香港中文大學
The Chinese University of Hong Kong

Efficient Computing of Deep Neural Networks

Bei Yu
Department of Computer Science & Engineering
Chinese University of Hong Kong
byu@cse.cuhk.edu.hk

September 30, 2021

Computer Vision

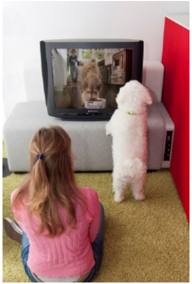
- Humans use their eyes and their brains to visually sense the world.
- Computers user their cameras and computation to visually sense the world

Objects
Activities
Scenes
Locations
Text
Faces
Gestures
Motions
Emotions…

"Eyes" "Brain"

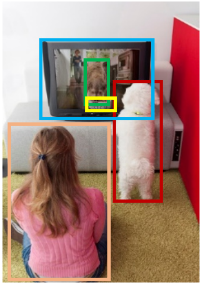
Jian Sun, "Introduction to Computer Vision and Deep Learning".

Few More Core Problems



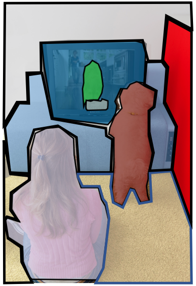
Classification

Image



Detection

Region



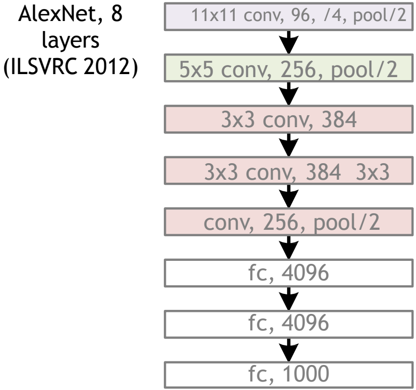
Segmentation

Pixel

Sequence

Video

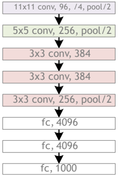
Revolution of Depth



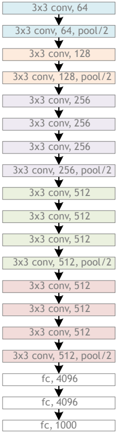
Slide Credit: He et al. (MSRA)

Revolution of Depth

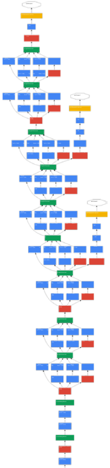
AlexNet, 8 layers (ILSVRC 2012)



VGG, 19 layers (ILSVRC 2014)



GoogleNet, 22 layers (ILSVRC 2014)



Slide Credit: He et al. (MSRA)

Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)

VGG, 19 layers (ILSVRC 2014)

ResNet, 152 layers (ILSVRC 2015)

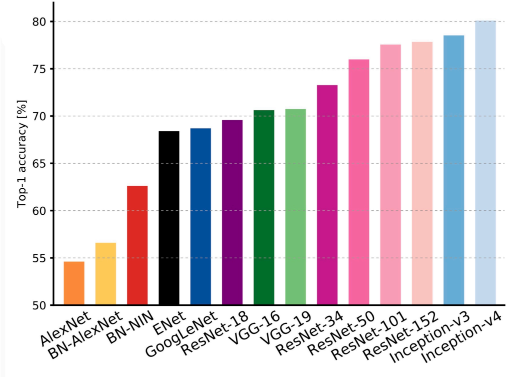
Slide Credit: He et al. (MSRA)

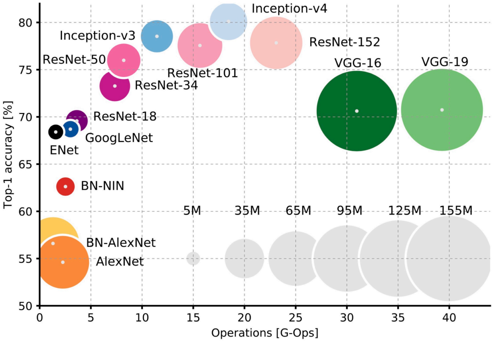
Some Recent Classification Architectures

- AlexNet (Krizhevsky, Sutskever, and E. Hinton 2012) 233MB
- Network in Network (Lin, Q. Chen, and Yan 2013) 29MB
- VGG (Simonyan and Zisserman 2015) 549MB
- GoogleNet (Szegedy, Liu, et al. 2015) 51MB
- ResNet (K. He et al. 2016) 215MB
- Inception-ResNet (Szegedy, Vanhoucke, et al. 2016)
- DenseNet (Huang et al. 2017)
- Xception (Chollet 2017)
- MobileNetV2 (Sandler et al. 2018)
- ShuffleNet (Zhang, Zhou, et al. 2018)

Some Recent Classification Architectures

- AlexNet (Krizhevsky, Sutskever, and E. Hinton 2012) 233MB
- Network in Network (Lin, Q. Chen, and Yan 2013) 29MB
- VGG (Simonyan and Zisserman 2015) 549MB
- GoogleNet (Szegedy, Liu, et al. 2015) 51MB
- ResNet (K. He et al. 2016) 215MB
- Inception-ResNet (Szegedy, Vanhoucke, et al. 2016) 23MB
- DenseNet (Huang et al. 2017) 80MB
- Xception (Chollet 2017) 22MB
- MobileNetV2 (Sandler et al. 2018) 14MB
- ShuffleNet (Zhang, Zhou, et al. 2018) 22MB





[1]

[1] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello (2017). "An analysis of deep neural network models for practical applications". In: *arXiv preprint*.

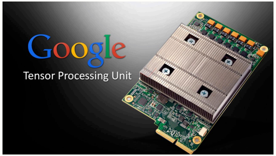
When Machine Learning Meets Hardware

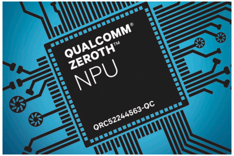
Convolution layer is one of the most expensive layers

- Computation pattern
- Emerging challenges

More and more end-point devices with limited memory

- Cameras
- Smartphone
- Autonomous driving

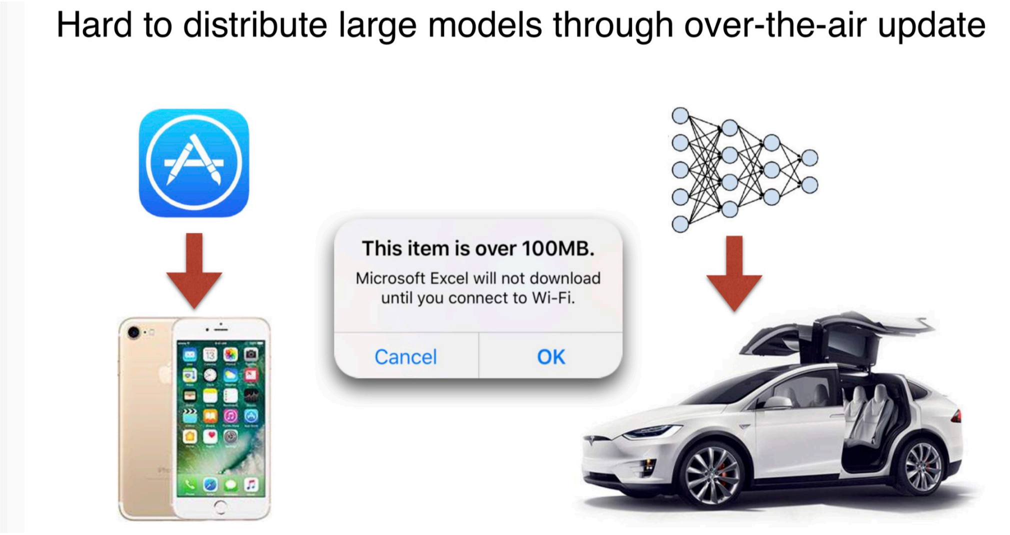




XILINX

MOBILEYE®
An Intel Company

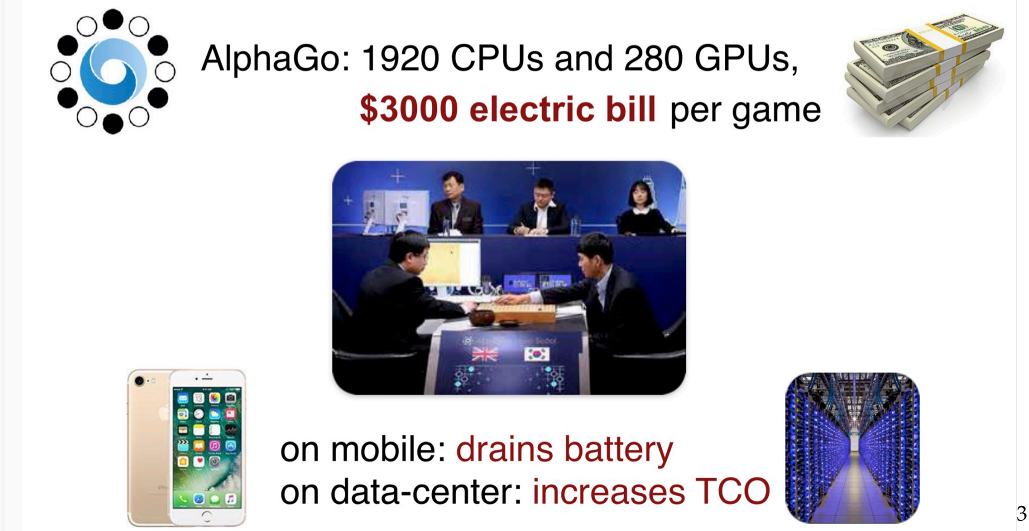
1st Challenge: Model Size



[2]

[2] Song Han and William J. Dally (2018). "Bandwidth-efficient Deep Learning". In: *Proc. DAC*, 147:1–147:6.

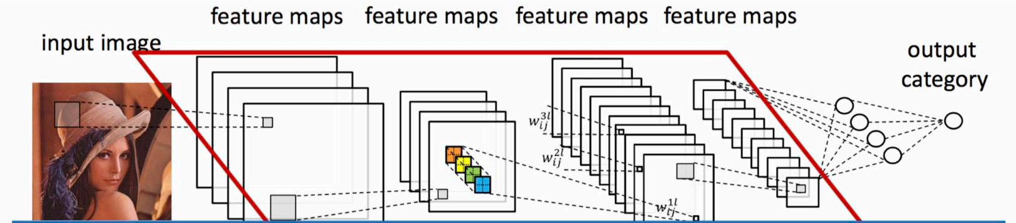
2nd Challenge: Energy Efficiency



[3]

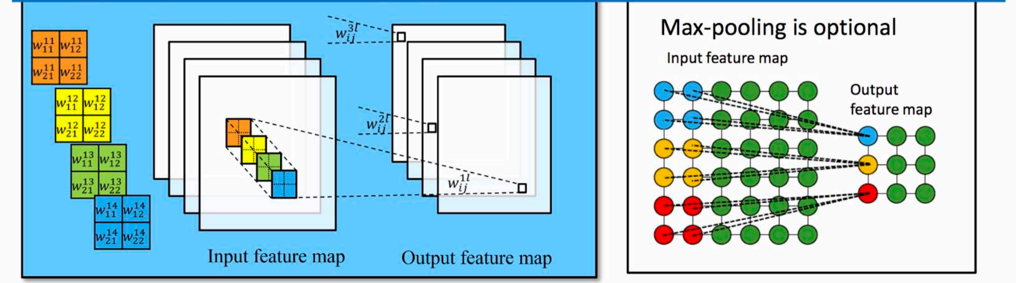
[3] Song Han and William J. Dally (2018). "Bandwidth-efficient Deep Learning". In: *Proc. DAC*, 147:1–147:6.

Convolution Is the Bottleneck



Convolutional layers account for over 90% computation

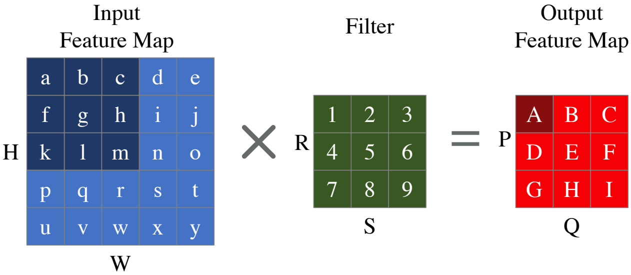
[1] A. Krizhevsky, etc. Imagenet classification with deep convolutional neural networks. NIPS 2012.
[2] J. Cong and B. Xiao. Minimizing computation in convolutional neural networks. ICANN 2014



Outline

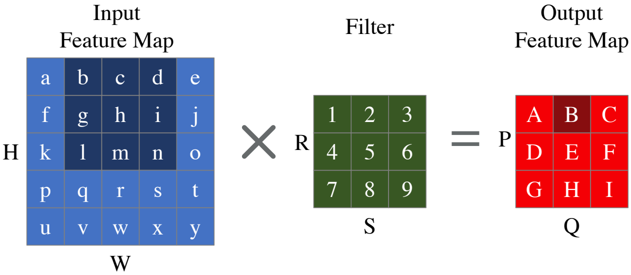
Convolution Basis

2D-Convolution



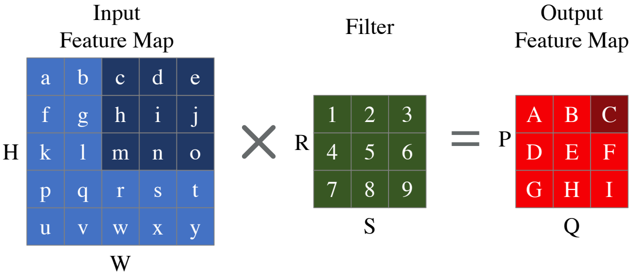
$$
\begin{aligned}
A = a \cdot 1 + b \cdot 2 + c \cdot 3 \\
+ f \cdot 4 + g \cdot 5 + h \cdot 6 \\
+ k \cdot 7 + l \cdot 8 + m \cdot 9
\end{aligned}
$$

- 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



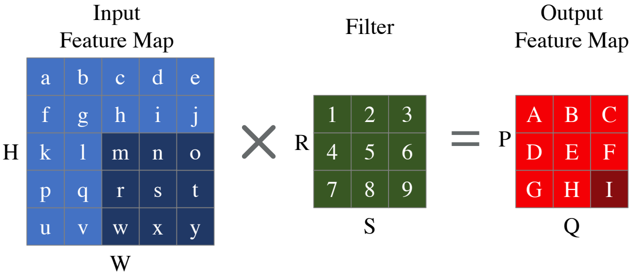
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step

2D-Convolution

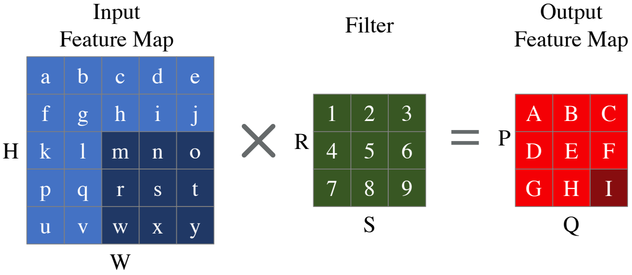


- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step

2D-Convolution

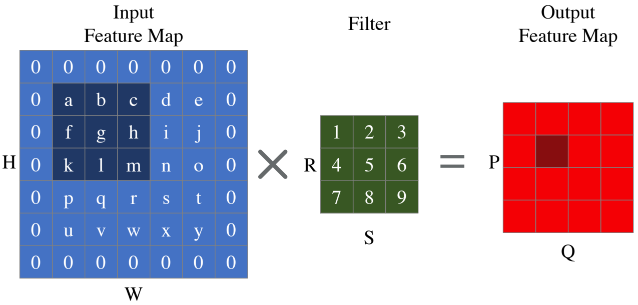


- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step



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

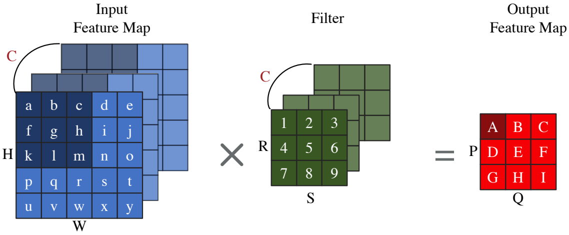
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step



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

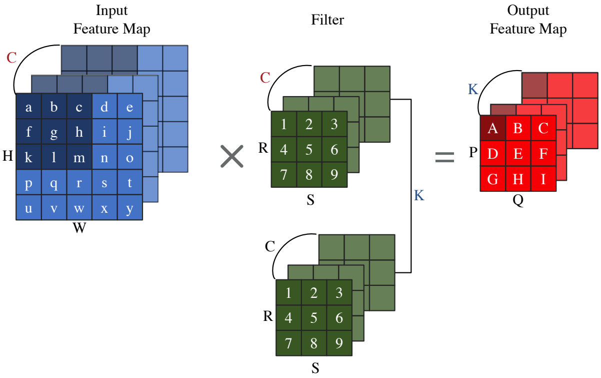
- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step
- padding: # of zero rows/columns added

3D-Convolution

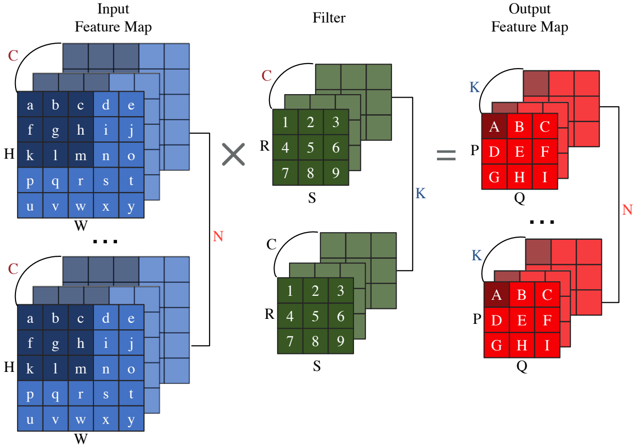


- H: Height of input feature map
- W: Width of input feature map
- R: Height of filter
- S: Width of filter
- P: Height of output feature map
- Q: Width of output feature map
- stride: # of rows/columns traversed per step
- padding: # of zero rows/columns added
- C: # of input channels

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



- 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

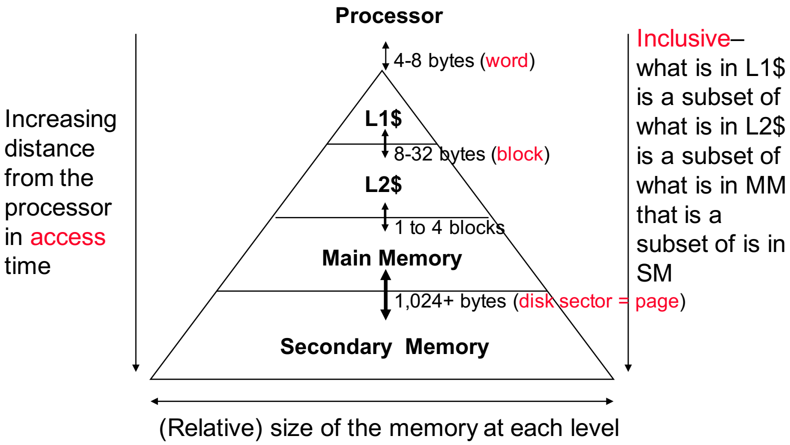
Convolution 101

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

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

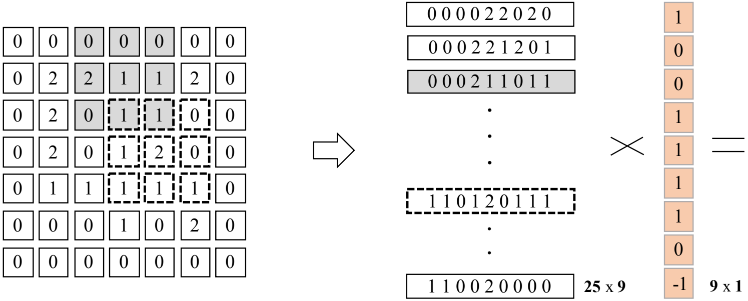
GEMM

```c
#include <stdio.h>
#include <memory.h>
#include <time.h>
#include <stdlib.h>
#include <sys/time.h>

double a_arr[1024][1024];
double b_arr[1024][1024];

int main()
{
    int N = 1024;
    for(int i=0;i<1024;i++) for(int j=0;j<1024;j++)
    {
        a_arr[i][j] = 1; b_arr[i][j] = 2;
    }
    double sum;

    struct timeval startTime,endTime;
    float Timeuse;
    gettimeofday(&startTime,NULL);

    // ===========================================
    //     following are the key operations
    for(int i=0; i<1024; i++){
        for(int j=0; j<1024; j++){
            sum += a_arr[j][i] * b_arr[j][i];
        }
    }
    // ===========================================

    gettimeofday(&endTime,NULL);
    Timeuse = 1000000 * (endTime.tv_sec-startTime.tv_sec) + (endTime.tv_usec-startTime.tv_us
    printf("============ Cache Optimization Demo ========== \n");
    printf("total timeuse = %.2f us \n",Timeuse);

    return 0;
}
```

```c
#include <stdio.h>
#include <memory.h>
#include <time.h>
#include <stdlib.h>
#include <sys/time.h>

double a_arr[1024][1024];
double b_arr[1024][1024];

int main()
{
    int N = 1024;
    for(int i=0;i<1024;i++) for(int j=0;j<1024;j++)
    {
        a_arr[i][j] = 1; b_arr[i][j] = 2;
    }
    double sum;

    struct timeval startTime,endTime;
    float Timeuse;
    gettimeofday(&startTime,NULL);

    // ===========================================
    //     following are the key operations
    for(int i=0; i<1024; i++){
        for(int j=0; j<1024; j++){
            sum += a_arr[i][j] * b_arr[i][j];
        }
    }
    // ===========================================

    gettimeofday(&endTime,NULL);
    Timeuse = 1000000 * (endTime.tv_sec-startTime.tv_sec) + (endTime.tv_usec-startTime.tv_u
    printf("============ Cache Optimization Demo ========== \n");
    printf("total timeuse = %.2f us \n",Timeuse);

    return 0;
}
```

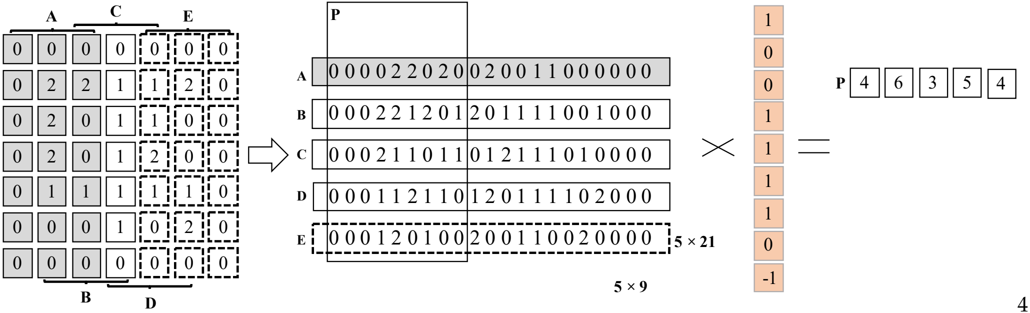
Same complexity; same real runtime?

Im2col (Image2Column) 2D-Convolution



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

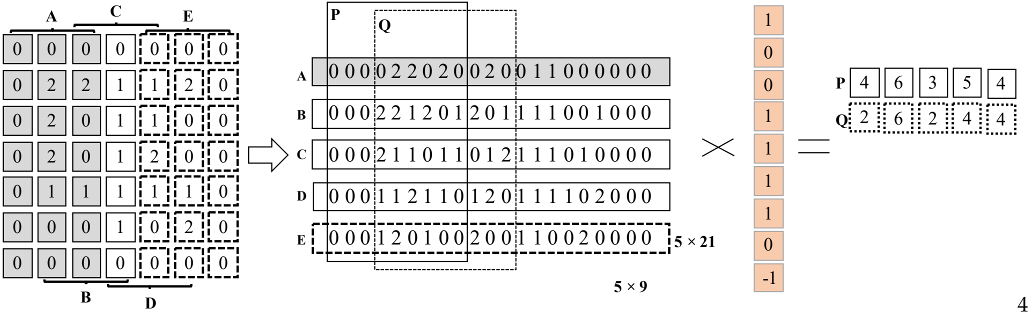
SOTA 1: Memory-efficient Convolution



4

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

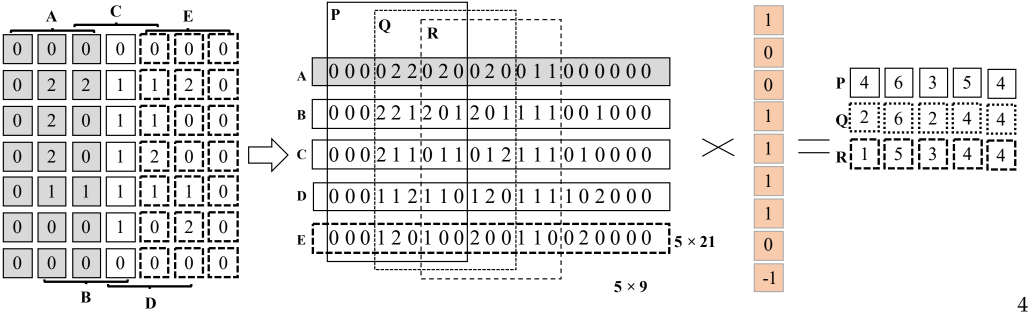
[4] Minsik Cho and Daniel Brand (2017). “MEC: memory-efficient convolution for deep neural network”. In: *Proc. ICML*.



4

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

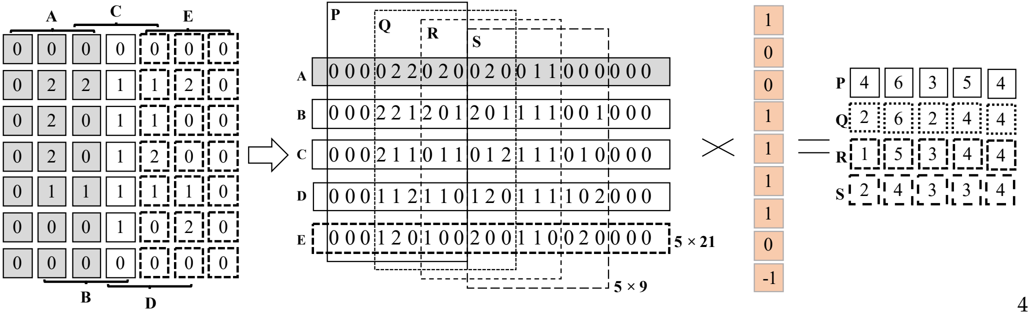
[4] Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.



[4]

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

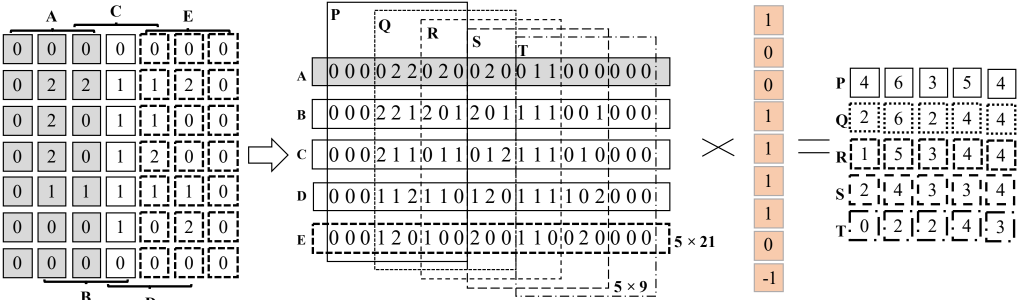
[4] Minsik Cho and Daniel Brand (2017). "MEC: memory-efficient convolution for deep neural network". In: *Proc. ICML*.



4

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

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



4

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

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

Over 2× memory saving:

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

im2col

0 0 0 0 2 2 0 2 0
0 0 0 2 2 1 2 0 1
0 0 0 2 1 1 0 1 1
.
.
.
1 1 0 1 2 0 1 1 1
.
.
1 1 0 0 2 0 0 0 0

25 x 9

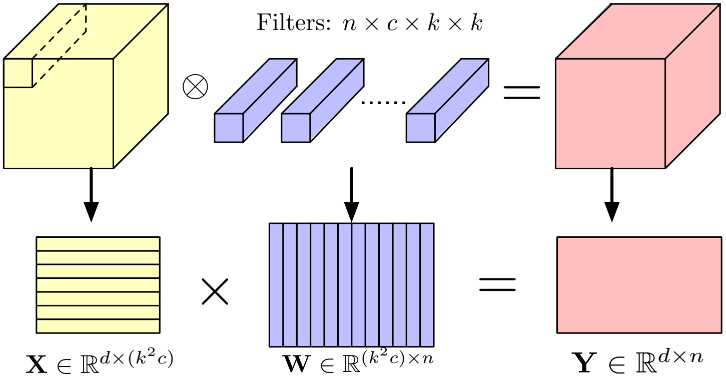
- Over 2x saving

MEC

0 0 0 0 2 2 0 2 0 0 2 0 0 1 1 0 0 0 0 0 0
0 0 0 2 2 1 2 0 1 2 0 1 1 1 1 0 0 1 0 0 0
0 0 0 2 1 1 0 1 1 0 1 2 1 1 1 0 1 0 0 0 0
0 0 0 1 1 2 1 1 0 1 2 0 1 1 1 1 0 2 0 0 0
0 0 0 1 2 0 1 0 0 2 0 0 1 1 0 0 2 0 0 0 0

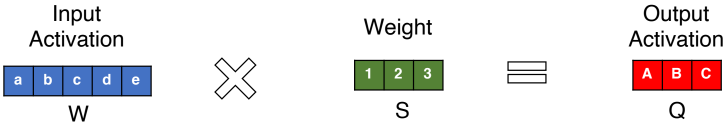
5 × 21

`Im2col` (Image2Column) 3D-Convolution



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

Direct Convolution



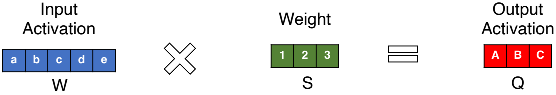

```
for(q=0; q<Q; q++){
  for (s=0; s<S; s++){
    OA[q] += IA[q+s] * W[s];
  }
}
```

Output Stationary (OS) Dataflow

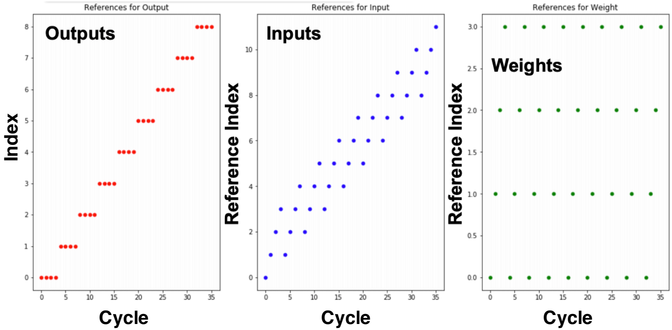
```
for (s=0; s<S; s++){
  for(q=0; q<Q; q++){
    OA[q] += IA[q+s] * W[s];
  }
}
```

Weight Stationary (WS) Dataflow

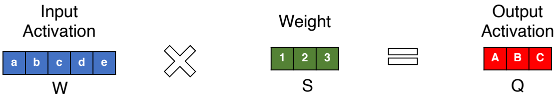
Buffer Access Pattern 1: Output Stationary



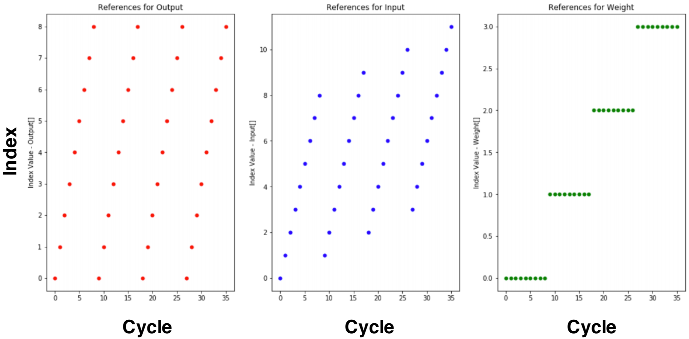

```
for(q=0; q<Q; q++){ // Q =9
   for (s=0; s<S; s++){ // S=4
      OA[q] += IA[q+s] * W[s];
   }
}
```



Buffer Access Pattern 2: Weight Stationary

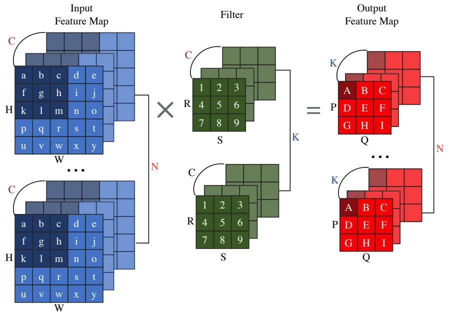



```
for (s=0; s<S; s++){// S=4
   for(q=0; q<Q; q++){// Q =9
      OA[q] += IA[q+s] * W[s];
   }
```



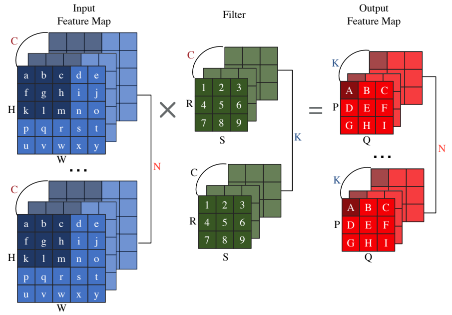
Direct Convolution

```
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];
} } } } } } }
```



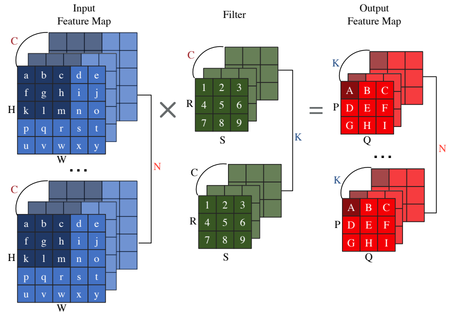
Direct Convolution: Loop Ordering

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

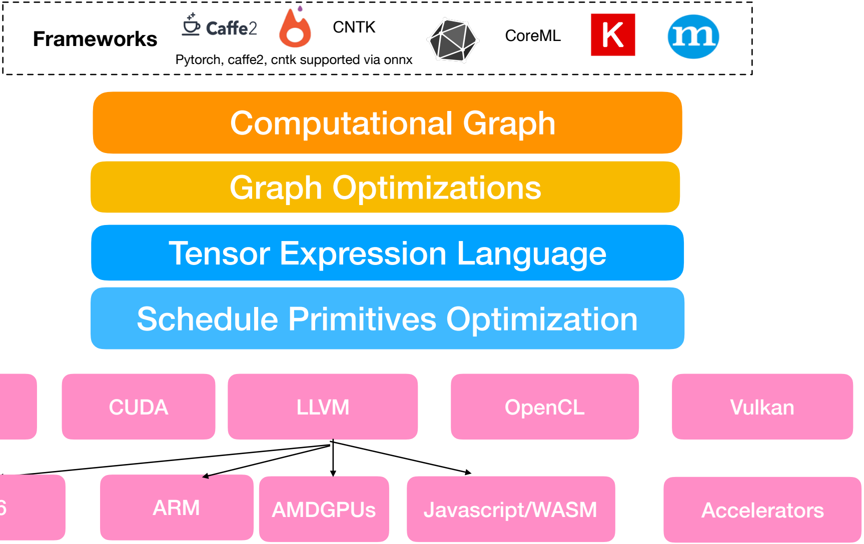
Direct Convolution: Loop Ordering + Unrolling + Tiling

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

Questions:

- How many configurations we have?
- How to search for the BEST configuration?
- How to handle different backend devices?

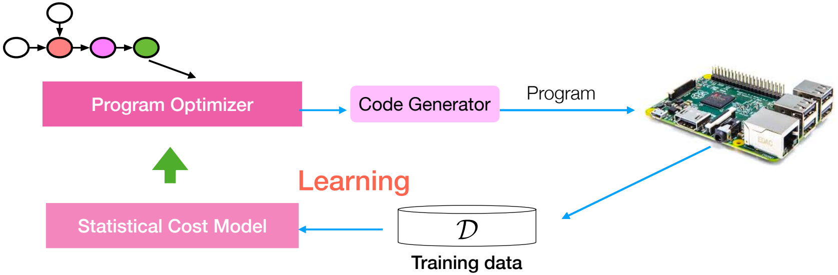
Deep Learning Compiler Example: TVM[5]



[5] Only SOTA on some AI chips; but NOT for x86 CPU (OpenVINO), GPU (TensorRT), or Xilinx FPGA (Vitis AI).

TVM: End to End Optimization

Layer-wise Optimization: Autotuning

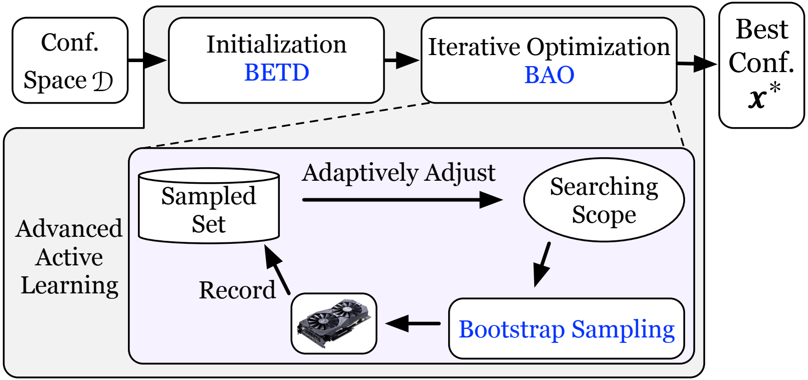


Tuning algorithms:

- Active learning.
- Transfer learning.
- Reinforcement learning.

Active Learning driven TVM [DATE'21][6]

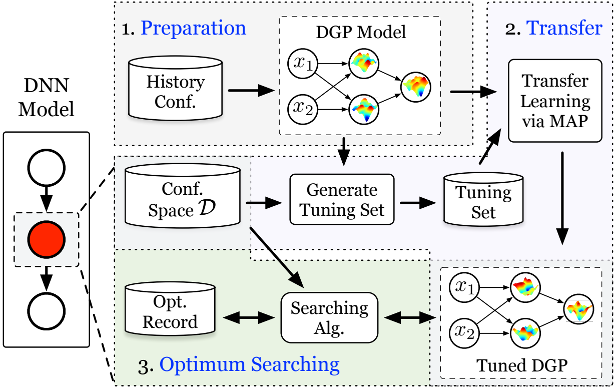
- Batch transductive experimental design
- Bootstrap-guided adaptive optimization



[6] Qi Sun, Chen Bai, Hao Geng, et al. (2021). "Deep neural network hardware deployment optimization via advanced active learning". In: *Proc. DATE*, pp. 1510–1515.

Deep Gaussian Transfer Learning [ICCV'21][7]

1. preparation: learn a deep Gaussian process model from historical data
2. transfer: transfer knowledge of the DGP model to new tasks
3. optimal searching: guide the optimization of new tasks with the tuned DGP model

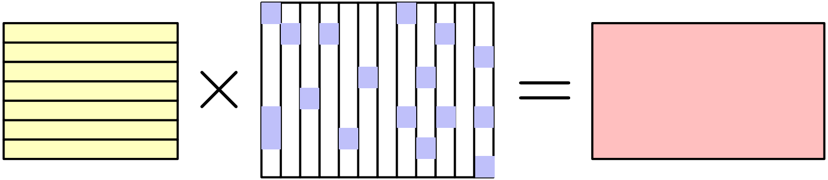


[7] Qi Sun, Chen Bai, Tinghuan Chen, et al. (2021). "Fast and Efficient DNN Deployment via Deep Gaussian Transfer Learning". In: *Proc. ICCV*.

Sparse Convolution

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

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

Algorithm 1 Sparse Convlution Naive 1

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

rowptr

- → row0 (3,2)
- → row1 (7,0)
- → row2 (4,2), (8,3)
- → row3 (6,0), (5,1), (3,2)
- → row4 (2,0), (1,3)
- → row5 (8,3)

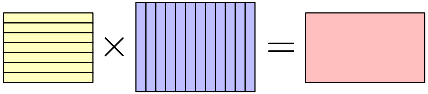
Compressed Sparse Row (CSR)

colptr

- → col0 (7,1), (6,3), (2,4)
- → col1 (5,3)
- → col2 (3,0), (4,2), (3,3)
- → col3 (8,2), (1,4), (8,5)

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

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

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

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

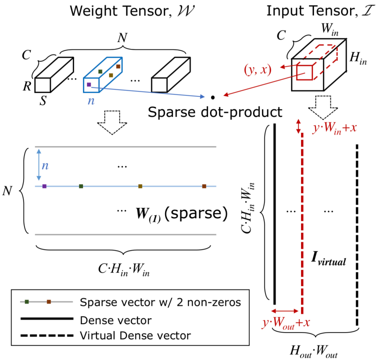


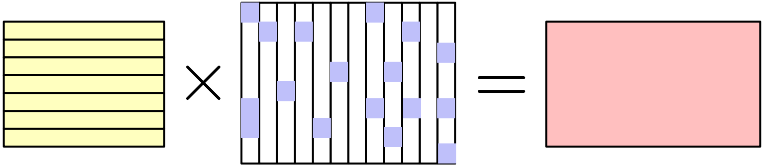
Figure 1: Conceptual view of the direct sparse convolution algorithm. Computation of output value at (y,x)th position of nth output channel is highlighted.

```
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 $(0,y,x)$. [8]

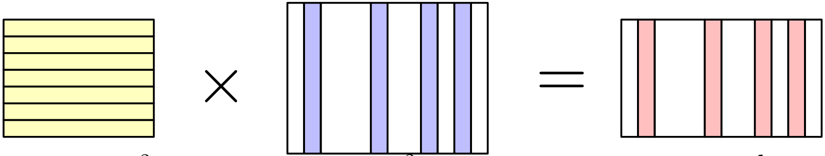
[8] Jongsoo Park et al. (2017). "Faster CNNs with direct sparse convolutions and guided pruning". In: *Proc. ICLR*.

Discussion: Randome Sparsity vs. Structured Sparsity



$\mathbf{X} \in \mathbb{R}^{d \times (k^2 c)}$ $\mathbf{S} \in \mathbb{R}^{(k^2 c) \times n}$ $\mathbf{Y} \in \mathbb{R}^{d \times n}$

Random Sparse Convolution



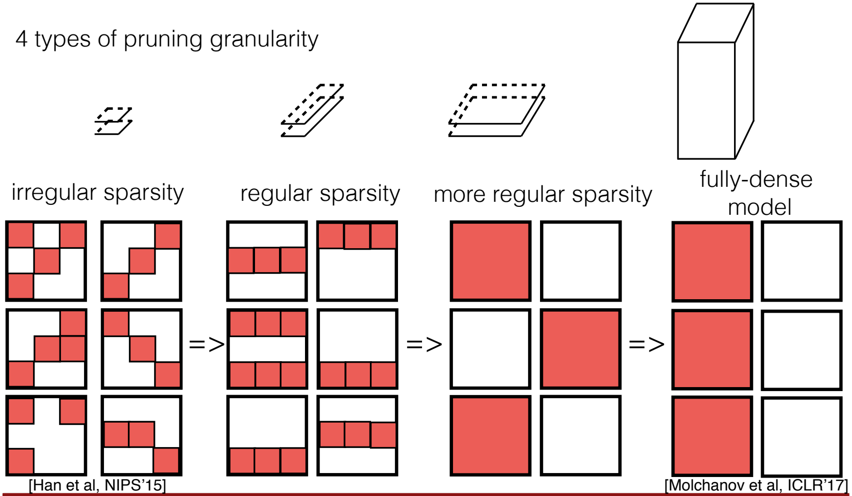
$\mathbf{X} \in \mathbb{R}^{d \times (k^2 c)}$ $\mathbf{S} \in \mathbb{R}^{(k^2 c) \times n}$ $\mathbf{Y} \in \mathbb{R}^{d \times n}$

Structured Sparse Convolution

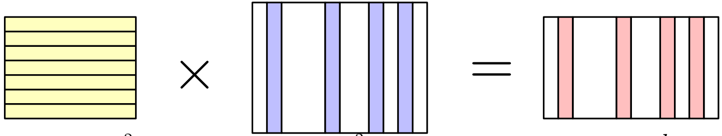
Discussion: Sparsity Granularity

Exploring the Granularity of Sparsity that is Hardware-friendly

4 types of pruning granularity



Compression Approach: Sparsity[9,10]



$$\mathbf{X} \in \mathbb{R}^{d\times(k^2c)} \quad \times \quad \mathbf{S} \in \mathbb{R}^{(k^2c)\times n} \quad = \quad \mathbf{Y} \in \mathbb{R}^{d\times n}$$

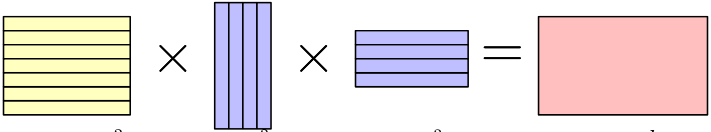
Sparse DNN

- *Sparsification*: weight pruning;
- *Compression*: compressed sparse format for storage;
- *Potential acceleration*: sparse matrix multiplication algorithm.

[9] Wei Wen et al. (2016). “Learning structured sparsity in deep neural networks”. In: *Proc. NIPS*, pp. 2074–2082.

[10] Yihui He, Xiangyu Zhang, and Jian Sun (2017). “Channel Pruning for Accelerating Very Deep Neural Networks”. In: *Proc. ICCV*.

Compression Approach: Low-Rank[11,12]



$\mathbf{X} \in \mathbb{R}^{d \times (k^2 c)}$ $\quad$ $\mathbf{U} \in \mathbb{R}^{(k^2 c) \times r}$ $\quad$ $\mathbf{V} \in \mathbb{R}^{1^2 r \times n}$ $\quad$ $\mathbf{Y} \in \mathbb{R}^{d \times n}$

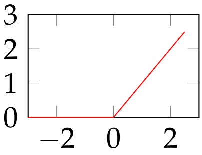
Low-rank DNN

- *Low-rank approximation*: matrix decomposition or tensor decomposition.
- *Compression and acceleration*: less storage required and less FLOP in computation.

[11] Xiangyu Zhang, Jianhua Zou, et al. (2015). "Efficient and accurate approximations of nonlinear convolutional networks". In: *Proc. CVPR*, pp. 1984–1992.

[12] Xiyu Yu et al. (2017). "On compressing deep models by low rank and sparse decomposition". In: *Proc. CVPR*, pp. 7370–7379.

Non-linearity Approximation[13]



ReLU

- Activation unit: `ReLU`
- Error more sensitive to positive response;
- Enlarge the solution space.

$$\min_{\boldsymbol{W}} \sum_{i=1}^{N} \|\boldsymbol{W}\boldsymbol{X}_i - \boldsymbol{Y}_i\|_F \rightarrow \min_{\boldsymbol{W}} \sum_{i=1}^{N} \|r(\boldsymbol{W}\boldsymbol{X}_i) - \boldsymbol{Y}_i\|_F$$

- $\boldsymbol{X}$: input feature map
- $\boldsymbol{Y}$: output feature map

[13] Xiangyu Zhang, Jianhua Zou, et al. (2015). "Efficient and accurate approximations of nonlinear convolutional networks". In: *Proc. CVPR*, pp. 1984–1992.

CENG 5030
Energy Efficient Computing

CENG5030 Energy Efficient (Deep Neural Network) Computing – Spring 2021

Lecture:	M 12:30-14:15	Venue: zoom
	W 16:30-18:15	Venue: zoom
Course Instructor:	Prof. Bei Yu	byu@cse.cuhk.edu.hk
Course Tutors:	Qi Sun	qsun@cse.cuhk.edu.hk

- Honesty in Academic Work
- Staff Student Expectations

Announcements

CENG5030

- Feb. 24, 2021: Course schedule is updated.
- Jan. 18, 2021: Grading policy is updated (in-class discussion and decent project are encouraged).
- Dec. 01, 2020: Course webpage is built up and the teaching schedule is online.

Description:

This course is an intensive and research oriented course, discussing some practical and fundamental techniques, skills, and tools for deep neural network acceleration, in particular inference acceleration. The students selecting this course are assumed to have solid DNN experience and some programming skills (e.g, C/C).

Course Requirements

- Lab (40%), in-class discussion (10%), midterm-project report (15%), final project (35%), and extra bonus.
- Please submit your lab reports through blackboard.
- The mid-term and final project report and presentation are judged very seriously, considering both idea novelty and workload.

Schedule

Week	Date	Topic	Remark
1	Jan. 11	L01 Introduction (slides)	
	Jan. 13	Lab01 PyTorch (report, code)	Due: Jan. 27
2	Jan. 18	Lab02 Training (report, code)	Due: Feb. 02
	Jan. 20	L02 CNN Training (slides)	
3	Jan. 25	L03 Pruning (slides)	
	Jan. 27	L04 Accurate Speed-up I (slides)	
4	Feb. 01	continue on L04	
	Feb. 03	Lab03 GEMM (report, code)	Due: Feb. 17
5	Feb. 08	L05 Quantization (slides)	
	Feb. 10	L06 Binary/Tenary Networks (slides)	
6	Feb. 15	n/a	Lunar New Year Holiday
	Feb. 17	n/a	Lunar New Year Holiday
7	Feb. 22	Lab04 Distiller (report, slides, code)	Due: Mar. 08

CMSC 5743
Efficient Computing of Deep Neural Networks

CMSC 5743 Efficient Computing of Deep Neural Networks – Fall 2021

Lecture:	F 18:30-21:15	WMY 506
		Zoom: https://cuhk.zoom.us/j/98497053271
Instructor:	Prof. Bei Yu	byu@cse.cuhk.edu.hk
Course Tutors:	Yang Bai	ybai@cse.cuhk.edu.hk
	Qi Sun	qsun@cse.cuhk.edu.hk

- Honesty in Academic Work
- Staff Student Expectations

CMSC5743

Announcements

- Aug. 04, 2021: Course schedule is updated – no final project but three homeworks are introduced.
- May. 05, 2021: Course webpage is built up and the teaching schedule is online.

Course Requirements

- Homework (30%), Lab (70%), and extra bonus.
- Please submit your lab reports and homework through blackboard (link).

Schedule

Date	Topic	Lab/Tutorial	Homework	Note
Sep. 10	L01 Introduction (slides)	Lab01 PyTorch (report, code)		Due on Sep. 19
Sep. 17	L02 Conv Speedup (slides)	Lab02 GEMM (report, code, slides)		Due on Sep. 26
Sep. 24	L03 Pruning (slides)	OCR Model (slides)	HW 1	Due on Oct. 14
Oct. 01	N/A			National Day
Oct. 08	L04 Decomposition (slides)	Data Augmentation		
Oct. 15	L05 CUDA	Lab03 CUDA		Due on Oct. 17
Oct. 22	L06 Quantization (slides)	OCR Deployment	HW 2	Due on Nov. 10
Oct. 29	N/A			
Nov. 05	L07 MNN (slides)	Lab04 MNN		Due on Nov. 14
Nov. 12	L08 Binary/Tenary Networks (slides)	FPGA Deployment	HW 3	Due on Nov. 30
Nov. 19	L09 TVM-1 (slides)	Lab05 TVM-1		Due on Nov. 21
Nov. 26	L10 TVM-2	Lab06 TVM-2		Due on Nov. 28
Dec. 03	L11 NAS (slides)	Lab07 TensorRT		Due on Dec. 05

References

- All papers mentioned in lecture slides.
- Sze, Vivienne, et al., "*Efficient processing of deep neural networks*". Synthesis Lectures on Computer Architecture, 2020.