CMSC 5743

Efficient Computing of Deep Neural Networks

Implementation 01: GEMM-1

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

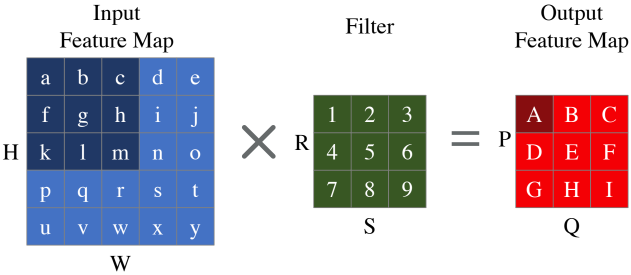
(Latest update: February 7, 2023)

2023 Spring

Overview

Convolution Basis

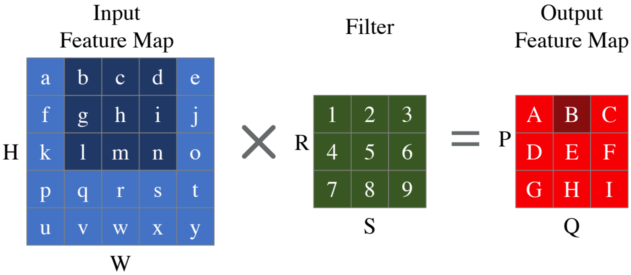
2D-Convolution



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

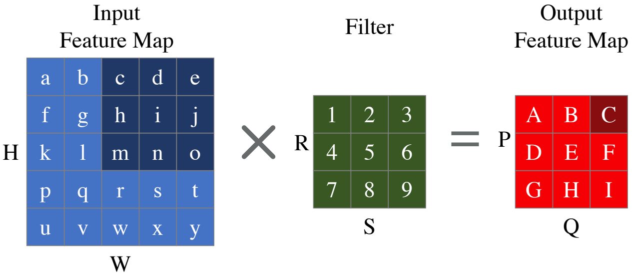
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map

2D-Convolution

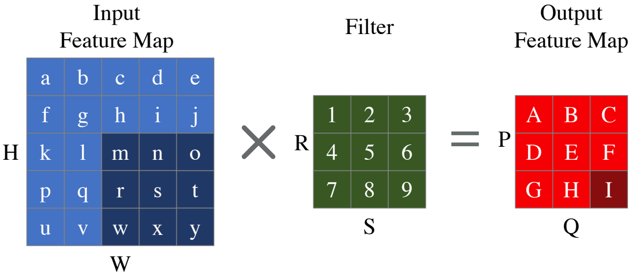


- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step

2D-Convolution

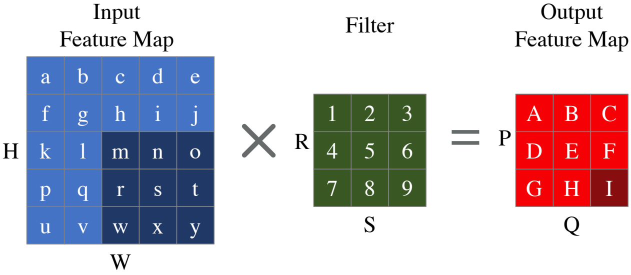


- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step



- 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

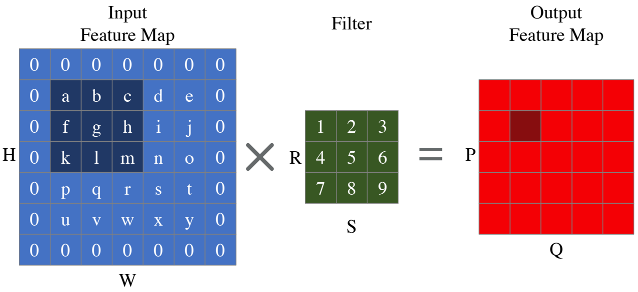
2D-Convolution



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

- 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

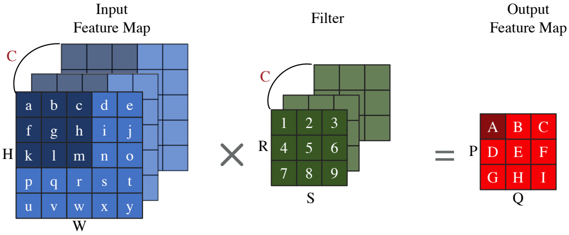


$$P = \frac{(H - R + 2 \cdot \text{pad})}{\text{stride}} + 1;$$

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

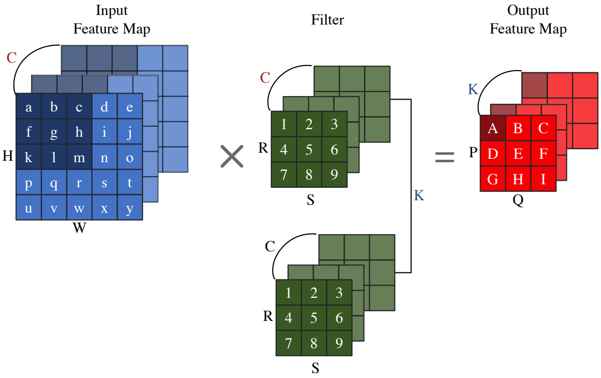
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step
- padding: # of zero rows/columns added

3D-Convolution



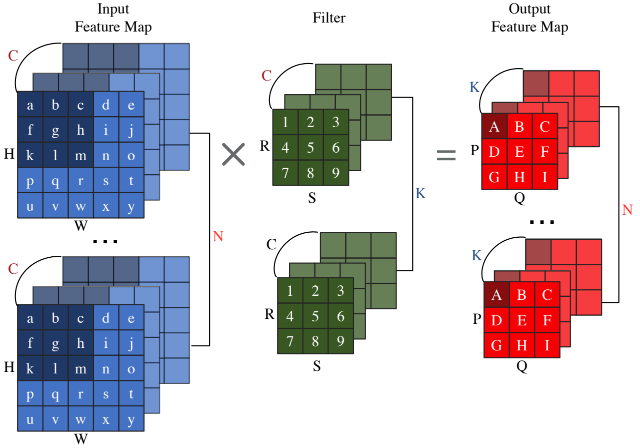
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step
- padding: # of zero rows/columns added
- C: # of input channels

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 output 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 output channels
- N: Batch size

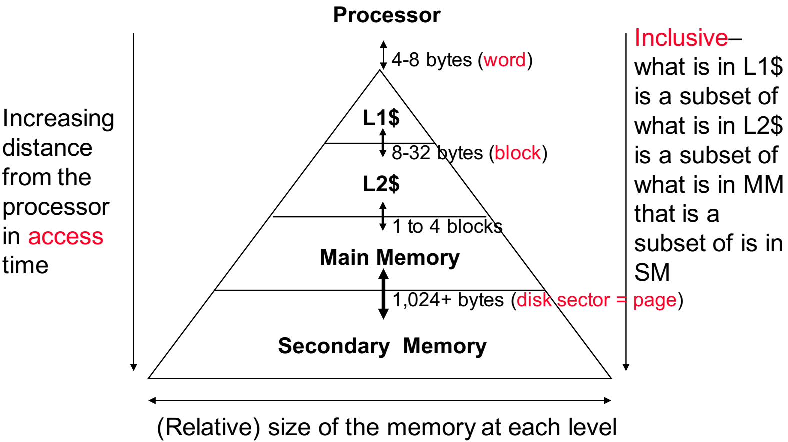
Convolution 101

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

Direct convolution: No extra memory overhead

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

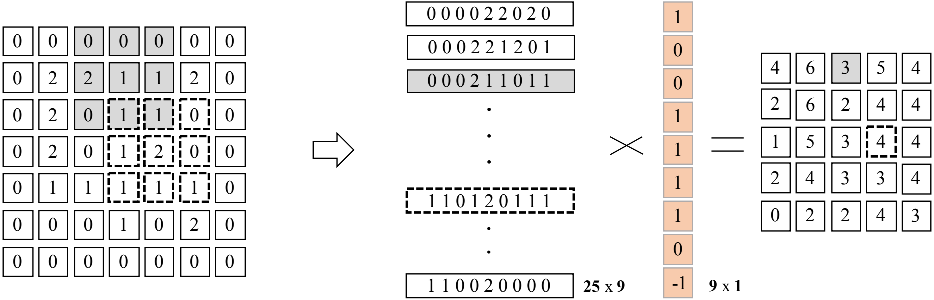
Background: Memory System



- Spatial locality
- Temporal Locality

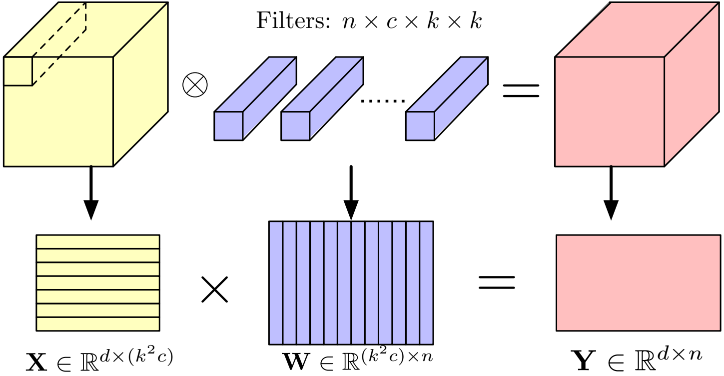
Im2Col

Im2col (Image2Column) Convolution



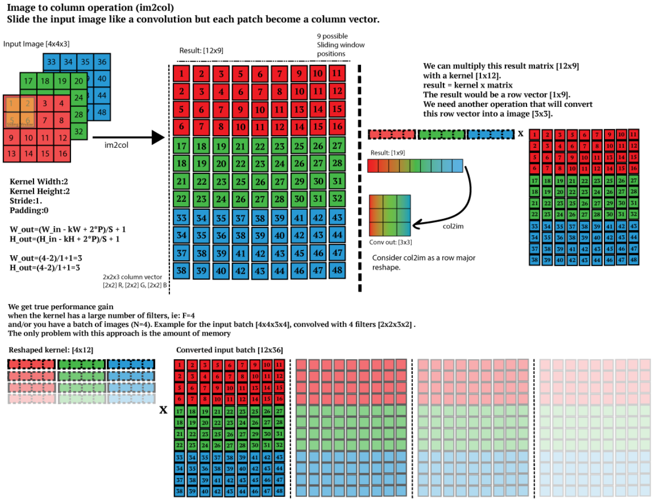
- Large extra memory overhead
- Good performance
- BLAS-friendly memory layout to enjoy SIMD/locality/parallelism
- Applicable for any convolution configuration on any platform

Im2col (Image2Column) Convolution



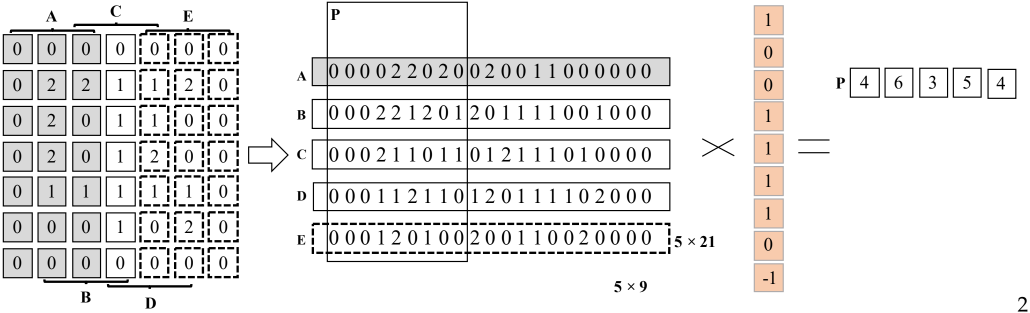
- Transform convolution to matrix multiplication
- Unified calculation for both convolution and fully-connected layers

Im2col (Image2Column): Another View



1

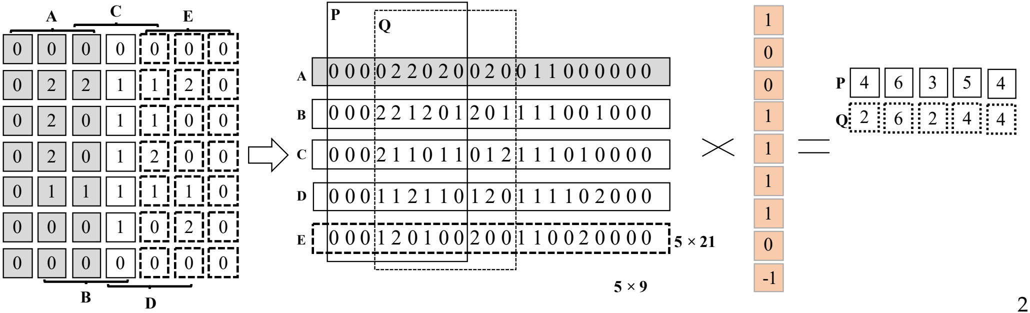
[1] https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/convolution_layer/making_faster



[2]

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

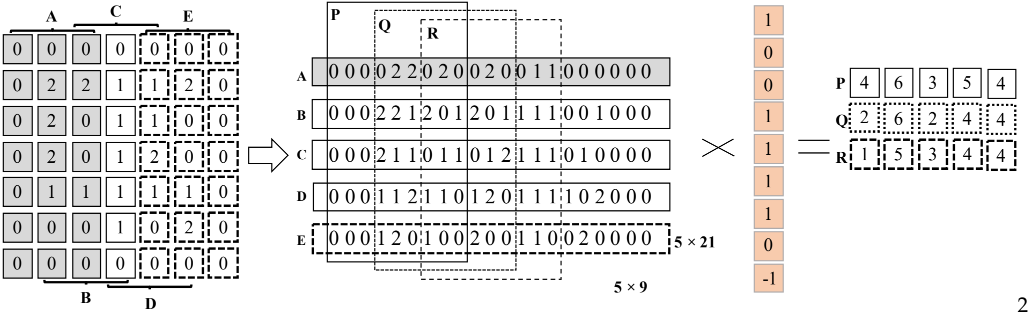
[2] Minsik Cho and Daniel Brand (2017). “MEC: memory-efficient convolution for deep neural network”. In: *Proc. ICML*.



[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*.

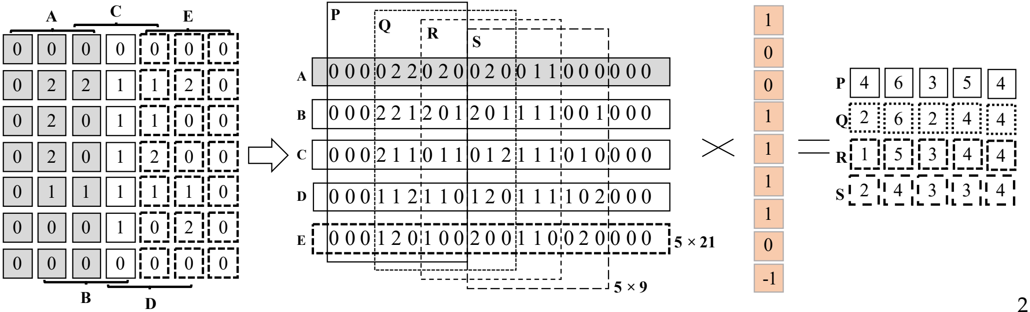


[2]

- 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

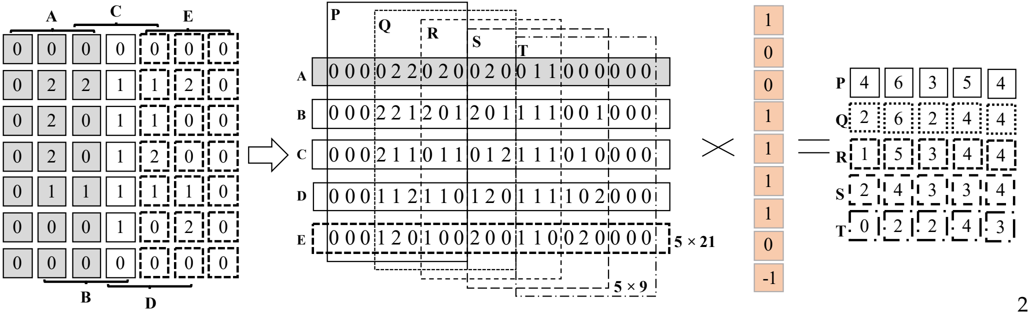


[2]

- 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*.

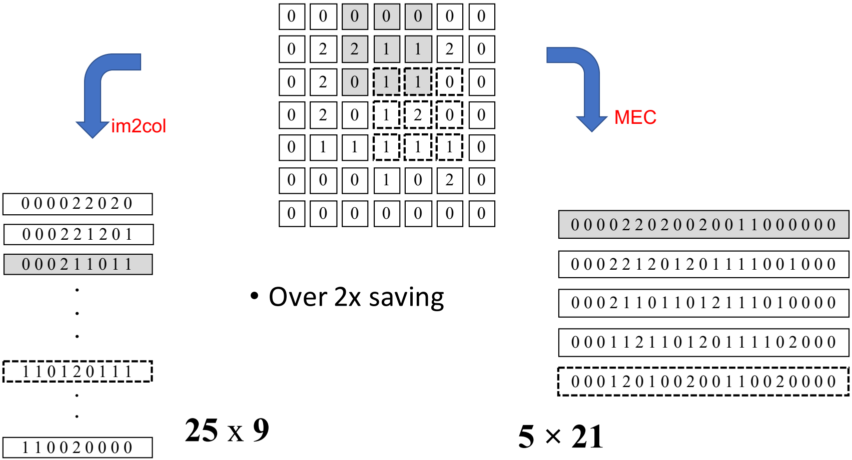
SOTA 1: Memory-efficient Convolution



- 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*.

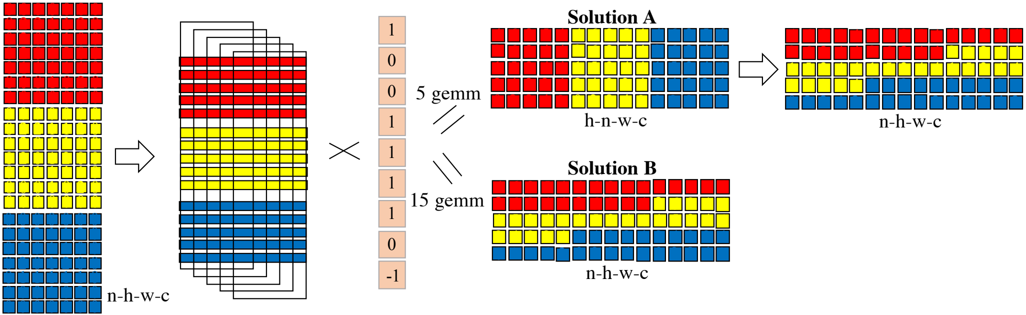
Over 2× memory saving[3]:



- Over 2x saving

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

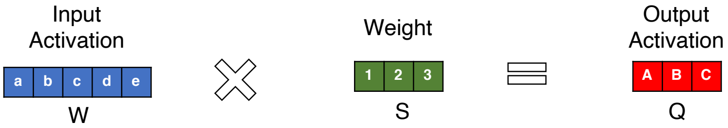
Mini-batch



- MEC w. mini-batch: can use `n-h-w-c` format
- Fusing convolution+pooling can be another solution

Direct Convolution

1D Convolution Example



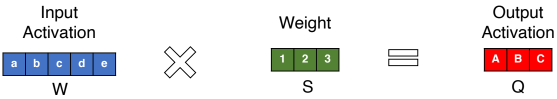

```
for(q=0; q<Q; q++){
   for (s=0; s<S; s++){
      OA[q] += IA[q+s] * W[s];
   }
}
```

Output Stationary (OS) Dataflow

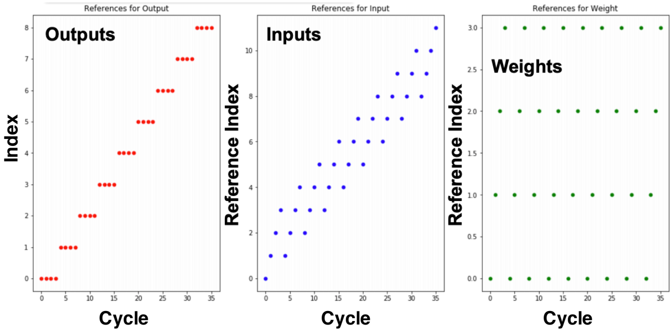
```
for (s=0; s<S; s++){
   for(q=0; q<Q; q++){
      OA[q] += IA[q+s] * W[s];
   }
}
```

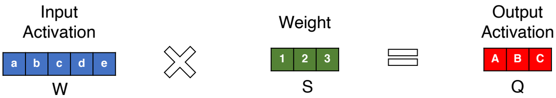
Weight Stationary (WS) Dataflow

Buffer Access Pattern 1: Output Stationary

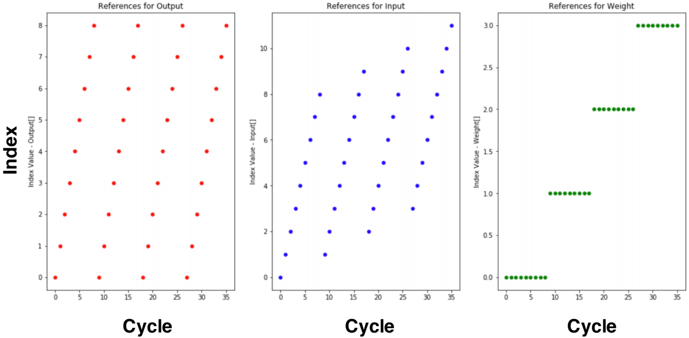



```
for(q=0; q<Q; q++){ // Q =9
  for (s=0; s<S; s++){ // S=4
    OA[q] += IA[q+s] * W[s];
  }
}
```

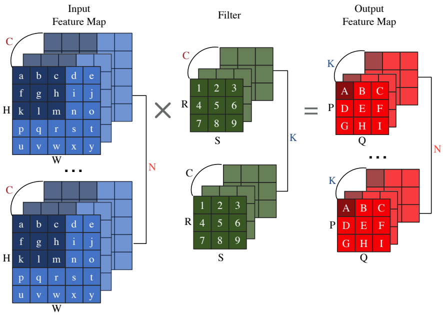





```
for (s=0; s<S; s++){// S=4
  for(q=0; q<Q; q++){// Q =9
    OA[q] += IA[q+s] * W[s];
  }
```

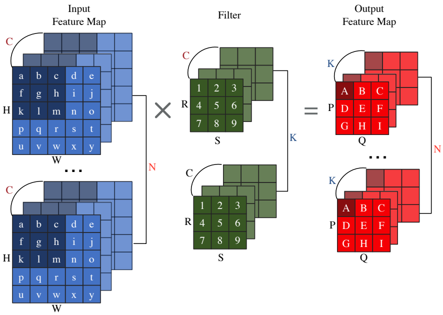


Direct Convolution



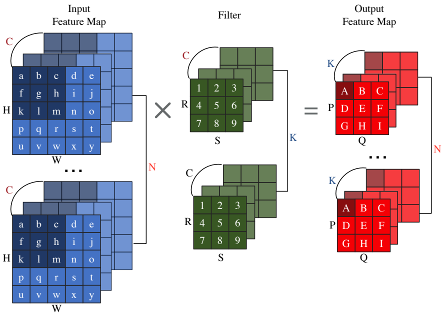

```
for (n=0; n<N; n++) {
for (k=0; k<K; k++) {
for (p=0; p<P; p++) {
for (q=0; q<Q; q++) {
    OA[n][k][p][q]= 0;
    for (r=0; r<R; r++) {
    for (s=0; s<S; s++) {
    for (c=0; c<C; c++) {
        h = p * stride - pad + r;
        w = q * stride - pad + s;
        OA[n][k][p][q] += IA[n][c][h][w]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

Direct Convolution: Loop Ordering

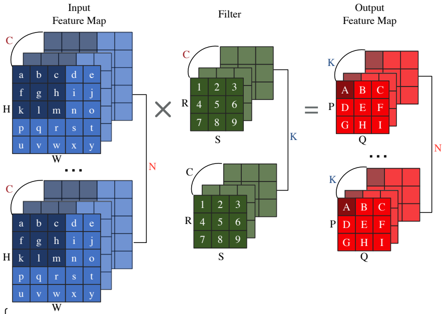



```
for (n=0; n<N; n++) {
for (r=0; r<R; r++) {
for (s=0; s<S; s++) {
for (c=0; c<C; c++) {
for (k=0; k<K; k++) {
    float curr_w = W[r][s][c][k];
    for (p=0; p<P; p++) {
    for (q=0; q<Q; q++) {
        h = p * stride - pad + r;
        w = q * stride - pad + s;
        OA[n][k][p][q] += IA[n][c][h][w]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




```
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;
} } } } } }
```

Memory Layout

Data Layout Formats[4]

- N is the batch size
- C is the number of feature maps
- H is the image height
- W is the image width

EXAMPLE

N = 1
C = 64
H = 5
W = 4

c = 0

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

c = 1

20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35
36	37	38	39

c = 2

40	41	42	43
44	45	46	47
48	49	50	51
52	53	54	55
56	57	58	59

...

c = 30

600	601	602	603
604	605	606	607
608	609	610	611
612	613	614	615
616	617	618	619

c = 31

620	621	622	623
624	625	626	627
628	629	630	631
632	633	634	635
636	637	638	639

c = 32

640	641	642	643
644	645	646	647
648	649	650	651
652	653	654	655
656	657	658	659

...

...

c = 62

1240	1241	1242	1243
1244	1245	1246	1247
1248	1249	1250	1251
1252	1253	1254	1255
1256	1257	1258	1259

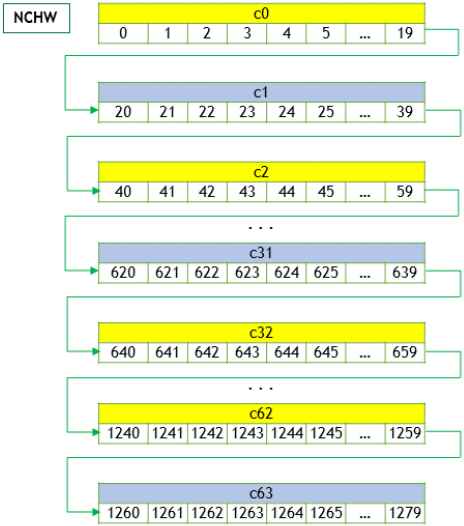
c = 63

1260	1261	1262	1263
1264	1265	1266	1267
1268	1269	1270	1271
1272	1273	1274	1275
1276	1277	1278	1279

[4] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html

NCHW Memory Layout

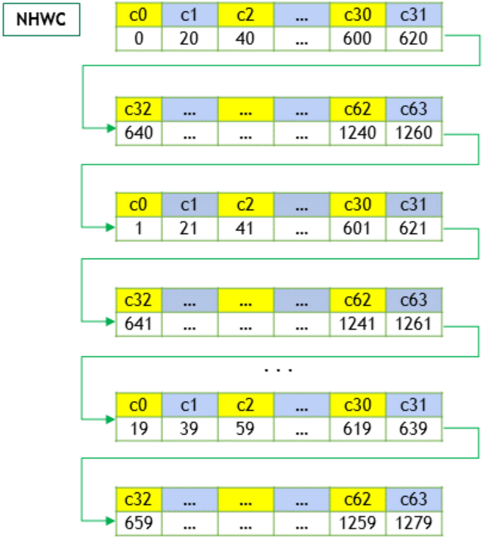
- Begin with first channel (c=0), elements arranged contiguously in row-major order
- Continue with second and subsequent channels until all channels are laid out



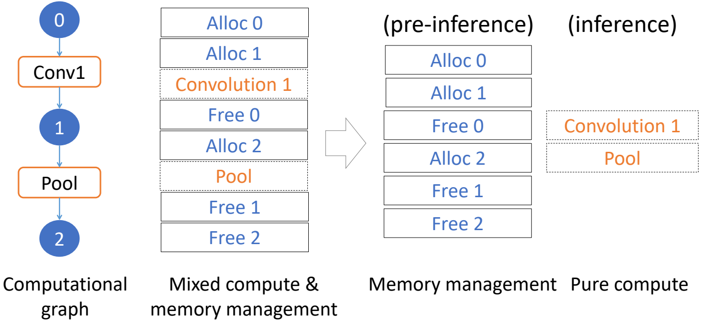
NHWC Memory Layout

- Begin with the first element of channel 0, then proceed to the first element of channel 1, and so on, until the first elements of all the C channels are laid out
- Next, select the second element of channel 0, then proceed to the second element of channel 1, and so on, until the second element of all the channels are laid out
- Follow the row-major order of channel 0 and complete all the elements
- Proceed to the next batch (if N is > 1)

NHWC Memory Layout



Memory optimization of MNN



- MNN can infer the exact required memory for the entire graph:
  - virtually walking through all operations
  - summing up all allocation and freeing

Reading List

- 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/`