

---

---

---

---

---



# Deep residual learning for image recognition.

○ 2015 ILSVRC 1st.  
○ 2016 Kaiming He, ...

- Layer의 input을 reference(참조)  $\rightarrow$  referenced function  
reference하지 않은 layer  $\rightarrow$  unreferenced function.

- The depth of representation

$\rightarrow$  network의 각 layer들을 거친 결과.

$\rightarrow$  보다 많은 layer로 부터 추출된 고차원의 feature들로 양질의 정보를 얻는다.

- Is learning better networks as easy as stacking more layers?

Problem  $\rightarrow$  notorious problem of vanishing/exploding gradients.  
which hamper convergence from beginning.

Solve  $\rightarrow$  'normalized initialization', 'intermediate normalization'  
(= Xavier/He initialization) (= batch normalization)

- However, when deeper networks are able to start converging,  
a degradation problem has been exposed.

(드레스다)

= (which the network depth increasing, accuracy gets saturated)  
and then degrades rapidly. (跌倒)

$\rightarrow$  not caused by overfitting

$\rightarrow$  adding more layers to a suitably deep learning model leads to  
high training error.  
(= architecture)

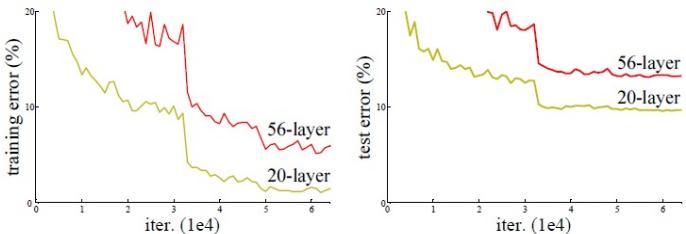


Fig 1. CIFAR-10 dataset with 'plain' networks.

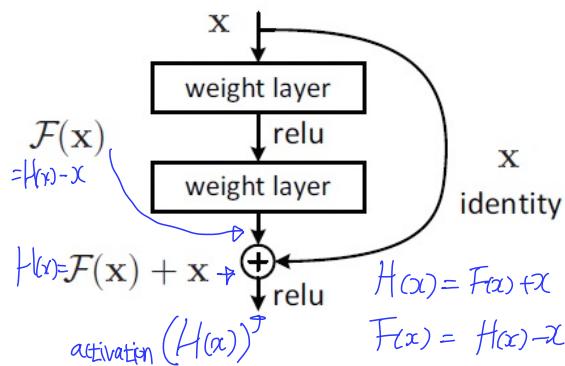


Fig 2. Residual learning: a building block

(입력  $x$ 가 2개의 stack layer를 거친 결과 (Plain 이전)와

identity인 입력  $x$ 를 더한 후에 nonlinearity layer (ReLU)를 통과.)

- In this paper, we address the degradation problem by introducing a deep residual learning framework.

$\rightarrow$  few stacked layer은 underlying mapping을 직접 학습하지 않고, residual mapping에

$\rightarrow$   $\approx$  underlying mapping, 원래의 original mapping은  $H(x)$ 로 나타내면 학습

Stacked nonlinear layer의 mapping인  $F(x)$ 는  $H(x) - x$ 를 학습

$\rightarrow$  따라서 original mapping이  $F(x) + x$ 로 재구성된 걸 볼 수 있음

$\rightarrow$  쌓여 있는 레이어가 (stacked layer) underlying mapping을 fit 하는 것보다

residual mapping을 fit 하는 것에 쉽다

$\rightarrow$  이전에 학습된 모델(layers)들의 출력과 학습된 layer의 출력의 차이값인

나머지 (residual)만 학습하면 되기에 연산이 간단해지고,

error 값 크기의 축면에서 학습이 더 쉽다.

## ○ 2가지의 실험결과를 제공

i) Plain network (simply stack layers)는 depth가 깊어짐에 따라  
더 높은 Training error를 보이는 것에 반해,  
제안한 deep residual network는 쉽게 최적화 가능

ii) deep residual network는 아주 깊어진 depth에서  
성능의 이득을 가져왔으며, 이전 연구보다 상상히 향상됨.

## ○ 또한, CIFAR-10 dataset에 대해서도 비슷한 현상을 보임

→ 저연한 네트워크 (residual network)는 특정 데이터에 구애되지 않음  
→ 100 layer를 넘어 1000 layer까지 실험.

## ○ Residual learning

→ 실제 상황에서  $H(x)$ 의 optimality identity mapping이 아닐지라도,  
이 reformulation은 문제에 precondition을 제공하는 효과를 줌.  
→ 만약 optimal function이 zero padding 보다 더 가깝다면,  
solver가 identity mapping을 참조하여 작은 변화  $F(x)$ 를 흡수하는  
새로운 function을 찾기 힘들다는 것보다 낫다

⇒ 즉 identity mapping이 합리적인 preconditioning을 제공하는 것은  
→  $F(x)+x$ 에서  $x$ 가 학습에 일종의 가이드 역할로써 학습을 도와줌.  
즉, identity가 optimal이 아닌 경우에도 이 reformulation은 여전히  
유익!

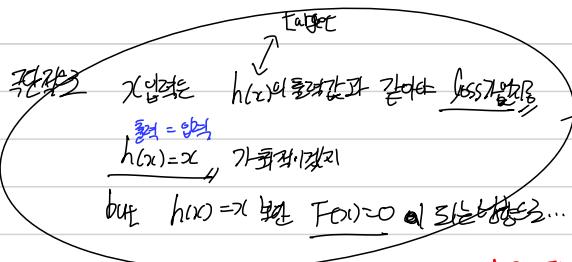
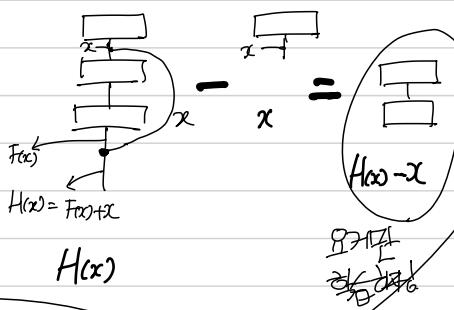
$$F(x)+x = h(x) \text{에서}$$

$h(x) = x$  일 때 optimal 하다  
 $(= F(x) = 0)$

⇒ 입력  $x$ 와 출력  $h(x)$ 가 같다면 (즉 단적으로, optimal 하게)  
그렇게 best고 이때

$$h(x) - x = 0 \Rightarrow F(x) = 0 \text{니까 } F(x) = 0 \text{이 되는거지!}$$

그러나 현실에서 실제로 이렇게 쉽지 않지만 guide 역할은  
작동!

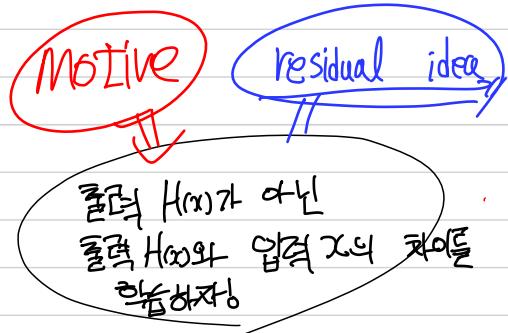


$(= F(x))$   
 $H(x)$ 는  $x$ 를 입력했을 때 그 출력값  $x$ 를 target값으로 mapping  
일반적으로  $H(x) - \text{target}$ 을 최소화 시키는 방향으로 가짐  
 $(= y)$

but 예전  $H(x) - x = 0$ 으로 만들어지는 방향으로 학습  
차단 학습하고 싶으면 왜냐하면  $H(x)$  자체가 identity function이라고 가정했기에.  
 $(= y)$

⇒ 출력값  $H(x)$ 와 입력값  $x$  간의 차이 (residual)을 학습!

⇒ residual learning



⇒ 출력  $H(x)$ 가 아닌  
출력  $H(x)$ 와 입력  $x$ 의 차이를  
학습하자는

= 작은 변화  $F(x)$

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

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

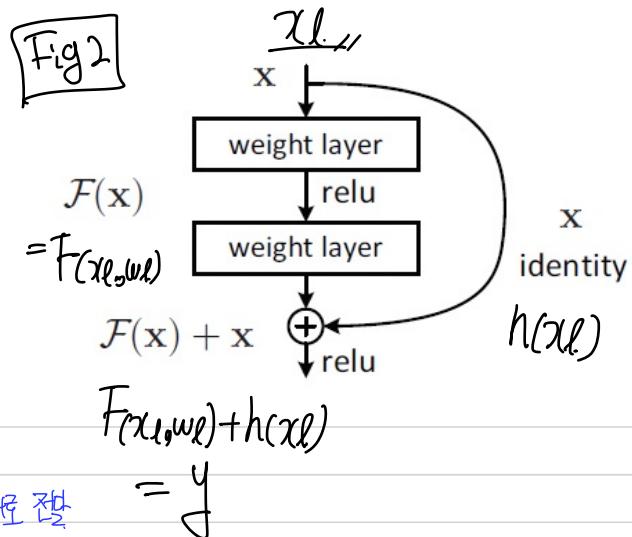
⇒  $x$ 를 위해서 선을 해야 그 어려움规避

⇒ Identity shortcut connection

역전파로 확인해보자.

$$y = h(x_l) + F(x_l, w_l)$$

② 네트워크의 output인  $y_L$ 는  
 기존 대체로의 연산인  $F(x_l, w_l)$ 에  $h(x_l)$ 를 더함.  
 ↓ convolution layer      ↗ weight 가 있는  
 shortcut connection  
 $= x_l$   
 $= \text{Identity } M$   
 $= \text{Input } \rightarrow \text{out } \oplus$



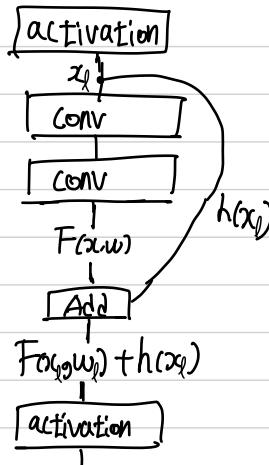
○  $y = h(x)$  이 Identity function이 아님을 증명하는가.

## Original Residual unit

$$\text{out}_l = y_l = F(x_l, w_l) + h(x_l)$$

$$\chi_{\ell+1} = f(y_\ell)$$

activation



$$x_{t+1} = y_t = F(x_t, w, \epsilon) + h(x_t)$$

$$f(F(x_p, w_p) + h(x_p))$$

이때 Precursive 하게 여러 lang2 표현

$$= \partial_{l+1}$$

$$= x_L$$

상각화하면 미니 배치마다 모든 테이터에서  $\lambda$ -인 경향은 점차  $\lambda$ 로  
가면서 weight 가 차단되고 gradient가 사라지면  $\lambda$ 을  
(vanishing gradient Problem)

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

어디부턴 증명해  
INPUT.      한 가짐  $l$  번째 layer의 output이자  
                 $l+1$  번째 layer의 input       $l \sim L$  사이의  
                convolution layer

$\Rightarrow$  Short cut connect의 역할

낮은 layer = 깊은  
상위 layer = 초반

## 이제 back Propagation 상태 확인

$$\frac{dE}{dx_l} = \frac{dE}{dx_l} \cdot \frac{dx_l}{dx_l} = \frac{dE}{dx_l} \left( x_l + \frac{1}{\lambda} F(x_l, w_i) \right)$$

$$= \frac{dE}{dx_i} \left( 1 + \frac{\sum_j}{T} F(x_{ij}, w_i) \right)$$

$$W_{t+1} = W_t - \gamma, \frac{\frac{1}{2}E}{J_{W_t}} \quad \begin{array}{l} \text{i) } \frac{\frac{1}{2}E}{J_{W_t}} \\ \text{ii) } \frac{\frac{1}{2}E}{J_{W_t}} \end{array} \quad \begin{array}{l} \text{conv} \\ \text{conv} \end{array} \quad \begin{array}{l} \text{2개만 가능} \\ \text{1-1} \end{array}$$

## ~~→ i) 0~~ 2번째 항

$$\frac{\partial E}{\partial x_i} \cdot \frac{1}{f(x)} F(x_i, w_i)$$

깊이에 관계없이 일정하게 back Propagation을 통해 전달  
 $\Rightarrow$  경과적으로 최소화의 gradient를 보장  $\rightarrow$  No vanishing gradient Problem  
 $\Rightarrow$  feature에 대한 변화를 일정률로 유지. gradient Problem

$\Rightarrow$  각 weight 가 total output에 기여하는 정도도  
 일정 수준 이상으로 무작위 가능  $\rightarrow \frac{dE}{dx_1} = \frac{dE}{x_1}, \frac{dE}{dx_2}, \dots, \frac{dE}{dx_n}$

## 3.2 Identity Mapping by shortcuts

$$y = F(x_e, w_e) + h(x)$$

identity

$y = F(x_e, w_e) + x$

input

$y = F(x_e, w_e) + \text{relu}(w_s x)$

output

Projection

○ 이때  $F(x_e, w_e) + x$ 은 shortcut connection과 element-wise add (원소 합)

○ add 이후 activation (ReLU) 사용.

○ 이때  $F(x_e, w_e)$  와  $x$ 는 둘의 dimension이 같아야함.  
 (channel 갯수)  
 (feature map size)  
 (node의 갯수)

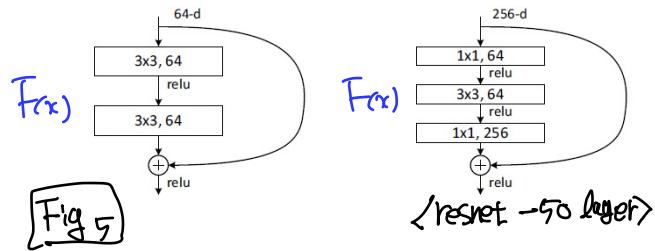
○  $F(x_e, w_e)$  와  $x$ 의 dimension이 다를 때

$x$  대신  $W_s x$ 를 사용.

(dimension을 맞추기 위해)

(= Linear Projection  $W_s$ )

○ identity  $x$  대신 projection  $W_s x$ 를 써도 되지만  $x$ 를 위해 identity  $x$ 를 끌면 효율적  
 ⇒ matching dimension 할 때.



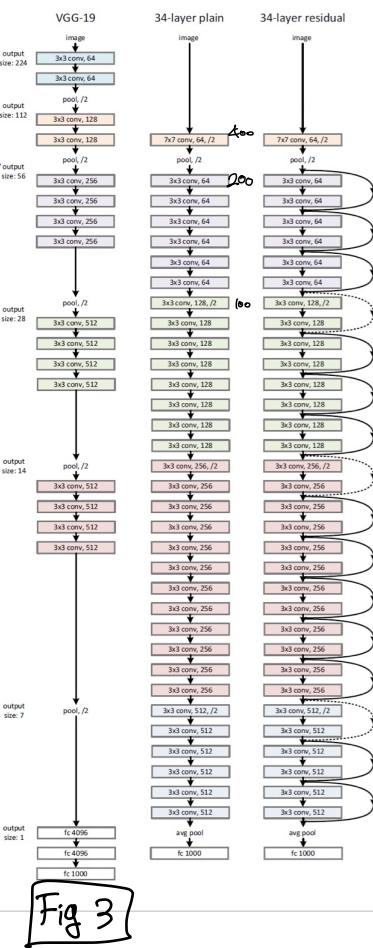
○ 이때  $F(x)$ 가 1 layer라면 Shortcut의 장점이 사라짐  
 ⇒ 2개, 3개 이상의 jump를 해야 효과가 있음.

### 3.3 Network Architectures

<rule>

Plain net

- VGGnet에서 inspired 함.
- 거진  $3 \times 3$  filter 사용  $\rightarrow 400, 200, 100$
- 같은 output feature map size를 갖는 layers들을 같은 filter 개수를 가진다.  $\rightarrow 32, 64, 128...$
- feature map size가 절반인 경우, time complexity per layer를 보존하기 위해 filtering 단계를 2번을 한다.
- 아래 down-sampling은 Pooling이 아닌 [Strides 2]로 사용.
- 마지막은 global-average-Pooling을 사용.
- 후에 헤드의 Fully-connected 사용.  
ex) 여기서 24 layer를 가짐



Residual net

<rule>

- input과 output의 동일한 dimension일 때 identity shortcut 사용 (설선)

- dimension이 증가할 때 (접선)

i) zero entry를 추가로 Padding 하기  
dimension matching 후 identity shortcut 사용.  
 $\Rightarrow$  zero padding으로 channel을 맞춘다.

ii) Projection shortcut을 dimension matching  
 $\Rightarrow 1 \times 1$  Conv로 dimension을 맞춘다.

- Shortcut이 다른 크기의 feature map all mapping 될 경우, 따로 identity로 [Strides 2]로 down-sampling

### 3.4 Implementation

- Scale augmentation을 위해

- Convolution과 activation 사이에 Batch-Normalization을 배치.
- weight initialize를 he normal을 사용.
- SGD optimizer 사용.
- batch size = 256
- learning rate = 0.1에서 시작, plateau가 생기면 rate를  $\frac{1}{10}$ 로 적용.
- weight decay = 0.0001, momentum = 0.9 callback에서 factor = 0.1, patience는 상황에 따라..
- iteration = 600k = 600,000

ex) keras,

Sgd = optimizers.SGD( $lr=0.1$ ,  $decay=1e-6$ ,  $momentum=0.9$ )  
model.compile( $loss='mse'$ ,  $optimizer=Sgd$ )

# 4.1 ImageNet classification.

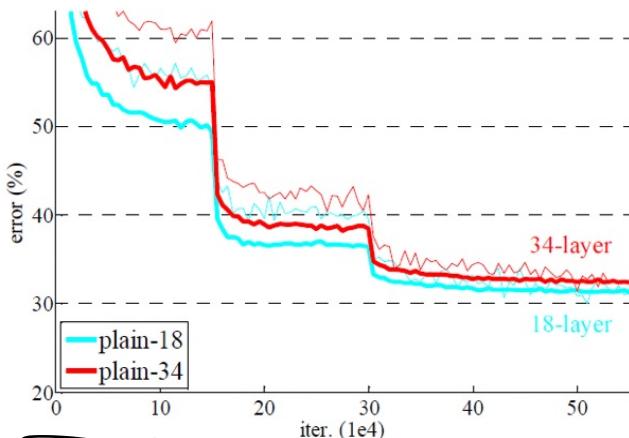
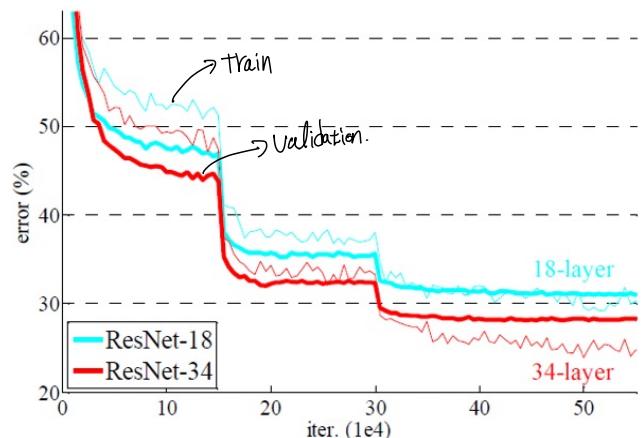


Fig 4.



- 1000 개의 class로 된 Imagenet 2012 classification dataset에서 평가
- 1.28M - training, 50k-validation, 100k-test data set.  
(90:3.6:7.2)

Plain net 18 > Plain net 34

Resnet 34 > Resnet 18

⇒ Plain net 18의 solution space는 Plain net 34의 subspace임에도 Plain net 34의 차이가 높다.

Plain net 과 같은 baseline architecture 사용.  
비교를 위해 Parameter가 더 들어가지 않는 zero Padding 방식의 Projection shortcut 사용.  
(Input과 output의 dimension이 같을 때 Identity shortcut)

⇒ 단순히 이 문제가 optimization difficulty 가 vanishing gradient에 의한 것 같지 않다고 생각함

⇒ why? Batch-Normalization으로 인해 forward Propagate signals이  
분산이 0이 아니도록 함.

and back propagation에서도 healthy form을 가지는 걸 확인함

⇒ 그럼에도 여전히 SGD의 작동을 험겨운

Exponentially low convergence rate를 가지며

이로 인해 Train error가 성기지 않았나 추측

(지수적으로 높은 수렴률)

optimizer의 weight decay 및 momentum rate가 계속 감소하기

다른 예?

Three observations.

① Resnet 34 > Resnet 18 이 더 error가 낮았으며

⇒ depth 차에도 합리적인 accuracy를 이제 얻을 수 있다.

② Plain net 34 보다 Resnet 34의 val-error가 3.5% 더 낮았다.

⇒ residual learning으로 극단적으로 깊은 model에서도 효율적인(정가운) 학습(accuracy)를 얻음.

③ Plain net 18 보다 Resnet 18 이 수렴이 더 빨리 됐다.

⇒ cost가 더 줄어들수 있게 된거지.

# Identity shortcut vs Projection shortcut.

① short cut의 방법에 종 3가지가 있음.

i) 모두 Zero Padding을 이용

⇒ zero padding을 이용하여 dimension을 확장시키고  
모든 shortcut의 Parameter는 free lr.  
(학습률)

ii) Projection shortcut + identity shortcut. 사용.

⇒ input과 output의 dimension이 다른 경우에는 Projection shortcut을,  
그 외에는 Identity shortcut을 사용.

iii) 모두 Projection shortcut 사용

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
ReLU-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	<b>21.43</b>	<b>5.71</b>

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.

result) 2번이 가장 뛰어남.

why) Table 3을 보면 Resnet A i) Resnet B ii) Resnet C iii) 가 Plain net 보단 일단 높은 편이다.

i < ii → zero padding으로 확장된 dimension은 residual learning이 되지 않을

ii < iii → 추가된 Parameter로 인해 성능이 조금 더 좋아진 걸로 보임.

그러나. 1, 2, 3의 대부분 차이는 본질적으로 성능 저하 문제를 다루지 않는다.

따라서 [model sizes]와 [memory/time complexity]를 줄여야 하는

2번을 사용 → 당연히 Parameter가 많아지니까.

# Deeper Bottleneck Architect

- layer가 depth 해결수록 감당할수 없는 training time에 대한 문제로 인해 block을 bottleneck design으로 수정한다.  
(=residual block)

2-layer → 3-layer

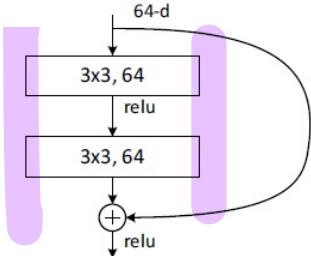
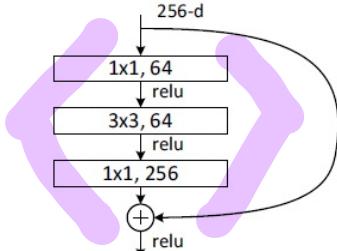


Fig 5



Resnet-50/101/152에서 bottleneck 사용.

- 이때 bottleneck의 3-layer에 있는 1x1 convolution을 이용
- 적은 연산으로 dimension을 늘리거나 줄일 수 있음.
- 결국 2개의 model은 비슷한 time complexity를 가짐.

(특히 bottleneck에서 Identity connection의 역할이 더 두드러짐)

만약 Projection connection으로 연결된다면. 2개의 high-dimensional 풀과 연결되므로 [time complexity]와 [model size]가 2배가 된다.

Parameter?

- bottleneck (3-layer)을 이용하여 Resnet 34를 Resnet 50으로 구성
- 이런 식으로 101-layer, 152-layer Resnet 구성. → 그럼에도 VGGnet - 16 layer보다 15.3 billion FLOPs

11.3 billion FLOPs

FLOPs가 낮다

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	<b>21.43</b>	<b>5.71</b>

2x2x2 병법  
(Projection + Identity)

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43 <sup>†</sup>
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 (2번재)	21.84	5.71
ResNet-34 C (3번재)	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	<b>19.38</b>	<b>4.49</b>

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.

bottleneck을 이용해 layer를 늘려 만든 model들이 IT error가 낮네.

그렇게 stacked 된 layer로 만든 Project들로  
Previous best single model 성능과 비교  
(= not ensemble) by Imagenet validation data

# Exploring Over 1000 layers

⇒  $n=200$  일 때 1201 layers - resnet 으로 극단적으로 학습했을 때  
110-layers - resnet 보다 test 결과가 좋지 않았다  
(training error가 낮았음에도..)

why?  
⇒ 19 Million 의 Flops 를 활용할 data 가 너무 적음.  
⇒ 당연히 overfitting //

그에 걸맞는 dataset 이 있었다면 overfitting이 안 될수 있을까?