Convolutional Neural Networks

Changyou Chen

Department of Computer Science and Engineering
University at Buffalo, SUNY
changyou@buffalo.edu

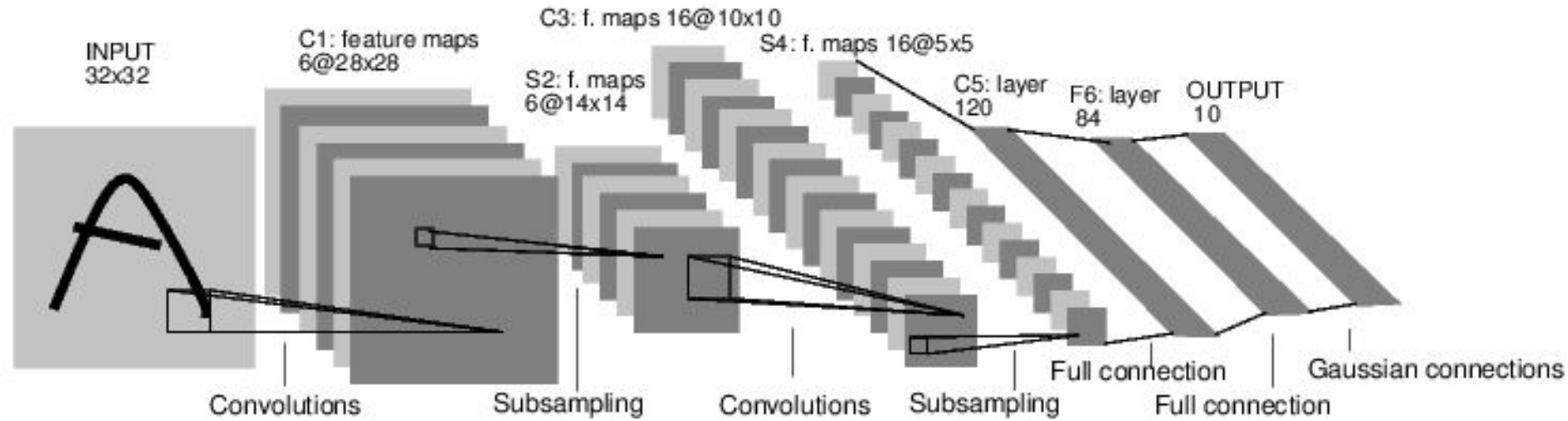
March 5, 2019

Introduction and History

1. Convolution as a specialized type of linear operation:
   - Sparse interaction
   - Parameter sharing
2. Convolutional neural networks (CNN) are simply neural networks that use convolution, instead of general matrix multiplication, in at least one of its layers.

Introduction and History

1. CNN has been introduced decades ago (1980's) by Lecun:
   - First applied on classifying handwritten digits, obtained state-of-the-art performance.
   - Fast development due to the success of computational power acceleration and neural network training algorithms.



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Convolutional Layers[1]

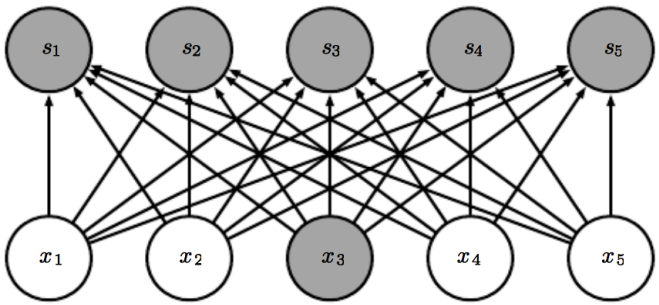
[1] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Convolutional Neural Networks

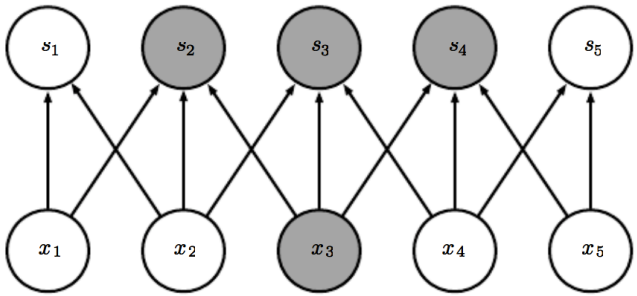
1. Scale up neural networks to process very large images/video sequences.
   - Sparse connections.
   - Parameter sharing.
2. Automatically generalize across spatial translations of inputs.
3. Applicable to any input that is laid out on a grid:
   - 1-D: signals
   - 2-D: gray images
   - 3-D: gray videos
   - . . .

Key Idea

1. Replace matrix multiplication in neural nets with convolution:
   - MLP uses matrix multiplication: with m inputs and n outputs, $m \times n$ parameters and $O(m \times n)$ runtime per example.
   - CNN uses convolutional with sparse interactions and parameter sharing: with k connections for each input, $k \times n$ parameters and $O(k \times n)$ runtime per example.



MLP



CNN

2. Everything else stays the same:
   - Maximum likelihood.
   - Back-propagation.
   - ...

Convolution on Continuous Domains

1-D convolution:

For two functions $f : \mathbb{R} \mapsto \mathbb{R}$ and $g : \mathbb{R} \mapsto \mathbb{R}$

$$(f * g)(t) \triangleq \int f(\tau) g(t - \tau) \mathrm{d}\tau$$

Extension to 2-D convolution:

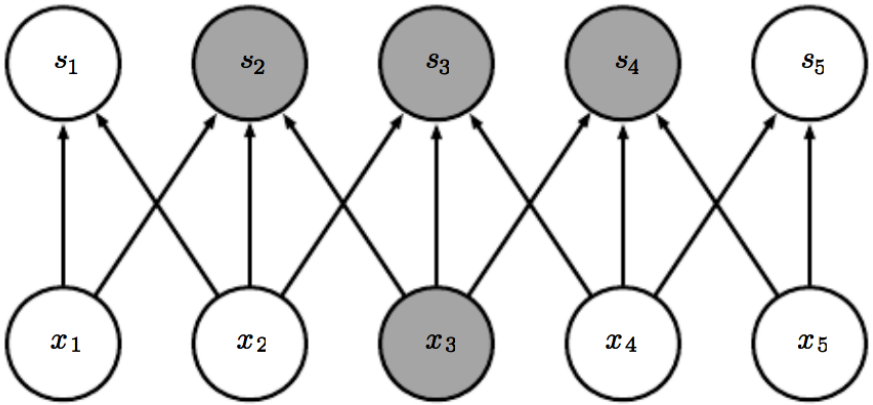
For two functions $f : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}$ and $g : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}$

$$(f * g)(t_1, t_2) \triangleq \int \int f(\tau_1, \tau_2) g(t_1 - \tau_1, t_2 - \tau_2) \mathrm{d}\tau_1 \mathrm{d}\tau_2$$

Convolution with Discrete Variables

1. Real data is usually presented in discrete domain.
2. Let **x** be a 1-dimensional input, **w** be a 1-dimensional kernel (filter), the output **s** is defined via the 1-D convolution:

$$\mathbf{s}(t) \triangleq (\mathbf{x} * \mathbf{w})(t) = \sum_{i=-\infty}^{\infty} \mathbf{x}(i)\,\mathbf{w}(t-i)$$



3. In ML applications,input is a multidimensional array of data, and the kernel is a multidimensional array of parameters to be learned.

Two-dimensional Convolution

- If we use a 2-D image **I** as input and use a 2-D kernel **K**, we have[2]

$$\mathbf{S}(i,j) \triangleq (\mathbf{I} * \mathbf{K})(i,j) = \sum_m \sum_n \mathbf{I}(m,n)\,\mathbf{K}(i-m, j-n)$$

0	1	3
1	2	2
0	0	0

$*$

0	**1**	**2**
2	2	0
0	1	2

$=$?

[2] Take the out-of-bound elements to be zero.

Two-dimensional Convolution

- If we use a 2-D image **I** as input and use a 2-D kernel **K**, we have[2]

$$\mathbf{S}(i,j) \triangleq (\mathbf{I} * \mathbf{K})(i,j) = \sum_m \sum_n \mathbf{I}(m,n)\,\mathbf{K}(i-m, j-n)$$

0	1	3
1	2	2
0	0	0

$*$

0	**1**	**2**
2	2	0
0	1	2

$=$

0	**0**	**1**
0	3	10
2	6	9

[2] Take the out-of-bound elements to be zero.

Commutativity of Convolution

1. Convolution is commutative:

$$\mathbf{S}(i,j) = \sum_m \sum_n \mathbf{I}(m,n)\,\mathbf{K}(i-m, j-n) = \sum_m \sum_n \mathbf{I}(i-m, j-n)\,\mathbf{K}(m,n)$$

2. Commutativity arises because we have flipped the kernel relative to the input:
   - As m increases, index to the input increases, but index to the kernel decreases.

Cross-Correlation

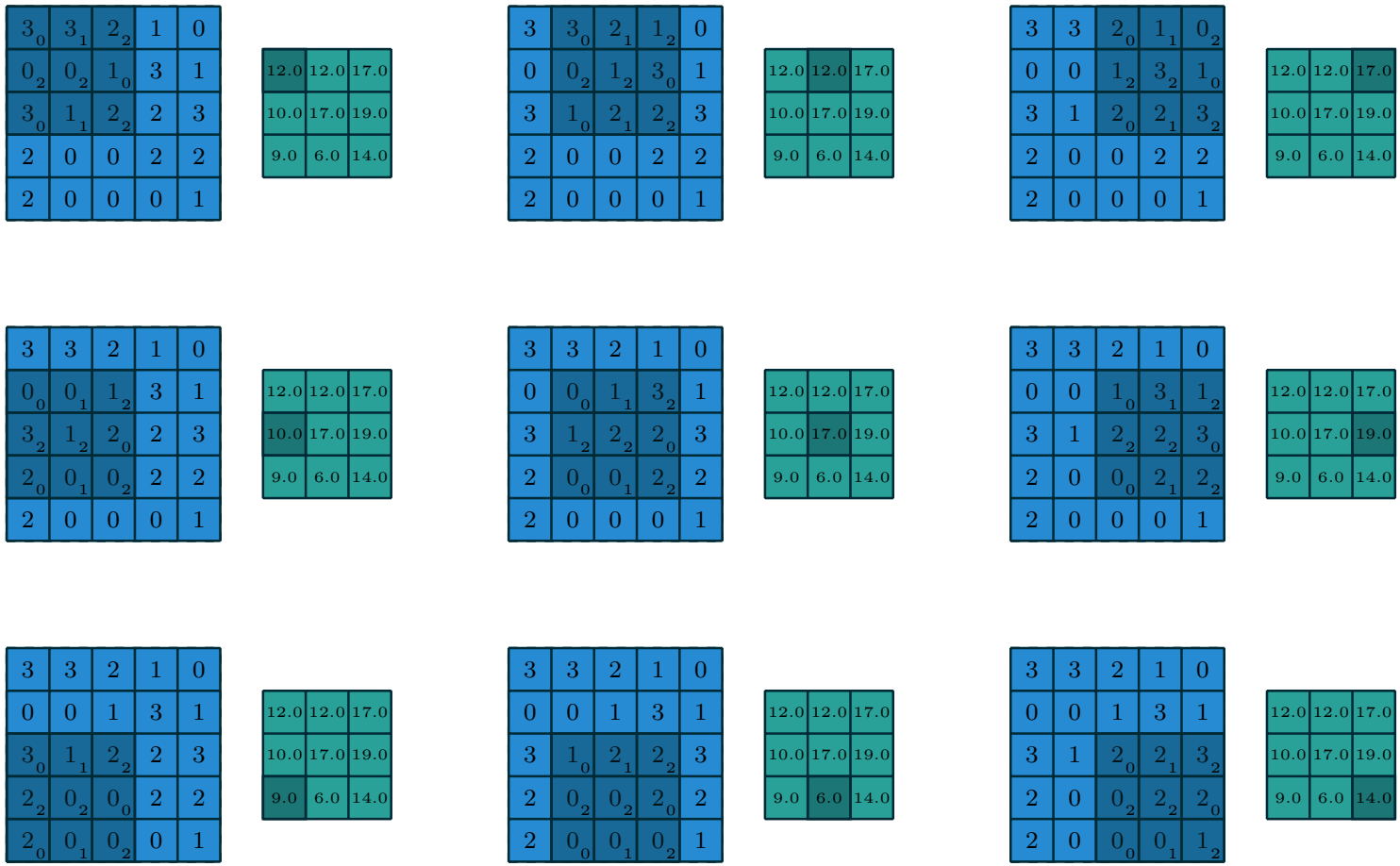
1. Cross-Correlation: same as convolution, but without flipping the kernel

$$\mathbf{S}(i,j) = \sum_m \sum_n \mathbf{I}(i+m, i+n)\, \mathbf{K}(m,n)$$

2. Both referred to as convolution, whether kernel is flipped or not.
3. In ML, cross-correlation is preferable because of computational convenience.
4. Generally, the two representations are equivalent if the kernel is learned.

Cross-Correlation Convolution Examples

$$\mathbf{S}(i,j) = \sum_m \sum_n \mathbf{I}(i+m, i+n)\, \mathbf{K}(m,n)$$

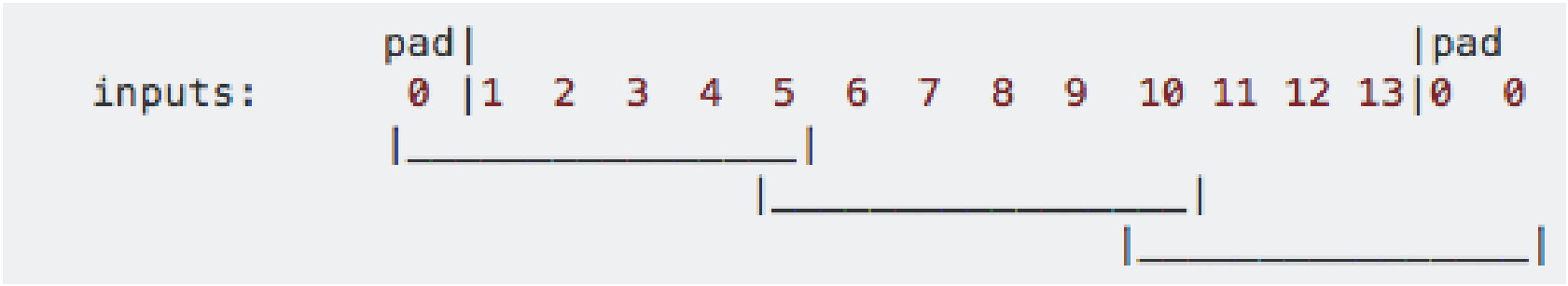


Padding

- Augment the input with zeros on the boundary.
- Padding can make the size of output equal to the size of input.

`"VALID"` = without padding:

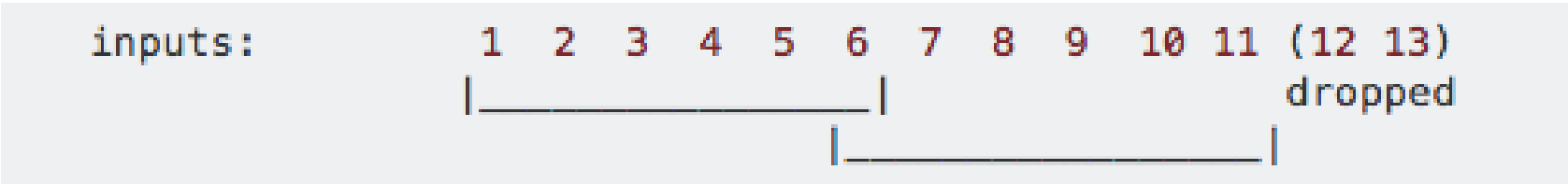
`"SAME"` = with zero padding:



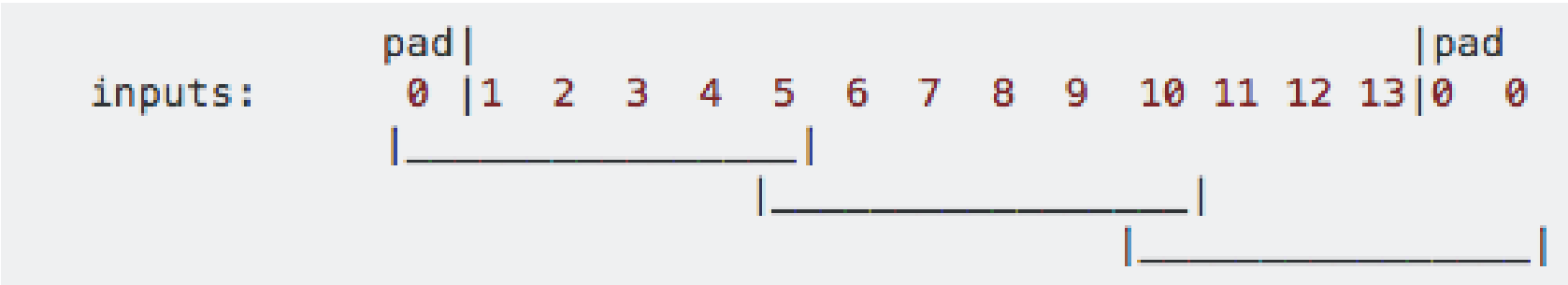
Padding

- Augment the input with zeros on the boundary.
- Padding can make the size of output equal to the size of input.

`"VALID"` = without padding:



`"SAME"` = with zero padding:



General rules for the output sizes and padding? $\Rightarrow$ later

Convolution with Strides

- Convolution with skipped indexes.
- Convolution with stride can be viewed as a way of doing dimensional reduction.

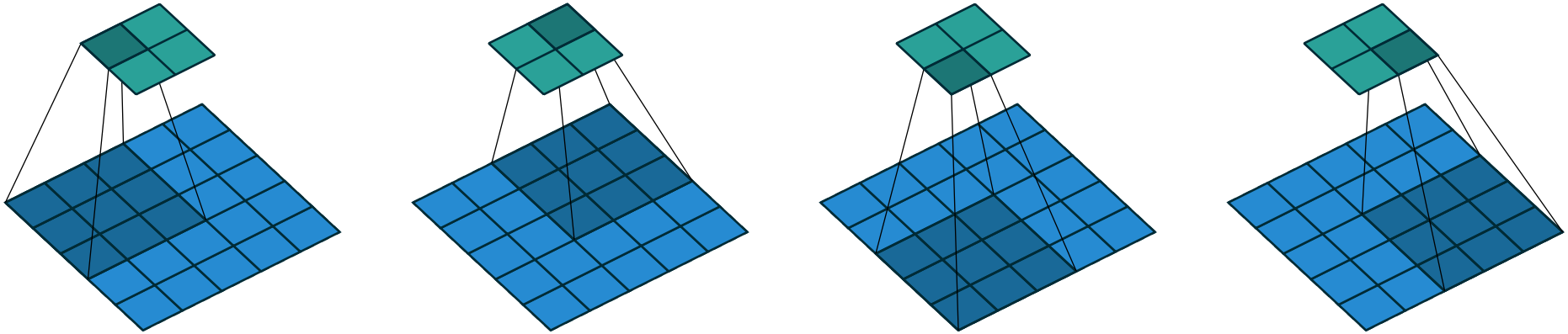


Figure: Convolution with stride of size 2×2 (sometimes simply say "stride 2"). 5×5 input, 2×2 output.

Convolution with Strides

- Convolution with skipped indexes.
- Convolution with stride can be viewed as a way of doing dimensional reduction.

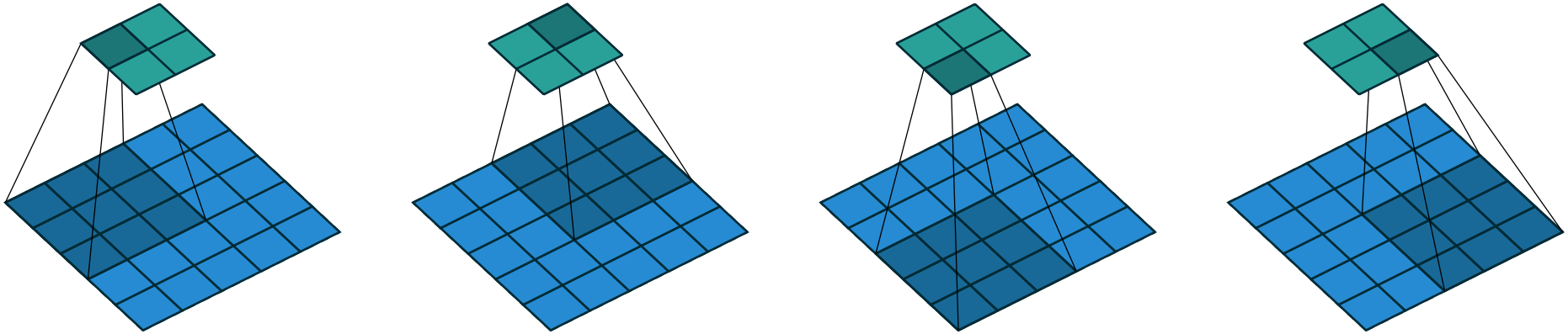


Figure: Convolution with stride of size 2×2 (sometimes simply say "stride 2"). 5×5 input, 2×2 output.

Relation of output size w.r.t. stride?

Convolution with Strides

- Convolution with skipped indexes.
- Convolution with stride can be viewed as a way of doing dimensional reduction.

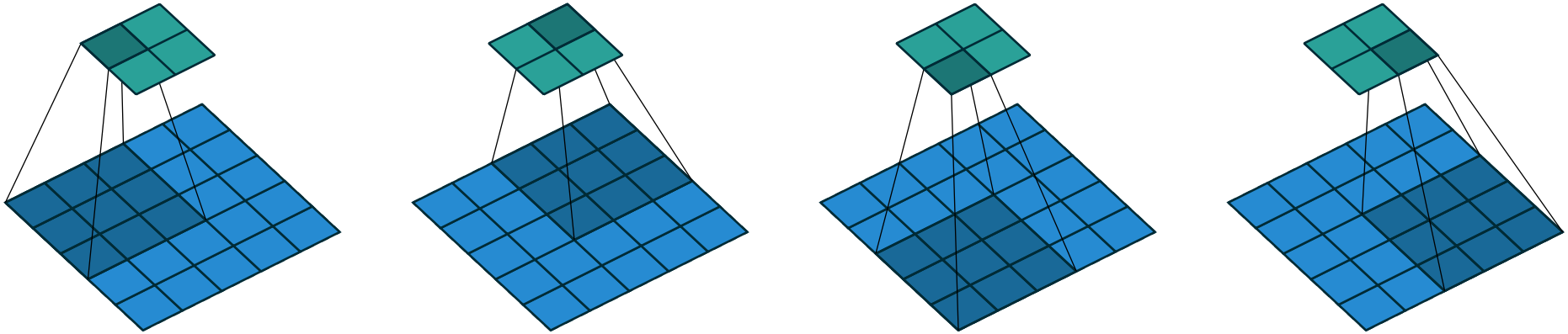
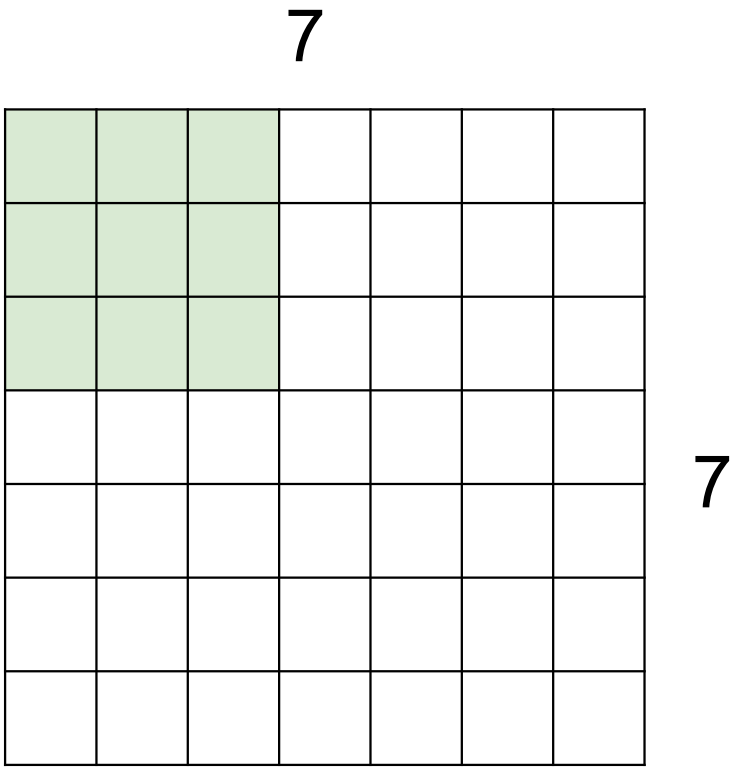


Figure: Convolution with stride of size 2×2 (sometimes simply say "stride 2"). 5×5 input, 2×2 output.

Relation of output size w.r.t. stride

output size = (input size - filter size) / stride + 1

A Closer Look at Strides

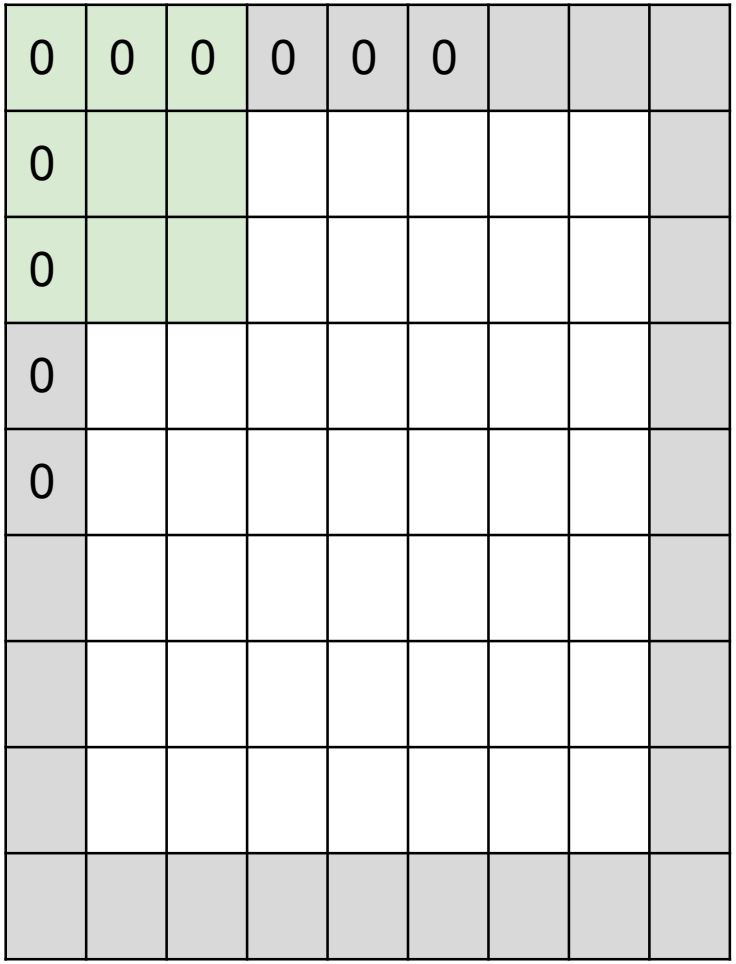


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

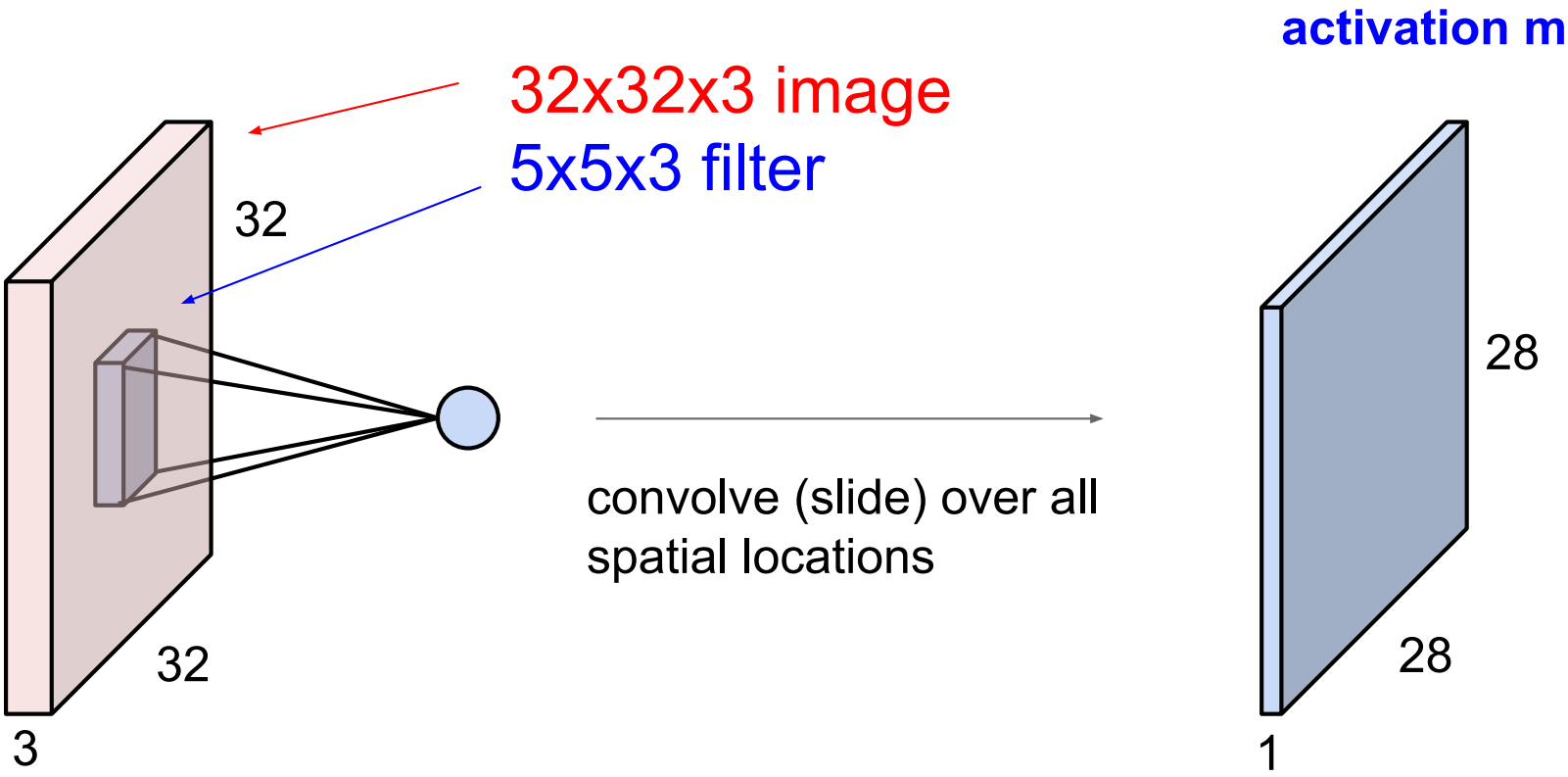
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Practical Setting

- In practice, common to set convolutional layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$:
  - will preserve input size



Convolution Layer

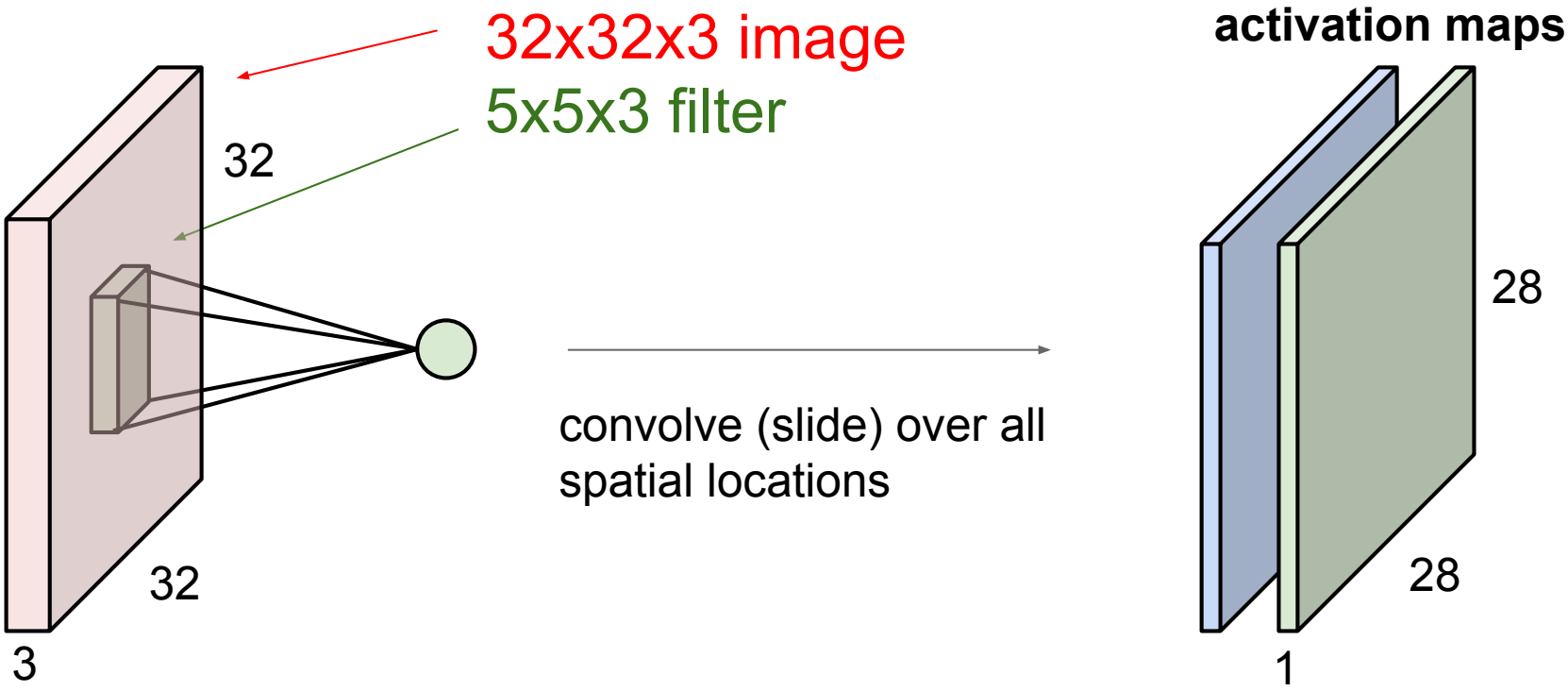


- Convolution and sum over the third dimension.

Extension to *n*-dimensional tensors

Convolution Layer

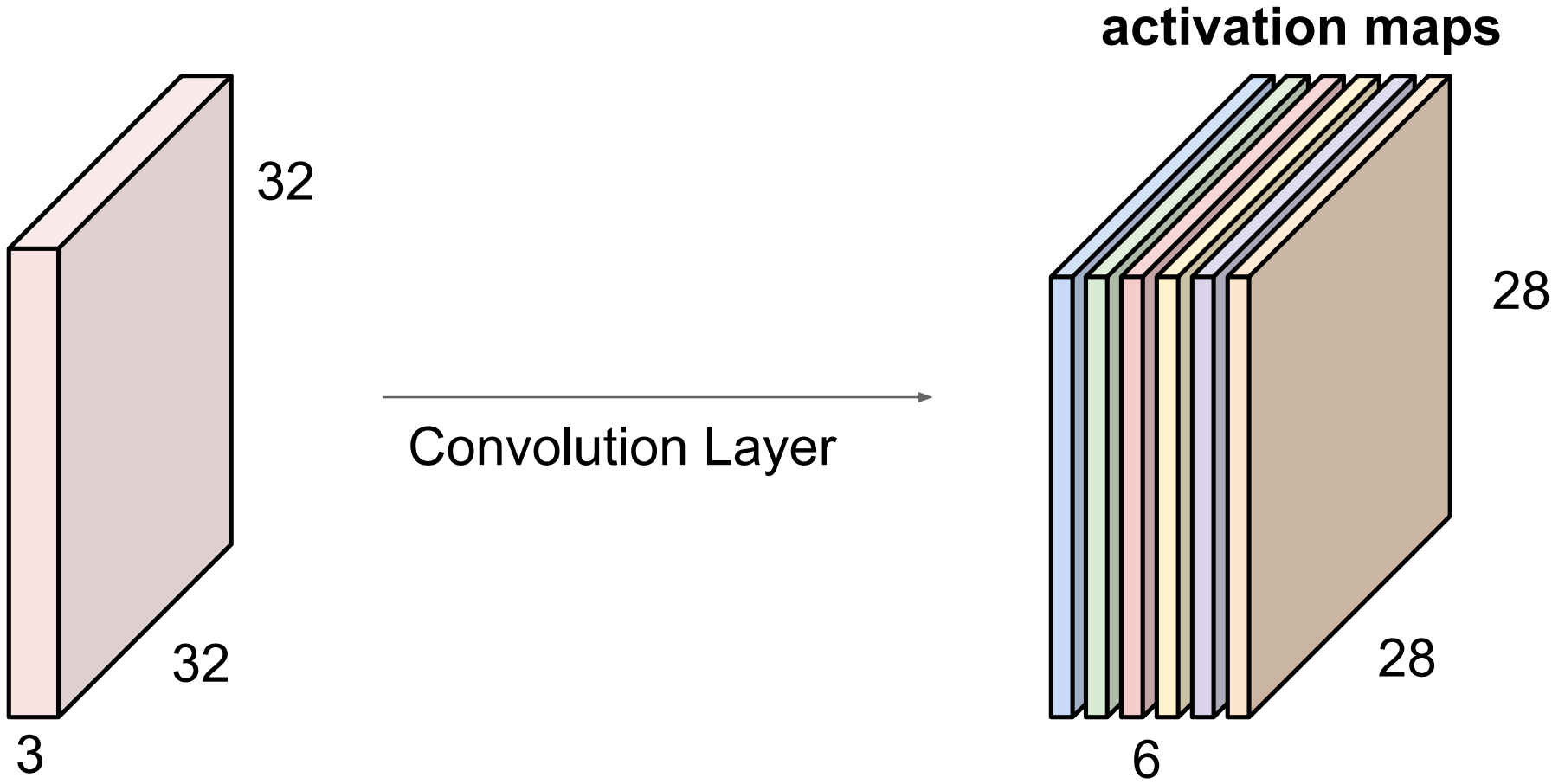
consider a second, green filter



- Convolution and sum over the third dimension.

Extension to n-dimensional tensors

If we had six $5 \times 5 \times 3$ filters[a], we'll get 6 separate activation maps.



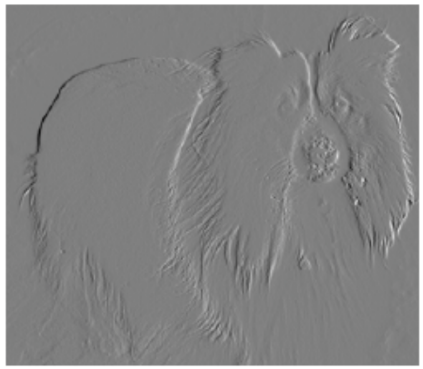
We stack these up to get a "new image" of size 28x28x6!

[a] We sometimes ignore the last dimension for simplicity, written as filter size of 5×5.

Example: Efficiency of Convolution for Edge Detection

1. Image on right formed by taking each pixel of input image and subtracting the value of its neighboring pixel on the left:
   - this is a measure of all the vertically oriented edges in input image, useful for object detection.
   - can be implemented as a convolution with a 1×2 kernel: $K = (1, -1)$.

Input image



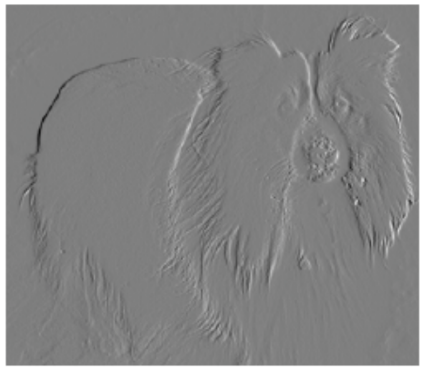
Both images are 280 pixels tall
Input image is 320 pixels wide
Output image is 319 pixels wide

2. Number of operations:

Example: Efficiency of Convolution for Edge Detection

1. Image on right formed by taking each pixel of input image and subtracting the value of its neighboring pixel on the left:
   - this is a measure of all the vertically oriented edges in input image, useful for object detection.
   - can be implemented as a convolution with a 1×2 kernel: $K = (1, -1)$.

Input image



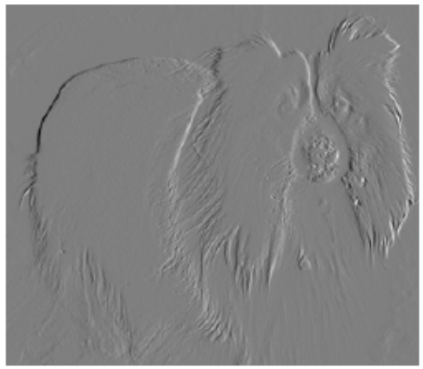
Both images are 280 pixels tall
Input image is 320 pixels wide
Output image is 319 pixels wide

2. Number of operations:
   - Convolution with one kernel of size 1×2:
   - MLP:

Example: Efficiency of Convolution for Edge Detection

1. Image on right formed by taking each pixel of input image and subtracting the value of its neighboring pixel on the left:
   - this is a measure of all the vertically oriented edges in input image, useful for object detection.
   - can be implemented as a convolution with a 1×2 kernel: $K = (1, -1)$.

Input image



Both images are 280 pixels tall
Input image is 320 pixels wide
Output image is 319 pixels wide

2. Number of operations:
   - Convolution with one kernel of size 1×2: $319 \times 280 \times 1 \times 2$.
   - MLP: $320 \times 280 \times 319 \times 280$.

Output Volumn Size w.r.t. Padding and Stride

- Input volume: 1×1, 5×5 filter with stride 1, pad 2.
- Output volume size?
  - 1×1

Output Volumn Size w.r.t. Padding and Stride

- Input volume: 1×1, 5×5 filter with stride 1, pad 2.
- Output volume size?
  - 1×1

Output Volumn Size w.r.t. Padding and Stride

- Input volume: 2×2, 5×5 filter with stride 1, pad 2.
- Output volume size?
  - 2×2

Output Volumn Size w.r.t. Padding and Stride

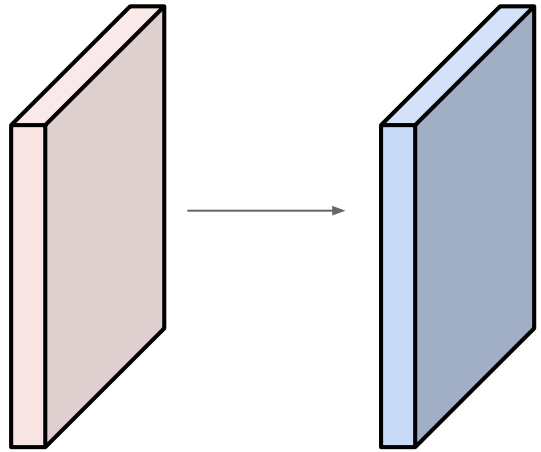
- Input volume: 2×2, 5×5 filter with stride 1, pad 2.
- Output volume size?
  - 2×2

Output Volumn Size w.r.t. Padding and Stride

Examples

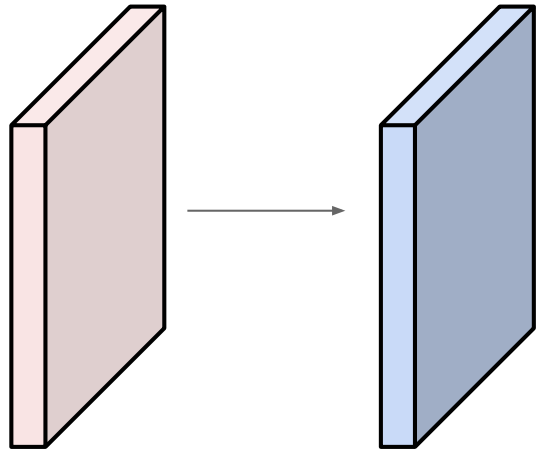
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Output Volumn Size w.r.t. Padding and Stride

Examples

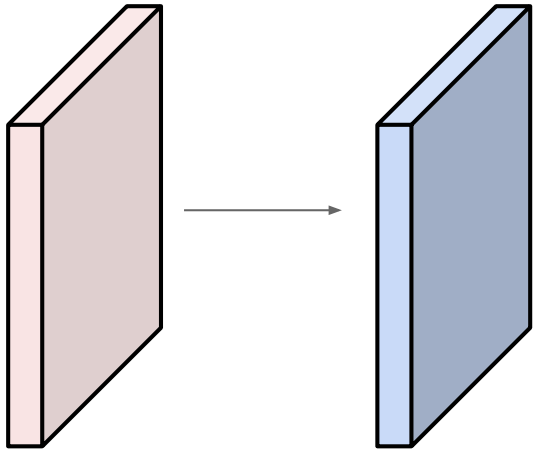


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Output Volumn Size w.r.t. Padding and Stride

Examples

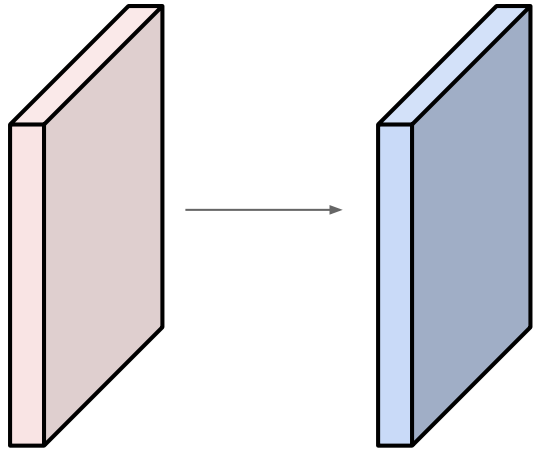


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Output Volumn Size w.r.t. Padding and Stride

Examples



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Summary

To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$.
- Requires four hyperparameters:
  - Number of filters K.
  - Filter size F.
  - Stride S.
  - Amount of zero-padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$:

$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$
$$D_2 = K$$

- With parameter sharing, it introduces $F^2 D_1$ weights per filter, for a total of $F^2 D_1 K$ weights and K biases.
- In the output volume, the d-th depth slices (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input with a stride of S, and then offset by d-th bias.

Summary

To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$.
- Requires four hyperparameters:
  - Number of filters K.
  - Filter size F.
  - Stride S.
  - Amount of zero-padding P.

Common settings:

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

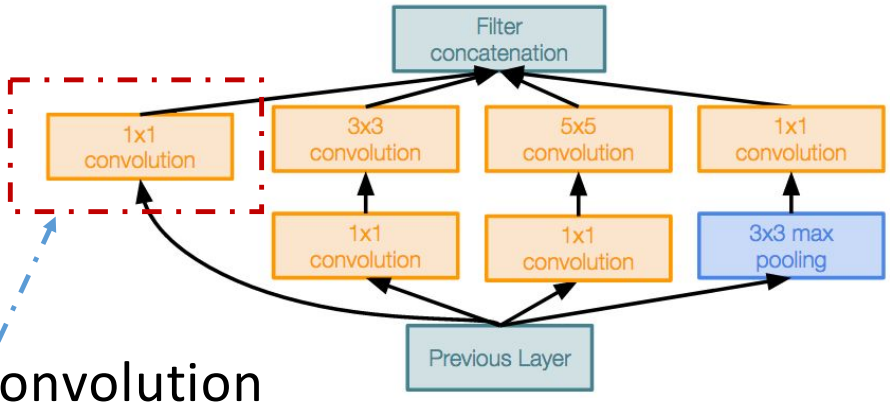
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

The Power of 1×1 Convolution[3]

[3] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

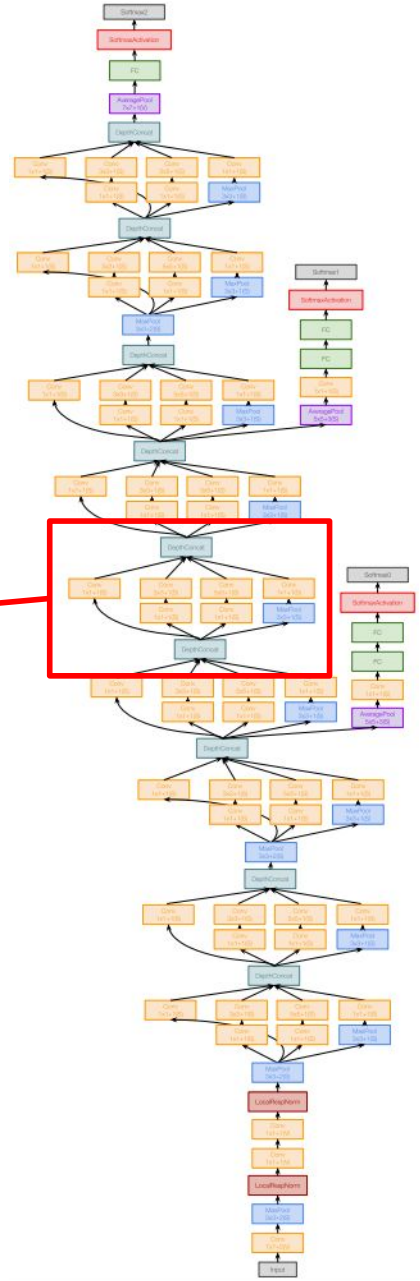
GoogleNet

Stack the module on top of each other!



1×1 convolution

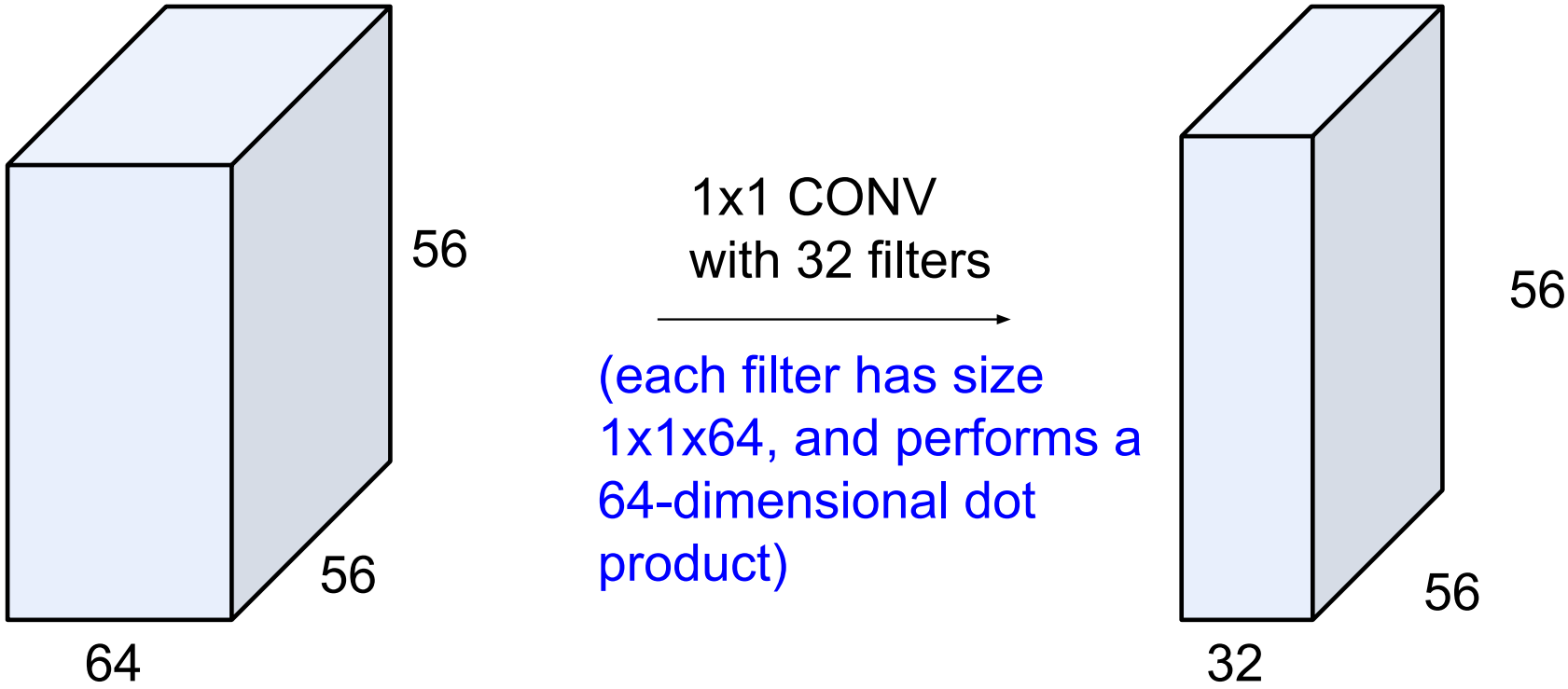
Inception module



1x1 Convolution Layers Make Perfect Sense

Used in GoogleNet (inception model)

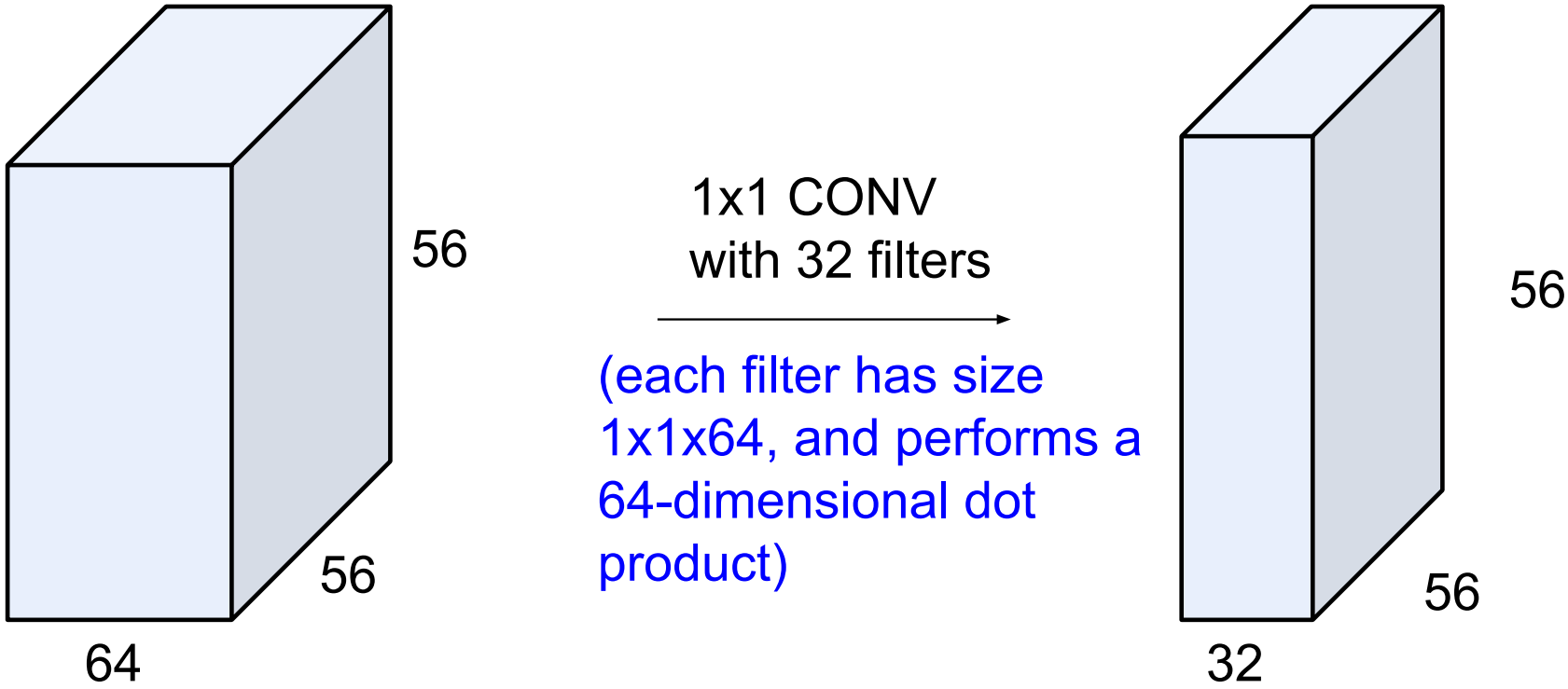
Perform like dimension reduction/extension on the third dimension.



1x1 Convolution Layers Make Perfect Sense

Used in GoogleNet (inception model)

Perform like dimension reduction/extension on the third dimension.



What happens with the 1×1 convolution here?

The Power of Small Filters

Why do we need small filters?

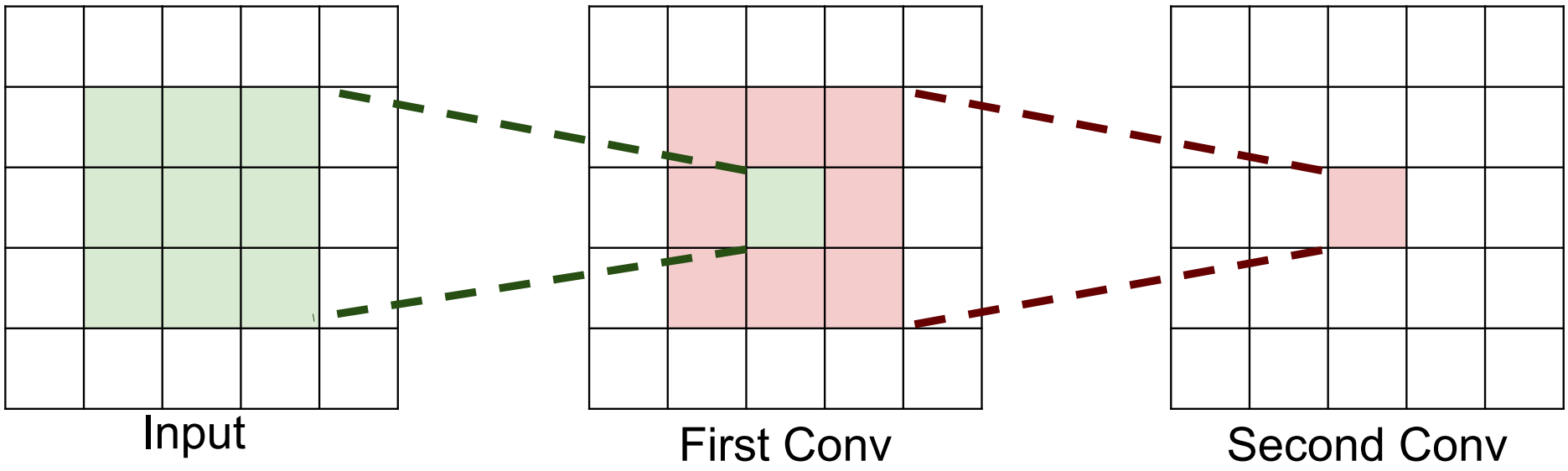
Proof Idea

- Define two kinds of CNNs which have different filter sizes such that they have the same representation power.
- Show the CNN with smaller filter size has less parameters.

The Power of Small Filters

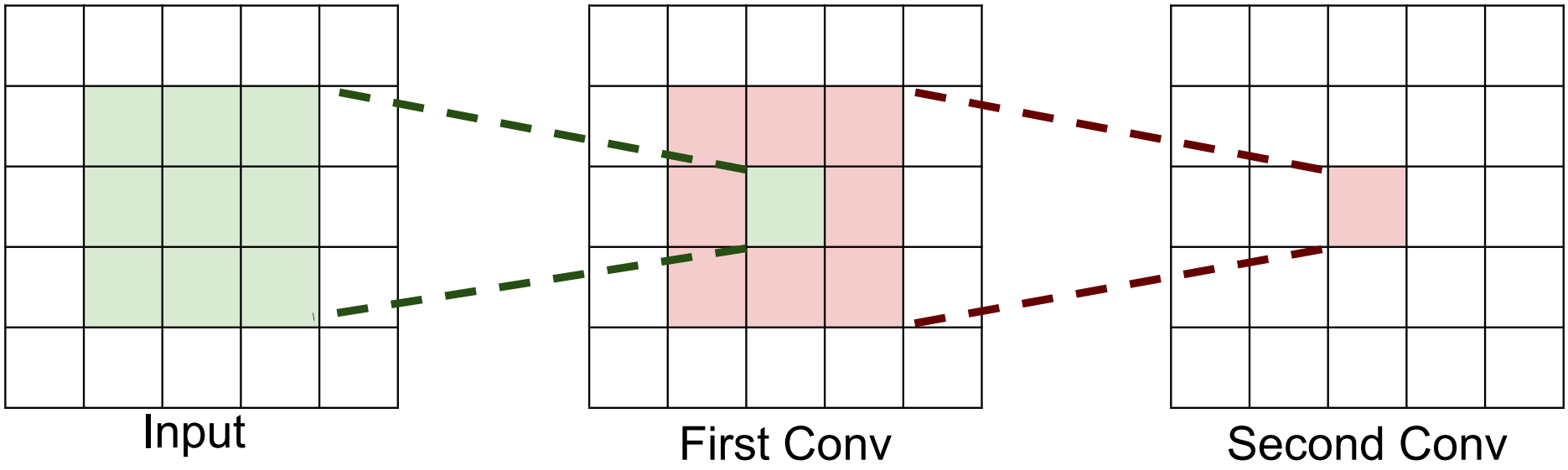
Suppose we stack two 3x3 (filter size) conv layers (stride 1):

- each neuron sees 3x3 region of previous activation map.



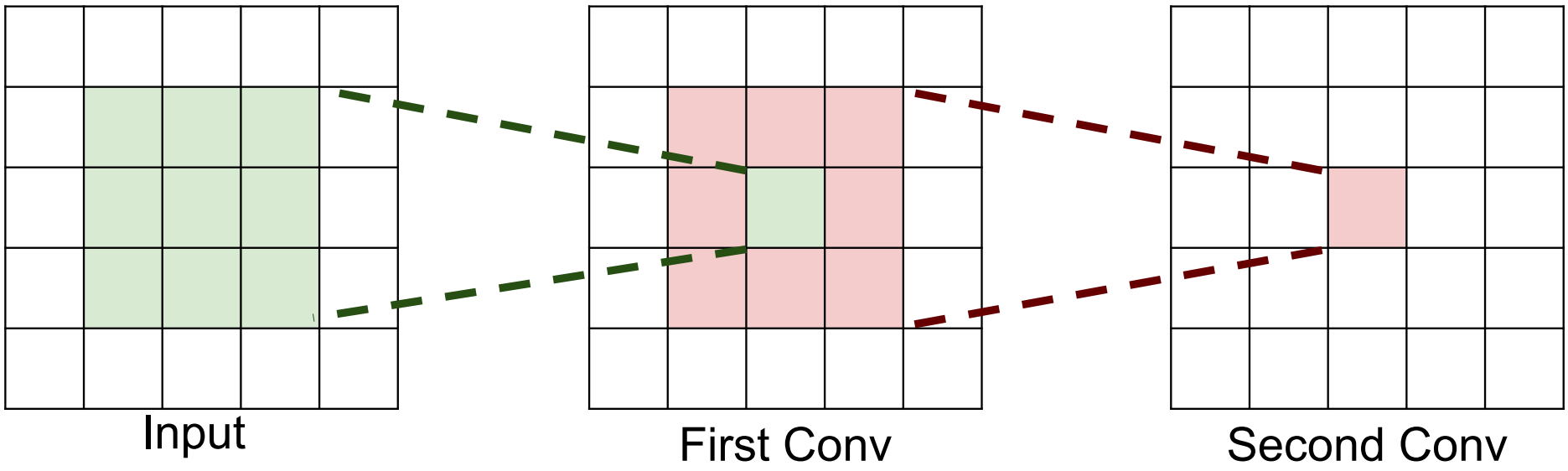
The Power of Small Filters

Question: How big of a region in the input does a neuron on the second conv layer see?



The Power of Small Filters

Question: How big of a region in the input does a neuron on the second conv layer see?
Answer: 5×5

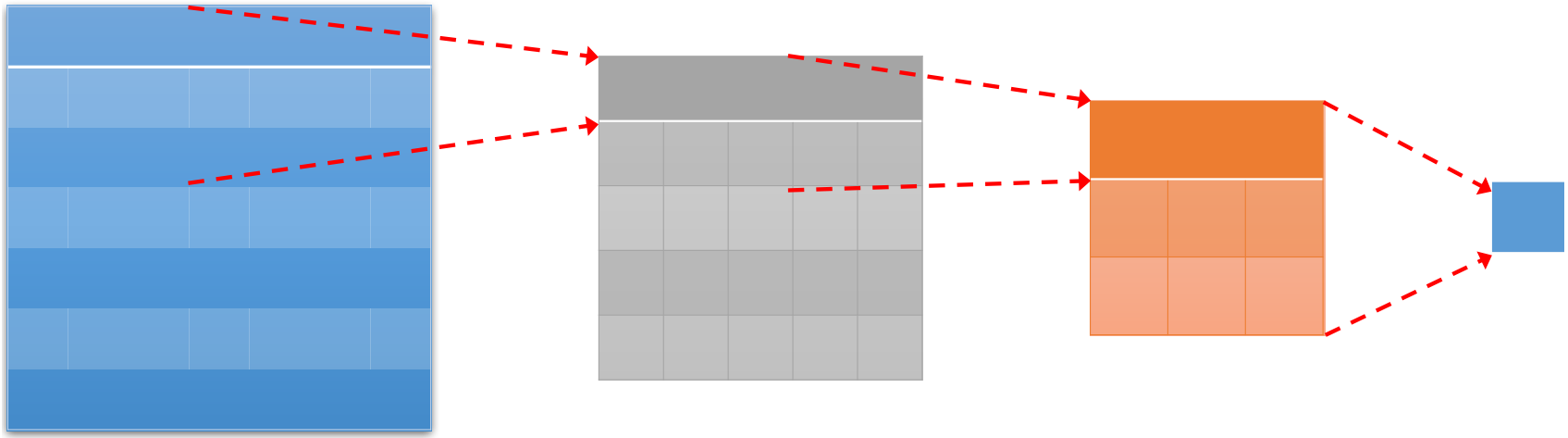


The Power of Small Filters

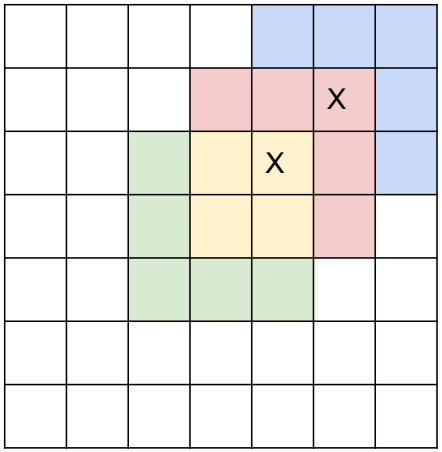
Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

The Power of Small Filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?



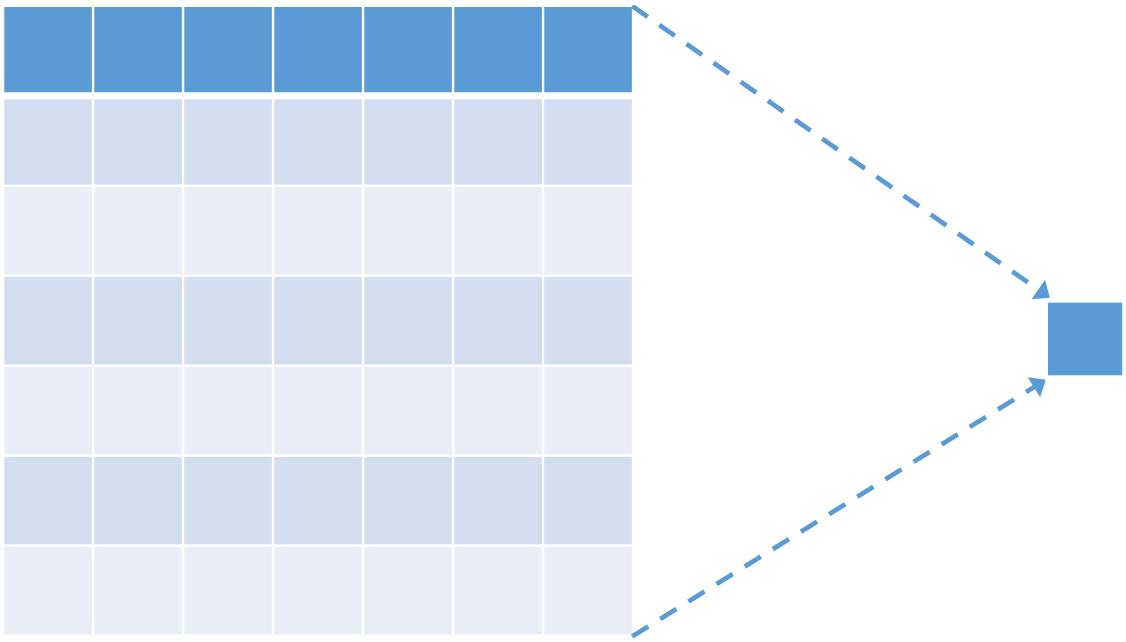
Answer: 7 x 7



The Power of Small Filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

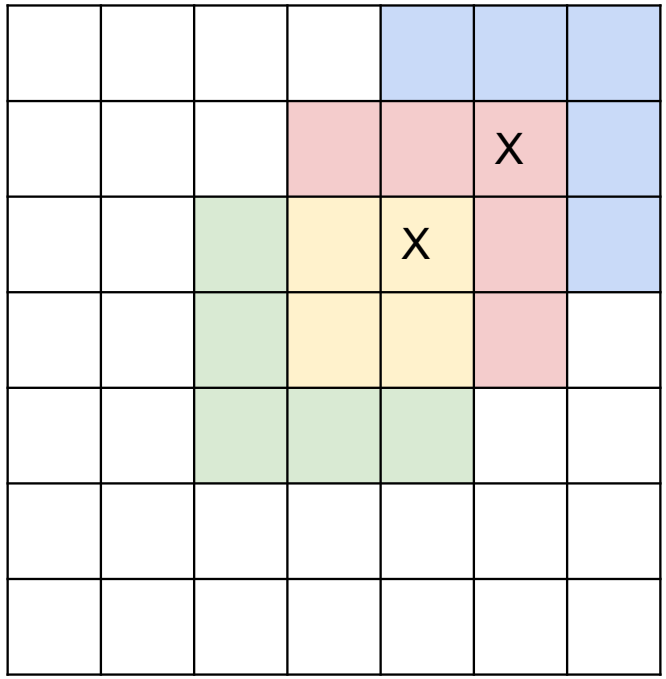
- If we use 1 convolution layer, we need to use a kernel of 7×7 in order to let a neuron in the output layer see a 7×7 region in the input.



The Power of Small Filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Answer: 7 x 7



Three 3 x 3 conv gives similar representational power as a single 7 x 7 convolution

While it has less parameters!

The Power of Small Filters

Suppose input is $H \times W \times C$ and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W).

- One CONV with 7×7 filters.
- Number of weights:
  $= C \times (7 \times 7 \times C) = 49C^2$
- Number of multiply-adds:
  $= (H \times W \times C) \times (7 \times 7 \times C) = 49HWC^2$

- Three CONV with 3 x 3 filters.
- Number of weights:
  $= 3 \times C \times (3 \times 3 \times C) = 27C^2$
  fewer parameters = GOOD
- Number of multiply-adds:
  $= 3 \times (H \times W \times C) \times (3 \times 3 \times C) = 27HWC^2$
  less compute = GOOD

The Power of Small Filters

Suppose input is $H \times W \times C$ and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W).

- One CONV with 7×7 filters.
- Number of weights:
  $$= C \times (7 \times 7 \times C) = 49C^2$$
- Number of multiply-adds:
  $$= (H \times W \times C) \times (7 \times 7 \times C) = 49HWC^2$$

- Three CONV with 3 x 3 filters.
- Number of weights:
  $$= 3 \times C \times (3 \times 3 \times C) = 27C^2$$
  fewer parameters = GOOD
- Number of multiply-adds:
  $$= 3 \times (H \times W \times C) \times (3 \times 3 \times C) = 27HWC^2$$
  less compute = GOOD

The Power of Small Filters

Suppose input is $H \times W \times C$ and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W).

- One CONV with 7×7 filters.
- Number of weights:
  $$= C \times (7 \times 7 \times C) = 49C^2$$
- Number of multiply-adds:
  $$= (H \times W \times C) \times (7 \times 7 \times C) = 49HWC^2$$

- Three CONV with 3 x 3 filters.
- Number of weights:
  $$= 3 \times C \times (3 \times 3 \times C) = 27C^2$$
  fewer parameters = GOOD
- Number of multiply-adds:
  $$= 3 \times (H \times W \times C) \times (3 \times 3 \times C) = 27HWC^2$$
  less compute = GOOD

The Power of Small Filters

Suppose input is $H \times W \times C$ and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W).

- One CONV with 7×7 filters.
- Number of weights:
  $= C \times (7 \times 7 \times C) = 49C^2$
- Number of multiply-adds:
  $= (H \times W \times C) \times (7 \times 7 \times C) = 49HWC^2$

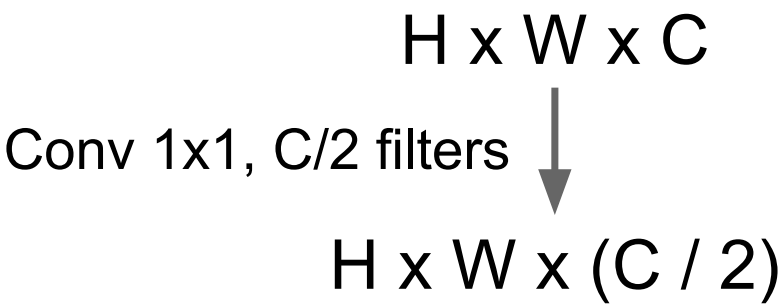
- Three CONV with 3 x 3 filters.
- Number of weights:
  $= 3 \times C \times (3 \times 3 \times C) = 27C^2$
  fewer parameters = GOOD
- Number of multiply-adds:
  $= 3 \times (H \times W \times C) \times (3 \times 3 \times C) = 27HWC^2$
  less compute = GOOD

The Power of Small Filters

Why stop at 3×3 filters? Why not try 1×1?

The Power of Small Filters

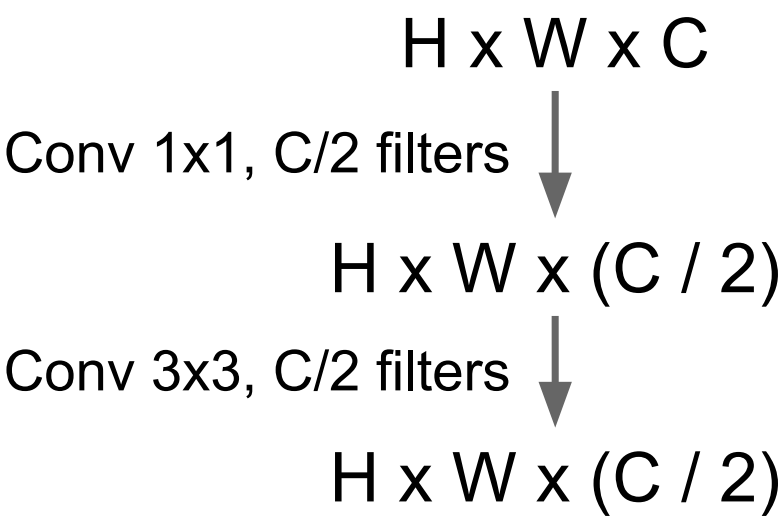
Why stop at 3×3 filters? Why not try 1×1?



1. "bottleneck" 1 x 1 conv to reduce dimension

The Power of Small Filters

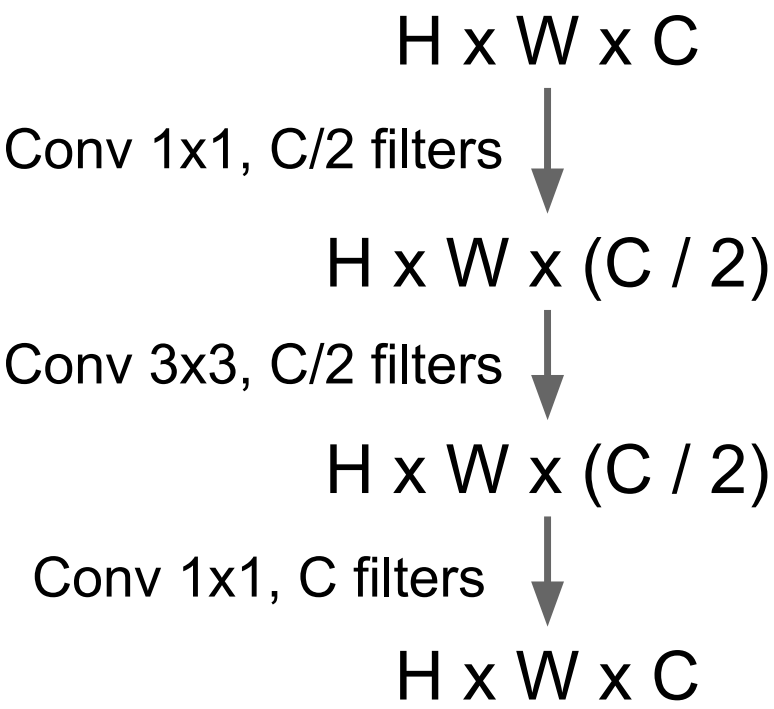
Why stop at 3×3 filters? Why not try 1×1?



1. "bottleneck" 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension

The Power of Small Filters

Why stop at 3×3 filters? Why not try 1×1?

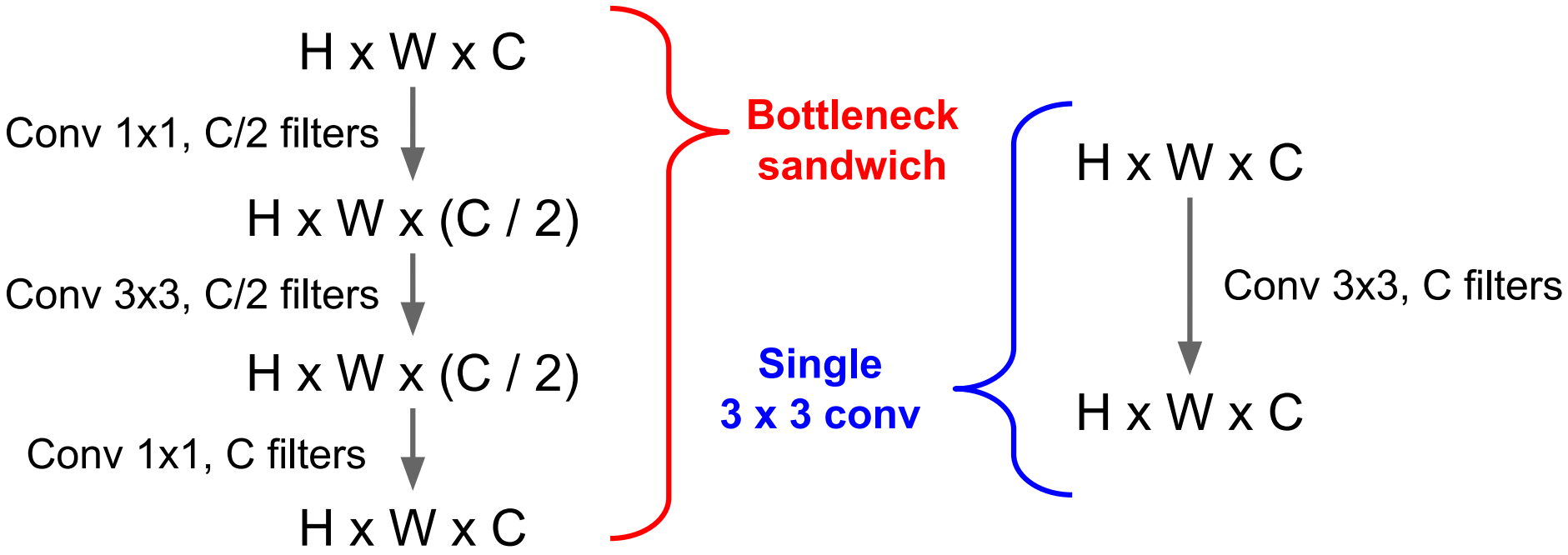


1. "bottleneck" 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension
3. Restore dimension with another 1 x 1 conv

[Seen in Lin et al, "Network in Network", GoogLeNet, ResNet]

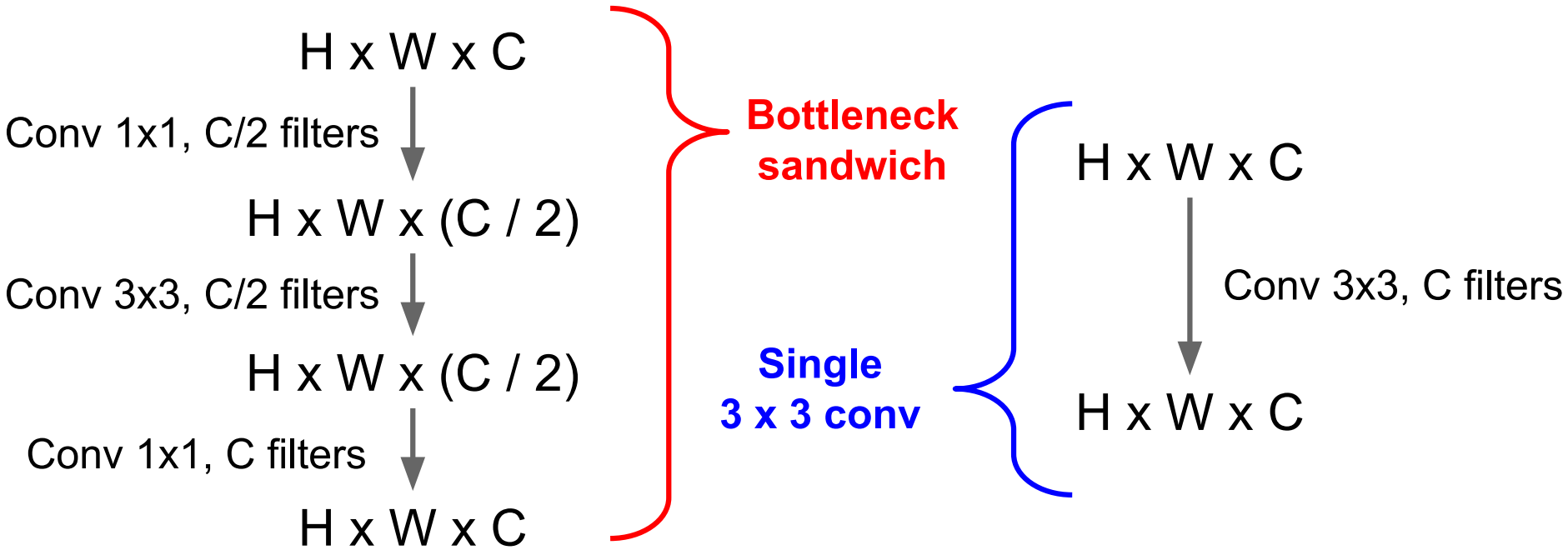
The Power of Small Filters

Why stop at 3×3 filters? Why not try 1×1?



The Power of Small Filters

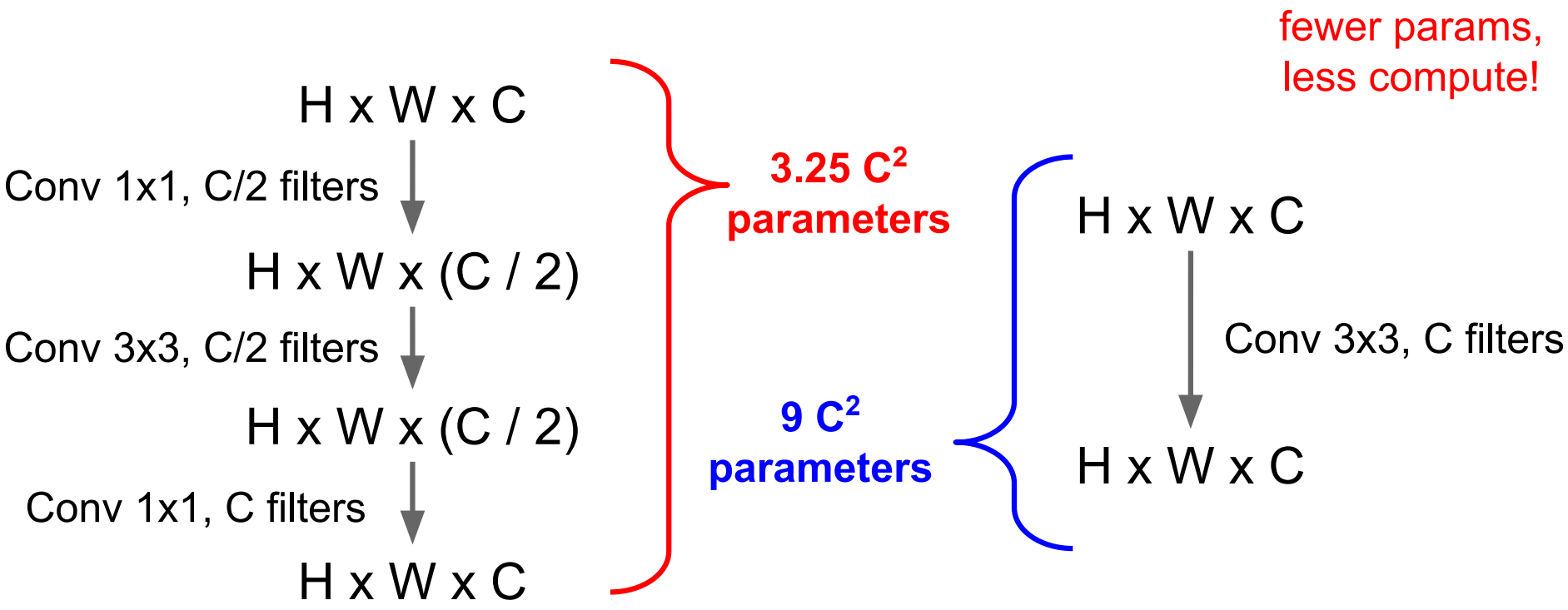
Why stop at 3×3 filters? Why not try 1×1?



How many parameters?

The Power of Small Filters

Why stop at 3×3 filters? Why not try 1×1?

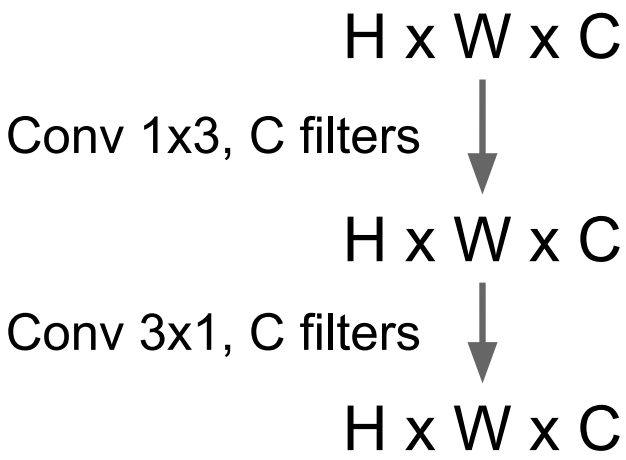


The Power of Small Filters

Still using 3×3 filters $\cdots$ can we break it up?

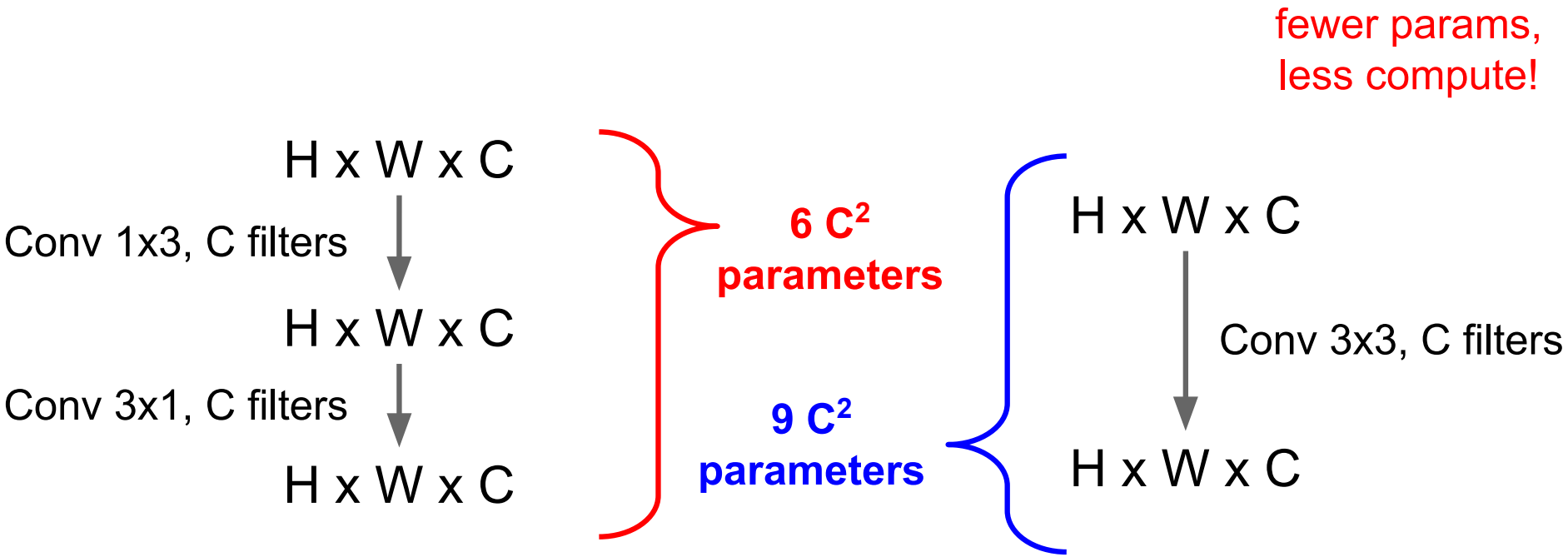
The Power of Small Filters

Still using 3×3 filters $\cdots$ can we break it up?



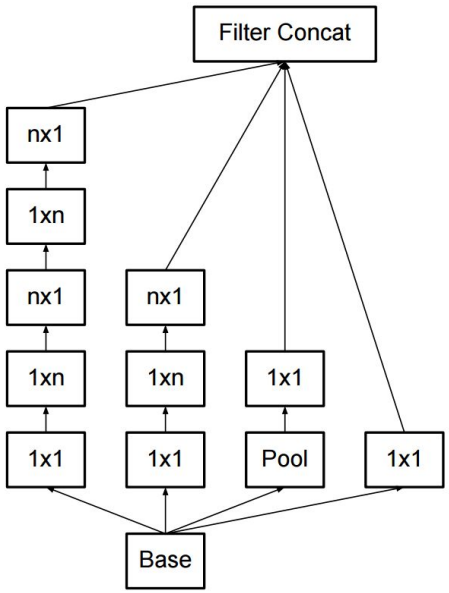
The Power of Small Filters

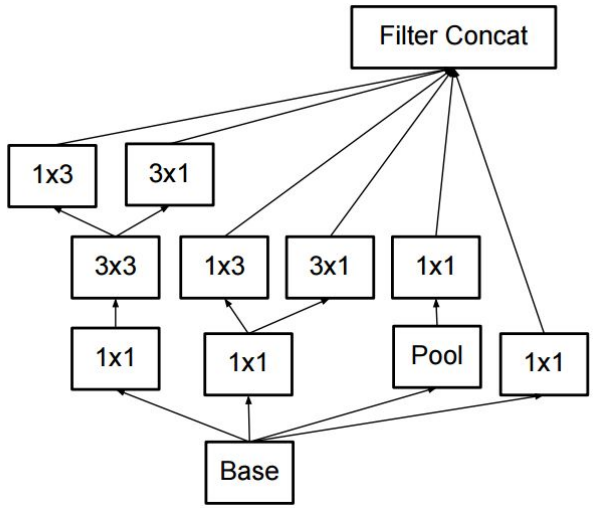
Still using 3×3 filters $\cdots$ can we break it up?

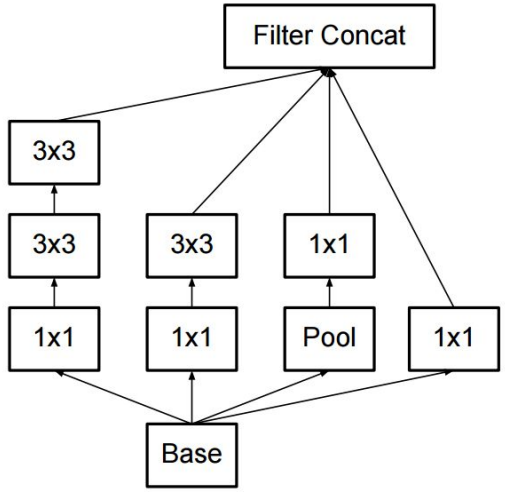


The Power of Small Filters

Latest version of GoogLeNet incorporates all these ideas







Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”

How to Stack Convolutions: Recap

1. Replace large convolutions (5×5, 7×7) with stacks of 3×3 convolutions.
2. 1×1 "bottleneck" convolutions are very efficient.
3. Can factor $N \times N$ convolutions into $1 \times N$ and $N \times 1$.
4. All of the above give fewer parameters and less compute.