# Introduction to Convolutional Neural Networks

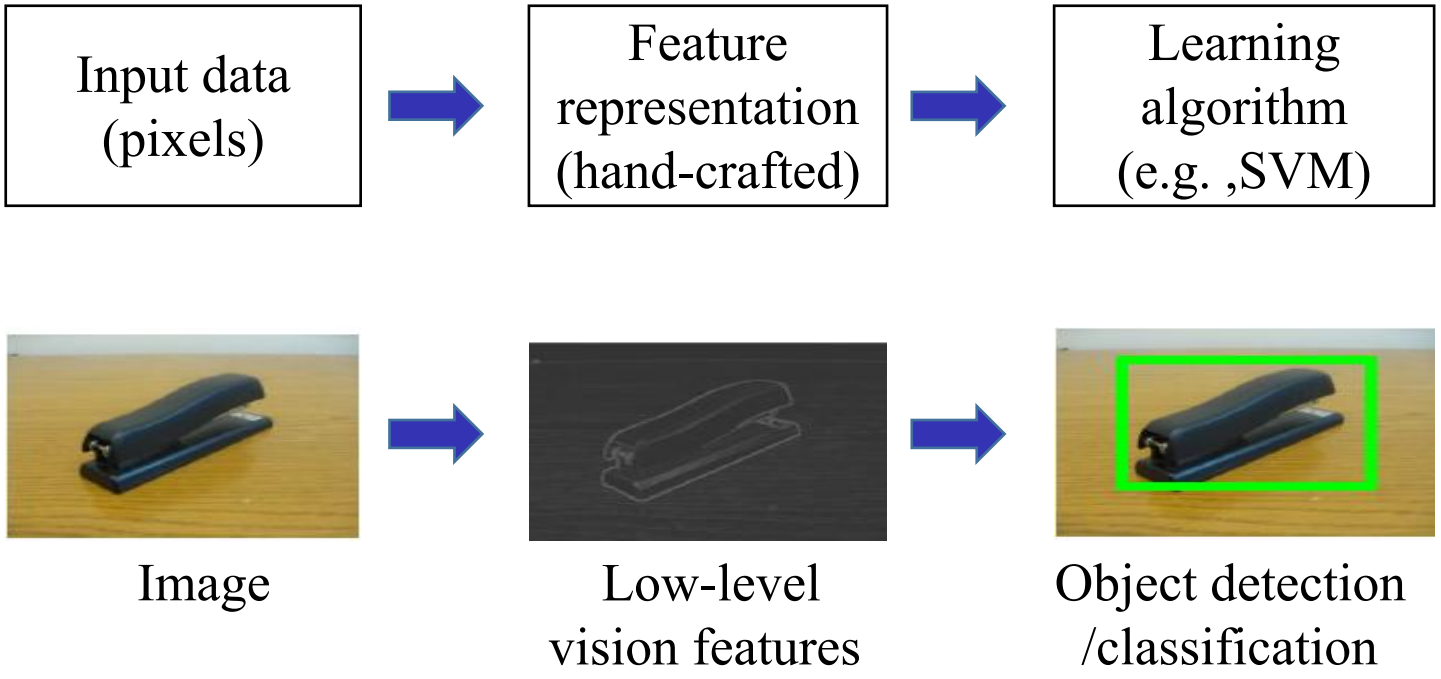
**Prof. Mingkui Tan**

SCUT Machine Intelligence Laboratory (SMIL)


# Contents

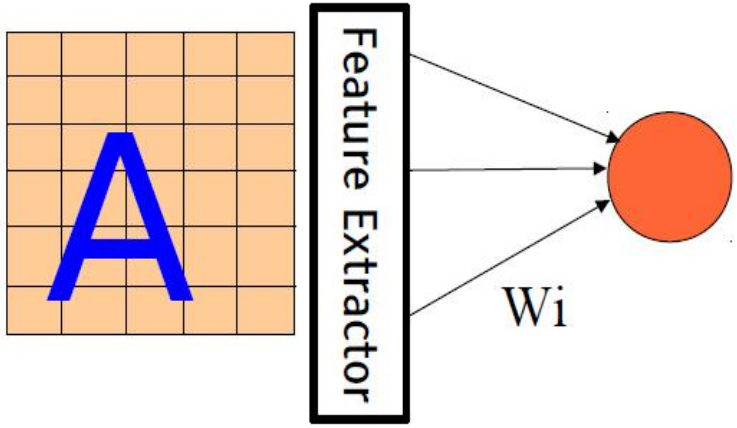
# Contents

# Traditional Recognition Methods



# Traditional Recognition Methods

$$y = \text{sign}\left(\sum_{i=1}^{N} \mathbf{W}_i F_i(\mathbf{X}) + \text{b}\right)$$

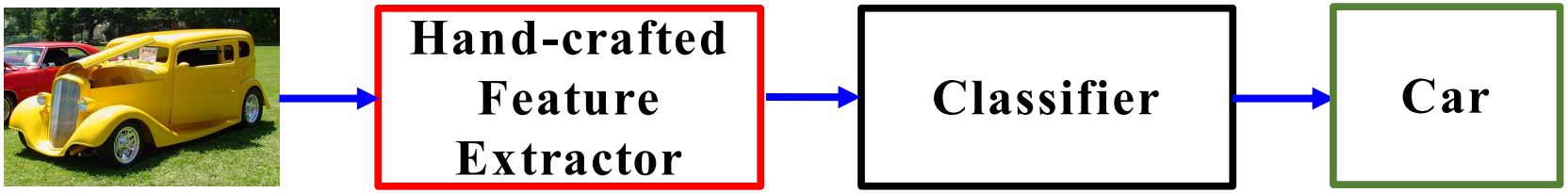



**Linear Classifier**

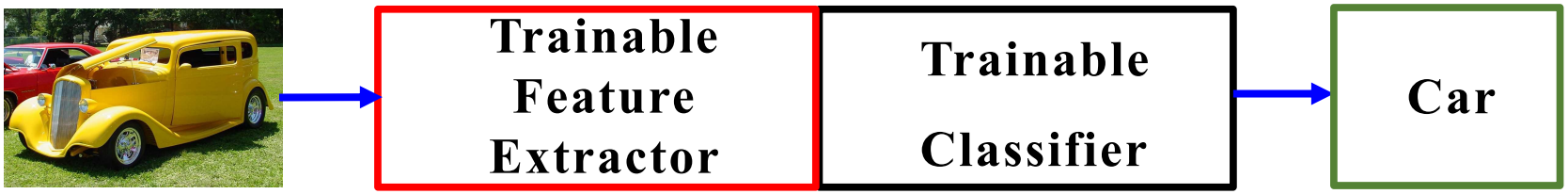
- **Linear classifier** is on top of a simple **feature extractor**
- Highly depend on the **quality of extracted feature**
- Designing a feature extractor requires **considerable efforts**

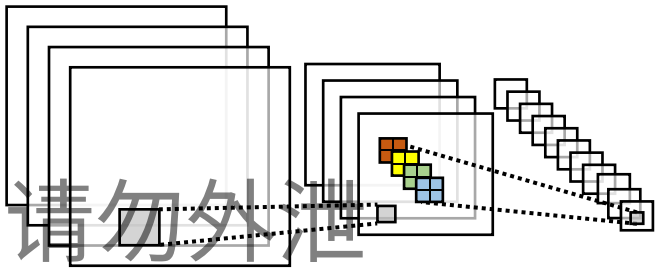
# Traditional Methods vs Deep Learning

- Traditional recognition with **hand-crafted feature**, e.g. SIFT and LBP feature

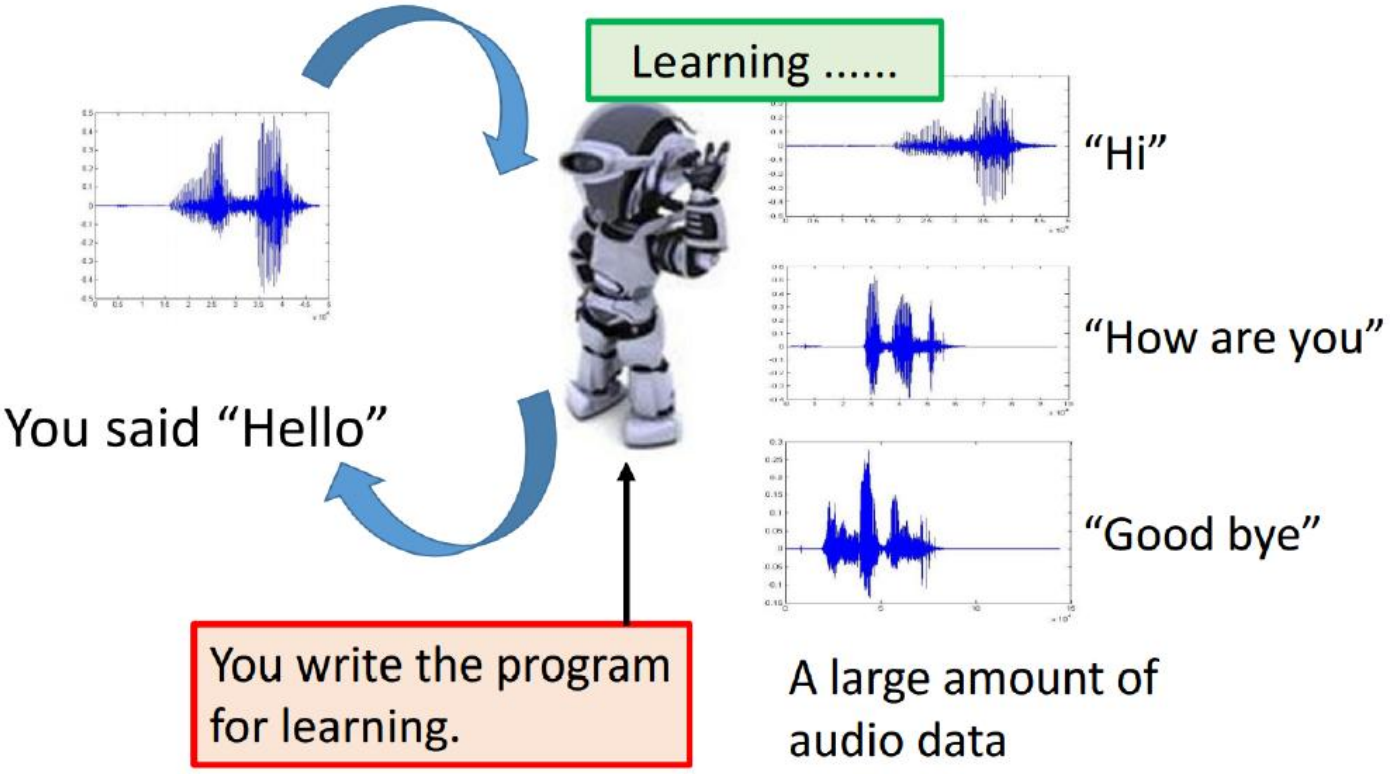



- Deep learning / Feature learning



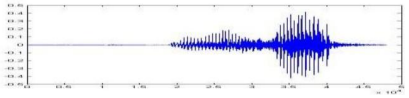
# What is Deep Learning?



# Deep Learning

**Looking for a Function $f$:**

- Speech Recognition

$f($  $)=$ "How are you"

- Image Recognition

$f($  $)=$ "Cat"

- Playing Go

$f($  $)=$ "5-5" (next move)

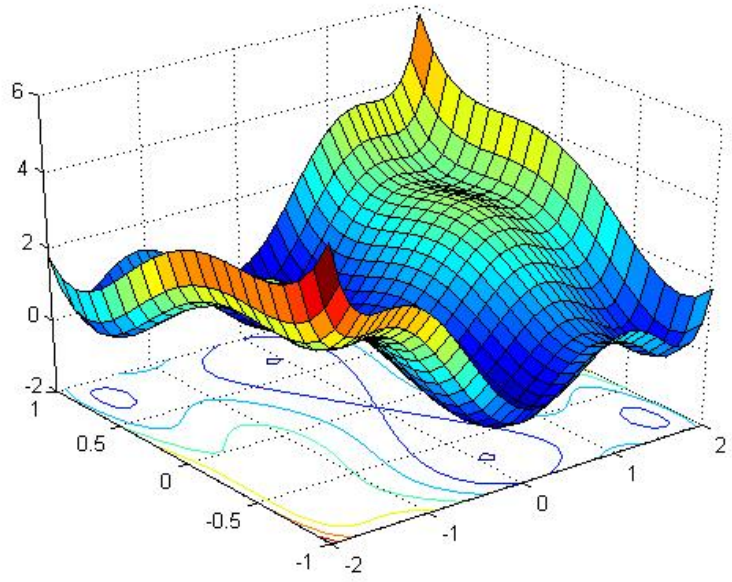
- Dialogue System

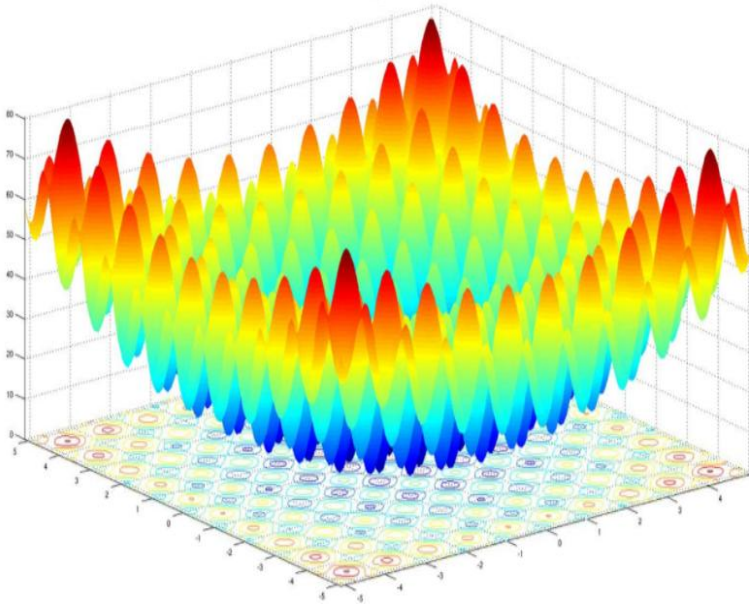
$f($ "Hi" $)=$ "Hello"

(what the user said) (system response)

# Deep Learning

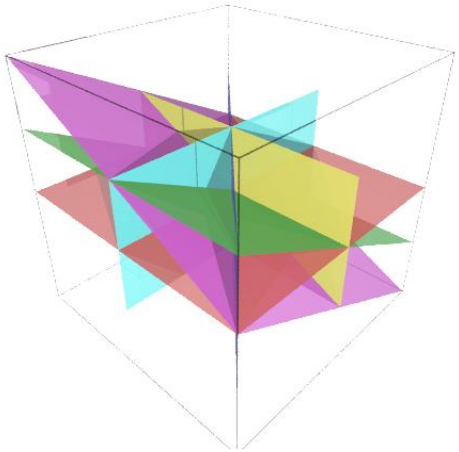
- Looking for a **Function** $f$:

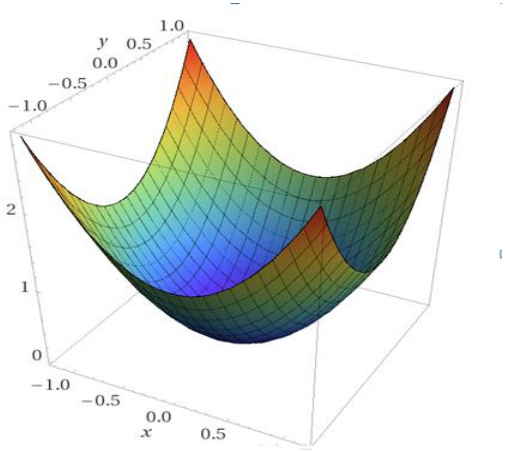


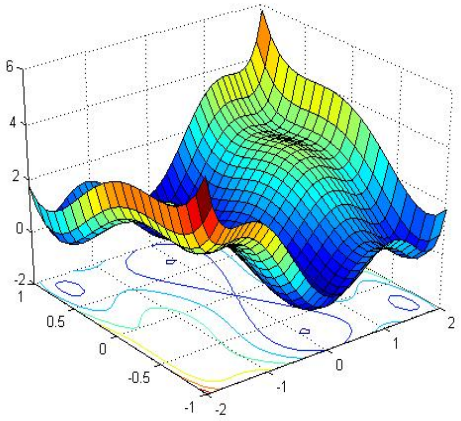



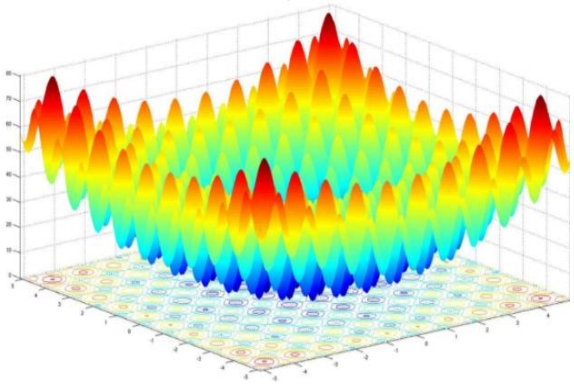
# Deep Learning

- What kind of functions can we choose?





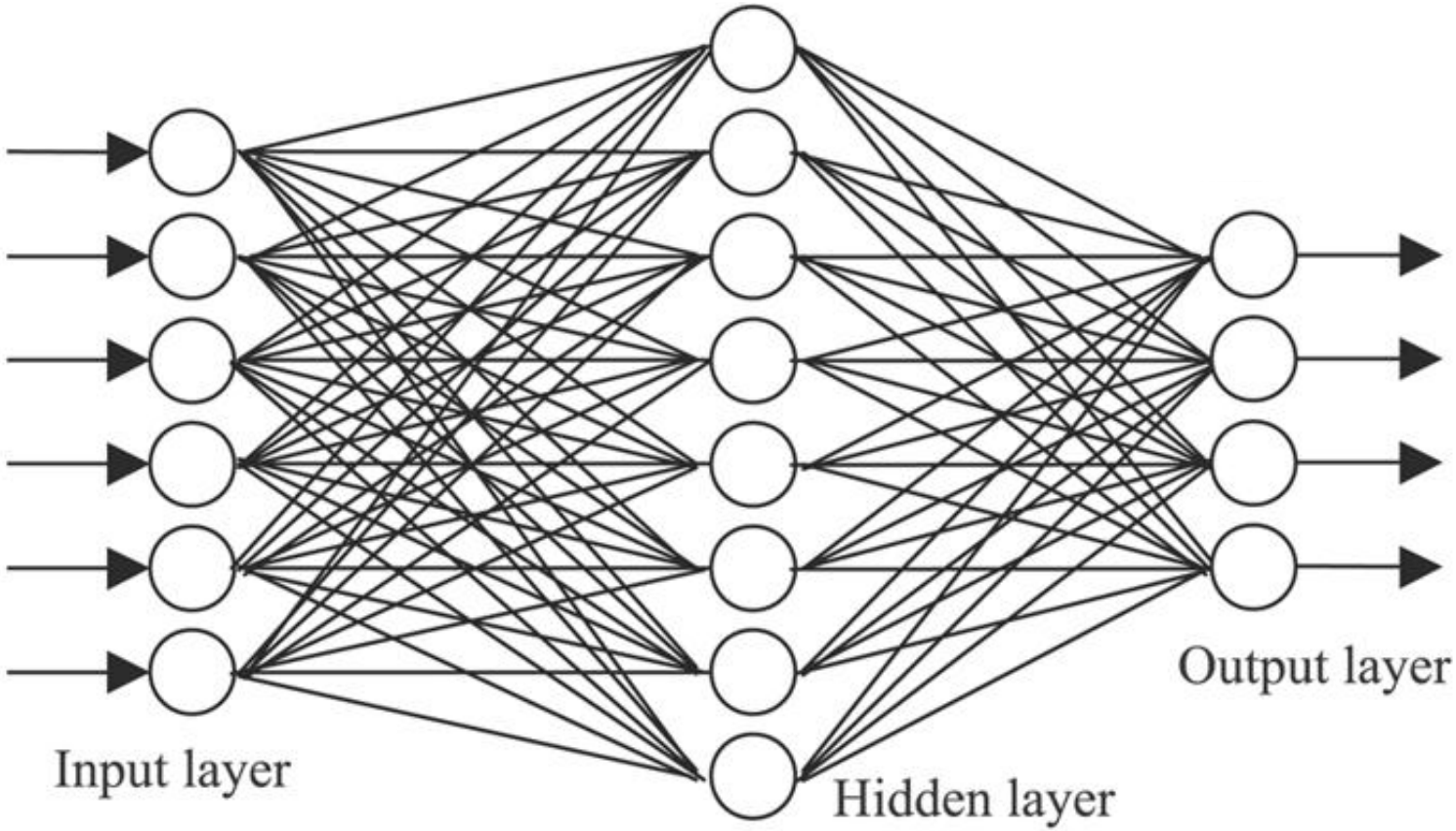



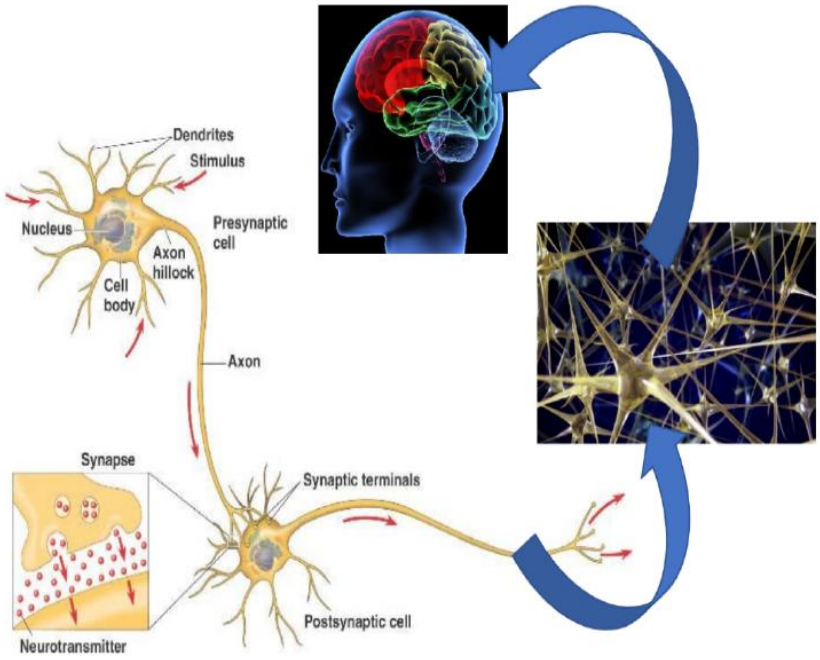
# Contents

# Neural Networks

- To approximate the function, we can build **neural networks**:

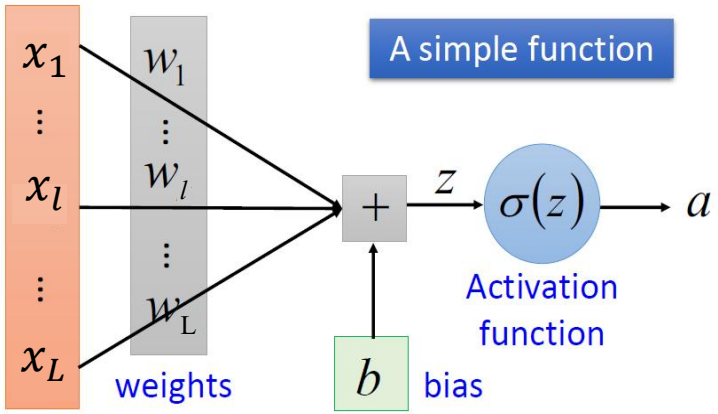


# Neurons in Human Brains

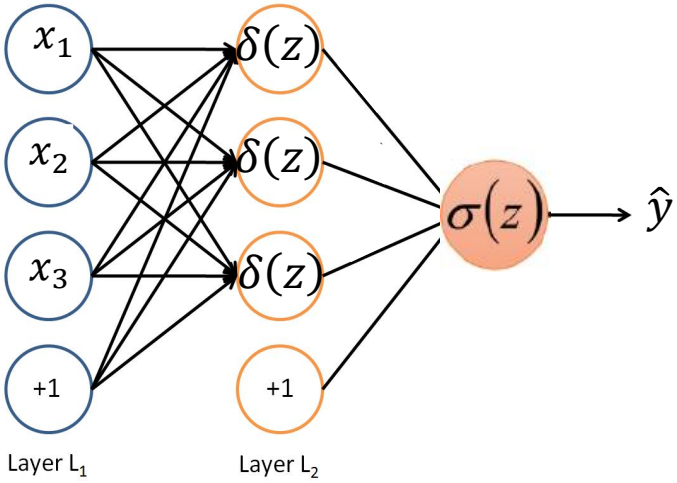



**Neuron**

$$z = x_1 w_1 + \cdots + x_l w_l + \cdots + x_L w_L + b$$



# Neural Networks

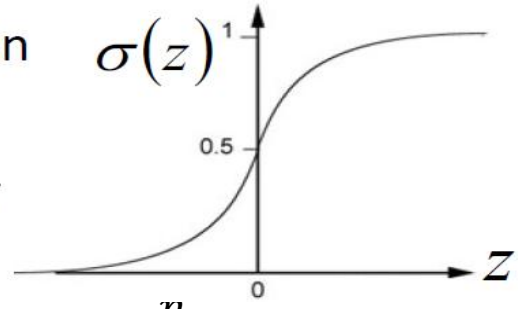



Non – Linear Activation Functions

$$\delta(z)=\delta(W^T X + b)$$

Sigmoid Function

$$\sigma(z)=\frac{1}{1+e^{-z}}$$



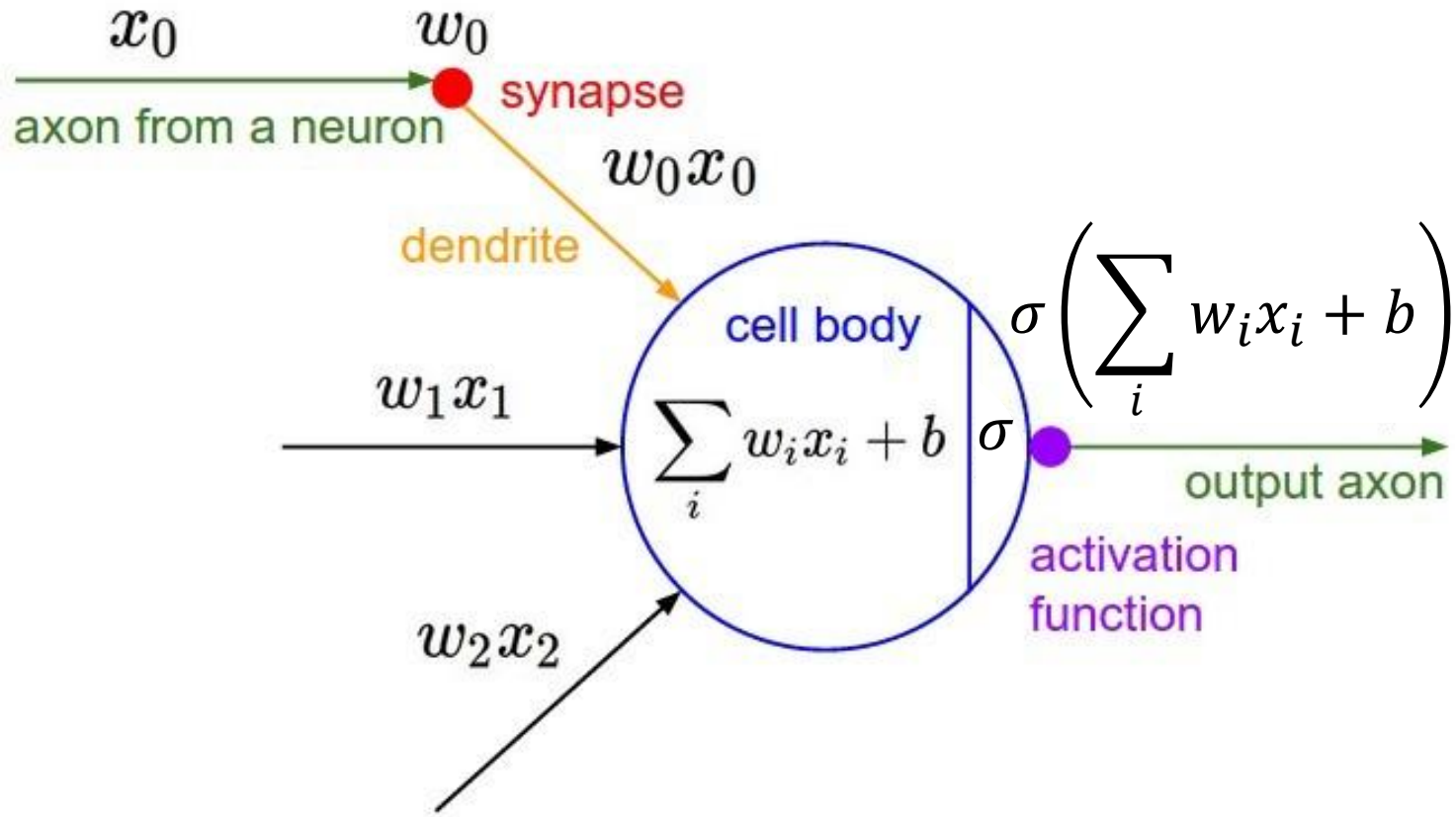

$$\text{Loss function } \mathcal{L}=\sum_{i=1}^{n}\|\hat{y}_i - y_i\|^2$$

## Representation Theory (Kurt Hornik, 1991)

A feed-forward neural network with **a single hidden layer** containing a **finite number of neurons** can approximate continuous function on compact subsets of $\mathbf{R}^n$ under mild assumptions on the **activation function**

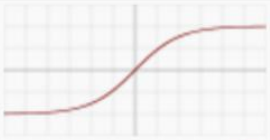
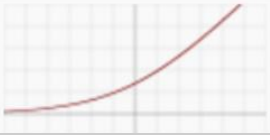
# Activation Function

- **Description**: keep value within an acceptable and useful range



# Activation Layer

- Activation functions: **nonlinear** and **continuously differentiable**

| Activation Function | Plot | Equation | Derivative |
|---|---|---|---|
| Identity |  | $f(x) = x$ | $f'(x) = 1$ |
| Sigmoid |  | $f(x) = \frac{1}{1+e^{-x}}$ | $f'(x) = f(x)\big(1 - f(x)\big)$ |
| Tanh |  | $f(x) = \frac{2}{1+e^{-2x}} - 1$ | $f'(x) = 1 - f(x)^2$ |
| ReLU |  | $f(x) = \max(x,\ 0)$ | $f'(x) = 1\{x \geq 0\}$ |
| SoftPlus |  | $f(x) = ln(1 + e^{x})$ | $f'(x) = \frac{1}{1+e^{-x}}$ |

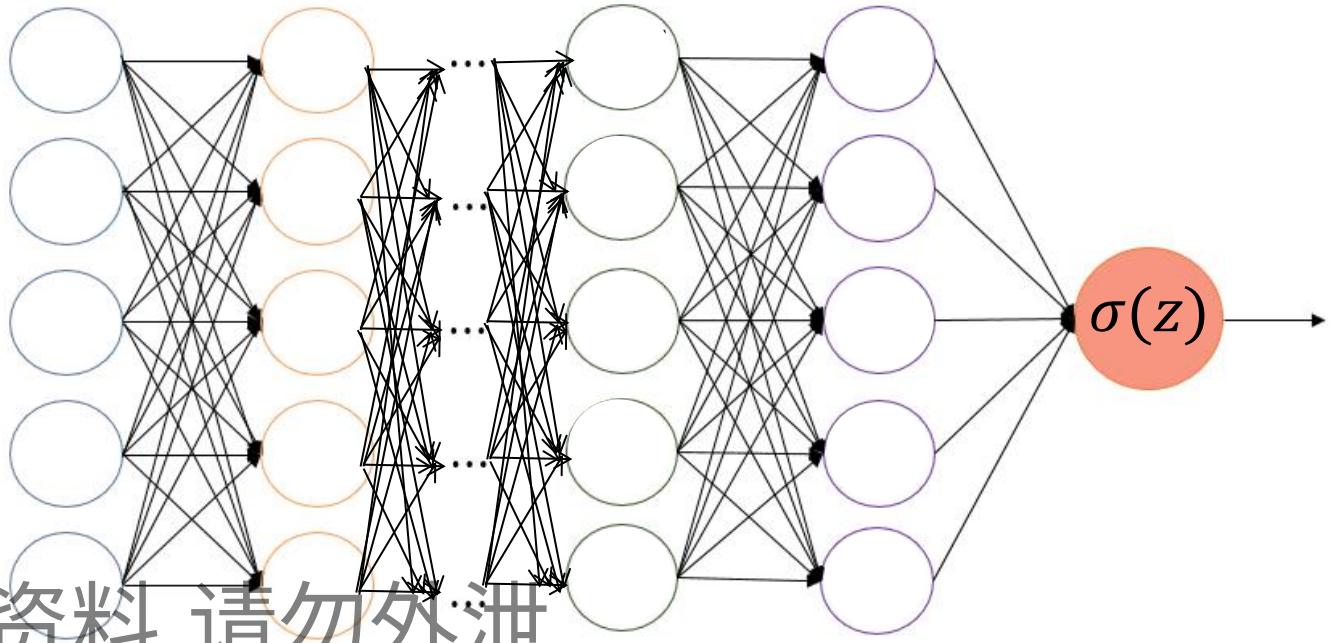
# From Shallow to Deep Neural Networks

## Function Approximation Theory (Shiyu Liang, 2017)

Shallow networks require **exponentially more neurons** than a deep network to achieve the same level of accuracy for function approximation

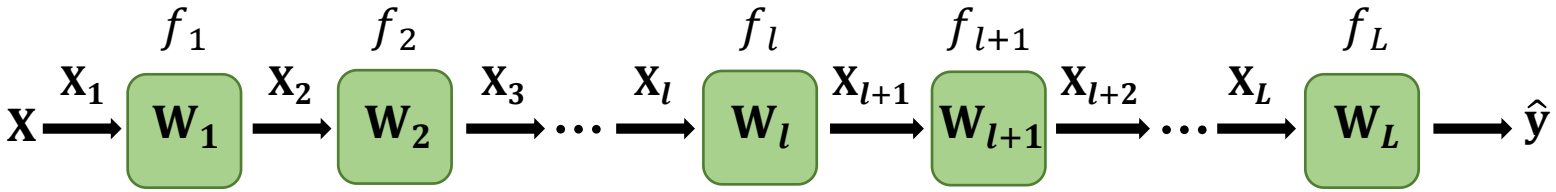
Shallow neural network → Need a lot of parameters → **Hard for training**

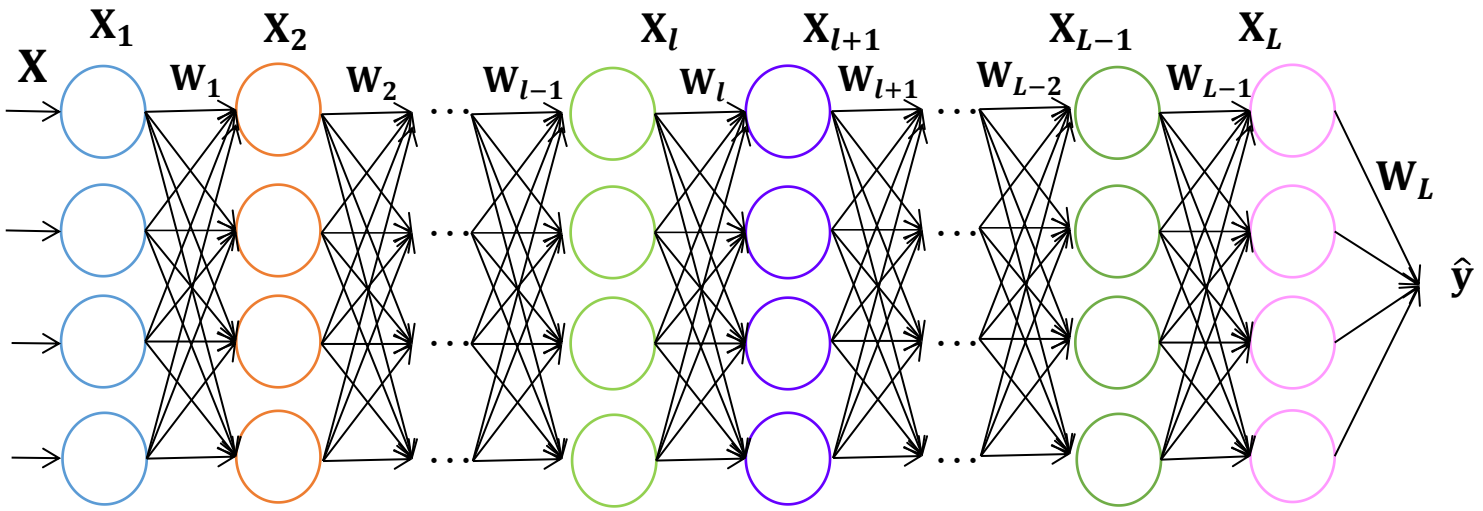
To approximate $F(\mathbf{X})$, we can build a **deep** neural network with less parameters:



# From Shallow to Deep Neural Networks

To approximate $F(\mathbf{X})$, we can build a **deep** neural network with **LESS parameters**:




$\mathbf{X}$ : input data

$\hat{\mathbf{y}}$ : prediction of the model

$\mathbf{X}_l$ : output of Layer $l$

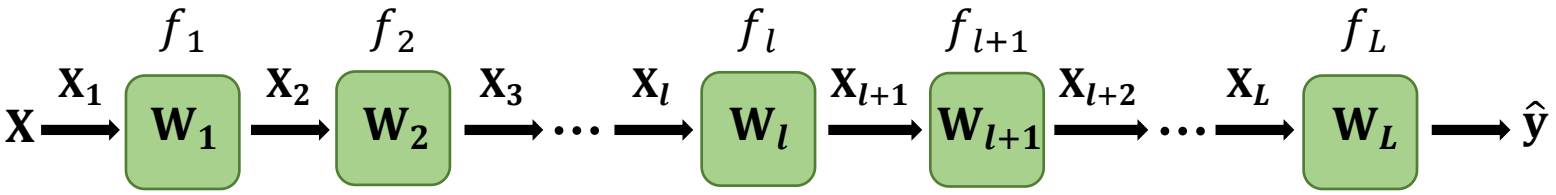
$f_l$ : function of Layer $l$

$\mathbf{W}_l$: parameters of Layer $l$

The output of layer $l$, i.e. $\mathbf{X}_{l+1}$ is the input of $f_{l+1}$

# Contents

# **Forward propagation** of neural networks




$$f = f_L \circ f_{L-1} \circ \cdots \circ f_2 \circ f_1$$

# Prediction for Networks $f$ : Forward Propagation

- Given input data $\mathbf{X}$, the forward propagation of a neural network $f$ is to **predict** the output $\hat{\mathbf{y}}$ of $\mathbf{X}$
- Rewrite the network as a Composite Function

$$\hat{\mathbf{y}} = \underset{W}{f}(\mathbf{X}) = f_L \circ f_{L-1} \circ \cdots \circ f_2 \circ f_1(\boldsymbol{X_1}, \boldsymbol{W_1})$$

- Detailed prediction process:

$$\begin{aligned} \mathbf{X}_1 &= \mathbf{X} \\ \mathbf{X}_2 &= f_1(\mathbf{X}_1, \mathbf{W}_1) \\ &\vdots \\ \mathbf{X}_{l+1} &= f_l(\mathbf{X}_l, \mathbf{W}_l) \\ &\vdots \\ \mathbf{X}_L &= f_{L-1}(\mathbf{X}_{L-1}, \mathbf{W}_{L-1}) \\ \hat{y} &= f_L(\mathbf{X}_L, \mathbf{W}_L) \end{aligned}$$

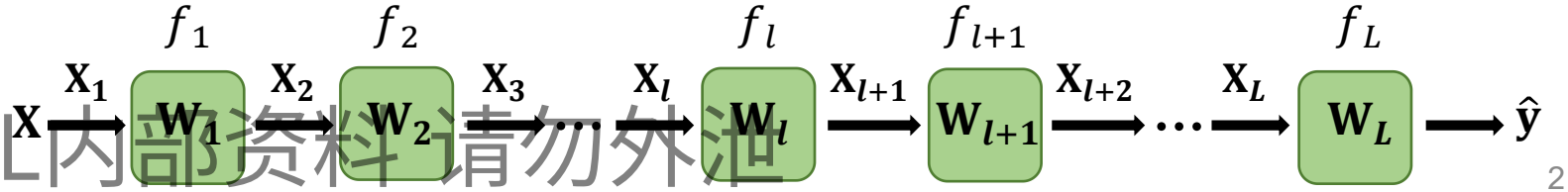
$$\hat{\mathbf{y}} = f(\mathbf{X})$$

$$f_l = \sigma(h_l(\mathbf{X}_l, \mathbf{W}_l))$$

Example:

$$h_l = \mathbf{W}_l * \mathbf{X}_l$$

$$\sigma = \mathrm{ReLU}(\cdot)$$



# Forward Propagation: Prediction of Networks

- Detailed prediction process:

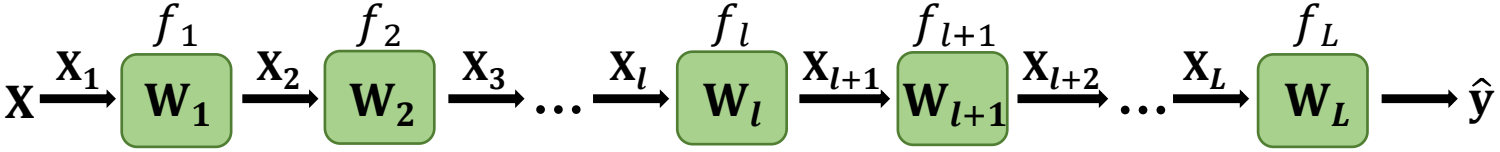
$$\mathbf{X}_1 = \mathbf{X}$$
$$\mathbf{X}_2 = f_1(\mathbf{X}_1, \mathbf{W}_1)$$
$$\vdots$$
$$\mathbf{X}_{l+1} = f_l(\mathbf{X}_l, \mathbf{W}_l)$$
$$\vdots$$
$$\mathbf{X}_L = f_{L-1}(\mathbf{X}_{L-1}, \mathbf{W}_{L-1})$$
$$\hat{\boldsymbol{y}} = f_L(\mathbf{X}_L, \mathbf{W}_L)$$

Compact Form

- Forward propagation for any layer $l$ :

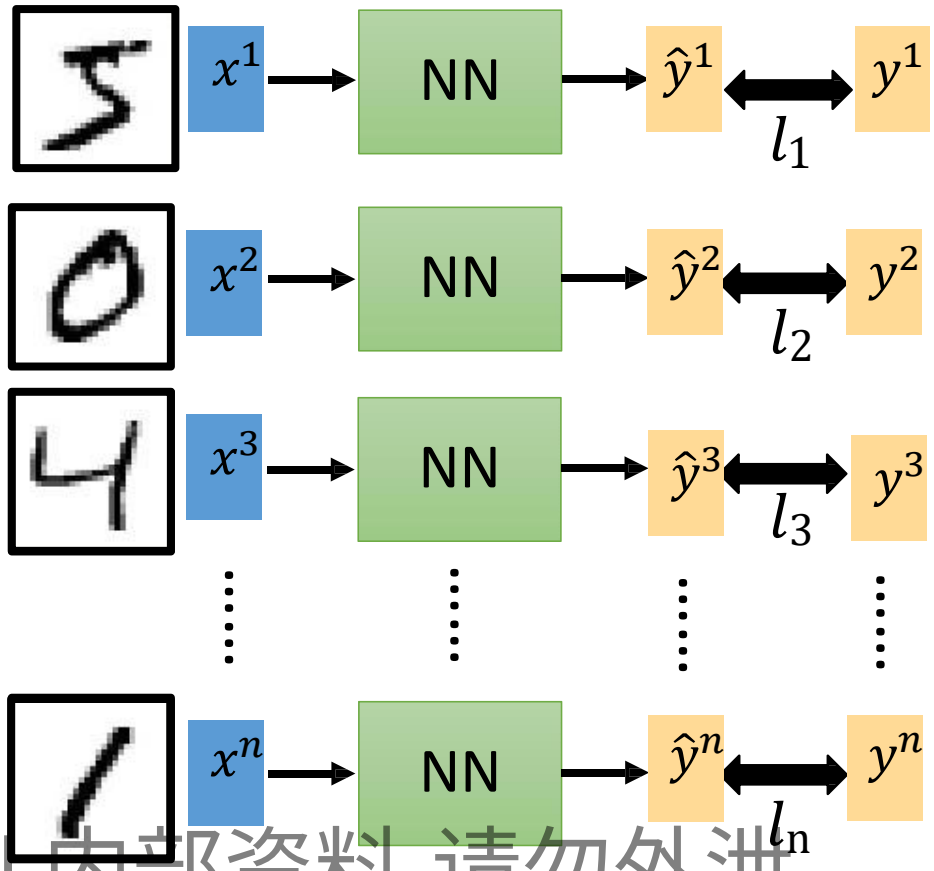
$$\begin{cases} \mathbf{Z}_l = h_l(\mathbf{X}_l, \mathbf{W}_l) \\ \mathbf{X}_{l+1} = \sigma(\mathbf{Z}_l) \end{cases}$$

$h_l$: some linear function
$\sigma$: some activation function



# Example of Forward Prediction

For all training data:




Total loss: $\mathcal{L} = \sum_{r=1}^{n} l_r$

As small as possible

Find ***a function in function set*** that minimizes total loss $\mathcal{L}$

Find ***the network parameters*** $\mathbf{W}^*$ that minimize total loss $\mathcal{L}$

# Contents

# **Backpropagation** of neural networks

# How to Learn Model Parameters $\{\mathbf{W}_l\}$?

- We minimize the following regularized loss function:

**Regularized Loss function:**

$$\min_{\mathbf{W}} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_W(\mathbf{y}_i, \hat{\mathbf{y}}_i) + \frac{1}{2}\lambda\|\mathbf{W}\|_F^2$$

$\mathbf{W}$ – parameters $\quad \mathcal{L}$ – loss function

$\mathbf{y}_i$ – ground-truth label $\quad \hat{\mathbf{y}}_i$ – predicted label

**Connection to previous models?**

- Linear Regression?
- Support Vector Machine?
- Logistic Regression?
- Softmax Regression?
- AdaBoost?
- GBDT?

- Gradient Descent: update the parameters $\mathbf{W}_l$ by

$$\mathbf{W}_l' = \mathbf{W}_l - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} \quad (1)$$

where $\eta$ is the learning rate

**How to compute the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$?**

# How to Compute Gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$?

- Recall the loss function can be rewritten as the Composite Function:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \mathcal{L} \circ f_L \circ f_{L-1} \circ \cdots \circ f_l(\mathbf{X}_l, \boldsymbol{W}_l) \circ \cdots \circ f_2 \circ f_1$$

**Computing the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$ is extremely difficult!**

- Let us resort to **Chain Rule** to compute the gradient

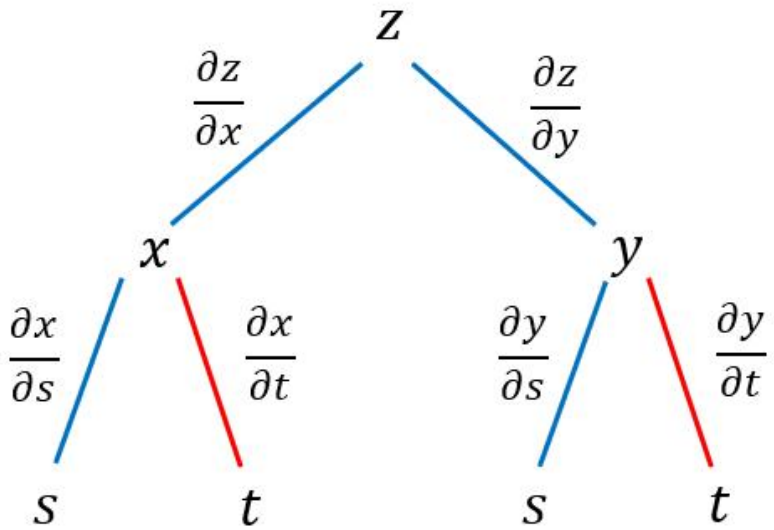
$$z = x \circ s \implies \boxed{\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s}} \quad ?$$

# Chain Rule Revisited

- Suppose we have the following differentiable functions:

$$\begin{cases} z = f(x, y) \\ x = g(s, t) \\ y = h(s, t) \end{cases}$$

**How to compute $\frac{\partial z}{\partial s}$ and $\frac{\partial z}{\partial t}$?**




- We have **Chain Rule** as below:

$$\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial s}$$

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial t}$$

# Chain Rule Revisited

- Vector product derivative example:

functions: $Y = X^T Q X$

$Q$ is Symmetric Matrices

**How to compute $\frac{\partial Y}{\partial X}$?**

$$\frac{\partial Y}{\partial x} = \frac{\partial((Q^T X_c)^T X)}{\partial x} + \frac{\partial((Q X_c)^T X)}{\partial x}$$

$$(X_c^T Q X)^T = (Q X_c)^T X$$

$$= Q^T X_c + Q X_c$$

$$Q^T = Q$$

$$= 2QX$$

# Gradient Computation of Neural Network $f$

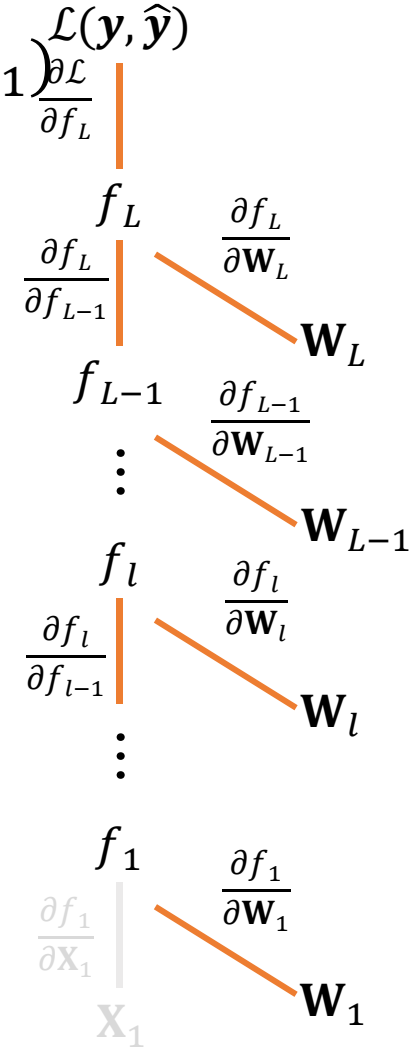
$$\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = \mathcal{L} \circ f_L \circ f_{L-1} \circ \cdots \circ f_l \circ \cdots \circ f_2 \circ f_1(\mathbf{X}_1, \boldsymbol{W}_1)$$

**How to compute the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}$?**

- According to **Chain Rule**, we have

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \cdots \cdot \frac{\partial f_3}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial \mathbf{W}_1}$$

Gradient Chain



# Gradient for Any Layer $l$

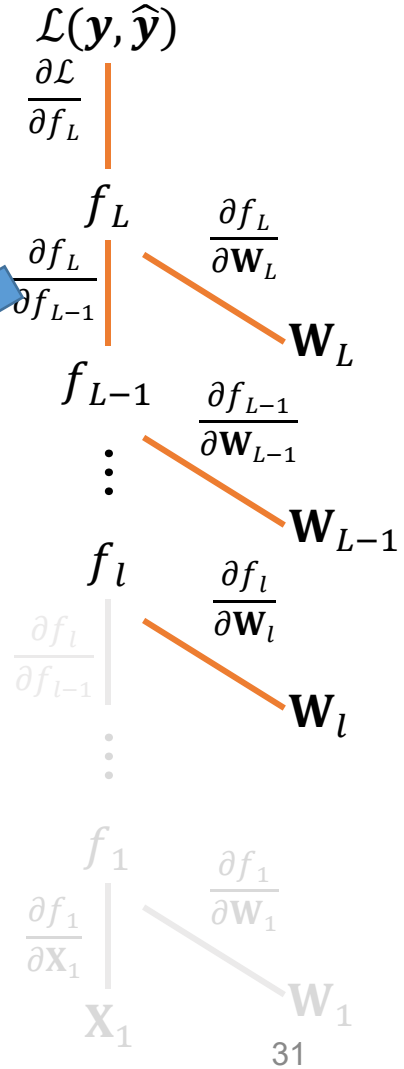
$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \mathcal{L} \circ f_L \circ f_{L-1} \circ \cdots \circ f_l(\mathbf{X}_l, \boldsymbol{W}_l) \circ \cdots \circ f_2 \circ f_1$$

**How to compute the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$?**

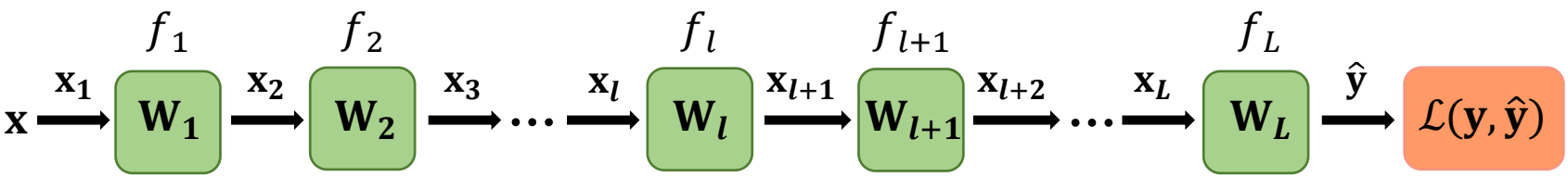
- According to **Chain Rule**, we have

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_l} = \boxed{\frac{\partial \mathcal{L}}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \cdots \cdot \frac{\partial f_{l+1}}{\partial f_l}} \cdot \frac{\partial f_l}{\partial \mathbf{W}_l}$$

Gradient Chain



# How to Compute Gradient in Practice




From deep to shallow

For Layer $L$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_L} = \frac{\partial \mathcal{L}}{\partial f_L} \cdot \frac{\partial f_L}{\partial \mathbf{W}_L}$$

For Layer $L-1$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_{L-1}} = \boxed{\frac{\partial \mathcal{L}}{\partial f_L}} \frac{\partial f_L}{\partial f_{L-1}} \cdot \frac{\partial f_{L-1}}{\partial \mathbf{W}_{L-1}}$$

$\vdots$

For Layer $l$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_l} = \boxed{\frac{\partial \mathcal{L}}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \cdots \frac{\partial f_{l+2}}{\partial f_{l+1}}} \frac{\partial f_{l+1}}{\partial f_l} \cdot \frac{\partial f_l}{\partial \mathbf{W}_l}$$

$\vdots$

For Layer 1

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \cdots \frac{\partial f_{l+2}}{\partial f_{l+1}} \cdot \frac{\partial f_{l+1}}{\partial f_l} \cdot \cdots \frac{\partial f_3}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial \mathbf{W}_1}$$

# How to Compute Gradient in Practice

Introduce intermediate variable $\{G_l\}_{l=1}^{L}$

For Layer $L$

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_L} = \frac{\partial \mathcal{L}}{\partial f_L} \cdot \frac{\partial f_L}{\partial \mathbf{W}_L}$$

For Layer $L-1$

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_{L-1}} = \boxed{\frac{\partial \mathcal{L}}{\partial f_L}} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \frac{\partial f_{L-1}}{\partial \mathbf{W}_{L-1}}$$

$\vdots$

For Layer $l$

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_l} = \boxed{\frac{\partial \mathcal{L}}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \ldots \cdot \frac{\partial f_{l+2}}{\partial f_{l+1}}} \cdot \frac{\partial f_{l+1}}{\partial f_l} \cdot \frac{\partial f_l}{\partial \mathbf{W}_l}$$

$\vdots$

For Layer 1

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \ldots \cdot \frac{\partial f_3}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial \mathbf{W}_1}$$

$$G_L = \frac{\partial \mathcal{L}}{\partial f_L}$$

$$G_{L-1} = G_L \cdot \frac{\partial f_L}{\partial f_{L-1}}$$

$\vdots$

$$G_l = G_{l+1} \cdot \frac{\partial f_{l+1}}{\partial f_l}$$

$\vdots$

$$G_1 = G_2 \cdot \frac{\partial f_2}{\partial f_1}$$

# How to Compute Gradient in Practice

For Layer $L$

$$G_L = \frac{\partial \mathcal{L}}{\partial f_L}$$

For Layer $L-1$

$$G_{L-1} = G_L \cdot \frac{\partial f_L}{\partial f_{L-1}}$$

$$\vdots$$

For Layer $l$

$$G_l = G_{l+1} \cdot \frac{\partial f_{l+1}}{\partial f_l}$$

$$\vdots$$

For Layer 1

$$G_1 = G_2 \cdot \frac{\partial f_2}{\partial f_1}$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_L} = G_L \cdot \frac{\partial f_L}{\partial \mathbf{W}_L}$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_{L-1}} = G_{L-1} \cdot \frac{\partial f_{L-1}}{\partial \mathbf{W}_{L-1}}$$

$$\vdots$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_l} = G_l \cdot \frac{\partial f_l}{\partial \mathbf{W}_l}$$

$$\vdots$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_1} = G_1 \cdot \frac{\partial f_1}{\partial \mathbf{W}_1}$$

# How to Compute Gradient in Practice

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_L} = G_L \cdot \frac{\partial f_L}{\partial \mathbf{W}_L}$$

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_{L-1}} = G_{L-1} \cdot \frac{\partial f_{L-1}}{\partial \mathbf{W}_{L-1}}$$

$$\vdots$$

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_l} = G_l \cdot \frac{\partial f_l}{\partial \mathbf{W}_l}$$

$$\vdots$$

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{W}_1} = G_1 \cdot \frac{\partial f_1}{\partial \mathbf{W}_1}$$

$$G_L = \frac{\partial \mathcal{L}}{\partial f_L}$$

$$G_{L-1} = G_L \cdot \frac{\partial f_L}{\partial f_{L-1}}$$

$$\vdots$$

$$G_l = G_{l+1} \cdot \frac{\partial f_{l+1}}{\partial f_l}$$

$$\vdots$$

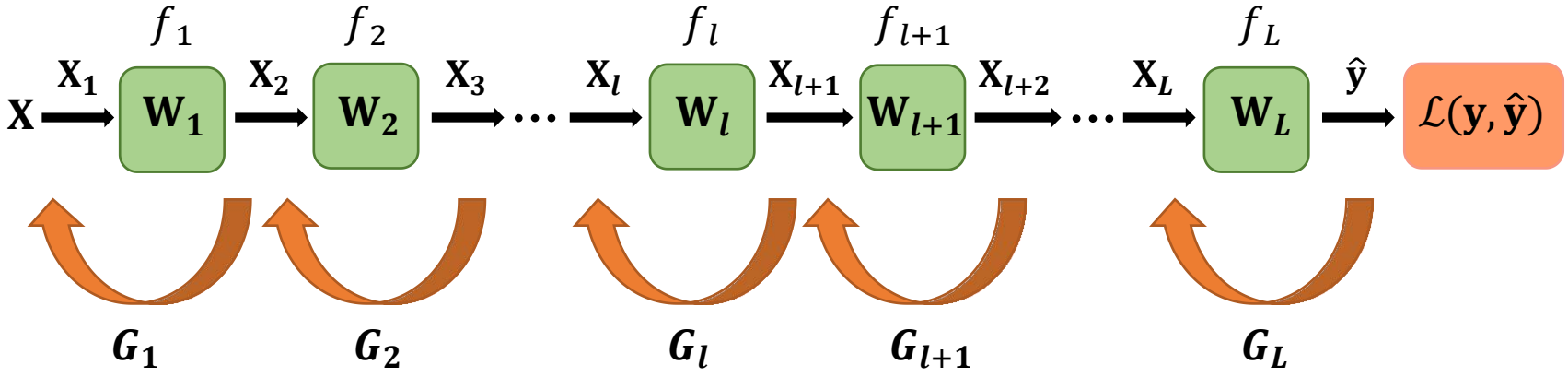
$$G_1 = G_2 \cdot \frac{\partial f_2}{\partial f_1}$$

$$f_l = \sigma(h_l(\mathbf{X}_l, \mathbf{W}_l)) \Longrightarrow \frac{\partial f_l}{\partial \mathbf{W}_l} = \frac{\partial \sigma}{\partial h_l} \cdot \frac{\partial h_l}{\partial \mathbf{W}_l} \qquad \frac{\partial f_l}{\partial f_{l-1}} = \frac{\partial \sigma}{\partial h_l} \cdot \frac{\partial h_l}{\partial f_{l-1}}$$

# Backpropagation for Neural Networks

## Backpropagation:

**Backpropagation** propagates the **error/loss** from the **last layer** to the **first layer** to compute the **gradient** for each layer



# Forward Propagation and Backpropagation

**Algorithm 1 Forward and Backward Propagation for an $L$-layer Neural Network**

**Require:** Input data $\mathbf{X}$, label $\mathbf{y}$, layer functions $\{f_l\}_{l=1}^{L}$, layer parameters $\{\mathbf{W}_l\}_{l=1}^{L}$

**Step1: Conduct forward propagation**

Compute $\hat{\mathbf{y}} = f_W(\mathbf{X}) = f_L \circ f_{L-1} \circ \cdots \circ f_l \circ \cdots \circ f_2 \circ f_1(\boldsymbol{X_1}, \boldsymbol{W_1})$

**Step2: Compute loss** $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$

**Step3: Conduct backpropagation**

Compute $G = \frac{\partial \mathcal{L}}{\partial f_L}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_L} = G \cdot \frac{\partial f_L}{\partial \mathbf{W}_L}$

**For** $l = L - 1$ to 1 **do**:

**propagate** $G_{l+1}$ to current layer: $G = G \cdot \frac{\partial f_{l+1}}{\partial f_l}$

**compute** gradient of current layer: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} = G \cdot \frac{\partial f_l}{\partial \mathbf{W}_l}$

**End**

Update parameter $\mathbf{W}_l$ for any layer $l$ with (**Stochastic**) Gradient Descent

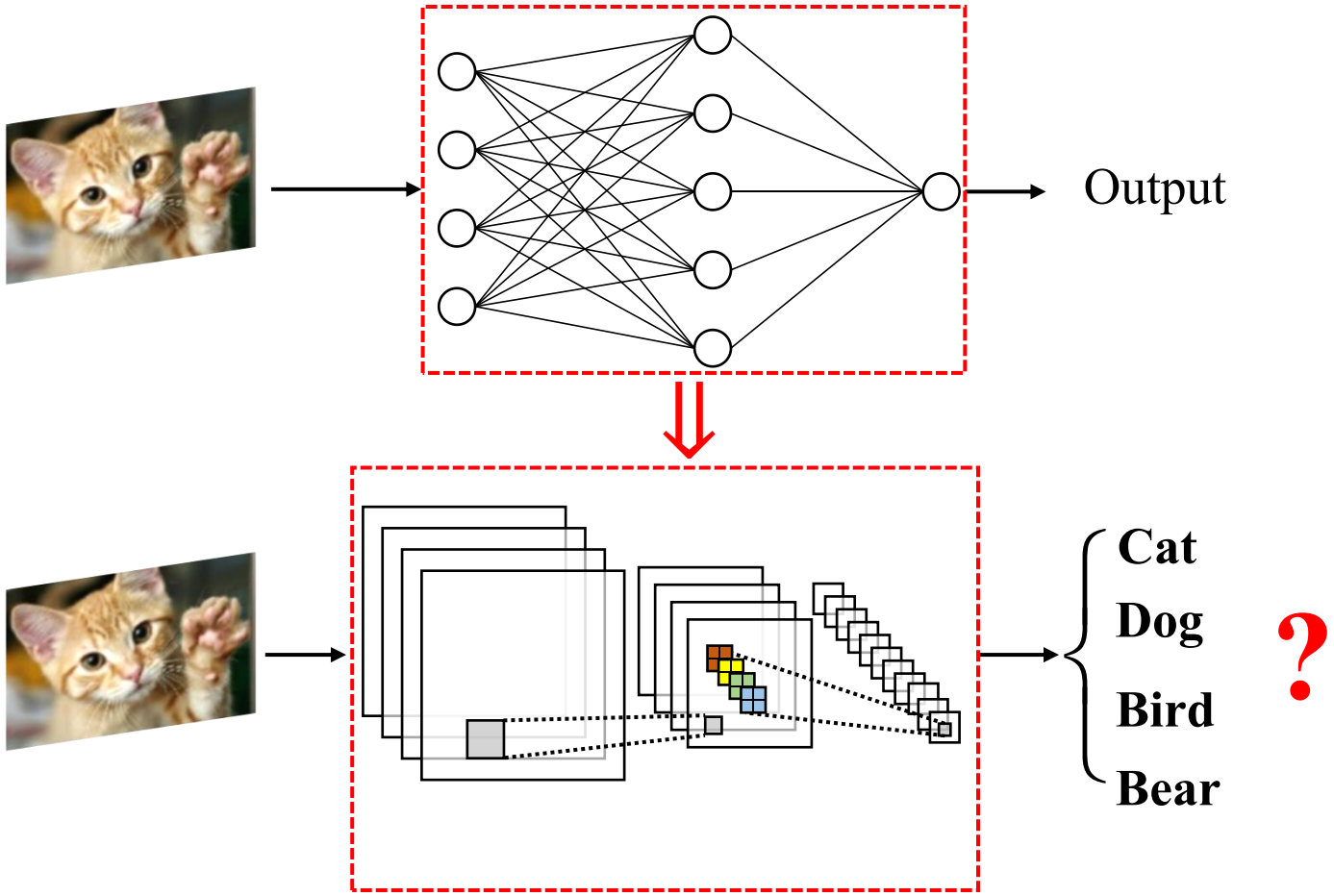
$$\mathbf{W}_l := \mathbf{W}_l - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$$

# Comparison of Training Algorithms

| Method | Advantages | Disadvantages |
| --- | --- | --- |
| Gradient Descent (GD) | 1. Accurate search direction<br>2. Suitable for parallelization | 1. Slow convergence speed<br>2. Need huge memory for big data |
| Stochastic Gradient Descent (SGD) | 1. Cheap for each step<br>2. Suitable for big data | 1. Need more steps<br>2. Not stable<br>3. Hard for parallelization |
| Mini-Batch Gradient Descent (MBGD) | 1. Converge faster than GD<br>2. More accurate update than SGD<br>3. Suitable for parallelization<br>4. Suitable for large data | 1. Require adding learning-decay to decrease the learning rate<br>2. Less stable than GD |

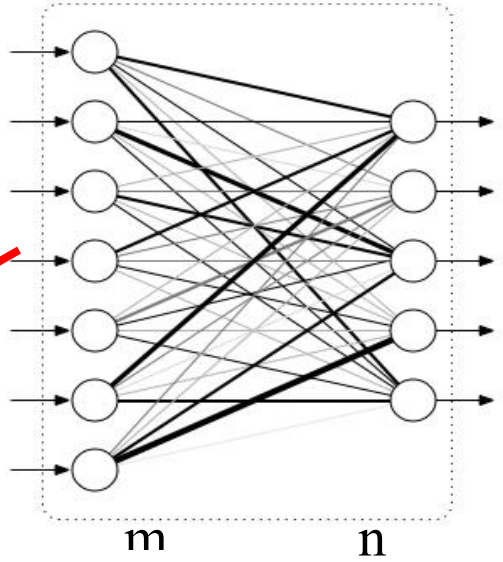
# Contents

# From Neural Networks to Convolutional Neural Networks

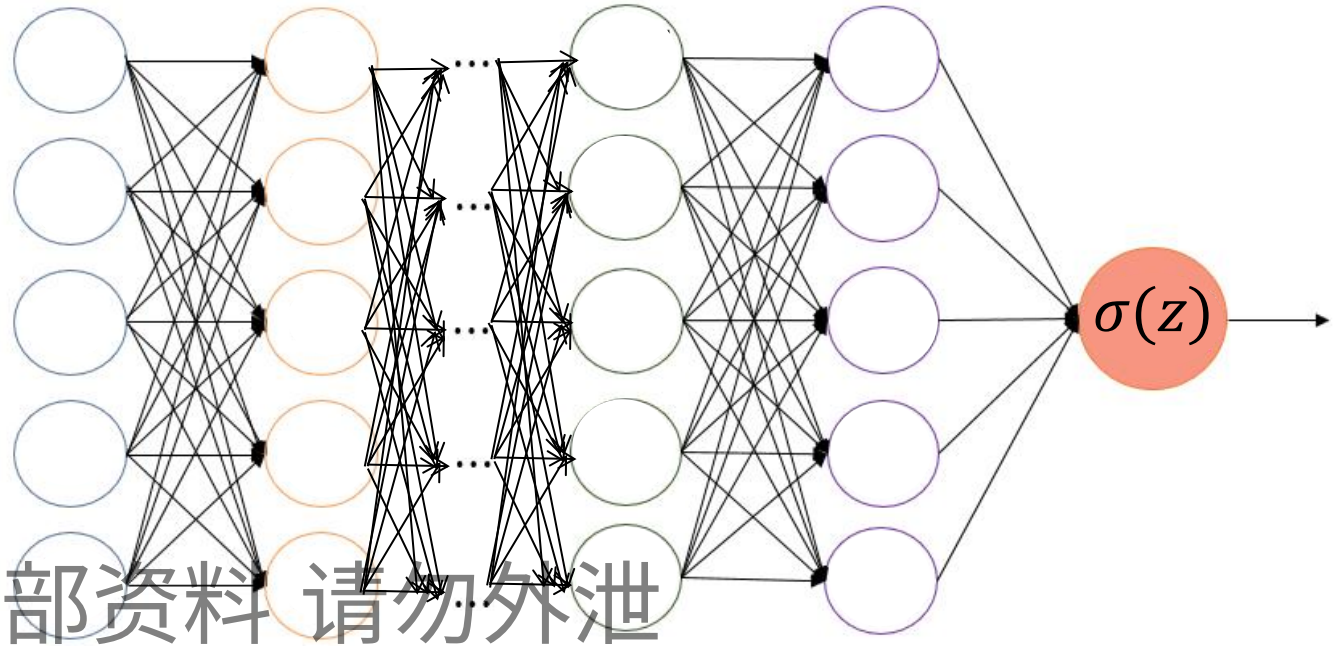


# Why Convolutional Layer

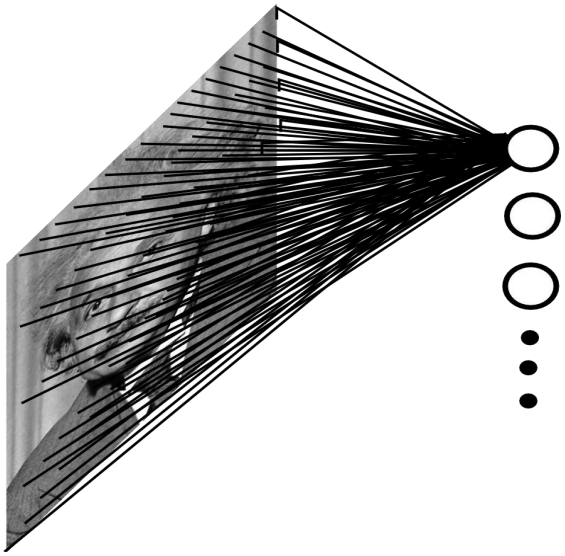
- Fully Connected Neural Networks
- $m \times n$ connections
- $m \times n$ can be very large




**Too many connections**



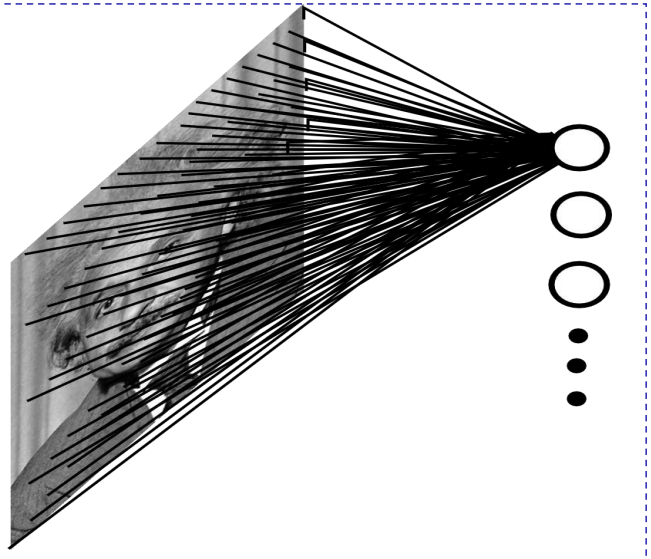
# Too Many Connections and Parameters



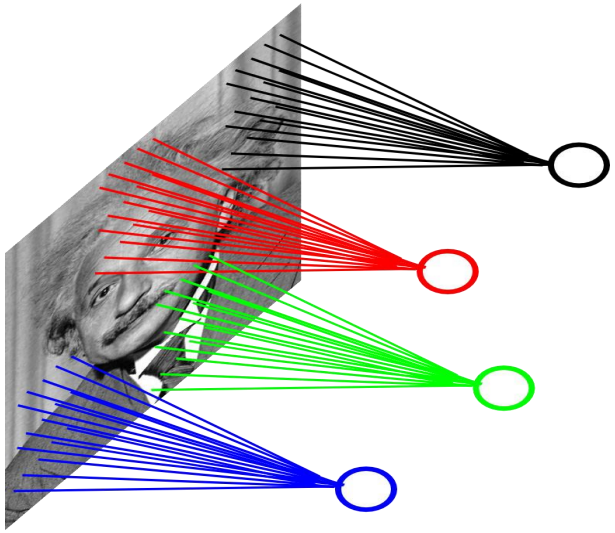
| Input Size | #hidden units | #Connections |
|---|---|---|
| 3x3 | n | 9n |
| 32x32 | n | 1024n |
| 224x224 | n | 50176n |
| 1000x1000 | n | $10^6$n |

**Too many parameters**

# From Full Connections to Local Connections



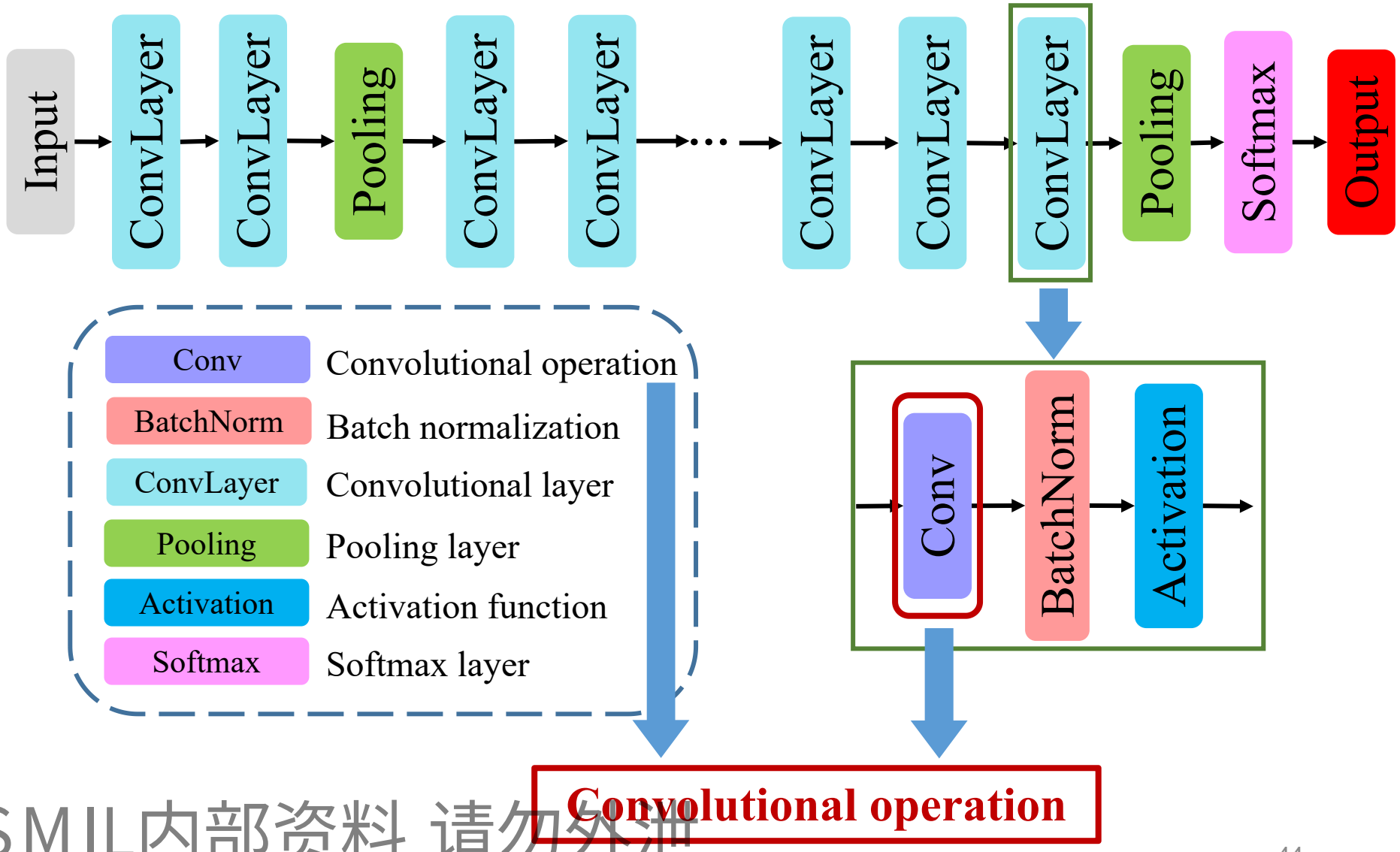
**Full connections**



**Local connections**

**Reduce the number of parameters**

# Basic Concepts of Convolutional Neural Networks



# The kernel in convolutional layer

## Kernel:

**A kernel / filter** is a matrix of values, called weights, that are trained to detect specific features.

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Input

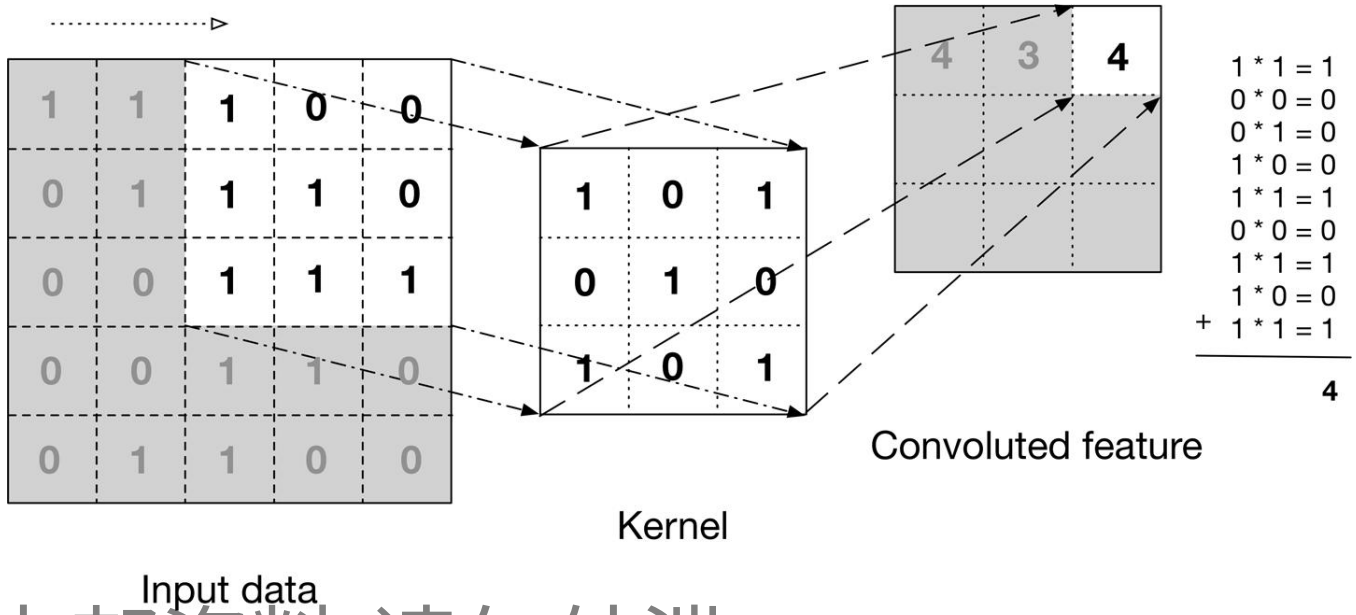
| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

Filter / Kernel

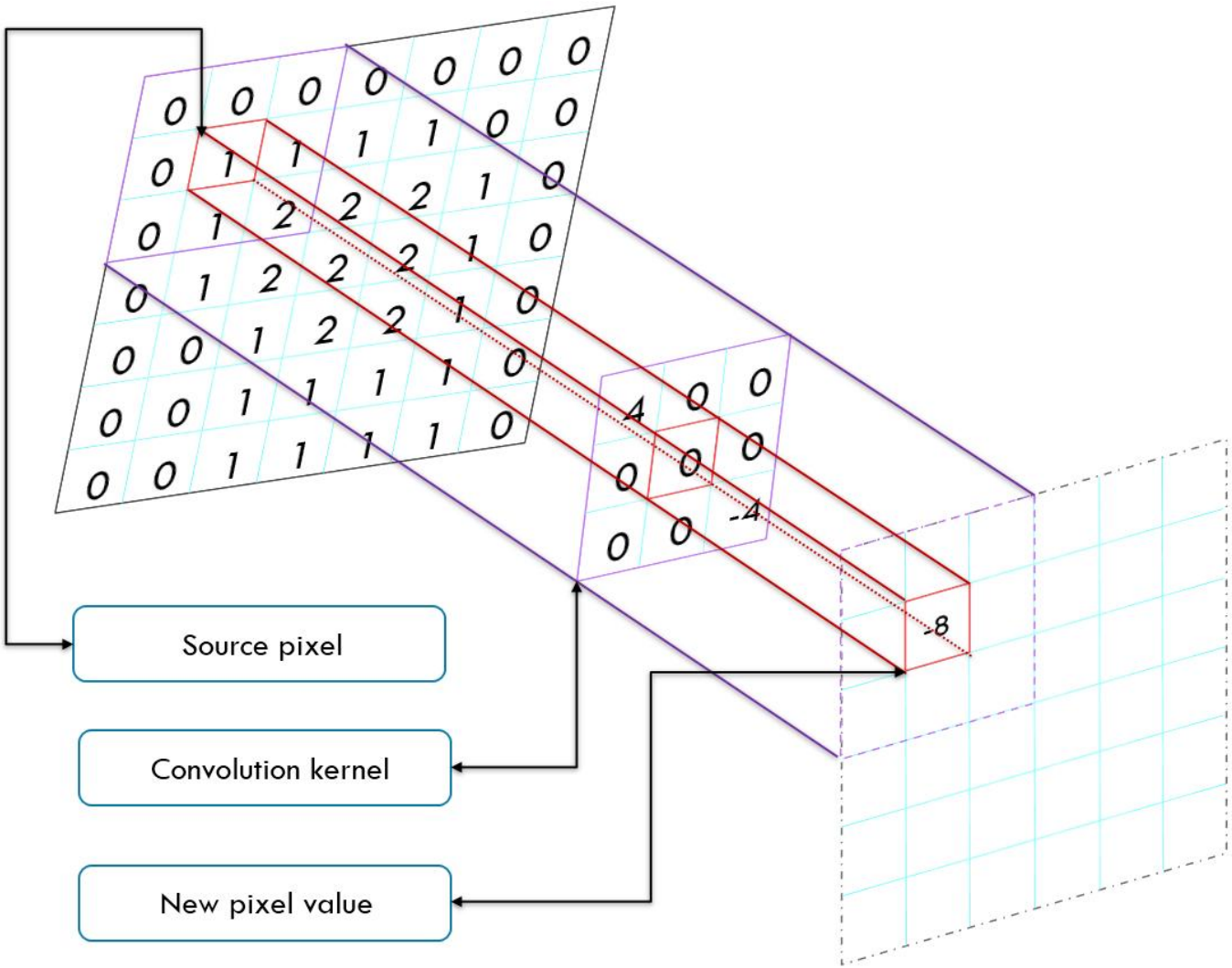
# The convolution operation in convolutional layer

## Convolution operation:

**Convolution operation** is an element-wise product and sum between two matrices.



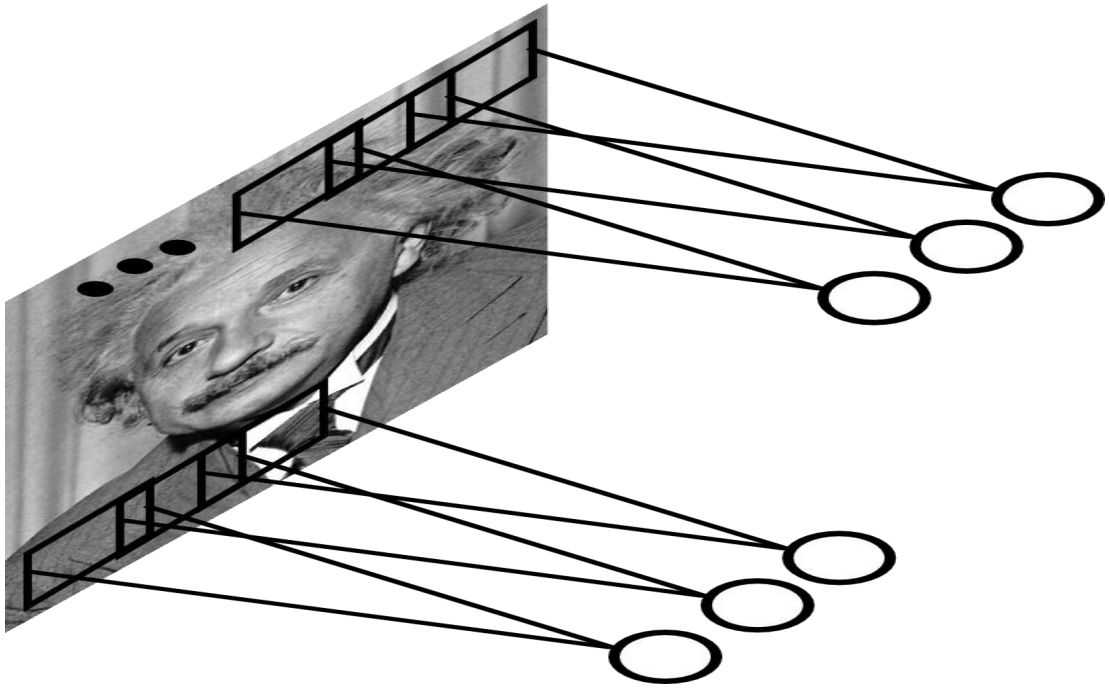
# Convolution as Local Connections




$$
\begin{aligned}
0 \times 4 &= 0 \\
0 \times 0 &= 0 \\
0 \times 0 &= 0 \\
0 \times 0 &= 0 \\
1 \times 0 &= 0 \\
1 \times 0 &= 0 \\
0 \times 0 &= 0 \\
+\ 1 \times 0 &= 0 \\
2 \times -4 &= -8 \\
\hline
&-8
\end{aligned}
$$

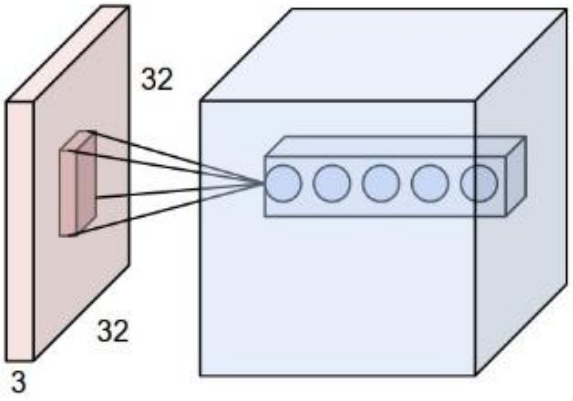
# Further Reduce Parameters: Weights Sharing

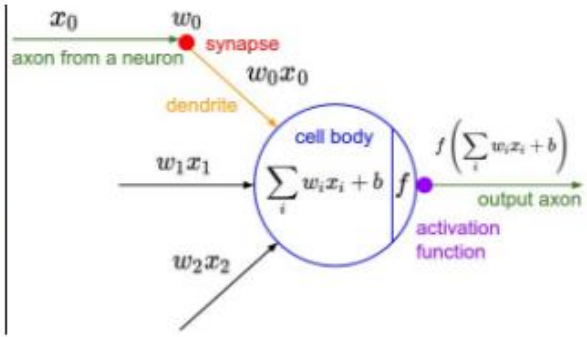
- Apply **the same filter** across the whole image
- Apply many kernels
- Each kernel is associated with one feature map

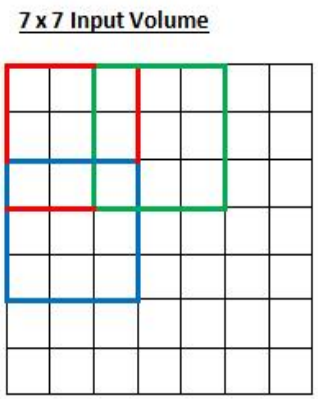


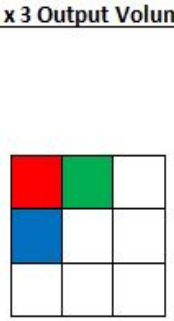
# Convolutional Layer

Filter

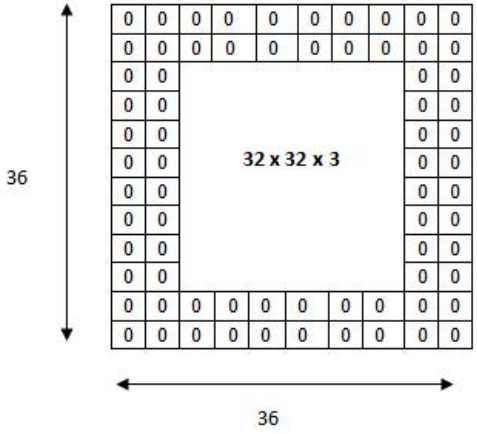



Stride




Padding

# Example: Convolution with Scanning

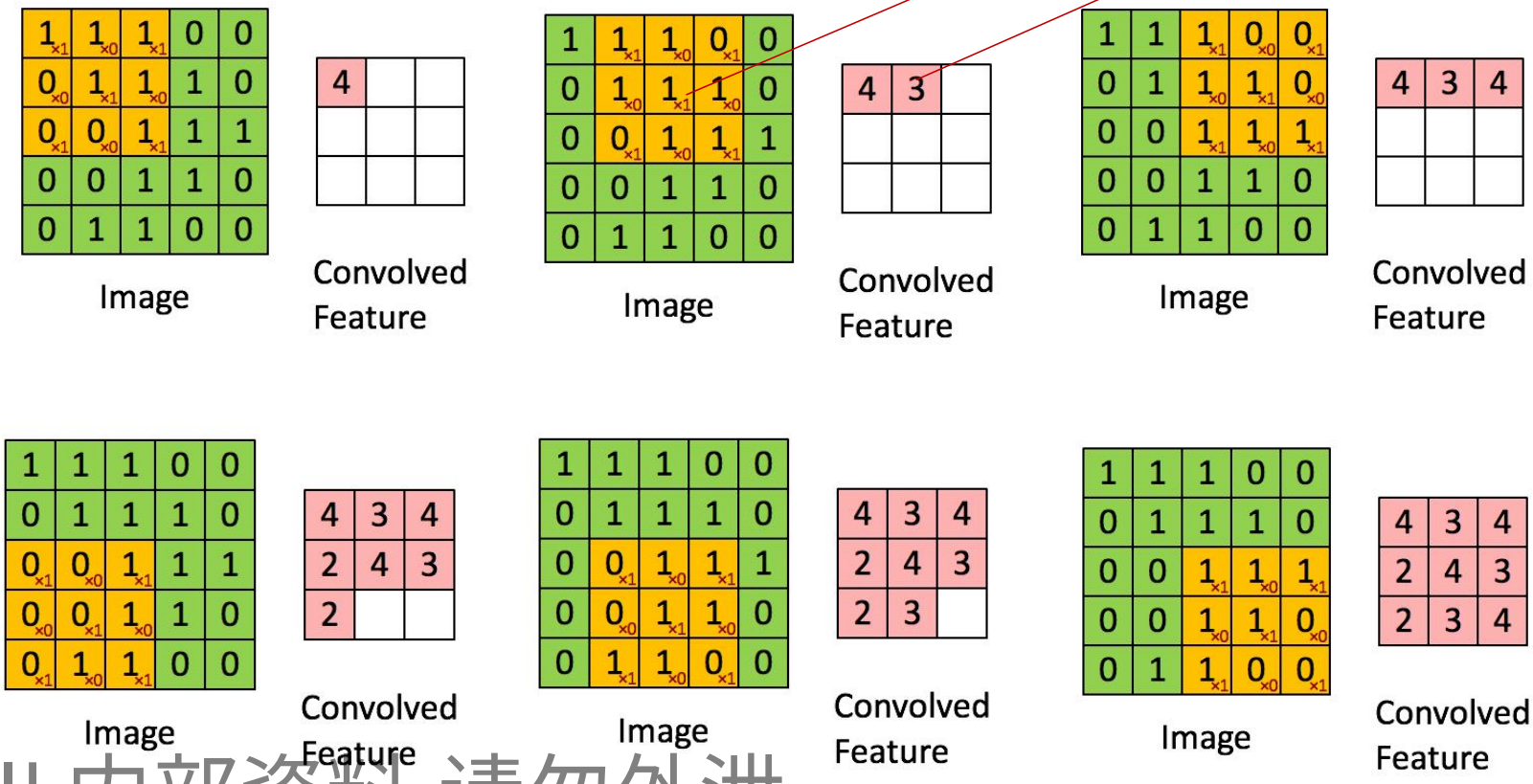
The kernel is

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

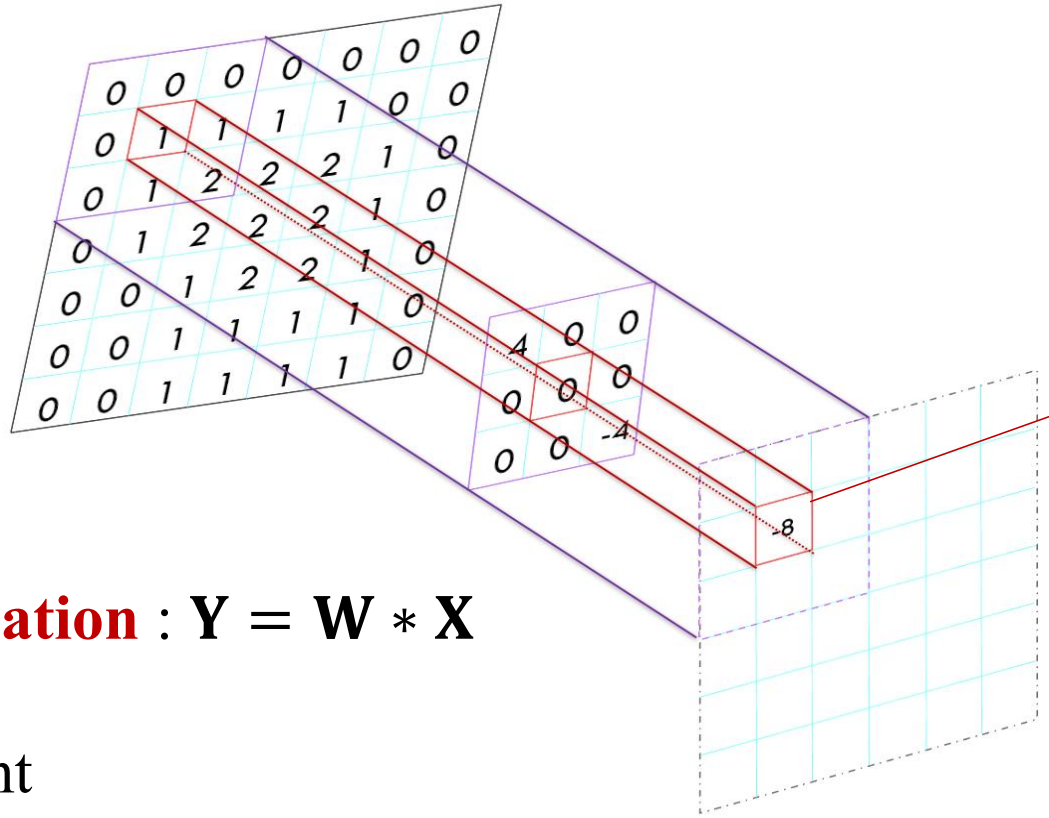
Example:

| | | |
|---|---|---|
| **1x1=1** | **1x0=0** | **0x1=0** |
| 1x0=0 | 1x1=1 | 1x0=0 |
| 0x1=0 | 1x0=0 | 1x1=1 |

$\sum$ = **3**



# Properties of Convolution



$$
\begin{aligned}
&0 \times 4 = 0 \\
&0 \times 0 = 0 \\
&0 \times 0 = 0 \\
&0 \times 0 = 0 \\
&1 \times 0 = 0 \\
&1 \times 0 = 0 \\
&0 \times 0 = 0 \\
+\;&1 \times 0 = 0 \\
&2 \times -4 = -8 \\
\hline
&-8
\end{aligned}
$$

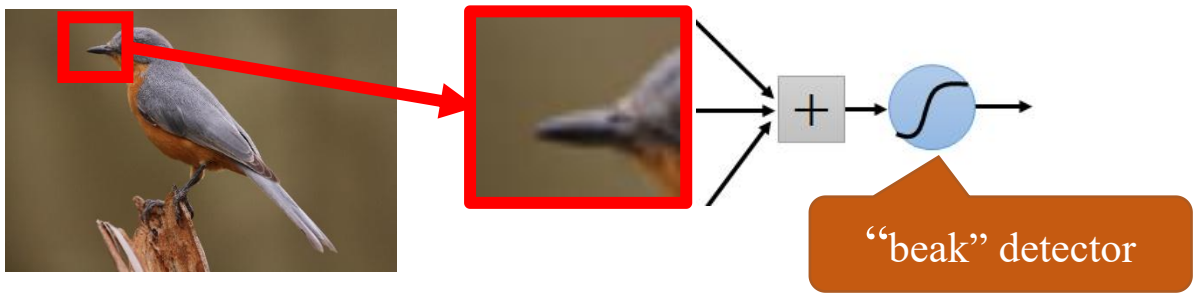
- **Linear Operation** : $\mathbf{Y} = \mathbf{W} * \mathbf{X}$
- Shift-invariant
- Size: odd by odd (3×3, 5×5, 7×7)

# Why Local Connections and Parameter Sharing

- Some patterns are much smaller than the whole image

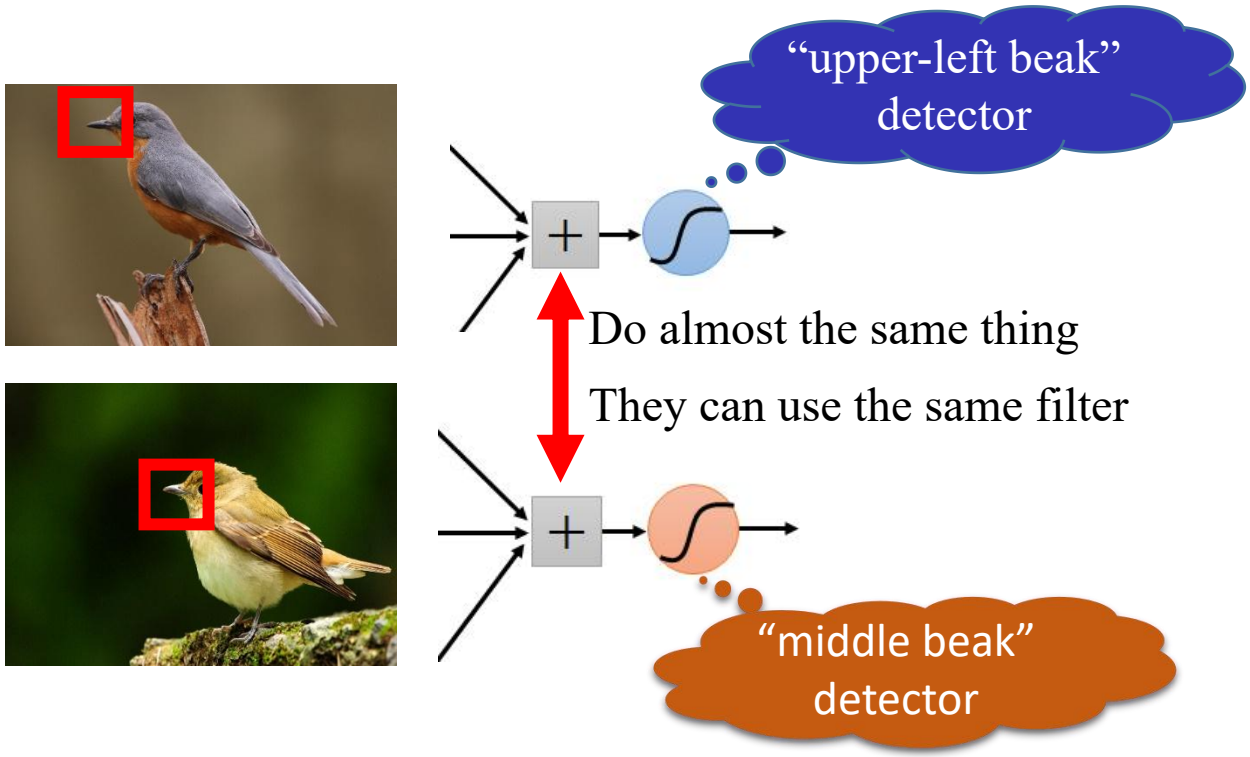
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters



# Why Local Connections

- The same patterns appear in different regions.



# Typical Convolution Examples

## Original

*

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

=

# Typical Convolution Examples

## Sharpen

$$*\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

# Typical Convolution Examples

## Sharpen with more details

$*$

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 2 | 2 | 2 | -1 |
| -1 | 2 | 8 | 2 | -1 |
| -1 | 2 | 2 | 2 | -1 |
| -1 | -1 | -1 | -1 | -1 |

$=$

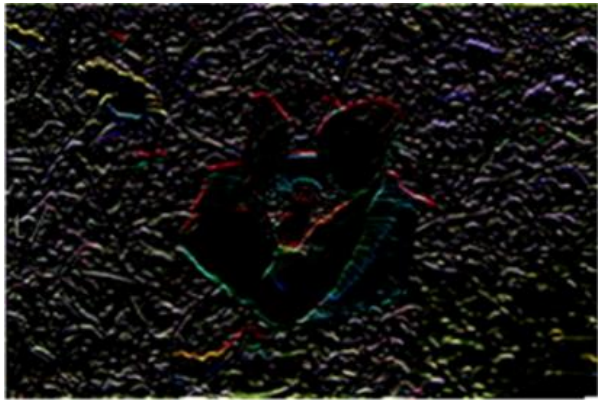
# Typical Convolution Examples

## Horizontal edge detection

$*$

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| -1 | -1 | 2 | -1 | -1 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

$=$



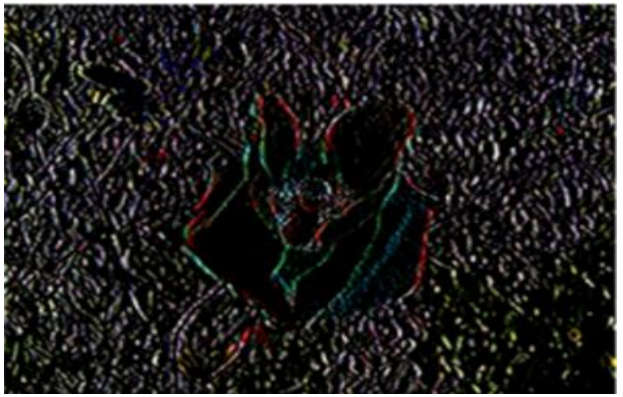
# Typical Convolution Examples

## Vertical edge detection

$*$

| | | | | |
|---|---|---|---|---|
| 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | 4 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 |

$=$



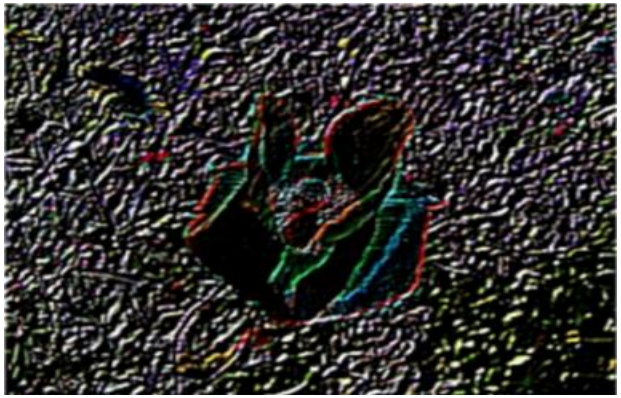
# Typical Convolution Examples

## 45 degree edge detection

*

| | | | | |
|---|---|---|---|---|
| -1 | 0 | 0 | 0 | 0 |
| 0 | -2 | 0 | 0 | 0 |
| 0 | 0 | 6 | 0 | 0 |
| 0 | 0 | 0 | -2 | 0 |
| 0 | 0 | 0 | 0 | -1 |

=



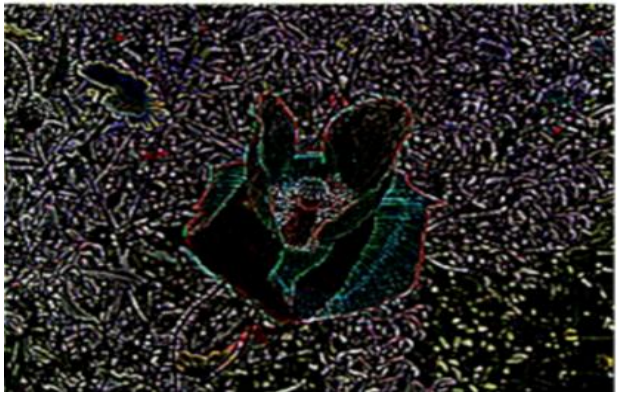
# Typical Convolution Examples

## All direction edge detection

*

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

=



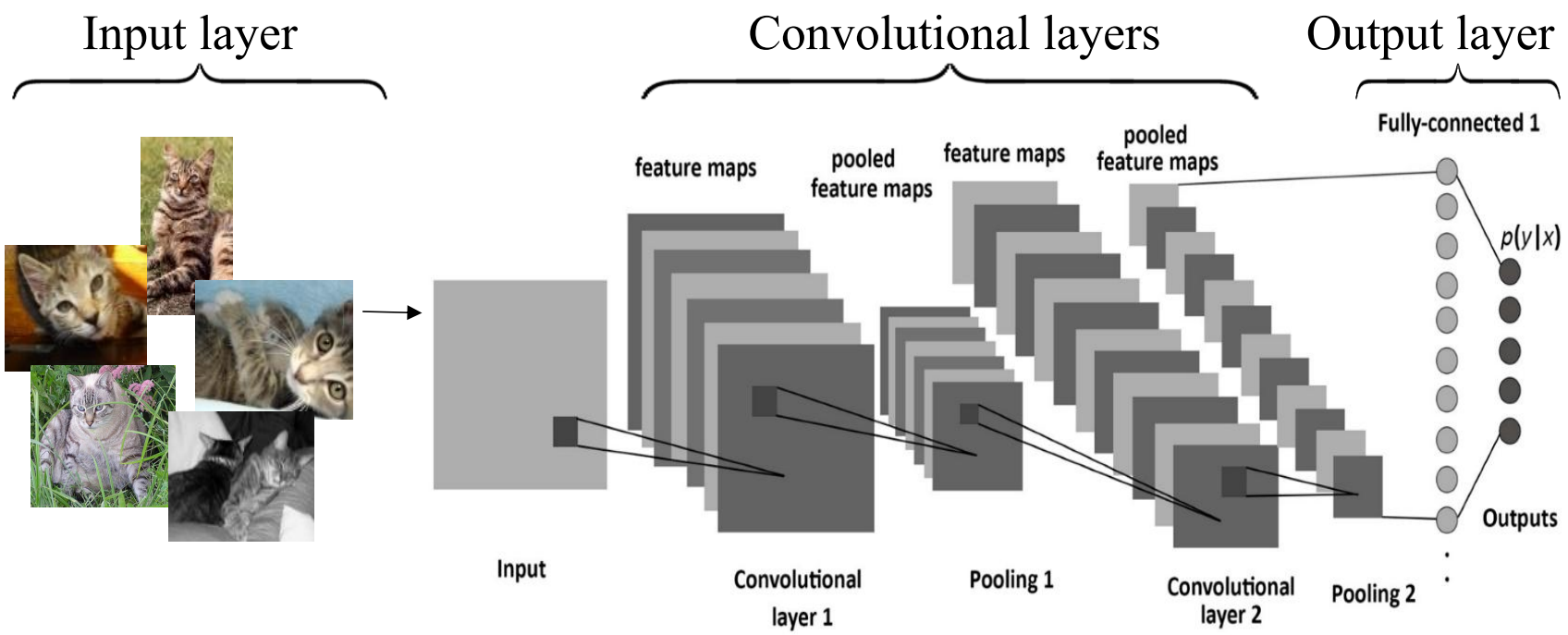
# Some Typical Kernels

| Operation | Kernel | Image result |
|---|---|---|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |

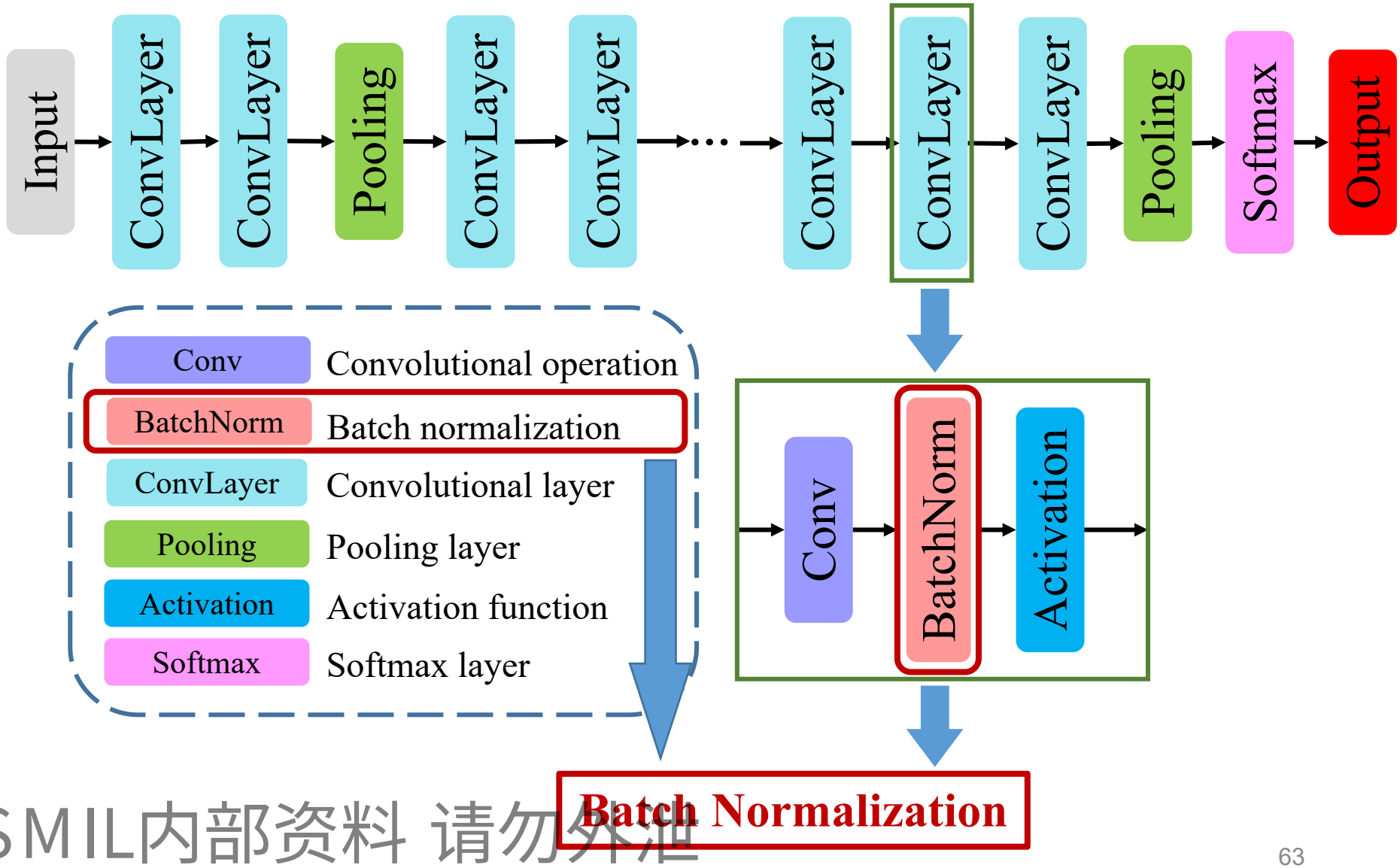


| Operation | Kernel | Image result |
|---|---|---|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |

# Convolutional Neural Networks



# Batch Normalization



# Batch Normalization Layer

- **Internal Covariate Shift**:
  features are normalized to have zero-mean and unit variance
- **Feature Normalization:**
  distribution changes during training
- **Batch Normalization:**
  **Input**: Values of $\mathbf{x}$ : $\mathbf{x} = \{x_1, \ldots, x_m\}$;
  Parameters to be learned: $\gamma, \beta$
  **Output**: $\{y_i = BN_{\gamma,\beta}(\mathbf{x})\}$

# Normalization in Each Channel

- **Compute data mean per channel**

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

- **Compute data variance per channel**

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$

- **Normalization: Linear Transformation**

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

- **Rescale the normalized value to restore the representation power**

$$y_i = \gamma \hat{x}_i + \beta$$

# Backpropagation for Batch Normalization Layer

$$\frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \cdot \gamma$$

$$\frac{\partial l}{\partial {\sigma_B}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2} ({\sigma_B}^2 + \varepsilon)^{-3/2}$$

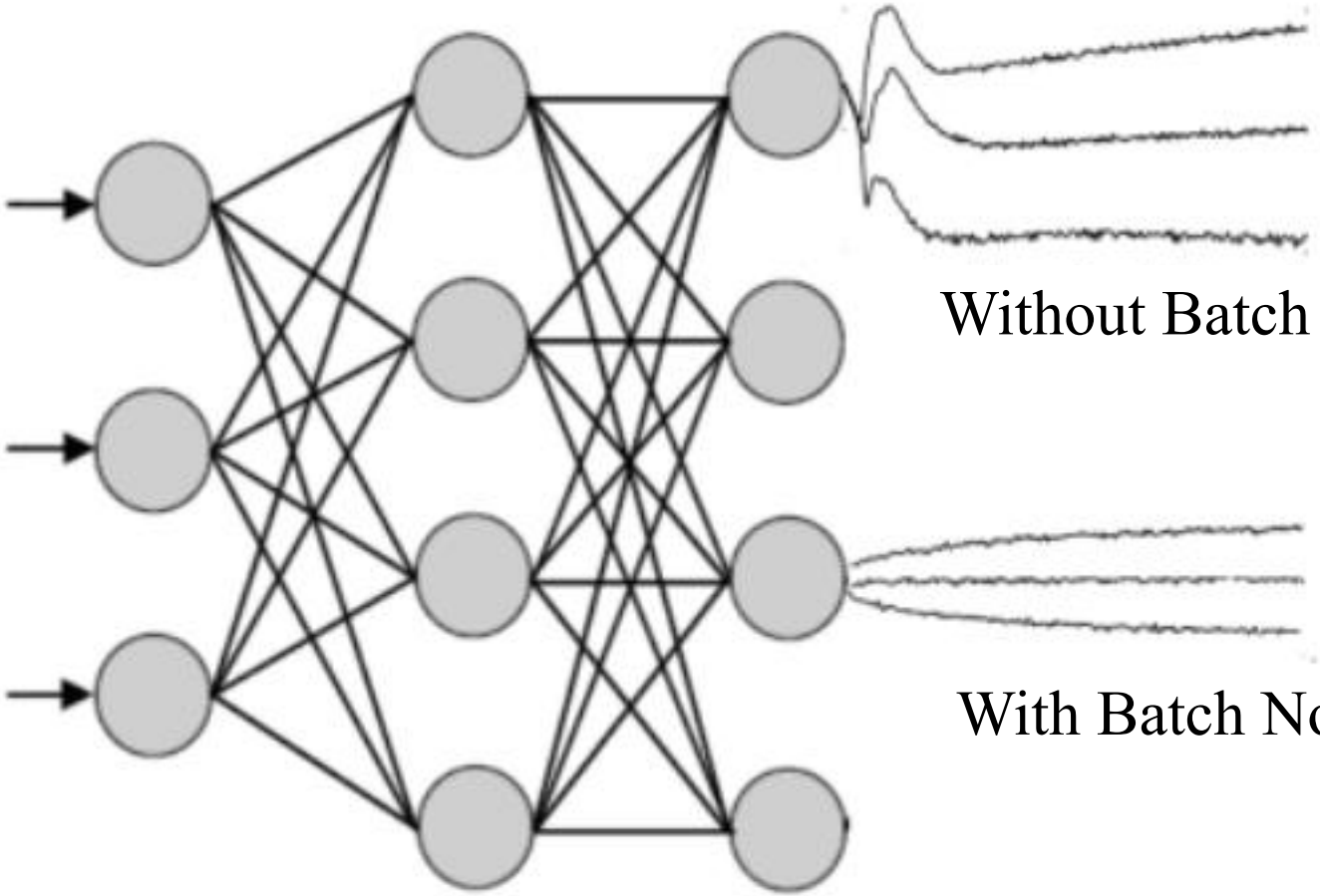
$$\frac{\partial l}{\partial \mu_B} = \left( \sum_{i=1}^{m} \frac{\partial l}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{{\sigma_B}^2 + \varepsilon}} \right) + \frac{\partial l}{\partial {\sigma_B}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_B)}{m}$$

$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{{\sigma_B}^2 + \varepsilon}} + \frac{\partial l}{\partial {\sigma_B}^2} \cdot \frac{2(x_i - \mu_B)}{m} + \frac{\partial l}{\partial \mu_B} \cdot \frac{1}{m}$$

$$\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i} \cdot \partial \hat{x}_i$$

$$\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i}$$

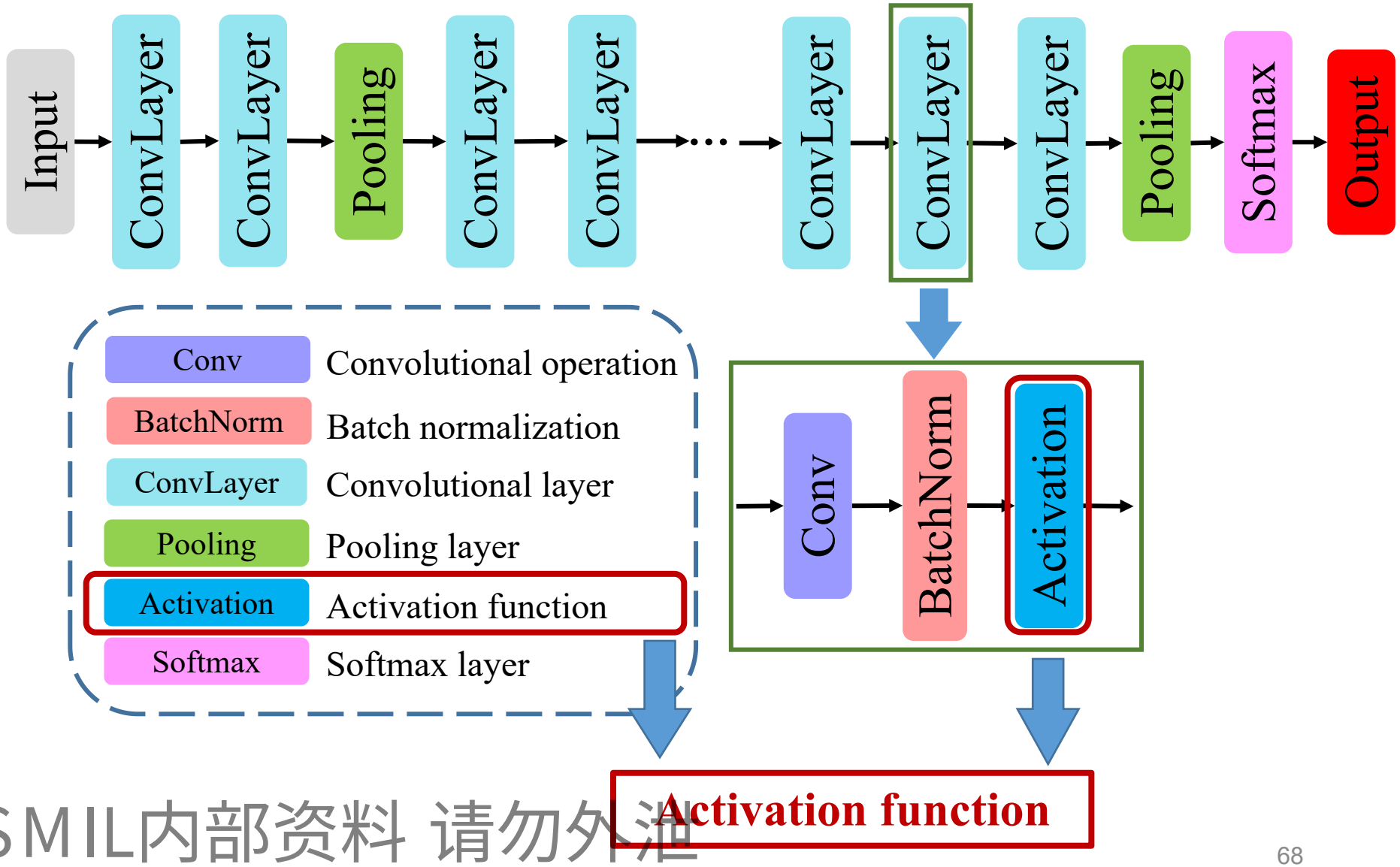
# Effect of Batch Normalization



Without Batch Normalization

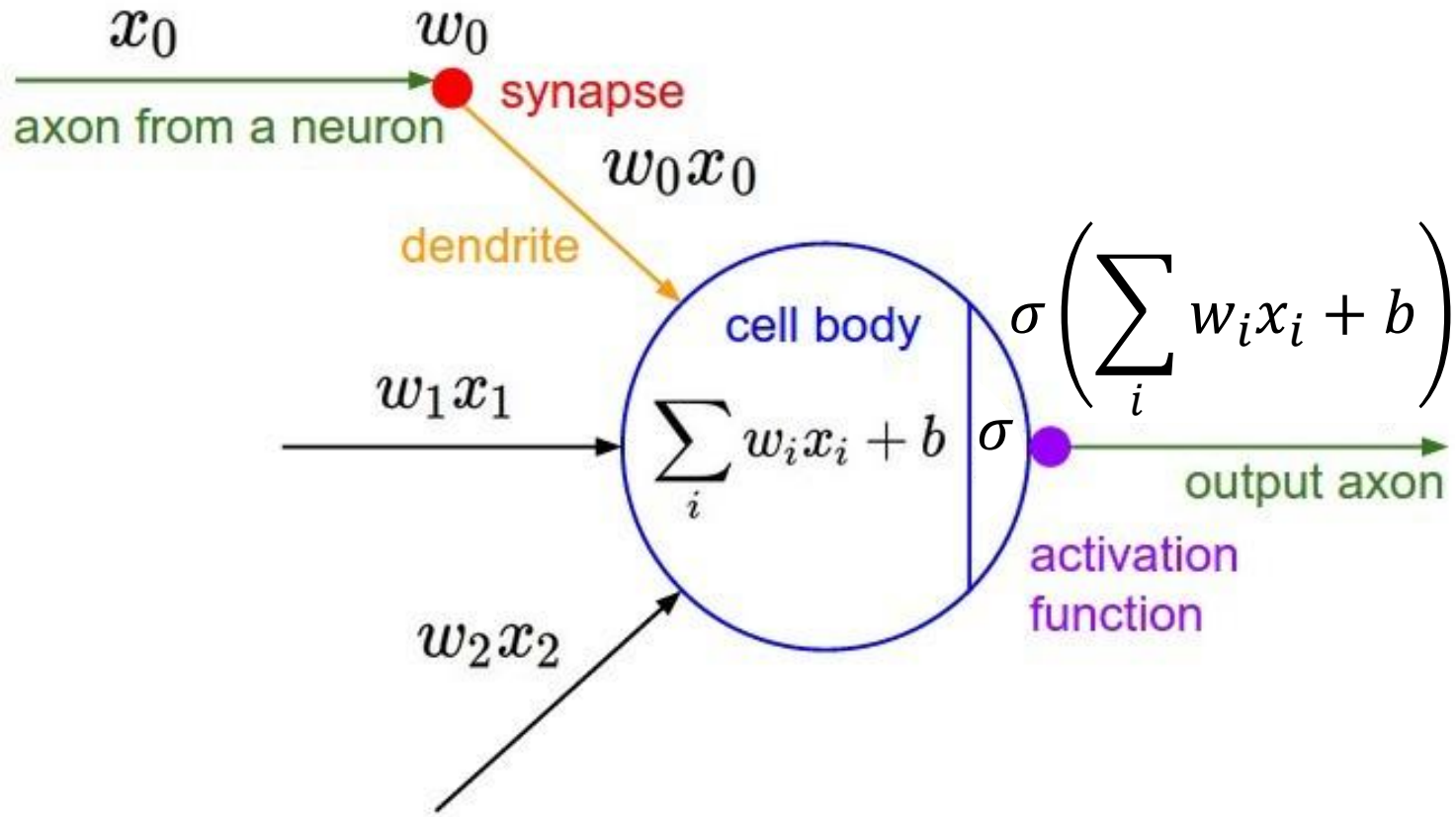
With Batch Normalization

# Activation Layer



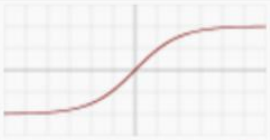
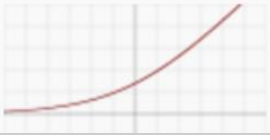
# Activation Function

- **Description**: keep value within an acceptable and useful range



# Activation Function

- Activation functions: **nonlinear** and **continuously differentiable**

| Activation Function | Plot | Equation | Derivative |
|---|---|---|---|
| Identity |  | $f(x) = x$ | $f'(x) = 1$ |
| Sigmoid |  | $f(x) = \frac{1}{1+e^{-x}}$ | $f'(x) = f(x)\big(1 - f(x)\big)$ |
| Tanh |  | $f(x) = \frac{2}{1+e^{-2x}} - 1$ | $f'(x) = 1 - f(x)^2$ |
| ReLU |  | $f(x) = \max(x,\ 0)$ | $f'(x) = 1\{x \geq 0\}$ |
| SoftPlus |  | $f(x) = ln(1 + e^{x})$ | $f'(x) = \frac{1}{1+e^{-x}}$ |

# Rectified Linear Unit (ReLU)

- **ReLU:**

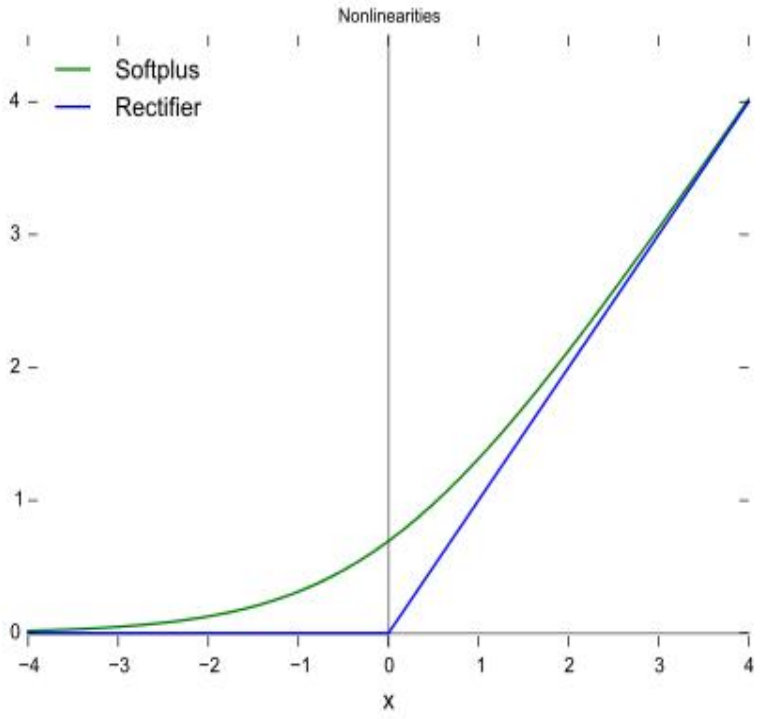
**Forward** $y = \max(x, 0)$

**Backward** $\frac{\partial y}{\partial x} = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$

- **Softplus:**

**Forward** $y = \ln[1 + e^x]$

**Backward** $\frac{\partial y}{\partial x} = \frac{1}{1 + e^{-x}}$



# Leaky ReLU

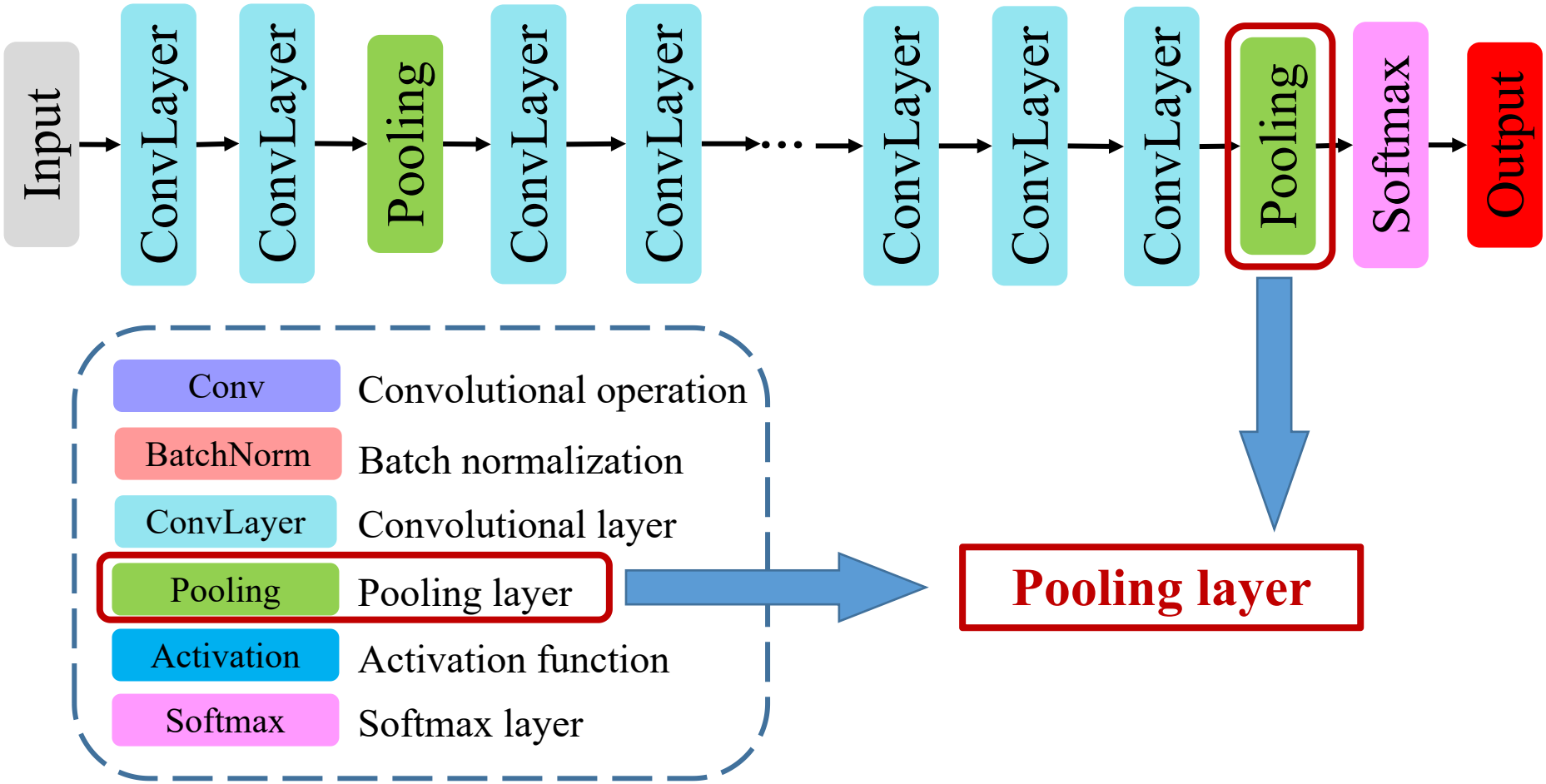
- Guarantee a **small** and **non-zero** gradient when the unit is not active.

**Forward** $y = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases} \Leftrightarrow y = \max(x, ax)$

where a<1

**Backward** $y = \begin{cases} 1 & \text{if } x > 0 \\ a & \text{otherwise} \end{cases}$

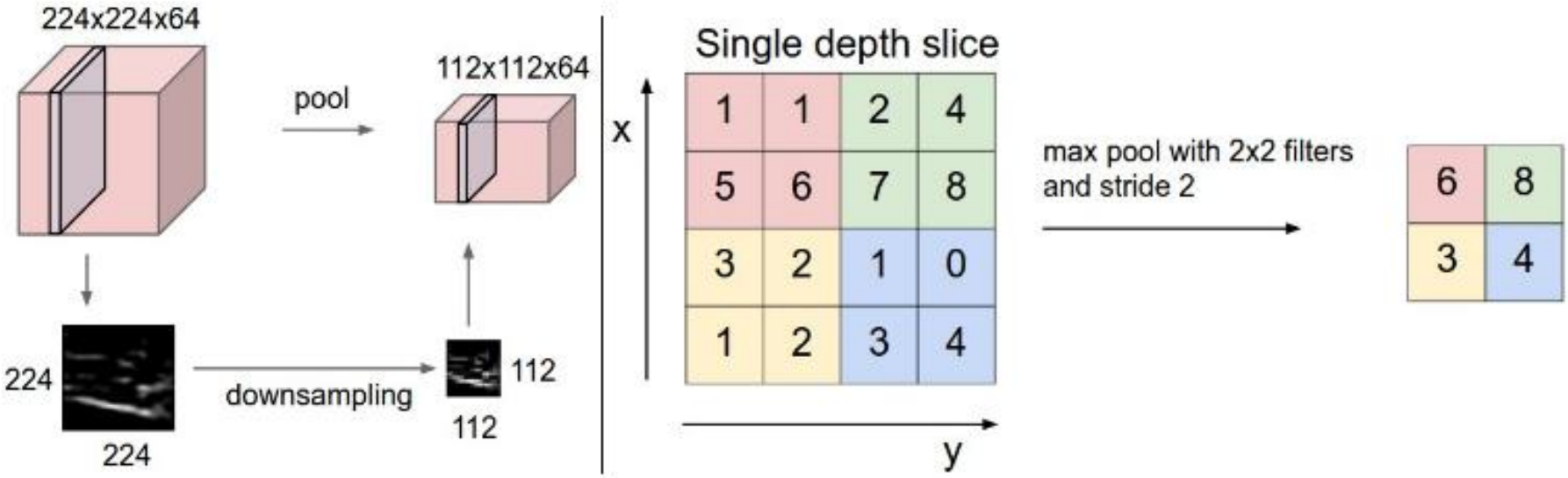
# Pooling Layer



# Pooling Layer

**Max pooling:**

- Max value of a cluster of neurons
- Nonlinear down-sampling



# Pooling Layer

Reduce **spatial size** of the representation and the **number of parameters**

**Forward**

- Input data size: $W_1 \times H_1 \times C_1$
- Output data size: $W_2 \times H_2 \times C_2$

$$W_2 = [(W_1 - F)/S + 1]$$

$$H_2 = [(H_1 - F)/S + 1]$$

$$C_2 = C_1$$

where $S$ is Spatial extent, $F$ is Stride, and C is the number of channel

# **Forward propagation** and **backpropagation** of Convolutional Neural Networks

# Prediction for CNN $f$ : Forward Propagation

- Given input data $\mathbf{X}$ and a convolutional neural network $f$, forward propagation is to compute the **prediction** $\hat{\mathbf{y}}$ of $\mathbf{X}$, namely

$$\hat{\mathbf{y}} = f(\mathbf{X})$$

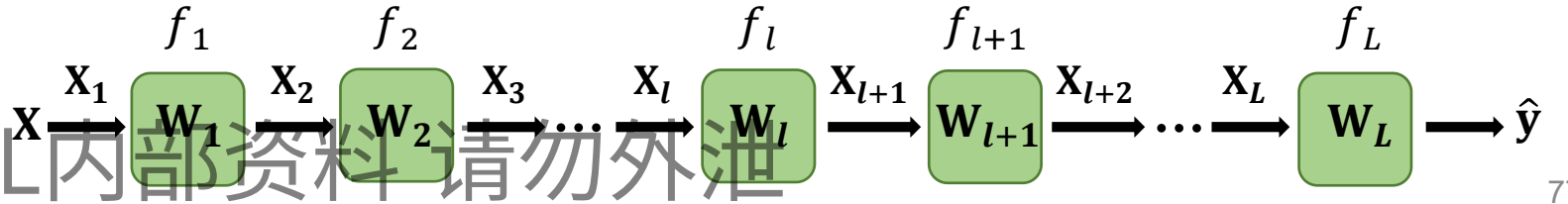
- Detailed prediction process:

$$\begin{aligned}
\mathbf{X}_1 &= \mathbf{X} \\
\mathbf{X}_2 &= f_1(\mathbf{X}_1, \mathbf{W}_1) \\
&\vdots \\
\mathbf{X}_{l+1} &= f_l(\mathbf{X}_l, \mathbf{W}_l) \\
&\vdots \\
\mathbf{X}_L &= f_{L-1}(\mathbf{X}_{L-1}, \mathbf{W}_{L-1}) \\
\hat{\mathbf{y}} &= f_L(\mathbf{X}_L, \mathbf{W}_L)
\end{aligned}$$

$f_l = \sigma(h_l(\mathbf{X}_l, \mathbf{W}_l))$

$\sigma$ is some activation function

$h_l = \mathbf{W}_l * \mathbf{X}_l$ is the Convolutional Operation—a linear operation



# How to Learn Parameters $\{\mathbf{W}_l\}$ in CNNs?

- We minimize the following regularized loss function:

**Regularized Loss Function:**

$$\min_{\mathbf{W}} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_W(\mathbf{y}_i, \hat{\mathbf{y}}_i) + \frac{1}{2}\lambda\|\mathbf{W}\|_F^2$$

$\mathbf{W}$ – parameters $N$ – number of samples $\mathcal{L}$ – loss function

$\mathbf{y}_i$ – $i^{th}$ ground-truth label $\hat{\mathbf{y}}_i$ – $i^{th}$ predicted label

- To minimize the loss, we update the parameters $\mathbf{W}_l$ by using **(Stochastic) Gradient Descent**:

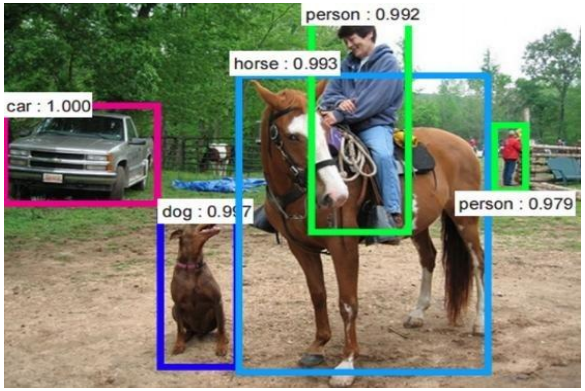
$$\mathbf{W}_l' = \mathbf{W}_l - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} \quad (2)$$

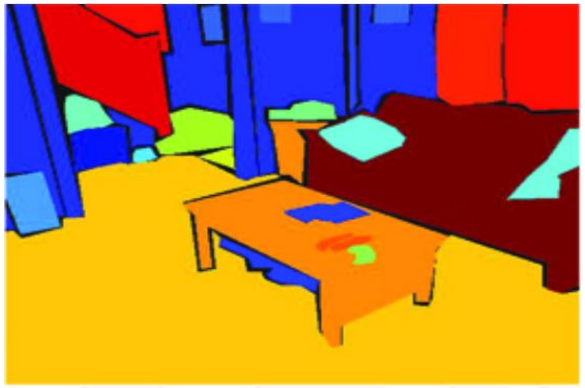
where $\eta$ is the learning rate and gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$ is computed by **backpropagation**.

# Contents

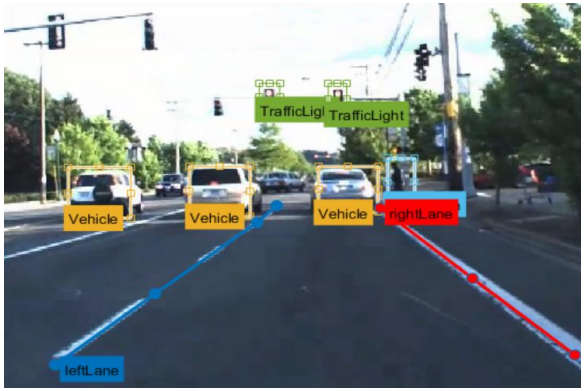
# Deep learning Applications

- Image Classification; Object Detection; Image Segmentation

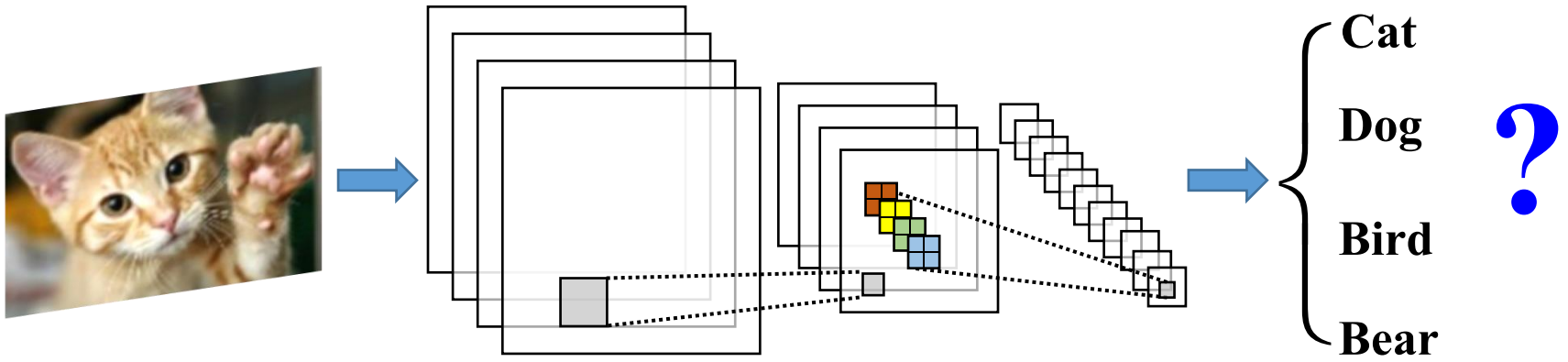


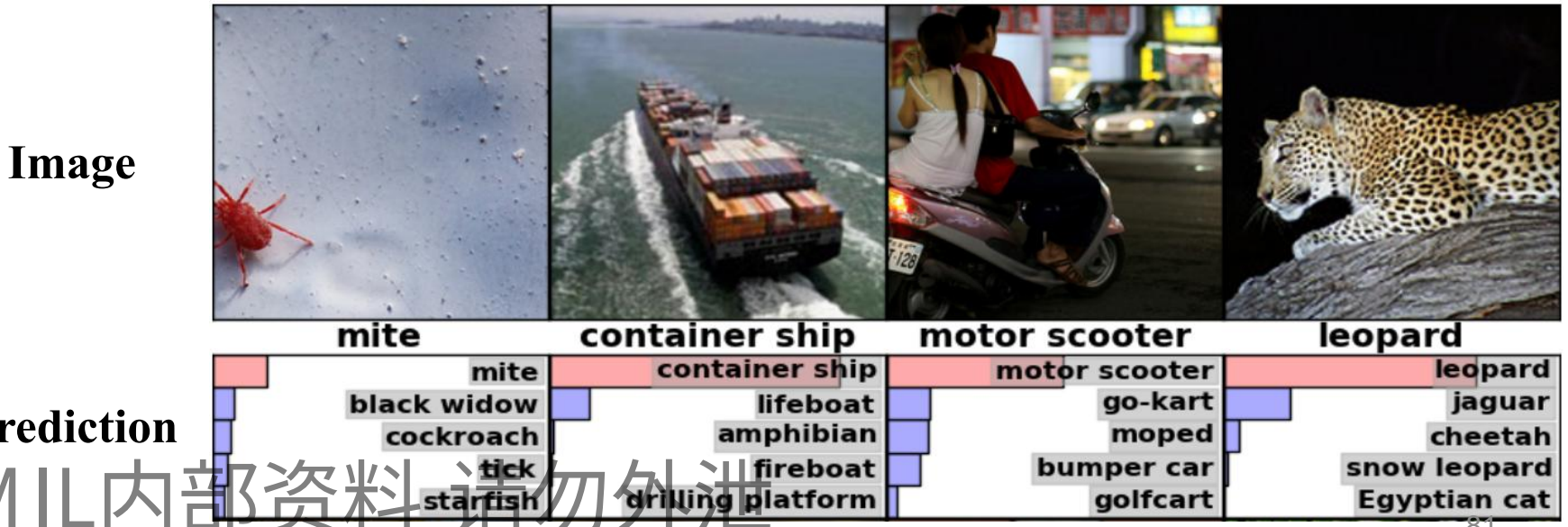



- Pedestrian Detection; Lane Detection; Traffic Sign Recognition

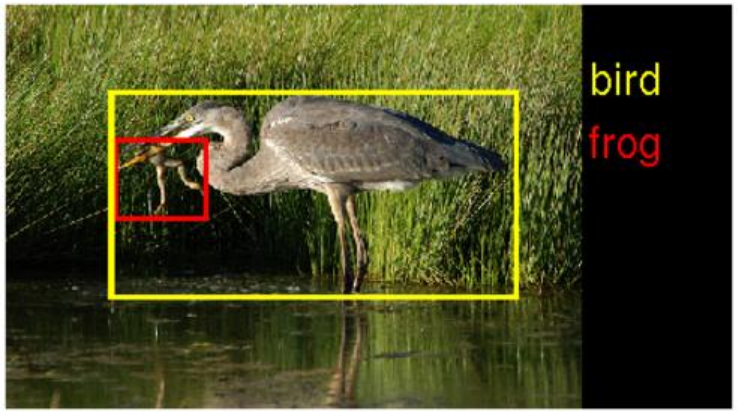



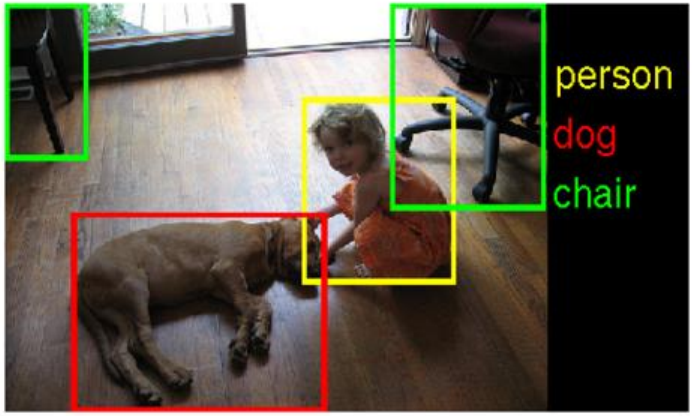
# Image Recognition



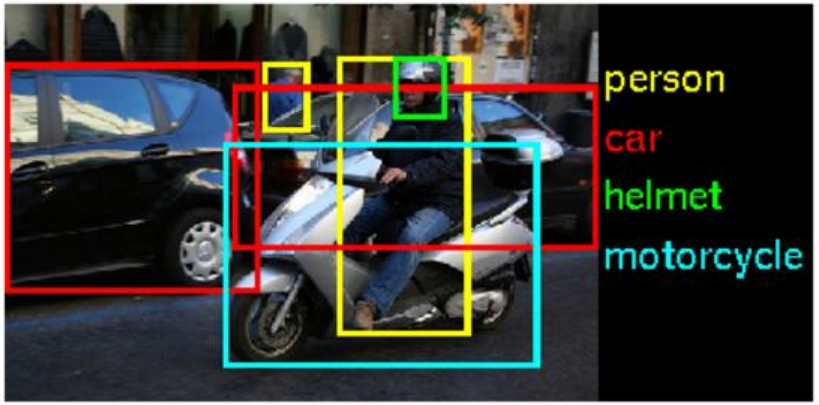



# Object Detection



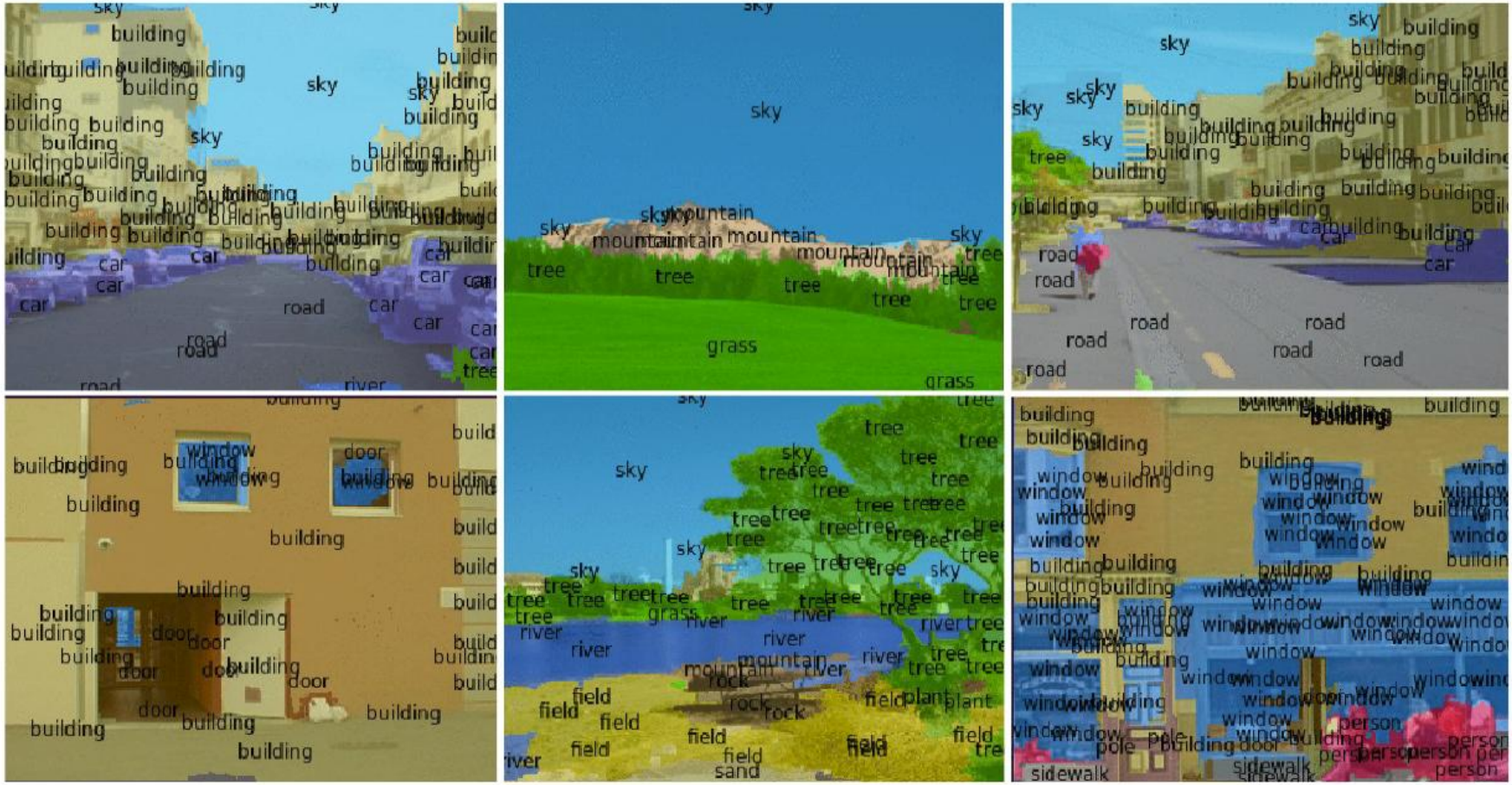







# Semantic Labeling

- Labeling every pixel with the object category it belongs to



# Scene Parsing

- Each output sees a large input context

# Other Recognition Applications

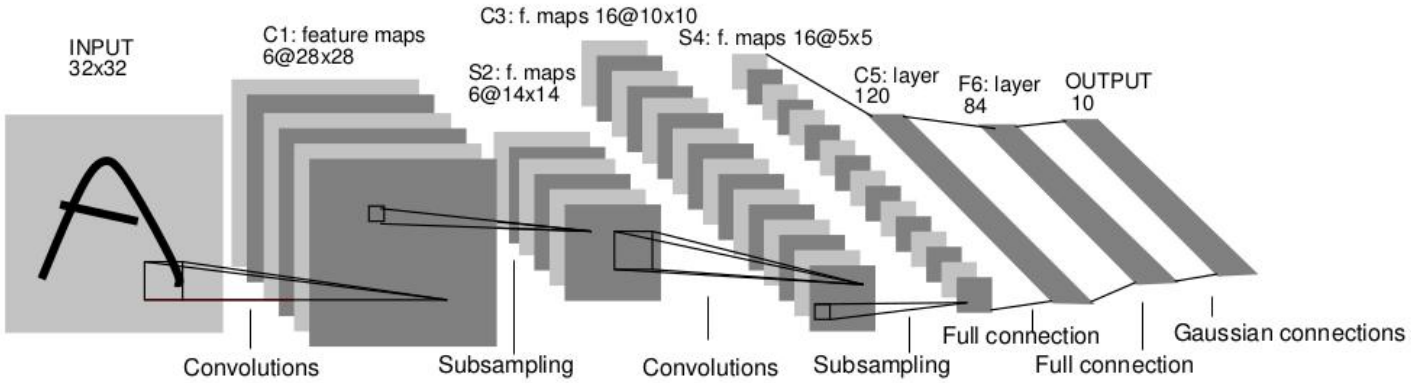
- Traffic Sign Recognition (GTSRB)
  - German Traffic Sign Reco Bench
  - 99.2% accuracy

- House Number Recognition (Google)
  - Street View House Numbers
  - 94.3 % accuracy

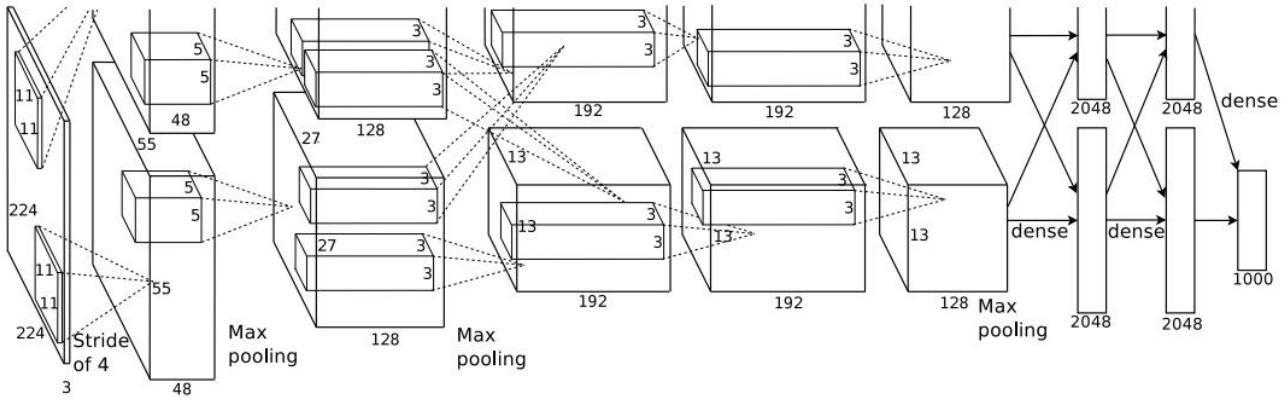
# Classical CNNs

# Classical CNNs

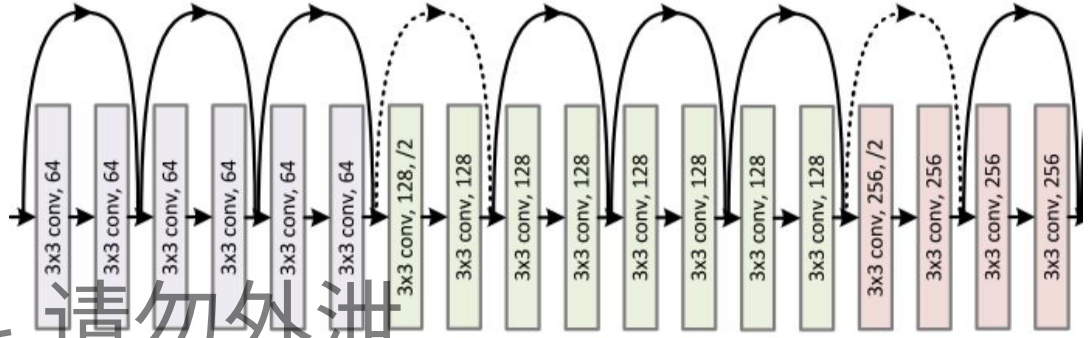
LeNet



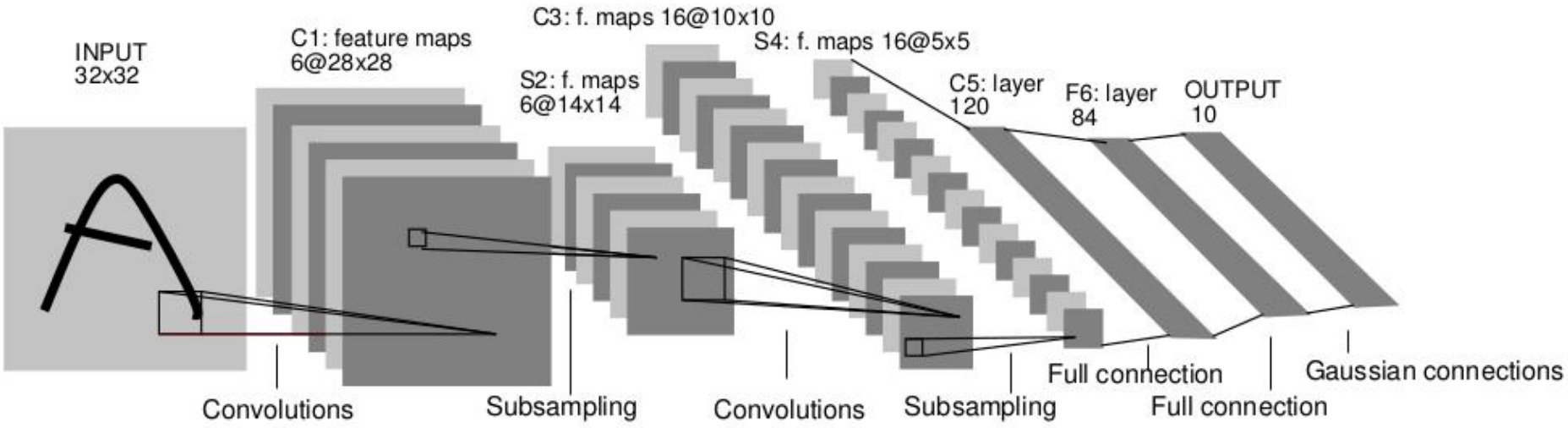

AlexNet



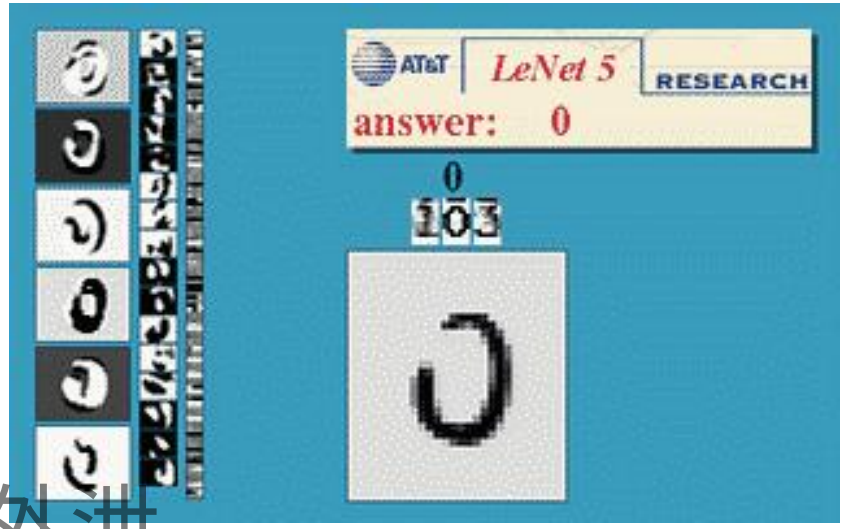

ResNet



# LeNet

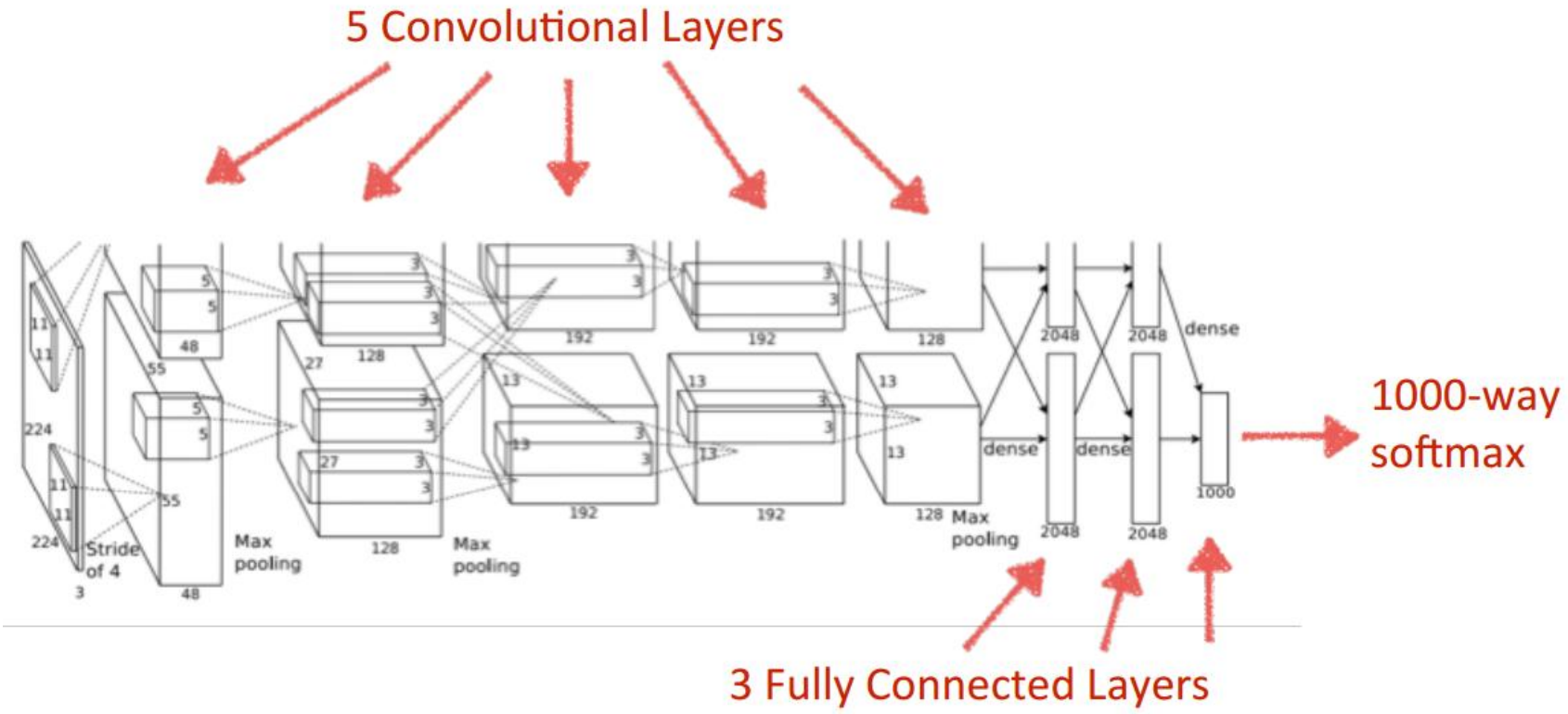



- **Handwriting number recognition**

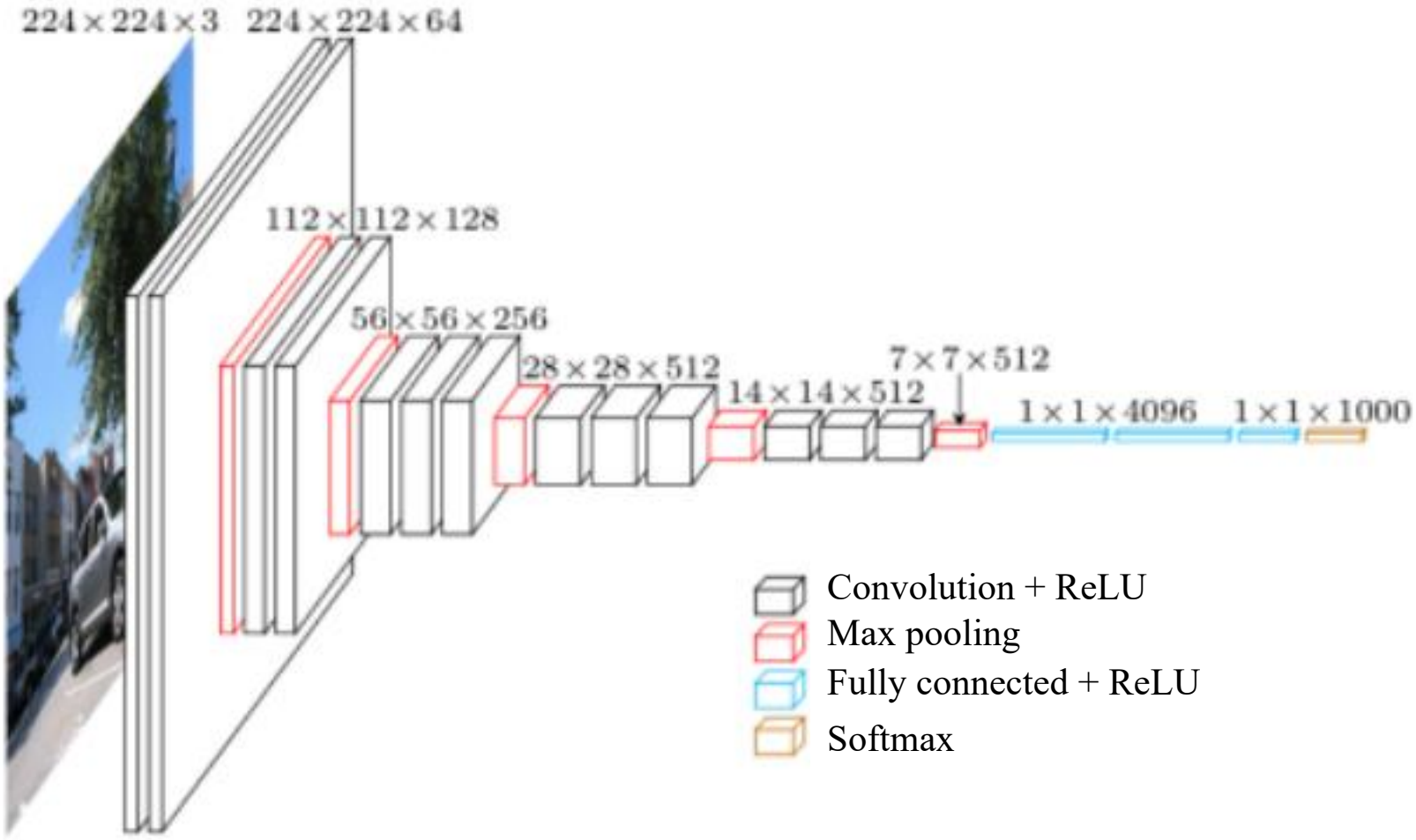


# AlexNet

- Trained the network on ImageNet.
- Used ReLU for the nonlinearity functions
- Implemented dropout layers to combat overfitting issue



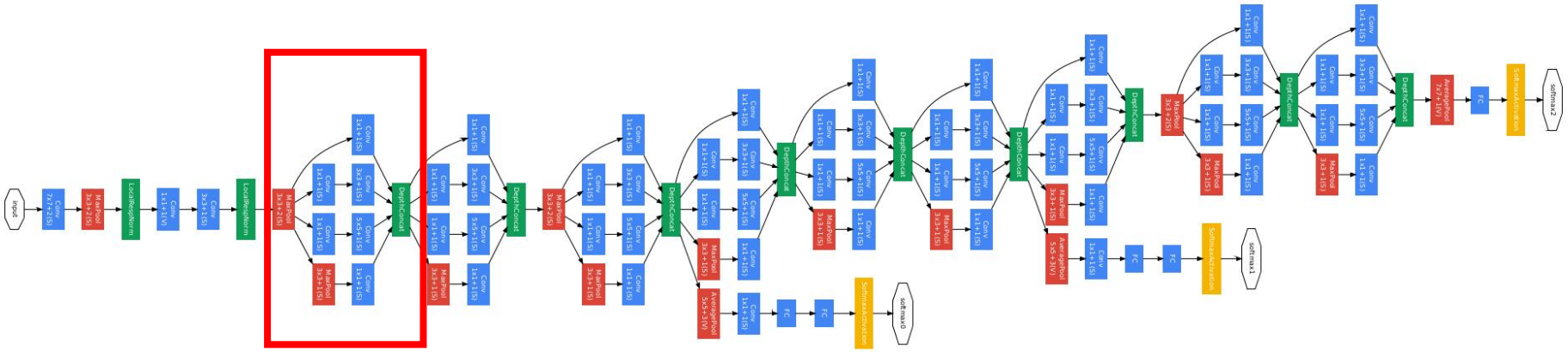
# VGG



# Six Different Architectures of VGG Net

| 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 imag ) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

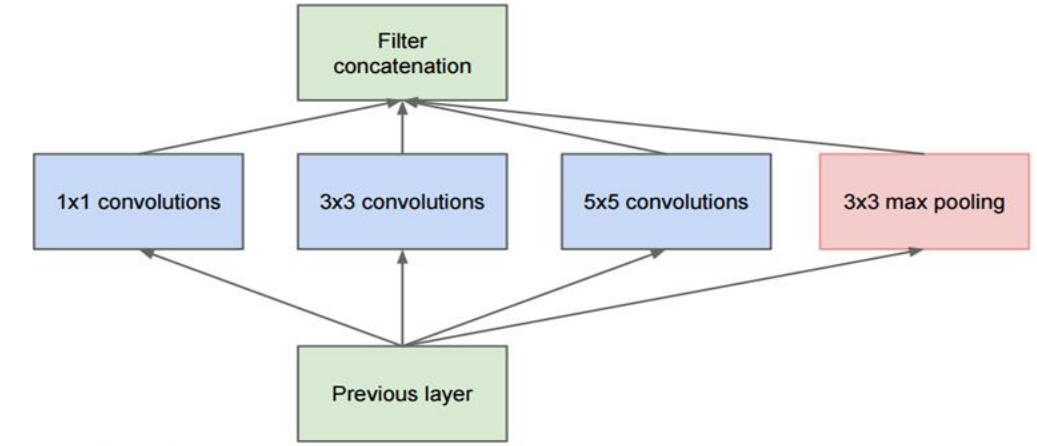
# GoogLeNet



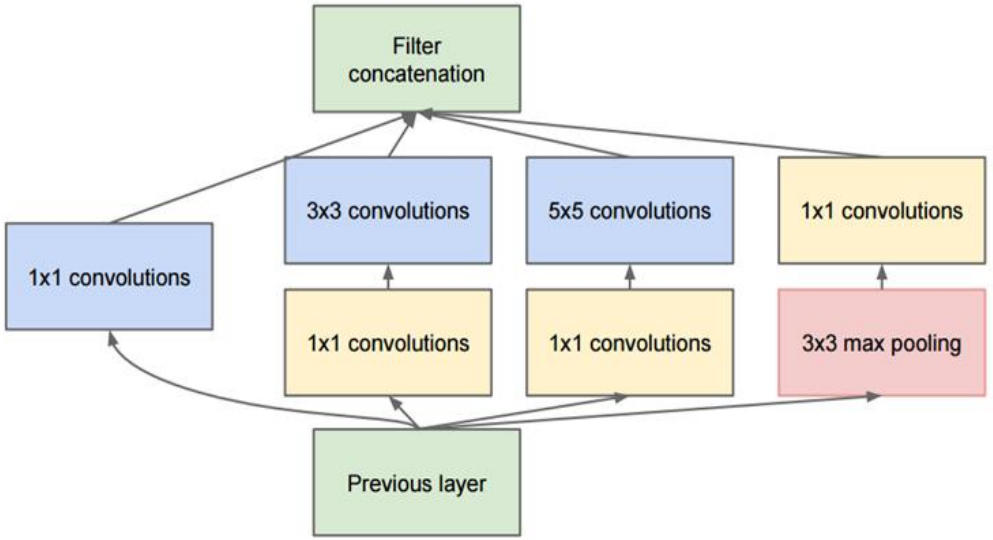
- Used 9 Inception modules, over **100** layers (very deep)
- Uses 12x **fewer parameters** than AlexNet
- Trained on **multiple GPUs** within a week

# Inception Module

- Basic Inception Module

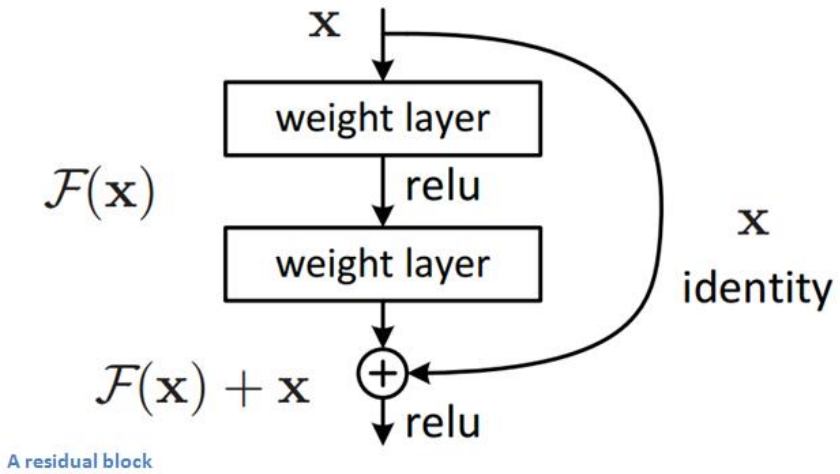



- Full Inception Module




Full inception module

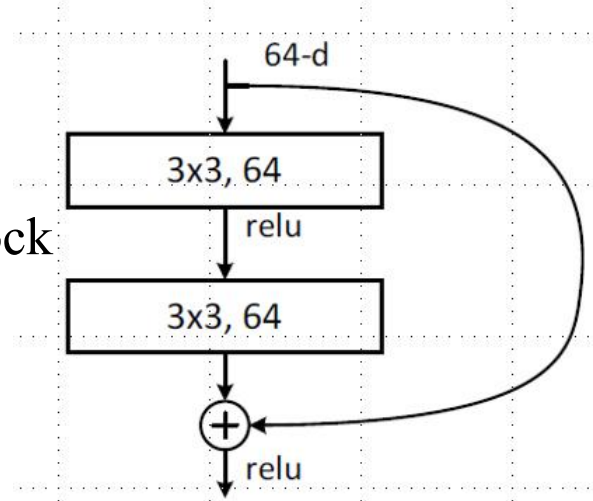
# Residual Network



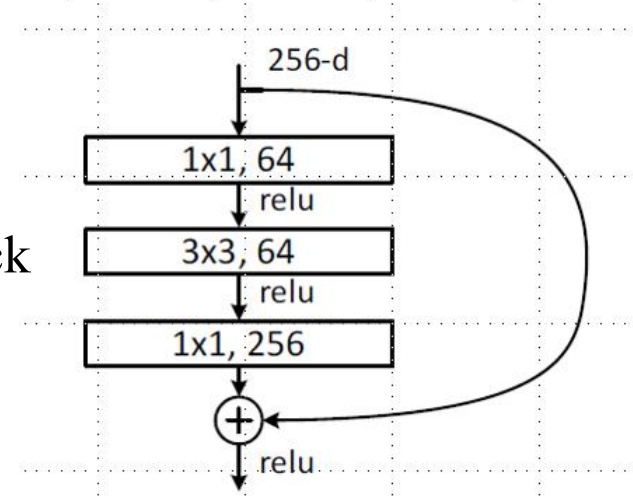

$$y = \mathcal{F}(x) + x$$

Residual Term

Basic Block

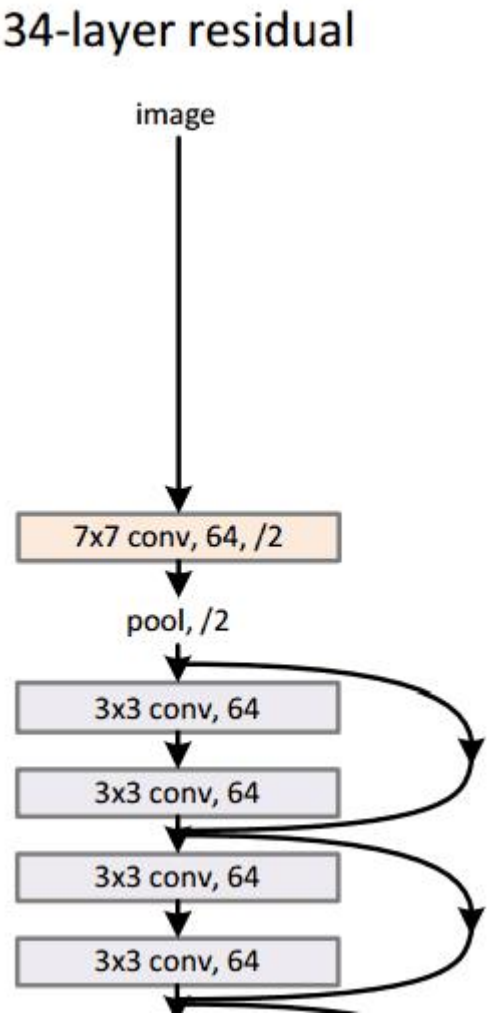



Bottleneck



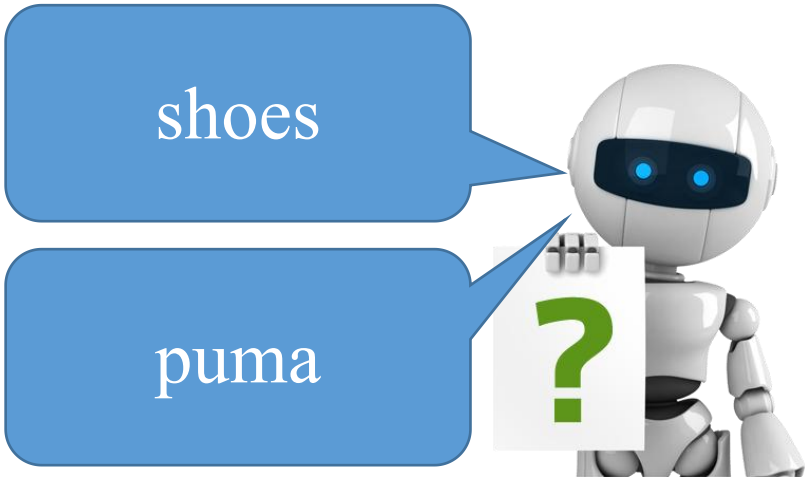
# Residual Network

- **Residual** learning
- **Ultra-deep** 152 and 200 layers
- Trained a **1202-layer** network
- Trained on an **8 GPUs** machine



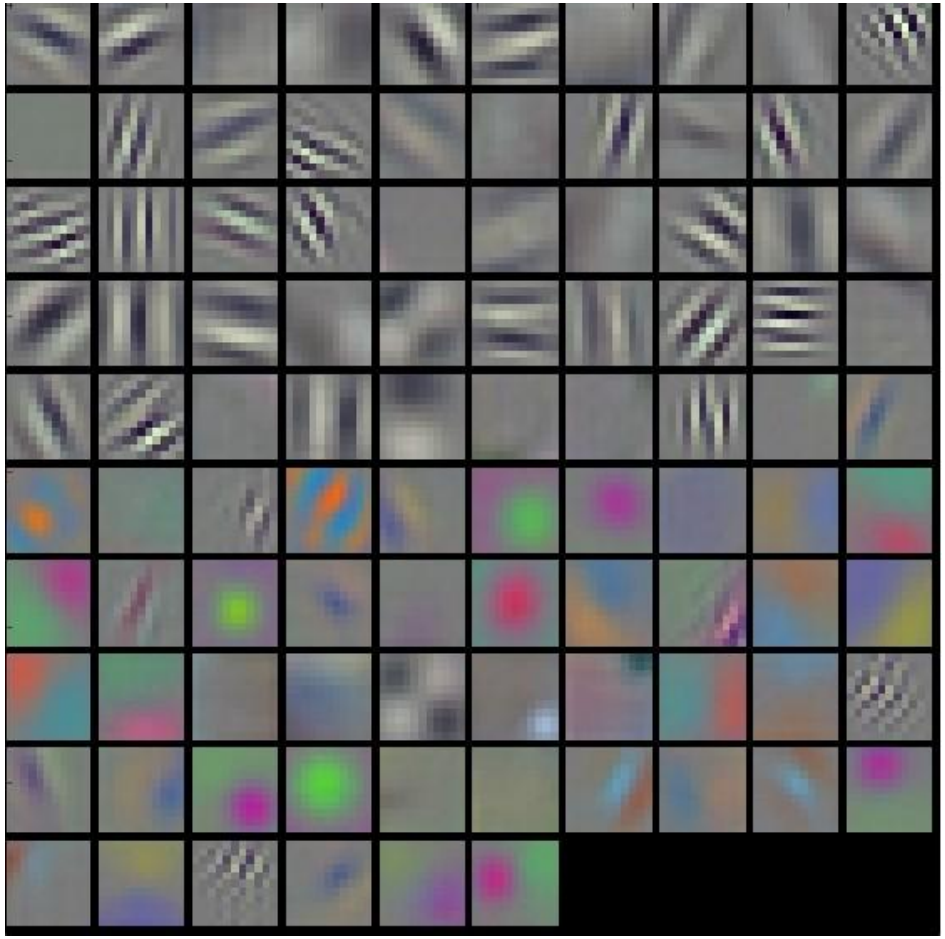
# What does machine learn in CNNs?

# What does Machine Learn?

# First Convolutional Layer

- Typical-looking filters on the trained first layer

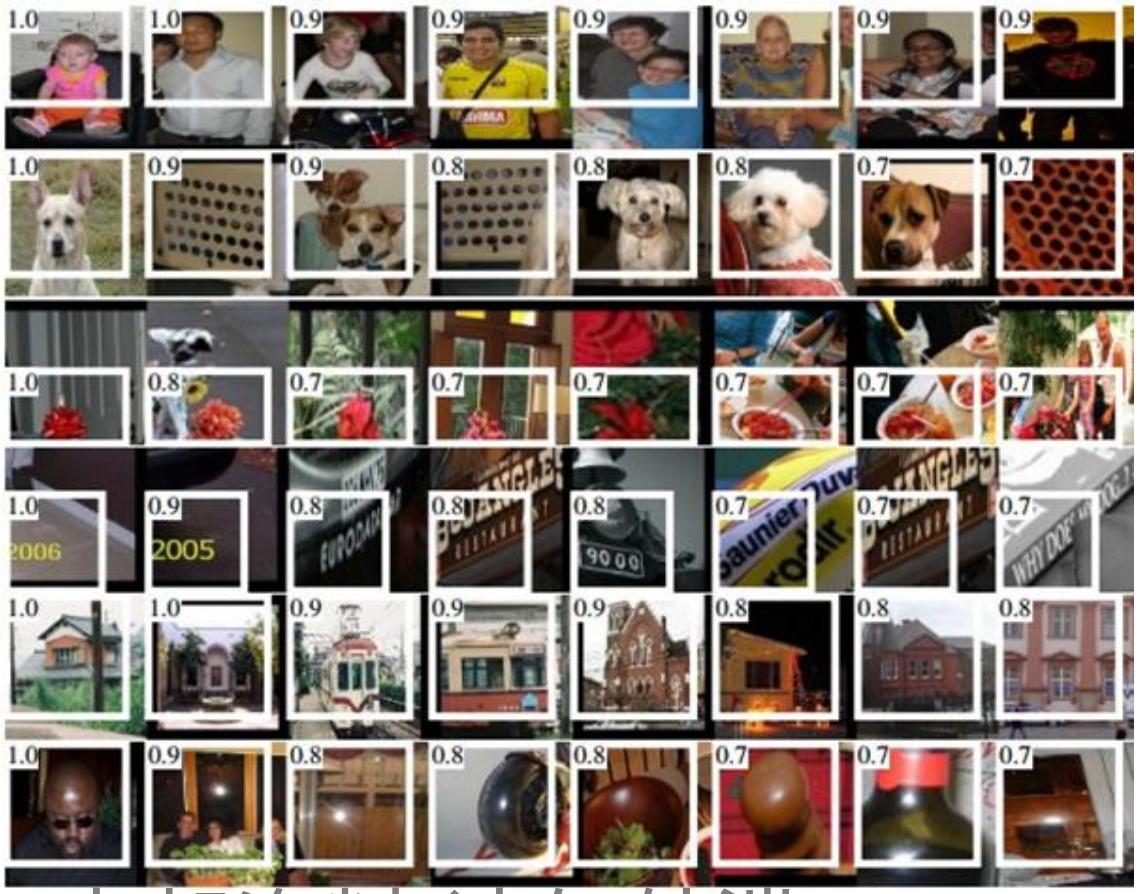


11 x 11
(AlexNet)

http://cs231n.github.io/understanding-cnn/

# How about Higher Layers?

- Which images make a specific neuron activate?

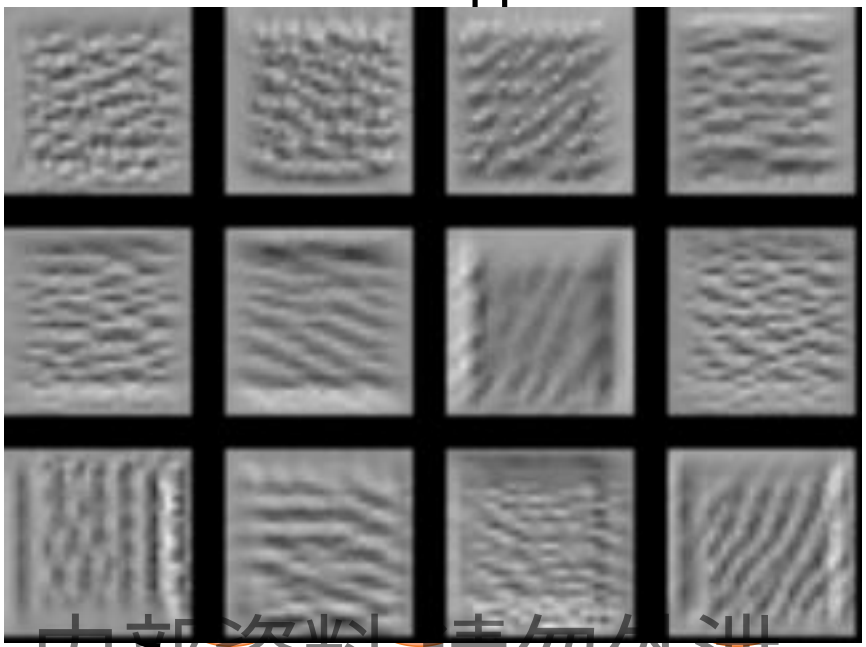


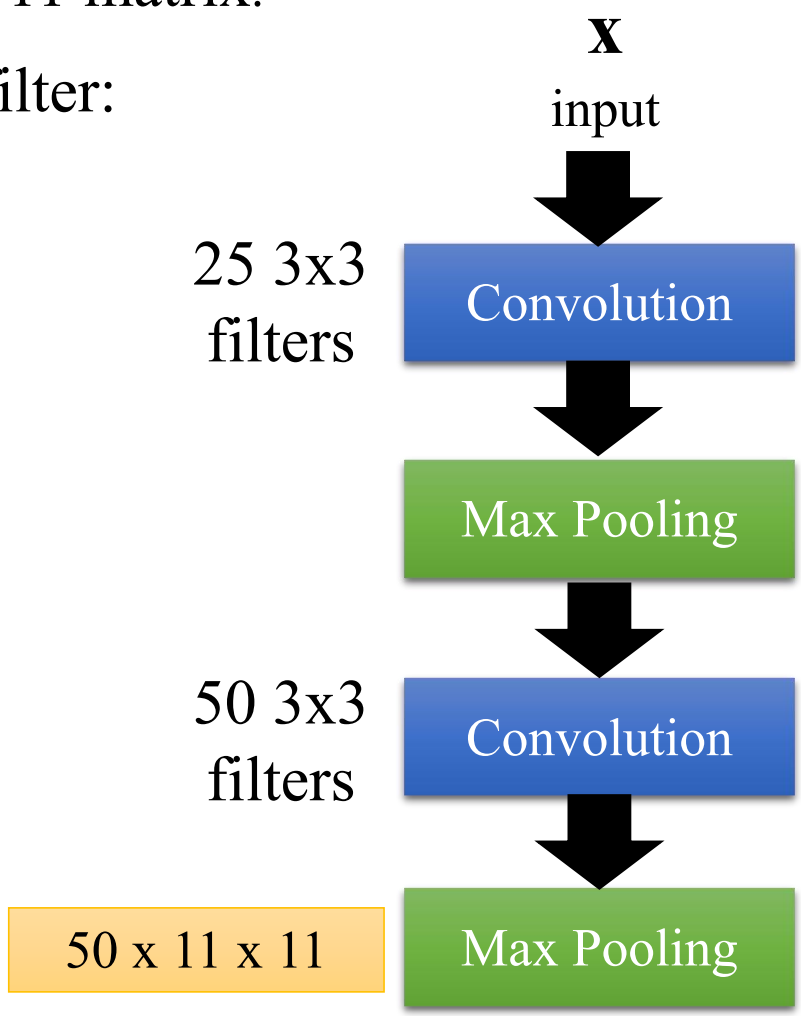
Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR, 2014

# What does Machine Learn?

- The output of the k-th filter is a 11 x 11 matrix.
- Degree of the activation of the k-th filter:

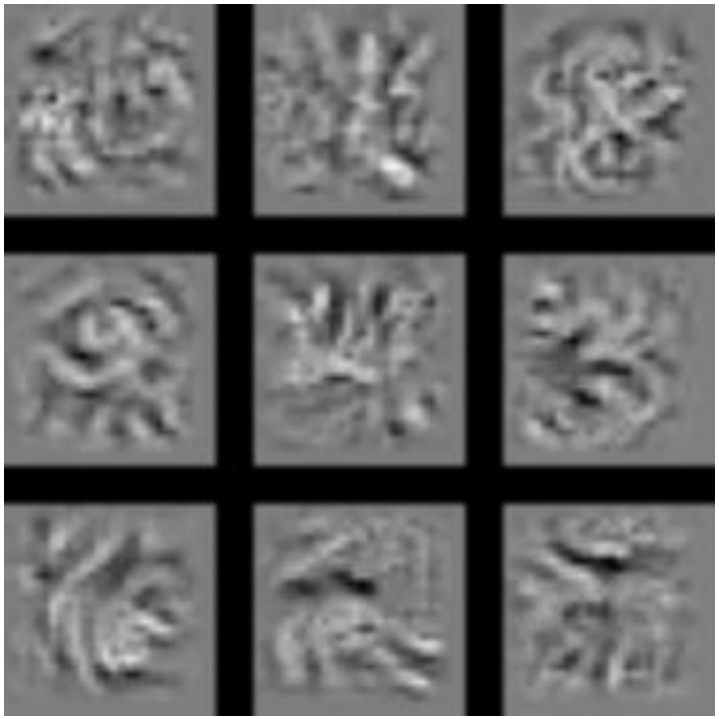
$$a^k = \sum_{i=1}^{11} \sum_{j=1}^{11} a_{ij}^k$$



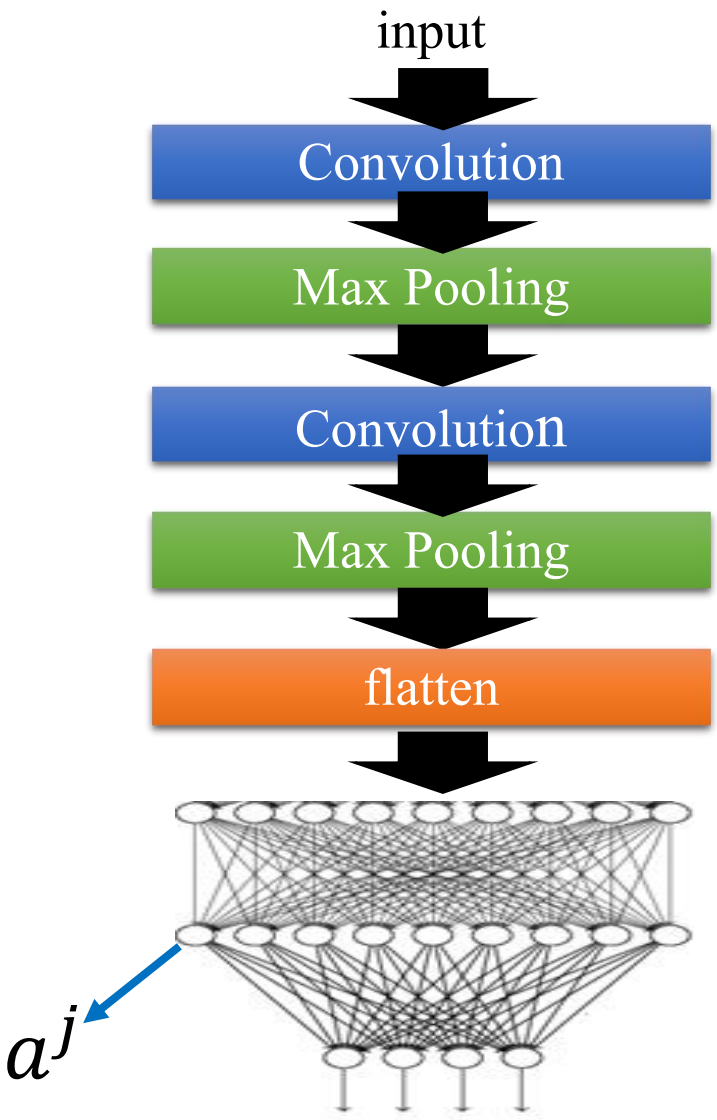


# What does Machine Learn?

- Find an image maximizing the output of neuron: $\mathbf{x}^* = \arg\max_{\mathbf{x}} a^j$

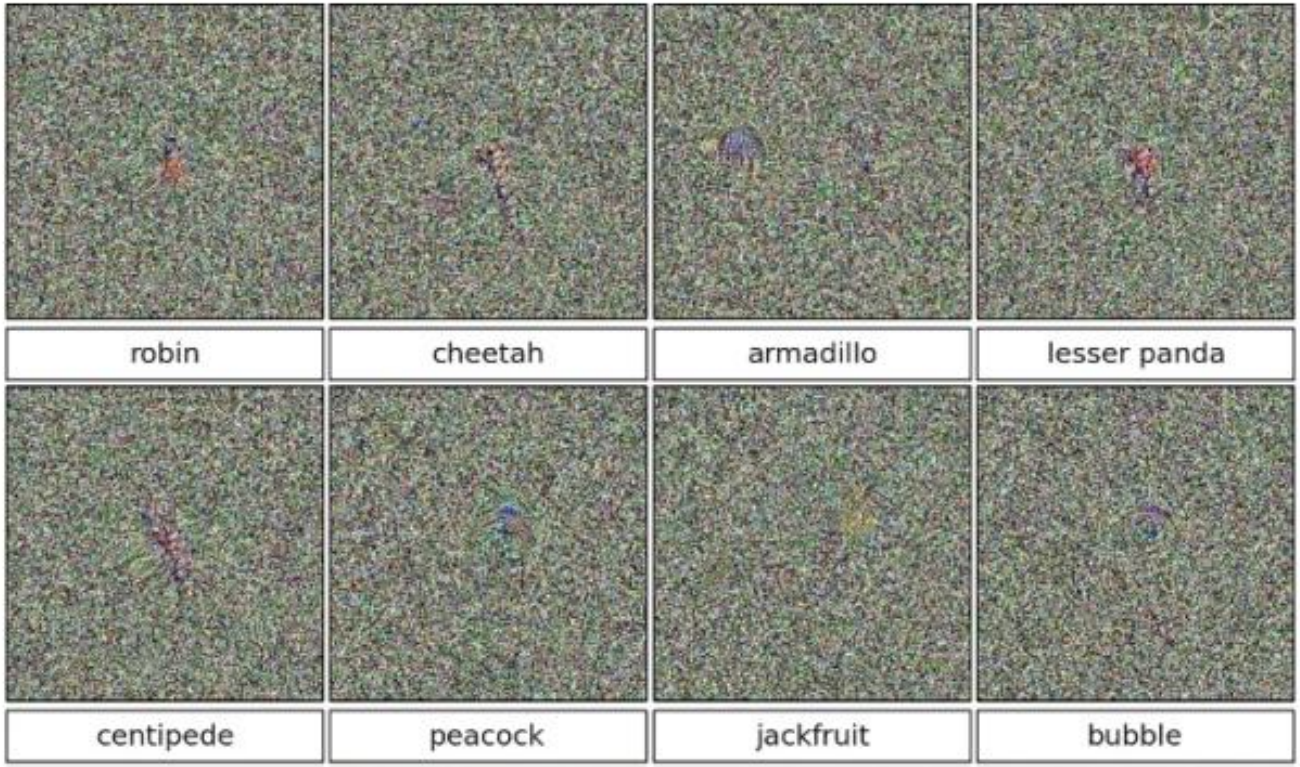


- Each figure corresponds to a neuron



# Will Deep Neural Networks be Fooled?

- Evolved images that are unrecognizable to humans but DNNs can recognize



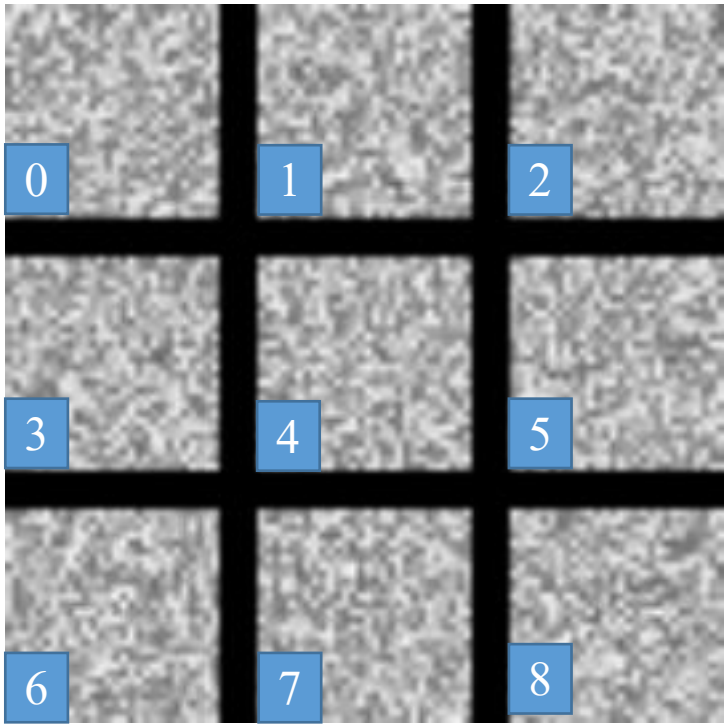

(Deep Neural Networks are Easily Fooled
https://www.youtube.com/watch?v=M2IebCN9Ht4)

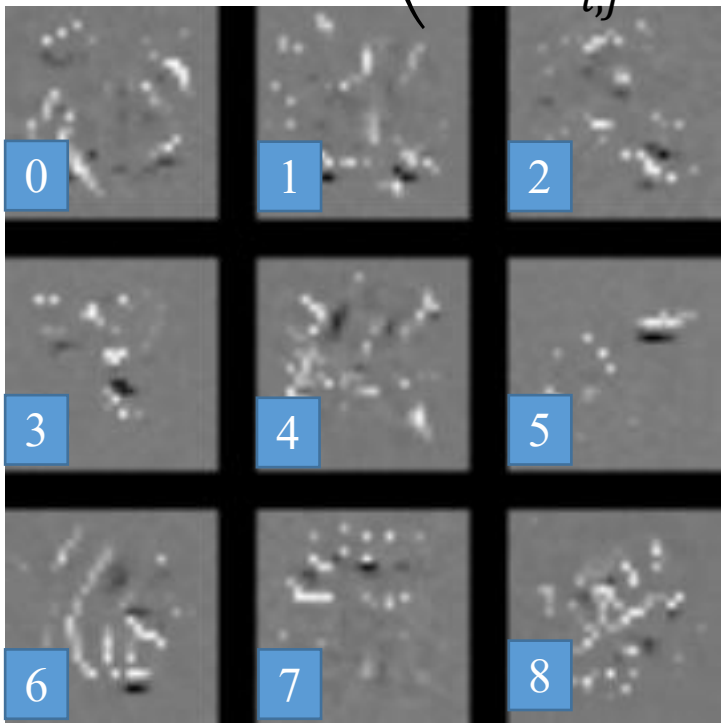
# What does Machine Learn?

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} y^i$$

Over all pixel values

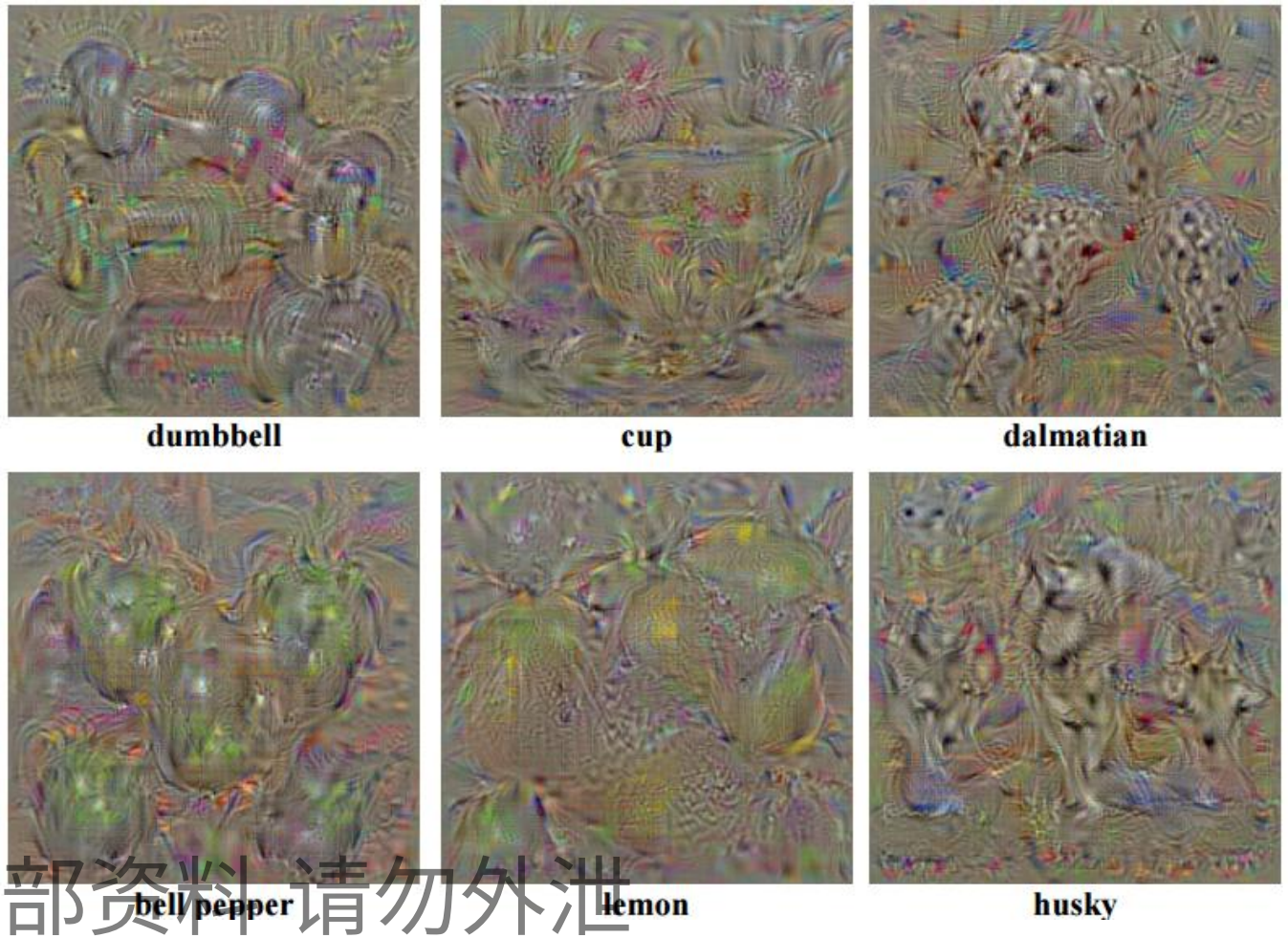
$$\mathbf{X}^* = \arg\max_{X} \left( y^i - \sum_{i,j} |\mathbf{X}_{ij}| \right)$$



# What does Machine Learn?

- Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR, 2014

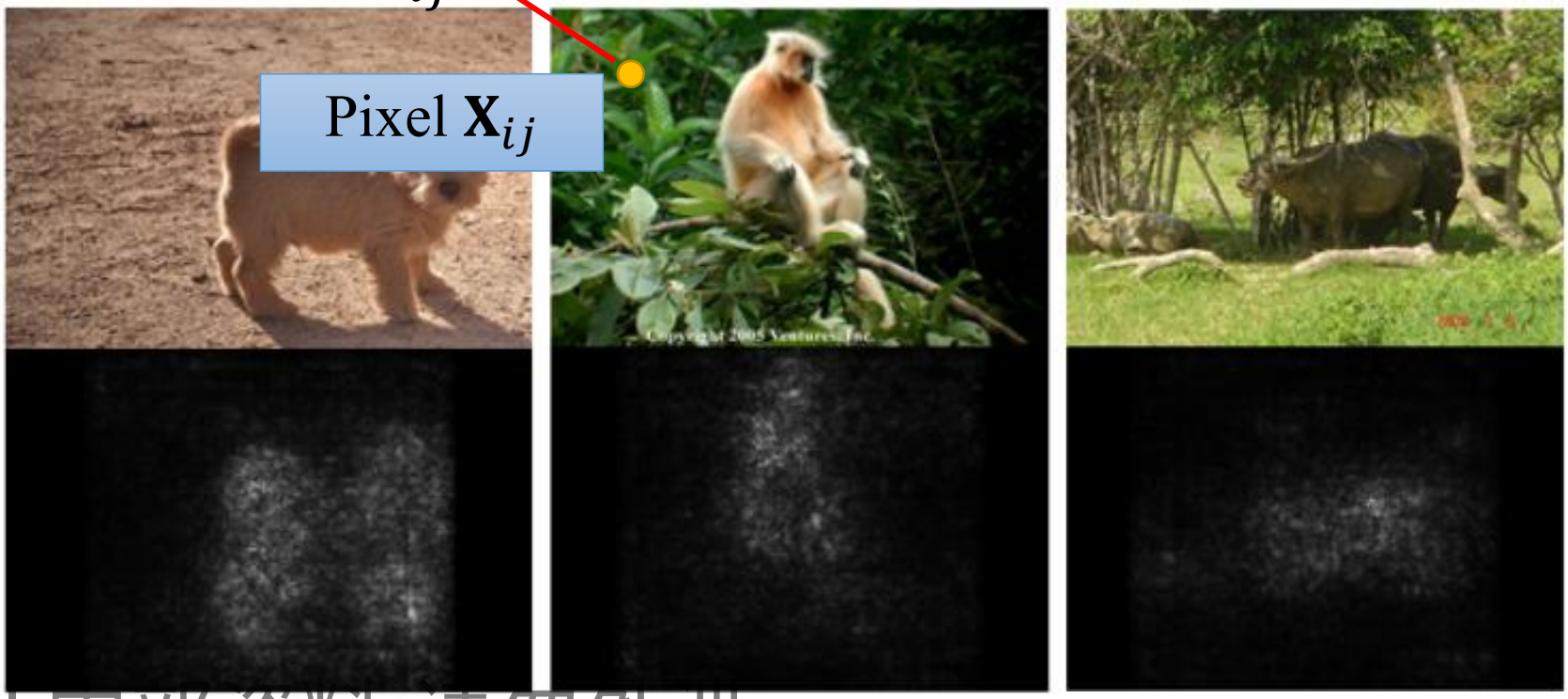


# What does Machine Learn?

- Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR, 2014

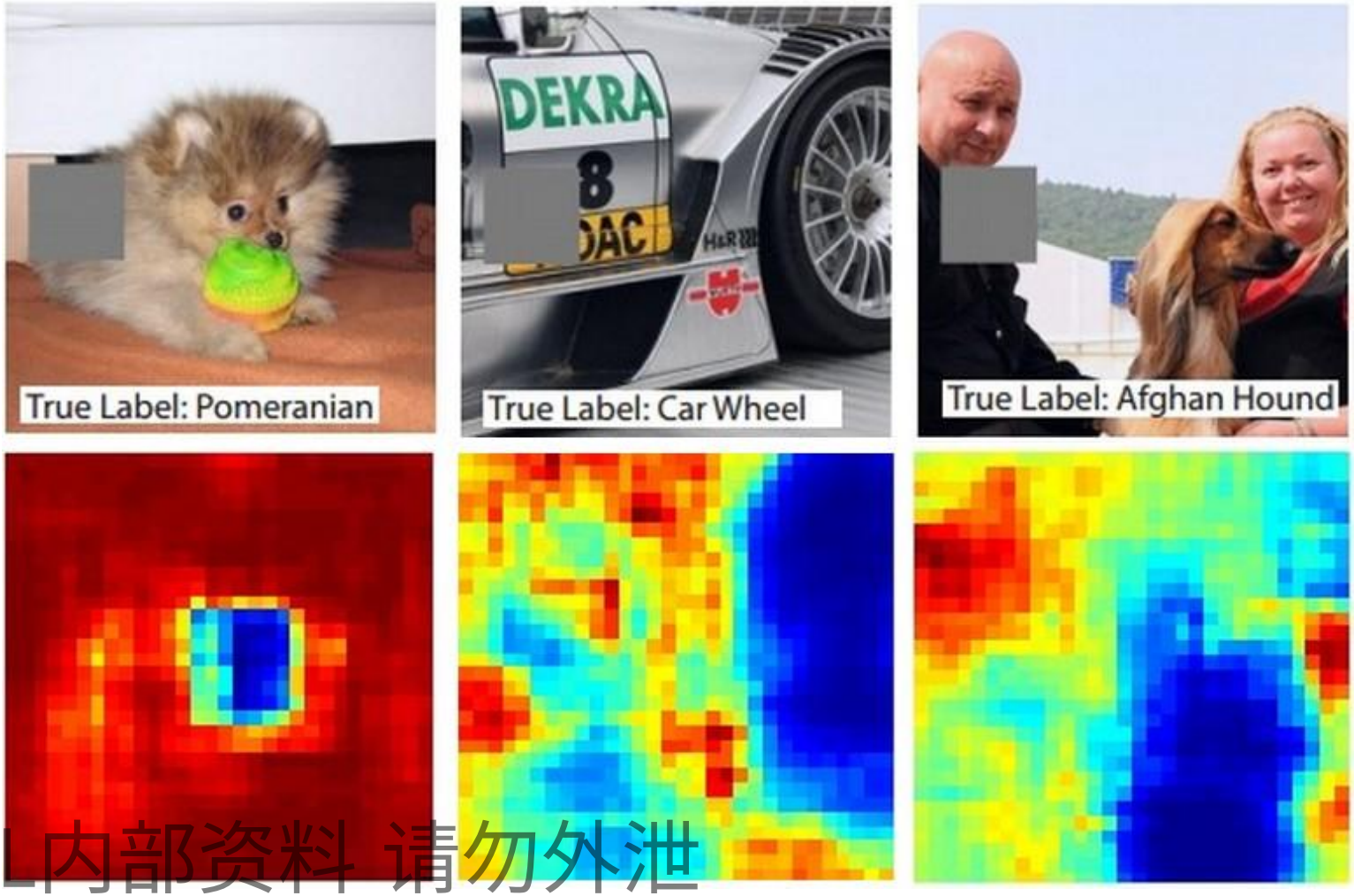
$$\left|\frac{\partial y_k}{\partial X_{ij}}\right|$$

$y_k$: the predicted class of the model



# What does Machine Learn?

- Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818-833)



Thank you