Programming Massively Parallel Processors

Chapter16 Deep learning

242AIG18 JiYeong Yi

Chapter16

Deep learning

Table of contents

15+ Results

- Graph dataset

Vertex Size	1000	4000	8000
Edge Size	9371	30380	67700

- GPU BFS traversal kernel
  1. Vertex-centric push
  2. Vertex-centric pull
  3. Edge-centric
  4. Frontier (Vertex-centric push)
  5. Frontier with privatization

Results

NVIDIA GeForce RTX 3080	
SM Count	68
Max resident threads per SM	1536
Max number of resident blocks per SM	16
Threads in warp	32
Max threads per block	1024
Max thread dimensions	(1024, 1024, 64)
Max grid dimensions	$(2^{31}-1, 2^{16}-1, 2^{16}-1)$
Shared Mem per SM	48 KB
Registers per SM	64 KB
Total constant Mem	64 KB
Total global Mem	10 GB

- Maximum total number of threads =
  # SM x Max resident threads per SM =
  68 X 1536 = 104448

15+ Results

Vertex Size	Edge Size
8000	67700

Level	0	1	2	3	4	5	6
Frontier size	1	13	87	792	4300	2788	19

<table>
<tr><th colspan="2"></th><th>GPU
(Vertex-Centric Push)
Execution Time</th><th>GPU
(Vertex-Centric Pull)
Execution Time</th><th>GPU
(Edge-Centric)
Execution Time</th><th>GPU
(Frontier)
Execution Time</th><th>GPU
(Privatization)
Execution Time</th></tr>
<tr><td rowspan="3">Level 0</td><td>Block size</td><td colspan="5">(256, 1, 1)</td></tr>
<tr><td>Grid size</td><td colspan="2">ceil(8000/256) = 32
(32, 1, 1)</td><td>ceil(67700 / 256) = 265
(265, 1, 1)</td><td colspan="2">ceil(1/256) = 1
(1, 1, 1)</td></tr>
<tr><td>Total # of threads</td><td colspan="2">8192</td><td>67840</td><td colspan="2">256</td></tr>
<tr><td rowspan="3">Level 1</td><td>Block size</td><td colspan="5">(256, 1, 1)</td></tr>
<tr><td>Grid size</td><td colspan="2">ceil(8000/256) = 32
(32, 1, 1)</td><td>ceil(67700 / 256) = 265
(265, 1, 1)</td><td colspan="2">ceil(13/256) = 1
(1, 1, 1)</td></tr>
<tr><td>Total # of threads</td><td colspan="2">8192</td><td>67840</td><td colspan="2">256</td></tr>
</table>

15+ Results

Vertex Size	Edge Size
8000	67700

Level	0	1	2	3	4	5	6
Frontier size	1	13	87	792	4300	2788	19

		GPU (Vertex-Centric Push) Execution Time	GPU (Vertex-Centric Pull) Execution Time	GPU (Edge-Centric) Execution Time	GPU (Frontier) Execution Time	GPU (Privatization) Execution Time
Level 2	Block size	(256, 1, 1)				
	Grid size	ceil(8000/256) = 32 (32, 1, 1)		ceil(67700 / 256) = 265 (265, 1, 1)	ceil(87/256) = 1 (1, 1, 1)	
	Total # of threads	8192		67840	256	
Level 3	Block size	(256, 1, 1)				
	Grid size	ceil(8000/256) = 32 (32, 1, 1)		ceil(67700 / 256) = 265 (265, 1, 1)	ceil(792/256) = 4 (4, 1, 1)	
	Total # of threads	8192		67840	1024	

15+ Results

Vertex Size	Edge Size
8000	67700

Level	0	1	2	3	4	5	6
Frontier size	1	13	87	792	4300	2788	19

		GPU (Vertex-Centric Push) Execution Time	GPU (Vertex-Centric Pull) Execution Time	GPU (Edge-Centric) Execution Time	GPU (Frontier) Execution Time	GPU (Privatization) Execution Time
Level 4	Block size	(256, 1, 1)				
	Grid size	ceil(8000/256) = 32 (32, 1, 1)		ceil(67700 / 256) = 265 (265, 1, 1)	ceil(4300/256) = 17 (17, 1, 1)	
	Total # of threads	8192		67840	4352	
Level 5	Block size	(256, 1, 1)				
	Grid size	ceil(8000/256) = 32 (32, 1, 1)		ceil(67700 / 256) = 265 (265, 1, 1)	ceil(2788/256) = 11 (11, 1, 1)	
	Total # of threads	8192		67840	2816	

15+ Results

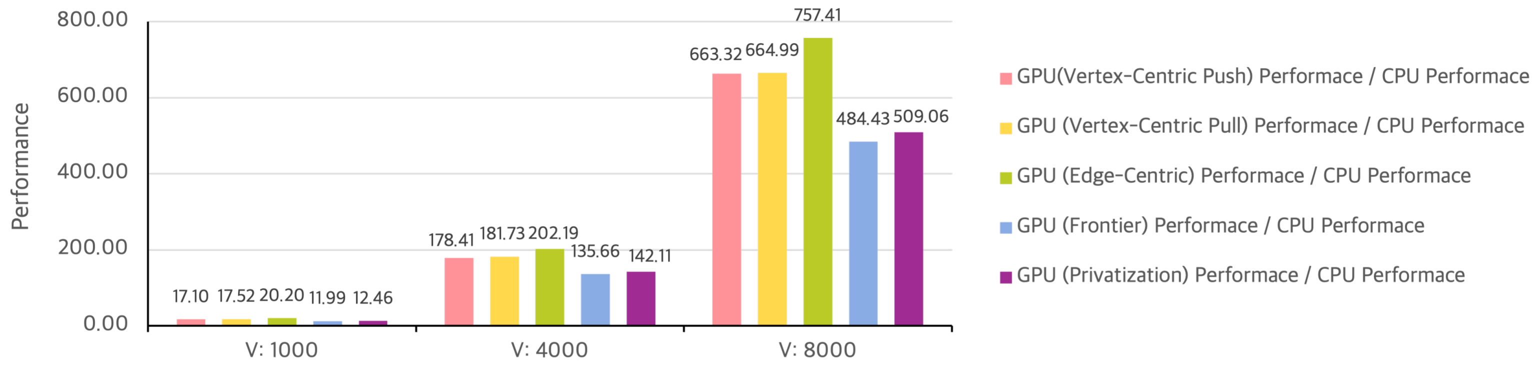
Vertex Size	Edge Size
8000	67700

Level	0	1	2	3	4	5	6
Frontier size	1	13	87	792	4300	2788	19

<table>
<tr><th colspan="2"></th><th>GPU
(Vertex-Centric Push)
Execution Time</th><th>GPU
(Vertex-Centric Pull)
Execution Time</th><th>GPU
(Edge-Centric)
Execution Time</th><th>GPU
(Frontier)
Execution Time</th><th>GPU
(Privatization)
Execution Time</th></tr>
<tr><td rowspan="3">Level 6</td><td>Block size</td><td colspan="5">(256, 1, 1)</td></tr>
<tr><td>Grid size</td><td colspan="2">ceil(8000/256) = 32
(32, 1, 1)</td><td>ceil(67700 / 256) = 265
(265, 1, 1)</td><td colspan="2">ceil(19/256) = 1
(1, 1, 1)</td></tr>
<tr><td>Total # of threads</td><td colspan="2">8192</td><td>67840</td><td colspan="2">256</td></tr>
</table>

15+ Results

Vertex Size	Edge Size	CPU Execution Time	GPU (Vertex-Centric Push) Execution Time	GPU (Vertex-Centric Pull) Execution Time	GPU (Edge-Centric) Execution Time	GPU (Frontier) Execution Time	GPU (Privatization) Execution Time
1000	9371	771.560 (us)	45.108 (us)	44.029 (us)	38.204 (us)	64.359 (us)	61.903 (us)
4000	30380	9.836 (ms)	55.131 (us)	54.125 (us)	48.647 (us)	72.506 (us)	69.213 (us)
8000	67700	38.363 (ms)	57.834 (us)	57.690 (us)	50.650 (us)	79.192 (us)	75.360 (us)



16.1 Background

- Machine learning is a field of computer science that studies methods for learning application logic from data rather than designing explicit algorithms.

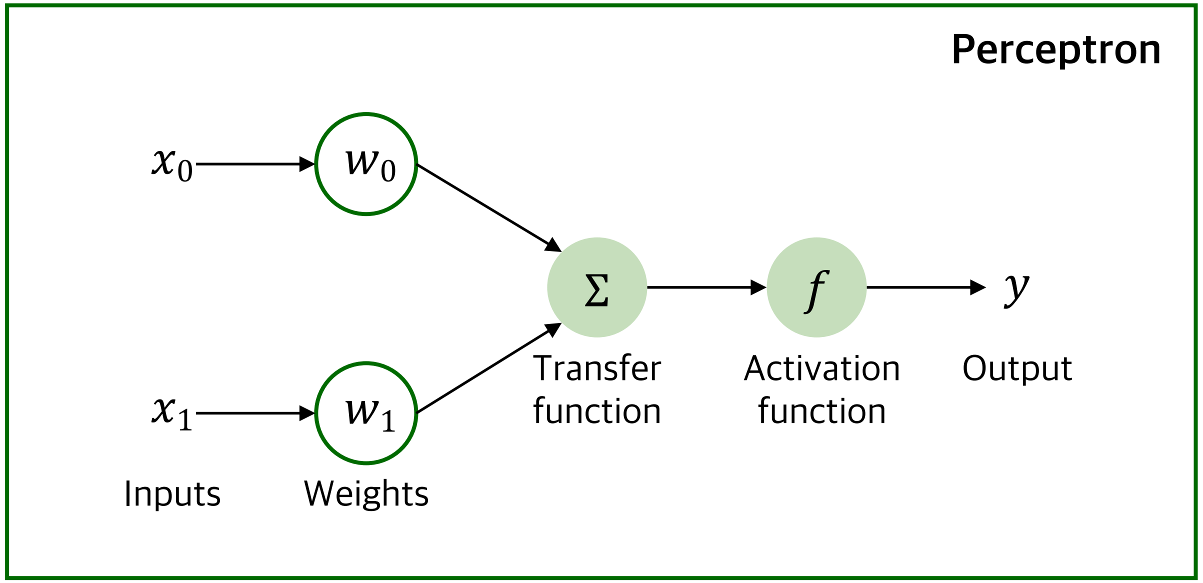
- There is a wide range of machine learning tasks.
  1) Classification
  2) Regression
  3) Transcription
  4) Translation
  5) Embedding
  6) …

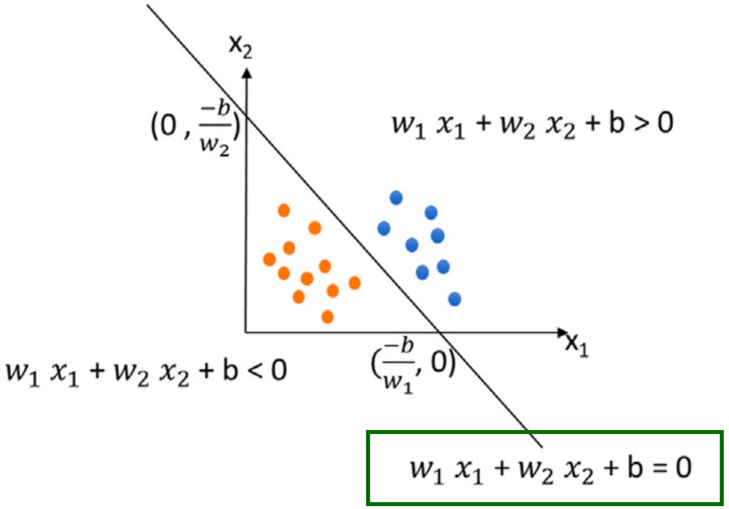
- Classification is to determine which of the k categories the input belongs to.
  An example is object recognition, such as determining which type of food is shown in a photo.

16.1 Background

- **Inference**
  The process of computing the class for an input is commonly referred to as inference for the classifier.

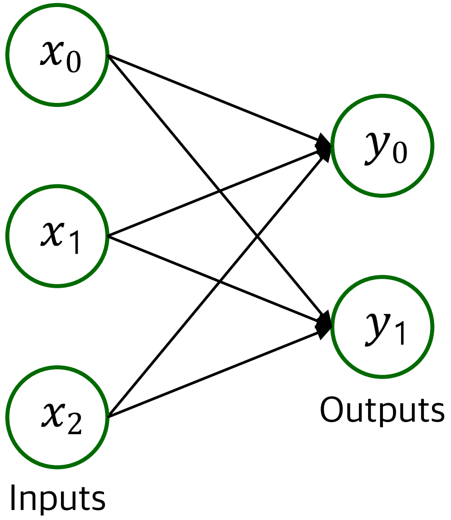
- **Training**
  The process of using data to determine the values of the model parameters θ, including the weights (w1, w2) and the bias b.





16.1 Background

- In general, in a **fully connected layer**, every one of the m outputs is a function of all the n inputs.
- All the weights of a fully connected layer form an m x n weight matrix W, where each of the m rows is the weight vector (of size n elements) to be applied to the input vector (of size n elements) to produce one of the m outputs.
- The process of evaluating all the outputs from the inputs of a fully connected layer is a **matrix-vector multiplication**.



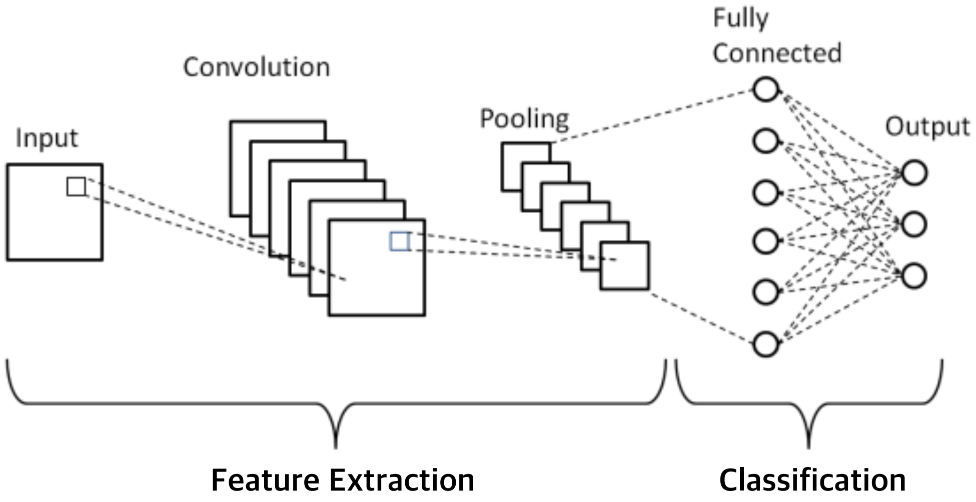
$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

16.2 Convolutional neural networks (CNN)

- In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyze visual imagery.

- The architecture of CNN is designed to extract meaningful features from complex visual data. This is achieved by using specialized layers within the network architecture, consisting of three basic layer types.

1. Convolutional layer
2. Pooling layer (subsampling layer)
3. Fully connected layer

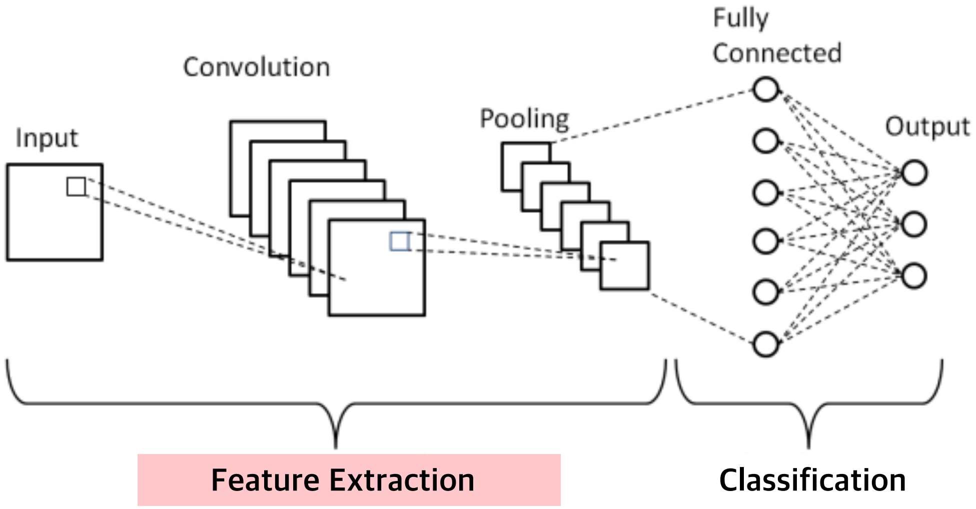


16.2 Convolutional neural networks (CNN)

- In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyze visual imagery.

- The architecture of CNN is designed to extract meaningful features from complex visual data. This is achieved by using specialized layers within the network architecture, consisting of three basic layer types.

1. Convolutional layer
2. Pooling layer (subsampling layer)
3. Fully connected layer

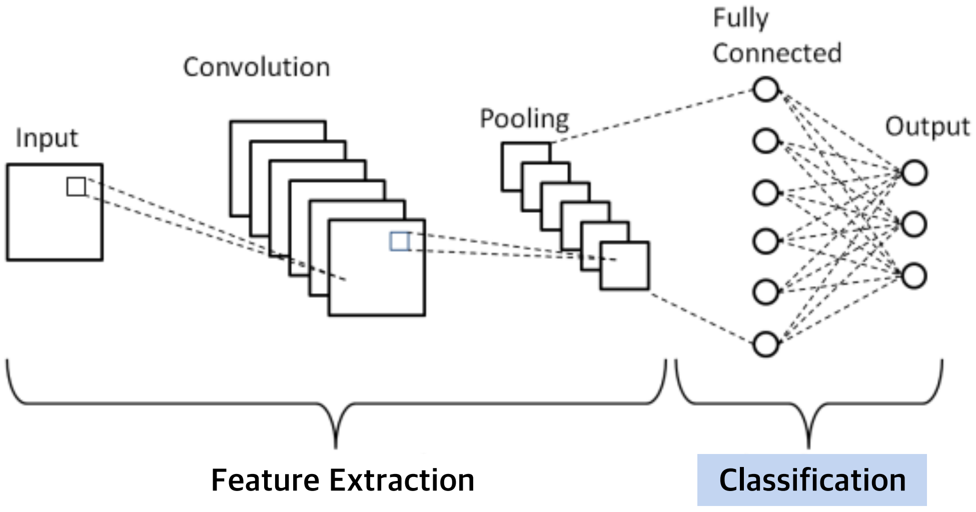


16.2 Convolutional neural networks (CNN)

- In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyze visual imagery.

- The architecture of CNN is designed to extract meaningful features from complex visual data. This is achieved by using specialized layers within the network architecture, consisting of three basic layer types.

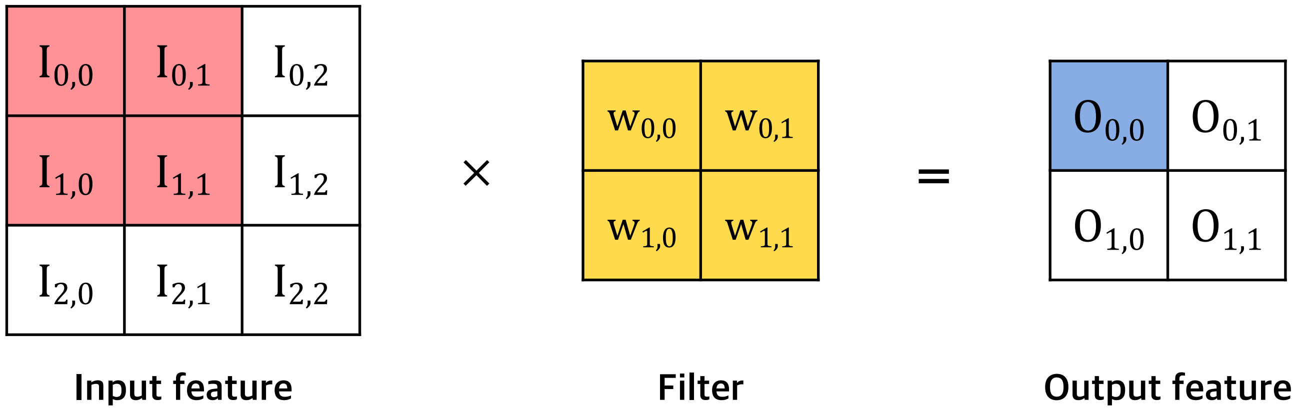
1. Convolutional layer
2. Pooling layer (subsampling layer)
3. Fully connected layer



16.2 Convolutional neural networks (CNN)

Convolutional layer

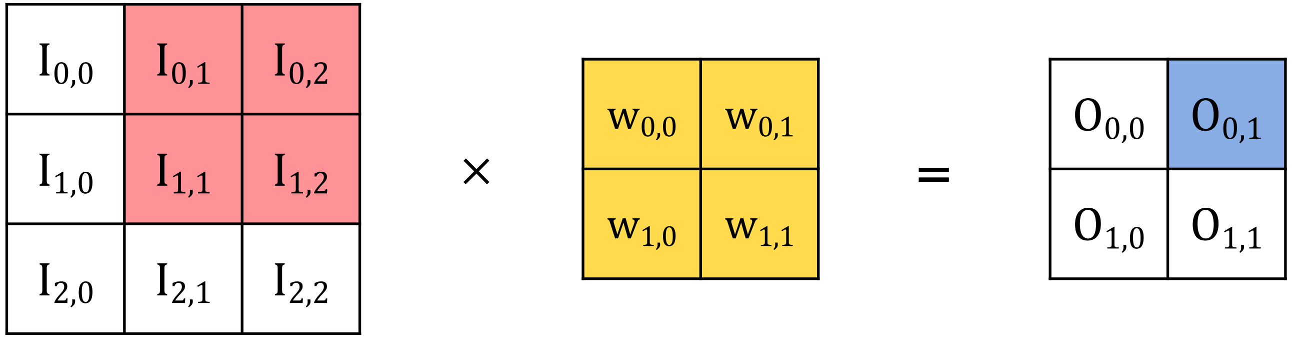
- The output feature maps to be produced for input feature maps consists of pixels, each of which is produced by performing a convolution between a small local patch of the feature map pixels produced by the previous layer and a set of weights called a filter bank.



16.2 Convolutional neural networks (CNN)

Convolutional layer

- The output feature maps to be produced for input feature maps consists of pixels, each of which is produced by performing a convolution between a small local patch of the feature map pixels produced by the previous layer and a set of weights called a filter bank.

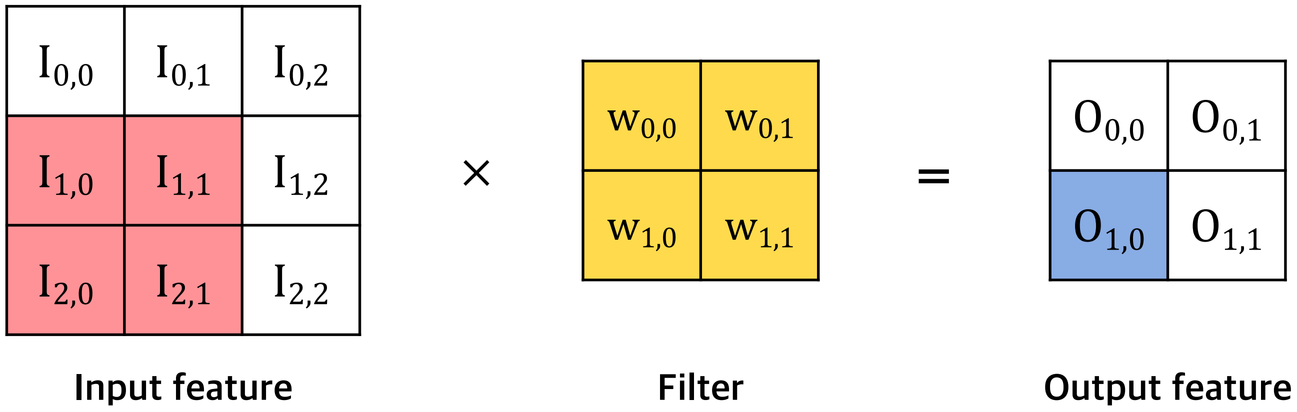


Input feature　　Filter　　Output feature

16.2 Convolutional neural networks (CNN)

Convolutional layer

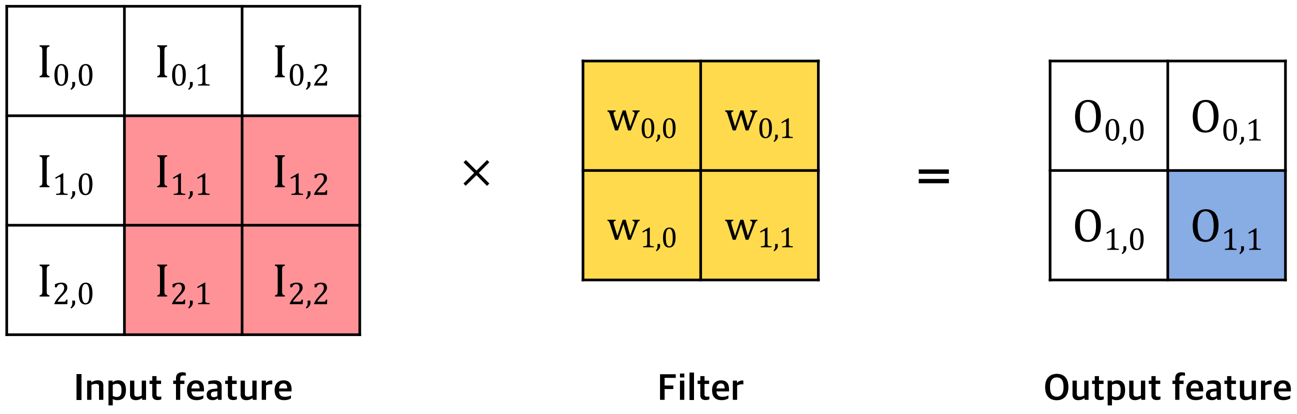
- The output feature maps to be produced for input feature maps consists of pixels, each of which is produced by performing a convolution between a small local patch of the feature map pixels produced by the previous layer and a set of weights called a filter bank.



16.2 Convolutional neural networks (CNN)

Convolutional layer

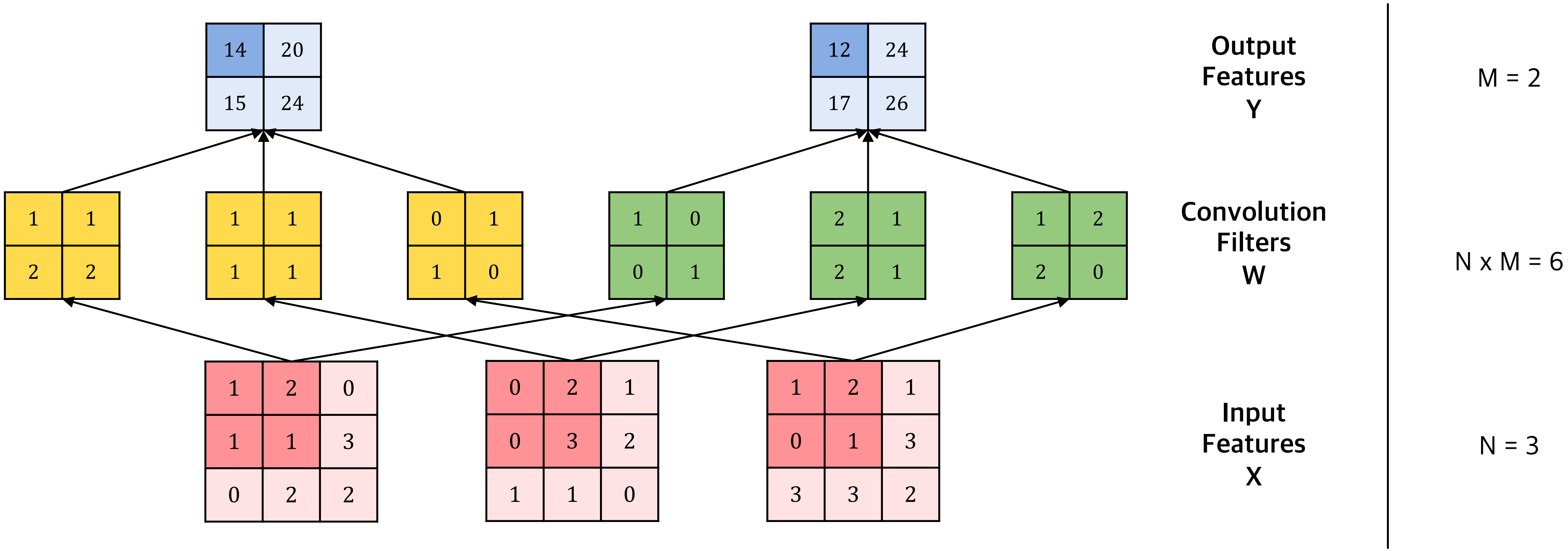
- The output feature maps to be produced for input feature maps consists of pixels, each of which is produced by performing a convolution between a small local patch of the feature map pixels produced by the previous layer and a set of weights called a filter bank.



16.2 Convolutional neural networks (CNN)

Convolutional layer

- In general, if a convolutional layer has n input feature maps and m output feature maps, n xm different 2D filter banks will be used.



16.2 Convolutional neural networks (CNN)

Pooling layer (subsampling layer)

- The output feature maps of convolution layer typically go through a pooling layer.
  A pooling layer <u>reduces the size of image maps</u> by combining pixels.

Ex) Max pooling : Selects the pixel with the maximum value to send to the output array.

1	2	1	4
2	3	3	5
6	7	4	5
2	4	6	2

→ **2x2 Max pooling with stride 2** →

3	3
7	4

16.3 Convolutional layer: a CUDA interface kernel

- The computation pattern in training a convolutional neural network is like matrix multiplication
  : It is both compute intensive and highly parallel.

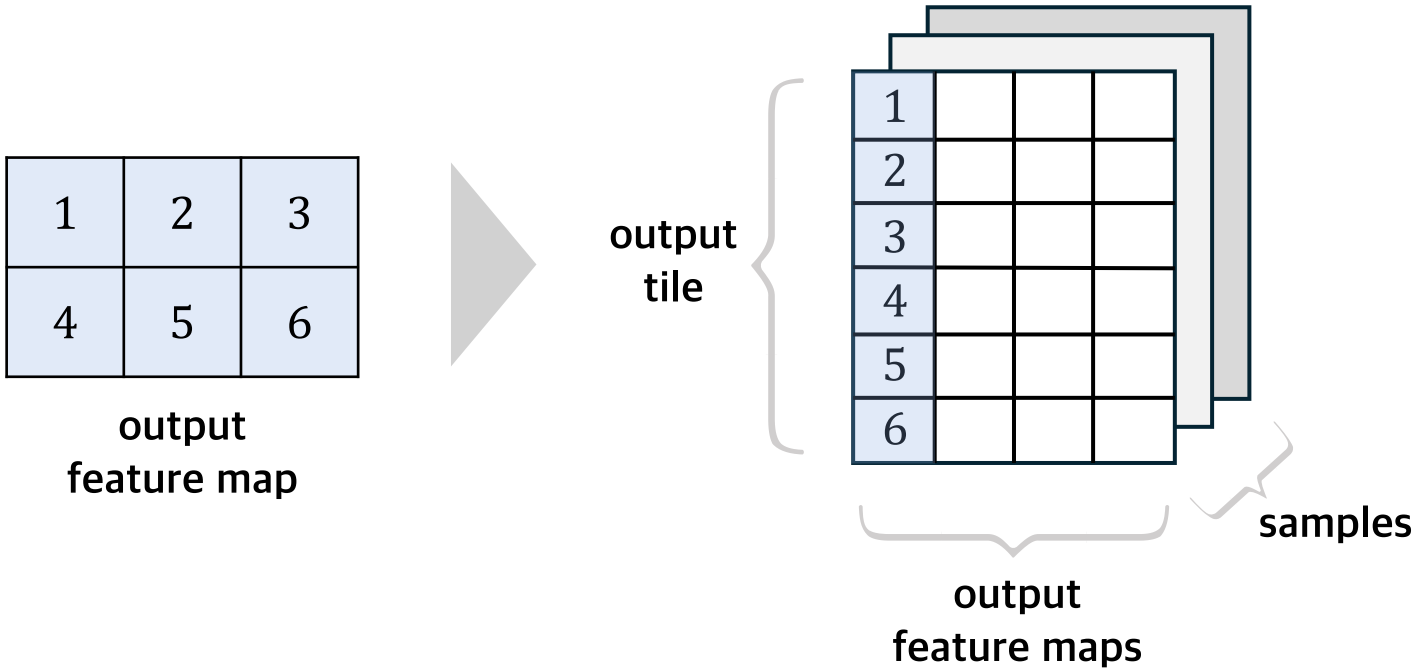
- Different samples in a minibatch, different output feature maps for the same sample, and different elements for each output feature map can be processed **in parallel**.

- Input feature maps and weights in a filter bank also offer a significant level of parallelism.
  However, to parallelize them, one would need to use **atomic operations** in accumulating into the output elements.

16.3 Convolutional layer: a CUDA interface kernel

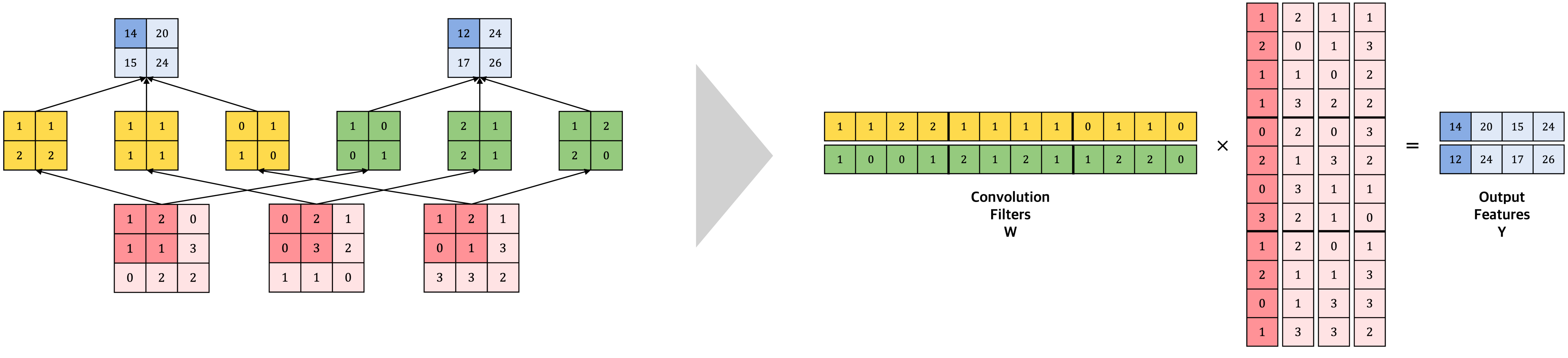
- 3-dimensional grid

  1. The first dimension (X) corresponds to the output features maps covered by each block.
  2. The second dimension (Y) reflects the location of a block's output tile inside the output feature map.
  3. The third dimension (Z) in the grid corresponds to samples in the minibatch.



16.4 Formulating a convolutional layer as GEMM

- A convolution layer can be built even faster by representing it as an equivalent matrix multiplication operation and then using a highly efficient **GEMM (general matrix multiply)** kernel from the CUDA linear algebra library cuBLAS.



16.4 Formulating a convolutional layer as GEMM

- The central idea is unfolding and duplicating input feature map pixels in such a way that all elements that are needed to compute one output feature map pixel will be stored as one sequential column of the matrix that is thus produced.

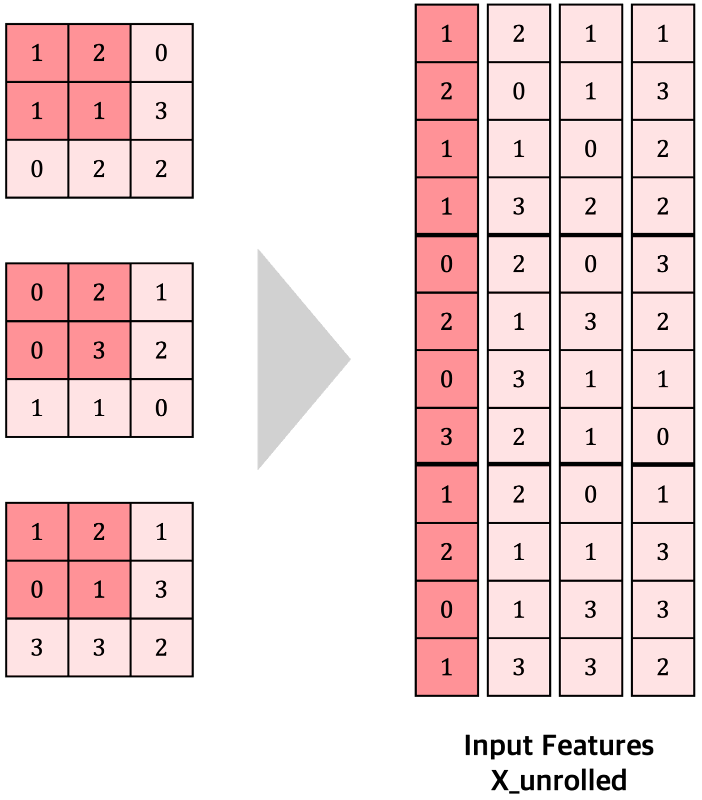
$$
\begin{bmatrix}
1 & 1 & 2 & 2 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 2 & 1 & 2 & 1 & 1 & 2 & 2 & 0
\end{bmatrix}
\times
\begin{bmatrix}
1 & 2 & 1 & 1 \\
2 & 0 & 1 & 3 \\
1 & 1 & 0 & 2 \\
1 & 3 & 2 & 2 \\
0 & 2 & 0 & 3 \\
2 & 1 & 3 & 2 \\
0 & 3 & 1 & 1 \\
3 & 2 & 1 & 0 \\
1 & 2 & 0 & 1 \\
2 & 1 & 1 & 3 \\
0 & 1 & 3 & 3 \\
1 & 3 & 3 & 2
\end{bmatrix}
=
\begin{bmatrix}
14 & 20 & 15 & 24 \\
12 & 24 & 17 & 26
\end{bmatrix}
$$

Convolution Filters
W

Input Features
X_unrolled

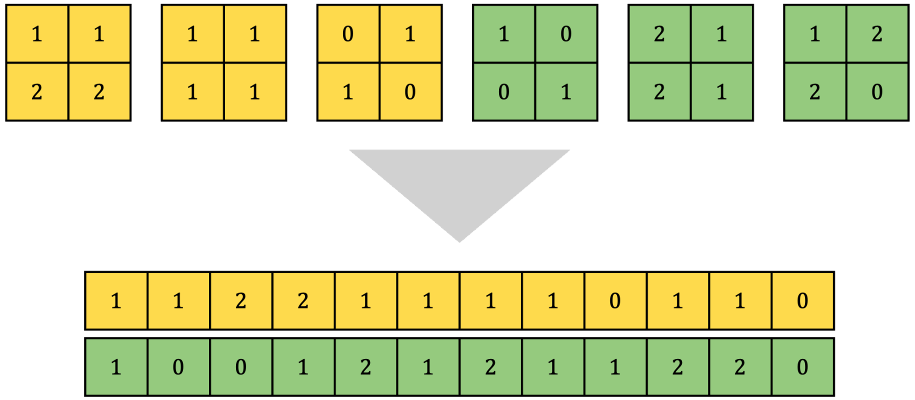
Output Features
Y

16.4 Formulating a convolutional layer as GEMM



- Since the results of the convolutions are summed across input features, the input features can be concatenated into one large matrix.
- Each input feature map becomes a section of rows in the large matrix.
- Each column of the resulting matrix contains all the input values necessary to compute one element of an output feature.

16.4 Formulating a convolutional layer as GEMM

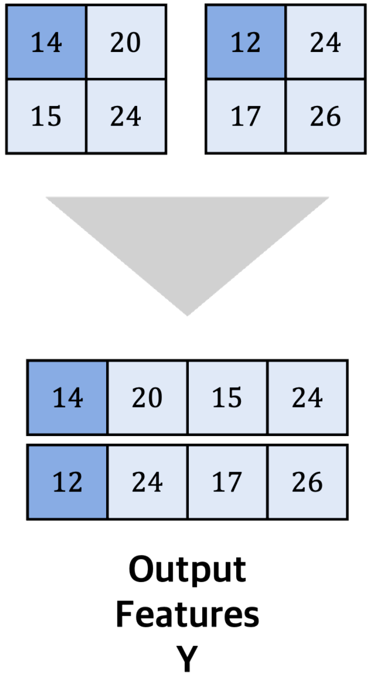


Convolution
Filters
W

- The filter banks are represented as a filter bank matrix in a fully linearized layout, in which each row contains all weight values that are needed to produce one output feature map.

- The height of the filter bank matrix is the number of output feature maps.

16.4 Formulating a convolutional layer as GEMM



- During matrix multiplication the row of the filter bank matrix and the column of the input feature matrix will produce one pixel of the output feature map.

16.4 Formulating a convolutional layer as GEMM

- Implementing convolutions with matrix multiplication can be very efficient, since matrix multiplication is highly optimized on all hardware platforms. Matrix multiplication is especially fast on GPUs because it has a high ratio of floating-point operations per byte of global memory data access.

- The **disadvantage** of forming the expanded input feature map matrix is that it involves duplicating the input data up to K*K (filter size) times, which can <u>require the allocation of a prohibitively large amount of memory</u>.

16.5 cuDNN library

- cuDNN is a library of optimized routines for implementing deep learning primitives.
  It provides highly tuned implementations for standard routines such as forward and backward convolution, attention, matmul, pooling, and normalization.

- cuDNN was designed to make it much easier for deep learning frameworks to take advantage of GPUs.
  It provides a flexible and easy-to-use C- language deep learning API that integrates neatly into existing deep learning frameworks (e.g., Caffe, Tensorflow, Theano, Torch).

16.5 cuDNN library

- cuDNN supports multiple algorithms for implementing a convolutional layer:
  matrix multiplication-based **GEMM** and Winograd, FFT-based, and so on.

- In the GEMM-based algorithm to implement the convolutions with a matrix multiplication, materializing the expanded input feature matrix in global memory can be costly in terms of both global memory space and bandwidth consumption.

- cuDNN avoids this problem by lazily generating and loading the expanded input feature map matrix X_unroll into on-chip memory only, rather than by gathering it in off-chip memory before calling a matrix multiplication routine.