layout: post
title: "FlOPs低，模型速度不一定快"
date: 2020-07-04
description: "深度学习实践"

tag: 深度学习实践

前一段时间在做图像分类的应用时，发现了一个问题，EfficientNet的网络模型是真的小，但是模型的推理速度却非常慢，一直也不懂这是咋回事，偶然看到极市公众号的文章，这里记录一下，参考链接：https://mp.weixin.qq.com/s/Wwb6k6_Wqo2nM_dDyky2cg

EfficientNet很低的FLOPs却伴随着较高的推理时间，比如B3版本的FLOPs不到ResNet50的一半，推理速度却是ResNet50的两倍。

FLOPs与模型推理速度的关系

大部分时候，对于GPU，算力瓶颈在于访存带宽。而同种计算量，访存数据量差异巨大

EfficientNet这样网络的特点，就是使用了大量的**低FLOPs、高数据读写量**的操作，更具体来说，就是**depthwise卷积**操作。这些具有高数据读写量的操作，加上GPU的访存带宽限制，使得模型把大量的时间浪费在了从显存中读写数据上，GPU的算力没有得到“充分利用”。

在ShuffleNet V2中已经提出，ShuffleNet V2看到了**GPU访存带宽对于模型推理时间**的影响，而不仅仅是模型复杂度，也就是**FLOPs和参数量对于推理时间**的影响。ShuffleNet V2把**depthwise卷积归为element-wise**操作，同时指出**element-wise操作具有低FLOPs，高推理时间**的特点。

depthwise卷积和普通卷积的区别，为什么depthwise卷积是低FLOPs，高推理时间，为什么ShuffleNet V2把depthwise卷积称为element-wise操作是合理？

RegNet中提出了**network activations**的概念，注意这里的activation跟ReLU这种激活函数不同，这里activation指的是**网络卷积层输出的tensor大小之和**，作者认为这一指标更能检验模型的推理速度。不难看出，其实这个输出的tensor大小，就可以看作模型进行推理时，需要从显存中读取的feature blob的大小，近似可以认为是访存数据量的大小。

那么我们不妨从activation这个角度看看EfficientNet，看看depthwise卷积与普通卷积的区别。二者在**同等FLOPs下，activation有什么不同**？ 这种不同为什么又会导致推理速度的不同？

为简化处理，以下内容只计算乘法，不计算加法。

假设一个大小为`56*56*100`的`feature（H*W*C）`，经过一个kernel size为`3x3`的普通卷积layer，卷积layer的输出channel也是100。其FLOPs计算过程如下：

```
一个卷积kernel的大小为：3*3*100，与feature上一个同等大小的blob进行卷积，这是一个逐元素点乘操作，总共有3*3*100次乘法。
然后卷积layer输出channel是100，说明有100个这样的卷积kernel，同时在feature的空间位置上，每个像素点都要重复一次卷积操作，共56*56次，
所以总的FLOPs为：3*3*100*100*56*56。卷积核参数总量为：3*3*100*100。
```

然后为了达到同样的FLOPs，我们假设另一个大小为`56*56*10000`的feature，经过一个kernel size为`3x3`的depthwise卷积layer，卷积layer的输出channel是10000。其FLOPs计算过程如下：

```
一个卷积kernel的大小为：3*3*1，与feature上一个同等大小的blob进行卷积，总共有3*3次乘法。
然后10000个channel通道，每个通道互相独立，对应着10000个不同的卷积kernel，所以重复这一卷积过程10000次。
同时在feature的空间位置上逐元素重复，总的FLOPs为：3*3*10000*56*56。卷积核参数总量为：3*3*1*10000。
```

可以看到，两个layer的FLOPs和参数量完全相同。但是推理速度方面，depthwise卷积要远远慢于普通卷积。其原因就是访存数据量的不同：

由于卷积计算本身已经是`flatten`的，不需要考虑重复读取问题，那么总共读取的数据量就是feature的大小加上卷积核weight的大小，
对于普通卷积来说，总读取数据量为：`100*56*56 + 3*3*100*100 = 4.0e+05`。
类似的，depthwise卷积读取的数据总量为：`56*56*10000 + 3*3*10000 = 3.1e+07`。

可以看到，在同等FLOPs的情况下，depthwise卷积对应的feature size比普通卷积大的多，受制于GPU访存带宽，过高的数据读取与写入量就成为了限制推理速度的瓶颈。

然后这与element-wise操作有什么关系呢？

与element-wise相对应的，其实是矩阵乘法操作，矩阵乘法操作的特点是**数据复用**。假设一个`100*1`的矩阵与`1*100`的矩阵相乘，总共`10000`次乘法，总共的数据量只有`2*100`。因为每一个元素都要参与100次乘法，大量的数据存在复用；而换做element-wise操作，10000次乘法就是10000维的向量与另一个10000维的向量进行点乘，10000次乘法互相独立，不存在任何数据复用。

而在网络层面，普通卷积操作都可以看作**矩阵乘法**，存在着数据复用。在上面提到的3x3普通卷积中，一个33100的feature blob会跟所有的卷积核相乘，每一个33100的卷积核又会与所有的feature blob相乘。而ReLU这样的激活函数，ResNet block中的shortcut等等，都是elment-wise操作，逐元素求取sigmoid/逐元素相加。这些过程中每次操作之间是不存在数据复用的。而depthwise卷积也有这样的特点，可以称作**channel-wise**或者说**kernel-wise**。depthwise卷积中每一个channel对应着不同的卷积核和feature blob，每次卷积操作之间不牵涉“数据复用”，因此从这个角度，可以说depthwise卷积某种程度上说也是一种“element-wise”操作。

我们再回头看EfficientNet，不难看出其中的“取巧”成分。数据访存量与feature size（RegNet中的activation）有关，而feature size又与空间尺寸以及channel通道数（或者换句话说，网络的宽度width）有关。**EfficientNet的一个核心就是增大空间尺寸或者网络宽度width以提升模型精度**。

由于depthwise卷积的存在，增大feature的空间尺寸，或者channel通道数（width）都不会显著地增加FLOPs。因此EfficientNet可以声称自己是低FLOPs，但不得不说，这是一种“FLOPs假象”。因为**feature size的增大会增加数据访存量，进而增加模型推理时间**，这是单纯的FLOPs反映不出来的。

RegNet的附录部分有一些对于depthwise卷积，以及swish激活函数的相关实验，虽然只有结论没有讨论，也是很值得一看，比如swish和depthwise卷积很搭，其他的一般

比如depthwise卷积在各种FLOPs区间都没有体现出相较于普通卷积的优势等等。

同时略“讽刺”的是，同等FLOPs情况下，RegNet跟EfficientNet比较“推理速度”，确实提升了5倍，但这其实相当于利用depthwise卷积低FLOPs高数据访存量的弱点，反过来打EfficientNet。

当然RegNet那些“炼金术公式”一般的结论也是挺有趣的，仿佛在观察NAS的炼金世界，然后找寻“自然规律”。

一个学术论文与工业界需求偏差的地方

好多使用attention的网络，比如x=x*sigmoid(x)，实际上需要把tensor拷贝一次，这其实增大了显存占用，而显存占用是影响工业界实际应用的。因为工业界考虑的不是FLOPs，甚至也不是单纯的inference time，考虑的是把一块儿GPU打满情况下的QPS（queries per second）。用图片分类为例，就是

一块儿GPU打满，这块儿GPU每秒钟能处理多少张图片。

而好多学术论文，测试inference time，一般都是用一个比较小的batchsize，同时为了“公平”，多个model比较还要用一样的batchsize。这是不对的，对占用显存少的网络不公平啊，ResNet说我明明能同时处理更多的图片，为什么不让？

举例说明，我刚刚测试了一下，batchsize=16，输入图片224x224（注意实际的EfficientNet-b3用的是300x300图片作为输入。。），

```
EfficientNet-b3，batchsize=16，测试QPS=268；
ResNet50，batchsize=16，测试QPS=416。
416/268=1.55
```

P40，24G显存，打满的情况呢

```
EfficientNet-b3，打满P40，batchsize=600，测试QPS=296；
ResNet50，打满P40，batchsize=1700，测试QPS=511。
511/296=1.73
```

嗯，套路大概就是这样。