Classification (2)

POSTECH MIP Lab.

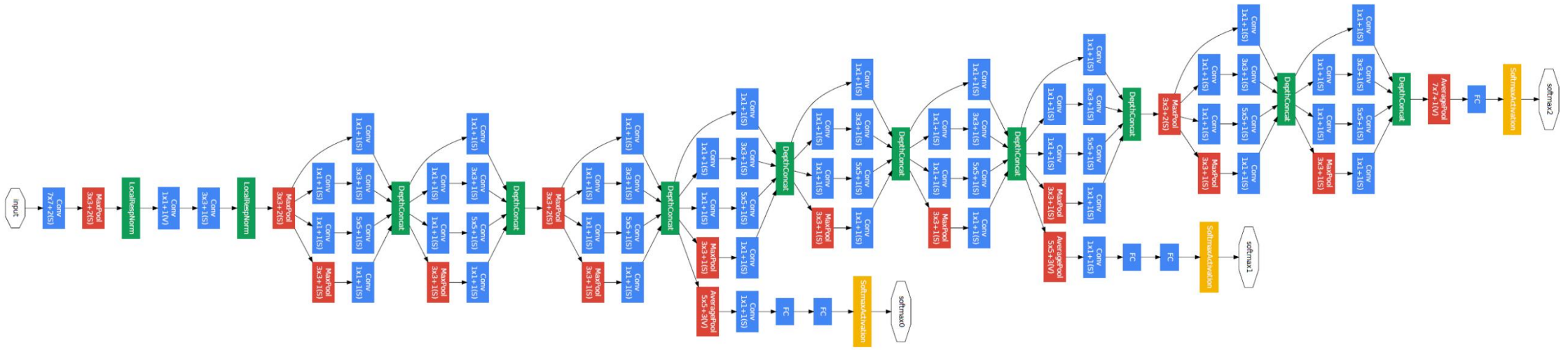
TA: Joonhyuk Park, Seunghun Baek, Soojin Hwang

Goal

- Understand classification task
- Understand the development of CNN based classification models (AlexNet, VGG, **GoogLeNet, ResNet**)
- Learn how to implement CNN models from architecture

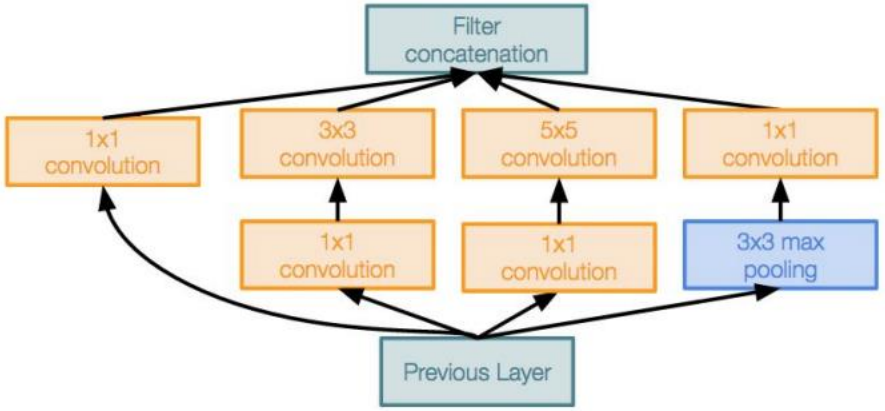
GoogLeNet (2014)

- Deeper networks (22 layers)
- Use efficient Inception module
  - Only 5 million parameters (12 times less than AlexNet)
- ILSVRC'14 winner



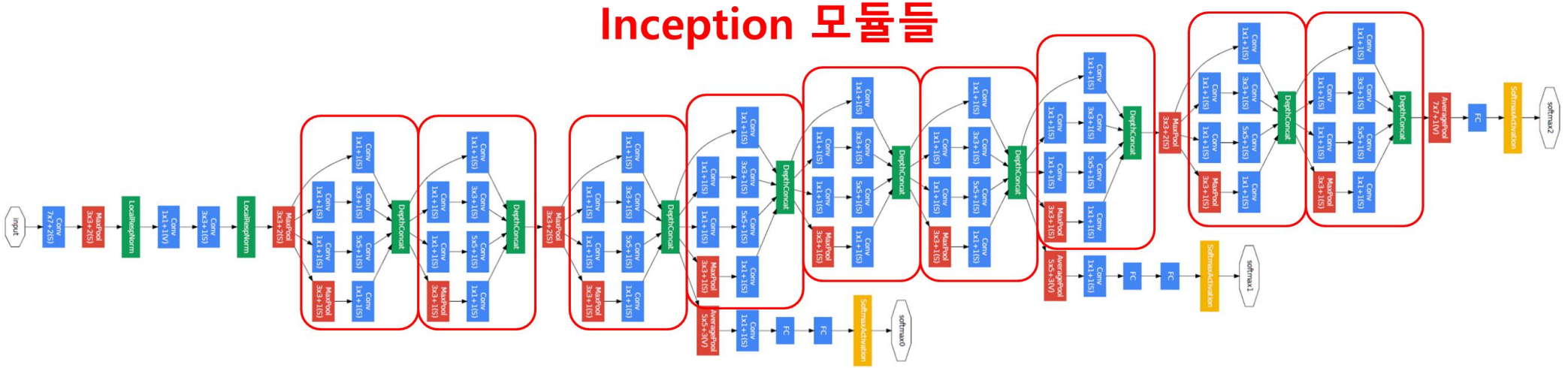
GoogLeNet (2014)

- Inception Module?



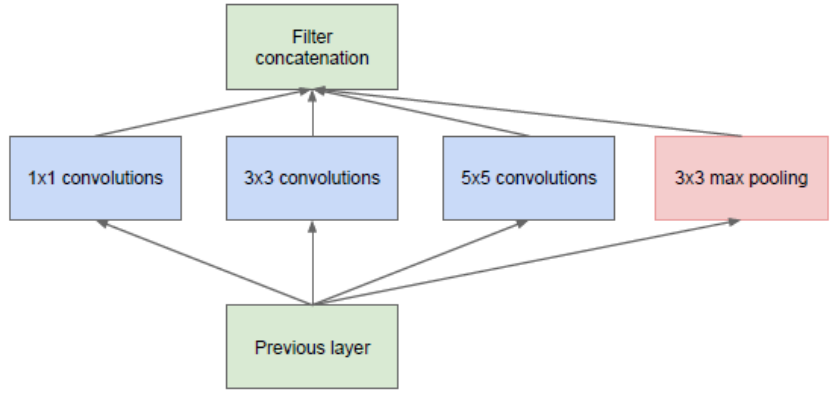
Inception module

Inception 모듈들

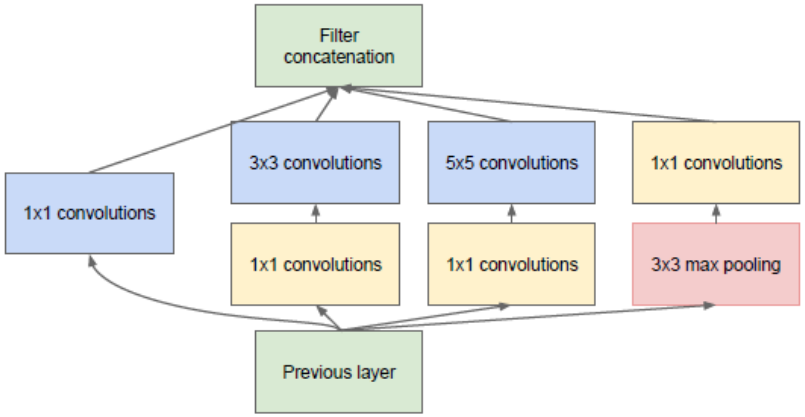


GoogLeNet (2014)

- Inception Module?
  - Apply parallel filter operations on the input from previous layer
  - 1*1 conv, 3*3 conv, 5*5 conv, 3*3 maxpool
  - -> problem: computational complexity



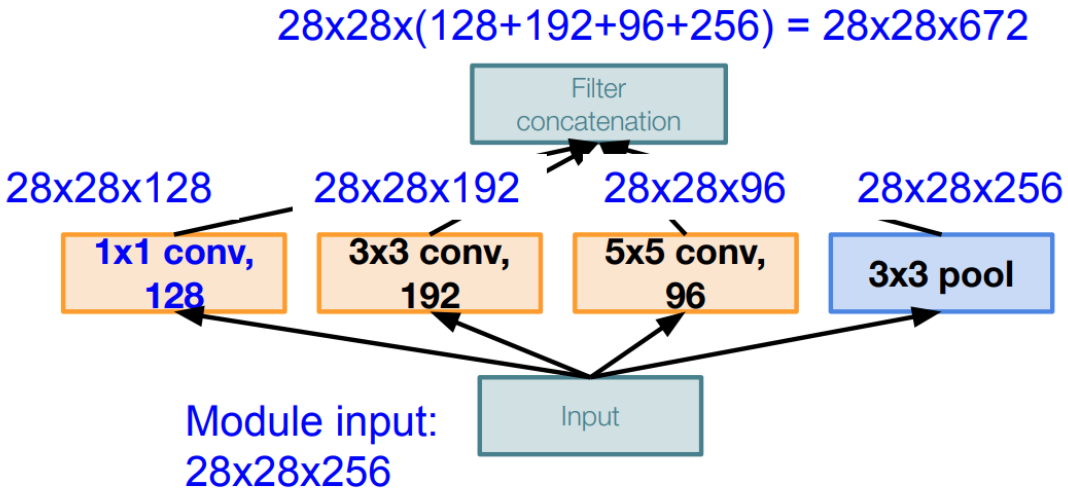
(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

GoogLeNet (2014)

- Inception Module?
  - problem: computational complexity
  - Ex)



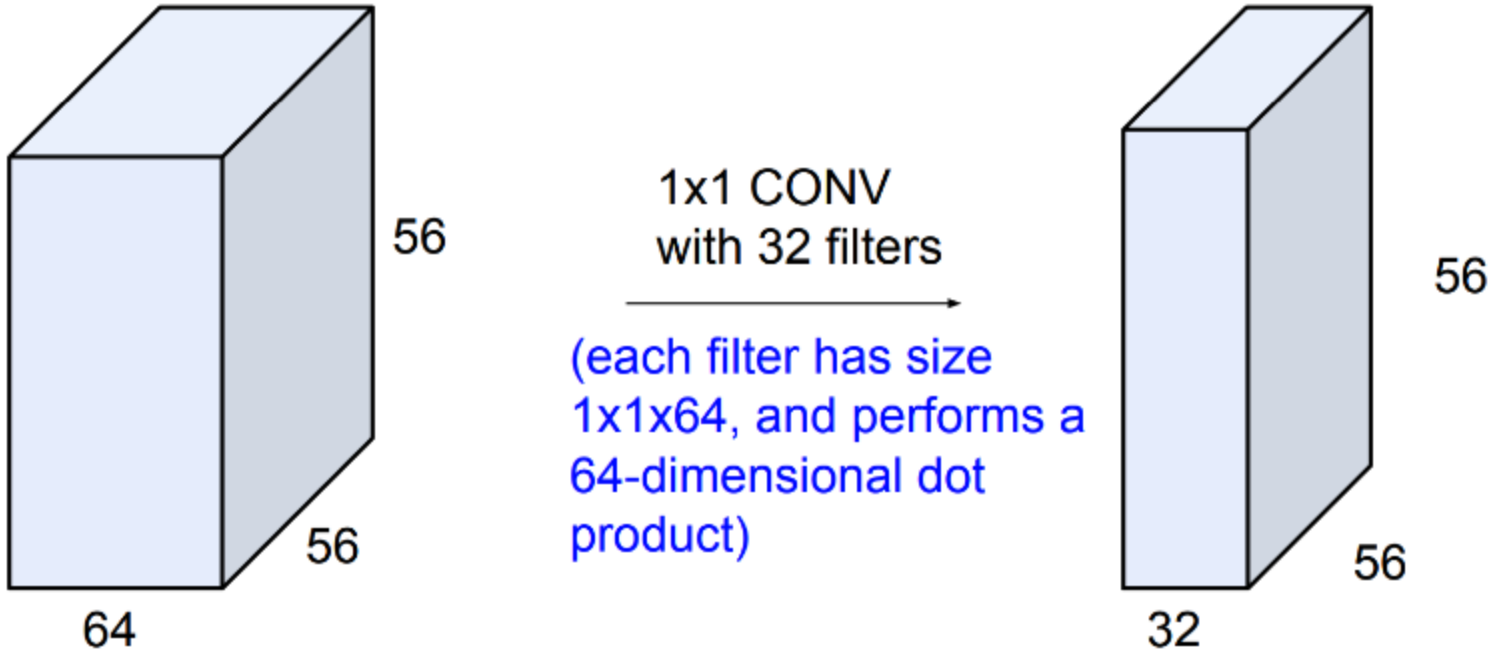
Naive Inception module

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x**192x3x3x256**
[5x5 conv, 96] 28x28x**96x5x5x256**
Total: 854M ops

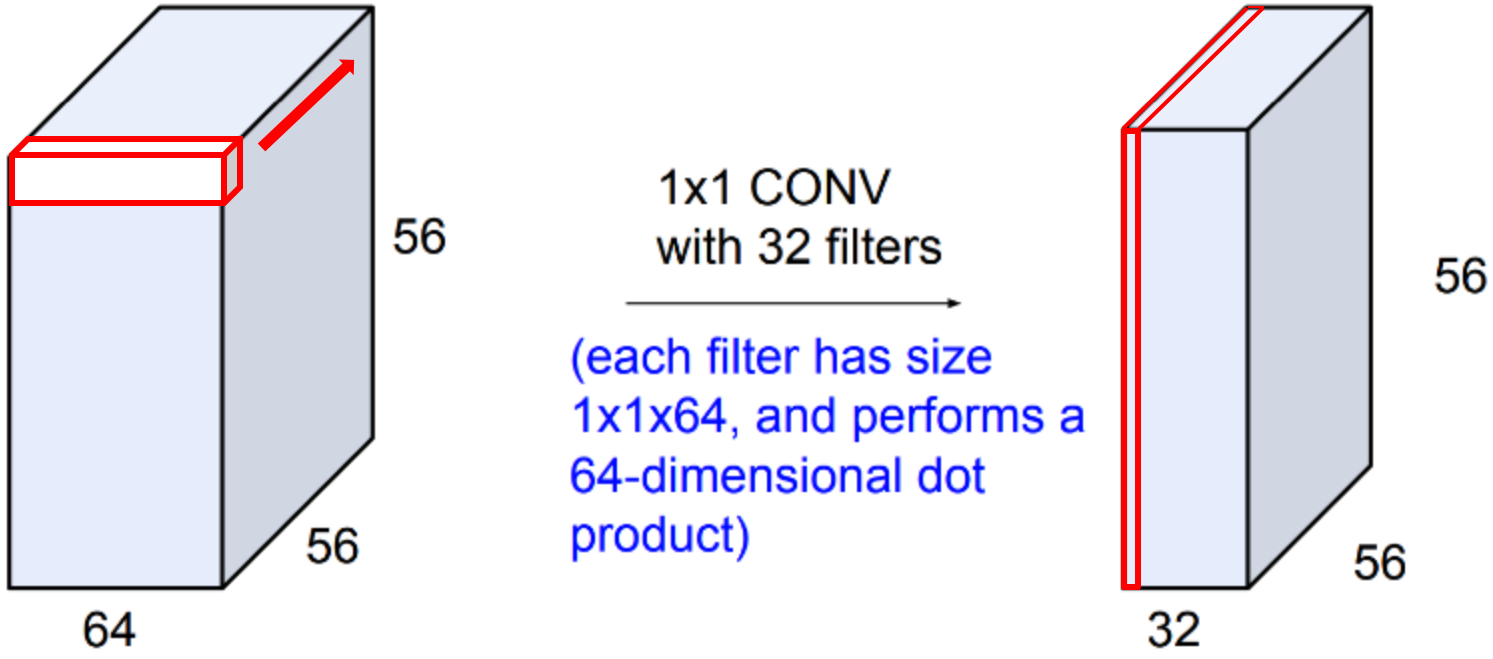
GoogLeNet (2014)

- Inception Module?
  - problem: computational complexity
  - Solution: bottleneck layer (1*1 convolutions)

1 x 1 Convolution



1 x 1 Convolution

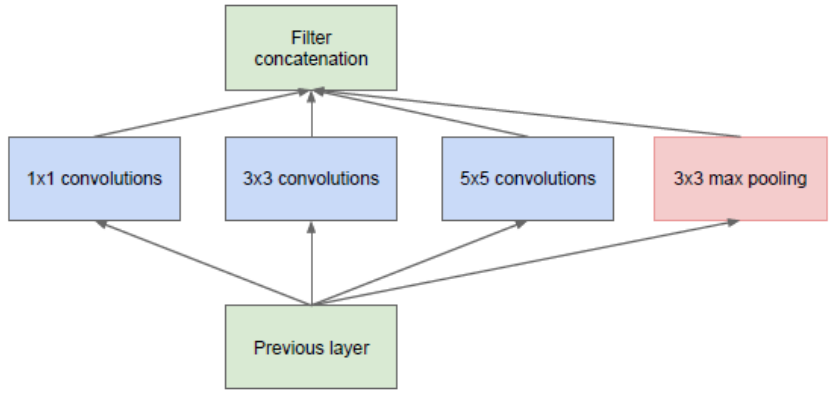


1 x 1 Convolution

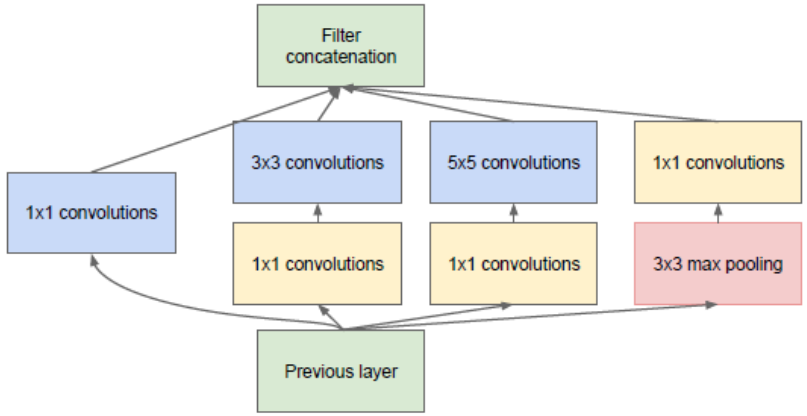
- Preserves spatial dimensions, but reduces depth!
- Projects depth to lower dimension (combination of feature maps)

GoogLeNet (2014)

- Inception Module?
  - problem: computational complexity
  - Solution: bottleneck layer (1*1 convolutions)



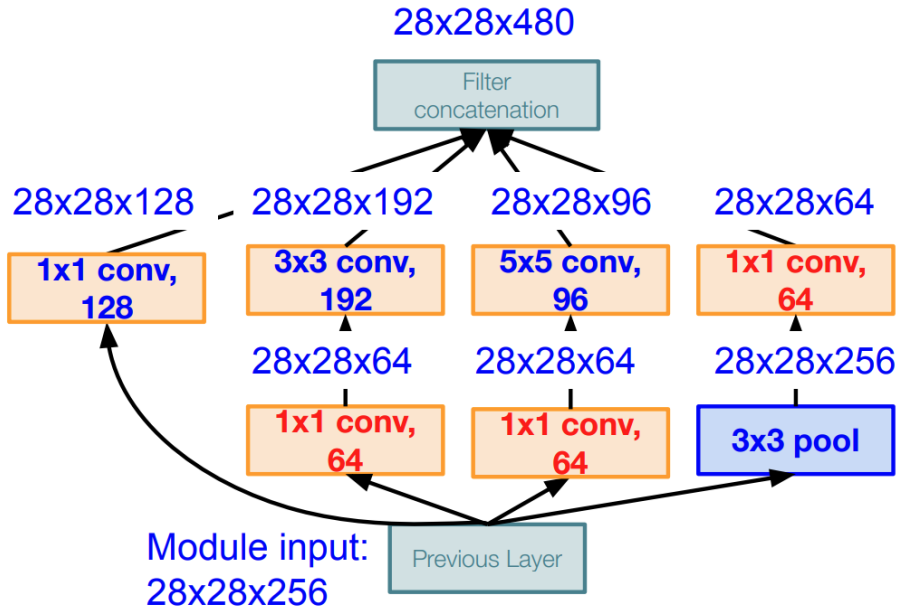
(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

GoogLeNet (2014)

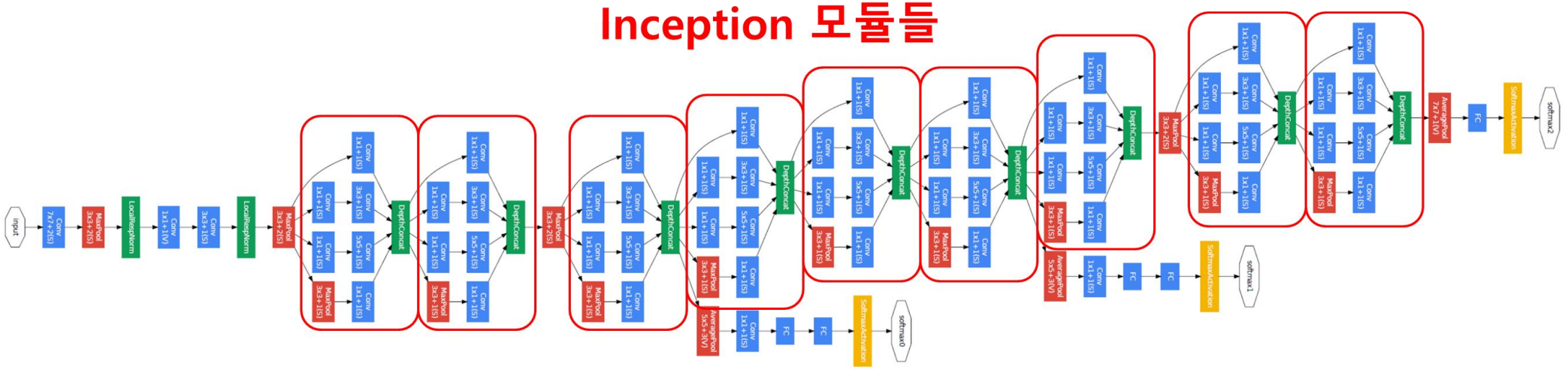
- Inception Module?
  - problem: computational complexity
  - Solution: bottleneck layer (1*1 convolutions)



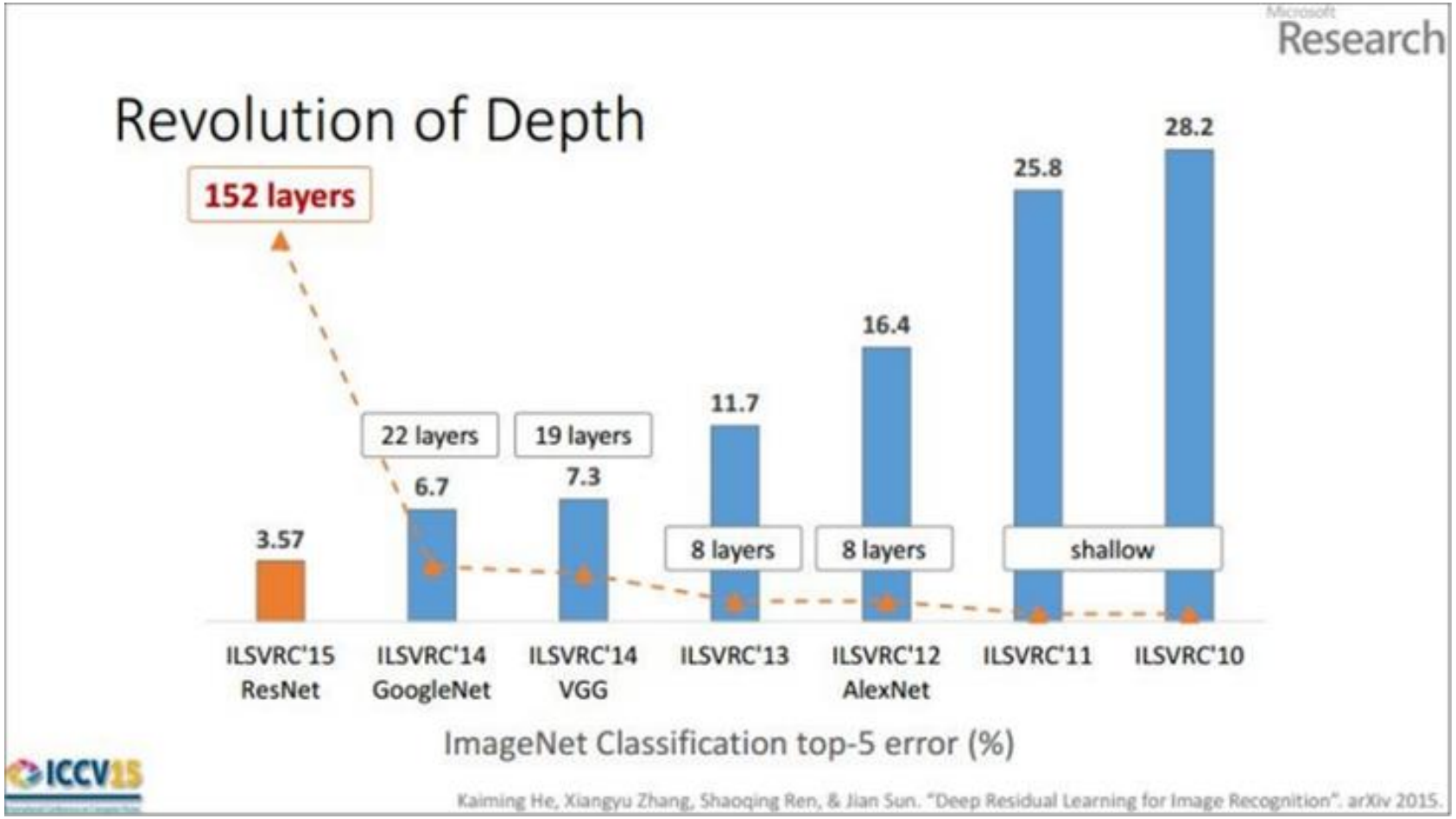
Inception module with dimension reduction

GoogLeNet (2014)

- Architecture details
  - Stack of inception modules
  - Use of auxiliary classifiers
  - No FC layer, use global average pooling



Common trend : Revolution of depth

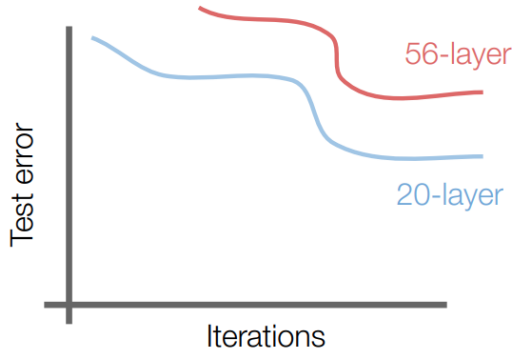


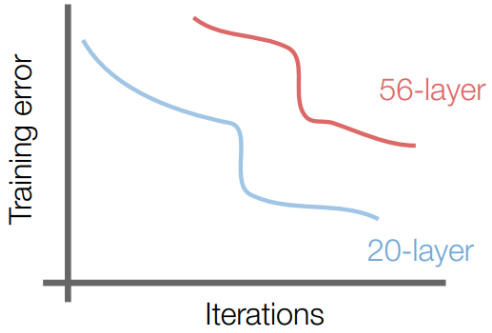
Residual Network (2015)

- Deeper networks with residual connection (152-layers)
- ILSVRC’15 winner

Residual Network (2015)

- Deep network has some problems
  - Vanishing/Exploding gradient
  - Too much parameter

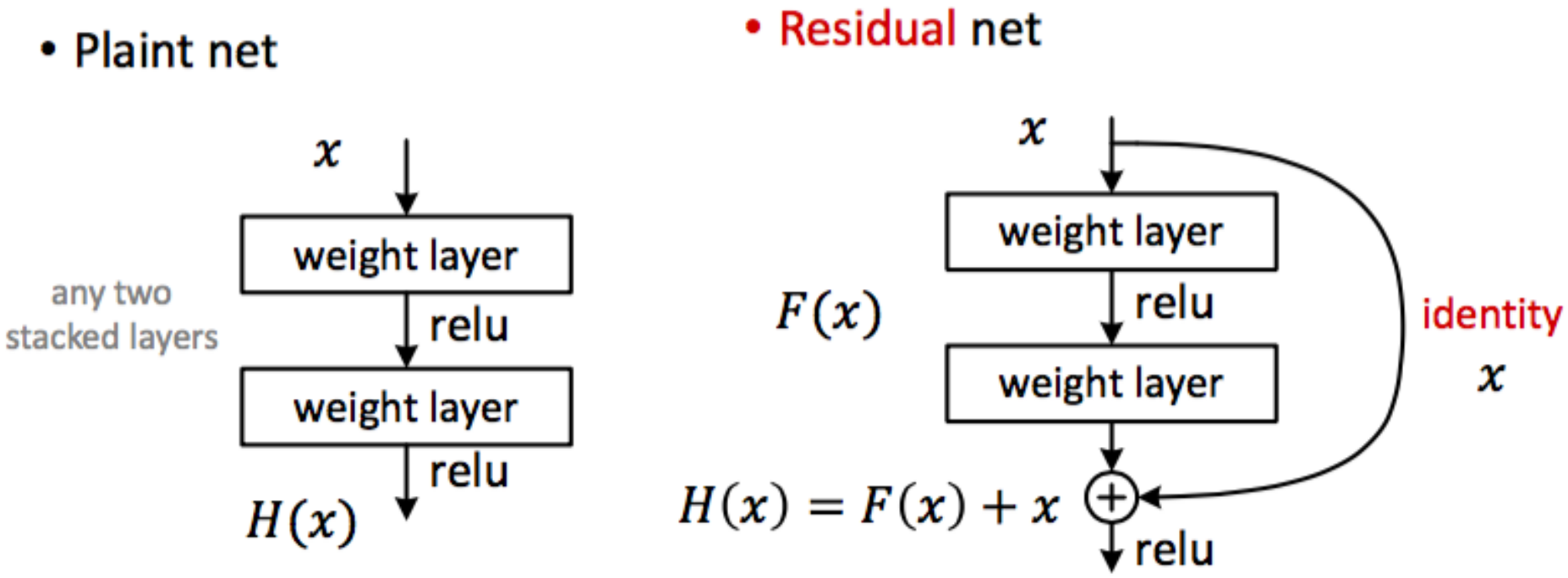




- Residual Network solves these problems

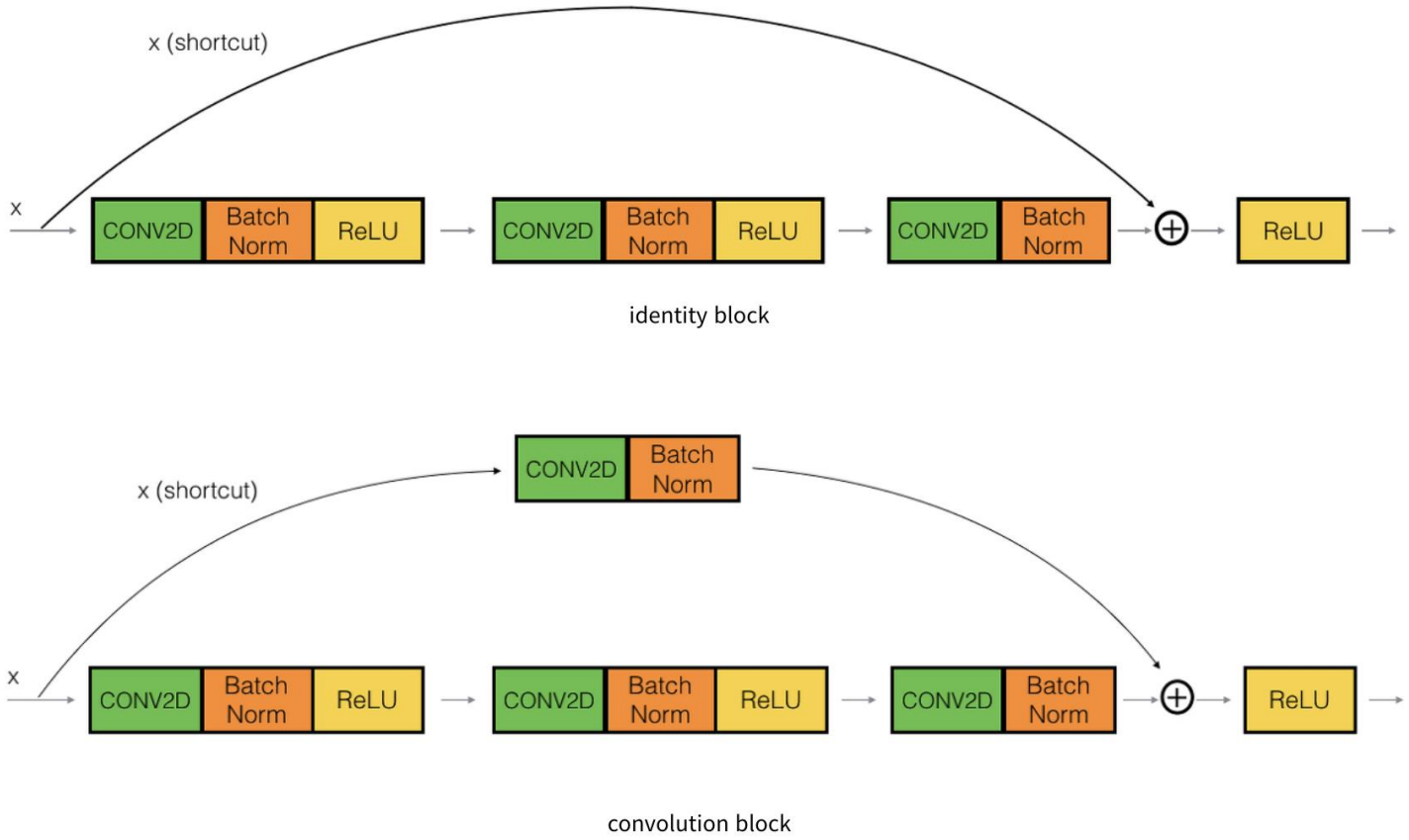
Residual Network (2015)

- Residual Network benefits
  - Easy to learn the residual
  - Solve gradient vanishing problem (미분해도 항상 1 이상)



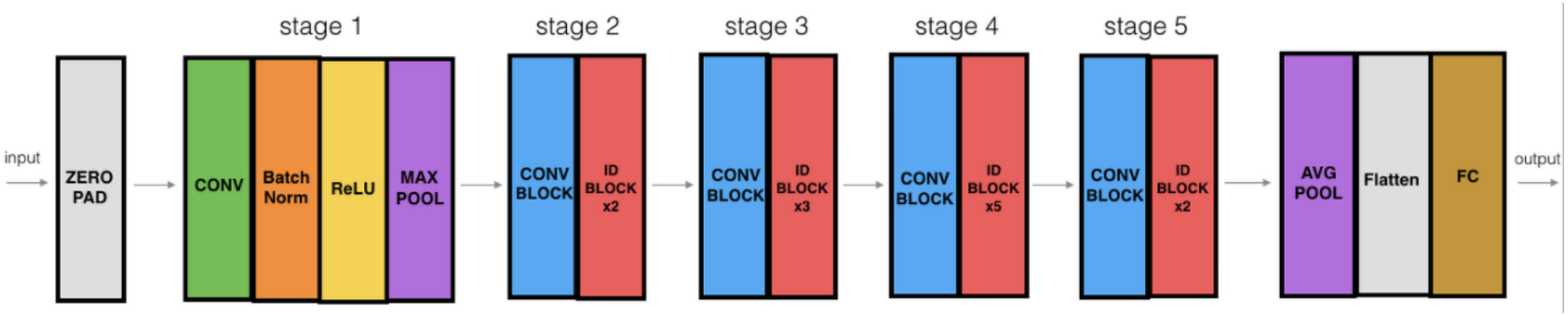
Residual Network (2015)

- Architecture

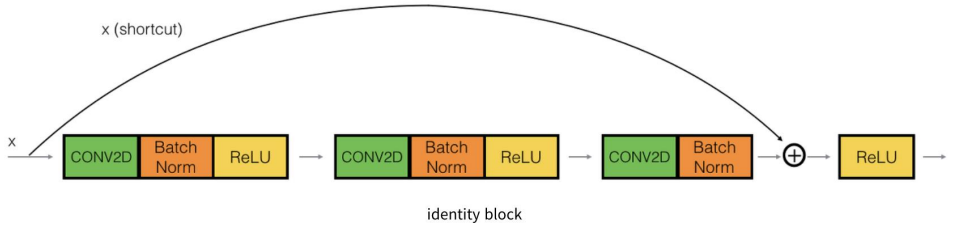


Residual Network (2015)

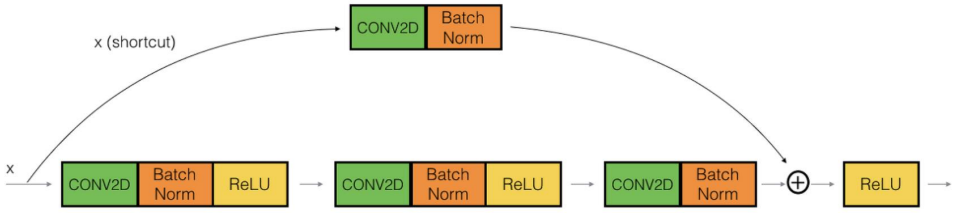
- Architecture



ResNet structure



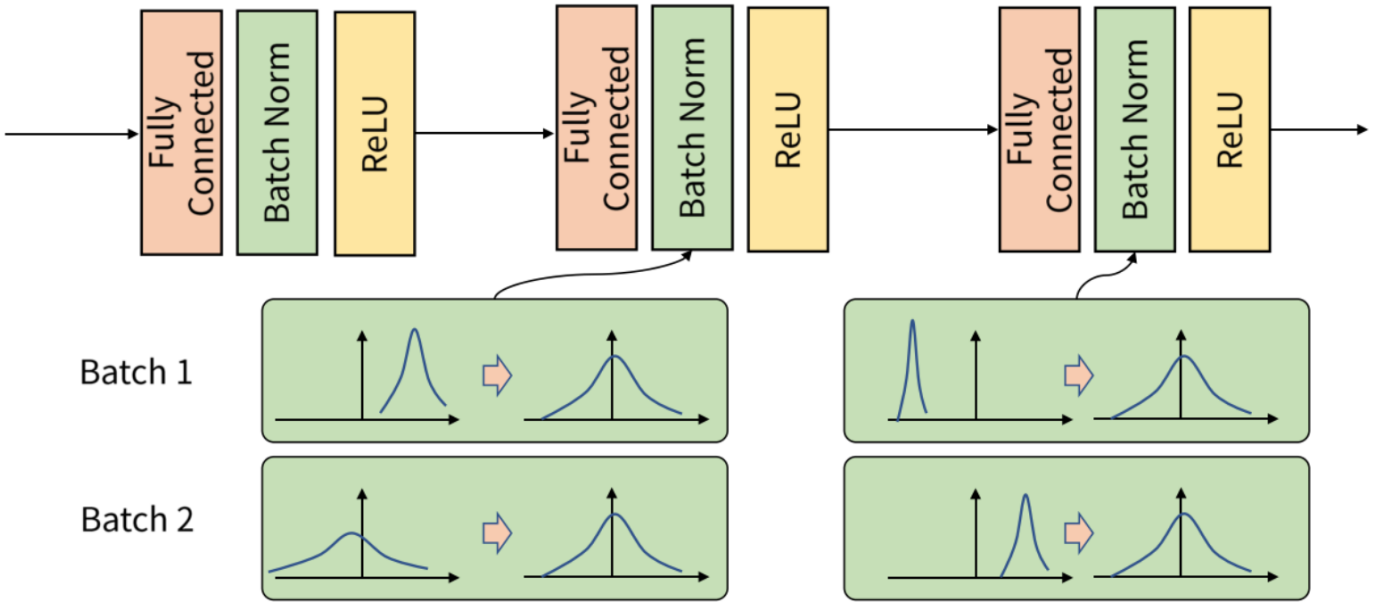
identity block



convolution block

Batch Normalization

- Internal Covariant Shift
  - Normalize each batch using mean and s.d of each batch



Residual Network (2015)

- Architecture

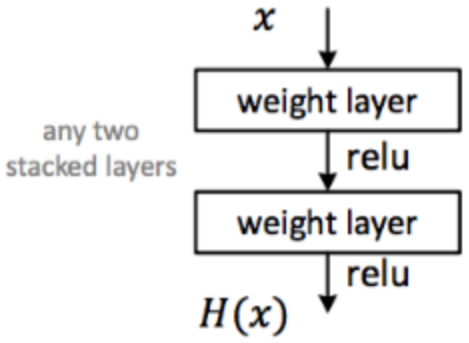
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix}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

Residual Network (2015)

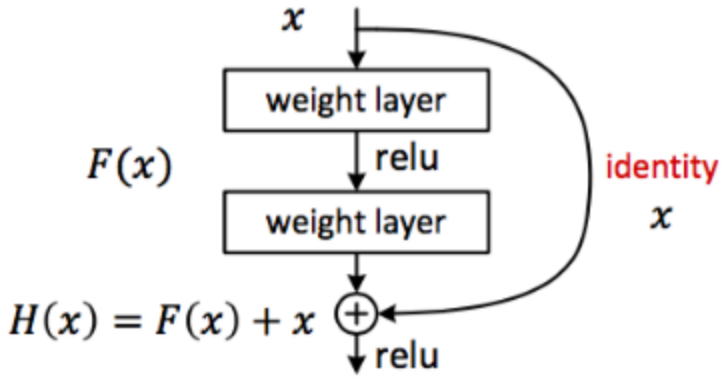
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

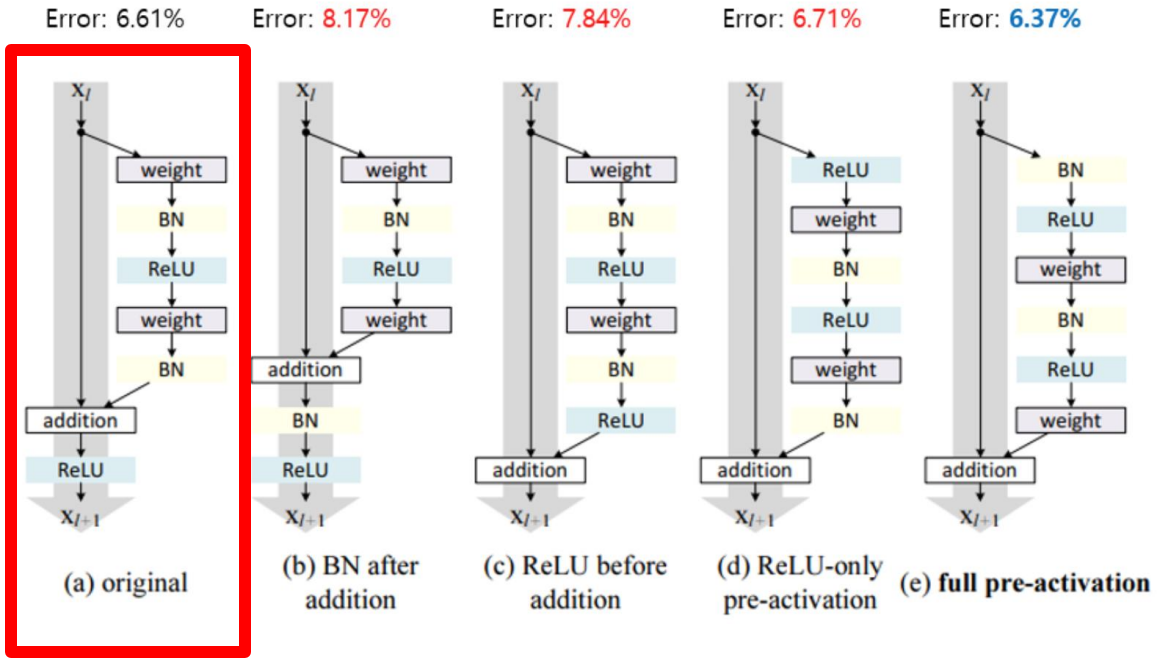
Exercise 3-1. Implement Residual Block

- Plaint net



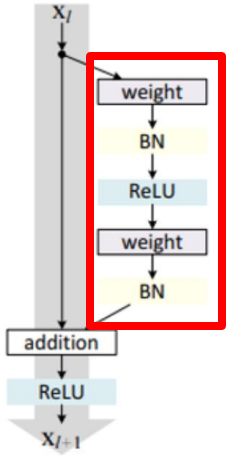
- Residual net





Residual Block

Error: 6.61%



(a) original

Weight = convolution layer

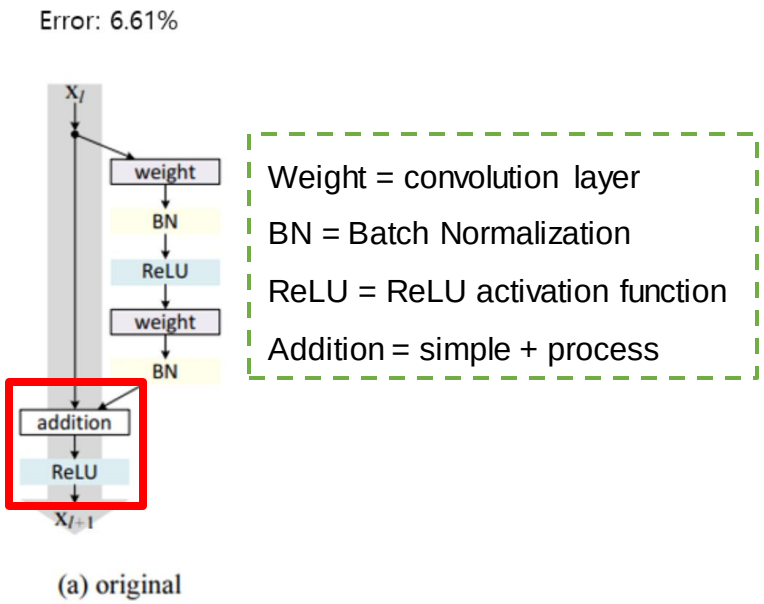
BN = Batch Normalization

ReLU = ReLU activation function

Addition = simple + process

```python
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()

        # BatchNorm에 bias가 포함되어 있으므로, conv2d는 bias=False로 설정합니다.
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        self.shortcut = nn.Sequential()

        self.relu = nn.ReLU()

        # in_channels와 out_channels가 다를 경우 맞춰줌
        if in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        x = self.residual_function(x) + self.shortcut(x)
        x = self.relu(x)
        return x
```

Residual Block



Weight = convolution layer

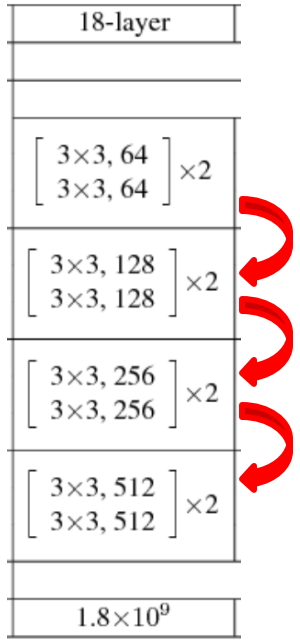
BN = Batch Normalization

ReLU = ReLU activation function

Addition = simple + process

```python
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()

        # BatchNorm에 bias가 포함되어 있으므로, conv2d는 bias=False로 설정합니다.
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        self.shortcut = nn.Sequential()

        self.relu = nn.ReLU()

        # in_channels와 out_channels가 다를 경우 맞춰줌
        if in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        x = self.residual_function(x) + self.shortcut(x)
        x = self.relu(x)
        return x
```

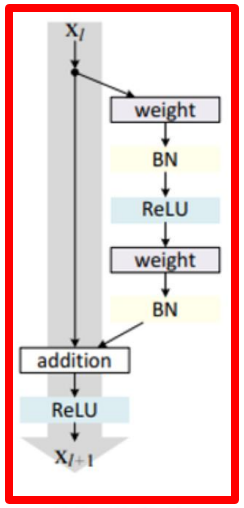
Residual Block




```python
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()

        # BatchNorm에 bias가 포함되어 있으므로, conv2d는 bias=False로 설정합니다.
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        self.shortcut = nn.Sequential()

        self.relu = nn.ReLU()

        # in_channels와 out_channels가 다를 경우 맞춰줌
        if in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        x = self.residual_function(x) + self.shortcut(x)
        x = self.relu(x)
        return x
```

Residual Block

Error: 6.61%



(a) original

Weight = convolution layer

BN = Batch Normalization

ReLU = ReLU activation function

Addition = simple + process

```python
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()

        # BatchNorm에 bias가 포함되어 있으므로, conv2d는 bias=False로 설정합니다.
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        self.shortcut = nn.Sequential()

        self.relu = nn.ReLU()

        # in_channels와 out_channels가 다를 경우 맞춰줌
        if in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        x = self.residual_function(x) + self.shortcut(x)
        x = self.relu(x)
        return x
```

Exercise 3-2. Implement ResNet-18

- Conv – input channels: 3, output channels: 64
- Maxpooling

- Conv – input channels: 64, output channels: 64 x 2
- Conv – input channels: 64, output channels: 64 x 2

- Conv – input channels: 64, output channels: 128
- Conv – input channels: 128, output channels: 128
- Conv – input channels: 128, output channels: 128 x 2

- Conv – input channels: 128, output channels: 256
- Conv – input channels: 256, output channels: 256
- Conv – input channels: 256, output channels: 256 x 2

- Conv – input channels: 256, output channels: 512
- Conv – input channels: 512 ,output channels: 512
- Conv – input channels: 512 ,output channels: 512 x 2
- Avgpooling
- FC

18-layer	34-layer	50-layer	101-layer	152-layer
	7×7, 64, stride 2			
	3×3 max pool, stride 2			
$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	average pool, 1000-d fc, softmax			
1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Every conv layer: kernel size: 3, padding: 1
max pooling layer: kernel size 3, stride 2
Avg pooling layer: nn.AdaptiveAvgPool2d((1,1))

ResNet-18

18-layer	34-layer	50-layer	101-layer	152-layer
	7×7, 64, stride 2			
	3×3 max pool, stride 2			
$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	average pool, 1000-d fc, softmax			
1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```python
class ResNet18(nn.Module):
    def __init__(self, block, num_classes=10):
        super(ResNet18, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding = 3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding = 1)
        )

        self.conv2_1 = block(64, 64)
        self.conv2_2 = block(64, 64)

        self.conv3_1 = block(64, 128)
        self.conv3_2 = block(128, 128)

        self.conv4_1 = block(128, 256)
        self.conv4_2 = block(256, 256)

        self.conv5_1 = block(256, 512)
        self.conv5_2 = block(512, 512)

        self.avg_pool = nn.AdaptiveAvgPool2d((1,1)) # make output h*w = 1*1
        self.fc = nn.Linear(512, num_classes)



    def forward(self,x):
        # x:[batch, 3, 32, 32]

        x = self.conv1(x) # x: [batch, 64, 8, 8]

        x = self.conv2_1(x) # [batch, 64, 8, 8]
        x = self.conv2_2(x) # [batch, 64, 8, 8]

        x = self.conv3_1(x) # [batch, 128, 8, 8]
        x = self.conv3_2(x) # [batch, 128, 8, 8]

        x = self.conv4_1(x)# [batch, 256, 8, 8]
        x = self.conv4_2(x)# [batch, 256, 8, 8]

        x = self.conv5_1(x)# [batch, 512, 8, 8]
        x = self.conv5_2(x)# [batch, 512, 8, 8]

        x = self.avg_pool(x)# [batch, 512, 1, 1]
        x = x.view(x.size(0), -1) # [batch, 512]
        x = self.fc(x) # [batch, 10]

        return x
```

ResNet-18

18-layer	34-layer	50-layer	101-layer	152-layer
7×7, 64, stride 2				
3×3 max pool, stride 2				
$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
average pool, 1000-d fc, softmax				
1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```python
class ResNet18(nn.Module):
    def __init__(self, block, num_classes=10):
        super(ResNet18, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding = 3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding = 1)
        )

        self.conv2_1 = block(64, 64)
        self.conv2_2 = block(64, 64)

        self.conv3_1 = block(64, 128)
        self.conv3_2 = block(128, 128)

        self.conv4_1 = block(128, 256)
        self.conv4_2 = block(256, 256)

        self.conv5_1 = block(256, 512)
        self.conv5_2 = block(512, 512)

        self.avg_pool = nn.AdaptiveAvgPool2d((1,1)) # make output h*w = 1*1
        self.fc = nn.Linear(512, num_classes)


    def forward(self,x):
        # x:[batch, 3, 32, 32]

        x = self.conv1(x) # x: [batch, 64, 8, 8]

        x = self.conv2_1(x) # [batch, 64, 8, 8]
        x = self.conv2_2(x) # [batch, 64, 8, 8]

        x = self.conv3_1(x) # [batch, 128, 8, 8]
        x = self.conv3_2(x) # [batch, 128, 8, 8]

        x = self.conv4_1(x)# [batch, 256, 8, 8]
        x = self.conv4_2(x)# [batch, 256, 8, 8]

        x = self.conv5_1(x)# [batch, 512, 8, 8]
        x = self.conv5_2(x)# [batch, 512, 8, 8]

        x = self.avg_pool(x)# [batch, 512, 1, 1]
        x = x.view(x.size(0), -1) # [batch, 512]
        x = self.fc(x) # [batch, 10]

        return x
```

ResNet-18

18-layer	34-layer	50-layer	101-layer	152-layer
	7×7, 64, stride 2			
	3×3 max pool, stride 2			
$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	average pool, 1000-d fc, softmax			
1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```python
class ResNet18(nn.Module):
    def __init__(self, block, num_classes=10):
        super(ResNet18, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding = 3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding = 1)
        )

        self.conv2_1 = block(64, 64)
        self.conv2_2 = block(64, 64)

        self.conv3_1 = block(64, 128)
        self.conv3_2 = block(128, 128)

        self.conv4_1 = block(128, 256)
        self.conv4_2 = block(256, 256)

        self.conv5_1 = block(256, 512)
        self.conv5_2 = block(512, 512)

        self.avg_pool = nn.AdaptiveAvgPool2d((1,1)) # make output h*w = 1*1
        self.fc = nn.Linear(512, num_classes)
```

```python
def forward(self,x):
    # x:[batch, 3, 32, 32]

    x = self.conv1(x) # x: [batch, 64, 8, 8]

    x = self.conv2_1(x) # [batch, 64, 8, 8]
    x = self.conv2_2(x) # [batch, 64, 8, 8]

    x = self.conv3_1(x) # [batch, 128, 8, 8]
    x = self.conv3_2(x) # [batch, 128, 8, 8]

    x = self.conv4_1(x)# [batch, 256, 8, 8]
    x = self.conv4_2(x)# [batch, 256, 8, 8]

    x = self.conv5_1(x)# [batch, 512, 8, 8]
    x = self.conv5_2(x)# [batch, 512, 8, 8]

    x = self.avg_pool(x)# [batch, 512, 1, 1]
    x = x.view(x.size(0), -1) # [batch, 512]
    x = self.fc(x) # [batch, 10]

    return x
```

ResNet-18

```
!pip install torchsummary
from torchsummary import summary
```

```
net = ResNet18(BasicBlock).cuda()
summary(net, batch_size=-1, input_size=(3, 32, 32), device='cuda')
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 16, 16]           9,408
       BatchNorm2d-2          [-1, 64, 16, 16]             128
              ReLU-3          [-1, 64, 16, 16]               0
         MaxPool2d-4            [-1, 64, 8, 8]               0
            Conv2d-5            [-1, 64, 8, 8]          36,864
       BatchNorm2d-6            [-1, 64, 8, 8]             128
              ReLU-7            [-1, 64, 8, 8]               0
            Conv2d-8            [-1, 64, 8, 8]          36,864
       BatchNorm2d-9            [-1, 64, 8, 8]             128
             ReLU-10            [-1, 64, 8, 8]               0
       BasicBlock-11            [-1, 64, 8, 8]               0
           Conv2d-12            [-1, 64, 8, 8]          36,864
      BatchNorm2d-13            [-1, 64, 8, 8]             128
             ReLU-14            [-1, 64, 8, 8]               0
           Conv2d-15            [-1, 64, 8, 8]          36,864
      BatchNorm2d-16            [-1, 64, 8, 8]             128
             ReLU-17            [-1, 64, 8, 8]               0
       BasicBlock-18            [-1, 64, 8, 8]               0
           Conv2d-19           [-1, 128, 8, 8]          73,728
      BatchNorm2d-20           [-1, 128, 8, 8]             256
             ReLU-21           [-1, 128, 8, 8]               0
           Conv2d-22           [-1, 128, 8, 8]         147,456
...
Forward/backward pass size (MB): 7.85
Params size (MB): 42.65
Estimated Total Size (MB): 50.51
----------------------------------------------------------------
```

18-layer	34-layer	50-layer	101-layer
	7×7, 64, stride 2		
	3×3 max pool, stride 2		
$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$
$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	average pool, 1000-d fc, softmax		
1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9