

인공지능 대학원 세미나

딥러닝 모델 경량화 및 최적화 기술 동향



Sogang University

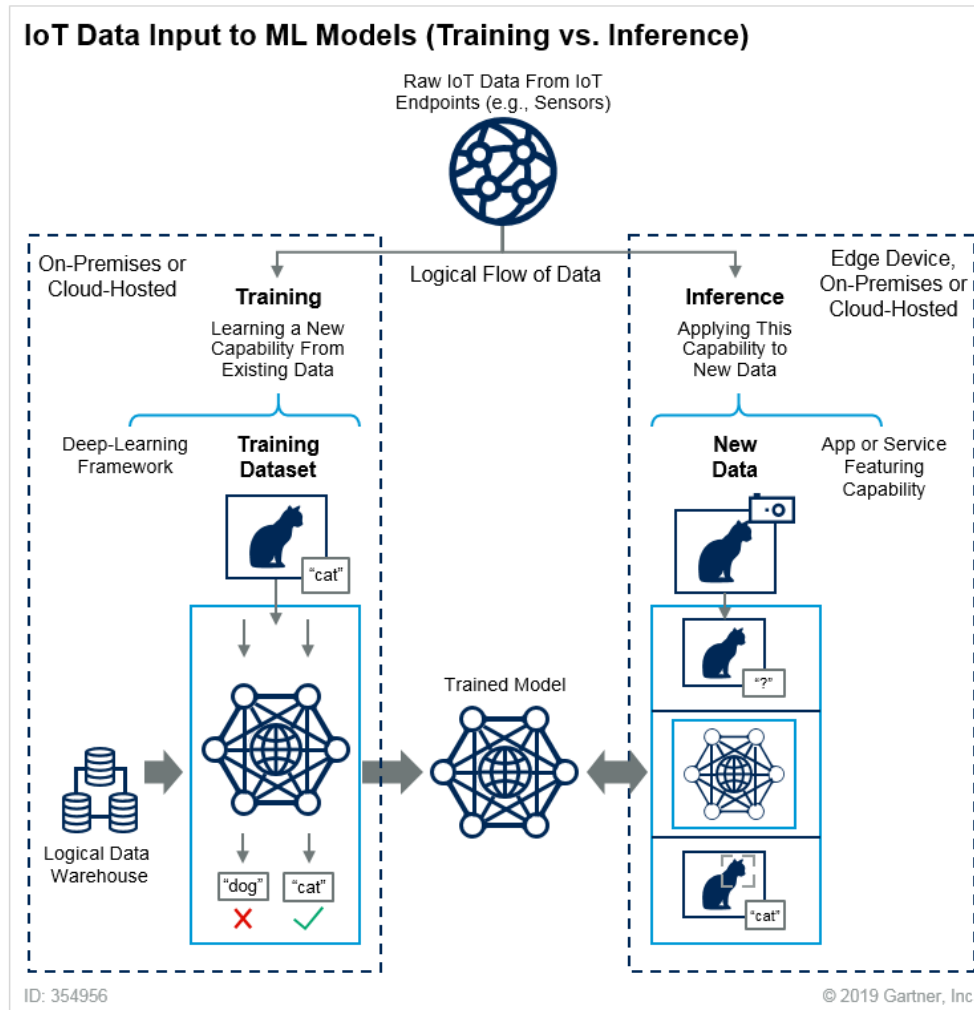
Vision & Display Systems Lab, Dept. of Electronic Engineering



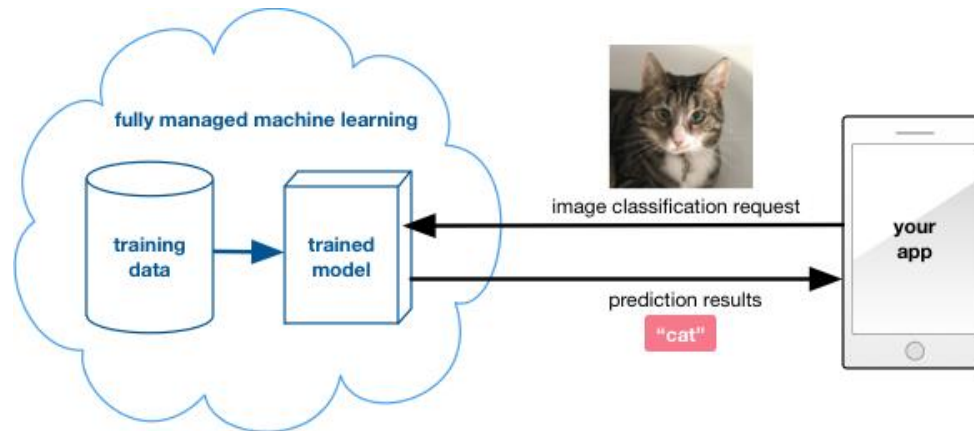
Presented By

Sukju-Kang

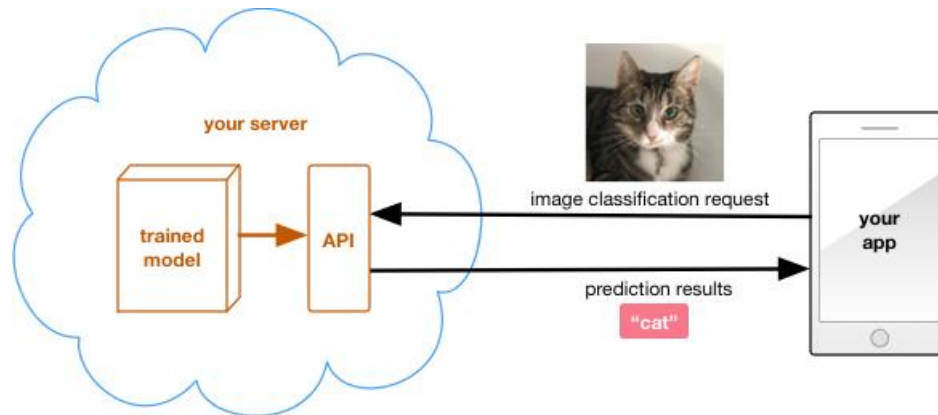
Training vs. Inference



Machine Learning on the Device or in the Cloud?



Inference in the cloud



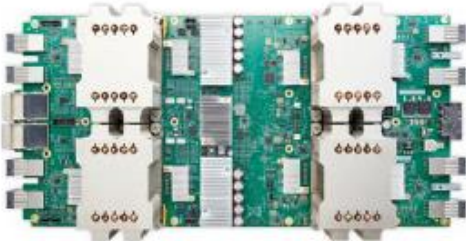
Inference on the device



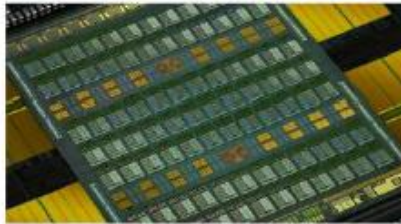
NPU are Everywhere

- With Specialized Hardware
 - From Cloud to Edge

Google Cloud TPU: 180 Tflops



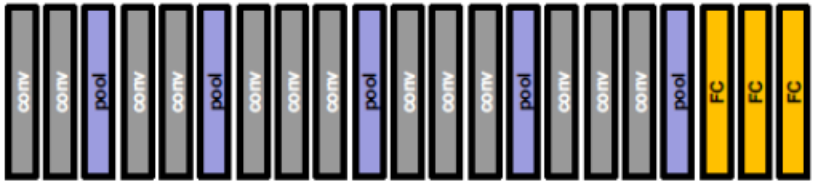
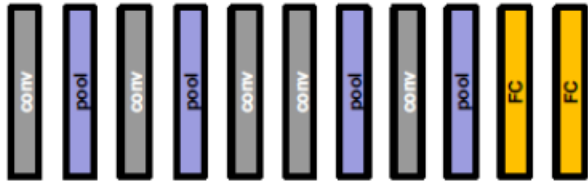
NVIDIA Volta: 100 Tflops



Apple Bionic A11: 0.6 Tflops

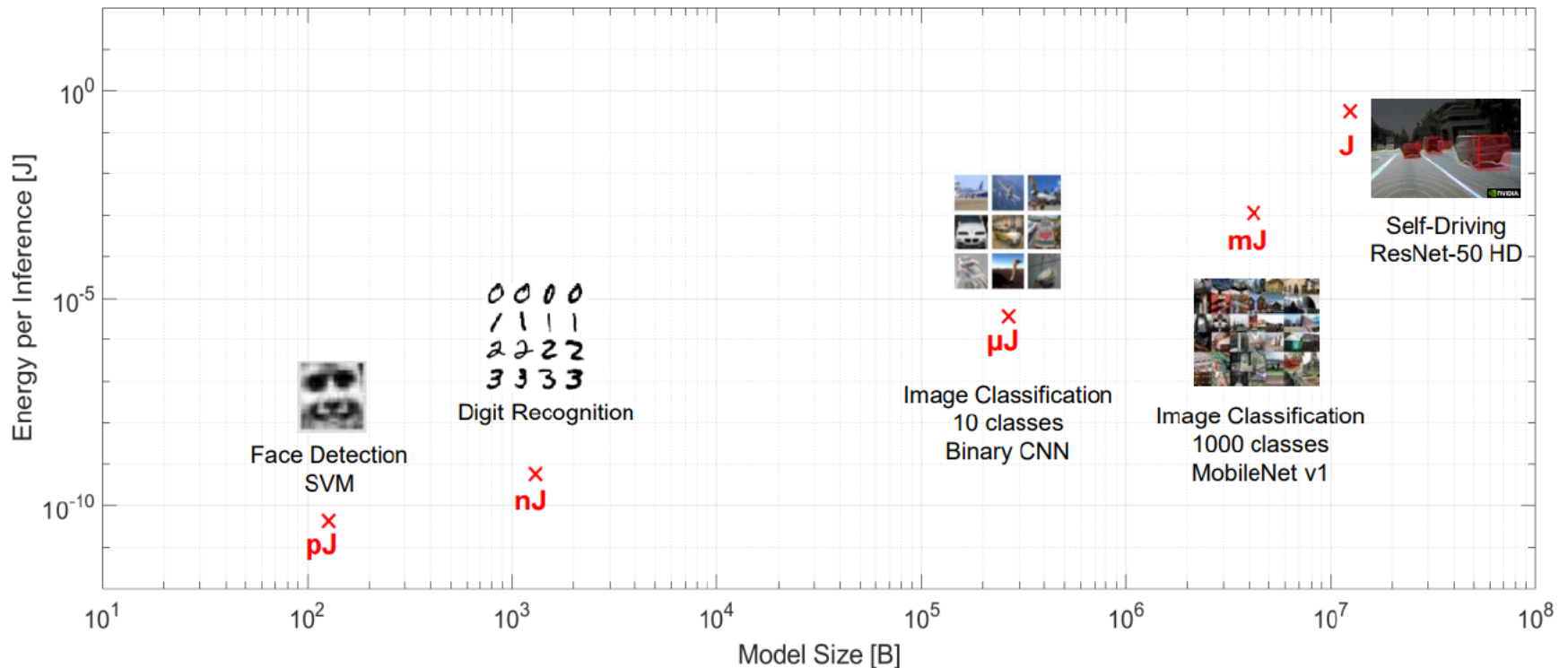


DNN Requirements on the Device or in the Cloud

Cloud-based DNN	Mobile/Embedded DNN
 <ul style="list-style-type: none"> • Cloud Intelligence • General AI <ul style="list-style-type: none"> - Very deep network • Communication latency • High Number Precision • High power / cooling cost • Large memory capacity 	 <ul style="list-style-type: none"> • On-device Intelligence • Application-specific AI <ul style="list-style-type: none"> - Specific DNN network • Real-time operation • Reduced Number Precision • Low-power consumption • Efficient memory arch.

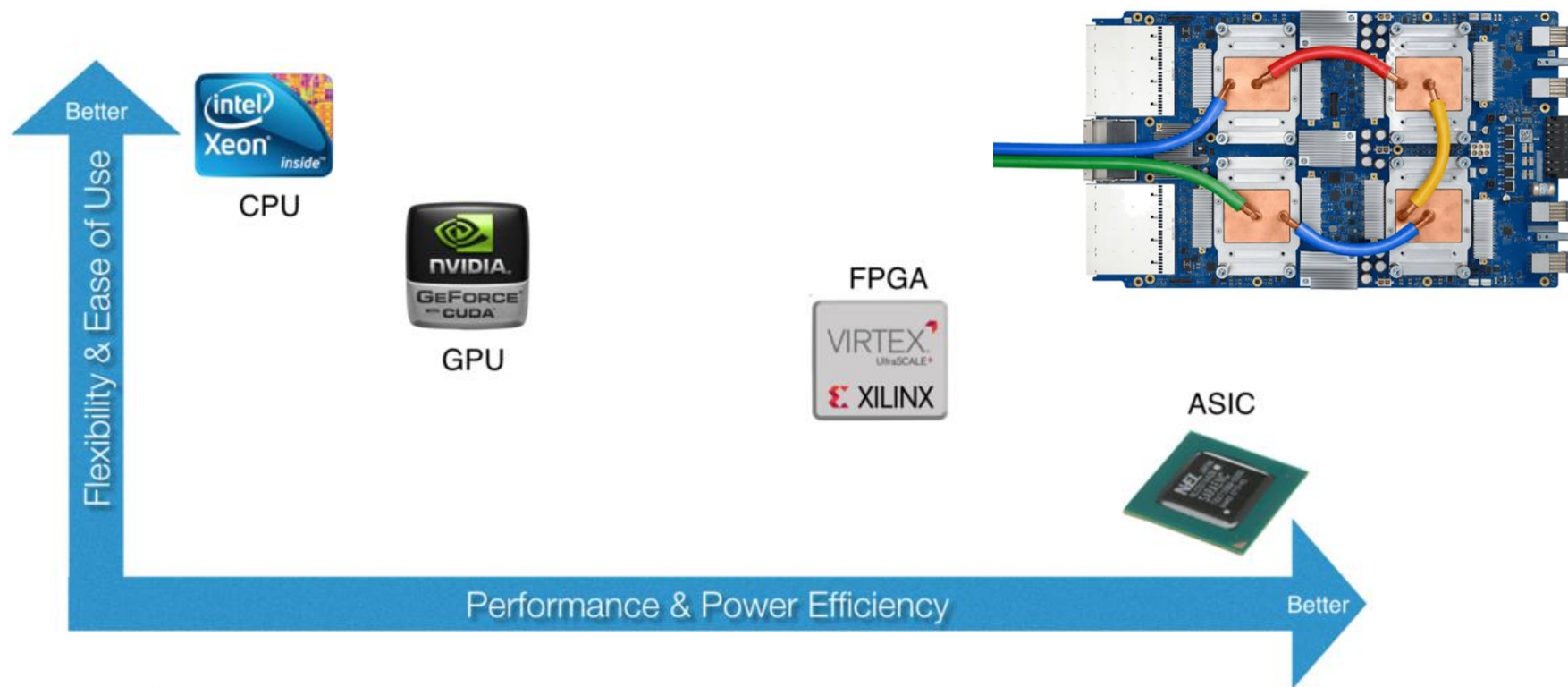
Need for On-device Acceleration

- Task complexity, memory, and classification energy

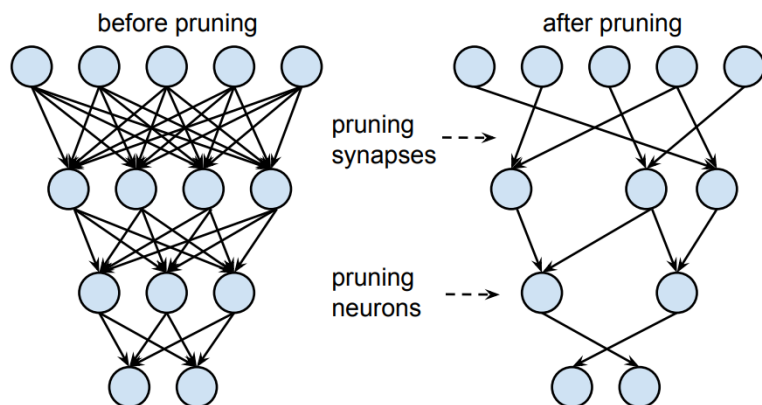


Choosing the Optimal ML Hardware

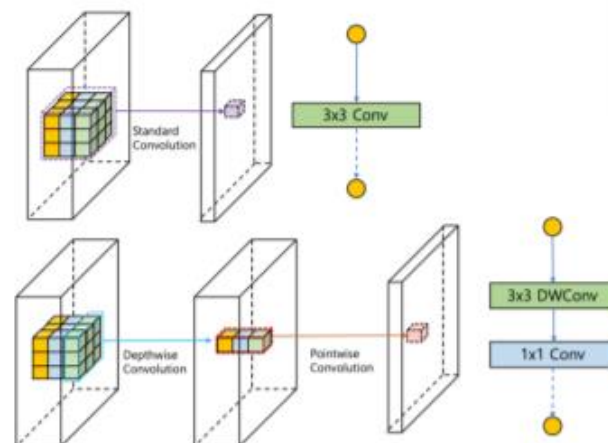
- Hardware preferences
 - Some networks are effective in GPUs, and others are accelerated by ML processors



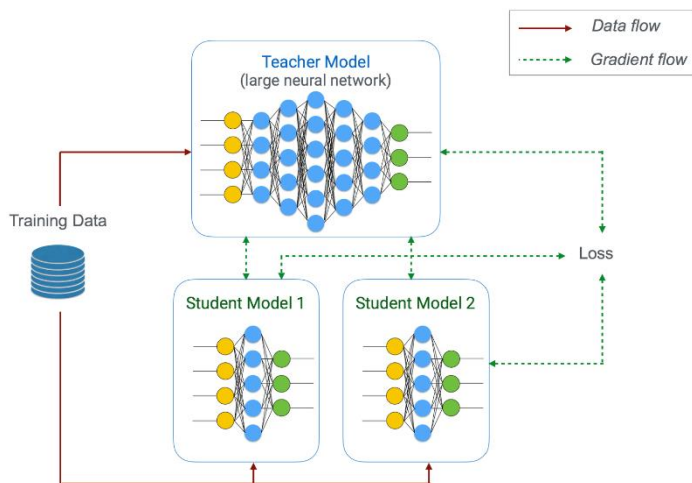
Lightweight Deep Neural Network



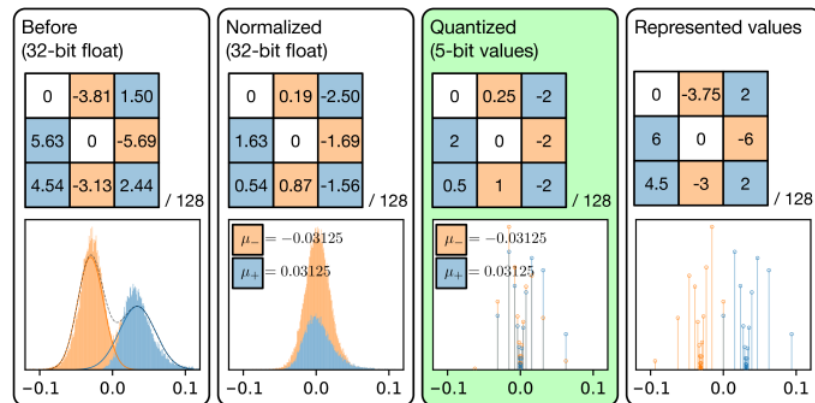
Network Compression: Deep Compression



Network Optimization: MobileNets



Knowledge Distillation



Quantization

Deep Compression

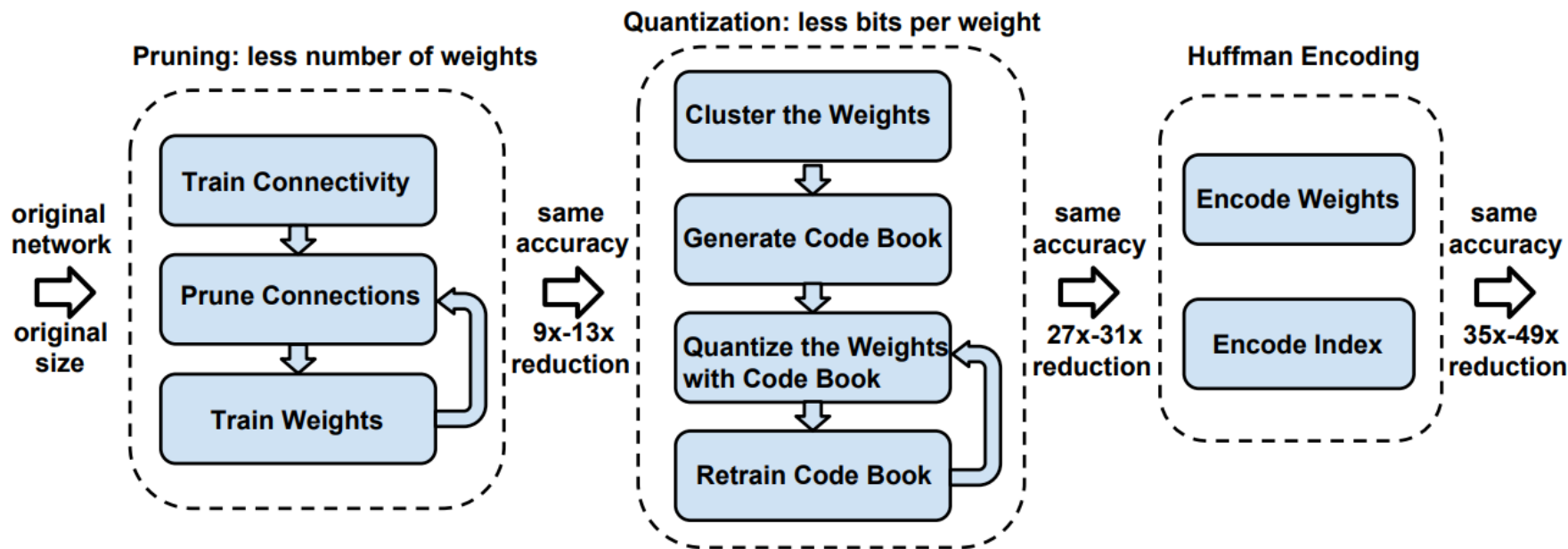
- Overall Algorithm
 - Network Pruning
 - Weight Quantization
 - Huffman Coding

Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding

Song Han, Huizi Mao, William J. Dally

NIPS Deep Learning Symposium, December 2015.

International Conference on Learning Representations (ICLR), May 2016, Best Paper Award.

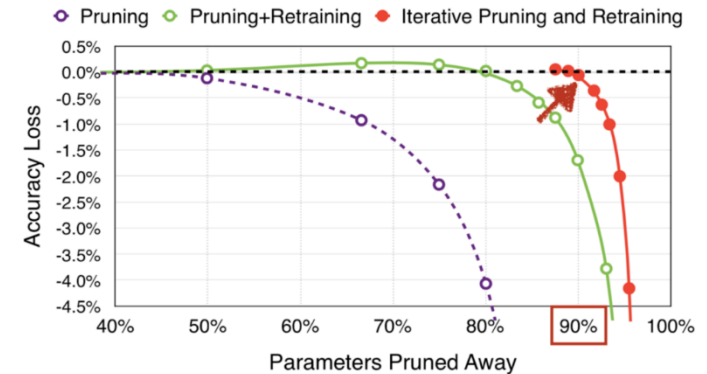
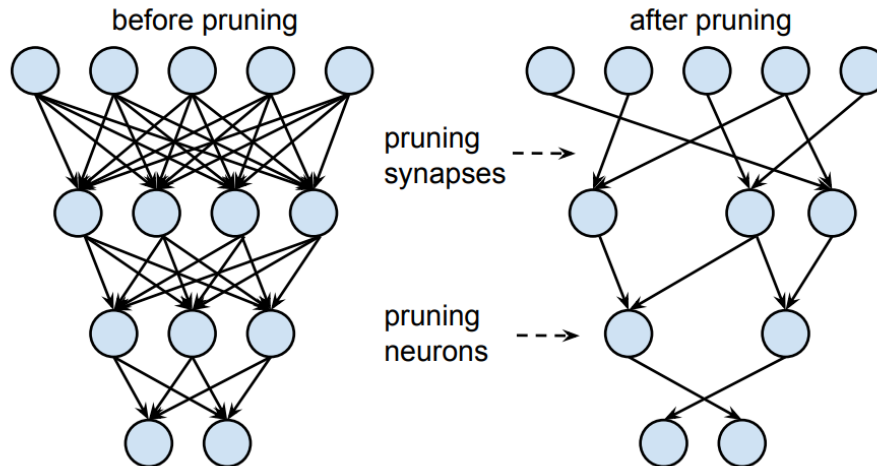
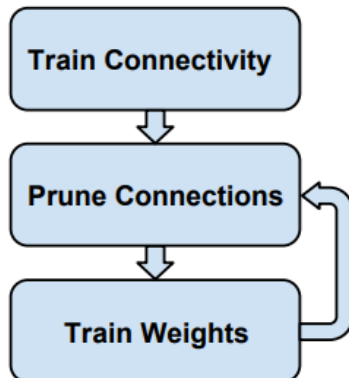


Deep Compression

- Network Pruning

- Three step processes

- Learning connectivity via normal network training
 - ⚡ Not learn final values of the weights, rather learn which connections are important
 - All connections with weights below a threshold are removed from the network
 - Retraining the network to learn the final weights of remaining sparse connections



Knowledge Distillation

- What is the knowledge distillation?
 - All methods of compressing the ensemble model (or big model) into a single model
 - Knowledge distillation refers to performance transfer
 - Through training between teacher and student

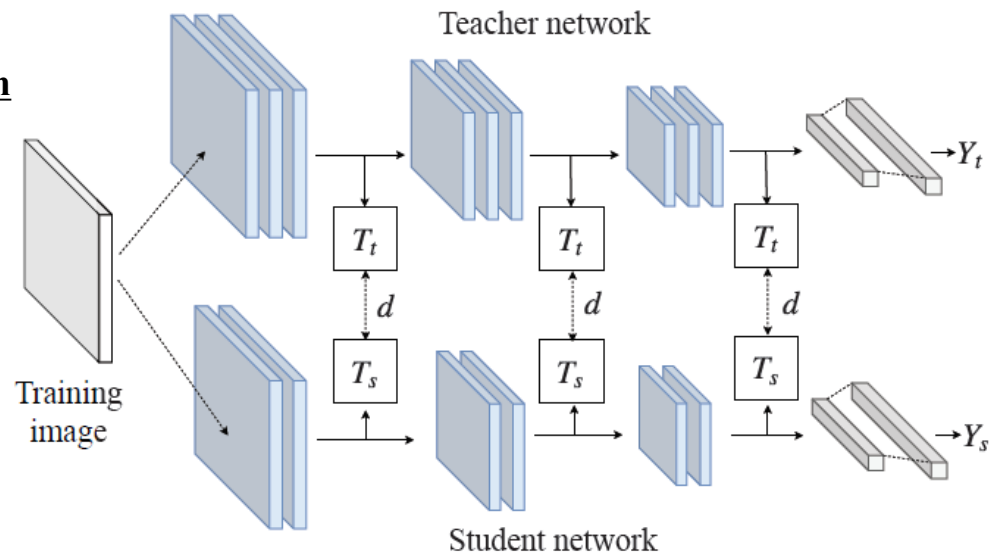
$$L_{distill} = d(T_t(F_t), T_s(F_s))$$

The general training scheme of feature distillation

Teacher transform T_t

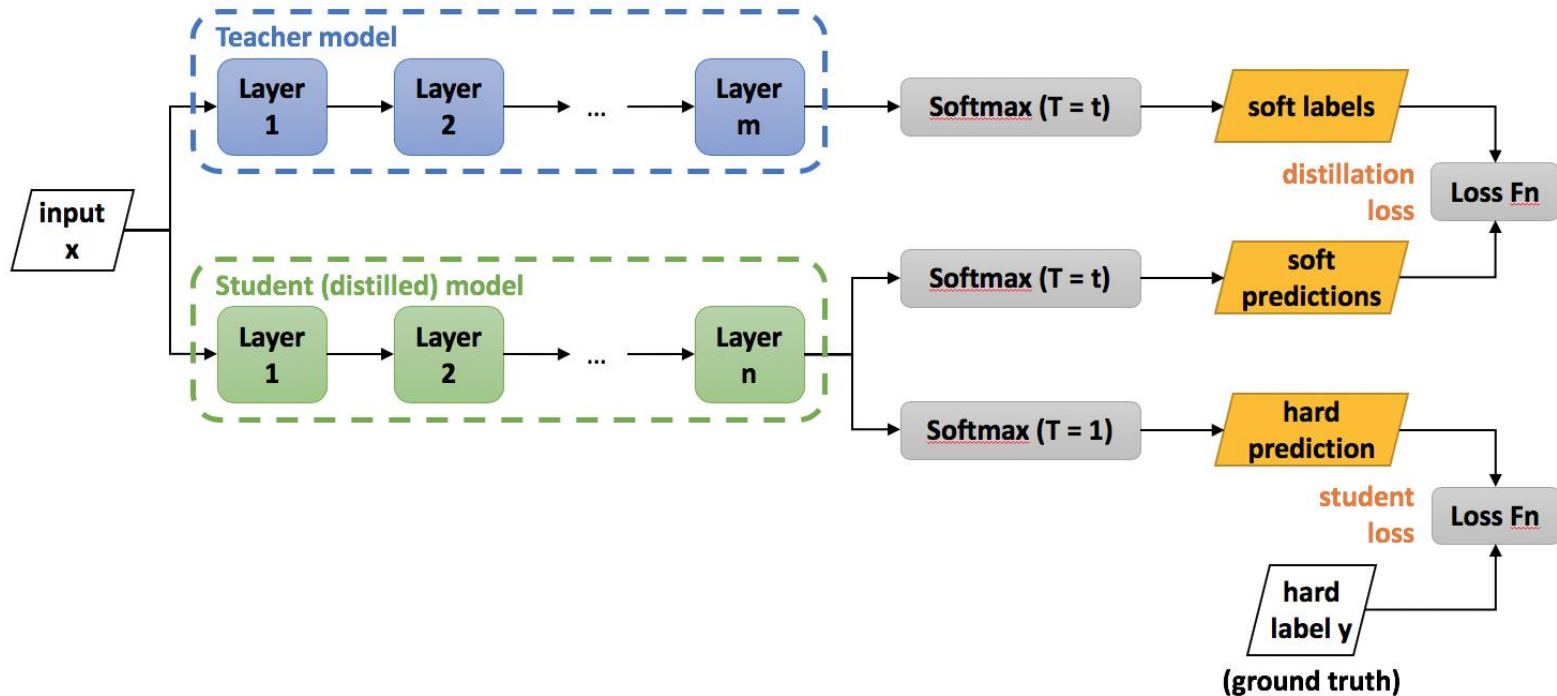
Student transform T_s

Distance d (differs from method to method)



Knowledge Distillation

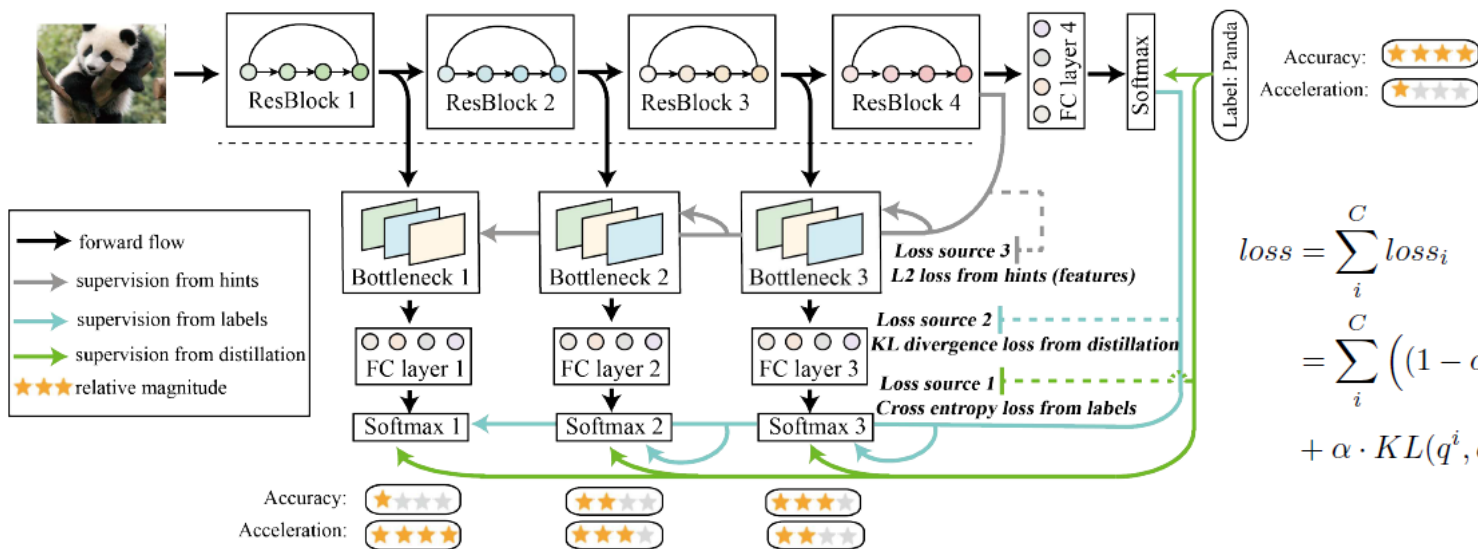
- Distilling the knowledge in a neural network
 - Different model adaptations for training and inference purposes
 - An ensemble (or big) model compression method using Softmax outputs



Knowledge Distillation

- Self Distillation (Overall procedure)

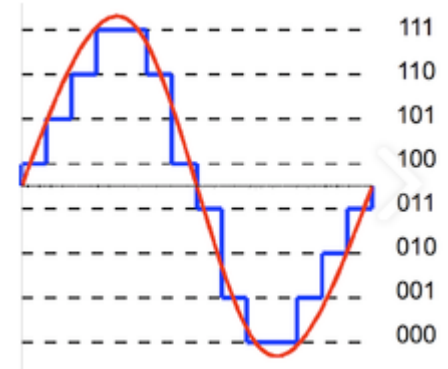
- ① ResNet has been divided into four sections according to their depth
- ② Additional bottlenecks and fully connected layers are set after each section, which constitutes multiple classifiers
- ③ All of the classifiers can be utilized independently, with different accuracy and response time
- ④ Each classifier is trained under three kinds of supervisions
- ⑤ Parts under the dash line can be removed in inference



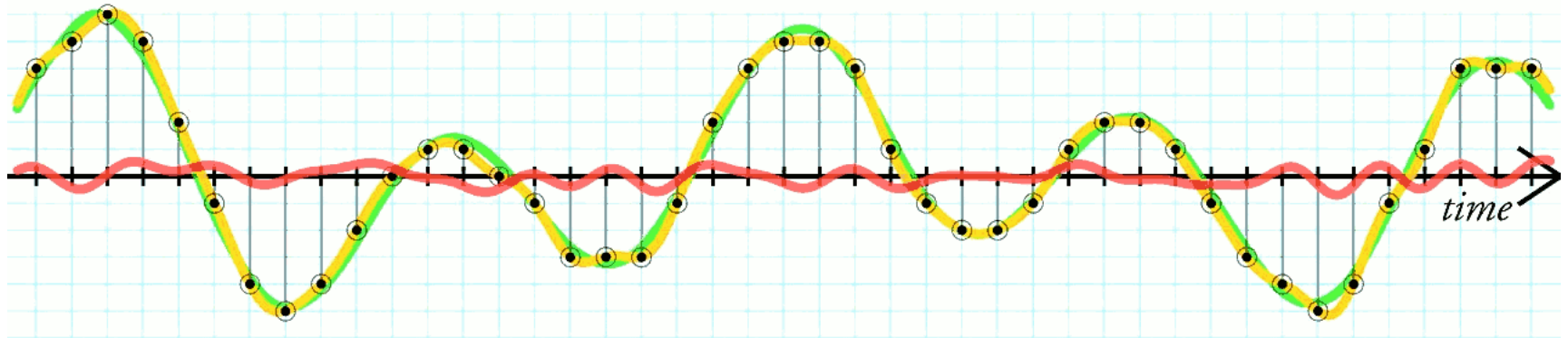
$$\begin{aligned}
 loss &= \sum_i^C loss_i \\
 &= \sum_i^C \left((1 - \alpha) \cdot CrossEntropy(q^i, y) \right. \\
 &\quad \left. + \alpha \cdot KL(q^i, q^C) + \lambda \cdot \|F_i - F_C\|_2^2 \right)
 \end{aligned}$$

Quantization

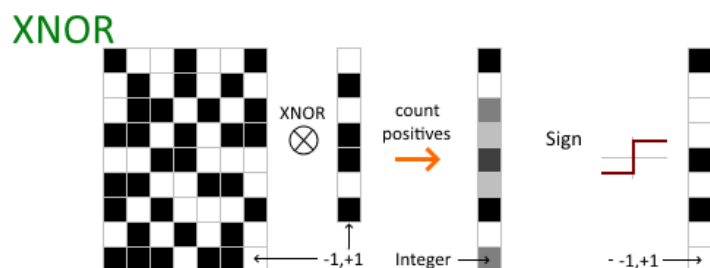
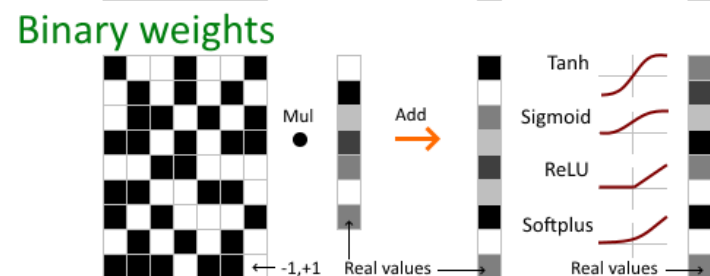
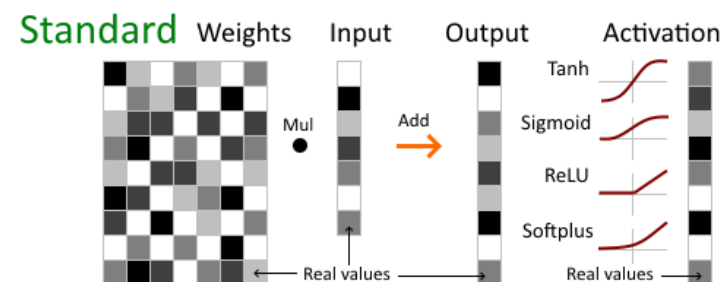
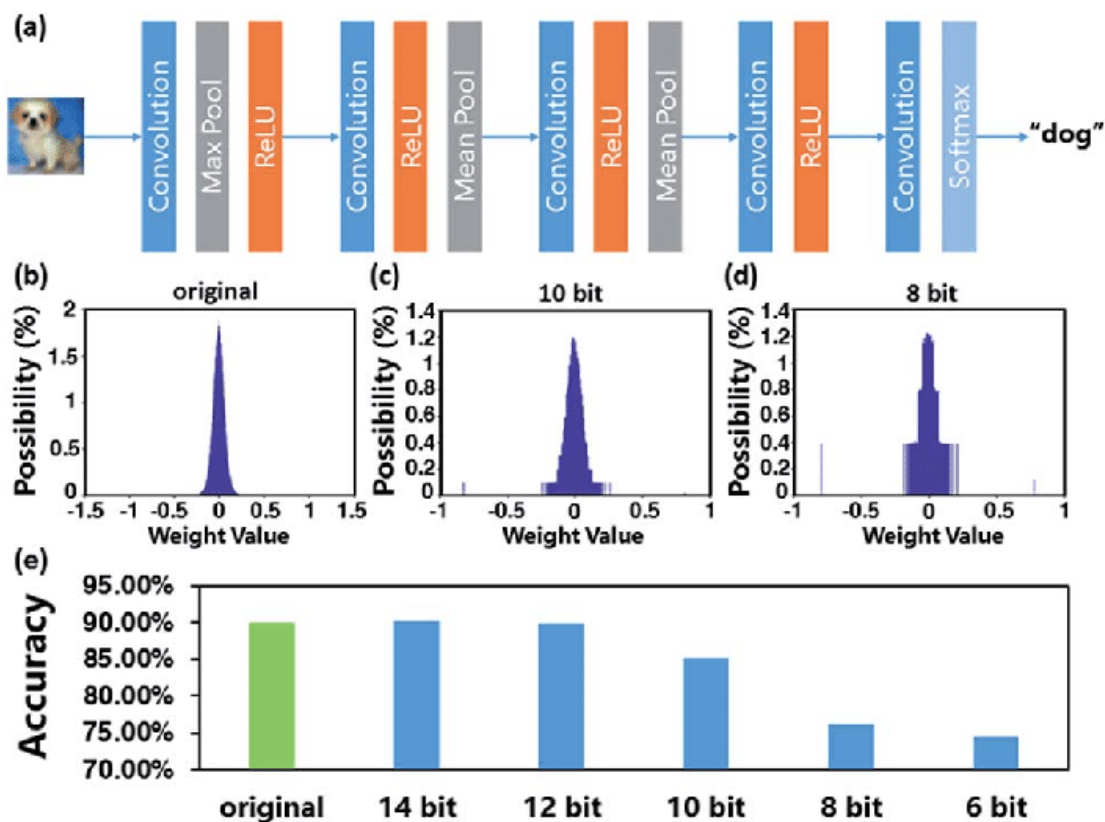
- Mapping input values from a continuous set to output values in a countable set with a finite number of elements



original signal
quantized signal
quantization noise

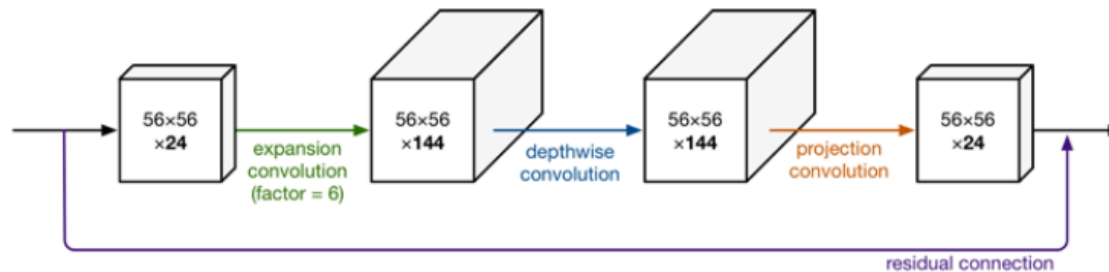


Quantization

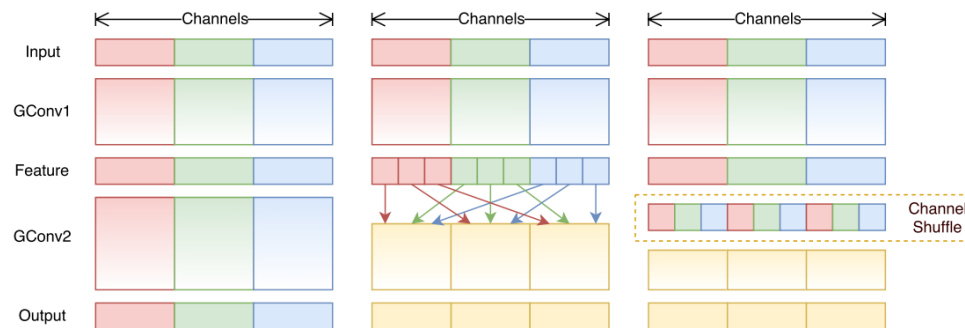


Network Optimization

- Using lightweight convolution block
 - MobileNetV2: Depth-wise separable convolution + Inverted residual block



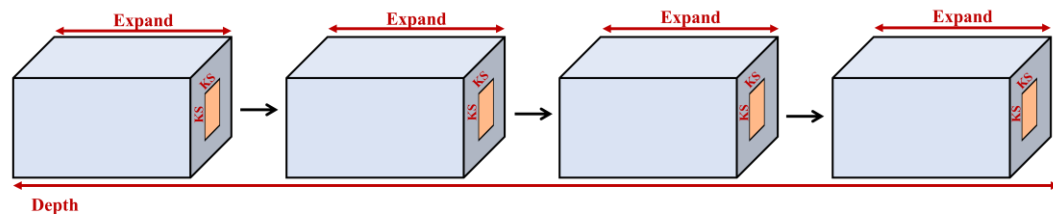
- ShuffleNet: Group convolution + channel shuffling



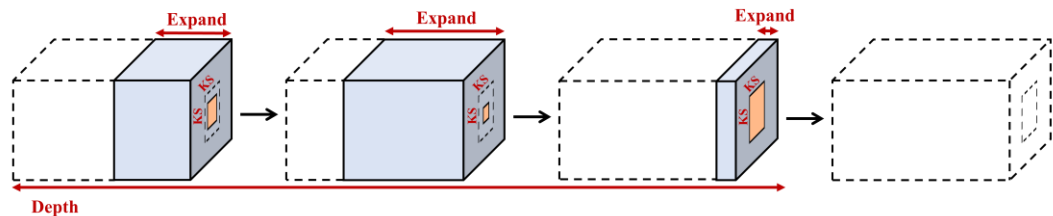
Network Optimization

- Neural architecture search: Weight-sharing Heuristic Search
 - Once-for-All: Train One Network and Specialize it for Efficient Deployment
 - Super-Network Architecture
 - ⚙️ Structure with max. values of Kernel Size, Depth Size, and Expand Ratio Size
 - ✓ Kernel Size: 3×3 , 5×5 , 7×7
 - ✓ Depth Size: # of Block (e.g., 2, 3, 4)
 - ✓ Expand Ratio Size: # of feature map (e.g., 3, 4, 6)

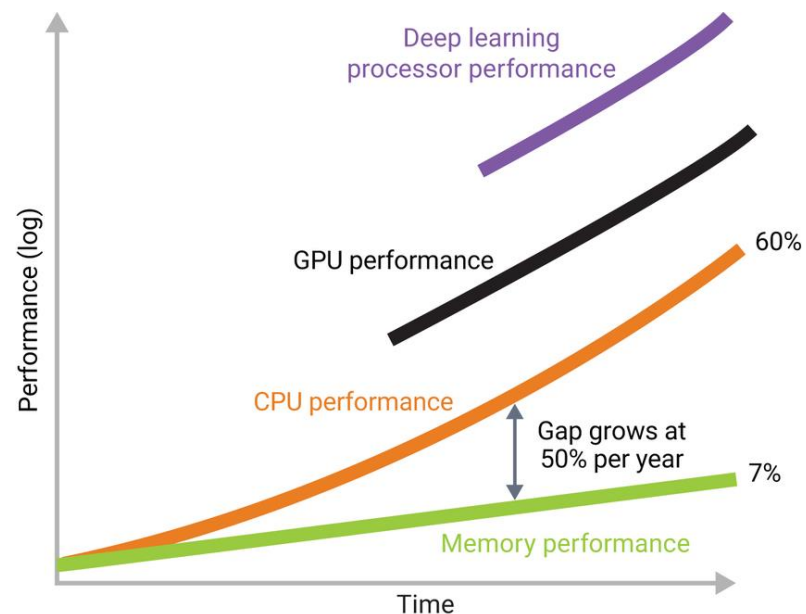
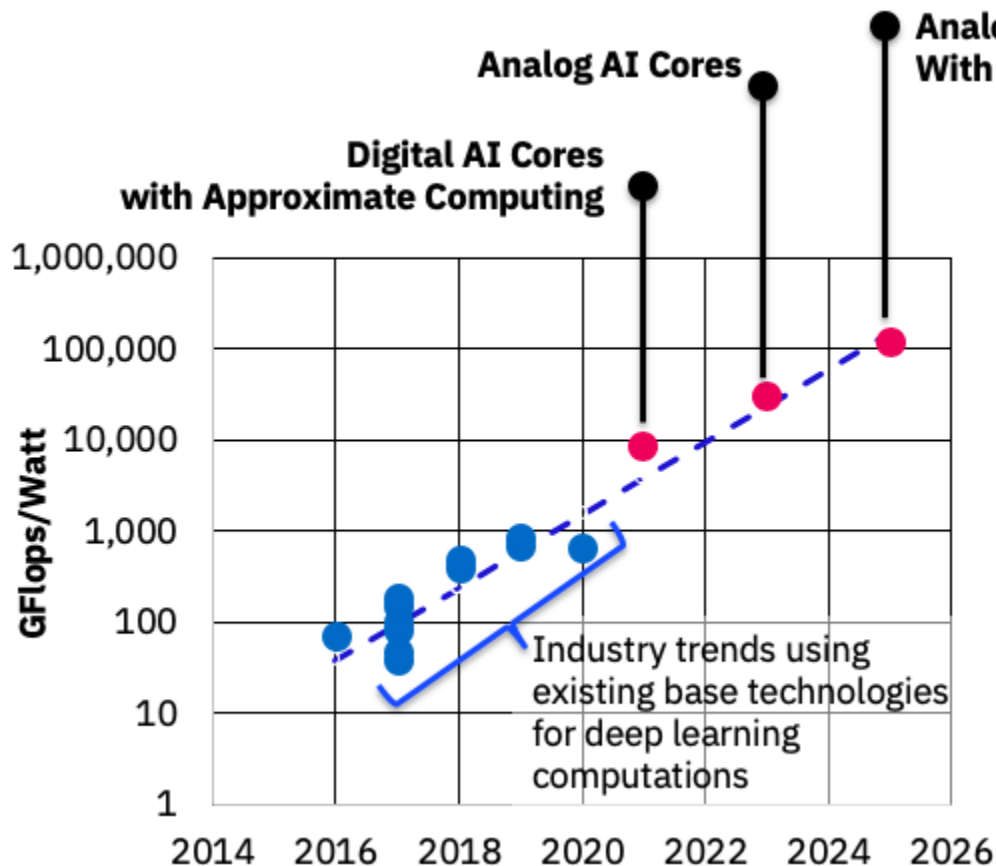
Block Group Architecture of Super-Network



Block Group Architecture of Sub-Network



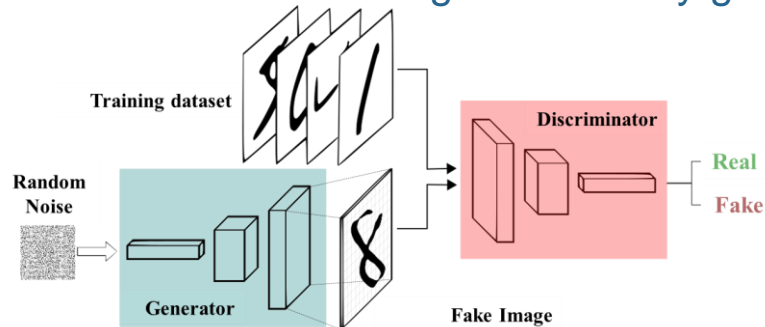
Why do we need DNN Accelerator?



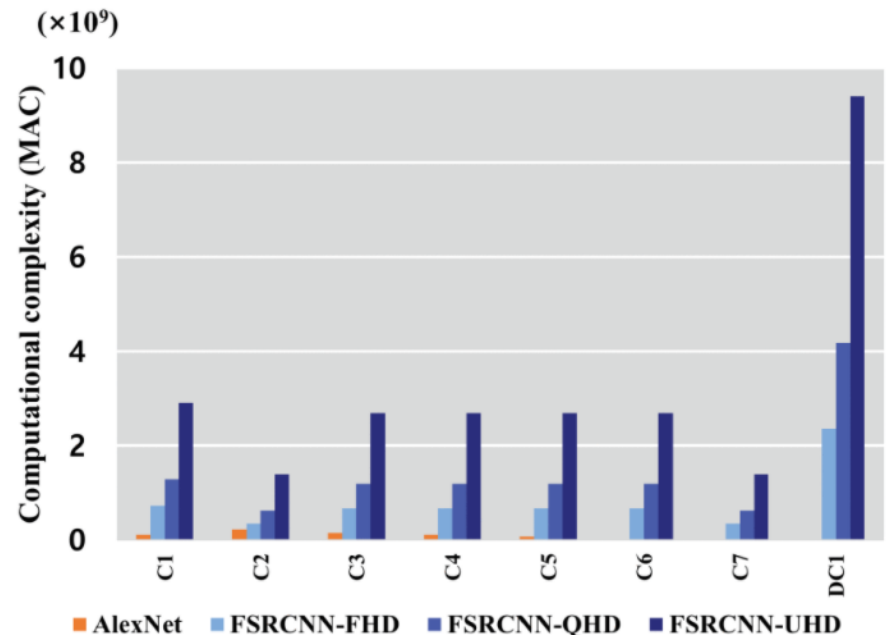
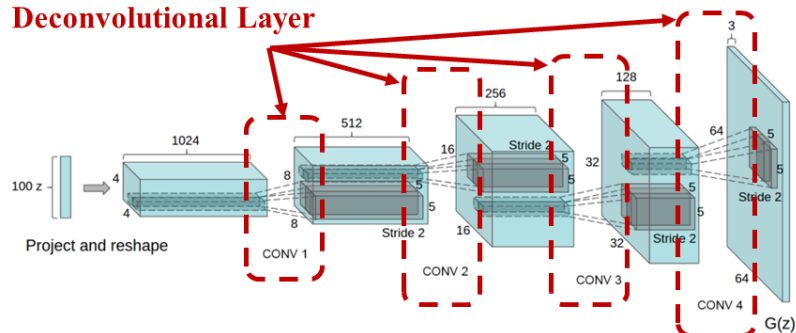
Our Approaches..

Introduction

- Generative adversarial networks (GANs)
 - Generator
 - Produce synthetic data similar to the original training data
 - Mainly composed of deconvolutional layers (transposed convolutional layers)
 - Use data augmentation by generating new data samples from GANs [1]



Deconvolutional Layer



[1] Geirhos, Robert, et al. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness." In *ICLR*, 2019.

Introduction

- Hardware Accelerator

- Improve the computational complexity

- High performance, energy efficiency and fast re-configurability

- Deconvolutional layer

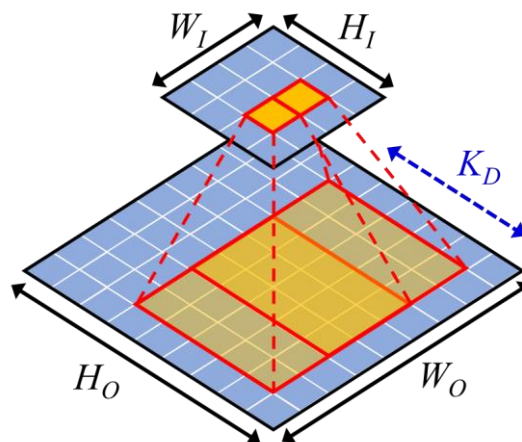
- Used to generate new images and image enhancement applications

- ✧ GANs, super-resolution (SR), and high dynamic range (HDR)

- Different type of mathematical operation

- Overlapping sum problem

- ✧ Interfere with the dataflow of the deconvolution accelerator

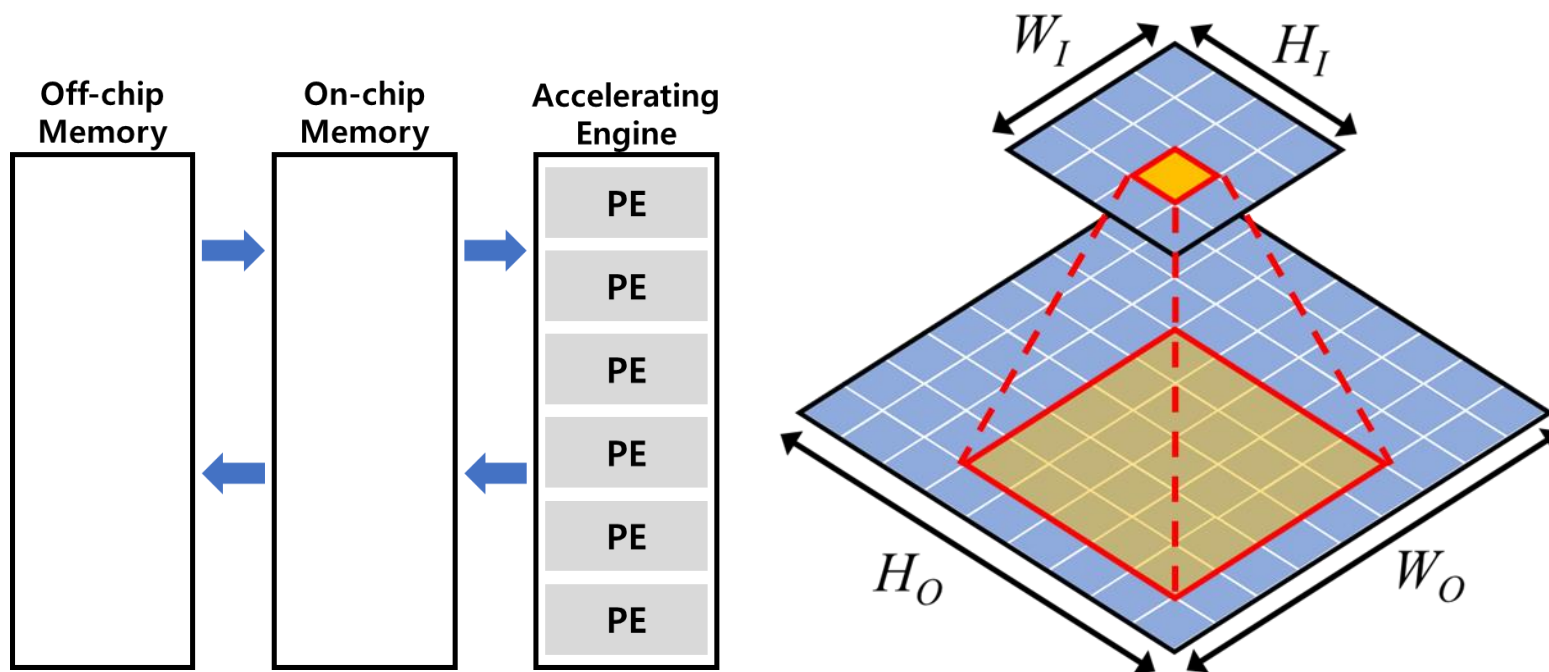


Introduction

- Deconvolutional Layer

- Overlapping sum problem

- Output blocks generated from neighboring input pixels overlap each other
 - Final output is not determined until all neighboring blocks are created

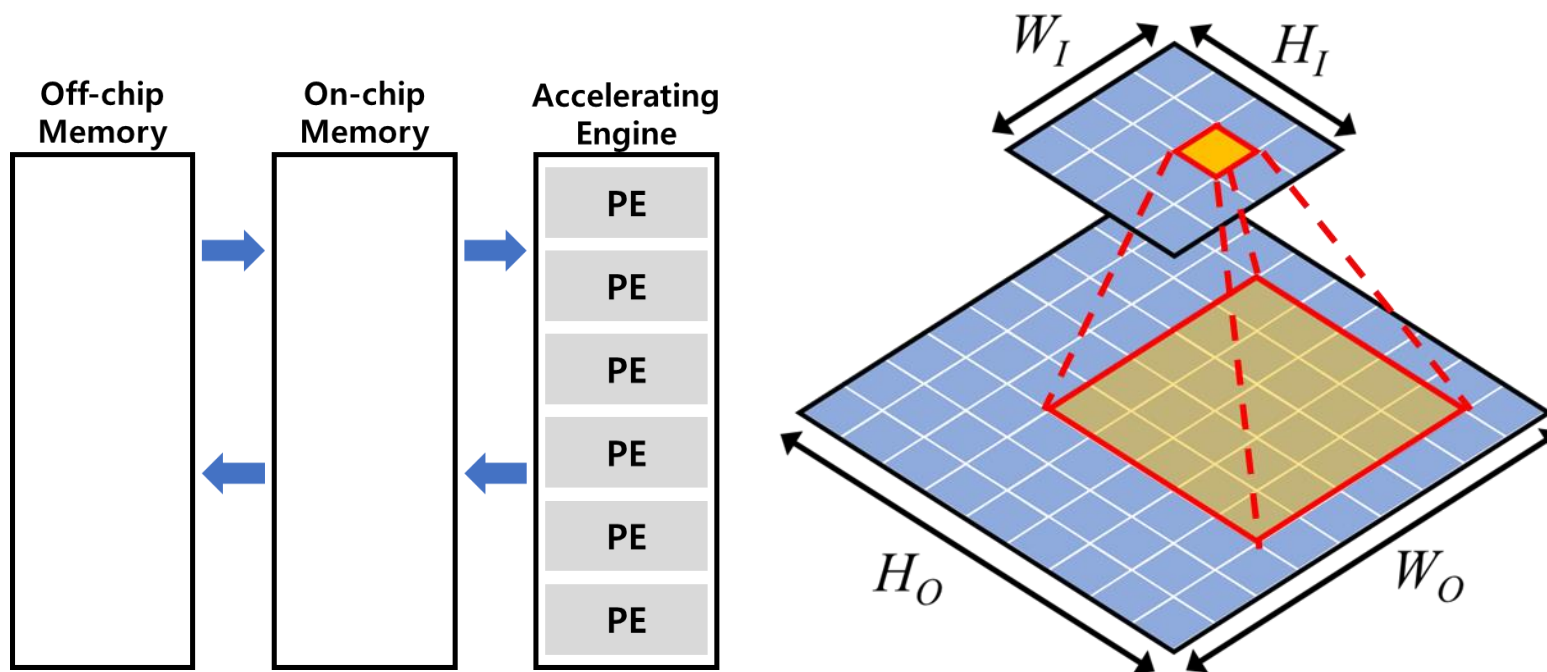


Introduction

- Deconvolutional Layer

- Overlapping sum problem

- Output blocks generated from neighboring input pixels overlap each other
 - Final output is not determined until all neighboring blocks are created

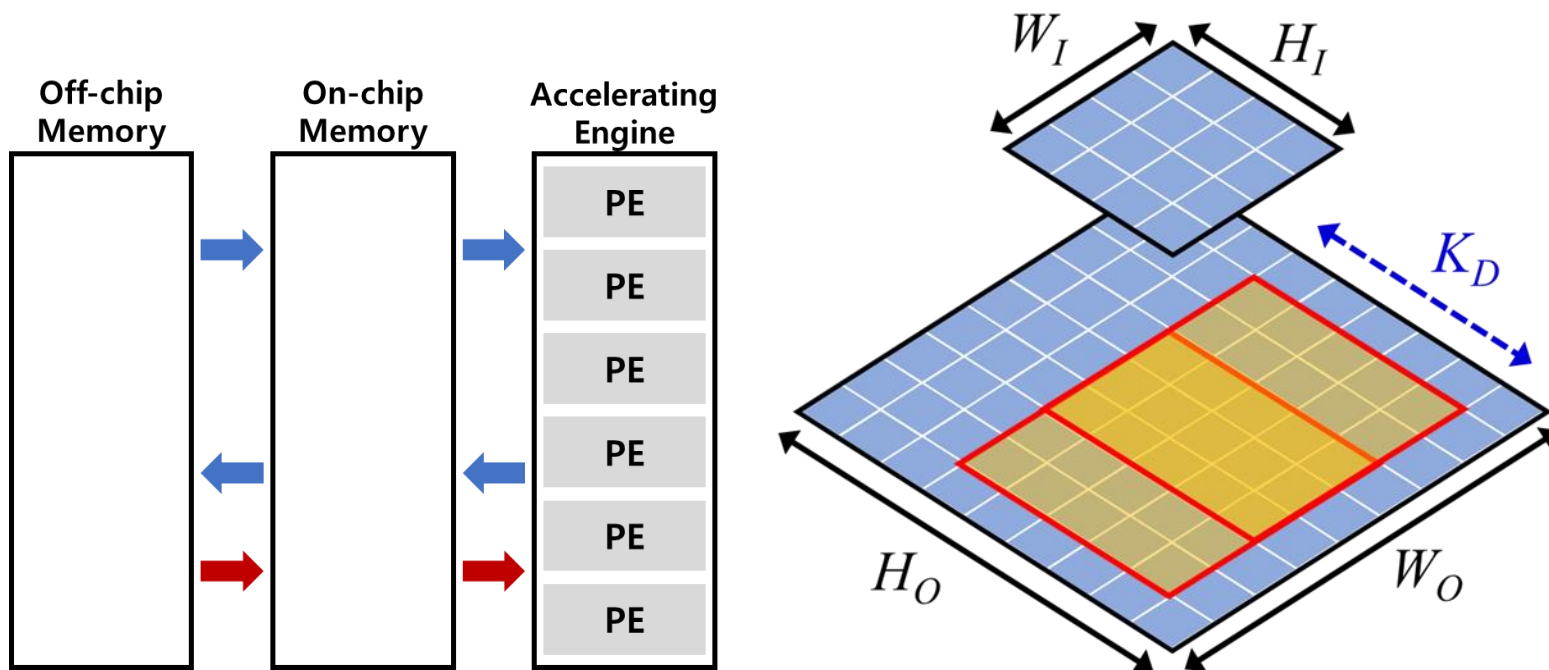


Introduction

- Deconvolutional Layer

- Overlapping sum problem

- Output blocks generated from neighboring input pixels overlap each other
 - Final output is not determined until all neighboring blocks are created
 - **Increase latency, energy consumption and additional hardware resources**

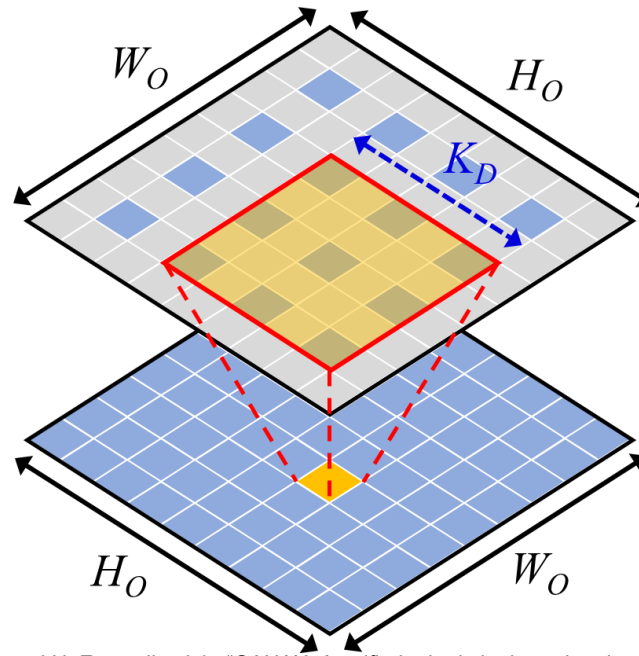


Introduction

- Deconvolutional Layer

- Zero-padded deconvolutional layer [2, 3, 4]

- Avoid the overlapping sum problem
 - Insert zeros to the input feature map
 - Inefficient implementation due to the large loop dimension



[2] A. Yazdanbakhsh, K. Samadi, N. S. Kim, and H. Esmailzadeh, "GANAX: A unified mimd-simd acceleration for generative adversarial networks," In ISCA, pp. 650-661, 2018.

[3] M. Song, J. Zhang, H. Chen, and T. Li, "Towards efficient microarchitectural design for accelerating unsupervised gan-based deep learning," In HPCA, pp. 66-77, 2018.

[4] D. Xu, K. Tu, Y. Wang, C. Liu, B. He, and H. Li, "FCN-engine: Accelerating deconvolutional layers in classic cnn processors," In ICCAD, 2018.

Introduction

- Deconvolutional Layer

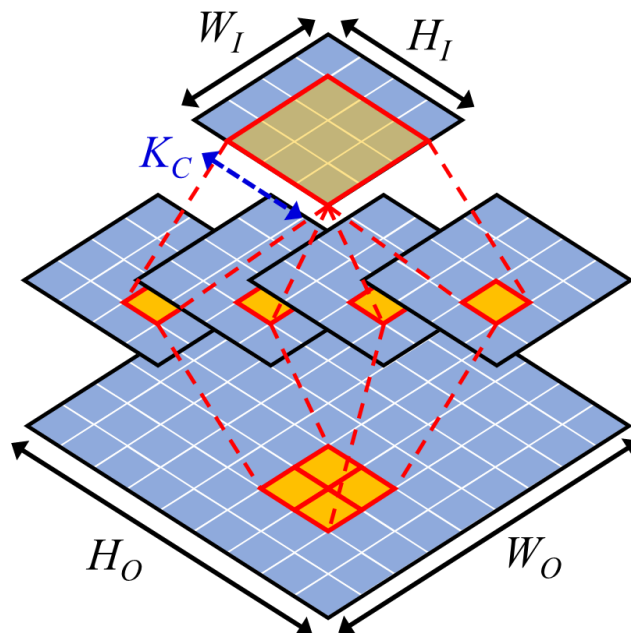
- TDC-based deconvolutional layer [5]

- Transforming a deconvolutional layer into a convolutional layer (TDC)

- ⚙ Spatial transform of the deconvolutional layer

- Improve throughput

- ⚙ Increasing the parallelism of the deconvolutional layer



[5] J.-W. Chang and S.-J. Kang, "Optimizing fpga-based convolutional neural networks accelerator for image super-resolution," In ASP-DAC, pp. 343-348, 2018.

Introduction

- Winograd Algorithm

- Improve resource efficiency

- Reducing the number of multipliers

- 1D-Winograd algorithm

- The number of multiplications is reduced from $m \times r$ to n

⚡ m outputs of an r -tap filter

$$n = m + r - 1$$

$$\begin{array}{ccc} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} z_0 & z_1 & z_2 \\ z_1 & z_2 & z_3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} & \xrightarrow{\text{red arrow}} & \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 + m_4 \end{bmatrix} \end{array} \quad \left| \quad \begin{array}{l} m_1 = (z_0 - z_2)x_0 \quad m_2 = (z_1 + z_2) \frac{x_0 + x_1 + x_2}{2} \\ m_4 = (z_1 - z_3)x_2 \quad m_2 = (z_2 + z_1) \frac{x_0 - x_1 + x_2}{2} \end{array} \right.$$

$m \times r$ n

- Using three transformation matrices, A , B , and G

$$Y = A^T [(Gf) \odot (B^T Z)] \quad \left| \quad B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad G = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix} \right.$$

\odot is element-wise multiplication

Z, f , and Y denote an input, a filter, and an output

Introduction

- Winograd Algorithm
 - Improve resource efficiency
 - Reducing the number of multipliers
 - 2D-Winograd algorithm
 - **The number of multiplications is reduced from $m^2 \times r^2$ to n^2**

$$Y = A^T [(GfG^T) \odot (B^T ZB)] A$$

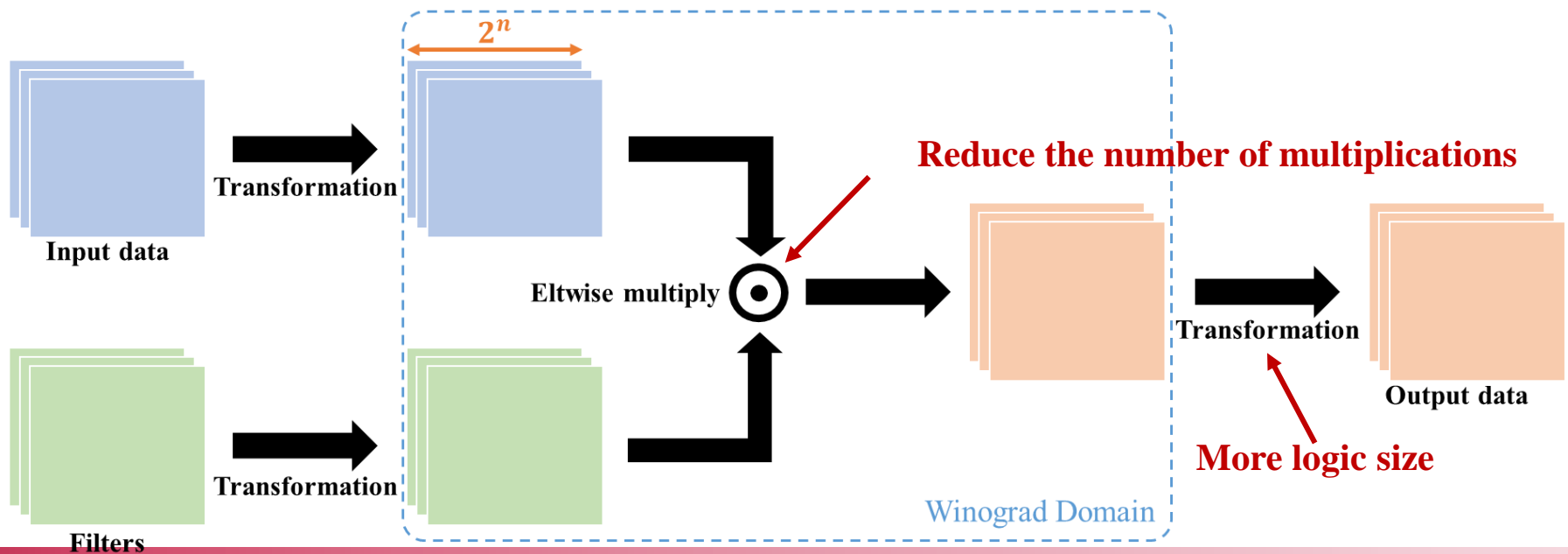
- The matrix size is power of 2 in the Winograd domain
- Deconvolutional layer has not been applied to fast algorithm
 - Different type of mathematical operation compared to convolutional layer

$$n = m + r - 1$$

Z, f , and Y denote an input, a filter, and an output

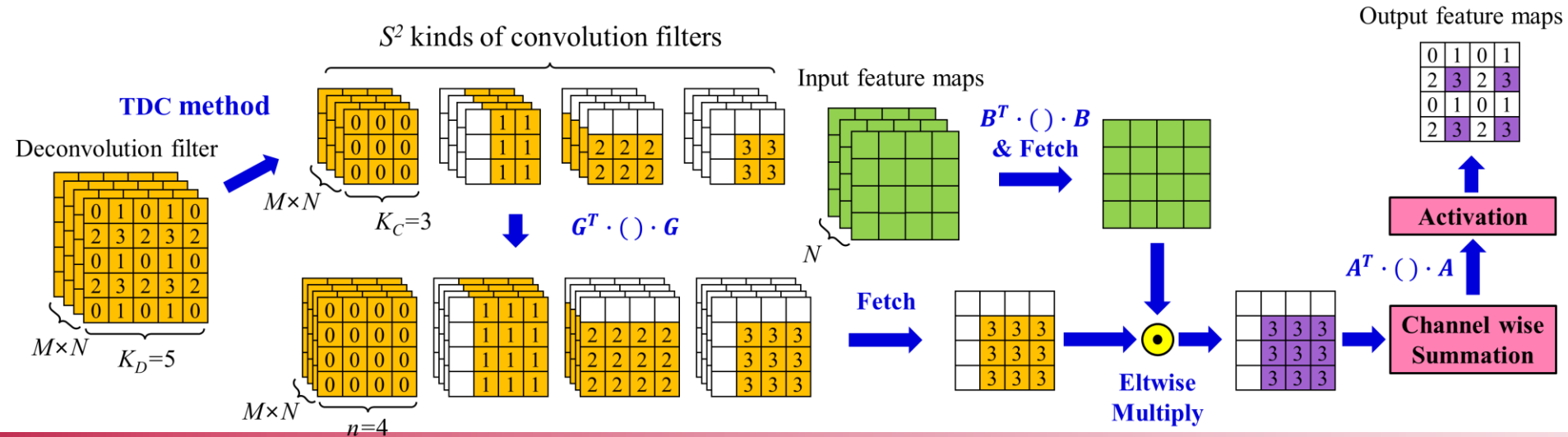
Introduction

- Winograd Algorithm
 - Low computational complexity
 - Reduce the number of multiplications
 - Improve resource efficiency
 - Require more logic size when implementing the hardware
 - Need to transform the input, filters, and output



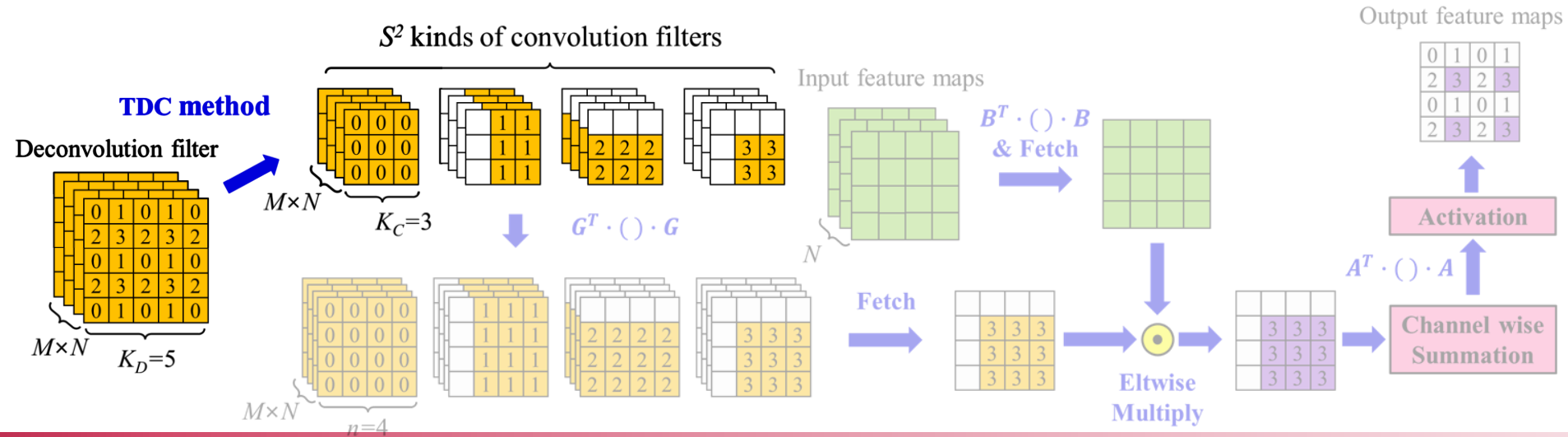
Proposed Method

- Winograd Deconvolution
 - Combination of the two orthogonal methods
 - TDC method and Winograd algorithm



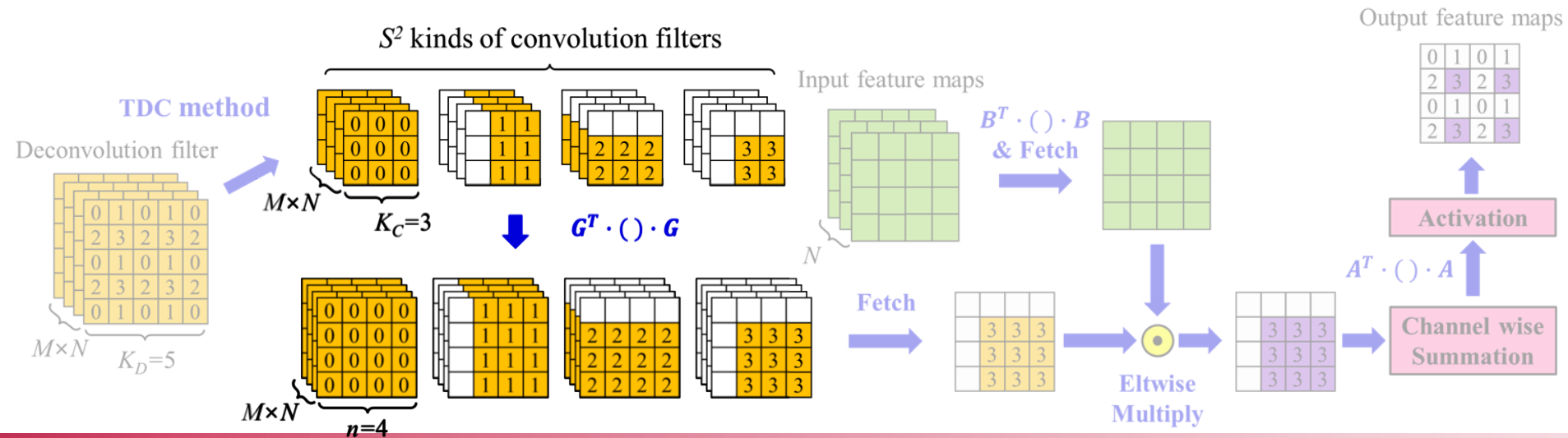
Proposed Method

- Winograd Deconvolution
 - Deconvolutional layer applied the TDC method
 - Convert $M \times N$ deconvolutional layer to the $S^2 \times M \times N$ convolutional layer



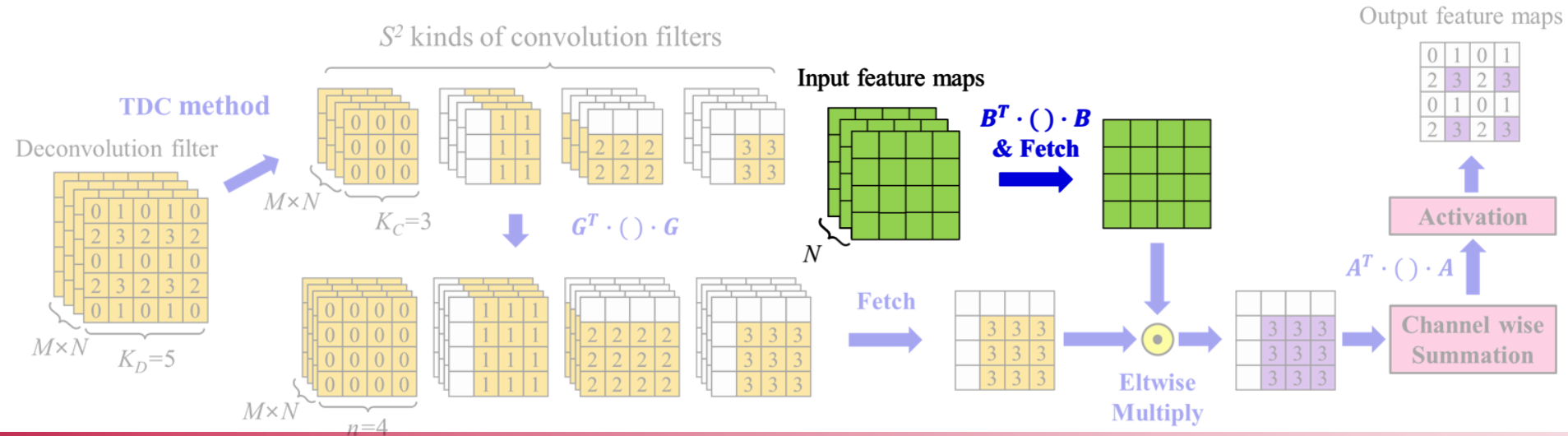
Proposed Method

- Winograd Deconvolution
 - Deconvolutional layer applied the TDC method
 - Convert $M \times N$ deconvolutional layer to the $S^2 \times M \times N$ convolutional layer
 - Apply Winograd minimal filtering algorithm to the newly created convolutional layer
 - Zero-valued weights are remained at fixed positions
 - ✧ Serious resource underutilization in FPGAs



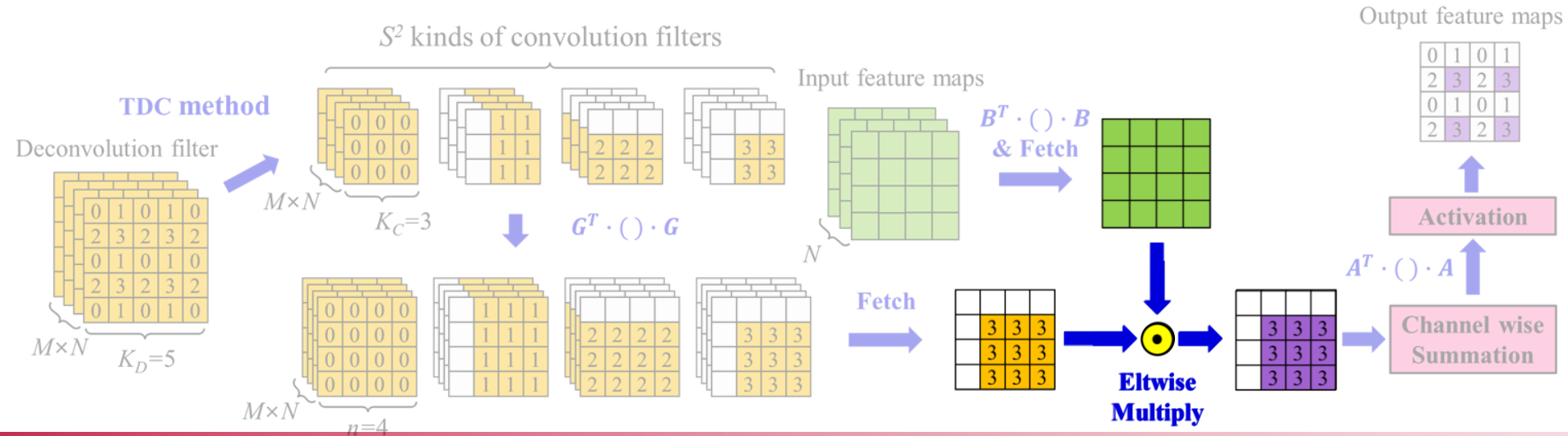
Proposed Method

- Winograd Deconvolution
 - Input tiles are moved to Winograd domain using transformation matrix B



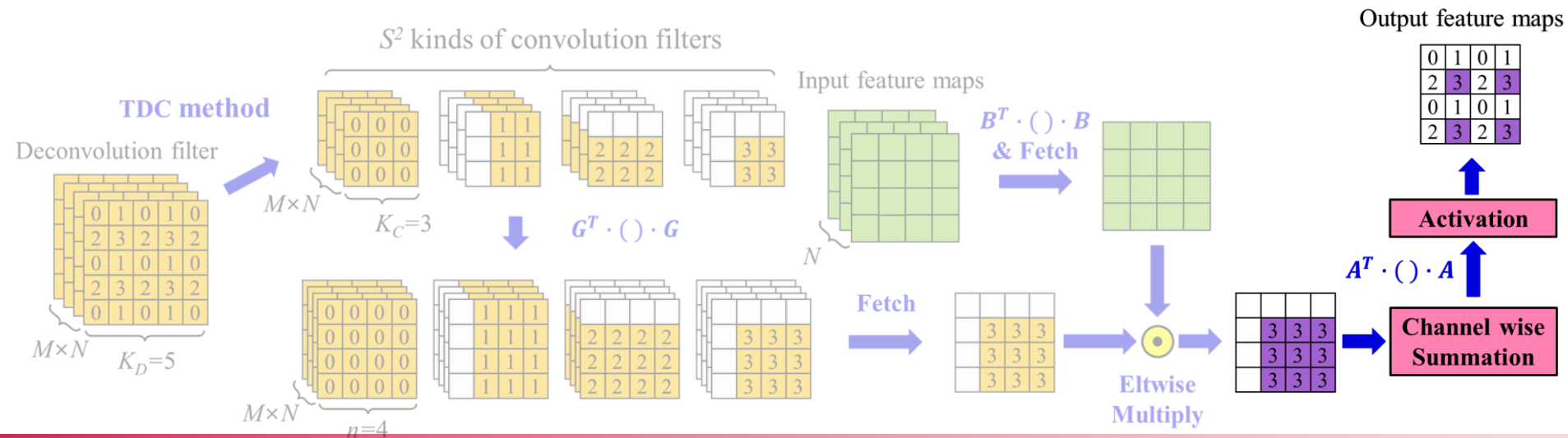
Proposed Method

- Winograd Deconvolution
 - Input tiles are moved to Winograd domain using transformation matrix B
 - Transformed filters and input tiles operate on element-wise multiplication
 - Except for the position of zero-valued weights



Proposed Method

- Winograd Deconvolution
 - Channel-wise summation the multiplication outputs
 - Process inverse transformation with matrix A



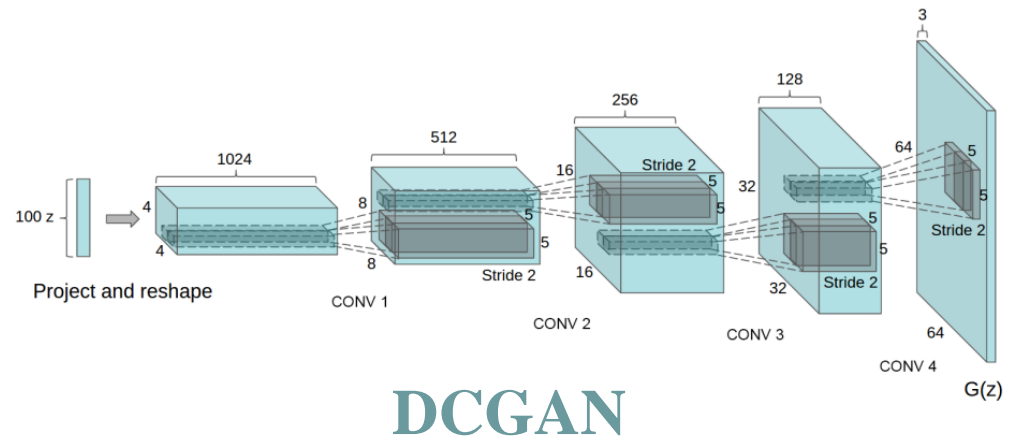
Experimental Environment

- Hardware Implementation Tool

- High-level synthesis
- Vivado HLS 2016.4
- Xilinx Virtex-7 485T FPGA
- Single-precision floating point

- CNN Models

- DCGAN [6]
- ArtGAN [7]
- DiscoGAN [8]
- GP-GAN [9]



[6] J.A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015.

[7] W. R. Tan, C. S. Chan, H. E. Aguirre, and K. Tanaka, "ArtGAN: Artwork synthesis with conditional categorical GANs," In ICIP, 2017.

[8] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover crossdomain relations with generative adversarial networks," In ICML, pp. 1857-1865, 2017.

[9] H. Wu, S. Zheng, J. Zhang, and Huang, "GP-GAN: Towards realistic high-resolution image blending," In ACM MM, pp. 2487-2495, 2019.

Experimental Results

- Resource Utilization
 - Our design required additional operations in pre-PE and post-PE
 - Implemented those PEs using LUTs
 - Same tiling parameters as [5]
 - Our design used more BRAMs
 - Store more transformed weights in the Winograd domain

	BRAM18K	DSP48E	LUT	FFs
[5]	384	2,560	94,264	107,626
Ours	520	2,560	142,711	151,395

Resource Utilization for DCGAN

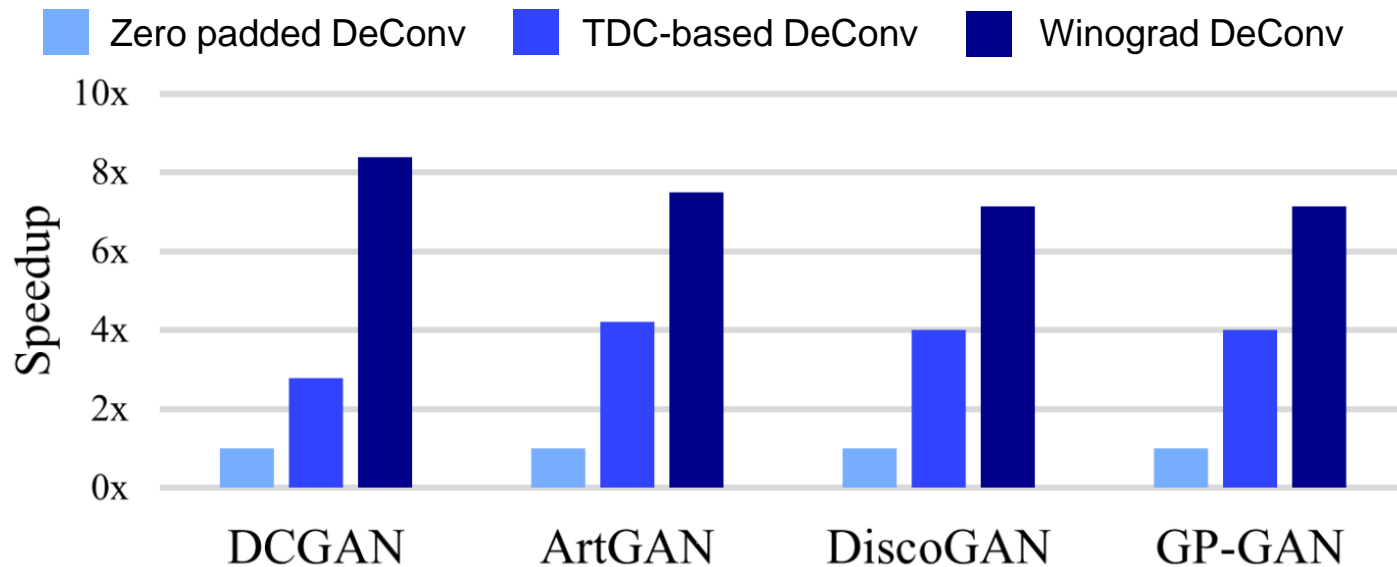
[5] J.-W. Chang and S.-J. Kang, "Optimizing fpga-based convolutional neural networks accelerator for image super-resolution," In ASP-DAC, pp. 343-348, 2018.

Experimental Results

- Performance Evaluation

- Winograd deconvolution

- Remove an overlapping sum problem by TDC method
 - Avoid redundant computation
 - Achieve maximum 8.38× faster than zero-padded deconvolution

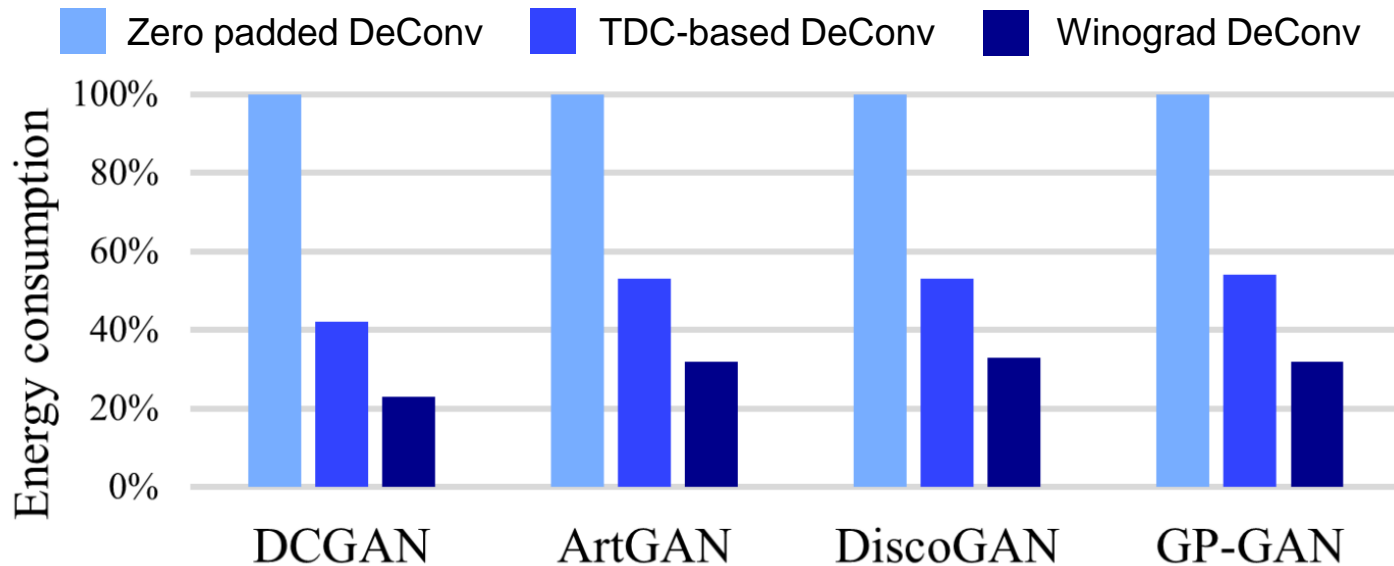


Experimental Results

- Energy Consumption

- Winograd deconvolution

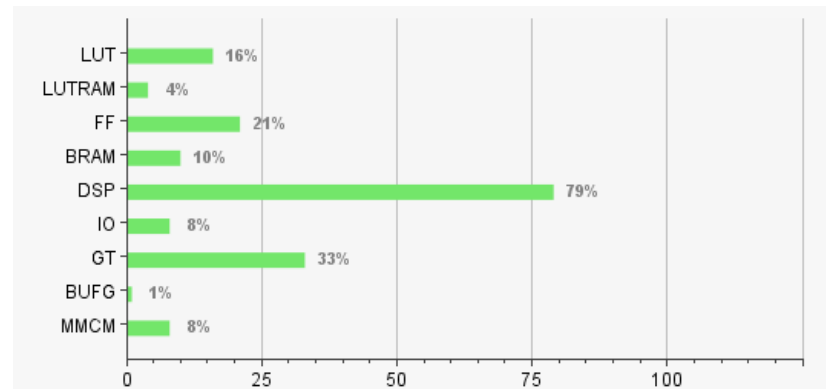
- Reduce the energy consumption by 3.65× over zero-padded deconvolution
 - Difference of the amount of data transfer between the on-chip buffer and the off-chip buffer memory
 - Reduce the number of multiplications



FPGA-based Implementation

- FPGA resource for our super resolution (Kintex 7 Ultrascale)
 - DSP usage is 79% and the remaining resources are less than 20%
 - Optimized results without timing issue
 - Adder is implemented with LUTs and FFs → Requiring less logic elements
 - Resources other than DSP are minimized and DSP utilization is maximized

Resource	Utilization	Available	Utilization %
LUT	106796	663360	16.10
LUTRAM	10332	293760	3.52
FF	277264	1326720	20.90
BRAM	208	2160	9.63
DSP	4348	5520	78.77
IO	47	624	7.53
GT	16	48	33.33
BUFG	14	1248	1.12
MMCM	2	24	8.33



FPGA resources

Thank You