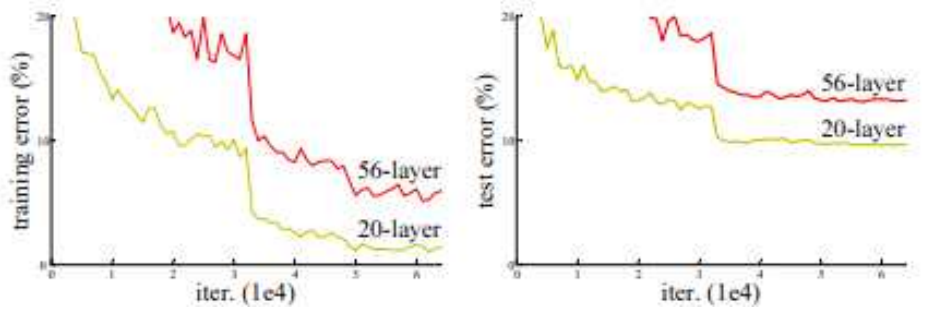
ResNet 논문 리뷰
- deep residual learning for image recognition (CVPR 2016), 이미지 분류 대회 우승
- 이해하기 쉬우면서 큰 성능 향상
- 네트워크를 깊게 만들기 위해 잔여(잔차)학습(residual learning) 제안

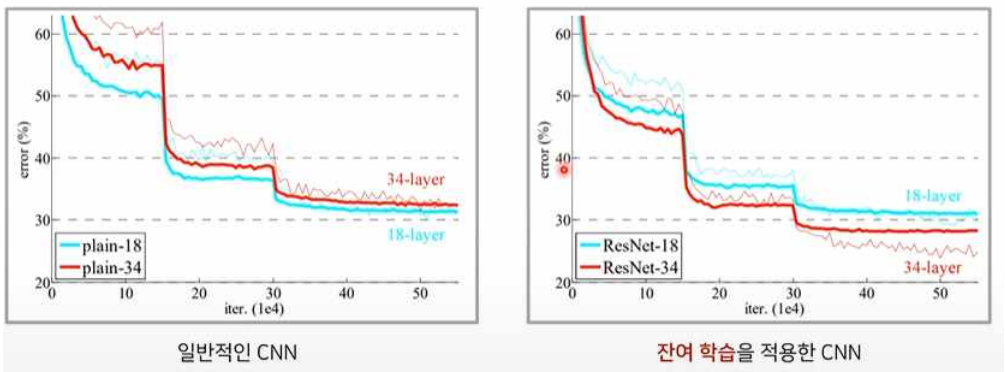
1. 배경
- 깊어질수록 풍부한 특징 추출 > 높은 성능
- 딥러닝 모델에서 layer의 수는 중요한 요소지만, layer를 쌓을수록 항상 성능이 좋아지는가?

- vanishing or exploding gradients problem 발생
: 가중치 업데이트 x, 학습 잘 안됨, 속도느림 or 지나치게 큰 가중치 업데이트 > 학습 불안정, 수렴X
▶ normalized initialization, intermediate normalization, SGD등으로 해결 가능

그럼에도 해결 되지 않는 문제
▶ Degradation(성능저하)



< ImageNet top-1 training error >

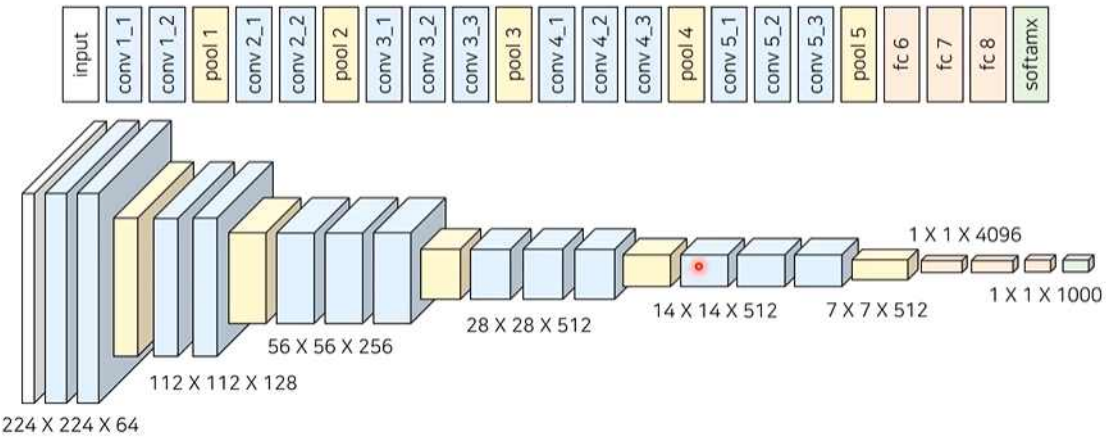


일반적인 CNN 잔여 학습을 적용한 CNN

- 기존 CNN : 네트워크 깊어질수록 성능 악화
- 잔여학습을 적용하였을 때, 18-layer보다 34-layer가 더 성능이 좋았음

*VGGNet(ICLR 2015)

- VGG 네트워크는 작은 크기의 3x3 컨볼루션 필터(filter)를 이용해 레이어의 깊이를 늘려 우수한 성능을 보입니다.



- 다운샘플링으로 너비, 높이는 줄어들고, 레이어 깊이를 늘려 채널 값은 증가 > 최종적으로 1000개의 클래스에 대한 확률값을 뽑아냄
- 단순하게 레이어를 깊게 한다고 성능 좋아지지 x & 큰 파라미터 수

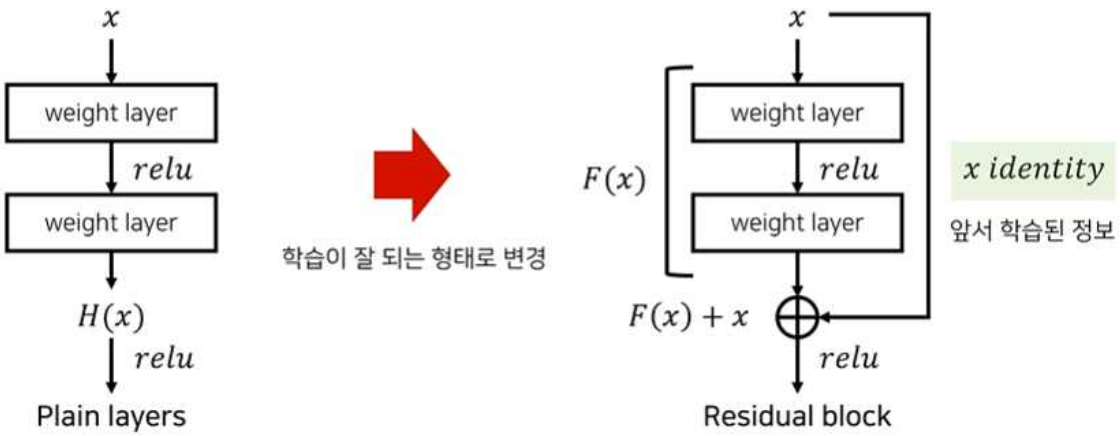
▶ Resnet 등장

2. 아이디어

> 잔여 블록(Residual block)

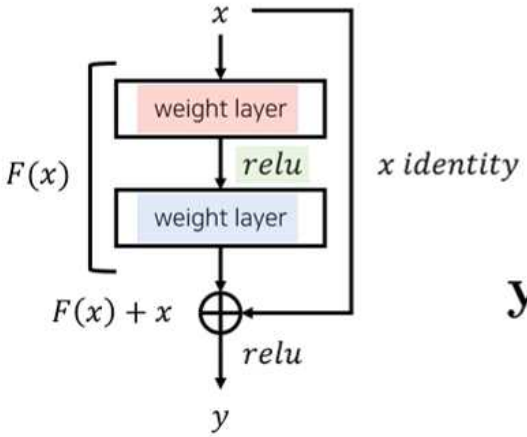
- 잔여블록을 이용해 네트워크 최적화 난이도↓
- 학습을 하고자 하는 mapping인 H(x)를 곧바로 학습하는 것은 어려움

-> 대신 F(x) = H(x)-x를 학습



- x > conv1 특징추출 > relu(비선형) > conv2 > relu > H(x) : 이상적인 함수
- H(x)는 각각 conv이 분리되어 있어 가중치를 개별적으로 학습해야함 > 수렴 난이도↑
- conv 두 번 거친 결과 F(x) + x = 앞서 학습된 정보 x를 그대로 가져오고 잔여한 정보인 F(x)를 더한다 > F(x)만 학습하면 되므로 수렴 난이도↓
- shortcut connection : 추가적인 파라미터 x(복잡도 증가x), 간단한 구현

*수식



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

일반적인 형태

$$\mathbf{y} = \underbrace{\mathcal{F}(\mathbf{x}, \{W_i\})}_{\text{multiple convolutional layers}} + \underbrace{W_s\mathbf{x}}_{\text{shortcut}}$$

- ((입력값 x * w1) > relu)*w2 = F(x)
- input(x), output의 차원이 같으면 identity mapping 수행
- 차원이 증가하면

*zero padding 적용 : identity mapping 효과

*linear projection mapping 사용(Ws) : 1x1 conv layer 활용

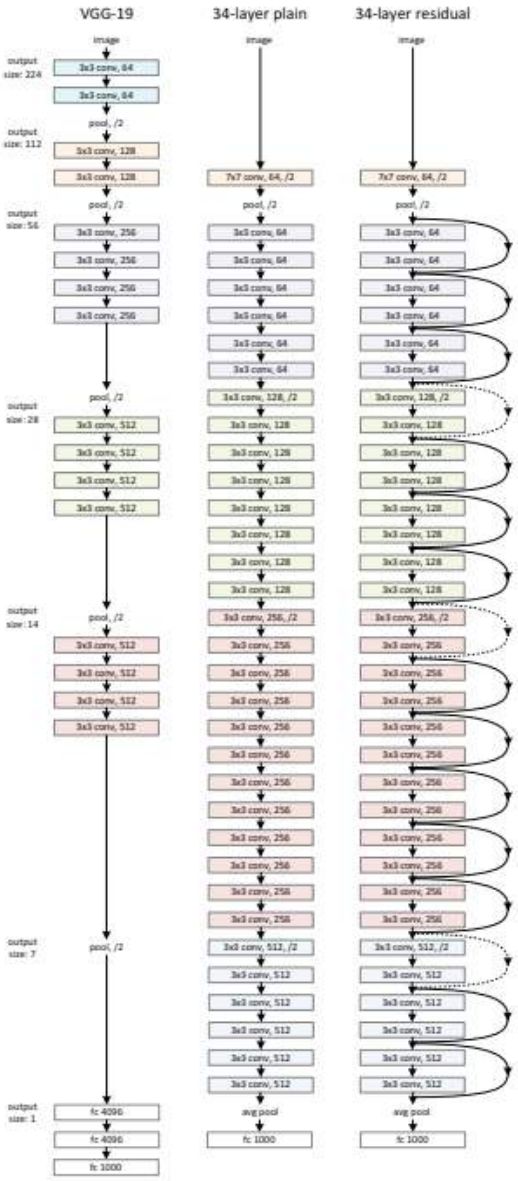
3. 구조

plain network

- VGGnet에서 착안
- 3x3 필터 사용
- output 특징맵 크기를 같게 하기 위해 같은 필터 수 사용
- 특징맵이 절반이 되면 채널 값을 2배로 늘림
- 풀링x, stride=2로 다운샘플링 진행
- GAP 사용, 1000개의 fc층 생성
- vgg보다 적은 파라미터 사용, 낮은 복잡도

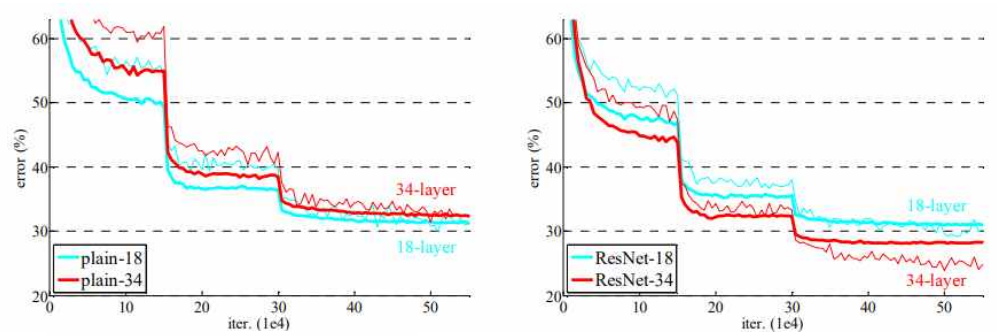
▶ vgg, 34-layer plain + residual = 34-layer residual

- 점선은 입출력 차원이 달라 zero padding/projection mapping 적용
- conv마다 배치정규화 적용



4. 성능

plain network vs residual 적용 network



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

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.

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.

A : zero padding 후 identity mapping B : 차원증가에만 projection 적용, c : 모든 shortcut에 projection 적용

▶ C가 가장 성능이 좋았지만, 필수라고 할 정도로 높은 성능 개선은 아님

5. 코드 설명

Basic Block : 2개의 conv layer(x>conv>배치>relu>conv2>배치+x > relu)

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