Artificial Intelligence

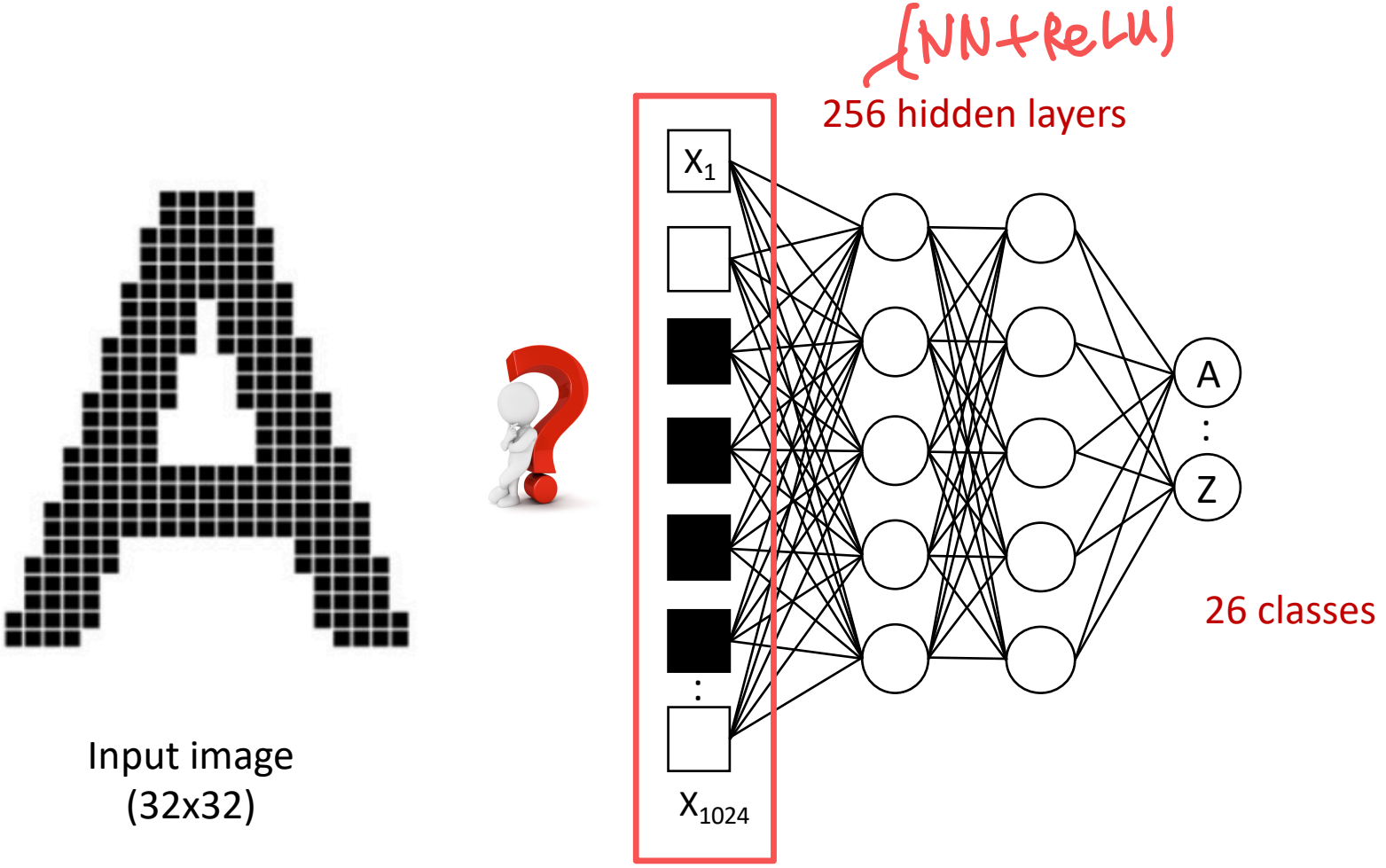
Yukyung Choi

Agenda

- Limitation of DNN
- Convolutional Neural Networks (CNN)
  - Kernel (filter, weight)
  - Receptive field
  - Stride
  - Padding
  - Pooling
    - Max / Average Pooling

Limitation of DNN

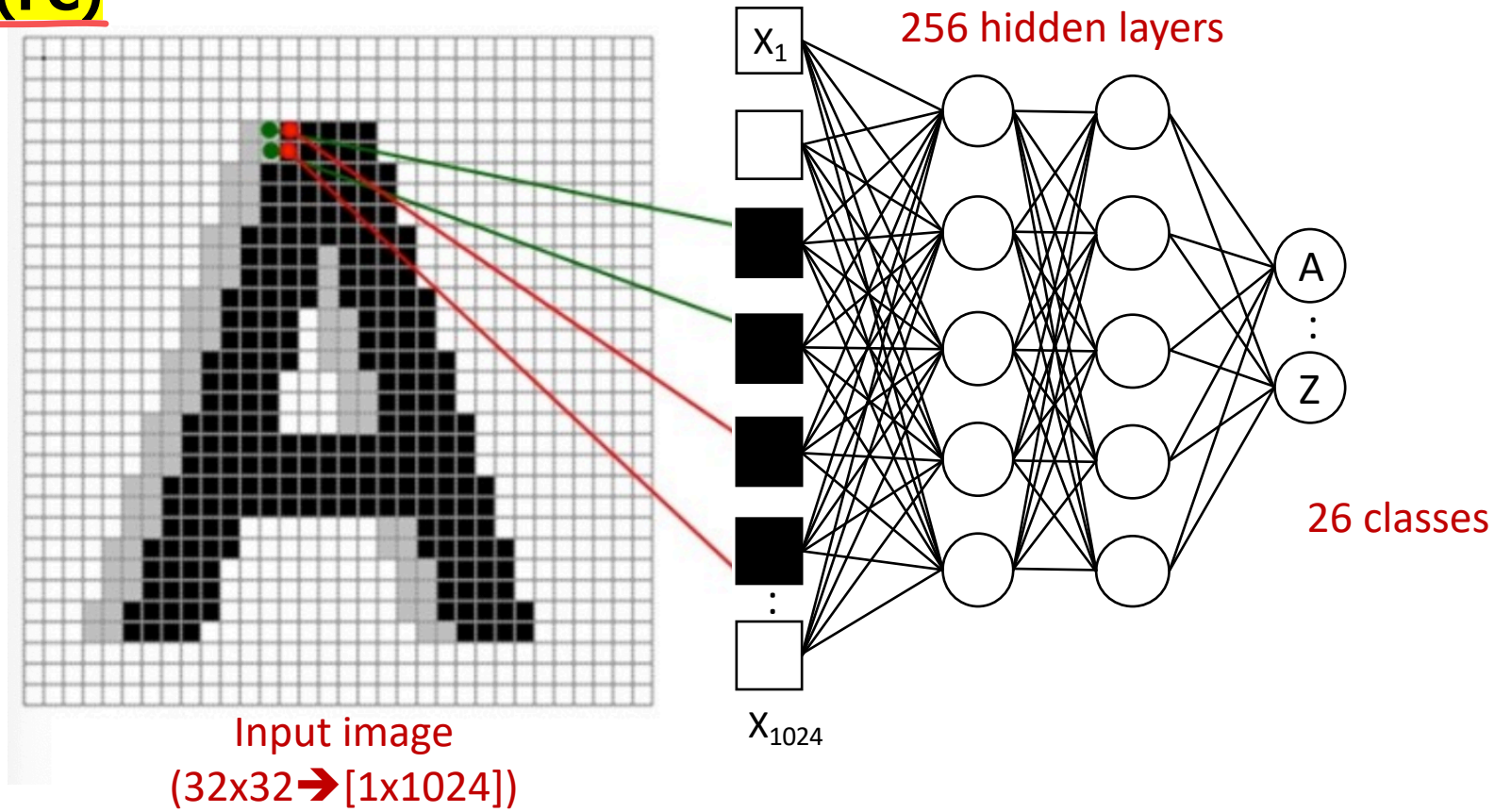
- **영상데이터(2D 데이터)에 DNN을 적용하여 분류 문제를 푼다면?**



Limitation of DNN

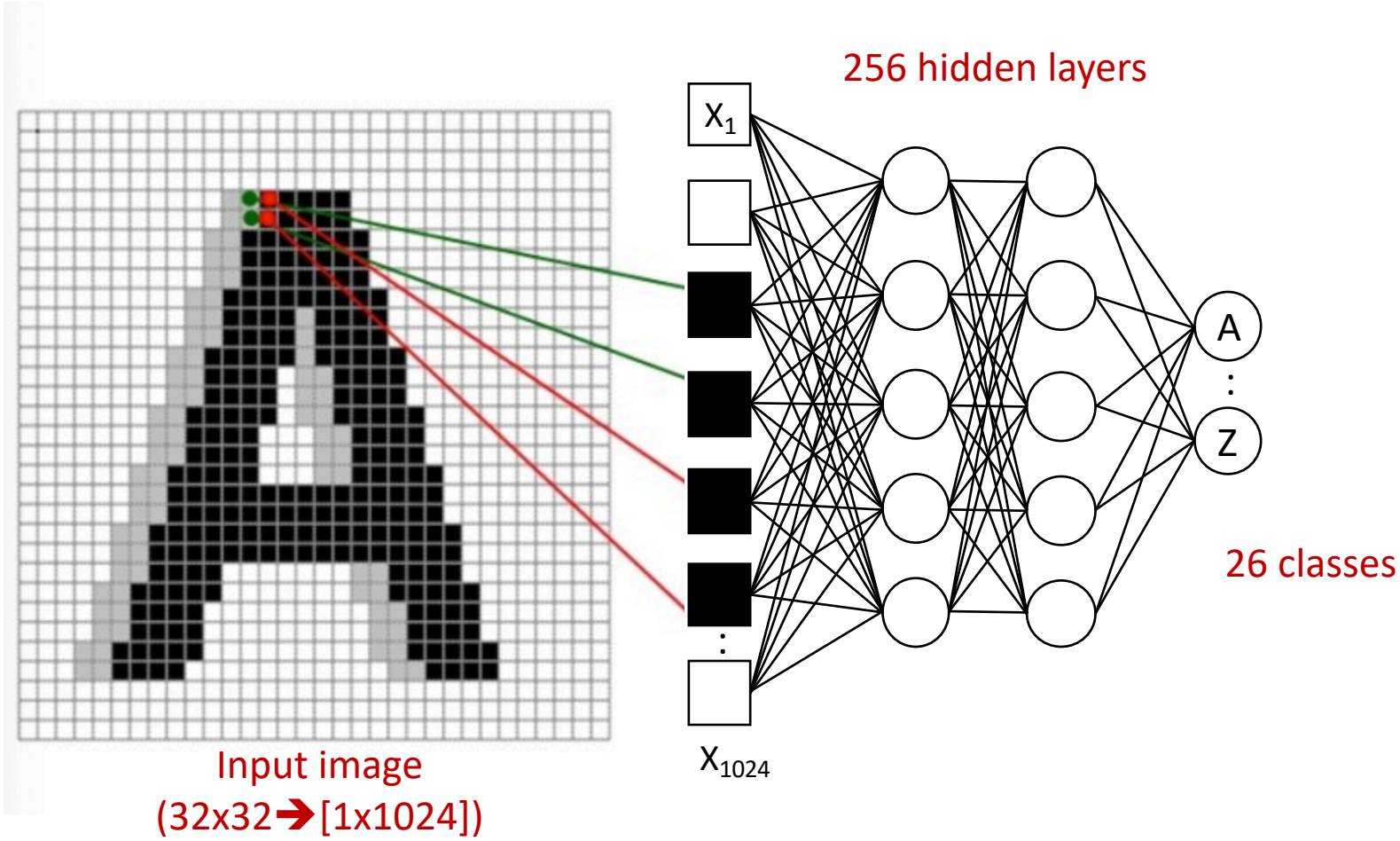
위치정보 중요

- **2D 데이터를 1D로 평탄화 하여 DNN 적용 가능**
- **위치에 상관없이 동일한 수준의 중요도를 갖음**
- **DNN = Multi-layer Perceptron = Feedforward NN = Fully Connected Layer (FC)**



Limitation of DNN

- 전체 글자가 2픽셀 이동만 하더라도 새로운 학습 데이터로 처리해야 함 (translation invariance 특성이 보장되지 않음)



Limitation of DNN

- 글자 크기(scale)가 달라지거나, 글자가 회전(rotation)하거나, 글자에 변형(distortion)이 생겨도 좋은 결과를 기대하기 어려움
  - Scale / Rotation / Distortion Invariance 특성이 보장되지 않음

Summary of Limitation (DNN)

- 학습시간 (training time) : 크기, 회전, 이동 등 모든 데이터 학습해야
- DNN 모델의 크기 (network size) : 입력영상수, layer 깊어짐
- 변수의 개수 (number of parameters)

→ 모델크기↑

→ 파라미터수↑

Convolutional Neural Network (CNN)

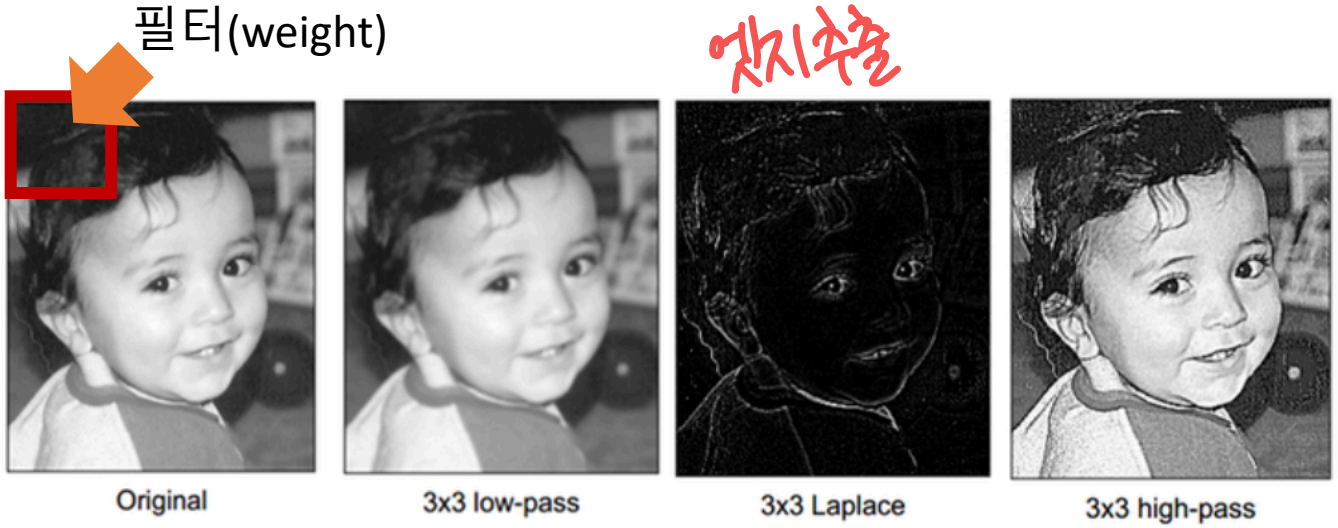
- CNN이란 무엇인가? (wikipedia)
  - **합성곱 신경망**(Convolutional neural network, CNN)은 시각적 이미지를 분석하는 데 사용되는 깊은 인공신경망의 한 종류이다.
  - CNN은 **변환 불변성 특성**에 기초하며, 이미지 및 비디오 인식, 추천 시스템, 이미지 분류, 의료 이미지 분석에 응용된다.
  - CNN은 다른 이미지 분류 알고리즘에 비해 상대적으로 **전처리를 거의 사용하지 않는다**. 이는 네트워크가 기존 알고리즘에서 수작업으로 제작된 여러 필터 역할을 스스로 학습한다는 것을 의미한다.

(DNN)

크기, 이동, 회전 등의 변화 X

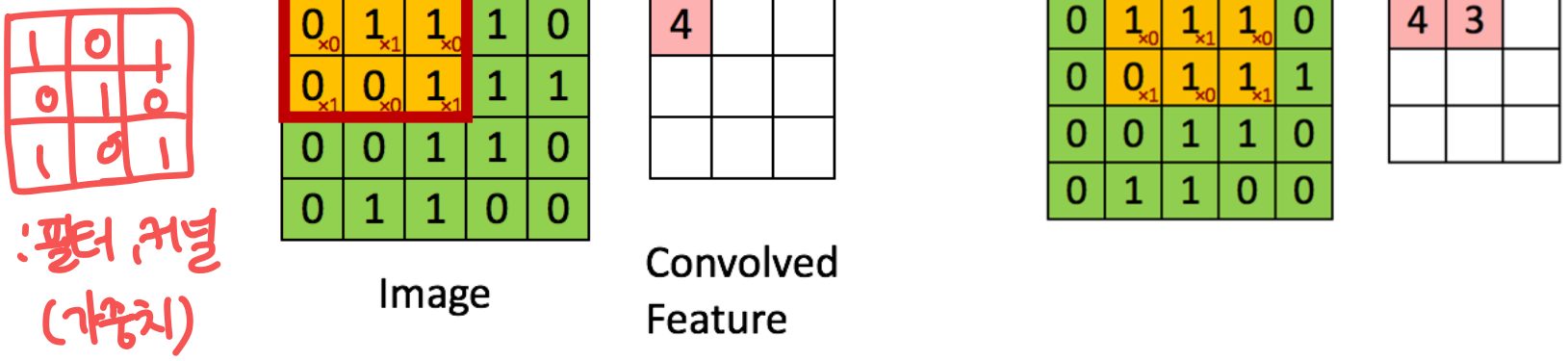
Convolution

- 영상처리 혹은 컴퓨터비전 에서의 컨볼루션이란?
  - Convolution은 주로 filter연산에서 사용되며, 영상으로 부터 특정 feature를 추출하고 싶을 때 사용한다.



Convolution

- 영상처리 혹은 컴퓨터비전 에서의 컨볼루션이란?
  - Convolution은 주로 filter연산에서 사용되며, 영상으로 부터 특정 feature를 추출하고 싶을 때 사용한다.

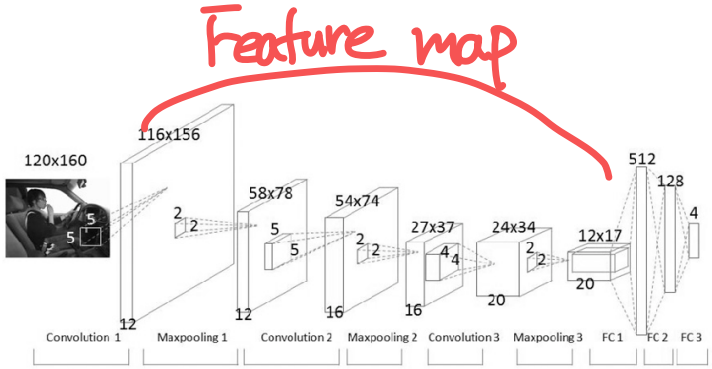


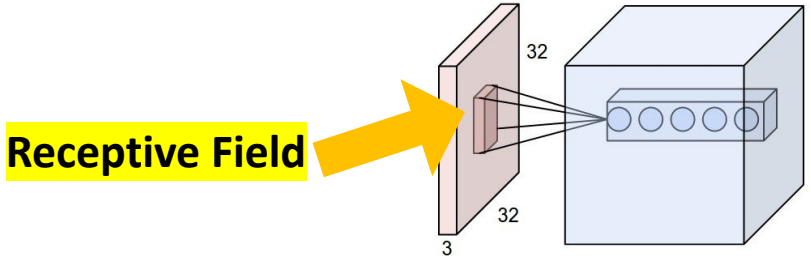
Green: 영상 이미지
Yellow: 컨볼루션이 일어나는 영역
Red: 컨볼루션 커널

Convolutional Neural Network (CNN)

- Convolution + Neural Network
  - CNN➔ Convolution특성을 살린 신경망 연산
  - 2번 이상의 CNN, 입력영상뿐만 아니라 중간 Feature map에도 Convolution 적용

- Receptive Field(수용영역)란 무엇인가?
  - **Receptive field** 는 출력 레이어의 뉴런 하나에 영향을 미치는 입력 뉴런들의 공간 크기이다.
  - 외부 자극이 전체 영향을 끼치는게 아니라 특정 영역에만 영향을 미친다.
  - 영상에서 특정 위치에 있는 픽셀들은 그 주변에 있는 일부 픽셀들과만 correlation 높을 뿐이며, 거리가 멀어질 수록 그 영향은 감소하게 된다.



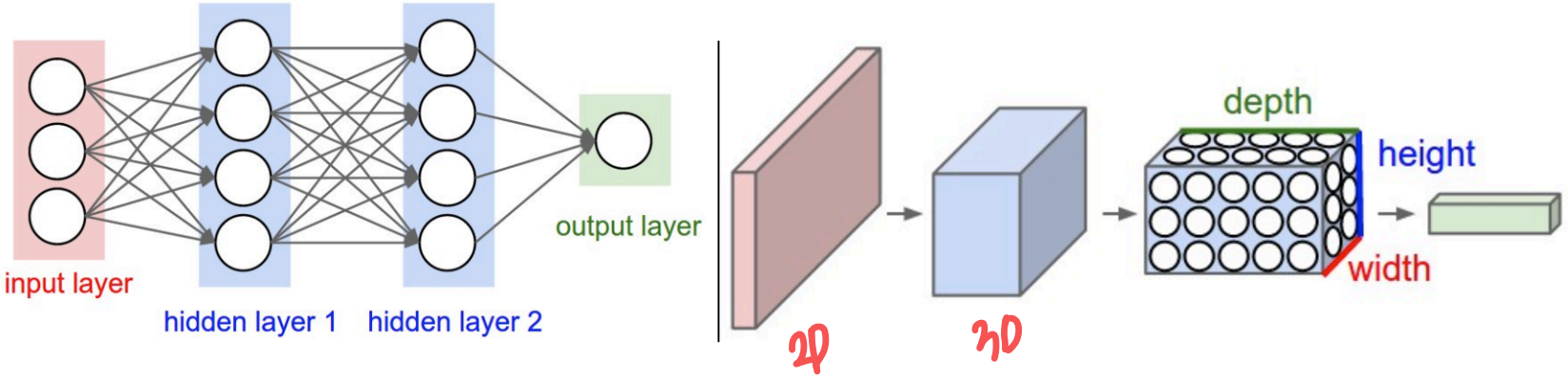


Advantage of CNN

- Conv layer는 형상을 유지한다.
- 즉, 입/출력 모두 3차원 데이터로 처리하기 때문에 공간적 정보를 유지할 수 있다.

평탄화 X

왼쪽은 일반적인 3-layer Neural Network(전결합), 오른쪽은 CNN을 3차원으로 표현한 그림이다.

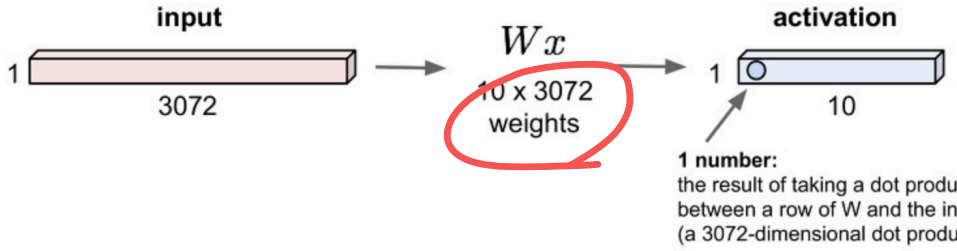


DNN **CNN**

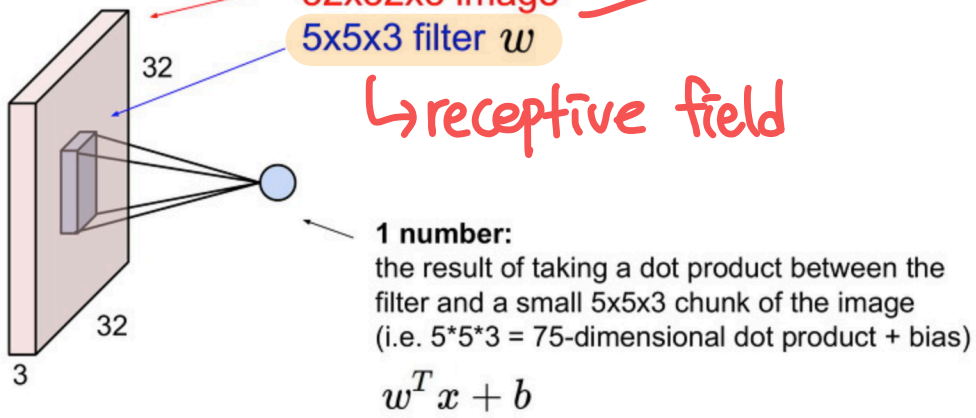
DNN vs CNN

Fully Connected Layer

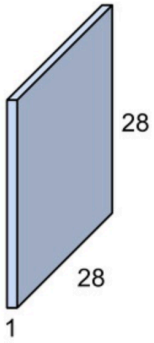
32x32x3 image -> stretch to 3072 x 1



Convolution Layer



↳receptive field

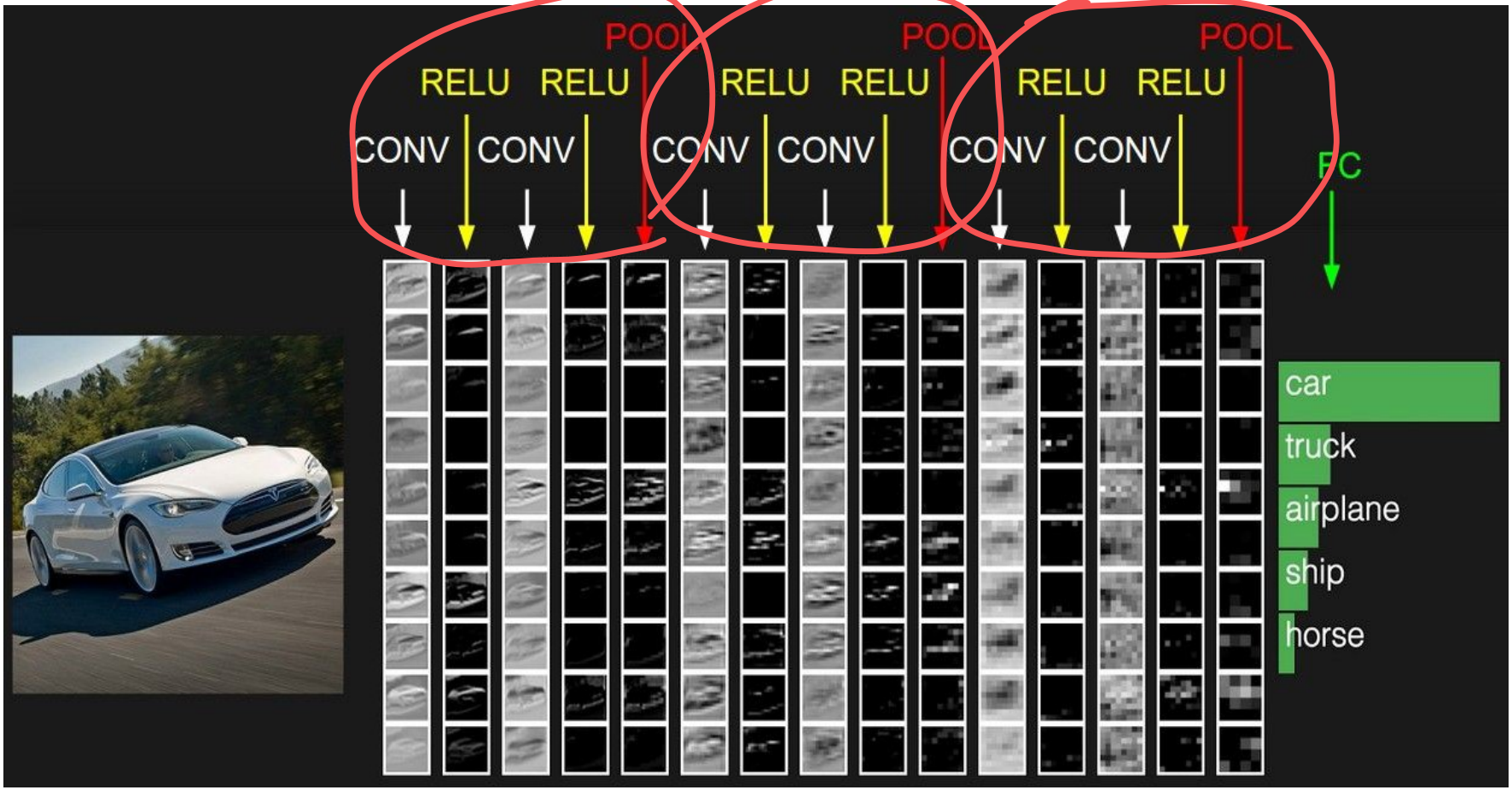


- 적은수의 가중치
- 부분의 합으로 판단하는 브레인 모사

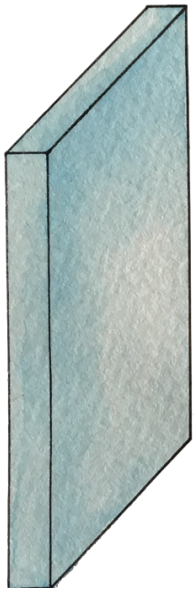
↓

kernel

Example: CNN based Classification



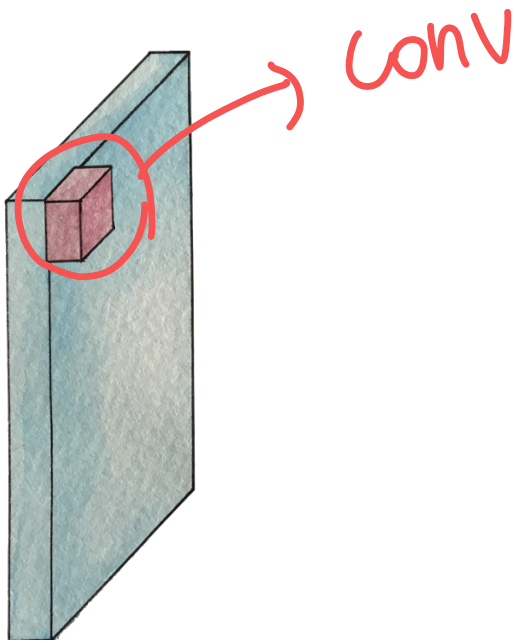
Start with an image (width x height x depth)



32x32x3 image

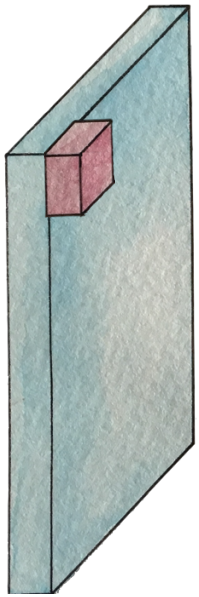
↳ color 채널

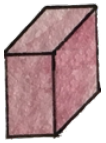
Let’s focus on a small area only



32x32x3 image

Let’s focus on a small area only (5x5x3)

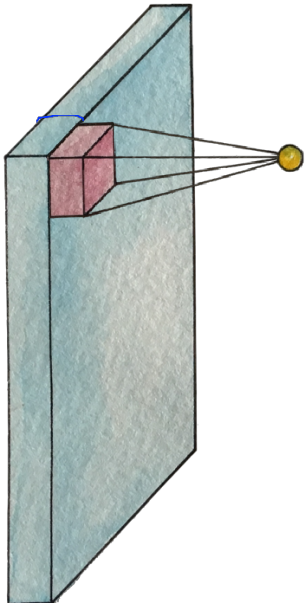




32x32x3 image

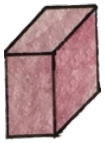
5x5x3 filter

Get one number using the filter (w)



one number!

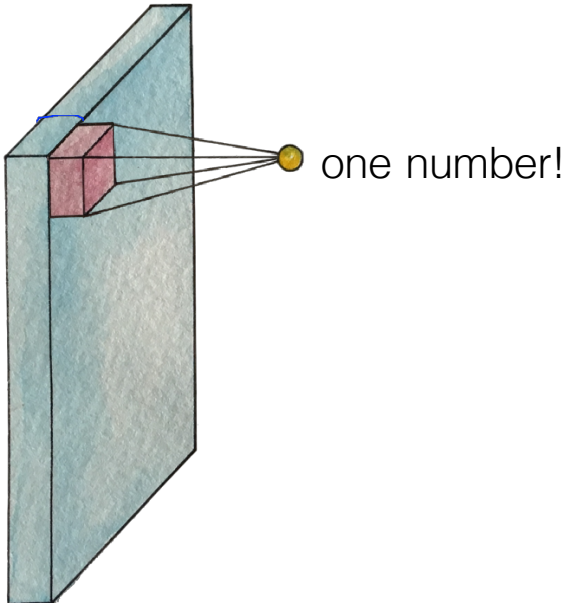
Wx+b [1]



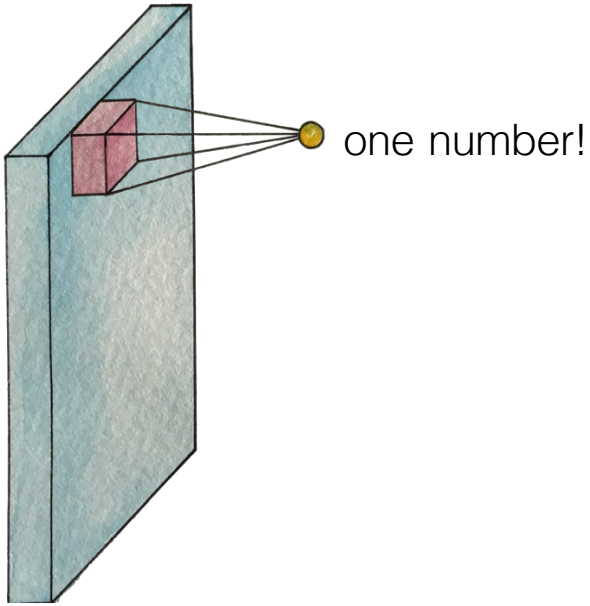
32x32x3 image

5x5x3 filter

Get one number using the filter (w)

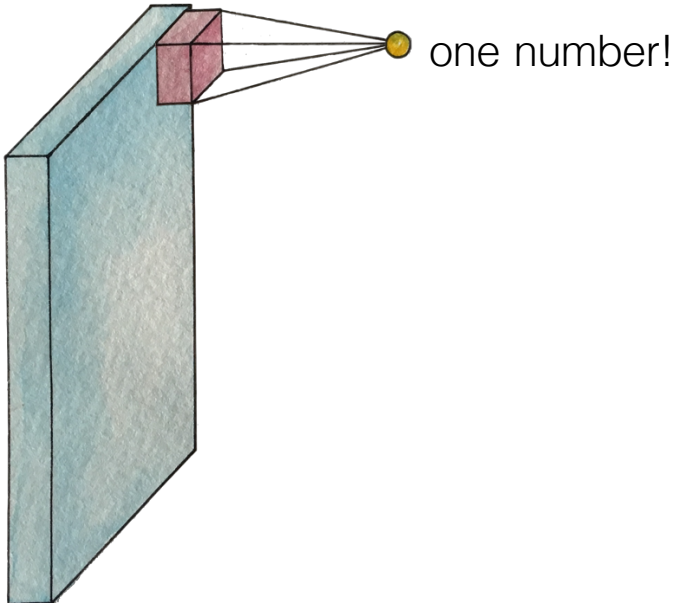


Get one number using the filter (w)



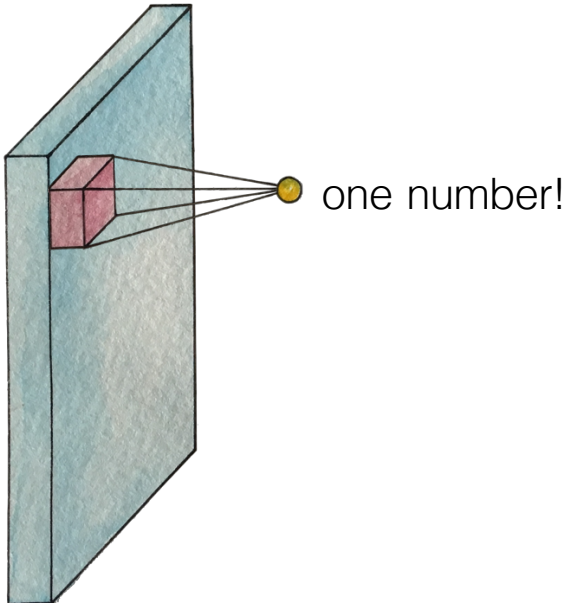
32x32x3 image

Get one number using the filter (w)



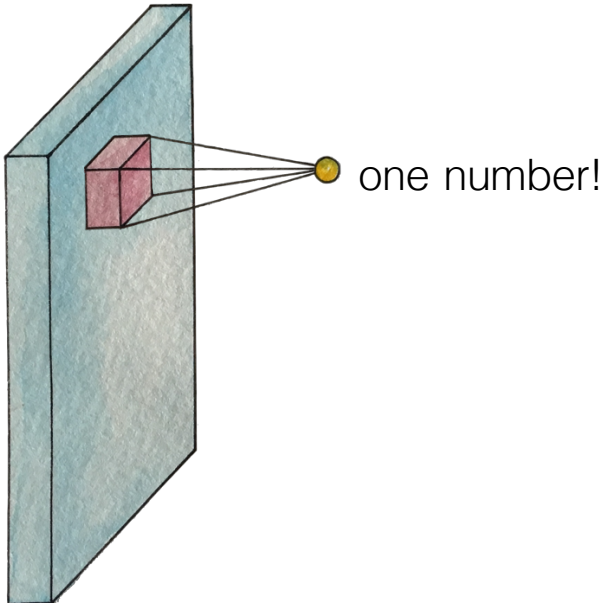
32x32x3 image

Get one number using the filter (w)



32x32x3 image

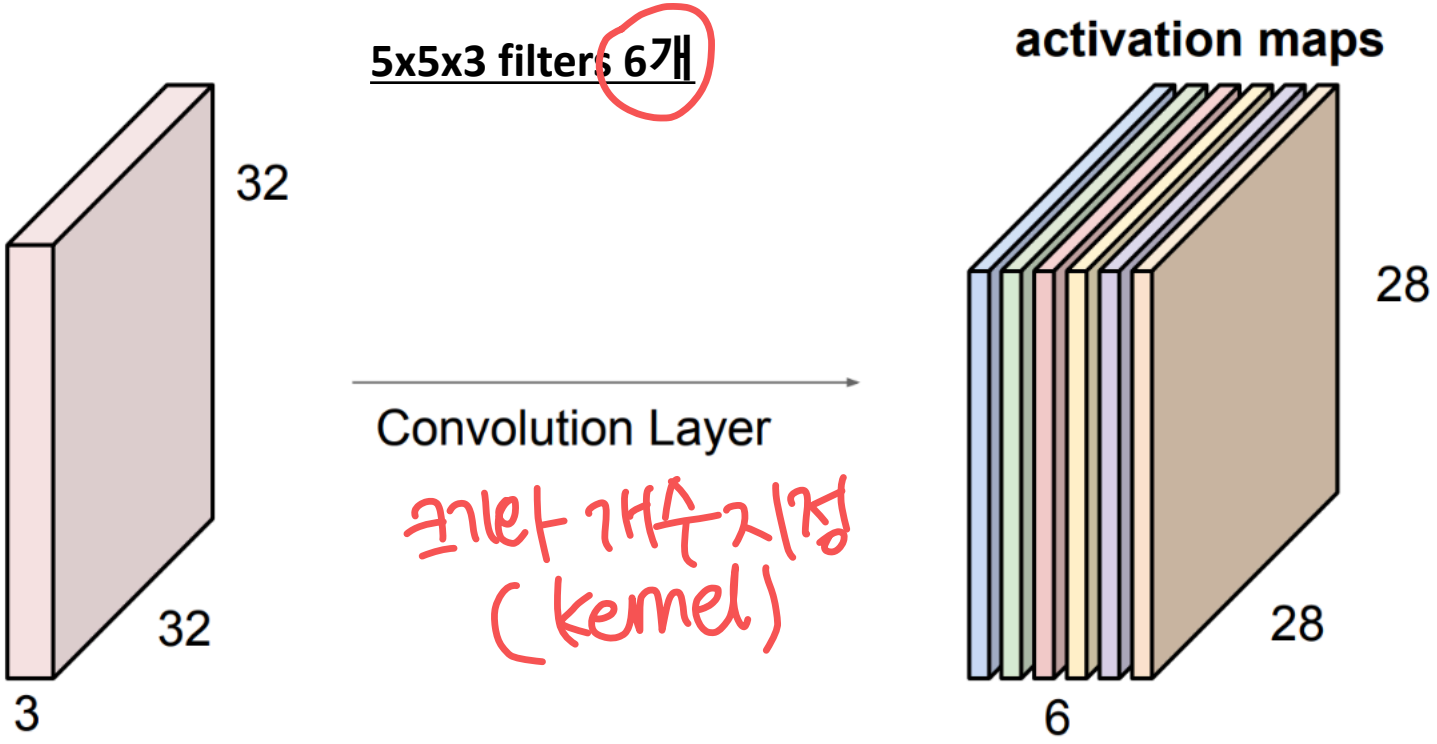
Get one number using the filter (w)



32x32x3 image

How many numbers can we get?

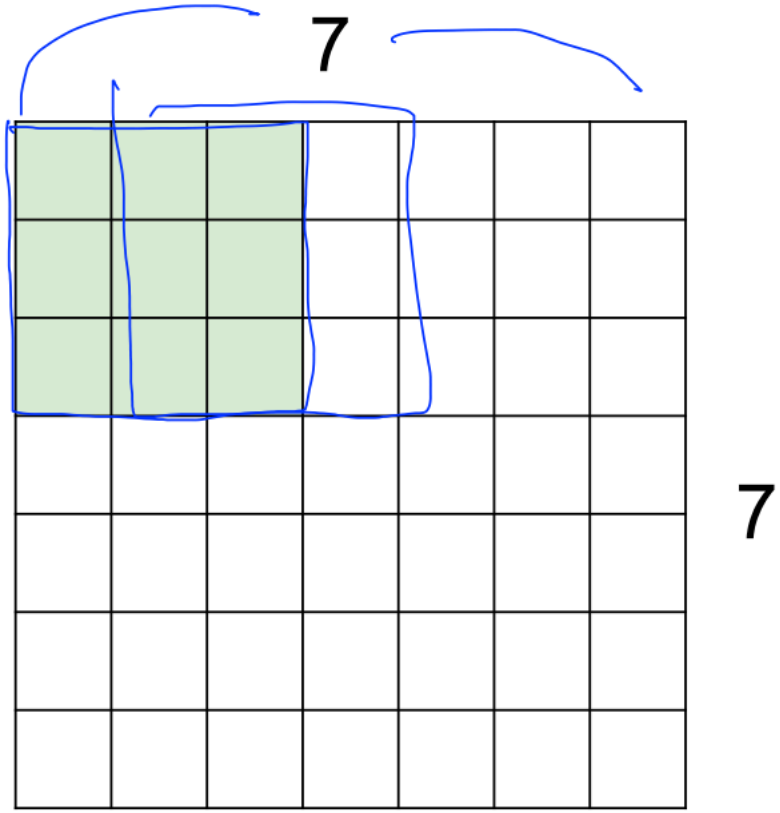
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

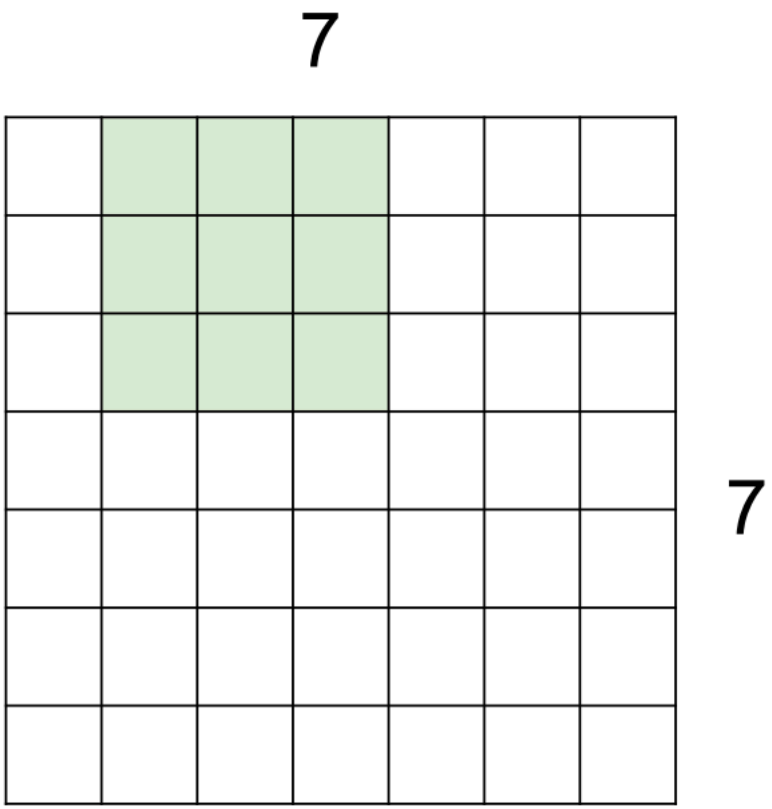
A closer look at spatial dimensions: stride==1

움직이는 칸수



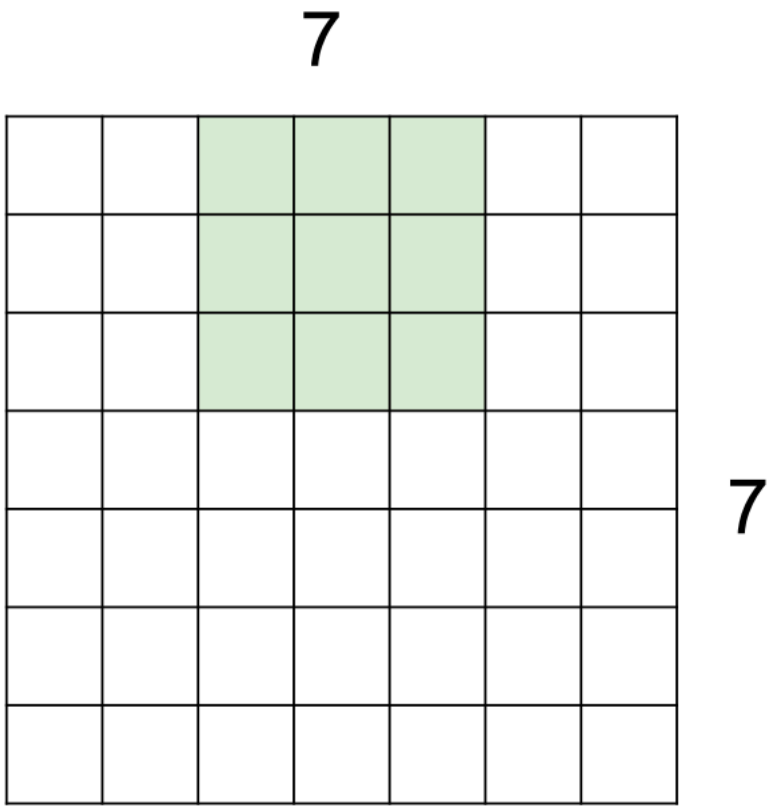
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



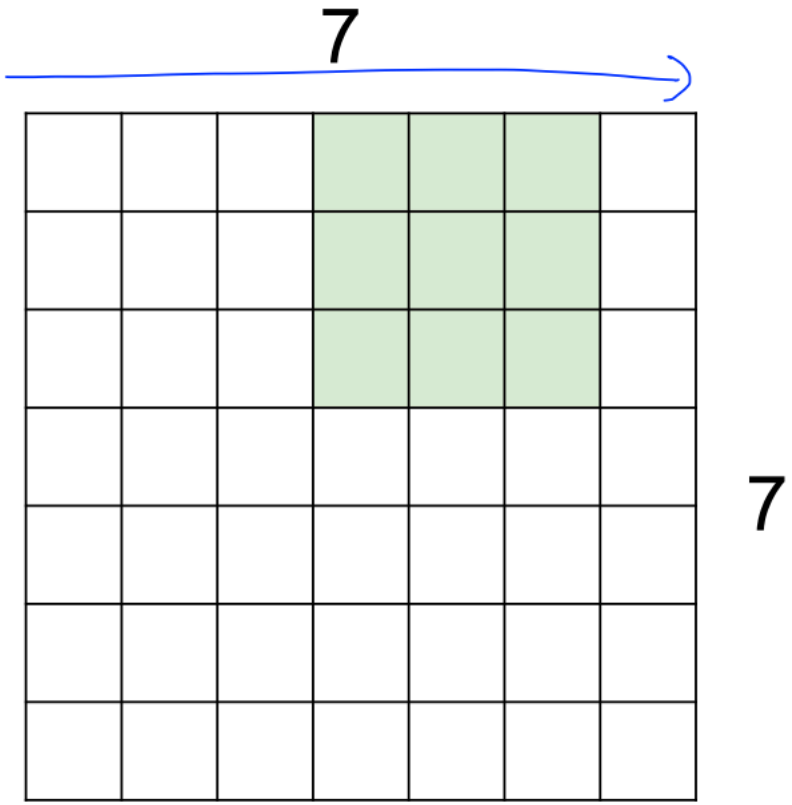
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

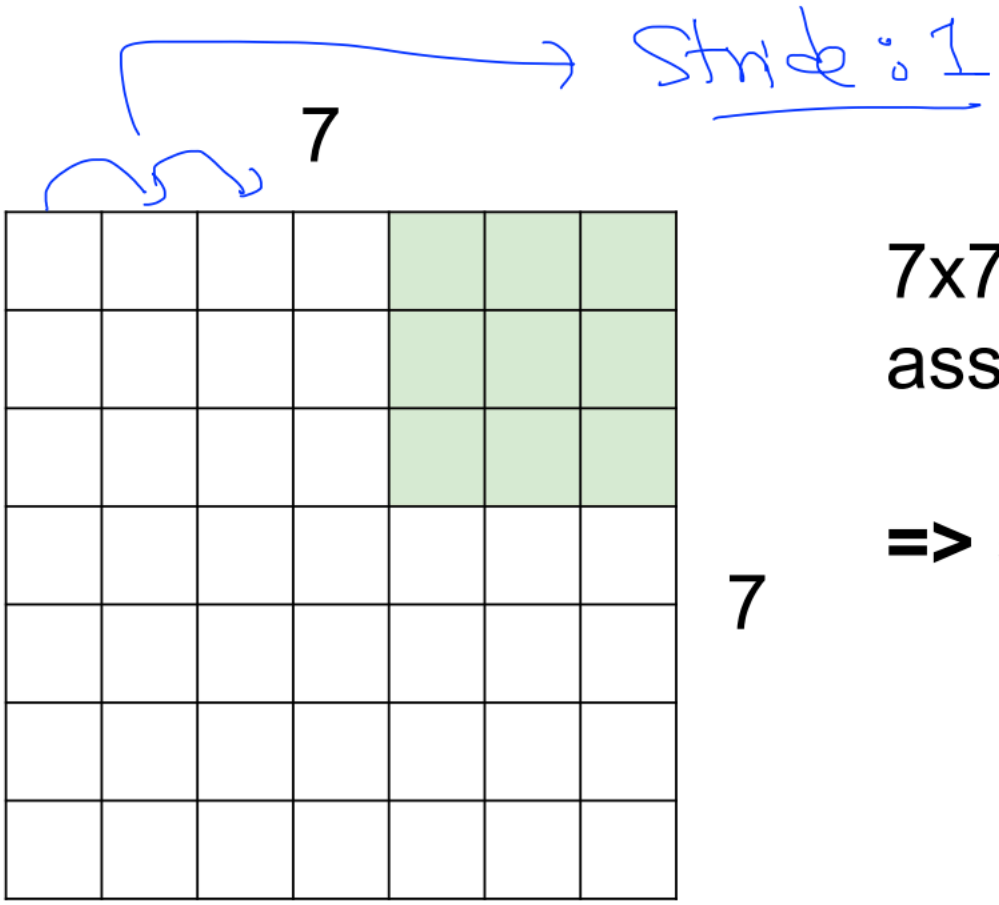
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

5x5

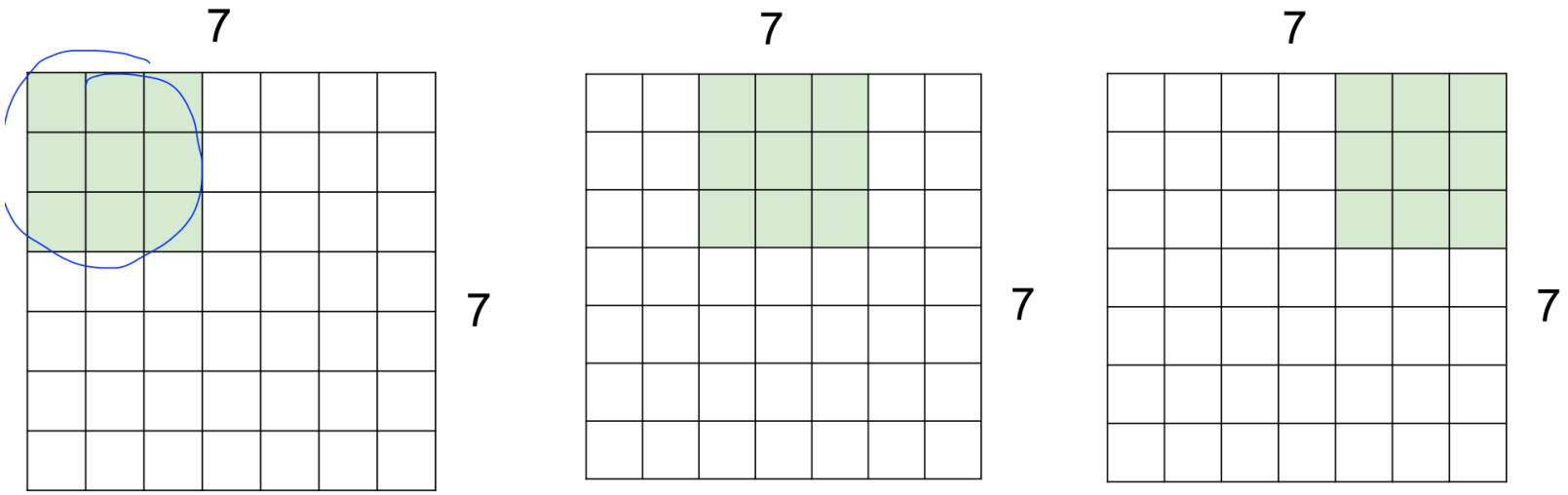
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

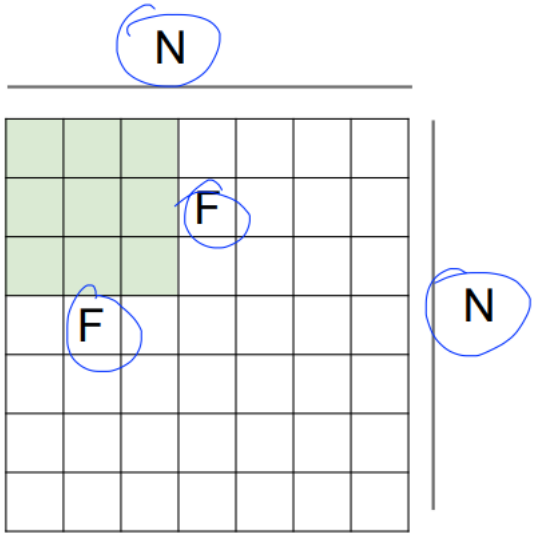
=> 5x5 output

A closer look at spatial dimensions: stride==2



7x7 input (spatially) assume 3x3 filter applied with stride 2 => **3x3 output!**

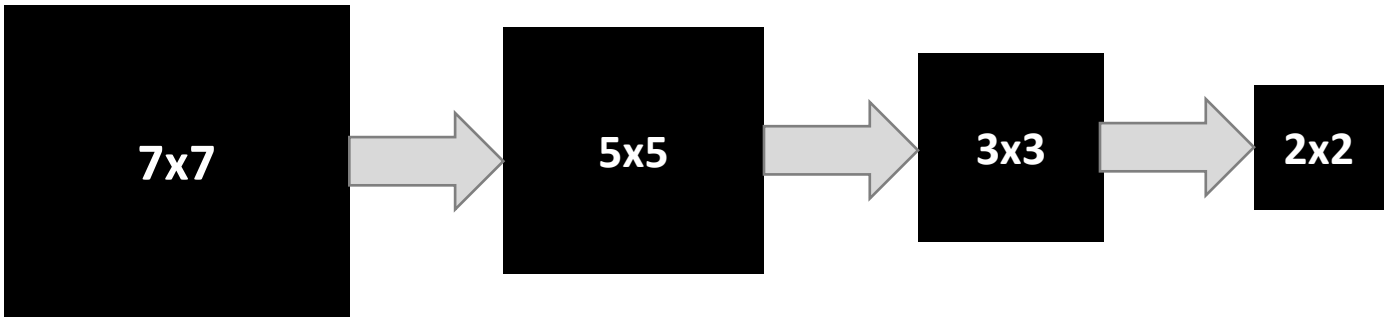
A closer look at spatial dimensions: stride==?



Output size:

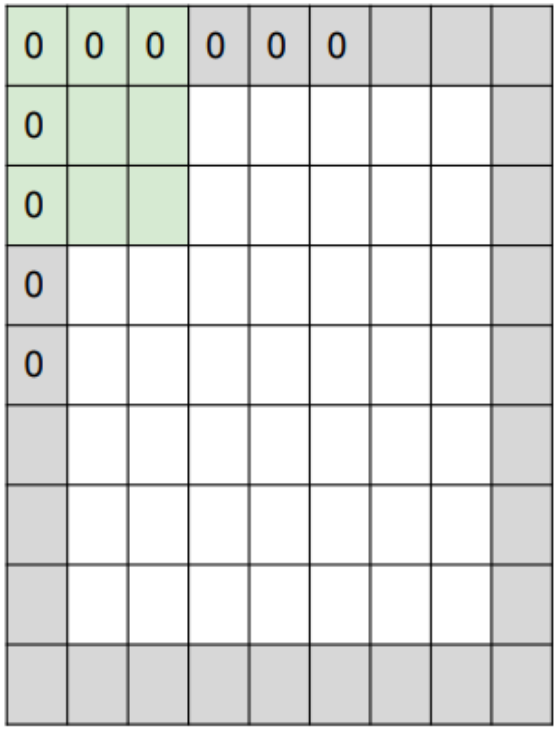
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = **5** (O)
stride 2 => (7 - 3)/2 + 1 = **3** (O)
stride 3 => (7 - 3)/3 + 1 = 2.33 (X)
Stride 4 => (7 - 3)/4 + 1 = **2** (O)



In practice: Common to zero pad the border

영상사이즈유지 → 패딩.



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

패딩사이즈 = [F/2]

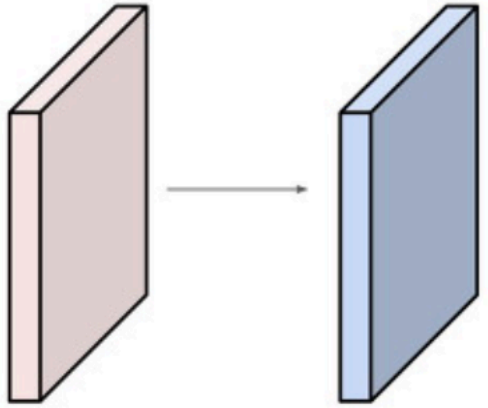
Question)

Examples time:

W H ch

Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2



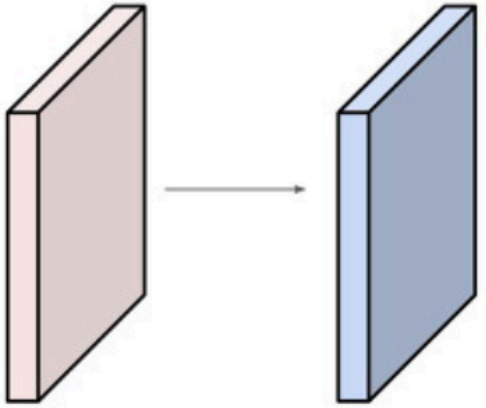
Number of parameters in this layer?

$(5 \times 5 \times 3 + 1) \times 10 = 760$개

weight bias 10개

Answer)

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

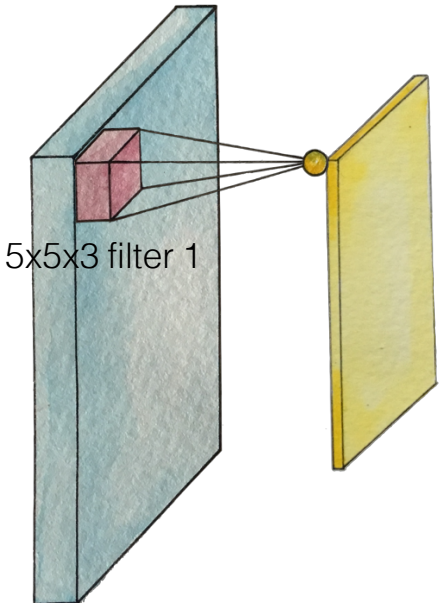
Summary

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

① Convolution Layers

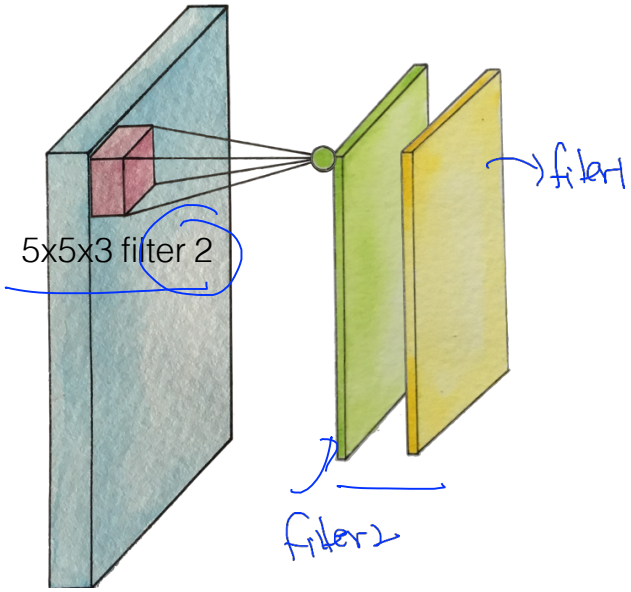
- Swiping the entire image



32x32x3 image

① Convolution Layers

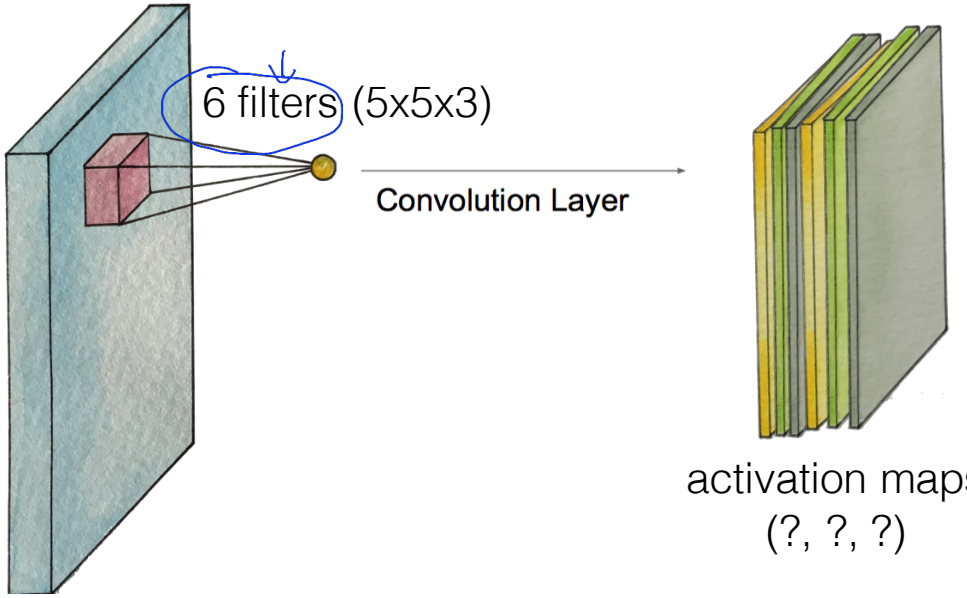
- Swiping the entire image



32x32x3 image

① Convolution Layers

- Swiping the entire image

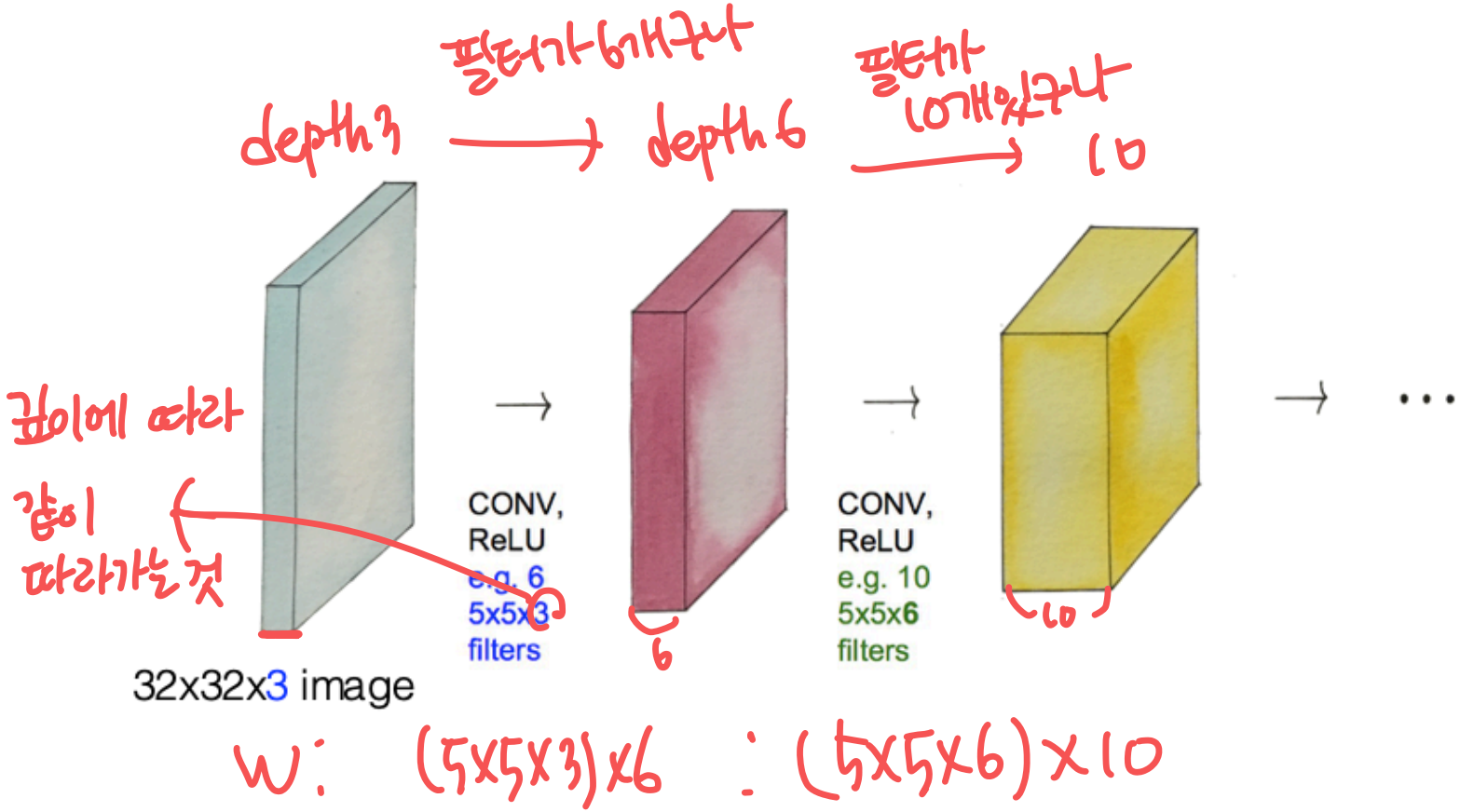


32x32x3 image

(28x28x6)

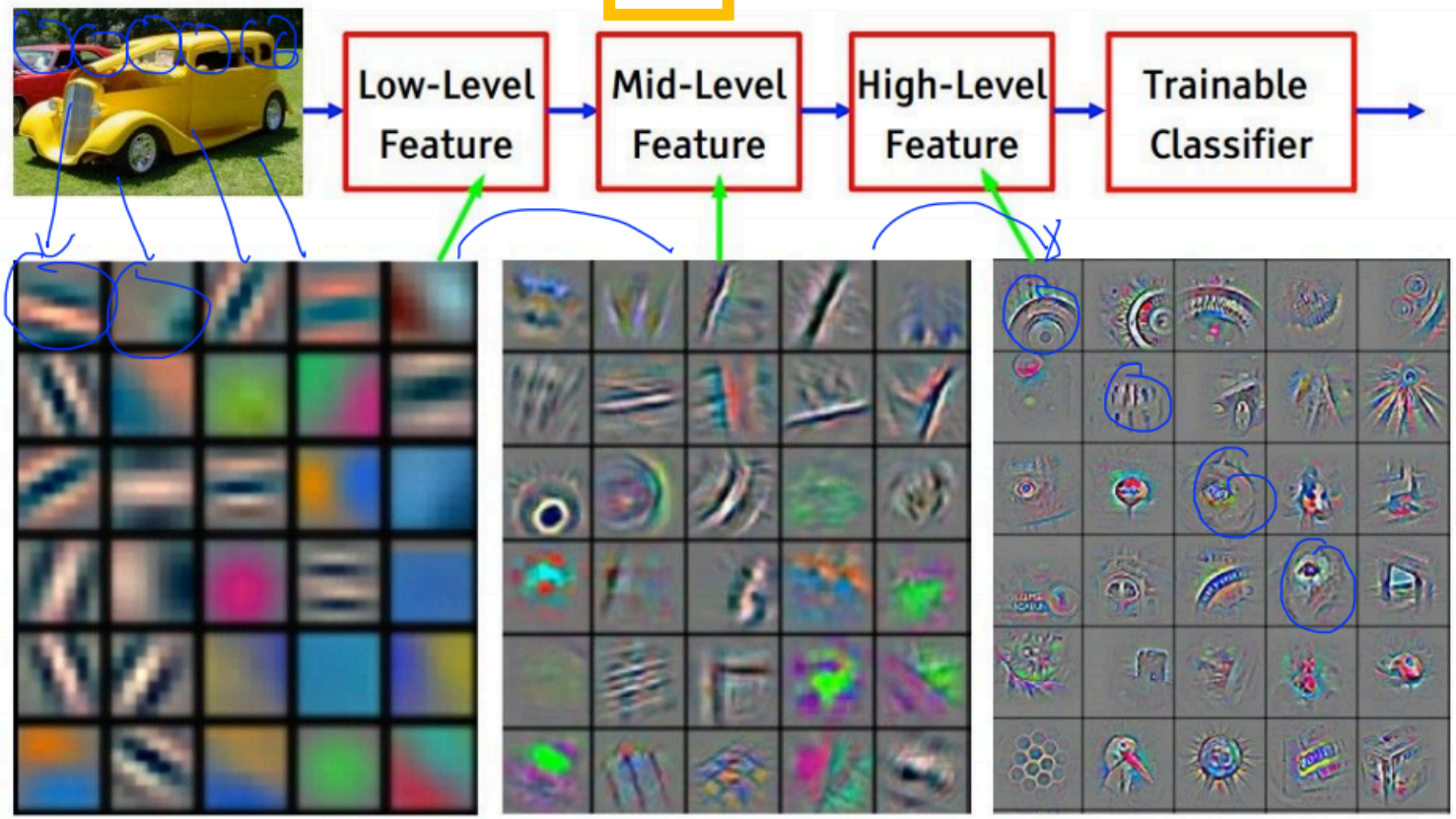
① Convolution Layers

- How many weight variables? How to set them?



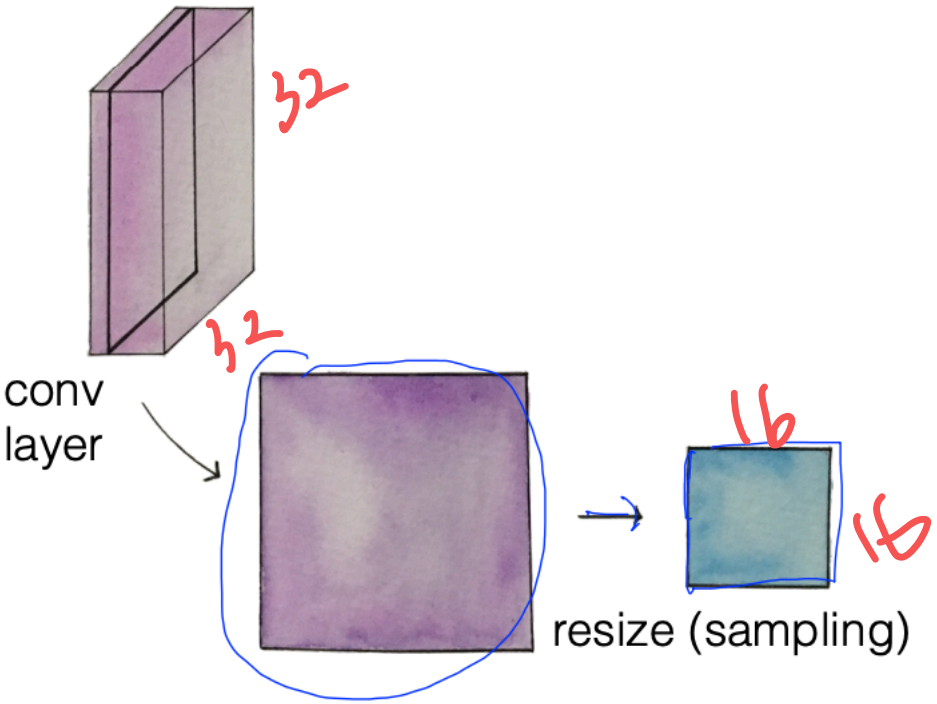
① Visualization of Activation Map (Feature Map)

[From recent Yann LeCun slides]

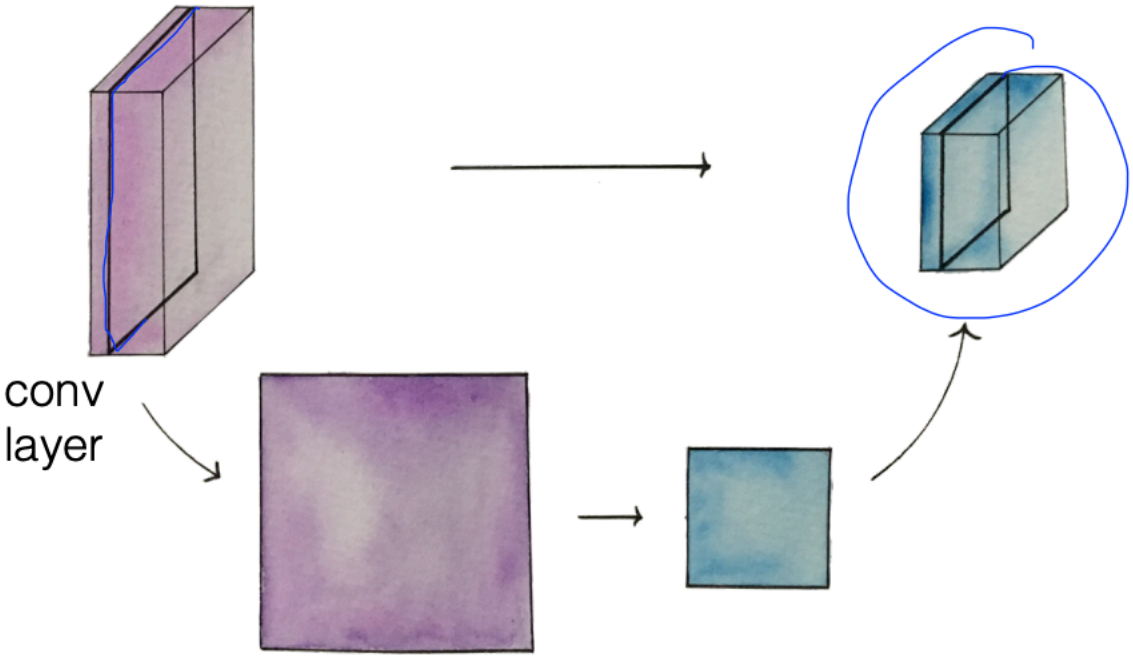


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

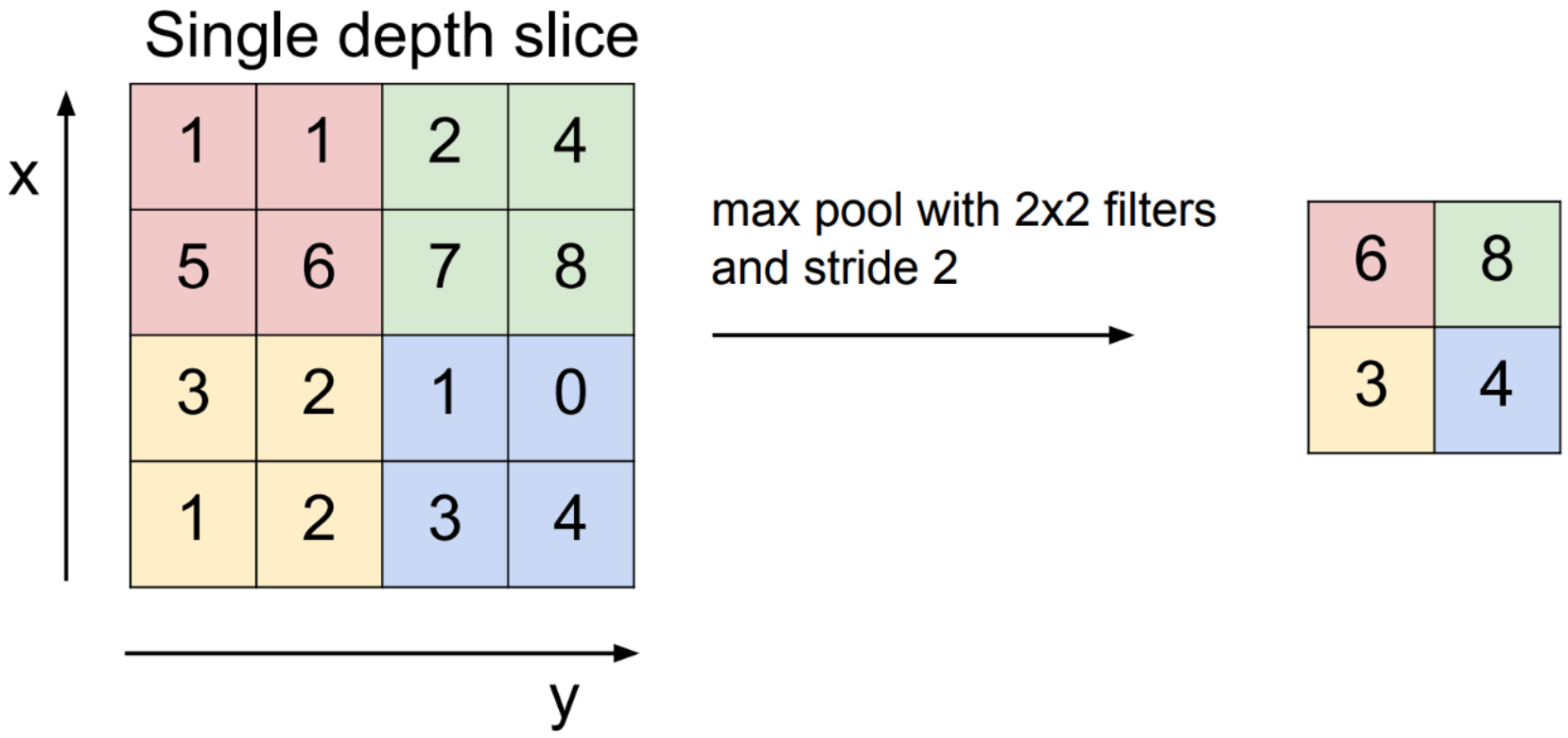
②Pooling Layer (sampling)



②Pooling Layer (sampling)



②Max Pooling



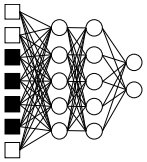
②Max Pooling

- (2x2 filter를 사용하는 경우) 전체 데이터의 75%를 버리고 25%만 선택
  - **Computational Complexity 감소**
- Filter 내에서 가장 큰 값을 선택
  - Average Pooling은 평균값을 선택.
  - Average Pooling은 Spatial Structure를 보존하되 이미지가 smooth해짐.
  - Max Pooling은 더 강한 특징만 남기는 방식
- Depth를 줄이지 않고 Spatially하게만 줄임(Height & Width)

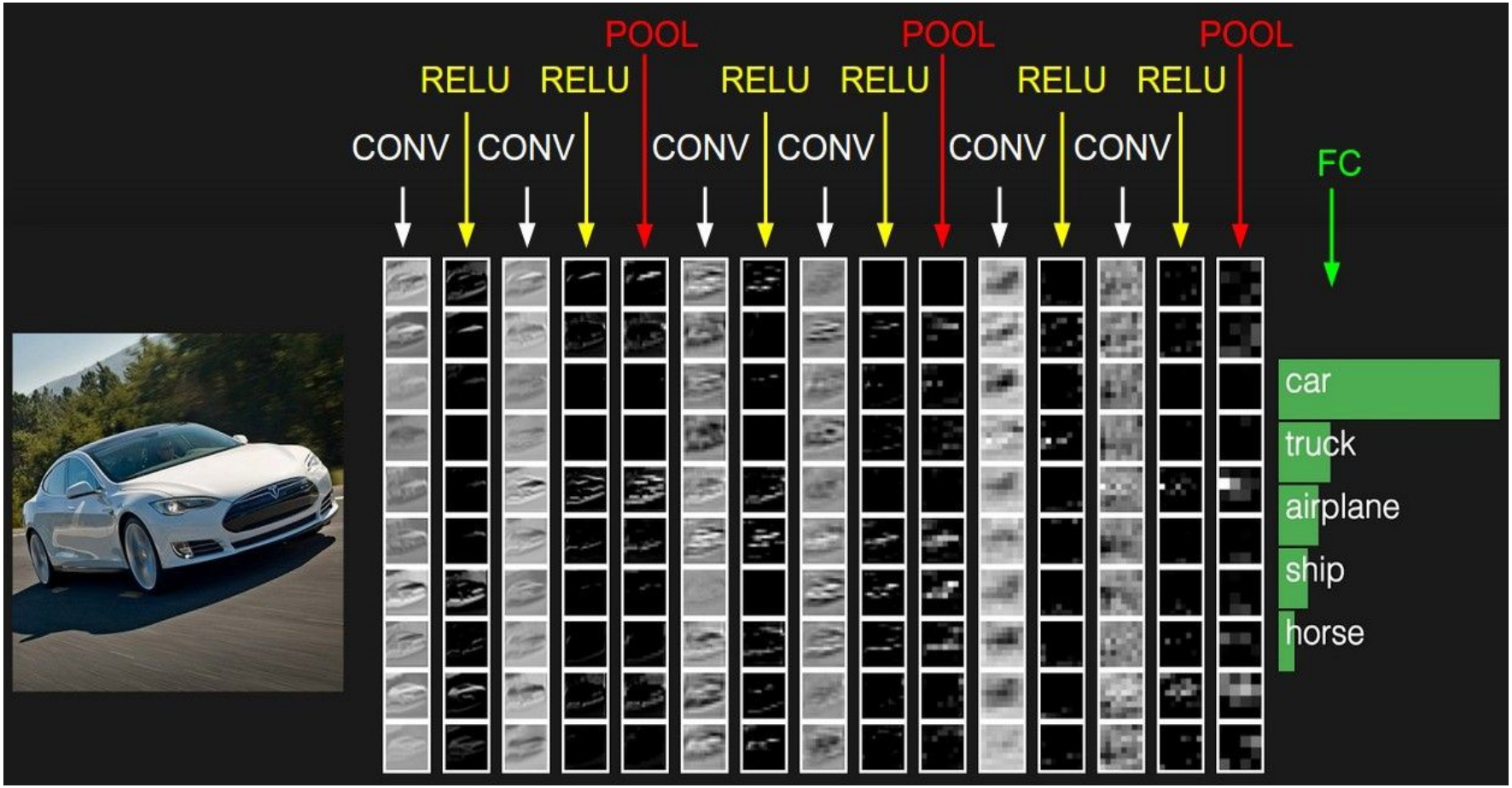
32x32x3 → 16x16x3 ⇒ depth 유지

- Q) stride와 pooling 모두 down-sampling인데 어떤 것을 사용해야 하나요?
- A) 최근 CNN 아키텍쳐는 stride를 사용하는 편이 많습니다. stride 추천합니다

③Fully Connected Layer (FC layer)



- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

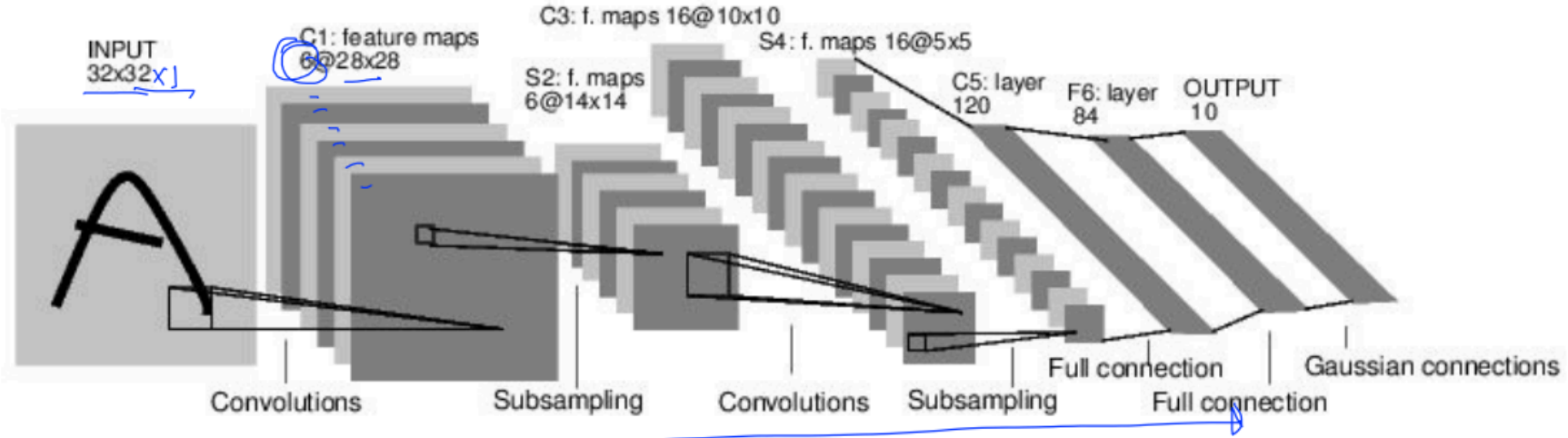


CNN Case Study

- LeNet-5
- AlexNet
- GoogLeNet
- ResNet

Case Study: LeNet-5

[LeCun et al., 1998]

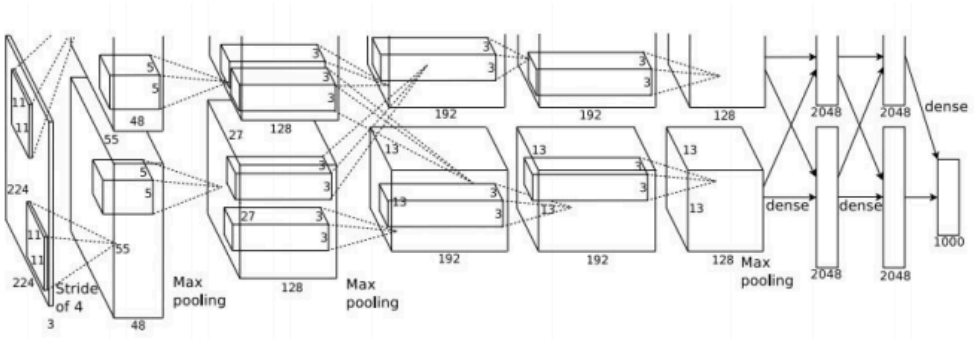


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

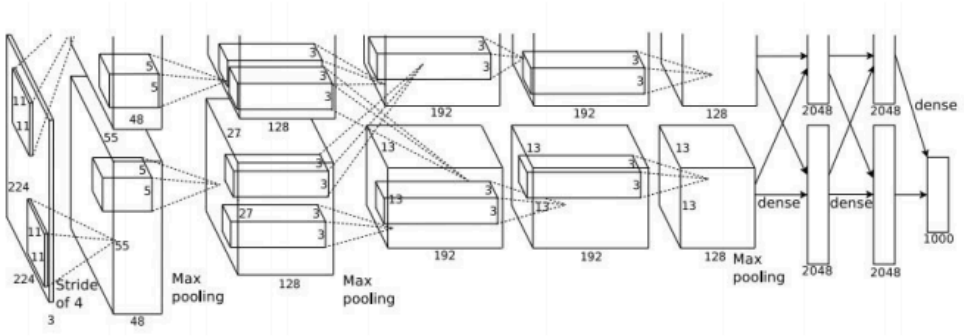
Output volume **[55x55x96]**

Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



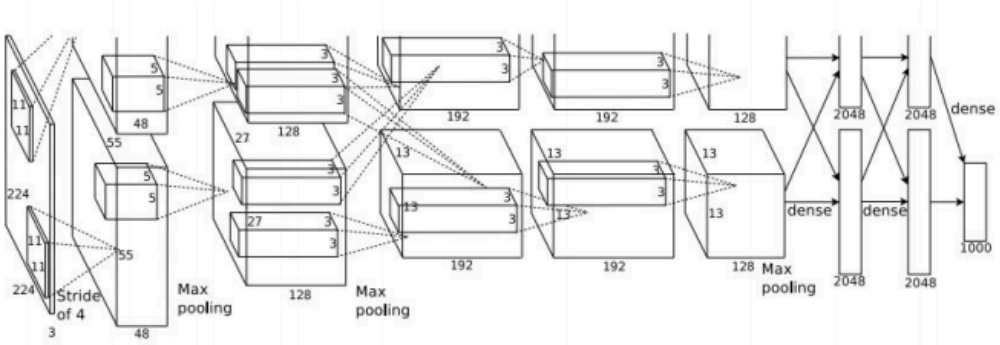
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Case Study: AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

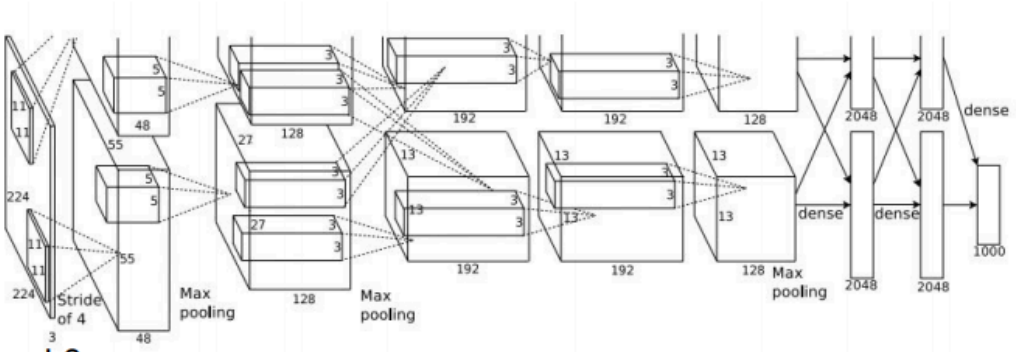


Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

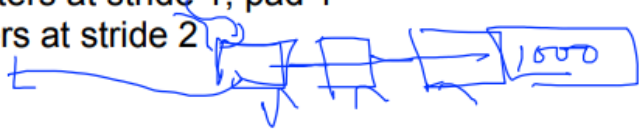
Case Study: AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



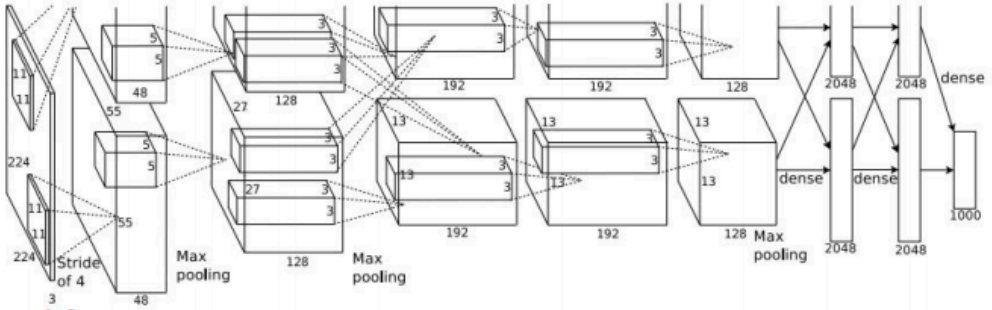
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] ~~NORM1: Normalization layer~~
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] ~~NORM2: Normalization layer~~
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)



Case Study: AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

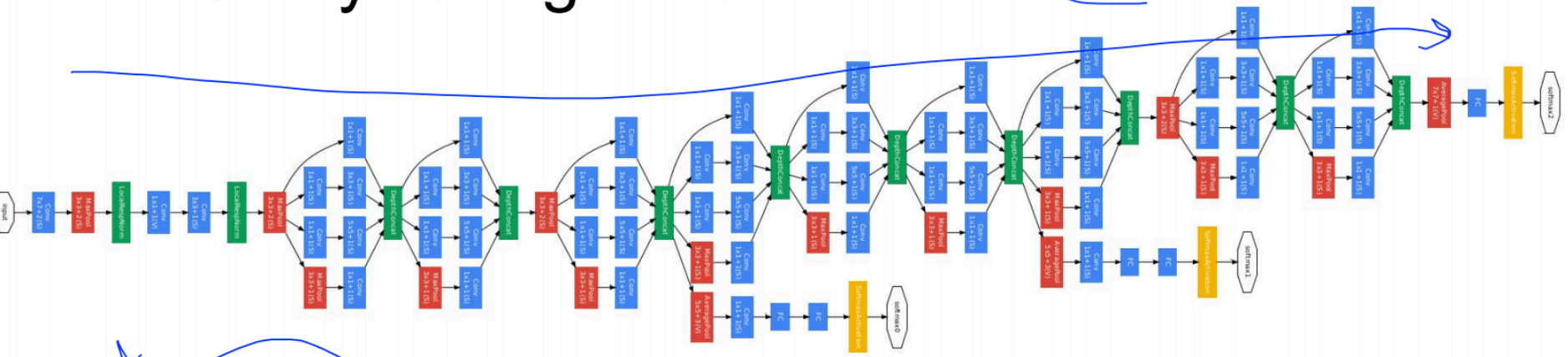


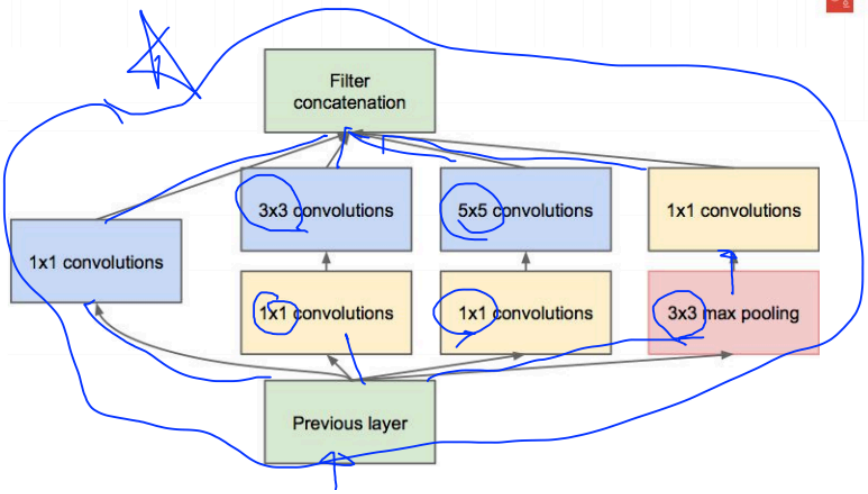
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study: GoogLeNet

[Szegedy et al., 2014]



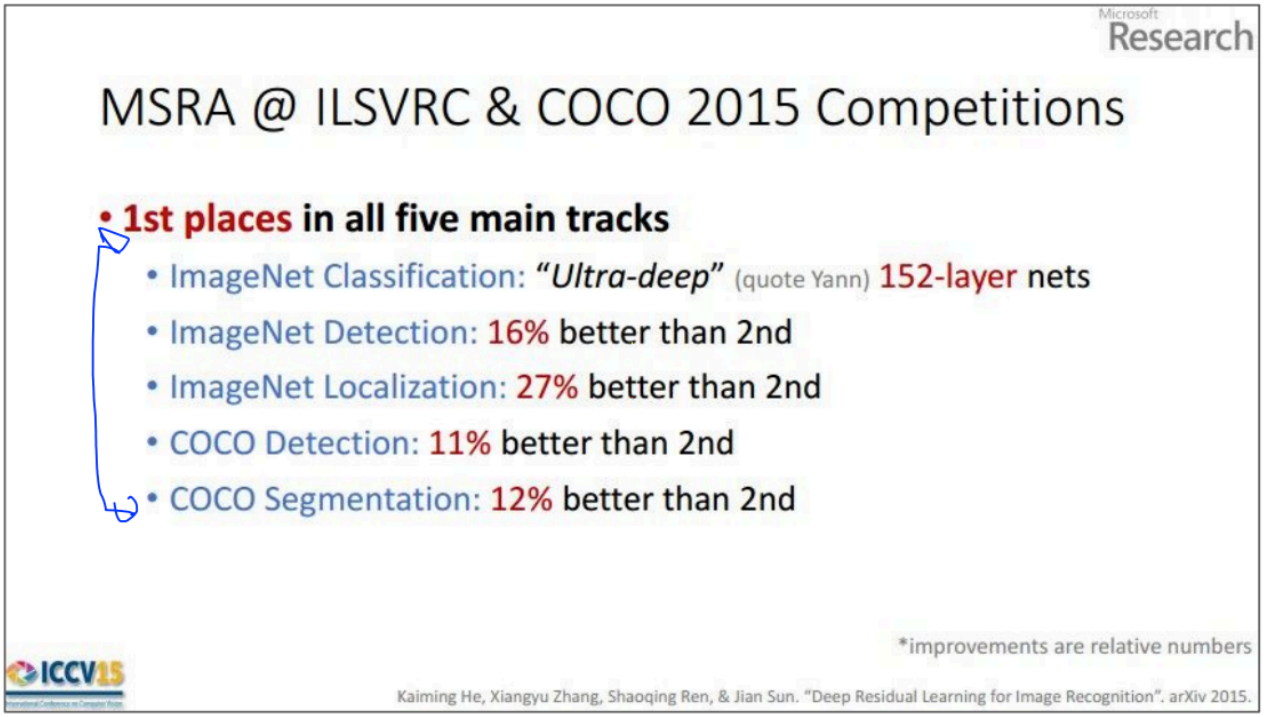


Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Case Study: ResNet [He et al., 2015]

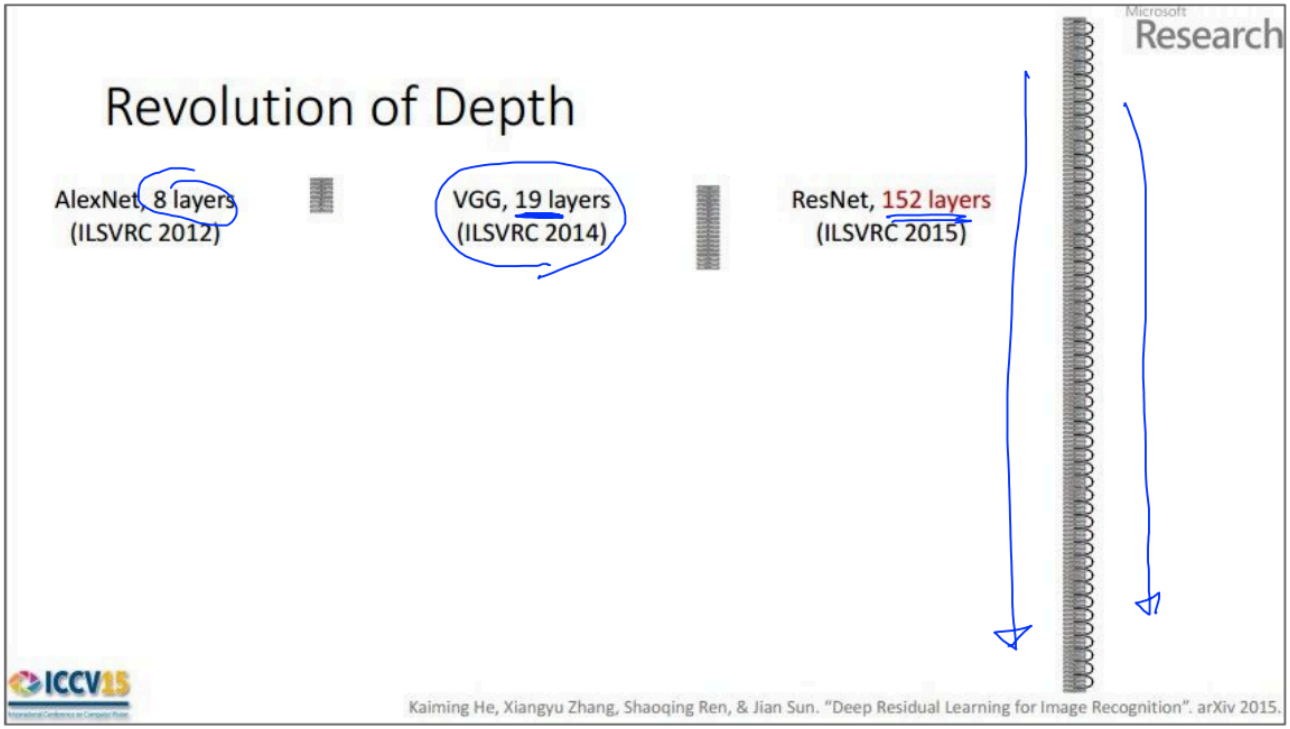
ILSVRC 2015 winner (3.6% top 5 error)



Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)



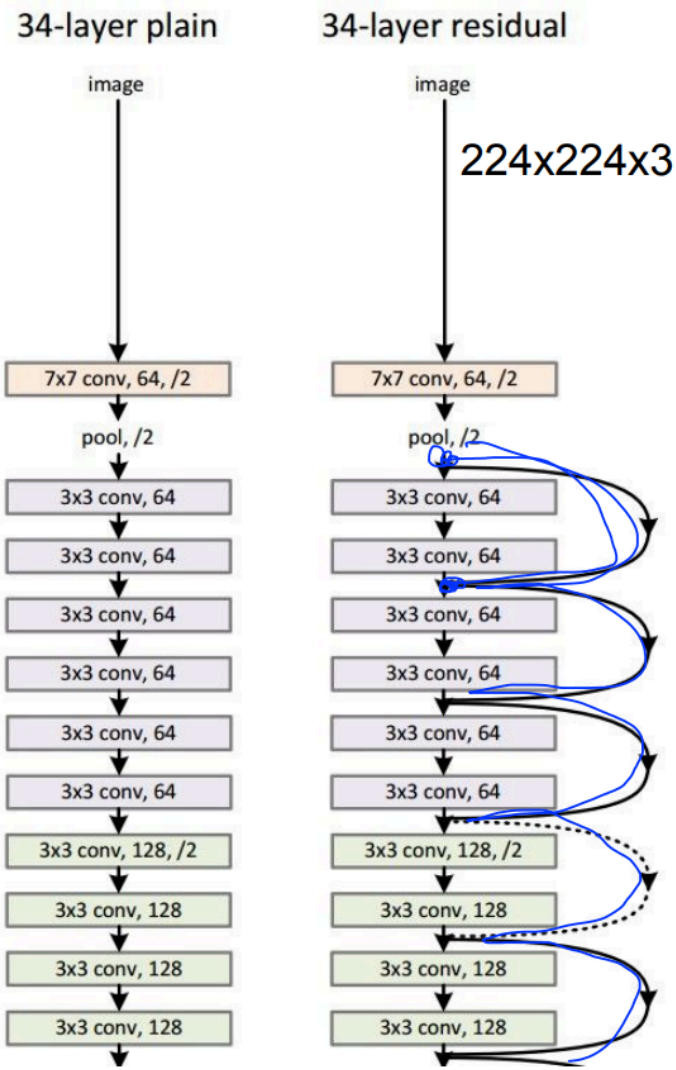
(slide from Kaiming He's recent presentation)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

Case Study: ResNet

[He et al., 2015]



Summary

- ConvNet stack “Conv+Pool+ReLU” + FC
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of Pool/FC layers