The Introduction To Artificial Intelligence

Yuni Zeng yunizeng@zstu.edu.cn

2022-2023-1

The Introduction to Artificial Intelligence

- Part I Brief Introduction to AI & Different AI tribes
- Part II Knowledge Representation & Reasoning
- Part III AI GAMES and Searching
- Part IV Model Evaluation and Selection
- Part V Machine Learning
- Part VI Neural Networks

Backpropagation

Conclusion: BP for FNN

Forward computing: $y = f\left(\sum_{i=1}^{n} w_i x_i\right)$

Define cost function: $J = J(w^1, \cdots, w^{L-1})$

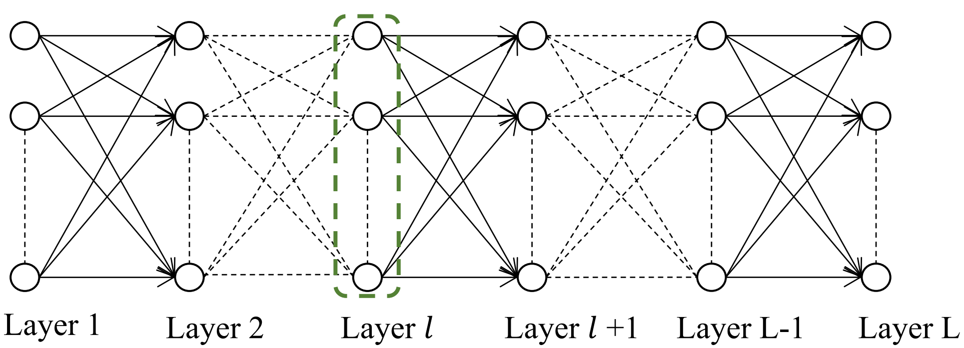
Updating rule: $w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$

Define δ: $\delta_i^l = \frac{\partial J}{\partial z_i^l}$

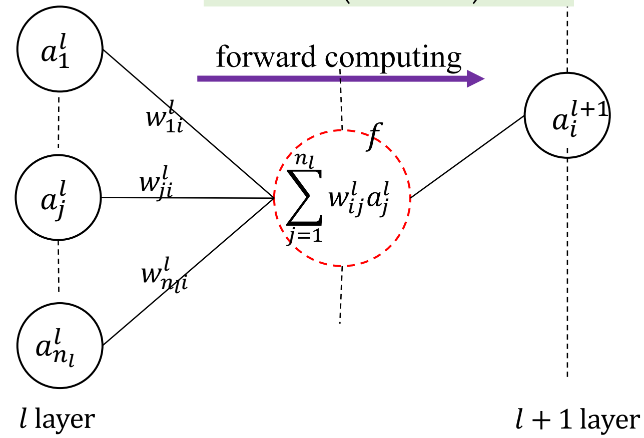
Find the relation: $\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$

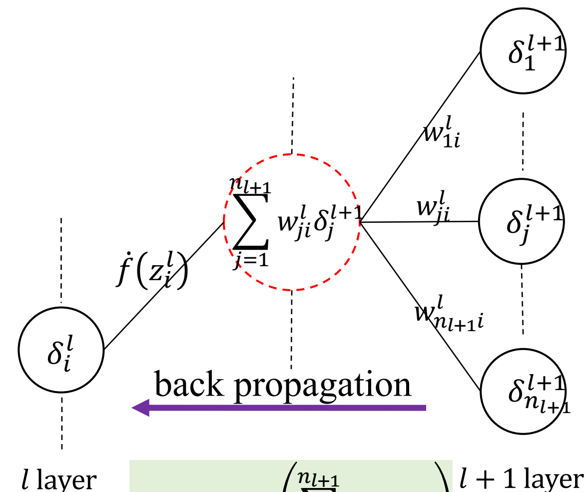
Back propagation: $\delta_i^L = \frac{\partial J}{\partial z_i^L} = (a_i^L - y_i^L) \cdot \dot{f}(z_i^L)$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} \delta_j^{l+1} \cdot w_{ji}^l \right)$$



$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l \right)$$





$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} w_{ji}^l \delta_j^{l+1} \right)$$

Neural Networks

- Brief review
- *Convolutional Neural Network*

    Classical Architecture

    Gradient-Based Learning Algorithm

    Achievements and Applications

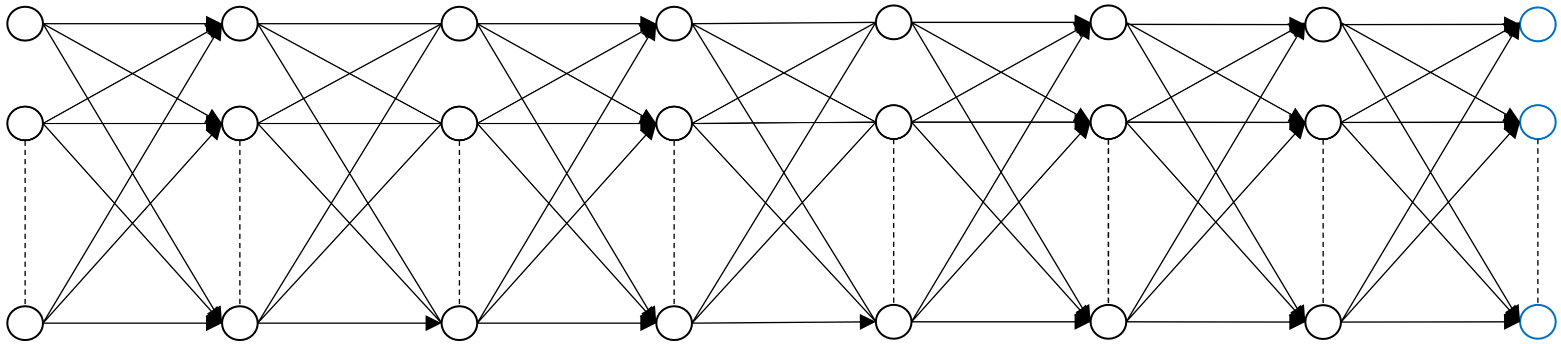
Convolutional Neural Network

Introduction

How to handle the very large colorful image in the size of 224x224x3, where the 3 denotes the R,G,B channels.

28×28

224x224x3



Convolutional Neural Network

Introduction

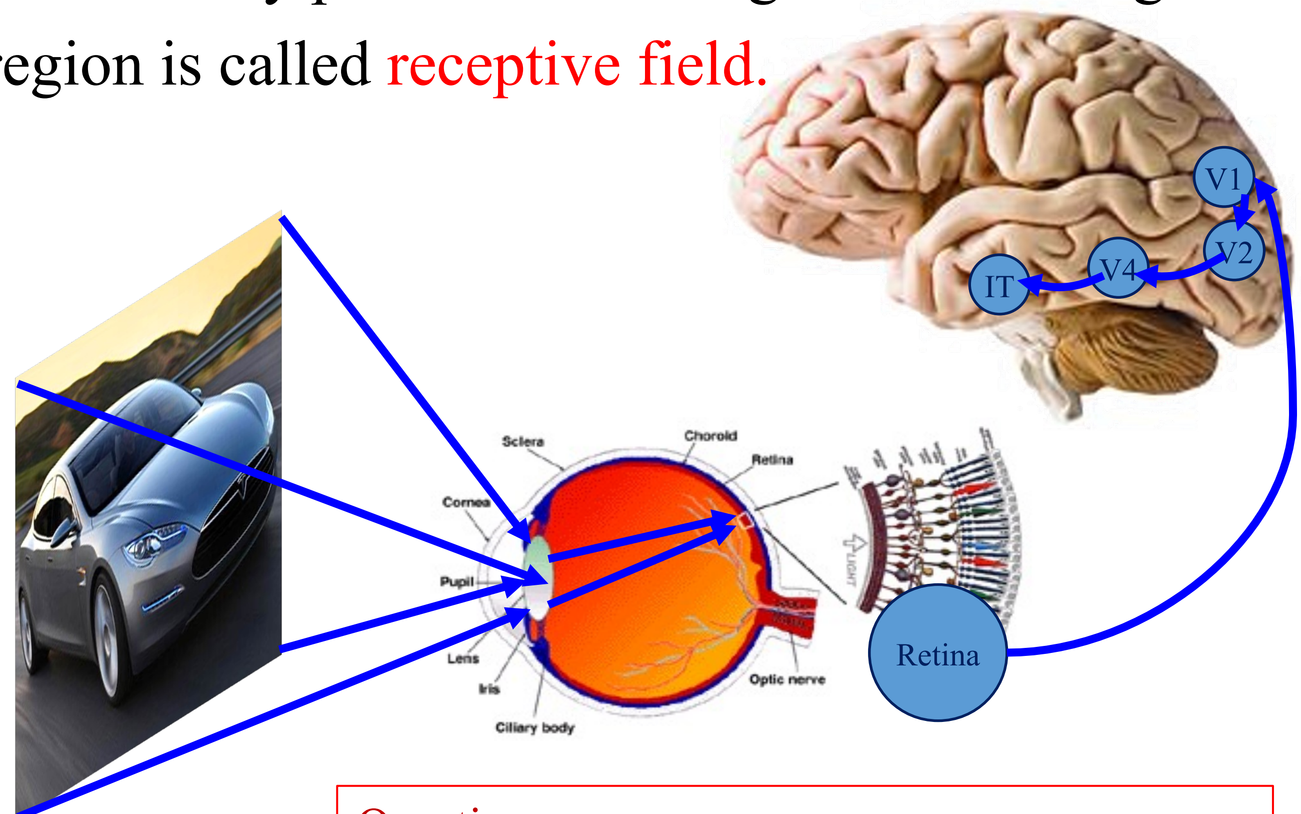
- Suppose the dimension of hidden layer is 100. The number of parameters in the first layer is 15,052,800, unacceptable !
- How does the brain process the image?

224x224x3

Convolutional Neural Network

Introduction

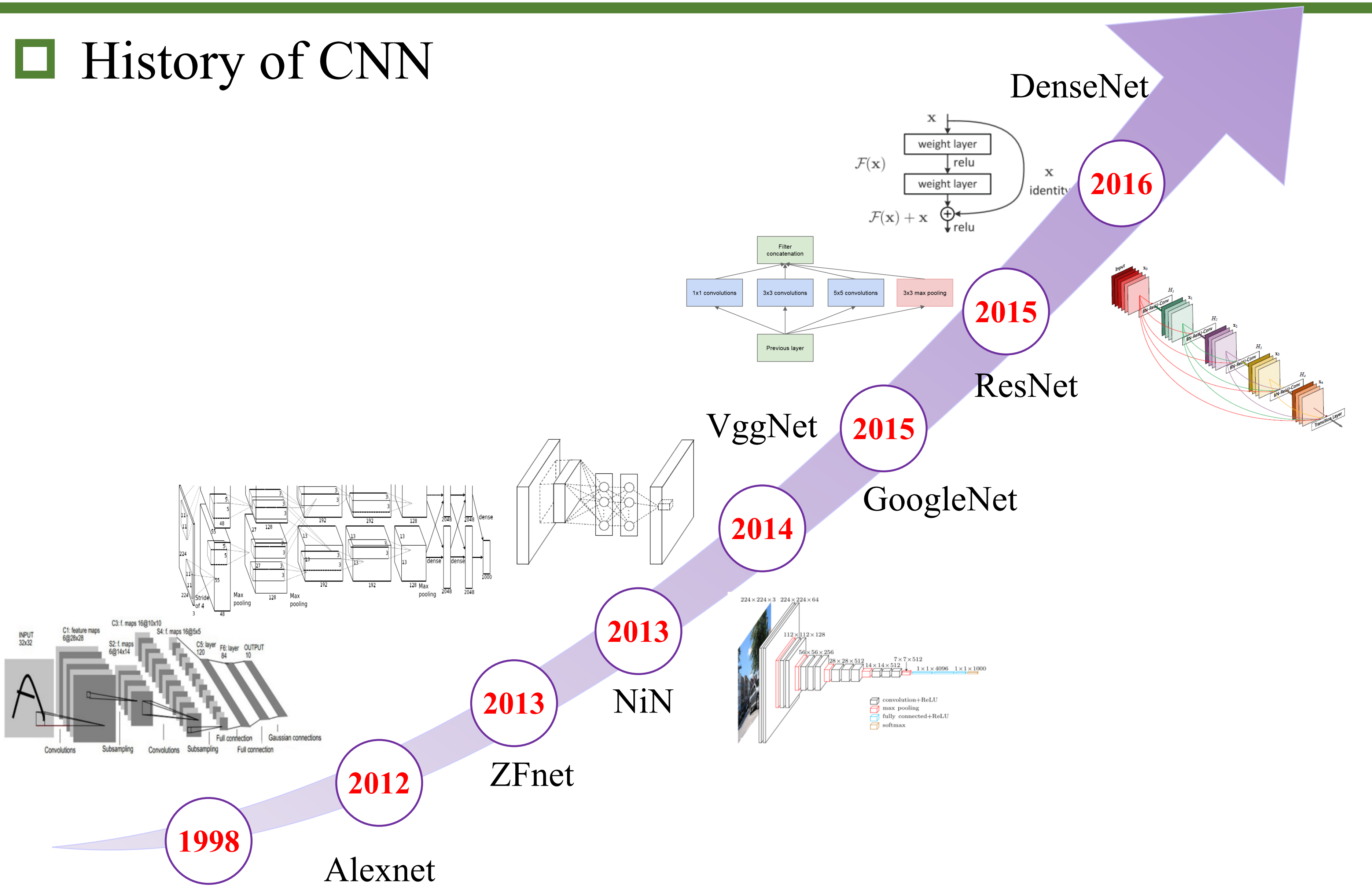
- Each neuron can only perceive a sub-region in the image.
- The sub-region is called receptive field.



Question:
How to build the model of receptive field?

Convolutional Neural Network

History of CNN



Convolutional Neural Network

Functional Architecture in Visual Cortex (1962)

David Hubel (right) and Torsten Wiesel (left) celebrate for Nobel Prize

- ***Receptive Field*** is a continuous sub-region of input space
- ***Simple Cells*** detect local features within a receptive field
- ***Complex Cells*** "pool" the output of Simple Cells within a receptive field

Convolutional Neural Network

- NeoCognitron (1979)
  - ***Neocognitron*** is a multilayered neural network that cascades models of Complex cells and Simple cells in Visual Cortex.

Kunihiko Fukushima

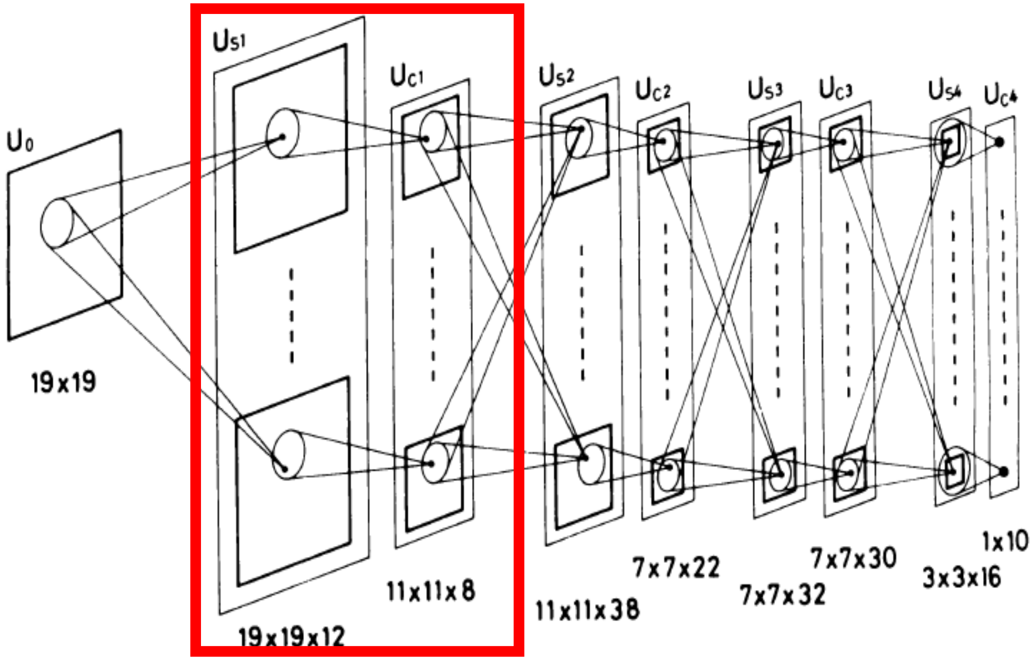


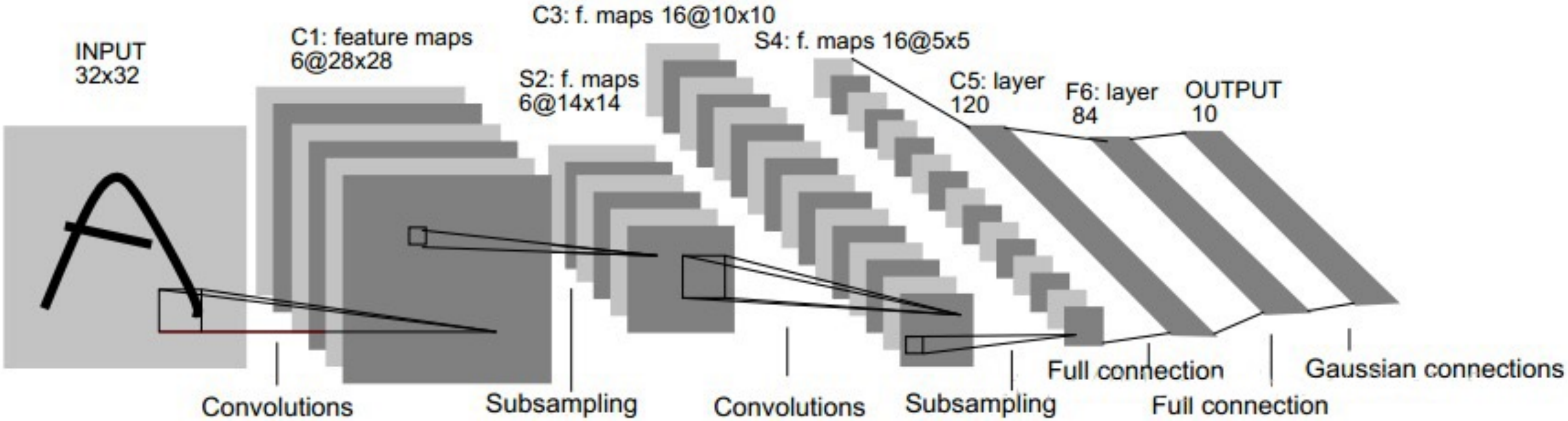
Fig. 2. Schematic diagram illustrating synaptic connections between layers in neocognitron.

Convolutional Neural Network

LeNet (1998)

- ***LeNet*** is a refinement of NeoCongnitron, which can be trained efficiently and achieve state of art results.

Yann LeCun



Neural Networks

- Brief review
- Convolutional Neural Network

    Classical Architecture

    Gradient-Based Learning Algorithm

    Achievements and Applications

Convolutional Neural Network

Classical Architecture

- Three Main Concepts in CNNs

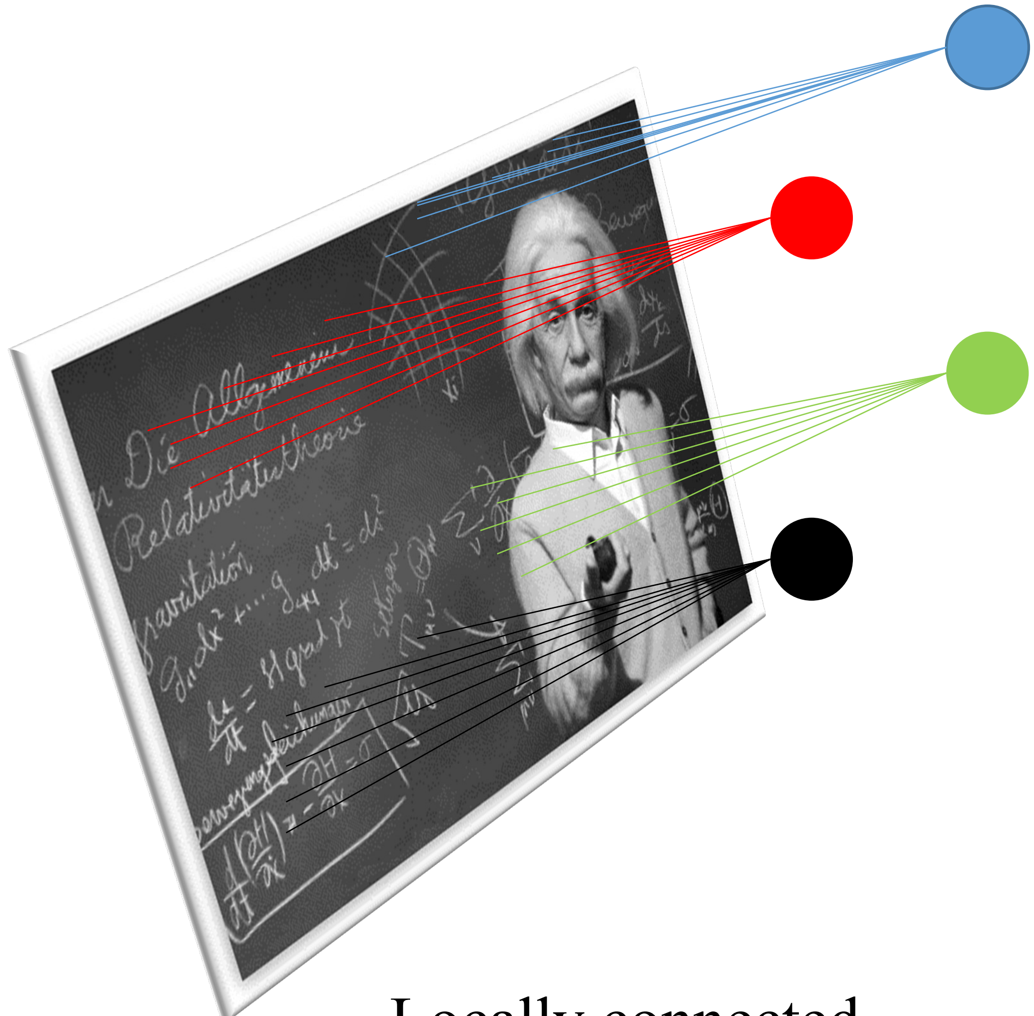
  - Receptive Field
  - Convolution (Simple Cell)
  - Pooling (Complex Cell)

Convolutional Neural Network

Classical Architecture

- Receptive Field

Fully connected



Locally connected

Convolutional Neural Network

Classical Architecture of CNN

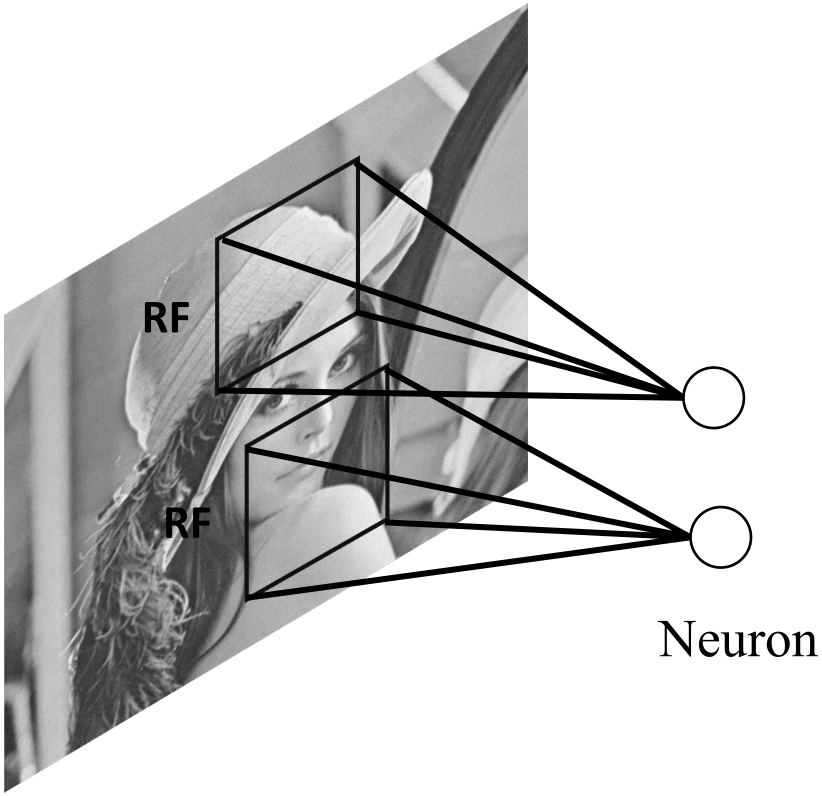
- Receptive Field

Receptive Field

- Receptive Field of a neuron is a ***continuous sub-region*** of the input space

Locally Connected

- Neuron is just connected to its own Receptive Field



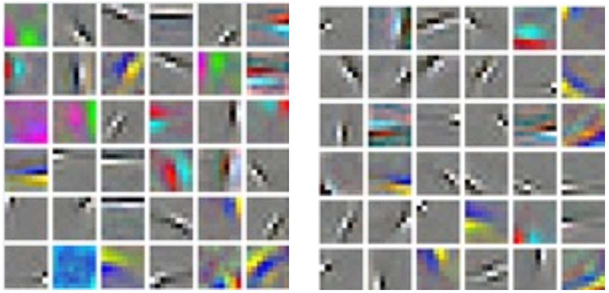
Convolutional Neural Network

Classical Architecture

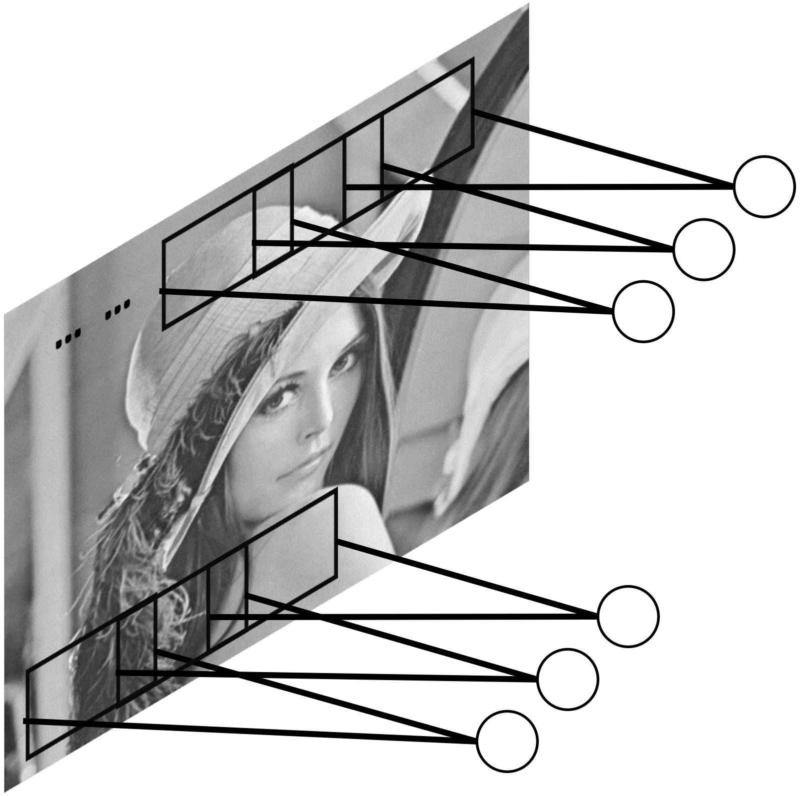
- Convolution (Simple Cell)

Simple Cell

- Simple Cells detect local feature.



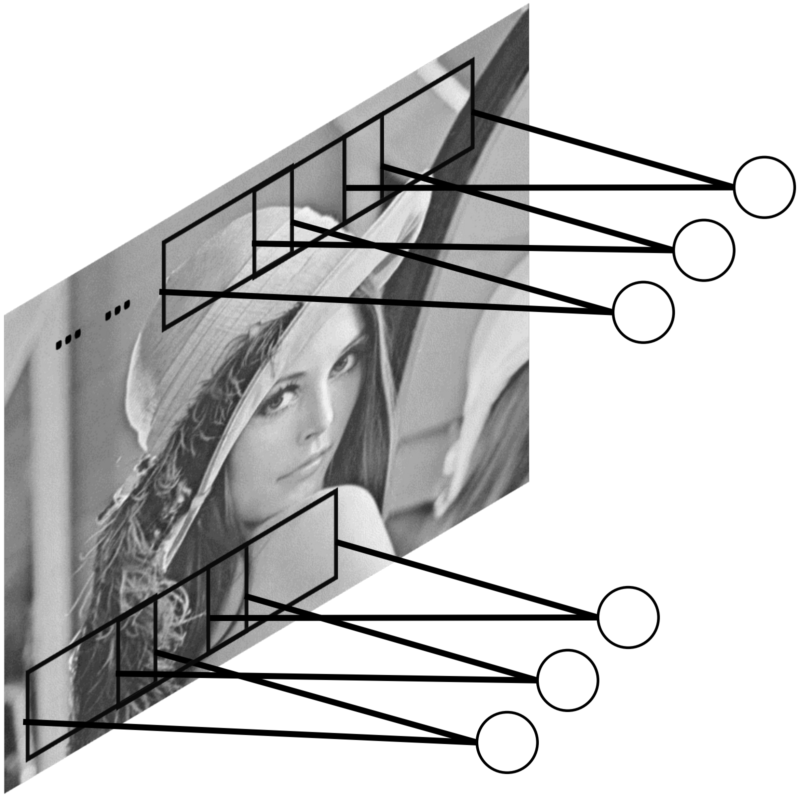
- Their Receptive Fields are ***overlapped***.



Convolutional Neural Network

Classical Architecture

- Convolution (Simple Cell)
  - **Shared Weight**
    - All units share the same set of weights
  - **Why Shared Weight**
    - Features that are useful on one part of the image and probably useful elsewhere
    - Shared Weight can reduce the number of parameters

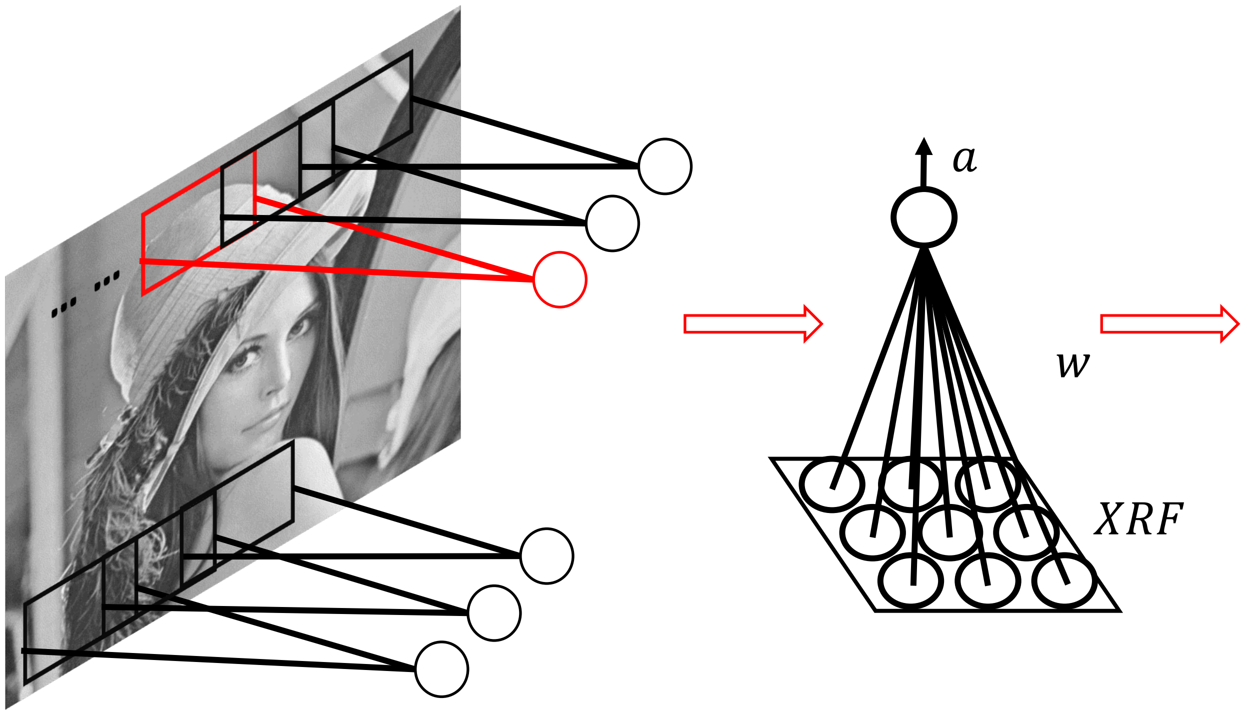


Convolutional Neural Network

Classical Architecture

- Convolution (Simple Cell)

  - Activation of a Neuron

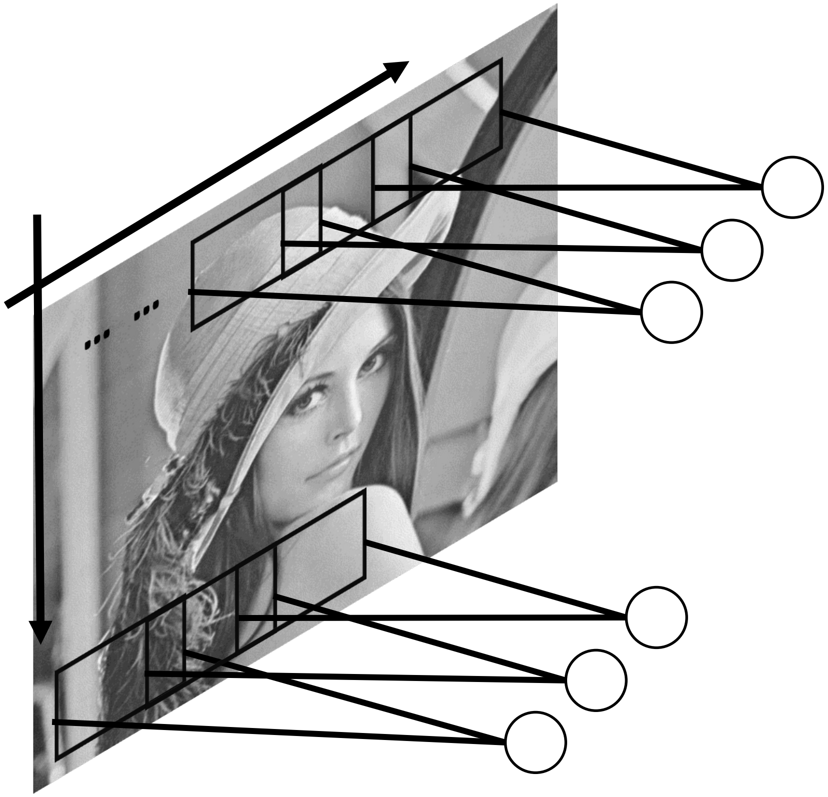


$$\begin{cases} z = \sum_{kl} W_{kl} XRF_{k,l} \\ a = f(z) \end{cases}$$

Convolutional Neural Network

Classical Architecture

- Convolution (Simple Cell)
  - Feature Map
  - Activations form a 2D array called Feature Map
  - $\begin{cases} z_{ij} = \sum_{kl} \mathrm{W}_{kl} X_{i+k,j+l} \\ a_{ij} = f(z_{ij}) \end{cases}$
  - where X_{ij} is receptive field of $ij-$th neuron



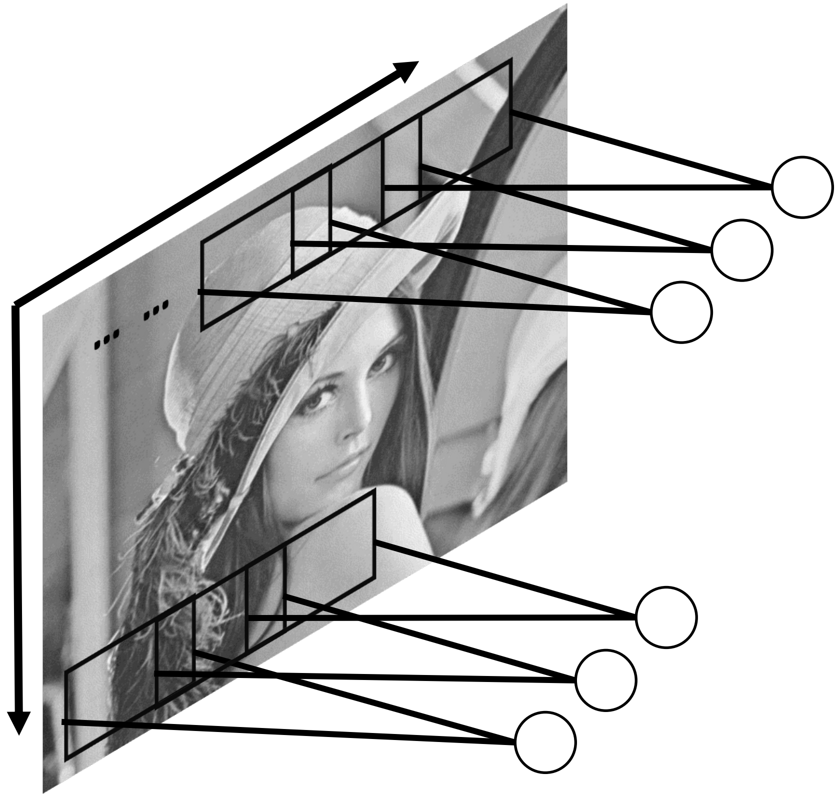
Convolutional Neural Network

Classical Architecture

- Convolution (Simple Cell)

Convolution

- With a learned filter (Weights W)
- Non-Linearity activation function
- $\begin{cases} z = W * X \\ a = f(z) \end{cases}$



Convolutional Neural Network

Classical Architecture - Convolution

0

1	0	1
1	0	1
1	0	1

1

1	1	1
0	0	0
1	1	1

Convolutional Neural Network

Classical Architecture - Convolution

0

1	0	1
1	0	1
1	0	1

1

1	1	1
0	0	0
1	1	1

0	0.5	0
0	1	0
0	0.5	0

Convolutional Neural Network

Classical Architecture - Convolution

$$\begin{array}{cc} 0 & \begin{bmatrix}1&0&1\\1&0&1\\1&0&1\end{bmatrix} \\ \\ 1 & \begin{bmatrix}1&1&1\\0&0&0\\1&1&1\end{bmatrix}\end{array} \times \begin{bmatrix}0&0.5&0\\0&1&0\\0&0.5&0\end{bmatrix} = \begin{array}{c}\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} \\ \\ \begin{bmatrix}0&0.5&0\\0&0&0\\0&0.5&0\end{bmatrix}\end{array} \quad \sum \quad \begin{array}{c}0 \\ \\ \\ \\ 1\end{array}$$

Element wise product

Convolutional Neural Network

Classical Architecture - Convolution

0

1	0	1
1	0	1
1	0	1

A

0	1	1
1	0	1
1	1	0

C

0	0.5	0
0	1	0
0	0.5	0

1

1	1	1
0	0	0
1	1	1

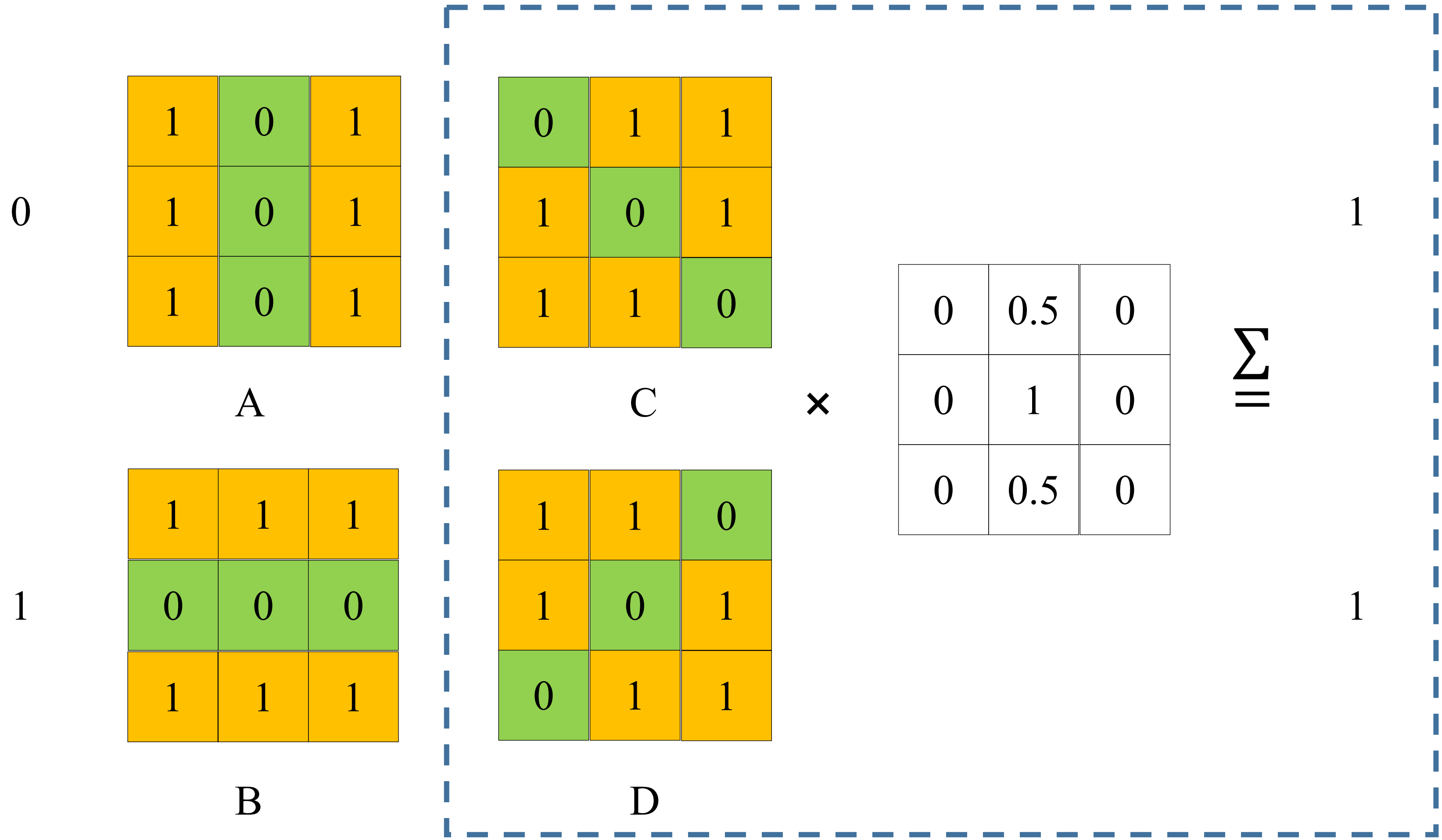
B

1	1	0
1	0	1
0	1	1

D

Convolutional Neural Network

Classical Architecture - Convolution



Convolutional Neural Network

Classical Architecture - Convolution

0

1	0	1
1	0	1
1	0	1

A

0	1	1
1	0	1
1	1	0

C

0	0.5	0
0	1	0
0	0.5	0

0.5	0	0
0	1	0
0	0	0.5

1

1	1	1
0	0	0
1	1	1

B

1	1	0
1	0	1
0	1	1

D

Convolutional Neural Network

Classical Architecture - Convolution

0

1	0	1
1	0	1
1	0	1

A

0	1	1
1	0	1
1	1	0

C

1

1	1	1
0	0	0
1	1	1

B

1	1	0
1	0	1
0	1	1

D

0	0.5	0
0	1	0
0	0.5	0

0.5	0	0
0	1	0
0	0	0.5

A	0	1
B	1	1
C	1	0
D	1	1

Convolutional Neural Network

Classical Architecture - Convolution

Convolutional Kernels are special masks(filters) that respond to specific patterns in the input image.

0	0.5	0
0	1	0
0	0.5	0

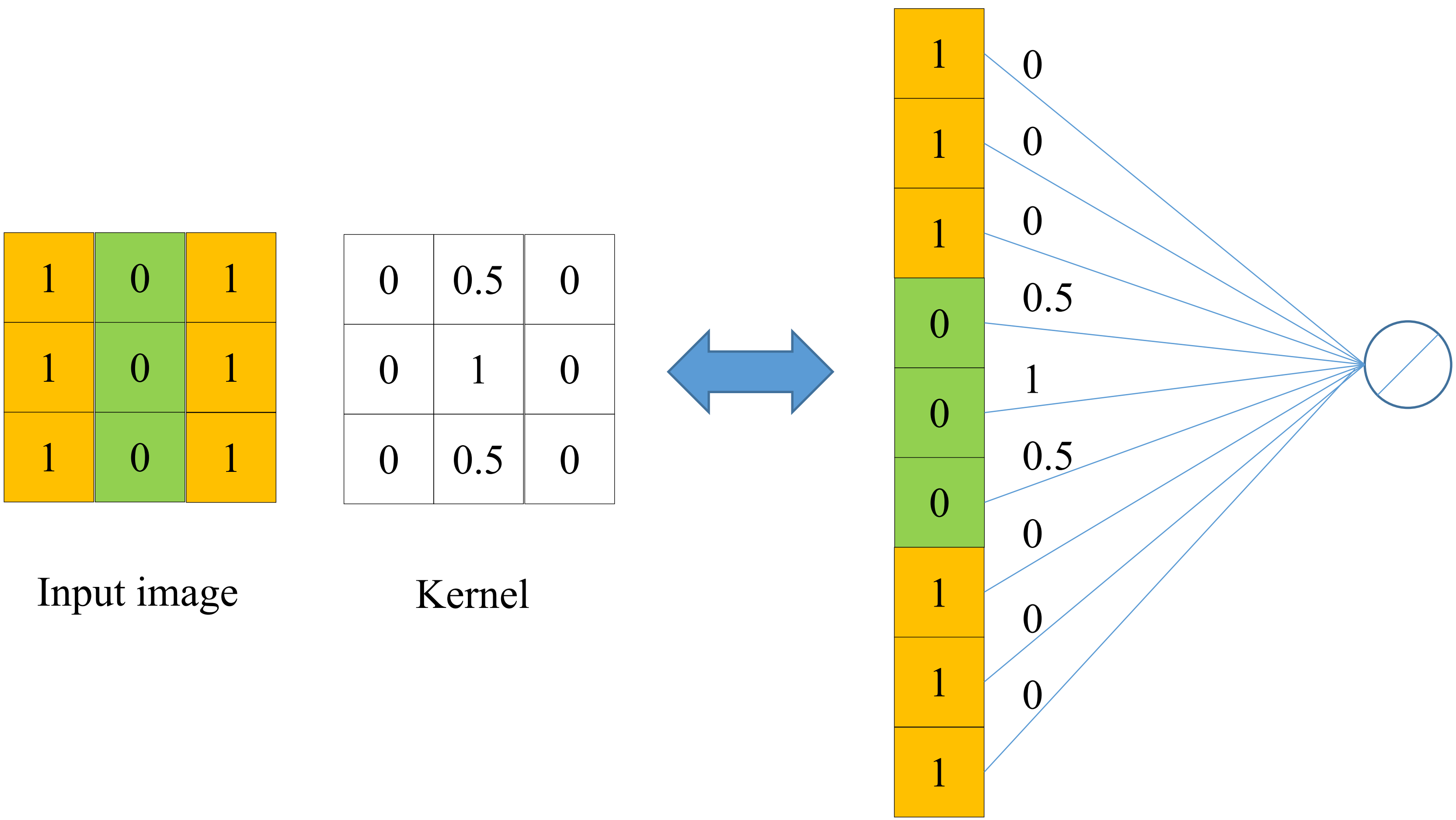
0.5	0	0
0	1	0
0	0	0.5

*	0	*
*	0	*
*	0	*

0	*	*
*	0	*
*	*	0

Convolutional Neural Network

Classical Architecture - Convolution



Convolutional Neural Network

Classical Architecture - Convolution

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

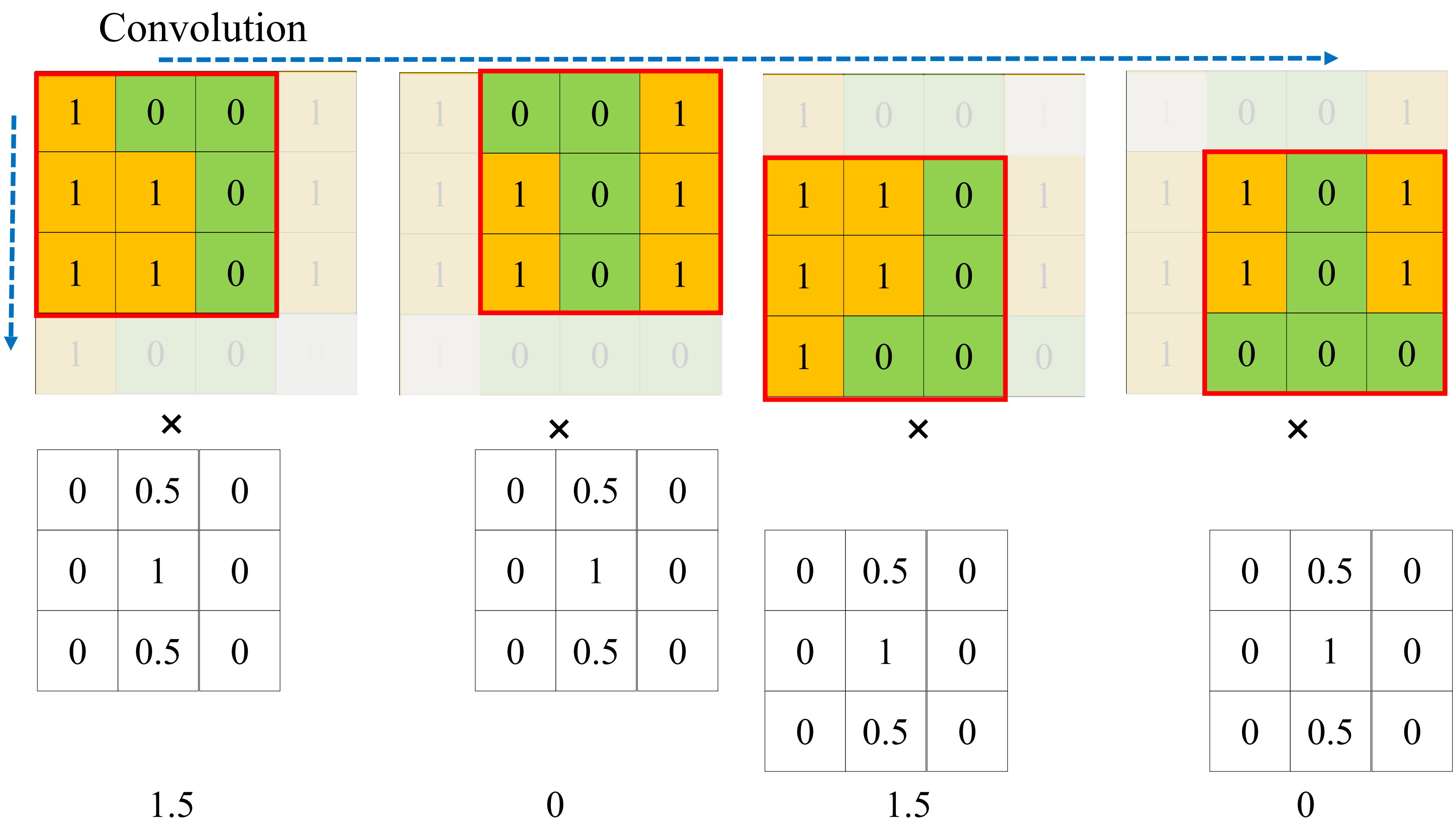
Input image

0	0.5	0
0	1	0
0	0.5	0

Kernel

Convolutional Neural Network

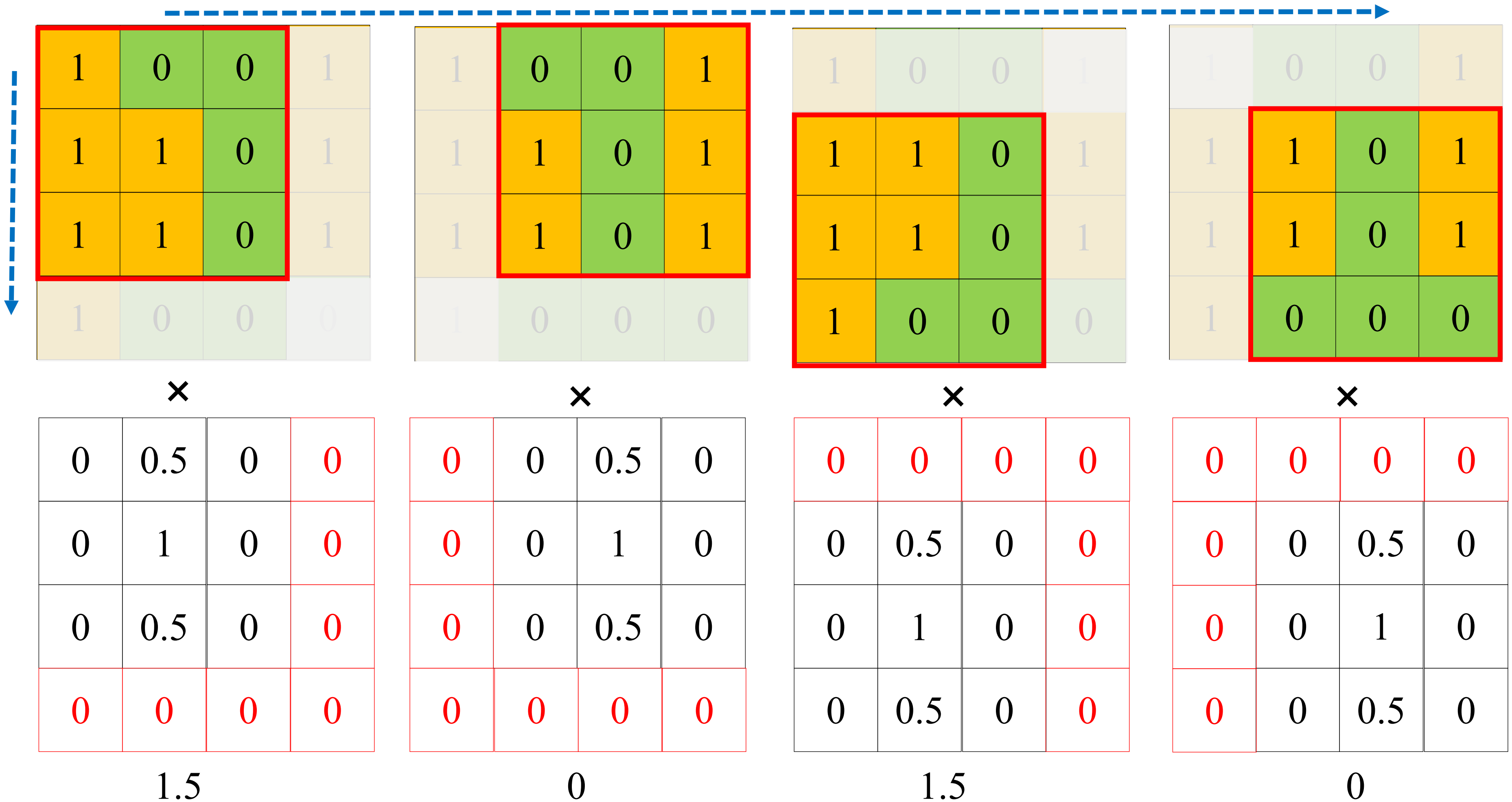
Classical Architecture - Convolution



Convolutional Neural Network

Classical Architecture - Convolution

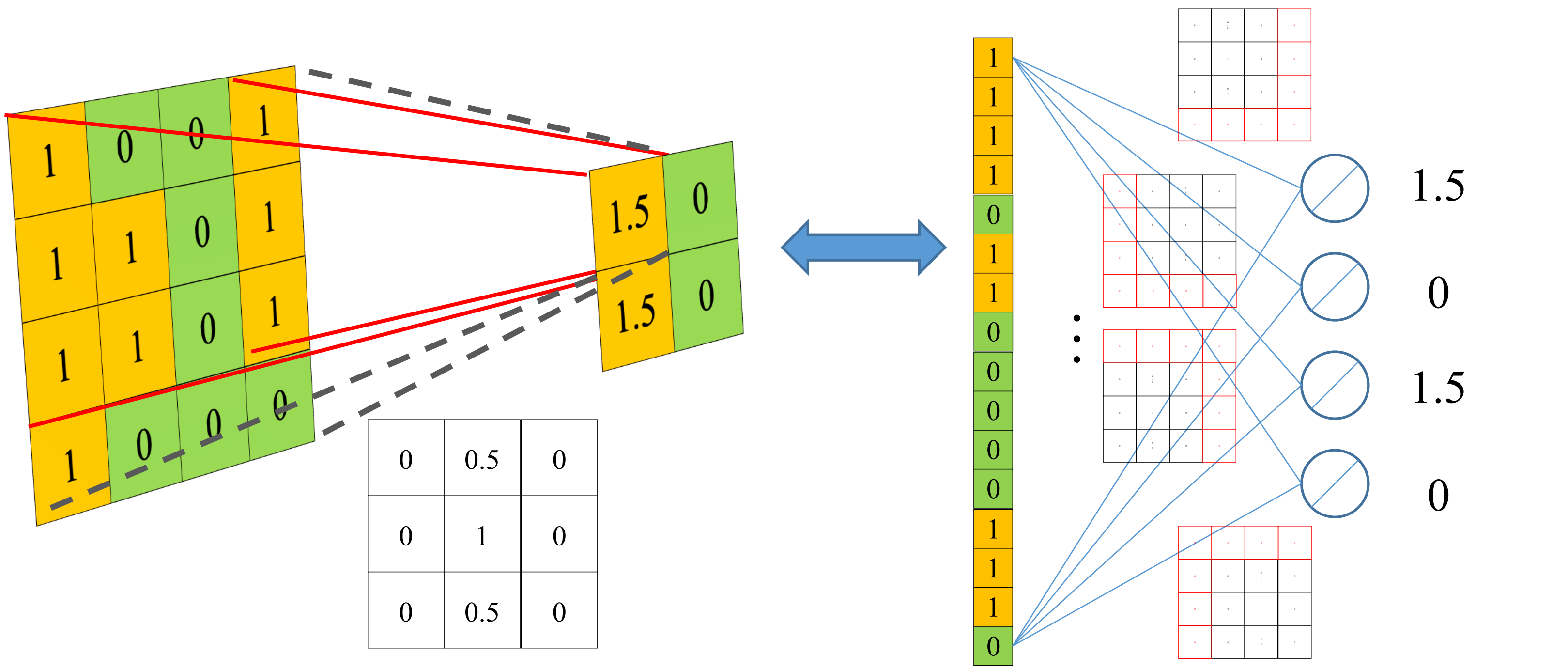
Convolution



Convolutional Neural Network

Classical Architecture - Convolution

Each convolutional kernel is equivalent to several spatially constrained FC neurons



$1\ convolution\ kernel$ ⟺ $(W - W_k + 1) \times (H - H_k + 1)\ neurons$

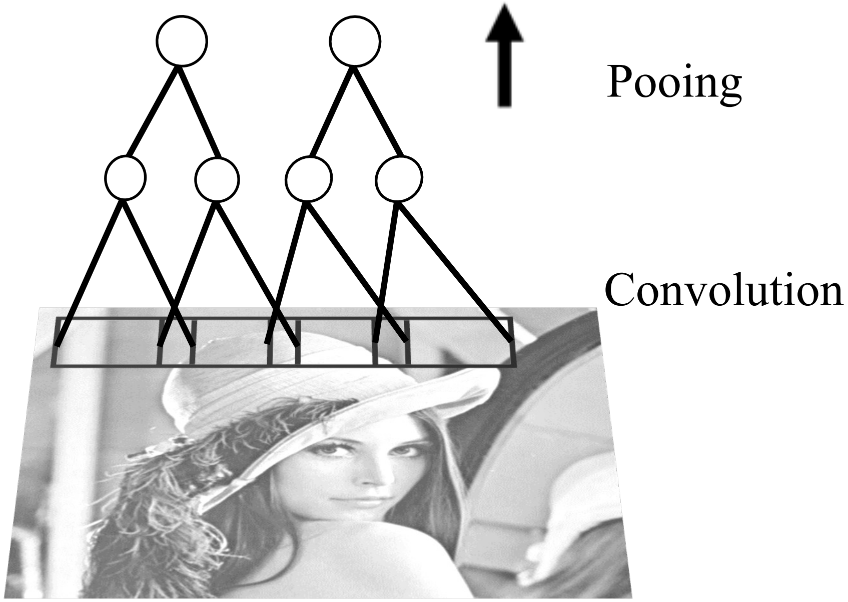
Convolutional Neural Network

Classical Architecture - Example

Input $X = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$

Filter $w = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

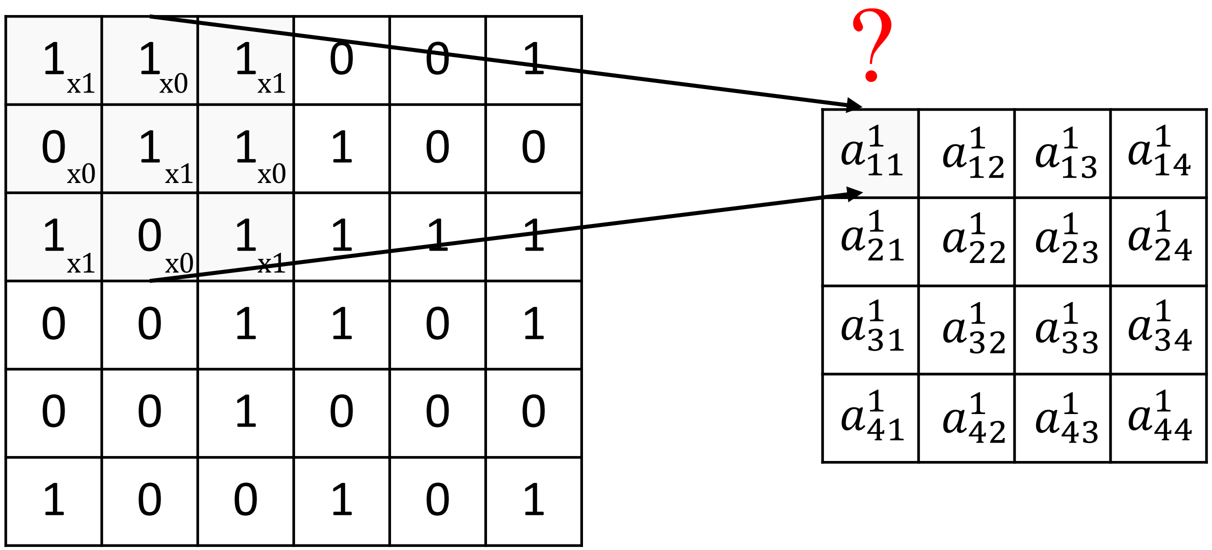
FeedForward $\begin{cases} z_{ij} = \sum_{kl} W_{kl} X_{i+k,j+l} \\ a_{ij} = f(z_{ij}) = z_{ij} \end{cases}$



Convolutional Neural Network

Classical Architecture - Example

$$
\begin{cases}
z_{11}^{1} = 1\times1 + 1\times0 + 1\times1 + 0\times0 + 1\times1 + 1\times0 + 1\times1 + 0\times0 + 1\times1 \\
\quad = 5 \\
a_{11}^{1} = f(z_{11}^{1}) = 5
\end{cases}
$$



Input

Feature Map

Convolutional Neural Network

Classical Architecture - Example

$$\begin{cases} z_{12}^{1} = 1\times1 + 1\times0 + 0\times1 + 1\times0 + 1\times1 + 1\times0 + 0\times1 + 1\times0 + 1\times1 \\ \quad\;\; = 3 \\ a_{12}^{1} = f(z_{12}^{1}) = 3 \end{cases}$$

1	1 $_{\times 1}$	1 $_{\times 0}$	0 $_{\times 1}$	0	1
0	1 $_{\times 0}$	1 $_{\times 1}$	1 $_{\times 0}$	0	0
1	0 $_{\times 1}$	1 $_{\times 0}$	1 $_{\times 1}$	1	1
0	0	1	1	0	1
0	0	1	0	0	0
1	0	0	1	0	1

Input

5	a_{12}^{1}	a_{13}^{1}	a_{14}^{1}
a_{21}^{1}	a_{22}^{1}	a_{23}^{1}	a_{24}^{1}
a_{31}^{1}	a_{32}^{1}	a_{33}^{1}	a_{34}^{1}
a_{41}^{1}	a_{42}^{1}	a_{43}^{1}	a_{44}^{1}

Feature Map

Convolutional Neural Network

Classical Architecture - Example

1	1	1_{x1}	0_{x0}	0_{x1}	1
0	1	1_{x0}	1_{x1}	0_{x0}	0
1	0	1_{x1}	1_{x0}	1_{x1}	1
0	0	1	1	0	1
0	0	1	0	0	0
1	0	0	1	0	1

Input

1	1	1	0_{x1}	0_{x0}	1_{x1}
0	1	1	1_{x0}	0_{x1}	0_{x0}
1	0	1	1_{x1}	1_{x0}	1_{x1}
0	0	1	1	0	1
0	0	1	0	0	0
1	0	0	1	0	1

Input

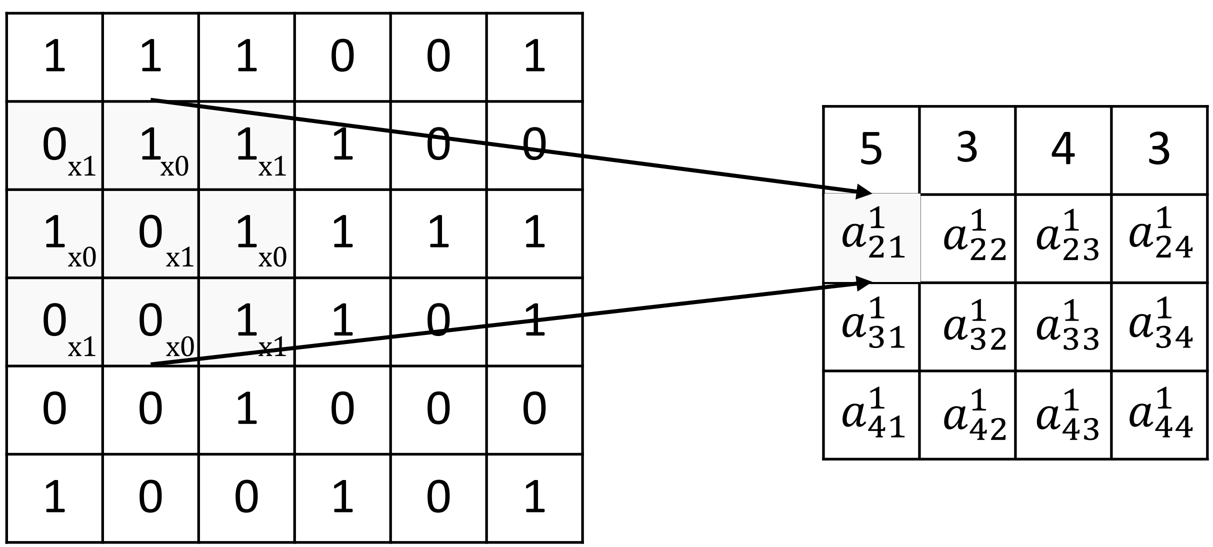
5	3	a_{13}^{1}	a_{14}^{1}
a_{21}^{1}	a_{22}^{1}	a_{23}^{1}	a_{24}^{1}
a_{31}^{1}	a_{32}^{1}	a_{33}^{1}	a_{34}^{1}
a_{41}^{1}	a_{42}^{1}	a_{43}^{1}	a_{44}^{1}

Feature Map

Convolutional Neural Network

Classical Architecture - Example

$$\begin{cases} z_{21}^{1} = \quad 0\times1 + 1\times0 + 1\times1 + 1\times0 + 0\times1 + 1\times0 + 0\times1 + 0\times0 + 1\times1 \\ \quad\;\; = 2 \\ a_{21}^{1} = f(z_{21}^{1}) = 2 \end{cases}$$



Input

Feature Map

Convolutional Neural Network

Classical Architecture - Example

$$\begin{cases} z^1 = W * X \\ a^1 = f(z^1) \end{cases}$$

1	1	1	0	0	1
0_{x1}	1_{x0}	1_{x1}	1	0	0
1_{x0}	0_{x1}	1_{x0}	1	1	1
0_{x1}	0_{x0}	1_{x1}	1	0	1
0	0	1	0	0	0
1	0	0	1	0	1

Input

5	3	4	3
2	4	3	4
3	3	4	2
2	3	2	4

Feature Map

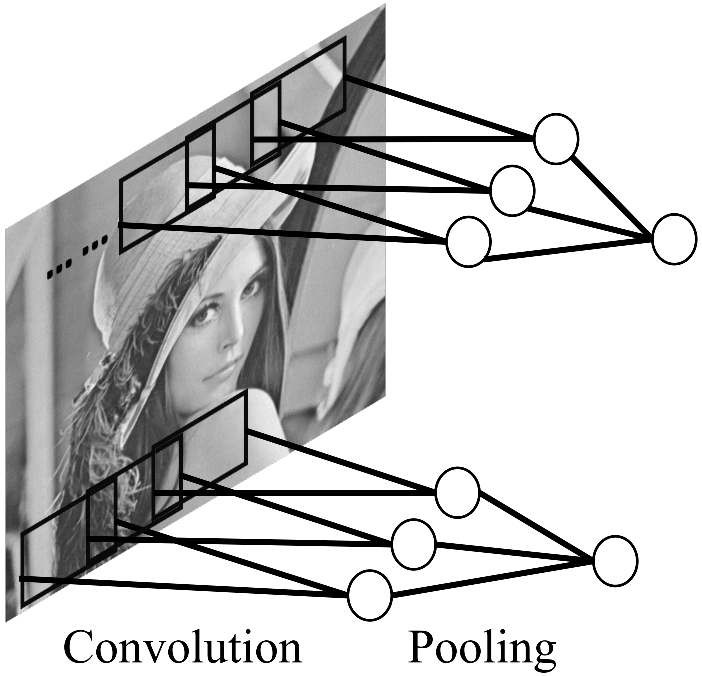
Convolutional Neural Network

Classical Architecture

- Pooling (Complex Cell)

  - Complex Cell

    - Complex Cell "pool" the output of Simple Cells within its receptive field
    - Their Receptive Fields are ***non-overlapped***.



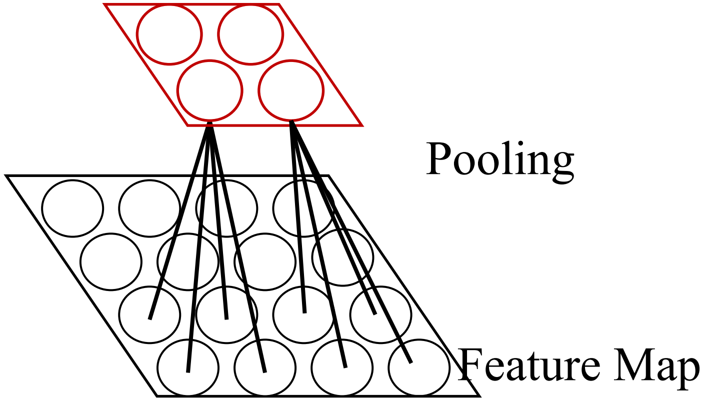
Convolutional Neural Network

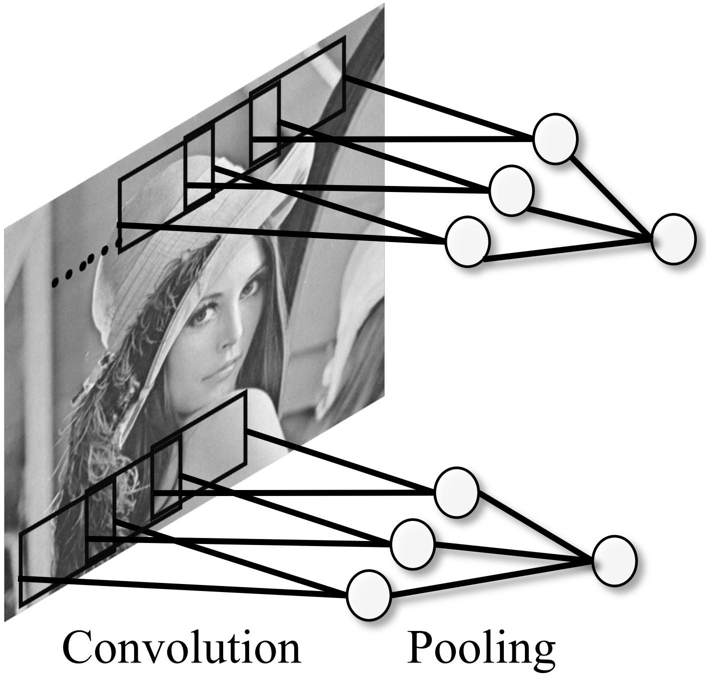
Classical Architecture

- Pooling (Complex Cell)

Pooling

- Pooling is an aggregation operation applied to receptive field on feature map
- Sometimes are max/mean pooling





Convolutional Neural Network

Classical Architecture - Example

Max Pooing

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

Input

5	3	4	3
2	4	3	4
3	3	4	2
2	3	2	4

Feature Map

?

a_{11}^2	a_{12}^2
a_{21}^2	a_{22}^2

Pooled Feature Map

Convolutional Neural Network

Classical Architecture - Example

Max Pooing

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

Input

5	3	4	3
2	4	3	4
3	3	4	2
2	3	2	4

Feature Map

5	a_{12}^2
a_{21}^2	a_{22}^2

Pooled Feature Map

Convolutional Neural Network

Classical Architecture - Example

Max Pooing

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

Input

5	3	4	3
2	4	3	4
3	3	4	2
2	3	2	4

Feature Map

5	4
a_{21}^{2}	a_{22}^{2}

Pooled Feature Map

Convolutional Neural Network

Classical Architecture - Example

Max Pooing

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

Input

5	3	4	3
2	4	3	4
3	3	4	2
2	3	2	4

Feature Map

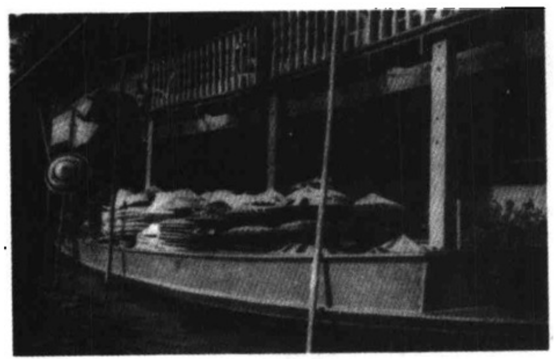
5	4
3	4

Pooled Feature Map

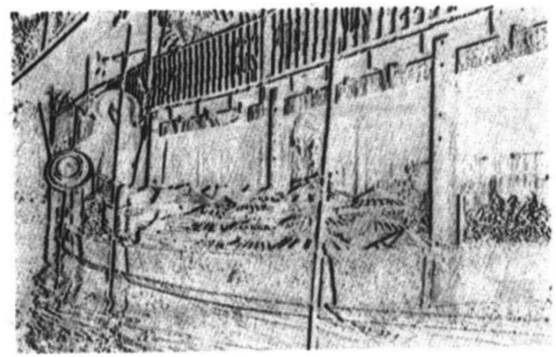
Convolutional Neural Network

Classical Architecture

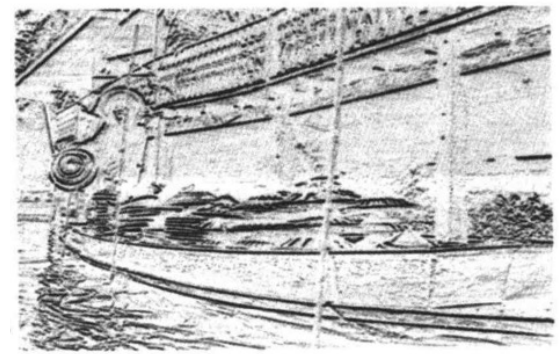
- Multiple Convolutions with Different Filters
- Different filters detect different features



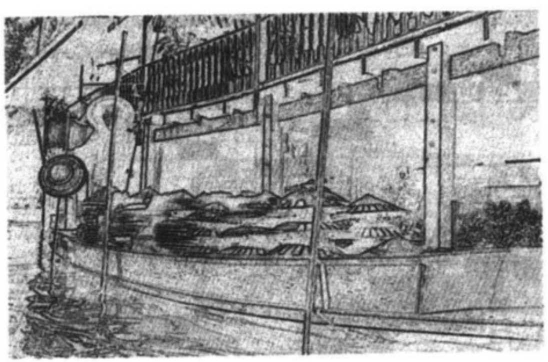
Raw Image



Vertical Edge Detect by w1

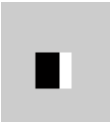


Horizontal Edge Detect by w2



Horizontal & Vertical

$$w1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

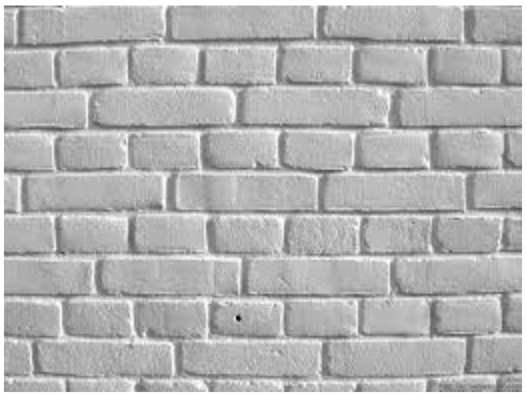


$$\text{w2}=\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolutional Neural Network

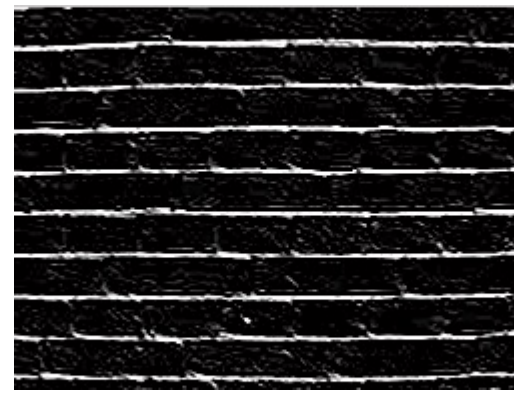
Classical Architecture

- Multiple Convolutions with Different Filters
  - Different filters detect different features

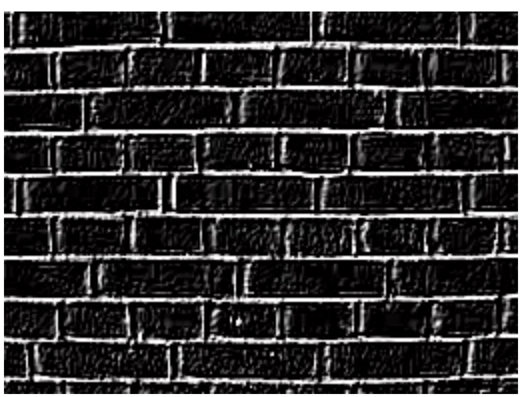


Raw Image

Vertical Edge Detect by w1



Horizontal Edge Detect by w2



Horizontal & Vertical

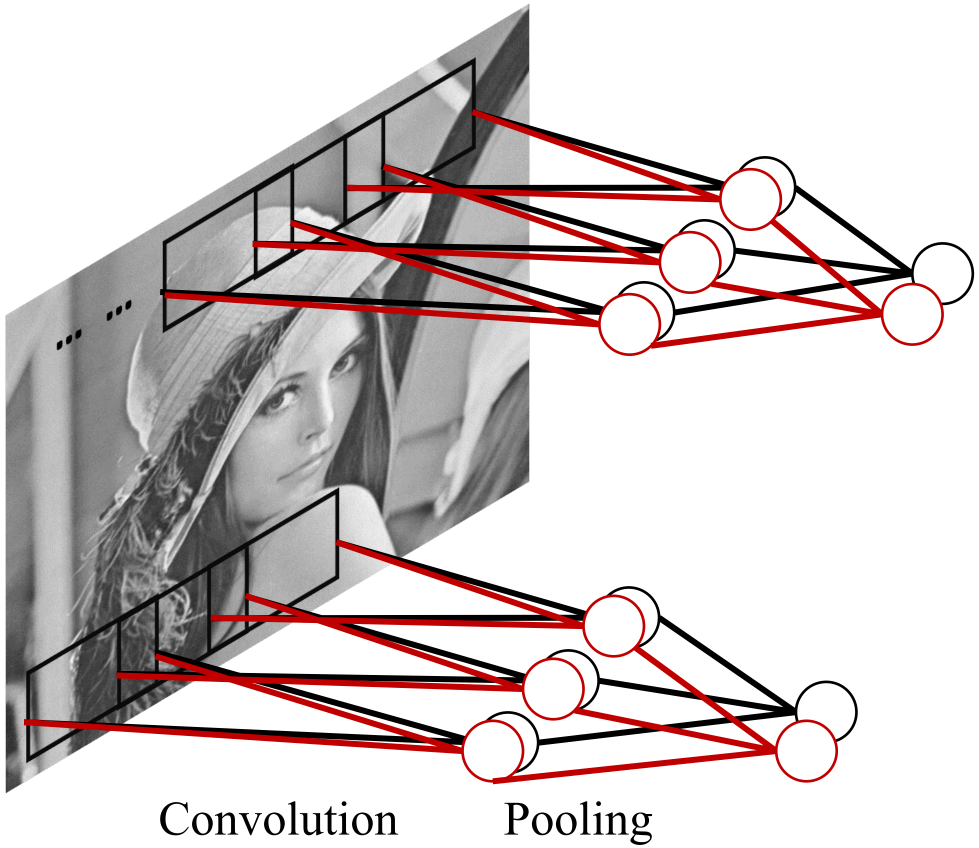
$$w1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$w2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolutional Neural Network

Classical Architecture

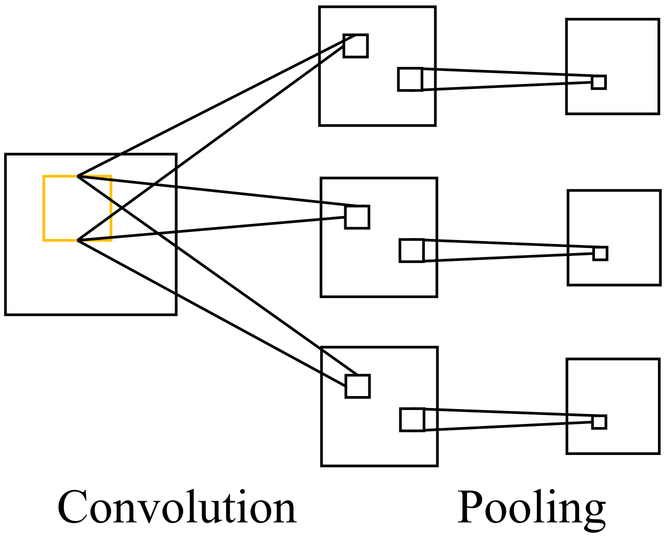
- Multiple Convolutions with Different Filters
  - Detect multiple features at each receptive field

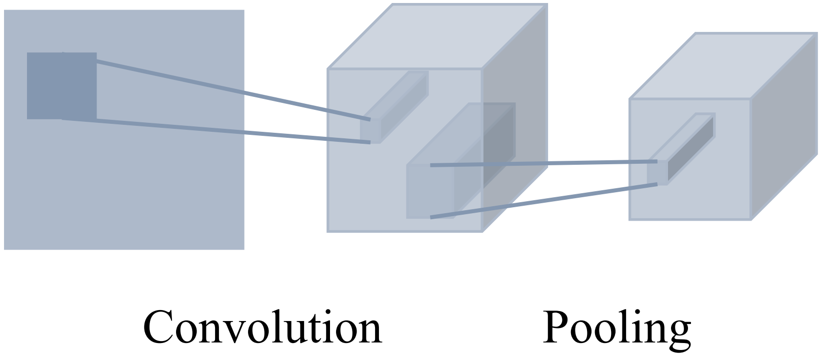


Convolutional Neural Network

Classical Architecture

- Multiple Convolutions with Different Filters
  - There results a 3D array, where each slice is a feature map.
  - Then pooling each feature map individually

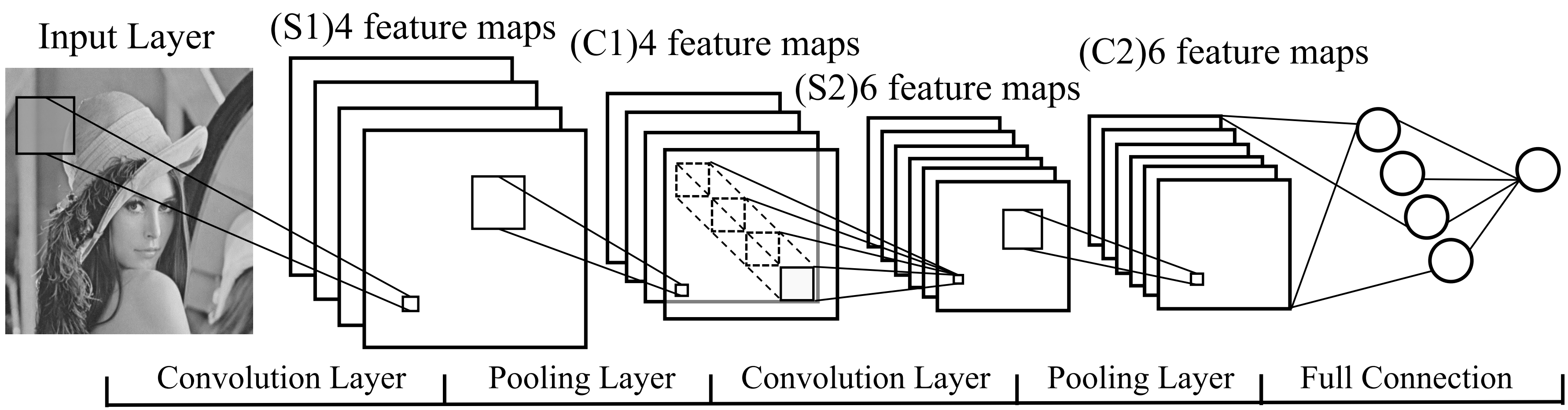




Convolutional Neural Network

Classical Architecture

- Added by Full Connection Layers



Neural Networks

- Brief review
- Convolutional Neural Network

  Classical Architecture

  Gradient-Based Learning Algorithm

  Achievements and Applications

Backpropagation

Conclusion: BP for FNN

Forward computing: $y = f\left(\sum_{i=1}^{n} w_i x_i\right)$

Define cost function: $J = J(w^1, \cdots, w^{L-1})$

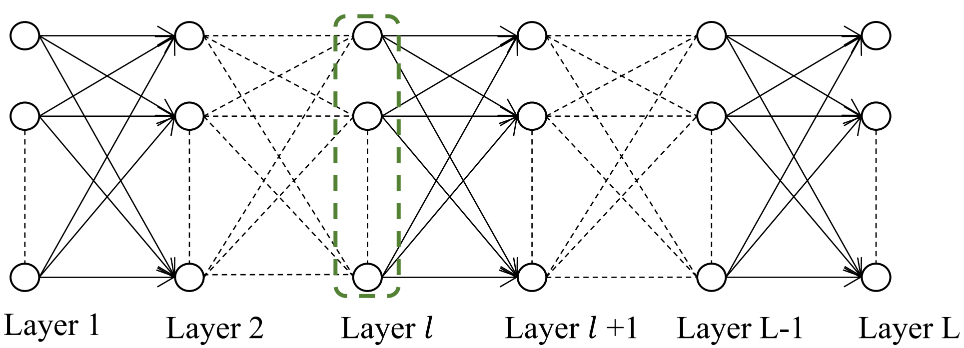
Updating rule: $w_{ji}^l \leftarrow w_{ji}^l - \alpha \cdot \frac{\partial J}{\partial w_{ji}^l}$

Define δ: $\delta_i^l = \frac{\partial J}{\partial z_i^l}$

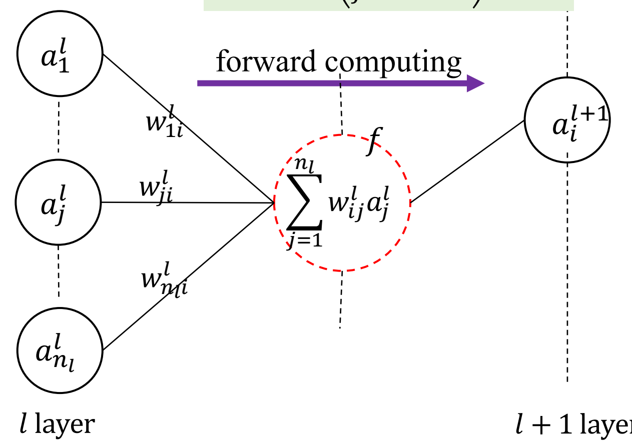
Find the relation: $\frac{\partial J}{\partial w_{ji}^l} = \delta_j^{l+1} \cdot a_i^l$

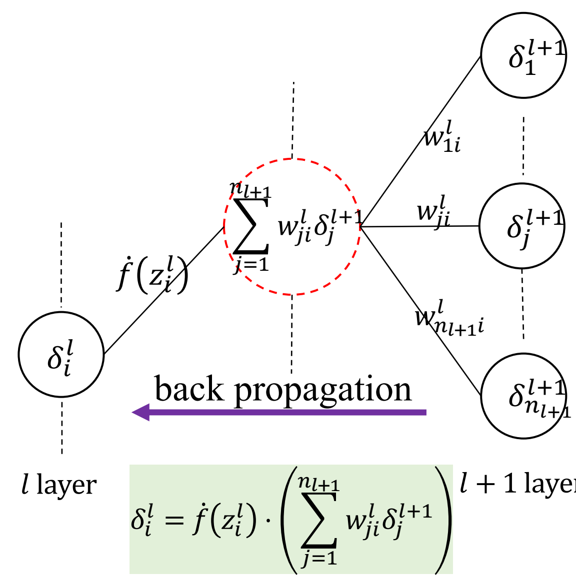
Back propagation: $\delta_i^L = \frac{\partial J}{\partial z_i^L} = (a_i^L - y_i^L) \cdot \dot{f}(z_i^L)$

$$\delta_i^l = \dot{f}(z_i^l) \cdot \left(\sum_{j=1}^{n_{l+1}} \delta_j^{l+1} \cdot w_{ji}^l \right)$$



$$a_i^{l+1} = f\left(\sum_{j=1}^{n_l} w_{ij}^l a_j^l\right)$$

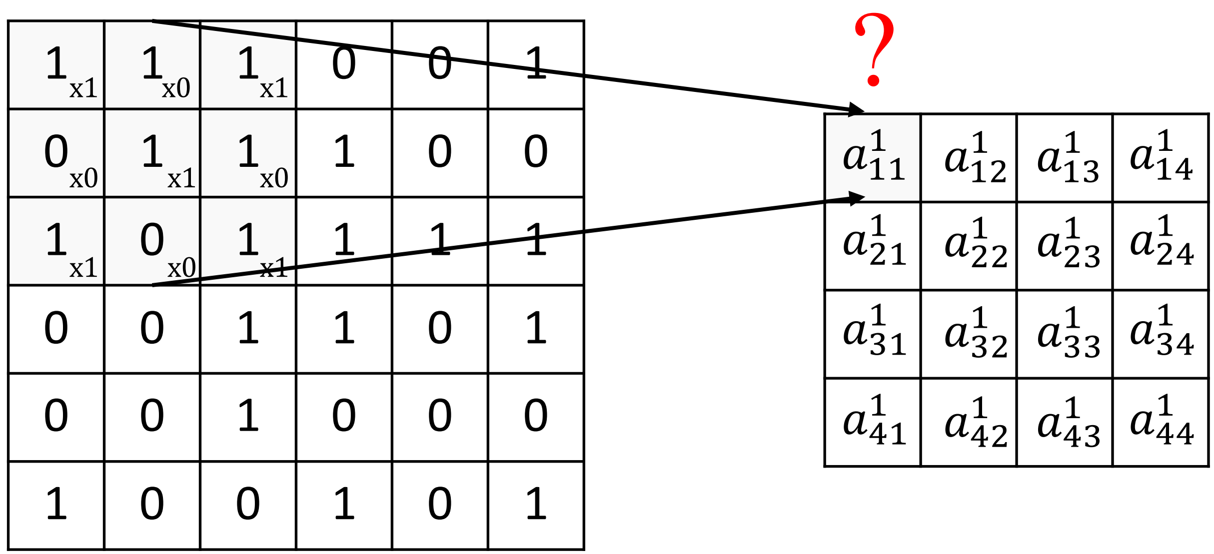




Convolutional Neural Network

Classical Architecture – Conv Example

$$
\begin{cases}
z_{11}^{1} = 1\times1 + 1\times0 + 1\times1 + 0\times0 + 1\times1 + 1\times0 + 1\times1 + 0\times0 + 1\times1 \\
\quad = 5 \\
a_{11}^{1} = f(z_{11}^{1}) = 5
\end{cases}
$$



Input | Feature Map

Convolutional Neural Network

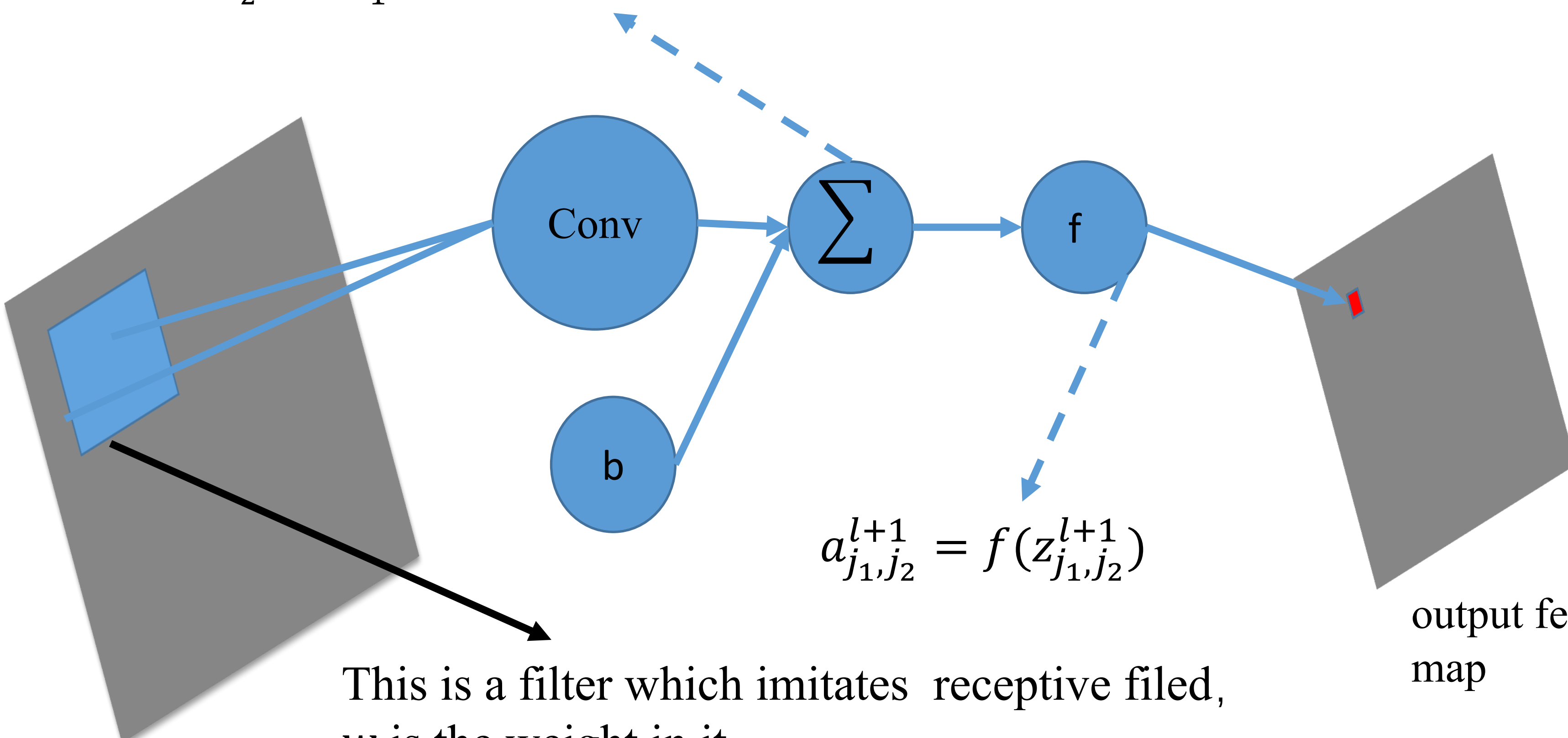
Gradient-Based Learning Algorithm

- Feedforward – Convolution (Conv)

Vector form

- $\begin{cases} z^{l+1} = W^{l} * a^{l} + b \\ a^{l+1} = f\left(z^{l+1}\right) \end{cases}$

$$z_{j_1,j_2}^{l+1} = \sum_{k_2=1}^{n_{k_2}} \sum_{k_1=1}^{n_{k_1}} a_{j_1+k_1-1,j_2+k_2-1}^{l} * w_{k_1,k_2}^{l} \quad + b$$



$$a_{j_1,j_2}^{l+1} = f(z_{j_1,j_2}^{l+1})$$

This is a filter which imitates receptive filed, *w* is the weight in it

output feature map

Convolutional Neural Network

Classical Architecture - Example

Max Pooing

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

Input

5	3	4	3
2	4	3	4
3	3	4	2
2	3	2	4

Feature Map

5	a_{12}^2
a_{21}^2	a_{22}^2

Pooled Feature Map

Convolutional Neural Network

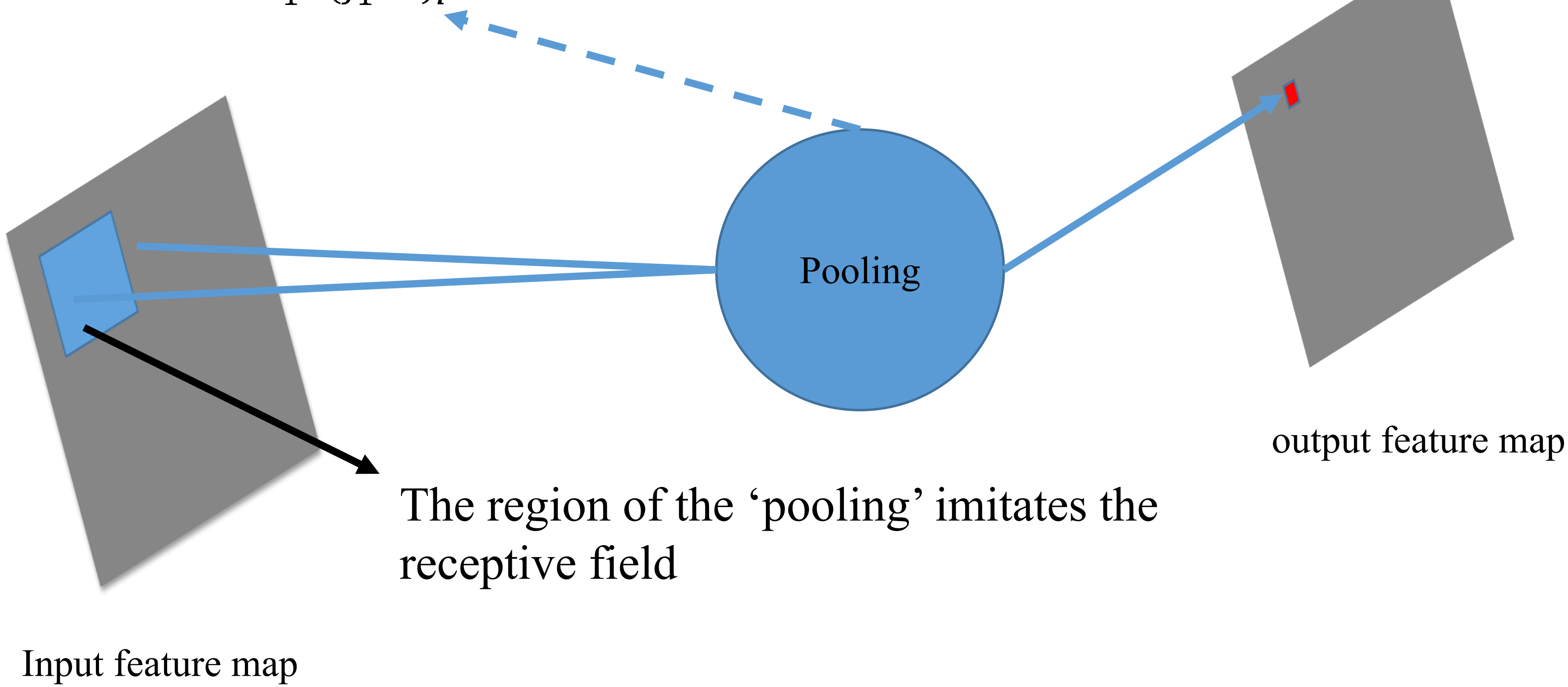
Gradient-Based Learning Algorithm

- Feedforward --Mean Pooling

$$z_{j_1,j_2}^{l+1} = \left(\frac{1}{p^2}\right) * \sum_{i_1=(j_1-1)p+1}^{j_1 p} \sum_{i_2=(j_2-1)p+1}^{j_2 p} a_{i_1,i_2}^{l}$$

Vector form

- $\begin{cases} z^{l+1} = downSample(a^l) \\ a^{l+1} = z^{l+1} \end{cases}$



Convolutional Neural Network

Gradient-Based Learning Algorithm

- Feedforward – vector form

● Convolution Layer

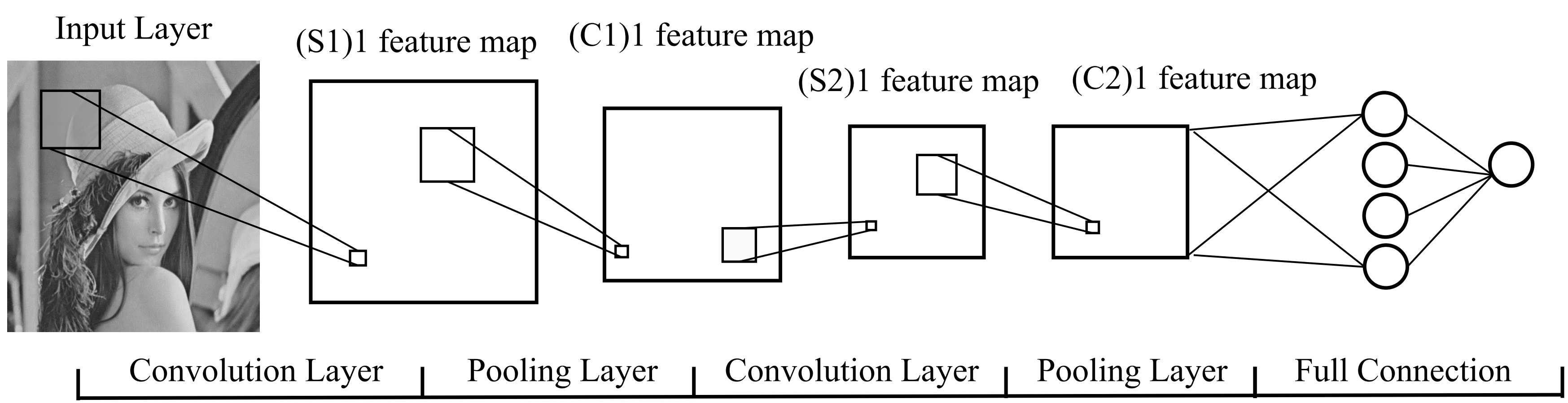
$$\begin{cases} z^{l+1} = W^l * a^l \\ a^{l+1} = f(z^{l+1}) \end{cases}$$

● Pooling Layer

$$\begin{cases} z^{l+1} = downSample(a^l) \\ a^{l+1} = z^{l+1} \end{cases}$$

● Full Connection Layer

$$\begin{cases} z^{l+1} = W^l a^l \\ a^{l+1} = f(z^{l+1}) \end{cases}$$



Convolutional Neural Network

Gradient-Based Learning Algorithm

- Backpropagation

Through BP :

$\delta^l = \frac{\partial J}{\partial z^l}$ (J is cost function of the convolution neural network)

$\delta^l = \frac{\partial J}{\partial z^{l+1}} \frac{\partial z^{l+1}}{\partial z^l} = \delta^{l+1} w^l \frac{\partial a^l}{\partial z^l} = \delta^{l+1} w^l f'(z^l)$

Which elements in the $(L+1)_{th}$ layer are related to the element z in the L_{th} layer ?

Convolutional Neural Network

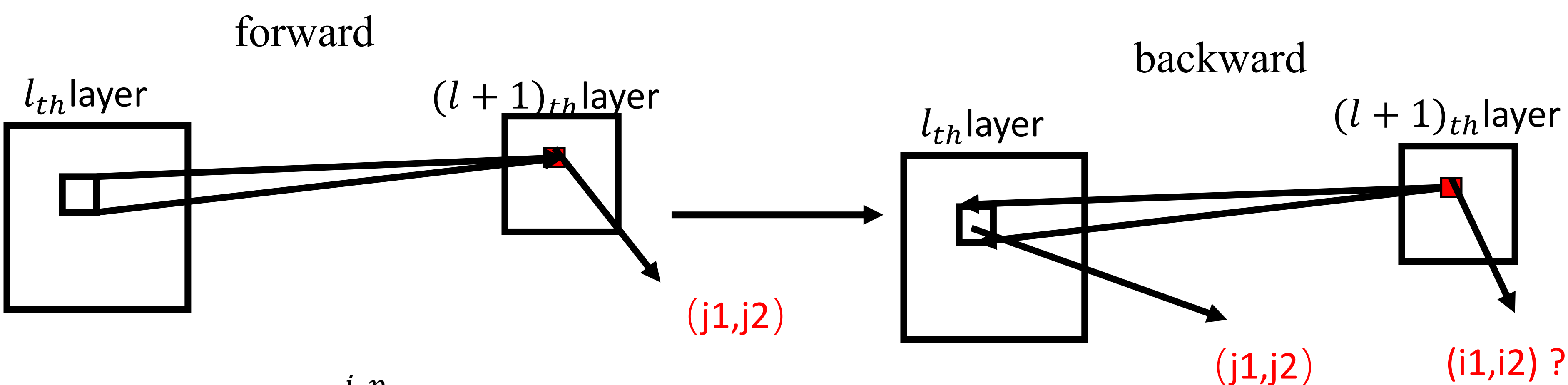
Gradient-Based Learning Algorithm

- Backpropagation
  - *NL-th* layer is output layer
    - $\delta^{NL} = \frac{\partial J}{\partial z^{NL}}$, where J is the cost function
  - *l*-th layer is a Full Connection Layer
    - $\begin{cases} \delta^{l} = \left((W^{l})^{T}\delta^{l+1}\right) \cdot f'(z^{l}) \\ \frac{\partial J}{\partial W^{l}} = \delta^{l+1}(a^{l})^{T} \end{cases}$

Convolutional Neural Network

Gradient-Based Learning Algorithm

- Backpropagation
- *l*-th layer is a **Convolution Layer,** *l*-th layer is pooling



$$z_{j_1,j_2}^{l+1} = \left(\frac{1}{\mathrm{p}^2}\right) * \sum_{i_1=(j_1-1)p+1}^{j_1 p} \sum_{i_2=(j_2-1)p+1}^{j_2 p} a_{i_1,i_2}^{l}$$

$$a_{j_1,j_2}^{l+1} = z_{j_1,j_2}^{l+1}$$

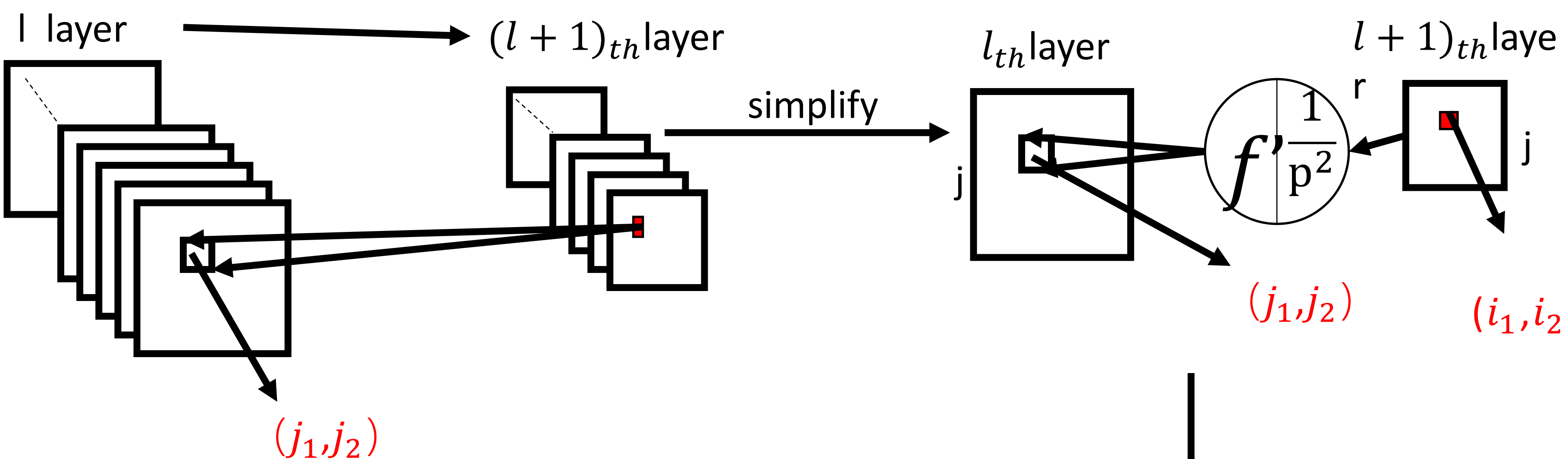
$i_1 = j_1/\mathrm{p} + 1$, $i_2 = j_2$ / p + 1; ('/'denotes 'divided with no remainder')

$$\delta_{j_1,j_2}^{l} = \frac{\partial J}{\partial z_{i_1,i_2}^{l+1}} \frac{\partial z_{i_1,i_2}^{l+1}}{\partial z_{j_1,j_2}^{l}} = \delta_{i_1,i_2}^{l+1} * \frac{1}{\mathrm{p}^2} * f'(z_{j_1,j_2}^{l})$$

Convolutional Neural Network

Gradient-Based Learning Algorithm

- Backpropagation
- *l*-th layer is a **Convolution Layer,** *l*-th layer is pooling



$$\delta^{l}_{j_1,j_2,j} = \frac{\partial J}{\partial z^{l+1}_{i_1,i_2,j}} \frac{\partial z^{l+1}_{i_1,i_2,j}}{\partial z^{l}_{j_1,j_2,j}} = \delta^{l+1}_{i_1,i_2,j} * \frac{1}{p^2} * f'(z^{l}_{j_1,j_2}$$

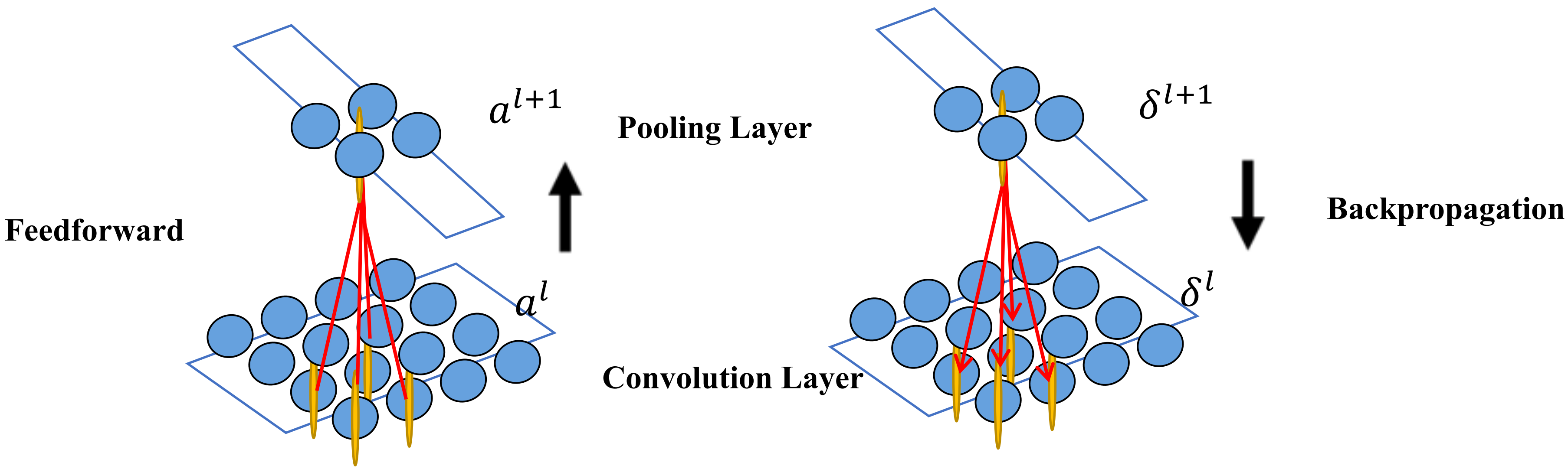
δ^{l}_{j} = 1/(p^2)*up (δ^{l+1}_{j})*$f'(z^{l}_{j})$ (matrix form)

Convolutional Neural Network

Gradient-Based Learning Algorithm

- Backpropagation
  - l-th layer is a **Convolution Layer**
    - $\delta^l = upSample(\delta^{l+1}) \cdot f'(z^l)$

$$\boxed{\begin{cases} z^{l+1} = downSample(a^l) \\ a^{l+1} = z^{l+1} \end{cases}}$$



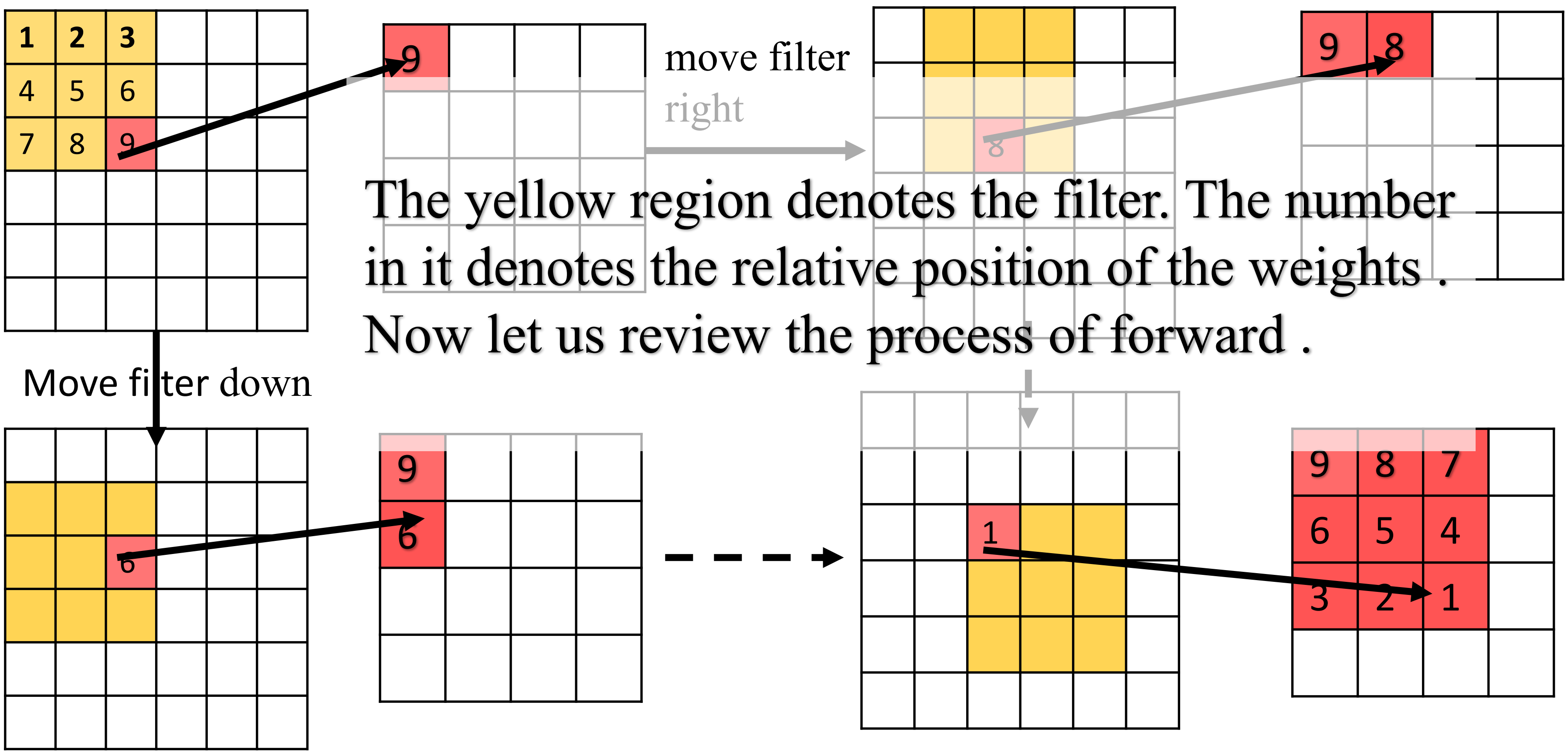
Convolution Layer is followed by a Pooling Layer

Convolutional Neural Network

Gradient-Based Learning Algorithm

➢ Backpropagation

- l-th layer is a **Pooling Layer,** $l+1$-th layer is a Conv layer



Convolutional Neural Network

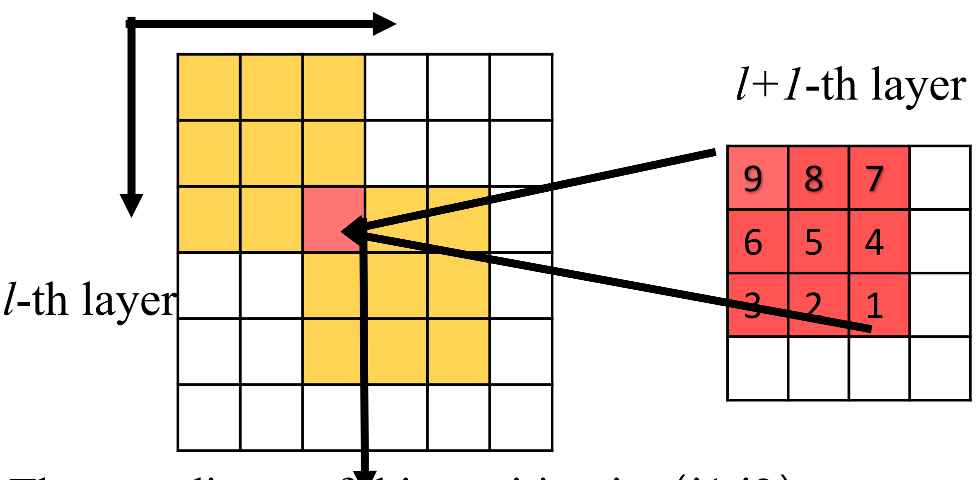
Gradient-Based Learning Algorithm

- Backpropagation
  - *l*-th layer is a **Pooling Layer,** *l+1*-th layer is a Conv layer

Hence: $\delta^{l}_{j_1,j_2} = \sum_{k_1=\mathrm{n}_{k_1}}^{1} \sum_{k_2=n_{k_2}}^{1} \boxed{\frac{\partial J}{\partial z^{l+1}_{j_1-k_1+1,j_2-k_2+1}}} \boxed{\frac{\partial z^{l+1}_{j_1-k_1+1,j_2-k_2+1}}{\partial z^{l}_{j_1,j_2}}}$

$\delta^{l}_{j_1,j_2} = \sum_{k_1=n_{k_1}}^{1} \sum_{k_2=n_{k_2}}^{1} \boxed{\delta^{l+1}_{j_1-k_1+1,j_2-k_2+1}} \boxed{w^{l}_{k_1,k_2} * f'(z^{l}_{j_1,j_2})}$

$f'(z^{l}_{j_1,j_2})=1$, so

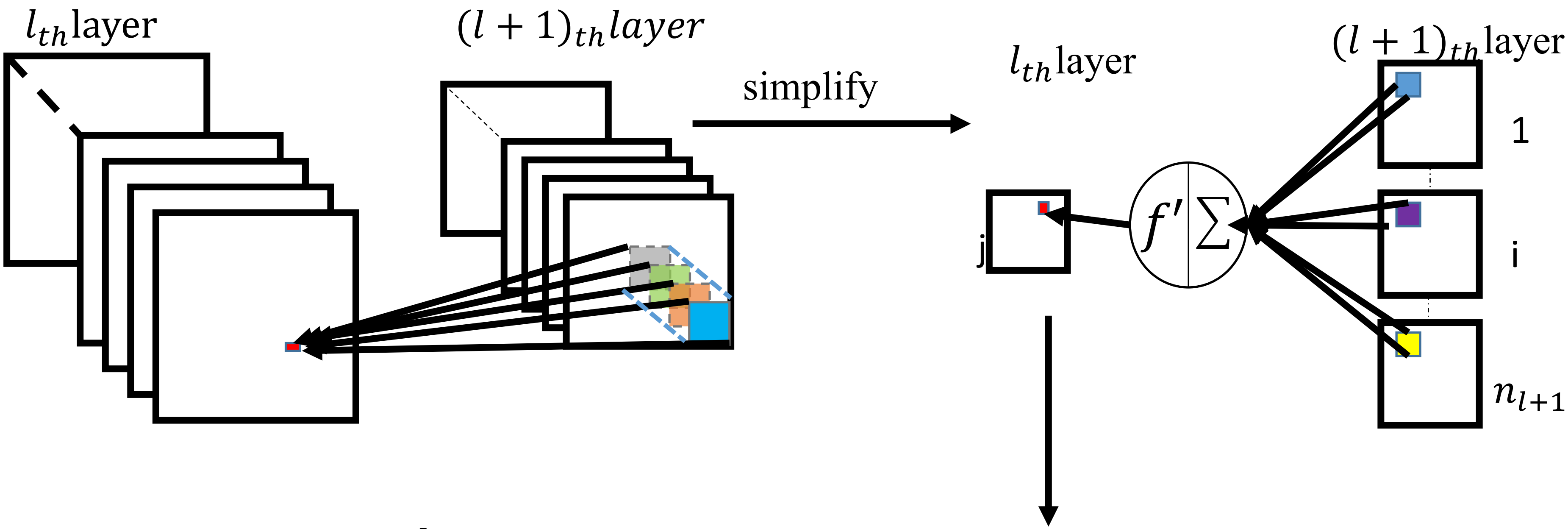


The coordinate of this position is (j1,j2)

Convolutional Neural Network

Gradient-Based Learning Algorithm

- Backpropagation
  - *l*-th layer is a **Pooling Layer,** *l+1*-th layer is a Conv layer



$$\delta^{l}_{j_1,j_2,j} = \sum_{k_1=n_{k_1}}^{1} \sum_{k_2=n_{k_2}}^{1} \delta^{l+1}_{j_1-k_1+1,j_2-k_2+1,j}\, w^{l}_{k_1,k_2,j,i}$$

$$\delta^{l}_{j} = \delta^{l+1}_{i} \times rot180(w^{l}_{j,i}) \text{ (matrix form)}$$

Convolutional Neural Network

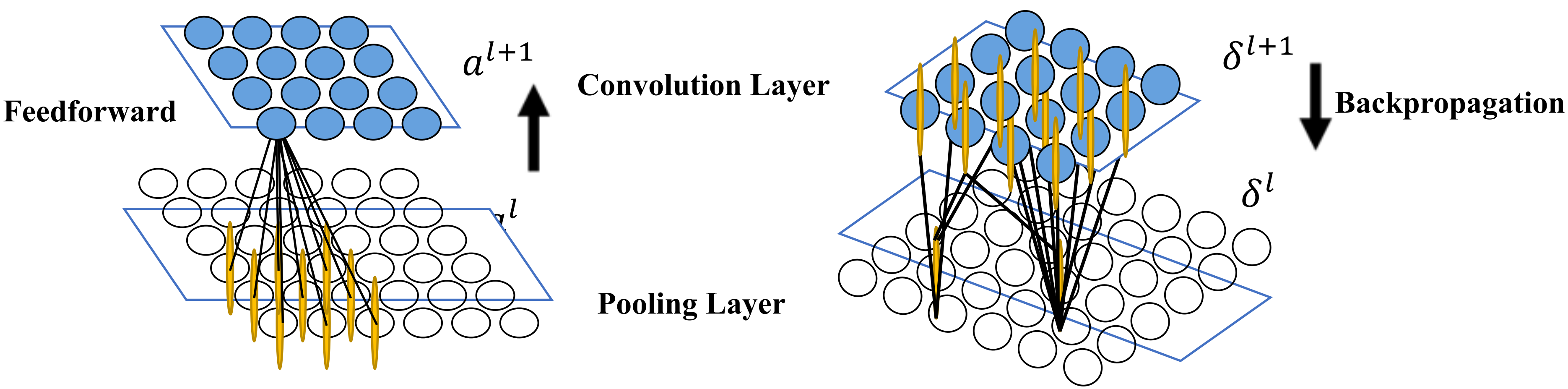
Gradient-Based Learning Algorithm

- Backpropagation

  - l-th layer is a **Pooling Layer**

    - $\delta^l = (rot180(\delta^{l+1}) * w^l)$
    - $\frac{\partial J}{\partial W^l} = a^l * \delta^{l+1}$

$$\begin{cases} z^{l+1} = W^l * a^l \\ a^{l+1} = f(z^{l+1}) \end{cases}$$



Pooling Layer is followed by a Convolution Layer

Neural Networks

- Brief review
- Convolutional Neural Network

  Classical Architecture

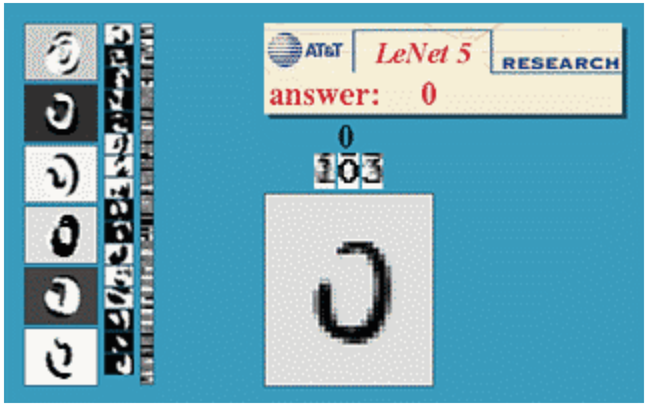
  Gradient-Based Learning Algorithm

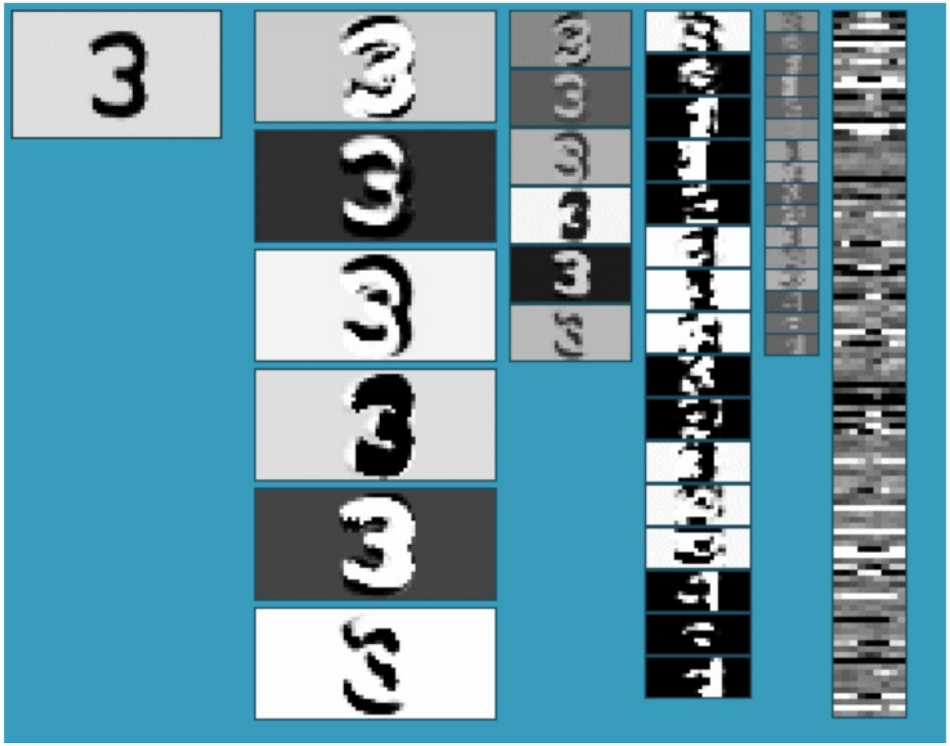
  Achievements

Convolutional Neural Network

Achievements and Applications

- Handwriting Recognition
  - LeNet LeCun



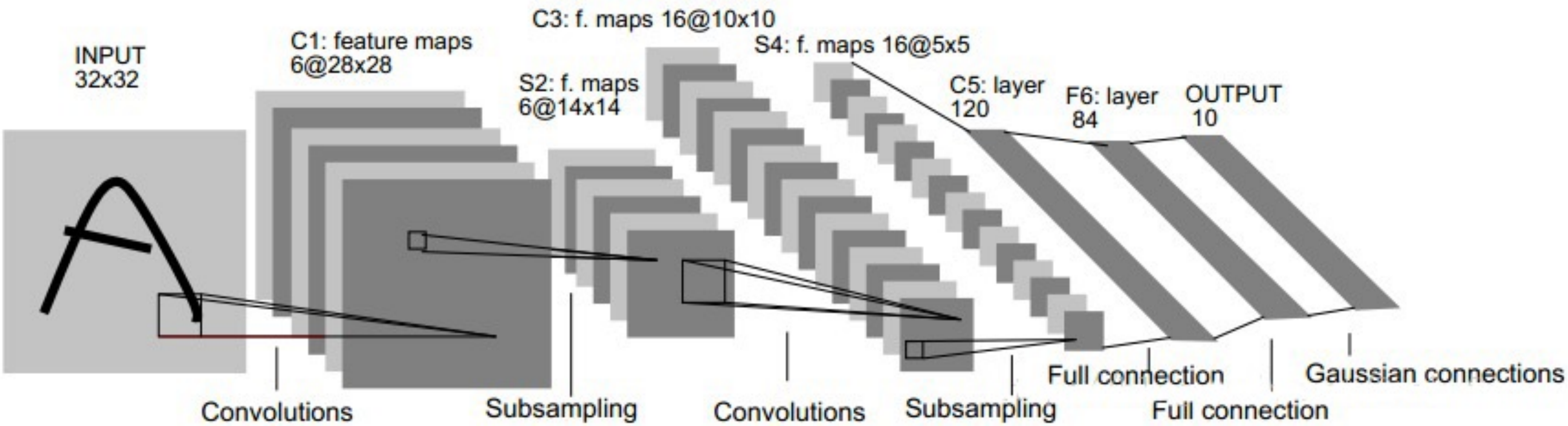


http://yann.lecun.com/exdb/len

Convolutional Neural Network

Achievements and Applications

LeNet

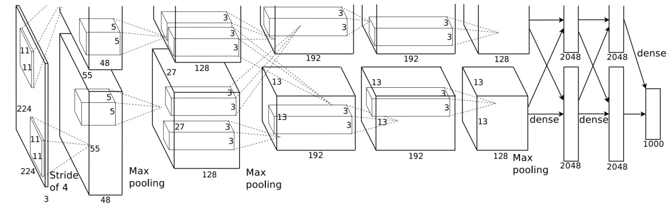


Kernel size	5*5 (conv)	2*2 (pooling)	5*5 (conv)	2*2 (pooling)

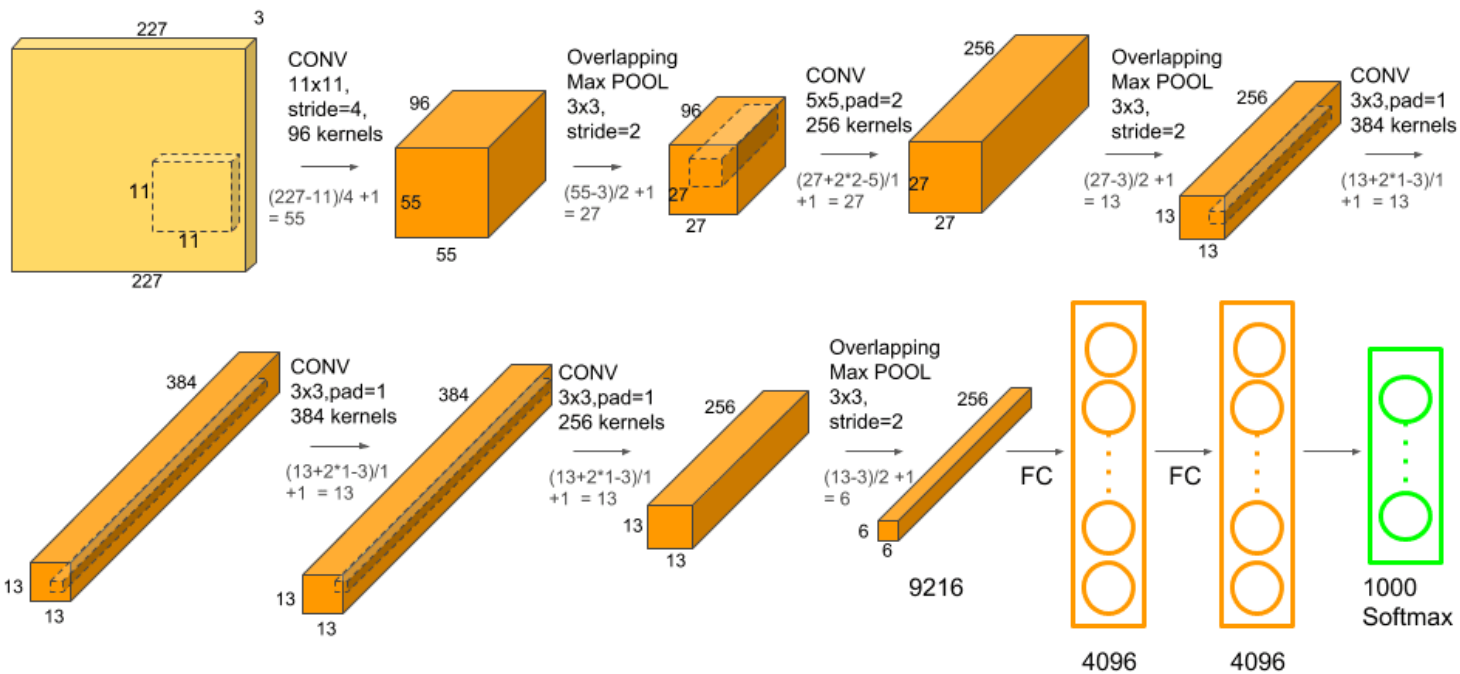
Convolutional Neural Network

Achievements and Applications

- AlexNet



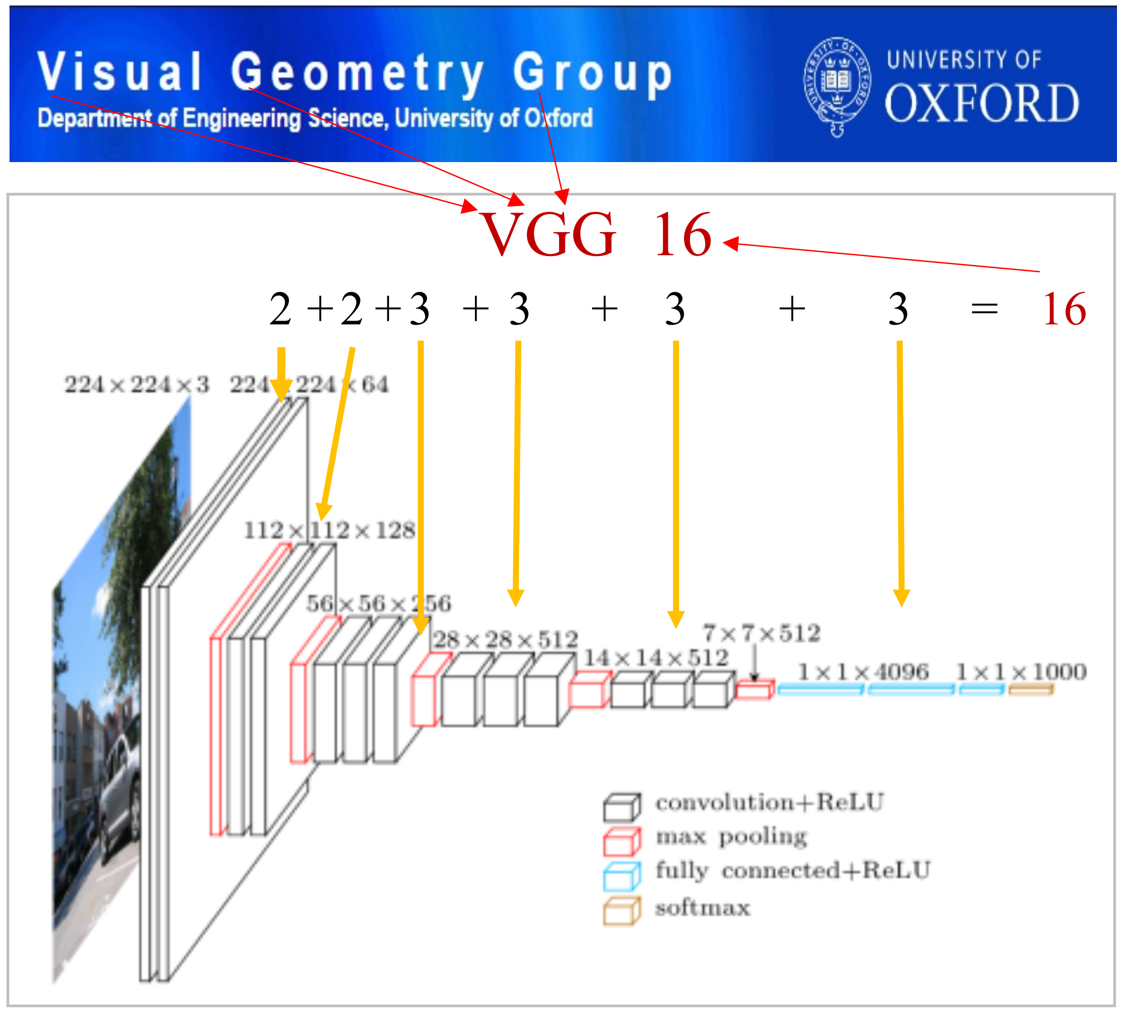
responsibilities
the layer-parts
mensional, and
4,896–43,264–



Convolutional Neural Network

Achievements

VGG Networks
- VGG11
- VGG13
- VGG16
- VGG19



Convolutional Neural Network

Achievements

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

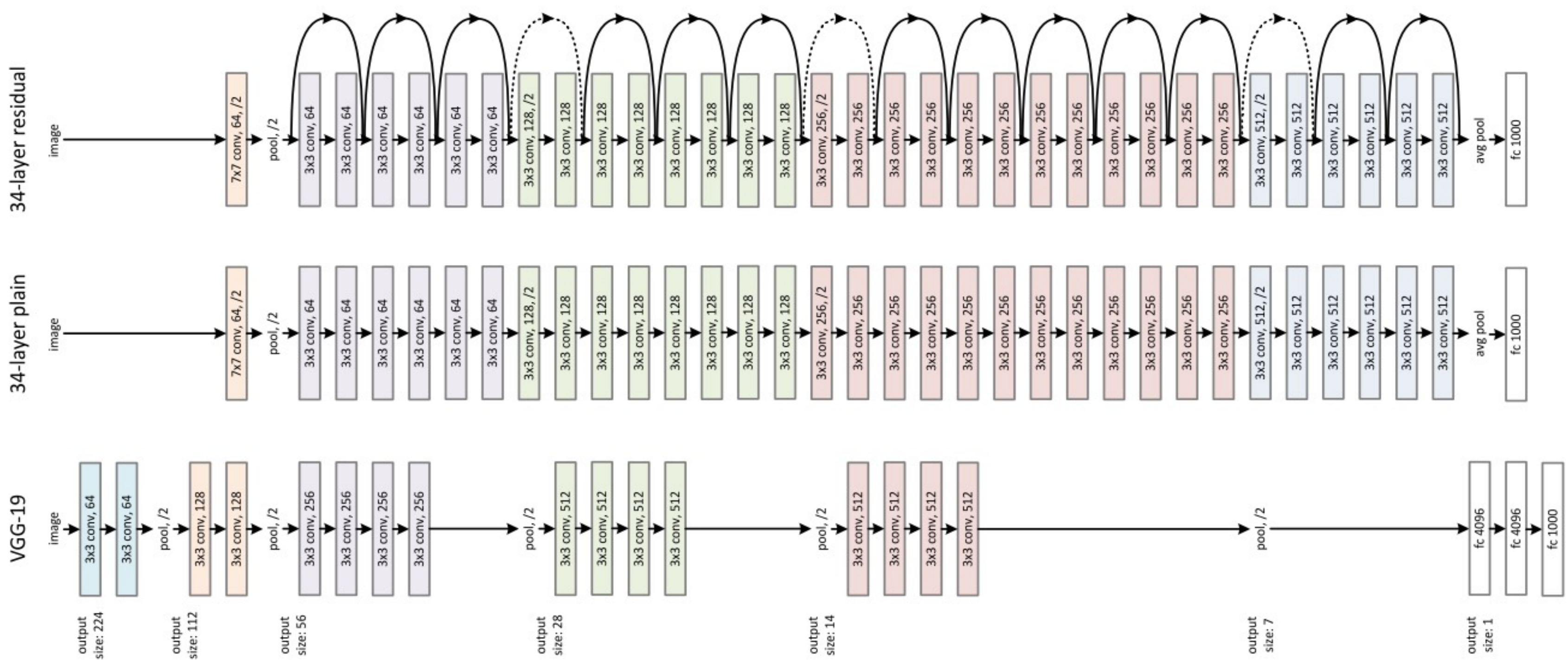
VGG16 - Features:

- 11-19 layers
- Same input, 3×224×224
- 3 fully-connected layers
- Softmax output
- 5 stages, maxpool in between
- (most)3×3 kernels
- Increasing kernel number

Convolutional Neural Network

Achievements and Applications

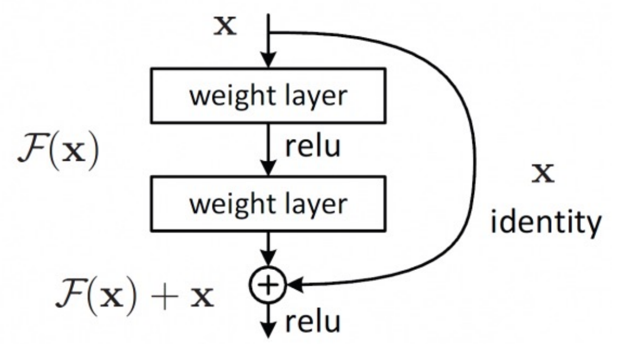
- ResNet (2015)



Convolutional Neural Network

Achievements and Applications

- ResNet

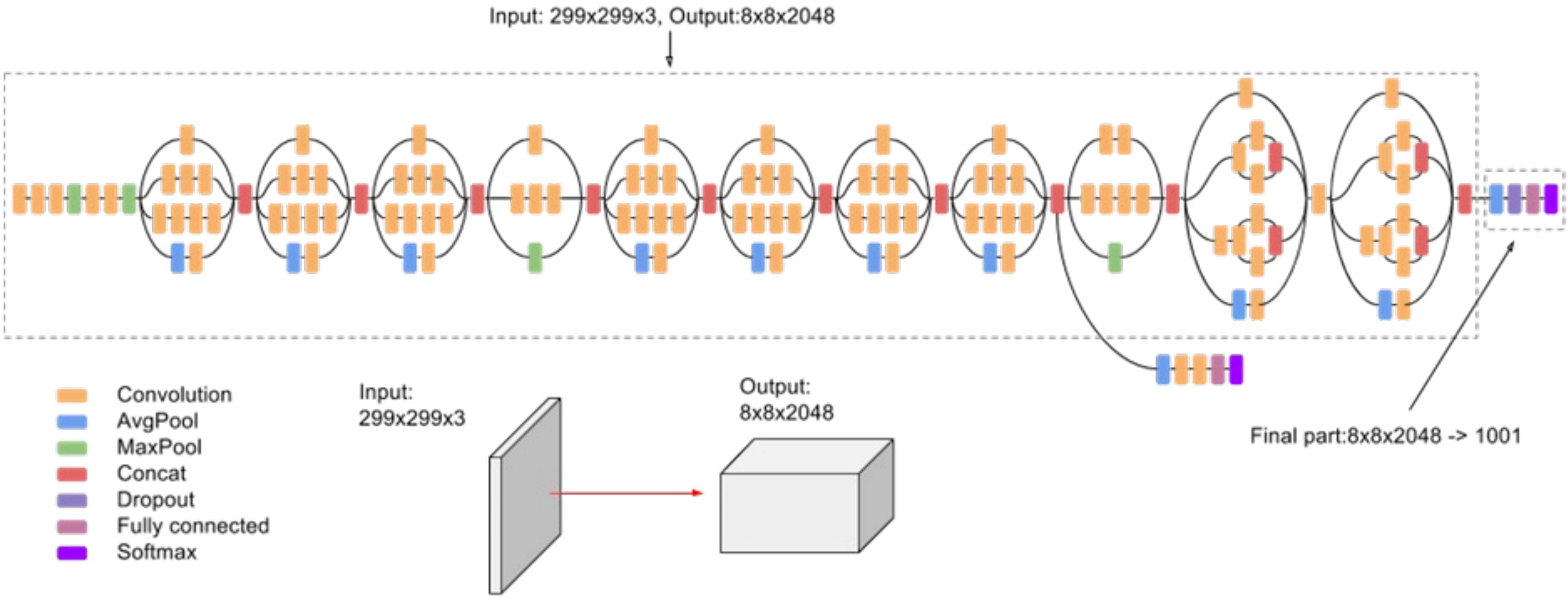


layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Convolutional Neural Network

Achievements and Applications

➢ Inception V3

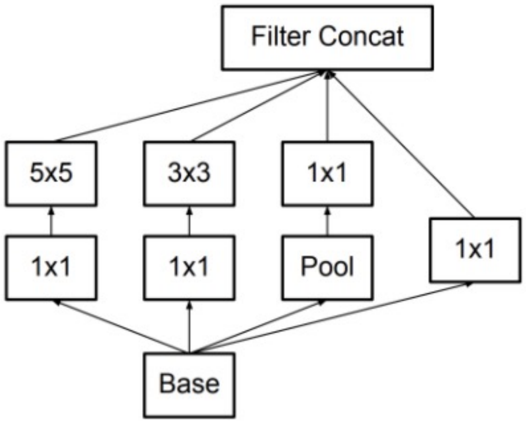


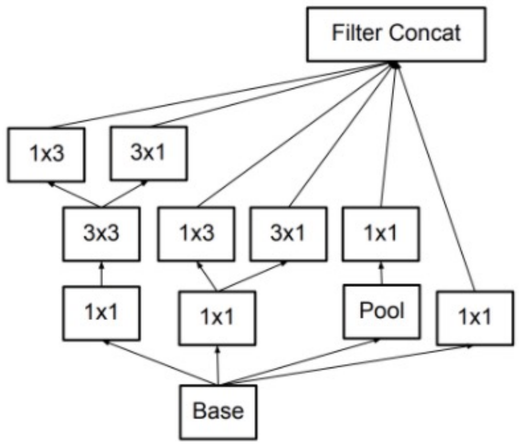
•42层深，但计算成本只比inception v1高2.5倍左右，而且比VGGNet高效得多.
•用卷积和池化并行的方法降低 Inception 模块的大小

Convolutional Neural Network

Achievements and Applications

- Inception V3

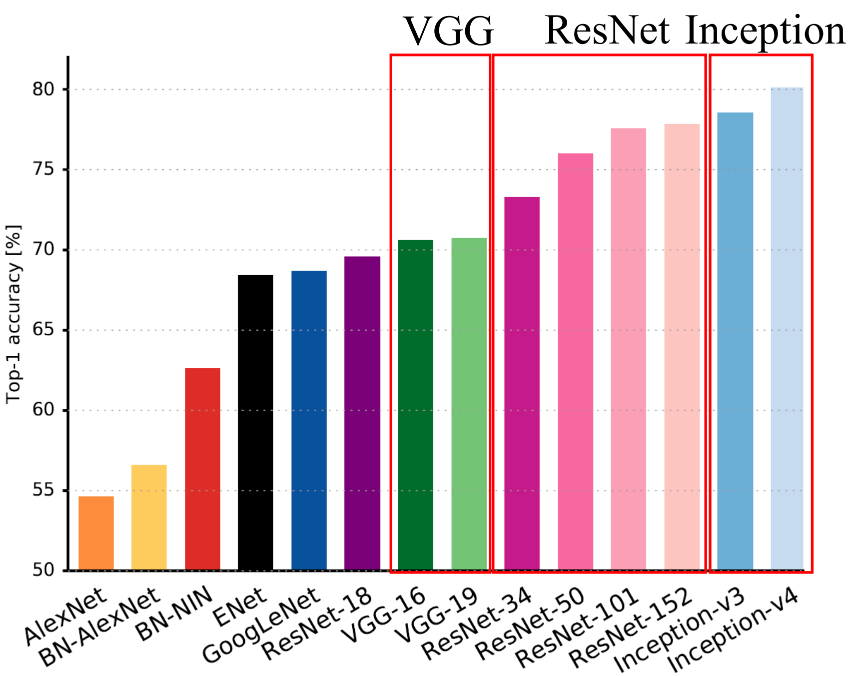




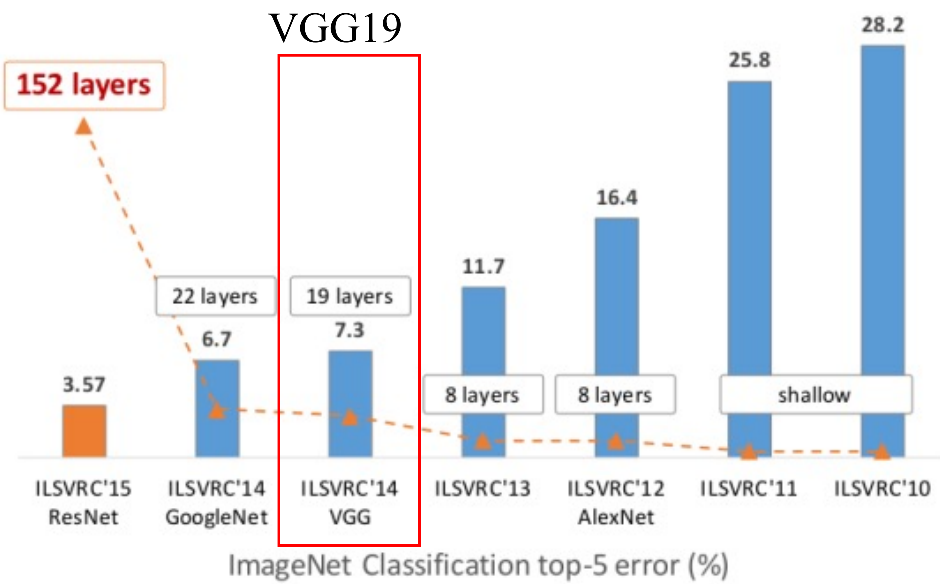
V1：分支卷积
V3：改为小卷积和非对称卷积

Convolutional Neural Network

Achievements -- ImageNet



Single-crop Top-1 validation accuracies on ImageNet



Reported Top-5 error on ImageNet

Convolutional Neural Network

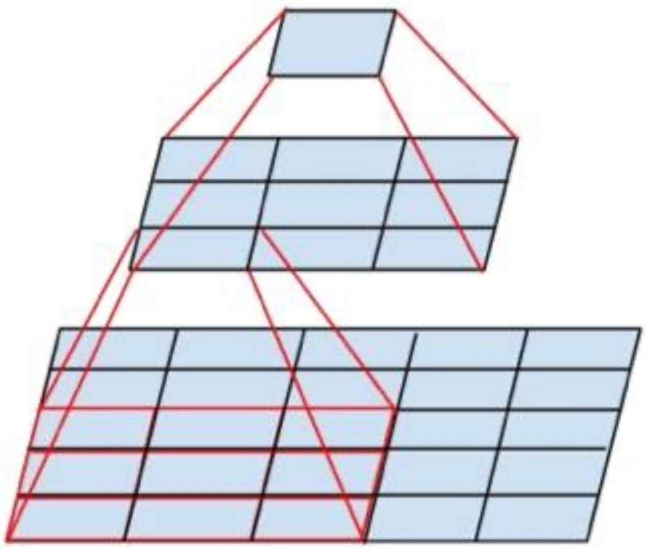
Thinking--Why that size of kernels?

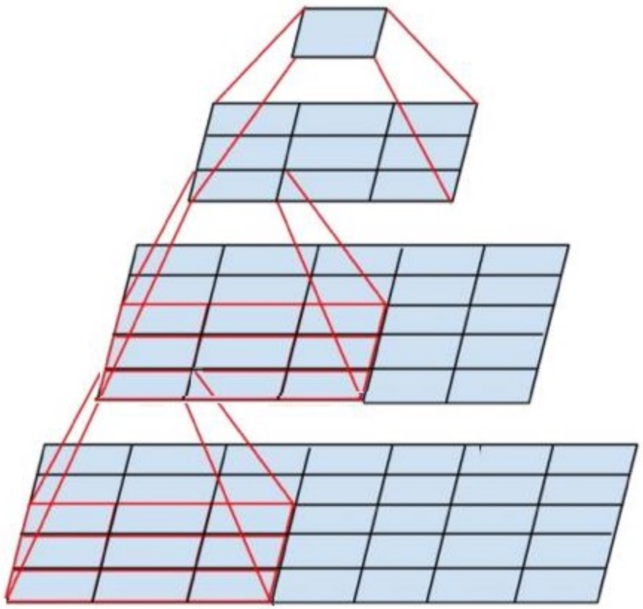
Network	Kernels Used	Deepth
Lenet	(5,5)	4
Alexnet	(11,11), (5,5), (3,3)	8
VGGs	(3,3)	11-19
ResNet	(3,3), (1,1)	
Inception V3	(1,3),(3,1),(3,3),(1,1)	

Convolutional Neural Network

Thinking– why 3×3 kernels?

a.) 3×3 kernels can simulate larger kernels like 5×5 or 7×7





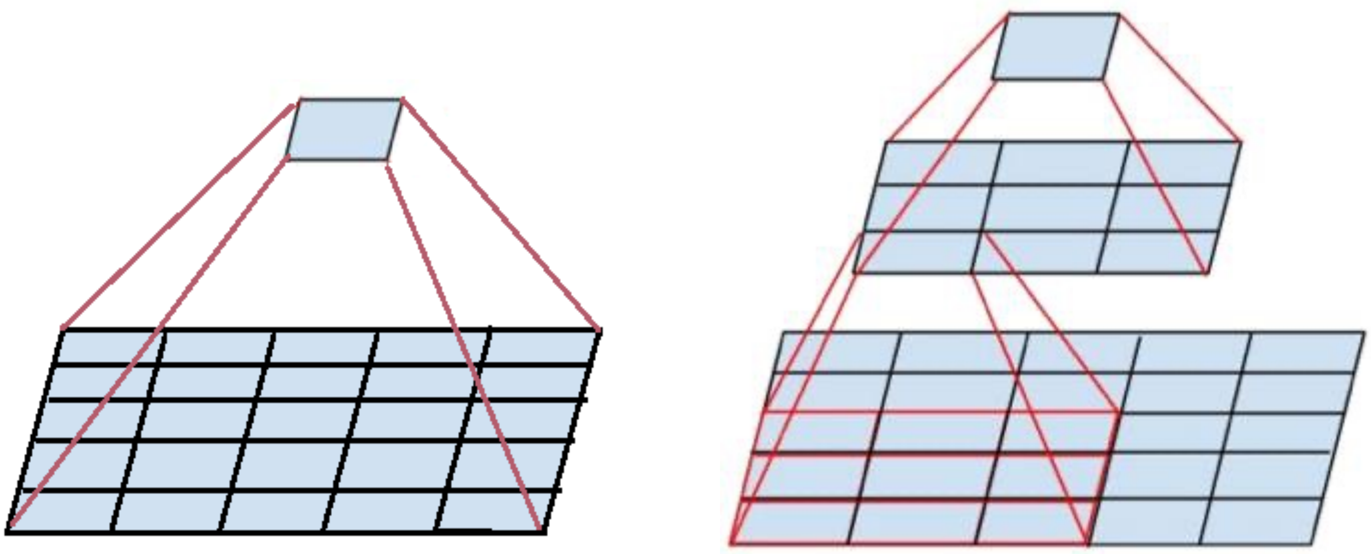
…

Convolutional Neural Network

- Thinking– why 3×3 kernels?

  a.) 3×3 kernels can simulate larger kernels like 5×5 or 7×7

  b.) by doing so, more nonlinearity is involved, thus more expressive ability



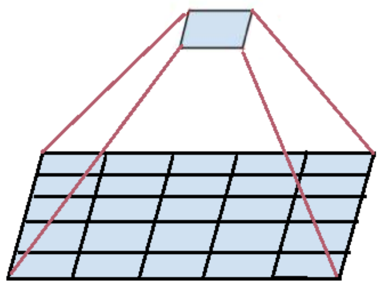
Convolutional Neural Network

Thinking– why 3×3 kernels?

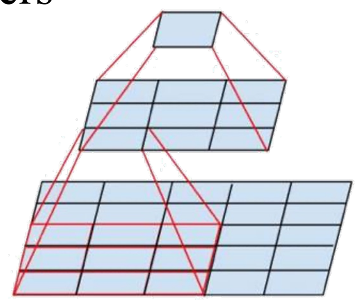
a.) 3×3 kernels can simulate larger kernels like 5×5 or 7×7

b.) by doing so, more nonlinearity is involved, thus more expressive ability

c.) 3×3 kernels can save a lot of parameters



\# of weights: 5×5 = 25



\# of weights: 2×(3×3) = 18

$\frac{18}{25} = 72\%$

General case:

5×5 ×input channel ×output channel

(3×3 ×input channel ×hidden channel) +(3×3 ×hidden channel ×output channel)