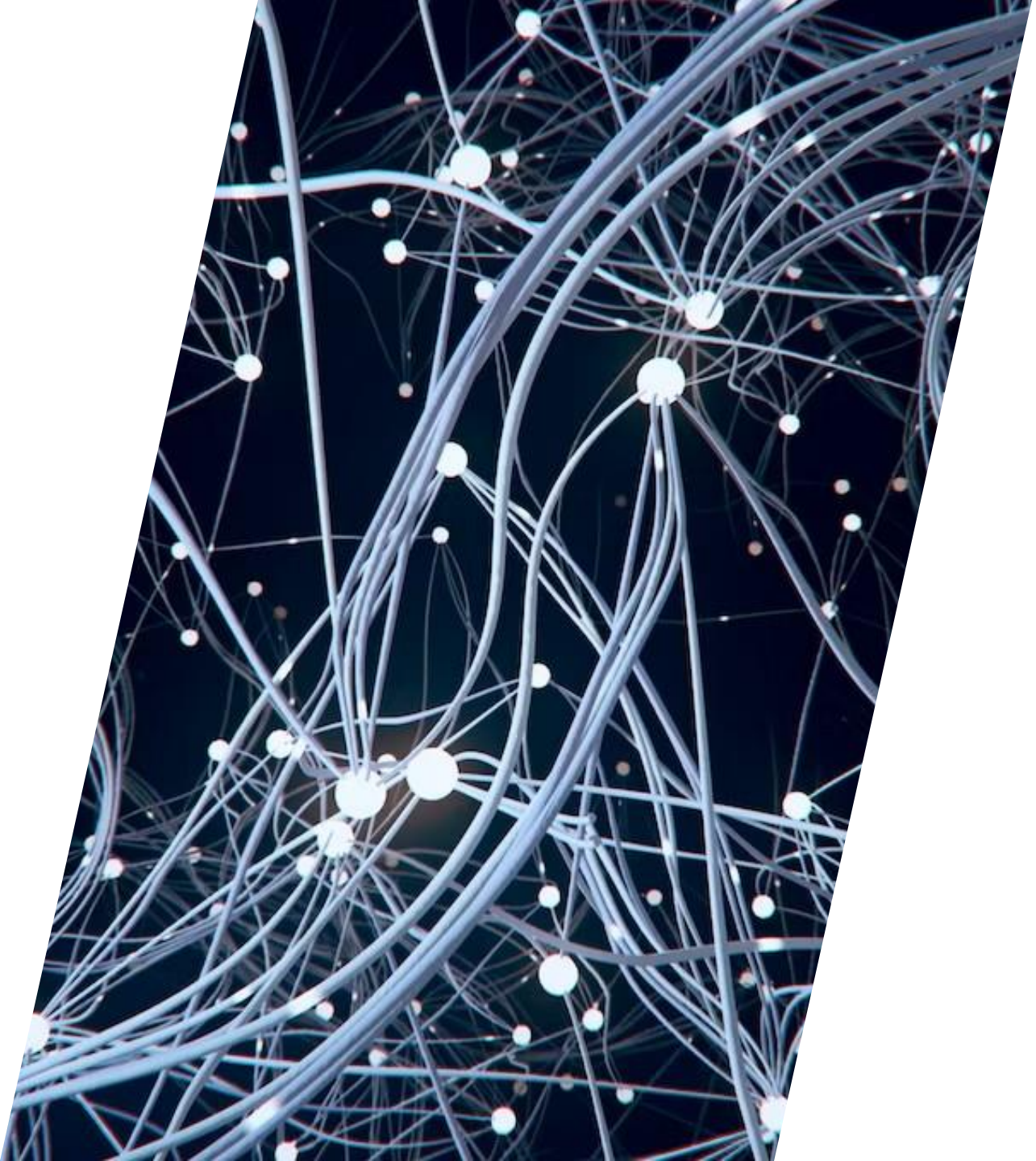
# DEEP RESIDUAL LEARNING

RESNET

컴퓨터소프트웨어학부
2021088304
박현준



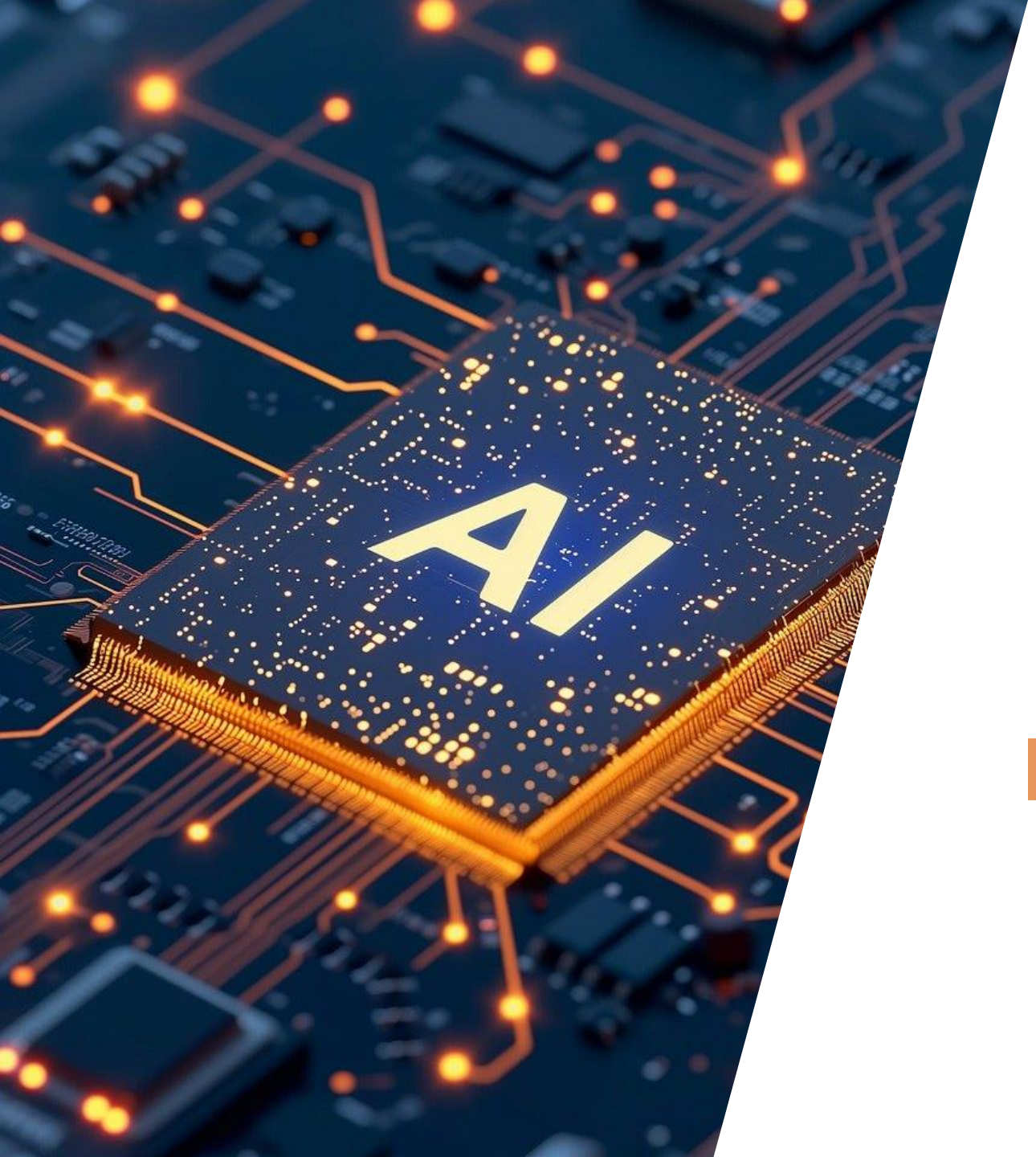
i. Introduction

ii. Deep Residual Learning

iii. Architecture

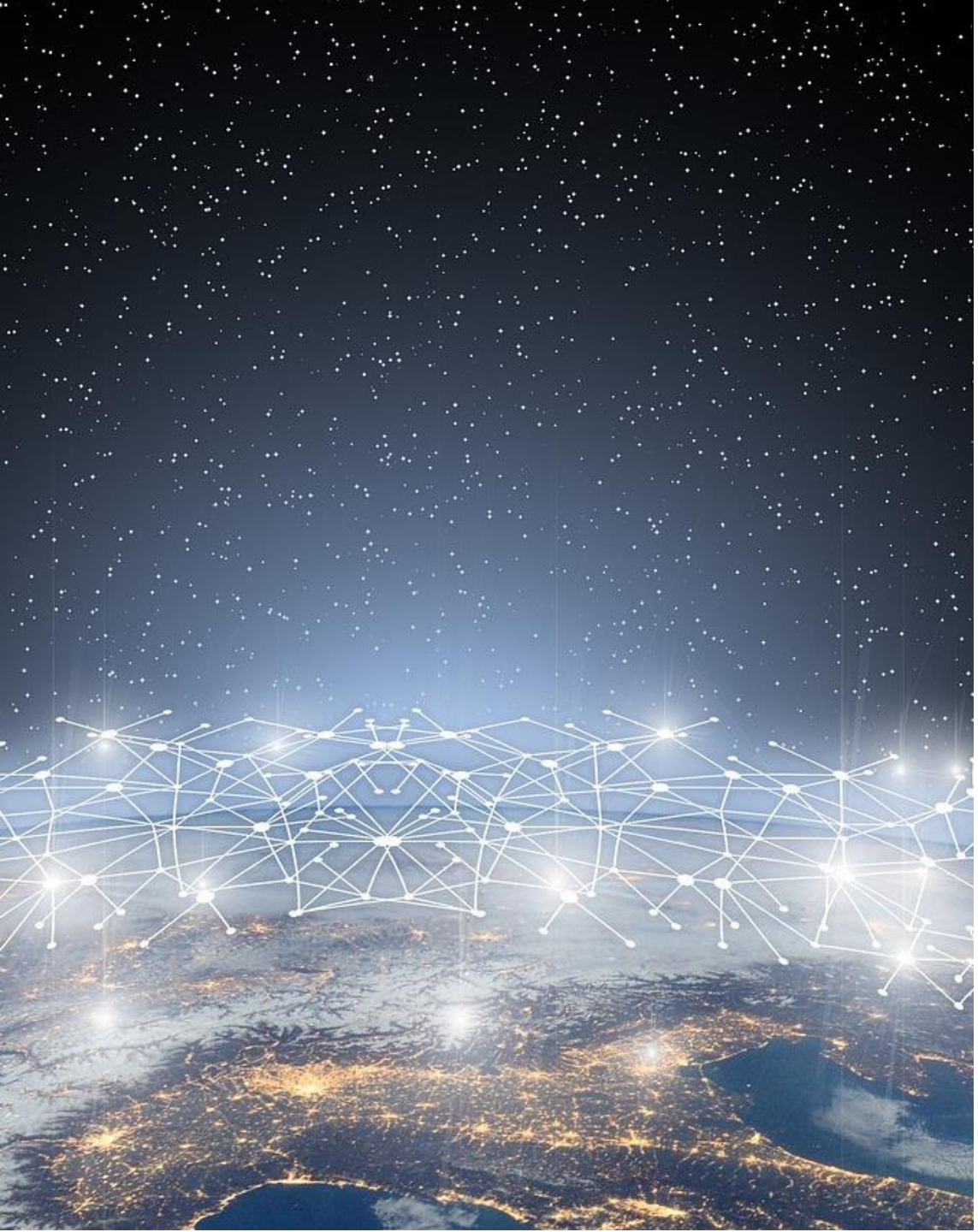
iv. Experiment

v. Conclusion




# INTRODUCTION

CHAPTER 1



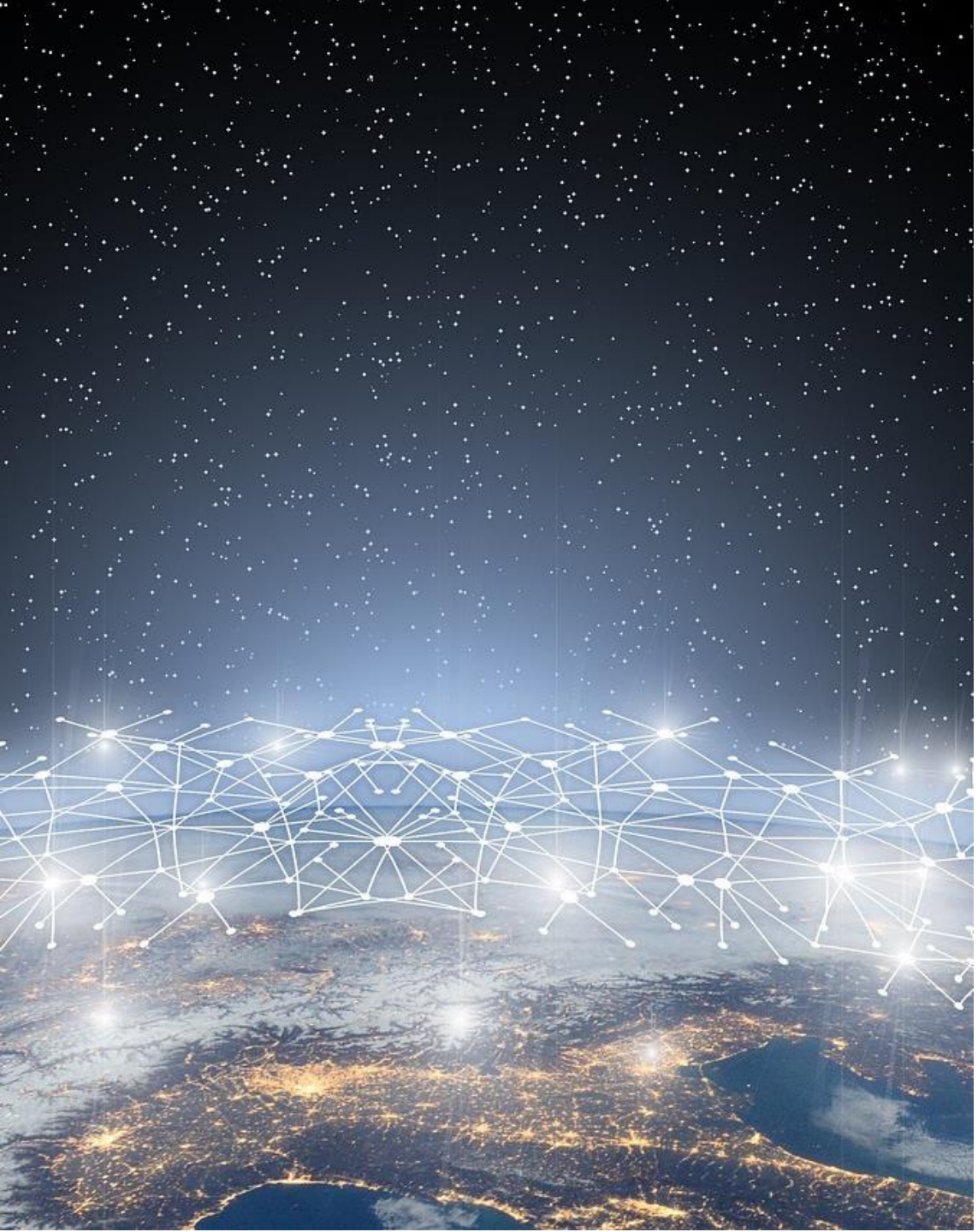
# INTRODUCTION

Question

**#Layer ↑**

**==**

**Network performance ↑**



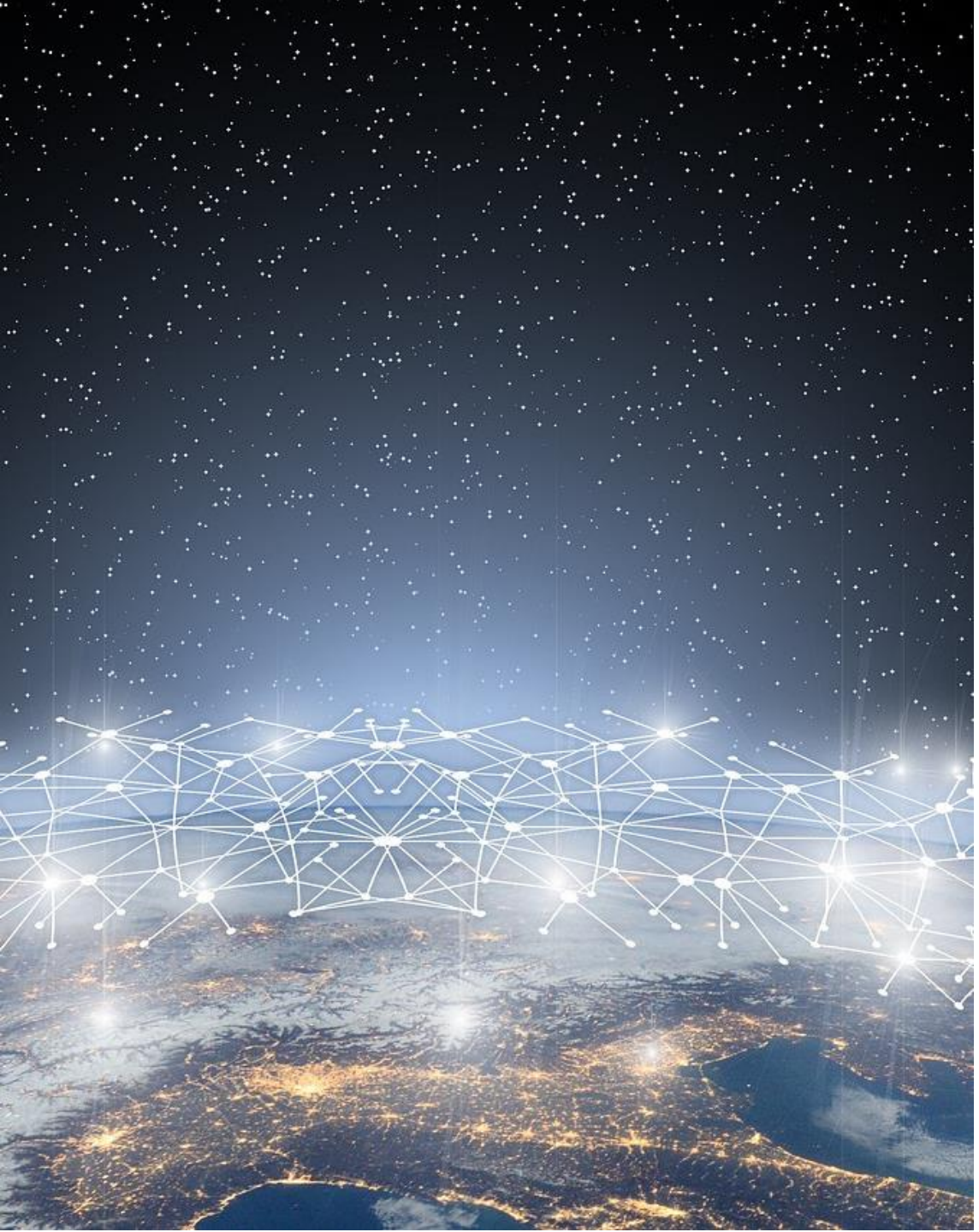
# INTRODUCTION

Problem

## Exploding / Vanishing

## Gradient

(by normalized initialization [23, 9, 37, 13] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with backpropagation [22].)



# INTRODUCTION

Degradation

## Degradation Problem

: Layer ↑ == Performance ↓

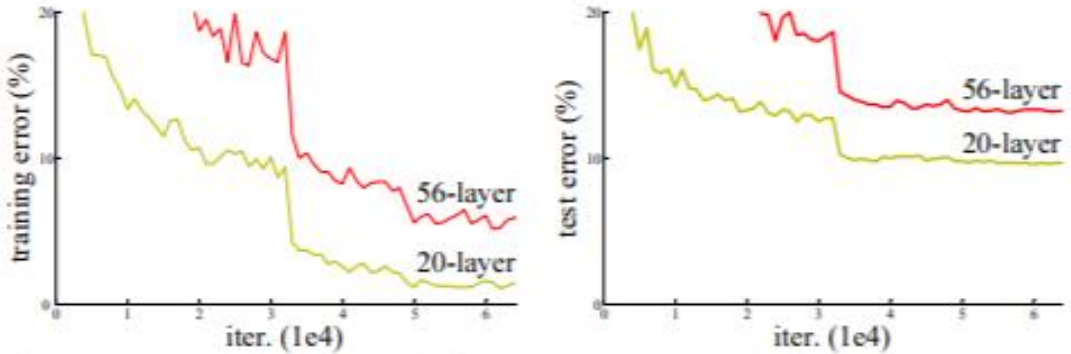
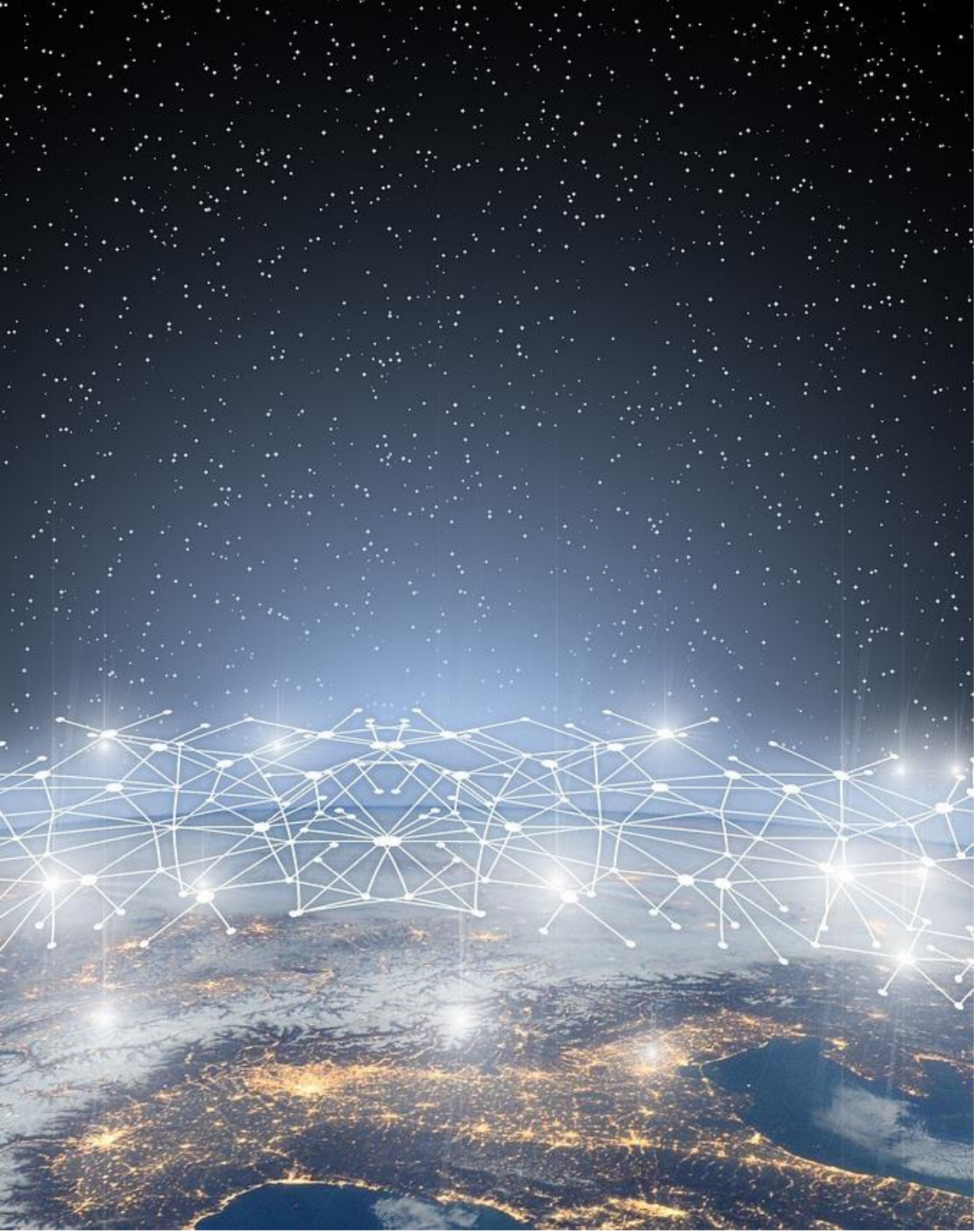



Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

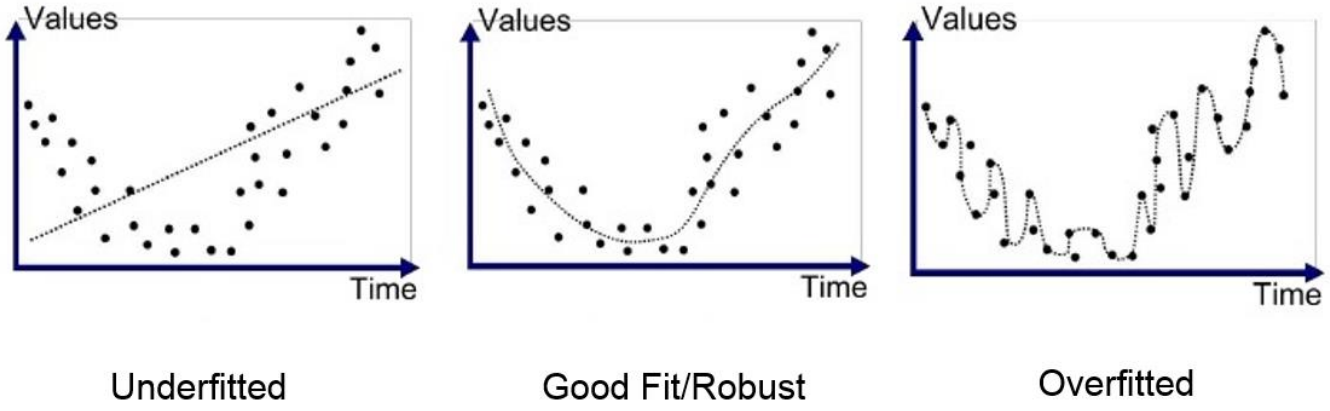


# INTRODUCTION

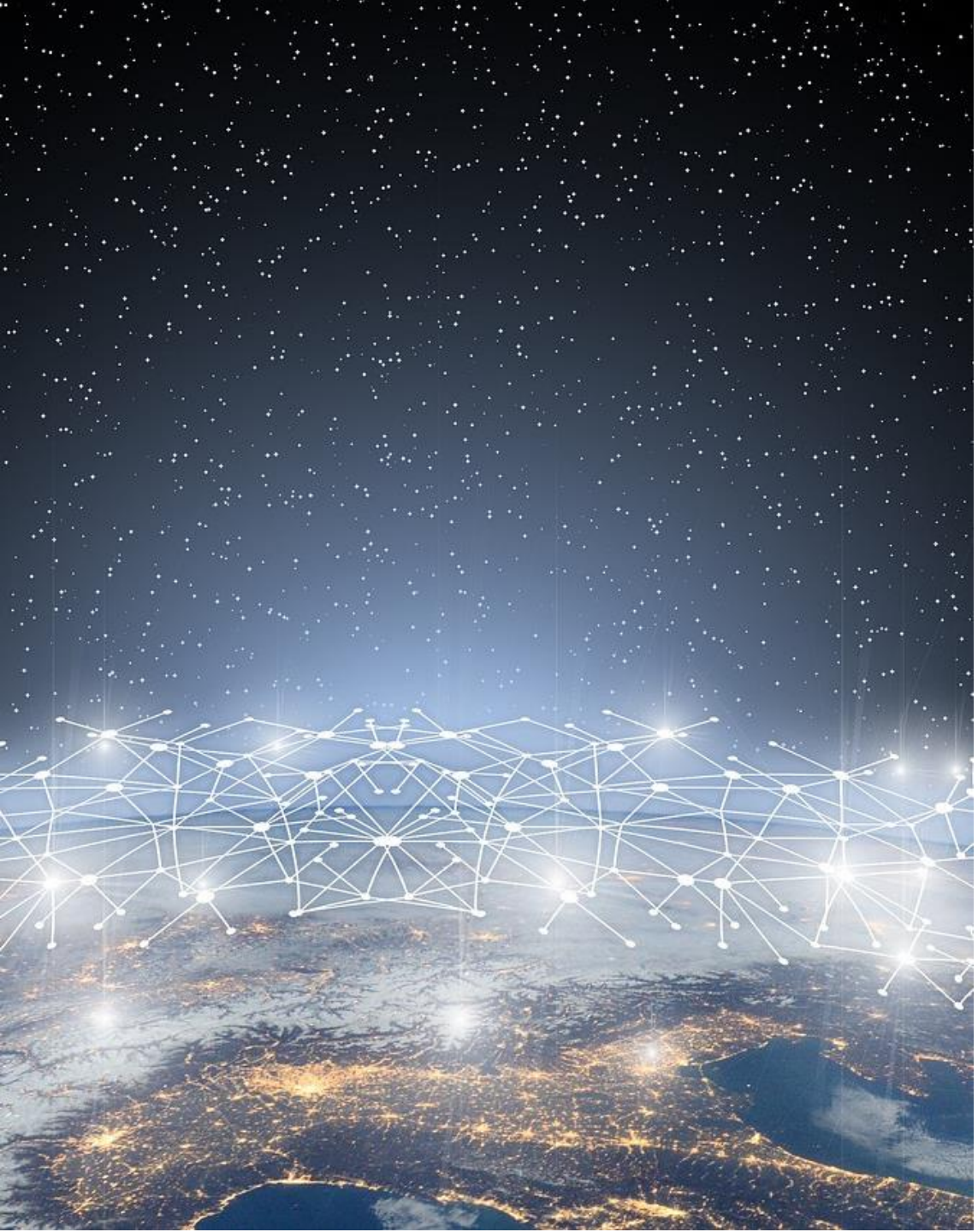
## Attention to ResNet

**Reason why ResNet is in spotlight**

: Nothing to do with overfitting




Underfitted | Good Fit/Robust | Overfitted



# INTRODUCTION

Idea from Degradation

## Result by degradation problem

"Not all systems are similarly easy to optimize"

↓ **Solution**

## Construction to the deeper model

"the added layers are identity mapping, and the other layers are copied from the learned shallower model"

→ Not good Solution

# INTRODUCTION

Idea

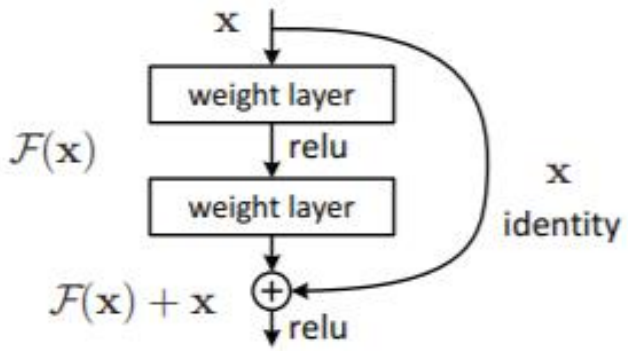



Figure 2. Residual learning: a building block.

$$F(x) = H(x) - x$$

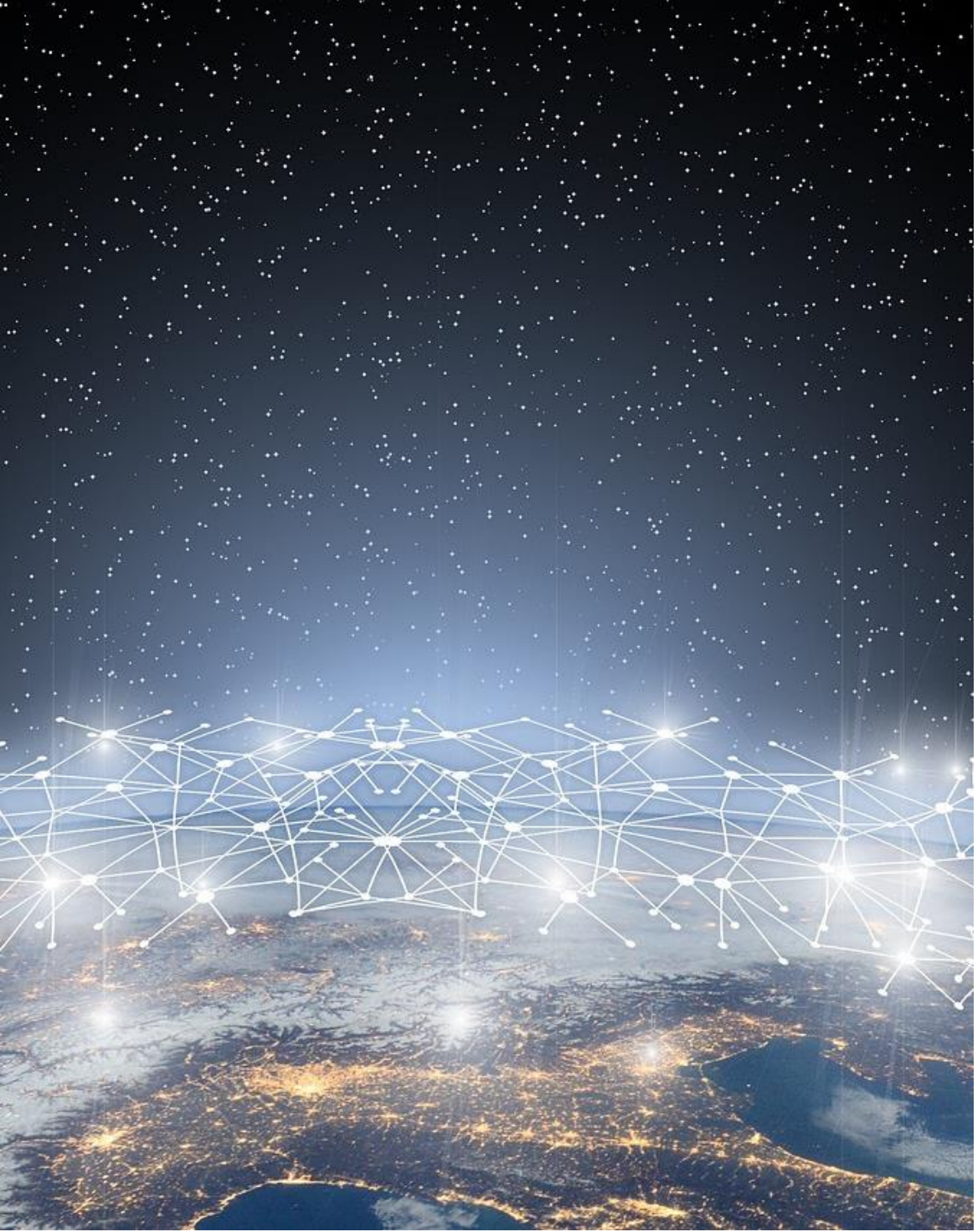
$$H(x) = F(x) + x$$

Postulate residual mapping is suitable for optimization

$F(x) + x$ = **Shortcut Connection**

→ **Skip layers one or more**

- Extra Parameter **X**
- Computational Complexity **X**



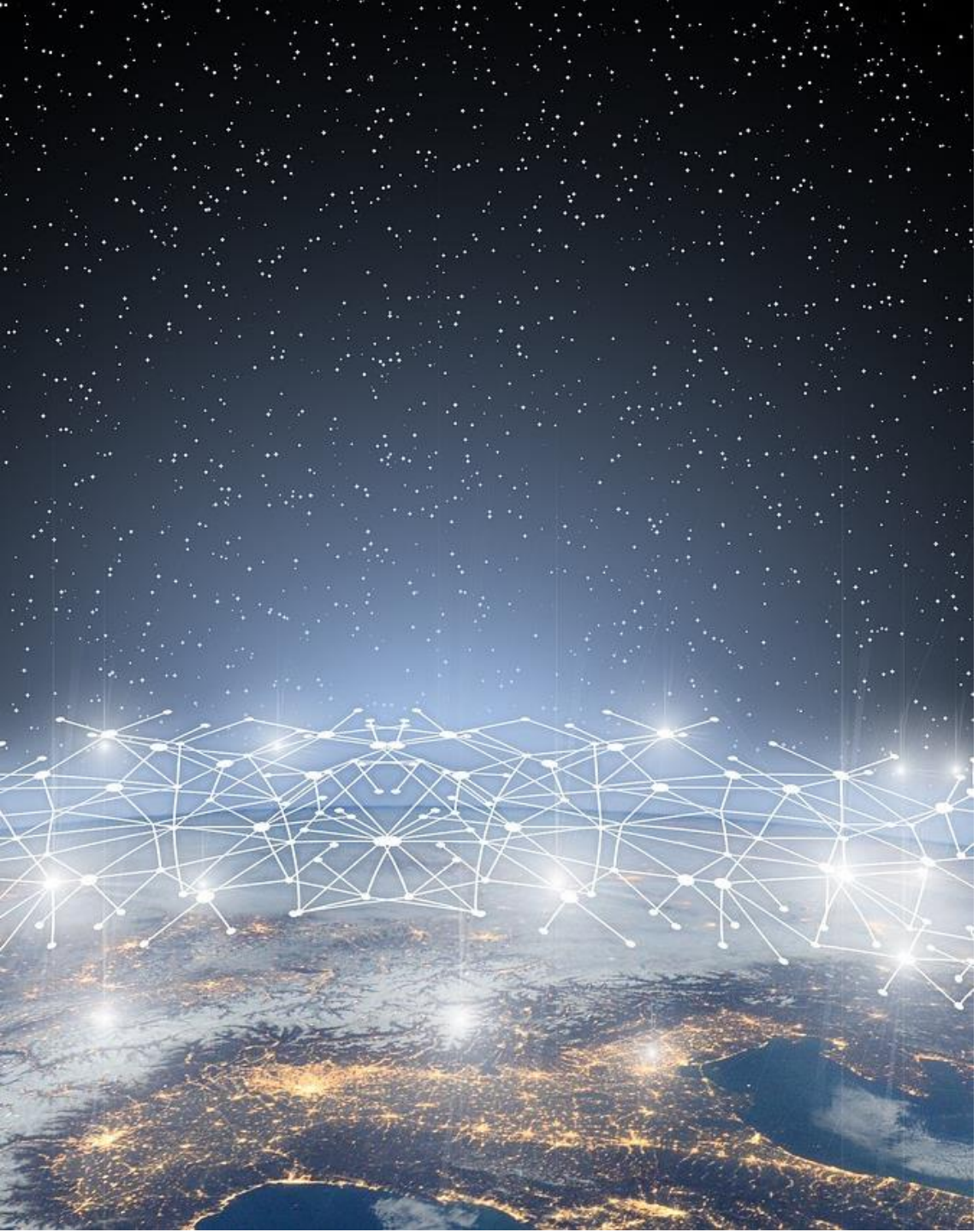
# INTRODUCTION

## Goal

1. **Proof optimization is easier in ResNet than Plain Net**
2. **Accuracy of ResNet is proportional to depth of network**


# DEEP RESIDUAL LEARNING

CHAPTER 2



# DEEP RESIDUAL LEARNING

Residual

## Residual

(A) 남은, 잔여의

(M) $|\hat{y} - y|$

# DEEP RESIDUAL LEARNING

## Shortcut

- Identity shortcut

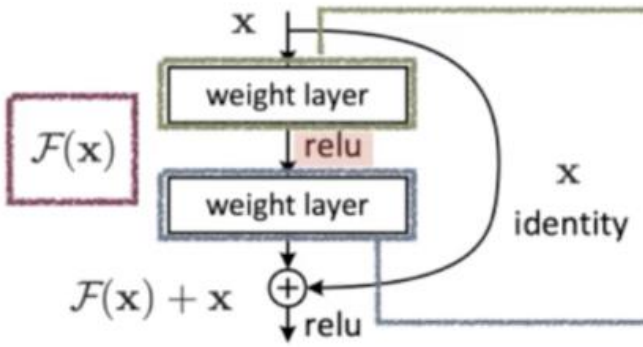
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

  - y: output vector of the layer
  - x: input (identity) vector of the layer
  - W: weight

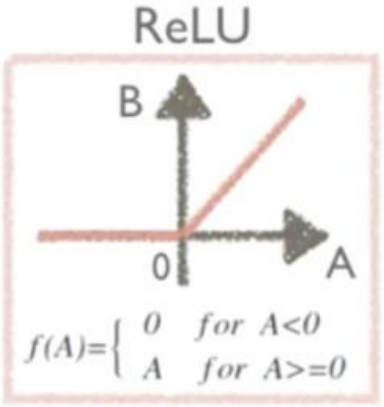
- Projection shortcut

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$$

  - y: output vector of the layer
  - x: input vector (identity) of the layer
  - W: weight
  - s: square matrix

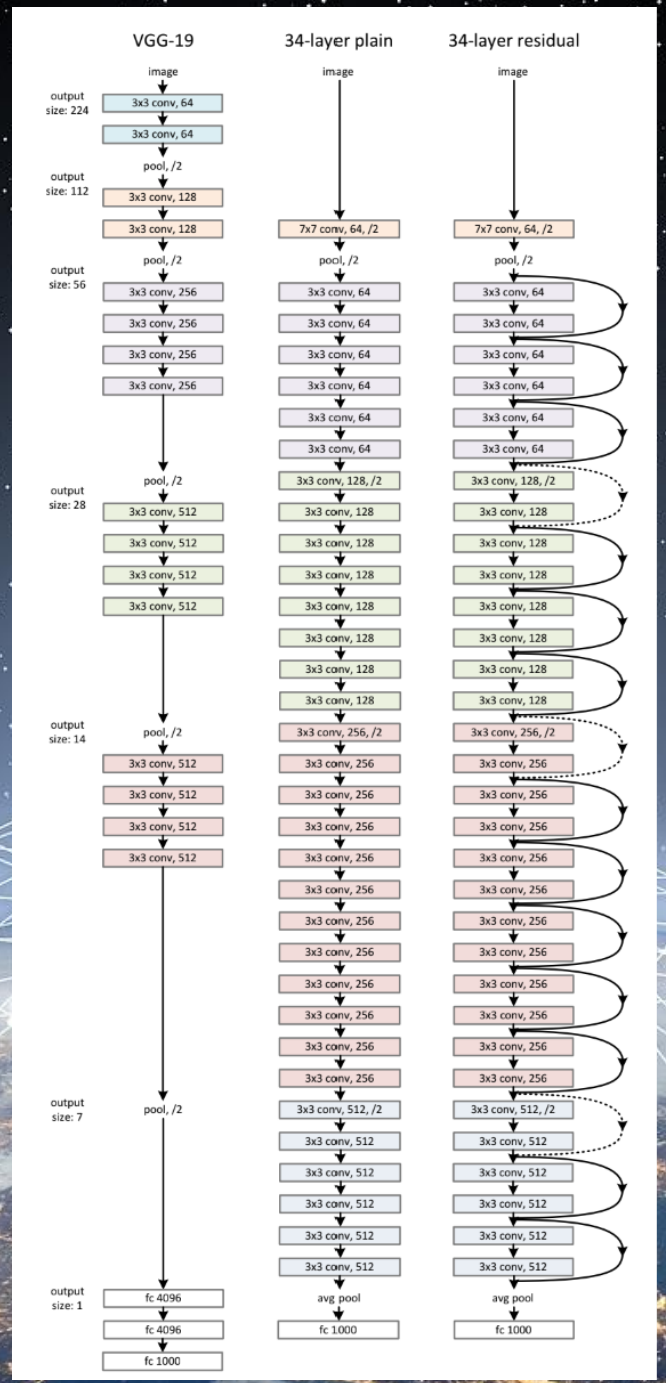



$$\mathcal{F} = W_2\sigma(W_1\mathbf{x})$$

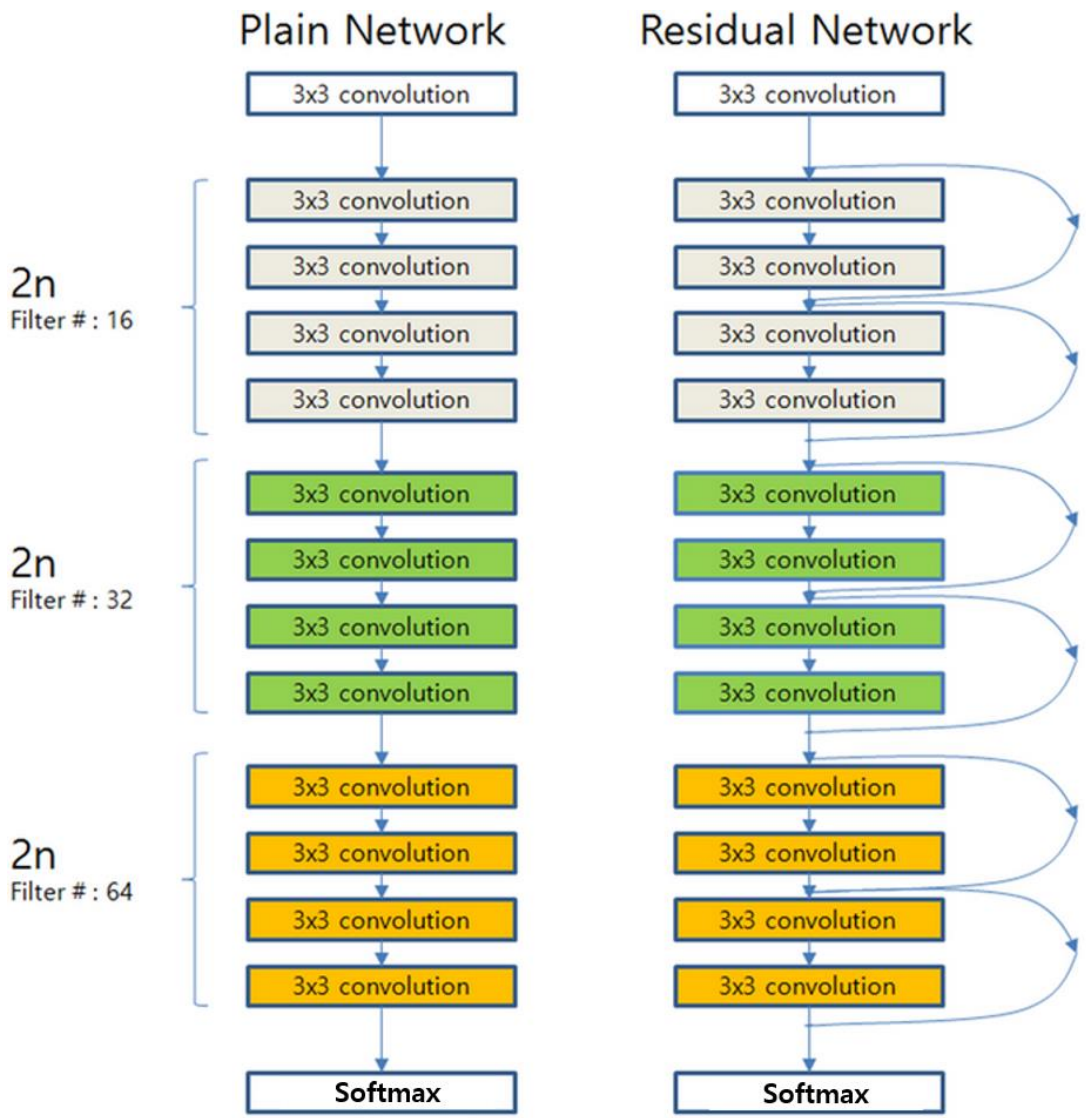



# ARCHITECTURE

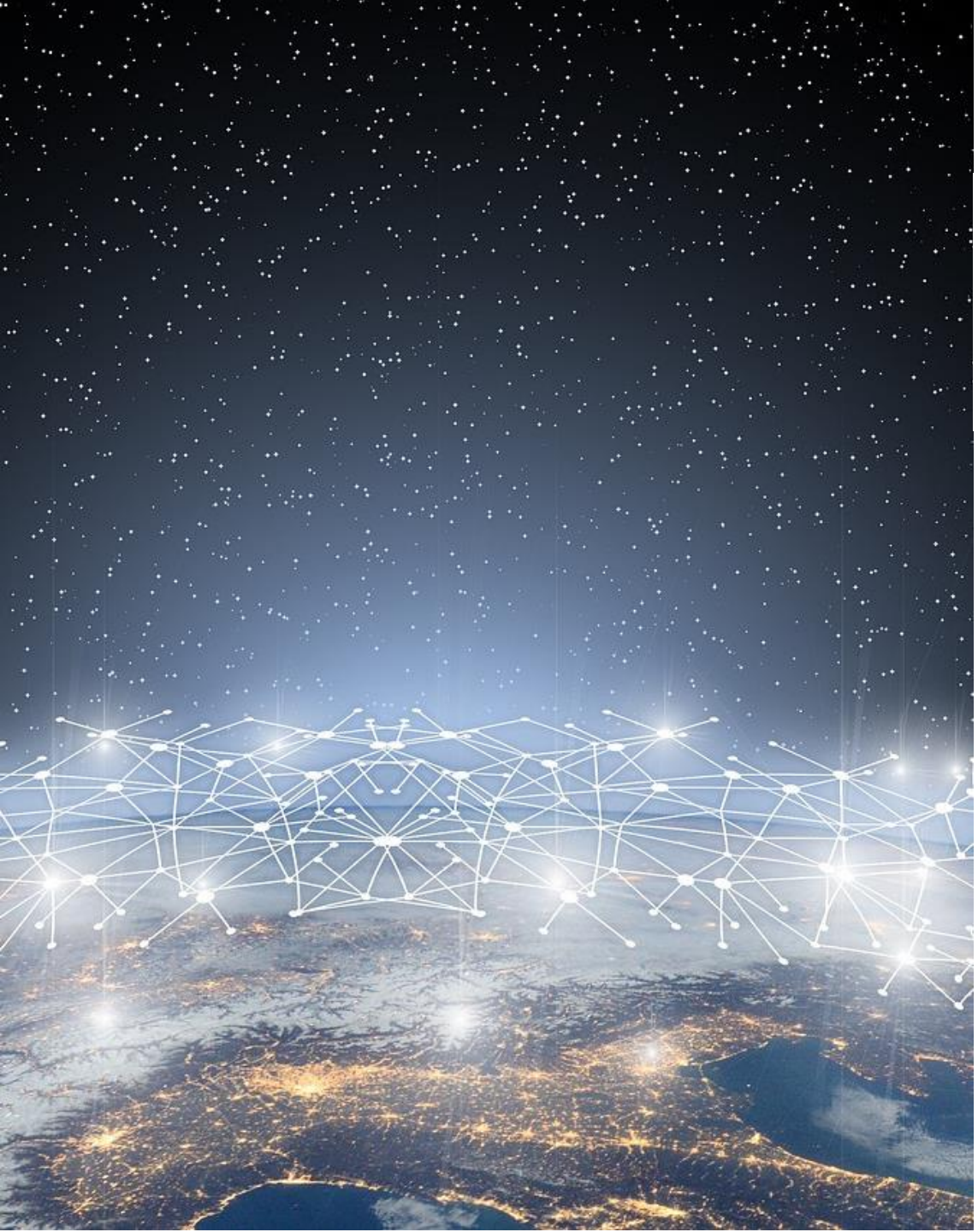
CHAPTER 3



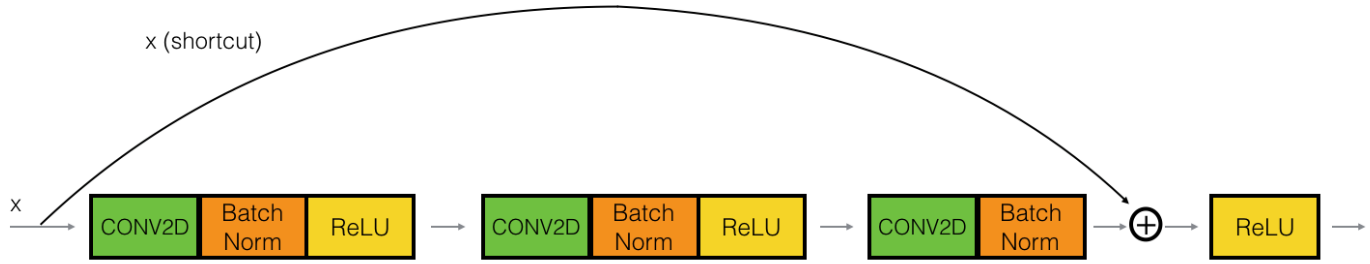

# Architecture

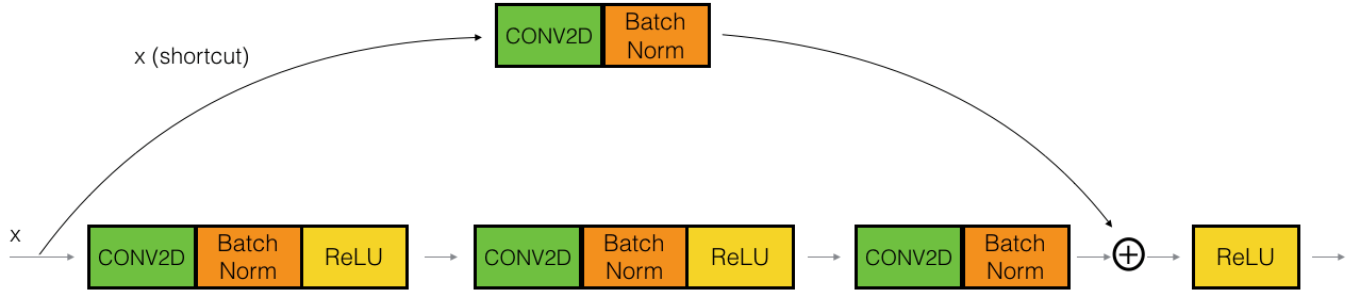


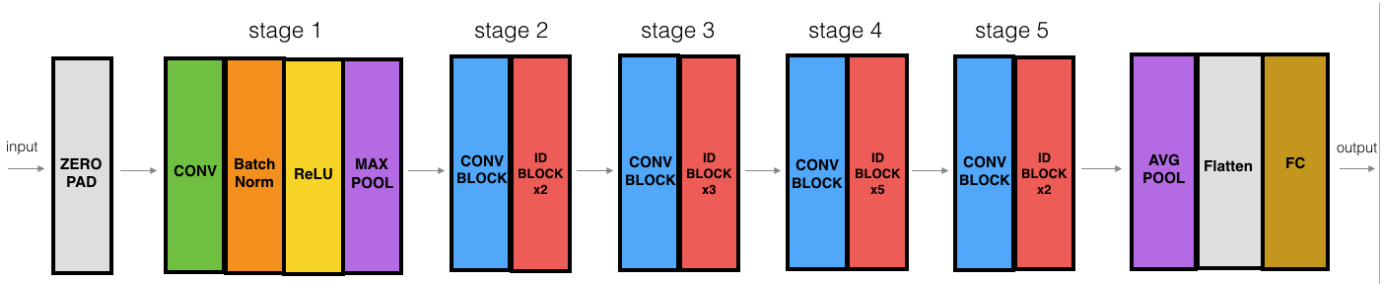

Importantly, dimension of Input and Output must be same

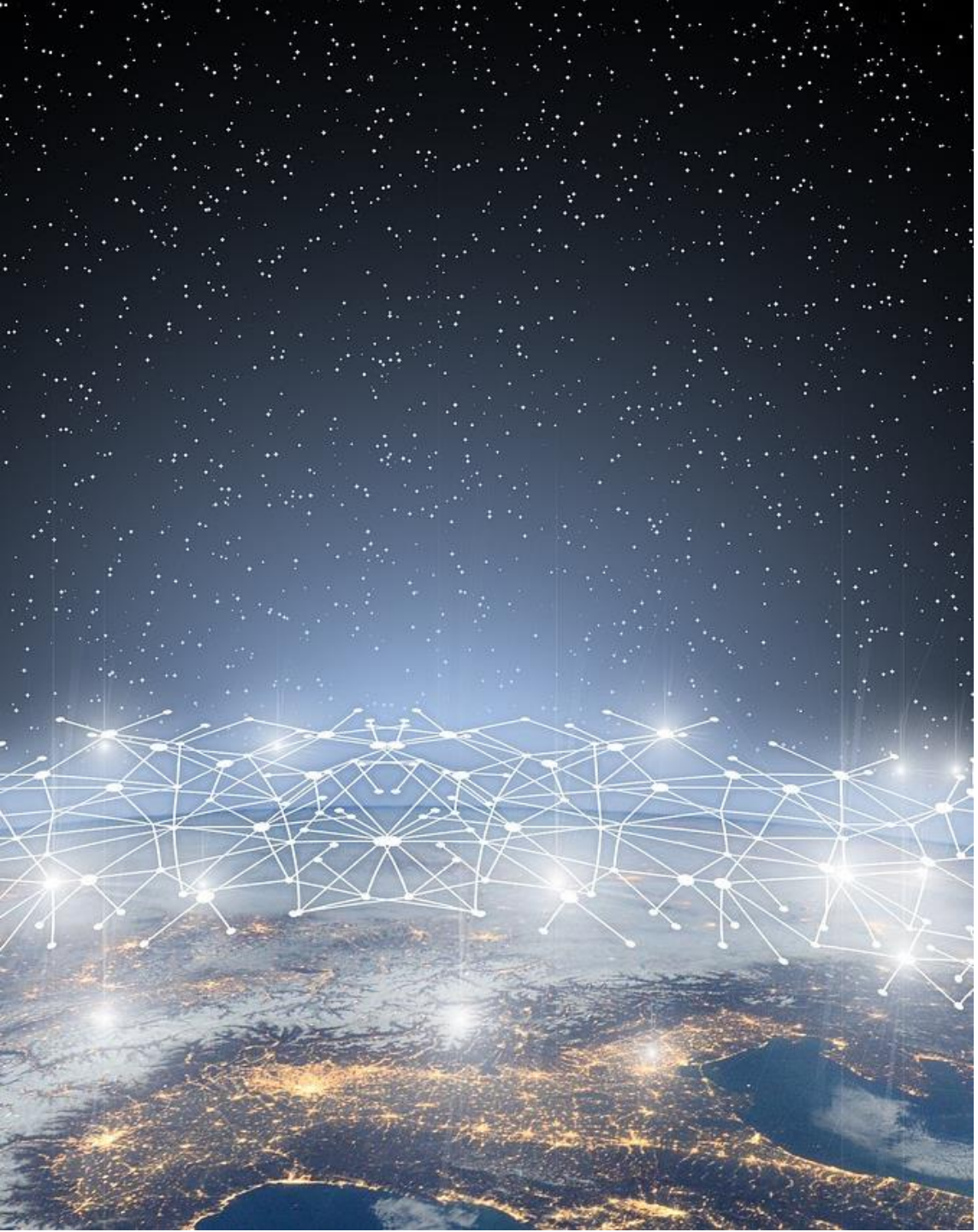


# Architecture



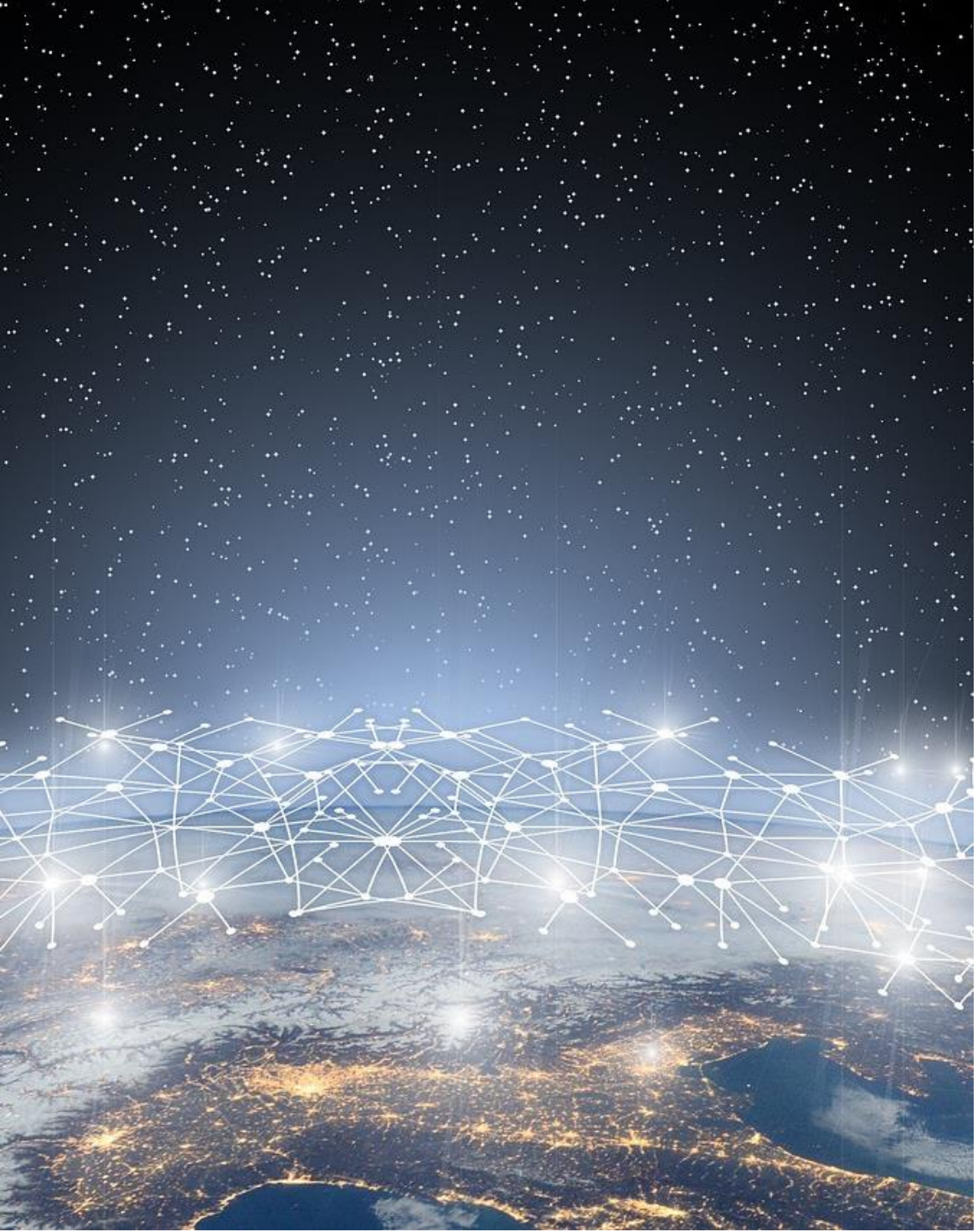





# ARCHITECTURE

## Plain vs Residual

| Plain | Residual |
|---|---|
| - VGG Net<br>- Same #filter<br>- Feature map size * ½ = #filter * 2<br>- Conv layer = 3X3, stride = 2<br>- Softmax<br>- 1000 fully-connected layer<br><br>-> 18% of VGG Net | - Plain Net + shortcut connection<br>- dim(Input) == dim(Output)<br><br>(A) Use Zero-padding to make dimension equal<br>– extra parameter X<br>(B) Use linear projection that is used in $y = F(x, \{W_i\}) + W_s x$ |



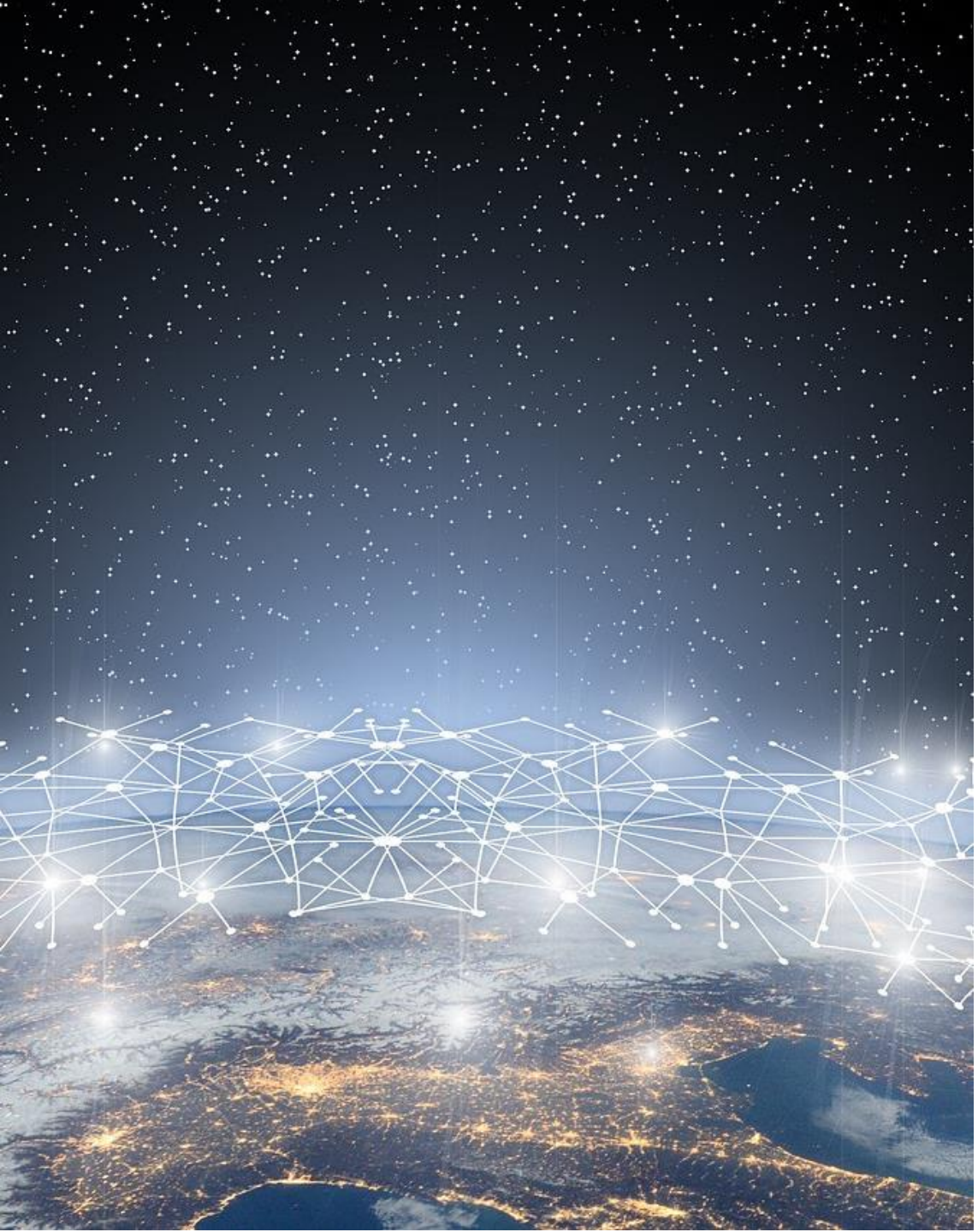
# ARCHITECTURE

Implementation

| Features |
| --- |
| 1. Image resized 224*224<br>2. Batch Normalization<br>3. Initialize Weights<br>4. SGD<br>5. 256 mini batches<br>6. Learning rate = 0.1<br>7. Iteration = 60*10^4<br>8. Weight decay = 0.0001<br>9. Momentum = 0.9<br>10. No dropout |


# EXPERIMENT

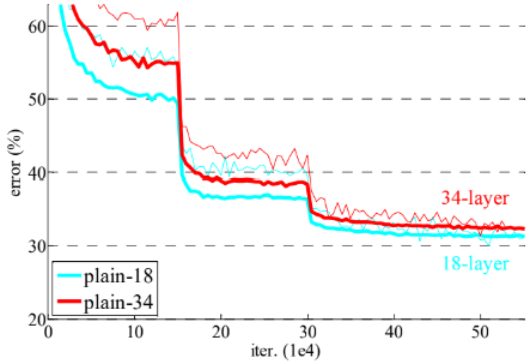
CHAPTER 4

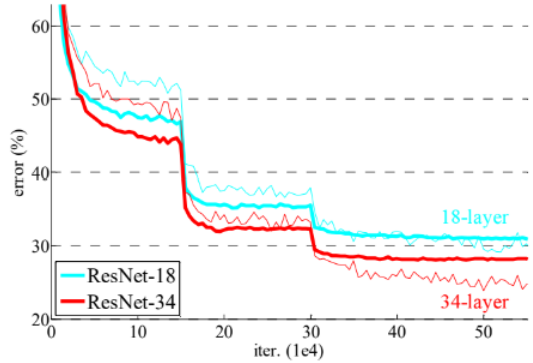


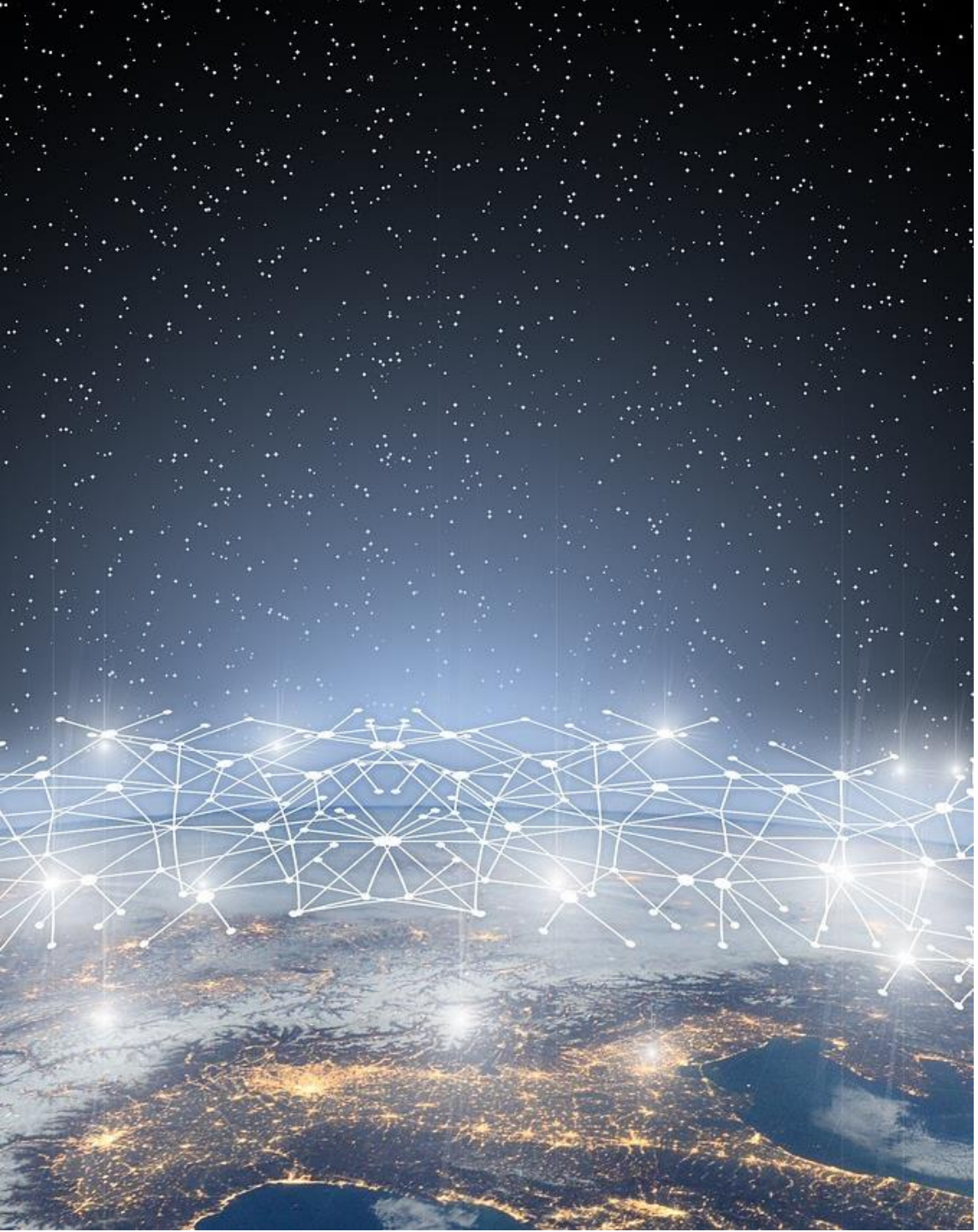
# EXPERIMENT

## ImageNet Classification

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| | | 3×3 max pool, stride 2 | | | | |
| conv2_x | 56×56 | [3×3, 64; 3×3, 64] ×2 | [3×3, 64; 3×3, 64] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 |
| conv3_x | 28×28 | [3×3, 128; 3×3, 128] ×2 | [3×3, 128; 3×3, 128] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×8 |
| conv4_x | 14×14 | [3×3, 256; 3×3, 256] ×2 | [3×3, 256; 3×3, 256] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×23 | [1×1, 256; 3×3, 256; 1×1, 1024] ×36 |
| conv5_x | 7×7 | [3×3, 512; 3×3, 512] ×2 | [3×3, 512; 3×3, 512] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |





# EXPERIMENT

## ImageNet Classification

| | plain | ResNet |
|---|---|---|
| 18 layers | 27.94 | 27.88 |
| 34 layers | 28.54 | **25.03** |

**<Residual Network>**

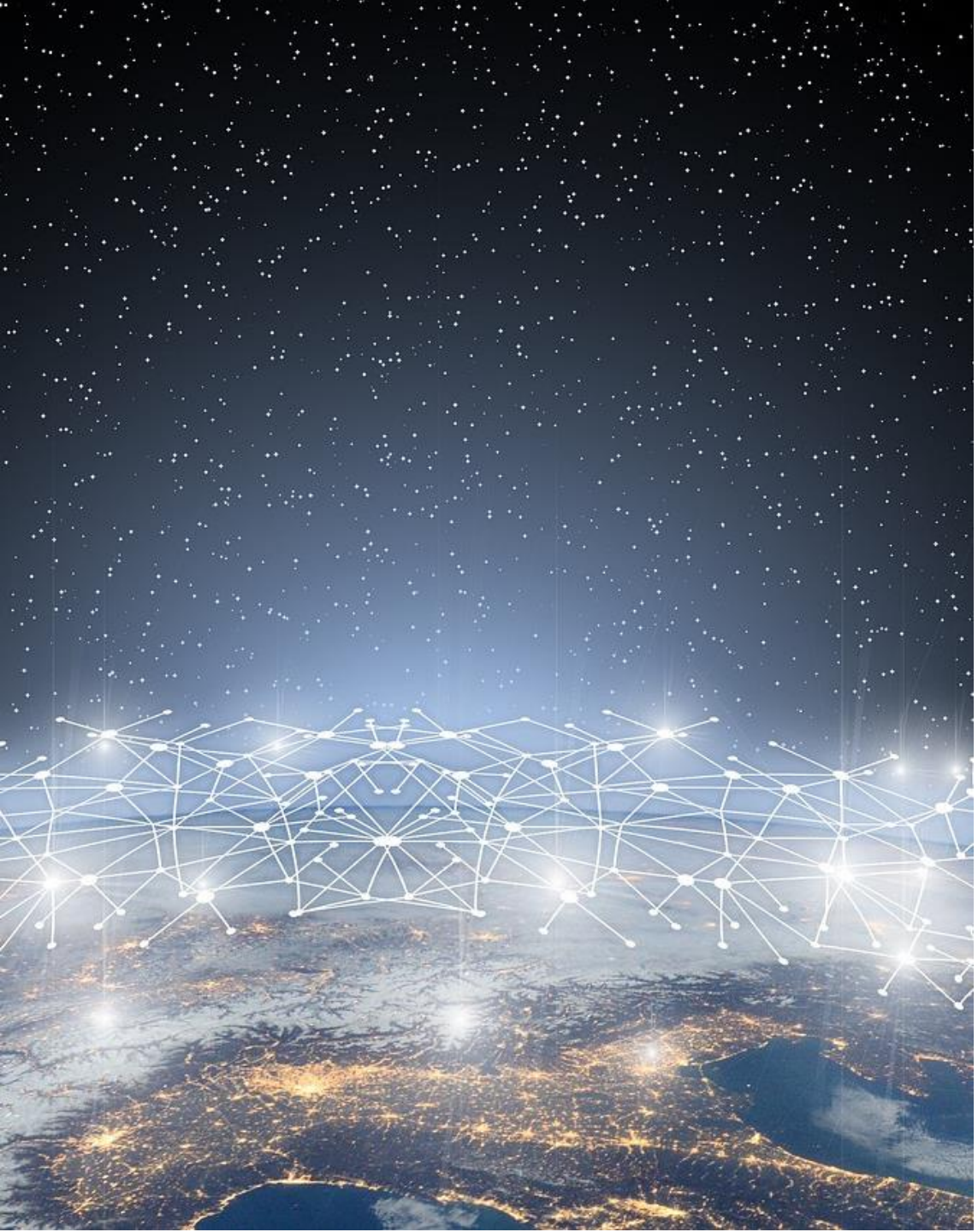
1. Performance : 18-Layer < 34-Layer

→ Degradation problem was effectively managed despite the increased depth of layers

2. Training error ↓

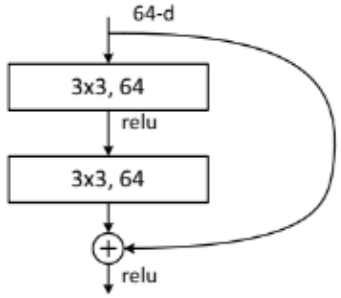
3. Similar accuracy, but 18-layer's convergence speed is faster

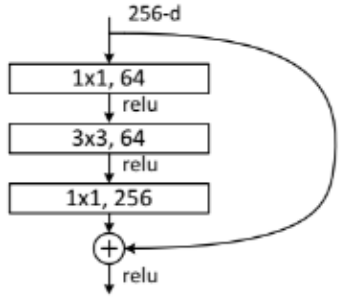
→ Optimization with SGD is easier in ResNet



# EXPERIMENT

## ImageNet Classification




**<Identity vs Projection Shortcuts>**

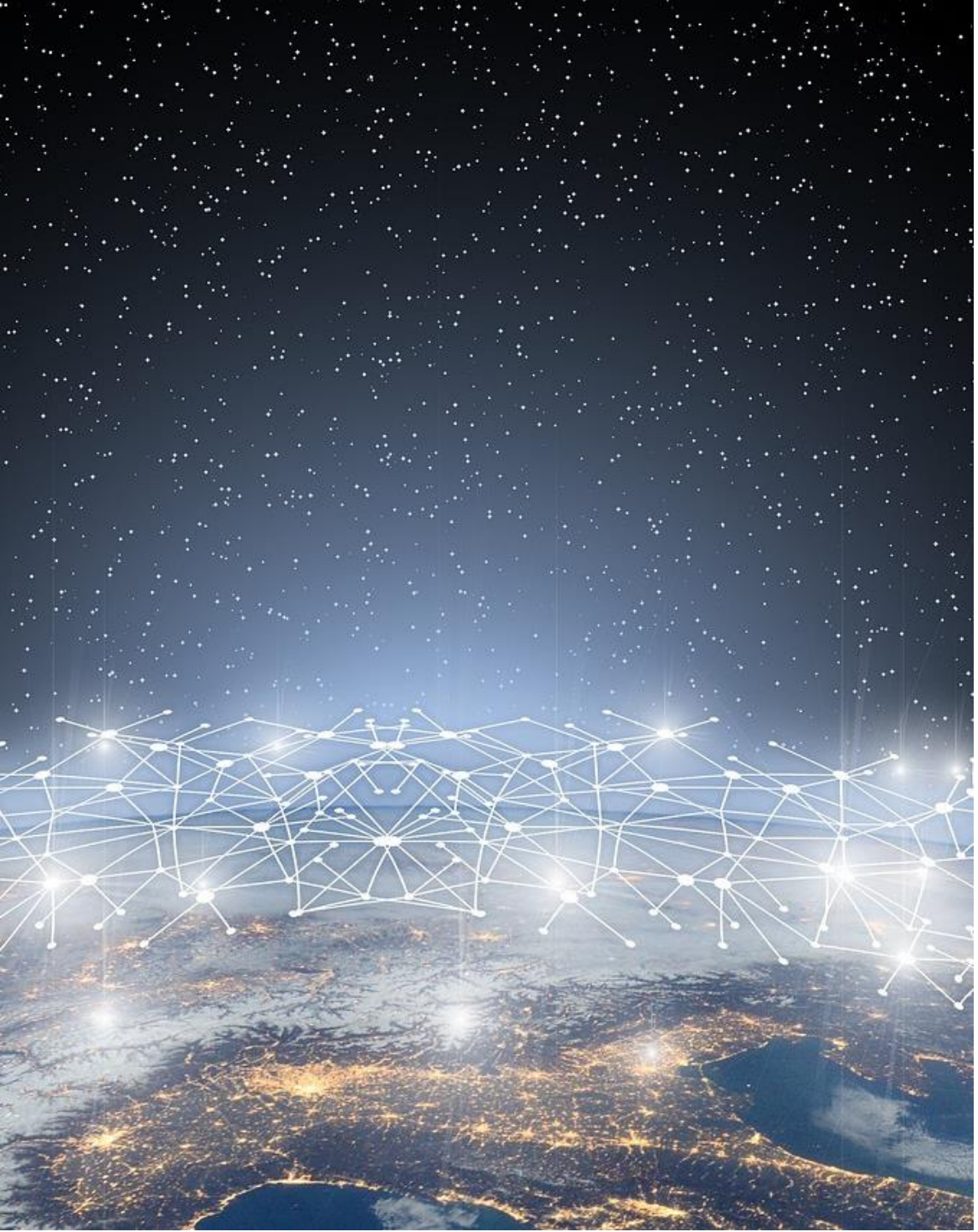
Compare three options

(A) zero-padding shortcuts (parameter free)

(B) projection shortcuts (others are identity)

(C) All shortcuts are projections

**→ degradation problem // projection shortcut**



# EXPERIMENT

## ImageNet Classification

**<Deeper Bottleneck Architecture>**

For reducing and then increasing (restoring) dimensions

(1 x 1) -> (3 x 3) -> (1 x 1)

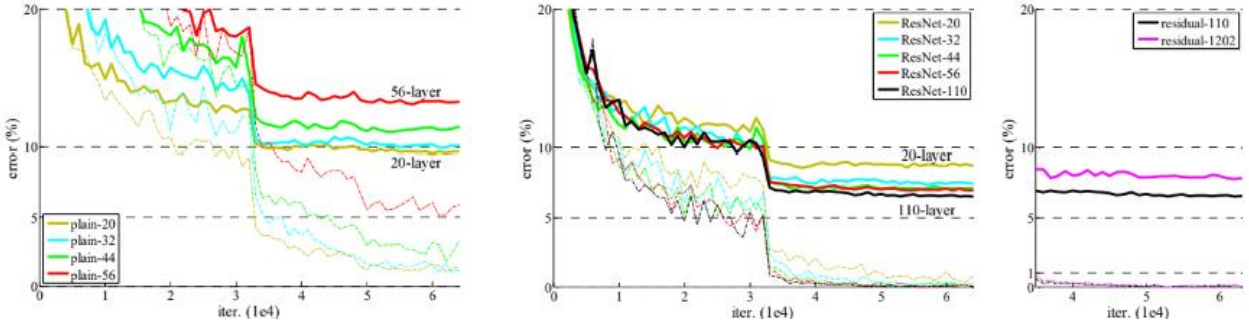
1. 50-layer
2. 101-layer and 152-layer ResNet
3. Comparisons with state-of-art Methods

| output map size | 32×32 | 16×16 | 8×8 |
|---|---|---|---|
| # layers | 1+2$n$ | 2$n$ | 2$n$ |
| # filters | 16 | 32 | 64 |

| method | | | error (%) |
|---|---|---|---|
| Maxout [10] | | | 9.38 |
| NIN [25] | | | 8.81 |
| DSN [24] | | | 8.22 |
| | # layers | # params | |
| FitNet [35] | 19 | 2.5M | 8.39 |
| Highway [42, 43] | 19 | 2.3M | 7.54 (7.72±0.16) |
| Highway [42, 43] | 32 | 1.25M | 8.80 |
| ResNet | 20 | 0.27M | 8.75 |
| ResNet | 32 | 0.46M | 7.51 |
| ResNet | 44 | 0.66M | 7.17 |
| ResNet | 56 | 0.85M | 6.97 |
| ResNet | 110 | 1.7M | **6.43** (6.61±0.16) |
| ResNet | 1202 | 19.4M | 7.93 |

# EXPERIMENT

## CIFAR-10 and Analysis

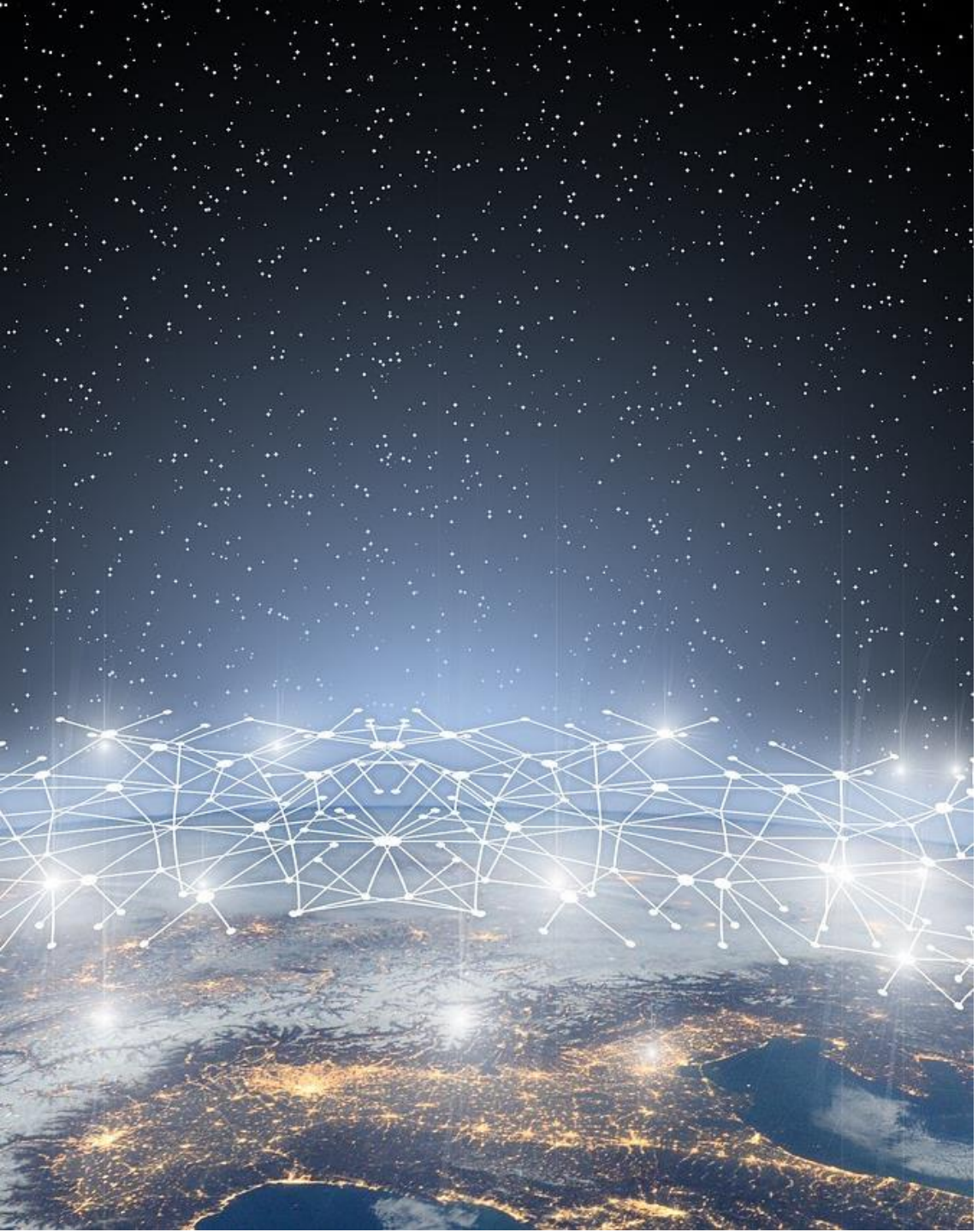



#layers and performances are proportional

But, when trying more than 1000 layers, its error rate is worse than before because of **overfitting**.

So, it has to apply regularization methods.

(ex) **maxout**, **dropout**



# EXPERIMENT

## Object Detection on PASCAL and MS COCO

| training data | 07+12 | 07++12 |
|---|---|---|
| test data | VOC 07 test | VOC 12 test |
| VGG-16 | 73.2 | 70.4 |
| ResNet-101 | **76.4** | **73.8** |

Table 7. Object detection mAP (%) on the PASCAL VOC 2007/2012 test sets using **baseline** Faster R-CNN. See also Table 10 and 11 for better results.

| metric | mAP@.5 | mAP@[.5, .95] |
|---|---|---|
| VGG-16 | 41.5 | 21.2 |
| ResNet-101 | **48.4** | **27.2** |

Table 8. Object detection mAP (%) on the COCO validation set using **baseline** Faster R-CNN. See also Table 9 for better results.

Adopt **Faster R-CNN** as detection method


# CONCLUSION

CHAPTER 5


# CONCLUSION

Proof

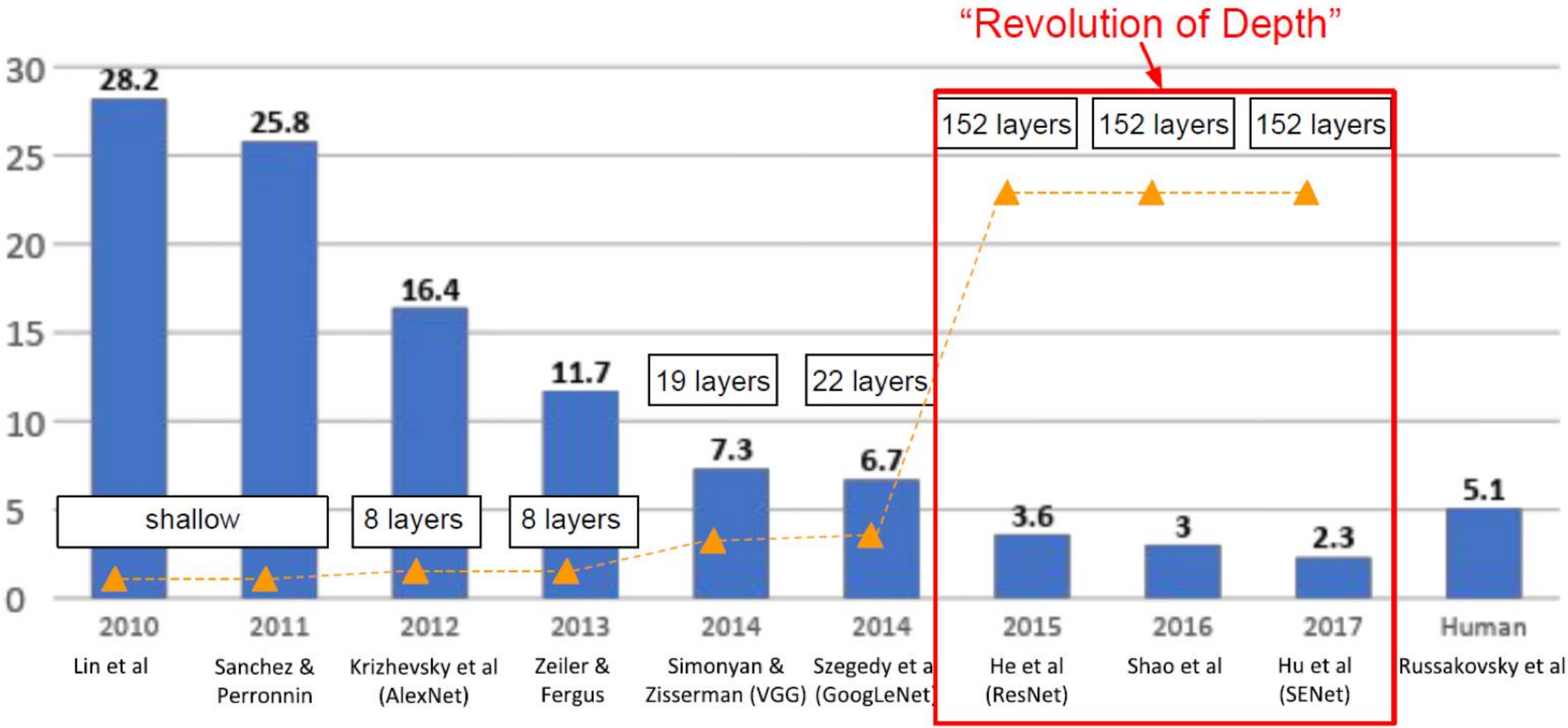
1. **Proof optimization is easier in ResNet than Plain Net**

   → degradation problem decreased

2. **Accuracy of ResNet is proportional to depth of network**

   → Training 1000+ layers

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# THANK YOU

The End