

# Convolutional Neural Network Quantization, BRAM

Xuan Truong  
2022.1.10 (Tue)



# Road map

Deep Learning Accelerator

Convolutional Neural  
Network (CNN)

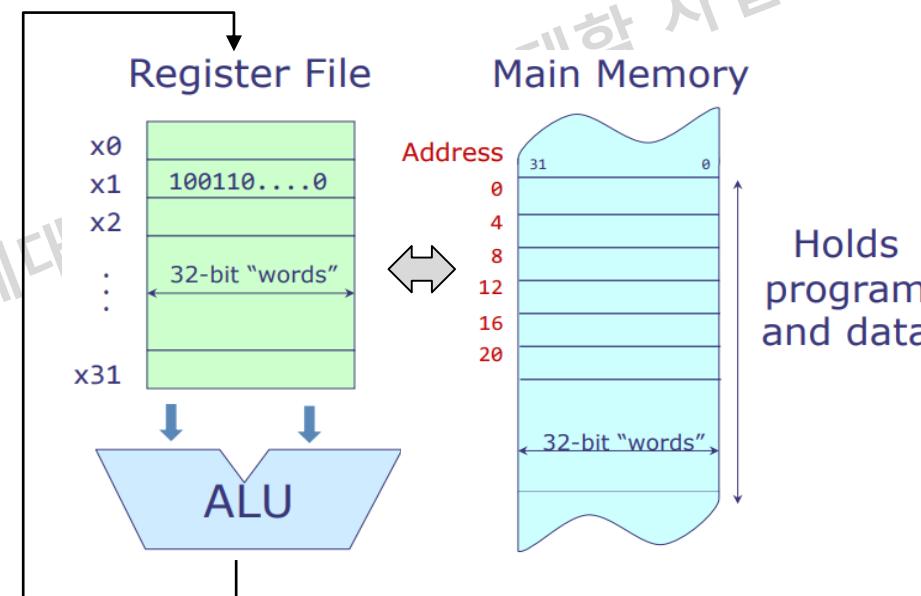
Quantization

Block memory

Reference S/W

# Performance Issue

- Each single-core PC has only **one computation unit**, i.e. ALU, multiplier.
  - Example: bright adjustment
    - For each pixel in an image for display
      - Increase or decrease its value by a number.
- ⇒ Complexity = total pixels in an image, i.e. 1,179,648 (= $768 \times 512 \times 3$ )
- Why doesn't a PC have more cores? How many cores?

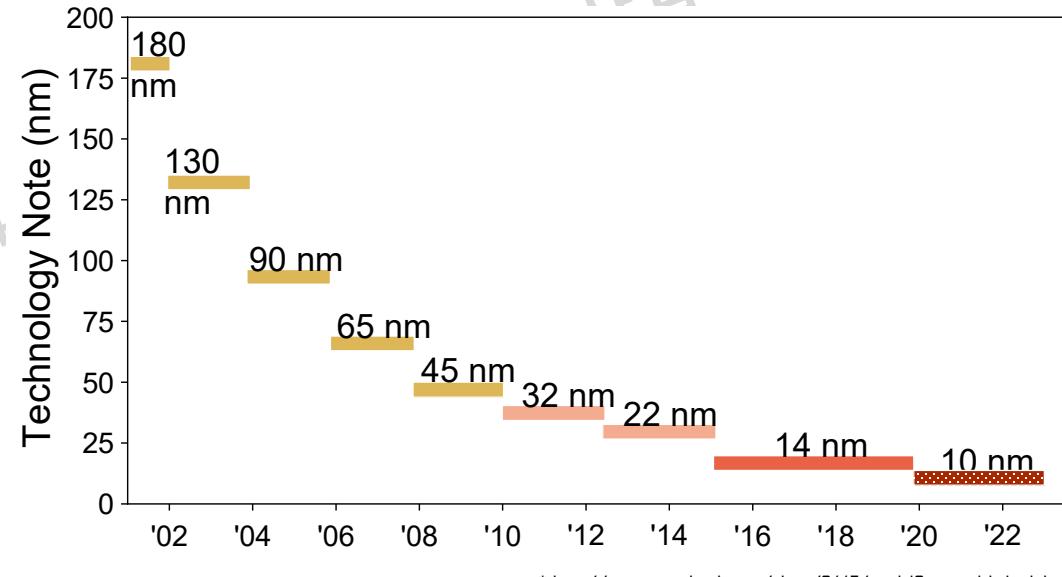


# Moore's Law

- Every two years:
  - Double the number of transistors
  - Build higher performance general-purpose processors
  - Make the transistors available to the masses
  - Increase performance ( $1.8 \times \uparrow$ )
  - Lower the cost of computing ( $1.8 \times \downarrow$ )



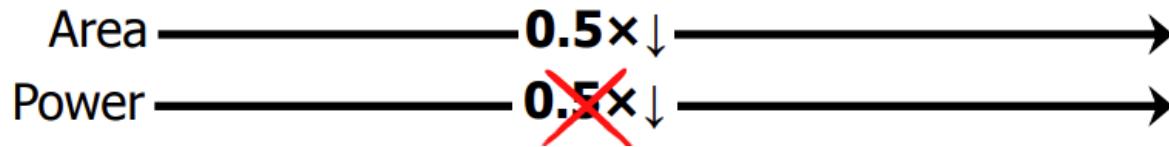
Gordon Moore



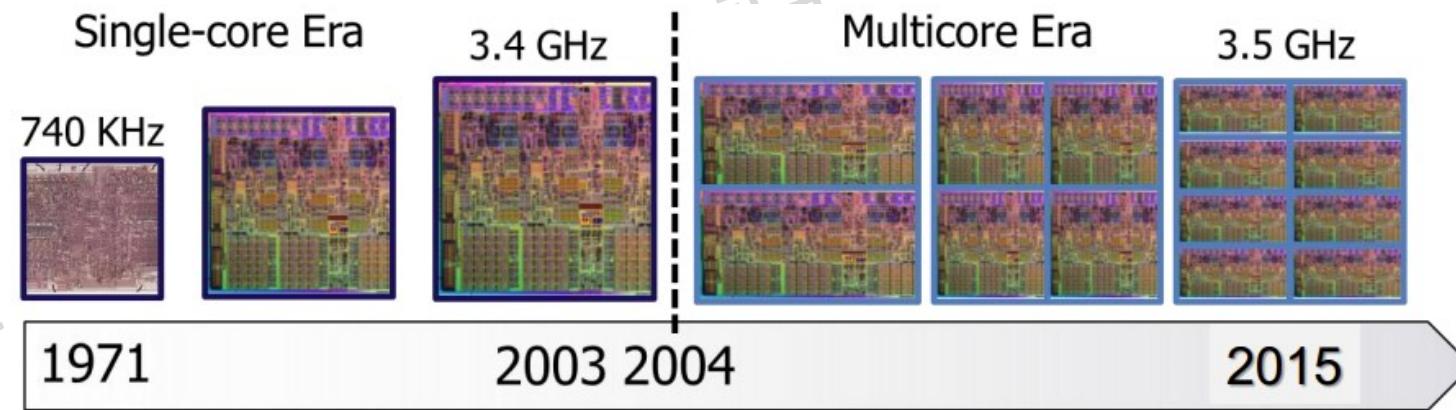
\* <http://www.anandtech.com/show/9447/intel-10nm-and-kaby-lake>

# Dark Silicon

- Dark silicon – the fraction of transistors that need to be powered off at all times (due to power + thermal constraints)

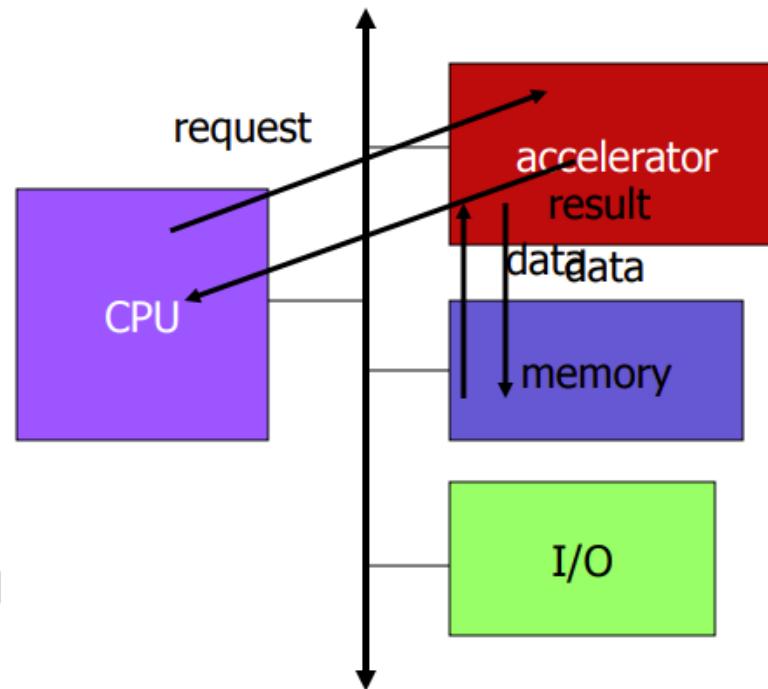


- Evolution of processors  
⇒ Expected to continue evolution towards HW specialization/acceleration



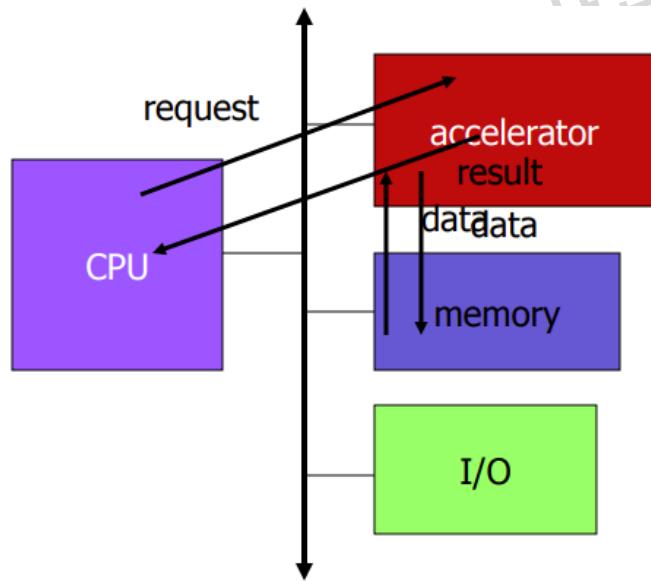
# Accelerated Systems

- Use additional computational units dedicated to some functionality
  - A H/W module for special applications
- Hardware/software co-design:
  - Joint design of hardware and software architectures



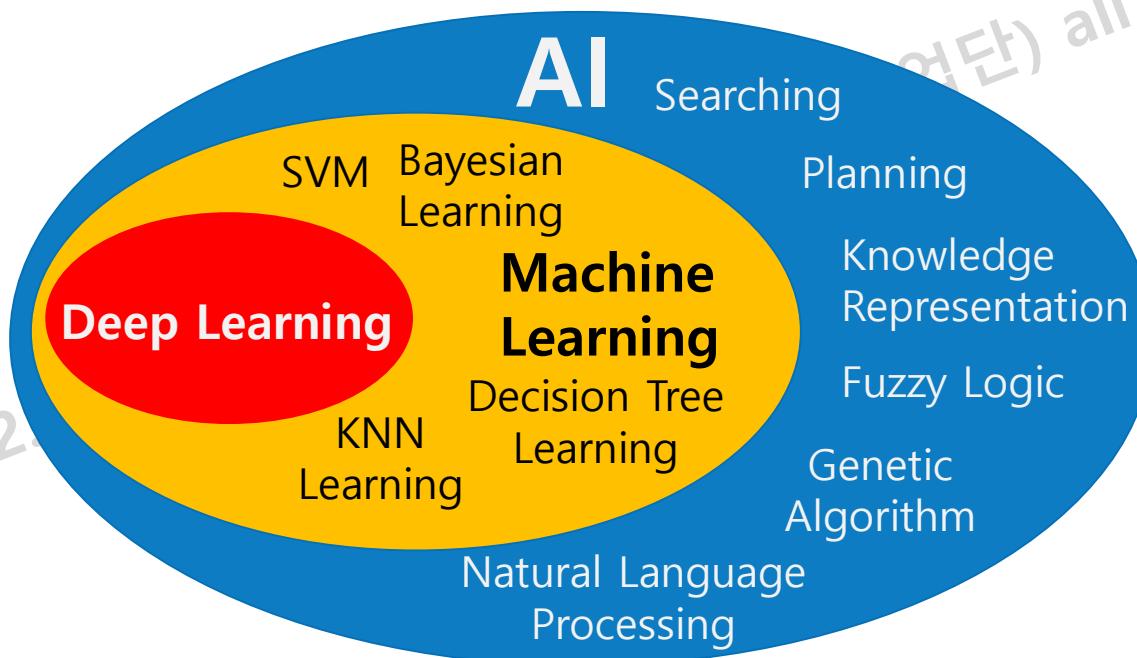
# Accelerator vs. Coprocessor

- A coprocessor executes instructions
  - Instructions are dispatched by the CPU
- An accelerator appears as a device on the bus
  - Typically controlled by registers (memory-mapped IO)
- When we talk about “HW Accelerator”, we talk about Efficiency
  - Not just about “correctness”, but how “GOOD” it is.



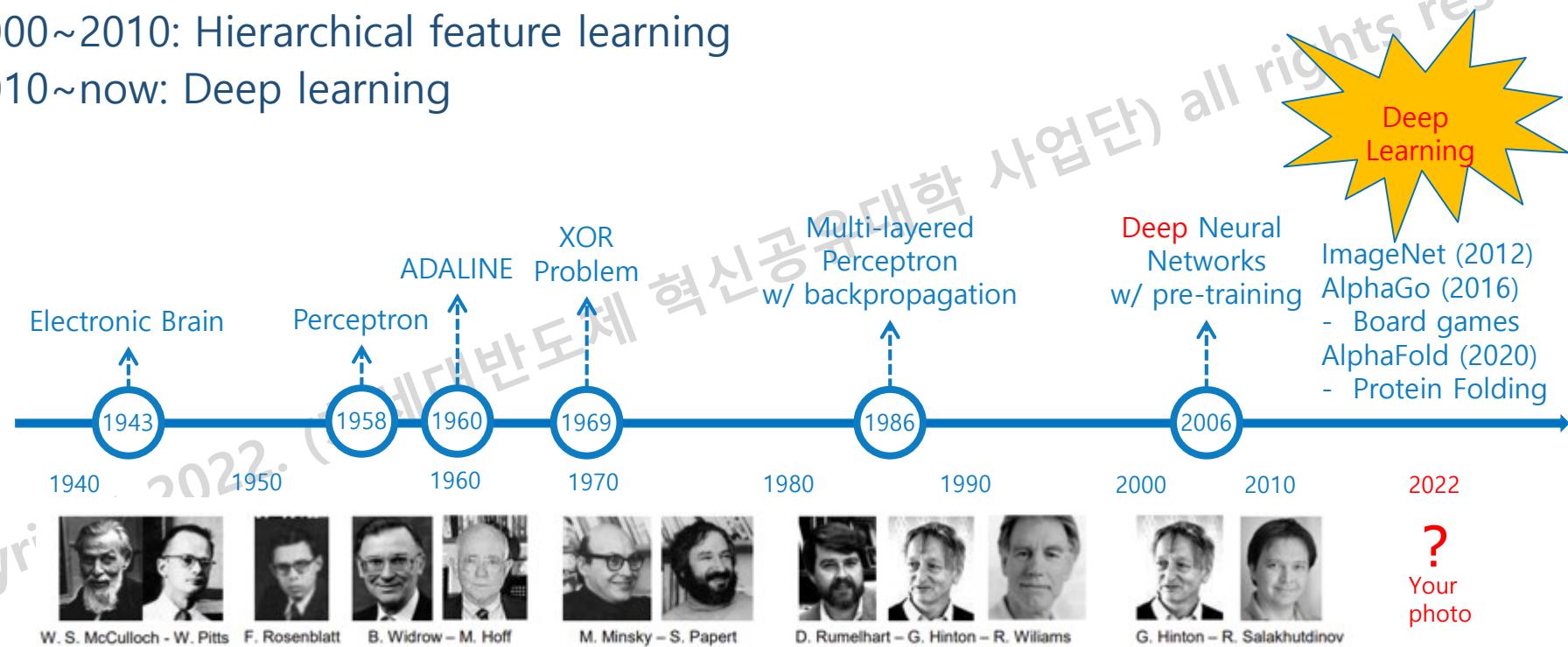
# Applications for Hardware Accelerators

- AI: Any technique that enables computer to mimic human behaviours.
- Machine learning (ML): AI techniques that have computers learn without being explicitly programmed.
- Deep learning (DL): A subset of ML techniques that makes computation of multi-layered/deep neural networks feasible



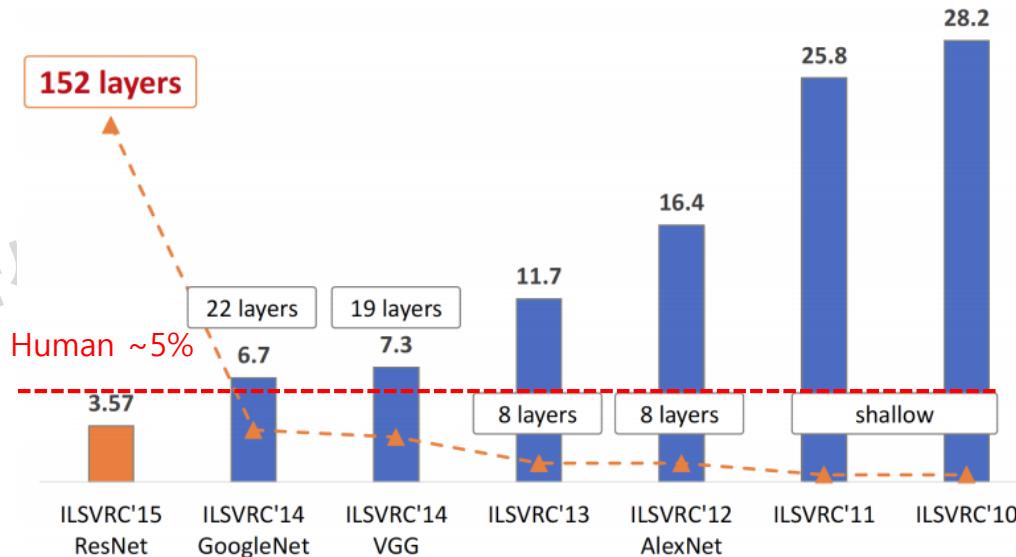
# History of Neural Networks

- Neural Networks have been developed and studied for many decades
  - 1940~1950: adjustable but not learnable
  - 1950~1970: learnable weights and threshold
  - 1980~1990: solved nonlinear problem, high computation, local optimal
  - 2000~2010: Hierarchical feature learning
  - 2010~now: Deep learning



# Deep Learning Revolution

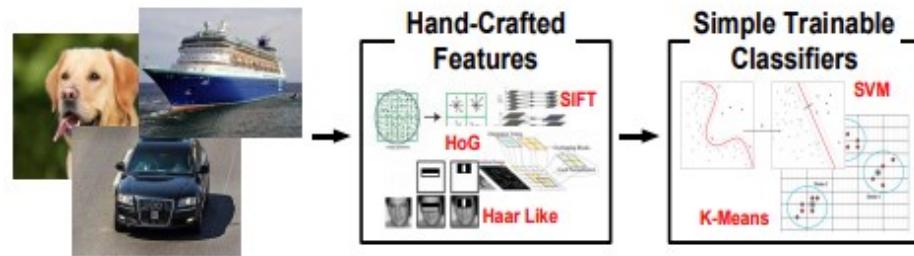
- Deep learning starts to surpass human-level recognition on specific tasks.
  - Human-level recognition: ~5% error
  - Deep learning
    - 2012: AlexNet, 8 layers, 16.4% error.
    - 2014: VGG, 19 layers, 7.3% error
    - 2014: GoogleNet, 22 layers, 6.7% error
    - 2015: ResNet, 152 layers, 3.57% error



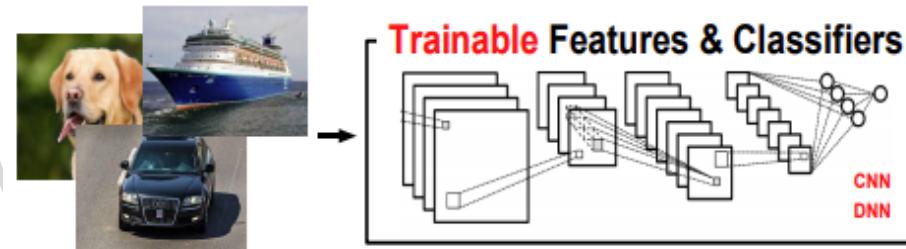
Deep residual learning for image  
recognition, 2017.

# Deep Learning Revolution

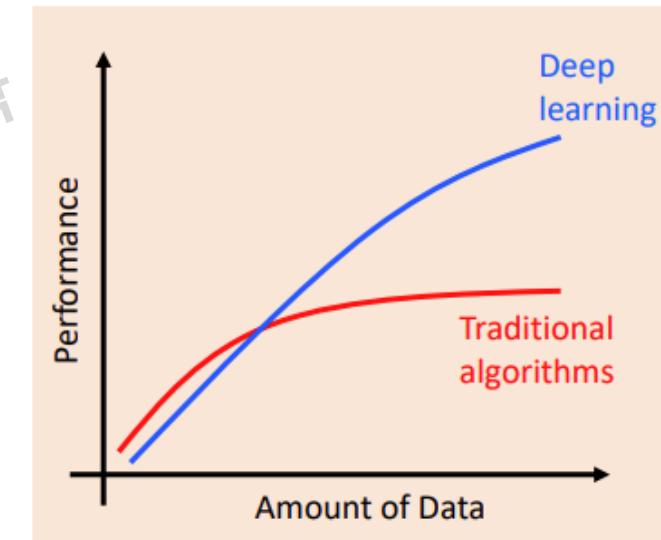
- Traditional Recognition: Hand-crafted features + simple classifiers
  - Performance becomes saturated when the amount of data increases.
- Deep Learning (DL): Trainable/Learnable features and classifiers
  - Performance dramatically increases when the amount of data increases.



(a) Traditional



(b) Deep Learning

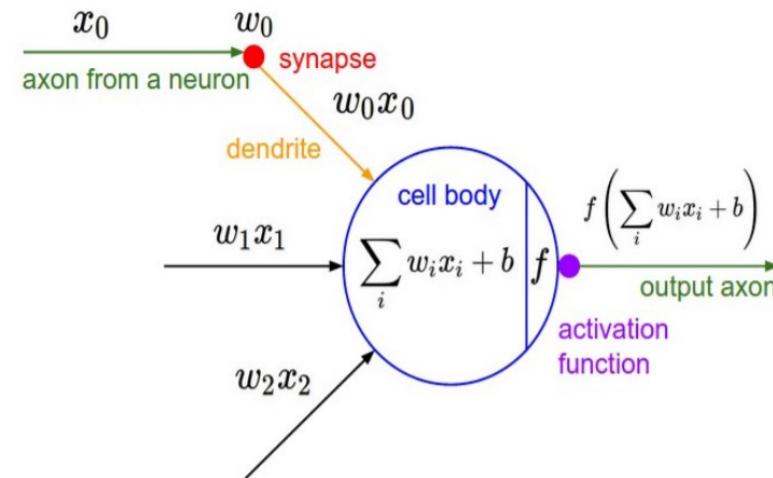
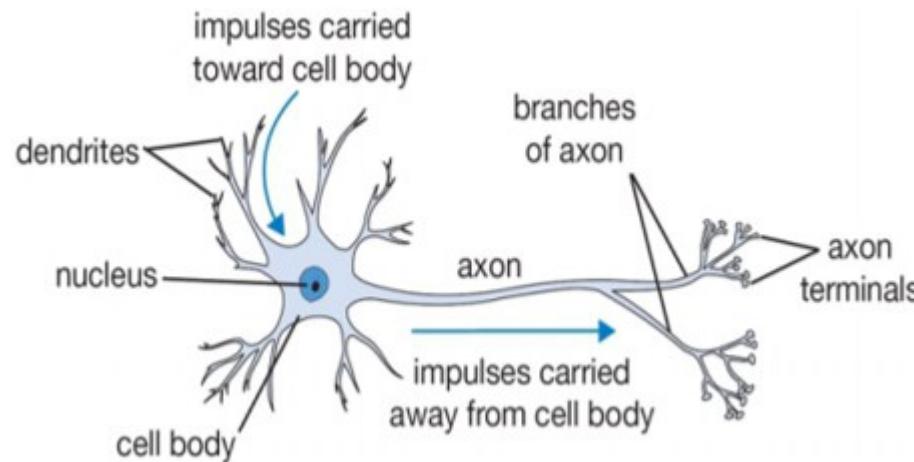


(c) Performance vs Data

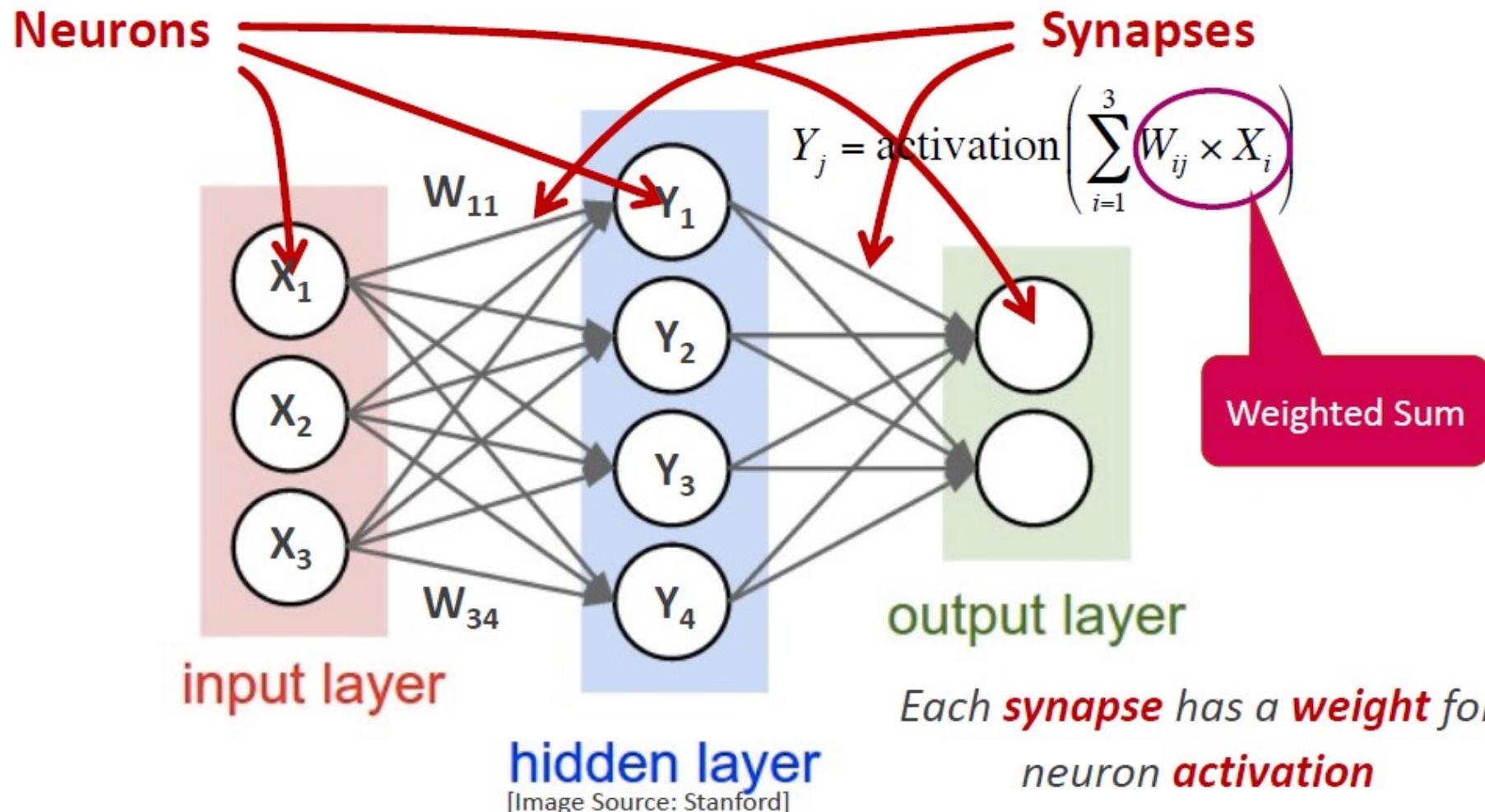
Courtesy to Andrew Ng, Stanford CS229 class

# Neural Networks

- Neuron: The basic computational unit of the brain (~86B)
- Synapses (Weights) connect neurons: Nearly  $10^{14} - 10^{15}$  synapses
  - Learnable & control influence strength
- Neural networks:
  - Neurons receive input signal from dendrites and produce output signal along axon
  - Axons interact with the dendrites of other neurons via synaptic weights
- Artificial Neural networks try to mimic human brain's behaviors

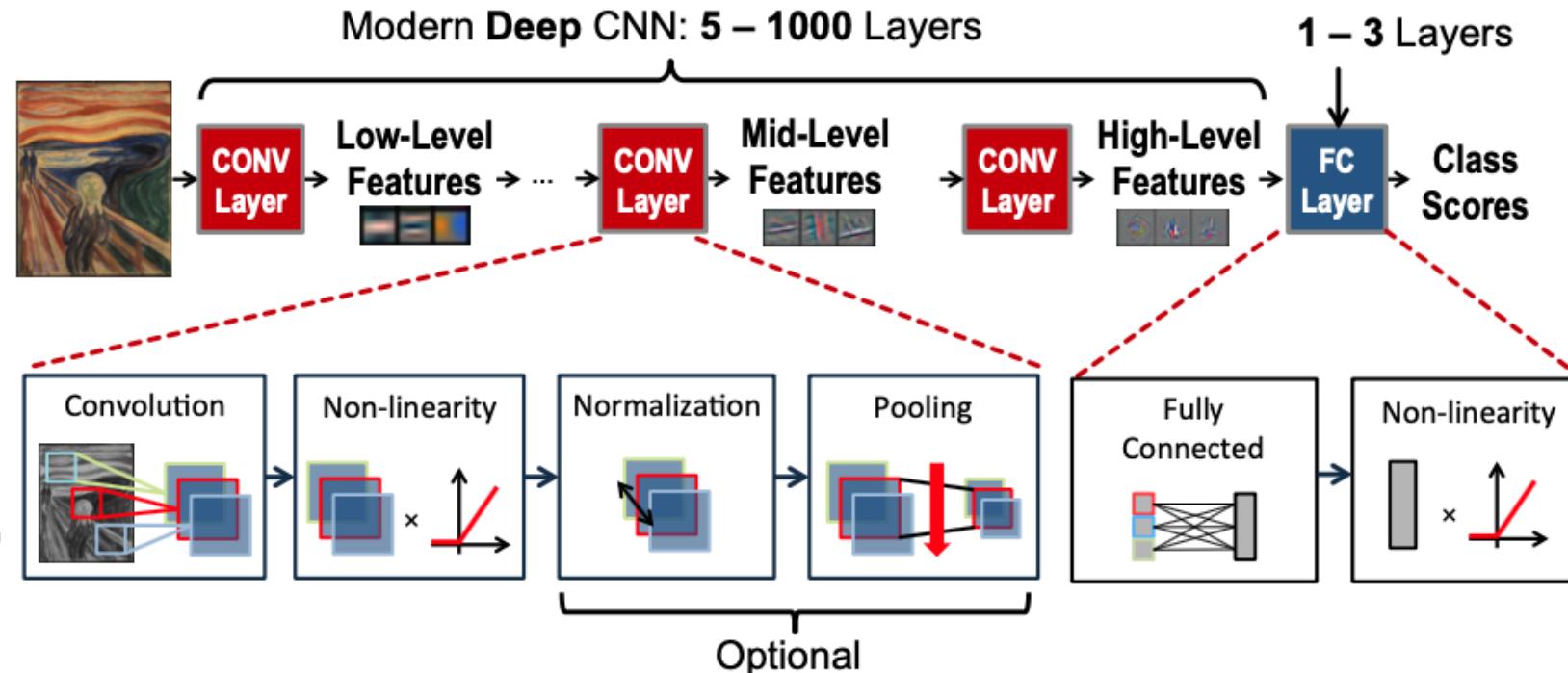


# What is a deep neural network?

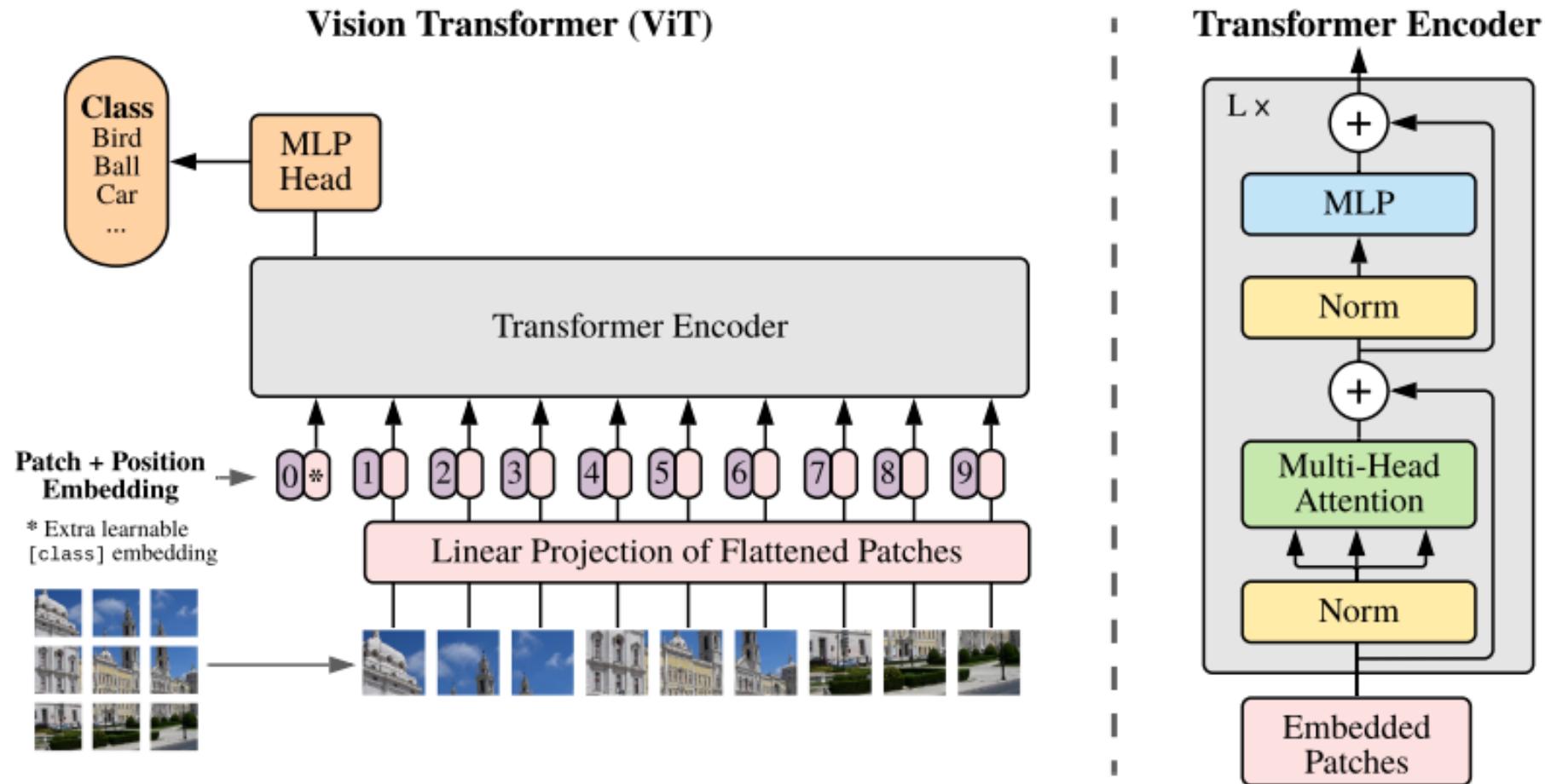


# Modern Deep Neural Networks

- Deep -> multiple layers
  - Convolutional layer followed by non-linearity activation
  - Each layer may include normalization or pooling
  - For classification task, it usually ends with a few full-connected layer.



# Modern Deep Neural Networks



# Road map

Deep Learning Accelerator

Convolutional Neural  
Network (CNN)

Quantization

Block memory

Reference S/W

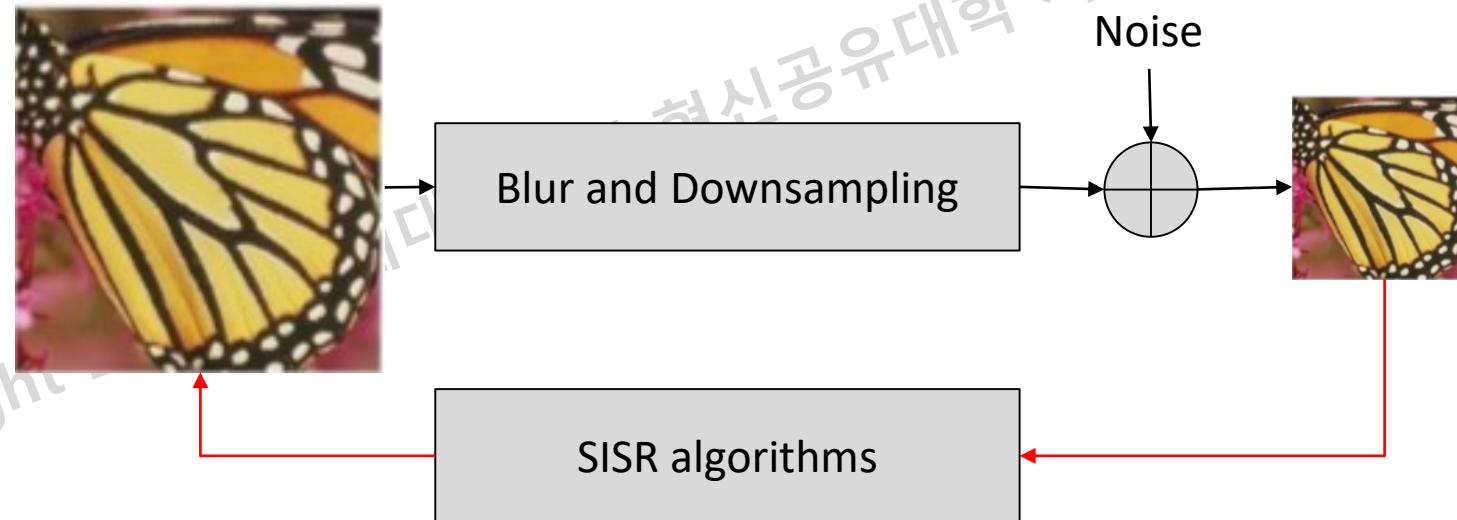
# Camera Field of View (FoV)

- The **field of view (FoV)** is the extent of the observable world that is seen at any given moment
  - In the case of optical sensors, it is a solid angle through which a detector is sensitive to electromagnetic radiation
- **Resolution:** An FoV can be represented by different grids of pixels.
  - Example: 640x480 (VGA), 1280x1080 (HD)



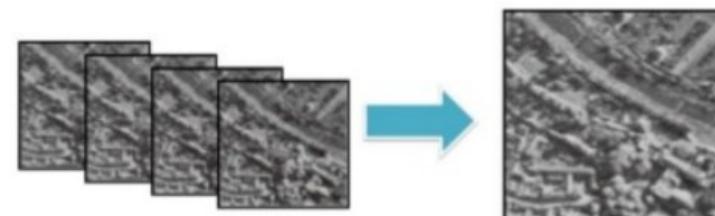
# Image Super-resolution

- Image Super-Resolution **recovers a High Resolution (HR)** image from a given Low Resolution (LR) image.
- An image may have a “lower resolution” due to a smaller spatial resolution (i.e. size) or due to a result of degradation (such as blurring).

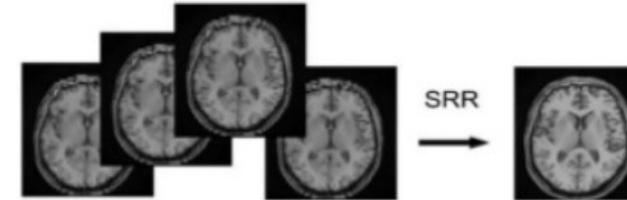


# Applications

- Satellite image processing
- Medical image processing
  - MRI
- Multimedia industry and video enhancement
  - TV, monitor

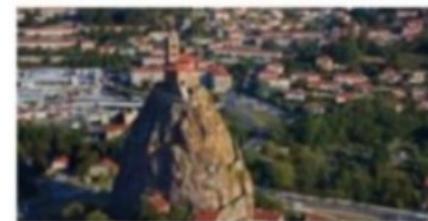


SR for satellite image

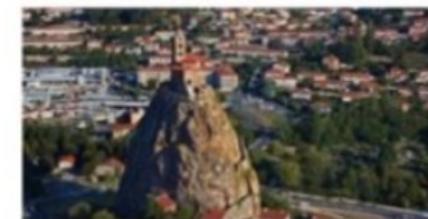
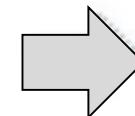


SR for medical imaging

TV & Monitor



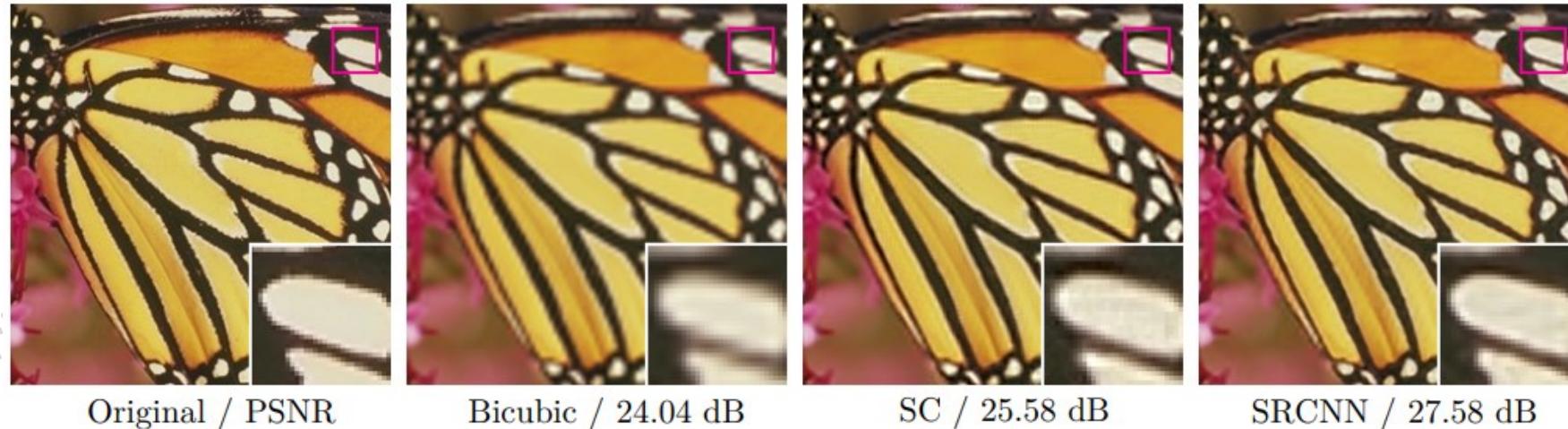
Before  
HD(1280x720), FHD(1920x1080)



AI Technology  
UHD(3840x2160)

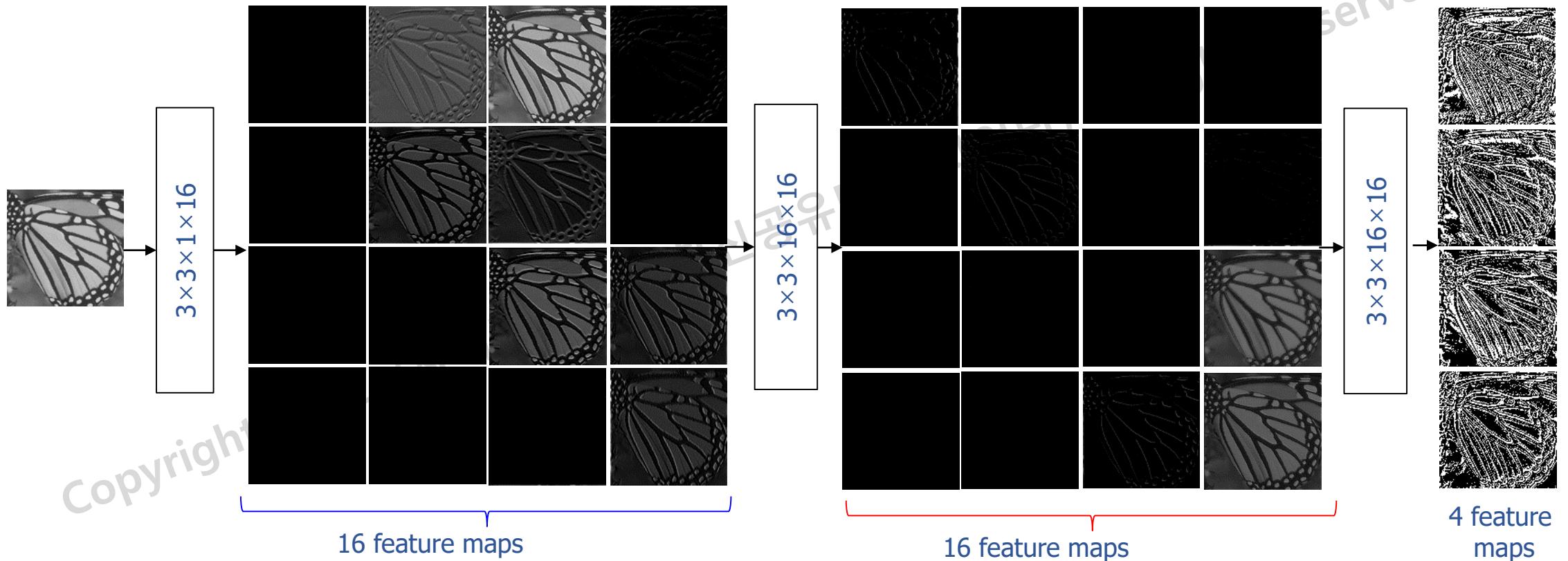
# Image Super-resolution

- Estimating the inverse degradation function is an ill-posed problem
  - Many solutions
- Methods
  - Interpolation-based methods
  - Reconstruction-based methods
  - (Deep) Learning-based methods



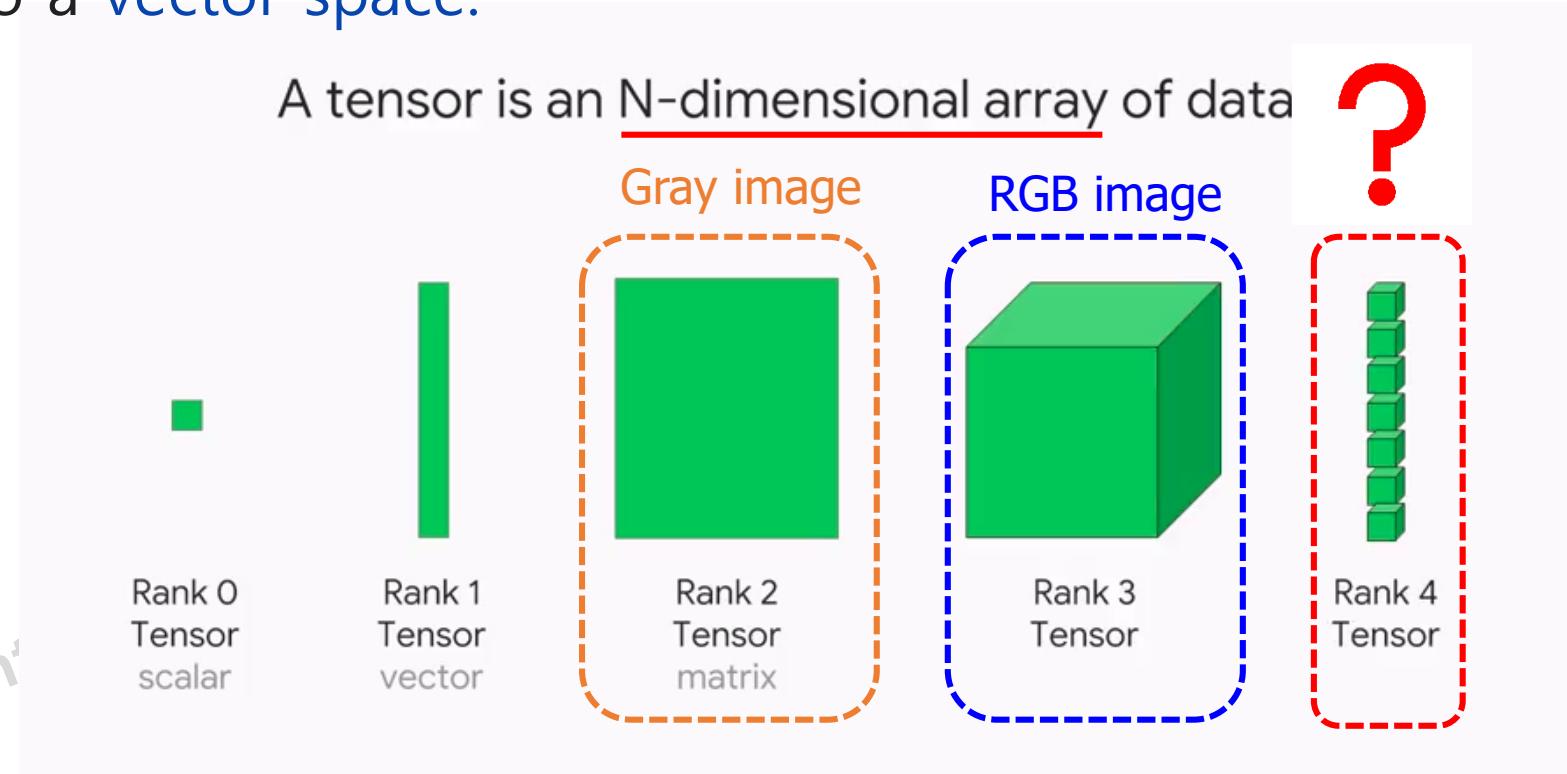
# Simple CNN for SR

- A simple CNN for SR has three convolutional layers
  - Input is a LR image
  - All three layers use  $3 \times 3$  CONV filters



# Tensor

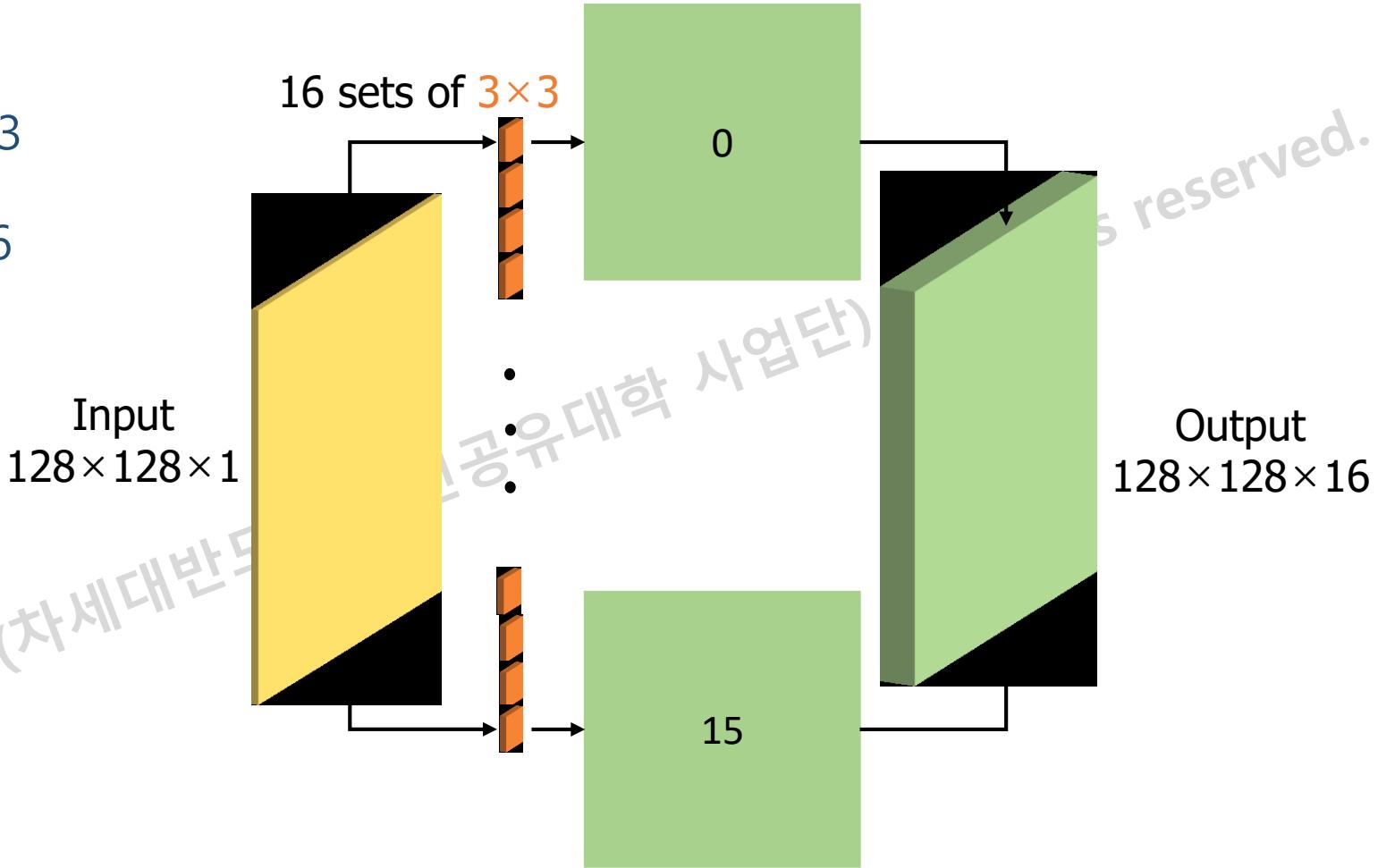
- From Wikipedia<sup>(1)</sup> : “In mathematics, a tensor is an algebraic object that describes a multilinear relationship between sets of algebraic objects related to a vector space.”



Source: <https://medium.com/mlait/tensors-representation-of-data-in-neural-networks-bbe8a711b93b>

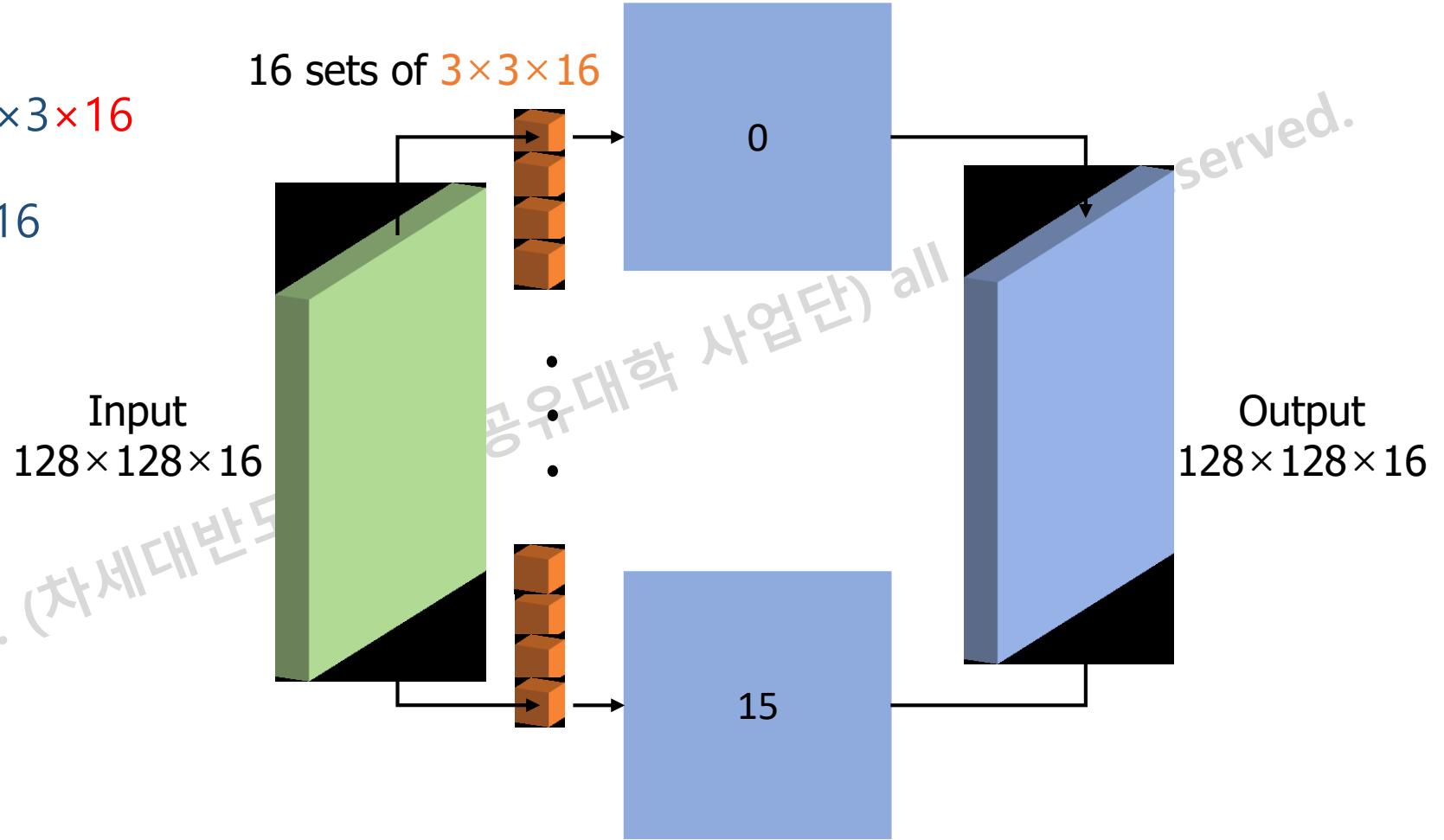
# Convolution: Layer 1

- Input:  $128 \times 128 \times 1$
- Filters: 16 sets of  $3 \times 3$
- Output:  $128 \times 128 \times 16$



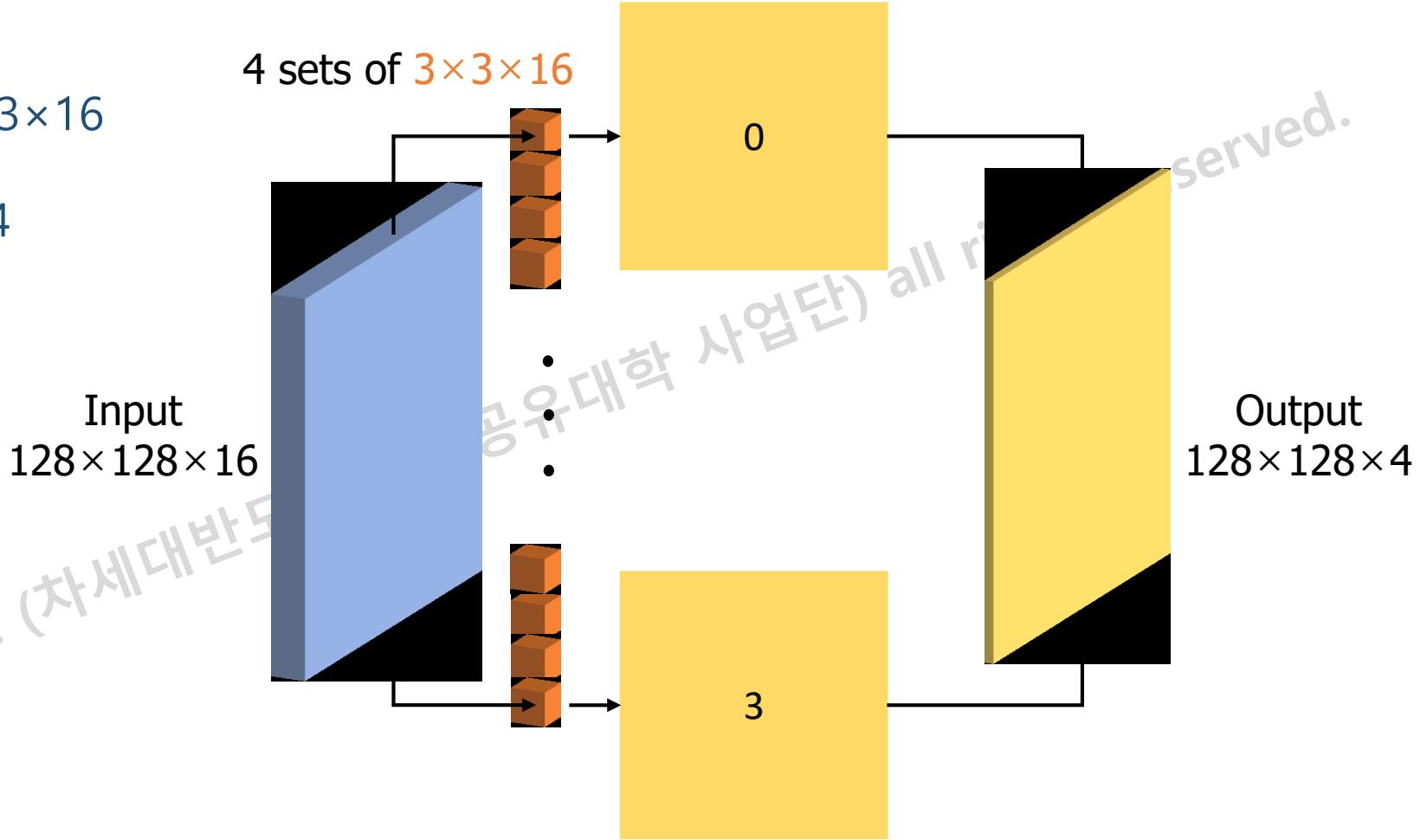
# Convolution: Layer 2

- Input:  $128 \times 128 \times 16$
- Filters: 16 sets of  $3 \times 3 \times 16$
- Output:  $128 \times 128 \times 16$



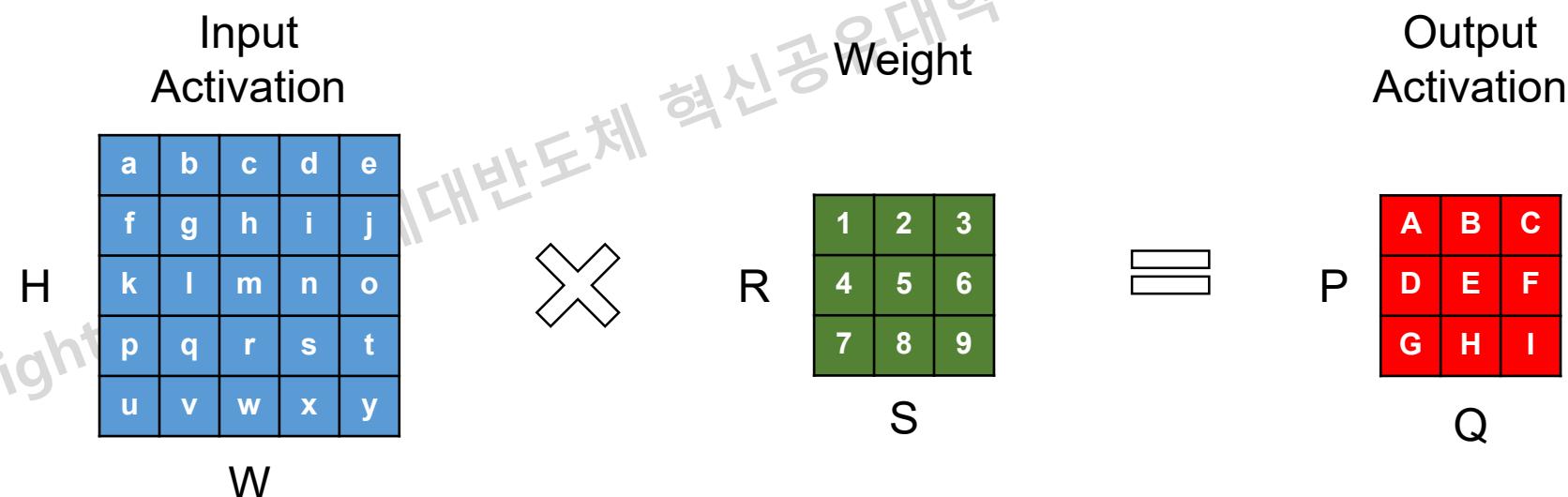
# Convolution: Layer 3

- Input:  $128 \times 128 \times 16$
- Filters: **4 sets** of  $3 \times 3 \times 16$
- Output:  $128 \times 128 \times 4$



# 2D convolution

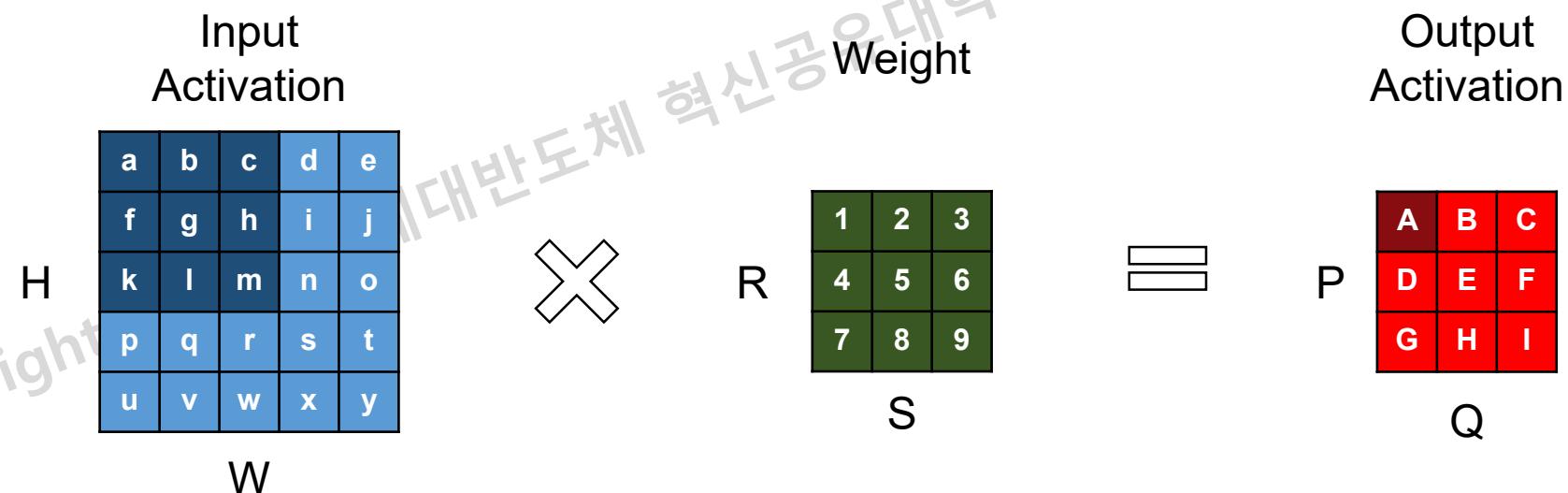
- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)



# 2D convolution

- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)

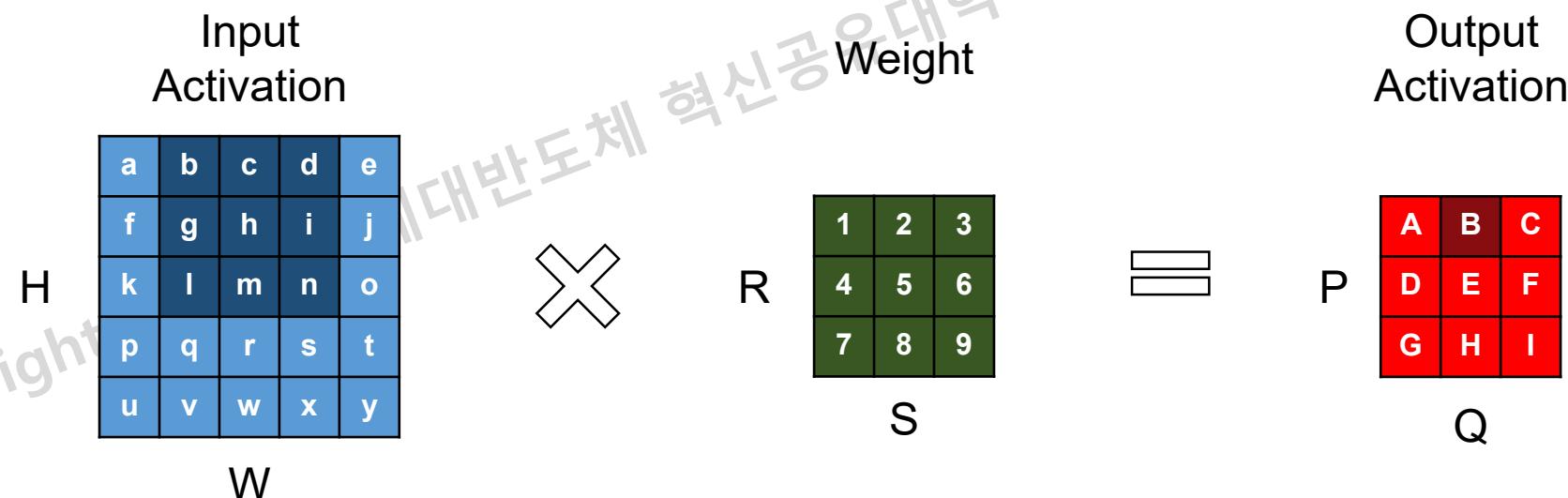
$$A = a * 1 + b * 2 + c * 3 \\ + f * 4 + g * 5 + h * 6 \\ + k * 7 + l * 8 + m * 9$$



# 2D convolution

- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)

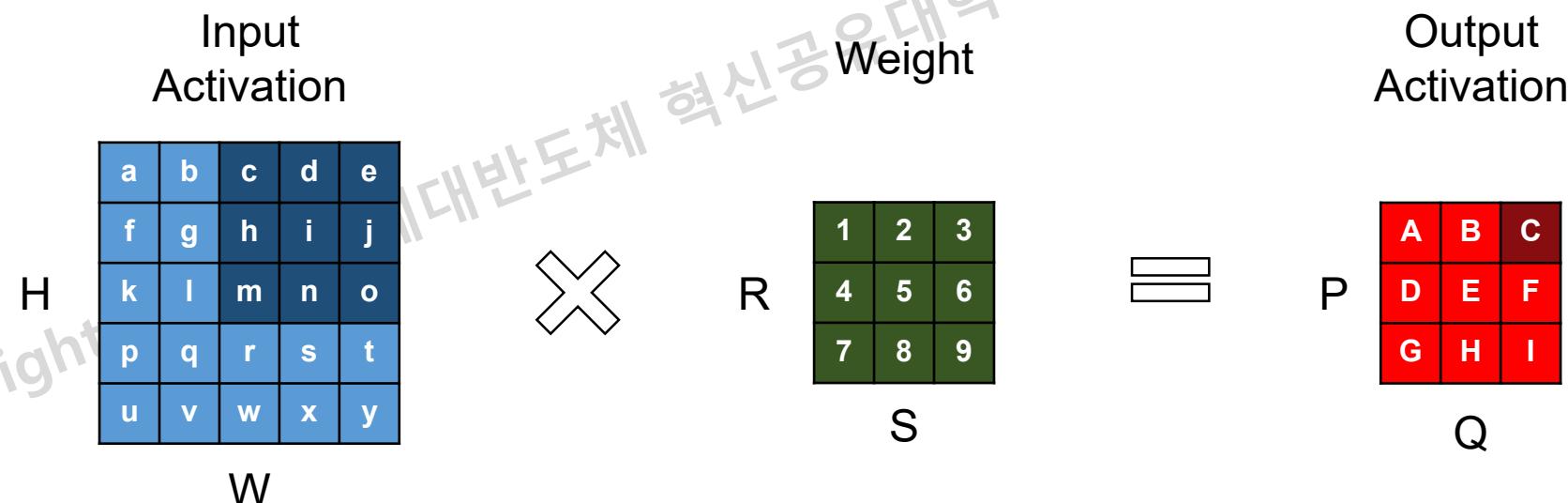
$$\begin{aligned}B = & b * 1 + c * 2 + d * 3 \\& + g * 4 + h * 5 + i * 6 \\& + l * 7 + m * 8 + n * 9\end{aligned}$$



# 2D convolution

- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)

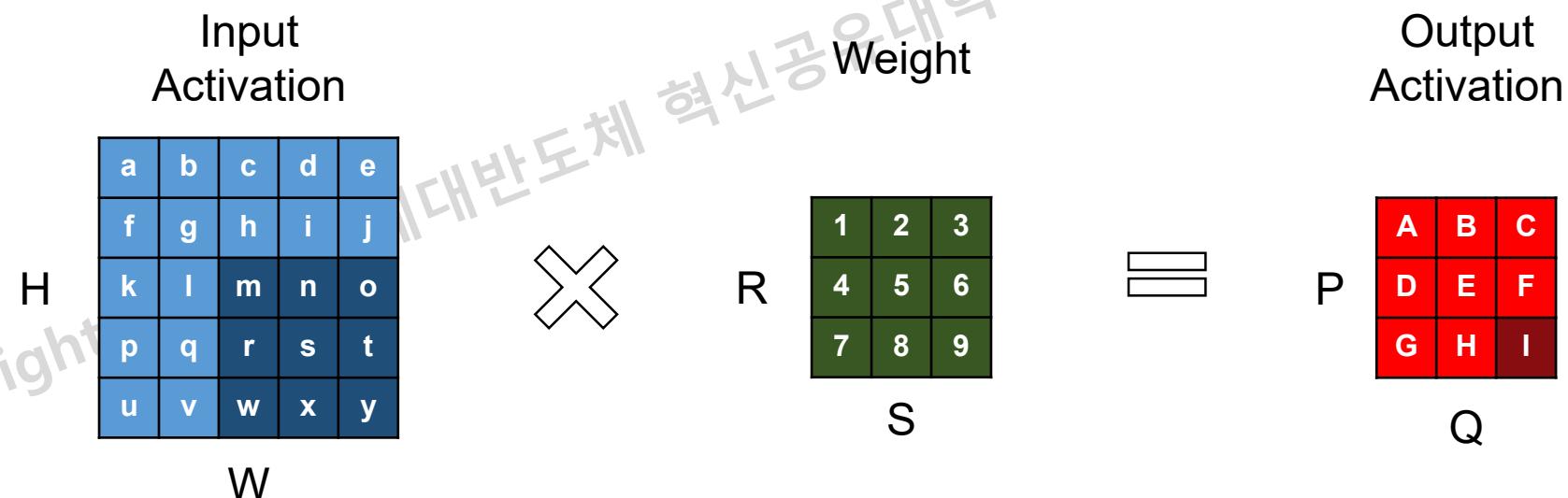
$$\begin{aligned}C = & c * 1 + d * 2 + e * 3 \\& + h * 4 + i * 5 + j * 6 \\& + m * 7 + n * 8 + o * 9\end{aligned}$$



# 2D convolution

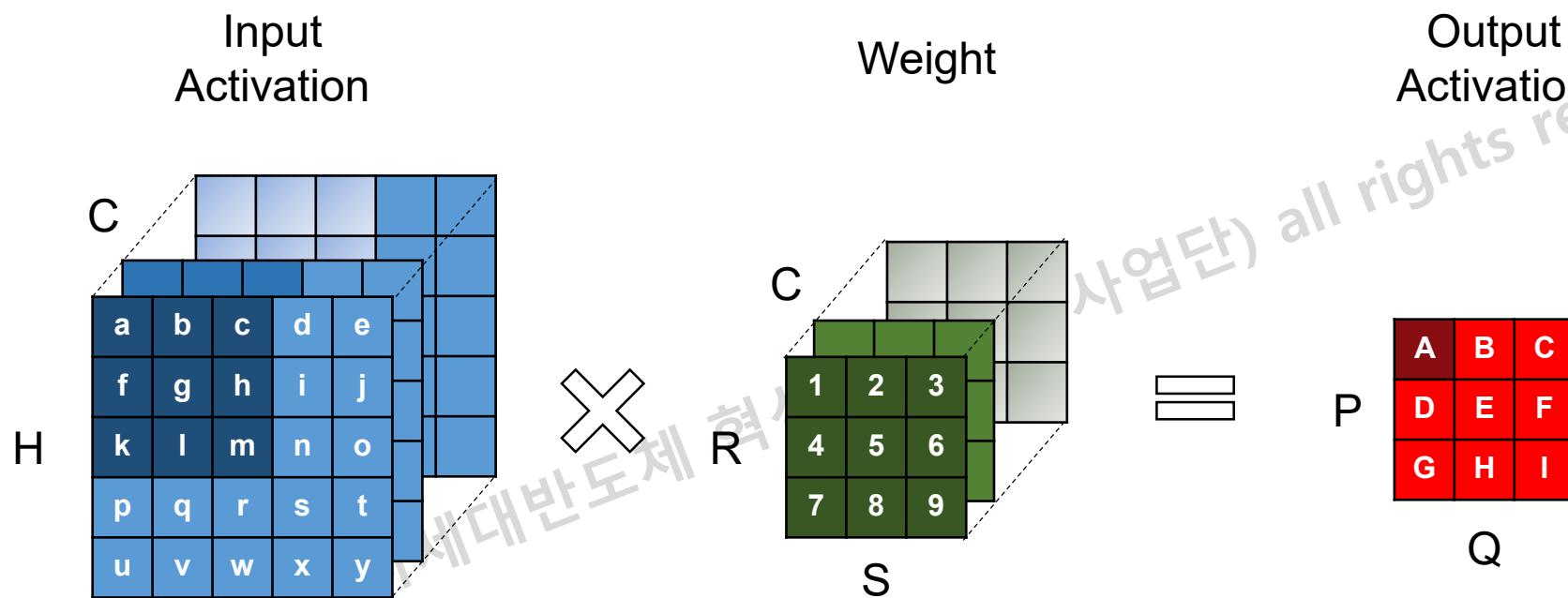
- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)

$$I = m * 1 + n * 2 + o * 3 + r * 4 + s * 5 + t * 6 + w * 7 + x * 8 + y * 9$$

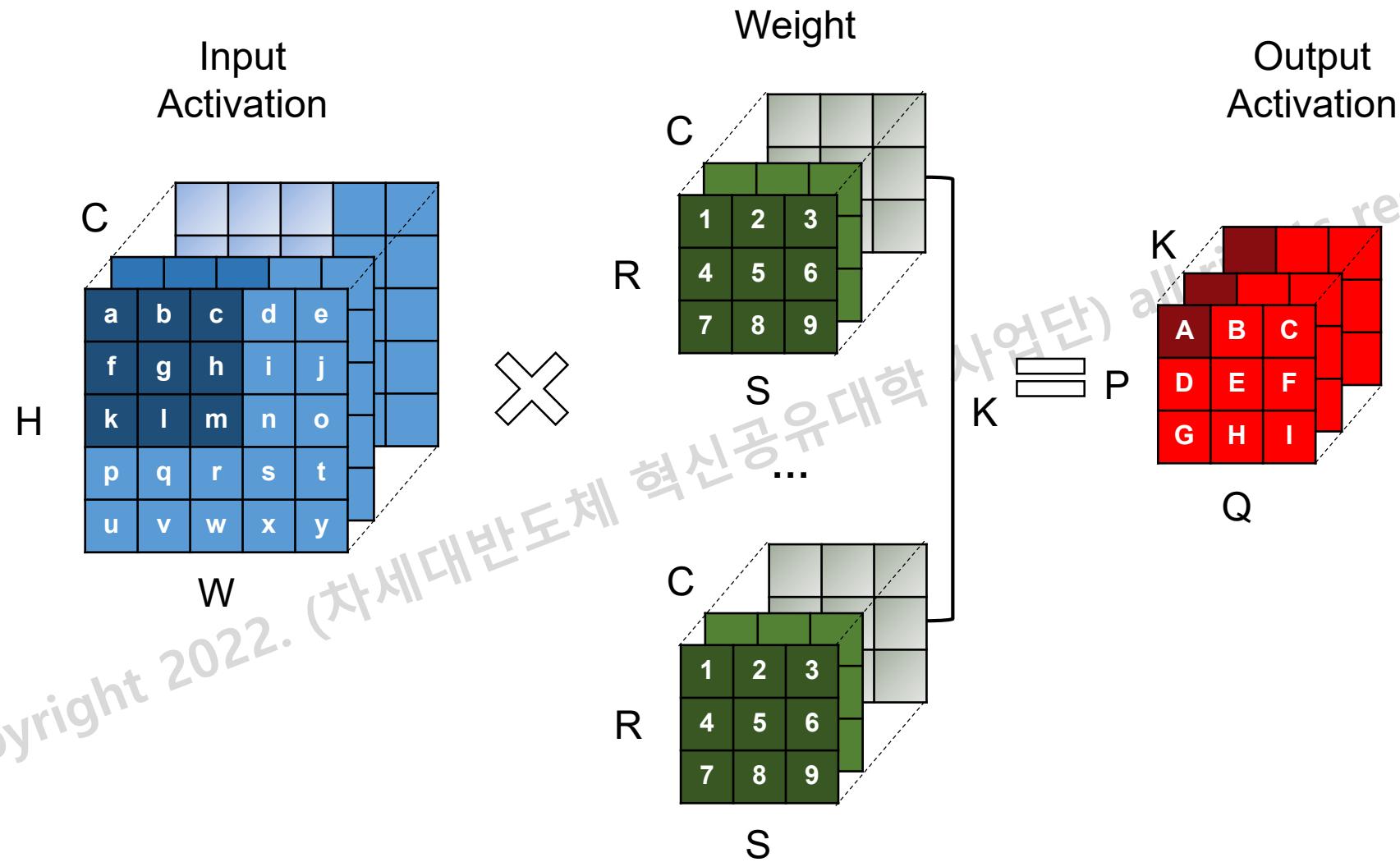


# 3-D Convolution: Multiple input channels

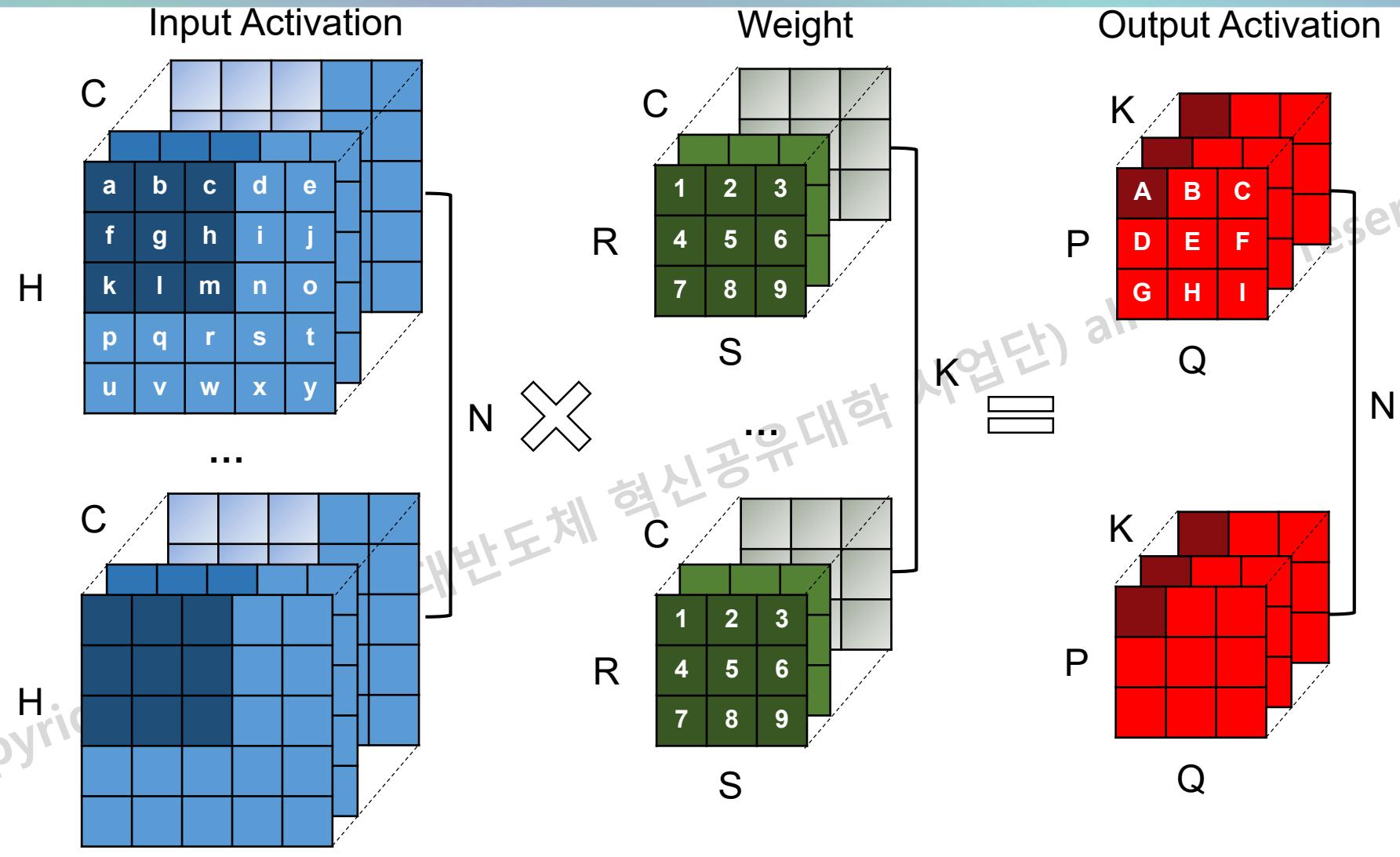
- **C: # of Input Channels**



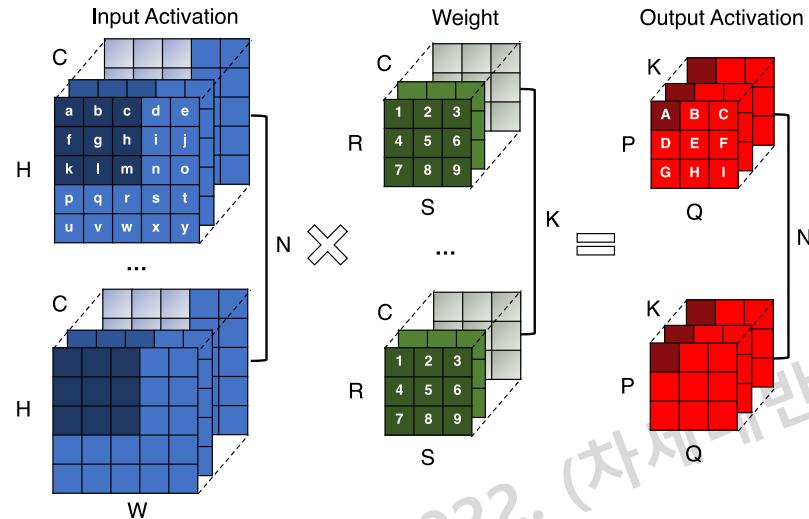
# 3-D Convolution: multiple output channels



# Batch: Multiple inputs



# CONV loop

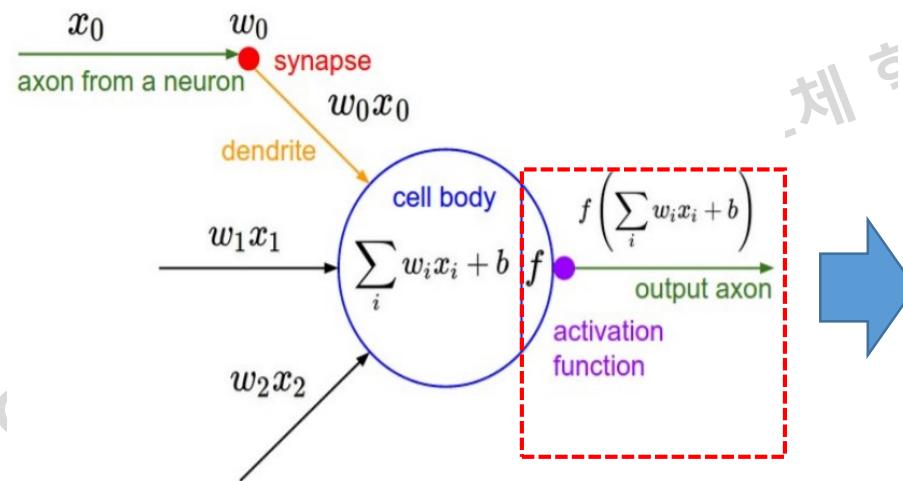


```
for (n=0; n<N; n++) {  
    for (k=0; k<K; k++) {  
        for (p=0; p<P; p++) {  
            for (q=0; q<Q; q++) {  
                OA[n][k][p][q] = 0;  
                for (r=0; r<R; r++) {  
                    for (s=0; s<S; s++) {  
                        for (c=0; c<C; c++) {  
                            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];  
                        }  
                    }  
                }  
            }  
        }  
    }  
}  
OA[n][k][p][q] = Activation(OA[n][k][p][q]);  
}
```

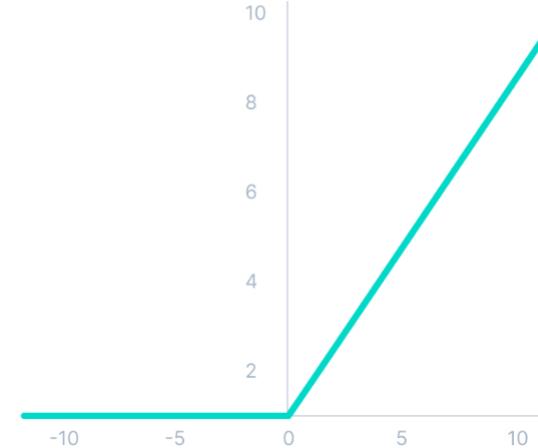
}      for each output activation  
}      Convolution window

# Activation/Transfer function

- Activation/Transfer: Depending on the nature and intensity of these input signals, the brain processes them and decides whether the neuron should be activated ("fired") or not.
- The primary role of the **Activation Function** is to transform the summed weighted input from the node into an output value to be fed to the next hidden layer or as output.
- The purpose of an activation function is to add non-linearity to the neural network.



$$\text{ReLU: } f(x) = \max(0, x)$$



$$\text{Linear: } f(x) = x$$



# Lab 1: Convolution

- Lab 1: Implement a convolution function
- Motivation: Before hardware implementation, we need to develop a reference software
- Why?
  - Software: abstract, high-level representation of a function, an algorithm
  - Generate a test vector for hardware verification

```
% Function: Convolution Only
% Input
%   - img: Input feature map
%   - kernel: Weight
%   - s: stride
%   - p: padding
% Output
%   - out
function out = convol2(img,kernel,s,p)
```

# Convolution Function: Data preparation

- Input Buffers
  - Padded input image for handling boundary pixels

```
function out = convol2(img,kernel,s,p)
    % Input Feature Maps
    h = size(img,1);      % height
    w = size(img,2);      % width
    c = size(img,3);      % number of input features

    % Padded image: zero padding
    pad_img = zeros(h+p,w+p,c);
    st_p = ceil(p/2);
    pad_img(1+st_p:st_p+h,1+st_p:st_p+w,:) = img;

    % Filter kernel size
    f = size(kernel,1);

    % Output feature maps
    c_out = size(kernel,4);           % Number of output features
    w_out = floor((w - f + p) / s + 1); % width
    h_out = floor((h - f + p) / s + 1); % height

    out = zeros(h_out,w_out,c_out);    % buffer of output feature maps
```

Copyright 2022  
All rights reserved.

# Convolution Function: Data preparation

- Output Buffers
  - Calculate the sizes
  - Allocate a buffer/variable for output feature maps

```
function out = convol2(img,kernel,s,p)
% Input Feature Maps
h = size(img,1);      % height
w = size(img,2);      % width
c = size(img,3);      % number of input features

% Padded image: zero padding
pad_img = zeros(h+p,w+p,c);
st_p = ceil(p/2);
pad_img(1+st_p:st_p+h,1+st_p:st_p+w,:) = img;

% Filter kernel size
f = size(kernel,1);

% Output feature maps
c_out = size(kernel,4);           % Number of output features
w_out = floor((w - f + p) / s + 1); % width
h_out = floor((h - f + p) / s + 1); % height

out = zeros(h_out,w_out,c_out);    % buffer of output feature maps
```

# Convolution Function: Data preparation

- Three loops
  - Number of output channels
  - Row: line by line
  - Column: from left to right
- Kernel operations
  - Element-wise multiplication (products)
  - Sum (sum of products)

```
% Convolution
for k = 1:c_out % Number of output channels
    for i = 1:h_out % Row
        for j = 1:w_out % Column
            % Element-wise Multiplication => Products
            scalar = kernel(:,:,:,:,k).* ...
            | pad_img(1+(i-1)*s:1+(i-1)*s+f-1, 1+(j-1)*s:1+(j-1)*s+f-1, :);
            % Sum (Sum of Products)
            out(i,j,k) = sum(scalar(:));
        end
    end
end
```

Copyr

# Test a sim-SR

- Load a pretrained model of sim-SR
  - Parameters: layer 1 (w1, b1), layer 2 (w2, b2), layer 3 (w3, b3)
  - Input files locate at bin/
  - Weights are reorganized in a typical format:

[Filter size, Filter size, input channels, output channels]

```
%% Load pre-trained convolutional parameter
% Load Layer 1
[w1,b1,~,~,~] = read_conv_param_module('bin/sim_espncn_3x3.weights_layer_0',3, 1,16,0);
w1 = permute(reshape(w1, 3, 3, 1, 16), [2, 1, 3, 4]);
% Load Layer 2
[w2,b2,~,~,~] = read_conv_param_module('bin/sim_espncn_3x3.weights_layer_1',3,16,16,0);
w2 = permute(reshape(w2, 3, 3, 16, 16), [2, 1, 3, 4]);
% Load Layer 3
[w3,b3,~,~,~] = read_conv_param_module('bin/sim_espncn_3x3.weights_layer_2',3, 16,4,0);
w3 = permute(reshape(w3, 3, 3, 16,4), [2, 1, 3, 4]);
```

# Test a sim-SR

- Load an input image and do preprocessing

## % Load an image

```
SF = 2; % Up scaling 2x
imGT = imread('img/butterfly_GT.bmp');
if size(imGT,3) > 1
    im = rgb2ycbcr(imGT);
else
    im = imGT;
end
```

```
imhigh = modcrop(im, SF);
imhigh = single(imhigh)/255;
imlow = imresize(imhigh, 1/SF, 'bicubic');
%imlow = imresize(imlow, SF, 'bicubic');
if size(imlow,3)>1
    imlowy = imlow(:,:,1);
    imlowy = max(16.0/255, min(235.0/255, imlowy));
else
    imlowy = imlow;
end
input = imlowy;

figure();
imshow(input);

% Now, we convert an image input to a 8-bit format
input = double(floor(imlowy * 255));
```

- Load an input image
- Transform RGB to YCbCr

- Generate a Low-resolution image  
→ An image input of our network

# Test a sim-SR

- Test a sim-SR with a given input image
  - Apply convolution to the input image

```
%% Demo sim-SR

% First Layer
conv_out = convol2(input, w1, 1, 2);
for j = 1:size(conv_out, 3)
    % Adding bias
    conv_out(:, :, j) = conv_out(:, :, j) + b1(j);
end
% Activation
conv_out_relu = conv_out;
conv_out_relu(conv_out_relu < 0) = 0;

%for i = 1:size(conv_out, 3)
%    figure(1)
%    subplot(1, 2, 1)
%    imshow(conv_out(:, :, i)); title(['ch = ', num2str(i)]);
%    subplot(1, 2, 2)
%    imshow(conv_out_relu(:, :, i)); title('After ReLU');
%    pause(1)
%end

% Insert your code for other layers
```

# Test a sim-SR

- Test a sim-SR with a given input image
  - Add a bias for each channel

```
%% Demo sim-SR

% First Layer
conv_out = convol2(input, w1, 1, 2);
for j = 1:size(conv_out, 3)
    % Adding bias
    conv_out(:, :, j) = conv_out(:, :, j) + b1(j);
end
% Activation
conv_out_relu = conv_out;
conv_out_relu(conv_out_relu < 0) = 0;

%for i = 1:size(conv_out, 3)
%    figure(1)
%    subplot(1, 2, 1)
%    imshow(conv_out(:, :, i)); title(['ch = ', num2str(i)]);
%    subplot(1, 2, 2)
%    imshow(conv_out_relu(:, :, i)); title('After ReLU');
%    pause(1)
%end

% Insert your code for other layers
```

# Test a sim-SR

- Test a sim-SR with a given input image
  - Apply ReLU on the output

```
%% Demo sim-SR

% First Layer
conv_out = convol2(input, w1, 1, 2);
for j = 1:size(conv_out, 3)
    % Adding bias
    conv_out(:,:,j) = conv_out(:,:,j) + b1(j);
end
% Activation
conv_out_relu = conv_out;
conv_out_relu(conv_out_relu < 0) = 0;

%for i = 1:size(conv_out, 3)
%    figure(1)
%    subplot(1,2,1)
%    imshow(conv_out(:,:,i));title(['ch = ',num2str(i)]);
%    subplot(1,2,2)
%    imshow(conv_out_relu(:,:,i));title('After ReLU');
%    pause(1)
%end

% Insert your code for other layers
```

# To do ...

- Complete the missing code (CONV kernel operation) in convol2.m
- Complete the missing code in test\_simSR.m:
  - Run for all three layers
  - Last layer uses *linear activation*, two first layers use *ReLU activation*
  - Show the feature maps
- Refer to figures in **featuremaps** folder for sample feature maps
  - (LayerNumber)\_(ChannelNumber).jpg

# Complete the missing code

```
function out = convol2(img,kernel,s,p)
    % Input Feature Maps
    h = size(img,1);      % height
    w = size(img,2);      % width
    c = size(img,3);      % number of input features

    % Padded image: zero padding
    pad_img = zeros(h+p,w+p,c);
    st_p = ceil(p/2);
    pad_img(1+st_p:st_p+h,1+st_p:st_p+w,:) = img;

    % Filter kernel size
    f = size(kernel,1);

    % Output feature maps
    c_out = size(kernel,4);           % Number of output features
    w_out = floor((w - f + p) / s + 1); % width
    h_out = floor((h - f + p) / s + 1); % height

    out = zeros(h_out,w_out,c_out);    % buffer of output feature maps

    % Convolution
    for k = 1:c_out                % Number of output channels
        for i = 1:h_out              % Row
            for j = 1:w_out          % Column
                % Insert your code
            end
        end
    end
end
```

convol2.m

```
%% Demo sim-SR
% First Layer
conv_out = convol2(input, w1, 1, 2);
for j = 1:size(conv_out, 3)
```

% Add bias  
conv\_out(:, :, j) = conv\_out(:, :, j) + b1(j);  
end

```
% Activation
conv_out_relu = conv_out;
conv_out_relu(conv_out_relu < 0) = 0;
```

```
for i = 1:size(conv_out, 3)
    figure(1)
    subplot(1,2,1)
    imshow(conv_out(:, :, i)); title(['ch = ', num2str(i)]);
    subplot(1,2,2)
    imshow(conv_out_relu(:, :, i)); title('After ReLU');
    pause(1)
end
```

% Insert your code for other layers

test\_simSR.m

Cor

# Road map

Deep Learning Accelerator

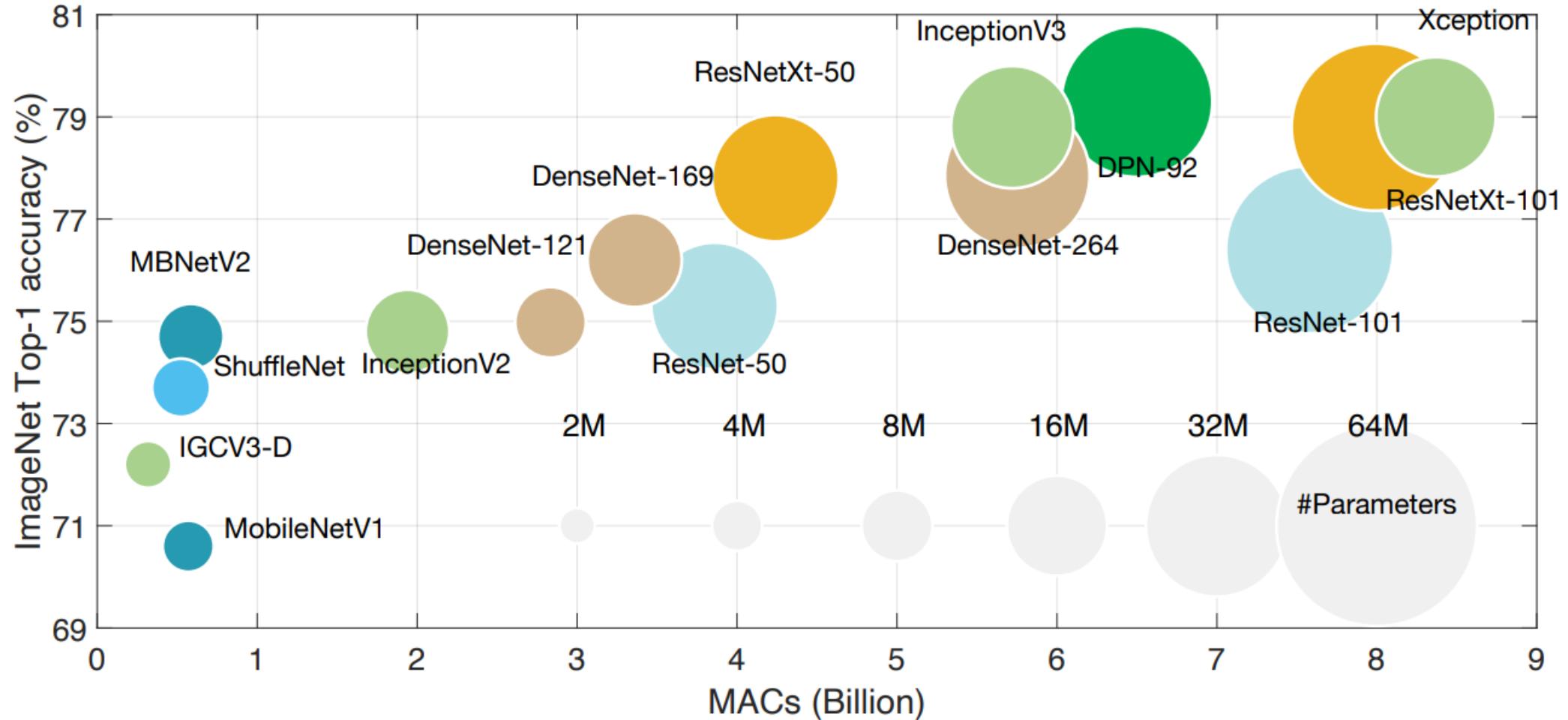
Convolutional Neural  
Network (CNN)

Quantization

Block memory

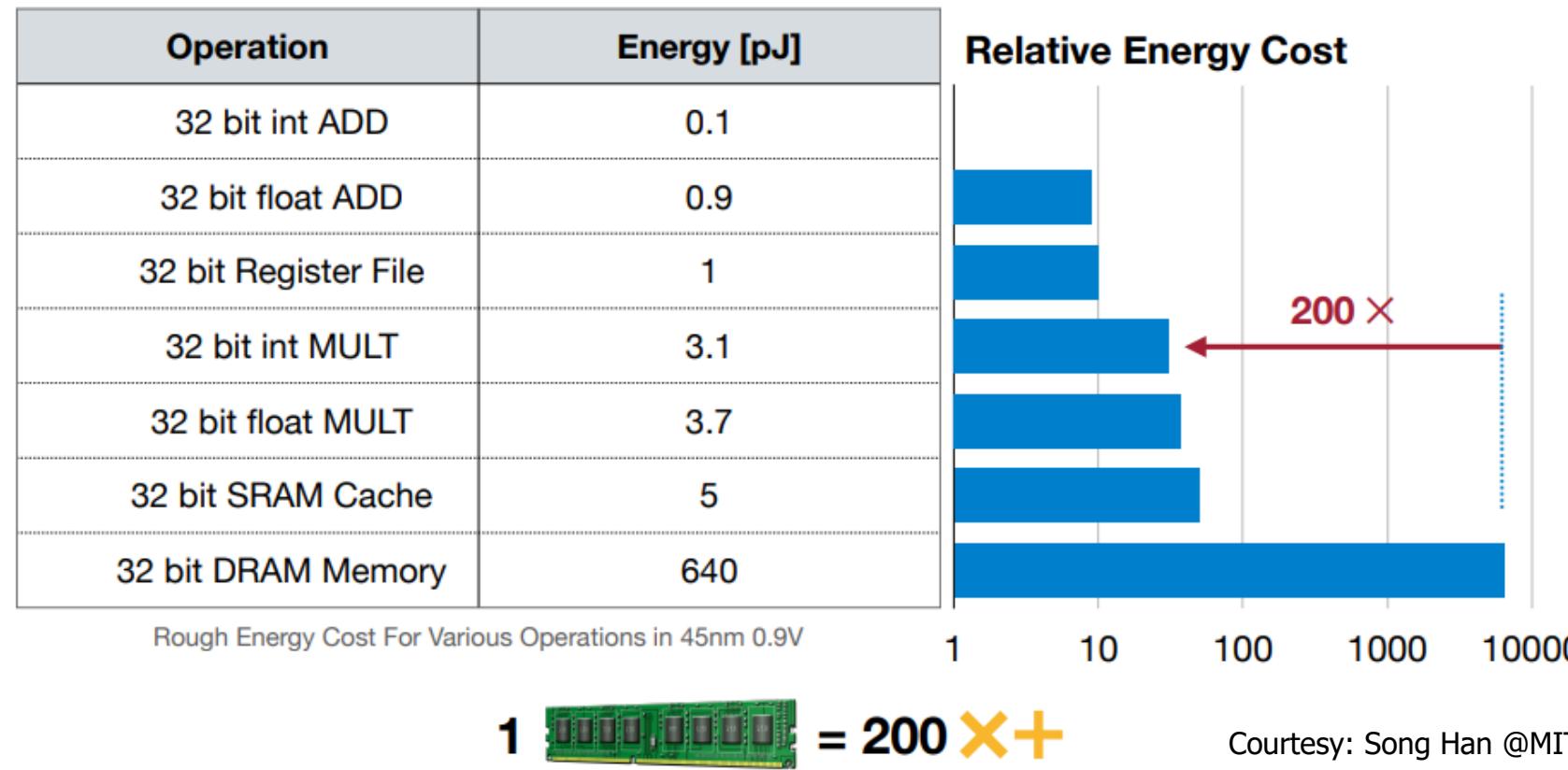
Reference S/W

# Today's AI is too BIG!



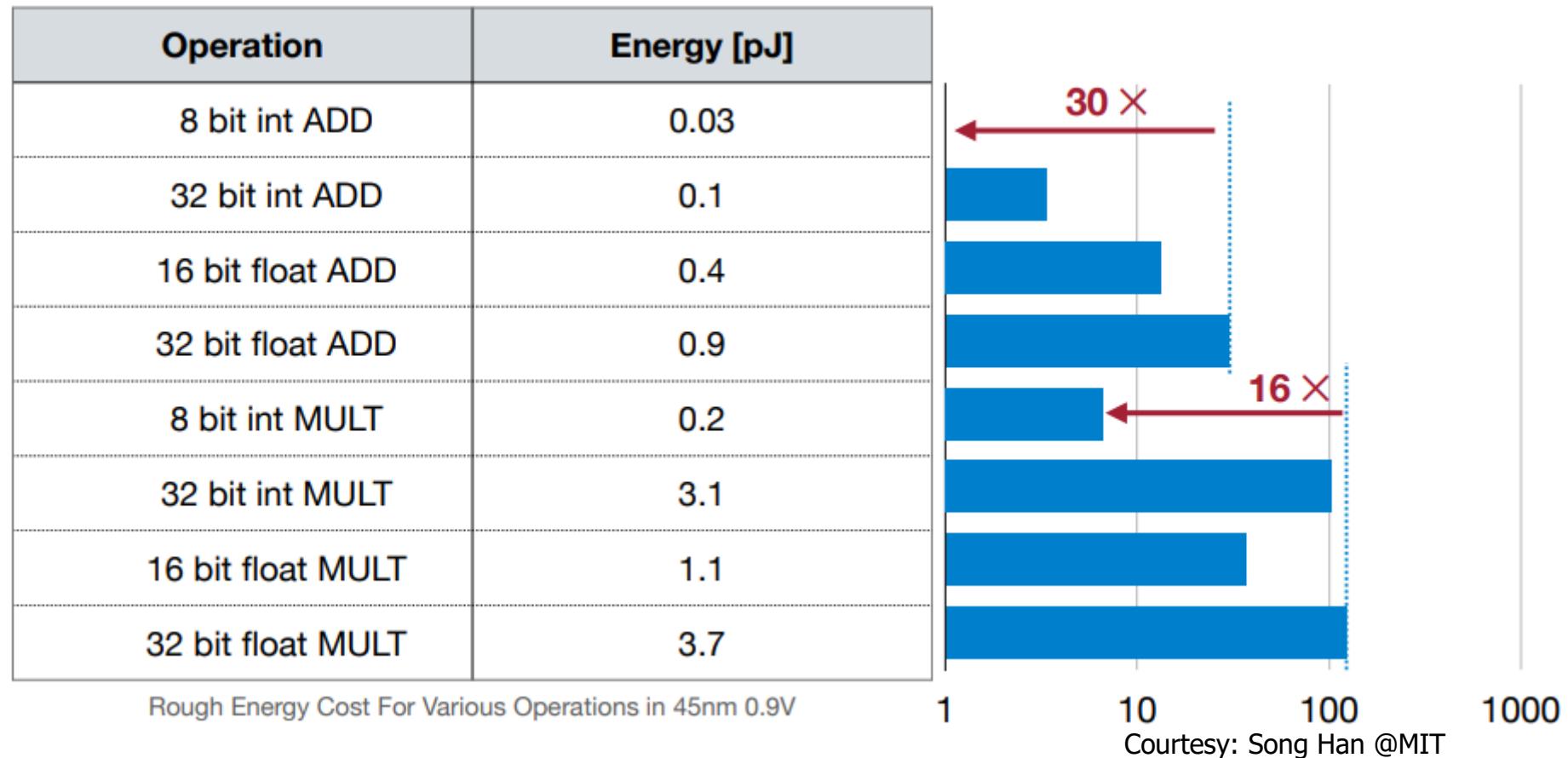
# Memory Access is Expensive

- Data Movement → More Memory Reference → More Energy



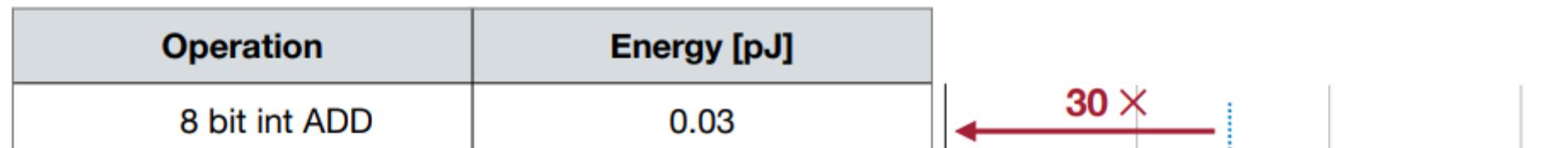
# Low Bit-Width Operations are Cheap

- Less Bit-Width → Less Energy

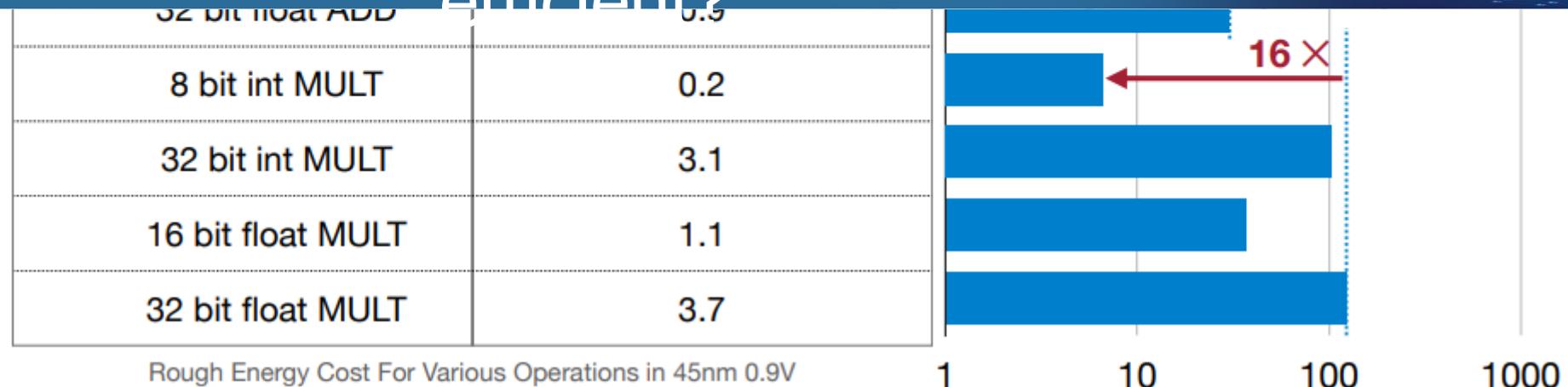


# Low Bit-Width Operations are Cheap

- Less Bit-Width → Less Energy

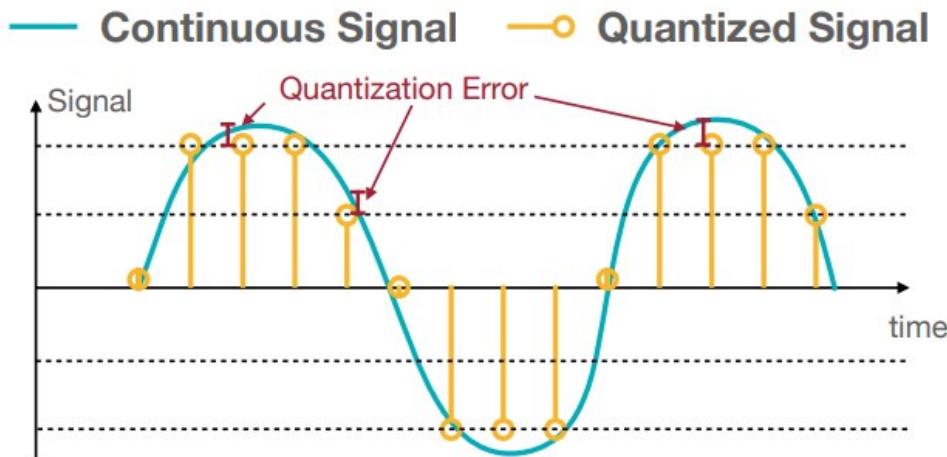


How should we make deep learning more efficient?

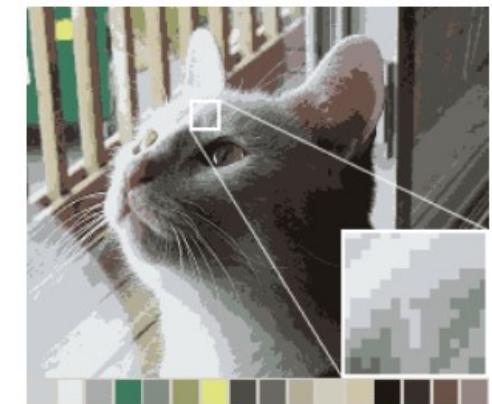
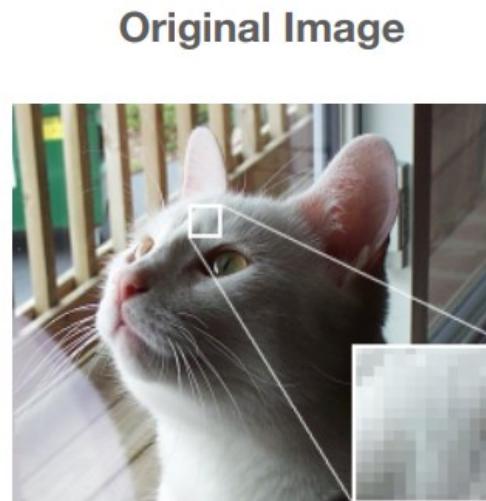


# What is Quantization?

*Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set*



The difference between an input value and its quantized value is referred to as quantization error.



Images are in the public domain.  
“Palettization”

# Numeric Data Types

- How is numeric data represented in modern computing systems?
- Data types
  - Integer
  - Fixed-point number
  - Floating-point number

# Integer

- Unsigned number
  - $n$ -bit: Range:  $[0, 2^n - 1]$
- Signed number
  - *Sign-Magnitude Representation*
  - $n$ -bit Range:  $[-2^{n-1} - 1, 2^{n-1} - 1]$
  - Both 00...0 and 10...0 represent 0
- Two's Complement Representation
  - $n$ -bit Range:  $[-2^{n-1}, 2^{n-1} - 1]$
  - 00...0 represents 0
  - 10...0 represents  $-2^{n-1}$

0	0	1	1	0	0	0	1
x	x	x	x	x	x	x	x

$$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 49$$

Sign Bit

1	0	1	1	0	0	0	1
x	x	x	x	x	x	x	x

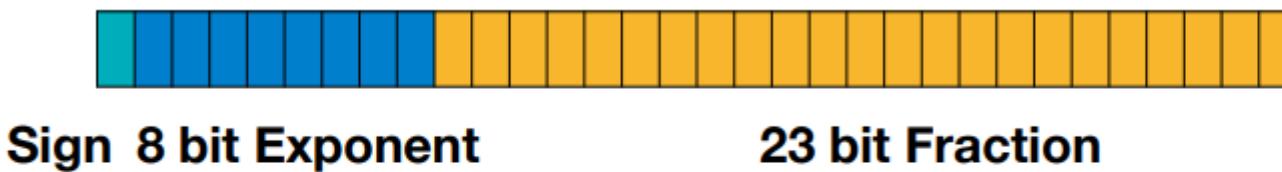
$$-2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$

1	1	0	0	1	1	1	1
x	x	x	x	x	x	x	x

$$-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$

# Floating-Point Number

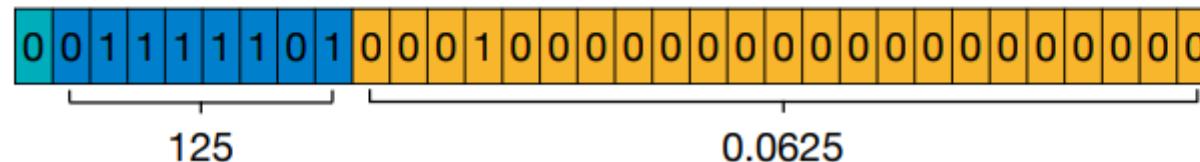
- Example: 32-bit floating-point number in IEEE 754



$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127} \quad \leftarrow \quad \text{Exponent Bias} = 127 = 2^{8-1}-1$$

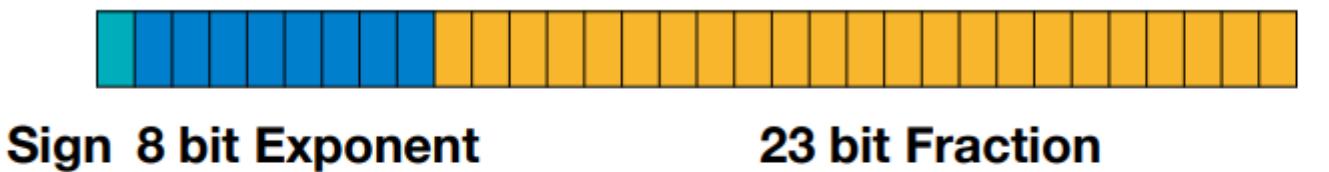
(significant / mantissa)

$$0.265625 = 1.0625 \times 2^{-2} = (1 + 0.0625) \times 2^{125-127}$$



# Floating-Point Number

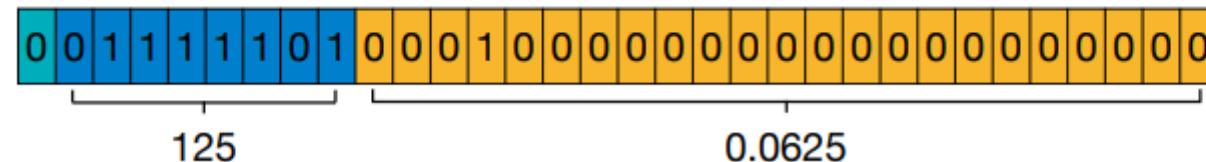
- Example: 32-bit floating-point number in IEEE 754



$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127} \quad \leftarrow \quad \text{Exponent Bias} = 127 = 2^{8-1}-1$$

(significant / mantissa)

$$0.265625 = 1.0625 \times 2^{-2} = (1 + 0.0625) \times 2^{125-127}$$



# Floating-Point Number

- Exponent Width → Range; Fraction Width → Precision

**IEEE 754 Single Precision 32-bit Float (IEEE FP32)**



**IEEE Half Precision 16-bit Float (IEEE FP16)**



**Brain Float (BF16)**



**Nvidia TensorFloat (TF32)**



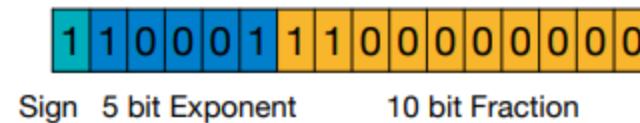
**AMD 24-bit Float (AMD FP24)**



	Exponent (bits)	Fraction (bits)	Total (bits)
IEEE 754 Single Precision 32-bit Float (IEEE FP32)	8	23	32
IEEE Half Precision 16-bit Float (IEEE FP16)	5	10	16
Brain Float (BF16)	8	7	16
Nvidia TensorFloat (TF32)	8	10	19
AMD 24-bit Float (AMD FP24)	7	16	24

# Numeric Data Types

- Question: What is the following IEEE half-precision (IEEE FP16) number in decimal?



Exponent Bias =  $15_{10}$

- Sign: -
- Exponent:  $10001_2 - 15_{10} = 17_{10} - 15_{10} = 2_{10}$
- Fraction:  $1100000000_2 = 0.75_{10}$
- Decimal Answer =  $-(1 + 0.75) \times 2^2 = -1.75 \times 2^2 = -7.0_{10}$

Copyright 2022. (차세대반도체 혁신공유대학 사업단)

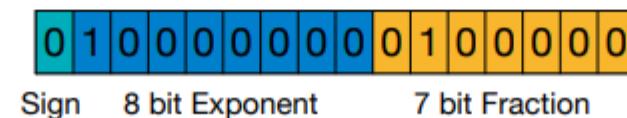
# Numeric Data Types

- Question: What is the decimal 2.5 in Brain Float (BF16)?

$$2.5_{10} = 1.\underline{25}_{10} \times 2^1$$

Exponent Bias =  $127_{10}$

- Sign: +
- Exponent Binary:  $1_{10} + 127_{10} = 128_{10} = 10000000_2$
- Fraction Binary:  $0.25_{10} = 0100000_2$
- Binary Answer

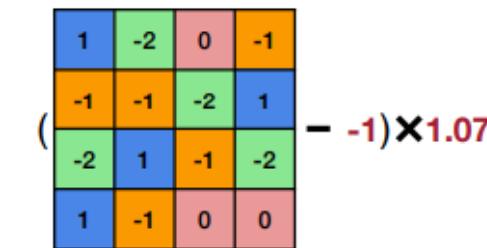


# Neural Network Quantization: Agenda

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

3: 2.00  
2: 1.50  
1: 0.00  
0: -1.00



1	0	1	1
1	0	0	1
0	1	1	0
1	1	1	1

## K-Means-based Quantization

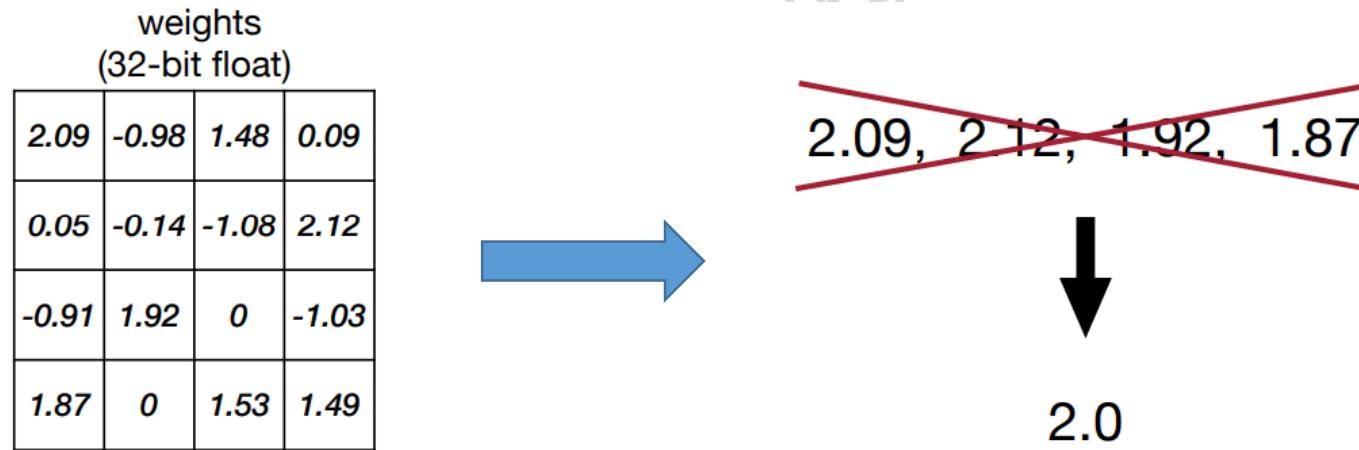
## Linear Quantization

## Binary/Ternary Quantization

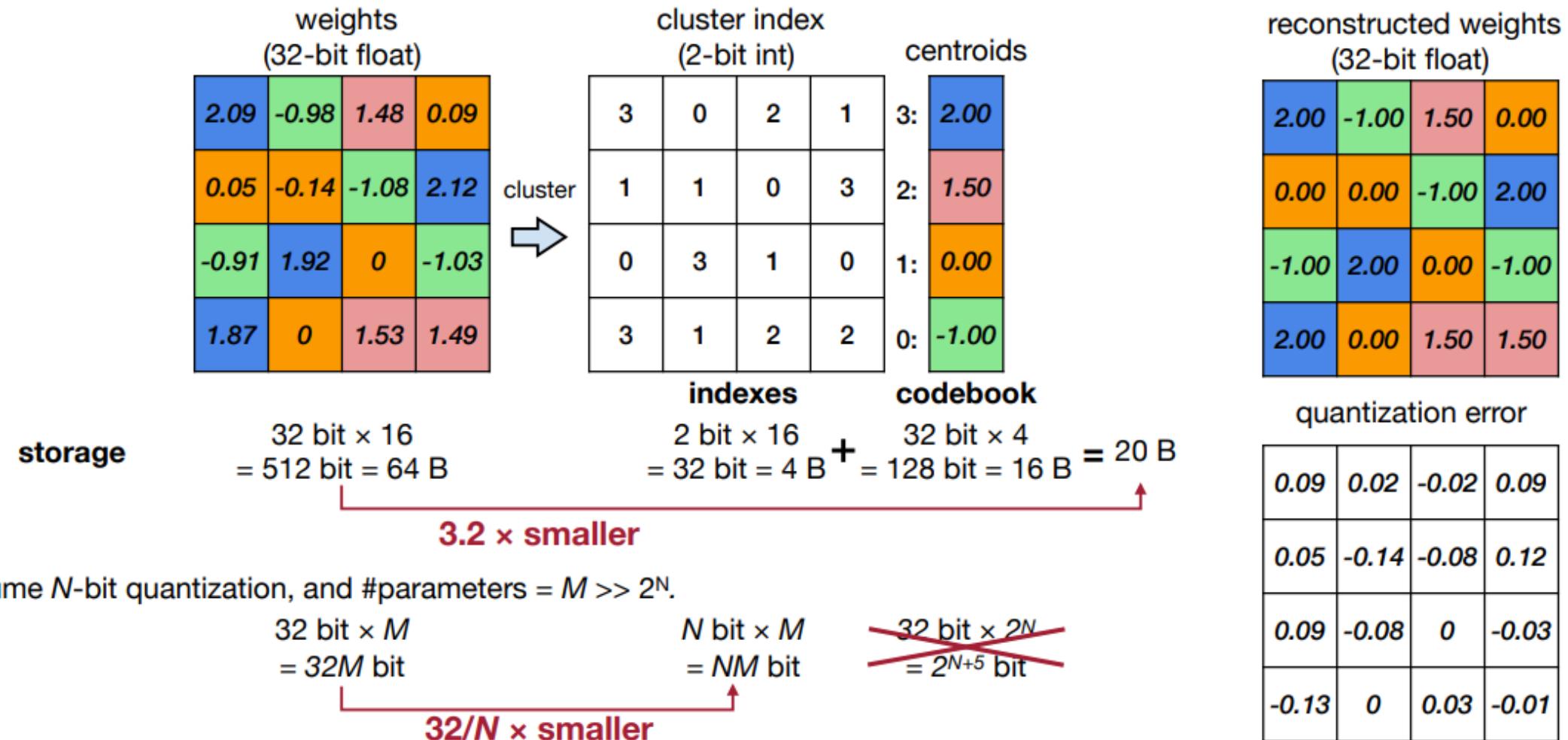
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights	Binary/Ternary Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic	Bit Operations

# Neural Network Quantization

- Model/Weight quantization: Reduce the storage
  - Data Movement: Less Memory Reference → Less Energy
  - Computation: No benefit for low-bit arithmetic
    - Incur some overheads for decoding compressed data.

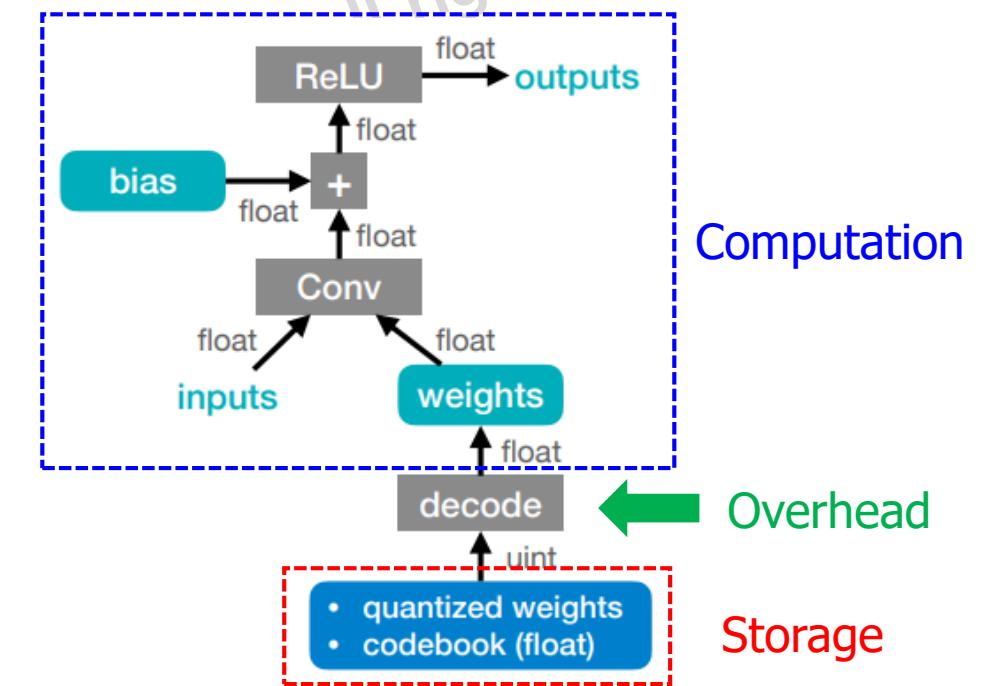
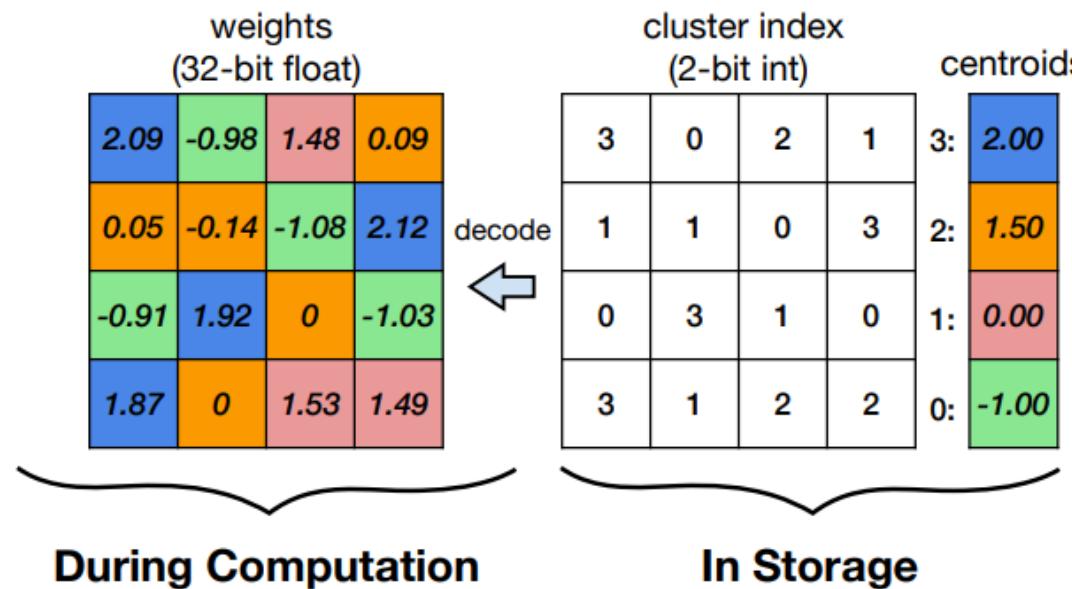


# K-Means-based Weight Quantization



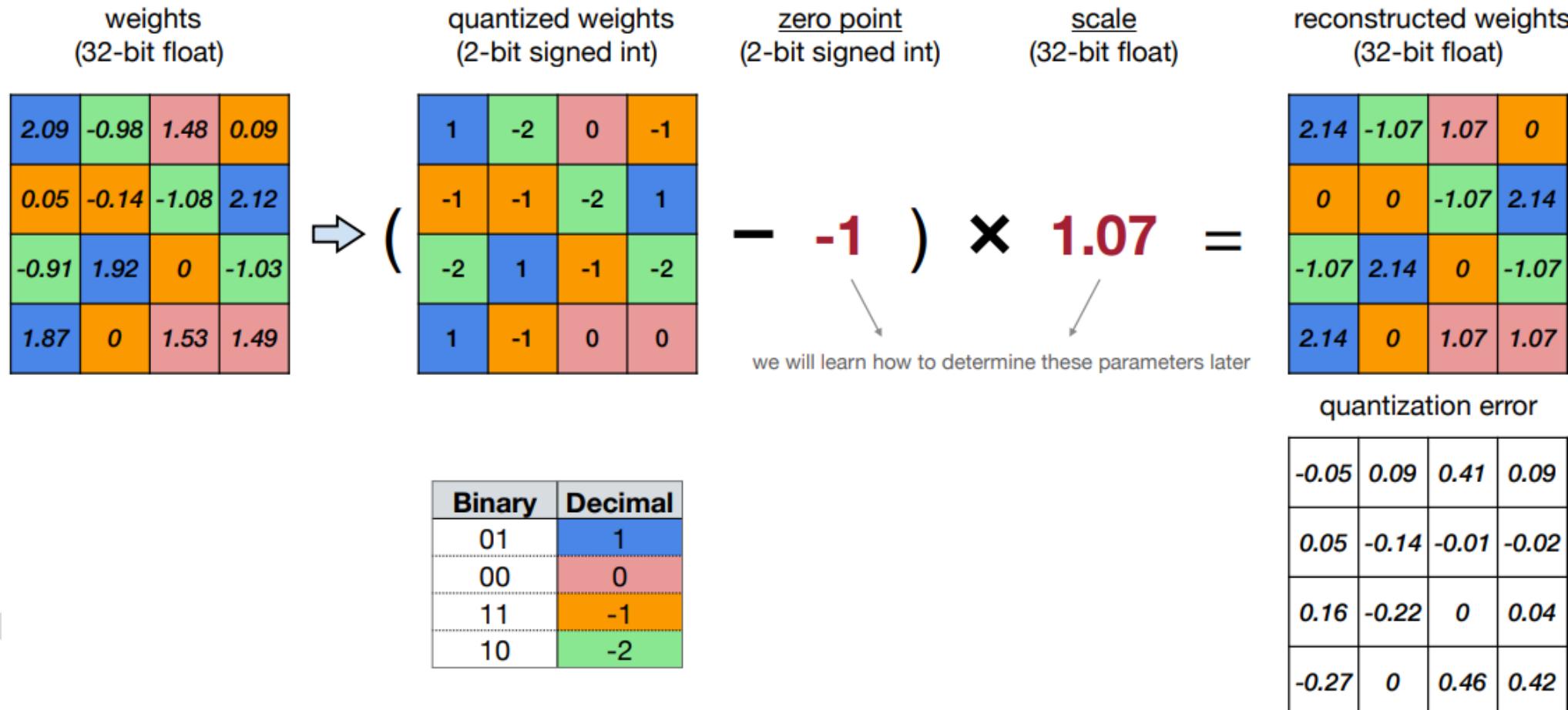
# K-Means-based Weight Quantization

- The weights are decompressed using a lookup table (i.e., codebook) during runtime inference.
  - K-Means-based Weight Quantization only saves the storage cost of a neural network model.
  - All the computation and memory access are still floating-point.



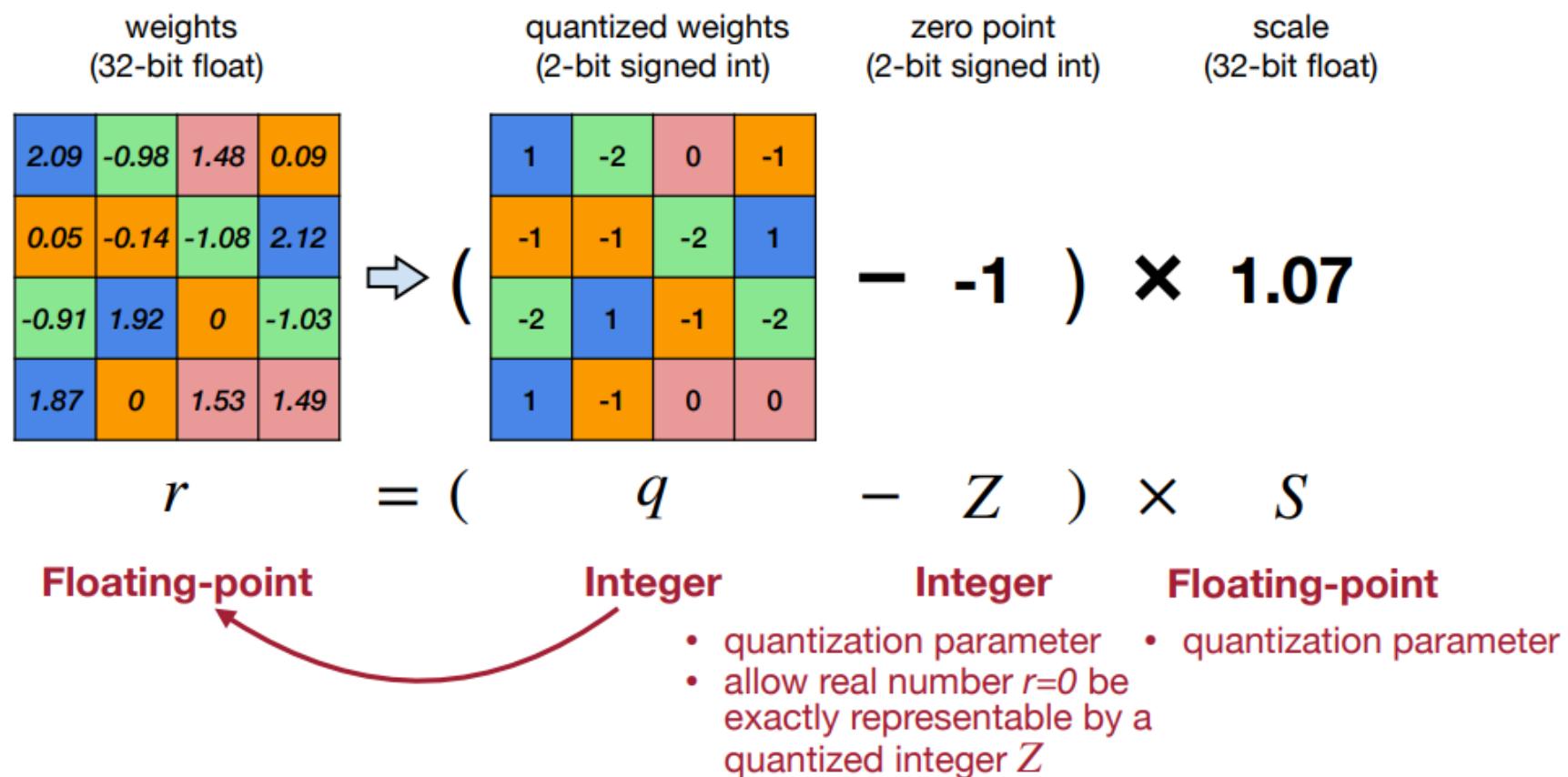
# What is Linear Quantization?

- An affine mapping of integers to real numbers



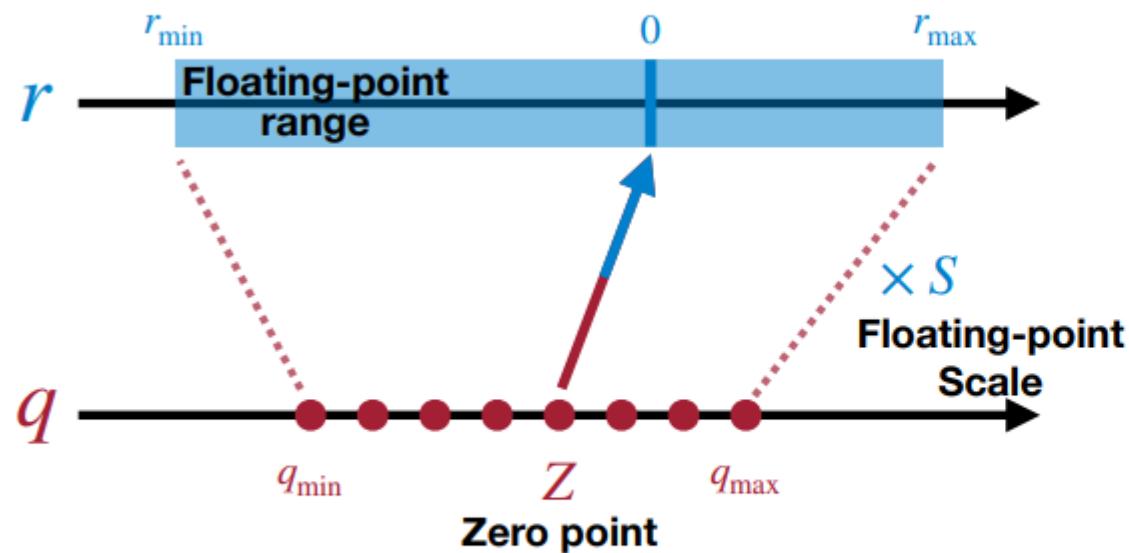
# What is Linear Quantization?

- An affine mapping of integers to real numbers  $r = S(q - Z)$



# Scale $S$ of Linear Quantization

- An affine mapping of integers to real numbers  $r = S(q - Z)$

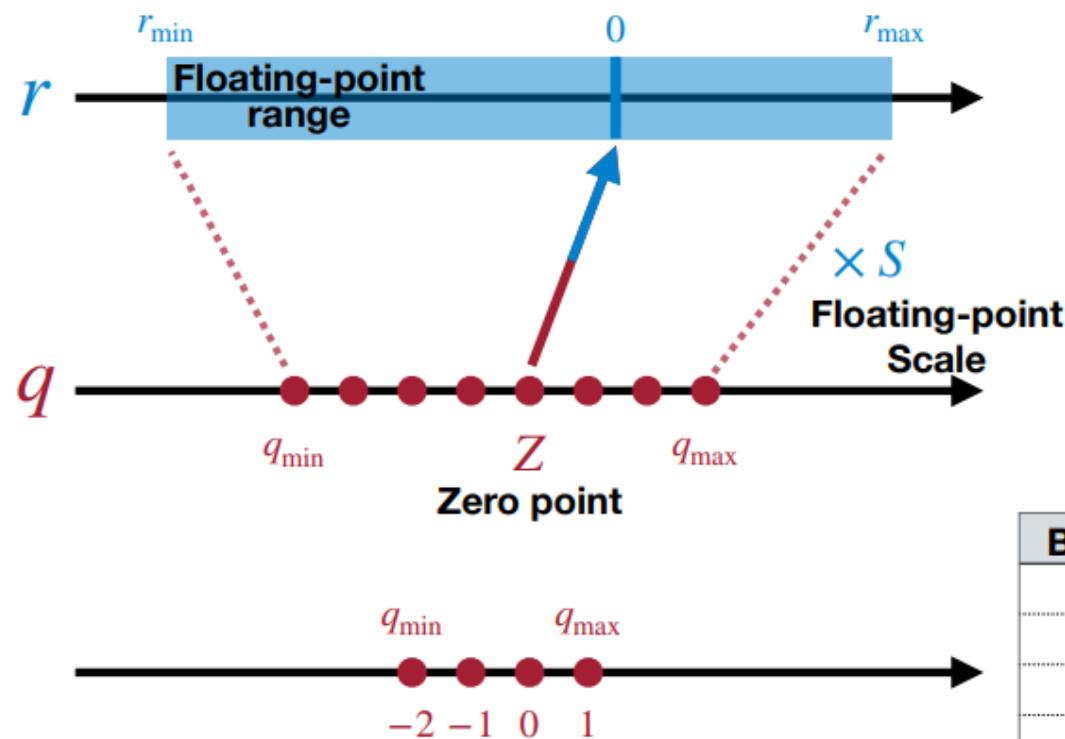


$$\begin{aligned} r_{\max} &= S(q_{\max} - Z) \\ r_{\min} &= S(q_{\min} - Z) \\ r_{\max} - r_{\min} &= S(q_{\max} - q_{\min}) \end{aligned}$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

# Scale $S$ of Linear Quantization

- An affine mapping of integers to real numbers  $r = S(q - Z)$



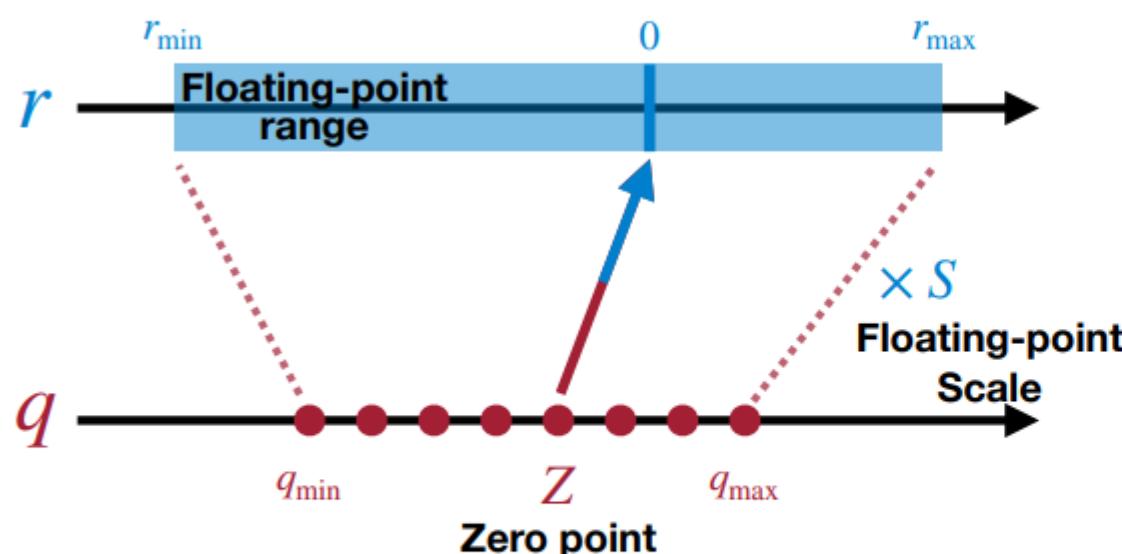
2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Binary	Decimal
01	1
00	0
11	-1
10	-2

$$\begin{aligned} S &= \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}} \\ &= \frac{2.12 - (-1.08)}{1 - (-2)} \\ &= 1.07 \end{aligned}$$

# Zero Point $Z$ of Linear Quantization

- An affine mapping of integers to real numbers  $r = S(q - Z)$



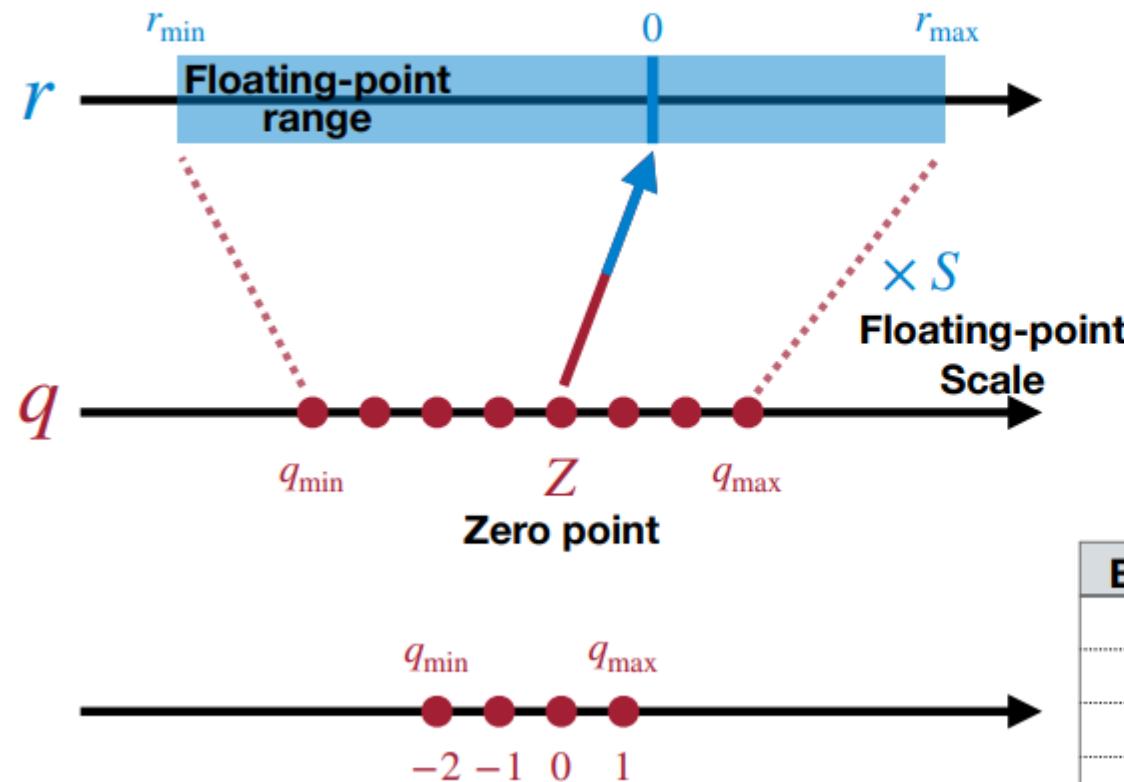
$$r_{\min} = S(q_{\min} - Z)$$

$$Z = q_{\min} - \frac{r_{\min}}{S}$$

$$Z = \text{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

# Zero Point $Z$ of Linear Quantization

- An affine mapping of integers to real numbers  $r = S(q - Z)$



2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

$$Z = q_{\min} - \frac{r_{\min}}{S}$$

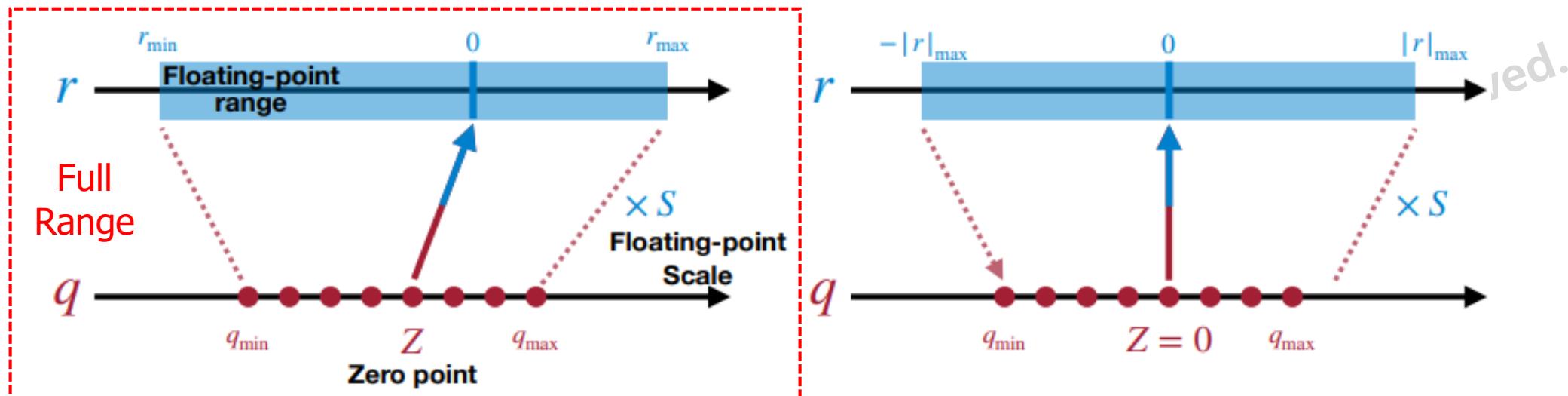
$$= \text{round}\left(-2 - \frac{-1.08}{1.07}\right)$$

$$= -1$$

Binary	Decimal
01	1
00	0
11	-1
10	-2

# Symmetric Linear Quantization

- Zero point  $Z = 0$  and Symmetric floating-point range



$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

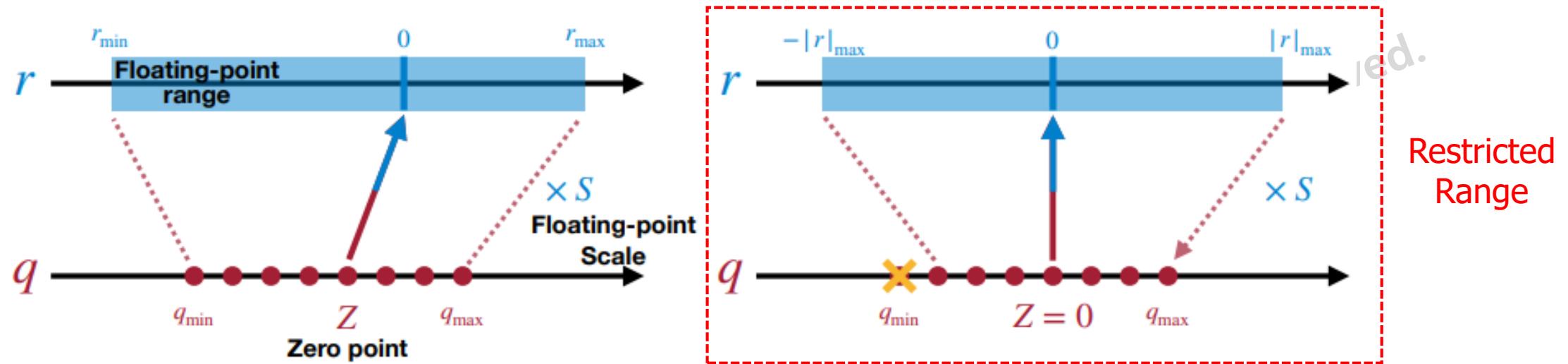
Bit Width	$q_{\min}$	$q_{\max}$
2	-2	1
3	-4	3
4	-8	7
N	$-2^{N-1}$	$2^{N-1}-1$

$$S = \frac{r_{\min} - Z}{q_{\min} - Z} = \frac{-|r|_{\max}}{q_{\min}} = \frac{|r|_{\max}}{2^{N-1}}$$

- use full range of quantized integers
- example: PyTorch's native quantization, ONNX

# Symmetric Linear Quantization

- Zero point  $Z = 0$  and Symmetric floating-point range



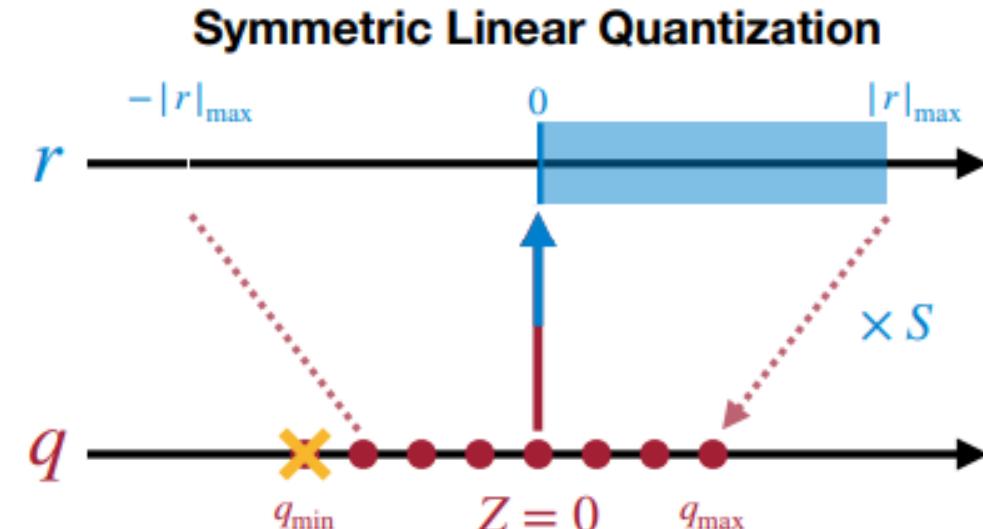
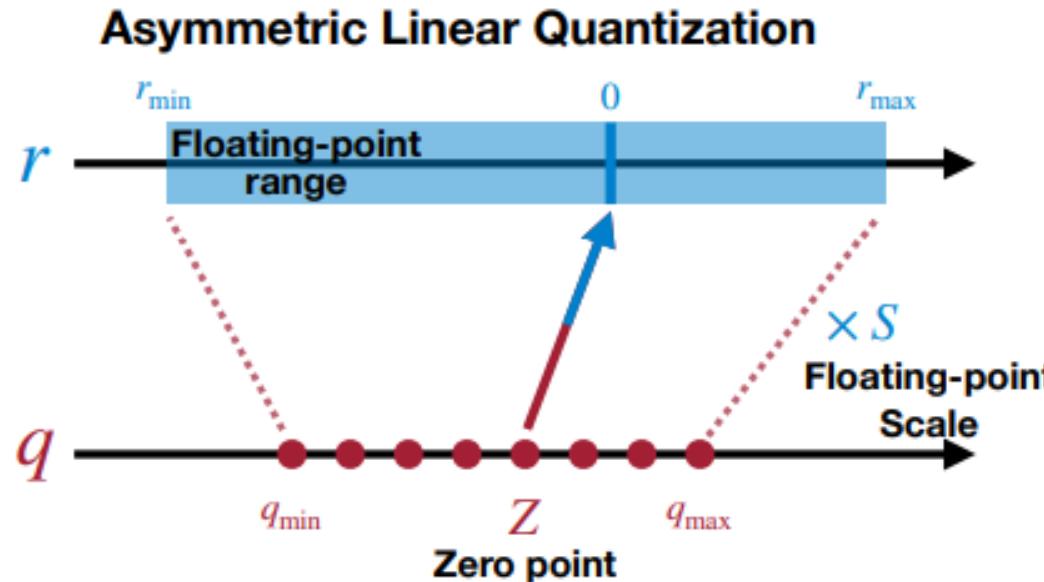
$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

$$S = \frac{r_{\max}}{q_{\max} - Z} = \frac{|r|_{\max}}{q_{\max}} = \frac{|r|_{\max}}{2^{N-1} - 1}$$

- example: TensorFlow, NVIDIA TensorRT, Intel DNNL

Bit Width	$q_{\min}$	$q_{\max}$
2	-2	1
3	-4	3
4	-8	7
N	$-2^{N-1}$	$2^{N-1}-1$

# Asymmetric vs Symmetric

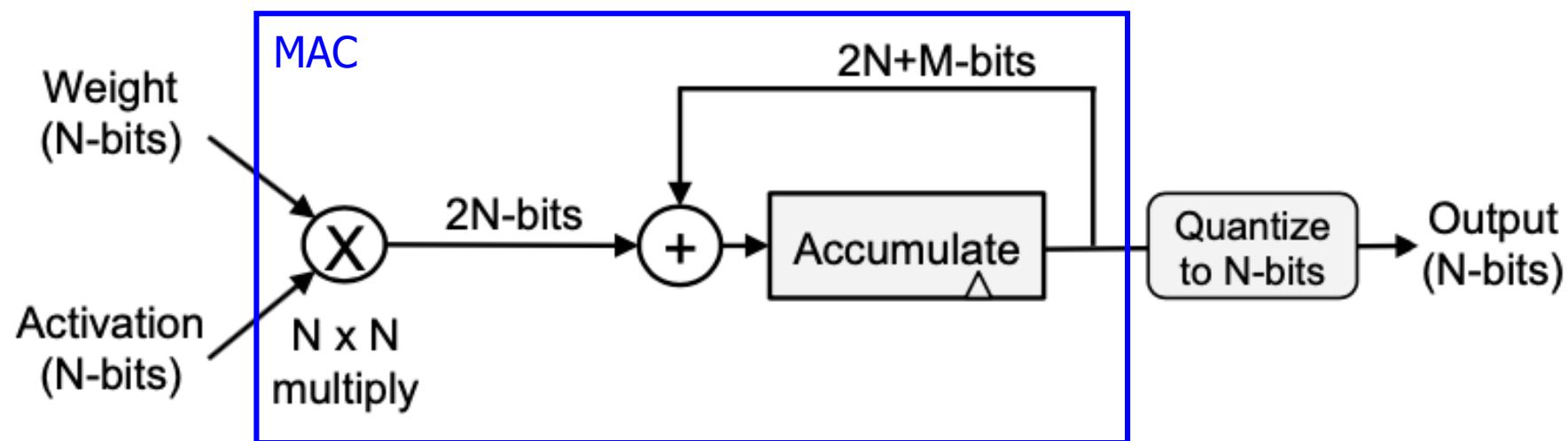


- The quantized range is fully used.
- The implementation is more complex, and zero points require additional logic in hardware.

- The quantized range will be wasted for biased float range.
  - Activation tensor is non-negative after ReLU, and thus symmetric quantization will lose 1 bit effectively.
- The implementation is much simpler.

# Multiplication and Accumulation (MAC)

- Accumulation requires higher precision than inputs.
- Inputs: two N-bit weight activation
- The accumulated result is quantized to a N-bit number (output activation)



# Lab 2: Quantization

- Load a model
- Complete the missing codes
  - Weight/Bias quantization: uniform\_quantize.m
  - Activation Quantization:
    - ReLU: hwu\_relu\_quantize.m
    - Linear: hwu\_linear\_quantize.m

# Load a CNN model

- Parameters: layer 1 (w1, b1), layer 2 (w2, b2), layer 3 (w3, b3)
- Input files locate at bin/
- Weights are reorganized in a typical format:  
[Filter size, Filter size, input channels, output channels]

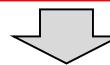
```
%% Load pre-trained convolutional parameter
% Load Layer 1
[w1,b1,~,~,~] = read_conv_param_module('bin/sim_espcn_3x3.weights_layer_0',3, 1,16,0);
w1 = permute(reshape(w1, 3, 3, 1, 16), [2, 1, 3, 4]);
% Load Layer 2
[w2,b2,~,~,~] = read_conv_param_module('bin/sim_espcn_3x3.weights_layer_1',3,16,16,0);
w2 = permute(reshape(w2, 3, 3, 16, 16), [2, 1, 3, 4]);
% Load Layer 3
[w3,b3,~,~,~] = read_conv_param_module('bin/sim_espcn_3x3.weights_layer_2',3, 16,4,0);
w3 = permute(reshape(w3, 3, 3, 16,4), [2, 1, 3, 4]);
```

# Weight quantization

- Convert a FP tensor into a fixed-point format
- Example: 8-bit symmetric representation
  - $-255/2, -254/2, \dots, -1/2, 1/2, \dots, 255/2$

Floating point weight

```
w1 (:,:,1,1) =  
  
-0.3302   -0.3016   -0.1170  
 0.3708   -0.2676   -0.1463  
-0.0760   -0.0007    0.3297
```



```
wq1 (:,:,1,1) =  
  
-169   -155    -59  
189   -137    -75  
-39     -1     169
```

Quantized weight

i bits = 0, f bits = 8

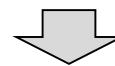
```
% Function: Symmetric Uniform quantization  
% Inputs  
% - x:      input tensor  
% - step: quantized step  
% - nbit: number of bits in fixed-point representation  
% Outputs  
% - output: Quantized value mapping to x  
% - output_store: Store value  
  
function [output, output_store] = uniform_quantize(x,step,nbit)  
% Initialization  
output = x;  
output_store = x;  
  
% Maximum and minimum ranges  
pos_end = 2 ^ nbit - 1;  
neg_end = -pos_end;  
  
% Quantized value  
output = 2 * round(x./step + 0.5) - 1;  
output (output > pos_end) = pos_end;  
output (output < neg_end) = neg_end;  
  
output_store = (output - 1) / 2;
```

# Weight quantization

- Convert a FP tensor into a fixed-point format
- Example: 8-bit symmetric representation
  - $-255/2, -254/2, \dots, -1/2, 1/2, \dots, 255/2$

Quantized weight

```
wq1(:,:,:,1,1) =  
  
-169   -155    -59  
189    -137    -75  
-39      -1     169
```



```
wq1_store(:,:,:,1,1) =  
  
-85    -78    -30  
94     -69    -38  
-20      -1     84
```

Store weights

```
% Function: Symetric Uniform quantization  
% Inputs  
% - x:      input tensor  
% - step: quantized step  
% - nbit: number of bits in fixed-point representation  
% Outputs  
% - output: Quantized value mapping to x  
% - output_store: Store value  
  
function [output, output_store] = uniform_quantize(x,step,nbit)  
% Initialization  
output = x;  
output_store = x;  
  
% Maximum and minimum ranges  
pos_end = 2 ^ nbit - 1;  
neg_end = -pos_end;  
  
% Quantized value  
output = 2 * round(x./step + 0.5) - 1;  
output(output > pos_end) = pos_end;  
output(output < neg_end) = neg_end;  
  
output_store = (output - 1) / 2;
```

# Activation Quantization (ReLU)

- Convert a FP tensor into a fixed-point format
  - Similar to weight/bias quantization
  - Normalize ranges of biases and convolution outputs

```
% Function: Activation quantization
% Inputs
% - x: input tensor
% - step: quantized step
% - nbit: number of bits in fixed-point representation
% - biases_shift
% Outputs
% - activations: Quantized value mapping to x
% - activations_store: Store value

function [activations, activations_store] = hwu_relu_quantize(x, step, nbit, biases_shift)
% Initialization
activations = x;
activations_store = x;

% Maximum quantized range
pos_end = 2 ^ nbit - 1;

% Quantization
activations_store = x;
activations_store(x >= 0) = round(x(x >= 0)/(2^biases_shift*(step)));
activations_store(x < 0) = 0;

% ReLU
activations_store(activations_store > pos_end) = pos_end;
activations = activations_store;
```

# Activation Quantization (Linear)

- Convert a FP tensor into a fixed-point format
  - Similar to weight/bias quantization
  - Normalize ranges of biases and convolution outputs

```
% Function: Activation quantization
% Inputs
% - x:      input tensor
% - step: quantized step
% - nbit: number of bits in fixed-point representation
% - biases_shift
% Outputs
% - activations: Quantized value mapping to x
% - activations_store: Store value
function [activations, activations_store] = hwu_linear_quantize(x, step, nbit, biases_shift)
% Initialization
activations = x;
activations_store = x;

% Maximum and minimum ranges
pos_end = 2 ^ (nbit-1) - 1;
neg_end = -pos_end - 1;

% Quantization
activations_store = round(x/(2^biases_shift*(step)));

% Linear activation
activations_store(activations_store > pos_end) = pos_end;
activations_store(activations_store < neg_end) = neg_end;

% Outputs for buffering/storing into memory
activations = activations_store;
```

# To do ...

- Complete the missing codes
  - Weight/Bias quantization: uniform\_quantize.m
  - Activation Quantization:
    - ReLU: hwu\_relu\_quantize.m
    - Linear: hwu\_linear\_quantize.m
- Complete the missing code in test\_simSR\_quant.m:
  - Run for all three layers
  - Last layer uses linear activation, two first layers use ReLU
  - Show the feature maps
- Refer to figures in **featuremaps\_quant** folder for sample feature maps
  - (LayerNumber)\_(ChannelNumber).jpg

# Complete the missing code

```
% Function: Symmetric Uniform quantization  
% Inputs  
% - x: input tensor  
% - step: quantized step  
% - nbit: number of bits in fixed-point representation  
% Outputs  
% - output: Quantized value mapping to x  
% - output_store: Store value
```

```
function [output, output_store] = uniform_quantize(x, step, nbit)  
% Initialization  
output = x;  
output_store = x;
```

uniform\_quantize.m

% Insert your code

```
% Function: Activation quantization  
% Inputs  
% - x: input tensor  
% - step: quantized step  
% - nbit: number of bits in fixed-point representation  
% - biases_shift  
% Outputs  
% - activations: Quantized value mapping to x  
% - activations_store: Store value
```

```
function [activations, activations_store] = hwu_relu_quantize(x, step, nbit, biases_shift)  
% Initialization  
activations = x;  
activations_store = x;
```

hwu\_relu\_quantize.m

% Insert your code

# Complete the missing code

```
% Function: Activation quantization
% Inputs
% - x: input tensor
% - step: quantized step
% - nbit: number of bits in fixed-point representation
% - biases_shift
% Outputs
% - activations: Quantized value mapping to x
% - activations_store: Store value
```

```
function [activations, activations_store] = hwu_linear_quantize(x, step, nbit, biases_shift)
    % Initialization
    activations = x;
    activations_store = x;
```

% Insert your code

hwu\_linear\_quantize.m

```
% % First Layer
conv_out = convol2(input, wq1, 1, 2);
for j = 1:size(conv_out, 3)
    % Add bias
    conv_out(:, :, j) = conv_out(:, :, j) + bq1(j);
end

% Activation
conv_out_relu = hwu_relu_quantize(conv_out, step, nbits, biases_shift);

% for i = 1:size(conv_out, 3)
%     figure(1)
%     subplot(1,2,1)
%     imshow(conv_out(:, :, i)); title(['ch = ', num2str(i)]);
%     subplot(1,2,2)
%     imshow(conv_out_relu(:, :, i)); title('After ReLU');
%     pause(1)
% end
```

test\_simSR\_quant.m

% Insert your code for other layers

# Road map

Deep Learning Accelerator

Convolutional Neural  
Network (CNN)

Quantization

Block memory

Reference S/W

Copyright 2022. (차세대반도체 혁신공유대학 사업단) all rights reserved.

# Lab 3: Block memory

- Lab 3:
  - Dual-port block ram
  - Initialize memory cell with quantized weights
  - Implement a test bench by complete missing code

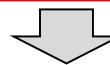
Copyright 2022. (차세대반도체 혁신공유대학 사업단) all rights reserved.

# Weight quantization

- Convert a FP tensor into a fixed-point format
- Example: 8-bit symmetric representation
  - $-255/2, -254/2, \dots, -1/2, 1/2, \dots, 255/2$

Floating point weight

```
w1 (:,:,1,1) =  
  
-0.3302   -0.3016   -0.1170  
 0.3708   -0.2676   -0.1463  
-0.0760   -0.0007    0.3297
```



```
wq1 (:,:,1,1) =  
  
-169   -155    -59  
189   -137    -75  
-39     -1     169
```

Quantized weight

i bits = 0, f bits = 8

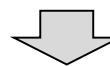
```
% Function: Symetric Uniform quantization  
% Inputs  
% - x:      input tensor  
% - step: quantized step  
% - nbit: number of bits in fixed-point representation  
% Outputs  
% - output: Quantized value mapping to x  
% - output_store: Store value  
  
function [output, output_store] = uniform_quantize(x,step,nbit)  
% Initialization  
output = x;  
output_store = x;  
  
% Maximum and minimum ranges  
pos_end = 2 ^ nbit - 1;  
neg_end = -pos_end;  
  
% Quantized value  
output = 2 * round(x./step + 0.5) - 1;  
output(output > pos_end) = pos_end;  
output(output < neg_end) = neg_end;  
  
output_store = (output - 1) / 2;
```

# Weight quantization

- Convert a FP tensor into a fixed-point format
- Example: 8-bit symmetric representation
  - $-255/2, -254/2, \dots, -1/2, 1/2, \dots, 255/2$

Quantized weight

```
wq1 (:,:,1,1) =  
  
-169   -155    -59  
189    -137    -75  
-39      -1     169
```



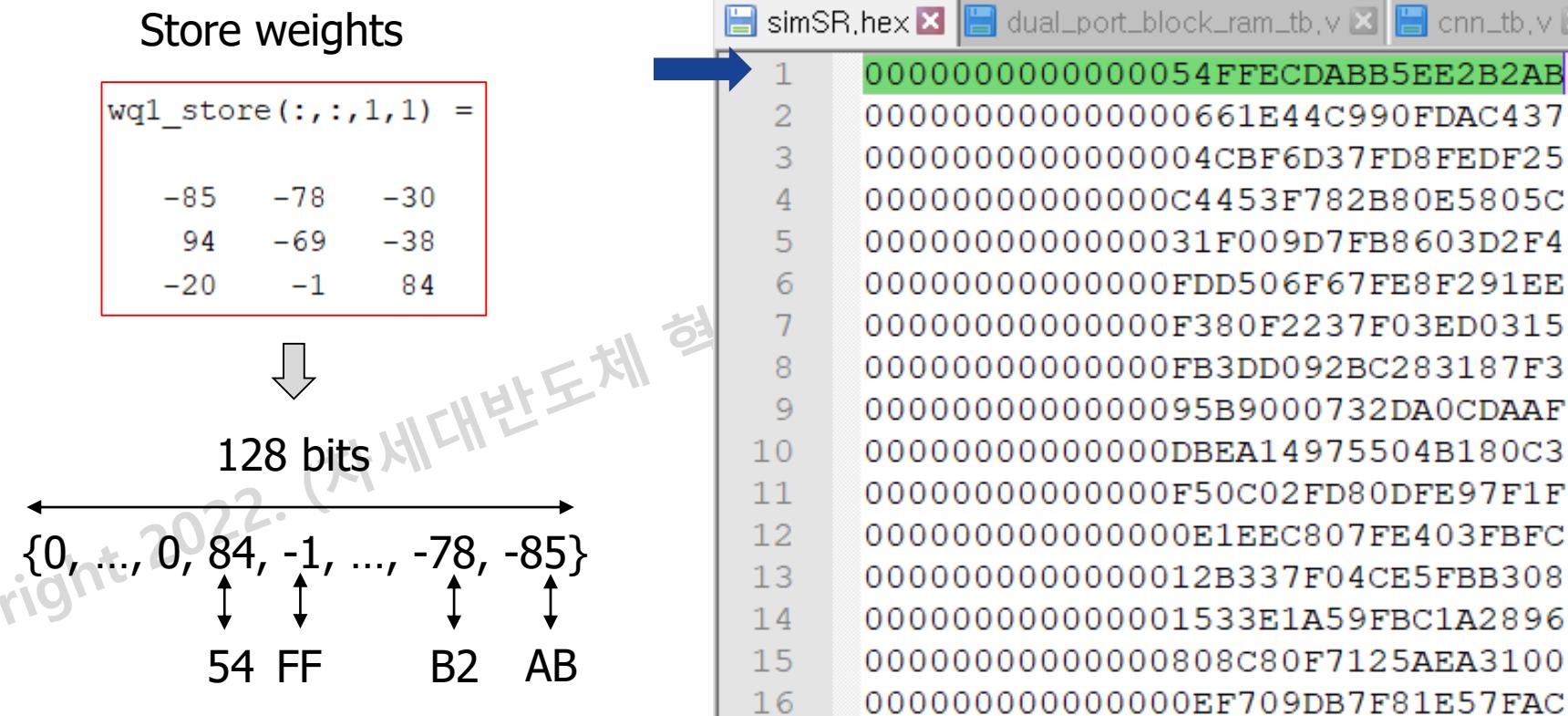
```
wq1_store (:,:,1,1) =  
  
-85    -78    -30  
94     -69    -38  
-20      -1     84
```

Store weights

```
% Function: Symetric Uniform quantization  
% Inputs  
%   - x:      input tensor  
%   - step: quantized step  
%   - nbit: number of bits in fixed-point representation  
% Outputs  
%   - output: Quantized value mapping to x  
%   - output_store: Store value  
  
function [output, output_store] = uniform_quantize(x,step,nbit)  
% Initialization  
output = x;  
output_store = x;  
  
% Maximum and minimum ranges  
pos_end = 2 ^ nbit - 1;  
neg_end = -pos_end;  
  
% Quantized value  
output = 2 * round(x./step + 0.5) - 1;  
output(output > pos_end) = pos_end;  
output(output < neg_end) = neg_end;  
  
output_store = (output - 1) / 2;
```

# Memory file generation: Channel-wise

- Convert a FP tensor into a fixed-point format  
⇒ Save fixed-point weights into a file



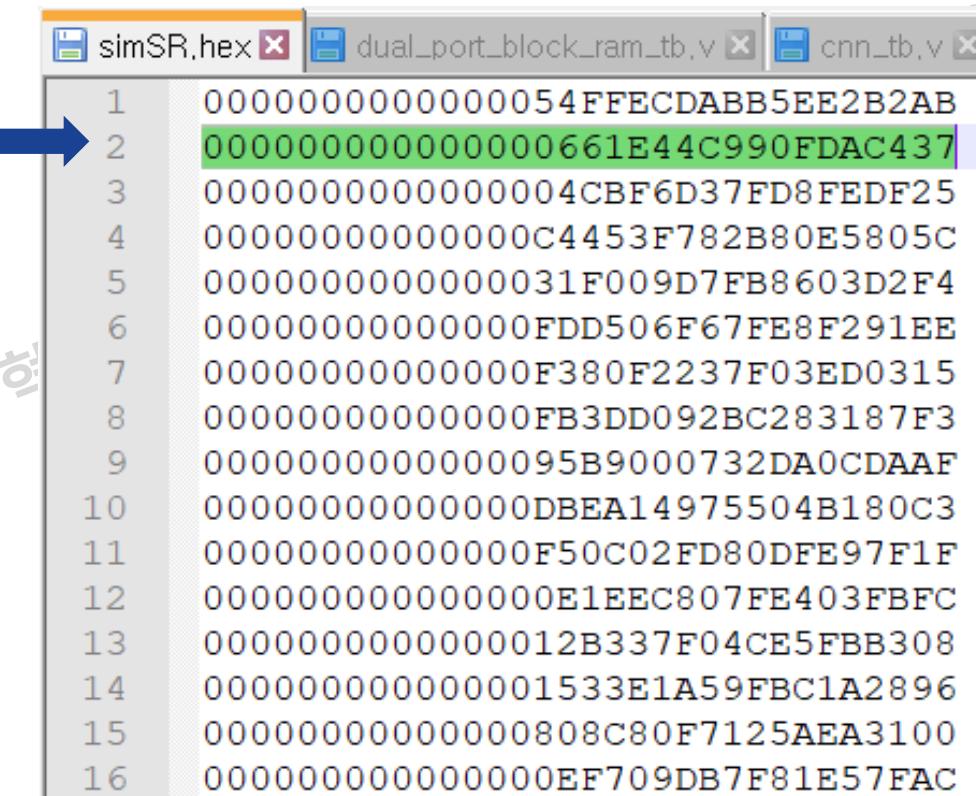
# Memory file generation: Channel-wise

- Convert a FP tensor into a fixed-point format  
⇒ Save fixed-point weights into a file

```
wq1_store(:,:,:,1,1) =  
  
-85    -78    -30  
 94    -69    -38  
-20     -1     84
```

```
wq1_store(:,:,:,1,2) =  
  
 55    -60    -38  
 15   -103     76  
-28     97      6
```

```
wq1_store(:,:,:,1,3) =  
  
 37    -33     -2  
-40     127    -45  
-10    -53      4
```



1	000000000000000054FFEC
2	0000000000000000661E44C990FDAC437
3	00000000000000004CBF6D37FD8FEDF25
4	0000000000000000C4453F782B80E5805C
5	000000000000000031F009D7FB8603D2F4
6	0000000000000000FDD506F67FE8F291EE
7	0000000000000000F380F2237F03ED0315
8	0000000000000000FB3DD092BC283187F3
9	000000000000000095B9000732DA0CDAAF
10	0000000000000000DBEA14975504B180C3
11	0000000000000000F50C02FD80DFE97F1F
12	0000000000000000E1EEC807FE403FBFC
13	000000000000000012B337F04CE5FBB308
14	00000000000000001533E1A59FBC1A2896
15	0000000000000000808C80F7125AEA3100
16	0000000000000000EF709DB7F81E57FAC

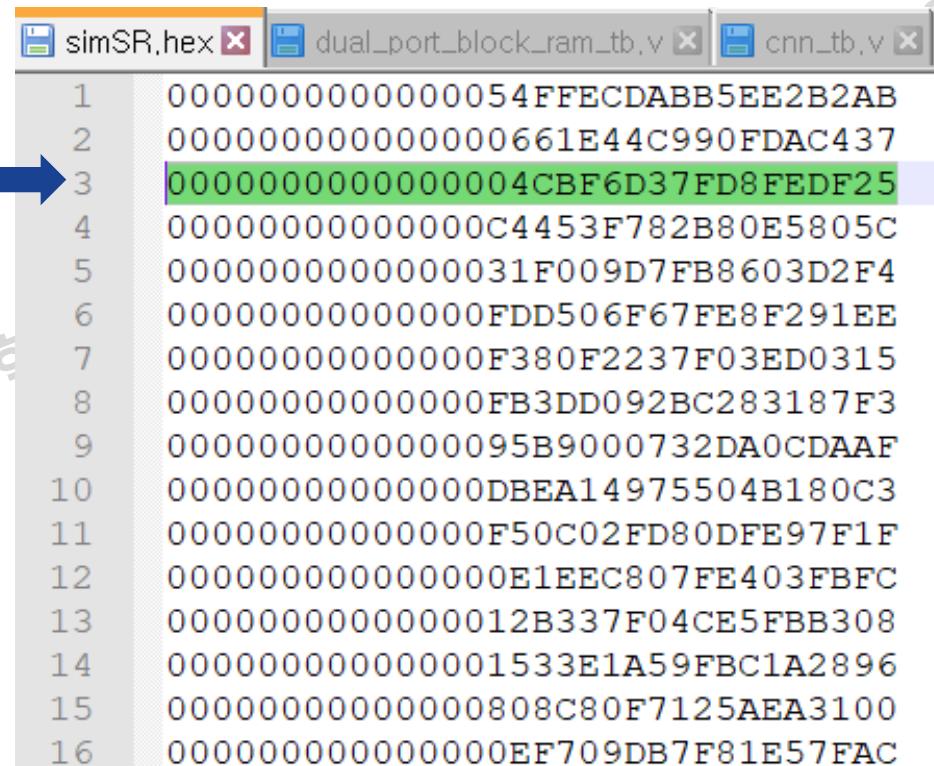
# Memory file generation: Channel-wise

- Convert a FP tensor into a fixed-point format  
⇒ Save fixed-point weights into a file

```
wq1_store(:,:,1,1) =  
  
-85    -78    -30  
 94    -69    -38  
-20     -1     84
```

```
wq1_store(:,:,1,2) =  
  
 55    -60    -38  
 15   -103     76  
-28     97      6
```

```
wq1_store(:,:,1,3) =  
  
 37    -33     -2  
-40     127    -45  
-10    -53      4
```



1	000000000000000054FFEC
2	0000000000000000661E44C990FDAC437
3	00000000000000004CBF6D37FD8FEDF25
4	0000000000000000C4453F782B80E5805C
5	000000000000000031F009D7FB8603D2F4
6	0000000000000000FDD506F67FE8F291EE
7	0000000000000000F380F2237F03ED0315
8	0000000000000000FB3DD092BC283187F3
9	000000000000000095B9000732DA0CDAAF
10	0000000000000000DBEA14975504B180C3
11	0000000000000000F50C02FD80DFE97F1F
12	0000000000000000E1EEC807FE403FBFC
13	000000000000000012B337F04CE5FBB308
14	00000000000000001533E1A59FBC1A2896
15	0000000000000000808C80F7125AEA3100
16	0000000000000000EF709DB7F81E57FAC

# Memory mapping

- Convert a FP tensor into a fixed-point format  
⇒ Save fixed-point weights into a file

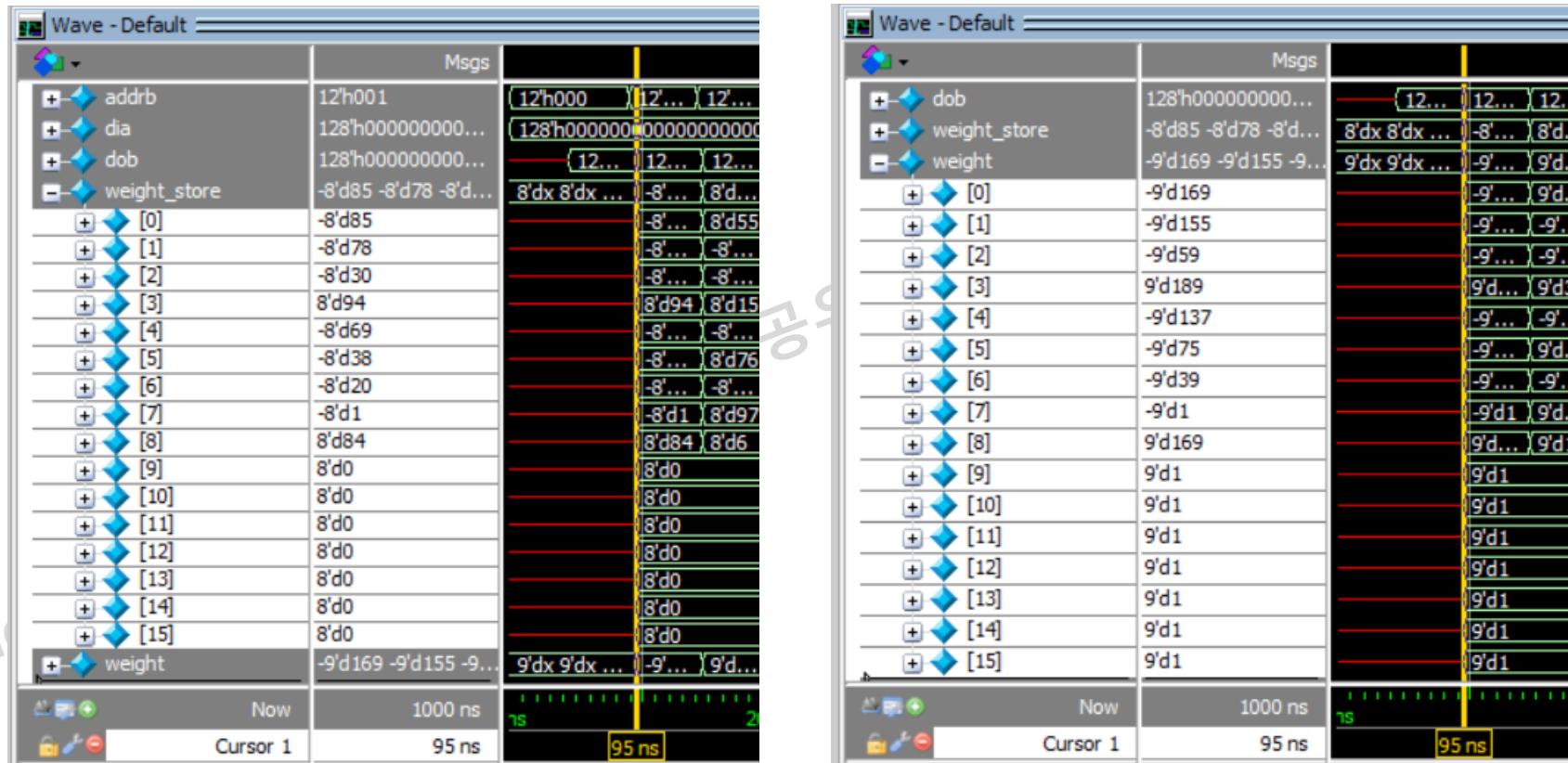
Copyright 신공유

Weights				
wq1_store(:,:,1,1) =				
-85	-78	-30		
94	-69	-38		
-20	-1	84		
wq1_store(:,:,1,2) =				
55	-60	-38		
15	-103	76		
-28	97	6		
Input				
input				
128x128 double				
1	2	3	4	
1	42	69	91	99
2	105	42	56	84

133	000000000000000031F009D7FB8603D2F4
134	0000000000000000FDD506F67FE8F291EE
135	0000000000000000F380F2237F03ED0315
136	0000000000000000FB3DD092BC283187F3
137	000000000000000095B9000732DA0CDAAF
138	0000000000000000DBEA14975504B180C3
139	0000000000000000F50C02FD80DFE97F1F
140	0000000000000000E1EEC807FE403FBFC
141	000000000000000012B337F04CE5FBB308
142	00000000000000001533E1A59FBC1A2896
143	0000000000000000808C80F7125AEA3100
144	0000000000000000EF709DB7F81E57FAC
145	006d006f006c006a0063005b0045002a
146	0058005c0061006200640067006a006d
147	004400440048004b004c005000520054
148	004e004e004d004b004b004a00470045
149	004a004800490048004a004c004e004d
150	006b0063005a00540053004f004b004b

# To do ...

- Complete the missing code in dual\_port\_block\_ram\_tb.v to load 16 conv. filters of Layer 1
- Show the stored weight and the original weights



# Dual port BRAM

- Frame buffer (lcd\_frame\_buffer.v):
  - Port A: Write
    - Read/write request (ena)
    - Write enable (wea)
    - Address (addra)
    - Write data (dina)
  - Port B: Read
    - Read/write request (enb)
    - Address (addrb)
    - Read data (dob)

```
module dual_port_block_ram(  
    clk ,  
    ena , // portA: enable  
    wea , // portA: primary synchronous write enable  
    addra , // portA: address for read/write  
  
    enb , // portB: enable  
    addrb , // portB: address for read  
  
    dia , // portA: primary data input  
    dob // portB: primary data output  
);
```

Copyright 2022. (차세대반도체)

# Explanation – dual\_port\_block\_ram.v

```
parameter W_DATA = 8;                                     Default parameters
parameter N_CELL = 512;
parameter W_CELL = $clog2(N_CELL);
parameter FILENAME = "";
parameter N_DELAY = 1;
```

```
// share memory                                         RAM (data storage)
reg [W_DATA-1:0] ram[N_CELL-1:0];
```

```
initial                                              Read data from FILENAME(simSR.hex) to ram
begin
    if (FILENAME != "") begin
        $display("### Loading internal memory from %s ###", FILENAME);
        $readmemh(FILENAME, ram);
    end
end
```

```
// write port                                           Data write to RAM
always @(posedge clk)
begin: write
    if(ena)
        begin
            if(wea)
                ram[addr] <= dia;
        end
    end
end
```

# Explanation – dual\_port\_block\_ram.v

```
generate
    if(N_DELAY == 1) begin: delay_1
        reg [W_DATA-1:0] rdata; // primary data output
        // read port
        always @(posedge clk)
        begin: read
            if(enb)
                rdata <= ram[addrb];
            end
            assign dob = rdata;
        end
    end

    else begin: delay_n
        reg [N_DELAY*W_DATA-1:0] rdata_r;
        always @(posedge clk)
        begin: read
            if(enb)
                rdata_r[0+:W_DATA] <= ram[addrb];
            end
        end

        always @(posedge clk) begin: delay
            integer i;
            for(i = 0; i < N_DELAY-1; i = i+1)
                if(enb)
                    rdata_r[(i+1)*W_DATA+:W_DATA] <= rdata_r[i*W_DATA+:W_DATA];
            end
            assign dob = rdata_r[(N_DELAY-1)*W_DATA+:W_DATA];
        end
    end
endgenerate
```

Data read from RAM : 1 cycle delay

Data read from RAM : more than 1 cycle delay  
(not used for this example! Just ignore)

Support N cycle data delay : 1 cycle delay for this example

Assign RAM output port

# Test bench (dual\_port\_block\_ram\_tb.v)

```
module dual_port_block_ram_tb;  
  
parameter IN_PIXEL_W = 8;  
parameter IN_PIXEL_NUM = 16;  
  
parameter W_DATA = 128;  
parameter N_CELL = 2192;  
parameter W_CELL = $clog2(N_CELL);  
parameter FILENAME = "C:/dsd_ta/L09/simSR.hex"; You should change this line to your own directory!  
parameter N_DELAY = 1;  
  
...  
reg [IN_PIXEL_W-1:0] weight_store [0:IN_PIXEL_NUM-1];  
reg [IN_PIXEL_W:0] weight [0:IN_PIXEL_NUM-1];  
  
...  
// Clock  
parameter CLOCK_PERIOD = 10;      //100MHz  
initial begin  
    clk = 1'b0;  
    forever #(CLOCK_PERIOD/2) clk = ~clk;  
end
```

Parameters : In simSR.hex, a row contains (IN\_PIXEL\_NUM) values, each one being a (IN\_PIXEL\_W) bit value.

For waveform visualization : refer to page 10

Clock operation

# Test bench (dual\_port\_block\_ram\_tb.v)

```
integer i;  
  
// load 16 conv. filters of layer 1 from dual_port_block_ram  
initial begin  
  
    // initialization  
    ena = 0;  
    wea = 0 ;  
    addra = 0;  
    enb = 0;  
    addrb = 0;  
    dia = 0;  
  
    // set signal for dual_port_block_ram access  
    for(/* Insert your code */) begin  
        #(4*CLOCK_PERIOD)    enb = 1'b1;  
                            /* Insert your code */  
        #(CLOCK_PERIOD)      enb = 1'b0;  
    end  
  
end
```

Generate control signals

Set initial signals

Set signals for RAM access

# To do ...

- Complete missing codes (dual\_port\_block\_ram\_tb.v)

```
// load 16 conv. filters of layer 1 from dual_port_block_ram
initial begin
    // initialization
    ena = 0;
    wea = 0 ;
    addra = 0;
    enb = 0;
    addrb = 0;
    dia = 0;

    // set signal for dual port block ram access
    for(/* Insert your code */) begin
        #(4*CLOCK_PERIOD) enb = 1'b1;
        /* Insert your code */
        #(CLOCK_PERIOD)   enb = 1'b0;
    end
end
```

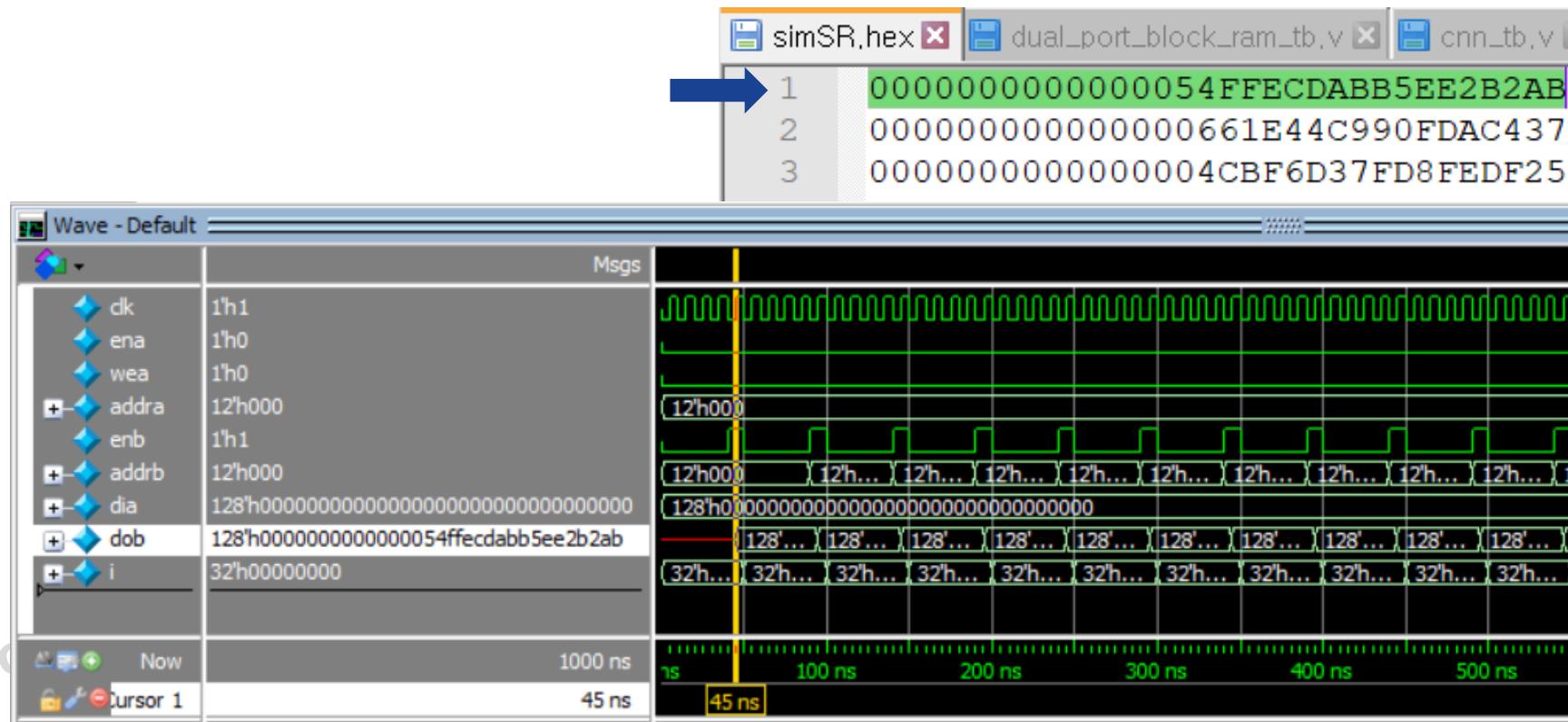
```
-----
// Visualize stored weights and weights
-----
integer j;

always@(posedge clk) begin
    if(enb) begin
        // TODO: set 'weight_store' and 'weight'
        /* Insert your code */

        /***** */
    end
end
endmodule
```

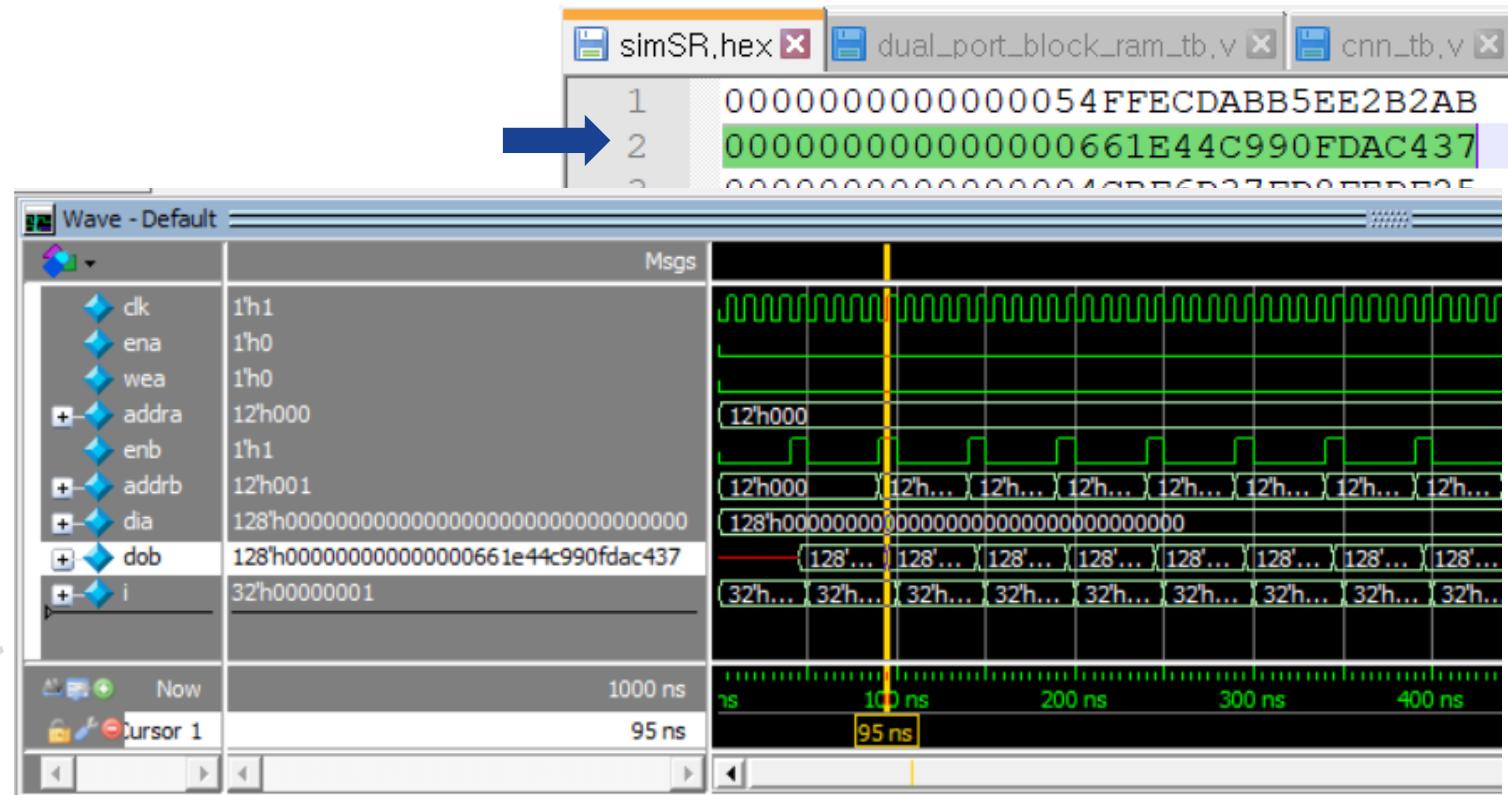
# Waveform

- Weight memory access:
  - The first set of filters



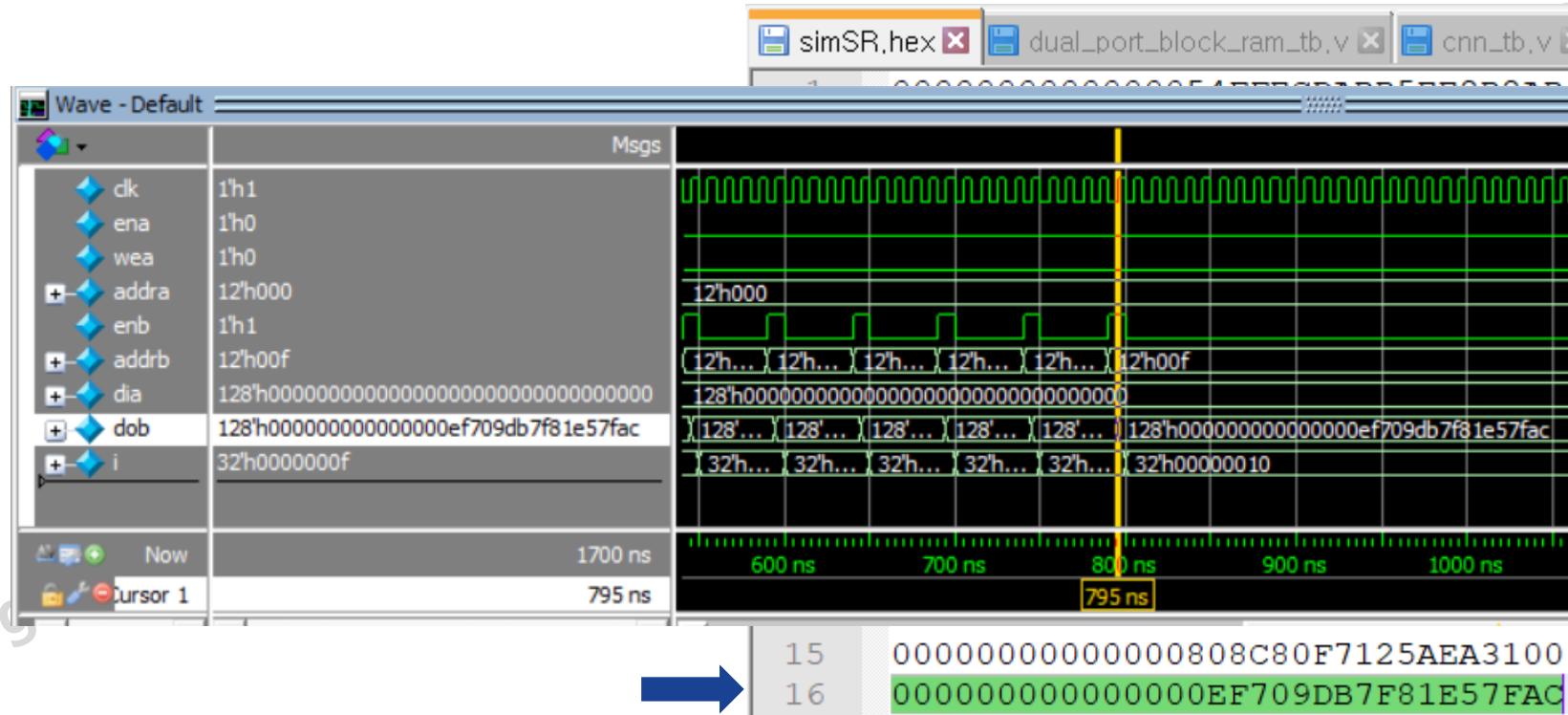
# Waveform

- Weight memory access:
  - The second set of filters

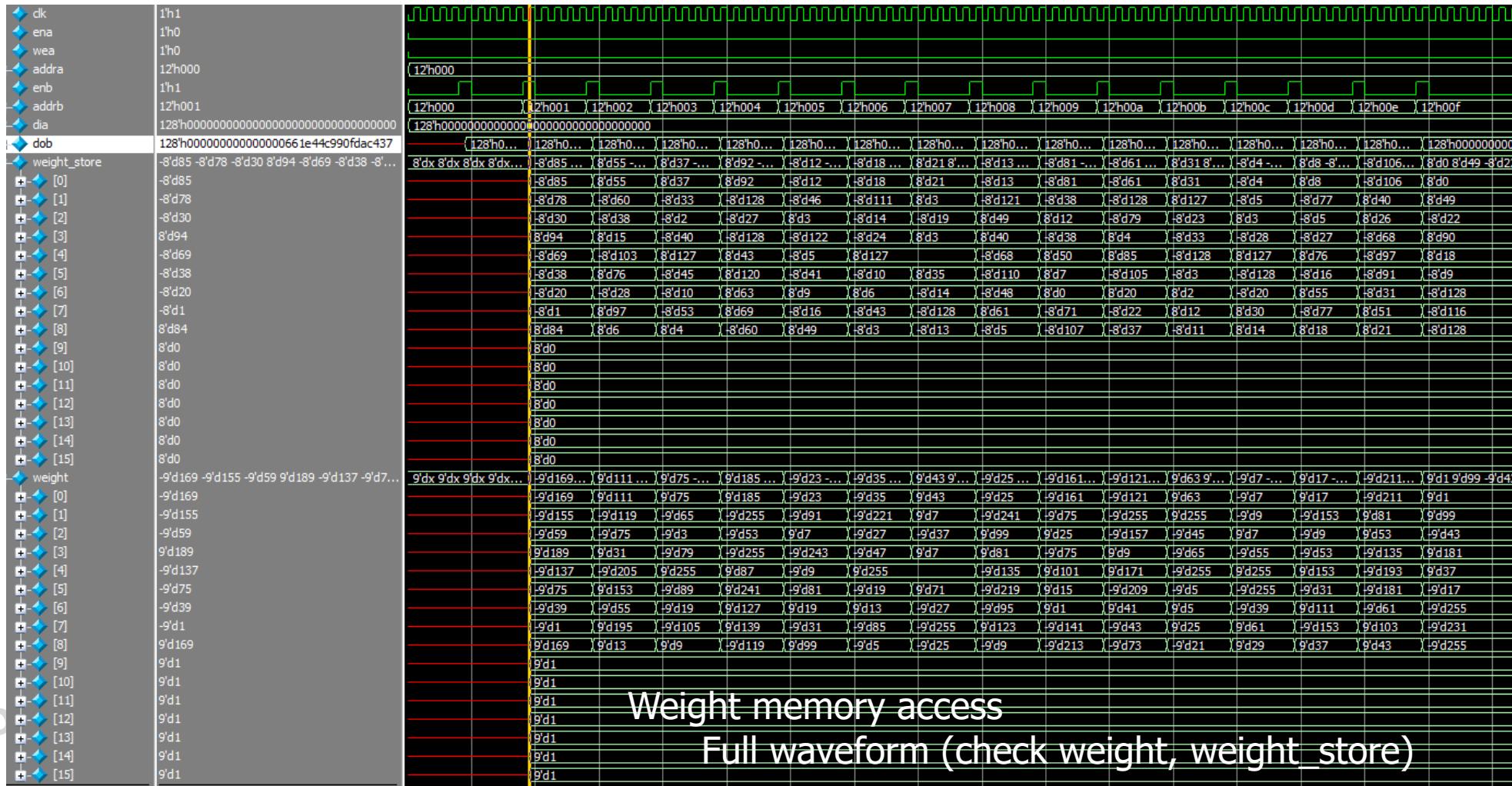


# Waveform

- Weight memory access
  - The 16<sup>th</sup> set of filters



# Waveform



# Road map

Deep Learning Accelerator

Convolutional Neural  
Network (CNN)

Quantization

Block memory

Reference S/W

# Lab 4: SRNPU reference S/W

- Top file (hw\_uniform\_architecture.m)
  - Define a network architecture (arch/sim\_espcn\_3x3.m)
  - Load weights from files
    - Do linear scale quantization
  - Do convolution
  - Evaluation

# Network architecture

- The network architecture is defined in the file arch/sim\_espncn\_3x3.m
- Each line defines the configurations of one layer including
  1. Convolution settings
  2. Activation quantization parameters
  3. Weight quantization parameters

```
architecture = { ...  
    ['conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1];  
    ['conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1];  
    ['conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1];  
    {'sr_flat'};  
    {'lp_sres'};  
};  
COP'
```

1

2

3

# Network architecture

- The network architecture is defined in the file arch/sim\_espncn\_3x3.m
- Convolution (conv)
  - Filter size (f) Padding (p) and stride (s), the number of filters.
  - Example:
    - Layer 1:
      - $f=3, p=2, s=1 \Rightarrow$  Filter 3x3
      - 16 output channels.

1

```
architecture = { ...  
    ['conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0 , ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1];  
    ['conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0 , ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1];  
    ['conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1];  
    {'sr_flat'};  
    {'lp_sres'};  
};  
COP'
```

# Network architecture

- The network architecture is defined in the file arch/sim\_espncn\_3x3.m
- Activation quantization: [nbit fbit sign]
  - The number of bit (nbit)
  - The number of fractional bits (fbit)
  - The signed bit
    - ReLU: 0
    - Linear: 1

2

```
architecture = { ...
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};
    {'conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};
    {'sr_flat'};
    {'lp_sres'};
};

COP'
```

# Network architecture

- The network architecture is defined in the file arch/sim\_espncn\_3x3.m
- Weight quantization: [nbit fbit sign]
  - The number of bit (nbit)
  - The number of fractional bits (fbit)
  - The signed bit
    - ReLU: 0
    - Linear: 1

3

```
architecture = { ...
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0 , ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};
    {'conv', 0, ps.conv_f3_p2_s1, 16, ps.act_relu_8_8_0 , ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};
    {'conv', 0, ps.conv_f3_p2_s1, 4, ps.act_lineq_8_8_1, ps.wts_scale_linear_8, ps.scales_16_1_1, ps.biases_16_10_1};
    {'sr_flat'};
    {'lp_sres'};
};

COV'
```

# Load weights

- Load weights from files
- Do quantization on weights, biases and scales.

```
case 'conv'
    [batchnorm, convol_settings, output_channels, ~, weights_settings] = architecture{i}{2:6};
    [kernel_size, ~, ~] = convol_settings{::};
    wts_scheme = weights_settings{1};

    input_channels = all_input_channels(i);
    param_file = [model_prefix num2str(i-1)];

    % Load weights, biases and scales from files
    [weightx, bias, scale, rolling mean, rolling var] = read conv param module(param file, kernel size, input channels, output channels);

    % Quantization
    if strcmp(wts_scheme, 'none')
        weight = weightx;
        w_bonus_scale_factor = ones(output_channels,1);
    elseif strcmp(wts_scheme, 'uniform')
        [wts_nbit, wts_fbit, ~] = weights_settings{2:end};
        wts_step = 2^-wts_fbit;
        [weight, ~] = uniform_quantize(weightx, wts_step, wts_nbit);
        w_bonus_scale_factor = ones(output_channels,1);
    elseif strcmp(wts_scheme, 'scale_linear')
        wts_nlevel = 2^weights_settings{2};
        [weight, w_bonus_scale_factor] = scale_linear_quantize(weightx, output_channels, wts_nlevel/2);
    elseif strcmp(wts_scheme, 'scale_linear_float')
        wts_nlevel = 2^weights_settings{2};
        [weight, w_bonus_scale_factor] = scale_linear_quantize_float(weightx, output_channels, wts_nlevel/2);
    end

    weights{i} = permute(reshape(weight, kernel_size, kernel_size, input_channels, output_channels), [2, 1, 3, 4]);
```

# Input preparation

- Load a test image from a file (imread)
- Resize an image to generate a Low-resolution image (imresize)
- Convert an RGB to YCbCr
  - Only work with the Y-channel image

```
%> work on illuminance only
im = imread(image_name);
input_img = modcrop(im, up_scale);
input_img = single(input_img)/255;
input_img = imresize(input_img, 1/up_scale, 'bicubic');
if size(im,3) > 1
    im_ycbcr = rgb2ycbcr(im);
    im = im_ycbcr(:,:,1);
end
im_gnd = modcrop(im, up_scale);
im_gnd = single(im_gnd)/255;
im_l = imresize(im_gnd, 1/up_scale, 'bicubic');

input = floor(im_l(:,:,1) * 255);
```



High-resolution



Low-resolution

# Do convolution

- Load quantized weights
- Do convolution (convol2)
- Do activation quantization (hwu relu quantize/hwu linear quantize)

```
weight = weights{i};

conv_out = convol2(input, weight, stride, pad);
for j = 1:size(conv_out, 3)
    conv_out(:,:,j) = conv_out(:,:,j) .* scales{i}(j);
    conv_out(:,:,j) = floor(conv_out(:,:,j) / 2^bit_shift(i)); %if floating point is
    conv_out(:,:,j) = conv_out(:,:,j) + biases{i}(j);
end

if strcmp(activation, 'float_relu')
    output = hwu_float_relu_activate(conv_out);
elseif strcmp(activation, 'relu')
    [act_nbit, act_fbit] = act_settings{2:3};
    act_step = 2^-act_fbit;
    [output, ~] = hwu_relu_quantize(conv_out, act_step, act_nbit, biases_fbit);
elseif strcmp(activation, 'line_q')
    [act_nbit, act_fbit] = act_settings{2:3};
    act_step = 2^-act_fbit;
    [output, ~] = hwu_linear_quantize(conv_out, act_step, act_nbit, biases_fbit);
else
    output = conv_out;
end
```

Copyright 2

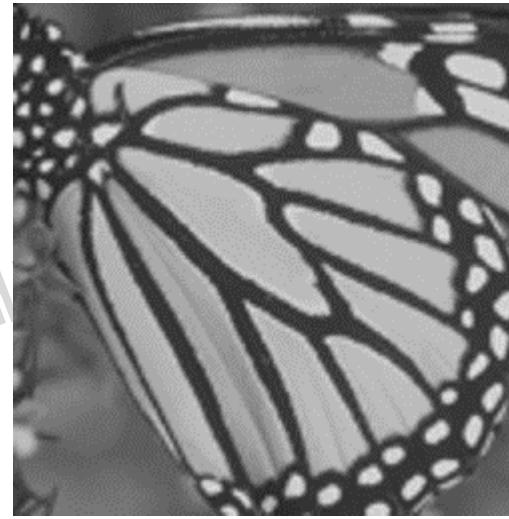
, reserved.

# Evaluation

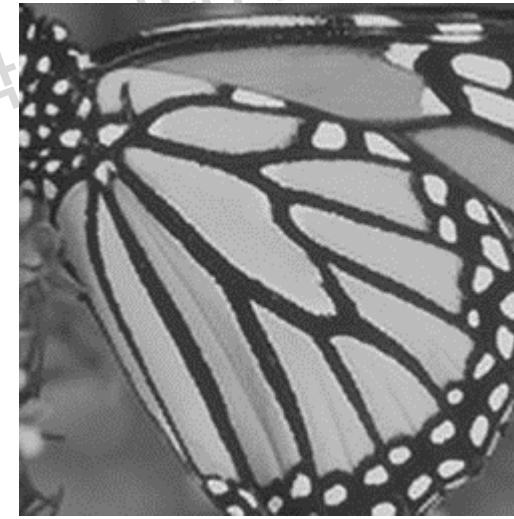
- How to evaluation?
  - Compare an approximated image with the original image.
- Interpolation
  - Bicubic: 27.43 dB, mean-average-error = 6.4
  - CNN-based SR: 30.98 dB, mean-average-error = 4.5



Low-resolution



Bicubic Interpolation



CNN-based SR

# To do ...

- Complete the missing codes (**Copied from the previous exercise**)
  - Convolution: func/convol2.m
  - Weight/Bias quantization: func/uniform\_quantize.m
  - Activation Quantization:
    - ReLU: func/hwu\_relu\_quantize.m
    - Linear: func/hwu\_linear\_quantize.m
- Run the test