

# CNN 모델의 발전



- LeNet

- CNN을 처음으로 개발한 얀 르쿤(Yann Lecun) 연구팀이 1998년에 개발한 CNN 알고리즘의 이름
- 논문: "Gradient-based learning applied to document recognition"
- 손으로 적힌 우편 번호를 전통적인 방법보다 효율적으로 확인하기 위해 고안
  - 전통적인 모델은 사람이 직접 만든 특징 추출기로 추출하고 분류기를 붙여 사용

- 전통적인 방법의 한계

- 사람이 만든 특징 추출기는 제한된 특징만 추출

- 설계자가 생각한 정보만 추출되어 활용됨

- 너무 많은 매개변수를 포함

- 분류기에 사용되는 FC의 경우 엄청나게 많은 가중치를 포함
    - 시스템의 capacity를 증가시키므로 더 많은 훈련 셋과 메모리 저장공간이 많이 필요

- 입력 값의 Topology가 완전히 무시됨

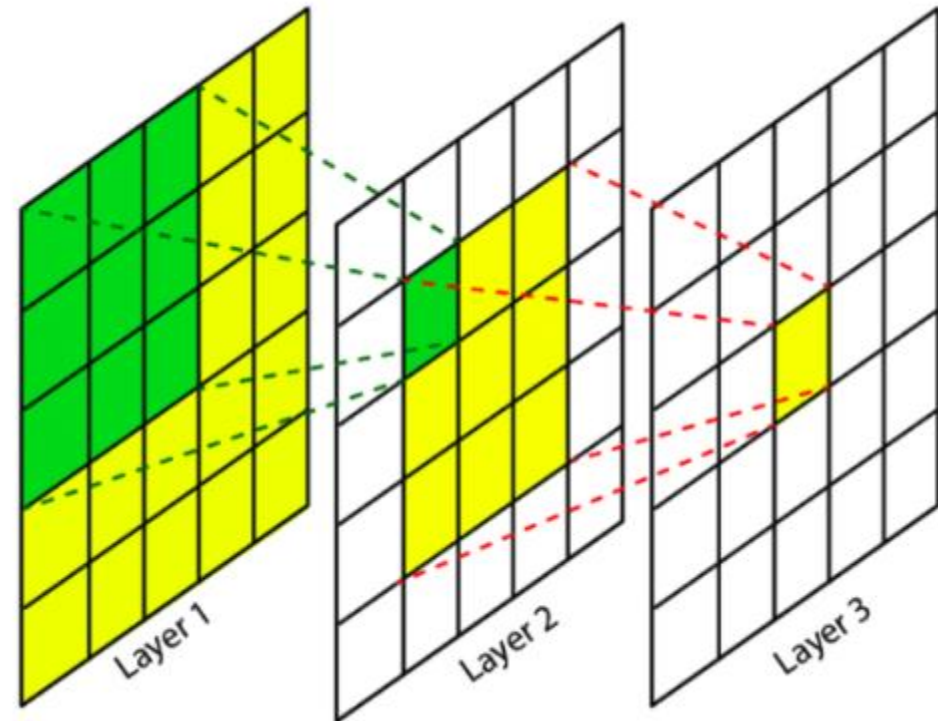
- 이미지는 공간적인 특징, 상관관계를 가지는데 FC는 공간적인 정보를 이용하지 못함
    - LeNet 발표 전에 활용되던 FC는 2차원 이미지를 1차원으로 펼쳐진 데이터를 입력함

- 문자 인식 업무에서의 CNN

- CNN은 약간의 shift, scale, distortion 불변성을 갖기 위해 세 개의 아이디어를 결합함

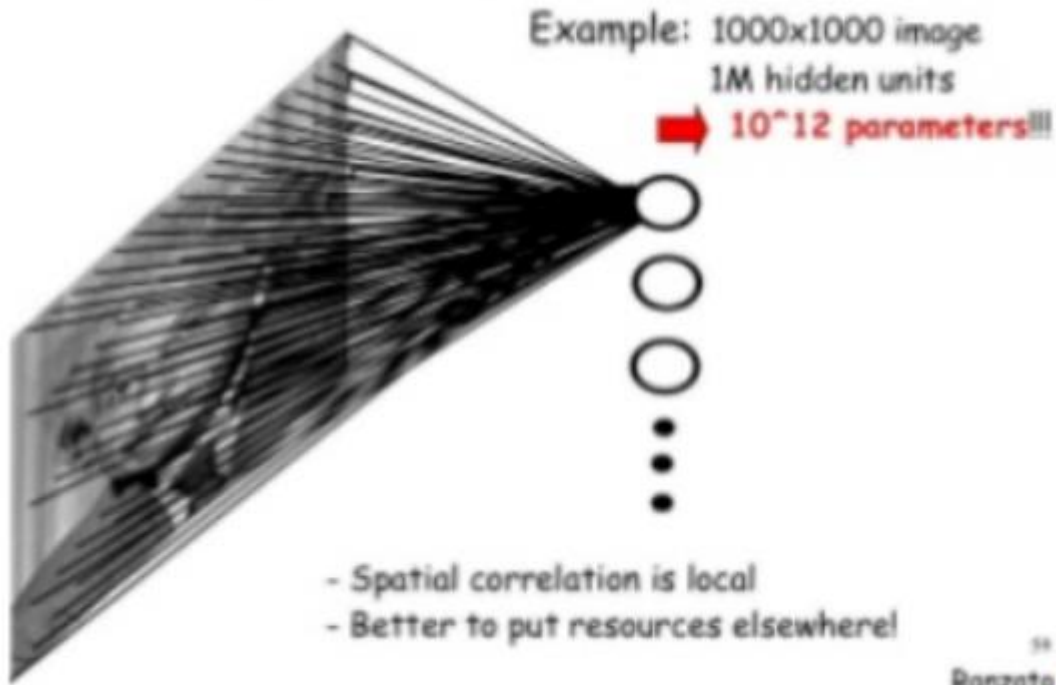
- 수용 영역(Receptive Field)

- hidden unit의 receptive field를 local로 제한함으로써 local feature를 추출



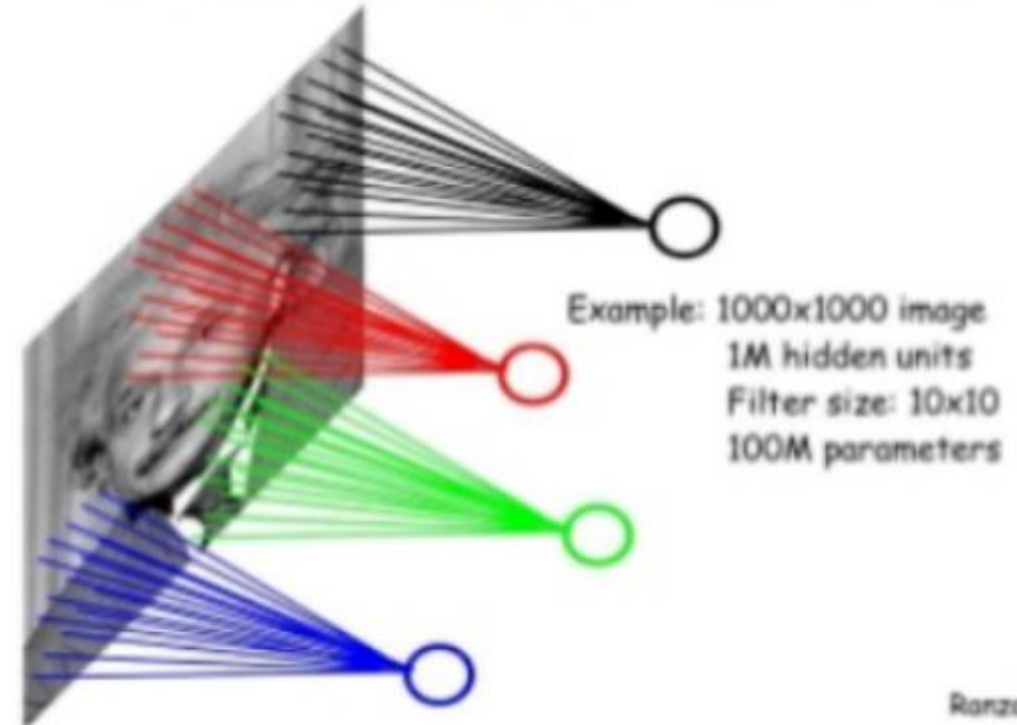
# 최초의 CNN 모델: LeNet

## FULLY CONNECTED NEURAL NET



58  
Ranzato

## LOCALLY CONNECTED NEURAL NET



- 가중치 공유 (Shared Weight)

- CNN은 가중치 배열을 강제로 복제함으로써 자동으로 shift 불변성 확보
- Feature map의 unit은 동일한 weights와 bias를 공유
  - 공유된 weight 집합을 컨볼루션 커널로 이용, 입력 데이터의 모든 위치에서 동일한 특징 추출
  - 예, 5x5 kernel은 5x5사이즈와 설정된 Stride에 맞춰 feature map를 돌아다니며 계산하지만, 5x5의 weight와 1개의 bias만 back propagation으로 학습 수행

# 최초의 CNN 모델: LeNet

- weight 공유 → 커널을 총 몇 개로 설정하는가에 따라 출력인 Feature Map의 수와 학습해야 하는 파라미터만 증가(학습 파라미터가 증가하는 것이 아님)
  - 요구 계산량 감소, 학습 파라미터 감소 → 과적합 방지 → 학습~검증 오류간 갭 감소
  - 실제로 LeNet-5에는 340,908개의 커넥션이 있지만 60,000개의 학습 파라미터만 존재
- 가중치 공유로 인하여 입력 이미지가 변환되면 feature map의 결과값도 동일한 양만큼 변환됨 → 입력의 왜곡과 변환에 대한 Robust(강건함) 확보

## • 서브 샘플링 (Subsampling)

- 현재 CNN 모델에서의 Pooling을 의미함 (LeNet-5에서는 Average Pooling 이용)
- 특징이 한 번 검출되면 위치 정보의 중요성은 감소함(이미 처리가 끝났으므로)
- 각 특징의 위치 정보는 패턴 식별과는 무관하며, 입력 값에 따라 특징이 나타나는 위치가 다를 가능성이 높음 → 잠재적 유해 정보가 됨
  - Feature Map으로 인코딩 되는 특징들의 위치에 대한 정확도를 감소시키기 위한 가장 간단한 방법 → Feature Map의 해상도를 감소시키는 것
  - 서브샘플링 레이어에서 지역평균값과 서브샘플링을 수행하여 Feature Map의 해상도를 감소시키고 왜곡(Distortion)과 이동(Shift)에 대한 민감도를 감소시킬 수 있음
- 위치 정보 소실로 인한 손실 → Feature Map 크기가 작아질수록 더 많은 Filter를 사용하여 다양한 Feature를 추출함으로써 보완함



## • LeNet-5의 구조

C1~F6까지 레이어의 활성화 함수는 tanh를 사용

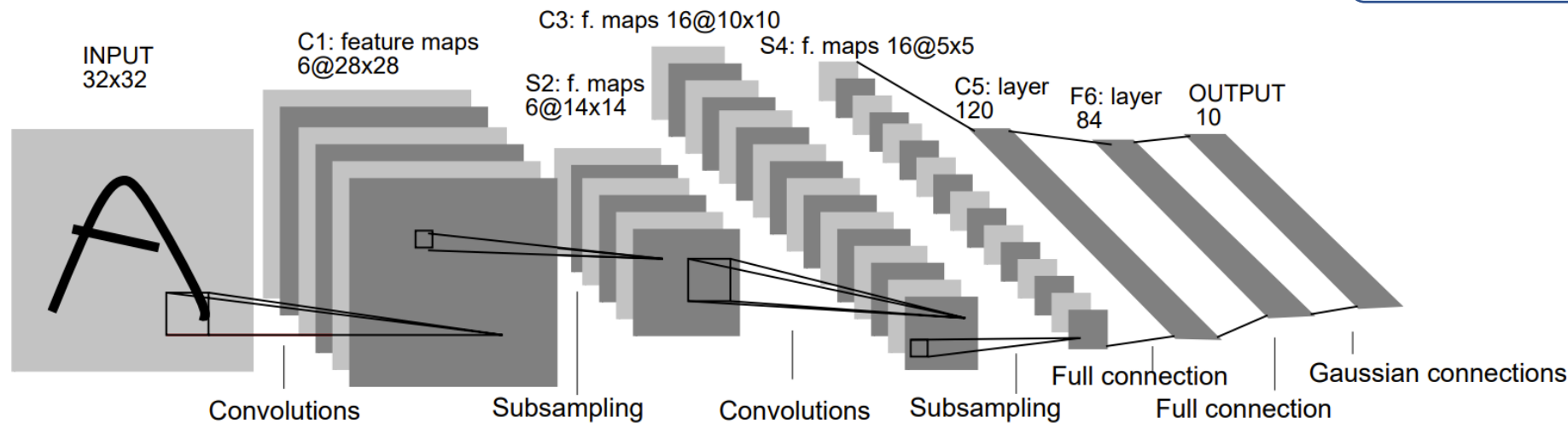


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- Input Layer, 3개의 Convolution Layer(C1, C2, C3), 2개의 Subsampling, 1층의 Full Connected Layer, Output Layer로 구성

- C1 레이어 (Convolution)

- 입력 영상(여기서는 32 x 32 사이즈의 이미지)을 6개의 5 x 5 필터와 컨볼루션 연산 수행
- 그 결과 6장의 28 x 28 특성 맵 취득

훈련해야 할 파라미터 개수: (가중치 \* 입력 맵 개수 + 바이어스) \* 특성 맵 개수 =  $(5*5 * 1 + 1) * 6 = 156$



- **S2 레이어 (Subsampling)**

- 6장의 28 x 28 특성 맵에 대해 서브샘플링 진행
- 2 x 2 필터를 stride 2로 설정해서 서브샘플링 하기때문에 결과적으로 28 x 28 사이즈의 특성 맵이 14 x 14 사이즈의 특성맵으로 축소됨
- 사용하는 서브샘플링 방법은 평균 풀링(average pooling)

훈련해야 할 파라미터 개수: (가중치 + 바이어스) \* 특성 맵 개수 =  $(1 + 1) * 6 = 12$

- 평균을 낸 후에 한 개의 훈련가능한 가중치(trainable weight)를 곱해주고
- 또 한 개의 훈련가능한 바이어스(trainable bias)를 더해 줌
- 그 값이 시그모이드 함수를 통해 활성화
  - 참고로 가중치와 바이어스는 시그모이드의 비활성도를 조절함

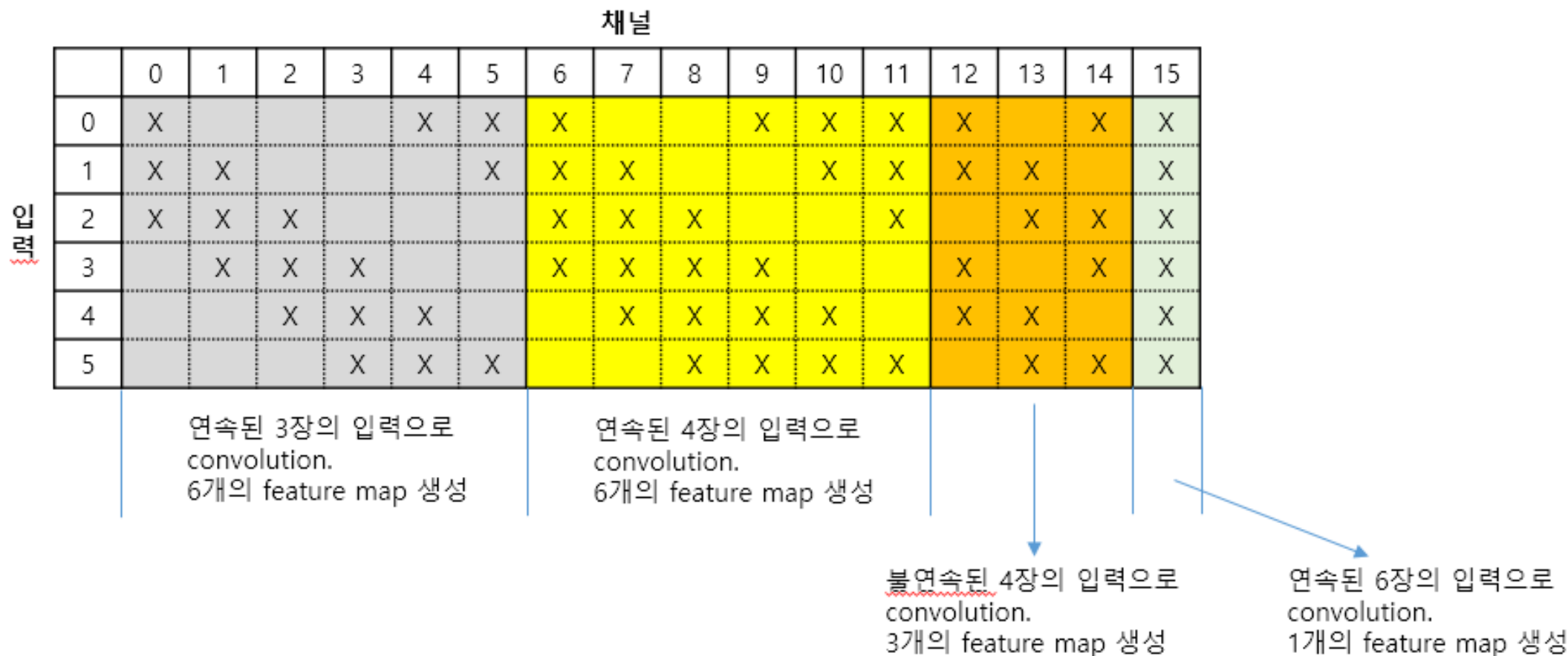
## • C3 레이어 (Convolution)

- 6장의 14 x 14 특성맵에 컨볼루션 연산을 수행해서 16장의 10 x 10 특성맵을 산출
  - 6장의 14 x 14 특성맵에서
    - 연속된 3장씩 모아서 5x5x3 크기 필터와 컨볼루션→(열0-5)→6장의 10x10 특성맵 산출
    - 연속된 4장씩 모아서 5x5x4 크기 필터와 컨볼루션→(열6-11)→6장의 10x10 특성맵 산출
    - 불연속한 4장씩 모아서 5x5x4 크기 필터와 컨볼루션→(열12-14)→3장의 10x10 특성맵 산출
    - 특성맵 모두로 5x5x6 크기 필터와 컨볼루션→(열15)→1장의 10x10 특성맵 산출
- 최종적으로 16장(6 + 6 + 3 + 1)의 10 x 10 특성맵 획득

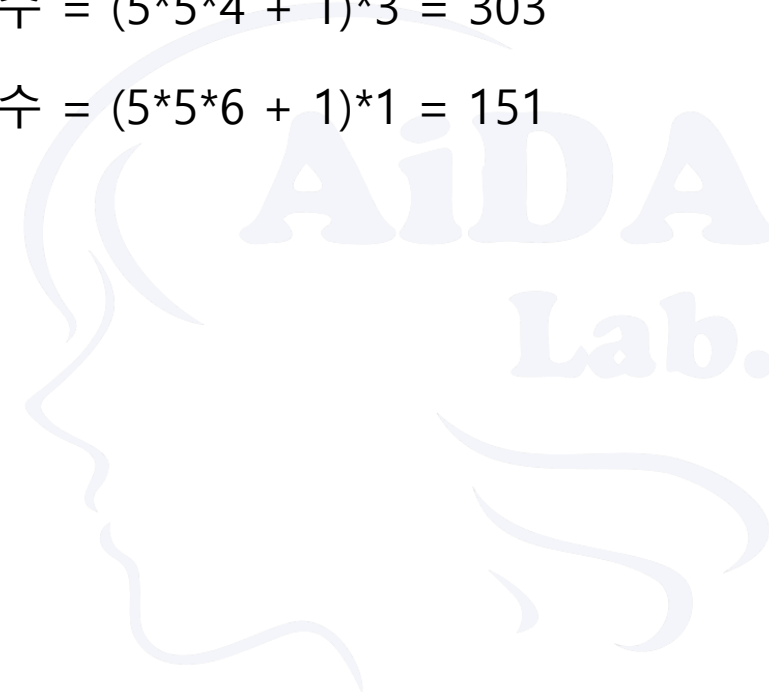
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

연속된 3장, 4장이나 불연속된 4장 등의 hyper parameter 값은 임의로 선택한 값 (논문에서 그렇게 밝히고 있음)

# 최초의 CNN 모델: LeNet



- 훈련해야 할 파라미터 수
  - 첫번째그룹=> (가중치\*입력맵개수+바이어스)\*특성맵 개수 =  $(5*5*3 + 1)*6 = 456$
  - 두번째그룹=> (가중치\*입력맵개수+바이어스)\*특성맵 개수 =  $(5*5*4 + 1)*6 = 606$
  - 세번째그룹=> (가중치\*입력맵개수+바이어스)\*특성맵 개수 =  $(5*5*4 + 1)*3 = 303$
  - 네번째그룹=> (가중치\*입력맵개수+바이어스)\*특성맵 개수 =  $(5*5*6 + 1)*1 = 151$
  - $456 + 606 + 303 + 151 = 1516$



- **S4 Layer (Subsampling)**

- 16장의 10 x 10 특성 맵에 대해서 서브샘플링을 진행해 16장의 5 x 5 특성 맵으로 축소 (2 x 2 필터, stride 2)

훈련해야 할 파라미터 개수: (가중치 + 바이어스) \* 특성 맵 개수 =  $(1 + 1) * 16 = 32$



- **C5 Layer (Convolution)**

- 16장의  $5 \times 5$  특성 맵을 120개의  $5 \times 5 \times 16$  사이즈의 필터와 컨볼루션 수행
- 결과적으로 120개의  $1 \times 1$  특성 맵 산출

훈련해야 할 파라미터 개수: (가중치 \* 입력 맵 개수 + 바이어스) \* 특성 맵 개수  
 $= (5*5*16 + 1) * 120 = 48120$



- **F6 Layer (Fully-Connected)**

- 84개의 유닛을 가진 피드포워드 신경망
- C5의 결과를 84개의 유닛에 연결
- LeNet에서는 아직 Backpropagation을 사용하지 않음

훈련해야 할 파라미터 개수: 연결개수  
= (입력개수 + 바이어스) \* 출력개수  
= (120 + 1) \* 84  
= 10164

출력 유닛이 84인 이유는...

아래의 ASCII set을 해석하기 적합한 형태로  
나와 주길 바라는 마음으로 설정했다고 함.

각각의 문자가 7x12 크기의 bitmap 이기 때문



- **Output Layer**

- 10개의 Euclidean radial basis function(RBF) 유닛들로 구성
- 각각 F6의 84개 유닛으로부터 입력을 받음
- 최종적으로 이미지가 속한 클래스 출력 → 10개의 출력에서 각각이 특정 이미지일 확률
- RBF에서는 학습할 때 역전파(Backpropagation)를 사용

LeNet-5를 제대로 가동하기 위해 훈련해야할 파라미터는  
총  $156 + 12 + 1516 + 32 + 48120 + 10164 = 60000$ 개

- AlexNet

- 2012년에 개최된 ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 대회의 우승을 차지한 CNN 구조

- Top 5 test error 기준 15.4%를 기록해 2위(26.2%)를 큰 폭으로 따돌리고 1위를 차지

Top 5 test error: 모델이 예측한 최상위 5개 범주 가운데 정답이 없는 경우의 오류율을 말함

- CNN의 부흥에 아주 큰 역할을 하였음

- 논문: "ImageNet Classification with Deep Convolutional Neural Networks"

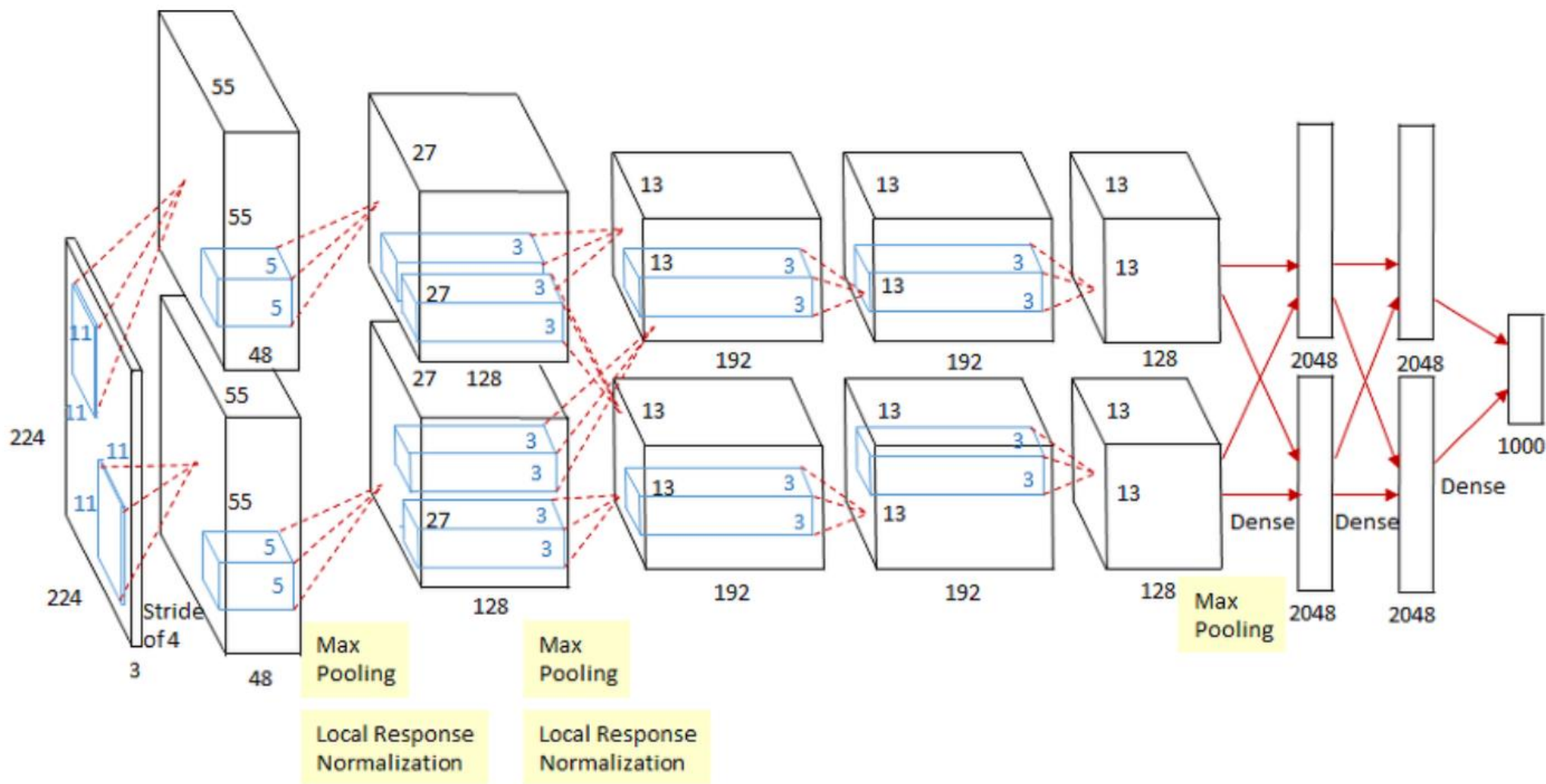
- 논문의 첫번째 저자가 Alex Krizhevsky이기 때문에 그의 이름을 따서 AlexNet 이라고 함

- AlexNet의 구조

- AlexNet의 기본구조는 LeNet-5와 크게 다르지 않음
- 2개의 GPU로 병렬연산을 수행하기 위해서 병렬적인 구조로 설계되었다는 점이 가장 큰 변화

- 8개의 레이어로 구성

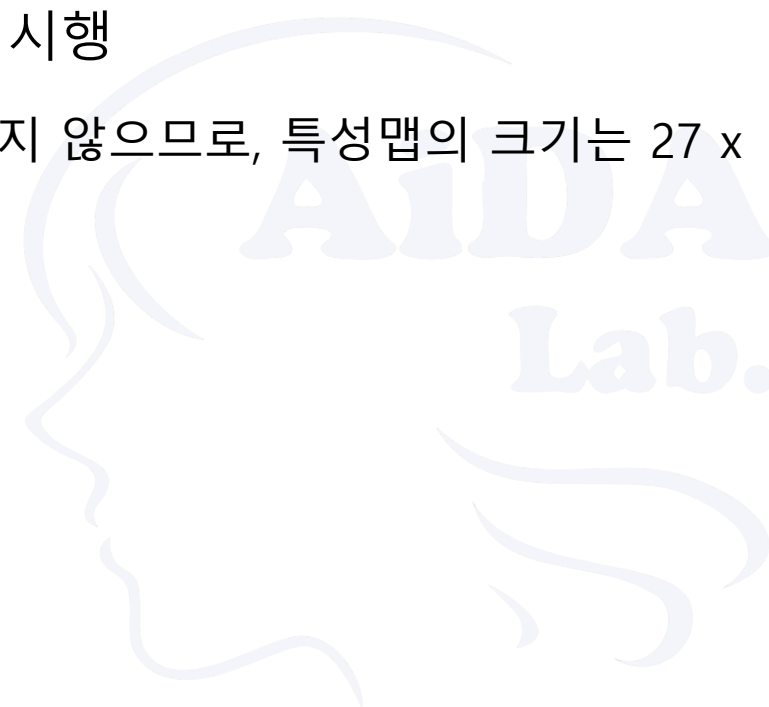
- 5개의 컨볼루션 레이어와 3개의 full-connected 레이어
- 2, 4, 5번 컨볼루션 레이어들은 전 단계의 같은 채널의 특성맵들과만 연결
- 3번 컨볼루션 레이어는 전 단계의 두 채널의 특성맵들과 모두 연결



- 첫번째 레이어(컨볼루션 레이어)

- 96개의  $11 \times 11 \times 3$  사이즈 필터커널로 입력 영상을 컨볼루션
- stride=4로 설정했으며 zero-padding은 사용하지 않음
  - zero-padding: 컨볼루션으로 인해 특성맵의 사이즈가 축소되는 것을 방지하기 위해, 또는 축소되는 정도를 줄이기 위해 영상의 가장자리 부분에 0을 추가하는 것
  - 결과적으로  $55 \times 55 \times 96$  특성맵(96장의  $55 \times 55$  사이즈 특성맵들) 산출됨
- ReLU 함수로 활성화

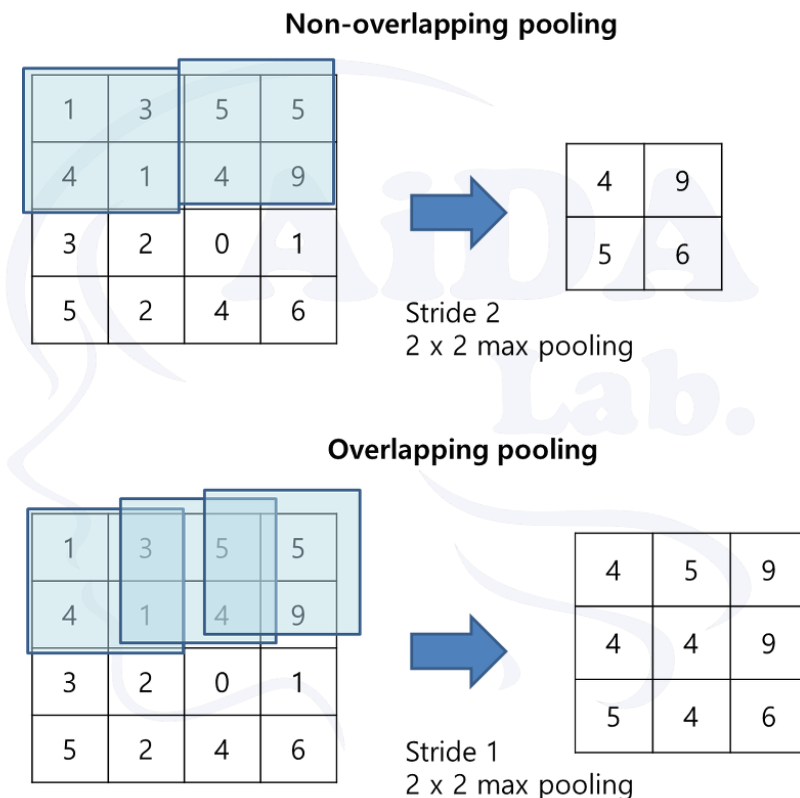
- 3 x 3 overlapping max pooling을 stride=2로 시행
  - 결과 27 x 27 x 96 특성맵을 가짐
- 수렴 속도를 높이기 위해 local response normalization 시행
  - local response normalization: 특성맵의 차원을 변화시키지 않으므로, 특성맵의 크기는 27 x 27 x 96으로 유지



- Overlapping Pooling

- CNN에서 pooling의 역할은 컨볼루션을 통해 얻은 특성맵의 크기를 줄이기 위함
- LeNet-5의 경우 Average Pooling이 사용된 반면, AlexNet에서는 Max Pooling 사용
- LeNet-5는 풀링 커널이 움직이는 Stride를 커널 크기보다 작게 하는 Overlapping Pooling 적용
  - LeNet-5는 Non-Overlapping Average Pooling 사용
  - AlexNet은 Overlapping Max Pooling 사용

overlapping 풀링을 하면 풀링 커널이 중첩되면서 지나가는 반면,  
non-overlapping 풀링을 하면 중첩없이 진행됨  
overlapping 풀링이 top-1, top-5 에러율을 줄이는데 좀 더 효과가 있다고 함





- local response normalization (LRN)
  - 신경생물학에서의 Lateral Inhibition : 활성화된 뉴런이 주변 이웃 뉴런들을 억누르는 현상
  - Lateral Inhibition 현상을 모델링한 것이 Local Response Normalization
  - 강하게 활성화된 뉴런의 주변 이웃들에 대해서 normalization을 실행
    - 주변에 비해 어떤 뉴런이 비교적 강하게 활성화되어 있다면, 그 뉴런의 반응은 더욱더 돋보이게 될 것
    - 반면 강하게 활성화된 뉴런 주변도 모두 강하게 활성화되어 있다면, LRN 이후에는 모두 값이 작아질 것

- 두번째 레이어(컨볼루션 레이어)

- 256개의  $5 \times 5 \times 48$  커널을 사용하여 전 단계의 특성맵을 컨볼루션
- stride는 =1로, zero-padding=2로 설정
  - $27 \times 27 \times 256$  특성맵(256장의  $27 \times 27$  사이즈 특성맵들) 산출
- ReLU 함수로 활성화
- $3 \times 3$  overlapping max pooling을 stride=2로 시행
  - $13 \times 13 \times 256$  특성맵 산출
- local response normalization이 시행
  - 특성맵의 크기는  $13 \times 13 \times 256$ 으로 그대로 유지



- 세번째 레이어(컨볼루션 레이어)

- 384개의  $3 \times 3 \times 256$  커널을 사용하여 전 단계의 특성맵을 컨볼루션
- stride와 zero-padding 모두 1로 설정
  - $13 \times 13 \times 384$  특성맵(384장의  $13 \times 13$  사이즈 특성맵들) 산출
- ReLU 함수로 활성화



- 네번째 레이어(컨볼루션 레이어)

- 384개의  $3 \times 3 \times 192$  커널을 사용하여 전 단계의 특성맵을 컨볼루션
- stride와 zero-padding 모두 1로 설정
  - $13 \times 13 \times 384$  특성맵(384장의  $13 \times 13$  사이즈 특성맵들) 산출
- ReLU 함수로 활성화



- 다섯번째 레이어(컨볼루션 레이어)

- 256개의  $3 \times 3 \times 192$  커널을 사용해서 전 단계의 특성맵을 컨볼루션
- stride와 zero-padding 모두 1로 설정
  - $13 \times 13 \times 256$  특성맵(256장의  $13 \times 13$  사이즈 특성맵들) 산출
- ReLU 함수로 활성화
- $3 \times 3$  overlapping max pooling을 stride=2로 시행
  - $6 \times 6 \times 256$  특성맵 산출



- **여섯번째 레이어(Fully connected layer)**

- 6 x 6 x 256 특성맵을 flatten으로 변환,  $6 \times 6 \times 256 = 9216$ 차원의 벡터로 만듦
- 생성된 벡터를 여섯번째 레이어의 4096개의 뉴런과 fully connected
- 결과를 ReLU 함수로 활성화

- **일곱번째 레이어(Fully connected layer)**

- 4096개의 뉴런으로 구성 → 전 단계의 4096개 뉴런과 fully connected
- 출력 값은 ReLU 함수로 활성화



- 여덟번째 레이어(Fully connected layer)

- 1000개의 뉴런으로 구성
- 전 단계의 4096개 뉴런과 fully connected
- 1000개 뉴런의 출력값에 softmax 함수를 적용해 1000개 클래스 각각에 속할 확률 계산

- 비교

- 약 6천만개의 파라미터가 훈련되어야 함 (LeNet-5는 약 6만개)
- 컴퓨팅 기술도 좋아졌고, 훈련시간을 줄이기 위한 방법들도 사용 → 훈련 가능
  - 예전 기술 같으면 상상도 못할 연산량

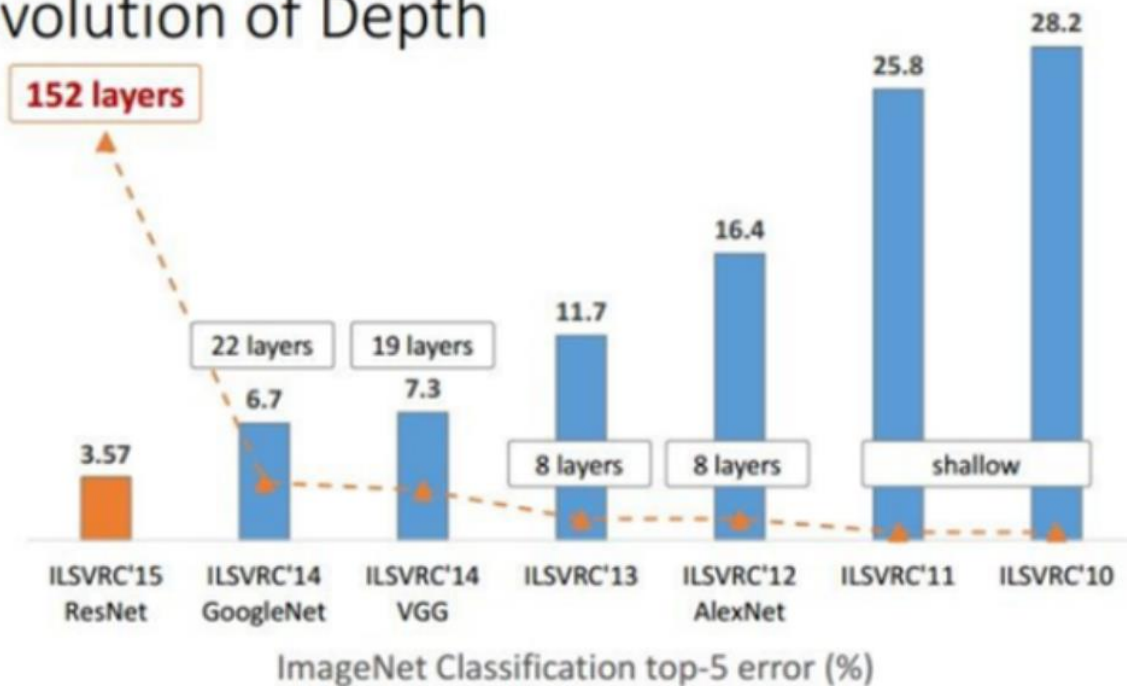
- VGGNet

- 옥스포드 대학의 연구팀 VGG에 의해 개발되었으며 2014년 이미지넷 이미지 인식 대회에서 준우승을 한 모델
- 16개 또는 19개의 층으로 구성됨 (VGG-16, VGG-19)
- 논문: "Very deep convolutional networks for large-scale image recognition"



## • 깊이의 혁명

### Revolution of Depth



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

역사적으로 봤을 때,  
VGGNet 모델부터 시작해서 네트워크의 깊이가 확 깊어 졌다.

- 2012년, 2013년 우승 모델들은 8개의 층으로 구성
- 2014년의 VGGNet(VGG19)는 19층으로 구성됨
- GoogLeNet은 22층으로 구성됨
- 2015년에는 152개의 층으로 구성된 ResNet이 제안됨
- 네트워크가 깊어질 수록 성능이 좋아졌음을 그림을 통해 확인 가능
- VGGNet은 사용하기 쉬운 구조와 좋은 성능 덕분에 그 대회에서 우승을 거둔 조금 더 복잡한 형태의 GoogLeNet보다 더 인기를 얻었다.

- VGGNet 연구의 핵심

- 네트워크의 깊이를 깊게 만드는 것이 성능에 어떤 영향을 미치는지를 확인하고자 한 것
- 깊이의 영향만을 최대한 확인하고자 컨볼루션 필터 커널 크기는 가장 작은 3x3으로 고정
  - 필터 커널 크기가 크면 그만큼 이미지의 사이즈가 금방 축소되기 때문에 네트워크의 깊이를 충분히 깊게 만들기는 불가능함

- VGG 연구팀은 총 6개의 구조 (A, A-LRN, B, C, D, E)를 만들어 성능을 비교함

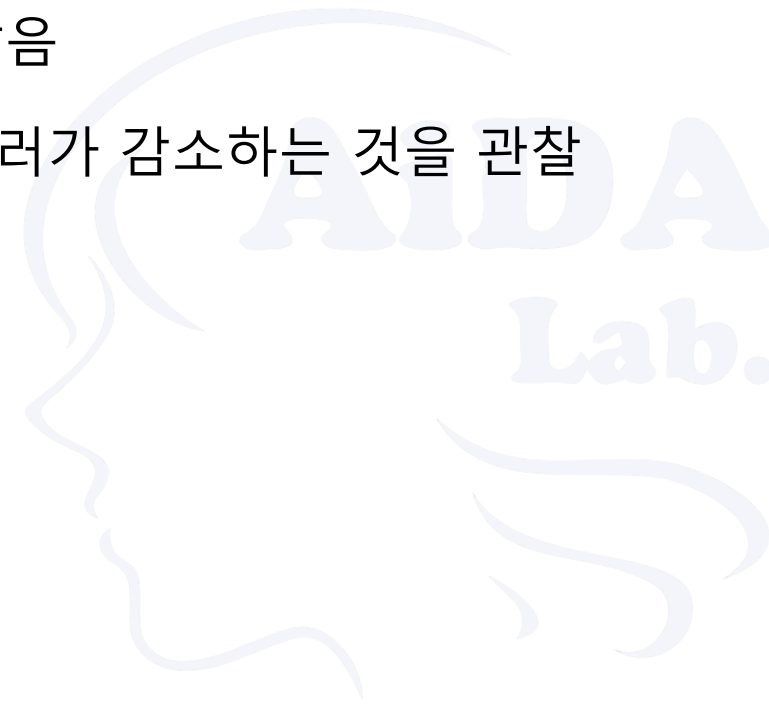
- 여러 구조를 만든 이유:
  - 기본적으로 깊이에 따른 성능 변화를 비교하기 위함
  - D 구조: VGG16
  - E 구조: VGG19

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGGNet 구조 설명 표 [출처: original 논문]

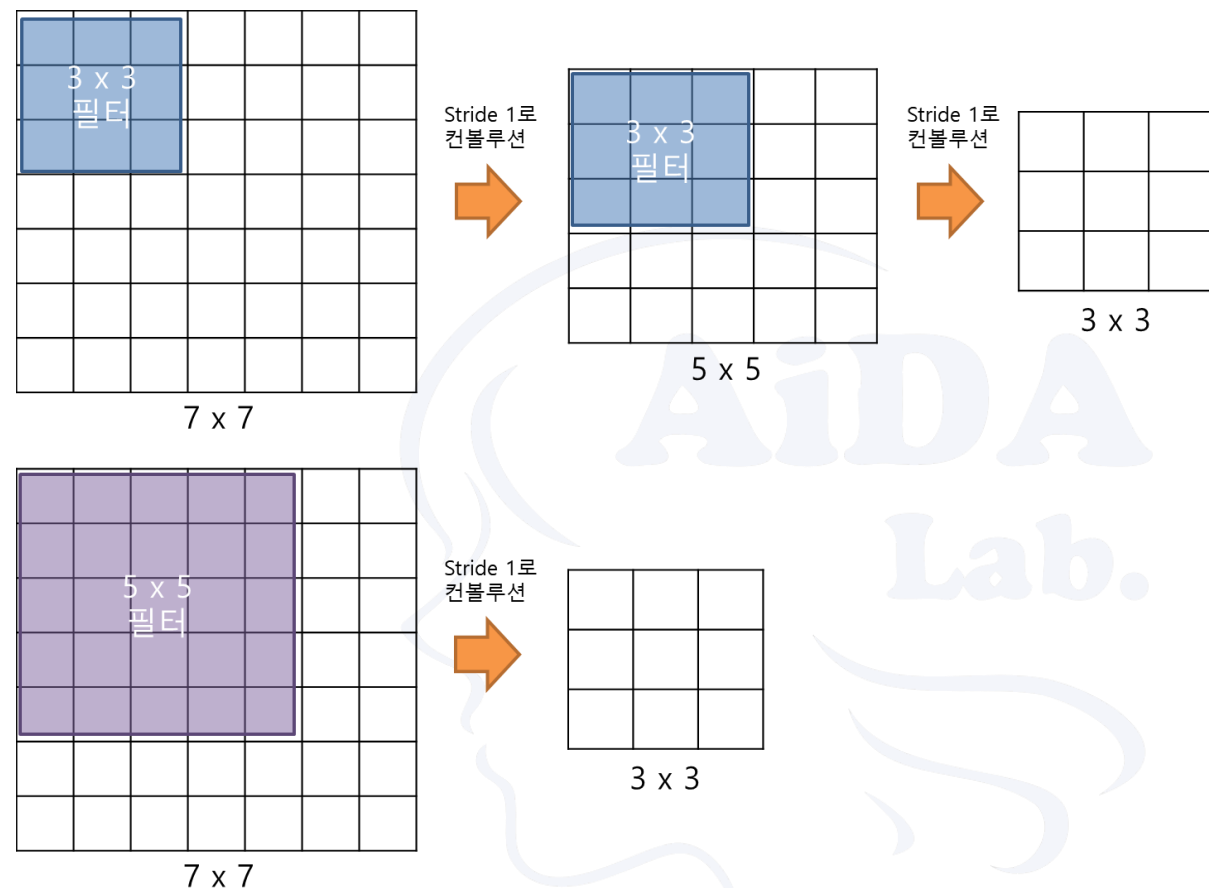
## • 연구 과정

- AlexNet 등이 사용한 Local Response Normalization(LRN)이
- A 구조 및 A-LRN 구조와의 성능 비교 결과, 성능 향상에 별로 효과가 없다는 것을 실험을 통해 확인 → B, C, D, E 구조에는 LRN을 적용하지 않음
- 깊이가 11층, 13층, 16층, 19층으로 깊어지면서 분류 에러가 감소하는 것을 관찰  
→ 깊어질수록 성능이 좋아진다



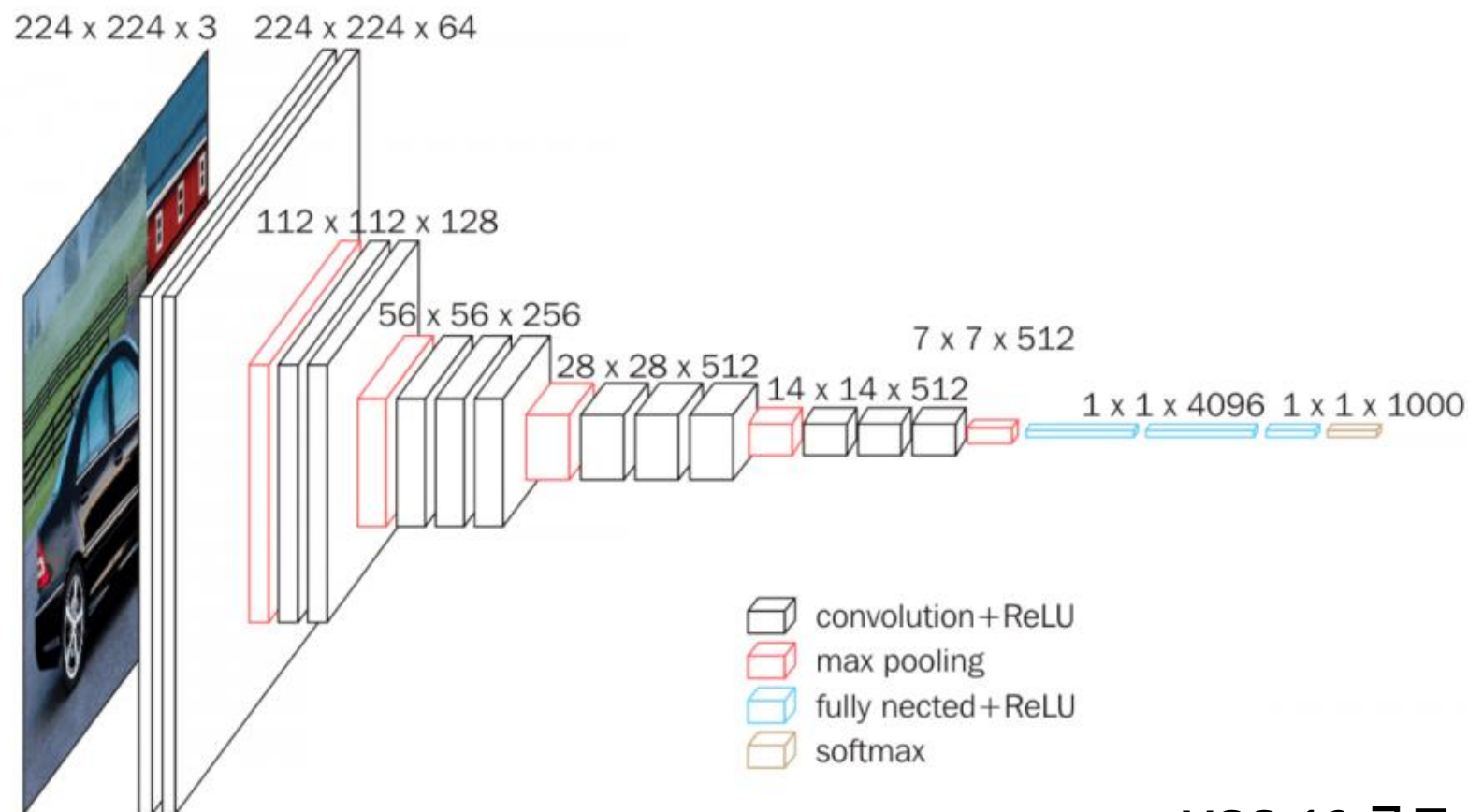
- VGGNet 구조 분석에 앞서..

- 3 x 3 필터로 두 차례 컨볼루션을 하는 것과 5 x 5 필터로 한 번 컨볼루션을 하는 것은 결과적으로 동일한 사이즈의 특성맵을 산출
- 3 x 3 필터로 세 차례 컨볼루션 하는 것은 7 x 7 필터로 한 번 컨볼루션 하는 것과 대응됨



- 3x3 필터로 세 차례 컨볼루션을 하는 것이 7x7 필터로 한 번 컨볼루션하는 것보다 나은 점은 무엇일까?
  - 가중치 또는 파라미터의 갯수의 차이
    - 3 x 3 필터가 3개면 총 27개의 가중치를 가지는 반면 7 x 7 필터는 49개의 가중치를 가짐
    - CNN에서 가중치는 모두 훈련이 필요한 것들이므로, 가중치가 적다는 것은 그만큼 훈련시켜야 할 것의 갯수가 작아짐을 의미함
    - 따라서 학습의 속도가 빨라지며, 동시에 층의 갯수가 늘어나면서 특성에 비선형성을 더욱 강화시키기 때문에 특성이 점점 더 유용해짐

## • VGGNet 구조



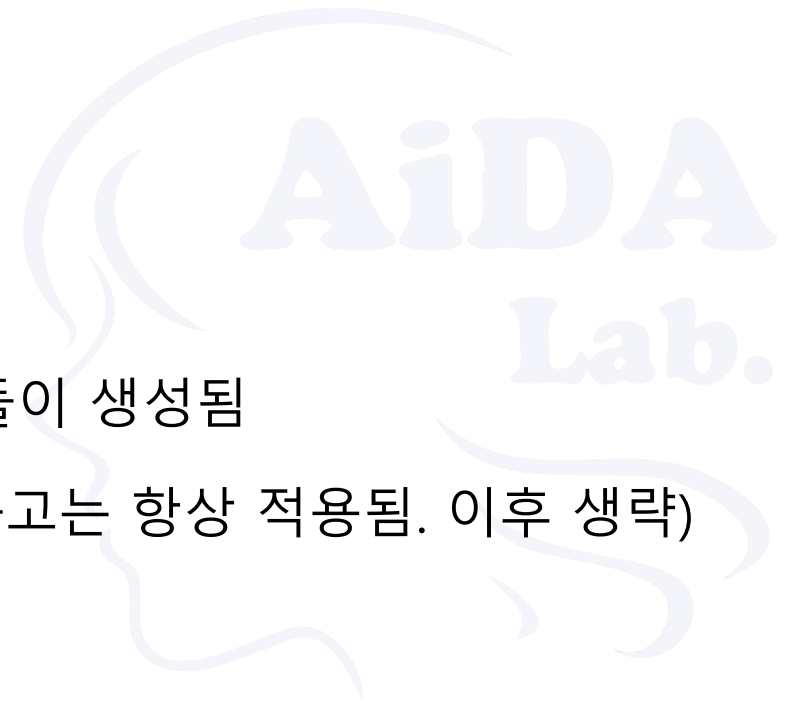
VGG-16 구조

- 입력

- 224 x 224 x 3 이미지(224 x 224 RGB 이미지)를 입력

- 1층(conv1\_1)

- 64개의 3 x 3 x 3 필터 커널로 입력 이미지를 컨볼루션
- zero padding=1, stride=1로 설정
- 결과적으로 64장의 224 x 224 특성맵(224 x 224 x 64)들이 생성됨
- ReLU 함수로 활성화(ReLU함수는 마지막 16층을 제외하고는 항상 적용됨. 이후 생략)





## • 2층(conv1\_2)

- 64개의  $3 \times 3 \times 64$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 64장의  $224 \times 224$  특성맵들( $224 \times 224 \times 64$ )이 산출됨
- $2 \times 2$  Max Pooling을 stride=2로 적용 → 특성맵의 사이즈를  $112 \times 112 \times 64$ 로 축소

\*conv1\_1, conv1\_2와 conv2\_1, conv2\_2등으로 표현한 이유는 해상도를 줄여주는 최대 풀링 전까지의 층등을 한 모듈로 볼 수 있기 때문

## • 3층(conv2\_1)

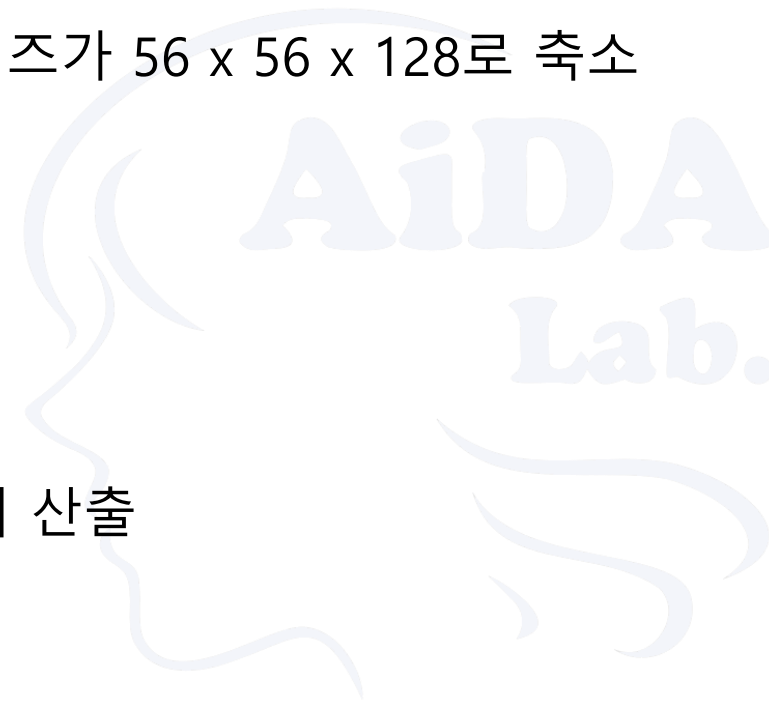
- 128개의  $3 \times 3 \times 64$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 128장의  $112 \times 112$  특성맵들( $112 \times 112 \times 128$ )이 산출

- **4층(conv2\_2)**

- 128개의  $3 \times 3 \times 128$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 128장의  $112 \times 112$  특성맵들( $112 \times 112 \times 128$ )이 산출
- $2 \times 2$  Max Pooling을 stride=2로 적용  $\rightarrow$  특성맵의 사이즈가  $56 \times 56 \times 128$ 로 축소

- **5층(conv3\_1)**

- 256개의  $3 \times 3 \times 128$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 256장의  $56 \times 56$  특성맵들( $56 \times 56 \times 256$ )이 산출



- **6층(conv3\_2)**

- 256개의  $3 \times 3 \times 256$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 256장의  $56 \times 56$  특성맵들( $56 \times 56 \times 256$ )이 산출

- **7층(conv3\_3)**

- 256개의  $3 \times 3 \times 256$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 256장의  $56 \times 56$  특성맵들( $56 \times 56 \times 256$ )이 산출
- $2 \times 2$  Max Pooling을 stride=2로 적용 → 특성맵의 사이즈가  $28 \times 28 \times 256$ 으로 축소

- **8층(conv4\_1)**

- 512개의  $3 \times 3 \times 256$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 512장의  $28 \times 28$  특성맵들( $28 \times 28 \times 512$ )이 산출

- **9층(conv4\_2)**

- 512개의  $3 \times 3 \times 512$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 512장의  $28 \times 28$  특성맵들( $28 \times 28 \times 512$ )이 산출

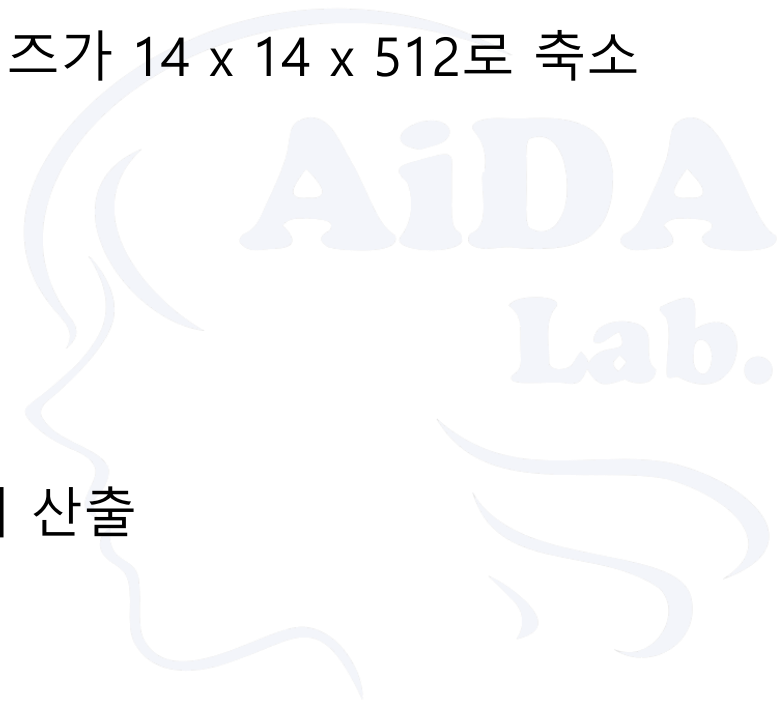


- **10층(conv4\_3)**

- 512개의  $3 \times 3 \times 512$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 512장의  $28 \times 28$  특성맵들( $28 \times 28 \times 512$ )이 산출
- $2 \times 2$  Max Pooling을 stride=2로 적용 → 특성맵의 사이즈가  $14 \times 14 \times 512$ 로 축소

- **11층(conv5\_1)**

- 512개의  $3 \times 3 \times 512$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 512장의  $14 \times 14$  특성맵들( $14 \times 14 \times 512$ )이 산출

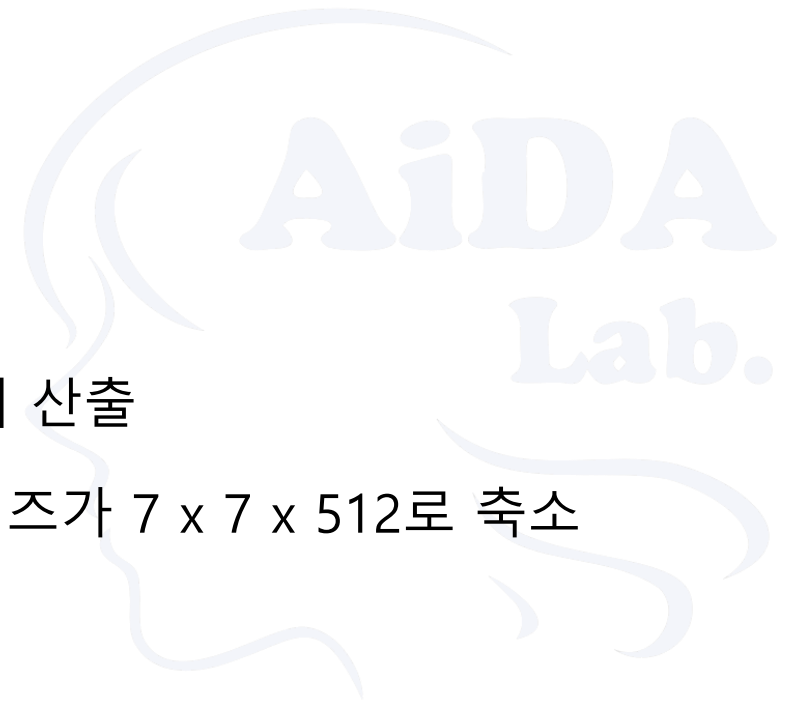


- **12층(conv5\_2)**

- 512개의  $3 \times 3 \times 512$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 512장의  $14 \times 14$  특성맵들( $14 \times 14 \times 512$ )이 산출

- **13층(conv5-3)**

- 512개의  $3 \times 3 \times 512$  필터 커널로 특성맵을 컨볼루션
- 결과적으로 512장의  $14 \times 14$  특성맵들( $14 \times 14 \times 512$ )이 산출
- $2 \times 2$  Max Pooling을 stride=2로 적용 → 특성맵의 사이즈가  $7 \times 7 \times 512$ 로 축소



- **14층(fc1)**

- 7 x 7 x 512의 특성맵을 flatten
  - flatten이라는 것은 전 층의 출력을 받아서 단순히 1차원의 벡터로 펼쳐주는 것을 의미
- 결과적으로 7 x 7 x 512 = 25088개의 뉴런이 되고, fc1층의 4096개의 뉴런과 fully connected 됨. 훈련 시 dropout이 적용됨

- **15층(fc2)**

- 4096개의 뉴런으로 구성
- fc1층의 4096개의 뉴런과 fully connected 됨. 훈련 시 dropout이 적용됨

- 16층(fc3)

- 1000개의 뉴런으로 구성
  - 1000개의 뉴런으로 구성 → 1000개의 클래스로 분류하는 목적으로 만들어진 네트워크
- fc2층의 4096개의 뉴런과 fully connected 됨
- 출력값들은 softmax 함수로 활성화





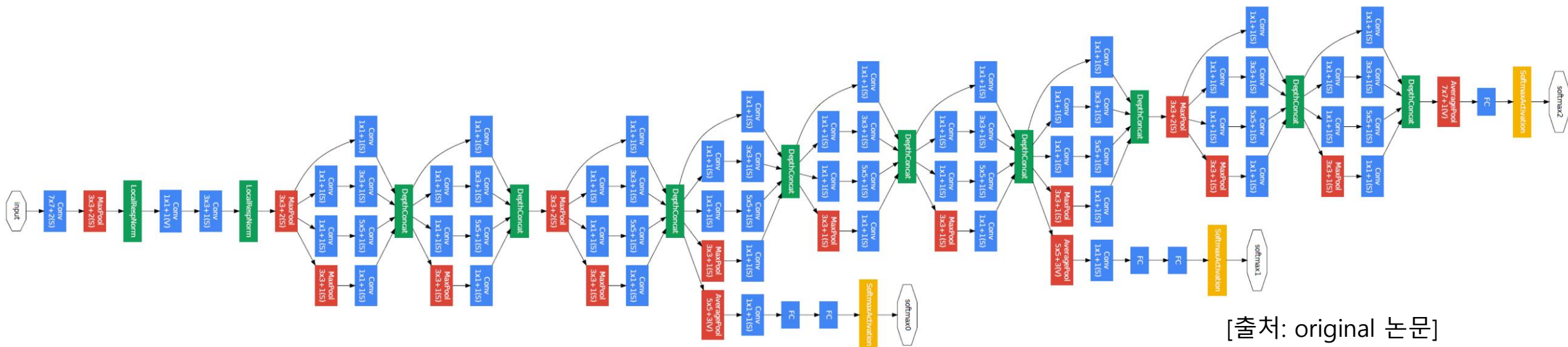
- GoogLeNet (Inception v1)

- 2014년 이미지넷 이미지 인식 대회(ILSVRC)에서 VGGNet(VGG19)을 이기고 우승을 차지한 알고리즘
- VGG19보다 좀 더 깊은 22층으로 구성됨
- 논문 : "Going Deeper with Convolutions"
- 구글이 이 알고리즘 개발에 참여



# GoogLeNet (Inception v1)

- GoogLeNet 구조



- 22개 층으로 구성 (파란색 블록의 층수를 세보면 22개 층임)

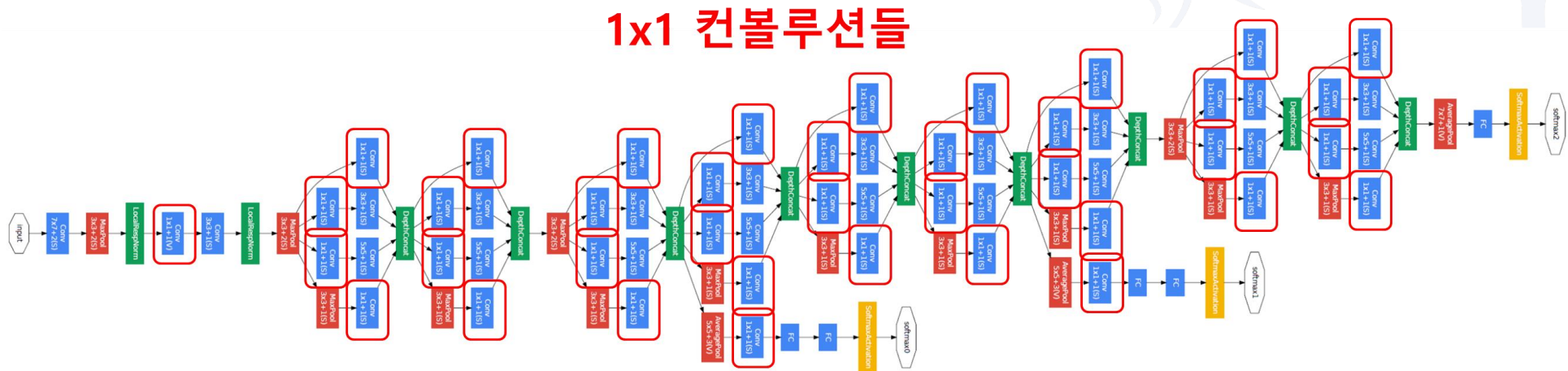
# GoogLeNet (Inception v1)

- 1 x 1 컨볼루션

- 먼저 주목해야할 것은 1 x 1 사이즈의 필터로 컨볼루션 해 주는 것
  - 구조도를 보면 곳곳에 1 x 1 컨볼루션 연산이 있음을 확인할 수 있음

- 1 x 1 컨볼루션의 의미

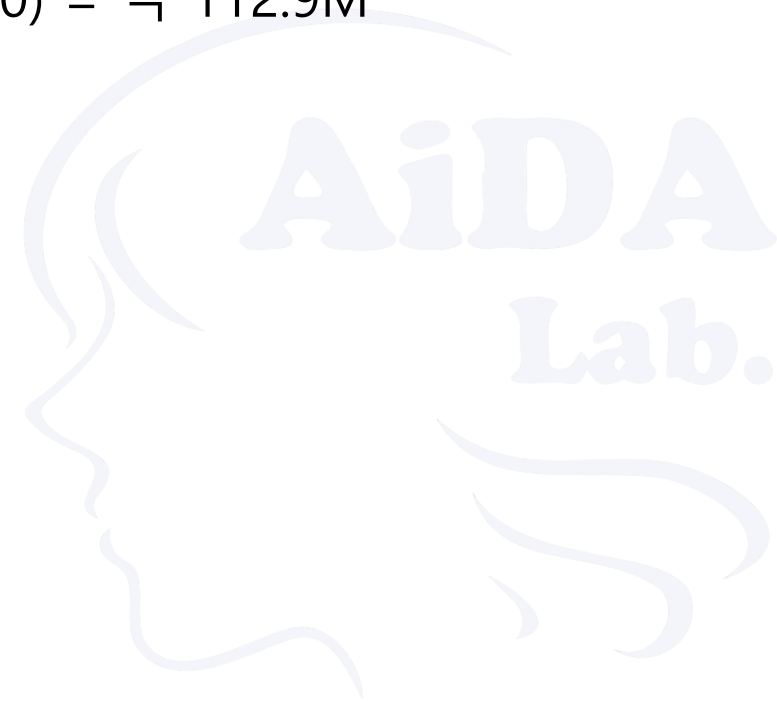
- GoogLeNet에서 1 x 1 컨볼루션은 특성맵의 갯수를 줄이는 목적으로 사용됨
- 특성맵의 갯수가 줄어들면 그만큼 연산량이 줄어듦



# GoogLeNet (Inception v1)



- 480장의  $14 \times 14$  사이즈의 특성맵( $14 \times 14 \times 480$ )이 있다고 가정
  - 48개의  $5 \times 5 \times 480$ 의 필터커널로 컨볼루션
  - 48장의  $14 \times 14$ 의 특성맵( $14 \times 14 \times 48$ ) 생성 (zero padding=2, stride=1로 가정)
  - 이때 필요한 연산횟수는  $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = \text{약 } 112.9\text{M}$

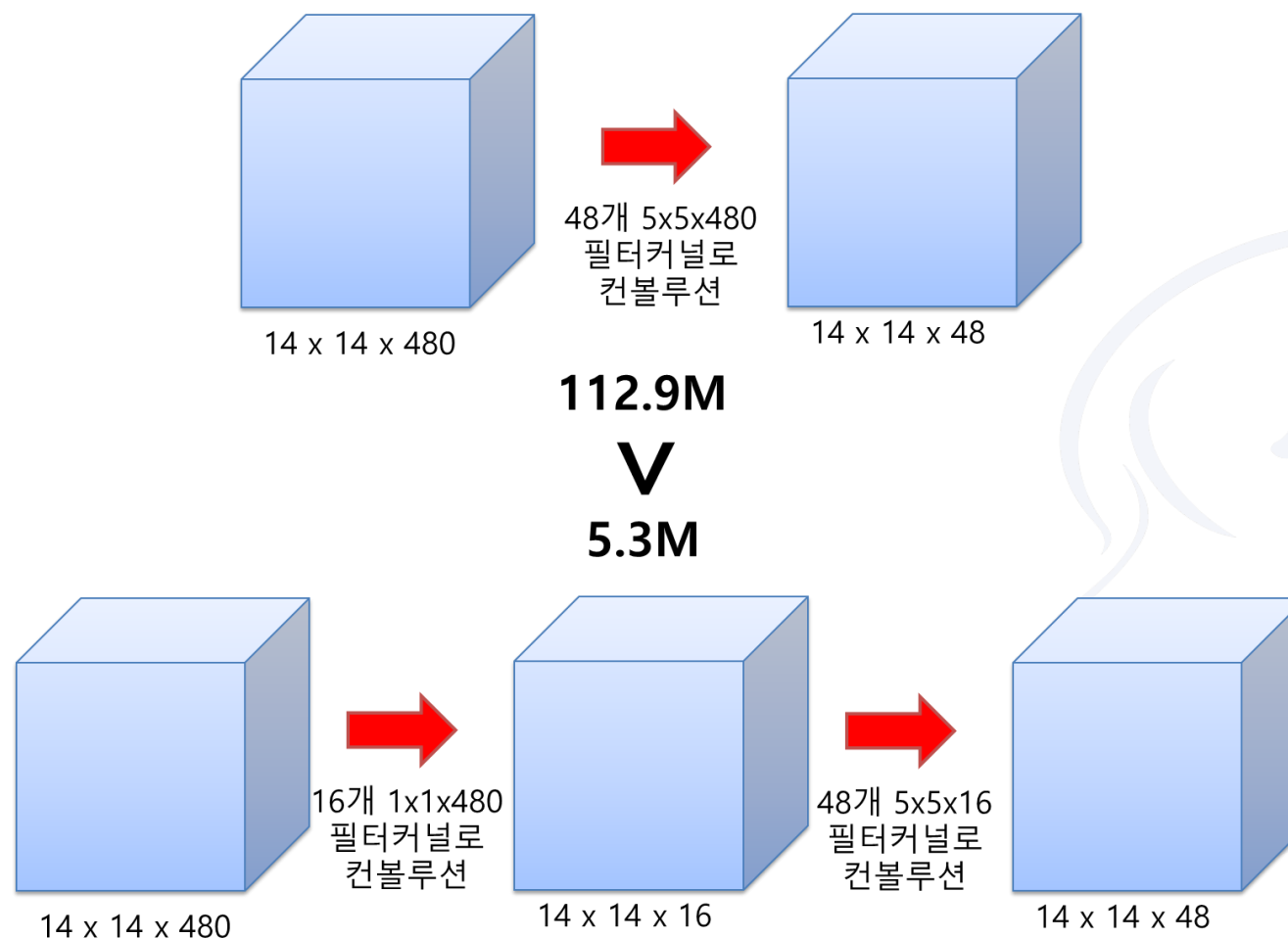


# GoogLeNet (Inception v1)

- 480장의  $14 \times 14$  특성맵( $14 \times 14 \times 480$ )
  - 16개의  $1 \times 1 \times 480$ 의 필터커널로 컨볼루션
  - 16장의  $14 \times 14$ 의 특성맵( $14 \times 14 \times 16$ ) 생성
  - 이 특성맵을 48개의  $5 \times 5 \times 16$ 의 필터커널로 컨볼루션
  - 48장의  $14 \times 14$ 의 특성맵( $14 \times 14 \times 48$ )이 생성  
( $1 \times 1$  컨볼루션이 없을 때와 결과적으로 산출된 특성맵의 크기와 깊이가 같다)
  - 이때 필요한 연산횟수는?  
$$(14 \times 14 \times 16) \times (1 \times 1 \times 480) + (14 \times 14 \times 48) \times (5 \times 5 \times 16) = \text{약 } 5.3\text{M}$$
- 112.9M에 비해 훨씬 적은 연산량 확인
- 연산량을 줄일 수 있다 → 네트워크를 더 깊이 만들 수 있게 해 준다는 점에서 중요

# GoogLeNet (Inception v1)

- 1 x 1 컨볼루션의 효과

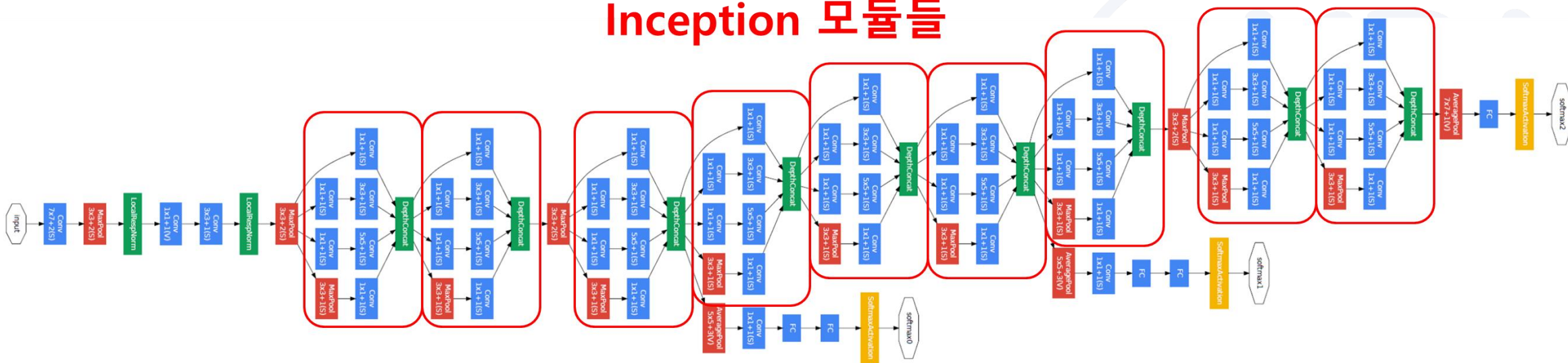


# GoogLeNet (Inception v1)

- Inception 모듈

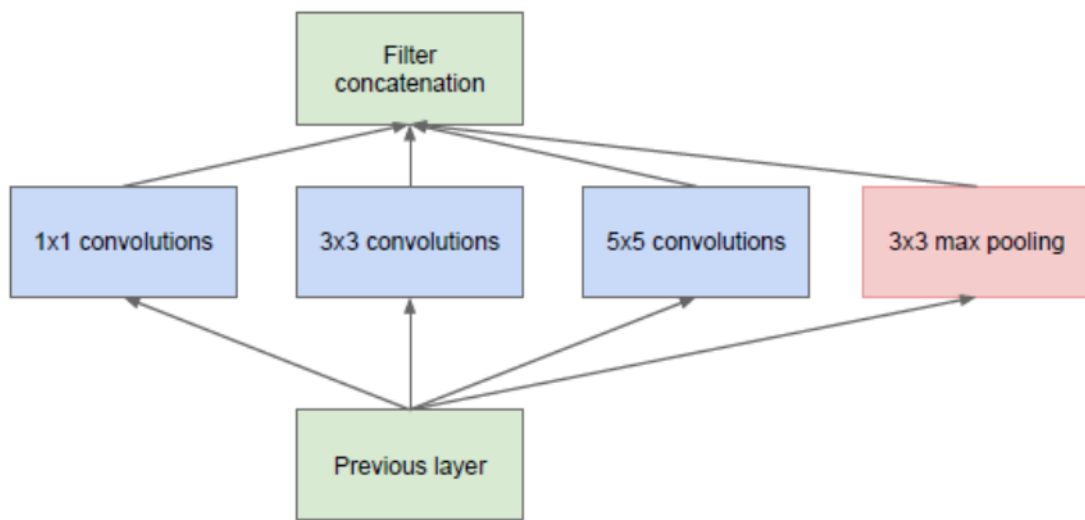
- GoogLeNet의 핵심
- GoogLeNet은 총 9개의 인셉션 모듈을 포함

## Inception 모듈들

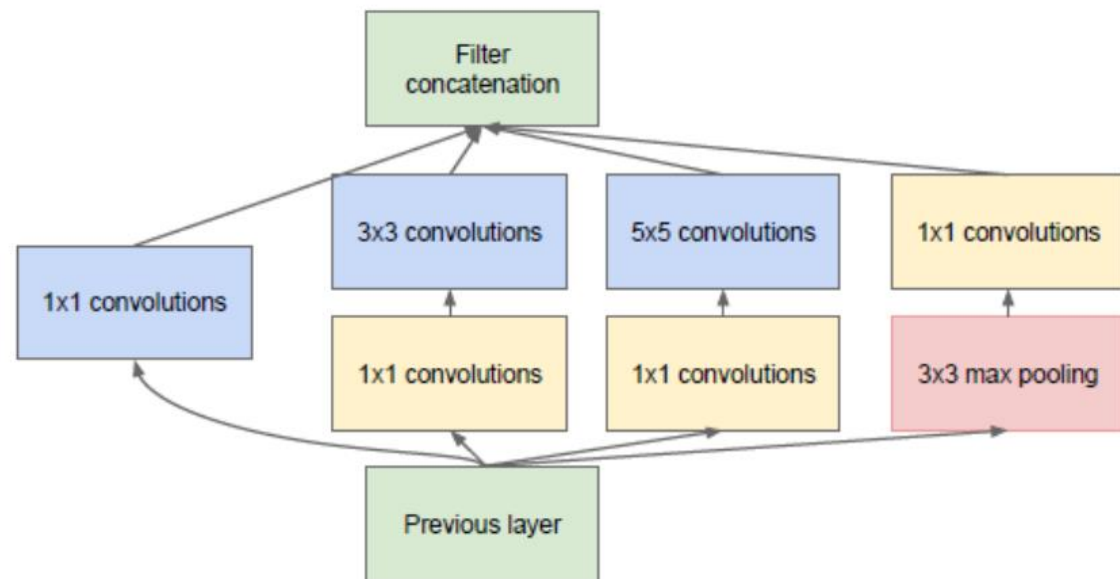


# GoogLeNet (Inception v1)

- 인셉션 모듈
  - GoogLeNet에 실제로 사용된 모듈은 1x1 컨볼루션이 포함된 (b) 모델



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction



- 1x1 컨볼루션은 특성맵의 장수를 줄여주는 역할
- 나이브(naive) 버전(노란색 블록으로 표현된 1x1 컨볼루션을 제외한 버전)
  - 이전 층에서 생성된 특성맵을 1x1 컨볼루션, 3x3 컨볼루션, 5x5 컨볼루션, 3x3 Max Pooling 을 적용하여 얻은 특성맵들을 모두 함께 쌓아줌 → 더 다양한 종류의 특성 도출
    - AlexNet, VGGNet 등의 이전 CNN 모델들은 한 층에서 동일한 사이즈의 필터커널을 이용해서 컨볼루션을 해줬던 것과 차이가 있음
- 여기에 1x1 컨볼루션이 포함되었으니 당연히 연산량은 많이 줄어들었을 것임

## • Global Average Pooling

AlexNet, VGGNet 등에서는 FC 층들이 망의 후반부에 연결됨. 그러나 GoogLeNet은 FC 방식 대신에 Global Average Pooling 방식을 사용

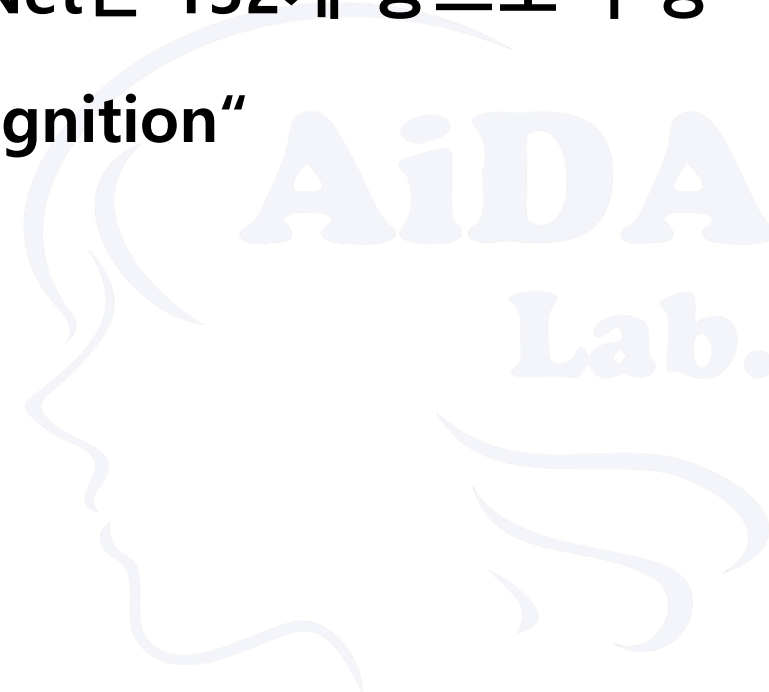
- 전 층에서 산출된 특성맵들을 각각 평균낸 것을 이어서 1차원 벡터를 만들어주는 것
- 1차원 벡터를 만들어야 최종적으로 이미지 분류를 위한 softmax 층을 연결 가능
- 만약 전 층에서 1024장의  $7 \times 7$ 의 특성맵이 생성되었다면  
→ 1024장의  $7 \times 7$  특성맵 각각의 평균을 계산하여 얻은 1024개의 값을 하나의 벡터로 연결함
- 장점: 가중치의 갯수를 상당히 많이 없애줌
  - FC 방식을 사용한다면 훈련이 필요한 가중치의 갯수가  $7 \times 7 \times 1024 \times 1024 = 51.3M$
  - global average pooling을 사용하면 가중치가 단 한개도 필요하지 않음

- **Auxiliary Classifier (보조 분류기)**

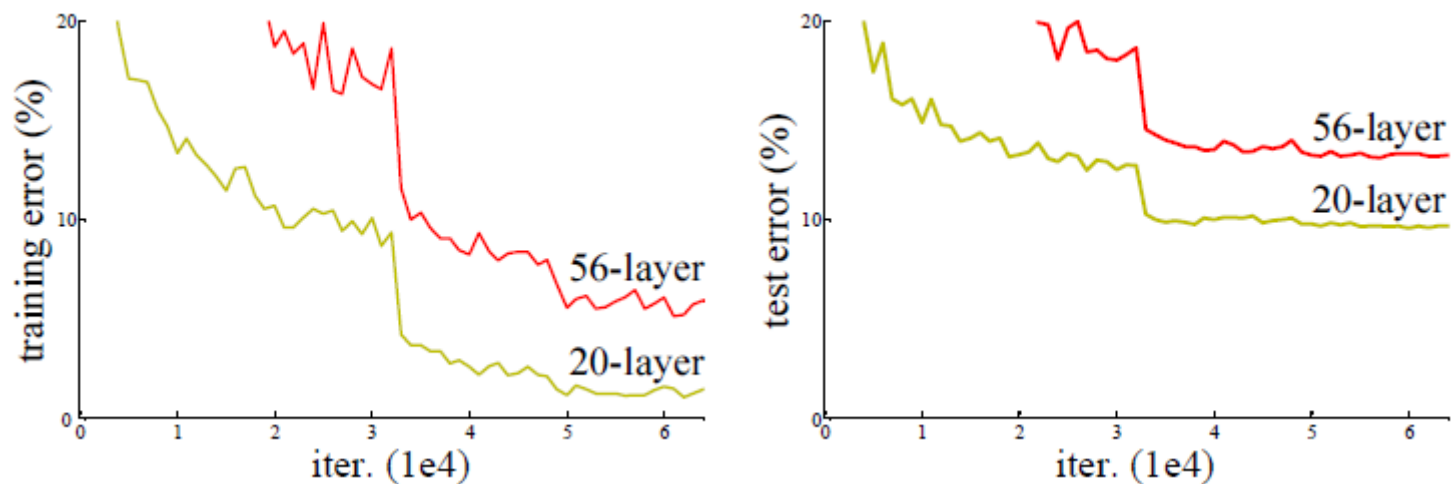
- 네트워크의 깊이가 깊어지면 깊어질수록 Vanishing Gradient 문제를 피하기 어려움
  - 가중치를 훈련하는 과정에 역전파(back propagation)를 주로 활용
  - 역전파과정에서 가중치 업데이트에 사용되는 gradient가 점점 작아져서 0이 되어버리는 문제
  - 네트워크 내의 가중치들이 제대로 훈련되지 않게 됨
- Vanishing Gradient 문제를 극복하기 위해서 네트워크 중간에 두 개의 보조 분류기 (auxiliary classifier)를 추가함
- 보조 분류기의 구성
  - 5 x 5 Max Pooling(stride=3) → 128개 1x1 필터 커널로 컨볼루션 → 1024 FC 층 → 1000 FC 층 → softmax
  - 보조 분류기들은 훈련시에만 활용되고 사용할 때는 제거함

- ResNet

- 마이크로소프트에서 개발한 알고리즘
- 2014년 GoogLeNet은 22개 층으로 구성 → ResNet은 152개 층으로 구성
- 논문: "Deep Residual Learning for Image Recognition"



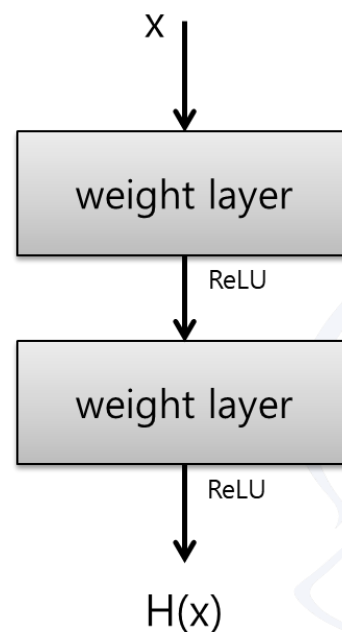
- 연구의 초점: 망을 깊게하면 무조건 성능이 좋아질까?
  - 20층의 컨볼루션 레이어, 56층의 Fully Connected 레이어를 만든 후 성능 테스트



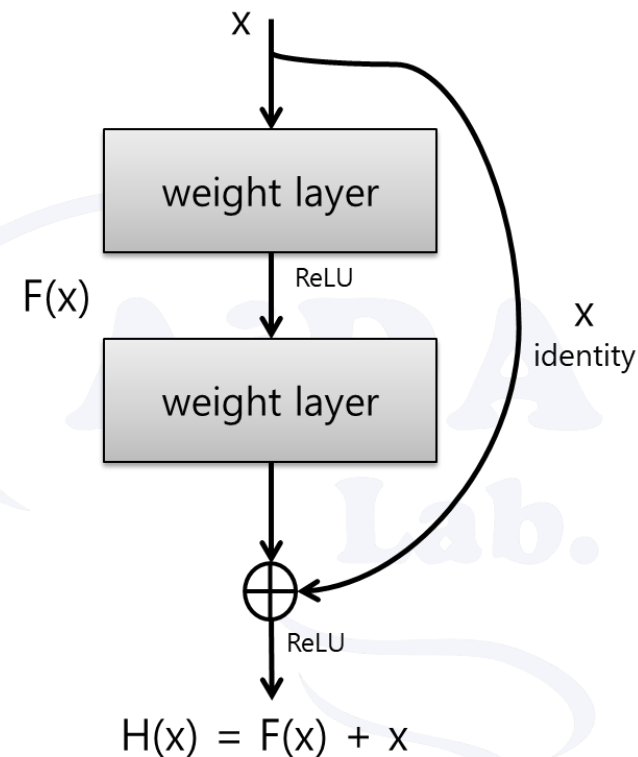
- 더 깊은 구조를 가진 56층의 네트워크가 20층의 네트워크보다 더 나쁜 성능을 보임
- 기존 방식으로 망을 무조건 깊게 한다고 되는 일이 아님
  - 뭔가 새로운 방법이 있어야 망을 깊게 만드는 효과를 볼 수 있다는 것을 확인

- **Residual Block (잔차 블록)**

- 기존 방식과의 차이는?
  - 입력값을 출력값에 더해줄 수 있도록 지름길(shortcut)을 하나 만들어준 것
- 기존의 신경망은 입력값  $x$ 를 타겟값  $y$ 로 매핑하는 함수  $H(x)$ 를 얻는 것이 목적
- ResNet은  $F(x) + x$ 를 최소화하는 것이 목적

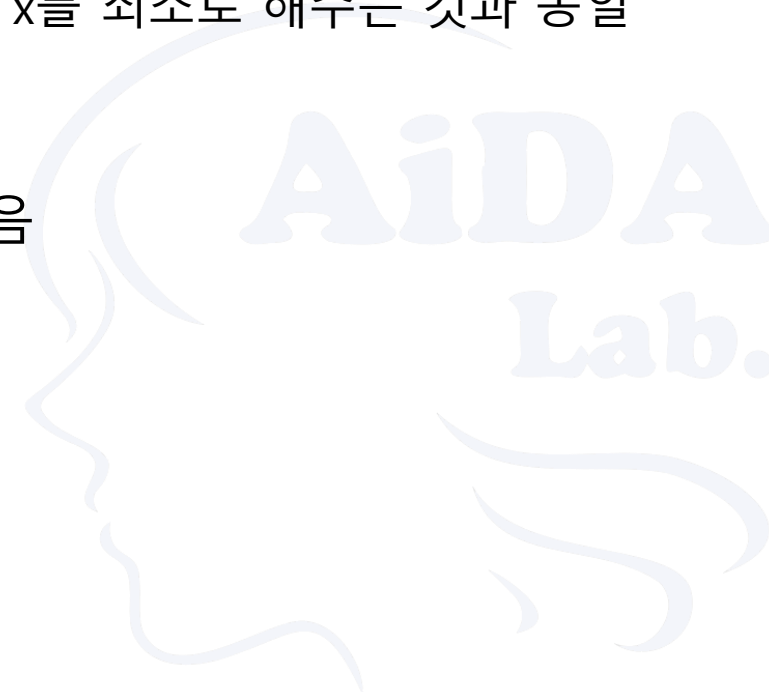


기존 방식



Residual block

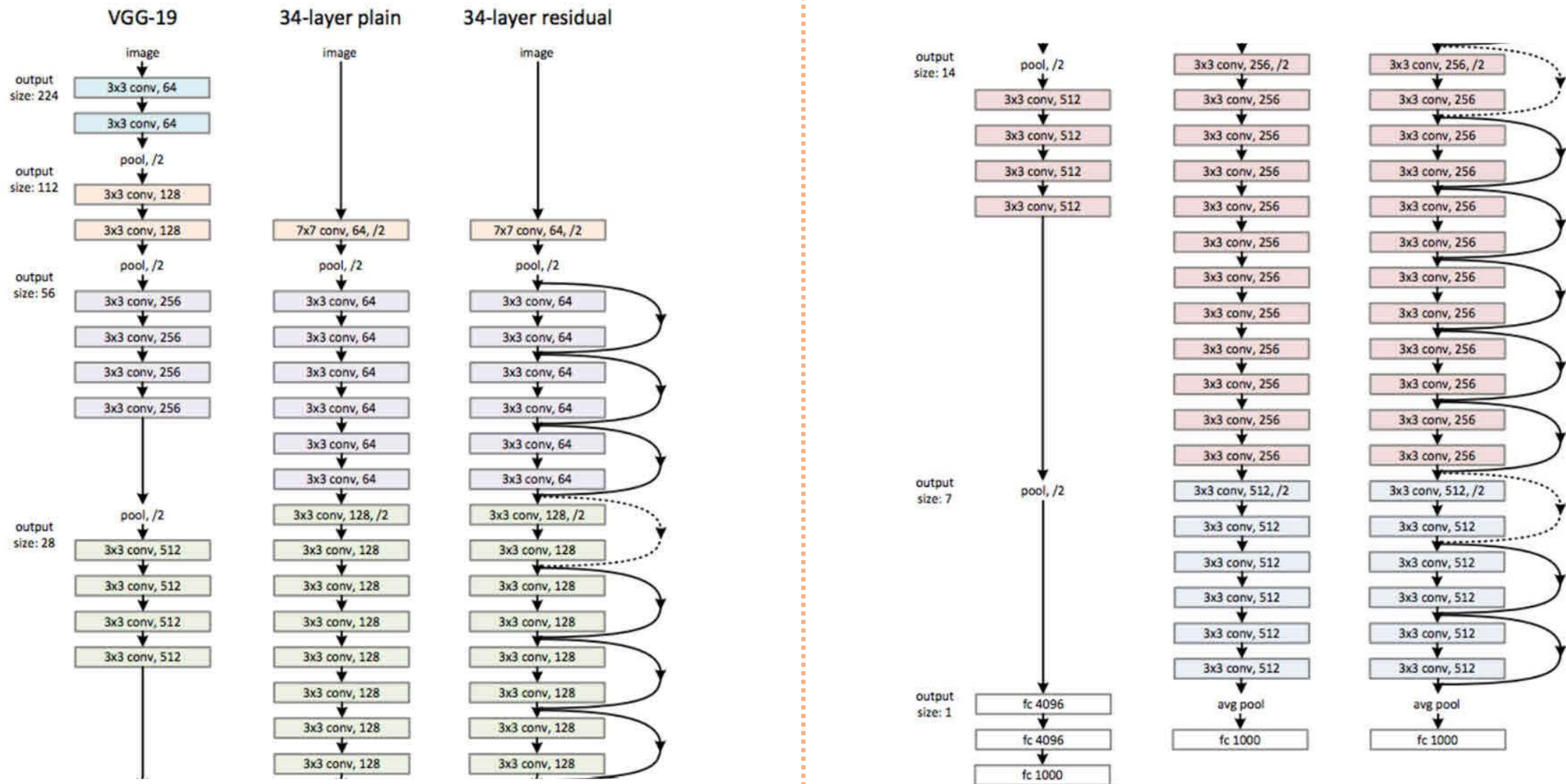
- $F(x) + x$ 를 최소화하려면
  - $x$ 는 현시점에서 변할 수 없는 값이므로  $F(x)$ 를 0에 가깝게 만드는 것이 목적이 됨
  - $F(x)$ 가 0이 되면 출력과 입력이 모두  $x$ 로 같아지게 됨
  - $F(x) = H(x) - x$ 이므로  $F(x)$ 를 최소화 해준다는 것은  $H(x) - x$ 를 최소화 해주는 것과 동일
- $H(x) - x$ 를 잔차(residual)라고 함
- 잔차를 최소화 해주는 것이므로 ResNet이란 이름이 붙음



- ResNet의 구조

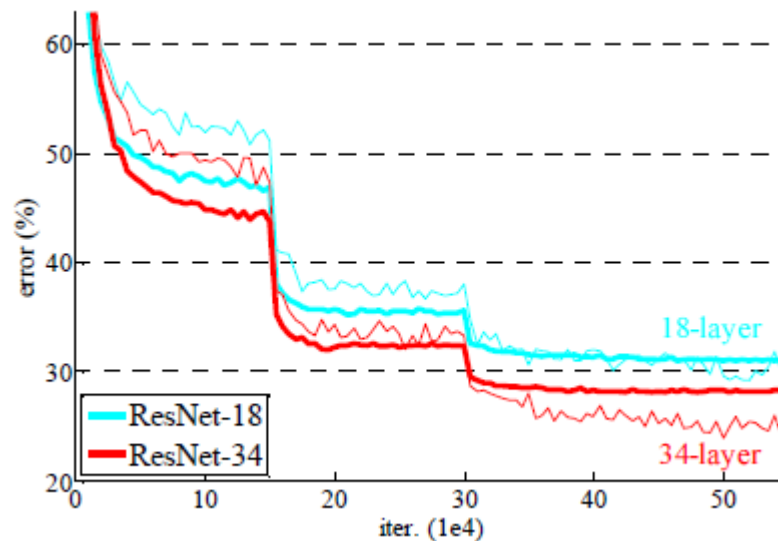
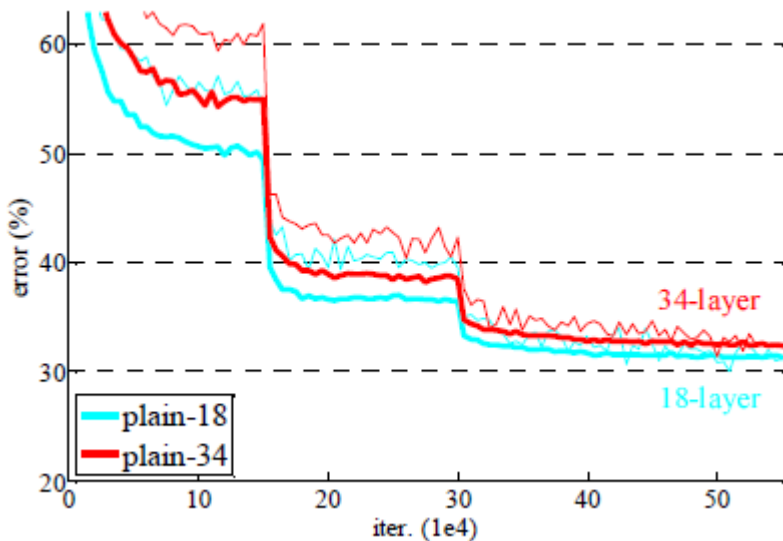
- ResNet은 기본적으로 VGG-19의 구조를 뼈대로 함
- VGG-19의 뼈대에 다수의 컨볼루션 레이어를 추가해서 깊게 만든 후, shortcut을 추가
- 참고 그림: 34층의 ResNet과 거기에서 shortcut들을 제외한 버전인 plain 네트워크의 구조





## • 구조 분석

- 34층의 ResNet은 처음을 제외하고는 균일하게  $3 \times 3$  사이즈의 컨볼루션 필터를 사용
- 특성맵의 사이즈가 반으로 줄어든 때, 특성맵의 깊이를 2배로 증가
- Residual block들이 효과가 있는지를 알기 위해 이미지넷에서 18층 및 34층의 plain 네트 워크와 ResNet의 성능을 비교



- 왼쪽 그래프
  - plain 네트워크는 망이 깊어지면서 오히려 에러가 커졌음
  - 34층의 plain 네트워크가 18층의 plain 네트워크보다 성능이 나쁨
- 오른쪽 그래프(ResNet)
  - 망이 깊어지면서 에러도 역시 작아짐
  - shortcut을 연결해서 잔차(residual)를 최소가 되게 학습한 효과가 있다는 것을 확인

- 18층, 34층, 50층, 101층, 152층의 ResNet의 구성

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				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

- 깊은 구조일수록 성능이 좋음 → 즉, 152층의 ResNet이 가장 성능이 뛰어남

- LeNet-5
  - <https://deep-learning-study.tistory.com/368>
  - <https://bskyvision.com/418>
- AlexNet → <https://bskyvision.com/421>
- VGG-16, VGG-19 → <https://bskyvision.com/504>
- GoogLeNet(inception v1) → <https://bskyvision.com/539>
- ResNet → <https://bskyvision.com/644>