Lecture 7:
Convolutional Networks

Reminder: A2

Due Monday, September 30, 11:59pm (Even if you enrolled late!)

Your submission must pass the validation script

Slight schedule change

Content originally planned for today got split into two lectures
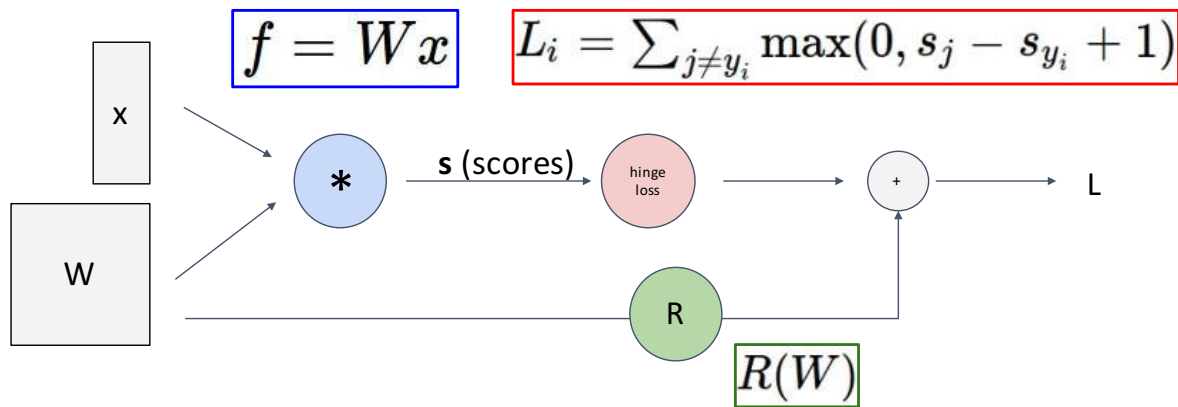
Pushes the schedule back a bit:

A4 Due Date: Friday 11/1 -> Friday 11/8
A5 Due Date: Friday 11/15 -> Friday 11/22
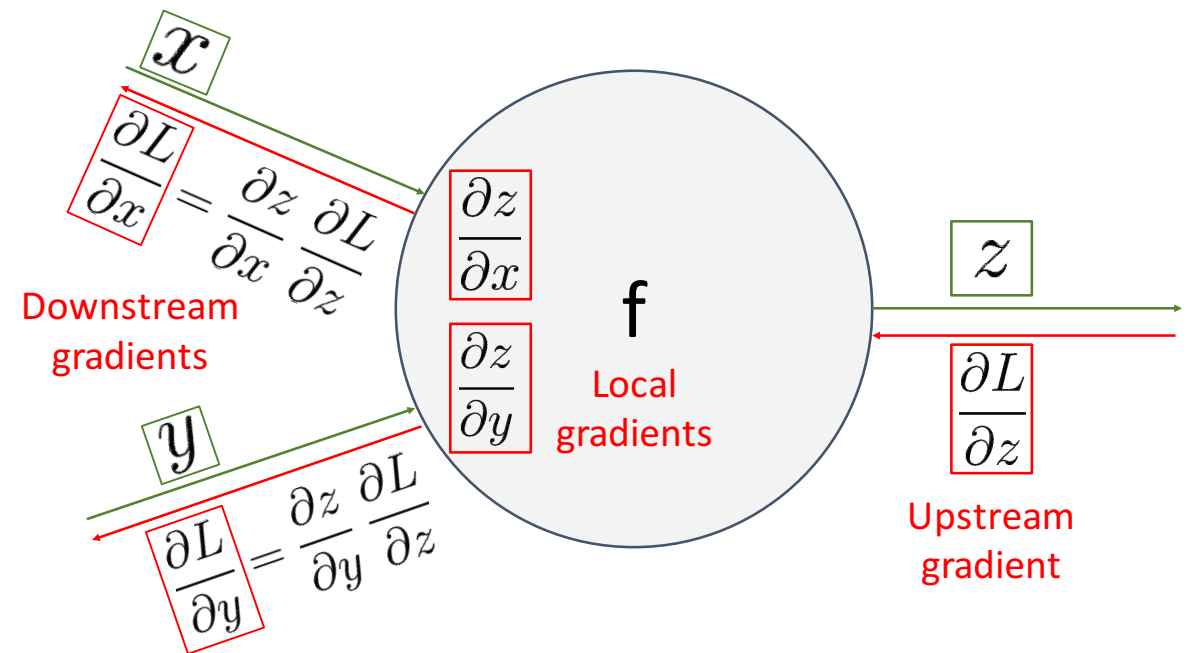A6 Due Date: Still Friday 12/6

Last Time: Backpropagation

Represent complex expressions as **computational graphs**

$$f = Wx \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

x

W

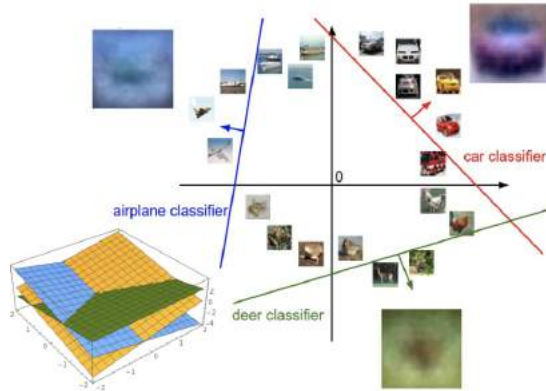* → **s** (scores) → hinge loss → + → L

R

$R(W)$

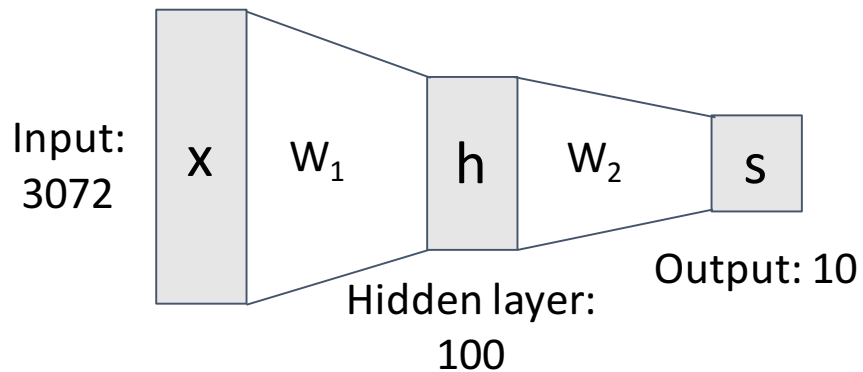Forward pass computes outputs

Backward pass computes gradients

During the backward pass, each node in the graph receives **upstream gradients** and multiplies them by **local gradients** to compute **downstream gradients**
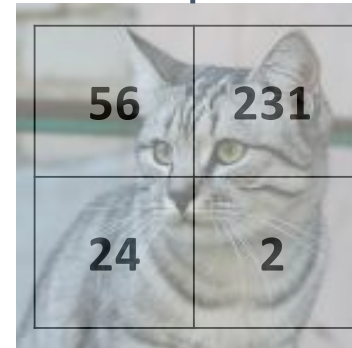
x

$$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z}$$

Downstream gradients

$\frac{\partial z}{\partial x}$

$\frac{\partial z}{\partial y}$

f

Local gradients

z

$\frac{\partial L}{\partial z}$

Upstream gradient

y

$$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y} \frac{\partial L}{\partial z}$$

f(x,W) = Wx

$$f = W_2 \max(0, W_1 x)$$

Input: 3072

x

W_1

h

W_2

s

Hidden layer: 100

Output: 10
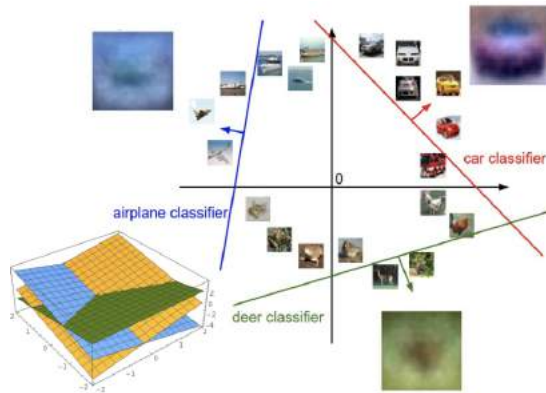
Stretch pixels into column

Input image (2, 2)

| 56 |
| 231 |
| 24 |
| 2 |

(4,)

Problem: So far our classifiers don't respect the spatial structure of images!

$f(x,W) = Wx$

$$f = W_2 \max(0, W_1 x)$$

Input: 3072

x W_1 h W_2 s

Hidden layer: 100

Output: 10

Stretch pixels into column

Input image (2, 2)

56
231
24
2

(4,)
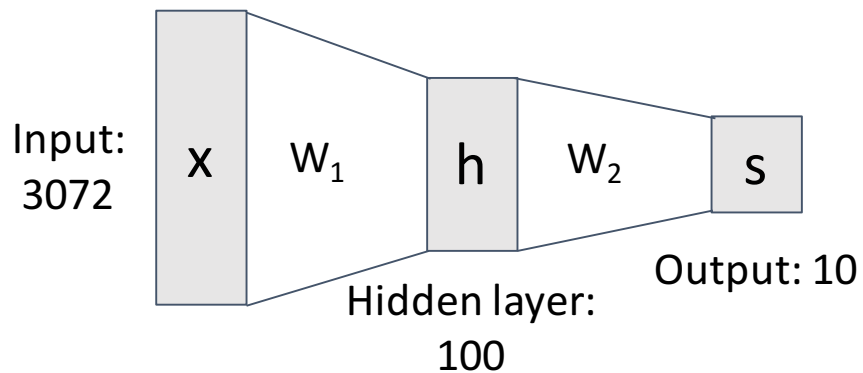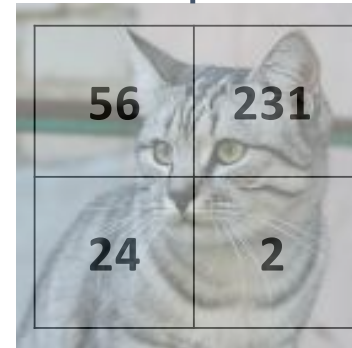
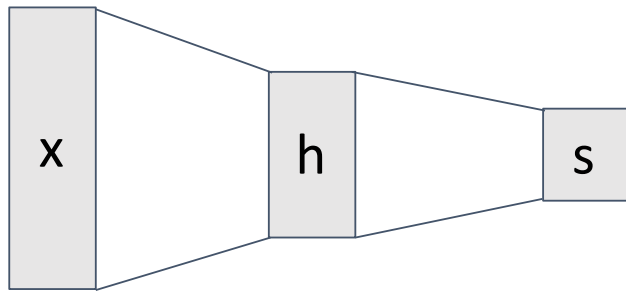**Problem**: So far our classifiers don't respect the spatial structure of images!

Solution: Define new computational nodes that operate on images!

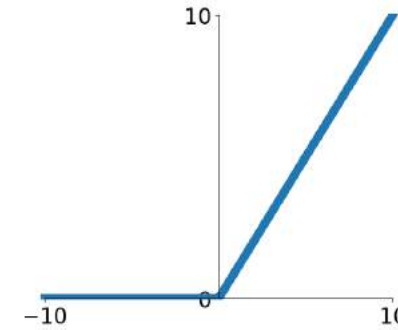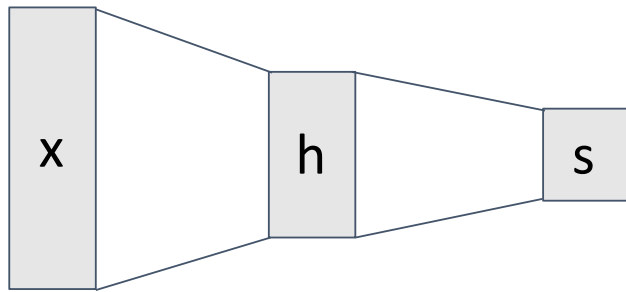# Components of a Full-Connected Network
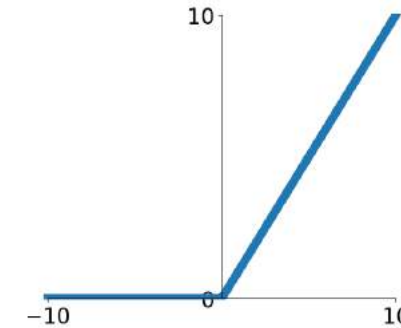
Fully-Connected Layers

Activation Function

Components of a Convolutional Network

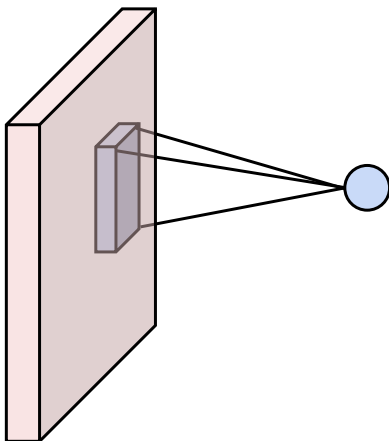## Fully-Connected Layers

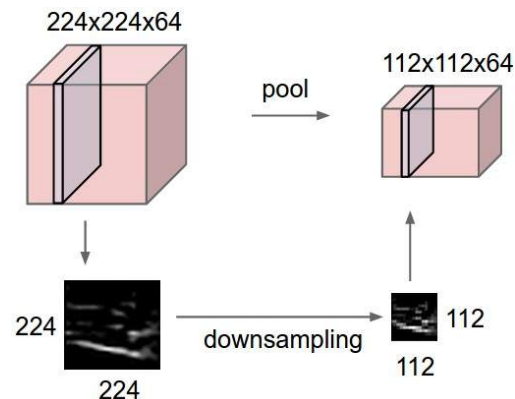x   h   s

Activation Function

Convolution Layers

Pooling Layers

224x224x64

pool

112x112x64

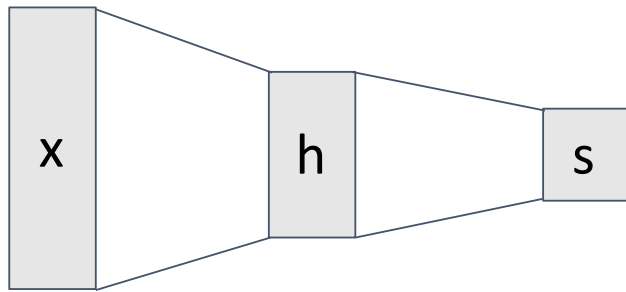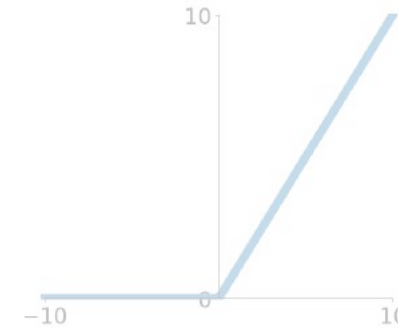224

downsampling

112

224

112

Normalization

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

Components of a Convolutional Network

Fully-Connected Layers

x   h   s

Activation Function

10

−10   0   10

Convolution Layers

Pooling Layers

224x224x64

pool

112x112x64

224

downsampling

112

224

112
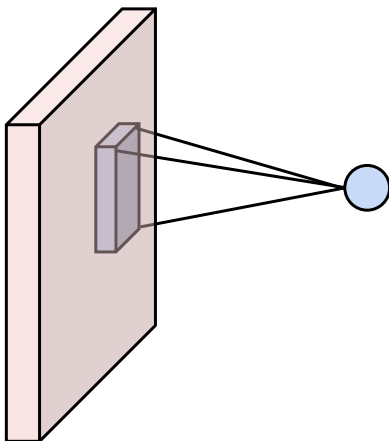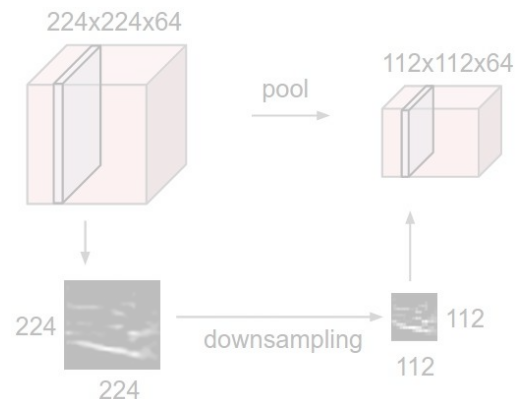
Normalization

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

Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1

Input

1

3072

$$Wx$$

10 x 3072
weights

Output

1

10

Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1

Input

1

3072

Wx

10 x 3072
weights

Output

1

10

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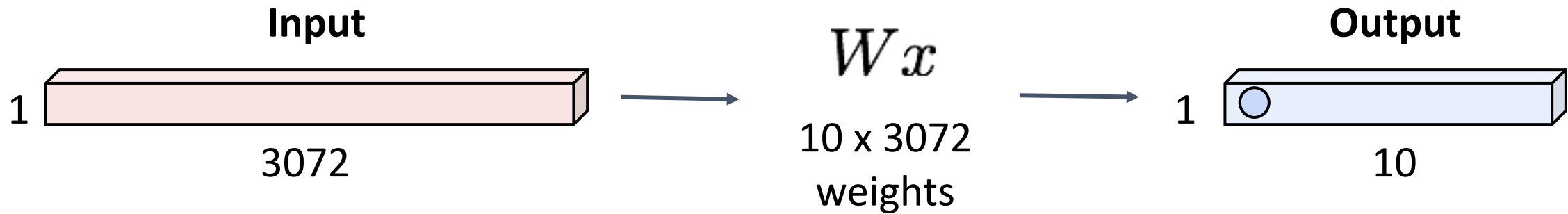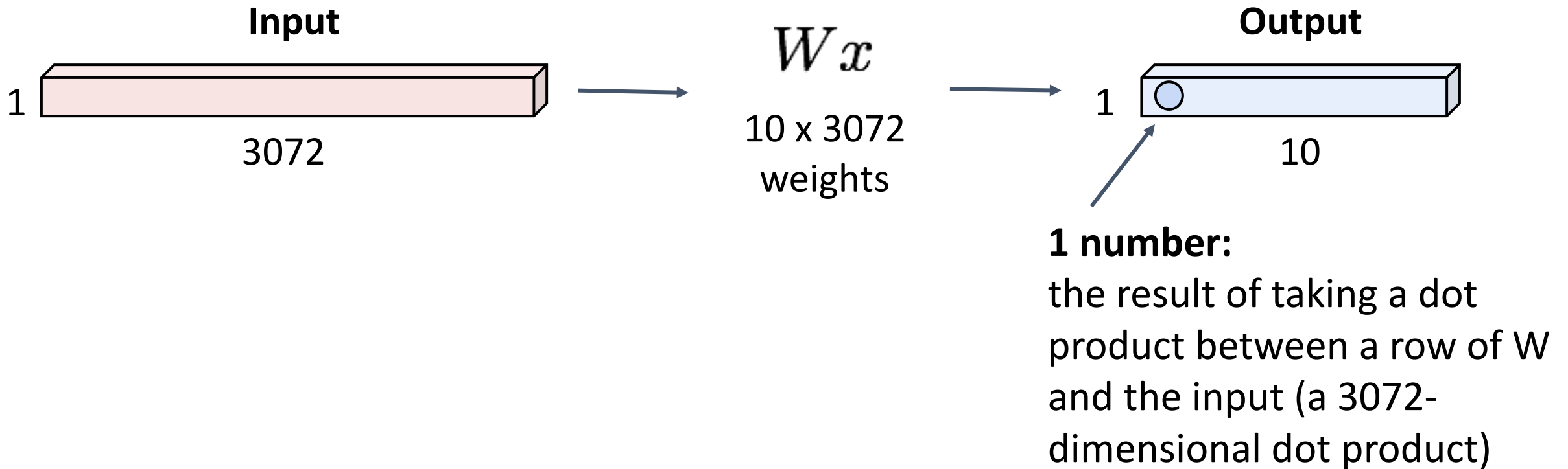
Convolution Layer

3x**32**x**32** image: preserve spatial structure
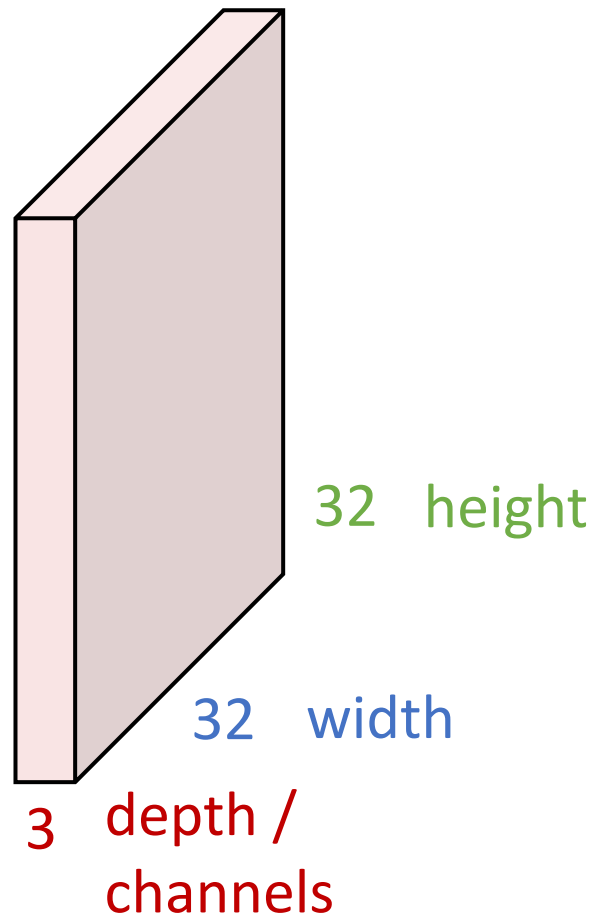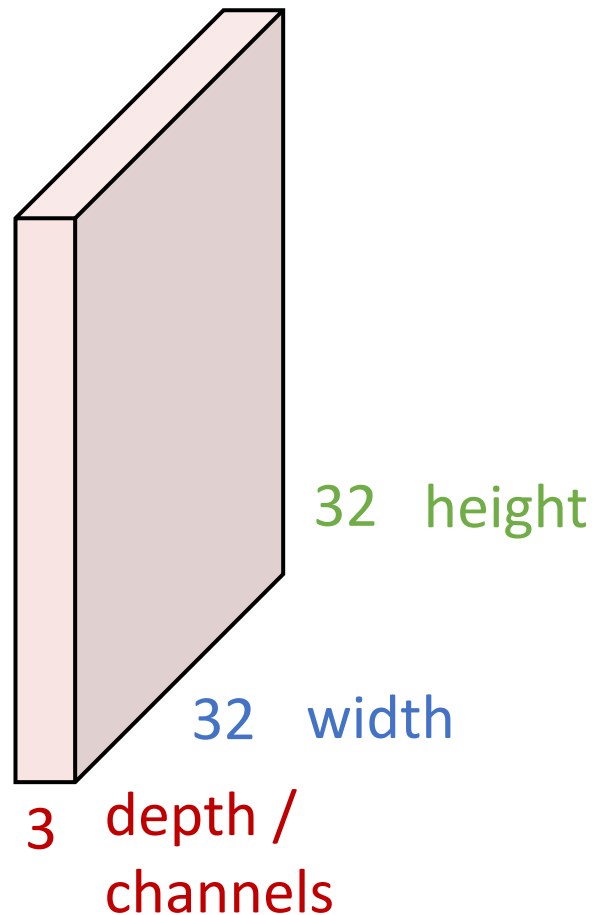
32   height

32   width

3   depth / channels

Convolution Layer

3x**32**x**32** image

3x5x5 filter

Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

32   height

32   width

3   depth / channels

Convolution Layer
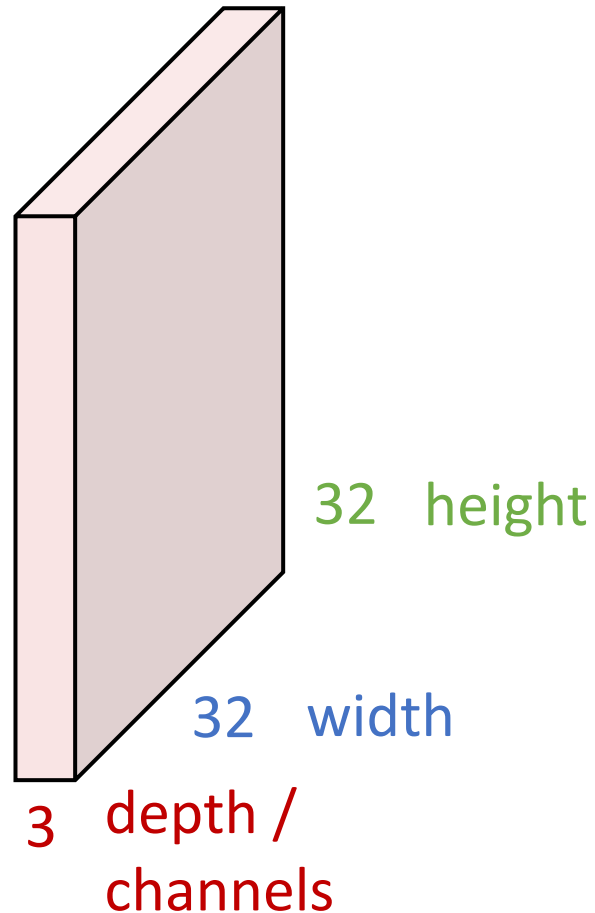
3x**32**x**32** image

Filters always extend the full depth of the input volume

3x5x5 filter

Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

32 height

32 width

3 depth / channels
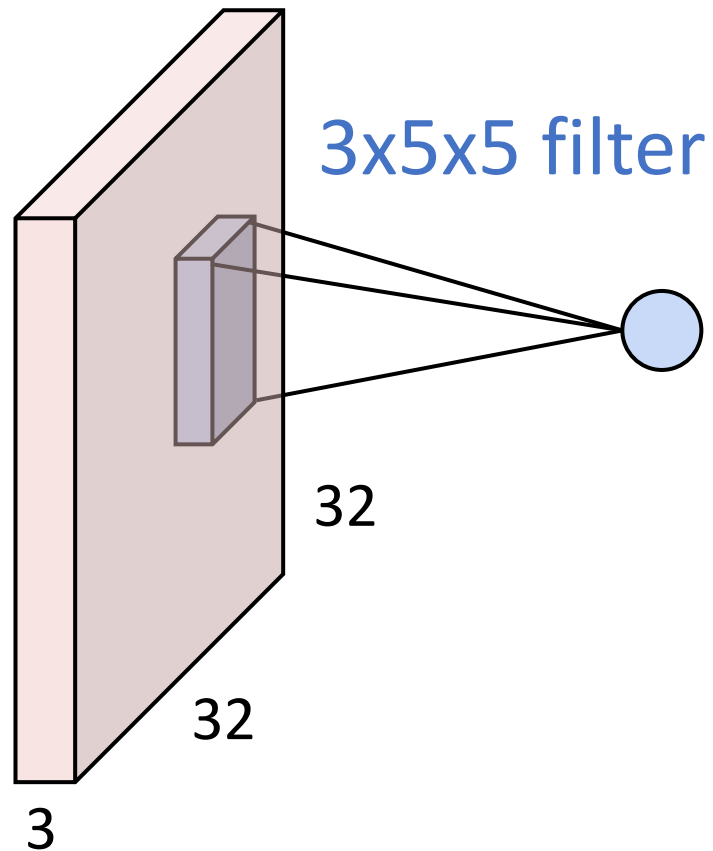
Convolution Layer

3x32x32 image

3x5x5 filter

32

32

3
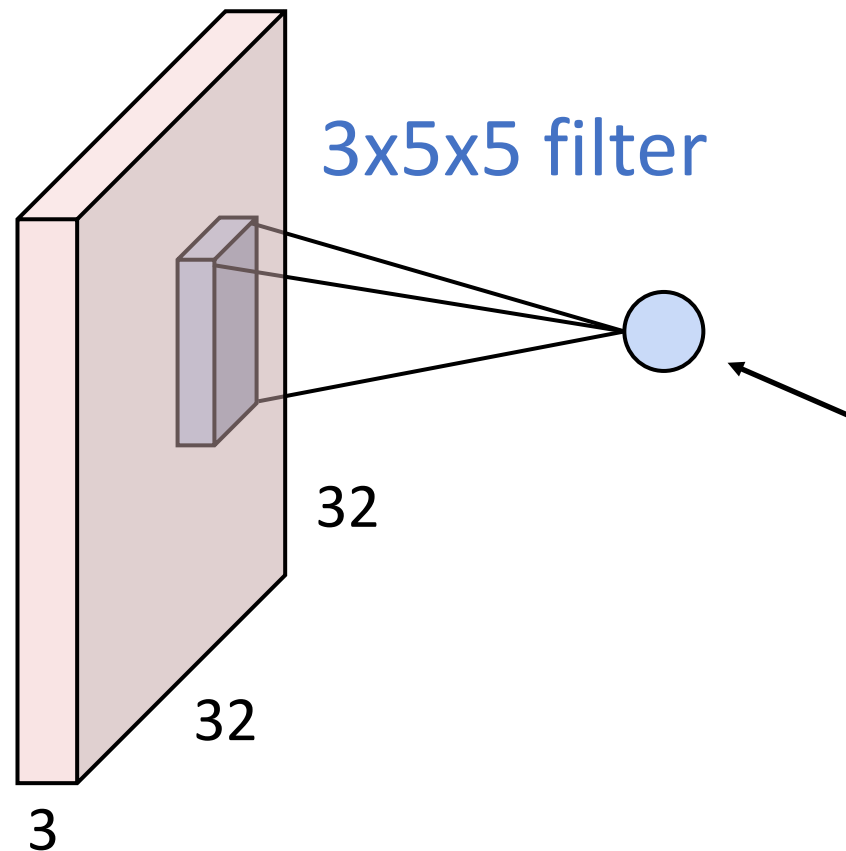
1 number:
the result of taking a dot product between the filter
and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

3x32x32 image

3x5x5 filter

32

32

3

1x28x28
activation map

28

28

1

convolve (slide) over
all spatial locations
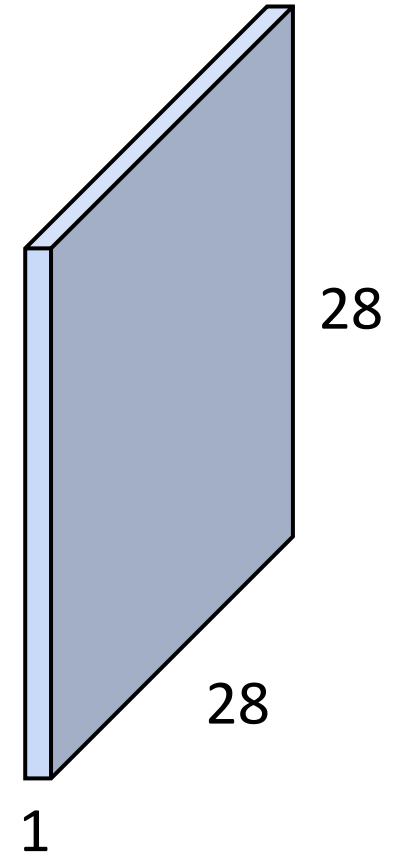
Convolution Layer

3x32x32 image

3x5x5 filter

Consider repeating with a second (green) filter:

two 1x28x28 activation map

convolve (slide) over all spatial locations

32

32

3

28

28

1 1
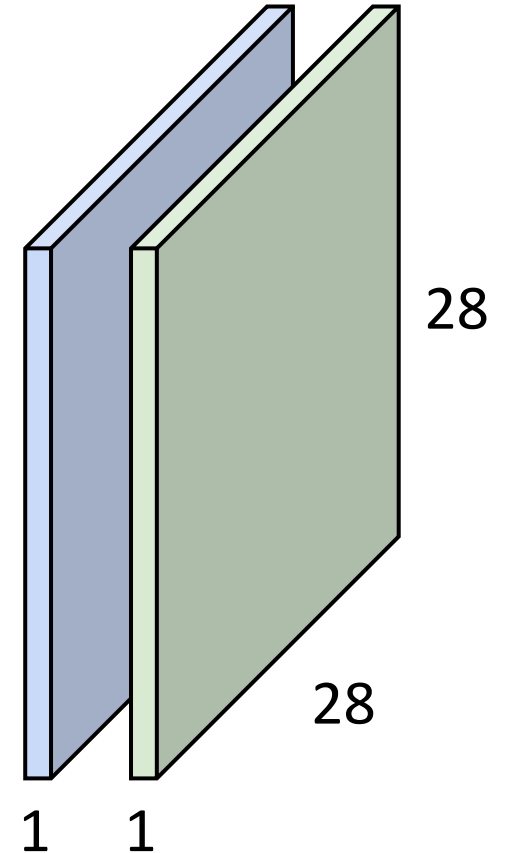
Convolution Layer
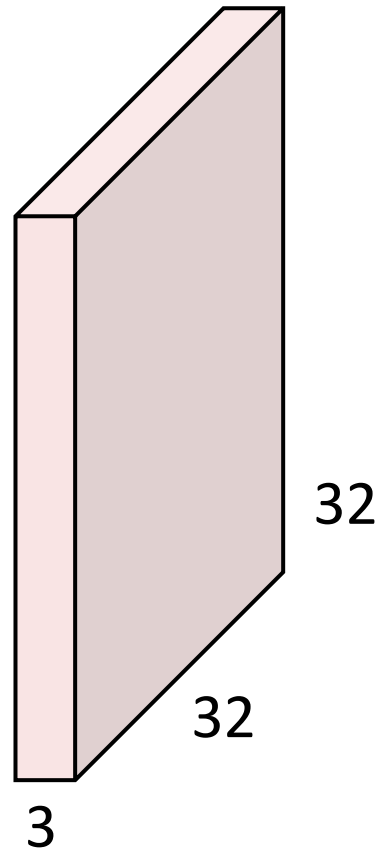
3x32x32 image

Consider 6 filters, each 3x5x5

6 activation maps, each 1x28x28
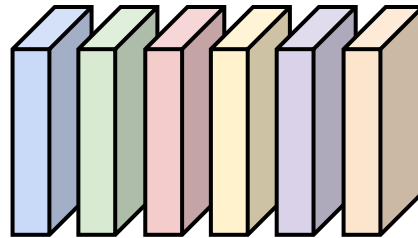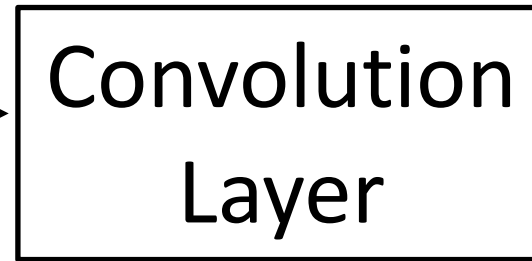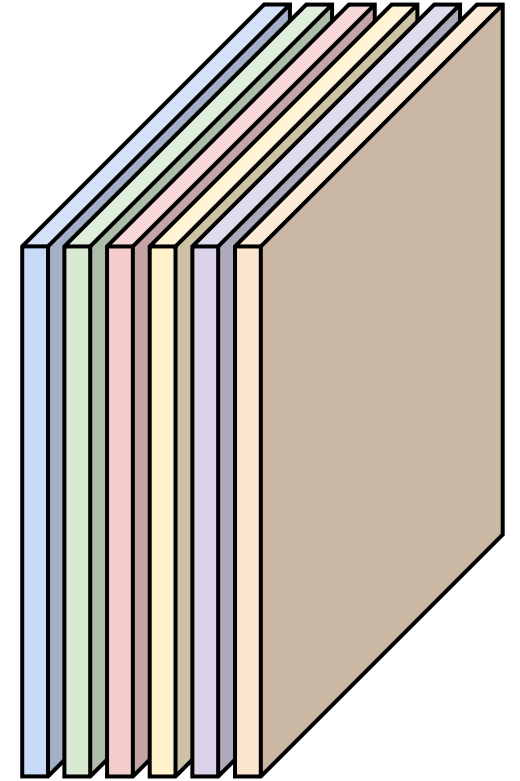
32

32

3

Convolution Layer

6x3x5x5 filters

Stack activations to get a 6x28x28 output image!

Convolution Layer

3x32x32 image

Also 6-dim bias vector:

6 activation maps,
each 1x28x28

32

32

3
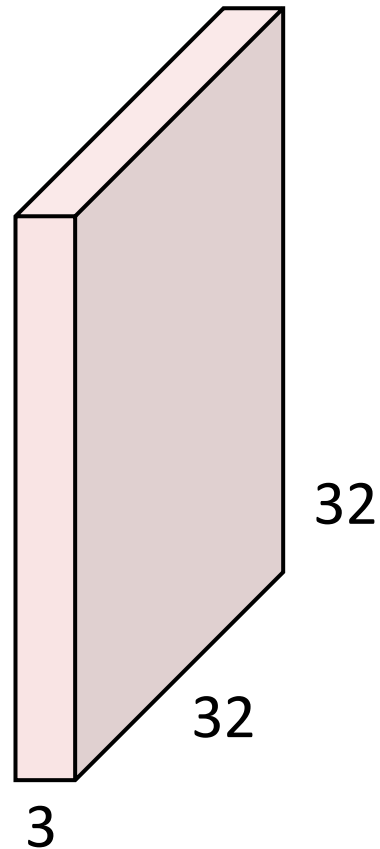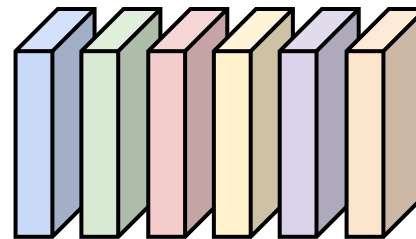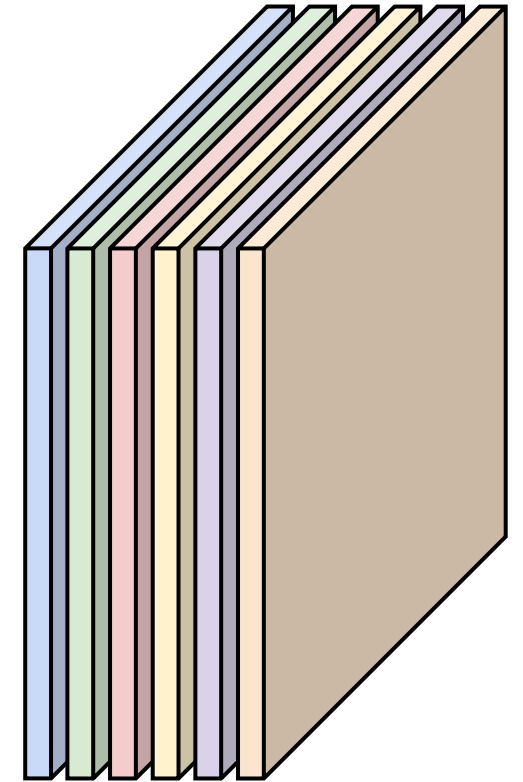
6x3x5x5
filters

Convolution
Layer

Stack activations to get a
6x28x28 output image!
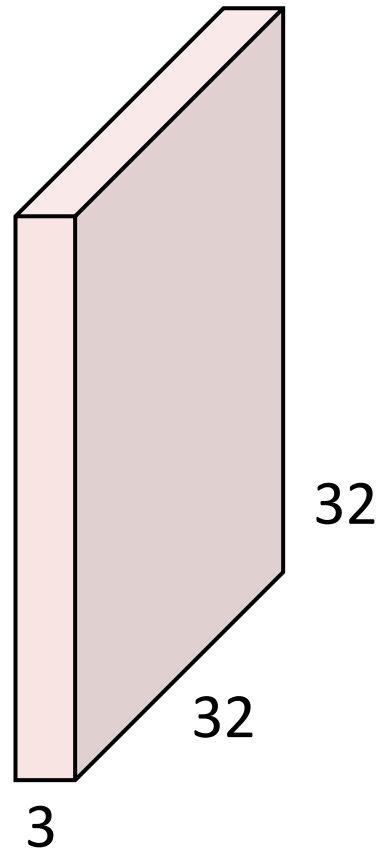
Convolution Layer

3x32x32 image

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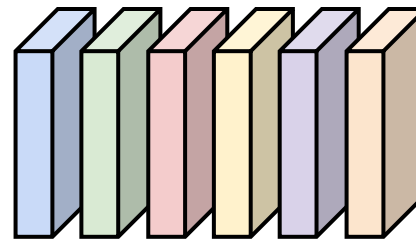
28x28 grid, at each point a 6-dim vector

32

32

3

Convolution Layer
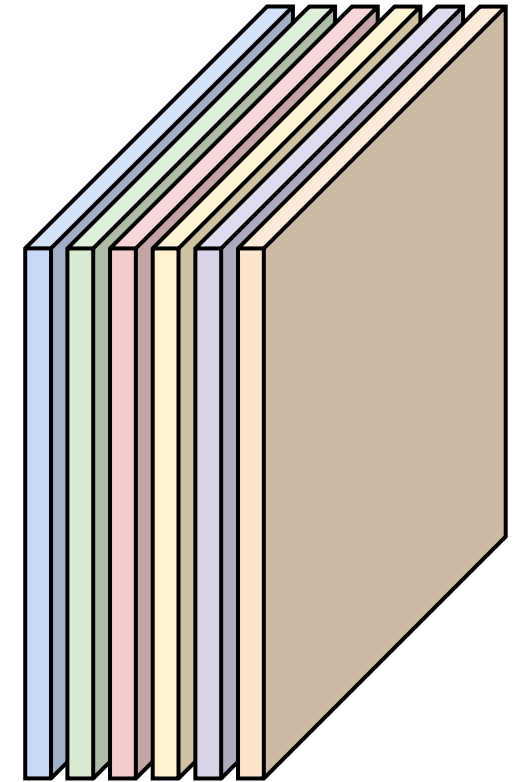
6x3x5x5 filters

Stack activations to get a 6x28x28 output image!

Convolution Layer

2x3x32x32
Batch of images
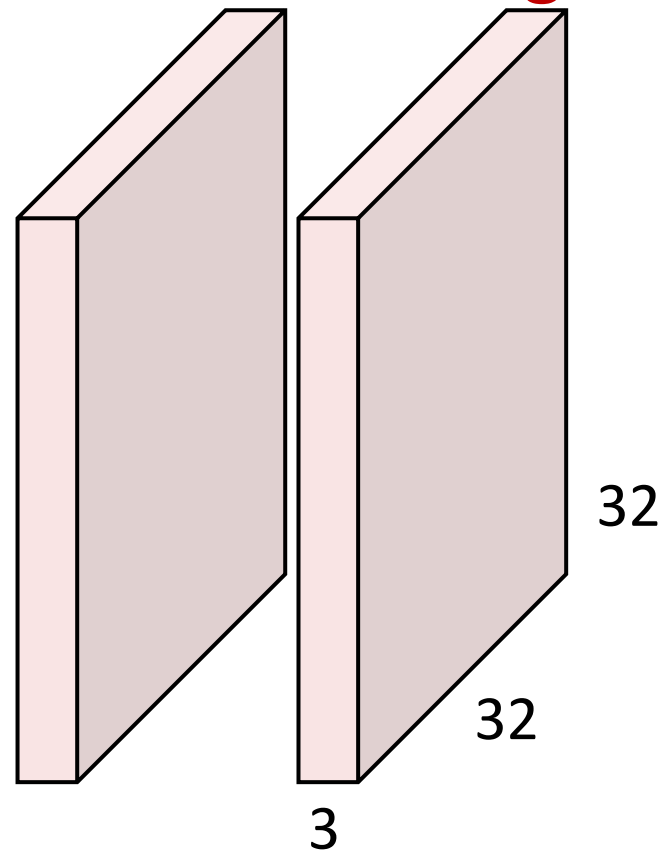
Also 6-dim bias vector:

Convolution Layer

32

32

3

6x3x5x5 filters
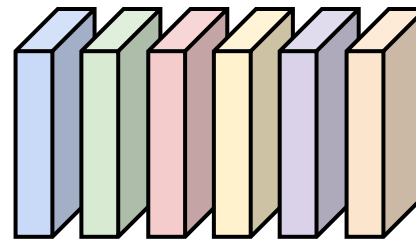
2x6x28x28
Batch of outputs
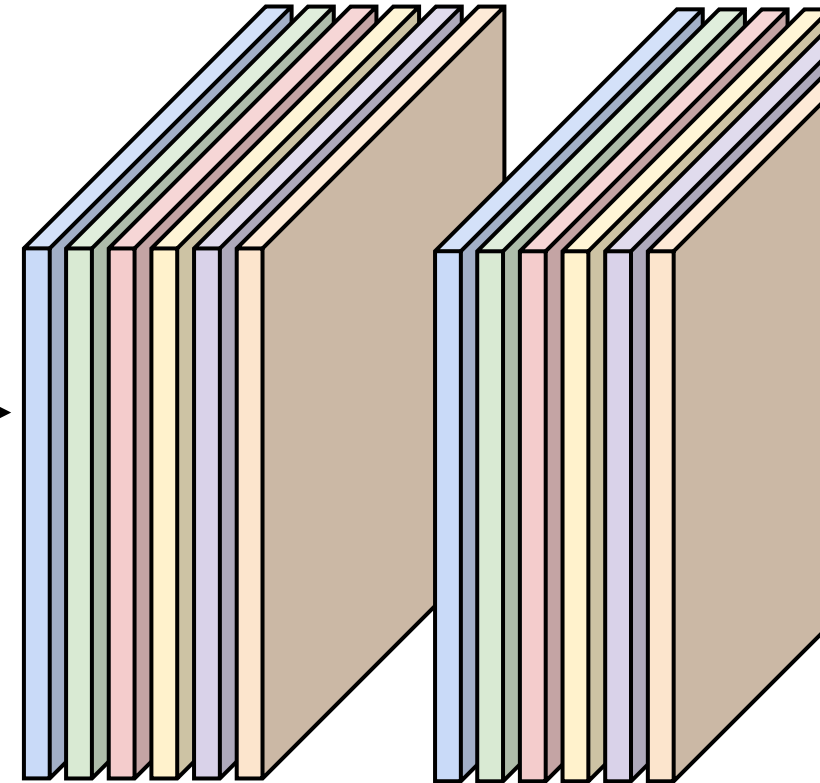
Convolution Layer

N x C_{in} x H x W
Batch of images
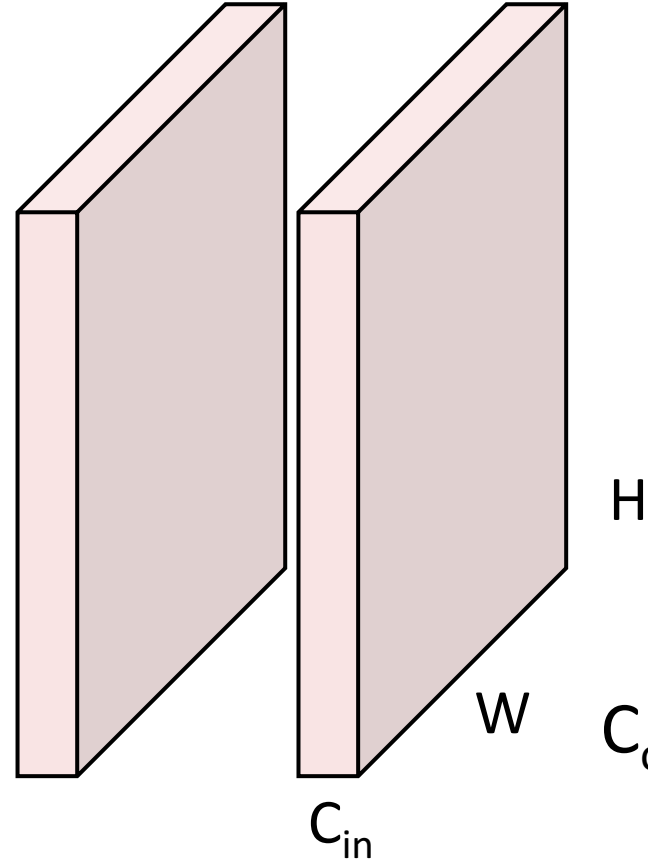
Also C_{out}-dim bias vector:

Convolution Layer
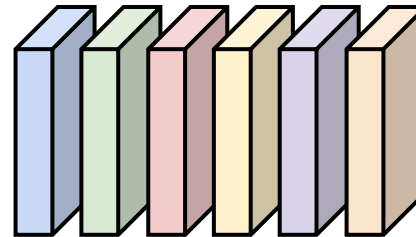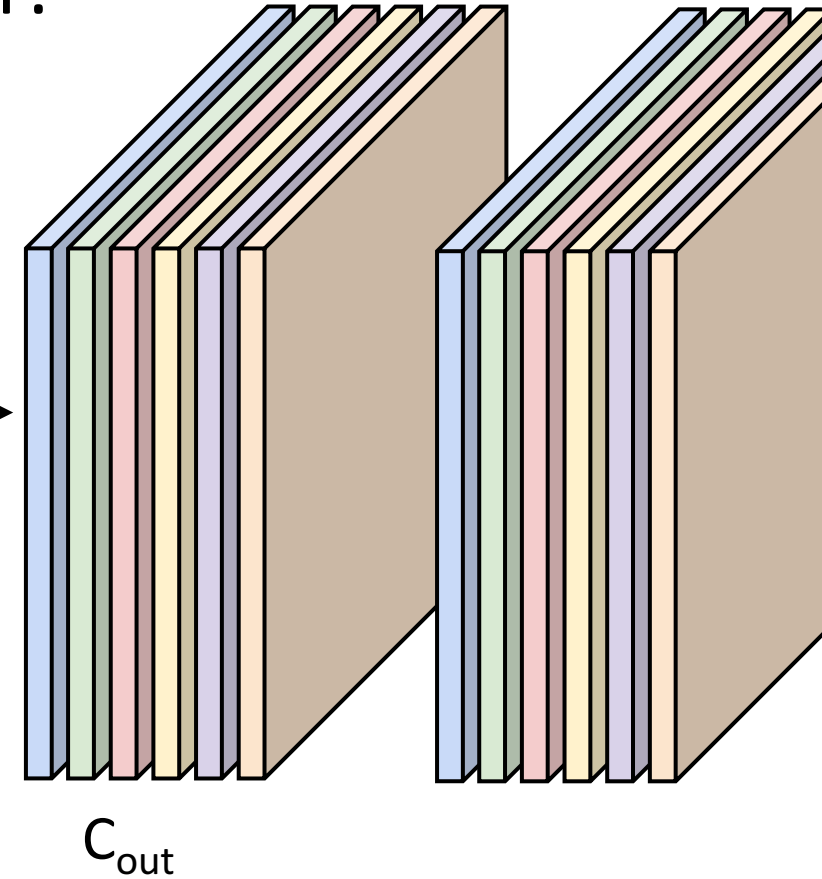
N x C_{out} x H' x W'
Batch of outputs

C_{out} x C_{in} x K_w x K_h
filters

H

W

C_{in}

C_{out}

Stacking Convolutions

W_1: 6x3x5x5
b_1: 5

W_2: 10x6x3x3
b_2: 10

W_3: 12x10x3x3
b_3: 12

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Stacking Convolutions

Q: What happens if we stack two convolution layers?

W₁: 6x3x5x5
b₁: 5

W₂: 10x6x3x3
b₂: 10

W₃: 12x10x3x3
b₃: 12

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26
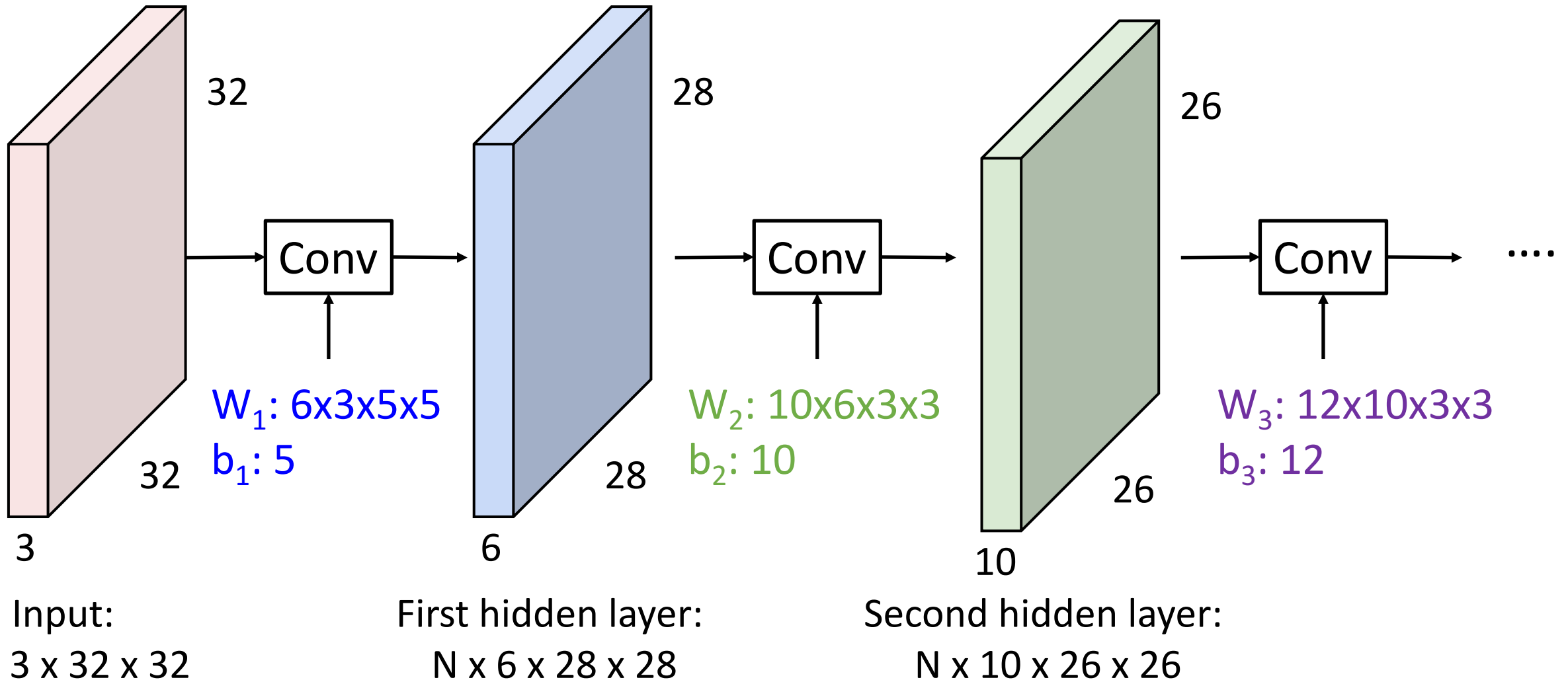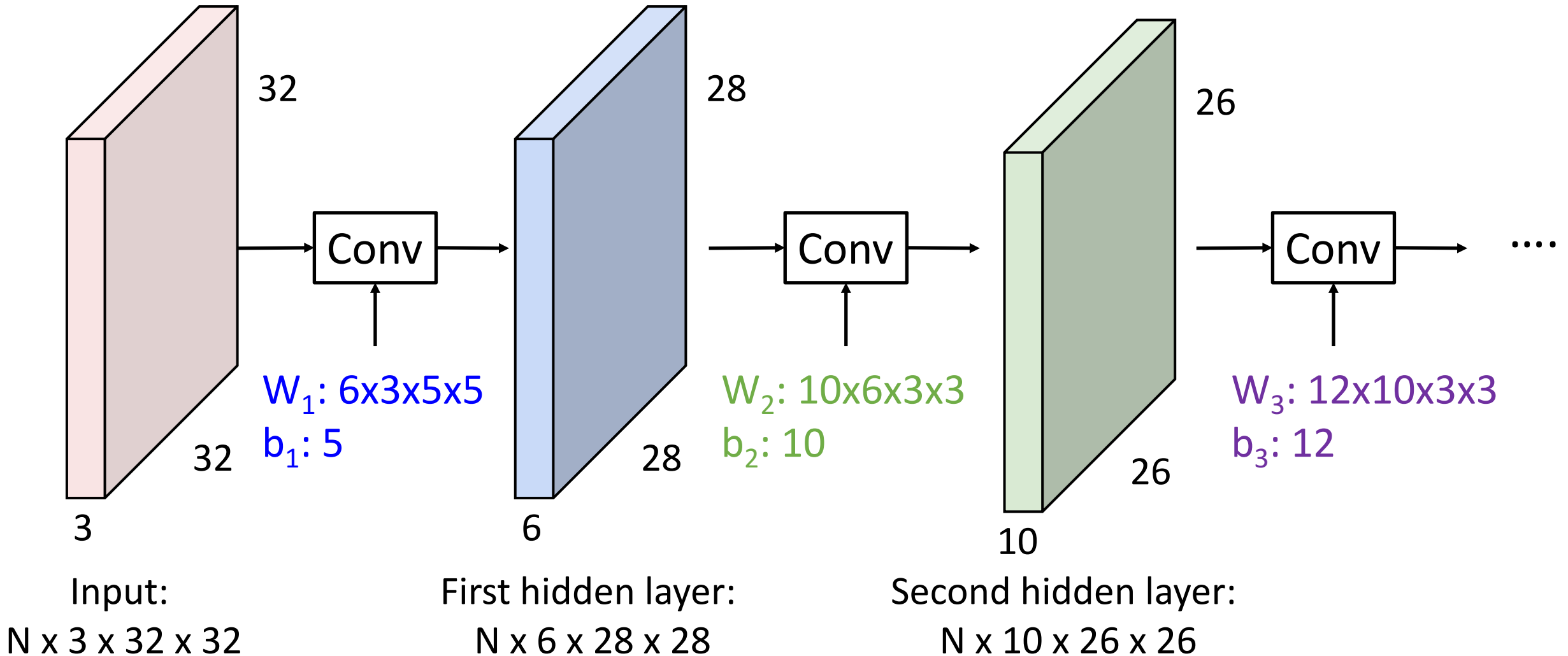
Stacking Convolutions

Q: What happens if we stack two convolution layers? (Recall $y = W_2 W_1 x$ is a linear classifier)

A: We get another convolution!

W_1: 6x3x5x5
b_1: 5

W_2: 10x6x3x3
b_2: 10

W_3: 12x10x3x3
b_3: 12

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26
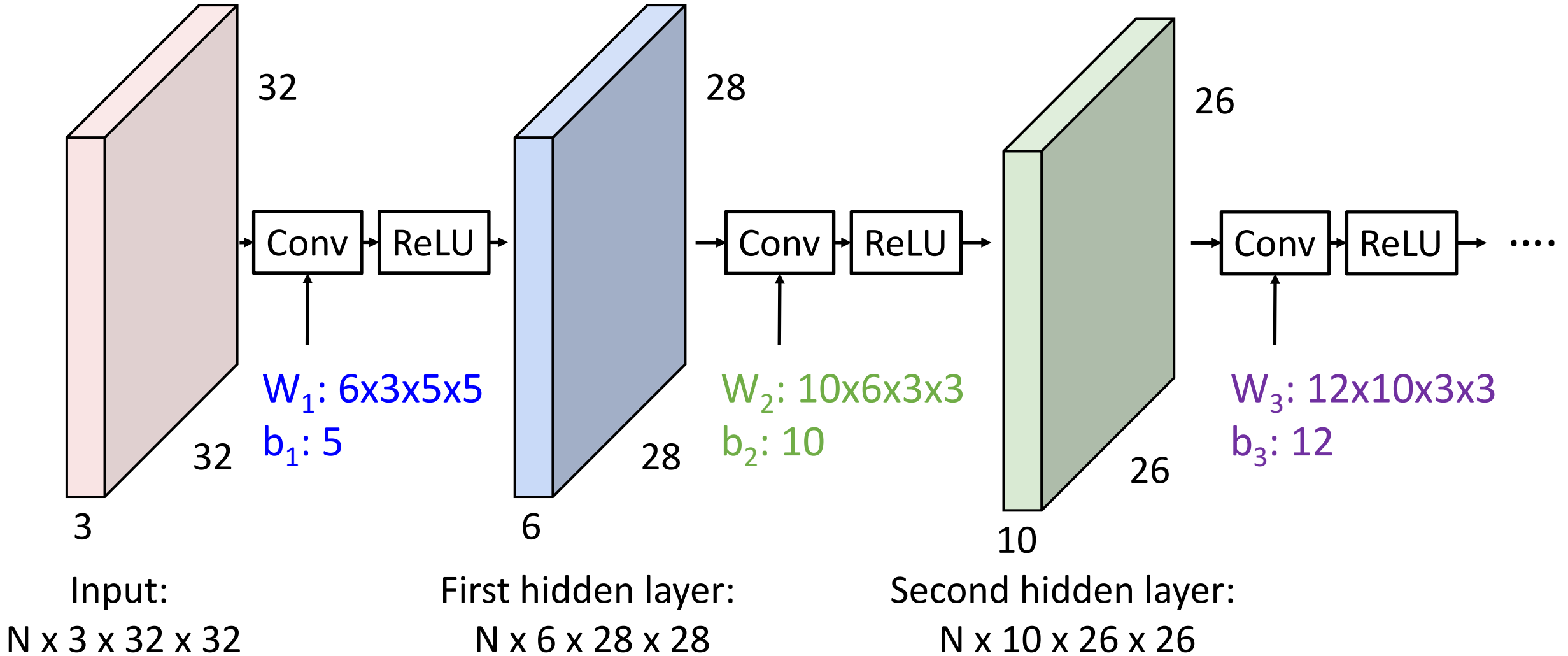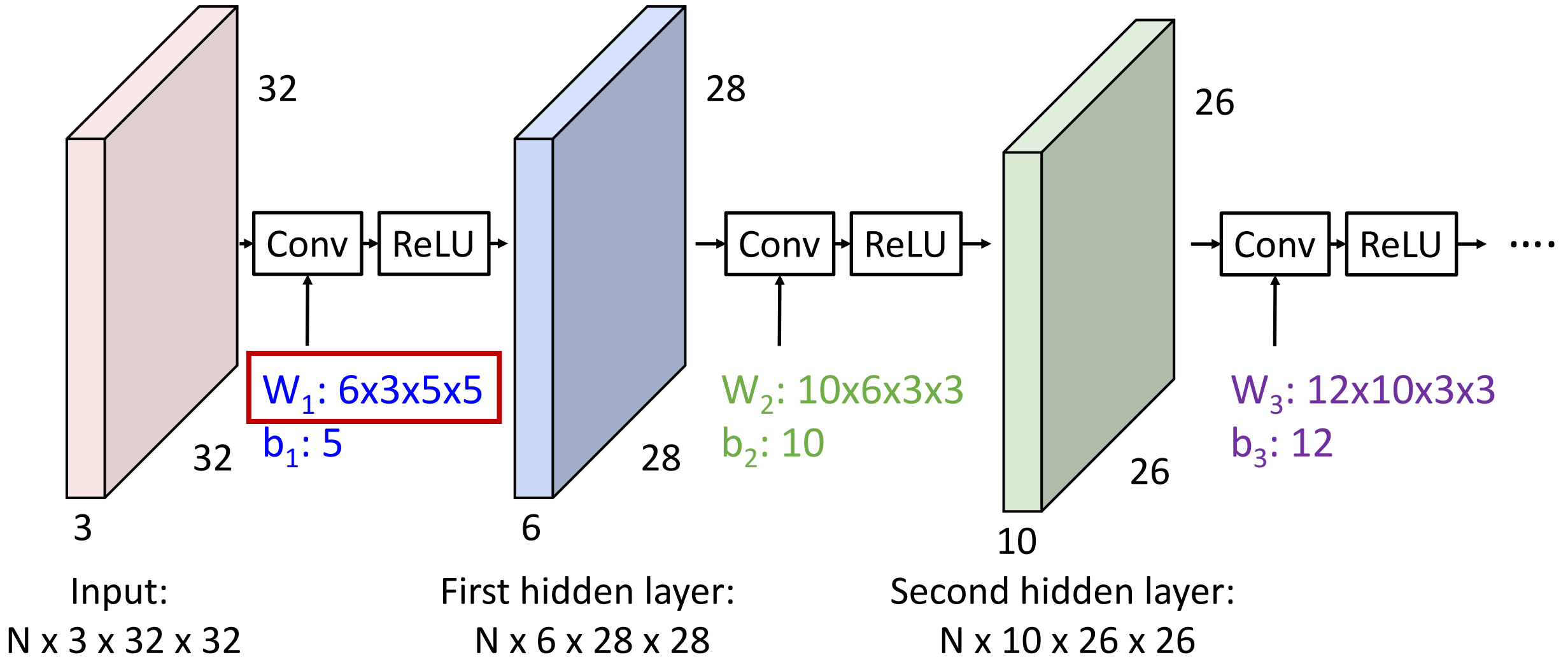
What do convolutional filters learn?

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

W_1: 6x3x5x5
b_1: 5

W_2: 10x6x3x3
b_2: 10

W_3: 12x10x3x3
b_3: 12

What do convolutional filters learn?

32

28

Conv → ReLU →

W₁: 6x3x5x5
b₁: 5

32

28

3

6
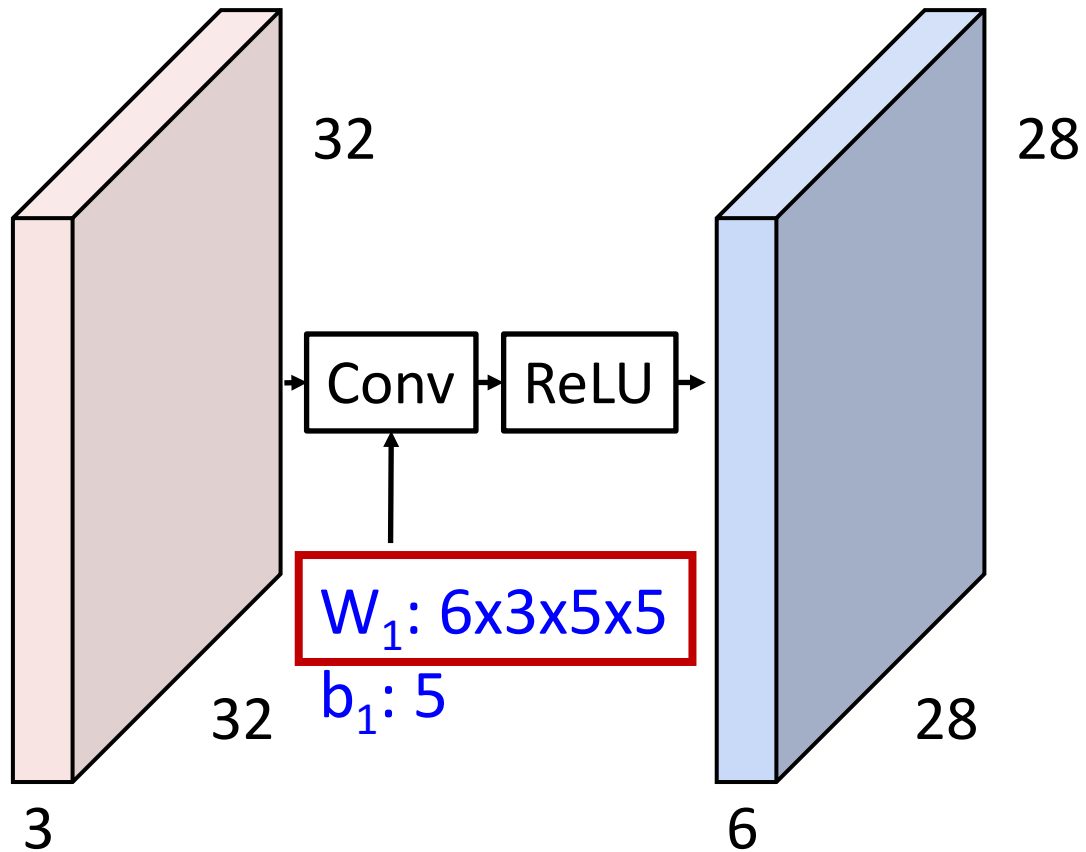
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Linear classifier: One template per class

plane   car   bird   cat   deer

dog   frog   horse   ship   truck

What do convolutional filters learn?

32

Conv → ReLU

W₁: 6x3x5x5
b₁: 5

32

3

Input:
N x 3 x 32 x 32

28

28

6
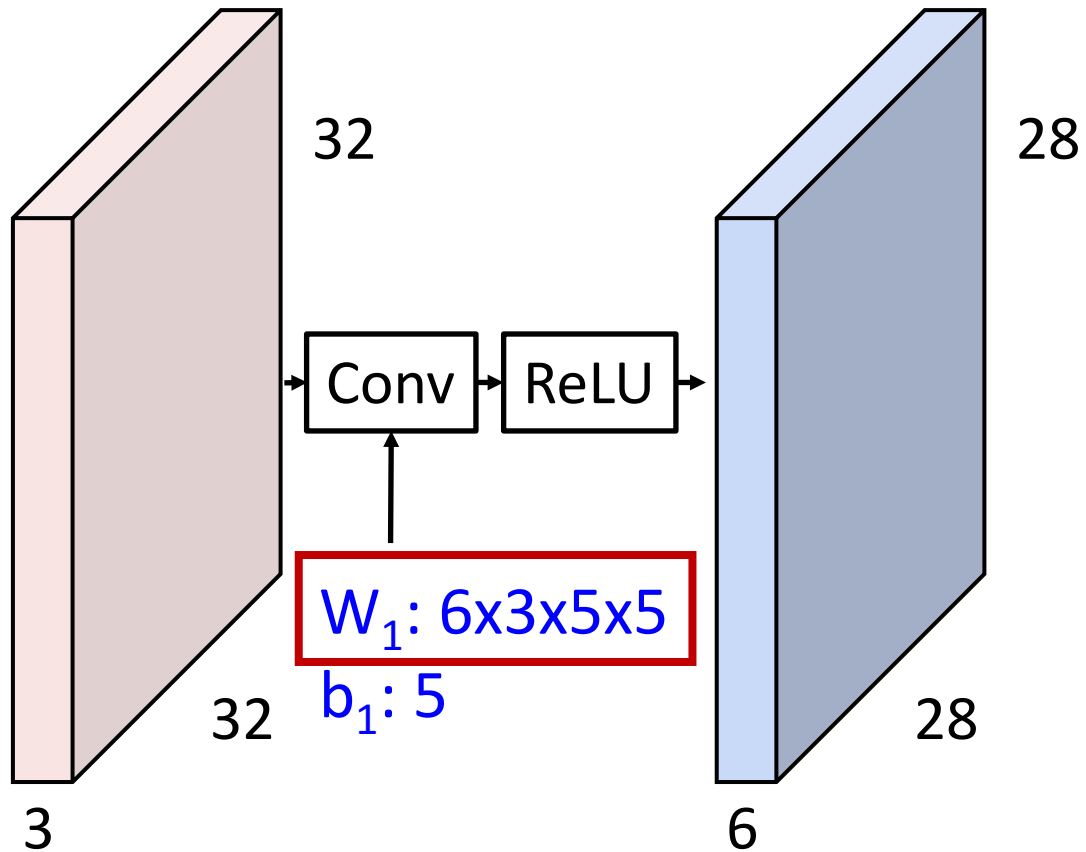
First hidden layer:
N x 6 x 28 x 28

MLP: Bank of whole-image templates

What do convolutional filters learn?

32

Conv → ReLU

W$_1$: 6x3x5x5
b$_1$: 5

32

3

28

28

6
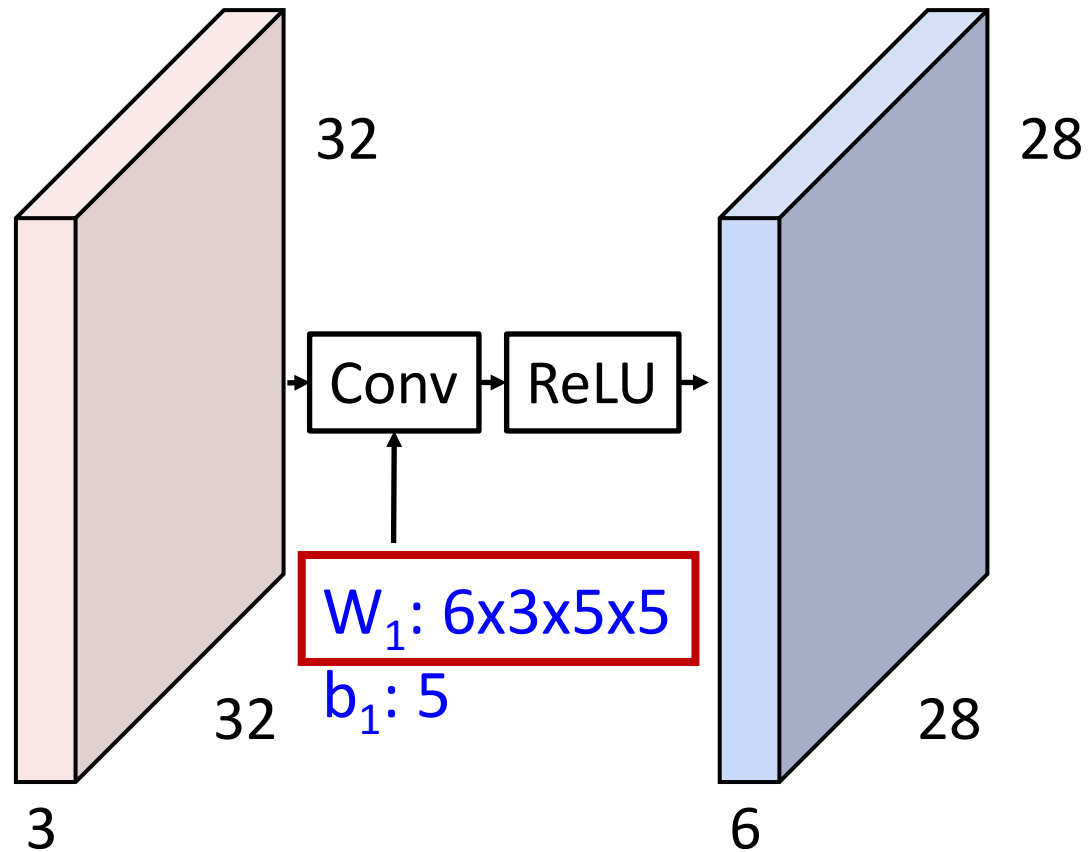
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)
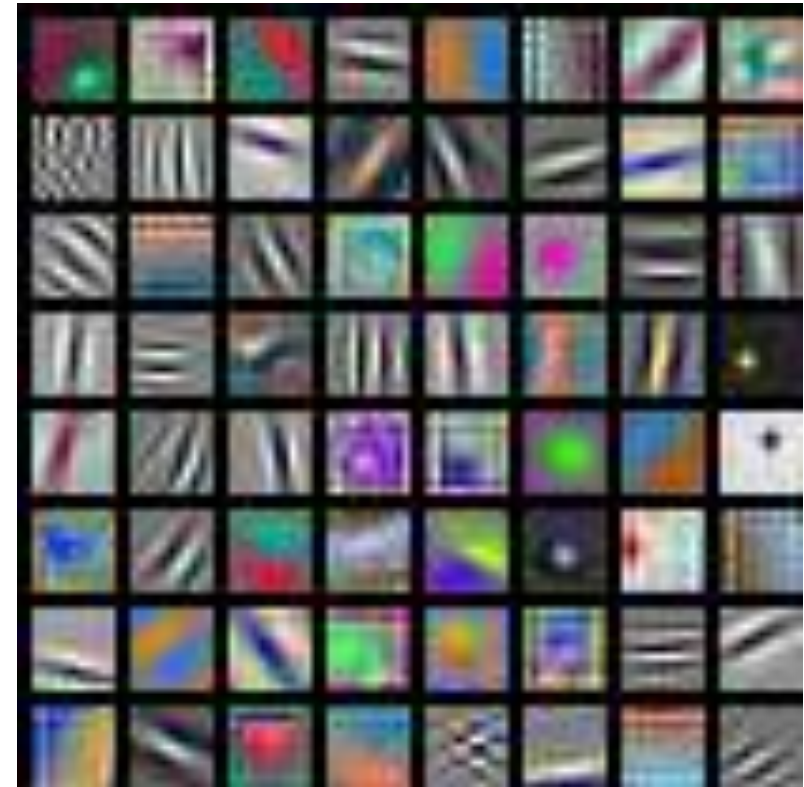
AlexNet: 64 filters, each 3x11x11

A closer look at spatial dimensions

32

28

Conv → ReLU →

W_1: 6x3x5x5
b_1: 5

32

28

3

6

Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

A closer look at spatial dimensions

Input: 7x7
Filter: 3x3

7

7

A closer look at spatial dimensions

Input: 7x7
Filter: 3x3

7

7

A closer look at spatial dimensions

Input: 7x7
Filter: 3x3

7

7

A closer look at spatial dimensions

Input: 7x7
Filter: 3x3

7

7

A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
Output: 5x5

7

7

A closer look at spatial dimensions

Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W − K + 1

Problem: Feature maps "shrink" with each layer!

7

7

A closer look at spatial dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem: Feature maps "shrink" with each layer!

Solution: **padding**
Add zeros around the input

A closer look at spatial dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Padding: P
Output: W – K + 1 + 2P

Very common:
Set P = (K – 1) / 2 to make output have same size as input!

Receptive Fields

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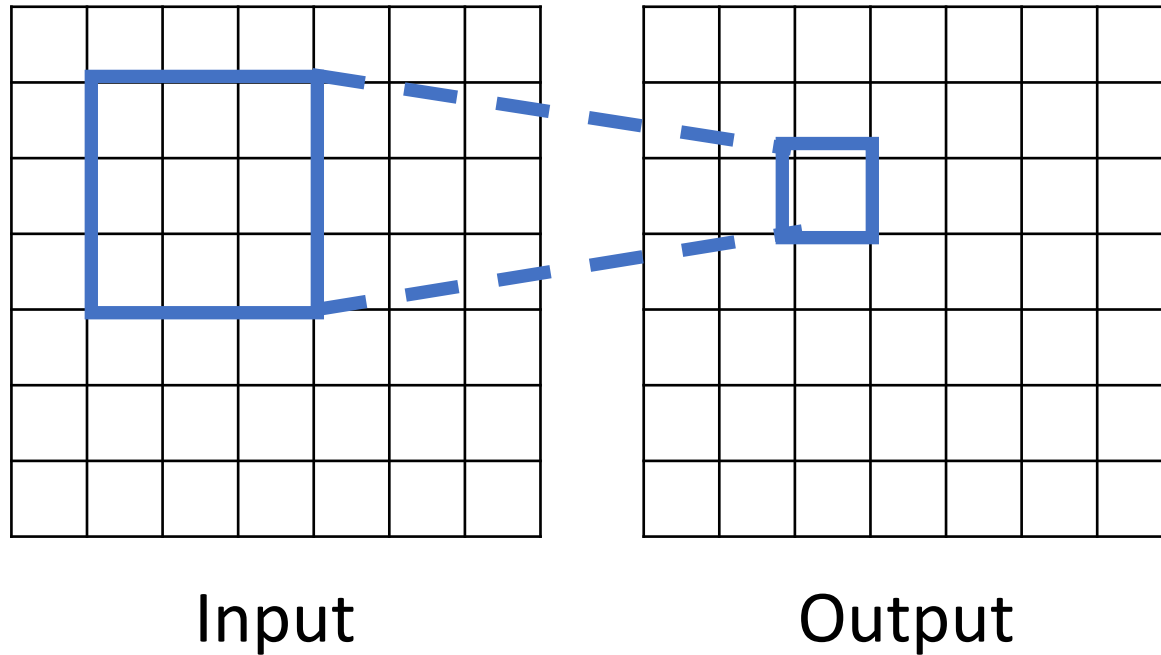
Input   Output

Receptive Fields

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

Input   Output
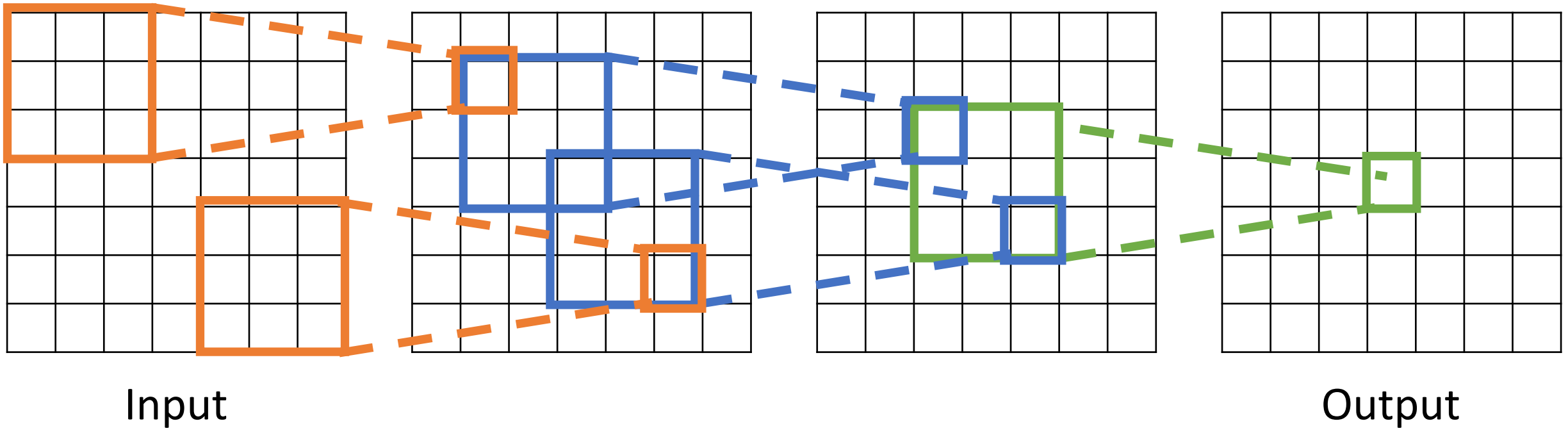
Be careful – "receptive field in the input" vs "receptive field in the previous layer"
Hopefully clear from context!

Receptive Fields

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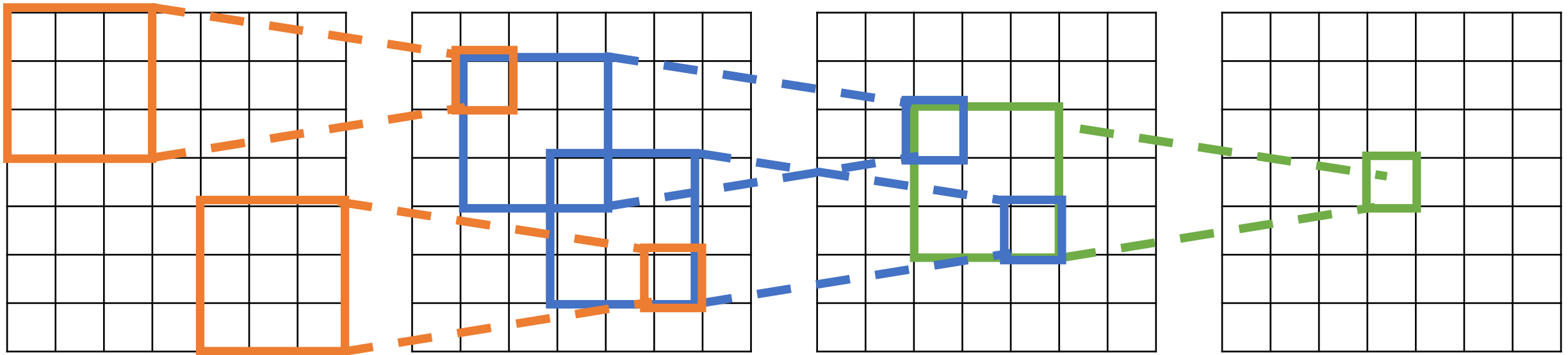
Input

Output

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

Receptive Fields

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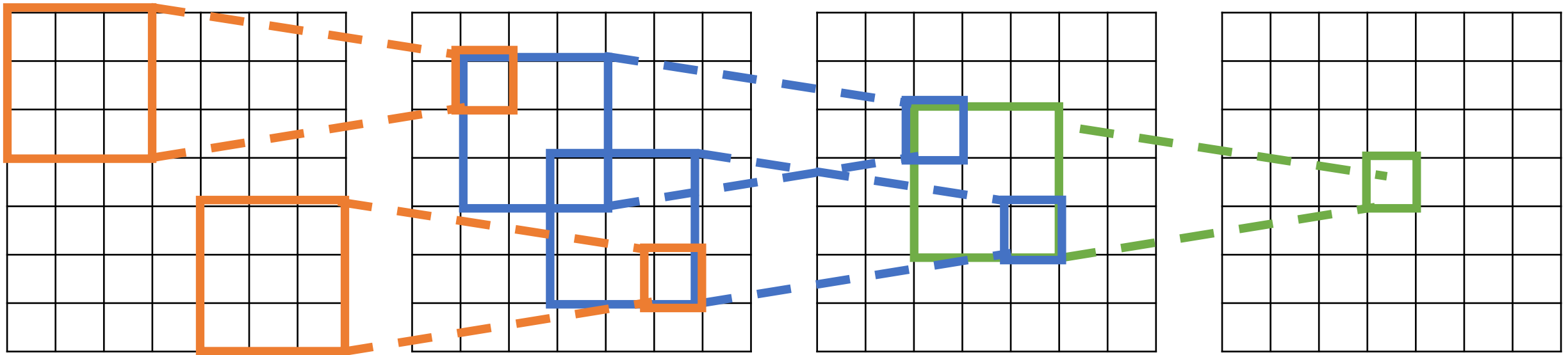
Input

Output

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

Solution: Downsample inside the network
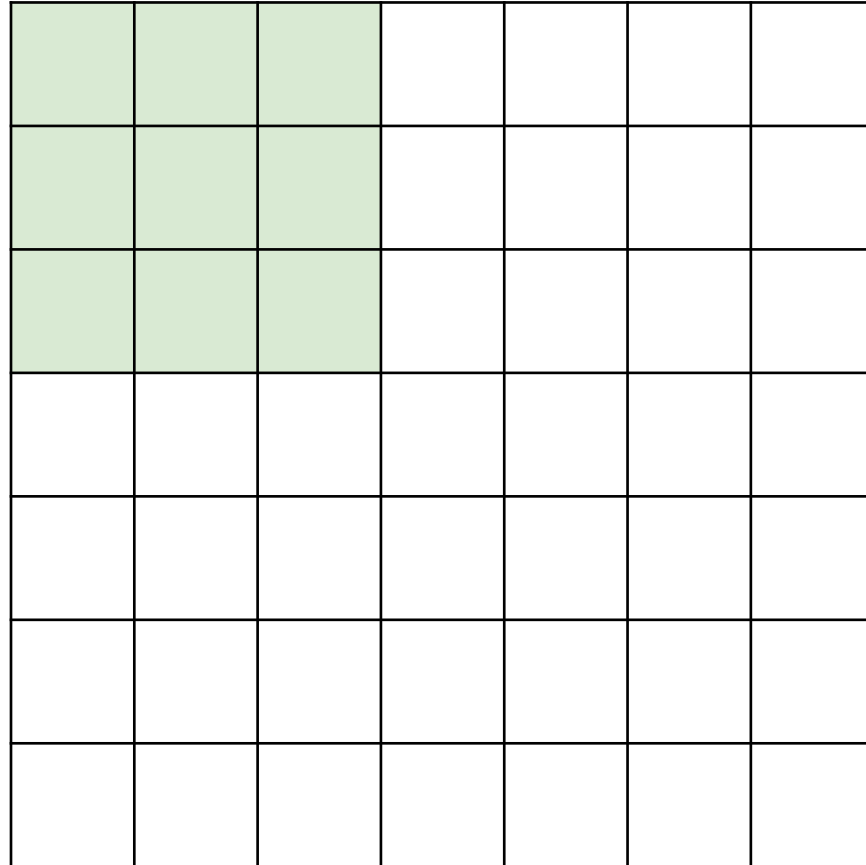
Strided Convolution

Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution
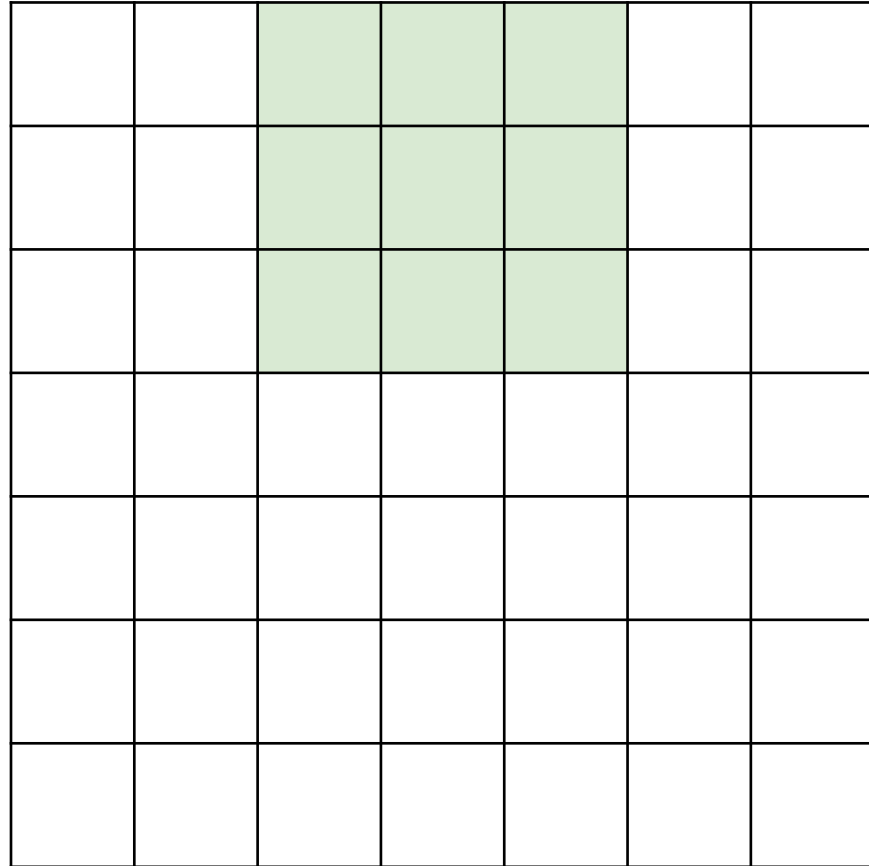
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution
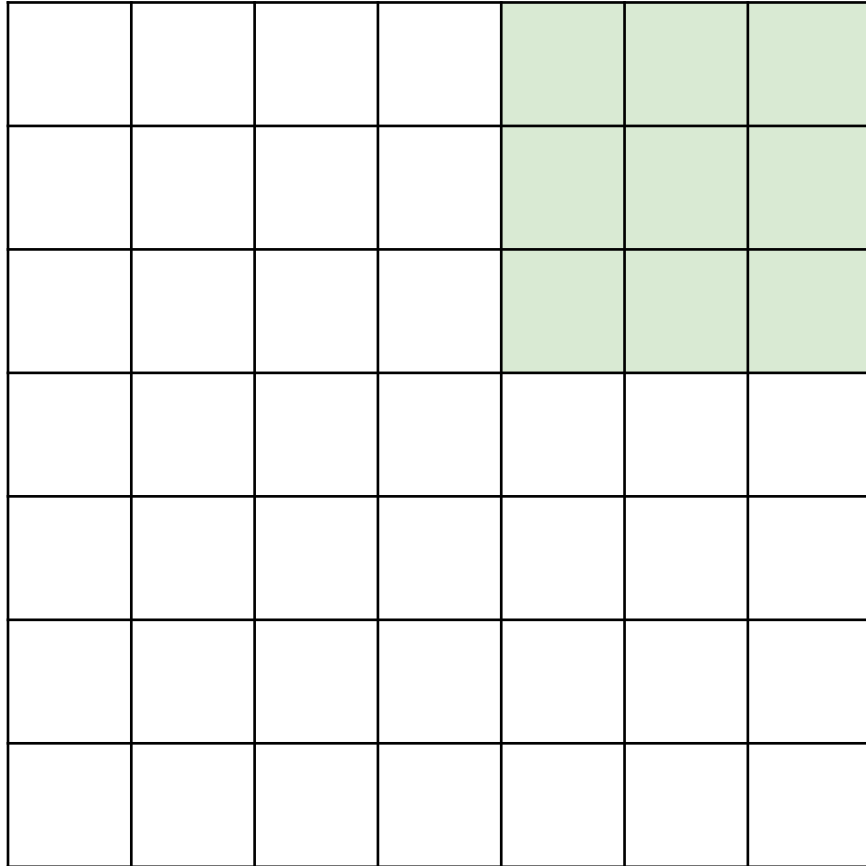
Input: 7x7
Filter: 3x3   Output: 3x3
Stride: 2

Strided Convolution
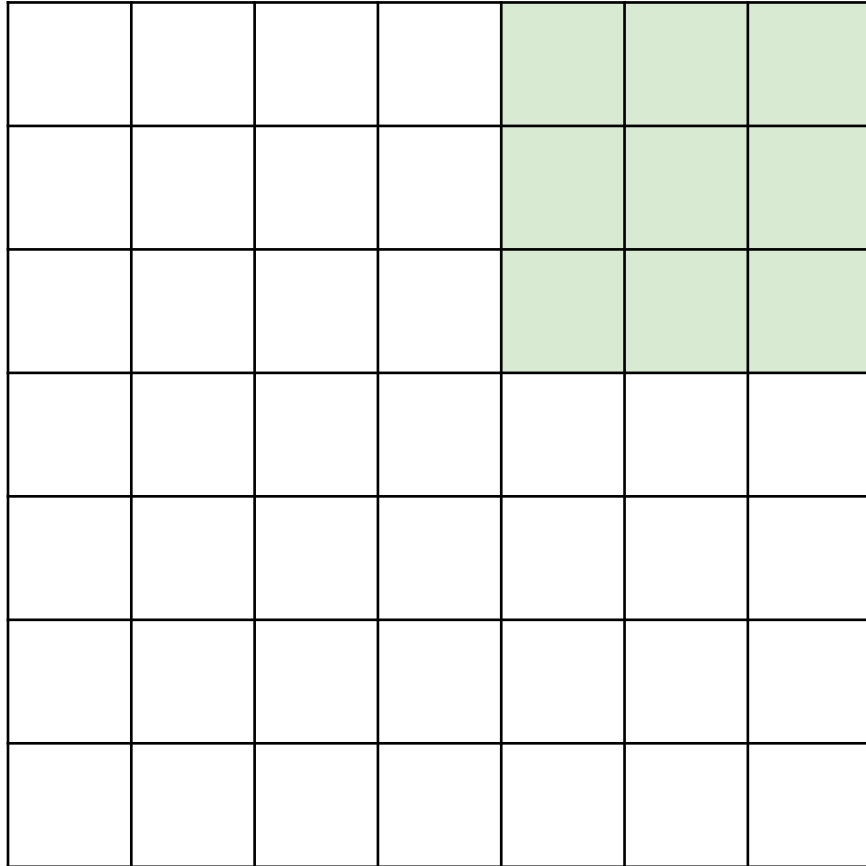
Input: 7x7
Filter: 3x3   Output: 3x3
Stride: 2

In general:
Input: W
Filter: K
Padding: P
Stride: S
Output: (W – K + 2P) / S + 1

Convolution Example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2
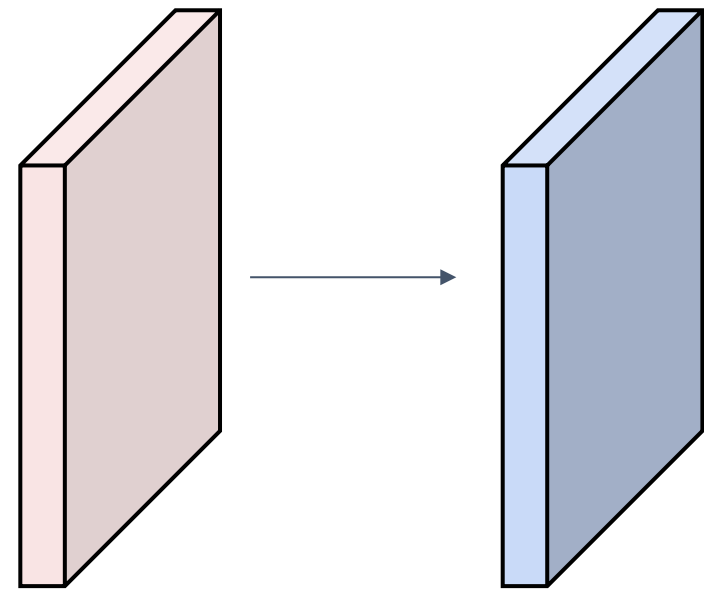
Output volume size: ?

Convolution Example

Input volume: 3 x **32** x **32**

10 **5x5** filters with stride **1**, pad **2**

Output volume size:

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

10 x 32 x 32
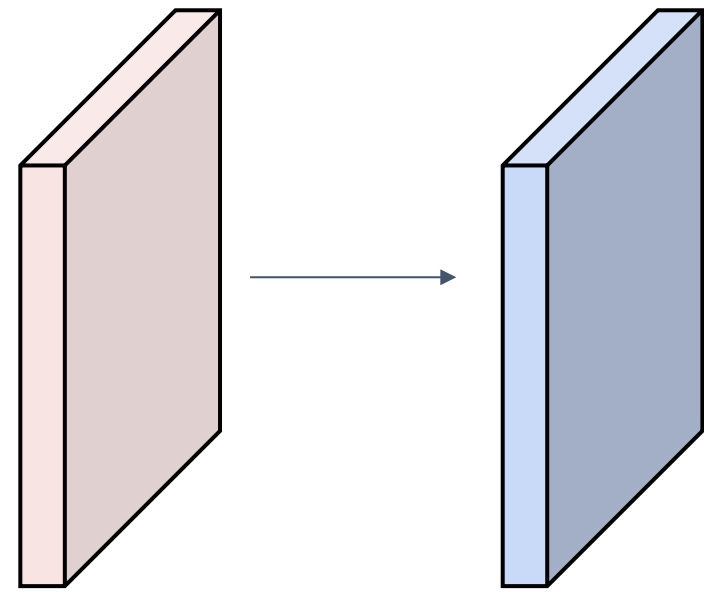
Convolution Example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: ?
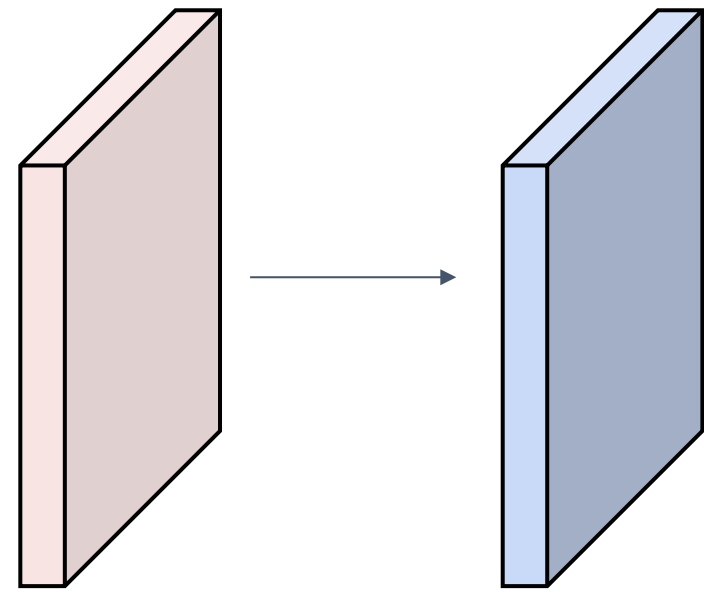
Convolution Example

Input volume: **3** x 32 x 32
10 **5**x**5** filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: **760**
Parameters per filter: **3*****5*****5** + 1 (for bias) = **76**
10 filters, so total is **10** * **76** = **760**
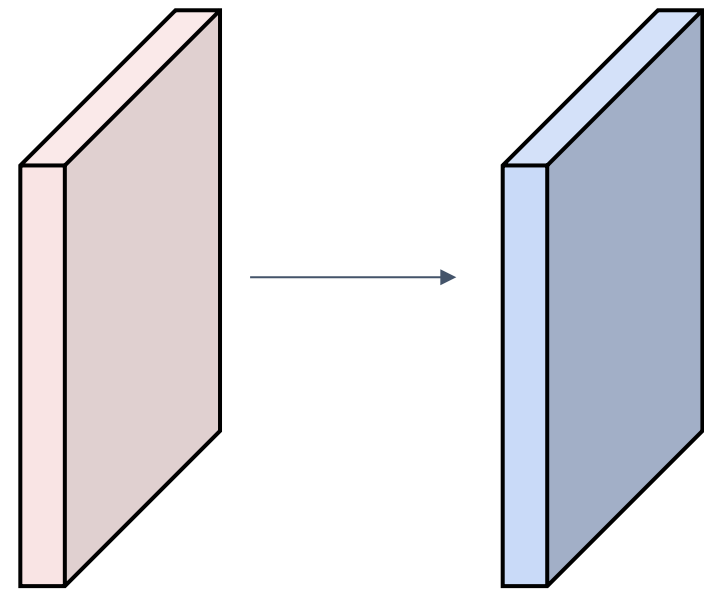
Convolution Example

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Number of multiply-add operations: ?
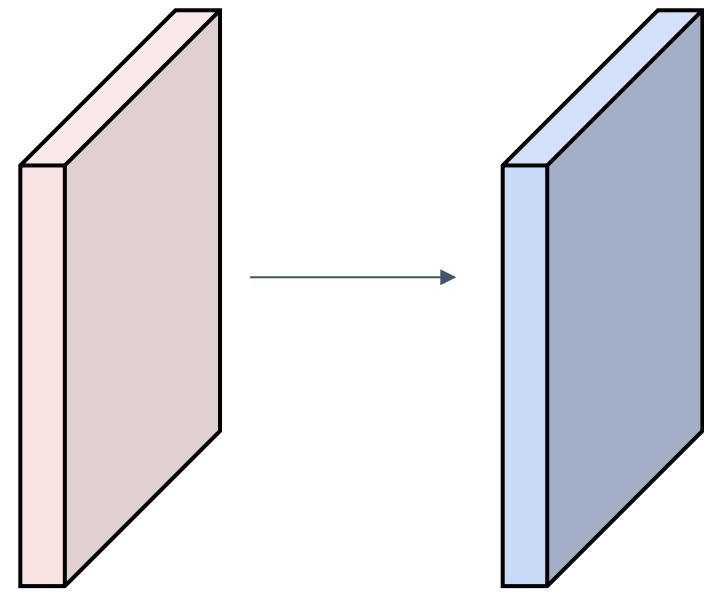
Convolution Example
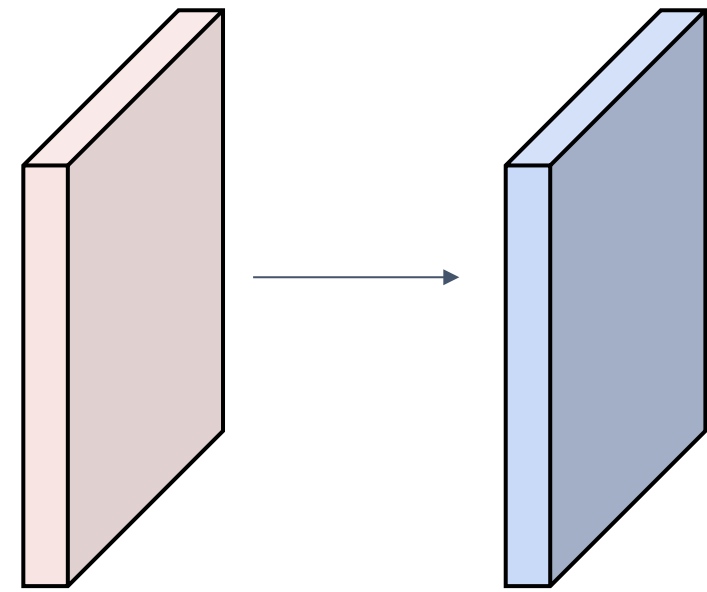
Input volume: **3** x 32 x 32

10 **5x5** filters with stride 1, pad 2

Output volume size: **10 x 32 x 32**

Number of learnable parameters: 760

Number of multiply-add operations: **768,000**

10*32*32 = 10,240 outputs; each output is the inner product of two **3**x**5x5** tensors (75 elems); total = 75*10240 = **768K**

Example: 1x1 Convolution

56

56

64

1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

56

56

32

Example: 1x1 Convolution

56

1x1 CONV
with 32 filters

56

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

56

56

Stacking 1x1 conv layers gives MLP operating on each input position

64

32
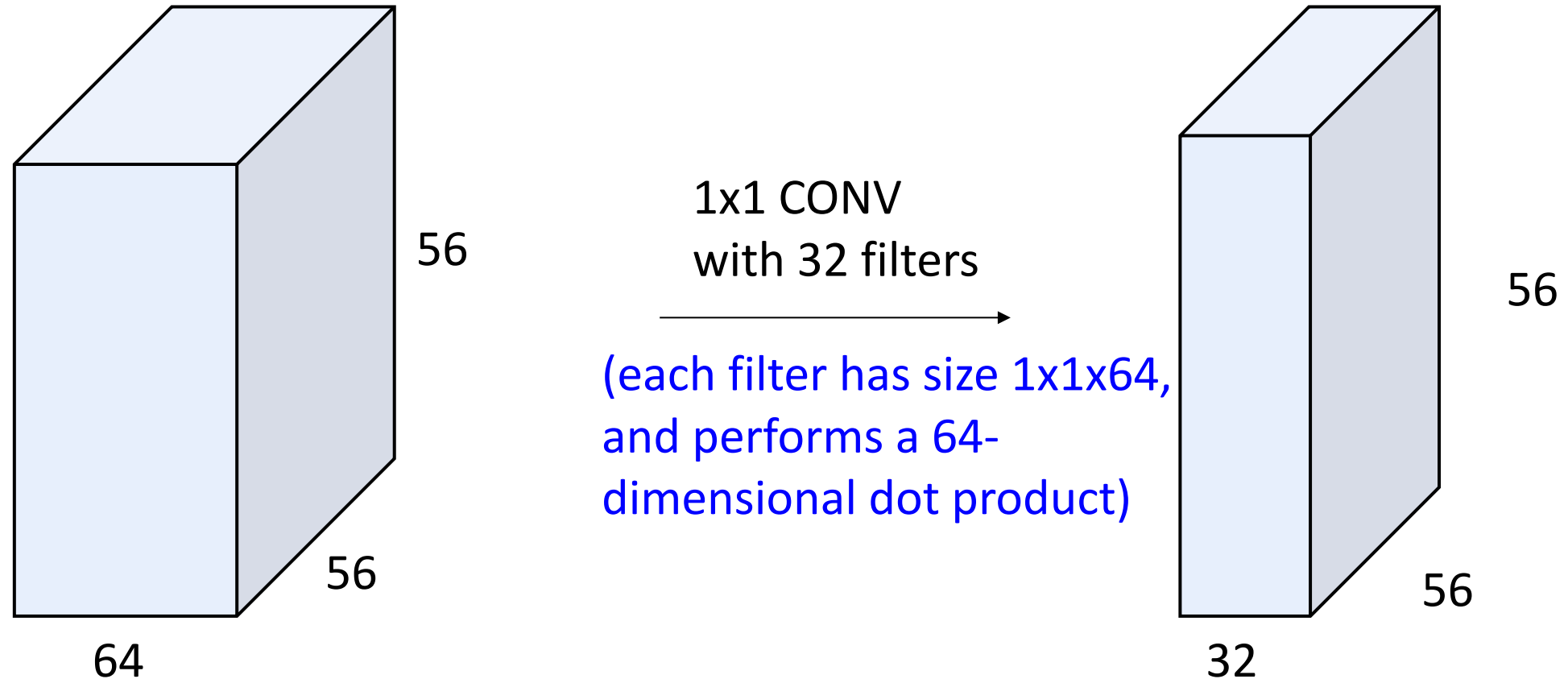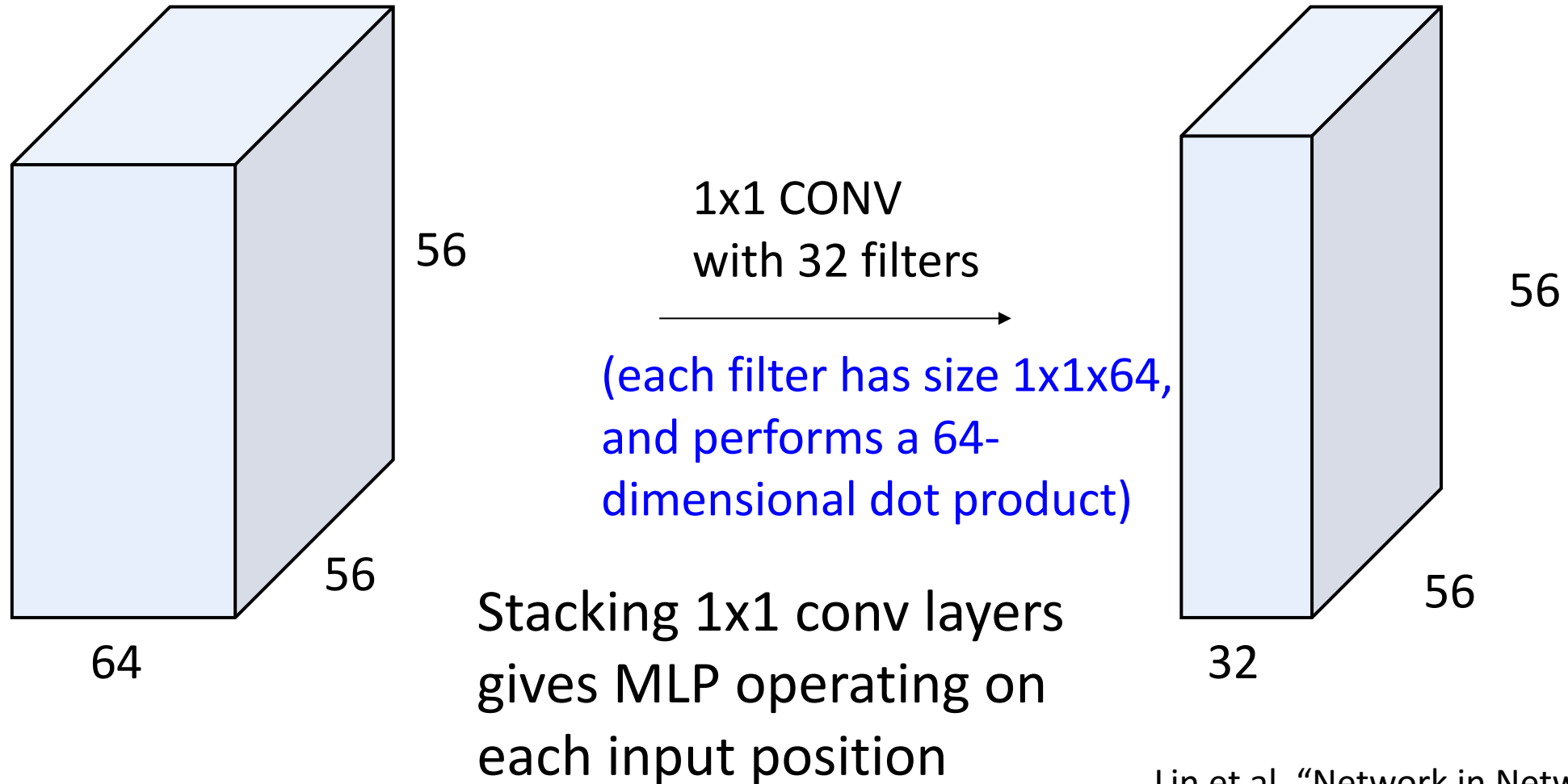
Lin et al, "Network in Network", ICLR 2014

Convolution Summary

Input: C_{in} x H x W

Hyperparameters:
- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:
- H' = (H − K + 2P) / S + 1
- W' = (W − K + 2P) / S + 1

Convolution Summary

Input: C_{in} x H x W

Hyperparameters:

- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:

- H' = (H − K + 2P) / S + 1
- W' = (W − K + 2P) / S + 1

Common settings:

$K_H = K_W$ (Small square filters)
P = (K − 1) / 2 ("Same" padding)
C_{in}, C_{out} = 32, 64, 128, 256 (powers of 2)
K = 3, P = 1, S = 1 (3x3 conv)
K = 5, P = 2, S = 1 (5x5 conv)
K = 1, P = 0, S = 1 (1x1 conv)
K = 3, P = 1, S = 2 (Downsample by 2)

Other types of convolution
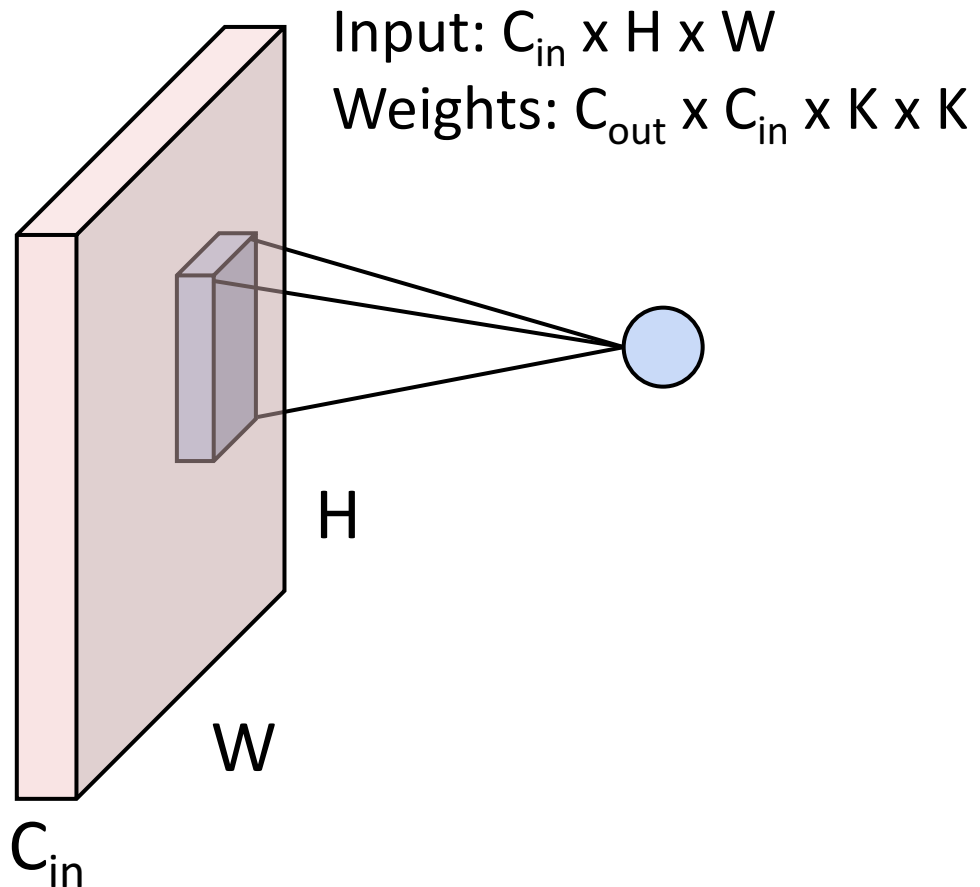
So far: 2D Convolution

Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K

H

W

C_{in}

Other types of convolution

So far: 2D Convolution

Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K

H

W

C_{in}

1D Convolution

Input: C_{in} x W
Weights: C_{out} x C_{in} x K

C_{in}

W

Other types of convolution

So far: 2D Convolution

Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K

3D Convolution

Input: C_{in} x H x W x D
Weights: C_{out} x C_{in} x K x K x K
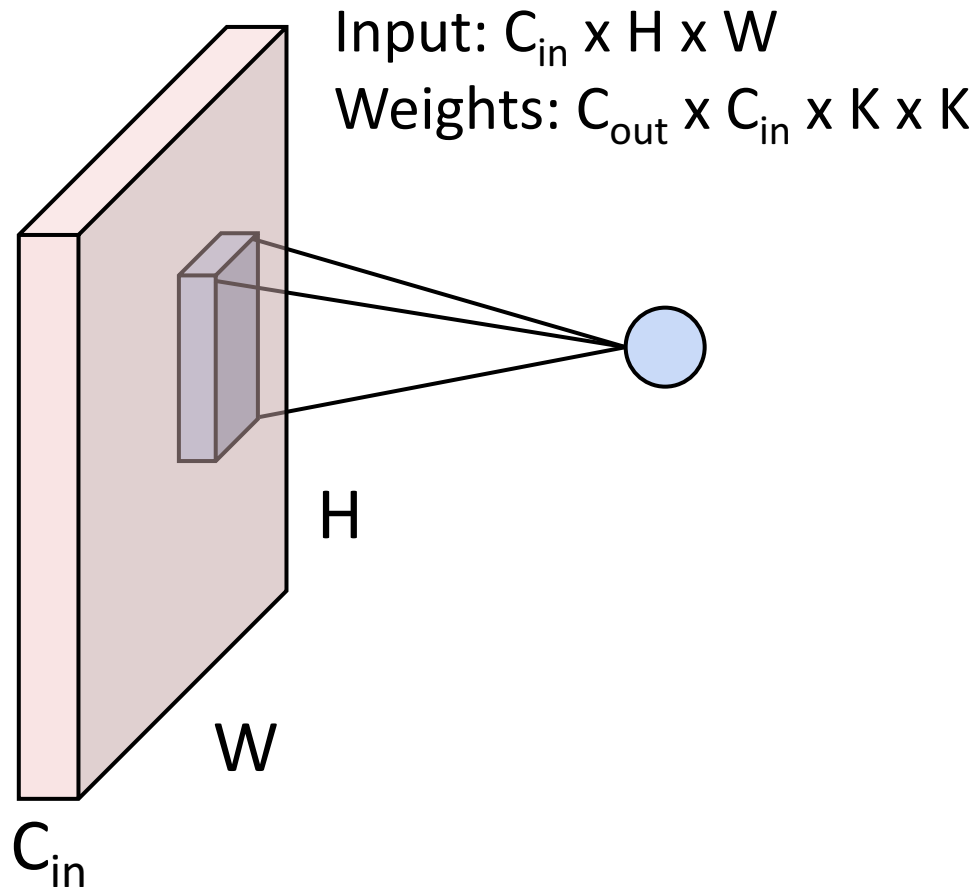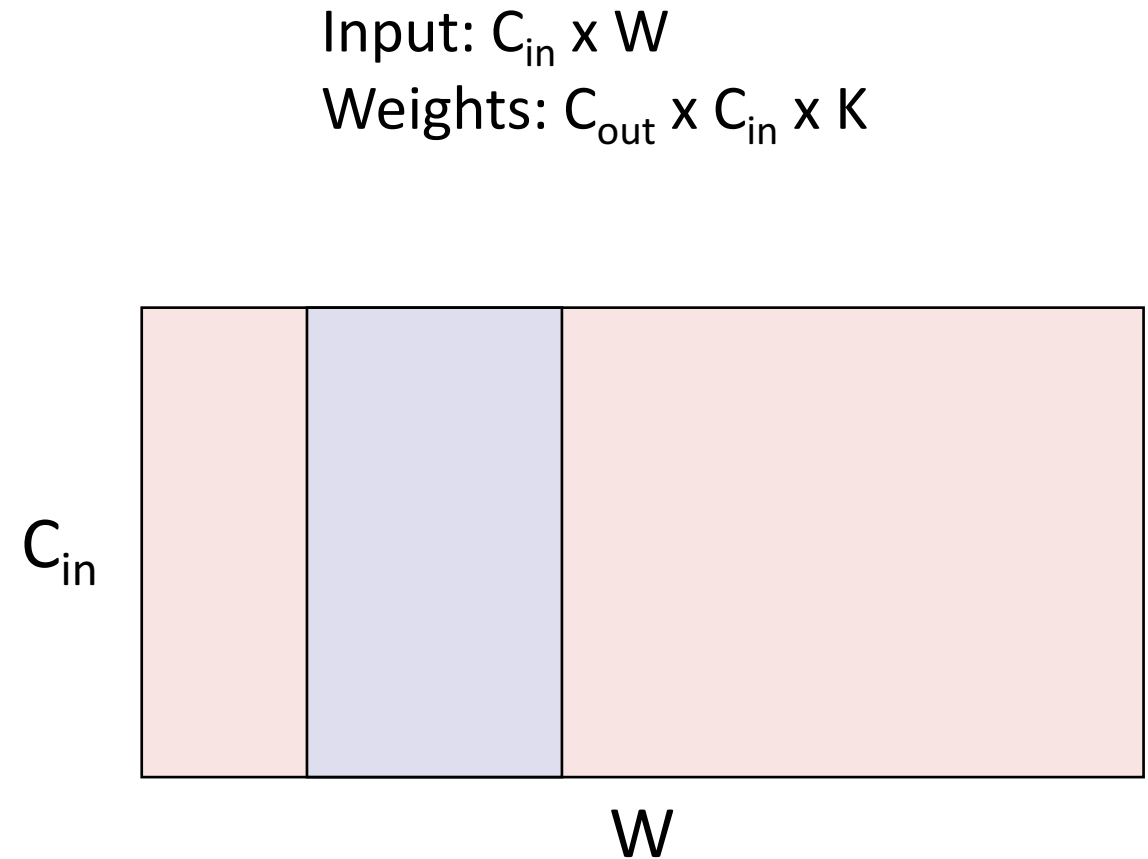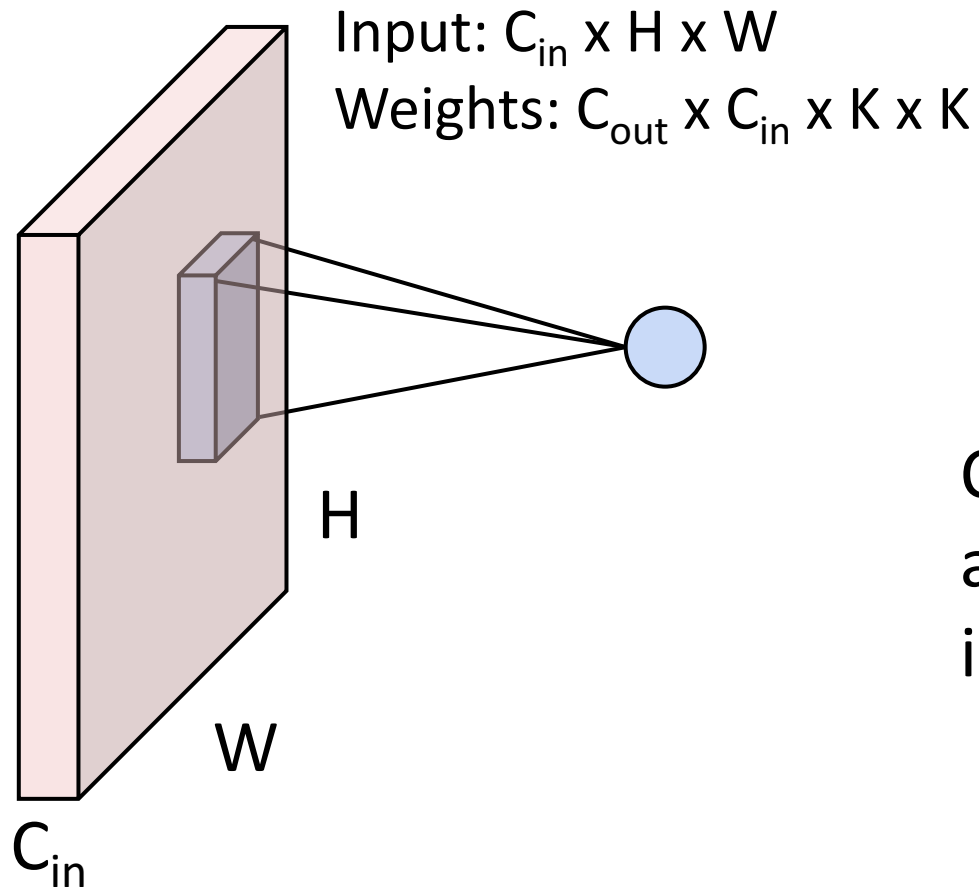
C_{in}-dim vector
at each point
in the volume

H

W

C_{in}

H

D

W
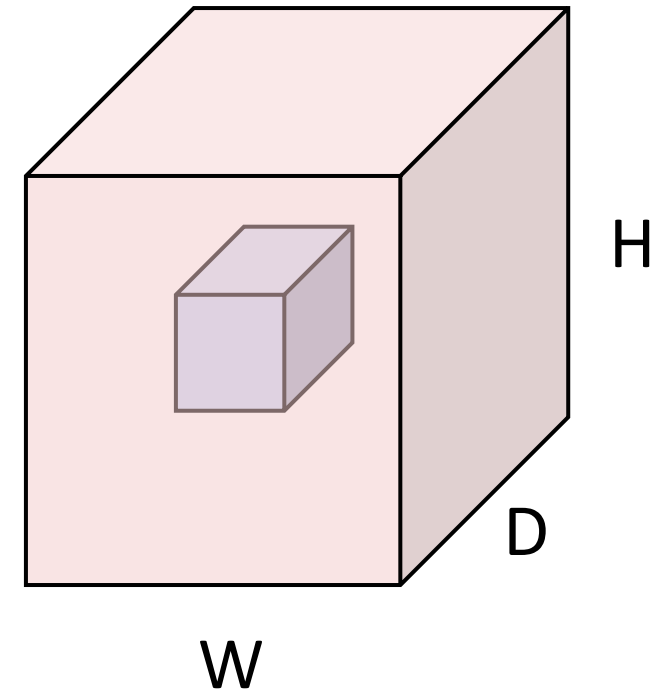
PyTorch Convolution Layer

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [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:

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

PyTorch Convolution Layers

Conv2d

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

Conv1d

CLASS torch.nn.Conv1d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'*)   [SOURCE] 🔗
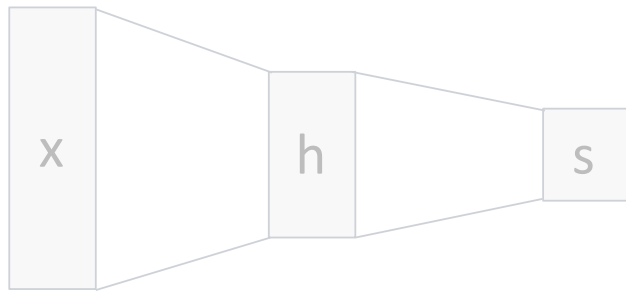
Conv3d

CLASS torch.nn.Conv3d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'*)   [SOURCE]

Components of a Convolutional Network

Fully-Connected Layers

Activation Function
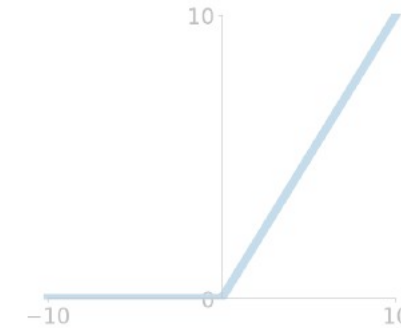
Convolution Layers
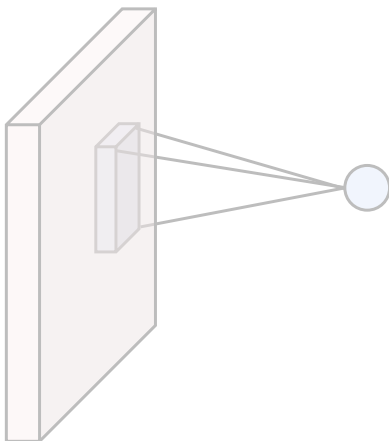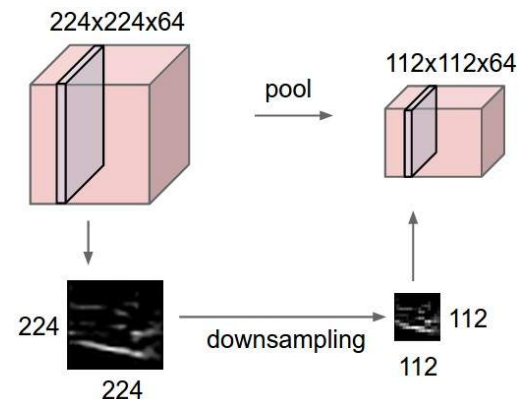
Pooling Layers

Normalization

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

Pooling Layers: Another way to downsample

224x224x64

pool →

112x112x64

224

224

downsampling

112

112
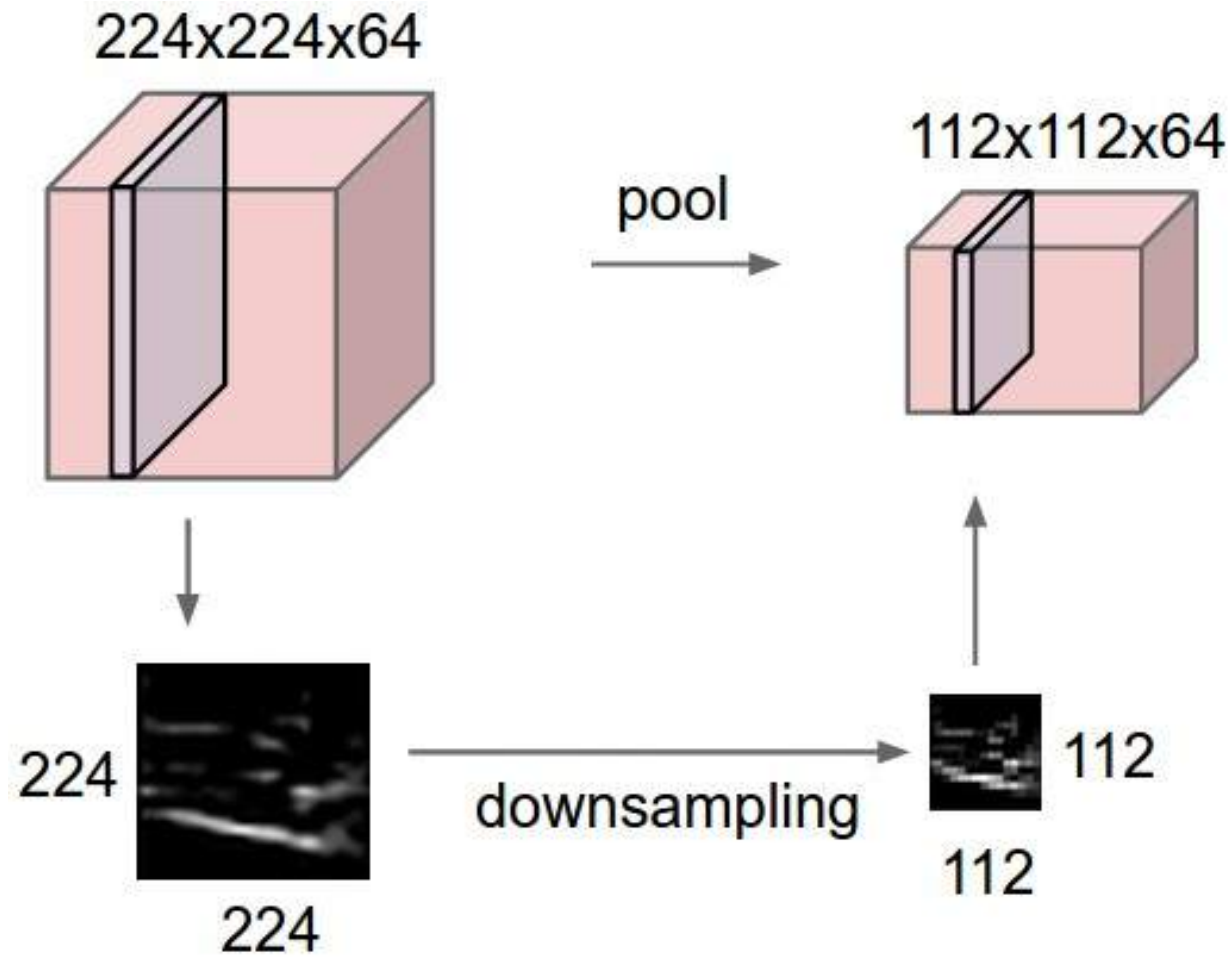
Hyperparameters:
Kernel Size
Stride
Pooling function

Max Pooling
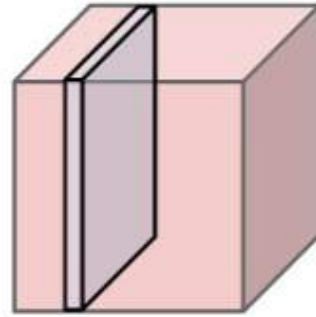
Single depth slice

224x224x64

x

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

y

Max pooling with 2x2 kernel size and stride 2

⟶

6	8
3	4

Introduces **invariance** to small spatial shifts
No learnable parameters!

Pooling Summary

Input: C x H x W

Hyperparameters:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Output: C x H' x W' where
- H' = (H − K) / S + 1
- W' = (W − K) / S + 1

Learnable parameters: None!

Common settings:
max, K = 2, S = 2
max, K = 3, S = 2 (AlexNet)

Components of a Convolutional Network

Fully-Connected Layers

x   h   s

Activation Function

Convolution Layers

Pooling Layers

224x224x64

pool

112x112x64

224

downsampling

112

224

112
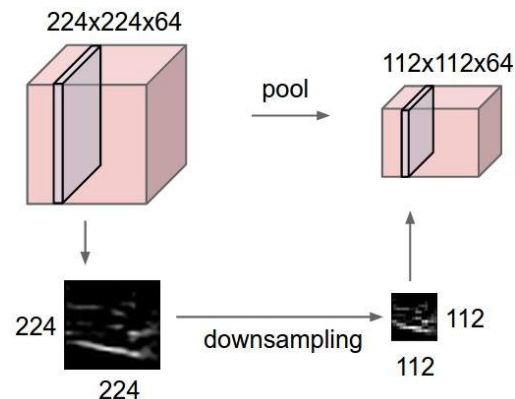
Normalization

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

Convolutional Networks

Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

Example: LeNet-5
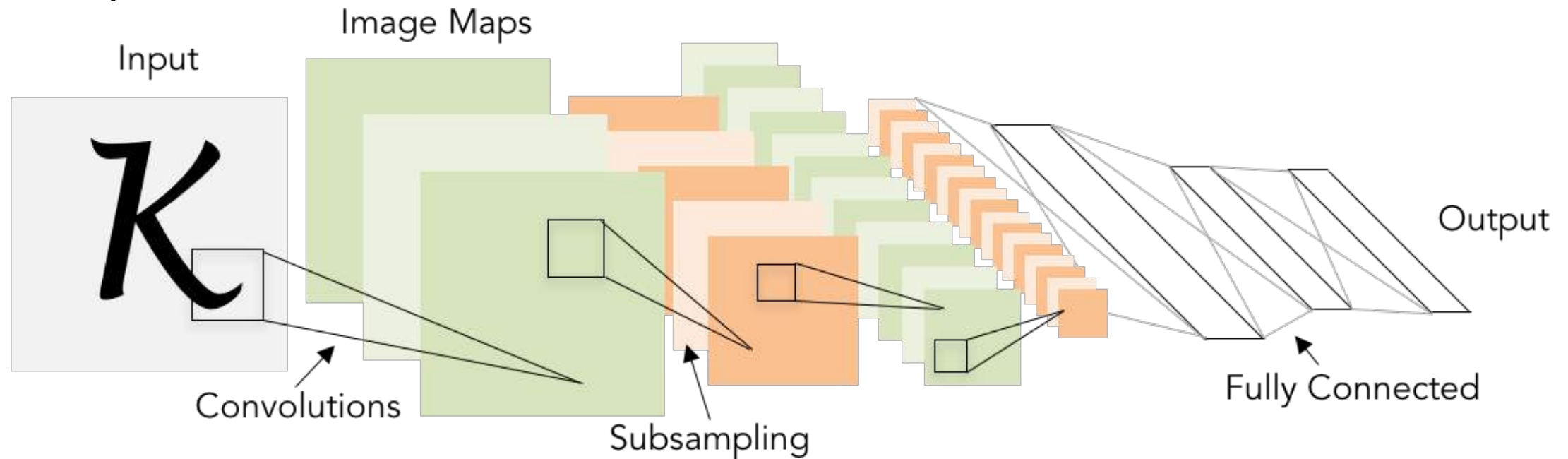
Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
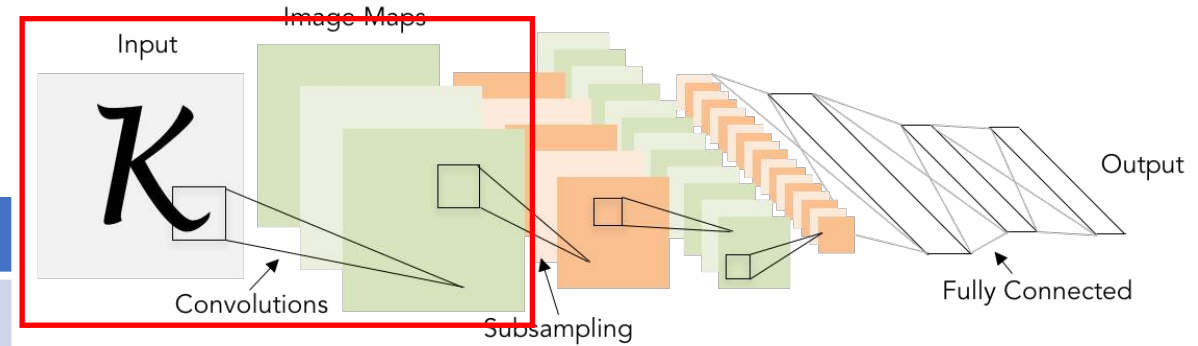
Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

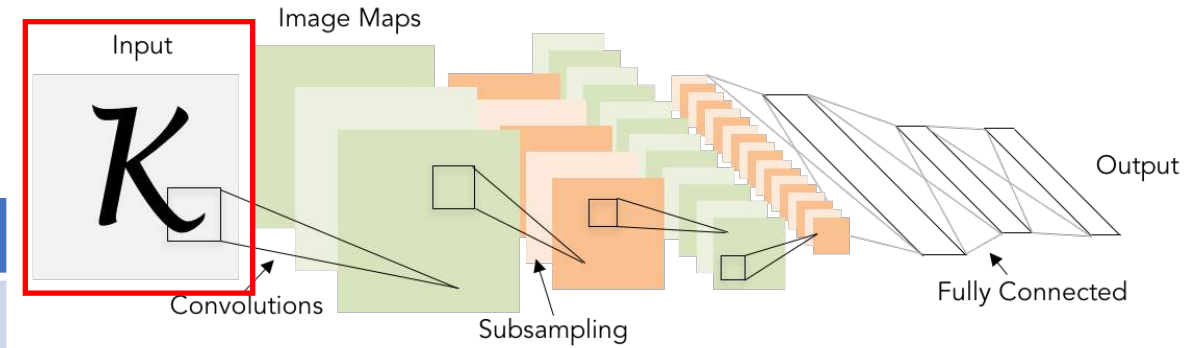| Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10

Lecun et al, "Gradient-based learning applied to document recognition", 1998
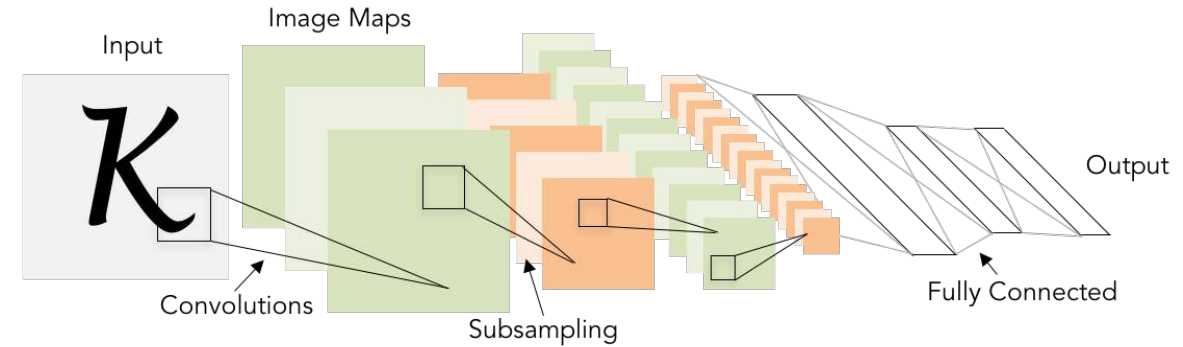
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10

As we go through the network:

Spatial size **decreases**
(using pooling or strided conv)

Number of channels **increases**
(total "volume" is preserved!)

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Problem: Deep Networks very hard to train!

Components of a Convolutional Network

Fully-Connected Layers

x   h   s

Activation Function

Convolution Layers

Pooling Layers

224x224x64

pool

112x112x64

224

downsampling

112

224

112
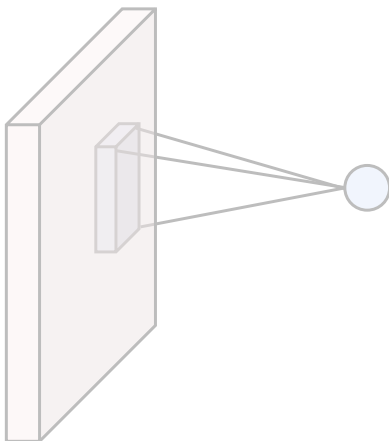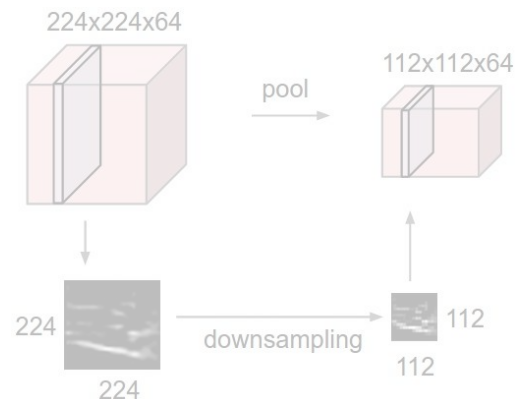
Normalization

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

Batch Normalization

Idea: "Normalize" the outputs of a layer so they have zero mean and unit variance

Why? Helps reduce "internal covariate shift", improves optimization

We can normalize a batch of activations like this:

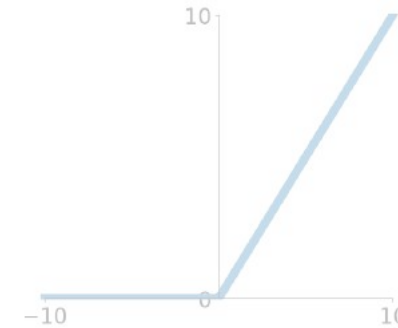$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

This is a **differentiable function**, so we can use it as an operator in our networks and backprop through it!
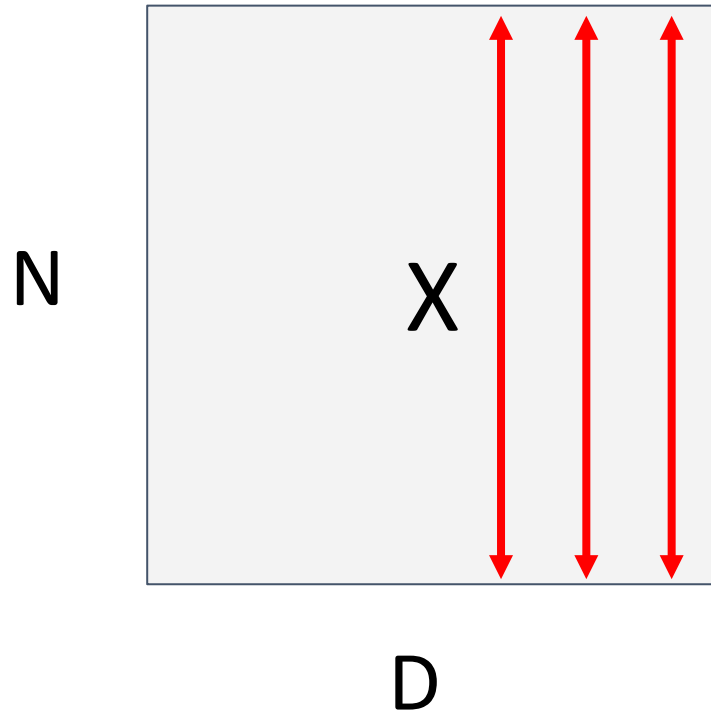
Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

Input: $x : N \times D$

N

X

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 std, 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

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

Input: $x : N \times D$

N

X

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 std, 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?

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

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

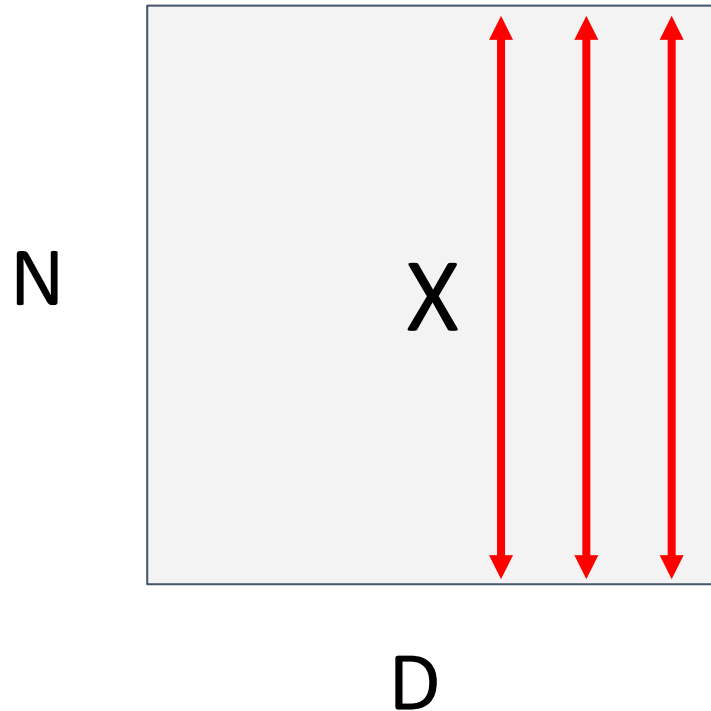$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$ Per-channel std, shape is D

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

$\mu_j =$ (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 std, 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: Test-Time

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 std, shape is D

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

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 for ConvNets

Batch Normalization for
fully-connected networks

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$\texttt{x: N × D}$$

$$\texttt{x: N×C×H×W}$$

Normalize

Normalize

$$\boldsymbol{\mu},\boldsymbol{\sigma}\texttt{: 1 × D}$$

$$\boldsymbol{\mu},\boldsymbol{\sigma}\texttt{: 1×C×1×1}$$

$$\texttt{ɣ},\beta\texttt{: 1 × D}$$

$$\texttt{ɣ},\beta\texttt{: 1×C×1×1}$$

$$\texttt{y = ɣ(x-}\boldsymbol{\mu}\texttt{)/}\boldsymbol{\sigma}\texttt{+}\beta$$

$$\texttt{y = ɣ(x-}\boldsymbol{\mu}\texttt{)/}\boldsymbol{\sigma}\texttt{+}\beta$$

Batch Normalization

FC

BN

tanh

FC

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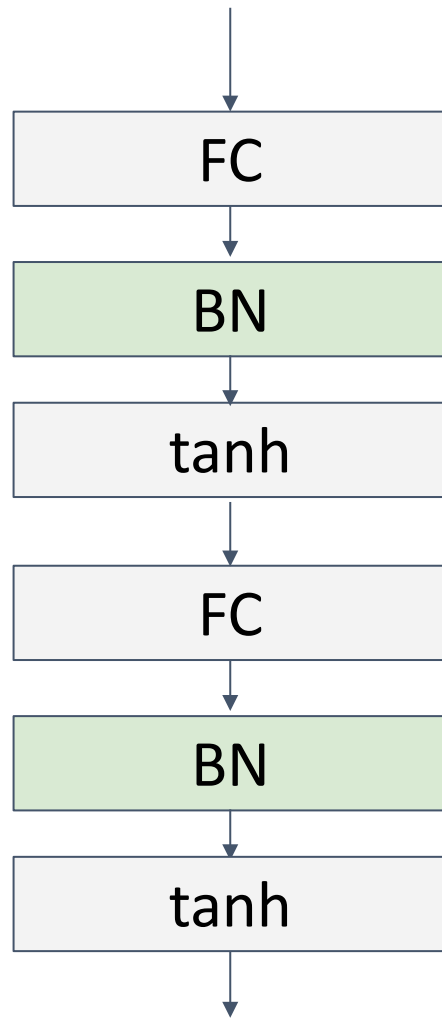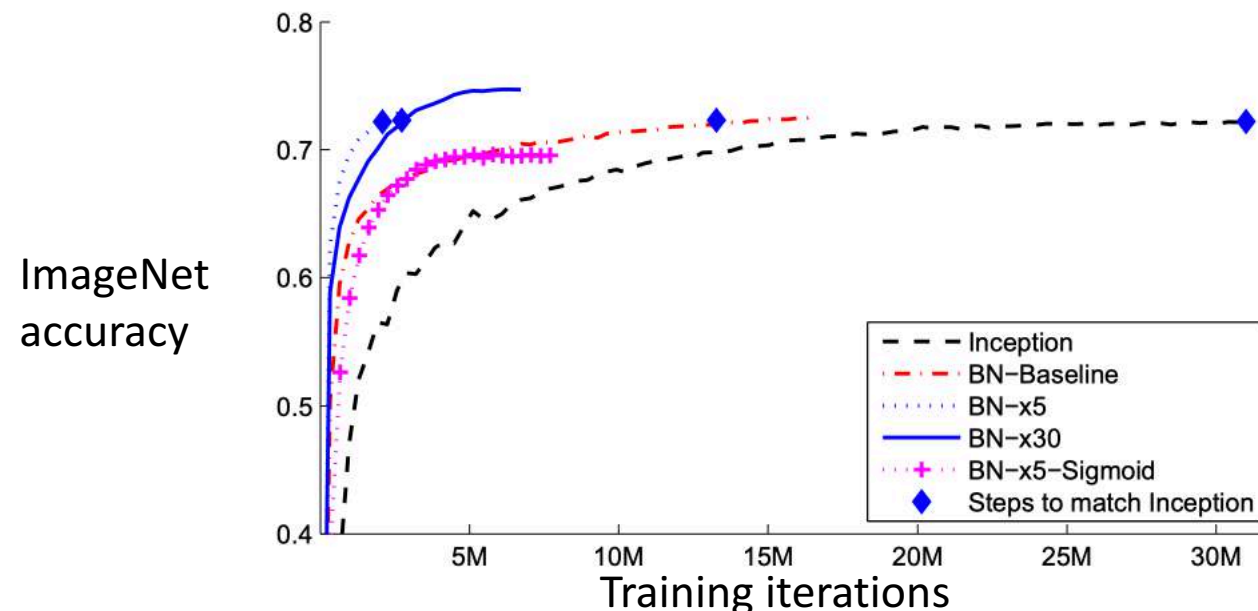
tanh
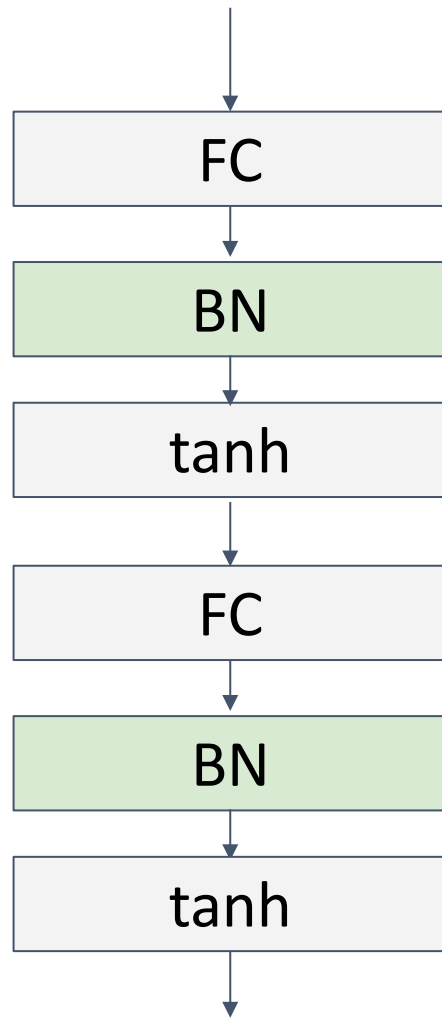
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

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

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015
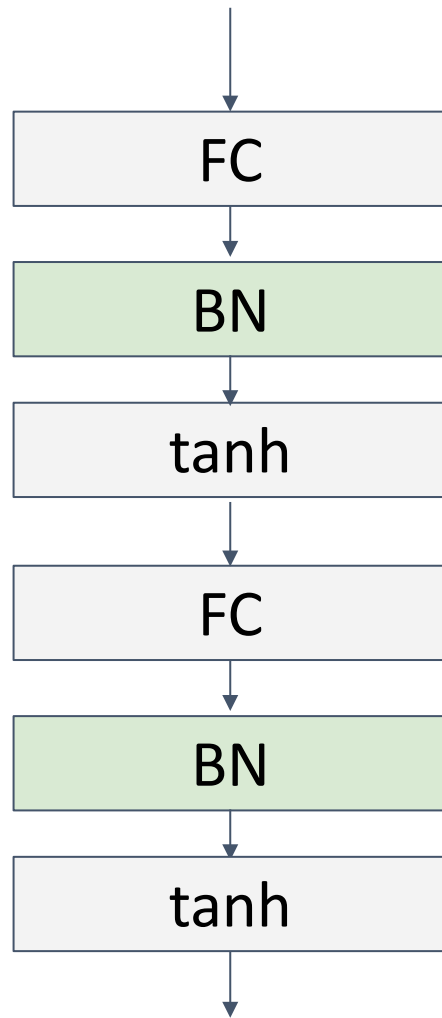
Batch Normalization

FC

BN

tanh

FC

BN

tanh

- Makes deep networks **much** easier to train!
- 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!

ImageNet accuracy

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

FC

BN

tanh

FC

BN

tanh

- Makes deep networks **much** easier to train!
- 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!
- Not well-understood theoretically (yet)
- Behaves differently during training and testing: this is a very common source of bugs!

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Layer Normalization

Layer Normalization for fully-connected networks
Same behavior at train and test!
Used in RNNs, Transformers

Batch Normalization for fully-connected networks

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

Normalize

$$\boldsymbol{\mu},\boldsymbol{\sigma}: \texttt{1} \times \texttt{D}$$

$$\texttt{ɣ},\beta: \texttt{1} \times \texttt{D}$$

$$\texttt{y} = \texttt{ɣ(x-}\boldsymbol{\mu}\texttt{)/}\boldsymbol{\sigma}\texttt{+}\beta$$

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

Normalize

$$\boldsymbol{\mu},\boldsymbol{\sigma}: \texttt{N} \times \texttt{1}$$

$$\texttt{ɣ},\beta: \texttt{1} \times \texttt{D}$$

$$\texttt{y} = \texttt{ɣ(x-}\boldsymbol{\mu}\texttt{)/}\boldsymbol{\sigma}\texttt{+}\beta$$

Ba, Kiros, and Hinton, "Layer Normalization", arXiv 2016

Instance Normalization

Batch Normalization for convolutional networks

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

Normalize

$$x: \texttt{N×C×H×W}$$

$$\mu, \sigma: \texttt{1×C×1×1}$$

$$\gamma, \beta: \texttt{1×C×1×1}$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Normalize

$$x: \texttt{N×C×H×W}$$

$$\mu, \sigma: \texttt{N×C×1×1}$$

$$\gamma, \beta: \texttt{1×C×1×1}$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

Comparison of Normalization Layers
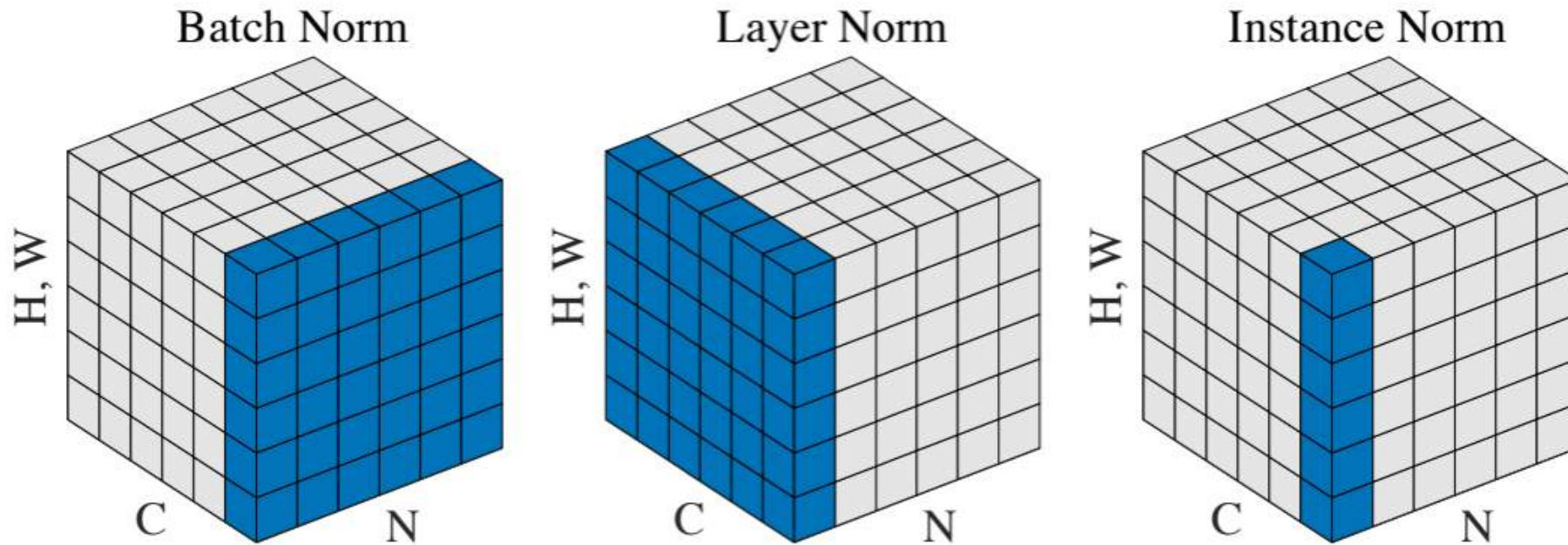
Wu and He, "Group Normalization", ECCV 2018
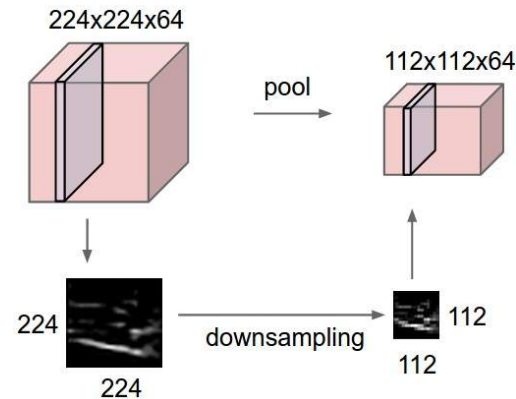
Group Normalization
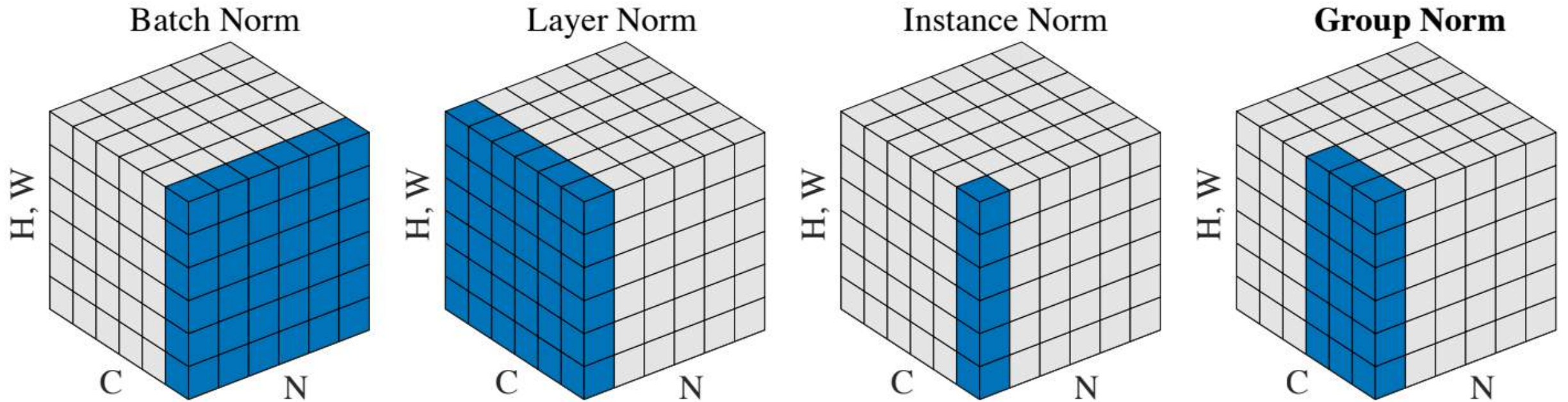
Wu and He, "Group Normalization", ECCV 2018
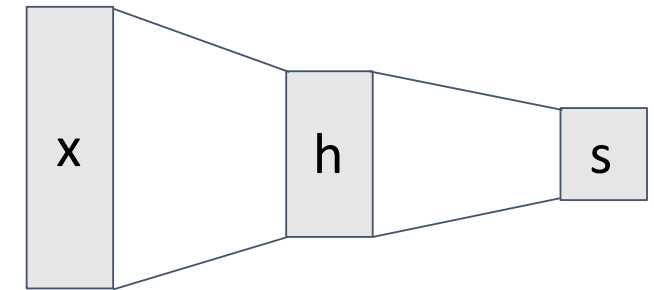
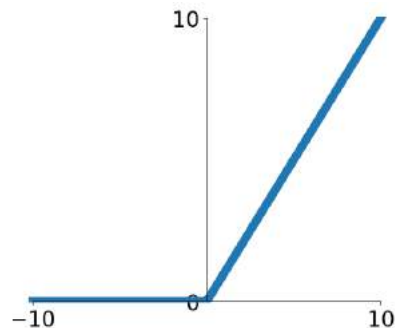# Components of a Convolutional Network

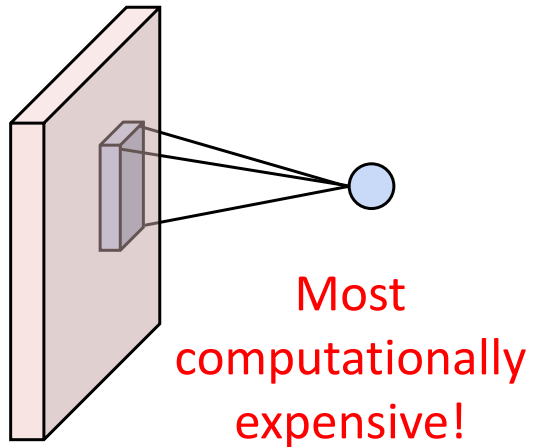**Convolution Layers**
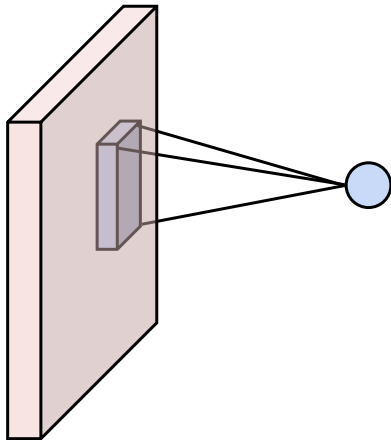
Pooling Layers

Fully-Connected Layers

224x224x64

pool

112x112x64

224

downsampling

112

224

112

x

h
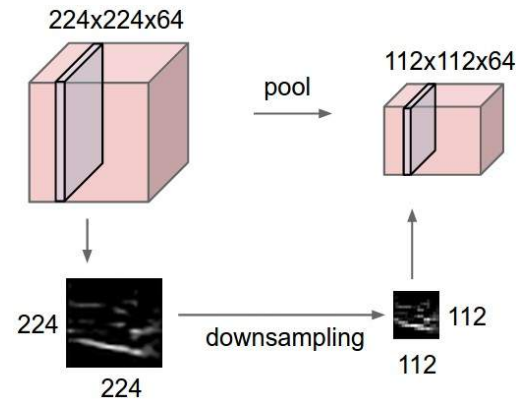
s

Activation Function

10

−10

0

10

Normalization

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

Components of a Convolutional Network

Convolution Layers
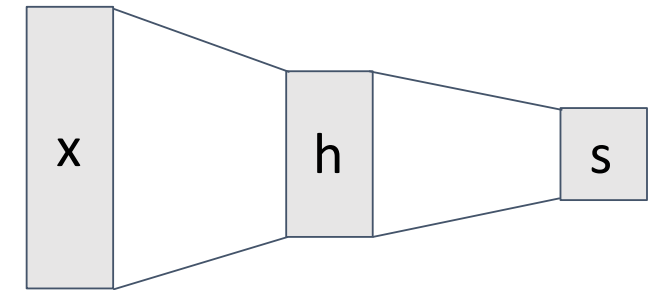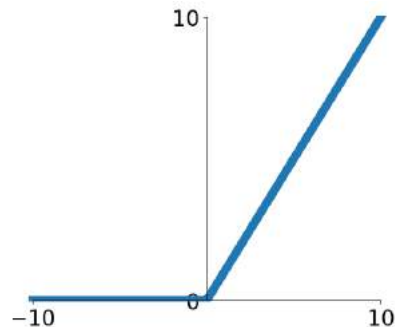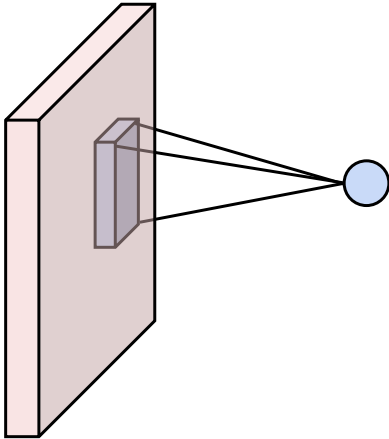
Most computationally expensive!

Pooling Layers

224x224x64

pool

112x112x64

224

224

downsampling

112

112

Fully-Connected Layers

x   h   s

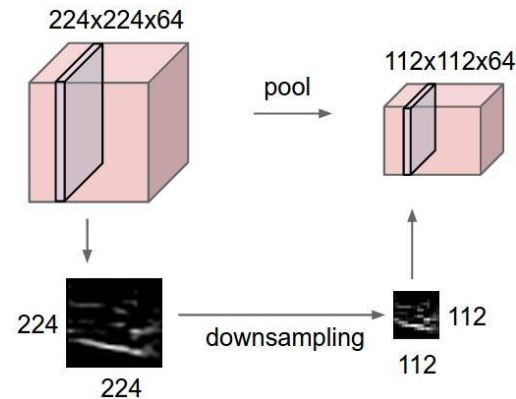**Activation Function**

10

-10   0   10

Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$
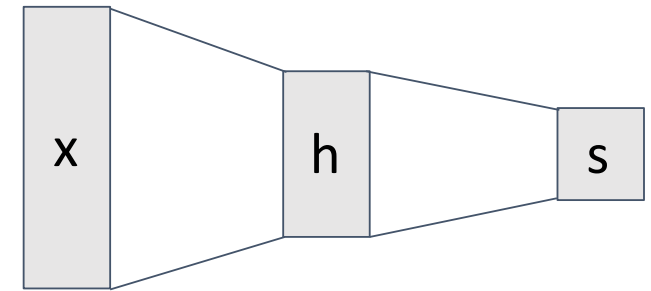
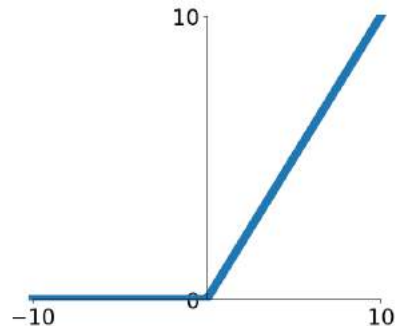# Summary: Components of a Convolutional Network

Convolution Layers

Pooling Layers

224x224x64

pool

112x112x64

224

downsampling

112

224

112

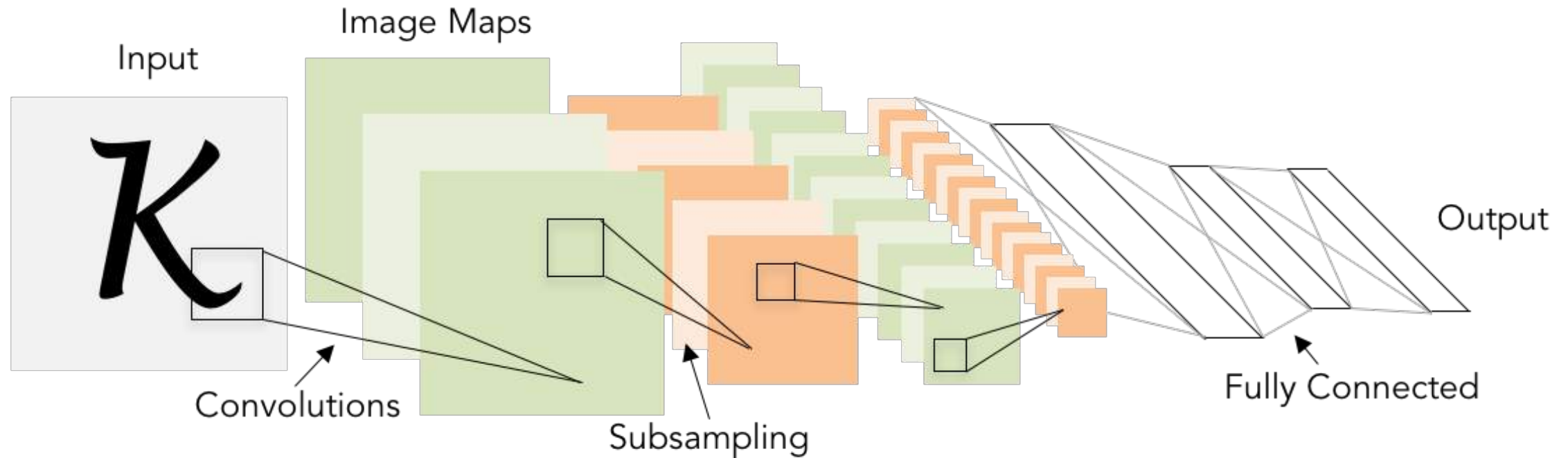**Fully-Connected Layers**

x

h

s

Activation Function

Normalization

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

Summary: Components of a Convolutional Network

Problem: What is the right way to combine all these components?

Next time:
CNN Architectures