

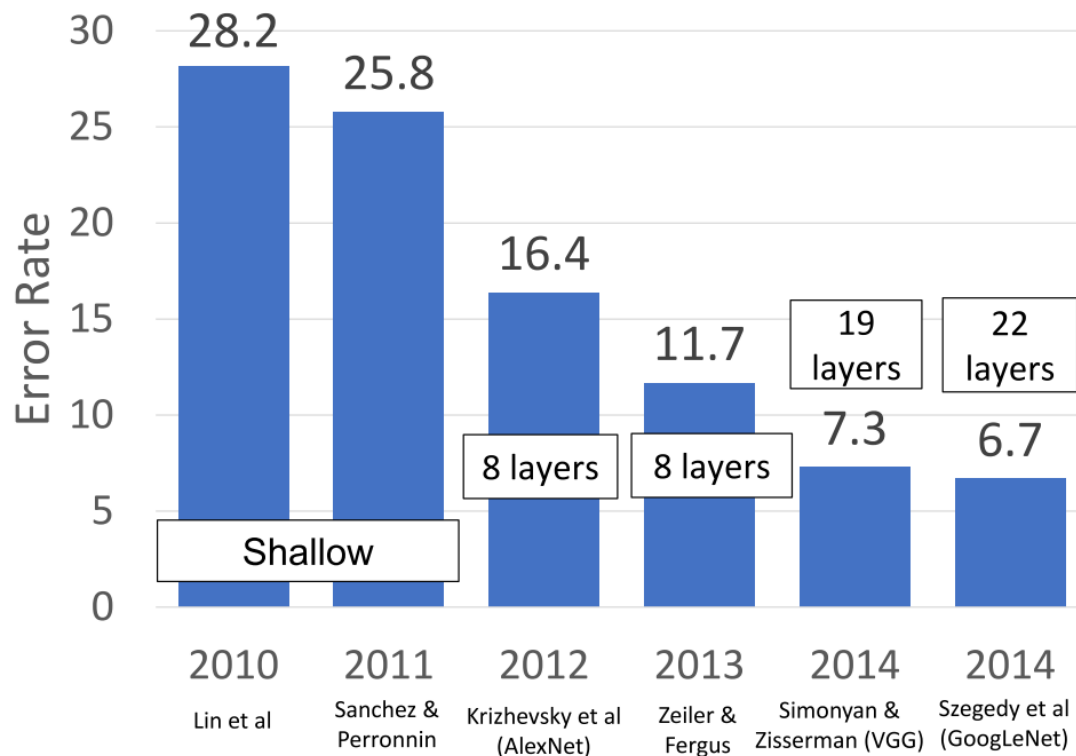
2021 DeepSleep Paper Review

Deep Residual Learning for Image Recognition (ResNet, 2015)

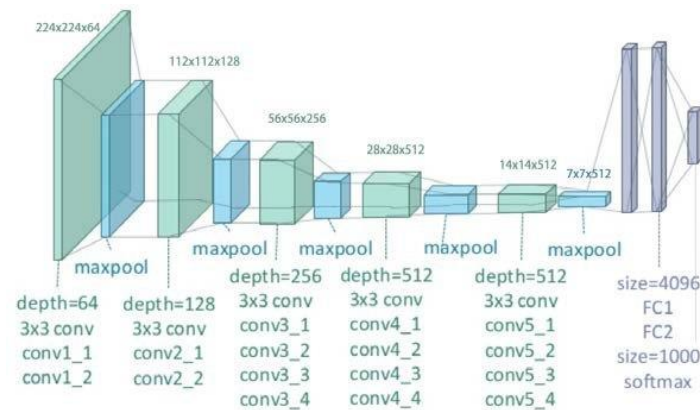
Presenter : Haram Lee

1. 배경

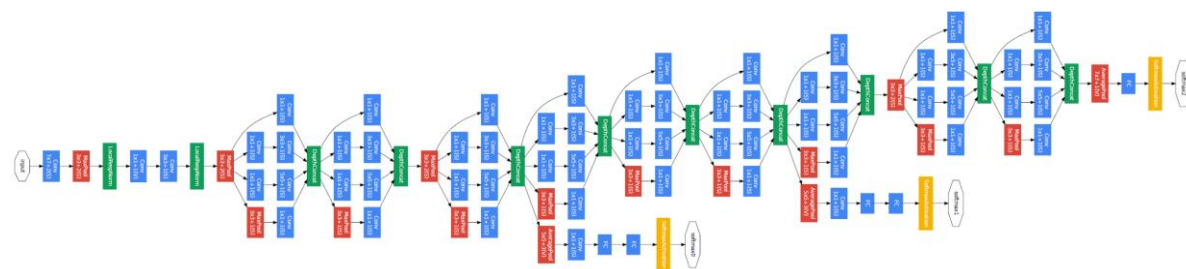
ImageNet Challenge



VGGNet (2014)




GoogLeNet (2015)



➔ VGGNet, GoogLeNet 등을 통해, **네트워크의 깊이**가 매우 중요하다는 사실을 깨달았다.

1. 배경

- 문제 1: vanishing/exploding gradients problem



1. *Vanishing gradient* $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$

2. *Exploding gradient* $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$

layer가 깊어질수록 backpropagation
중에 gradient가 기하급수적으로 빠르게
0이 되거나 무한대가 되는 현상

이미지 출처 : <https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>

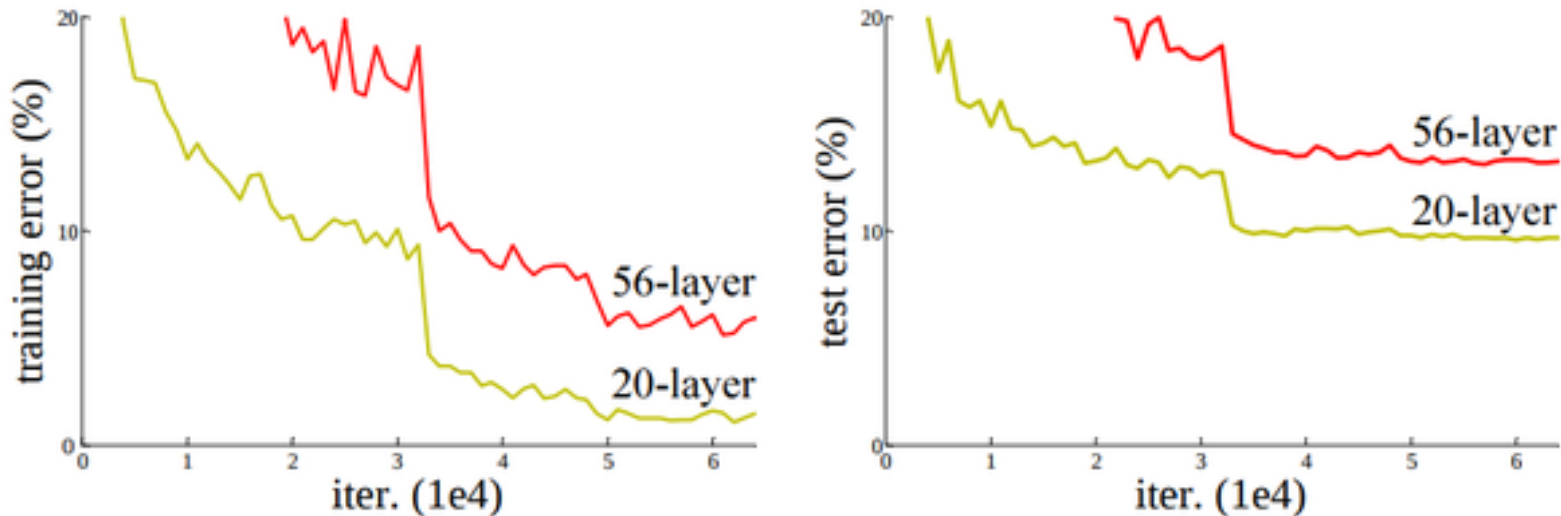
- 해결 방법

1. **kaiming initialization** : 애초에 weight 초기화를 잘하자
2. **batch normalization** : 레이어를 통과하면서 입력의 분포가 변하는 현상을 안정화시키자

➔ 이를 통해, 수십개의 레이어를 가진 네트워크를 수렴시킬 수 있게 되었다.

1. 배경

- 문제 2 : degradation problem

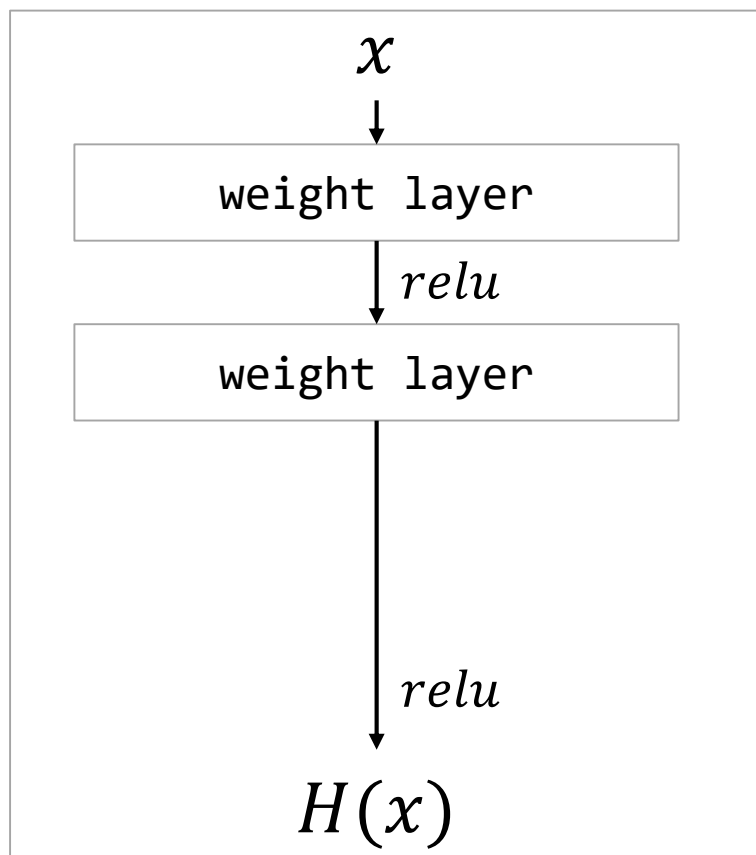


- 적절한 깊이의 모델에 레이어를 추가하였더니 error가 더 커지는 현상
 - 오버피팅 문제가 아님, 최적화가 힘든 것
-
- 해결 방법
 - 더 깊은 모델이 더 얇은 모델보다 더 비슷하거나 더 나아야 한다.
 - 따라서 모델에 **identity mapping** 을 추가한다.

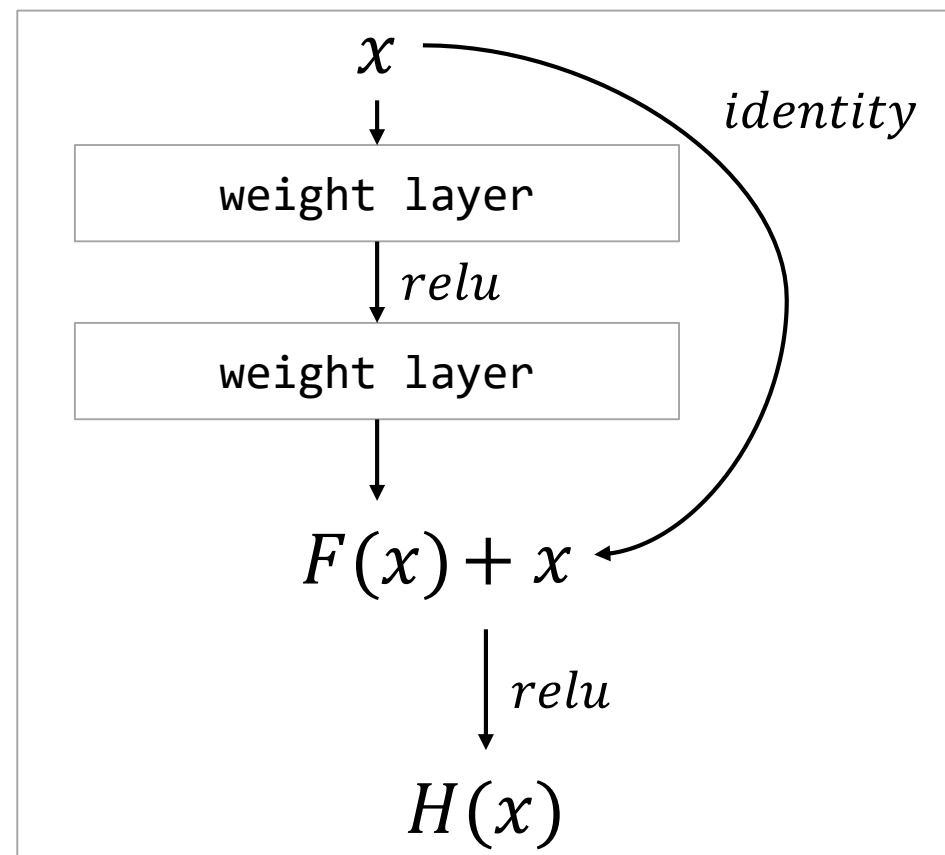
➡ **Deep Residual Learning Framework**

2. 구조

- Residual Learning



- ✓ Residual : 잔여의
- ✓ $F(x) = H(x) - x$



➔ $H(x) = x$ 를 직접 배우는 것보다, $F(x) = 0$ 을 배우는 것이 더 쉽다.

2. 구조

- Identity Mapping by Shortcuts

$$y = F(x, W_i) + W_s x$$

- x : input
- y : output
- $F(x, W_i)$: residual mapping
- $F + x$: shortcut connection and element-wise addition
- W_s : 차원을 맞추기 위해서만 사용

특징

- ✓ shortcut connection에서는 추가 매개 변수나 계산 복잡성이 발생하지 않는다.
- ✓ F 가 하나의 레이어로 구성된다면 그냥 선형 레이어와 유사해지고, 이점이 없어진다.
- ✓ F 는 FC layer 뿐만 아니라, 여러 개의 Conv layers 가 될 수 있다.

2. 구조

- **Network Architectures**

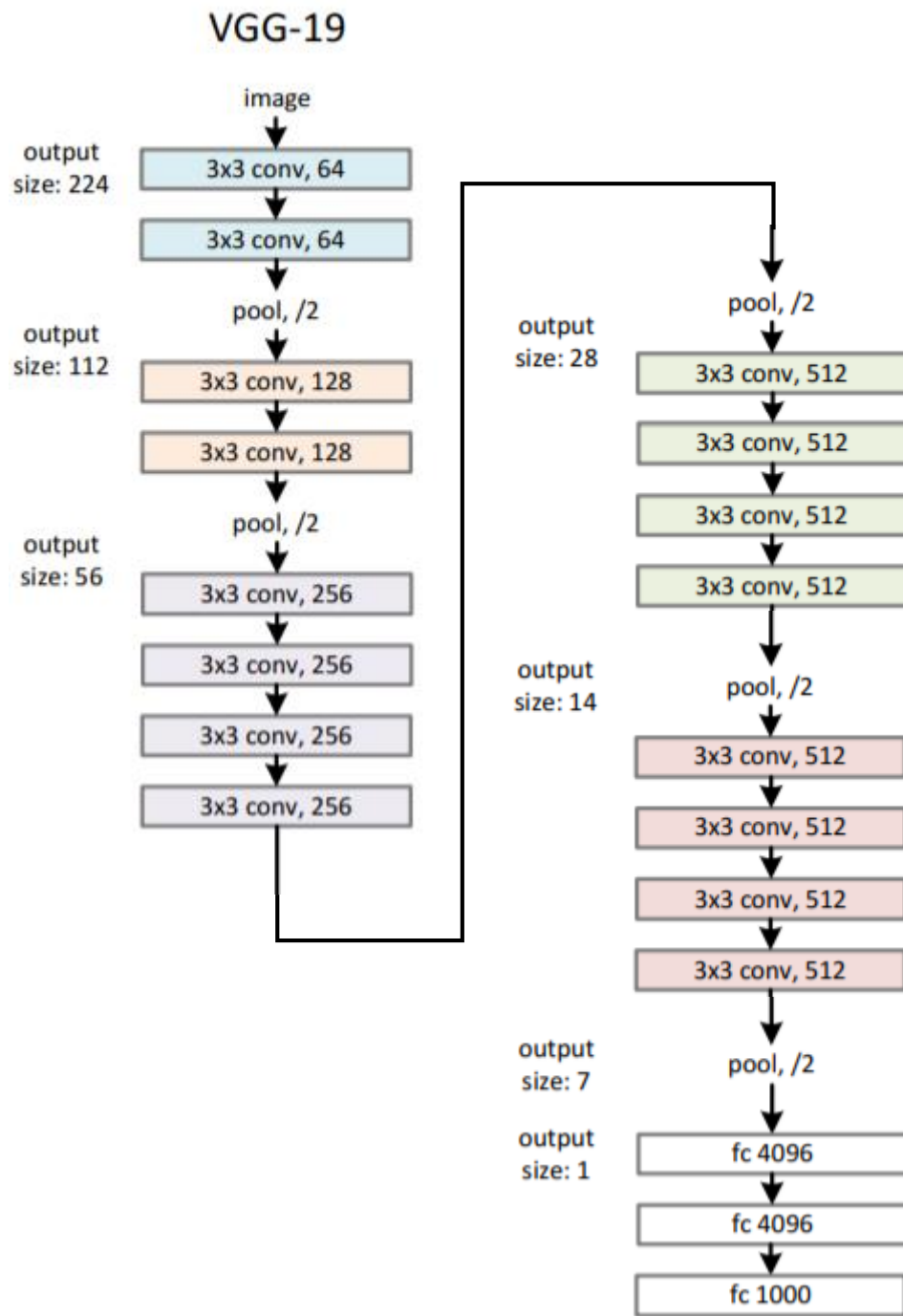
Plain Network

- 주로 VGGNet 의 철학에서 영감을 받아서 구성되었다.
 - 3x3 conv 만 사용한다.
 - 동일한 출력 피쳐 맵 크기에 대해 레이어는 동일한 수의 필터를 갖는다.
 - 피쳐 맵 크기가 절반으로 줄어들면 필터 수가 두 배로 증가하여 계층당 시간 복잡성을 줄인다.

Residual Network

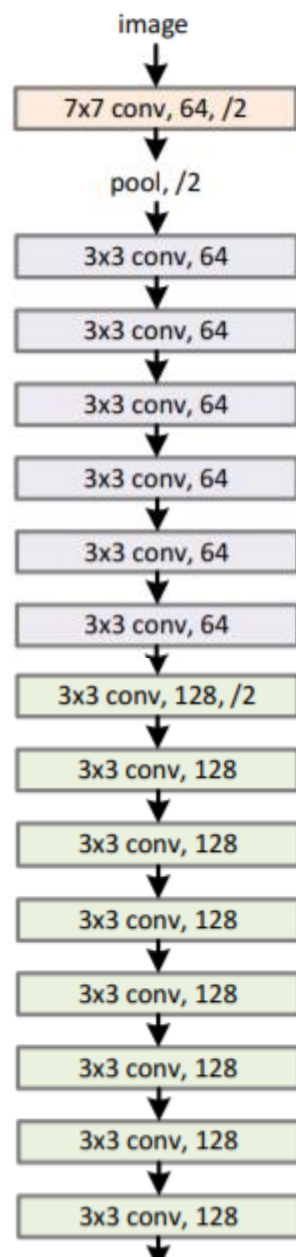
- 위의 Plain Network를 기반으로, shortcut connections를 넣어서 대응되는 residual 버전으로 만든다.
- shortcut
 - dimension 이 같은 경우
 - identity mapping
 - dimension 이 다른 경우
 - A. zero-padding shortcut
 - B. projection shortcut : 1x1 conv 이용해서 dimension 맞춰주기

2. 구조

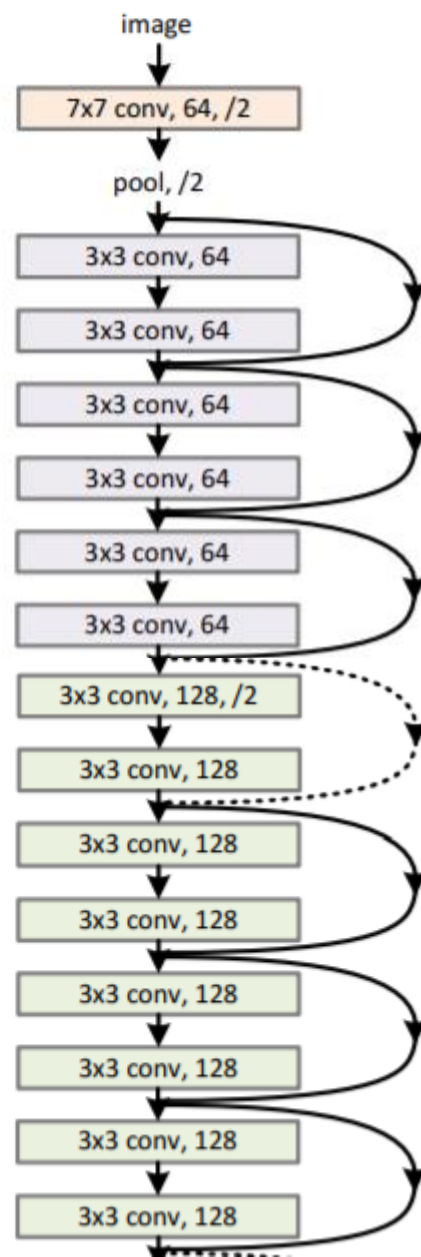


2. 구조

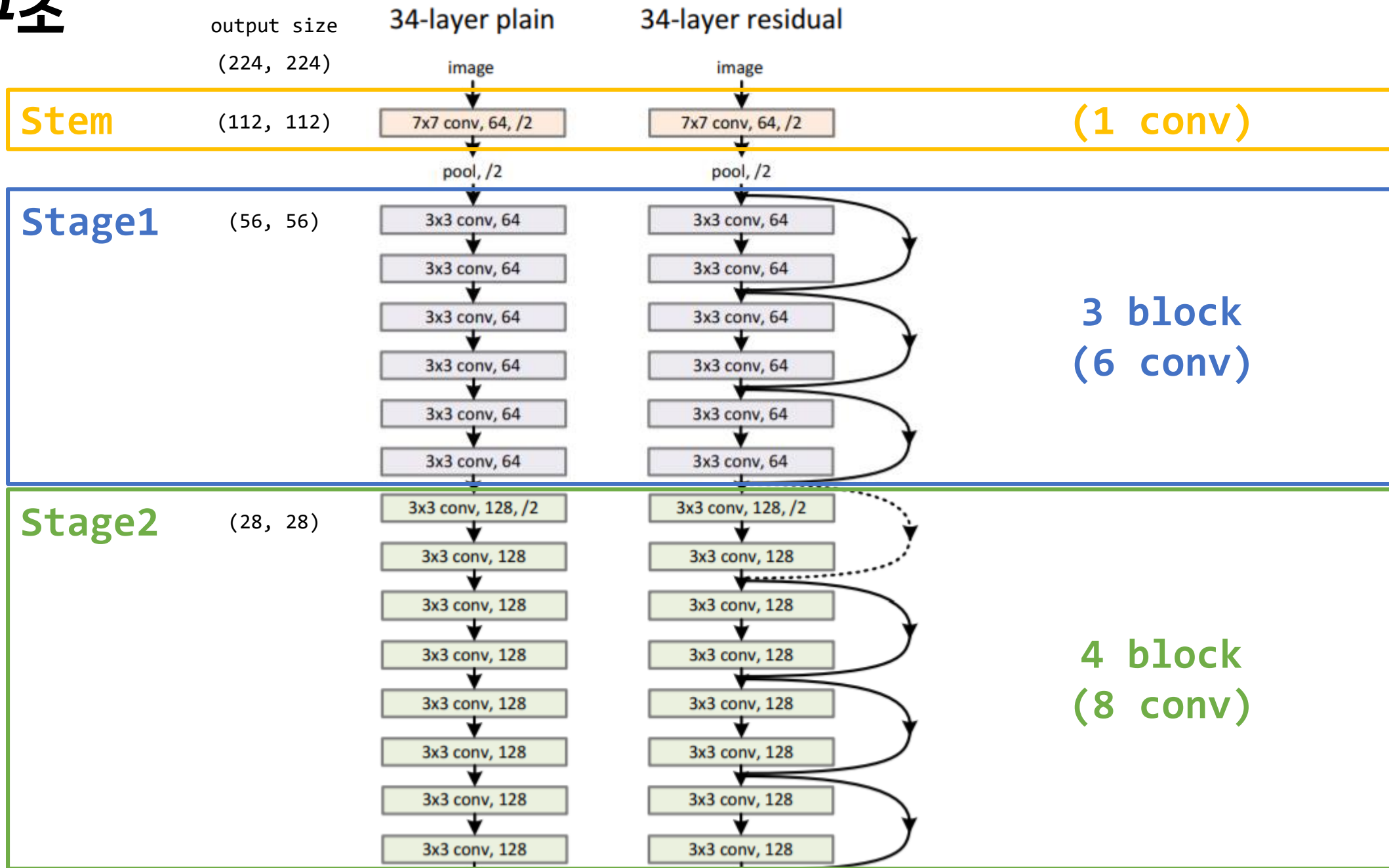
34-layer plain



34-layer residual

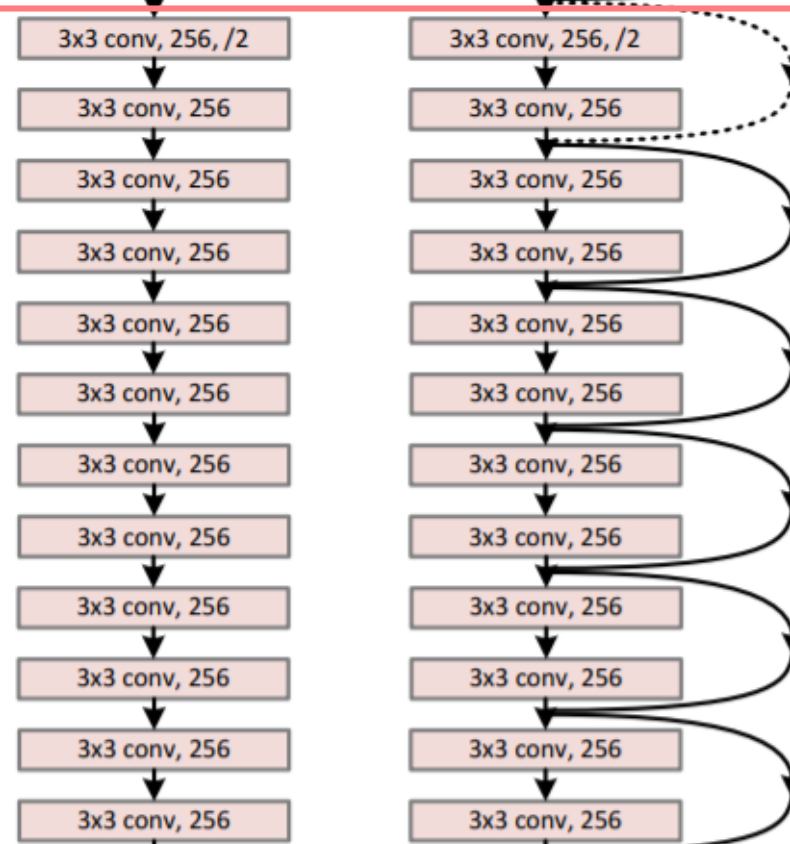


2. 구조



Stage3

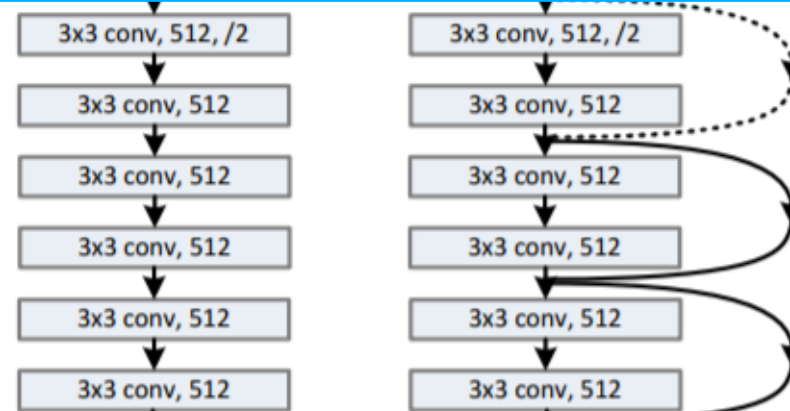
(14, 14)



6 block
(12 conv)

Stage4

(7, 7)



3 block
(6 conv)

avg pool

avg pool

Linear

fc 1000

fc 1000

3. 구현

```
# ===== Plain/Residual Block =====
class Block(nn.Module):
    def __init__(self, in_c=64, double=True, residual=False):
        super(Block, self).__init__()

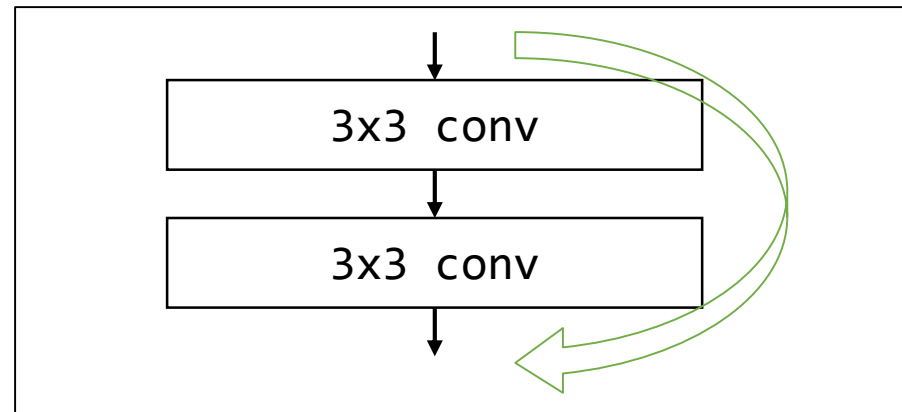
        self.residual = residual
        self.double = double

        out_c = 2*in_c if double else in_c
        stride = 2 if double else 1

        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=in_c, out_channels=out_c, kernel_size=3, stride=stride, padding=1),
            nn.BatchNorm2d(num_features=out_c),
            nn.ReLU()
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=out_c, out_channels=out_c, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(num_features=out_c)
        )
        self.downsample = nn.Sequential(
            nn.Conv2d(in_c, out_c, kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(out_c)
        )
        self.relu = nn.ReLU()

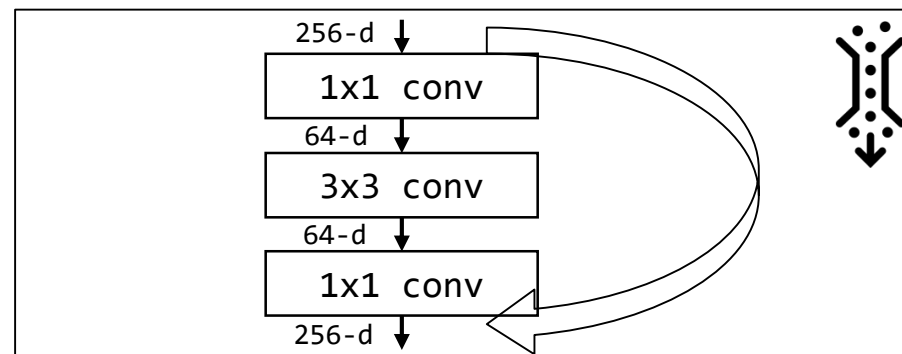
    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        # === The only difference between Plain and Residual ===
        if self.residual:
            if self.double: # match dimension
                x = self.downsample(x) # option (B) projection
            out = out + x
        # =====
        out = self.relu(out)
        return out
```

• Basic Block



- ✓ 매개변수에 따라서 출력 필터 수와 stride를 결정
- ✓ residual이면 shortcut connection 연결

※ (참고) Bottleneck Block



- ✓ ResNet-50 부터는 Bottleneck block 사용

3. 구현

```
# ===== Plain/Residual Stage =====
class Stage(nn.Module):
    def __init__(self, num_blocks=3, in_c=64, double=True, residual=False):
        super(Stage, self).__init__()

        out_c = 2*in_c if double else in_c
        doubles = [double] + [False]*(num_blocks-1)
        channels = [in_c] + [out_c]*(num_blocks-1)

        self.stage = nn.Sequential(OrderedDict([]))
        for i in range(num_blocks):
            self.stage.add_module(f'block{i}', Block(in_c=channels[i], double=doubles[i], residual=residual))

    def forward(self, x):
        return self.stage(x)
```

```
# ===== Plain/Residual Network =====
class Network(nn.Module):
    def __init__(self, residual=False):
        super(Network, self).__init__()
        # input : (3, 224, 224) (ignore batch size here)
        self.stem = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=7, stride=2, padding=2), # out : (64, 112, 112)
            nn.MaxPool2d(kernel_size=2, stride=2, padding=1), # out : (64, 56, 56)
            nn.BatchNorm2d(num_features=64),
            nn.ReLU()
        )
        self.stage1 = Stage(num_blocks=3, in_c=64, double=False, residual=residual) # out : (64, 56, 56)
        self.stage2 = Stage(num_blocks=4, in_c=64, double=True, residual=residual) # out : (128, 28, 28)
        self.stage3 = Stage(num_blocks=6, in_c=128, double=True, residual=residual) # out : (256, 14, 14)
        self.stage4 = Stage(num_blocks=3, in_c=256, double=True, residual=residual) # out : (512, 7, 7)
        self.avgpool = nn.AvgPool2d(kernel_size=7) # out : (512, 1, 1)
        self.fc = nn.Linear(in_features=512, out_features=1000) # out : (1000)
```

```
def forward(self, x):
    N = x.shape[0] # batch size
    x = self.stem(x)
    x = self.stage1(x)
    x = self.stage2(x)
    x = self.stage3(x)
    x = self.stage4(x)
    x = self.avgpool(x)
    x = self.fc(x.reshape(N, -1))
    return x
```

- Stage

- ✓ block 개수에 따라서 모듈 쌓기

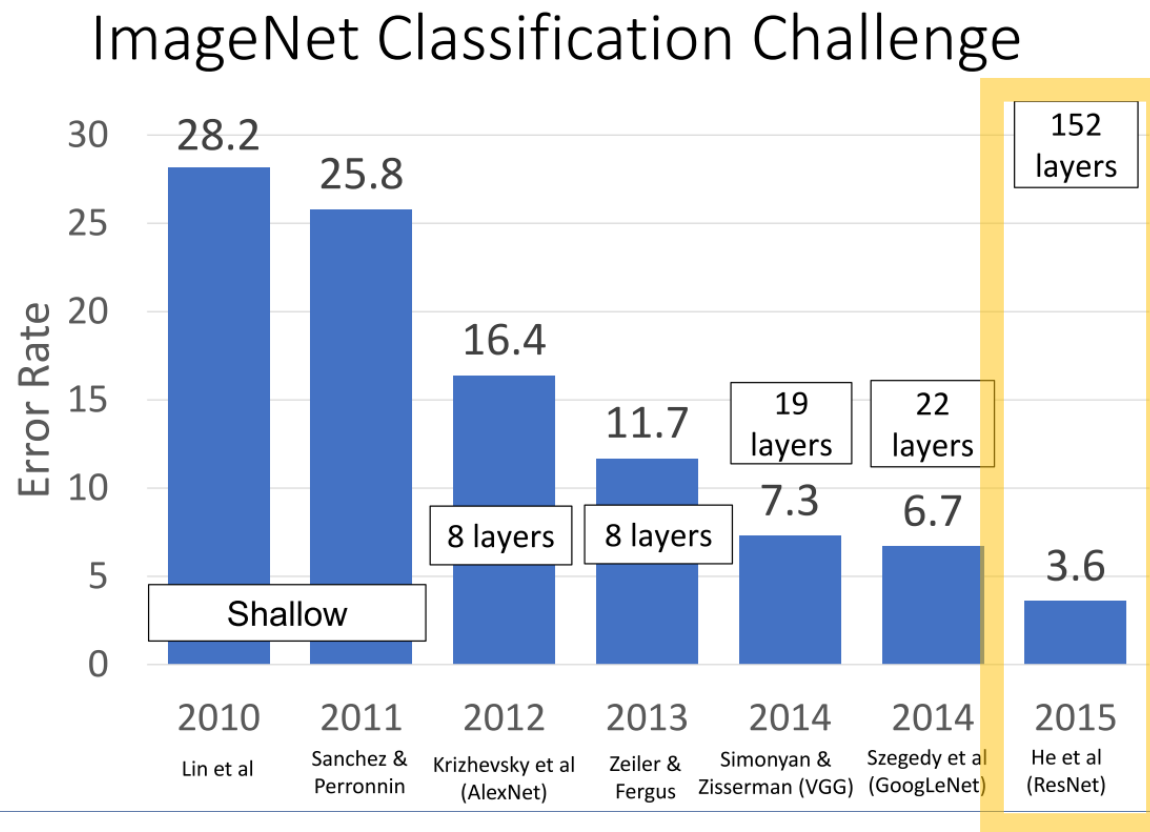
- Network

- ✓ 각 단계를 연결하여 계산하여 출력

4. 결과

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	25.03

Table 2. Top-1 error (% , 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.



Q & A