ECE 884 Deep Learning

Lecture 13-14: Convolutional Neural Networks

03/09,11/2021

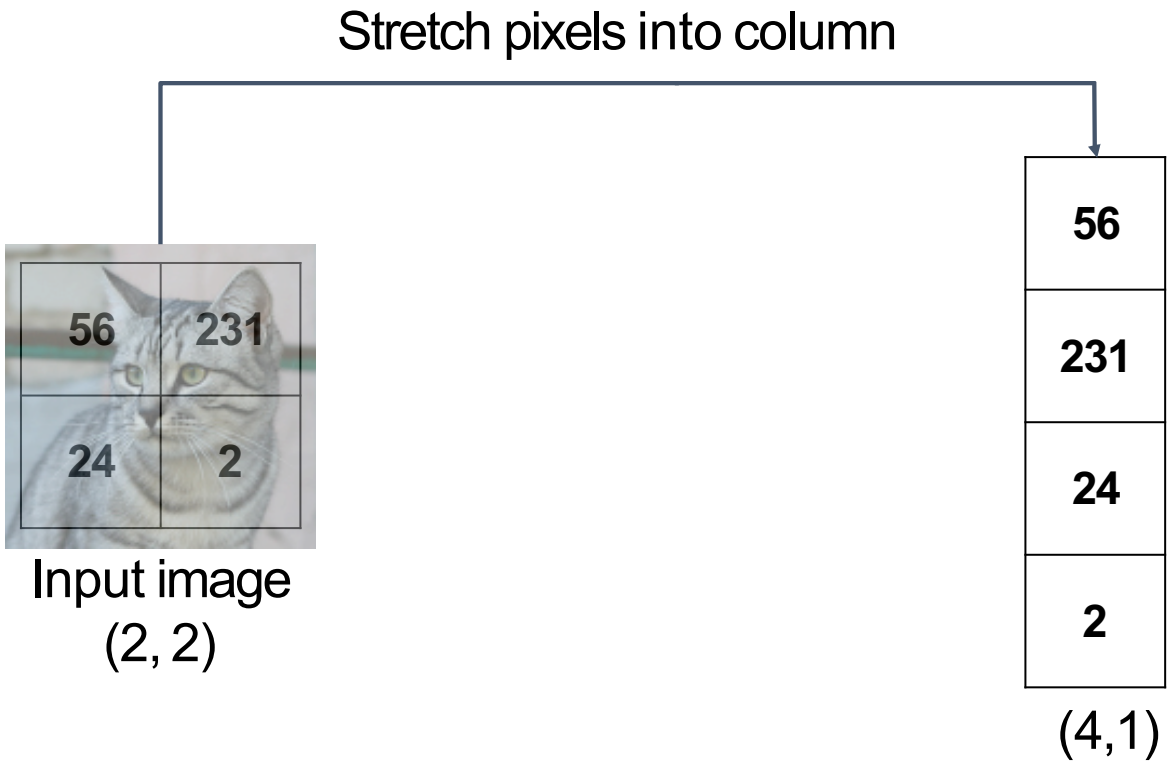
Review of last lecture

- Deep Neural Network Training Pipeline
  - Data preprocessing
  - Regularization for Neural Networks
  - Learning Rate Schedules

Today’s lecture

- Convolutional Neural Network (ConvNet)

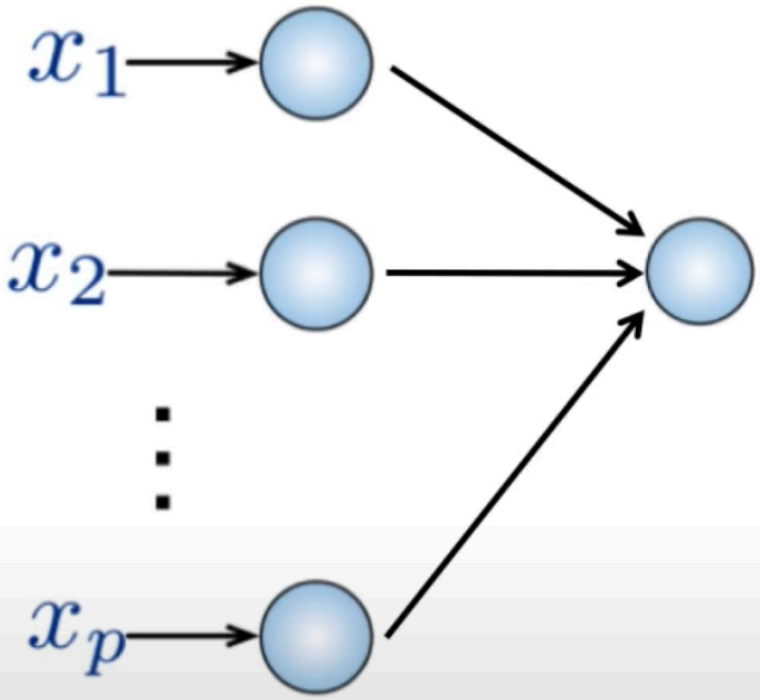
Problem: Sofar our classifiers don't respect the spatial structure of images



Problem: Sofar our classifiers don't respect the spatial structure of images

56
231
24
2

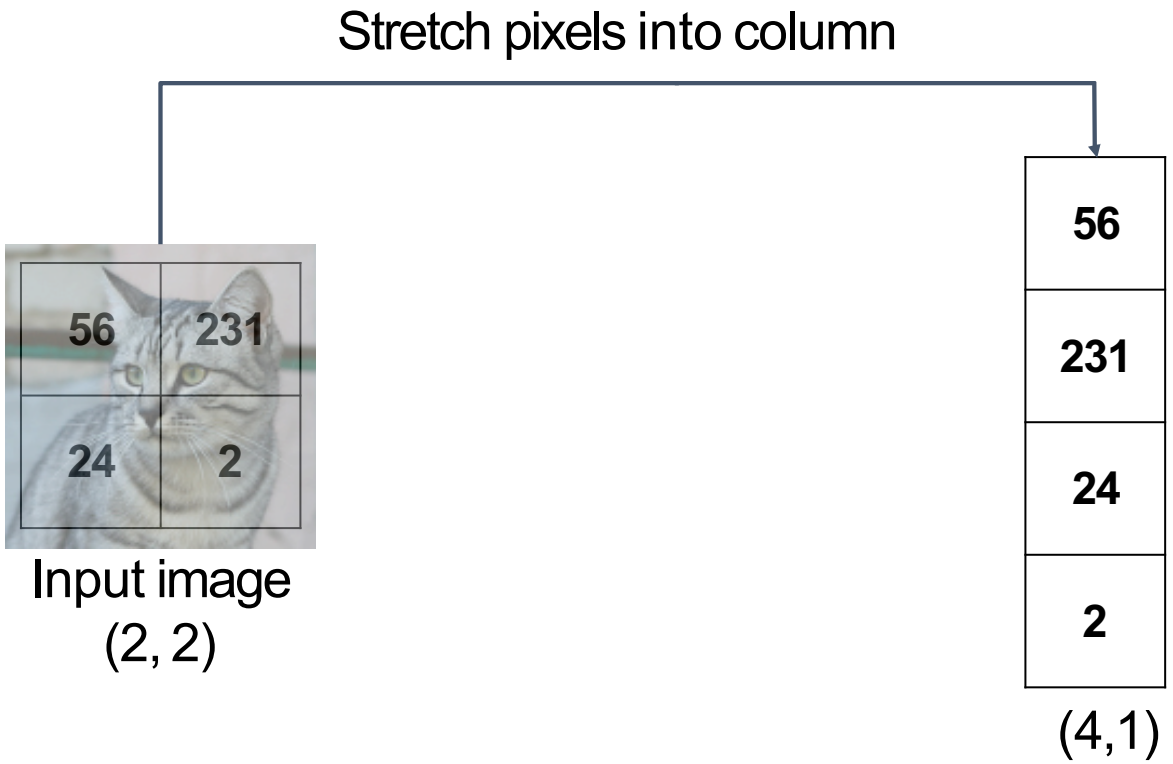
(4,1)



Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

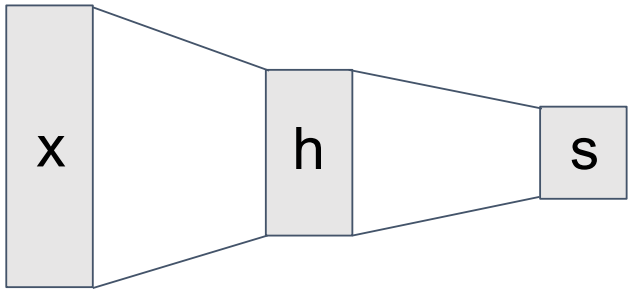
Problem: Sofar our classifiers don't respect the spatial structure of images



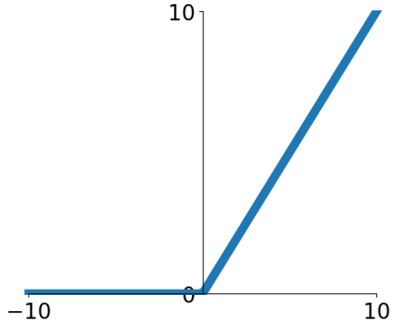
Solution: Define new computational nodes that operate on images!

Components of a Fully-Connected Network

Fully-Connected Layers

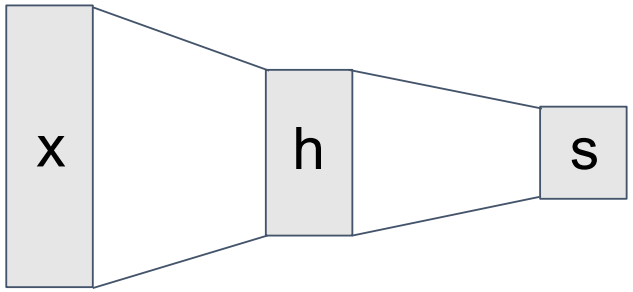


Activation Function

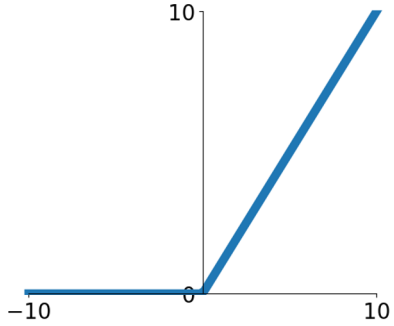


Components of a Convolutional Network

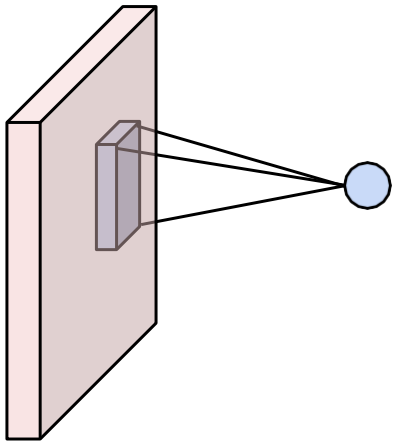
Fully-Connected Layers



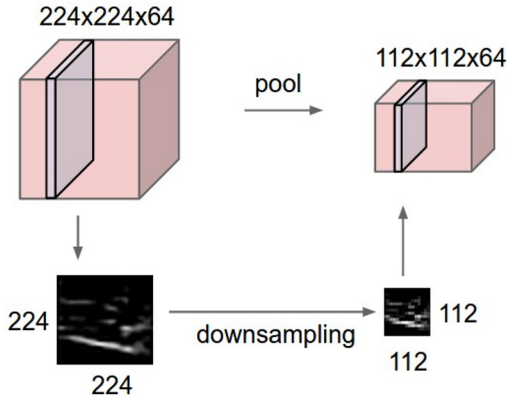
Activation Function



Convolution Layers



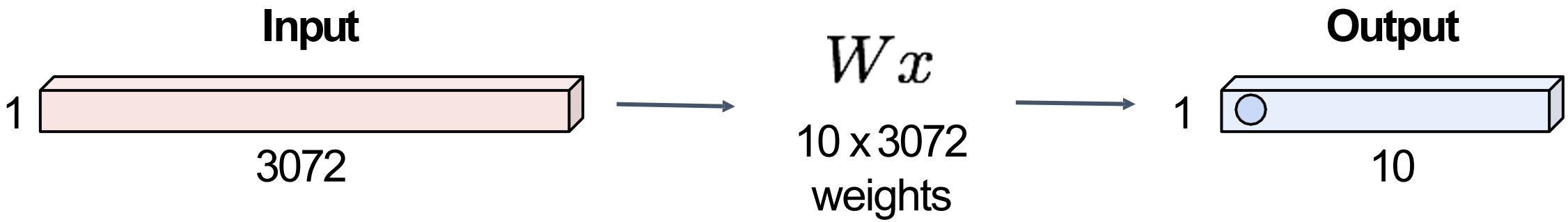
Pooling Layers



Batch Normalization Layers

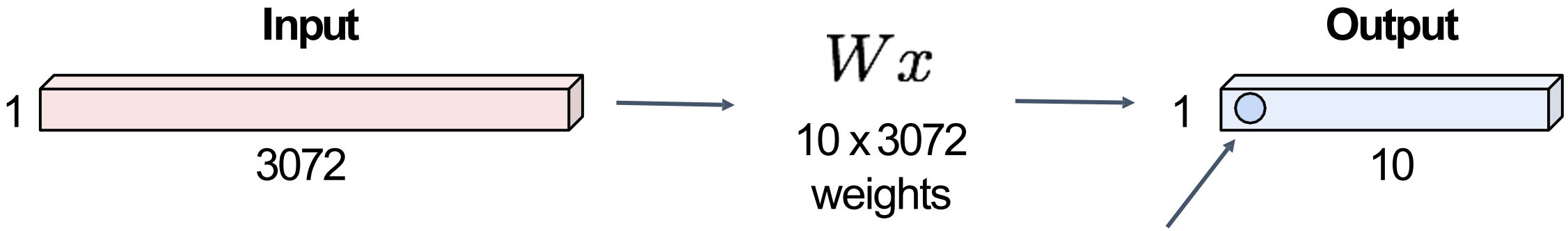
Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1



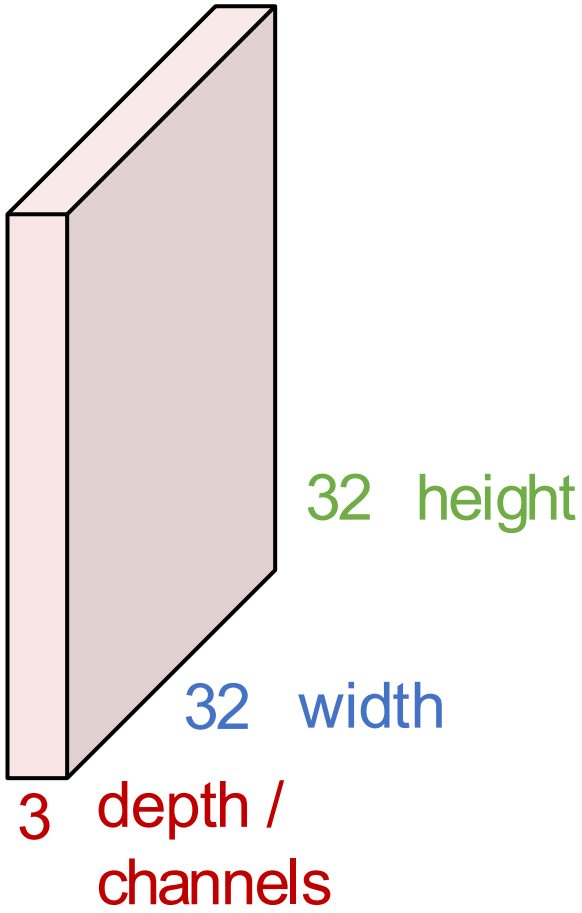
Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1



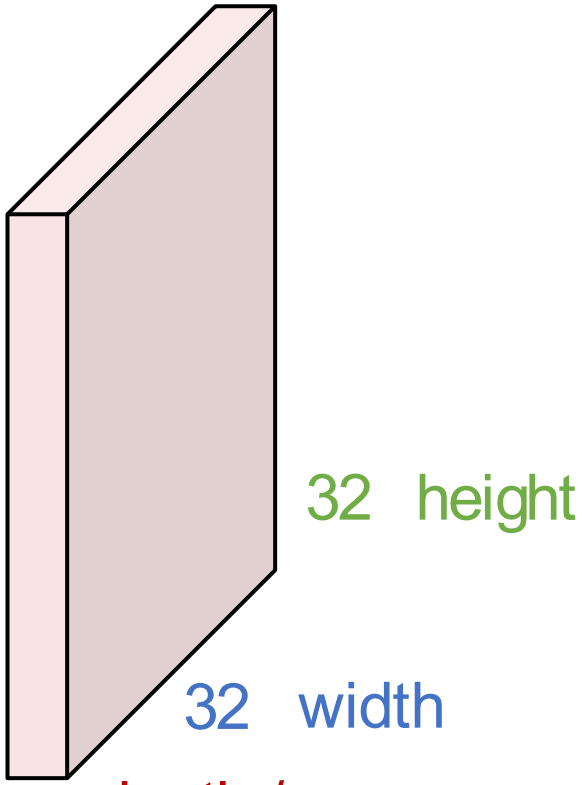
Convolution Layer

3x32x32 image: preserve spatial structure

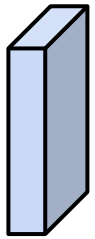


Convolution Layer

3x32x32 image



3x5x5 filter

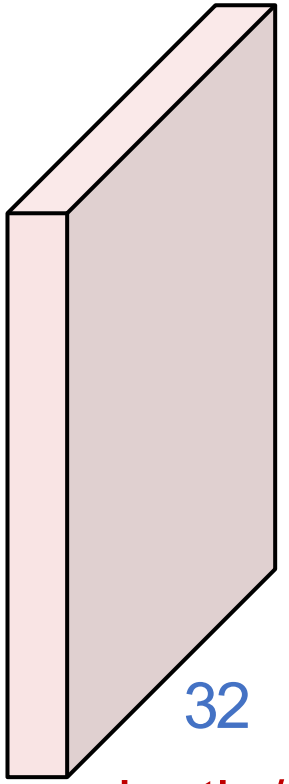


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

Filters always extend the full depth of the input volume

3x32x32 image

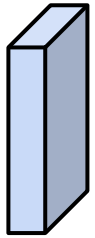


32 height

32 width

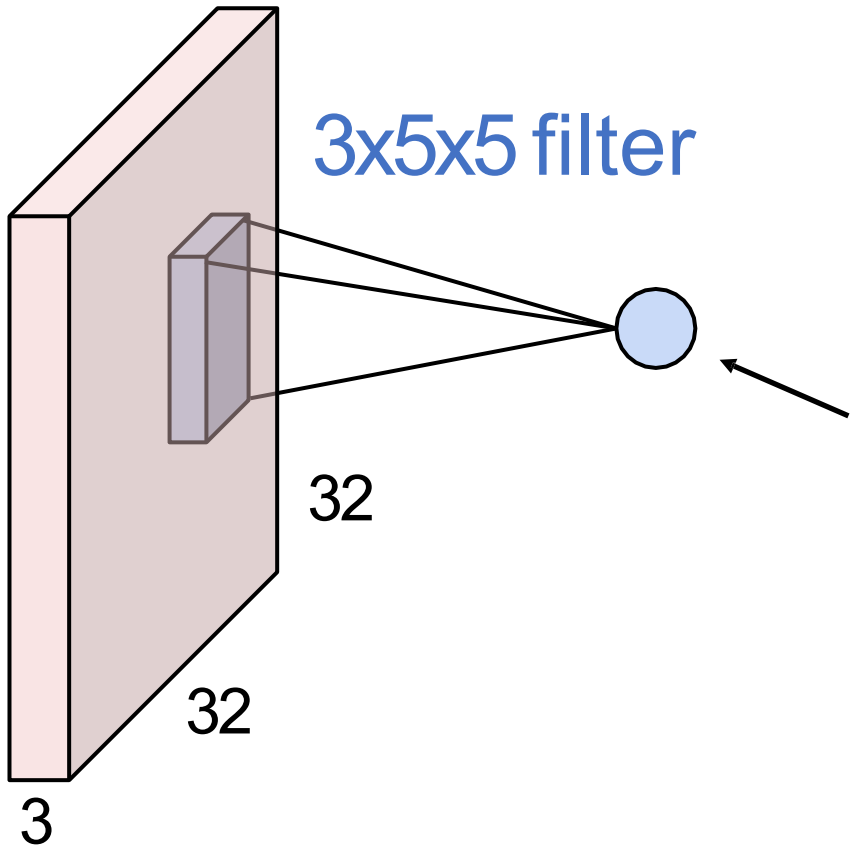
3 depth / channels

3x5x5 filter



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

Convolution Layer



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)

$$w^T x + b$$

2D Convolution

Input Image

5×5

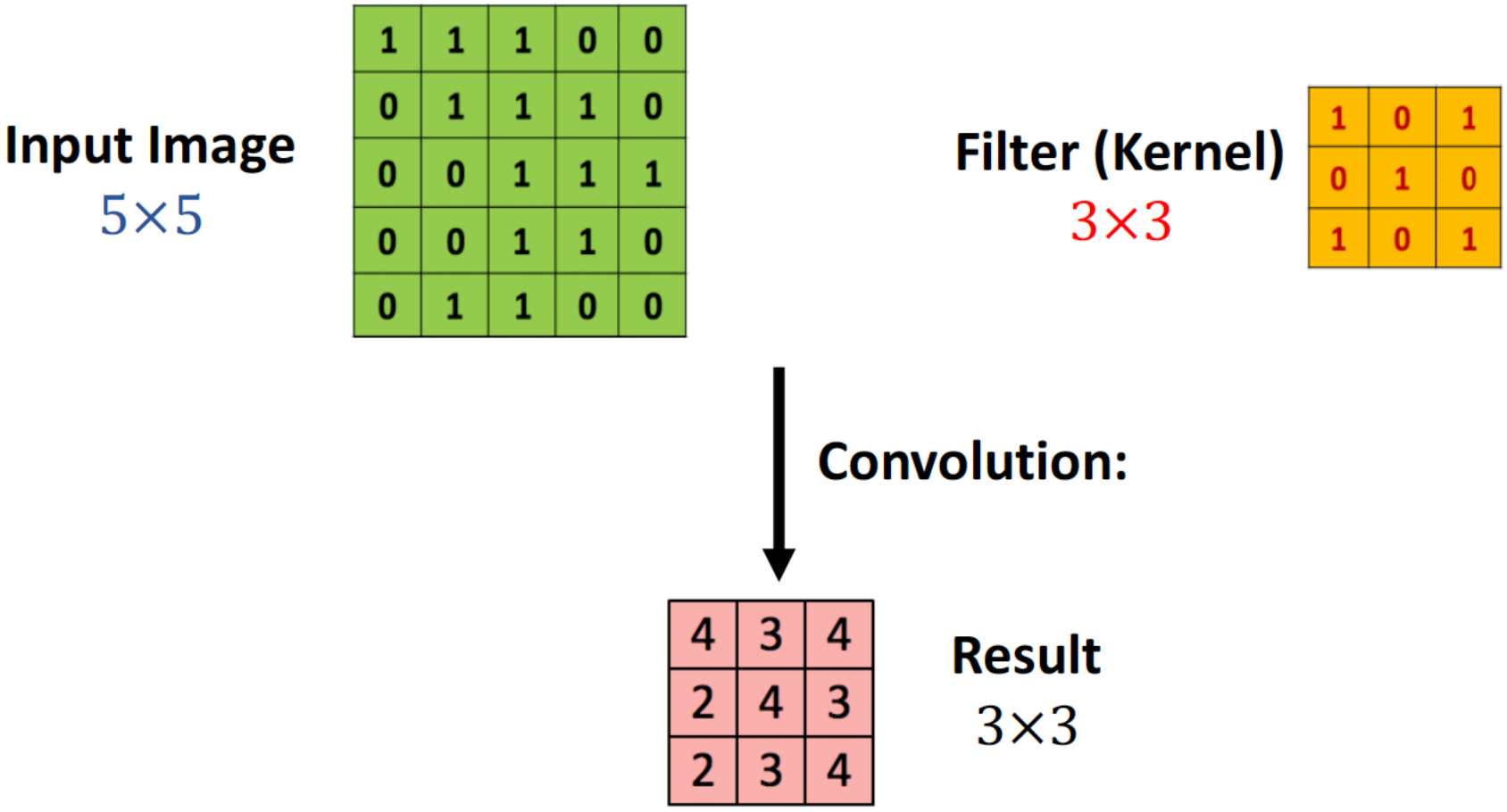
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter (Kernel)

3×3

1	0	1
0	1	0
1	0	1

2D Convolution



2D Convolution

Input Image
5×5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

Convolution:

4	3	4
2	4	3
2	3	4

Result
3×3

The value 4 is the inner product of the patch

1	1	1
0	1	1
0	0	1

and the filter

1	0	1
0	1	0
1	0	1

.

2D Convolution

Input Image
5×5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

Convolution:

4	3	4
2	4	3
2	3	4

Result
3×3

The value 3 is the inner product of the patch

1	1	0
1	1	1
0	1	1

and the filter

1	0	1
0	1	0
1	0	1

.

2D Convolution

Input Image
5×5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

Convolution:

4	3	4
2	4	3
2	3	4

Result
3×3

The value 4 is the inner product of the patch

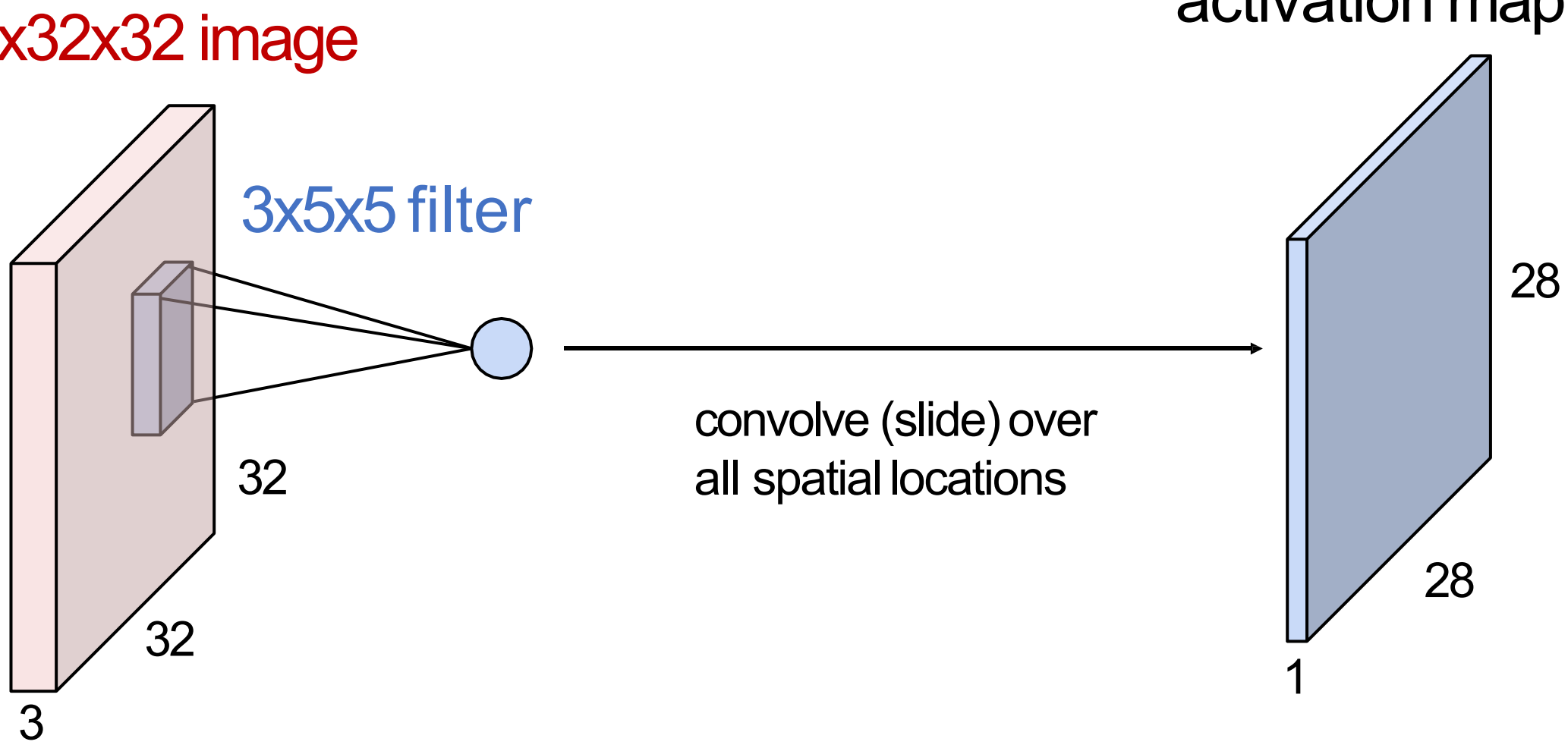
1	1	1
1	1	0
1	0	0

and the filter

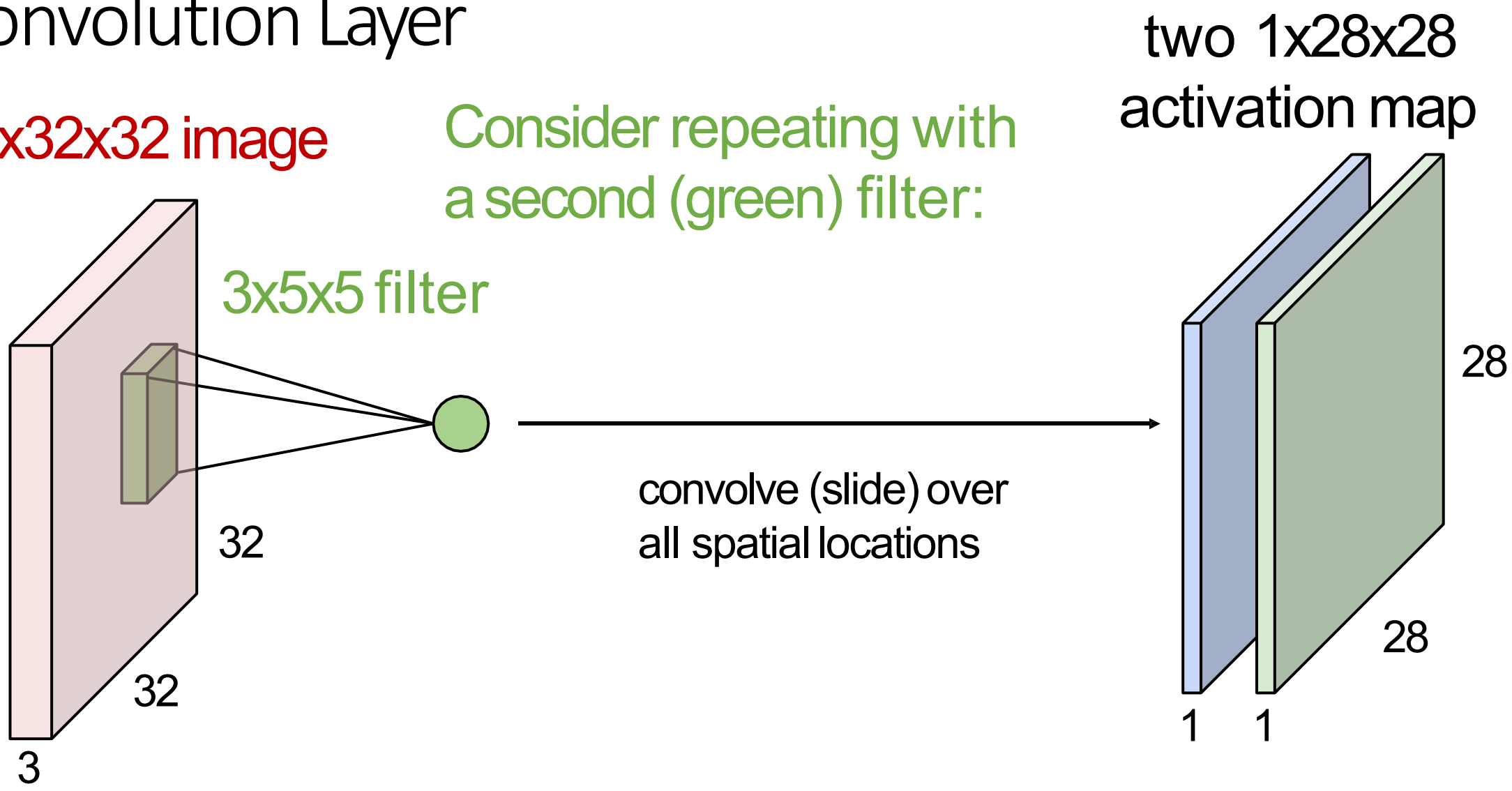
1	0	1
0	1	0
1	0	1

.

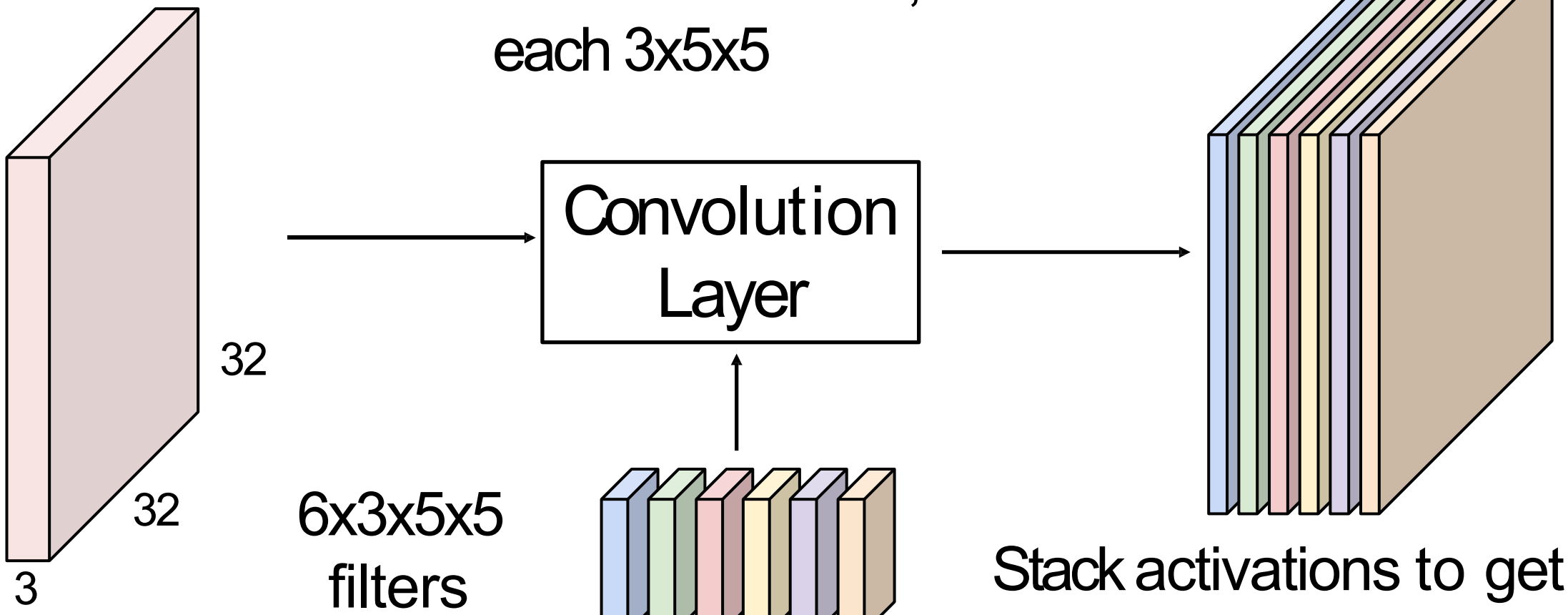
Convolution Layer



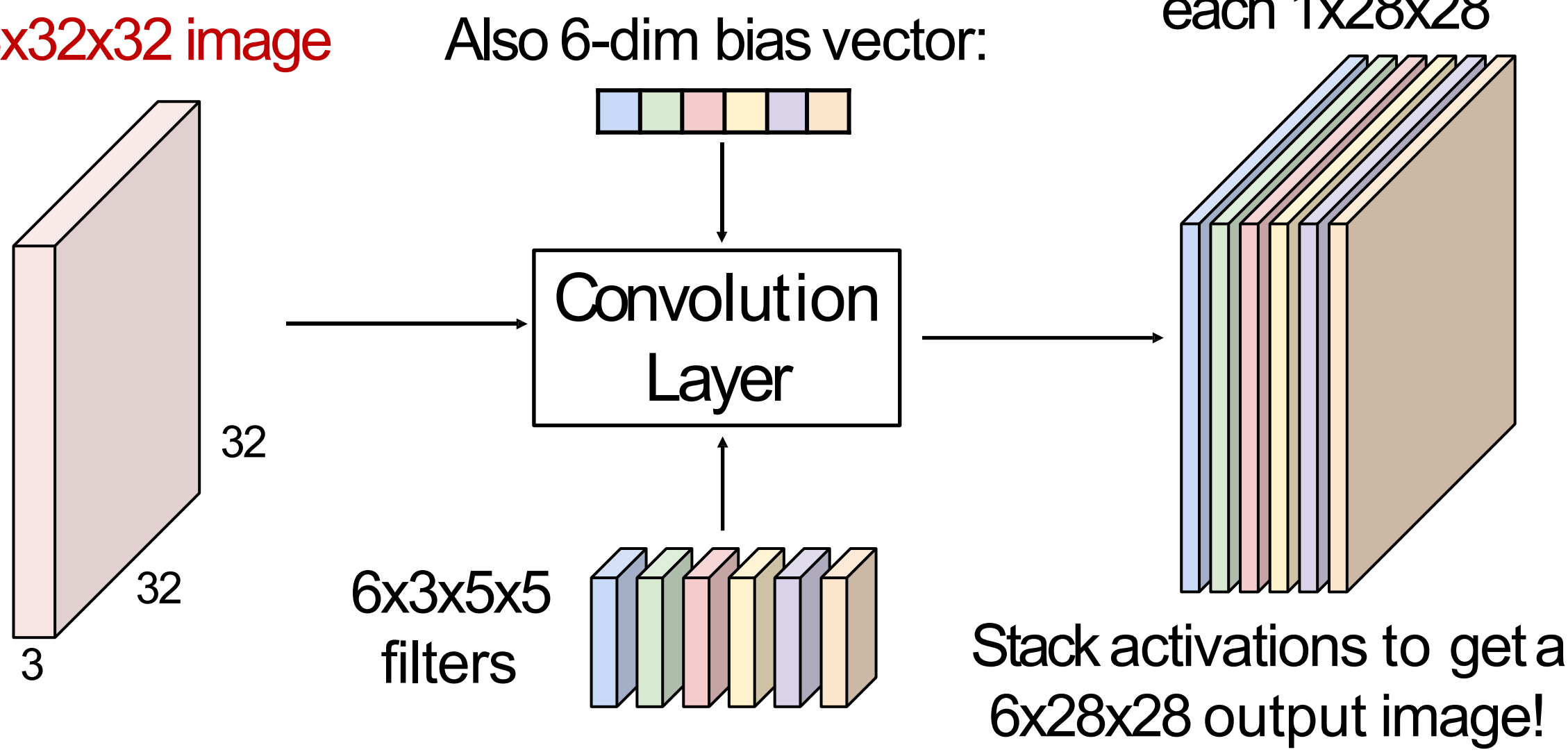
Convolution Layer



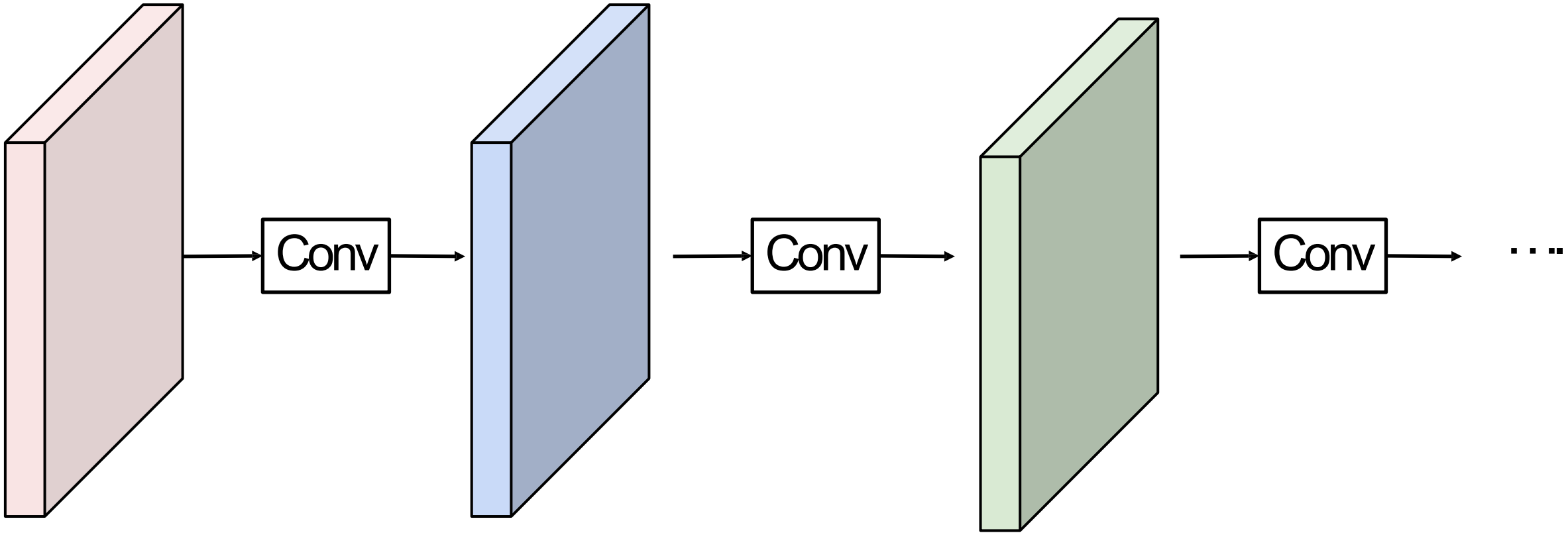
Convolution Layer



Convolution Layer

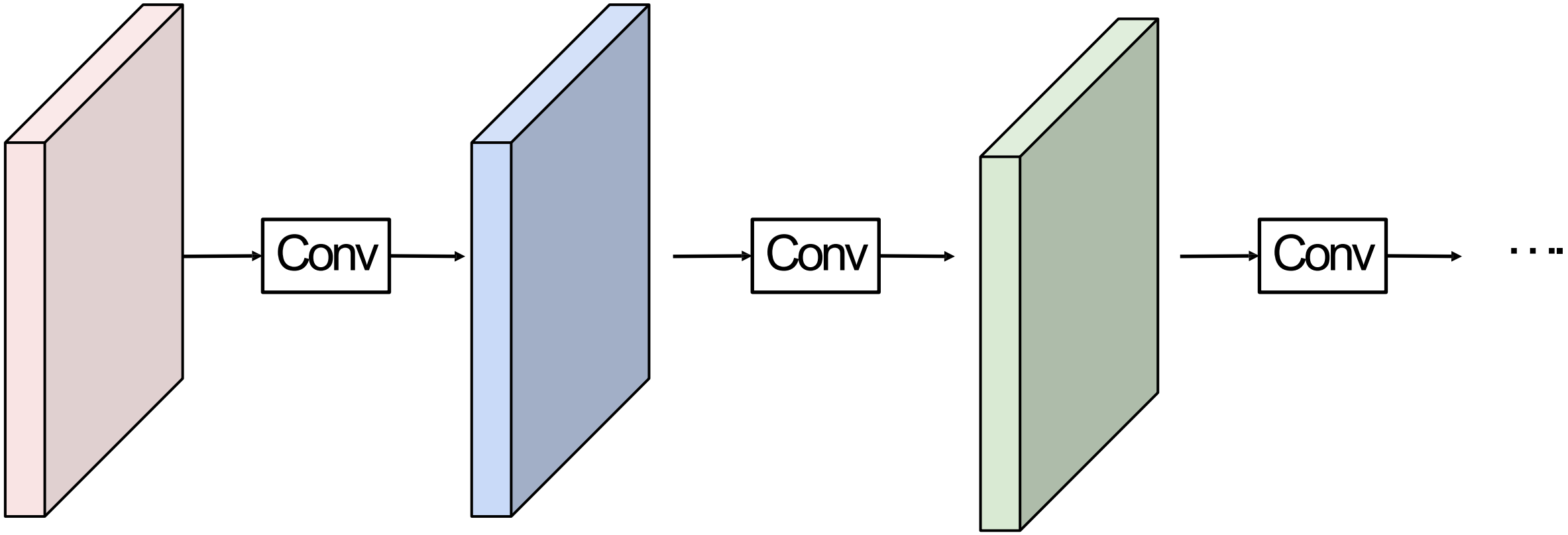


Stacking Convolutions



Stacking Convolutions

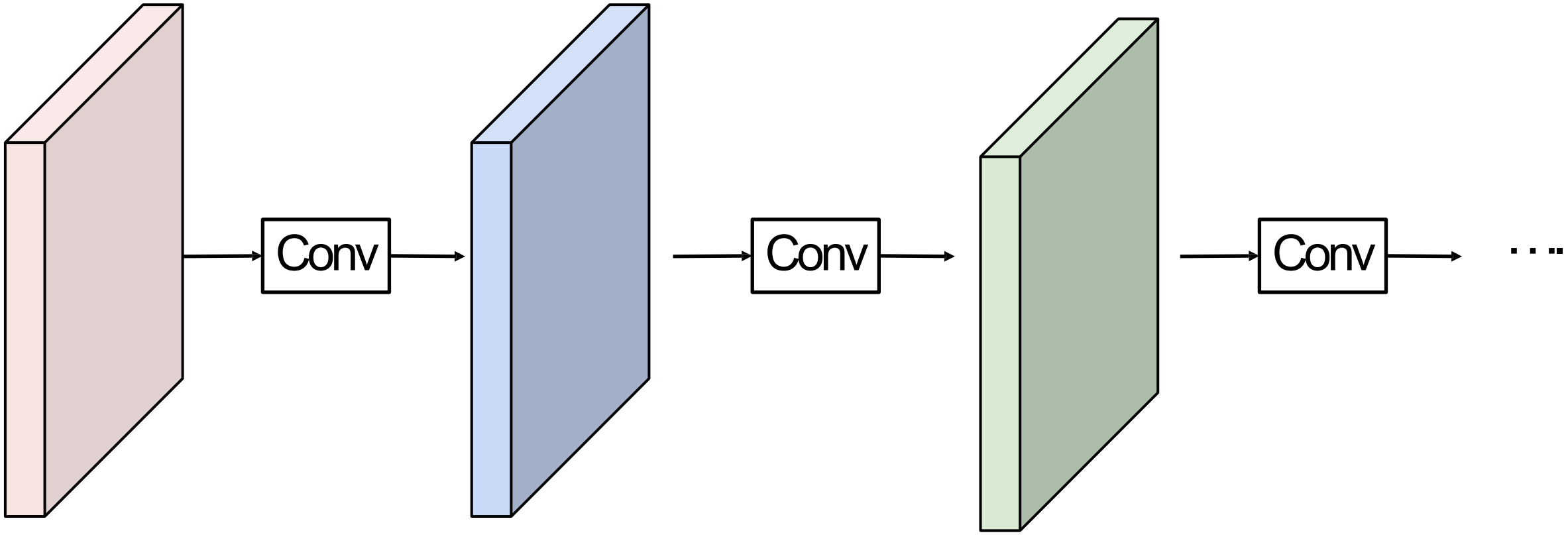
Q: What happens if we stack two convolution layers?



Stacking Convolutions

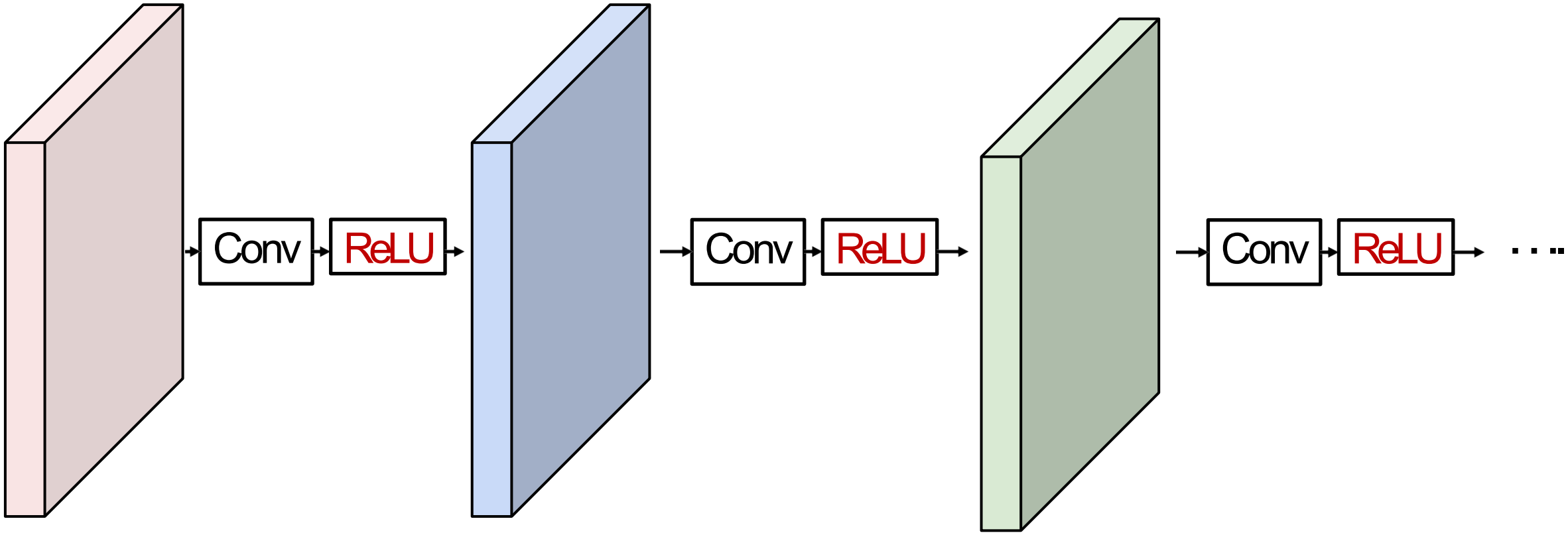
Q: What happens if we stack two convolution layers?

A: We get another convolution!

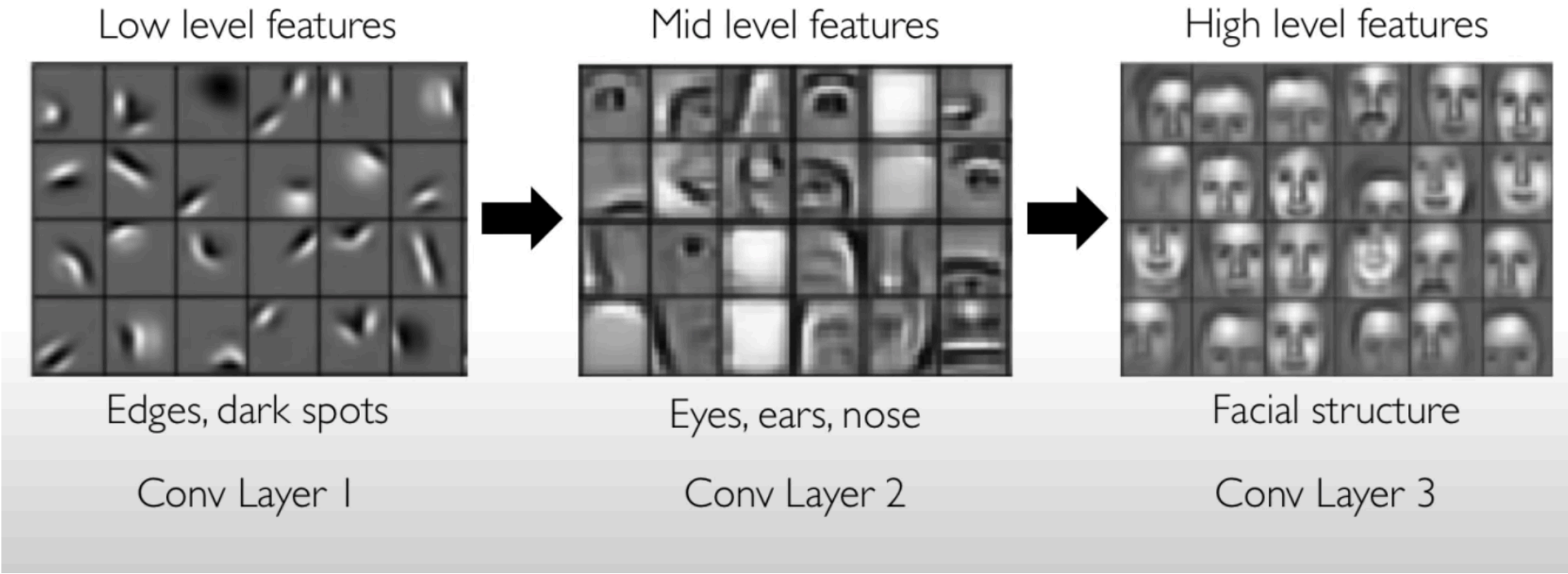


Convolution is a linear operation!

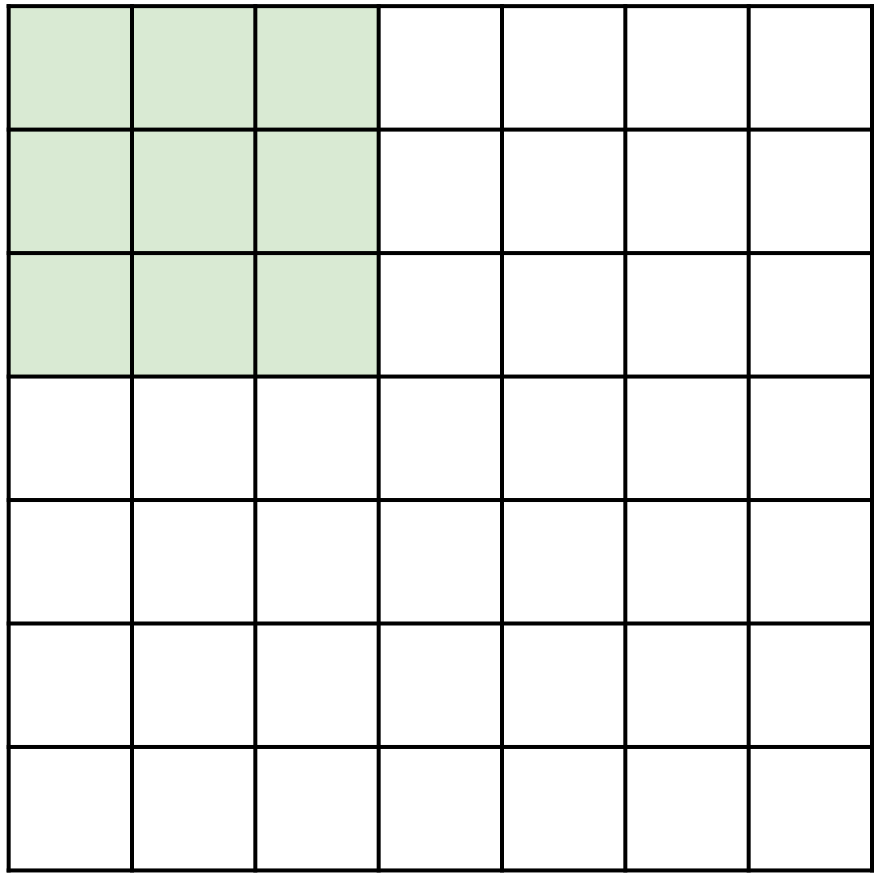
Stacking Convolutions



What do convolutional filters learn?

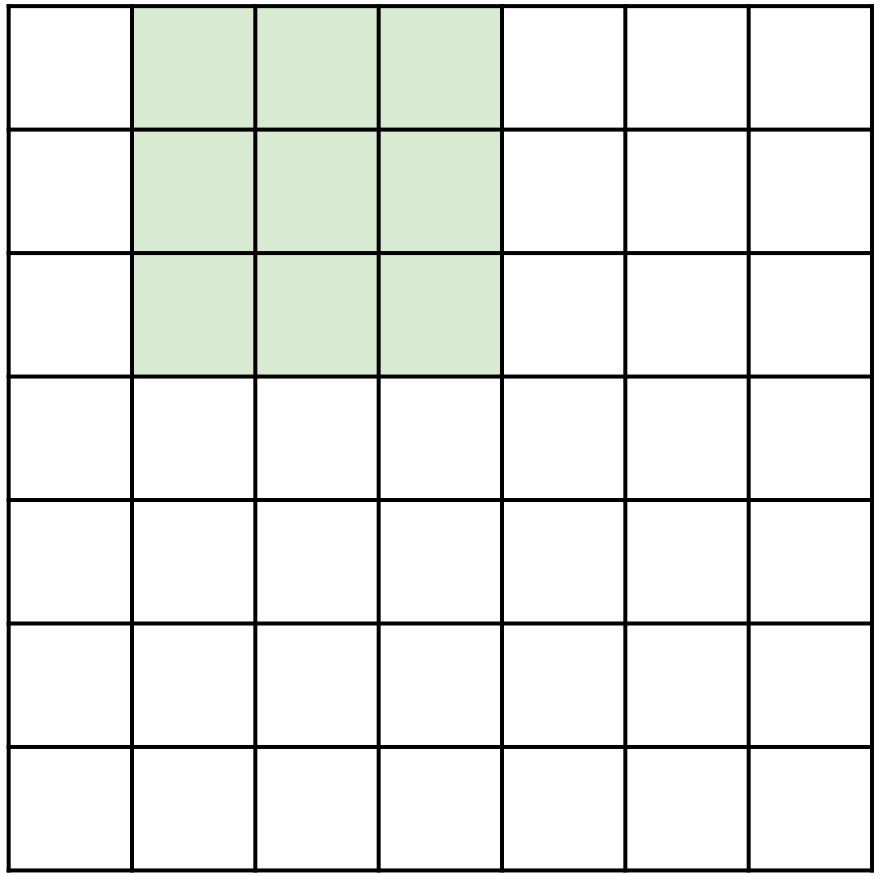


A closer look at spatial dimensions



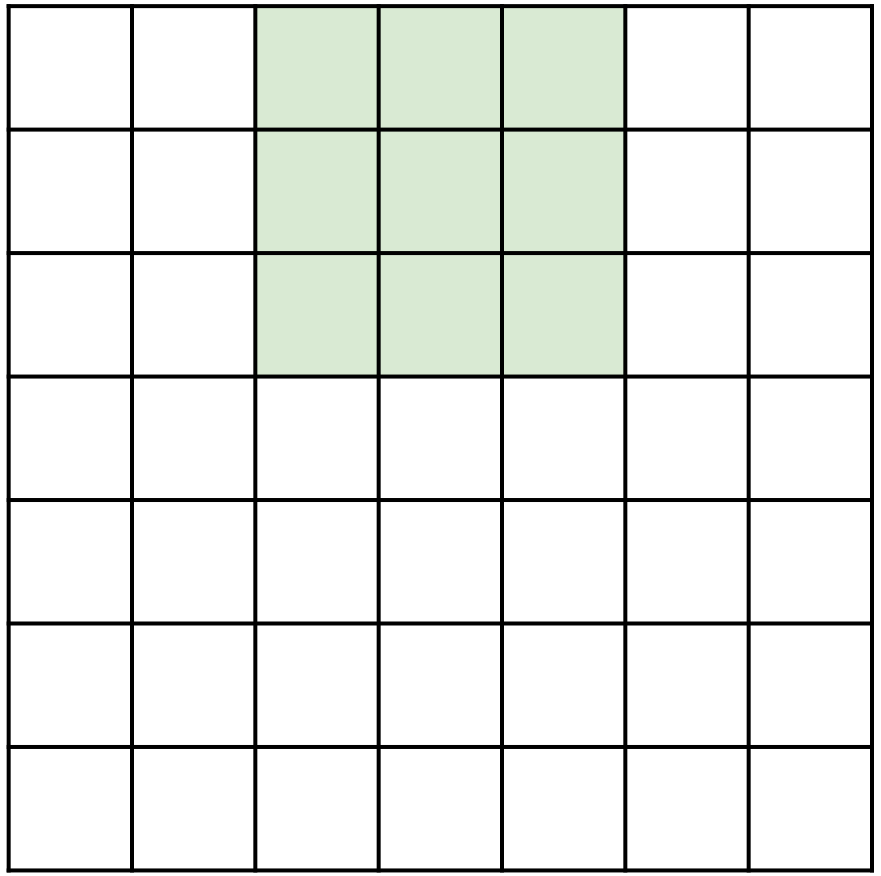
Input: 7x7
Filter: 3x3

A closer look at spatial dimensions



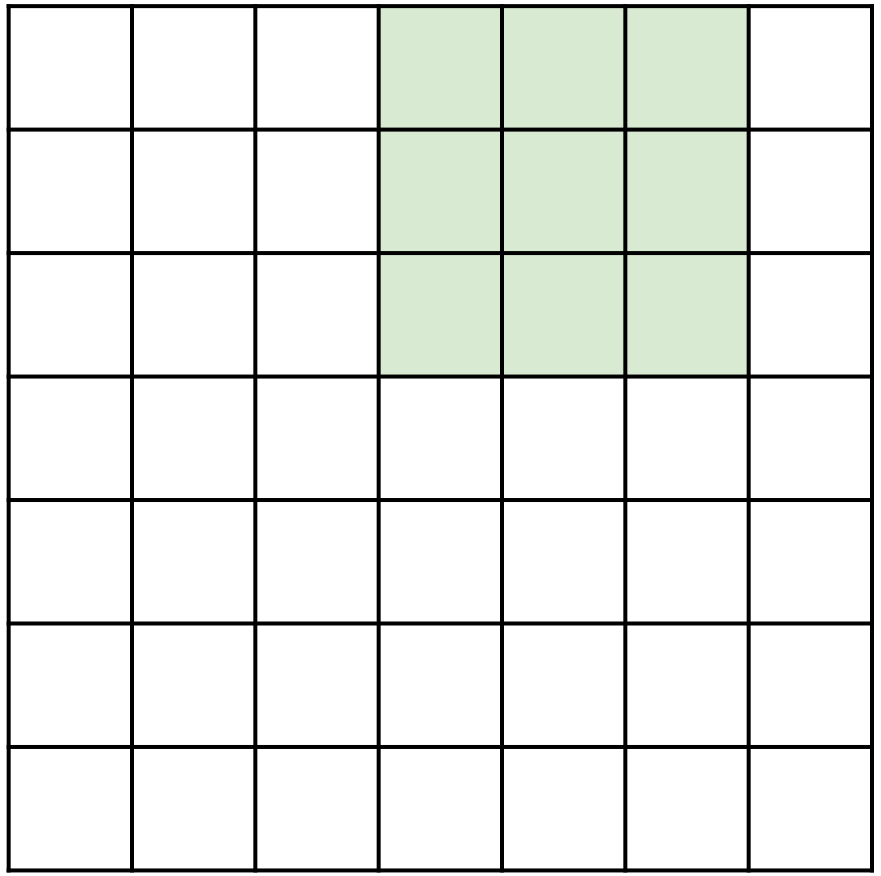
Input: 7x7
Filter: 3x3

A closer look at spatial dimensions



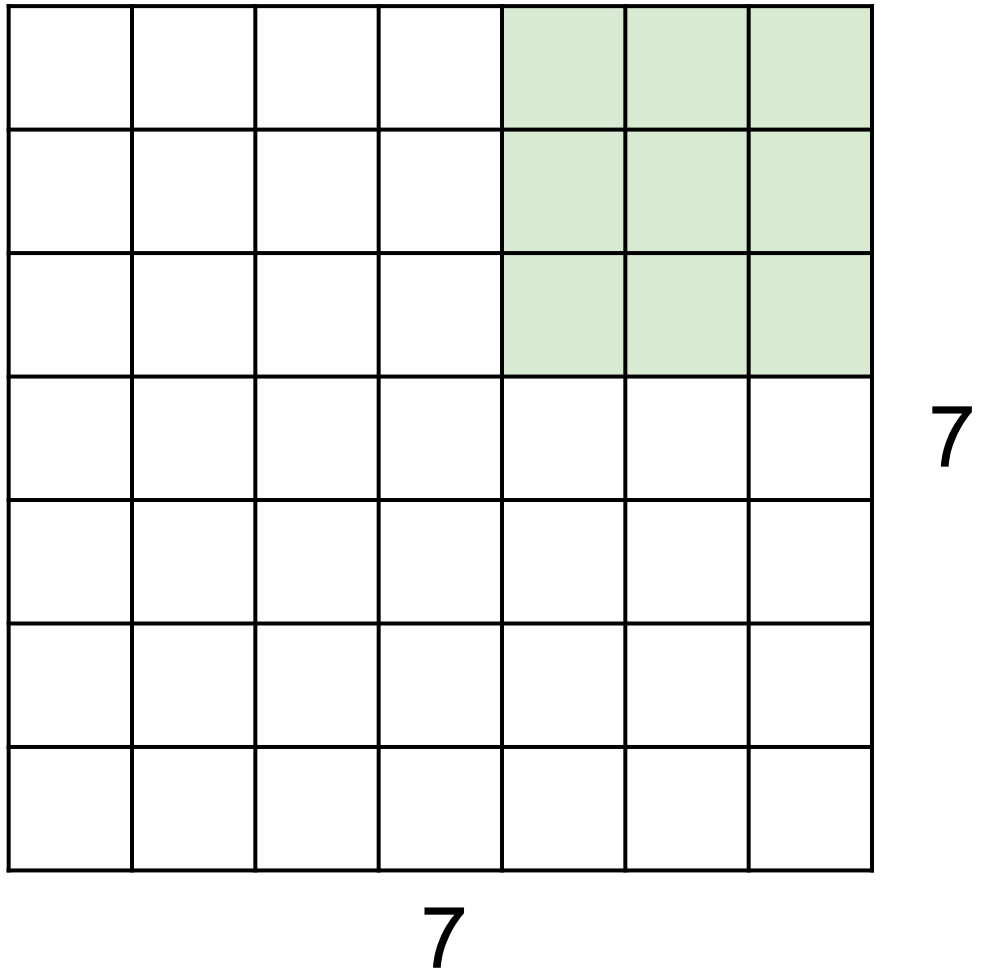
Input: 7x7
Filter: 3x3

A closer look at spatial dimensions



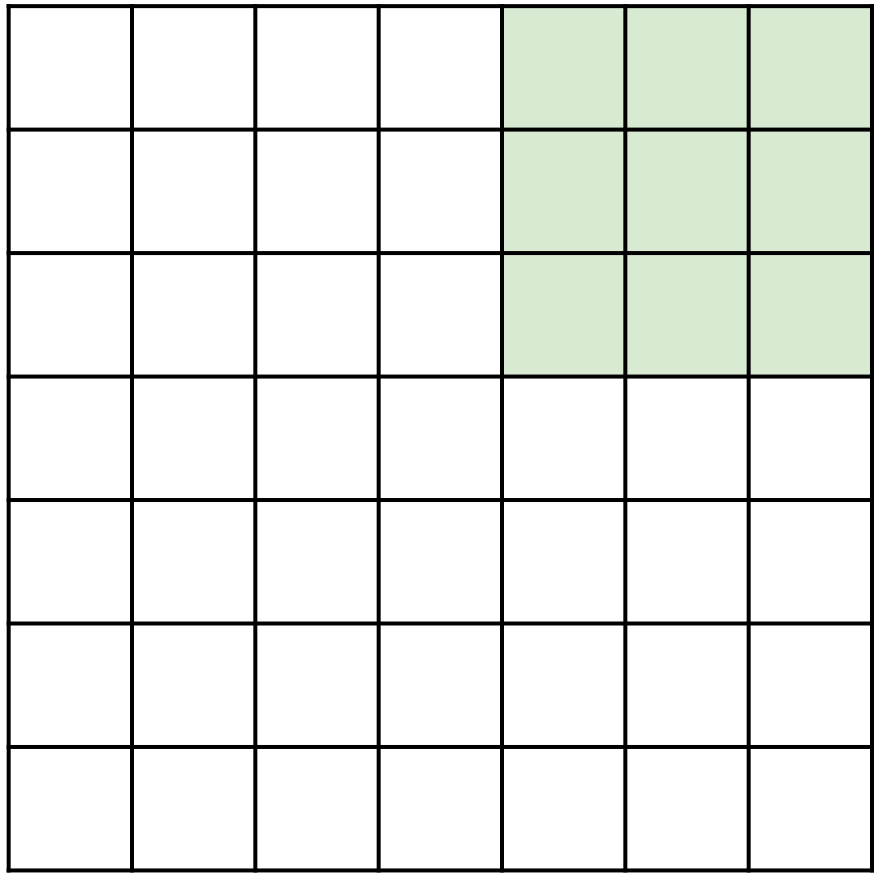
Input: 7x7
Filter: 3x3

A closer look at spatial dimensions



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

A closer look at spatial dimensions

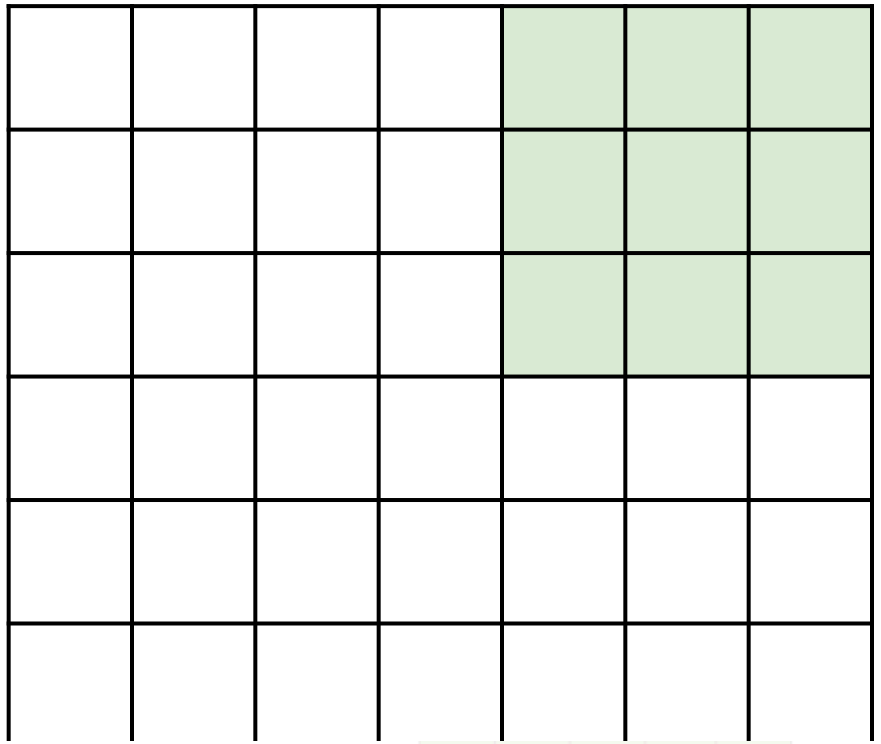


Input: 7x7
Filter: 3x3
Output: 5x5

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

Problem: Feature maps "shrink" with each layer!

A closer look at spatial dimensions



7

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

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

Problem: Feature maps "shrink" with each layer!

Problem: the output is smaller than the input!

- If the input is 28×28 and the filter is 3×3, you can have at most 13 convolutional layers. (The output of the 13^{th} layer is 2×2.)

Zero Padding

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

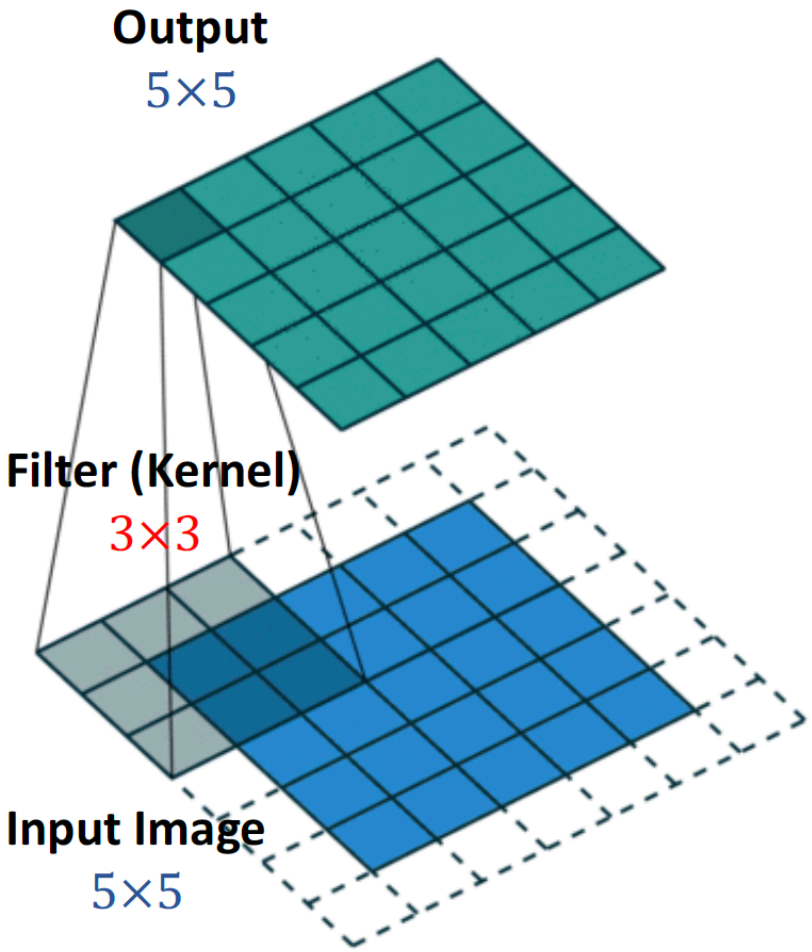
Input: 7x7
Filter: 3x3
Output: 5x5

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

Problem: Feature maps "shrink" with each layer!

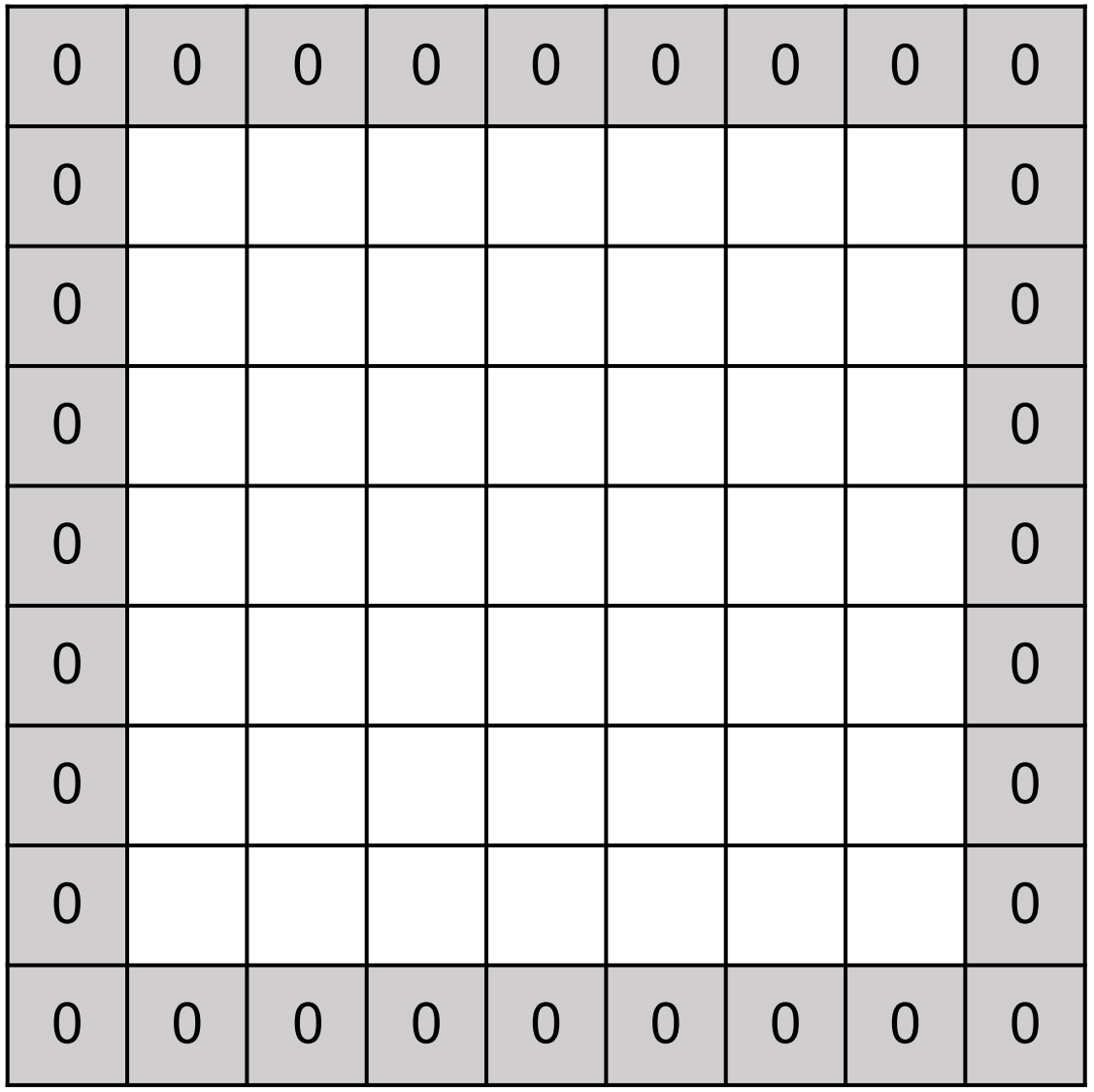
Solution: **padding**
Add zeros around the input

Zero Padding



- Add a “boarder” of all-zeros.
- Increase the input shape:
  - From $d_1 \times d_2$ to $(d_1 + 2) \times (d_2 + 2)$.
- If the filter is 3×3, the output is $d_1 \times d_2$.

Zero Padding



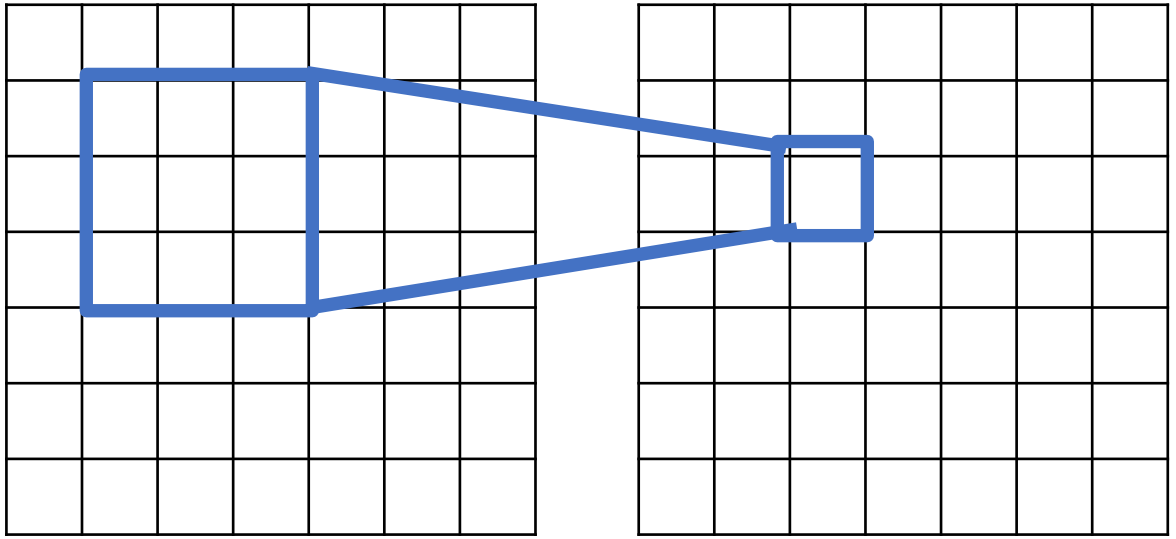
Input: 7x7
Filter: 3x3
Output: 5x5

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

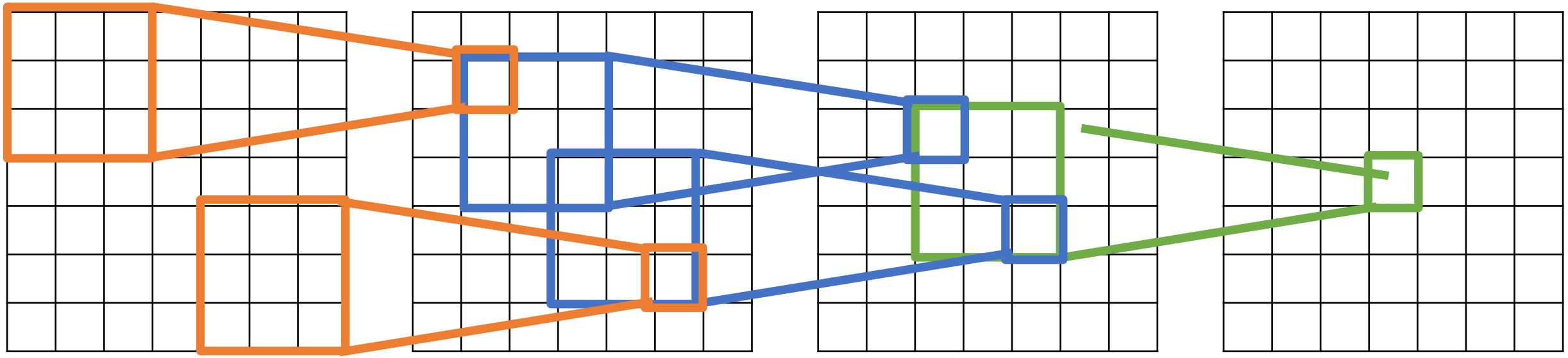
For convolution with kernel size K, each element in the output depends on a Kx K local **receptive field** in the input



Input Output

Receptive Fields

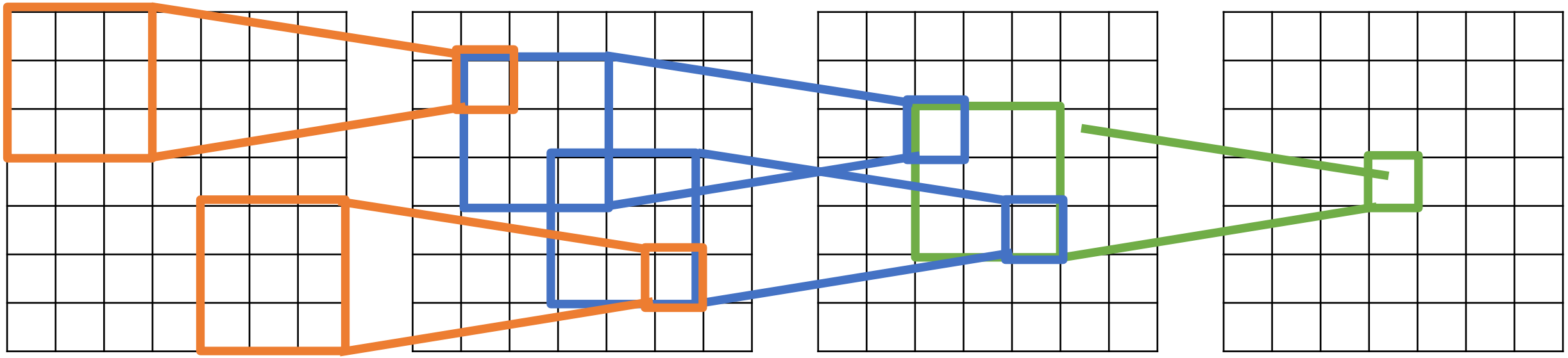
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L* (K – 1)



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)



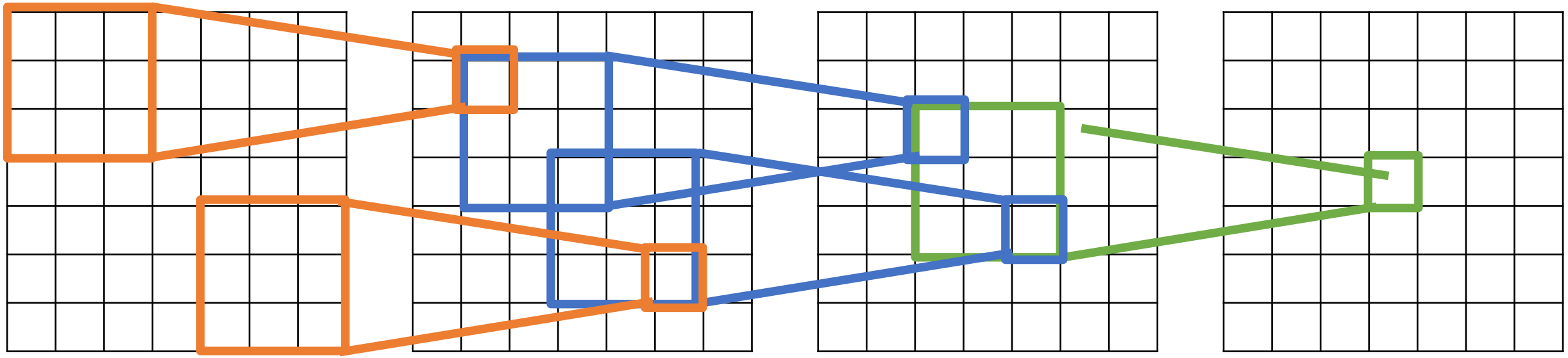
Input

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)



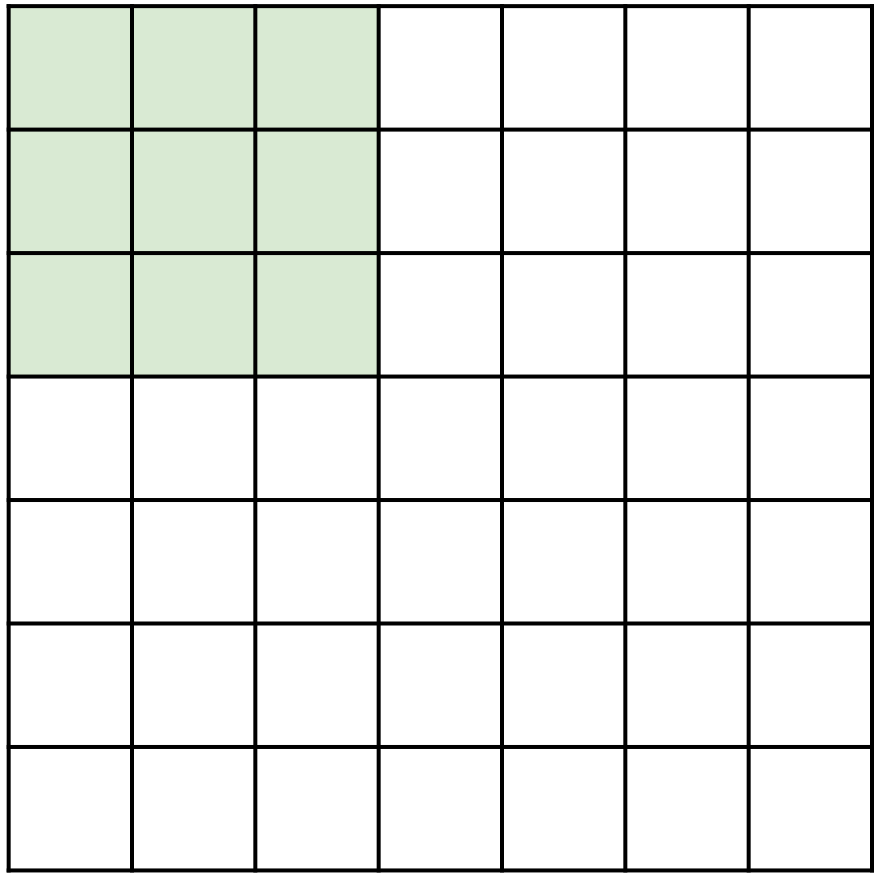
Input

Output

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

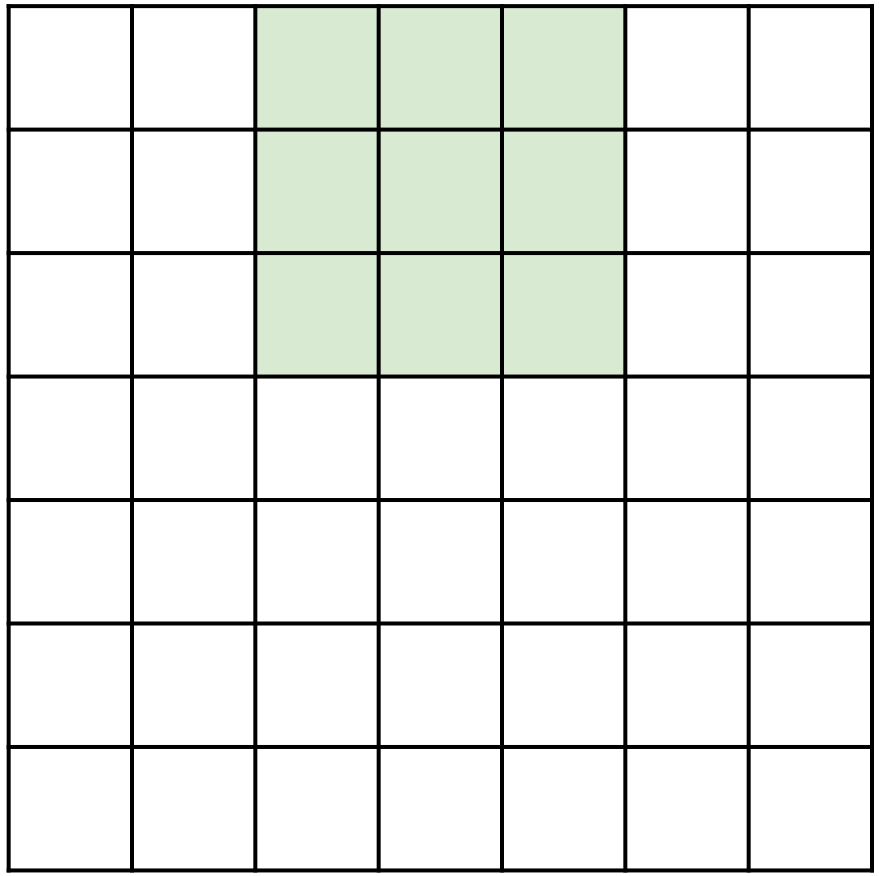
Solution: Downsample inside the network

Strided Convolution



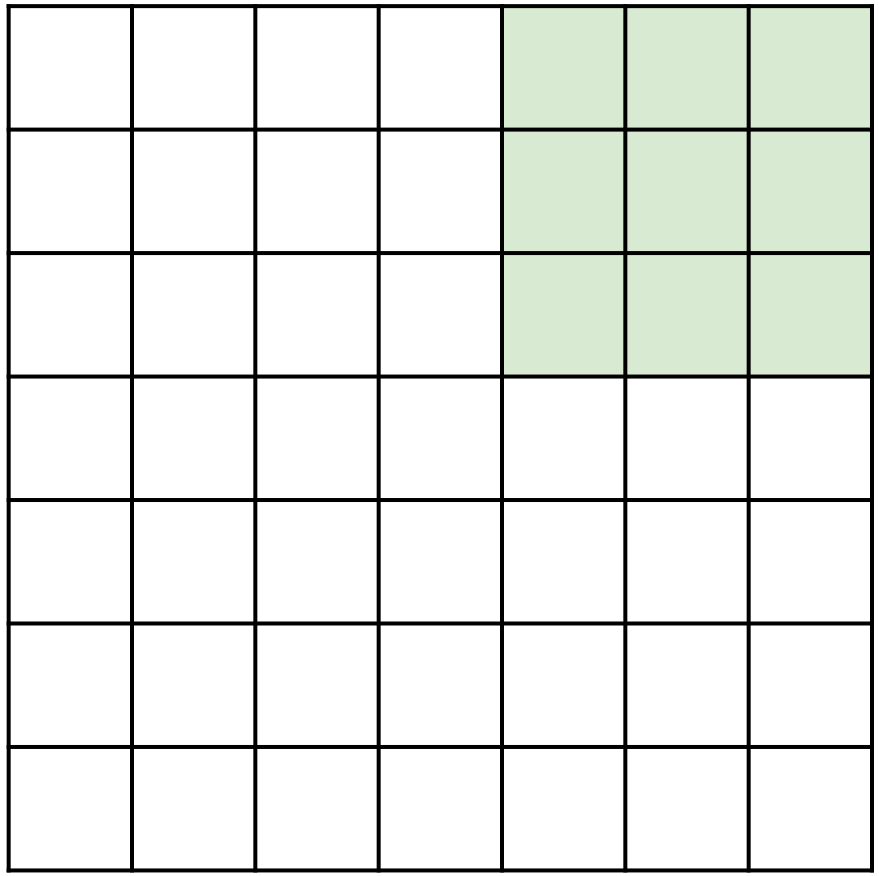
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

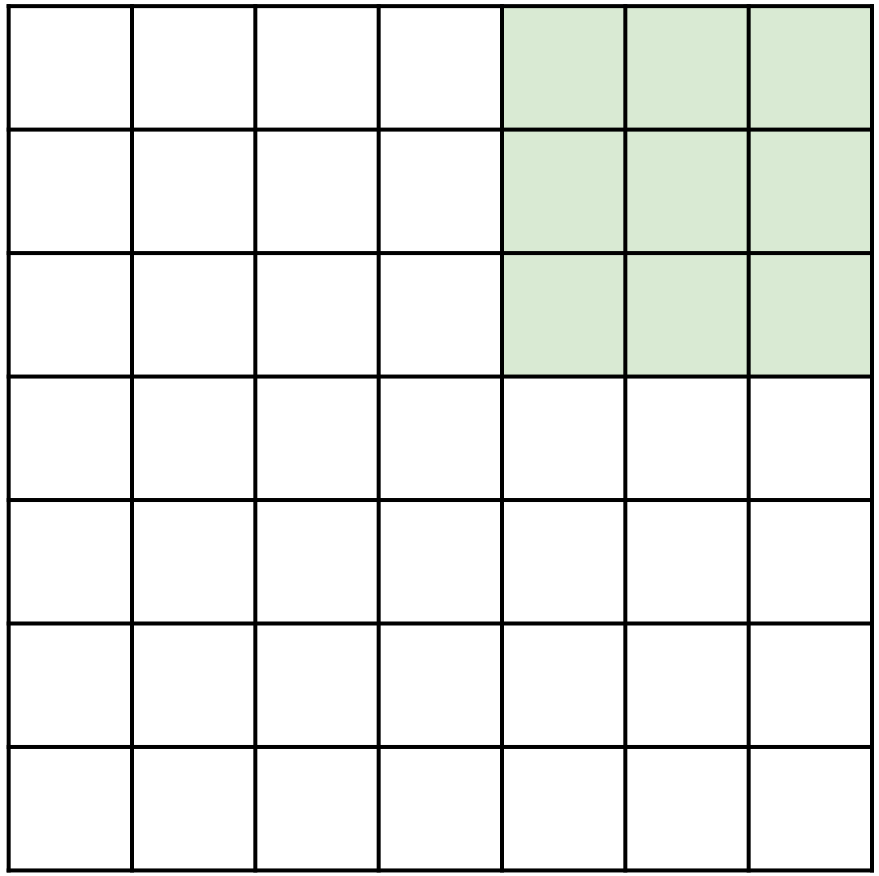
Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

Output: 3x3

Strided Convolution

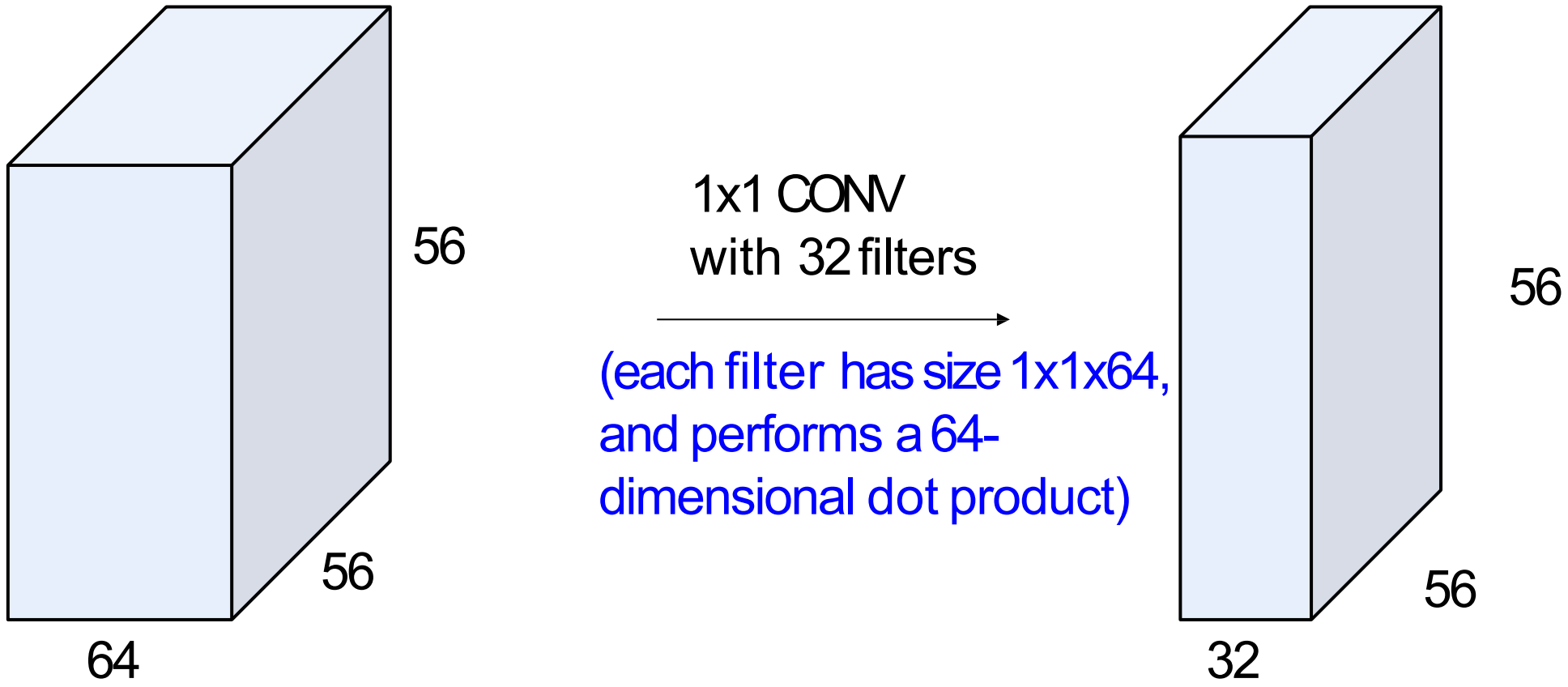


Input: 7x7
Filter: 3x3
Stride: 2

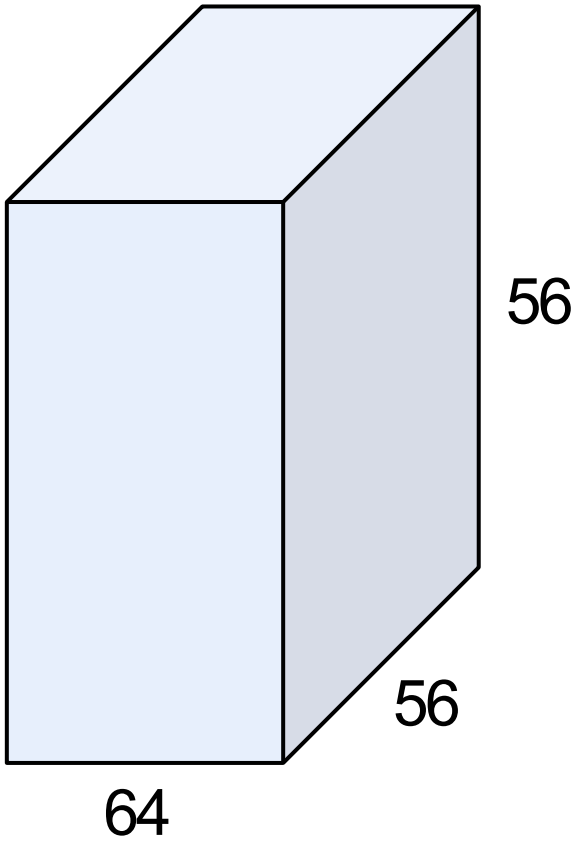
Output: 3x3

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

1x1 Convolution

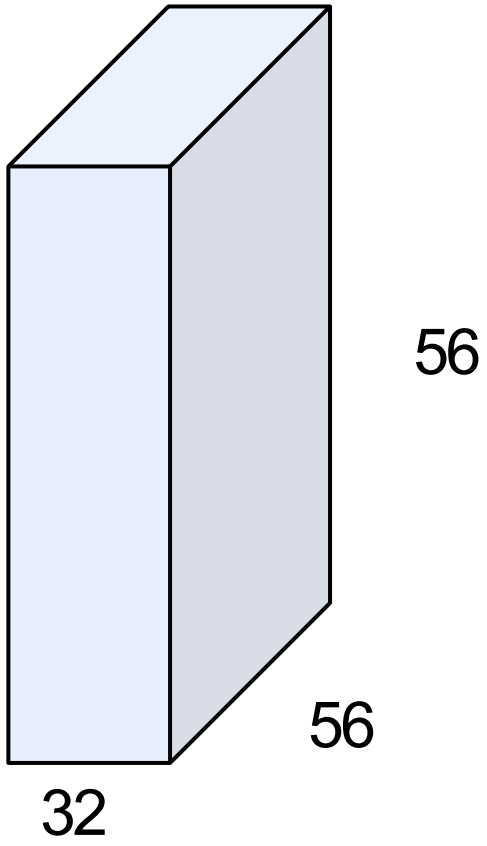


1x1 Convolution



1x1 CONV
with 32 filters

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



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

Lin et al, “Network in Network”, ICLR2014

Convolution Summary

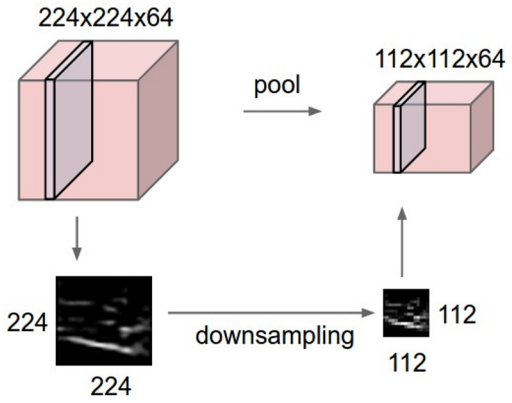
Input: C_{in} x H x W

Hyperparameters:

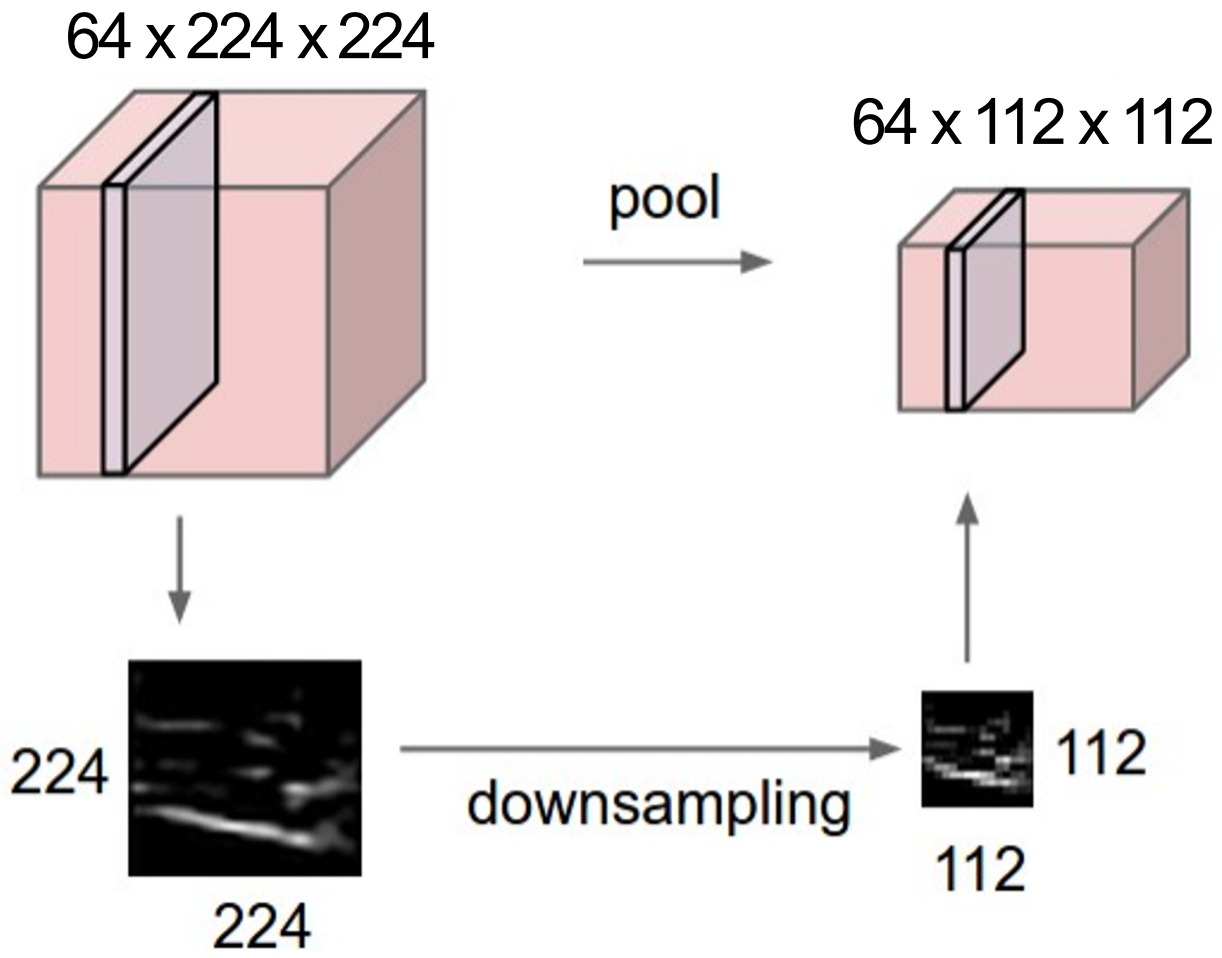
- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Components of a Convolutional Network

Pooling Layers



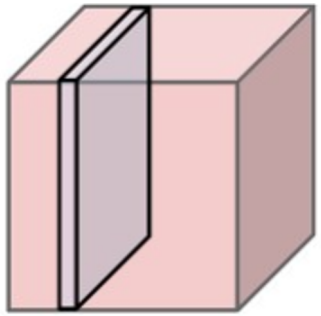
Pooling Layers: Another way to downsample



Hyperparameters:
Kernel Size
Stride
Pooling function

Max Pooling

64 x 224 x 224



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 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

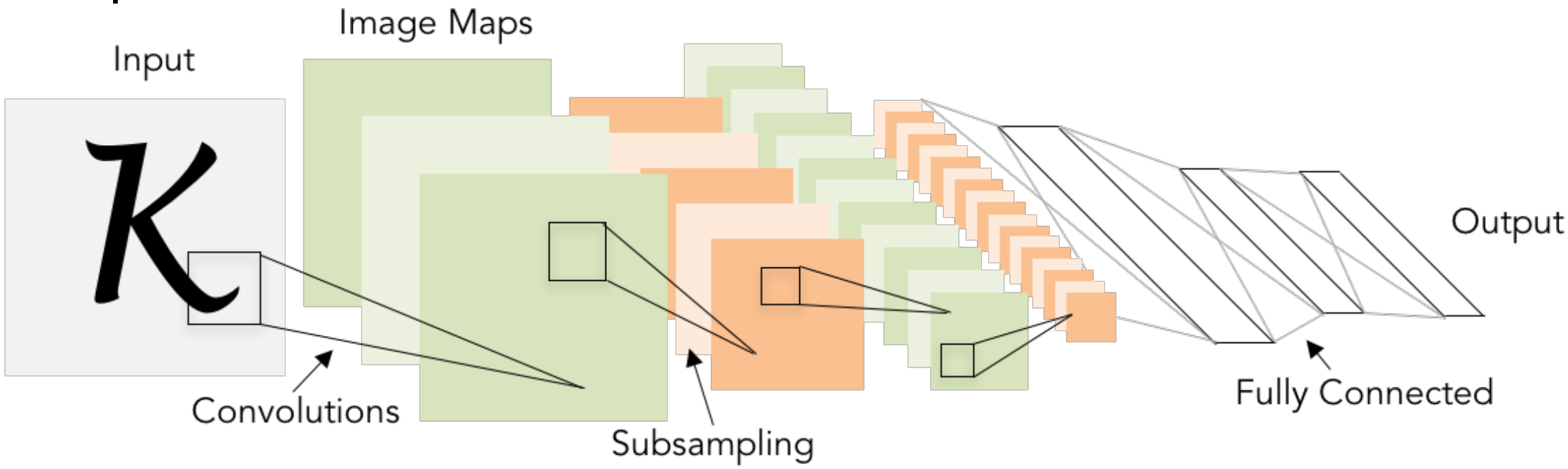
- H' = (H – K) / S + 1
- W' = (W – K) / S + 1

Learnable parameters: None!

Convolutional Networks

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

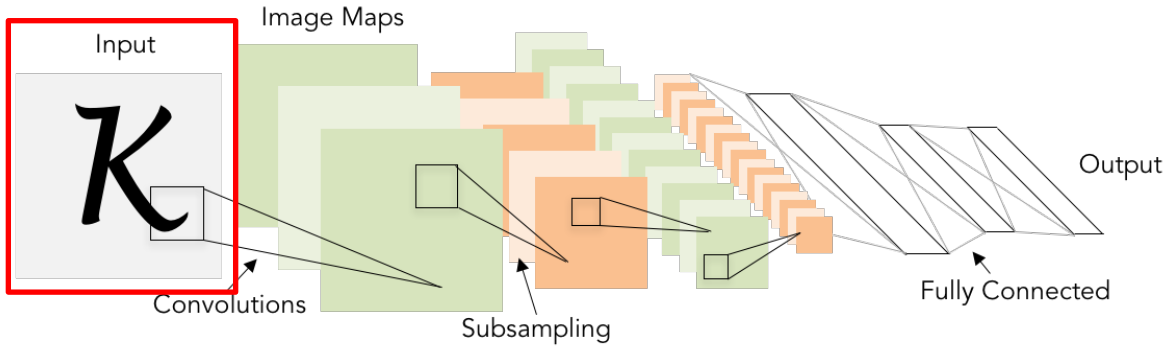
Example: LeNet-5



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

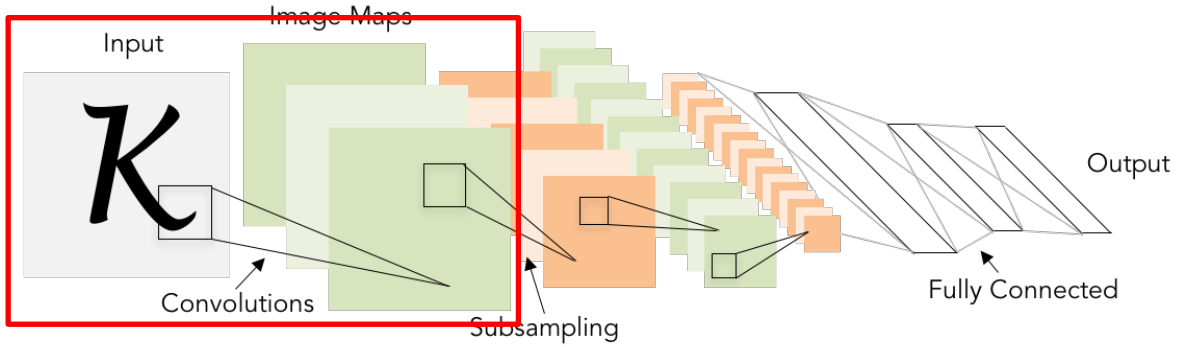
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	



Example: LeNet-5

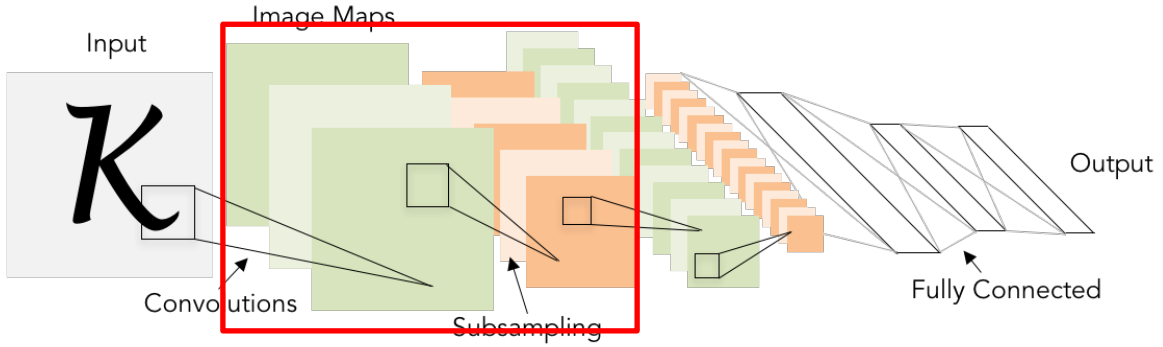
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	



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

Example: LeNet-5

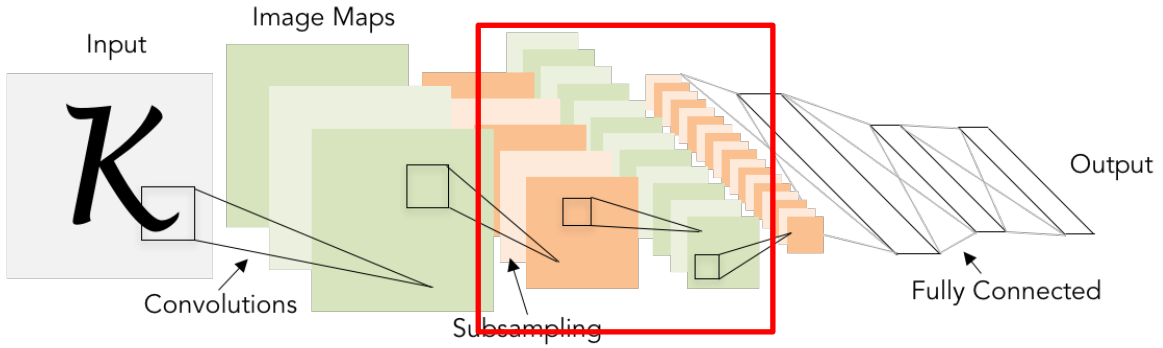
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	



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

Example: LeNet-5

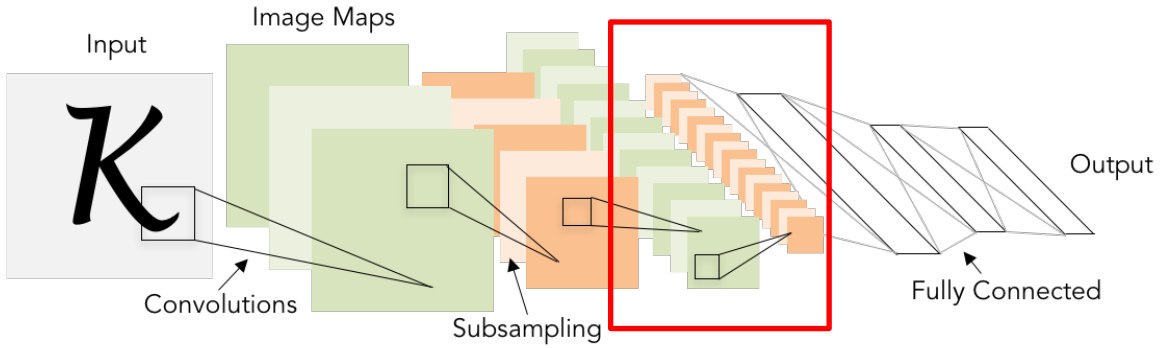
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	



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

Example: LeNet-5

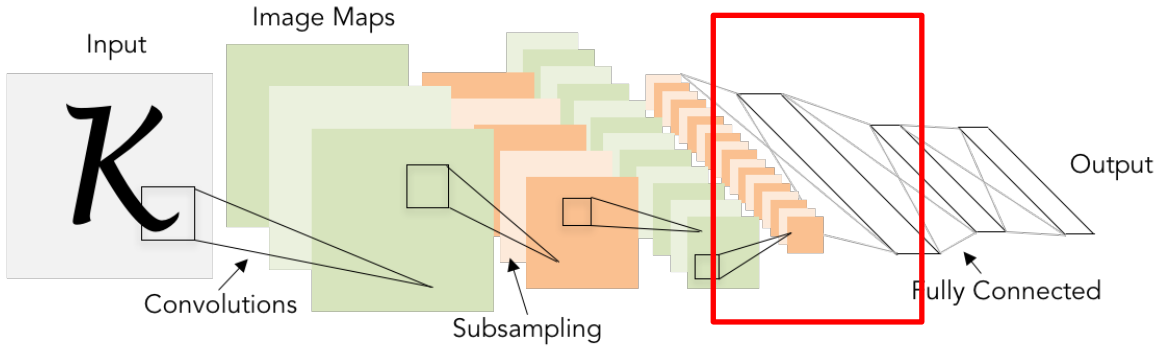
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	



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

Example: LeNet-5

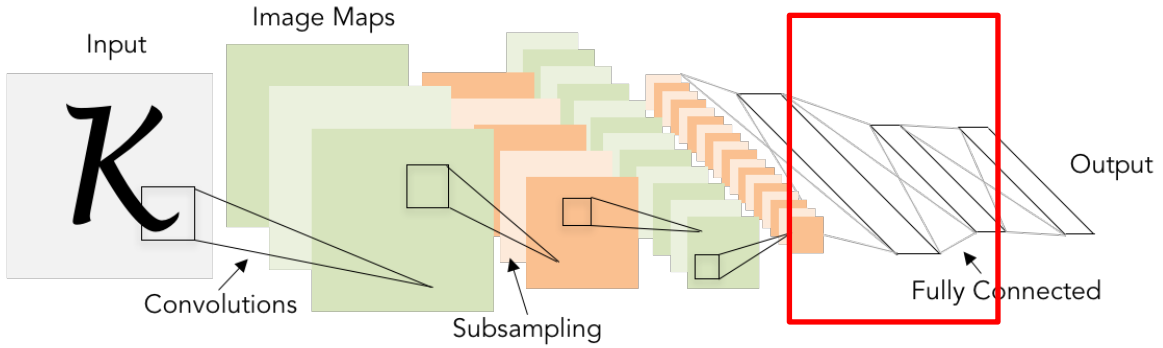
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	



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

Example: LeNet-5

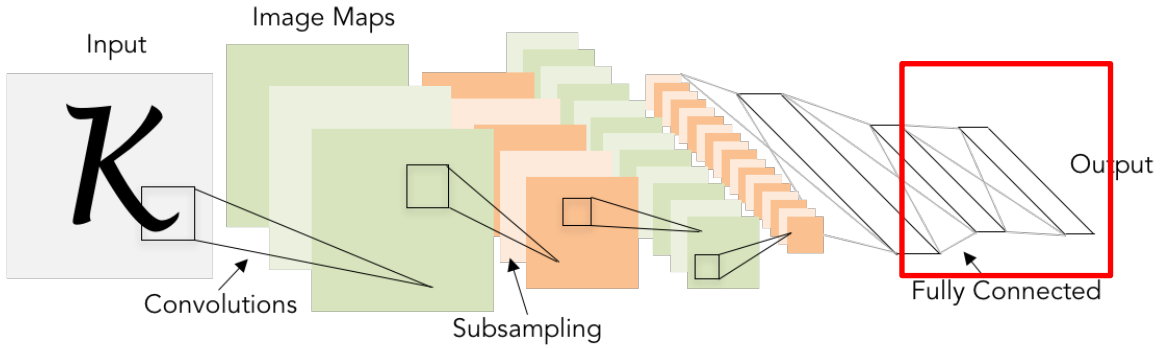
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 ->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



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

Example: LeNet-5

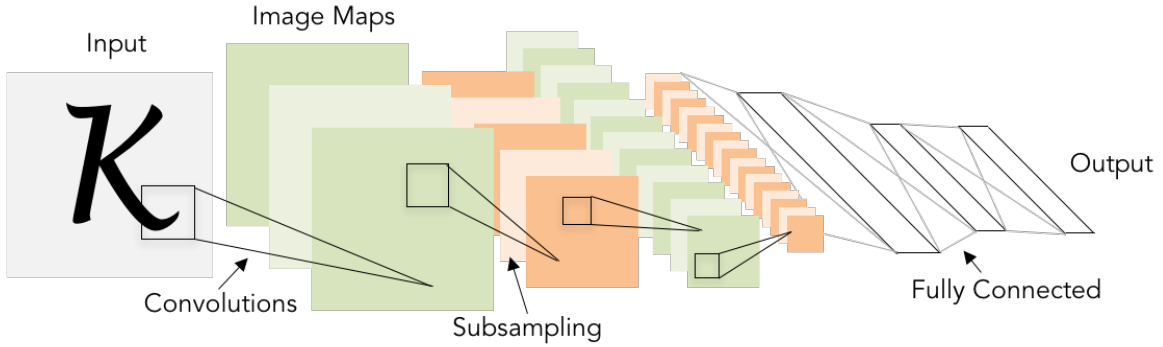
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 ->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



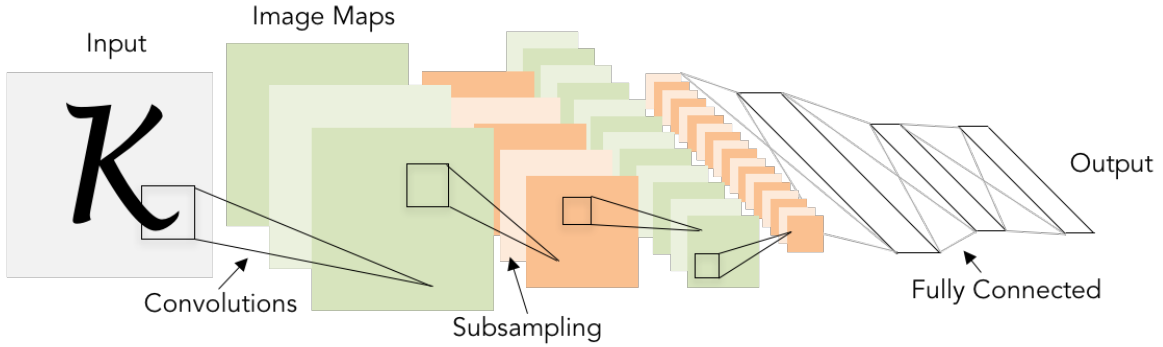
As we go through the network:

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

Hyperparameters for CNN



- Convolutional layers
  - # Filters
  - Filter shape
  - Stride
  - Zero-padding
- Pooling layers
  - MaxPool or AvgPool
  - Pool size
  - Pool stride
- Fully-connected layers
  - Width
- Activation functions
  - ReLU
  - SoftMax
  - Sigmoid

Batch Normalization

Batch Normalization: Standardization of Hidden Layers

Recall: we do normalization for the input of the DNN.

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

Why? Helps speed up the training.

Batch Normalization: Standardization of Hidden Layers

- Let $\mathbf{x}^{(k)} \in \mathbb{R}^d$ be the output of the k-th hidden layer.
- $\hat{\boldsymbol{\mu}} \in \mathbb{R}^d$: sample mean of $\mathbf{x}^{(k)}$ evaluated on a batch of samples.
- $\hat{\boldsymbol{\sigma}} \in \mathbb{R}^d$: sample std of $\mathbf{x}^{(k)}$ evaluated on a batch of samples.

- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.

Batch Normalization: Standardization of Hidden Layers

- Let $\mathbf{x}^{(k)} \in \mathbb{R}^d$ be the output of the k-th hidden layer.
- $\widehat{\boldsymbol{\mu}} \in \mathbb{R}^d$: sample mean of $\mathbf{x}^{(k)}$ evaluated on a batch of samples.
- $\widehat{\boldsymbol{\sigma}} \in \mathbb{R}^d$: sample std of $\mathbf{x}^{(k)}$ evaluated on a batch of samples.
- $\boldsymbol{\gamma} \in \mathbb{R}^d$: scaling parameter (trainable).
- $\boldsymbol{\beta} \in \mathbb{R}^d$: shifting parameter (trainable).
- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \widehat{\mu}_j}{\widehat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \cdot \gamma_j + \beta_j$, for $j = 1, \cdots, d$.

Batch Normalization: Standardization of Hidden Layers

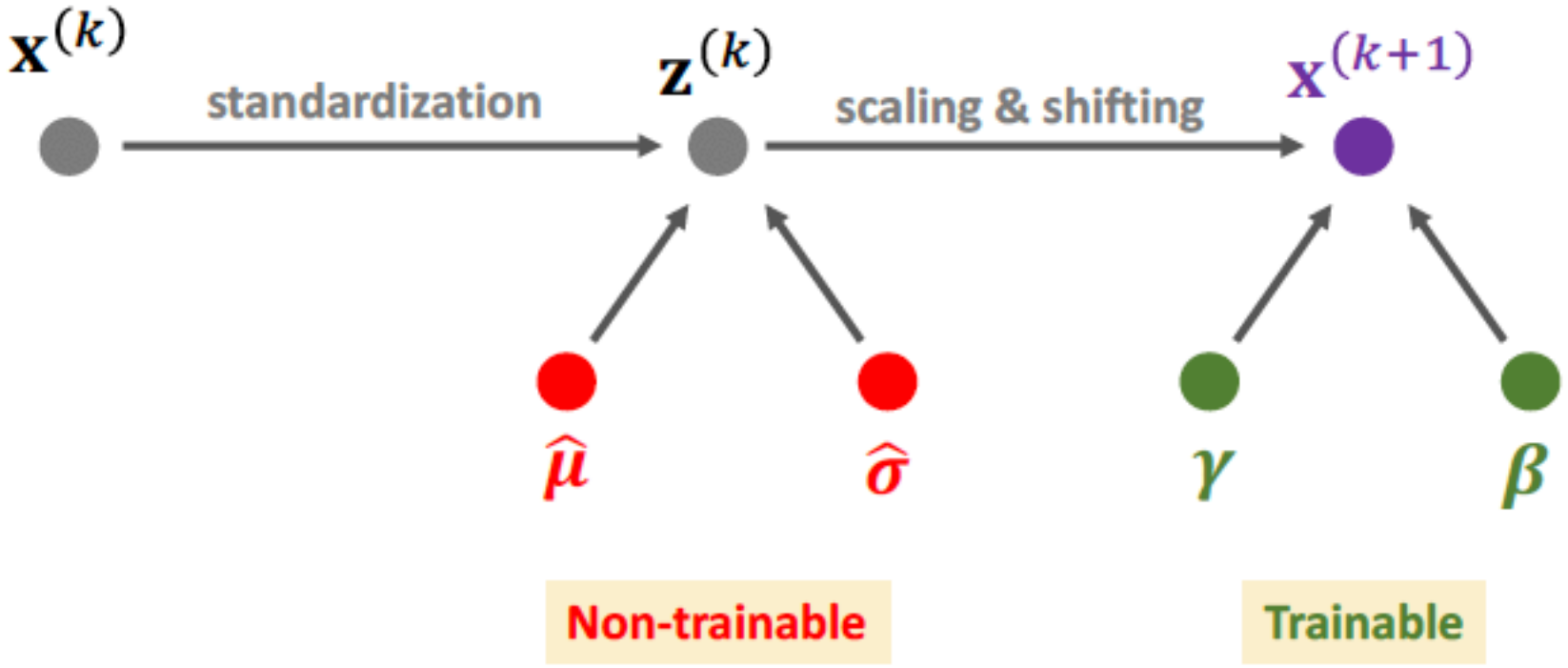
- Let $\mathbf{x}^{(k)} \in \mathbb{R}^d$ be the output of the k-th hidden layer.
- $\hat{\boldsymbol{\mu}} \in \mathbb{R}^d$:
- $\hat{\boldsymbol{\sigma}} \in \mathbb{R}^d$:

Non-trainable. Just record them in the forward pass; use them in the backpropagation.

- $\boldsymbol{\gamma} \in \mathbb{R}^d$: scaling parameter (trainable).
- $\boldsymbol{\beta} \in \mathbb{R}^d$: shifting parameter (trainable).
- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \cdot \gamma_j + \beta_j$, for $j = 1, \cdots, d$.

Batch Normalization: Standardization of Hidden Layers

- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \circ \gamma_j + \beta_j$, for $j = 1, \cdots, d$.



Batch Normalization

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

Input: $x \in \mathbb{R}^{N \times D}$

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^{D}$$

Learning $\gamma = \sigma, \beta = \mu$ will recover the identity function (in expectation)

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

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: $x \in \mathbb{R}^{N \times D}$

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

Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

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

Per-channel std, shape is D

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

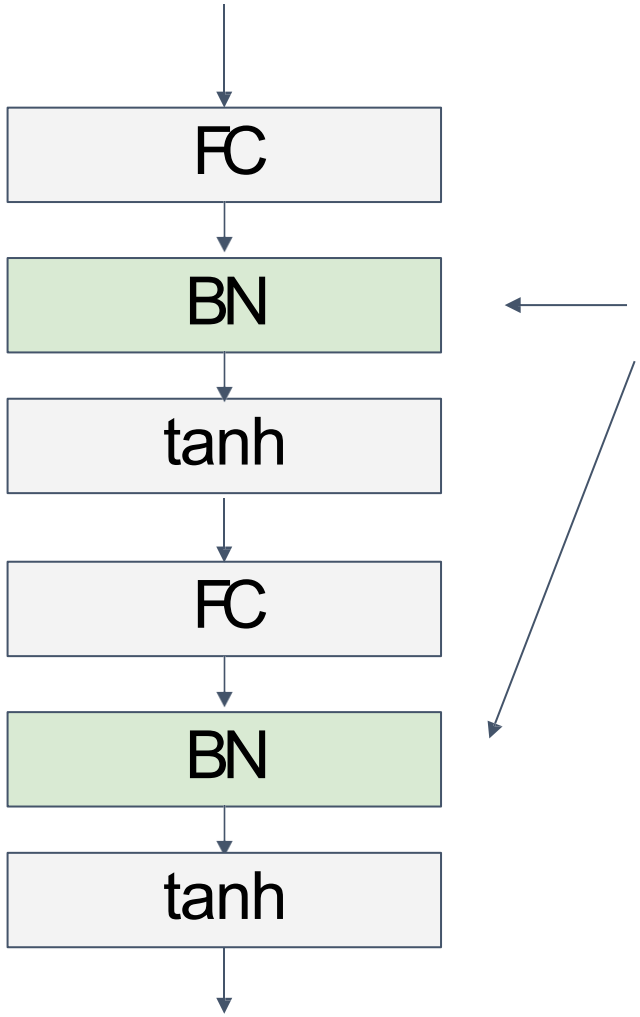
Normalized x, Shape is N x D

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

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

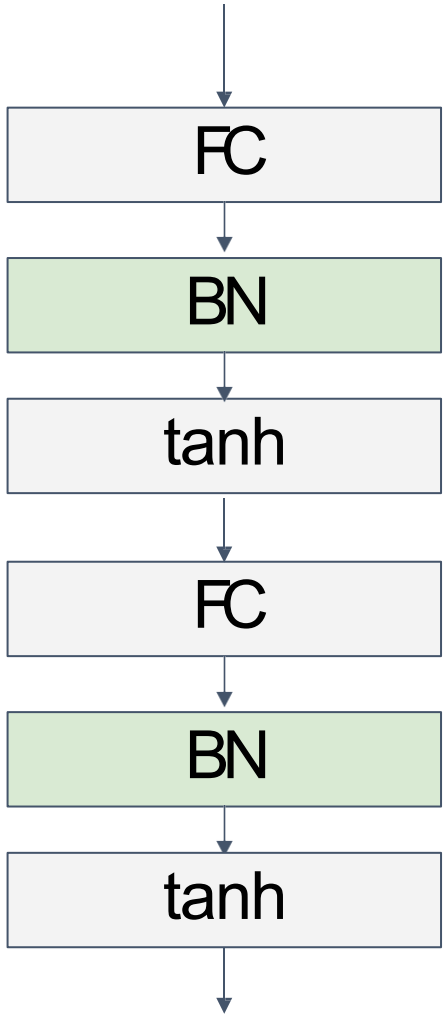
Output, Shape is N x D

Batch Normalization

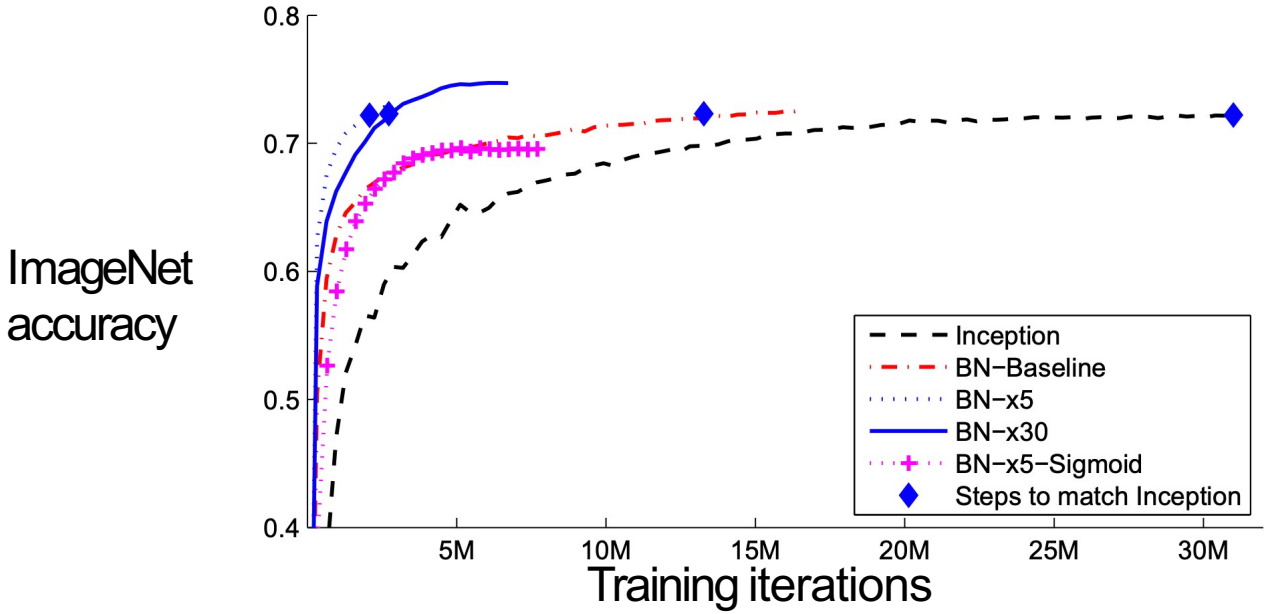


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

Batch Normalization



- 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!



Any Question ?