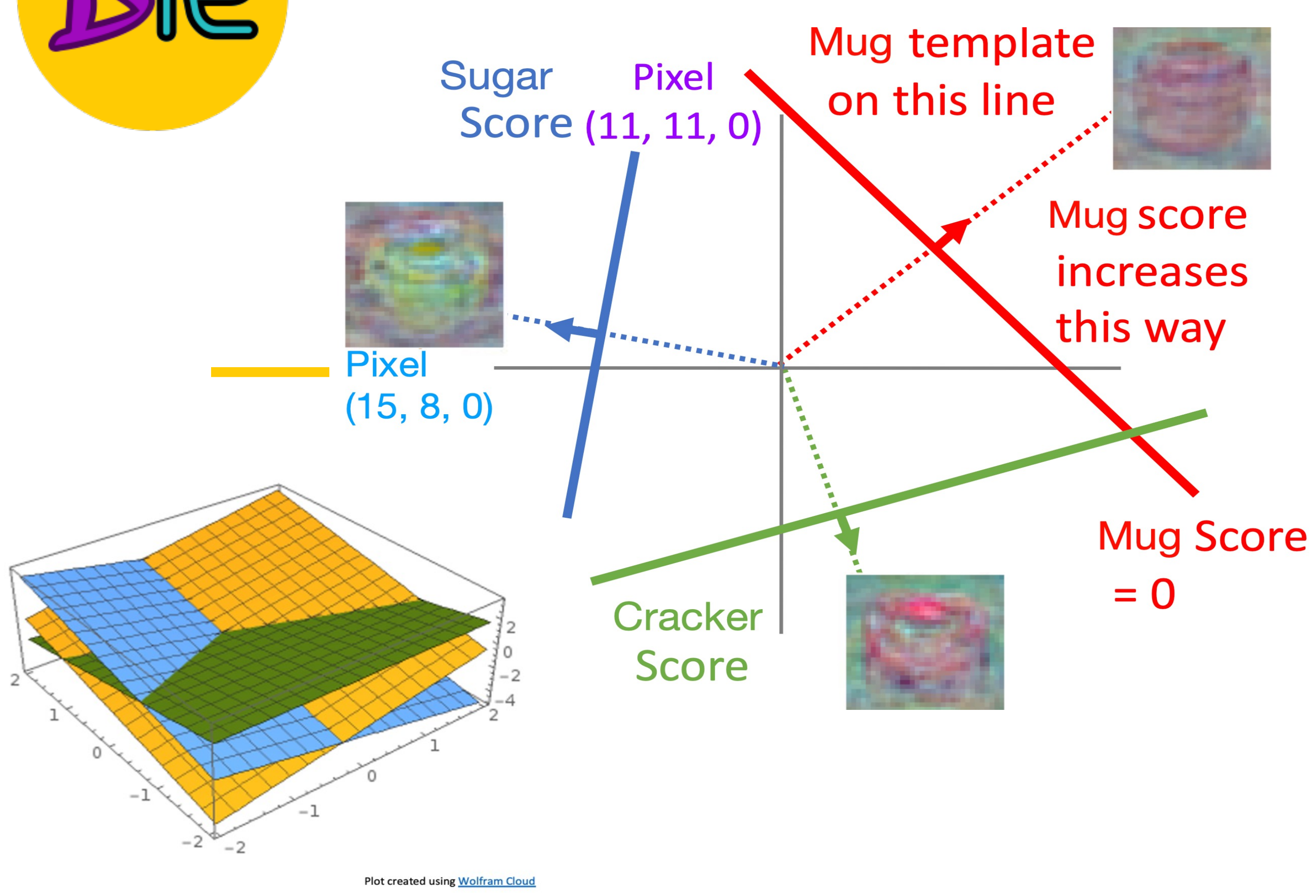
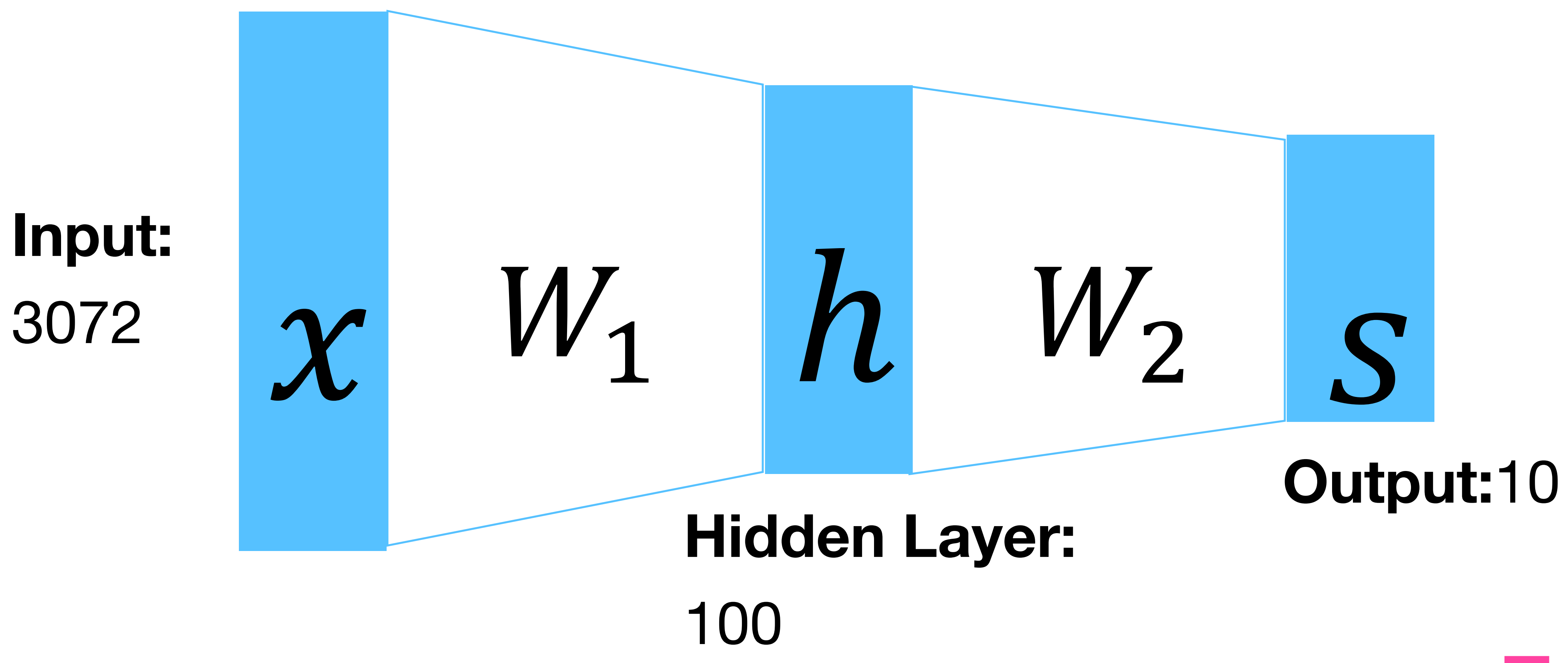
# DeepRob

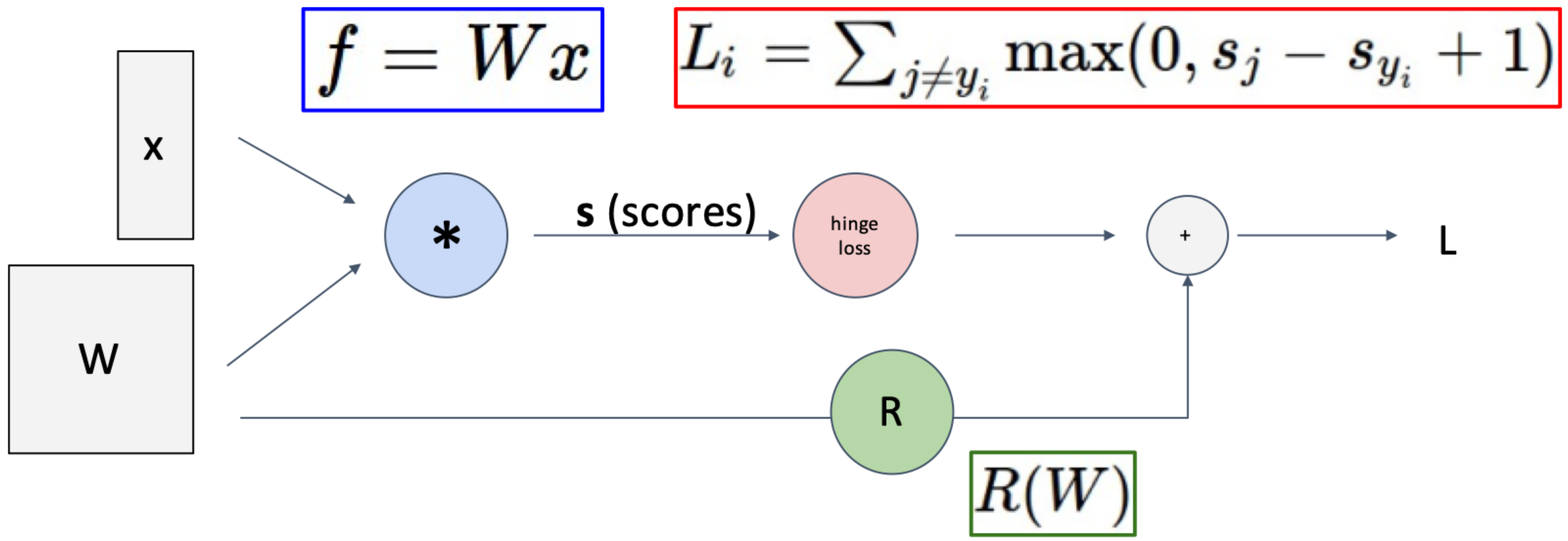
**Lecture 6**
**Convolutional Neural Networks**
**University of Michigan | Department of Robotics**

# Recap: Backpropagation




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



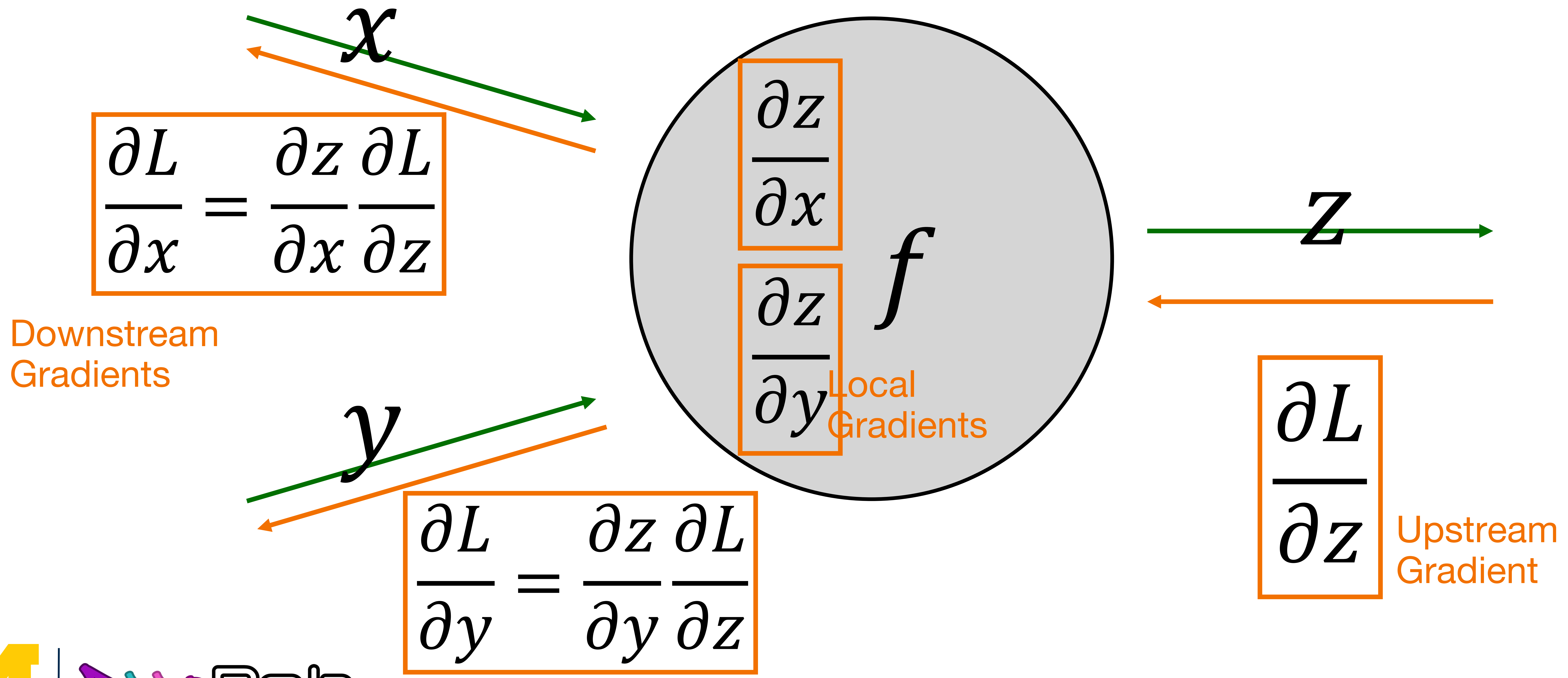




Forward pass

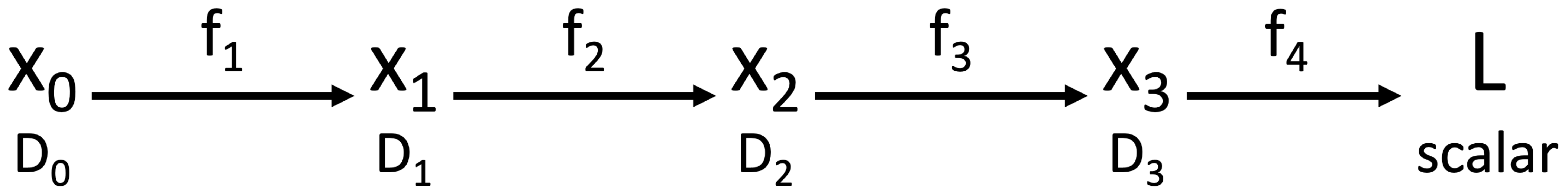
Backward pass (Backprop)

Computational Graph

# Recap: Backpropagation



# Recap: “The Chain Rule”

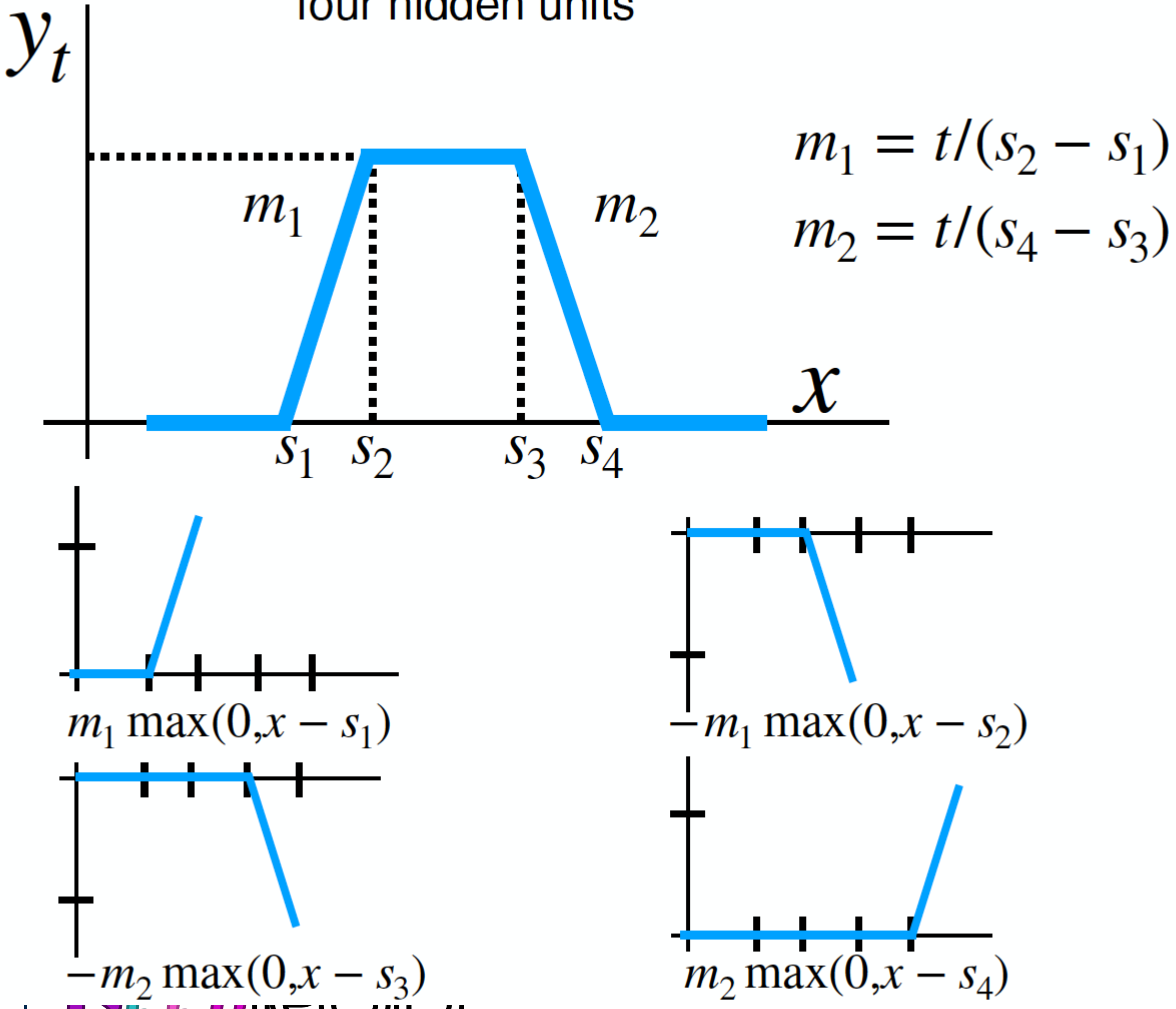



Chain rule

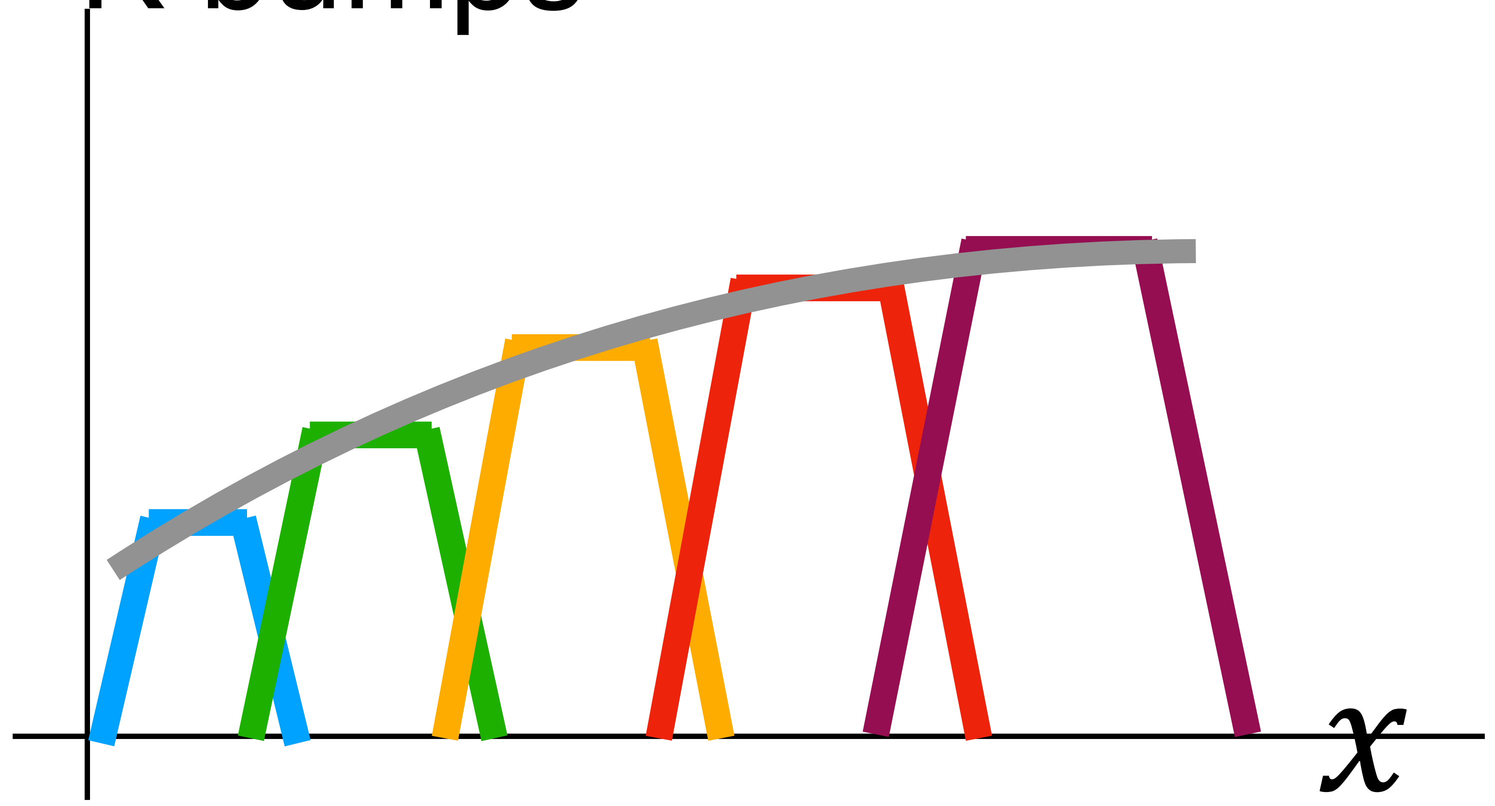
$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0}\right)\left(\frac{\partial x_2}{\partial x_1}\right)\left(\frac{\partial x_3}{\partial x_2}\right)\left(\frac{\partial L}{\partial x_3}\right)$$

# Recap: Universal Approximation

We can build a “bump function” using four hidden units

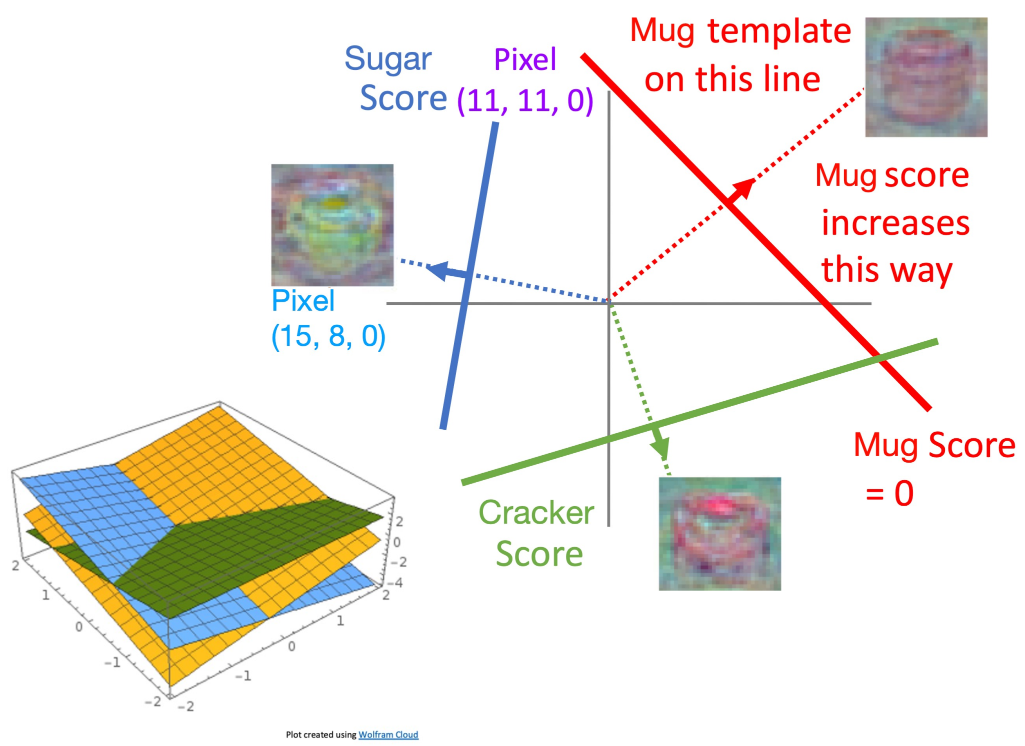



With 4K hidden units we can build a sum of K bumps

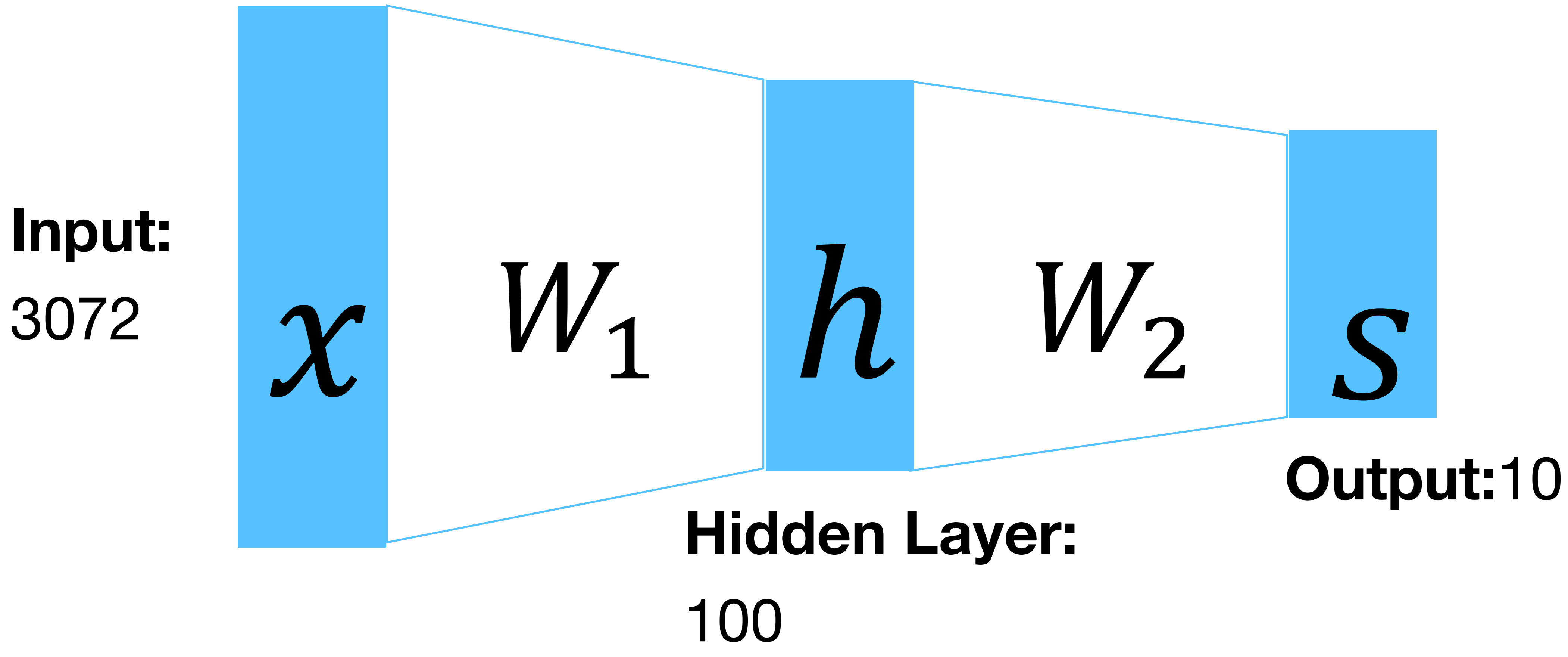



Approximate functions with bumps!

# Spatial Structure?

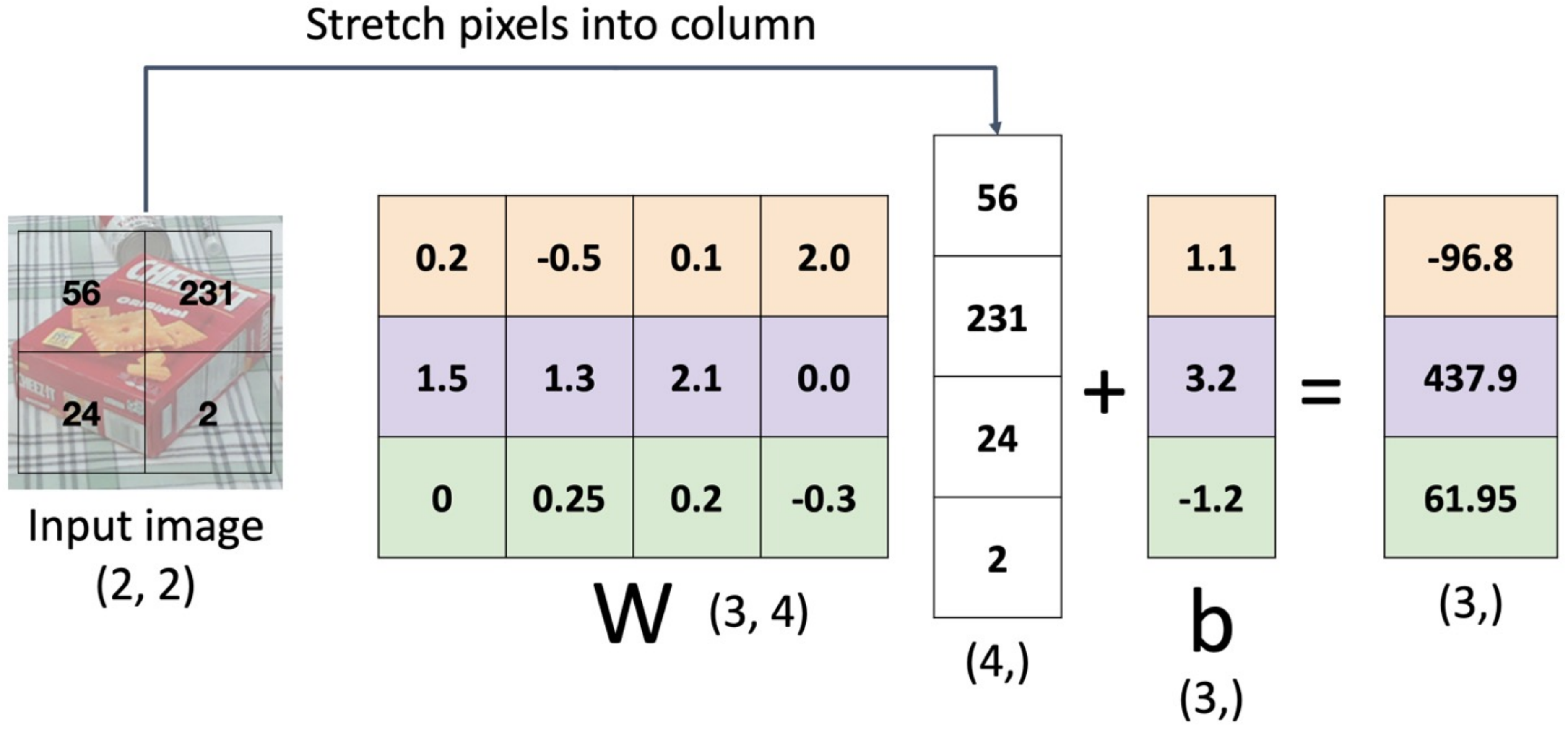



$$f(x) = W_2 max(0, W_1 x + b_1) + b_2$$



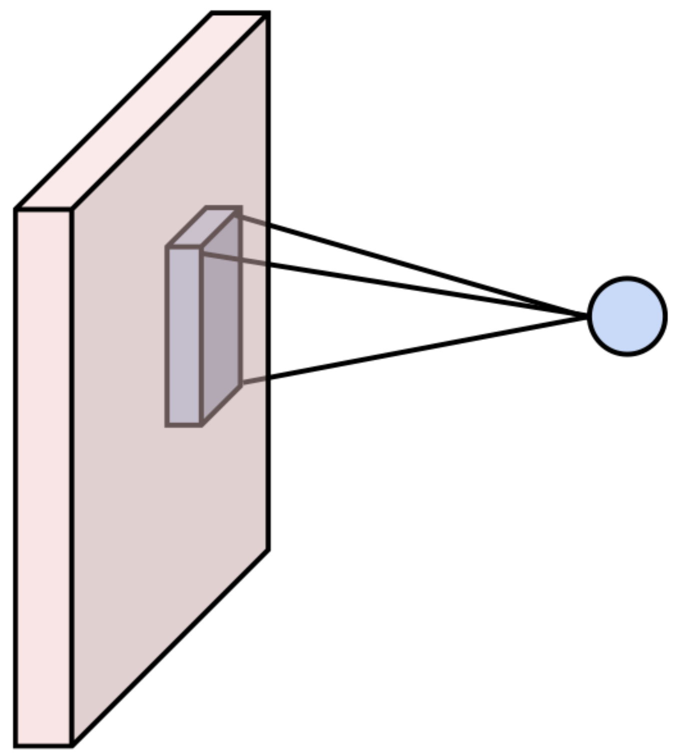

**Problem**: So far our classifiers don't respect the spatial structure of images!

**Solution**: Define new computational nodes that operate on images!

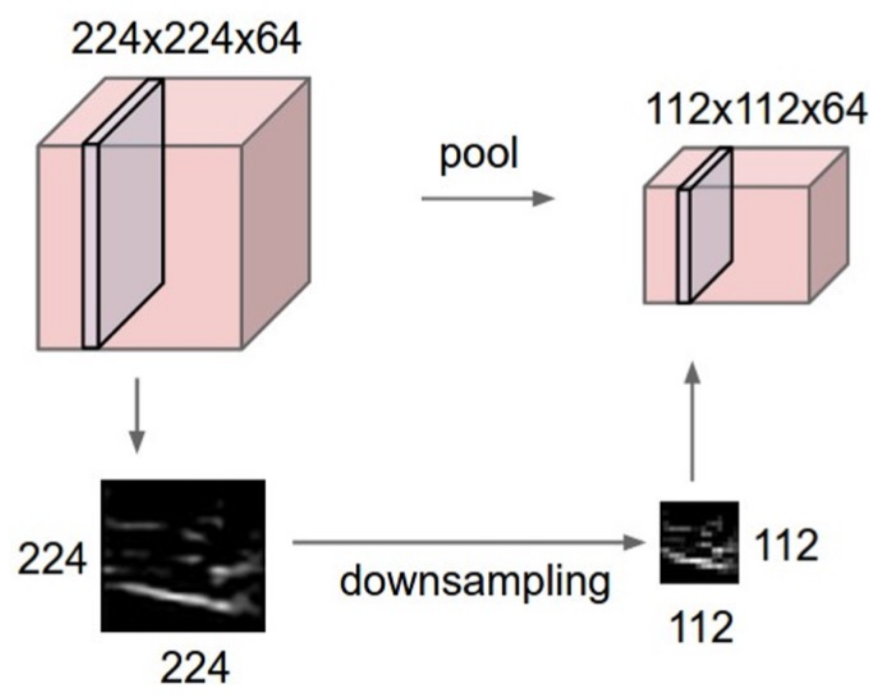


# Components of Convolutional Networks

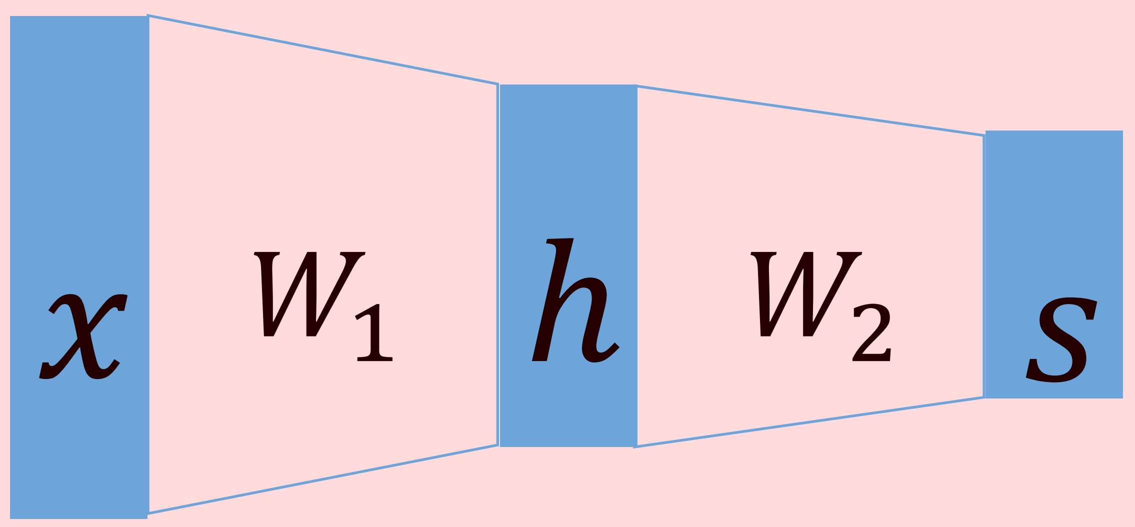
## Convolution Layers



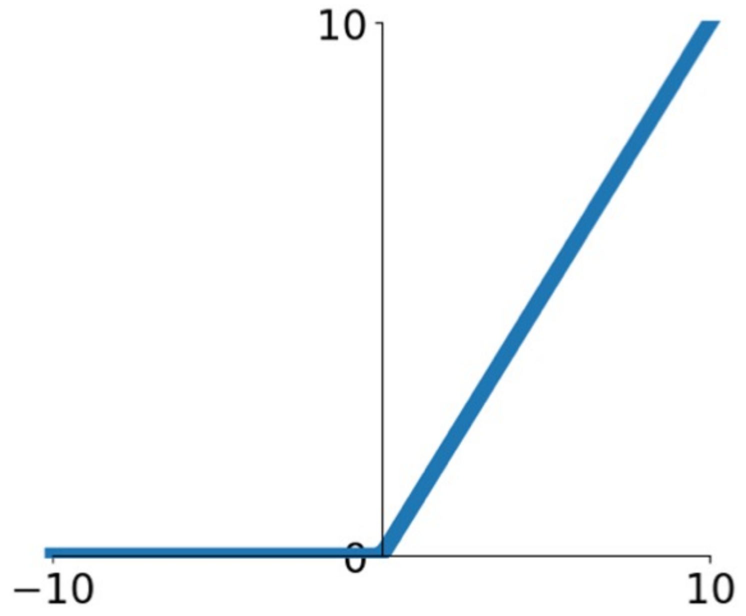
## Pooling Layers




## Fully-Connected Layers



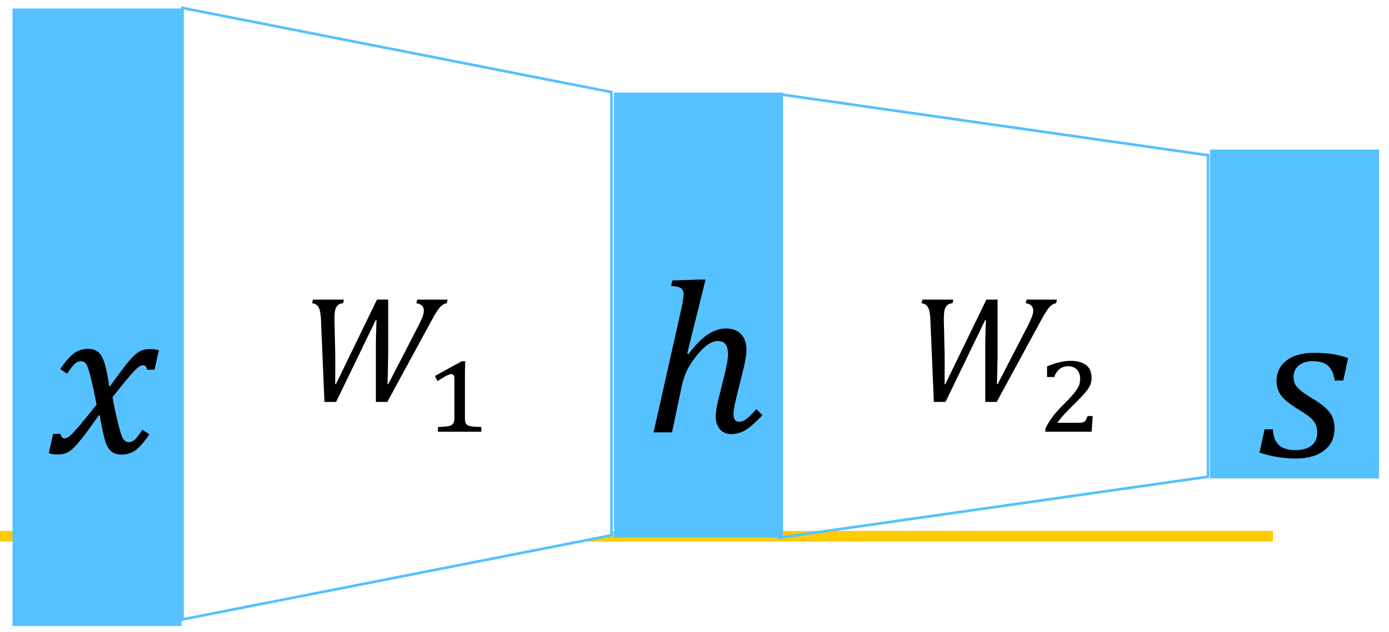

## Activation Function

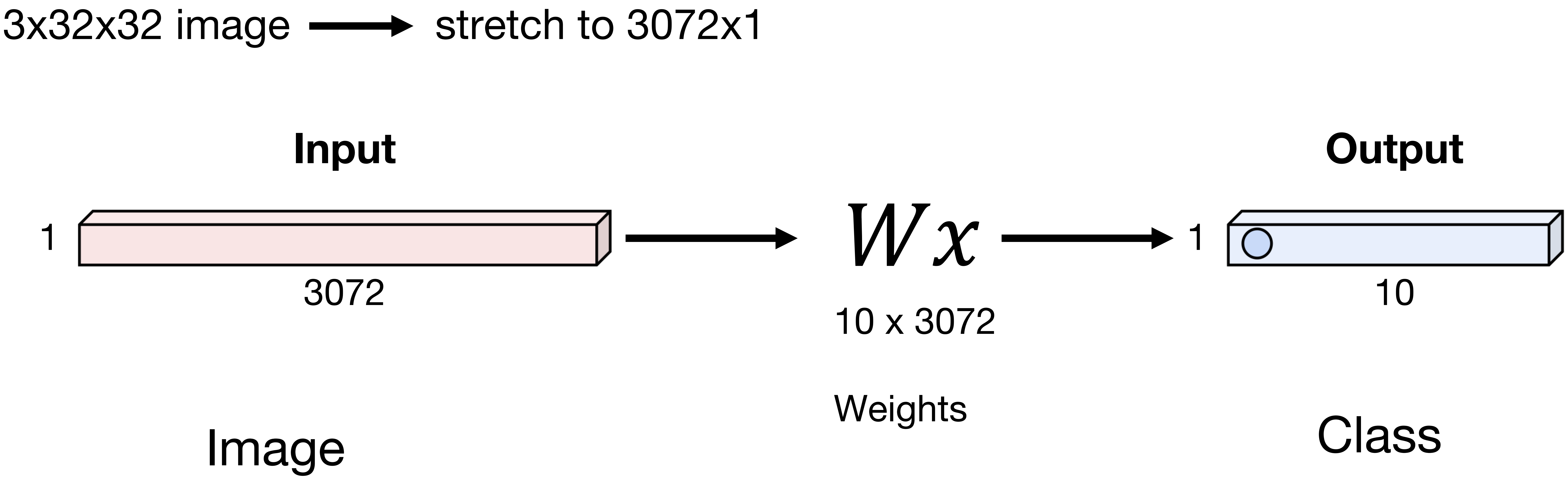



## Normalization

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

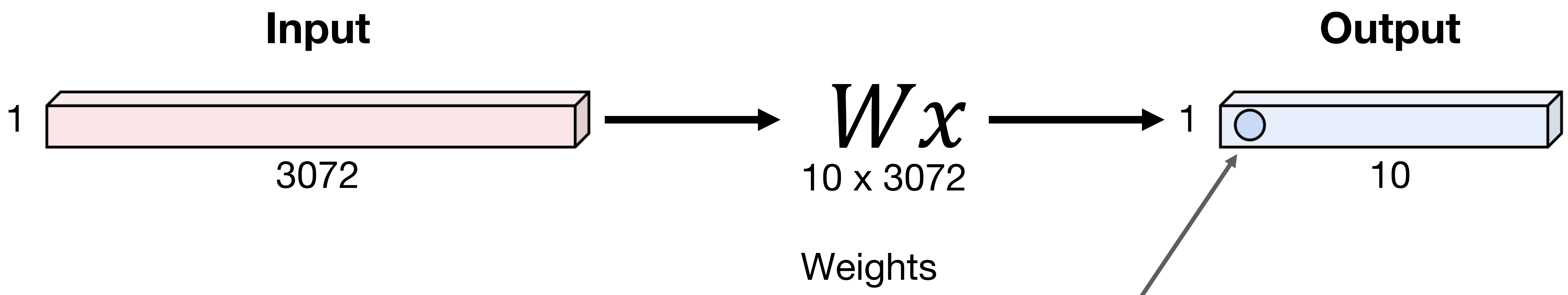
# Fully-Connected Layer



# 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

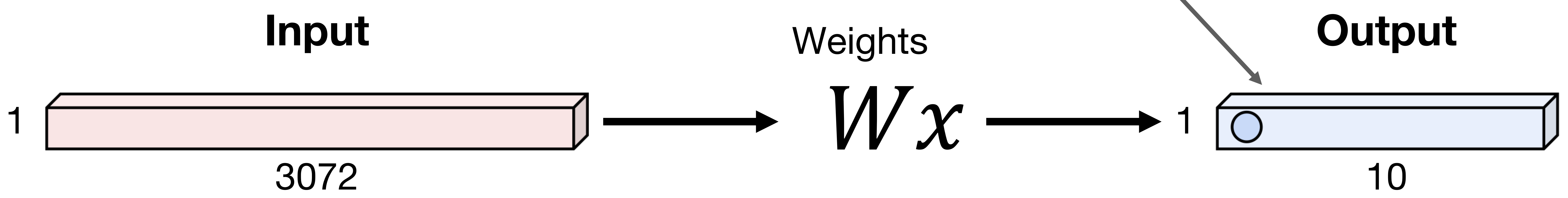
# Fully-Connected Layer

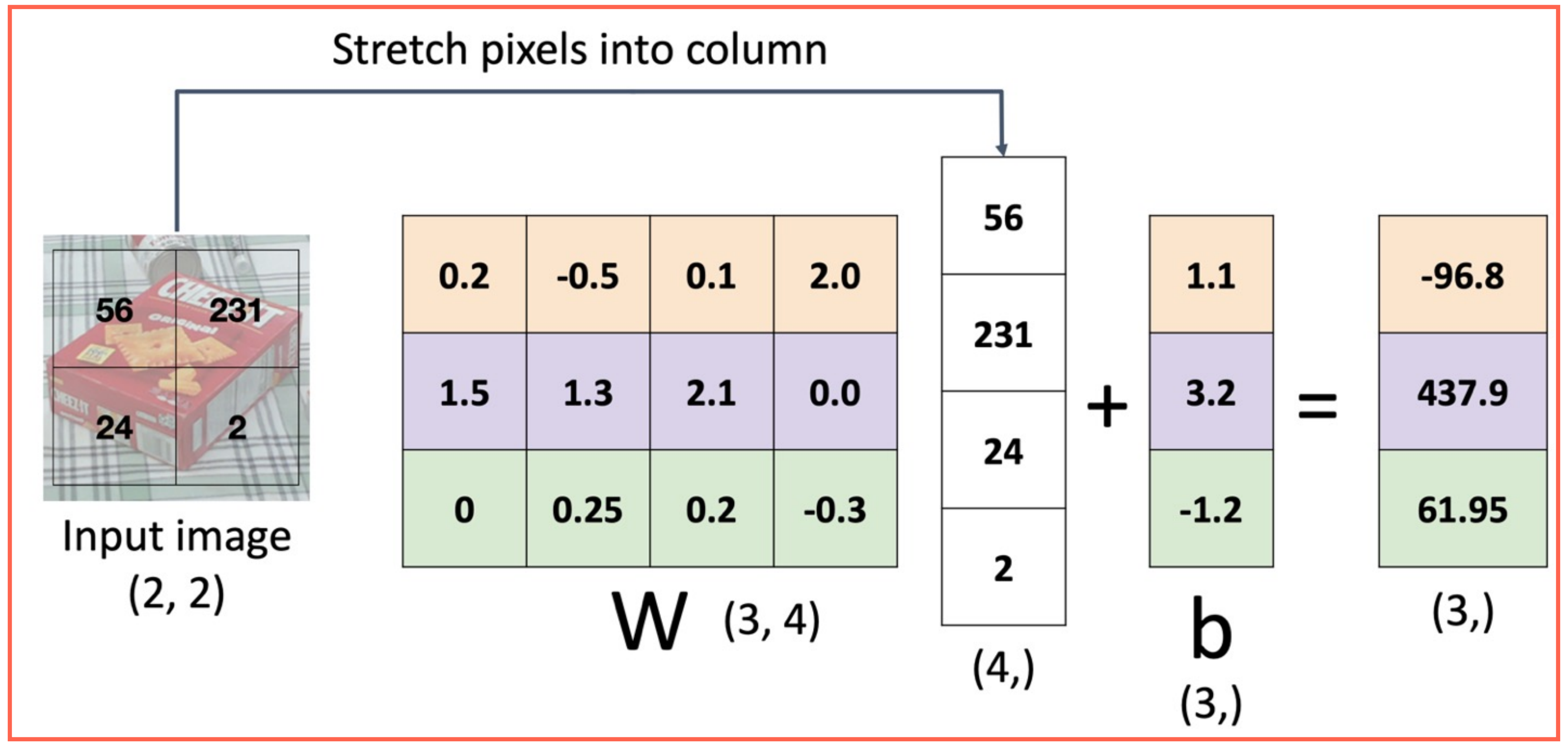
3x32x32 image → stretch to 3072x1

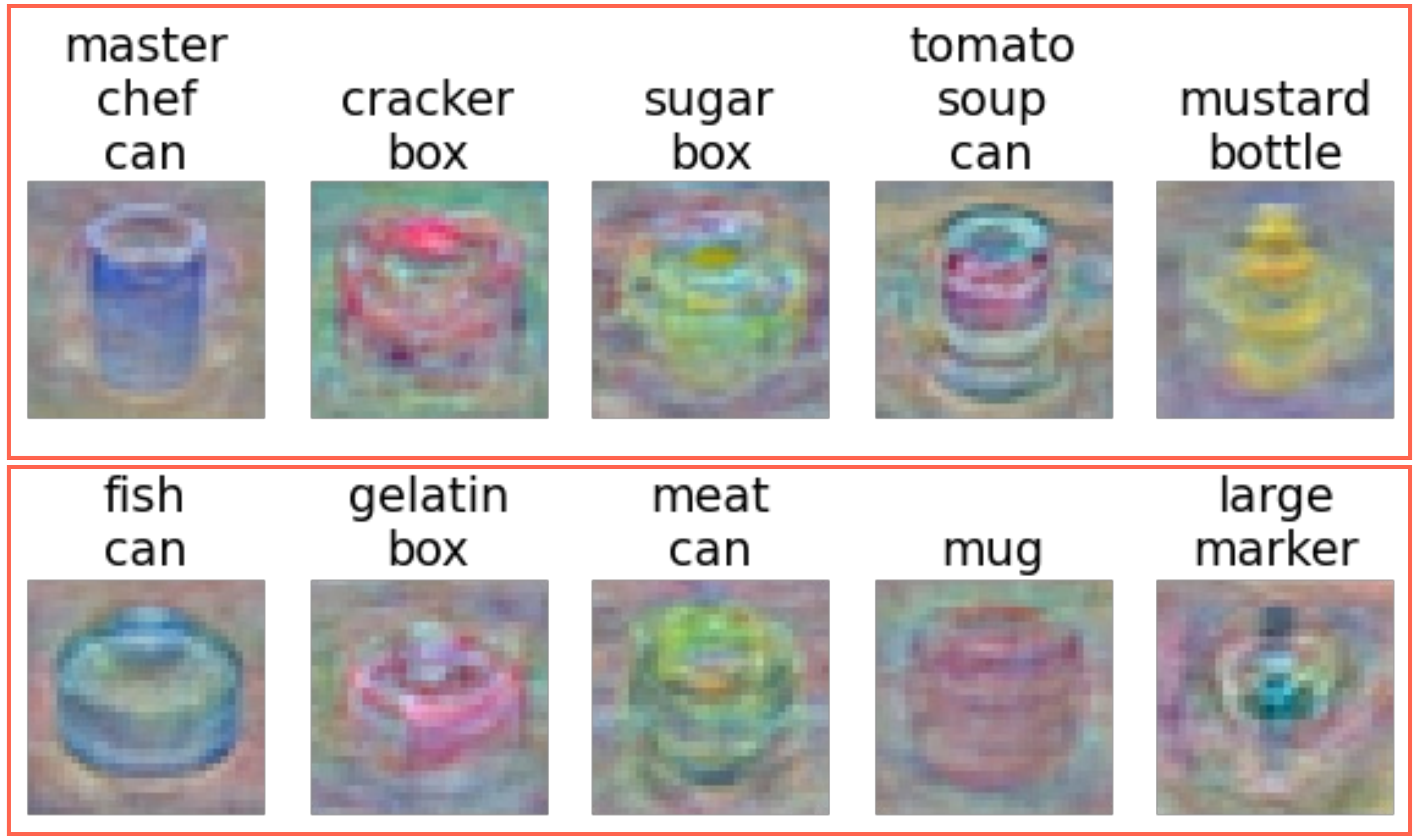
10 x 3072

**1 number:**

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

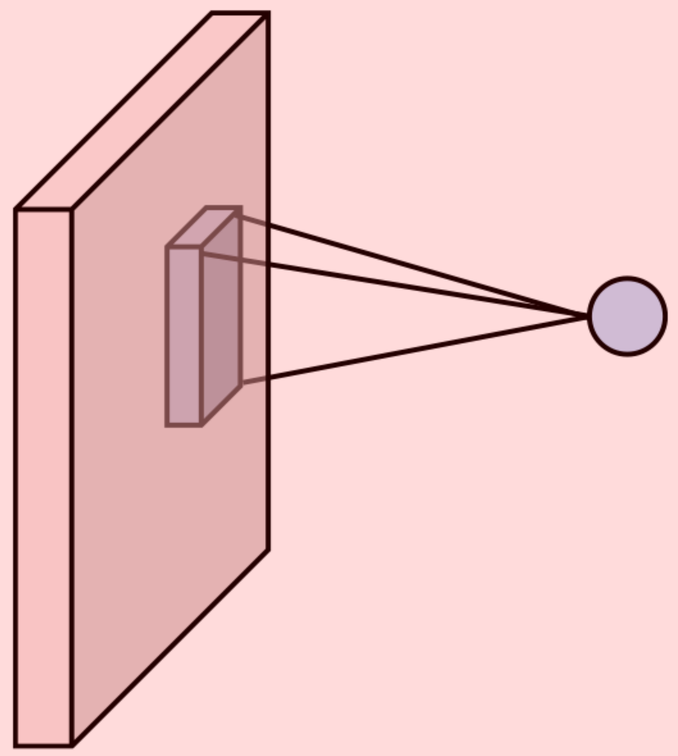


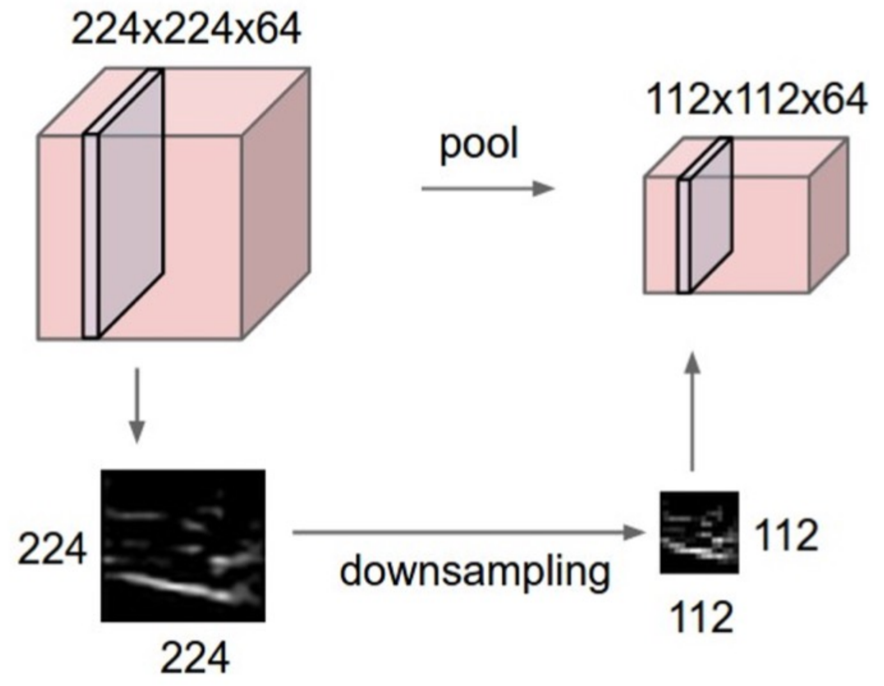



# Components of Convolutional Networks

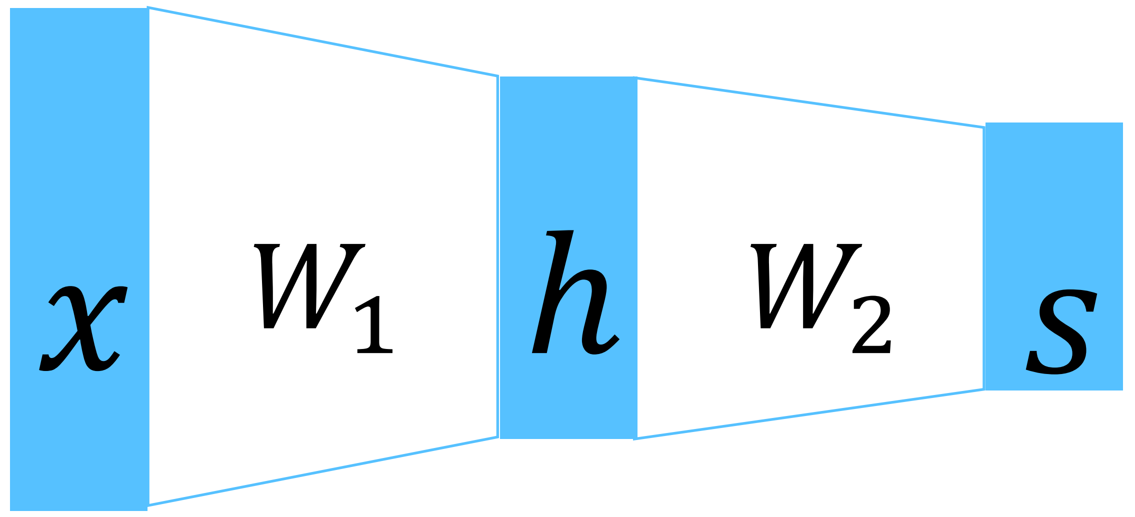
## Convolution Layers



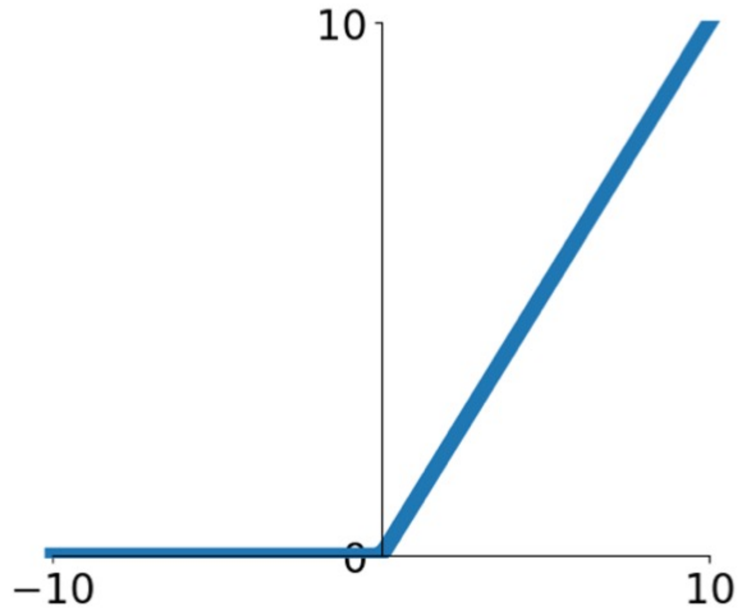
## Pooling Layers




## Fully-Connected Layers



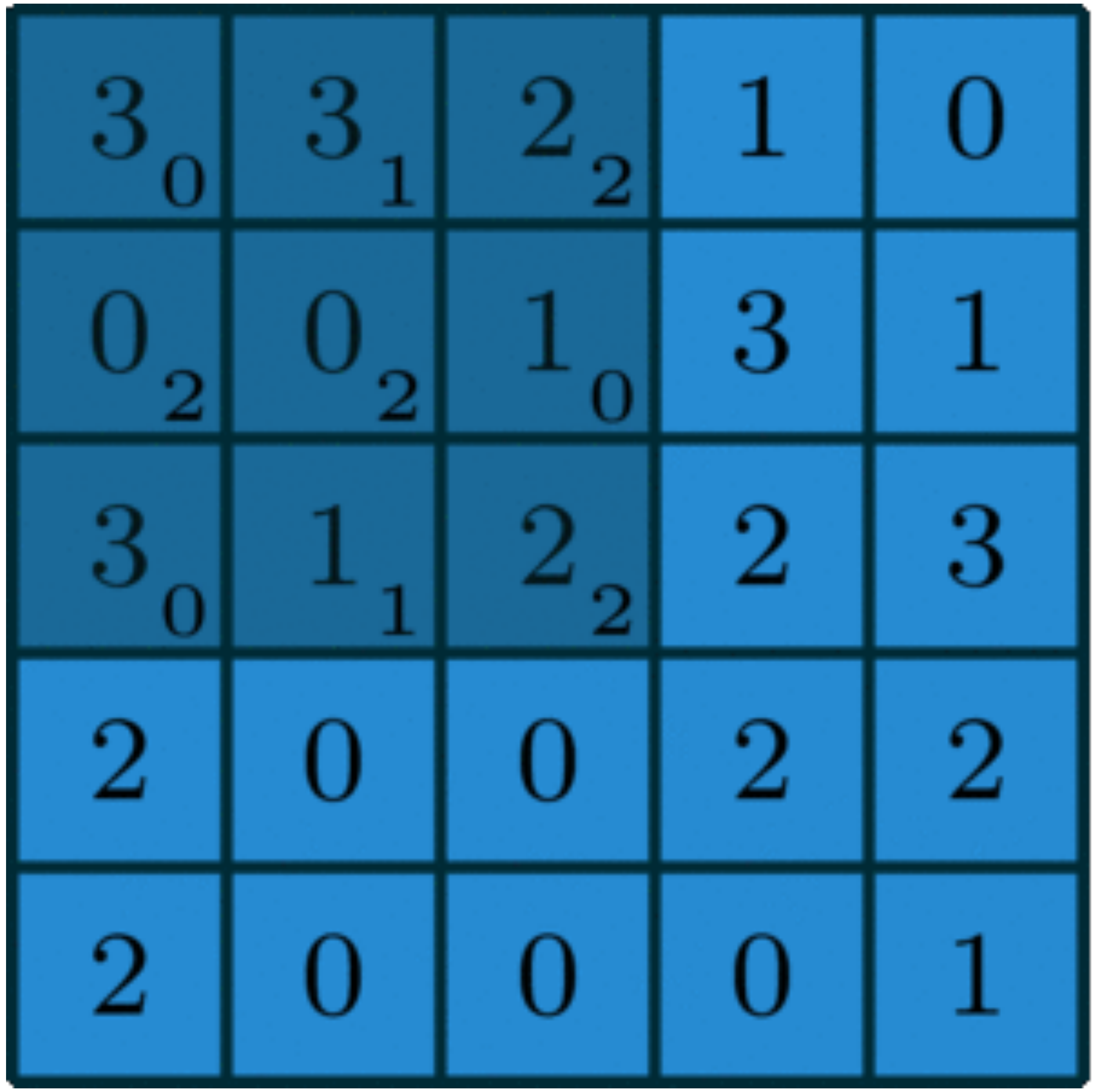

## Activation Function




## Normalization

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

# Convolution Operation





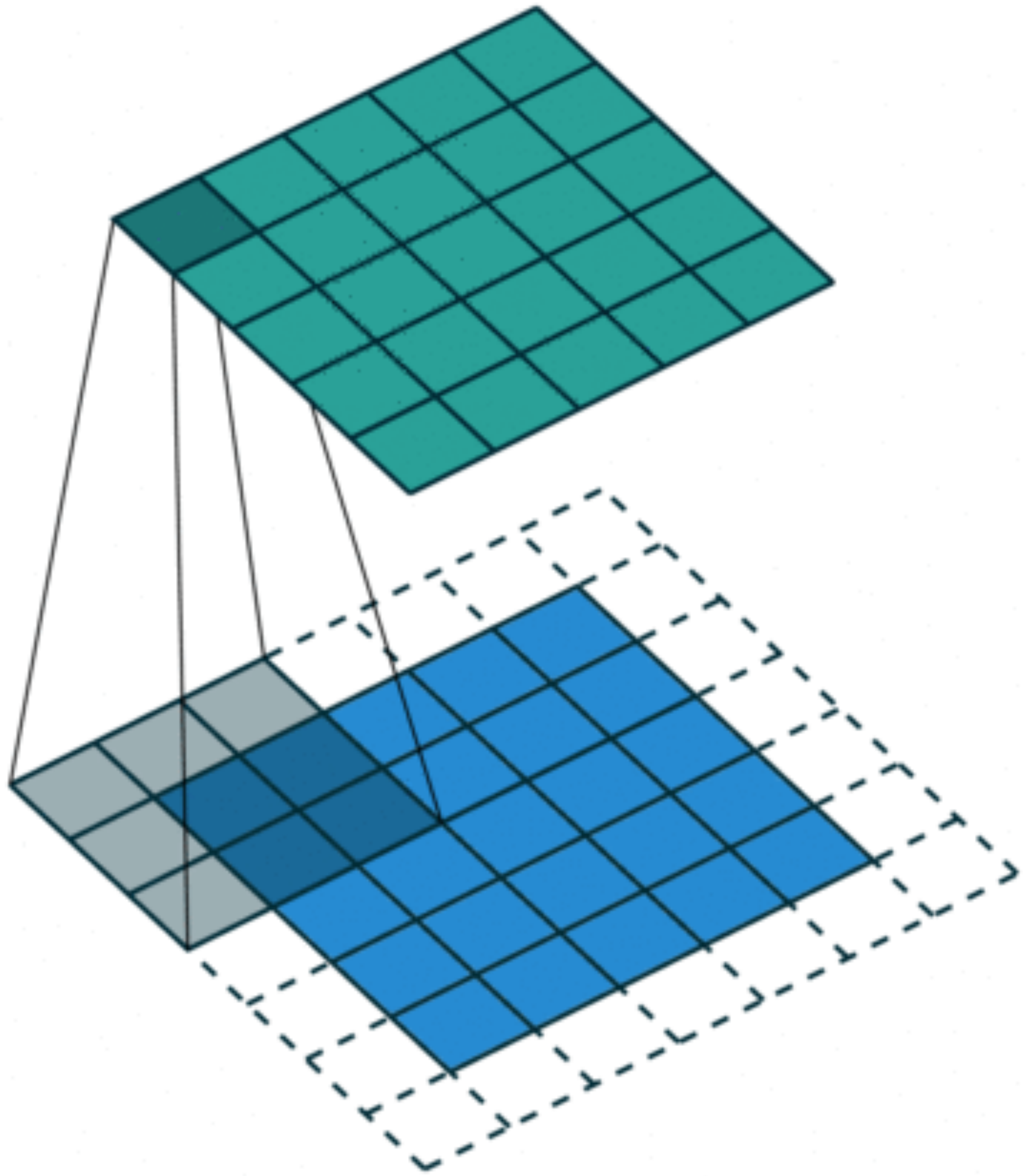
Kernel:

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 2 | 2 | 0 |
| 0 | 1 | 2 |

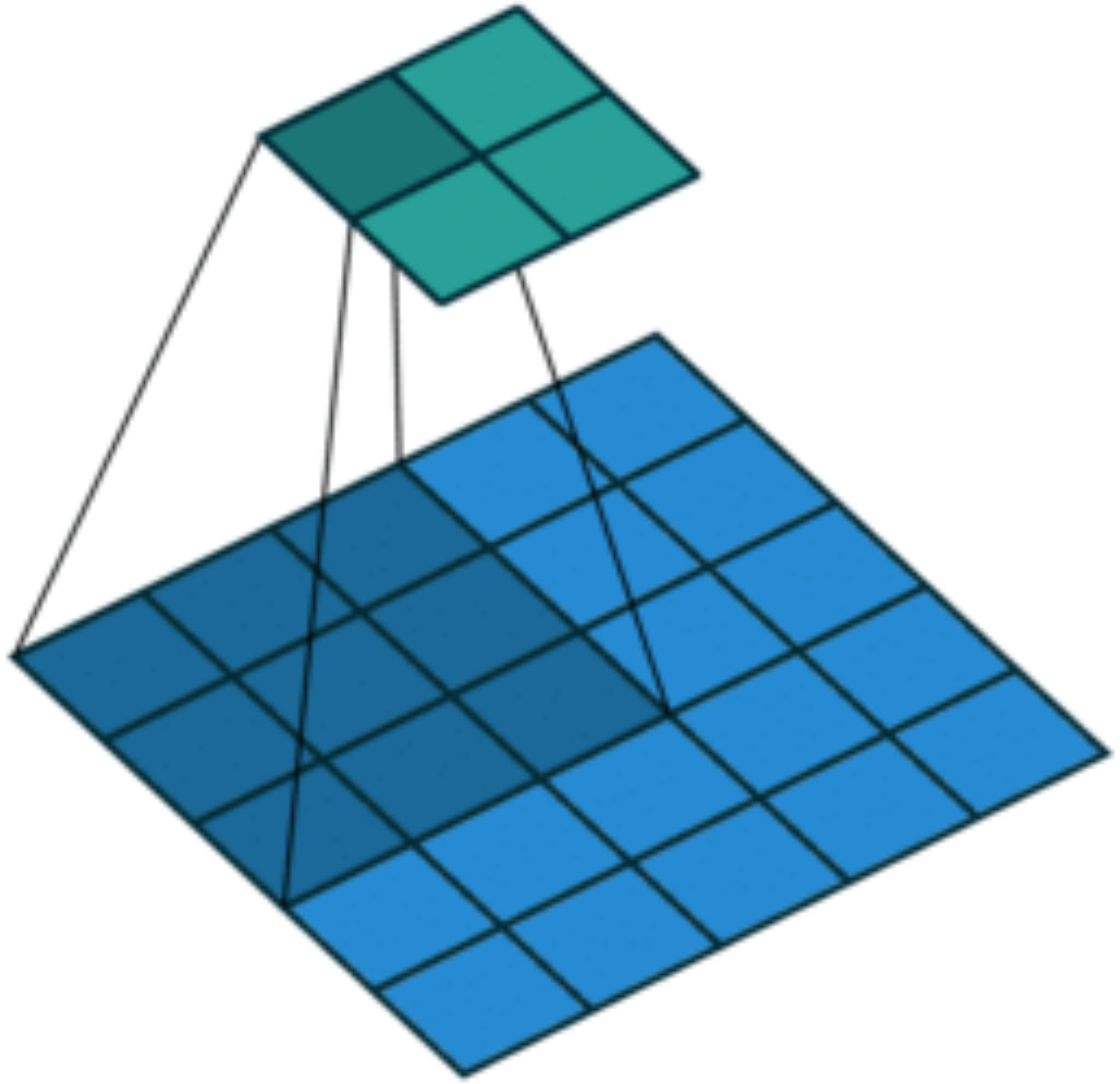
(3x3)

https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

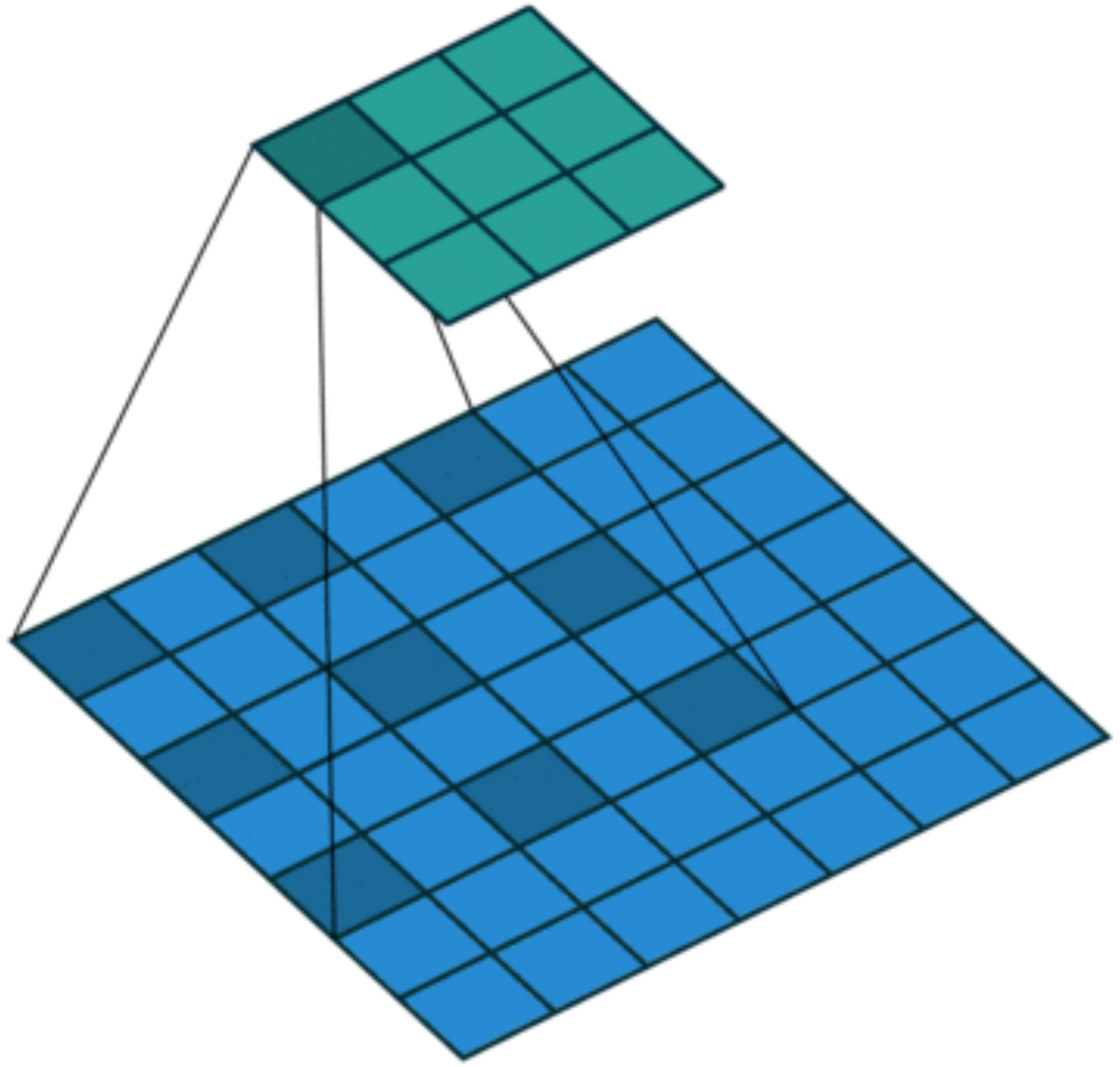
# Convolution Operation



Padding

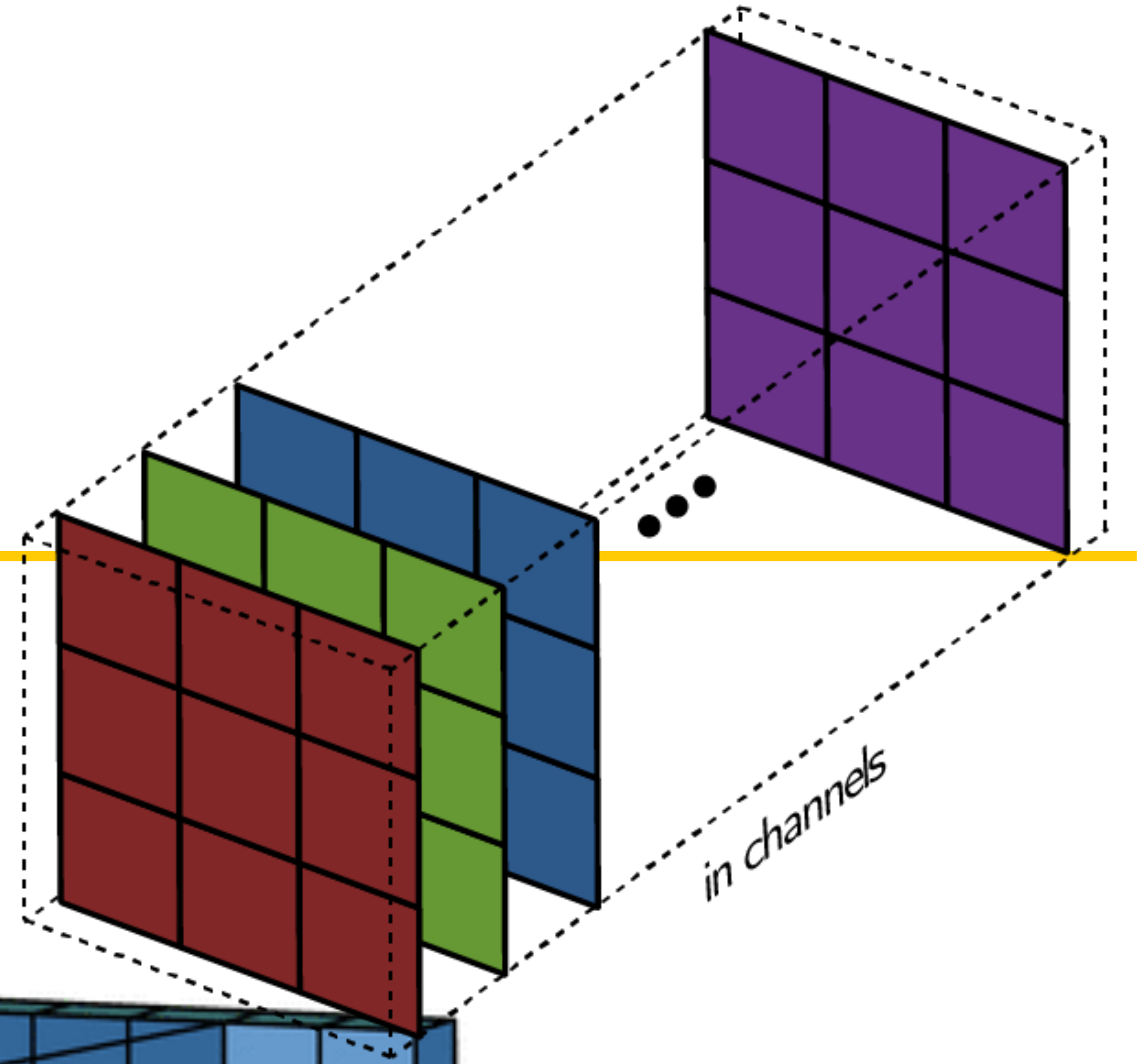


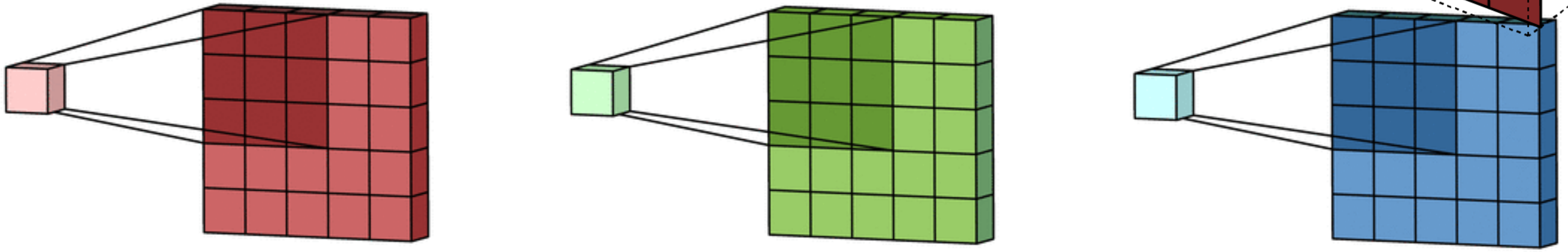
Stride = 2



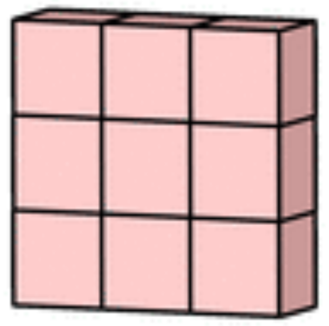
dilation = 2

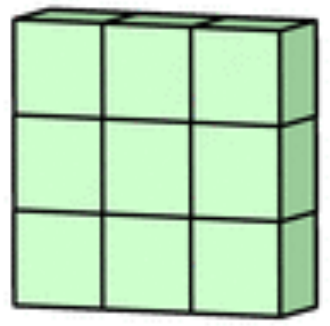
# Convolution Filters

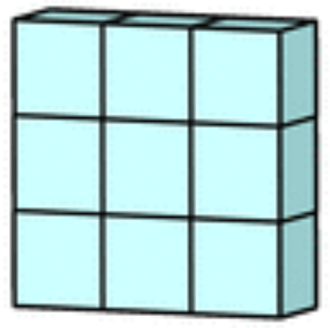


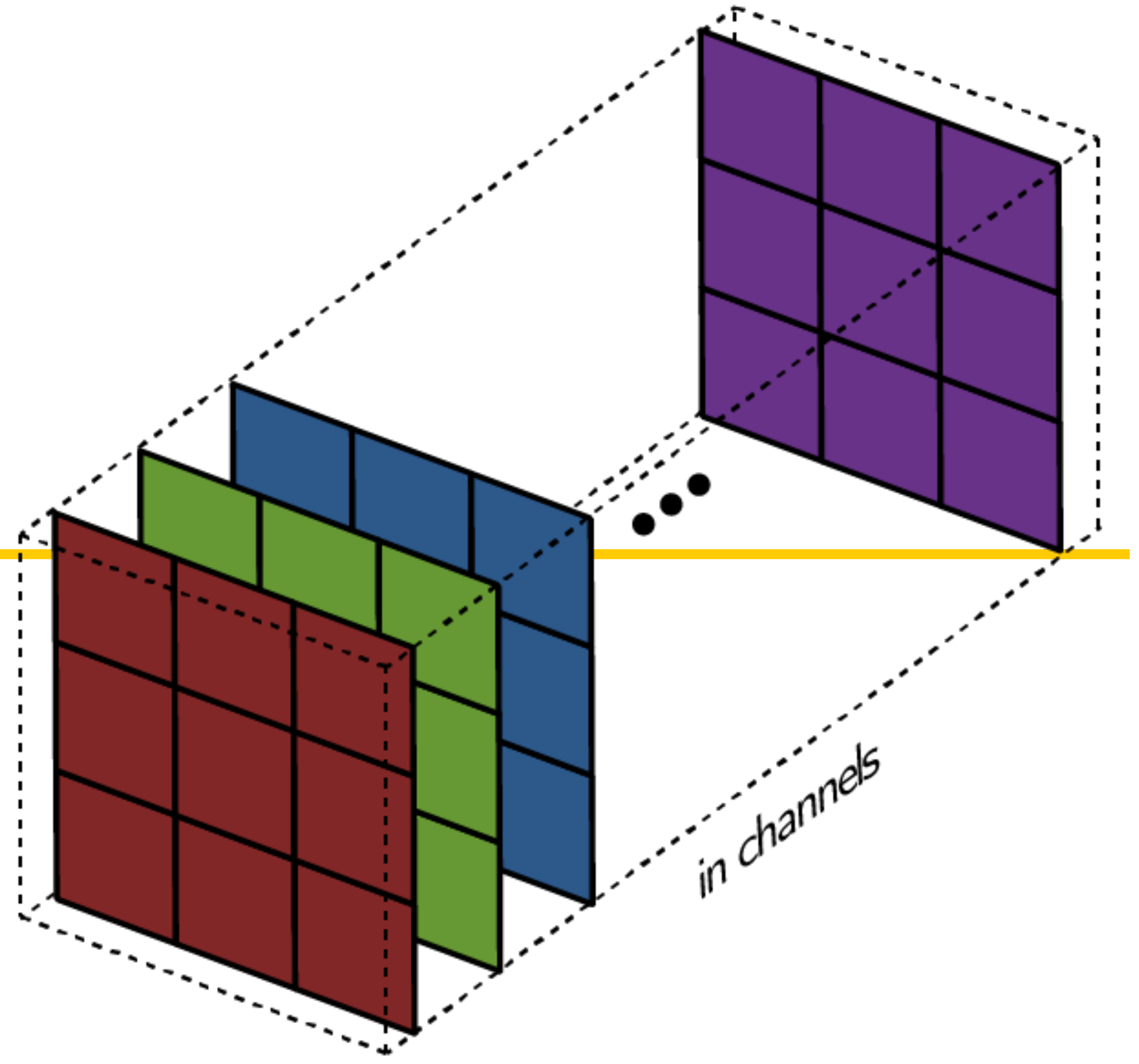



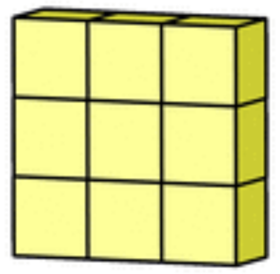
# Convolution Filters









# Convolution Layer

3x32x32 image:  preserve spatial structure

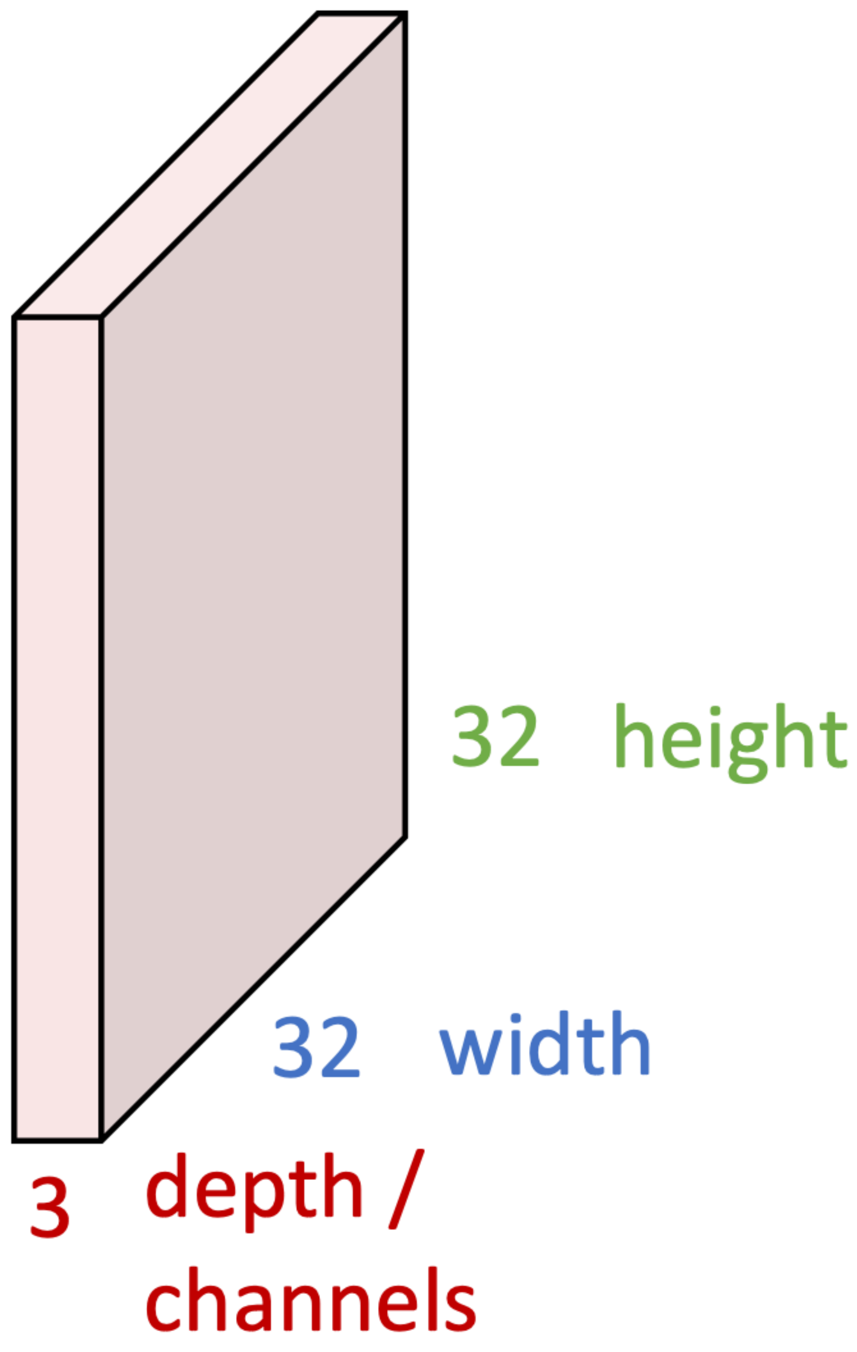

**Convolve** the filter with the image i.e., "slide over the image spatially, computing dot products"

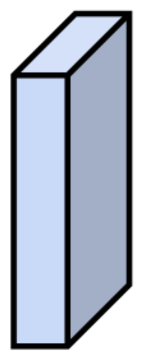
# Convolution Layer

3x32x32 image

Filters always extend the full depth of the input volume

3x5x5 filter



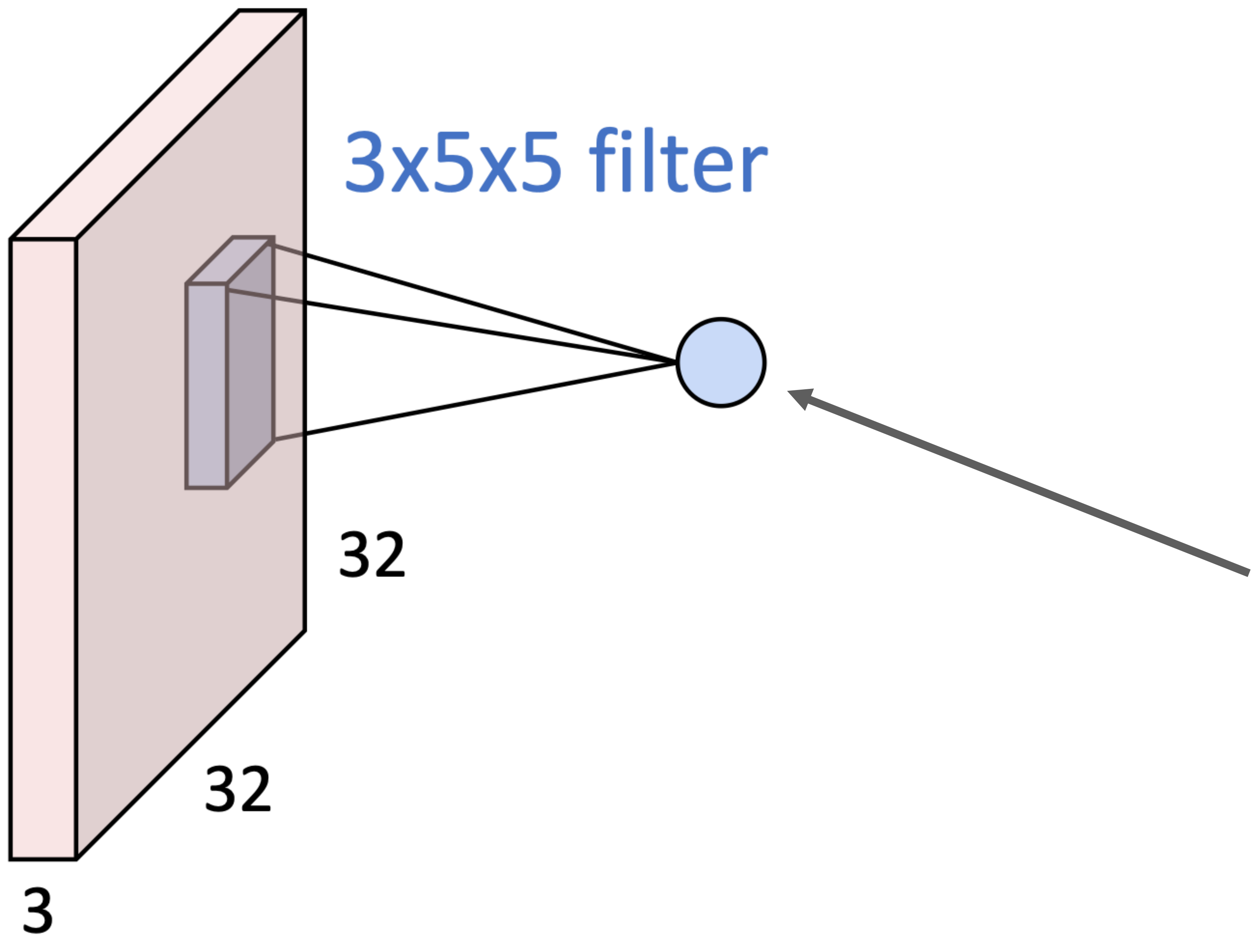



**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”

# Convolution Layer

3x32x32 image

L_out = (L_in + 2 * padding - dilation * (kernel - 1) - 1) / stride + 1



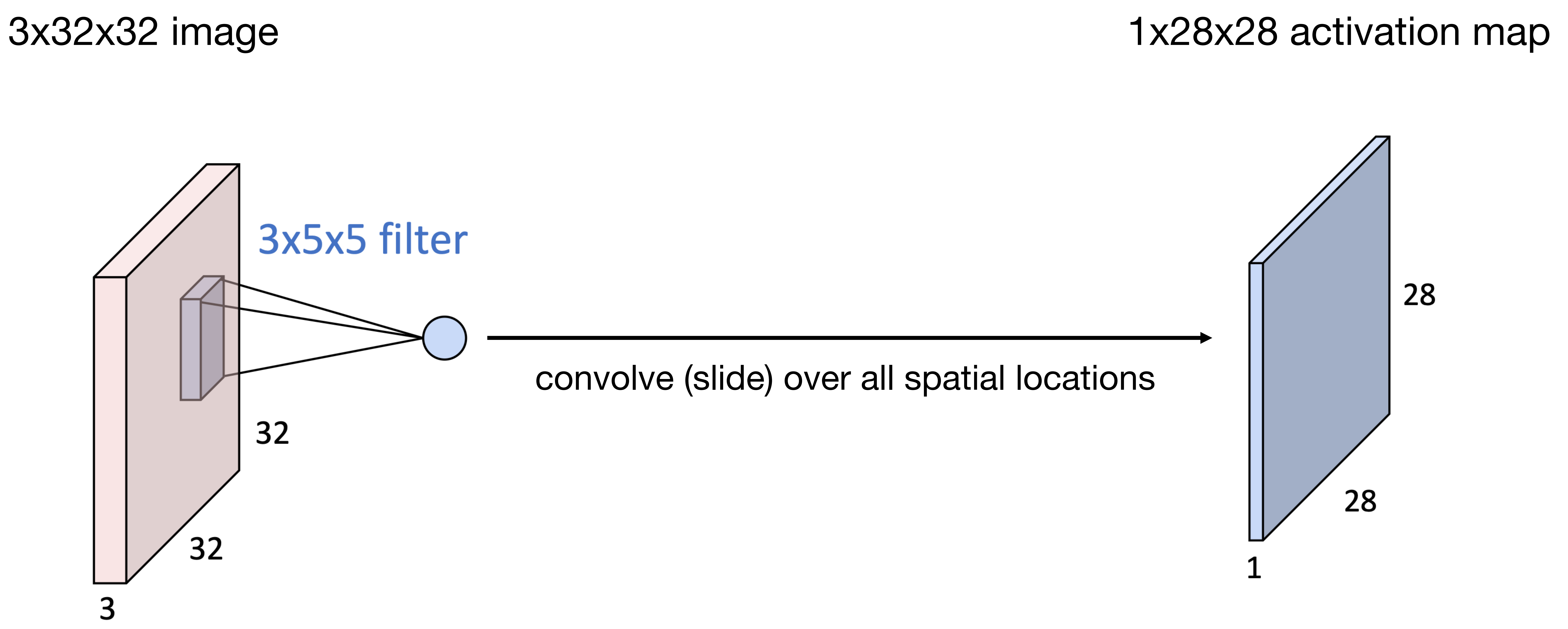

**1 number:**

The result of taking a dot product between the filter and a small 3x5x5 portion of the image

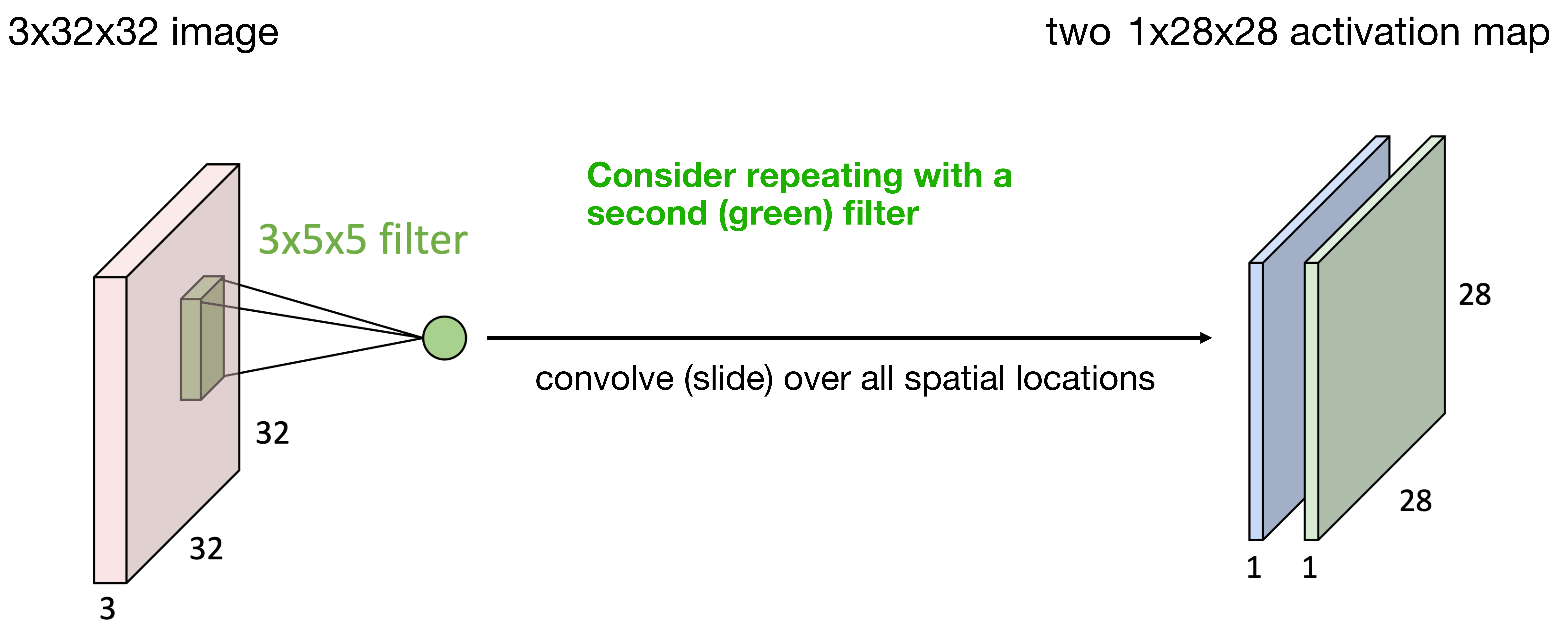
(i.e. 3*5*5=75-dimensional dot product + bias)

$$w^T x + b$$

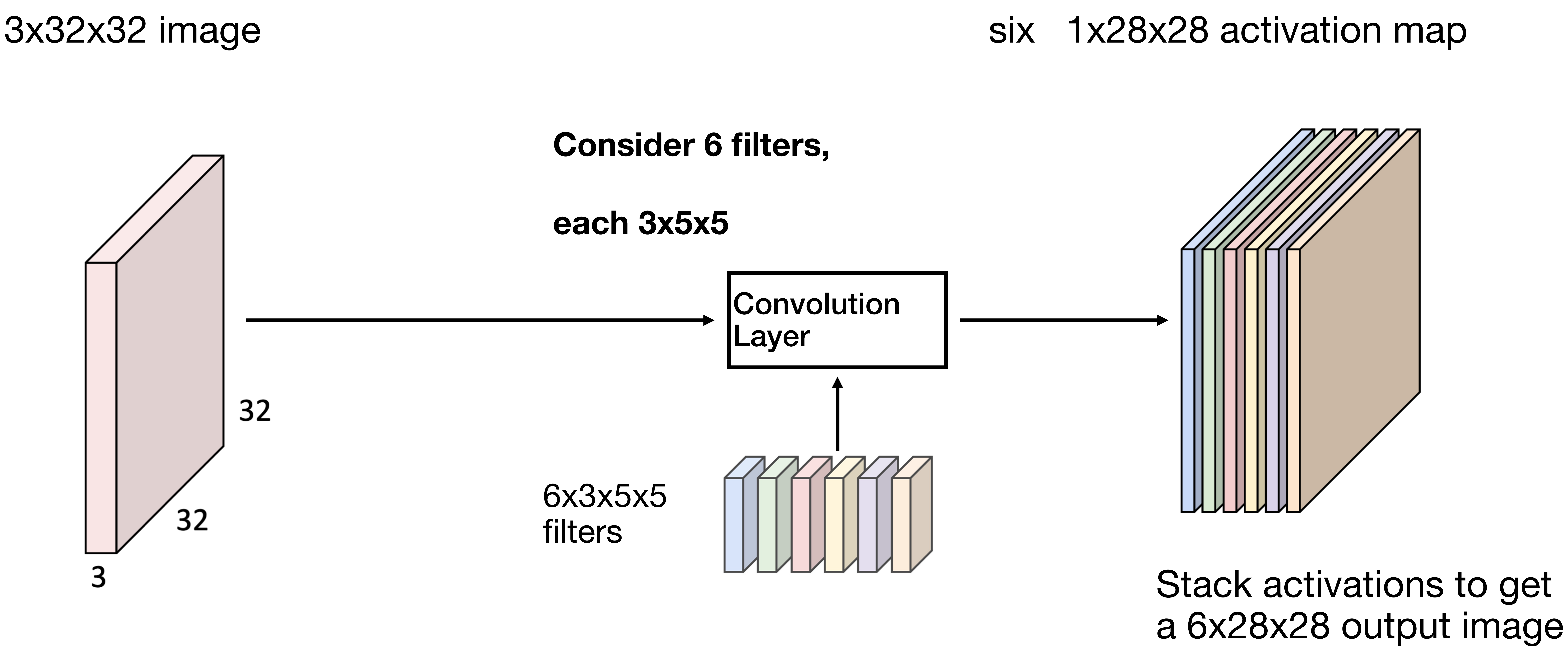
# Convolution Layer



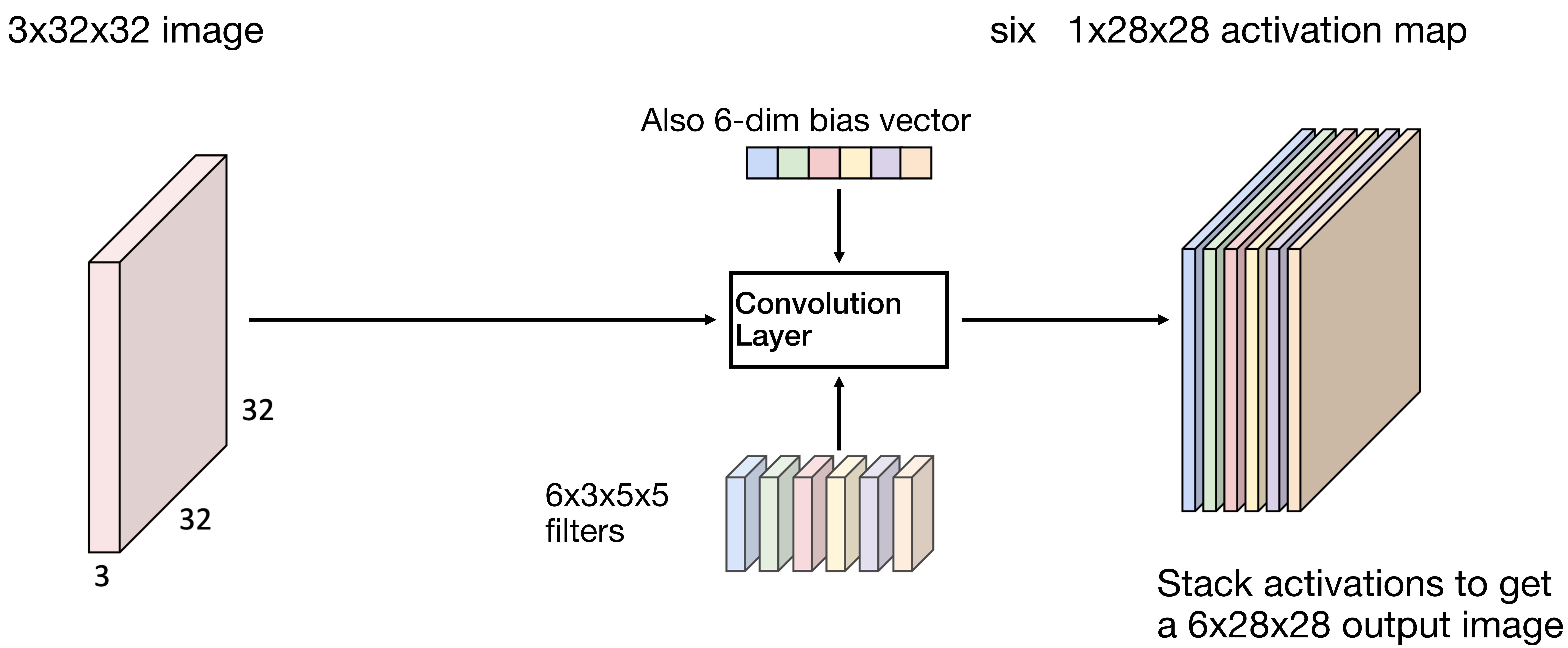
# Convolution Layer



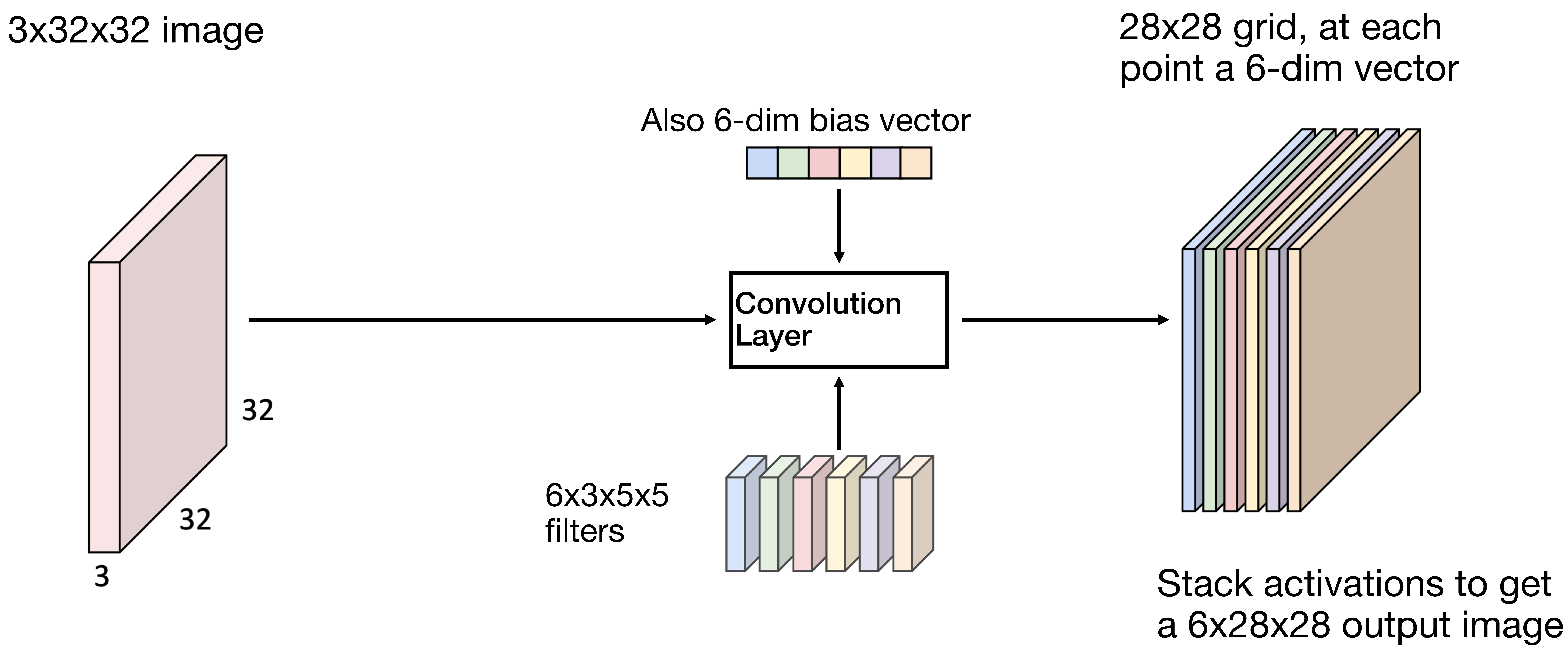
# Convolution Layer



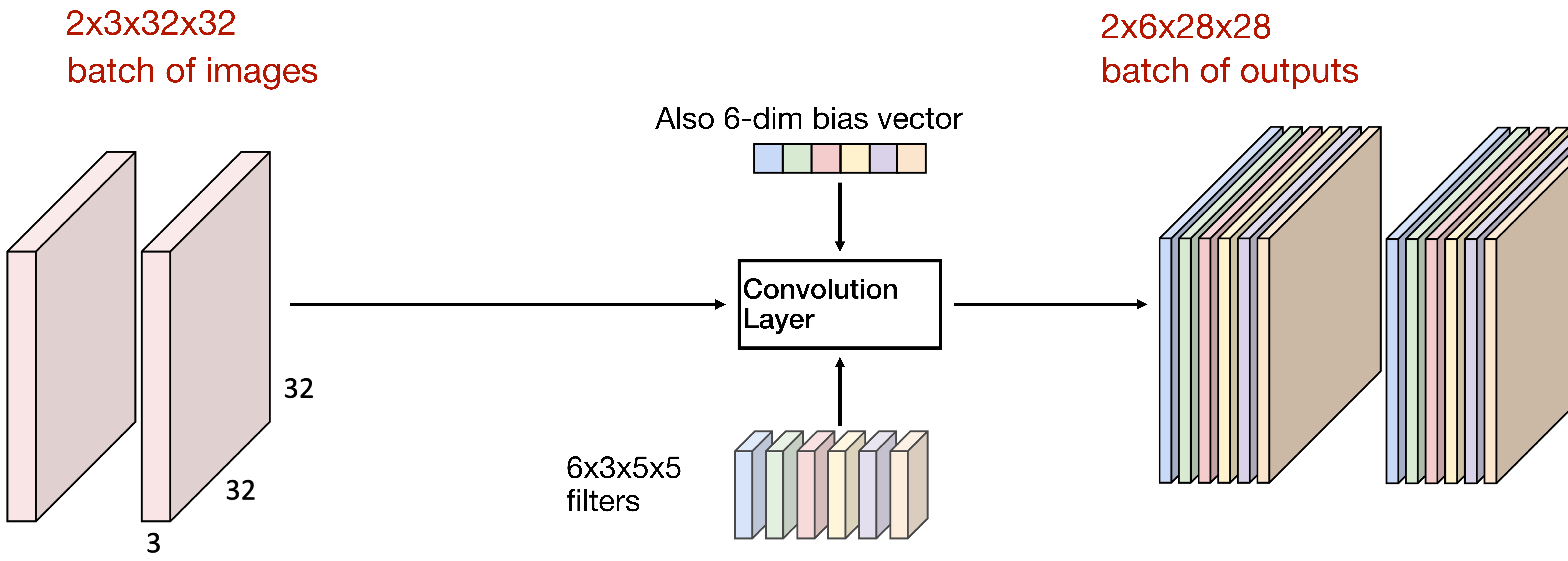
# Convolution Layer



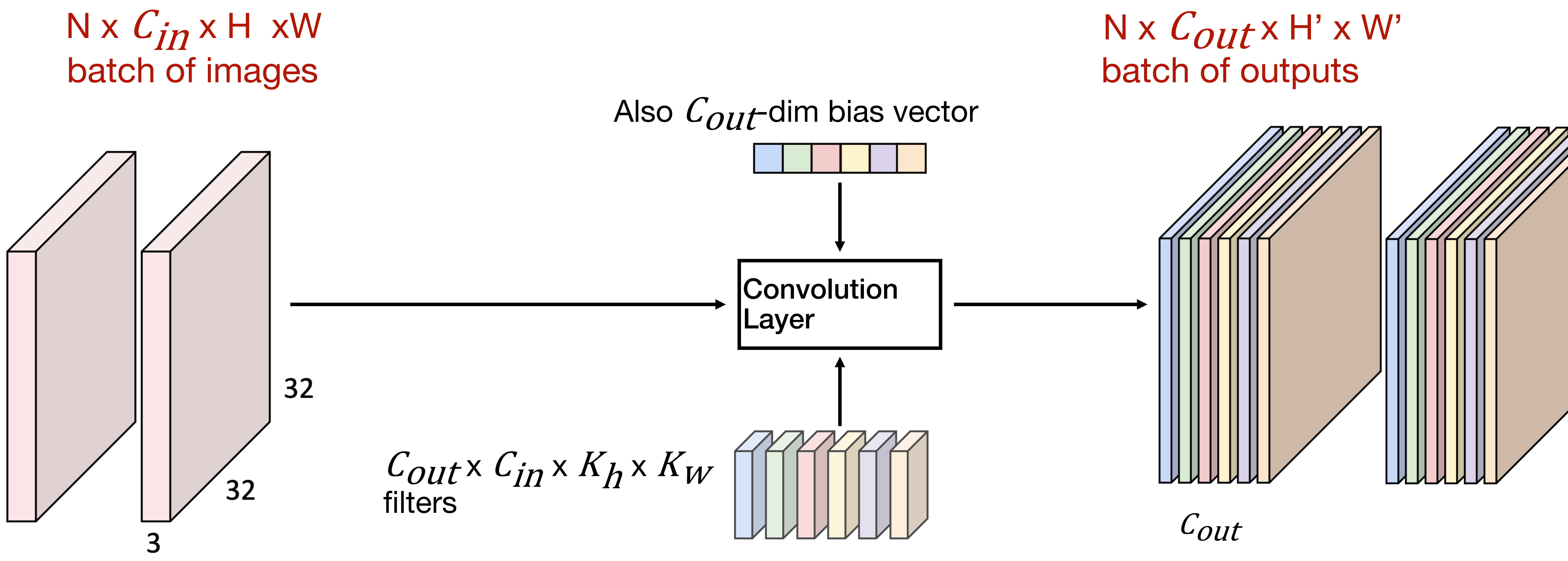
# Convolution Layer



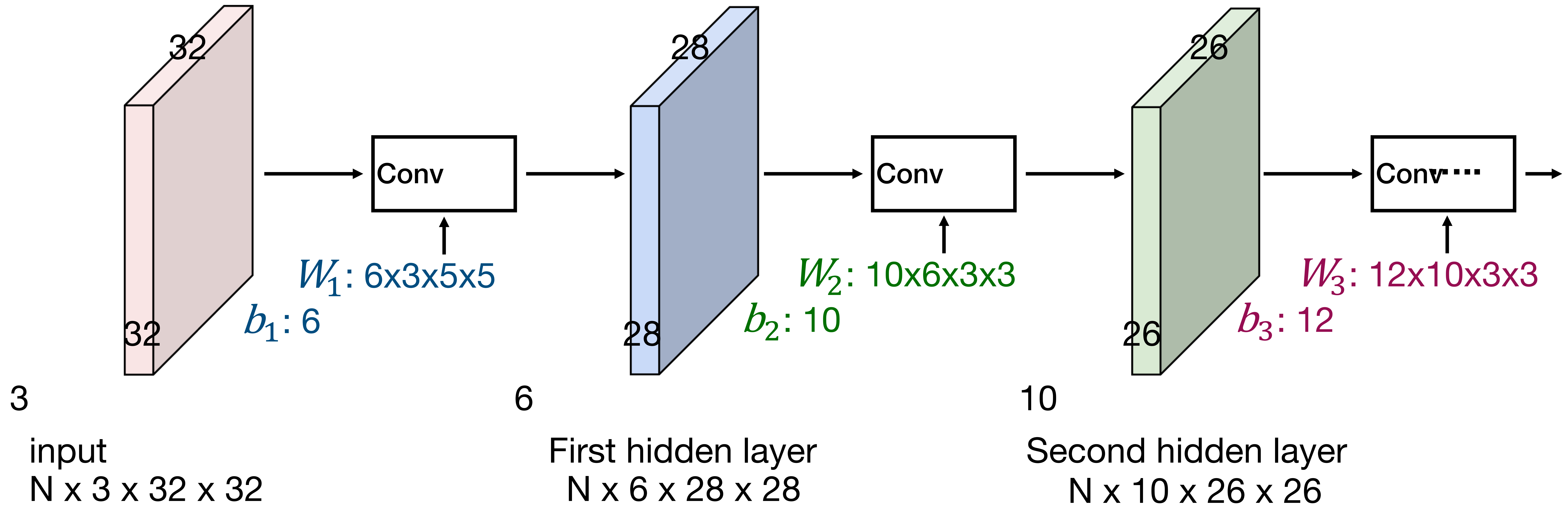
# Convolution Layer



# Convolution Layer: General dimensions

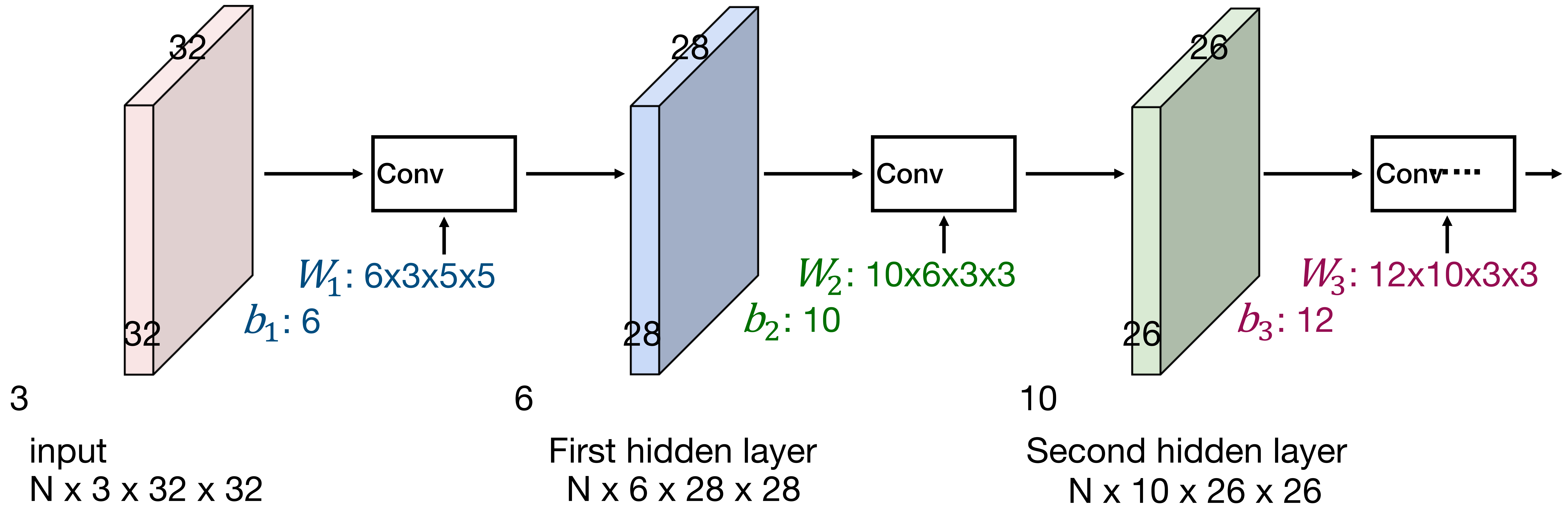


# Stacking Convolutions



# Stacking Convolutions

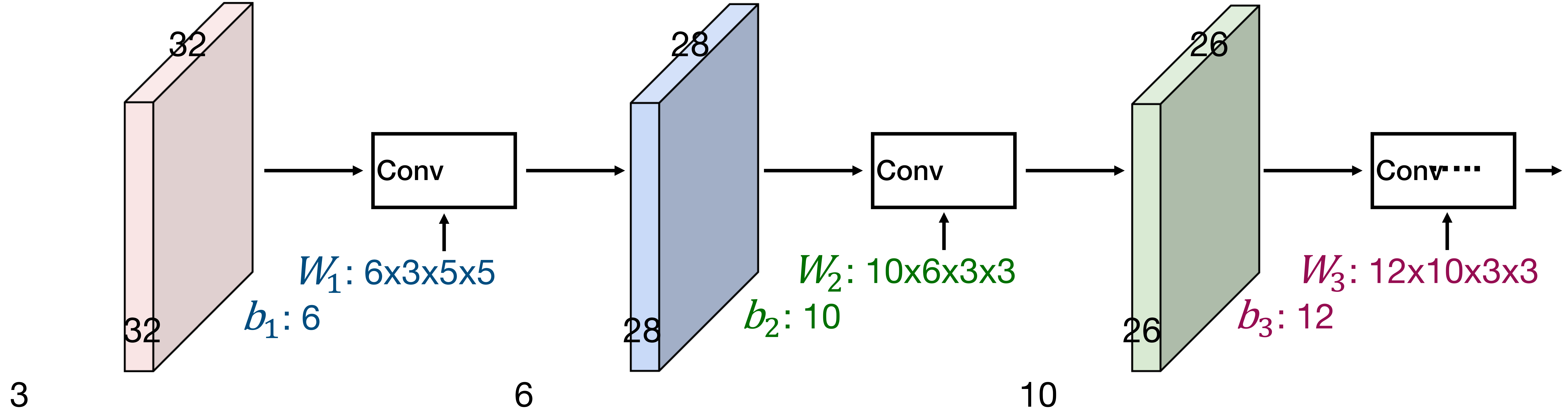
**Q:** What happens if we stack two convolution layers?



# Stacking Convolutions

**Q:** What happens if we stack two convolution layers?

(Recall y= $W_2 W_1$x is a linear classifier)




input
N x 3 x 32 x 32

First hidden layer
N x 6 x 28 x 28

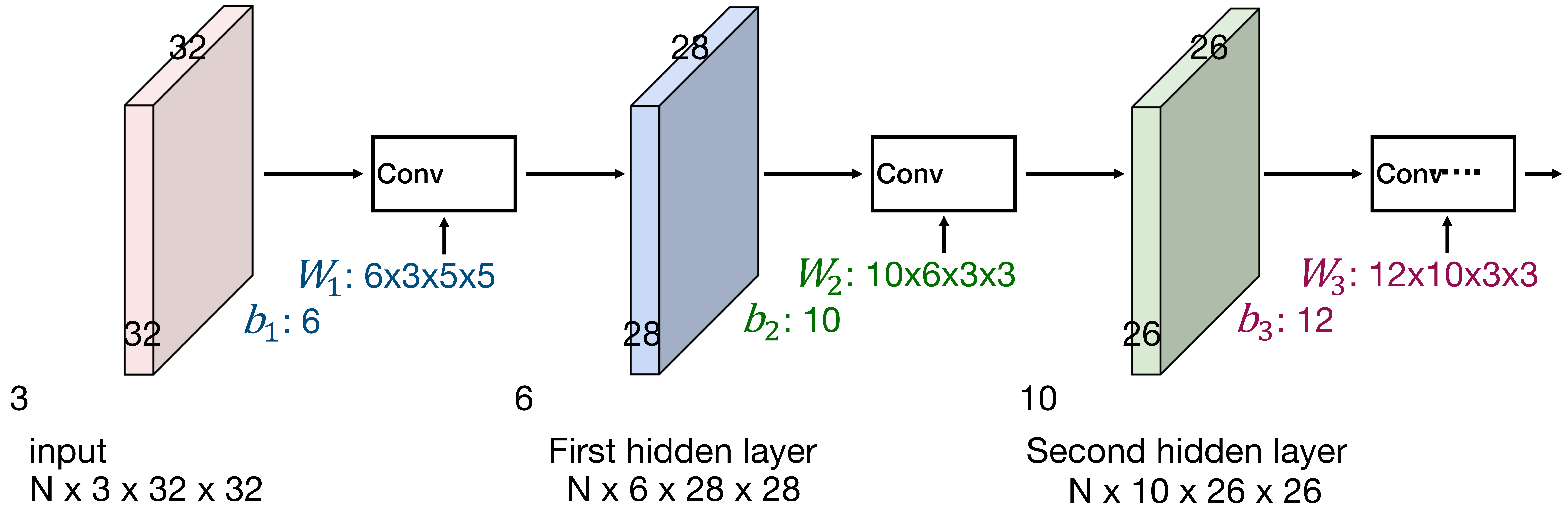
Second hidden layer
N x 10 x 26 x 26

# 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!

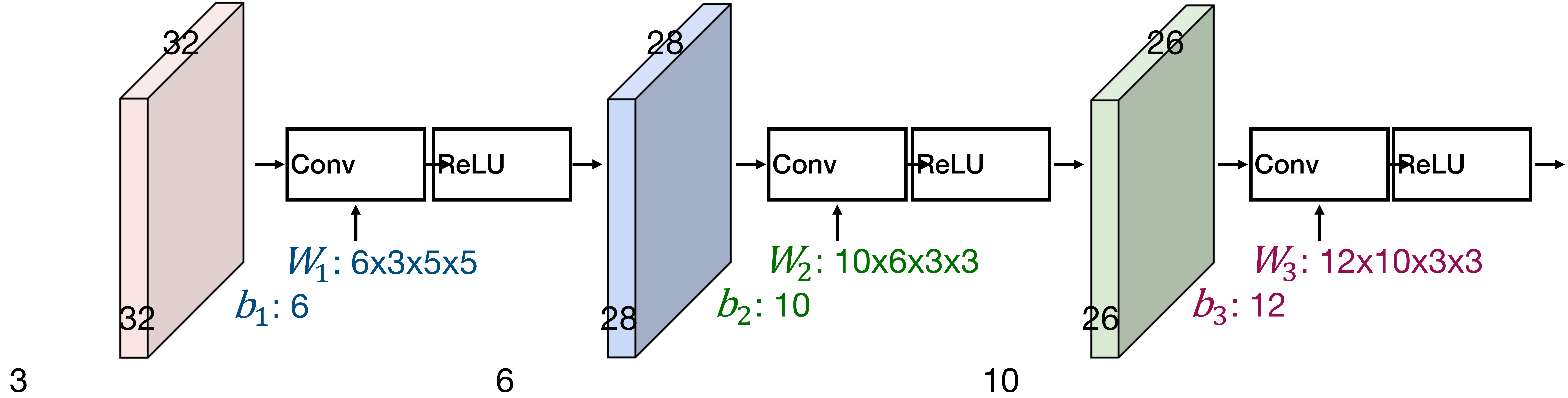


# Stacking Convolutions: insert activation function

**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!



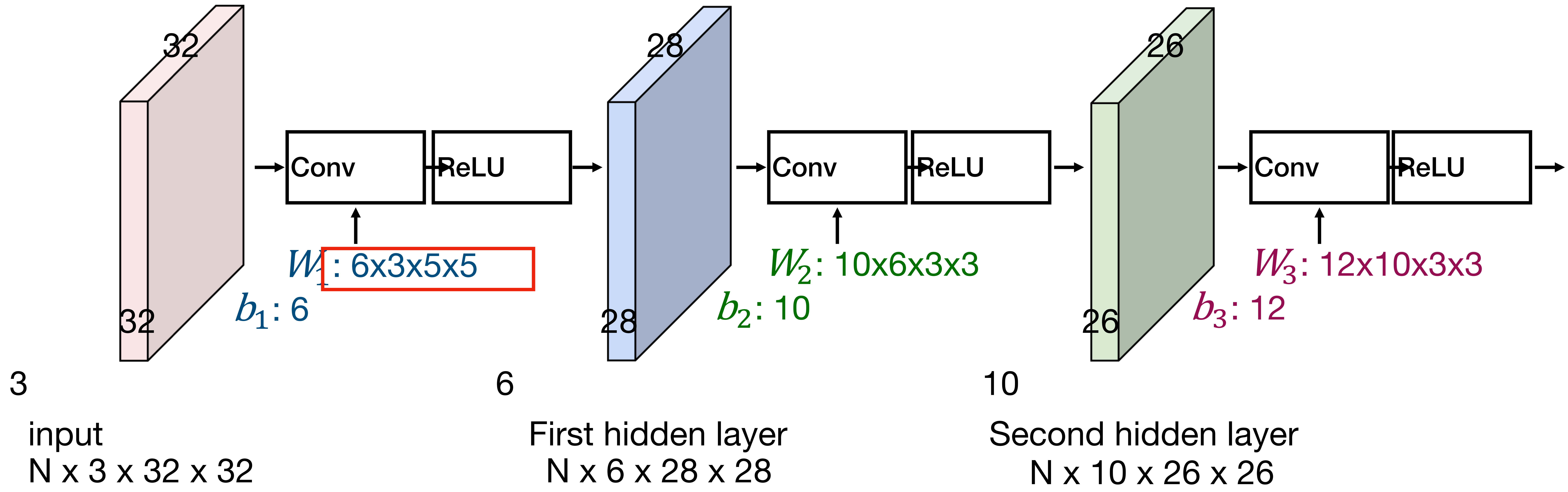

input
N x 3 x 32 x 32

First hidden layer
N x 6 x 28 x 28

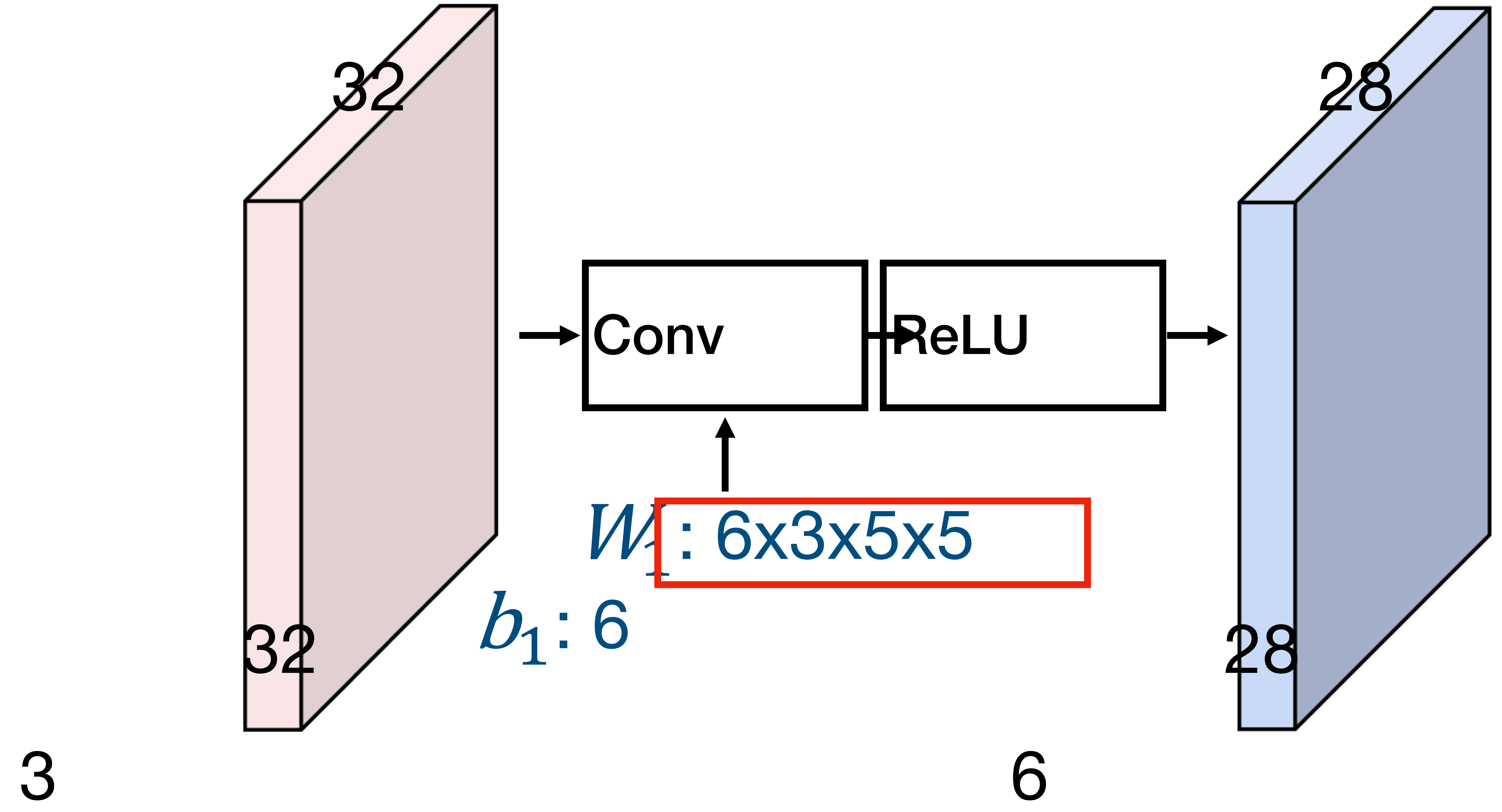
Second hidden layer
N x 10 x 26 x 26

Non-linear relationships

# What do convolutional filters learn?



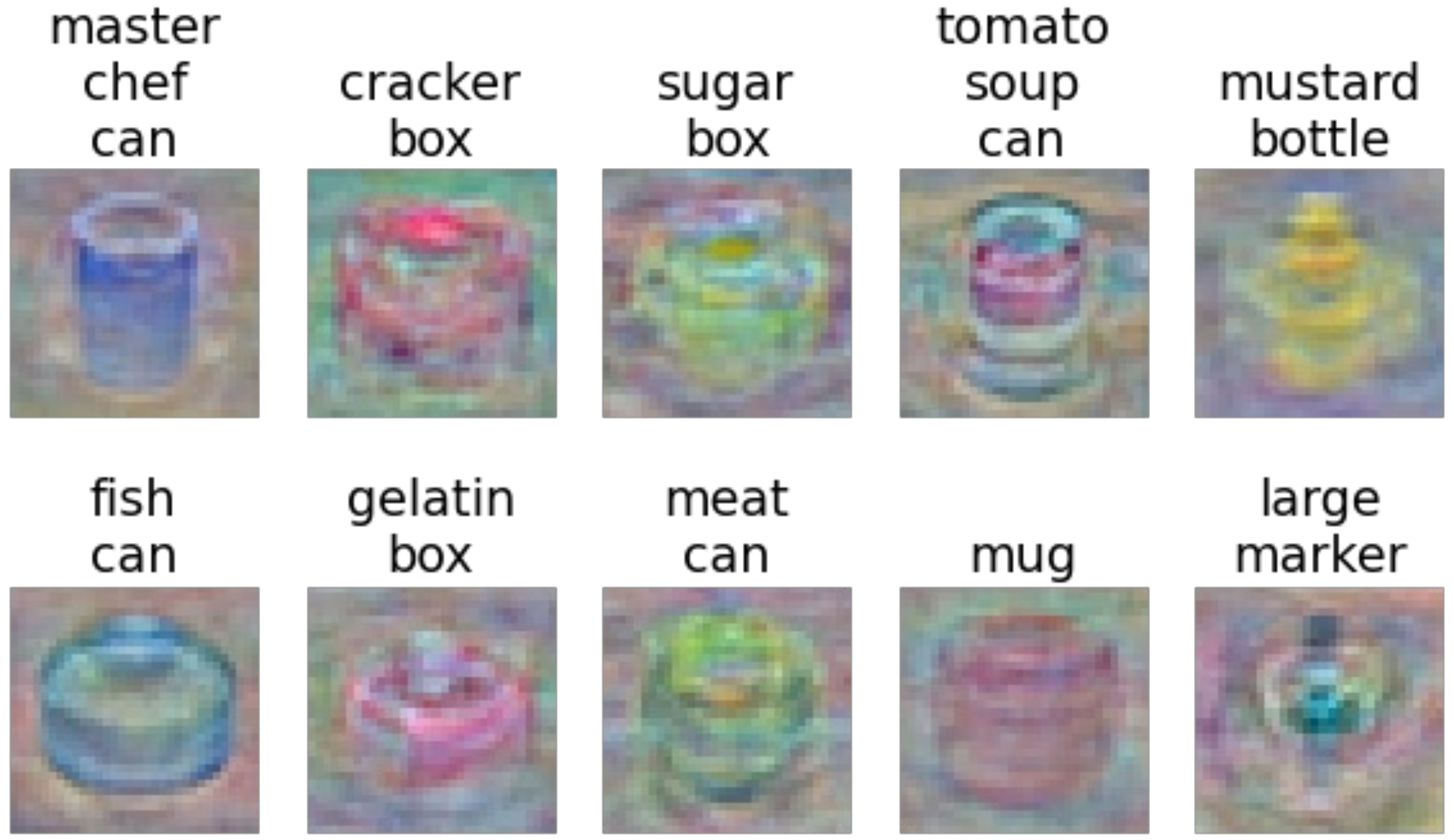
# What do convolutional filters learn?



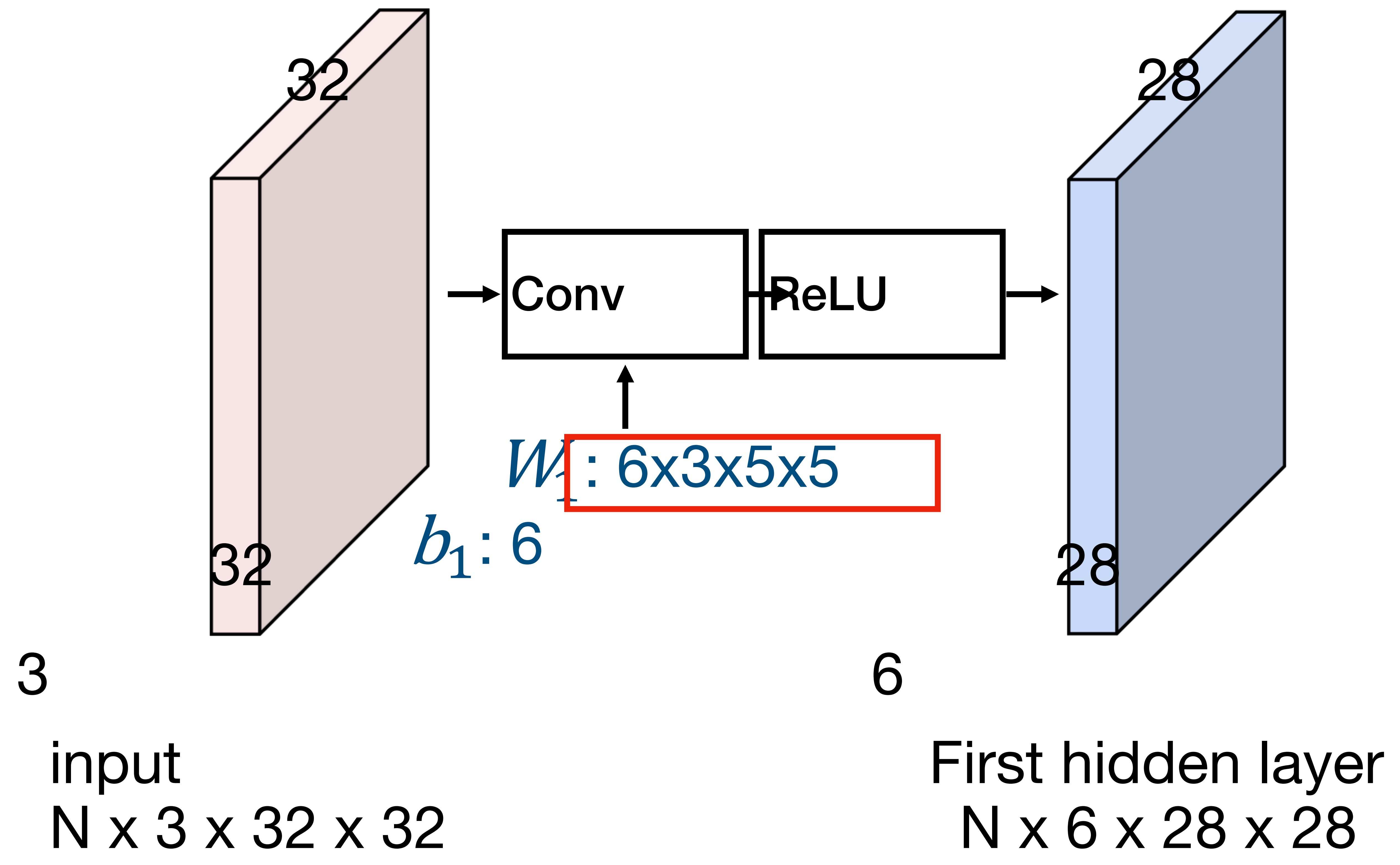

input
N x 3 x 32 x 32

First hidden layer
N x 6 x 28 x 28

Linear classifier: One template per class



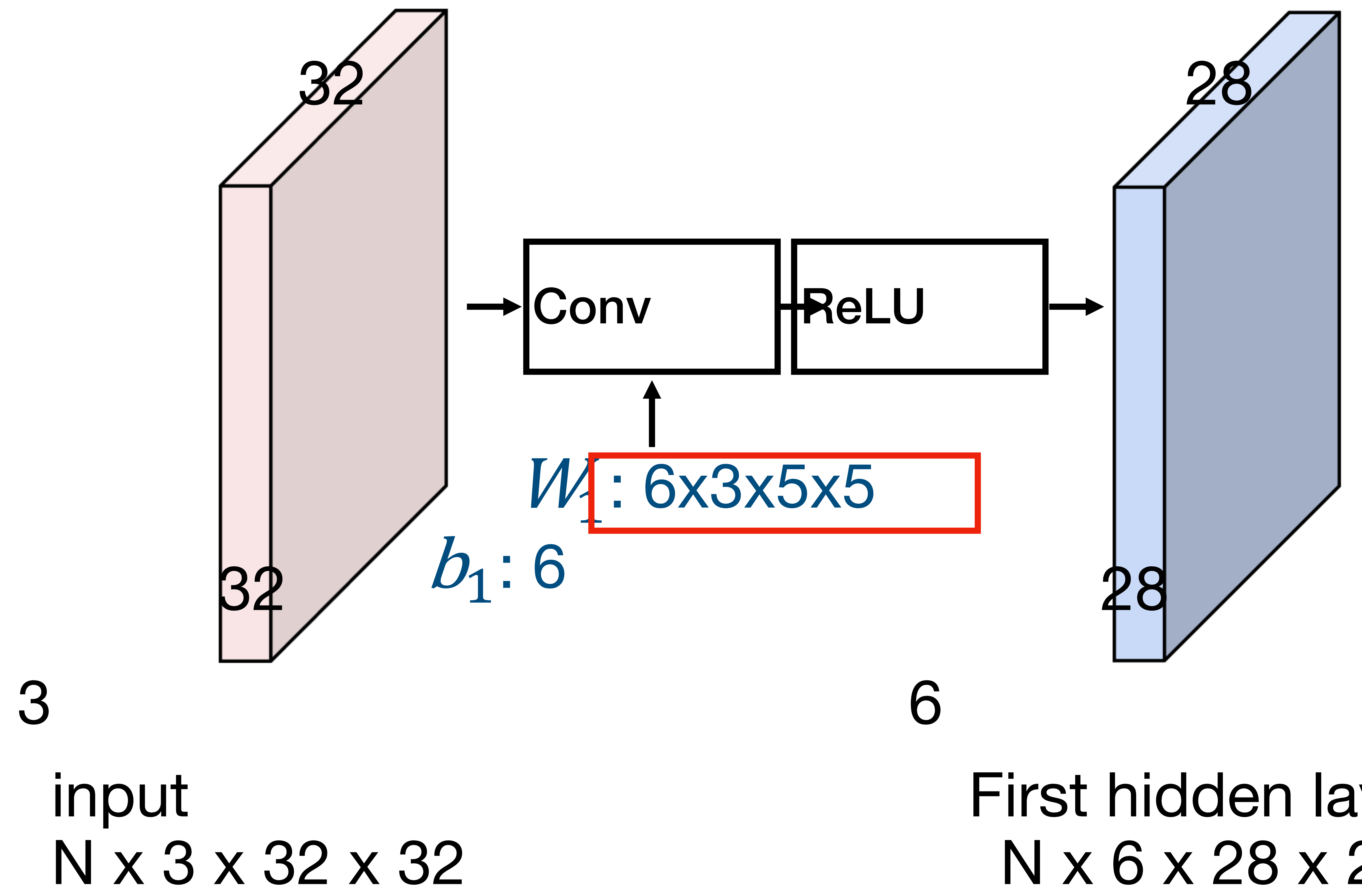
# What do convolutional filters learn?




MLP: Bank of whole-image templates

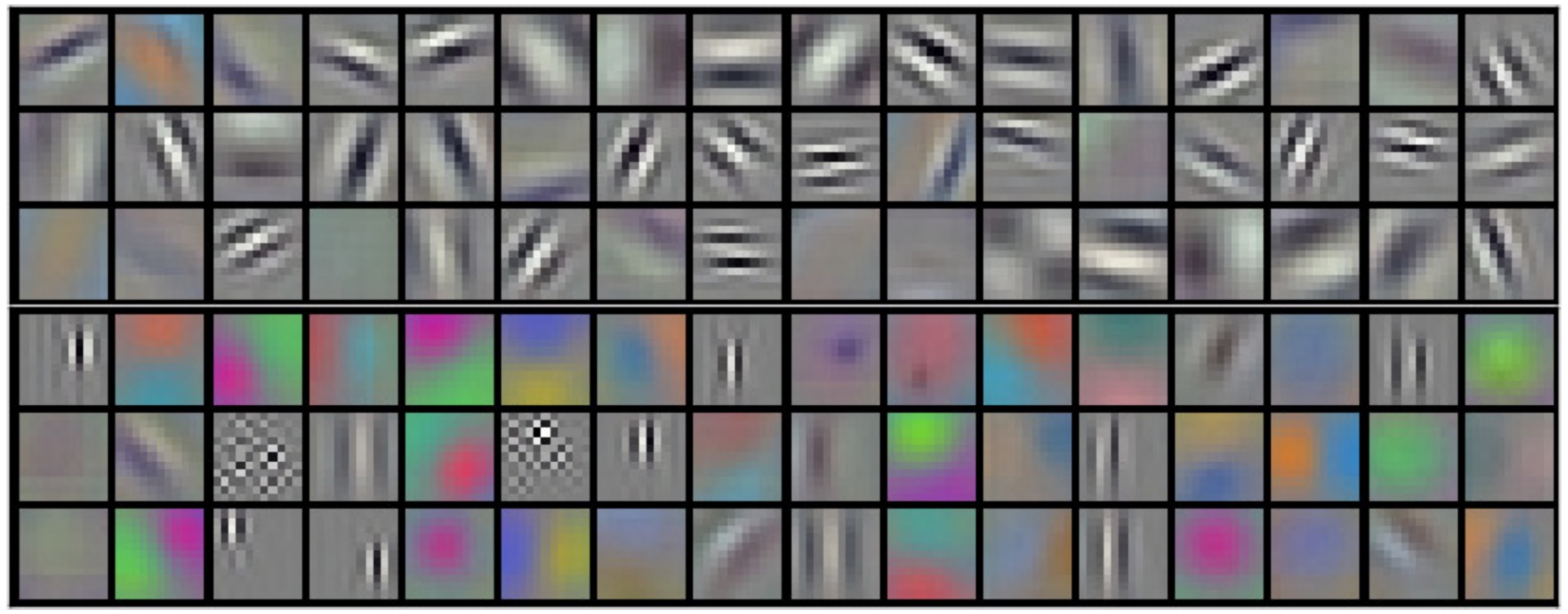
* Global wrt. the entire image

# What do convolutional filters learn?




First-layer conv filters: local image templates

(often learns oriented edges, opposing colors)

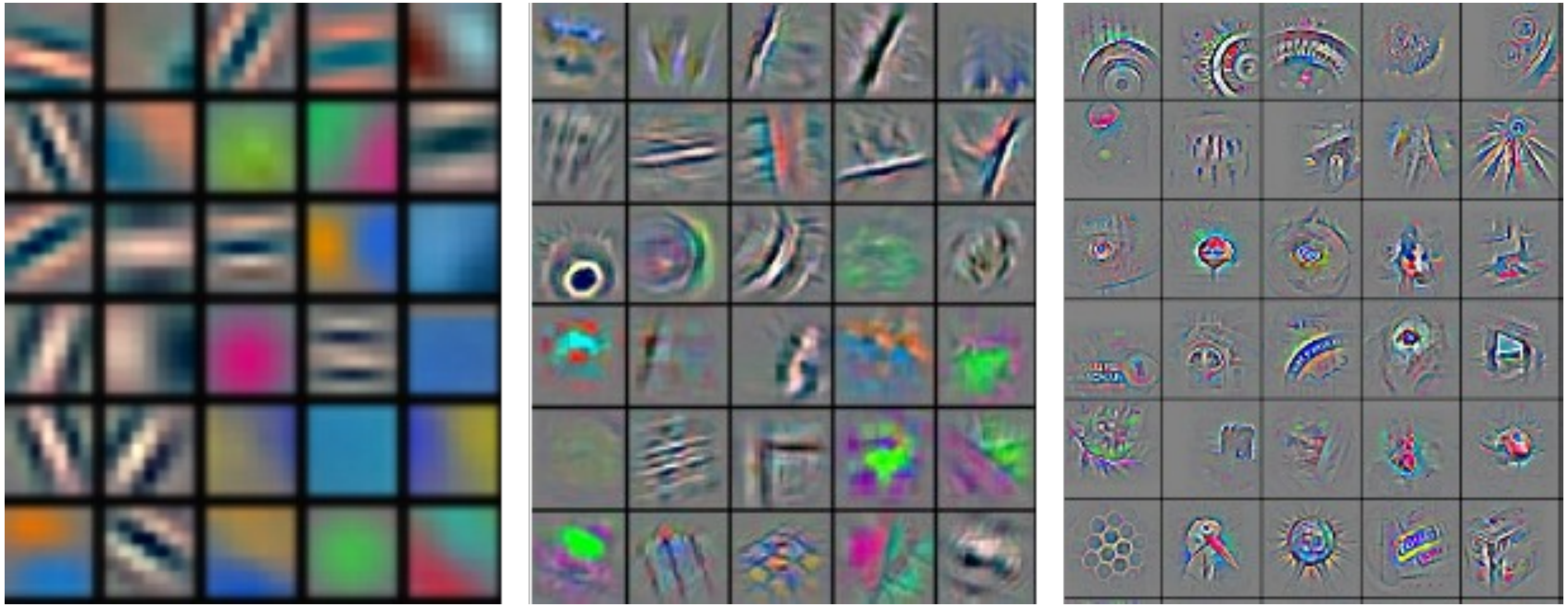


AlexNet: 96 filters, each 3x11x11

* Local

# What do convolutional filters learn?

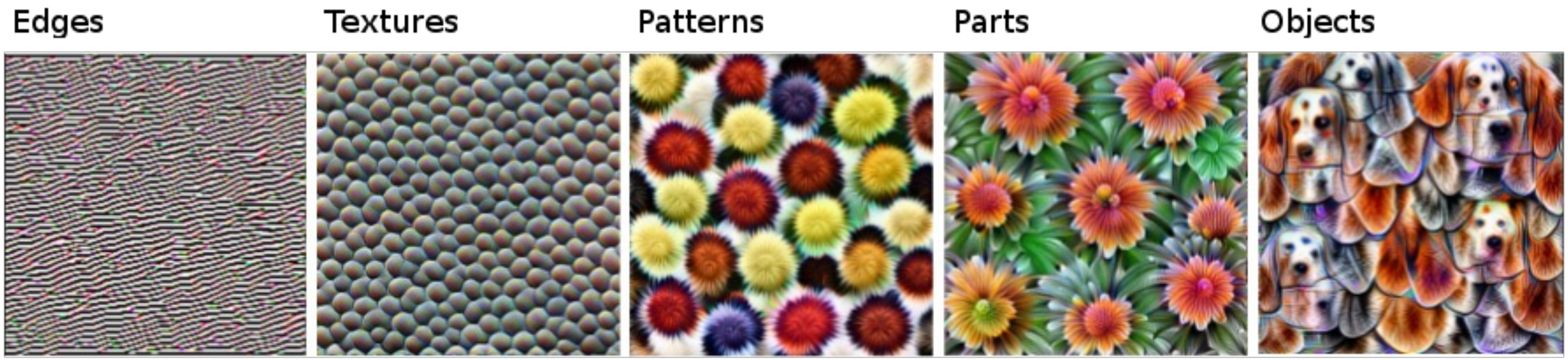
## Feature visualization

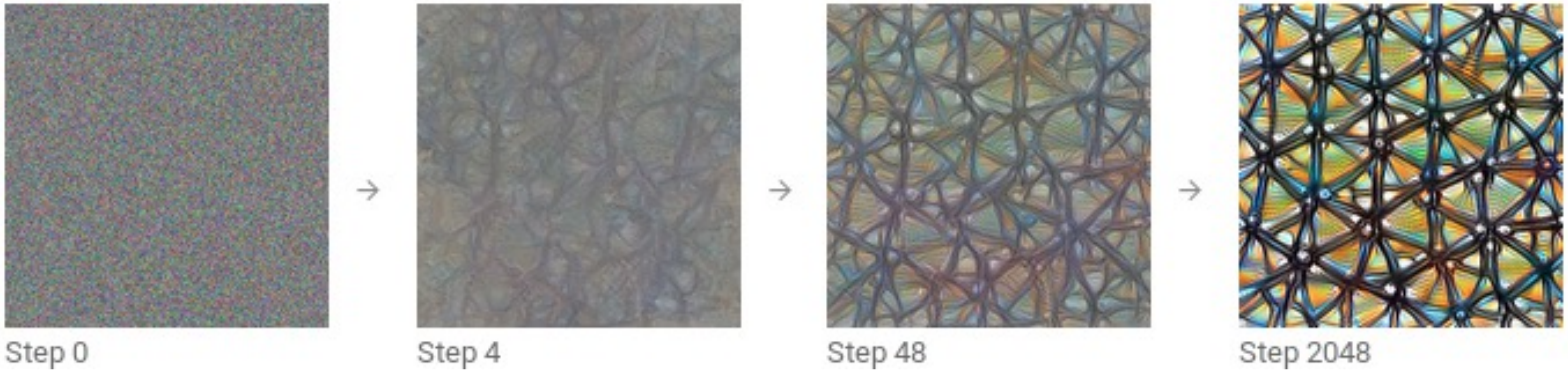


# What do convolutional filters learn?

## Feature visualization

distill.pub

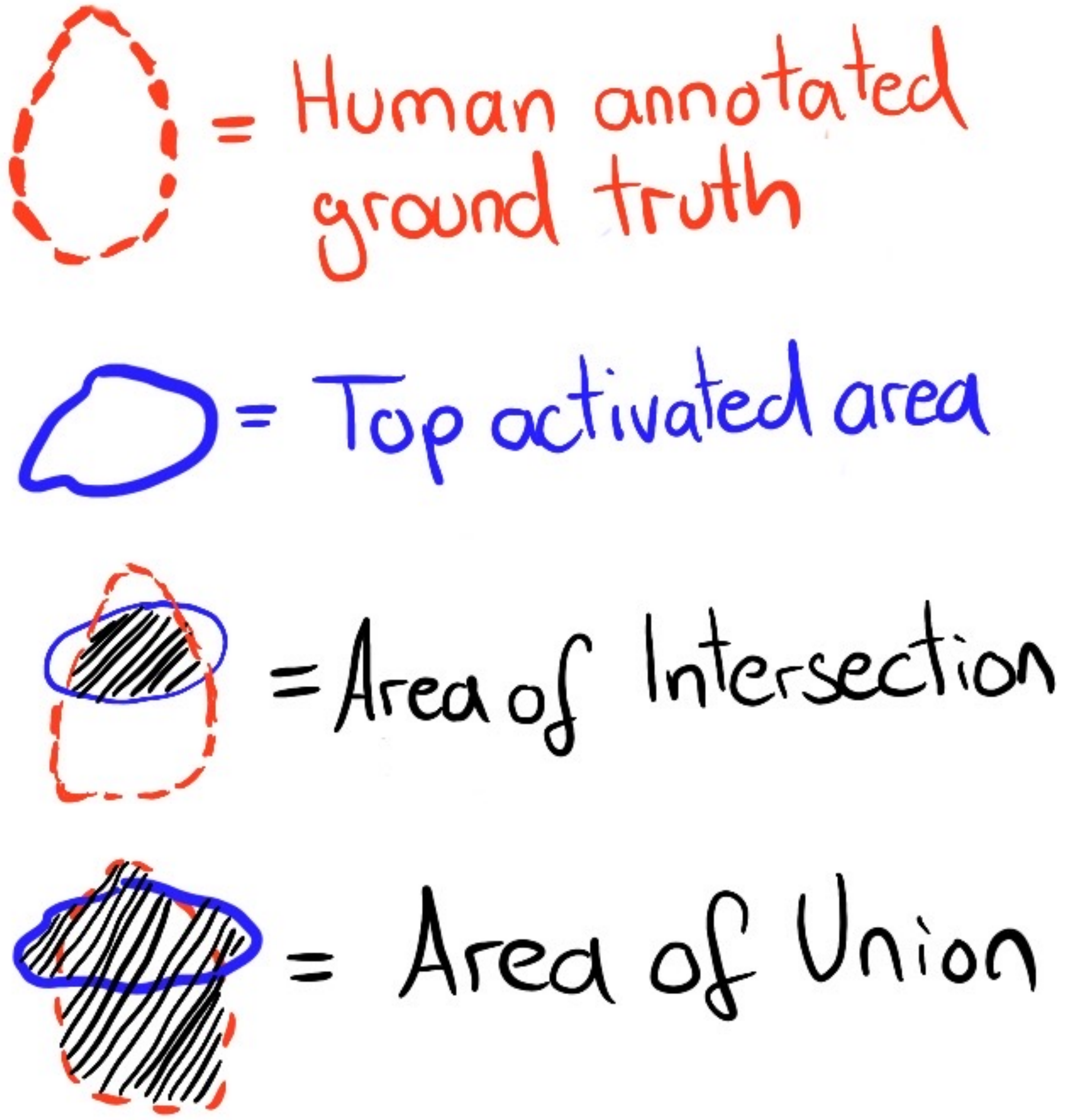


# What do convolutional filters learn?

## Activation mask

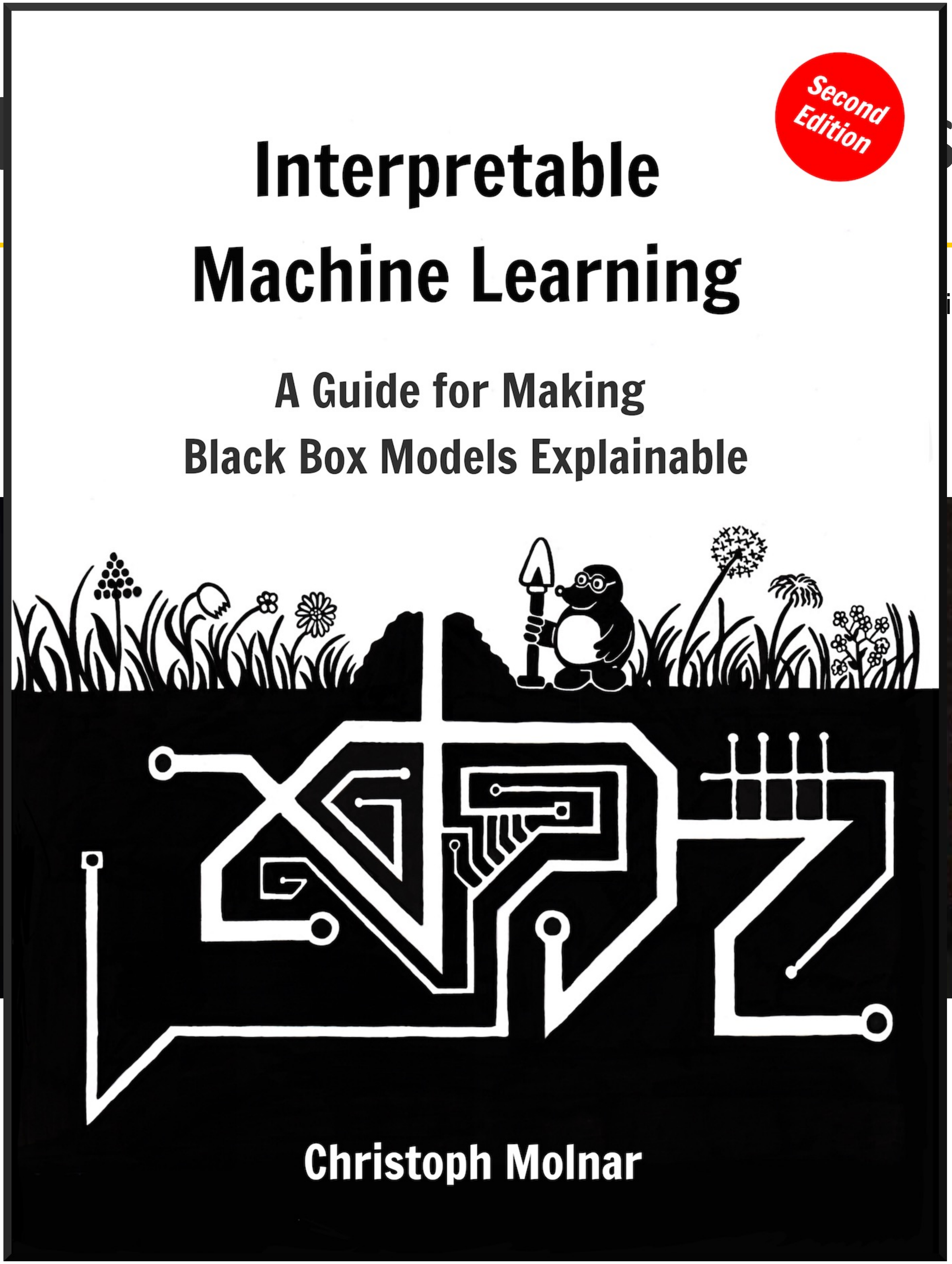
https://christophm.github.io/interpretable-ml-book/cnn-features.html

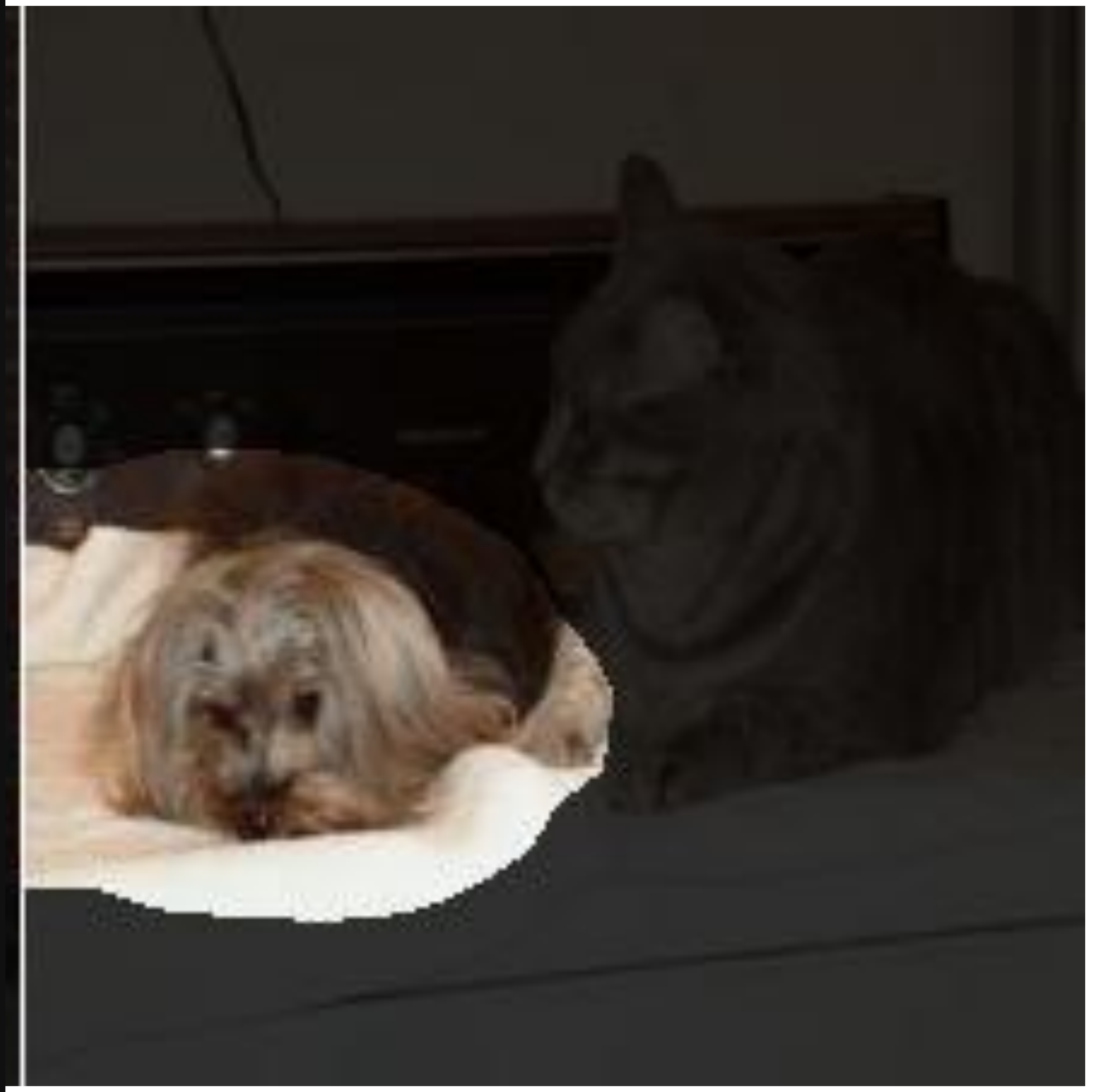


# What d s learn?

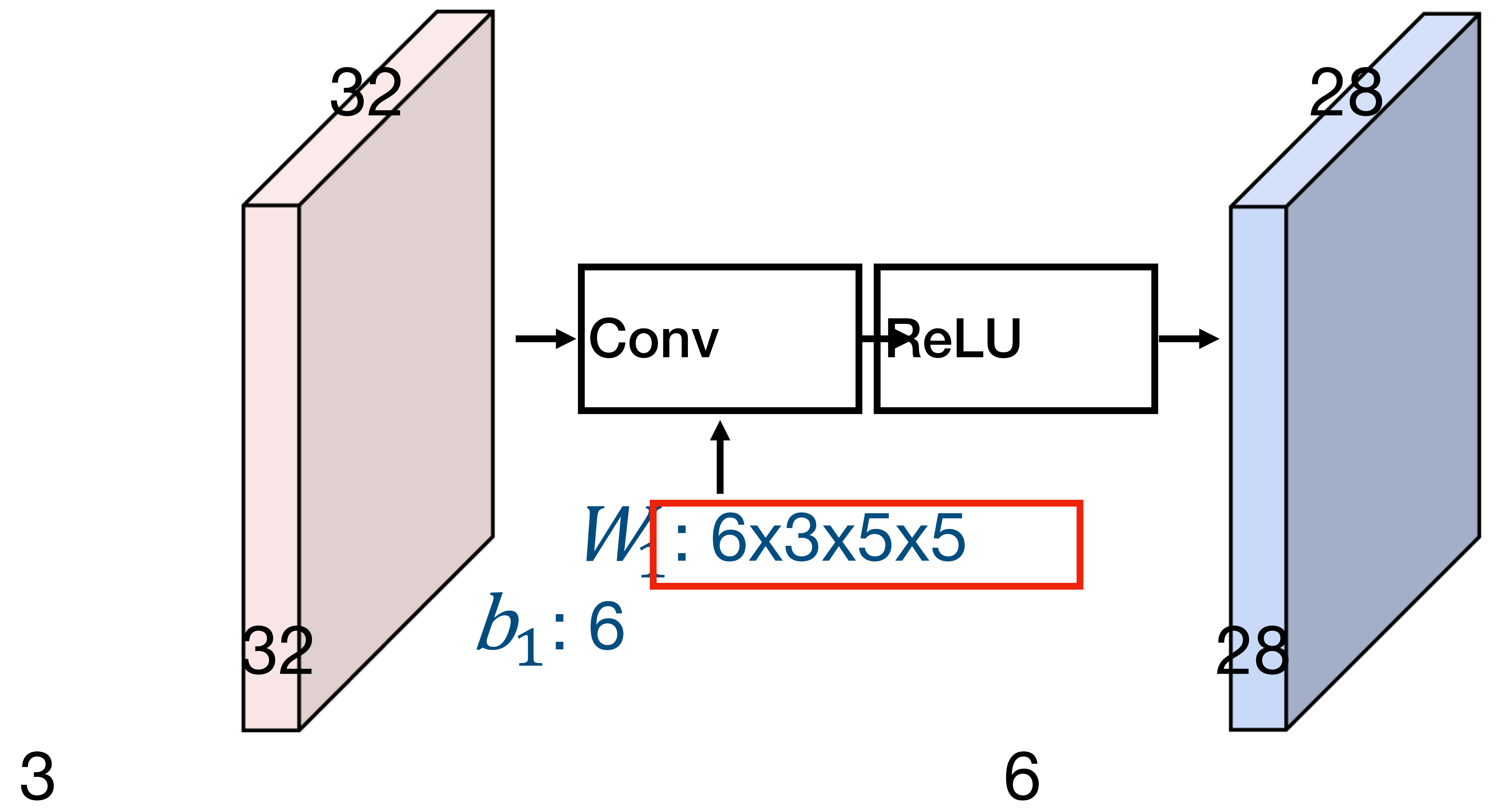
## Activation mask

ithub.io/interpretable-ml-book/cnn-features.html



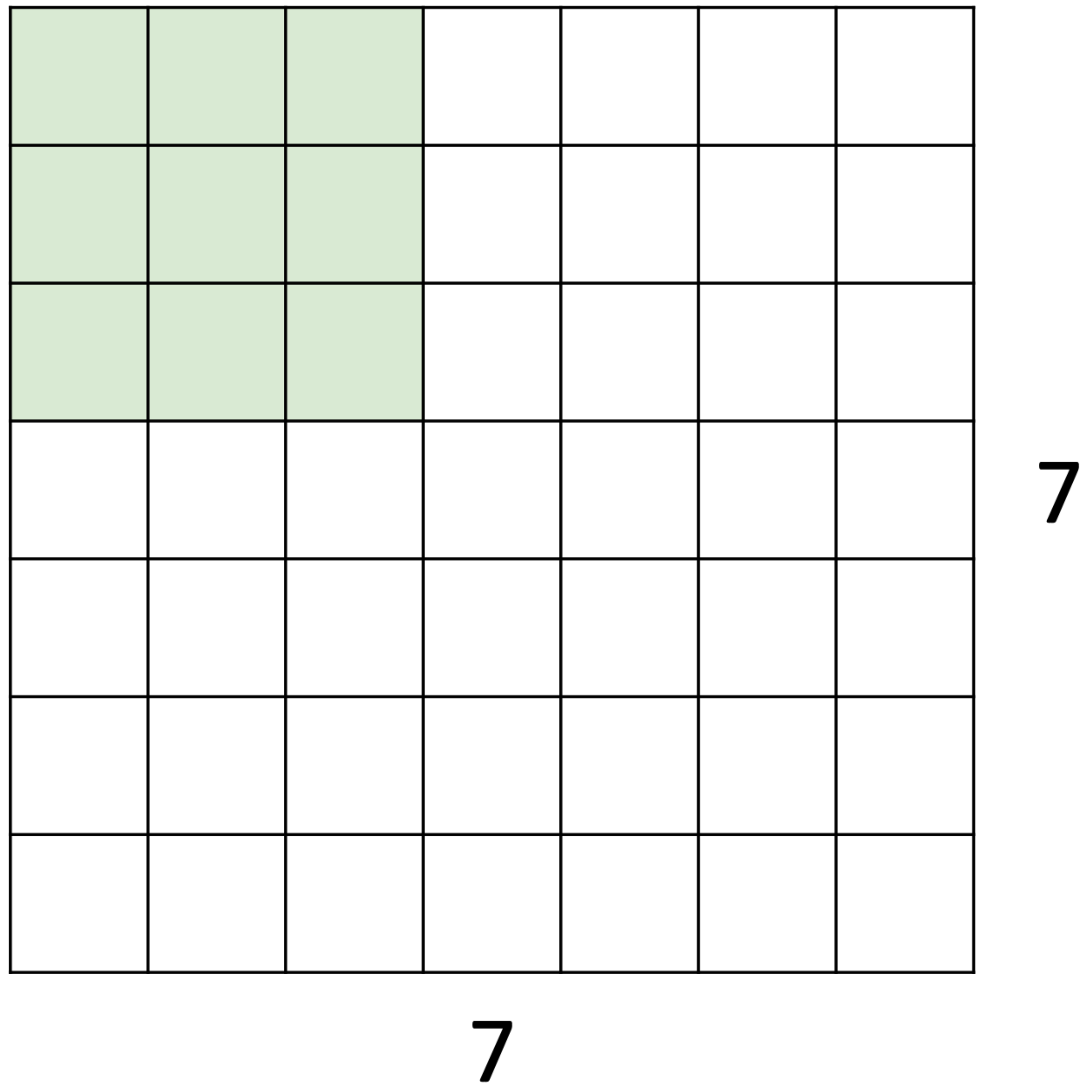
# A closer look at spatial dimensions




input
N x 3 x 32 x 32

First hidden layer
N x 6 x 28 x 28

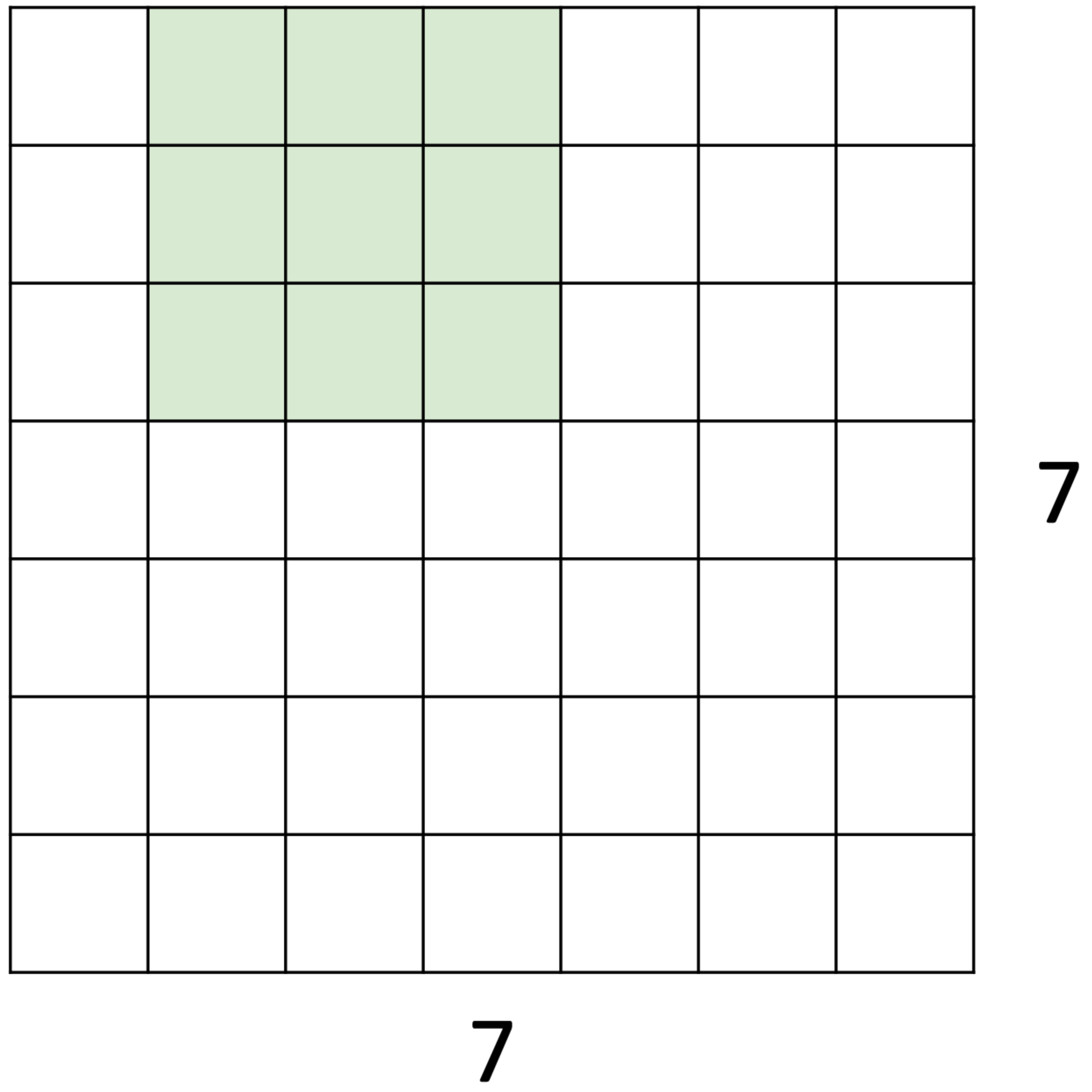
# A closer look at spatial dimensions




Input: 7x7

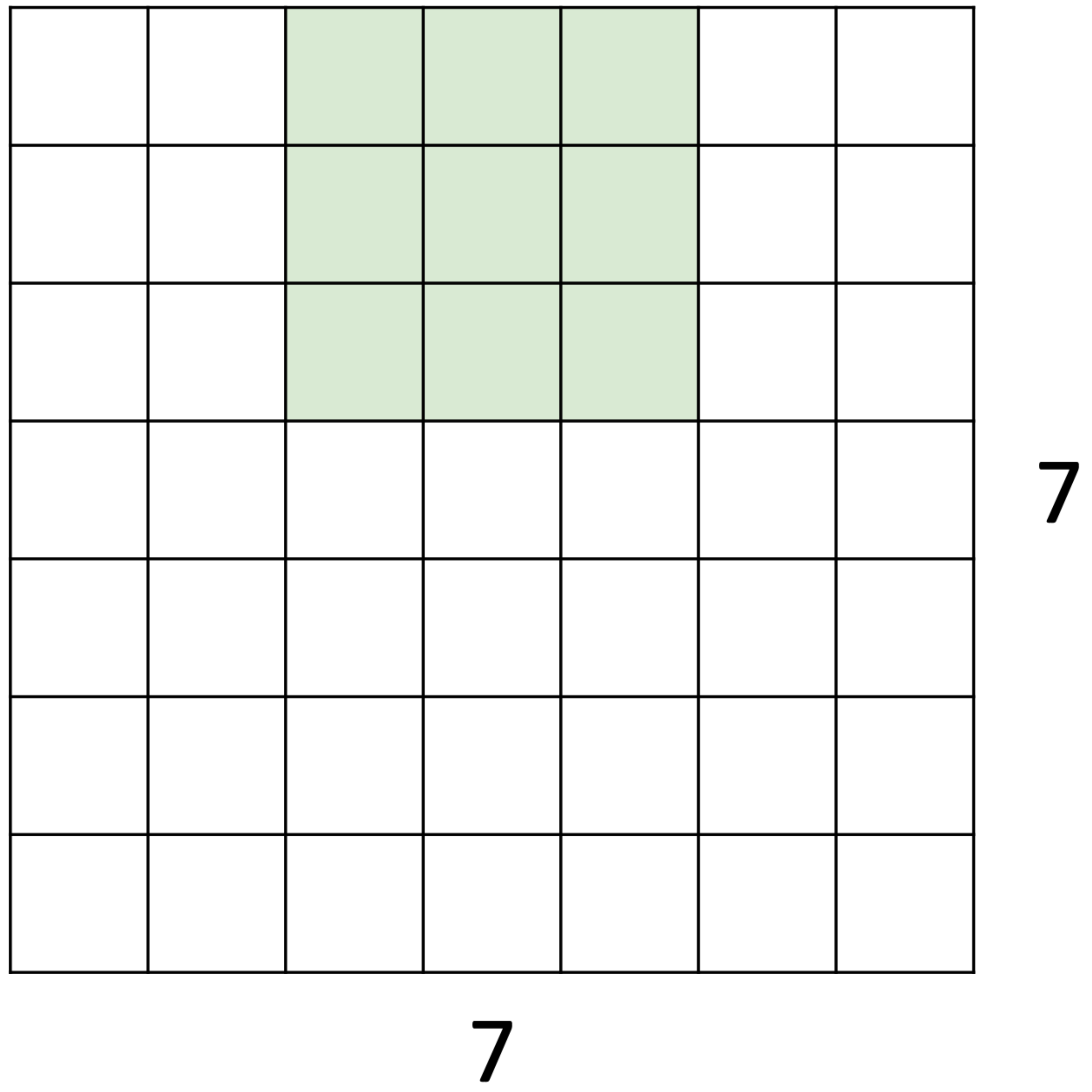
Filter: 3x3

# A closer look at spatial dimensions



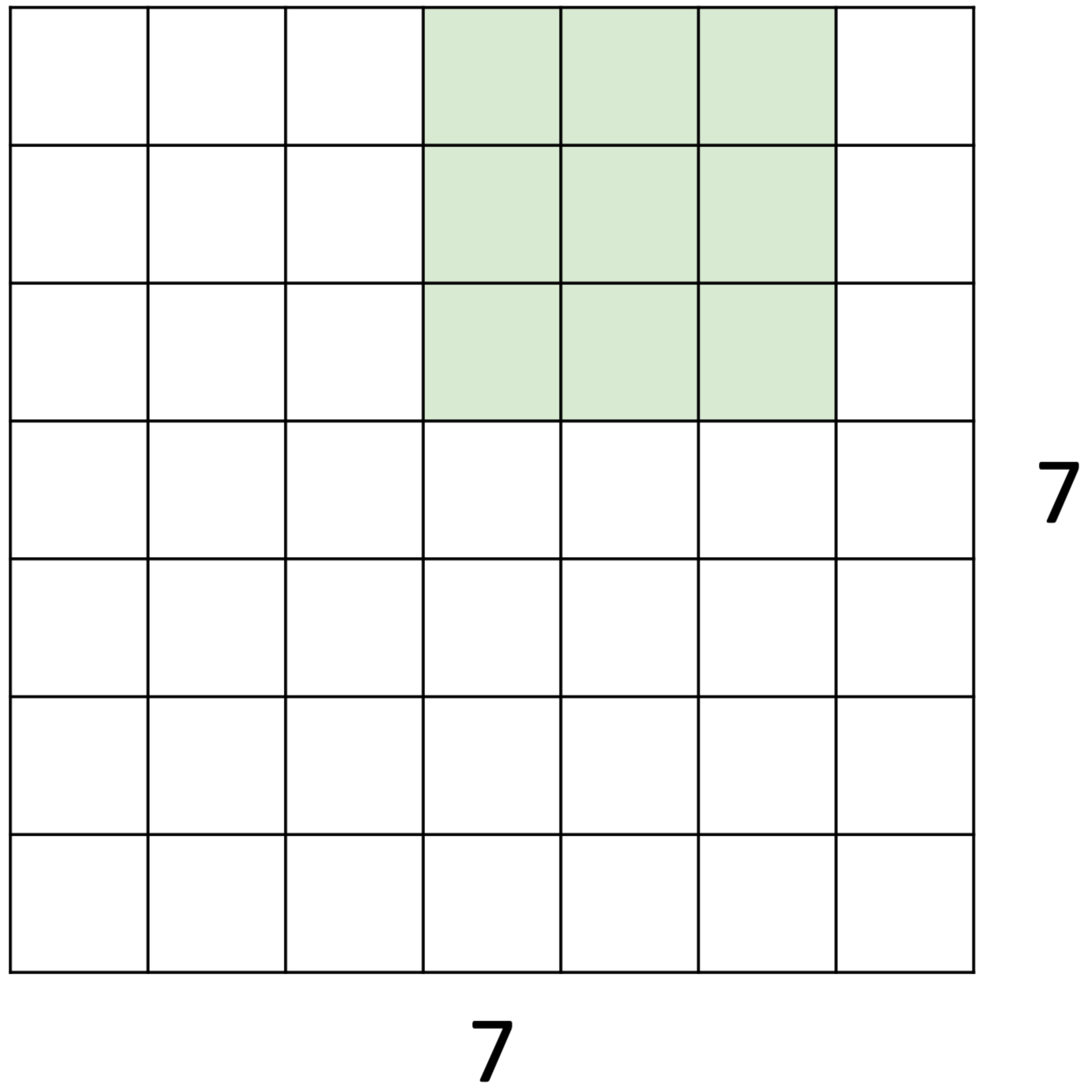

Input: 7x7
Filter: 3x3

# A closer look at spatial dimensions



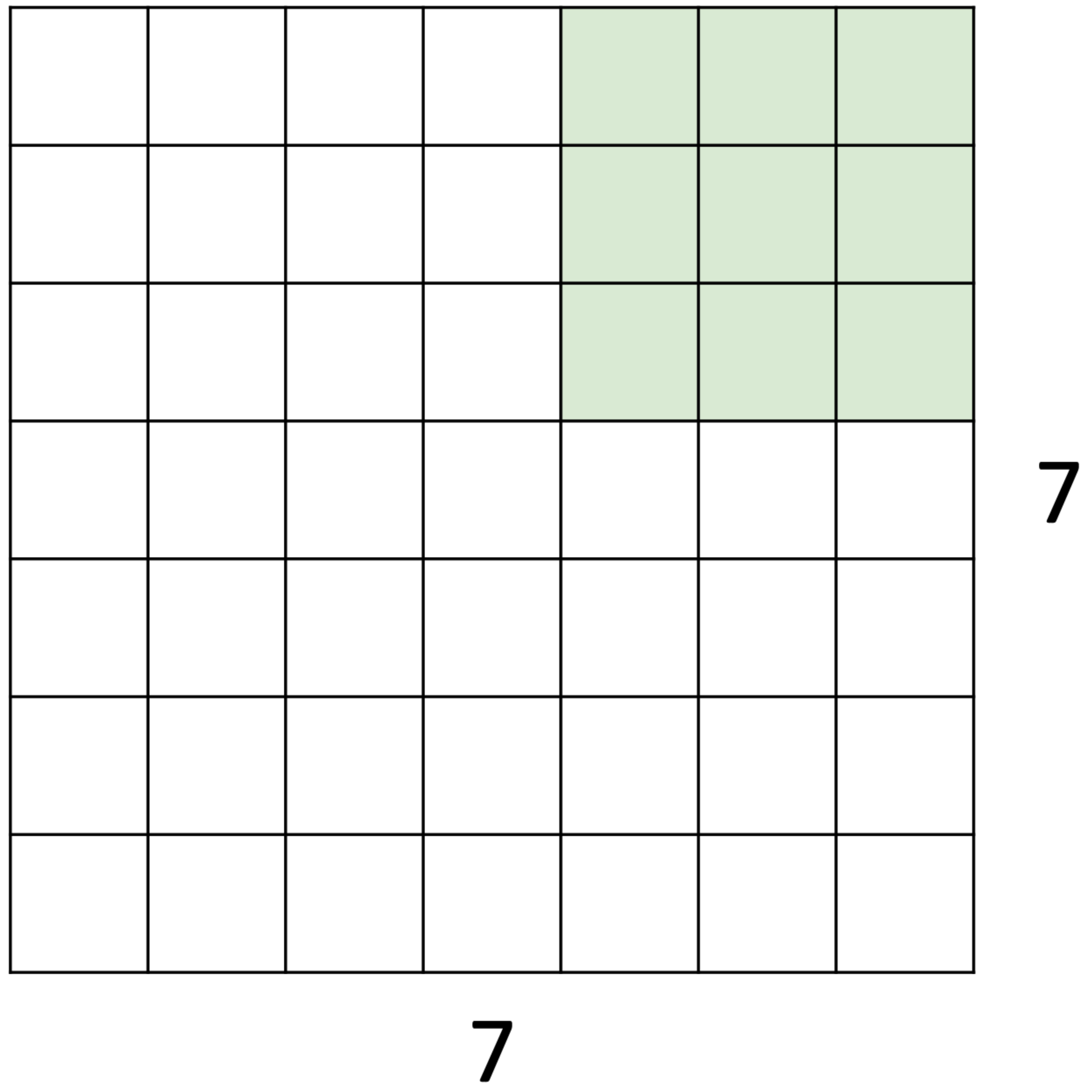

Input: 7x7
Filter: 3x3

# A closer look at spatial dimensions



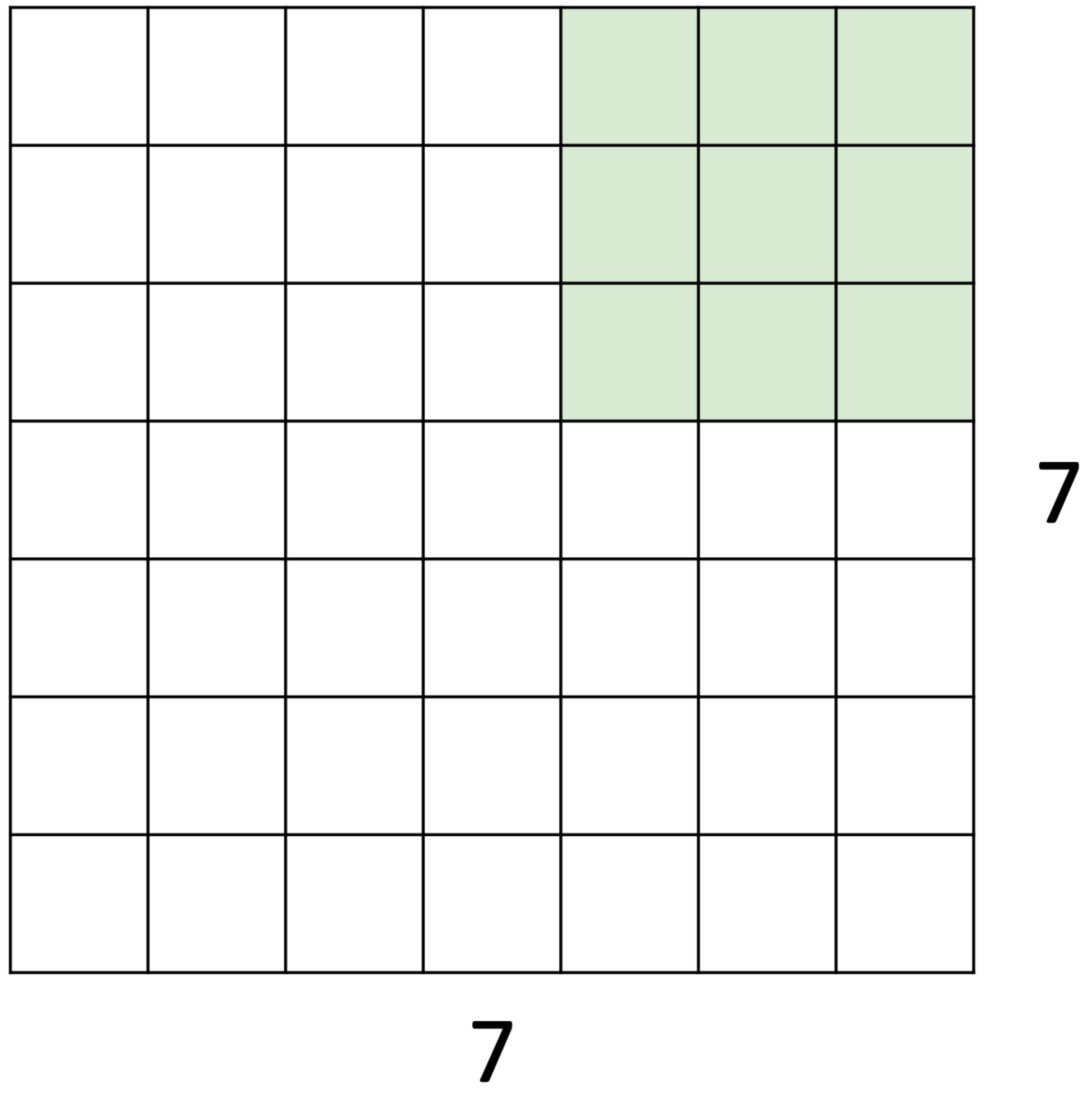

Input: 7x7
Filter: 3x3

# 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 |

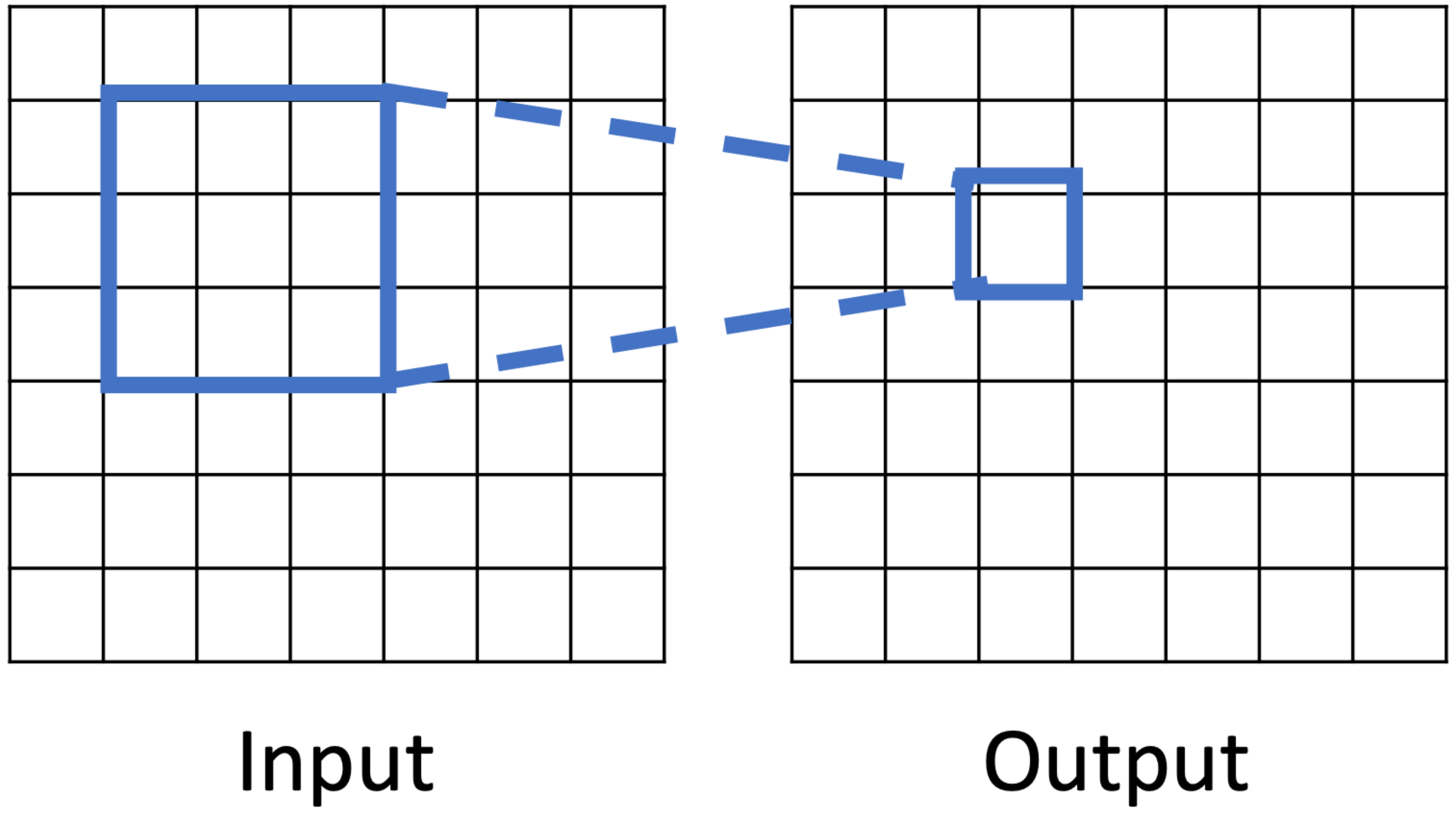
Input: 7x7
Filter: 3x3
Output: 5x5

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!

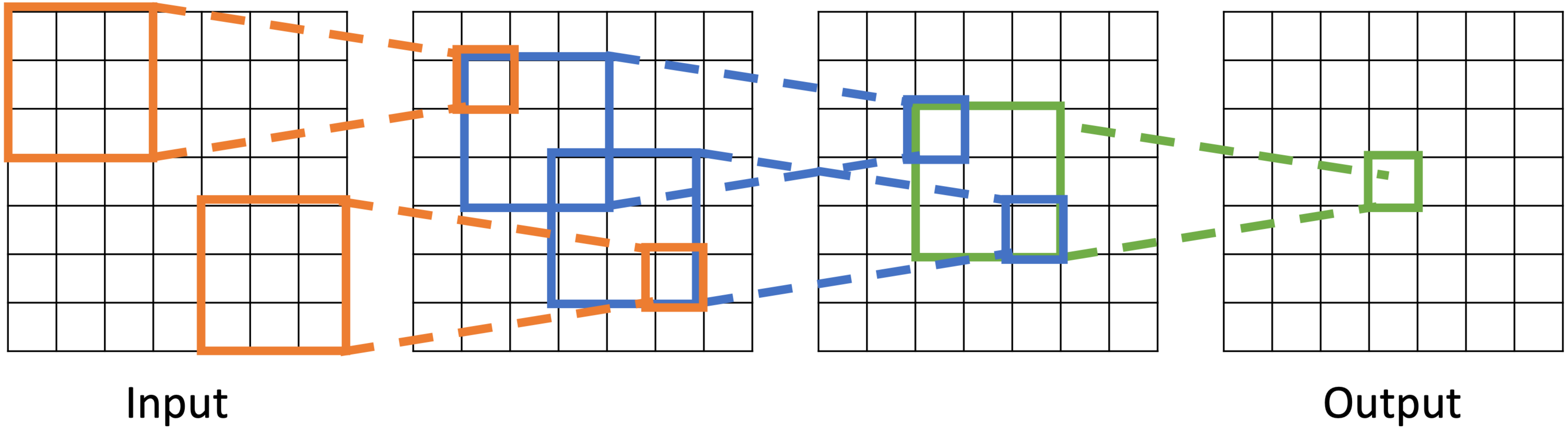
# Receptive Fields

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



# Receptive Fields

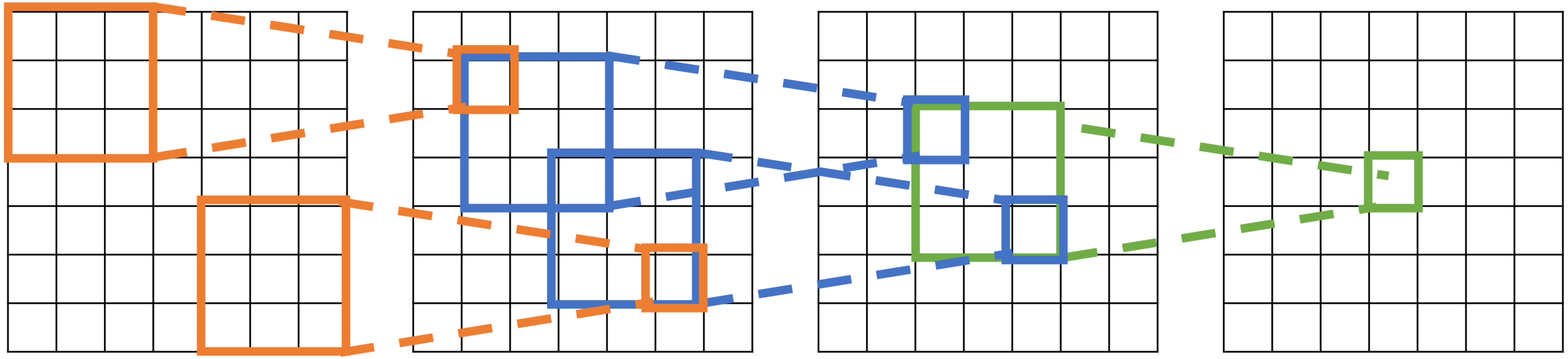
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)




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)



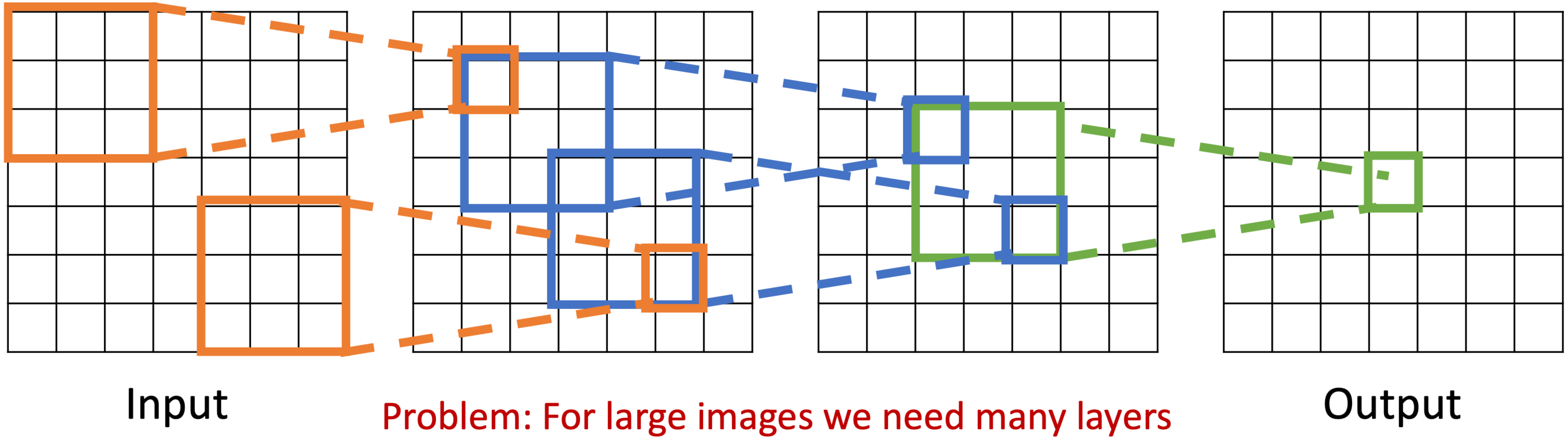
Input

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)

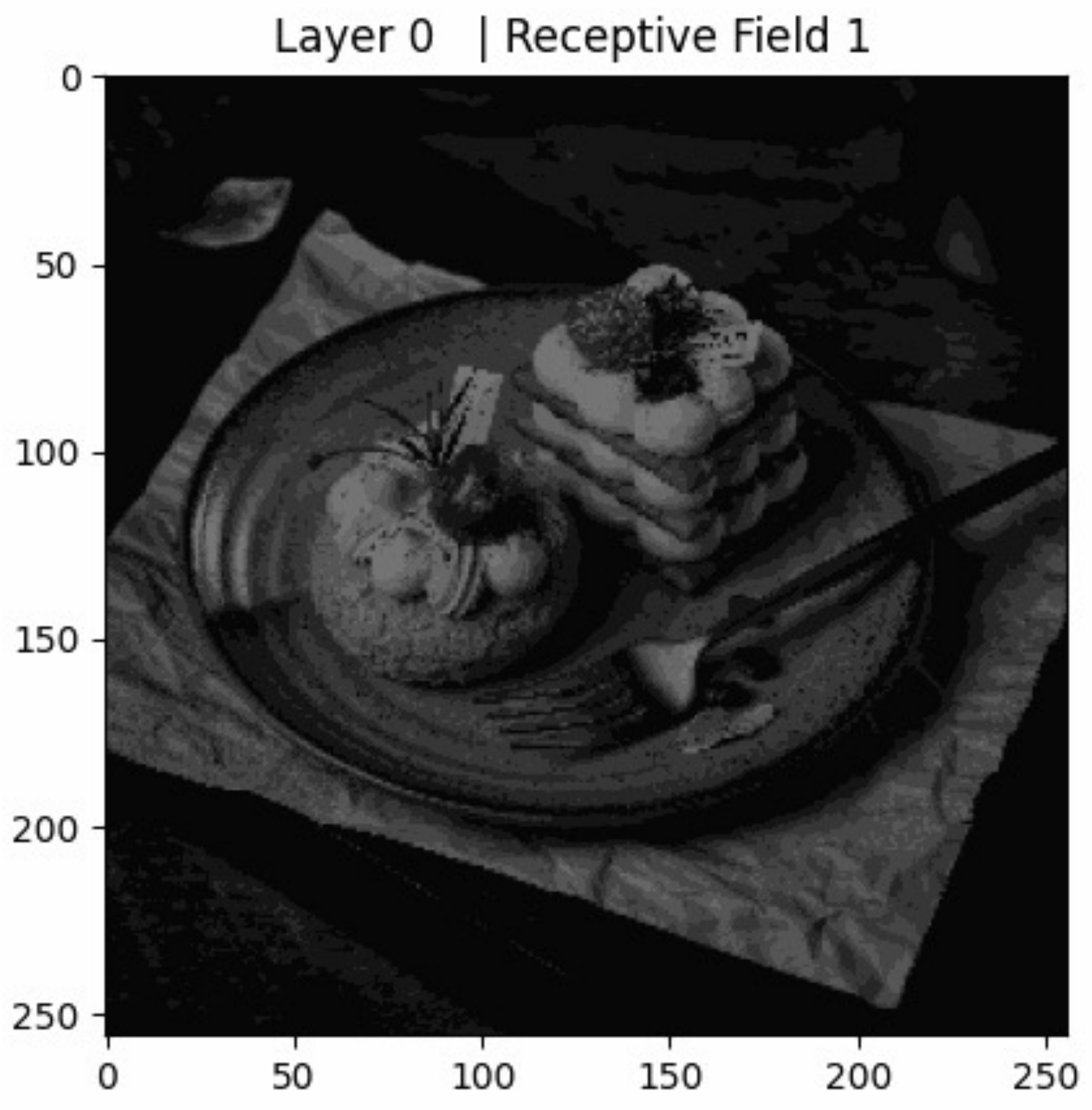



Problem: For large images we need many layers for each output to “see” the whole image image

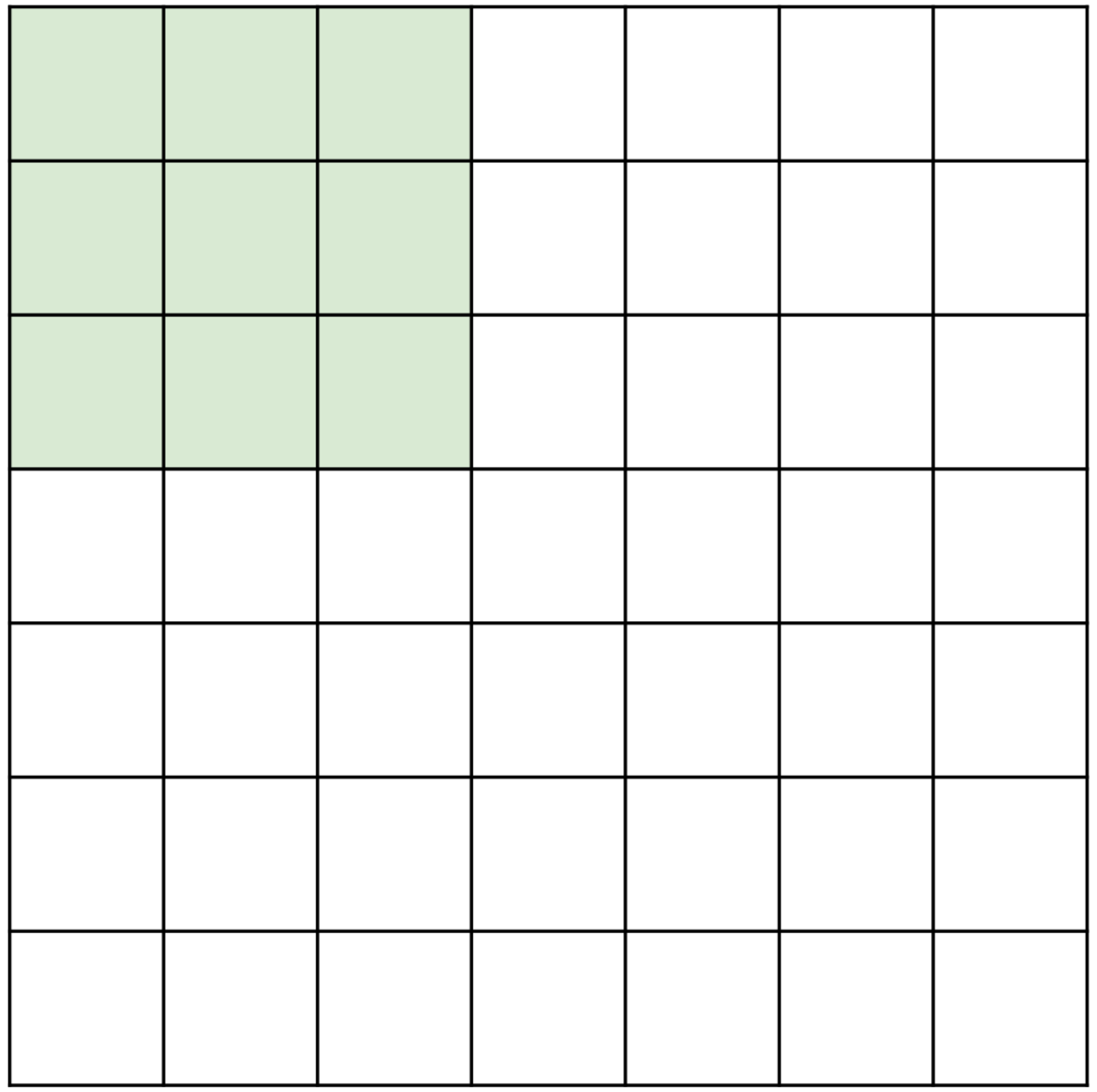
Solution: Downsample inside the network

# Receptive Fields

https://github.com/Fangyh09/pytorch-receptive-field



# Strided Convolution

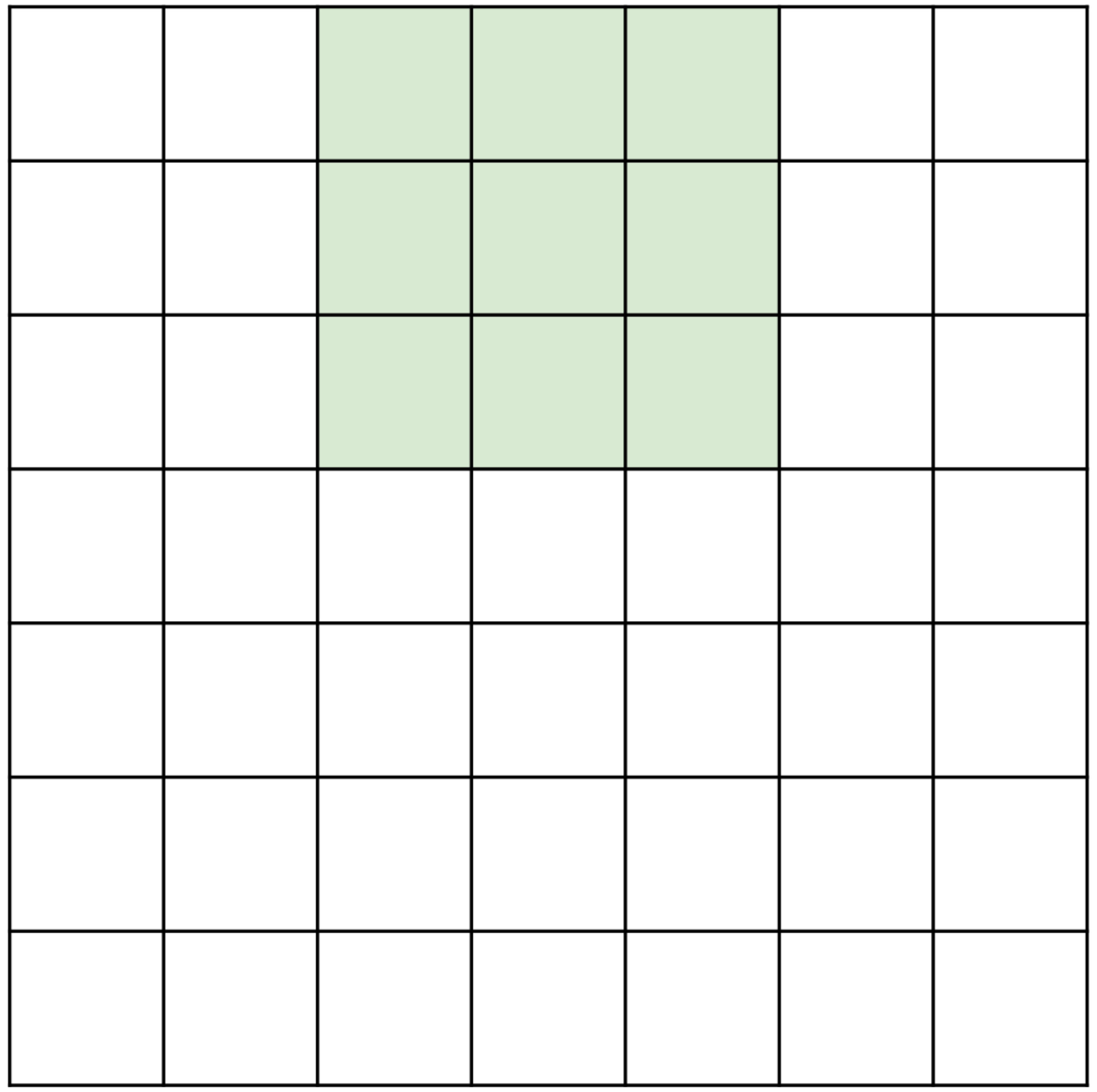


Input: 7x7

Filter: 3x3

Stride: 2

# Strided Convolution

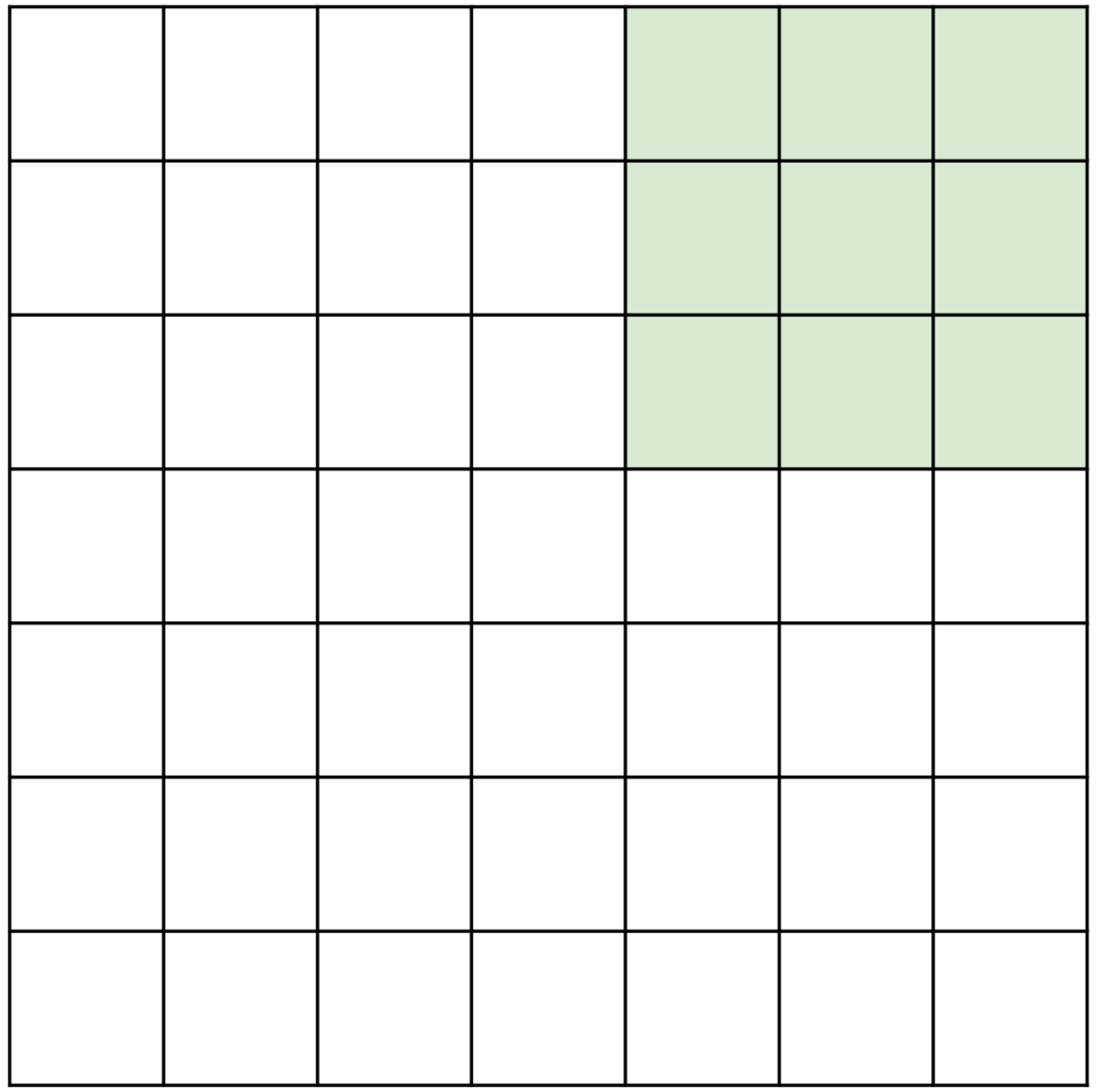


Input: 7x7

Filter: 3x3

Stride: 2

# Strided Convolution



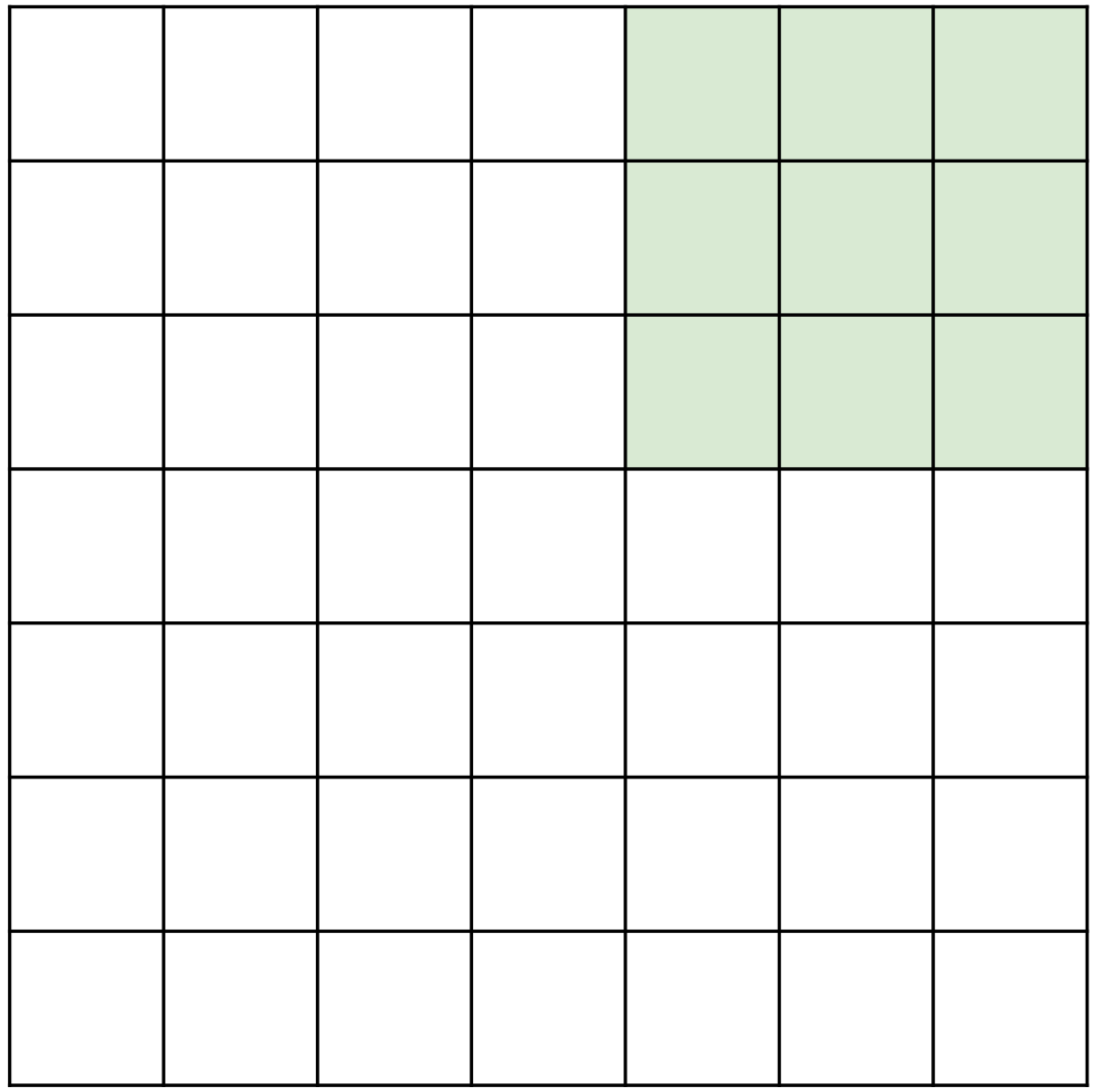
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

# Strided Convolution



Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

In general:

Input: W

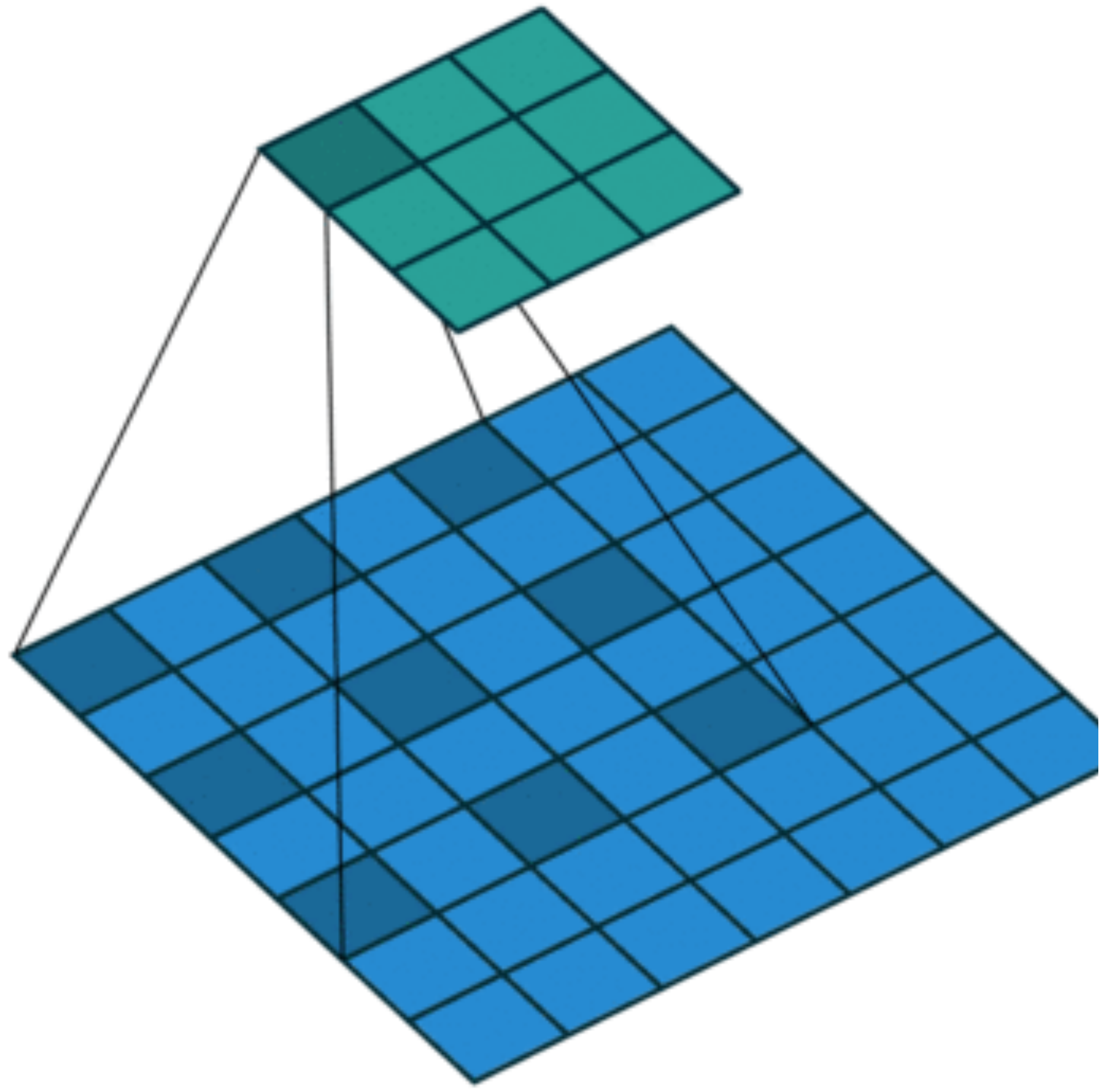
Filter: K

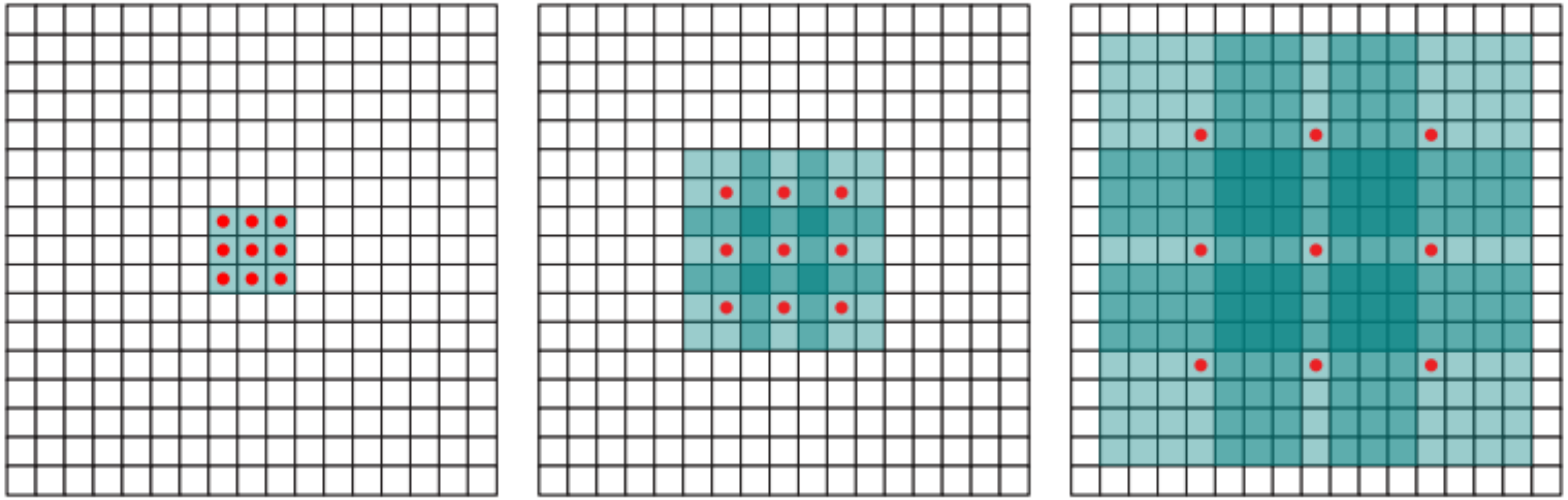
Padding: P

Stride: S

Output: (W – K + 2P) / S + 1

# Dilated Convolution





(a) (b) (c)

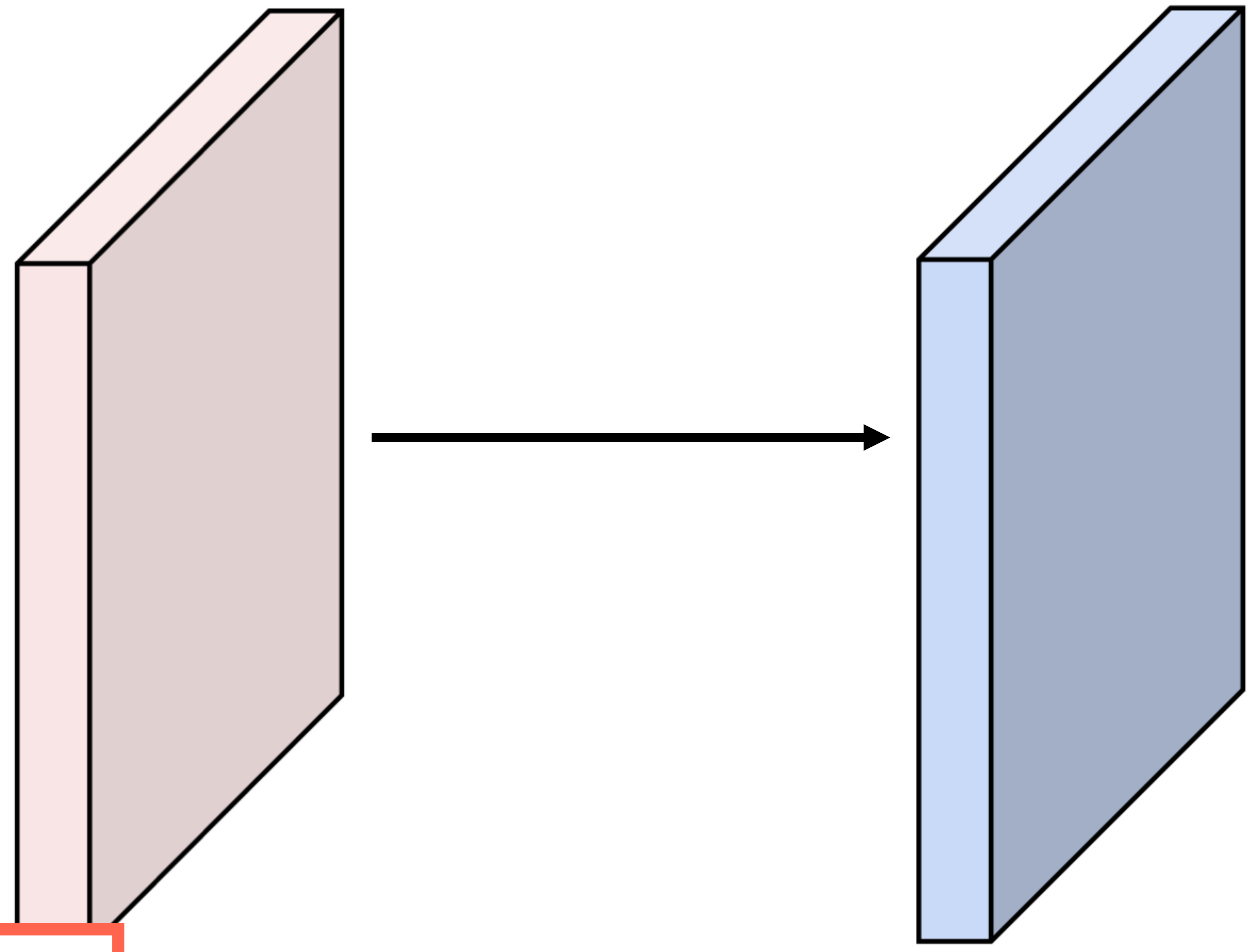
Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) $F_1$ is produced from $F_0$ by a 1-dilated convolution; each element in $F_1$ has a receptive field of $3\times3$. (b) $F_2$ is produced from $F_1$ by a 2-dilated convolution; each element in $F_2$ has a receptive field of $7\times7$. (c) $F_3$ is produced from $F_2$ by a 4-dilated convolution; each element in $F_3$ has a receptive field of $15\times15$. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

# Convolution Example

Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

**Q:** What is the output volume size?



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

# Convolution Example

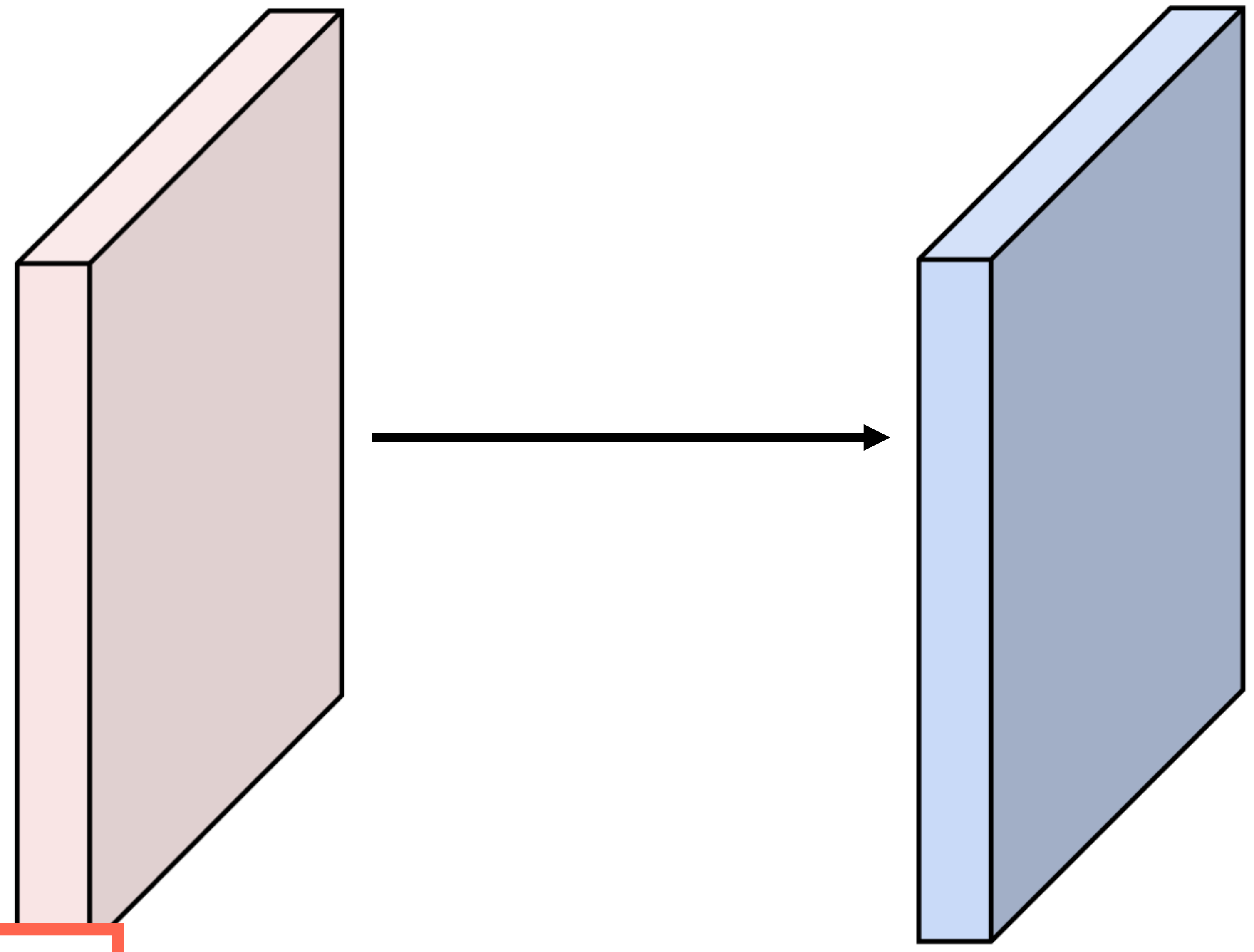
Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

**Q:** What is the output volume size?

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

So, 10 x 32 x 32 output



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

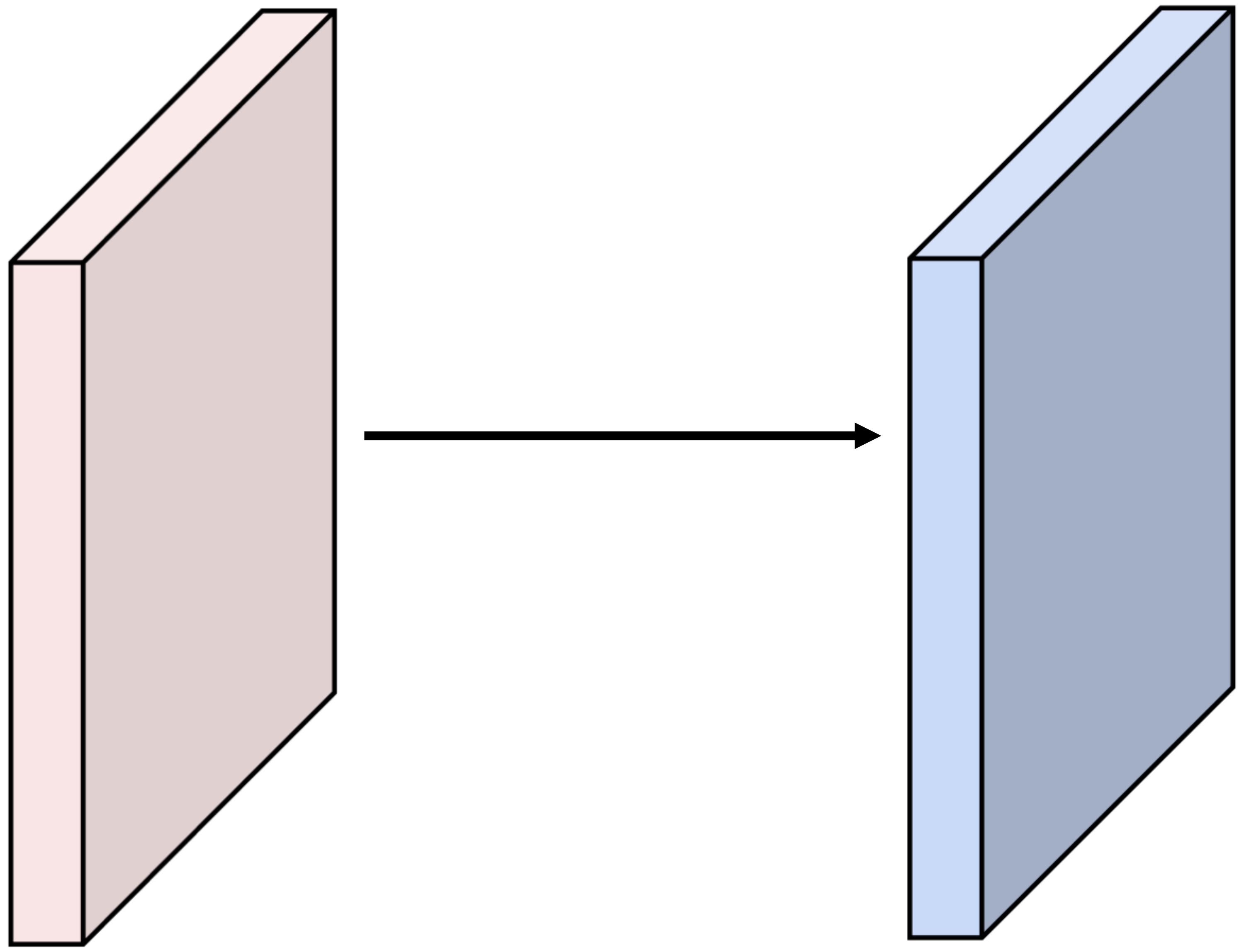
# Convolution Example

Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32

**Q:** What is the number of learnable parameters?



# Convolution Example

Input volume: 3 x 32 x 32

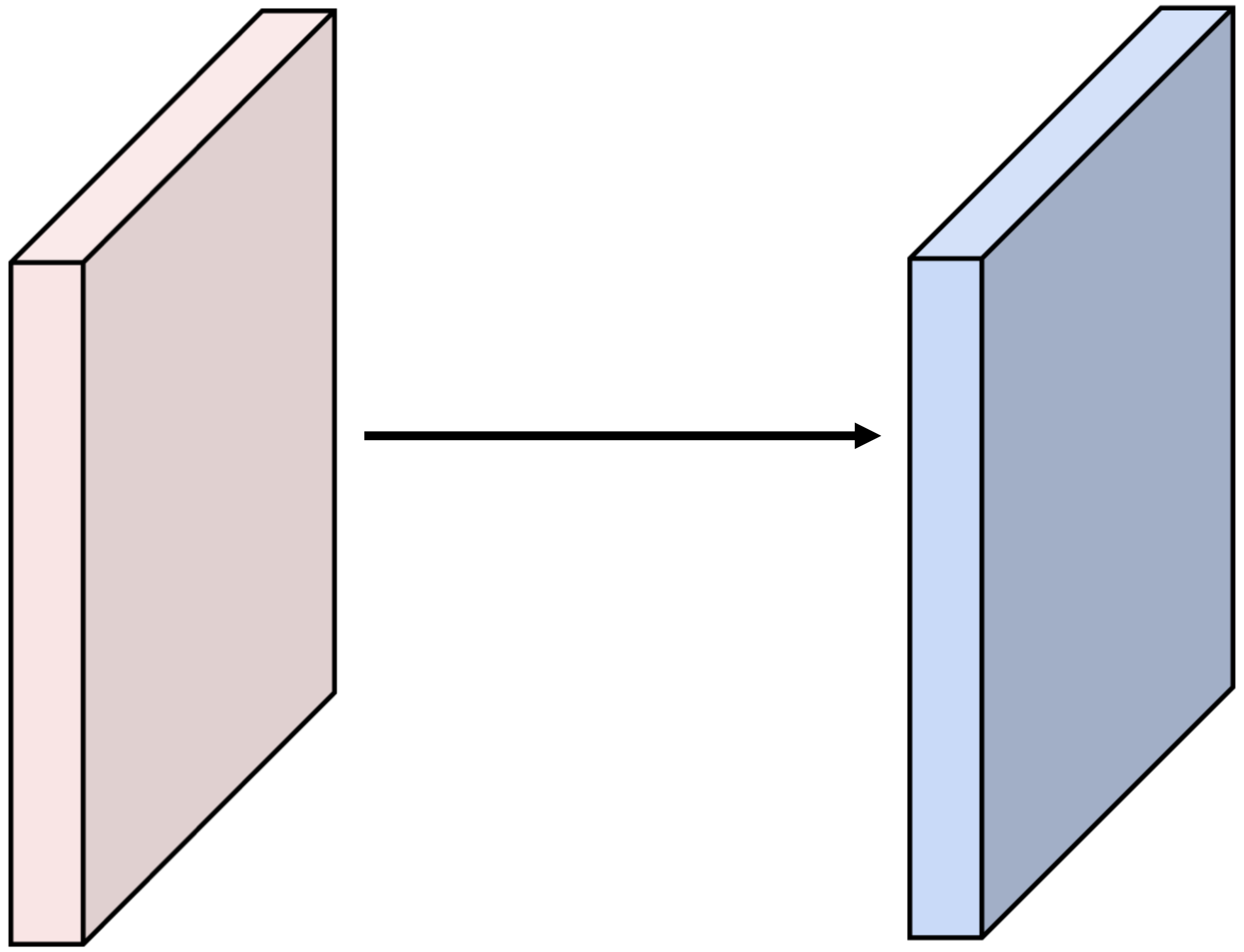
10 5x5 filters with stride 1, pad 2

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

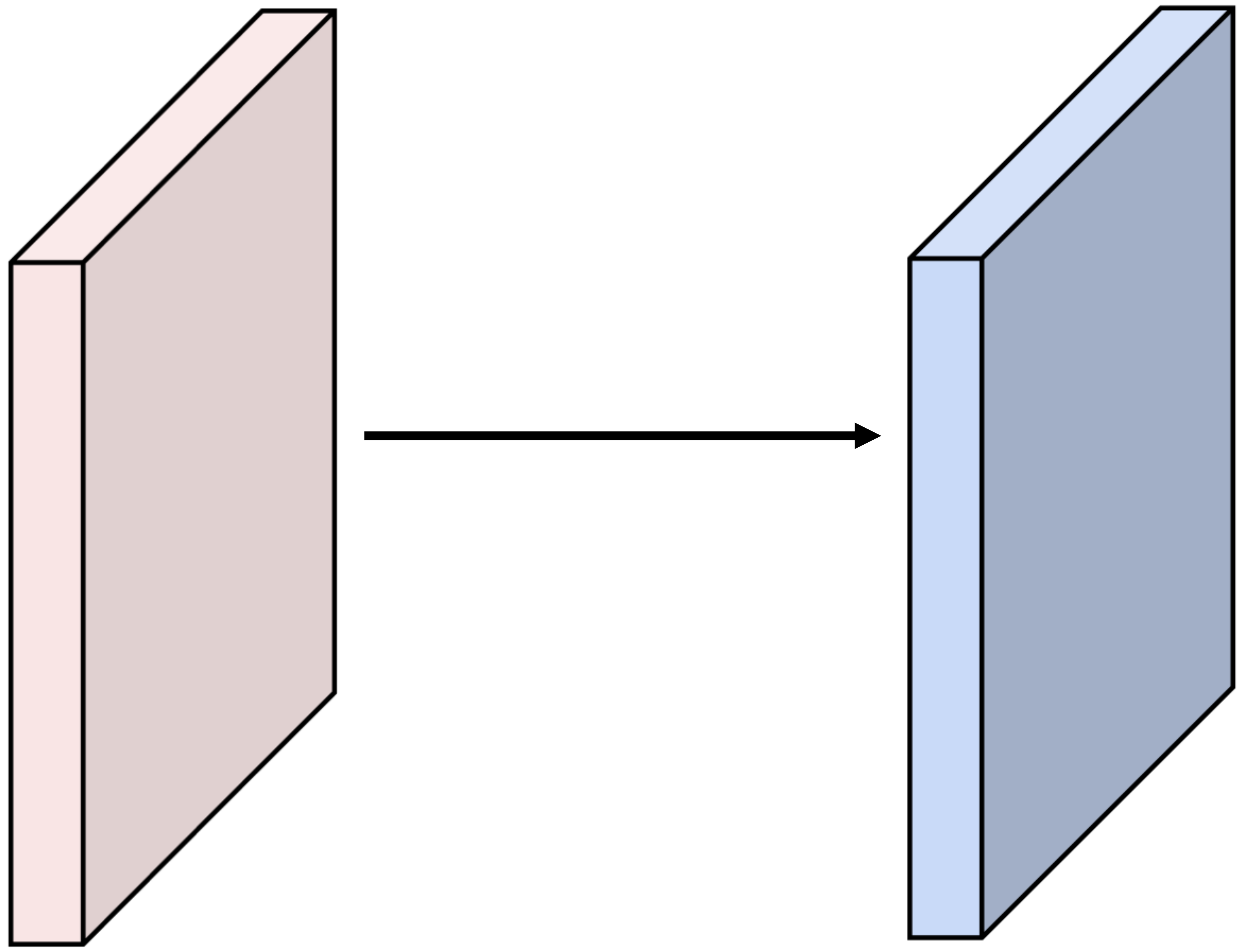


# Convolution Example

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?

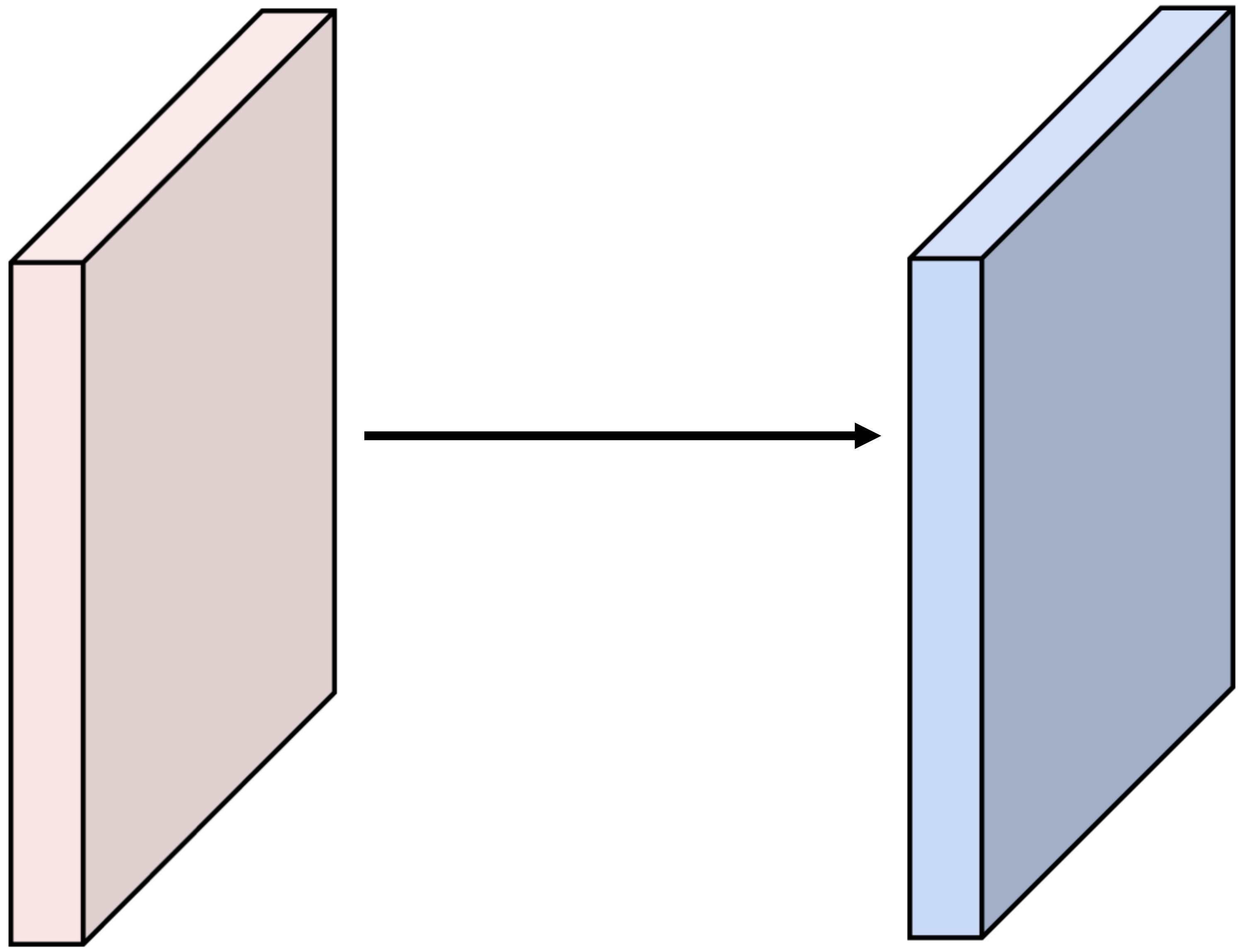


# Convolution Example

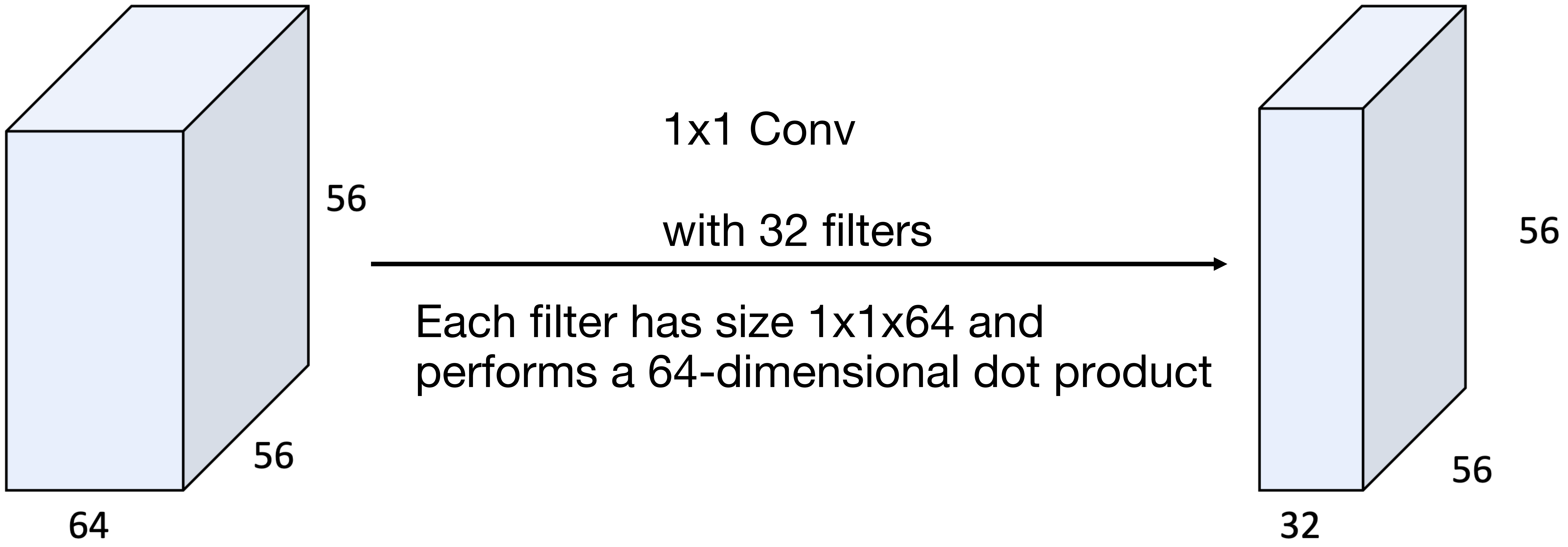
Input volume: 3 x 32 x 32

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**

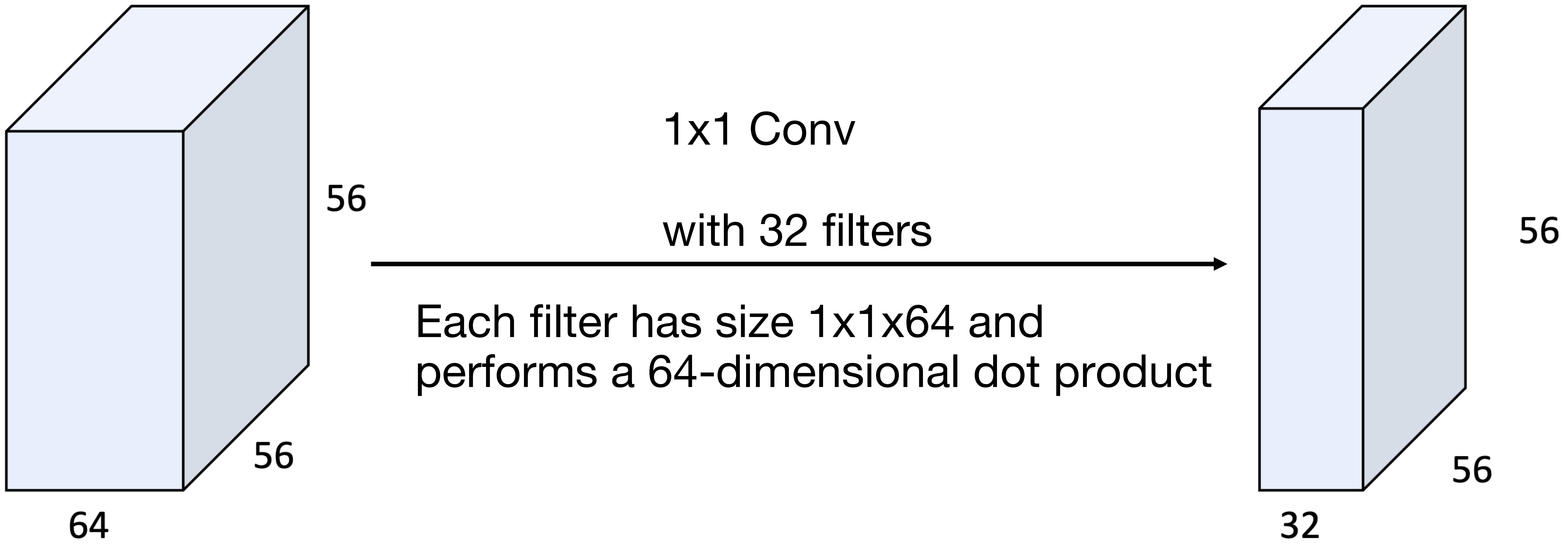


# Example: 1x1 Convolution




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

# Example: 1x1 Convolution




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

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

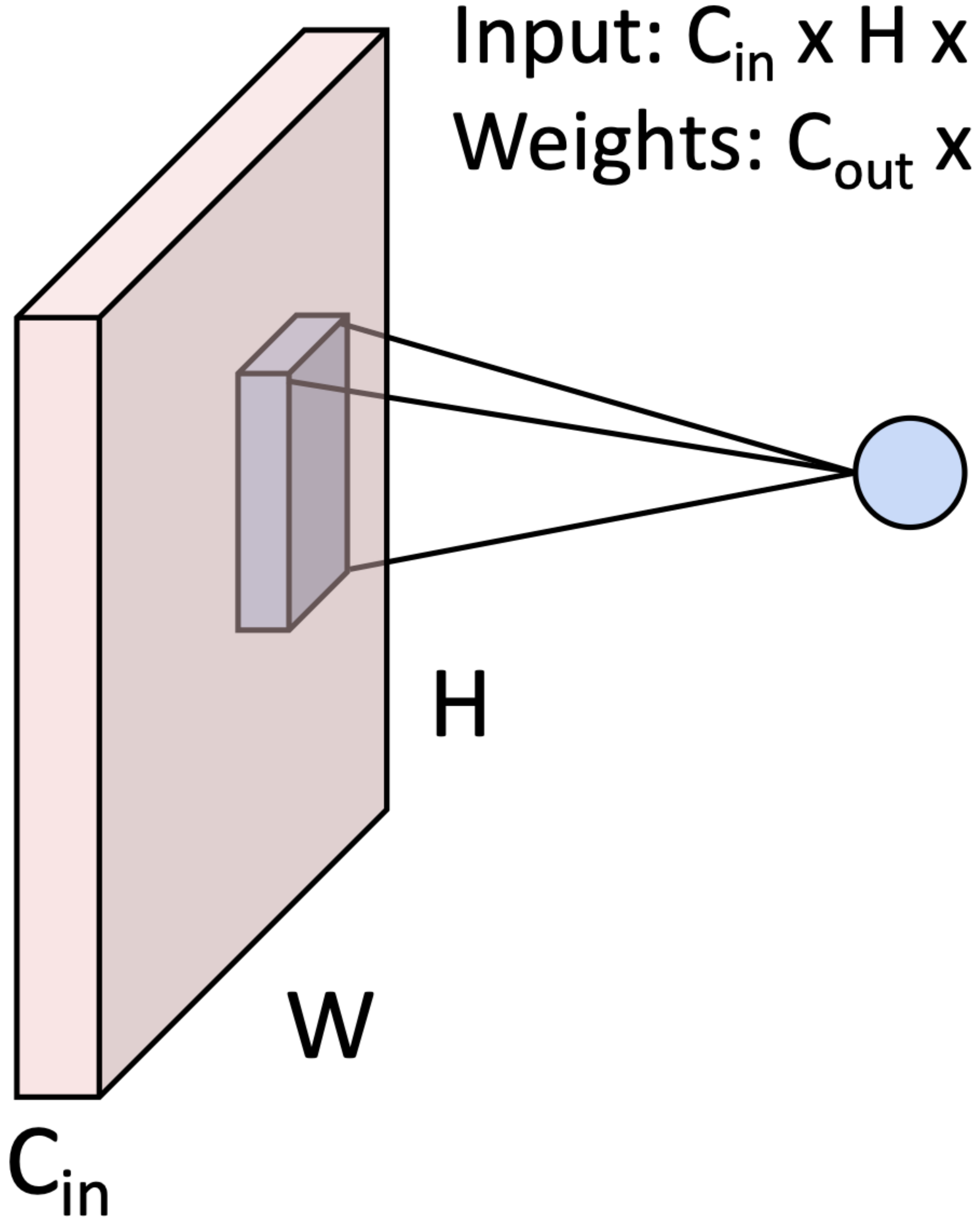
Common settings:
$K_H = K_W$ (Small square filters)
P = (K – 1) / 2 ("Same" padding)
$C_{in}$, $C_{out}$ = 32, 64, 128, 256 (powers of 2)
K = 3, P = 1, S = 1 (3x3 conv)
K = 5, P = 2, S = 1 (5x5 conv)
K = 1, P = 0, S = 1 (1x1 conv)
K = 3, P = 1, S = 2 (Downsample by 2)

# Other types of convolution

So far: 2D Convolution

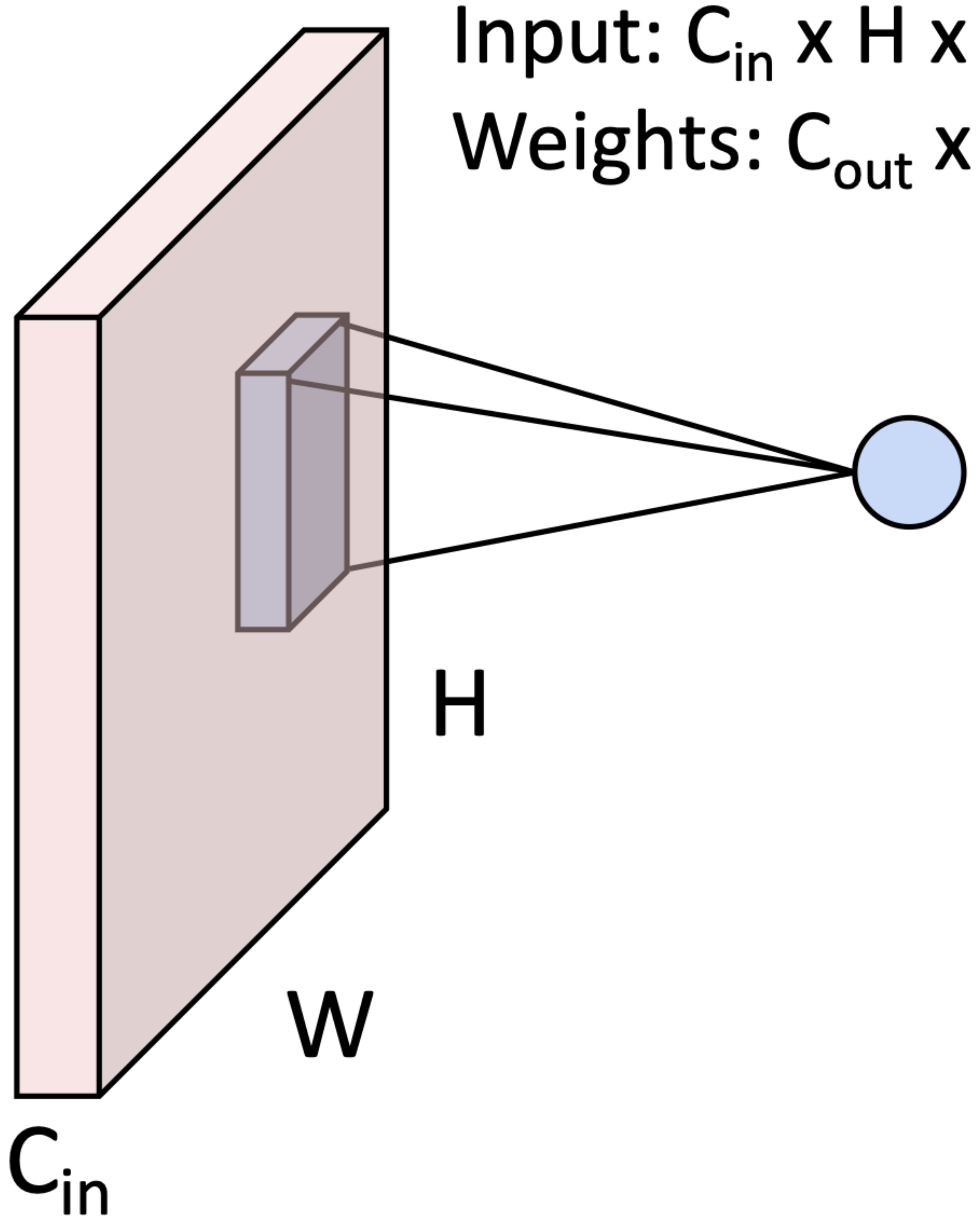
Input: $C_{in}$ x H x W

Weights: $C_{out}$ x $C_{in}$ x K x K



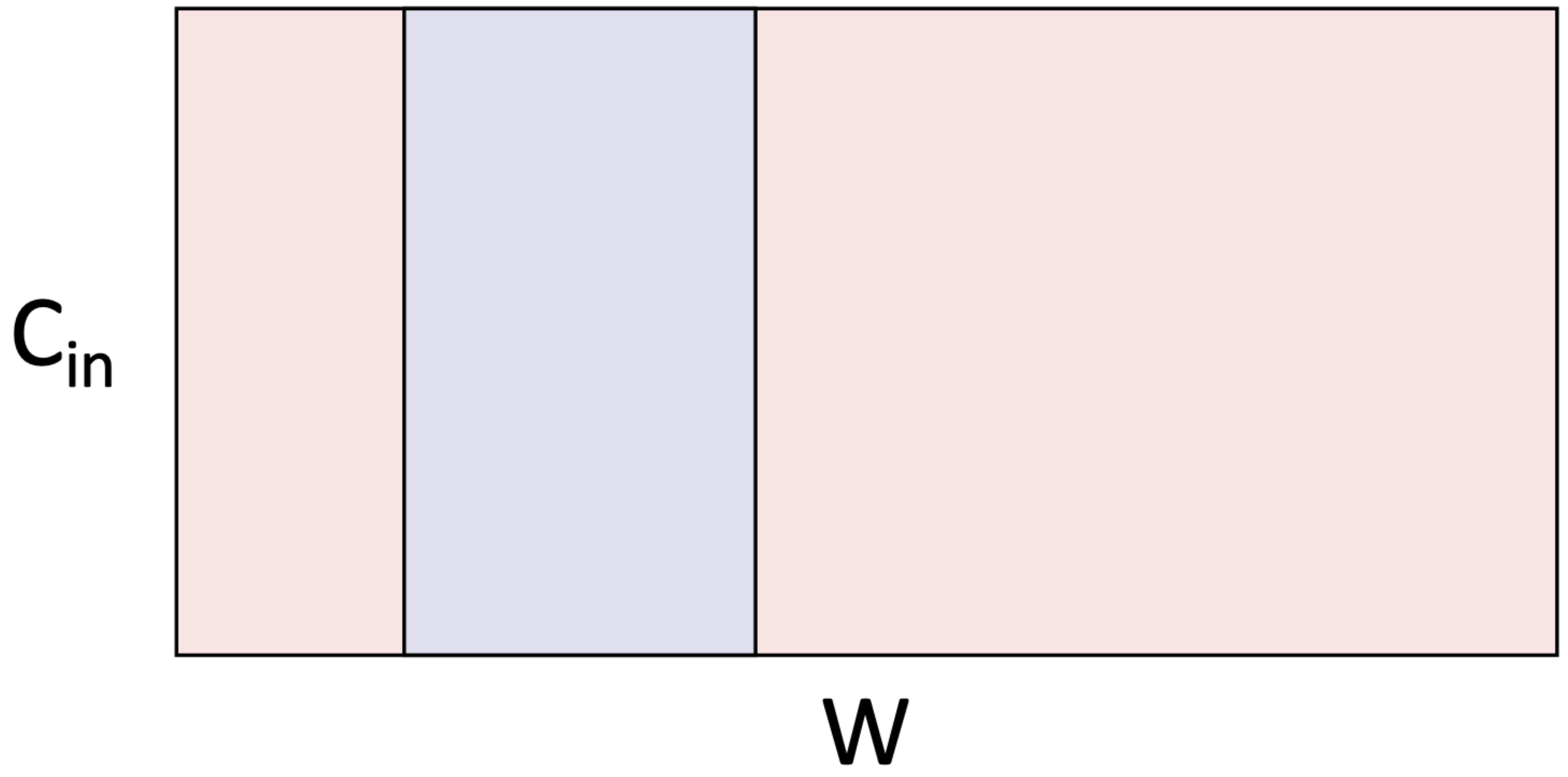
# Other types of convolution

## So far: 2D Convolution




## 1D Convolution

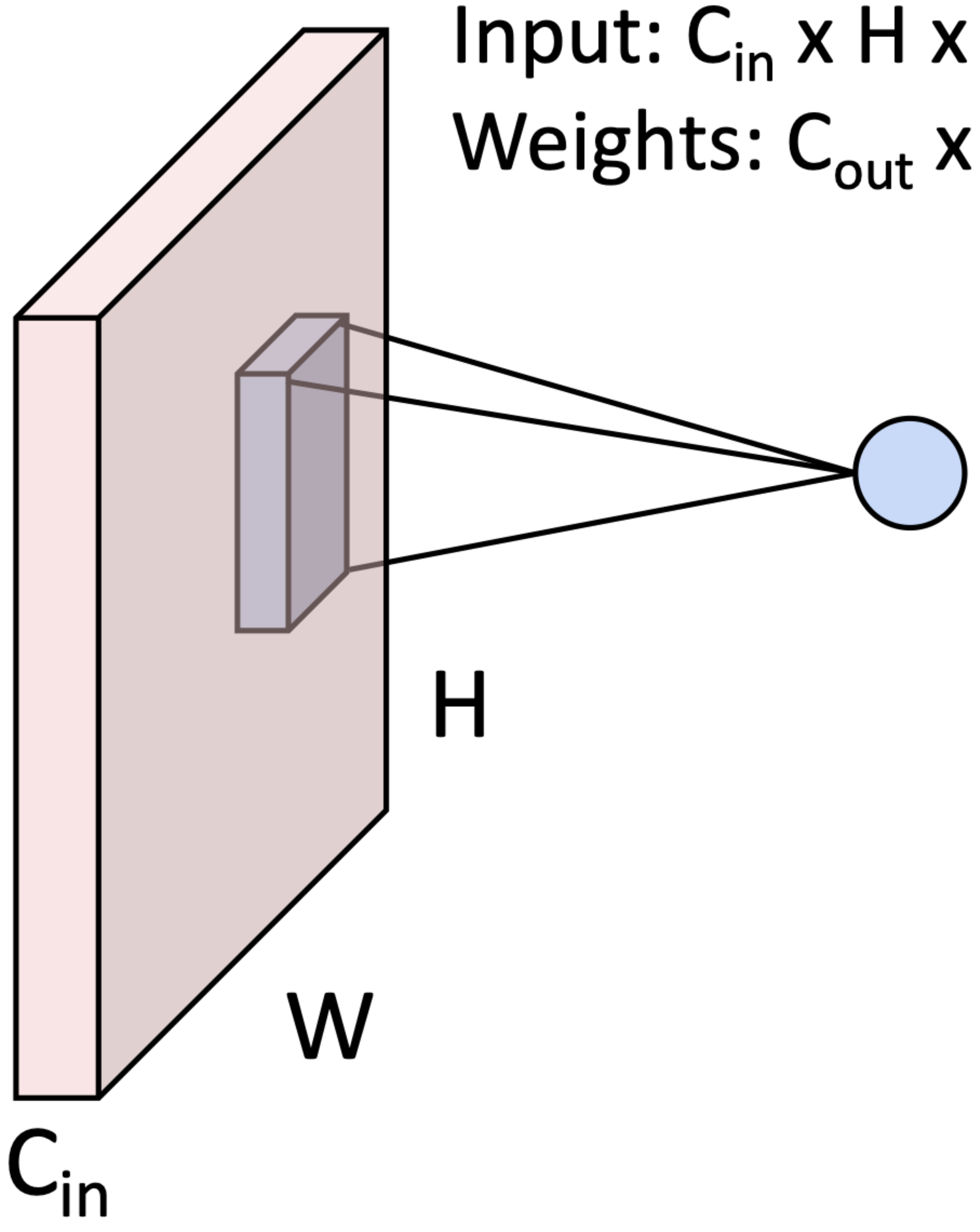
Input: $C_{in}$ x W
Weights: $C_{out}$ x $C_{in}$ x K



# Other types of convolution

## So far: 2D Convolution

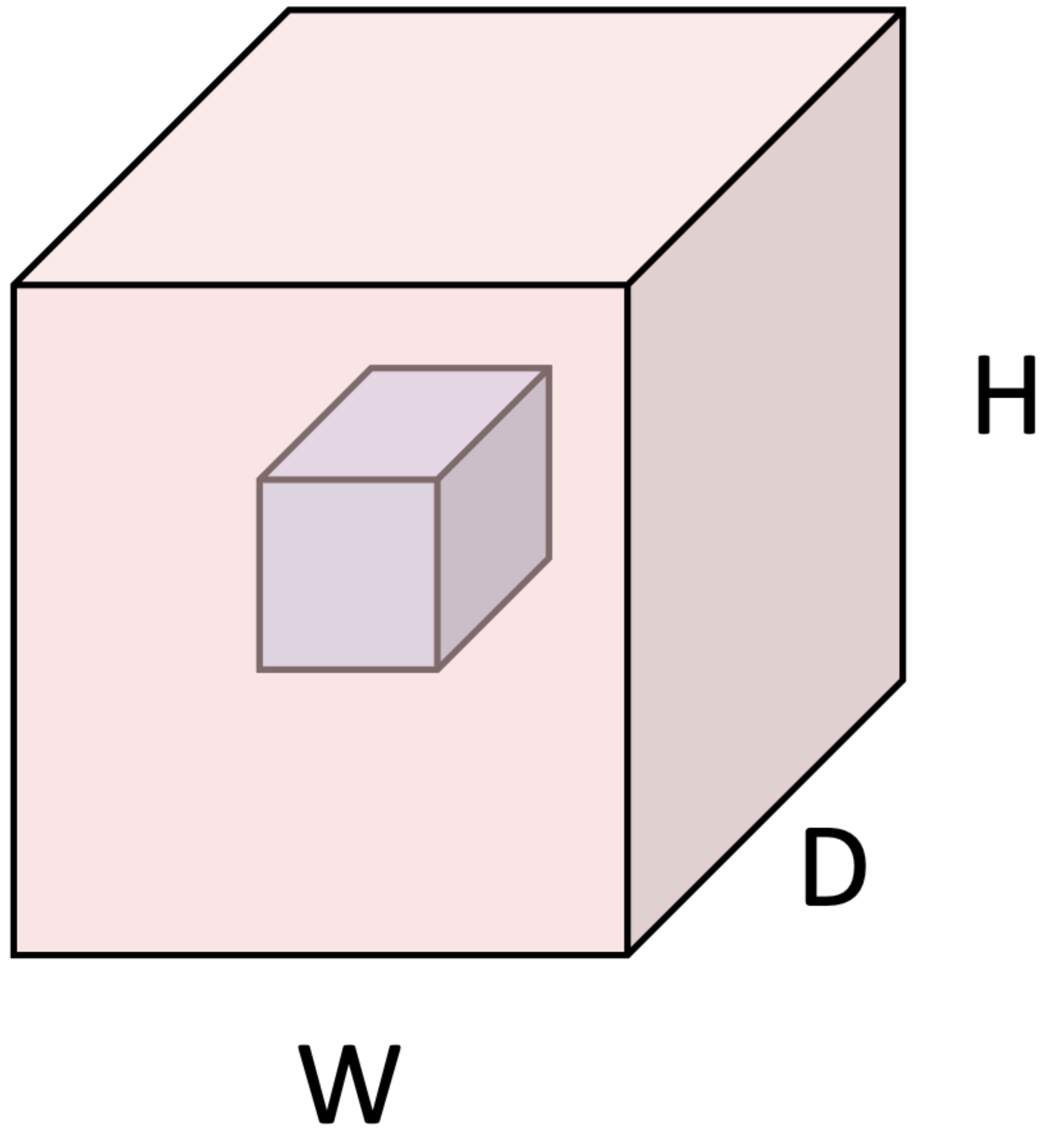
Input: $C_{in}$ x H x W
Weights: $C_{out}$ x $C_{in}$ x K x K




## 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 $(N, C_{\text{in}}, H, W)$ 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

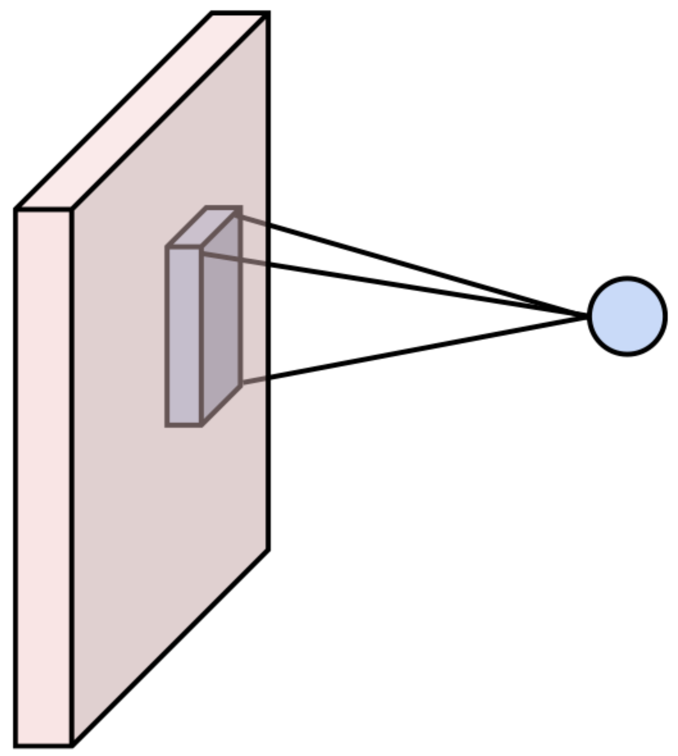
CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

## Conv3d

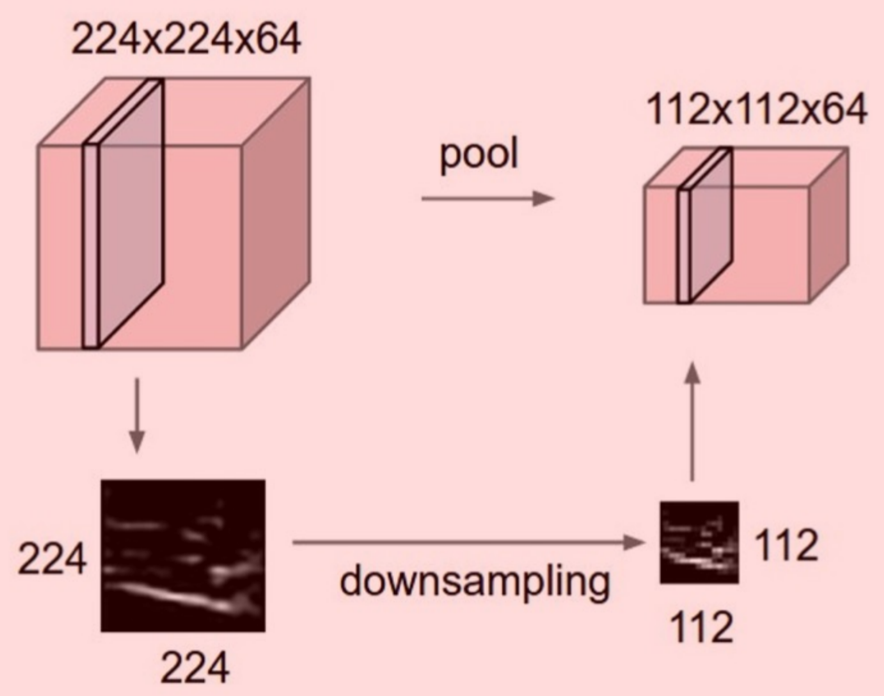
CLASS `torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

# Components of Convolutional Networks

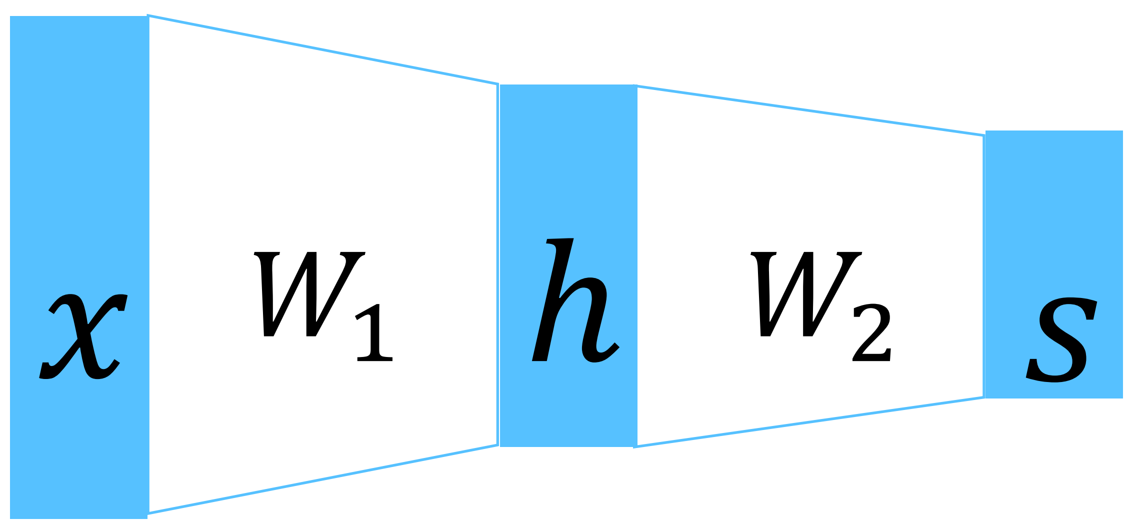
## Convolution Layers



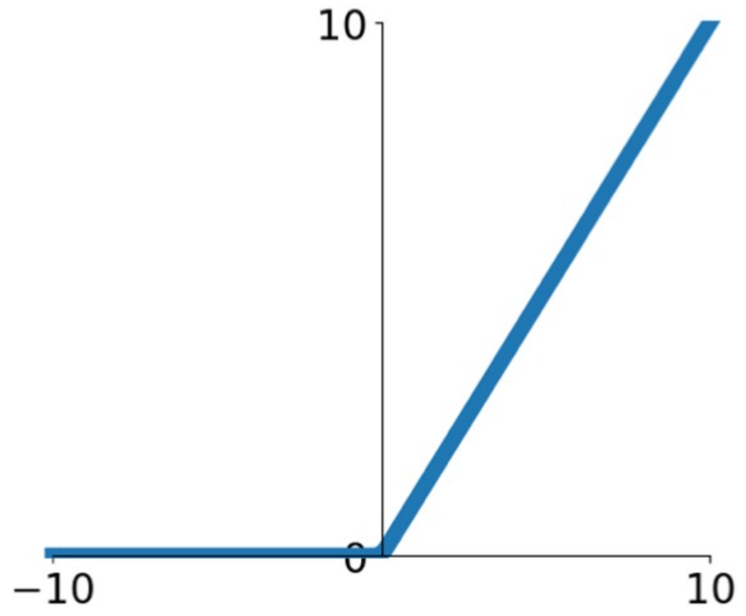
## Pooling Layers




## Fully-Connected Layers



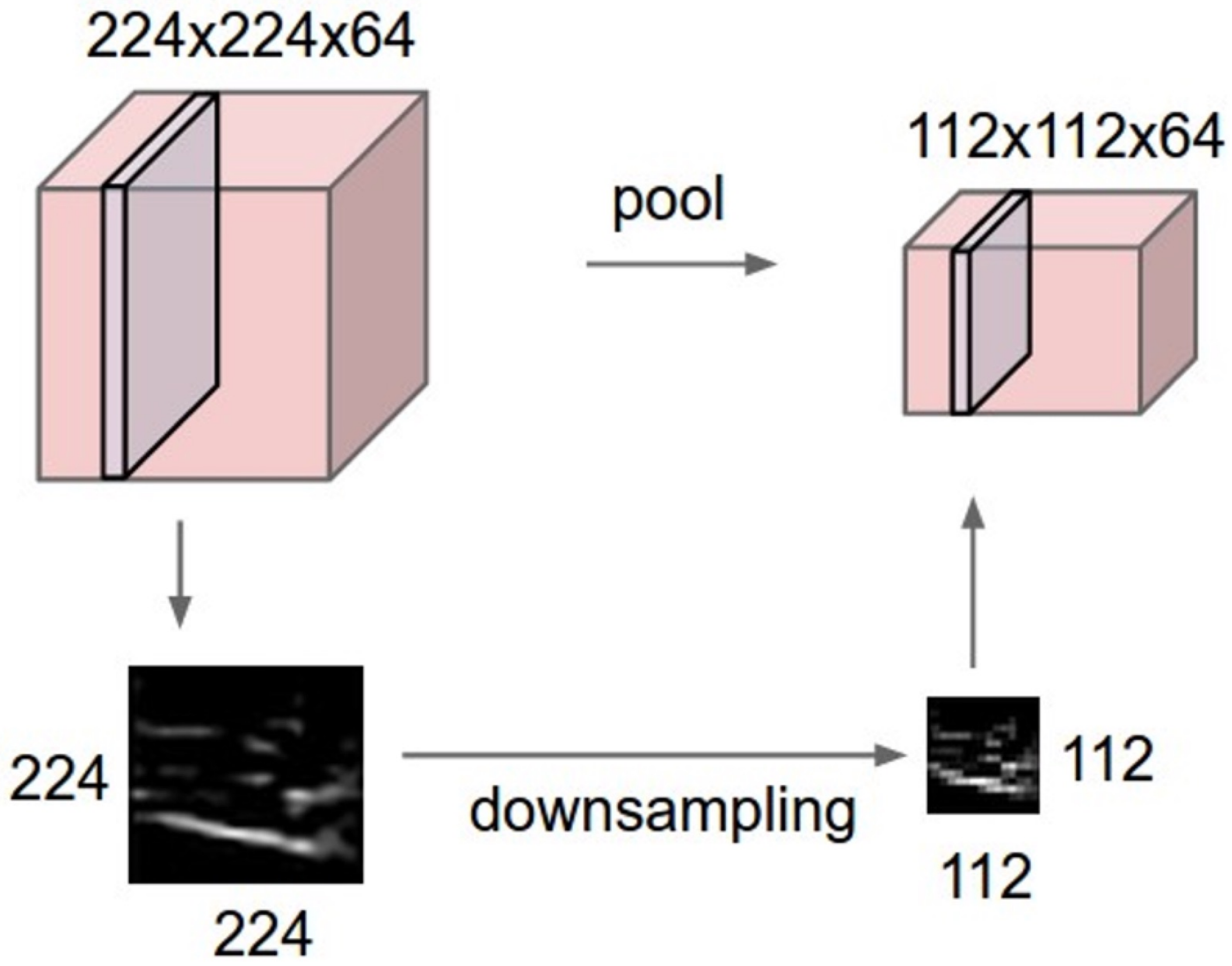

## Activation Function




## Normalization

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

# Pooling Layers: Another way to downsample



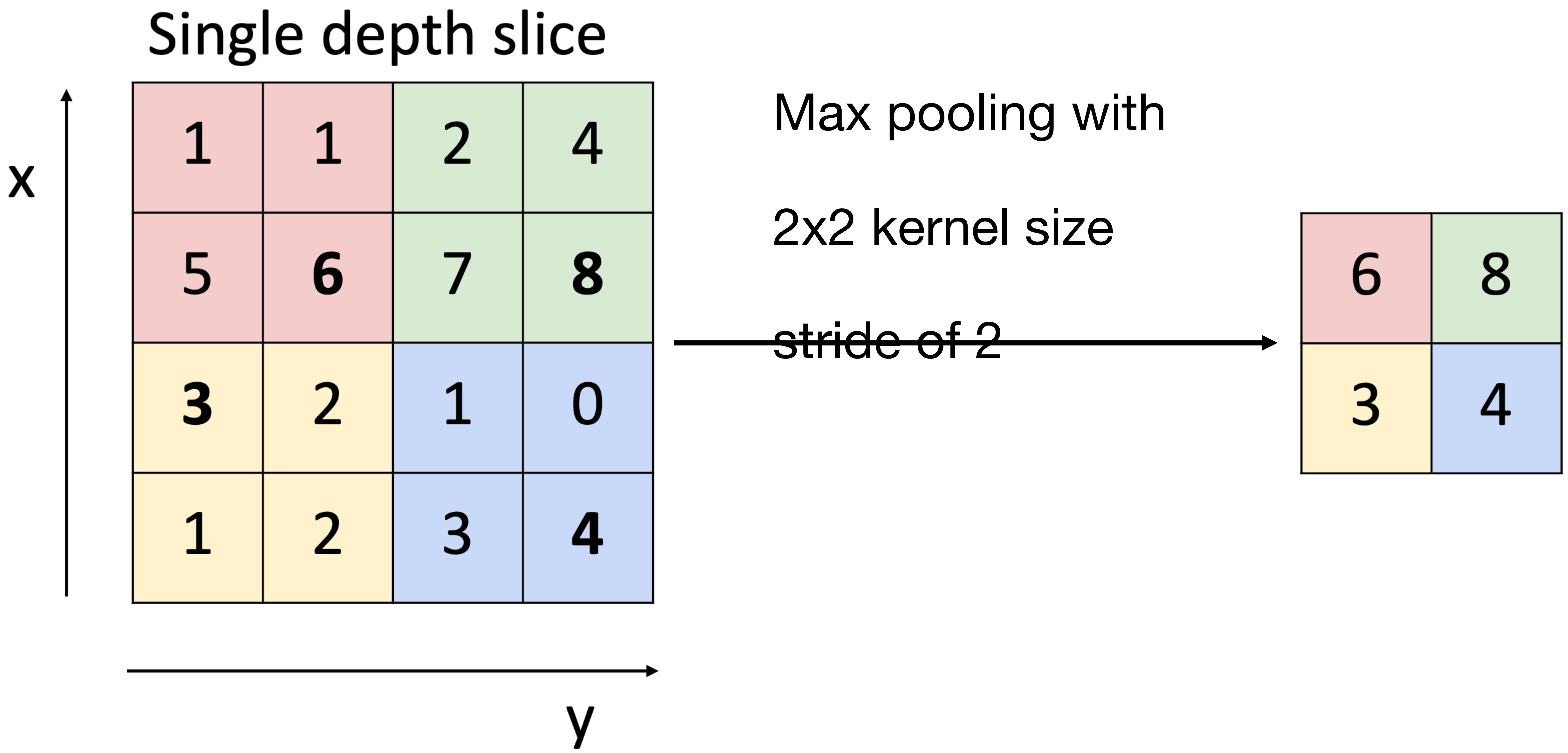

**Hyperparameters:**

Kernel size

Stride

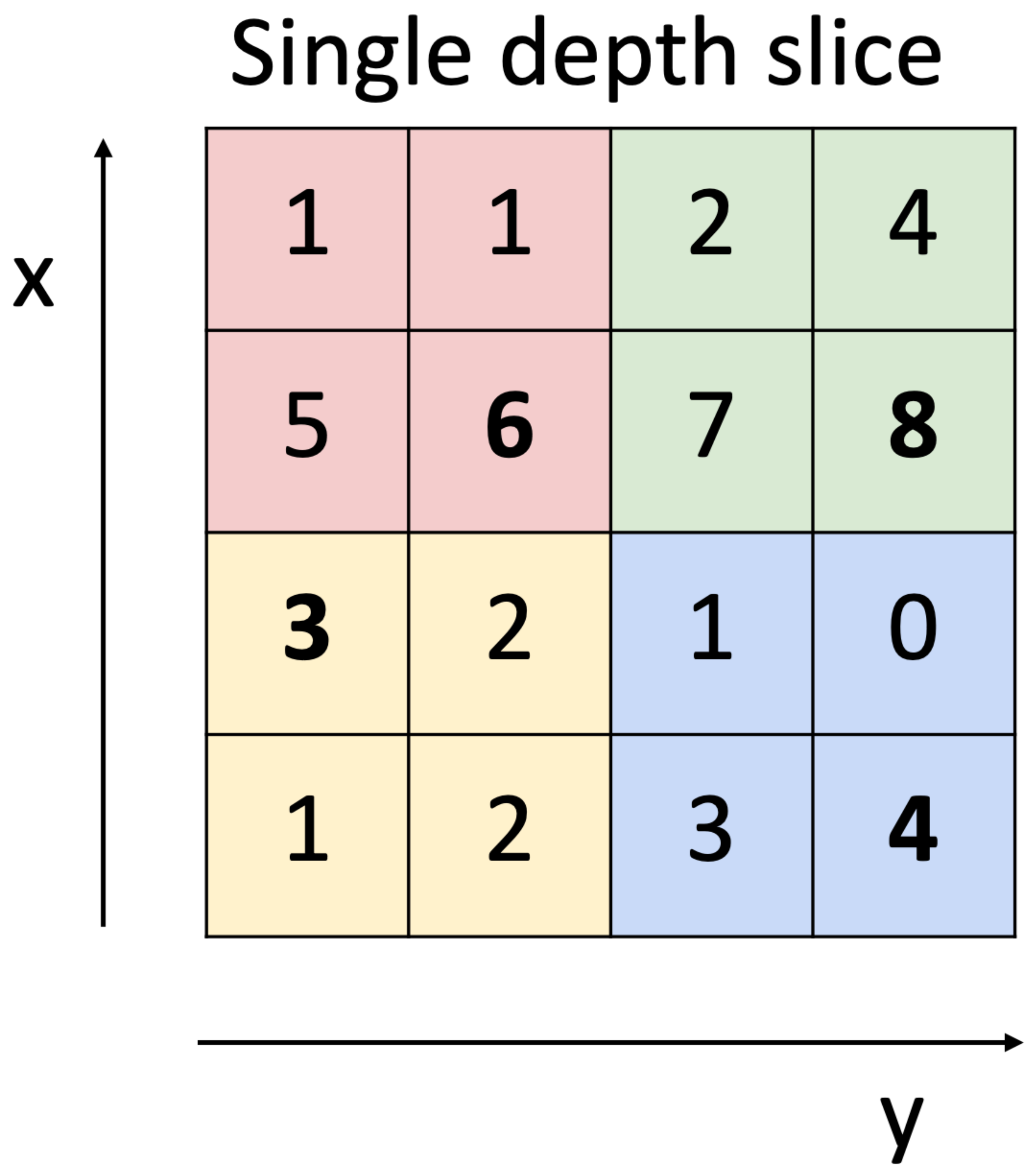
Pooling function

# 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

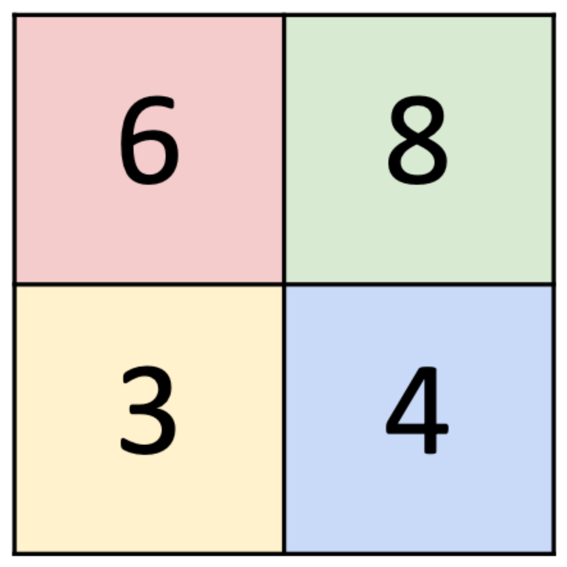
# Max Pooling




Max pooling with

2x2 kernel size

stride of 2




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)

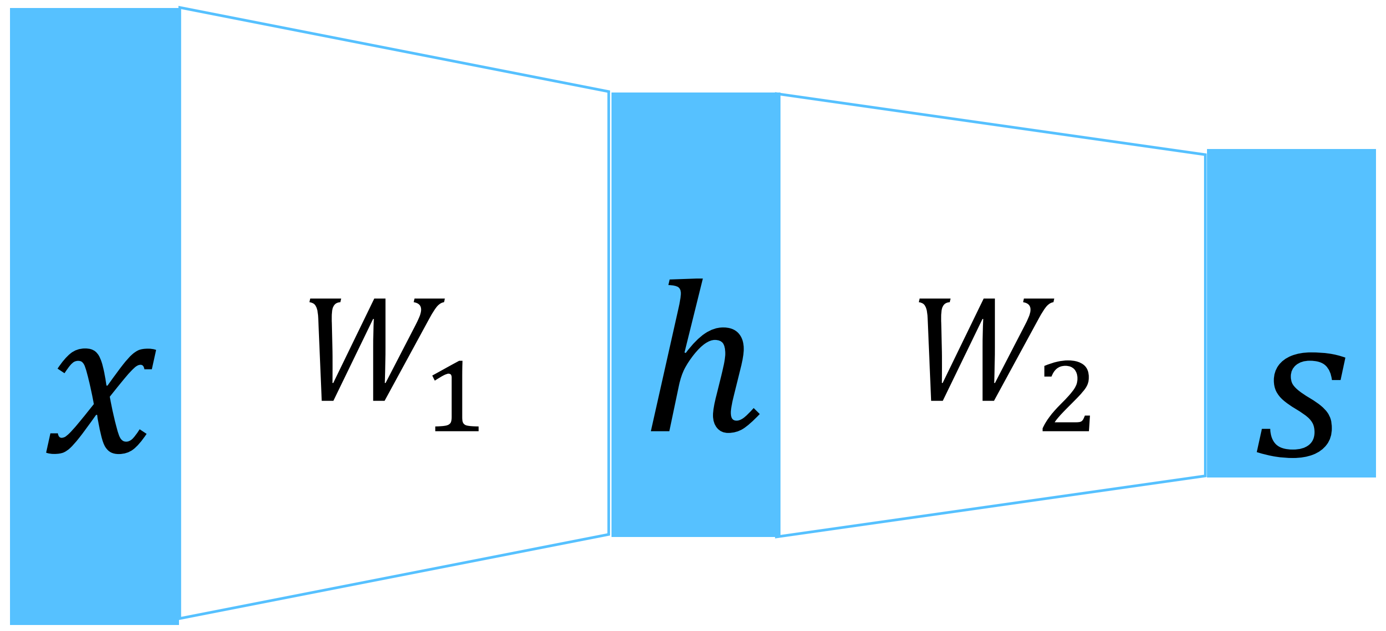
**Output**: C x H’ x W’ where
- H’ = (H – K) / S + 1
- W’ = (W – K) / S + 1

**Learnable parameters**: None!

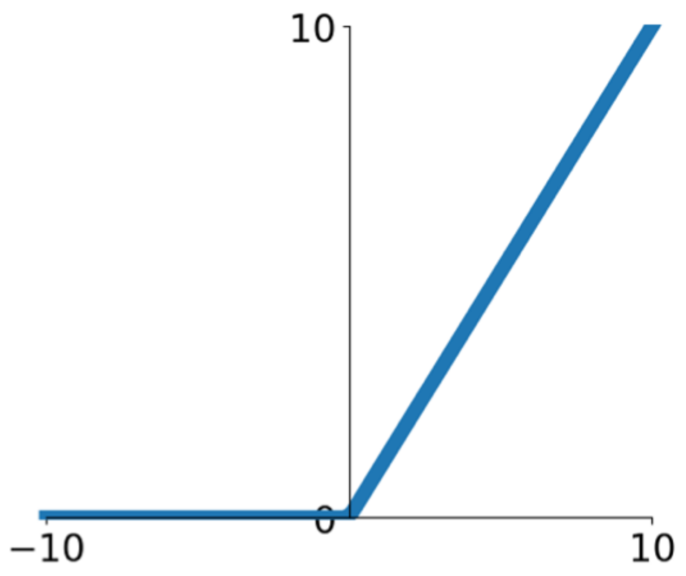
Common settings:
max, K = 2, S = 2
max, K = 3, S = 2 (AlexNet)

# Components of Convolutional Neural Networks

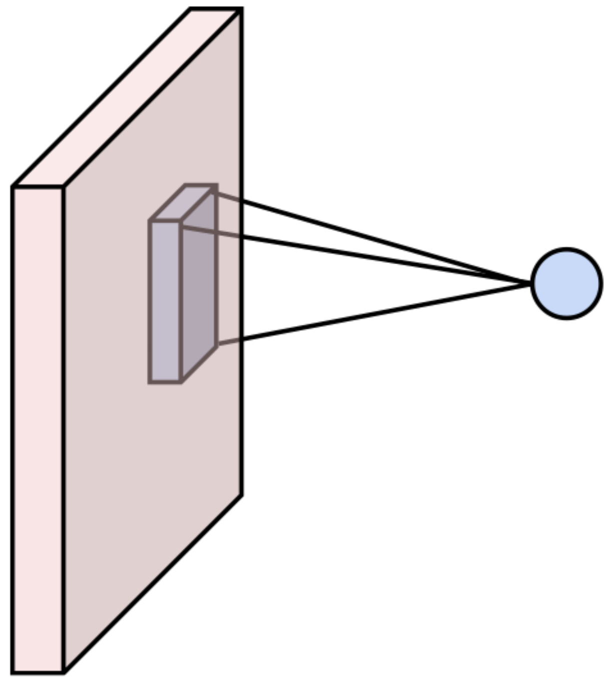
## Fully-Connected Layers



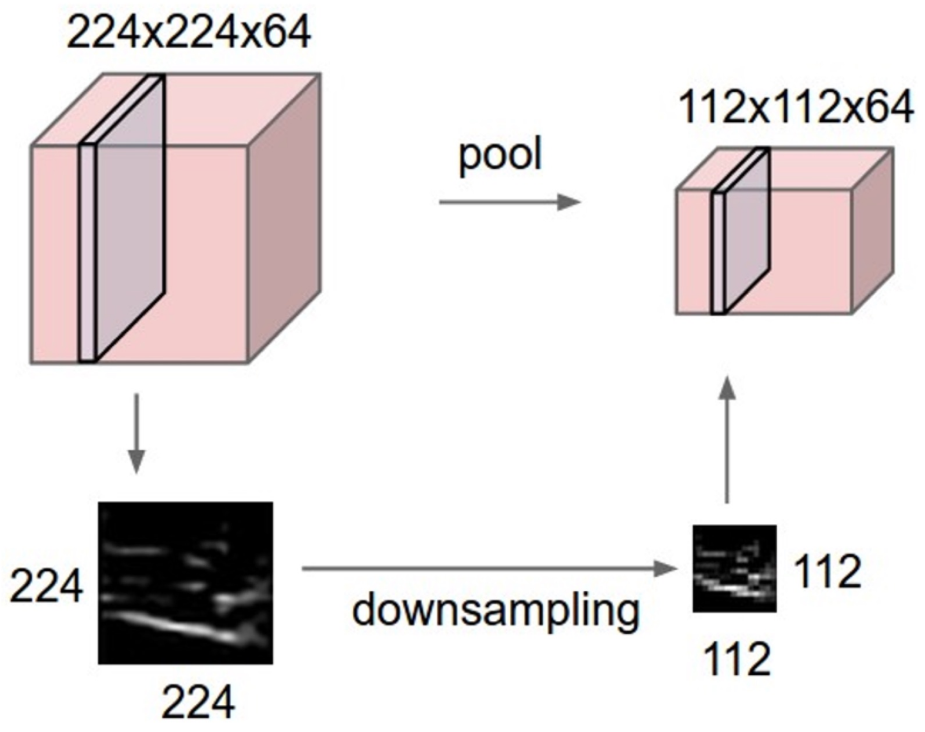

## Activation Functions




## Convolution Layers



## Pooling Layers



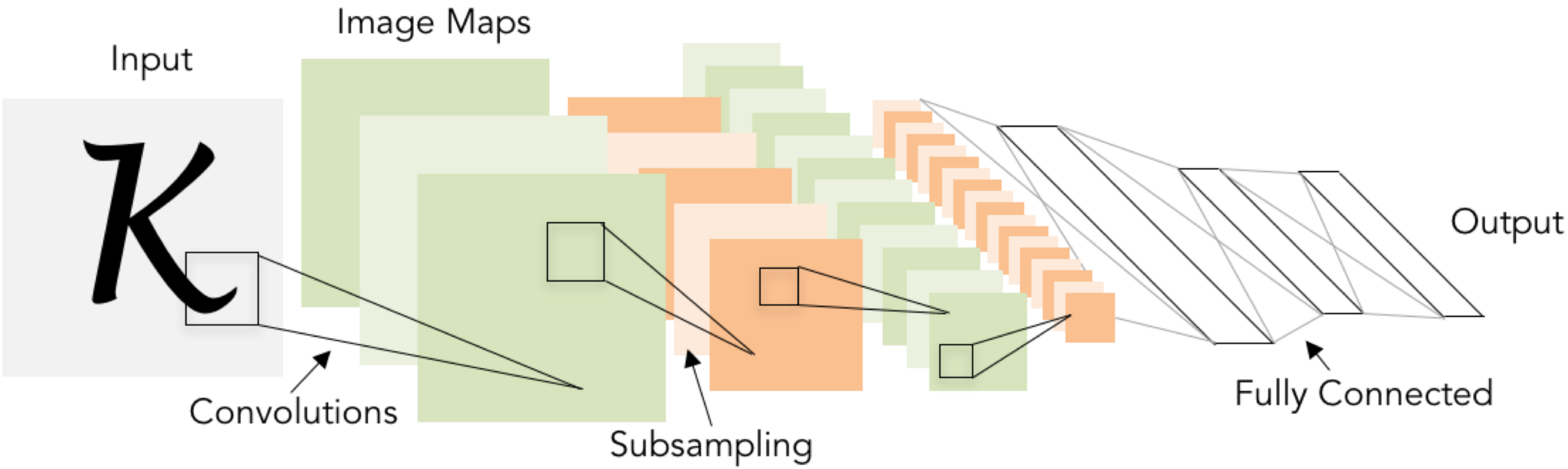

## Normalization

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

# Convolutional Neural Networks

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

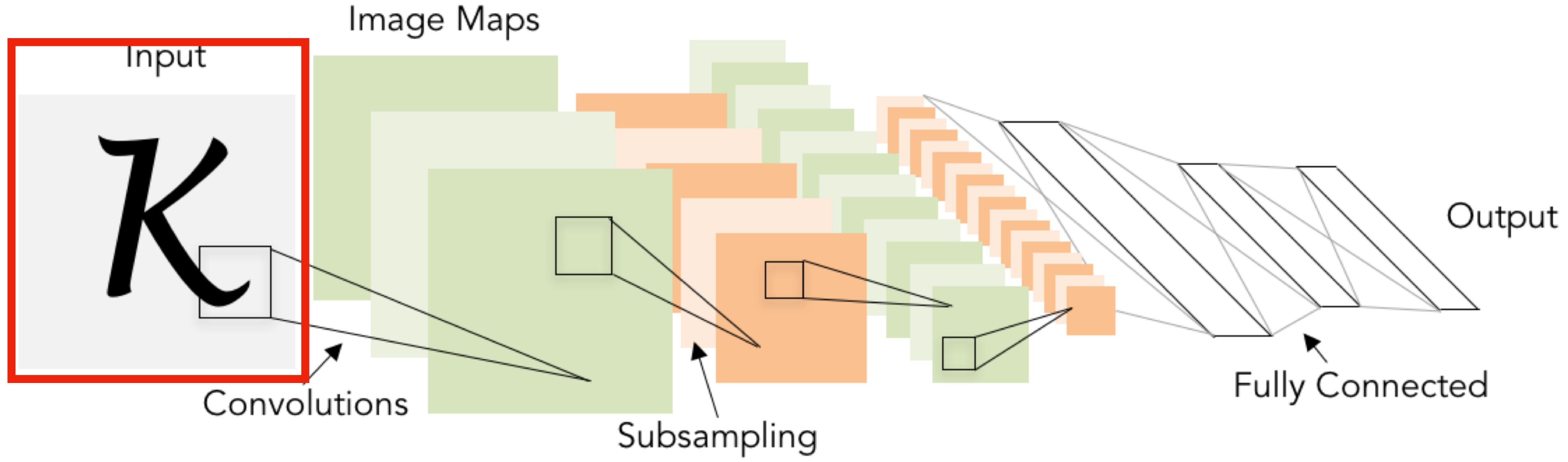
Example: LeNet-5




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

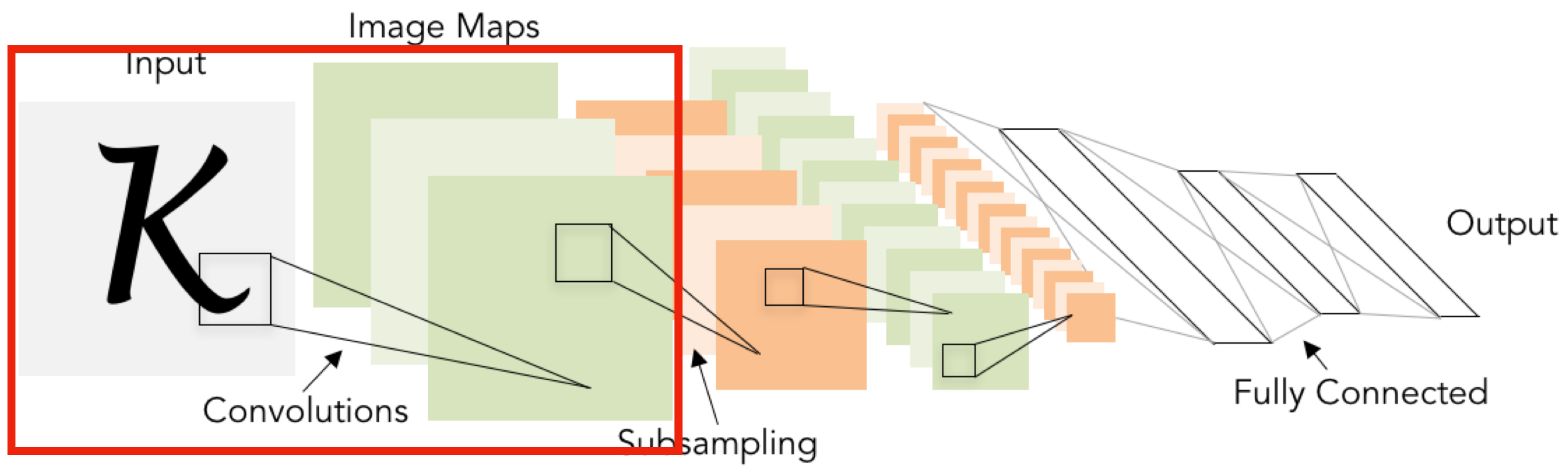
# Example: LeNet-5

| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |



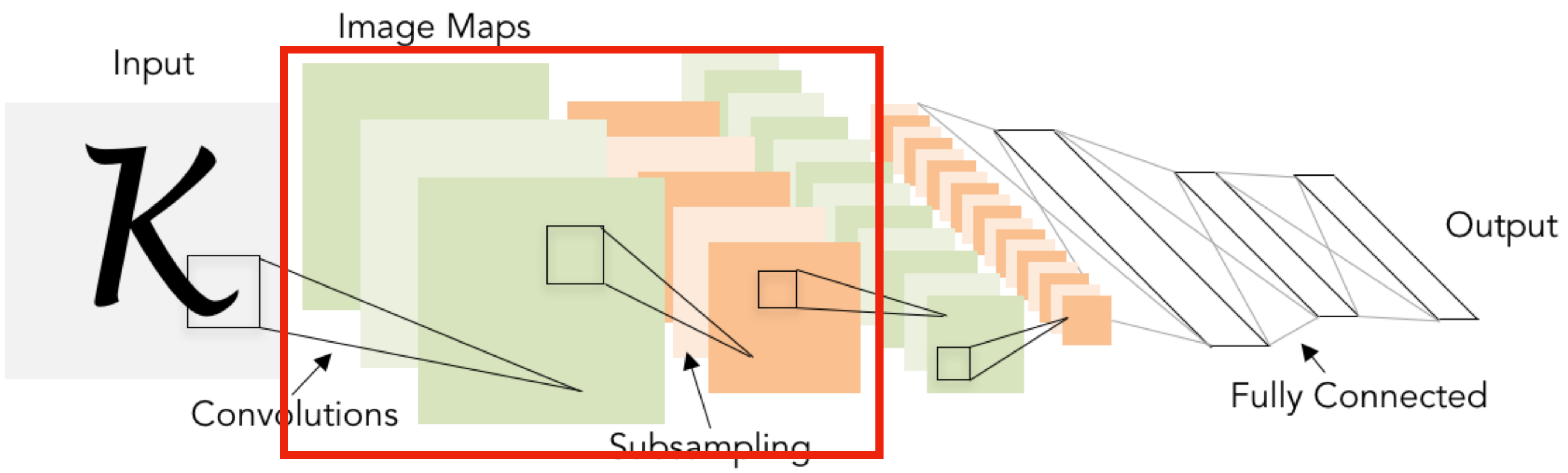
# Example: LeNet-5

| 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 | |



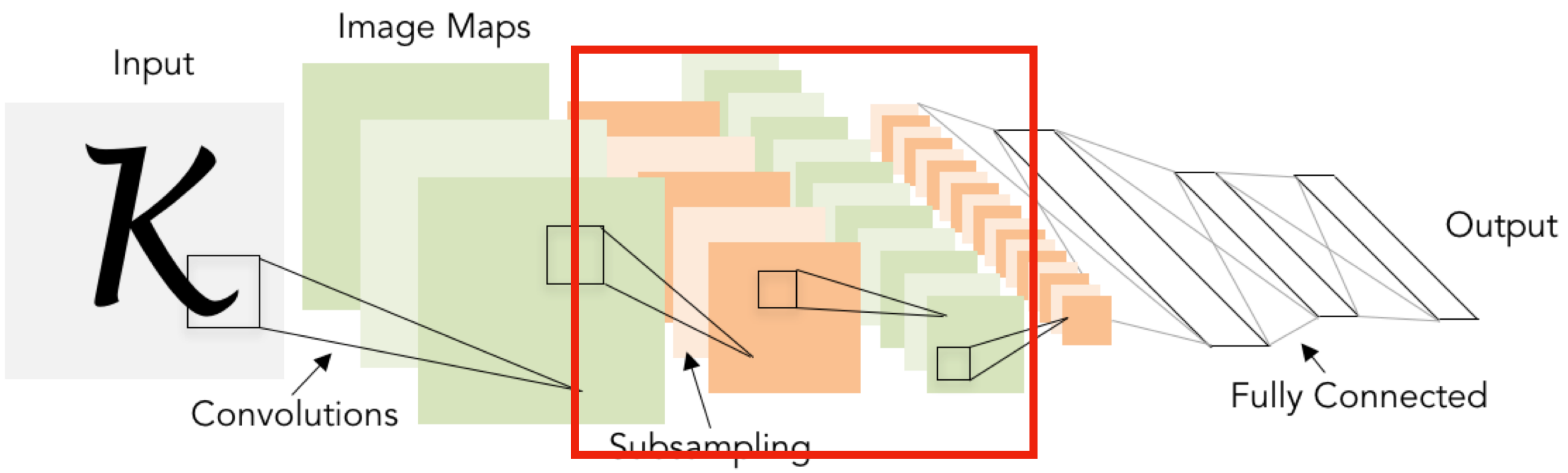
# Example: LeNet-5

| 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 | |



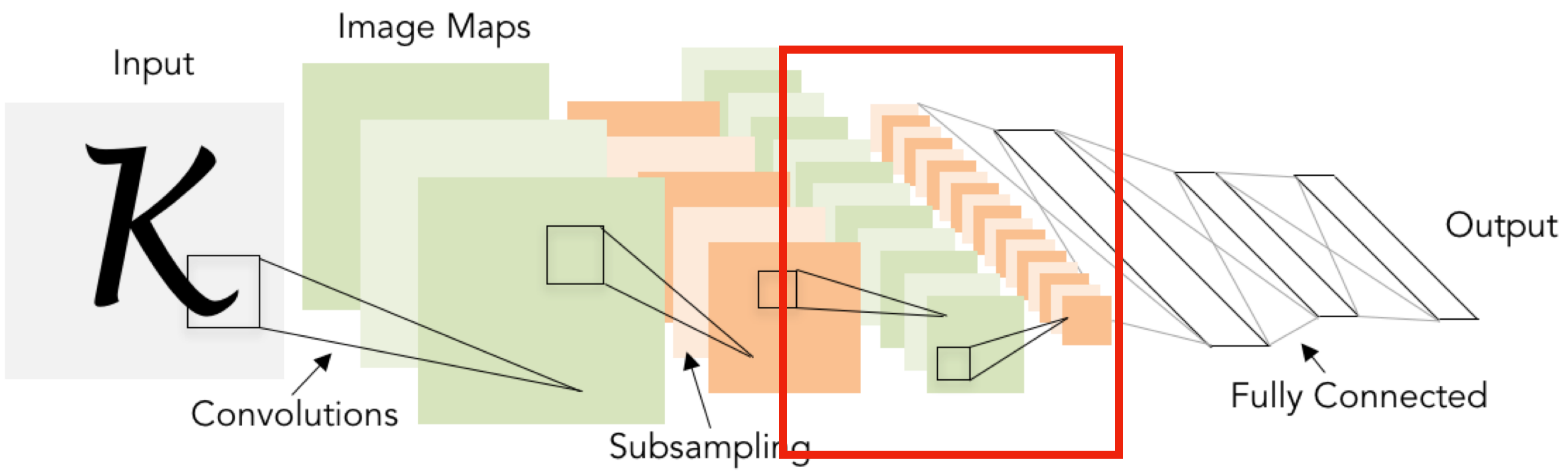
# Example: LeNet-5

| 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 | |



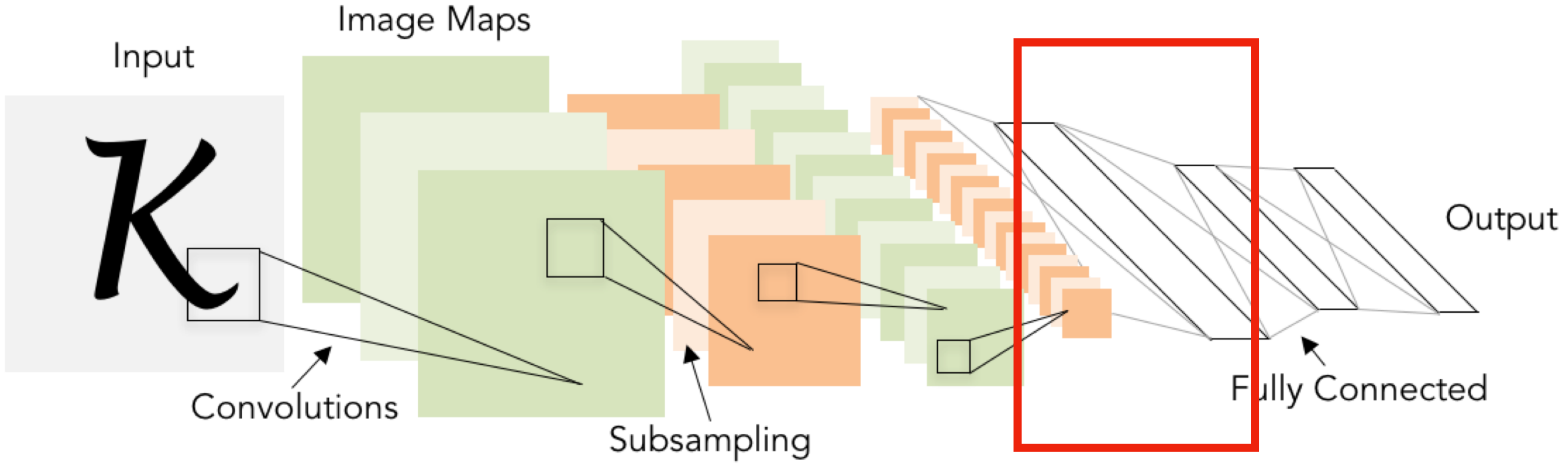
# Example: LeNet-5

| 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 | |



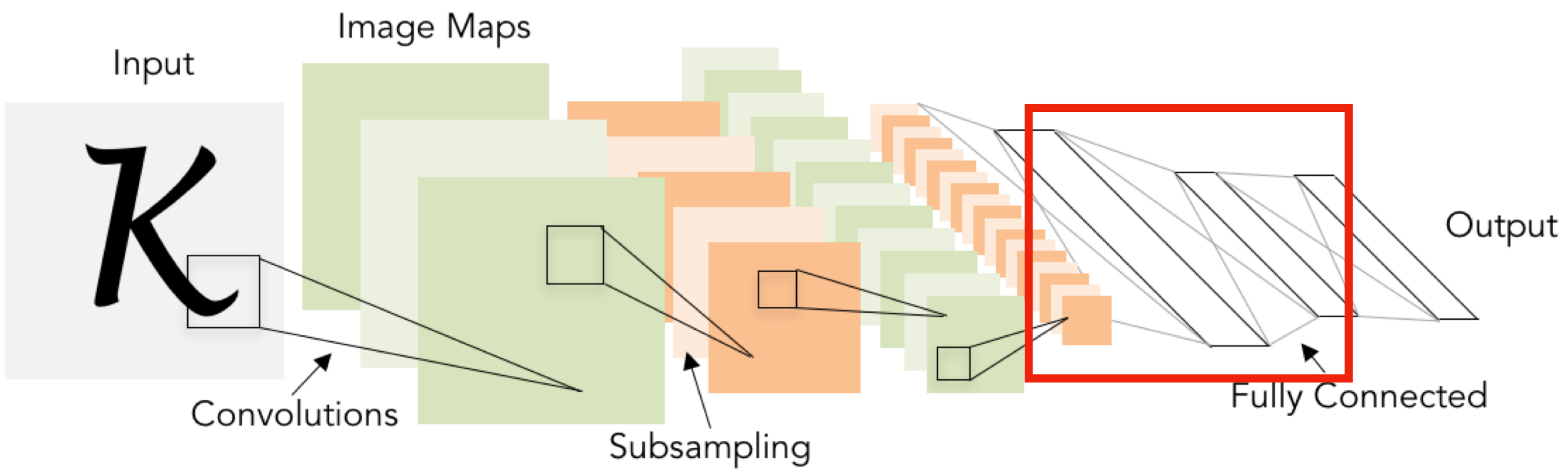
# Example: LeNet-5

| 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 | |



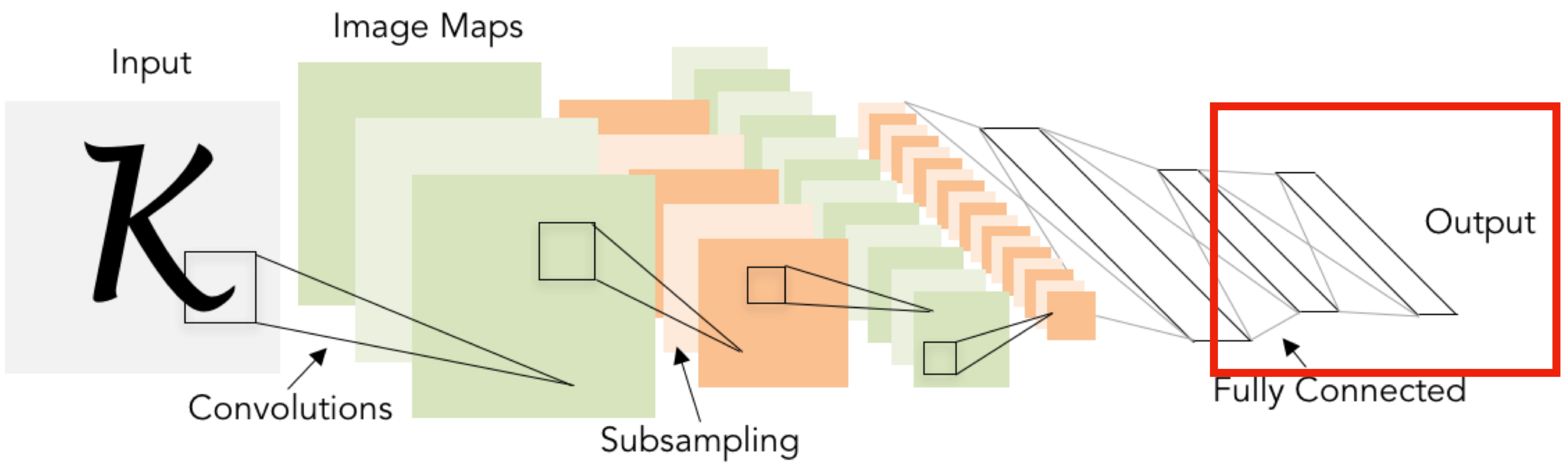
# Example: LeNet-5

| 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 | |



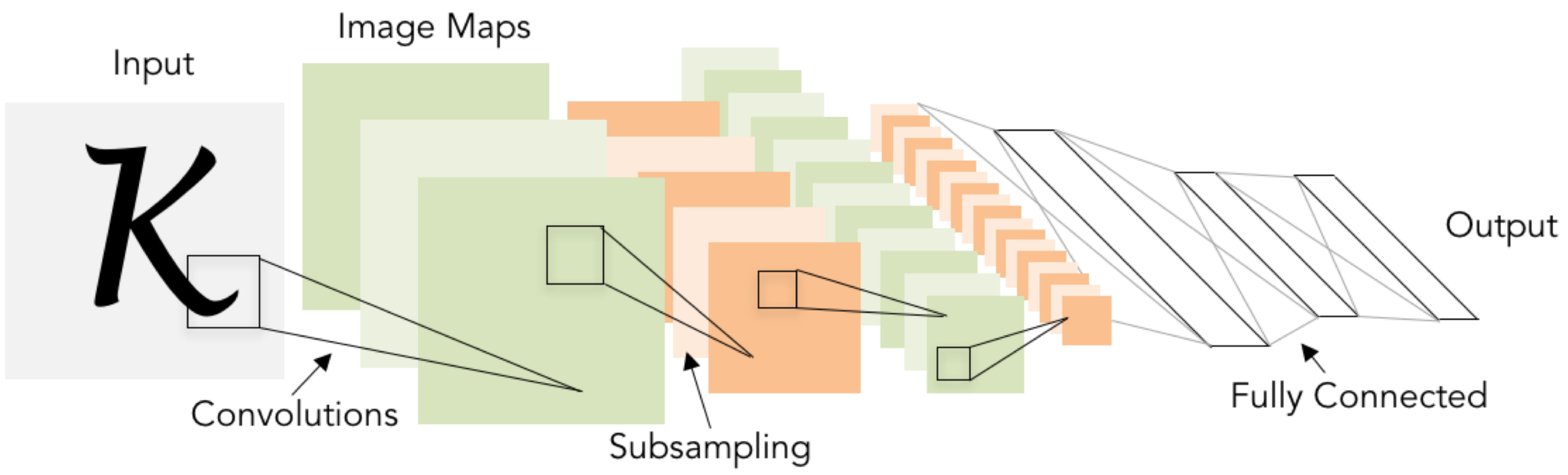
# Example: LeNet-5

| 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 |



# 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:

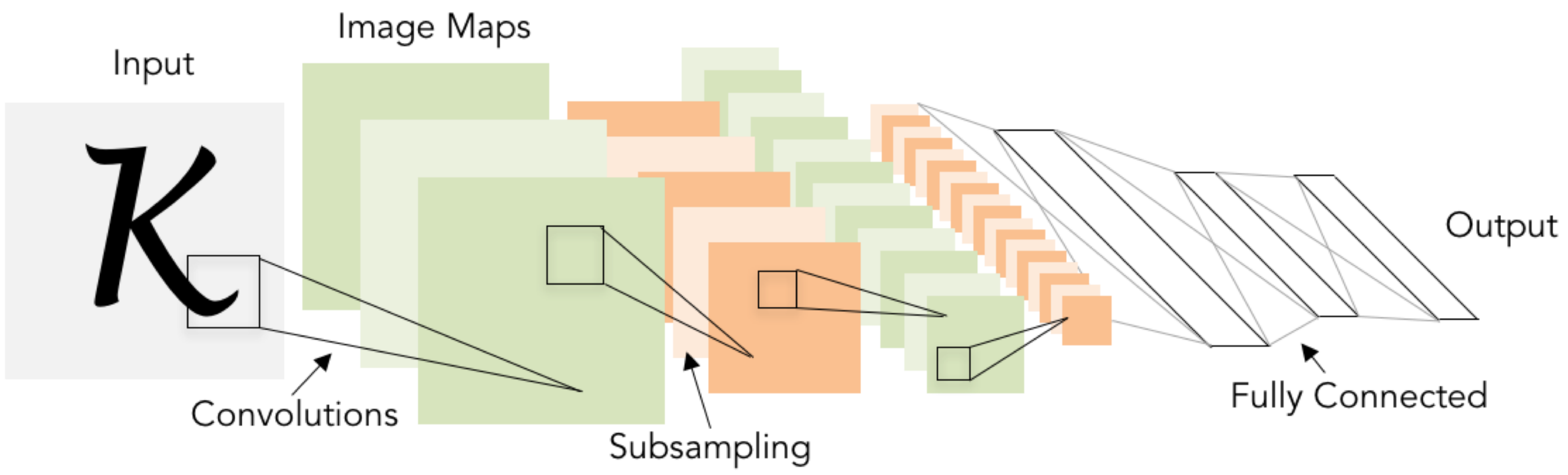
Spatial size decreases

(using pooling or striped convolution)
Number of channels increases

(total “volume” is preserved!)

# 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**

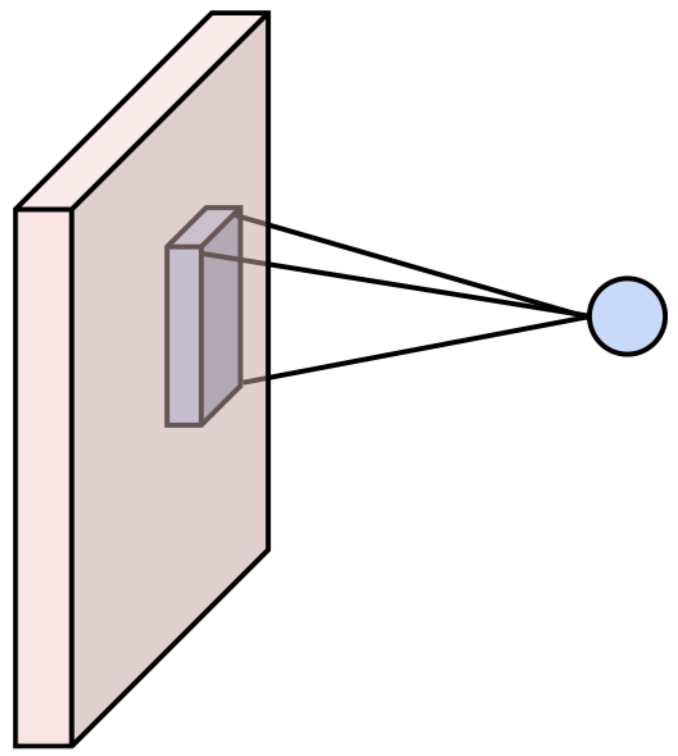
(using pooling or striped convolution)
Number of channels **increases**

(total "volume" is preserved!)

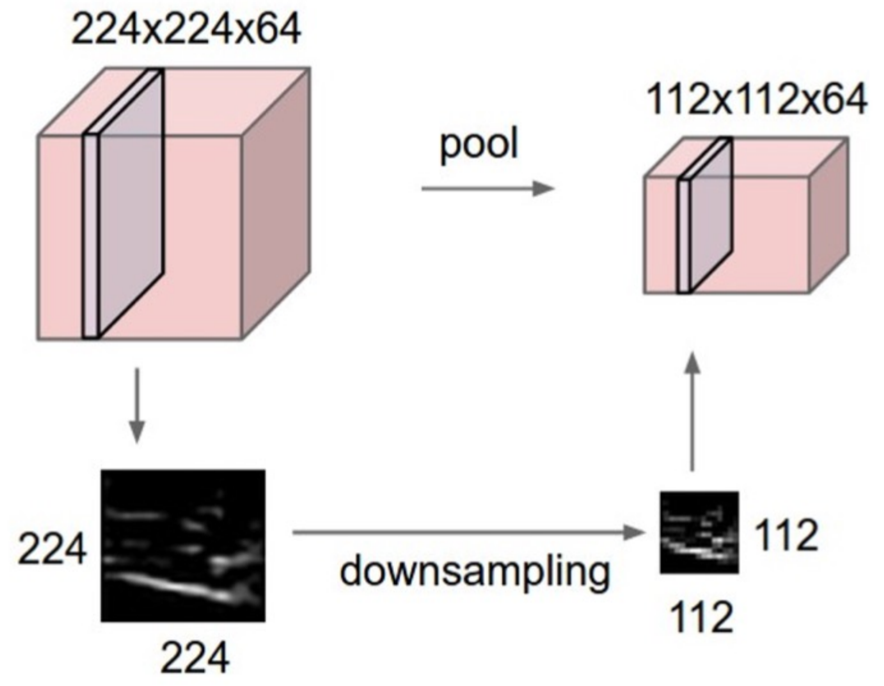
Some modern architectures break this trend—stay tuned!

# Components of Convolutional Networks

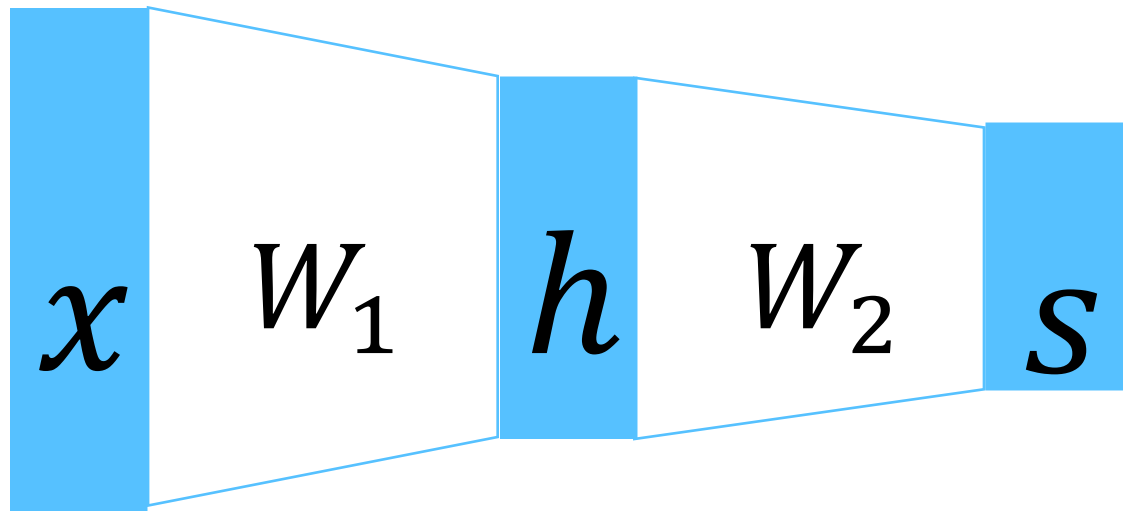
## Convolution Layers



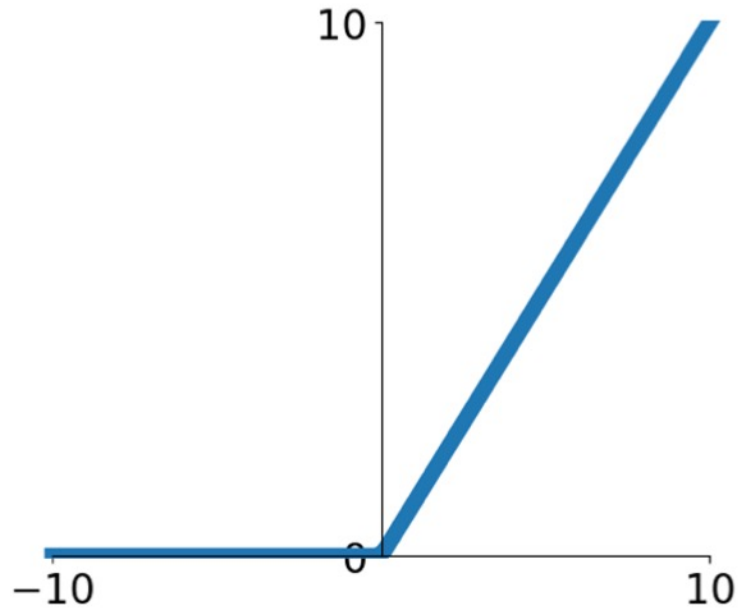
## Pooling Layers




## Fully-Connected Layers




## Activation Function




## Normalization

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

Problem: Deep Networks very hard to train

# 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

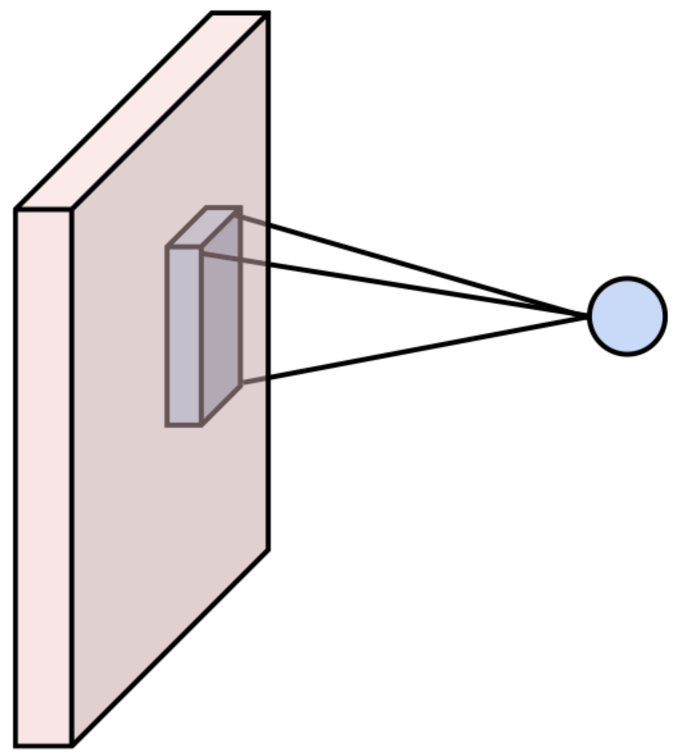
We can normalize a batch of activations using:

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

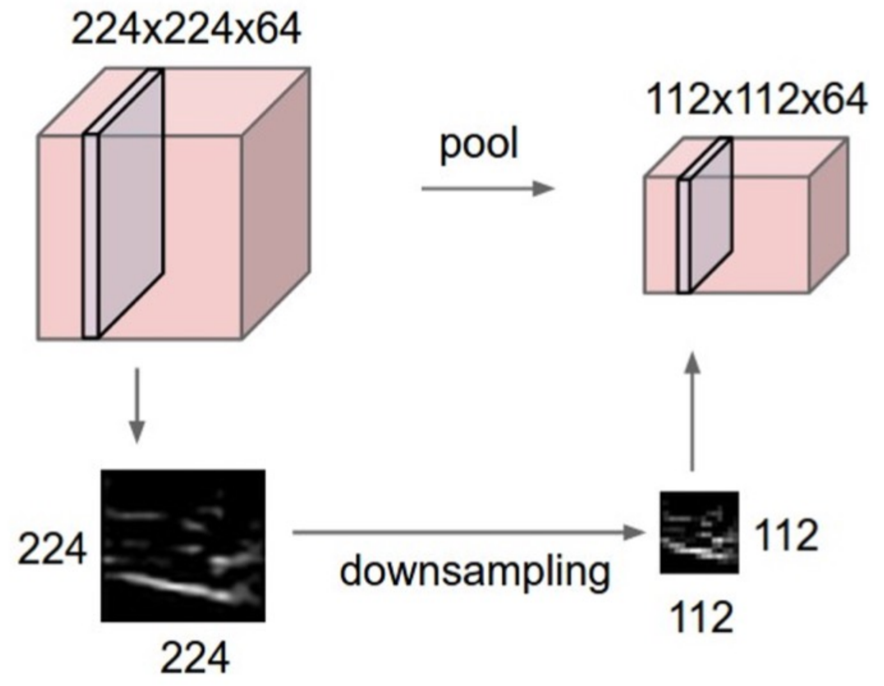
This is a **differentiable function**, so we can use it as an operator in our networks and backdrop through it!

# Components of Convolutional Networks

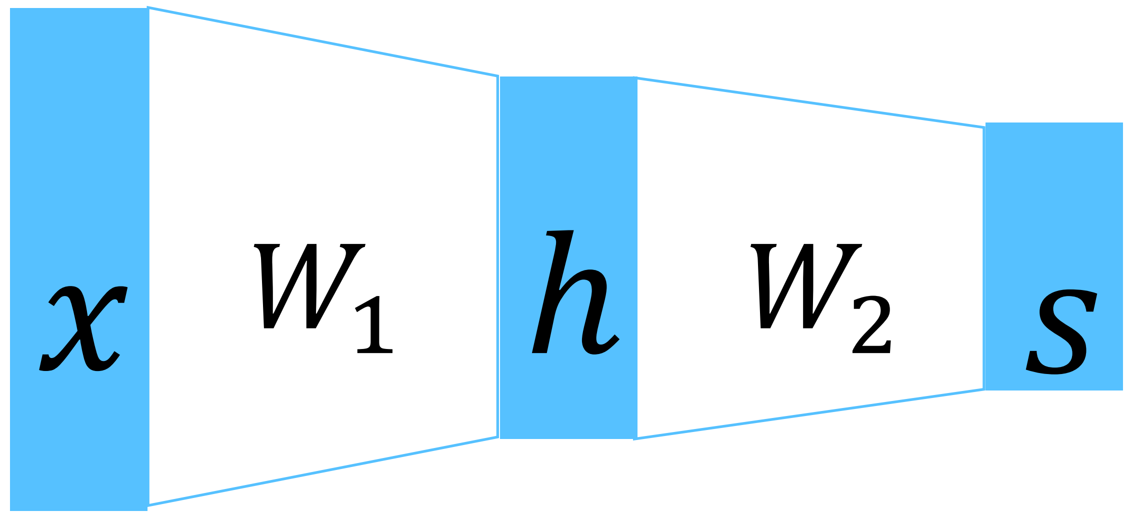
## Convolution Layers



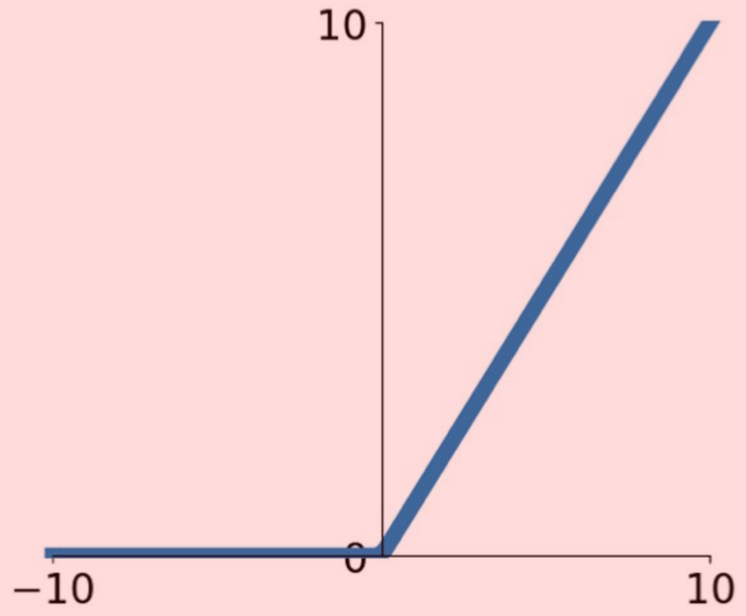
## Pooling Layers




## Fully-Connected Layers




## Activation Function




## Normalization

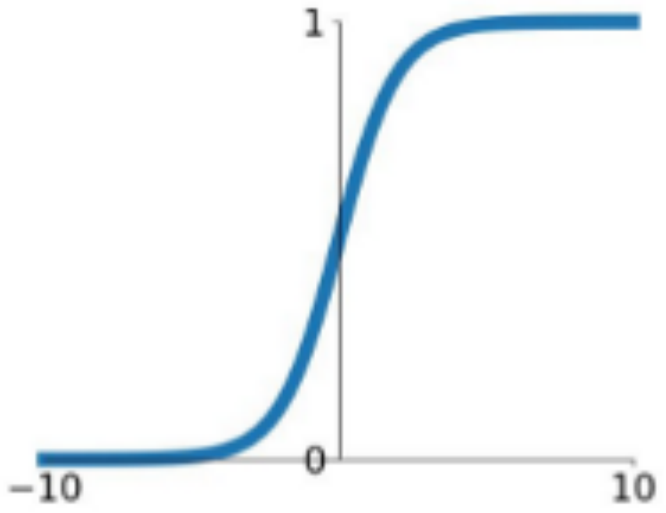
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Problem: Deep Networks very hard to train

# Activation Functions

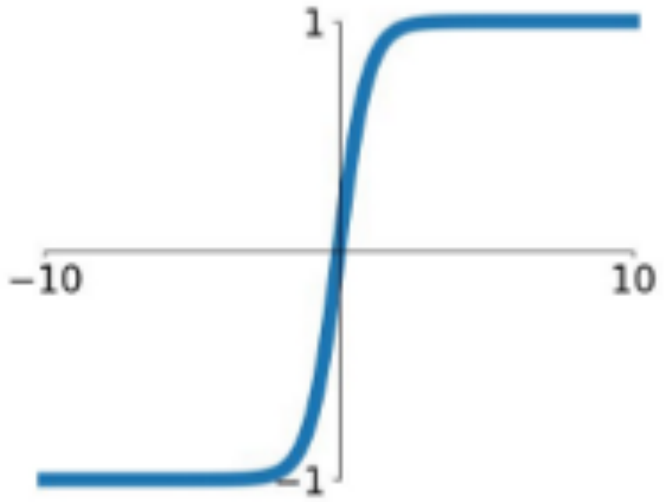
**Sigmoid**

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



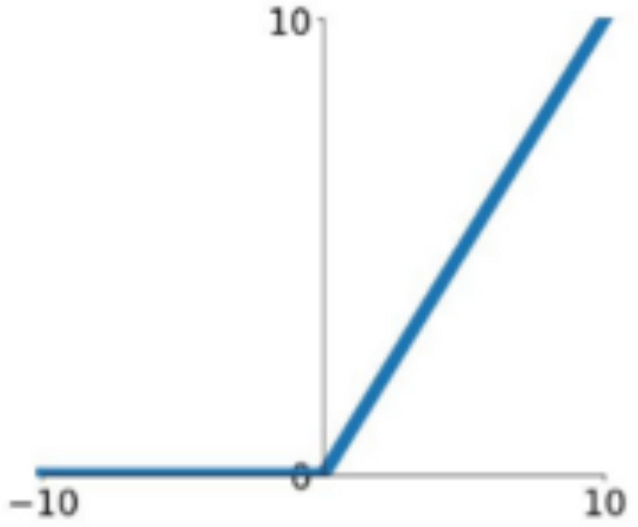

**tanh**

$\tanh(x)$



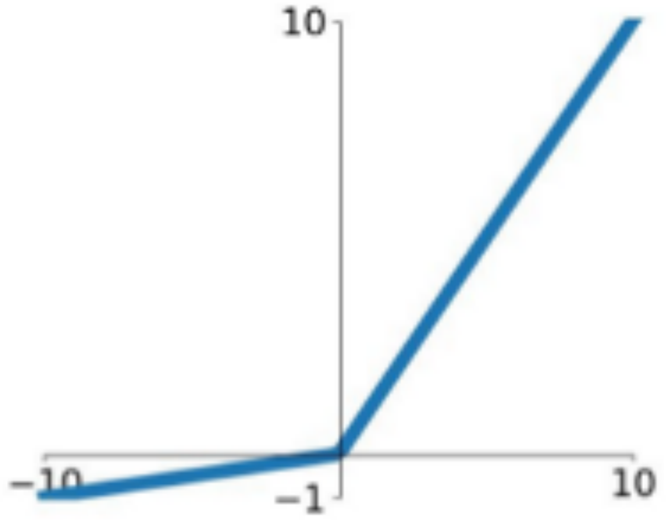

**ReLU**

$\max(0, x)$




**Leaky ReLU**

$\max(0.1x, x)$

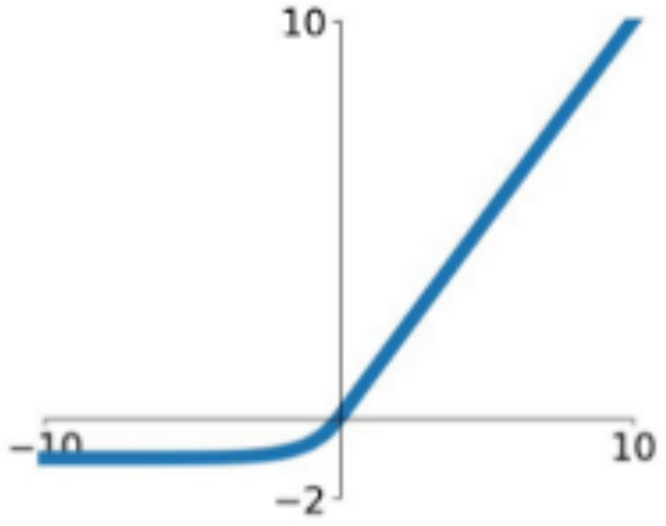



**Maxout**

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

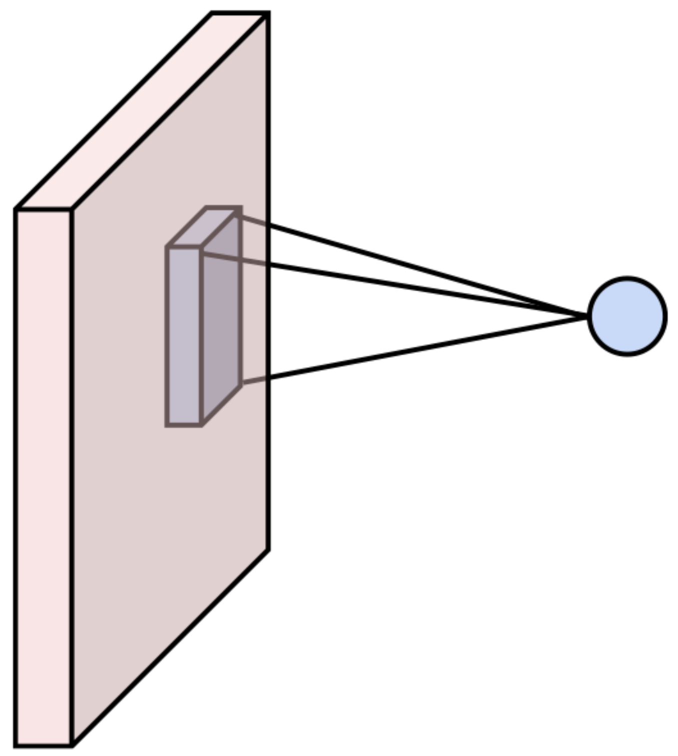
**ELU**

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

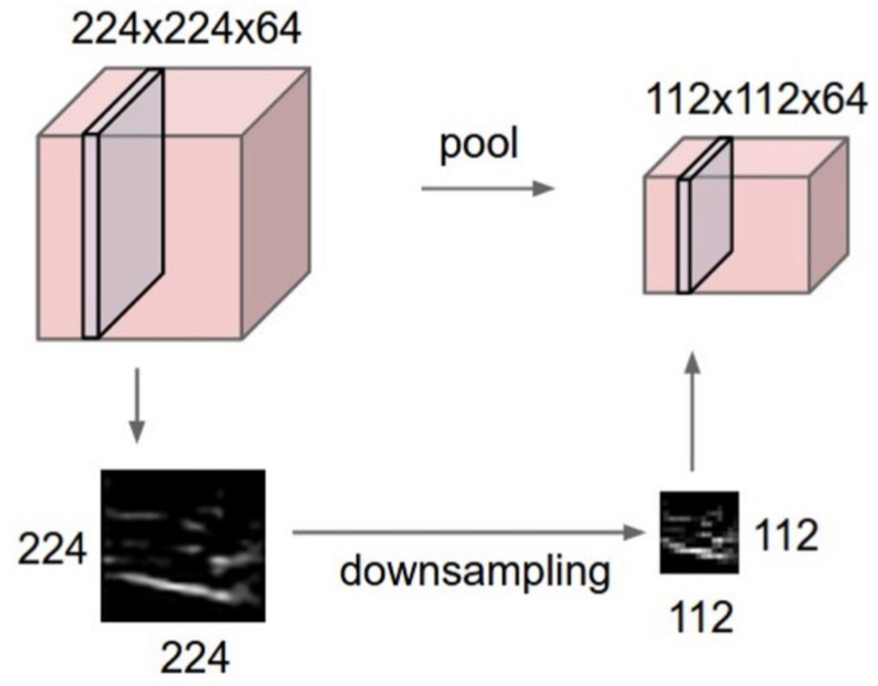


# Summary: Components of Convolutional Networks

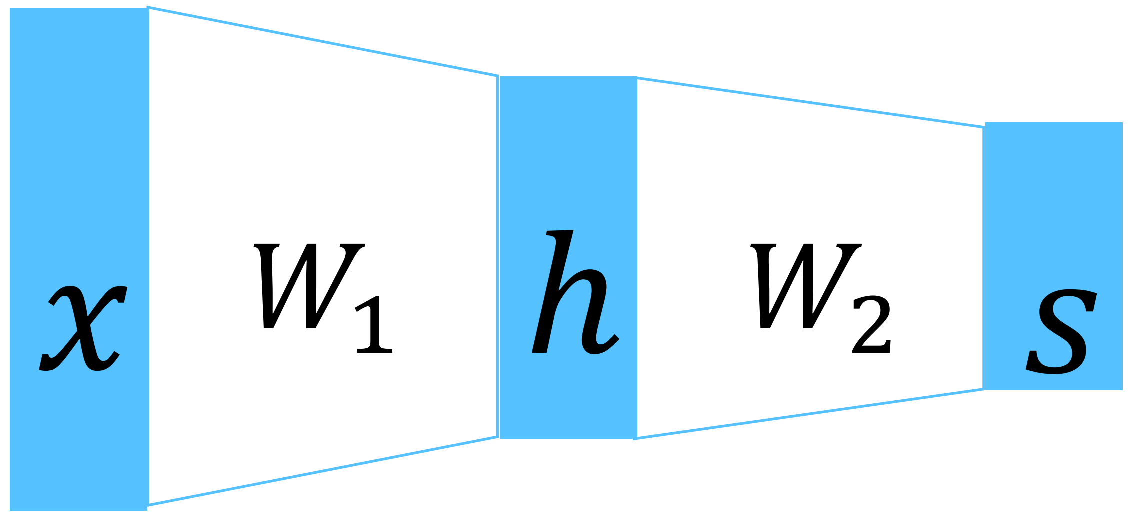
## Convolution Layers



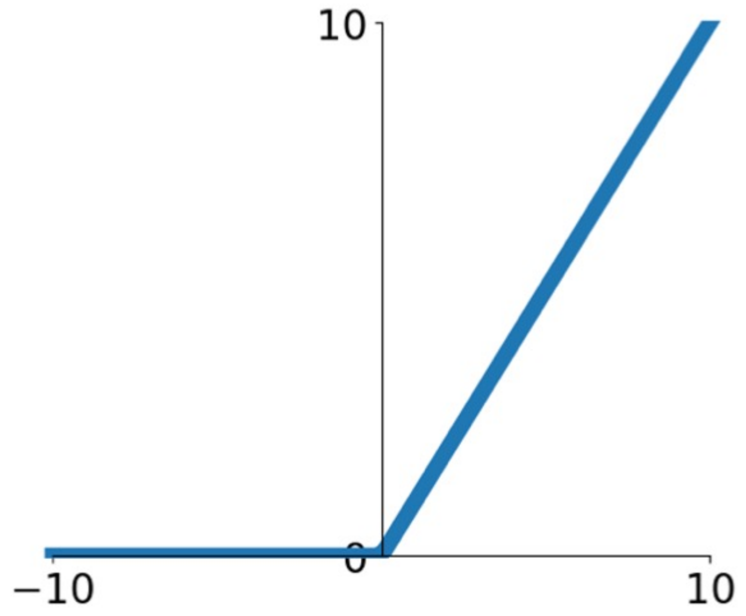
## Pooling Layers




## Fully-Connected Layers




## Activation Function

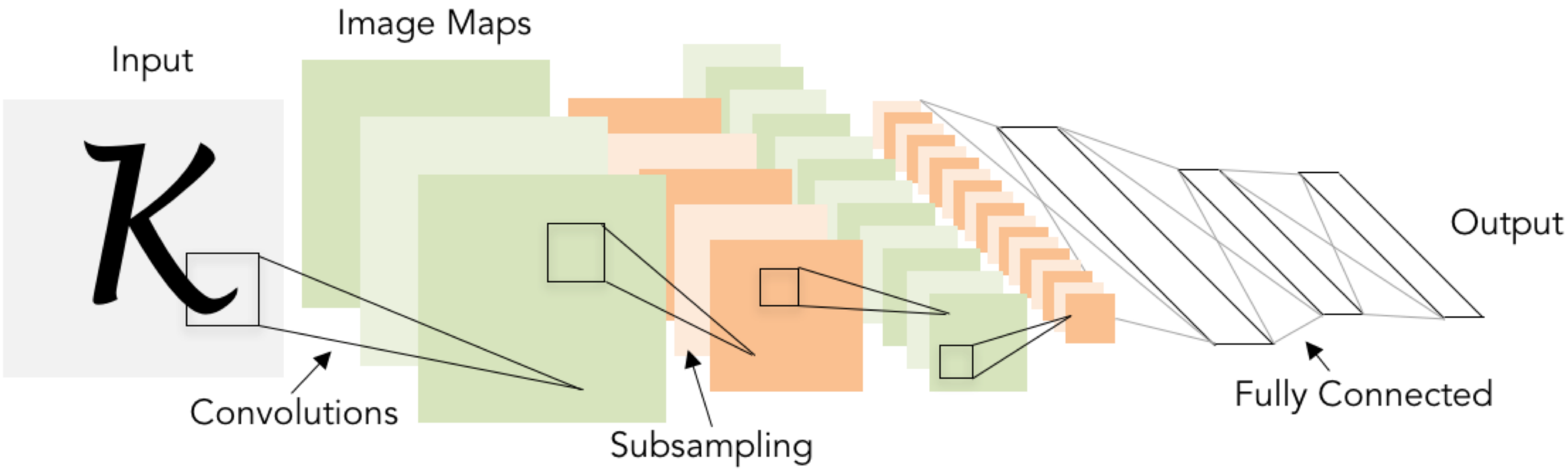



## Normalization

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

# Summary: Components of Convolutional Network

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



# Project 1—Reminder

- Instructions and code available on the website
  - Here: deeprob.org/projects/project1/
- Implement KNN, linear classifier, and fully connected NN
- **Due Thursday, Feb.1, 11:59 PM EST**
- **Discussion section: Your Thoughts?**
- **Late policy: 3 late tokens (24hrs each with no penalty); 25% deduction for every day the submission was late after using all three late tokens**

# Helpful References

- https://cs231n.github.io/linear-classify/
- https://cs231n.github.io/optimization-1/
- https://cs231n.github.io/optimization-2/
- https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

# Final Project Overview

- Research-oriented final project

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

completed in teams

# Final Project Deliverables

1. A written paper review
2. In-class paper presentation
3. Reproduce published results
4. Extend results with new idea, technique or dataset
5. Document results in written report

# (1) Paper Review and (2) Presentation

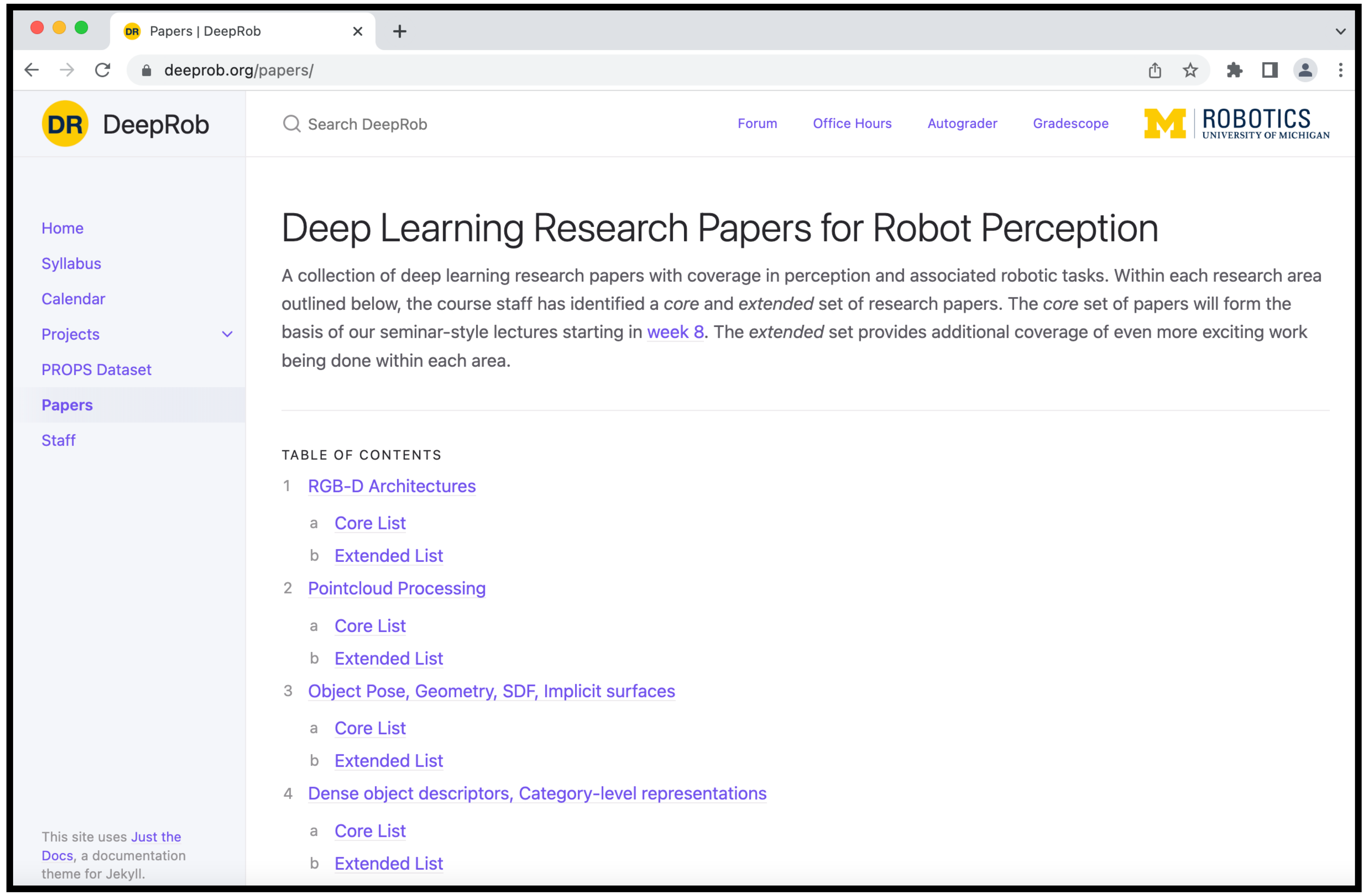
## Final project teams will be based on overlapping interest

Students will choose from the 'core' list of papers on course website

Each team will be assigned one of the 'core' papers to review and present in-class

The 1-page paper review will be due **1-week before** the scheduled presentation

Presentation schedule will be based on paper topic as shown in course calendar




More details on review and presentation criteria in following lectures

# (3) Paper Reproduction and (4) Extension

Each team will choose a paper relating to deep learning and robot perception

Doesn't have to be same paper you presented in class

Then reimplement and reproduce at least one of the paper's published results **(not necessarily all the results)**

Then, each team will test one of their own ideas!

By extending the paper's model using new architecture or technique or dataset

**Your chance to experiment with deep learning and contribute to the field!**

More details on reproduction and extension in following lectures

# (5) Project Report

- The final deliverable for your final project
- A report/paper
  - What problem within robot perception or manipulation?
  - What work has been done in this area?
  - What approach did you investigate?
  - What questions and directions exist for future work?

More details on report in following lectures

# DeepRob

**Lecture 6**
**Convolutional Neural Networks**
**University of Michigan | Department of Robotics**