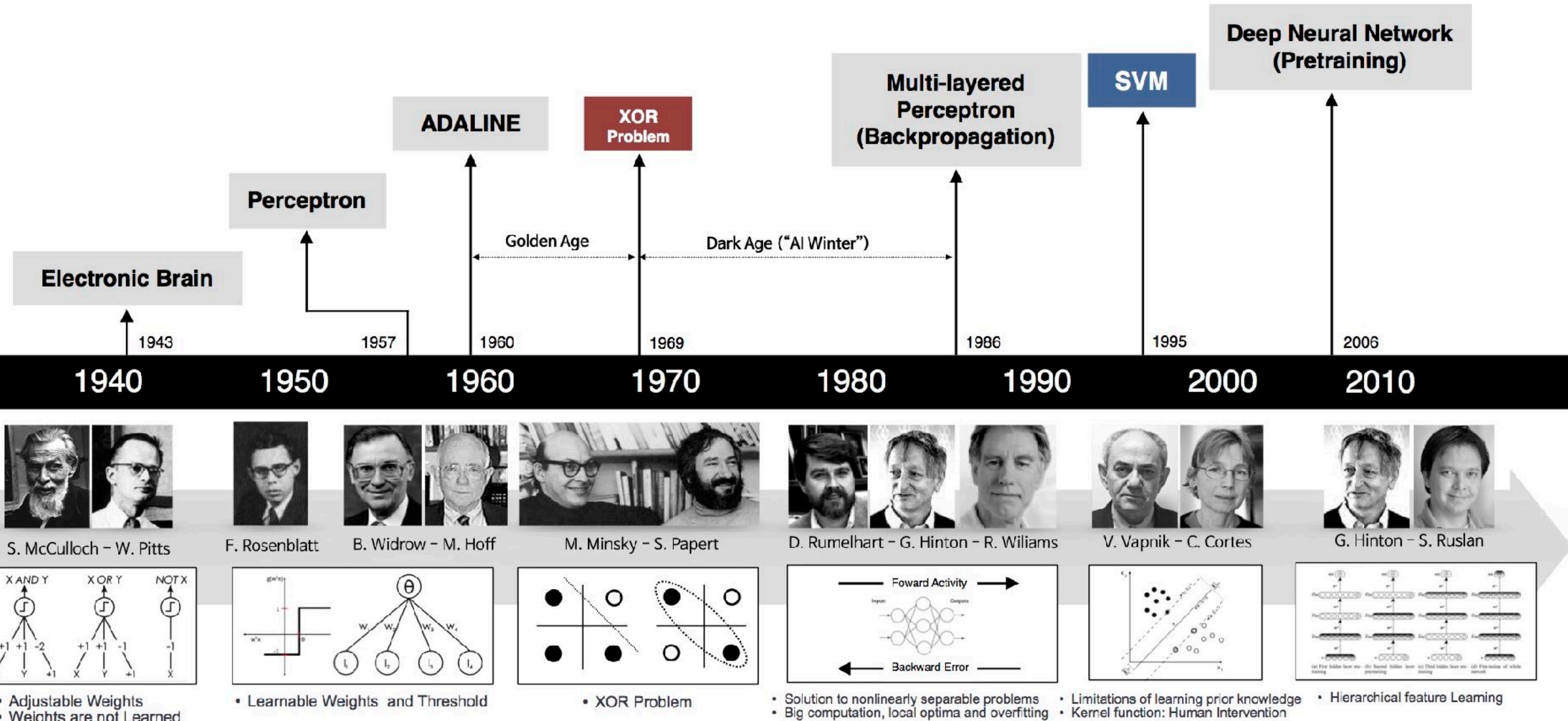


# **Recent trends in Image Recognition**

# Overview

# 인공신경망의 역사

딥러닝의 근간이 되는 인공신경망이라는 개념이 발전하는데는 많은 고난이 있었다.(XOR Problem, SVM, etc)



# 인공신경망의 부활 - 음성 인식

딥러닝이 가장 먼저 두각을 드러낸 분야는 음성인식. **Switchboard**라는 데이터셋이 큰 역할을 담당하였다.



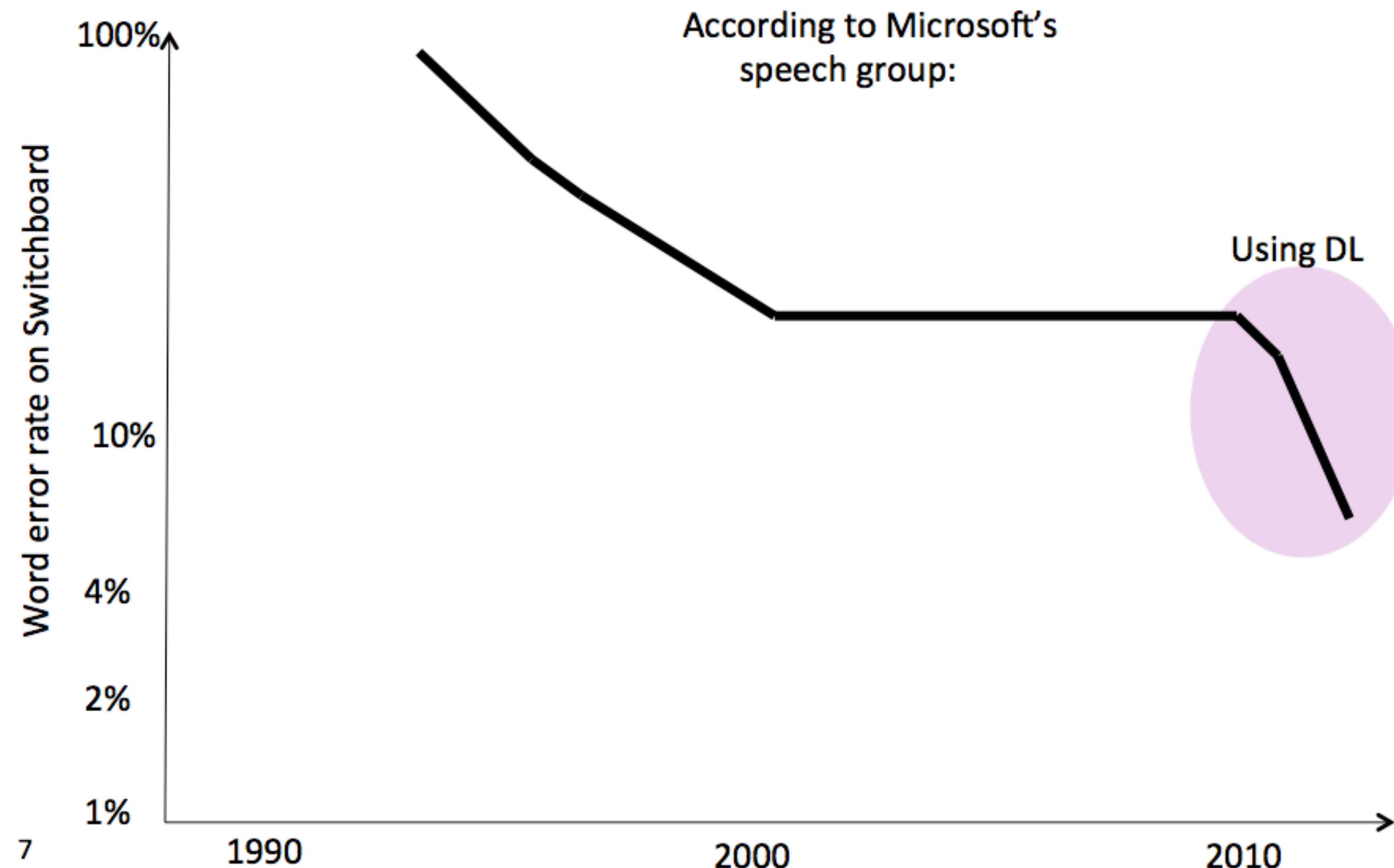
## Switchboard

300  
hours

4,870  
speakers

# 인공신경망의 부활 - 음성 인식

딥러닝 알고리즘을 통해 음성인식의 정확도는 크게 향상되었고, 이는 딥러닝의 가능성을 보여준 첫 사례였다.



# 인공신경망의 부활 - 이미지 인식

ILSVRC (일명 ImageNet Challenge)가 등장. Switchboard와 비등한 규모의 이미지 데이터셋



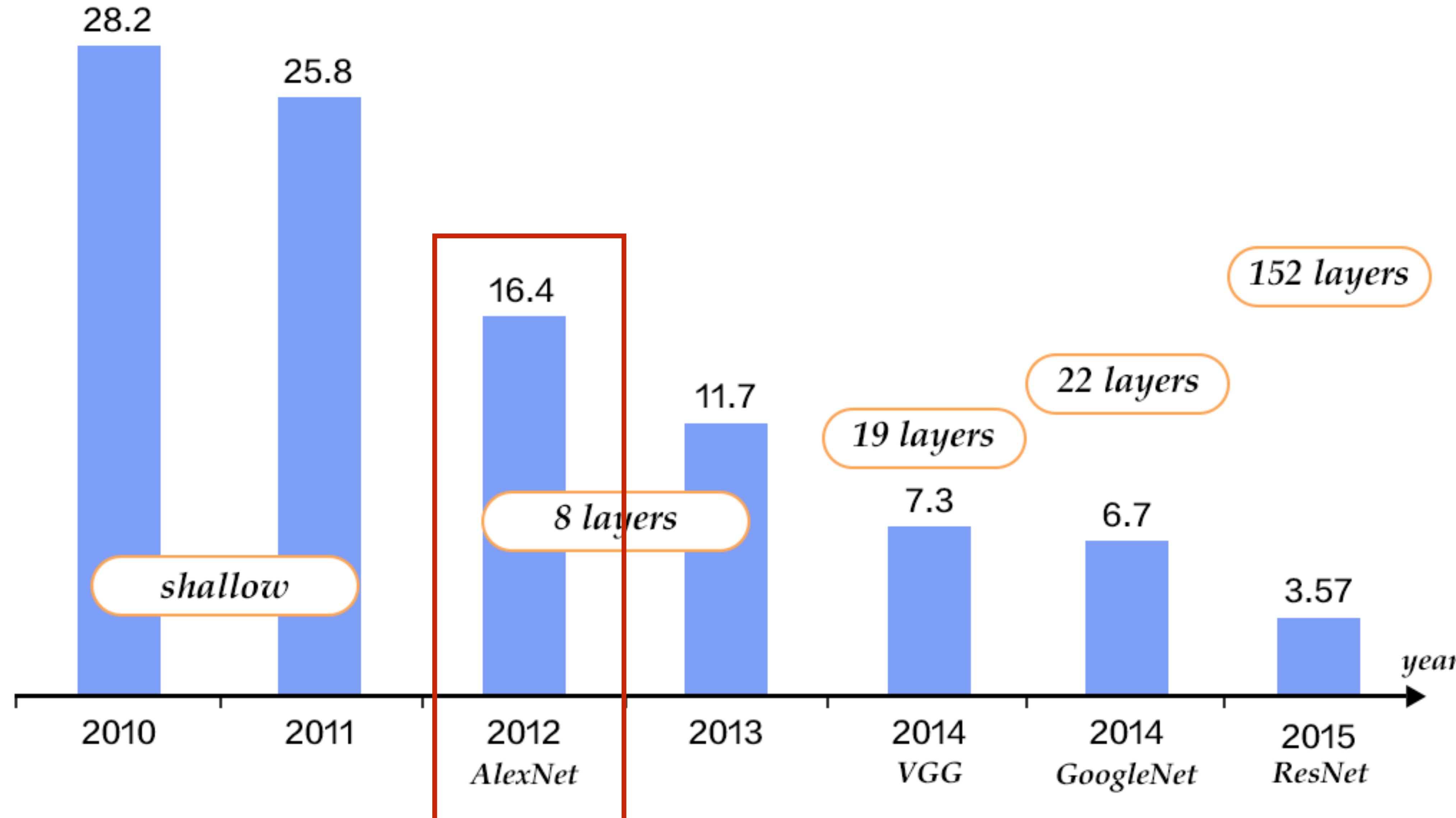
[www.image-net.org](http://www.image-net.org)

**22K** categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
  - Food
  - Materials
- Structures
  - Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
  - Sport Activities

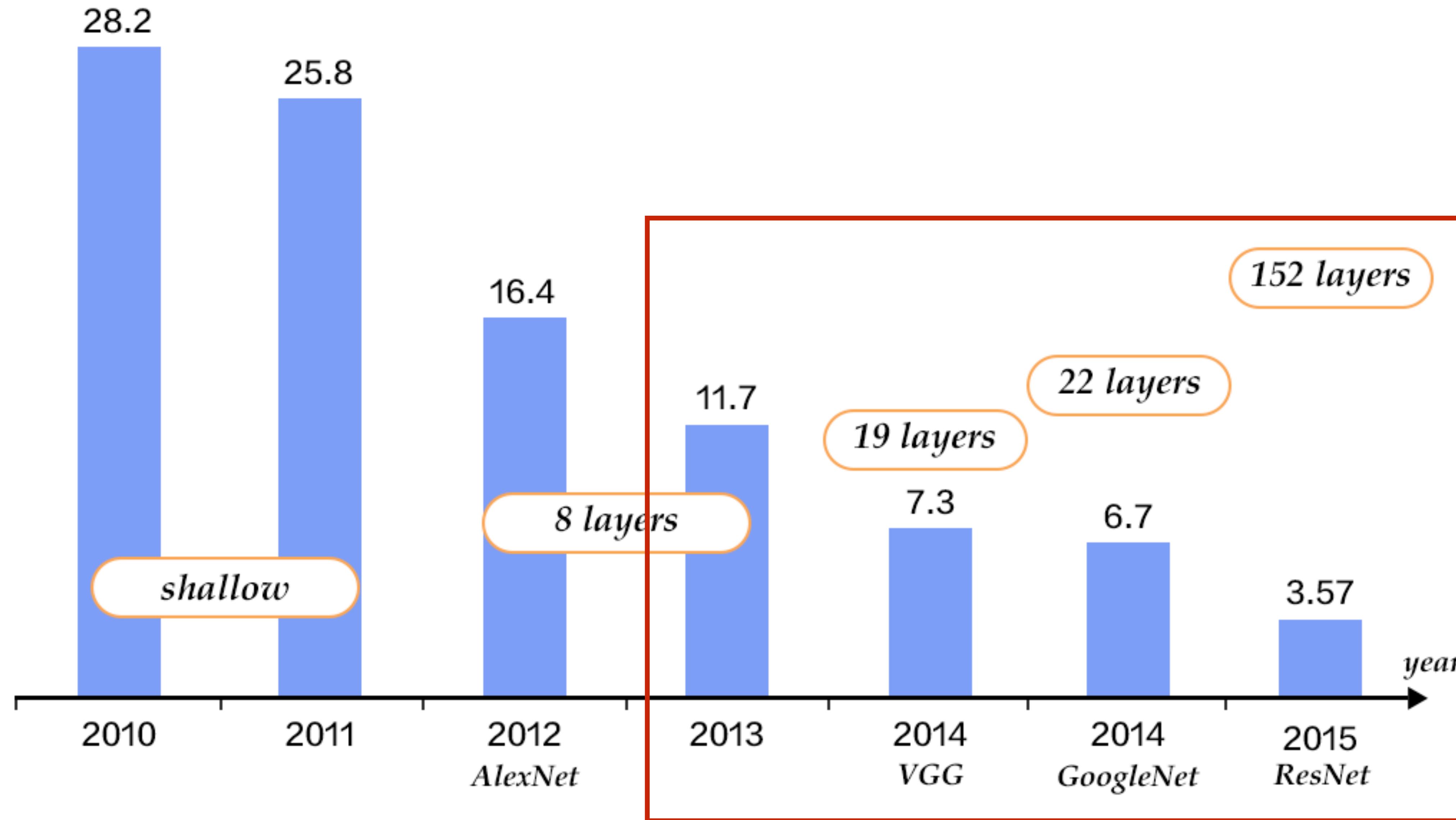
# 인공신경망의 부활 - 이미지 인식

Alex Krizhevsky가 AlexNet으로 1등을 차지. 이후 딥러닝이 ImageNet Challenge를 지배하는 알고리즘이 된다



# 인공신경망의 부활 - 이미지 인식

이후 2015년도의 ResNet(Deep Residual Network)이 인간의 한계 (5%)를 넘어서게 되고,  
이미지 인식(Image Classification) 경진대회는 더 이상 열리지 않게 된다



# 이미지 인식 알고리즘의 발전사

# LeNet5

# LeNet5

Convolutional Neural Network의 시초

매우 적은 메모리 사용량과 연산으로 MNIST 데이터셋에서 99%에 달하는 정확도를 보여주었다

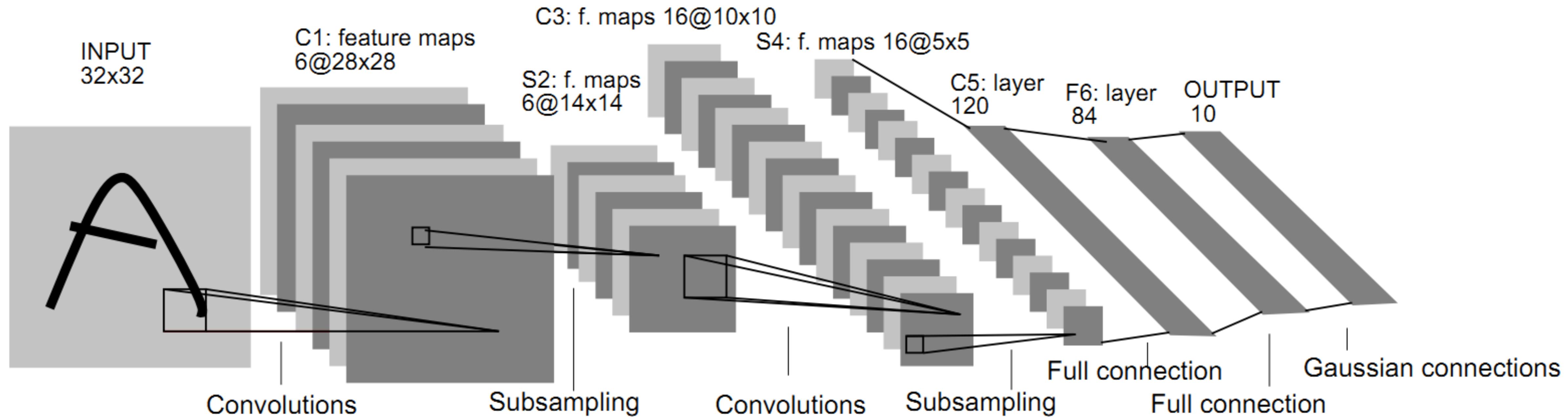


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

# LeNet5

Convolutional Neural Network의 시초

매우 적은 메모리 사용량과 연산으로 MNIST 데이터셋에서 99%에 달하는 정확도를 보여주었다

## Layer Configuration

- 입력: 32x32px
- CONV1: 1@32x32 => CONV(6@1x5x5) => 6@28x28
- POOL1: 6@28x28 => max pooling => 6@14x14
- CONV2: 6@14x14 => CONV(16@6x5x5) => 16@10x10
- POOL2: 16@10x10 => max pooling => 16@5x5
- FLATTEN: 16@5x5 => 400
- FC1: 400 x 120 => 120
- FC2: 120 x 84 => 84
- FC3(output): 84 x 10 => 10

## Optimizer Configuration

- Optimizer: SGD(Stochastic Gradient Descent)
- epochs: 20
- learning\_rate
  - 처음 2 epoch는 0.0005
  - 다음 3 epoch는 0.0002
  - 다음 4 epoch은 0.00005
  - 나머지는 0.00001

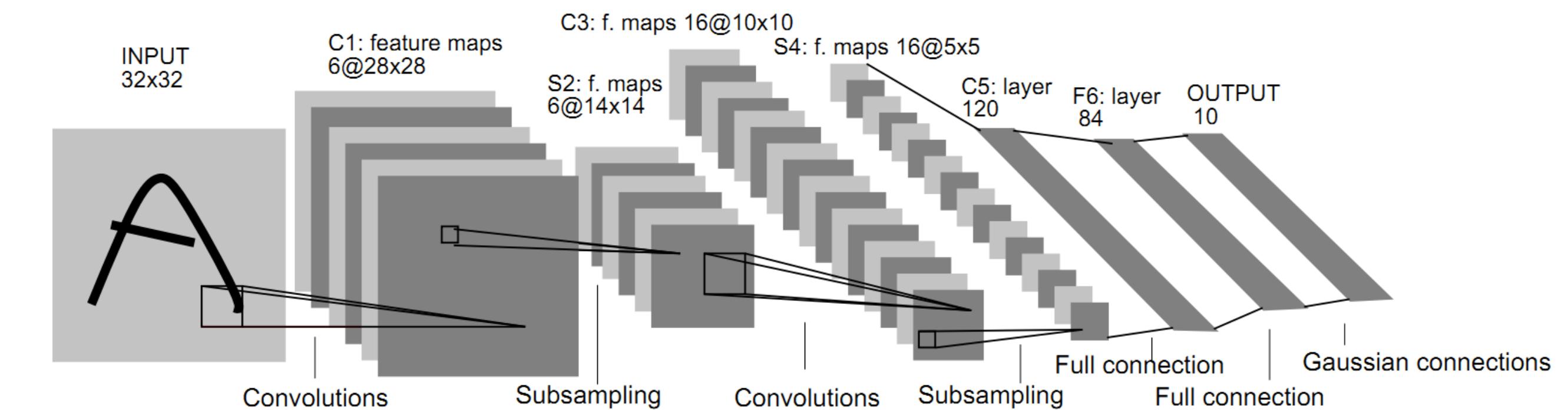
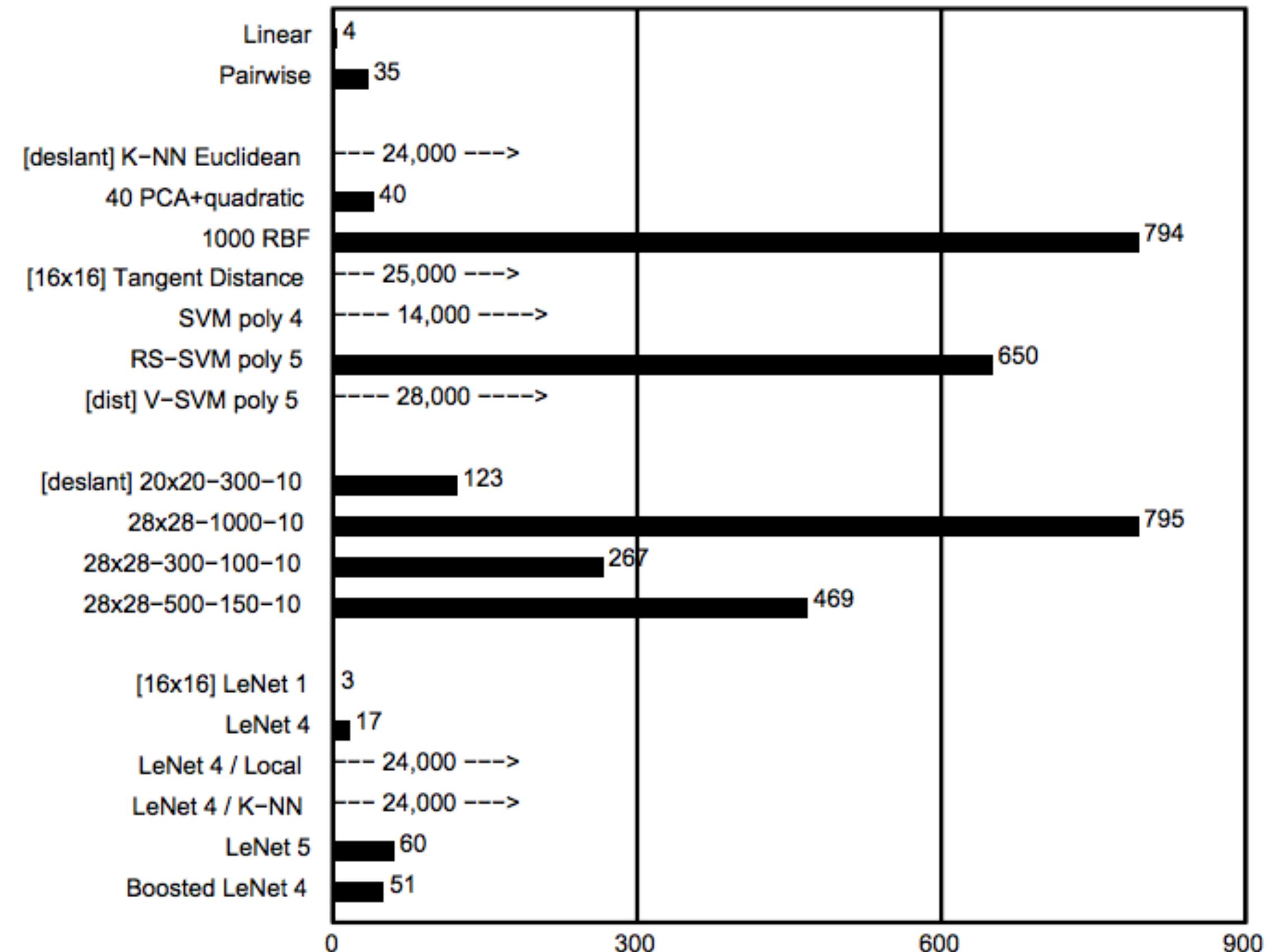
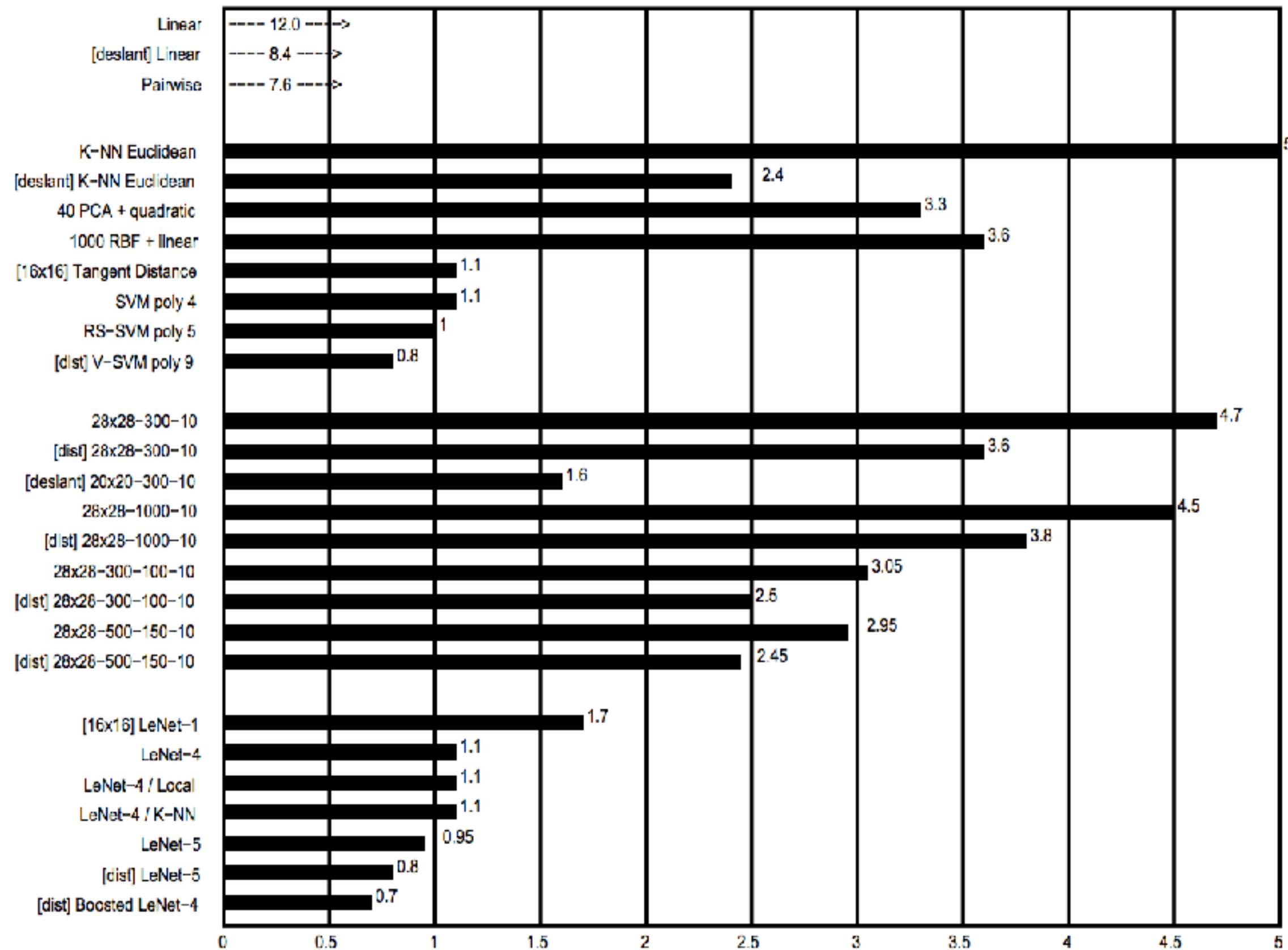


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

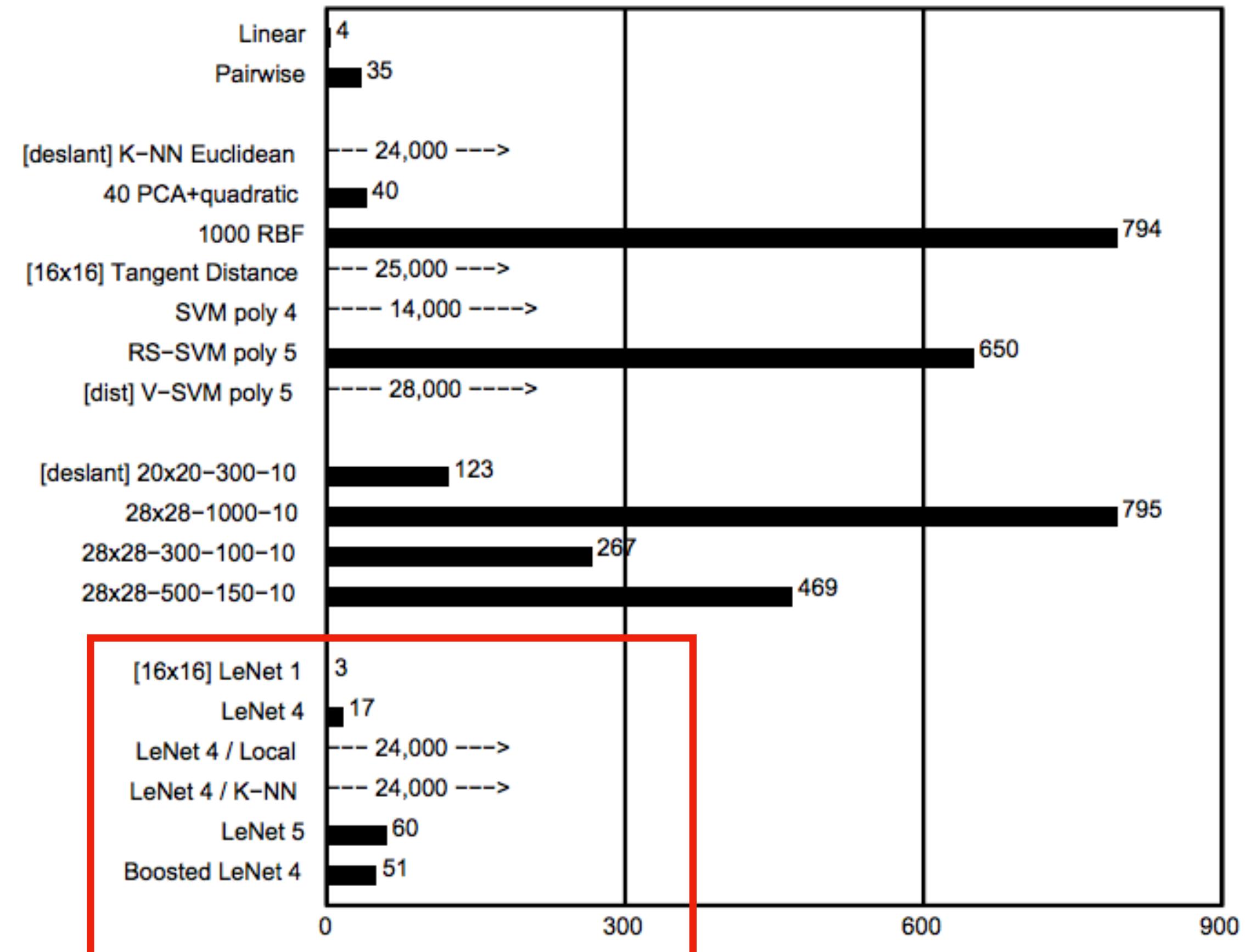
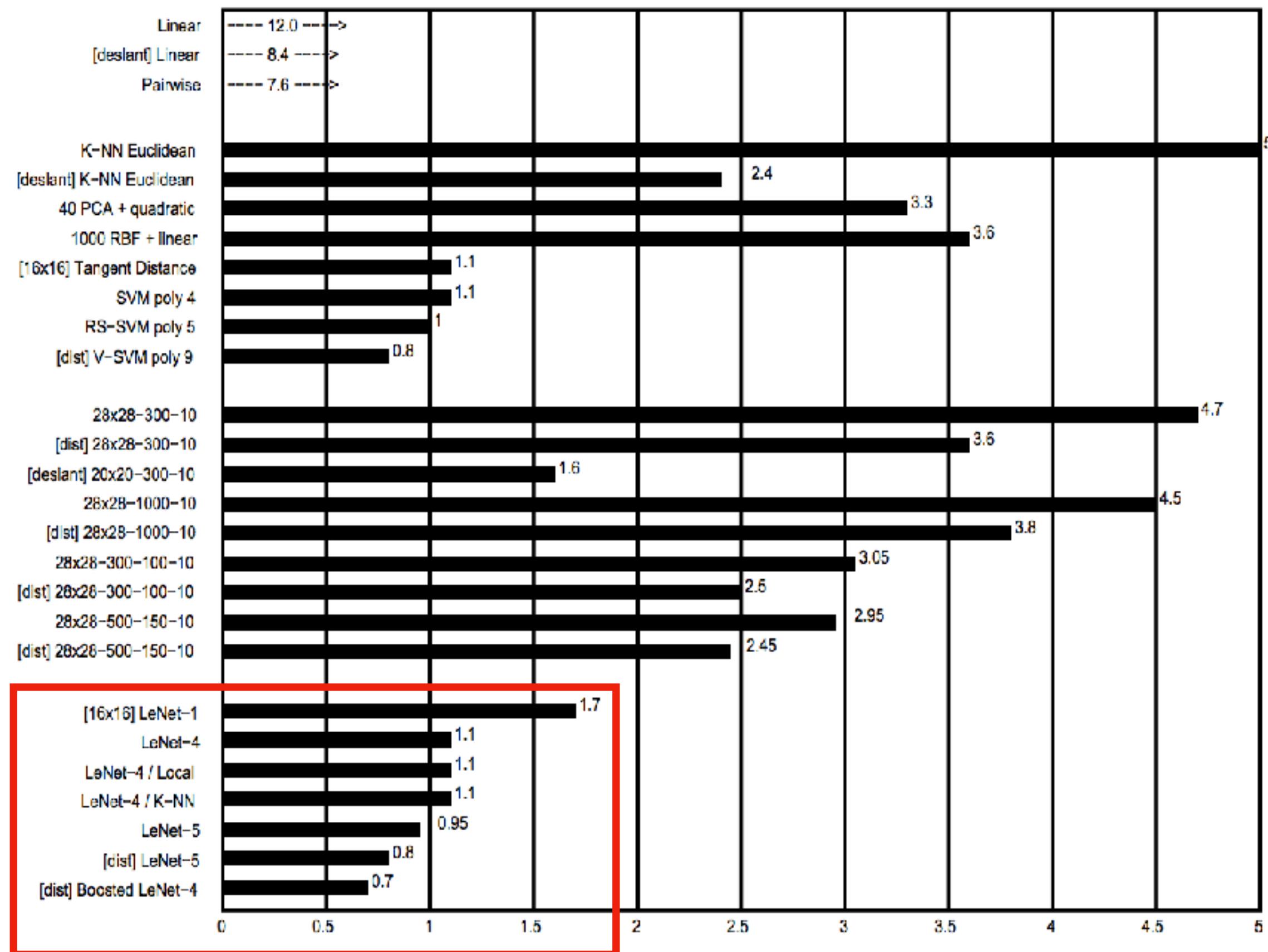
# LeNet5

좌측은 다른 머신러닝 모델과의 정확도 비교 결과, 우측은 다른 머신러닝 모델과의 메모리 사용량 비교 결과  
**LeNet5(CNN)**이 다른 모델에 비해 성능도 더 좋고 메모리도 가장 효율적이라는 것을 알 수 있다



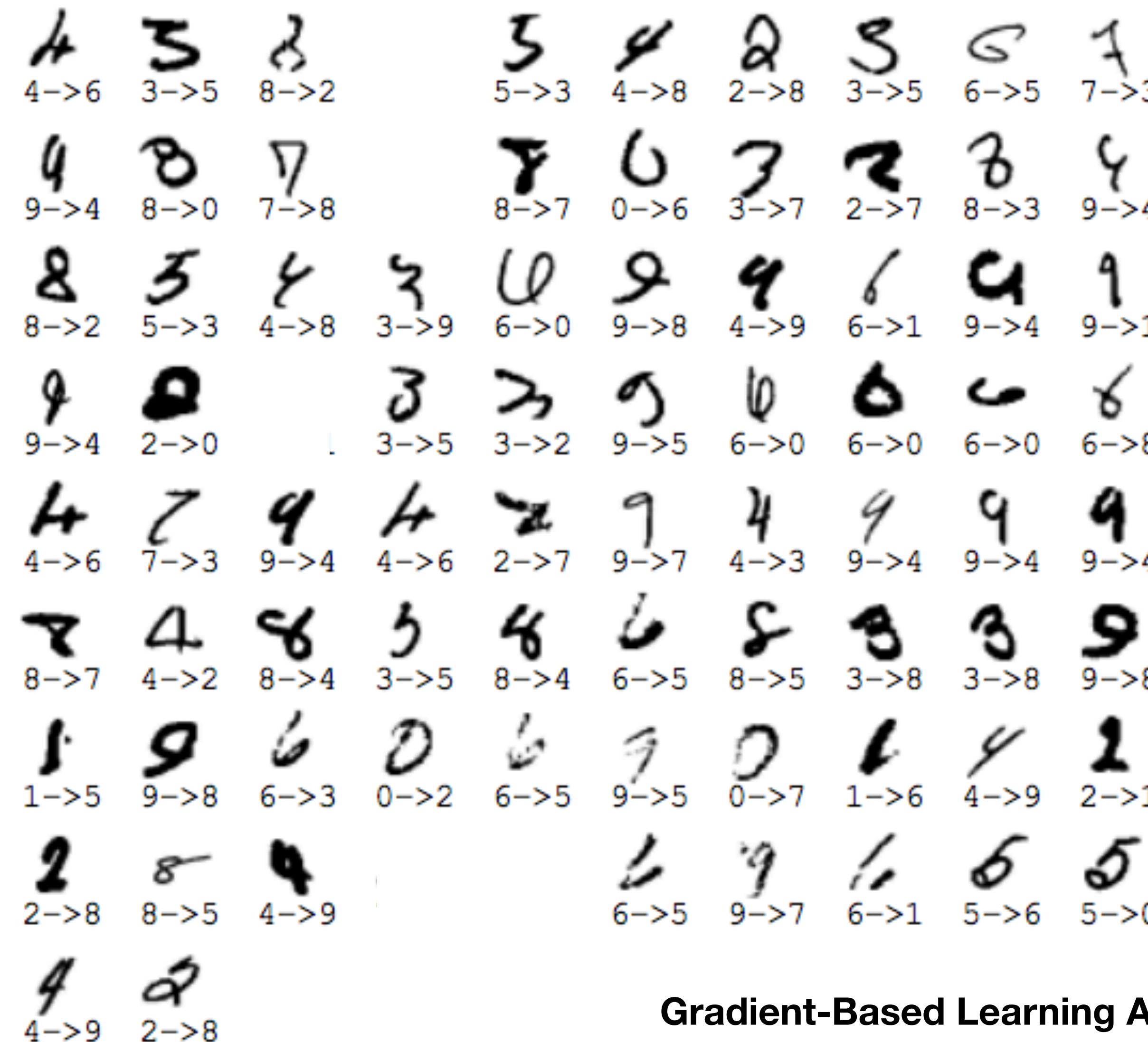
# LeNet5

좌측은 다른 머신러닝 모델과의 정확도 비교 결과, 우측은 다른 머신러닝 모델과의 메모리 사용량 비교 결과  
LeNet5(CNN)이 다른 모델에 비해 성능도 더 좋고 메모리도 가장 효율적이라는 것을 알 수 있다



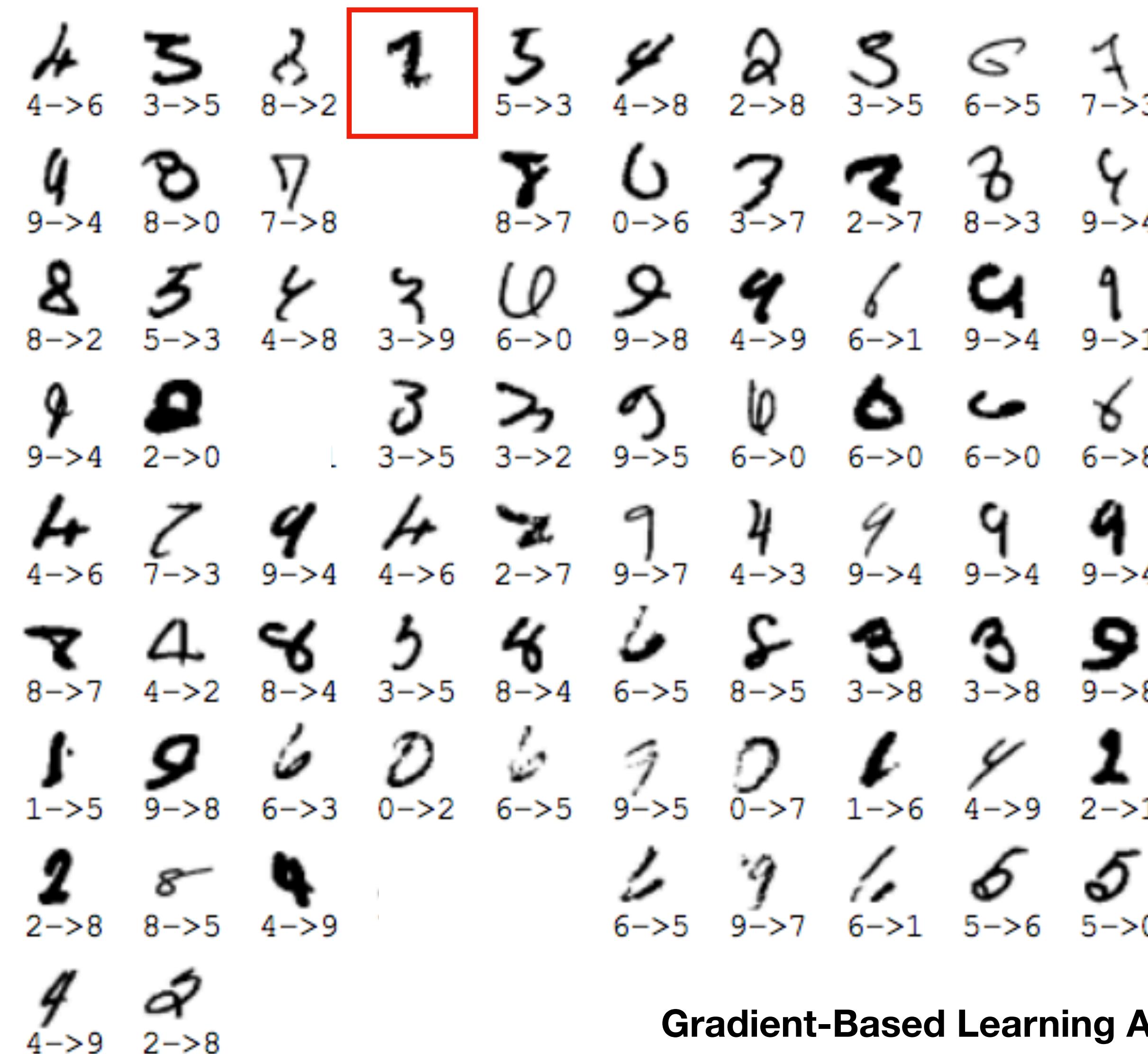
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다



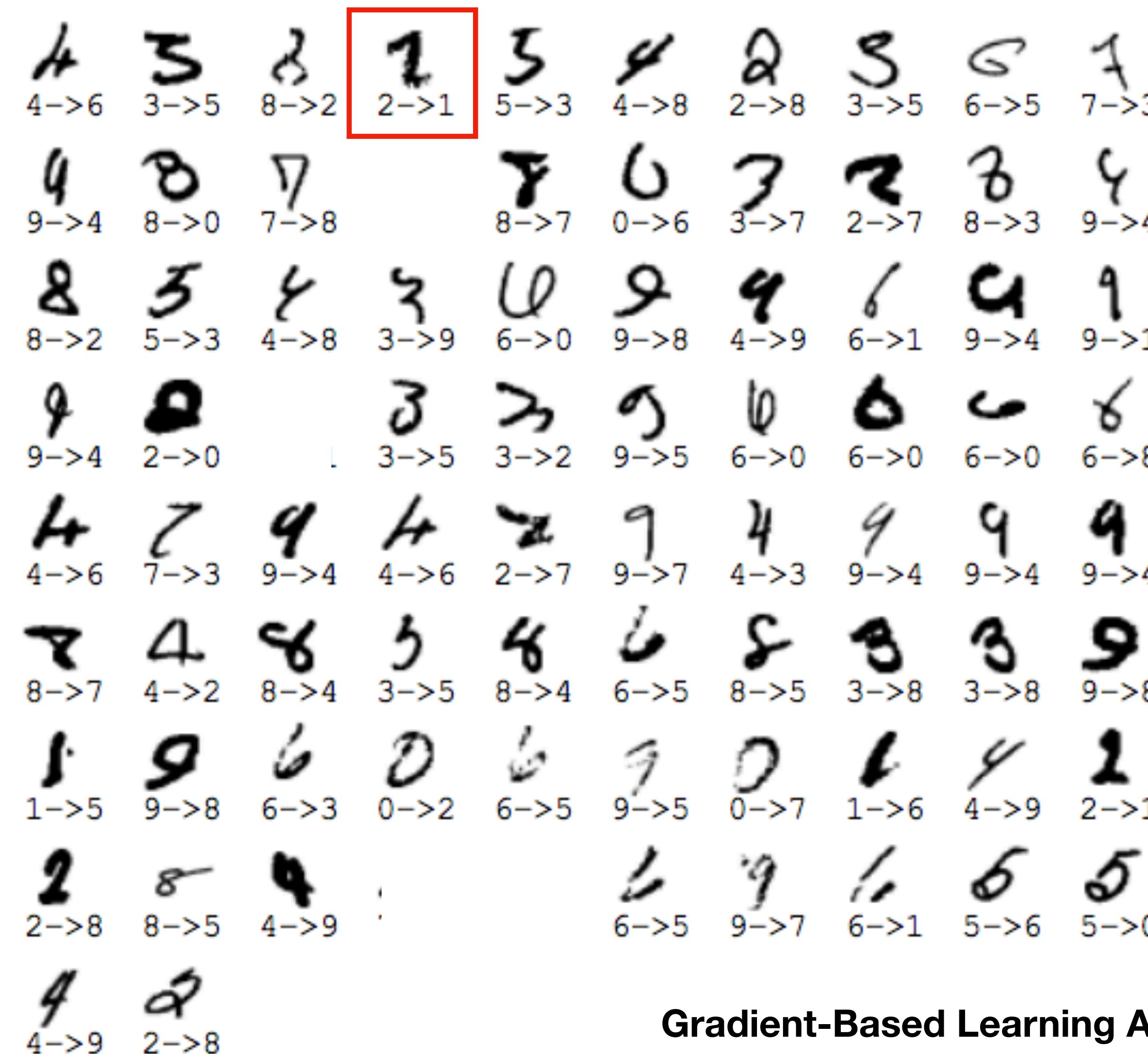
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다



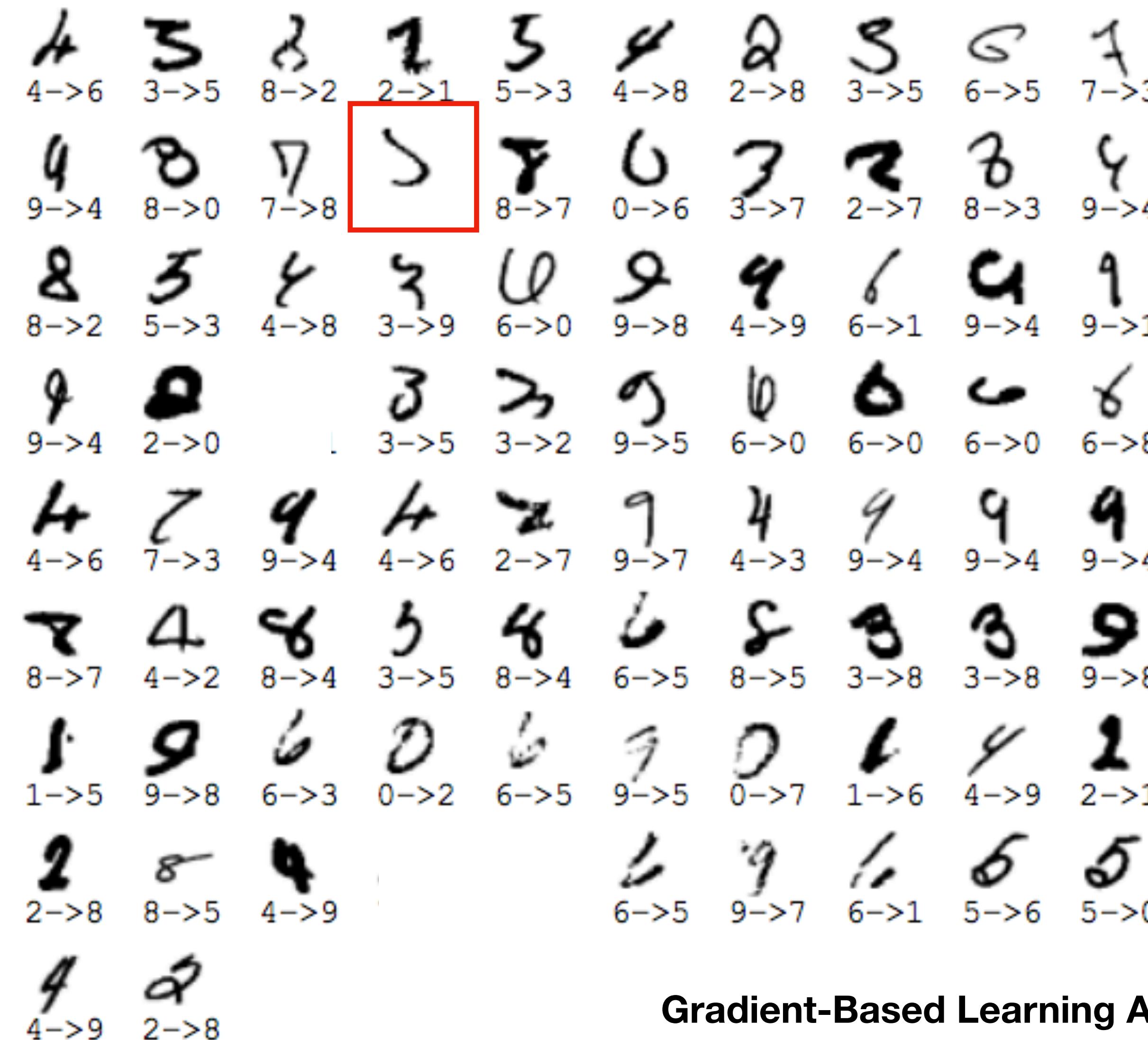
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다



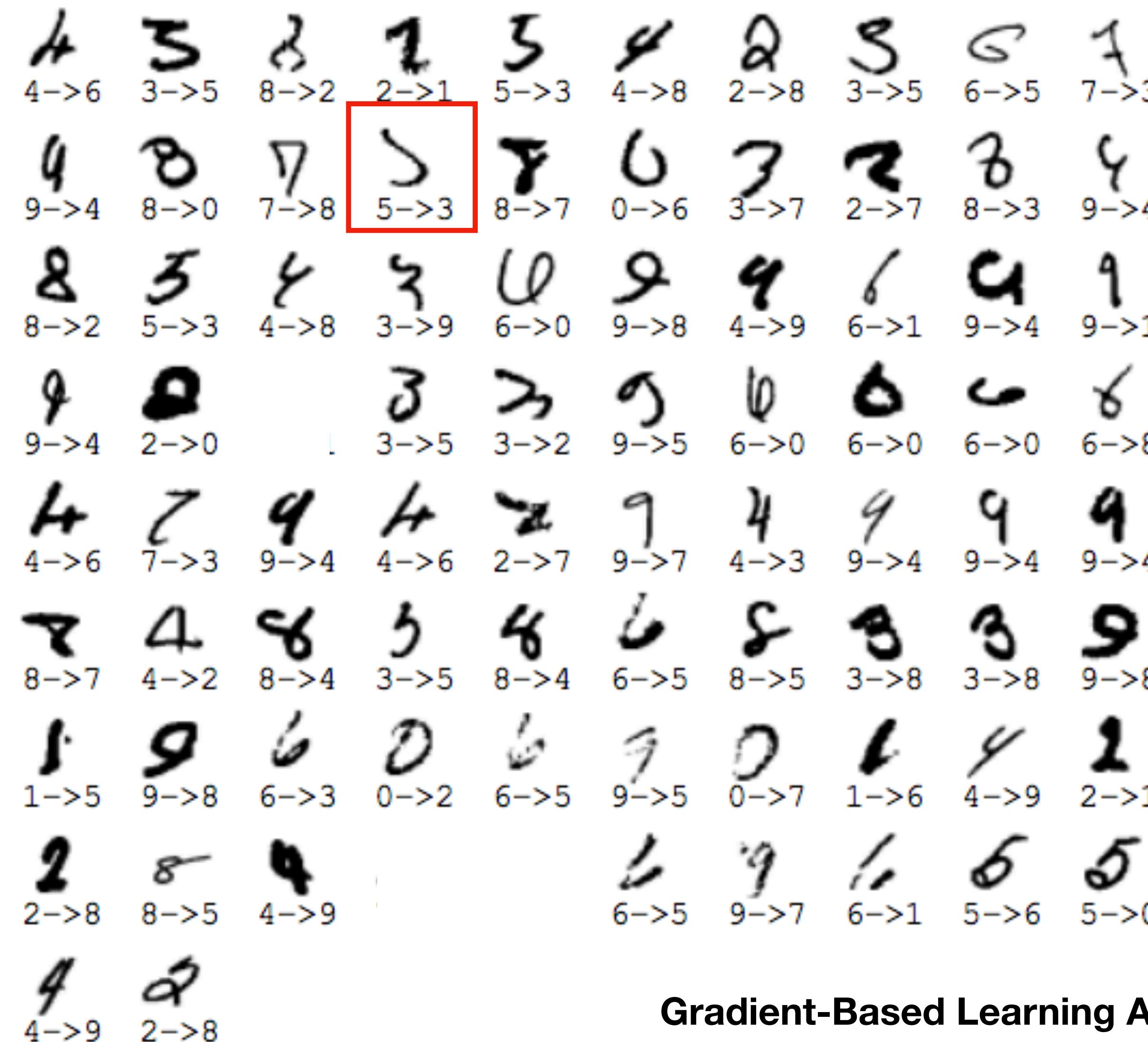
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다



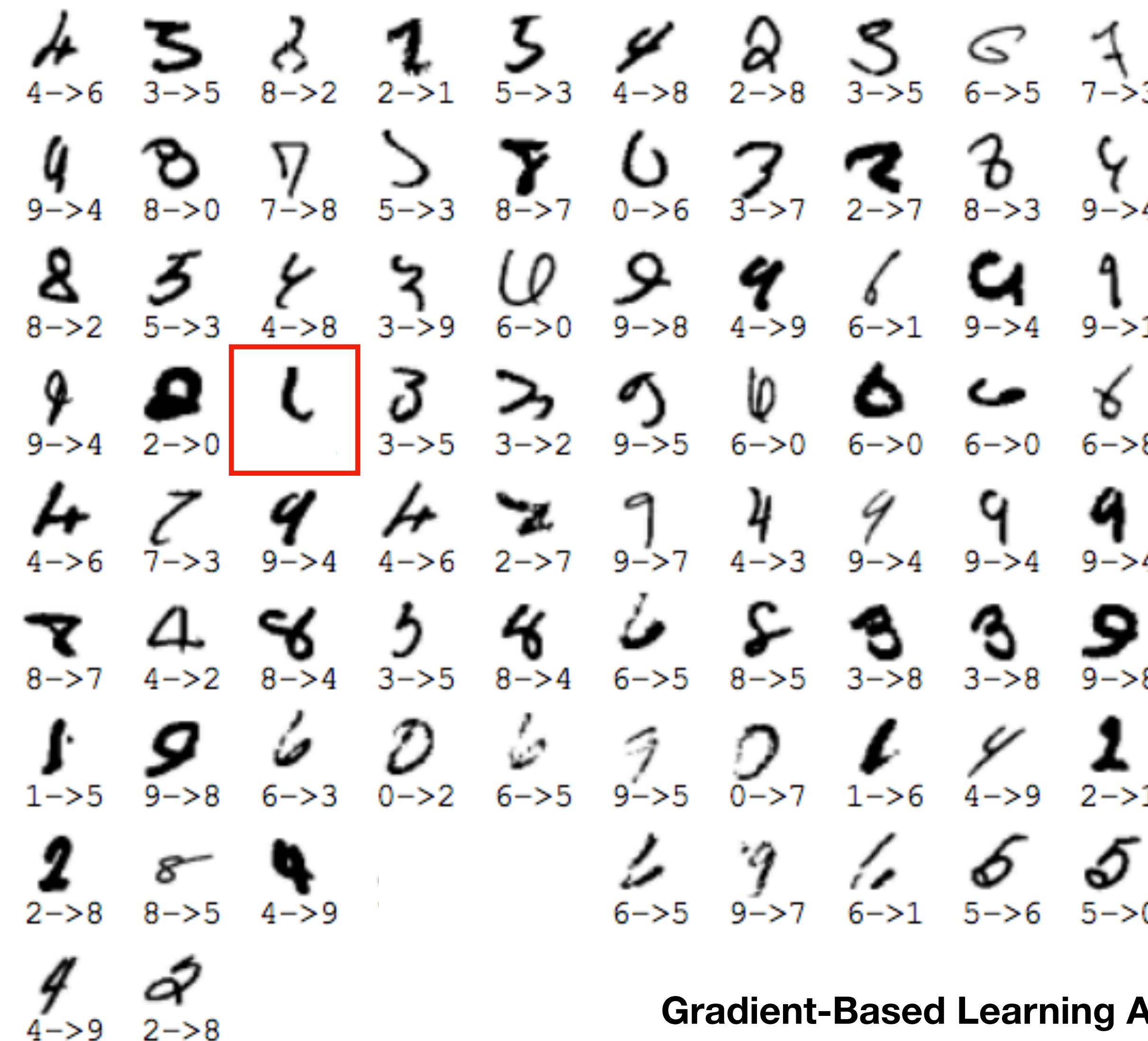
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다



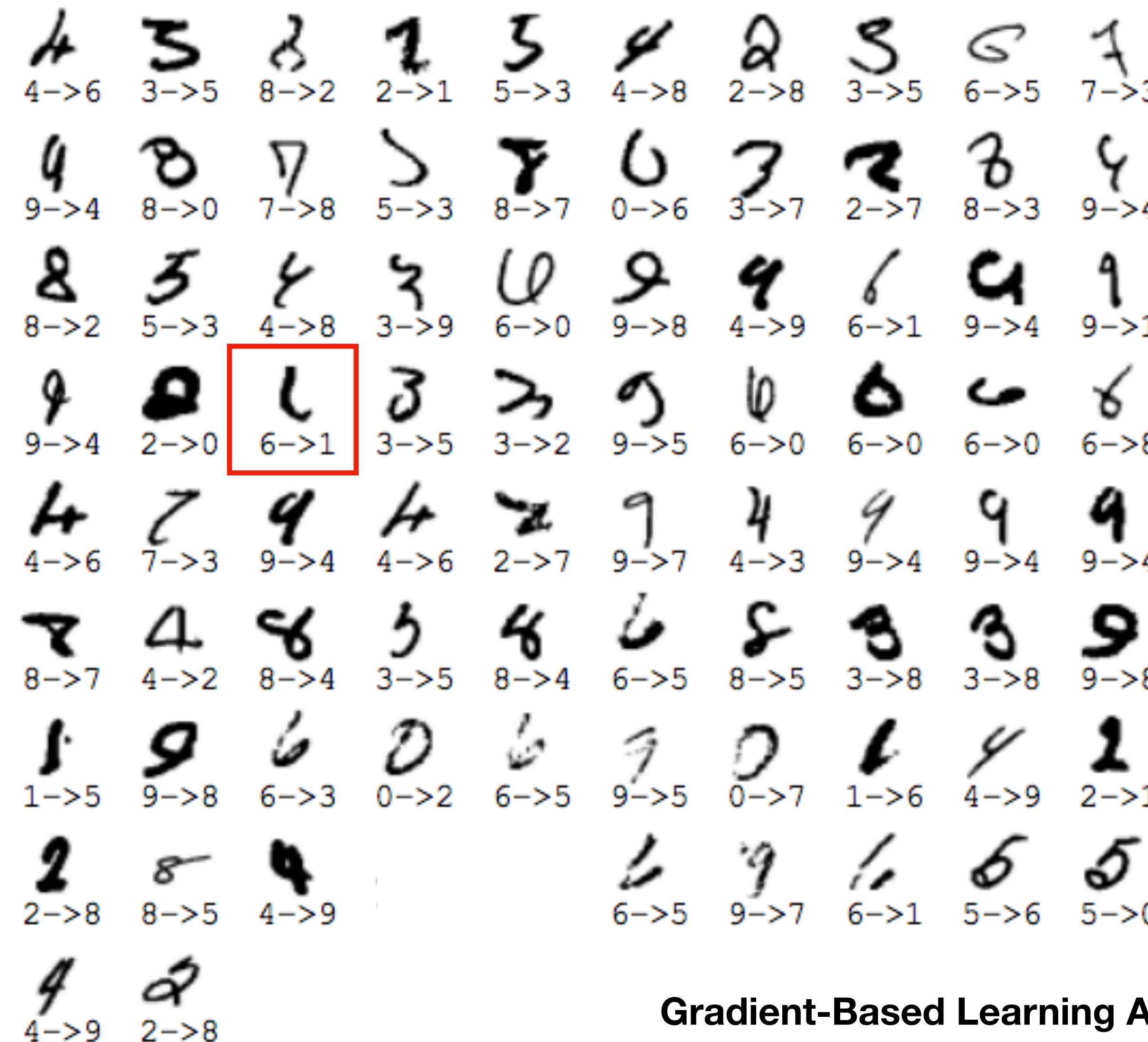
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다



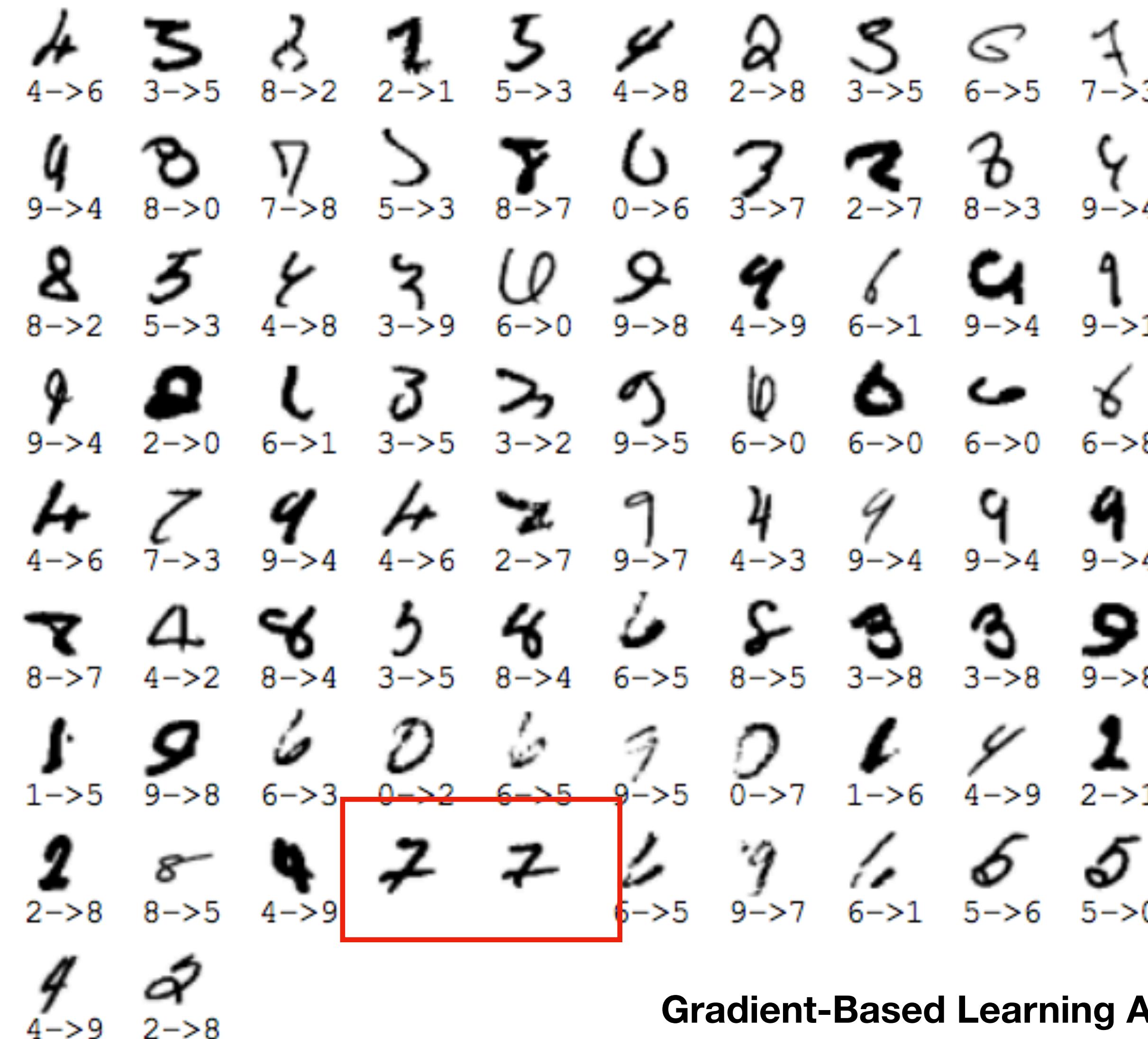
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다



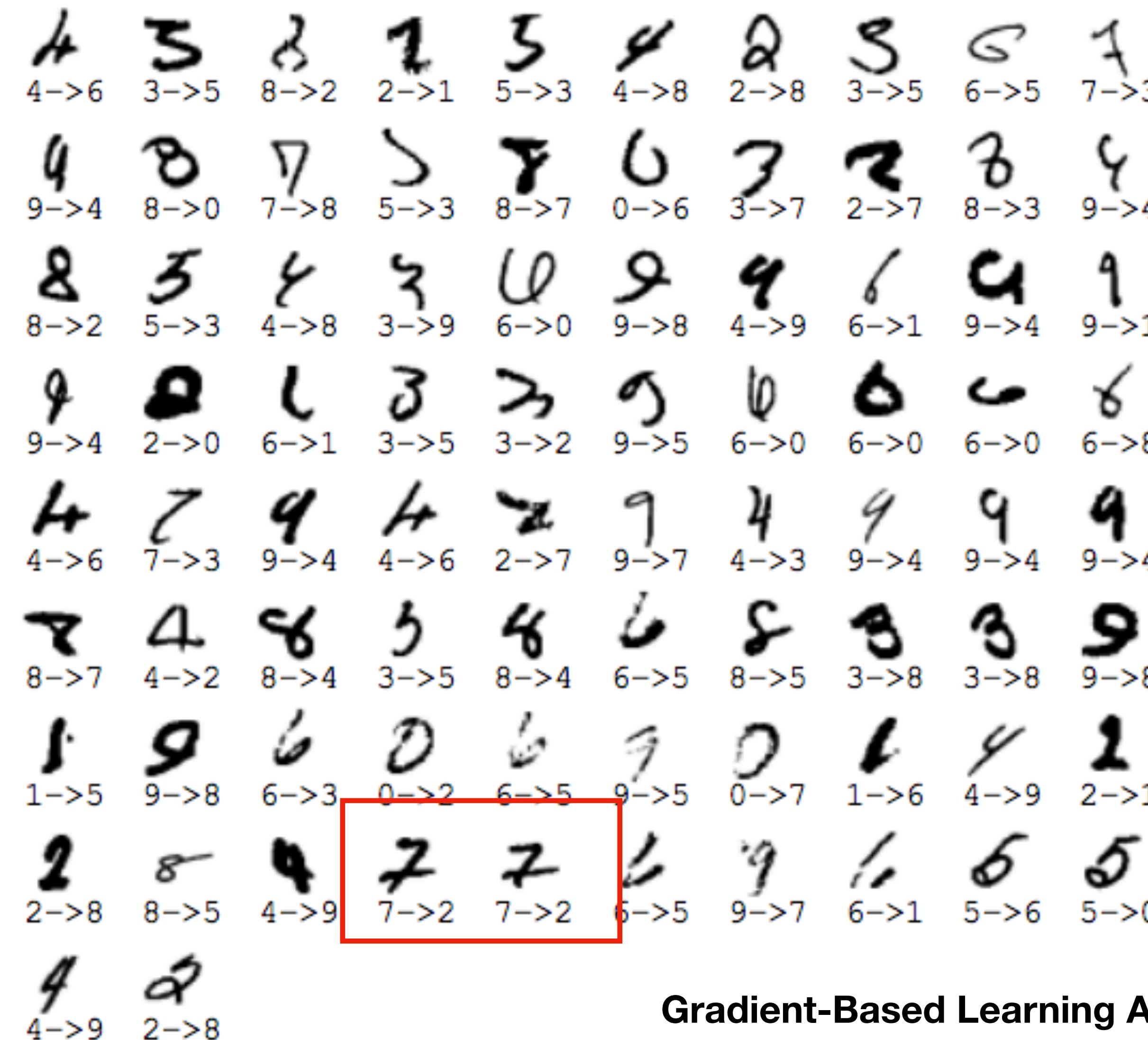
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다



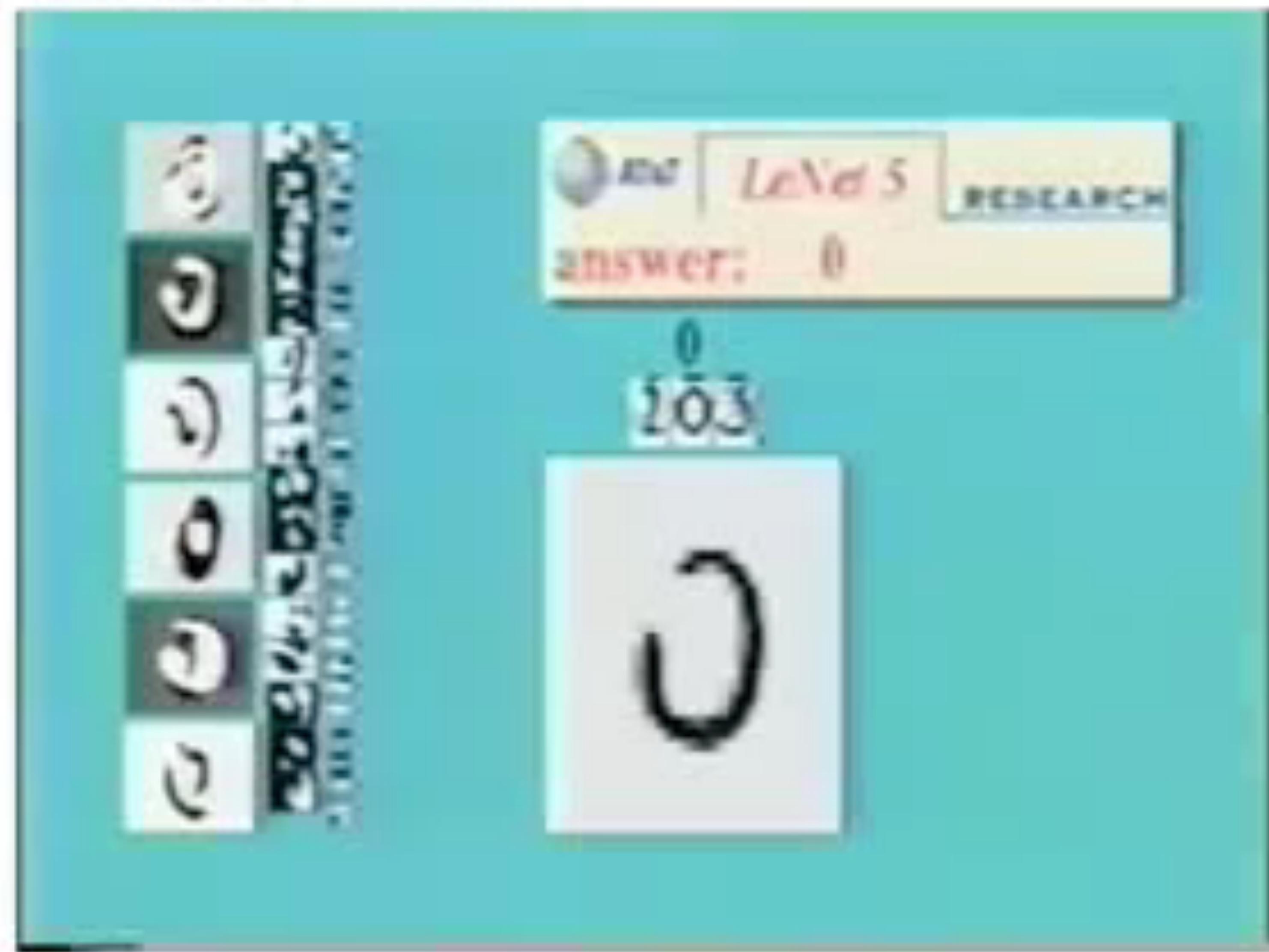
# LeNet5

LeNet5가 맞추지 못한 0.8%의 테스트 데이터들. (총 81개) 그림 아래쪽 좌측이 정답, 우측이 LeNet5의 예측값이다  
사람이 보더라도 구분할 수 없는 예제들이 몇몇 보인다





# Handwritten digit classification

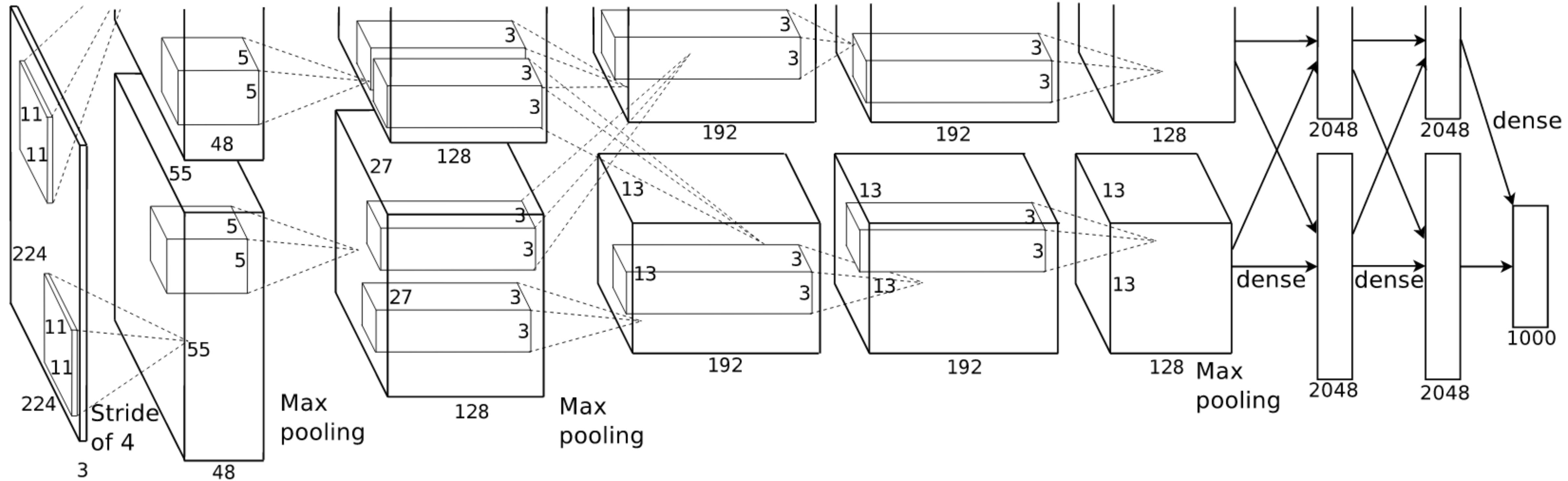


# AlexNet

# AlexNet

ImageNet Challenge 2012년 우승 모델

2010년도부터 일어난 일어난 딥러닝, 나아가 인공지능 혁명을 일으킨 최초의 모델이라고 볼 수 있다



ImageNet Classification with Deep Convolutional Neural Networks

Krizhevsky et al, 2012. <https://goo.gl/hqAxZD>

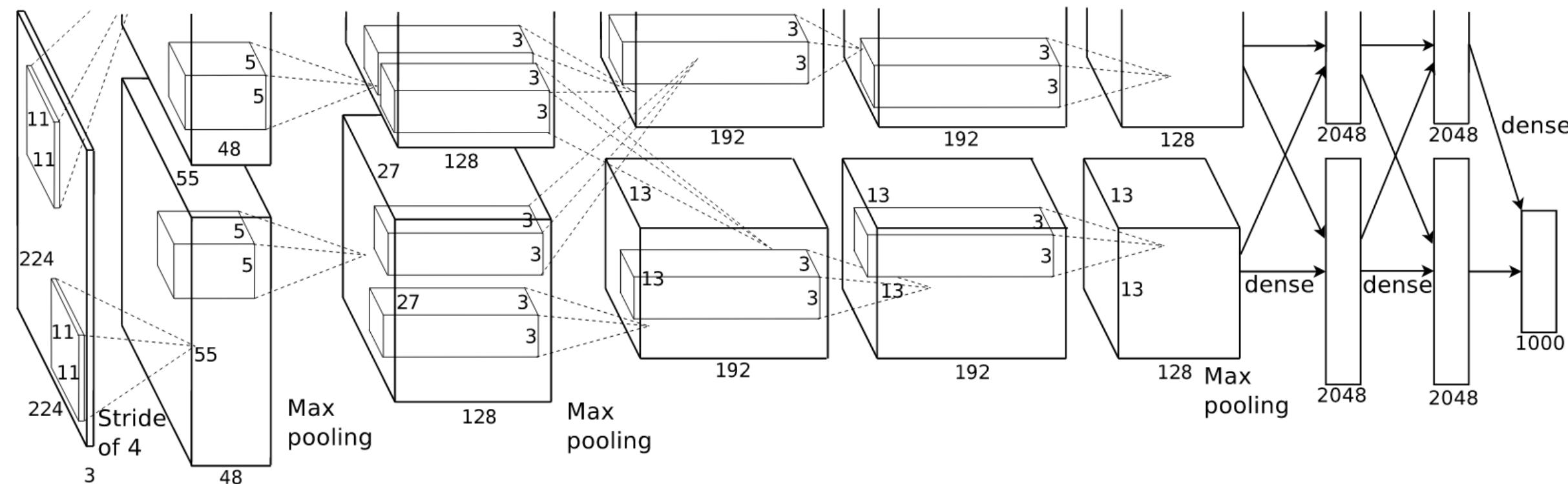
# AlexNet

모델의 기본적인 구조는 우리가 배운것과 거의 동일하다

**CONV -> POOL -> CONV -> POOL -> ... -> FC -> OUTPUT** 구조. 여기서 몇 가지만 추가된다

## Layer Configuration

- 입력: 227x227px, RGB Channels(3)
- CONV1: 96@11x11, **stride 4, pad 0**
- **POOL1: 3x3 filters, stride 2**
- **NORM1: Normalization layer**
- CONV2: 256@5x5, stride 1, pad 2
- POOL1: 3x3 filters, stride 2
- NORM2: Normalization layer
- CONV3: 384@3x3 filters, stride 1, pad 1
- CONV4: 384@3x3 filters, stride 1, pad 1
- CONV5: 256@3x3 filters, stride 1, pad 1
- POOL3: 3x3 filters, stride 2
- FC6: 4096 neurons
- FC7: 4096 neurons
- FC8(output): 1000 neurons



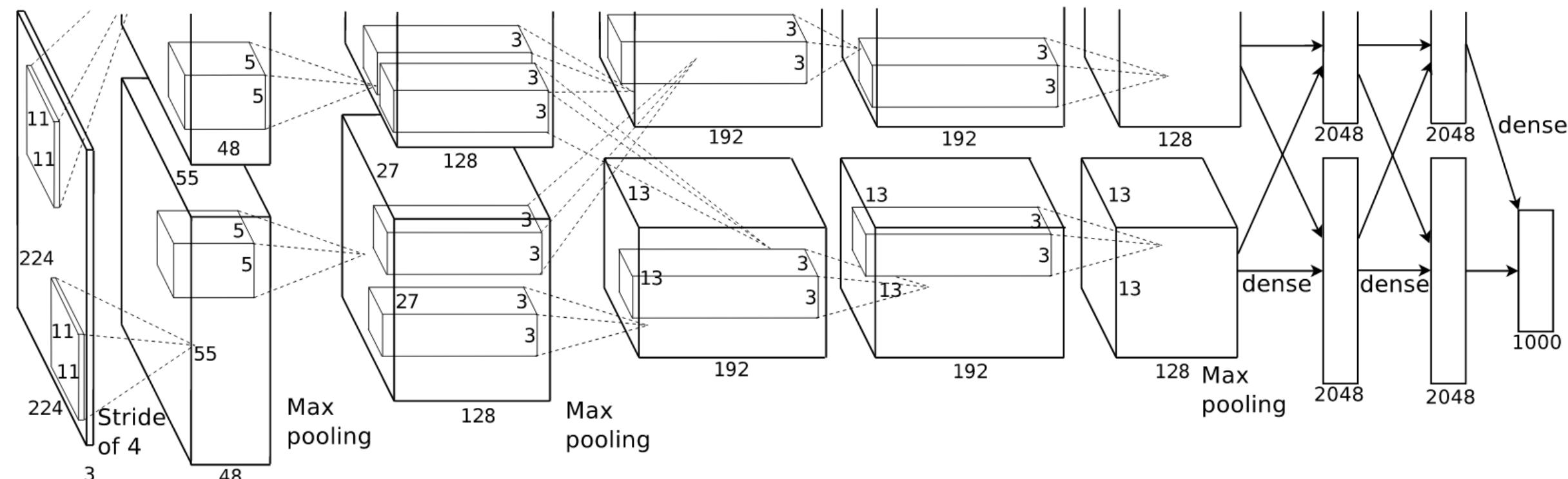
# AlexNet

모델의 기본적인 구조는 우리가 배운것과 거의 동일하다

**CONV -> POOL -> CONV -> POOL -> ... -> FC -> OUTPUT** 구조. 여기서 몇 가지만 추가된다

## Other Configuration

- Weight는 Gaussian Distribution으로 초기화. 평균은 0, 표준편자는 0.01.
- Sigmoid를 ReLU로 대체
- Local Response Normalization layer를 도입. (요즘은 사용하지 않음)
- Dropout: Dropout Rate 0.5
- Batch-size: 128
- Optimizer: SGD
  - Momentum(가속도)를 0.9 추가
  - Weight Decay를 0.0005 추가
- Learning-rate
  - 처음에는 0.01.
  - loss가 안 떨어질 때마다 1/10씩 감소. 총 3번
- 같은 AlexNet을 7개 만들어서 섞어씀. (Ensemble)



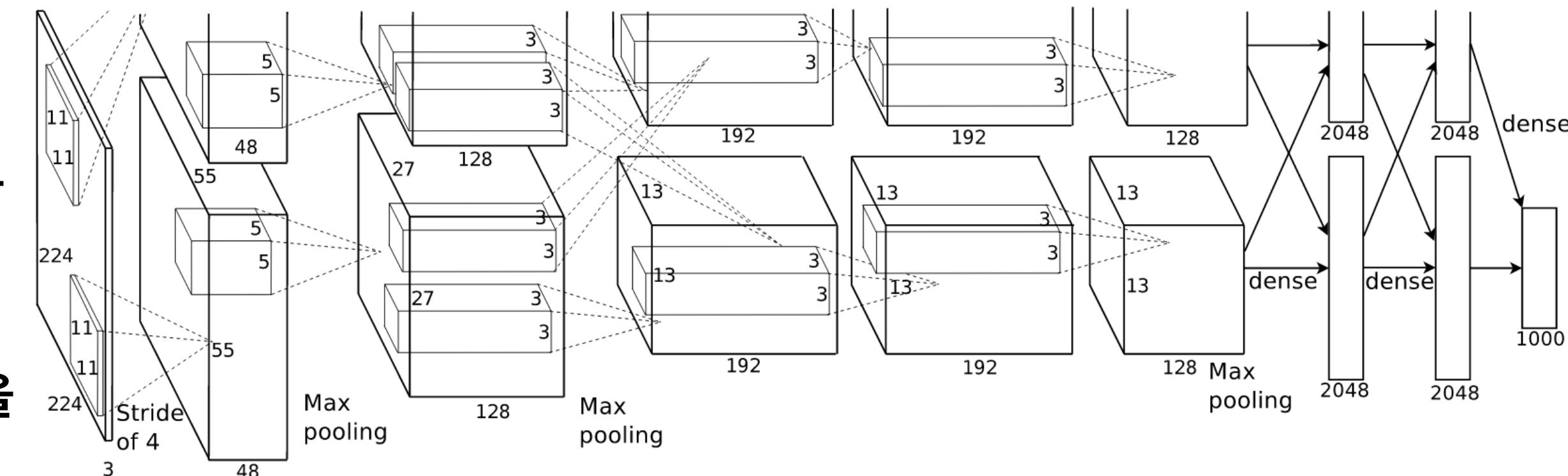
# AlexNet

모델의 기본적인 구조는 우리가 배운것과 거의 동일하다

**CONV -> POOL -> CONV -> POOL -> ... -> FC -> OUTPUT** 구조. 여기서 몇 가지만 추가된다

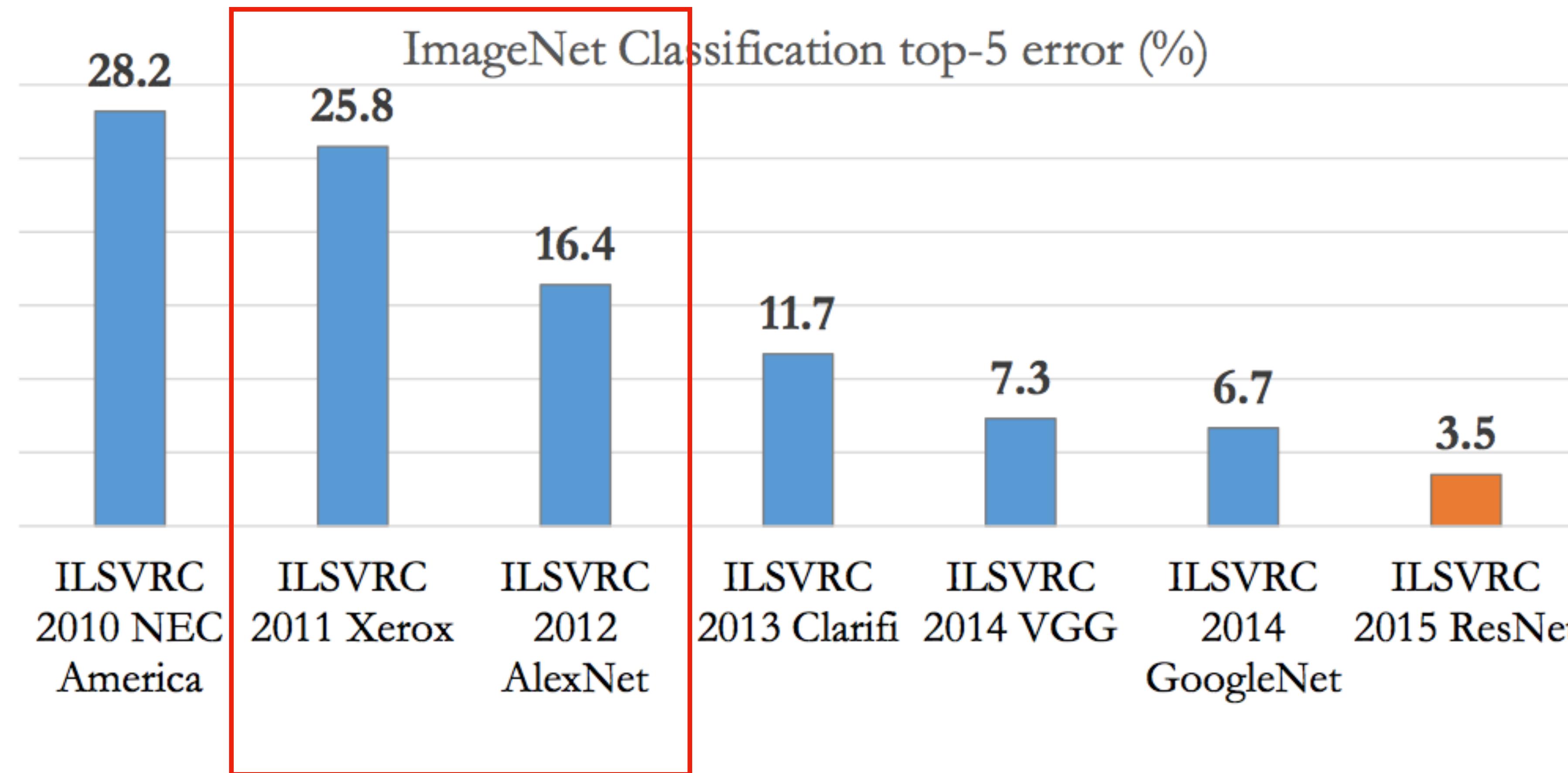
## Others

- GPU를 적극적으로 활용. 다만 지원이 제대로 안 되어서 모델을 둘로 쪼개서 넣음
- **Data Augmentation**을 굉장히 많이 함
  - 좌우 반전
  - 학습할 때는 이미지를 256x256px로 줄인 후, 랜덤하게 224x224px로 잘라서 사용. 이 방식으로 데이터를 2048배 불려서 사용함
  - 테스트할 때는 256x256px 이미지에서 상-하-좌-우-가운데를 224x224px로 자른 뒤, 좌우반전해서 10개를 만듬. 10개의 결과의 평균을 사용.
- 이미지를 넣기 전에 PCA(Principal component analysis)로 RGB Channel을 조정해주는 방식을 사용. (하지만 현재는 쓰지 않는다)



# AlexNet

이전 우승자(Xerox)에 비해 10% 가까이 에러를 낮출 수 있었다.  
이후 ImageNet Challenge에 도전하는 모든 모델은 딥러닝을 사용하게 된다.



# AlexNet

사진의 정답과 AlexNet이 예측한 정답 후보와 그 확률  
위에는 전부 잘 맞추었으며, 아래는 못 맞추었지만 AlexNet이 제시한 정답도 나름대로의 이유가 있다



# ImageNet Classification with Deep Convolutional Neural Networks

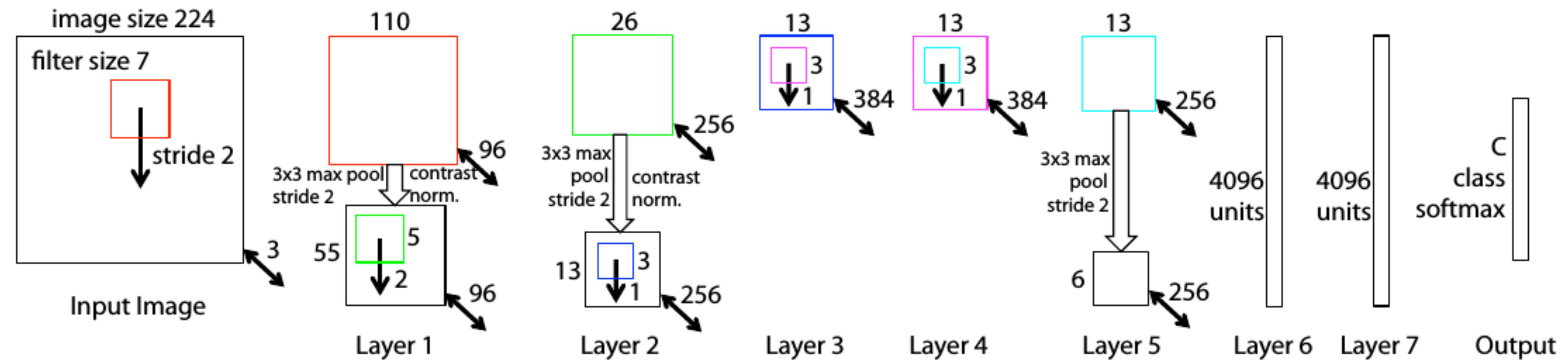
Krizhevsky et al, 2012. <https://goo.gl/hqAxZD>

# ZFNet

# ZFNet

ImageNet Challenge 2013년 우승 모델

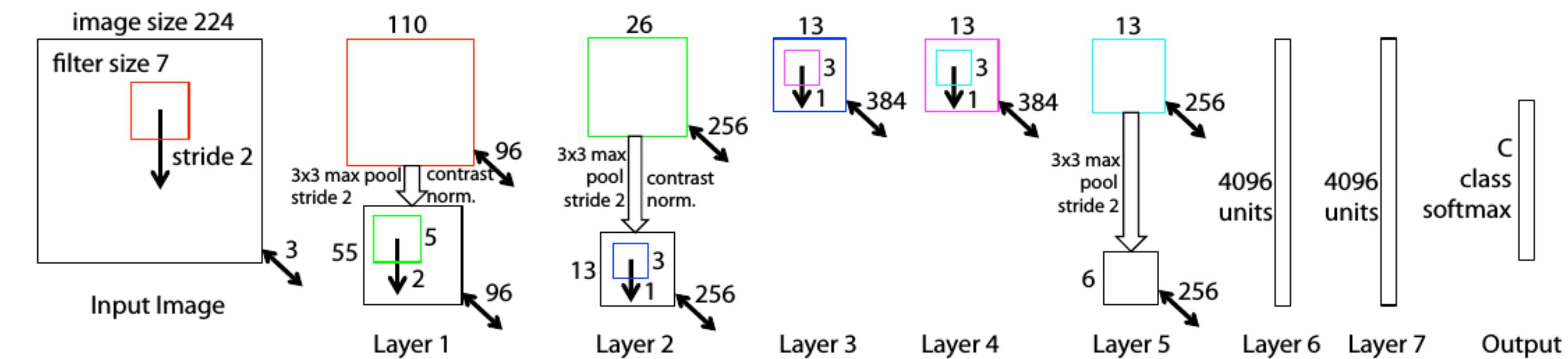
AlexNet에서 약간의 개선을 통해 성능을 향상시켰다. (16.4% -> 14.8%)



ImageNet Challenge 2013년 우승 모델  
AlexNet에서 약간의 개선을 통해 성능을 향상시켰다. (16.4% -> 14.8%)

## 변경 사항

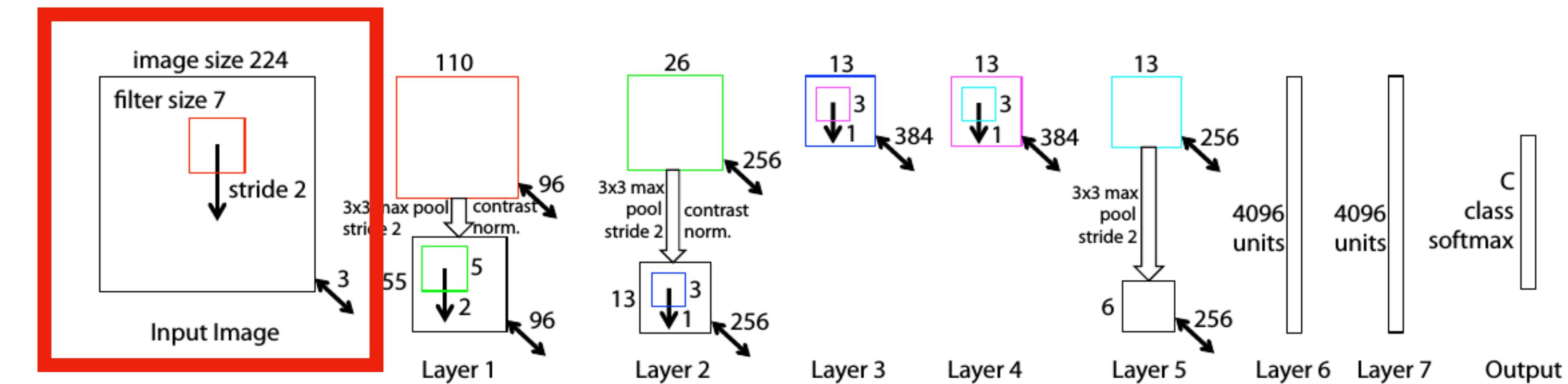
- 첫 Convolutional Layer
  - Before: 96@ $11 \times 11$ , stride 4
  - After: 96@ $7 \times 7$ , stride 2
- 3, 4, 5번째 Convolutional Layer
  - Before: depth가 384, 384, 256
  - After: depth가 512, 1024, 512



ImageNet Challenge 2013년 우승 모델  
AlexNet에서 약간의 개선을 통해 성능을 향상시켰다. (16.4% -> 14.8%)

## 변경 사항

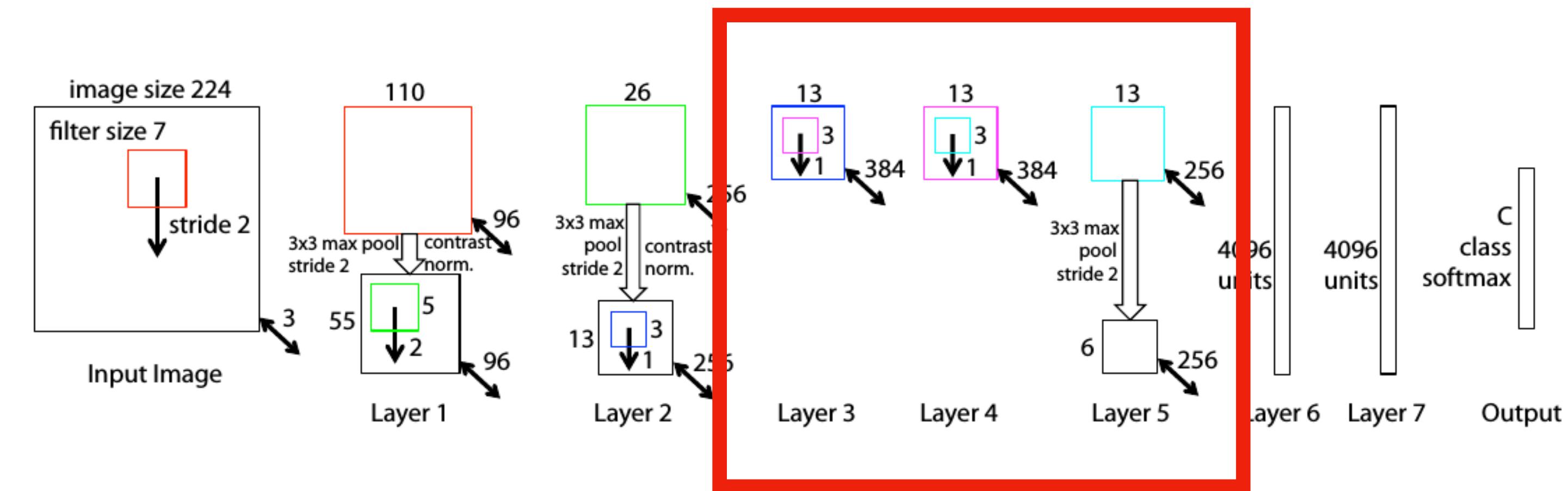
- 첫 Convolutional Layer
  - Before: 96@ $11 \times 11$ , stride 4
  - **After: 96@ $7 \times 7$ , stride 2**
- 3, 4, 5번째 Convolutional Layer
  - Before: depth가 384, 384, 256
  - After: depth가 512, 1024, 512



ImageNet Challenge 2013년 우승 모델  
AlexNet에서 약간의 개선을 통해 성능을 향상시켰다. (16.4% -> 14.8%)

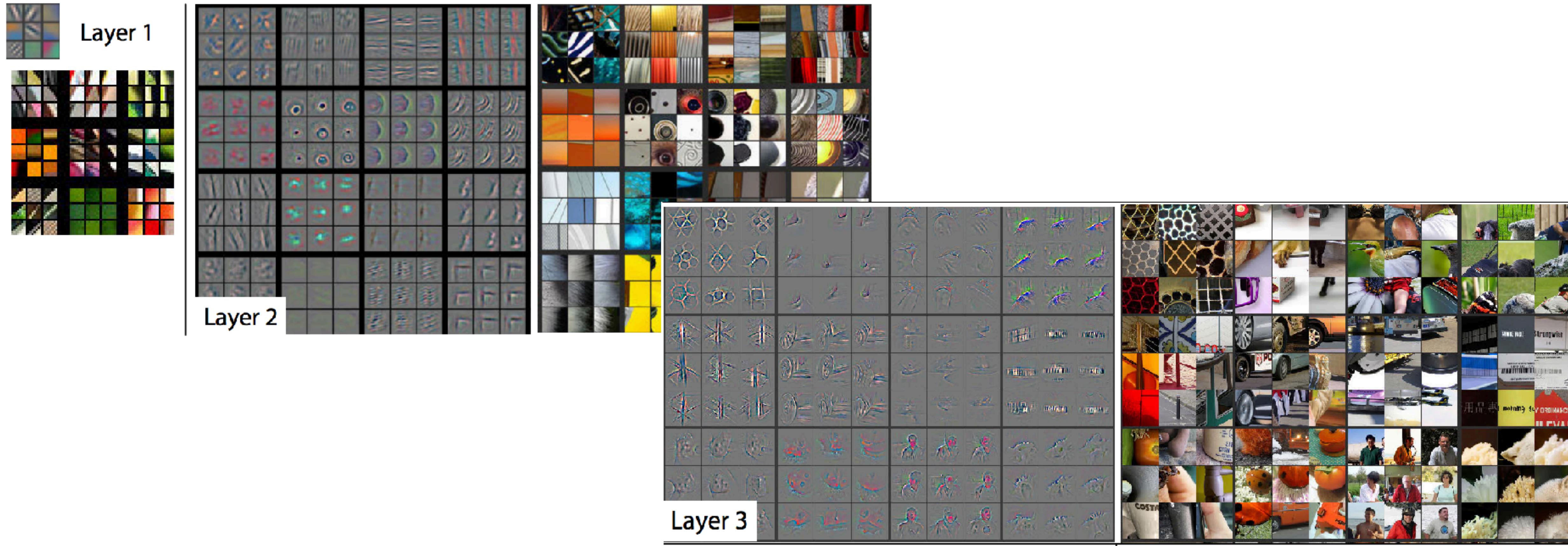
## 변경 사항

- 첫 Convolutional Layer
  - Before: 96@ $11 \times 11$ , stride 4
  - After: 96@ $7 \times 7$ , stride 2
- 3, 4, 5번째 Convolutional Layer
  - Before: depth가 384, 384, 256
  - **After: depth가 512, 1024, 512**



# ZFNet

ZFNet은 딥러닝 모델의 성능 개선에 새로운 가능성을 제시하진 않았지만, Convolutional Layer를 시각화하여 그 결과를 분석하는 시도를 했다는 것에 더 의의가 있다.

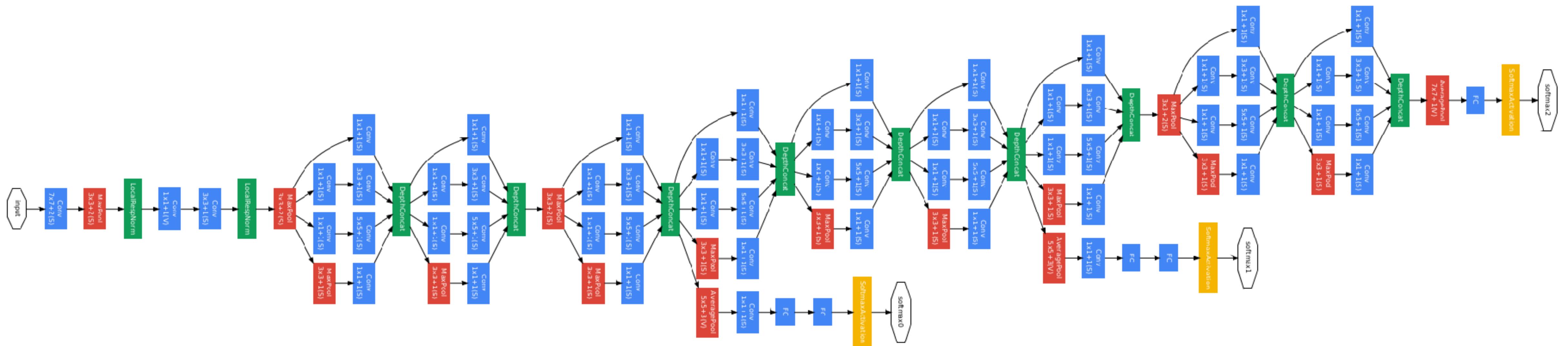


# GoogLeNet

# GoogLeNet

ImageNet Challenge 2014년 우승 모델

InceptionNet0이라고 불리운다. ZFNet에 비해 비약적인 성능 향상을 보여주었다. (14.8% -> 6.7%)

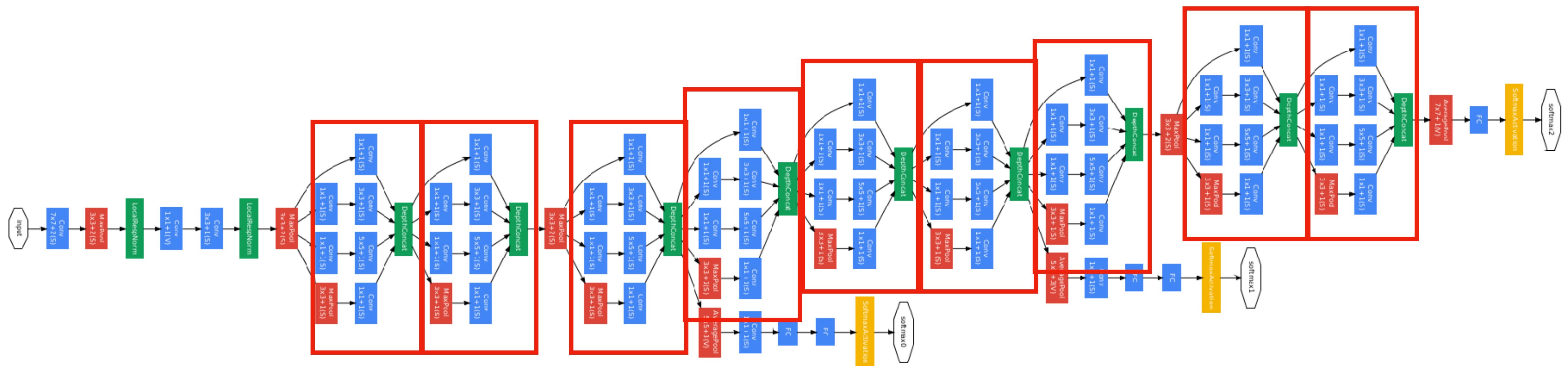


Going Deeper with Convolutions

Szegedy et al, 2014. <https://goo.gl/FA9ebp>

# GoogLeNet

얼핏 보면 굉장히 복잡해 보이지만, 자세히 보면 같은 레이어가 여러장 겹쳐 있는 것을 알 수 있다.  
이 레이어를 Inception 모듈이라고 부른다



# GoogLeNet

Inception 모듈은 사용할 수 있을법만 모든 사이즈의 Convolutional Layer( $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$ )를 한 번에 집어넣은 모듈이다. 이렇게 하면 이미지에 맞는 레이어를 딥러닝 모델이 스스로 선택할 수 있으며, 이미지에 사이즈에 구애받지 않고 예측할 수 있다.

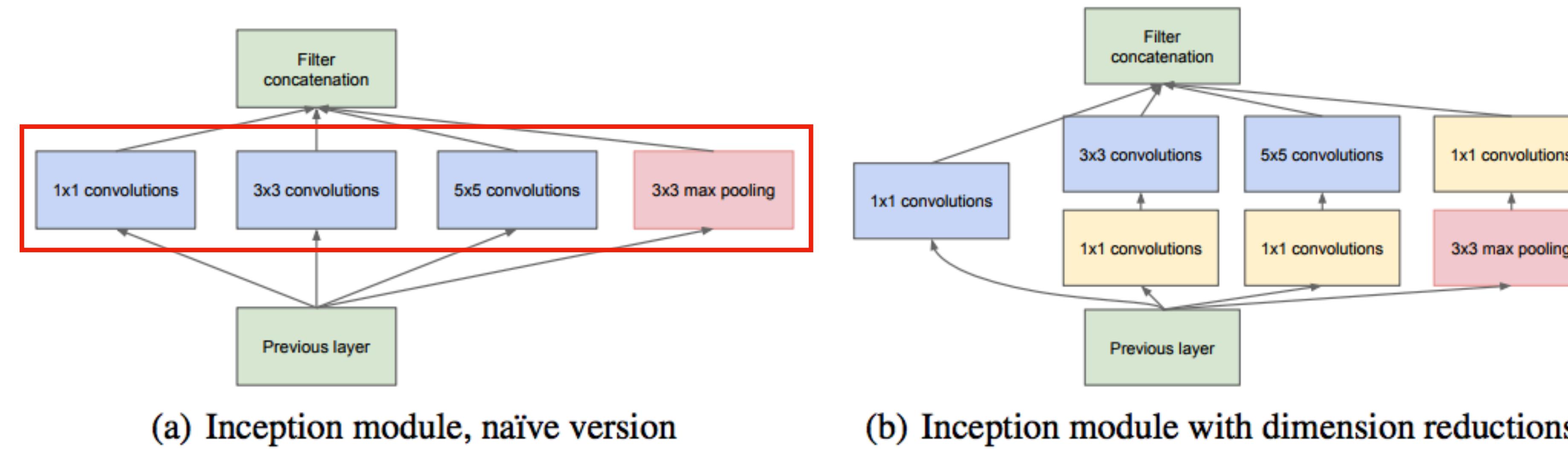


Figure 2: Inception module

# GoogLeNet

Inception 모듈은 사용할 수 있을법만 모든 사이즈의 Convolutional Layer( $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$ )를 한 번에 집어넣은 모듈이다. 이렇게 하면 이미지에 맞는 레이어를 딥러닝 모델이 스스로 선택할 수 있으며, 이미지에 사이즈에 구애받지 않고 예측할 수 있다.

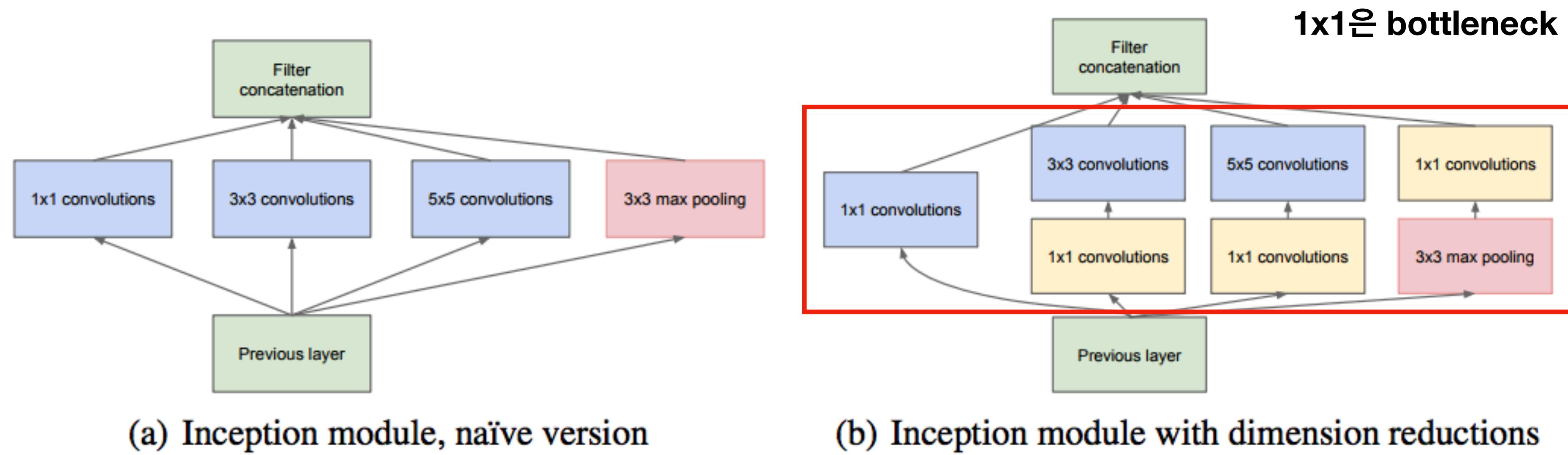


Figure 2: Inception module

# VGGNet

# VGGNet

ImageNet Challenge 2014년 준우승 모델(6.8% error rate)  
GoogleNet에 비해 굉장히 간결하며 직관적이다. 또한 Single Model의 경우 GoogLeNet보다 성능이 높다!

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 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	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 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# VGGNet

ImageNet Challenge 2014년 준우승 모델(6.8% error rate)  
GoogleNet에 비해 굉장히 간결하며 직관적이다. 또한 Single Model의 경우 GoogLeNet보다 성능이 높다!

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 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	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 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

실전에서는 여기서 D모델만 사용한다.

# VGGNet

VGGNet의 핵심은 Convolutional Layer의 사이즈  
3x3의 작은 Conv Layer를 가능한 깊게(deep) 쌓는 것이 효율적이라는 것을 증명하였다

## 학습 방식

- A모델(3x3 Conv, 11 layers)을 학습시킨다
- A모델에서 2개의 레이어를 추가한 뒤(13 layers) 다시 한 번 학습시킨다.  
여기서 이전에 학습한 11개의 layer의 weight값은 유지한다. 이 모델을  
**B모델**이라고 명명한다.
- 같은 방식으로 레이어를 3개 추가해 C모델, D모델을 만들어 학습시킨다.
- 같은 방식으로 레이어를 3개 추가해 E모델을 학습 시킨다.

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

VGGNet의 핵심은 Convolutional Layer의 사이즈  
3x3의 작은 Conv Layer를 가능한 깊게(deep) 쌓는 것이 효율적이라는 것을 증명하였다

## 학습 방식

- A모델(3x3 Conv, 11 layers)을 학습시킨다
- A모델에서 2개의 레이어를 추가한 뒤(13 layers) 다시 한 번 학습시킨다.  
여기서 이전에 학습한 11개의 layer의 weight값은 유지한다. 이 모델을  
**B모델**이라고 명명한다.
- 같은 방식으로 레이어를 3개 추가해 C모델, D모델을 만들어 학습시킨다.
- 같은 방식으로 레이어를 3개 추가해 E모델을 학습 시킨다.

Table 4: ConvNet performance at multiple test scales.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train ( $S$ )	test ( $Q$ )		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	<b>24.8</b>	7.5
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	<b>24.8</b>	7.5

학습 결과 레이어를 추가할수록 점점 성능이 올라가는 것을 볼 수 있다.  
(단 D모델에서 E모델로 넘어갈 땐 성능 향상이 일어나지 않는다)

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	conv3-256 conv3-256 conv1-256	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	conv3-512 conv3-512 conv1-512	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	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096	FC-4096	FC-4096	FC-4096	FC-1000	soft-max

# VGGNet

VGGNet의 핵심은 Convolutional Layer의 사이즈  
3x3의 작은 Conv Layer를 가능한 깊게(deep) 쌓는 것이 효율적이라는 것을 증명하였다

## Layer Configuration

- Convolutional Layer는 언제나 3x3, 1 stride, 1 pad.
- Convolutional Layer의 depth는 64로 시작해서 두 배수로 늘어남.
- 1x1 convolutional layer를 사용했으나 여기서 큰 효과는 없었다. (하지만 나중에 효과를 발휘한다!)
- Max pooling은 언제나 2x2, 2 strides.
- Dropout은 FC layer에만 적용. Dropout rate는 0.5. (50%)

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	conv3-64	conv3-64	conv3-64	conv3-64
LRN					
maxpool					
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	conv3-256	conv3-256	conv3-256	conv3-256
conv1-256					
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv1-512					
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv1-512					
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# VGGNet

VGGNet의 핵심은 Convolutional Layer의 사이즈  
3x3의 작은 Conv Layer를 가능한 깊게(deep) 쌓는 것이 효율적이라는 것을 증명하였다

## Layer Configuration

- Convolutional Layer는 언제나 3x3, 1 stride, 1 pad.
- Convolutional Layer의 depth는 64로 시작해서 두 배수로 늘어남.
- 1x1 convolutional layer를 사용했으나 여기서 큰 효과는 없었다. (하지만 나중에 효과를 발휘한다!)
- Max pooling은 언제나 2x2, 2 strides.
- Dropout은 FC layer에만 적용. Dropout rate는 0.5. (50%)

## Other Configuration

- Optimizer는 SGD, learning rate는 0.1 (x10), momentum은 0.9 (동일), weight decay는 0.005 (x10)
- error가 줄어들지 않을 때마다 learning\_rate를 1/10로 줄임. 이를 3번 반복. (동일)
- Weight는 Gaussian Distribution으로 초기화. 평균은 0, 분산은 0.1. (다름)
- 평소에는 D모델을 사용(7.2%), 성능을 더 높이고 싶으면 Ensemble을 통해 D모델과 E모델을 섞어서 활용(6.8%)
- D모델은 GoogLeNet의 단일 모델에 비해 성능이 더 높으며( $7.0\% > 7.9\%$ ), 양상 모델도 GoogLeNet에 비해 0.1%밖에 차이가 나지 않는다. ( $6.8\% < 6.7\%$ )

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
maxpool					
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 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 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 conv3-512 conv3-512
maxpool					
FC-4096	FC-4096	FC-4096	FC-1000		soft-max

## VGG는 학습할 때 굉장히 다양한 Data Preprocessing과 Augmentation을 했다. 또한 이 결과를 면밀하게 벤치마킹 하여 결과를 공개하였다.

### Preprocessing

- 이미지를 집어넣을 때 RGB값 각각의 평균을 뺀 뒤 집어넣었다. (이렇게 하면 loss가 빨리 수렴한다)
- 이미지 사이즈는 256x256px만 하지 않고, 512x512px도 변환하여 집어넣었다. 여기서 224x224px로 랜덤하게 crop하여 딥러닝 모델에 학습시킨다.
- 학습할 때 랜덤하게 좌우 반전을 한다. 테스트 할 때는 일반 버전과 좌우 반전 버전 두 개를 동시에 넣고, 그 결과의 평균을 이용한다.
- 다양한 시도를 해보았으나, 결과적으로는 256x256px 과 512x512px을 다 넣고 학습한 뒤, 테스트 할 때도 256x256px, 384x384px, 512x512px을 다 넣어서 평균을 내는 것이 점수가 좋았다.

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 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# VGGNet

**VGG는 학습할 때 굉장히 다양한 Data Preprocessing과 Augmentation을 했다.  
또한 이 결과를 면밀하게 벤치마킹 하여 결과를 공개하였다.**

Table 3: ConvNet performance at a single test scale.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train ( $S$ )	test ( $Q$ )		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	<b>25.5</b>	<b>8.0</b>

Single Scale Evaluation (하나의 가로x세로 사이즈로 결과를 측정)

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	conv3-64	conv3-64	conv3-64	conv3-64					
LRN	<b>conv3-64</b>	<b>conv3-64</b>	conv3-64	conv3-64	conv3-64					
maxpool										
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128					
	<b>conv3-128</b>	<b>conv3-128</b>	conv3-128	conv3-128	conv3-128					
maxpool										
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256					
<b>conv1-256</b>										
<b>conv3-256</b>										
maxpool										
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512					
<b>conv1-512</b>										
<b>conv3-512</b>										
maxpool										
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512					
<b>conv1-512</b>										
<b>conv3-512</b>										
maxpool										
FC-4096										
FC-4096										
FC-1000										
soft-max										

# VGGNet

VGG는 학습할 때 굉장히 다양한 Data Preprocessing과 Augmentation을 했다.

또한 이 결과를 면밀하게 벤치마킹 하여 결과를 공개하였다.

Table 3: ConvNet performance at a single test scale.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train ( $S$ )	test ( $Q$ )		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	<b>25.5</b>	<b>8.0</b>

Single Scale Evaluation (하나의 가로x세로 사이즈로 결과를 측정)

Table 4: ConvNet performance at multiple test scales.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train ( $S$ )	test ( $Q$ )		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256;512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256;512]	256,384,512	<b>24.8</b>	<b>7.5</b>
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256;512]	256,384,512	<b>24.8</b>	<b>7.5</b>

Single Scale Evaluation (여러개의 가로x세로 사이즈로 결과를 측정)

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	conv3-64	conv3-64	conv3-64	conv3-64
LRN		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	conv3-256	conv3-256	conv3-256	conv3-256
			conv1-256	conv3-256	conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	conv3-512	conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	conv3-512	conv3-512
maxpool					
FC-4096					
	FC-4096				
		FC-1000			
			soft-max		

Very Deep Convolutional Networks for Large-Scale Image Recognition

Simonyan and Zisserman, 2014. <https://goo.gl/mbKWhq>

# VGGNet

VGG는 학습할 때 굉장히 다양한 Data Preprocessing과 Augmentation을 했다.  
또한 이 결과를 면밀하게 벤치마킹 하여 결과를 공개하였다.

Table 5: ConvNet evaluation techniques comparison. In all experiments the training scale  $S$  was sampled from [256; 512], and three test scales  $Q$  were considered: {256, 384, 512}.

ConvNet config. (Table 1)	Evaluation method	top-1 val. error (%)	top-5 val. error (%)
D	dense	24.8	7.5
	multi-crop	24.6	7.5
	multi-crop & dense	<b>24.4</b>	<b>7.2</b>
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	<b>24.4</b>	7.1

이미지를 다양하게 잘라넣어서 측정한 결과

Table 6: Multiple ConvNet fusion results.

Combined ConvNet models	Error		
	top-1 val	top-5 val	top-5 test
ILSVRC submission			
(D/[256;256,288], (D/[384;352,384,416], (D/[256;512]/[256,384,512])			
(C/[256;256,288], (C/[384;352,384,416])	24.7	7.5	7.3
(E/[256;256,288], (E/[384;352,384,416])			
post-submission			
(D/[256;512]/[256,384,512], (E/[256;512]/[256,384,512]), dense eval.	24.0	7.1	7.0
(D/[256;512]/[256,384,512], (E/[256;512]/[256,384,512]), multi-crop	23.9	7.2	-
(D/[256;512]/[256,384,512], (E/[256;512]/[256,384,512]), multi-crop & dense eval.)	<b>23.7</b>	<b>6.8</b>	<b>6.8</b>

양상을 결과. D모델과 E모델을 섞은 것이 가장 성능이 높았다

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
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
LRN	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	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	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	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	conv3-512	conv3-512	conv3-512	conv3-512
maxpool					
FC-4096	FC-4096	FC-4096	FC-4096	FC-4096	FC-4096
FC-1000	FC-1000	FC-1000	FC-1000	FC-1000	FC-1000
soft-max					

# VGGNet

VGG는 학습할 때 굉장히 다양한 Data Preprocessing과 Augmentation을 했다.  
또한 이 결과를 면밀하게 벤치마킹 하여 결과를 공개하였다.

Table 7: Comparison with the state of the art in ILSVRC classification. Our method is denoted as “VGG”. Only the results obtained without outside training data are reported.

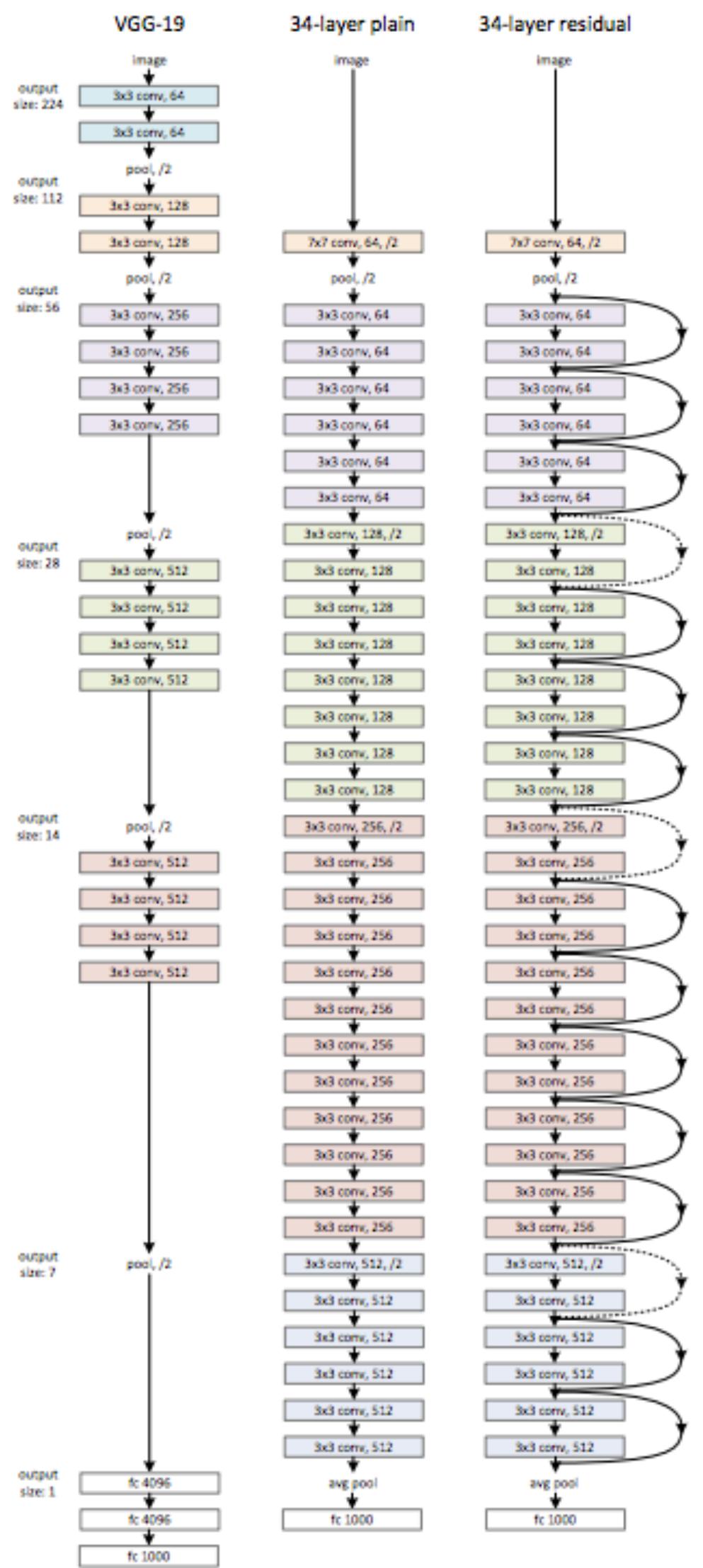
Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	<b>23.7</b>	<b>6.8</b>	<b>6.8</b>
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	6.7	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

결과적으로는 GoogLeNet에 0.1% 모잘랐지만  
단일 모델로는 GoogLeNet에 0.9%가까이 좋은 성능을 발휘하였다.

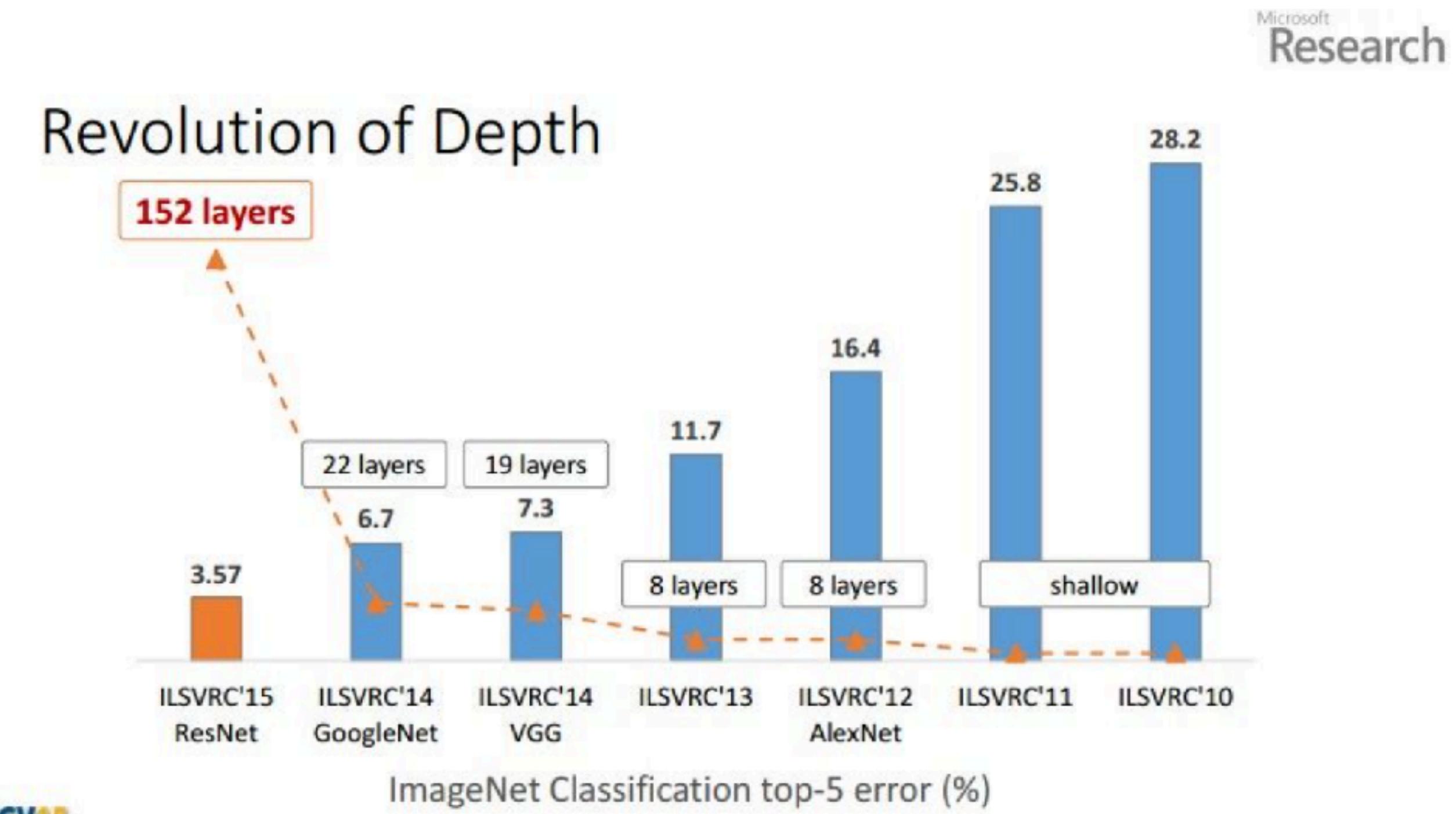
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128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

# ResNet

# ResNet(Deep Residual Network)



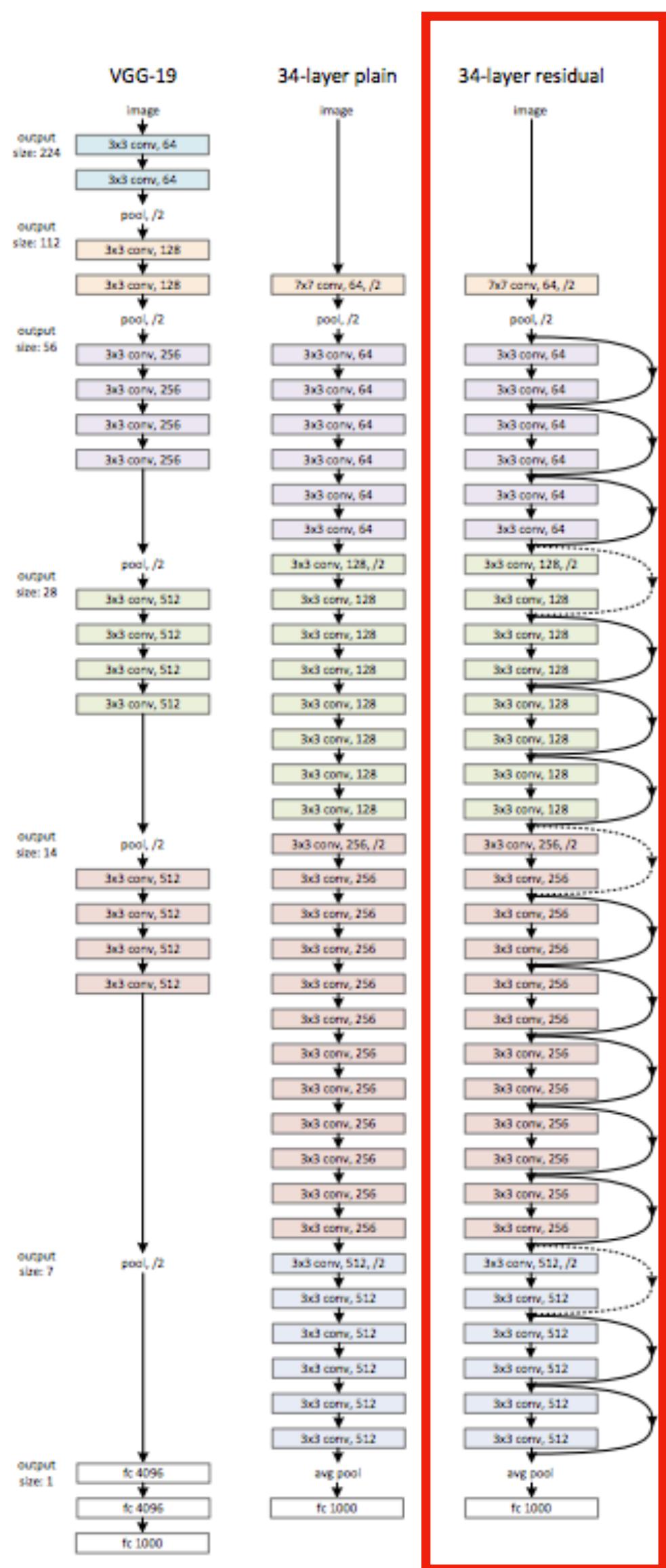
**ImageNet Challenge 2015년 우승 모델. (3.57% error rate)**  
인간의 한계(5%)를 돌파한 최초의 Image Classification 모델이 되었다



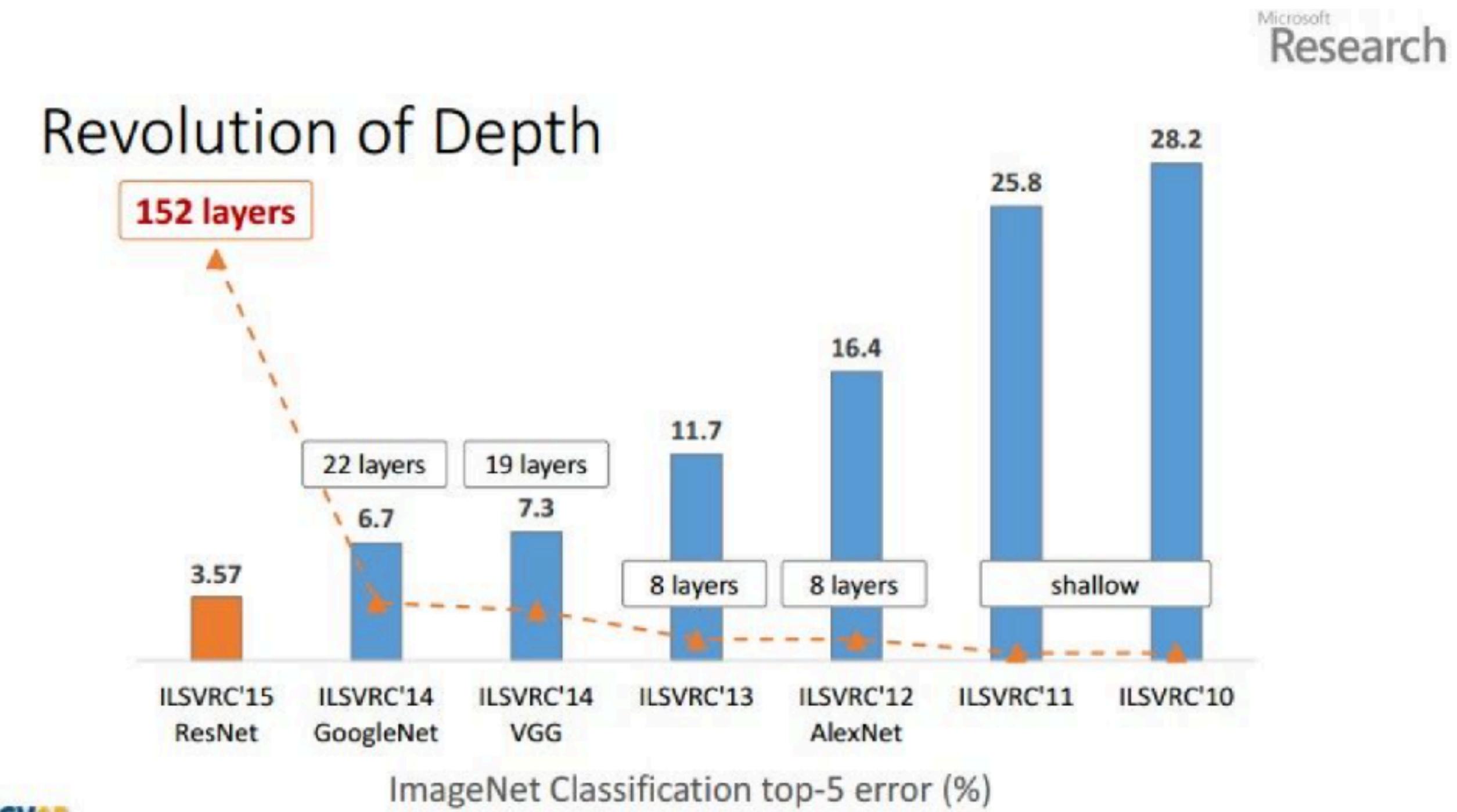
**GoogLeNet(22 layers)0|나 VGGNet(19 layers)에 비해  
압도적으로 많은 레이어를 쌓아 우승했다. (152 layers)**

**Deep Residual Learning for Image Recognition**  
He et al, 2015. <https://goo.gl/ZCZVgB>

# ResNet(Deep Residual Network)



ImageNet Challenge 2015년 우승 모델. (3.57% error rate)  
인간의 한계(5%)를 돌파한 최초의 Image Classification 모델이 되었다



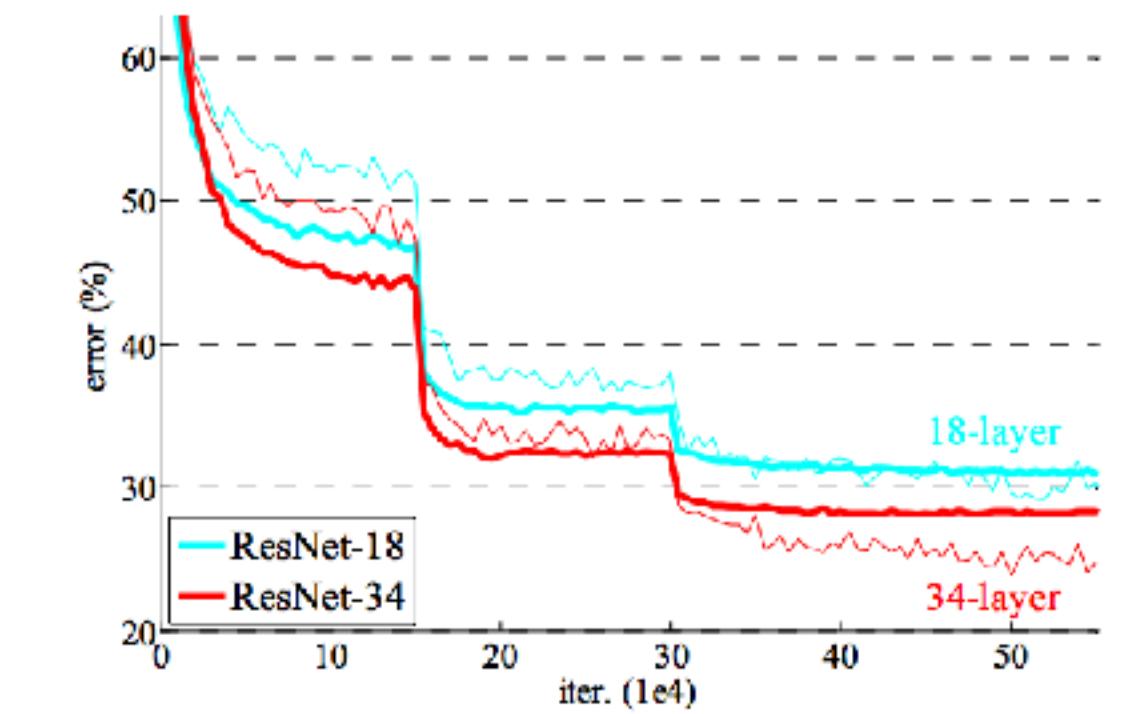
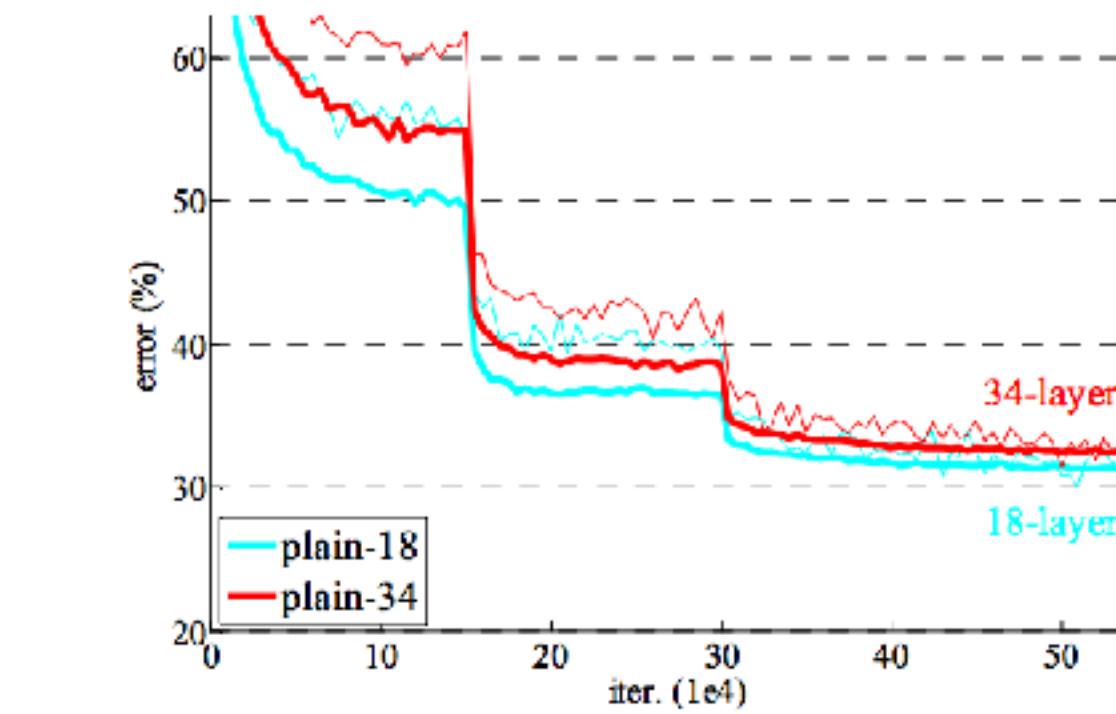
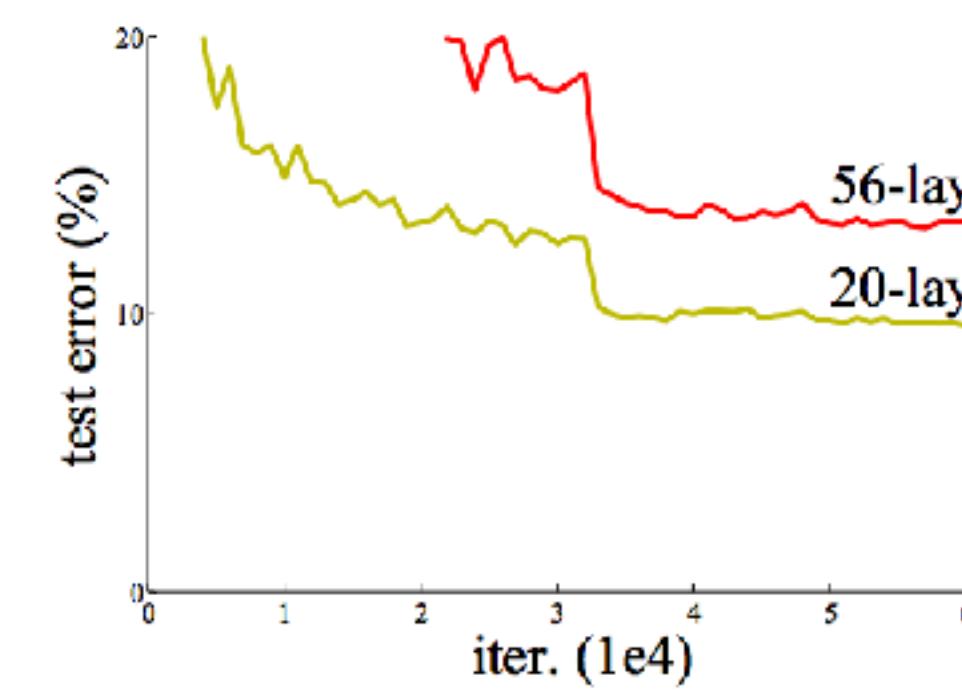
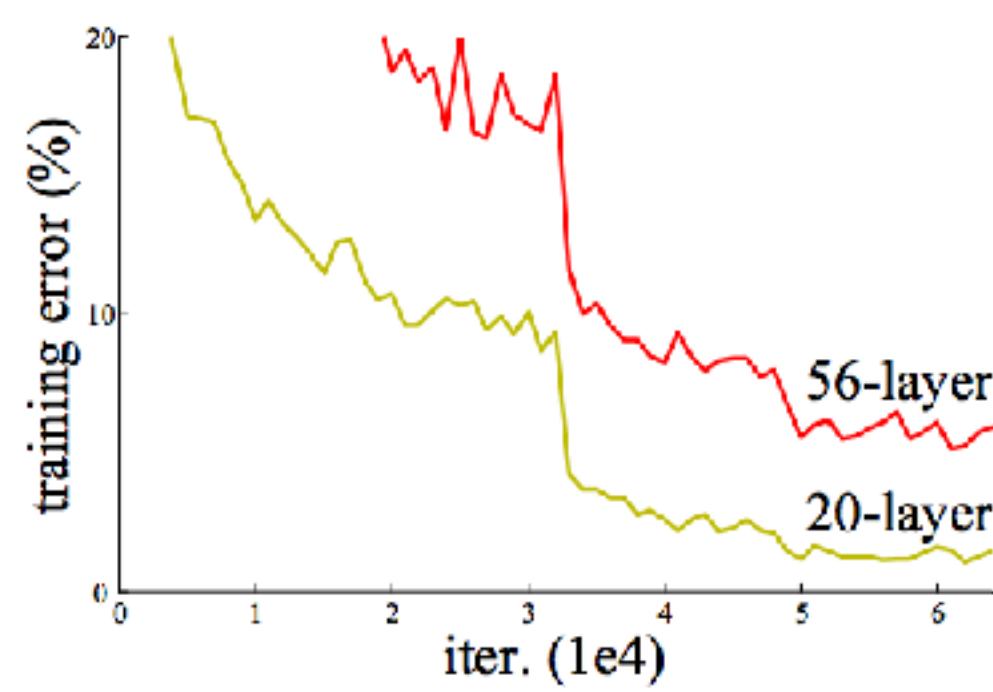
GoogLeNet(22 layers)이나 VGGNet(19 layers)에 비해  
압도적으로 많은 레이어를 쌓아 우승했다. (152 layers)

Deep Residual Learning for Image Recognition  
He et al, 2015. <https://goo.gl/ZCZVgB>

# ResNet(Deep Residual Network)

이전까지는 layer를 더 깊게 쌓고 싶어도 딥러닝 모델이 제대로 학습되지 않는 현상이 발생했다. (degradation problem)

ResNet은 레이어에 Skip-Connection이라는 개념을 추가한 Residual Block이라는 개념을 도입하여 해결하였다

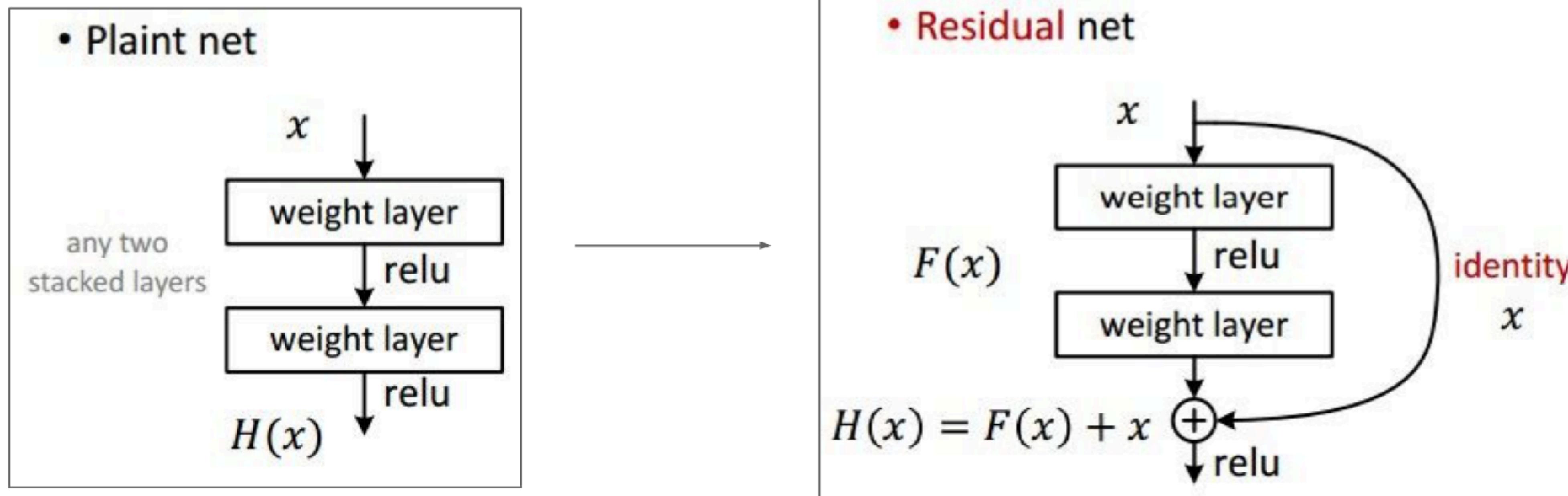


레이어가 20개에서 56개로 늘어나면  
training error와 test error가 동시에 늘어난다.

하지만 Residual Block을 적용하면  
레이어를 더 늘려도 train erro와 test error가 떨어진다. (우측)

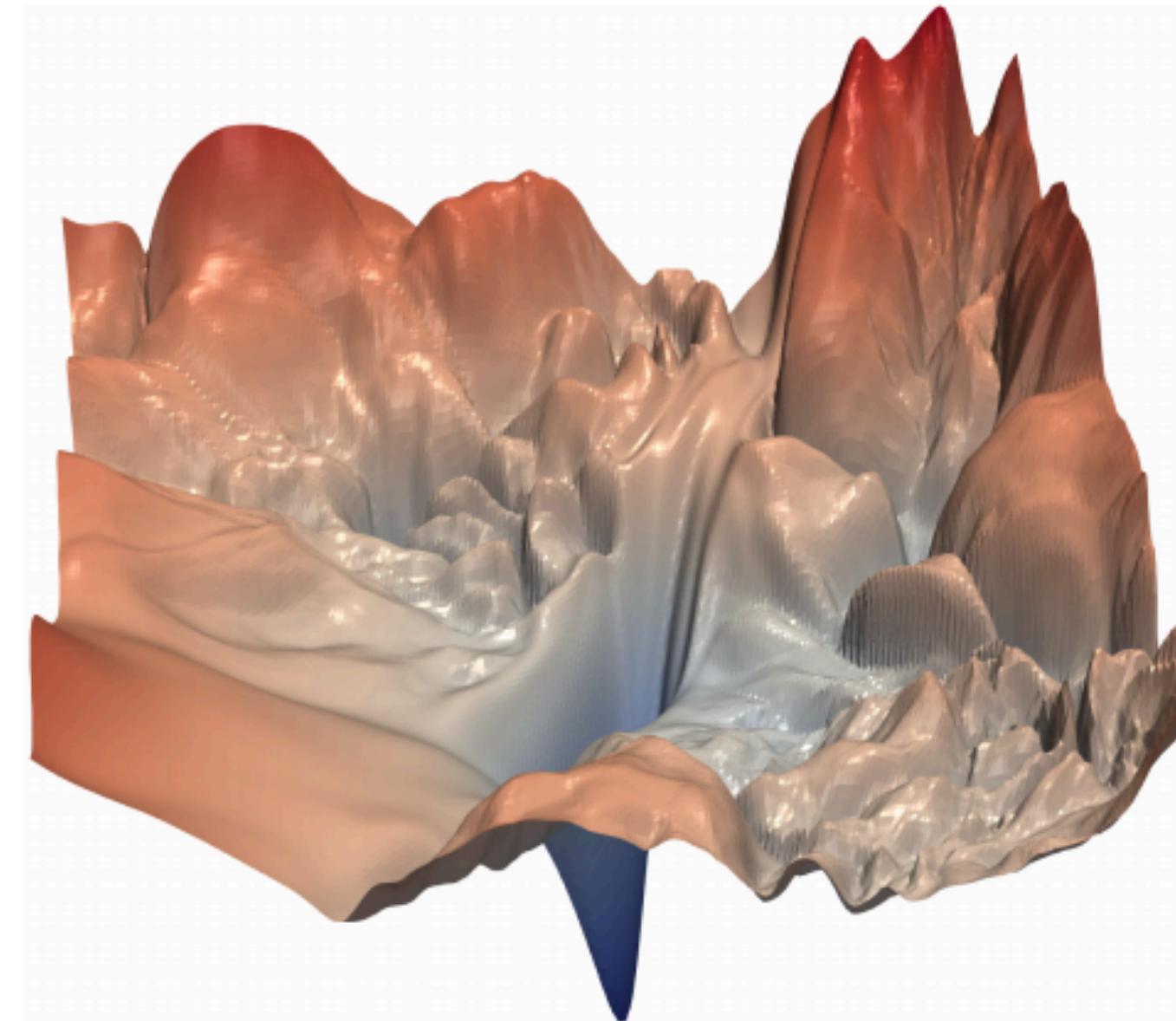
# ResNet(Deep Residual Network)

Residual block은 레이어의 앞에 있던 이전 Input을 그대로 더하는 skip-connection을 개념을 도입한 개념이다 이를 통해 Hidden Layer가 “Input으로 Feature를 만든다”가 아닌, “Input에 변화를 주는 additional한 값을 적용한다.”로 바뀐다.

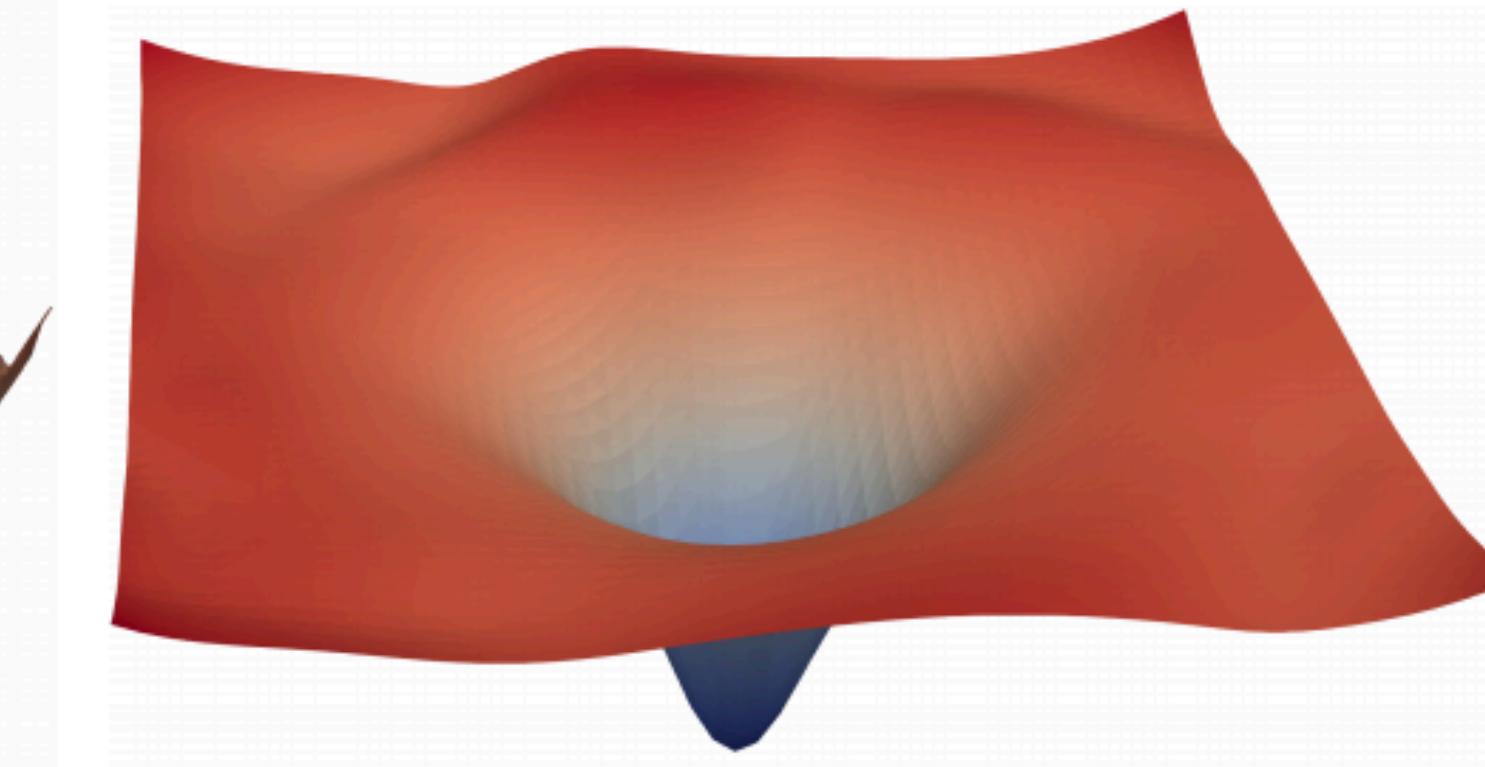


# ResNet(Deep Residual Network)

그러므로 skip-connection을 적용하면  
Hidden Layer가 새로운 Feature를 만들어내야 하는 부담이 줄어들기 때문에 훨씬 더 효율적인 training이 가능하다.



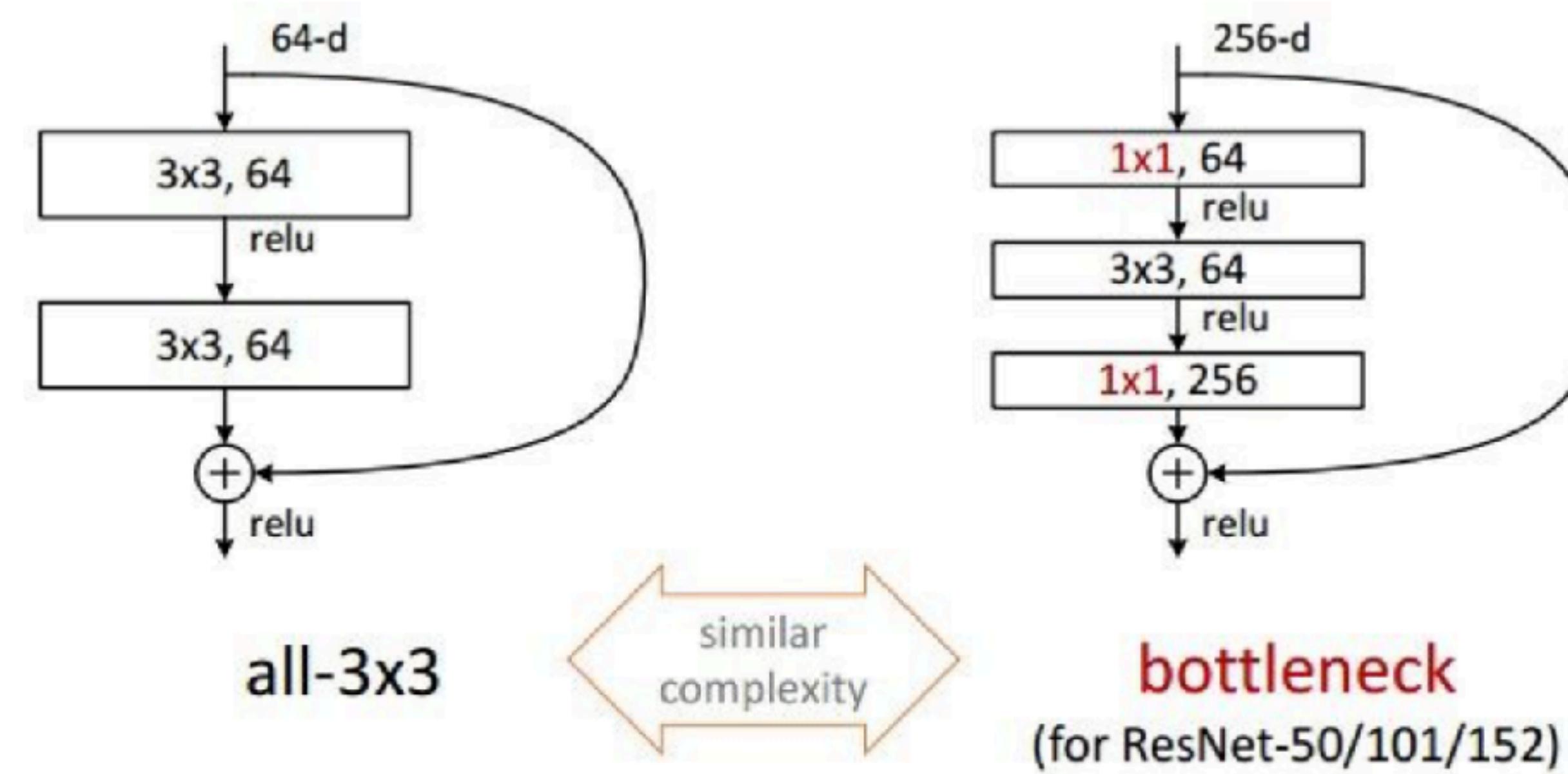
(a) without skip connections



(b) with skip connections

# ResNet(Deep Residual Network)

또한  $1 \times 1$  Convolutional layer를 활용하여 Feature Map의 차원을 축소하는 (dimensionality reduction) bottleneck 전략을 적용하였다.



# ResNet(Deep Residual Network)

기존 VGG와 거의 동일한 설정을 바탕으로, skip-connection을 넣은 residual block을 최대한 많이 쌓는다  
총 152개의 레이어를 쌓을 수 있으며, 이를 통해 모델의 성능을 크게 향상시킬 수 있었다



## Layer Configuration

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

## Training Configuration

- 모든 레이어에 Batch Normalization을 적용.
- Xavier을 바탕으로 한 He Initialization 적용. (본인이 직접 만들었다)
- Optimizer는 SGD, momentum은 0.9, weight decay는 1e-5
- Learning은 0.1, 마찬가지로 error가 줄어들지 않을 때마다 1/10로 감소시킨다.
- Dropout은 사용하지 않는다.

# ResNet(Deep Residual Network)

기존 VGG와 거의 동일한 설정을 바탕으로, skip-connection을 넣은 residual block을 최대한 많이 쌓는다  
총 152개의 레이어를 쌓을 수 있으며, 이를 통해 모델의 성능을 크게 향상시킬 수 있었다



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	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	<b>19.38</b>	<b>4.49</b>

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

단일 모델 기준 4.49%,  
앙상블 모델 기준 3.57%를 달성

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
<b>ResNet (ILSVRC'15)</b>	<b>3.57</b>

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

# After ILSVRC2015

# After ILSVRC2015

이미지넷2015를 기점으로 이미지 분류 Task는 사라졌으나,  
많은 연구자들이 그 이후에도 새로운 이미지 분류 모델을 연구하고 있다.

Ordered by classification error

Team name	Entry description	Classification error	Localization error
Trimps-Soushen	Ensemble 2	0.02991	0.077668
Trimps-Soushen	Ensemble 3	0.02991	0.077087
Trimps-Soushen	Ensemble 4	0.02991	0.077429
ResNeXt	Ensemble C, weighted average, tuned on val. [No bounding box results]	0.03031	0.737308
CU-DeepLink	GrandUnion + Fused-scale EnsembleNet	0.03042	0.098892
CU-DeepLink	GrandUnion + Multi-scale EnsembleNet	0.03046	0.099006
CU-DeepLink	GrandUnion + Basic Ensemble	0.03049	0.098954

# After ILSVRC2015

이미지넷2015를 기점으로 이미지 분류 Task는 사라졌으나,  
많은 연구자들이 그 이후에도 새로운 이미지 분류 모델을 연구하고 있다.

Ordered by classification error

Team name	Entry description	Classification error	Localization error
Trimps-Soushen	Ensemble 2	0.02991	0.077668
Trimps-Soushen	Ensemble 3	0.02991	0.077087
Trimps-Soushen	Ensemble 4	0.02991	0.077429
ResNeXt	Ensemble C, weighted average, tuned on val. [No bounding box results]	0.03031	0.737308
CU-DeepLink	GrandUnion + Fused-scale EnsembleNet	0.03042	0.098892
CU-DeepLink	GrandUnion + Multi-scale EnsembleNet	0.03046	0.099006
CU-DeepLink	GrandUnion + Basic Ensemble	0.03049	0.098954

# Wide Residual Networks

모델의 깊이(depth)보다는 너비(width)를 늘리는 쪽이 더 효율도 좋고 정확도도 높게 나온다는 것을 검증하였다  
50레이어의 Wide ResNet이 152레이어의 ResNet보다 더 좋은 성능을 보인다

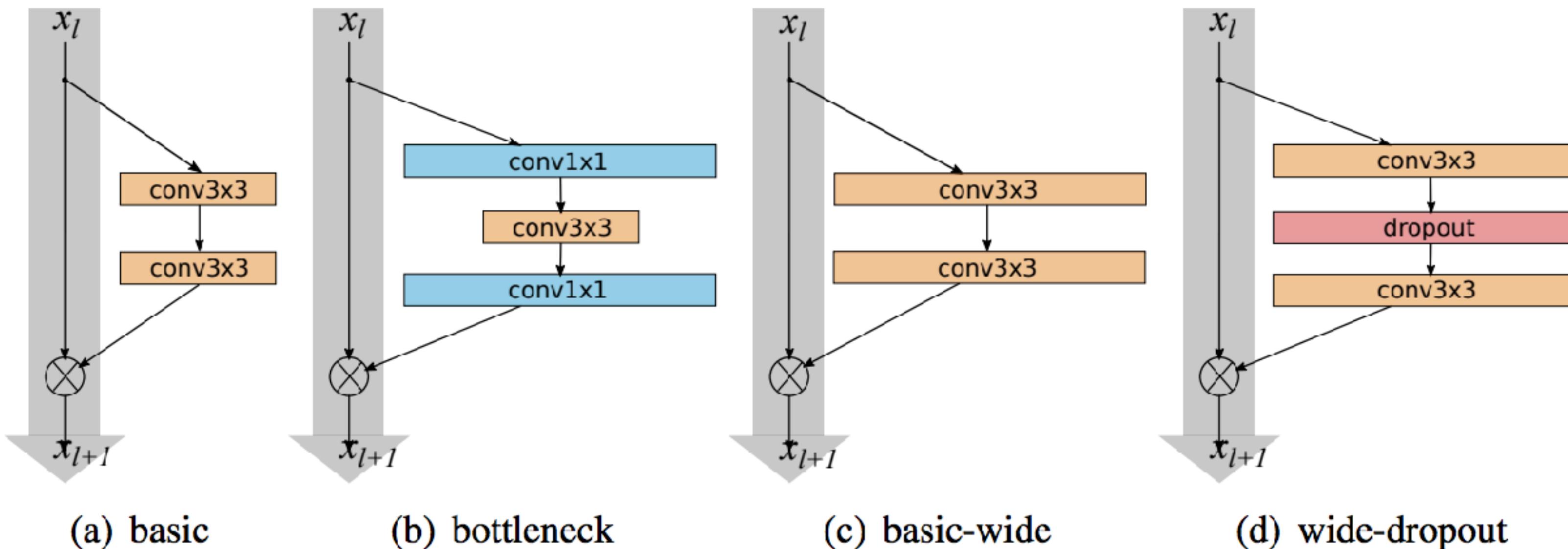
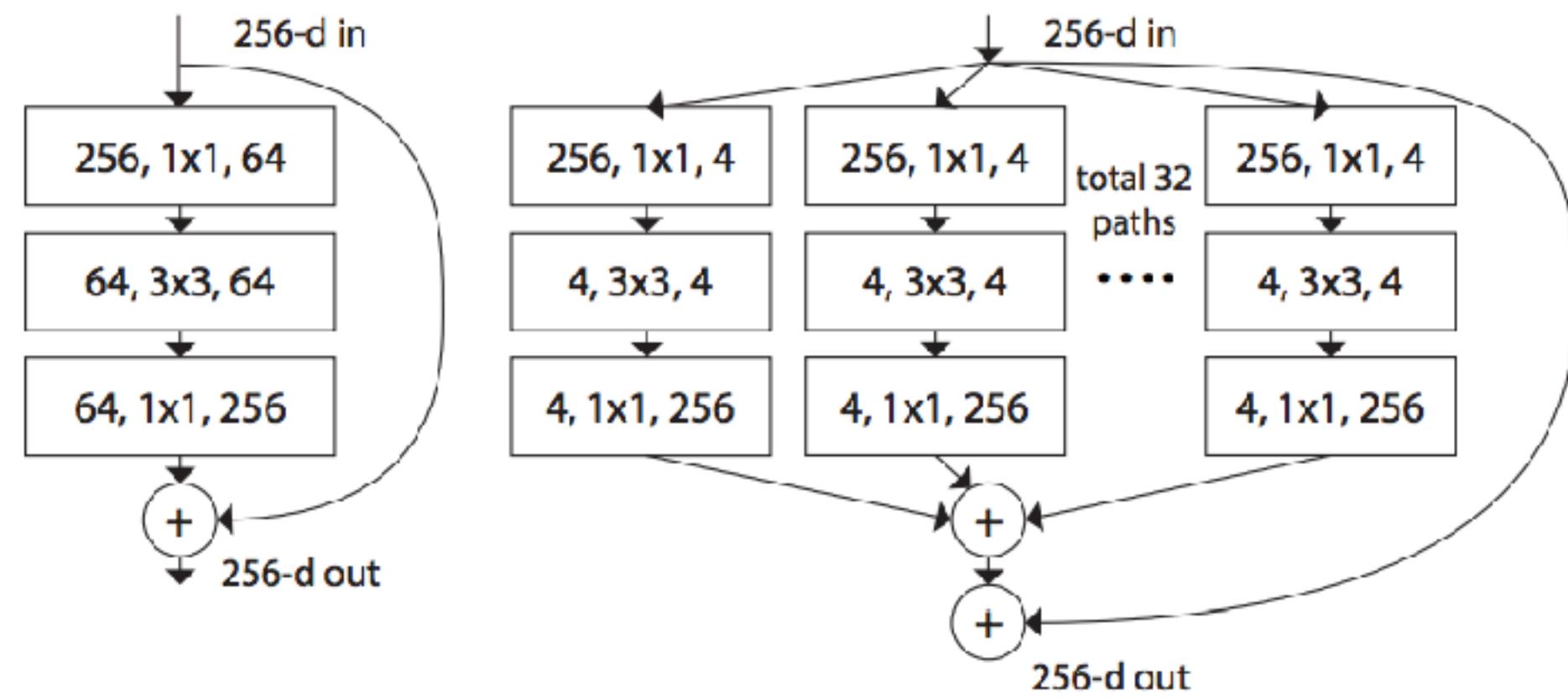


Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)

# ResNeXt

이미지넷2016 2등 모델 (3.03% top5 error rate)

한 층에 Convolutional Layer 여러개를 따로 연산한 뒤 최종적으로 합치는 모델로 구성하였다



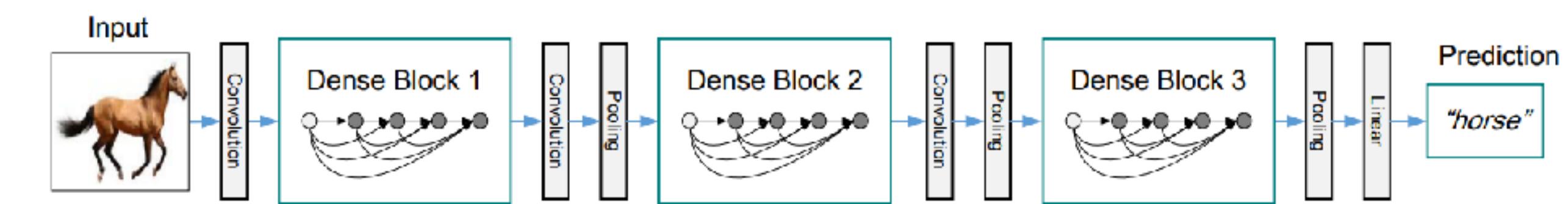
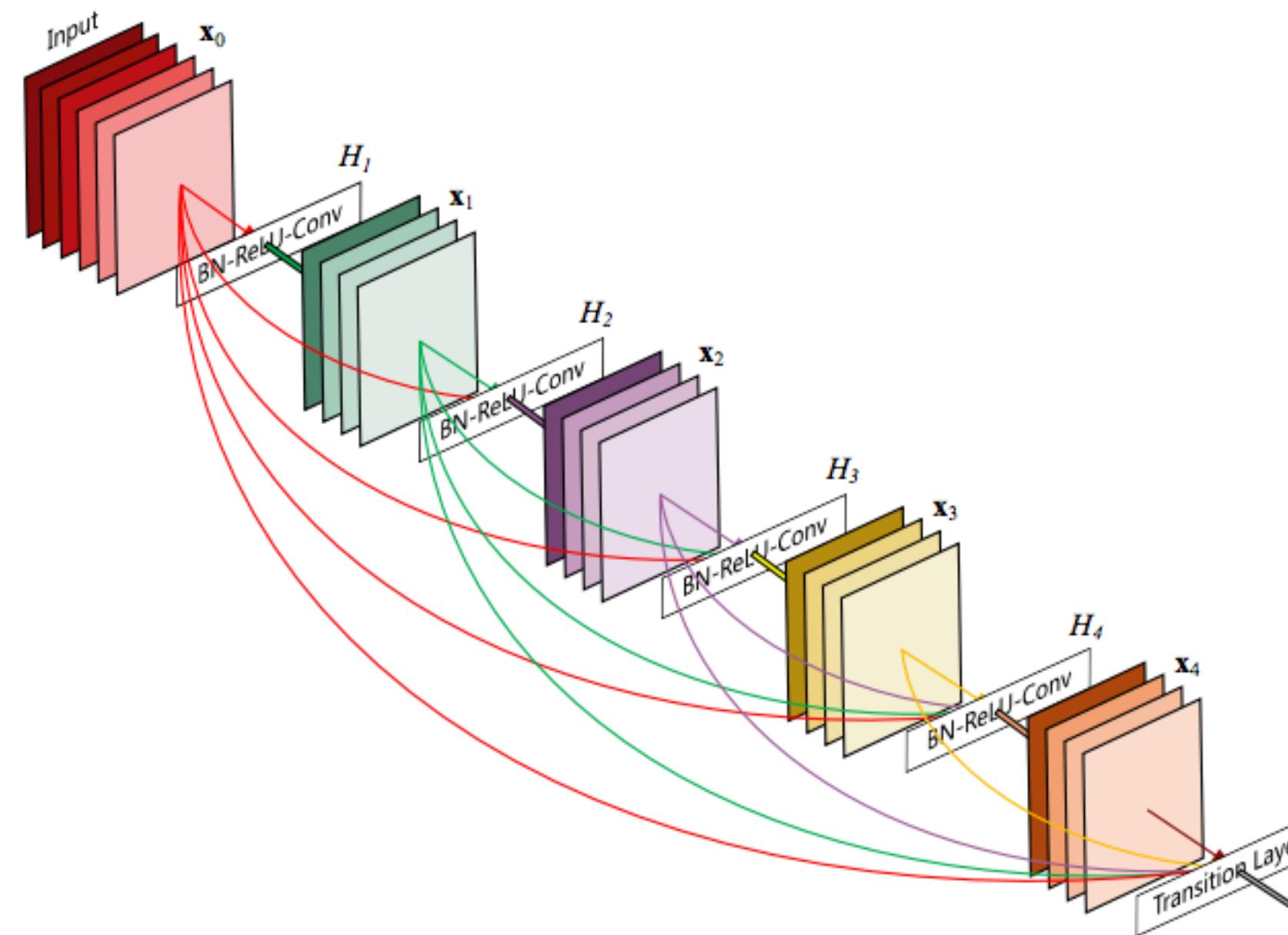
Convolutional Layer의 필터를  
하나만 쓰지 않고 여러개로 나눠서 쓰면 더 효율적으로 모델을 구성할 수 있다

stage	output	ResNet-50	ResNeXt-50 (32x4d)
conv1	112x112	7x7, 64, stride 2	7x7, 64, stride 2
conv2	56x56	3x3 max pool, stride 2 $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	3x3 max pool, stride 2 $\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128, C=32 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3	28x28	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256, C=32 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4	14x14	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512, C=32 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5	7x7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 1024 \\ 3 \times 3, 1024, C=32 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1x1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$25.5 \times 10^6$	$25.0 \times 10^6$
FLOPs		$4.1 \times 10^9$	$4.2 \times 10^9$

50개 레이어의 ResNet와 ResNeXt의 비교  
ResNeXt가 2배의 필터를 사용하지만 parameter와 연산량(FLOPS)은 동일하다

# Densely Connected Convolutional Networks

skip-connection이 효과를 발휘한다는 아이디어를 확장하여  
이전 레이어의 결과를 통채로 가져온 뒤 다음 레이어와 함께 연산



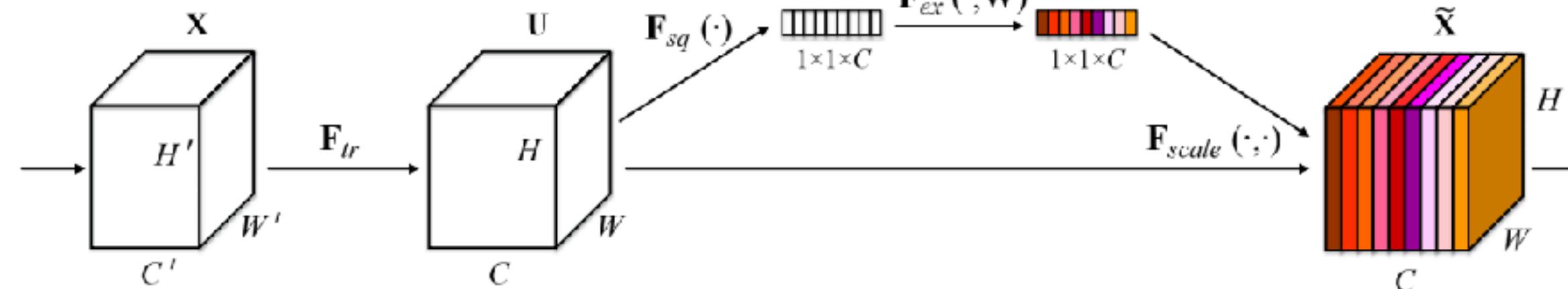
이전 레이어의 결과는 그 다음 레이어들에 언제나 반영된다.  
이를 Dense Block이라 명명한다

딥러닝 모델의 중간중간에 이 Dense Block을 끼워넣는다.

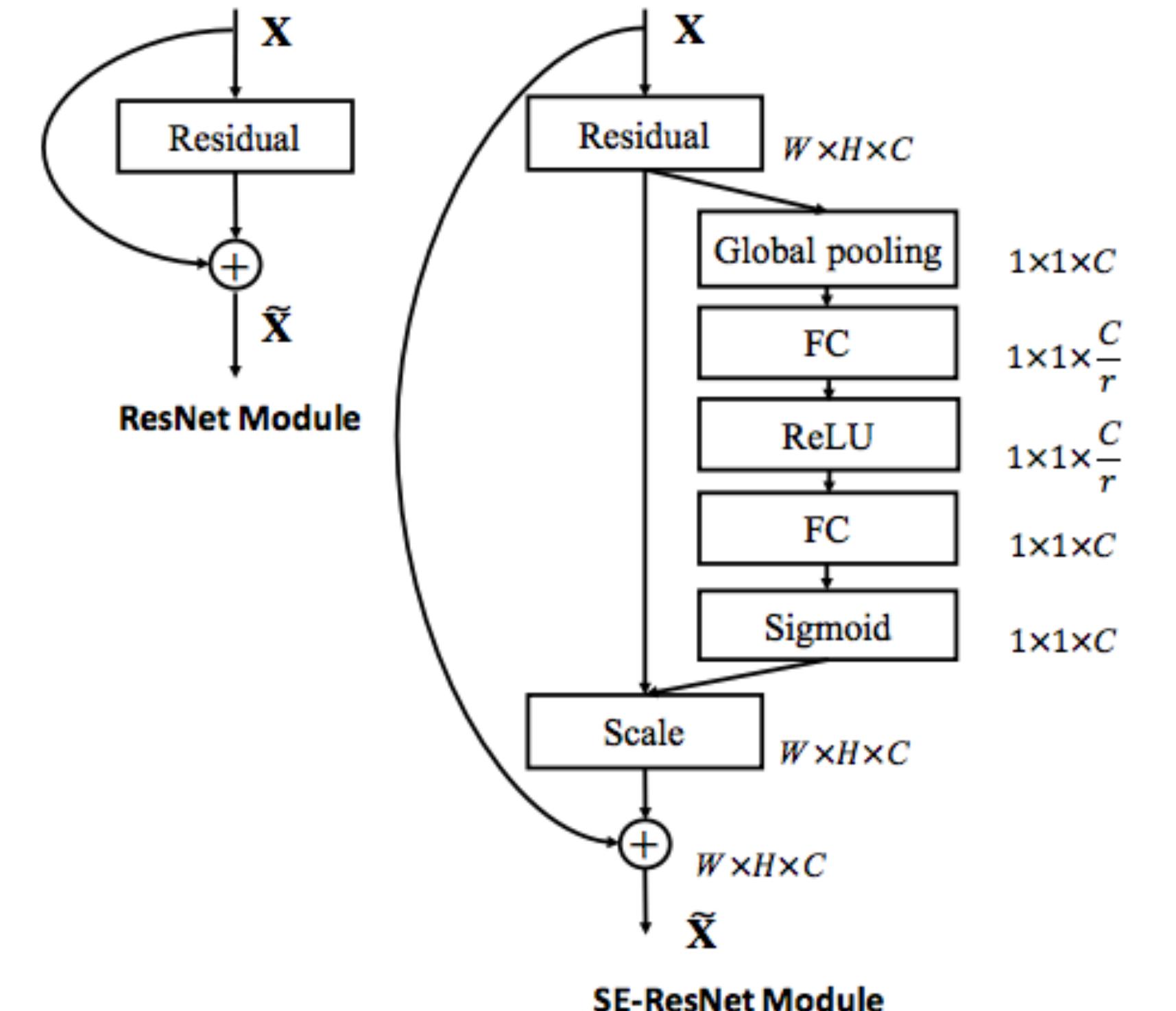
# Squeeze-and-Excitation Networks

이미지넷 2017 1등 모델 (2.25% top5 error rate)

Convolutional Layer 연산 후, 이를 압축(squeeze)한 뒤 필요한 채널만 활성화(excitation)하는 새로운 레이어를 만든다.



Convolutional Layer의 결과를 그대로 사용하지 않고  
각 채널마다의 배점을 따로 주는 Squeeze-and-Excitation Block을 추가한다



Squeeze-and-Excitation Block의 세부 구조

Squeeze-and-Excitation Networks  
Hu et al, 2017. <https://goo.gl/3BJW64>

# Summary

딥러닝의 근간이 되는 인공신경망 연구는 아주 오래 전 (1950년대)부터 진행되어왔다.

중간중간 위기(XOR Problem, SVM, etc)가 있었지만, 많은 연구자들의 노력으로 이를 극복하였다.

LeNet5를 바탕으로 CNN을 활용한 Image Classification 연구가 시작되었으며, 2012년에 Alexnet이 ILSVRC 2012를 우승함으로서 딥러닝의 부흥기가 찾아왔다.

이후 ZFNet, GoogLeNet, VGGNet과 같은 다양한 딥러닝 모델이 생겨나면서, Image Classification은 더욱 발전해나갔고, 결국 ResNet이 인간의 한계라고 알려진 5%의 벽을 돌파하였다. 하지만 연구자들은 아직도 꾸준히 Image Classification에 대한 연구를 진행하고 있다.

다음 시간부터는 LeNet5를 바탕으로, 아직 배우지 못한 부분을 추가하여 더욱 발전된 모델을 만들 것이다.

- **Activation:** tanh, ReLU, Leacky ReLU, etc
- **Weight Initialization:** Xavior Initialization, He Initialization, Batch Normalization
- **Optimizer:** Momentum, Nesterov momentum, AdaGrad, RMSProp, Adam
- **Data Preprocessing & Augmentation:** 좌우반전, mean subtraction
- **Others:** Dropout, Ensemble

**Q & A**