深度學習實作與應用
Deep learning and its applications

Convolutional Networks

IM5062, Spring 2024

Some slides adopted from Alex Smola, Chien-Yao Wang

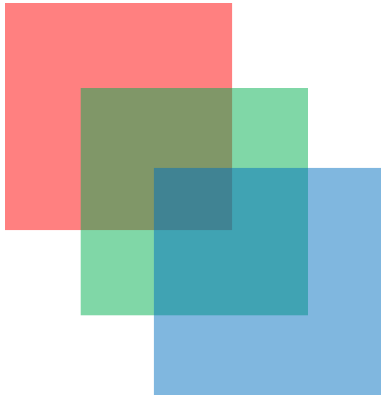
黃意婷

Outline

- What is Convolution
- Use Convolution
- Convolution (d2l.ch7)
  - Kernel/Filter
  - Padding
  - Stride
- Pooling
  - Max/Average Pooling
- Multiple Channels
- Summary

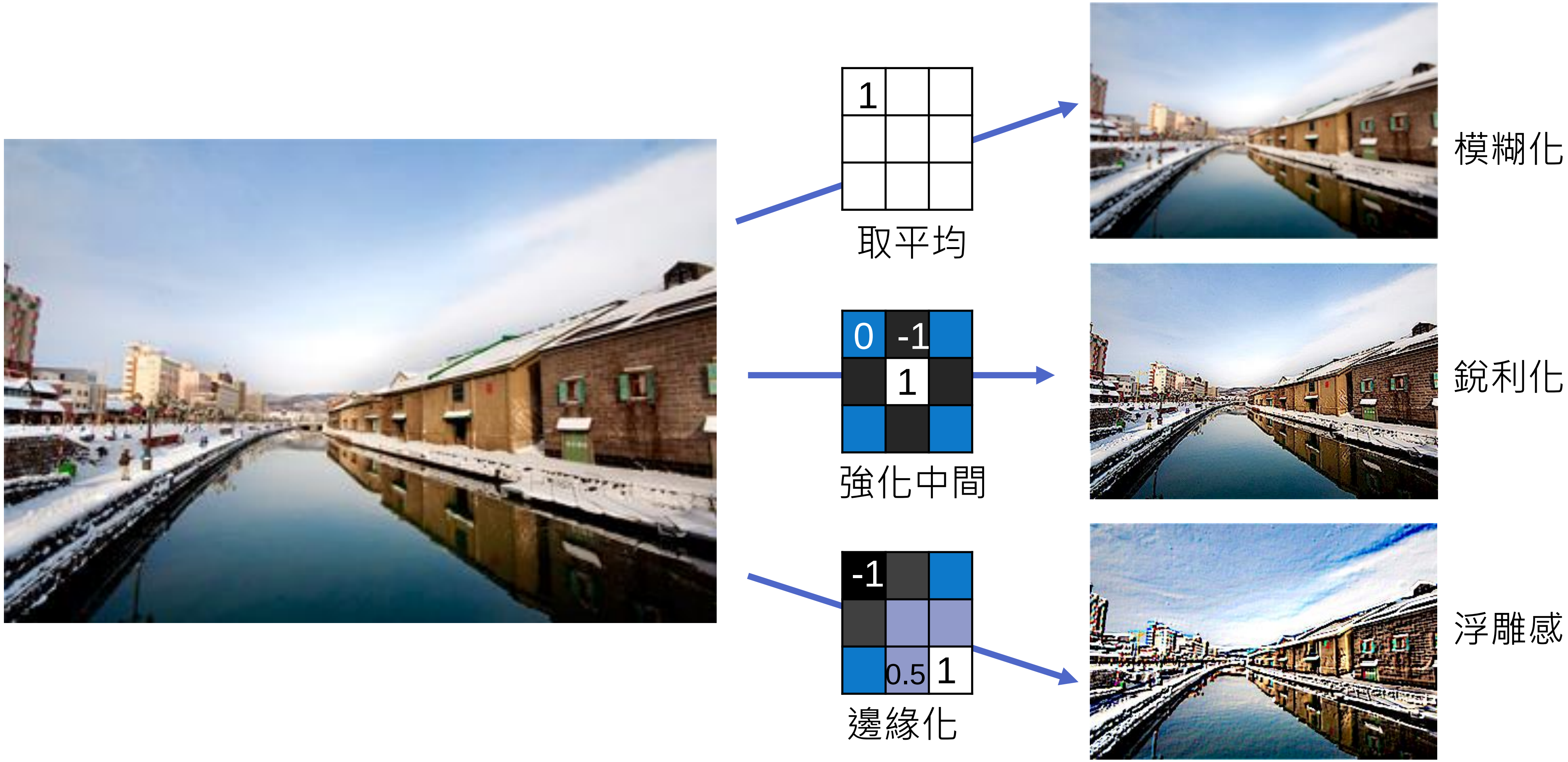
- LeNet/AlexNet (d2l.ch8.1)
- CNN for Go & Text Classification
- Batch normalization (d2l.ch8.5)
- Network in Network (d2l.ch8.3)
- RasNet (d2l.ch8.6)

Convolution



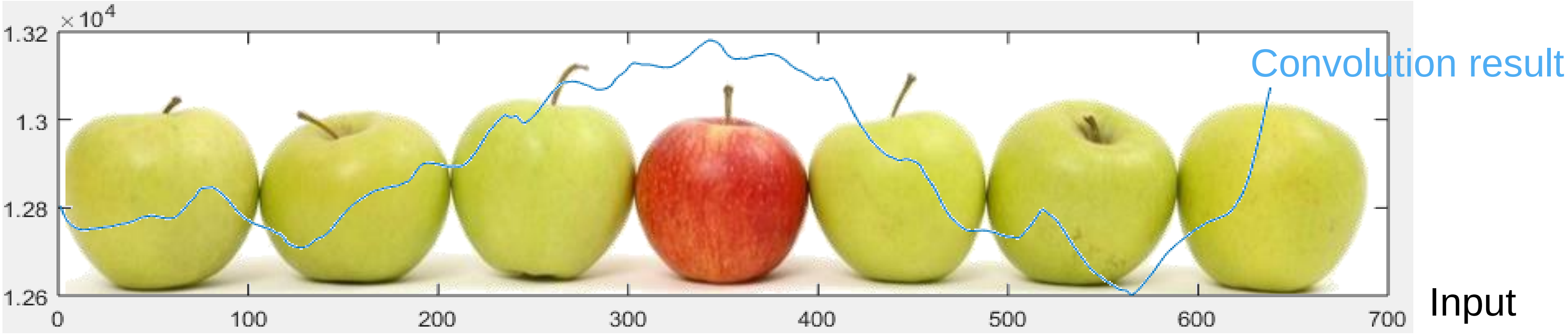
Convolution

-1 是弱化
1 是強化



Source

Why use Convolution



Input

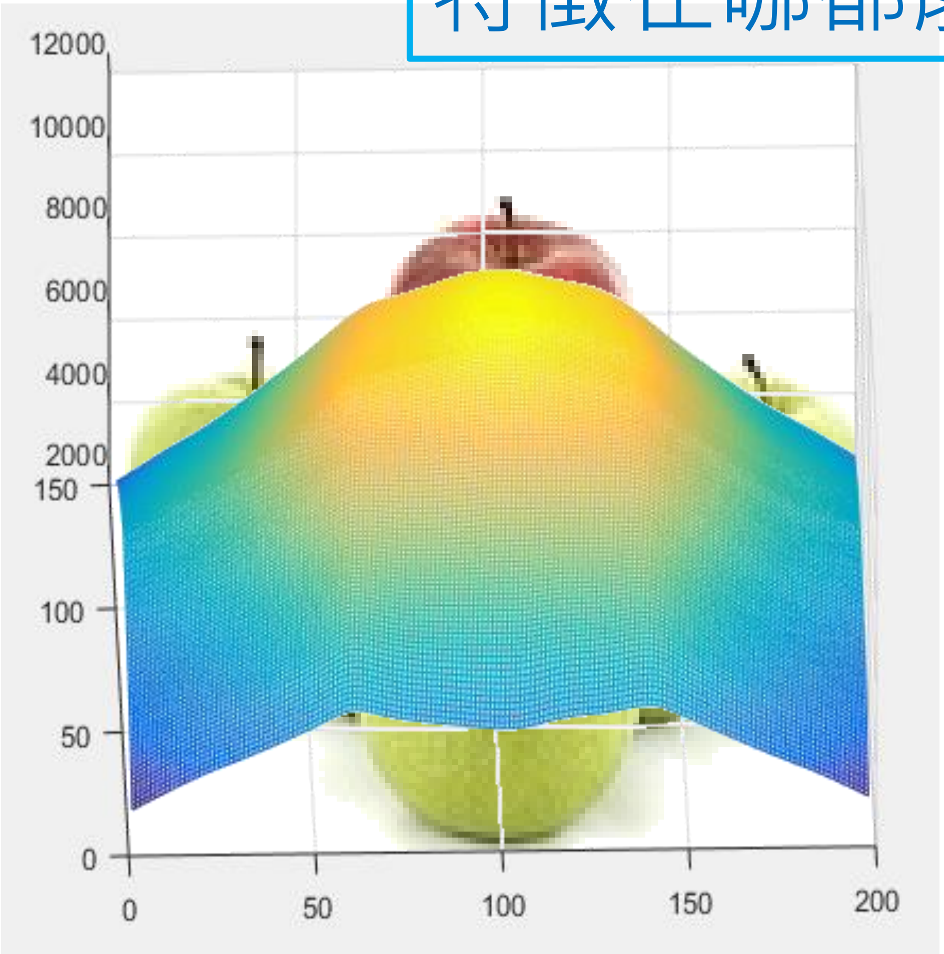
Filter

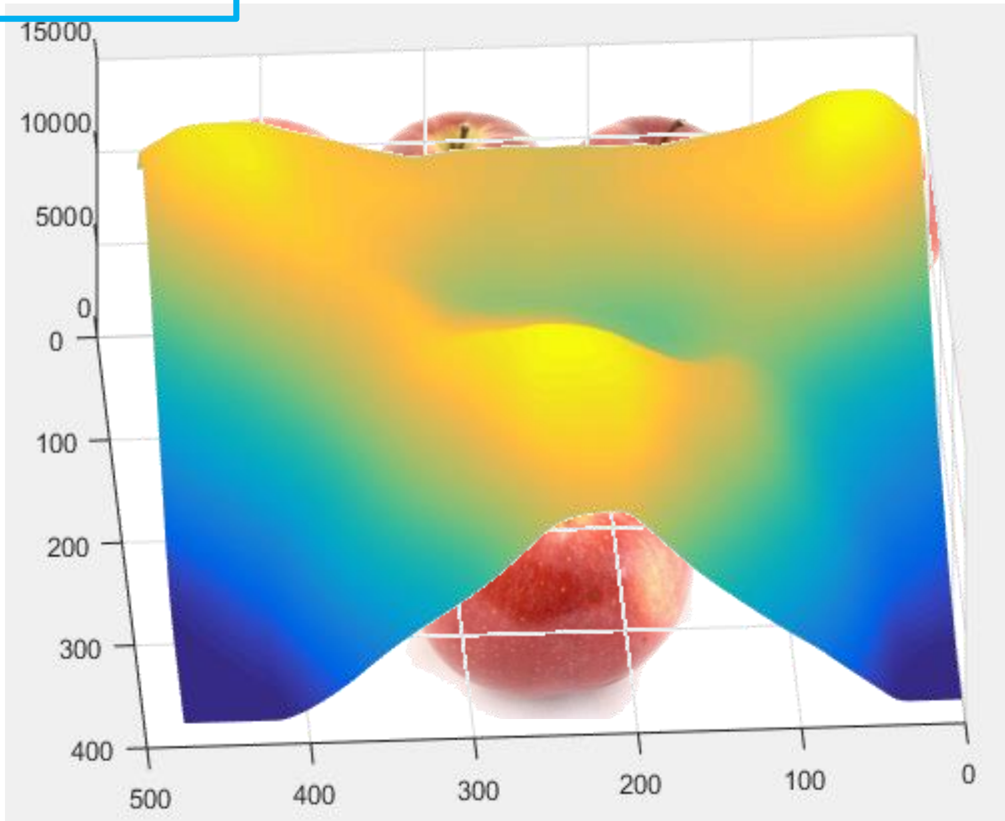
可以找到特定的特徵

Why use Convolution

Translation Invariance

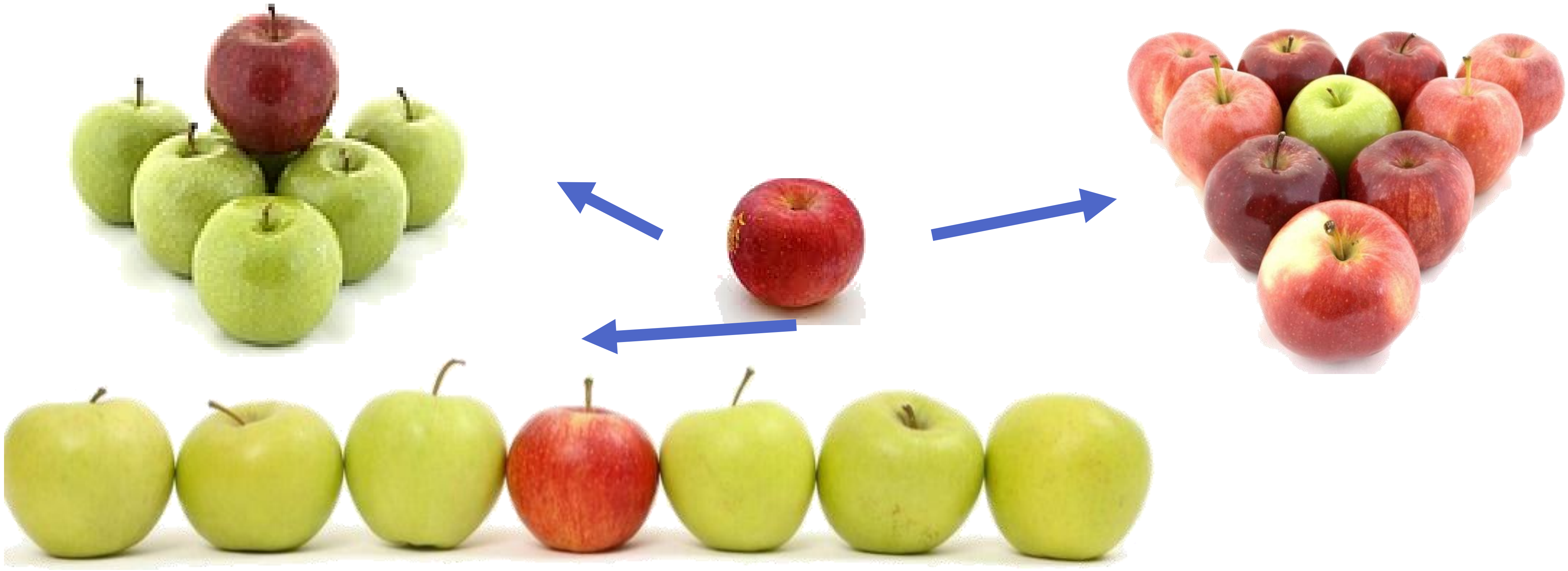
特徵在哪都能被找到





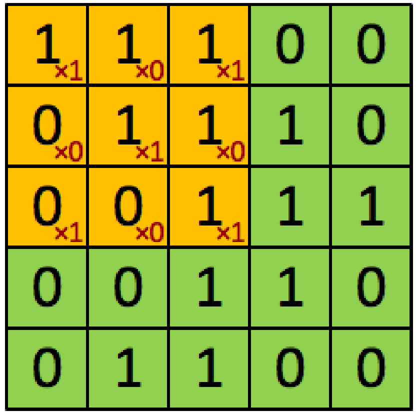
Why use Convolution

任何輸入大小都能做

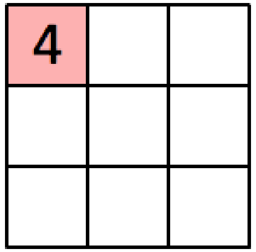


Convolutional Neural Network

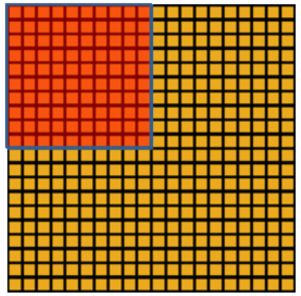
- Convolution
- Pooling
- Full Connection



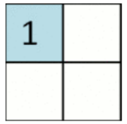
Image



Convolved Feature



Convolved feature



Pooled feature

Full Connection

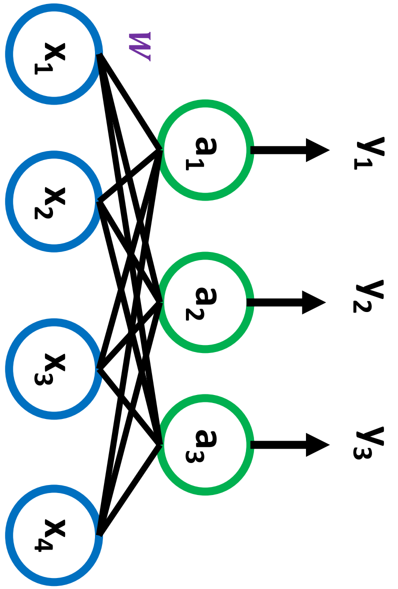
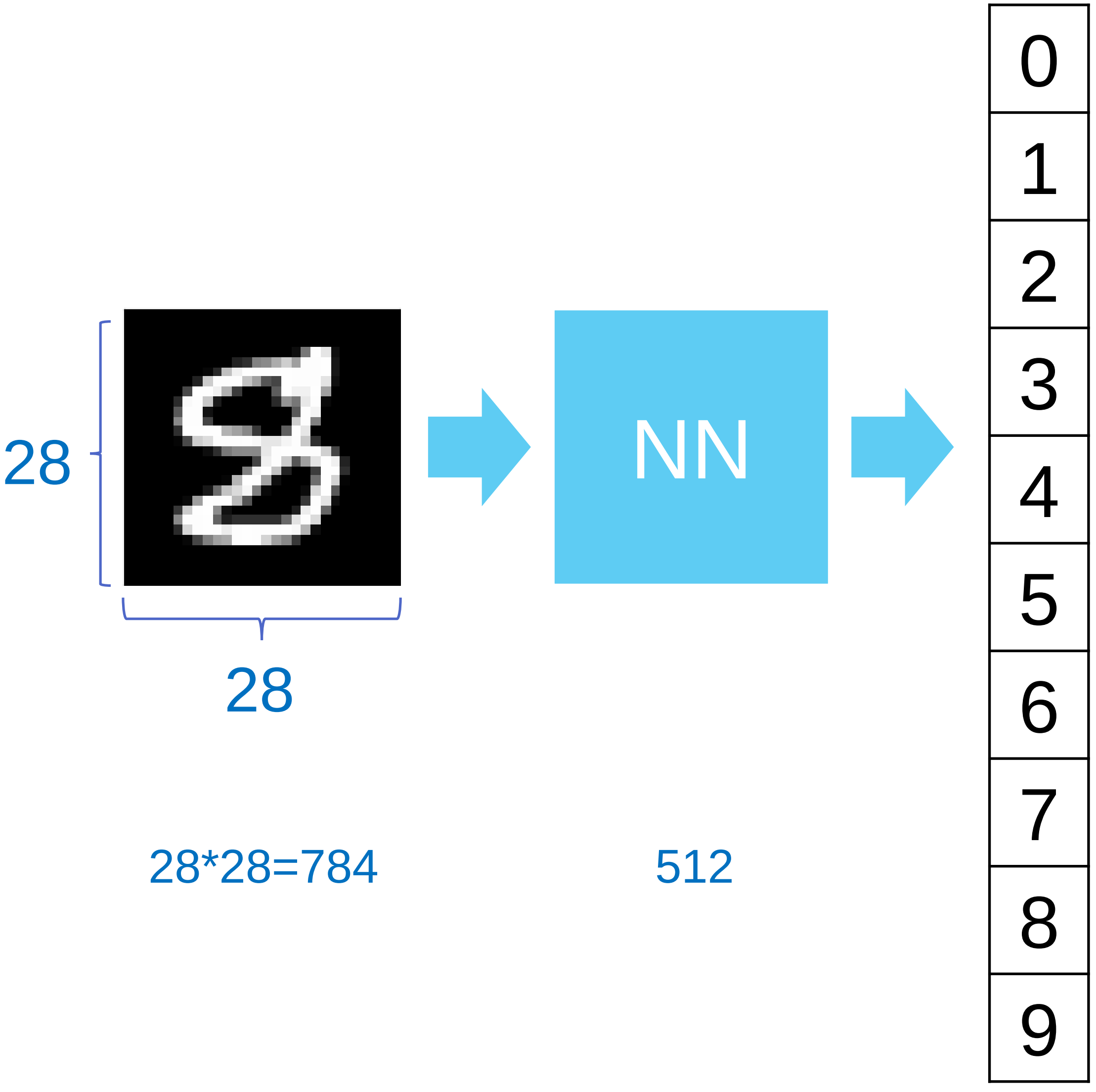


Image classification

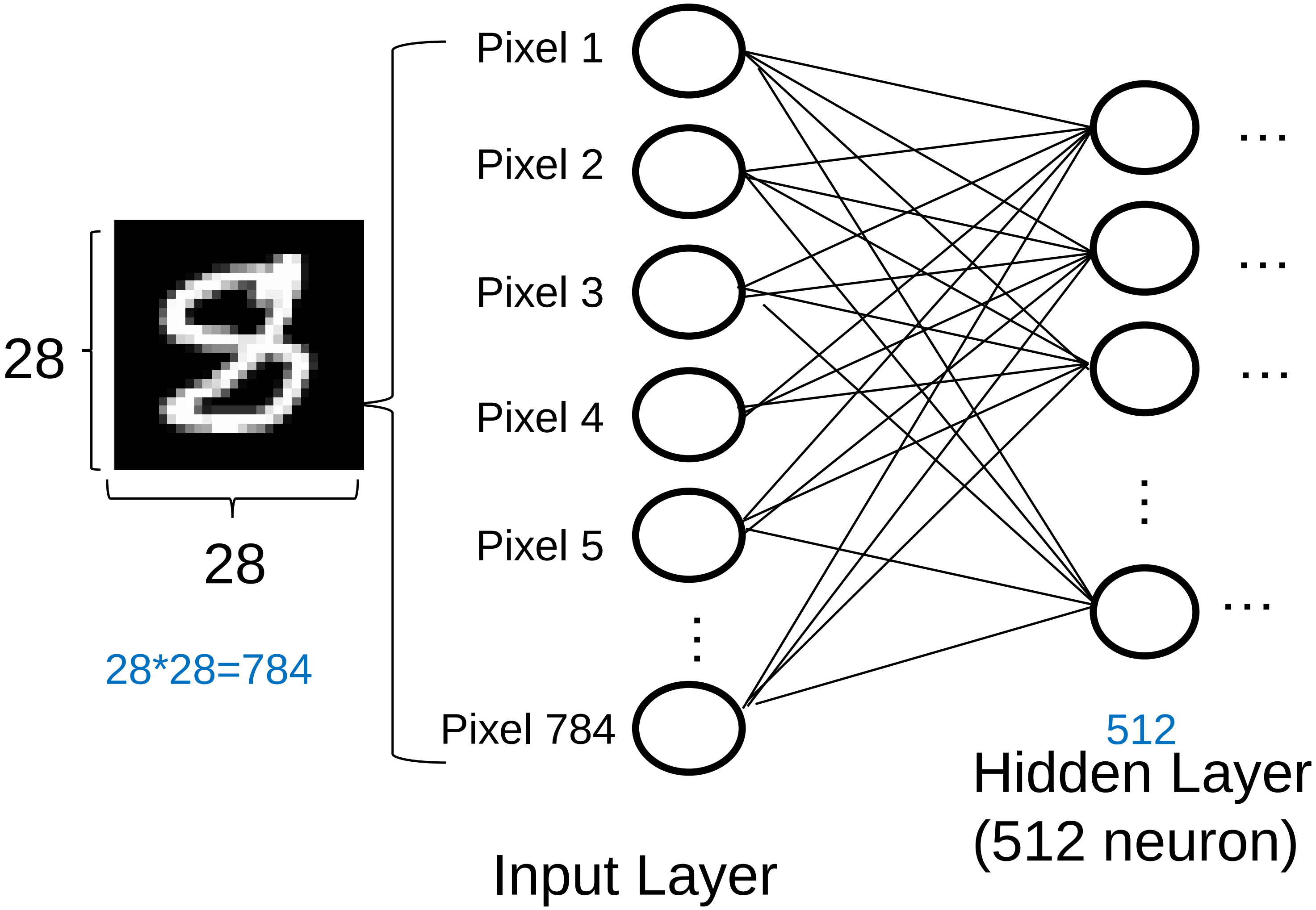
MINST dataset

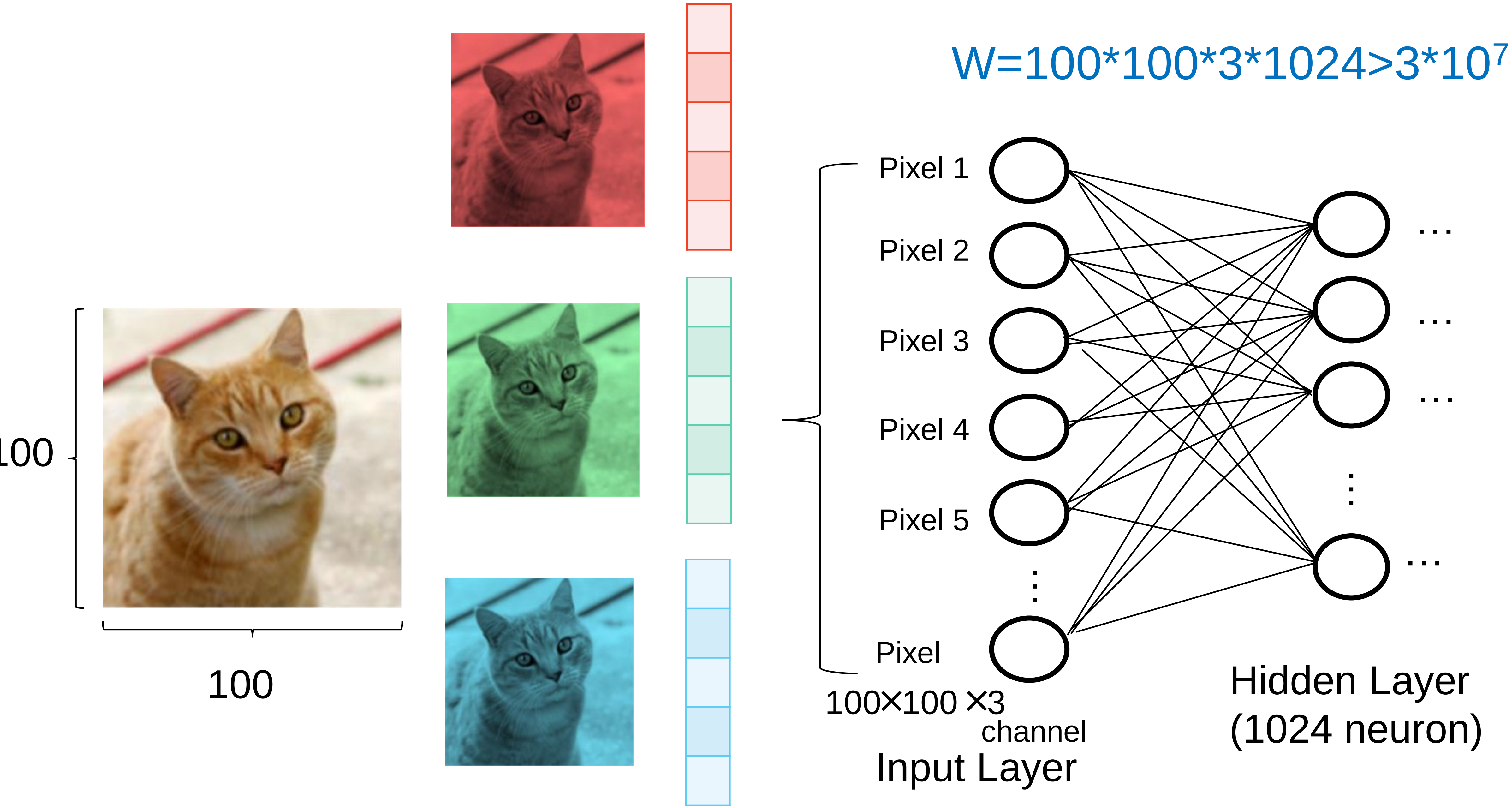


10

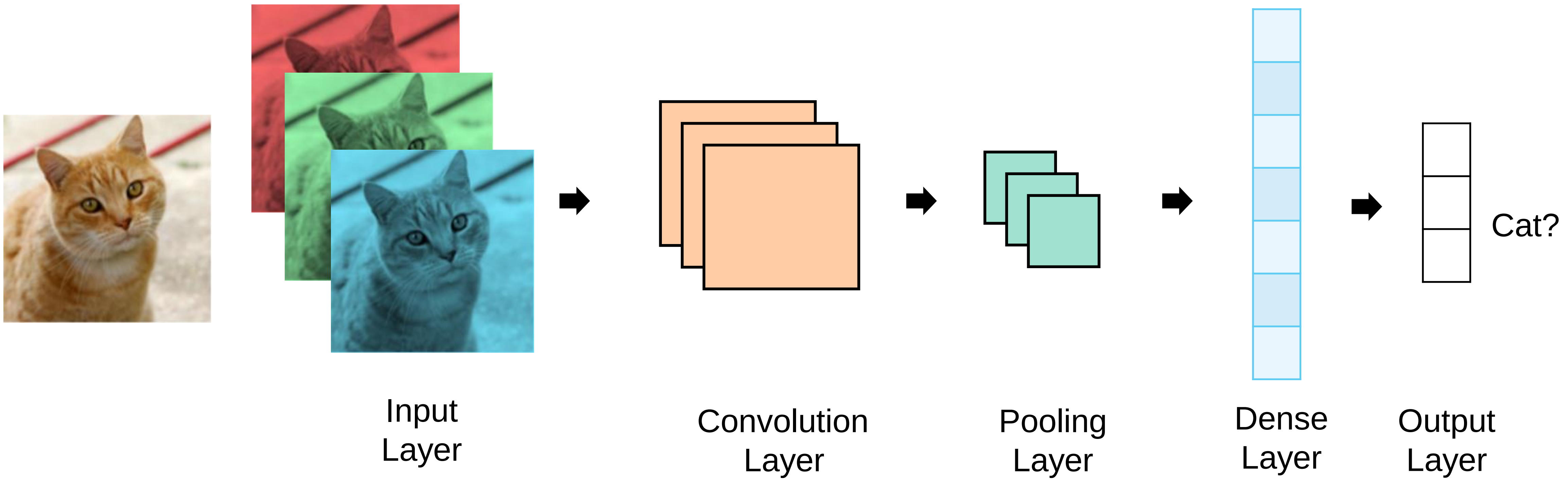
Image classification

W=784*512=401,408




W=100*100*3*1024>3*10^7
100
100
Pixel 1
Pixel 2
Pixel 3
Pixel 4
Pixel 5
Pixel
100×100 ×3
channel
Input Layer
Hidden Layer
(1024 neuron)

Convolutional network



2D convolutional layer

Input X

0	1	2
3	4	5
6	7	8

*

Kernel W (Filter)

0	1
2	3

=

Output Y

19	

$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$

$n_h \times n_w$
3×3

$k_h \times k_w$
2×2

$(n_h - k_h + 1) \times (n_w - k_w + 1)$
$(3 - 2 + 1) \times (3 - 2 + 1)$
2×2

2D convolutional layer

Input X | Kernel W (Filter) | Output Y

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|}\hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}$$

0 × 0 + 1 × 1 + 3 × 2 + 4 × 3 = 19
1 × 0 + 2 × 1 + 4 × 2 + 5 × 3 = 25

$n_h \times n_w$
3×3

$k_h \times k_w$
2×2

$(n_h - k_h + 1) \times (n_w - k_w + 1)$
$(3 - 2 + 1) \times (3 - 2 + 1)$
2×2

2D convolutional layer

Input X		Kernel W (Filter)		Output Y
$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$	*	$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$	=	$\begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$
$n_h \times n_w$ 3×3		$k_h \times k_w$ 2×2		$(n_h - k_h + 1) \times (n_w - k_w + 1)$ $(3 - 2 + 1) \times (3 - 2 + 1)$ 2×2

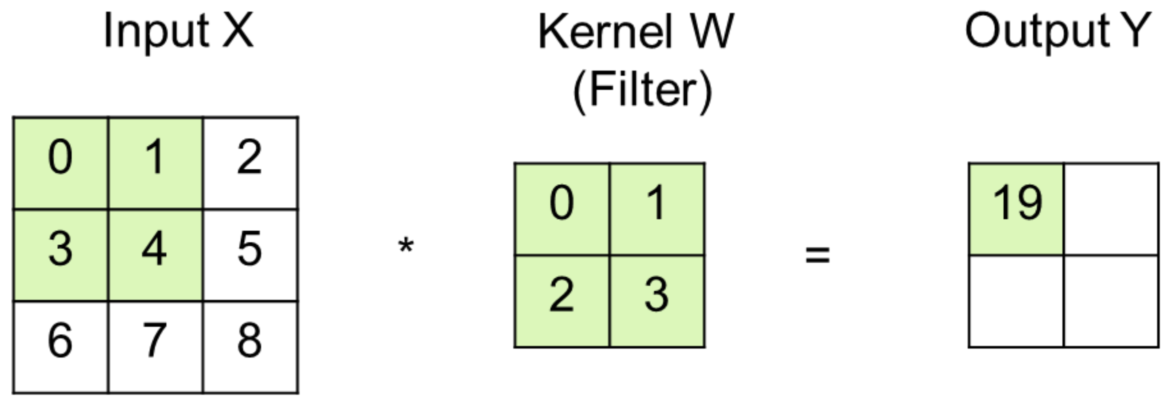
$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$

$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25$

$\mathbf{3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37}$

$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43$

2D convolutional layer



- $\boldsymbol{X}$: $n_h \times n_w$ input matrix
- $\boldsymbol{W}$: $k_h \times k_w$ kernel matrix
- b: scalar bias
- $\boldsymbol{Y}$: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\boldsymbol{Y} = \boldsymbol{X} \otimes \boldsymbol{W} + b$$

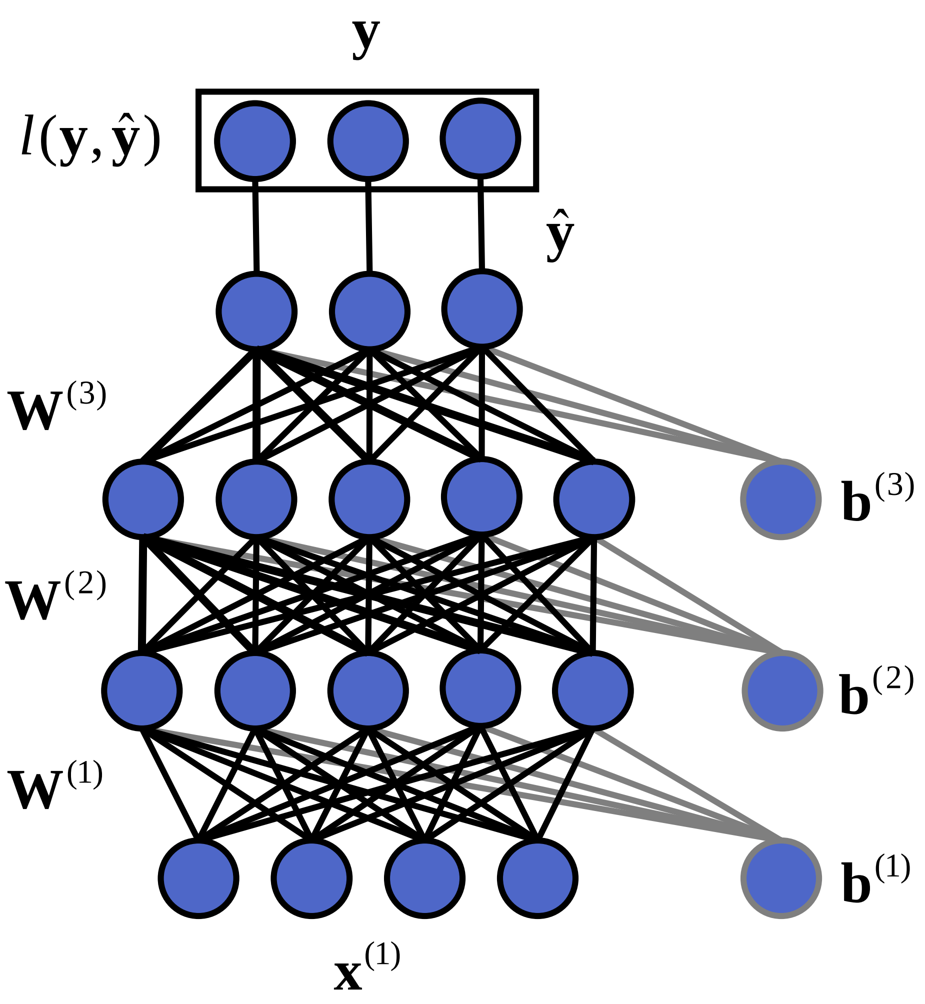
- **W** and b are learnable parameters.

Fully-Connected Multilayer Perceptron

5*3+3

5*5+5

4*5+5

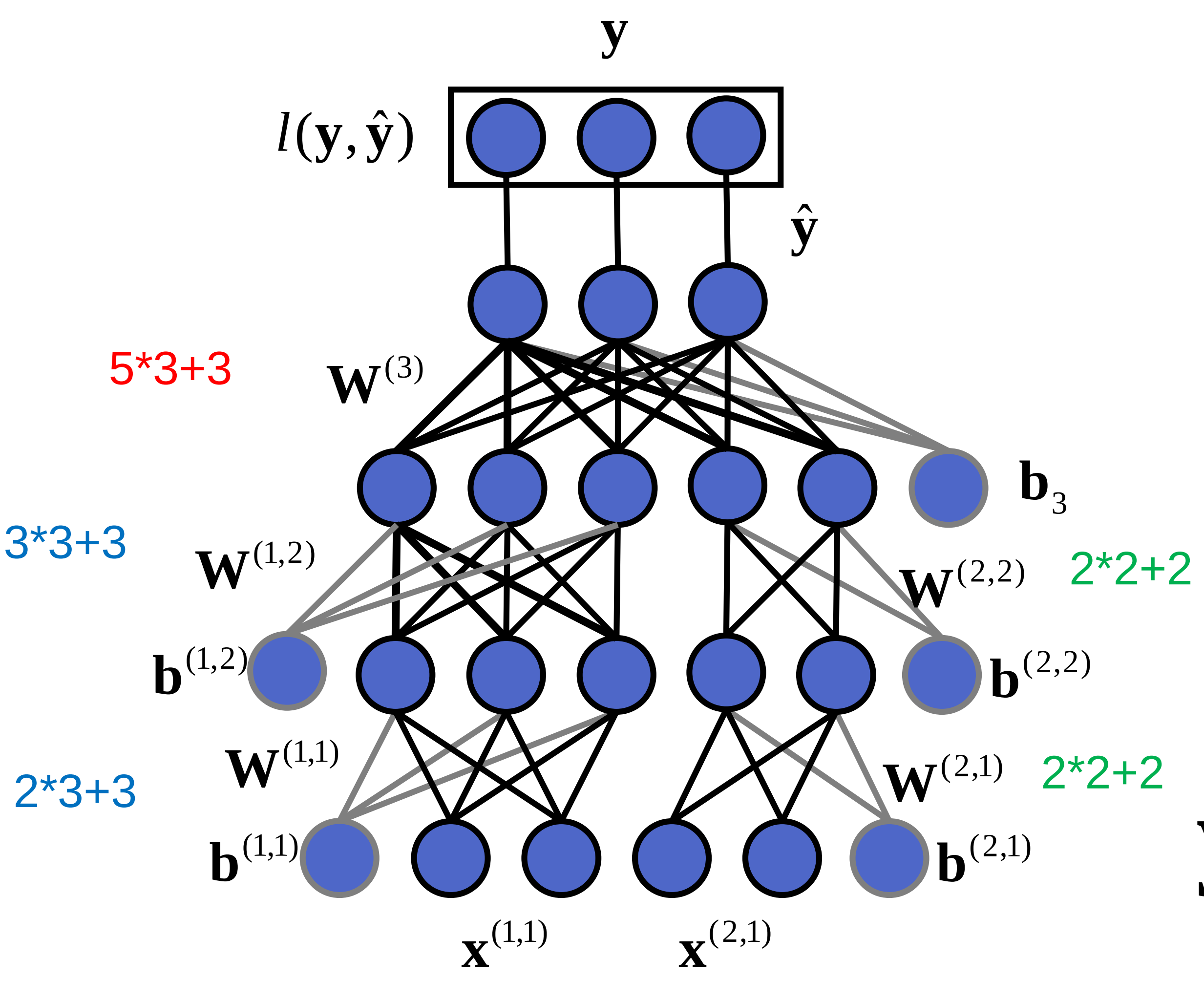


$$\mathbf{x}^{(2)} = \mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)}$$

$$\mathbf{x}^{(3)} = \mathbf{W}^{(2)}\mathbf{x}^{(2)} + \mathbf{b}^{(2)}$$

$$\hat{\mathbf{y}} = \mathbf{W}^{(3)}\mathbf{x}^{(3)} + \mathbf{b}^{(3)}$$

Locally-Connected Multilayer Perceptron



$$\mathbf{x}^{(1,2)} = \mathbf{W}^{(1,1)}\mathbf{x}^{(1,1)} + \mathbf{b}^{(1,1)}$$

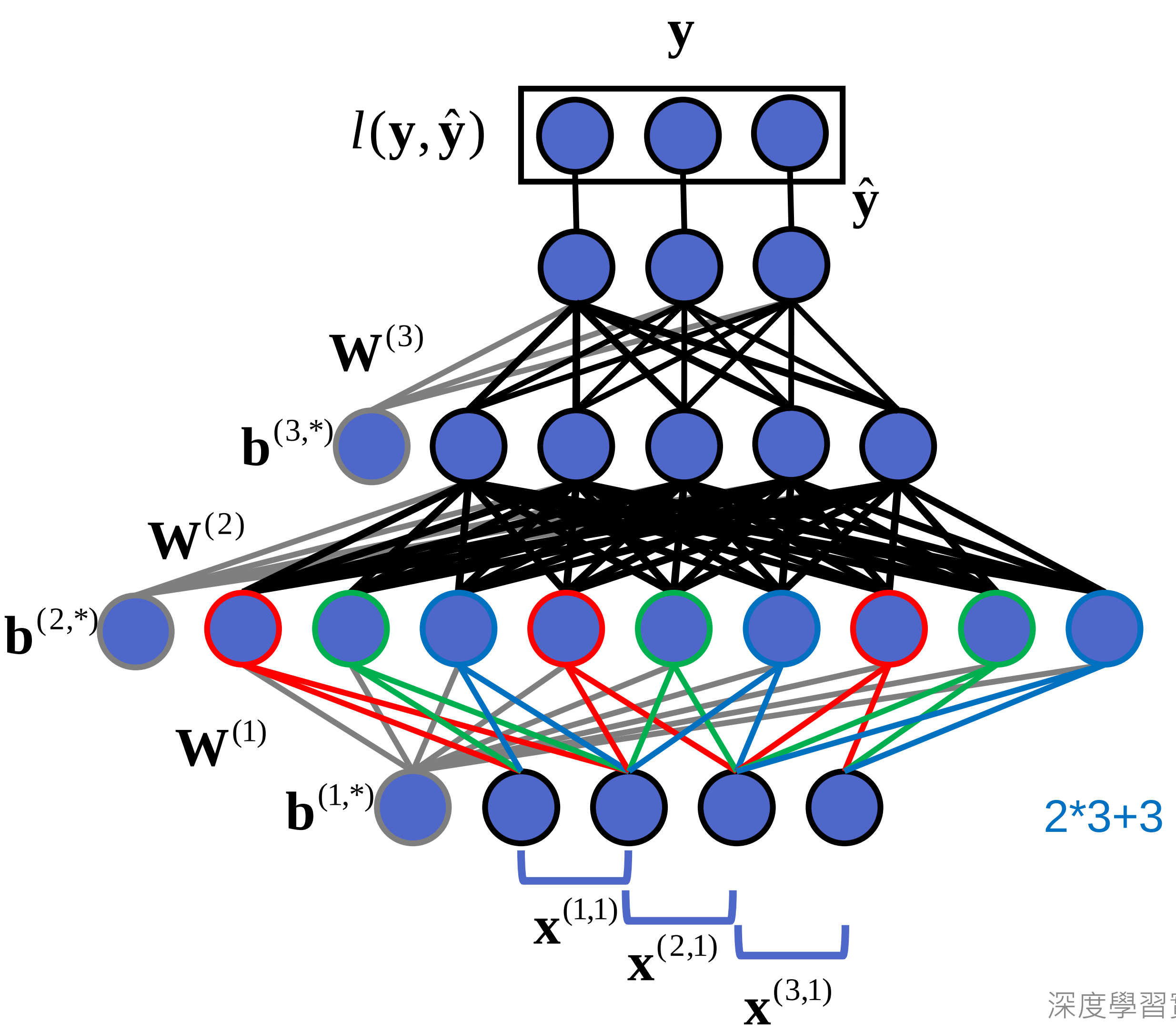
$$\mathbf{x}^{(1,3)} = \mathbf{W}^{(1,2)}\mathbf{x}^{(1,2)} + \mathbf{b}^{(1,2)}$$

$$\mathbf{x}^{(2,2)} = \mathbf{W}^{(2,1)}\mathbf{x}^{(2,1)} + \mathbf{b}^{(2,1)}$$

$$\mathbf{x}^{(2,3)} = \mathbf{W}^{(2,2)}\mathbf{x}^{(2,2)} + \mathbf{b}^{(2,2)}$$

$$\hat{\mathbf{y}} = \mathbf{W}^{(3)}[\mathbf{x}^{(1,3)}, \mathbf{x}^{(2,3)}] + \mathbf{b}^{(3)}$$

Convolutional Neural Network

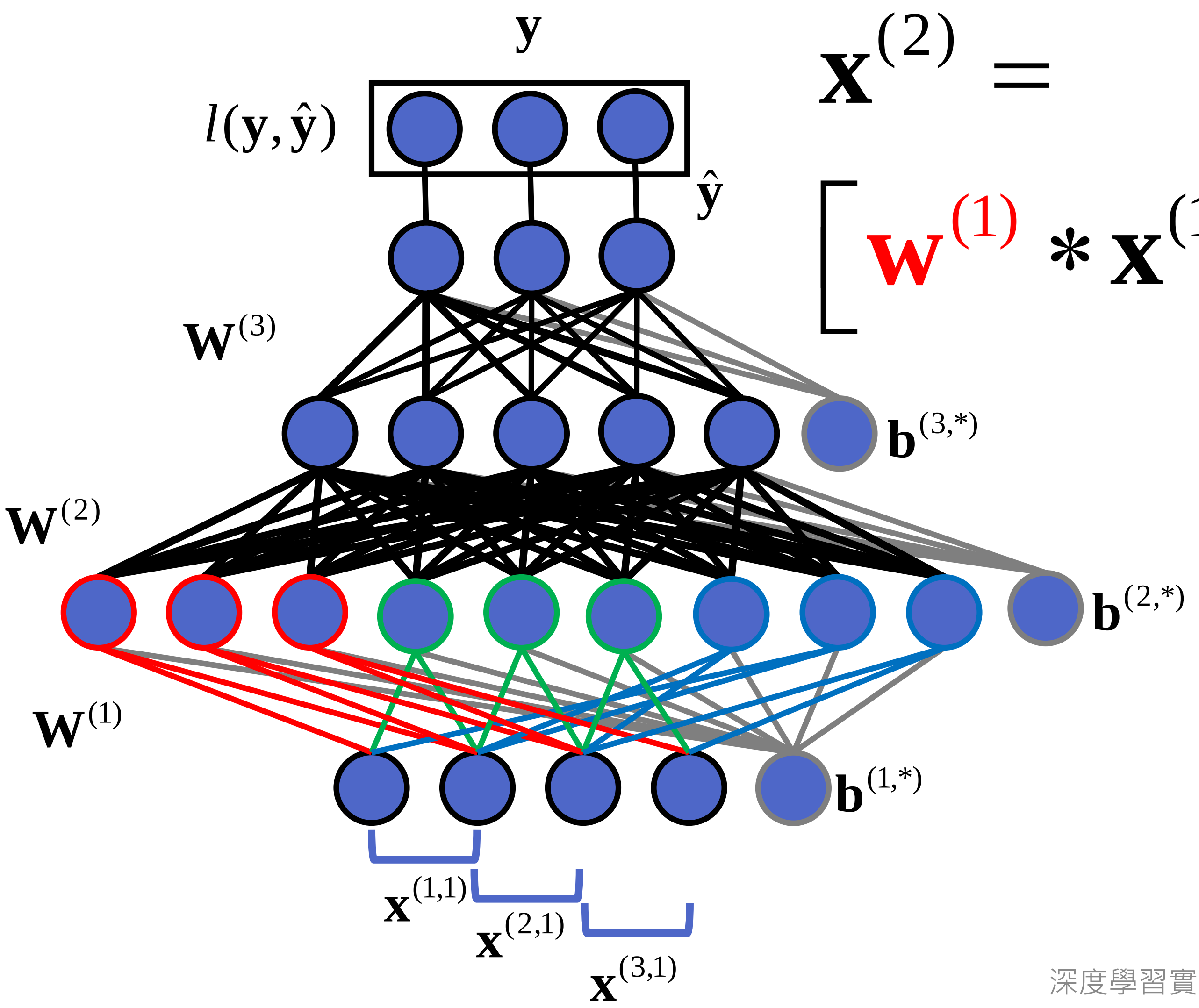


$$\mathbf{x}^{(2)} = \mathbf{W}^{(1)} * \mathbf{x}^{(1)} + \mathbf{b}^{(1)}$$

$$\mathbf{x}^{(3)} = \mathbf{W}^{(2)}\mathbf{x}^{(2)} + \mathbf{b}^{(2)}$$

$$\hat{\mathbf{y}} = \mathbf{W}^{(3)}\mathbf{x}^{(3)} + \mathbf{b}^{(3)}$$

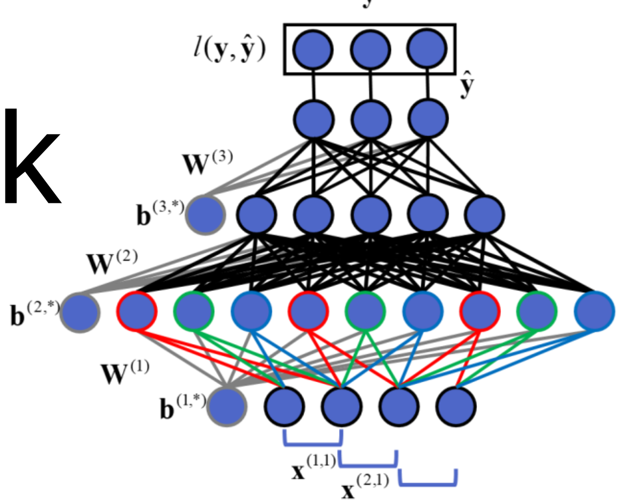
Convolutional Neural Network



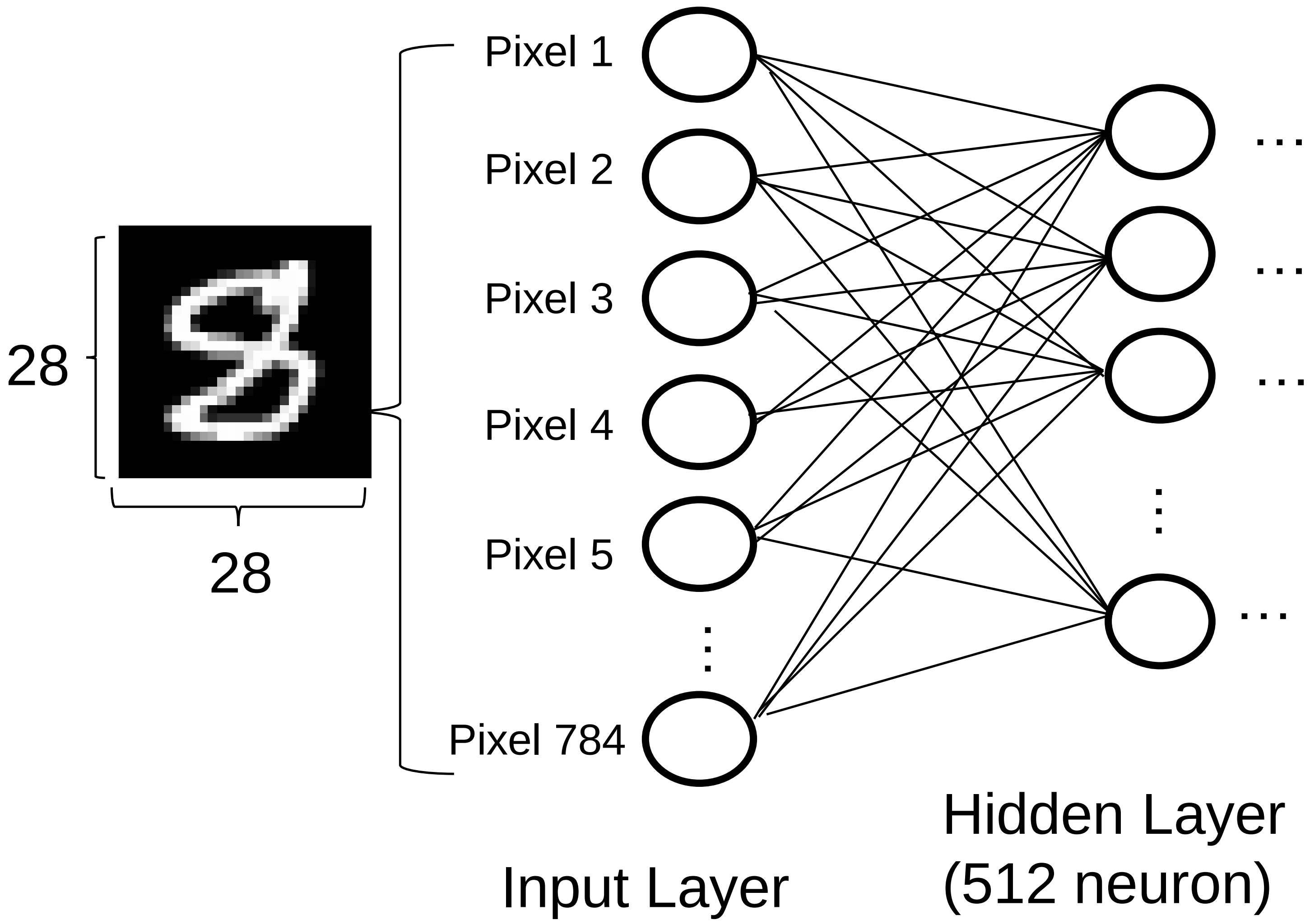
$$\mathbf{x}^{(2)} = \left[\color{red}{\mathbf{w}^{(1)}} * \mathbf{x}^{(1)}, \color{blue}{\mathbf{w}^{(1)}} * \mathbf{x}^{(1)}, \color{green}{\mathbf{w}^{(1)}} * \mathbf{x}^{(1)}\right] + \mathbf{b}^{(1)}$$

$$\mathbf{x}^{(3)} = \mathbf{W}^{(2)}\mathbf{x}^{(2)} + \mathbf{b}^{(2)}$$

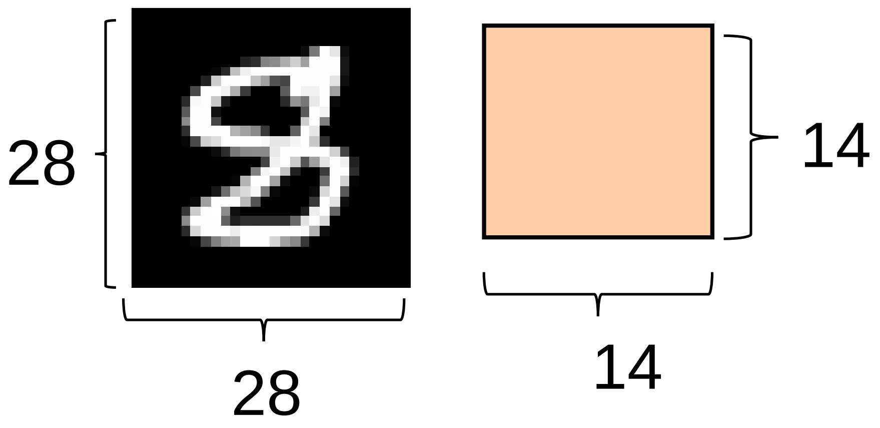
$$\hat{\mathbf{y}} = \mathbf{W}^{(3)}\mathbf{x}^{(3)} + \mathbf{b}^{(3)}$$



DNN & CNN

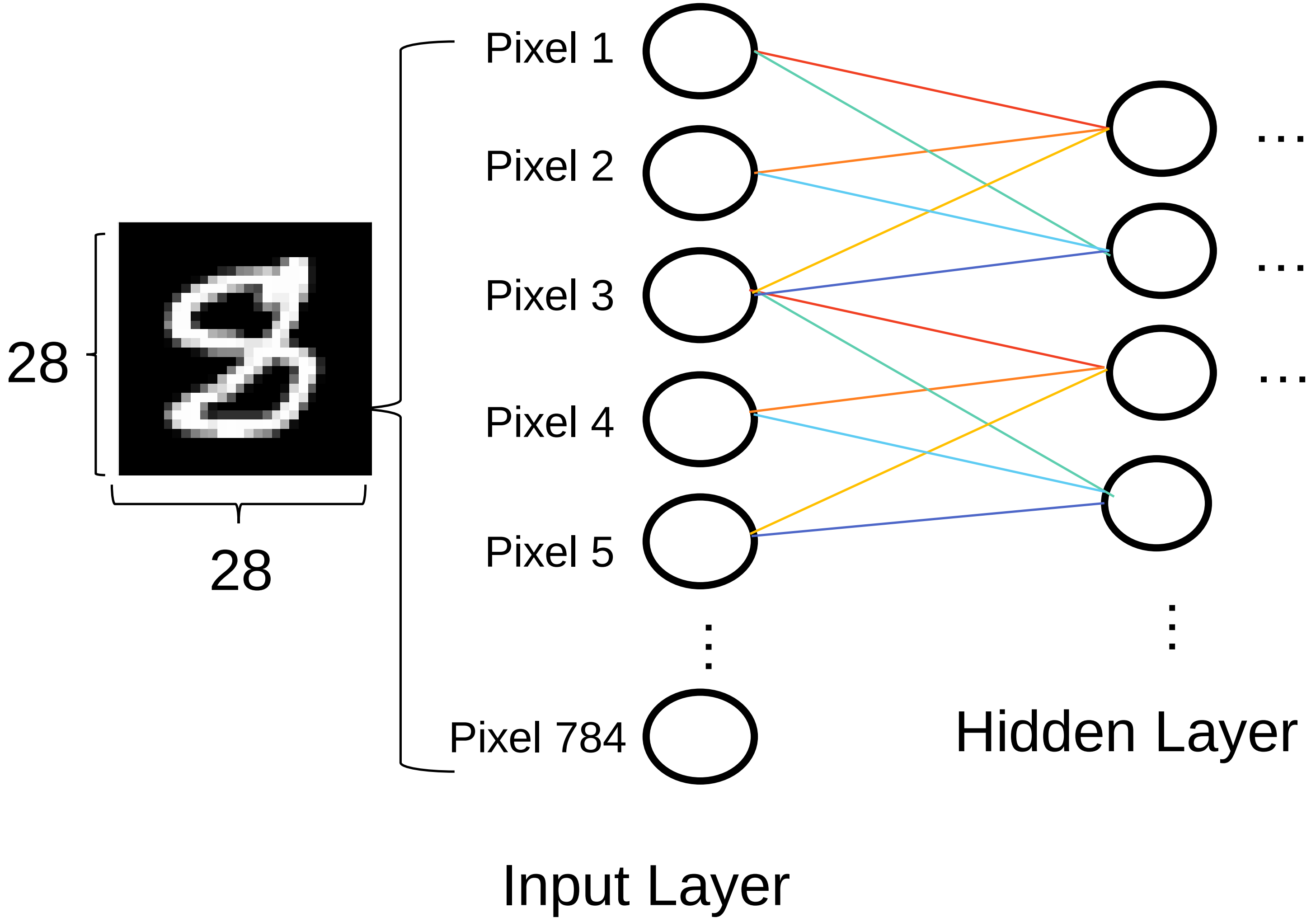


28
28
Pixel 1
Pixel 2
Pixel 3
Pixel 4
Pixel 5
Pixel 784
Input Layer
Hidden Layer
(512 neuron)

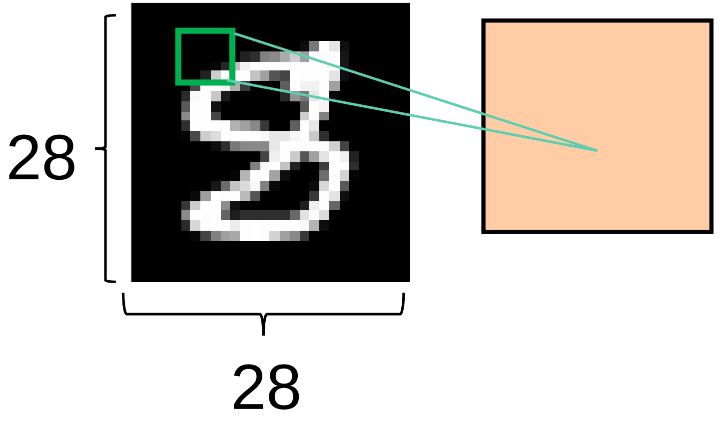


28
28
14
14

DNN & CNN



- Each neuron only connects to part of previous layer.
  → parameter sharing
  → Less parameters then fully connected layer.



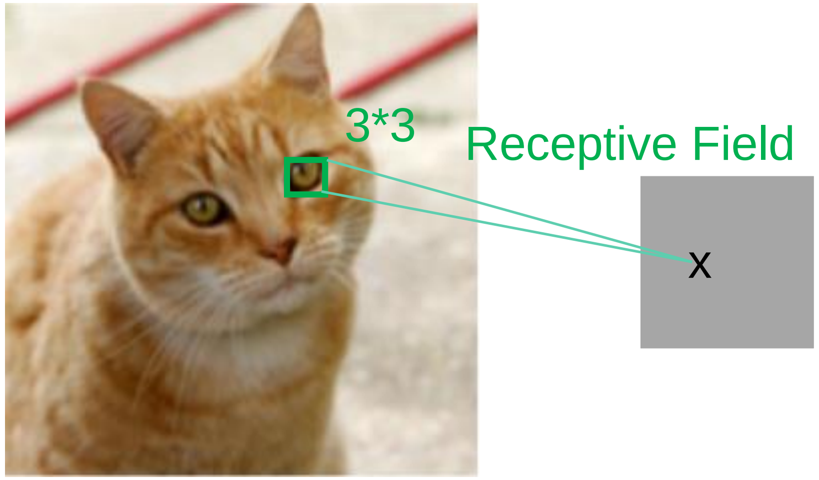
Due to parameter sharing

Idea #1 – Translation Invariance

Idea #2 – Locality

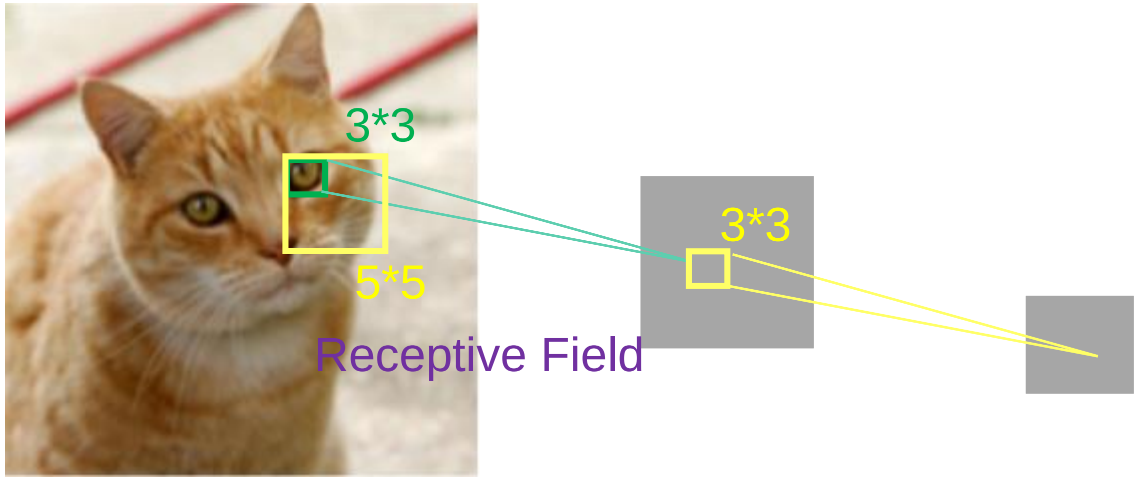
Receptive Field (filter size)

- For any element x of some layer, its receptive field refers to all the elements (from all the previous layers) that may affect the calculation of x during the forward propagation.

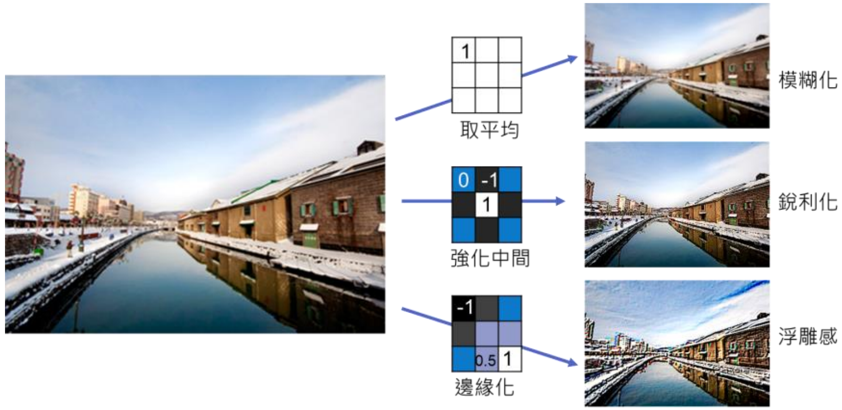


Receptive Field (filter size)

- For any element x of some layer, its receptive field refers to all the elements (from all the previous layers) that may affect the calculation of x during the forward propagation.



Object Edge Detection in Images



Feature map

Input X

10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0
10	10	10	10	0	0	0	0

*

Kernel W (Filter)

1	0	-1
1	0	-1
1	0	-1

=

Output Y

0	0	30	30	0	0
0	0	30	30	0	0
0	0	30	30	0	0
0	0	30	30	0	0
0	0	30	30	0	0
0	0	30	30	0	0

Edge

Padding

- Padding adds row/columns around input.

Input X

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

Kernel W (Filter)

0	1
2	3

=

Output Y

0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0

$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$

$(n_h + p_h) \times (n_w + p_w)$
$(3 + 2) \times (3 + 2)$
5×5

$p_h \times p_w$
2×2

$k_h \times k_w$
2×2

$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$
$(3 - 2 + 2 + 1) \times (3 - 2 + 2 + 1)$
4×4

Padding

- Padding adds row/columns around input.

Input X

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

Kernel W (Filter)

0	1
2	3

=

Output Y

0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0

$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$

$0 \times 0 + 0 \times 1 + 0 \times 2 + 1 \times 3 = 3$

$(n_h + p_h) \times (n_w + p_w)$

$(3 + 2) \times (3 + 2)$

5×5

$p_h \times p_w$

2×2

$k_h \times k_w$

2×2

$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$

$(3 - 2 + 2 + 1) \times (3 - 2 + 2 + 1)$

4×4

Padding for consistent input/output shape

- Padding p_h rows and p_w columns, output shape will be $(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$

- A common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$
  - Odd k_h : pad $\frac{p_h}{2}$ on both sides
    - e.g., $k_h = 3, \frac{p_h}{2}$=1(上下各一); $k_w = 3,\ \frac{p_w}{2}$=1(左右各一)
  - Even k_h: pad $\left\lceil\frac{p_h}{2}\right\rceil$ rows on the top, $\left\lfloor\frac{p_h}{2}\right\rfloor$ rows on the bottom.
    - e.g., $k_h = 2, p_h = 1,\ \left\lceil\frac{p_h}{2}\right\rceil$= $\left\lceil\frac{1}{2}\right\rceil$=1, $\left\lfloor\frac{p_h}{2}\right\rfloor$=$\left\lfloor\frac{1}{2}\right\rfloor$=0

Stride (How to move)

- Stride is the #rows/#columns per slide

Input X

Strides of 2 for width s_w

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

Kernel W (Filter)

0	1
2	3

=

Output Y

0	8

$0\times0+0\times1+1\times2+2\times3=8$

Stride

- Stride is the #rows/#columns per slide

Strides of 2 for width s_w

Strides of 3 for height s_h

Input X

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

Kernel W (Filter)

0	1
2	3

=

Output Y

0	8
6	

$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$

$\mathbf{0 \times 0 + 0 \times 1 + 0 \times 2 + 6 \times 0 = 3}$

Stride

- Stride is the #rows/#columns per slide

Strides of 2 for width s_w

Strides of 3 for height s_h

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{Input X}} * \underbrace{\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}}_{\text{Kernel W (Filter)}} = \underbrace{\begin{bmatrix} 0 & 8 \\ 6 & 8 \end{bmatrix}}_{\text{Output Y}}$$

$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$

$0 \times 0 + 0 \times 1 + 0 \times 2 + 6 \times 0 = 3$

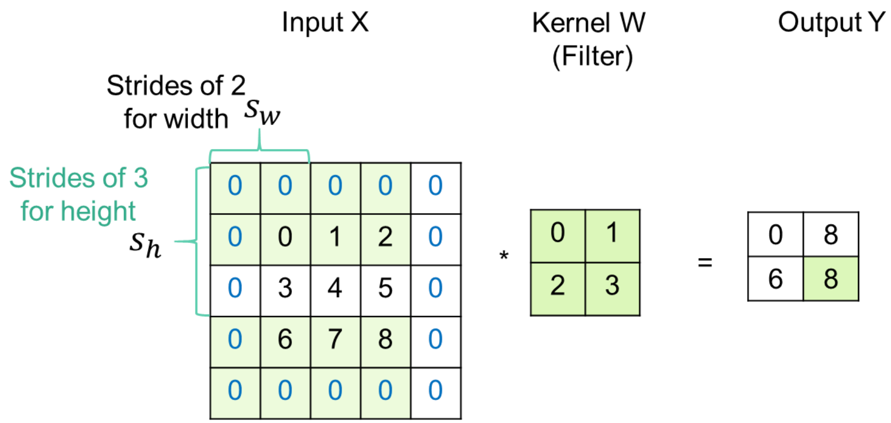
$7 \times 0 + 8 \times 1 + 0 \times 2 + 0 \times 3 = 8$

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

$$(3 - 2 + 2 + 3)/3 \times (3 - 2 + 2 + 2)/2$$

$$2 \times 2$$

Stride



- Given stride s_h for the height and stride s_w for the width, the output shape is

$$(n_h - k_h + p_h + s_h)/s_h \times (n_w - k_w + p_w + s_w)/s_w$$

- With $p_h = k_h - 1$ and $p_w = k_w - 1$ (超出範圍的不計算)

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- If input height/width are divisible by strides

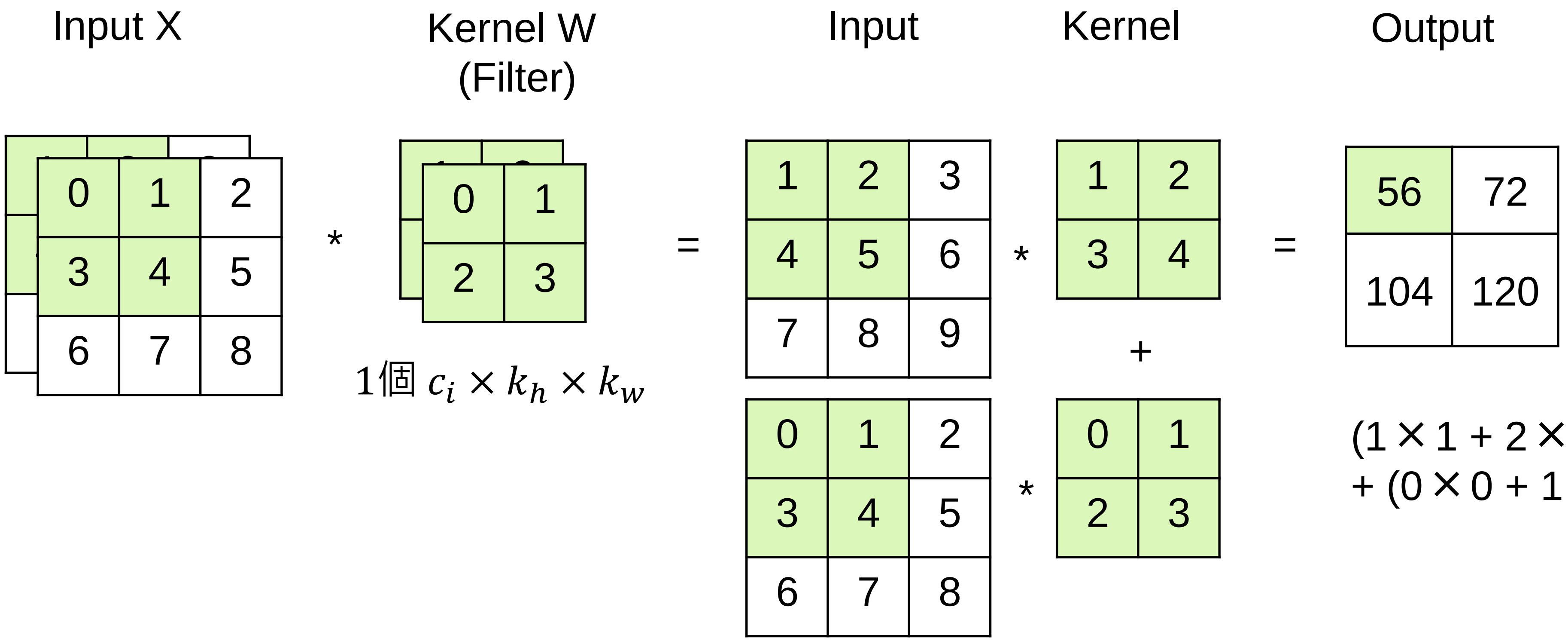
$$(n_h/s_h) \times (n_w/s_w)$$

Multiple Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

- Have a kernel for each channel, and then **sum** results over channels.



$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4)$
$+ (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$

Multiple Input Channels

- X: $c_i \times n_h \times n_w$ input
- W: $c_i \times k_h \times k_w$ kernel
- Y: $m_h \times m_w$ output

$$Y = \sum_{i=0}^{c_i} X_{i,:,:} \otimes W_{i,:,:}$$

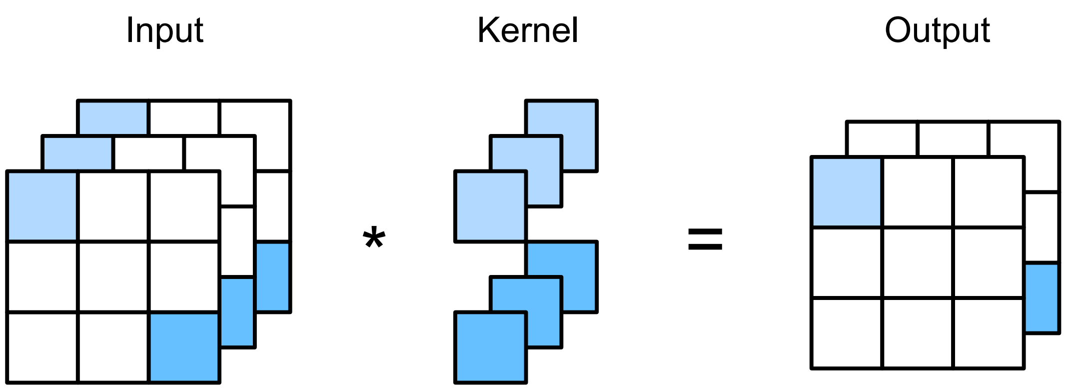
Input, Kernel, Output

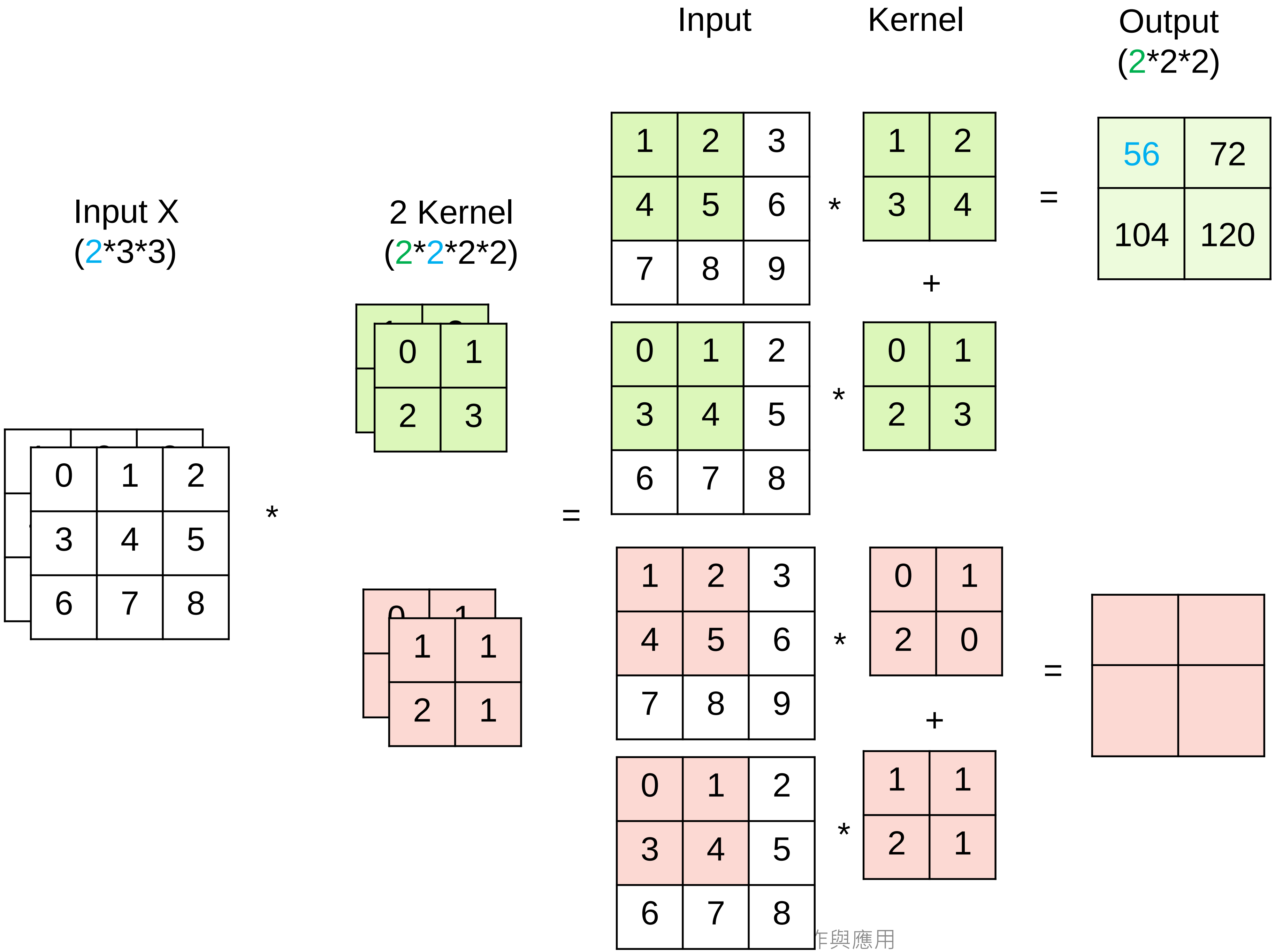
$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} * \begin{bmatrix}1&2\\3&4\end{bmatrix} + \begin{bmatrix}0&1&2\\3&4&5\\6&7&8\end{bmatrix} * \begin{bmatrix}0&1\\2&3\end{bmatrix} = \begin{bmatrix}56&72\\104&120\end{bmatrix}$$

$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$

Multiple Output Channels

- No matter how many inputs channels, so far we always get single output channel
- We can have multiple 3-D kernels, each one generates a output channel
- Input X: $c_i \times n_h \times n_w$ (3*3*3)
- Output Y: $c_o \times m_h \times m_w$ (2*3*3)
- Kernel W: $c_o \times c_i \times k_h \times k_w$ (2*1*1*1)




Input X
(2*3*3)
0 1 2
3 4 5
6 7 8
*
2 Kernel
(2*2*2*2)
0 1
2 3
1 1
2 1
=
Input
Kernel
Output
(2*2*2)
1 2 3
4 5 6
7 8 9
*
1 2
3 4
+
0 1 2
3 4 5
6 7 8
*
0 1
2 3
=
56 72
104 120
1 2 3
4 5 6
7 8 9
*
0 1
2 0
+
0 1 2
3 4 5
6 7 8
*
1 1
2 1
=

Pooling

- Convolution is sensitive to position

- We need some degree of invariance to translation
  - Lighting, object positions, scales, appearance vary among images

Pooling

- Max Pooling
  - Return the maximal value in the sliding windows
  - The strongest pattern signal in a window

Input

0	1	2
3	4	5
6	7	8

2 × 2 Max Pooling

=

Output

4	5
7	8

max(0,1,3,4) = 4

max(1,2,4,5) = 5

max(3,4,6,7) = 7

max(4,5,7,8) = 8

Pooling

- Average Pooling
  - Return the mean value in the sliding windows
  - The average signal in a window

Input

0	1	2
3	4	5
6	7	8

2 × 2 Avg Pooling

=

Output

2	3
5	6

avg(0,1,3,4) = 2

avg(1,2,4,5) = 3

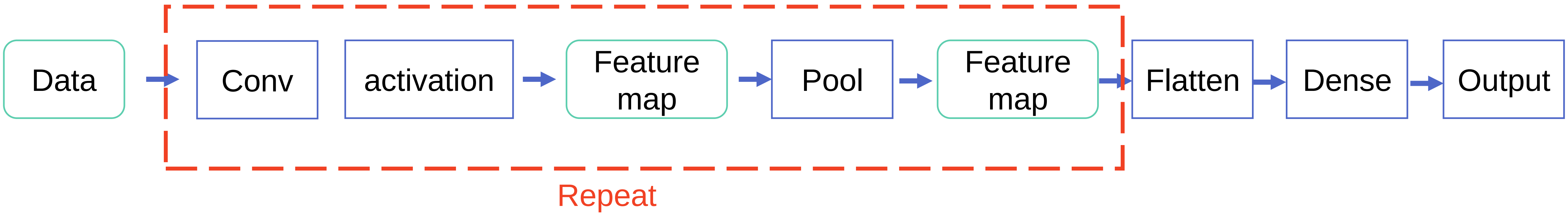
avg(3,4,6,7) = 5

avg(4,5,7,8) = 6

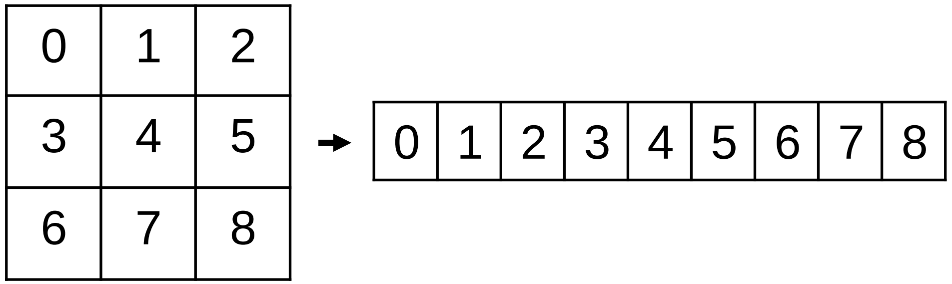
Pooling

- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel (每個channel獨立做pooling)
- # output channels = # input channels

CNN loop and flatten



Flatten



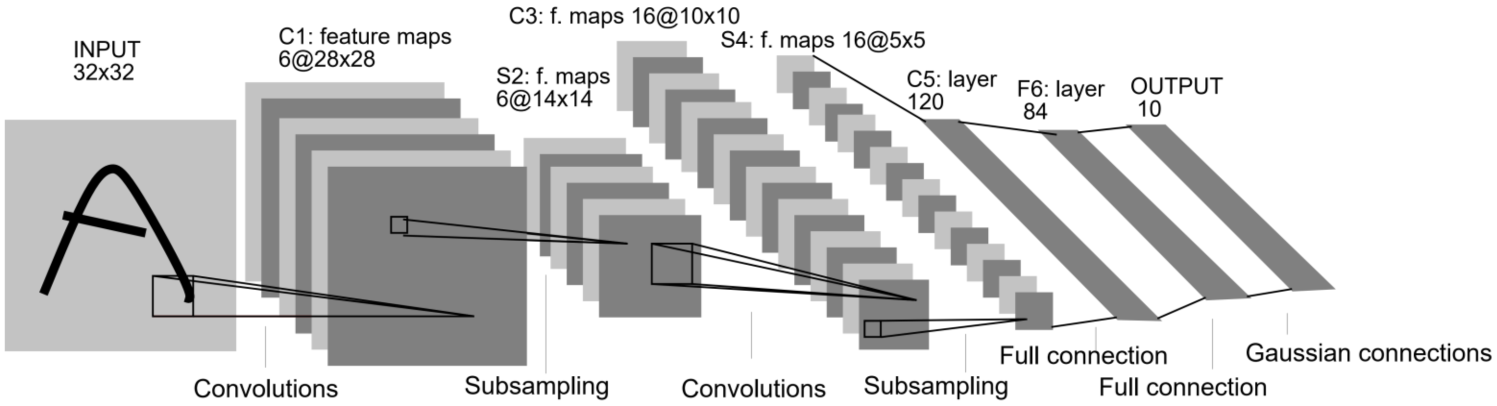
Summary

- Convolutional layer
  - Reduced model capacity compared to dense layer
  - Efficient at detecting spatial patterns
  - Control output shape via padding, strides (o_h & o_w) and (output) channels

- Max/Average Pooling layer
  - Provides some degree of invariance to translation

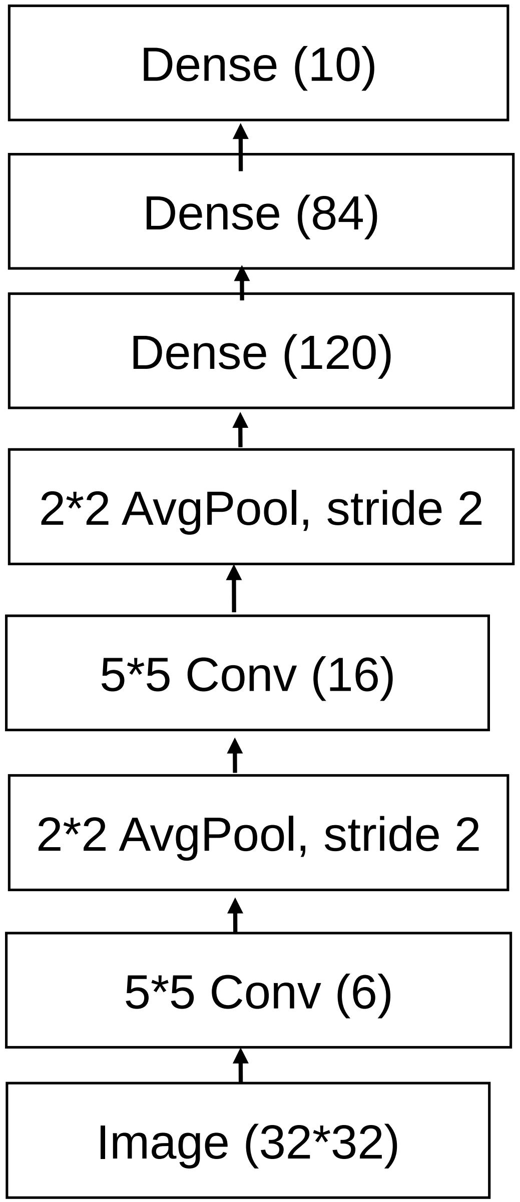
	MINST	ImageNet
		
Images	Gray image for hand-written digits	Color images with nature objects
Size	28 * 28	469 * 387
# examples (training)	60K	1.2M
# classes	10	1000

LeNet



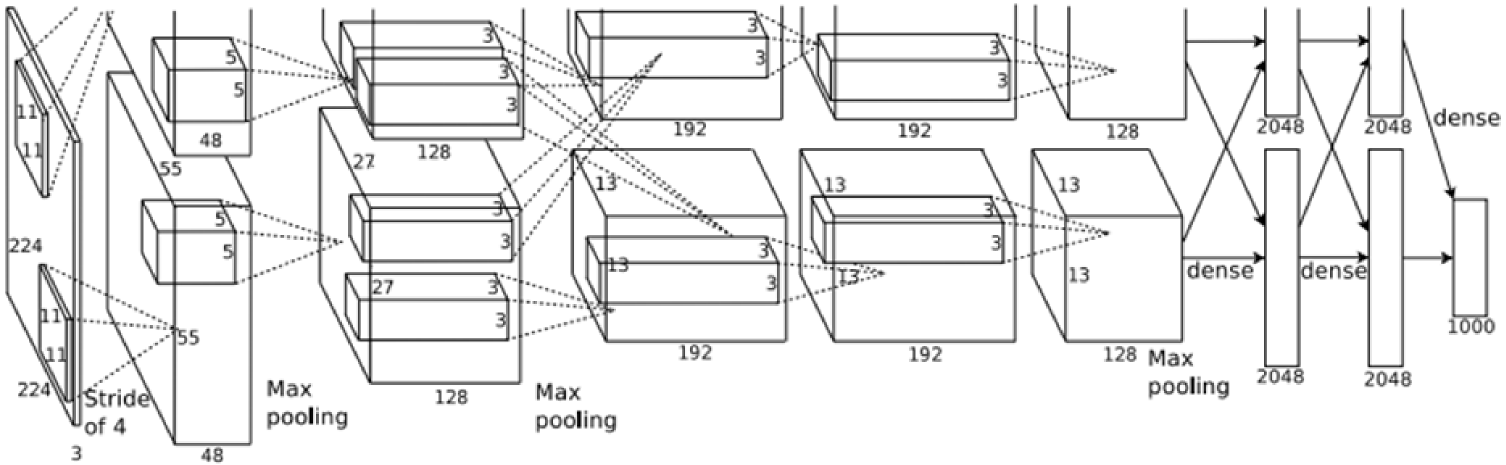
LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86, no. 11 (1998): 2278-2324.

Source: LeCun et al. (1998).



LeNet

AlexNet



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012): 1097-1105.

3*3 MaxPool, stride 2

↑

3*3 Conv (384), pad 1

↑

3*3 Conv (384), pad 1

↑

3*3 Conv (384), pad 1

↑

3*3 MaxPool, stride 2

↑

5*5 Conv (256), pad 2 上下左右各2

↑

3*3 MaxPool, stride 2

↑

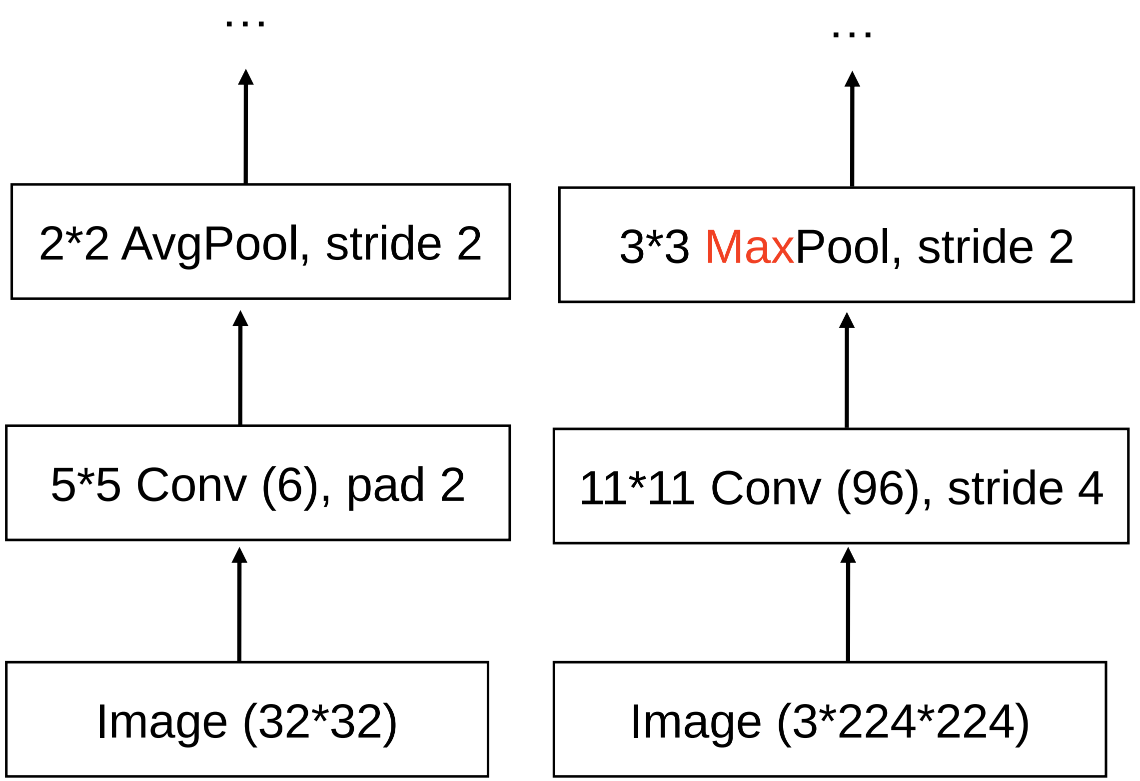
11*11 Conv (96), stride 4

↑

Image (3*224*224)

Source: Krizhevsky et al., 2012

LeNet vs. AlexNet

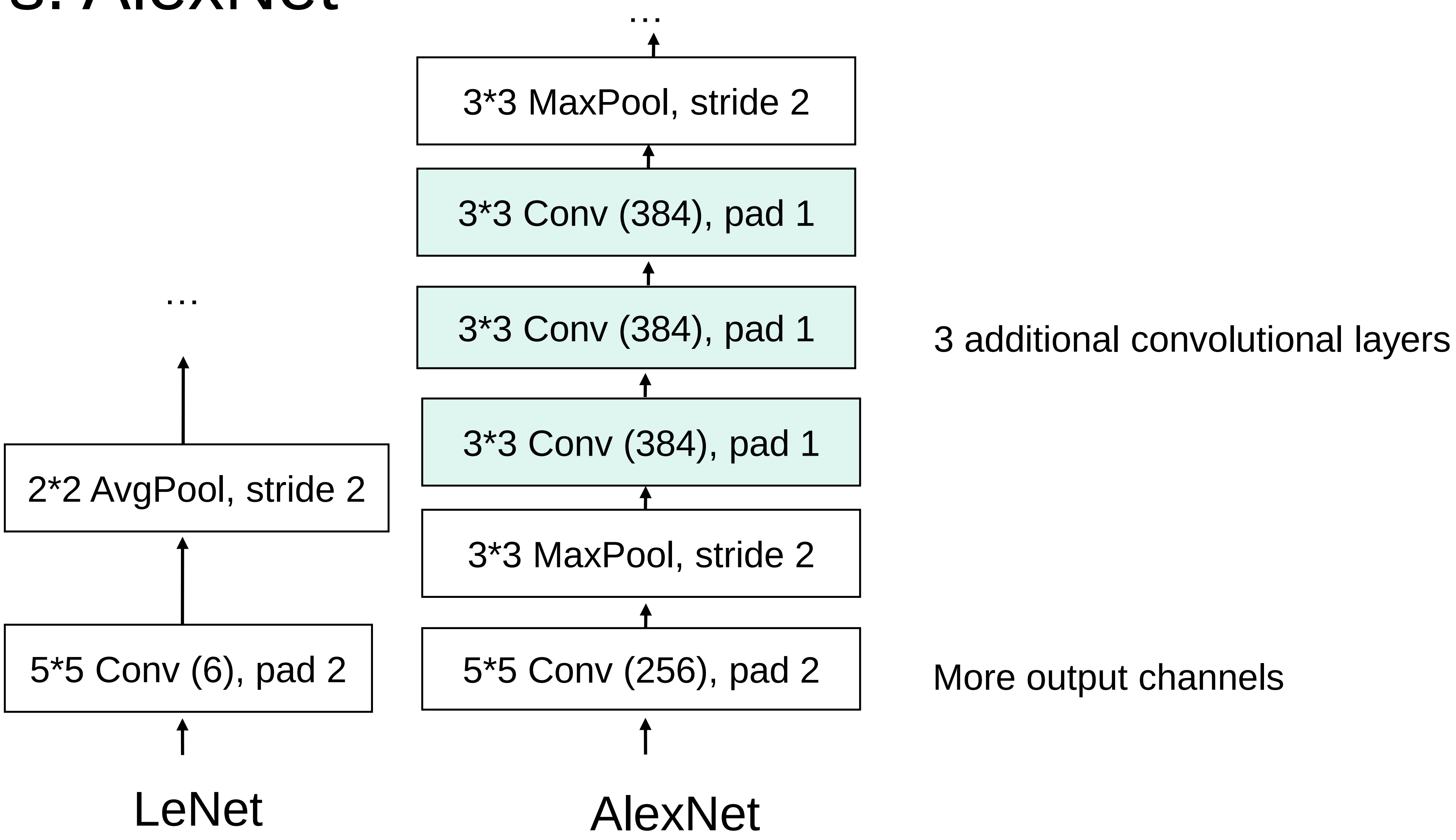


Larger pool size, change to max pooling

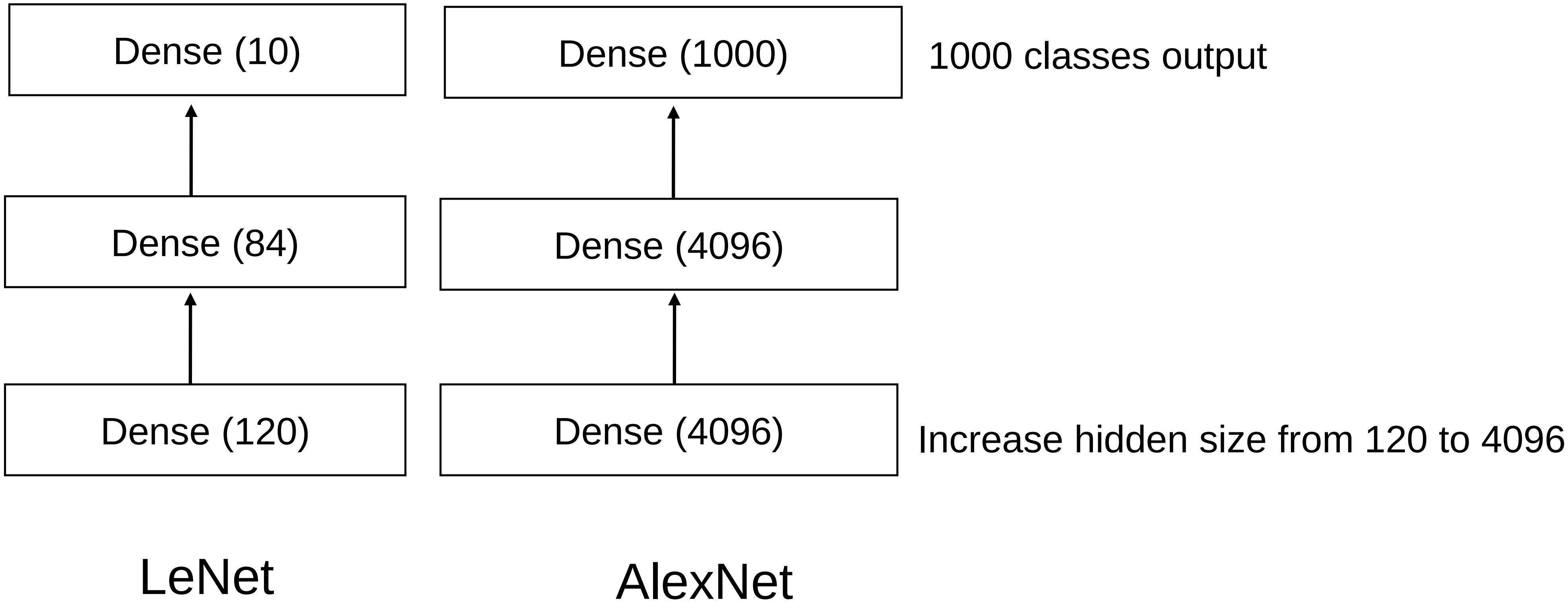
Larger kernel size, stride, because of the increased image size and more output channel

LeNet AlexNet

LeNet vs. AlexNet



LeNet vs. AlexNet



AlexNet

- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNext
- Key modifications
  - Change activation from sigmoid to ReLU (no more vanish gradient)
  - Add a dropout layer after two hidden dense layers (better robustness / regularization)
  - MaxPooling

1D, 2D and 3D Convolution

- 1-D

$$y_{i,j} = \sum_{a=1}^{h} v_a x_{i+a}$$

- Text
- Time series
- Voice

- 2-D

$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} v_{a,b} x_{i+a,j+b}$$

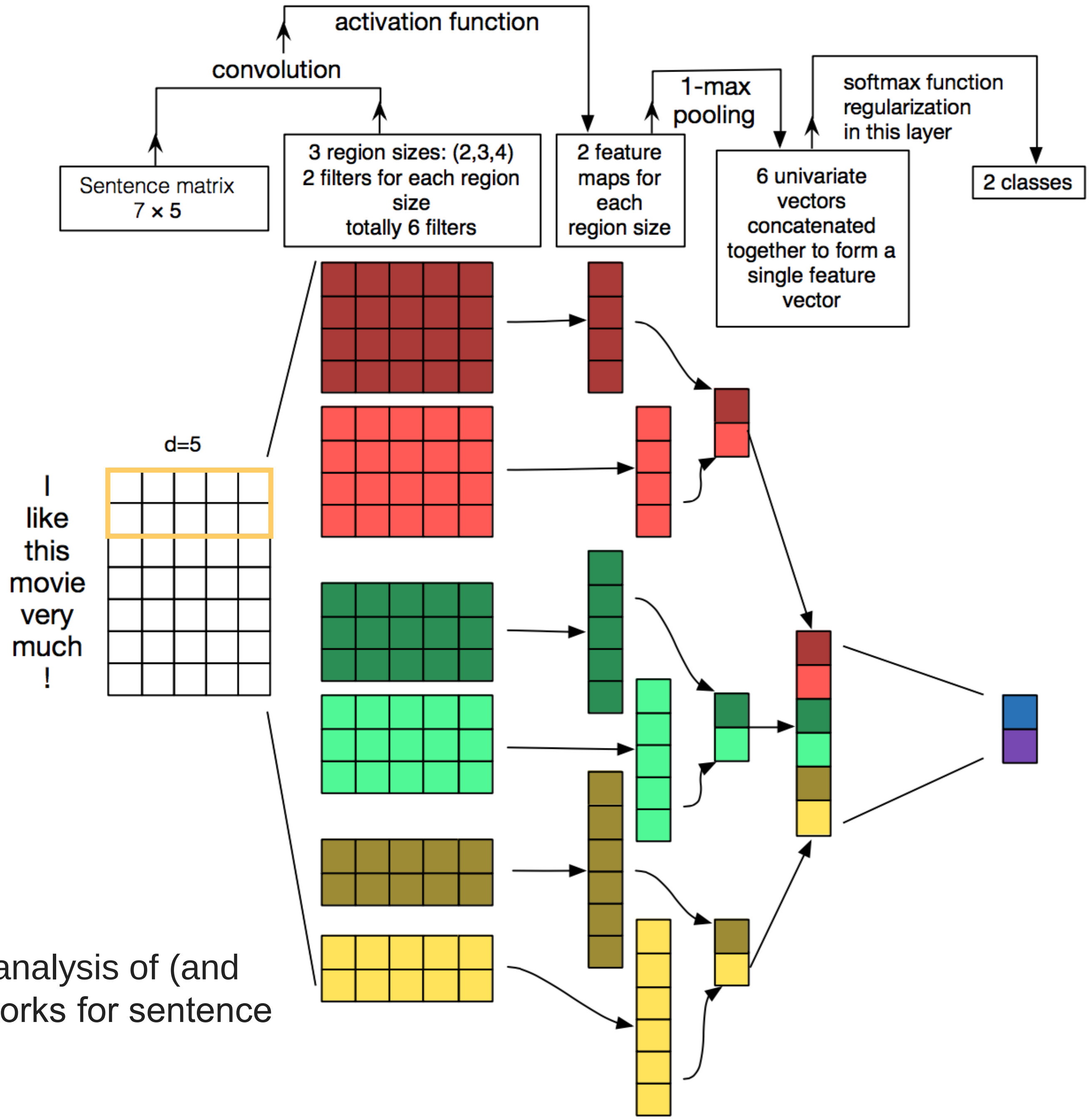
- Images
- Voice

- 3-D

$$y_{i,j,k} = \sum_{a=1}^{h} \sum_{b=1}^{w} \sum_{c=1}^{d} v_{a,b,c} x_{i+a,j+b,k+c}$$

- Video
- Medical images

Text Classification



Zhang, Ye, and Byron C. Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification." IJCNLP2017.

Problems (Internal Covariate Shift)

- Changes in model parameters during learning change the distributions of the outputs of each hidden layer.
  - Loss occurs at last layer
  - Data is at bottom layer
  - Bottom layers (weights) change – everything changes

- This means that later layers need to adapt to these (often noisy) changes during training.
- Can we avoid changing last layers while learning first layer?

Recall input standardization

- Numerical inputs
  - Standardize: rescaling features to **zero mean** and **unit variance**

$$x = \frac{x - \mu}{\sigma}$$

    - First, it proves convenient for optimization.
    - Second, because we do not know *a priori* which features will be relevant, we do not want to penalize coefficients assigned to one feature more than on any other.

Batch Normalization

$$x = \frac{x - \mu}{\sigma}$$

Input: Values of x over a mini-batch: $\mathrm{B} = \{x_1, \ldots x_m\}$

Learnable parameters: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

// Fix mean and variance

$$\mu_B = \frac{1}{|B|} \sum_{i \in B} x_i, \ \sigma_B^2 = \frac{1}{|B|} \sum_{i \in B} (x_i - \mu_B)^2$$

// normalize

$$\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

// scale and shift

$$y_i = \gamma \, \widehat{x_i} + \beta$$

Batch normalization during testing

- Solution 1:
  - Mean and variance were calculated using the whole training data, rather than mini-batch.

- Solution 2:
  - The moving average of mean and variance of the batches were computed during training
  - running_mean = momentum * running_mean + (1 - momentum) * sample_mean
  - running_var = momentum * running_var + (1 - momentum) * sample_var

e.g. momentum = 0.1

Batch Normalization 好處

- **解決 Internal Covariate Shift 的問題**
- 有**正則化**的效果 (可以不使用Dropout)
- **減緩梯度消失**
- **加速收斂**

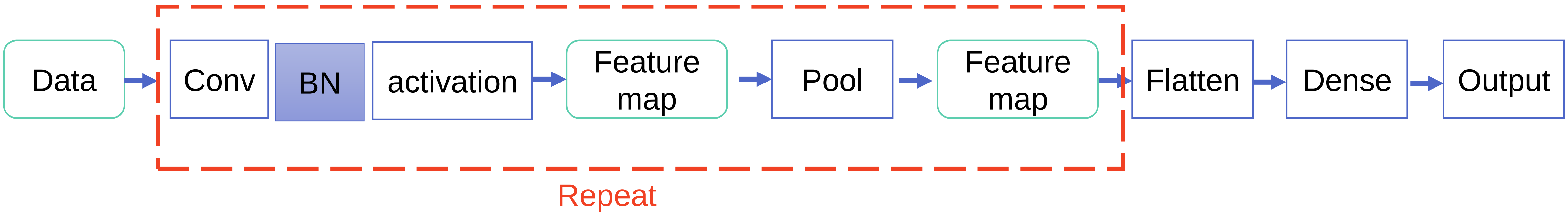
Batch Normalization

- Fully connected layer (before activation function):

$$\boldsymbol{h} = \phi(\mathrm{B}N(\boldsymbol{Wx} + \boldsymbol{b}))$$

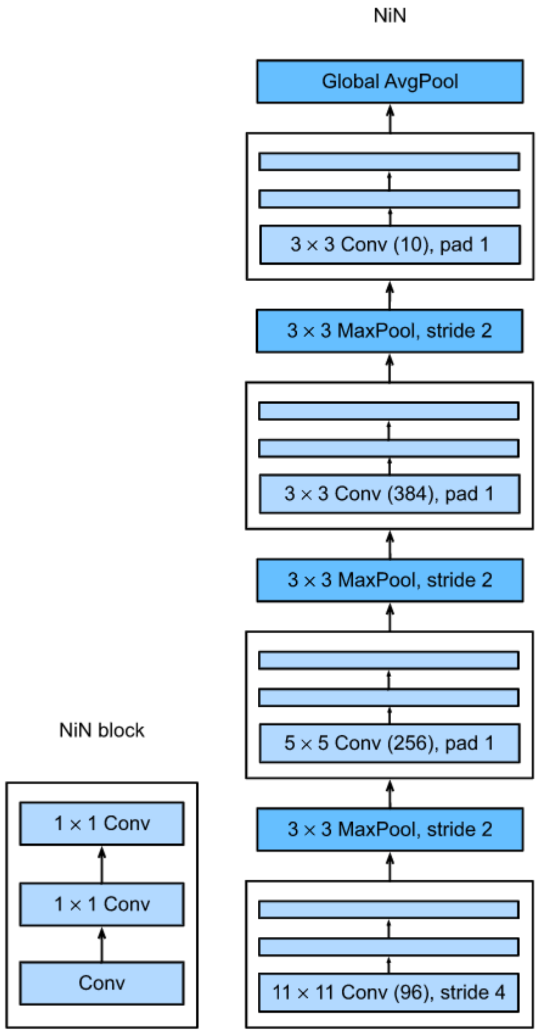
- CNN:
  - We can apply batch normalization after the convolution and before the nonlinear activation function.
  - When the convolution has multiple output channels, one normalization per channel, each channel has its own scale and shift parameters.
- Before: 速度快，因為BN可以merge到weight & bias
- After: 效果佳，真的讓output distribution被正規化

Batch Normalization in CNN



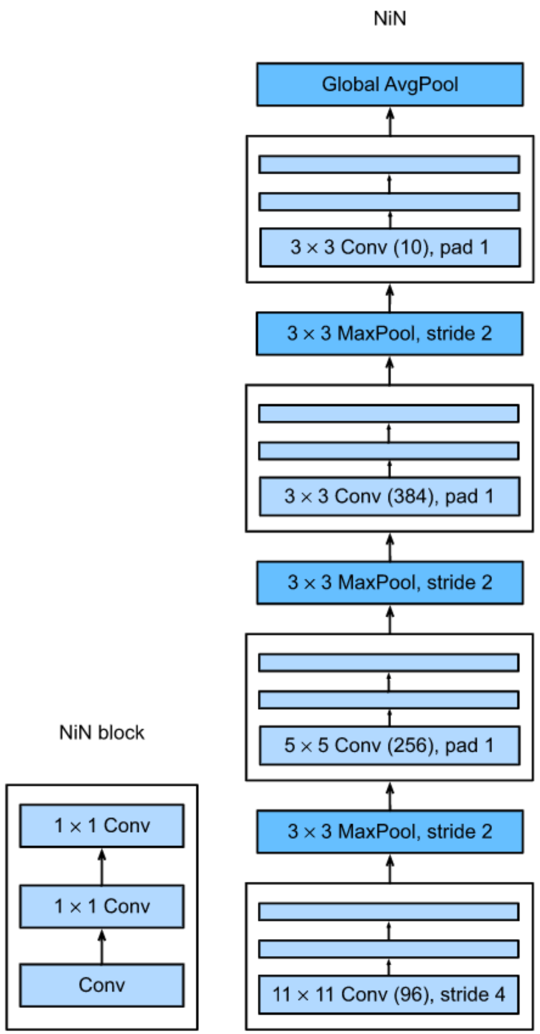
- Assume that our minibatches contains *m* examples,
  - The output of convolution has height *p* and width *q*
  - *m*p*q* elements per channel are normalized.

Network in Network (NiN)



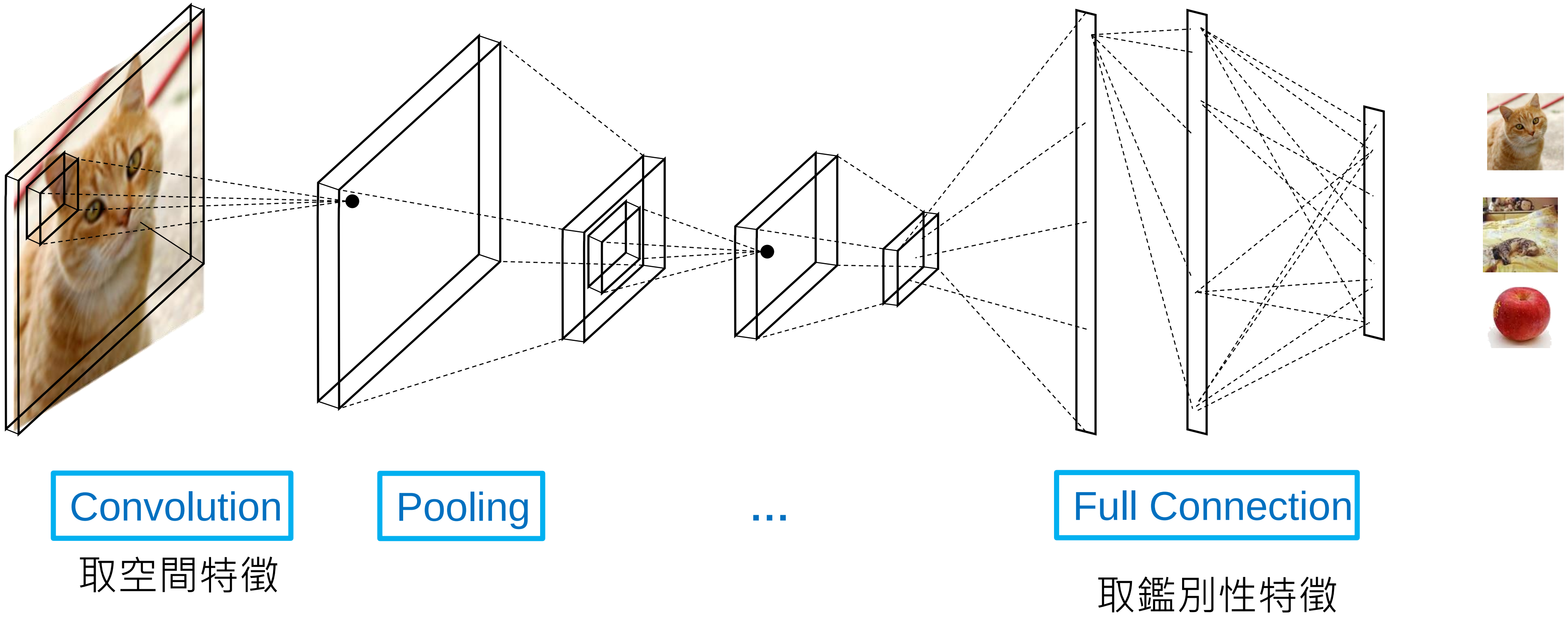
- The *network in network* (*NiN*) blocks (Lin *et al.*, 2013):
  - (i) use 1×1 convolutions to add local nonlinearities across the channel activations
  - (ii) use global average pooling to integrate across all locations in the last representation layer.
    - Note that global average pooling would not be effective, were it not for the added nonlinearities.

Network in Network (NiN)

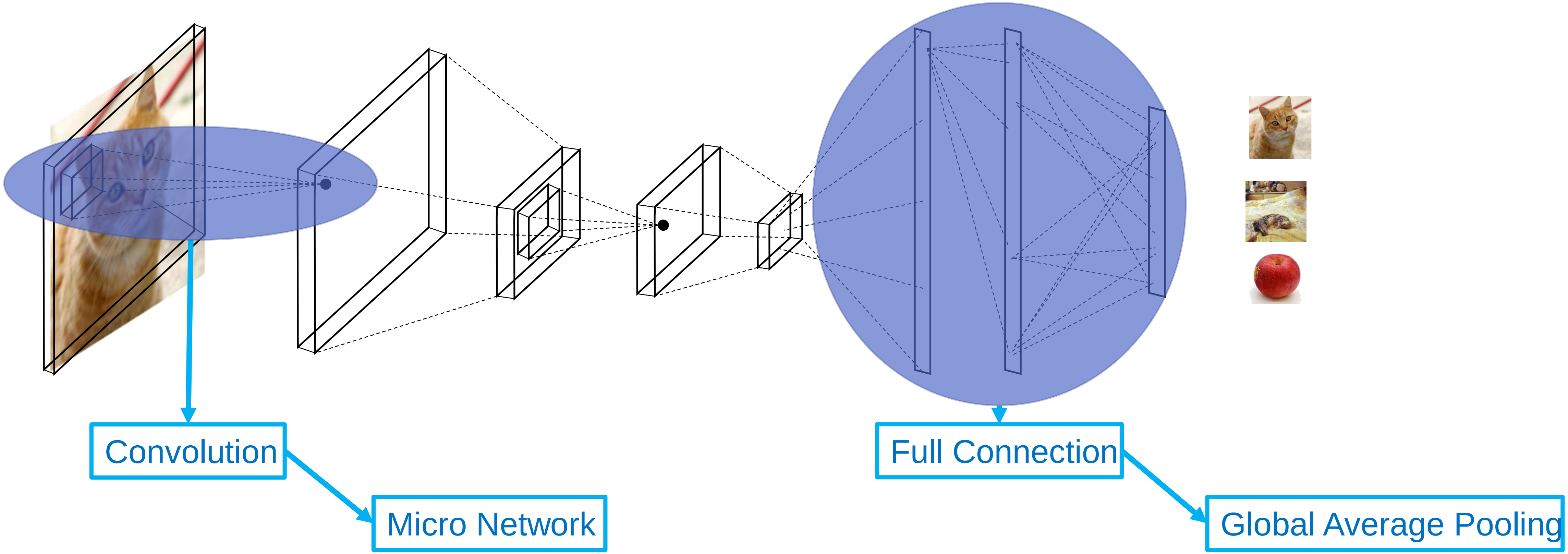


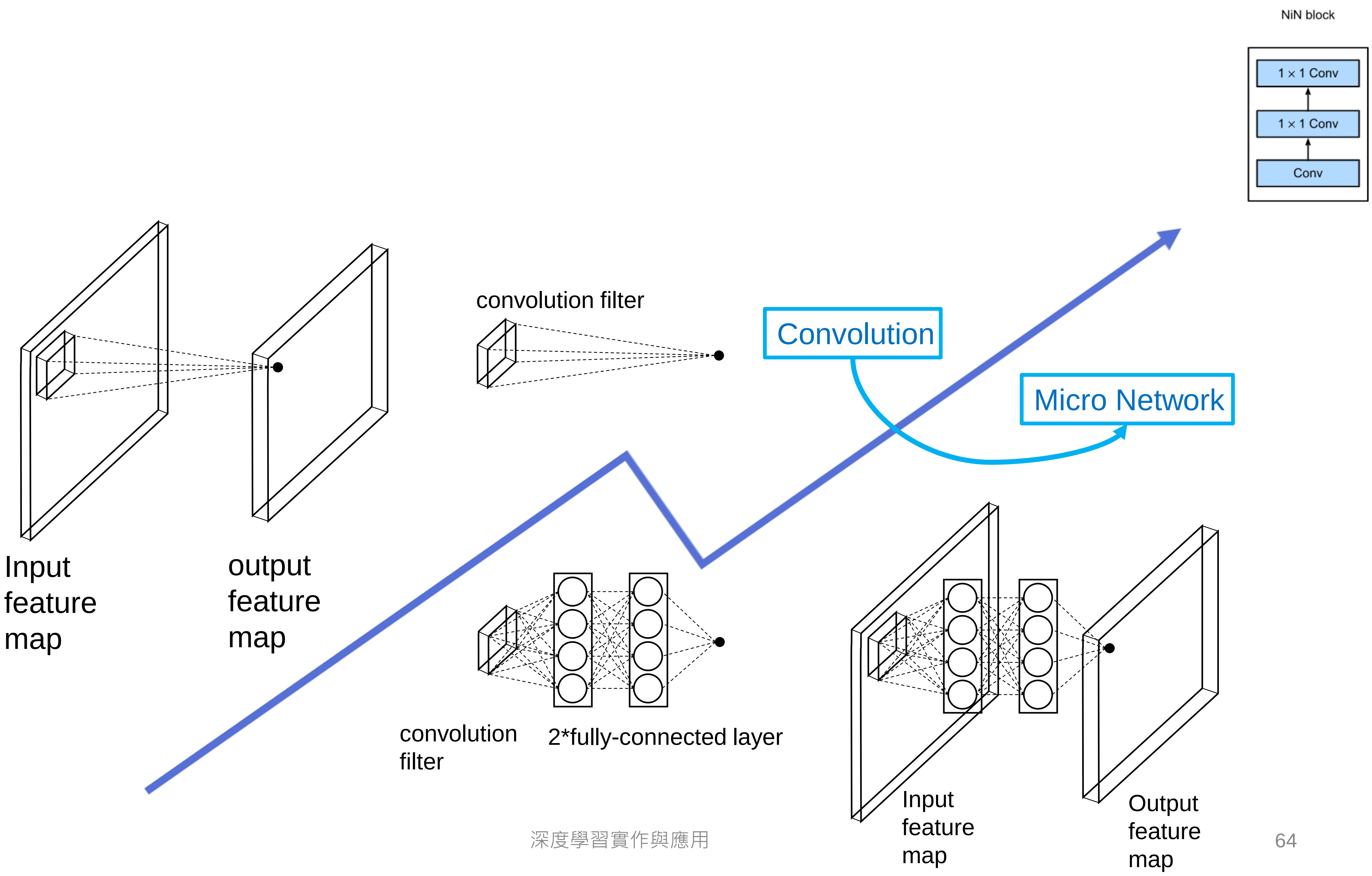
- NiN block:
  - The resulting 1×1 convolution can be thought as a fully connected layer acting independently on each pixel location.

Convolutional Neural Network

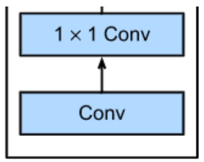


Network In Network

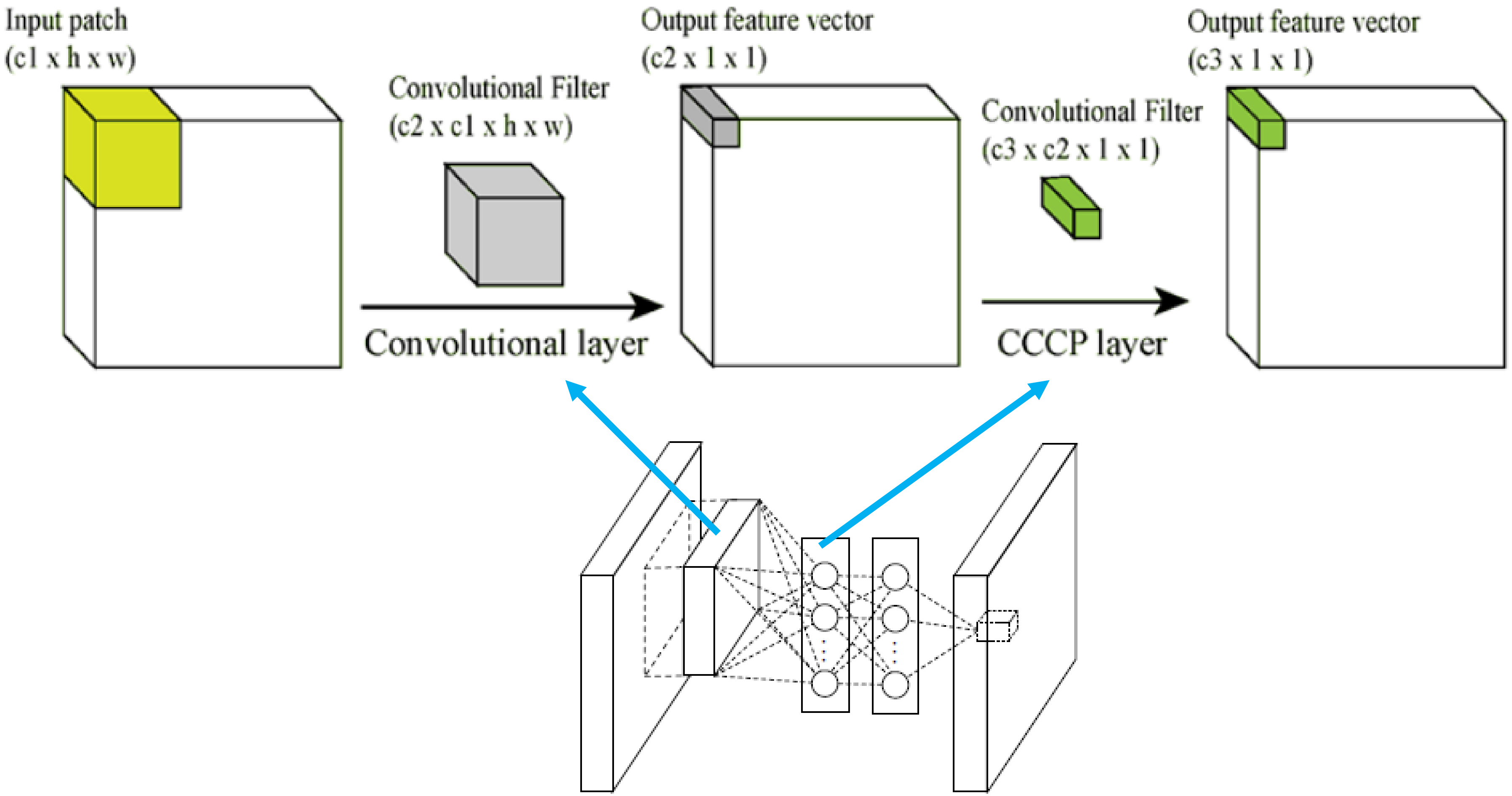



NiN block
1 × 1 Conv
1 × 1 Conv
Conv
convolution filter
Convolution
Micro Network
Input
feature
map
output
feature
map
convolution
filter
2*fully-connected layer
Input
feature
map
Output
feature
map

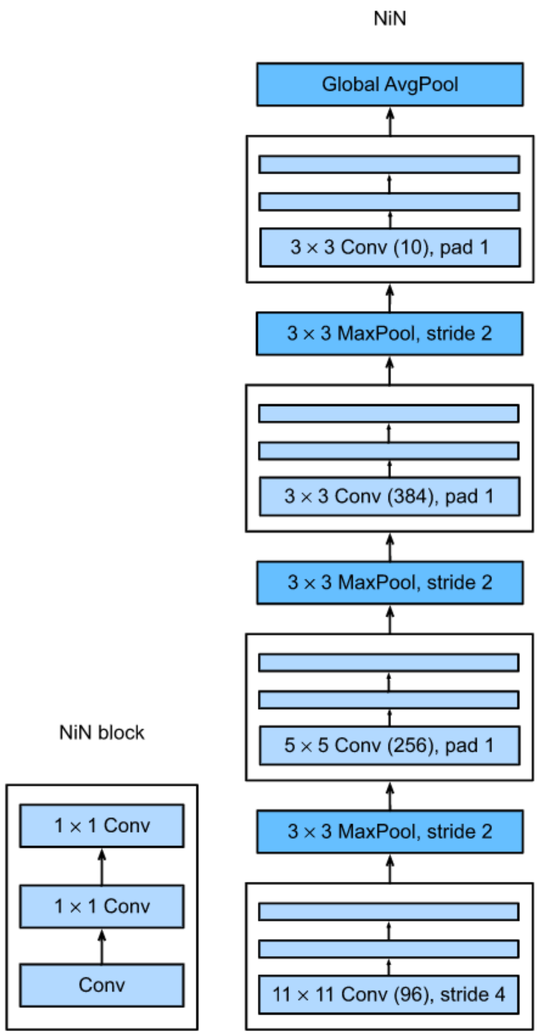
Network In Network



- Cascade Cross Channel Parametric Pooling = 1 x 1 Convolution

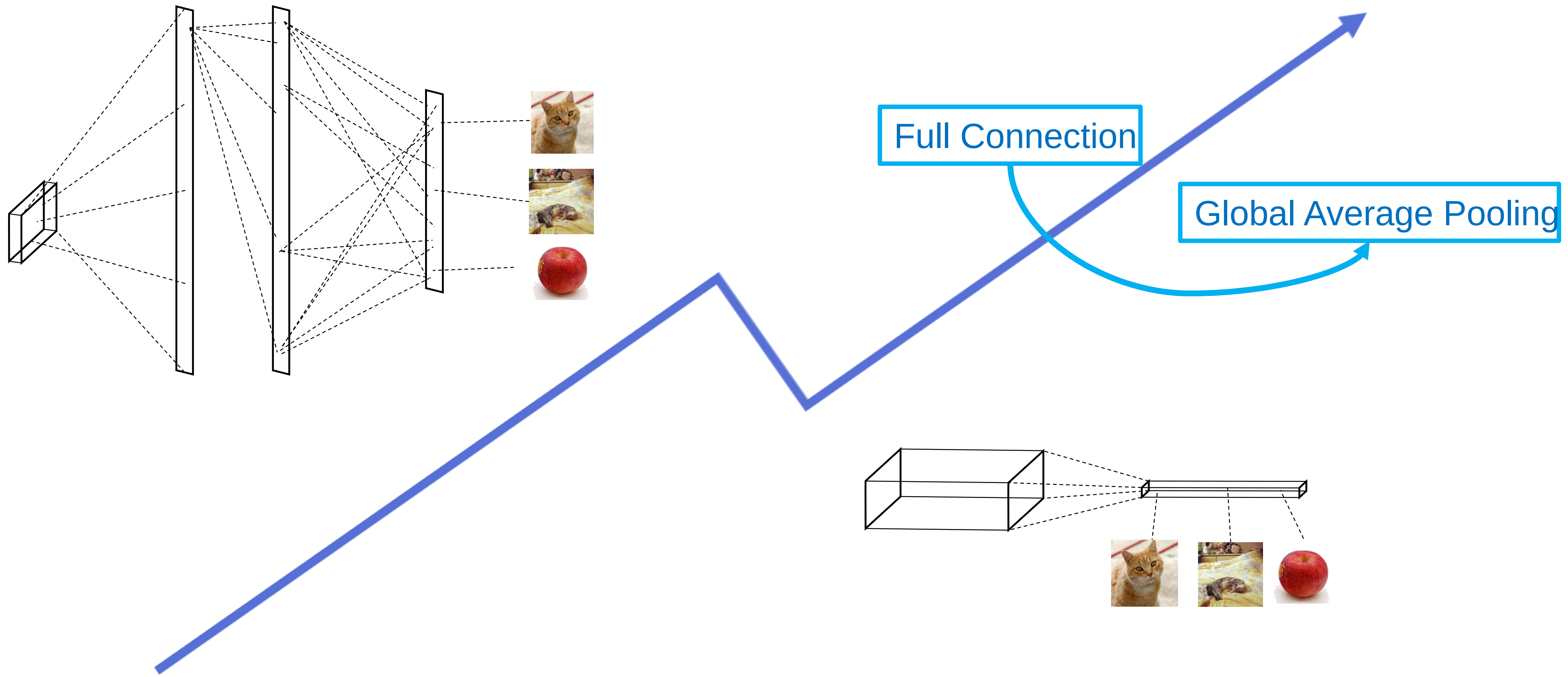


Network in Network (NiN)

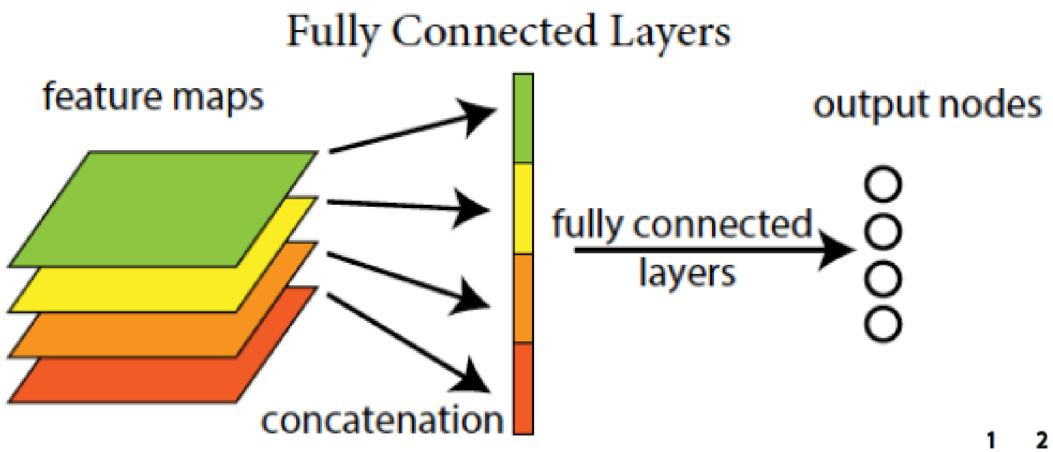


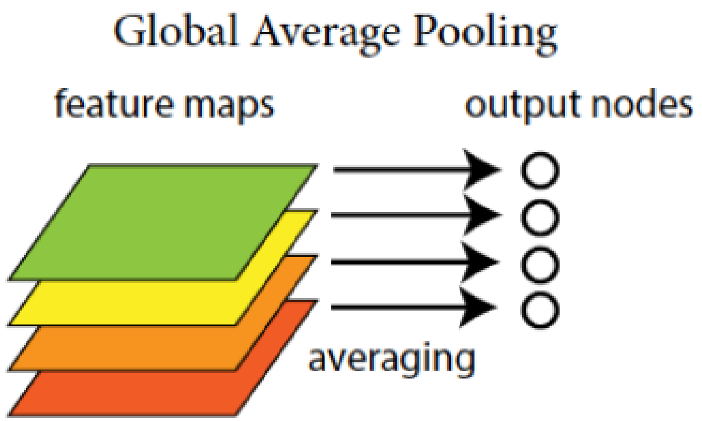
- Global AvgPool:
  - NiN avoids fully connected layers altogether.
  - Instead, NiN uses a NiN block with a number of output channels equal to the number of label classes, followed by a *global* average pooling layer, yielding a vector of logits.
  - This design significantly reduces the number of required model parameters, albeit at the expense of a potential increase in training time.

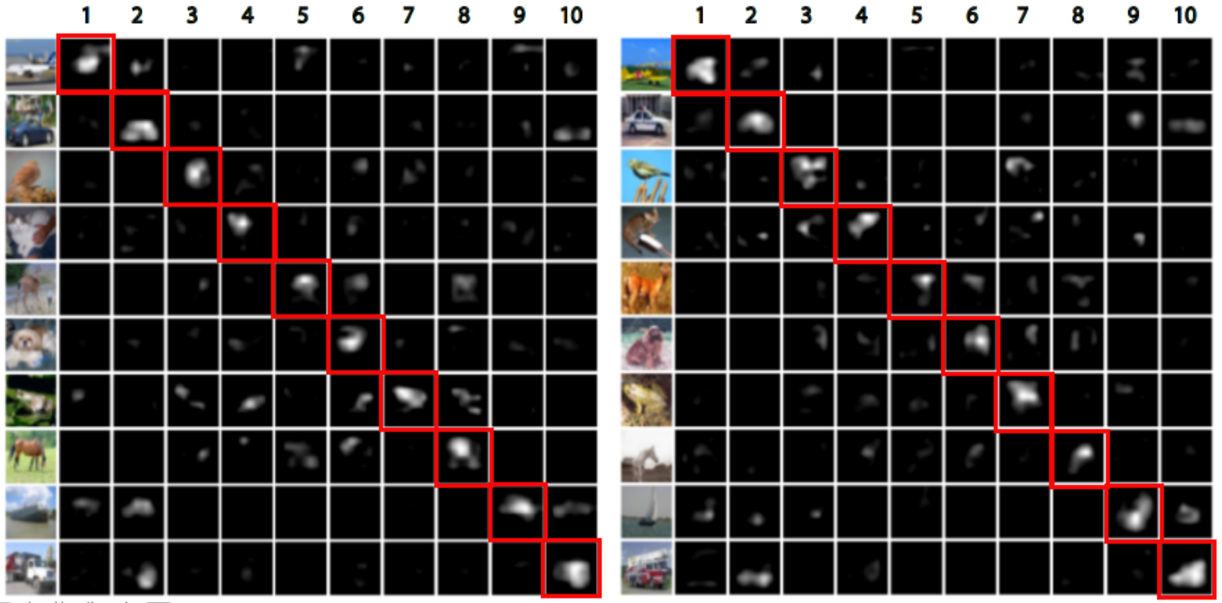
Global Average Pooling



Network In Network

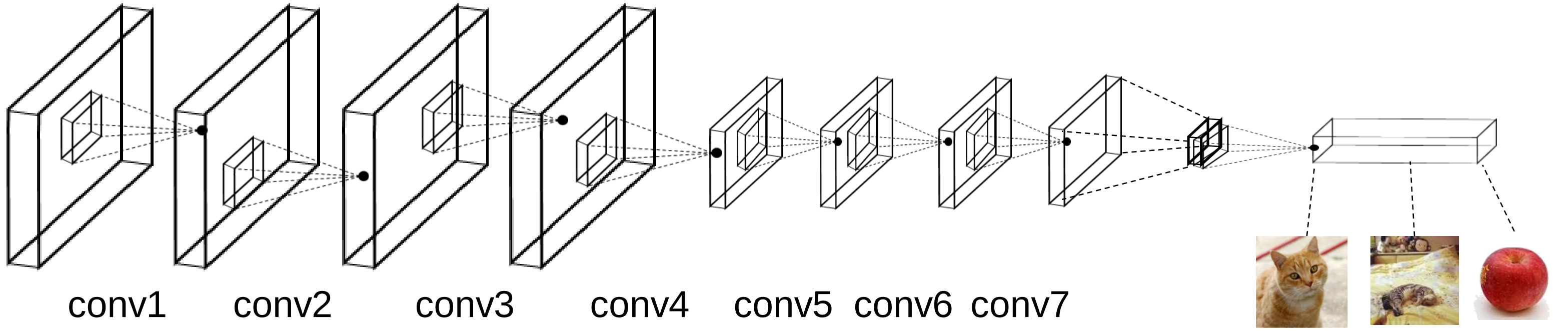




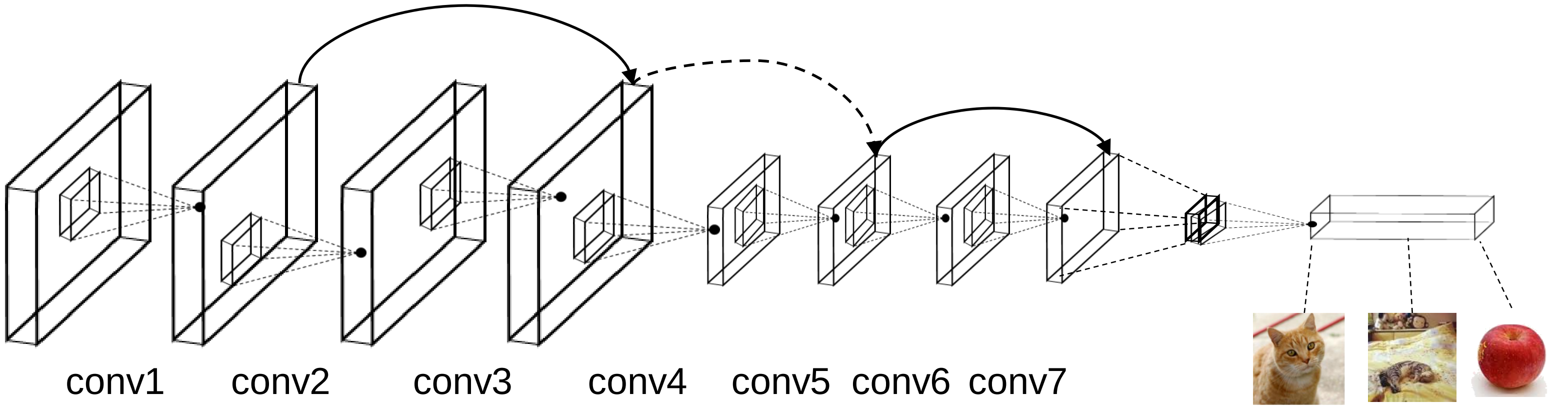


What are Residual Networks

- Plain Networks: stacking of convolutional layers. (LeNet, AlexNet, VGG, …)

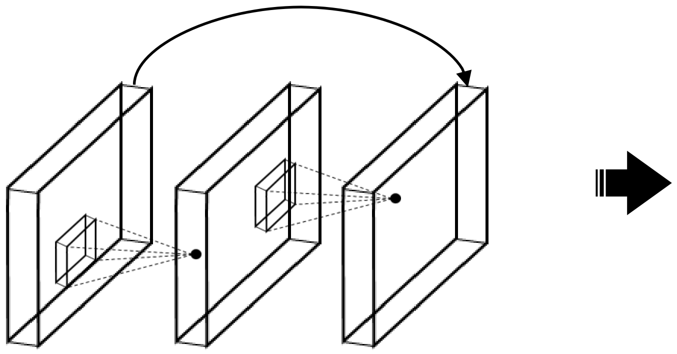


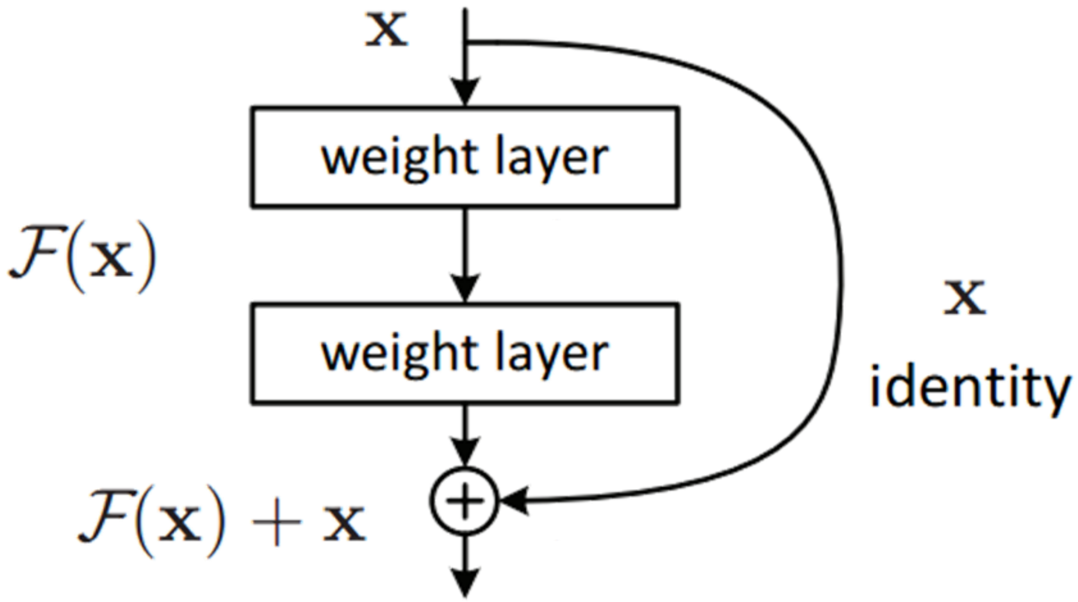
- Residual Networks: shortcut connections between feature maps.



Residual Block of Residual Networks

- Residual Block





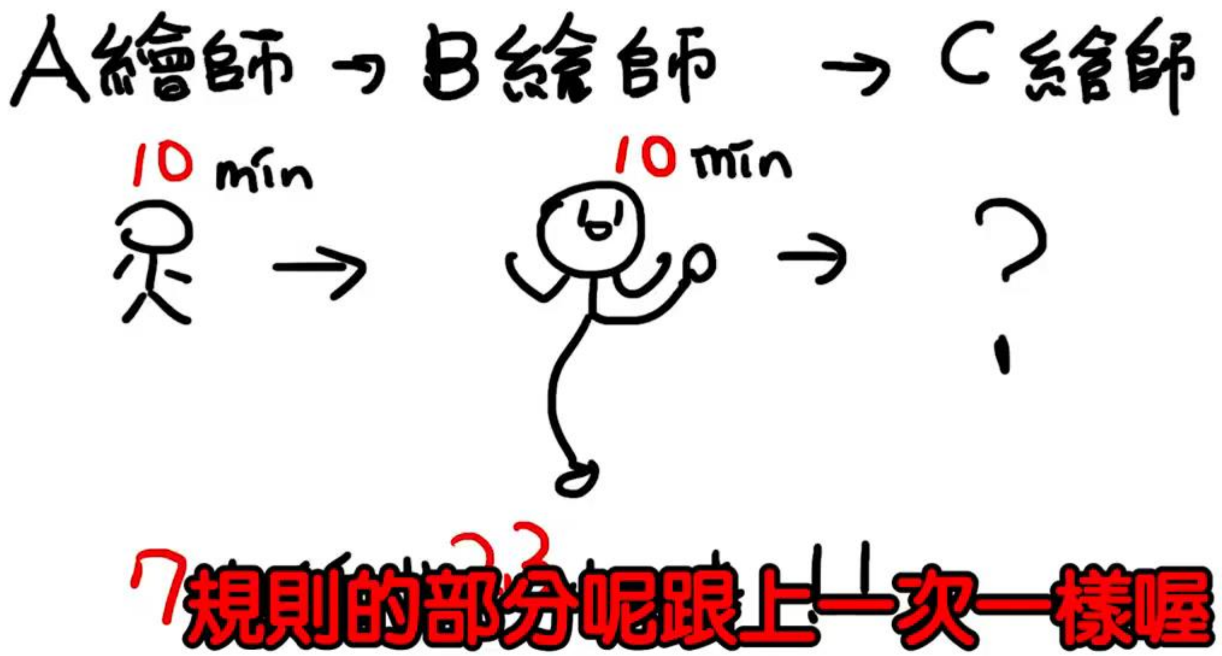
Why Residual Networks work (3/5)

- Plain Networks are just like…

Why Residual Networks work (4/5)

- Residual Networks know what following layers know.

https://www.youtube.com/watch?v=mIHdYIwPZFA



Why Residual Networks work (5/5)

- Plain Networks vs Residual Networks

Plain Networks

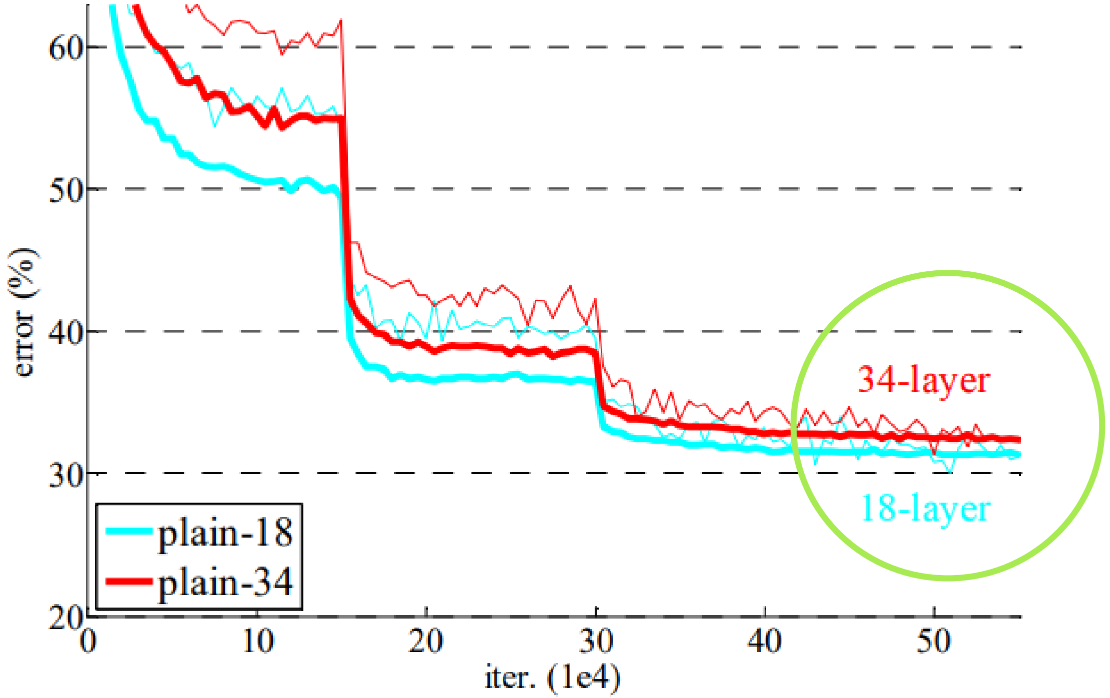
Each layer need to learn all things
by information from next layer

Residual Networks

Each layer only need to learn things
that other layers not yet learned

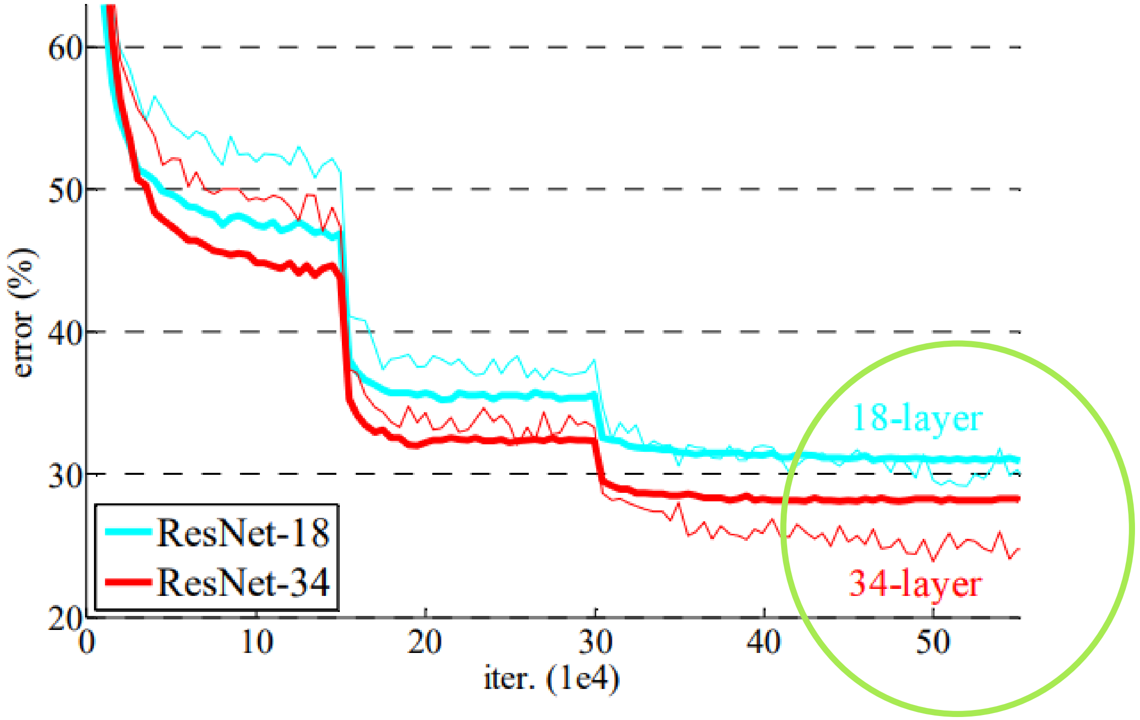
Why Residual Networks are important

- Residual networks make **super deep networks learn well**.



Plain Networks

#layers ↑, Accuracy ↓

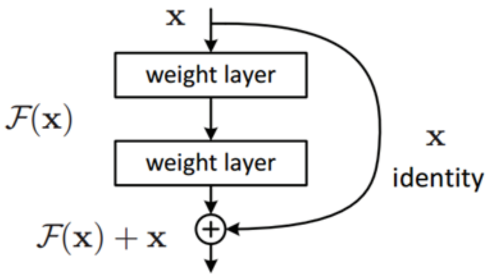


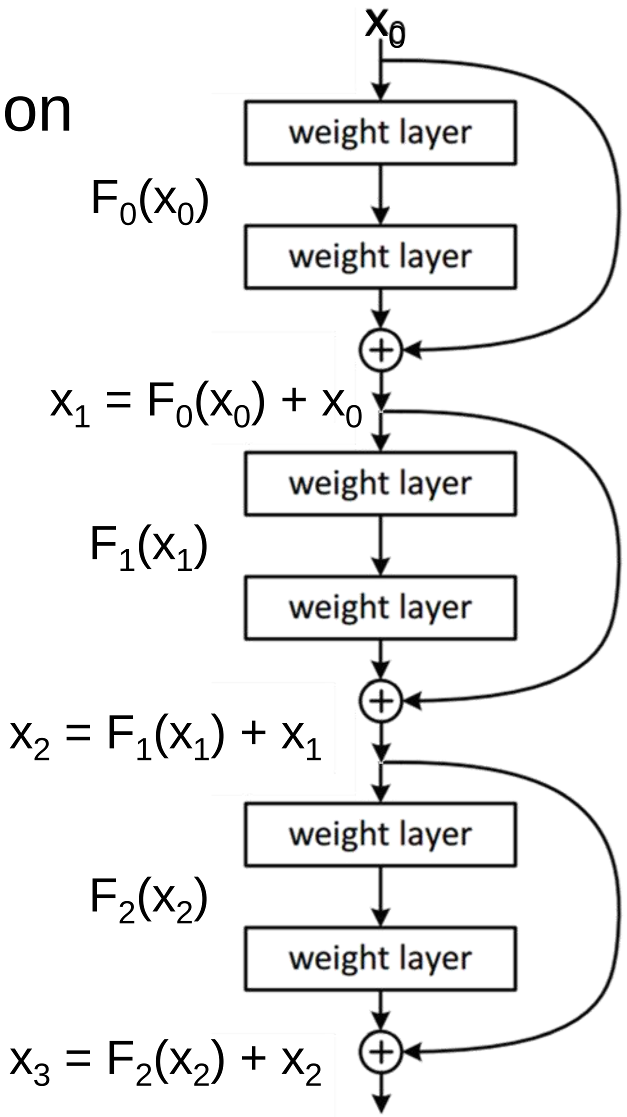
Residual Networks

#layers ↑, Accuracy ↑

Why Residual Networks work (1/5)

- Learning residual information

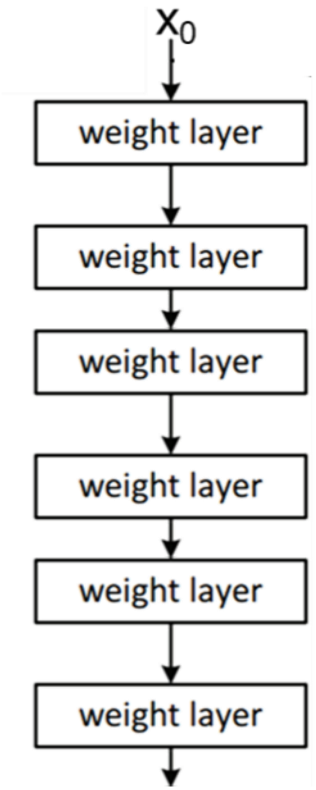




$$H(x_0) = x_0+F_0(x_0)+F_1(x_1)+F_2(x_2)$$

Why Residual Networks work (2/5)

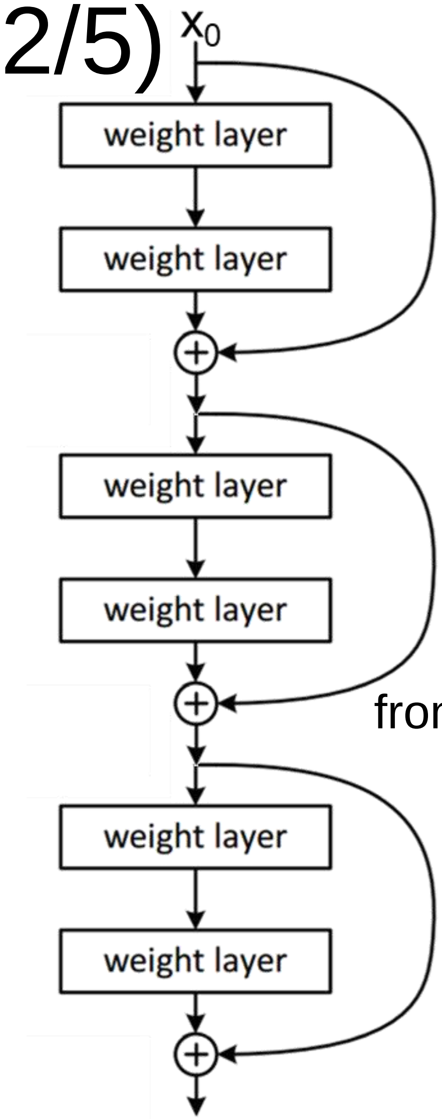
- Plain Networks vs Residual Networks



Can only learn information from next layer

Plain Networks

$H(x_0) = F_2(F_1(F_0(x_0)))$



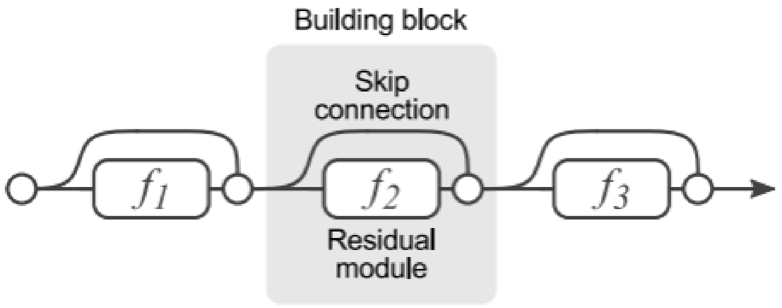
Directly learn information from **label** and following layers

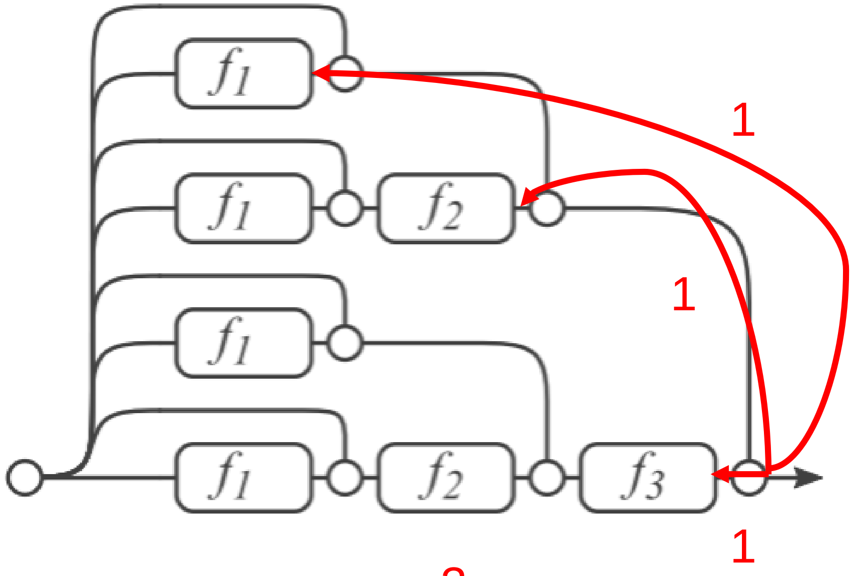
Residual Networks

$H(x_0) = x_0+F_0(x_0)+F_1(x_1)+F_2(x_2)$

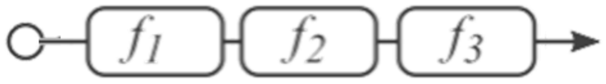
Properties of Residual Networks

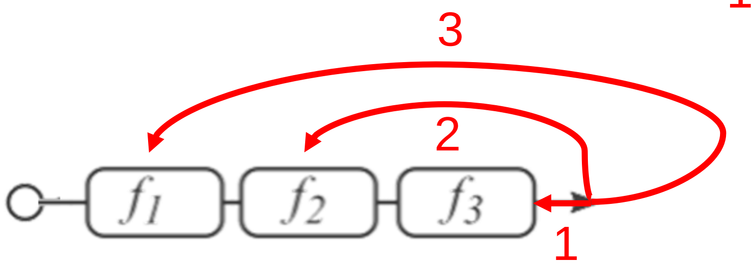
- Residual networks behave like ensembles of relatively shallow networks, it avoid the vanishing gradient problem (think about chain rule) by short paths.





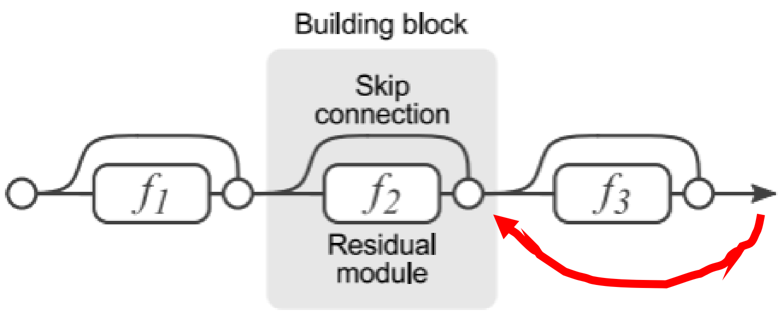
Residual Networks

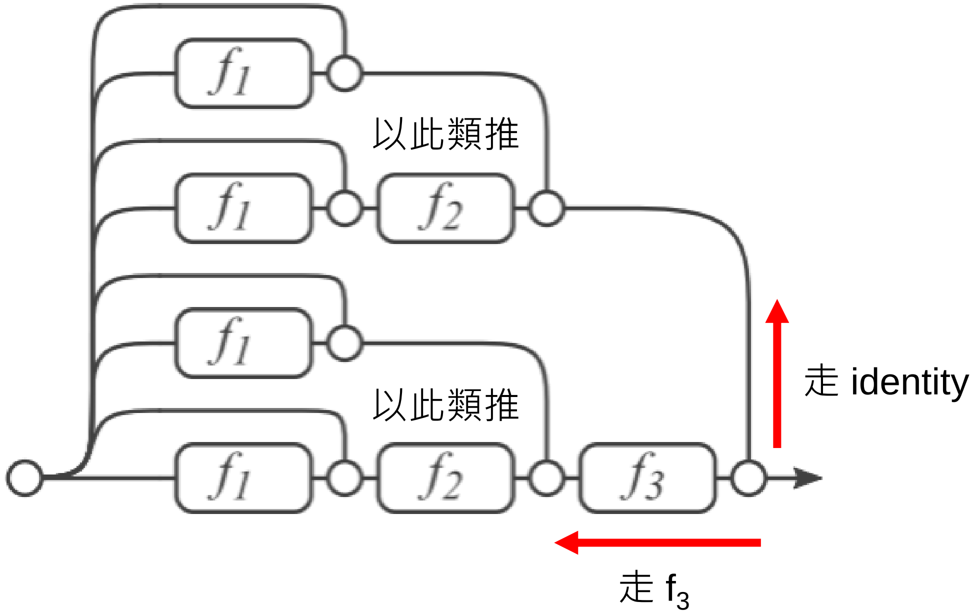




Plain Networks

Veit, A., Wilber, M. J., & Belongie, S. (2016). Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, *29*.



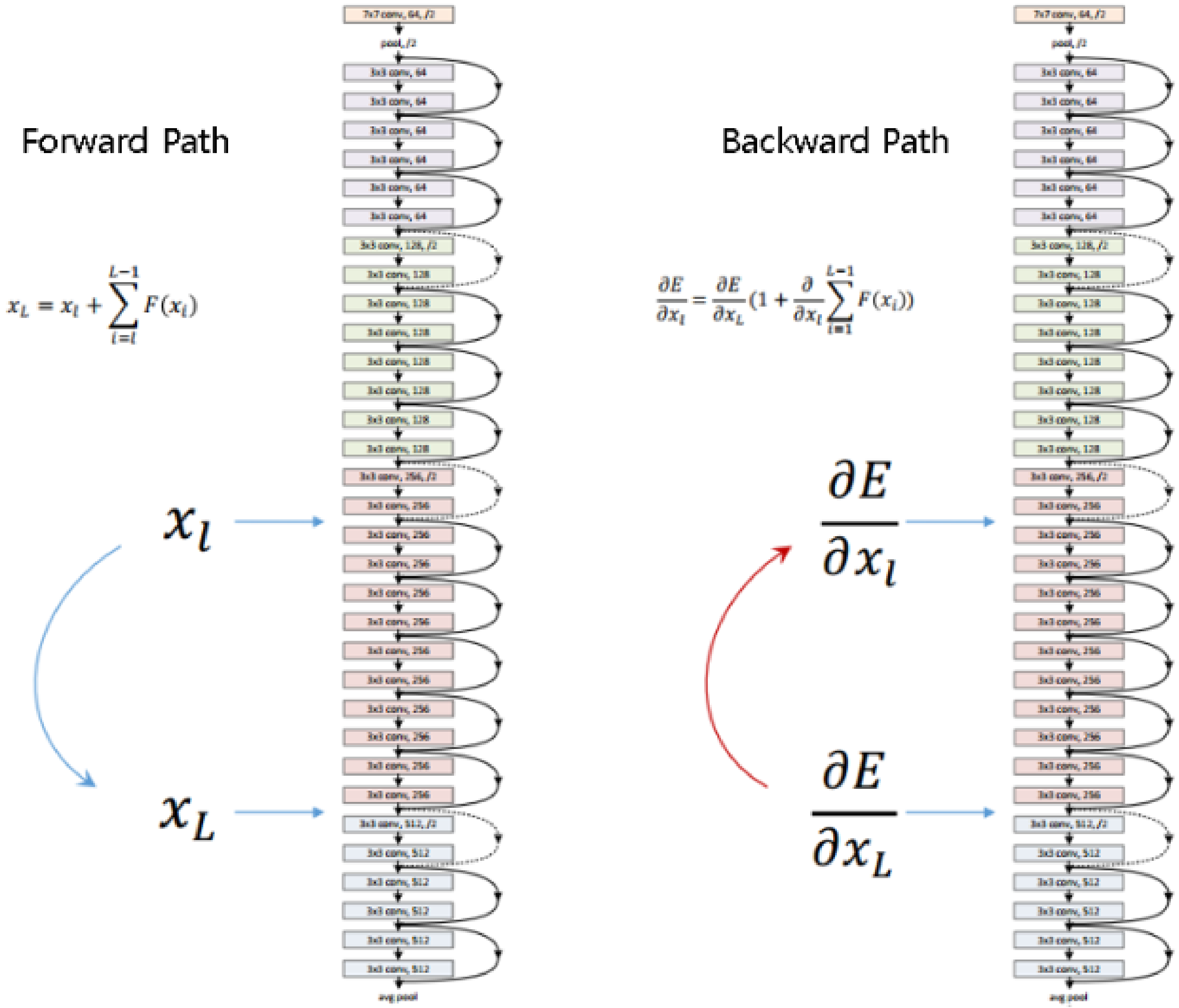


從一個 o 走到前一個 o 都有兩條路可以選擇 (1. 走identity; 2. 走 f_i)。
所以三個residual layers就有 2 x 2 x 2 共 8 條路。

我們把 (1. 走identity; 2. 走 f_i) 展開，
就可以得到等效結構。

Gradient computation of Residual Networks

- Simply add gradients from following layers.



https://m.blog.naver.com/laonple/220793640991

Assignment 2

- Will be announced on 3/29.
- Will be addressed next week.
- Due date will be 4/8.