【模型推理】谈谈为什么卷积加速更喜欢 NHWC Layout

本文主要讨论一下为什么卷积加速更加喜欢 NHWC 的数据排布。

我目前接触过的数据排布类型(主要针对卷积)有 NCHW (pytorch、caffe)，NHWC (Tensorflow，也是 TVM GPU 和 寒武纪 MLU Core 上更喜欢的 data Layout)， CHW (TensorRT里不考虑动态batch的话是把 N 拿出来了，只剩三维)，NCHWC0 (华为昇腾 AI Core 的五维 Layout，C0 INT8时为 32，FP16时为16)。为什么会有这么多数据类型排布呢，原因可能是源于不同的训练框架，比如 pytorch 和 tensorflow (大部分人的炼丹炉) 就不一样，而在推理时，更多的会考虑硬件 / 推理性能更喜欢什么样的数据类型排布。

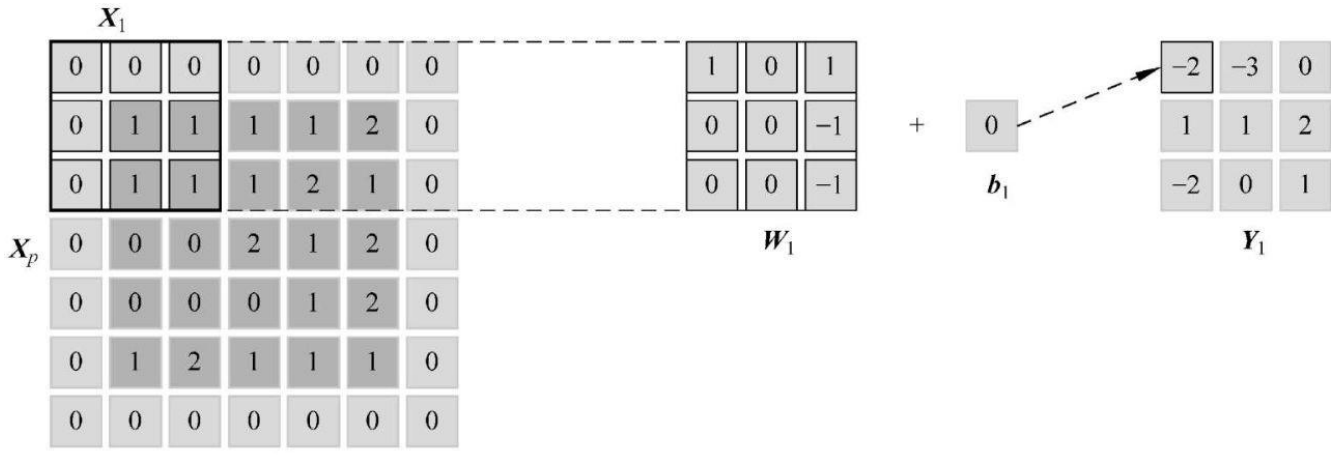
这里主要谈谈对于 img2col+gemm 和 winograd 卷积加速算法来说，为什么 NHWC 比 NCHW 更合适 (GPU上)，也是我个人的理解。

1、img2col+gemm 和 winograd 算法原理

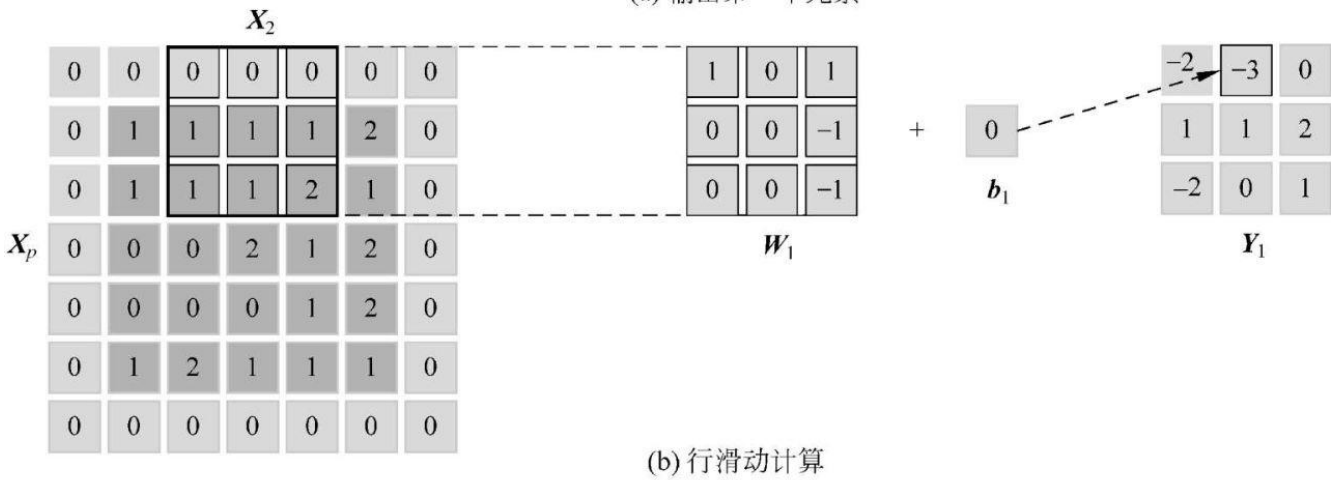
img2col+gemm 的详细原理可以看我的这篇《【模型推理】一文看懂Img2Col卷积加速算法》，就是先把 feature map 按卷积核走过的 "踪迹" 展开、拼接 与 拉扁拉长的卷积核 进行 gemm 就好了。winograd 应该是比较常用的，Cudnn 里也会用到，这里大致说一下，后面会写篇详细的。winograd 做 conv_2d_3x3 加速的本质是降低乘加计算次数 (主要是乘)，最开始还是会做 img2col，然后进行计算切块、转换及卷积核常量先计算，最终达到降低乘加次数的目的，也就是能加速。

2、为什么 NHWC 更好

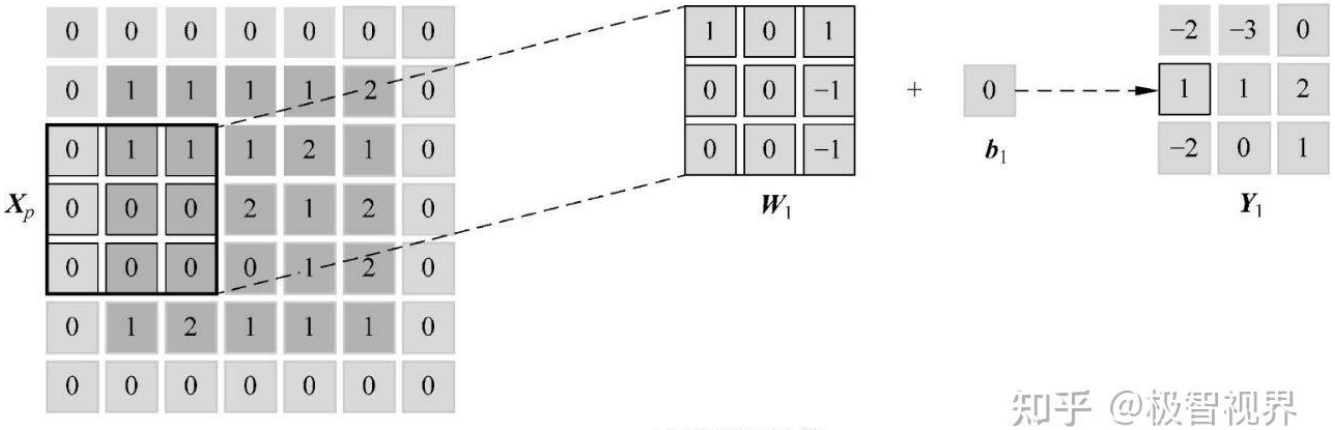
我的观点是 NHWC 相比 NCHW 优化了 feature map 数据取的过程，因为不管 img2col+gemm 还是 winograd，最开始就是把 feature map 和 kernel 展开，类似这样：



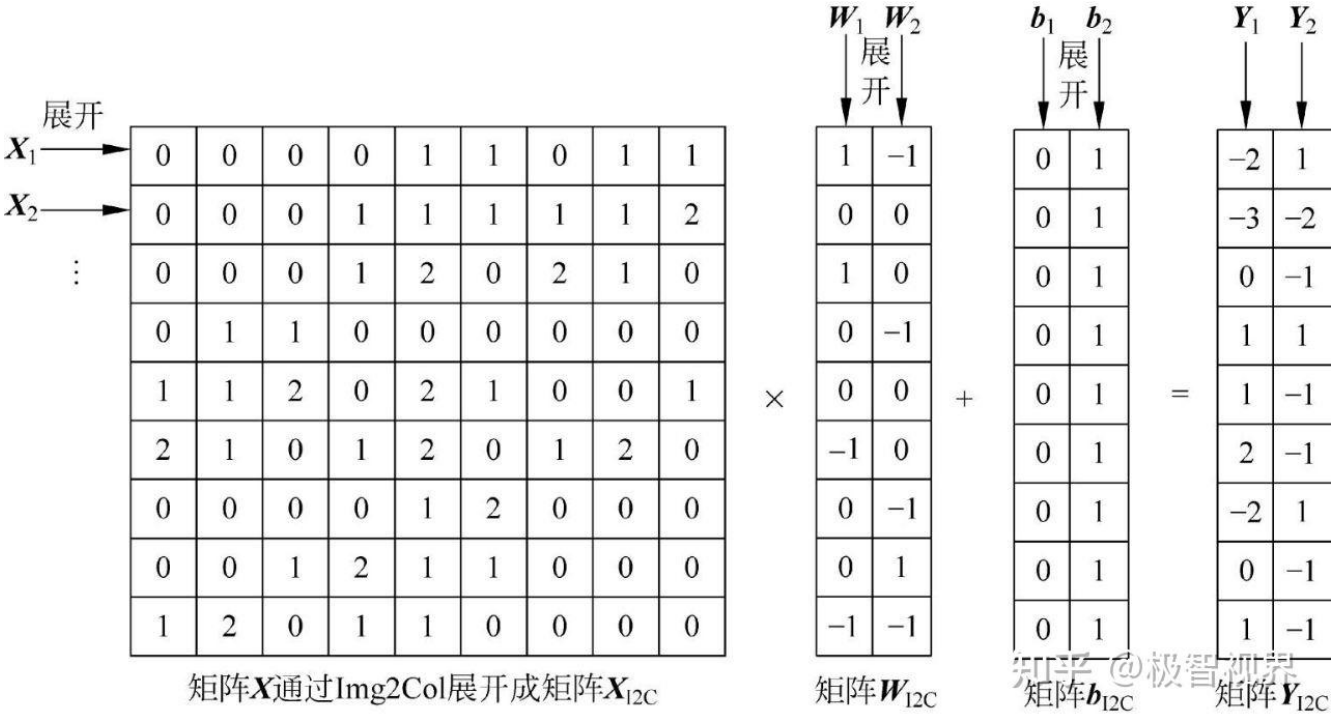
(a) 输出第一个元素



(b) 行滑动计算



(c) 列滑动计算



卷积过程最开始会涉及到你是怎么取 feature map 中卷积块数据的问题。NCHW 的实际存储方式是 "RRRGGGBBB"，同一通道的所有像素存储在一起；而 NHWC 的实际存储方式是 "RGBRGBRGB"，多个通道的同一位置的像素值顺序存储在一起。（本来想画个图的，懒得画了哈哈，用代码替代吧）

```
# feature map NCHW 通道C=3
# 通道C=0         通道C=1         通道C=2
a0 b0 c0 d0    a1 b1 c1 d1   a2 b2 c2 d2
e0 f0 g0 h0    e1 f1 g1 h1   e2 f2 g2 h2
i0 j0 k0 l0    i1 j1 k1 l1   i2 j2 k2 l2
m0 n0 o0 p0    m1 n1 o1 p1   m2 n2 o2 p2

# kernel
A0 B0 C0        A1 B1 C1      A2 B2 C2
D0 E0 F0        D1 E1 F1      D2 E2 F2
G0 H0 I0        G1 H1 I1      G2 H2 I2
```

如果用 NCHW 的 Layout，行主序存储（row major）来说，feature map 的数据存储方式为 a0b0c0d0e0f0g0h0i0j0k0l0m0no0p0a1b1c1d1e1f1g1h1i1j1k1l1m1n1o1p1a2b2c2d2e2f2g2h2i2j2k2l2m2n2o2p2。以 3_x_3 的卷积核，对于一次卷积动作来说，feature map 需要 n * kernel_size 次数据取动作，分别是 a0b0c0、e0f0g0、i0j0k0、a1b1c1、e1f1g1、i1j1k1、a2b2c2、e2f2g2、i2j2k2，这里 n=3, kernel_size=3 就是 **9** 次取数据。

下面来看看 NHWC 的情况，如下

```
# feature map NHWC C=3
      a2 b2 c2 d2
   a1 b1 c1 d1 h2
a0 b0 c0 d0 h1 l2
e0 f0 g0 h0 l1 p2
i0 j0 k0 l0 p1
m0 n0 o0 p0
```

把上面的想象成长方体，画的是三维的，哈哈，能想象吗，直观上看就是第一个平面是第一个通道，也就是 C=0，往后第二个平面是第二个通道 C=1，最后是第三个通道 C=2，有点抽象哈哈。kernel 还是一样，如下：

```
# kernel
A0 B0 C0        A1 B1 C1      A2 B2 C2
D0 E0 F0        D1 E1 F1      D2 E2 F2
G0 H0 I0        G1 H1 I1      G2 H2 I2
```

对于 NHWC Layout 来说，feature map 的数据存储方式为 a0a1a2b0b1b2c0c1c2d0d1d2e0e1e2f0f1f2g0g1g2h0h1h2i0i1i2j0j1j2k0k1k2l0l1l2m0m1m2n0n1n2o0o1o2p0p1p2。以 3_x_3 卷积，对于一次卷积动作来说，feature map 只需要 **3** 次数据取动作，分别是 a0a1a2b0b1b2c0c1c2、e0e1e2f0f1f2g0g1g2、i0i1i2j0j1j2k0k1k2。这样，仅一个卷积动作，NHWC 就比 NCHW 减少了 **6** 次数据取操作。

分析得出，对于一次卷积动作来说，NHWC 取数据的次数为 kernel_size 次，而 NCHW 取数据的次数为 kernel_size * n 次，所以 NHWC 对于卷积加速数据访存来说是更好的，而且这种好，随着 n 的增大会更更加好。

收工了~

【csdn 传送】