DeepRob

Lecture 7
Convolutional Neural Networks
University of Michigan and University of Minnesota

Project 1—Reminder

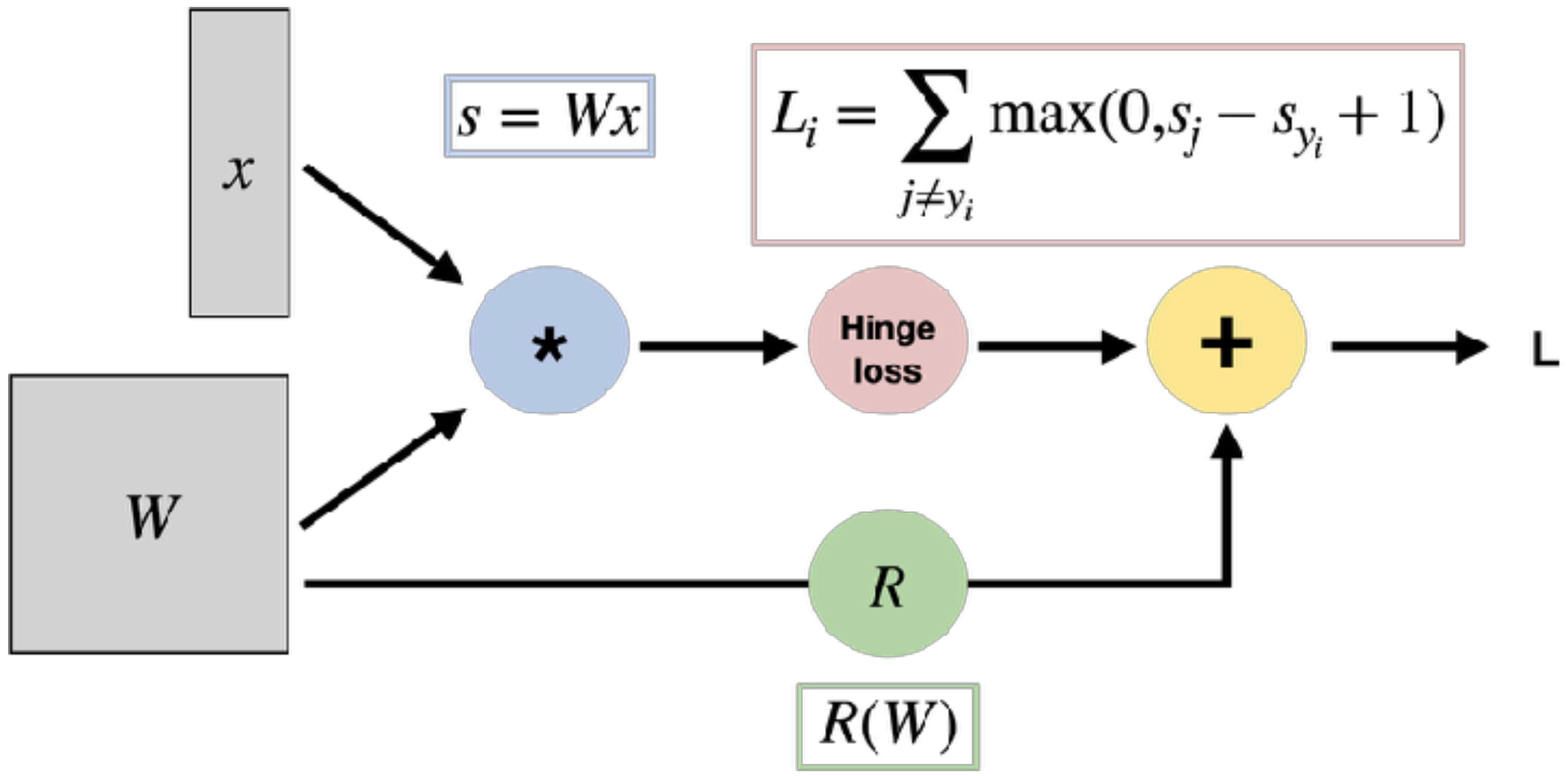
- Instructions and code available on the website
  - Here: https://rpm-lab.github.io/CSCI5980-Spr23-DeepRob/projects/project1/
- Uses Python, PyTorch and Google Colab
- Implement KNN, linear SVM, and linear softmax classifiers
- **Autograder is online!**
- **Due ~~Tuesday, February 7th~~, Thursday, February 9th 11:59 PM CT**

Quiz 3 was today!

- Quiz 4 will be on Thursday Feb 9th!

Recap from Previous Lecture

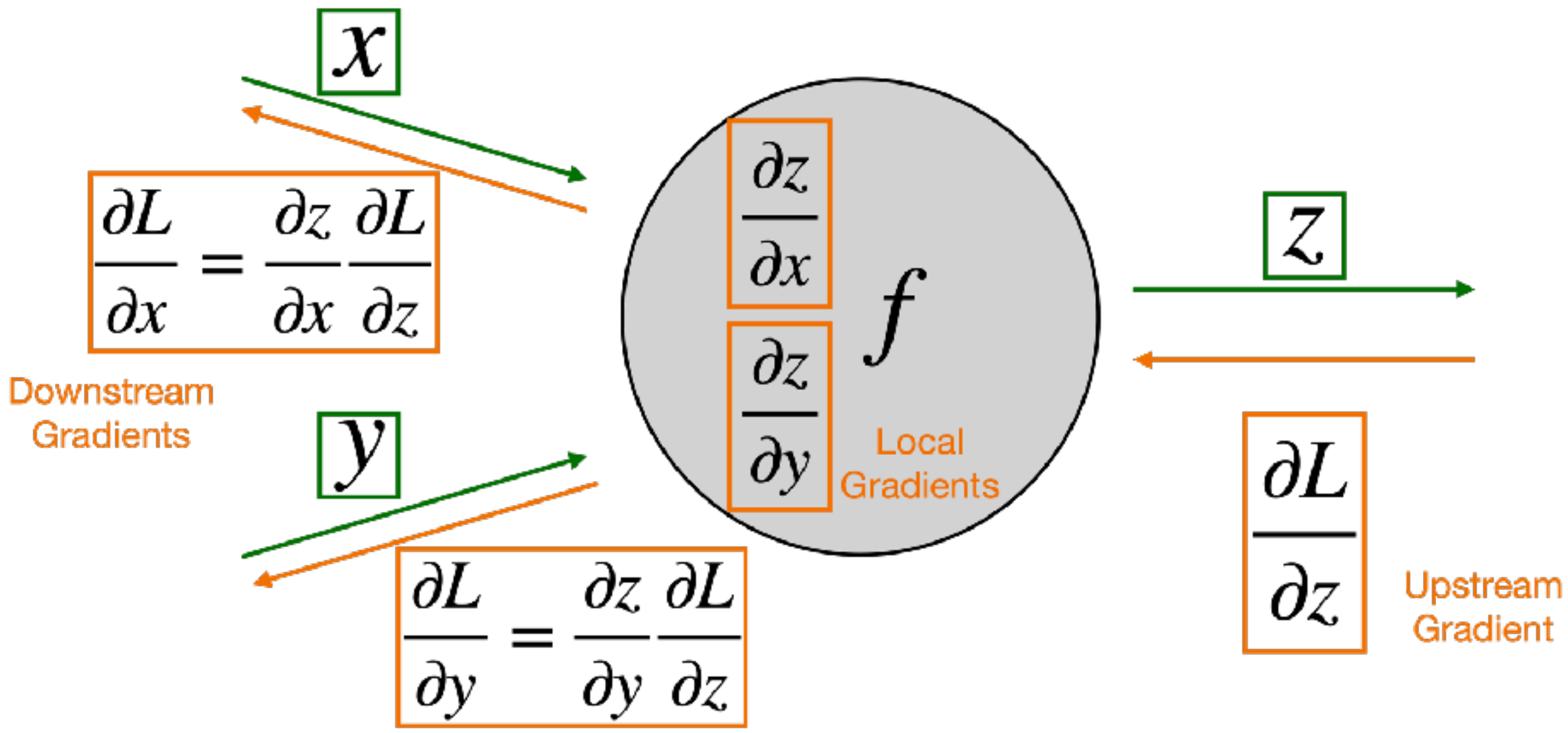
Represent complex expressions as **computational graphs**



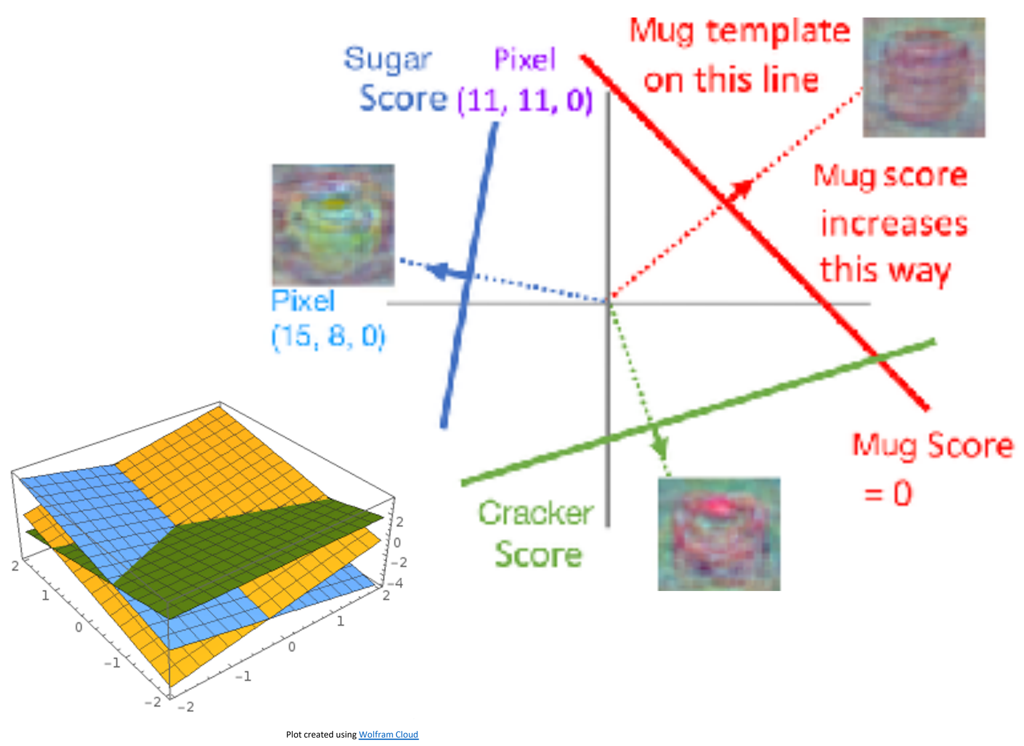
1. **Forward pass**: Compute outputs

2. **Backward pass**: Compute gradients

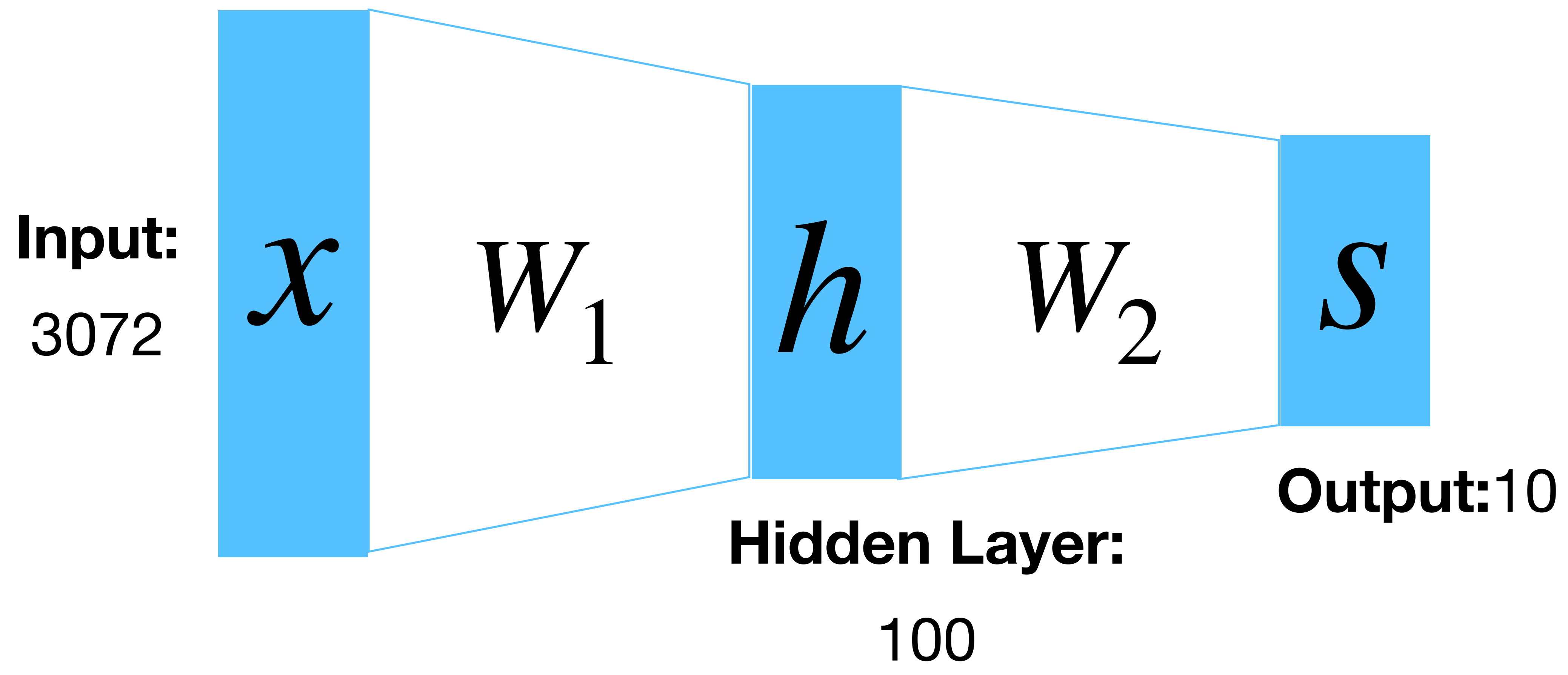
During the backward pass, each node in the graph receives **upstream gradients** and multiplies them by **local gradients** to compute **downstream gradients**



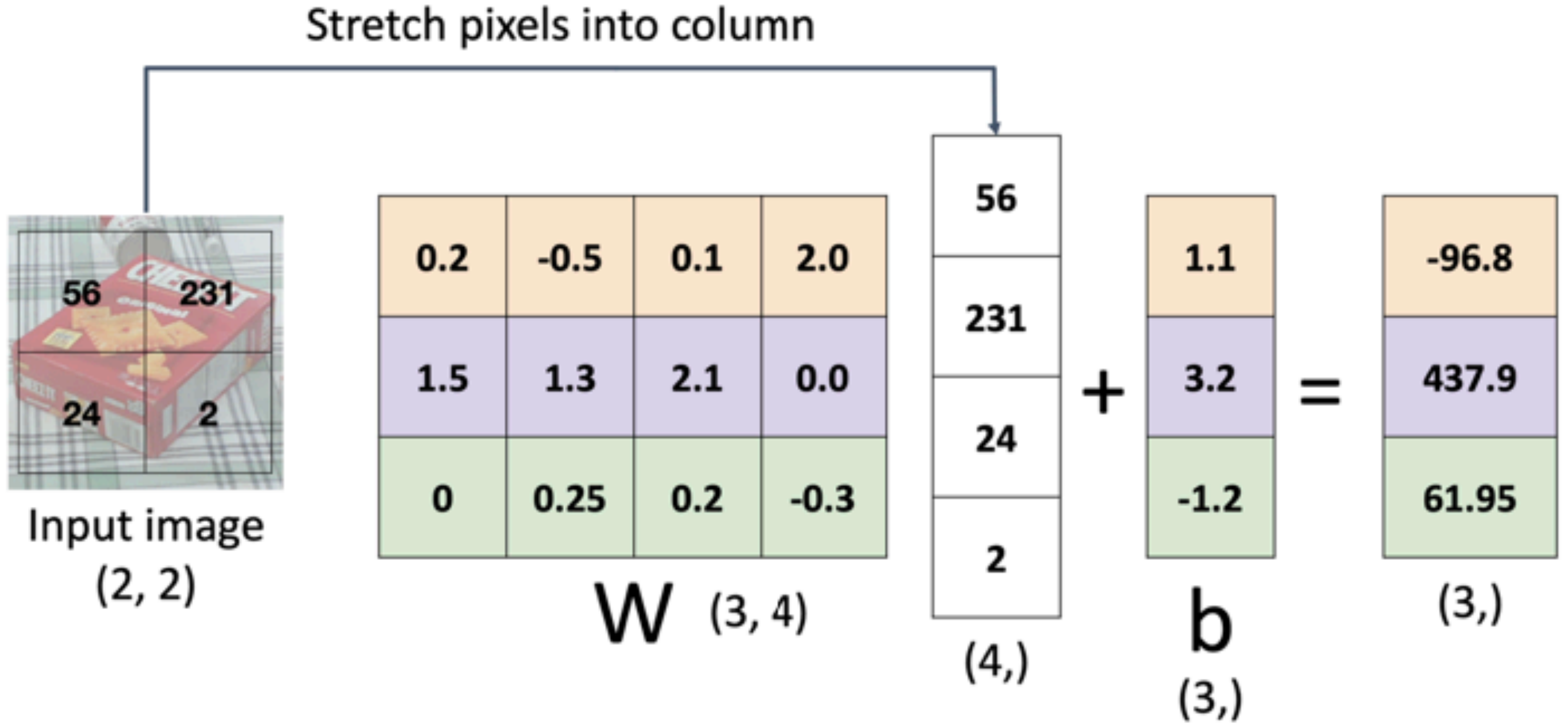
Recap from Previous Lecture



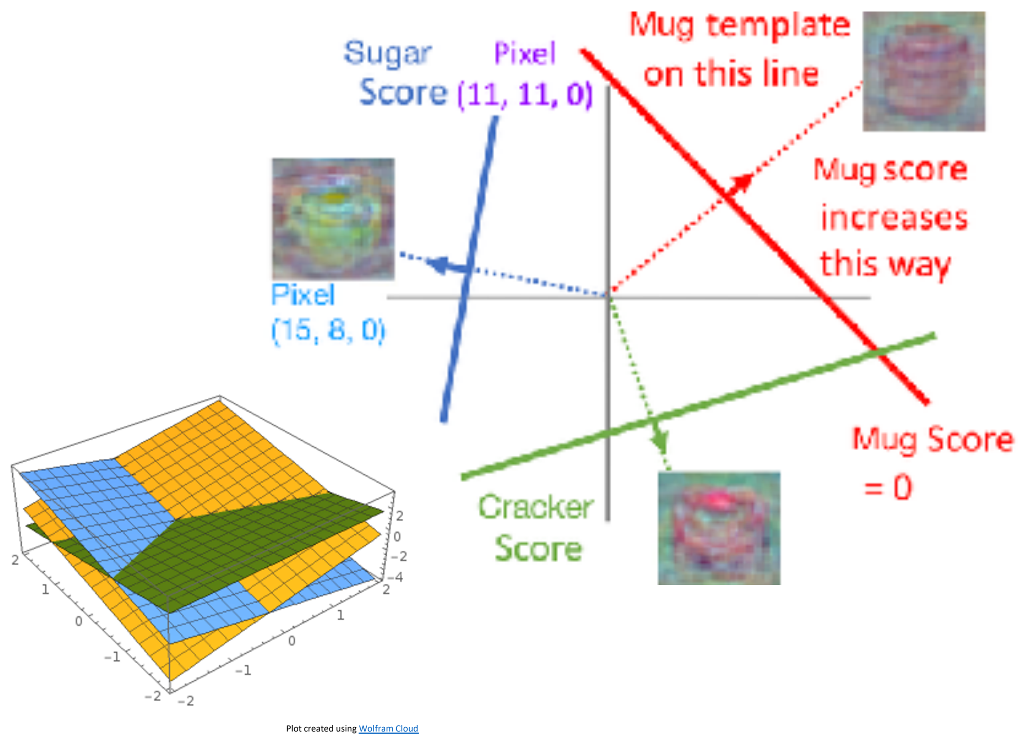
$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



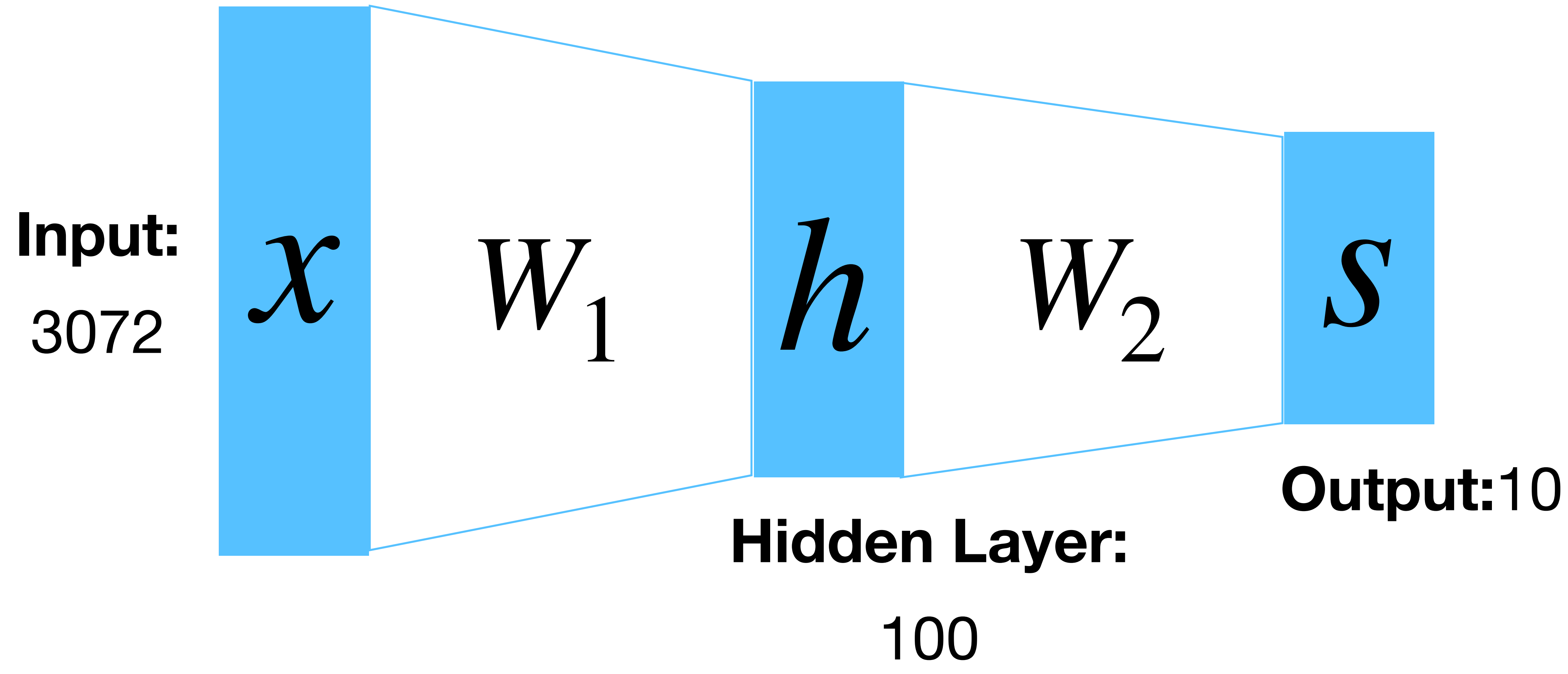
Problem: So far our classifiers don't respect the spatial structure of images!



Recap from Previous Lecture

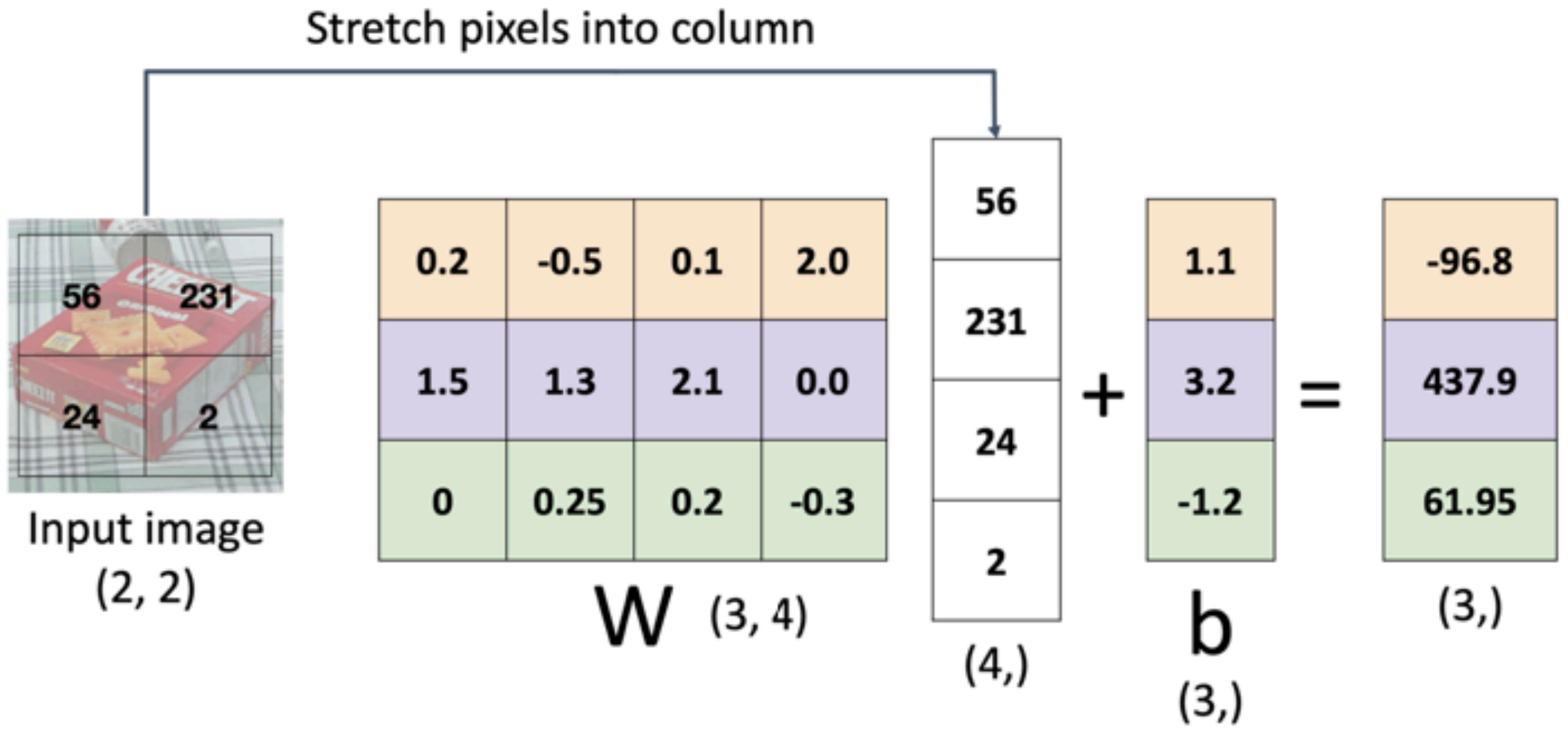


$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



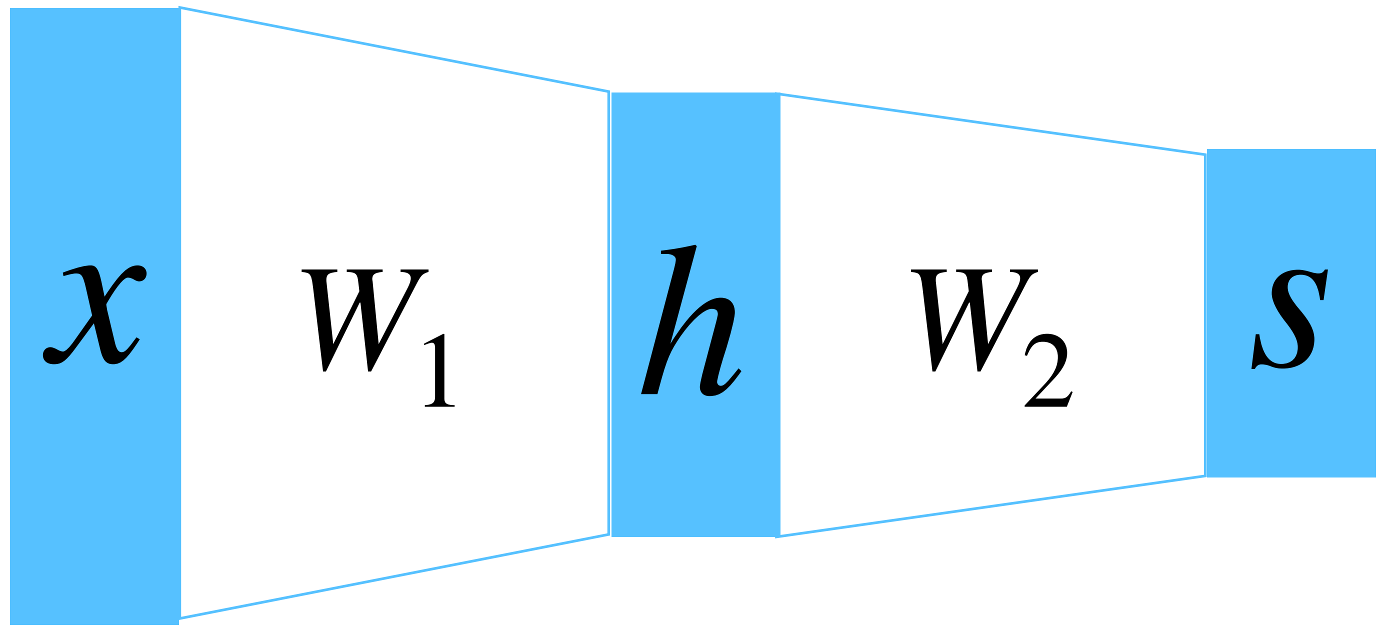
Problem: So far our classifiers don't respect the spatial structure of images!

Solution: Define new computational nodes that operate on images!

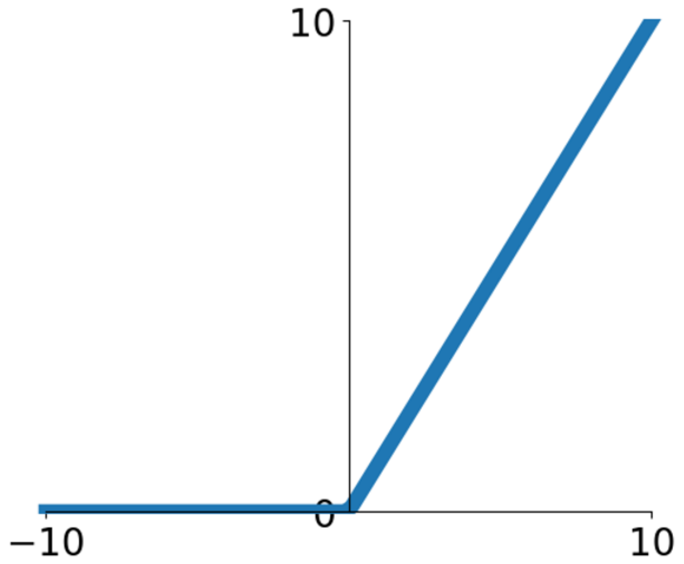


Components of Fully-Connected Networks

Fully-Connected Layers

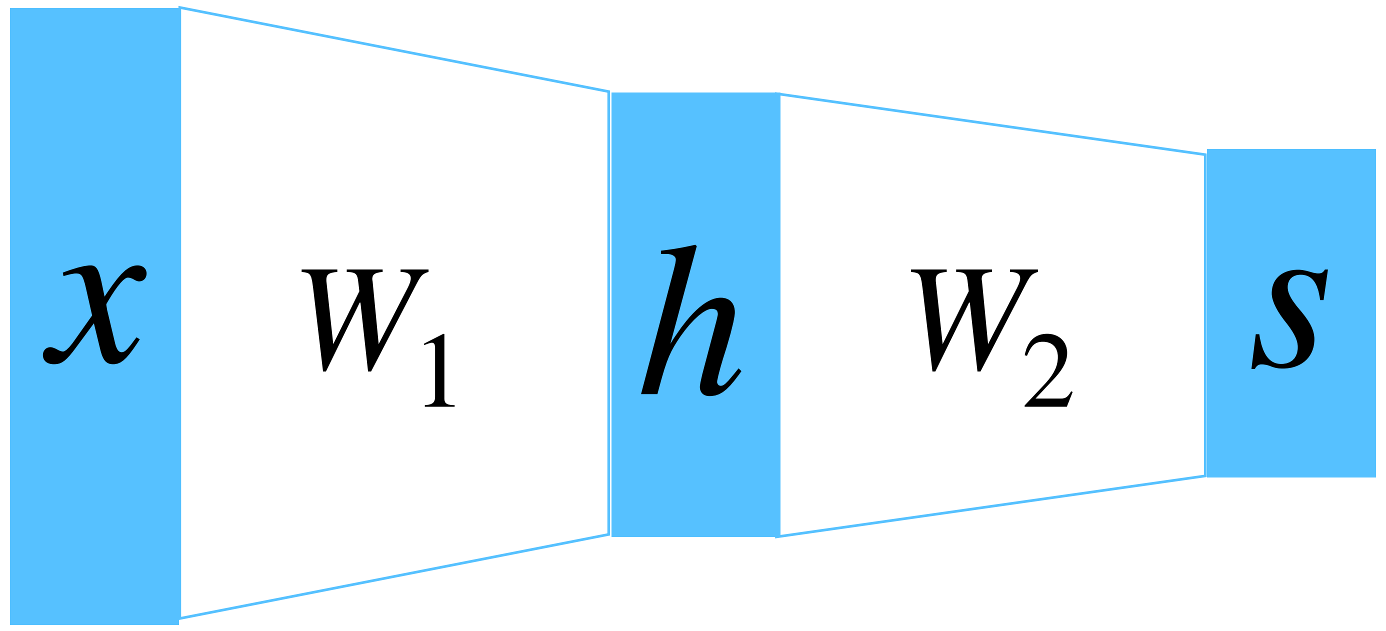


Activation Functions

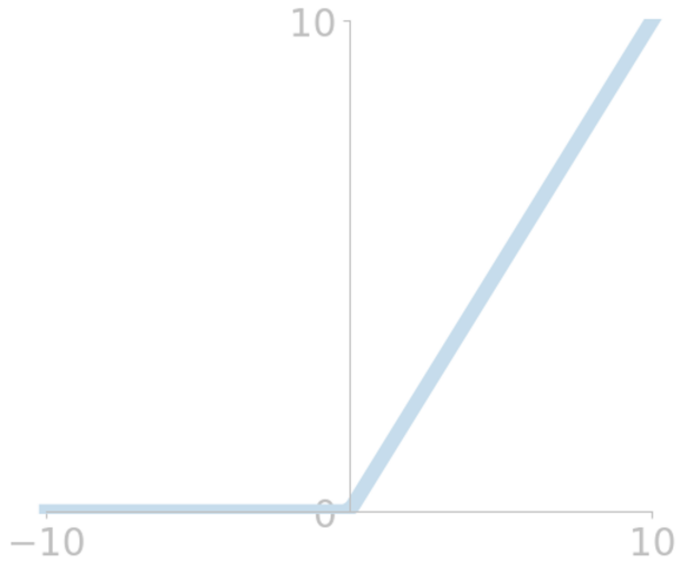


Components of Convolutional Neural Networks

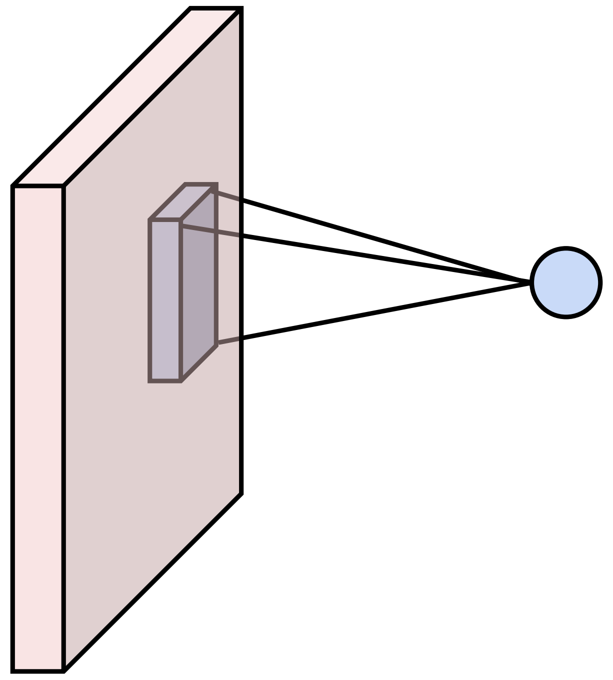
Fully-Connected Layers



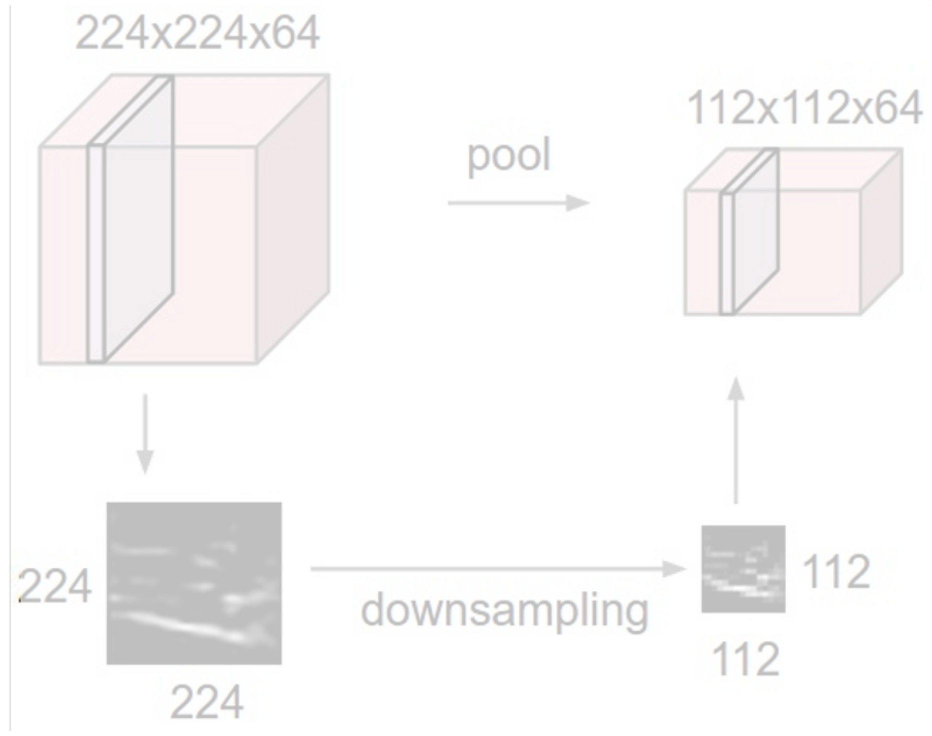
Activation Functions



Convolution Layers



Pooling Layers

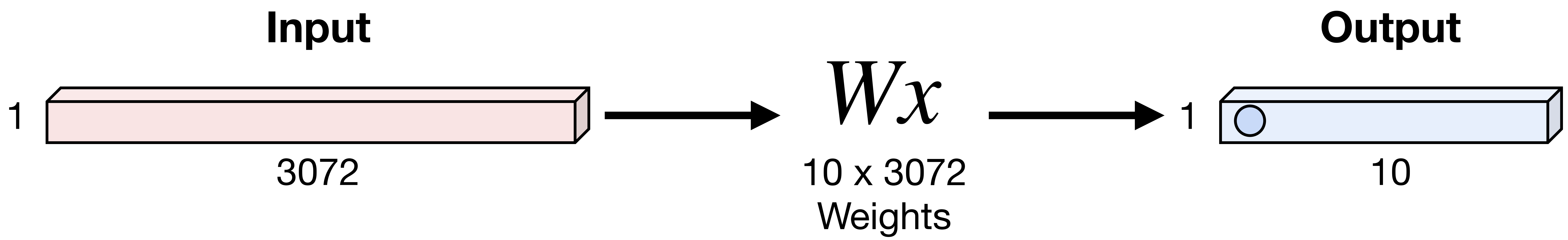


Normalization

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

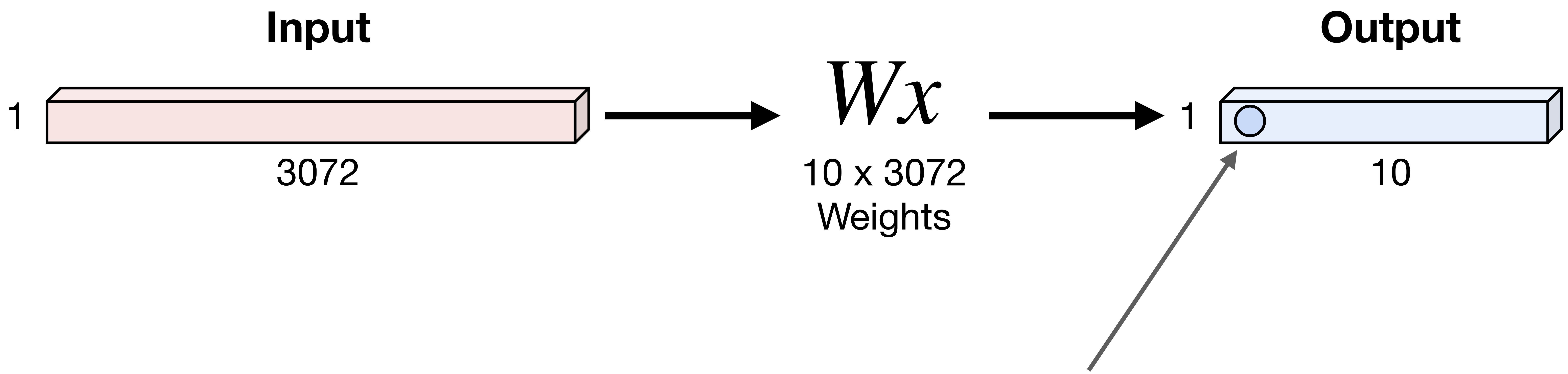
Fully-Connected Layer

3x32x32 image → stretch to 3072x1



Fully-Connected Layer

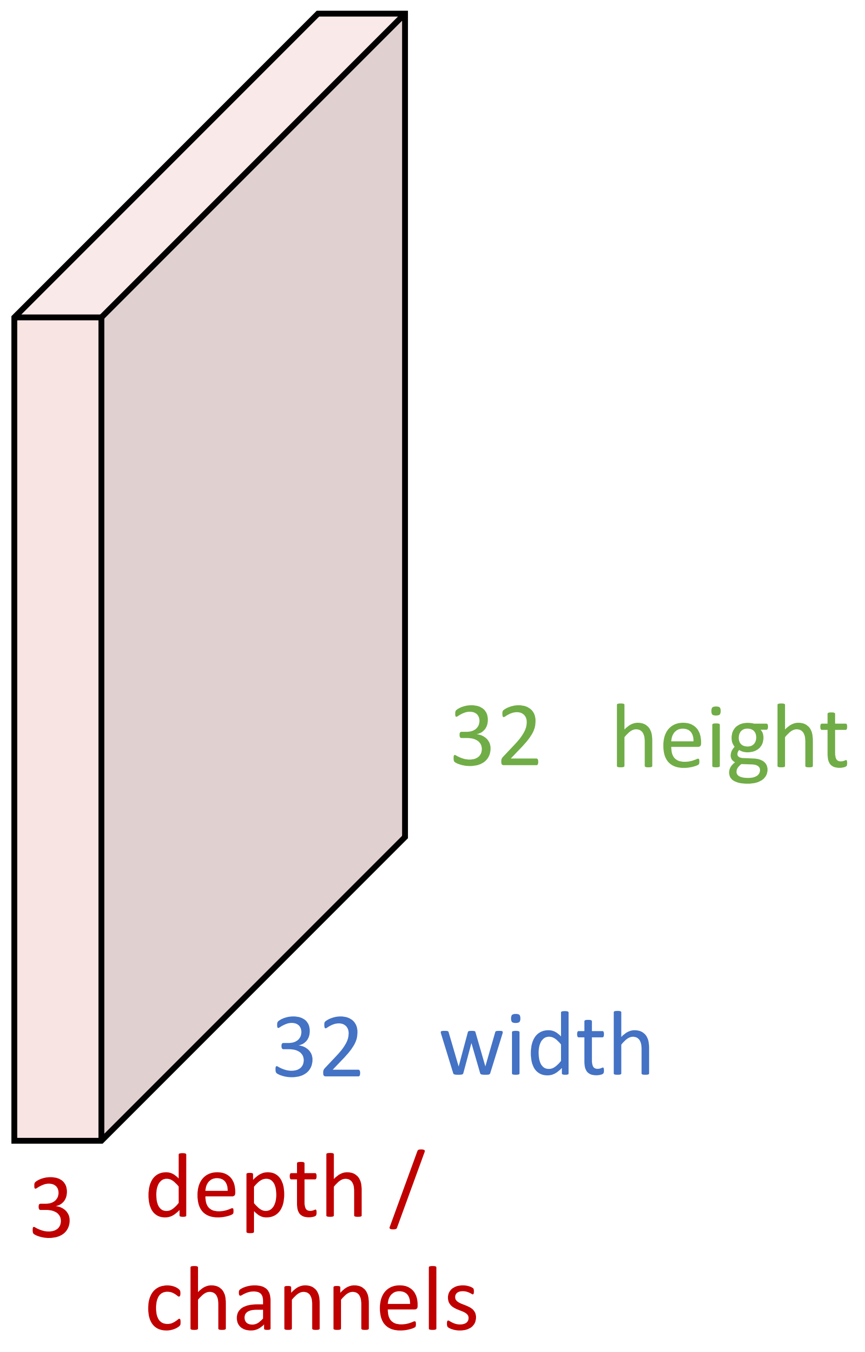
3x32x32 image → stretch to 3072x1



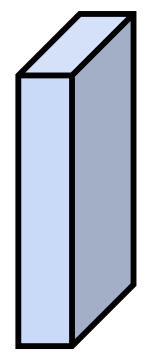
1 number:
The result of taking a dot product between a row of W and the input

Convolution Layer

3x32x32 image: preserve spatial structure



3x5x5 filter



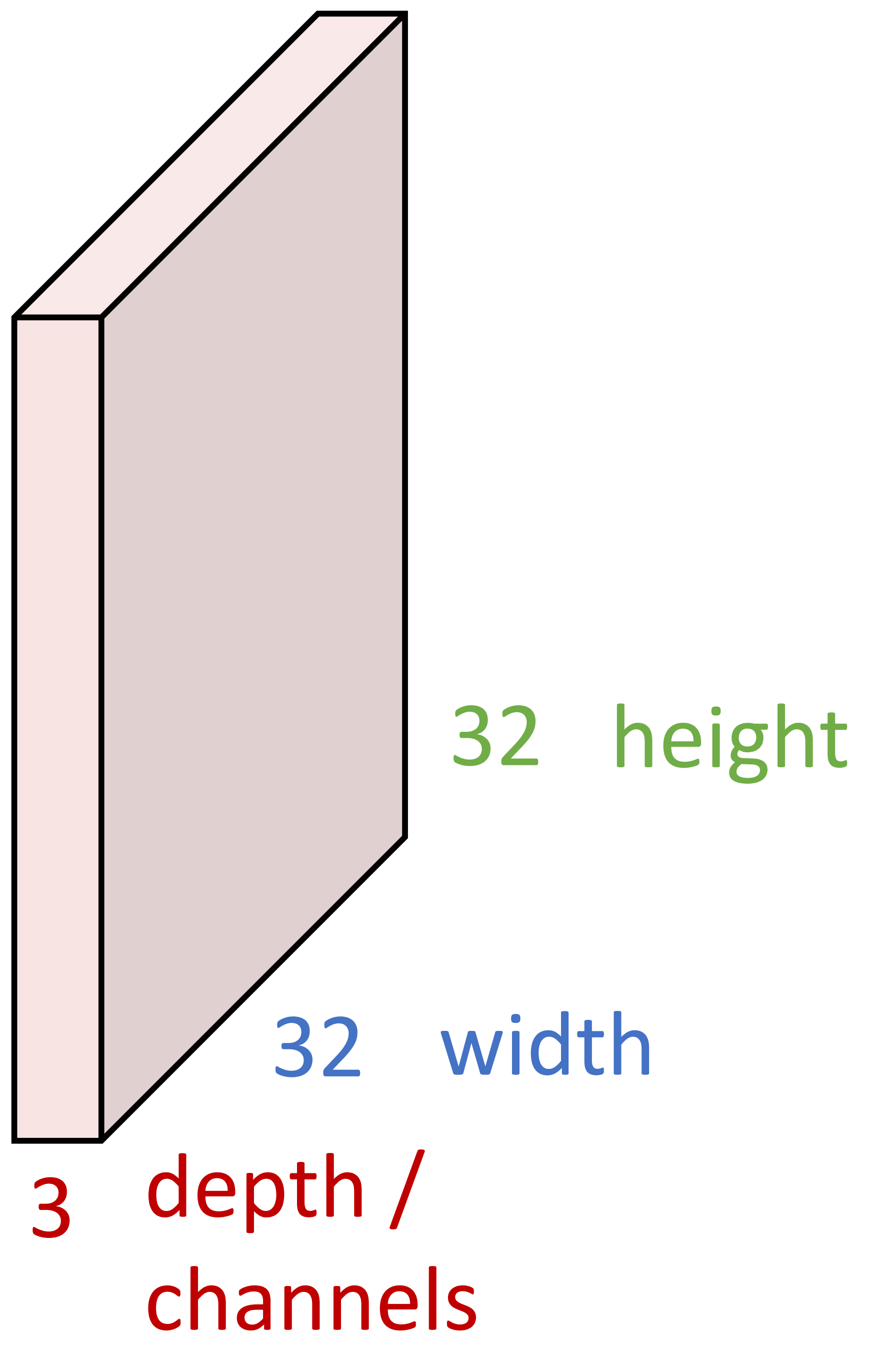
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

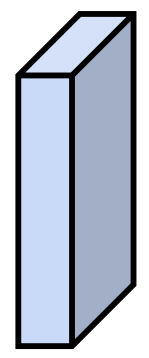
Convolution Layer

3x32x32 image

Filters always extend the full depth of the input volume

3x5x5 filter

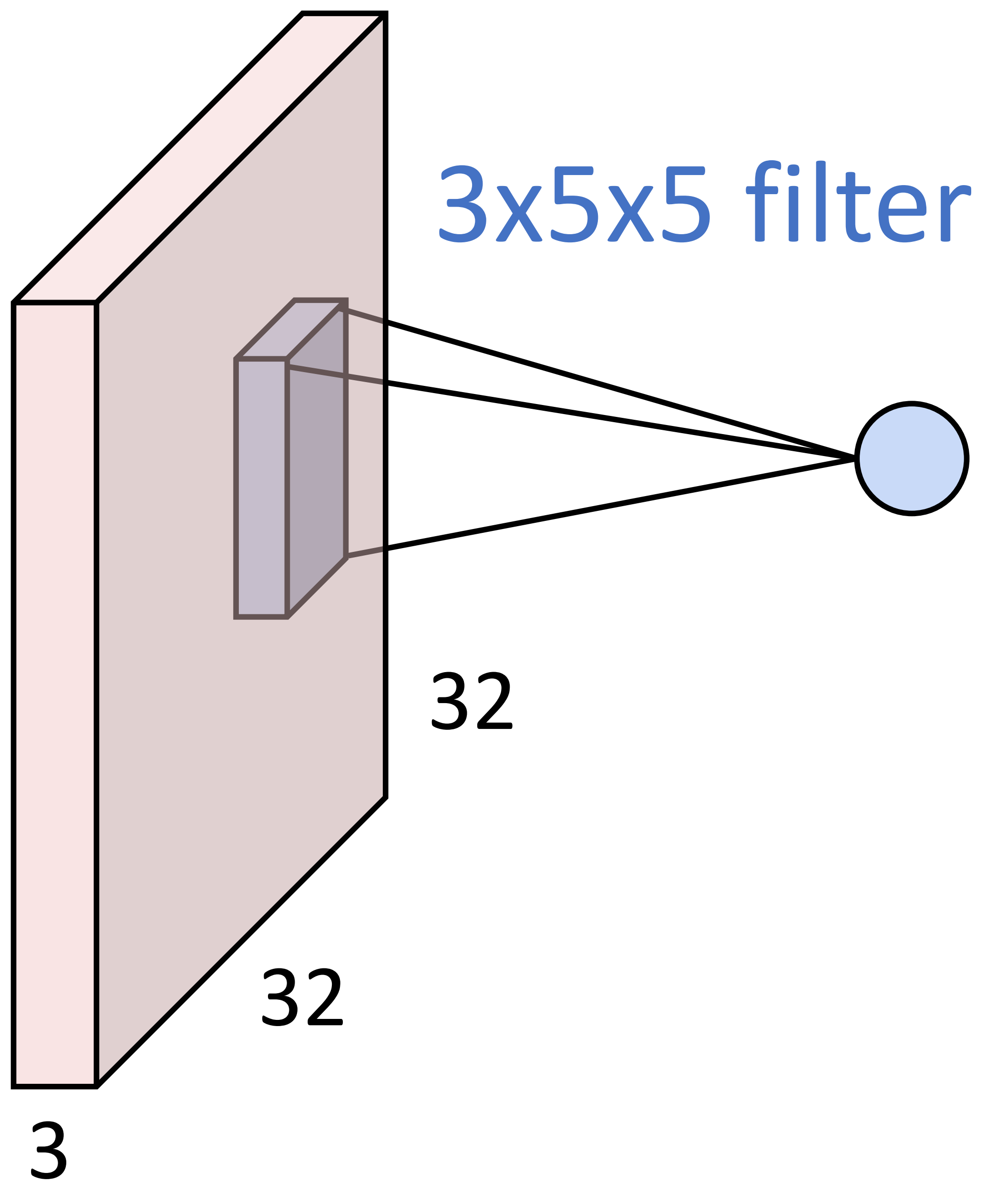




Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Convolution Layer

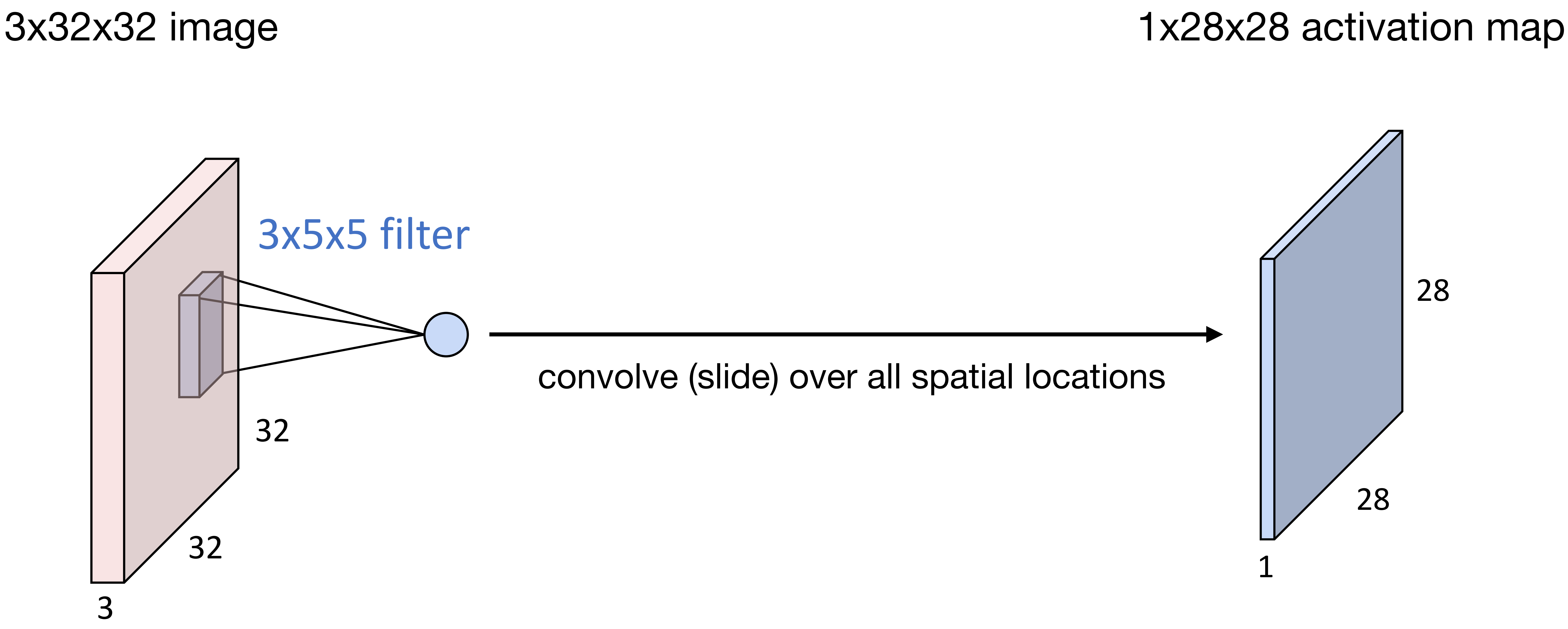
3x32x32 image



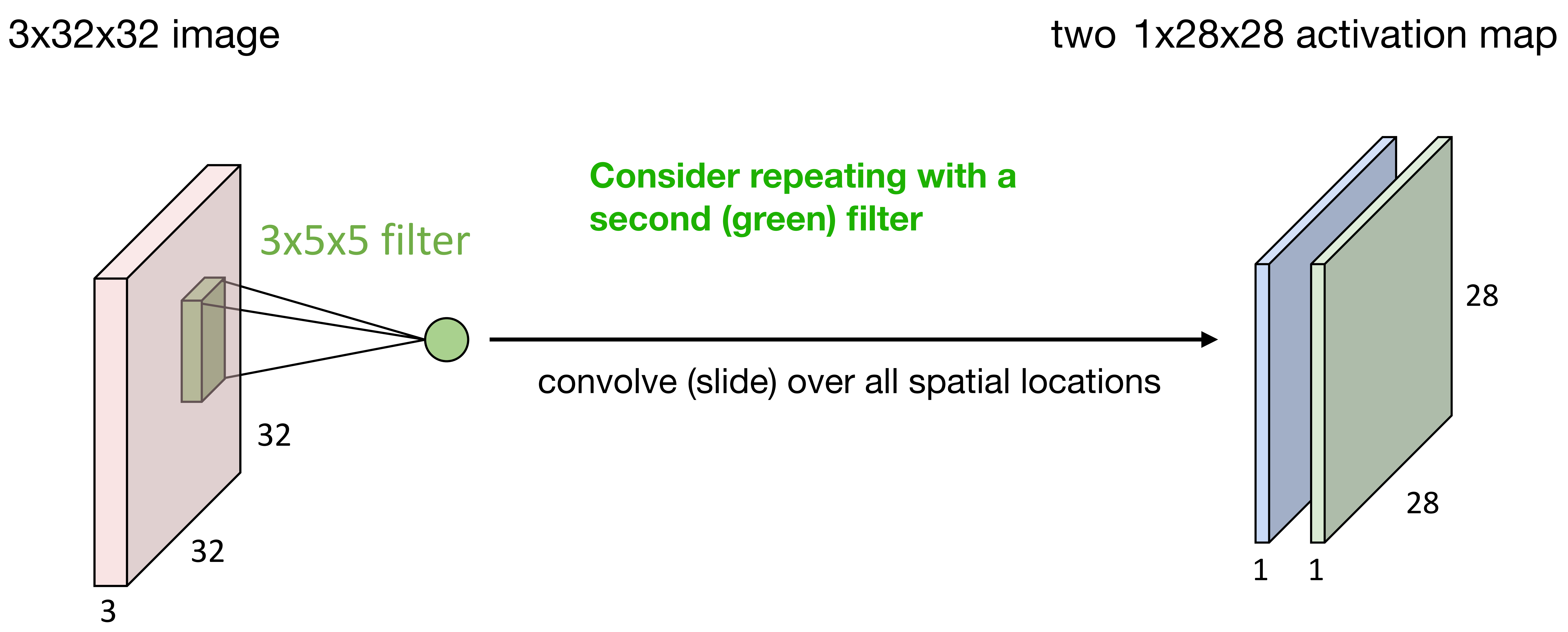
1 number:
The result of taking a dot product between the filter and a small 3x5x5 portion of the image
(i.e. 3*5*5=75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer



Convolution Layer

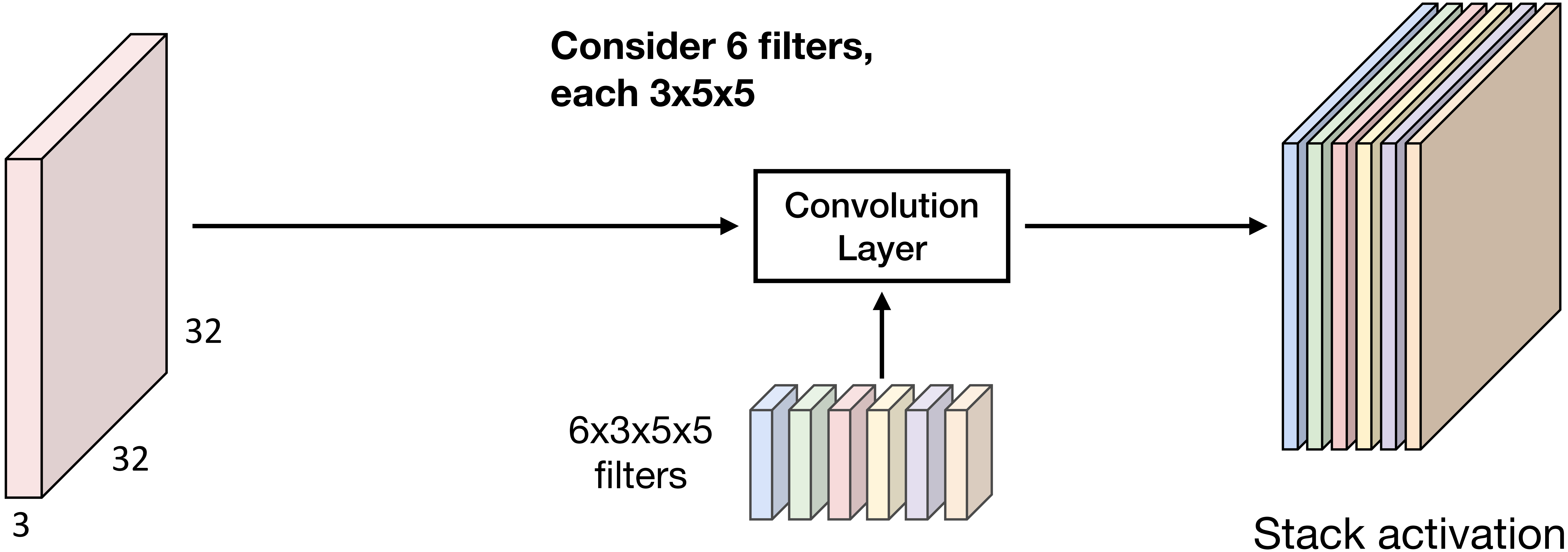


Convolution Layer

3x32x32 image

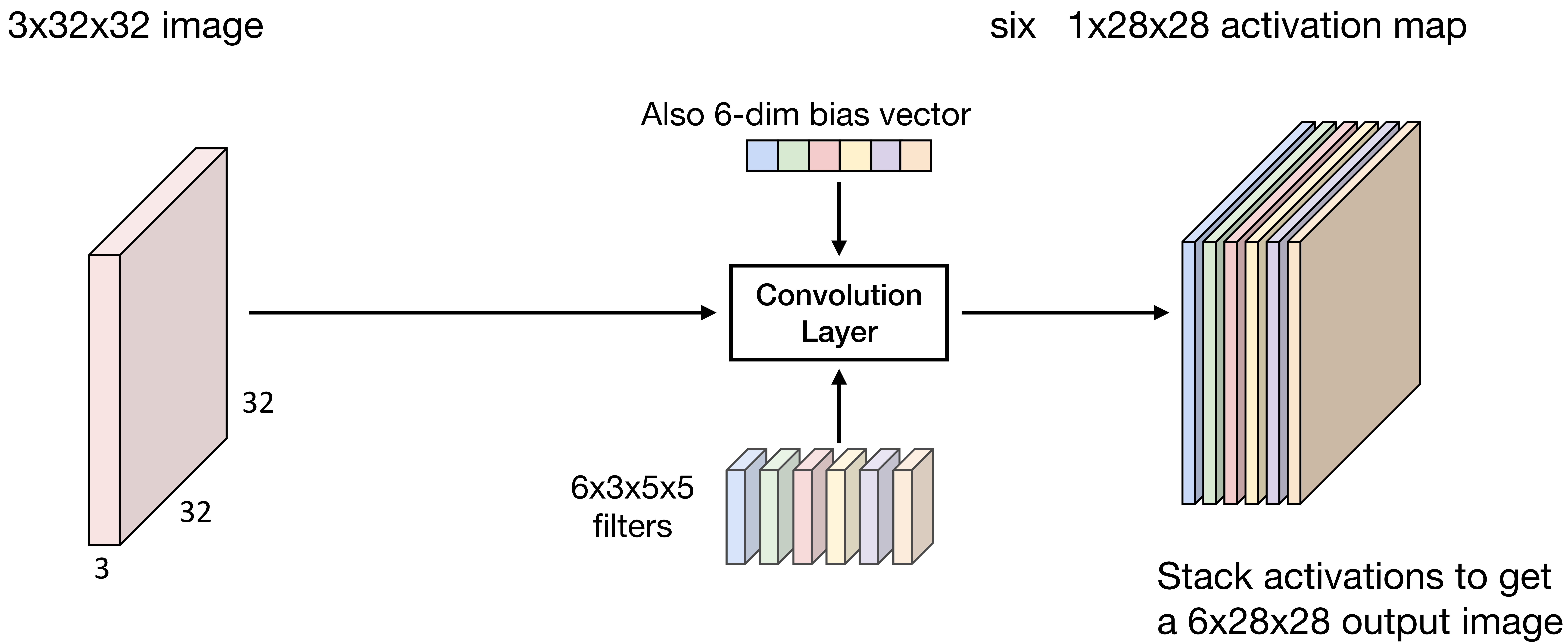
six 1x28x28 activation map

Consider 6 filters, each 3x5x5

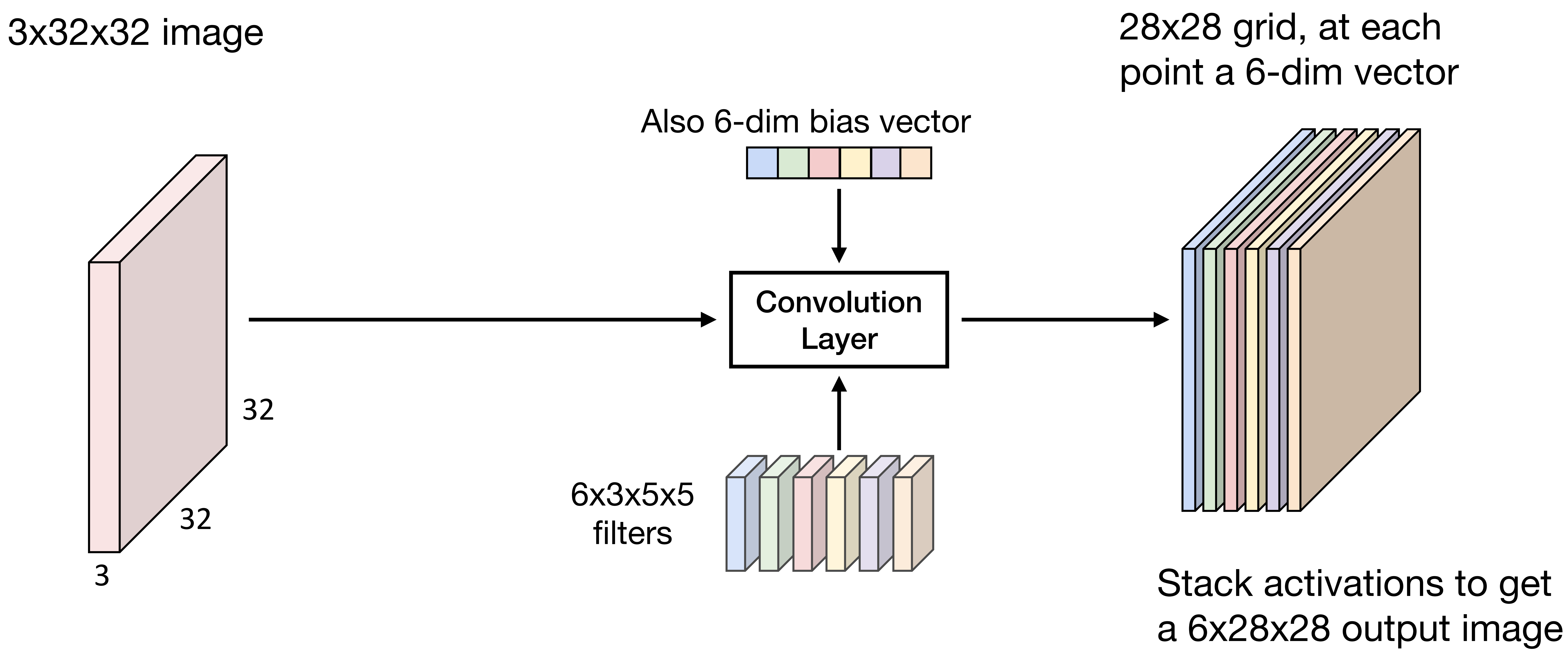


Stack activations to get a 6x28x28 output image

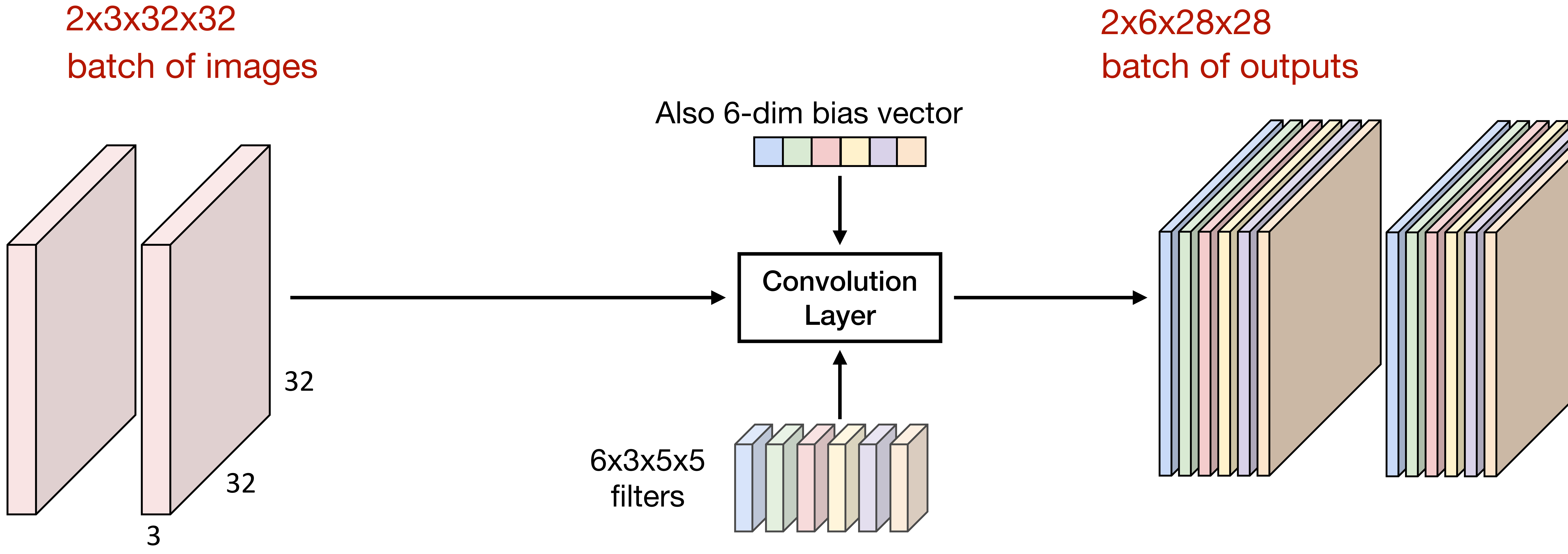
Convolution Layer



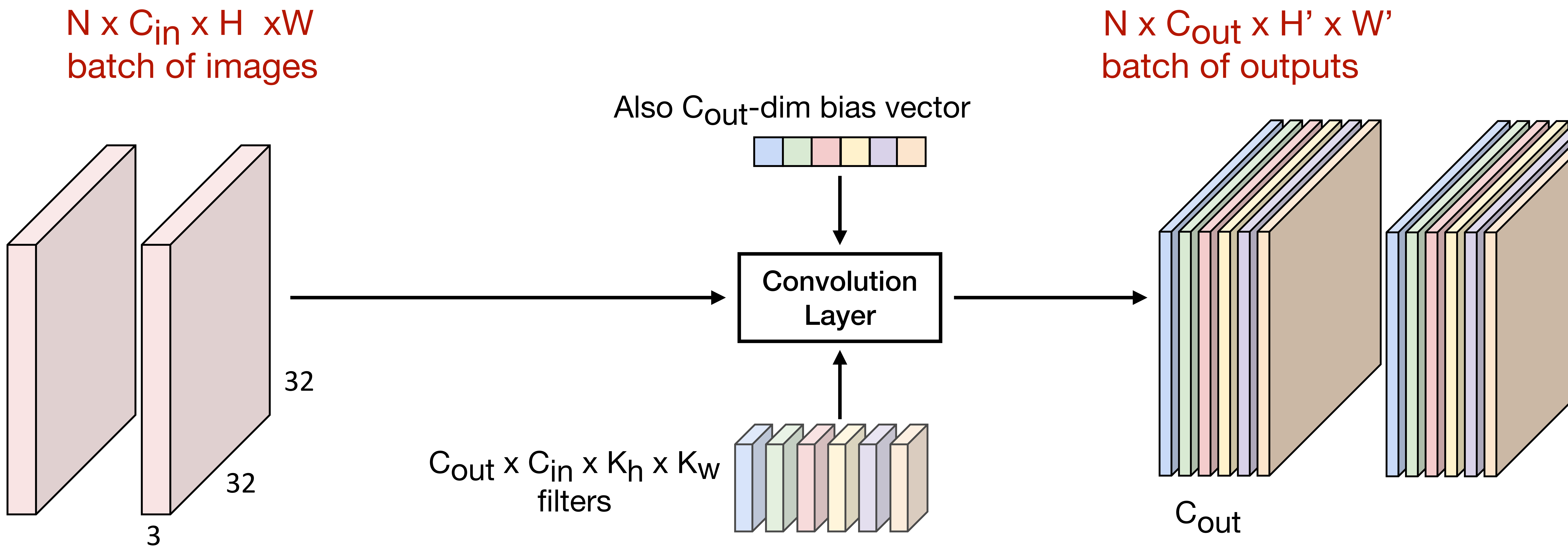
Convolution Layer



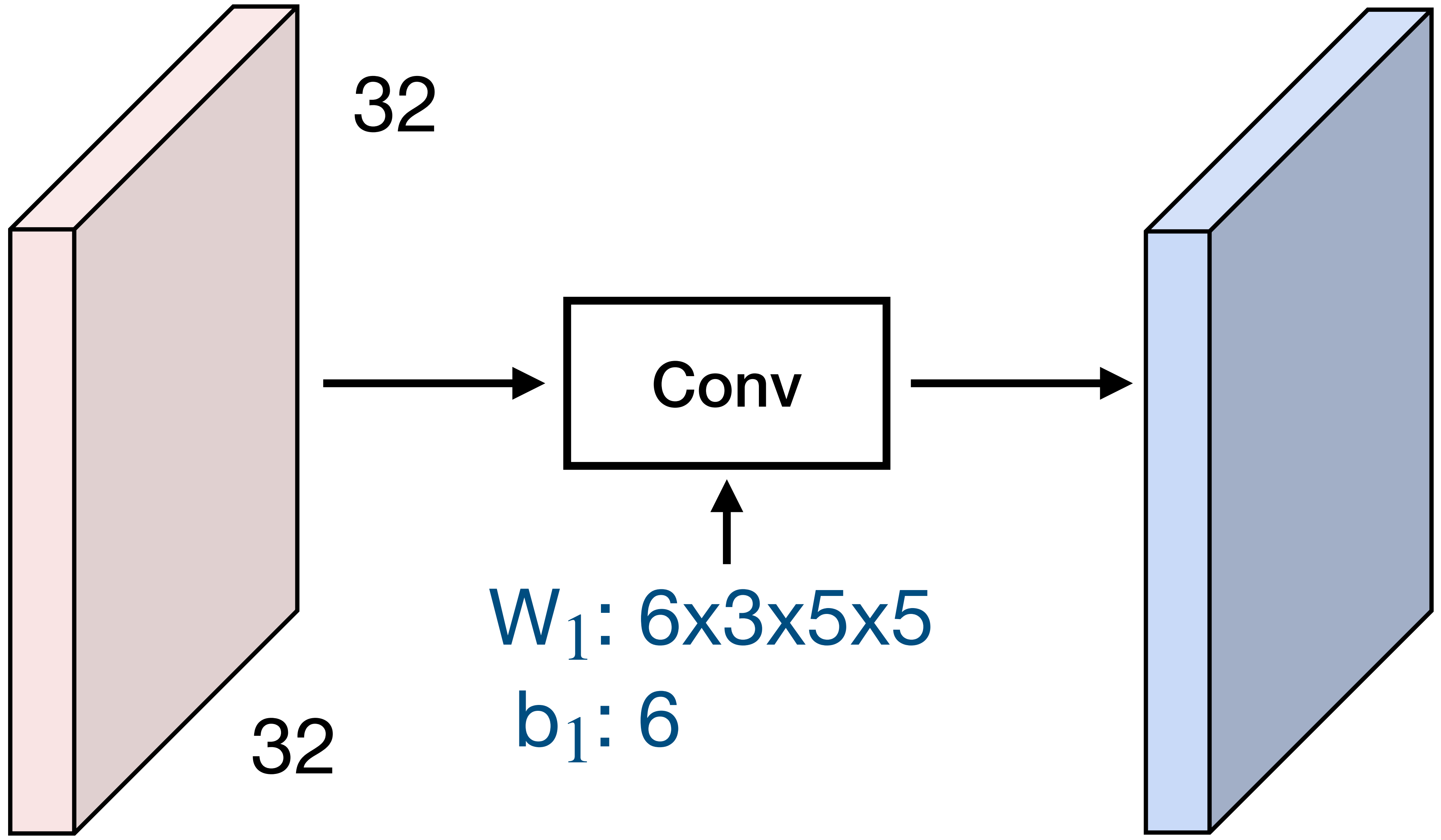
Convolution Layer



Convolution Layer



Stacking Convolutions

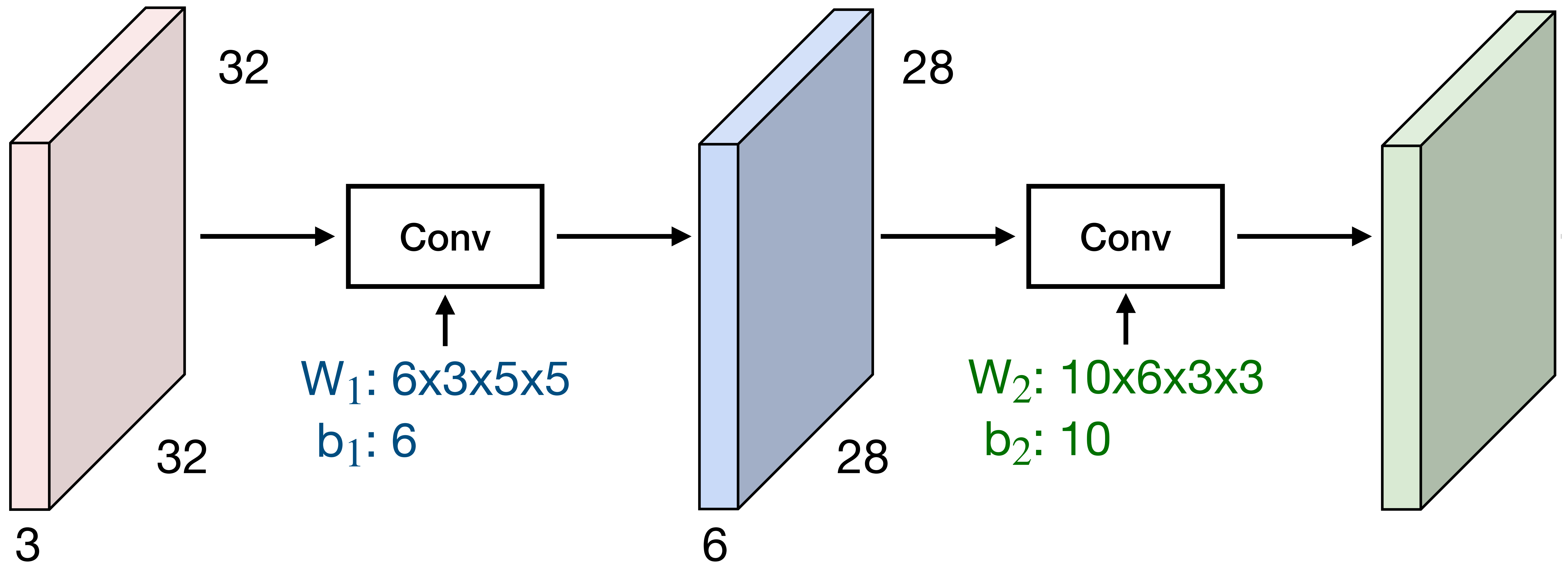


3

input

N x 3 x 32 x 32

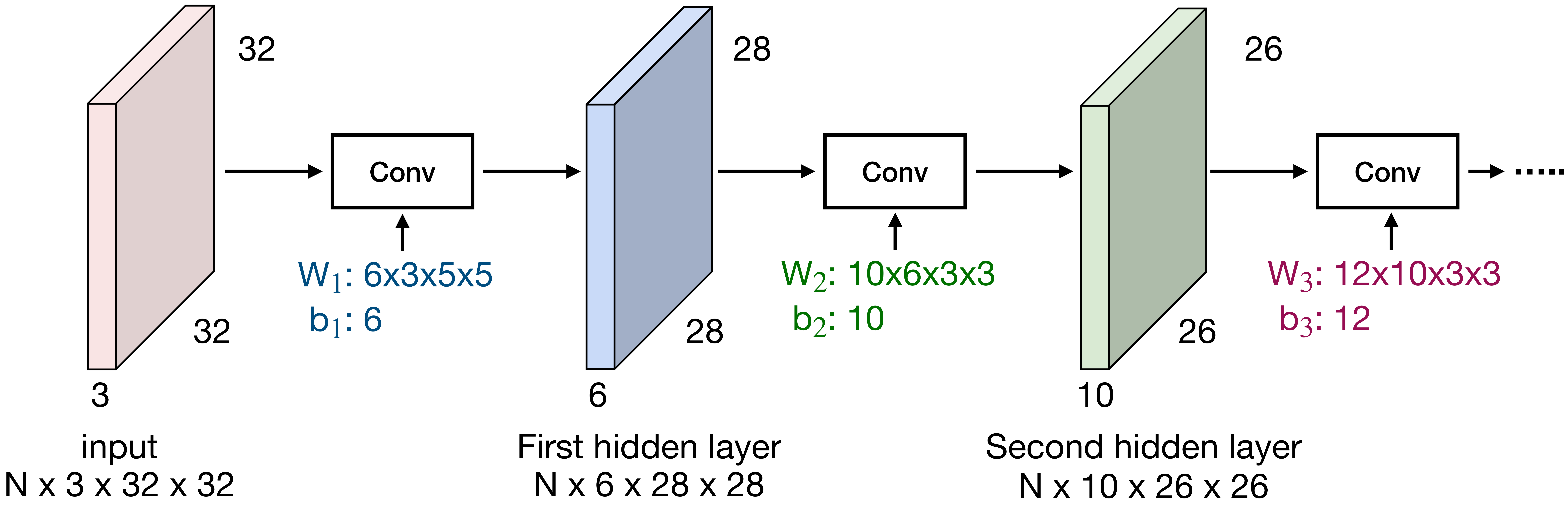
Stacking Convolutions



input
N x 3 x 32 x 32

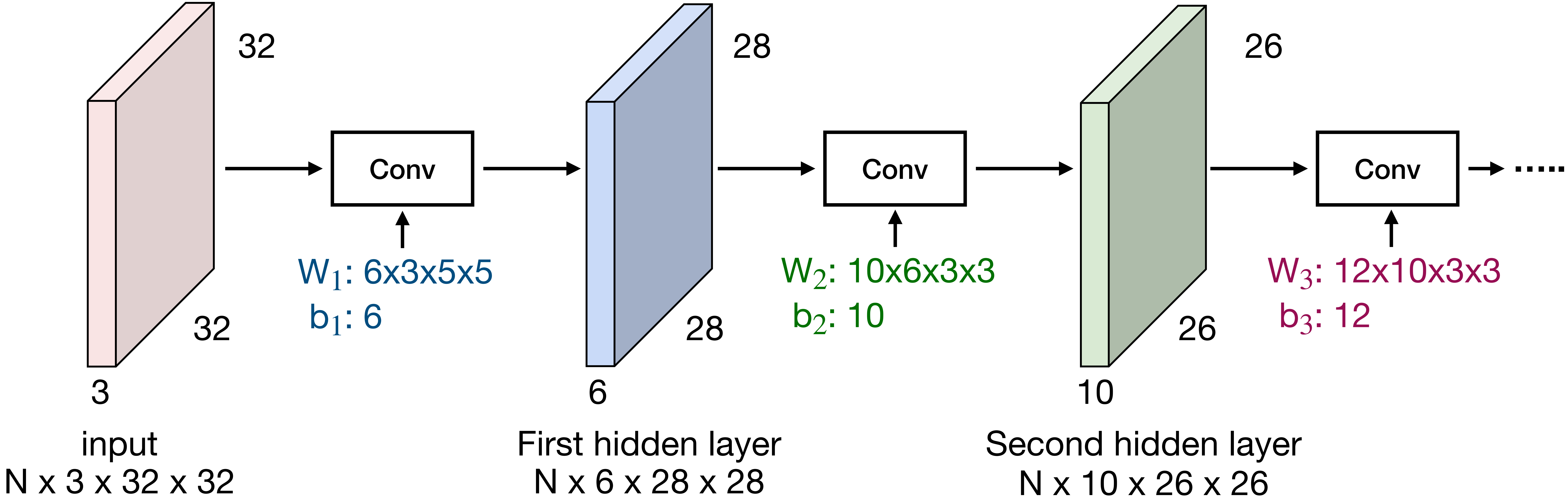
First hidden layer
N x 6 x 28 x 28

Stacking Convolutions



Stacking Convolutions

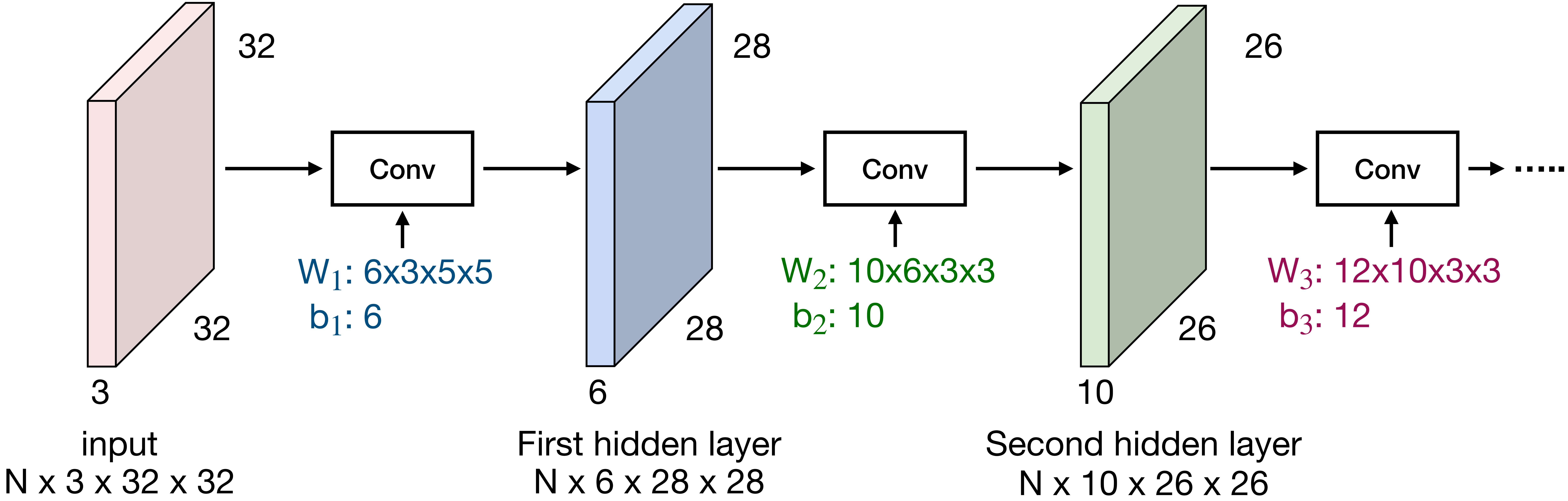
Q: What happens if we stack two convolution layers?



Stacking Convolutions

Q: What happens if we stack two convolution layers?

(Recall $y=W_2W_1x$ is a linear classifier)

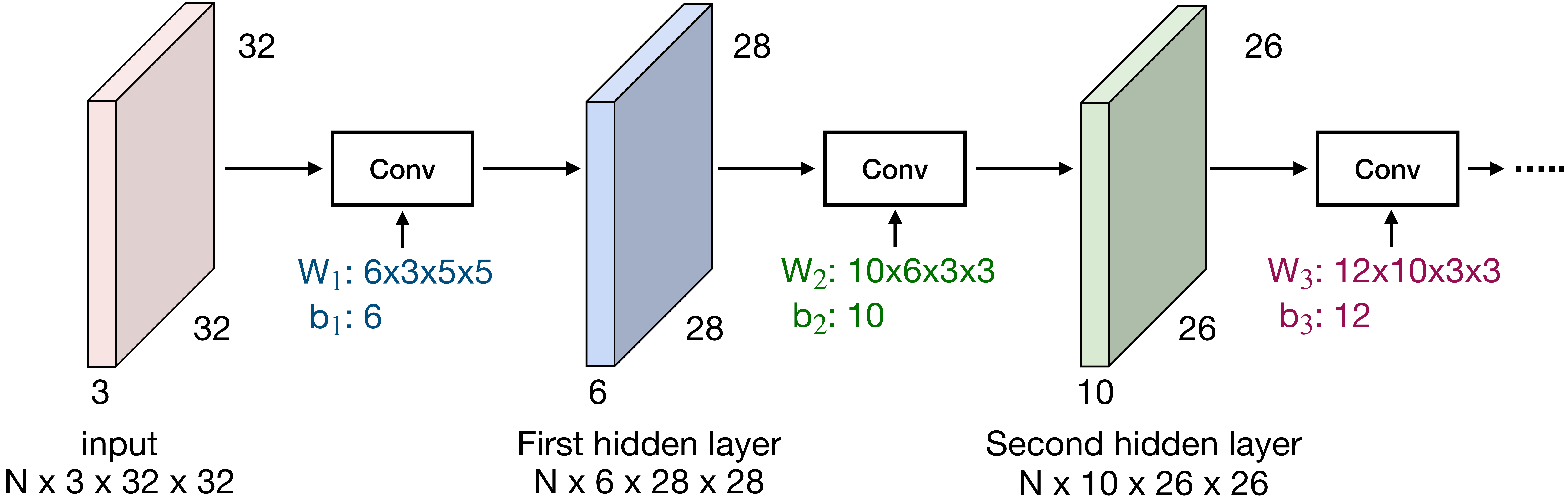


Stacking Convolutions

Q: What happens if we stack two convolution layers?

(Recall $y=W_2W_1x$ is a linear classifier)

A: We get another convolution!

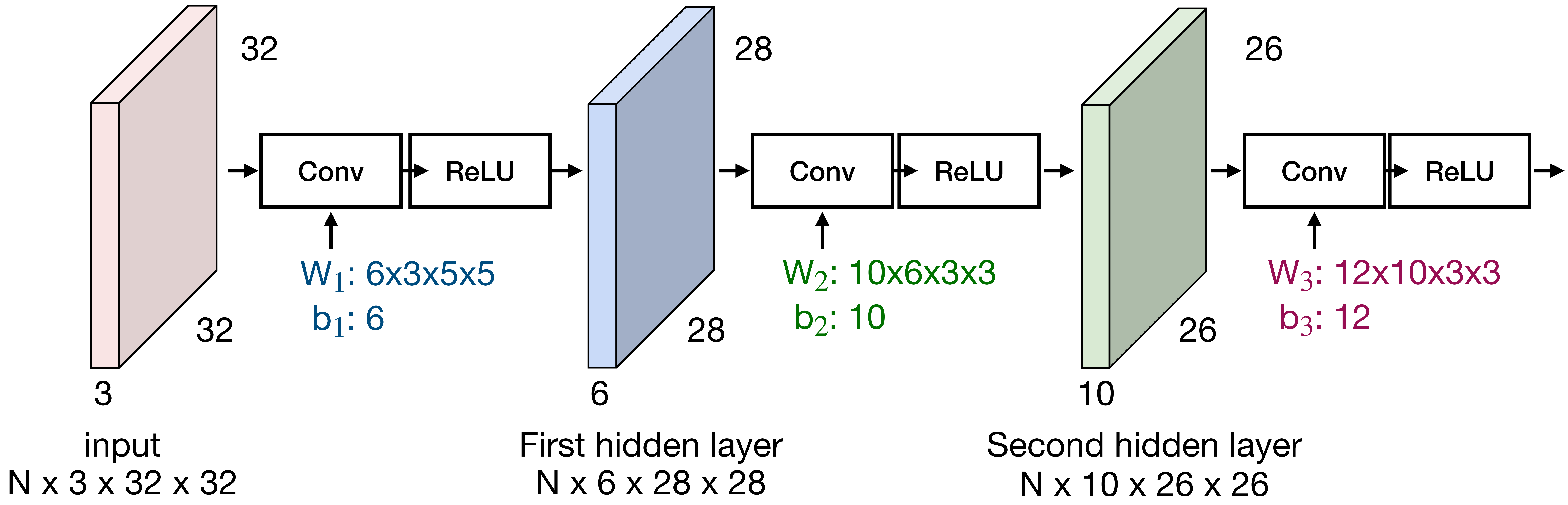


Stacking Convolutions

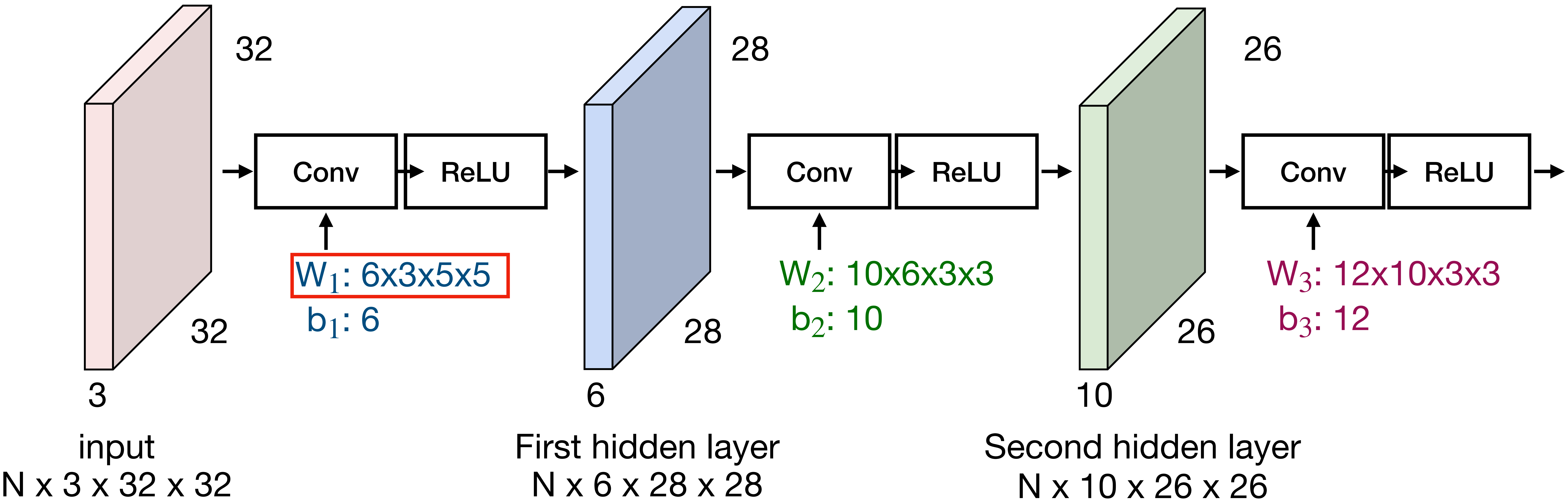
Q: What happens if we stack two convolution layers?

(Recall $y=W_2W_1x$ is a linear classifier)

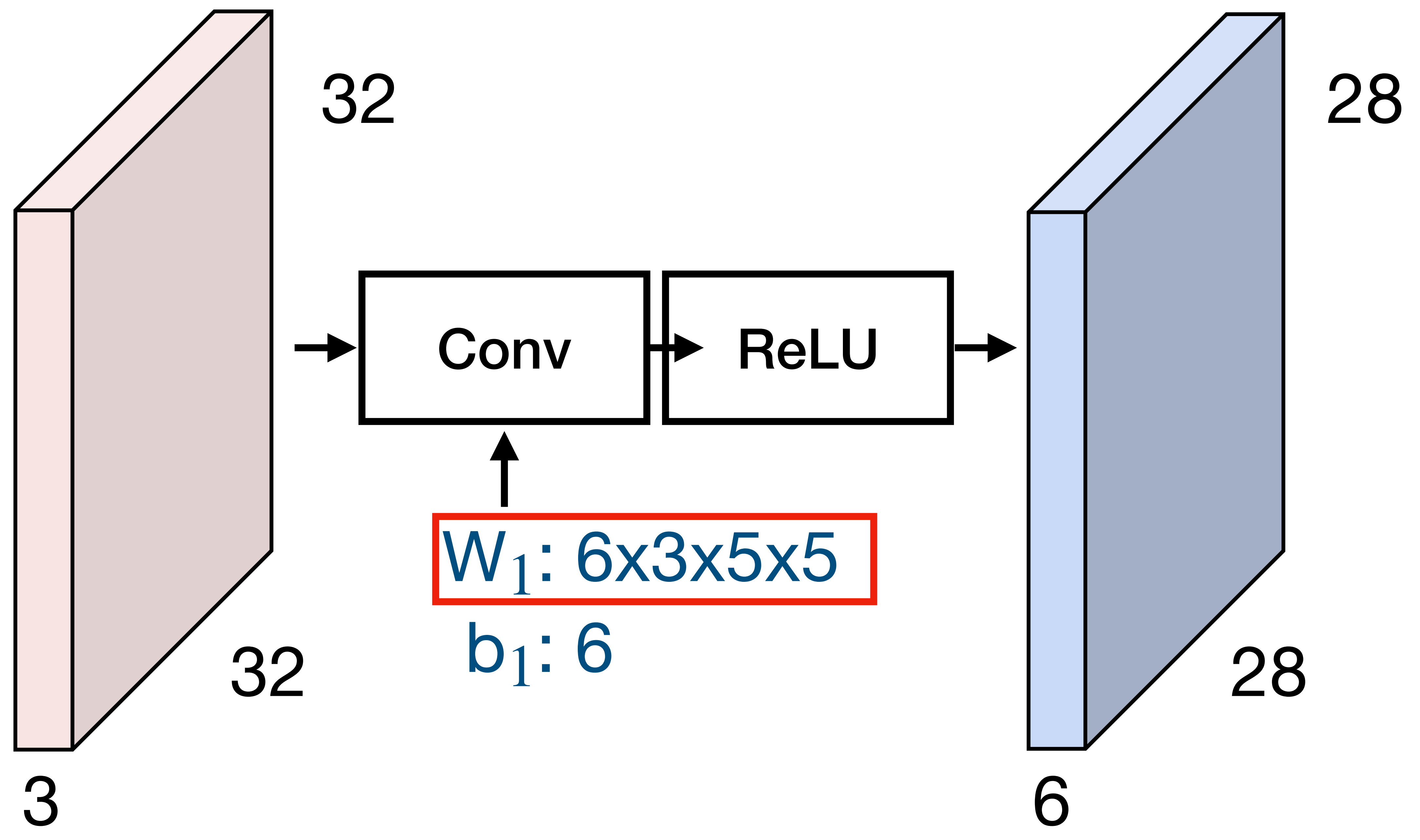
A: We get another convolution!



What do convolutional filters learn?



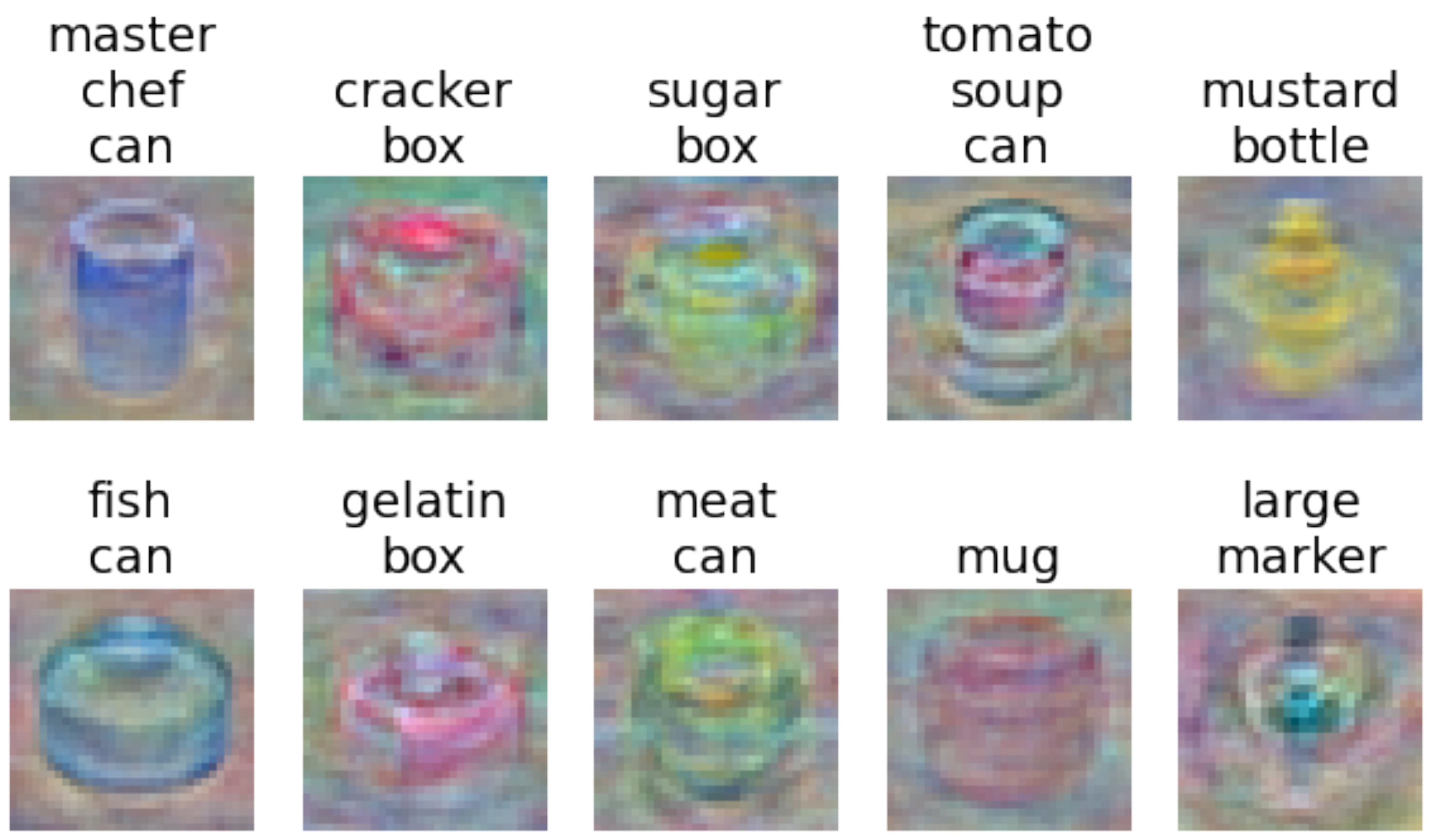
What do convolutional filters learn?



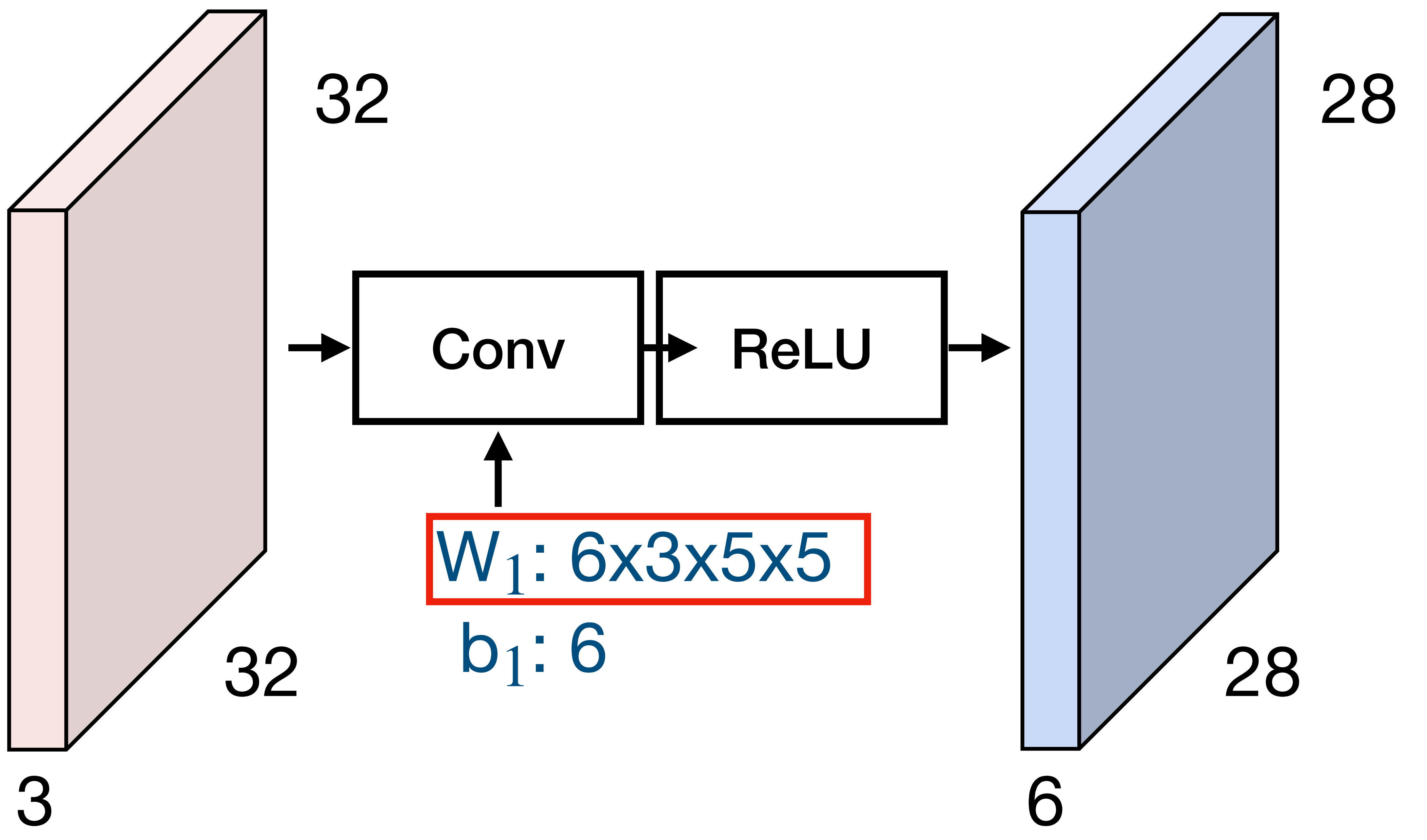
input
N x 3 x 32 x 32

First hidden layer
N x 6 x 28 x 28

Linear classifier: One template per class



What do convolutional filters learn?

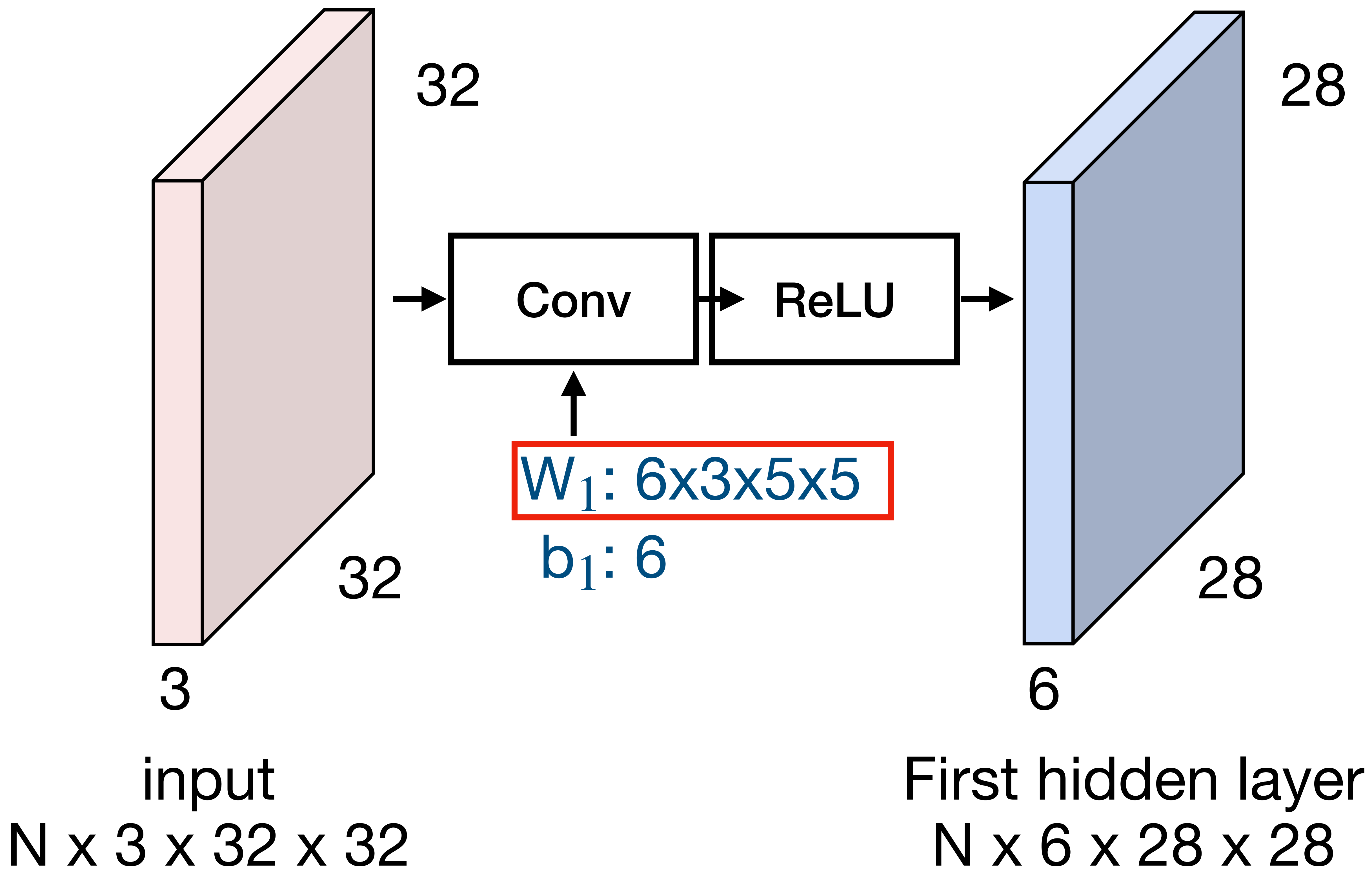


input
N x 3 x 32 x 32

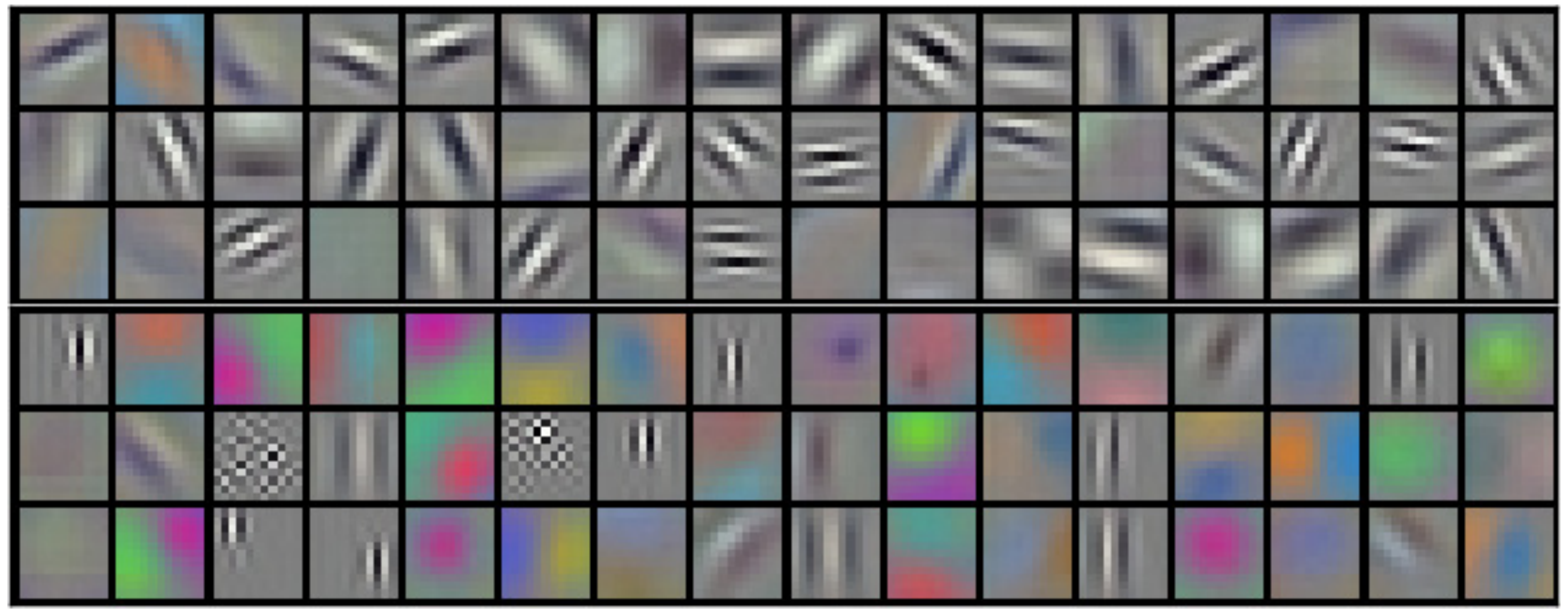
First hidden layer
N x 6 x 28 x 28

MLP: Bank of whole-image templates

What do convolutional filters learn?

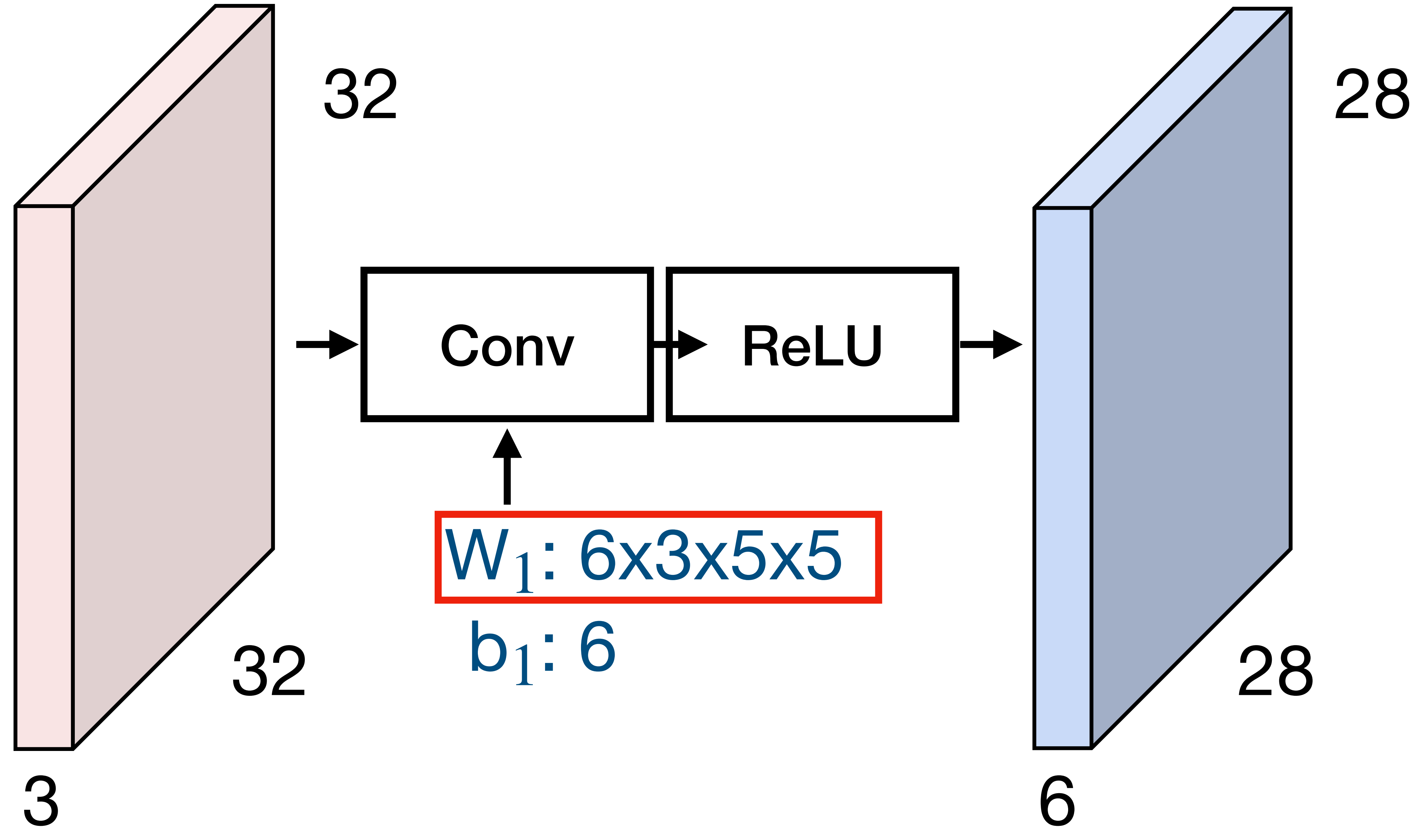


First-layer conv filters: local image templates (often learns oriented edges, opposing colors)



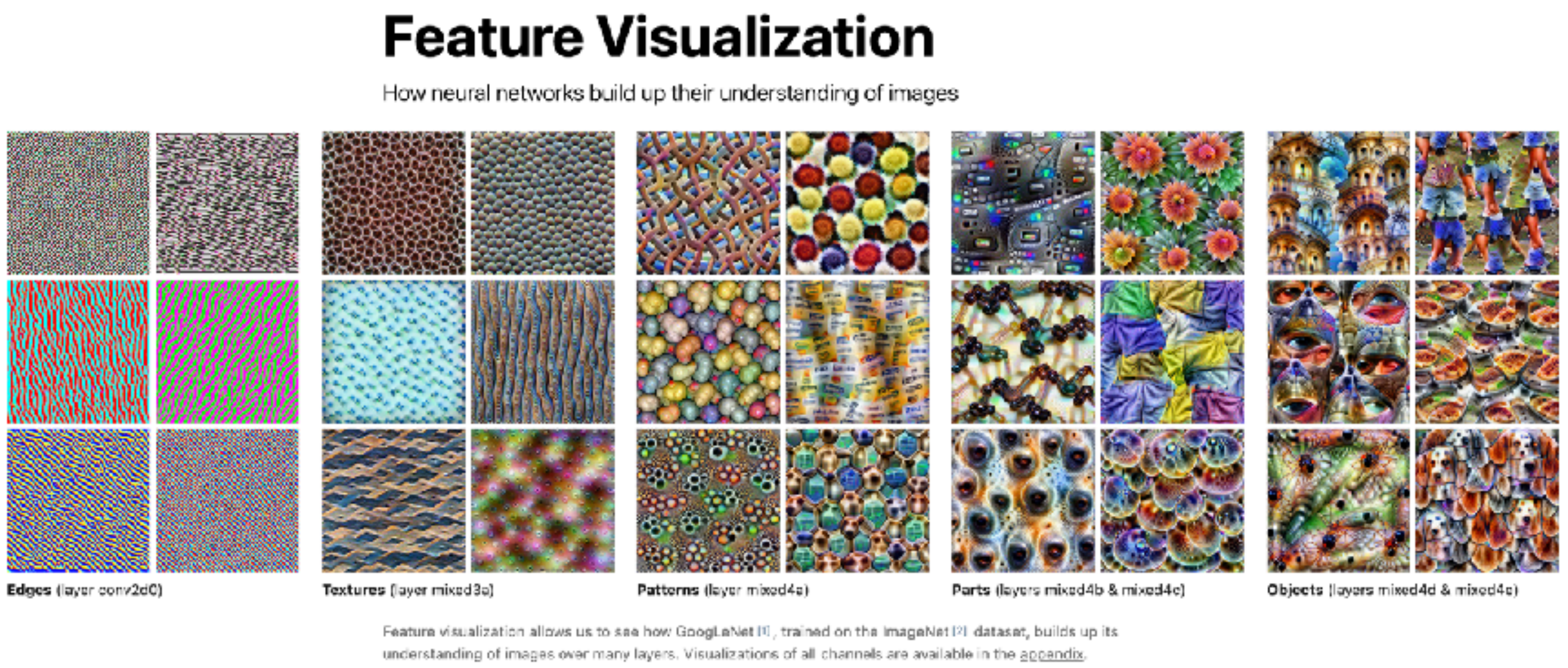
AlexNet: 96 filters, each 3x11x11

What do convolutional filters learn?



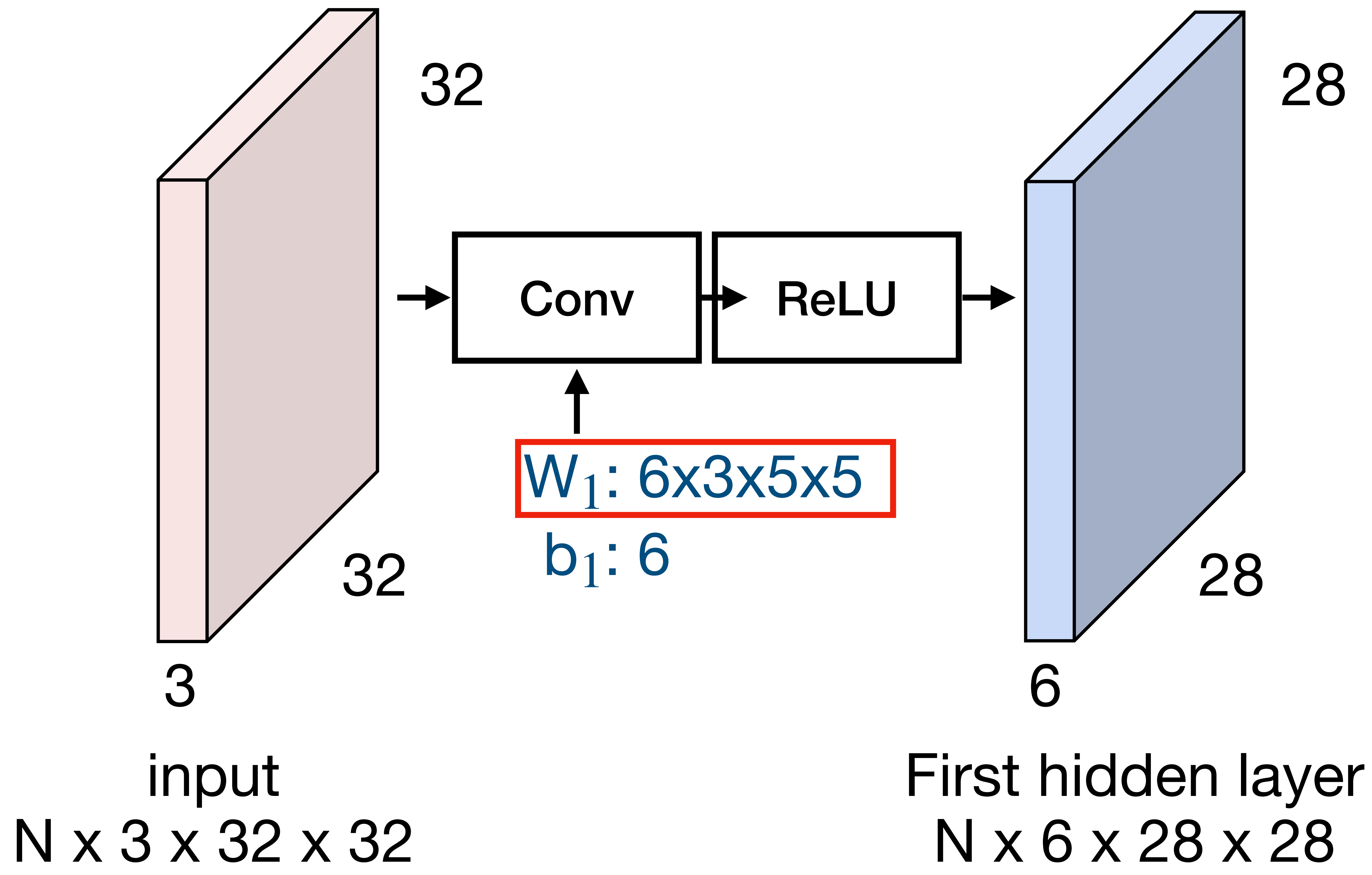
input
N x 3 x 32 x 32

First hidden layer
N x 6 x 28 x 28

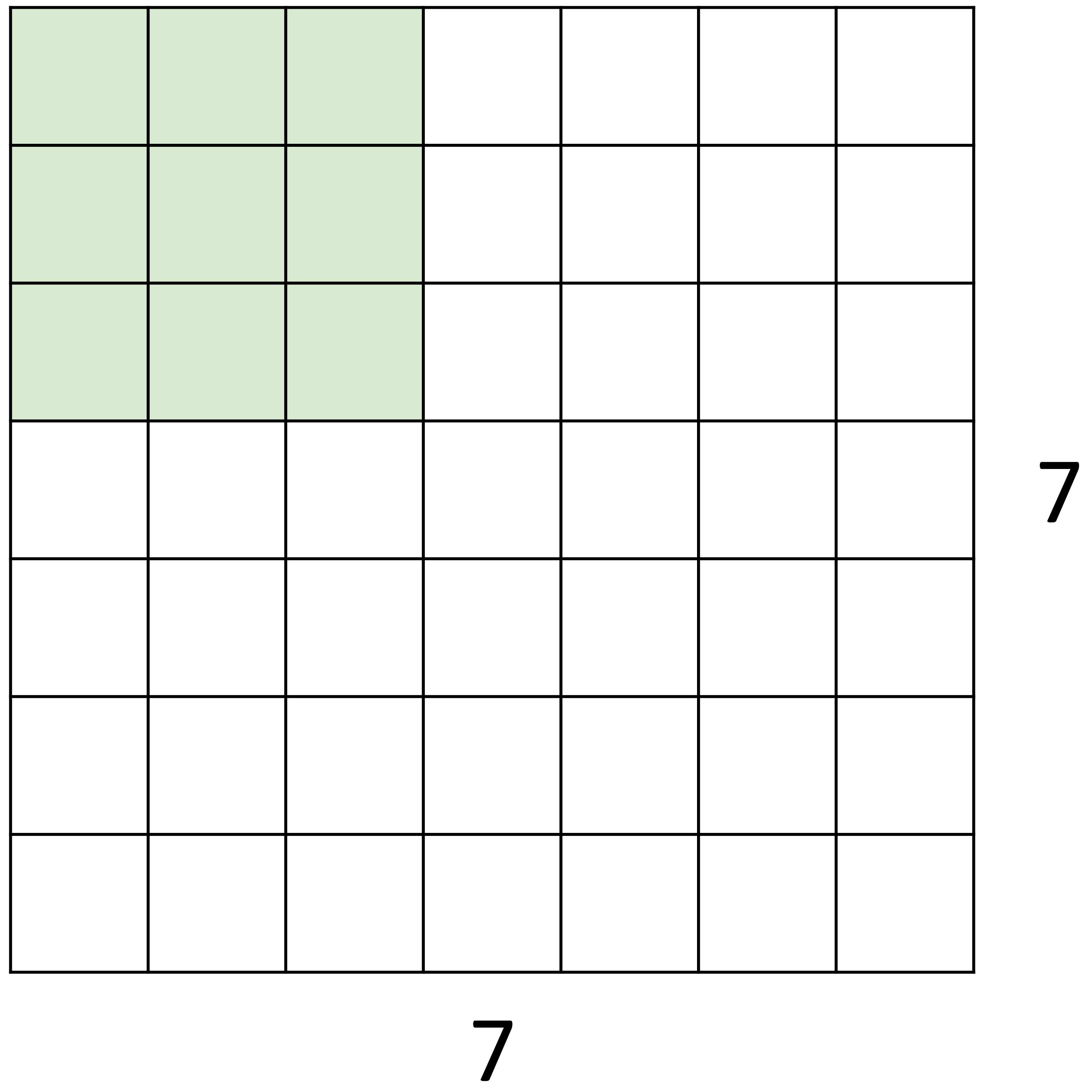


Olah, et al., "Feature Visualization", Distill, 2017.

A closer look at spatial dimensions



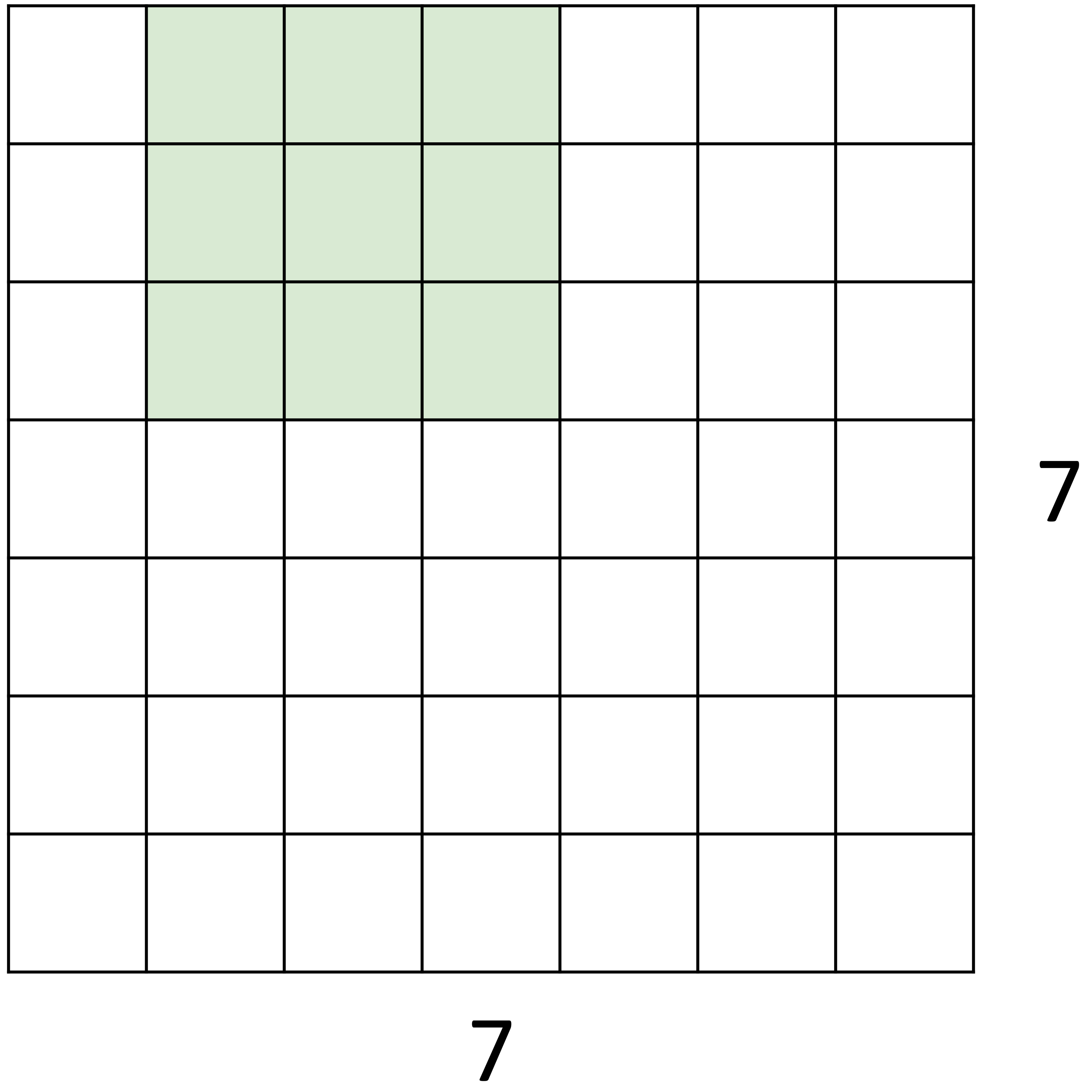
A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

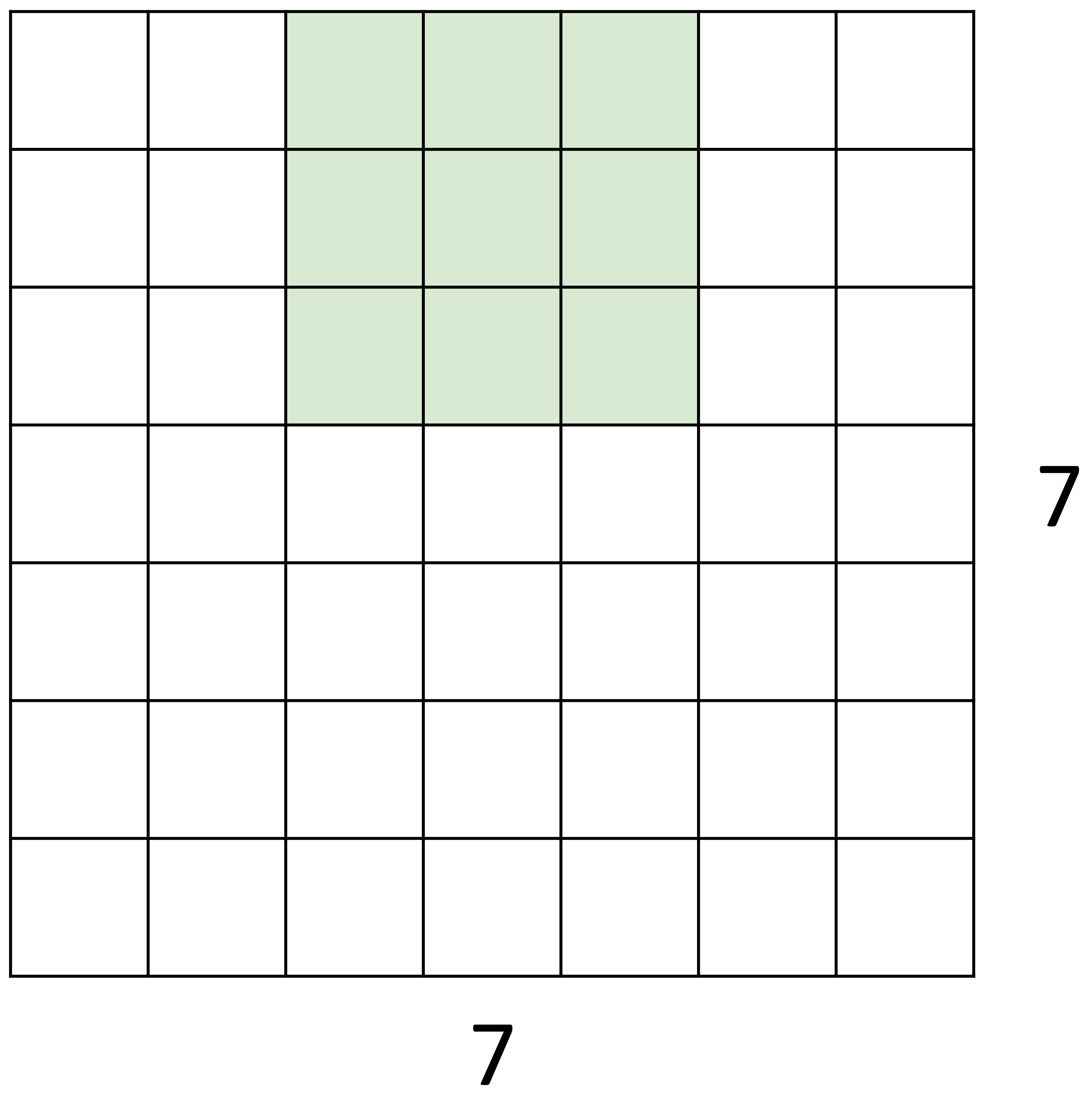
A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

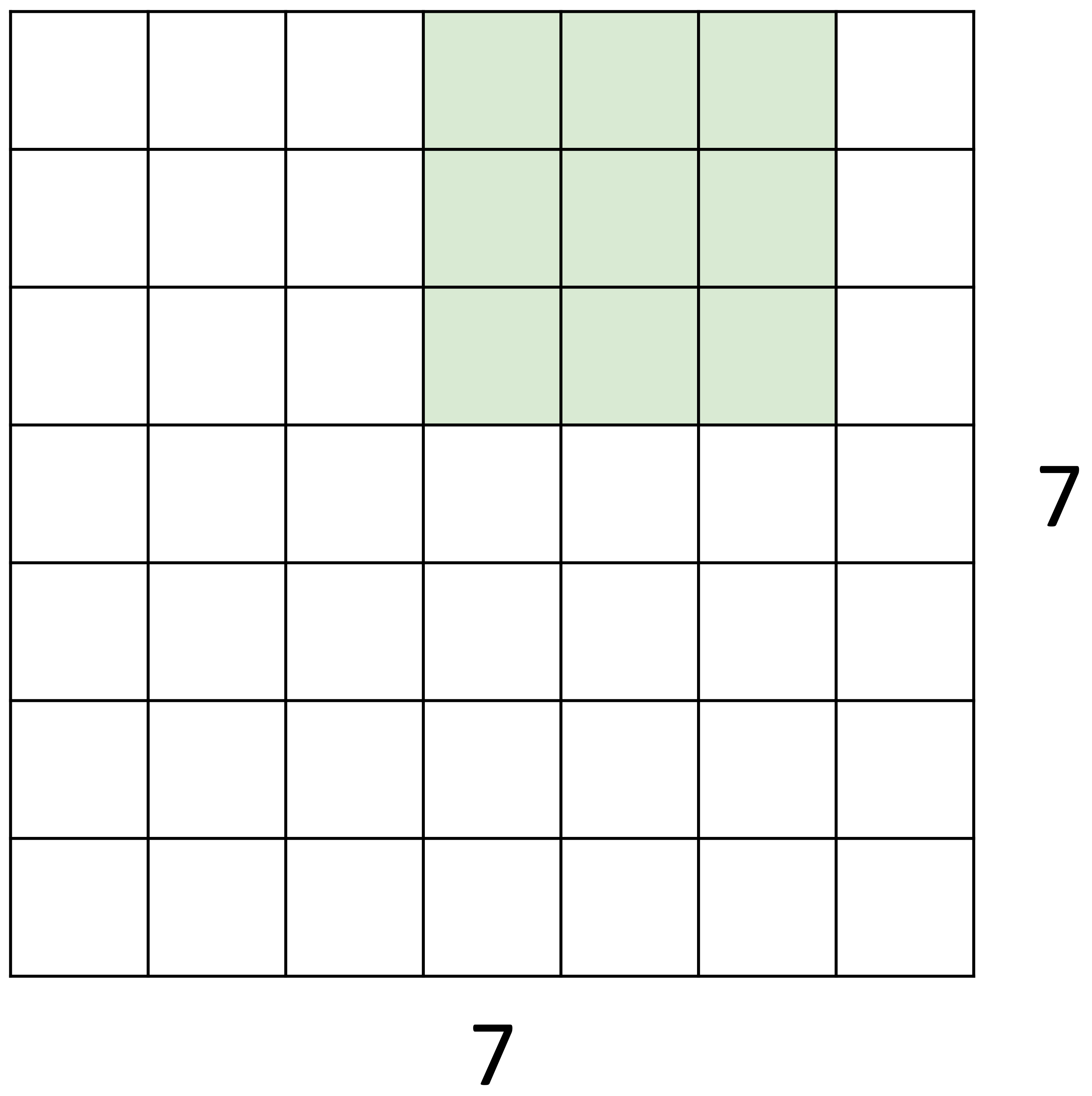
A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

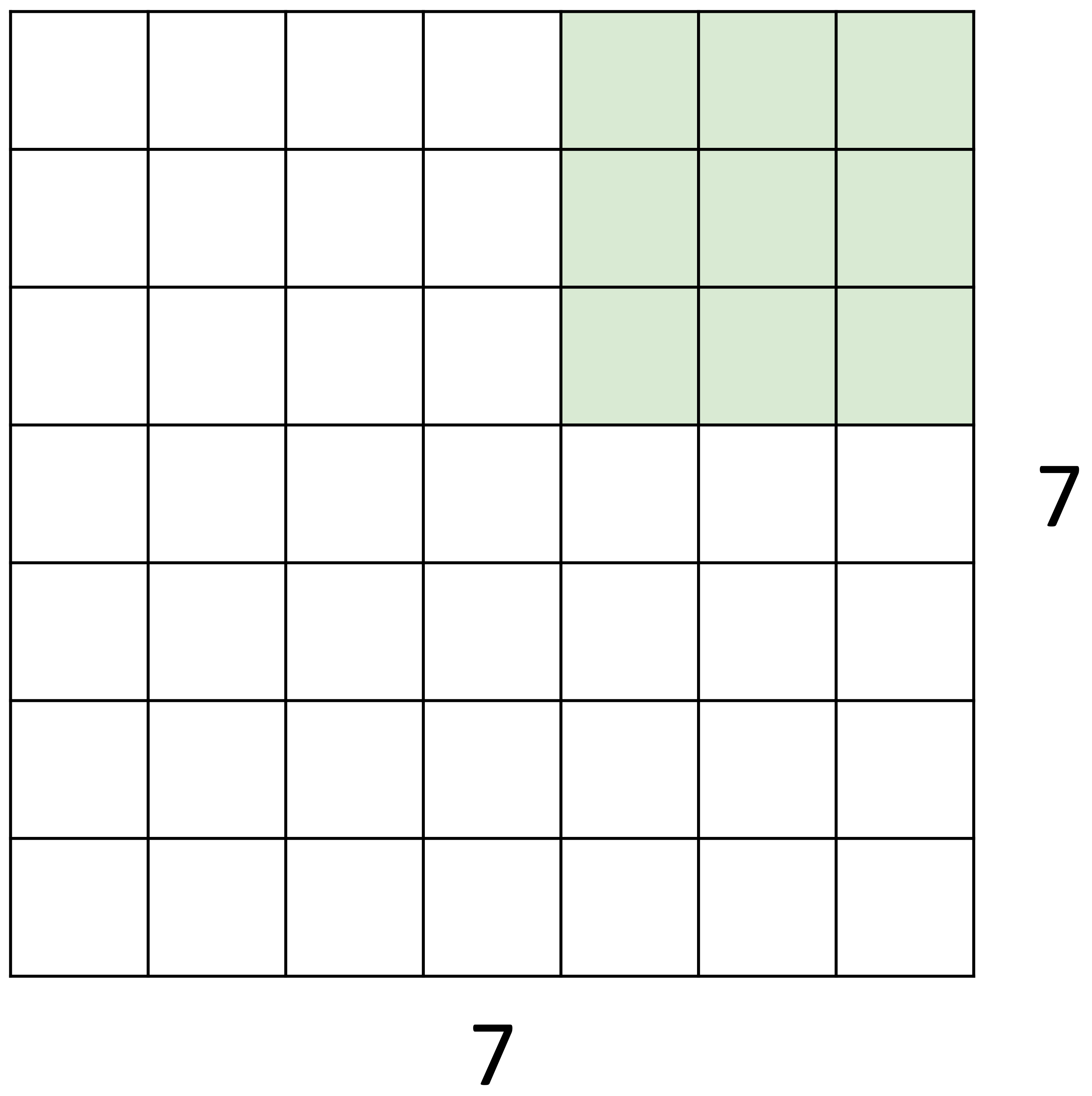
A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

A closer look at spatial dimensions

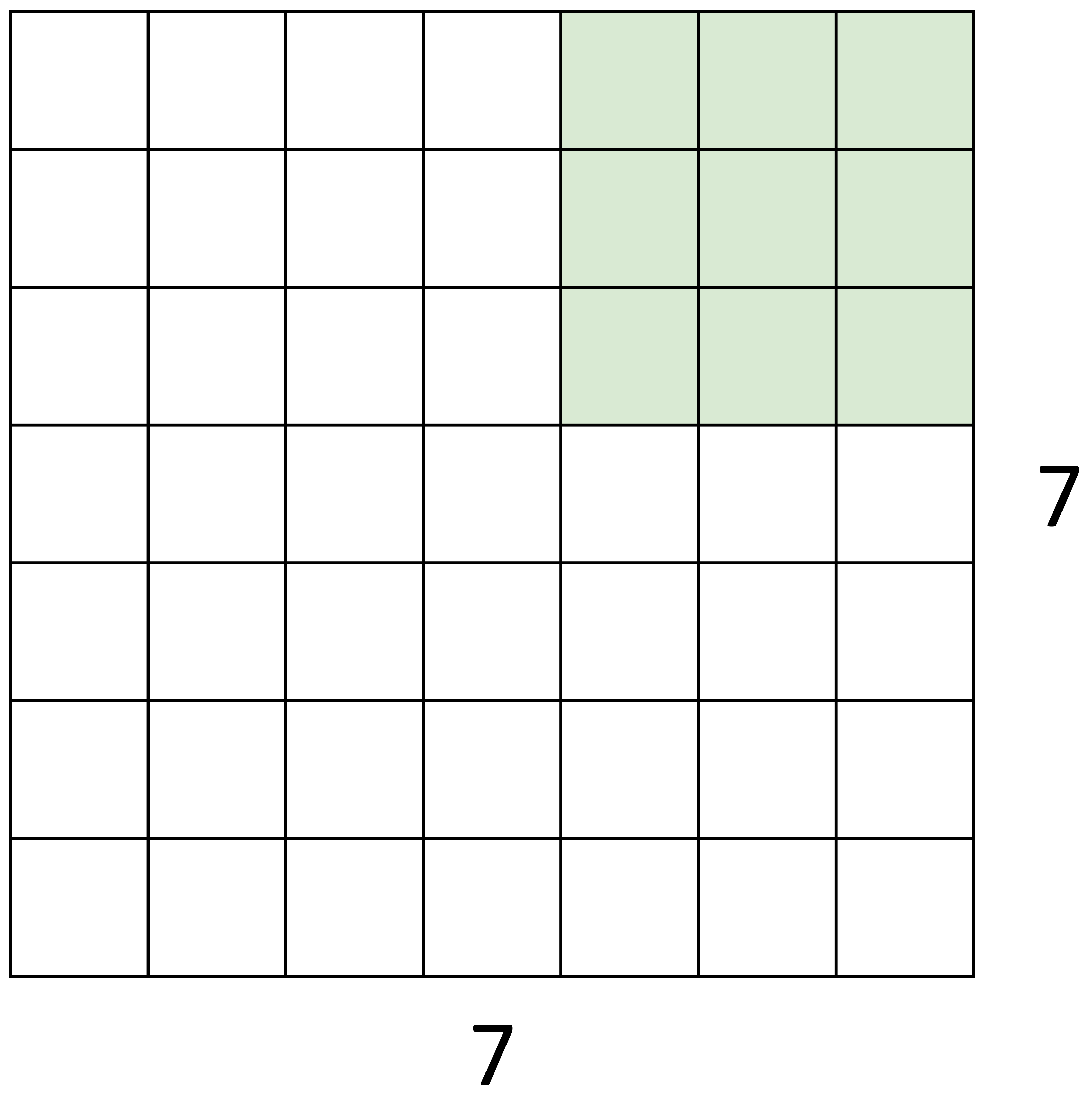


Input: 7x7

Filter: 3x3

Output: 5x5

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!

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

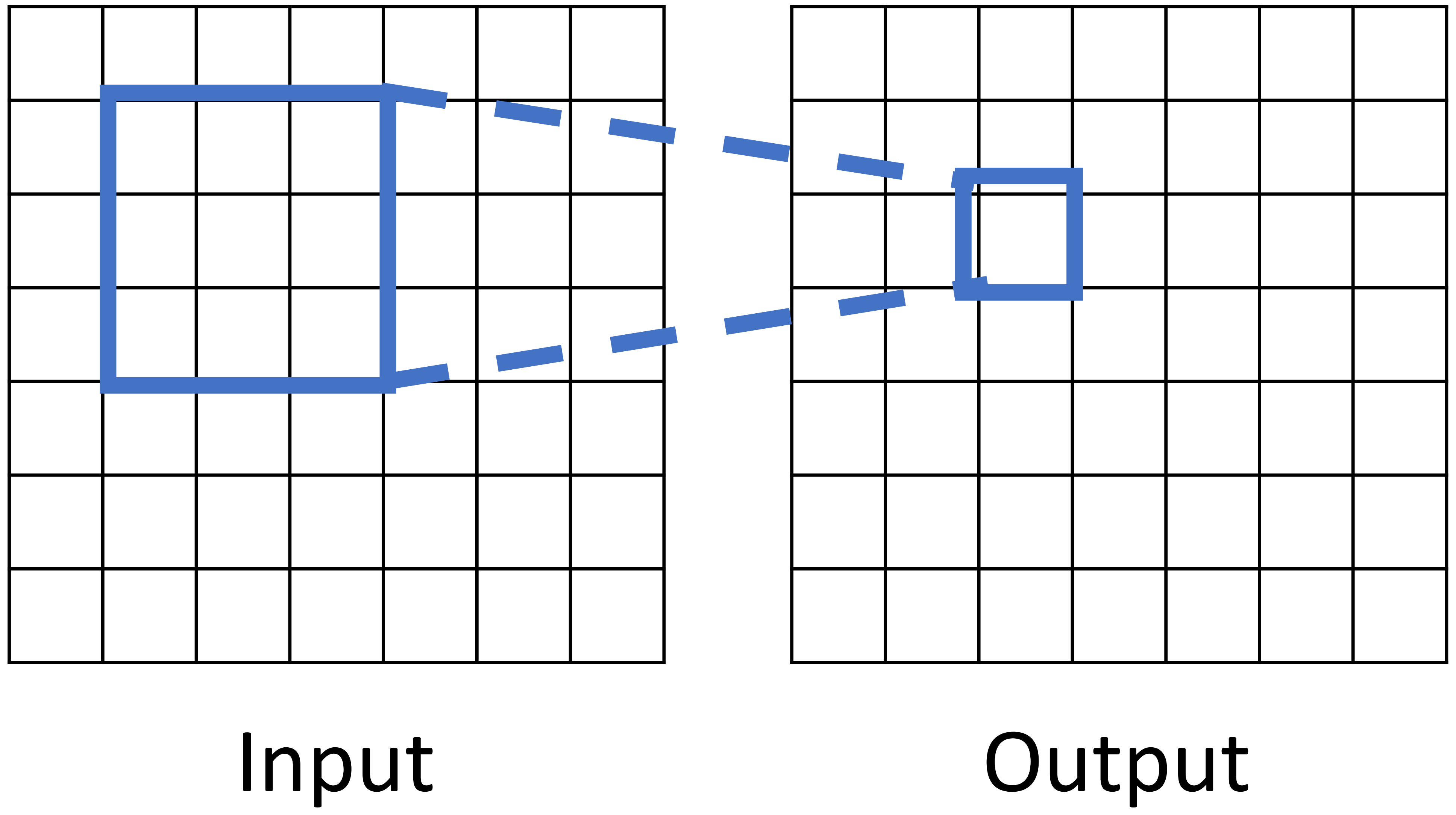
Input: 7x7
Filter: 3x3
Output: 5x5

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

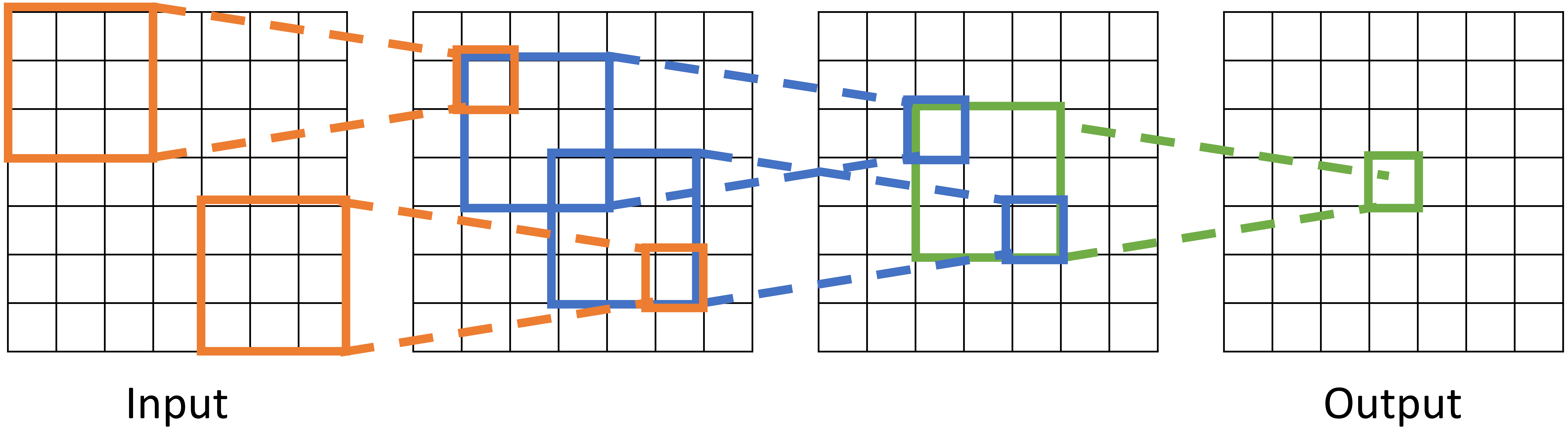


Formally, it is the region in the input space that a particular CNN's feature is affected by.

Informally, it is the part of a tensor that after convolution results in a feature.

Receptive Fields

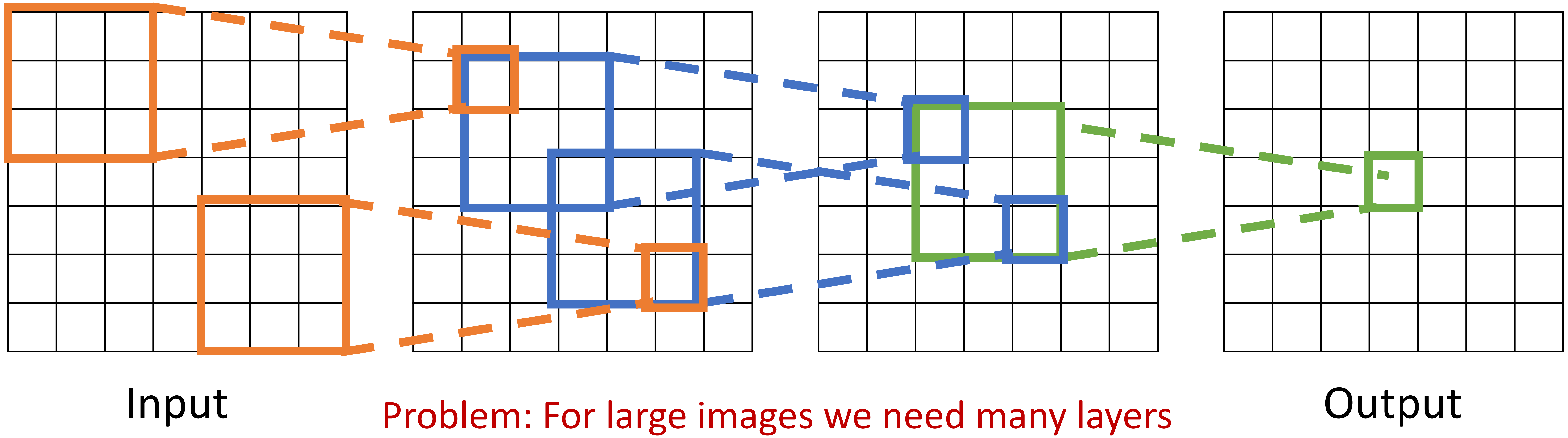
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



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

Receptive Fields

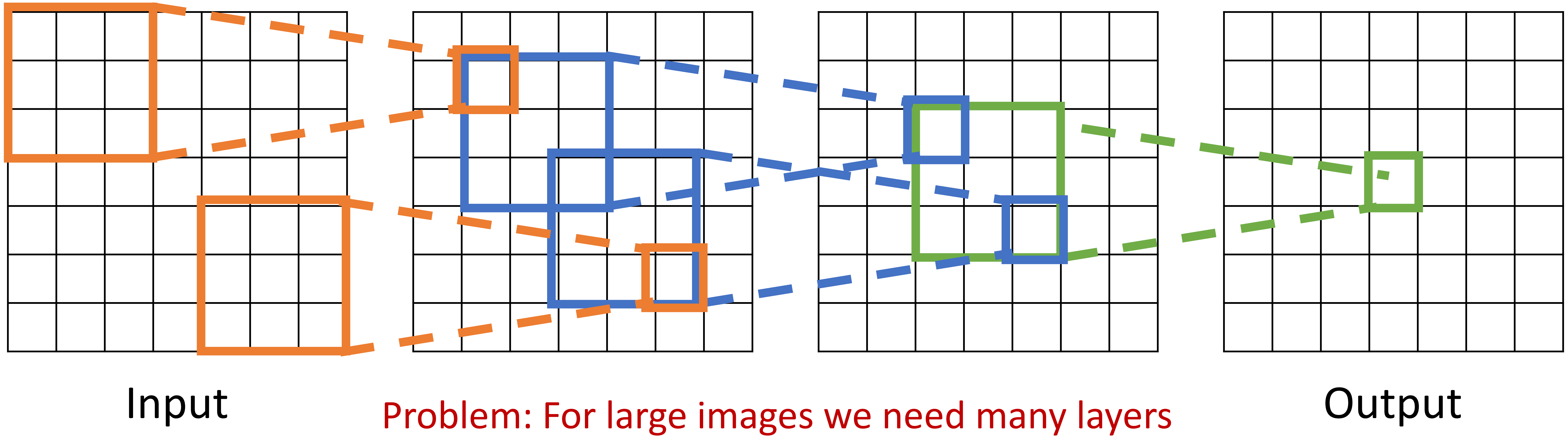
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



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

Receptive Fields

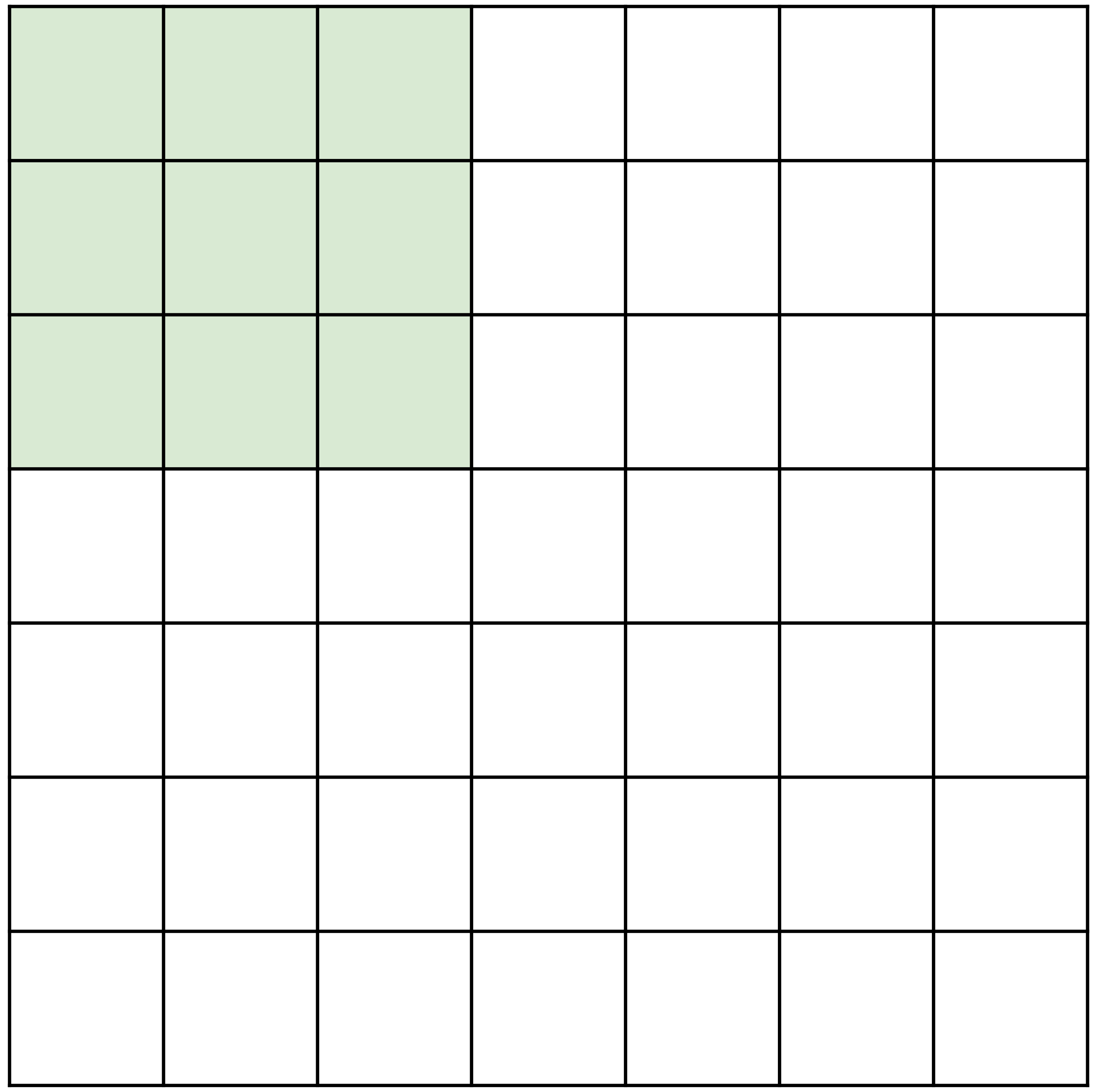
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



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

Solution: Downsample inside the network

Strided Convolution

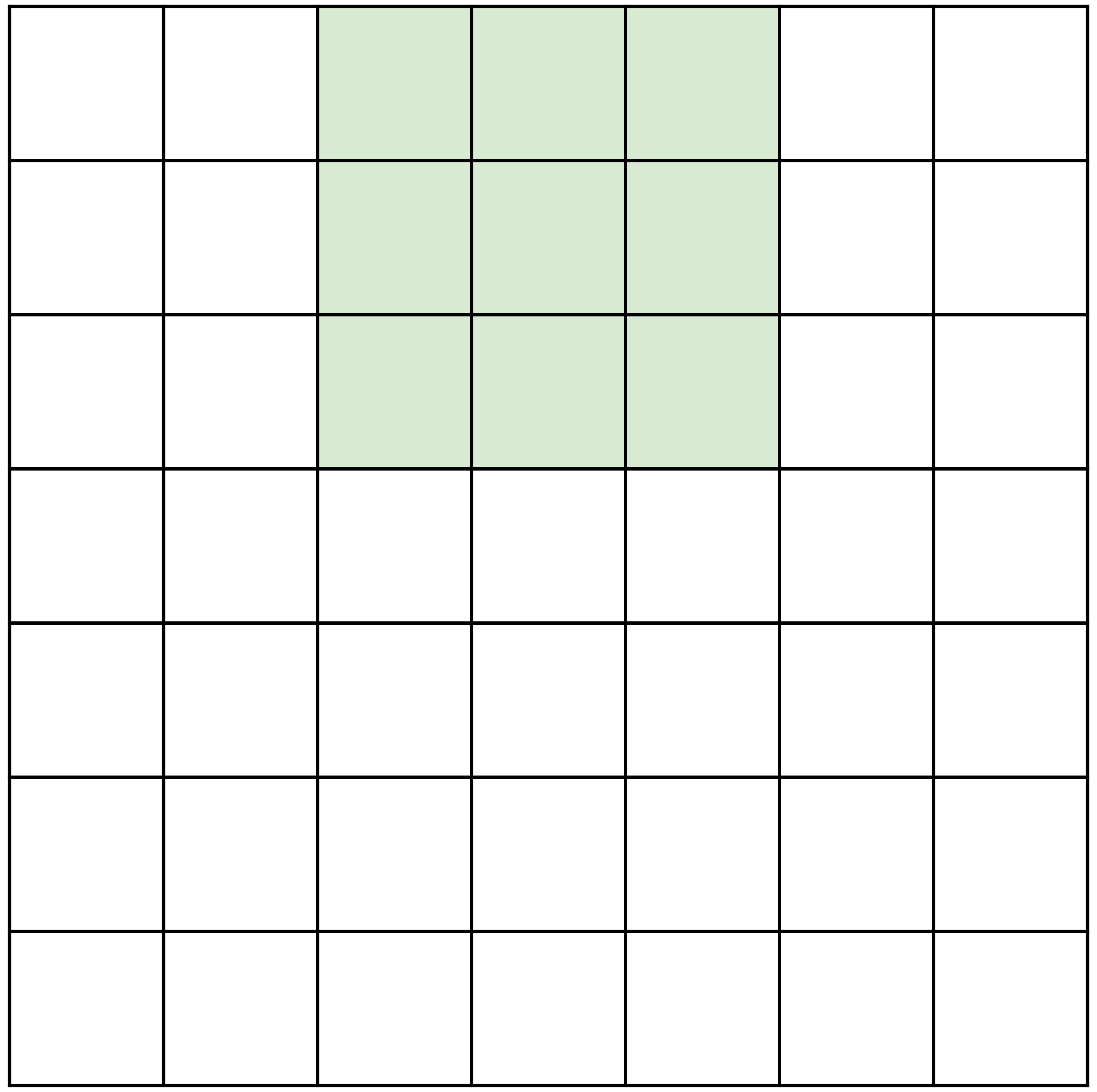


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution

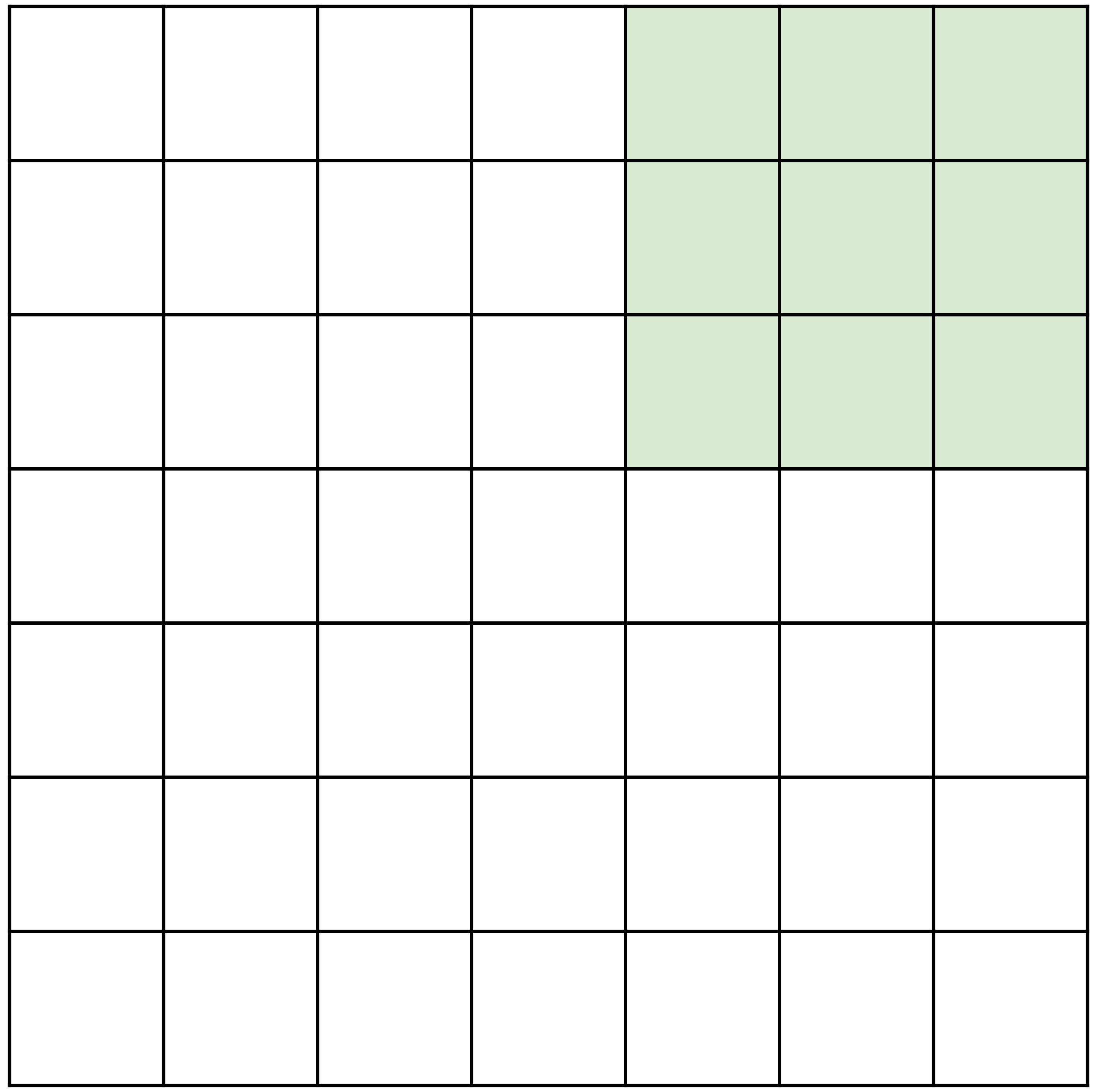


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution



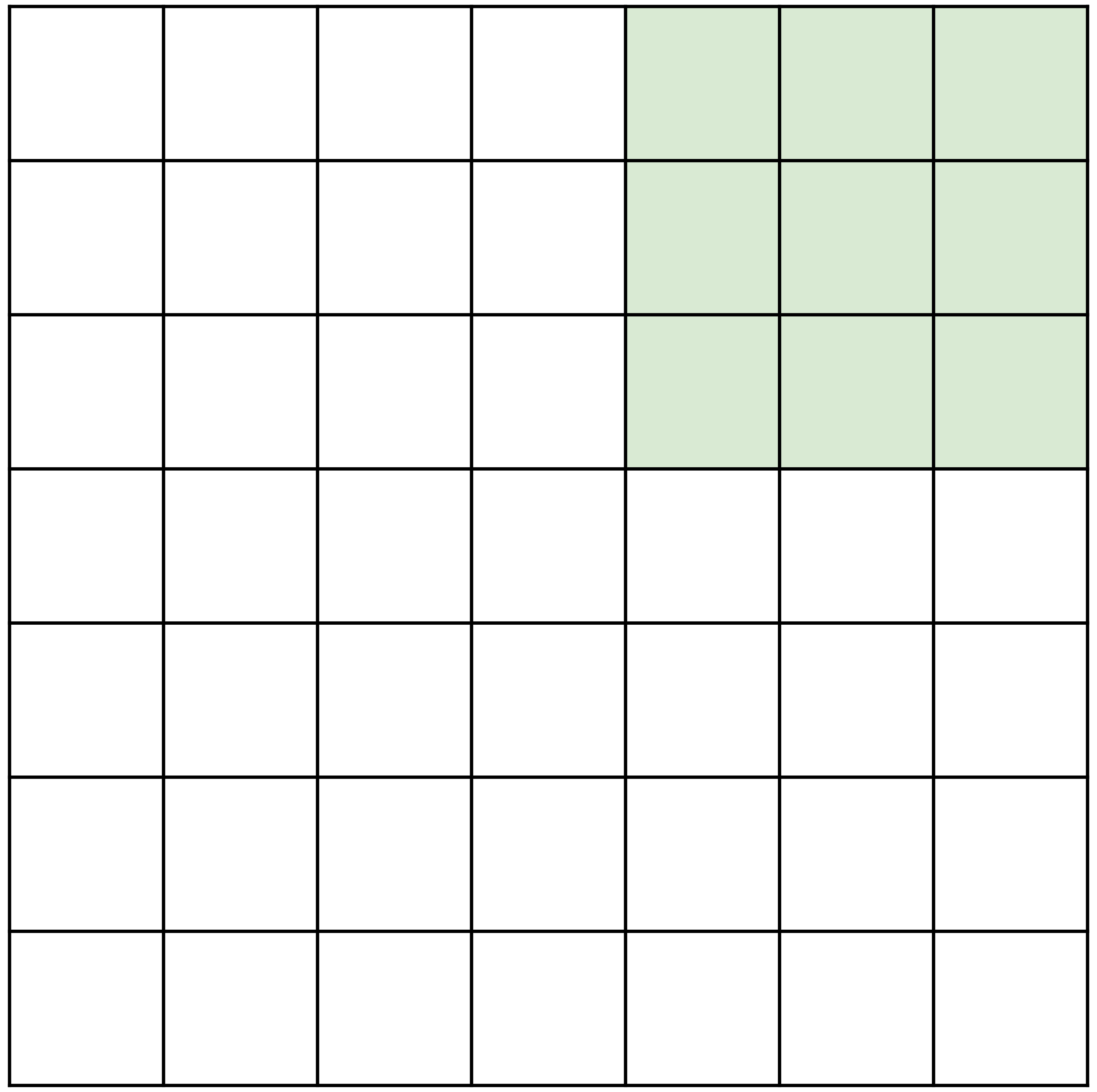
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

Strided Convolution



Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

In general:

Input: W

Filter: K

Padding: P

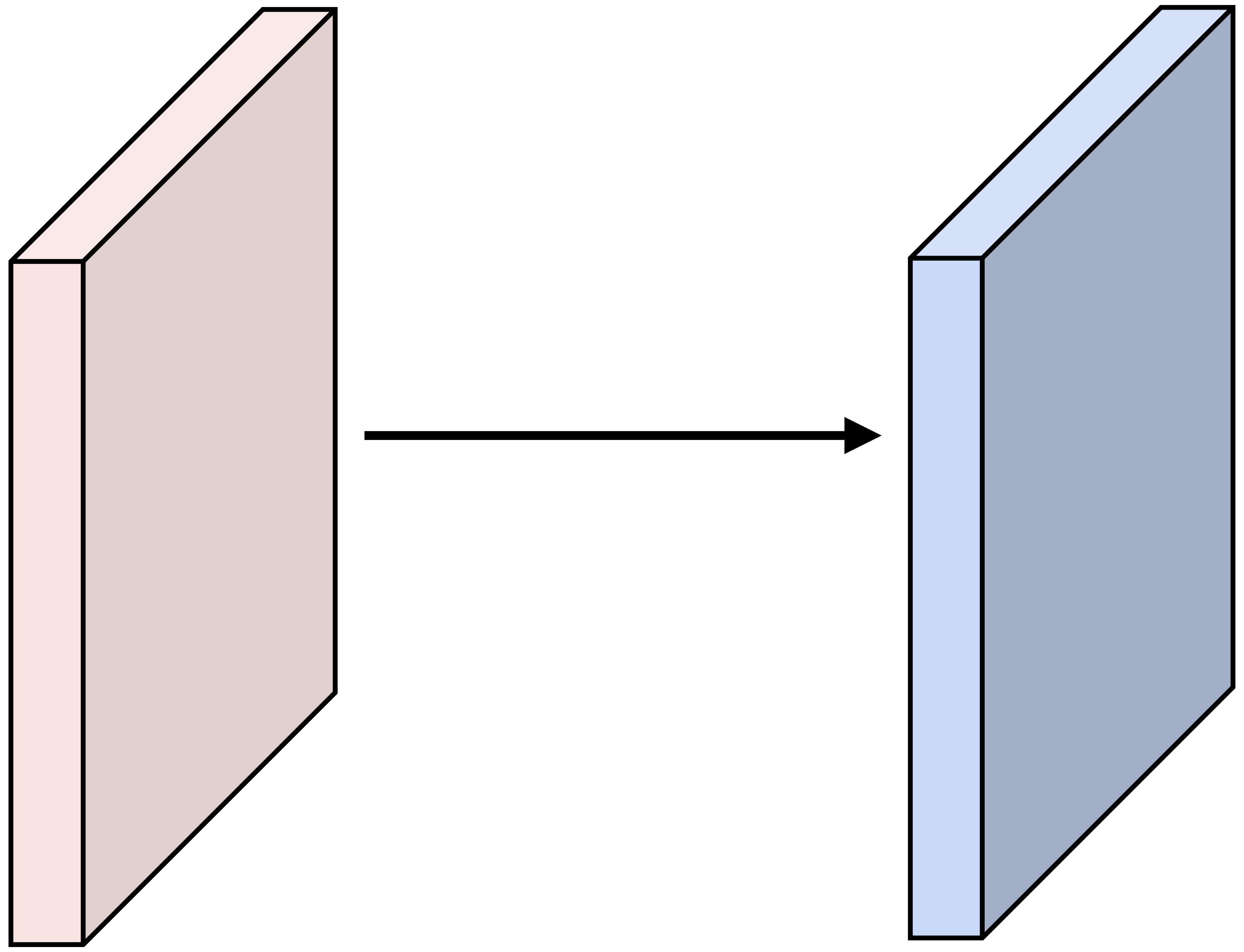
Stride: S

Output: (W – K + 2P) / S + 1

Convolution Example

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

Q: What is the output volume size?



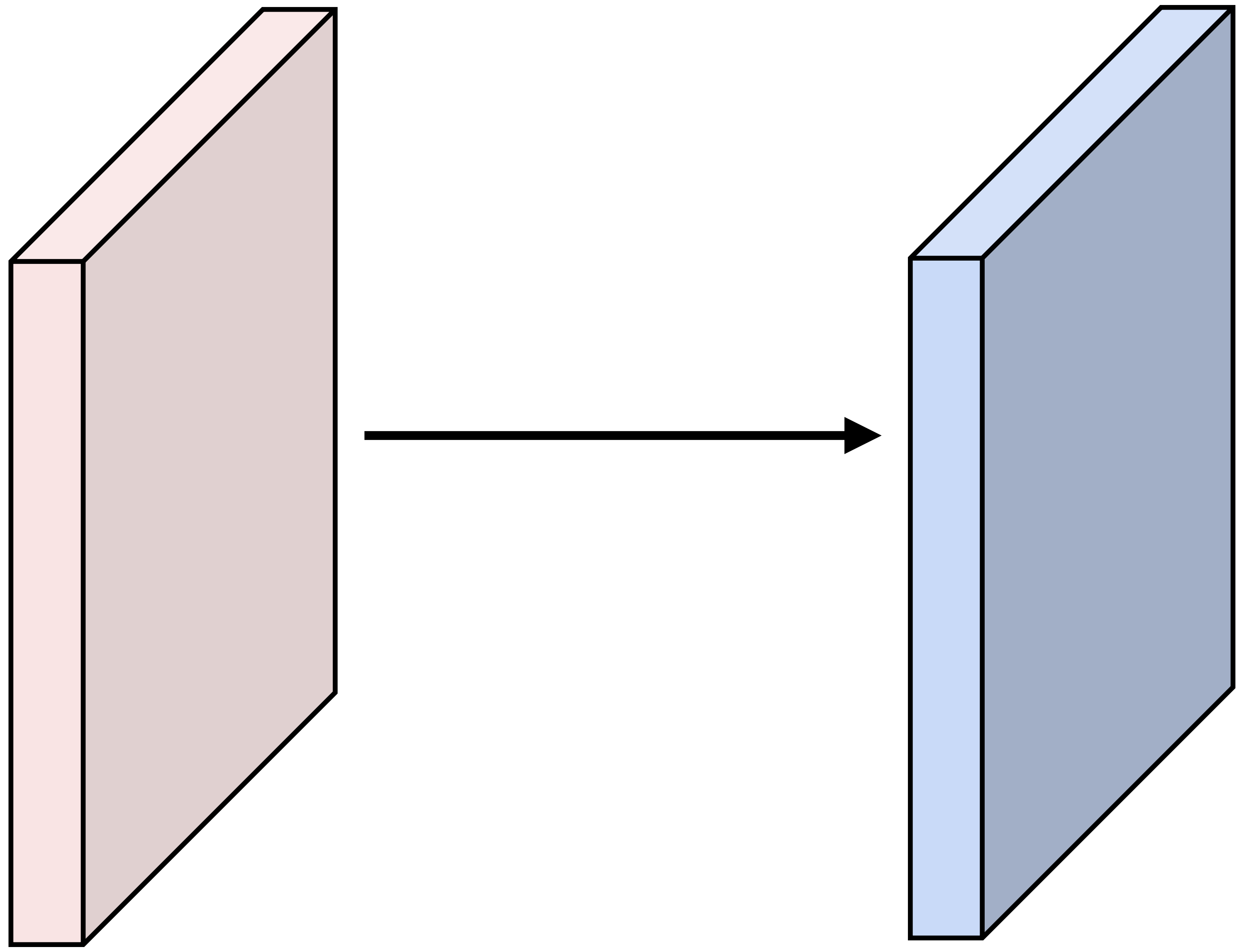
Convolution Example

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

Q: What is the output volume size?

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

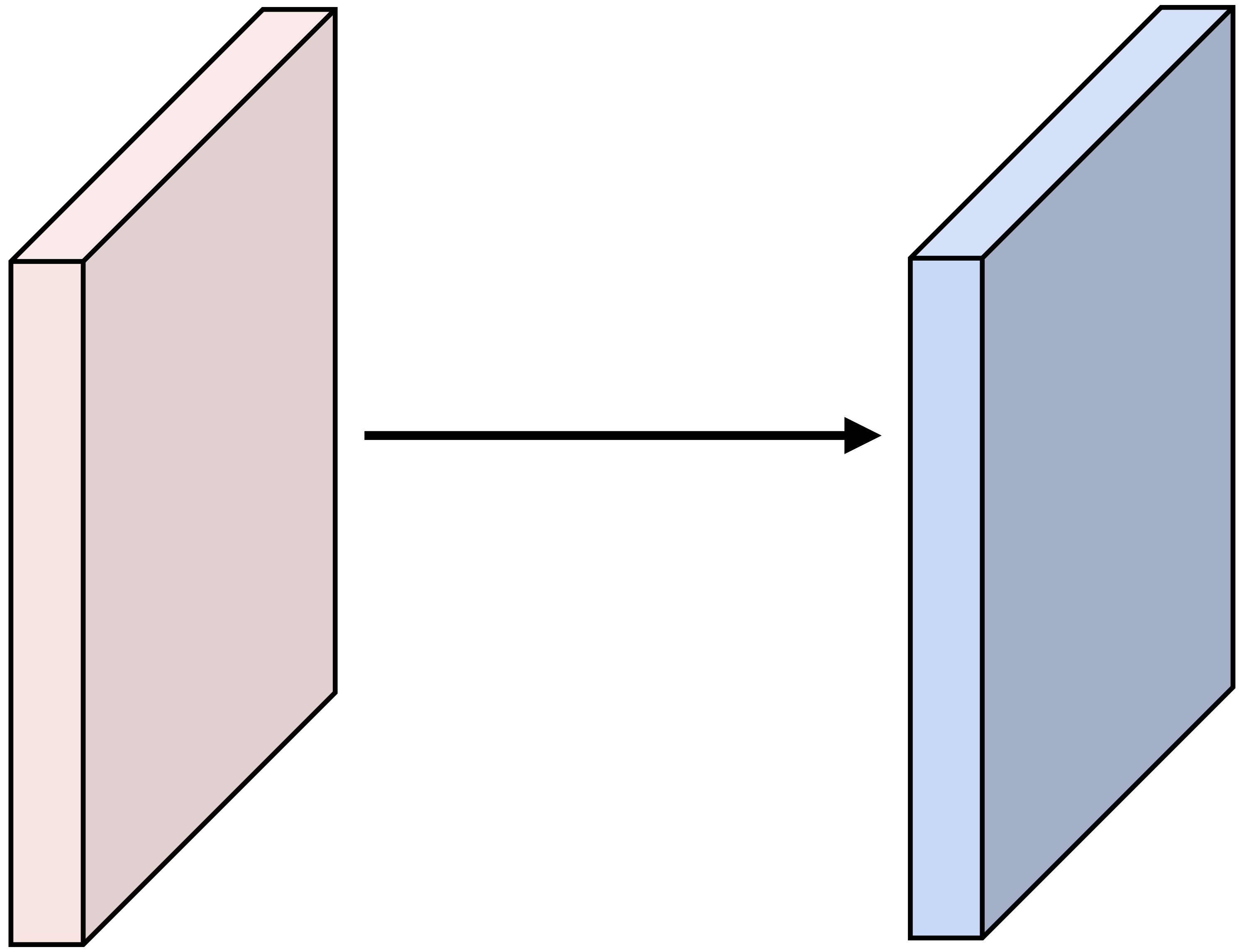
So, 10 x 32 x 32 output



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
Q: What is the number of learnable parameters?



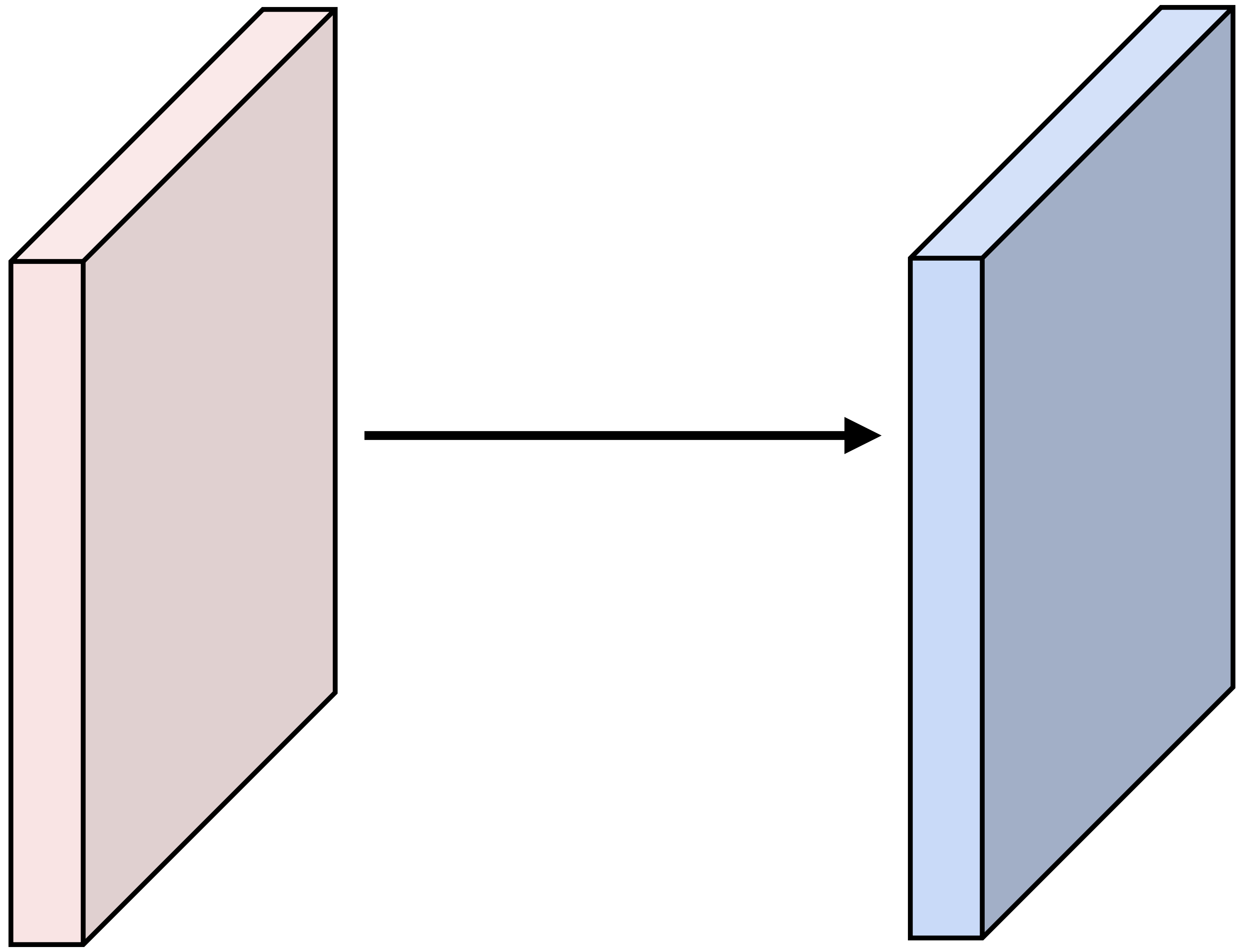
Convolution Example

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

Output volume size: 10 x 32 x 32
Q: What is the number of learnable parameters?

Parmeters per filter: (3*5*5) + 1 = 76

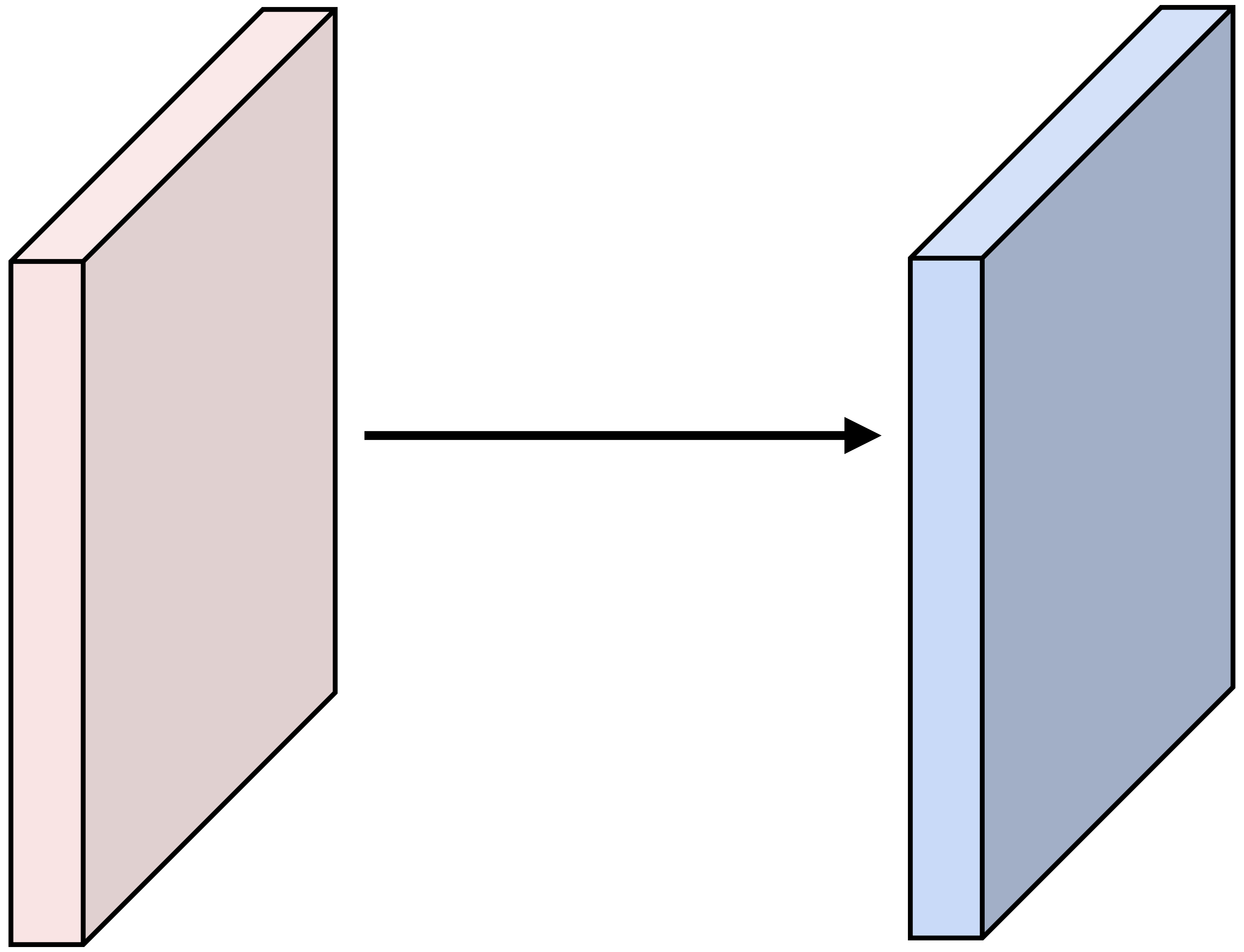
10 filters, so total is 10*76 = 760



Convolution Example

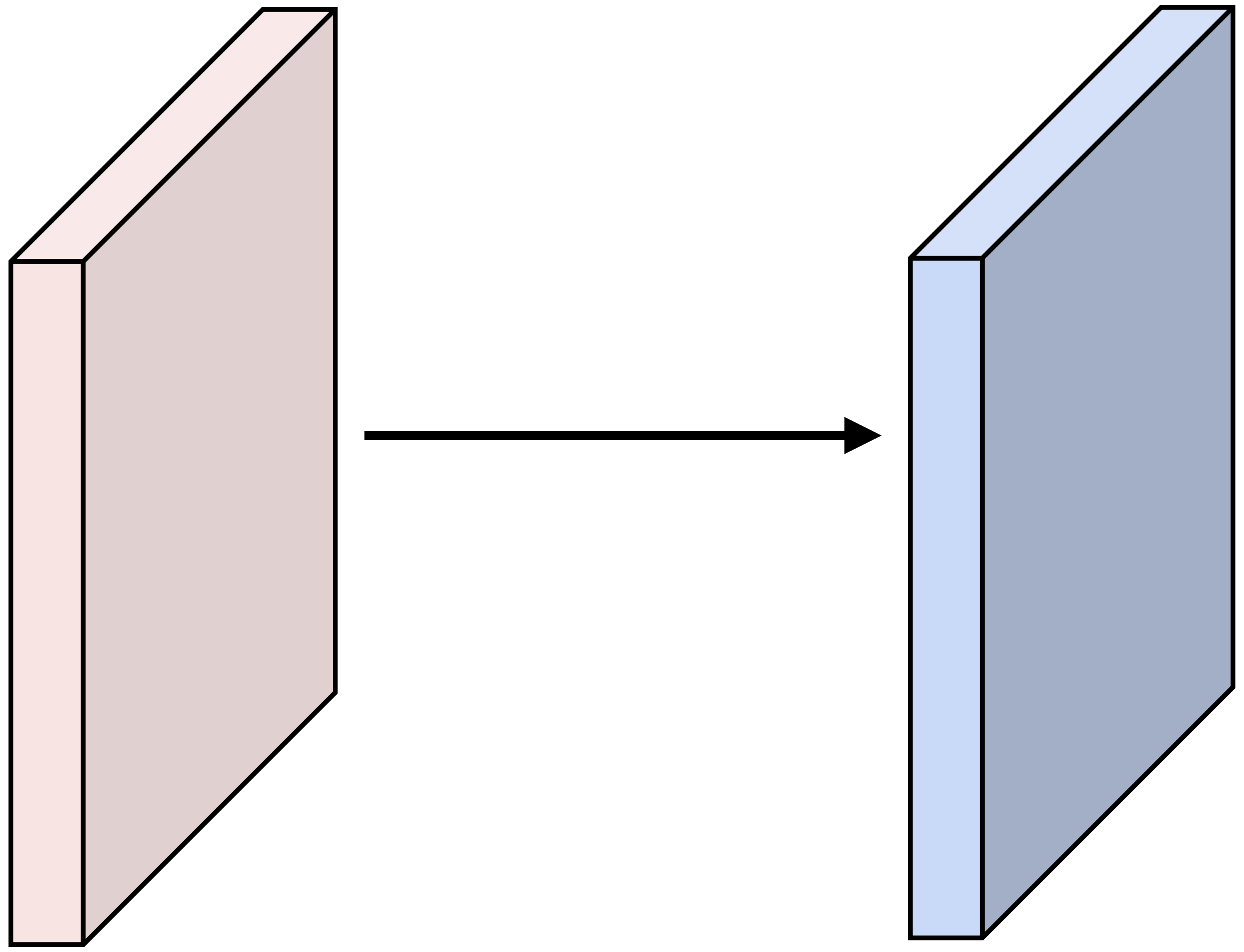
Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Q: What is the number of multiply-add operations?



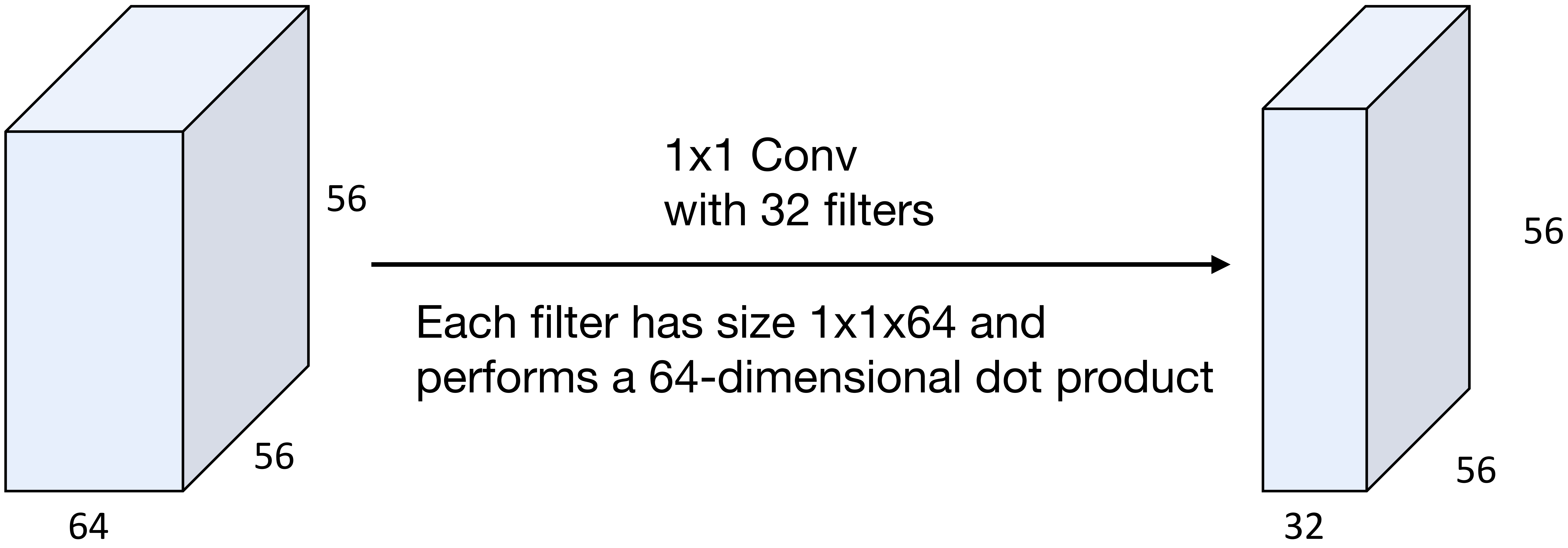
Convolution Example

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

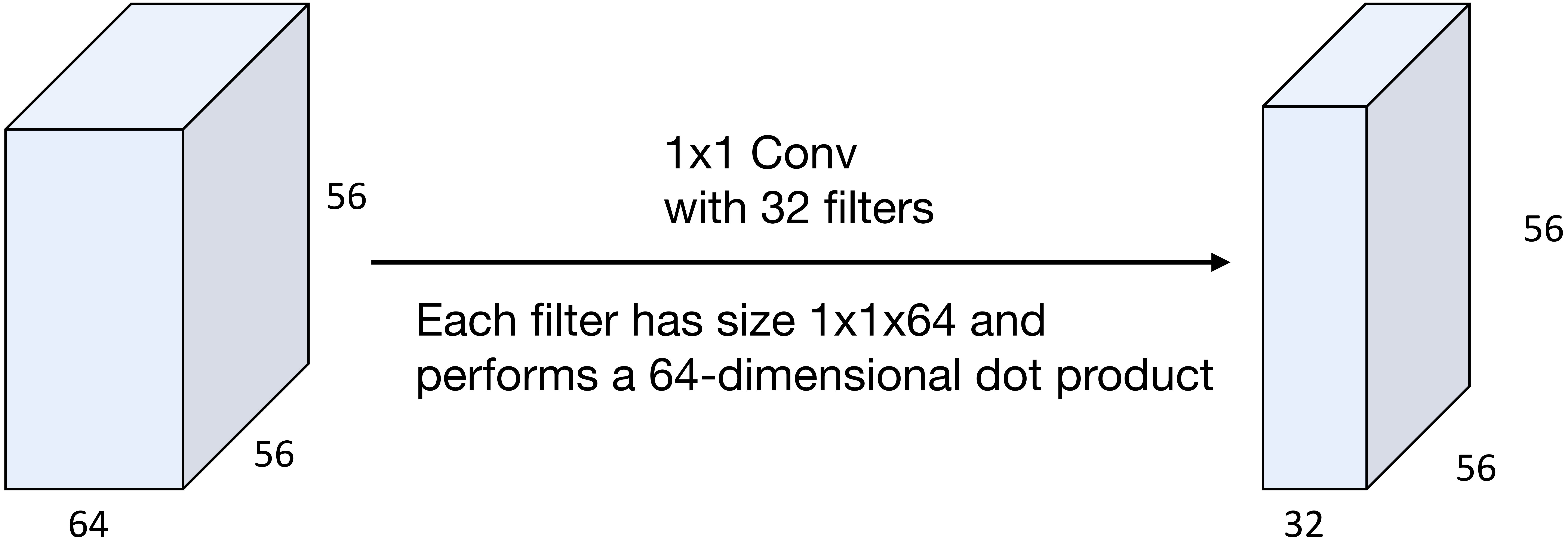


Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Q: What is the number of multiply-add operations?
10*32*32=10,240 outputs, each from inner product
of two 3x5x5 tensors, so total = 75 * 10,240 = **768,000**

Example: 1x1 Convolution



Example: 1x1 Convolution



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

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}

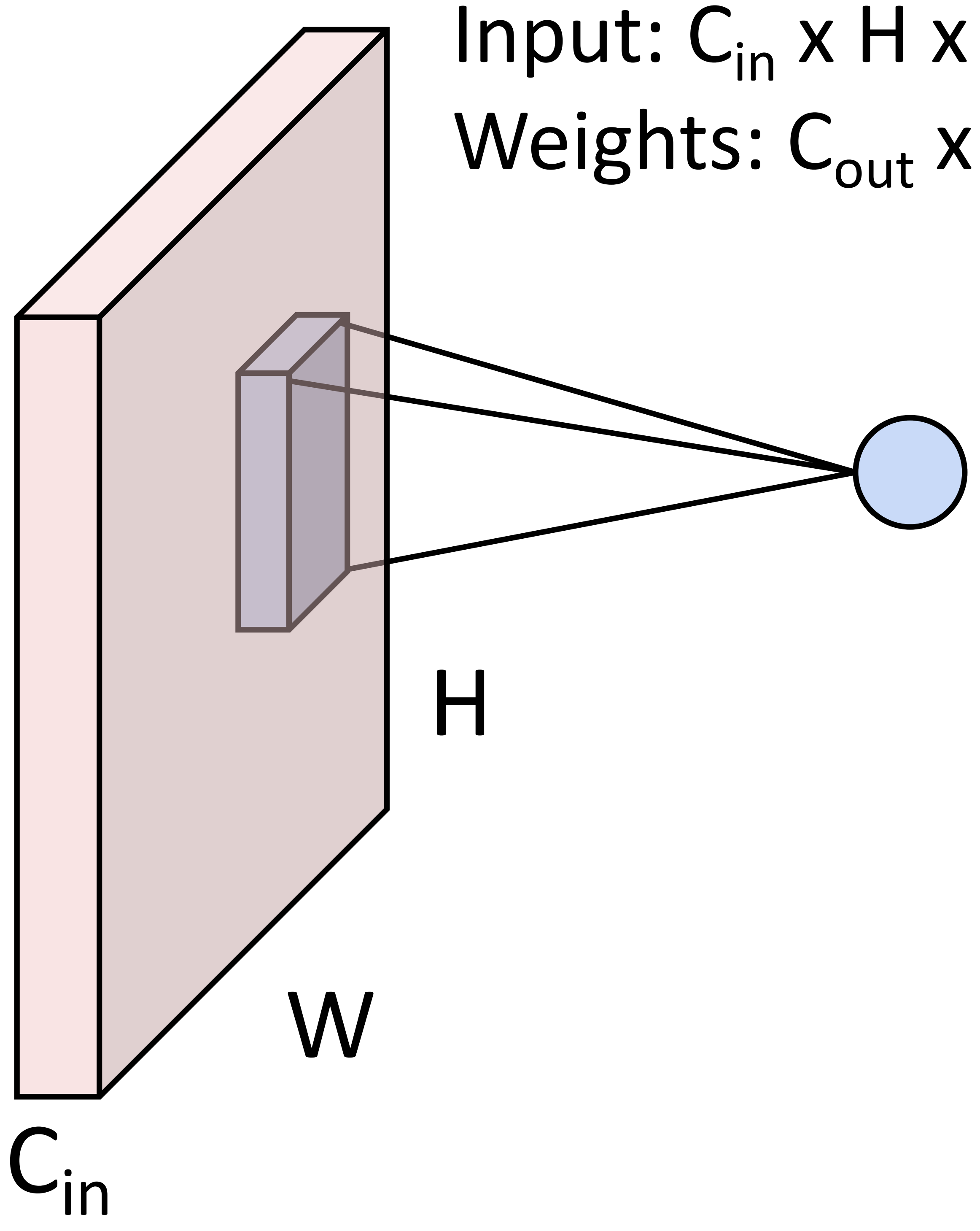
Output size: C_{out} x H' x W' where:
- H' = (H – K + 2P) / S + 1
- W' = (W – K + 2P) / S + 1

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

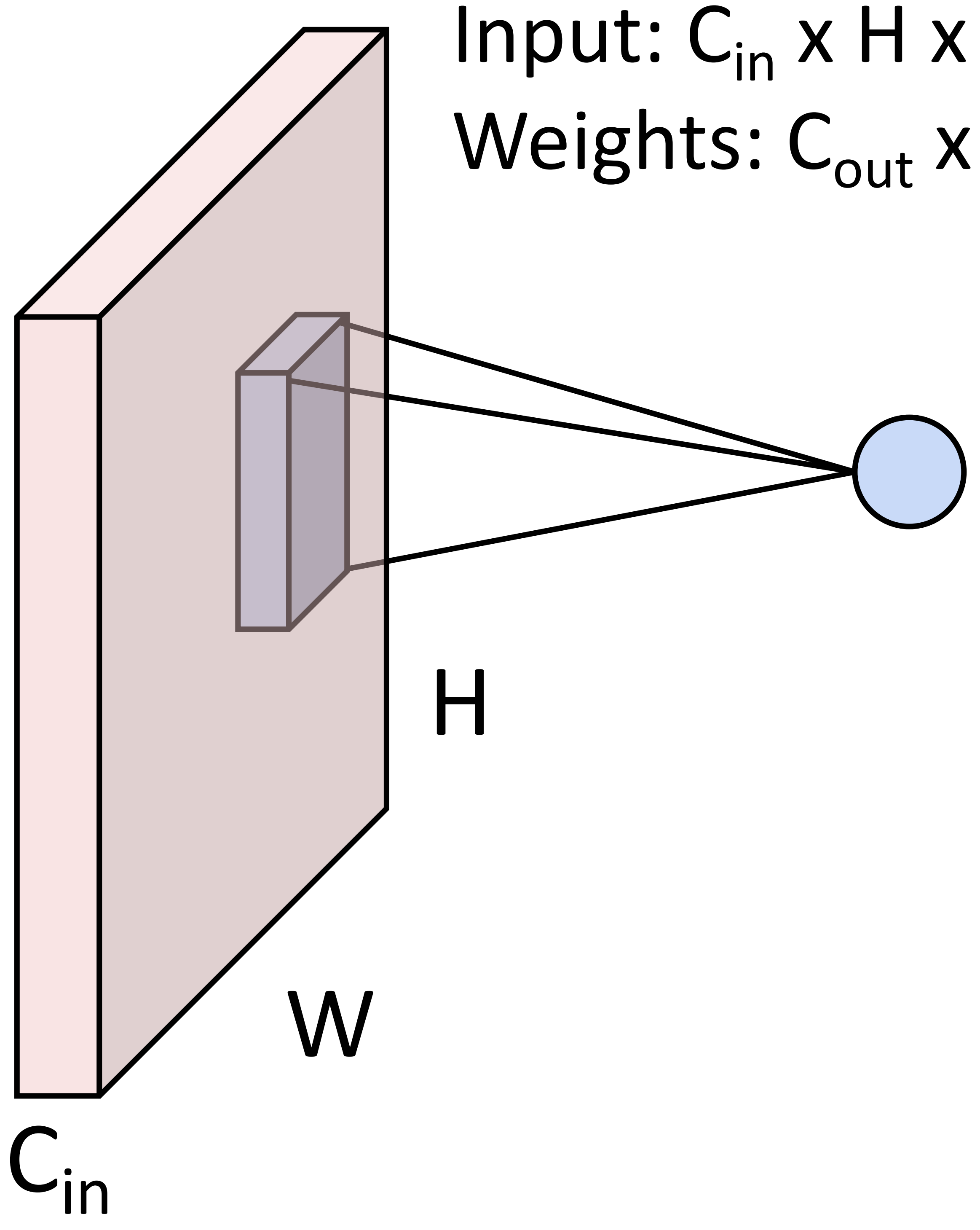
Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



Other types of convolution

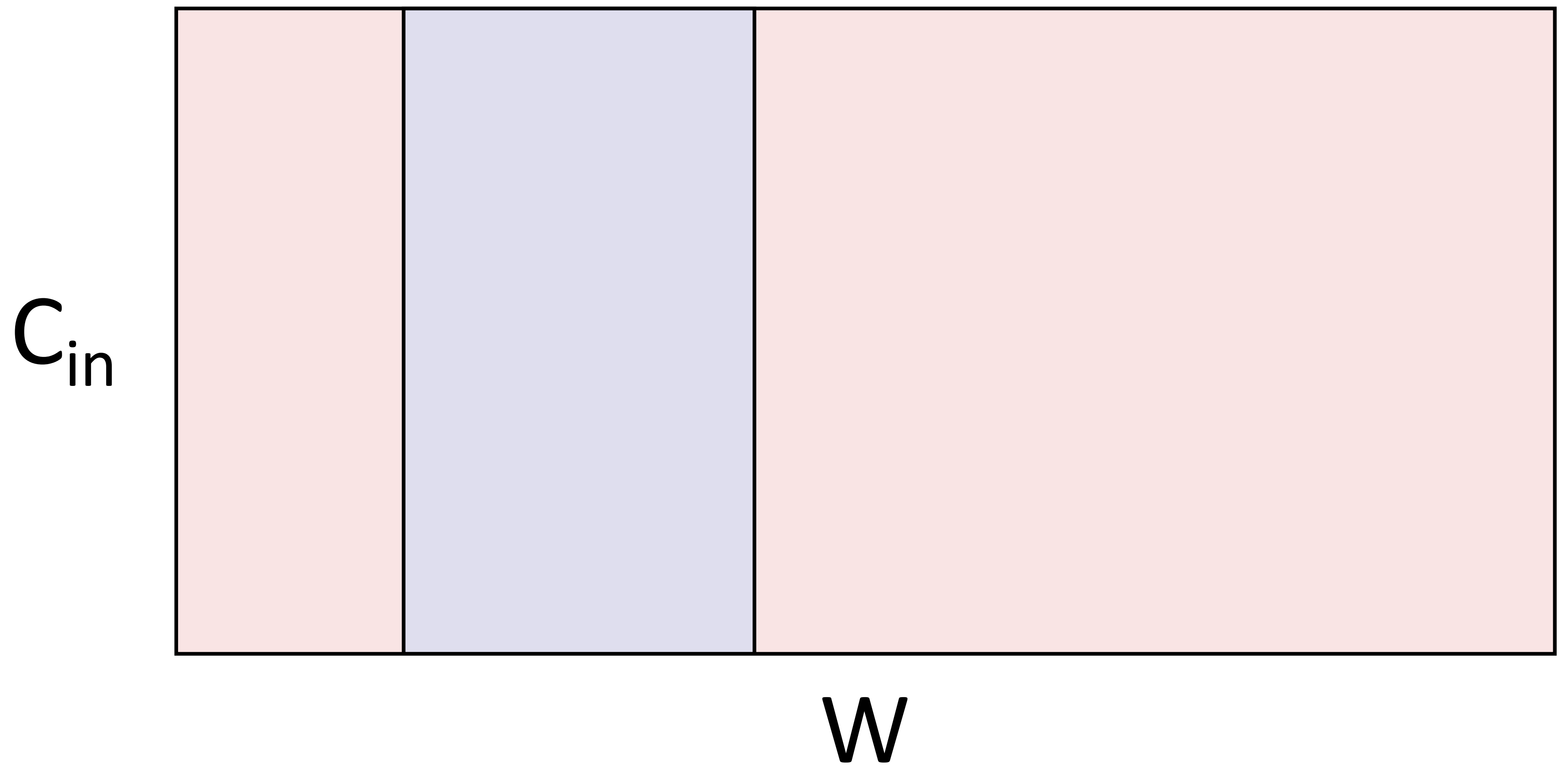
So far: 2D Convolution

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



1D Convolution

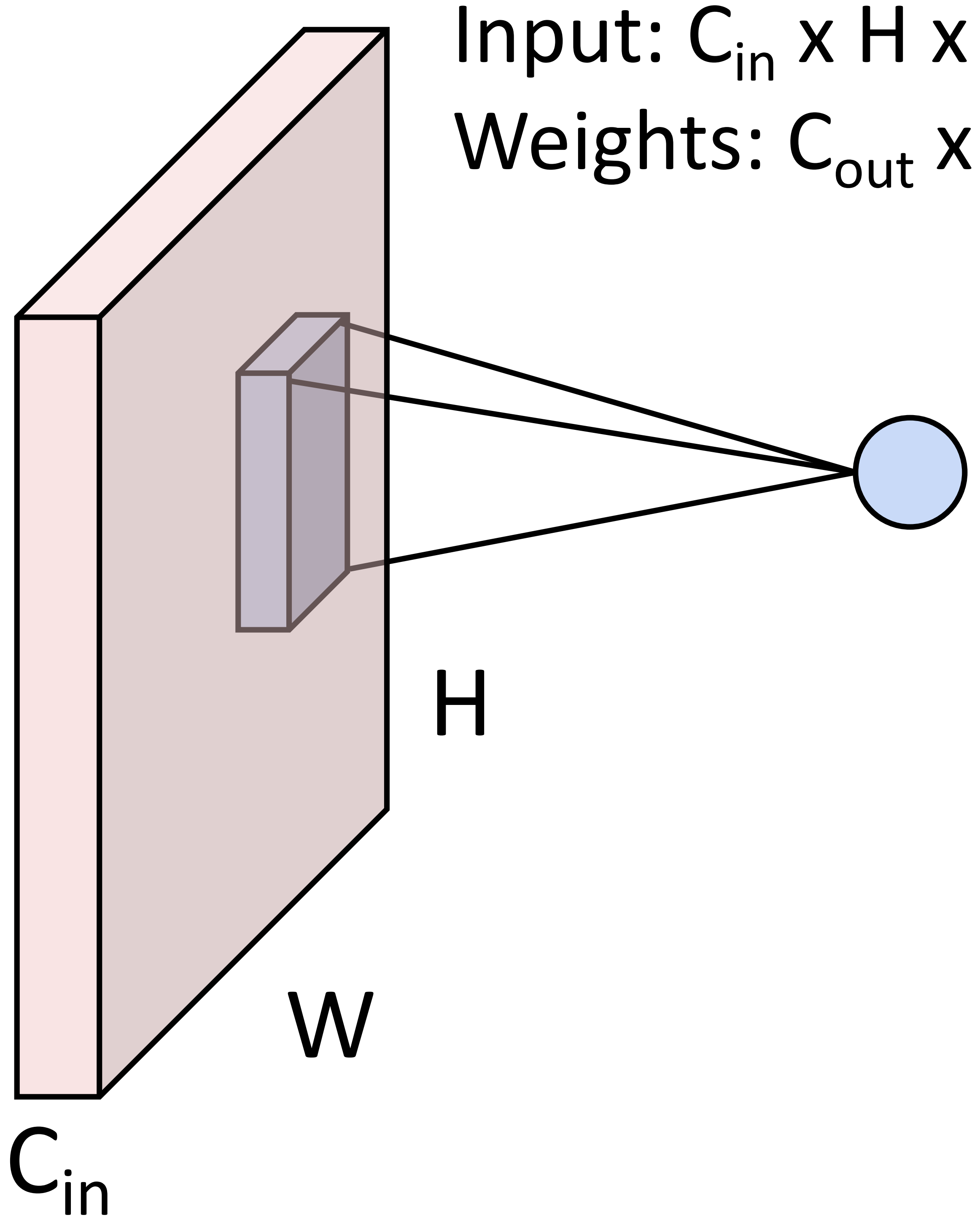
Input: C_{in} x W
Weights: C_{out} x C_{in} x K



Other types of convolution

So far: 2D Convolution

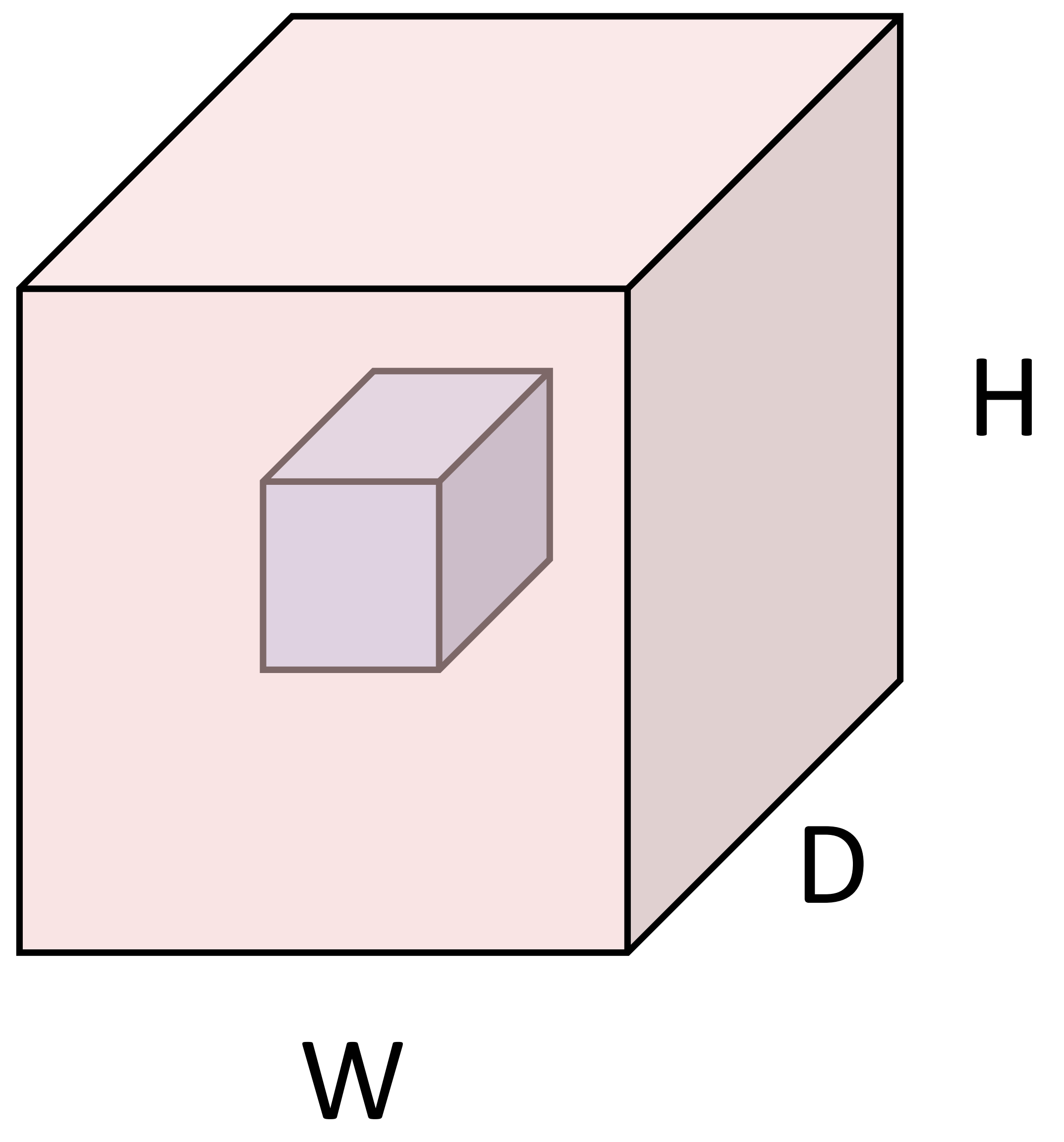
Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



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



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 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]

Conv1d

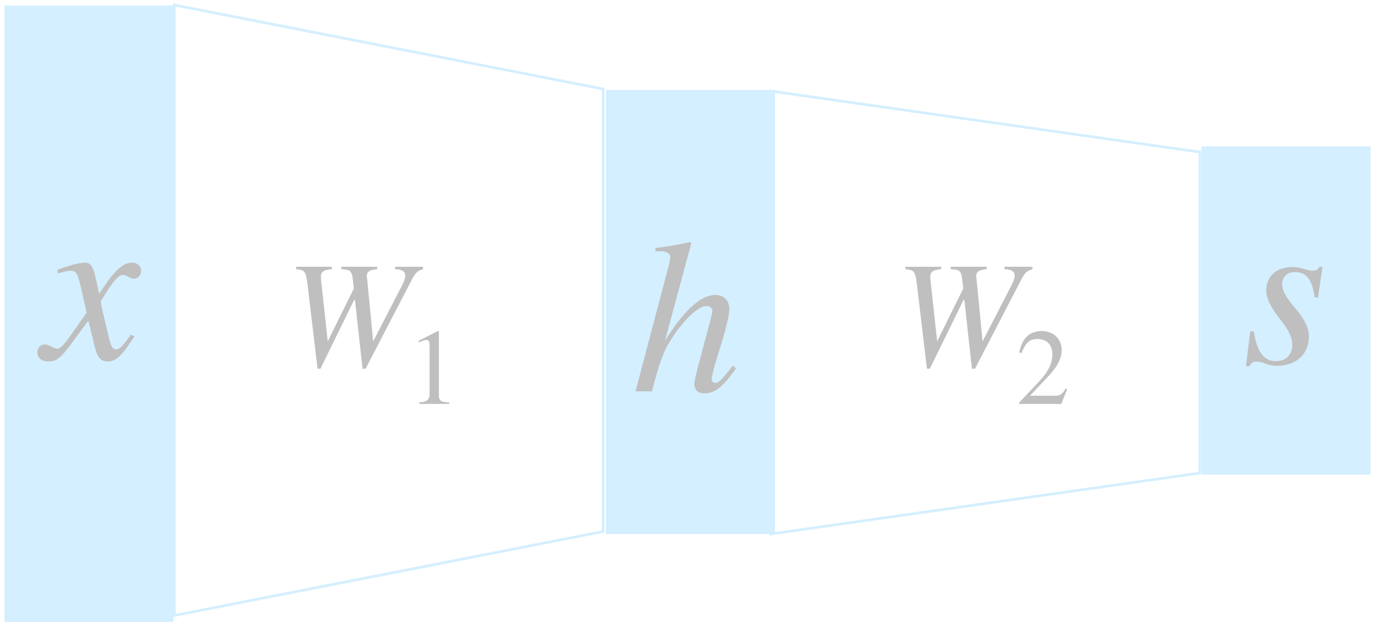
CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Conv3d

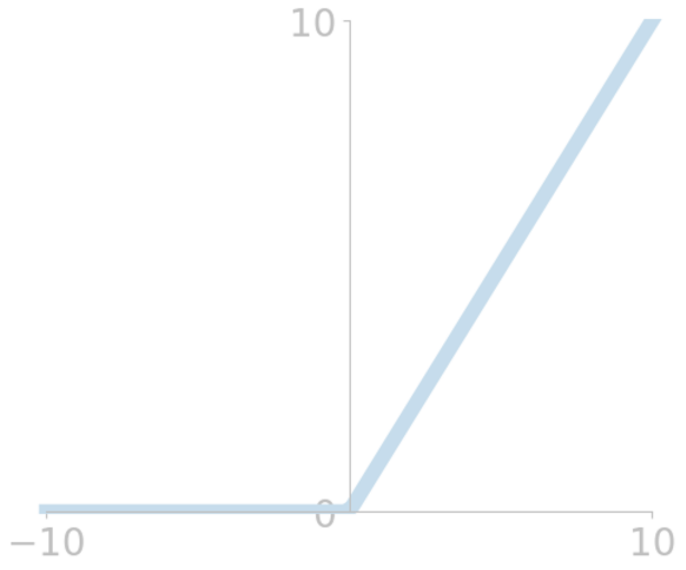
CLASS `torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Components of Convolutional Neural Networks

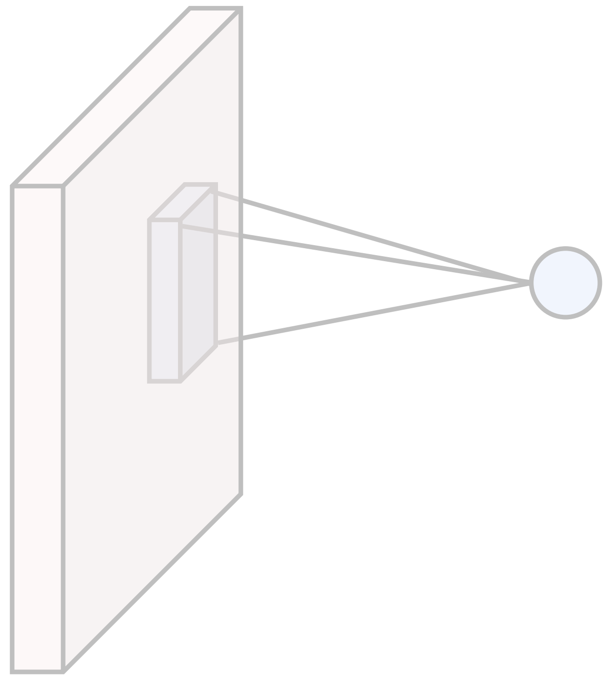
Fully-Connected Layers



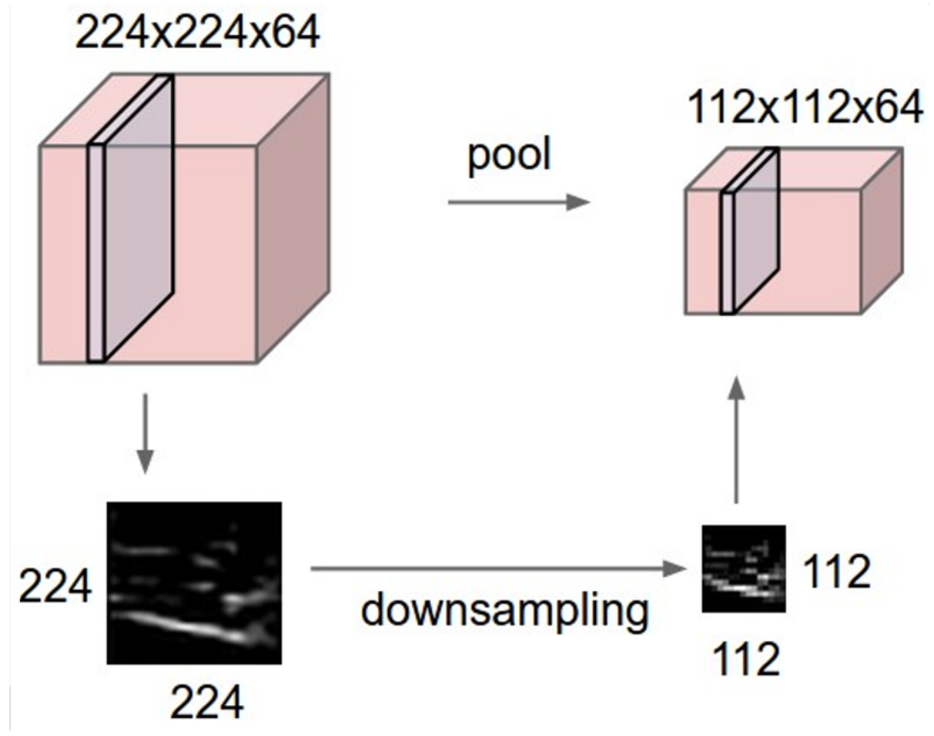
Activation Functions



Convolution Layers



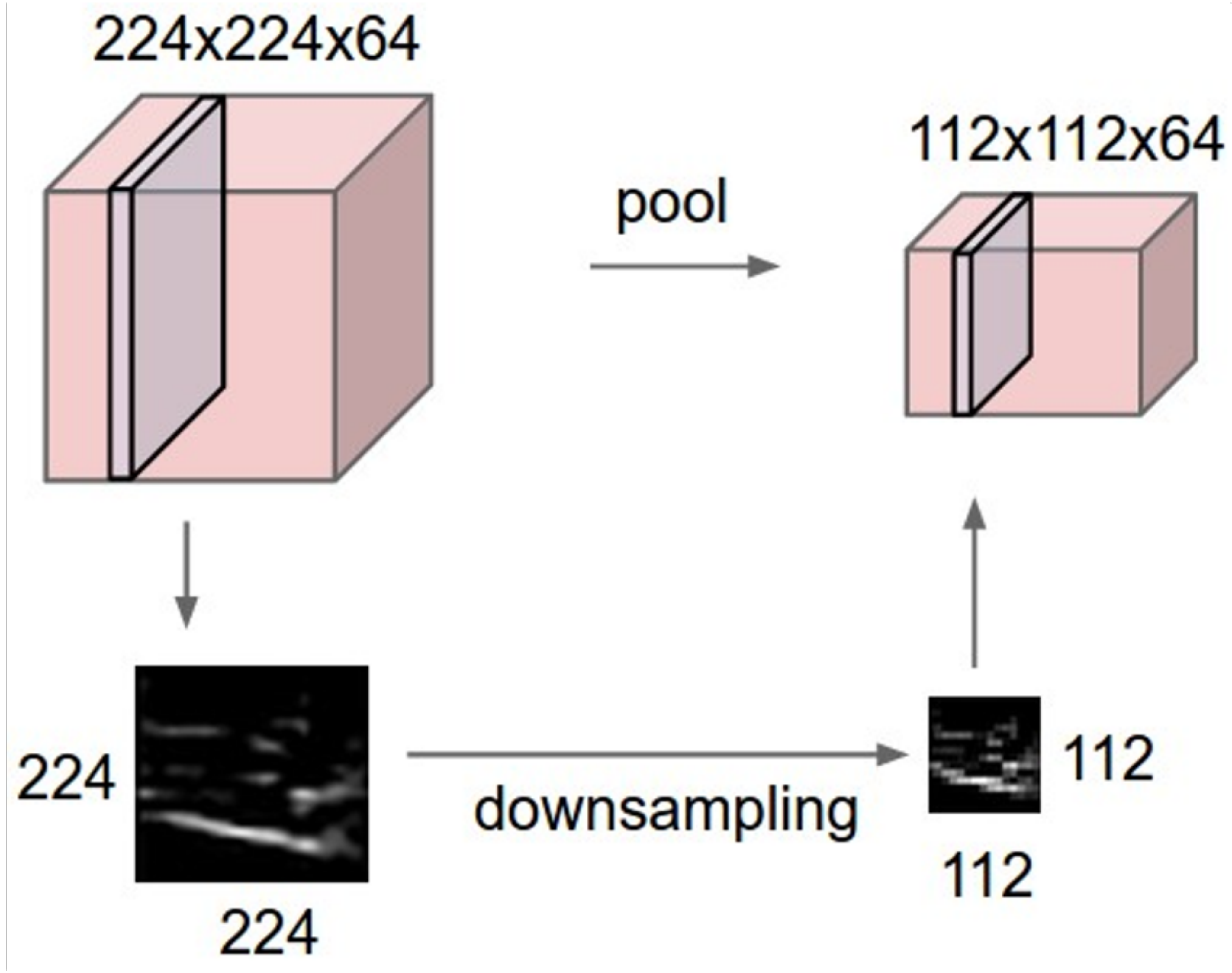
Pooling Layers



Normalization

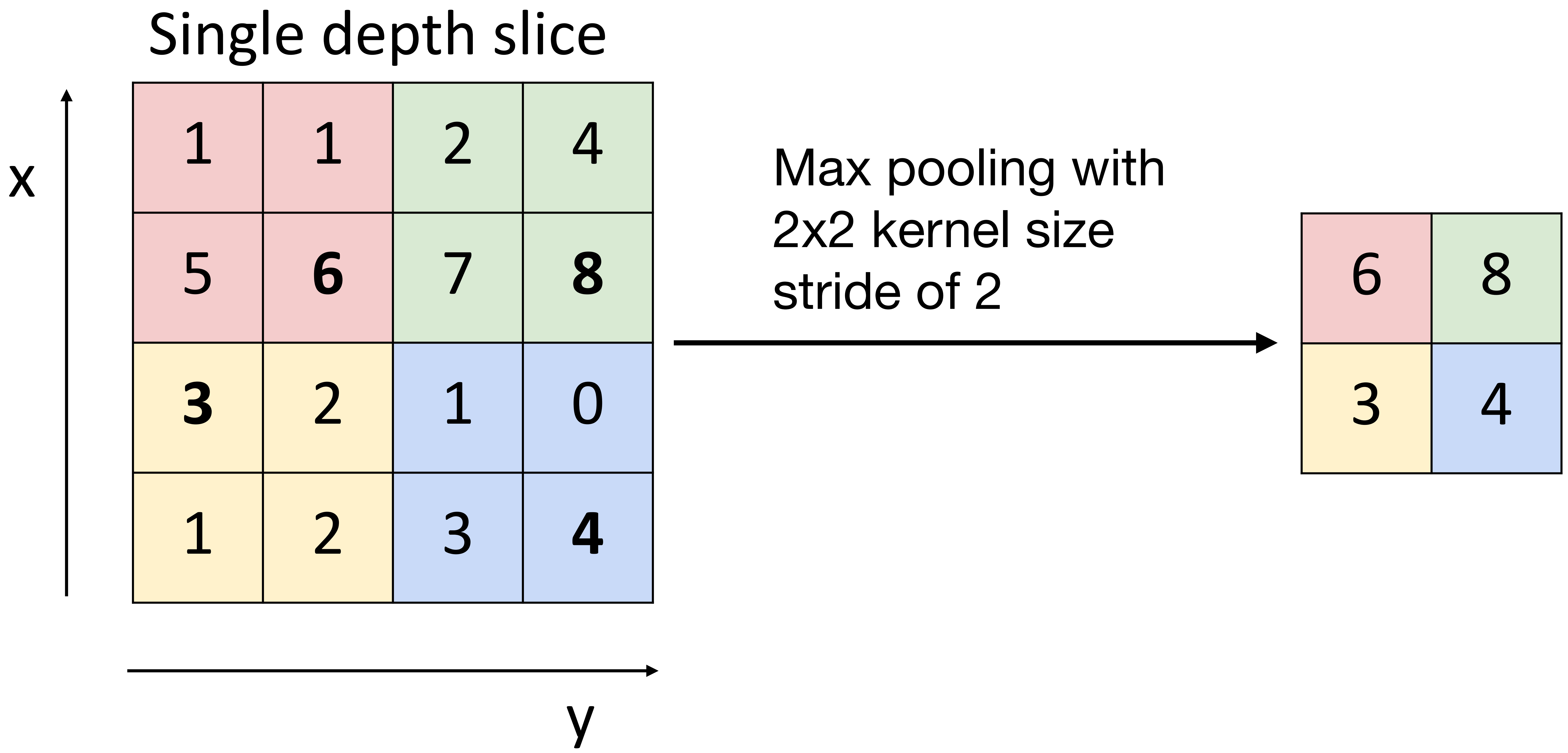
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Pooling Layers: Another way to downsample



Hyperparameters:
Kernel size
Stride
Pooling function

Max Pooling



Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
Max pooling with
2x2 kernel size
stride of 2
6 8
3 4

Max Pooling

Single depth slice

x

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

y

Max pooling with
2x2 kernel size
stride of 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)

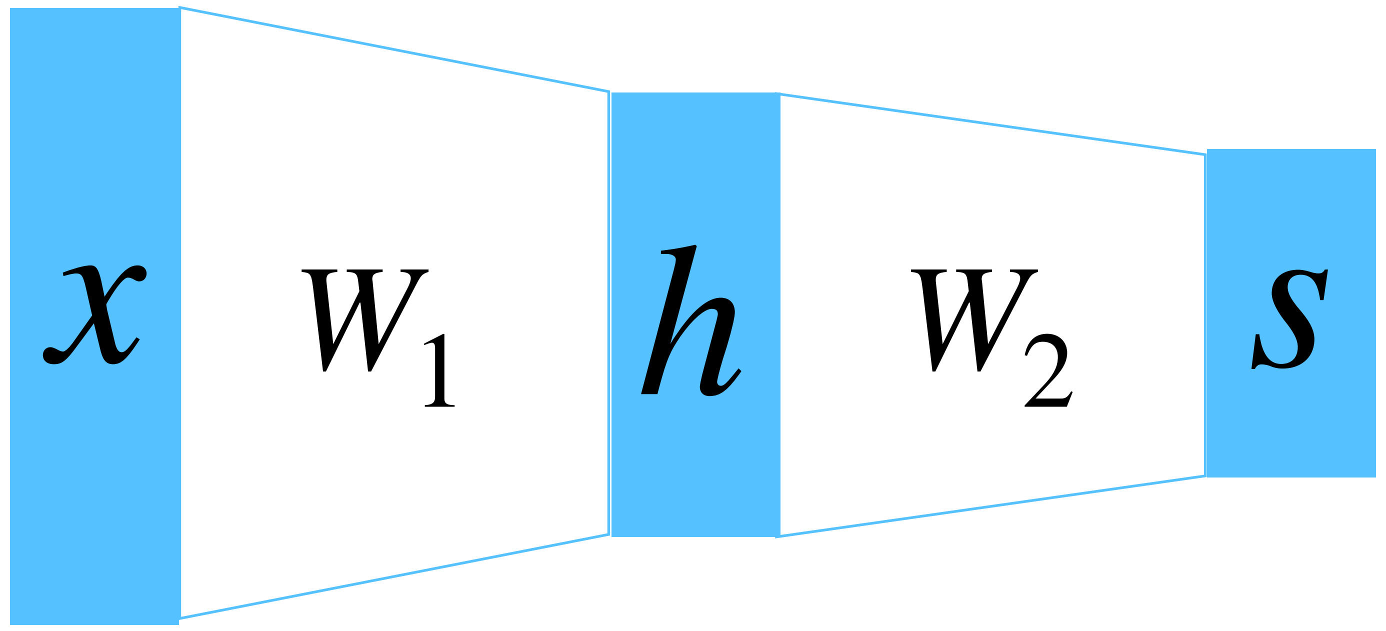
Output: C x H' x W' where
- H' = (H – K) / S + 1
- W' = (W – K) / S + 1

Learnable parameters: None!

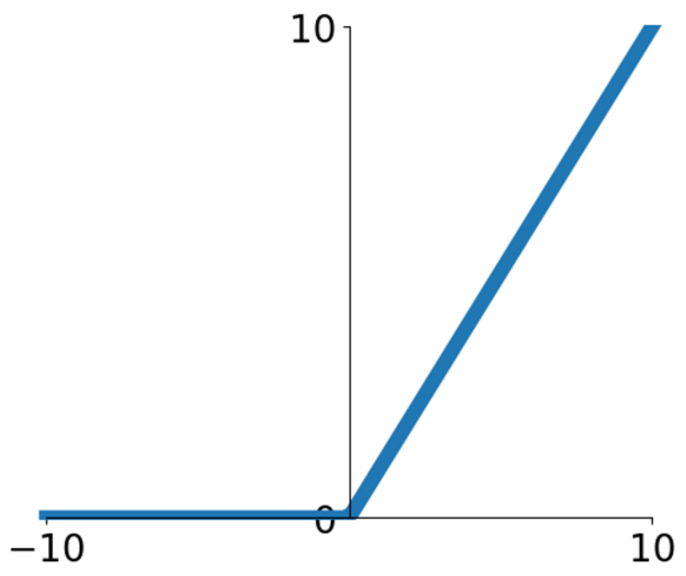
Common settings:
max, K = 2, S = 2
max, K = 3, S = 2 (AlexNet)

Components of Convolutional Neural Networks

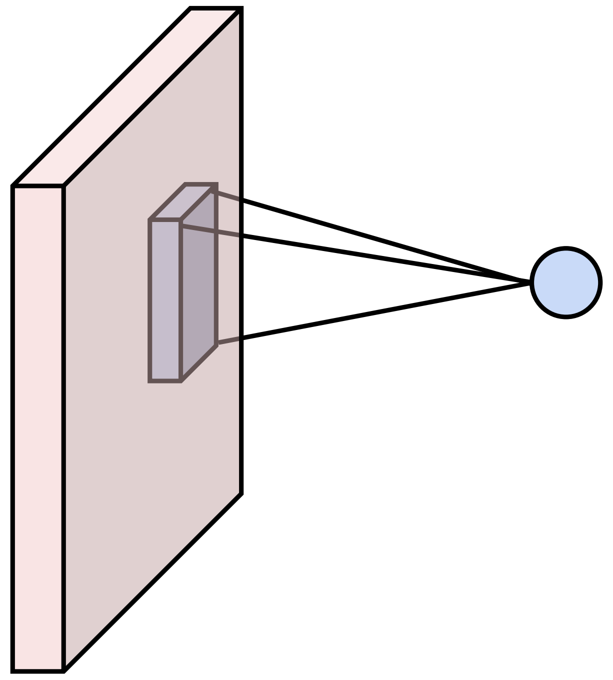
Fully-Connected Layers



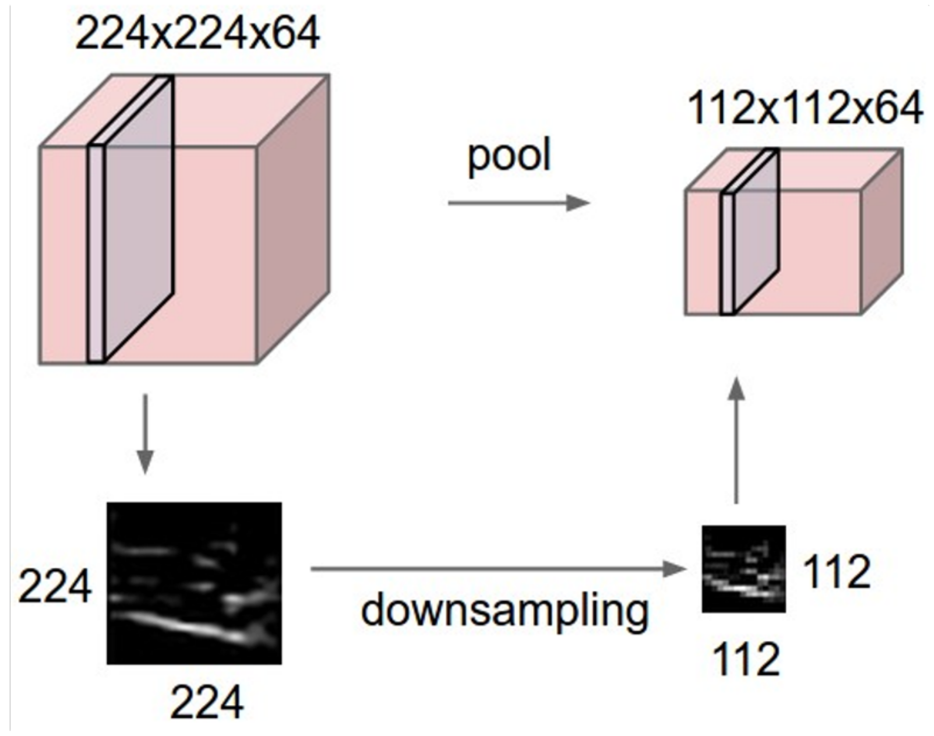
Activation Functions



Convolution Layers



Pooling Layers



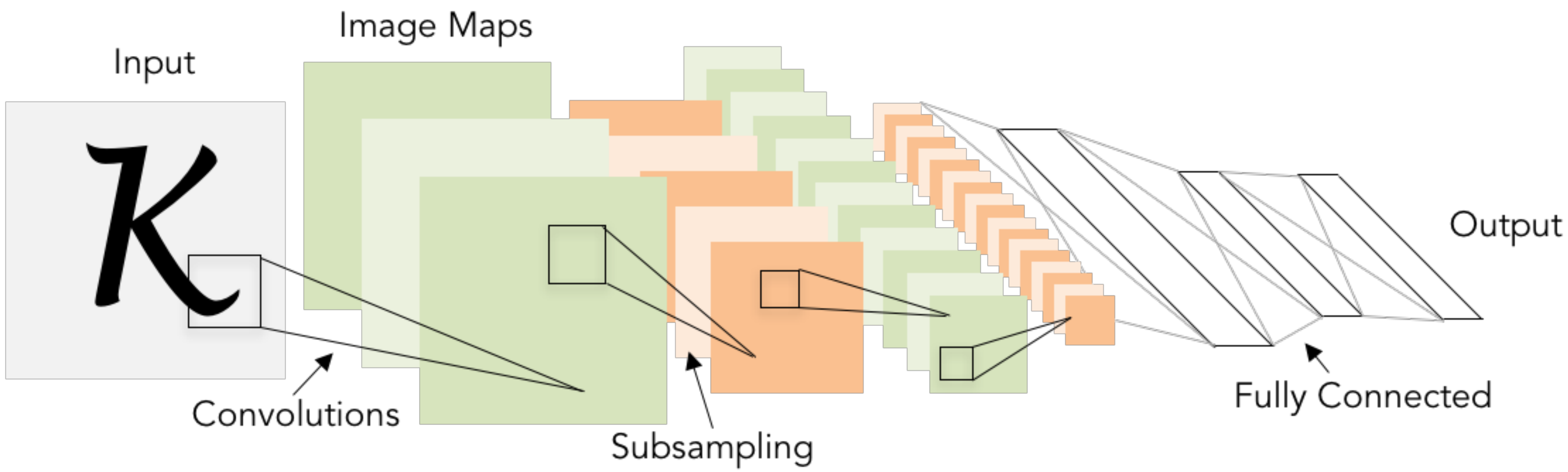
Normalization

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

Convolutional Neural Networks

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

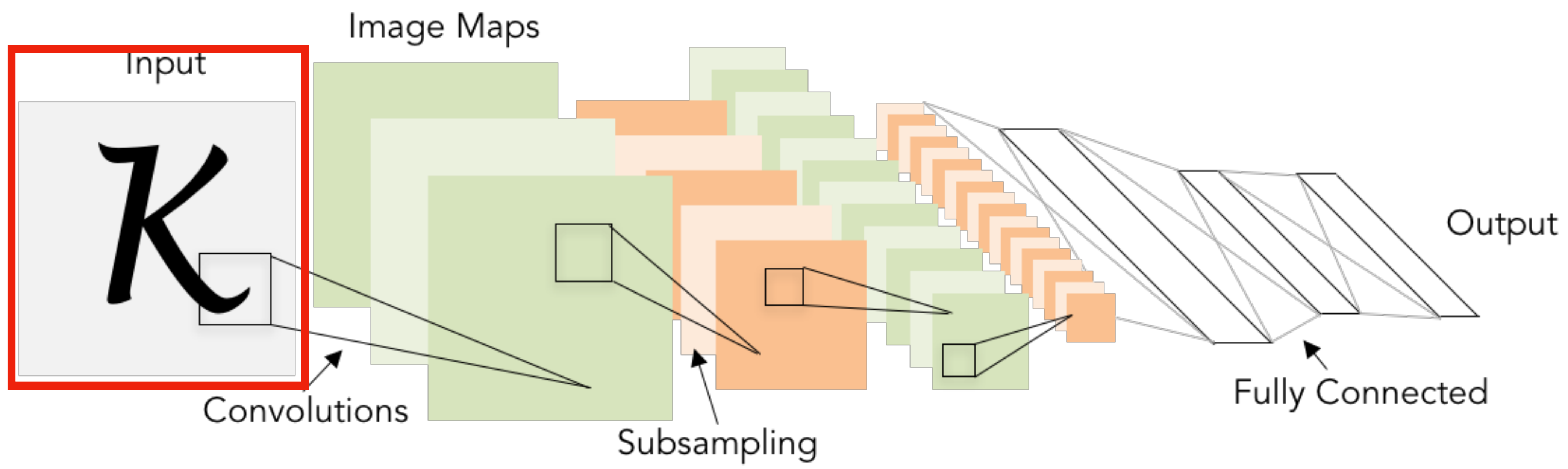
Example: LeNet-5



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

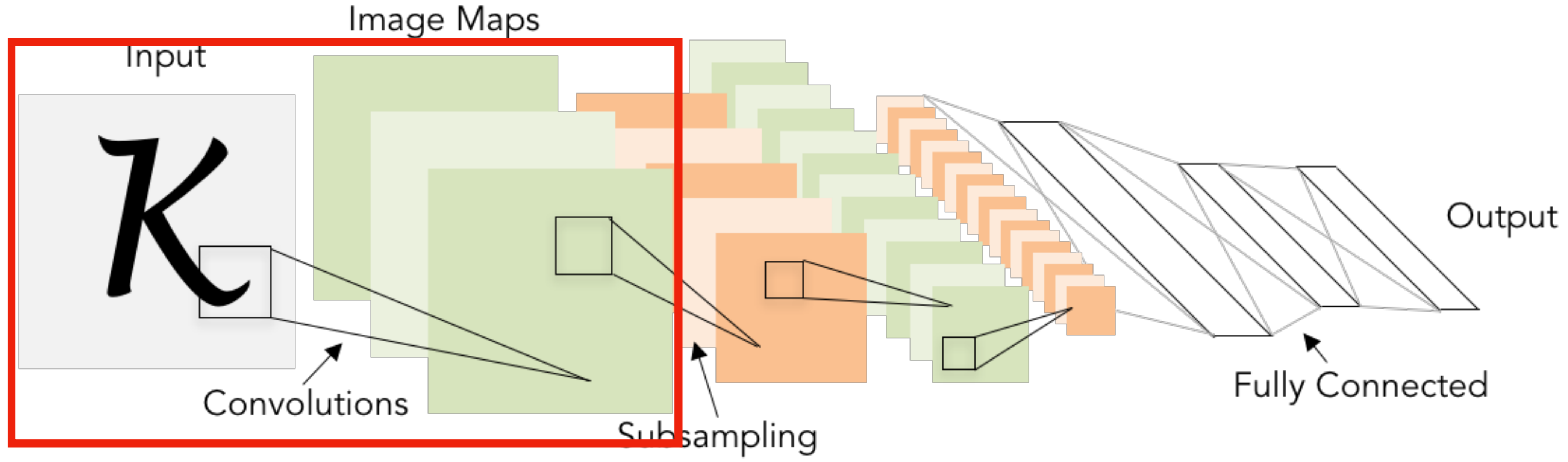
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	



Example: LeNet-5

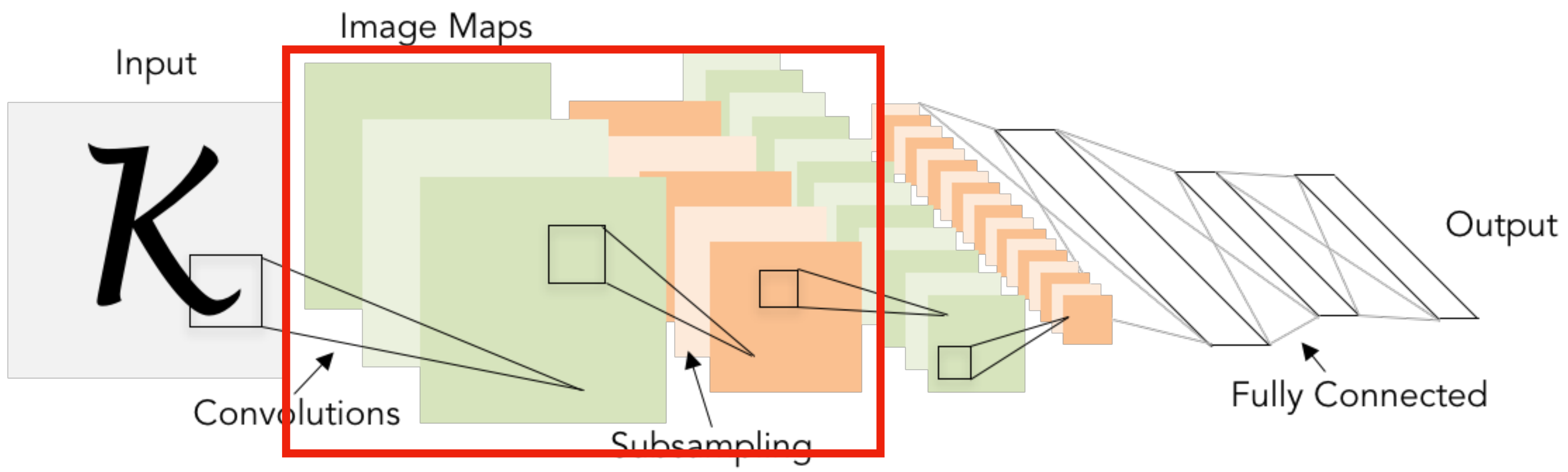
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	



Lecun et al., “Gradient-based learning applied to document recognition”, 1998

Example: LeNet-5

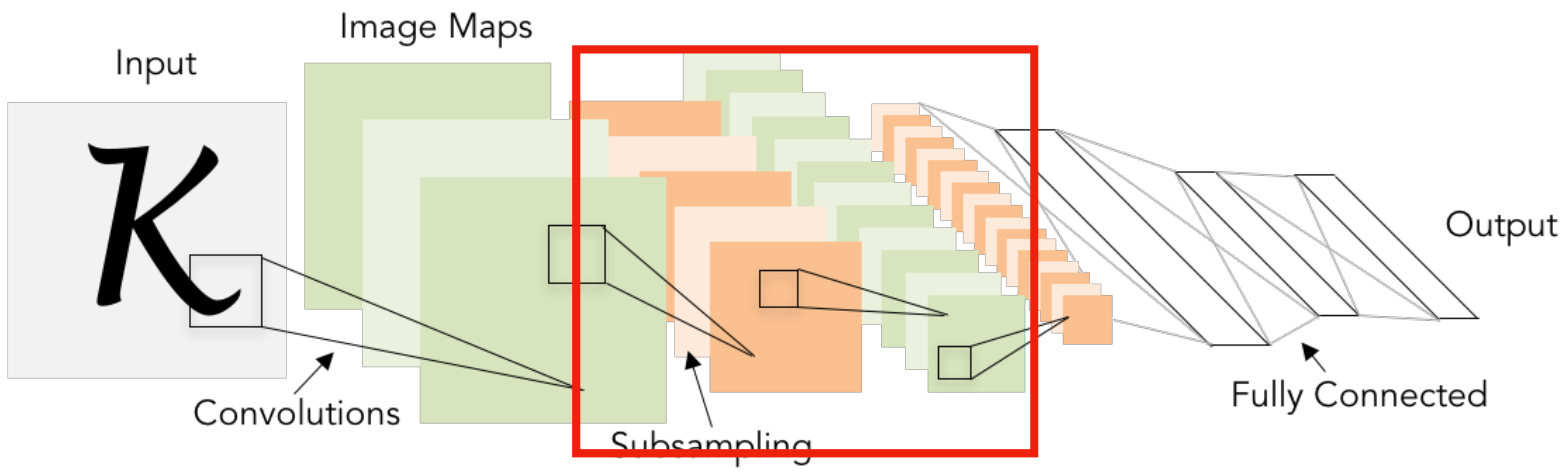
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	



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

Example: LeNet-5

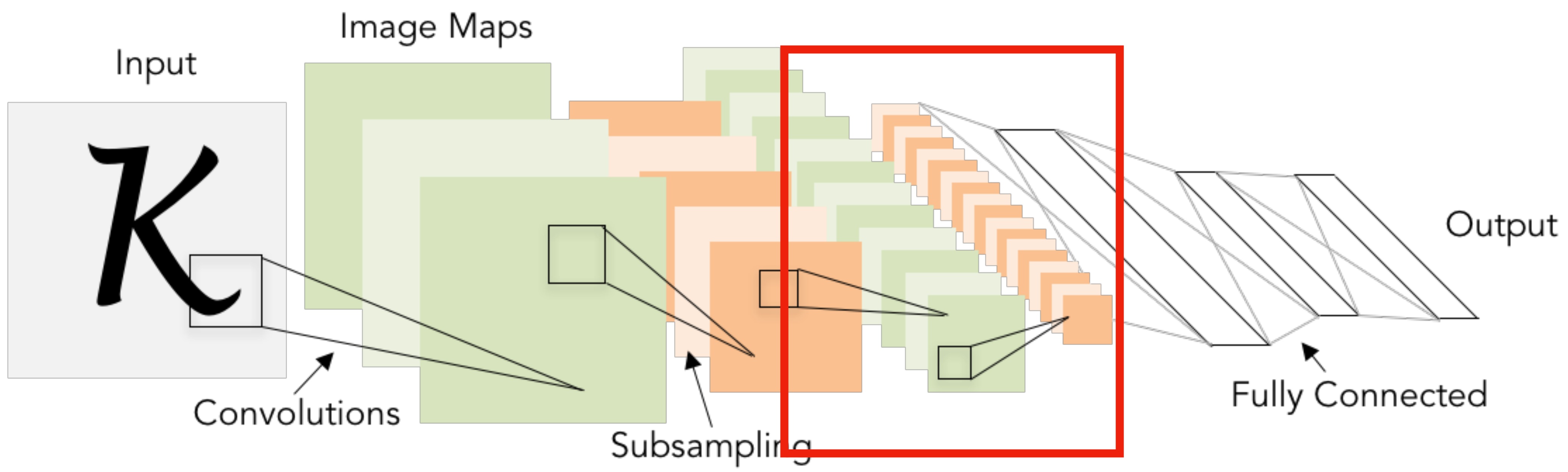
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	



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

Example: LeNet-5

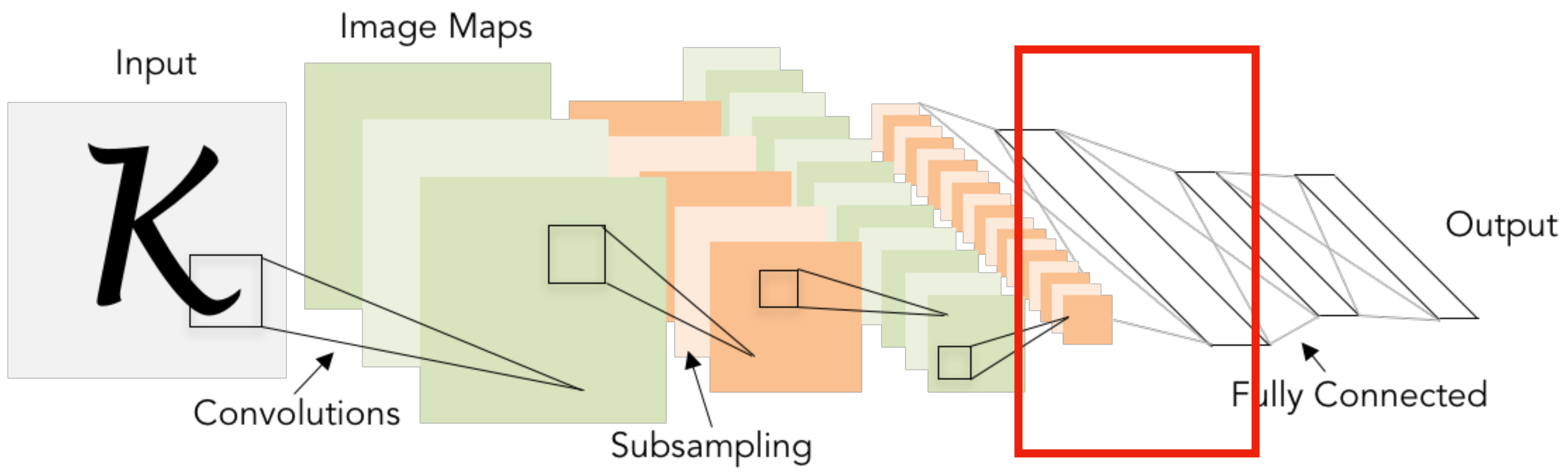
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	



Lecun et al., “Gradient-based learning applied to document recognition”, 1998

Example: LeNet-5

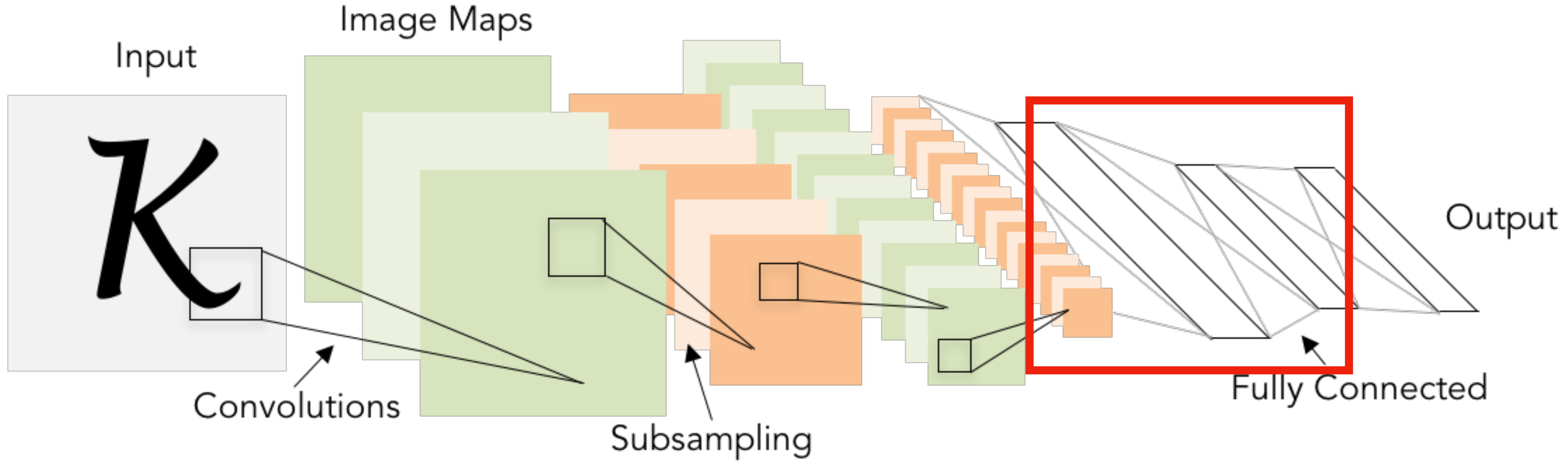
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	



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

Example: LeNet-5

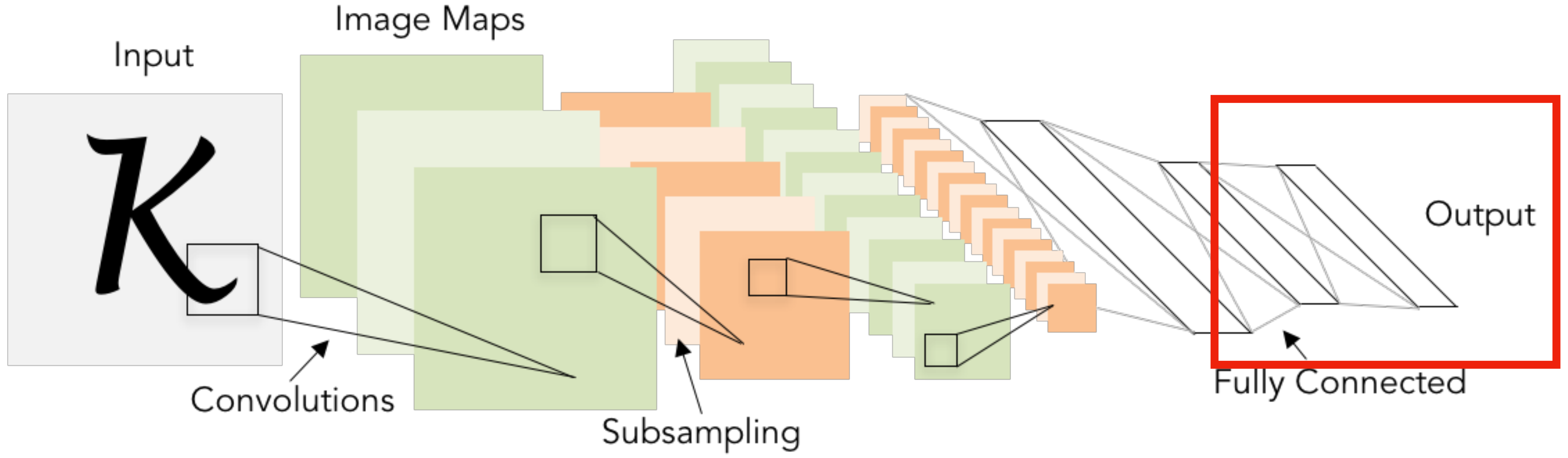
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 ->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



Lecun et al., “Gradient-based learning applied to document recognition”, 1998

Example: LeNet-5

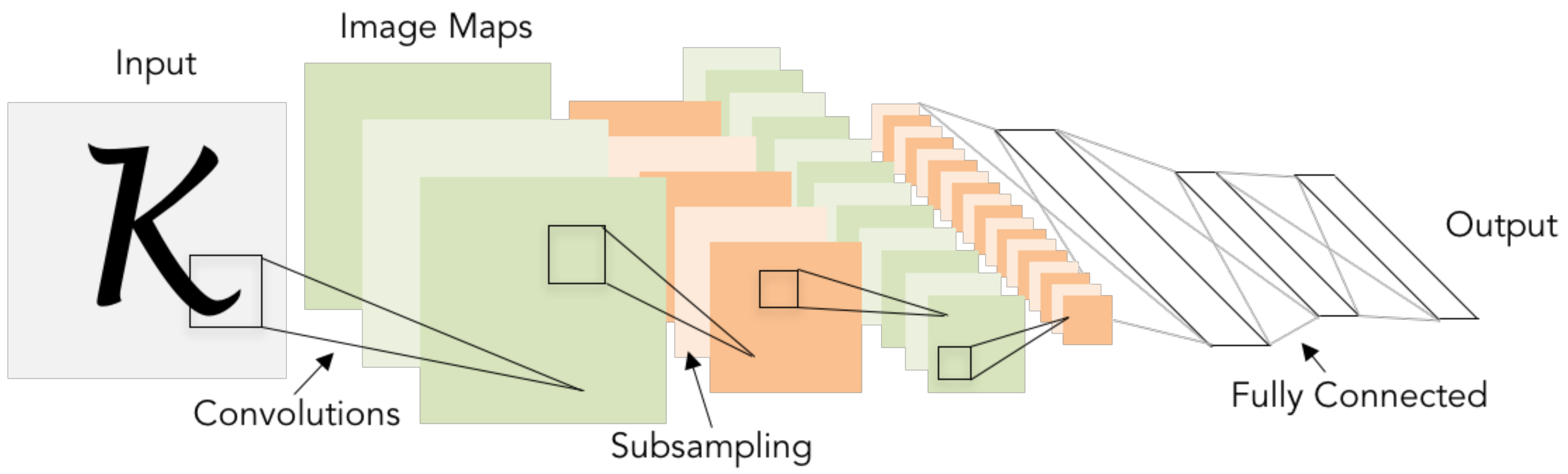
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 ->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

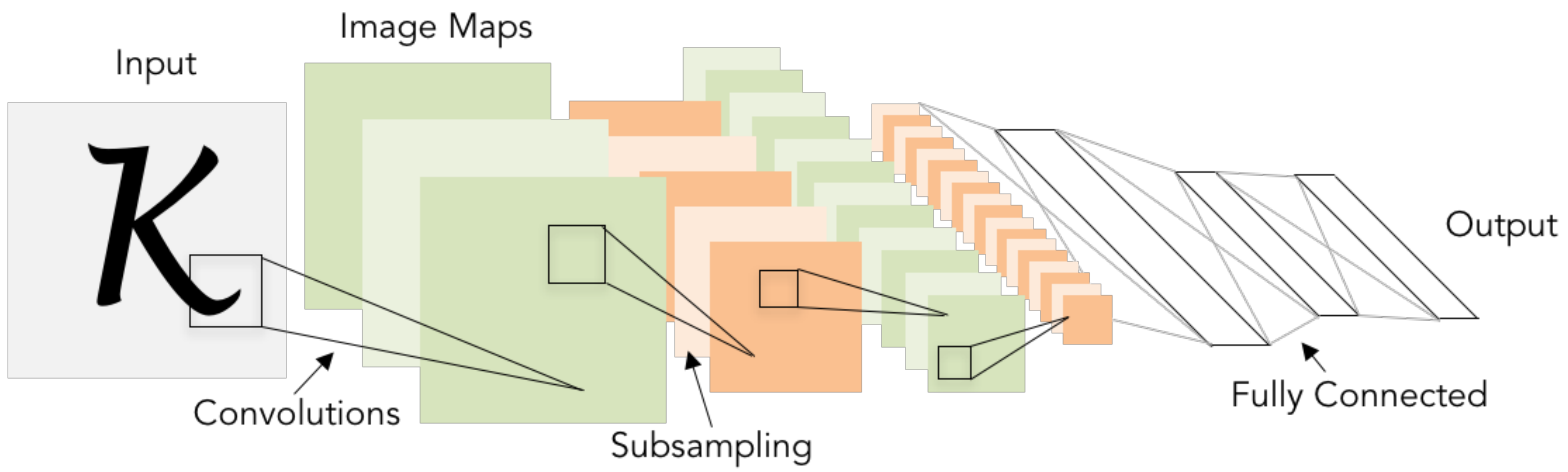
Spatial size **decreases**
(using pooling or striped convolution)

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

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 progress through the network:

Spatial size **decreases**
(using pooling or striped convolution)

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

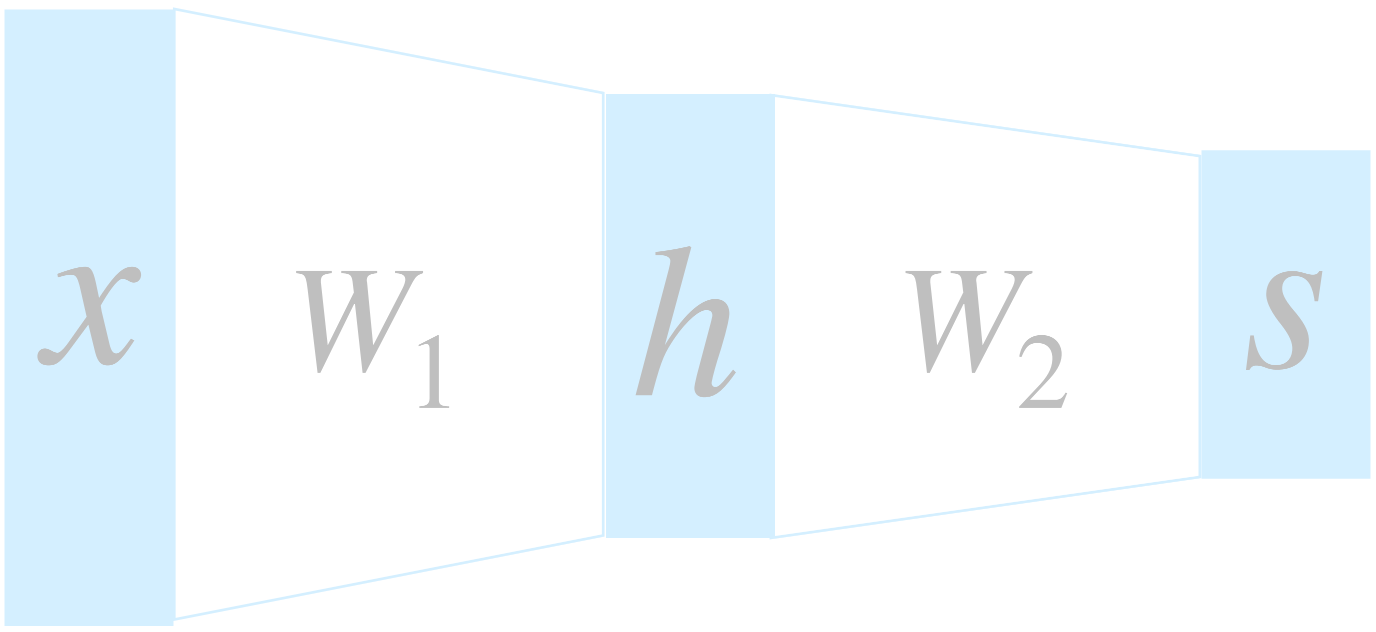
Some modern architectures
break this trend—stay tuned!

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

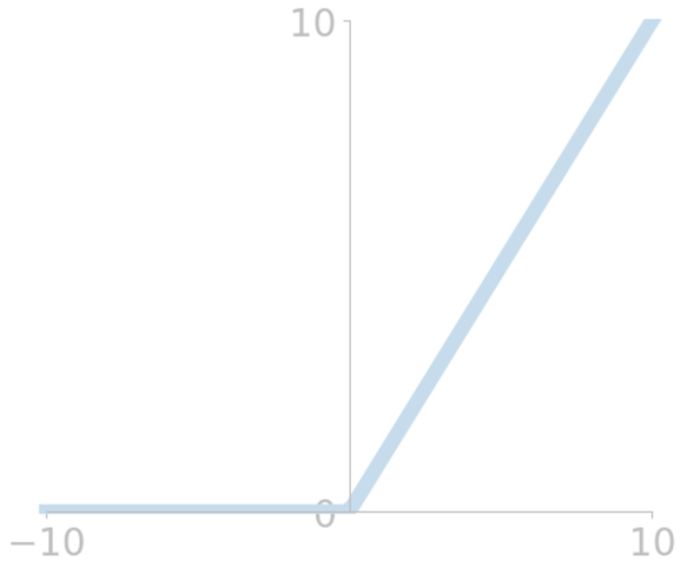
Problem: Deep Networks very hard to train

Components of Convolutional Neural Networks

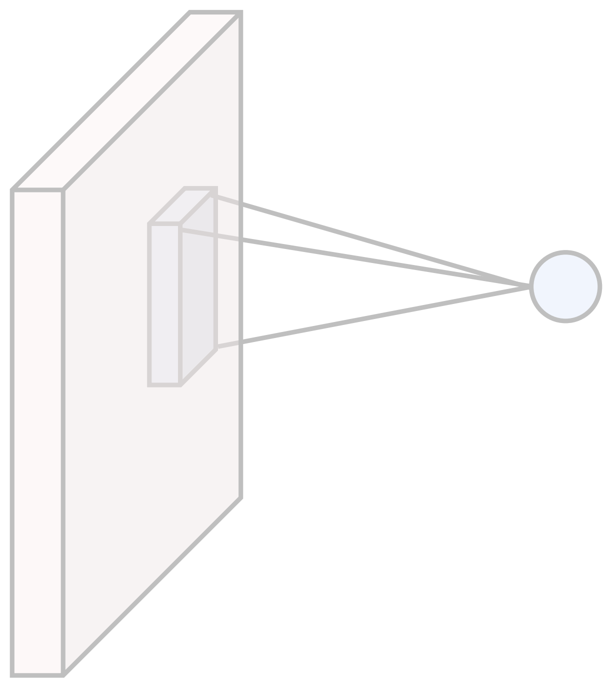
Fully-Connected Layers



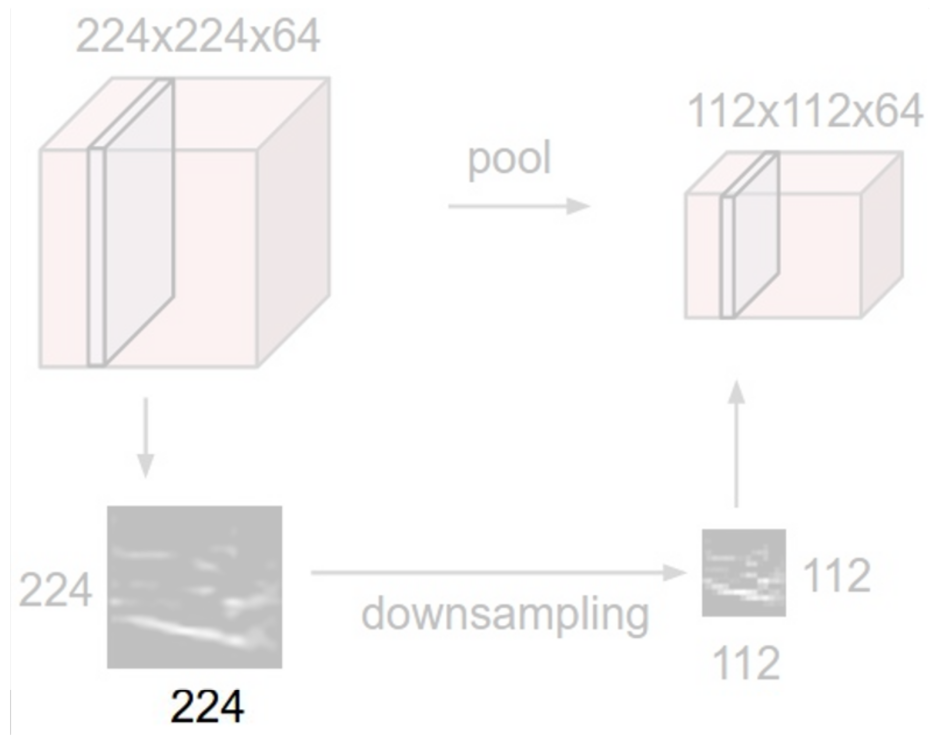
Activation Functions



Convolution Layers



Pooling Layers



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 results

We can normalize a batch of activations using:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

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

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 results

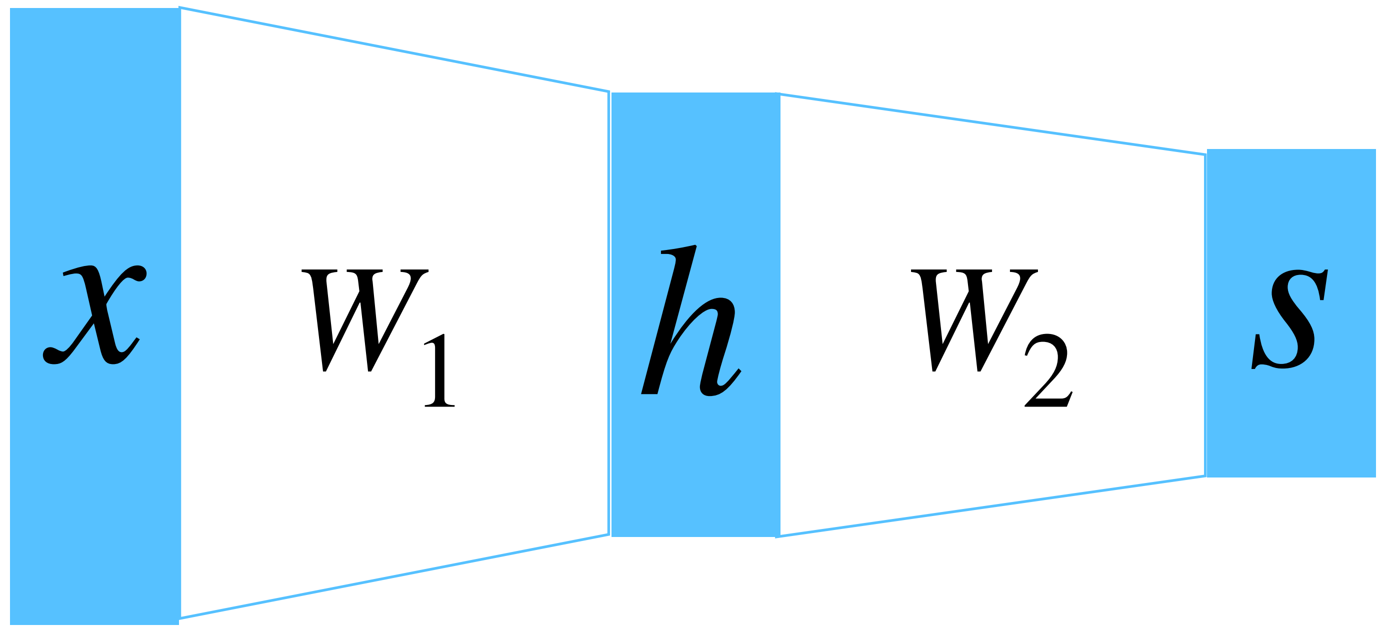
We can normalize a batch of activations using:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

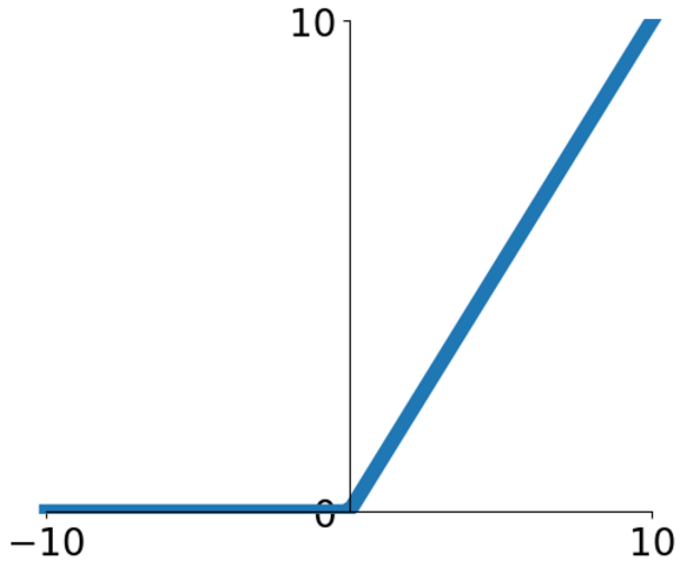
This is a **differentiable function**, so we can use it as an operator in our networks and backdrop through it!

Summary: Components of Convolutional Network

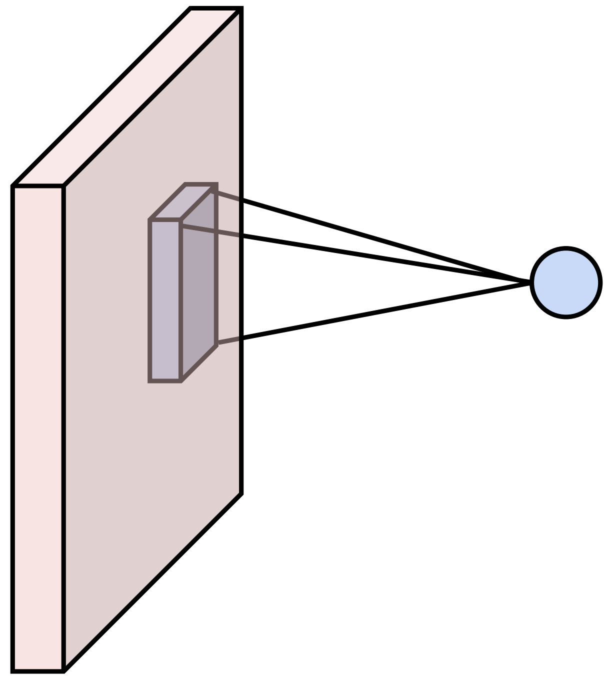
Fully-Connected Layers



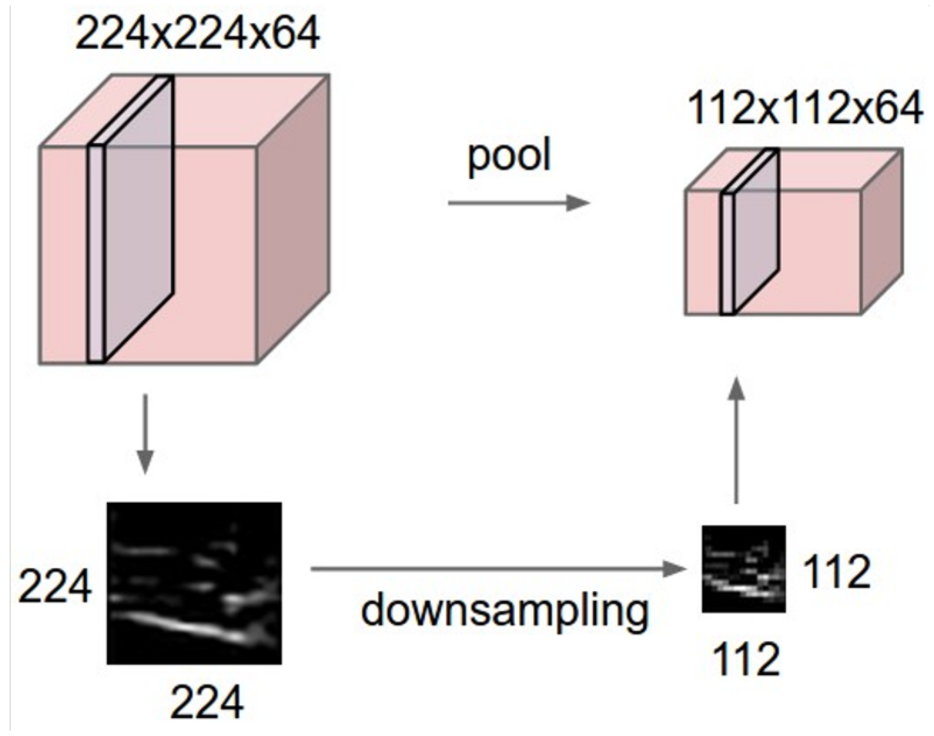
Activation Functions



Convolution Layers



Pooling Layers

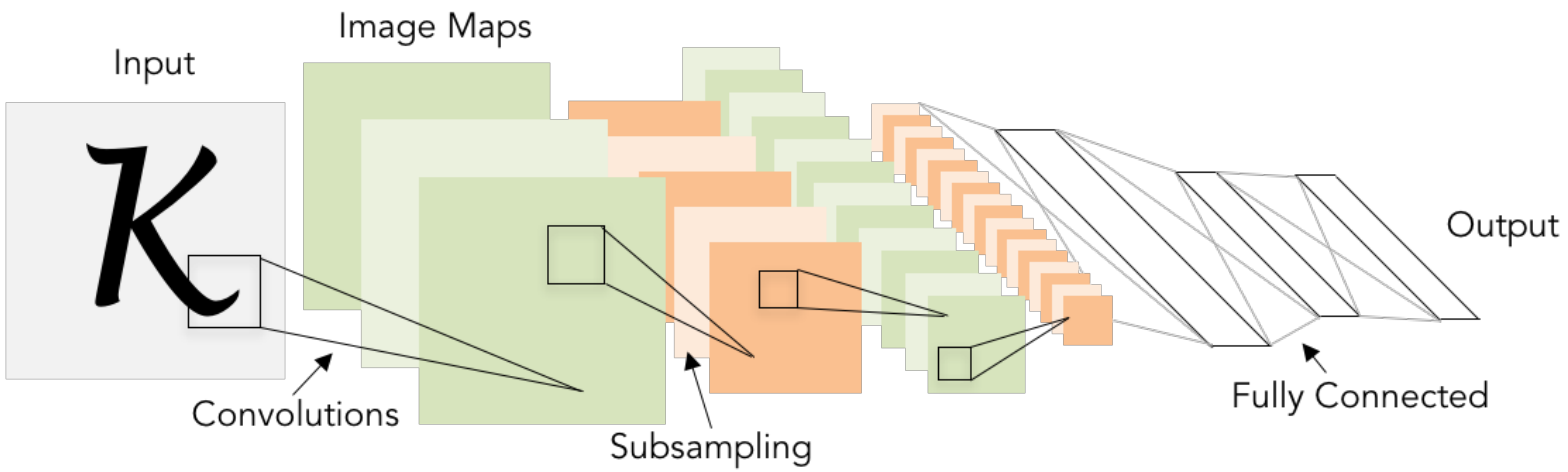


Normalization

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

Summary: Components of Convolutional Network

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



Next time: CNN Architectures

Final Project Overview

- Research-oriented final project
  - Instead of a final exam!

Can be completed in teams of 2-3 people

- Objectives
  - Gain experience reading literature
  - Reproduce published results
  - Propose a new idea and test the results!

Final Project Tasks

1. [Graded] Final Project Proposal document submission (2%)
2. [Graded] In-class topic-paper(s) presentation (4%)
3. In-class final project pitch
4. In-class final project checkpoint
5. [Graded] Reproduce published results (12%)
   - Algorithmic extension to obtain results with new idea, technique or dataset
6. [Graded] Video Presentation + Poster (4%)
7. [Graded] Final Report (2%)

Calendar updates

March

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Project pitch (14)

April

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Project checkpoint (11)

Final presentation
Posters & videos (27)

DeepRob

Lecture 7
Convolutional Neural Networks
University of Michigan and University of Minnesota