推理引擎-模型转换与优化

计算图优化

ZOMI

Talk Overview

1. **推理系统介绍**
   - 推理系统架构
   - 推理引擎叫故
2. **模型小型化**
   - CNN小型化结构
   - Transform小型化结构
3. **离线优化压缩**
   - 低比特量化
   - 模型剪枝
   - 知识蒸馏
4. **模型转换与优化**
   - 架构与流程
   - 模型转换技术
   - 计算图优化
5. **Runtime与在线优化**
   - 动态batch
   - bin Packing
   - 多副本并行

Talk Overview

1. 计算图优化

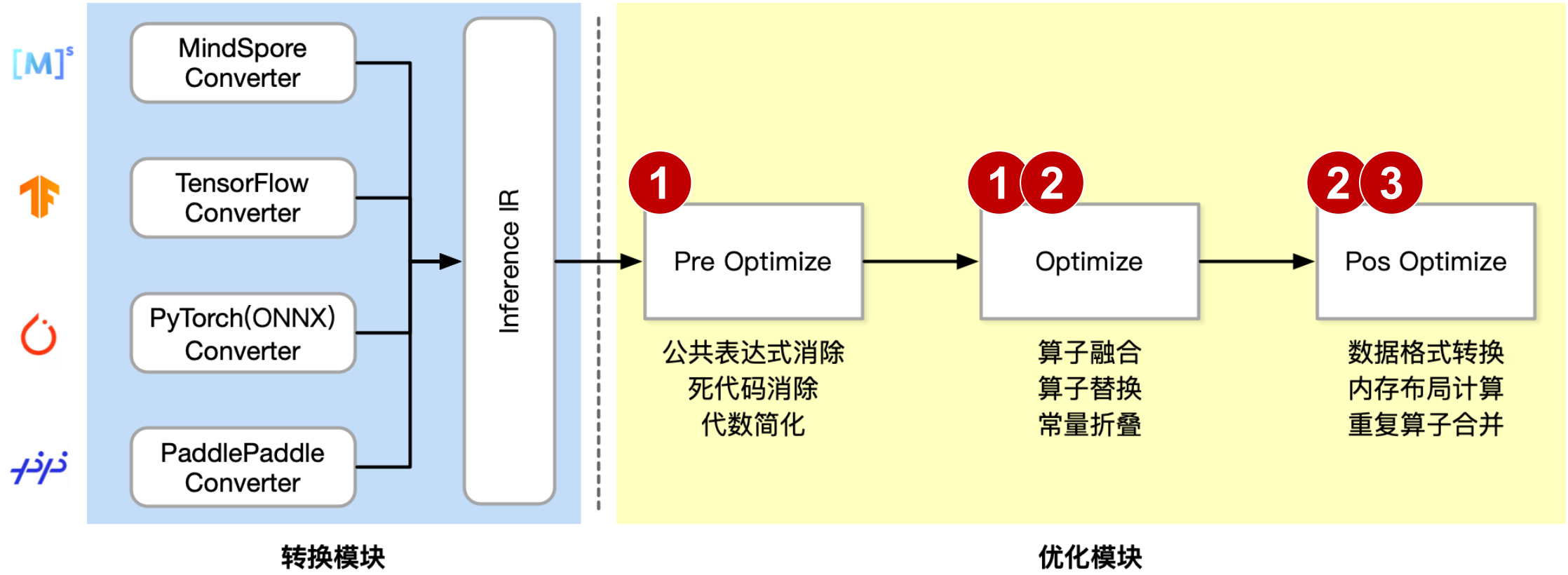
- Challenges and Architecture - 挑战与架构
- graph Optimization - 计算图优化
- Example - ONNX Runtime 图优化
- Optimize Details - 计算图优化详解

- **代数简化**
- **算子优化**

图优化方式

1. **Basic:** 基础优化涵盖了所有保留计算图语义的修改，如：O1常量折叠、O2冗余节点消除和O3有限数量的算子融合。
2. **Extended:** 扩展优化仅在运行特定后端，如 CPU、CUDA、NPU 后端执行提供程序时适用。其针对硬件进行特殊且复杂的 Kernel 融合策略和方法。
3. **Layout & Memory:** 布局转换优化，主要是不同 AI 框架，在不同的硬件后端训练又在不同的硬件后端执行，数据的存储和排布格式不同。

工作流程



AI编译器之前端优化

播放全部

结构在快速演化，底层计算硬件技术更是层出不穷，对于广大开发者来说不仅要考虑如何在复杂多，还要应对计算框架的持续迭代。AI编译器就成了应对以上问题广受关注的技术方向，让用户仅...

默认排序　升序排序　编辑

AI编译器前端"图层优化"内容概览！！【AI编译器】系列之前端优

600　2022-12-13

图层IR(Graph IR)是什么？AI编译器如何接收图层IR进行优化呢？【AI编

519　2022-12-13

算子融合了解下！AI编译器如何实现算子融合的？【AI编译器】系列

915　2022-12-16

编译器为什么要对数据布局转换呢Layout Transformations？【AI编

404　2022-12-17

详解AI编译器数据布局转换方法！华为昇腾处理器的数据布局格式！

225　2022-12-17

AI编译器内存优化算法！动态内存和静态内存区别！【AI编译器】前

328　1-16

常量折叠原理！AI编译器常量折叠跟传统编译器什么关系？计算图也

356　2022-12-18

编译器公共子表达式消除的方法！AI编译器消除公共子表达式【AI编译

270　2022-12-20

编译器死代码消除的原理！AI编译器死代码消除【AI编译器】系列之

375　2022-12-20

编译器的代数化简原理！AI编译器的代数化简来啦！【AI编译器】系

315　2022-12-21

计算图优化 (a.k.a., 图优化)

- **图优化**：基于一系列预先写好的模板，减少转换模块生成的计算图中的冗余计算，比如 Convolution 与 Batch Normal / Scale 的合并，Dropout 去除等。图优化能在特定场景下，带来相当大的计算收益，但相当依赖根据先验知识编写的模板，相比于模型本身的复杂度而言注定是稀疏的，无法完全去除结构冗余。

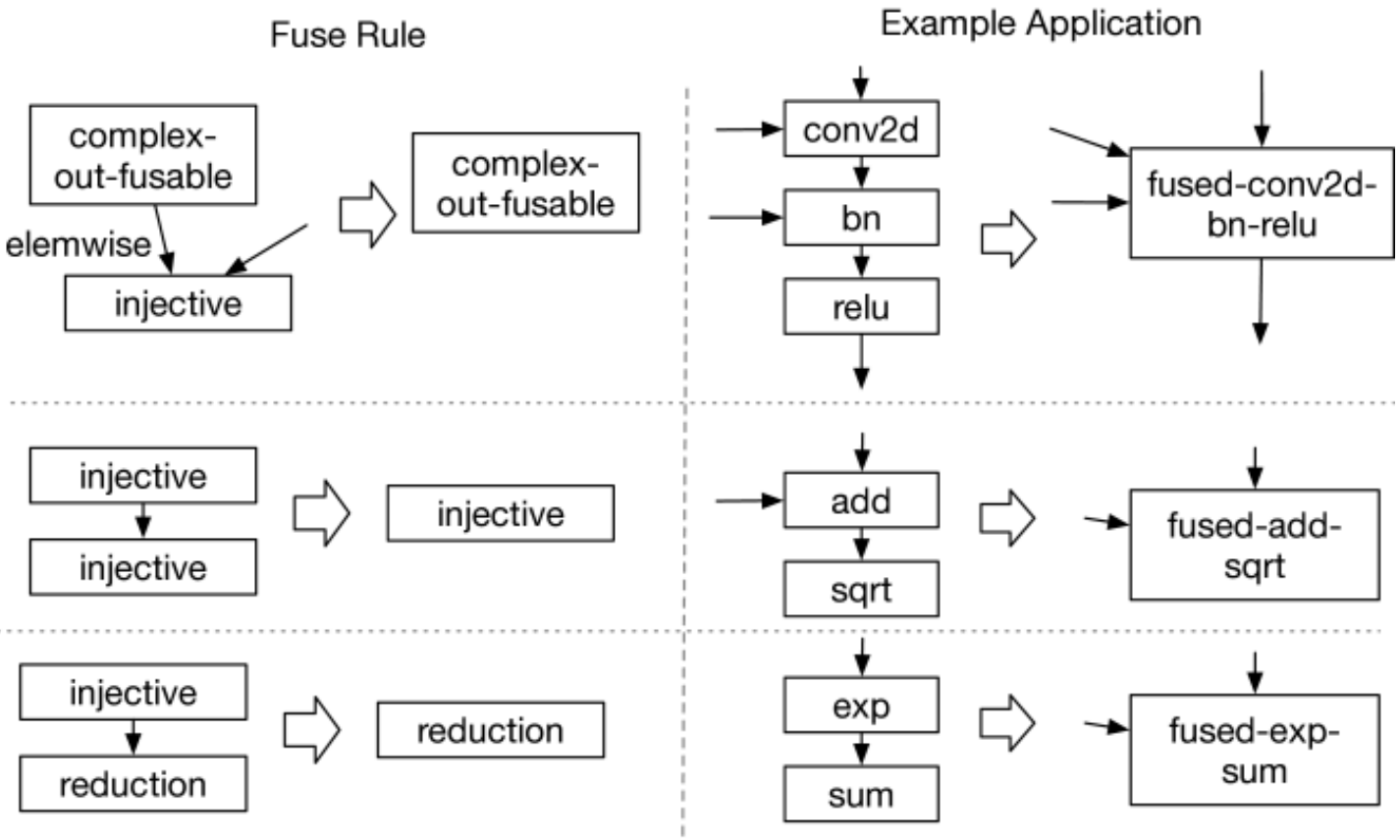
TVM 算子融合流程

1. 通过AST转换为Relay IR，遍历Relay IR；
2. 建立DAG用于后支配树分析；
3. 应用算子融合算法

3. 遍历每个Node到它的支配带你的所有路径是否符合融合规则，完成融合后，遍历节点创新的DAG图

Rules for operator fusion

- Injective(one-to-one map)：映射函数，如Add，Pointwise
- Reduction：约简函数，输入到输出具有降维性质，如sum/max/min
- Complex-out-fusable：an fuse element-wise map to output，计算复杂类型，如conv2d
- Opaque(cannot be fused) ：无法被融合的算子，比如sort



Question?

- 为什么AI框架或者AI编译器的图优化采用基于规则树或者特殊IR树的方式进行优化融合，
- 而推理引擎的图优化采用 Hard Code 或者模板匹配呢？

计算图优化详解 (I)

1 Basic Graph Optimizations 基础图优化

- These are semantics-preserving graph rewrites which remove redundant nodes and redundant computation. They run before graph partitioning and thus apply to all the execution providers. 基础优化涵盖了所有保留语义的修改，如常量折叠、冗余节点消除和有限数量的节点融合。

1. Constant folding 常量折叠
2. Redundant eliminations 冗余节点消除
3. Operation fusion 算子融合
4. Operation Replace 算子替换
5. Operation Forward 算子前移

计算图优化

常量折叠

1 O1: Constant Folding 常量折叠

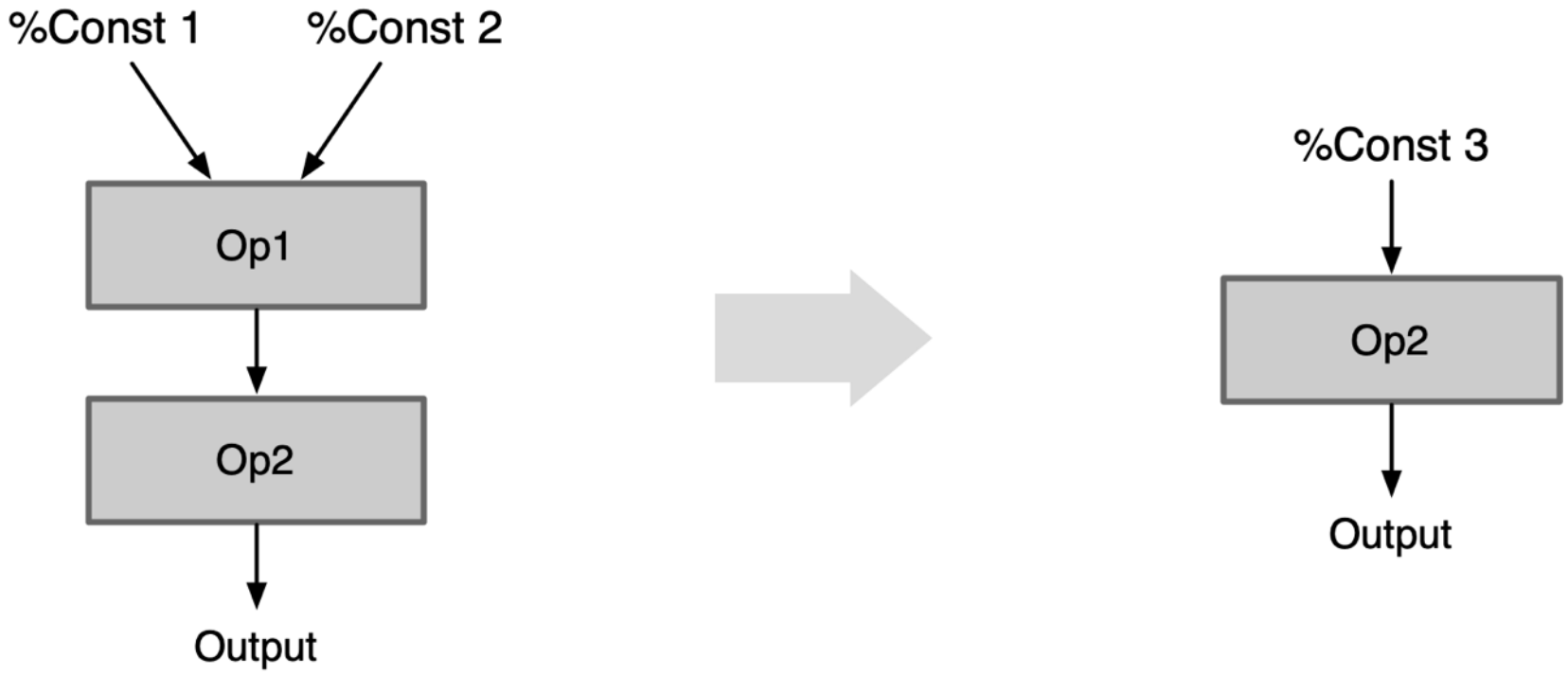
- Constant folding，常量折叠，编译器优化技术之一，通过对编译时常量或常量表达式进行计算来简化代码。
- Statically computes parts of the graph that rely only on constant initializers. This eliminates the need to compute them during runtime. 常量折叠是将计算图中可以预先可以确定输出值的节点替换成常量，并对计算图进行一些结构简化的操作。

1 O1: Constant Folding 常量折叠

Constant Folding	Const 折叠	• 常量折叠，如果一个 Op 所有输入都是常量 Const，可以先计算好结果Const 代替该 Op，而不用每次都在推理阶段都计算一遍
Fold Const To ExpandDims	ExpandDims 折叠	• ExpandDims Op 指定维度的输入是常量 Const，则把这个维度以参数的形式折叠到 ExpandDims 算子中
Fuse Const To Binary	Binary 折叠	• Binary Op 第二个输入是标量 Const ，把这个标量以参数的形式折叠到 Binary Op 的属性中

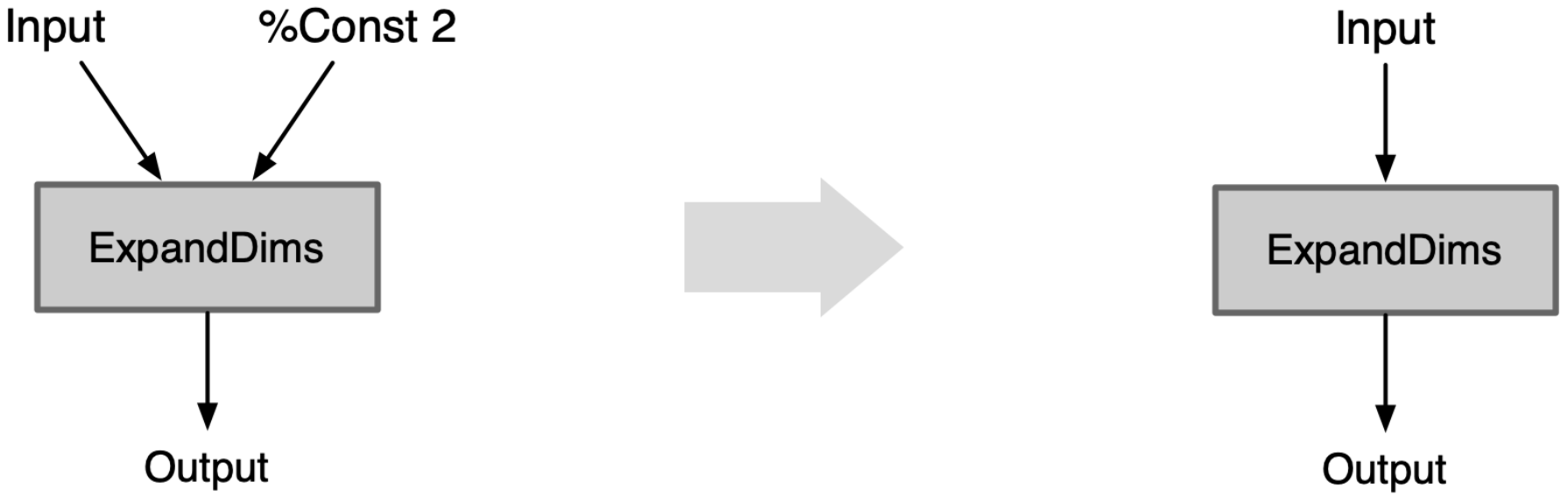
1 O1: Constant Folding 常量折叠

- **Constant Folding 常量折叠**：如果一个 Op 所有输入都是常量 Const，可以先计算好结果Const 代替该 Op，而不用每次都在推理阶段都计算一遍。



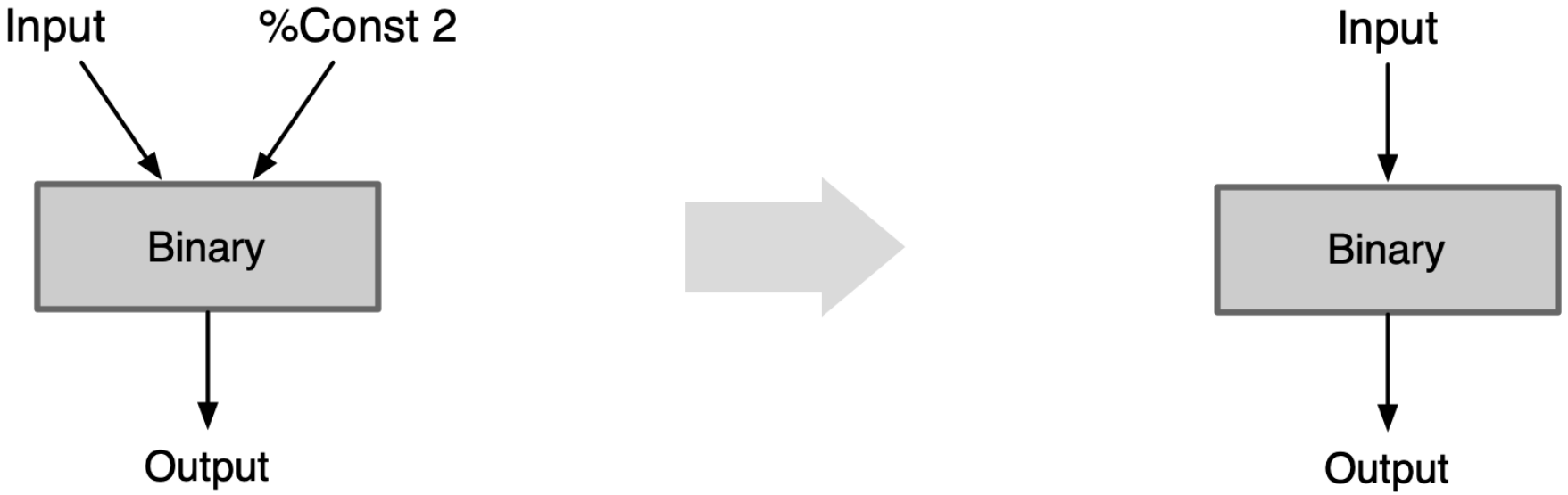
1 O1: Constant Folding 常量折叠

- **ExpandDims 折叠：** ExpandDims Op 指定维度的输入是常量 Const，则把这个维度以参数的形式折叠到 ExpandDims 算子中。



1 O1: Constant Folding 常量折叠

- **Binary 折叠：** Binary Op 第二个输入是标量 Const ，把这个标量以参数的形式折叠到 Binary Op 的属性中。



计算图优化

冗余节点消除

1 O2: Redundant node eliminations 冗余节点消除

- Remove all redundant nodes without changing the graph structure. 在不改变图形结构的情况下删除所有冗余节点。The following such optimizations are currently supported:
  1. Identity Elimination
  2. Slice Elimination
  3. Unsqueeze Elimination
  4. Dropout Elimination

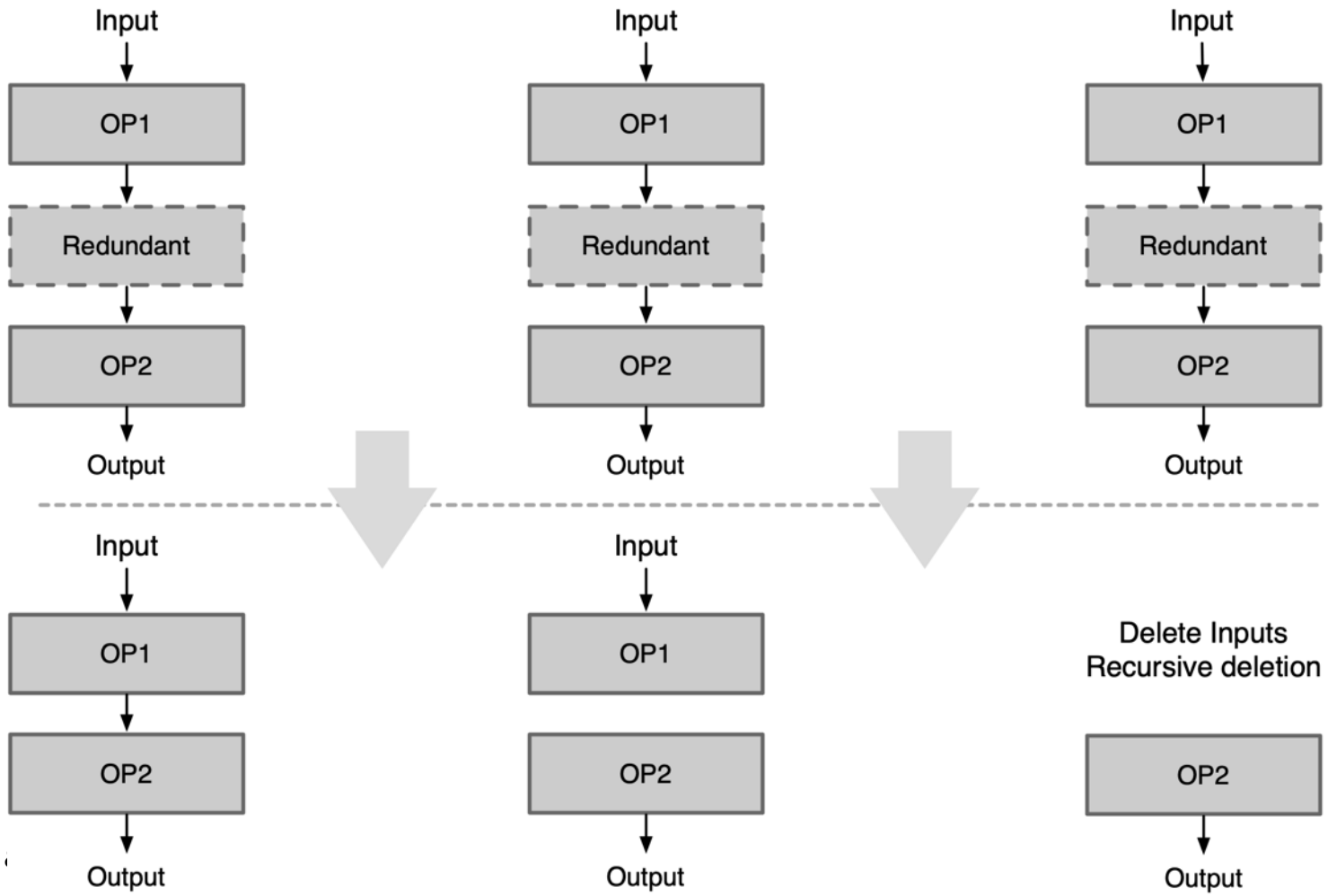
1 O2: Redundant node eliminations 冗余节点消除 I

- **Op本身无意义**：有些 Op 本身不参与计算，在推理阶段可以直接去掉对结果没有影响。

Remove Unuseful Op	冗余算子删除	• 去掉 Seq2Out, Identity, NoOp, Print, Assert, StopGradient, Split 等冗余算子
		• Cast 转换前后数据类型相等
		• Concat 只有一个输入 Tensor
Remove Dropout	Dropout 删除	• 删除因为抑制过拟合的方法引入的 Dropout 节点

1 O2: Redundant node eliminations 冗余节点消除 I

- **Op本身无意义**：有些 Op 本身不参与计算，在推理阶段可以直接去掉对结果没有影响。



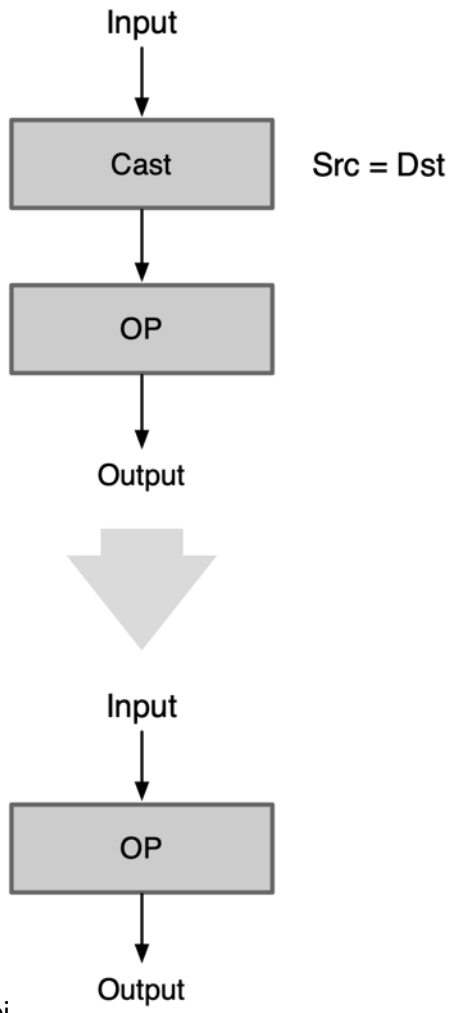
1 O2: Redundant node eliminations 冗余节点消除 II

- **Op 参数无意义**：有些 Op 本身是有意义，但是设置成某些参数后就变成了无意义了的 Op。

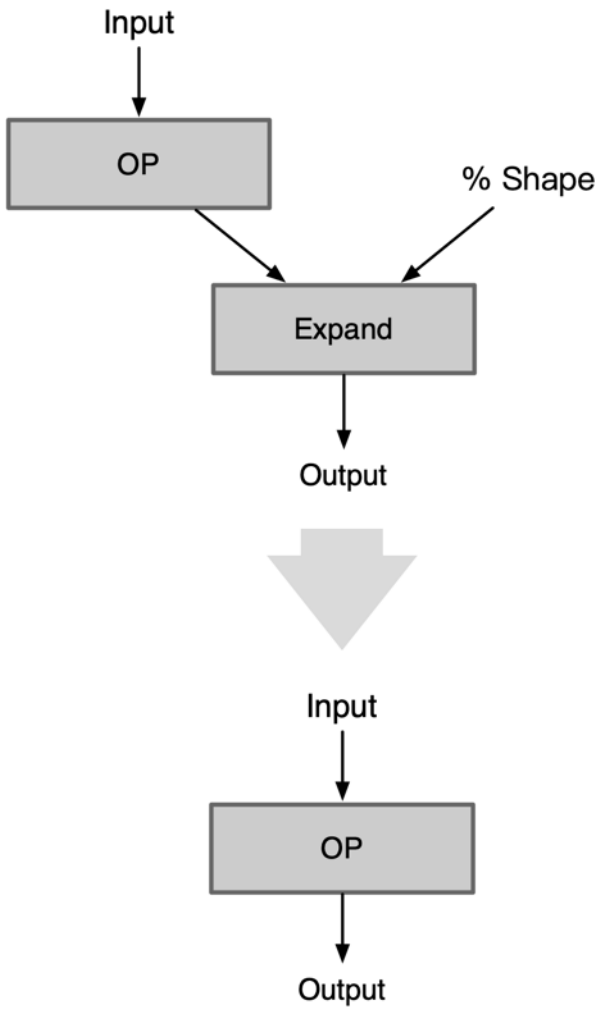
Tensor Cast	Cast 消除	• Tensor 转换数据排布格式时当参数 Src 等于 Dst 时，该 Op 无意义可删除
Slice Elimination	Slice 场景消除	• Slice Op 的 index_start 等于0或者 index_end 等于 c-1 时，该Op无意义可删除
Expand Elimination	Expand 消除	• Expand Op 输出 shape 等于输入 shape 时，该 Op 无意义可删除
pooling1x1 Elimination	pooling 消除	• Pooling Op 对滑窗 1x1 进行池化操作

1 O2: Redundant node eliminations 冗余节点消除 II

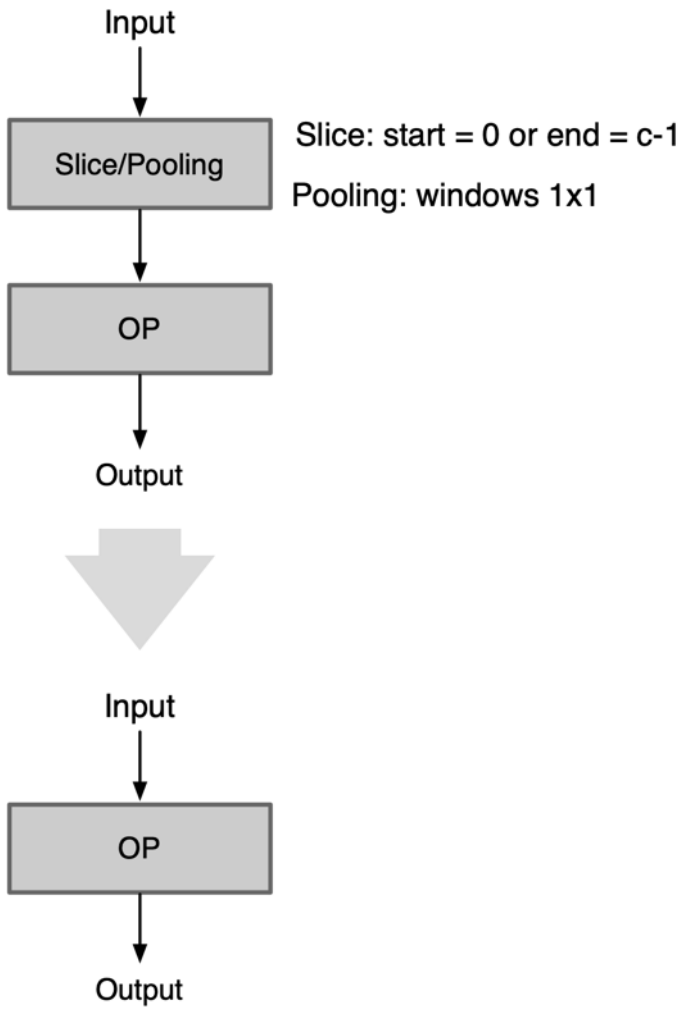
Tensor Cast



Expand Elimination



Slice/pooling1x1 Elimination



1 O2: Redundant node eliminations 冗余节点消除 III

- **Op 位置无意义**：一些 Op 在计算图中特殊位置会变得多余无意义；

Remove Output Cast	输出后消除	• 模型的输出不需要进行内存排布转换
Unsqueeze Elimination	Unsqueeze 消除	• 当 Unsqueeze Op 输入是 Const 时，可以将 Const 执行 Unsqueeze 操作后直接删除 Unsqueeze
Orphaned Eliminate	孤儿节点消除	• 网络模型中存在一些数据节点，当没有其他 Op 将该 Const Op 作为输入时可认为其为孤儿 orphaned 节点删除
Reshape before binary Eliminate	Reshape 消除	• 在 Binary Op 前面有 Reshape 算子，则将其删除
Reshape/Flatten after global pooling Eliminate	Reshape 消除	• Reshape 为 flatten 时（即期望 Reshape 后 Tensor w=1,h=1,c=c），而 global pooling 输出 Tensor w=1,h=1,c=c，则将其删除

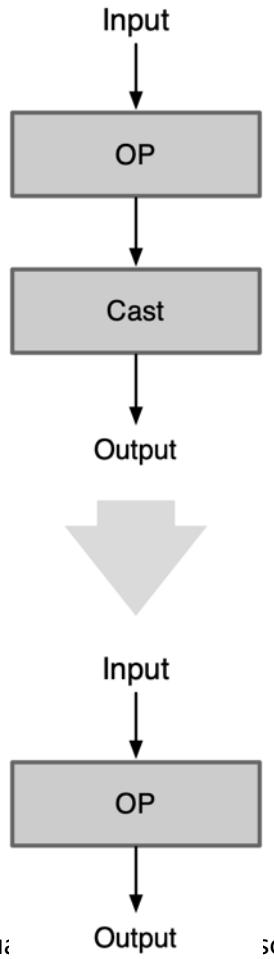
1 O2: Redundant node eliminations 冗余节点消除 III

- **Op 位置无意义**：一些 Op 在计算图中特殊位置会变得多余无意义；

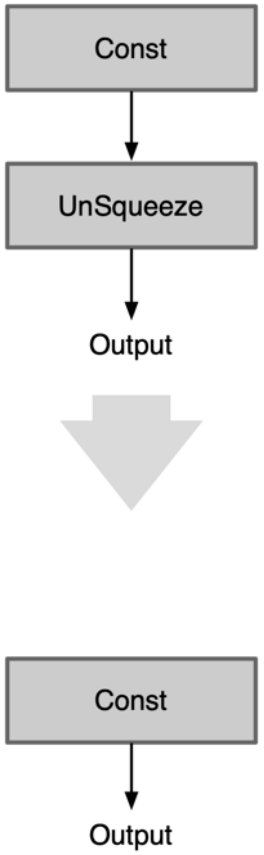
Flatten after linear Eliminate	Flatten 消除	• linear 全连接层输出 tensor 为 w=1,h=1,c=c时，后续 Flatten Op 可删除
Duplicate Reshape Eliminate	重复消除	• 连续 Reshape 只需要保留最后一个 Reshape
Duplicated Cast Eliminate	重复消除	• 连续的内存排布转换或者数据转换，只需要保留最后一个

1 O2: Redundant node eliminations 冗余节点消除 III

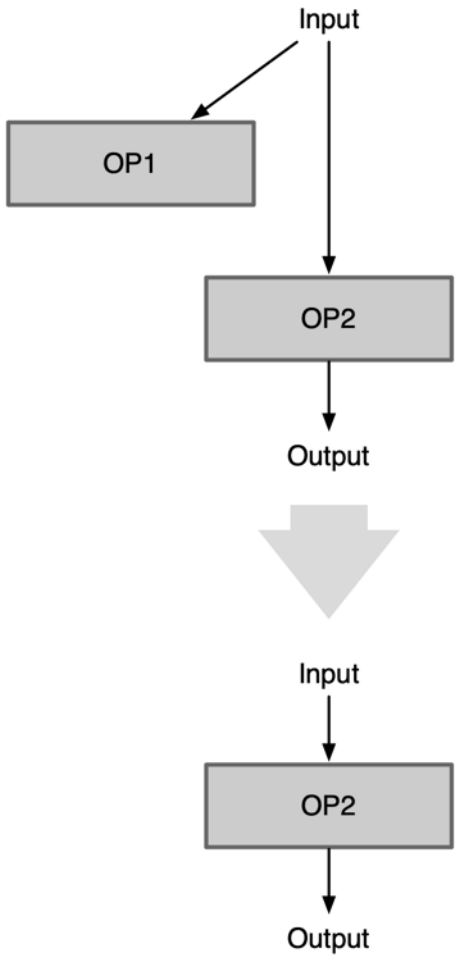
Remove Output Cast



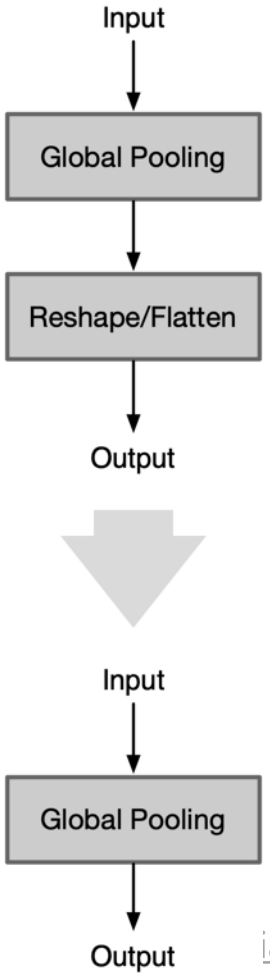
Unsqueeze Elimination



Orphaned Eliminate

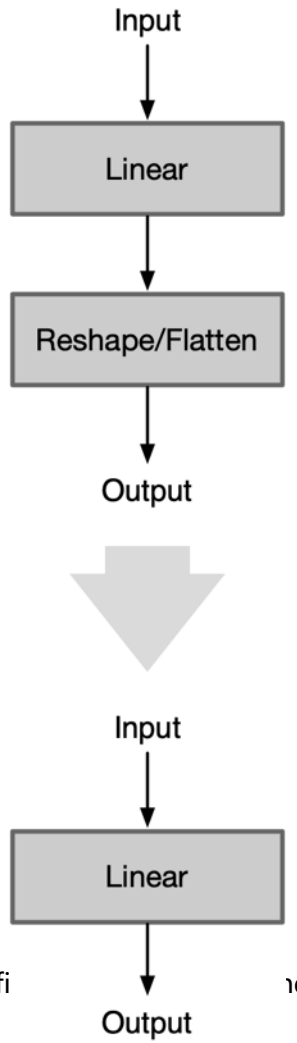


Reshape after global pooling

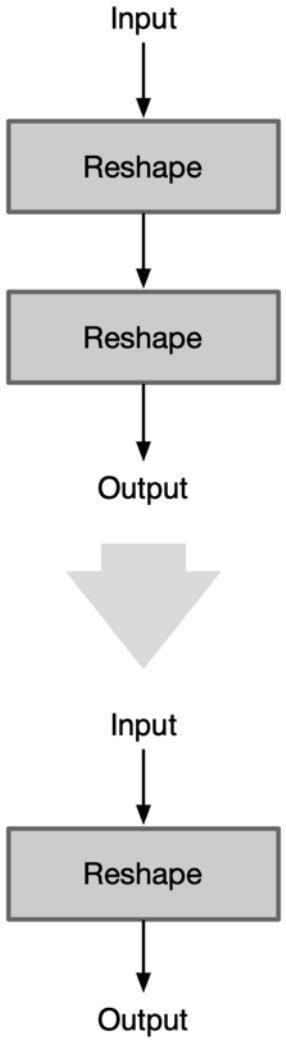


1 O2: Redundant node eliminations 冗余节点消除 III

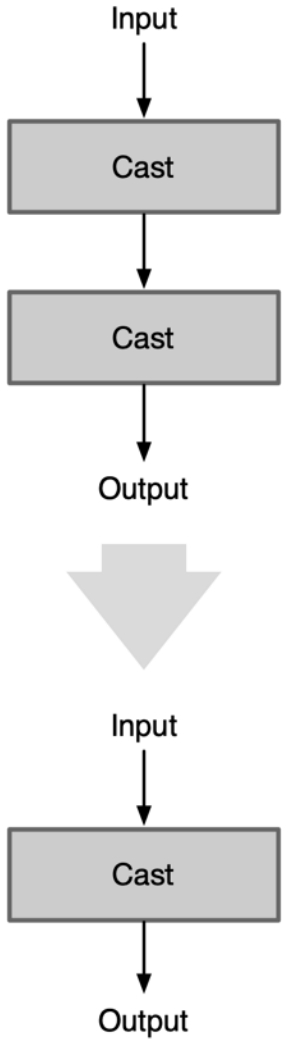
Flatten after linear Eliminate



Duplicate Reshape Eliminate



Duplicated Cast Eliminate



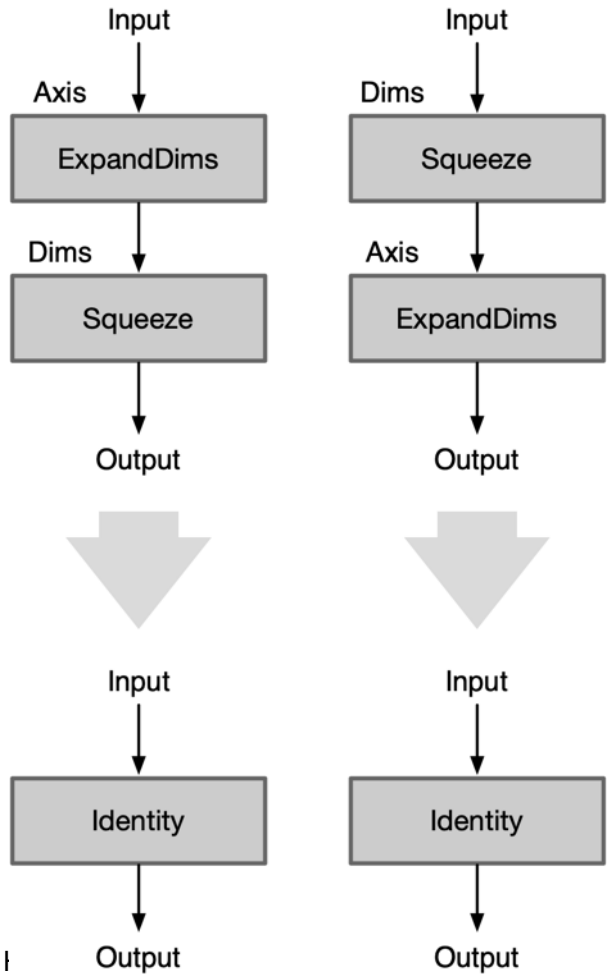
1 O2: Redundant node eliminations 冗余节点消除 IV

- **Op 前后反义**：前后两个相邻 Op 进行操作时，语义相反的两个 Op 都可以删除；

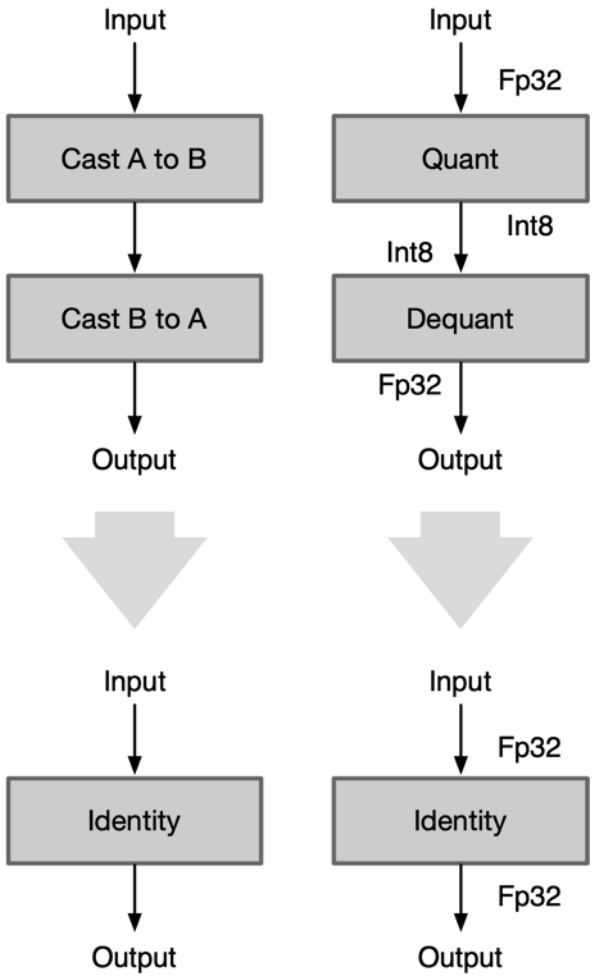
Squeeze ExpandDims Eliminate	• Squeeze和ExpandDims这两个Op是反义的,一个压缩维度，一个是拓展维度，当连续的这两个Op指定的axis相等时即可同时删除这两个Op
Inverse Cast Eliminate	• 当连续的两个内存排布转换Op的参数前后反义，即src1等于dst2,可同时删除这两个 Op
Quant Dequant Eliminate	• 连续进行量化和反量化，可同时删除这两个 Op
Concat Slice Elimination	• 合并后又进行同样的拆分，可同时删除这两个 Op

1 O2: Redundant node eliminations 冗余节点消除 IV

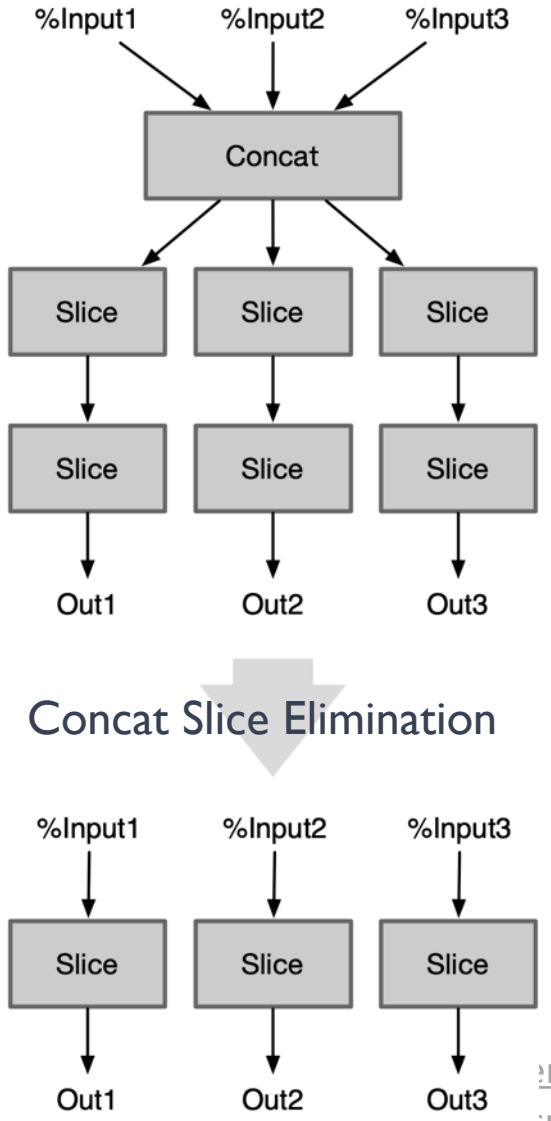
Squeeze ExpandDims Eliminate



Inverse Cast/ Quant Dequant Eliminate



Concat Slice Elimination



1 O2: Redundant node eliminations 冗余节点消除 V

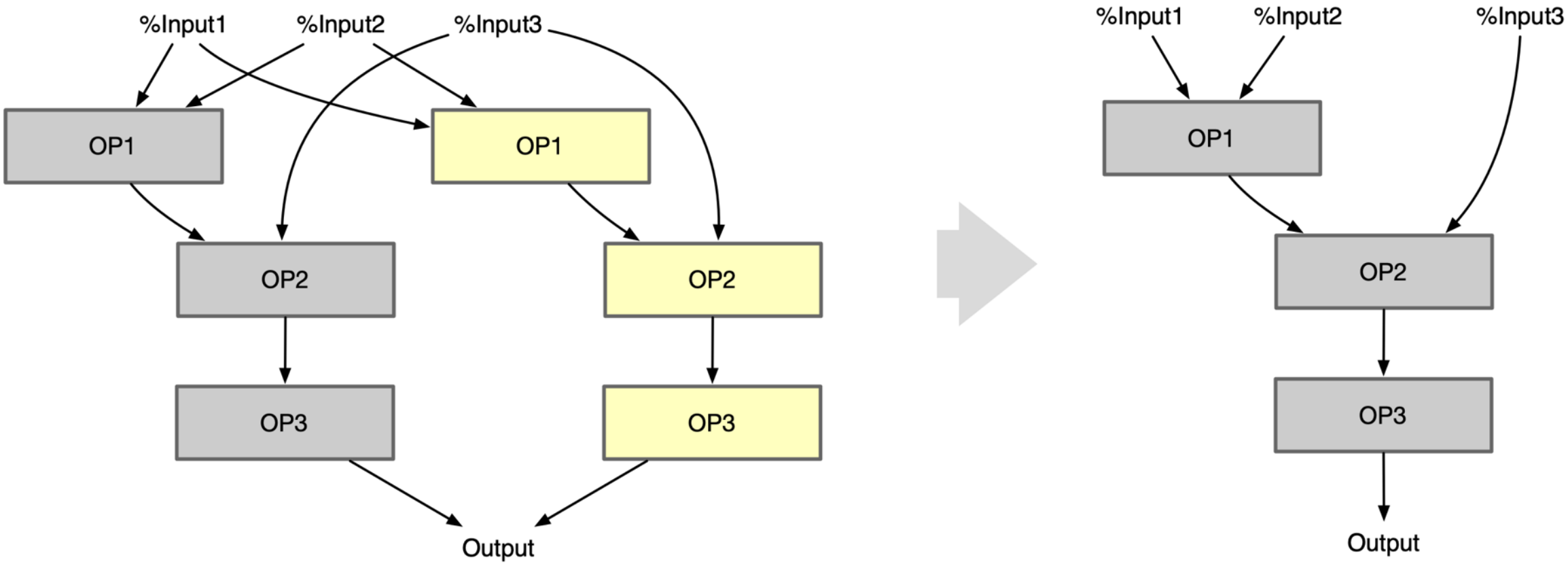
- 公共子图： 最大公共子图问题是给定两个图，要求去掉一些点后，两个图都得到一个节点数至少为b的子图，且两个子图完全相同。

Common Subexpression Elimination

- 当模型当中出现了公共子图，如一个输出是另外两个同类型同参数的Op的输入，则可进行删除其中一个Op。（同时这是一个经典的Leetcode算法题目，寻找公共子树，有兴趣可自行搜索）

---

1 O2: Redundant node eliminations 冗余节点消除 V



计算图优化

算子融合

1 O3: Operation fusions 算子融合

- Fuse/fold multiple nodes into a single node. For example, Conv Add fusion folds the Add operator as the bias of the Conv operator. The following such optimizations are currently supported:
  1. Conv Add Fusion
  2. Conv Mul Fusion
  3. Conv BatchNorm Fusion
  4. Relu Clip Fusion
  5. Reshape Fusion

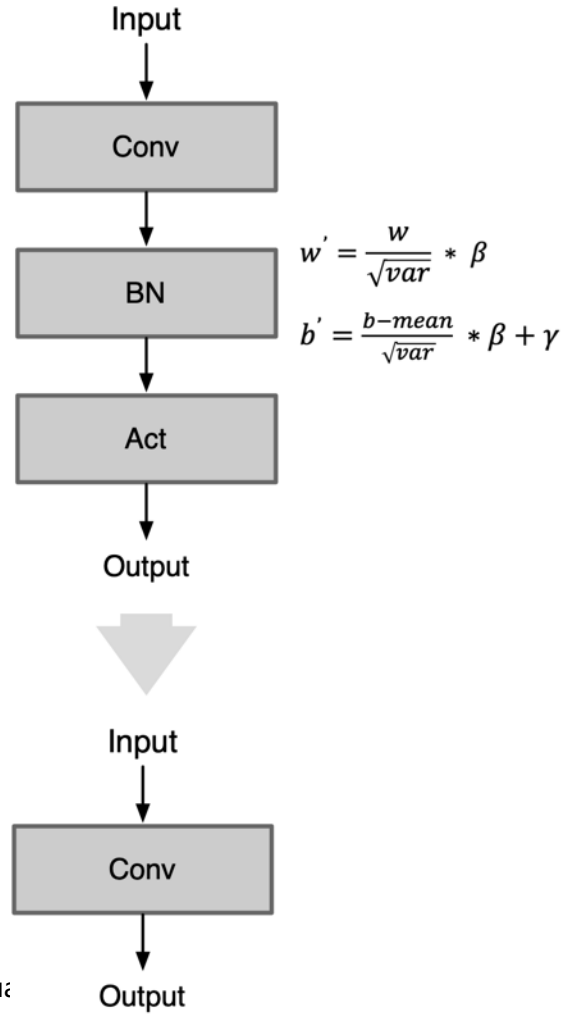
1 O3: Operation fusions 算子融合

- **Op线性融合**：相邻 Op 存在数学上线性可融合的关系；

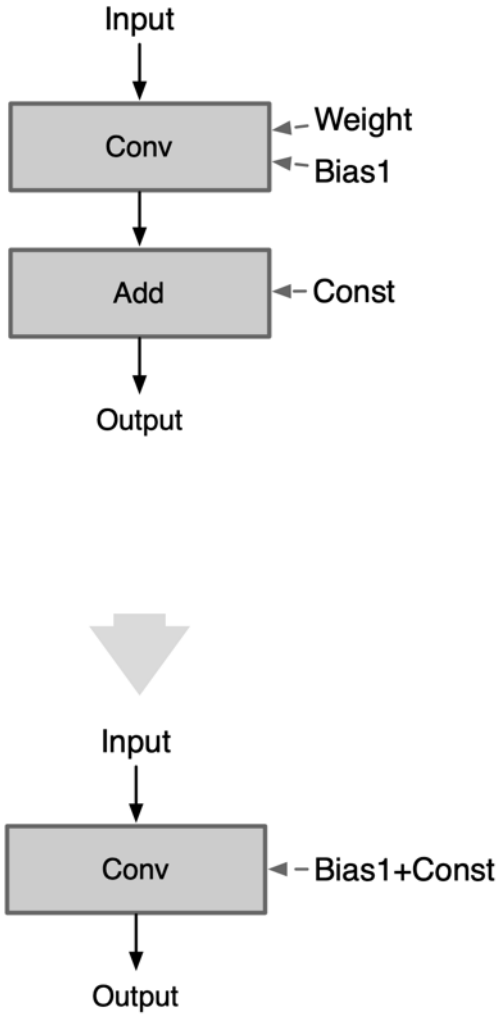
Conv + BN + Act	• Conv Op 后跟着的 Batch Normal 的算子可以把 BN 的参数融合到Conv里面
Conv + Bias + Add	• Conv Op 后跟着的 Add 可以融合到 Conv 里的 Bias 参数里面
Conv + Scale + Act	• Conv Op 后跟着的 Scale 可以融合到 Conv 里的 Weight 里面
Conv + MatMul + Act	• Conv Op 后跟着的 MatMul 可以融合到 Conv 里的 Weight 里面

1 O3: Operation fusions 算子融合 I

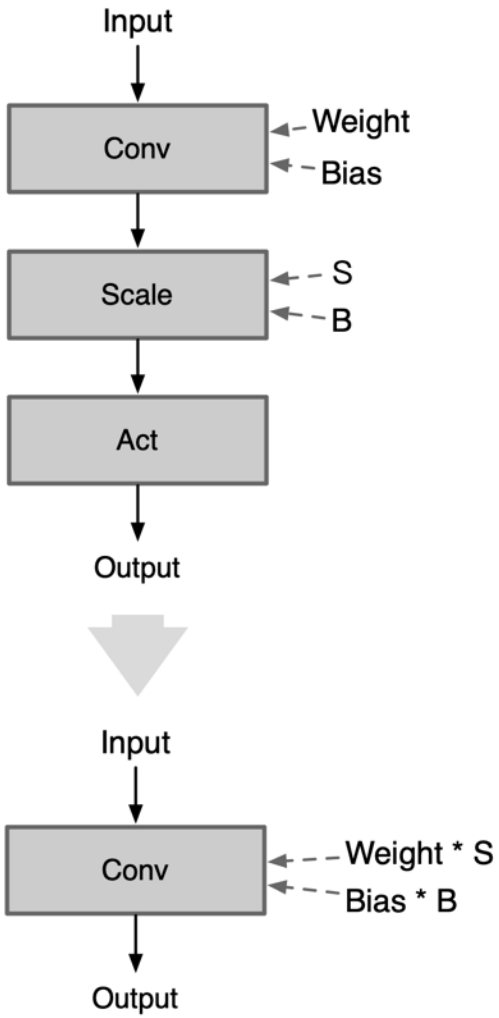
Conv + BN + Act



Conv + Bias + Add



Conv + Scale + Act/ Conv + MatMul + Act



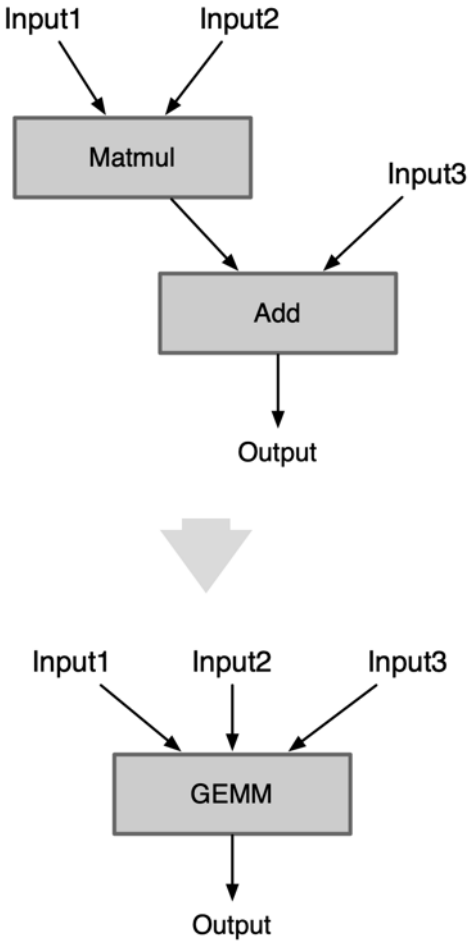
1 O3: Operation fusions 算子融合 II

- **Op线性融合**：相邻 Op 存在数学上线性可融合的关系；

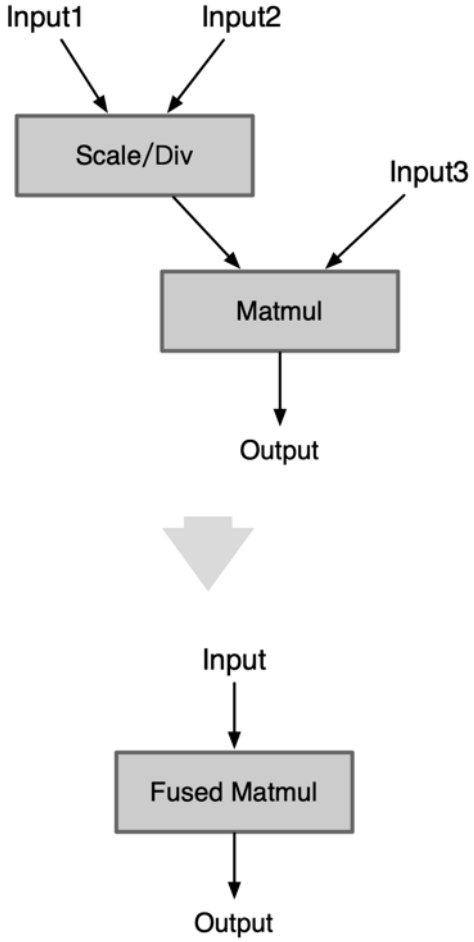
Matmul + Add	• 使用 GEMM 代替矩阵乘 Matmul + Add
Matmul + Scale	• Matmul 前或者后接 Scale / Div 可以融合到 Matmul 的相乘系数 alpha 里
Mean + Add	• 使用 Mean 后面跟着 Add，使用 Layer Norm 代替
Batch Norm + Scale	• scale 的 s 和 b 可以直接融合到 BN Op 里
Matmul + Batch Norm	• 与 Conv + BN 相类似
Matmul + Add	• 全连接层后 Add 可以融合到全连接层的 bias 中

1 O3: Operation fusions 算子融合 II

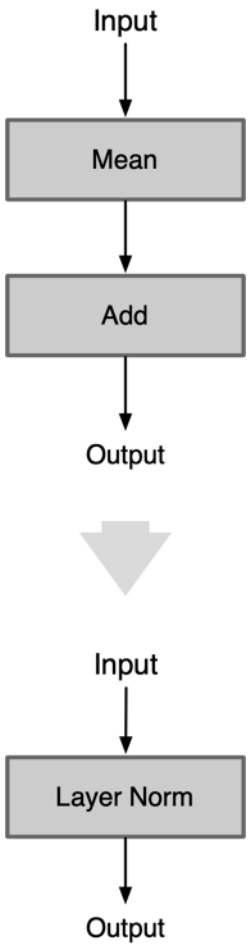
Matmul + Add



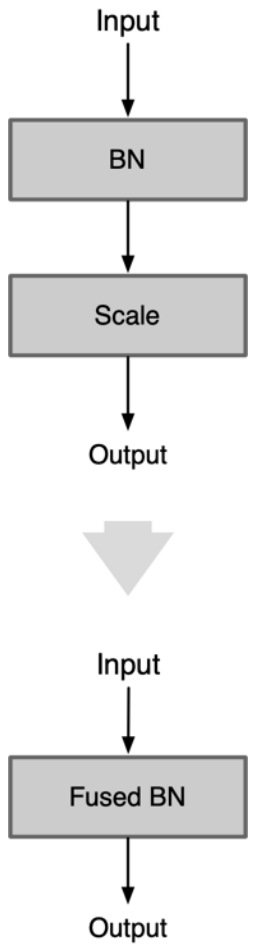
Matmul + Scale



Mean + Add



Batch Norm + Scale



1 O3: Operation fusions 算子融合 III

- **Op激活融合**：算子与后续的激活相融合；

Conv + ReLU

Conv + ReLU6

Conv + Act

- Act 激活操作和 Conv 操作虽然是连续但是计算过程是独立的，在推理的时候是先计算 Conv 层：访问 Conv 输出位置，再计算 ReLU 层（即第二次访存）。因此造成了访问两遍输出output，增加了访存时间降低了推理效率。
- 如果计算出 Conv 结果后立马进行 Act 激活计算，把最终结果输出，则只需要访存一次。计算量不变，减少访存次数，也能提高推理速度。

计算图优化

算子替换

1 O3: Operation Replace 算子替换

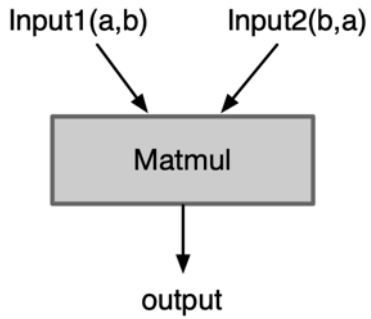
- Replace node with another node. 算子替换，即将模型中某些算子替换计算逻辑一致但对于在线部署更友好的算子。算子替换的原理是通过合并同类项、提取公因式等数学方法，将算子的计算公式加以简化，并将简化后的计算公式映射到某类算子上。算子替换可以达到降低计算量、降低模型大小的效果。

1 O3: Operation Replace 算子替换 I

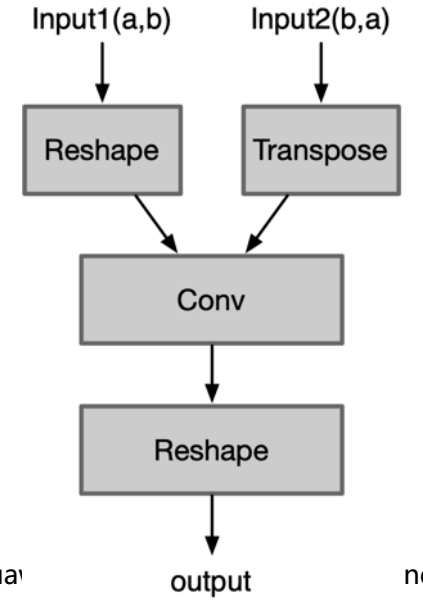
- **One to One**：将某 Op 以另外 Op 代替，能减少推理引擎需要单独实现及支持的 OP

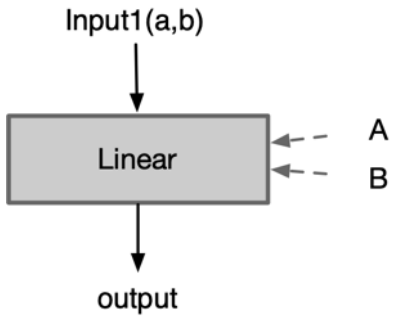
MatMul -> Conv2D	• 将矩阵乘变成Conv，因为一般框架对Conv是做了更多的优化
Linear -> Conv2D	• 将全连接层转变成1x1 Conv，因为对Conv做了更多的优化
Batch Normal -> Scale	• BN是等价于Scale Op的，转换成Scale计算量更少，速度更快
pReLU -> Leaky ReLU	• 将 pReLU 转变成 Leaky ReLU，不影响性能和精度的前提下，聚焦有限算法
Conv -> Linear After global pooling	• 在 Global Pooling 之后 Conv 算子转换成为全连接层

1 O3: Operation Replace 算子替换 I

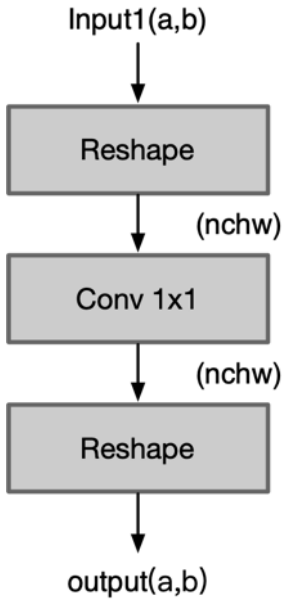


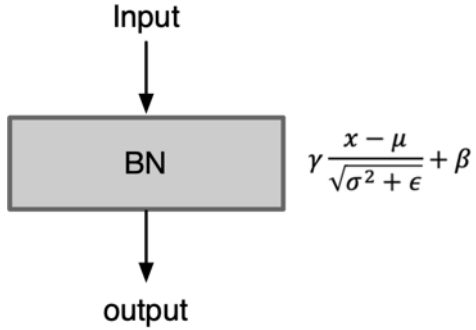
- MatMul -> Conv2D





- Linear -> Conv2D

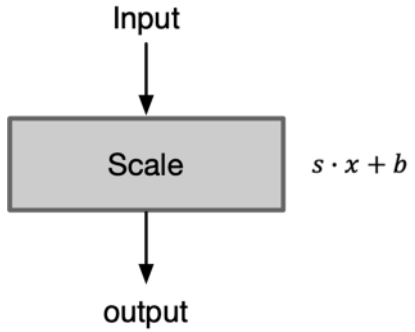


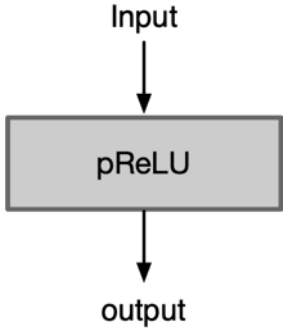


$$b = \beta - \frac{\gamma \cdot \mu}{\sqrt{\sigma^2 + \epsilon}}$$

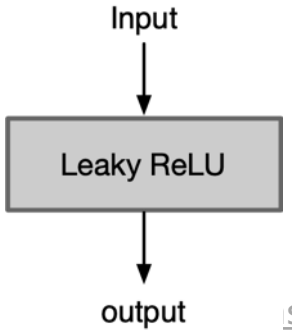
$$s = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}}$$

- Batch Normal -> Scale





- pReLU -> Leaky ReLU

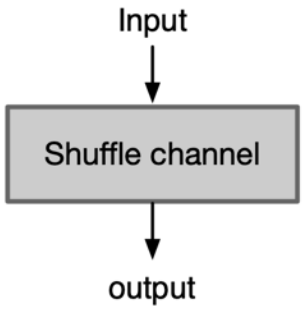


1 O3: Operation Replace 算子替换 II

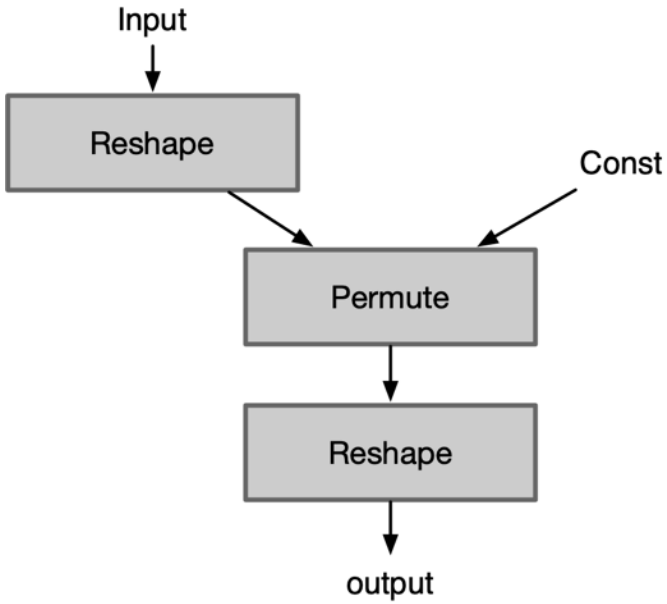
- **一换多**：将某 Op 以其他 Op 组合形式代替，能减少推理引擎需要单独实现及支持 Op 数量

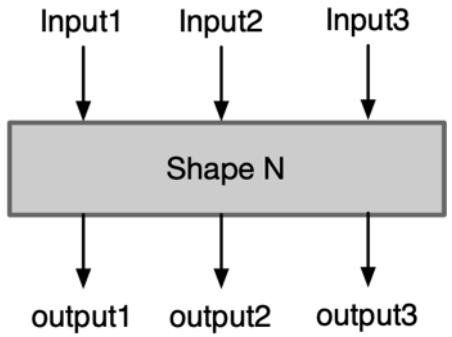
Shuffle Channel Replace	• Shuffle Channel Op 大部分框架缺乏单独实现，可以通过组合 Reshape + Permute实现
Pad Replace	• 将老版onnx的pad-2的pads从参数形式转成输入形式
ShapeN Replace	• 将 ShapeN Op 通过组合多个 Shape 的方式实现
Group Conv Replace	• 把Group 卷积通过组合 Slice、Conv 实现

1 O3: Operation Replace 算子替换 II

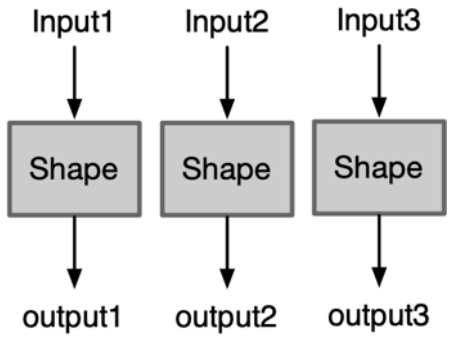


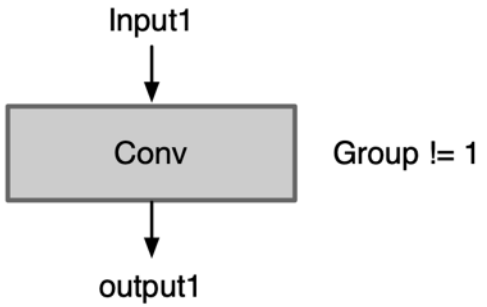
- Shuffle Channel Replace



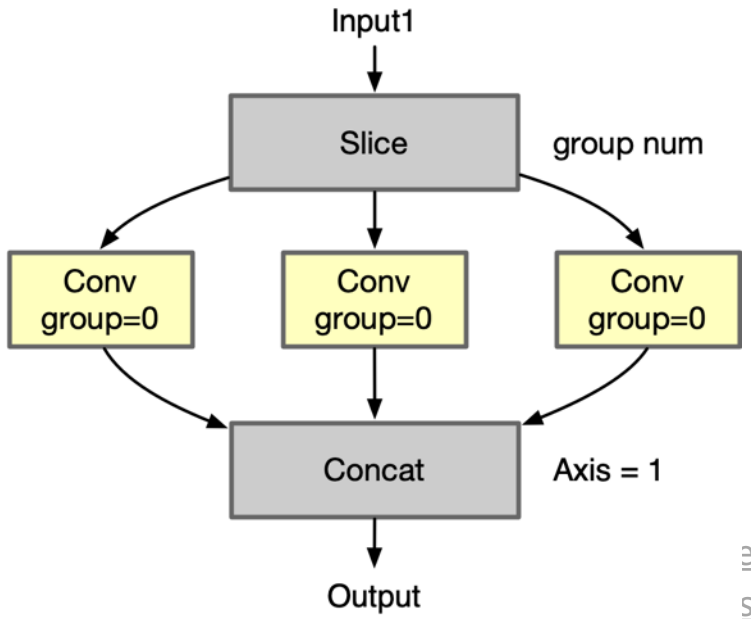


- ShapeN Replace





- Group Conv Replace



计算图优化

算子前移

1 O3: Operation Forward 算子前移

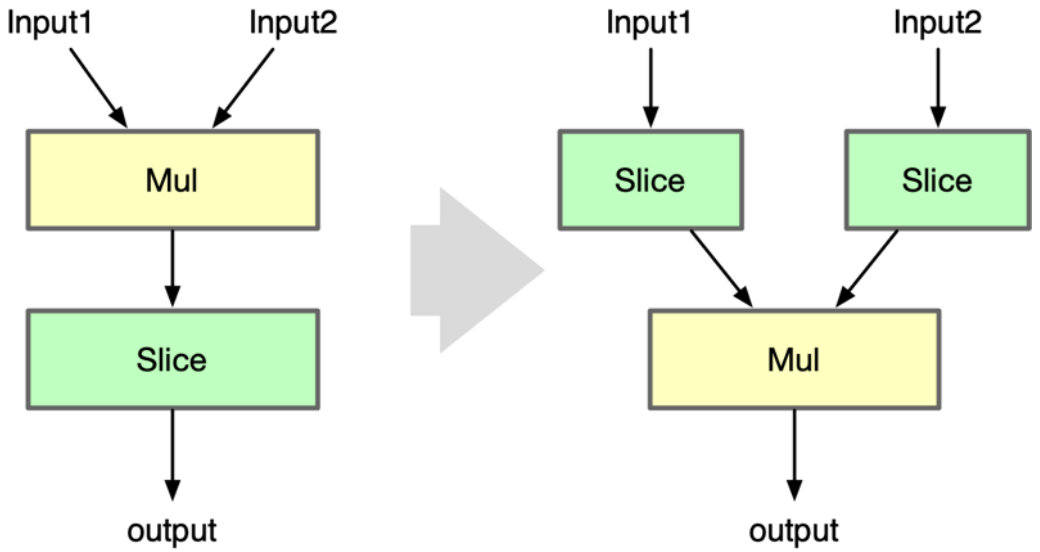
- Replace node with another node. 算子替换，即将模型中某些算子替换计算逻辑一致但对于在线部署更友好的算子。算子替换的原理是通过合并同类项、提取公因式等数学方法，将算子的计算公式加以简化，并将简化后的计算公式映射到某类算子上。算子替换可以达到降低计算量、降低模型大小的效果。

Slice and Mul	• Shuffle Channel Op 大部分框架缺乏单独实现，可以通过组合 Reshape + Permute实现
Bit shift and Reduce Sum	• 利用算术简化中的交换律，对计算的算子进行交换减少数据的传输和访存次数

1 O3: Operation Forward 算子前移

- 算子前移

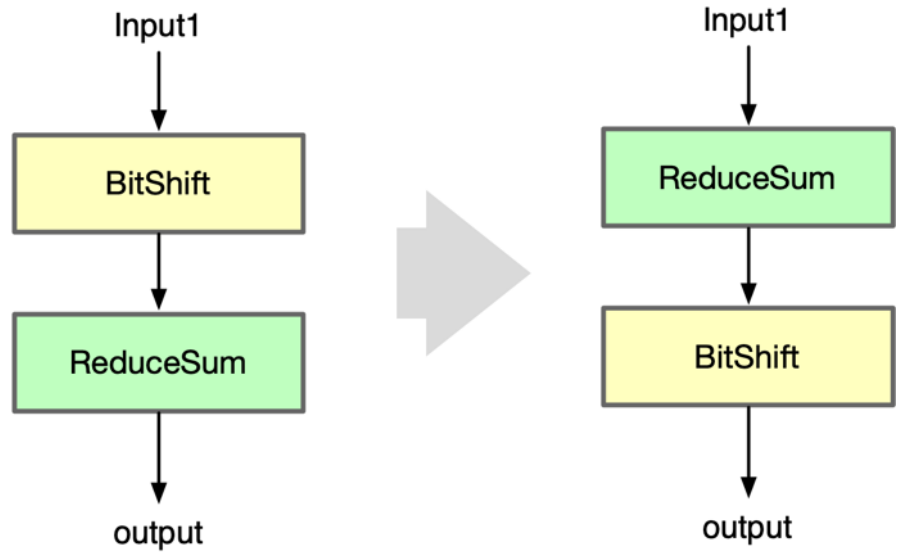
移动slice相关算动到计算图前，减少冗余计算



- 算术简化

$$ReduceSum(BitShift(A)) \rightarrow BitShift(ReduceSum(A))$$

$$ReduceProd(Exp(A)) \rightarrow Exp(ReduceSum(A))$$



1 O3: Operation Replace 算子替换 I

$$\gamma \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}+\beta$$

$$s=\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$$

$$b=\beta-\frac{\gamma \cdot \mu}{\sqrt{\sigma^2+\epsilon}}$$

$$s \cdot x+b$$