

본 수업자료는 2025년도 과학기술 정보통신부 및 정보통신기획평가원의 ‘SW중심대학사업’ 지원을 받아 제작 되었습니다.

# ComputerVision

Week4 – 5

2025-2

Mobile Systems Engineering

Dankook University

# Why Go Deeper?

---

## ■ The Promise of Depth in Deep Neural Networks

- Deep neural networks have revolutionized visual recognition.
- More layers = Better representations:
  - Capture low-level (edges), mid-level (textures), and high-level (objects) features.
- Empirically proven
  - VGG, GoogLeNet: depth correlates with improved performance on ImageNet and COCO.



**BUT... deeper networks are hard to train!**

# The Optimization Challenges

---

## ■ When More Layers Make Things Worse

### • 1. Vanishing/Exploding Gradients

- As gradients are backpropagated, they can diminish (vanish) or blow up (explode).
- Leads to unstable or slow training.

### • 2. Saturation of Accuracy

- Adding layers sometimes leads to **higher training error**, not just test error.
- Known as the “**degradation problem**”.

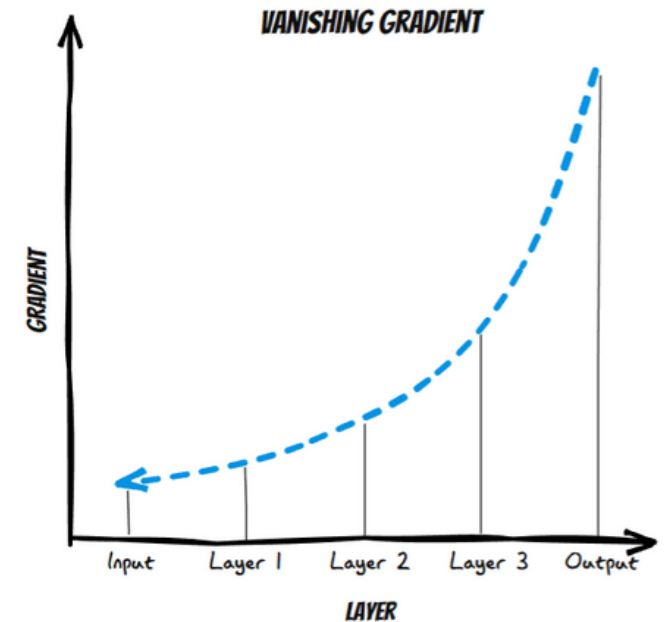
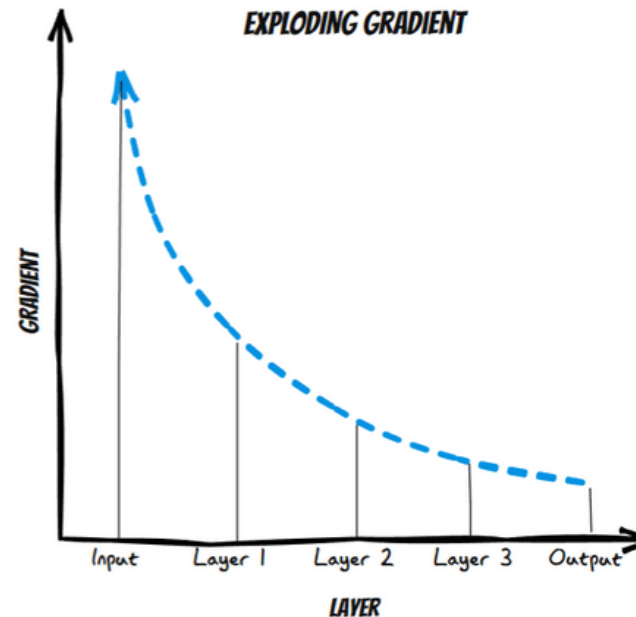
### • 3. Intuition

- If we take a shallower network and add more layers, the deeper one should perform at least as well.
- But in reality: deeper plain networks perform worse.

# What Are Vanishing and Exploding Gradients?

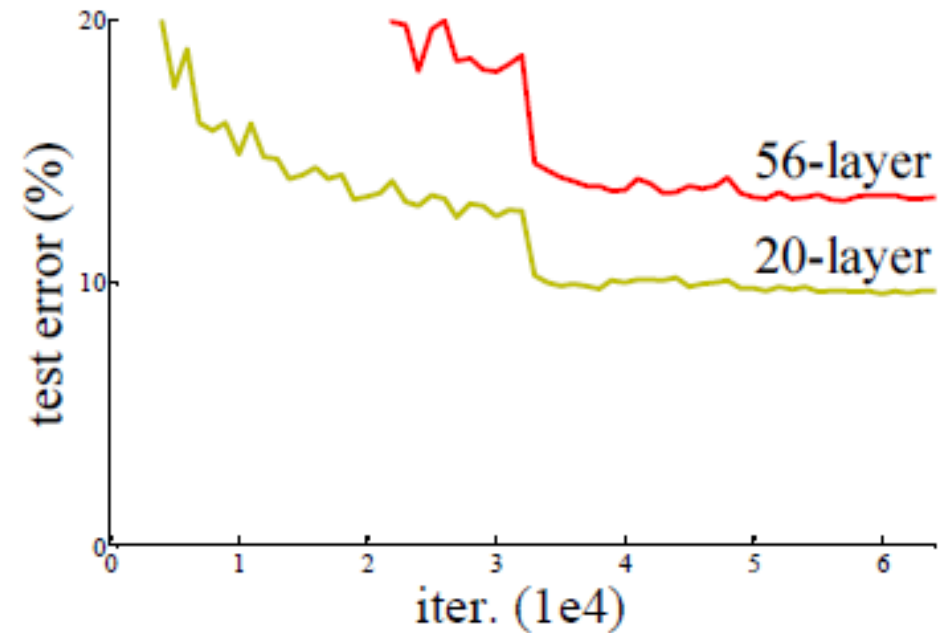
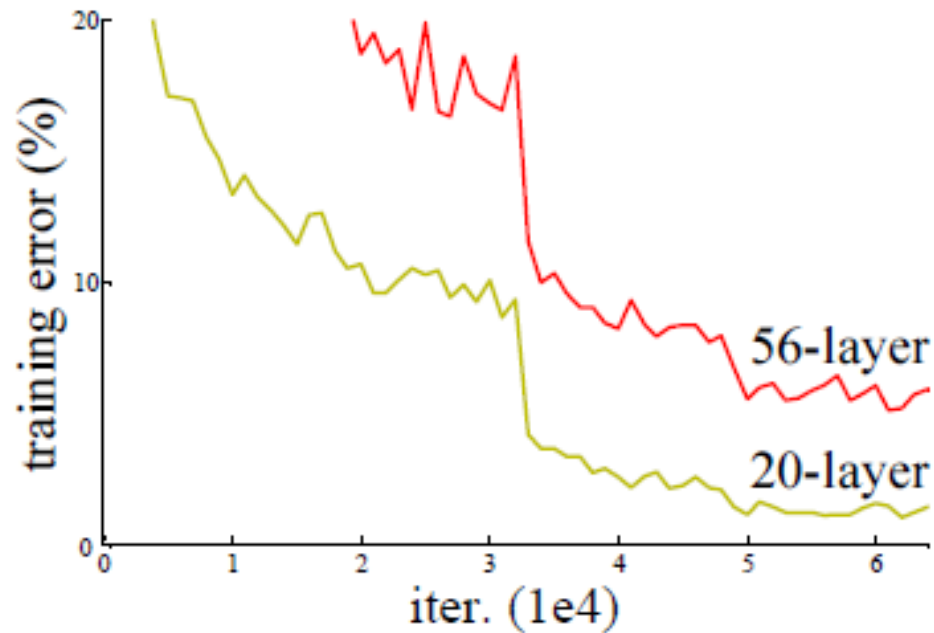
## ■ Vanishing and Exploding Gradients

- As we train deep neural networks using **backpropagation**, gradients are propagated backward through layers.
- During this process
  - Gradients may **shrink toward zero** (vanish)
  - Or **grow excessively large** (explode)
- Both scenarios make training **unstable or slow**.



# Degradation Problem (Empirical Evidence)

- Deeper Plain Networks Have Higher Training Error
  - Observation from Deep Residual Learning for Image Recognition (ResNet) – He et al. (2015)



- Train plain networks of 20 and 56 layers on CIFAR-10
- Despite more capacity, **56-layer network performs worse** (training + test error ↑)
- *“Indicates **optimization failure**, not overfitting!”*

# Understanding the Degradation Problem

---

- **There Exists a Simple Solution — But It's Hard to Learn**

- The deeper model has a **constructed solution**

- Suppose you have a well-trained shallow network.
- You can always create a deeper version by:
  - ✓ Copying the original layers.
  - ✓ Adding extra layers that perform **identity mapping** (i.e., output = input).
- In theory, this should produce at least equal performance.

- **So why does performance degrade in practice?**

- It's not due to **overfitting** (since training error increases).
- It's not due to **vanishing gradients** (since techniques like BatchNorm are applied).

# Understanding the Degradation Problem

---

- **There Exists a Simple Solution — But It's Hard to Learn**

- **The real problem**

- Stochastic Gradient Descent (SGD) fails to find the identity mapping.
    - The network **struggles to learn a perfect "pass-through" behavior through non-linear transformations.**
    - Even a seemingly trivial task — like learning to output the same input — becomes difficult for deep non-linear layers.

- **Key Insight**

- Depth alone is not the problem — **learning identity mappings through standard layers is.**

# Need for Better Formulation

## ■ Residual Learning Reformulation From Learning Mappings to Learning Residuals

### • Key Insight

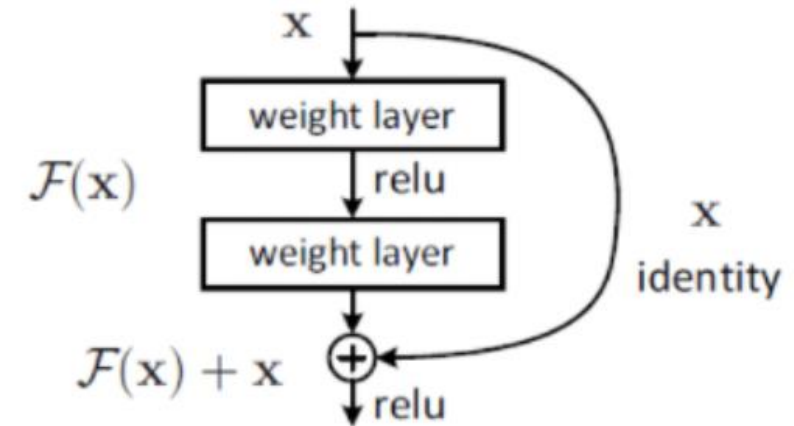
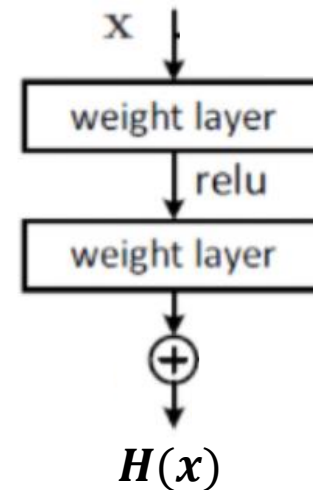
- Rather than learn the full mapping  $H(x)$ , let's learn only what is missing from the input.
- Define residual function:  $F(x) =$   $\Rightarrow H(x) =$

### • Residual Learning Reformulation

- Original mapping:
- Reformulated:

### • Why this helps

- If the optimal mapping is close to identity, then  
✓  $F(x)$  is close to zero  $\rightarrow$  easier to learn.
- Even if  $H(x)$  is complex, it may still be **easier to express the difference** from the input than to learn the entire function from scratch.



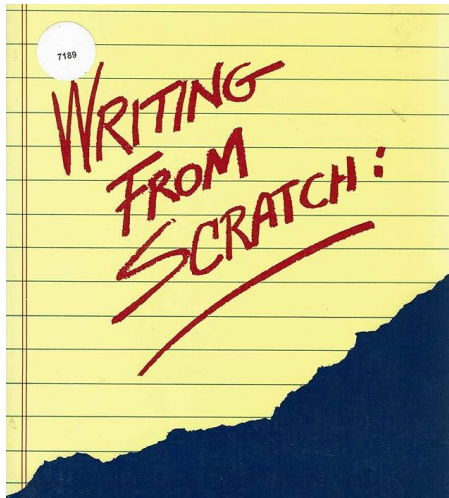
*"This concept is like **preconditioning** in numerical optimization — reshaping the problem to make it easier for solvers to converge."*

# Analogy: Learning Perturbations

## ■ Residuals as Minor Adjustments to Known Inputs

- Think of  $F(x)$  as a **small correction or delta** to the input
  - For example, if the correct output is close to the input, the network only needs to learn the difference.
- **Identity mapping as the “default path”**
  - The skip connection directly carries the input forward.
  - If the network doesn’t need to change it much, the residual function learns small tweaks.
  - If the input needs major changes, the residual function takes full control.

## • Example – Writing



VS



# Summary of Motivation

---

## ■ Why Residual Learning Became a Breakthrough

- Deep networks are **theoretically powerful**, but practically **difficult to optimize**.
- As depth increases, **plain networks**
  - Encounter vanishing gradients, even with tricks like BatchNorm.
  - Show **increased training error** — a clear sign of optimization issues.
- **Residual learning provides a solution**
  - Skip connections allow gradients to flow unimpeded.
  - Identity mappings are **easy to learn** with this architecture.
  - Residual blocks make it easier for the optimizer to converge.
- **Impact**
  - Enables networks with **>100 layers** to be trained effectively.
  - Became the foundation for
    - ✓ ResNet, ResNeXt
    - ✓ Faster R-CNN (backbone)
    - ✓ Mask R-CNN, and more.

# What Are Vanishing and Exploding Gradients?

---

## ■ What is a Computational Graph?

- A computational graph is a **directed acyclic graph** representing a function.
  - **Nodes:** operations (e.g., +, ×, ReLU, etc.)
  - **Edges:** flow of values (scalars, vectors)
- Enables automatic differentiation via backpropagation
- **Example**
  - If  $z = x + y$ , and  $L = \sin(z)$ , then:
  - **Forward pass:** compute  $z = x + y$ , then  $L = \sin(z)$
  - **Backward pass:** compute  $\frac{dL}{dz'} \frac{dL}{dx'} \frac{dL}{dy}$

# What Are Vanishing and Exploding Gradients?

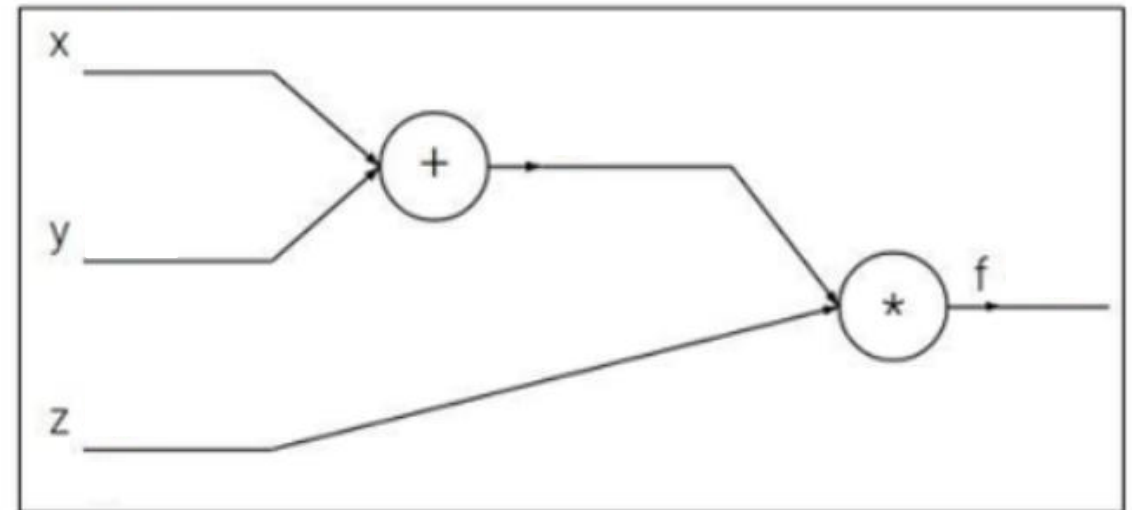
## ■ Computational Graph Example

- **Function:**  $L = (x + y) \cdot z$
- **Graph**
  - **Inputs:**  $x, y, z$
  - **Ops:** addition  $x + y$ , multiplication with  $z$
  - **Final node:** loss  $L$
- **→ Forward Pass:** compute each value
  - **Intermediate variable:**  $q = x + y =$
  - **Final output:**  $f = q \cdot z =$

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

$$\text{e.g. } x = -2, y = 5, z = -4$$



# What Are Vanishing and Exploding Gradients?

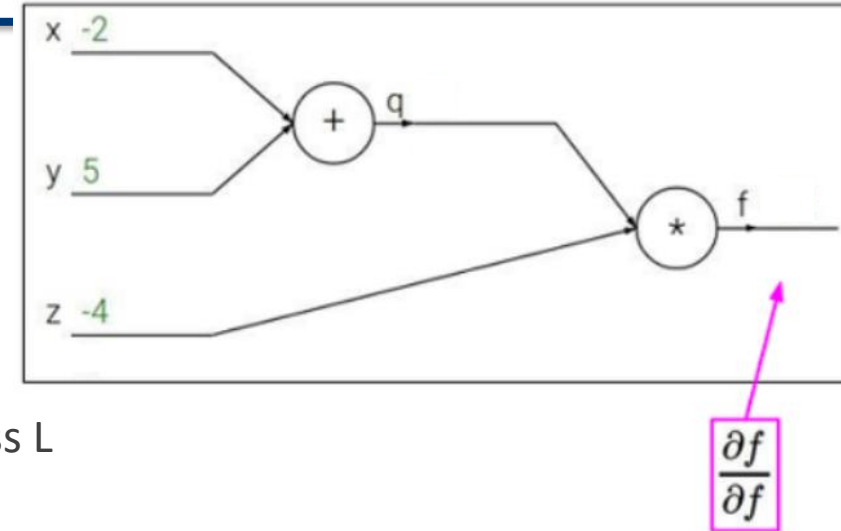
## ■ Scalar Backpropagation

- **Function:**  $L = (x + y) \cdot z$

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

$$\text{e.g. } x = -2, y = 5, z = -4$$



- **Graph**

- **Inputs:**  $x, y, z$  / **Ops:** addition  $x + y$ , multiplication with  $z$  / **Final node:** loss  $L$

- **← Backward Pass (Backpropagation)**

- We want  $-\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

- **Step 1** – From output to  $q$  and  $z$ :  $\frac{\partial f}{\partial q} =$  ,  $\frac{\partial f}{\partial z} =$

- **Step 2** – From  $q$  to  $x$  and  $y$ :  $\frac{\partial q}{\partial x} =$  ,  $\frac{\partial q}{\partial y} =$

✓ Apply chain rule

$$\text{➤ } \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} = -4 \cdot 1 = -4$$

$$\text{➤ } \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y} = -4 \cdot 1 = -4$$

# What Are Vanishing and Exploding Gradients?

## ■ Vector Backpropagation

### • Extending Backpropagation to Matrix Operations

- **Goal:** Given a vector input  $x$  and a weight matrix  $W$ , we aim to compute the gradient of the function

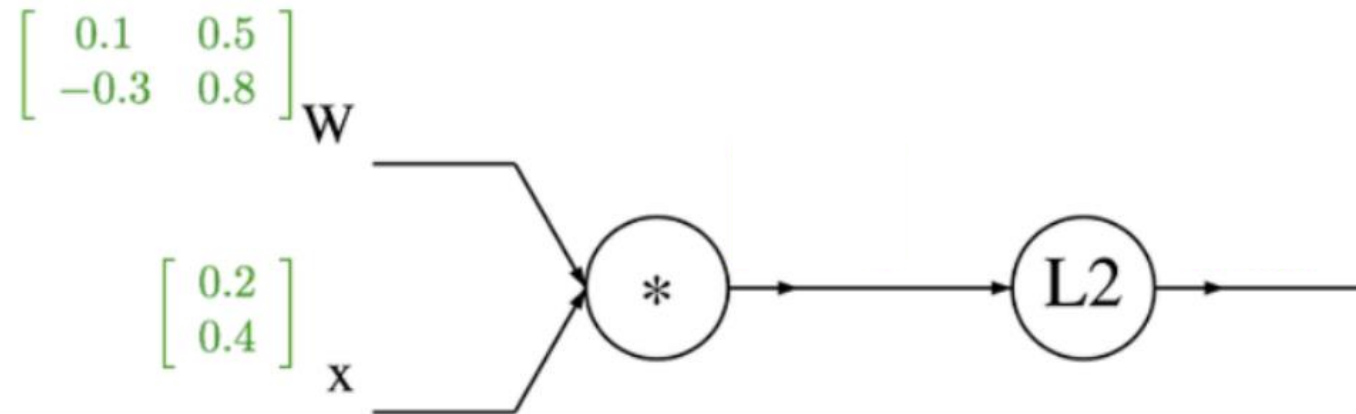
$$f(x, W) =$$

✓  $x \in \mathbb{R}^d$ : input vector (column)

✓  $W \in \mathbb{R}^{n \times d}$ : weight matrix (trainable parameters)

✓  $q = Wx \in \mathbb{R}^n$ : intermediate vector

✓  $f = \|q\|^2 = q^T q$ : scalar output



# What Are Vanishing and Exploding Gradients?

## ■ Vector Backpropagation

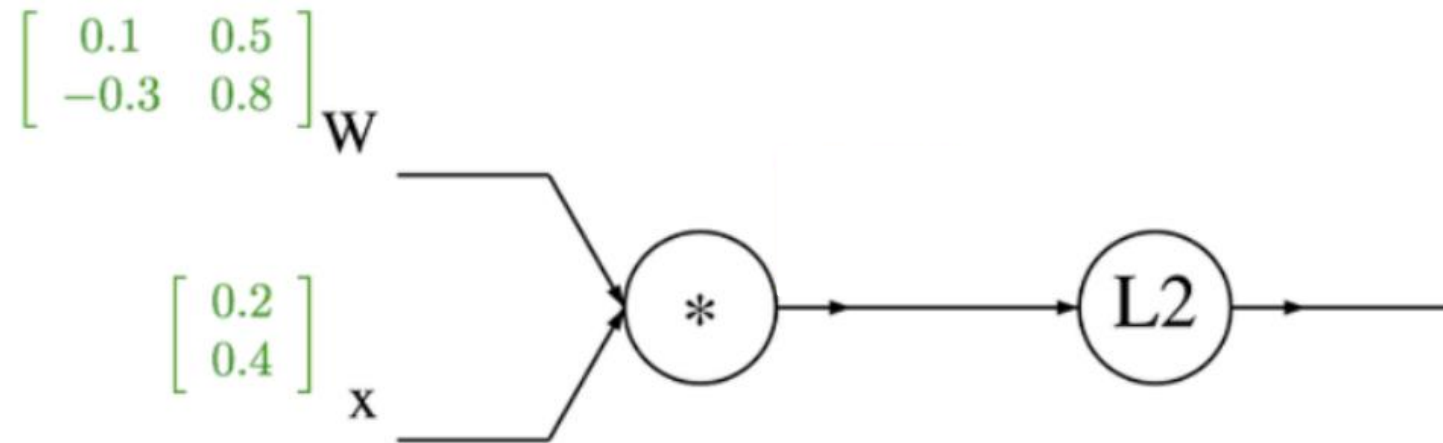
### • Forward and Backward Pass (Step-by-Step)

#### ○ Step 1: Forward Pass

$$\checkmark x = \quad , W =$$

$$\checkmark q =$$

$$\checkmark f =$$



# What Are Vanishing and Exploding Gradients?

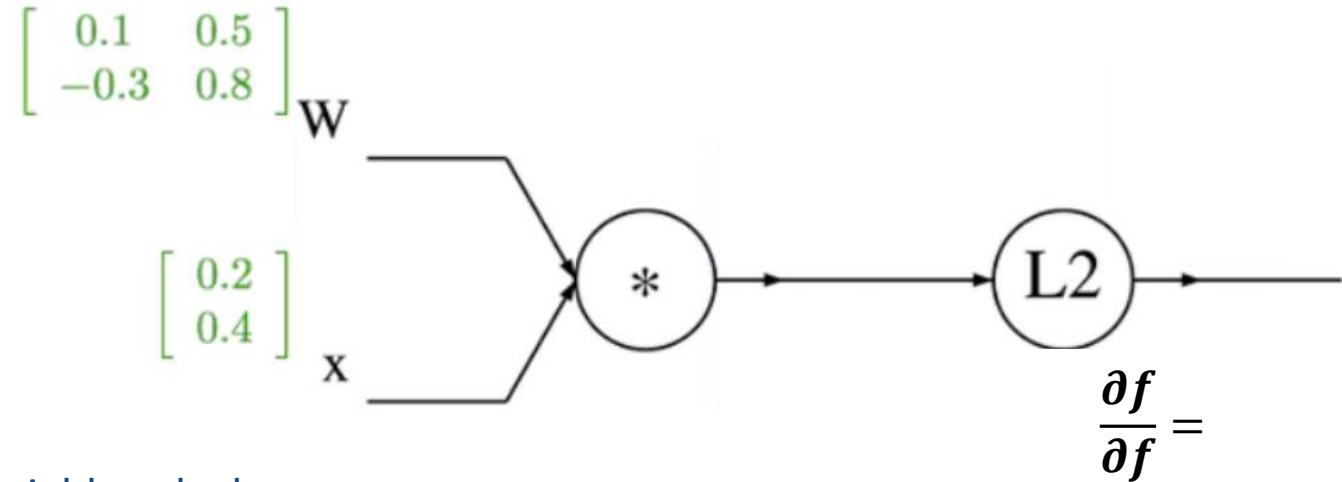
## ■ Vector Backpropagation

### • Forward and Backward Pass (Step-by-Step)

#### ○ Step 2.1: Backward Pass – Gradient w.r.t. $q$

✓  $f(x, W) =$

✓  $\frac{\partial f}{\partial q} =$



✓  $\nabla$ : It represents a **gradient operator** in multivariable calculus.

➤ If you have a function:  $f(x_1, x_2, \dots, x_n)$ , then the **gradient of  $f$**  ( $\nabla f$ ) is denoted by

➤ The gradient is a **vector** that points in the direction of **steepest increase** of the function  $f$ .

#### ✓ In Backpropagation

$$\nabla f =$$

➤  $\nabla_x f$ : How the scalar output  $f$  changes with respect to vector input  $x$ .

➤  $\nabla_W f$ : The matrix of partial derivatives showing how each weight affects the loss.

➤ **Example:** If  $f(x, y) = x^2 + y^2$ , then

$$\nabla f =$$

# What Are Vanishing and Exploding Gradients?

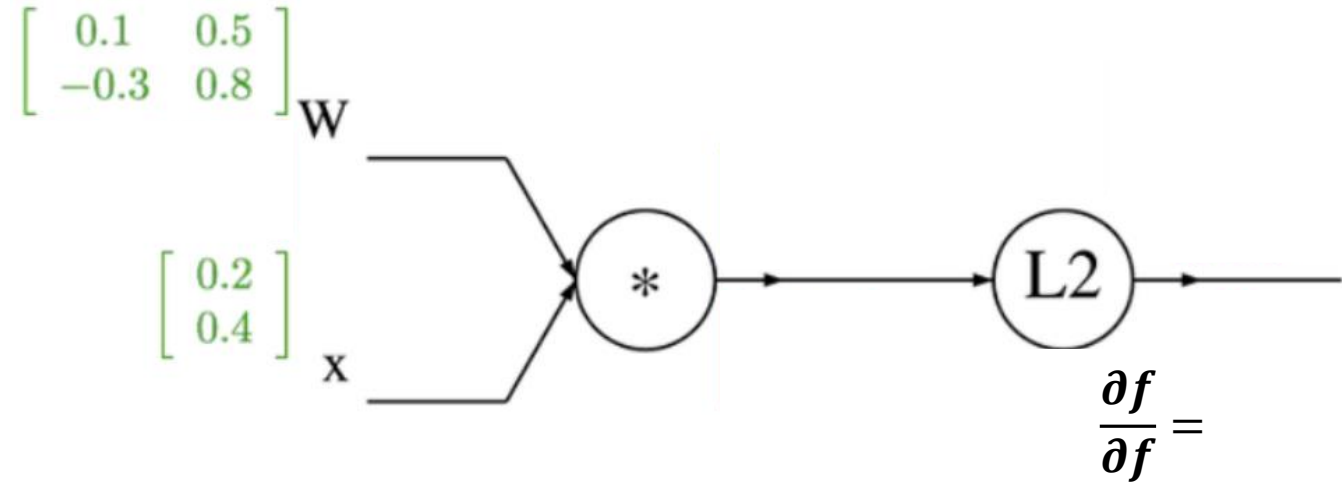
## ■ Vector Backpropagation

### • Forward and Backward Pass (Step-by-Step)

#### ○ Step 2.1: Backward Pass – Gradient w.r.t. $q$

✓  $f(x, W) =$

✓  $\frac{\partial f}{\partial q} =$



#### ○ Step 2.1: Step-by-Step Derivation

✓ Step 1: Expand the Function

➤  $f(q) = q^T q = \sum_{i=1}^n q_i^2$

✓ Step 2: Differentiate Each Component

➤ We compute  $\frac{\partial f}{\partial q_i} =$

➤ Since  $q_i^2$  is the only term that depends on  $q_i$ , the derivative is  $\frac{\partial f}{\partial q_i} =$

# What Are Vanishing and Exploding Gradients?

## ■ Vector Backpropagation

### • Forward and Backward Pass (Step-by-Step)

#### ○ Step 2.1: Step-by-Step Derivation

##### ✓ Step 1: Expand the Function

➤  $f(q) =$

##### ✓ Step 2: Differentiate Each Component

➤ We compute  $\frac{\partial f}{\partial q_i} =$

➤ Since  $q_i^2$  is the only term that depends on  $q_i$ , the derivative is  $\frac{\partial f}{\partial q_i} =$

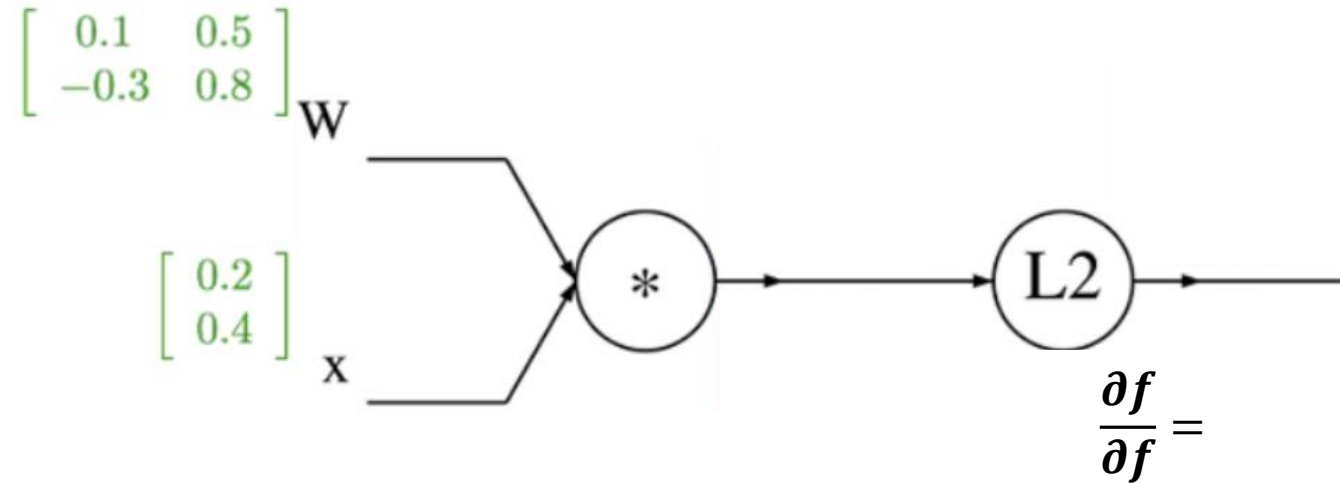
##### ✓ Step 3: Write the Gradient as a Vector

➤ Putting all partial derivatives together →

$$\nabla_q f =$$

##### ✓ Final Result

➤  $\frac{\partial}{\partial q}(q^T q) = 2q$



# What Are Vanishing and Exploding Gradients?

## ■ Vector Backpropagation

### • Forward and Backward Pass (Step-by-Step)

#### ○ Step 2.2: Backward Pass – Gradient w.r.t. $W$

✓ Using chain rule and matrix calculus:  $\nabla_W f =$

✓ Step-by-Step – Derivation of the Gradient

➤ Given Function:  $f(x, W) =$

➤ Step1. Gradient w.r.t.  $q$

We first rewrite  $f$  in terms of  $q$ :

Then take the gradient of  $f$  with respect to the vector  $q$ :

➤ Step2. Apply the Chain Rule – To get the gradient of  $f$  with respect to  $W$ , we apply the chain rule

$$\frac{\partial f}{\partial W} =$$

#### ○ Step 2.1: Backward Pass – Gradient w.r.t. $q$

$$\checkmark f(x, W) =$$

$$\checkmark \frac{\partial f}{\partial q} =$$

# What Are Vanishing and Exploding Gradients?

## ■ Vector Backpropagation

### • Forward and Backward Pass (Step-by-Step)

#### ○ Step 2.2: Backward Pass – Gradient w.r.t. $W$

##### ✓ Step-by-Step – Derivation of the Gradient

##### ➤ Step3. Derivative of $q = Wx$

Each component of  $q$  is  $q_i =$

So the partial derivative is  $\frac{\partial q_i}{\partial W_{ij}} =$

$$\frac{\partial f}{\partial W_{ij}} =$$

$$\Rightarrow \frac{\partial f}{\partial W} =$$

=

➤ Given Function:  $f(x, W) =$

➤ Step1. Gradient w.r.t.  $q$ :  $\frac{\partial f}{\partial q} =$

➤ Step2.  $\frac{\partial f}{\partial W} =$

# What Are Vanishing and Exploding Gradients?

## ■ Backpropagation in Fully Connected Layers

### • Gradient Computation Layer by Layer

- Let
  - ✓  $\mathbf{z}^l =$  : Linear transformation
  - ✓  $\mathbf{a}^l =$  : Activation (e.g., ReLU, sigmoid, tanh)
- Then
  - ✓ Backpropagation equation for the gradient signal  $\rightarrow \delta^l =$
  - ✓ Gradient with respect to the weights  $\rightarrow \frac{\partial L}{\partial \mathbf{W}^l} =$
  - ✓ Here,  $\delta^l$  is the **error signal** passed from layer  $l + 1$  back to layer  $l$ , scaled by the derivative of the activation.

- $\mathbf{z}^k \in \mathbb{R}^{d_k}$ : Pre-activation output of layer  $k$  (i.e., before applying the activation function)
- $\mathbf{W}^k$ : Weight matrix of the  $k$ -th layer
- $\mathbf{a}^{k-1}$ : Activation output from the previous layer
- $\mathbf{b}^k$ : Bias vector

Term	Meaning	Mathematical Definition
$\mathbf{z}^k$		
$\mathbf{a}^k$		
$\delta^k$		
$L$		

# What Are Vanishing and Exploding Gradients?

---

## ■ Backpropagation in Fully Connected Layers

### • Why Gradients Change Across Layers

- As we apply backpropagation repeatedly across many layers,

$$\delta^l = (W^{l+1})^T \delta^{l+1} \circ f'(z^l)$$

- We are multiplying gradient vectors by weight matrices and activation derivatives at each step.
- This means
  - ✓ If those values are **less than 1**, gradients **shrink**.
  - ✓ If they're **greater than 1**, gradients **grow**. This is the root cause of vanishing/exploding gradients.

# What Are Vanishing and Exploding Gradients?

## ■ Vanishing Gradient — Root Cause

- Why gradients shrink too much

- If

*and*

- then

*as*

- This often happens when

- ✓ You use **sigmoid** or **tanh**
    - ✓ Weights are poorly initialized
    - ✓ The network is very **deep**

- Effect

- ✓ Gradients are **too small to update early layers**
    - ✓ Training becomes **very slow or fails entirely**

- $\mathbf{z}^k \in \mathbb{R}^{d_k}$ : Pre-activation output of layer  $k$  (i.e., before applying the activation function)
- $\mathbf{W}^k$ : Weight matrix of the  $k$ -th layer
- $\mathbf{a}^k = f(\mathbf{z}^k)$ : Output of layer  $k$  (post-activation)
- $f'(\mathbf{z}^k)$ : the derivative of the activation function  $f$  evaluated at  $\mathbf{z}^k$

# What Are Vanishing and Exploding Gradients?

## ■ Theoretical View – Exponential Bounds

- If the Jacobian norms satisfy

- If

$$\leq$$

- Then

$$\leq$$

- $k$ : the index of layers

- $L$ : the total number of layers

- Where

- ✓  $C$ : constant based on loss and inputs

- ✓  $\lambda$ : upper bound on gradient propagation at each layer

- ✓  $L$ : number of layers

- Interpretation

- ✓ If  $\lambda < 1$  → gradients vanish

- ✓ If  $\lambda > 1$  → gradients explode

- ✓ If  $\lambda = 1$  → gradients remain stable (ideal!)

# What Are Vanishing and Exploding Gradients?

---

- Think of **deep backpropagation** as simply applying the **chain rule multiple times**, just like in vector backpropagation—but layer by layer.

- Backpropagation

$$\delta^l =$$

- Gradient

$$\frac{\partial L}{\partial W^l} =$$

- This mirrors

$$\frac{\partial f}{\partial W} =$$

Error Signal



Input Influence

## ■ From Simple to Deep

- **Start with vector form** to understand outer products and error  $\times$  input structures.
- **Extend to deep nets** using recursive formulas and activation derivatives

# Recap – Summary of Motivation

---

## ■ Why Residual Learning Became a Breakthrough

- Deep networks are **theoretically powerful**, but practically **difficult to optimize**.
- As depth increases, **plain networks**
  - Encounter vanishing gradients, even with tricks like BatchNorm.
  - Show **increased training error** — a clear sign of optimization issues.
- **Residual learning provides a solution**
  - Skip connections allow gradients to flow unimpeded.
  - Identity mappings are **easy to learn** with this architecture.
  - Residual blocks make it easier for the optimizer to converge.
- **Impact**
  - Enables networks with **>100 layers** to be trained effectively.
  - Became the foundation for
    - ✓ ResNet, ResNeXt
    - ✓ Faster R-CNN (backbone)
    - ✓ Mask R-CNN, and more.

# What is ResNet?

---

- **Deep Residual Networks – Introducing ResNet**

- **A Breakthrough in Very Deep Neural Networks**

- Proposed by **Kaiming He et al.** in **CVPR 2016**
- Paper: *“Deep Residual Learning for Image Recognition”*
- Won **ILSVRC 2015** with a **top-5 error of 3.57%**
- Designed to enable **very deep neural networks** (e.g., 152 layers)

- **Problem Addressed**

- As depth increases, plain CNNs suffer from **optimization difficulties** and even **higher training error** → the **degradation problem**

- **ResNet’s solution**

- Introduce **skip connections** and learn **residual functions** instead of direct mappings

# Why Is ResNet Needed?

---

- **Going Deeper Isn't Always Better**

- **Deeper Networks Face Degradation Without the Right Design**

- **Before ResNet:** VGG, GoogLeNet made progress by going deeper
- **But:** *simply adding layers doesn't guarantee better performance!*
- Example
  - ✓ 56-layer plain CNN performs worse than 20-layer CNN on CIFAR-10

- This is **not** due to overfitting

- It's due to optimization issues — **gradient flow weakens**, training becomes harder

- ResNet reformulates the learning task to address this!

# Why Is ResNet Needed?

## ■ ResNet Architecture at a Glance

### • High-Level Overview

#### ○ Main structure

##### ✓ Initial Layer

- 7×7 convolution with stride 2
- Followed by 3×3 max pooling

##### ✓ Four Stages of Residual Blocks

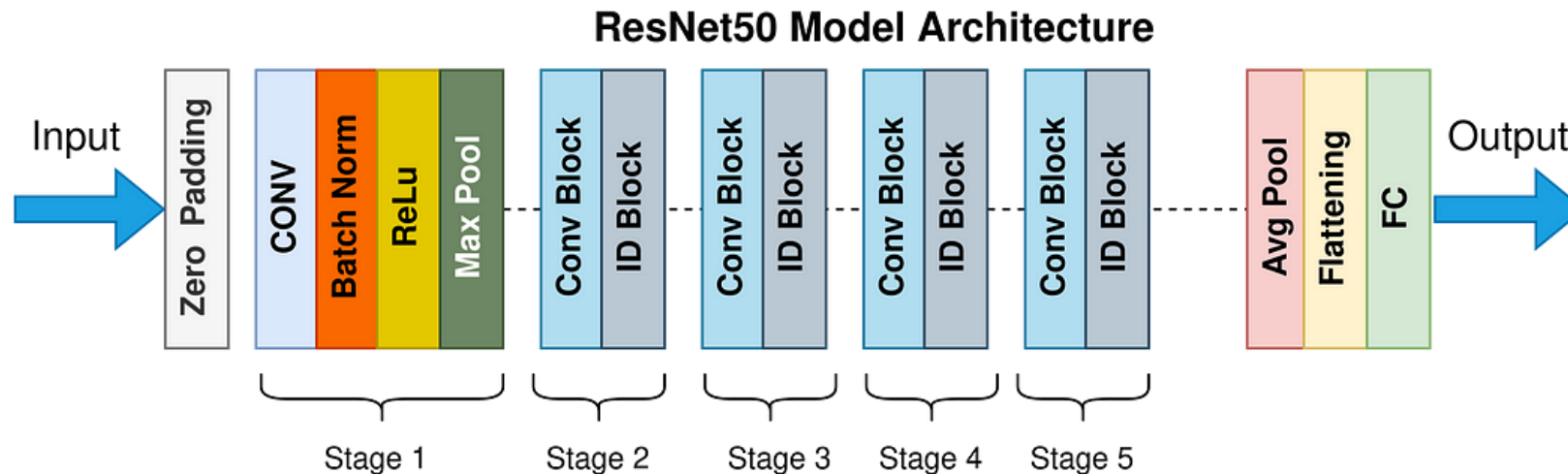
- Conv2\_x, Conv3\_x, Conv4\_x, Conv5\_x
- Spatial resolution halves, channels double each stage

##### ✓ Global Average Pooling (GAP)

- Aggregates features into a 1D vector

##### ✓ Fully Connected Layer + Softmax

- Produces final classification scores



Example: ResNet-50 has 50 convolutional layers, mostly within residual blocks

# ResNet Variants – Different Depths

## ■ ResNet Variants for Different Use Cases

- Deeper ≠ Slower When Designed Right

Model	Depth	Block Type	Parameters
ResNet-18	18	Basic (2×3×3 convs)	~11M
ResNet-34	34	Basic	~21M
ResNet-50	50	Bottleneck	~25M
ResNet-101	101	Bottleneck	~44M
ResNet-152	152	Bottleneck	~60M

- **Basic block** is used in shallow networks (18/34)
- **Bottleneck block** enables efficient deeper networks (50+)

# Key Components of ResNet

- Components That Make ResNet Work

- What's Inside the Architecture?

Component	Role in the Network
Residual Block	Learns a residual function: $F(x)$ , and outputs $F(x) + x$
Skip Connection	Directly passes input through identity or projection mapping
Bottleneck Block	Reduces $\rightarrow$ processes $\rightarrow$ restores dimensions via $1 \times 1$ / $3 \times 3$ / $1 \times 1$ convs
BatchNorm + ReLU	Applied after each convolution layer

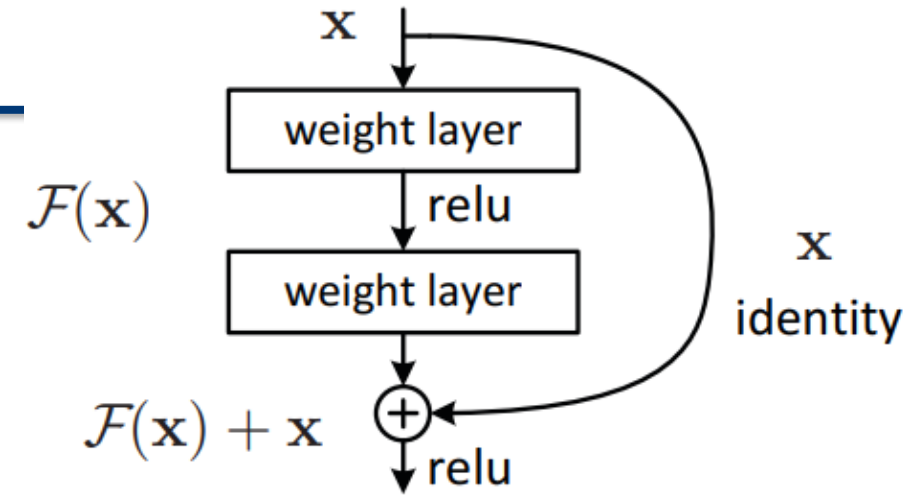
- These design choices ensure effective training of **very deep** models

# The first Key Components of ResNet

## ■ Introduction to the Residual Block

### • What is a Residual Block?

- A **Residual Block** is the core building unit of ResNet (Residual Network).
- It was designed to address the **degradation problem** in deep networks, where adding more layers leads to **worse performance**.
- Instead of directly learning the mapping  $H(x)$ , it learns a **residual function**  $F(x) =$  , and then reconstructs  $H(x)$  as  $H(x) =$



### • Basic Structure of a Residual Block

- The standard formulation:  $y =$ 
  - ✓ Where  $x$  = input feature map,  
 $F(x, \{W_i\})$  = residual function (usually 2–3 convolutional layers with weights  $\{W_i\}$ ),  
 $y$  = output feature map
- **Key Concept**
  - ✓ Instead of learning the full transformation, we let the stacked layers only learn the "difference"  $F(x)$  from the identity.

# The first Key Components of ResNet

---

## ■ Introduction to the Residual Block

### • Why Add the Input Back?

#### ○ Key Idea

- ✓ Residual learning reformulates the desired mapping  $H(x)$  into

$$H(x) =$$

→ The network only needs to learn the residual function  $F(x)$ , not the full transformation  $H(x)$

#### ○ Why is this helpful? (Advantages)

##### ✓ 1. Focus on what's new

- The layer learns only the part that needs to change (i.e., the residual difference \_\_\_\_\_ ). This makes optimization easier, especially in deep networks.

➤ **Naming Origin:** The term “ResNet” comes from the word **residual**

→ The network learns to **minimize the residual** \_\_\_\_\_. Hence, **Residual Network**.

# The first Key Components of ResNet

---

## ■ Introduction to the Residual Block

### • Why Add the Input Back?

#### ○ Why is this helpful? (Advantages)

#### ✓2. Convergence behavior

➤ As the depth increases and the network is well-trained, the input  $x$  becomes increasingly close to the output

$H(x)$ , so the residual \_\_\_\_\_ .

➤ This stabilizes training and encourages minimal necessary updates.

#### ✓3. Implementation simplicity

➤ (1) No major change in architecture is required.

➤ (2) Simply add a **shortcut connection** from input to output.

➤ (3) No extra parameters are introduced since  $x$  is reused.

➤ (4) Aside from the **final addition operation**, the shortcut adds **almost no computational cost**.

# The Second Key Components of ResNet

---

## ■ Introduction to Skip (i.e., Shortcut) Connections

### • Types of Shortcut (i.e., Skip) Connections

○ When input and output dimensions do not match,

✓ 1. Identity Mapping with Padding

$$y =$$

✓ 2. Projection Mapping

$$y =$$

→ where  $W_s$  is a 1x1 convolution used to match dimensions



**What is 1x1 Convolution?**

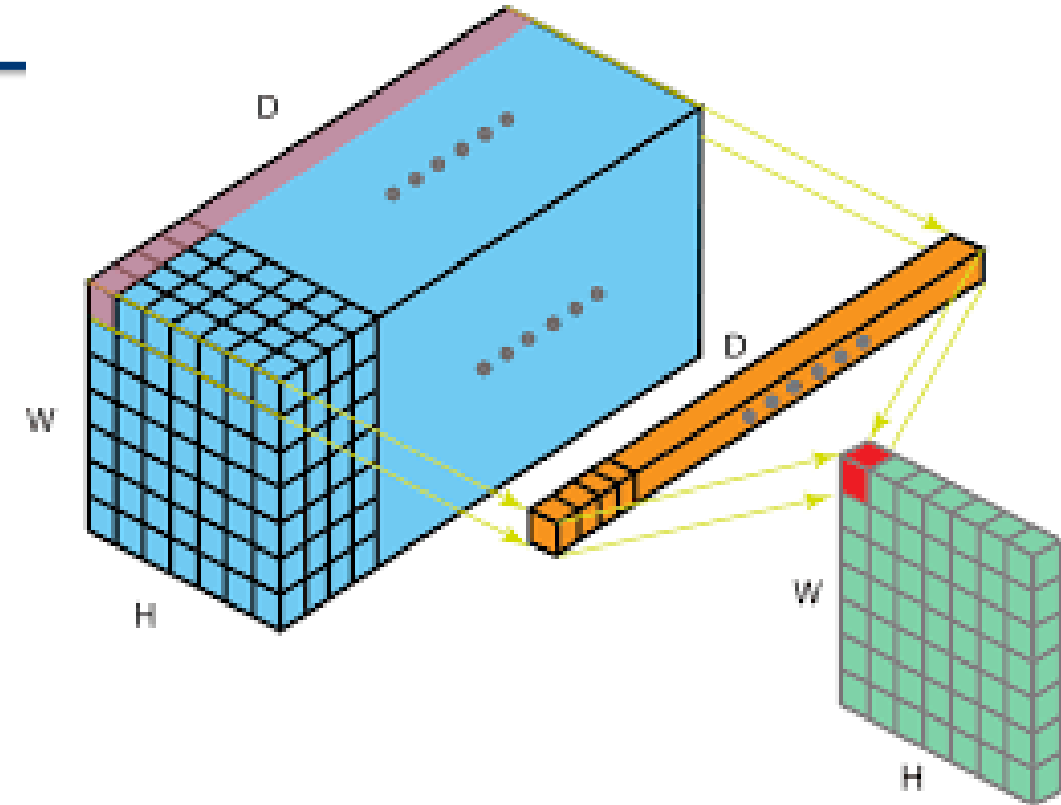
# The Second Key Components of ResNet

## ■ Introduction to Skip (i.e., Shortcut) Connections

### • What is $1\times 1$ Convolution?

- A  **$1\times 1$  convolution** is a convolutional operation where the filter (kernel) has a spatial dimension of 1 (i.e.,  $1\times 1$ ), but spans the full **depth (channel)** of the input.

- Key Purposes of  $1\times 1$  Convolution



Purpose	Explanation
Channel Adjustment	Reduces or expands the number of channels without changing the spatial size ( $W\times H$ )
Computational Efficiency	Reduces the number of parameters and operations, especially before expensive convolutions (e.g., $3\times 3$ , $5\times 5$ )
Adding Nonlinearity	When followed by an activation (e.g., ReLU), it increases model expressiveness
Bottleneck Design Enabler	Used in ResNet's bottleneck blocks ( $1\times 1 \rightarrow 3\times 3 \rightarrow 1\times 1$ ) for compression-expansion

# The Second Key Components of ResNet

## ■ Introduction to Skip (i.e., Shortcut) Connections

- What is 1×1 Convolution?

- Example

- ✓ Scenario: 5×5 Conv on 28×28×192 → 28×28×32

- Direct 5×5 convolution

- $\approx$  operations

- ✓ With 1×1 Conv Compression

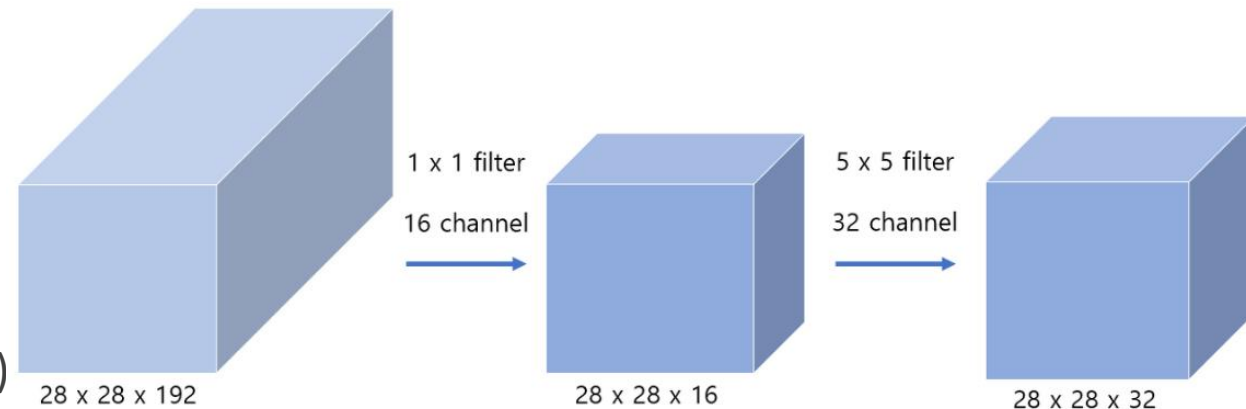
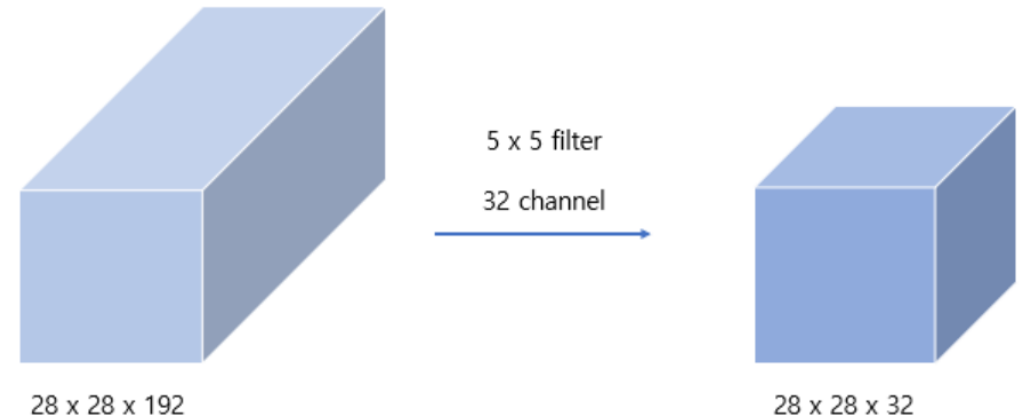
- Step 1. Reduce channels to 16 using 1×1 Conv

- $\approx$

- Step 2. Apply 5×5

- $\approx$

→ Total  $\approx$   $M$  operations ( $\sim 10\times$  smaller)



# The Third Key Components of ResNet

## ■ Introduction to the Bottleneck Block

### • Residual Block Variants in ResNet Implementation

#### ○ Basic Residual Block (used in ResNet-18, 34)

✓ **Shortcut Structure:** two 3×3 convolutions

✓ **Shortcut type**

➤ **Mostly Identity Mapping** is used when input and output dimensions are the same.

➤ **BUT! Projection Mapping** (via 1×1 convolution) is used when needed

- The number of channels increases
- The spatial resolution changes (e.g., stride = 2)

#### ○ Bottleneck Block (used in ResNet-50, 101, 152)

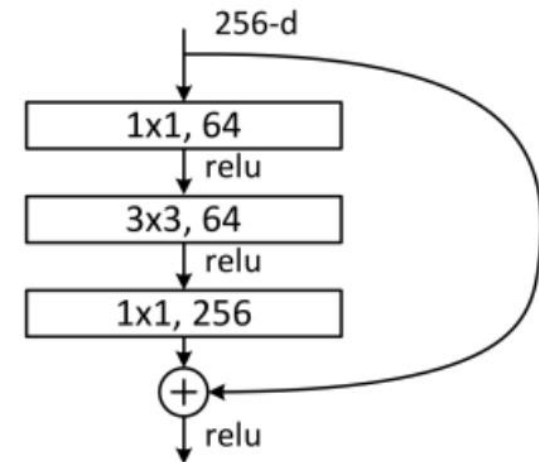
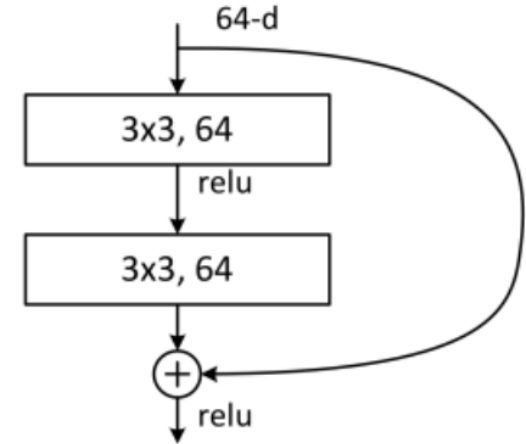
✓ **Shortcut Structure:** three layers consisting of 1×1 → 3×3 → 1×1

✓ **Shortcut type**

➤ **Mostly Projection Mapping** is used in most blocks, especially because

- The bottleneck design frequently changes dimensions
- Downsampling occurs regularly (stride = 2)

✓ Reduces and restores dimensions to lower computation while preserving representational power.



# Key Components of ResNet

---

## ■ Experimental Results – Does ResNet Really Work?

- Purpose

- To evaluate whether **shortcut connections (residual blocks)** truly improve learning in deep networks.

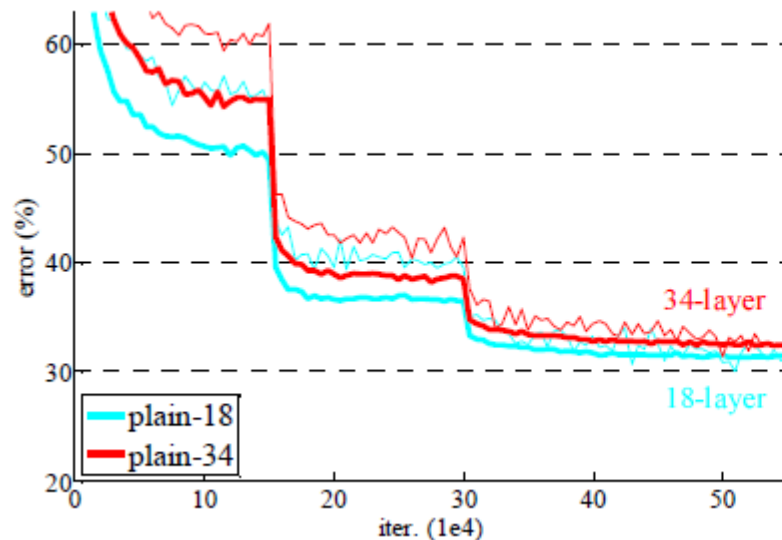
- Experimental Setup

- **Dataset:** ImageNet
  - **Models Compared:** (1) Plain-18 vs. Plain-34, (2) ResNet-18 vs. ResNet-34

# Key Components of ResNet

## ■ Experimental Results – Does ResNet Really Work?

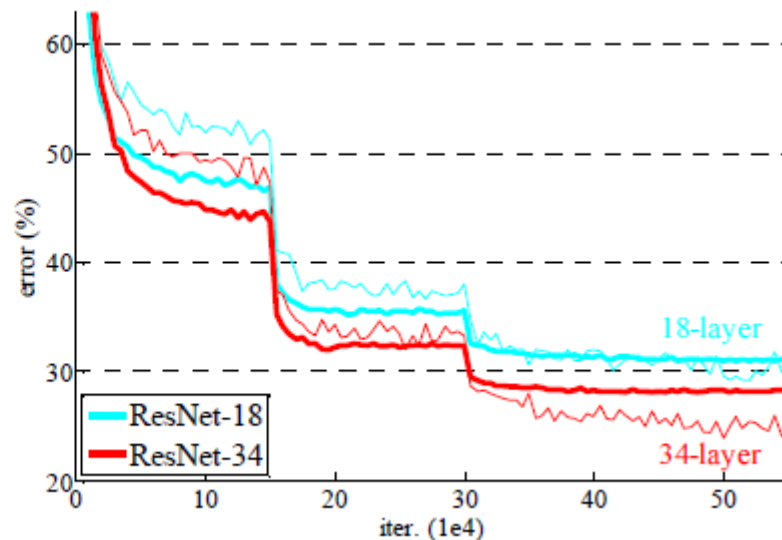
### • Observation



### ○ Left: Plain Network

- ✓ As the network gets deeper (34-layer), performance worsens
- ✓ The **Plain-34** model has **higher training and validation error** than Plain-18
- ✓ This is a classic case of the **degradation problem**

### ○ Right: Residual Network



- ✓ **ResNet-34** outperforms ResNet-18 across all iterations
- ✓ **Deeper = better performance**, as expected
- ✓ Residual learning via shortcuts enables effective optimization of deeper network

# Key Components of ResNet

---

## ■ Why Residual Blocks Prevent Vanishing Gradients (i.e., Better Performance)

### • Forward Pass (Residual Formulation)

- Let the activation function be ReLU and the residual mapping be defined as

$$\checkmark x_{l+1} =$$

➤ Assuming  $f$  is identity for simplicity.

➤ Each term in the equation represents

- $x_l$ : The input to the  $l$ -th layer (or the output from the previous layer)
- $W_l$ : The learnable weights of the  $l$ -th layer
- $F(x_l, W_l)$ : The residual function applied in the  $l$ -th layer
- $f$ : The activation function (e.g., ReLU)
- $x_{l+1}$ : The output from the  $l$ -th layer (input to the next layer)

- This leads to a general form across  $L$  layers

$$\checkmark x_L =$$

# Key Components of ResNet

---

## ■ Why Residual Blocks Prevent Vanishing Gradients (i.e., Better Performance)

### • Backward Pass (Gradient Computation)

- Let the loss be  $\mathcal{L}$ , then by the chain rule: (where the symbol  $\mathcal{L}$  represents the loss function)

$$\frac{\partial \mathcal{L}}{\partial x_1} =$$

- The gradient splits into **two terms**

#### ✓(1) Identity Path (i.e., Residual Connection)

*(Identity Path)*  $\Rightarrow$

- This term bypasses all weight layers  $\rightarrow$  gradient **flows unimpeded**

#### ✓(2) Residual Path

*(Residual Path)*  $\Rightarrow$

- This term propagates through weights  $\rightarrow$  may diminish or explode

# Key Components of ResNet

---

## ■ Why Residual Blocks Prevent Vanishing Gradients (i.e., Better Performance)

### • Why This Matters

- Residual connection ensures **at least one term in the gradient is preserved** (term 1)
- Unlike plain networks, where gradients might vanish as depth increases, ResNet ensures

$$(i.e., term\ 1) \neq 0$$

✓ Even if part of the gradient shrinks, the **identity shortcut preserves signal**, avoiding full gradient collapse

### ○ Practical Implication

$$\frac{\partial L}{\partial x_1} =$$

✓ In order to become gradient vanishing in real training (mini-batch SGD), the gradients in the first term become -1. BUT, the gradients vary and often don't exactly become -1.

✓ Still, residual design avoids full cancellation

✓ Thus, vanishing gradients are prevented and deep networks can be trained effectively

# The Fourth Key Components of ResNet

---

- **What is Batch Normalization?**

- **Batch Normalization (BN) – Making Deep Networks Easier to Train**

- BN is a technique to normalize the input of each layer so that its distribution remains stable throughout training.

- **Why Do We Need It?**

- Deep networks often suffer from

- ✓ **Gradient Vanishing/Exploding:** Especially with deep networks and activation functions like sigmoid/tanh.

- ✓ **Internal Covariate Shift:** Input distribution to each layer changes during training.

# The Fourth Key Components of ResNet

## ■ What is Batch Normalization?

### • Why Do We Normalize?

#### ○ Intuition Behind Normalization

- ✓ Normalization helps the model **learn faster** and **avoid getting stuck** in bad solutions.
- ✓ During optimization, models can get trapped in **local optima** if the loss surface is highly irregular or poorly scaled.
- ✓ By normalizing, we **reshape the loss surface**, making it smoother and easier to traverse.



- **Without normalization:** Optimizer may get stuck in **sharp local minima**
- **With normalization:** Loss surface becomes **flatter**, reducing the chance of being stuck in suboptimal regions

*Goal: Help the optimizer reach the **global optimum** by simplifying the search space.*

# The Fourth Key Components of ResNet

---

## ■ Batch Normalization

### • Motivation – 1. The Gradient Vanishing / Exploding Problem

#### ○ Why Deep Networks Are Hard to Train

- ✓ Training deep neural networks is challenging due to the **Gradient Vanishing** or **Gradient Exploding** problem.
- ✓ Gradients become **too small** (vanish) or **too large** (explode) as they propagate backward through many layers.
- ✓ This makes it hard to update parameters effectively, resulting in
  - Poor convergence
  - High error rate
  - Inability to train deep models

#### ○ What Causes It?

- ✓ Common activation functions like **sigmoid** or **tanh** squash the output to narrow ranges (e.g.,  $[0, 1]$ ), especially in deep layers.
- ✓ This results in
  - **Large variations in input** leading to **tiny changes in output**
  - Gradients becoming extremely small
- ✓ A typical solution is to use **ReLU** (Rectified Linear Unit), which preserves gradients better in deep models.

# The Fourth Key Components of ResNet

---

## ■ Batch Normalization

### • Motivation – 1. The Gradient Vanishing / Exploding Problem

#### ○ Workarounds (but not ideal)

- ✓ **Change activation:** Use ReLU instead of sigmoid/tanh
- ✓ **Careful weight initialization:** Helps maintain gradient scale
- ✓ **Small learning rate:** Prevents gradients from exploding

*However, these are patches, not fundamental fixes.*

#### ○ Core Idea: Stabilize the Entire Learning Process

- ✓ This leads to the introduction of **Batch Normalization (BN)**
- ✓ BN normalizes inputs to each layer, so that
  - The distribution of inputs stays consistent throughout training
  - The network becomes **less sensitive** to initialization
  - Training becomes **faster** and **more stable**

✓ BN effectively prevents both gradient vanishing and exploding by keeping activations within a healthy range.

# The Fourth Key Components of ResNet

## ■ Batch Normalization

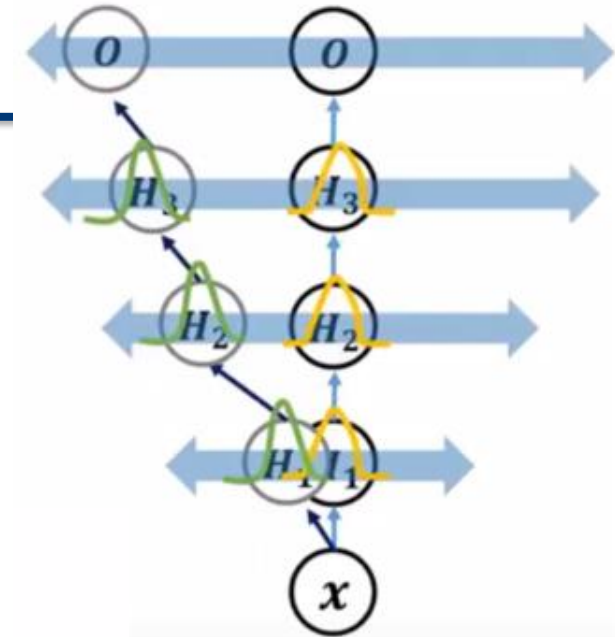
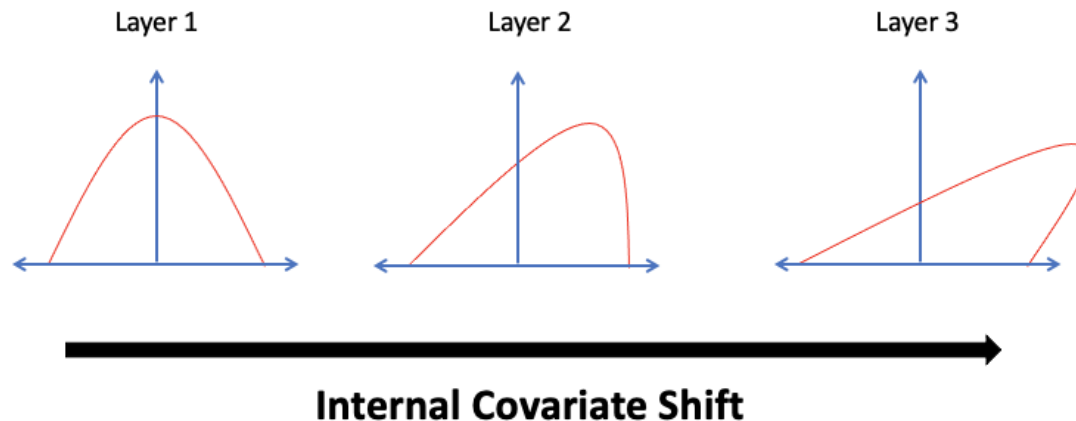
### • Motivation – 2. Internal Covariate Shift

#### ○ What Is Internal Covariate Shift?

- ✓ During training, the distribution of inputs to each layer **keeps changing** as the parameters of previous layers update.
- ✓ This shift slows down training and makes it harder for the network to converge.

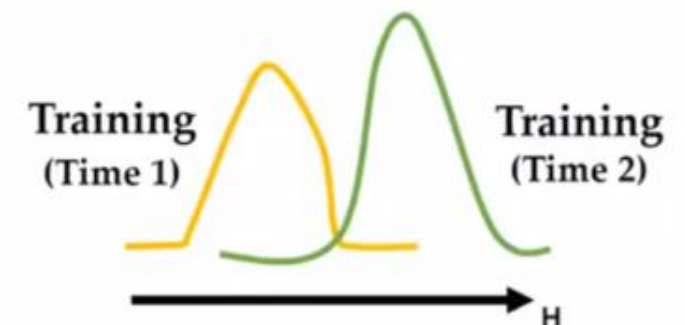
○ **Covariate Shift:** When the input distribution to a model (or a layer) changes due to external factors (e.g., changing dataset).

○ **Internal Covariate Shift:** When the input distribution to a layer **changes during training** due to updates in earlier layers.



## Learning Problem in DNN

This variance is called 'Internal Covariate Shift'



# The Fourth Key Components of ResNet

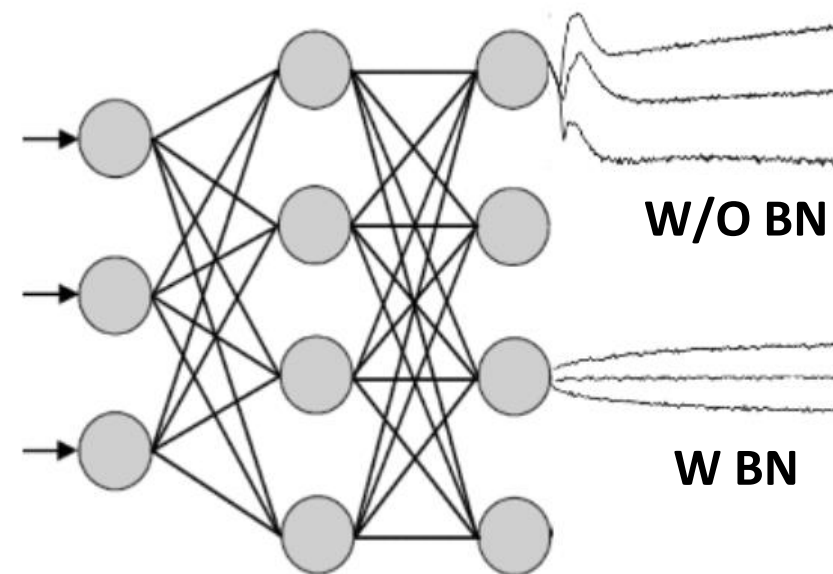
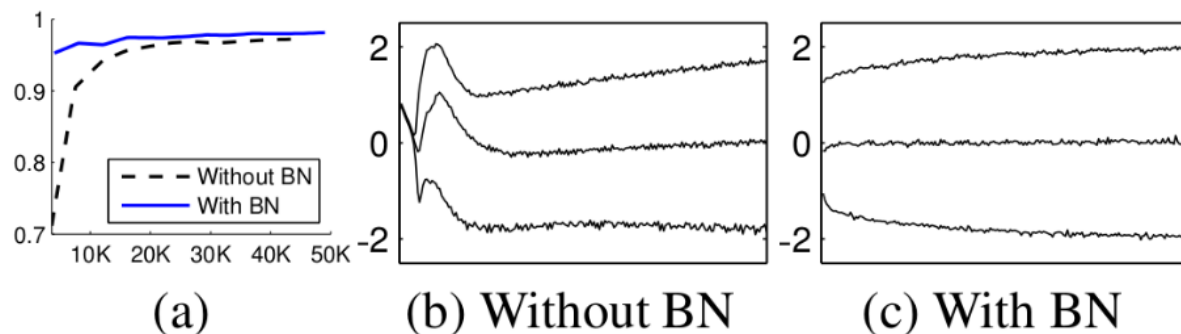
## ■ Batch Normalization

### • Motivation – 2. Internal Covariate Shift

#### ○ Why It Matters

- ✓ Neural networks are deep compositions of functions.
- ✓ As each layer is updated during training, the input distribution to the next layer **shifts**, making it difficult for the next layer to adapt.
- ✓ This leads to **slower convergence** and **unstable training**.

#### ○ Empirical Evidence



Batch Normalization reduces internal covariate shift  
→ resulting in **faster training, better stability, and higher accuracy**

- ✓ The graph (right) shows how **BN stabilizes input distributions** across layers.
- ✓ With BatchNorm: input distributions become **more consistent**, helping the network learn faster and more reliably.

# The Fourth Key Components of ResNet

---

## ▪ Batch Normalization

### • How BN Works

- Let  $\mathbf{x} = \{x_1, x_2, \dots, x_m\}$  be a mini-batch of inputs

#### ✓ 1. Compute mean and variance

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

#### ✓ 2. Normalize the inputs

$$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B}$$

#### ✓ 3. Apply scale and shift

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

➤ Where

$\epsilon$ : small constant for numerical stability

$\gamma$ : learnable **scale** parameter

$\beta$ : learnable **shift** parameter

# The Fourth Key Components of ResNet

## ■ Batch Normalization

### • Backpropagation in Batch Normalization

#### ○ Learnable Parameters

✓ **BN** introduces two learnable parameters

➤  $\gamma$  (gamma): controls the **scale**,  $\beta$  (beta): controls the **shift**

#### ○ Chain Rule Derivatives

✓ Given loss  $L$ , we backpropagate through BN using the following gradients. The normalized input  $\hat{x}_i$  consists of  $\mu_B$  and  $\sigma_B^2$ .

$$\frac{\partial L}{\partial \hat{x}_i} =$$

➤ Which means that we need to compute the partial derivative of the loss  $L$  with respect to  $\mu_B$  and  $\sigma_B^2$ .

#### ○ Why?

✓ In Batch Normalization, we normalize the input using the batch mean  $\mu_B$  and variance  $\sigma_B^2$ , then scale and shift it using learnable parameters  $\gamma$  and  $\beta$ .

✓ To update the model through backpropagation, we must compute the gradient of the loss with respect to these internal variables, including the variance.

1. Compute mean and variance:

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

2. Normalize the inputs:

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

3. Apply scale and shift:

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

# The Fourth Key Components of ResNet

## ■ Batch Normalization

- Backpropagation in Batch Normalization

- Chain Rule Derivatives

- ✓ Step 1. Compute the partial derivative of the loss  $L$  with respect to the **variance**  $\sigma_B^2$  of a mini-batch.

$$\frac{\partial L}{\partial \sigma_B^2} =$$

=

=

1. Compute mean and variance:

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

2. Normalize the inputs:

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

3. Apply scale and shift:

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

# The Fourth Key Components of ResNet

## ■ Batch Normalization

- Backpropagation in Batch Normalization
  - Chain Rule Derivatives
    - ✓ Step 2. Compute the partial derivative of the loss  $L$  with respect to the **variance**  $\mu_B$  of a mini-batch.

$$\frac{\partial L}{\partial \mu_B} =$$

=

=

=

=

1. Compute mean and variance:

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

2. Normalize the inputs:

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

3. Apply scale and shift:

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

# The Fourth Key Components of ResNet

## ■ Batch Normalization

### • Backpropagation in Batch Normalization

#### ○ Chain Rule Derivatives

➤ Step 3. Compute the partial derivative of the loss  $L$  with respect to the **input**  $x_i$  of a mini-batch.

- (1) Direct path through  $\hat{x}_i$

$$\frac{\partial L}{\partial x_i} = \quad = \quad =$$

- (2) Direct path through  $\sigma_B^2$

$$\frac{\partial L}{\partial x_i} = \quad = \quad =$$

1. Compute mean and variance:

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

2. Normalize the inputs:

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

3. Apply scale and shift:

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

# The Fourth Key Components of ResNet

## ■ Batch Normalization

- Backpropagation in Batch Normalization

- Chain Rule Derivatives

- Step 3. Compute the partial derivative of the loss  $L$  with respect to the **input**  $x_i$  of a mini-batch.

- (3) Direct path through  $\mu_B$

$$\frac{\partial L}{\partial x_i} = \quad = \quad =$$

- (4) Final Expression Combining All Three Terms

$$\frac{\partial L}{\partial x_i} =$$

1. Compute mean and variance:

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

2. Normalize the inputs:

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

3. Apply scale and shift:

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

# The Fourth Key Components of ResNet

---

## ■ Batch Normalization

- Backpropagation in Batch Normalization

- Chain Rule Derivatives

- Step 4. Complete the final expression combining the results from Step1 and Step2.

- Result from Step 1

$$\frac{\partial L}{\partial \sigma_B^2} =$$

- Result from Step 2

$$\frac{\partial L}{\partial \mu_B} =$$

- Final Expression

$$\frac{\partial L}{\partial x_i} =$$

# The Fourth Key Components of ResNet

## ■ Batch Normalization

- Backpropagation in Batch Normalization

- Chain Rule Derivatives

- Step 4. Complete the final expression combining the results from Step1 and Step2.

- Final Expression

$$\frac{\partial L}{\partial x_i} =$$

3. Apply scale and shift:

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

The final expression can be written entirely in terms of  $\frac{\partial L}{\partial \hat{x}_i}$ . It tells us how much the normalized input  $\hat{x}_i$  contributes to the final loss.

⇒

# The Fourth Key Components of ResNet

## ■ Batch Normalization

### • Backpropagation in Batch Normalization

#### ○ Chain Rule Derivatives

✓ The final expression can be written entirely in terms of  $\frac{\partial L}{\partial \hat{x}_i}$ . It tells us how much the normalized input  $\hat{x}_i$  contributes to the final loss.

$$y_i = \gamma \hat{x}_i + \beta \Rightarrow \frac{\partial L}{\partial \hat{x}_i} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial \hat{x}_i} = \frac{\partial L}{\partial y_i} \cdot \gamma$$

### • Summary of the Full Process

#### ○ Forward Pass

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad y_i = \gamma \hat{x}_i + \beta$$

#### ○ Backward Pass Goal

Compute  $\frac{\partial L}{\partial y_i}$

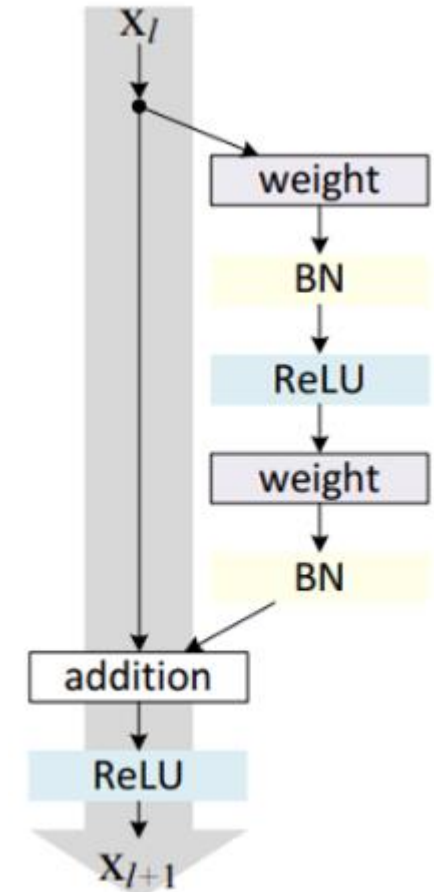
We don't "observe"  $\frac{\partial L}{\partial y_i}$  directly. BUT! we can compute it from  $\frac{\partial L}{\partial y_i}$ , which is what backprop actually gives us by using the final expression with  $\frac{\partial L}{\partial \hat{x}_i}$ .

# The Fourth Key Components of ResNet

## ■ Batch Normalization

### • How BN is used in ResNet

- In the original ResNet architecture (shown on the left), **Batch Normalization** is applied **after each convolutional layer**, before the ReLU activation.
- The sequence inside a Residual Block is
  - ✓ 1. Conv (i.e., weight in the figure) → BN → ReLU
  - ✓ 2. Conv → BN
  - ✓ 3. Addition
  - ✓ 4. ReLU
- **Pre-activation ResNet (Proposed Improvement)**
  - ✓ In deeper networks (like ResNet-164), researchers observed that training becomes harder.
  - ✓ They proposed a **pre-activation variant**, where **BN and ReLU come before convolution**.
  - ✓ This leads to smoother optimization and better gradient flow.



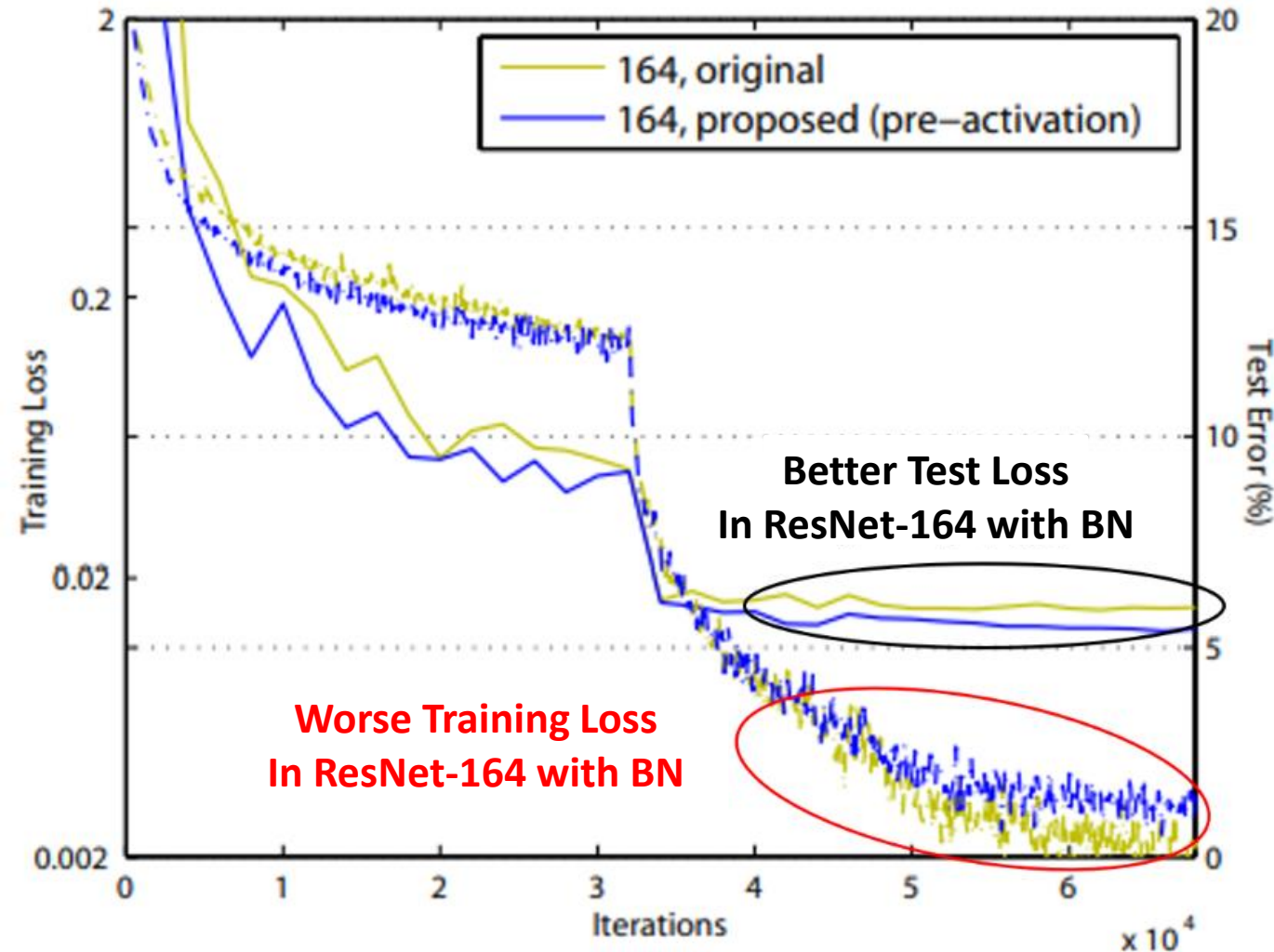
# The Fourth Key Components of ResNet

## ■ Batch Normalization

### • Performance Comparison (ResNet-164)

- Yellow: Original post-activation ResNet
- Blue: Proposed pre-activation ResNet
- Training Loss
  - ✓ Blue curve (pre-activation) shows worse training loss (highlighted in red).
- Test Error
  - ✓ Blue curve generalizes better with lower test error (highlighted in black).

Indicates less overfitting.  
Leads to **improved generalization performance.**



# Summary – Key Ideas Behind ResNet

---

## ■ Why Do We Need ResNet?

- Deeper networks offer stronger representational power.
- But **deeper  $\neq$  better**, due to training difficulties.
- Challenges like **Vanishing/Exploding Gradients** and the **Degradation Problem** hinder performance as depth increases.

## ■ Key Components of ResNet

Component	Role in the Network
Residual Block	Learns a residual function: $F(x)$ , and outputs $F(x) + x$
Skip Connection	Directly passes input through identity or projection mapping
Bottleneck Block	Reduces $\rightarrow$ processes $\rightarrow$ restores dimensions via $1\times 1$ / $3\times 3$ / $1\times 1$ convs
BatchNorm + ReLU	Applied after each convolution layer

# Summary – Key Ideas Behind ResNet

---

## ■ Role of Batch Normalization

- Mitigates gradient vanishing/exploding
- Addresses **Internal Covariate Shift** → keeps input distribution stable across layers
- Used in the sequence: Conv → BN → ReLU  
(Later, **Pre-activation**: BN → ReLU → Conv is proposed and found to perform better)

## ■ Takeaways

- **Residual learning** enables the training of very deep networks
- ResNet architecture is not just about going deeper, but about training **effectively and reliably**
- Still widely used as a **backbone** in modern deep learning models