# Resnet Paper Review

## 한줄 요약

Resnet은 **identical mapping을 하는 layer를 삽입**해서, **layer의 개수는 늘리되, vanishing gradient는 방지**한다.

---

## Why Resnet?

일반적으로, layer가 깊어지면 깊어질수록 Error는 줄어들어야 한다.

Layer가 많다는 것은 그만큼 연산과정이 더 많고, 더 많이 고려를 했다는 것이기에 줄어들어야 한다. 하지만, 기존의 CNN은 layer가 일정수준 이상으로 많아지게 되면 error rate가 높아지는 현상을 보인다.

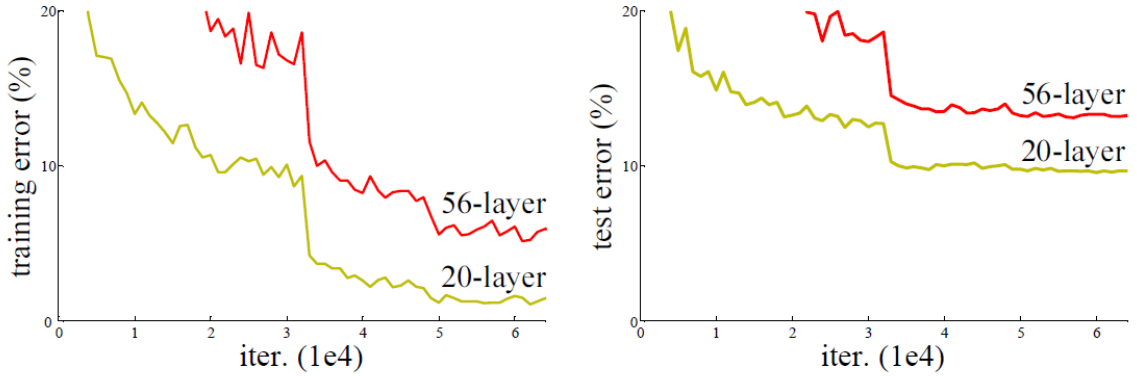



Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

위 그래프는 CIFAR-10 데이터에 대해서  CNN을 구축해서 Training시킨 것이다.

위 그래프를 확인하면 알 수 있듯이 20개 layer를 사용했을 때가, 56개 layer를 사용했을 때보다 더 정확도가 높고 error rate가 낮은 것을 확인할 수 있다.

### what cause the error rate?

1. vanishing/exploding gradients 이 이런 현상이 나타나는 원인이다.

이는 역전파과정에서 gradient가 사라지는 것이다. layer의 수가 더 많아지다보니 역전파가 제대로 이뤄지지 않아서 error rate가 상승한다. 그러나 이 경우에는 **normalized initialization and intermediate normalization layers** 을 이용함으로써 해결을 하였다.

하지만 이를 해결하고 나서도, 여전히 성능저하의 문제가 발생하였다.

2. 하지만, 이는 overfitting 같은 것이 아니라 적절하게 layer를 추가했음에도 위와 같이 현상이 나타나는 것을 "실험적으로" 확인하였다. **이를 degradation problem이라 한다.**

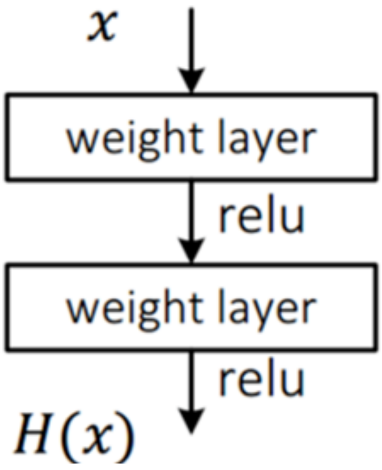
---

# What is Resnet?

**Resnet(deep residual lerning framework)**은 위에서 언급한 degradation problem을 해결하기 위해 만든 것이다.

## Plain CNN

기존의 CNN의 경우에는 아래와 같이 생겼다. layer를 들어가기 전에 input을 x라 하고, layer를 거친 다음 output을 H(x)라 하면 아래와 같이 표현할 수 있다. 이때, H(x)는 target output이다.




이 과정을 거친 다음에 우리는 $H(x)\ ?=y_l$ 을 확인하였고, $H(x)\ -y_l$ 을 최소로 만들고자 하였다.

## Resnet

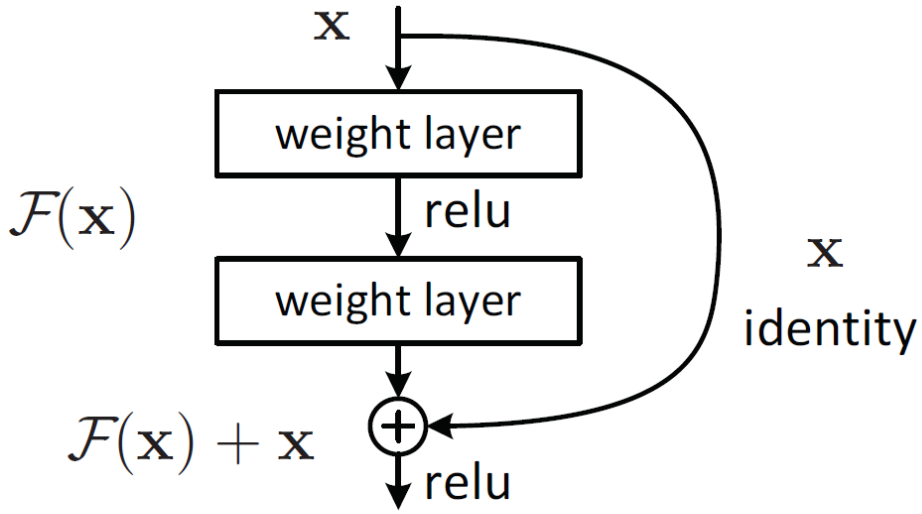



Figure 2. Residual learning: a building block.

위 사진은 Resnet의 한 블럭이다. 이는 identity mapping과 skip connection 을 이용하게 된다. 즉, x와 H(x)가 같게 만드는 mapping이다. skip connection은 shortcut connection이라고 논문에서 사용하고 있지만, layer를 뛰어넘어서 feature를 사용하는 것을 말한다. 여기서는 identity x가 layer를 넘어서 계속해서 사용한다는 점에서 skip connection이 언급된다.

Residual learning의 가장 중요한 점은 input x와 H(x) 사이의 차이를 학습시킨다는 것이다.

$$
\begin{gathered}
F(x): \text{output value of layer} \\
\text{Want to Learn F(x) - x to x} \\
So, F(x) + x \sim= x \\
then\ F(x) \sim= 0
\end{gathered}
$$

이렇게 학습을 시키게 되면, 다음과 같이 output값이 변화하게 된다.

$$
\begin{aligned}
y_l &= x_l + F(x_l, W_l). \\
x_{l+1} &= y_l = x_l + F(x_l, W_l). \\
x_L &= x_l + \sum_{i=l}^{L-1} F(x_i, W_i)
\end{aligned}
$$

결국에는 $H(x) - x$를 0으로 가까워지게 layer를 만들고 training시키는 것이다. 그래서 이 이름이 residual learning이다. 이처럼 하게되면, 역전파 과정에서 gradient가 소실되는 것을 방지할 수 있다.

# Proc of Resnet

## vanishing/exploding gradient의 방지

### Plain CNN

기존의 CNN에서는 Layer를 늘릴 때, Gradient가 다음과 같았다.

$x_l \rightarrow x_{l+1} \rightarrow x_{l+2} \rightarrow x_{l+3} \rightarrow x_{l+4} \rightarrow \cdots \rightarrow x_L$

위처럼 layer가 있다 하자.

$$x_L = x_l + \triangledown loss$$

$$\frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L} * \frac{\partial x_L}{\partial x_{L-1}} * \frac{\partial x_{L-1}}{\partial x_{L-2}} * \cdots * \frac{\partial x_{L-n}}{\partial x_l}$$

그러면 위처럼 chain rule이 되게 되는 데, 각각의 gradient에 대해서 1이상이거나 0이하 이게 되면 쌓이면서 총합 gradient가 굉장히 커지거나 작아지거나 하게 된다. 이는 CNN 성능 저하에 직결된다.

### Resnet

하지만, Resnet에서는 식이 다음과 같게 된다.

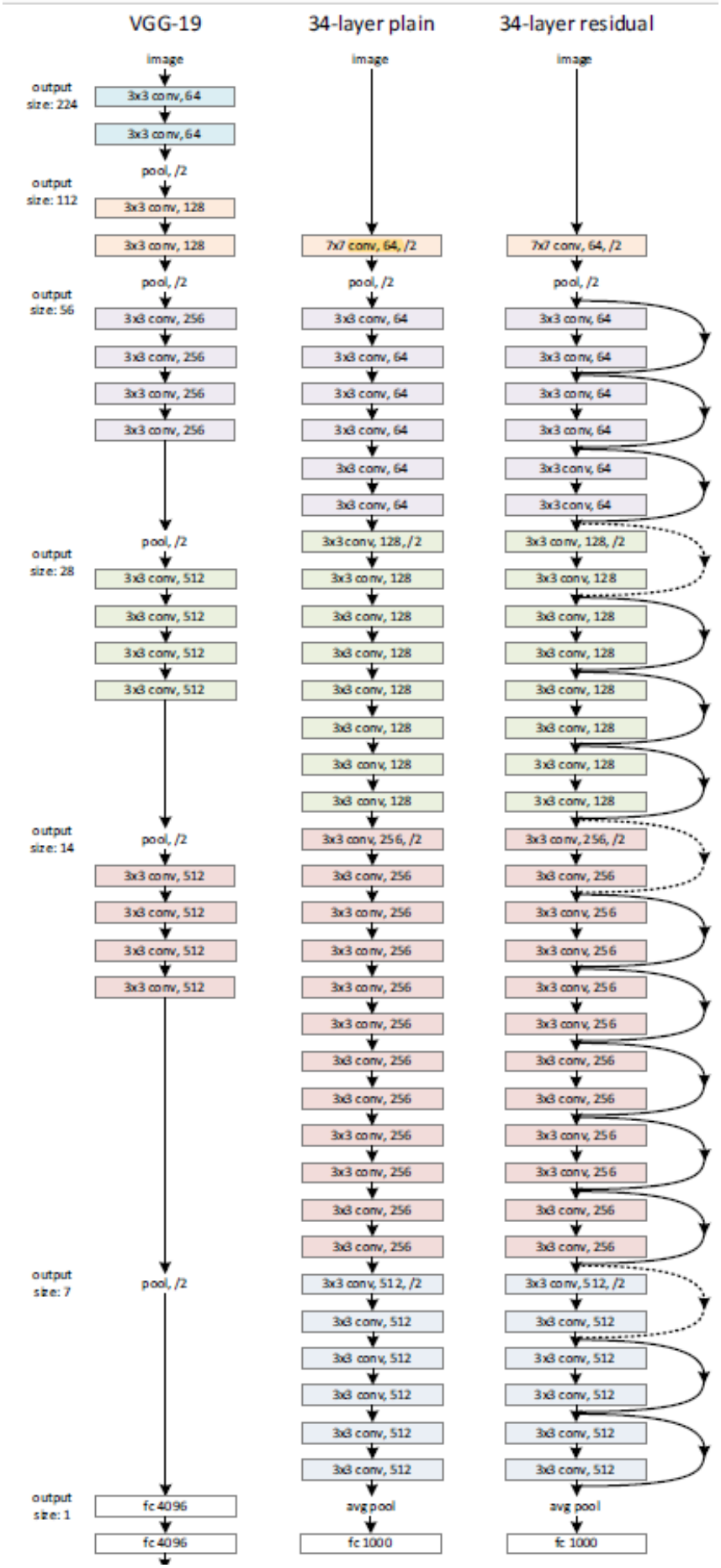
$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$$

$$\frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L} * \frac{\partial x_L}{\partial x_l} = \frac{\partial loss}{\partial x_L}\left(1 + \frac{\partial \sum_{i=l}^{L-1} F(x_i, W_i)}{\partial x_l}\right)$$

위를 확인하게 되면, gradient가 최소 1은 확보가 된것을 확인할 수 있다. 또한, $\sum$부분이 0에 가깝기에 gradient가 vanishing/exploding 하는 것을 방지할 수 있다.

---

# What is Residual learning?


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
output size: 1
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 256, /2
3x3 conv, 512, /2
avg pool
fc 1000

VGG model과 Plain model, Renet model을 비교하는 사진이다.

그 중에서도 Plain 모델과 Resnet medel에 대해서 비교를 해보자.

**Plain모델은 다음과 같은 방법으로 설계를 하였다.**

- 모든 convolution filter는 $3 \times 3$ 사이즈이다.
- output size가 같은 layer들은 모두 같은 수의 convolution filter의 수를 사용
- output size가 줄어들면 filter의 수 반비례하게 증가. (여기서는 2배)
- output size를 줄일 때는 pooling 대신에 filter의 stride를 2로 하여 사용
- FC layer를 생성할 때, GAP(Global Average Pooling)을 이용했다.

**Plain 모델에서는 모든 Layer를 그대로 연결하고 있고, Resnet에서는 Plain model에 Shortcut connection을 적용시킨 버전이다.**

이때, Resnet Network에서 주의해야하는 부분은 위 사진에서 점선으로 표시되어 있는 부분이다.

점선으로 되어있는 부분은 zero padding을 해주는 경우도 있고, $1 \times 1$ convolution filter로 증가시키기도 한다.

---

# Implementation

논문참고.

실험을 하였던 환경에 대해 서술하고 있음.

---

# Is Resnet really good?(Experiments)

## Image Classification

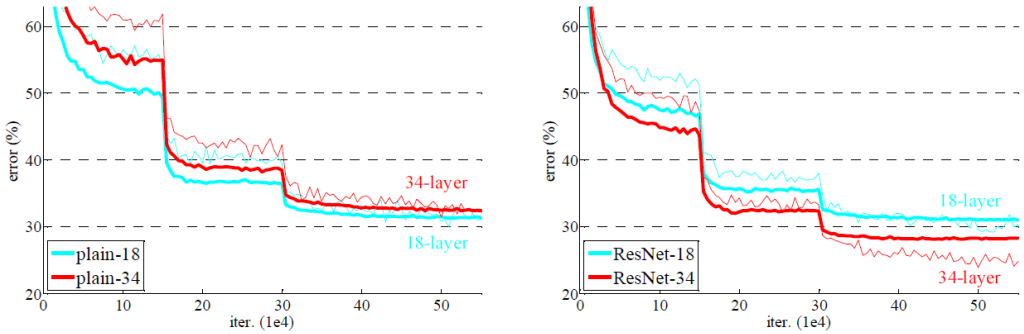



Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

왼쪽의 그래프와 오른쪽 그래프를 비교했을 때, 확실히 Resnet을 이용하면 더 Deep 하게 Network를 생성할 수 있었다. 위 실험을 통해서 확인할 수 있었던 것은 아래와 같다.

- Resnet을 이용하게 되면, Degradation Problem (성능저하)를 해결할 수 있다.
- More layer, More Accurancy
- Plain보다 최고 성능에 더 먼저 도달한다.

## Identity vs Projection Shortcuts

### A (zero-padding shortcuts) vs B (project shortcuts) vs C (shortcut할 때 projection)

아래 결과를 보면 알 수 있듯이 A,B,C 모두 근소한 차이밖에 나지 않았다. 그래서 별로 상관없다. (하지만, parameter의 개수를 고려할 때, A까 제일 낫지 않을 까 싶다.)

| model | top-1 err. | top-5 err. |
|---|---|---|
| VGG-16 [41] | 28.07 | 9.33 |
| GoogLeNet [44] | - | 9.15 |
| PReLU-net [13] | 24.27 | 7.38 |
| plain-34 | 28.54 | 10.02 |
| ResNet-34 A | 25.03 | 7.76 |
| ResNet-34 B | 24.52 | 7.46 |
| ResNet-34 C | 24.19 | 7.40 |
| ResNet-50 | 22.85 | 6.71 |
| ResNet-101 | 21.75 | 6.05 |
| ResNet-152 | **21.43** | **5.71** |

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

## Deeper Bottlenect Architecture

Traning Time을 위해서 block을 bottlenect design을 이용한다. bottlenect design은 아래 사진에서 오른쪽에 해당한다.

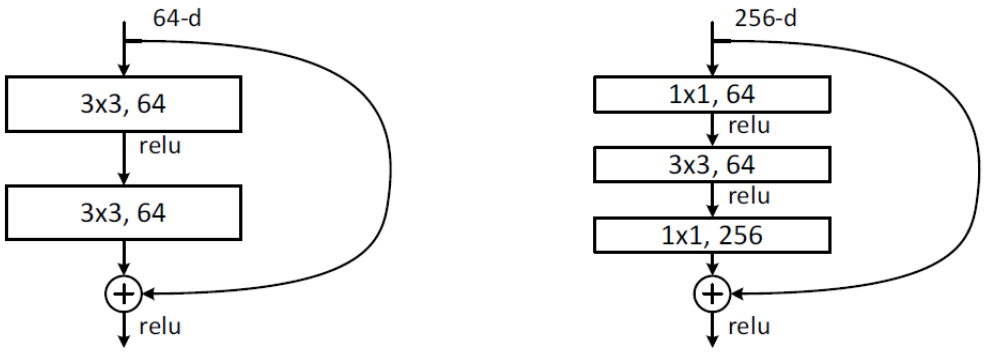



Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

이는 실제 실험과정 중에서도 확인할 수 있다.

| 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 | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

50-layer 이상부터는 bottlenect design을 이용하는 것을 확인할 수 있다. (입력사이즈가 64에서 256으로 변화하는 것은 zero padding 일 것이라 추측함)

## Performance

그렇게 했을 때 성능은 다음과 같다.

top-5 error가 4.49% (single-model) , 3.57%(ensenbles) 까지 줄였음.

| method | top-1 err. | top-5 err. |
|---|---|---|
| VGG [41] (ILSVRC'14) | - | 8.43† |
| GoogLeNet [44] (ILSVRC'14) | - | 7.89 |
| VGG [41] (v5) | 24.4 | 7.1 |
| PReLU-net [13] | 21.59 | 5.71 |
| BN-inception [16] | 21.99 | 5.81 |
| ResNet-34 B | 21.84 | 5.71 |
| ResNet-34 C | 21.53 | 5.60 |
| ResNet-50 | 20.74 | 5.25 |
| ResNet-101 | 19.87 | 4.60 |
| ResNet-152 | **19.38** | **4.49** |

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

| 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** |

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

## Reference


https://s3-us-west-2.amazonaws.com/secure.notion-static.com/96b5f9e7-f4ac-4c66-928b-629b457382dd/1512.03385.pdf

https://89douner.tistory.com/64

https://m.blog.naver.com/laonple/221259295035

https://ganghee-lee.tistory.com/41