ResNet 논문 study

- 작성자: 22기 이세훈
- 논문 저자: He, et. al (2015) ← 마이크로소프트 리서치팀 소속으로 발표
- 원논문명: **'Deep Residual Learning for Image Recognition'**
- DOI: https://doi.org/10.48550/arXiv.1512.03385

📍 3줄요약 by 세훈

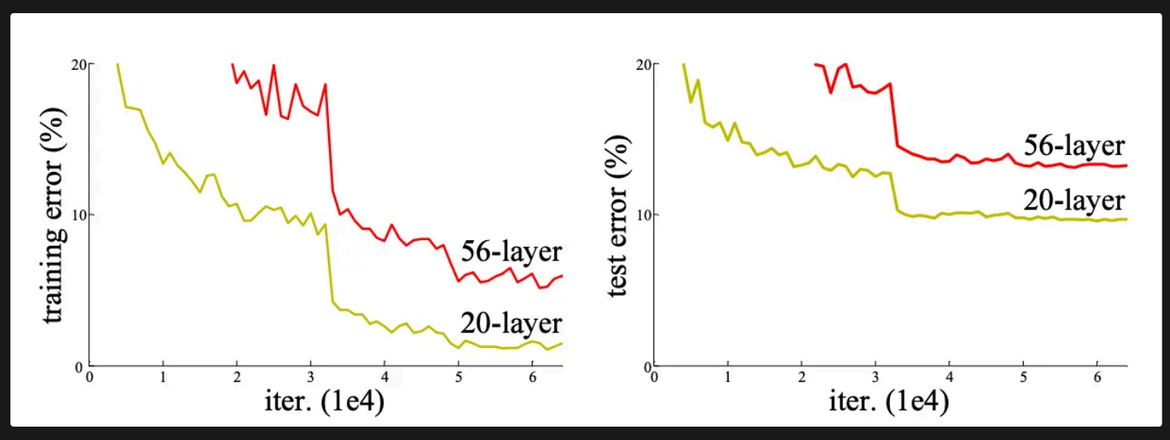
1. 딥러닝(특히 이미지) 네트워크의 레이어를 많이 쌓아도 error가 오히려 커지는 현상 발견됨

   → 어떻게 깊이를 더 늘려 결과를 좋게 할 수 있을까?
2. 'Residual Learning' (Skip connection) 도입!!

   → f(x) + x (입력값 한 번 더함) 형태로 더 깊게 레이어를 쌓을 수 있게됨 (미분해도 1이 남음)
3. 깊은(ex: 152) 네트워크에서도 성능 저하 없이 정확도가 향상, 다양한 벤치마크에서 SOTA를 달성

---

1. Introduction



- backgrounds
  - 'deep' CNN은 이미지 분류 task에서 큰 성과를 내며, 레이어가 쌓일수록 저~고수준 특징을 모두 함께 학습할 수 있다고 알려짐
  - 최근 연구(2015년 기준!)는 네트워크의 깊이(depth)가 성능 향상에 매우 중요하다는 것을 보임.
    - ImageNet SOTA 모델들은 16~30층 이상의 'very deep'한 구조 사용
  - 분류 뿐 아니라 다른 vision 과제(물체 인식, 분할 등)에서도 **"deeper → better"** 경향
- 문제 제기: *"Is learning better networks as easy as stacking more layers?"*
  - 여태까지는 vanishing / exploding gradient 문제가 학습의 주요한 걸림돌.
    - 그러나 여러 initialization, normalization으로 완화 가능했음 (지지난 주차 참고)
  - 새로운 문제: Degradation Problem (더 general한, 깊이로 인한 성능 저하 문제)
    - 단순한 overfitting이 아님
      - 깊이 증가 → training error 증가 (figure 참고! overfitting이라면 train은 56-layer가 더 낮고 test error만 높아야..)
    - 단순히 layer를 추가하면 identity mapping을 구현하는 해가 존재해도 잘 찾지 못함
- Residual Learning을 해봅시다

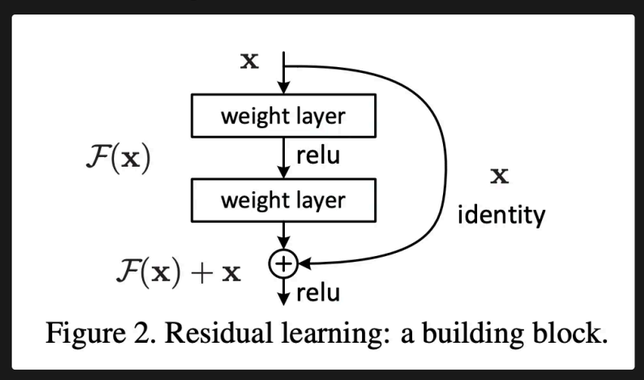


Figure 2. Residual learning: a building block.

  - 기존: 여러 층이 직접 목표 H(x) 근사하도록 학습 (recall: universal approximation)
  - 제안: 대신 **잔차(residual) 함수** F(x) = H(x) - x를 학습하고, 최종 출력은 F(x) + x 형태로 만들자
    - 만약 항등매핑이 최적인 경우, F(x)를 0으로 만드는 것이 더 쉽기 때문에 학습이 단순해짐
      - 이해를 위한 보충: 20층 레이어에서 진짜 굿 성능

        → 21층에서 "F(x) 너는 차라리 뭐 하려 하지말고 그냥 가만히 있어"
        - 기존에는 어떻게든 또 학습해서 새롭게 F(x)를 찾아서 H(x)를 근사했어야.
  - 구현: **Shortcut Connection**을 사용해 x를 다음 층 출력에 더함.
    - 항등 매핑(x)인 경우 파라미터와 연산량 증가 없음
    - 네트워크 전체를 end-to-end로 학습 가능! (한 번에 모든 웨이트를 업데이트)

2. Related work

2.1. Residual Representation

- 이미 예전부터 이미지 표현에 '잔차' 개념이 쓰임.
  - ex. VLAD, Fisher Vector → 딕셔너리나 평균값과의 차이(잔차)를 인코딩에 사용
  - Multigrid 등 수치해석 기법: 잔차 문제로 바꿔 풀어 최적화를 쉽게 함
- 공통점: **원본 값을 직접 다루기보다, 차이(잔차)를 다루면 학습·최적화가 단순해진다**는 점.

2.2. Shortcut Connections

- MLP, Inception 구조에서 입력을 중간·출력에 직접 연결하는 지름길 아이디어
- Highway Networks는 게이트가 달린 shortcut을 사용

  → 열려있다가 필요 시 닫히고, 파라미터를 갖는다는 특징
- ResNet은 **항상 열려 있는 항등 shortcut**을 사용

  → 파라미터 없이 입력이 그대로 흐르고, 블록은 오직 잔차만 학습하는 개념

콘텐츠로 건너뛰기

S **Sehun** 8월 10일 (편집됨)
'vanishing / exploding gradient' , 'Degradation Problem'

S **Sehun** 8월 10일
State of the Art. 짱 먹었다는 뜻

S **Sehun** 8월 10일
이미지넷은 대규모 데이터베이스로, 이미지 분류를 위한 벤치마크를 제공한다.[1] 딥러닝과 컴퓨터 비전 분야 발전을 목표로 하며, 비상업적 용도에 한하여 해당 분야 연구원들에게 무료로 제공한다. 2023년 기준으로 약 1,420만 개의 이미지와 20,000개 이상의 ... 더 보기

S **Sehun** 8월 10일 (편집됨)
layer를 더하는 것 자체의 optimization solving에 구조적인 문제가 있다.. 라고 이해하면 될듯

이해를 위한 보충설명 +)
setting: 이미 학습이 완료된 얕은 ... 더 보기

S **Sehun** 8월 10일
잔차(Residual): 입력 x와 목표 출력 H(x)의 차이 F(x) = H(x) - x를 의미

- 네트워크가 직접 전체 함수를 학습하는 대신 변화량(차이)만 학습하도록 하는 개념.
- ResNet에서는 이 잔차를 이용해 ... 더 보기

S **Sehun** 8월 10일
Inception 구조 : 1x1 conv를 일반 conv 사이에 사용하고 max pooling을 적절히 활용, 연산을 간단히함

3. Deep Residual Learning (모델 아키텍쳐)

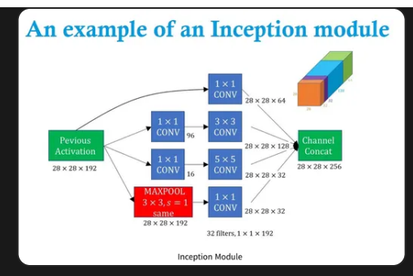
▼ 3.1. Residual Learning

- **기존 방식**: 여러 층이 쌓인 블록이 목표 함수 $H(x)$ 자체를 근사하도록 학습.
- **ResNet 방식**: H(x) 대신 **잔차 함수**

  $F(x) = H(x) - x$를 학습, 최종 출력은 $H(x) = F(x) + x$

▼ 3.2. Identity Mapping by Shortcuts

- 블록 형태: $y = F(x, \{W_i\}) + x$
  - x: shortcut connection을 통해 (앞의 입력값과) 그대로 전달된 입력
  - 마지막에 element-wise 덧셈 후 활성함수(ReLU) 적용
  - Conv 레이어에도 동일하게 적용 가능 → feature map 채널별 덧셈을 하면 됨!



S **Sehun** 8월 10일

S **Sehun** 8월 10일

하이웨이 네트워크 구조

S **Sehun** 1일

텐서 위치가 같은 성분끼리 다 더함.

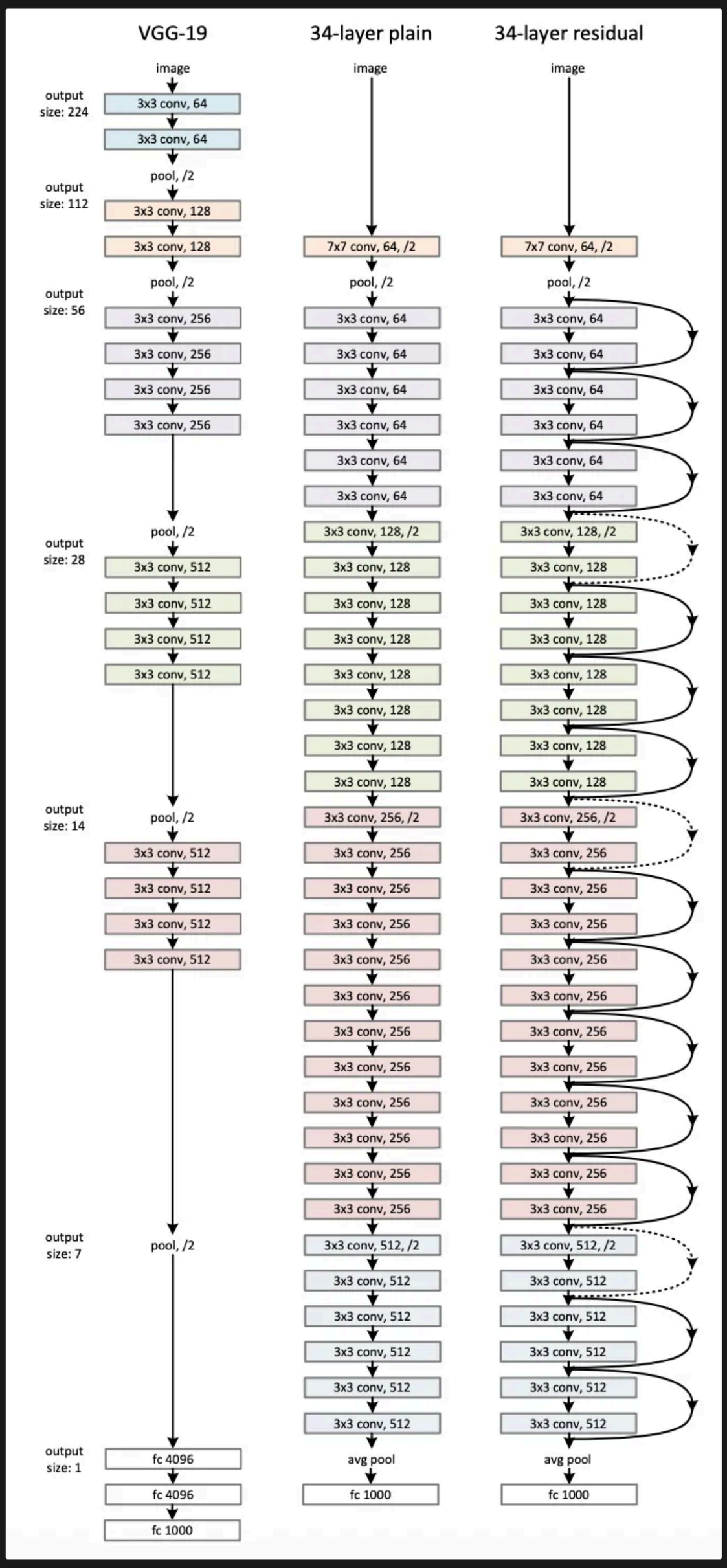
필요조건: 텐서 크기, 채널 수가 동일해야함.

S **Sehun** 1일

작은 커널 여러개 사용, 모든 conv와 FC 층 뒤에 ReLU 사용해서 빠른 학습 및 설명력 + 파라미터수 줄이기

콘텐츠로 건너뛰기

▼ 3.3. Network Architectures



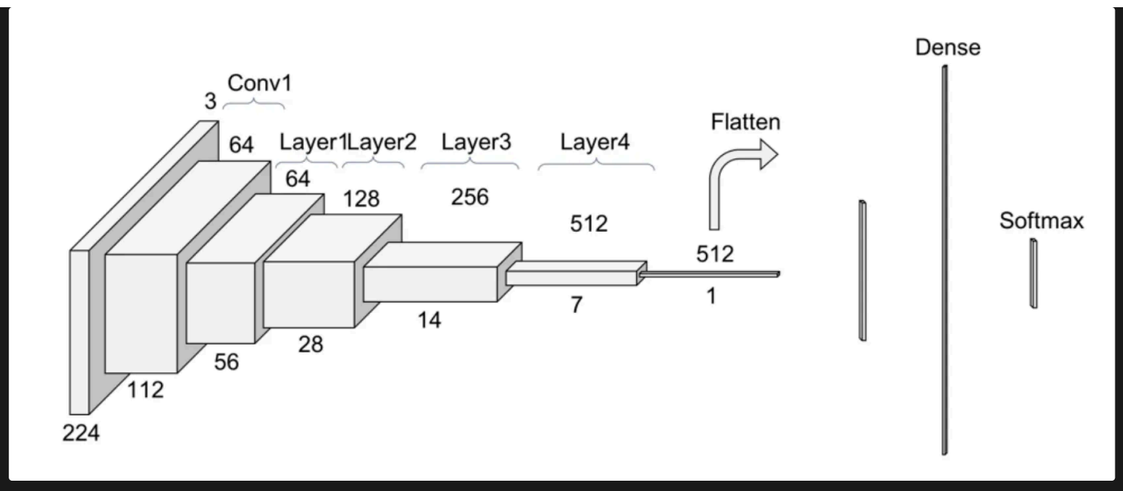
좌: 19layer- VGG / 중: 34 layer plain / 우: 34 layer ResNet

- **Plain Network**: VGG 스타일 3×3 conv 반복, downsampling 시 채널 수 2배
- **Residual Network**: 같은 구조에 각 두 개의 3×3 conv 블록마다 shortcut 추가
- 차원 증가 시 두 가지 옵션:
  - (A) 항등매핑 + zero-padding
  - (B) projection(1×1 conv) 사용
- **Bottleneck 블록**: (1×1 → 3×3 → 1×1) 구조로 차원을 줄였다 복원

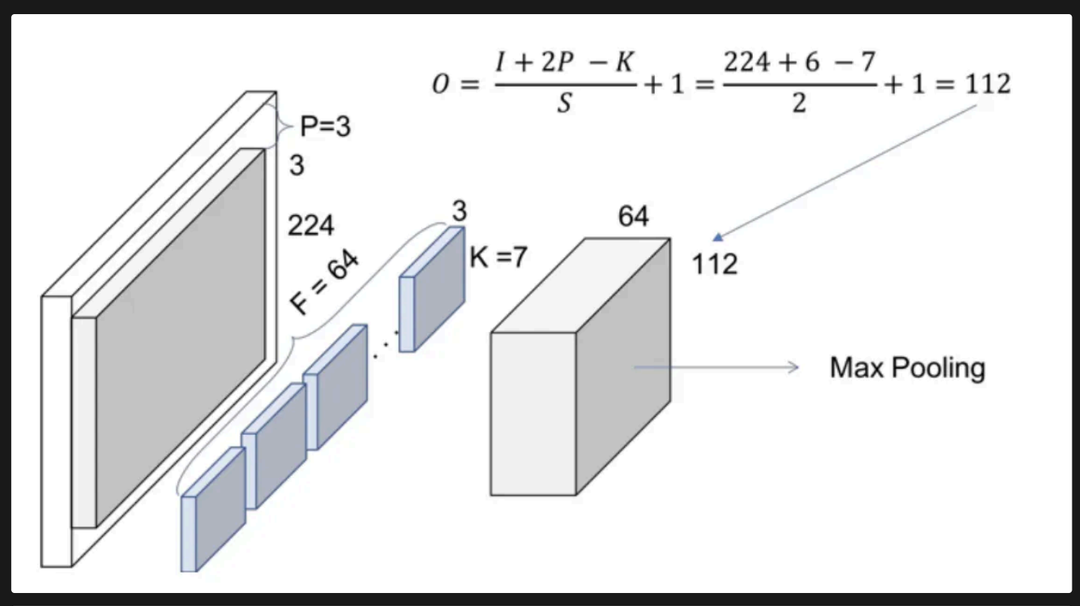
  → 더 깊은 네트워크를 효율적으로 구성 가능.

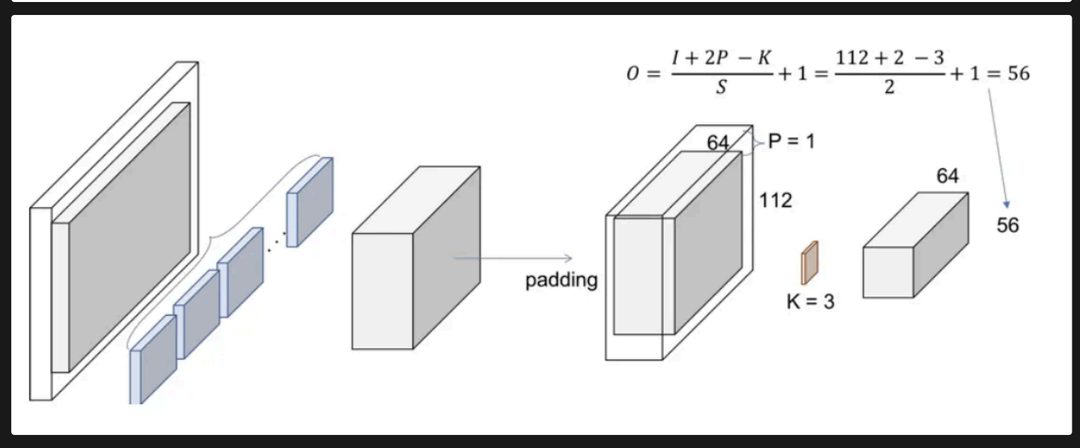
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
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

콘텐츠로 건너뛰기



- size, layer에 초점을 맞춘 네트워크 아키텍쳐 구성표
  - 그림자료 참고: https://wikidocs.net/164800
- ‘conv1’





  - 공통적인 block 전에 7x7*64 conv (stride 2, padding 3) + BN + max pool
    - (224x224x3) → (112x112x64) (conv) +BN
    - (112x112x64) → (56x56x64)
    - conv로 feature map을 깊게 학습한 뒤 max pooling까지 써서 크기 조정

콘텐츠로 건너뛰기