\left(abs(x_f)\right)}$	$s_x = \frac{2^{n-1}-1}{max\left(abs(x_f)\right)}$

非对称量化和对称量化的主要差异可以由下图简要表示：



2.2.3 按张量量化 VS 逐通道量化

根据量化算法映射系数的计算粒度不同可分为按张量（per-tensor）量化和逐通道（per-axis 或 per-channel）量化。按张量量化即对张量x_f整体计算出一对量化系数s_x和zp_x，逐通道（通常使用输出通道即 per output channel）量化则会对通道中的每一子项都计算一对s_x和zp_x，最终的量化系数即为向量$\hat{s_x}$和$\hat{zp_x}$.

逐通道量化会多消耗计算和存储资源，不过其对张量数值分布的反映更加细致，理论上具有更低的效果损失。

2.2.4 端到端量化 VS 部分量化

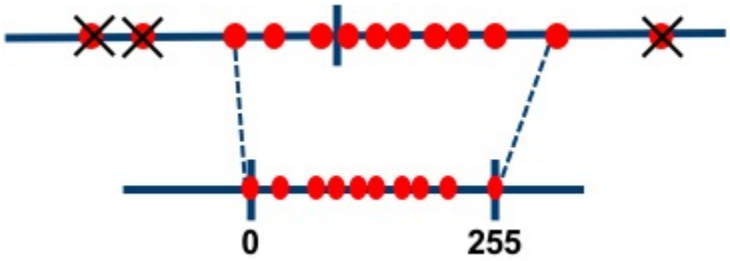
如果模型从输入到输出的全部运算全是在整数域上进行，即端到端量化，反之，则称为部分量化。部分量化可以是只对模型权重参数量化（主要用于压缩模型），实际前向计算时先反量化至浮点域再与各层的激活值运算，也可以是对部分层的权重参数和激活值都做量化运算于整数域，剩余层则运算于浮点域（比如某些层量化后效果损失过大，则不加以量化而完全用浮点数计算，注意此种情况下激活值从量化层传递到非量化层时需要先做反量化运算以映射回浮点域，同样的，激活值由非量化层传递到量化层时需要先做量化运算以映射到整数域）。

2.2.5 校准算法

由公式（1）可知，给定量化位宽和量化方式后，量化的结果主要取决于张量x_f的 min, max 值的选取（在 8bit 以下的超低 bit 量化情形，round 算法的选取也变得非常重要 [18]）。一般来说，直接统计数值上的最大最小值去做量化很可能

[18] Up or Down? Adaptive Rounding for Post-Training Quantization. https://arxiv.org/pdf/2004.10568.pdf

受到离群点的干扰而无法准确反应原始张量的实际分布规律，尤其对于训练后量化而言（量化训练由于训练过程中即考虑进了量化损失的影响，故基本不存在此种情况）。如下图所示，浮点域的最大最小值附近仅仅聚集了极少量的离群点，实际量化时最好剔除其影响以更精确地反应数值分布规律。



为了能够获得模型网络中各层激活值的真实分布信息，需要准备一个与模型训练集同分布的校准数据集（不需要标签信息），基于此数据集对模型做前向运算，并统计出各层激活值张量的所有必要信息（权重参数张量的信息统计不依赖于此校准数据集，因此既可以单独统计，也可以同时一并加以统计），然后根据这些统计信息应用相关的算法确定每个张量最终用于量化的 min, max 值。这个过程称为校准，常用的校准算法有：

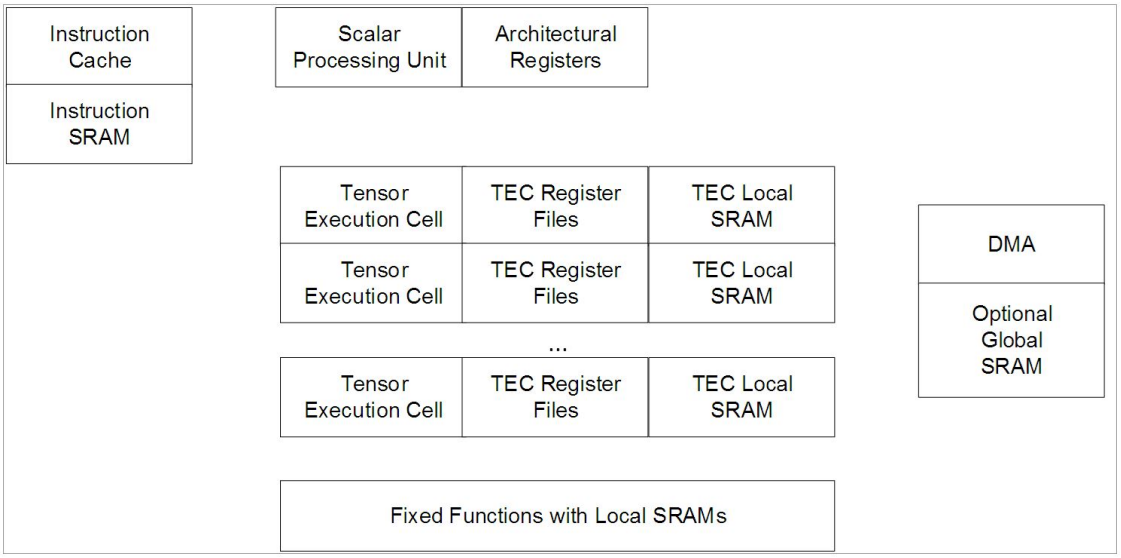
- extrema: 令max_i, min_i分别表示第i个批次的样本统计出的最大最小值，则最终采用的最大最小值分别为$max(max_1, max_2, \ldots, max_i, \ldots)$, $min(min_1, min_2, \ldots, min_i, \ldots)$。
- mean: 令max_i, min_i分别表示第i个批次的样本统计出的最大最小值，则最终采用的最大最小值分别为$\sum \alpha_i * max_i$, $\sum \alpha_i * min_i$ (where $\alpha_i \in [0,1]$ and $\sum \alpha_i = 1.0$)。
- Nstd: 令$mean$和std分别表示统计出的均值和标准差，则最终采用的最大最小值分别为$mean + N * std$, $mean - N * std$, 其中 N 是可配置参数。
- KLD: 利用 KL 散度度量量化后的张量数值分布和原始分布的差异，遍历求解使得 KL 散度最小化的门限值作为最终量化的最大最小值，具体算法步骤参见 [19]。

[19] 8-bit inference with tensorrt. page37-39. https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf

3 周易 AIPU 模型量化方案概述

3.1 周易 AIPU 硬件架构简介

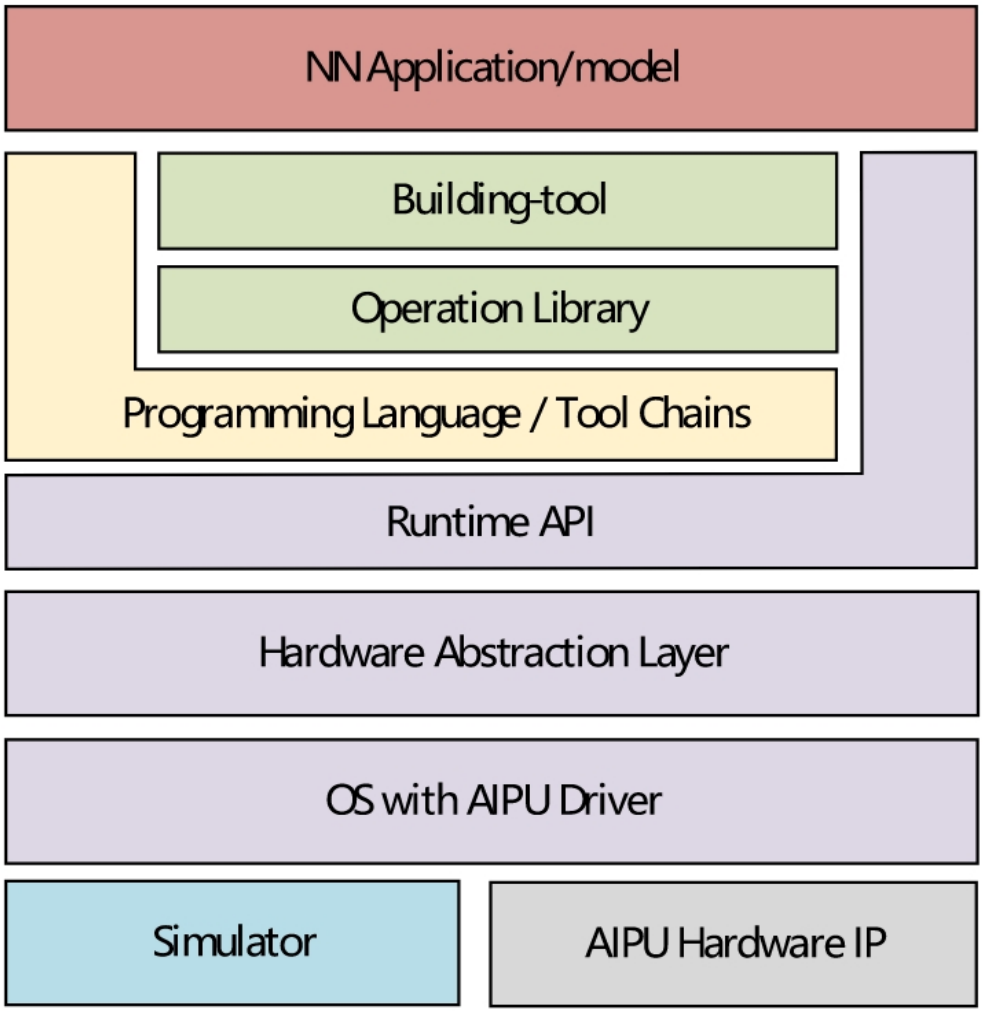
周易 AIPU 同时具备（Scalar, Vector, Tensor）通用算力模块和（Convolution, Fully-connected, BatchNormalizaion, Pooling, Eltwise, ...）AI 专用算力模块。典型的周易 AIPU 硬件架构如下图所示：



通常，AI 模型的大多数密集计算型的 layer 都能用专用算力模块 AIFF（AI Fixed Functions）来完成计算，其余的逻辑密集型的 layer（如 NMS 等后处理层）则使用通用算力模块完成计算。详细的模块构造和相应的参数细则可参考硬件规格文档。

3.2 周易 AIPU 软件栈简介

周易软件产品包括了：Toolchain（含 compiler, assembler, linker, debugger 等），NN compiler（又叫 building-tool, 含 parser, optimizer, gbuilder 等），Operation Library（含内置的各种典型 AI 算子库），Driver, Runtime, Simulator。典型的周易 AIPU 软件架构如下图所示：



对于模型的离线量化（即将 float IR 格式的模型转换优化为 quantized IR 格式的模型，关于周易 AIPU 的模型 IR 格式，具体可参考 *Arm China AI Platform Zhouyi Compass IR Definition Application Note*）由 optimizer 完成，相关模块的使用教程参考 *Arm China AI Platform Zhouyi Compass Software Programming Guide*。

3.3 周易 AIPU 模型量化方案

3.3.1 概要说明

为较好地兼顾模型的性能和效果损失，周易 AIPU 当前默认优先支持全对称量化方式，后续版本将会优先支持对权重参数的逐通道对称量化和对激活特征的逐张量非对称量化。Optimizer 对模型的量化由如下主要步骤构成：

- **statistic**: 计算模型中各张量的统计量，如 extrema, mean, std, histogram 等。
- **calibration**: 根据选择的校准算法确定各张量量化时使用的的 min,max 值。
- **quantization**: 计算模型中各张量的量化系数，并将权重、偏置等常量参数量化取整。
- **metric and serialization**: 计算对比量化前后的模型效果指标，并将量化后的模型序列化保存。

同时，为了获得部分算子的性能提升，会在上述步骤中间穿插一些相应的计算图优化（主要针对量化后整数域前向运算，更通用的计算图优化假定在原始模型上已经完成），具体内容会在下一章中各算子量化方案介绍时体现。

3.3.2 举例说明

3.3.2.1 四则运算

- 加法：

  由$c_f = a_f + b_f$

  可得$(c_q + zp_c)/s_c = (a_q + zp_a)/s_a + (b_q + zp_b)/s_b$

  从而$c_q = \left((a_q + zp_a)s_b + (b_q + zp_b)s_a\right)(s_c/(s_a s_b)) - zp_c$

  其中$s_a, s_b, s_c/(s_a s_b)$均为浮点数，且可在离线量化过程中确定，为便于在整数域计算，需离线阶段将这三个浮点数近似转换为$M/2^N$的规范形式（M, N 均为整数，整数域上除以2^N可用移位实现）

  最后容易简化得到：$c_q \approx \left(\left((a_q + zp_a)M_1 + (b_q + zp_b)M_2\right)\right)M_3/2^N) - zp_c$

- 减法：

  与加法类似，容易得到：$c_q \approx \left(\left((a_q + zp_a)M_1 - (b_q + zp_b)M_2\right)\right)M_3/2^N) - zp_c$

- 乘法：

  由$c_f = a_f b_f$

  可得$(c_q + zp_c)/s_c = \left((a_q + zp_a)/s_a\right)\left((b_q + zp_b)/s_b\right)$

  从而$c_q = (a_q + zp_a)(b_q + zp_b)(s_c/(s_a s_b)) - zp_c$

  其中$s_c/(s_a s_b)$为浮点数，且可在离线量化过程中确定，为便于在整数域计算，近似转换为$M/2^N$的规范形式（M, N 均为整数，整数域上除以2^N可用移位实现）

  最后容易简化得到：$c_q \approx (a_q + zp_a)(b_q + zp_b)(M/2^N) - zp_c$

- 除法：

  由$c_f = a_f/b_f$

  可得$(c_q + zp_c)/s_c = \left((a_q + zp_a)/s_a\right)/\left((b_q + zp_b)/s_b\right)$

  从而$c_q = \left((a_q + zp_a)/(b_q + zp_b)\right)((s_c s_b)/s_a) - zp_c$

  其中$(s_c s_b)/s_a$为浮点数，且可在离线量化过程中确定，为便于在整数域计算，近似转换为$M/2^N$的规范形式（M, N 均为整数，整数域上除以2^N可用移位实现）

  最后容易简化得到：$c_q \approx \left((a_q + zp_a)M//(b_q + zp_b)\right)/2^N - zp_c$，//表示在整数域中做整除运算（也可以进一步算出余数，并据此决定是否对商做进一步的四舍五入，实际工程实现时也可用$x/y \approx (x + y//2)//y$达成）。

3.3.2.2 非线性函数运算

对于复杂的非线性函数运算，效率起见，通常采取 lut 查找表方案（lut 表越大，相应的计算精度损失越小，但占用的存储计算资源也越多，如何合理利用 lut 表，在下一章会结合相应算子的量化方案再加以展开）实现，以 sigmoid 函数为例：

由 $y_f = sigmoid(x_f)$ 可得 $(y_q + zp_y)/s_y = sigmoid\left((x_q + zp_x)/s_x\right)$，从而 $y_q = sigmoid\left((x_q + zp_x)/s_x\right)s_y - zp_y$。因此可在离线量化时对$x_q$的可能值集合向量$\widehat{x_q}$先用$x$的量化系数反量化，然后计算 sigmoid 函数值，再用y的量化系数量化得到最终的 lut 表，实际前向运算时只需要查表取值即可。需要说明的是，受限于硬件资源，lut 不可能任意大，当$\widehat{x_q}$范围较大时，可以利用部分函数的对称性只存储半张表，或者直接对$\widehat{x_q}$做降采样以缩小 lut 表，在前向运算时再通过镜像取反或插值的方式获得最终的查找值。

4 周易 AIPU 模型量化方案分算子介绍

4.1 Abs

算子定义

Absolute takes one input data (Tensor) and produces one output data (Tensor) where the absolute is, y = abs(x), is applied to the tensor elementwise.

量化方法

由$y_f = abs(x_f)$可得$(y_q + zp_y)/s_y = abs\left((x_q + zp_x)/s_x\right)$, 从而$y_q = abs(x_q + zp_x)s_y/s_x - zp_y$。当采用对称量化时，可以进一步令$s_y$等于$s_x$，则整数域上的 abs 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.2 ArgMax

算子定义

Returns the index with the largest value of the input tensor X along the specified axis.

量化方法

由$y_f = ArgMax(x_f)$可得$(y_q + zp_y)/s_y = ArgMax\left((x_q + zp_x)/s_x\right)$, 从而$y_q = (ArgMax(x_q) + zp_x)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 ArgMax 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.3 ArgMin

算子定义

Returns the index with the smallest value of the input tensor X along the specified axis.

量化方法

略。参考 ArgMax。

规格说明

略。参考 ArgMax。

4.4 AveragePooling2D

算子定义

AveragePooling2D takes an input tensor X and applies average pooling across the tensor by the kernel size, stride size, and padding. AveragePooling-2D consists of computing the average on all values of a subset of the input tensor according to the kernel size and down sampling the data into the output tensor Y for further processing.

量化方法

由算子定义可知，核心的计算为求平均即$y_f = (\sum_j^N x_f)/N$, 易得$(y_q + zp_y)/s_y = \left(\sum_j^N \left((x_q + zp_x)/s_x\right)\right)/N$, 从而$y_q = \left((\sum_j^N x_q)/N + zp_x\right)(s_y/s_x) - zp_y$。令$s_y$等于$s_x$，$zp_y$等于$zp_x$，则整数域上的运算方式可与浮点域上一致，无需多余的量化相关操作。由于 N 为确定的常数，为进一步加速计算，可将$1/N$提前转换表示为$M/2^N$的规范形式（M, N 均为整数，整数域上除以2^N可用移位实现）。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.5 BatchNormalization

算子定义

Computes the batch normalization according to the model's running.

量化方法

仅考虑前向运算，BatchNormalization 是典型的线性运算： $y = Wx + b$ 。
由 $(y_q + zp_y)/s_y = \left((W_q + zp_W)/s_W\right)\left((x_q + zp_x)/s_x\right) + (b_q + zp_b)/s_b$,
可得： $y_q = \left((W_q + zp_W)(x_q + zp_x) + (b_q + zp_b)(s_w s_x/s_b)\right)s_y/s_w s_x - zp_y$,
对偏置项的量化，通常取$s_b = s_w s_x, zp_b = 0$, 同时可离线将$s_y/s_w s_x$表示为规范形式$M/2^N (M, N$均为整数),
从而： $y_q = \left((W_q + zp_W)(x_q + zp_x) + b_q\right)M/2^N - zp_y$。

BatchNormalization 一般对于量化位宽比较敏感，通常会默认对其权重参数取较高的量化位宽（16bit）。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
Weight 0:
   data_type  : int
   granularity: per-tensor or per-channel
Bias 0:
   data_type  : int
   granularity: per-tensor or per-channel
restriction: bias_scale, bias_zero_point = input_scale * weight_scale, 0.
```

4.6 BatchToDepth

算子定义

BatchToDepth rearranges data from the batch into blocks. More specifically, this operator outputs a copy of the input tensor where values from the batch dimension are moved into blocks of the Channel dimensions.

量化方法

由$y_f = BatchToDepth(x_f)$可得$(y_q + zp_y)/s_y = BatchToDepth(x_q + zp_x)/s_x)$，从而$y_q = (BatchToDepth(x_q) + zp_x)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.7 BatchToSpace

算子定义

BatchToSpace rearranges (permutes) data from the batch into blocks of spatial data. This is the reverse transformation of SpaceToBatch. More specifically, this operator outputs a copy of the input tensor where values from the batch dimension are moved in spatial blocks to the 'Height' and 'Width' dimensions.

量化方法

由$y_f = BatchToSpace(x_f)$可得$(y_q + zp_y)/s_y = BatchToSpace(x_q + zp_x)/s_x)$，从而$y_q = (BatchToSpace(x_q) + zp_x)s_y/s_x - zp_y$。令$s_y$等于$s_x$，$zp_y$等于$zp_x$，则整数域上的运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.8 BiasAdd

算子定义

Performs bias addition to the input data tensor X.

量化方法

由算子定义$y_f = x_f + B$。
可得$(y_q + zp_y)/s_y = (x_q + zp_x)/s_x + (B_q + zp_B)/s_B$,
即$y_q = \left((x_q + zp_x) + (B_q + zp_B)s_x/s_B\right)s_y/s_x - zp_y$,
为简化计算，令$s_B = s_x, zp_B = 0$，并离线将s_y/s_x表示为规范形式$M/2^N(M, N$均为整数),
从而$y_q = \left((x_q + zp_x) + B_q\right)M/2^N - zp_y$。
（将$zp_x + B_q$离线计算存储为新的B'_q可进一步加速前向运算）

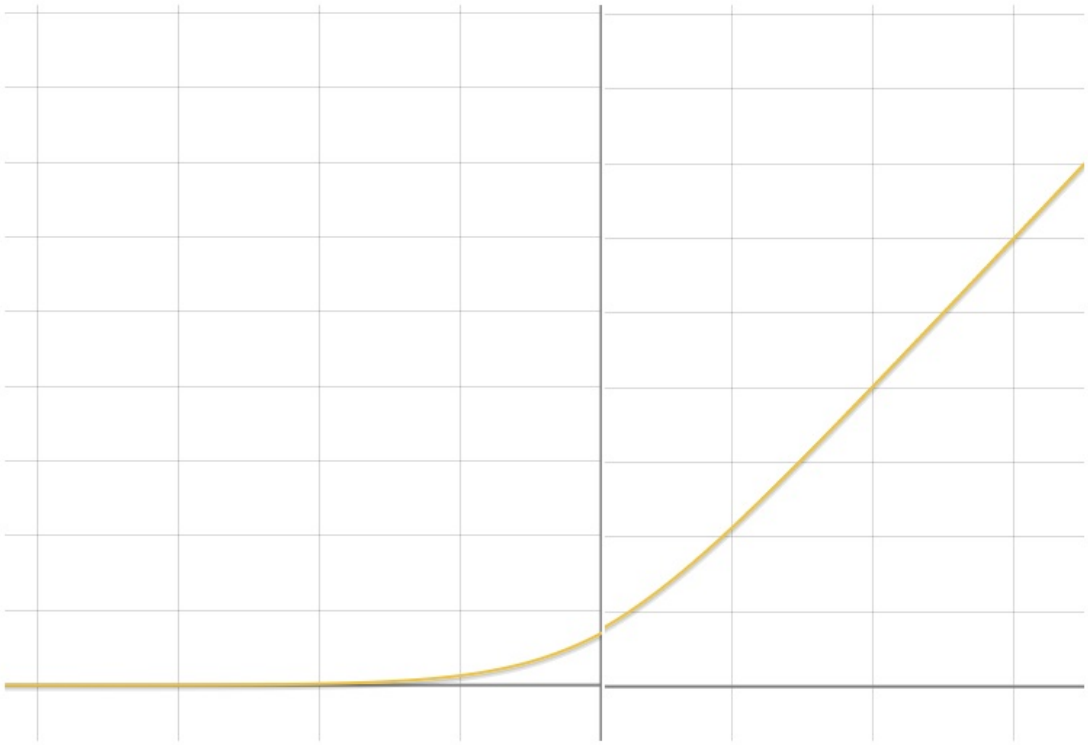
规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
Bias 0:
   data_type  : int
   granularity: per-tensor or per-channel
restriction: bias_scale, bias_zero_point = input_scale, 0
```

4.9 BNLL

算子定义

Takes one input tensor and produces one output tensor, where the function `y = x + log(1 + exp(-x)) if x > 0, otherwise y = log(1 + exp(x))`, is applied to the tensor elementwise.



量化方法

对于$y = BNLL(x)$有：$(y_q + zp_y)/s_y = BNLL\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = BNLL\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 BNLL 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。同时需要注意的是 BNLL 函数沿 x 负半轴缓慢下降趋近于零，若输入x的 min 取值过小则会导致量化后的值大量趋同无法区分（即量化后表达精度不够），影响整体算子精度，也浪费了 lut 空间，故在调节 BNLL 算子精度时需重点调节x的 min 取值。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.10 Cast

算子定义

Casts an input tensor (X) with a given data type and return a new output tensor (Y).

量化方法

由$y_f = Cast(x_f)$有$(y_q + zp_y)/s_y = Cast\left((x_q + zp_x)/s_x\right)$, 从而$y_q = Cast\left((x_q + zp_x)/s_x\right)s_y - zp_y$，进而可以有$y_q \approx (Cast(x_q) + zp_x)s_y/s_x - zp_y$。若令$s_y = s_x, zp_y = zp_x$，则整数域上的运算仅需将$x_q$的数据类型转换为$y_q$的数据类型即可。相关的量化损失主要来自于$x_f$到$x_q$的量化损失。

如果对于 Cast 算子精度有更高要求，还可以直接通过 lut 表实现$y_q = Cast\left((x_q + zp_x)/s_x\right)s_y - zp_y$，即：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Cast 结果获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算时通过查表（如果 lut 是采样后的小表，还需做相应的插值操作），得到最终的y_q。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.11 Clip

算子定义

Limits the given input tensor within an interval. The interval is specified by the inputs "cmin" and "cmax".

量化方法

由$y_f = Clip(x_f, cmin_f, cmax_f)$有$(y_q + zp_y)/s_y = Clip\left((x_q + zp_x)/s_x, cmin_f, cmax_f\right)$, 从而：

$$y_q = Clip\left((x_q + zp_x)/s_x, cmin_f, cmax_f\right)s_y - zp_y$$
$$y_q = Clip\left((x_q + zp_x)s_y/s_x,\ cmin_f * s_y,\ cmax_f * s_y\right) - zp_y$$
$$y_q = Clip\left((x_q + zp_x)s_y/s_x - zp_y,\ cmin_f * s_y - zp_y,\ cmax_f * s_y - zp_y\right)$$

离线将s_y/s_x表示为规范形式$M/2^N(M, N$均为整数)，$cmin_f, cmax_f$按y的量化系数量化为$cmin_q, cmax_q$, 则最终：
$y_q = Clip\left((x_q + zp_x)M/2^N - zp_y,\ cmin_q,\ cmax_q\right)$。
为最大限度利用量化位宽，y量化时所用的 min,max 即可定为 cmin,cmax。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.12 Concat

算子定义

This operator concatenates a list of tensors into a single tensor. All input tensors must have the same shape, except for the dimension size of the axis to concatenate on.

量化方法

由$y_f = Concat\left(x_{1_f}, x_{2_f}, \dots\right)$可得：$\left(y_q + zp_y\right)/s_y = Concat\left(\left(x_{1_q} + zp_{x_1}\right)/s_{x_1}, \left(x_{2_q} + zp_{x_2}\right)/s_{x_2}, \dots\right)$, 从而：

$y_q = Concat\left(\left(x_{1_q} + zp_{x_1}\right)/s_{x_1}, \left(x_{2_q} + zp_{x_2}\right)/s_{x_2}, \dots\right)s_y - zp_y$,

$y_q = Concat\left(\left(x_{1_q} + zp_{x_1}\right)s_y/s_{x_1}, \left(x_{2_q} + zp_{x_2}\right)s_y/s_{x_2}, \dots\right) - zp_y$,

可离线将$s_y/s_{x_1}, s_y/s_{x_2}, \dots$表示为规范形式$M_1/2^{N_1}, M_2/2^{N_2}, \dots$, 则有：

$y_q = Concat\left(\left(x_{1_q} + zp_{x_1}\right)M_1/2^{N_1}, \left(x_{2_q} + zp_{x_2}\right)M_2/2^{N_2}, \dots\right) - zp_y$,

为进一步加速计算，可在不影响精度的条件下，微调各输入分支的量化系数使得$s_y = s_{x_1} = s_{x_2} = \dots$, $zp_y = zp_{x_1} = zp_{x_2} = \dots$, 从而无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : int or uint
   granularity: per-tensor
Input 2:
   data_type  : int or uint
   granularity: per-tensor
...
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.13 Constant

算子定义

Creates a constant tensor as a given format. Generally, the data type is inferred from the type of value.

量化方法

若表征的常量为不需要量化的数值（如表征 index 相关的量），则强令其量化系数为$s = 1.0, zp = 0$以避免错误量化带来的损失。周易 AIPU 会通过在 float IR 中为 Constant 节点的 layer_top_type 字段指定 int 类型（由 parser 模块完成）来标示对应的 Constant 为不需要量化的数值。

规格说明

```
Weight 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: depending on the layer_top_type field in float IR to indicate whether the
 scale, zero_point should be set to 1.0 and 0
```

4.14 Convolution2D

算子定义

Performs a filter on the input tensor (X) and produces the output tensor (Y). If bias is necessary, a bias vector will be added to the output tensor. Similarly, if the activation function is necessary, it is applied to the output tensor as well.

量化方法

根据是否含有融合的激活函数（当前支持融合的激活函数为 CLIP, RELU, RELU6, LEAKYRELU）分别讨论：

- 无激活函数，即$y_f = Conv(x_f) = w_f x_f + b_f$。
  由$(y_q + zp_y)/s_y = \left((w_q + zp_w)/s_w\right)\left((x_q + zp_x)/s_x\right) + (b_q + zp_b)/s_b$,
  可得$y_q = \left((w_q + zp_w)(x_q + zp_x) + (b_q + zp_b)s_w s_x/s_b\right) s_y/s_w s_x - zp_y$,
  对偏置项的量化通常取$s_b = s_w s_x, zp_b = 0$, 同时可离线将$s_y/s_w s_x$表示为规范化的形式$M, 2^N$(M, N均为整数),
  从而$y_q = \left((w_q + zp_w)(x_q + zp_x) + b_q\right) M/2^N - zp_y$。
- 融合激活函数 CLIP, 即$y_f = CLIP\left(Conv(x_f), c_f, d_f\right) = CLIP\left(w_f x_f + b_f,\ c_f,\ d_f\right)$.
  由$(y_q + zp_y)/s_y = CLIP\left(\left((w_q + zp_w)/s_w\right)\left((x_q + zp_x)/s_x\right) + (b_q + zp_b)/s_b,\ c_f,\ d_f\right)$,
  可得$y_q = CLIP\left(\left((w_q + zp_w)(x_q + zp_x) + (b_q + zp_b)s_w s_x/s_b\right) s_y/s_w s_x - zp_y,\ c_f s_y - zp_y,\ d_f s_y - zp_y\right)$,
  对偏置项的量化通常取$s_b = s_w s_x, zp_b = 0$, 同时可离线将$s_y/s_w s_x$表示为规范化的形式$M, 2^N$(M, N均为整数),
  c_f, d_f按y的量化系数量化为c_q, d_q,
  从而$y_q = CLIP\left(\left((w_q + zp_w)(x_q + zp_x) + b_q\right) M/2^N - zp_y,\ c_q,\ d_q\right)$。
  为最大限度利用量化位宽，y量化时所用的 min,max 即可定为c_f, d_f。

- 融合激活函数 RELU, 即$y_f = RELU(Conv(x_f), c_f, d_f)$。
  RELU 可看成是特殊的 CLIP, 即$y_f = CLIP((Conv(x_f), c_f, d_f) = CLIP(Conv(x_f),\ 0,\ +\infty)$，则量化方法等效于 CLIP 的情况。
- 融合激活函数 RELU6, 即$y_f = RELU6(Conv(x_f), c_f, d_f)$。
  RELU6 可看成是特殊的 CLIP, 即$y_f = CLIP((Conv(x_f), c_f, d_f) = CLIP(Conv(x_f),\ 0,\ 6)$，则量化方法等效于 CLIP 的情况。
- 融合函数 LEAKYRELU, 即$y_f = LEAKYRELU\left(Conv(x_f)\right) = maximum(Conv(x_f), 0) + minimum(Conv(x_f), 0)\ l_f$。
  由$(y_q + zp_y)/s_y = maximum\left(Conv\left((x_q + zp_x)/s_x\right),\ 0\right) + minimum\left(Conv\left((x_q + zp_x)/s_x\right),\ 0\right)\ (l_q + zp_l)/s_l$,
  可得$y_q = maximum\left(Conv\left((x_q + zp_x)/s_x\right)\ s_y,\ 0\right) + minimum\left(Conv\left((x_q + zp_x)/s_x\right)\ s_y,\ 0\right)\ (l_q + zp_l)/s_l - zp_y$，为简便计算，令$s_l = 2^{N_l}, zp_l = 0$($N_l$为整数，$l_q = round(l_f * 2^{N_l})$即量化后的 negative_slope),
  即得$y_q = maximum\left(\left((w_q + zp_w)(x_q + zp_x) + b_q\right)M/2^N, 0\right) + minimum\left(\left((w_q + zp_w)(x_q + zp_x) + b_q\right)M/2^N, 0\right)l_q/2^{N_l} - zp_y$。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
Weight 0:
   data_type  : int
   granularity: per-tensor or per-channel
Bias 0:
   data_type  : int
   granularity: per-tensor or per-channel
restriction: bias_scale, bias_zero_point = input_scale * weight_scale, 0.
```

4.15 ConvTranspose2D

算子定义

ConvTranspose2D is really the transpose of convolution2D. Performs a filter on the input tensor (X) and produces the output tensor (Y).

量化方法

ConvTranspose2D 和 Convolution2D 一样都可以抽象为线性运算+激活函数，其量化方法参考 Convolution2D。

规格说明

略。参考 Convolution2D。

4.16 Cosine

算子定义

Calculates the cosine of the given input tensor by element-wise.

量化方法

对于$y = Cosine(x)$有：$(y_q + zp_y)/s_y = Cosine\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Cosine\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Cosine 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。同时需要注意的是，对 Cosine 函数有对称性 Cosine(x)=Cosine(-x), 故实际制作 lut 表可只对 x 的正数部分做表以节省 lut 空间。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.17 Crelu

算子定义

Performs a Relu operation on the input tensor (X) which selects the positive part of the activation and negative part of the activation respectively, then concatenate them as an output tensor (Y) along with the specified axis. That is,
Crelu(x) = Concat [Relu(x), Relu(-x)].

量化方法

由$y_f = Concat\left(Relu(x_f),\ Relu(-x_f)\right)$可得：

$$(y_q + zp_y)/s_y = Concat\left(Relu\left((x_q + zp_x)/s_x\right),\ Relu\left(-(x_q + zp_x)/s_x\right)\right)$$
$$y_q = Concat\left(Relu\left((x_q + zp_x)/s_x\right)s_y,\ Relu\left(-(x_q + zp_x)/s_x\right)s_y\right) - zp_y$$
$$y_q = Concat\left(Relu\left((x_q + zp_x)s_y/s_x\right),\ Relu\left(-(x_q + zp_x)s_y/s_x\right)\right) - zp_y$$

可离线将s_y/s_x表示为规范形式$M/2^N$(M, N均为整数)，即有：

$$y_q = Concat\left(Relu\left((x_q + zp_x)M/2^N\right),\ Relu\left(-(x_q + zp_x)M/2^N\right)\right) - zp_y$$

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.18 Crop

算子定义

Crops the input tensor into the output tensor with the shape computing by the given beginning and end index (exclusive the ending indices) and the shape of the output tensor.

量化方法

由$y_f = Crop(x_f)$可得$(y_q + zp_y)/s_y = Crop\left((x_q + zp_x)/s_x\right)$，从而$y_q = (Crop(x_q) + zp_x)s_y/s_x - zp_y$。令$s_y$等于$s_x$，$zp_y$等于$zp_x$，则整数域上的 Crop 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.19 CTCGreedyDecoder

算子定义

Performs greedy decoding on the logits given in the input tensor, including the score tensor and sequence length tensor.

量化方法

由算子定义可知$(y_q + zp_y)/s_y = CTCGreedyDecoder\left((x_q + zp_x)/s_x\right)$，从而$y_q = (CTCGreedyDecoder(x_q) + zp_x)s_y/s_x - zp_y$。令$s_y$等于$s_x$，$zp_y$等于$zp_x$，则整数域上的 CTCGreedyDecoder 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.20 DataStride

算子定义

Performs an input data tensor (X) to produce a output tensor (Y) with a stride combination, which means taking kernel size numbers of data after every stride operation and gathering all these data along each axis respectively.

量化方法

由算子定义可知$(y_q + zp_y)/s_y = DataStride\left((x_q + zp_x)/s_x\right)$, 从而$y_q = \left(DataStride(x_q) + zp_x\right)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 DataStride 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.21 DepthToSpace

算子定义

DepthToSpace permutes data from depth into blocks of spatial data. This is the reverse transformation of SpaceToDepth. More specially, this operator outputs a copy of the input tensor where values from the depth dimension are moved in spatial blocks to the 'height' and 'width' dimensions.

量化方法

由算子定义可知$(y_q + zp_y)/s_y = DepthToSpace\left((x_q + zp_x)/s_x\right)$, 从而$y_q = \left(DepthToSpace(x_q) + zp_x\right)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 DepthToSpace 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.22 DepthwiseConvolution

算子定义

Performs a different filter to each channel of input tensor (X) and then concatenates the results of convolution together.

量化方法

略。参考 Convolution2D，需要说明的是由于 DepthwiseConvolution 的 Weight 在不同 Channel 上分布差异很大，一般默认采用 per-channel 量化方式以保证量化精度。

规格说明

略。参考 Convolution2D。

4.23 Div

算子定义

Performs element-wise division (with Numpy-style broadcasting support).

量化方法

由$y_f = a_f/b_f$可知$(y_q + zp_y)/s_y = \left((a_q + zp_a)/s_a\right) / \left((b_q + zp_b)/s_b\right)$，从而$y_q = \left((a_q + zp_a)/(b_q + zp_b)\right)(s_y s_b/s_a) - zp_y$。可离线将$s_y s_b/s_a$表示为规范形式$M/2^N (M, N$均为整数)，则有$y_q = \left((a_q + zp_a)M/(b_q + zp_b)\right)/2^N - zp_y$，计算$\left((a_q + zp_a)M\right)/(b_q + zp_b)$可以用整除（进一步可以得到余数，并据此对商做四舍五入以减少误差）。

为进一步减少计算，也可以用 lut 查表替代上述的整除运算，即

$$y_q = \left((a_q + zp_a)M/(b_q + zp_b)\right)/2^N - zp_y$$
$$y_q = \left((a_q + zp_a)M\left(2^{N_b}/(b_q + zp_b)\right)\right)/2^{N+N_b} - zp_y$$
$$y_q \approx (a_q + zp_a)\ M\ \widehat{lut(b_q)}\ /\ 2^{N'} - zp_y$$

其中$\widehat{lut(b_q)} = round\left(2^{N_b}/(b_q + zp_b)\right)$，$N' = N + N_b$且$N', N, N_b$均为整数。同时注意到$\frac{1}{x} = -\frac{1}{-x}$，因此实际制作 lut 表时可只对正数部分做表以节约 lut 空间。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.24 ElementwiseAdd

算子定义

Performs elementwise addition of each of the input tensors (with Numpy-style broadcasting support).

量化方法

由$y_f = a_f + b_f$可知$(y_q + zp_y)/s_y = (a_q + zp_a)/s_a + (b_q + zp_b)/s_b$，从而$y_q = \left((a_q + zp_a)s_b + (b_q + zp_b)s_a\right)s_y/s_a s_b - zp_y$。可离线将$s_a, s_b, s_y/s_a s_b$分别表示为规范形式$M_1/2^{N_1}, M_2/2^{N_2}, M_3/2^{N_3}$并且假定$N_1 >= N_2$，则有：

$$y_q = \left((a_q + zp_a)M_2 2^{N_1-N_2} + (b_q + zp_b)M_1\right) M3/2^{N_1+N_3} - zp_y$$
$$y_q = \left((a_q + zp_a)M_2 M_3 2^{N_1-N_2} + (b_q + zp_b)M_1 M_3\right)/2^{N_1+N_3} - zp_y$$
$$y_q = \left((a_q + zp_a)M' + (b_q + zp_b)M''\right)/2^{N'} - zp_y$$

其中$M' = M_2 M_3 2^{N_1-N_2}, M'' = M_1 M_3, N' = N_1 + N_3$且$M_1, M_2, M_3, M', M'', N_1, N_2, N_3, N'$均为整数。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.25 ElementwiseMax

算子定义

Performs elementwise maximum of each of the input tensors (with Numpy-style broadcasting support).

量化方法

由$y_f = maximum(a_f, b_f)$可得$(y_q + zp_y)/s_y = maximum\left((a_q + zp_a)/s_a,\ (b_q + zp_b)/s_b\right)$，即：

$$y_q = maximum\left((a_q + zp_a)s_y/s_a,\ (b_q + zp_b)s_y/s_b\right) - zp_y$$

可离线将$s_y/s_a, s_y/s_b$表示为规范形式$M_1/2^{N_1}, M_2/2^{N_2}$，且假定$N_1 >= N_2$，则：

$$y_q = maximum\left((a_q + zp_a)M_1/2^{N_1},\ (b_q + zp_b)M_2/2^{N_2}\right) - zp_y$$
$$y_q = maximum\left((a_q + zp_a)M_1,\ (b_q + zp_b)M_2 2^{N_1-N_2}\right)\ 2^{N_1} - zp_y$$
$$y_q = maximum\left((a_q + zp_a)M_1,\ (b_q + zp_b)M_3\right)\ 2^{N_1} - zp_y$$

其中$M_3 = M_2 2^{N_1-N_2}$且M_1, M_2, M_3, N_1, N_2均为整数。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.26 ElementwiseMin

算子定义

Performs elementwise minimum of each of the input tensors (with Numpy-style broadcasting support).

量化方法

略。参见 ElementwiseMax。

规格说明

略。参见 ElementwiseMax。

4.27 ElementwiseMul

算子定义

Performs element-wise multiplication of the input tensors.

量化方法

由$c_f = a_f b_f$可得$(c_q + zp_c)/s_c = \left((a_q + zp_a)/s_a\right)\left((b_q + zp_b)/s_b\right)$，从而$c_q = (a_q + zp_a)(b_q + zp_b)(s_c/(s_a s_b)) - zp_c$。其中$s_c/(s_a s_b)$可在离线量化过程中近似转换为$M/2^N$的规范形式（M, N 均为整数），则有：$c_q \approx (a_q + zp_a)(b_q + zp_b)(M/2^N) - zp_c$。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.28 ElementwiseSub

算子定义

Performs elementwise subtraction of each of the input tensors (with Numpy-stype broadcasting support).

量化方法

略。参见 ElementwiseAdd。

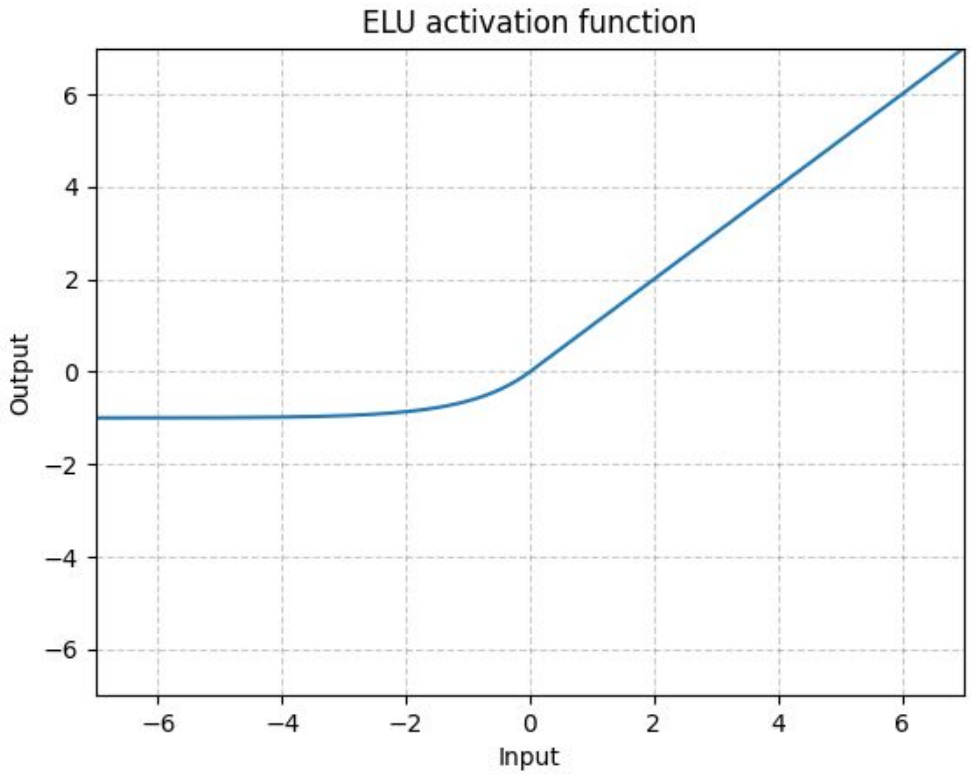
规格说明

略。参见 ElementwiseAdd。

4.29 Elu

算子定义

Takes an input tensor (X) and produces one output tensor (Y). Where, the function `f(x) = alpha * (exp(x) -1) for x < 0, f(x) = x for x >= 0`, is applied to the tensor elementwise.



量化方法

对于$y = Elu(x)$有：$(y_q + zp_y)/s_y = Elu\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Elu\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Elu 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。同时需要注意的是 Elu 函数沿 x 负半轴缓慢下降趋于平坦，若输入x的 min 取值过小则会导致量化后的值大量趋同无法区分（即量化后表达精度不够），影响整体算子精度，也浪费了 lut 空间，故在调节 Elu 算子精度时需重点调节x的 min 取值。另外，由于$x >= 0$时 Elu 其实是恒等函数，故为节约 lut 空间，还可以只对$x < 0$的情况做 lut 表。

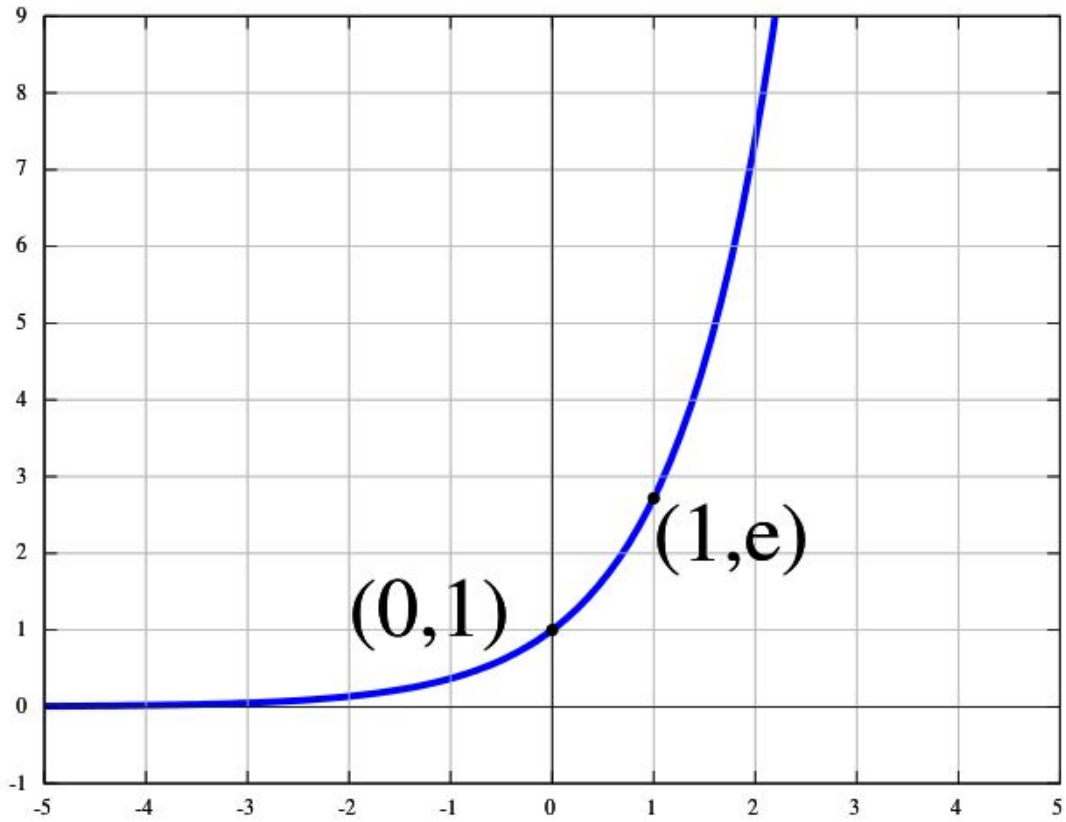
规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.30 Exp

算子定义

Calculates the exponential of the given input tensor (X), where the function f(x) = exp(x), is applied to the tensor elementwise.



量化方法

对于$y = Exp(x)$有：$(y_q + zp_y)/s_y = Exp\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Exp\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$

- 对$\hat{x_f}$计算 Exp 函数值获得$\hat{y_f}$
- 量化$\hat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。同时需要注意的是 Exp 函数：沿 x 负半轴缓慢下降趋于平坦，若输入x的 min 取值过小则会导致量化后的值大量趋同无法区分（即量化后表达精度不够）；沿 x 正半轴快速上升趋于无穷大，若输入x的 max 取值过大则会导致量化后的值过大从而要求其输出y用较大的位宽来量化以保证精度。故在调节 Exp 算子量化精度时需适当考虑提升该层以及后续关联层的量化位宽。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.31 Filter

算子定义

Takes one input data tensor (X) and filter tensor (B) to return an output tensor (Y) after filtration. Where the element will be filtered with accordingly zero selector in filter tensor B, otherwise remaining. Here, the filter tensor is a vector which composed of `0` or `1` with a same length as the specified axis in data tensor X. Similarity, multi input tensors (X[i]) will return multi outputs after the same filtration.

量化方法

由$y_f = Filter(x_f, B)$可得$(y_q + zp_y)/s_y = Filter\left((x_q + zp_x)/s_x, B\right)$, 从而$y_q = \left(Filter(x_q, B) + zp_x\right)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 Filter 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1 (the filter tensor):
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.32 FullyConnected

算子定义

Takes input tensor (X) and computes the class scores and outputs the 1-D array of size equal to the number of classes. In other words, it performs a 'dot(input_tensor, kernel)' operation with a bias addition, then outputs the tensor after the activation function.

量化方法

由算子定义，可知 FullyConnected 等效于特殊的卷积操作，其量化方法参见 Convolution2D。

规格说明

略。参见 Convolution2D。

4.33 Gather

算子定义

Given parameters and indices tensor, gathers entries of the parameters tensor indexed by indices and concatenates them in an output tensor. The indices are a tensor of integer, and there will be an error if any of the index values are out of bounds. Generally, a negative value in the indices tensor means counting the index from the back.

量化方法

由$y_f = Gather(x_f, Indices)$可得$(y_q + zp_y)/s_y = Gather\left((x_q + zp_x)/s_x, Indices\right)$,
从而$y_q = \left(Gather(x_q, Indices) + zp_x\right)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 Filter 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1 (Indices):
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.34 GatherND

算子定义

Given the input indices tensor, then gather slices from input tensor X into the output tensor Y with shape specified by indices. The output tensor has the shape `indices.shape[ :-1] + X.shape[indices.shape[-1] + batch_dims: ]` with the left closed and right open interval. Generally, `indices[-1] + batch_dims` should be less than or equal to `rank(X)`. Note that the negative value in indices tensor means counting the index from the back.

量化方法

略。参见 Gather。

规格说明

略。参见 Gather。

4.35 GroupConvolution

算子定义

Performs a filter on the input tensor (X) and produces the output tensor (Y). If bias is necessary, a bias vector will be added to the output tensor. Similarly, if the activation function is necessary, it is applied to the output tensor as well.

量化方法

由算子定义，可知 GroupConvolution 等效于特殊的卷积操作，其量化方法参见 Convolution2D。

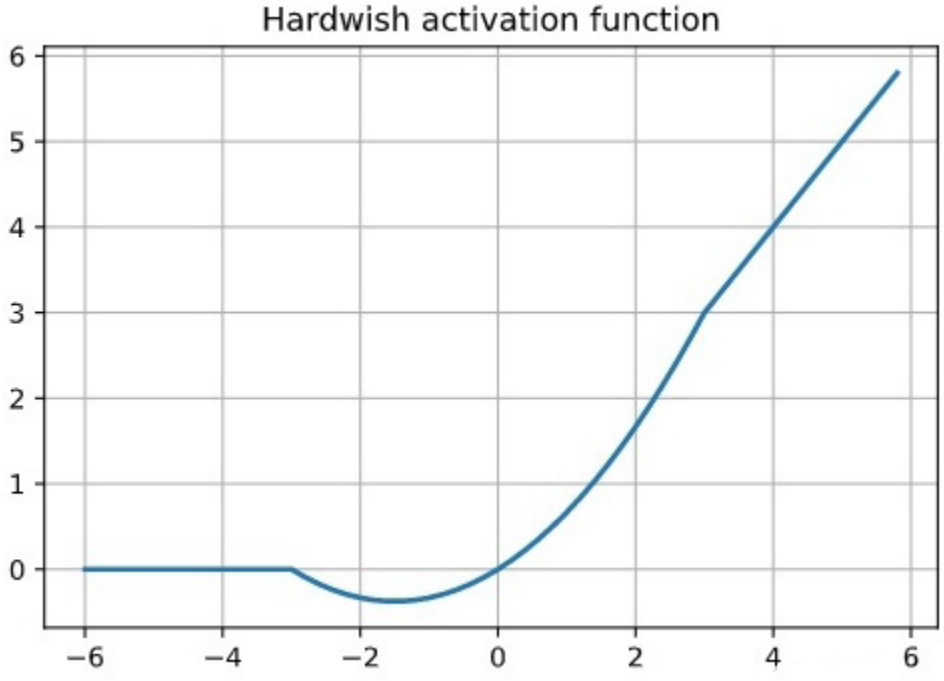
规格说明

略。参见 Convolution2D，需要说明的是由于 GroupConvolution 的 Weight 在不同 Group 上分布差异很大，建议采用 per-channel 量化方式以保证量化精度（尤其当 Group 较多时）。

4.36 HardSwish

算子定义

Applies the hardswish function element-wise. That is, `HardSwish(x) = 0 if x <= -3; HardSwish(x) = x if x >= 3; HardSwish(x) = (x * (x + 3)) /6 if -3 < x <3`. Also, the formula can be expressed as HardSwish(x) = (x * Relu6(x + 3)) / 6.



量化方法

对于$y = HardSwish(x)$有：$\left(y_q + zp_y\right)/s_y = HardSwish\left(\left(x_q + zp_x\right)/s_x\right)$,
从而：$y_q = HardSwish\left(\left(x_q + zp_x\right)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 HardSwish 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.37 Histogram

算子定义

Applies to the input numeric tensor X and returns a 1-D histogram to count the number of entries in numeric tensor that fell into every bin. Generally, the bins are equal width and determined by the parameters `min`, `max` and `nbins`.

量化方法

根据定义，容易得出，对于$y_f = Histogram(x_f, a_f, b_f, n)$其中$a, b, n$代表 `min`, `max` 和 `nbins`，有$y_q = Histogram(x_q, round(a_f * s_x - zp_x), round(b_f * s_x - zp_x), n)$。即仅需要将直方图统计的上下界阈值按输入张量的量化系数离线量化即可，前向运算过程和浮点域上的计算方式一样，无需多余的量化相关运算。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.38 InTopK

算子定义

Performs the input tensor (X) with `bach_size` classes to check whether the targets are in the top k predictions, and outputs a `batch_size` bool array. That is, the output Y[i] returns `true` if the predication for the target class exists among the top k predications,
Y[i] = `true` if predications[`target_classes_id (Si)`] ϵ {top-k predications}.

量化方法

根据定义，容易得出，对于$y_f = InTopK(x1_f, x2_f)$有$y_q = InTopK(x1_q, x2_q)$。即前向运算过程和浮点域上的计算方式一样，无需多余的量化相关运算。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.39 LeakyRelu

算子定义

Takes one input tensor (X) and produces one output tensor (Y), where the rectified linear function `y = alpha * x for x < 0, y = x for x >= 0`, is applied to the tensor elementwise.

量化方法

由定义有$(y_q + zp_y)/s_y = maximum\left((x_q + zp_x)/s_x, 0\right) + minimum\left((x_q + zp_x)/s_x, 0\right)(\alpha_q + zp_\alpha)/s_\alpha$，即：
$y_q = maximum\left((x_q + zp_x)s_y/s_x, 0\right) + minimum\left((x_q + zp_x)s_y/s_x, 0\right)(\alpha_q + zp_\alpha)/s_\alpha - zp_y$
取$s_\alpha = 2^{N_\alpha}, zp_\alpha = 0$，并离线将$s_y/s_x$表示为规范形式$M/2^N$（其中$N_\alpha, M, N$均为整数），则：
$y_q = maximum\left((x_q + zp_x)M/2^N, 0\right) + minimum\left((x_q + zp_x)M/2^N, 0\right)\alpha_q/2^{N_\alpha} - zp_y$

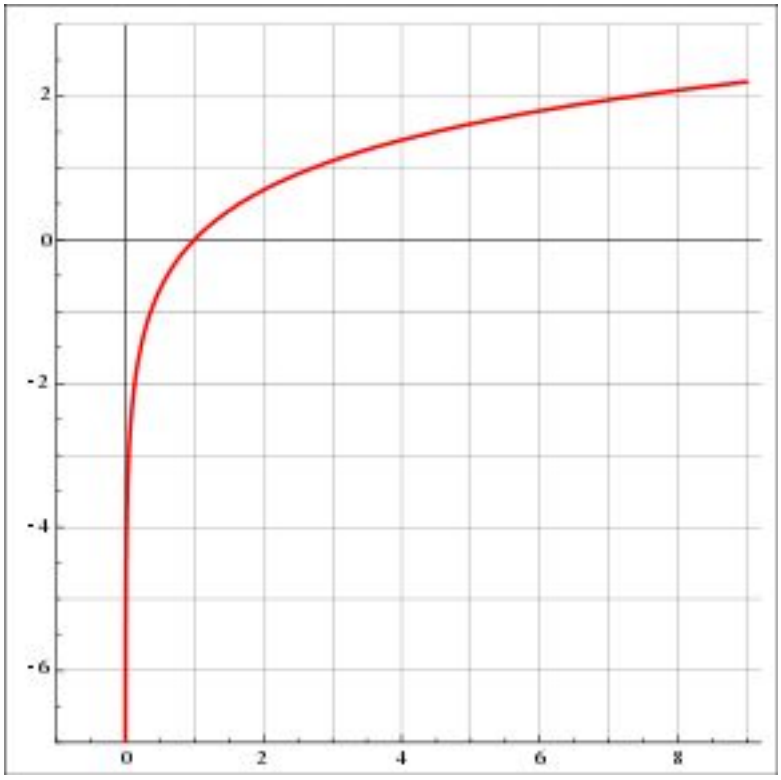
规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.40 Log

算子定义

Calculates the natural log of the given input tensor (X), where the function $f(x) = log_e(x)$, is applied to the tensor elementwise.



量化方法

对于$y = Log(x)$有：$\left(y_q + zp_y\right)/s_y = Log\left(\left(x_q + zp_x\right)/s_x\right)$,
从而：$y_q = Log\left(\left(x_q + zp_x\right)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Log 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。

规格说明

```
Input 0:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.41 Logical

算子定义

Returns the tensor resulted from performing logical operation elementwise on the input tensor. It supports multidirectional broadcasting as Numpy-style and returns a tensor of Boolean values.

量化方法

对于数值比较的逻辑操作：EQUAL, NOT_EQUAL, GREATER, GREATER_EQUAL, LESS, LESS_EQUAL。可抽象为$y_f = L(a_f, b_f)$，即有：

$$(y_q + zp_y)/s_y = L\left((a_q + zp_a)/s_a,\ (b_q + zp_b)/s_b\right)s_y - zp_y$$
$$y_q = L\left((a_q + zp_a)/s_a,\ (b_q + zp_b)s_a/s_b s_a\right)s_y - zp_y$$
$$y_q = L\left((a_q + zp_a),\ (b_q + zp_b)s_a/s_b\right)s_y - zp_y$$

由于y_f, y_q均为布尔类型，故$s_y = 1, zp_y = 0$，同时可离线将s_a/s_b表示为规范形式$M/2^N$（M, N均为整数），则有：

$$y_q = L\left((a_q + zp_a),\ (b_q + zp_b)M/2^N\right)$$
$$y_q = L\left((a_q + zp_a)2^N,\ (b_q + zp_b)M\right)$$

对于逻辑运算的操作：OR, AND, XOR, NOT. 其输入输出均为布尔类型，运算过程与浮点域上一致，无需多余的量化操作。

规格说明

```
Input 0, 1:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.42 MatMul

算子定义

Matrix product of two input tensors (X, Y) as Numpy-style. If either argument is N-D, N>2, it is treated as a stack of matrices residing in the last two indexes and broadcast accordingly.

量化方法

对于$Z_f = MatMul(X_f, Y_f)$有$(Z_q + zp_Z)/s_Z = MatMul\left((X_q + zp_X)/s_X, (Y_q + zp_Y)/s_Y\right)$，即：

$$Z_q = MatMul\left((X_q + zp_X)/s_X, (Y_q + zp_Y)/s_Y\right)s_Z - zp_Z$$
$$Z_q = MatMul\left((X_q + zp_X), (Y_q + zp_Y)\right)s_Z/s_X s_Y - zp_Z$$

可离线将$s_Z/s_X s_Y$表示为规范形式$M/2^N$（M, N均为整数），则：

$$Z_q = MatMul\left((X_q + zp_X), (Y_q + zp_Y)\right)M/2^N - zp_Z$$

规格说明

```
Input 0:
   data_type  : int or uint
```

```
   granularity: per-tensor
Input 1:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.43 MaxPooling2D

算子定义

MaxPooling-2D takes an input tensor X and applies max pooling across the tensor by the kernel size, stride size, and padding.

量化方法

由$y_f = MaxPooling2D(x_f)$可得$(y_q + zp_y)/s_y = MaxPooling2D\left((x_q + zp_x)/s_x\right)$,
从而$y_q = \left(MaxPooling2D(x_q) + zp_x\right)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.44 MaxPoolingWithArgMax

算子定义

Performs max pooling on the inputs and returns both max values and indices.

量化方法

略。参见 MaxPooling2D。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
Output 1:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.45 MaxRoiPool

算子定义

Region of interest (ROI) pooling is used for converting all the proposals to fixed shape as required by the next special layers, especially in object detection network. That is, ROI pooling produces the fixed size of feature maps from non-uniform inputs by commonly doing max-pooling on the input tensors. Generally, the number of output channels is equal to the number of input channels for this layer. And it takes two inputs, one is feature map obtained from a Convolutional Neural Network after multiple convolutions and pooling layers; the other is `num_rois` proposal or ROIs from Region proposal network (so-called bounding box). Each proposal has five values, the first one indicating the `batch_indices` and the rest of the four are proposal coordinates in original image. The four coordinates indicate the top-left and bottom-right corner coordinates of the proposal and the shape will be [num_rois, 5]. Where the value 5 means [batch_indices, x1, y1, x2, y2].

量化方法

主体的 pool 计算部分与 MaxPooling2D 一致，无需多余的量化相关计算。仅注意到，浮点类型的 spatial_scale 参数（spatial_scale_x, spatial_scale_y）在整数域的前向计算时需要提前转换成整数形式，具体方法为对其放大整数倍变为$M/2^N$形式以保留一定精度的小数部分。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.46 Moments

算子定义

Computes the mean and variance of the input tensor (X) along the specified axis.

量化方法

对于 mean 的计算由$\bar{x} = \frac{1}{n}\sum x_i$，有$(\bar{x}_q + zp_{\bar{x}})/s_{\bar{x}} = \frac{1}{n}\sum\left(x_{i_q} + zp_x\right)/s_x$，即$\bar{x}_q = \frac{1}{n}\sum\left(x_{i_q} + zp_x\right)s_{\bar{x}}/s_x - zp_{\bar{x}}$。取$s_{\bar{x}} = s_x$，$zp_{\bar{x}} = zp_x$则有$\bar{x}_q = \frac{1}{n}\sum x_{i_q}$。对于除以 n 既可以用整数域上的整除实现，也可以离线将$1/n$转换为规范形式$M/2^N$（其中M, N均为整数）。

对于 variance 的计算由$\delta = \overline{x^2} - \overline{x}^2 = \frac{\sum {x_i}^2}{n} - (\frac{\sum x_i}{n})^2$，有$(\delta_q + zp_\delta)/s_\delta = \frac{\sum((x_{i_q} + zp_x)/s_x)^2}{n} - (\frac{\sum(x_{i_q} + zp_x)/s_x}{n})^2$，即：
$\delta_q = (\frac{\sum(x_{i_q} + zp_x)^2}{n} - (\frac{\sum(x_{i_q} + zp_x)}{n})^2){s_\delta}/{s_x}^2 - zp_\delta = (\frac{\sum {x_{i_q}}^2}{n} - (\frac{\sum x_{i_q}}{n})^2){s_\delta}/{s_x}^2 - zp_\delta$
取$s_\delta = {s_x}^2, zp_\delta = 0$, 则有：

$\delta_q = \frac{\sum {x_{i_q}}^2}{n} - (\frac{\sum x_{i_q}}{n})^2$

对于无偏修正的情况，同样易得：

$\delta_q = (\frac{\sum {x_{i_q}}^2}{n} - (\frac{\sum x_{i_q}}{n})^2)\frac{n}{n-1}$

对于$\frac{n}{n-1}$部分的计算可以先乘以n再整除$n-1$, 或者离线将$\frac{n}{n-1}$转换为规范形式$M_u/2^{N_u}$（其中M_u, N_u均为整数）。

需要注意的是，若不取$s_\delta = {s_x}^2$，只需离线将$s_\delta/{s_x}^2$转换为规范形式$M_\delta/2^{N_\delta}$（其中M_δ, N_δ均为整数）。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
Output 1:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.47 Negative

算子定义

Computes numerical negative value to the input tensor X by elements-wise and returns the output tensor Y. The equation is Y(i) = -X(i), where 'I' means the index.

量化方法

由$y_f = -x_f$，有$(y_q + zp_y)/s_y = -(x_q + zp_x)/s_x$，即$y_q = -(x_q + zp_x)s_y/s_x - zp_y$. 取$s_y = s_x, zp_y = -zp_x$, 即有$y_q = -x_q$, 整数域上前向计算与浮点域上一致，无需多余量化相关计算。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.48 NMS

算子定义

Non-maximum suppression (NMS) filters out boxes that have high intersection-over-union (IoU) overlap with previously selected boxes. Bounding boxes with score less than Score_Threshold are dropped or discarded in the last output. Bounding box are supplied as [y1, x1, y2, x2], where (y1, x1) and (y2, x2) are the coordinates of any diagonal pair of box corners and the coordinates can be provided as normalized (i.e., lying in the interval [0, 1]) or absolute. Note that this algorithm is agnostic to where the origin is in the coordinate system

and more generally is invariant to orthogonal transformations and translations of the coordinate system; so, translating or reflections of the coordinate system result in the same boxed being selected by the algorithm. The selected output is a set of integers indexing into the input collection of bounding boxes representing the selected boxes. The bounding box coordinates corresponding to the selected indices can then be obtain using the Gather operation.

Inputs:

- X1 means the coordinates of the maximum output boxes per batch;
- X2 means the boxes number of ervery classes per batch;
- X3 means the classes of every batch;
- X4 means the scores of all the boxes per batch;

Outputs:

- Y1 means the coordinates of the selected boxes per batch;
- Y2 means the boxes number of ervery classes per batch after selection;
- Y3 means the scores of all the selected boxes per batch;
- Y4 means the indices of the selected boxes per batch in input tensor X4.

量化方法

由算子定义可知，在整数域上对 box 的过滤操作，主体逻辑上与浮点域上一致，无需多余的量化计算，仅需注意到，对于浮点类型的 Score_Threshold 参数在整数域的前向计算时需要提前转换成整数形式，具体方法为对齐放大整数倍变为 $M/2^N$以保留一定精度的小数部分。

规格说明

```
Input 0:
   data_type  : uint
   granularity: per-tensor
Input 1:
   data_type  : uint
   granularity: per-tensor
Input 2:
   data_type  : uint
   granularity: per-tensor
Input 3:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
Output 1:
   data_type  : uint
   granularity: per-tensor
Output 2:
   data_type  : uint
   granularity: per-tensor
Output 3:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.49 OneHot

算子定义

Return a one-hot tensor (Y) based on the input `indice` tensor and input scalar `depth`. Generally, the index value in the "indices" input tensor means there is a "on_value" in that locations and the other will have a value of "off_value" in the output tensor.

量化方法

由算子定义有$(y_q + zp_y)/s_y = OneHot(x_q, d_q, v_{on}, v_{off})$，其中$x_q = x_f, s_x = 1, zp_x = 0$为 `indice`, $d_q = d_f, s_d = 1, zp_d = 0$为 `depth`, v_{on}, v_{off}分别为 `on_value`, `off_value`。从而：

$$y_q = OneHot(x_q, d_q, v_{on}, v_{off})\ s_y - zp_y$$
$$y_q = OneHot(x_q, d_q, v_{on}s_y - zp_y, v_{off}s_y - zp_y)$$

即仅需离线将 `on_value`, `off_value` 用 Y 的量化系数量化，前向计算时无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.50 Pad

算子定义

Given the input tensor containing the data to be padded, a tensor containing the number of the start and end pad values for specified axis and mode.

量化方法

由$y_f = Pad(x_f)$可得$(y_q + zp_y)/s_y = Pad\left((x_q + zp_x)/s_x\right)$，从而：

$$y_q = Pad\left((x_q + zp_x)/s_x\right)s_y - zp_y$$
$$y_q = \left(Pad(x_q) + zp_x\right)s_y/s_x - zp_y$$

令$s_y = s_x, zp_y = zp_x$，即：

$$y_q = Pad(x_q)$$

前向计算时无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
```

```
   granularity: per-tensor
restriction: none
```

4.51 Pow

算子定义

Takes input data tensor (X) and exponent Tensor (Y), then produces an output tensor (Z) where the function z = x ^exponent (that is, exponent is from exponent tensor Y), is applied to the data tensor elementwise.

量化方法

若 exponent 是一个离线可知的常数α，则可直接用 lut 查表，离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算$Pow(x_f, \alpha)$获得$\widehat{z_f}$
- 量化$\widehat{z_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。

若 exponent 是变量 y，则有：

$$z_f = {x_f}^{y_f} = \left(Sign(x_f)\right)^{y_f}\ \left(Abs(x_f)\right)^{y_f} = \left(Sign(x_f)\right)^{y_f}\ Exp\left(Log\left(\left(Abs(x_f)\right)^{y_f}\right)\right)$$
$$= \left(Sign(x_f)\right)^{y_f}\ Exp\left(y_f\ Log\left(Abs(x_f)\right)\right)$$

对于$u_f = \left(Sign(x_f)\right)^{y_f}$有$(u_q + zp_u)/s_u = \left(Sign\left((x_q + zp_x)/s_x\right)\right)^{y_f}$

即$u_q = \left(Sign\left((x_q + zp_x)/s_x\right)\right)^{(y_q+zp_y)/s_y} s_u - zp_u = \left(Sign(x_q + zp_x)\right)^{(y_q+zp_y)/s_y} s_u - zp_u$

令$s_u = 1, zp_u = 0$以简化计算，即$u_q = \left(Sign(x_q + zp_x)\right)^{(y_q+zp_y)/s_y}$

对于$L_f = Log\left(Abs(x_f)\right)$有$(L_q + zp_L)/s_L = Log\left(Abs\left((x_q + zp_x)/s_x\right)\right)$

即$L_q = Log\left(Abs\left((x_q + zp_x)/s_x\right)\right) s_L - zp_L$，对$L_q$可通过 lut 查表，离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算$Log\left(Abs(x_f)\right)$获得$\widehat{L_f}$
- 量化$\widehat{L_f}$获得最终的 lut 表

对于$v_f = y_f\ L_f$有$(v_q + zp_v)/s_v = \left((y_q + zp_y)/s_y\right)\left((L_q + zp_L)/s_L\right)$，即$v_q = (y_q + zp_y)(L_q + zp_L)s_v/s_y s_L - zp_v$。离线将$s_v/s_y s_L$表示为规范形式$M/2^N$，其中$M, N$均为整数，则$v_q = (y_q + zp_y)(L_q + zp_L)M/2^N - zp_v$。

如此，则$(z_q + zp_z)/s_z = (u_q + zp_u)/s_u\ Exp\left((v_q + zp_v)/s_v\right)$，即$z_q = u_q\ Exp\left((v_q + zp_v)/s_v\right)s_u - zp_u$，对 Exp 的计算再次通过离线做 lut 查找表：

- 根据v_q的数据类型枚举得到值域集合$\widehat{v_q}$（若v_q范围较大，可采样获得一个子集）

- 对$\widehat{v_q}$反量化获得$\widehat{v_f}$
- 对$\widehat{v_f}$计算$Exp(v_f)$获得$\widehat{E_f}$
- 量化$\widehat{E_f}$获得最终的 lut 表

需要注意的是两次 lut 查表会一定程度增加量化的精度损失，因此，可适当提升中间变量L_q的量化位宽以提升整体精度。

最后即有：$z_q = \left(Sign(x_q + zp_x)\right)^{(y_q+zp_y)/s_y}\ lut_{exp}\left((y_q + zp_y)(lut_{log}(x_q) + zp_L)M/2^N - zp_v\right)$。
当y_f在浮点域上即为整数型时可令$s_y = 1, zp_y = 0$，从而
$z_q = \left(Sign(x_q + zp_x)\right)^{y_q}\ lut_{exp}\left((y_q + zp_y)(lut_{log}(x_q) + zp_L)M/2^N - zp_v\right)$。
当y_f在浮点域上带小数时，则要求x_f不能为负数（否则会产生复数结果，当前无法表示支持），那么

$$z_q = lut_{exp}\left((y_q + zp_y)(lut_{log}(x_q) + zp_L)M/2^N - zp_v\right) \quad if \quad x_q + zp_x > 0 \quad else \quad 0$$

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: input0 must be nonnegative if the input1 is float type.
```

4.52 Prelu

算子定义

Takes one input tensor (X) and produces one output tensor (Y), where the rectified linear function y = alpha * x for x < 0, y = x for x >= 0, is applied to the tensor elementwise. But the difference with RELU Operation is that 'alpha' is a learned array with the same shape as the input tensor (X).

量化方法

略。参见 LeakyRelu。需要说明的是当 alpha 为向量时（维度同 input channels），可对 alpha 采用 per-channel 量化以进一步提升精度。

规格说明

略。参见 LeakyRelu。

4.53 Reduce

算子定义

Reduces the input tensor's element along the provided axis. The reduce method, where MIN computes the minimum value of the input tensor's element along the axis. MAX computes the maximum value of the input tensor's element along the axis. ANY computes the 'Logical OR' value of the input tensor's element along the

axis. ALL computes the "Logical AND" value of the input tensor's element along the axis. SUM computes the sum value of the input tensor's element along the axis. MEAN computes the mean value of the input tensor's element along the axis.

量化方法

由$y_f = Reduce(x_f)$，有$(y_q + zp_y)/s_y = Reduce\left((x_q + zp_x)/s_x\right)$，即$y_q = Reduce\left((x_q + zp_x)/s_x\right)s_y - zp_y$。

对于'MIN, MAX, MEAN', $y_q = (Reduce(x_q) + zp_x))s_y/s_x - zp_y$, 令$s_y = s_x, zp_y = zp_x$；对于'ANY, ALL', 输入输出均为布尔类型，令$s_y = 1, zp_y = 0, s_x = 1, zp_x = 0$。则$y_q = Reduce(x_q)$，即整数域上的运算与浮点域上一致，无需多余的量化操作。

对于'SUM'，$y_q = Reduce(x_q + zp_x)s_y/s_x - zp_y$，可离线将$s_y/s_x$表示为规范形式$M/2^N$，其中$M, N$均为整数。则$y_q = Reduce(x_q + zp_x)M/2^N - zp_y$。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.54 Relu

算子定义

Takes one input tensor and produces one output tensor, where the rectified linear function y = max(0, x), is applied to the tensor elementwise.

量化方法

由算子定义$y_f = maximum(0, x_f)$，有$(y_q + zp_y)/s_y = maximum(0, (x_q + zp_x)/s_x)$，即$y_q = maximum\left(0, (x_q + zp_x)\right)s_y/s_x - zp_y$, 可离线将$s_y/s_x$表示为规范形式$M/2^N$，其中$M, N$均为整数，则$y_q = maximum\left(0, (x_q + zp_x)\right)M/2^N - zp_y$。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.55 Relu6

算子定义

Takes one input tensor and produces one output tensor, where the rectified linear function y = min(max(0, x), 6) = Clip(x, 0, 6), is applied to the tensor elementwise.

量化方法

略。参见 Clip。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.56 Repeat

算子定义

Repeat elements of input tensor X along the corresponding axis. Repeat tensor is a 1-D inter tensor which means the number of repetitions for each element.

量化方法

由$y_f = Repeat(x_f)$可得$(y_q + zp_y)/s_y = Repeat\left((x_q + zp_x)/s_x\right)$, 从而$y_q = \left(Repeat(x_q) + zp_x\right)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 Repeat 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.57 Reshape

算子定义

Returns a tensor that has the same value as the input tensor with shape changing.

量化方法

由$y_f = Reshape(x_f)$可得$(y_q + zp_y)/s_y = Reshape\left((x_q + zp_x)/s_x\right)$, 从而$y_q = (Reshape(x_q) + zp_x)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 Reshape 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.58 Resize

算子定义

Resizes the input tensor (X) using the specified method. The method to resize the input tensor: BILINEAR, NEAREST.

量化方法

由$y_f = Resize(x_f)$可得$(y_q + zp_y)/s_y = Resize\left((x_q + zp_x)/s_x\right)$, 从而$y_q = (Resize(x_q) + zp_x)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 Resize 运算方式可与浮点域上一致，无需多余的量化相关操作。

需要说明的是插值时候的缩放系数以及双线性插值时候的加权系数项为浮点数，在整数域前向计算时需先放大2^N整数倍以保留部分小数的精度，最后再除以2^N.

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.59 ReverseSequence

算子定义

Reverses batch of sequences having different lengths specified by 'sequence_lens'. For each slice 'i' iterating on the batch axis, the operator reverses the first 'sequence_lens[i] ' element to 'time_axis', and copies elements whose indexes are beyond 'sequence_lens[i] ' to the output. So, the output slice 'i' includes reverse sequences elements in the first 'sequence_lens[i] ' and original elements copied beyond the 'sequence_lens' index.

量化方法

由$y_f = ReverseSequence(x_f)$可得$(y_q + zp_y)/s_y = ReverseSequence\left((x_q + zp_x)/s_x\right)$，从而$y_q = (ReverseSequence(x_q) + zp_x)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x, 则整数域上的 ReverseSequence 运算方式可与浮点域上一致，无需多余的量化相关操作。

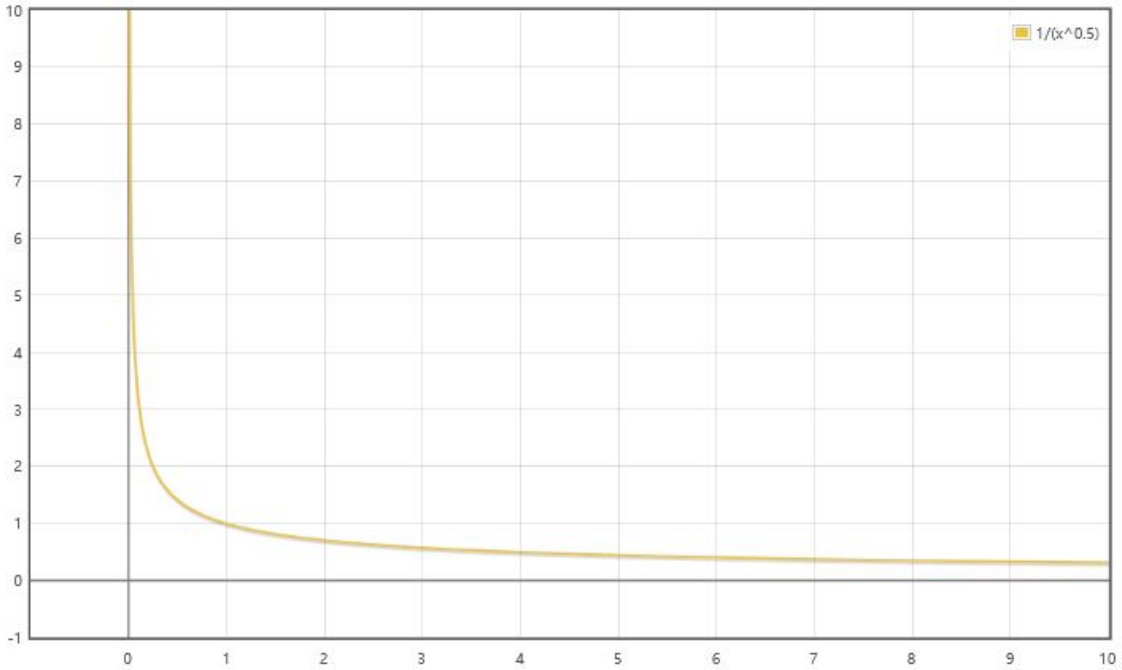
规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.60 Rsqrt

算子定义

Computes reciprocal of square root of the input tensor (X) element-wise as the output tensor (Y), where the equation, y = 1/ (x^0.5), is applied to the tensor elementwise.



量化方法

对于$y = Rsqrt(x)$有：$(y_q + zp_y)/s_y = Rsqrt\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Rsqrt\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widetilde{x_q}$（若x_q范围较大，可采样获得一个子集）

- 对$\hat{x_q}$反量化获得$\hat{x_f}$
- 对$\hat{x_f}$计算 Rsqrt 函数值获得$\hat{y_f}$
- 量化$\hat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。

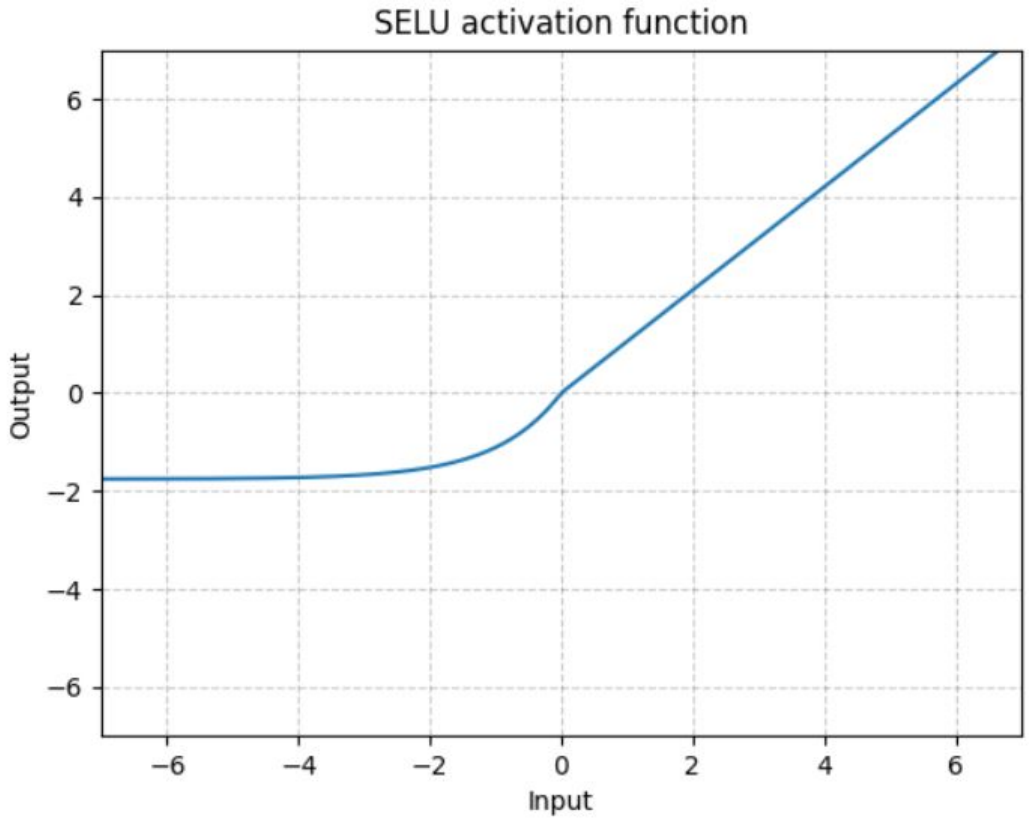
规格说明

```
Input 0:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.61 Selu

算子定义

Takes an input tensor (X) and prodeces one output tensor (Y), where the scaled exponential linear unit function `f(x) = gamma * (alpha * (exp(x) -1)) for x <= 0, f(x) = gamma * x for x > 0`, is applied to the tensor elementwise.



量化方法

对于$y = Selu(x)$有：$(y_q + zp_y)/s_y = Selu\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Selu\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Selu 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。需要注意的是 Selu 函数沿 x 负半轴缓慢下降趋于平坦，若输入x的 min 取值过小则会导致量化后的值大量趋同无法区分（即量化后表达精度不够），影响整体算子精度，也浪费了 lut 空间，故在调节 Selu 算子精度时需重点调节x的 min 取值。

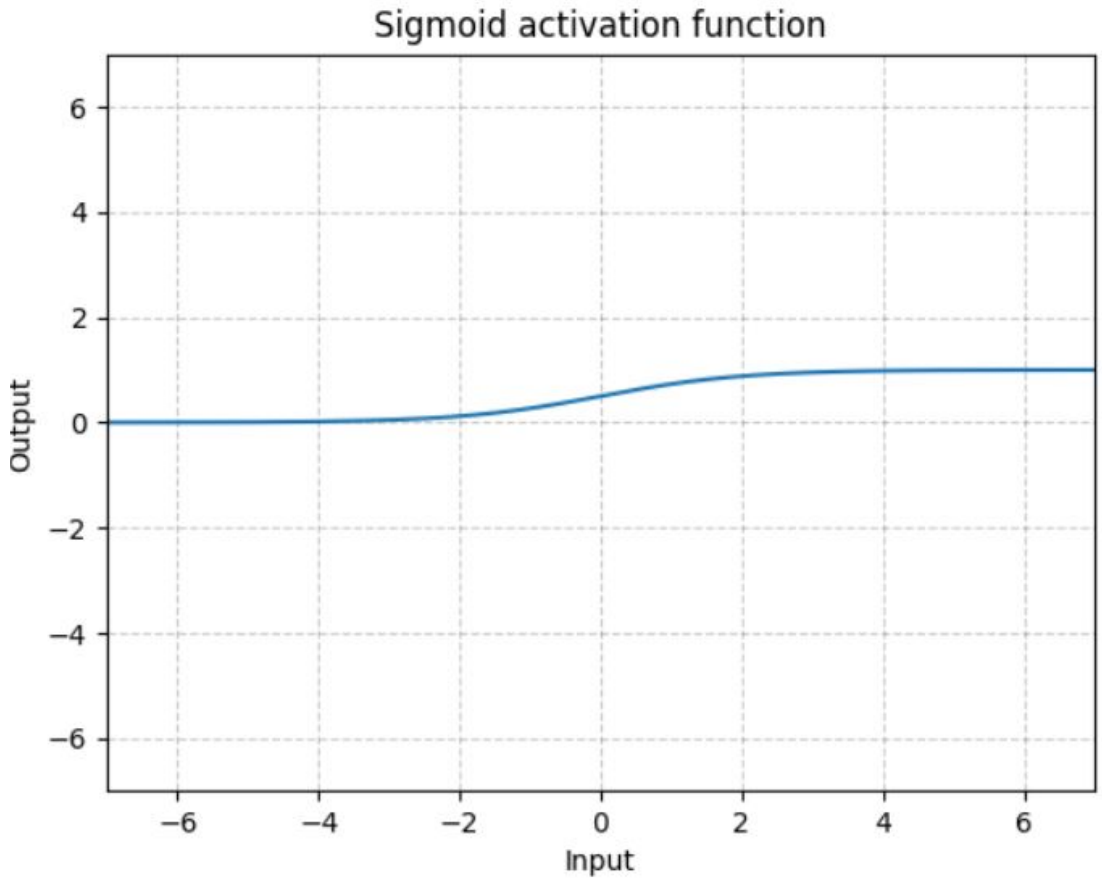
规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.62 Sigmoid

算子定义

Takes one input tensor and produces one output tensor, where the sigmoid function y = 1 / (1 + exp(-x)), is applied to the tensor elementwise.



量化方法

对于$y = Sigmoid(x)$有：$(y_q + zp_y)/s_y = Sigmoid\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Sigmoid\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Sigmoid 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。需要注意的是 Sigmoid 函数沿 x 正负半轴均缓慢趋于平坦，若输入x的 min 取值过小或 max 取值过大则会导致量化后的值大量趋同无法区分（即量化后表达精度不够），影响整体算子精度，调节精度时输入的 min,max 取值可于较近处截断（如经验值 6.0 附近）。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.63 Sine

算子定义

Calculates the sine of the given input tensor by element-wise.

量化方法

略。参见 Cosine。

规格说明

略。参见 Cosine。

4.64 Sign

算子定义

Calculates the sign of the given input tensor (X) element-wise. That is, `f(x) = 1 if x >0; f(x) = 0 if x == 0; f(x) = -1 if x < 0`.

量化方法

由$y_f = Sign(x_f)$，有$(y_q + zp_y)/s_y = Sign\left((x_q + zp_x)/s_x\right)$，即
$y_q = Sign\left((x_q + zp_x)/s_x\right)s_y - zp_y$，
$y_q = Sign(x_q + zp_x)s_y - zp_y$。
令$s_y = 1, zp_y = 0$，即有：$y_q = Sign(x_q + zp_x)$

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.65 Slice

算子定义

Produces a slice of the input tensor (X) along multiple axes.

量化方法

略。参见 Crop。

规格说明

略。参见 Crop。

4.66 Softmax

算子定义

Applies the Softmax function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range [0,1] and sum to 1.

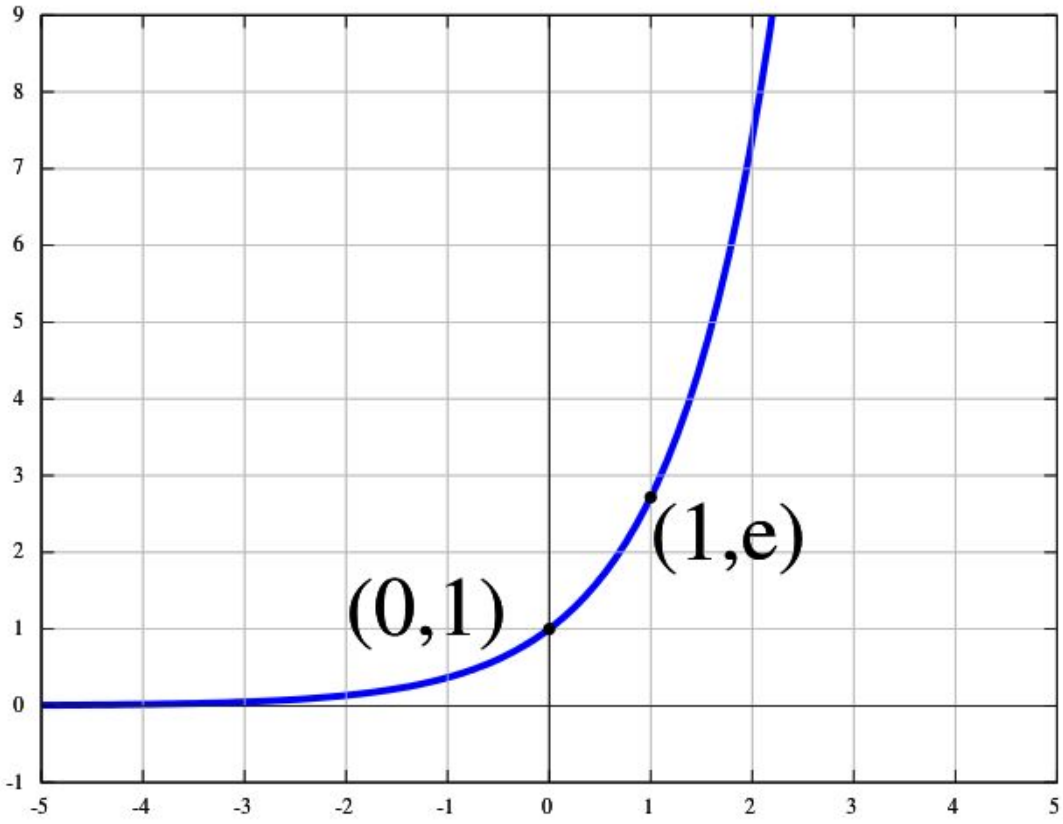
Softmax is defined as:

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_i}} \qquad (2)$$

量化方法

由$y_f = Softmax(x_f)$, 可得$(y_q + zp_y)/s_y = Softmax\left((x_q + zp_x)/s_x\right)$, 从而$y_q = \left(e^{(x_q+zp_x)/s_x} / \sum_j e^{(x_q+zp_x)/s_x}\right)s_y - zp_y$。

对于e^x计算，采用 LUT(lookup table)查表的方式计算，由于指数函数（见下图）对于x的负数值非常平坦（此类值量化后大量变成 0 或 1 导致精度丢失），对于x的正数值迅速攀升（LUT 不能存储过大的数），故限定 LUT 中存储的最大值恰好不得超过2^{20}（在使用 32bit 数做累加器时，则最多允许$2^{12} = 4096$个项求和），实现时通过对输入值做整体平移使得最大输入值接近log(2^{20})（易看出对于 softmax 函数，有 softmax(x)=softmax(x+a), 其中 a 为常量）。



LUT 表具体构建如下：

Step1：确定 LUT 允许的最大值 2^20 对应的输入值即 xm_f = log(2^20)。

Step2：确定 xm_f 对应（按 softmax 的输入 tensor 的量化参数）量化后的值 xm_q。

Step3：根据 LUT 表的大小 size 确定定义域 Xq。

Step4：将定义域 Xq（按 softmax 的输入 tensor 的量化参数）反量化得到 Xf。

Step5：对 Xf 中各项计算指数函数结果并取整，然后存入 LUT。

LUT 表离线构建完成后，量化 forward 在线运算时步骤如下：

Step1：对于输入值 Xq 查表得到其指数函数结果。

Step2：对指数函数结果做累加运算。

Step3：对 Softmax 公式中分子放大 M 倍（即乘以$s_y \approx M/2^N$中的 M, M,N 均为整数）。

Step4：进行 Softmax 公式中的除法计算（即整数域上的整除和求余，并根据余数对商做四舍五入）。

Step5：执行剩余的移位，减 zero point 运算（精度允许的条件下，可取$s_y = 2^B - 1, zp_y = 0$以省略本步骤，其中 B 为最终结果的量化比特数）。

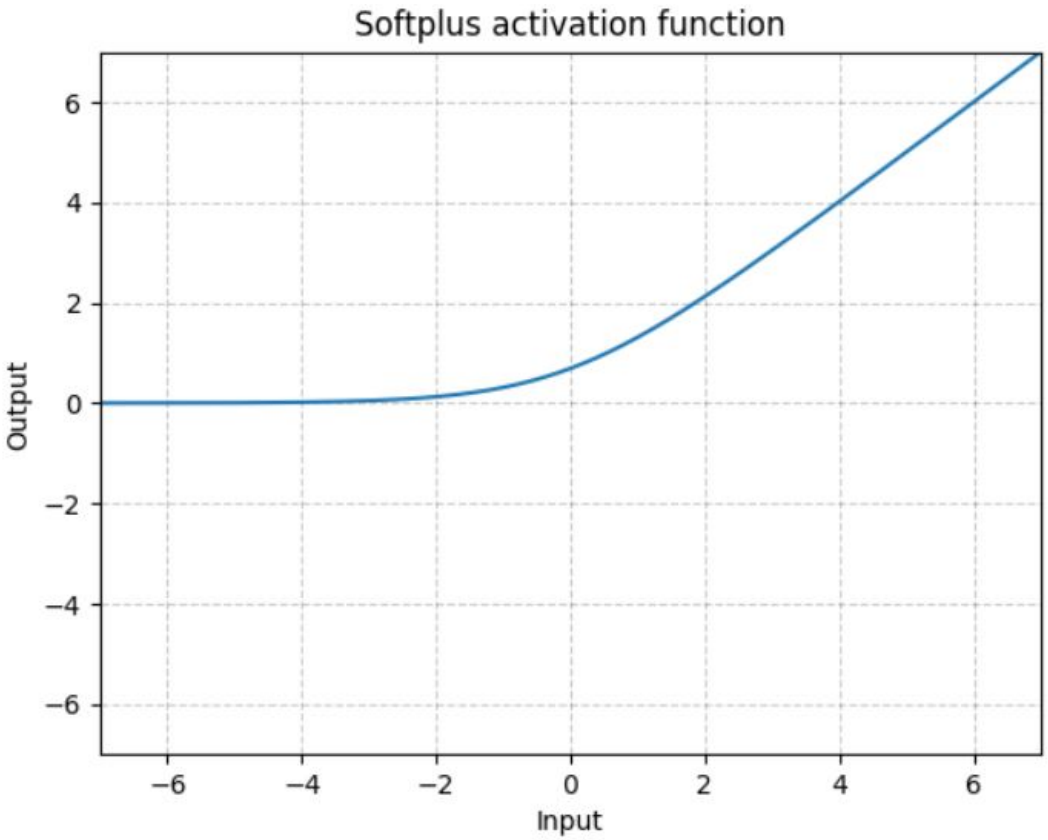
规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: only support j <= 4096 in softmax formula, due to numerical precision iss
ue
```

4.67 Softplus

算子定义

Takes one input tensor (X) and produces one output tensor (Y), where the function y = loge(exp(x) + 1), is applied to the tensor elementwise.



量化方法

对于$y = Softplus(x)$有：$\left(y_q + zp_y\right)/s_y = Softplus\left(\left(x_q + zp_x\right)/s_x\right)$,
从而：$y_q = Softplus\left(\left(x_q + zp_x\right)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Softplus 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。同时需要注意的是 Softplus 函数沿 x 负半轴缓慢下降趋于平坦，若输入x的 min 取值过小则会导致量化后的值大量趋同无法区分（即量化后表达精度不够），影响整体算子精度，也浪费了 lut 空间，故在调节算子精度时需重点调节x的 min 取值。

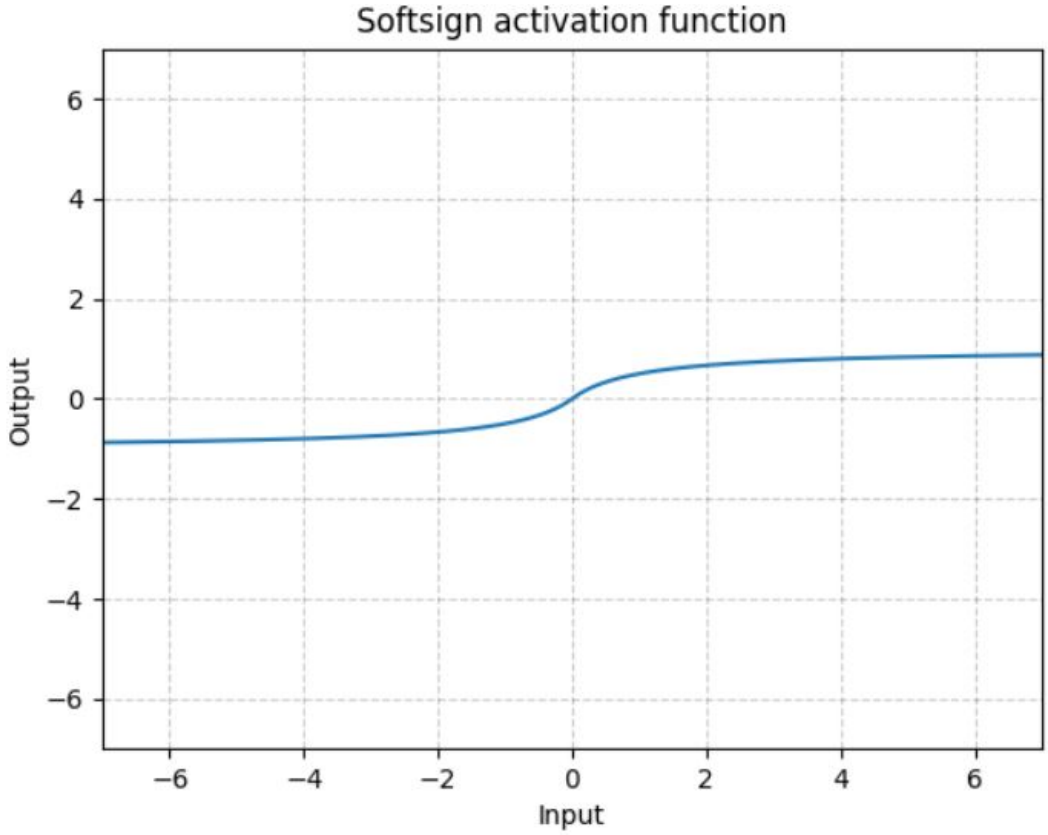
规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.68 Softsign

算子定义

Takes one input tensor (X) and produces one output tensor (Y), where the function `y = x / (abs(x) + 1)`, is applied to the tensor elementwise.



量化方法

对于$y = Softsign(x)$有：$(y_q + zp_y)/s_y = Softsign\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Softsign\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Softsign 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。同时需要注意的是由对称性 Softsign(-x)=-Softsign(x)，实际可只对正半轴存储 lut 表以节约空间。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
```

```
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.69 SpaceToBatch

算子定义

Rearranges (permutes) data from blocks of spatial data into batch. This is the reverse transformation of BatchToSpace. More specifically, this operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the batch dimensions.

量化方法

略。参见 BatchToSpace。

规格说明

略。参见 BatchToSpace。

4.70 SpaceToDepth

算子定义

Rearranges (permutes) data from blocks of spatial data into depth. This is the reverse transformation of DepthToSpace. More specifically, this operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimensions.

量化方法

略。参见 DepthToSpace。

规格说明

略。参见 DepthToSpace。

4.71 Split

算子定义

Splits a tensor into a list of tensors along the specified 'axis' direction.

量化方法

略。参见 Crop。

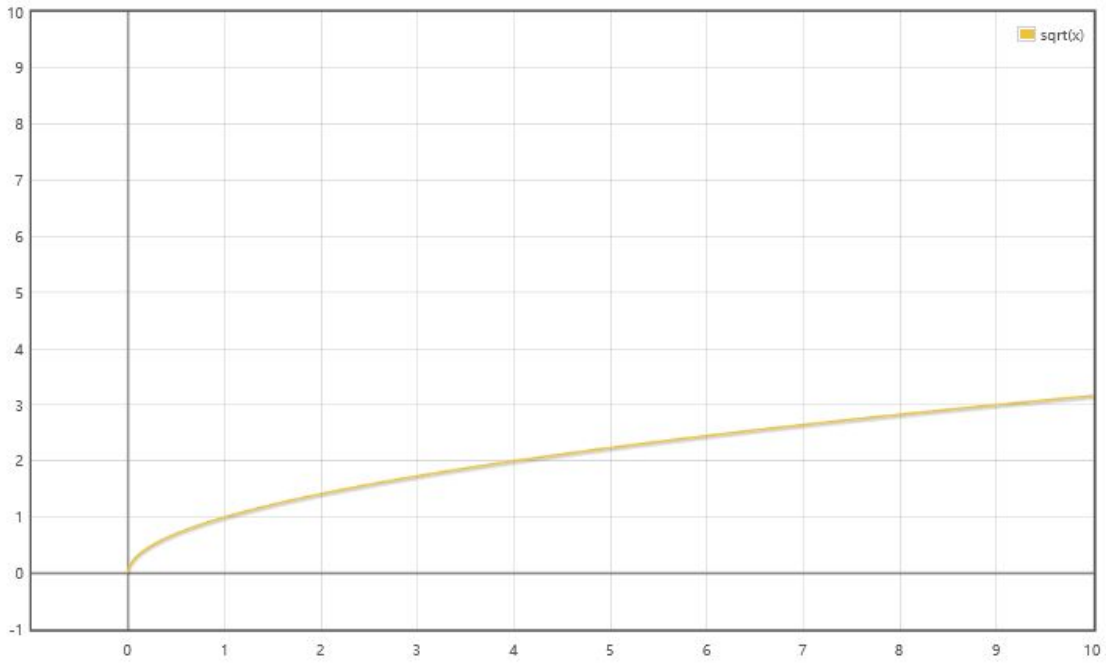
规格说明

略。参见 Crop。

4.72 Sqrt

算子定义

Square root takes one input tensor (X) and produces one output tensor (Y), where the square root is, y = x^0.5, is applied to the tensor elementwise.



量化方法

对于$y = Sqrt(x)$有：$(y_q + zp_y)/s_y = Sqrt\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Sqrt\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Sqrt 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。

规格说明

```
Input 0:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.73 Square

算子定义

Computes square of input tensor (X) elements-wise, where the equation is y = x^2.

量化方法

略，可参见 ElementwiseMul。

为快速计算也可采用 lut 查表方式计算，并且利用 Square(-x)=Square(x)对称性只存储正半轴的表以节约空间。

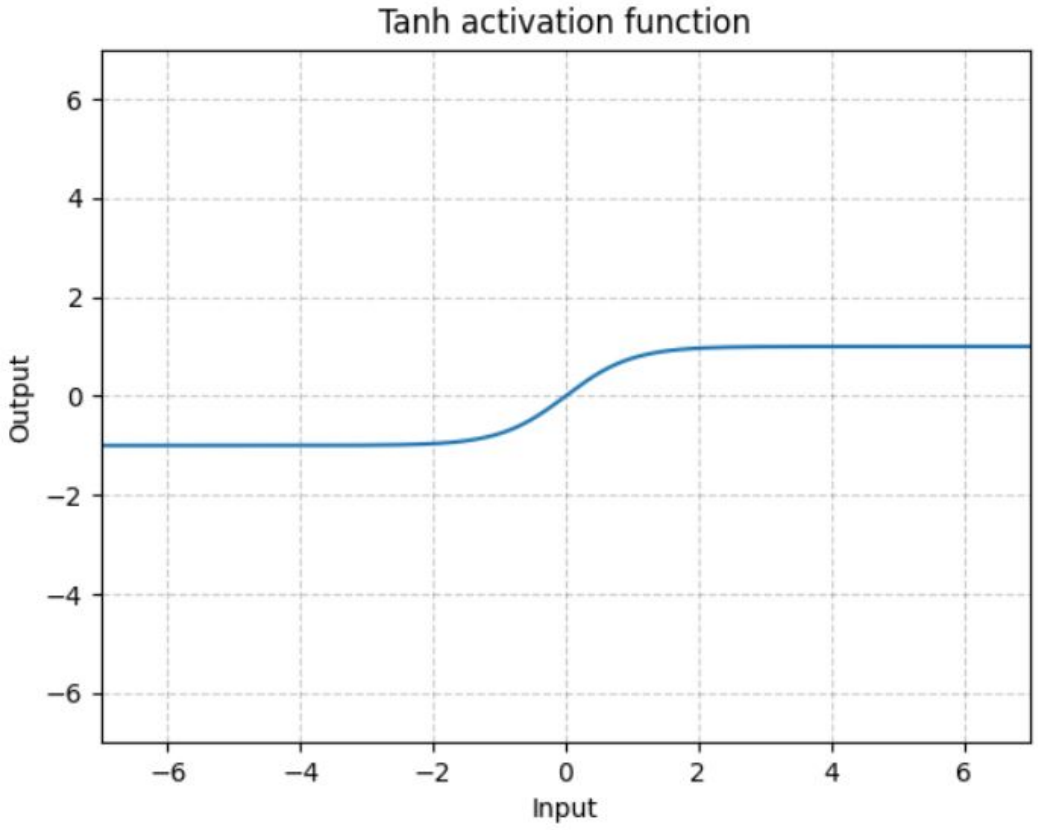
规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.74 Tanh

算子定义

Calculates the hyperbolic tangent of the given input tensor element-wise.



量化方法

对于$y = Tanh(x)$有：$(y_q + zp_y)/s_y = Tanh\left((x_q + zp_x)/s_x\right)$,
从而：$y_q = Tanhc\left((x_q + zp_x)/s_x\right)s_y - zp_y$,
因此可离线做 lut 查找表：

- 根据x_q的数据类型枚举得到值域集合$\widehat{x_q}$（若x_q范围较大，可采样获得一个子集）
- 对$\widehat{x_q}$反量化获得$\widehat{x_f}$
- 对$\widehat{x_f}$计算 Tanh 函数值获得$\widehat{y_f}$
- 量化$\widehat{y_f}$获得最终的 lut 表

前向计算查表时如果 lut 是采样后的小表，则需要做相应的插值操作。需要注意的是 Tanh 函数沿 x 正负半轴均缓慢趋于平坦，若输入x的 min 取值过小或 max 取值过大则会导致量化后的值大量趋同无法区分（即量化后表达精度不够），影响整体算子精度，调节精度时输入的 min,max 取值可于较近处截断（如经验值 3.0 附近）。同时注意到对称性 Tanh(-x)=-Tanh(x)，可只对正半轴做表以节约 lut 空间。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int
   granularity: per-tensor
restriction: none
```

4.75 Tile

算子定义

Constructs a tensor by tiling a given tensor.

量化方法

略。参见 Crop。

规格说明

略。参见 Crop。

4.76 TopK

算子定义

Retrieves the top-K largest or smallest elements along a specified axis, it returns two outputs: Values, Indices.

量化方法

根据定义，容易得出，对于$y_f, z_f = TopK(x_f)$有$y_q, z_q = TopK(x_q)$。即前向运算过程和浮点域上的计算方式一样，无需多余的量化相关运算。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
Output 1:
   data_type  : uint
   granularity: per-tensor
restriction: none
```

4.77 Transpose

算子定义

Permutes the input tensor similarly to Numpy-style.

量化方法

略。参见 Reshape。

规格说明

略。参见 Reshape。

4.78 UpsampleByIndex

算子定义

Performs Upsample operation on the inputs according to the index in corresponding indices tensor and returns the output tensor Y.

量化方法

由算子定义可知$(y_q + zp_y)/s_y = UpsampleByIndex\left((x_q + zp_x)/s_x,\ indices\right)$,
从而$y_q = \left(UpsampleByIndex(x_q,\ indices) + zp_x\right)s_y/s_x - zp_y$。令$s_y$等于$s_x$, zp_y等于zp_x，则整数域上的 UpsampleByIndex 运算方式可与浮点域上一致，无需多余的量化相关操作。

规格说明

```
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : uint
   granularity: per-tensor
Output 0:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

4.79 Where

算子定义

Selects elements from x or y, depending on different conditions. That is, the condition tensor acts as a mask that chooses, based on the value at each element, whether the corresponding element or row in the output should be taken from x (condition `True`) or y (condition `False`). The parameters are in order of [condition, X1, X2]. Generally, the data type of input condition tensor (C1) is an integer in float or int IR, and its value is 0 or 1.

量化方法

由算子定义$z_f = where(c_f, x_f, y_f)$，有$(z_q + zp_z)/s_z = where\left((c_q + zp_c)/s_c,\ (x_q + zp_x)/s_x,\ (y_q + zp_y)/s_y\right)$，从而：

$$z_q = where\left((c_q + zp_c)/s_c,\ (x_q + zp_x)/s_x,\ (y_q + zp_y)/s_y\right)s_z - zp_z$$

$$z_q = where\left((c_q + zp_c)/s_c,\ (x_q + zp_x)s_z/s_x - zp_z,\ (y_q + zp_y)s_z/s_y - zp_z\right)$$

因 c 为布尔类型，令$s_c = 1, zp_c = 0$。同时可离线将$s_z/s_x, s_z/s_y$分别表示为规范形式$M_0/2^{N_0}, M_1/2^{N_1}$，其中M_0, N_0, M_1, N_1均为整数。则最终：

$$z_q = where(c_q,\ (x_q + zp_x)M_0/2^{N_0} - zp_z,\ (y_q + zp_y)M_1/2^{N_1} - zp_z)$$

规格说明

```
Input 0:
   data_type  : uint
   granularity: per-tensor
Input 0:
   data_type  : int or uint
   granularity: per-tensor
Input 1:
   data_type  : int or uint
   granularity: per-tensor
Output 1:
   data_type  : int or uint
   granularity: per-tensor
restriction: none
```

5 周易 AIPU 模型量化实例教程

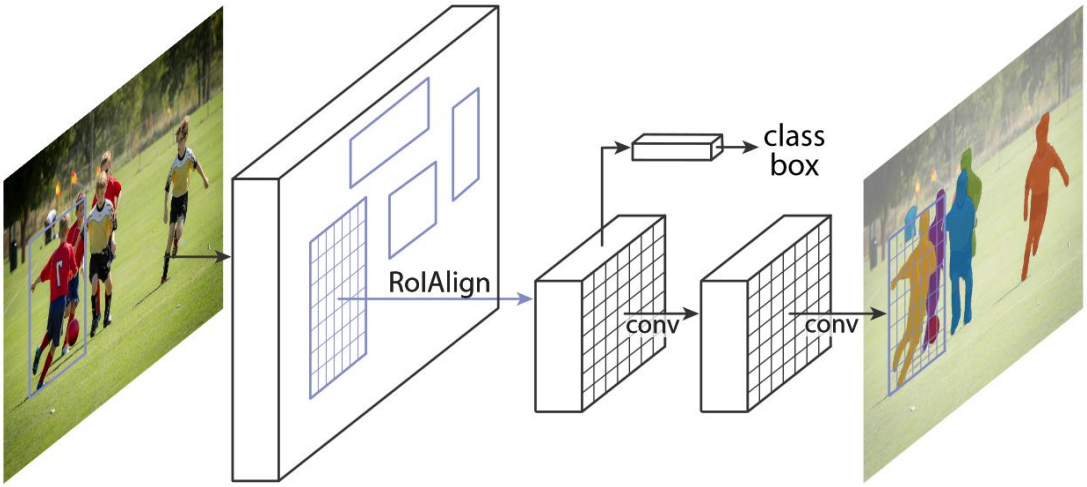
5.1 Mask R-CNN

Mask R-CNN[20] [21] [22]是 ICCV2017 的 best paper，在一个网络中同时做目标检测（object detection）和实例分割（instance segmentation）。在 COCO 数据集的三个挑战赛：instance segmentation、bounding-box object detecton、person keypoint detection 中的效果都要优于之前的单模型算法（包括 COCO2016 比赛的冠军算法）。

5.1.1 模型结构简介

从名字就可知道，它与 R-CNN 系列有着密不可分的关系，若对 Faster R-CNN[23]熟悉的话就会发现其结构有很大的相似性，接下来我们先大致分析下其异同点（从量化角度来考量）。

模型整体结构如下图所示，原始图片经过特征提取之后，与 Faster R-CNN 的 RoIPooling 不同的是这里经过 RoiAlign 进行处理从而得到固定大小的特征图，而后进行分类、分割等任务处理。



需要注意的是，Mask R-CNN 不是指某一种具体的算法，而是一个灵活的框架，其具有很好的拓展性以及易用性，大致认为在 Faster R-CNN 负责检测的基础上加入不同的分支就可以完成不同的任务，比如可以实现目标分类、目标检测、语义分割、实例分割、人体姿势识别等。

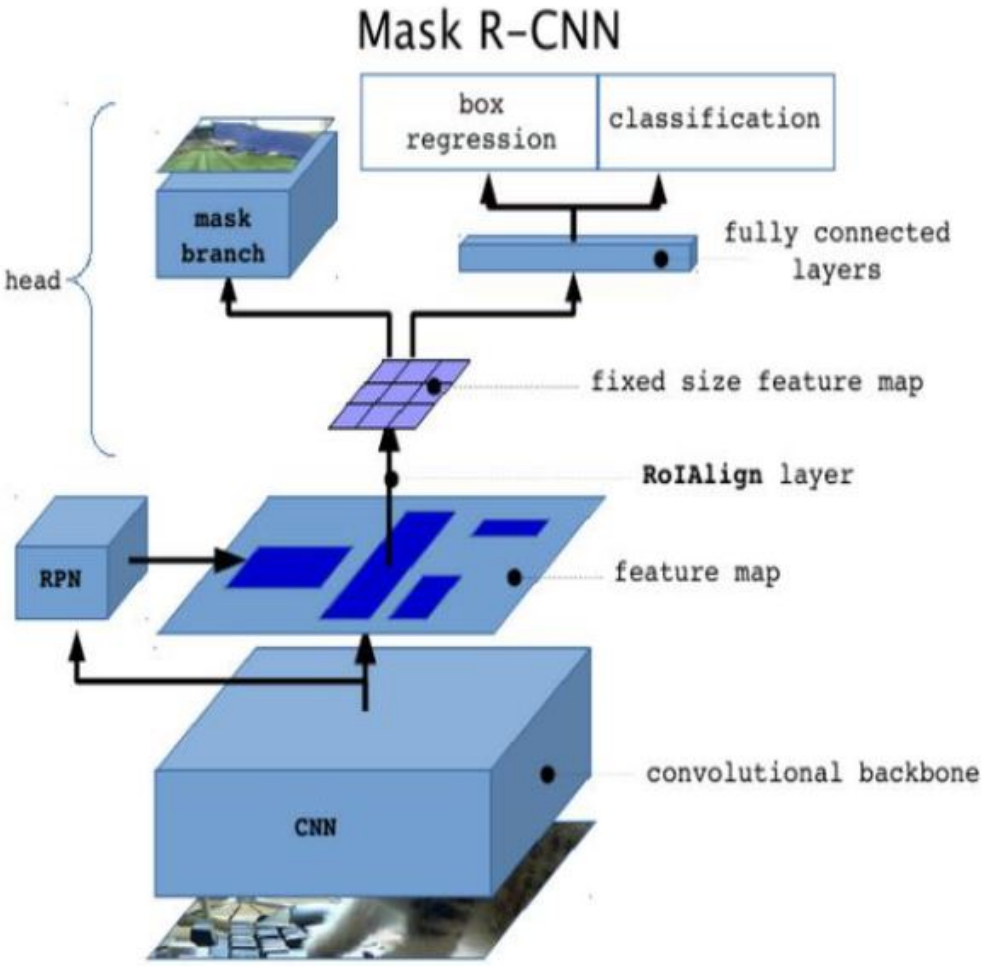
整个模型的算法思路非常直观，就是在原始的 Faster R-CNN 算法基础上增加了 FCN 来产生对应的 mask 分支，具体到 tensorflow 实现版本来说就是，FPN 特征金字塔模型进行特征提取，然后 ROIalign 进行特征固定 size，最后分别走类 Faster R-CNN 的检测分支，跟 FCN 分支，整体如下图所示:

[20] Mask R-CNN. https://arxiv.org/pdf/1703.06870.pdf

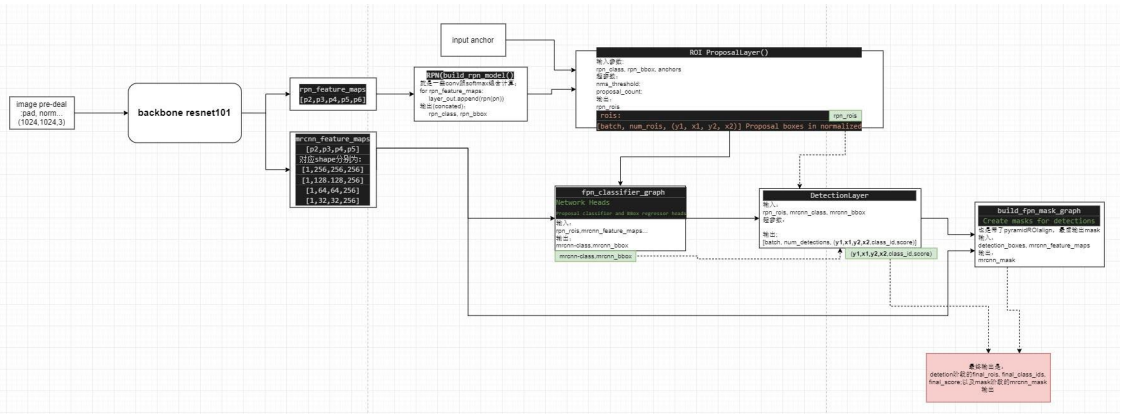
[21] mask r-cnn source code and pre-trained model, tensorflow version. https://github.com/matterport/Mask_RCNN

[22] mask r-cnn source code and pre-trained model, pytorch/caffe2 version. https://github.com/facebookresearch/detectron2

[23] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. https://arxiv.org/pdf/1506.01497.pdf



我们就 TensorFlow 版的代码进行一下梳理，可以得到以下的具体实现细节：



- 原始的输入图像尺寸是 1024*1024*3，经过 backbone 模型 resnet101，并通过 FPN 结构得到 rpn 层的特征层（P2,P3,P4,P5,P6），得到 mrcnn 特征层(P2,P3,P4,P5)；
- rpn 的每一个特征层经过一系列 conv 以及 softmax 计算，得到 rpn_class,rpn_bbox,将不同层的输出拼接起来，跟 anchor 节点一起送入 ROI proposal 层，基于预设的一些筛选超参数，最终输出固定 size 的 rpn_rois 格式为：`[batch, num_rois, (y1, x1, y2, x2)] Proposal boxes in normalized`；

- 这部分 rois 第一阶段候选框会首先进入分类头网络 fpn_classifier,在里面先 PyramidROIAlign 得到固定 size 的特征（需要结合 mrcnn 的各层特征来做），然后同样的接 FC、softmax 得到 mrcnn_probs, mrcnn_bbox 输出（给 detection 使用的）；
- detection 层拿到第二阶段分类后的结果就进行调整框、NMS 等操作了，最后输出一系列的框，这就是模型最后确定的输出框了；
- 然后 mask 分支拿到第二阶段确定好的框，以及 mrcnn_feature_maps，就开始用 FCN 进行分割计算了，其内部也是先进行 PyramidROIAlign 操作，得到固定大小的 shape，然后全卷积网络 FCN 直接算，最后经过 sigmoid 输出每个像素的概率。

再后面就是最后的数据后处理了，并没有包含在网络内，因此暂不考虑，实践中直接放到 CPU 中计算即可。

5.1.2 模型算子一览

经过上述的分析，我们可以把需要量化的算子逐一筛选出来，然后分别量化，最后进行整网组合,追加细致微调即可实现 Mask R-CNN 量化：

- 由于 backbone 是 resnet101，基于已有经验，这不是量化瓶颈点，因此直接用常规量化方案即可；
- proposal layer 是由一系列的逻辑操作完成的，因此我们需要把这部分当做一个整体来考虑量化；
- NMS 层在全定点模式下需要微调精度，同时需要考虑硬件可适配性，因此需要特殊设计量化方案；
- PyramidRoIalign 层需要特殊量化以保证精度；
- fpn_classifier 层中的 softmax 层需要特殊量化以保证模型整体精度；
- 其他例如 concat, eltwise 层等都需要结合模型微调方可达到很好的精度。

5.1.3 量化精度调节

量化精度主要分为两个步骤，第一步把上一小节拆分的算子，先单个算子粒度量化完成，然后进行 block 粒度、graph 粒度的精度调节即可。

5.1.3.1 backbone resnet101 量化

resnet[24]系列网络得整体结构如下所示：

[24] Deep Residual Learning for Image Recognition. https://arxiv.org/pdf/1512.03385.pdf

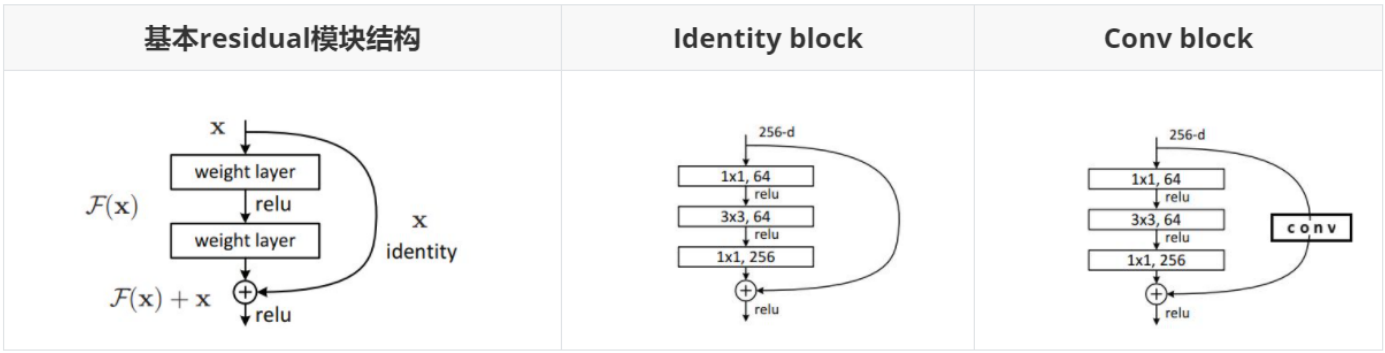
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Mask R-CNN 选择的是 resnet101，包括如下主要算子需要量化:

1. Padding
2. Conv2D
3. BatchNormalization
4. ReLU
5. MaxPooling2D
6. **Conv_block**
7. **Identity_block**
8. AveragePooling2D
9. Softmax
10. ...

除了 6、7 号算子，其他算子均在前面章节有介绍，因此不再赘述。

而 6、7 号算子则是下图所示的模块，identity_block 则是右图所示结构，包含 Conv2D，BN，ReLU，ElementwiseAdd 等组成部分，**而 conv_block 跟 identity_block 的唯一区别在于残差分支（shortcut）上有一个 conv 层。**



小结：这一部分的量化直接遵循常规量化方案即可，基本无量化精度风险点。

5.1.3.2 proposal layer 量化

这一层是紧接在 RPN 层之后的，输入参数为 RPN 输出的候选框跟对应前后景概率，以及预先生成的 anchors。

> Region Proposal Network 层：把（P2~P6）每一层都进行映射，映射成对应的候选框跟对应的前后景概率，每层的输出再分别拼接（Concatenate）起来；

对应源码接口如下所示，proposal_count 指定第一阶段筛选出多少个框，nms_threshold 顾名思义作为阈值。

```
rpn_rois = ProposalLayer(
    proposal_count=proposal_count,
    nms_threshold=config.RPN_NMS_THRESHOLD,
    name="ROI",
    config=config)([rpn_class, rpn_bbox, anchors])
```

实现细节参见源码 ProposalLayer。计算流程大致为：

1. 输入 anchors,rpn_scores,rpn_bbox(为 deltas，调整参数)，为了性能考虑，首先基于 NUMS = min(PRE_NMS_LIMIT=6000, anchor 数量)参数，取 rpn_scores 的 TopK，然后把这 NUMS 个数据抽离出来，当然对应的 anchor 也需要这样抽离一次；
2. 先粗选一遍，然后把剩下的 NUMS 个框直接 applay_box_deltas()操作，也就是进行框的位置调整；
3. 调整完之后，框有可能是超出 0~1 归一化范围的，因此把超出范围的框给截取掉，此时接一个 clip 操作即可完成；
4. 此时已是调整之后的框了，我们需要把框进行 NMS 操作，选出第一阶段的输出框。

TopK 量化：

由于 8bit 量化时进来的 sores 是 int8 量化的，同时 topk 选出来的个数为 6000，因此很明显，精度可能会面临不够的问题，保守设计，我们在此处，scores 采用 int16 精度进行表示。

采用了混合精度量化，为保证精度，score 之前的 softmax 也必须采用 int16，softmax 采用的是 LUT 方案，因此，对应的 softmax 之前的 conv2D 层采用了 int8 activation * int8 weights --> int16 output 的量化方案，至此，精度链条才得以保证。

applay_box_deltas()量化：

从 backbone 过来的数据为 int8 类型，是从 RPN 输出的值进行 concatenate 得到的。

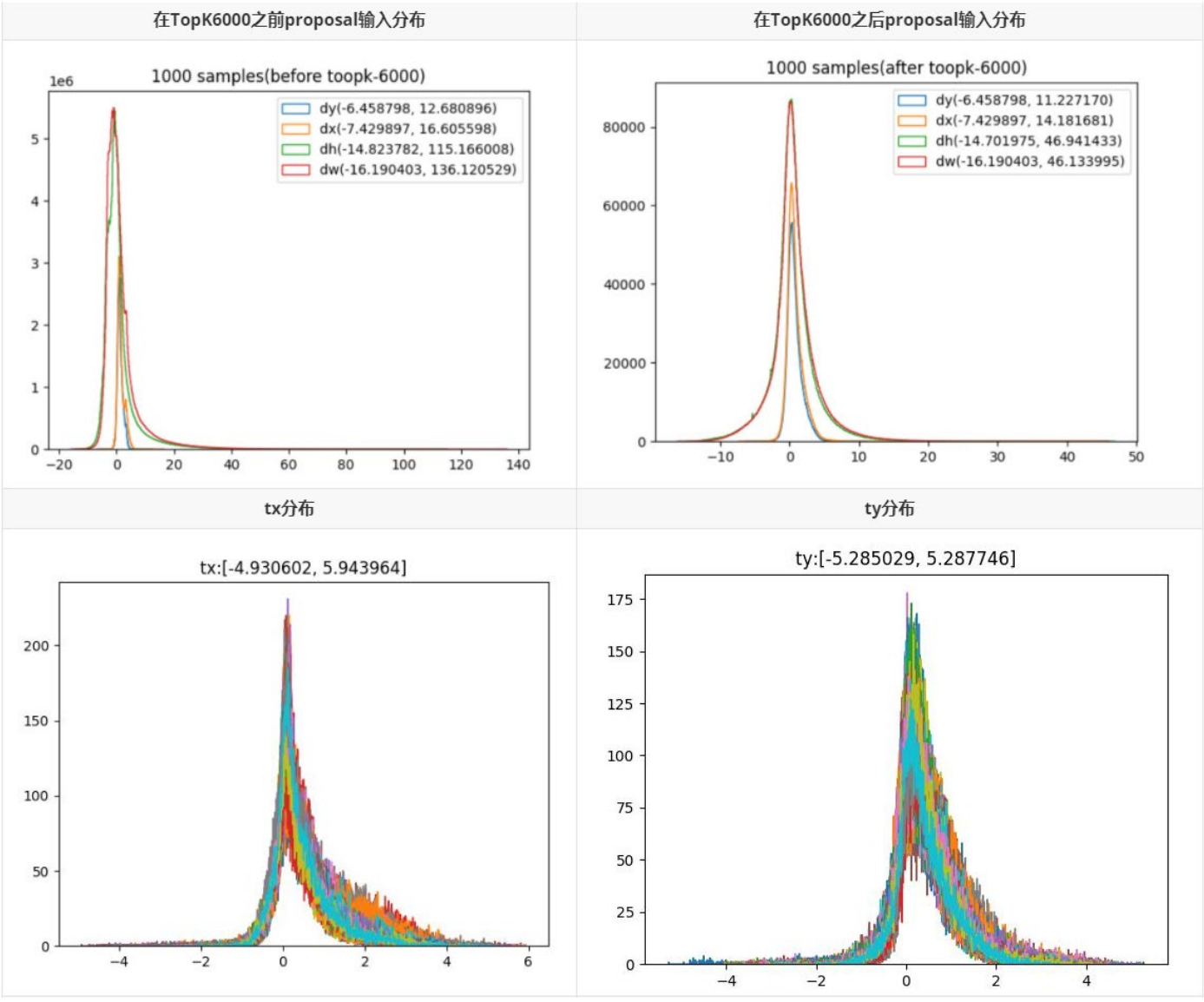
```
# Note that P6 is used in RPN, but not in the classifier heads.
rpn_feature_maps = [P2, P3, P4, P5, P6]
mrcnn_feature_maps = [P2, P3, P4, P5]
anchors = input_anchors
# RPN Model
rpn = build_rpn_model(config.RPN_ANCHOR_STRIDE,
        len(config.RPN_ANCHOR_RATIOS),
        config.TOP_DOWN_PYRAMID_SIZE)
# Loop through pyramid layers
layer_outputs = []  # list of lists
for p in rpn_feature_maps:
    layer_outputs.append(rpn([p]))
# Concatenate layer outputs
# Convert from list of lists of level outputs to list of lists
# of outputs across levels.
# e.g. [[a1, b1, c1], [a2, b2, c2]] => [[a1, a2], [b1, b2], [c1, c2]]
```

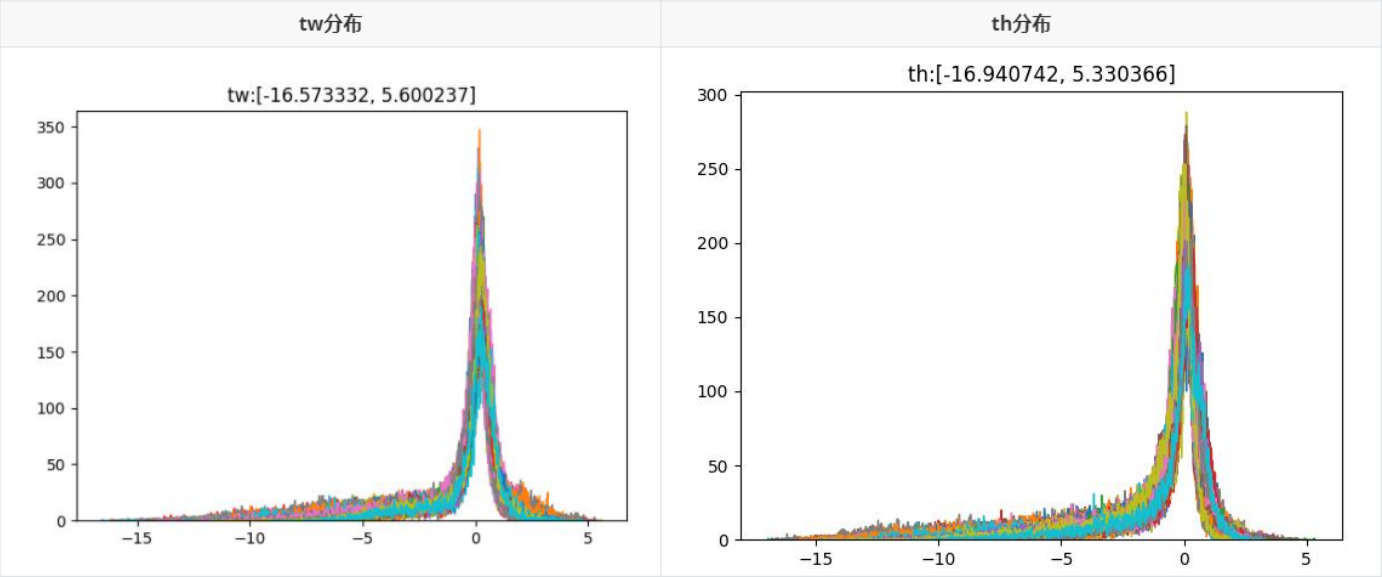
```
output_names = ["rpn_class_logits", "rpn_class", "rpn_bbox"]
outputs = list(zip(*layer_outputs))
outputs = [KL.Concatenate(axis=1, name=n)(list(o))
           for o, n in zip(outputs, output_names)]

rpn_class_logits, rpn_class, rpn_bbox = outputs
```

此处需要注意的是，由于不同 rpn_feature 层[P2, P3, P4, P5, P6]的数据分布范围不一样，因此在量化时，量化模型就变为:

> out_f = [P2_f*scale2, P3_f*scale3, P4_f*scale4, P5_f*scale5, P6_f*scale6]
> 此时在进行量化的时候，则面临选择哪个 scale 作为 out_scale 的问题了！不同的 scale 选择在当前节点 tensor 下有不同的精度损失，同时，在整个模型的精度表现中又有不同的精度损失贡献，而且这个节点是处于 backbone 跟 postprocess 过程的交界处，因此这一部分精度的重要性不言而喻。

我们量化的设计原则是：**“先保证单个算子精度，再融合进模型中进行微调或者全 graph 调优。”** 因此，我们设计 proposal layer 层量化的时候，先假设进来的值是只有一次量化 rounding 误差的，于是，我们来统计其量化分布范围（Mask R-CNN stage1 box_encoding 数据）：





通过统计中间计算过程的数据分布，可以得到一系列量化 scale，其中 exp 函数的查找表设计细节可以从前面章节找到对应部分，在此不再赘述。

clip 操作在此处直接饱和即可。

实现思路及细节：

这一部分接收 anchors 以及 RPN 输出的 deltas，做框的调整操作，跟第二阶段的 decode_box 是相同的操作流程，因此可以把这一部分给抽离出来，形成一个单独的算子，名为 boundingbox 层：

```
def apply_box_deltas_graph(boxes, deltas):
    """Applies the given deltas to the given boxes.
    boxes: [N, (y1, x1, y2, x2)] boxes to update
    deltas: [N, (dy, dx, log(dh), log(dw))] refinements to apply
    """
    # Convert to y, x, h, w
    height = boxes[:, 2] - boxes[:, 0]
    width = boxes[:, 3] - boxes[:, 1]
    center_y = boxes[:, 0] + 0.5 * height
    center_x = boxes[:, 1] + 0.5 * width
    # Apply deltas
    center_y += deltas[:, 0] * height
    center_x += deltas[:, 1] * width
    height *= tf.exp(deltas[:, 2])
    width *= tf.exp(deltas[:, 3])
    # Convert back to y1, x1, y2, x2
    y1 = center_y - 0.5 * height
    x1 = center_x - 0.5 * width
    y2 = y1 + height
    x2 = x1 + width
    result = tf.stack([y1, x1, y2, x2], axis=1, name="apply_box_deltas_out")
    return result
```

大致思路就是要保证计算过程中的任何一个节点不会产生过大的量化误差，误差来源有 floor 误差、round 误差以及主要关注的溢出误差。

5.1.3.3 NMS 量化

NMS:

TensorFlow 标准 NMS 调用流程如下所述：将框、分数、NMS 内部选取个数、nms 阈值四个参数传给 tf 标准实现，返回来的是有效框的下标位置，基于此位置信息抽取有效框，假如小于 self.proposal_count 个数则进行补零（这是为了方便后续处理，以及底层实现）。

```
# Non-max suppression
def nms(boxes, scores):
    indices = tf.image.non_max_suppression(
        boxes, scores, self.proposal_count,
        self.nms_threshold, name="rpn_non_max_suppression")
    proposals = tf.gather(boxes, indices)
    # Pad if needed
    padding = tf.maximum(self.proposal_count - tf.shape(proposals)[0], 0)
    proposals = tf.pad(proposals, [(0, padding), (0, 0)])
    return proposals
proposals = utils.batch_slice([boxes, scores], nms,
                              self.config.IMAGES_PER_GPU)
return proposals
```

我们先看下 float 实现过程：

```
def single_nms(self, box, score, max_nms_box_num):
    """
    多分类时的NMS 也是按照每一类分别进行NMS 的;
    params:
        box: 候选框
        score：候选框对应的概率;
        max_nms_box_num： 输出框个数上限;
    return:
        目标框对应的下标;

    """
    nms_box = torch.zeros((max_nms_box_num, 4))
    nms_score = torch.zeros((max_nms_box_num))
    keep = torch.zeros((max_nms_box_num))

    y0 = box[:, 0]
    x0 = box[:, 1]
    y1 = box[:, 2]
    x1 = box[:, 3]
    iou_threshold =  self.get_param('iou_threshold')
    areas = (y1 - y0) * (x1 - x0)# 算出所有候选框的面积;
    order = score[:].argsort(dim=-1,descending=True) #降序排序，选出概率最大的框;
    keep_idx = 0
    boxNum_perclass_single = 0
    device = x0[0].device
    while order.size()[0] > 0:
        boxNum_perclass_single += 1
        i = order[0]  #把分数最高的先存起来，因为，这个最有可能是目标框;

        keep[keep_idx] = i
        nms_box[keep_idx, :] = box[i, :]
        nms_score[keep_idx] = score[i]
```

```
        keep_idx += 1
        if keep_idx >= max_nms_box_num: #目标框够了，不能再多了，返回。
            break

        xx0 = torch.max(x0[i], x0[order[1:]])
        yy0 = torch.max(y0[i], y0[order[1:]])
        xx1 = torch.min(x1[i], x1[order[1:]])
        yy1 = torch.min(y1[i], y1[order[1:]])
        w = torch.max(torch.tensor(0.0).to(device), xx1 - xx0)
        h = torch.max(torch.tensor(0.0).to(device), yy1 - yy0)
        inter = w * h
        union = areas[i] + areas[order[1:]] - inter # 并集=A+B-交集;

        inter_area_thresh = union * iou_threshold # 交并比除法转成乘法操作;
        inds = torch.where(inter <= inter_area_thresh)[0] # IoU 大于阈值的则大概率是重叠框，去除掉;
        order = order[inds + 1] # IoU 小于阈值的才有可能是另外一个检测目标，存起来，下次用。

    boxNum_perclass = boxNum_perclass_single
    nms_box = nms_box[0:boxNum_perclass, :]
    nms_score = nms_score[0:boxNum_perclass]
    keep = keep[0:boxNum_perclass]
    return nms_box, nms_score, boxNum_perclass, keep
```

NMS 定点化中会面临的几个问题点：

1. 计算过程中存在大量连续计算步骤，对应溢出风险较大，如何解决？
2. 溢出处理时如何处理精度跟溢出的权衡？
3. NMS 出现小框丢失的情况时，量化如何调优？
4. iou_thresh 如何在全定点部署环境中实现量化同时尽可能保证精度？

我们的量化算法包中均已较好解决上述问题，限于篇幅，在此不赘述。

5.1.3.4 PyramidRoIAlign 量化

PyramidRoIAlign:

Mask R-CNN 中源码如下：

```
class PyramidROIAlign(KE.Layer):
    """Implements ROI Pooling on multiple levels of the feature pyramid.
    """

    def __init__(self, pool_shape, **kwargs):
        super(PyramidROIAlign, self).__init__(**kwargs)
        self.pool_shape = tuple(pool_shape)

    def call(self, inputs):
        # Crop boxes [batch, num_boxes, (y1, x1, y2, x2)] in normalized coords
        boxes = inputs[0]

        # Image meta
        # Holds details about the image. See compose_image_meta()
        image_meta = inputs[1]

        # Feature Maps. List of feature maps from different level of the
        # feature pyramid. Each is [batch, height, width, channels]
```

```
        feature_maps = inputs[2:]

        # Assign each ROI to a level in the pyramid based on the ROI area.
        y1, x1, y2, x2 = tf.split(boxes, 4, axis=2)
        h = y2 - y1
        w = x2 - x1
        # Use shape of first image. Images in a batch must have the same size.
        image_shape = parse_image_meta_graph(image_meta)['image_shape'][0]


        # 这部分计算ROI属于哪个level时，为了兼容移植，可以考虑判断实现方案；
        # Equation 1 in the Feature Pyramid Networks paper. Account for
        # the fact that our coordinates are normalized here.
        # e.g. a 224x224 ROI (in pixels) maps to P4
        image_area = tf.cast(image_shape[0] * image_shape[1], tf.float32)
        # roi_level 限制在[2.5]之间，对应[P2,P3,P4,P5]
        roi_level = log2_graph(tf.sqrt(h * w) / (224.0 / tf.sqrt(image_area)))
        roi_level = tf.minimum(5, tf.maximum(
            2, 4 + tf.cast(tf.round(roi_level), tf.int32)))
        # 压缩roi_level的axis=2的轴，shape变化[batch, num_boxes,1]->[batch, num_boxes]
        roi_level = tf.squeeze(roi_level, 2)

        # Loop through levels and apply ROI pooling to each. P2 to P5.
        pooled = []
        box_to_level = []
        for i, level in enumerate(range(2, 6)):
            ix = tf.where(tf.equal(roi_level, level))
            # 提取满足索引的对应boxes
            level_boxes = tf.gather_nd(boxes, ix)

            # Box indicies for crop_and_resize.
            # 指定第i个方框要使用的图像，即指定第i个box对应batch中哪一张图像
            box_indices = tf.cast(ix[:, 0], tf.int32)

            # Keep track of which box is mapped to which level
            box_to_level.append(ix)

            # Stop gradient propogation to ROI proposals
            level_boxes = tf.stop_gradient(level_boxes)
            box_indices = tf.stop_gradient(box_indices)

            # Crop and Resize
            # Result: [batch * num_boxes, pool_height, pool_width, channels]
            # 使用tf.image.crop_and_resize实现简化的ROIAling，每个bin只取一个点
            pooled.append(tf.image.crop_and_resize(
                feature_maps[i], level_boxes, box_indices, self.pool_shape,
                method="bilinear"))

        # Pack pooled features into one tensor
        pooled = tf.concat(pooled, axis=0)

        # Pack box_to_level mapping into one array and add another
        # column representing the order of pooled boxes
        # 将box_to_level映射到一个数组中，并新增一列表示池化顺序
        box_to_level = tf.concat(box_to_level, axis=0)
        box_range = tf.expand_dims(tf.range(tf.shape(box_to_level)[0]), 1)
        box_to_level = tf.concat([tf.cast(box_to_level, tf.int32), box_range],
                                 axis=1)

        # Rearrange pooled features to match the order of the original boxes
        # Sort box_to_level by batch then box index
        # TF doesn't have a way to sort by two columns, so merge them and sort.
        '''由于所有经过RPN推荐产生的ROIs都经过refined,
```

```
        因此其scale没有严格遵守RPN_ANCHOR_SCALES = (32, 64, 128, 256, 512)变化,
        这会导致pooled_features的顺序与原始boxes的顺序不同，因此需要重新排序。
        '''
        # 排序原则是首先根据batch排序，再根据box_index排序
        # 这里box_to_level[:, 0]为batch轴，×100000是为了加大权重，得到排序tensor
        sorting_tensor = box_to_level[:, 0] * 100000 + box_to_level[:, 1]
        # 根据sorting_tensor得到其top_all的排序索引，转置后得到升序索引
        ix = tf.nn.top_k(sorting_tensor, k=tf.shape(
            box_to_level)[0]).indices[::-1]
        # ix为pooled_features的重排序升序索引，排序依据为batch->box_index
        ix = tf.gather(box_to_level[:, 2], ix)
        pooled = tf.gather(pooled, ix)

        # Re-add the batch dimension
        pooled = tf.expand_dims(pooled, 0)
        return pooled

    def compute_output_shape(self, input_shape):
        return input_shape[0][:2] + self.pool_shape + (input_shape[2][-1], )
```

PyramidRoIAlign 比 RoIAlign 就多了个特征金字塔部分，整体关系是，输入进来的 nor_box，先判断是属于哪个特征层，紧接着就是直接做 RoIAlign 操作。

首先计算 ROI 属于哪一个特征层，计算代码如下：

```
# Equation 1 in the Feature Pyramid Networks paper. Account for
# the fact that our coordinates are normalized here.
# e.g. a 224x224 ROI (in pixels) maps to P4
image_area = tf.cast(image_shape[0] * image_shape[1], tf.float32)
# roi_level 限制在[2.5]之间，对应[P2,P3,P4,P5]
roi_level = log2_graph(tf.sqrt(h * w) / (224.0 / tf.sqrt(image_area)))
roi_level = tf.minimum(5, tf.maximum(
    2, 4 + tf.cast(tf.round(roi_level), tf.int32)))
# 压缩roi_level的axis=2的轴，shape变化[batch, num_boxes,1]->[batch, num_boxes]
roi_level = tf.squeeze(roi_level, 2)
```

这里可能会有些疑问，比如为什么是这个计算公式？为什么超参值是 224？

可以参见 https://github.com/matterport/Mask_RCNN/issues/128 上的回答。

知道属于哪一层了，紧接着就每一个 nor_box 做 RoIAlign，整体流程如下所述：

```
for box in nor_box:
    roi_level = box_to_level(box)
    RoIAlign(box, roi_level)
```

注：

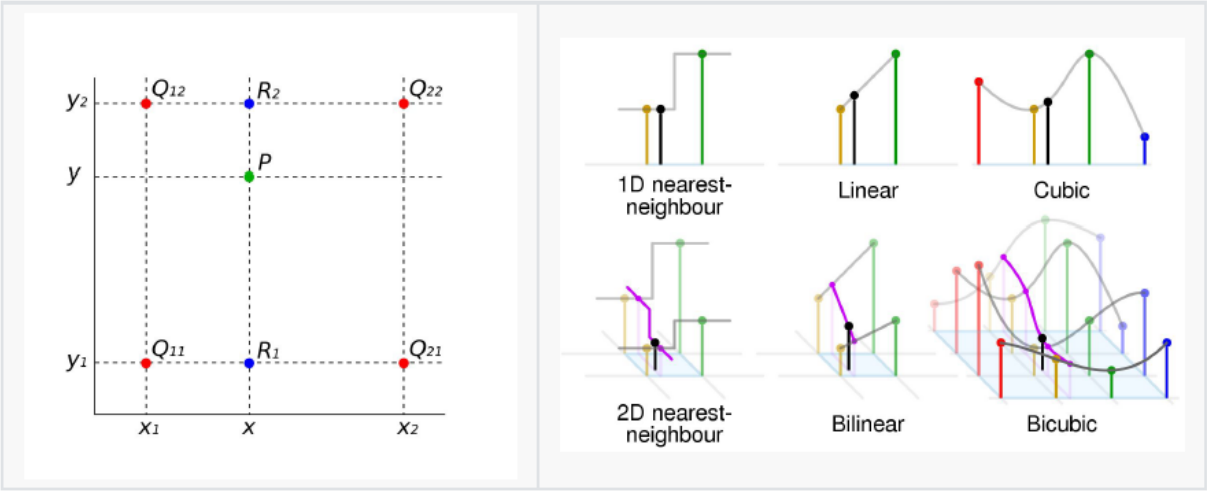
> RoIAlign 与 RoIpooling 两者在算法上的异同点，即 ROI Pooling 采用了两次量化操作，而 ROIAlign 则没有采用量化操作，而是使用了线性插值算法。
>
> 可以认为 RoIAlign 是 RoIPooling 的改进版，解决了映射误差和均分误差的问题，提高了总体的检测准确率。RoIAlign 取消了量化操作，利用双线性插值的方法获得坐标为浮点数的像素点上的图像数值，从而将量化误差给最小化。

代码移植:

在双线性插值时，我们把 ROIs 一个一个进行处理，先做下预处理，把 RIOS 归一化坐标取出来，换算出 ROI 的 w，然后除以 pool_size_w，得到了每个 section 的步长，结合起点坐标就得到了每个 section 的坐标，然后把超出 feature map 的坐标剔除掉。

```
    resize_feature = torch.zeros((nor_box.shape[1], resize_height_, resize_width_, channel_))
    # for batchidx in range(feature.shape[0]):
    batch_idx = 0
    for boxidx in range(nor_box.shape[1]):
        y0 = nor_box[batch_idx, boxidx, 0]
        x0 = nor_box[batch_idx, boxidx, 1]
        y1 = nor_box[batch_idx, boxidx, 2]
        x1 = nor_box[batch_idx, boxidx, 3]
        image_width=feature.shape[1]
        image_height=feature.shape[2]
        channel=feature.shape[3]
        height_scale = (y1 - y0) * (image_height - 1) / (resize_height_ - 1)
        width_scale = (x1 - x0) * (image_width - 1) / (resize_width_ - 1)

        x = (image_width-1) * x0 + torch.arange(0,resize_width_, device=out.device) * width_scale
        y = (image_height-1) * y0 + torch.arange(0,resize_height_, device=out.device) * height_scale

        yy = torch.clamp(y, 0,  image_width- 1)
        xx = torch.clamp(x, 0, image_height - 1)

        top_y_index = (torch.floor(yy)).int()
        bottom_y_index = (torch.ceil(yy)).int()
        y_lerp = (yy - top_y_index).reshape(resize_height_,1).repeat(1, channel)
        # y_lerp = torch.repeat(y_lerp, channel).reshape(resize_height_, channel)
        left_x_index =(torch.floor(xx)).int()
        right_x_index=(torch.ceil(xx)).int()
        x_lerp = (xx - left_x_index).reshape(resize_width_,1).repeat(1,channel)
        # x_lerp=torch.repeat(x_lerp,channel).reshape(resize_width_,channel)
```

针对上面算出来的每一个点，可以直接求出临近的四个像素坐标（floor 加 ceil 操作），然后按照双线性插值的公式 [25]直接求出结果即可，如下图所示：



代码中按照公式直接进行计算即可。

```
    for idxh in range(resize_height_):
        for idxw in range(resize_width_):
            #get 4 pointq
            top_left     = feature[0, top_y_index[idxh], left_x_index[idxw], :] #Q12
            top_right    = feature[0, top_y_index[idxh], right_x_index[idxw], :] #Q22
```

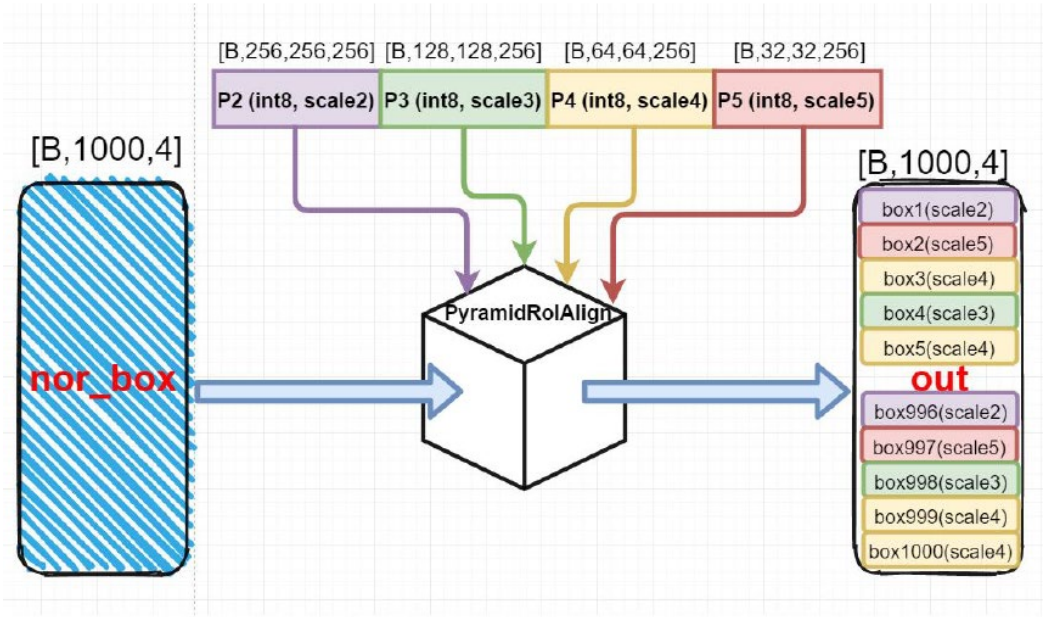
[25] https://en.wikipedia.org/wiki/Bilinear_interpolation

```
            bottom_left  = feature[0, bottom_y_index[idxh], left_x_index[idxw], :]#Q11
            bottom_right = feature[0, bottom_y_index[idxh], right_x_index[idxw], :]#Q21

            #bilinear interpretate
            # f(x,y)=Q12*(1-x_lerp)*(1-y_lerp)+Q22*x_lerp*(1-y_lerp)+Q11*(1-
x_lerp)*y_lerp+Q21*x_lerp*y_lerp
            # Q11=bottom_left
            # Q12=top_left
            # Q21=bottom_right
            # Q22=top_right
            # data=top_left*(1-x_lerp[idxw,:])*(1-y_lerp[idxh,:])+top_right*(1-
y_lerp[idxh,:])*x_lerp[idxw,:]
            # data=data+bottom_left*(1-
x_lerp[idxw,:])*y_lerp[idxh,:]+bottom_right*x_lerp[idxw,:]*y_lerp[idxh,:]
            xy=y_lerp[idxh,:]*x_lerp[idxw,:]
            fourpoint_sum=(top_left+bottom_right-top_right-bottom_left)*xy
            top = top_left + (top_right-top_left)* x_lerp[idxw,:]
            bottom=(bottom_left - top_left) * y_lerp[idxh,:]
            data = fourpoint_sum+top+bottom

            resize_feature[boxidx, idxh, idxw, :] = data
```

PyramidRoIAlign 量化方案:

主要关注下溢出问题即可，保证计算过程中的值累加不溢出。

其次，我们知道，P2~P5 的分布是不一样的，这就导致量化后会出现如下问题，输出的 box 中有四种 scale，而且位置是没有规律的，于是想要严格量化将是一个很难的问题。



我们可以选择加权该量化参数，从而平均最小化量化误差，我们看个实际的例子：

	P2	P3	P4	P5	nor_box
max	26.95	25.42	31.63	19.58	1.0
min	-26.11	-25.2	-25.29	-20.97	0

可见，P2~P5 的误差不是很大，因此我们可以直接选择 max 接近中位数的 P2，作为输出的量化误差。实测精度较好（见 4.1.4 结果分析部分），将来需要进一步调优模型时可以考虑进一步优化此部分。

关于具体实现细节，限于篇幅，在此不做详细介绍。

5.1.4 结果分析

如下表所示：

mAP@0.5 @100samples	float	quant	drop
int8 @ 1 sample calibration	68.01%	66.89%	1.12%
int8 @ 100 sample calibration	68.01%	66.68%	1.33%
int16 @ 1 sample calibration	68.01%	67.35%	0.66%
int16 @ 100 sample calibration	68.01%	67.65%	0.36%

6 FAQ

6.1 如何度量模型量化后的精度损失?

最可靠的度量方式为收集一个目标应用场景下的标准测试集，在此测试集上分别测试量化前后模型的精度指标（top-k accuracy, mAP, mIoU, ...）并加以比较。如果因为种种原因，无法获得带标签的测试集，那么可以收集或构造一批不带标签的目标应用场景下的测试样本，然后分别输入给量化前后的模型得到对应的输出张量y_f和y_q, 在应用合适的距离度量函数计算y_f和y_q的相似度，常用的距离度量函数为余弦相似度$\cos(y_f, y_q)$, 当碰到y_f或y_q含有较多比例零值等不利于用余弦相似度的情况，也常将y_f按其量化系数s_y, zp_y量化后得到y_q', 再比较y_q'和y_q的差异（如$abs(y_q' - y_q)$的最大元素、非零值比例等）。

6.2 如何定位量化精度损失的瓶颈层?

首先可以对量化前后模型的相应各层输出张量计算其相似度（如余弦相似度），然后观察哪些层（或者是哪些算子类型）对应的相似度（Optimizer 会自动将相似度信息记录在输出的 *model_name*_opt_template.json 文件里）相对大幅偏低，对于重点怀疑的张量，可以进一步将量化前后的数据分布可视化加以深入分析。最后，根据相似度分析的结果，可以做模型精度指标影响的最终验证，验证方式为将疑似的瓶颈层量化前向运算时实际反量化后在浮点域上执行（在 Optimizer 上可通过 fake_quant_scopes_for_debug 参数指定相应的疑似瓶颈层），并观察模型精度是否可明显改善。

6.3 如何降低模型量化后的精度损失?

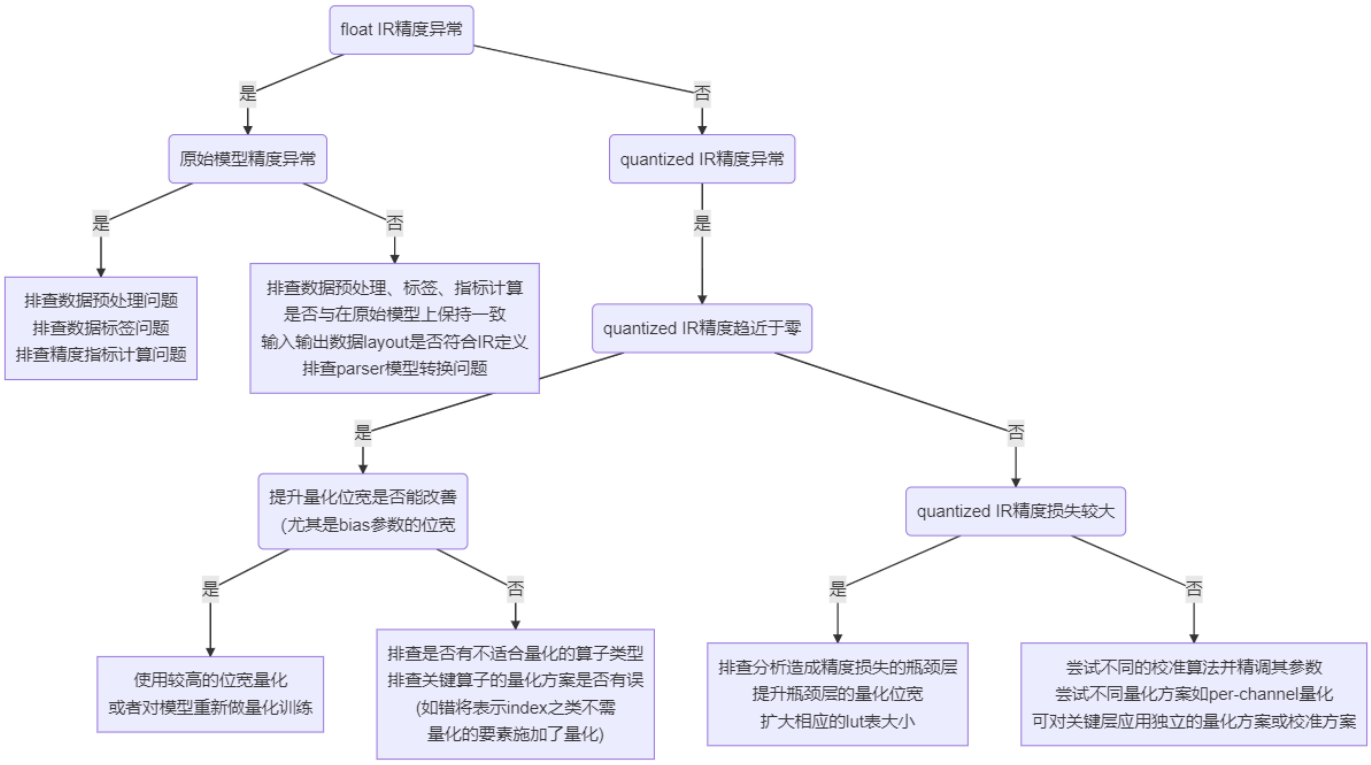
首先，如果能够获得模型对应的训练数据，则最好采用量化训练的方式，将量化潜在的误差引入训练阶段并加以优化解决。其次，如果因为种种原因无法获得训练数据，则可以针对模型应用场景挑选合适的（各类别均衡，涵盖大部分输入样本空间数值范围）无标签校准数据集，同时针对模型参数和激活响应的分布特点，选取尝试不同的校准算法并微调其参数。最后，可以分析影响量化精度的瓶颈层，并针对性提高其量化位宽（对应有使用 lut 的非线性函数运算的，还可以提升其 lut 表大小）或通过指定其 min,max 值精调其量化系数，或者直接换用精度更优的量化算法方案。

6.4 模型量化的比特位宽该如何选择?

为达到性能和精度的较好平衡，一般建议使用默认的 8bit 位宽量化，当分析出部分算子类型或模型网络中的部分层量化损失过大时，可以提升其量化位宽，经验上来说，一般较深的层产生的激活响应张量尺度较小，提升量化位宽带来的计算开销增加较少，同时对模型精度的改善贡献较大，提升其量化位宽性价比较高。

6.5 模型精度异常问题如何具体排查?

如果是在模拟器或者真实平台上执行发现精度异常，但在 Optimizer 上度量结果正确，则可以在 Optimizer 上将每层的参数和输入输出张量通过 dump 功能保存出来，逐一和目标平台上的结果对比排查。如果在 Optimizer 上度量结果异常，可以参考以下流程排查：



多数情况下，模型量化后的精度损失不会很大，通过调节校准算法参数即可进一步提升。需要注意的是，常用的校准算法都是基于校准数据集上的各种统计指标加以综合得到最终的 min,max 值，因此除了校准算法本身的参数外，影响统计指标的参数也同样可以调节，比如校准数据集的规模、batch_size、是否 shuffle、batch 间的统计量加权参数 momentum、直方图的 bins 大小等。

6.6 如何提升模型量化后的运行性能?

首先在模型量化前，参考第一章的内容，可以运用模型裁剪、模型稀疏化、计算图优化等方法获得一个尽可能精简的部署用模型。其次在量化时根据硬件特性尽量选取适当的量化位宽，同时设计量化算法时尽量减少在前向运算时引入不必要的量化相关运算（如冗余的 scale 对齐操作，冗余的多次查表操作等）。最后在算子实现时不断通过 perf 数据迭代优化相应的核心算子。