CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

CSDS 451: Designing High Performant Systems for AI

Lecture 11

9/30/2025

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

Outline

- Sparsity in Convolutions

Announcements

- WA 2 due by Friday
- PA 1 will be out by Saturday Morning
- Midterm next Thursday - 10/9
  - Will discuss logistics and other details in the next lecture
  - Will include topics till today's lecture

Project

- Two Types of Project

- Type #1 (HIP based):
  - Take a CNN based Machine Learning Model
  - Calculate the inference time of the model using pytorch
  - Implement each convolution layer using one of the convolution algorithm discussed in class using HIP
    - Convert the layer in correct matrix dimensions
    - Use your PA1 implementation of blocked matrix multiplication and gemm based matrix multiplication
  - Compare the inference times of pytorch, blocked mm with best parallelism parameter, and gemm based MM
  - Optional (for teams of size 3): we can discuss some addons, compare CO2 emissions of all three techniques, compare performance between different convolution algorithms, choose best convolution algorithm for each layer and then compare performance

Project

- Type #2
  - Take a LLM based machine learning model
  - Use pytorch lightning for distributed training and calculate time (baseline)
  - Using mpi4python, implement your own distributed training framework and compare performance
  - Optional (for teams of size 3): Use your HIP MM codes for implementing the matrix multiplication of the LLM

Project

- Not expecting an end to end framework that takes any model and performs the deployment.
- You can have code that calculates the times for each layer separately and sums them up
- Functional correctness is secondary, but there shouldn’t be obvious flaws.

Project

- Not expecting your implementation will definitely beat the performance of pytorch or pytorch-lightning
  - (It is definitely possible, my research student has implementations that perform 2-3x better than pytorch)

- Taking risks and failing will not be penalized
  - As long as you can show reasonable effort was put into the project

Project Deliverables

- 1 page project proposal (5 points)
  - Objective:
  - Approach:
  - Baselines for Comparison:
  - Metrics used for Comparison:
  - Definition of Success:
- Send me after having a 1-1 meeting with me

Project Deliverables

- Demonstration of the Project (10 points), includes
  - Code + data submission
  - Instructions and comments so that TA can reproduce the results
  - in-person or online walkthrough of the results

- Project Report (5 points)
  - 2 pages (can be larger too)
  - Expand the approach section of the project proposal with actual implementation details and challenges that you faces
  - Add an Experimental Results section to the project proposal that
    - Discusses your findings
    - Your thoughts or comments about any patterns you saw in the finding (most critical – simply saying algorithm A performs better than algorithm B in all cases is not a very thoughtful comment. Tell us why you think that is the case)

Project – Next Action

- Team information due by 10/16
  - I will upload an assignment for this
- Preferred Team size – 2 students per team
- 3 or 1 will be allowed only if you make a strong case for it

Outline

- Sparsity in Convolutions

Reading Materials

- Cheng, Hongrong, Miao Zhang, and Javen Qinfeng Shi. "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations." *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

Key Operation in CNN

- Forward Propagation:

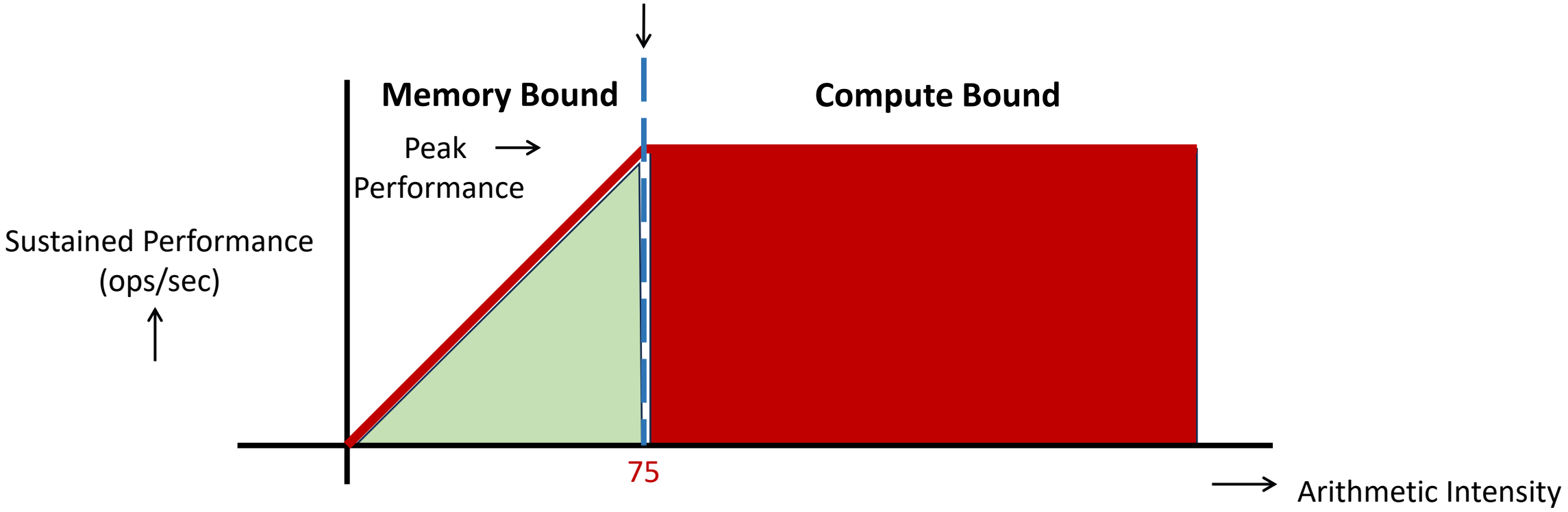
$$O_l = Conv(W_l, I_l)$$

- Backward Propagation

$$\frac{\delta E(W)}{\delta W_l} = Conv(I_l, \frac{\delta E(W)}{\delta O_l})$$

$$\frac{\delta O_{l+1}}{\delta O_l} = Conv(W_{180}, \frac{\delta E(W)}{\delta O_{l+1}})$$

CNN on A100 Roofline Plot (assuming infinite cache)



Which area will the CNN Performance lie in? Compute Bound

Implications

- Essentially a Compute Bound Problem, assuming large cache
- Objective: How to ensure efficiency is $O(1)$, i.e., all compute units are fully utilized
- If cache is limited
  - Objective 1: How to improve data reuse so that compute units have enough to work on?
  - Objective 2: How to ensure efficiency is $O(1)$, i.e., all compute units are fully utilized
- Another approach: increase Number of Devices
  - + Larger On-chip memory
  - + Higher bandwidth
  - - Communication/Synchronization overheads

Convolution as Matrix Multiplications

- Transform the input features and kernels into matrices and perform matrix multiplication.
- Theoretically, matrix multiplication algorithms achieve cost optimality and good data reuse
- Practically, every hardware vendor provides optimized implementations of matrix operations

Im2Col Algorithm

- Input Matrix -$(k \times k \times C_{in}) \times (H \times W)$
  - $k \times k$ increase in size -> leads to increase in memory transfers
- Filter Matrix - $(\mathrm{C}_{out}) \times (k \times k \times C_{in})$
  - No increase in size
- Output Matrix - $C_{out} \times (H \times W)$
  - No increase in size

Kn2Row Algorithm

- Input Matrix -$(C_{in}) \times (H \times W)$
- Filter Matrix - $\left(\mathrm{C}_{out} \times k^2\right) \times (C_{in})$
  - No increase in size
- Output Matrix -$(H \times W) \times (C_{out} \times k^2)$
  - $k \times k$ increase in size -> leads to increase in memory transfers

Scalar Matrix Multiplication

- Input Matrix -$(C_{in}) \times (H \times W)$
  - No increase in size
- Filter Matrix - $(\mathrm{C}_{out}) \times (k \times k \times C_{in})$
  - No increase in size
- Output Matrix - $C_{out} \times (H \times W)$
  - No increase in size
- Performs Matrix Scaling and Addition Operations

Sparsity in Convolution

- Algorithms discussed till now do not change CNN models
  - Functionally Correct
  - No change in the number of operations performed

- **Sparsity:** An orthogonal research direction is to reduce the number of operations performed and memory transferred in the CNN, while ensuring accuracy loss is not too high

Introducing Sparsity

- Two main techniques
- #1: Remove some weights of the kernels - pruning
- #2: Reduce the size of data type used to represent weights of the kernels - quantization

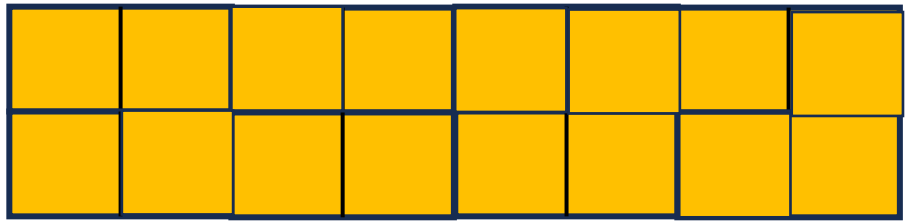
Pruning

- Two types:
- Unstructured Pruning: Remove weights randomly (or using some rules) without considering the impact on hardware execution
- Structured Pruning: Remove weights in a structured manner, so that
  - Non-zero elements are packed together, data transfers utilize maximum bandwidth due to sequential access

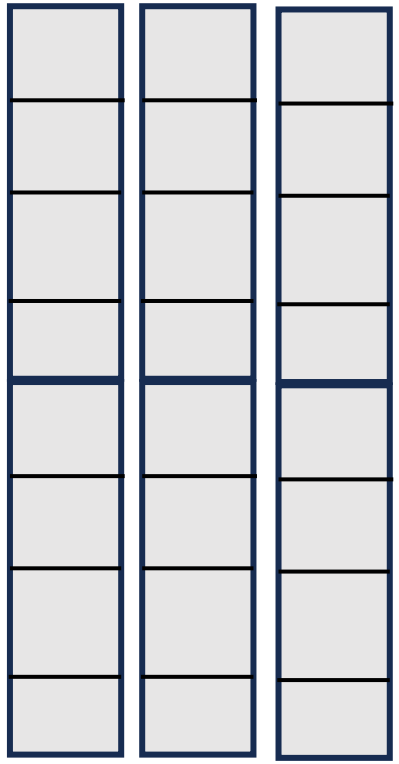
Unstructured Pruning

- Remove weights randomly (or using some rules) without considering the impact on hardware execution

- Consider Im2col based convolution



Kernel

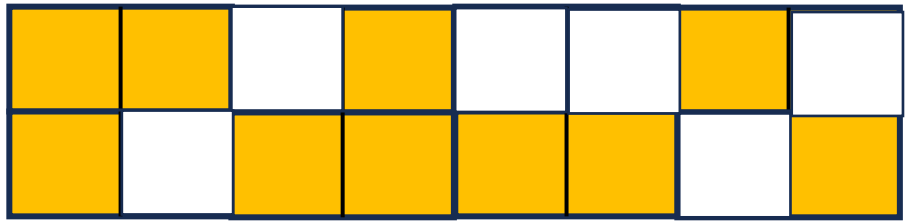


Inputs

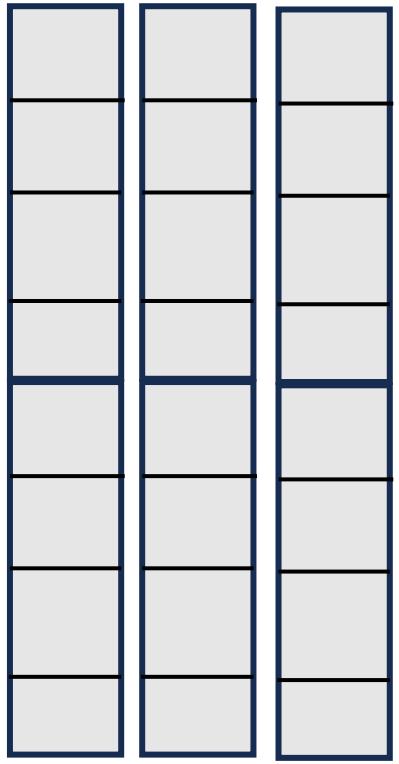
Unstructured Pruning

- Remove weights randomly (or using some rules) without considering the impact on hardware execution

- Consider Im2col based convolution



Kernel



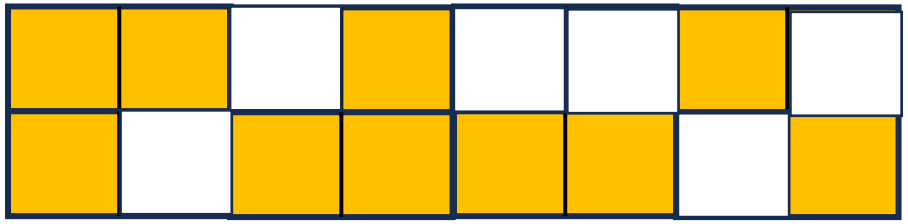
Inputs

Unstructured Pruning - Approaches

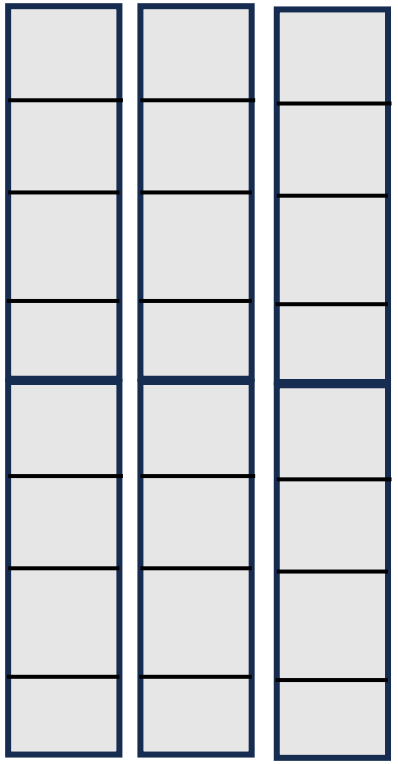
- Magnitude based pruning
  - Example: In a trained model, keep only top-k percentage of weights in terms of magnitude or norm
  - Can also be done iteratively: Train a model, remove x% of weights, fine-tune the model, remove another x% and so on.
- Gradient based pruning
  - Removes weights whose gradients with respect to the error function is lower
- Refer to the survey paper for a thorough discussion on various pruning techniques.
  - We are simply interested in how to efficiently run a pruned model in this class

Unstructured Pruning

- Consider Im2Col based Convolution



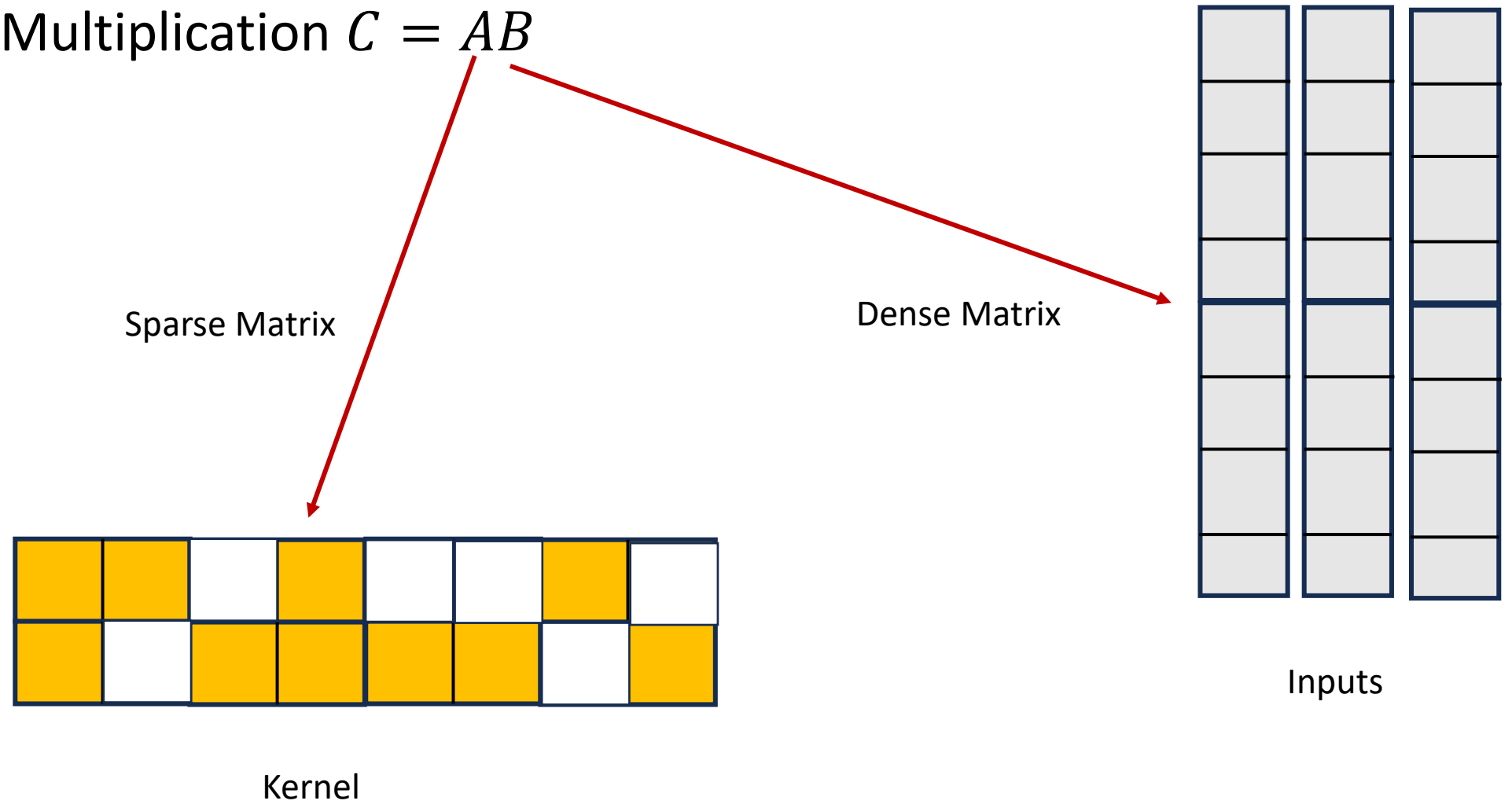
Kernel



Inputs

Unstructured Pruning

- Matrix Multiplication $C = AB$

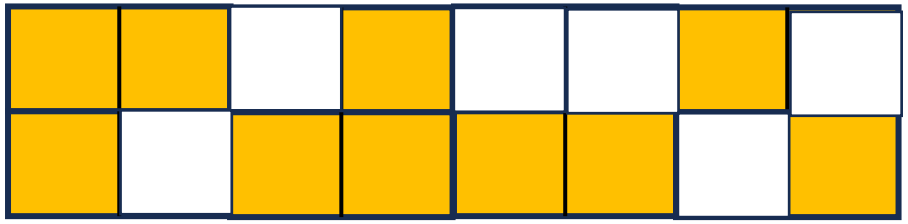


Sparse Dense Matrix Multiplication (SpMM)

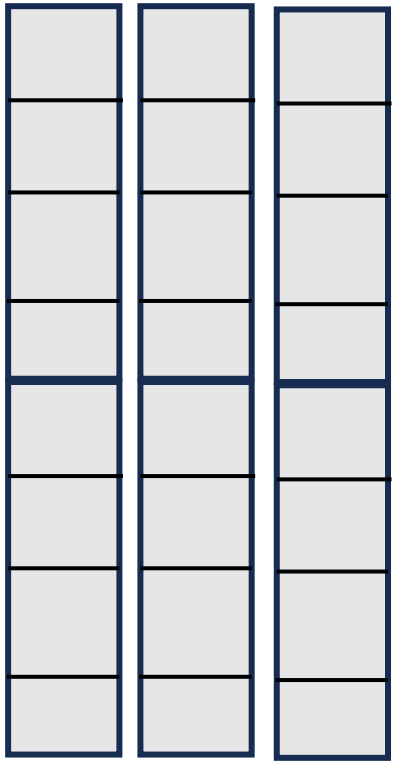
- An important kernel in Scientific Computing as well as Machine Learning applications
- Extensive Research has been performed (and still continuing) on accelerating SpMM
- Key Challenge: Cannot use dense-dense matrix multiplication techniques
  - Inefficient Storage
  - Non-useful computations
- Optimizations
  - Efficient Storage of Sparse Matrices
  - Work optimal task scheduling
- **We will discuss SpMM acceleration in Transformer lectures.**

Unstructured Pruning

- Consider Im2Col based Convolution



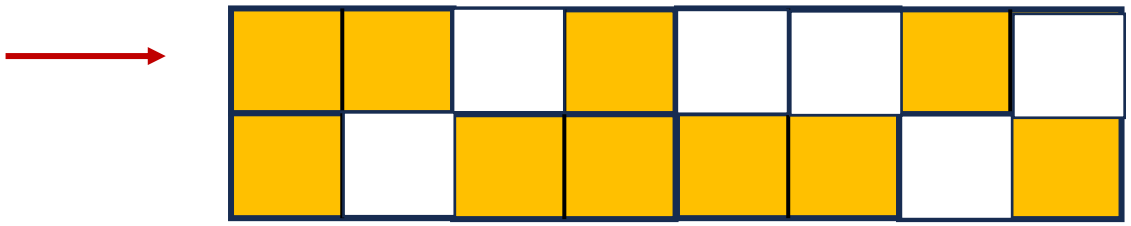
Kernel



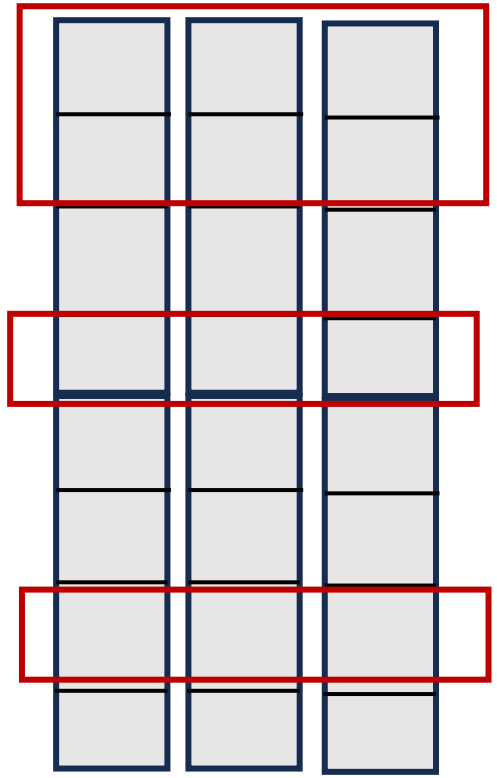
Inputs

Unstructured Pruning

- Consider Im2Col based Convolution
- Highlighted elements of Input need to be contiguous



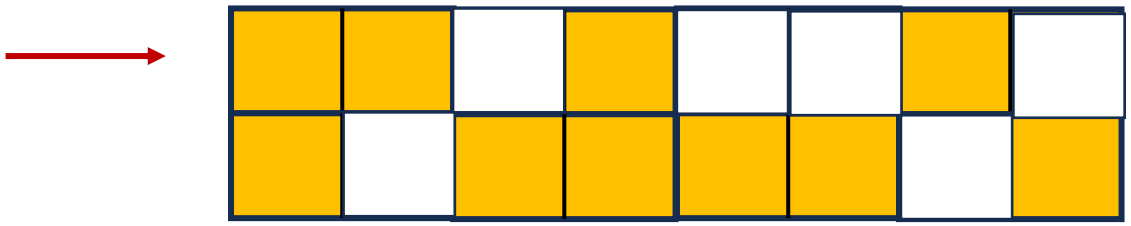
Kernel



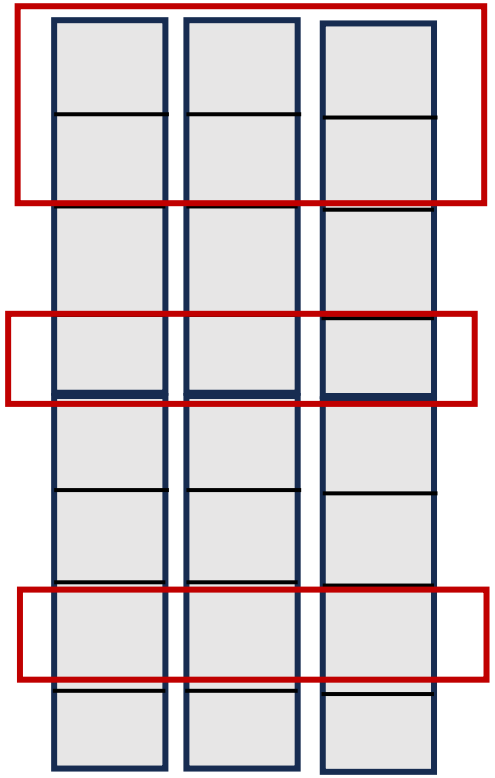
Inputs

Unstructured Pruning

- Consider Im2Col based Convolution
- Highlighted elements of Input need to be contiguous why??



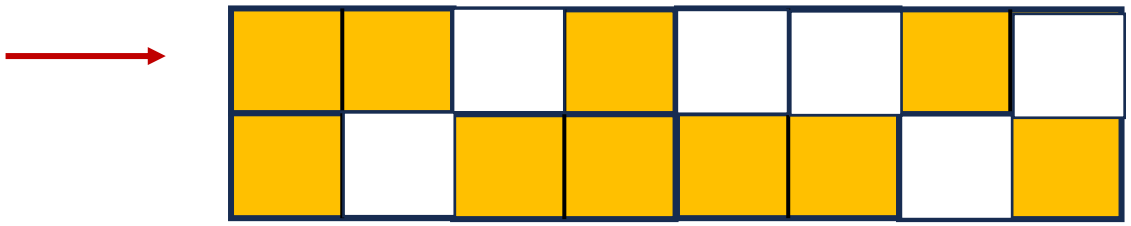
Kernel



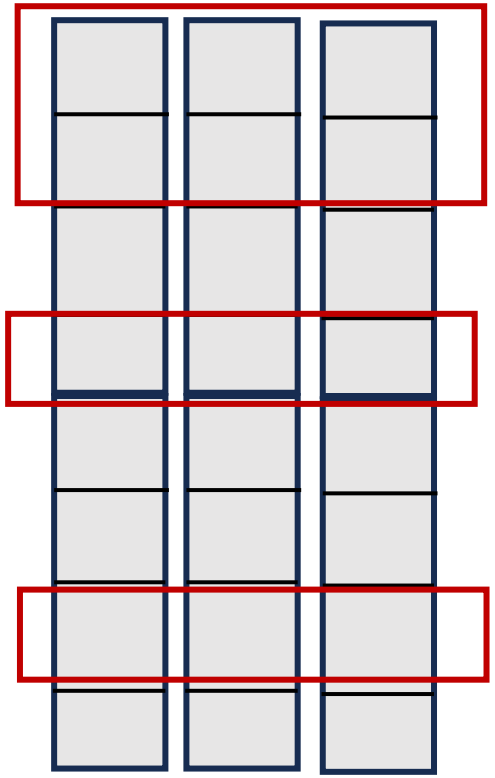
Inputs

Unstructured Pruning

- Consider Im2Col based Convolution
- Highlighted elements of Input need to be contiguous
  - Sequential layout leads to bandwidth maximization and latency reduction



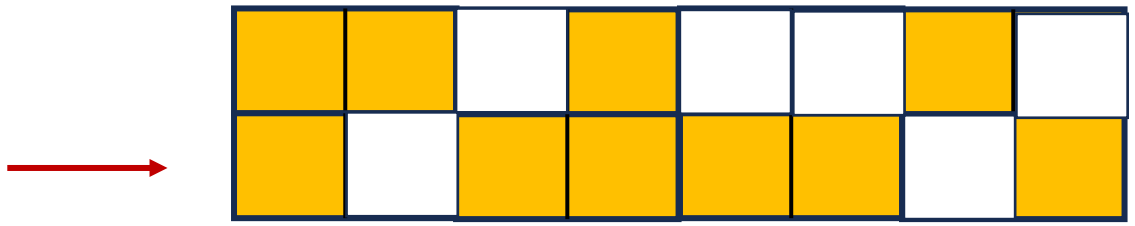
Kernel



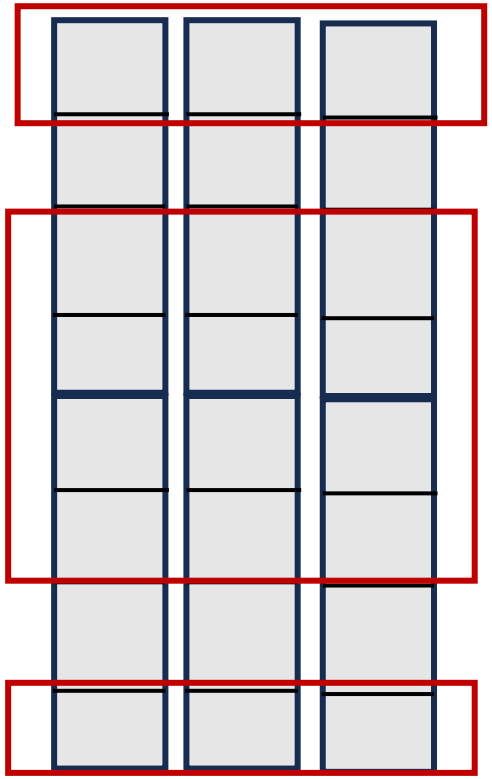
Inputs

Unstructured Pruning

- Consider Im2Col based Convolution
- Highlighted elements of Input need to be contiguous
- No single sequential storage of Inputs works for both rows of the kernel



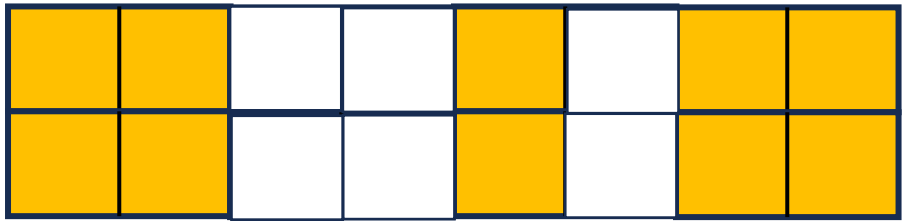
Kernel



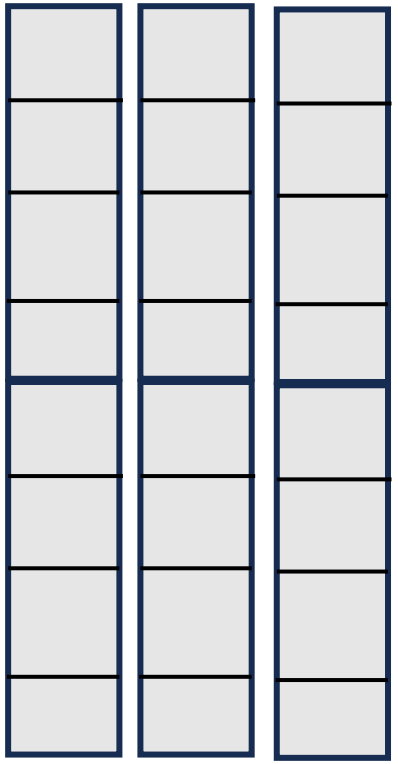
Inputs

Structured Pruning

- Consider Im2Col based Convolution



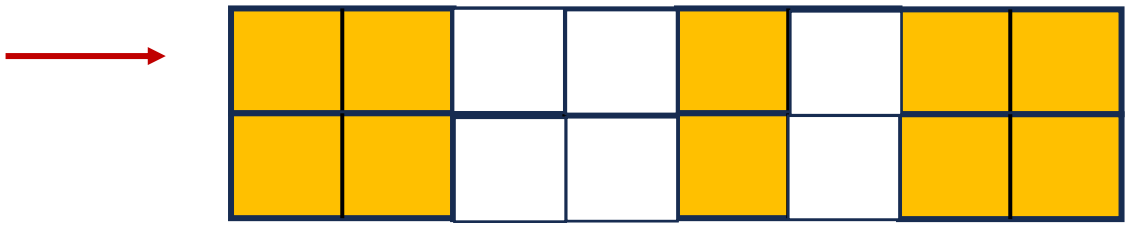
Kernel



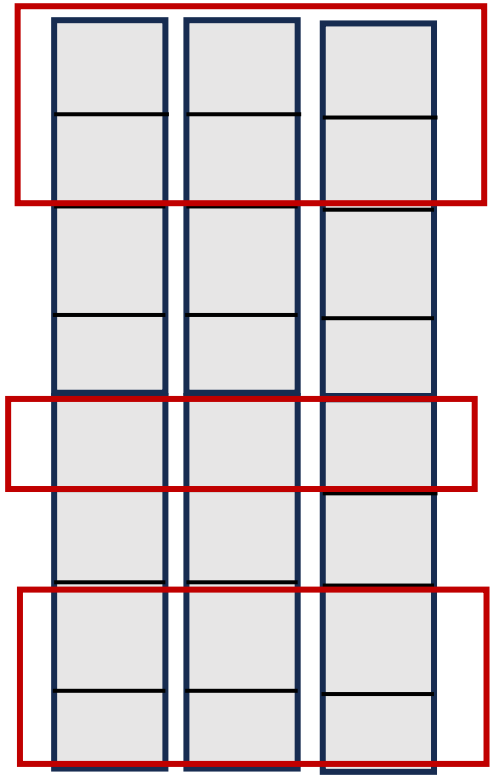
Inputs

Structured Pruning

- Consider Im2Col based Convolution
- Highlighted elements of Input need to be contiguous



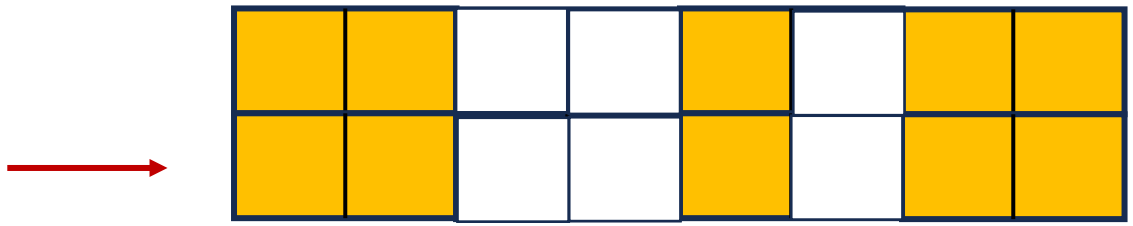
Kernel



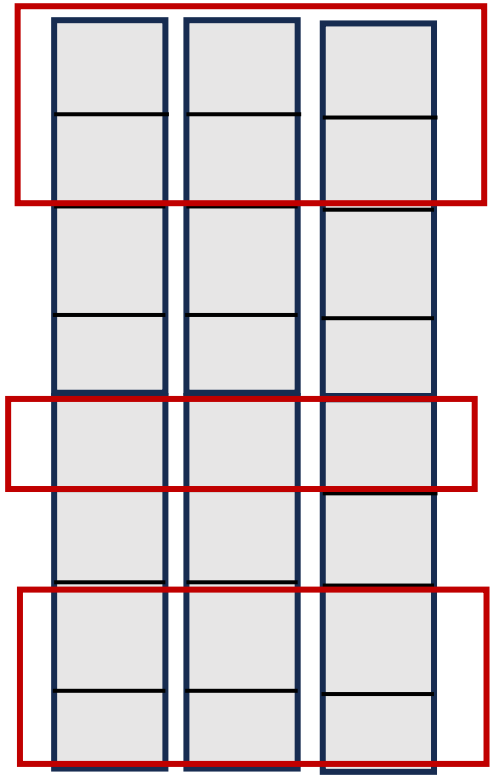
Inputs

Structured Pruning

- Consider Im2Col based Convolution
- Highlighted elements of Input need to be contiguous
- Same sequential storage of Inputs works for both rows of the kernel



Kernel



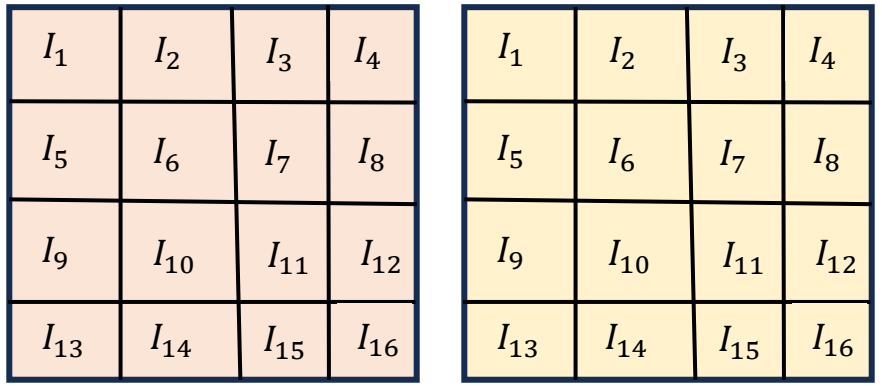
Inputs

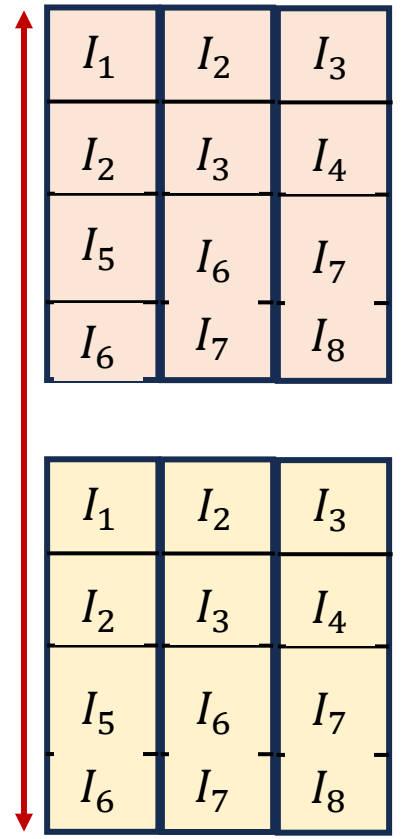
Structured Pruning

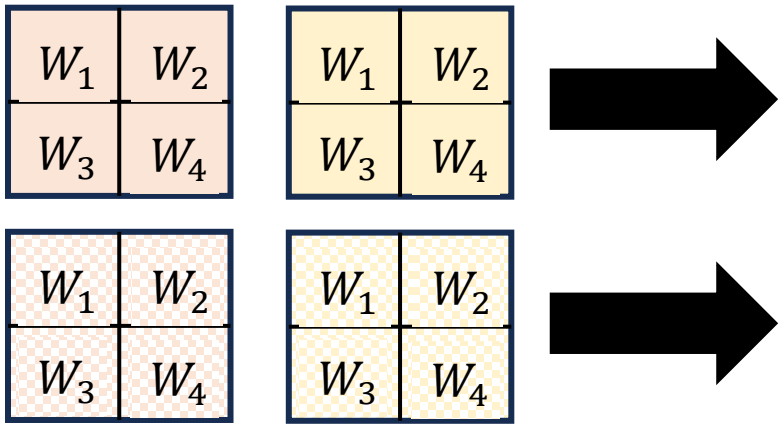
- Objective is to achieve structure in pruning that makes it hardware friendly
  - For example, by ensuring same sequential storage of inputs work for all channels in the previous example

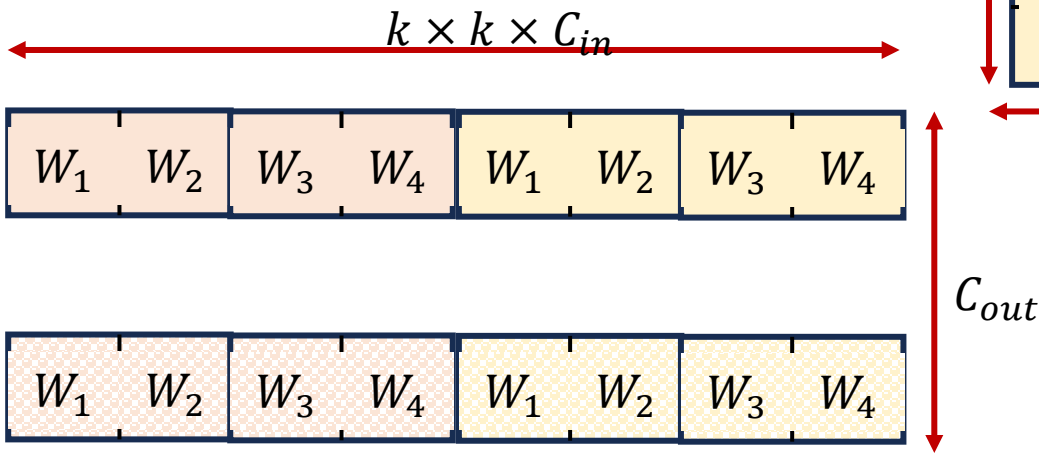
- Examples of “structures” that can be pruned
  - Entire Channels
  - Entire Filters
  - Entire Layers

Structured Pruning – Output Channel Pruning

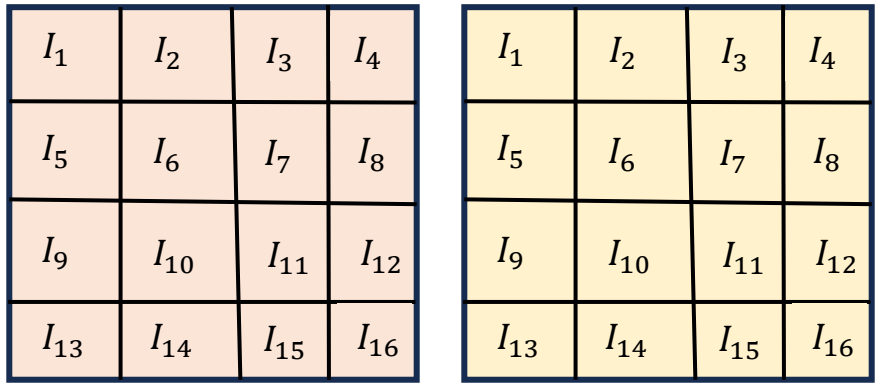


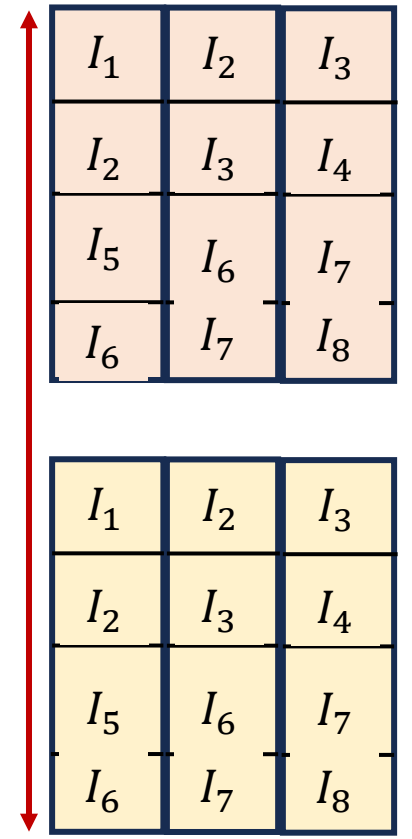


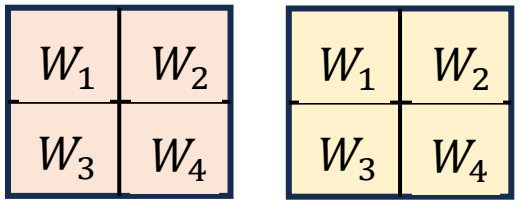


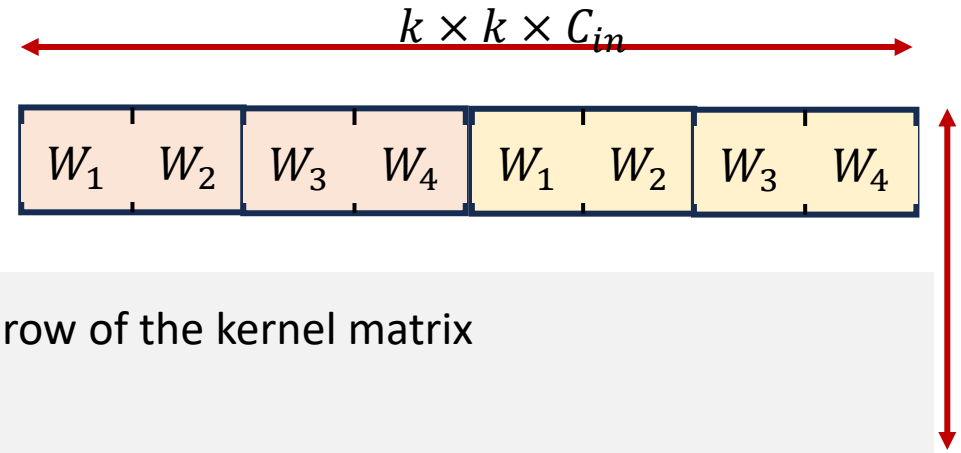


Structured Pruning – Output Channel Pruning





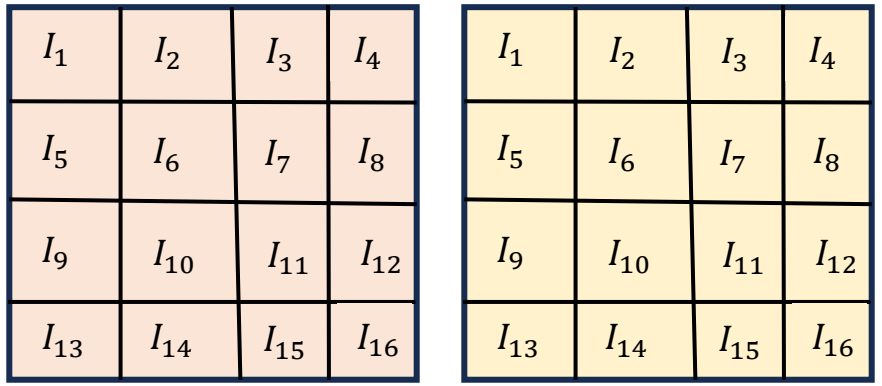


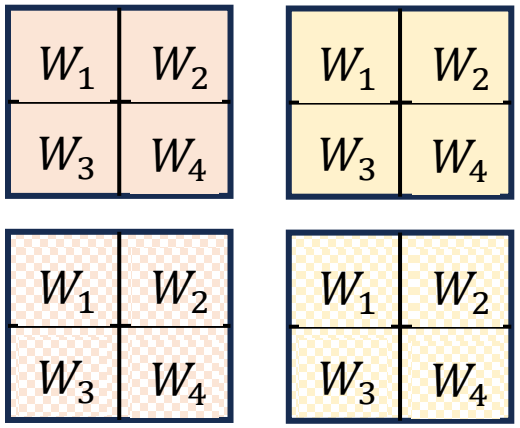


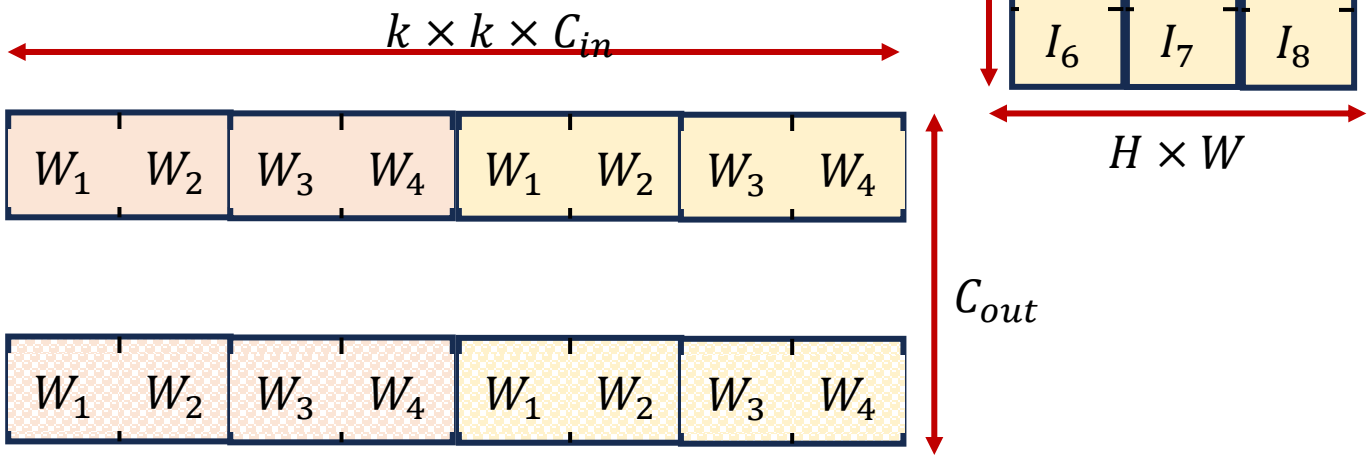
Avoid Matrix-Vector multiplication for an entire row of the kernel matrix

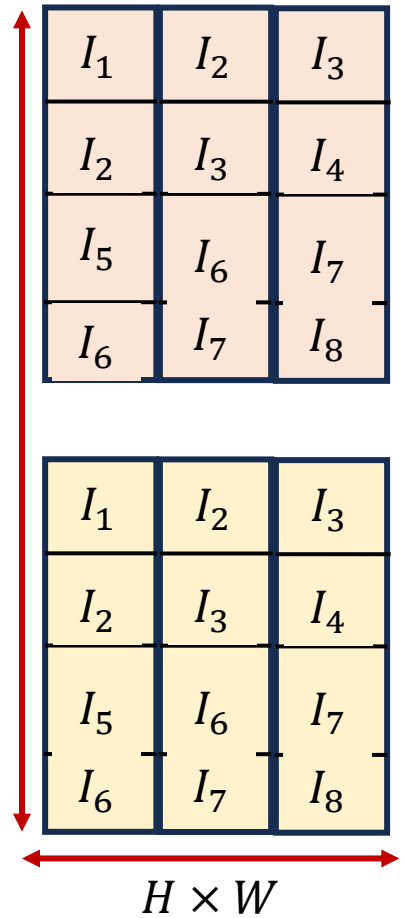
- Save $k \times k$ memory
- Save $k \times k \times C_{in} \times H \times W$ computations

Structured Pruning – Input Channel Pruning







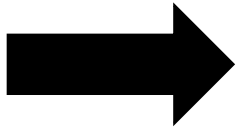


Structured Pruning – Input Channel Pruning

I_1	I_2	I_3	I_4
I_5	I_6	I_7	I_8
I_9	I_{10}	I_{11}	I_{12}
I_{13}	I_{14}	I_{15}	I_{16}

Avoid Matrix-Matrix Multiplication for a smaller block

- Save $k \times k \times (C_{out}+HW)$ memory
- Save $C_{out}HWk^2$ computations



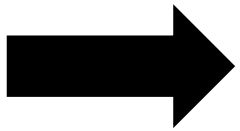
I_1	I_2	I_3
I_2	I_3	I_4
I_5	I_6	I_7
I_6	I_7	I_8

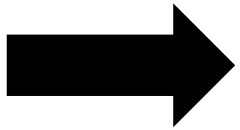
$k \times k \times C_{in}$

$H \times W$

W_1	W_2
W_3	W_4

W_1	W_2
W_3	W_4





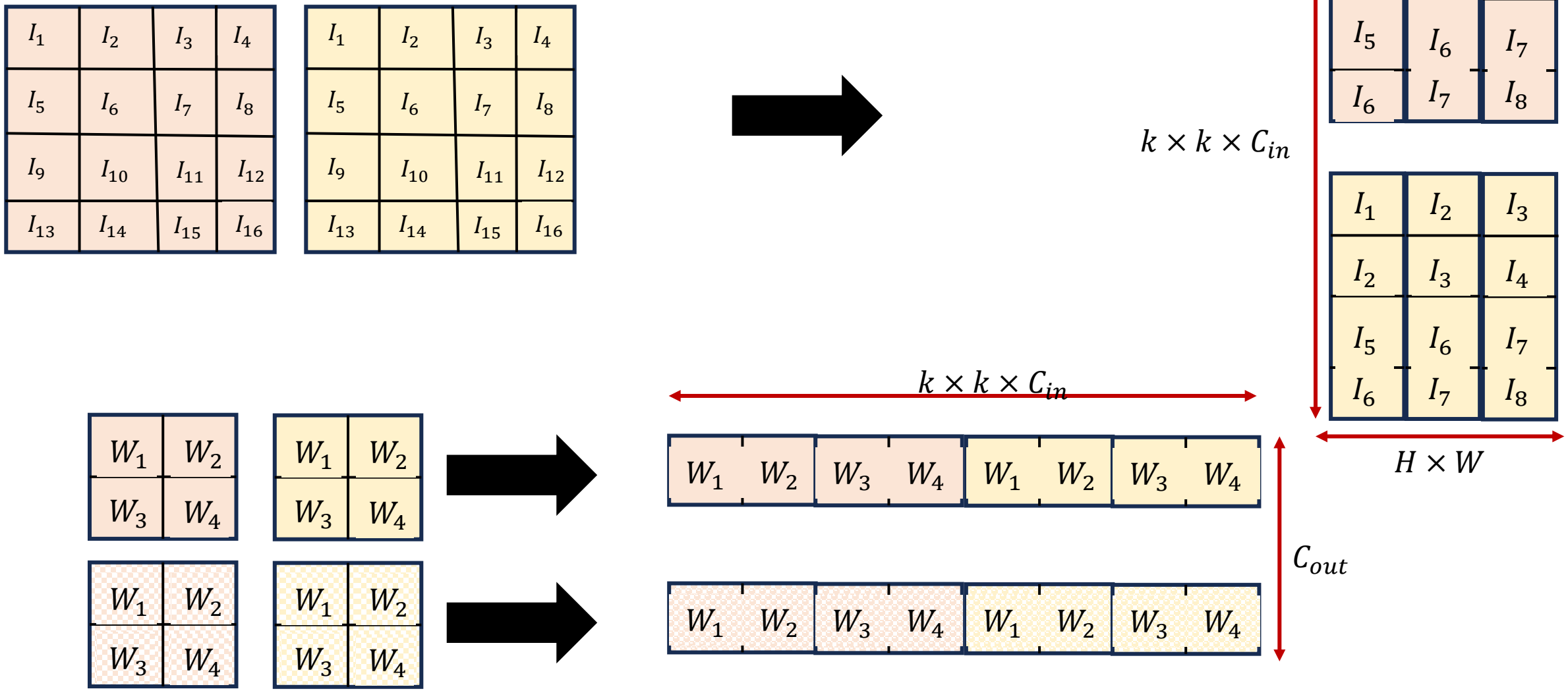
$k \times k \times C_{in}$

W_1	W_2	W_3	W_4

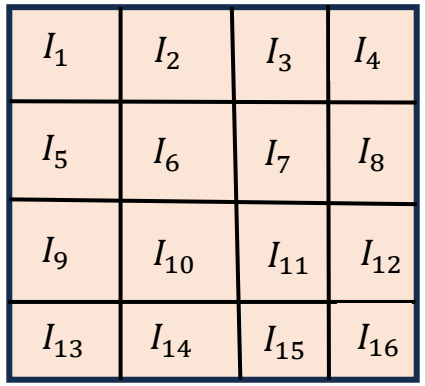
W_1	W_2	W_3	W_4

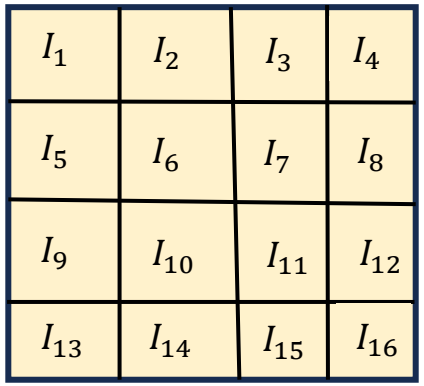
C_{out}

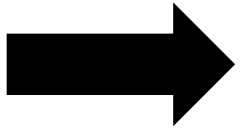
Structured Pruning – Filter/Connection Pruning



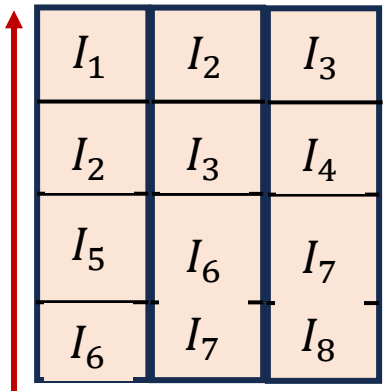
Structured Pruning – Filter/Connection Pruning

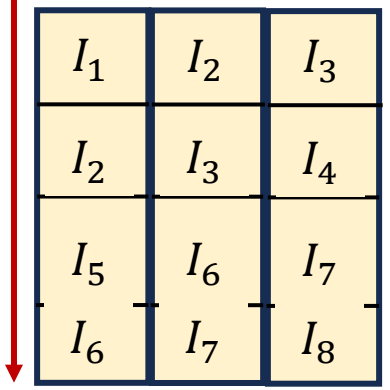






$k \times k \times C_{in}$

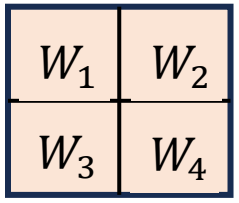


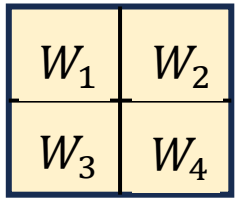


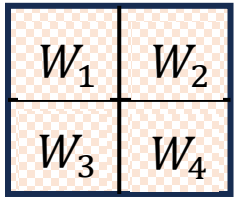
$H \times W$

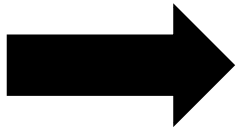
Avoid Matrix-Vector Multiplication for a smaller block

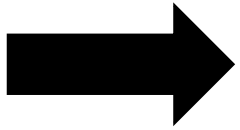
- Save $k \times k$ memory
- Save HWk^2 computations



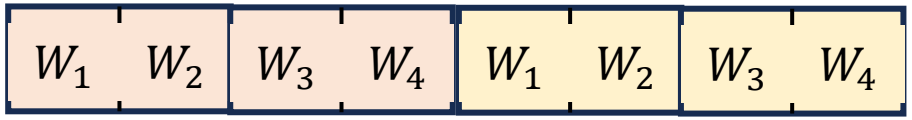


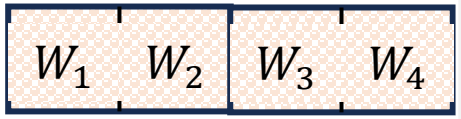






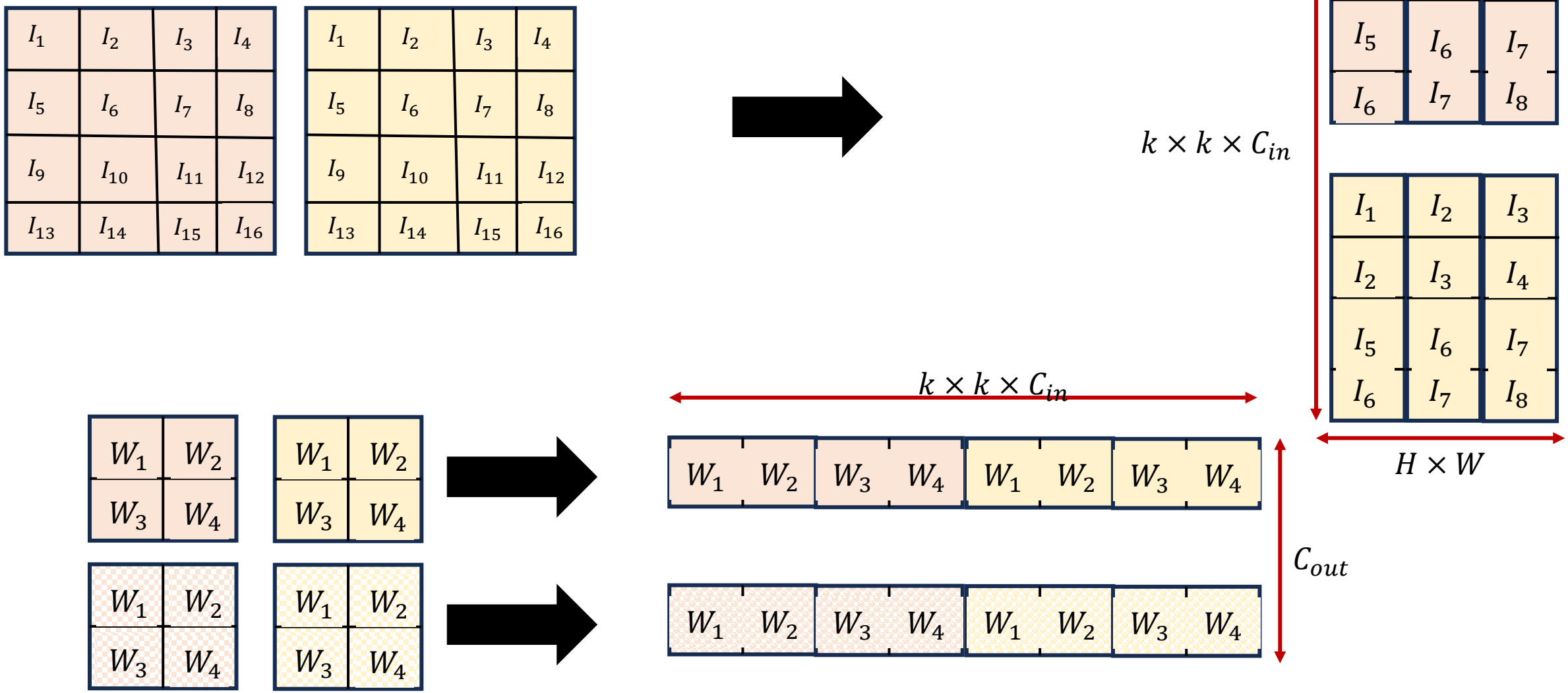
$k \times k \times C_{in}$





C_{out}

Structured Pruning – Layer Pruning



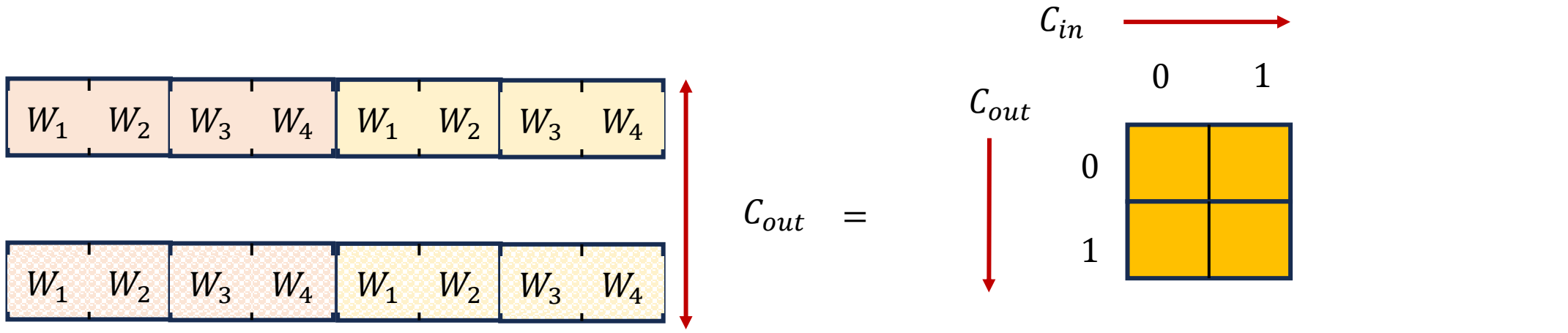
Structured Pruning – Layer Pruning

Structured Pruning

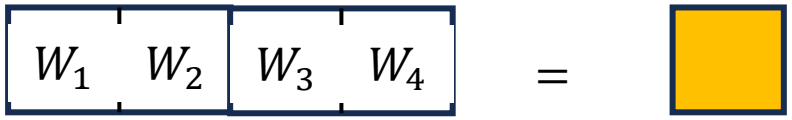
- Pruning entire layers or channels lead to significant reduction in memory and computation
  - But may lead to significant accuracy drop
- Pruning filters/connections lets you prune at a finer granularity level (compared to layers or channels) and may preserve accuracy
- In practice, the best pruning technique entirely depends upon the application domain

Grouped Convolutions

- A type of Filter/Connection pruning



Each cell here is an entire $k \times k$ kernel



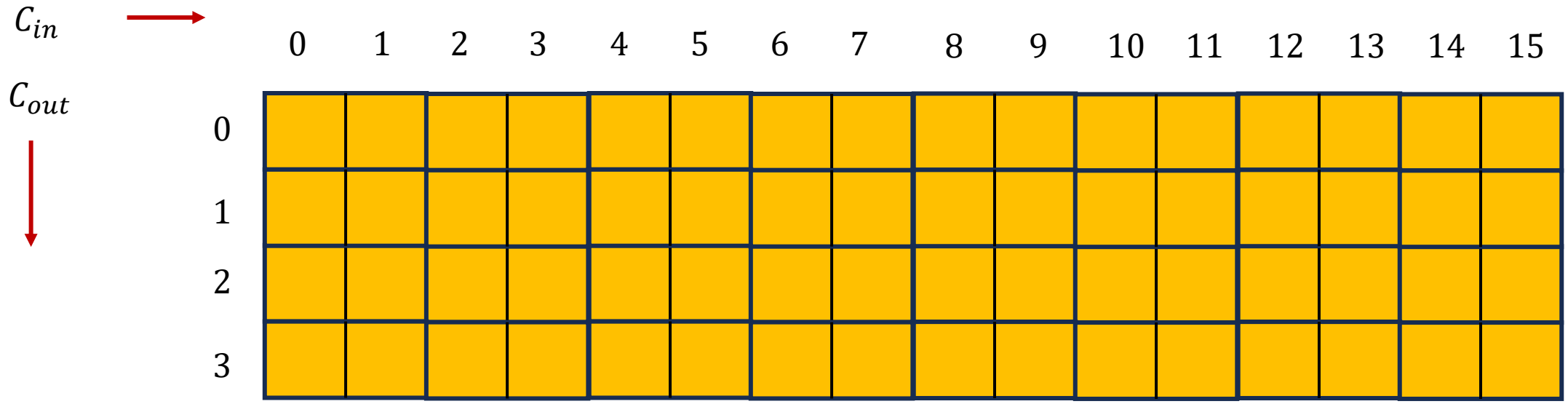
Grouped Convolutions

- Recall the definition of CNN
- $C_{in} \times C_{out}$ kernels of size k^2
- Each kernel is a connection between $c_i \in C_{in}$ input channel and $c_o \in C_{out}$ output kernel

Grouped Convolution

- Key Idea: Remove a subset of $C_{in} \times C_{out}$ connections by:
  - Partition C_{in} input channels into g groups
  - Partition C_{out} output channels into g groups
  - Only include connections between input and output channels when both belong to same group id $g_{id} \in g$

Grouped Convolution

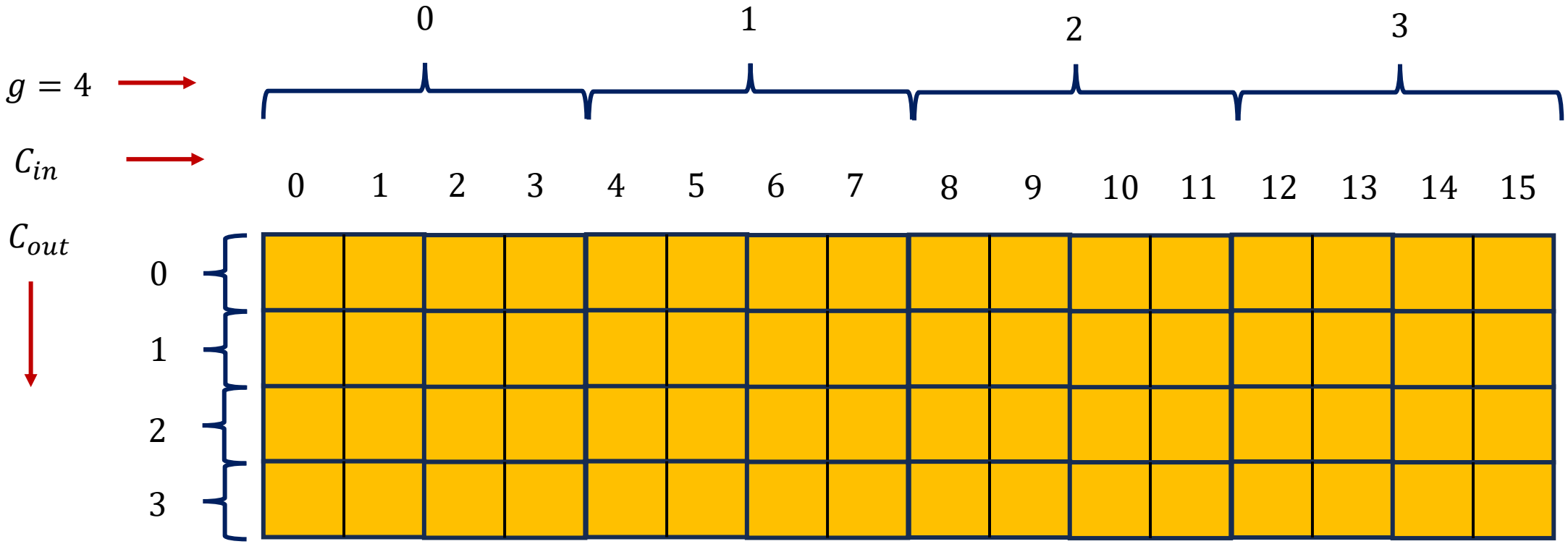


Note: Each cell here is an entire $k \times k$ kernel

Kernel

Grouped Convolution

- Partition C_{in} input channels into g groups
- Partition C_{out} output channels into g groups

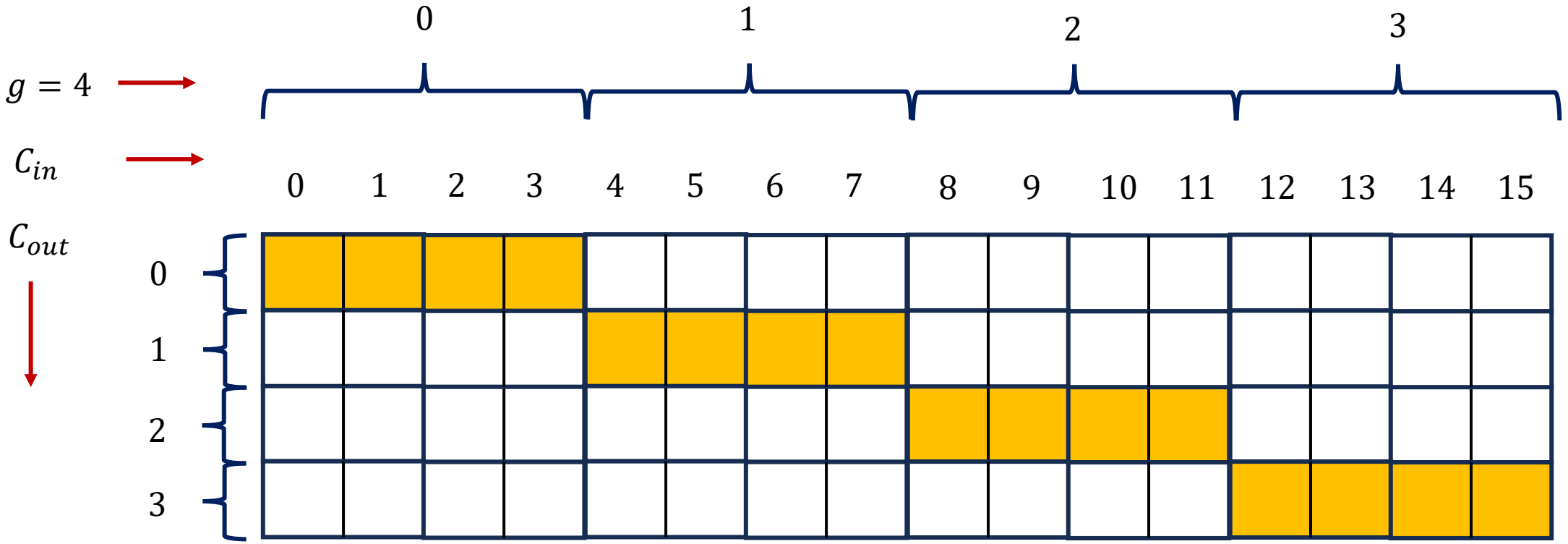


Note: Each cell here is an entire $k \times k$ kernel

Kernel

Grouped Convolution

- Only include connections between input and output channels when both belong to same group id $g_{id} \in g$



Note: Each cell here is an entire $k \times k$ kernel

Kernel

Grouped Convolution

- Total number of connections per group

$$\frac{C_{in}}{g} \times \frac{C_{out}}{g}$$

- Total number of groups = g

- So, total connections:

$$g \times \frac{C_{in}}{g} \times \frac{C_{out}}{g}$$

Grouped Convolution

- Total connections:

$$g \times \frac{C_{in}}{g} \times \frac{C_{out}}{g}$$

- Operations per connection: $H \times W \times k^2$

- So, total operations = $H \times W \times k^2 \times C_{in} \times C_{out} \times \frac{1}{g}$

- A factor of g reduction in computation

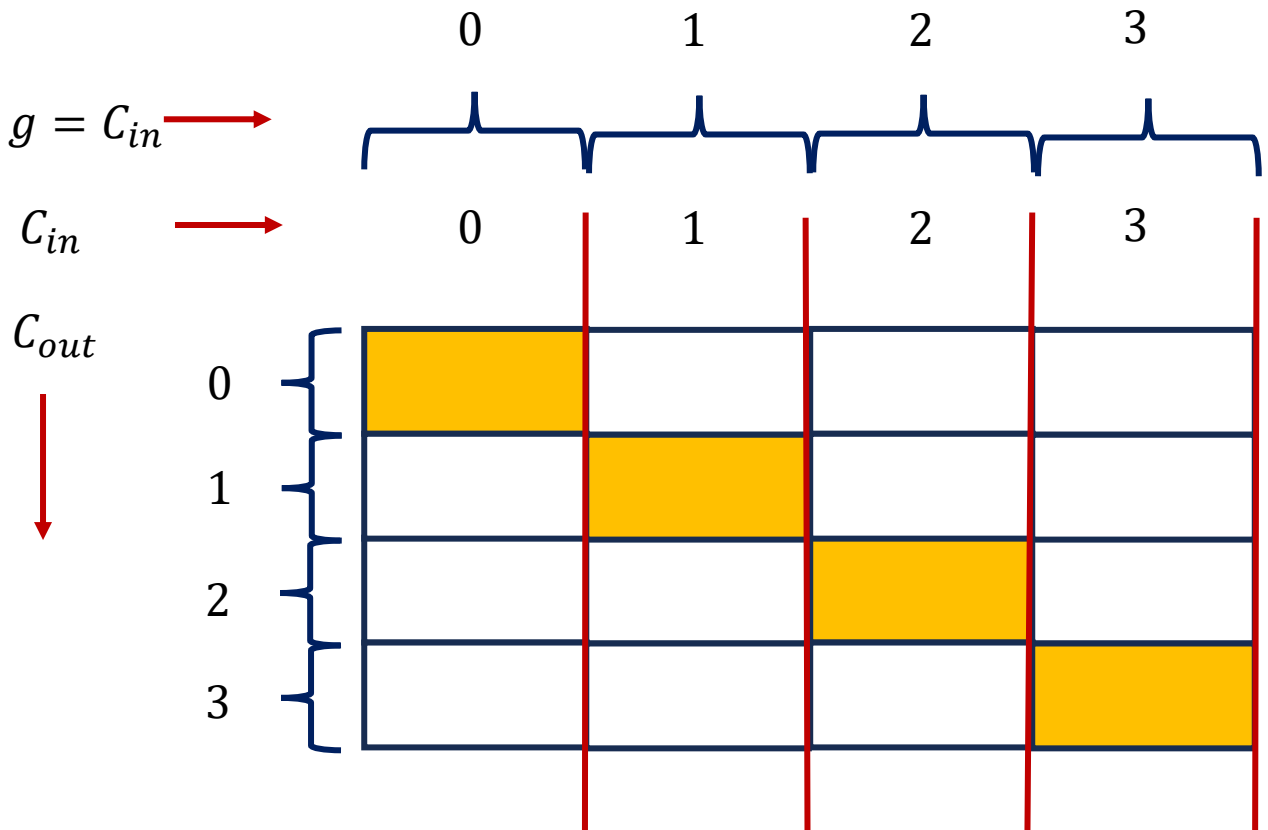
Depthwise Separable Convolution

- Key to achieving extremely memory and compute efficient CNN models, e.g., MobileNets

- Each Convolution Layer composed of two operations
  - Depthwise Convolution
  - Pointwise Convolution

Depthwise Separable Convolution

- Depthwise Convolution: Grouped Convolution with following characteristics
- $C_{out} = C_{in}, g = C_{in}$



Note: Each cell here is an entire $k \times k$ kernel

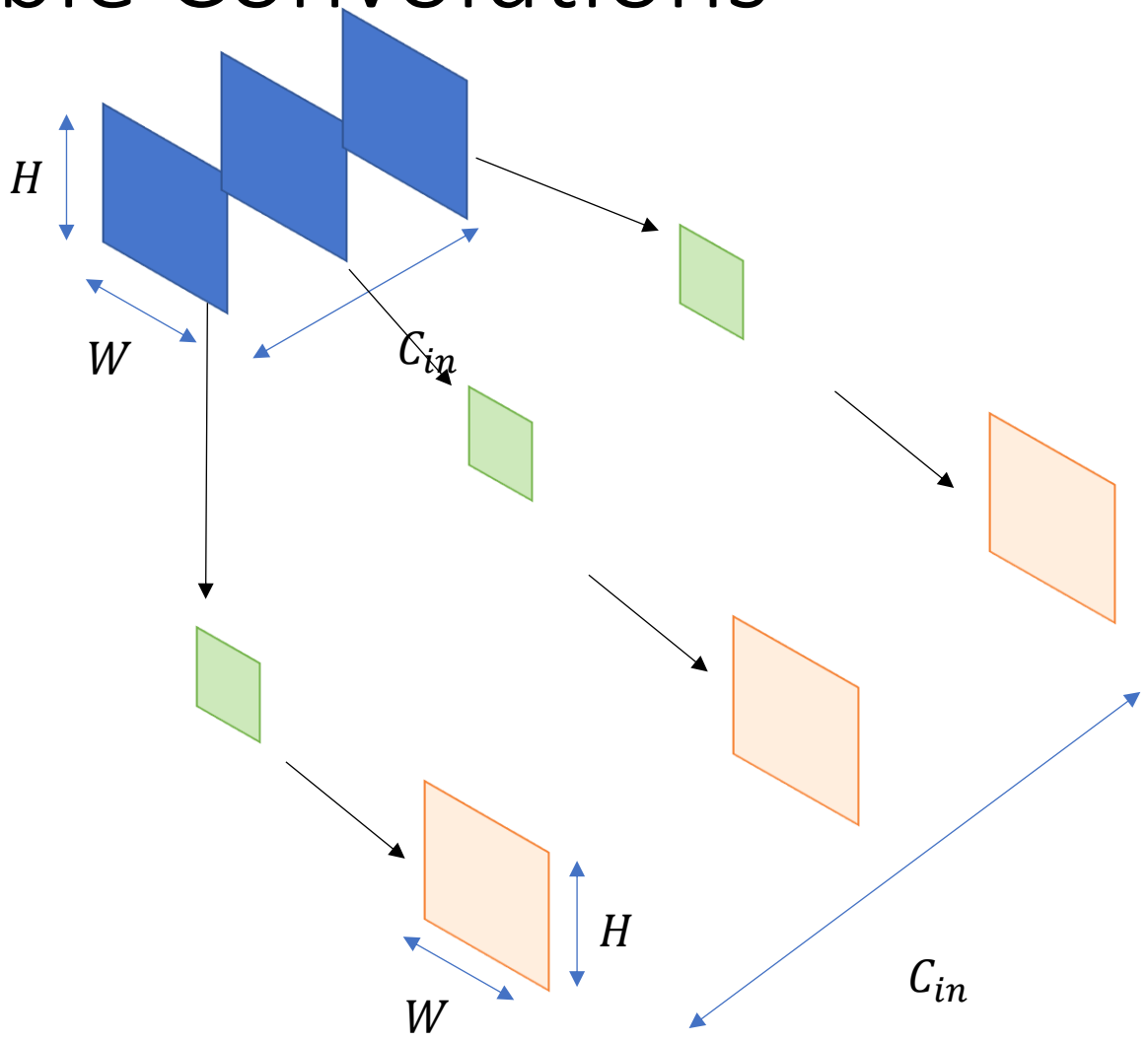
Depthwise Separable Convolution

- Output of depthwise convolution is C_{in} images of size $H \times W$

- Pointwise Convolution:
  - $C_{in} \times 1 \times 1$ Convolutions on the images for each output channel
  - So, C_{out} stacks of $C_{in} \times 1 \times 1$ convolutions

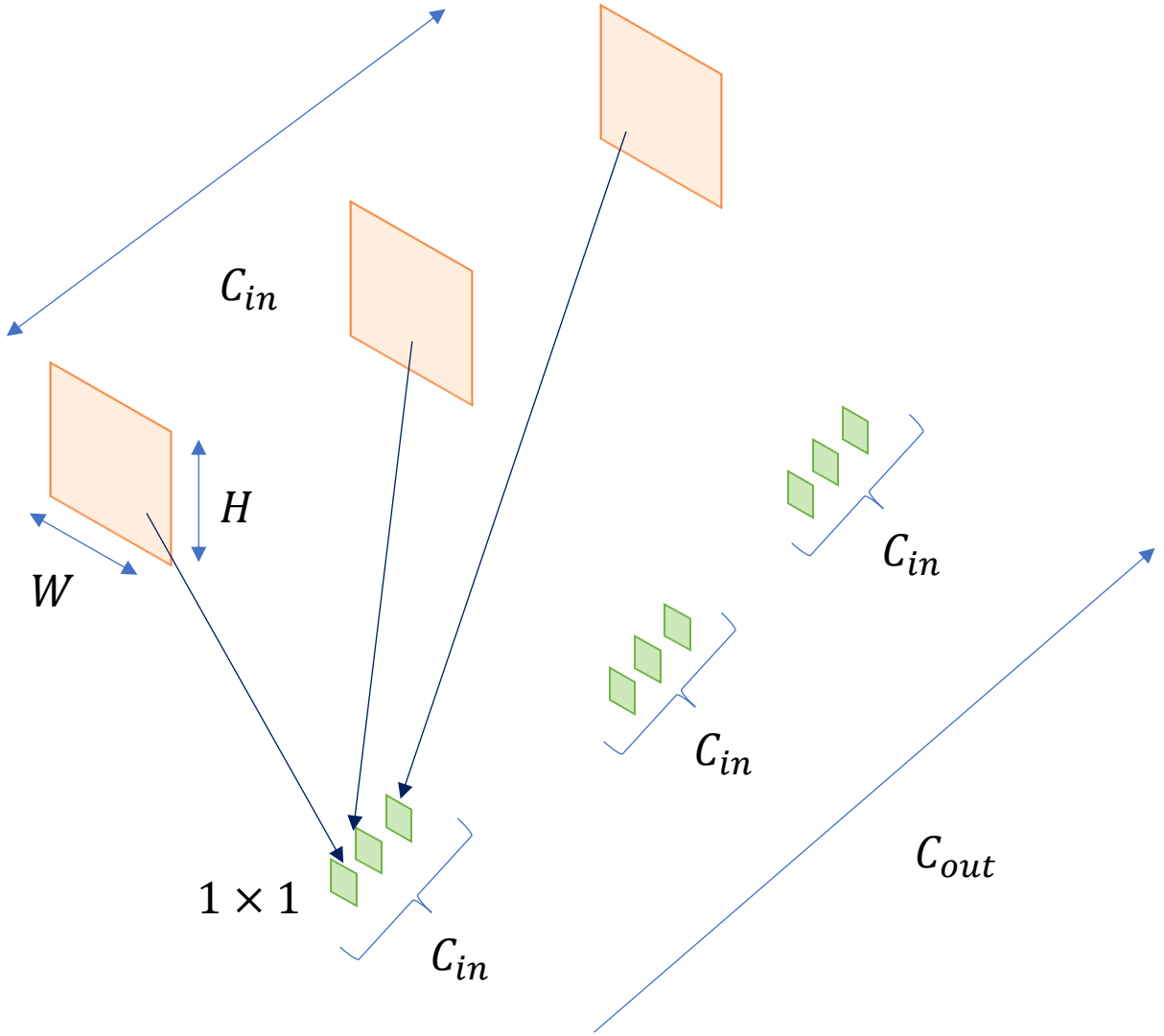
Depthwise Separable Convolutions

- Depthwise Convolution
- Note: No pixel wise addition



Depthwise Separable Convolutions

- Pointwise Convolution
- Each C_{out} stacks of kernels perform 1×1 convolution on $C_{in} \times H \times W$ images



Depthwise Separable Convolutions

- Depthwise Convolutions Number of Operations (substitute $C_{out} = C_{in}, g = C_{in}$ in grouped convolution equation)

$$H \times W \times k^2 \times C_{in}$$

- Pointwise Convolution (Full convolution with $k = 1$)

$$H \times W \times C_{in} \times C_{out}$$

- Original Convolution: $H \times W \times C_{in} \times C_{out} \times k^2$

Maintaining Accuracy

- Train or re-train models with sparsity constraints
- This ensures model is trained properly using reduced number of weights

Advanced Group Convolution Algorithms

- ShuffleNets – Group Convolution followed by shuffling of connections
- CondenseNets – Connections between channels part of training
- Optimal scheduling of groups on GPUs
  - Pytorch scheduling is vary naïve – may lead to sub-optimal GPU resource utilization for models such as CondenseNets
  - My PhD student has a knapsack+ILP based algorithm for optimal scheduling – if you are interested, I can share the paper with you

Quantization

- Reduce the size of data type used to represent weights of the kernels
- Instead of using 64 bit floating point, use 16 bits or 8 bits
- Model size reduced by 4 to 8 times
- Number of operations remain the same, but current GPUs have the capabilities to process them 4 to 8 times faster
- We will look at one such technique in Transformer Acceleration

Next Class

- 10/2 Lecture 12
  - GPU Programming using AMD HIP

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Thank You

- Questions?

- Email: sanmukh.kuppannagari@case.edu