Fast 2D Convolution Algorithms summary

P. Bassia, Prof. Ioannis Pitas
Aristotle University of Thessaloniki
pitas@csd.auth.gr
www.aiia.csd.auth.gr
Version 4.2

Artificial Intelligence &
Information Analysis Lab

Outline

- 2D linear systems
- 2D convolutions

  Discrete-time 2D Systems

  Linear & Cyclic 2D convolutions

  2D Discrete Fourier Transform, 2D Fast Fourier Transform
- Other convolution algorithms

  Winograd algorithm

  Block methods
- Applications in Machine Learning

  Convolutional neural networks

Convolution and correlation

2D convolution applications:

- Machine Learning (Convolutional neural networks)
- Image processing

2D correlation applications:

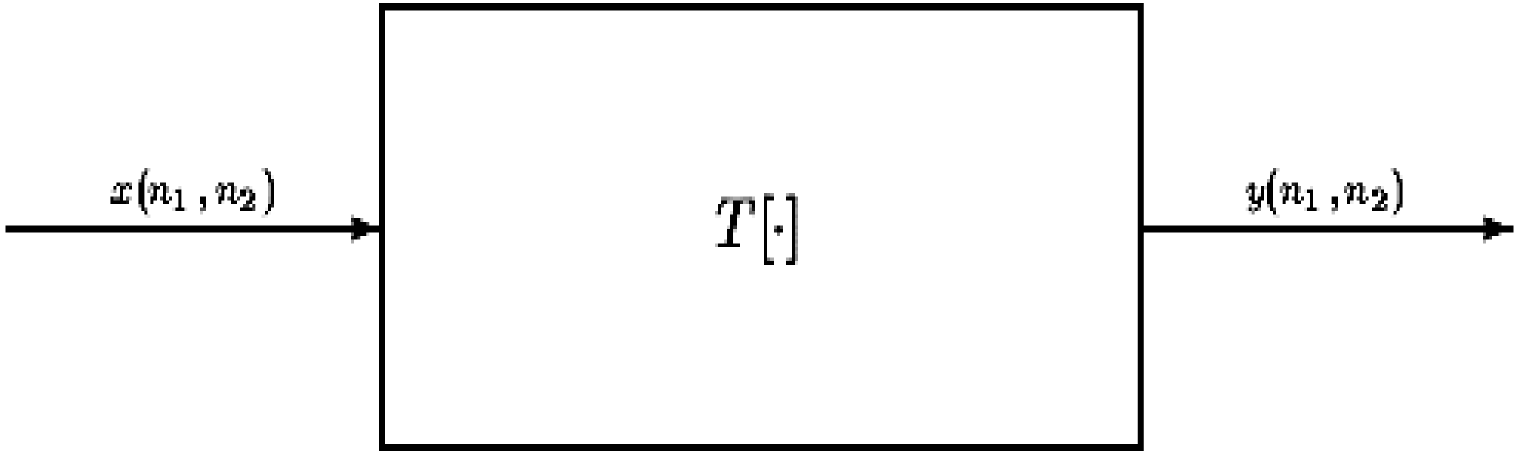
- Feature matching
- Template matching
- Object detection and tracking

2D Discrete Systems

2D system:

- It transforms a 2D discrete input signal $x(n_1, n_2)$ into a 2D discrete-time output signal $y(n_1, n_2)$:

$$y(n_1, n_2) = T[x(n_1, n_2)].$$



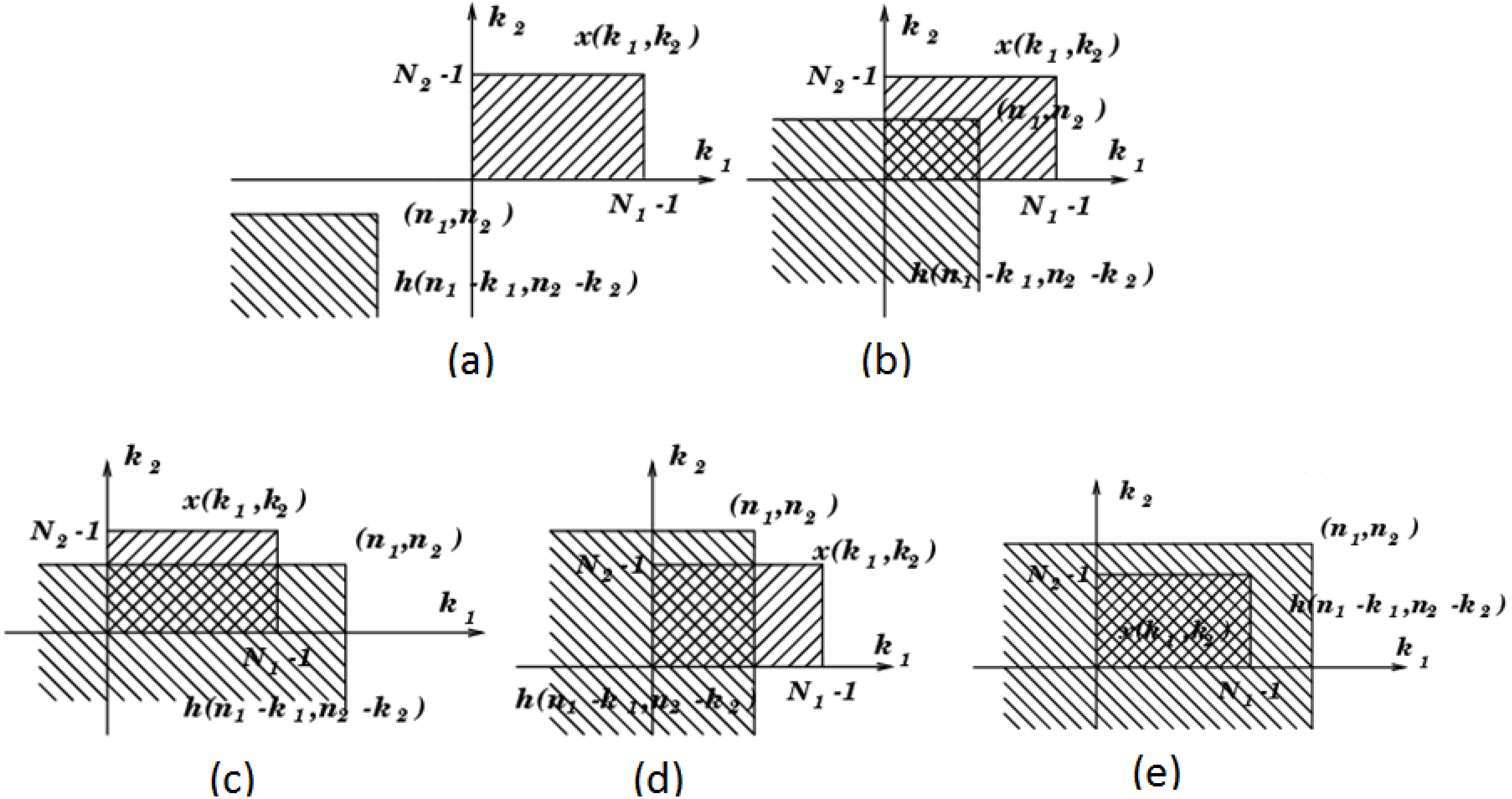
2D Discrete Systems

- A ***Linear Shift Invariant*** (***LSI***) system is described by a 2D convolution of input x with a convolutional kernel h:

$$y(k_1,k_2) = h(k_1,k_2) ** x(k_1,k_2) = \sum_{i_1}\sum_{i_2} h(i_1,i_2)x(k_1-i_1,k_2-i_2).$$

- Input x has typically limited region of support (size), e.g., it can be an image of $N_1 \times N_2$ pixels.
- Convolutional kernel h may have limited or finite region of support $M_1 \times M_2$ pixels.

2D Discrete Systems



Visualization of 2D convolution calculation.

2D Discrete Systems

- ***Finite impulse response*** (***FIR***) systems:

  $h(n_1,n_2)$ is zero outside some filter mask (region) $M_1 \times M_2$,

  $0 \leq n_1 < M_1, 0 \leq n_2 < M_2$.

- FIR filters are described by a 2D linear convolution with convolutional kernel h of size $M_1 \times M_2$ is given by:

$$y(k_1,k_2) = h(k_1,k_2) ** x(k_1,k_2) = \sum_{i_1=0}^{M_1-1} \sum_{i_2=0}^{M_2-1} h(i_1,i_2)x(k_1-i_1,k_2-i_2).$$

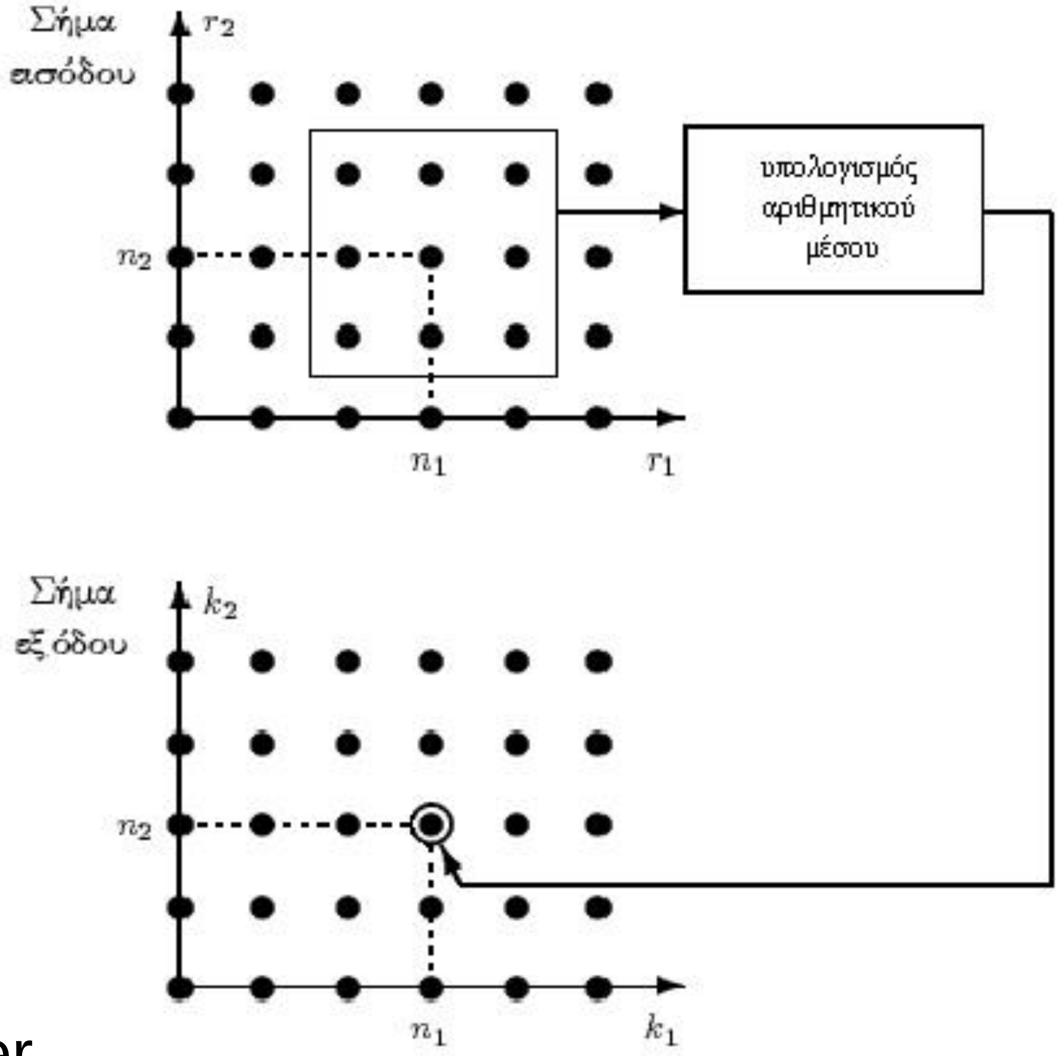
- Usually discrete systems without feedback are FIR ones.

2D Discrete Systems

FIR filter example

- The moving average filter $M_1 \times M_2$, $M_i = 2v_i + 1$:

$$y(n_1, n_2) = \frac{1}{M_1 M_2} \sum_{k_1=-v_1}^{v_1} \sum_{k_2=-v_2}^{v_2} x(n_1 - k_1, n_2 - k_2)$$

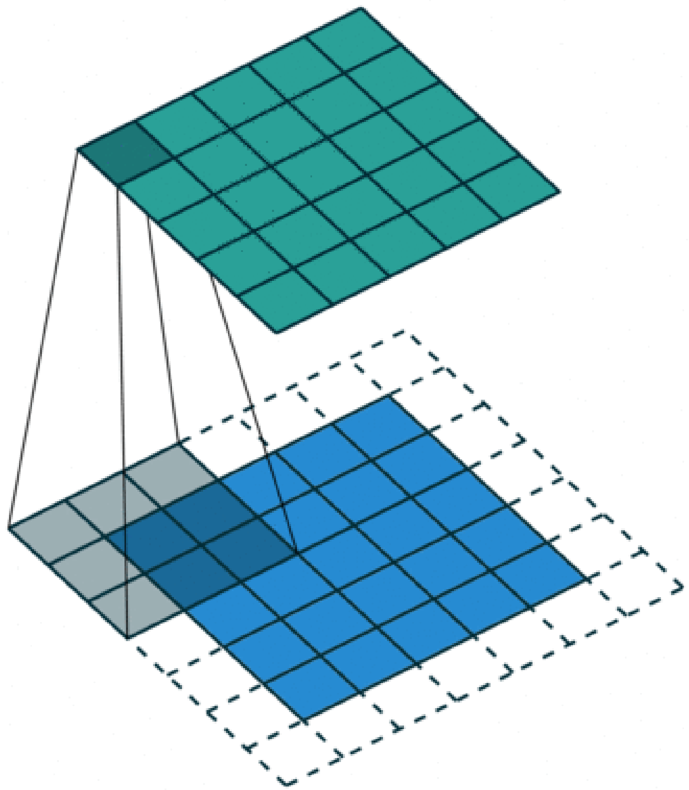


3×3 moving average filter.

2D Discrete Systems

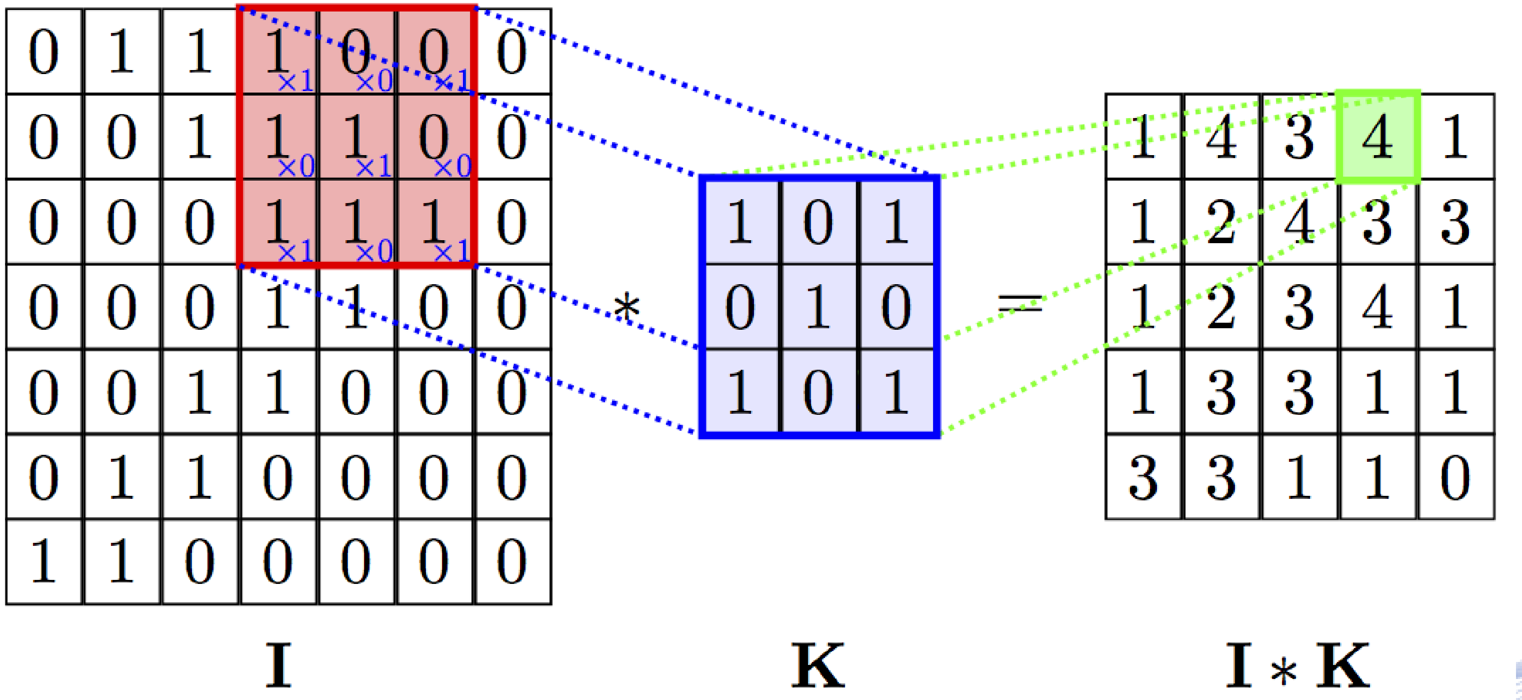
a) Image Lena; b) 5×5 moving average filter output.

2D Discrete Systems



Animation of 2D Convolution with input padding.

2D Discrete Systems



Example of 2D Convolution with input padding.

2D Discrete Systems

- A 2D linear convolution of convolutional kernel h of size $M_1 \times M_2$ operating on an image x of size $N_1 \times N_2$ of size produces an output image y:
  - of size $M_1 M_2$ using zero padding
    - ***Complexity***: $N_1 N_2 M_1 M_2$ multiplications.
  - of size $(N_1 - M_1 + 1)(N_2 - M_2 + 1)$, without input image border padding.
    - ***Complexity***: $(N_1 - M_1 + 1)(N_2 - M_2 + 1) M_1 M_2$ multiplications.
- In both cases complexity is $O(N^4)$, if N_1, N_2, M_1, M_2 are of order *N*.

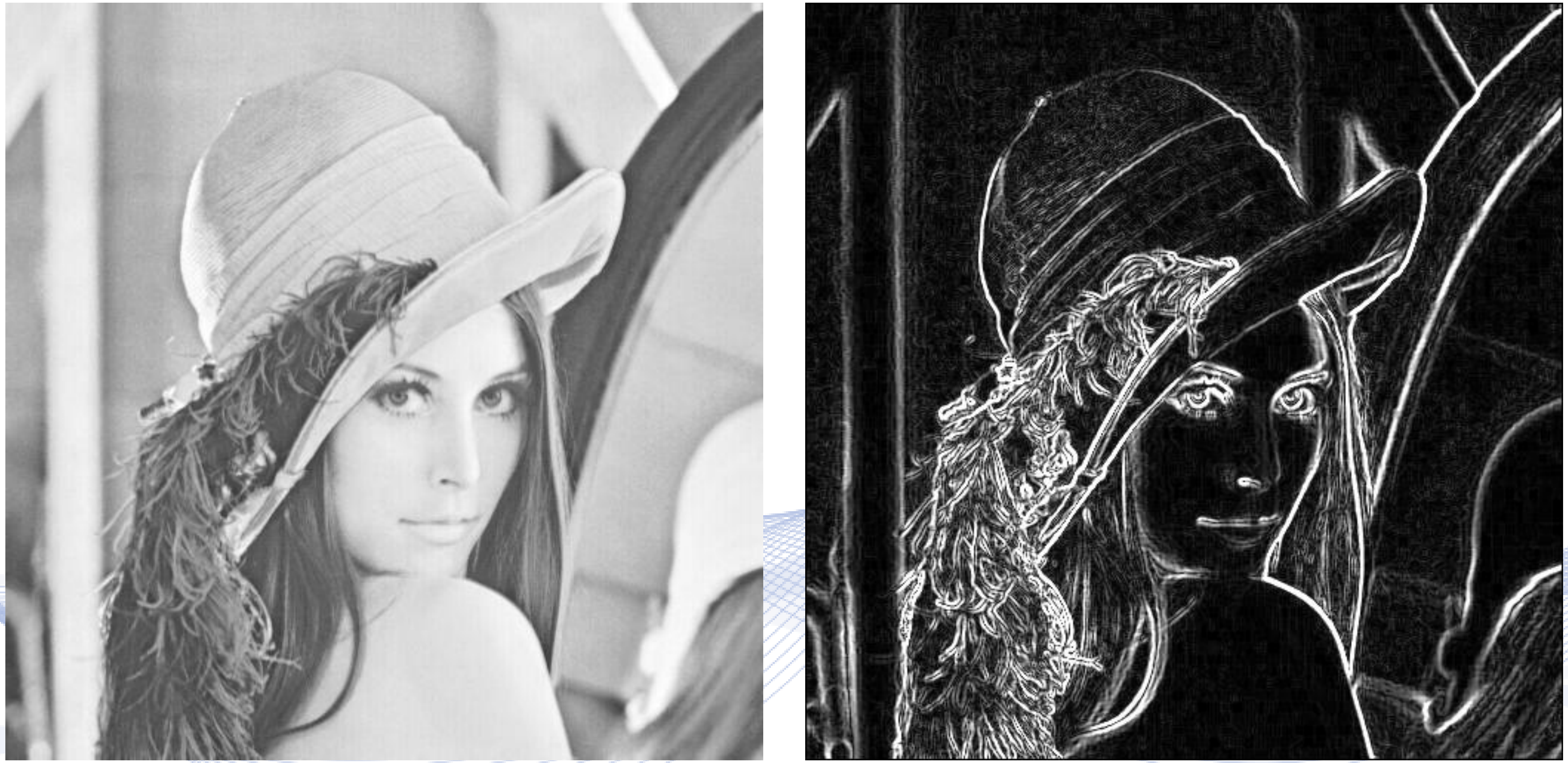
2D Discrete Systems

A 2D FIR filter output can also be described as inner product:

$$y(n_1,n_2) = \sum_{k_1=0}^{M_1-1} \sum_{k_2=0}^{M_2-1} h(k_1,k_2)\, x(k_1-n_1,k_2-n_2) = \mathbf{h}^T\mathbf{x}(n_1,n_2).$$

- $\mathbf{h} = [h(0,0),\ldots,h(M_1-1,M_2-1)]^T$: template image vector.
- $\mathbf{x}(n_1,n_2) = [x(n_1,n_2),\ldots,x(n_1-M_1+1,n_2-M_2+1)]^T$: local neighborhood (window) image vector.
- GPU computing and fast linear algebra libraries (e.g., cuBLAS) can be used for 2D convolution and correlation computations.

2D Discrete Systems



IIR Edge Detector output.

2D linear correlation

2D ***correlation*** of template image h and input image x (inner product):

$$r_{hx}(n_1,n_2)=\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}h(k_1,k_2)\,x(n_1+k_1,n_2+k_2)=\mathbf{h}^T\mathbf{x}(n_1,n_2).$$

- $\mathbf{h}=[h(0,0),\dots,h(N_1-1,N_2-1)]^T$: template image vector.
- $\mathbf{x}(n_1,n_2)=[x(n_1,n_2),\dots,x(n_1+N_1-1,n_2+N_2-1)]^T$: local neighborhood (window) image vector.

2D linear and cyclic convolutions

- Two-dimensional linear convolution with convolutional kernel h of size $M_1 \times M_2$ is given by:

$$y(k_1,k_2) = h(k_1,k_2) ** x(k_1,k_2) = \sum_{i_1=0}^{M_1-1}\sum_{i_2=0}^{M_2-1} h(i_1,i_2)x(k_1-i_1,k_2-i_2).$$

- Its two-dimensional cyclic convolution counterpart of support $N_1 \times N_2$ is defined as:

$$y(k_1,k_2) = h(k_1,k_2) \circledast\circledast\, x(k_1,k_2) = \sum_{i_1=0}^{N_1-1}\sum_{i_2}^{N_2-1} h(i_1,i_2)x\left((k_1-i_1)_{N_1},(k_2-i_2)_{N_2}\right).$$

2D Discrete Fourier Transform

$$X(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1,n_2) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2}$$

$$x(n_1,n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1,k_2)\, W_{N_1}^{-n_1 k_1} W_{N_2}^{-n_2 k_2}.$$

- Complex roots of unity:

$$W_{N_i} = \exp\left(-i \frac{2\pi}{N_i}\right), \qquad i = 1,2.$$

2D Discrete Fourier Transform

- Cyclic spatial translation (shift):

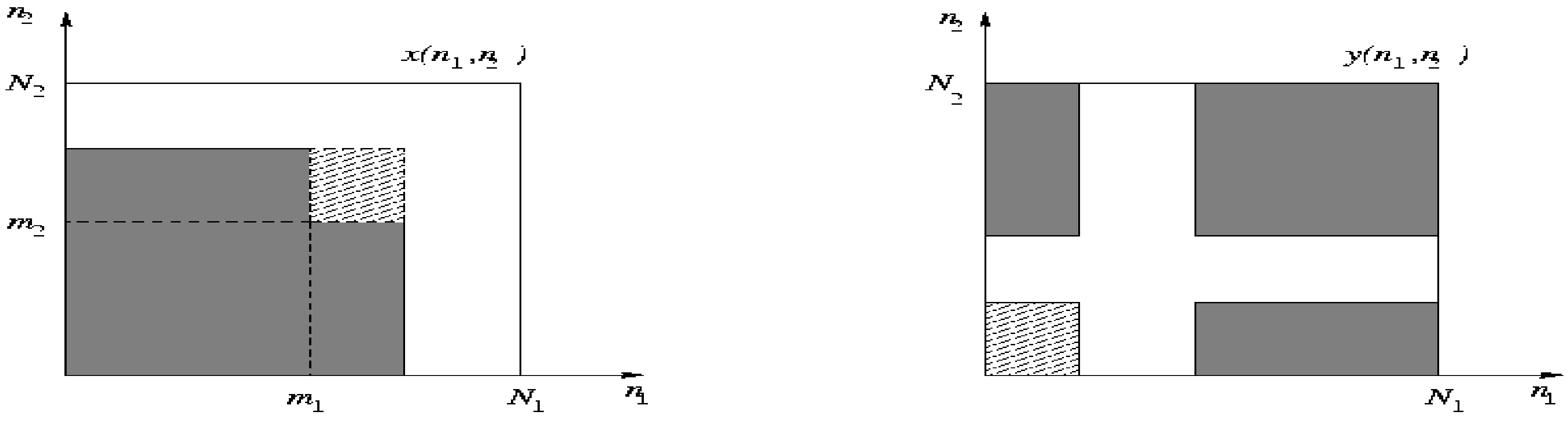
$$y(n_1,n_2) = x(((n_1-m_1))_{N_1},((n_2-m_2))_{N_2}) \leftrightarrow$$
$$Y(k_1,k_2) = W_{N_1}^{m_1k_1}W_{N_2}^{m_2k_2}X(k_1,k_2),$$

$$((n))_N \triangleq n \mod N.$$

- Frequency shift:

$$x(n_1,n_2) = W_{N_1}^{-n_1l_1}W_{N_2}^{-n_2l_2}w(n_1,n_2) \leftrightarrow$$
$$X(k_1,k_2) = X(((k_1-l_1))_{N_1},((k_2-l_2))_{N_2})$$

2D Discrete Fourier Transform



Circular shift of a 2D sequence.

2D Discrete Fourier Transform

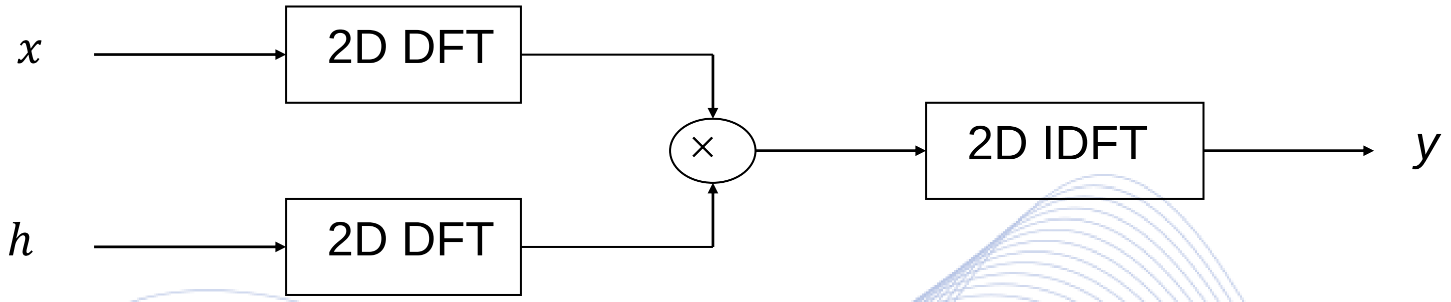
- Cyclic Convolution Theorem:

$$y(n_1,n_2) = x(n_1,n_2) \circledast\circledast\, h(n_1,n_2),$$
$$Y(k_1,k_2) = X(k_1,k_2)H(k_1,k_2).$$

- Cyclic Correlation:

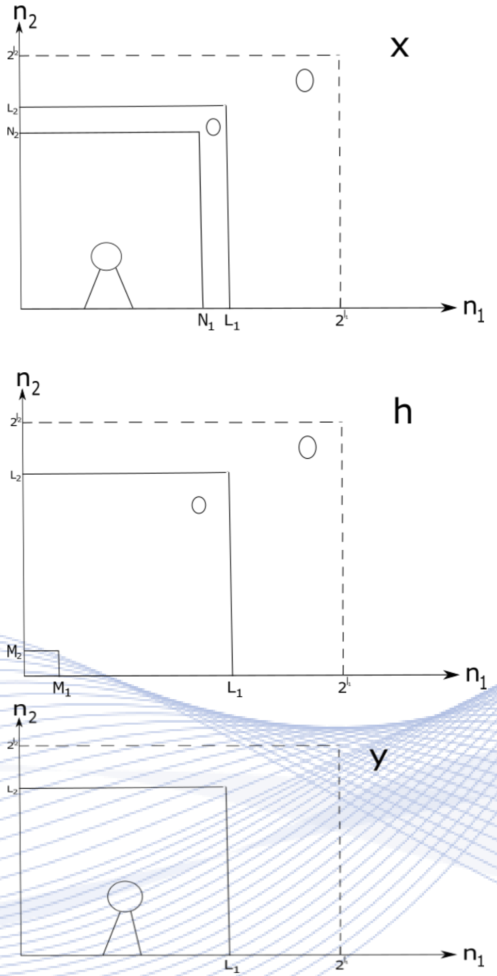
$$r_{hx}(n_1,n_2) = h(n_1,n_2) \circledast\circledast\, x(-n_1,-n_2),$$
$$R_{hx}(k_1,k_2) = H^*(k_1,k_2)X(k_1,k_2).$$

2D Cyclic Convolution Calculation with DFT



2D convolution calculation using the DFTs.

2D Linear and Circular Convolution



Zero padding for embedding a 2D linear convolution to a cyclic one.

2D Linear Convolution with DFT

- Compute the $N_1 \times N_2$ 2D DFTs of $x_p(n_1, n_2)$ and $h_p(n_1, n_2)$;
- Compute $Y_p(k_1, k_2)$ as the product of $X_p(k_1, k_2)$ and $H_p(k_1, k_2)$;
- Compute $y_p(n_1, n_2)$ as the inverse 2D DFT of $Y_p(k_1, k_2)$;
- The result is the region $[0, L_1) \times [0, L_2)$ of $y_p(n_1, n_2)$.

- 2D DFTs are calculated fast through ***2D Fast Fourier Transform*** (***FFT***) algorithms.
- Typically, 2D DFT length is chosen to be a power of 2:

$$L_i = 2^{l_i} \geq N_i + M_i - 1, \qquad i = 1, 2.$$

Convolutions using 2D FFT

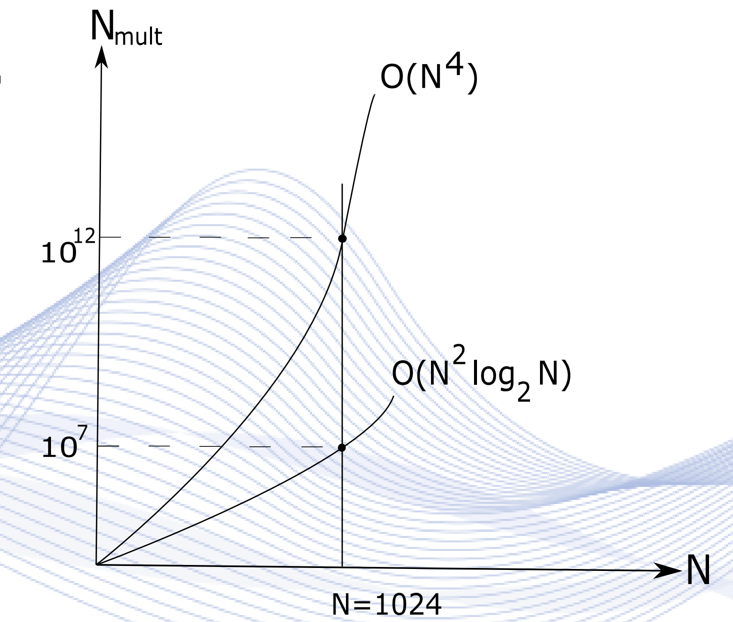
- Memory requirements are $\times 8$ for direct computation and $\times 16$ using the FFT.

- Direct approach is faster for a small filter $M_1 \times M_2$ when:

$$M_1 M_2 < 6\log_2(N_1 N_2) + 4.$$

Convolutions using 2D FFT

- For larger filters (close to the image size), computational complexity is:
  - $O(kN^4)$ for the direct method.
  - $O(kN^2 \log_2 N)$ using 2D FFT.



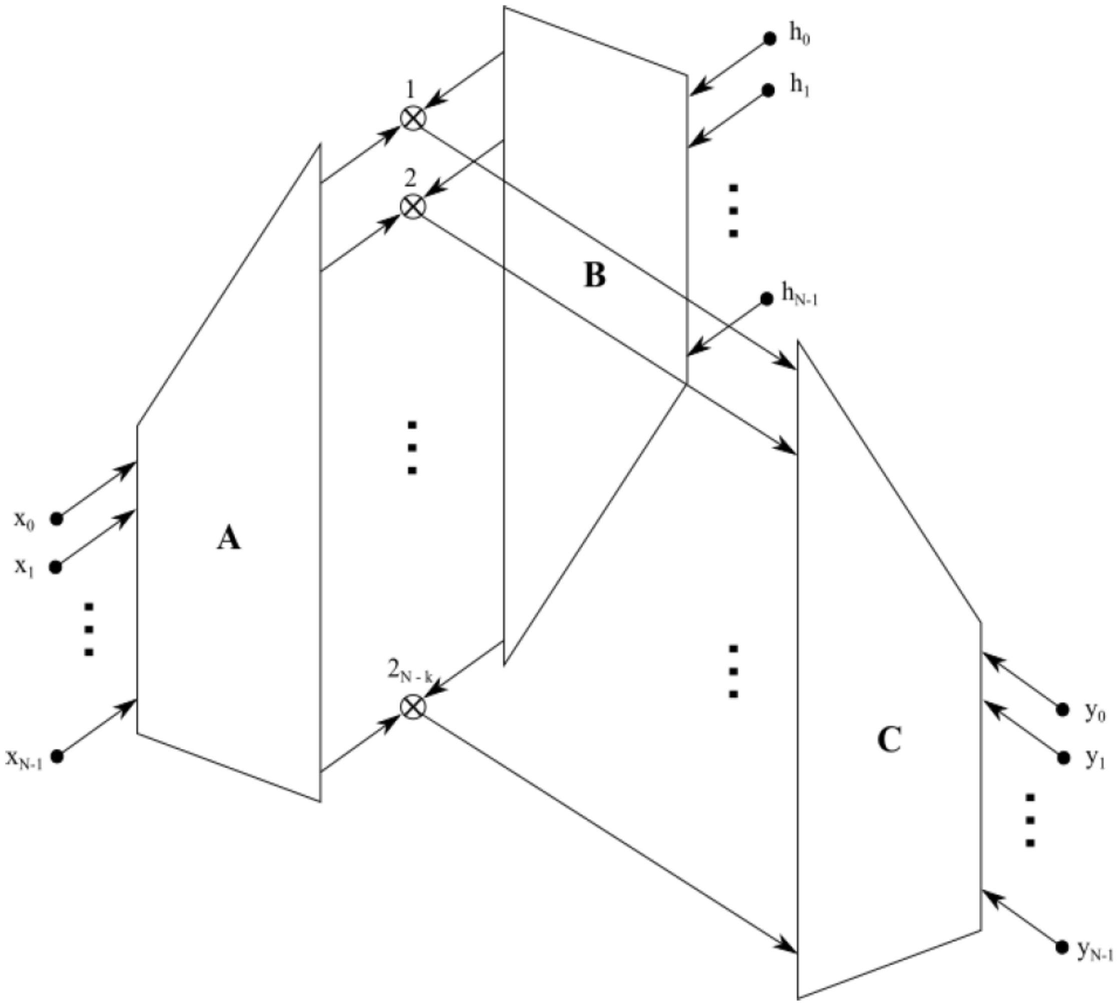
Computational complexity of 2D FIR filters.

Winograd 2D cyclic convolutions

- Winograd 2D convolution algorithms or fast 2D filtering:

$$\mathbf{y} = \mathbf{C}(\mathbf{Ax} \otimes \mathbf{Bh}).$$

- GEneral Matrix Multiplication (GEMM) BLAS or cuBLAS routines can be used.



Winograd 2D cyclic convolutions

- Alternative Winograd algorithm formulation:
$$\mathbf{y} = \mathbf{R}\mathbf{B}^T(\mathbf{A}\mathbf{x} \otimes \mathbf{C}^T\mathbf{R}\mathbf{h}).$$
- Matrices $\mathbf{A}, \mathbf{B}$ typically have elements $0, +1, -1$.
- Multiplications $\mathbf{C}^T\mathbf{R}\mathbf{h}$, $\mathbf{R}\mathbf{B}^T\mathbf{y}'$ are done only by additions/subtractions.
- $\mathbf{R}$ is an $N \times N$ permutation matrix.
- $\mathbf{R}\mathbf{h}$ can be precomputed.
- ***It has theoretically minimal computational complexity***.

Winograd 3×3 cyclic convolution

- 2D 3×3 cyclic convolution definition as 2D polynomial product:

$$Y(z_1, z_2) = H(z_1, z_2)X(z_1, z_2) \mod (z_1{}^3 - 1), (z_2{}^3 - 1),$$

where:

$$X(z_1, z_2) = x_{00} + x_{01}z_2 + x_{02}z_2{}^2 + x_{10}z_1 + x_{11}z_1z_2 \\ + x_{12}z_1z_2{}^2 + x_{20}z_1{}^2 + x_{21}z_1{}^2z_2 + x_{22}z_1{}^2z_2{}^2,$$

$$H(z_1, z_2) = h_{00} + h_{01}z_2 + h_{02}z_2{}^2 + h_{10}z_1 + h_{11}z_1z_2 \\ + h_{12}z_1z_2{}^2 + h_{20}z_1{}^2 + h_{21}z_1{}^2z_2 + h_{22}z_1{}^2z_2{}^2.$$

Winograd 3×3 cyclic convolution

Factorization of:

$$z^3 - 1 = (z-1)(z^2+z+1)$$

can be used to decompose $X(z_1, z_2)$ as follows:

- $X_1(z_1, z_2) = X(z_1, z_2) \bmod (z_1 - 1), (z_1 - 1),$
- $X_2(z_1, z_2) = X(z_1, z_2) \bmod (z_1 - 1), ({z_2}^2 + z_2 + 1),$
- $X_3(z_1, z_2) = X(z_1, z_2) \bmod (z_2 - 1), ({z_1}^2 + z_1 + 1),$
- $X_4(z_1, z_2)$
  $= X(z_1, z_2) \bmod ({z_1}^2 + z_1 + 1), ({z_2}^2 + z_2 + 1).$

Artificial Intelligence & Information Analysis Lab

Winograd 3×3 cyclic convolution

$$
\mathbf{A} = \mathbf{B} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1 \\
0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 \\
1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 \\
0 & 0 & 0 & 1 & 1 & 1 & -1 & -1 & -1 \\
1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 \\
1 & 0 & -1 & 0 & -1 & 1 & -1 & 1 & 0 \\
0 & 1 & -1 & 1 & -1 & 0 & -1 & 0 & 1 \\
1 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & -1 \\
1 & 0 & -1 & -1 & 1 & 0 & 0 & -1 & 1 \\
0 & 1 & -1 & -1 & 0 & 1 & 1 & -1 & 0 \\
1 & -1 & 0 & 0 & 1 & -1 & -1 & 0 & 1
\end{bmatrix}
$$

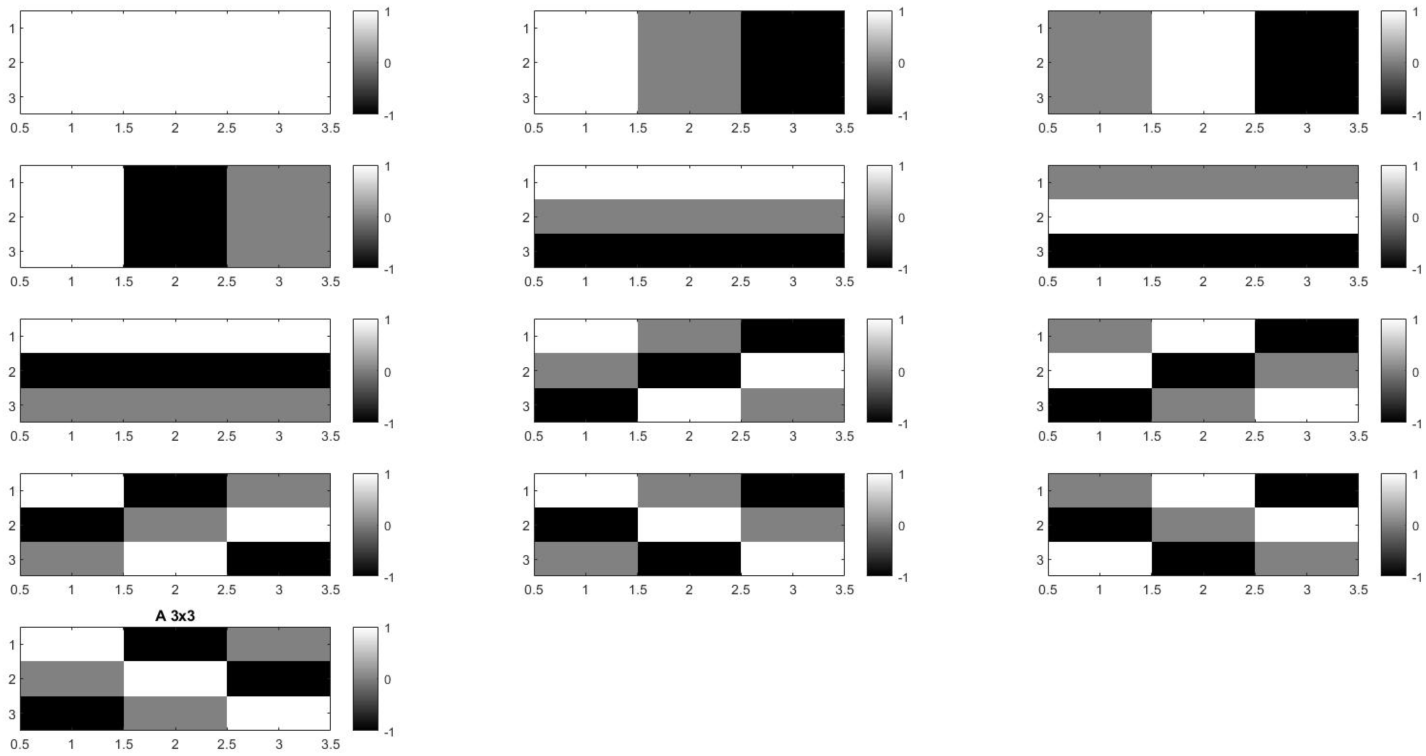
Arrays $\mathbf{A}, \mathbf{B}$ of Winograd 3×3 cyclic convolution.

Winograd 3×3 cyclic convolution

$$\mathbf{C} = \frac{1}{27}\begin{bmatrix} 3 & 3 & -6 & 3 & 3 & -6 & 3 & 3 & -6 & 3 & 3 & -6 & 3 \\ 3 & 3 & 3 & -6 & 3 & -6 & 3 & 3 & 3 & -6 & 3 & 3 & -6 \\ 3 & -6 & 3 & 3 & 3 & -6 & 3 & -6 & 3 & 3 & -6 & 3 & 3 \\ 3 & 3 & -6 & 3 & 3 & 3 & -6 & 3 & 3 & -6 & -6 & 3 & 3 \\ 3 & 3 & 3 & -6 & 3 & 3 & -6 & -6 & 3 & 3 & 3 & -6 & 3 \\ 3 & -6 & 3 & 3 & 3 & 3 & -6 & 3 & -6 & 3 & 3 & 3 & -6 \\ 3 & 3 & -6 & 3 & -6 & 3 & 3 & -6 & 3 & 3 & 3 & 3 & -6 \\ 3 & 3 & 3 & -6 & -6 & 3 & 3 & 3 & -6 & 3 & -6 & 3 & 3 \\ 3 & -6 & 3 & 3 & -6 & 3 & 3 & 3 & 3 & -6 & 3 & -6 & 3 \end{bmatrix}$$

Array $\mathbf{C}$ of Winograd 3×3 Cyclic Convolution.

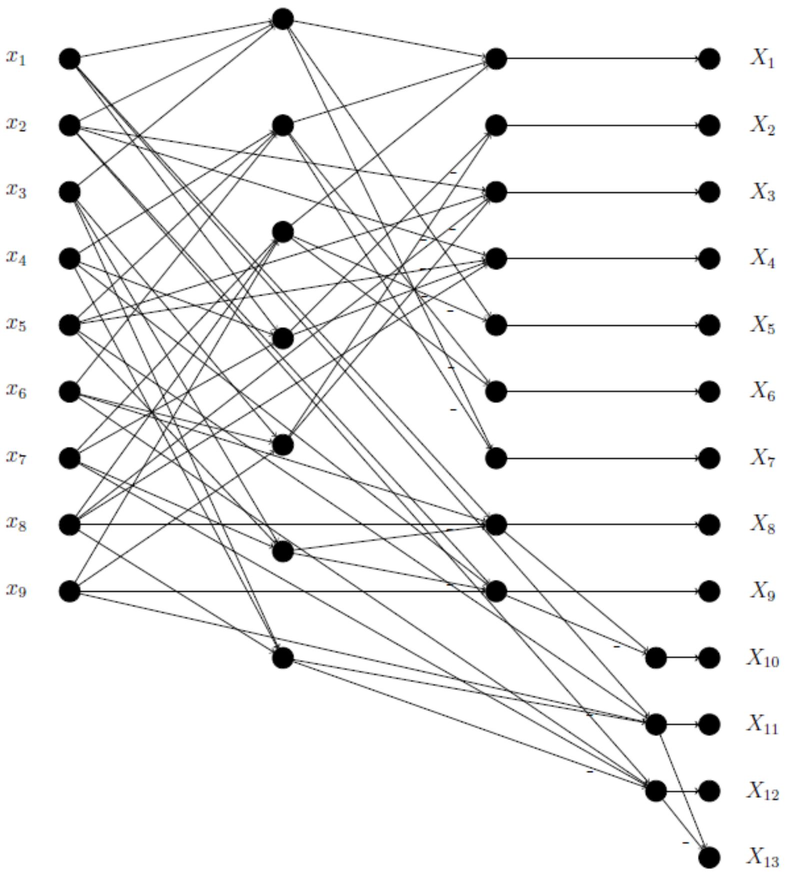
Winograd 3×3 cyclic convolution



Visualization of Array $\mathbf{A}$ of Winograd 3×3 cyclic convolution.

Winograd 3×3 cyclic convolution

- The original 117+13 additions were reduced to 44 additions (***only 36,6% of the original number***).
- Only 9 out of the 13 multiplications are needed because only the last one output element is kept.



Reducing additions in $\mathbf{Ax}$ product of Winograd 3×3 cyclic convolution.

Winograd 4×4 cyclic convolution

$$
\mathbf{A} = \mathbf{B} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\
0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \\
1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\
1 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 \\
0 & 1 & 0 & -1 & 1 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & -1 & 0 & 1 & 0 \\
1 & -1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 \\
1 & 0 & -1 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 1 \\
0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & -1 & 0 \\
1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & 1 & 1
\end{bmatrix},
$$

Arrays $\mathbf{A}, \mathbf{B}$ of Winograd 4×4 cyclic convolution.

Winograd 4×4 cyclic convolution

$$
\mathbf{C} = \frac{1}{16}\begin{bmatrix}
1 & 1 & 2 & -2 & 0 & 1 & 1 & 2 & -2 & 0 & 2 & -2 & 0 & 2 & -2 & 0 & 2 & -2 & 0 & 2 & -2 & 0 \\
1 & -1 & 2 & 2 & -2 & 1 & -1 & 2 & 2 & -2 & 2 & -2 & 0 & -2 & 2 & 0 & 2 & 2 & -2 & 2 & 2 & -2 \\
1 & 1 & -2 & 2 & 0 & 1 & 1 & -2 & 2 & 0 & 2 & -2 & 0 & 2 & -2 & 0 & -2 & 2 & 0 & -2 & 2 & 0 \\
1 & -1 & -2 & -2 & 2 & 1 & -1 & -2 & -2 & 2 & 2 & -2 & 0 & -2 & 2 & 0 & -2 & -2 & 2 & -2 & -2 & 2 \\
1 & 1 & 2 & -2 & 0 & -1 & -1 & -2 & 2 & 0 & 2 & 2 & -2 & 2 & 2 & -2 & 2 & 2 & -2 & -2 & -2 & 2 \\
1 & -1 & 2 & 2 & -2 & -1 & 1 & -2 & -2 & 2 & 2 & 2 & -2 & -2 & -2 & 2 & -2 & 2 & 0 & 2 & -2 & 0 \\
1 & 1 & -2 & 2 & 0 & -1 & -1 & 2 & -2 & 0 & 2 & 2 & -2 & 2 & 2 & -2 & -2 & -2 & 2 & 2 & 2 & -2 \\
1 & -1 & -2 & -2 & 2 & -1 & 1 & 2 & 2 & -2 & 2 & 2 & -2 & -2 & -2 & 2 & 2 & -2 & 0 & -2 & 2 & 0 \\
1 & 1 & 2 & -2 & 0 & 1 & 1 & 2 & -2 & 0 & -2 & 2 & 0 & -2 & 2 & 0 & -2 & 2 & 0 & -2 & 2 & 0 \\
1 & -1 & 2 & 2 & -2 & 1 & -1 & 2 & 2 & -2 & -2 & 2 & 0 & 2 & -2 & 0 & -2 & -2 & 2 & -2 & -2 & 2 \\
1 & 1 & -2 & 2 & 0 & 1 & 1 & -2 & 2 & 0 & -2 & 2 & 0 & -2 & 2 & 0 & 2 & -2 & 0 & 2 & -2 & 0 \\
1 & -1 & -2 & -2 & 2 & 1 & -1 & -2 & -2 & 2 & -2 & 2 & 0 & 2 & -2 & 0 & 2 & 2 & -2 & 2 & 2 & -2 \\
1 & 1 & 2 & -2 & 0 & -1 & -1 & -2 & 2 & 0 & -2 & -2 & 2 & -2 & -2 & 2 & -2 & -2 & 2 & 2 & 2 & -2 \\
1 & -1 & 2 & 2 & -2 & -1 & 1 & -2 & -2 & 2 & -2 & -2 & 2 & 2 & 2 & -2 & 2 & -2 & 0 & -2 & 2 & 0 \\
1 & 1 & -2 & 2 & 0 & -1 & -1 & 2 & -2 & 0 & -2 & -2 & 2 & -2 & -2 & 2 & 2 & 2 & -2 & -2 & -2 & 2 \\
1 & -1 & -2 & -2 & 2 & -1 & 1 & 2 & 2 & -2 & -2 & -2 & 2 & 2 & 2 & -2 & -2 & 2 & 0 & 2 & -2 & 0
\end{bmatrix}.
$$

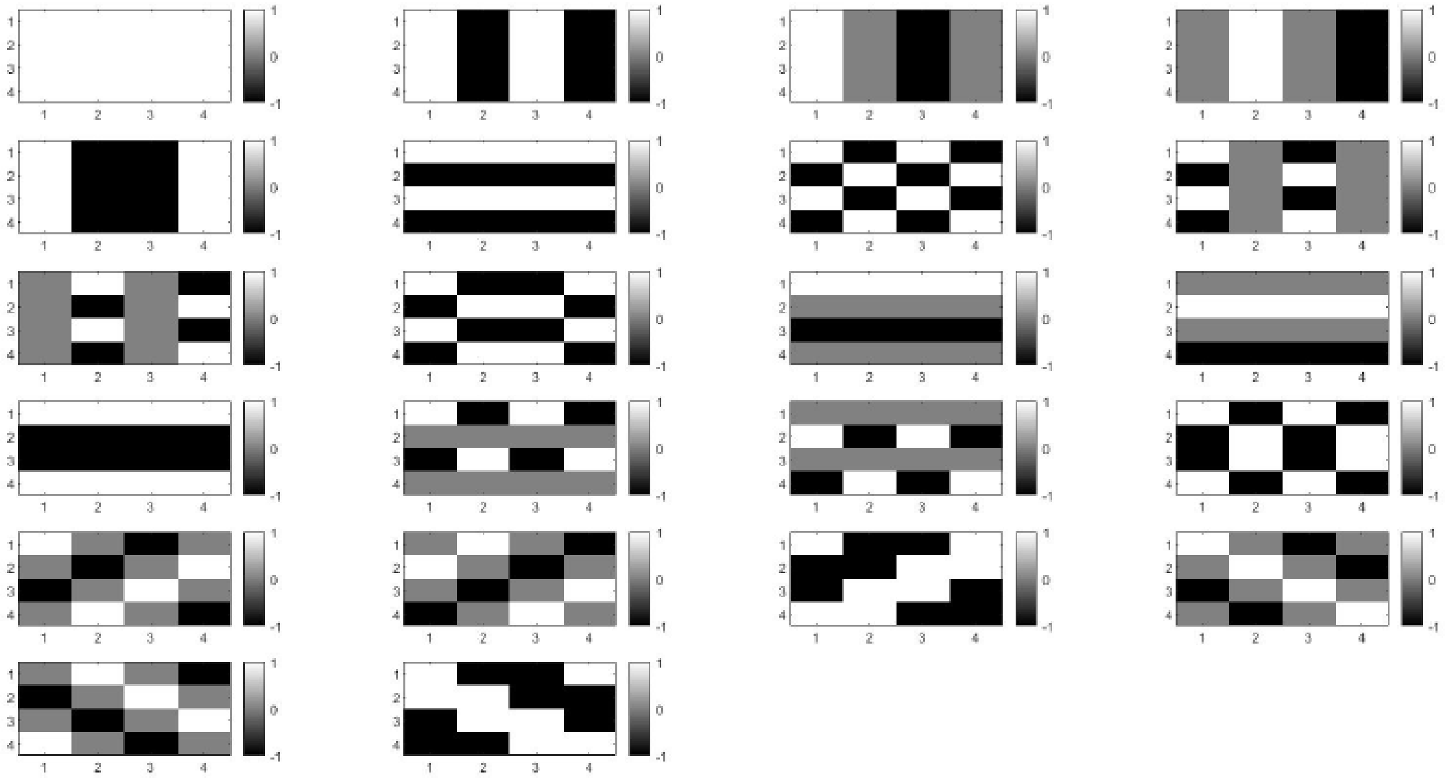
Array $\mathbf{C}$ of Winograd 4×4 cyclic convolution.

Winograd 4×4 cyclic convolution

- Reduction of the total number of additions of a Winograd 4×4 Cyclic Convolution to 26.6% of their original number, using precalculated sums.

	Precalc. Additions	Final Additions	Total	Naive approx.
$\mathbf{Ax}$	40	36	76	352
$\mathbf{RA}^T(\mathbf{Ax} \otimes \mathbf{C}^T\mathbf{Rh})$	17	24	41	88
Total additions	57	60	**117**	**440**

Winograd 4×4 cyclic convolution

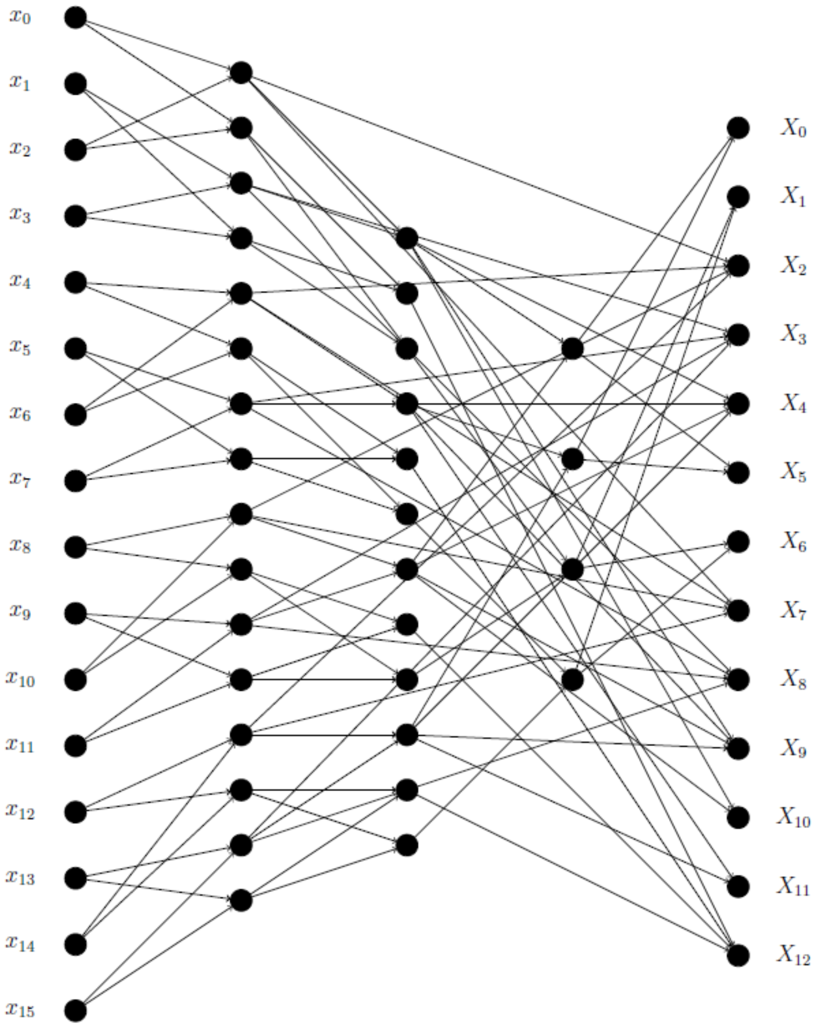


Visualization of Array $\mathbf{A}$ of Winograd 4×4 cyclic convolution.

Artificial Intelligence & Information Analysis Lab

Winograd 4×4 cyclic convolution

- Reducing additions in $\mathbf{Ax}$ product of Winograd 3×3 cyclic convolution.
- For illustration simplicity, only 13 of 22 $\mathbf{Ax}$ product entries are shown.



Artificial Intelligence & Information Analysis Lab

Nested convolutions

- Winograd algorithms exist for relatively short convolution lengths.
- Use of efficient short-length convolution algorithms iteratively to build long convolutions
- Does not achieve minimal multiplication complexity
- Good balance between multiplications and additions

Decomposition:

- 2D convolution : $N \times N = N_1 N_2 \times N_1 N_2$, for N_1, N_2 coprime integers $(N_1, N_2) = 1$, can be implemented using nested $N_1 \times N_1$, $N_2 \times N_2$ convolutions.

Block-based 2D convolution

2D overlap-add algorithm is based on the distributive property of convolution:

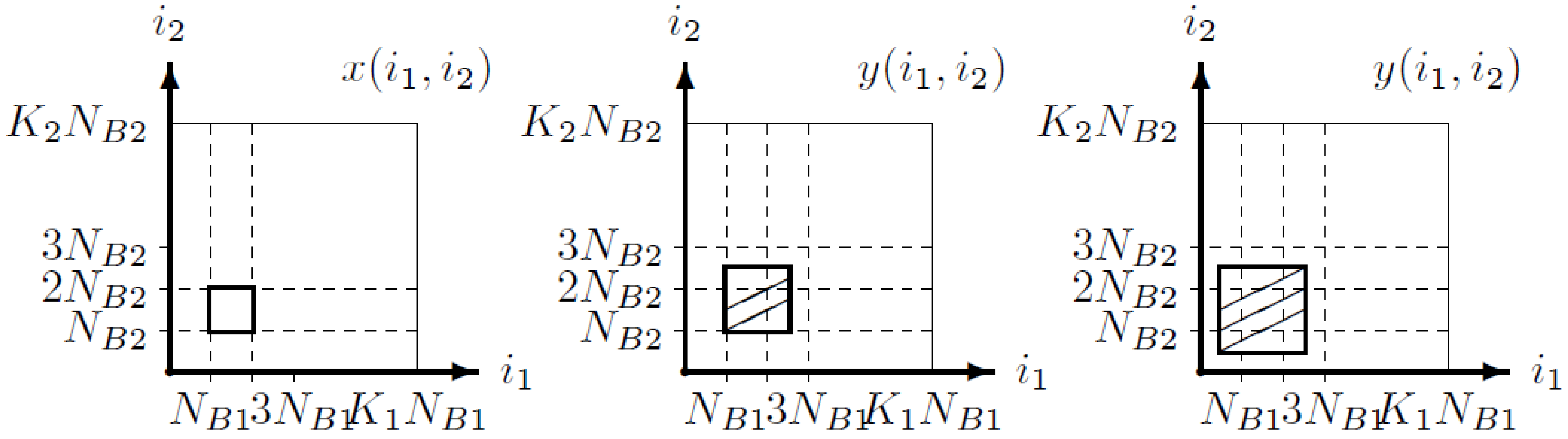
- An image $x(i_1, i_2)$ can be divided into $K_1 \times K_2$ non-overlapping subsequences, having dimensions $N_{B1} \times N_{B2}$ each:

$$x_{k_1k_2}(i_1, i_2) = \begin{cases} x(i_1, i_2) & k_1N_{B1} \le i_1 < (k_1+1)N_{B1},\ k_2N_{B2} \le i_2 < (k_2+1)N_{B2} \\ 0 & \text{otherwise.} \end{cases}$$

- The linear convolution output $y(n_1, n_2)$ is the sum of the convolution outputs produced by the input sequence blocks:

$$y(i_1, i_2) = x(i_1, i_2) ** h(i_1, i_2) = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{k_1k_2}(i_1, i_2) ** h(i_1, i_2) = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} y_{k_1k_2}(i_1, i_2).$$

Overlap-add algorithm

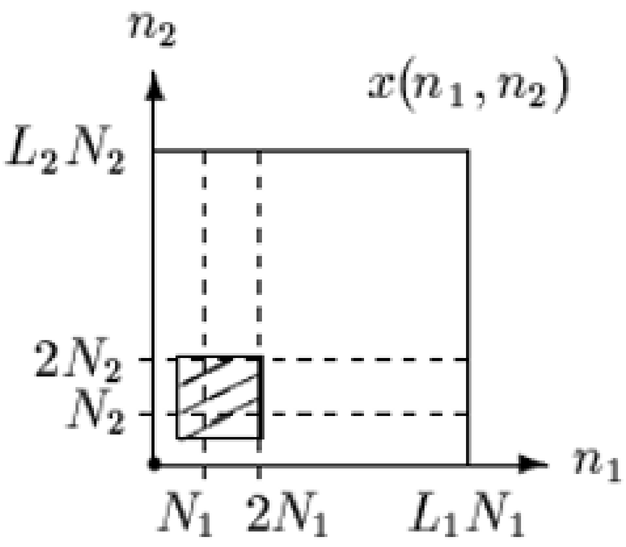


Overlap-add algorithm.

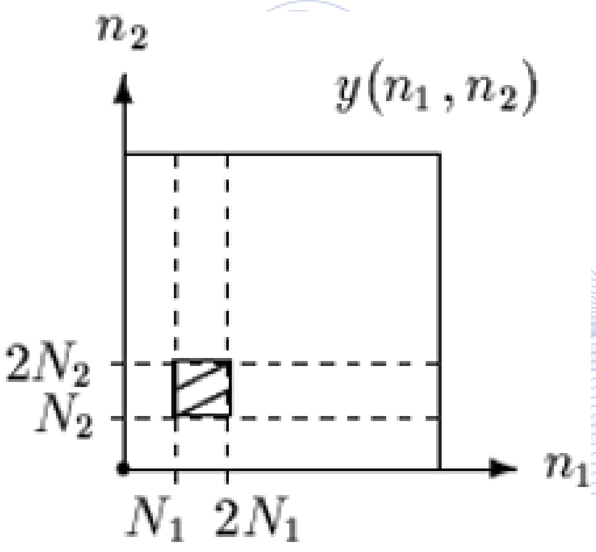
Overlap-save algorithm

- Every x_{ij} item is non-zero, therefore only the inner $N_1 \times N_2$ part is correct;
- Addition all the 'trimmed' boxes to get the output.

- Block-based algorithms can be easily parallelized.
- They are suitable for GPU computing.



(β)

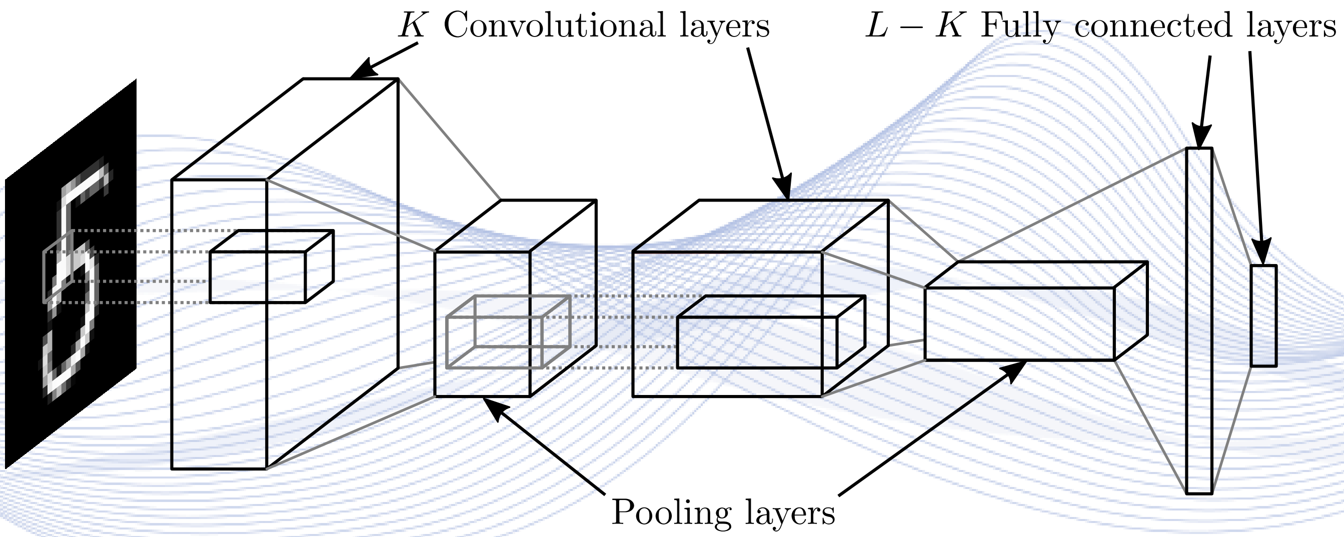


(γ)

Convolutional Neural Networks

Convolutional Neural Networks (***CNN***):

- employ image convolutions in the first layers.
- They may employ fully connected MLPs in the last layers.



Basic CNN structure.

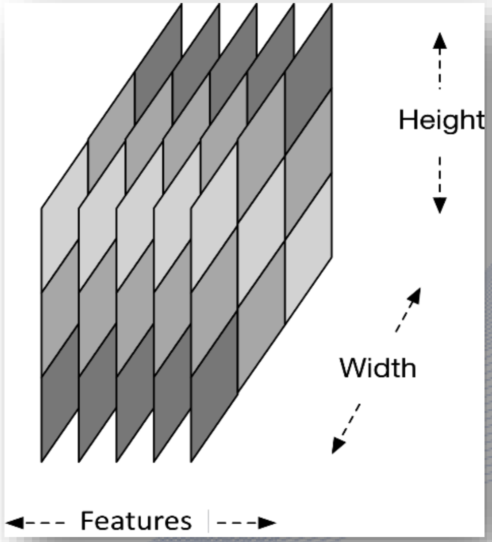
Convolutional Neural Networks

Convolutional Layers employ:

- A d_{out} -dimensional ***feature descriptor*** vector $\mathbf{z}_{ij} = [z_{ijo},\ o = 1, \ldots, d_{out}]^T$ holds all output features for a feature map location $[i, j]^T$.

- The convolution kernel is described by a ***4D tensor***:

$\mathbf{W} = [w_{k_1 k_2 r o}: \quad k_1 = 1, \ldots, M_1, k_2 = 1, \ldots, M_2, r = 1, \ldots, d_{in}, o = 1, \ldots, d_{out}] \in \mathbb{R}^{M_1 \times M_2 \times d_{in} \times d_{out}}$.



Convolutional Neural Networks

- For a convolutional layer l with an activation function $f^{(l)}(\cdot)$, multiple incoming features d_{in} and one single output feature o:

$$a_{ijo}^{(l)} = f^{(l)}\left(\sum_{r=1}^{d_{in}} \sum_{k_1=-v_1^{(l)}}^{v_1^{(l)}} \sum_{k_2=-v_2^{(l)}}^{v_2^{(l)}} w_{k_1k_2ro}^{(l)}\, a_{(i-k_1)(j-k_2)r}^{(l-1)} + b_o^{(l)}\right).$$

- The input to the first convolutional layer is a multichannel image x_{ijr}:

$$a_{ijr}^{(0)} = x_{ijr}.$$

Deep Learning Frameworks

Intel Neon

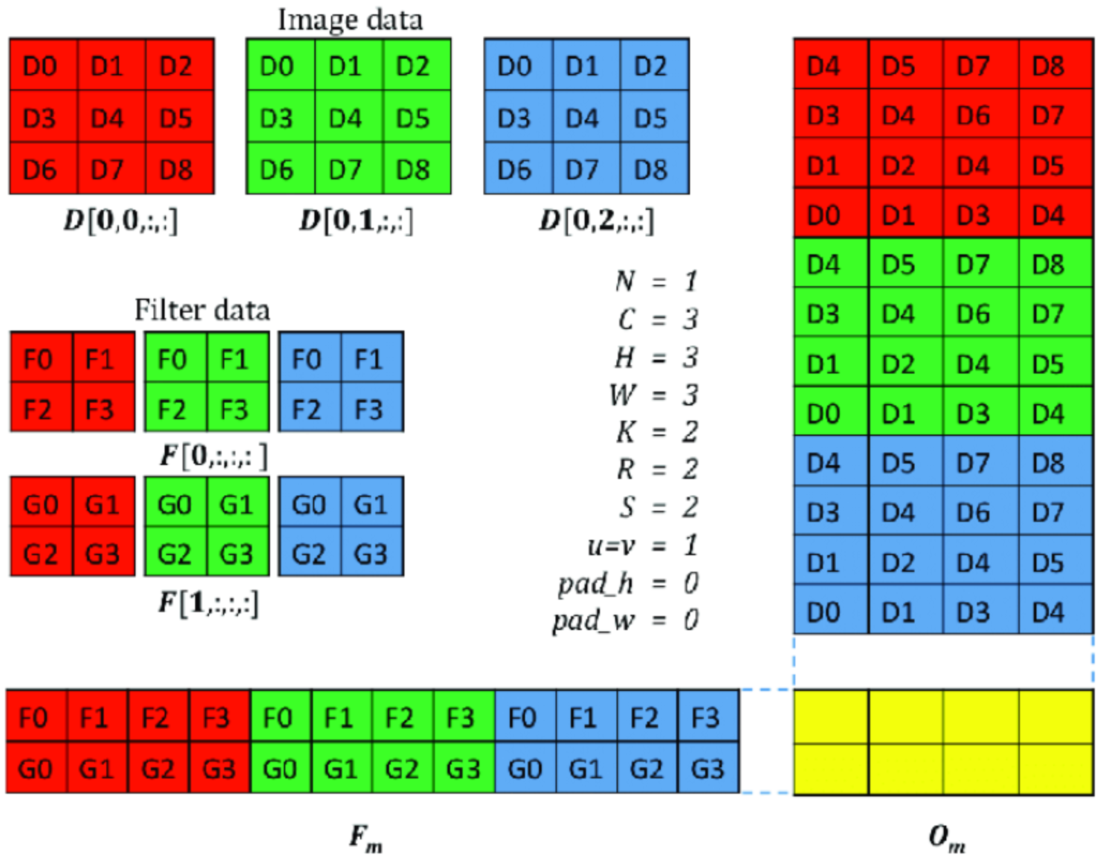
- Neon is a modern deep learning framework created by Nervana Systems.
- Implemented in Python, while Nervana Caffe framework is written in C and C++.
- Impressively fast compared to other frameworks.
- Image processing oriented (not general purpose enough).

2D Convolution Routines

cuDNN convolution algorithms:

- DIRECT
  - CUDNN-CONVOLUTION-FWD-ALGO-DIRECT;
- FFT
  - CUDNN-CONVOLUTION-FWD-ALGO-FFT;
  - CUDNN-CONVOLUTION-FWD-ALGO-FFT-TILING;
- GEMM
  - CUDNN-CONVOLUTION-FWD-ALGO-GEMM;
  - CUDNN-CONVOLUTION-FWD-ALGO-IMPLICIT-GEMM;
  - CUDNN-CONVOLUTION-FWD-ALGO-IMPLICIT-PRECOMP-GEMM;
- WINOGRAD
  - CUDNN-CONVOLUTION-FWD-ALGO-WINOGRAD;
  - CUDNN-CONVOLUTION-FWD-ALGO-WINOGRAD-NONFUSED;

2D Convolution Routines



Data transformation performed for the GEMM convolution approach.

Artificial Intelligence & Information Analysis Lab

2D Convolution Routines

Fastest cuDNN algorithm

- The preferred cuDNN algorithm ***can be chosen either by the developer or by cuDNN parameter CUDNN-CONVOLUTION-FWD-PREFER-FASTEST***, based on input and kernel size.
- GEMM algorithms (IMPLICIT GEMM in particular) are the fastest cuDNN algorithms for tested input and kernel size.
- GEMM-based algorithms transform the inputs and filter to be able to exploit high-performance matrix-matrix multiply operations.

2D Convolution Routines

cuDNN convolution parameters:

- 3×3 convolution kernel
- 512×512 pixel input image

Winograd 4×4 cyclic convolution parameters:

- ***Same image and convolution kernel size, as in cuDNN convolution.***
- Input image blocks (tiles)
  - 65536 2×2 input image tiles
- 4×4 cyclic convolution.
- Overlap-Save block-based convolution implementation.

2D Convolution Routines

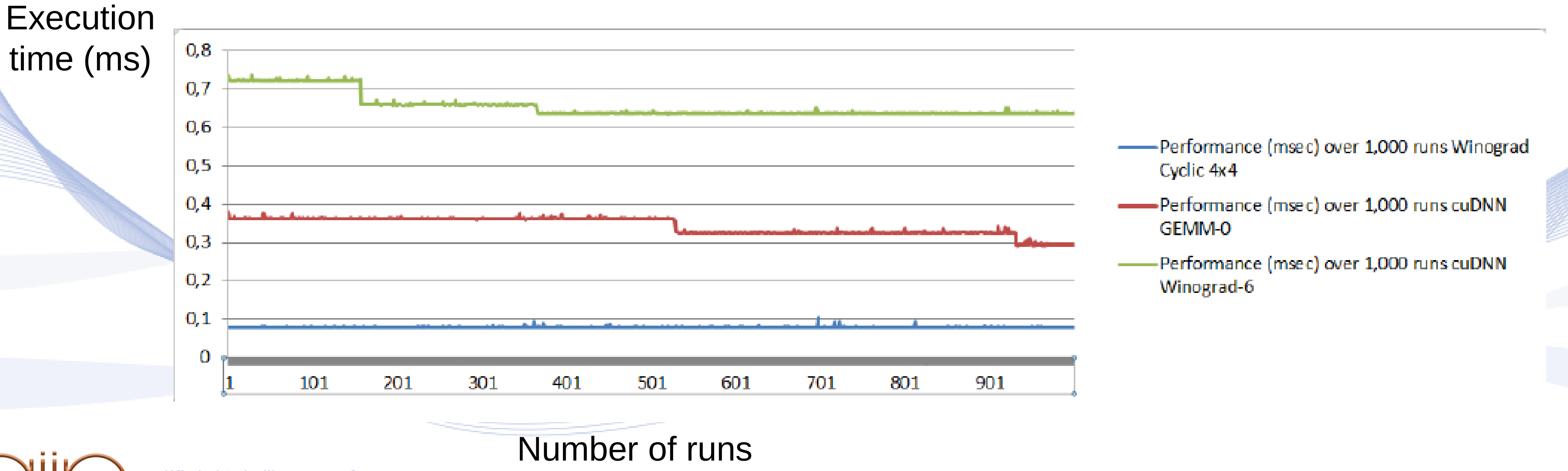
- Speed Comparisons of various 2D Convolution Routines.
- Winograd 4×4 cyclic convolution routine performs $\mathbf{4.77} \times$ ***faster*** than the faster cuDNN convolution and $\mathbf{11.33} \times$ ***faster*** than the corresponding cuDNN Winograd ***linear*** convolution routine.
- GEMM-0 is the fastest cuDNN algorithm.
- Winograd-6 is based on Linear Winograd convolution algorithm.

	SNR	Time (ms)	xSlower than Winograd Cyclic 4x4
4x4 Winograd Cyclic	142.54	0.0809	
cuDNN GEMM-0	140.75	0.3860	4.77
cuDNN GEMM-1	140.75	0.4575	5.66
cuDNN GEMM-2	140.75	0.3901	4.82
cuDNN Winograd-6	140.75	0.9168	11.33
cuDNN Winograd-7	140.75	8.8710	109.66

2D Convolution Routines

GTX1060 Visual Studio IDE implementation

- Performance Comparison on GTX1060 GPU (1,280 CUDA cores) between Winograd 4×4 cyclic convolution and cuDNN Library algorithms for over 1,000 runs.



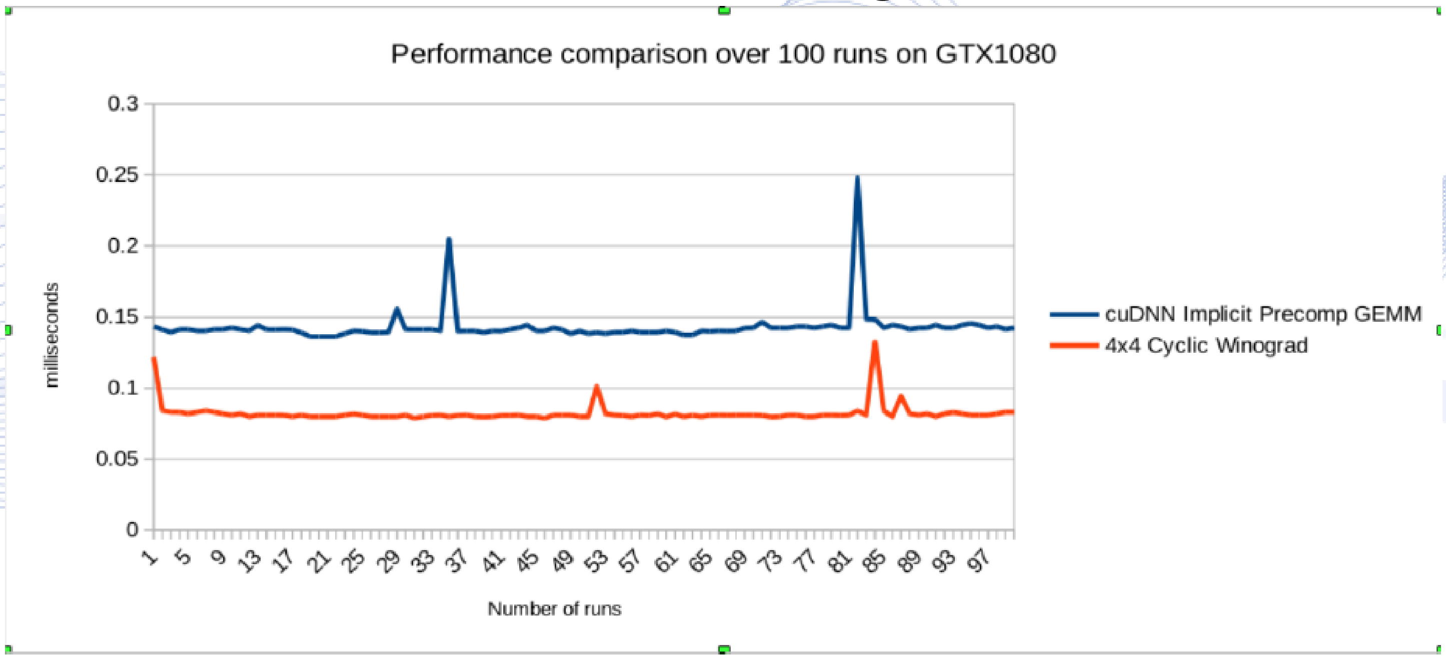
Artificial Intelligence & Information Analysis Lab

2D Convolution Routines

GTX1080 Eclipse IDE

- Performance Comparison on GTX1080 GPU (2,560 CUDA cores) over 1,000 runs between a) Winograd 4×4 cyclic convolution and b) cuDNN Library algorithm GEMM-0 which is the fastest cuDNN algorithm in this case.

Execution time (ms)



Bibliography

[PIT2021] I. Pitas, “Computer vision”, Createspace/Amazon, in press.

[PIT2017] I. Pitas, “Digital video processing and analysis” , China Machine Press, 2017 (in Chinese).

[PIT2013] I. Pitas, “Digital Video and Television” , Createspace/Amazon, 2013.

[NIK2000] N. Nikolaidis and I. Pitas, “3D Image Processing Algorithms”, J. Wiley, 2000.

[PIT2000] I. Pitas, “Digital Image Processing Algorithms and Applications”, J. Wiley, 2000.

Q & A

Thank you very much for your attention!

More material in
http://icarus.csd.auth.gr/cvml-web-lecture-series/

Contact: Prof. I. Pitas
pitas@csd.auth.gr