Convolutional Neural Networks

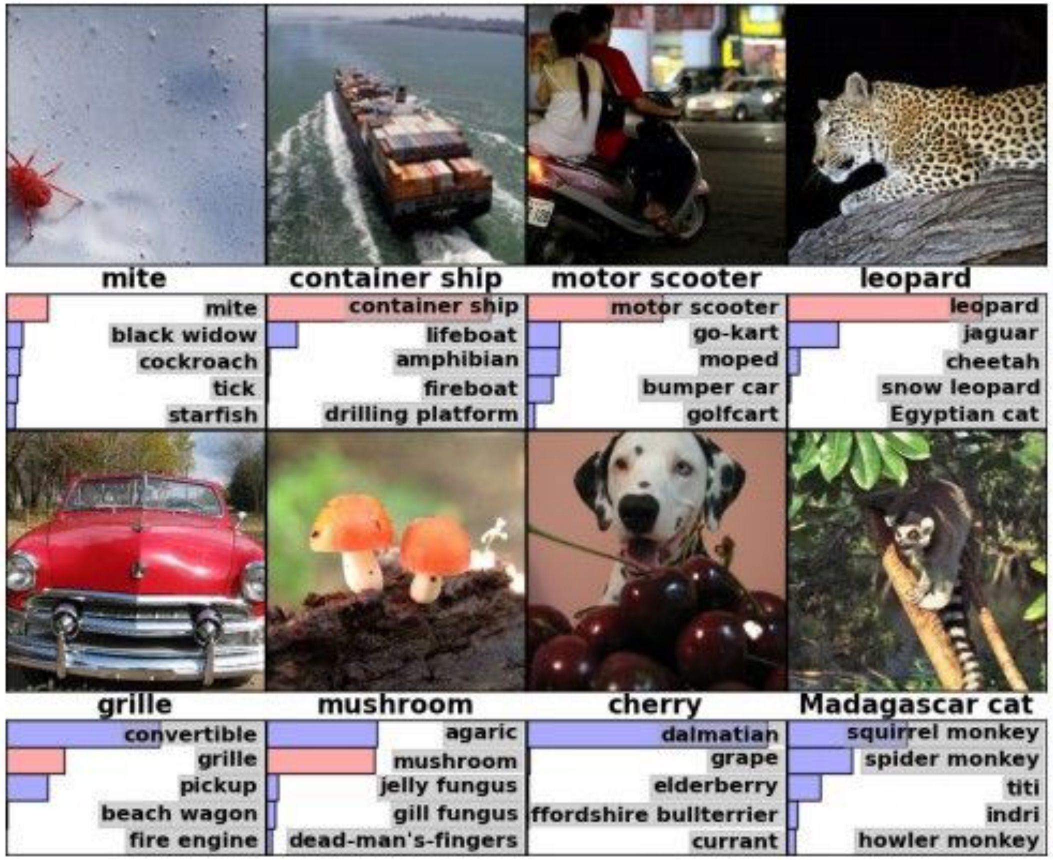
Longbin Jin

Artificial Inteligence & Computer Vision Laboratory

Slide reference: Stanford University cs231n, Fei-Fei Li

ConvNets are everywhere

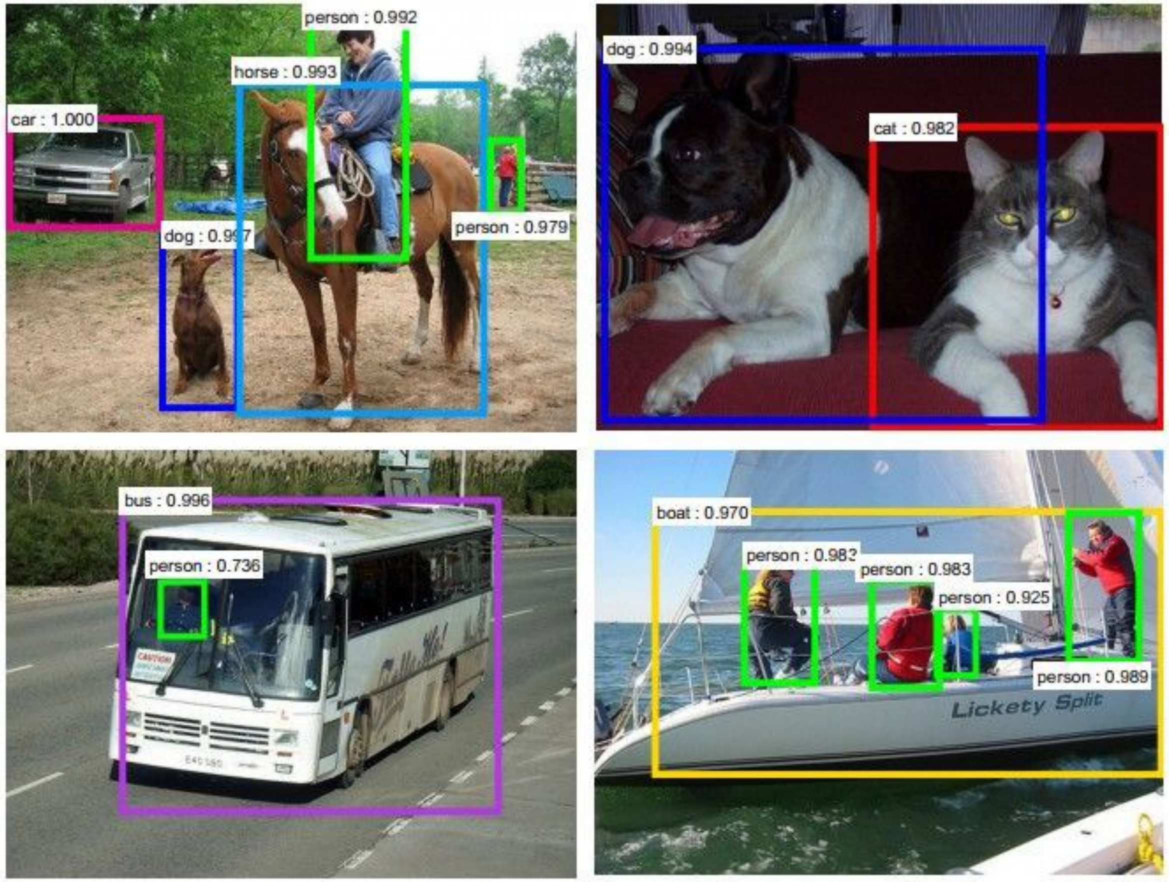
Classification



Retrieval

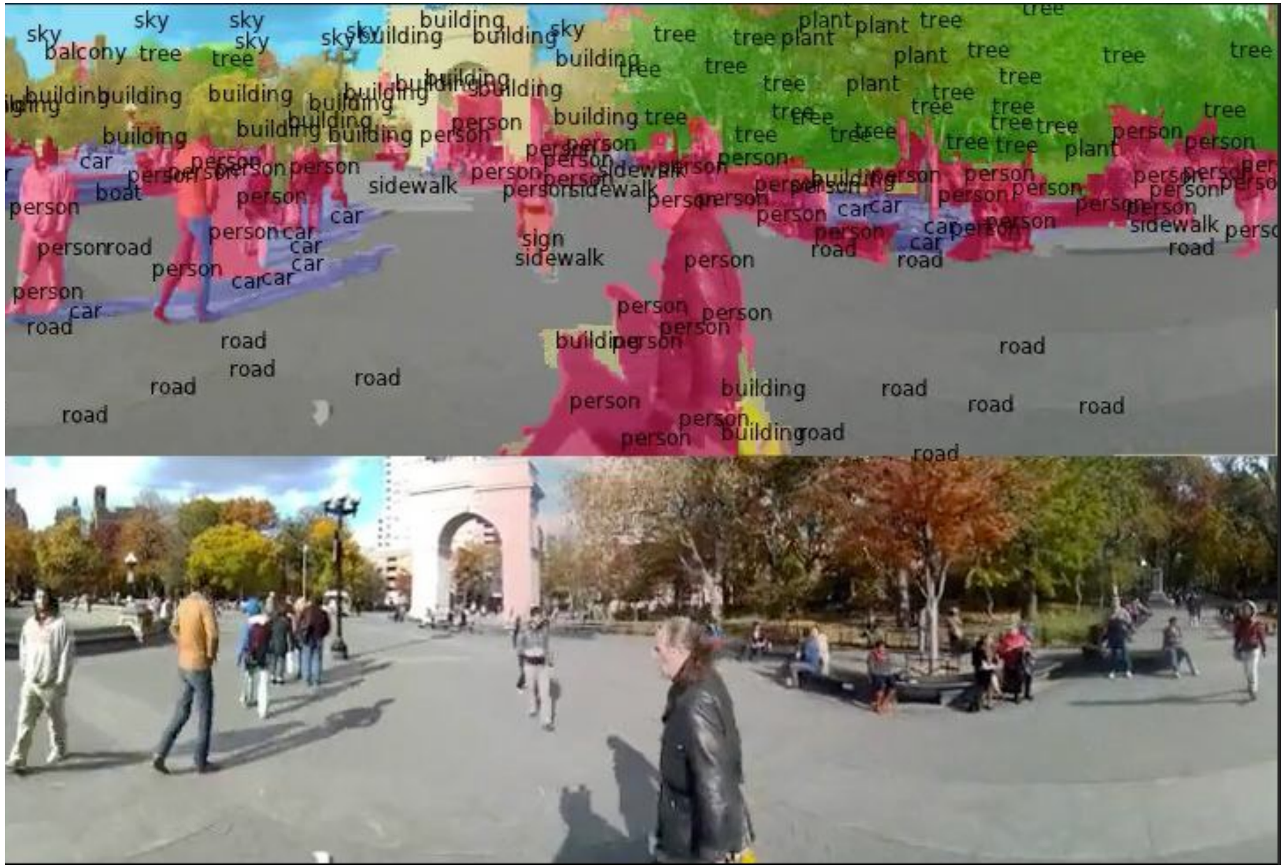
ConvNets are everywhere

Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015

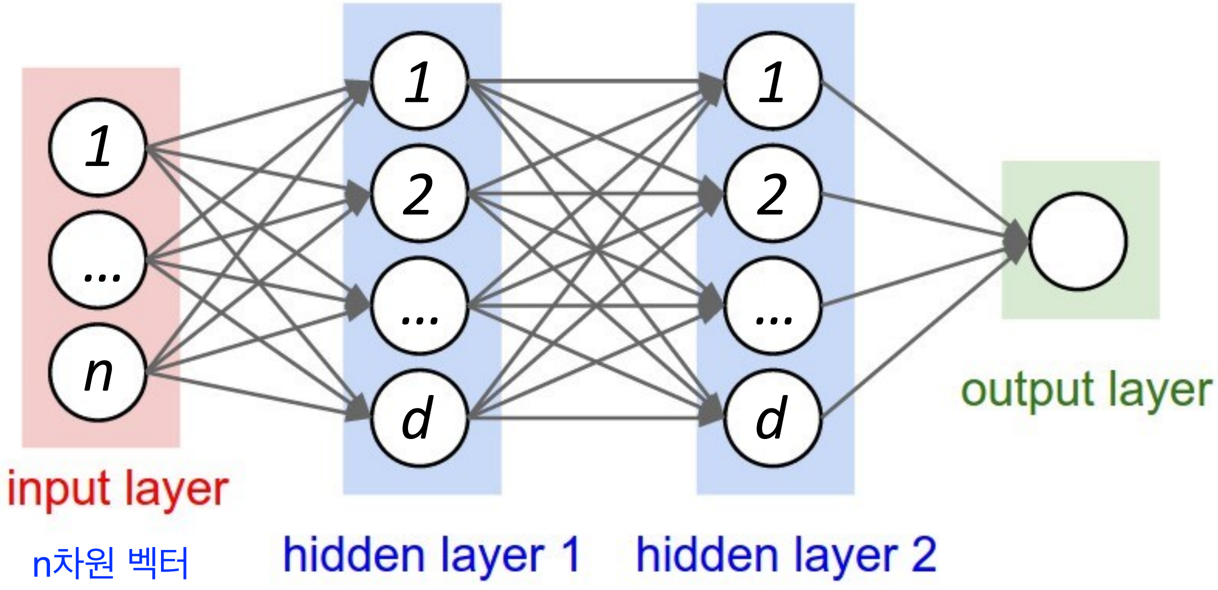
Segmentation



Figures copyright Clement Farabet, 2012

Recap: Neural Networks

- Regular Neural Networks
  - Receive an input (a single feature vector)
  - Transform it through a series of hidden layer
    - each neuron is fully connected to all neurons in the previous layer
  - Perform the classification at last fully connected layer (output layer)



Recap: Neural Networks

단점
1. parameter 너무 많음
image : 32 * 32 * 3 픽셀 존재

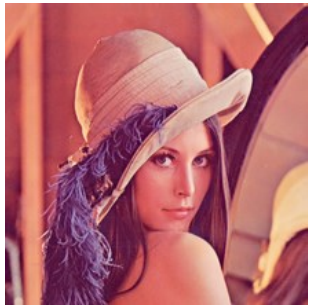
- Limitation
  - Too many parameters: $nd + d^2 + d$
  - Overfitting
  - Not scalable to large datasets
  - Not translation invariant

exam : 파라미터를 계산해보세요
= nd + d^2 + d + bias
bias 안 쓰면 틀림

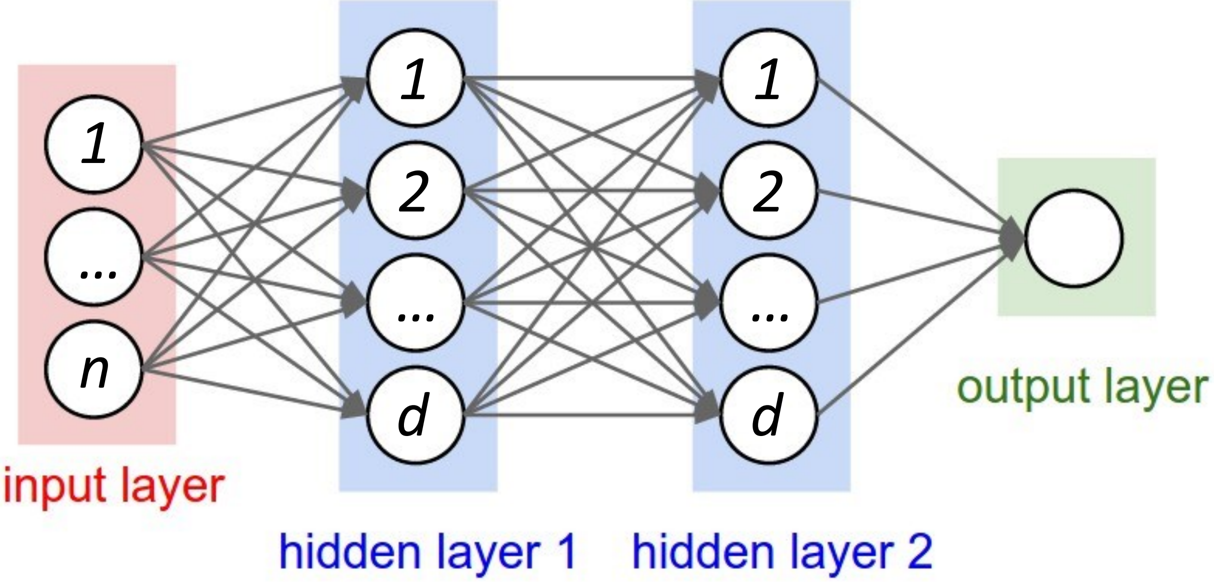
32 * 32 * 3 이상의 크기는 다룰 수 없음

32

32

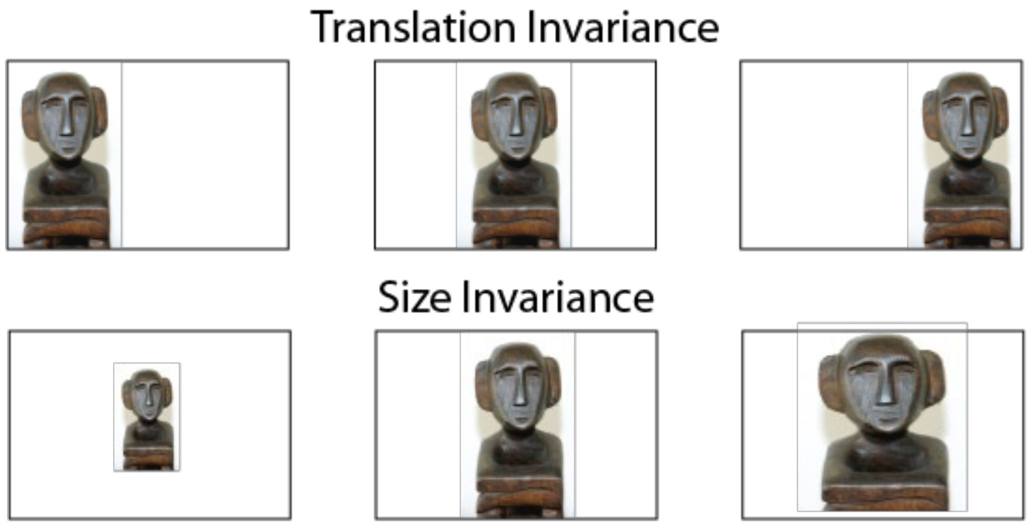


$n = 32 \times 32 \times 3 = 3072$

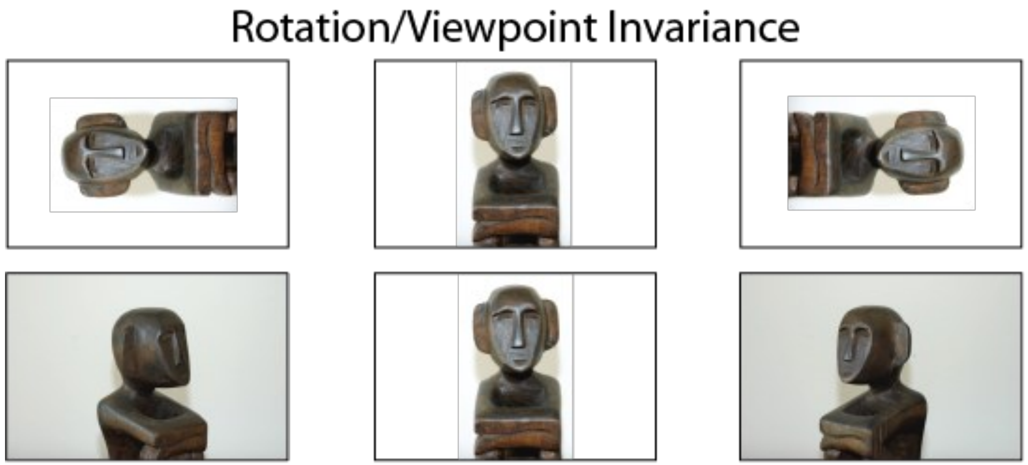


Invariant Representation

- Different kinds of invariances = 불변성



어디에 위치해도 같은 object로 분류됨



이렇게 돌아간다해도 달라지지 않음?

Prior Assumption to reduce parameter

1. Low-level features are local

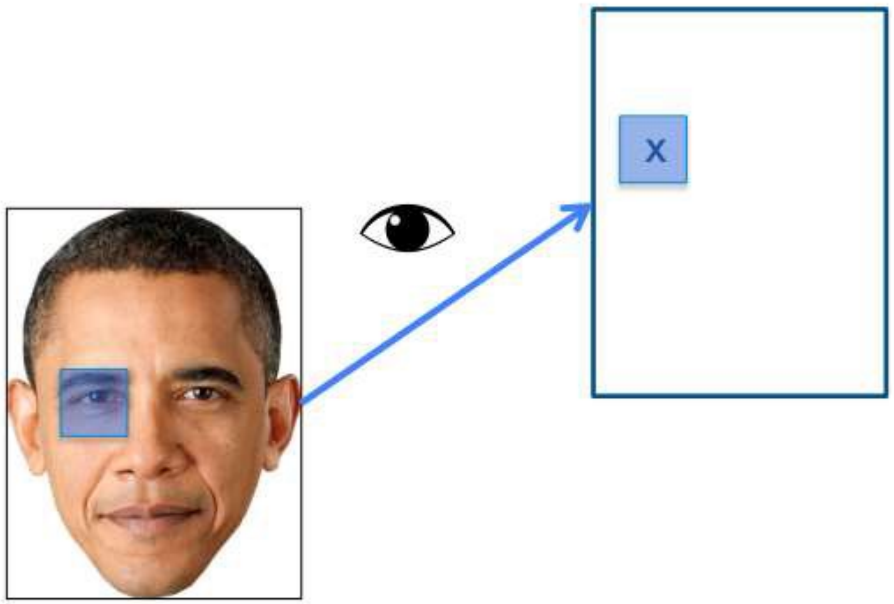
2. Features are translational invariant

3. High-level features are composed of low-level features

Prior Assumption to reduce parameter

1. Low-level features are local

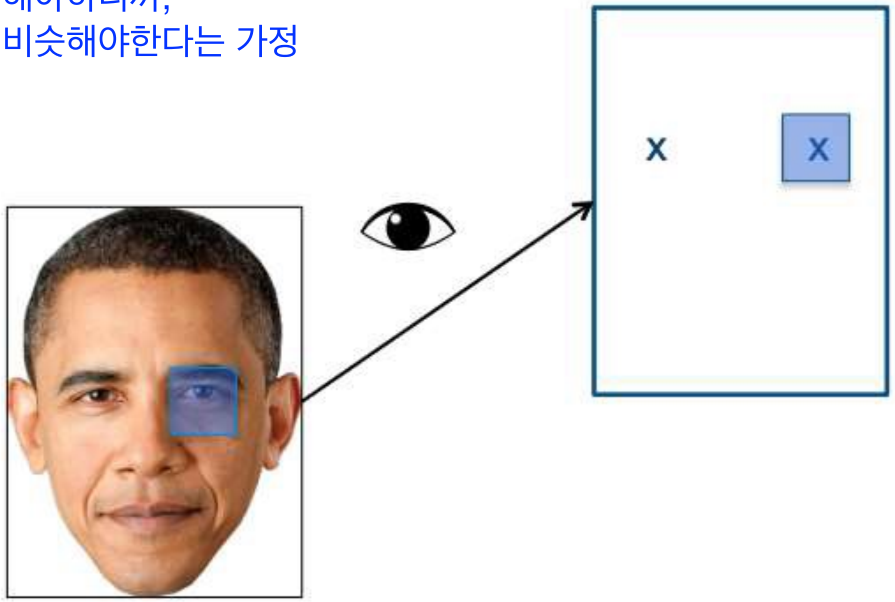
초기에는 전체 이미지의 전체 픽셀을 다 볼 필요 없고,
특정 픽셀과 주위 픽셀만 보고
어떤 특징이 있는지 뽑아냄



Prior Assumption to reduce parameter

2. Features are translational invariant

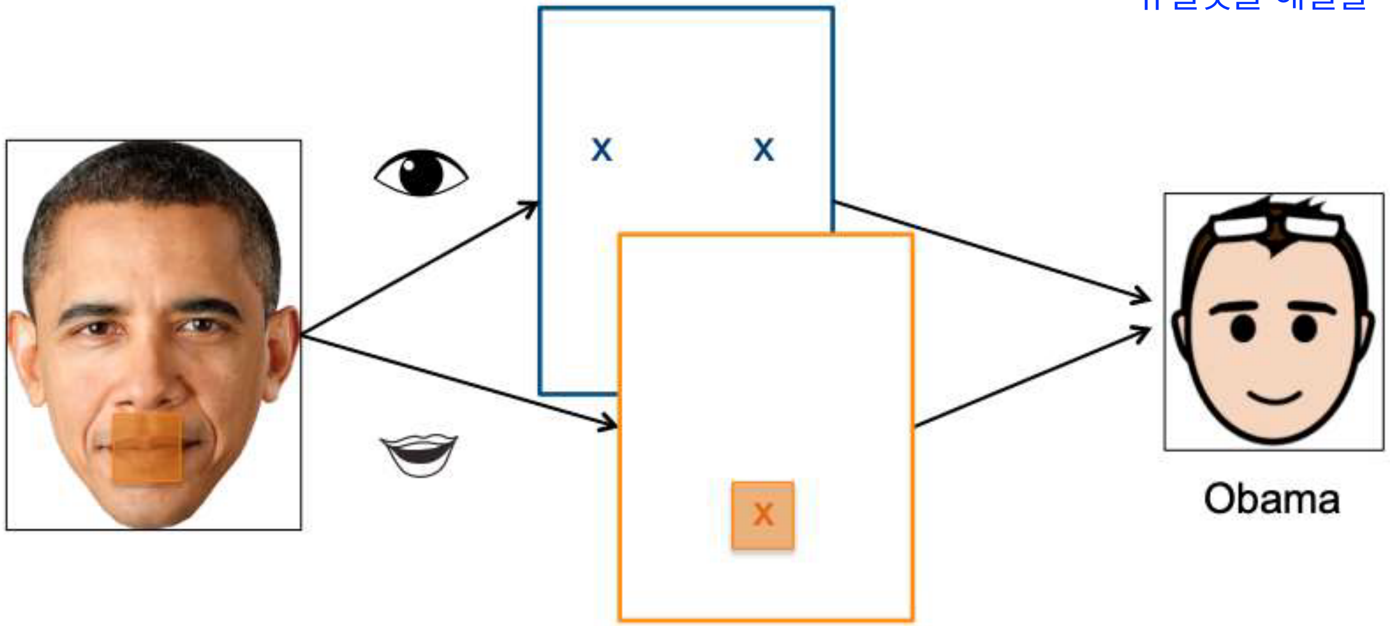
눈에 대한 두개의 픽셀은 비슷해야하니까,
비슷한 특징을 뽑아낸 픽셀은 비슷해야한다는 가정



Prior Assumption to reduce parameter

3. High-level features are composed of low-level features

눈이든 입이든 그런 특징을 조합해서
이게 사람인지, 동물인지 알아내서
뉴럴넷을 해결할 수 있음



Prior Assumption to reduce parameter

여러 문제 (위에 나온 단점)들이 해결되서 일반적인 성능이 올라감

1. Low-level features are local

2. Features are translational invariant

3. High-level features are composed of low-level features

- Fewer parameters
- Better generalization
- Better scalability to large datasets

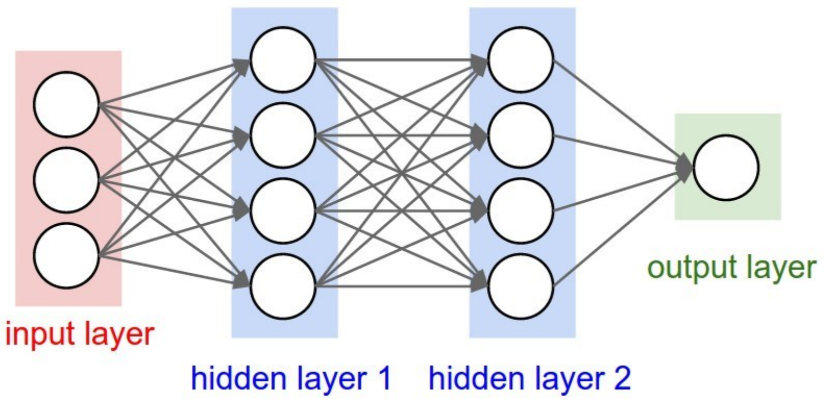
큰 모델도 처리할 수 있게됨

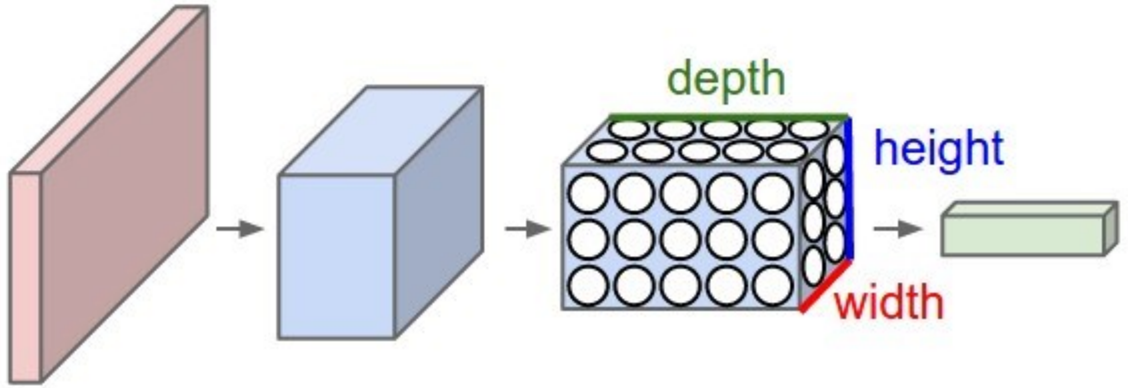
Convolutional Neural Networks (CNNs)

- CNNs (LeCun, 1989)
  - CNNs are a specialized kind of neural network for processing data that has a known grid-like topology.
  - 1-D grid (sound), 2-D grid (grey image), 3-D grid (color image).
  - Convolution is a specialized kind of linear operation.

중요 -> 아래에서 설명

격자 데이터를 잘 처리함 (CNN이)
격자(grid) : matrix라고 이해해도 됨
흔히 쓰는 컬러 이미지는 3차원



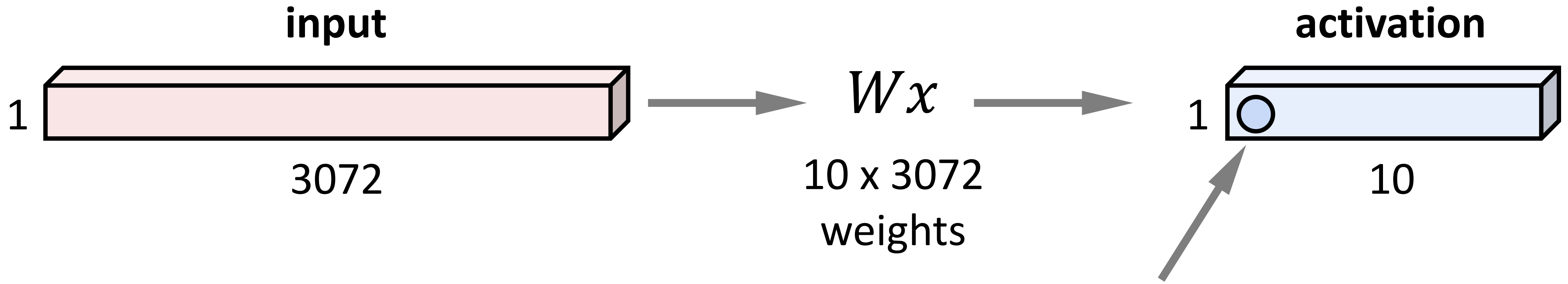


왜 중요한지 설명

Fully Connected Layer

- 32x32x3 image -> stretch to 3072x1

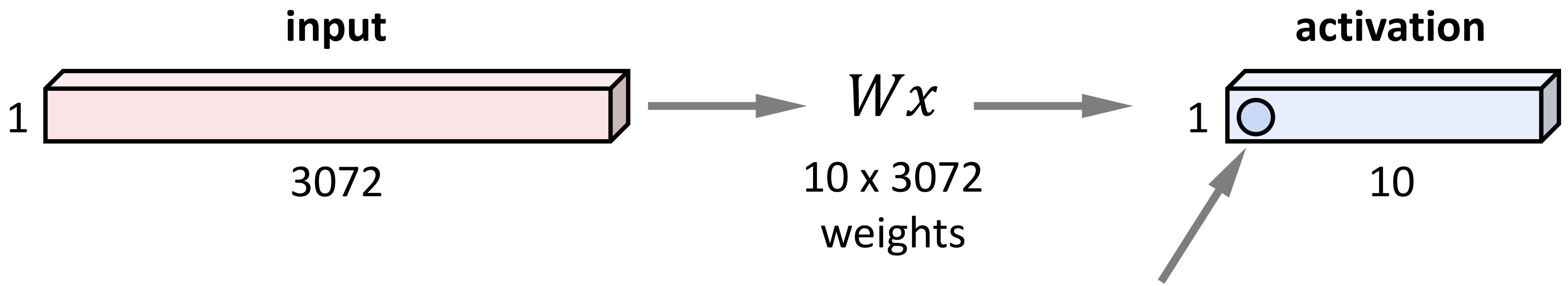
여기 설명 이해 몬하게씀



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Fully Connected Layer

- 32x32x3 image -> stretch to 3072x1



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Linear classifier: One template per class

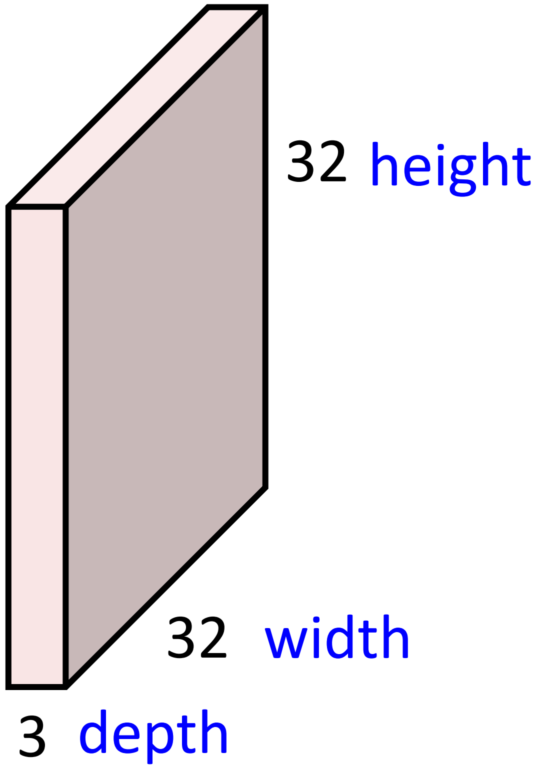
중심에 위치한 자동차만 나오고 있음
만약 자동차의 모서리만 나오면
잘 이해하고 처리할 수 없음

Convolution Layer

input

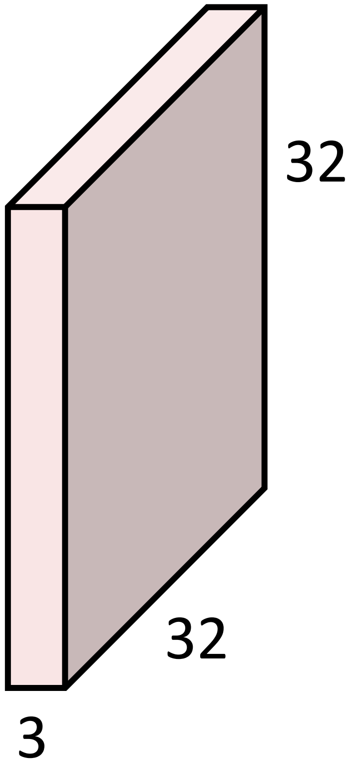
- 32x32x3 (width, height, depth) image -> preserve spatial structure

장점 : 공간적인 구조를 기억함

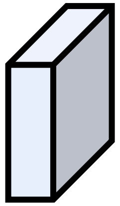


Convolution Layer

- 32x32x3 image



5x5x3 filter



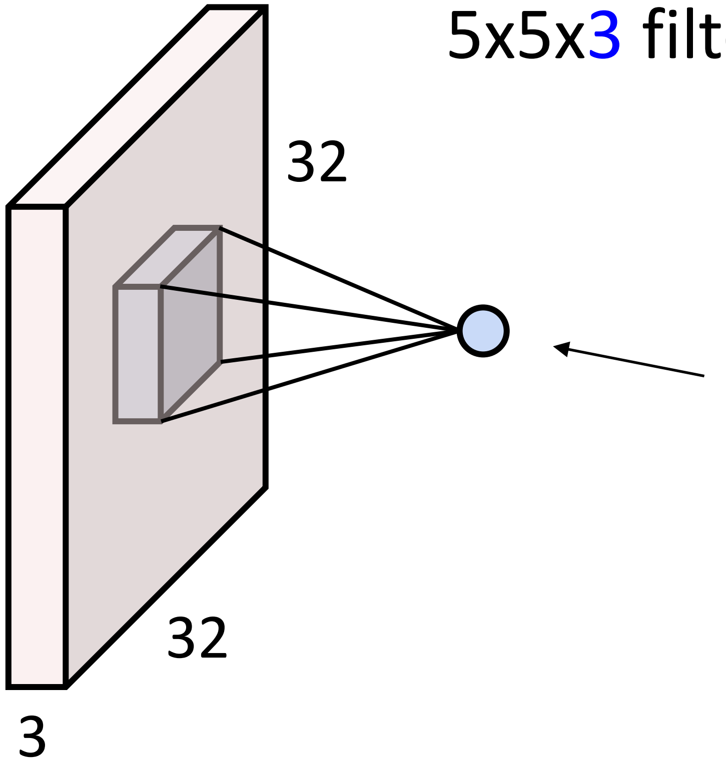
연산
Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”
내적

Filter always extend the full depth of input volume

5x5, 3x3 다 가능하지만
depth 차원은 5x5x3 같다고 가정?

Convolution Layer

- 32x32x3 image x

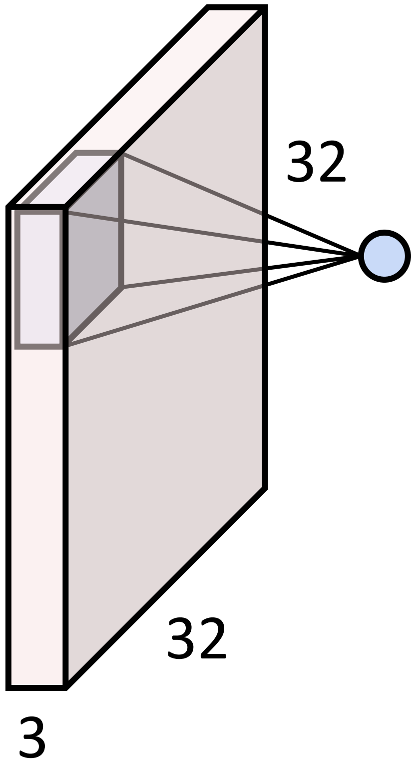


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

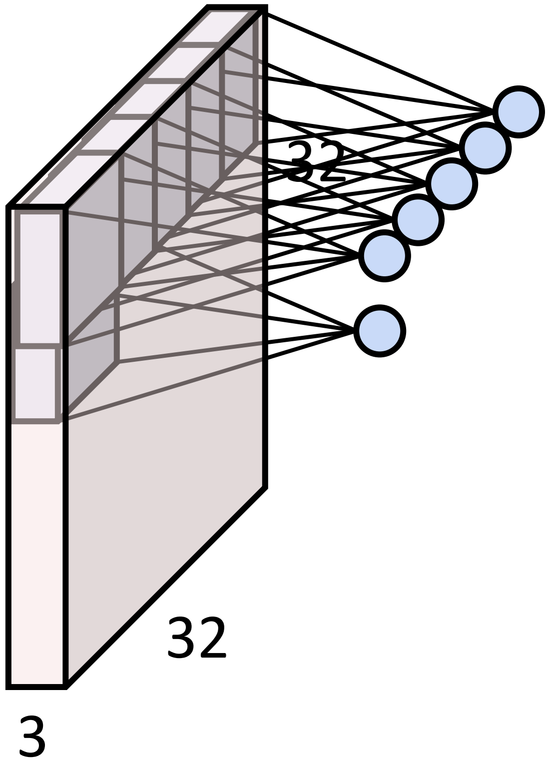
Convolution Layer

- 32x32x3 image x



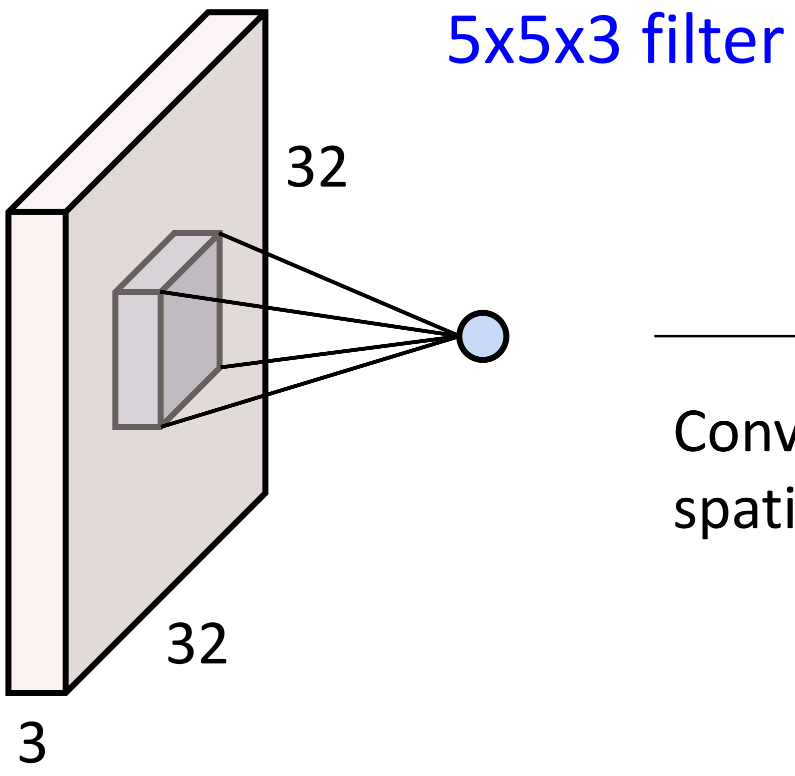
Convolution Layer

- 32x32x3 image x



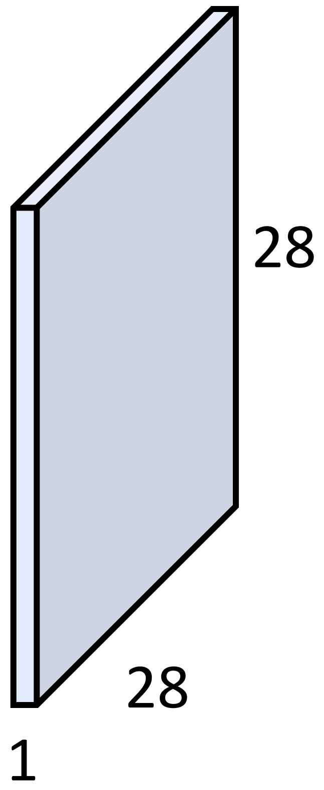
Convolution Layer

- 32x32x3 image x



Convolve (slide) over all spatial location

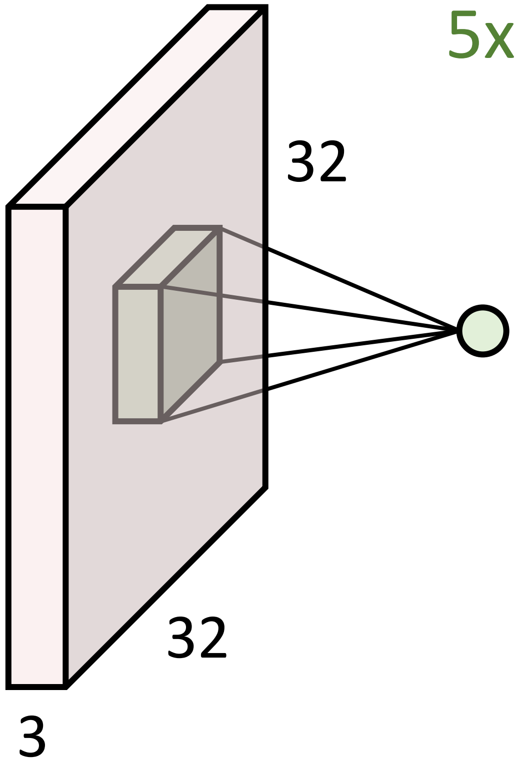
activation map



Consider a second, green filter

Convolution Layer

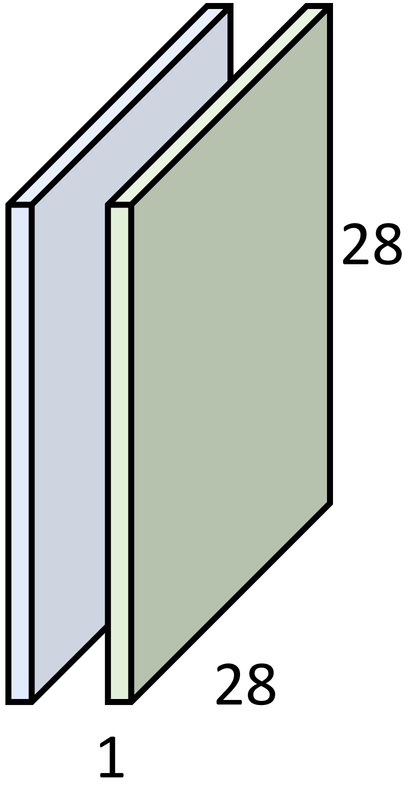
- 32x32x3 image x



5x5x3 filter

Convolve (slide) over all spatial location

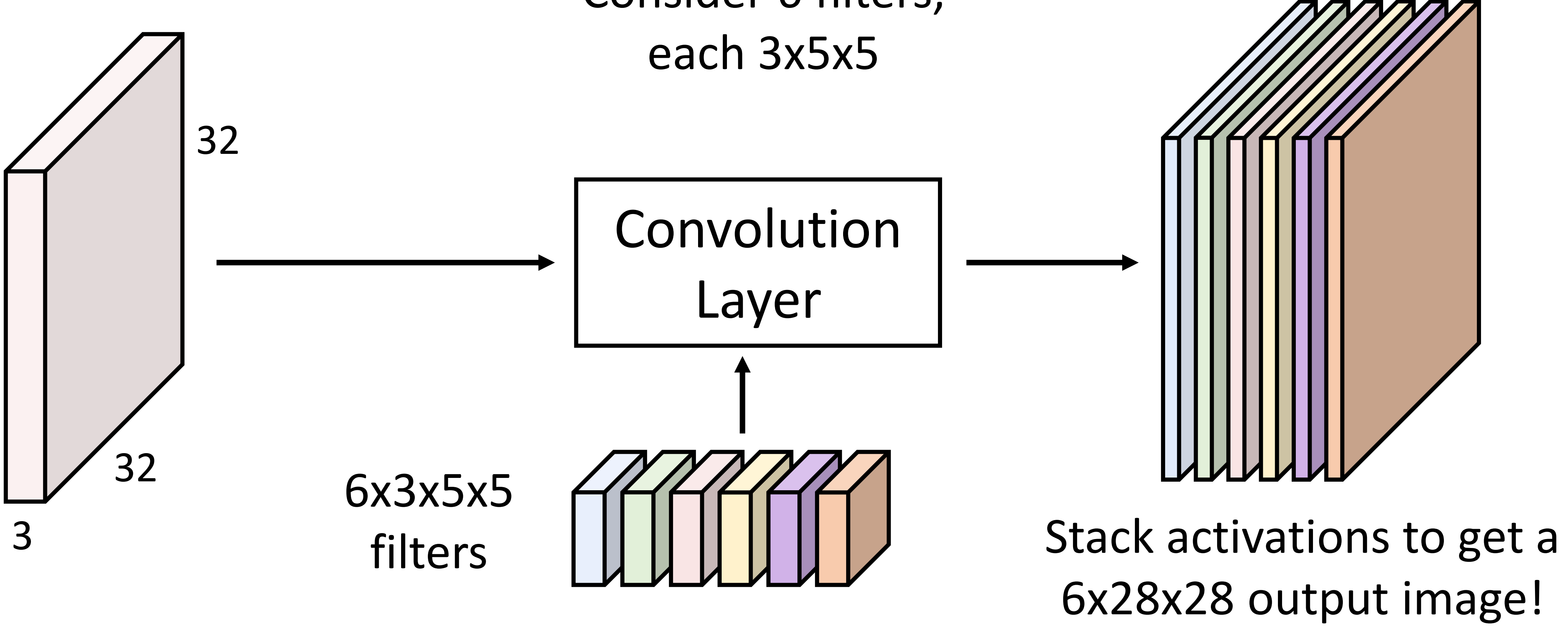
activation map



Convolution Layer

32x32x3에서 차원이 변경됨 => 원래 돌릴때 이렇게 함

- 3x32x32 image



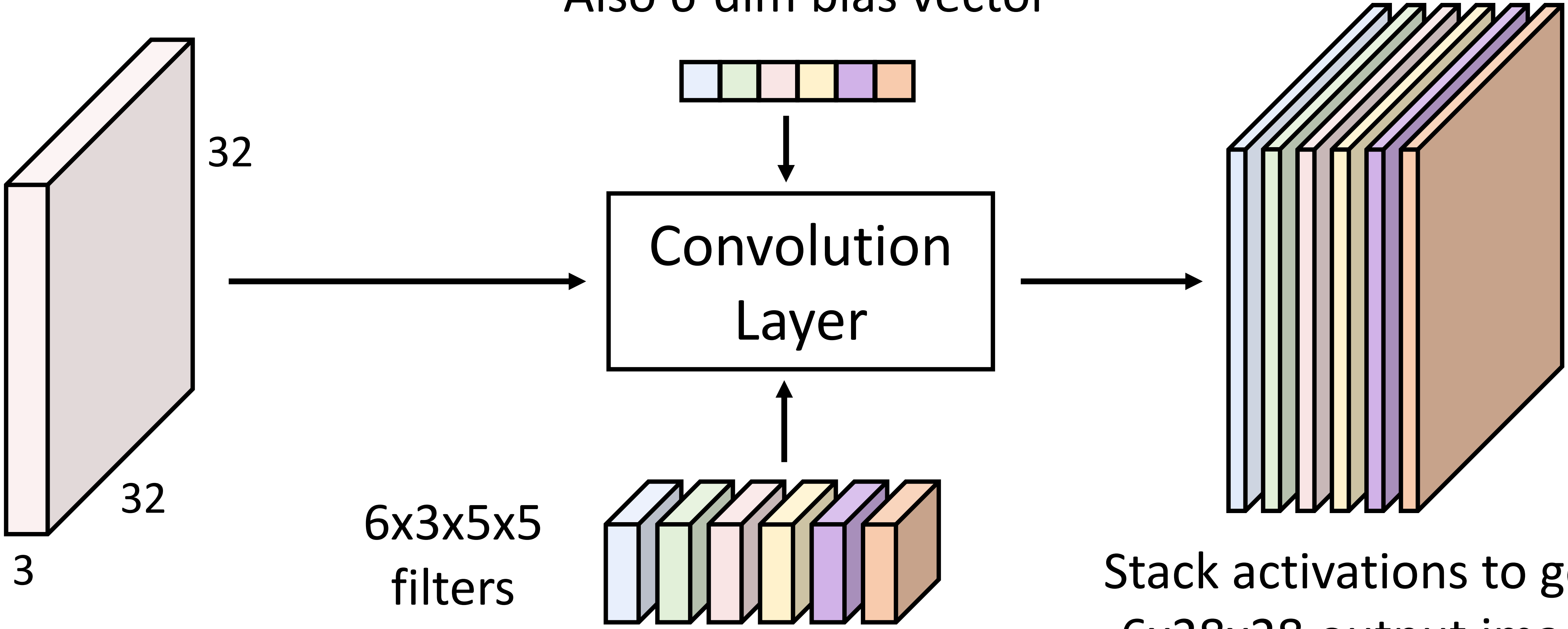
Convolution Layer

- 3x32x32 image

bias 가 항상 존재하는데
0으로 존재한다고 가정

Also 6-dim bias vector

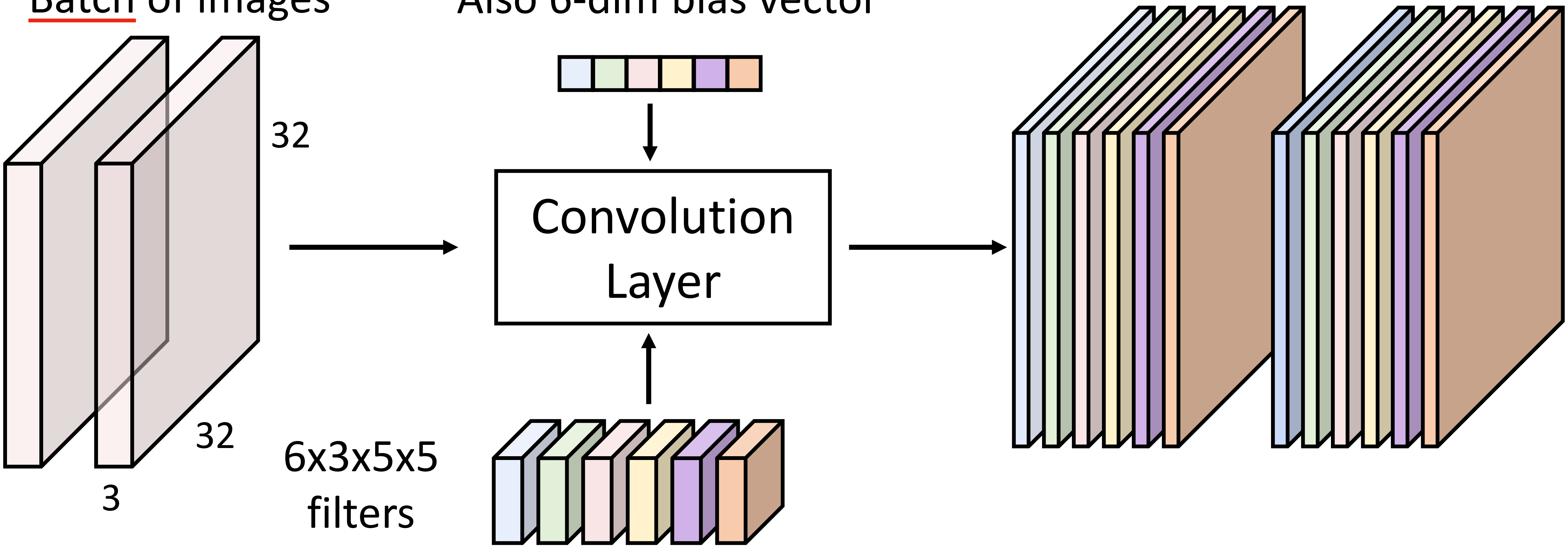
28x28 grid, at each point a 6-dim vector



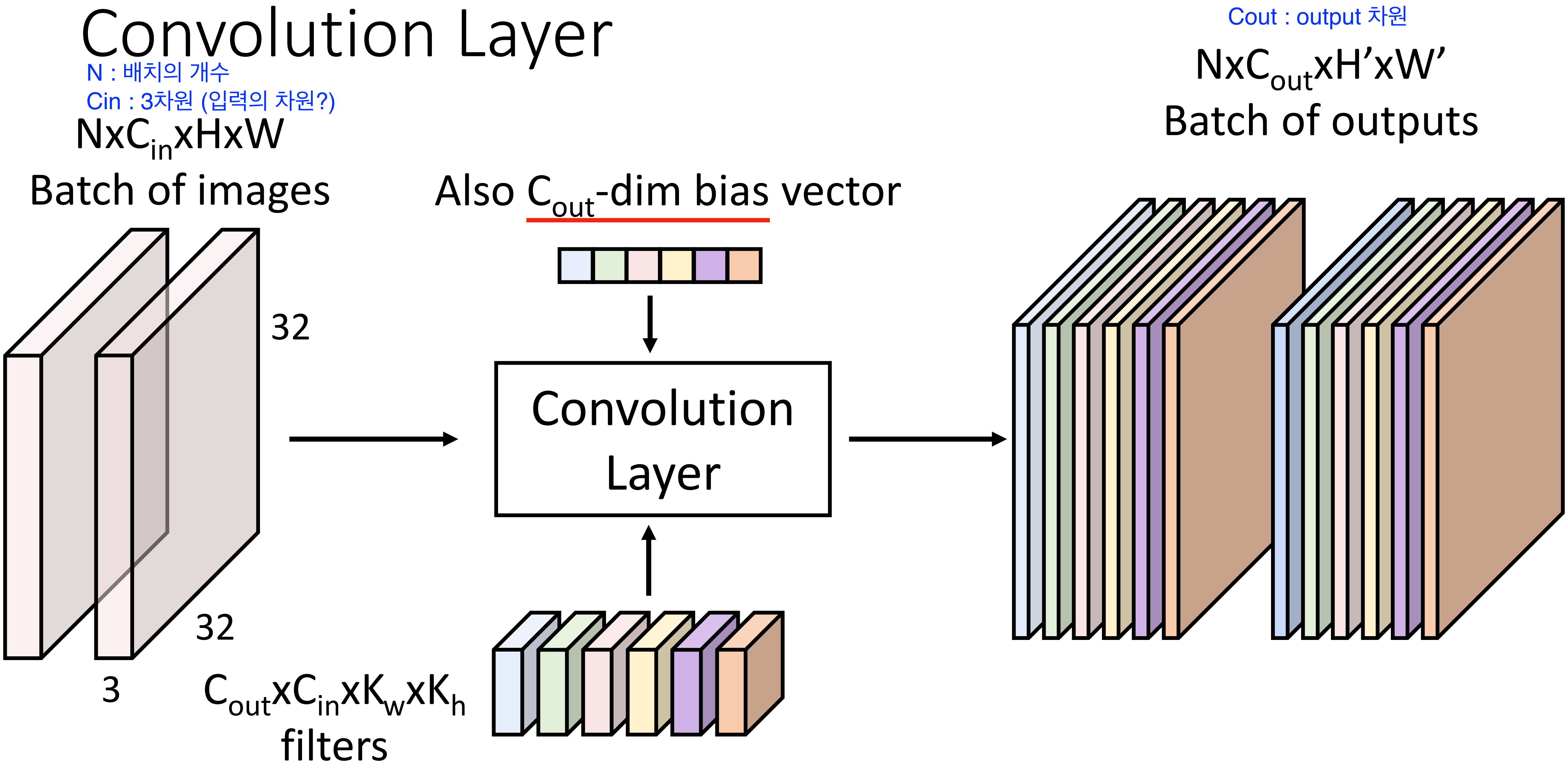
Stack activations to get a 6x28x28 output image!

Convolution Layer

2개의 이미지
2x3x32x32
Batch of images

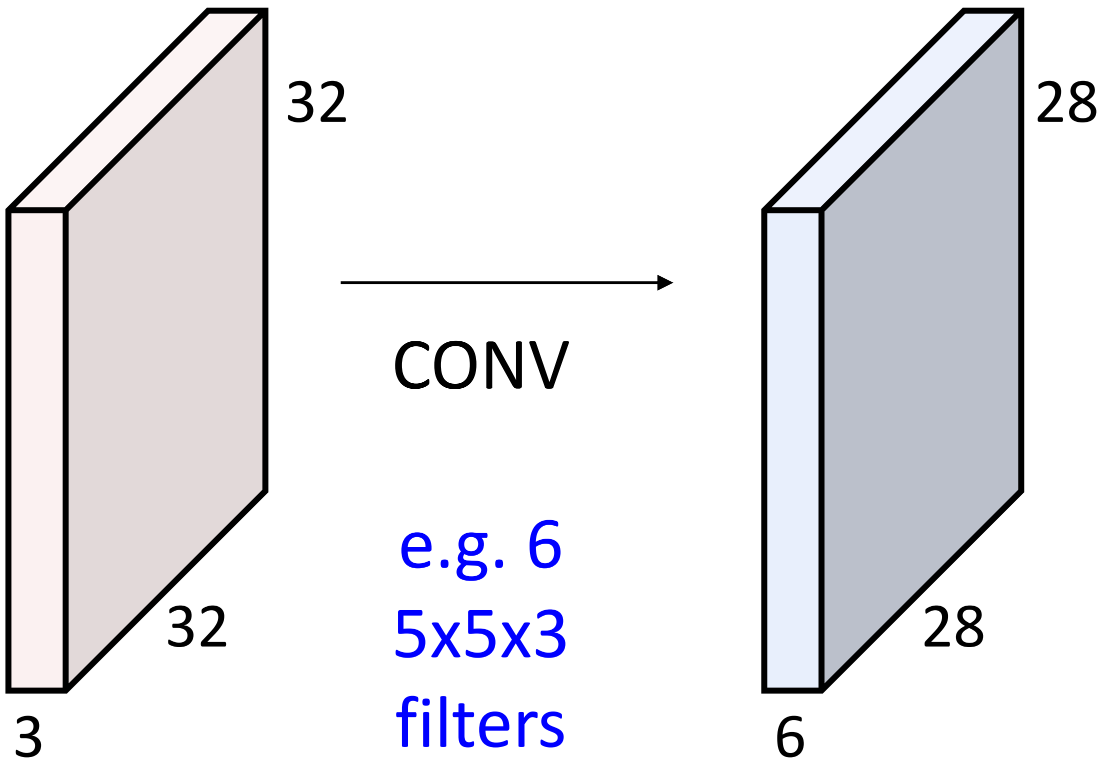


Convolution Layer



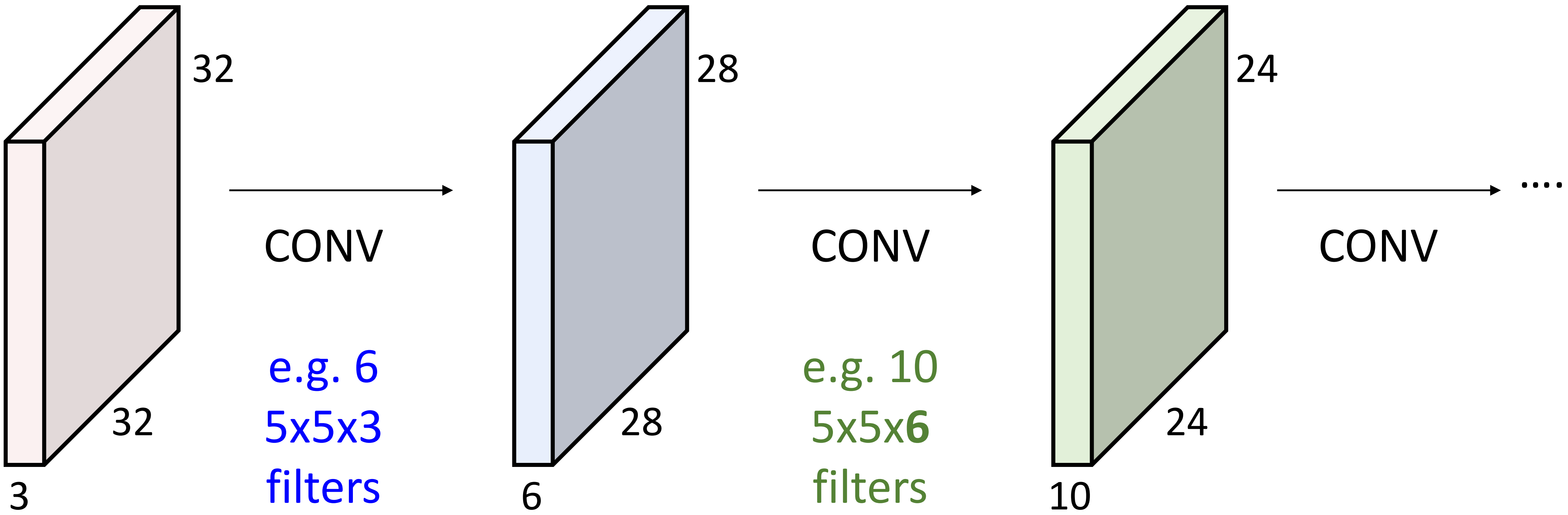
ConvNet

- ConvNet is a sequence of Convolution Layers



ConvNet

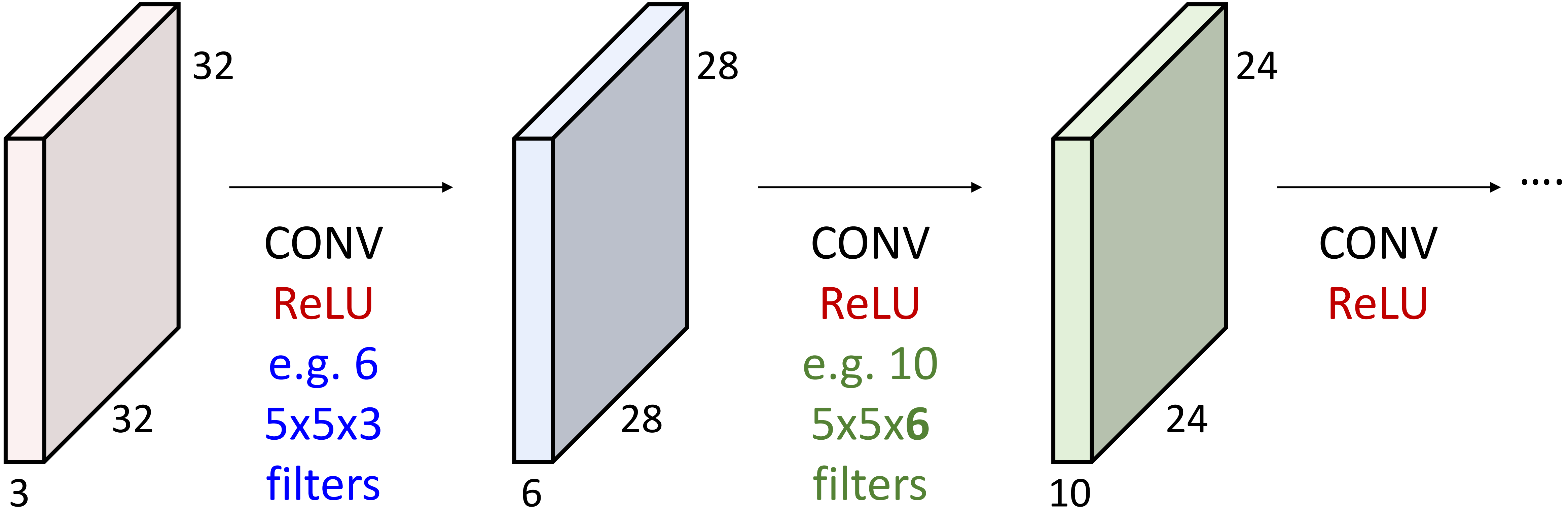
- ConvNet is a sequence of Convolution Layers



ConvNet

- ConvNet is a sequence of Convolution Layers, interspersed with activation functions

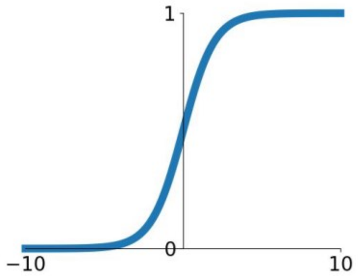
relu : activation func



Activation Functions

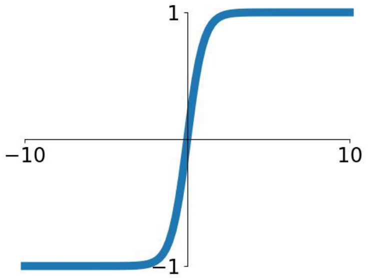
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



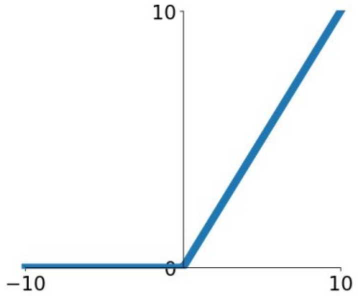
tanh

$$\tanh(x)$$



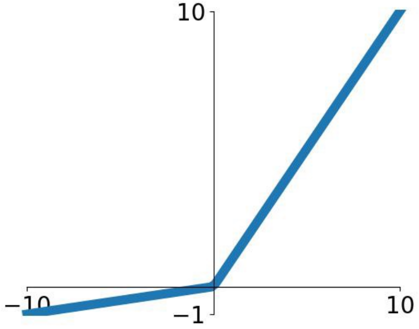
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

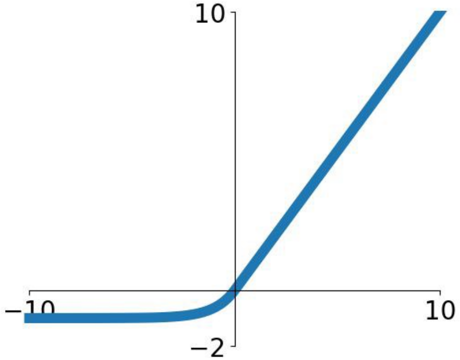


Maxout

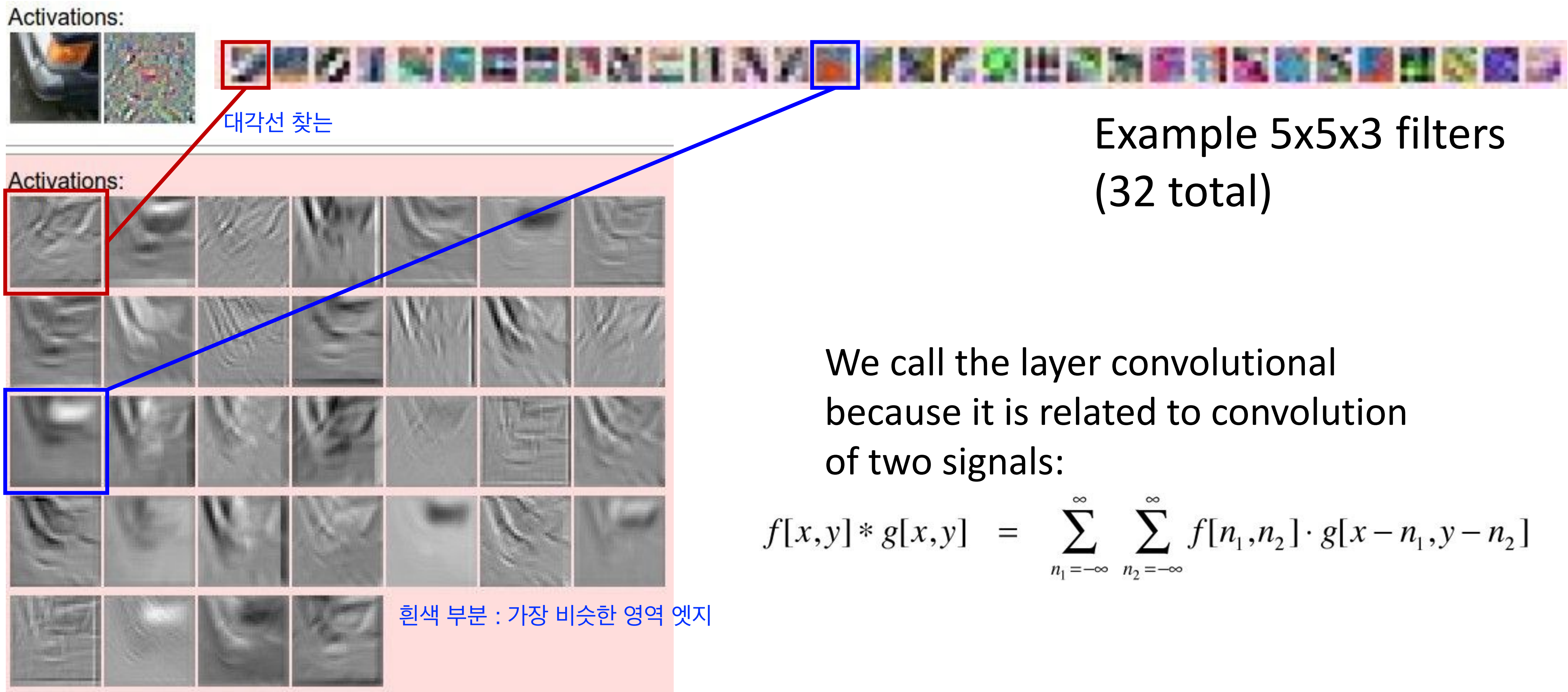
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



What do convolutional filters learn?

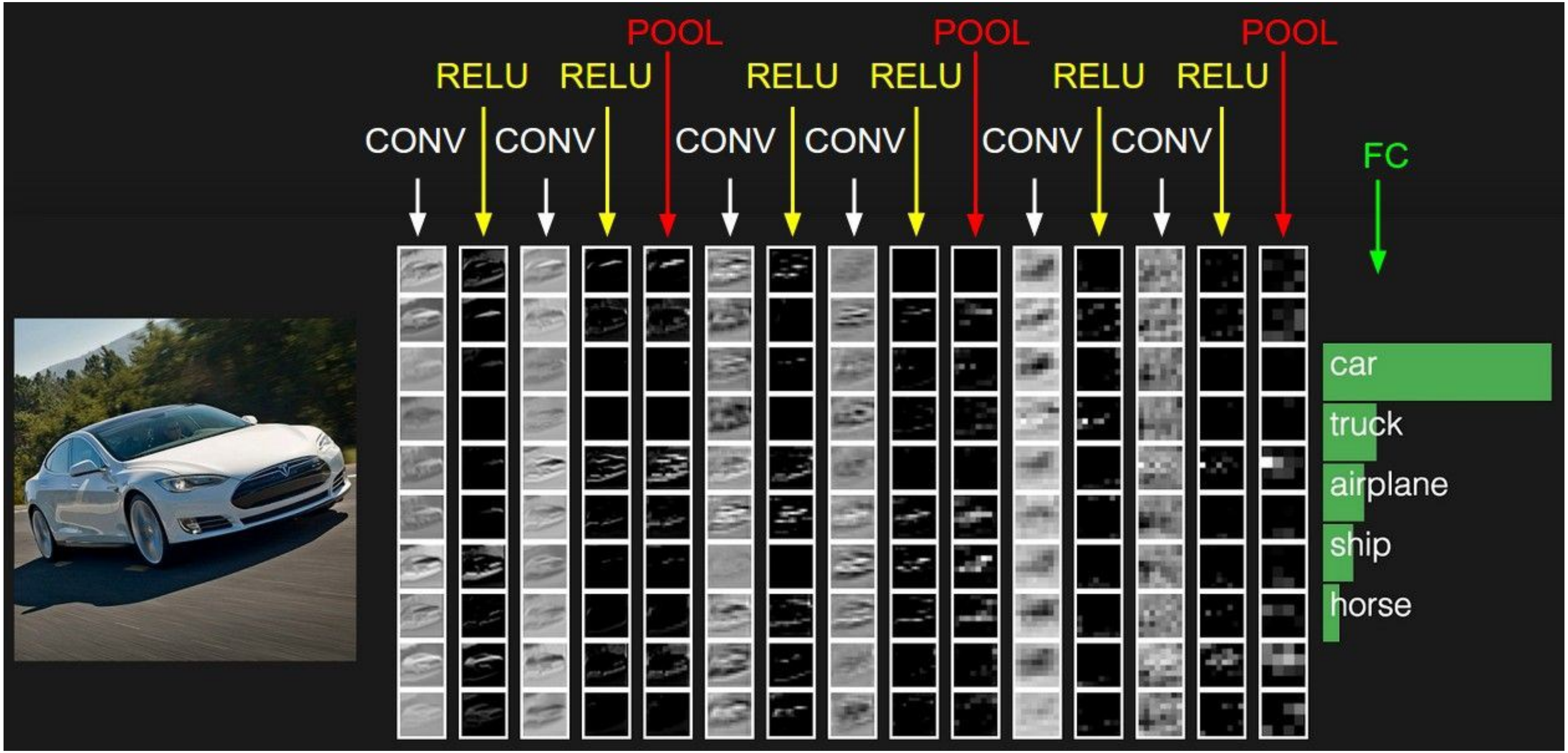


Example 5x5x3 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

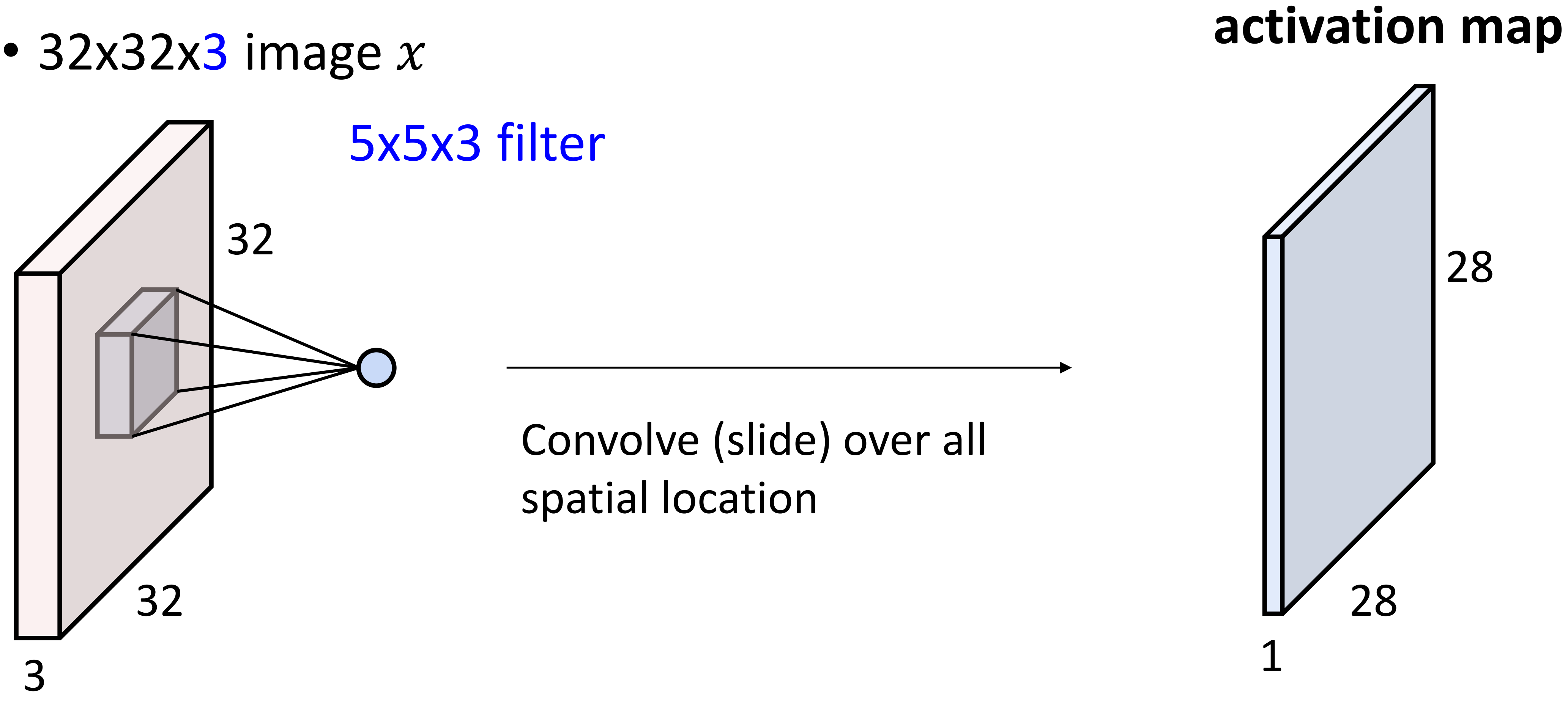
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$

ConvNet

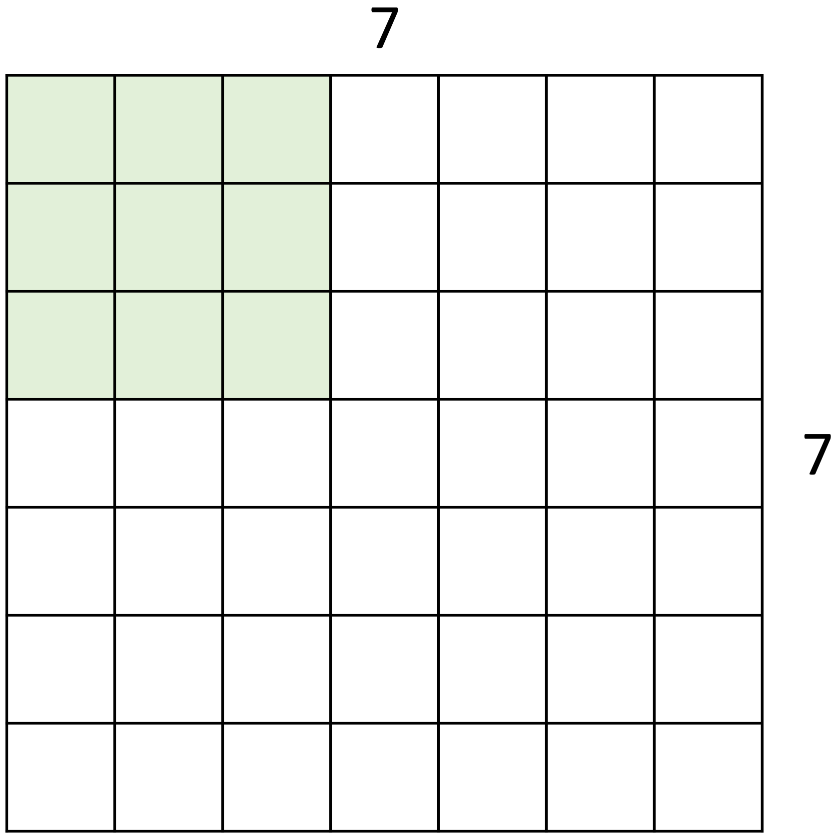

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Convolution Layer

- 32x32x3 image x

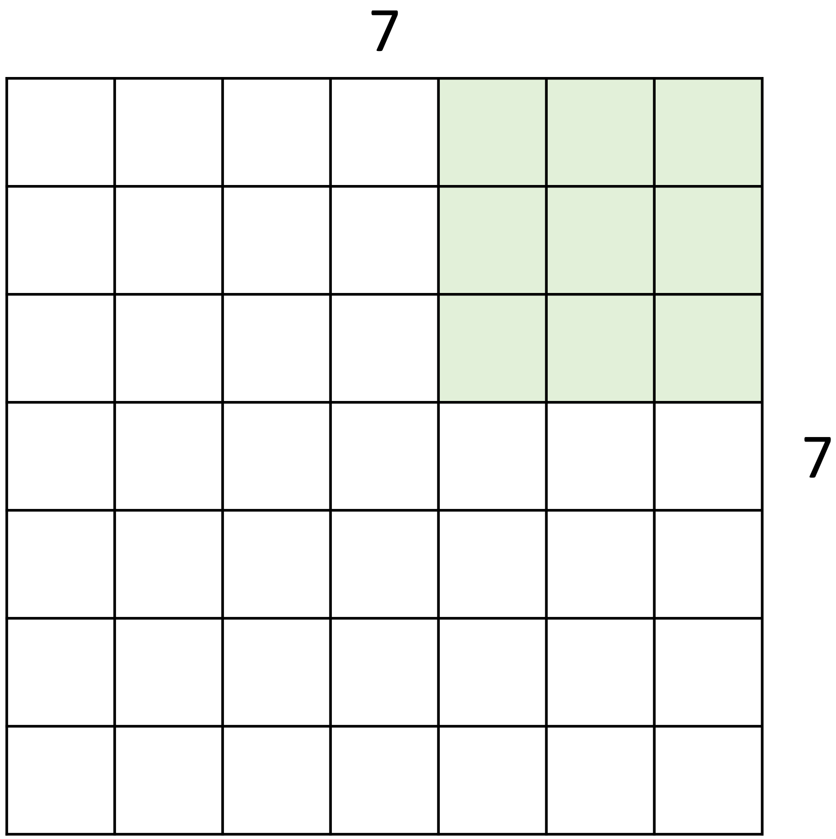


A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions

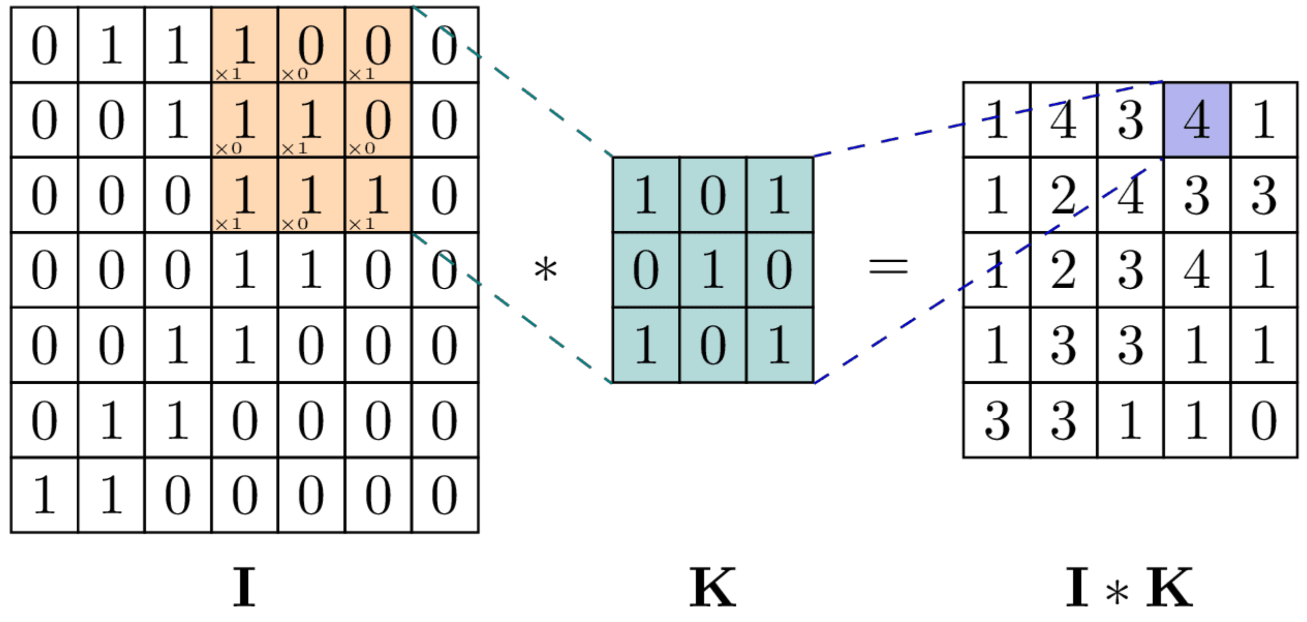


7x7 input (spatially)
assume 3x3 filter

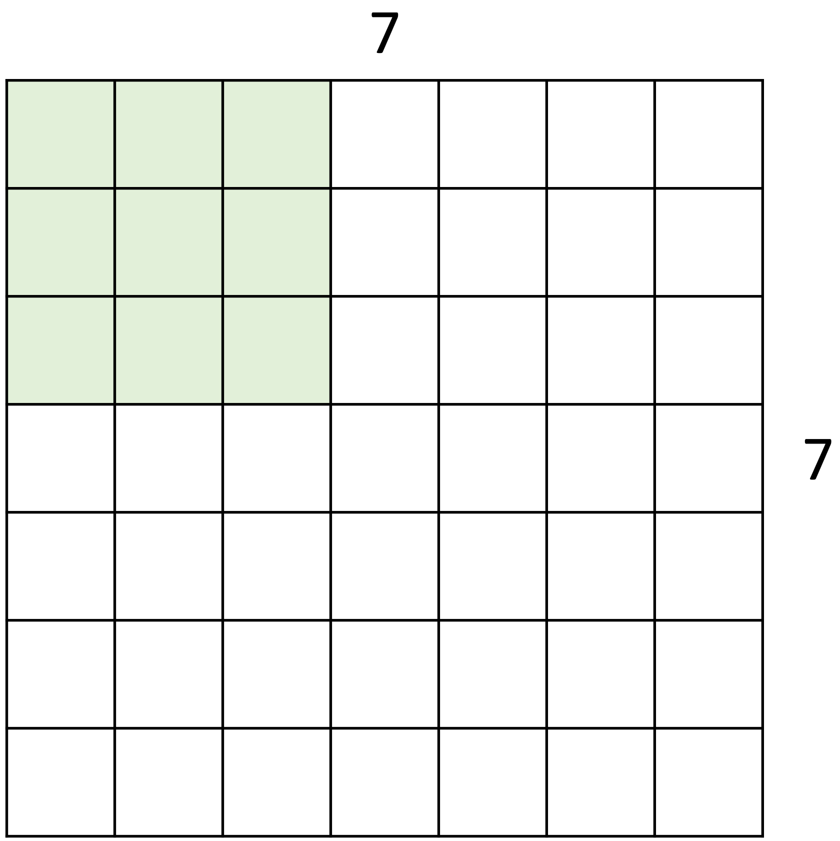
한칸씩 5번 이동하니까

=> 5x5 output

A closer look at spatial dimensions

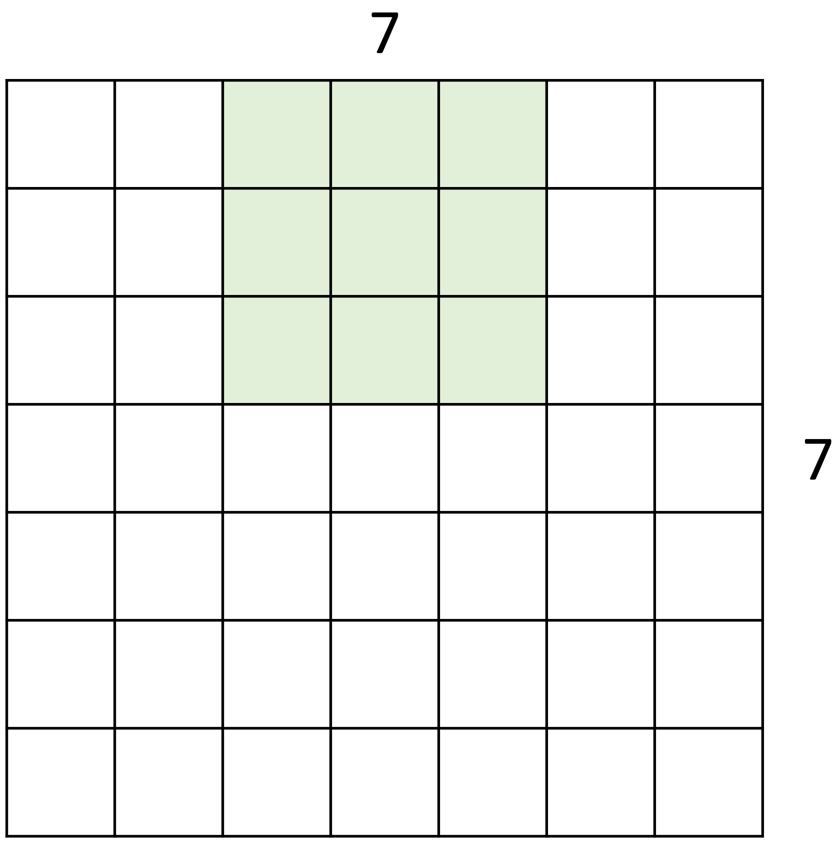


A closer look at spatial dimensions



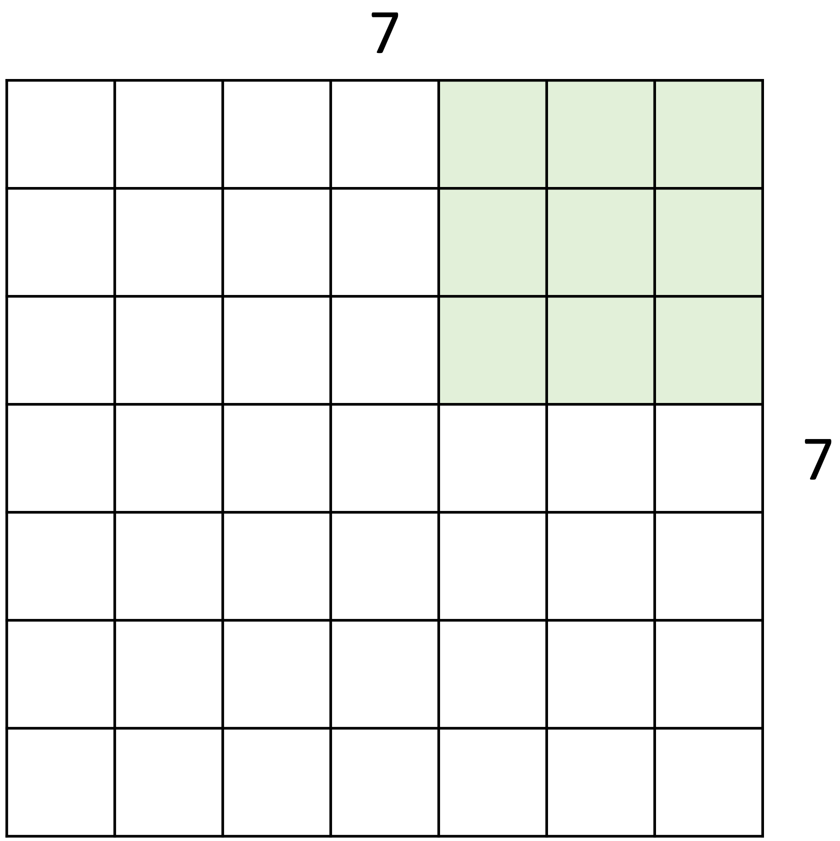
7x7 input (spatially)
assume 3x3 filter
Applied **with stride 2**

A closer look at spatial dimensions



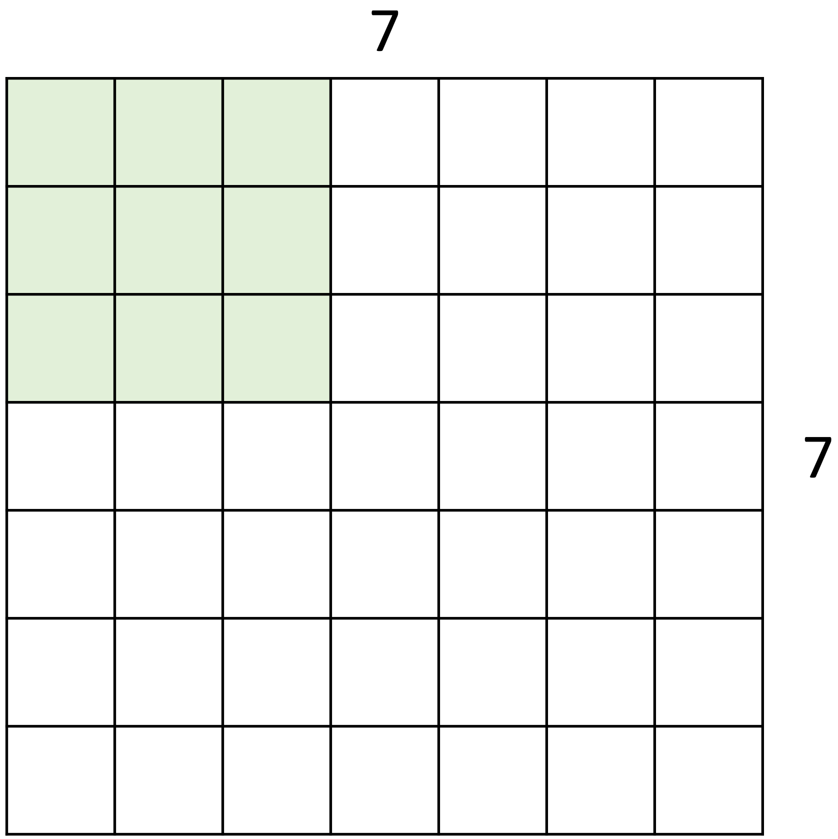
7x7 input (spatially)
assume 3x3 filter
Applied **with stride 2**

A closer look at spatial dimensions



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

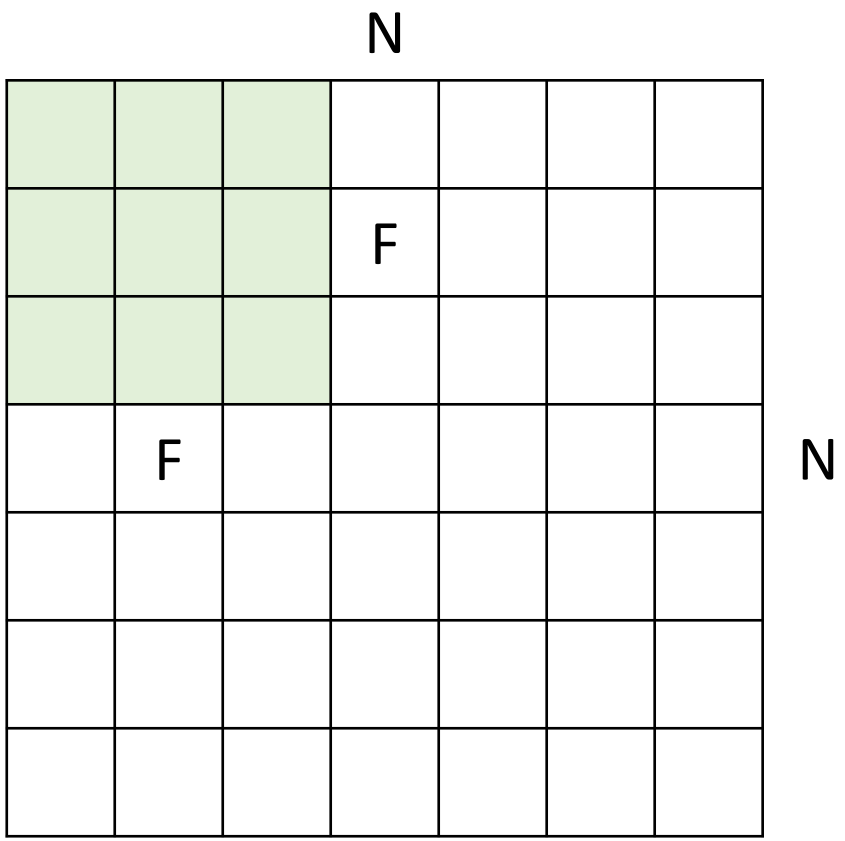
A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter
Applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.
마지막 열과 행의 정보가 손실되기 때문에 사용 안 함

A closer look at spatial dimensions



Output size:
(N-F) / stride + 1

e.g. N = 7, F = 3
stride 1 => (7 – 3)/1 + 1 = 5
stride 2 => (7 – 3)/2 + 1 = 3
stride 3 => (7 – 3)/3 + 1 = 2.33

그래서 쓰는게 padding

In practice: Common to zero pad the border

그림에서는 실수로 6x6으로 들어갔는데
실제로는 7x7이 맞음

사이즈 유지를 하기 위해 padding 하는 것

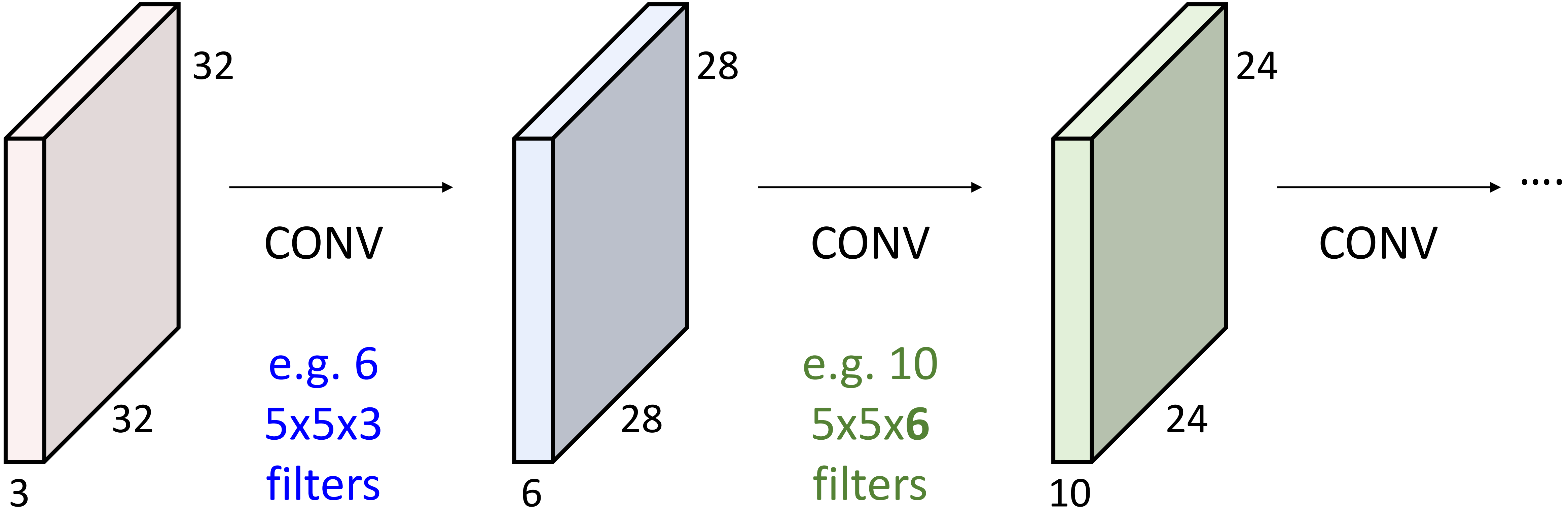
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border
$\Rightarrow$ what is the output?
Hint: (N+2*P-F) / S + 1

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

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

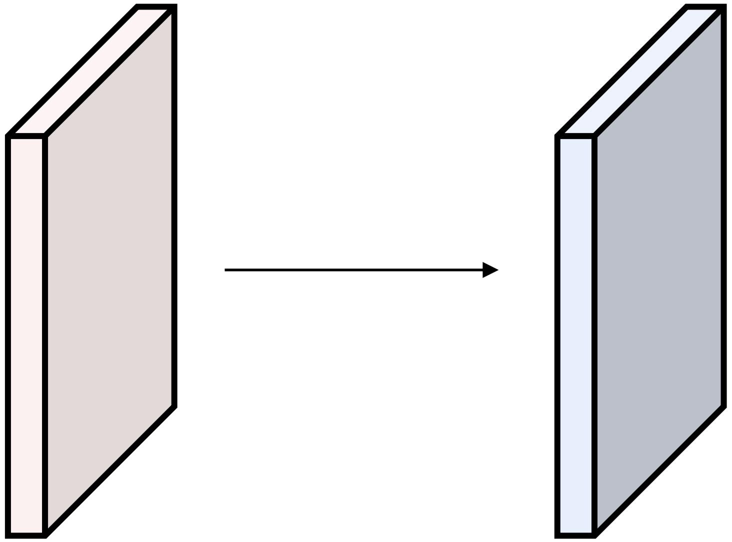


Examples time:

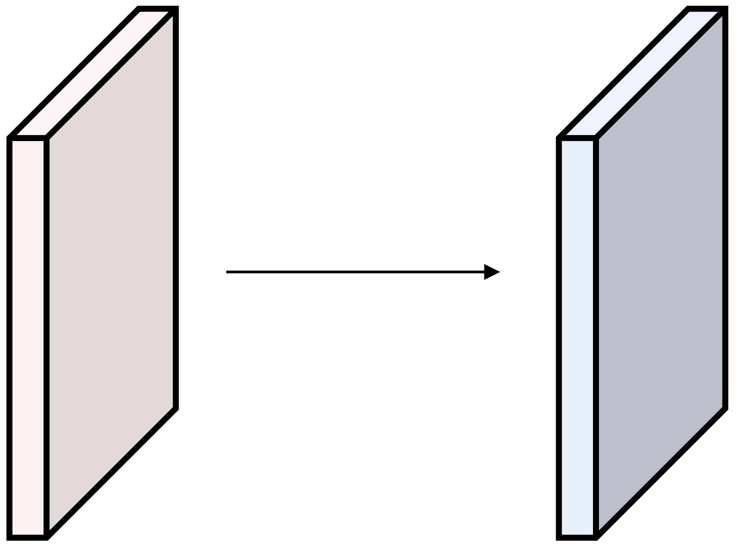
Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples time:



Input volume: **32x32x3**

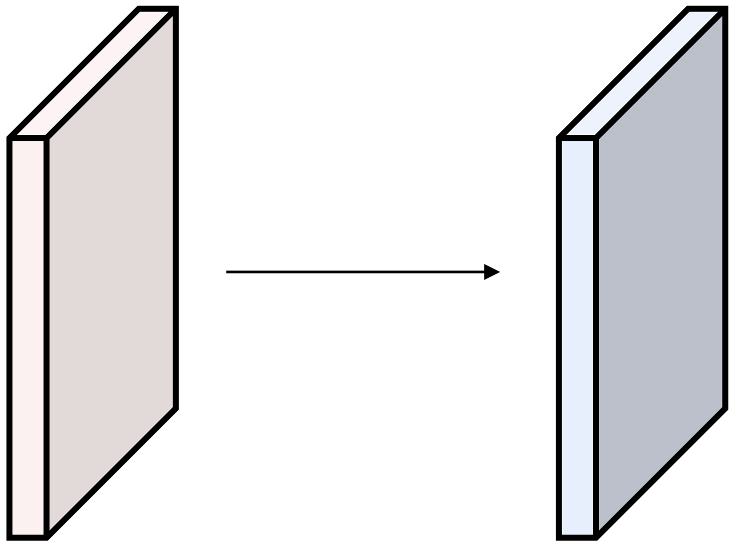
10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

Examples time:



Input volume: **32x32x3**

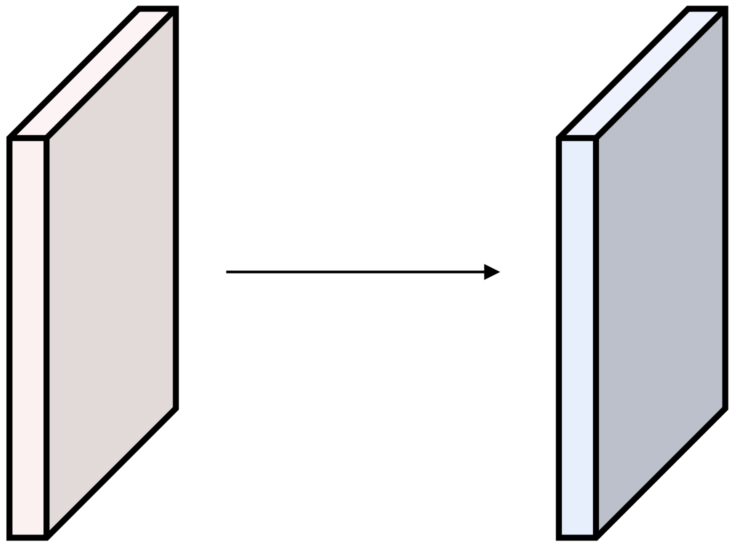
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

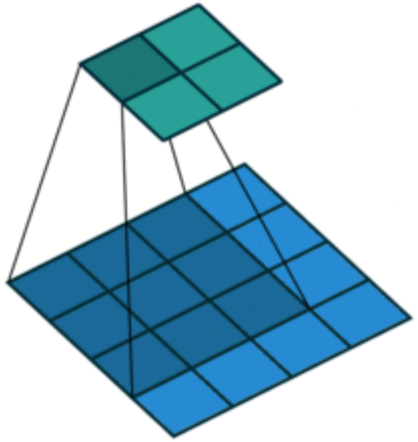


Number of parameters in this layer?

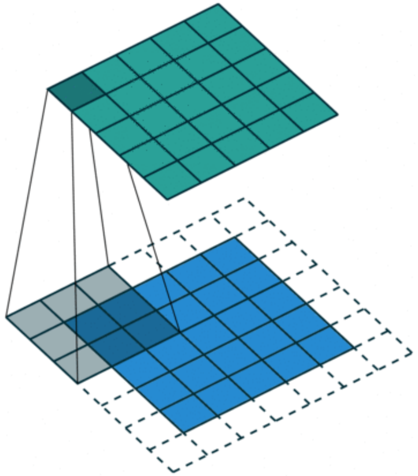
Each filter has 5*5*3 + 1 = 76 params

=> 76*10 = **760**

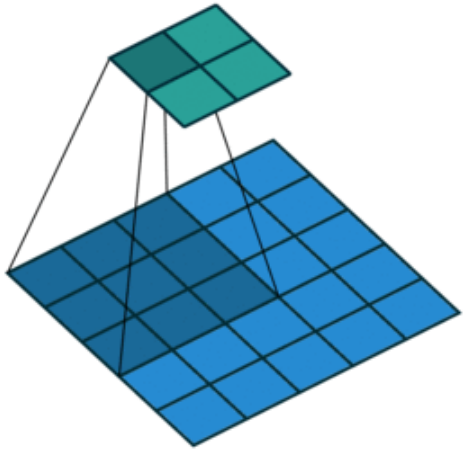
Padding & Stride



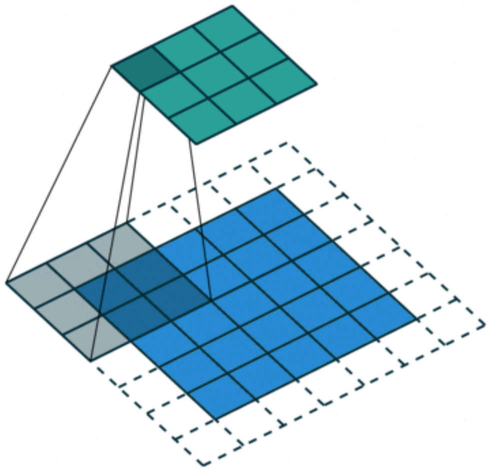
No padding
No stride



Padding
No stride



No padding
Stride

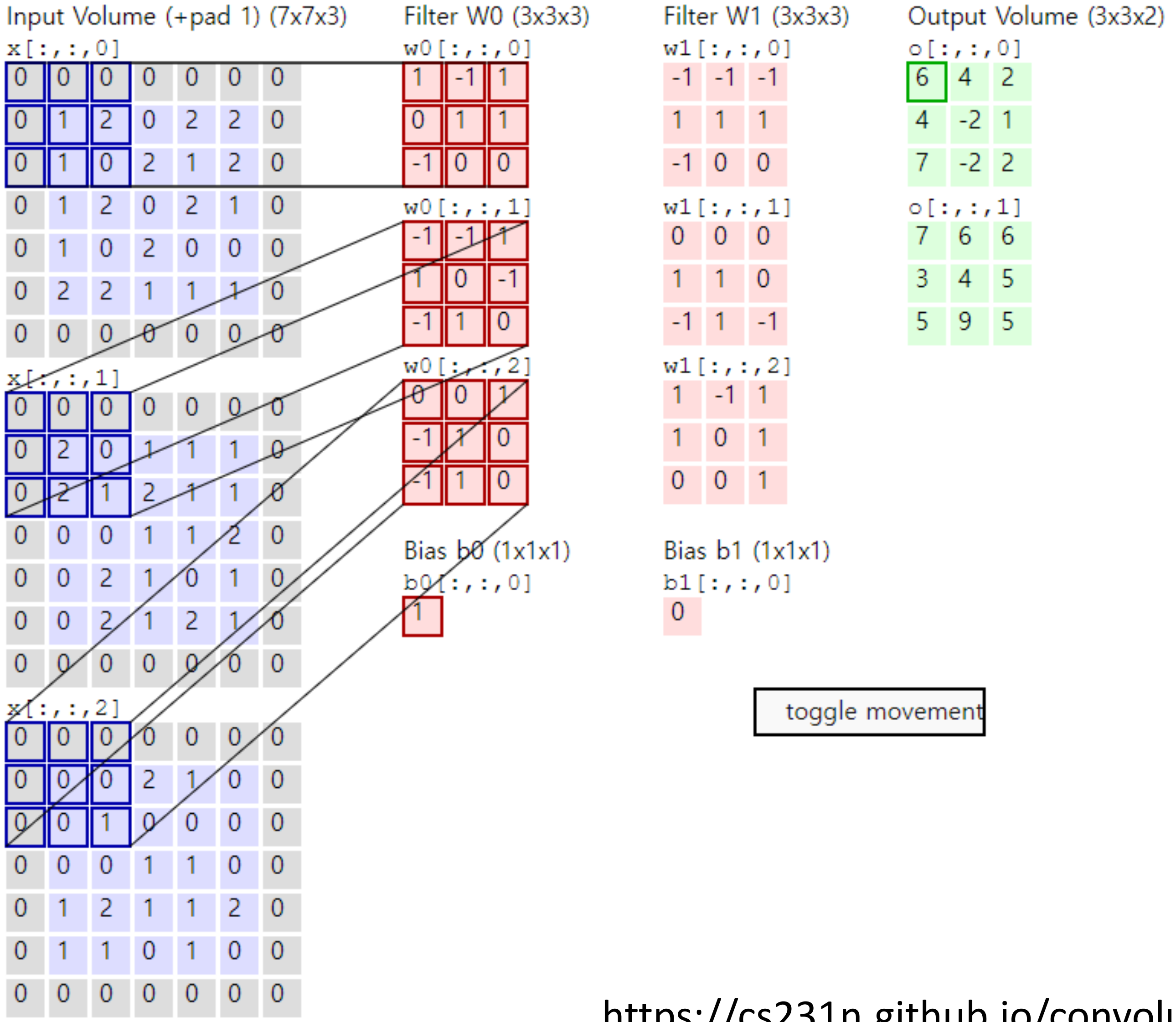


Padding
Stride

https://github.com/vdumoulin/conv_arithmetic

A closer look at spatial dimensions

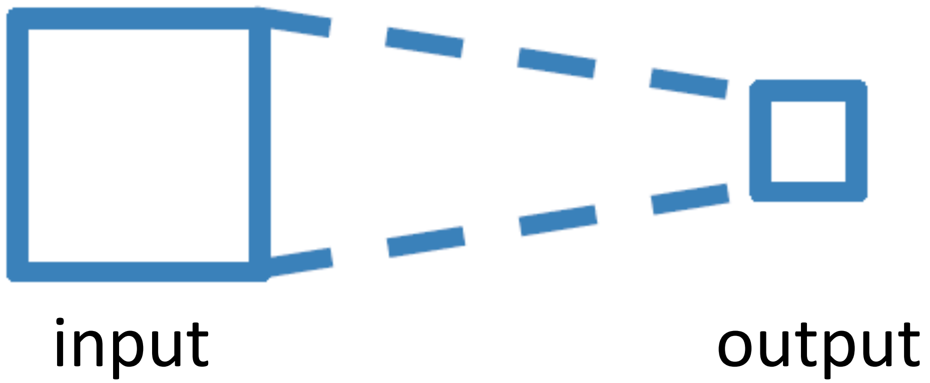
Padding 1
Stride 2



https://cs231n.github.io/convolutional-networks/

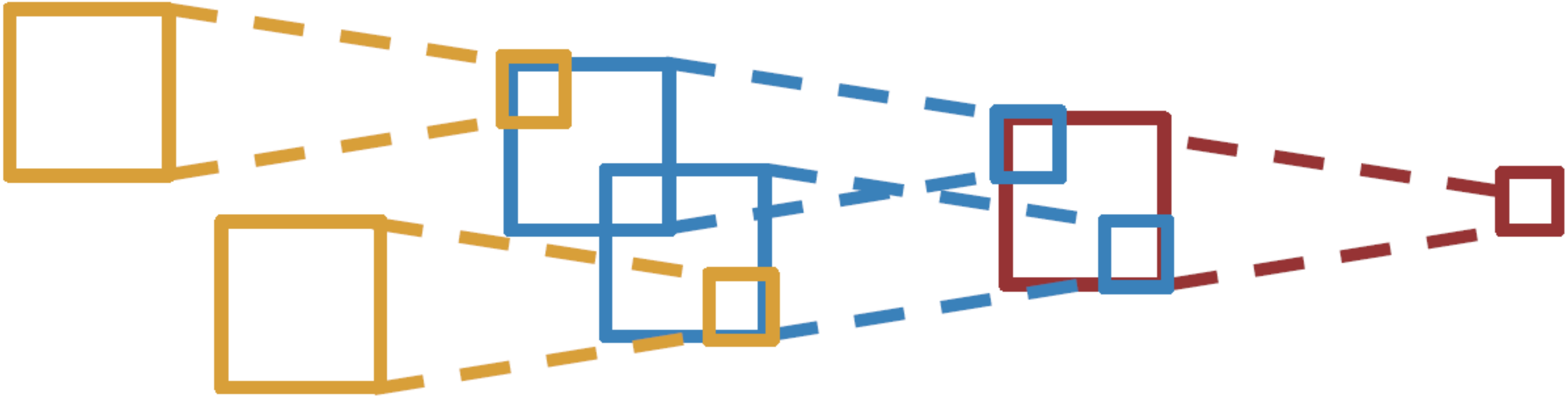
Receptive Fields

- For convolution with kernel size K, each element in the output depends on a **K x K receptive field** in the input



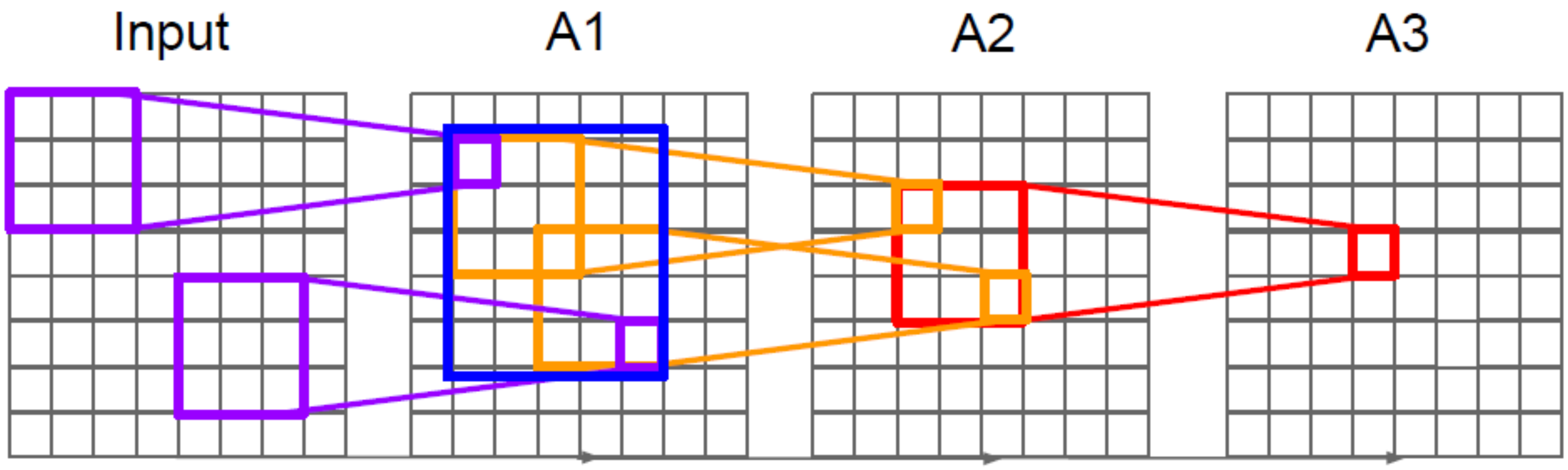
Receptive Fields

- Each successive convolution adds K – 1 to the receptive field size
- With L layers the receptive field size is 1 + L * (K – 1)



Receptive Fields

- Each successive convolution adds K – 1 to the receptive field size
- With L layers the receptive field size is 1 + L * (K – 1)



Problem: For large images we need many layers for each output to "see" the whole image

Solution: Downsample inside the network (Stride)

Convolution layer: summary

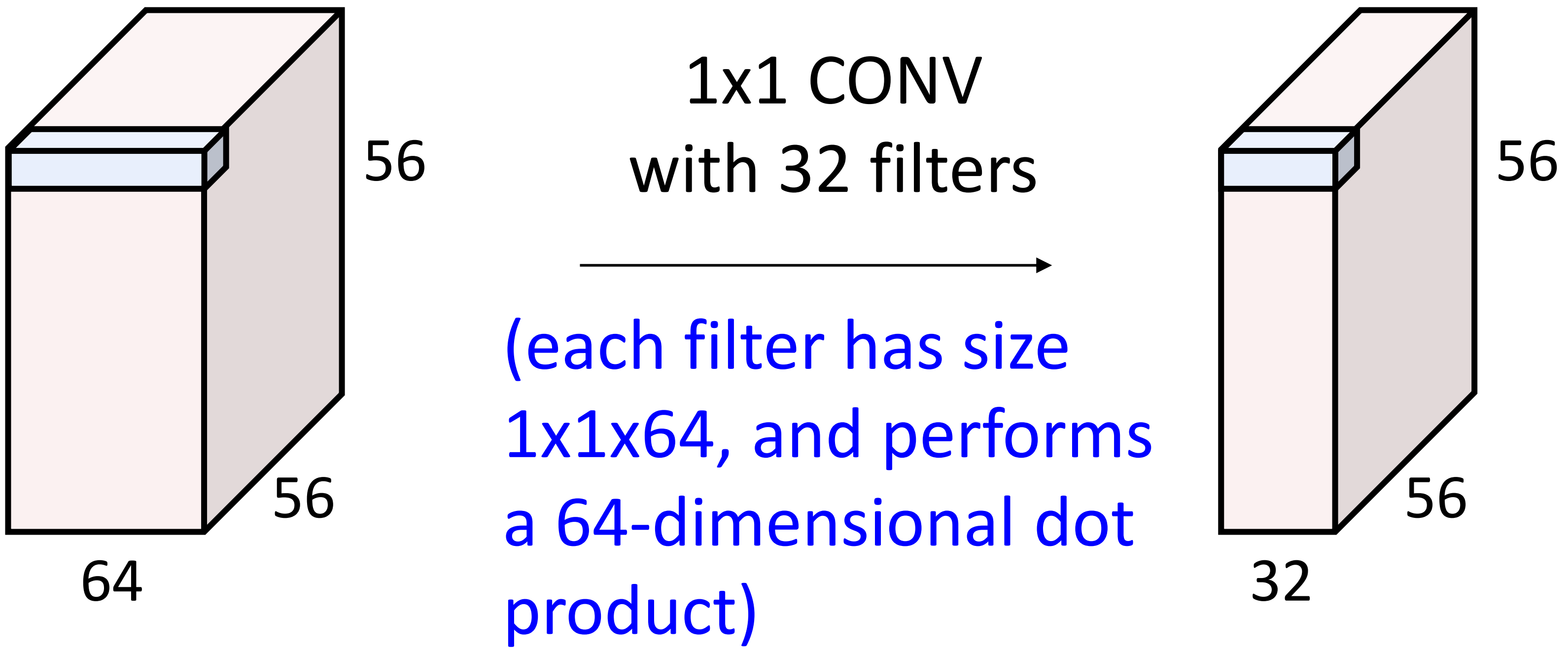
- Let's assume input is W_1 x H_1 x C
- Conv layer needs 4 hyperparameters:
  - - Number of filters **K**
  - - The filter size **F**
  - - The stride **S**
  - - The zero padding **P**
- This will produce an output of W_2 x H_2 x K
- where:
  - - $W_2 = (W1 - F + 2P)/S + 1$
  - - $H_2 = (H1 - F + 2P)/S + 1$
- Number of parameters: F^2CK and K biases

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

1x1 convolution layers make perfect sense



CONV layer in PyTorch

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

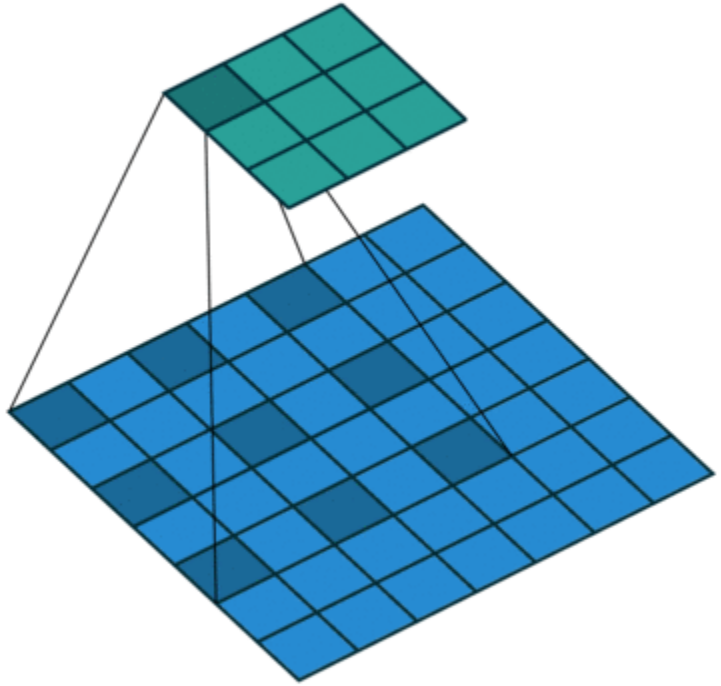
- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At groups=1, all inputs are convolved to all outputs.
  - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At groups= `in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$.

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

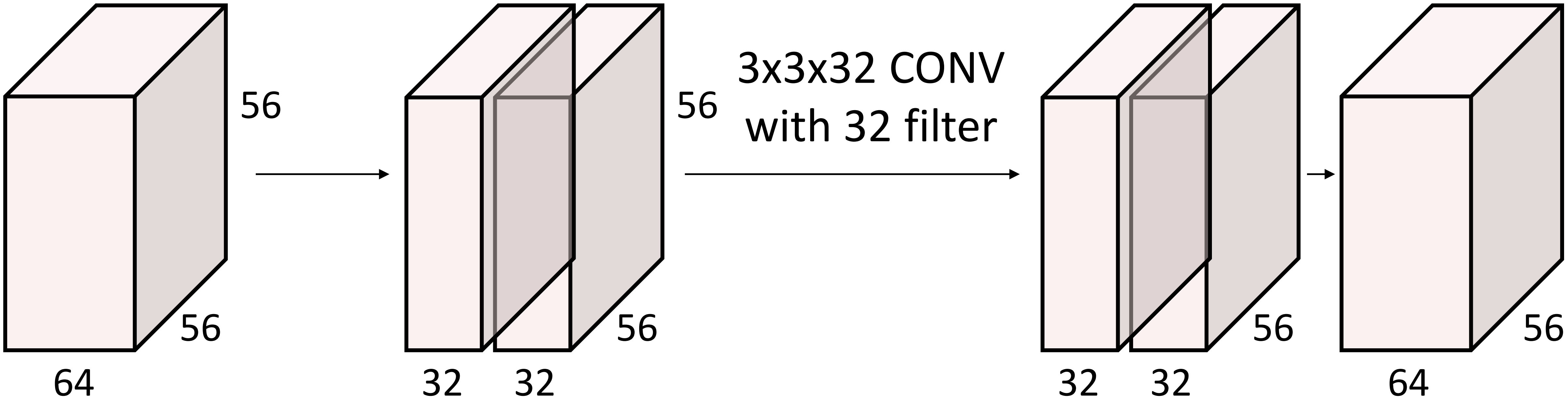
Dilated convolution

- Reduce the parameters of convolution

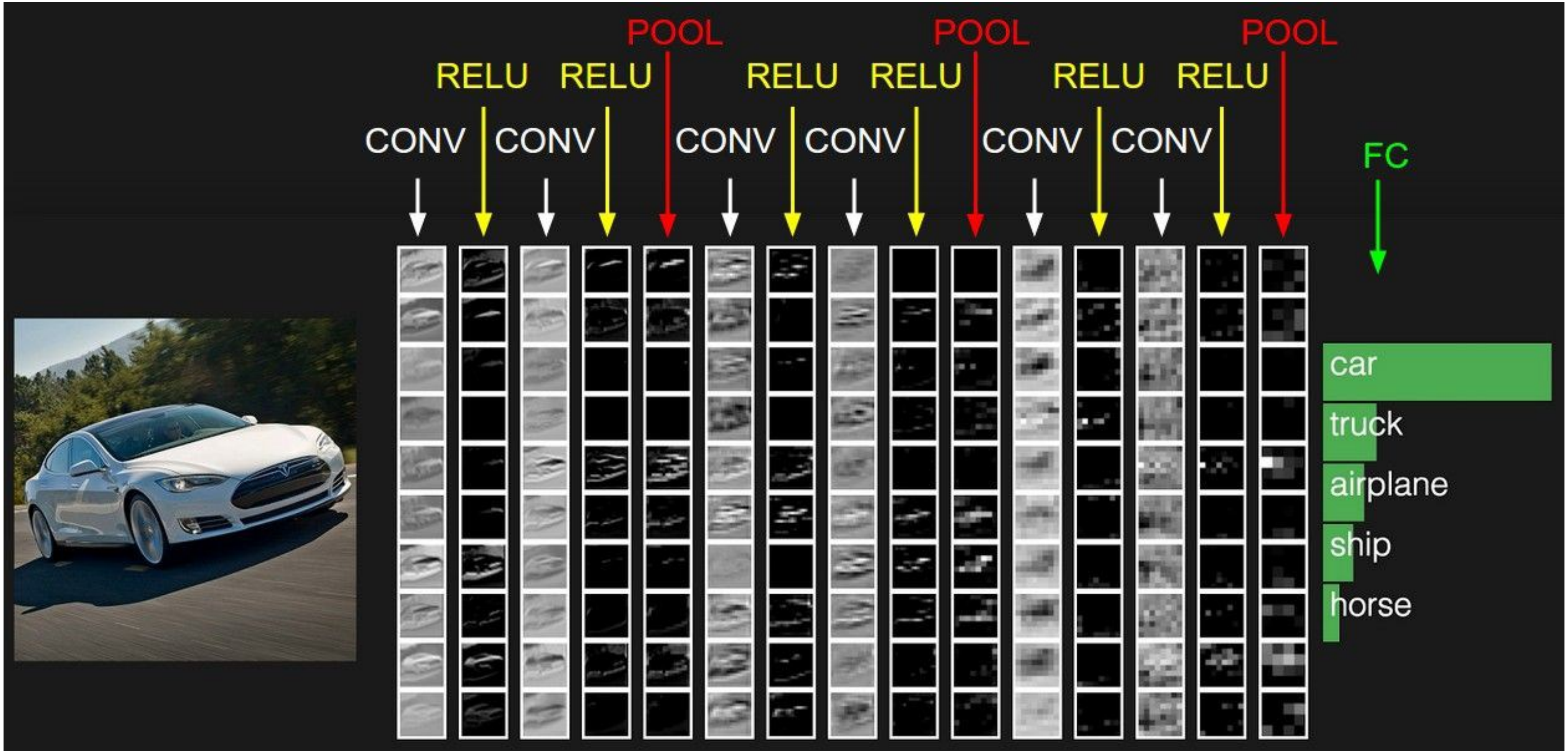


Group convolution

- Reduce the parameters of convolution



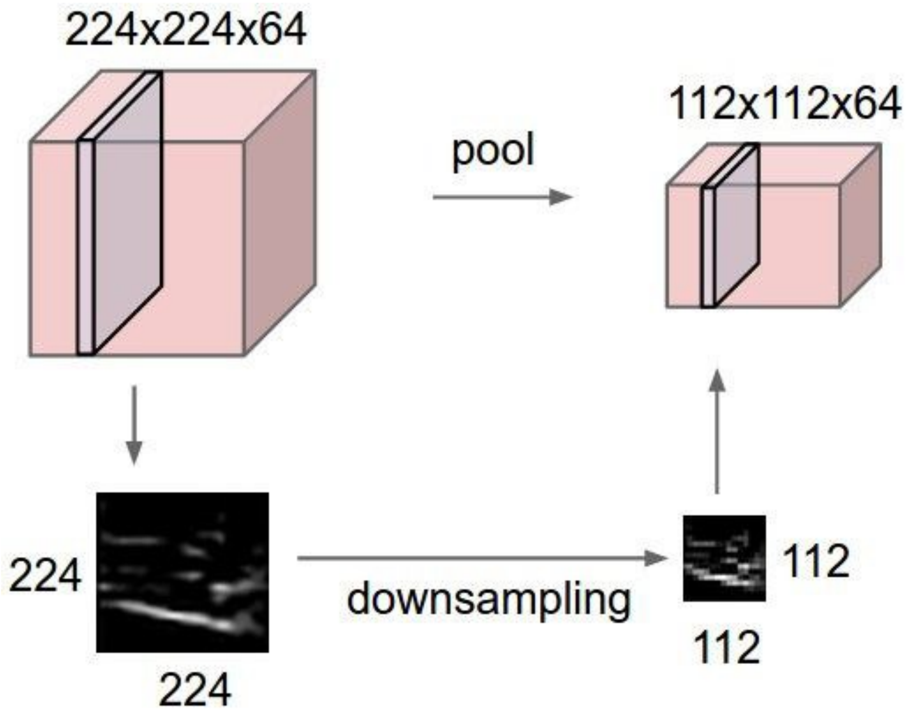
ConvNet



POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently



MAX Pooling

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Max pool with 2x2 filters
and stride 2

6	8
3	4

- No learnable parameters
- Introduces spatial invariance

Pooling layer: summary

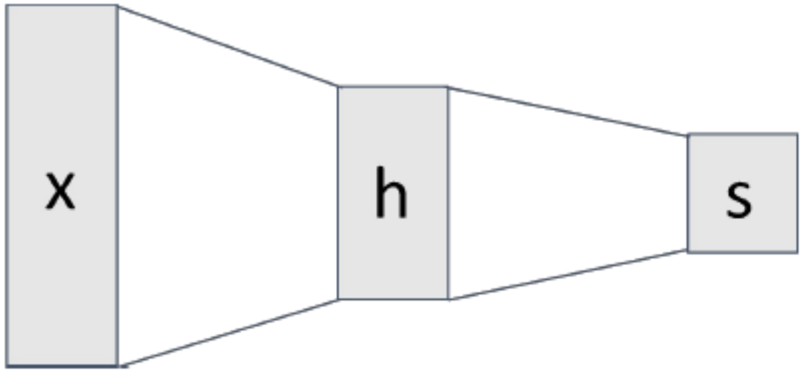
- Let's assume input is W_1 x H_1 x C
- Conv layer needs 2 hyperparameters:
  - The spatial extent **F**
  - The stride **S**

- This will produce an output of W_2 x H_2 x C where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$

- Number of parameters: 0

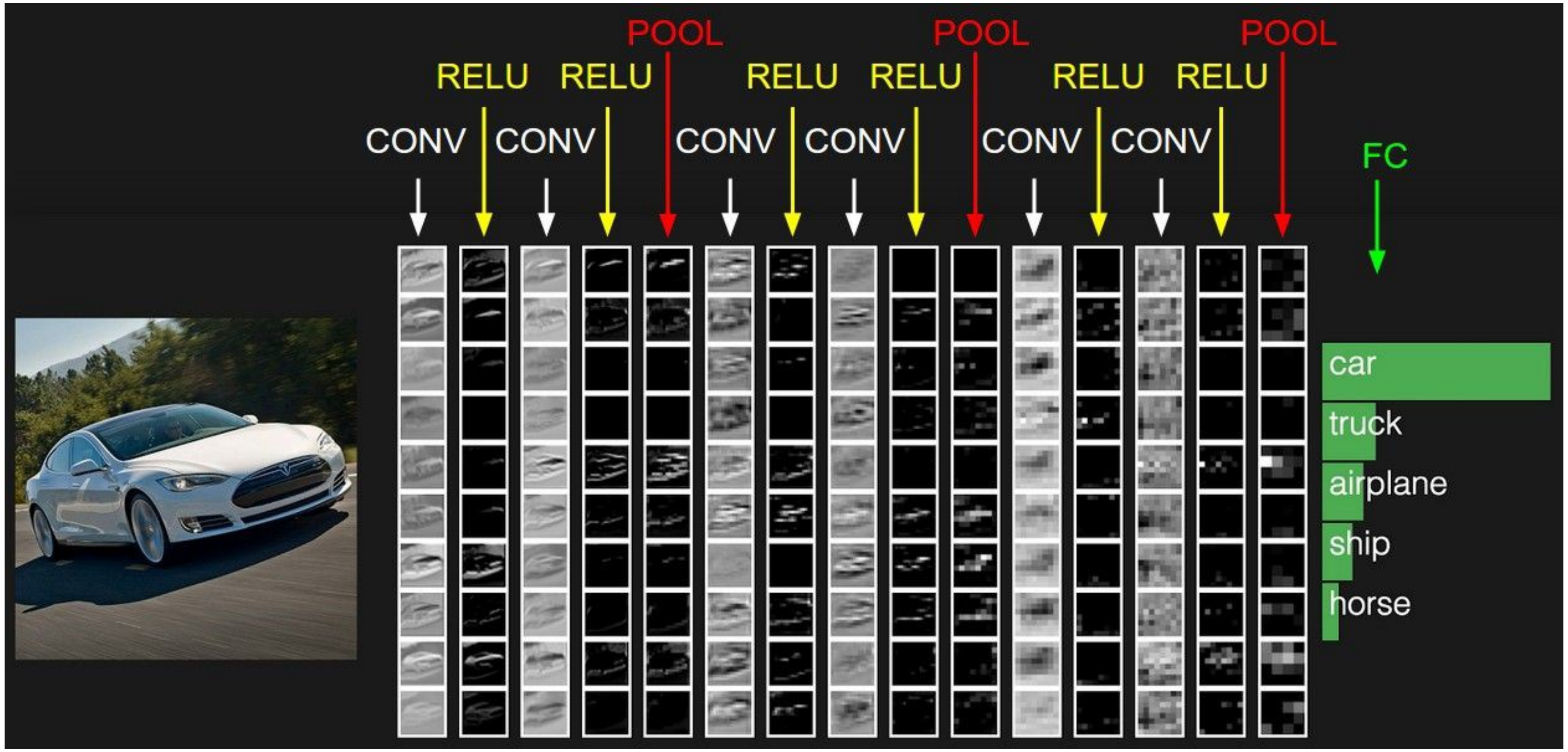
Fully Connected Layer (FC layer)

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



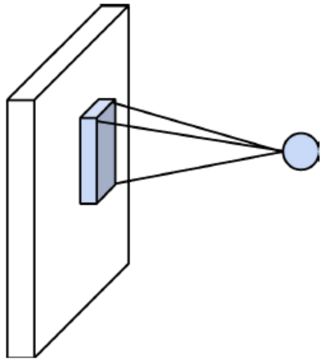
Fully Connected Layer (FC layer)

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

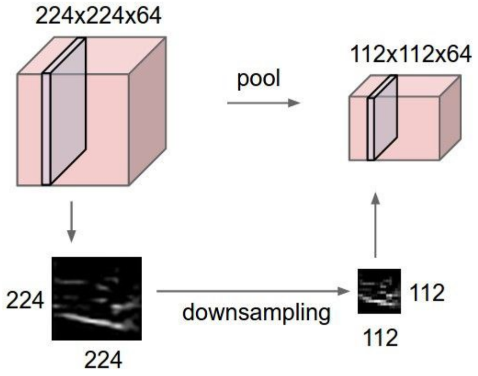


Components of CNNs

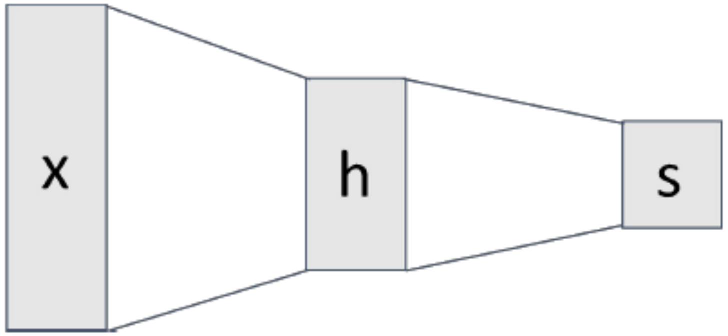
Convolution Layers



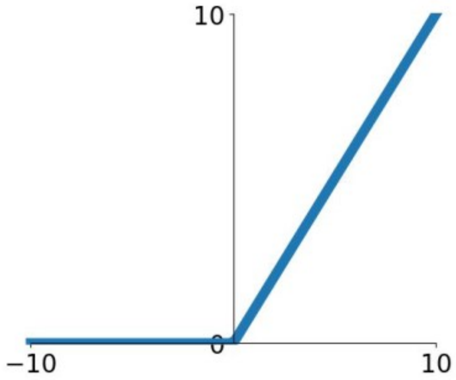
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Batch Normalization

- Consider a single layer $y = wx + b$

- The following could lead to tough optimization:
  - Inputs x are not centered around zero (need large bias)
  - Inputs x have different scaling per-element
    (entries in W will need to vary a lot)

- Idea: force inputs to be “nicely scaled” at each layer!

Batch Normalization

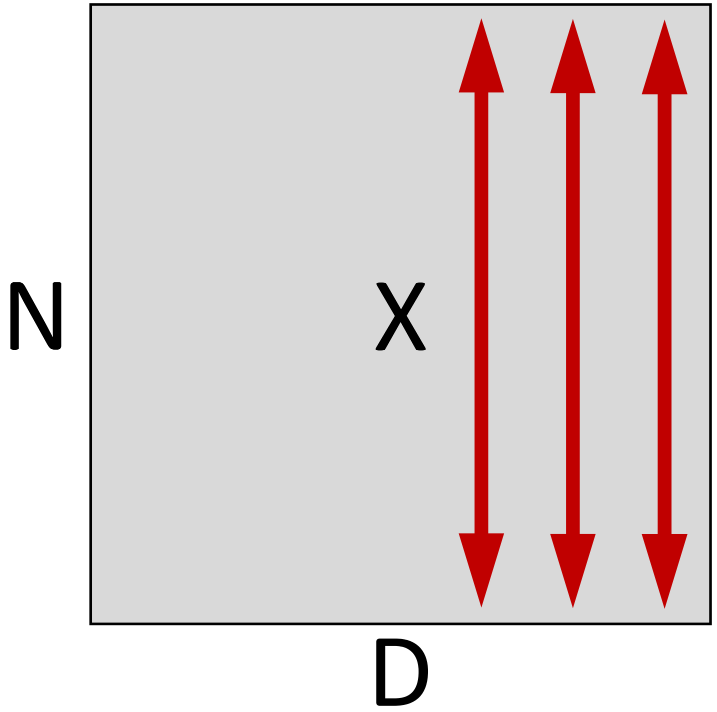
- Consider a batch of activations at some layer. To make each dimension **zero-mean unit-variance**, apply:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Ioffe and Szegedy, 2015

Batch Normalization

Input: $x: N \times D$



$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, shape is N x D

Problem: What if zero-mean, unit variance is too hard of a constraint?

Batch Normalization

Input: x: $N \times D$

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta : D$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, shape is N x D

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output shape is N x D

Batch Normalization

Estimates depend on minibatch; can't do this at test-time!

Input: x: $N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output shape is N x D

Batch Normalization

Input: x: $N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

During testing batchnorm becomes a linear operator! Can be fused with the previous fully-connected or conv layer

$\mu_j =$ (Running) average of values seen during training

Per-channel mean, shape is D

$\sigma_j^2 =$ (Running) average of values seen during training

Per-channel var, shape is D

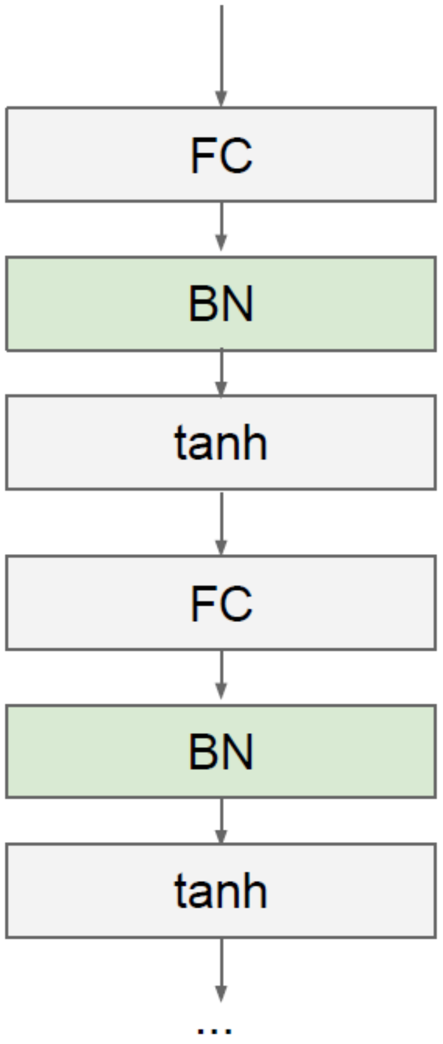
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output shape is N x D

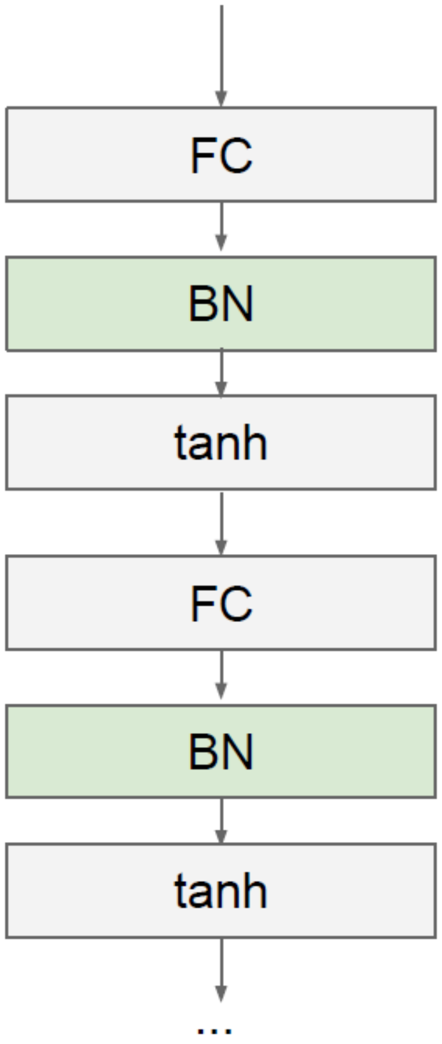
Batch Normalization



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization



Makes deep networks much easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Batch Normalization for ConvNets

Batch Normalization for **fully-connected** networks

$$\boldsymbol{x}: \boldsymbol{N} \times \boldsymbol{D}$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \mathbf{1} \times \boldsymbol{D}$$
$$\boldsymbol{\gamma}, \boldsymbol{\beta}: \mathbf{1} \times \boldsymbol{D}$$
$$\boldsymbol{y} = \boldsymbol{\gamma}(\boldsymbol{x} - \boldsymbol{\mu})/\boldsymbol{\sigma} + \boldsymbol{\beta}$$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$$\boldsymbol{x}: \boldsymbol{N} \times \boldsymbol{C} \times \boldsymbol{H} \times \boldsymbol{W}$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \mathbf{1} \times \boldsymbol{C} \times \mathbf{1} \times \mathbf{1}$$
$$\boldsymbol{\gamma}, \boldsymbol{\beta}: \mathbf{1} \times \boldsymbol{C} \times \mathbf{1} \times \mathbf{1}$$
$$\boldsymbol{y} = \boldsymbol{\gamma}(\boldsymbol{x} - \boldsymbol{\mu})/\boldsymbol{\sigma} + \boldsymbol{\beta}$$

Layer Normalization

Batch Normalization for
fully-connected networks

$$x: N \times D$$

Normalize

$$\mu, \sigma: 1 \times D$$
$$\gamma, \beta: 1 \times D$$
$$y = \gamma(x - \mu)/\sigma + \beta$$

Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$$x: N \times D$$

Normalize

$$\mu, \sigma: N \times 1$$
$$\gamma, \beta: N \times 1$$
$$y = \gamma(x - \mu)/\sigma + \beta$$

Instance Normalization

Batch Normalization for **fully-connected** networks

$$x: N \times D$$

Normalize

$$\mu, \sigma: 1 \times D$$
$$\gamma, \beta: 1 \times D$$
$$y = \gamma(x - \mu)/\sigma + \beta$$

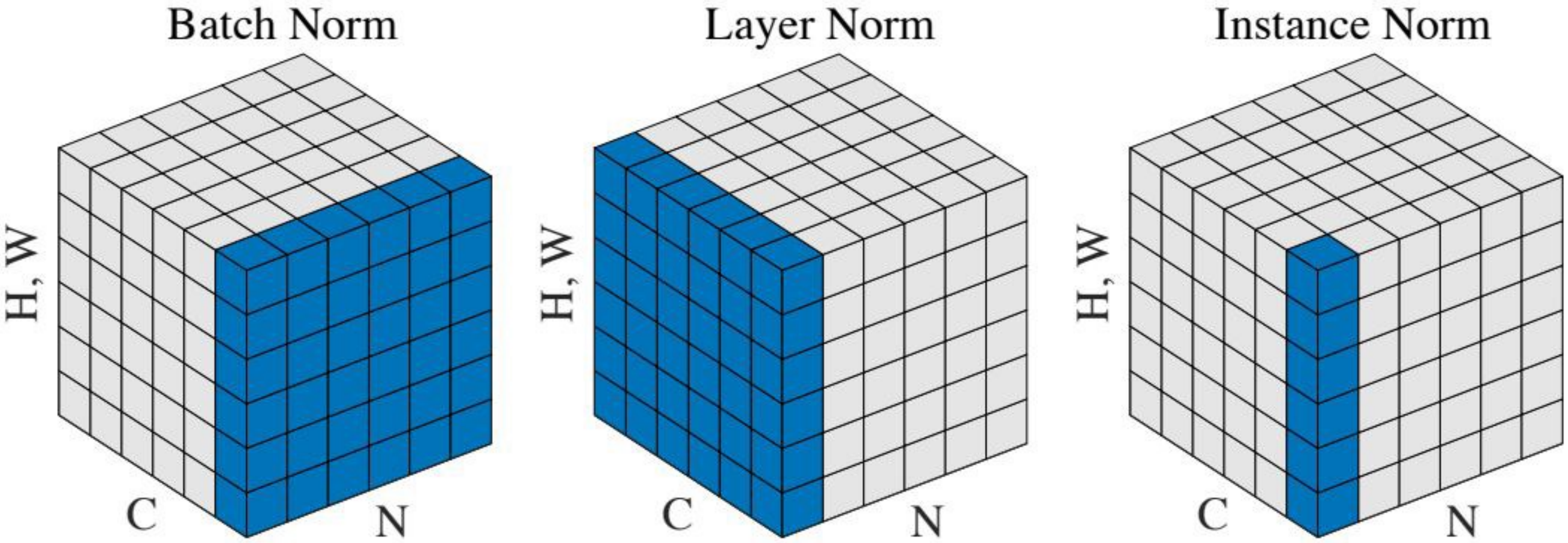
Instance Normalization for convolutional networks
Same behavior at train / test!

$$x: N \times C \times H \times W$$

Normalize

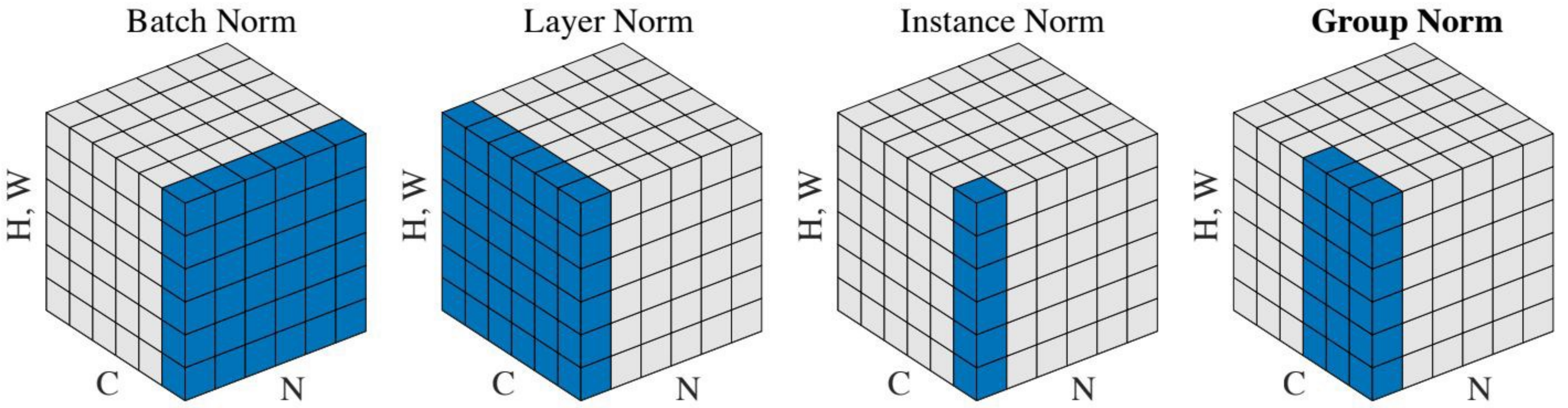
$$\mu, \sigma: N \times C \times 1 \times 1$$
$$\gamma, \beta: N \times C \times 1 \times 1$$
$$y = \gamma(x - \mu)/\sigma + \beta$$

Comparison of Normalization Layers



Wu and He, “Group Normalization”, ECCV 2018

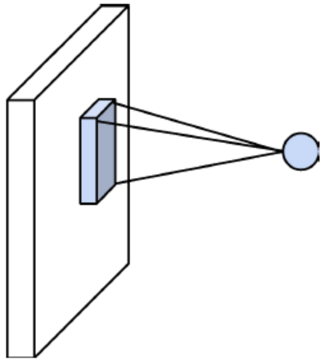
Comparison of Normalization Layers



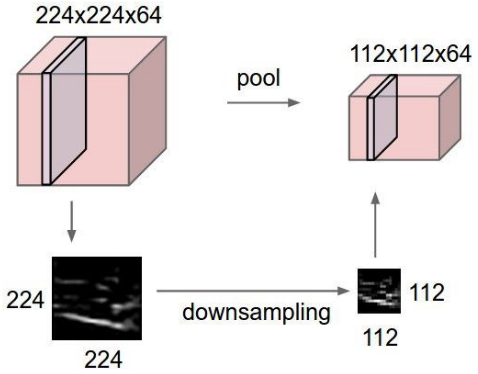
Wu and He, “Group Normalization”, ECCV 2018

Components of CNNs

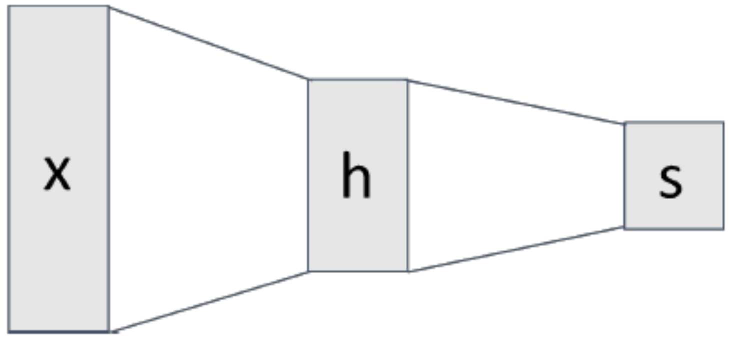
Convolution Layers



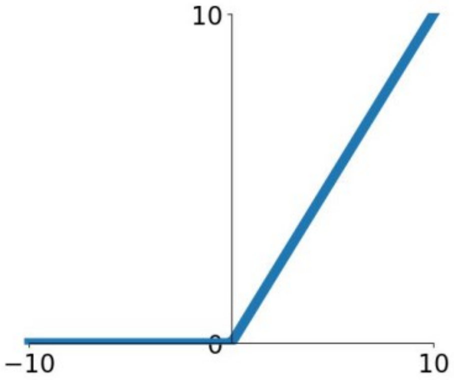
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Question: How should we put them together?

Next time: CNN Architectures