CMSC 5743

Efficient Computing of Deep Neural Networks

Lecture 02: Convolution Speedup

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Latest update: September 22, 2021)

Fall 2021

These slides contain/adapt materials developed by

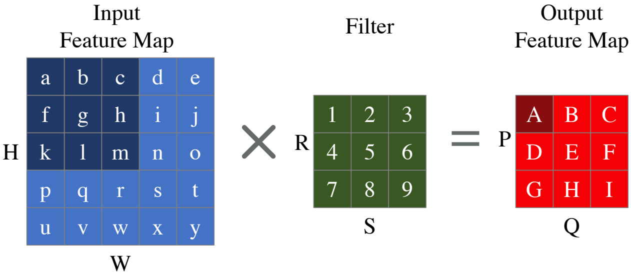
- Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*
- Asit K. Mishra et al. (2017). "Fine-grained accelerators for sparse machine learning workloads". In: *Proc. ASPDAC*, pp. 635–640
- Jongsoo Park et al. (2017). "Faster CNNs with direct sparse convolutions and guided pruning". In: *Proc. ICLR*
- UC Berkeley EE290: "Hardware for Machine Learning"
  https://inst.eecs.berkeley.edu/~ee290-2/sp20/

Overview

1. Convolution Basis
2. GEMM
3. Direct Convolution
4. Sparse Convolution

Convolution Basis

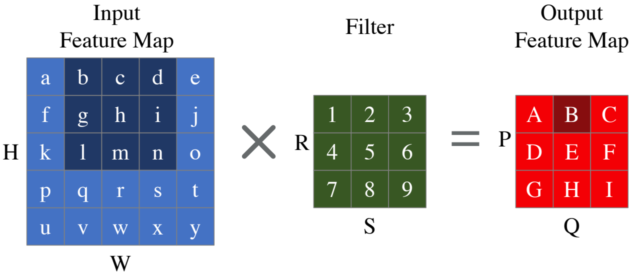
2D-Convolution



$$
\begin{aligned}
A = {} & a \cdot 1 + b \cdot 2 + c \cdot 3 \\
& + f \cdot 4 + g \cdot 5 + h \cdot 6 \\
& + k \cdot 7 + l \cdot 8 + m \cdot 9
\end{aligned}
$$

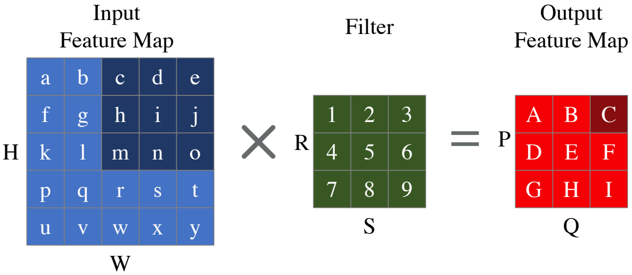
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map

2D-Convolution



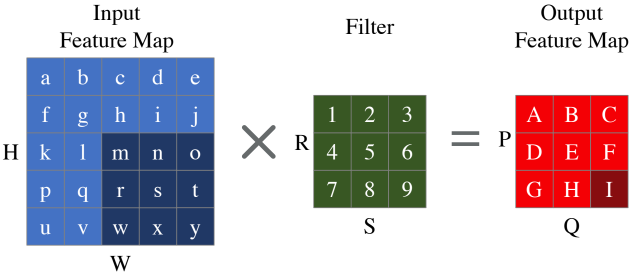
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step

2D-Convolution



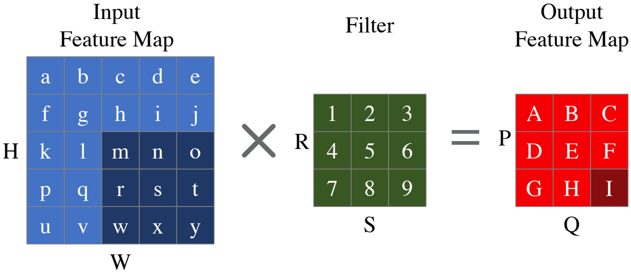
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step

2D-Convolution



- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step

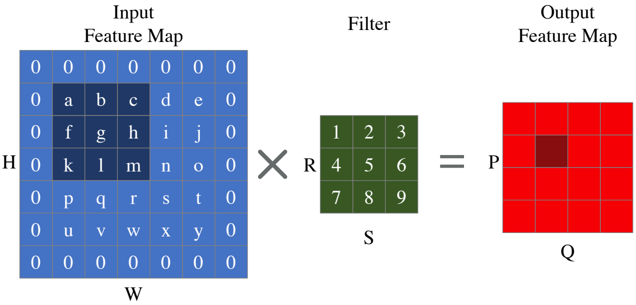
2D-Convolution



$$P = \frac{(H - R)}{\text{stride}} + 1;$$
$$Q = \frac{(W - S)}{\text{stride}} + 1.$$

- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step

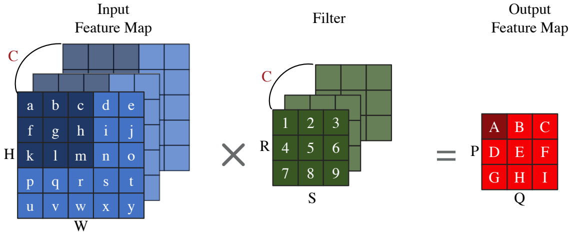
2D-Convolution



$$P = \frac{(H - R + 2 \cdot \text{pad})}{\text{stride}} + 1;$$
$$Q = \frac{(W - S + 2 \cdot \text{pad})}{\text{stride}} + 1.$$

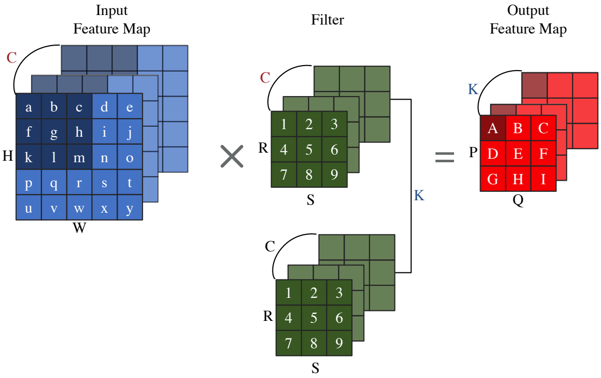
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step
- padding: # of zero rows/columns added

3D-Convolution



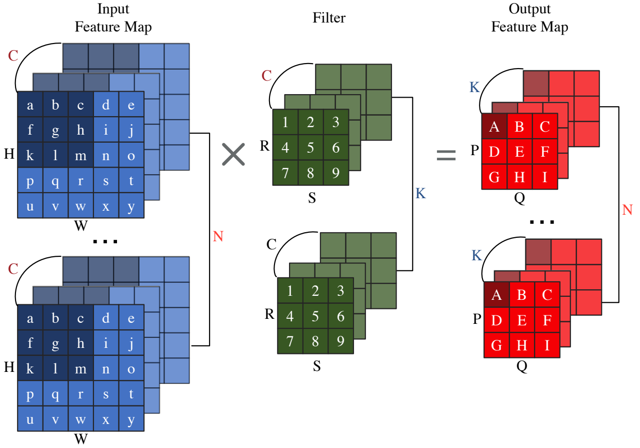
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step
- padding: # of zero rows/columns added
- C: # of input channels

3D-Convolution



- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step
- padding: # of zero rows/columns added
- C: # of input channels
- K: # of uutput channels

3D-Convolution



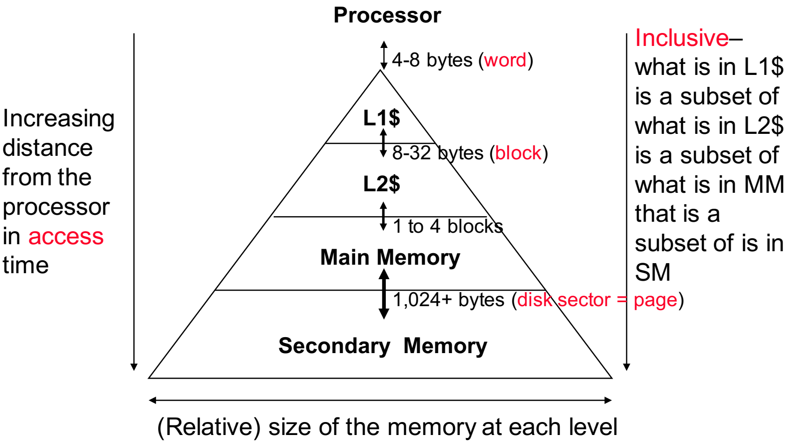
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step
- padding: # of zero rows/columns added
- C: # of input channels
- K: # of uutput channels
- N: Batch size

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 2 & 1 & 1 & 2 & 0 \\
0 & 2 & 0 & 1 & 1 & 0 & 0 \\
0 & 2 & 0 & 1 & 2 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\ast
\begin{bmatrix}
1 & 0 & 0 \\
1 & 1 & 1 \\
1 & 0 & -1
\end{bmatrix}
=
\begin{bmatrix}
4 & 6 & 3 & 5 & 4 \\
2 & 6 & 2 & 4 & 4 \\
1 & 5 & 3 & 4 & 4 \\
2 & 4 & 3 & 3 & 4 \\
0 & 2 & 2 & 4 & 3
\end{bmatrix}
$$

Direct convolution: No extra memory overhead

- Low performance
- Poor memory access pattern due to geometry-specific constraint
- Relatively short dot product

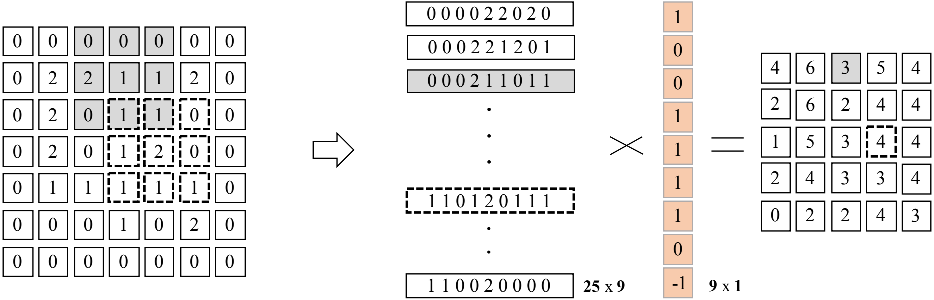
Background: Memory System



- Spatial locality
- Temporal Locality

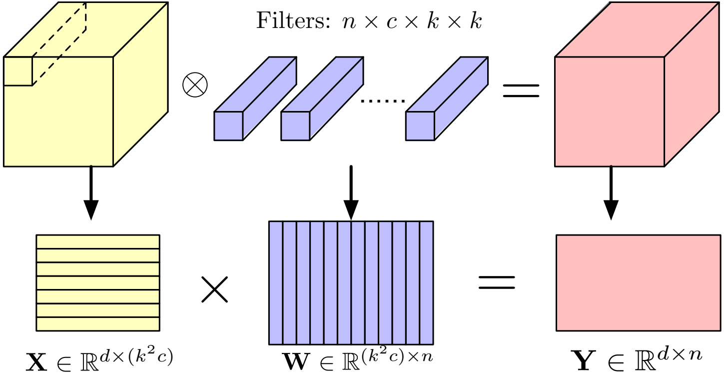
GEMM

Im2col (Image2Column) Convolution



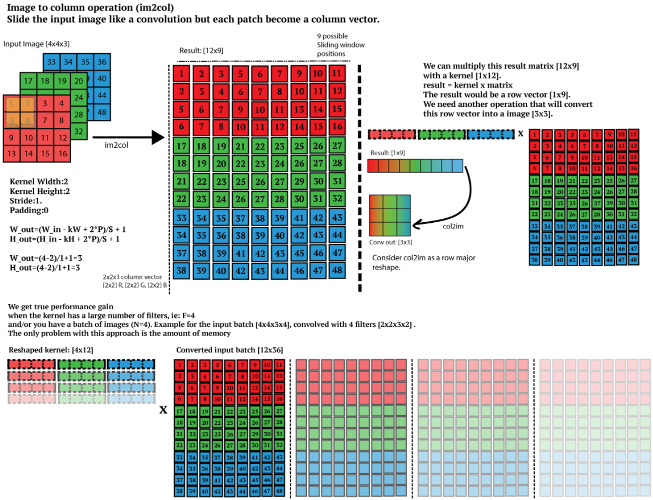
- Large extra memory overhead
- Good performance
- BLAS-friendly memory layout to enjoy SIMD/locality/parallelism
- Applicable for any convolution configuration on any platform

`Im2col` (Image2Column) Convolution



- Transform convolution to matrix multiplication
- Unified calculation for both convolution and fully-connected layers

Im2col (Image2Column): Another View

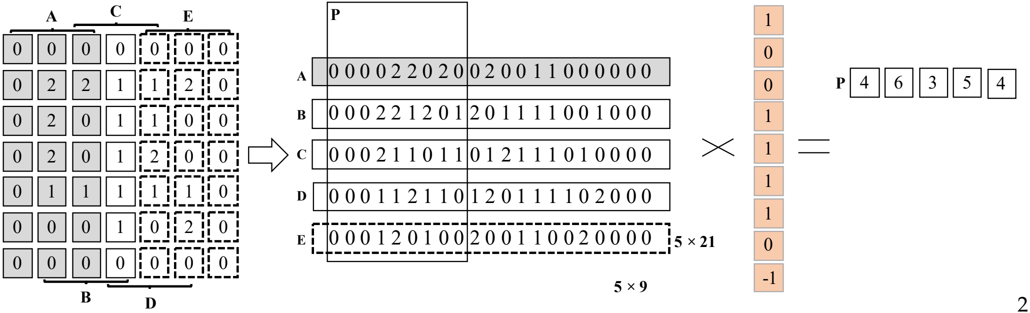


1

---

[1] https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/convolution_layer/making_faster

SOTA 1: Memory-efficient Convolution

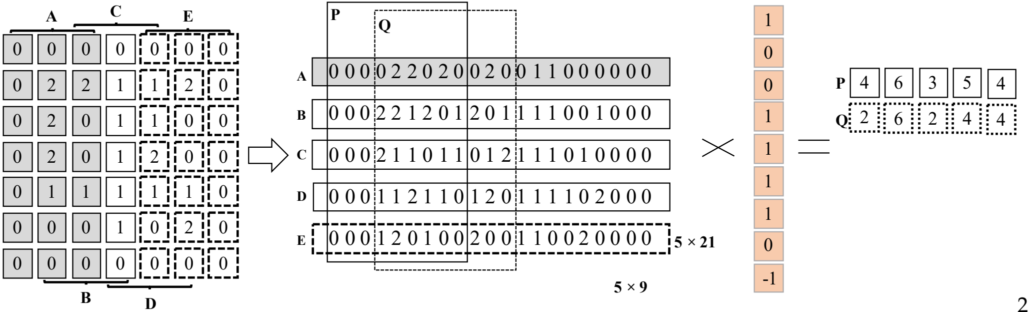


[2]

- Sub matrices in the lowered matrix will be "sgemm" ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2]Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.

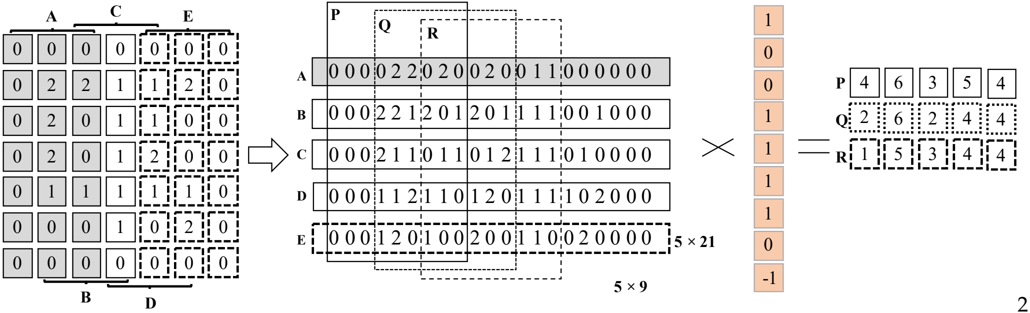
SOTA 1: Memory-efficient Convolution



2

- Sub matrices in the lowered matrix will be "sgemm" ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

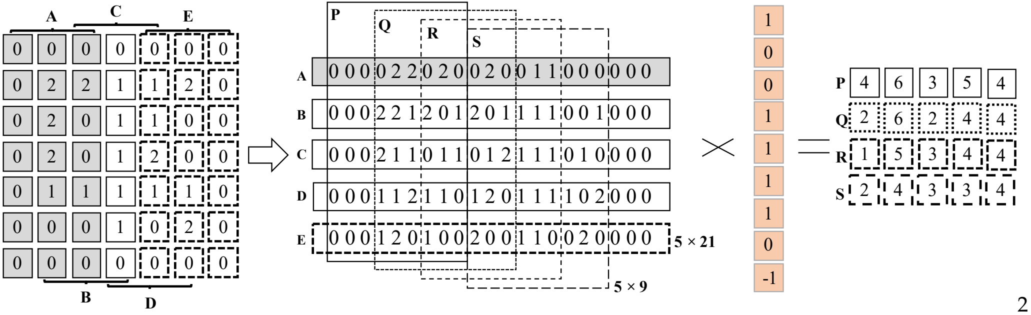
[2]Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.



[2]

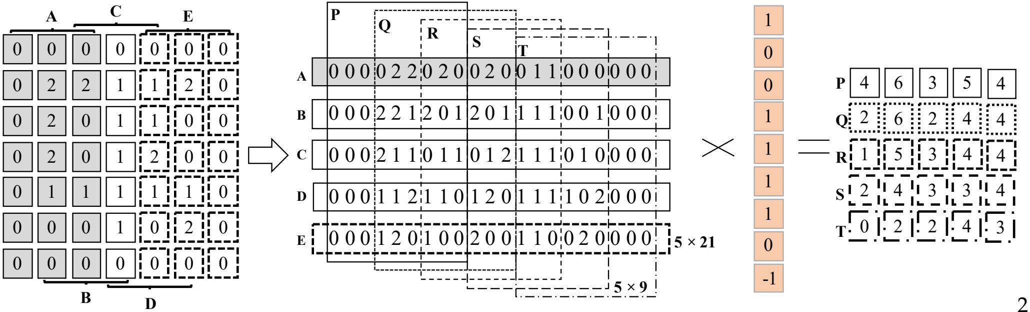
- Sub matrices in the lowered matrix will be "sgemm" ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2] Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.



- Sub matrices in the lowered matrix will be "sgemm" ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2] Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.



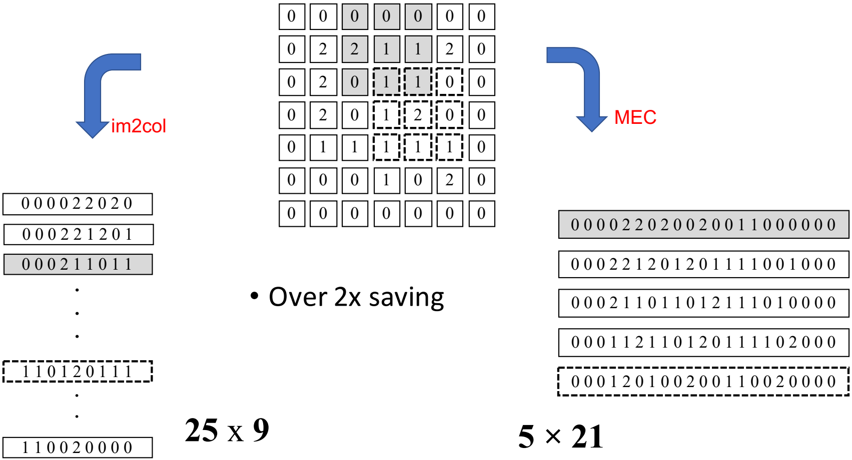
[2]

- Sub matrices in the lowered matrix will be “sgemm” ed in parallel
- Smaller memory foot print, cache locality, and explicit parallelism

[2]Minsik Cho and Daniel Brand (2017). “MEC: memory-efficient convolution for deep neural network”. In: *Proc. ICML*.

SOTA 1: Memory-efficient Convolution

Over 2$\times$ memory saving[3]:

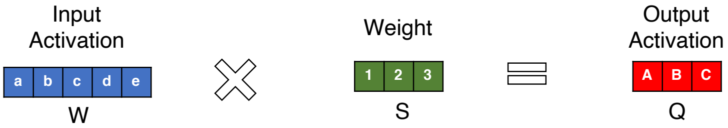


- Over 2x saving

[3] Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.

Direct Convolution

1D Convolution Example

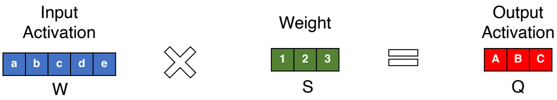



```
for(q=0; q<Q; q++){
  for (s=0; s<S; s++){
    OA[q] += IA[q+s] * W[s];
  }
}
```

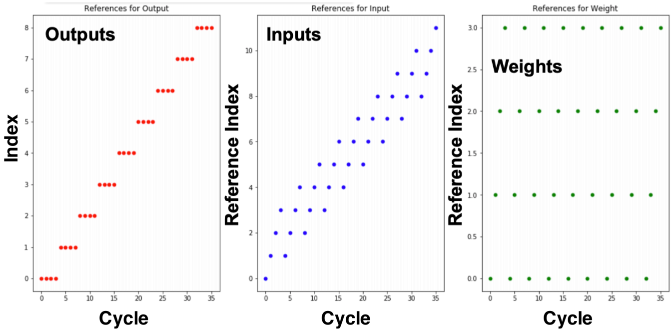
Output Stationary (OS) Dataflow

```
for (s=0; s<S; s++){
  for(q=0; q<Q; q++){
    OA[q] += IA[q+s] * W[s];
  }
}
```

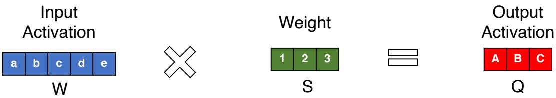
Weight Stationary (WS) Dataflow



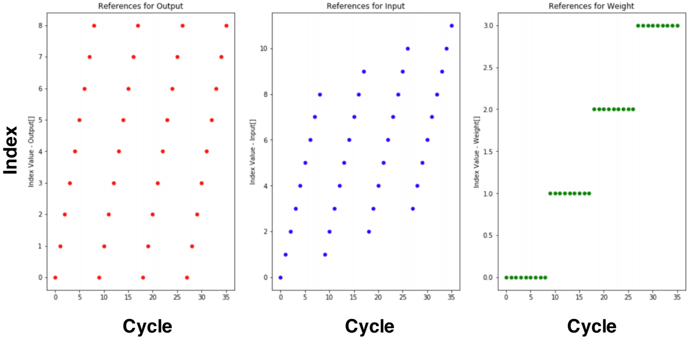

```
for(q=0; q<Q; q++){ // Q =9
  for (s=0; s<S; s++){ // S=4
    OA[q] += IA[q+s] * W[s];
  }
}
```



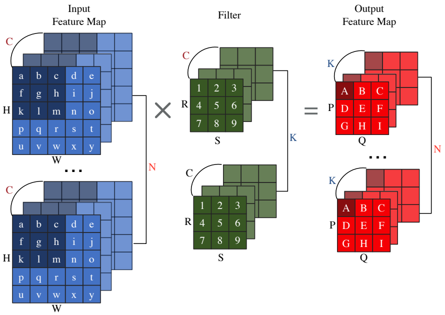
Buffer Access Pattern 2: Weight Stationary




```
for (s=0; s<S; s++){// S=4
  for(q=0; q<Q; q++){// Q =9
    OA[q] += IA[q+s] * W[s];
  }
```

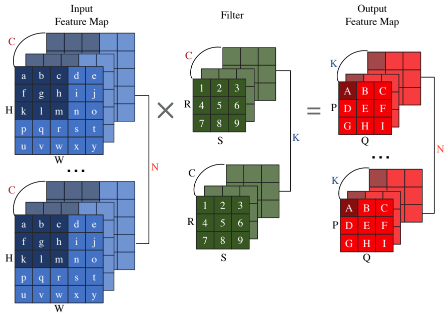


Direct Convolution



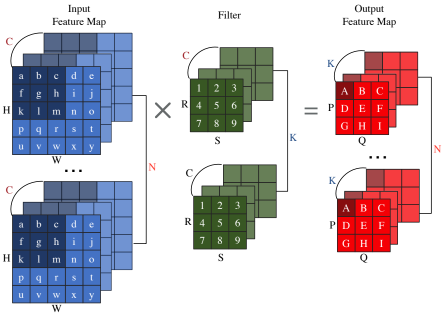

```
for (n=0; n<N; n++) {
for (k=0; k<K; k++) {
for (p=0; p<P; p++) {
for (q=0; q<Q; q++) {
    OA[n][k][p][q]= 0;
    for (r=0; r<R; r++) {
    for (s=0; s<S; s++) {
    for (c=0; c<C; c++) {
        h = p * stride - pad + r;
        w = q * stride - pad + s;
        OA[n][k][p][q] += IA[n][c][h][w] * W[k][c][r][s];
} } } } } } }
```

Direct Convolution: Loop Ordering



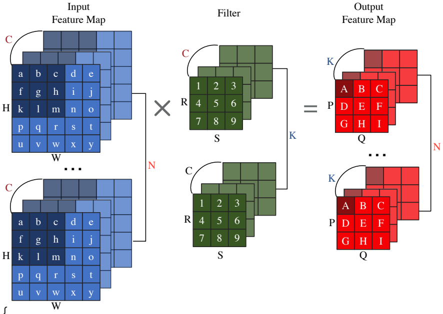

```
for (n=0; n<N; n++) {
for (r=0; r<R; r++) {
for (s=0; s<S; s++) {
for (c=0; c<C; c++) {
for (k=0; k<K; k++) {
    float curr_w = W[r][s][c][k];
    for (p=0; p<P; p++) {
    for (q=0; q<Q; q++) {
        h = p * stride - pad + r;
        w = q * stride - pad + s;
        OA[n][k][p][q] += IA[n][c][h][w] * curr_w;
} } } } } } }
```

Direct Convolution: Loop Ordering + Unrolling




```
for (n=0; n<N; n++) {
for (r=0; r<R; r++) {
for (s=0; s<S; s++) {
    spatial_for (c=0; c<C; c++) {
    spatial_for (k=0; k<K; k++) {
    float curr_w = W[r][s][c][k];
    for (p=0; p<P; p++) {
    for (q=0; q<Q; q++) {
        h = p * stride - pad + r;
        w = q * stride - pad + s;
        OA[n][k][p][q] += IA[n][c][h][w] * curr_w;
} } } } } } }
```

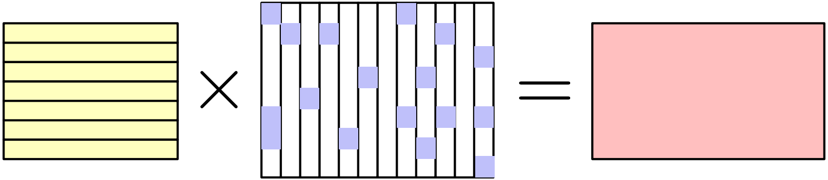
Direct Convolution: Loop Ordering + Unrolling + Tiling




```
for (n=0; n<N; n++) {
for (r=0; r<R; r++) {
for (s=0; s<S; s++) {
for (c_t=0; c_t<C/16; c_t++) {
for (k_t=0; k_t<K/64; k_t++) {
spatial_for (c_s=0; c_s<16; c_s++) {
spatial_for (k_s=0; k_s<64; k_s++) {
    int curr_c = c_t * 16 + c_s;
    int curr_k = k_t * 64 + k_s;
    float curr_w = W[r][s][curr_c][curr_k];
    for (p=0; p<P; p++) for (q=0; q<Q; q++) {
        h = p * stride - pad + r; w = q * stride - pad + s;
        OA[n][curr_k][p][q] += IA[n][curr_c][h][w] * curr_w;
} } } } } }
```

Sparse Convolution

- Our DNN may be redundant, and sometimes the filters may be sparse
- Sparsity can be helpful to overcome over-fitting



Sparse Convolution: Naive Implementation 1

X

0	0	3	0
7	0	0	0
0	0	4	8
6	5	3	0
2	0	0	1
0	0	0	8

*

w

0
0
4
8

Algorithm 1 Sparse Convlution Naive 1

---

for all w[i] **do**
    if w[i] = 0 **then**
        Continue;
    end if
    output feature map $\boldsymbol{Y} \leftarrow \boldsymbol{X} \times w$[i];
end for

Sparse Convolution: Naive Implementation 1

X

0	0	3	0
7	0	0	0
0	0	4	8
6	5	3	0
2	0	0	1
0	0	0	8

*

w

0
0
$\boxed{4}$
8

Algorithm 2 Sparse Convlution Naive 1

for all w[i] **do**
    if w[i] = 0 **then**
        Continue;
    end if
    output feature map $\boldsymbol{Y} \leftarrow \boldsymbol{X} \times \boldsymbol{w}$[i];
end for

BAD implementation for Pipeline!

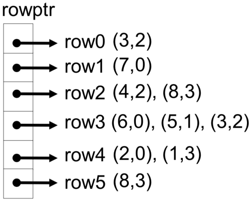
Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Sparse Matrix Representation

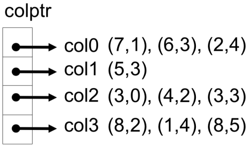
A

0	0	3	0
7	0	0	0
0	0	4	8
6	5	3	0
2	0	0	1
0	0	0	8

A matrix example

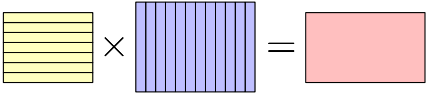


Compressed Sparse Row (CSR)



Compressed Sparse Column (CSC)

- CSR: Good for operation on feature maps
- CSC: Good for operation on filters
- We have better control on filters, thus usually CSC.



matrix * sparse vector

$$\overset{X}{\begin{bmatrix} 0 & 0 & 3 & 0 \\ 7 & 0 & 0 & 0 \\ 0 & 0 & 4 & 8 \\ 6 & 5 & 3 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 8 \end{bmatrix}} * \overset{w}{\begin{bmatrix} 0 \\ 0 \\ \boxed{4} \\ 8 \end{bmatrix}} = \overset{Y}{\begin{bmatrix} 12 \\ 0 \\ 16 \\ 12 \\ 0 \\ 0 \end{bmatrix}}$$

$$\begin{bmatrix} 0 & 0 & 3 & 0 \\ 7 & 0 & 0 & 0 \\ 0 & 0 & 4 & 8 \\ 6 & 5 & 3 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 8 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 4 \\ \boxed{8} \end{bmatrix} = \begin{bmatrix} 12 \\ 0 \\ 80 \\ 12 \\ 8 \\ 64 \end{bmatrix}$$

- BAD implementation for Spatial Locality!
- Poor memory access patterns

SOTA 2: Sparse Convolution

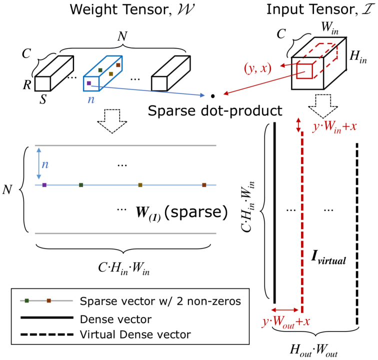


Figure 1: Conceptual view of the direct sparse convolution algorithm. Computation of output value at (y,x)th position of nth output channel is highlighted.

```
for each output channel n {
 for j in [W.rowptr[n], W.rowptr[n+1]) {
  off = W.colidx[j]; coeff = W.value[j]
  for (int y = 0; y < H_OUT; ++y) {
   for (int x = 0; x < W_OUT; ++x) {
    out[n][y][x] += coeff*in[off+f(0,y,x)]
   }
  }
 }
}
```

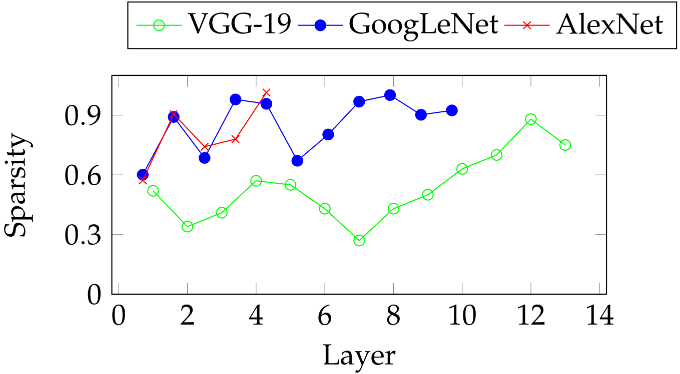
Figure 2: Sparse convolution pseudo code. Matrix **W** has *compressed sparse row* (CSR) format, where `rowptr[n]` points to the first non-zero weight of nth output channel. For the jth non-zero weight at (n,c,r,s), `W.colidx[j]` contains the offset to (c,r,s)th element of tensor `in`, which is pre-computed by layout function as $f(c,r,s)$. If `in` has CHW format, $f(c,r,s) = (cH_{in}+r)W_{in}+s$. The "virtual" dense matrix is formed on-the-fly by shifting `in` by $(0,y,x)$.

4

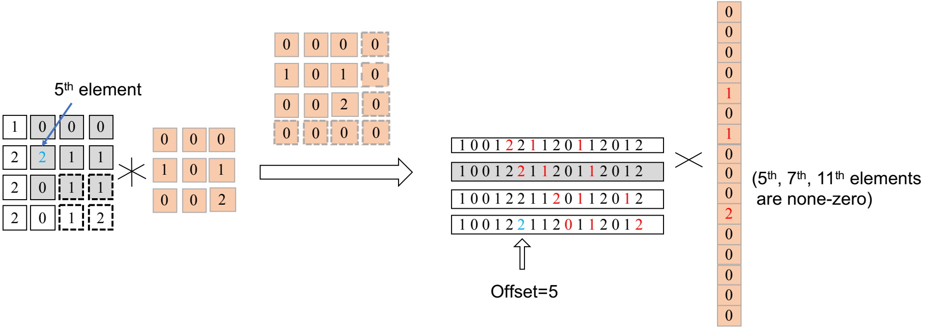
[4] Jongsoo Park et al. (2017). "Faster CNNs with direct sparse convolutions and guided pruning". In: *Proc. ICLR*.

Discussion: Sparse-Sparse Convolution

- Sparsity is a desired property for computation acceleration. (`cuSPARSE` library, direct sparse convolution, etc.)
- Sometimes not only the filters but also the input feature maps are sparse.



Discussion: Sparse-Sparse Convolution



- Efficient programming implementation required; (Improve pipeline efficiency)
- When sparsity$(input) = 0.9$, sparsity$(weight) = 0.8$, more than $10\times$ speedup;
- Some other issues:
  - How to be compatible with pooling layer?
  - Transform between dense & sparse formats