上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

CS433 Final Project Presentation

Parallel and Distributed Programming

饮 水 思 源 • 爱 国 荣 校

Members of Group17:
Ren Zhou, Sirong Zheng, Jiaxi Hu

Date:
December 31st, 2021

CONTENT

SHANGHAI JIAO TONG UNIVERSITY

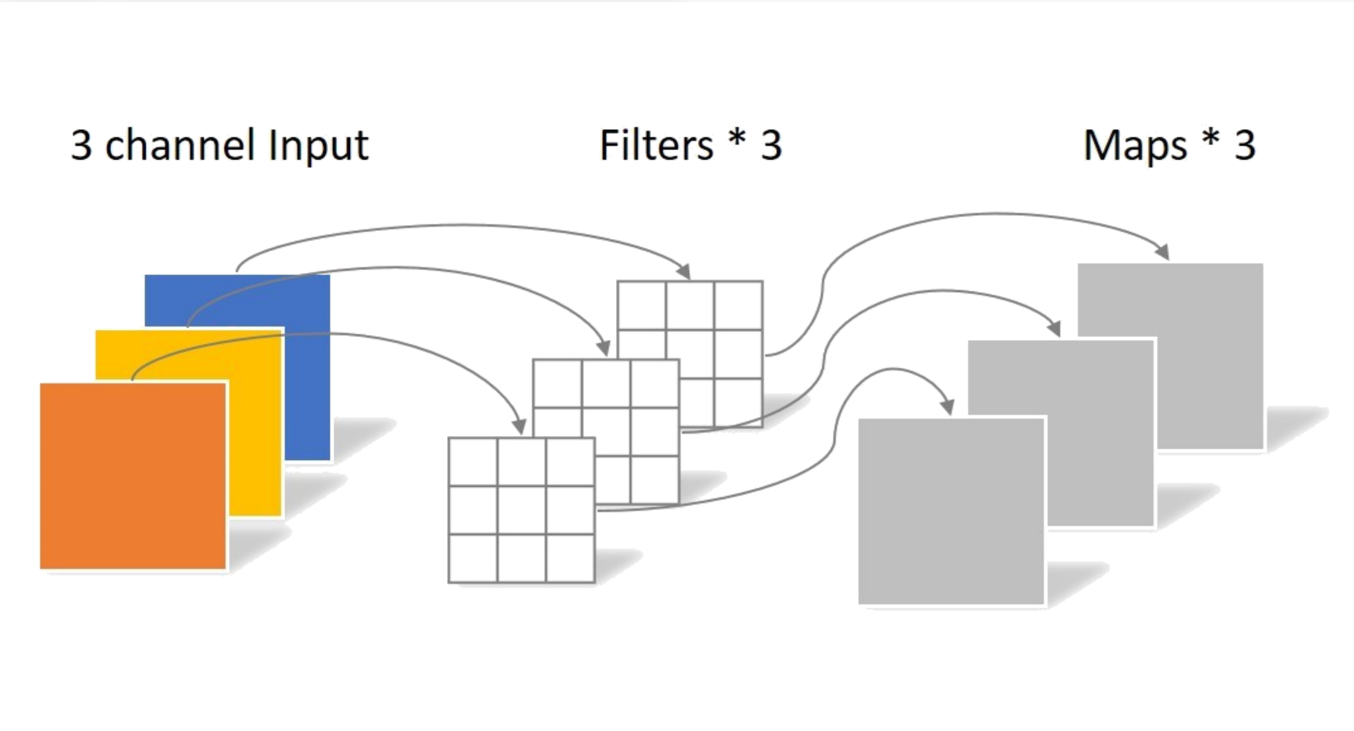
01

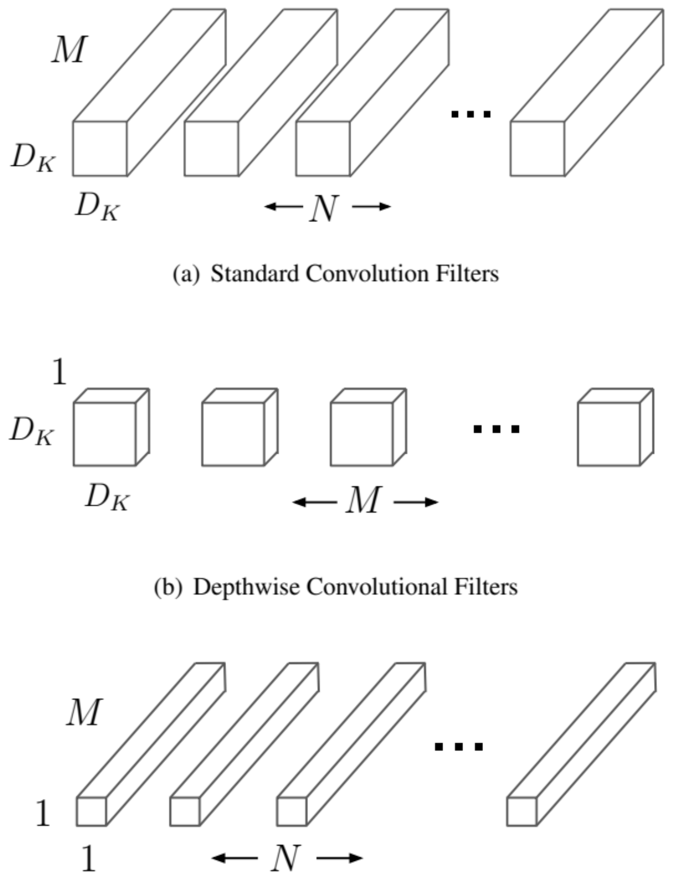
Introduction

- ☐ Prerequisite Knowledge
- ☐ Convolution Analysis

Introduction

Depth-wise Separable Convolution



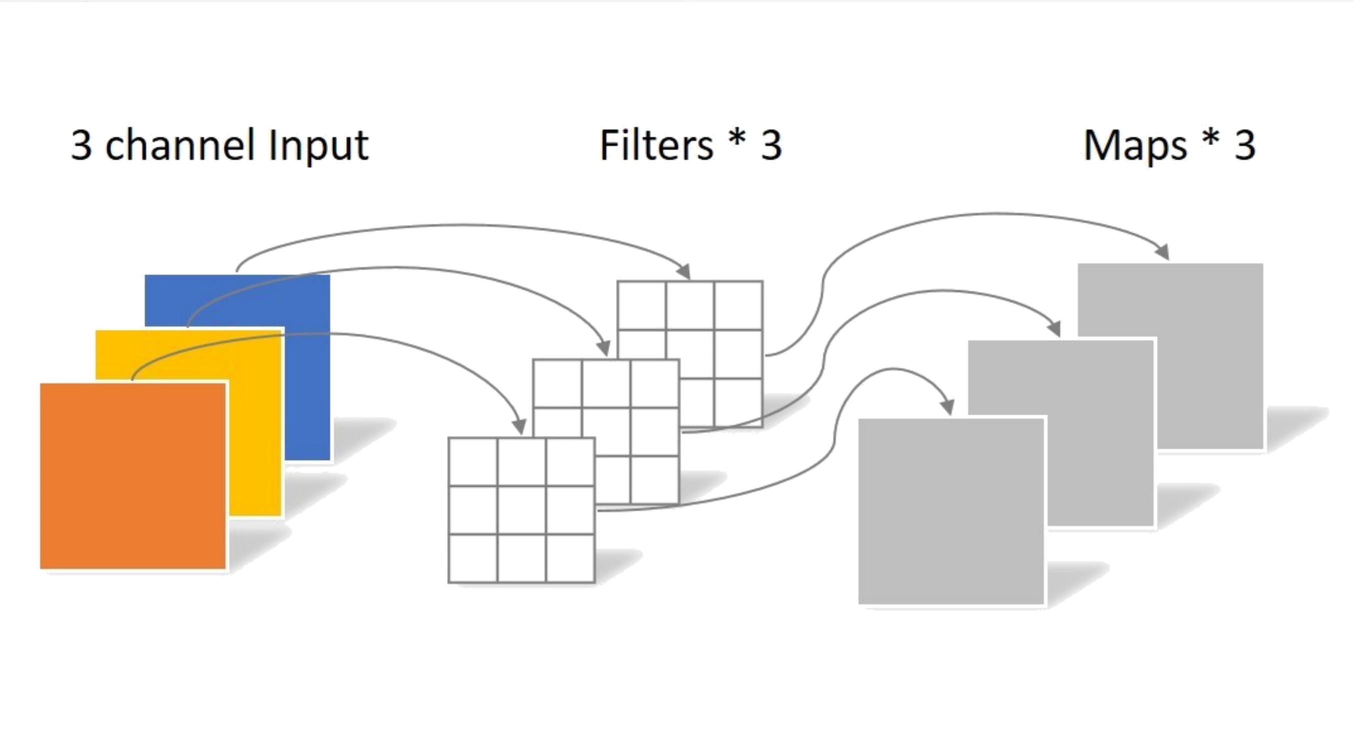


(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Depth-wise Separable Convolution



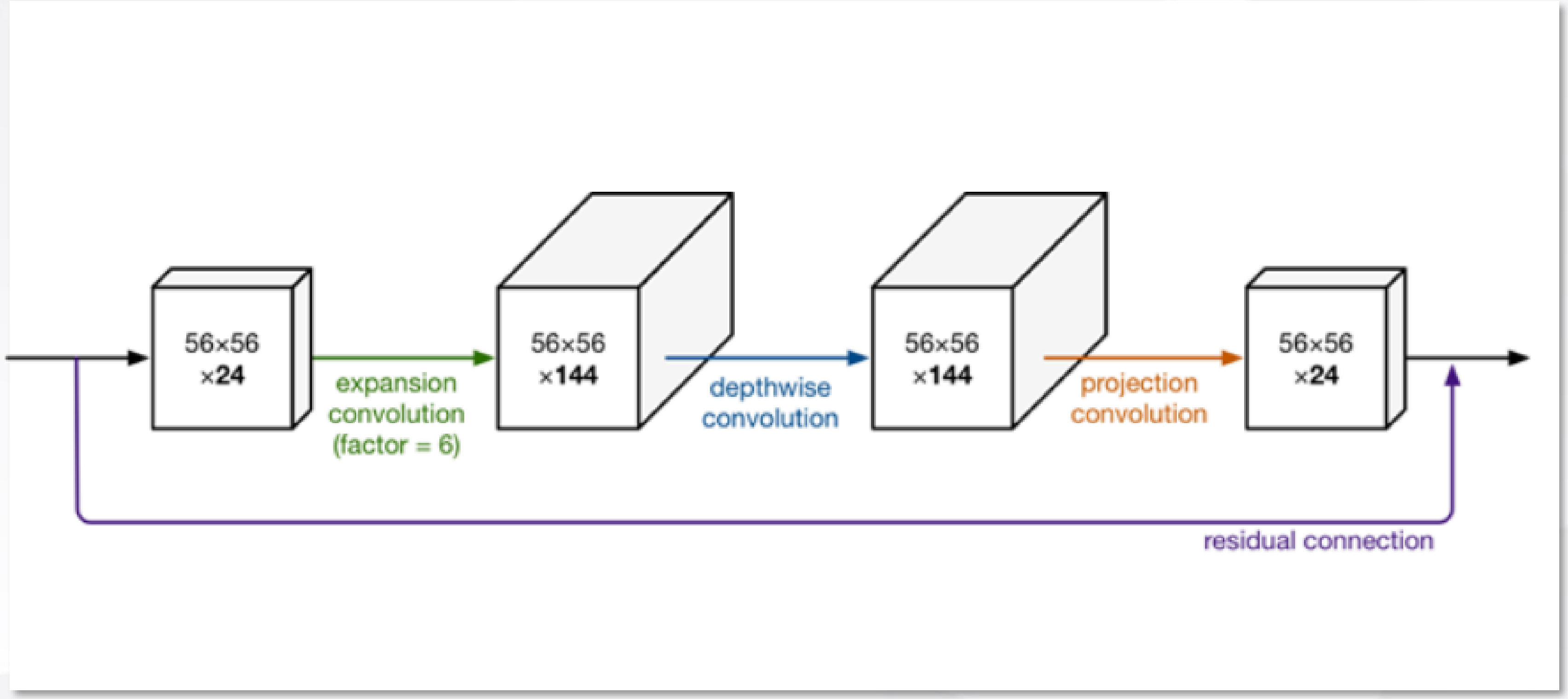
Standard Convolution:

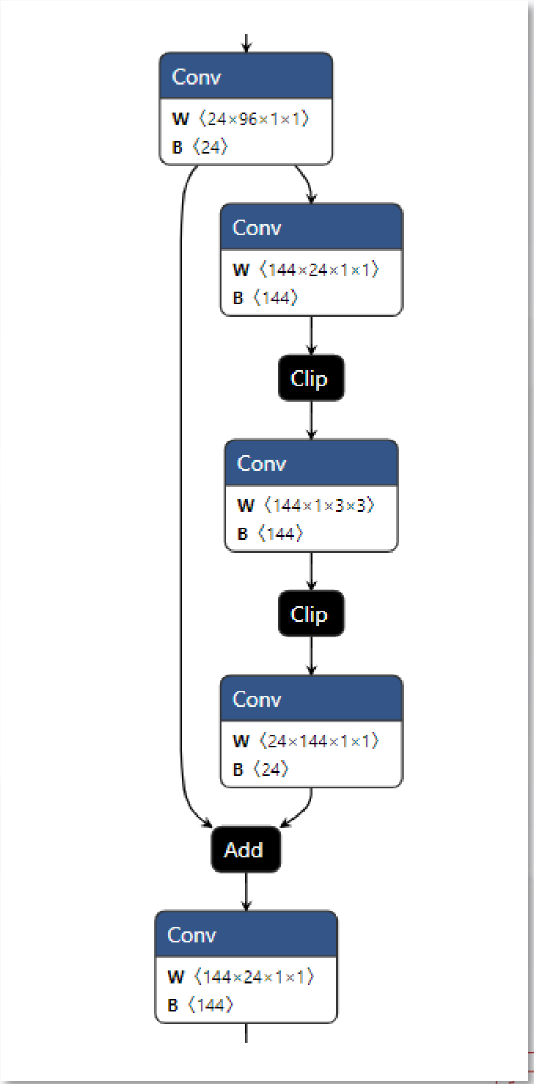
Input Map: C * H * W
Filter Kernel: C * k * k
Output Map: C * H’ * W’
params = k^2 * C
flops = k^2 * C * H’ * W’

Group Convolution:

Input Map: (C/g) * H * W * g
Filter Kernel: (C/g) * k * k * g
Output Map: C * H’ * W’ * g
params = k^2 * C
flops = k^2 * C * H’ * W’

Inverted Residual Block



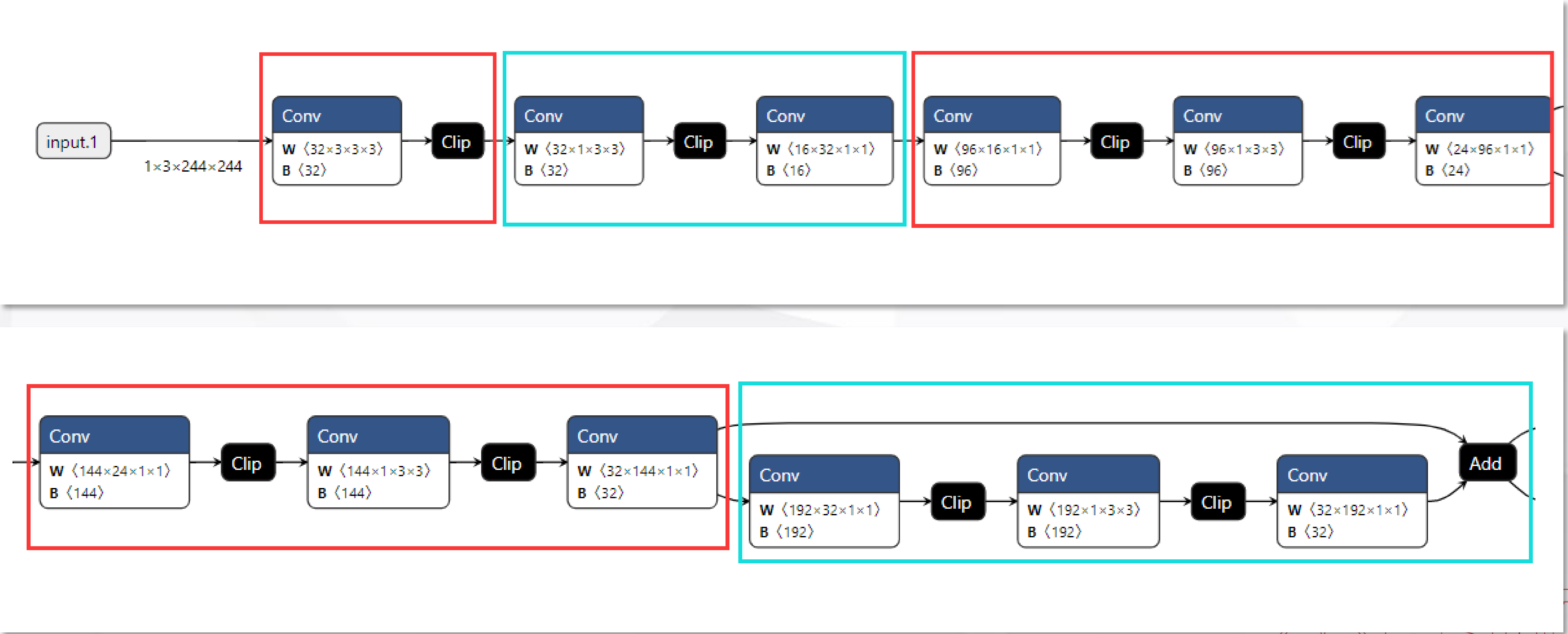


02

Basic Structure

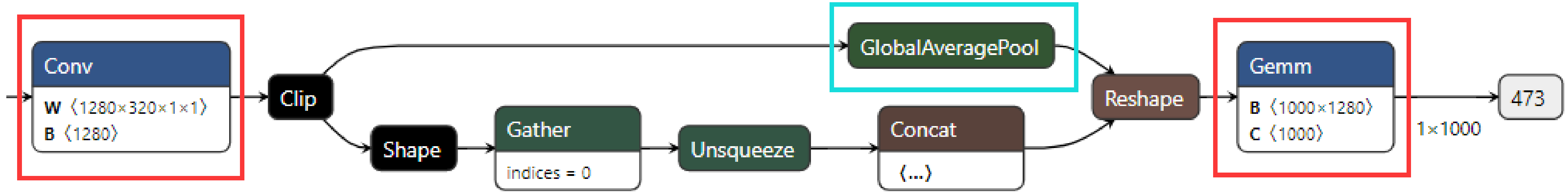
- ☐ Model Block Division
- ☐ Basic Layers

Model Block Division



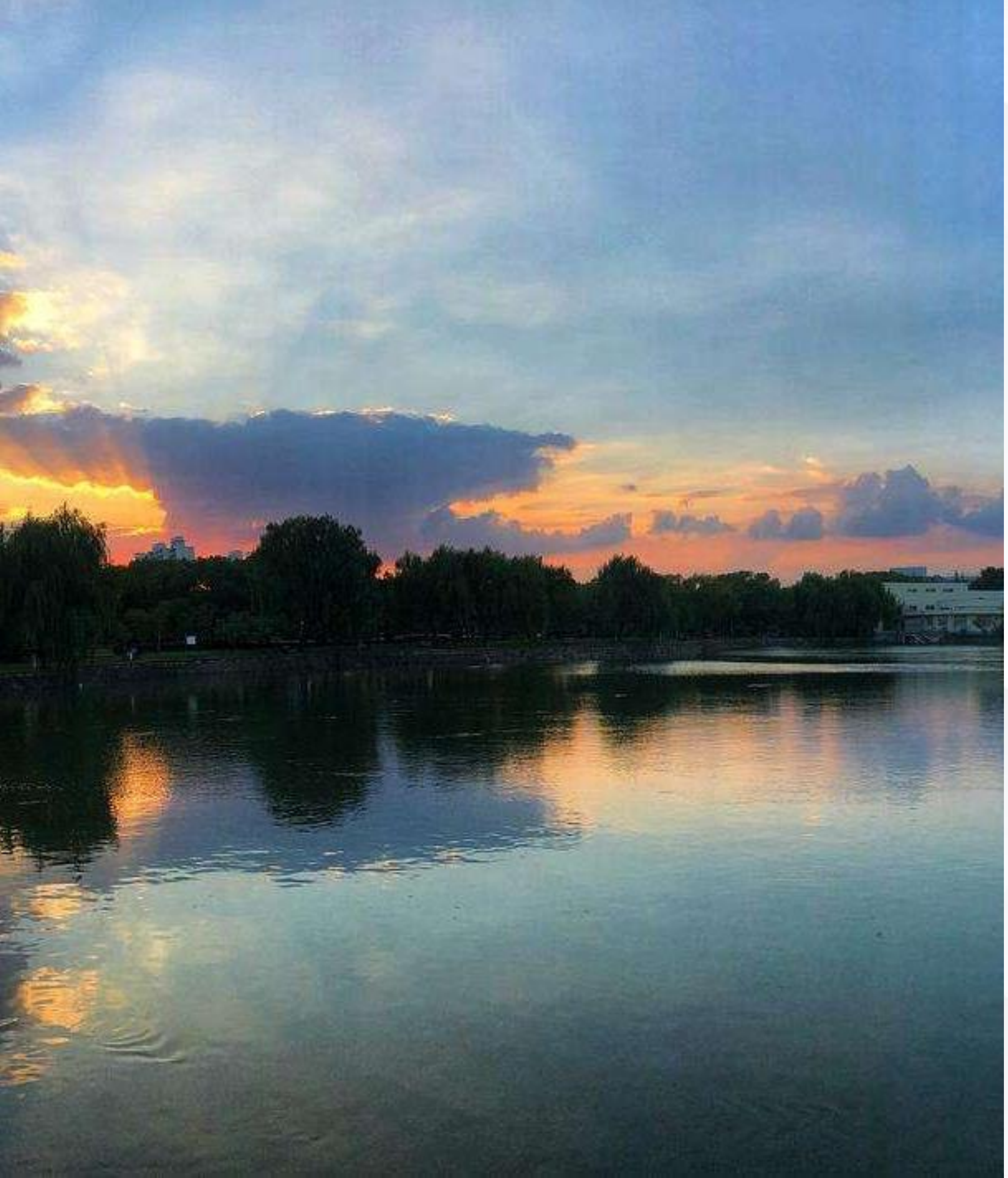
Basic Structure

Model Block Division



Basic Layers

- Standard Convolution 3D
- Depth-wise Convolution
- Point-wise Convolution
- Skip Connection Layer
- Global Average Pool
- Relu6
- Temporary Store Layer
- Full Connection Layer



03

Realization Details

- Parameters Processing
- Image to Column
- Other optimizations

Realization Details

Preparation

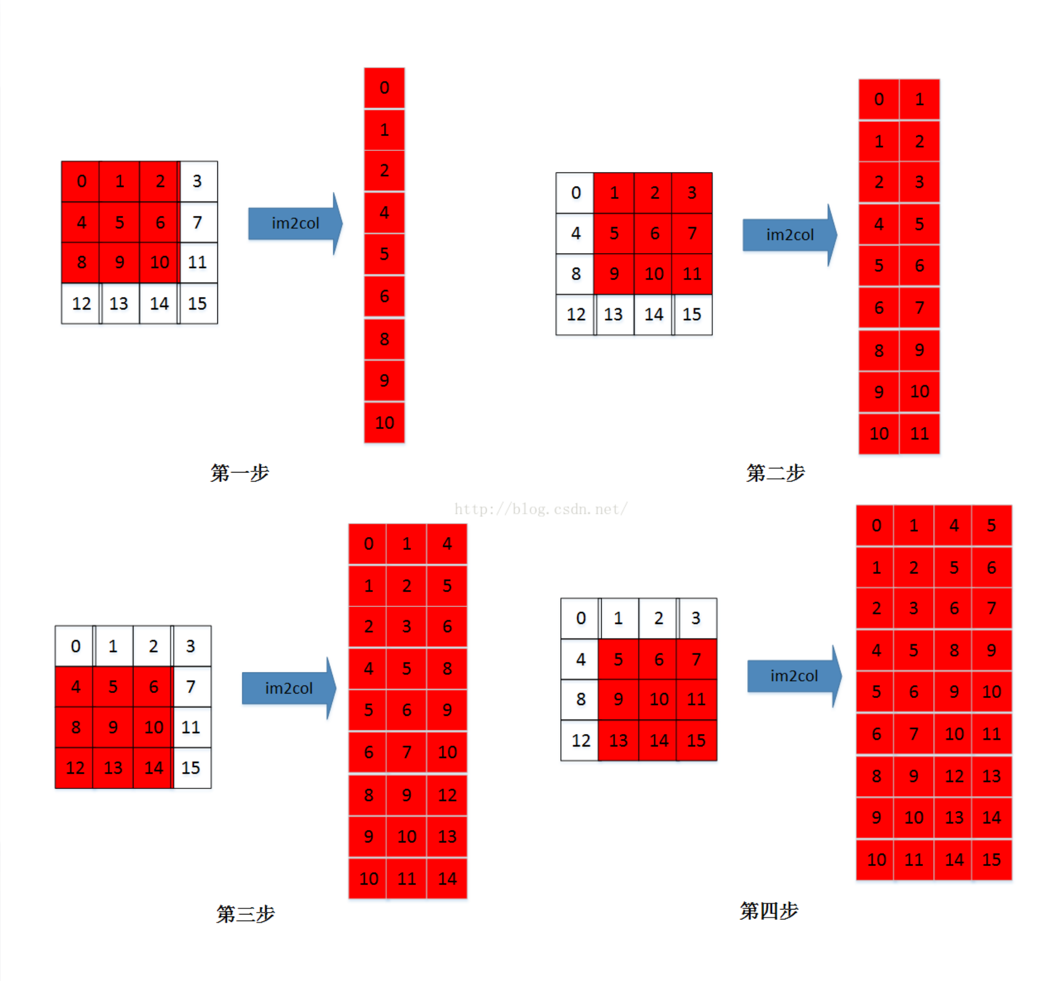
Onnx Model Params

```
31 568 (384, 1, 3, 3) 569 (384,)
32 571 (96, 384, 1, 1) 572 (96,)
33 574 (576, 96, 1, 1) 575 (576,)
34 577 (576, 1, 3, 3) 578 (576,)
35 580 (96, 576, 1, 1) 581 (96,)
36 583 (576, 96, 1, 1) 584 (576,)
37 586 (576, 1, 3, 3) 587 (576,)
38 589 (96, 576, 1, 1) 590 (96,)
39 592 (576, 96, 1, 1) 593 (576,)
40 595 (576, 1, 3, 3) 596 (576,)
41 598 (160, 576, 1, 1) 599 (160,)
42 601 (960, 160, 1, 1) 602 (960,)
43 604 (960, 1, 3, 3) 605 (960,)
44 607 (160, 960, 1, 1) 608 (160,)
45 610 (960, 160, 1, 1) 611 (960,)
46 613 (960, 1, 3, 3) 614 (960,)
47 616 (160, 960, 1, 1) 617 (160,)
48 619 (960, 160, 1, 1) 620 (960,)
49 622 (960, 1, 3, 3) 623 (960,)
50 625 (320, 960, 1, 1) 626 (320,)
51 628 (1280, 320, 1, 1) 629 (1280,)
52 classifier.1.weight (1000, 1280) classifier.1.bias (1000,)
```

Device Query

```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla V100-PCIE-32GB"
  CUDA Driver Version / Runtime Version          10.2 / 10.2
  CUDA Capability Major/Minor version number:    7.0
  Total amount of global memory:                 32510 MBytes (34089730048 bytes)
  (80) Multiprocessors, ( 64) CUDA Cores/MP:     5120 CUDA Cores
  GPU Max Clock rate:                            1380 MHz (1.38 GHz)
  Memory Clock rate:                             877 Mhz
  Memory Bus Width:                              4096-bit
  L2 Cache Size:                                 6291456 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 7 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 59 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.2, CUDA Runtime Version = 10.2, NumDevs = 1
Result = PASS
```

Img2Col: rearrange data



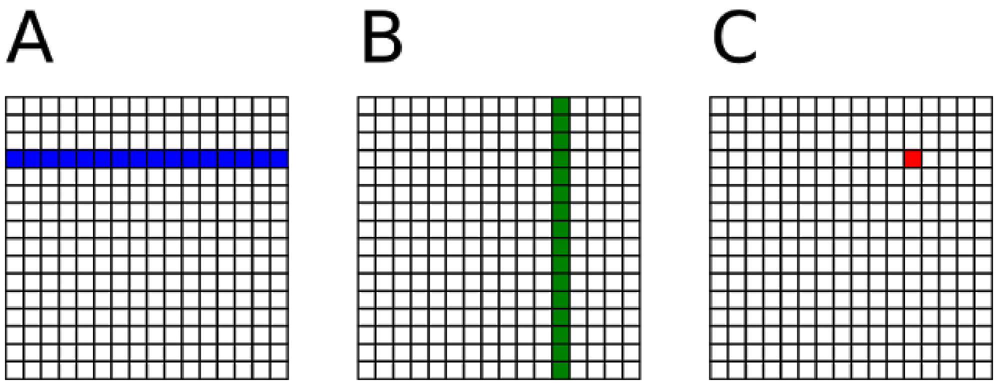
$$x^{(1)} = \begin{bmatrix} x_{1,1,1} & x_{1,1,2} & x_{1,1,3} \\ x_{1,2,1} & x_{1,2,2} & x_{1,2,3} \\ x_{1,3,1} & x_{1,3,2} & x_{1,3,3} \\ x_{2,1,1} & x_{2,1,2} & x_{2,1,3} \\ x_{2,2,1} & x_{2,2,2} & x_{2,1,3} \\ x_{2,3,1} & x_{2,3,2} & x_{2,3,3} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \\ x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} \\ x_{16} & x_{17} & x_{18} \end{bmatrix}$$

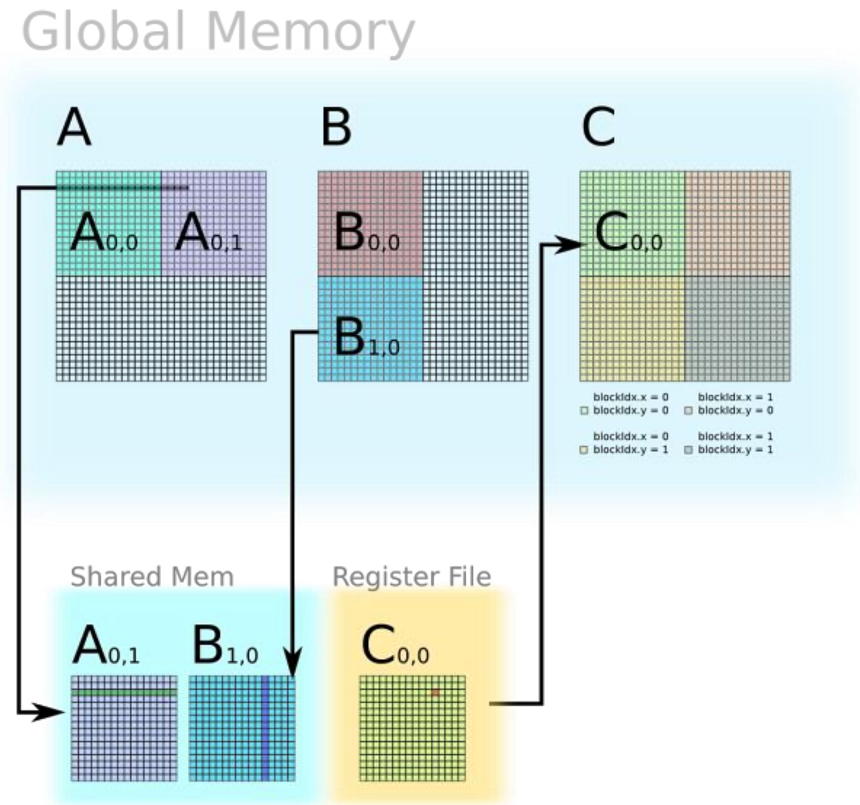
$$X = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \\ x^{(4)} \end{bmatrix} \qquad W_1 = \begin{bmatrix} w_1 \\ \vdots \\ w_{18} \end{bmatrix} \qquad W = [W_1 \quad W_2]$$

$$A = XW = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \\ x^{(4)} \end{bmatrix} [W_1 \quad W_2] = \begin{bmatrix} a_1 & a_5 \\ a_2 & a_6 \\ a_3 & a_7 \\ a_4 & a_8 \end{bmatrix}$$

Matrix Multiplication

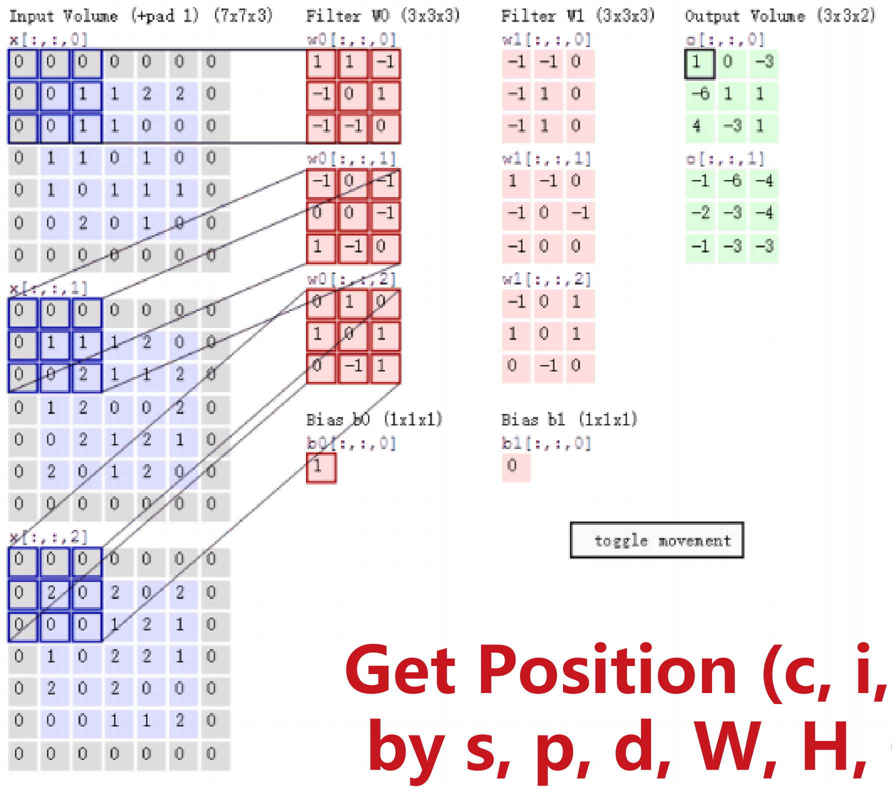
- ☐ Naïve Multiplication
- ☐ Tiling in shared memory
- ☐ Increasing work per thread





Realization Details

Depth-wise Convolution



Get Position (c, i, j) by s, p, d, W, H, C

为什么depthwise convolution 比convolution更加耗时?

知乎 · 15 个回答 · 371 关注 ›

cs sun

＋ 关注

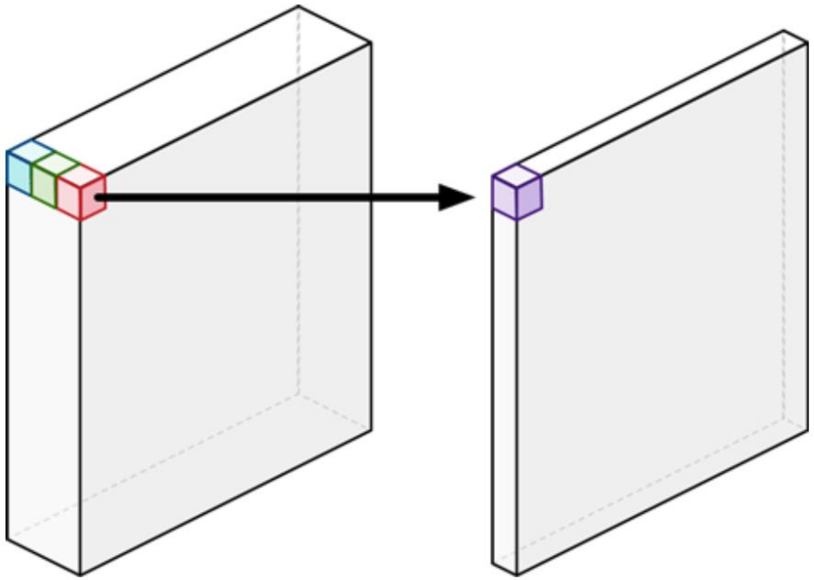
98 人赞同了该回答

首先，caffe原先的gpu实现group convolution很糟糕，用for循环每次算一个卷积，速度极慢。

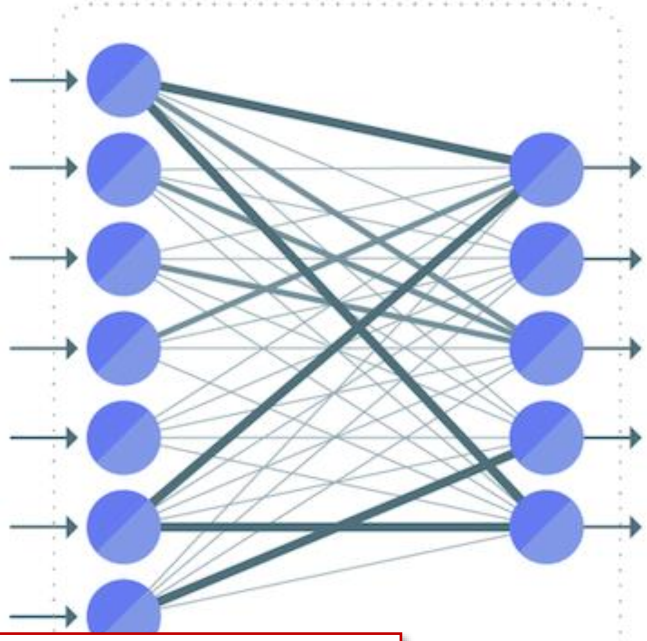
第二，cudnn7.0及之后直接支持group convolution，但本人实测，速度比github上几个直接写cuda kernel计算的dw convolution速度慢。例如对于n=128, c=512, h=32, w=32, group=512的卷积跑100次，cudnn 7.0里的group convolution需要4秒多，而yonghenglh6/DepthwiseConvolution大概只需要1秒。

Realization Details

Point-wise convolution



Full connection layer

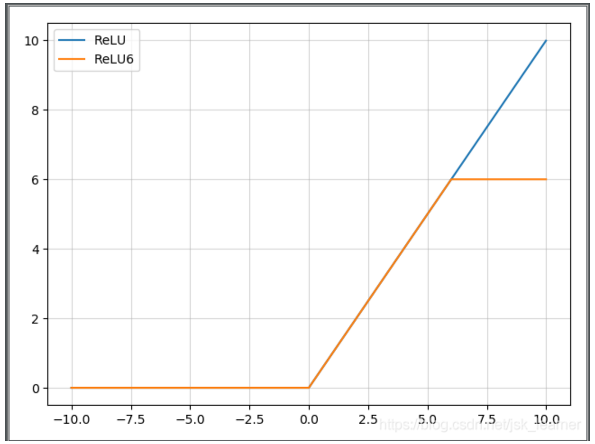


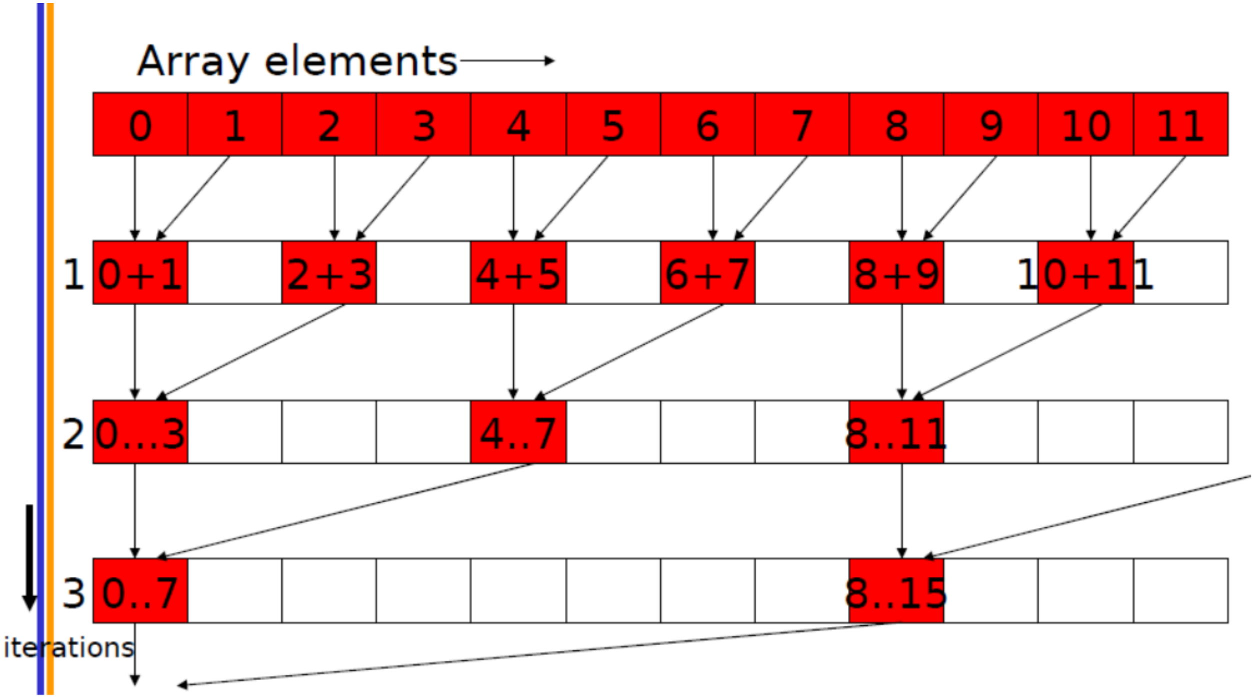
Convert to Matrix Multiplication!

Realization Details

Other Layers

- ☐ Add Layer
- ☐ Global Average Pool
- ☐ Relu6





Apply Const Memory when Add Bias!

04

Result Demonstration

- Benchmark
- Performance Comparison

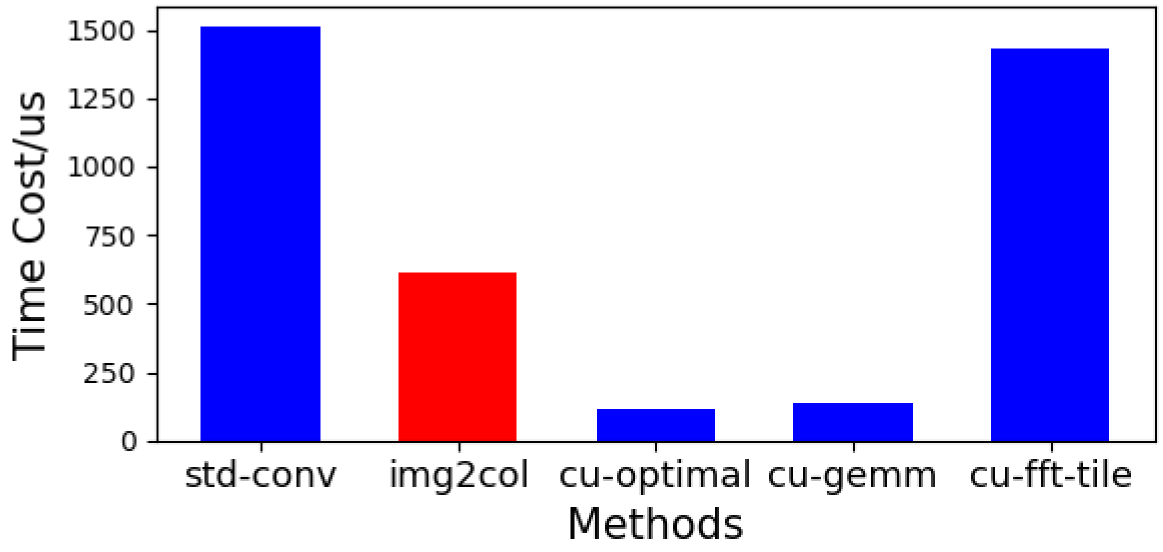
SJTU

Result Demonstration

Img2Col vs Standard Convolution

Convolution 3D

(32, 3, 3, 3) @ (1, 3, 244, 244)
-> (1, 32, 122, 122)



1509, 611, 112, 138, 1428 us

Methods for Test

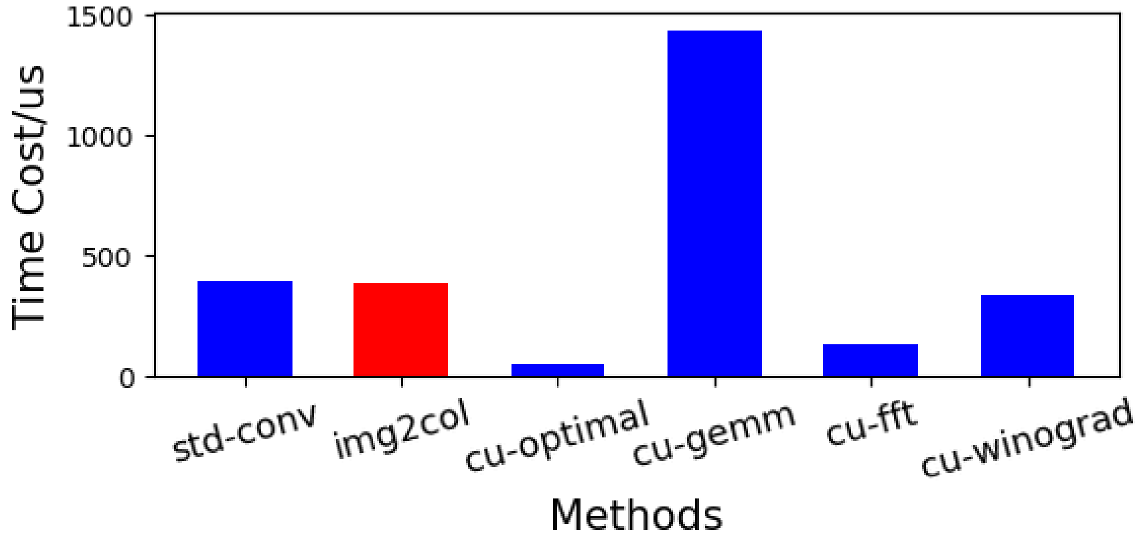
- Standard Conv by Loop
- Image to Column
- FFT Conv(cudnn)
- FFT Tiling Conv (cudnn)
- Winograd Conv (cudnn)
- Img2Col Conv (cudnn)

Result Demonstration

Img2Col vs Standard Convolution

Depth-wise Conv

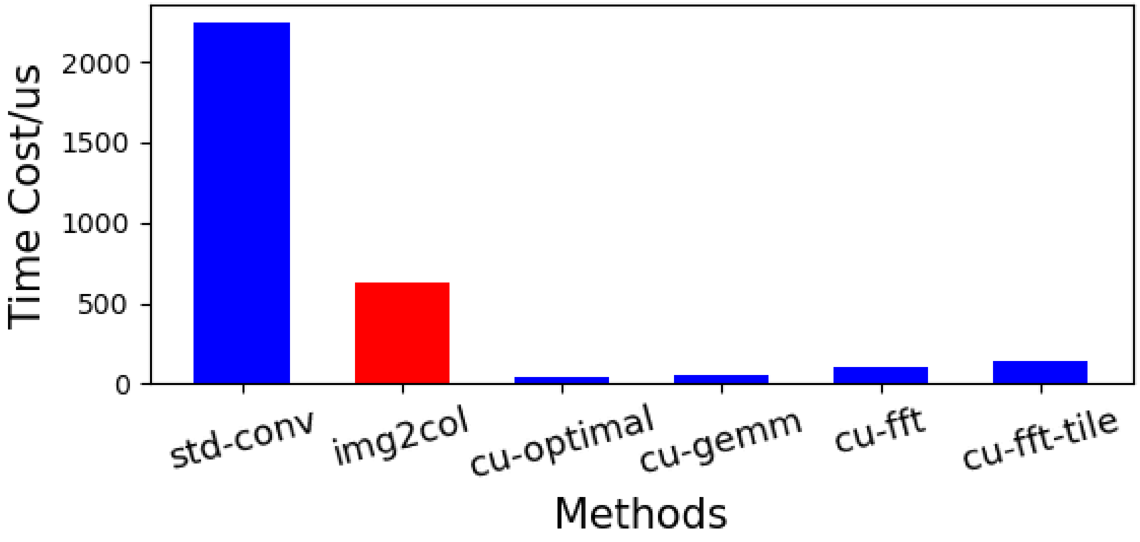
(32, 1, 3, 3) @ (1, 32, 122, 122)
-> (1, 32, 122, 122)



396, 388, 51, 1440, 138, 338 us

Point-wise Conv

(16, 32, 1, 1) @ (1, 32, 122, 122)
-> (1, 16, 122, 122)

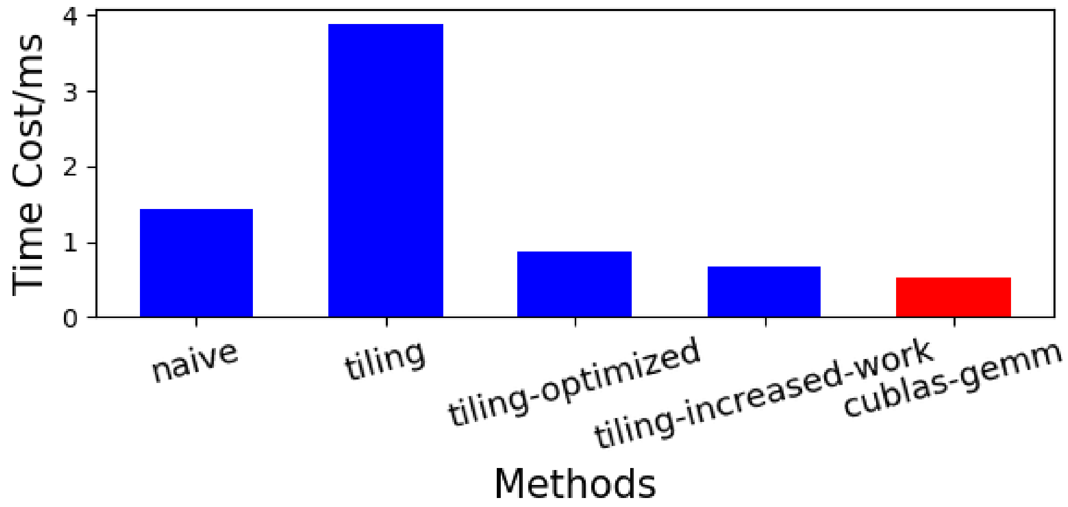


2246, 634, 45, 50, 108, 144 us

Matrix Multiplication Comparison

Matrix Multiplication

(32, 27) @ (27, 122 * 122)
-> (32, 14884)



1.43, 3.90, 0.87, 0.66, 0.53 ms

Large Cost for Cudnn to Create Handle!

```
cudnnHandle_t handle;
t1_handle = clock();
checkCUDNN(cudnnCreate(&handle));
t2_handle = clock();
printf("handle: %lf\n", (double)(t2_
```

Cudnn Handle: 461ms
Convolution Descriptor: 15ms

Other Optimizations

Rearrange Data

Output Position

Single Index (c, i, j)

Input Position

Input Position

Multiple Index (c, i, j)

Output Position

11-13us

8-13us

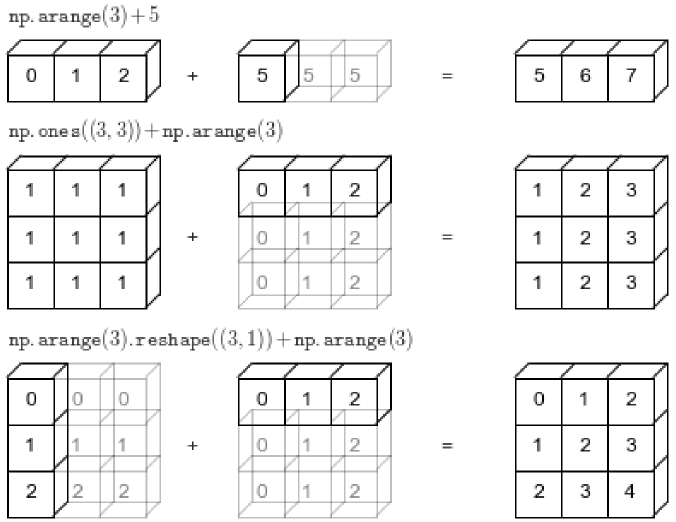
(K * K * C * H’ * W’) = (3, 3, 3, 122, 122)

Apply Const Memory

Bias Broad-Casting

W (32, 3, 3, 3)

b (32,)



上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

Many Thanks

Happy New Year!

饮 水 思 源　爱 国 荣 校