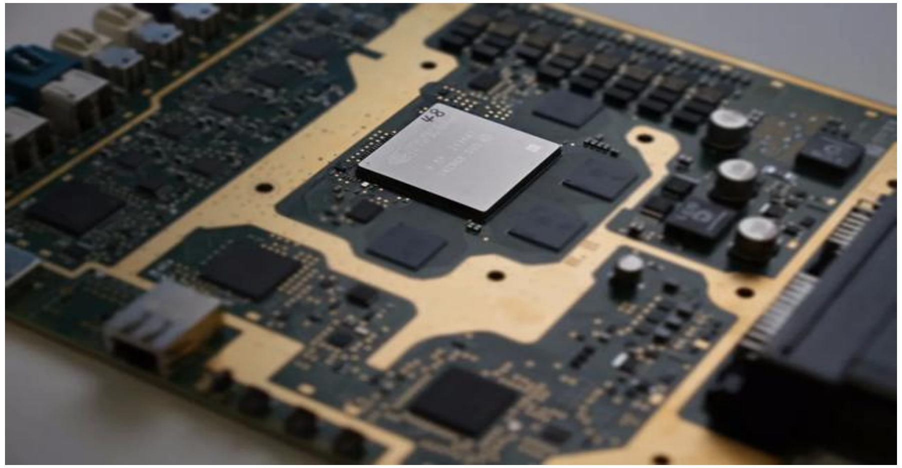
模型量化的原理与实践

——基于YOLOv5实践目标检测的PTQ与QAT量化

1、Tops是什么意思？

1TOPS代表处理器每秒可进行一万亿次（10^12）操作



公司	芯片	算力	功耗	能效比	生产制程
特斯拉	FSD芯片	144 Tops	72W	2Tops/W	14nm
英伟达	Orin	200 Tops	45W	4.5Tops/W	7nm
	Xavier	30 Tops	30W	1Tops/W	12nm
MobileEye	EyeQ5	24 Tops	10W	2.4Tops/W	7nm
	EyeQ5 Ultra	176 Tops	小于100W	约1.7Tops/W	5nm
华为	昇腾310	16 Tops	8W	2Tops/W	12nm
地平线	征程5	96Tops	20W	4.8Tops/W	16nm
黑芝麻	华山2号A1000	58 Tops	8-10W	约6Tops/W	16nm

3、什么是定点数?

大家都知道，数字既包括整数，又包括小数，如果想在计算机中，既能表示整数，也能表示小数，关键就在于这个小数点如何表示?

于是计算机科学家们想出一种方法，即约定计算机中小数点的位置，且这个位置固定不变，小数点前、后的数字，分别用二进制表示，然后组合起来就可以把这个数字在计算机中存储起来，这种表示方式叫做**定点表示法**，用这种方法表示的数字叫做**定点数**。

也就是说「**定**」是指固定的意思，「**点**」是指小数点，所以小数点位置固定的数即为**定点数**。

定点数的表示方式如下：

$$\boldsymbol{S n.m}$$

其中S表示有符号(Signed)，n和m分别代表定点数格式中的整数和小数位数，但是考虑到二进制的补码形式表示负数，总的位数为n+m+1，比如$S2.13$比对应n=2，m=13，二进制的长度为n+m+1=16，也就是说用的16位的定点化表示。

4、定点数转换

给定一个 $Sn.m$ 格式的定点数二进制形式，那么他对应的数值为：

$$\boldsymbol{x} = \boldsymbol{a} \times 2^{-\boldsymbol{m}}$$

$$\boldsymbol{a} = \boldsymbol{Binary} \Rightarrow \boldsymbol{Dec}$$

这里给出两个示例进行说明，用 $S_{10.5}$说明：

示例1：2.71875，这里通过上式知道：

$$2.71875 = a \times 2^{-5}$$

$$a = Round\left(\frac{2.71875}{2^{-5}}\right) = Round\left(\frac{2.71875}{0.03125}\right) = 87$$

这样可以得到***a*=**87的二进制值为：1010111，这里要补全前面所说的10+5+1=16位定点表示，可以知道2.71875的***S*10.5** 结果为：0,0000000010,10111

4、定点数转换

示例2：-0.499878，这里用$S_{2.13}$定点化表达：

$$-0.499878 = a \times 2^{-13}$$

$$a = Round\left(\frac{-0.499878}{2^{-13}}\right) = Round\left(\frac{-0.499878}{0.00012207\quad 03125}\right) = -4095$$

这样可以得到***a*** = -4095，4095的二进制值为：0,00,0111111111111，这里因为是有符号表示，“-”号要对4095的二进制结果进行反码+1。1,11,1000000000001。

这里也将1,11,1000000000001转换为浮点值：

由于第一位为1，因此断定是个负数，因此需要先对二进制-1，然后取反码可以得到：

~(1,11,1000000000001-1)=0,00,0111111111111

然后将其二进制转换为十进制，为4095，同时再把“-1”乘回来，便可以得到结果为-4095，然后再按照如下进行计算：

$$x = -4095 \times 2^{-13} = -0.4998779$$

可以看到定点化后精度还是比较高的。

4、定点数转换

示例3：2，这里用$S_{7.0}$定点化表达：

$$2 = a \times 2^{-0}$$

$$a = Round\left(\frac{2}{2^{-0}}\right) = 2$$

这样可以得到$\boldsymbol{a} = 2$，2的二进制值为：0,0000010。

这里也将0,0000010转换为浮点值：

然后将其二进制转换为十进制，为2，然后再按照如下进行计算：

$$\boldsymbol{x} = 2 \times 2^{-0} = 2$$

可以看到定点化后整型的定点化是没有任何精度损失的。这也是后面为什么要映射到int8整型进行计算的原因。

4、定点数转换

示例4：2.71875，这里用$S_{7.0}$定点化表达：

$$2.71875 = a \times 2^{-0}$$

$$a = \frac{2.71875}{2^{-0}} = Round\left(\frac{2.71875}{\mathbf{1}}\right) = 2$$

这样可以得到$a=2$，2的二进制值为：0,0000010。

这里也将0,0000010转换为浮点值：

然后将其二进制转换为十进制，为2，然后再按照如下进行计算：

$$x = 2 \times 2^{-0} = 2$$

可以看到定点化后随着定点位置越小精度就越大。这里因为$S_{7.0}$定点分辨率就是1，因此小数点后的结果就会被直接舍弃。

后面的线性映射便可以解决这样的问题。

2、什么是量化?

通过前面的浮点定点化到不同的数据格式可以看出，使用不同的定点化表达数据会带来不同的浮点精度损失，那我们又应该如何缓解这种定点化带来的误差呢?

能不能做个过渡的过程呢? 比如，在浮点数进行定点化之前能否先进行一次缩放操作呢? 这里我们将缩放操作看作一次简单的线性映射过程，依旧以定点转换示例4来进行操作:

首先直接把2.71875作为浮点的最大值，这里依旧采用$S_{7.0}$的定点计算方式（也就是int8的计算方式），其表示范围也就是[-128,127]之间，这里使用线性映射来进行映射:

1、计算线性映射的缩放值Scale

Scale=2.71875/127=0.0214075

2、根据映射关系计算整型结果:

Float=Scale×Quant

Quant=Float/Scale

Quant=2.71875/0.0214075=127

3、将整型127进行$S_{7.0}$定点化:

127==> $S_{7.0}$==>01111111==>Dec==>127

4、整个流程（量化的原理示意）:

量化: 2.71875/0.0214075==>127==> $S_{7.0}$==>01111111

反量化: 01111111==>Dec==>127×0.0214075==>2.71875

2、什么是量化?

量化

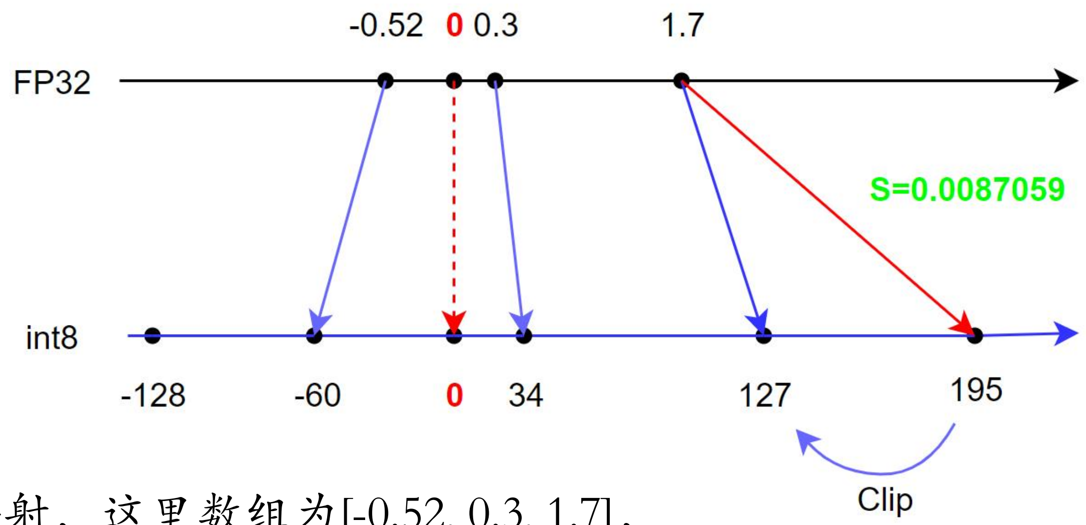
$$Q = Round\left(\frac{R}{Scale}\right)$$

$$Scale = \frac{R_{max} - R_{min}}{Q_{max} - Q_{min}}$$

反量化

$$R = Q \times Scale$$

$$Scale = \frac{R_{max} - R_{min}}{Q_{max} - Q_{min}}$$



这里我们用映射的方法来进行一个数组的映射，这里数组为[-0.52, 0.3, 1.7]，

- 首先，通过求组数组的最大最小值之差然后除以int8的表示范围可以得到Scale=0.0087059
- 然后，根据Scale和上式的量化映射方法，可以得到映射后的int8数组为[-60, 34, 195]；
- 最后，由于int8的定点表示范围为[-128, 127]，因此对于前面得到的结果需要进行Clip截断操作，将[-60, 34, 195]截断为[-60, 34, 127]。

前面得到了映射后的数组为:[-60, 34, 127]，这里根据反量化表达式与Scale的值进行反量化计算可以得到，上述结果反量化后的结果为： [-0.52236, 0.296, 1.1065]，而原始的数据为[-0.52,0.3,1.7]，可以看到由于截断带来的精度损失还是比较大的。

怎么解决这个问题呢?

答：1、偏移；2、最大绝对值对称法

2、什么是量化?

量化

$$Q = Round\left(\frac{R}{Scale} + Z\right)$$

$$Scale = \frac{R_{\max} - R_{\min}}{Q_{\max} - Q_{\min}}$$

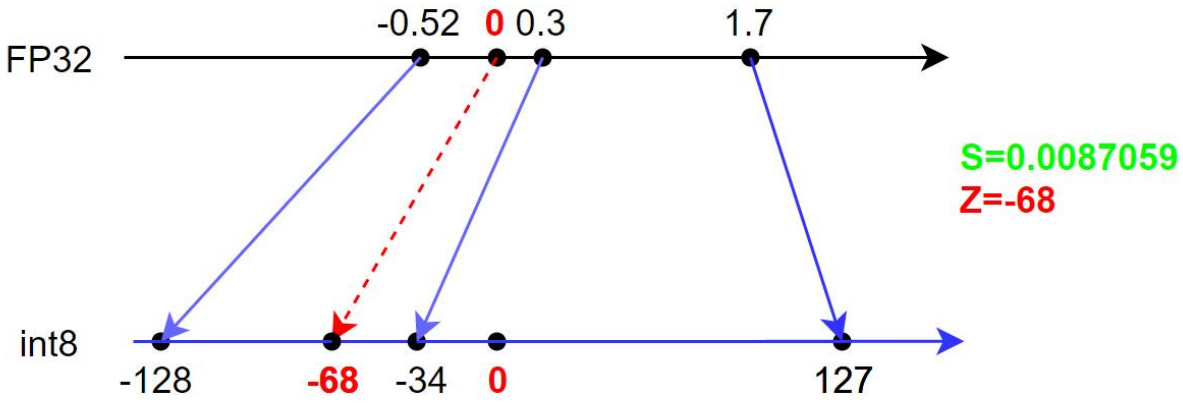
$$Z = Q_{\max} - Round\left(\frac{R_{\max}}{Scale}\right)$$

反量化

$$R = Q \times Scale + Z$$

$$Scale = \frac{R_{\max} - R_{\min}}{Q_{\max} - Q_{\min}}$$

$$Z = Q_{\max} - Round\left(\frac{R_{\max}}{Scale}\right)$$



解决方案1：偏移法

通过前面的问题可以知道，我们Clip截断的部分其实就是浮点最大值量化后超过int8最大表示范围的值，因此我们直接向左平移68姐可以得到无需截断的结果。

通过偏移68，得到的int8结果是:[-60, 34, 195]-68 = [-128,-34,127]

其实这个结果就是所谓的Z-Point，也就是浮点值的0点通过偏移在int8量化表示范围中的位置；

Z-Point的计算方式也很简单：

$$Z = Q_{\max} - Round\left(\frac{R_{\max}}{Scale}\right)$$

2、什么是量化?

量化

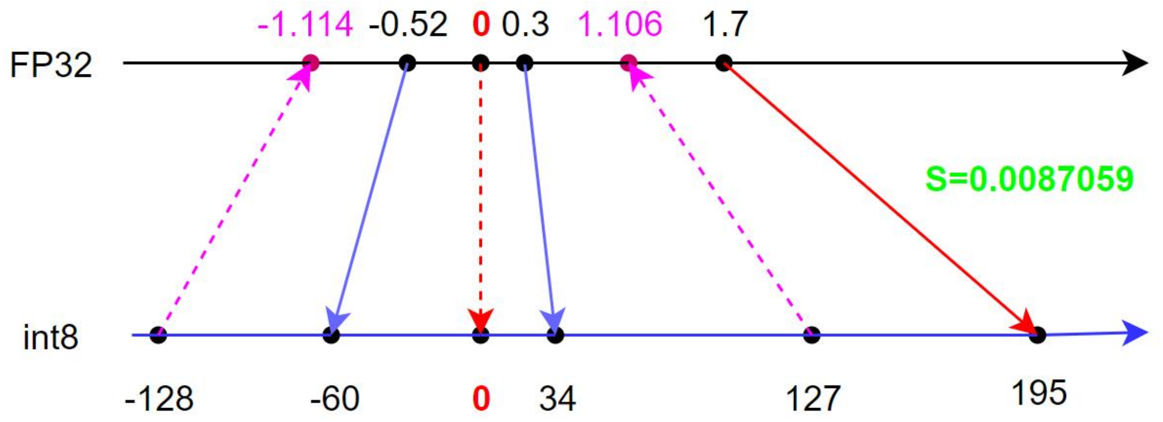
$$Q = Round\left(\frac{R}{Scale}\right)$$

$$Scale = \frac{|R_{max}|}{|Q_{max}|}$$

反量化

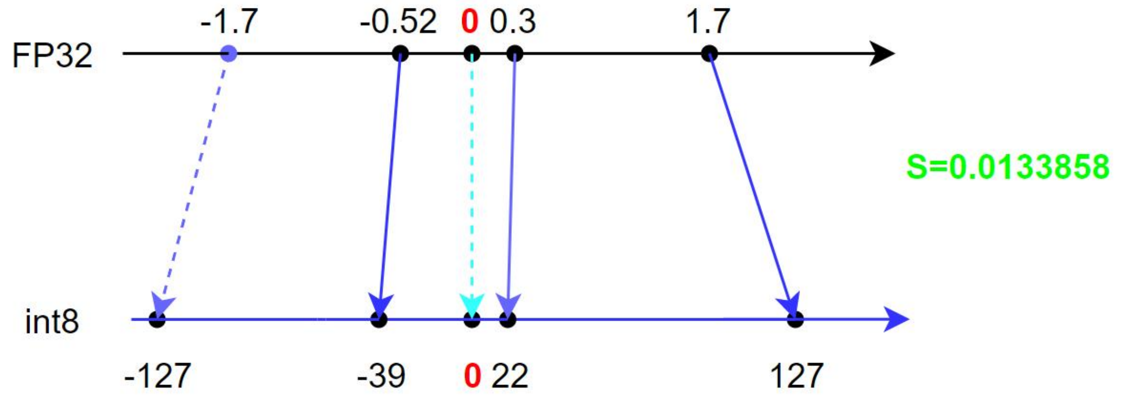
$$R = Q \times Scale + Z$$

$$Scale = \frac{|R_{max}|}{|Q_{max}|}$$



解决方案2：最大绝对值法

这里先分析一下为什么会被截断？通过将127和-128与Scale进行反量化得知，该方法需要保证正浮点数不大于1.106，负浮点数不小于-1.114，而1.7则是大于零这个边界，因此需要强行拉回到127边界，这里拉多少呢？其实也很简单，就是Round((1.7-1.106)/s)=68



换句话说，这里如果将-128舍弃，直接用对称结构是不是就不会出现这种对不齐的问题？话不多说直接尝试：

直接去1.7为最大值，然后对称到负边界，而此时的int8负边界已经舍弃了-128，因此Scale=(2*1.7/254)

然后直接计算int8的结果为：[-39, 22, 127]。

4、定点计算

1、首先，对随机生成的浮点数据进行量化，转换为定点数据；
2、将原始输出，量化后的数据，反量化后的数据进行对比并计算误差；
3、对比量化前后的数据内存占用情况；
4、通过数据自身求和 n 次来测试运算速度

```python
import sys
import time
import numpy as np

# 随机生成一些浮点数据（Float32）
data_float32 = np.random.randn(10).astype('float32')

# 量化上下限（UInt8）
Qmin = 0
Qmax = 255

# 计算缩放因子（Scale）
S = (data_float32.max() - data_float32.min()) / (Qmax - Qmin)

# 计算零点（Zero Point）
Z = Qmax - data_float32.max() / S

# 将浮点数据（Float32）量化为定点数据（UInt8）
data_uint8 = np.round(data_float32 / S + Z).astype('uint8')

# 将定点数据（UInt8）反量化为浮点数据（Float32）
data_float32_ = ((data_uint8 - Z) * S).astype('float32')
```

4、定点计算——误差计算

```
# 使用均方误差计算差异
mse = ((data_float32-data_float32_)**2).mean()

print("原始数据：", data_float32)
print("反量化后数据：", data_float32_)
print("量化后数据：", data_uint8)
print("原始数据和反量化后数据的均方误差：", mse)
```

```
原始数据：
[-0.47069794  0.8608386  -0.947035   -1.0697421   0.10035864
  0.0013437 0.6782814   0.7141242  -0.09668107  0.4391506 ]

反量化后数据：
[-0.4716406   0.8608386  -0.94860756 -1.069742   0.10374816
 -0.00224451 0.6791369   0.7169914  -0.09309536  0.43686795]

量化后数据：
[ 79 255  16   0 155 141 231 236 129 199]

原始数据和反量化后数据的均方误差： 5.4746083e-06
```

4、定点计算——内存对比

```python
# 空数组的内存占用
empty_size = sys.getsizeof(np.array([]))

# 计算实际数据的内存占用
float32_size = (sys.getsizeof(data_float32) - empty_size)
uint8_size = (sys.getsizeof(data_uint8) - empty_size)

print("原始数据内存占用：%d Bytes" % float32_size)
print("量化后数据内存占用：%d Bytes" % uint8_size)
print("量化后数据与原始数据内存占用之比：", uint8_size / float32_size)
```

```
原始数据内存占用：40 Bytes
量化后数据内存占用：10 Bytes
量化后数据与原始数据内存占用之比： 0.25
```

4、定点计算——速度对比

```
原始数据求和 10000 次耗时 : 0.017538 s
量化后数据求和 10000 次耗时: 0.008749 s
量化后数据与原始数据计算耗时之比: 0.49884448069603043
```

从上面的实例中可以看出量化操作的几个优点:

1、计算快，效率高，计算时的耗能降低;
2、内存占用小，方便数据文件的储存和传输;

当然缺点也是有的，就是计算精度会有一定程度的下降。

```python
# 预热次数
warmup = 100
# 重复次数
repeat = 10000
# 预热
sum = data_float32
for i in range(warmup):
    sum += data_float32

sum = data_uint8
for i in range(warmup):
    sum += data_uint8

# 速度测试
start = time.time()
sum = data_float32
for i in range(repeat):
    sum += data_float32
float32_time = time.time() - start

start = time.time()
sum = data_uint8
for i in range(repeat):
    sum += data_uint8
uint8_time = time.time() - start

print("原始数据求和 %d 次耗时 : %f s" % (repeat, float32_time))
print("量化后数据求和 %d 次耗时: %f s" % (repeat, uint8_time))
print("量化后数据与原始数据计算耗时之比: ", uint8_time / float32_time)
```

2、量化有什么优缺点?

作用：将FP32的浮点计算转化为低bit位的计算，从而达到模型压缩和运算加速的目的。比如int8量化，就是让原来32bit存储的数字映射到8bit存储。int8范围是[-128,127]（实际工程量化用的时候不会考虑-128），uint8范围是[0,255]。

模型量化优点：

1. 加快推理速度，访问一次32位浮点型可以访问4次int8整型，整型运算比浮点型运算更快；
2. 减少存储空间，在边缘侧存储空间不足时更具有意义；
3. 减少设备功耗，内存耗用少了推理速度快了自然减少了设备功耗；
4. 易于在线升级，模型更小意味着更加容易传输；
5. 减少内存占用，更小的模型大小意味着不再需要更多的内存；

模型量化缺点：

1. 模型量化增加了操作复杂度，在量化时需要做一些特殊的处理，否则精度损失更严重；
2. 模型量化会损失一定的精度，虽然在微调后可以减少精度损失，但推理精度确实下降；