THE UNIVERSITY OF BRITISH COLUMBIA

Topics in AI (CPSC 532S): Multimodal Learning with Vision, Language and Sound

Lecture 5: Convolutional Neural Networks (Part 2)

Logistics:

Assignment 2 will be out tonight

— Data is reusable

get it on your Google Drive, will be used for Assignment 3 & 4

you **can** reduce the dataset size (e.g., 1/2 or 1/4 of data)

— Piazza (make questions public)

Logistics:

Assignment 2 will be out tonight

— Data is reusable

get it on your Google Drive, will be used for Assignment 3 & 4

you **can** reduce the dataset size (e.g., 1/2 or 1/4 of data)

— Piazza (make questions public)

Office Hours

Logistics:

Assignment 2 will be out tonight

— Data is reusable

get it on your Google Drive, will be used for Assignment 3 & 4

you **can** reduce the dataset size (e.g., 1/2 or 1/4 of data)

— Piazza (make questions public)

Office Hours

Projects

— Some guidelines

— November 1 & 3 **Project Proposals** (two lectures week before the break)

Logistics:

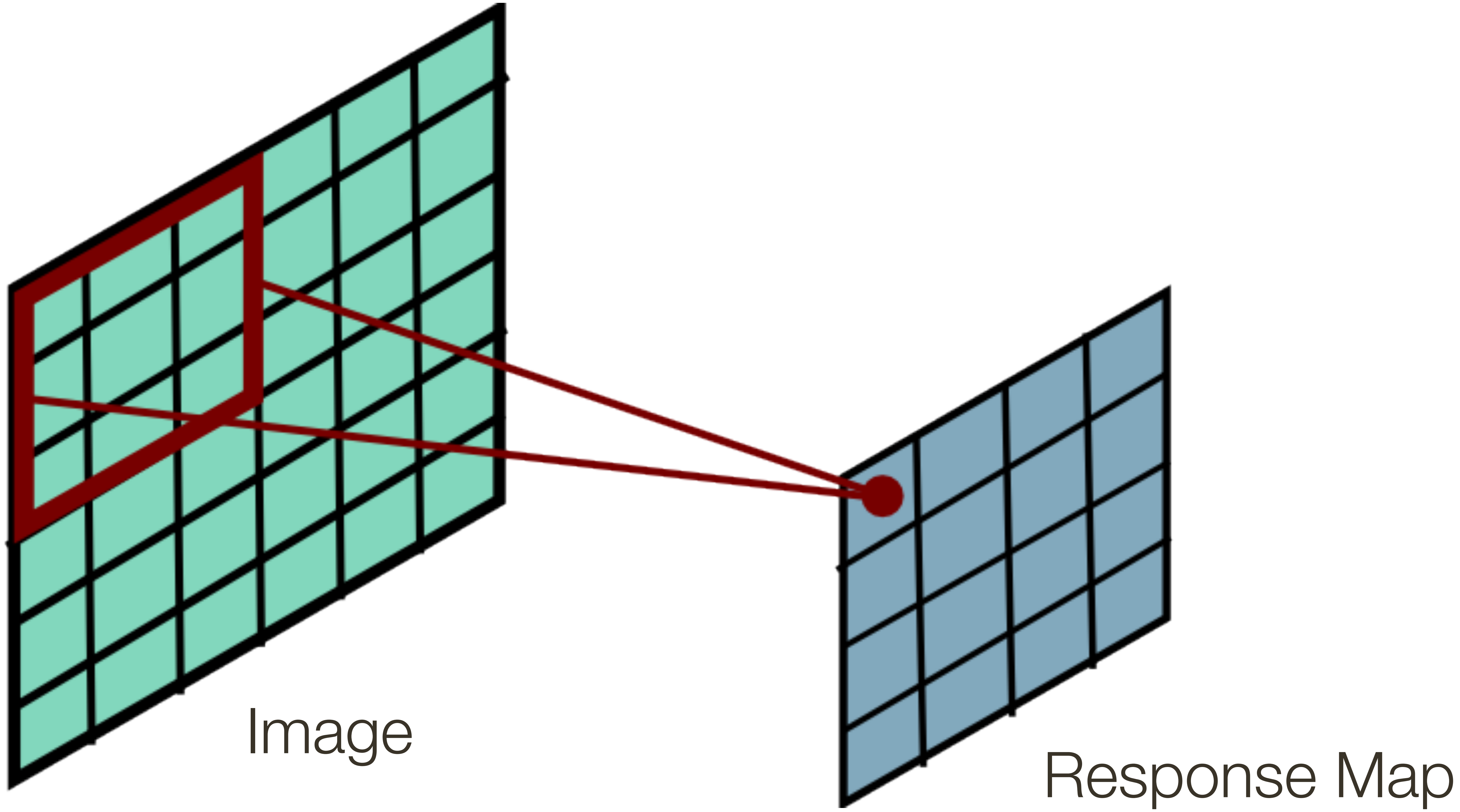
Assignment 2 will be out tonight
— Data is reusable
    get it on your Google Drive, will be used for Assignment 3 & 4
    you **can** reduce the dataset size (e.g., 1/2 or 1/4 of data)
— Piazza (make questions public)

Office Hours

Projects
— Some guidelines
— November 1 & 3 **Project Proposals** (two lectures week before the break)

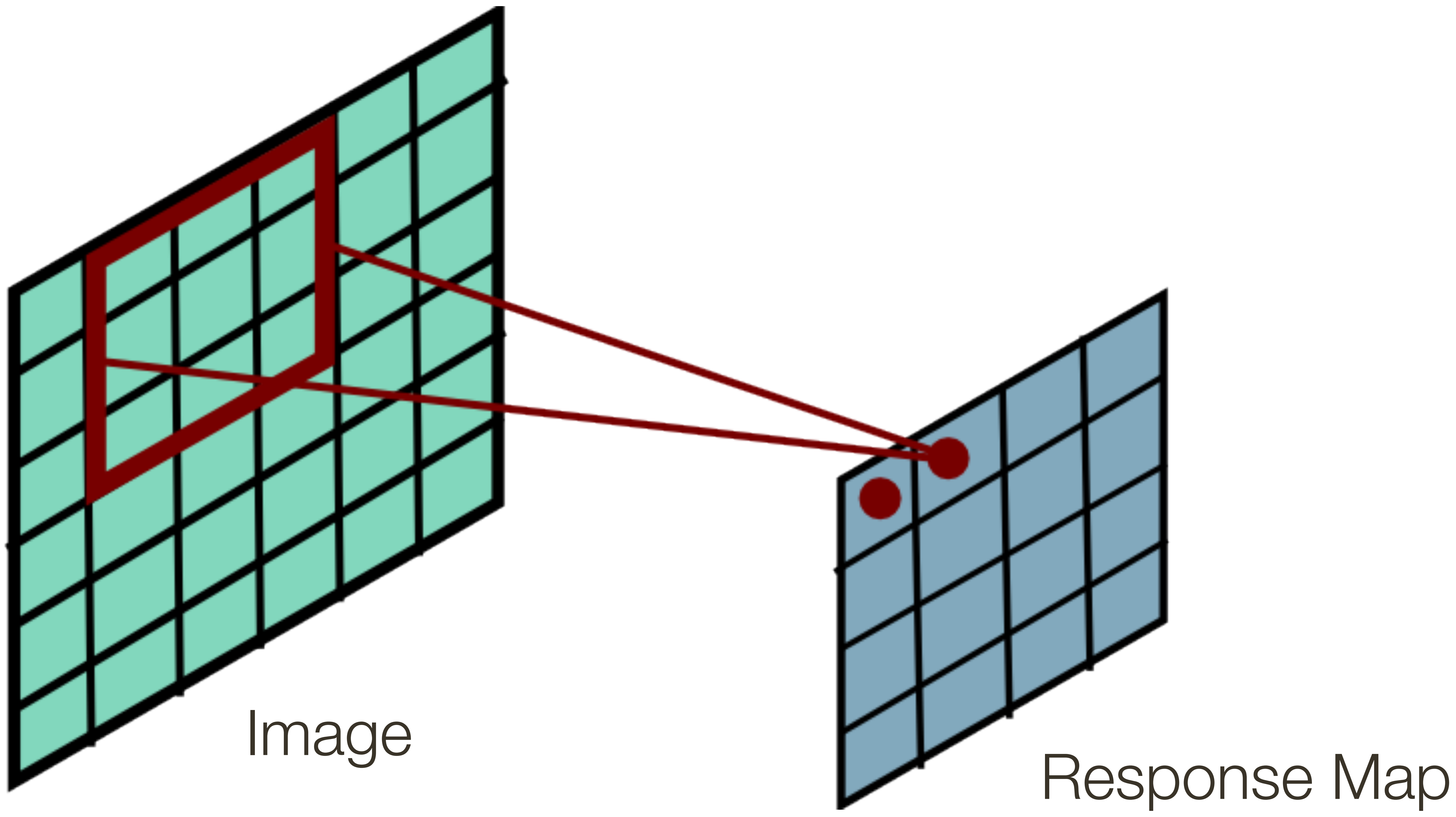
Paper presentations
— List of papers will be made available in the next 1 week

Last time: Convolutional Layer



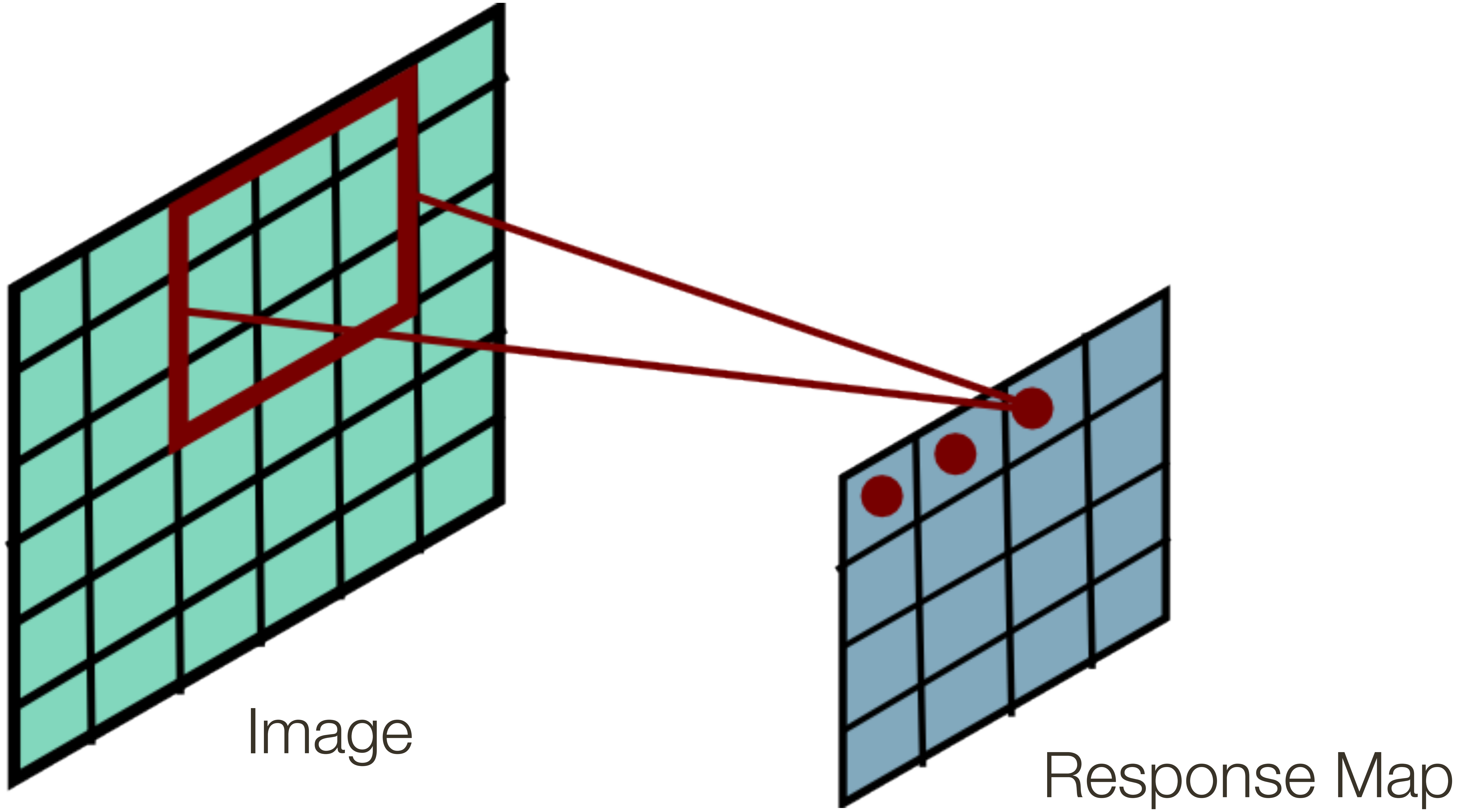
* slide from Marc'Aurelio Renzato

Last time: **Convolutional** Layer



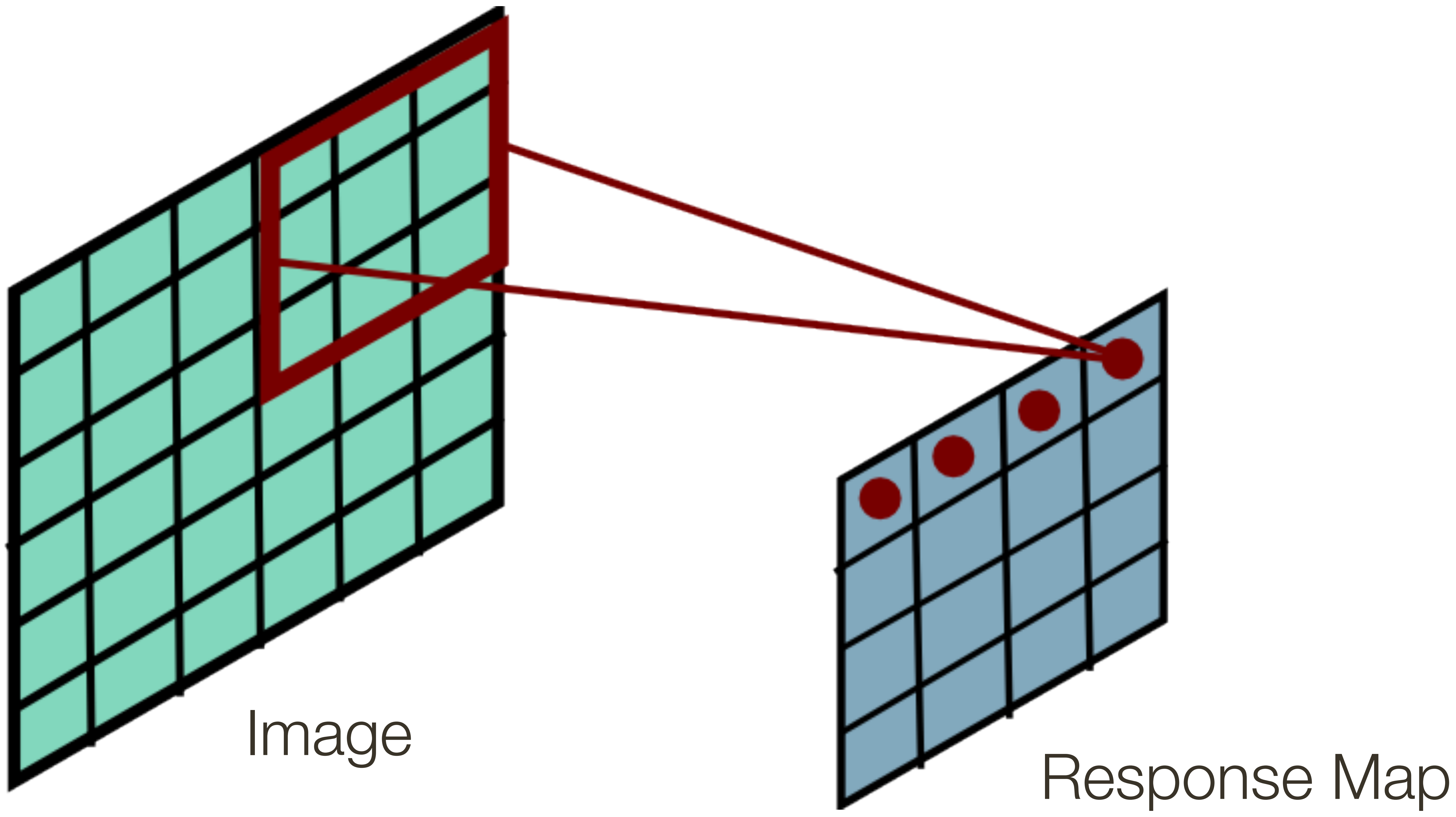
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



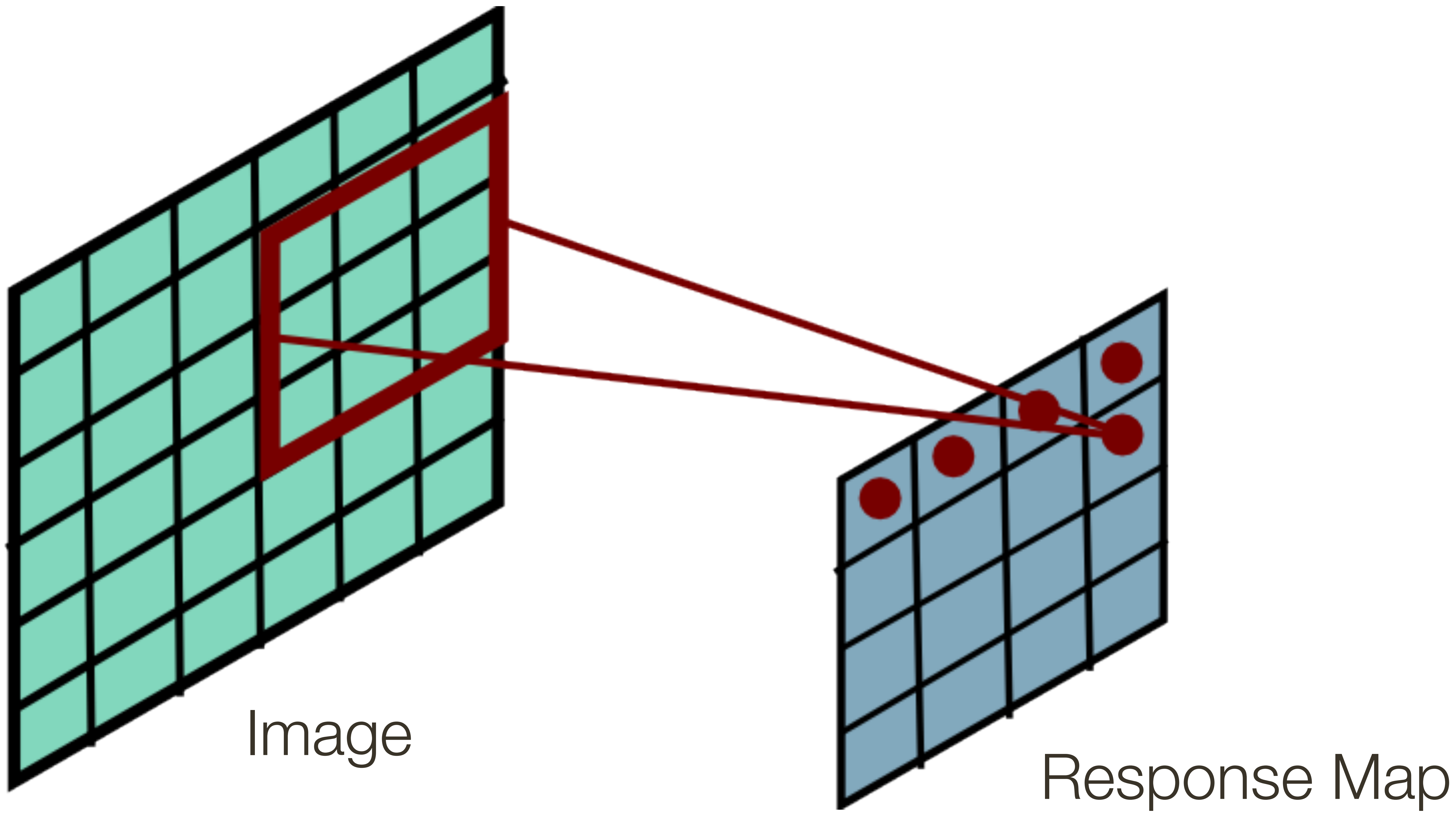
* slide from Marc'Aurelio Renzato

<u>Last time</u>: Convolutional Layer



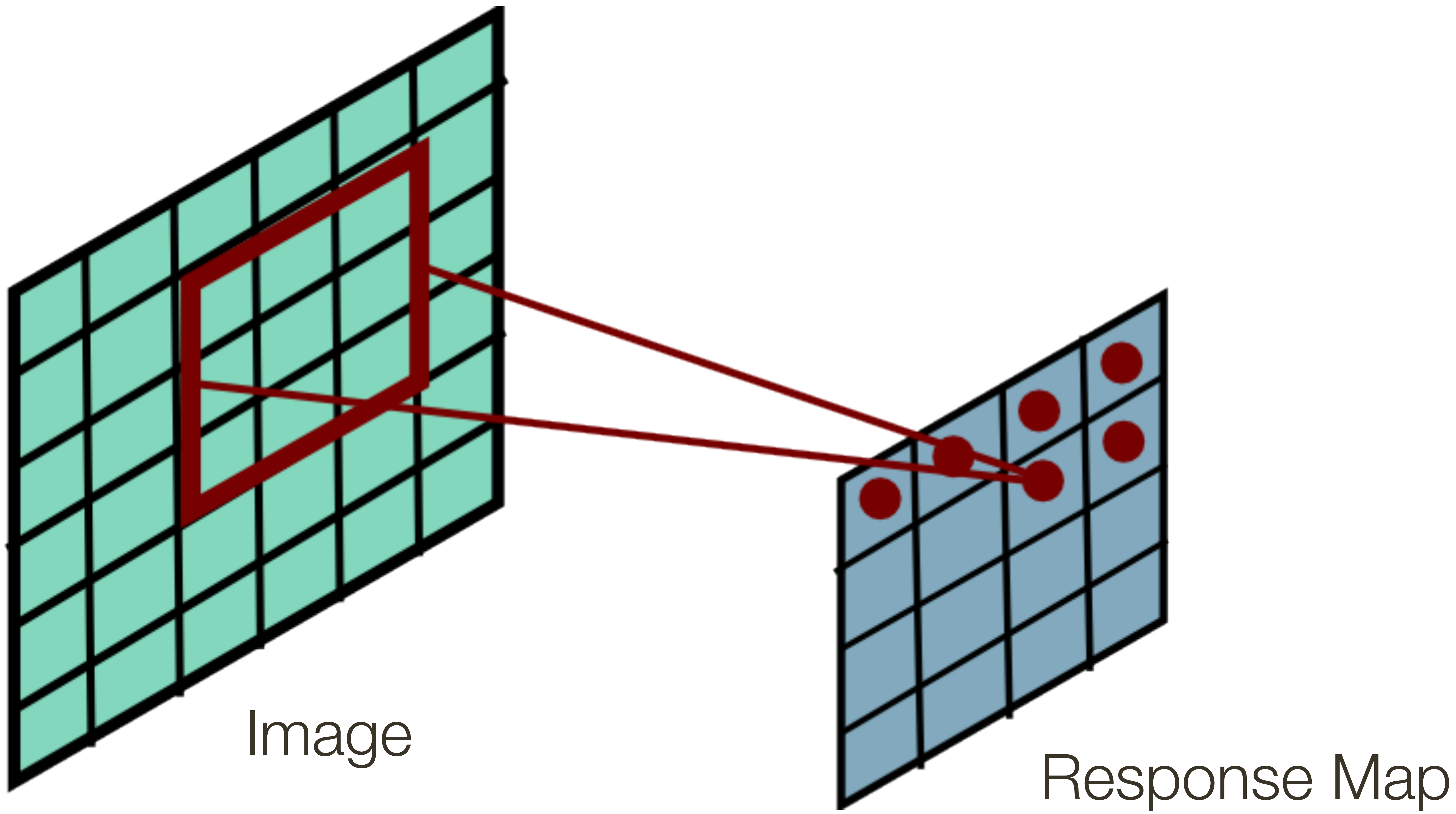
* slide from Marc'Aurelio Renzato

<u>Last time</u>: Convolutional Layer



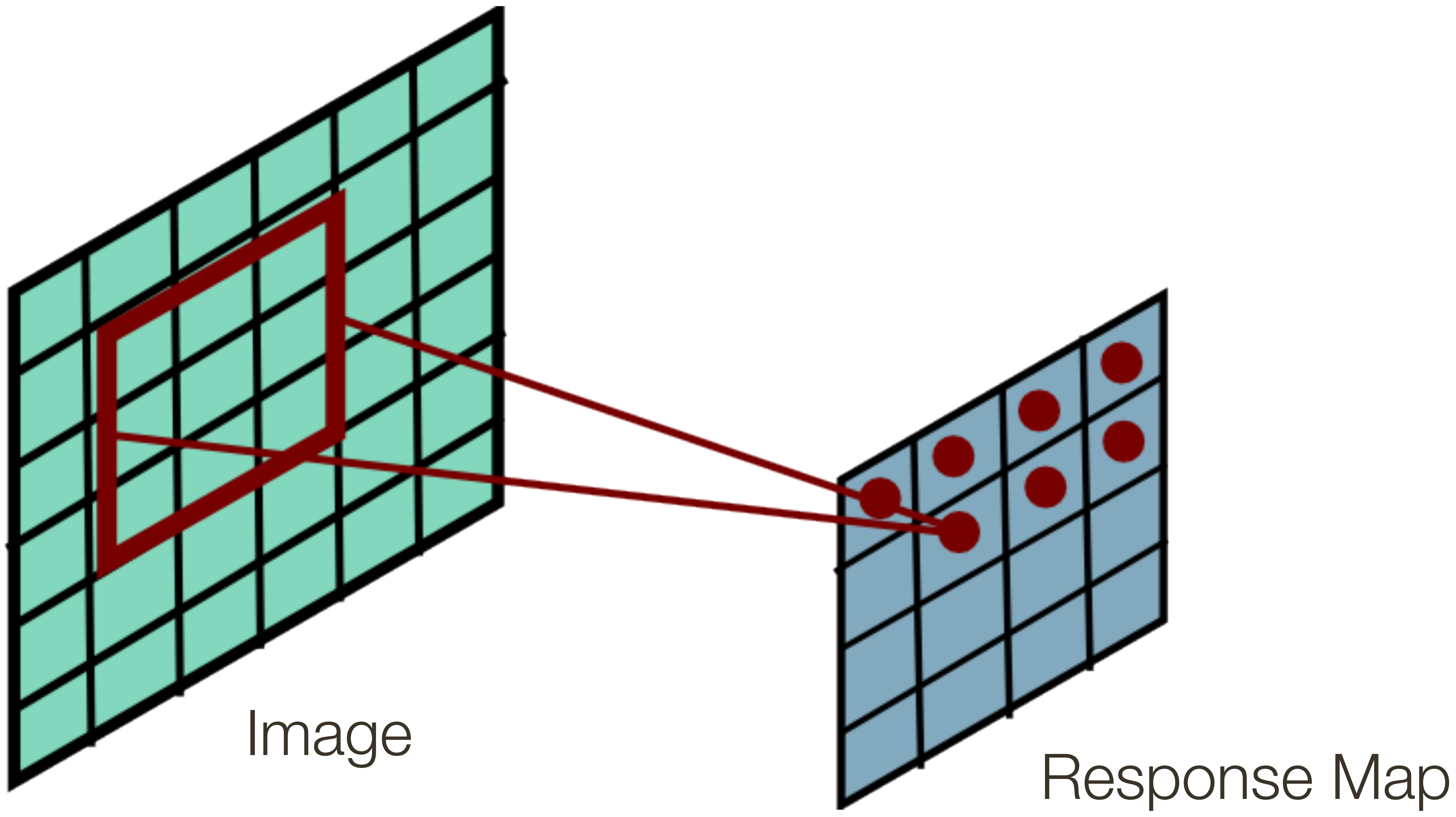
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



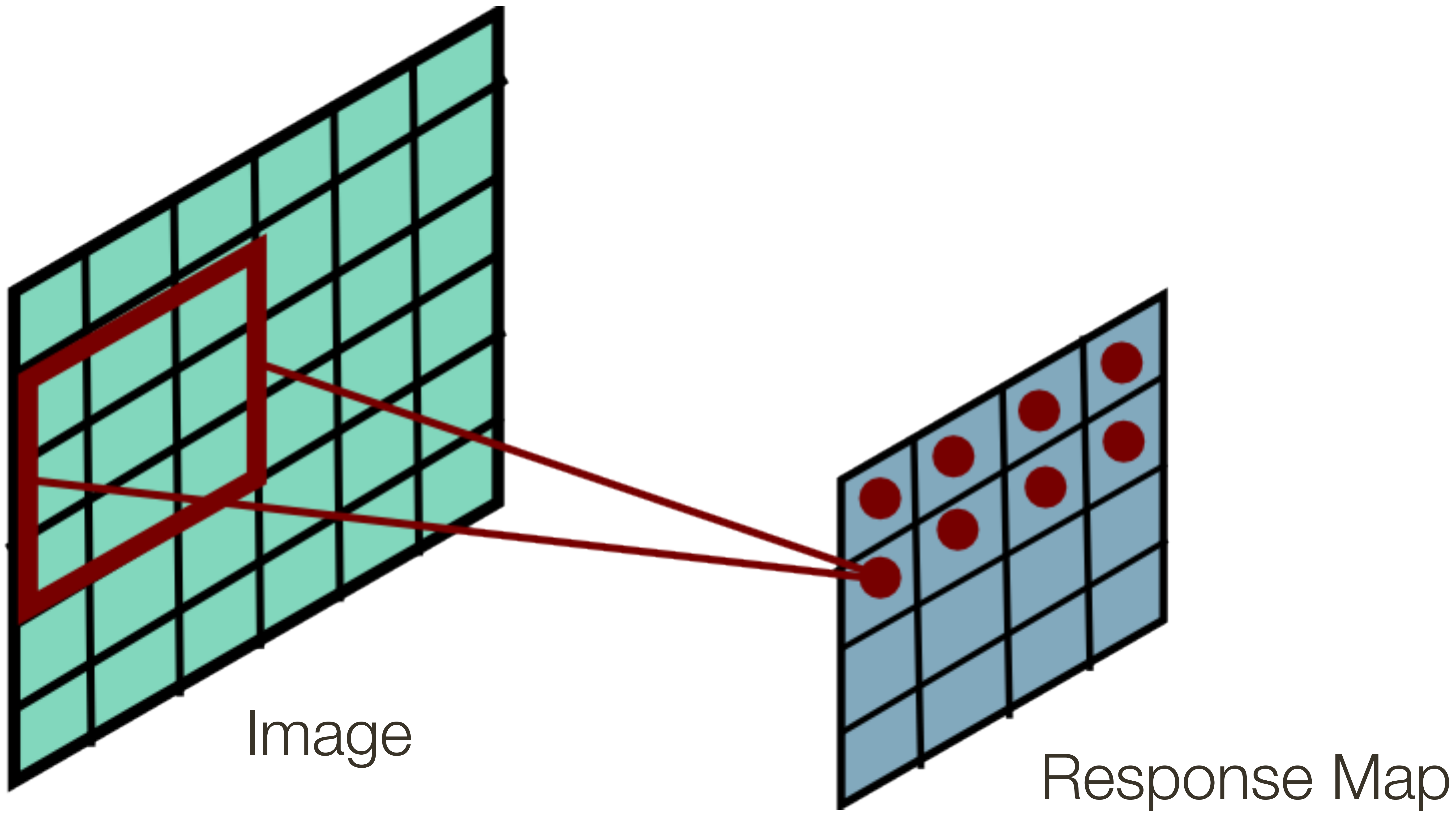
* slide from Marc'Aurelio Renzato

Last time: **Convolutional** Layer



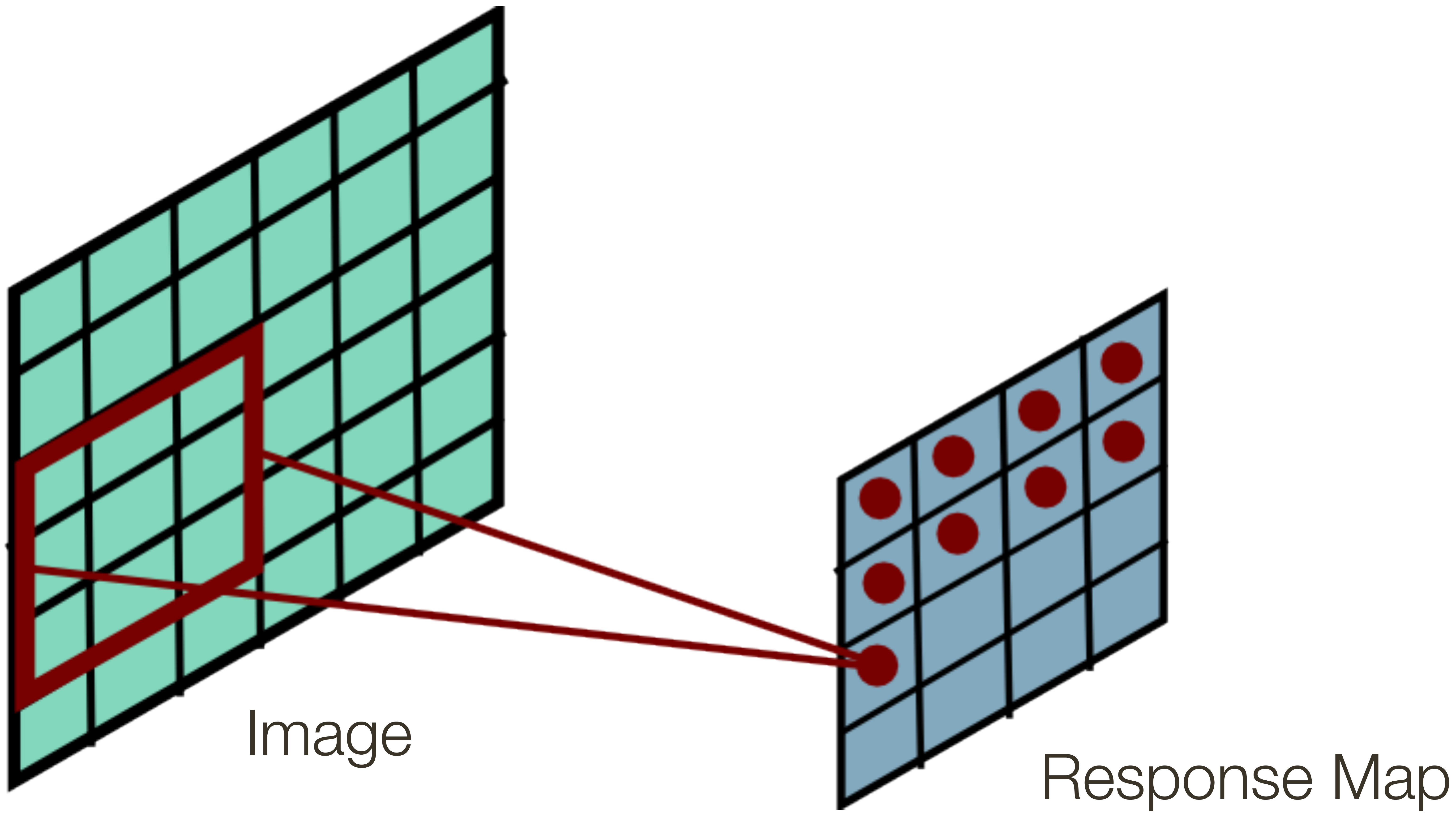
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



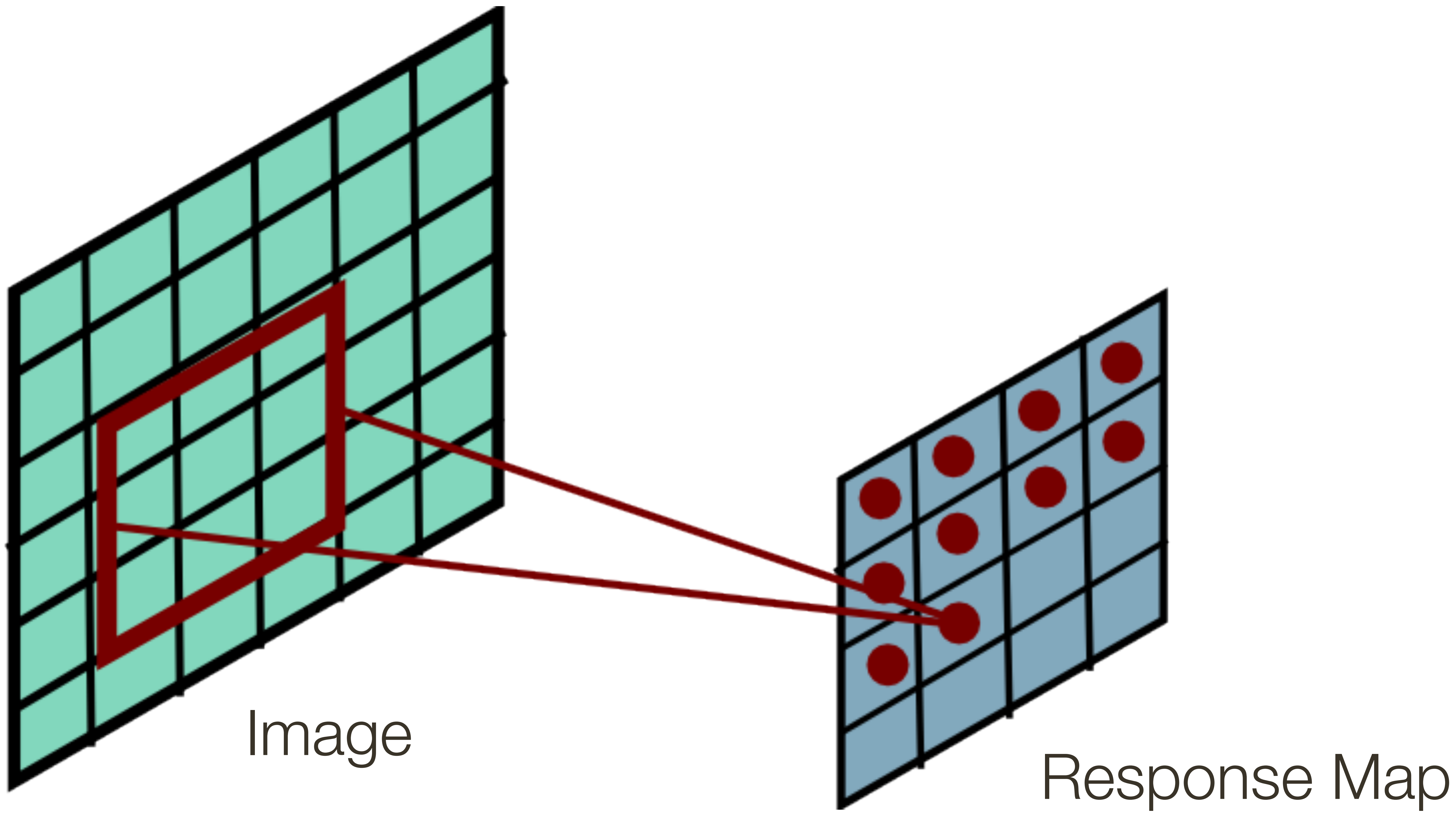
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



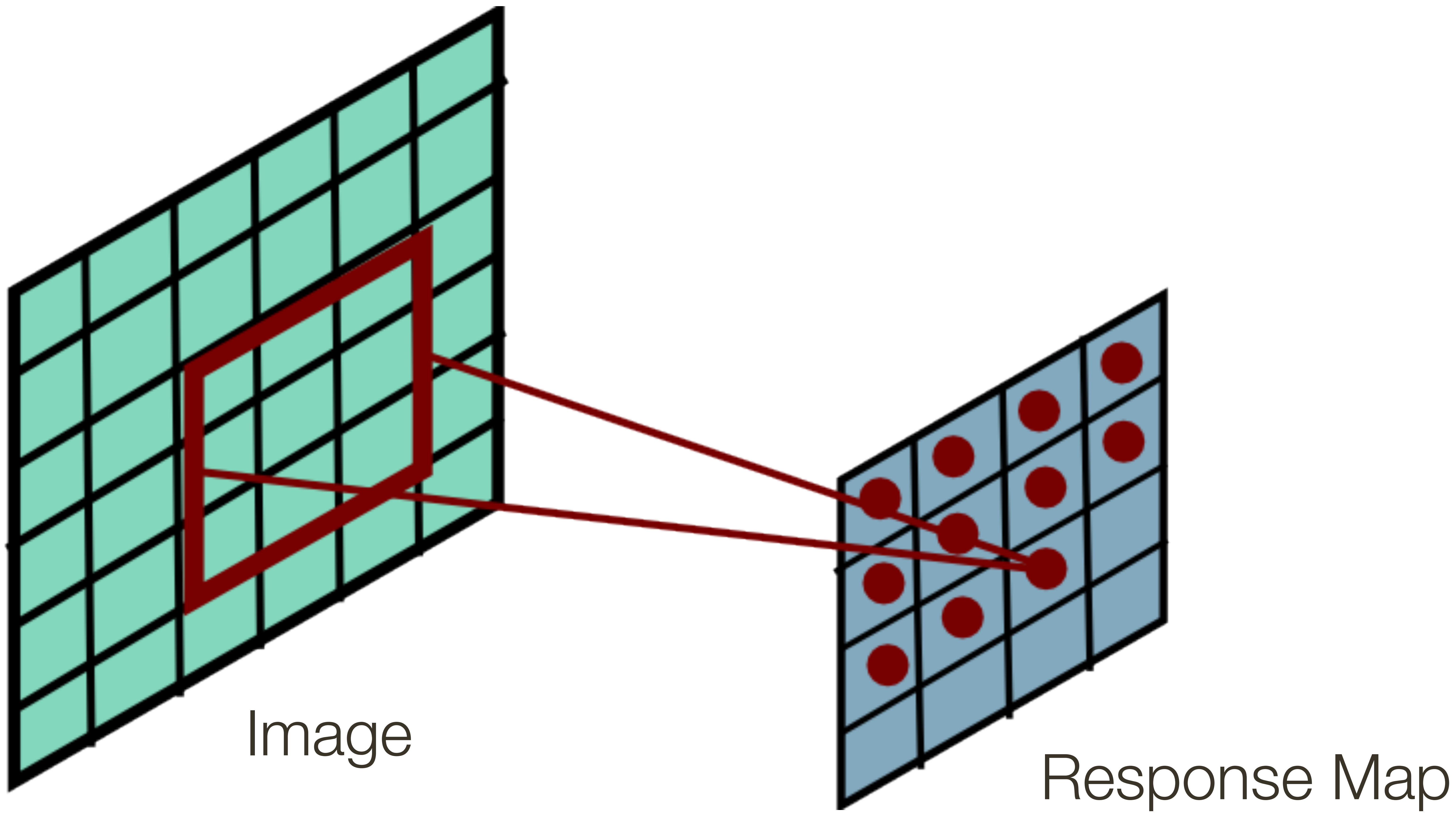
* slide from Marc'Aurelio Renzato

<u>Last time</u>: Convolutional Layer



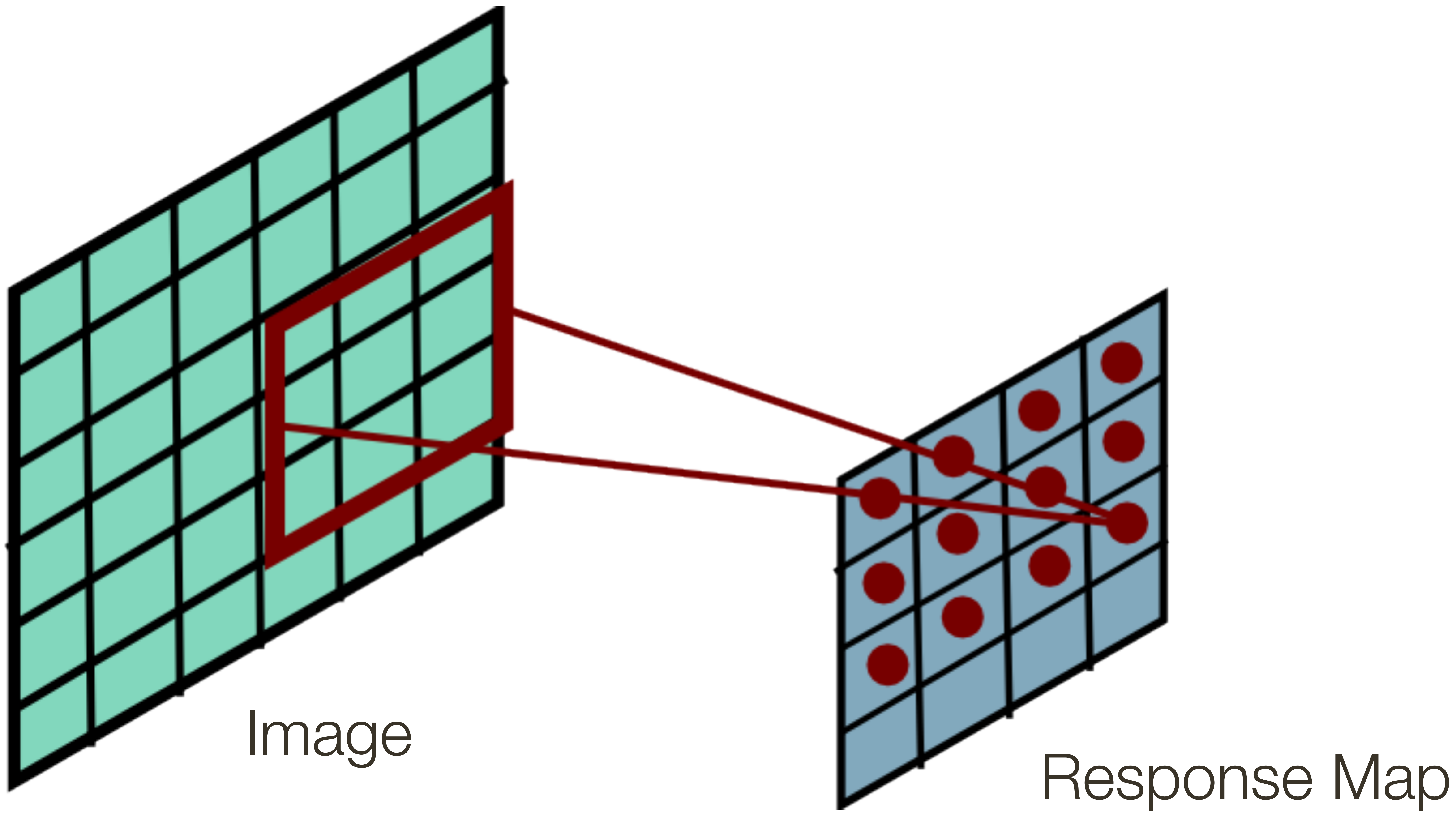
* slide from Marc'Aurelio Renzato

<u>Last time</u>: Convolutional Layer



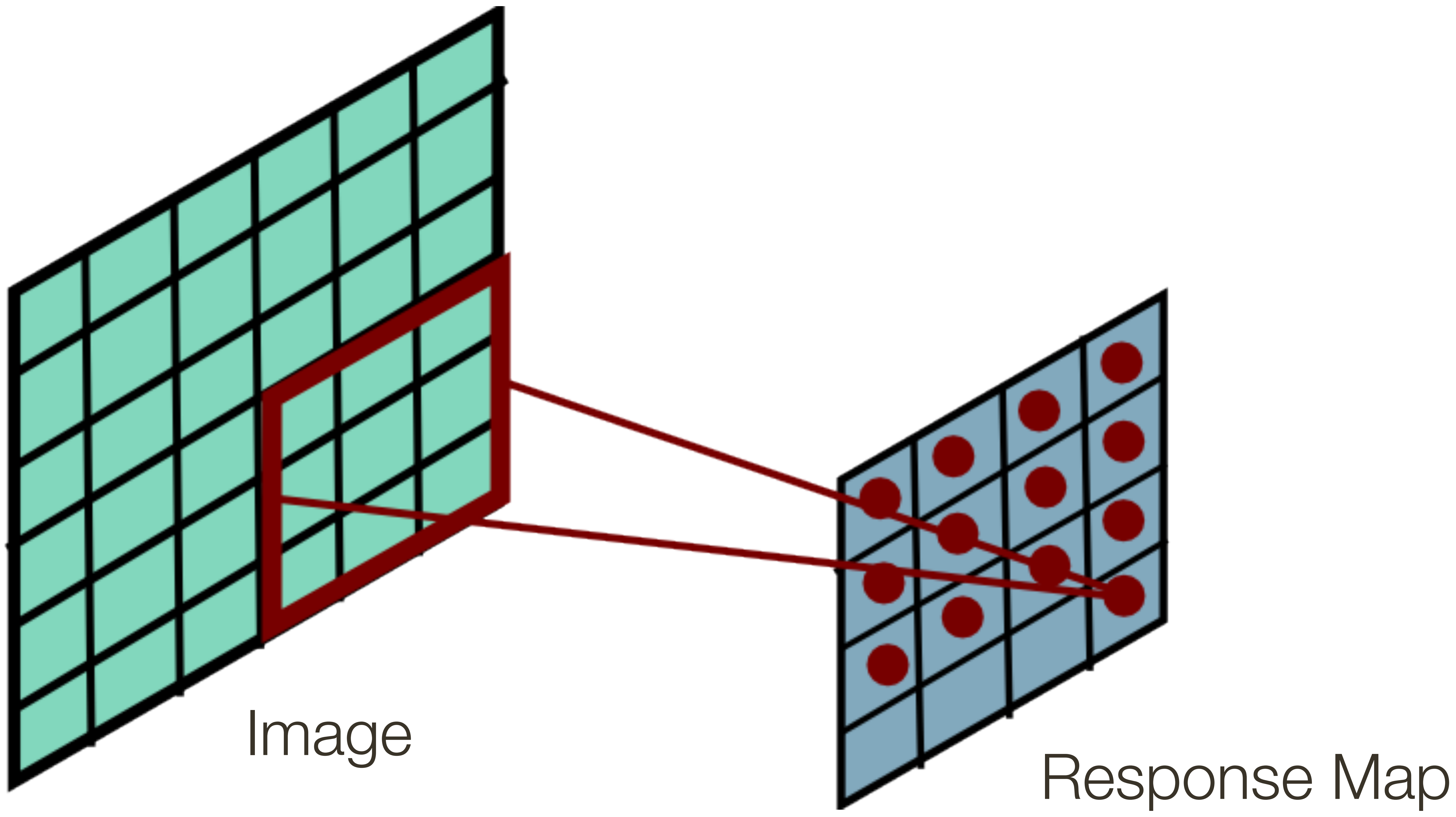
* slide from Marc'Aurelio Renzato

Last time: **Convolutional** Layer



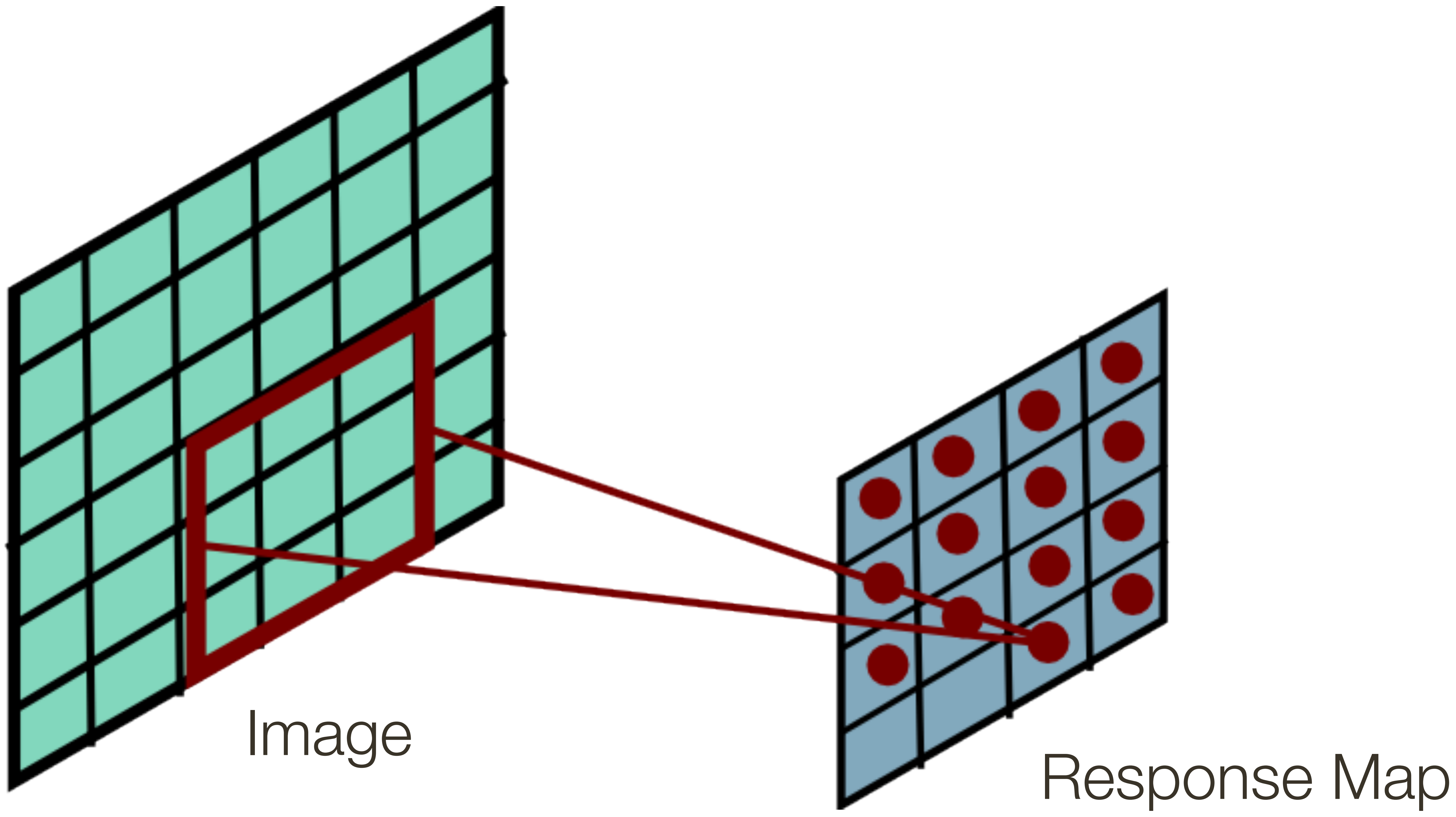
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



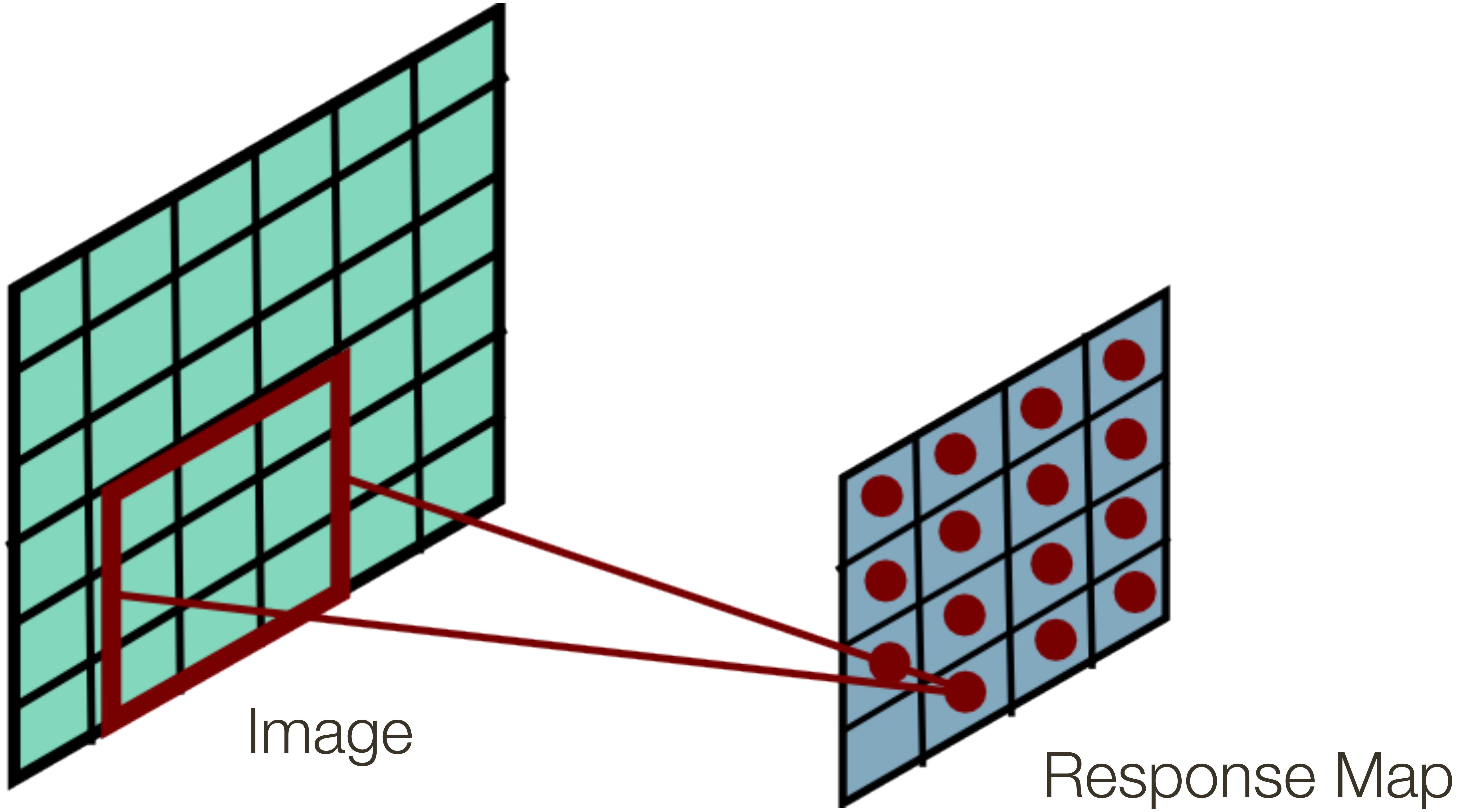
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



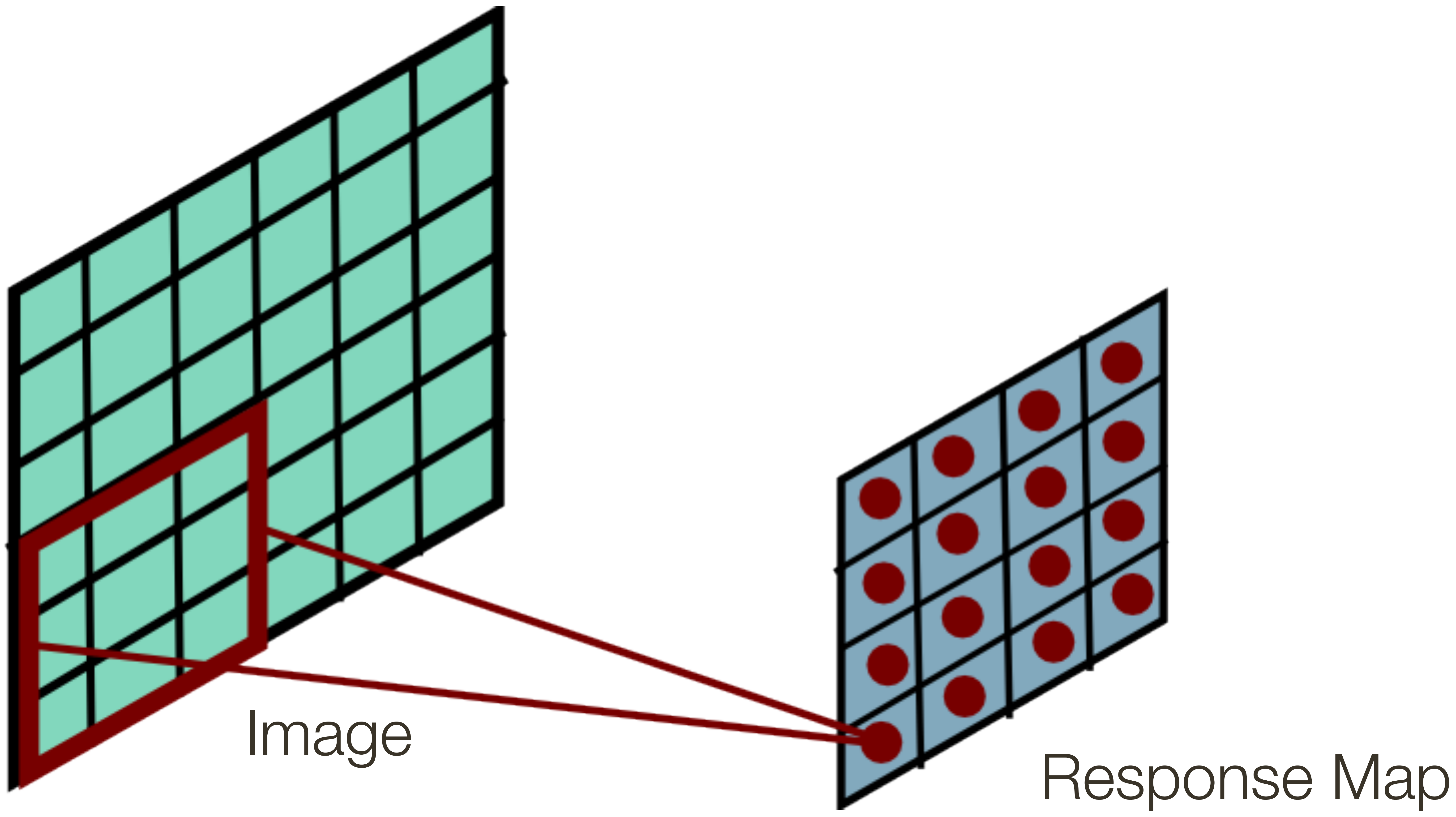
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



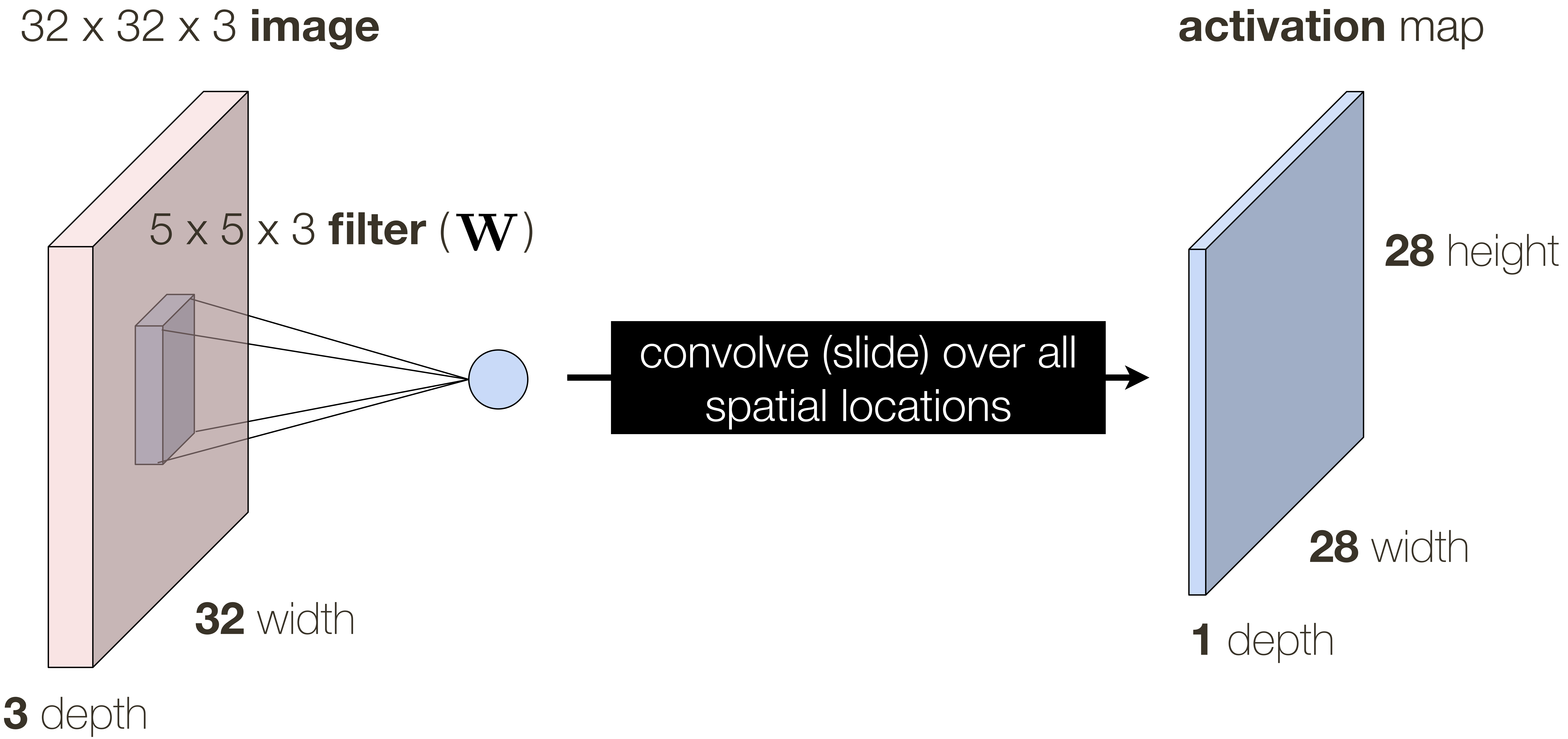
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



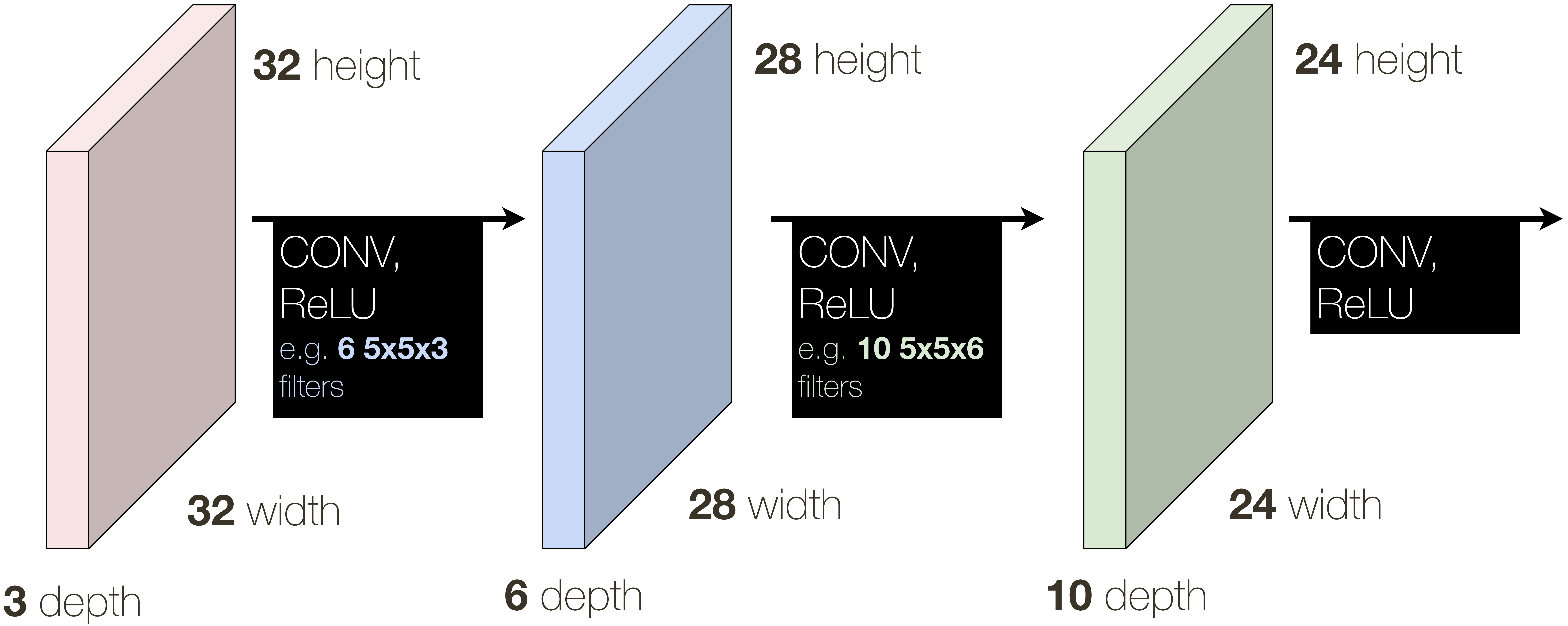
* slide from Marc'Aurelio Renzato

Last time: Convolutional Layer



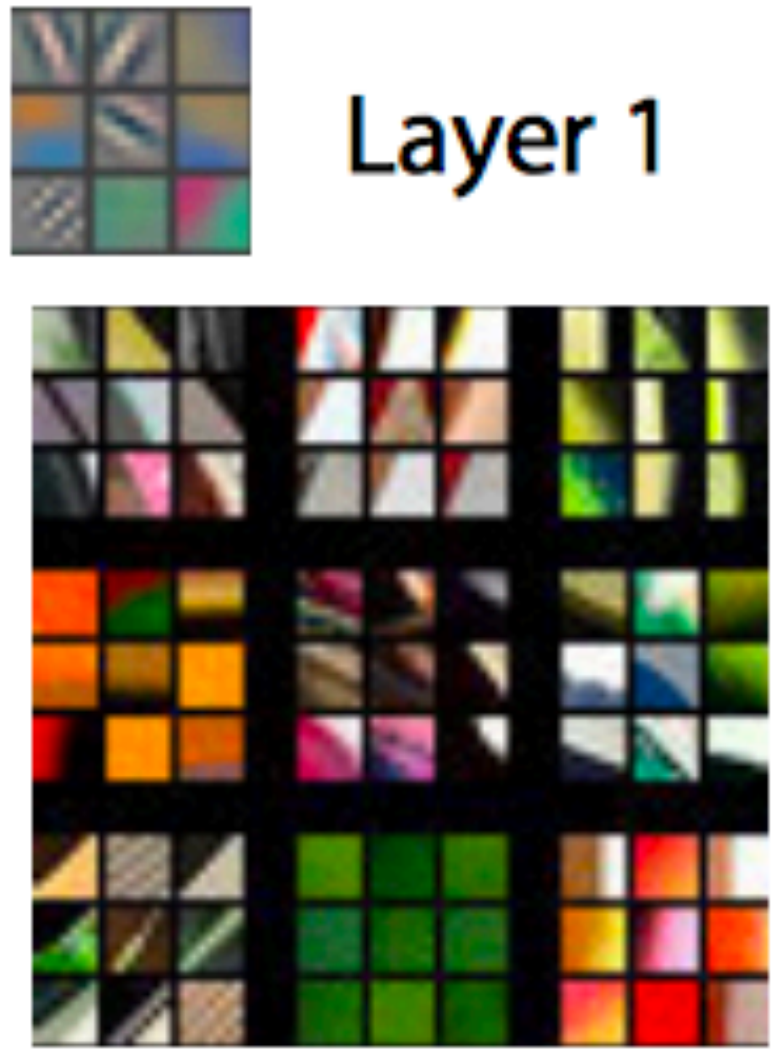
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

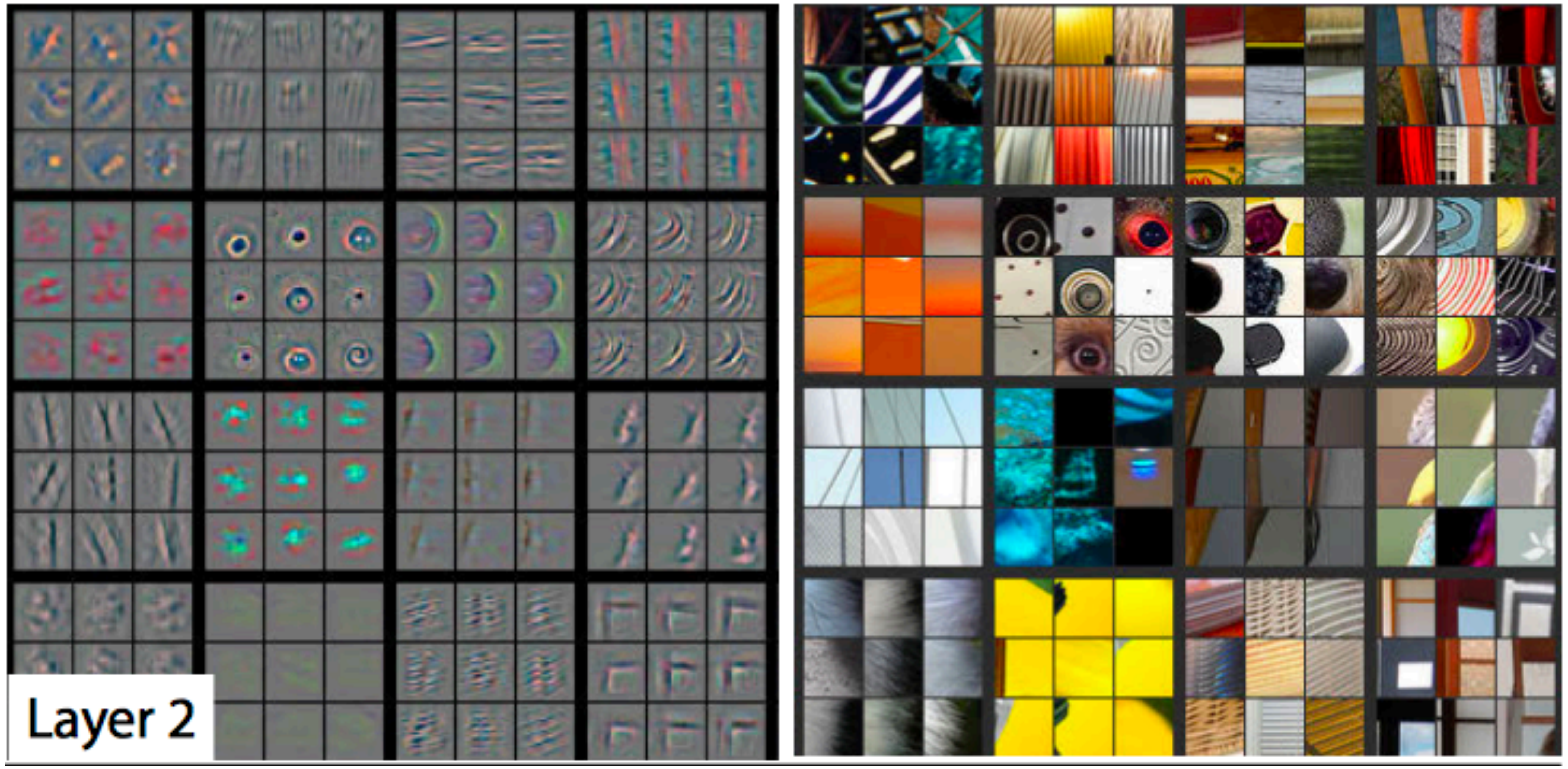
Last time: **Convolutional** Neural Network (ConvNet)



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

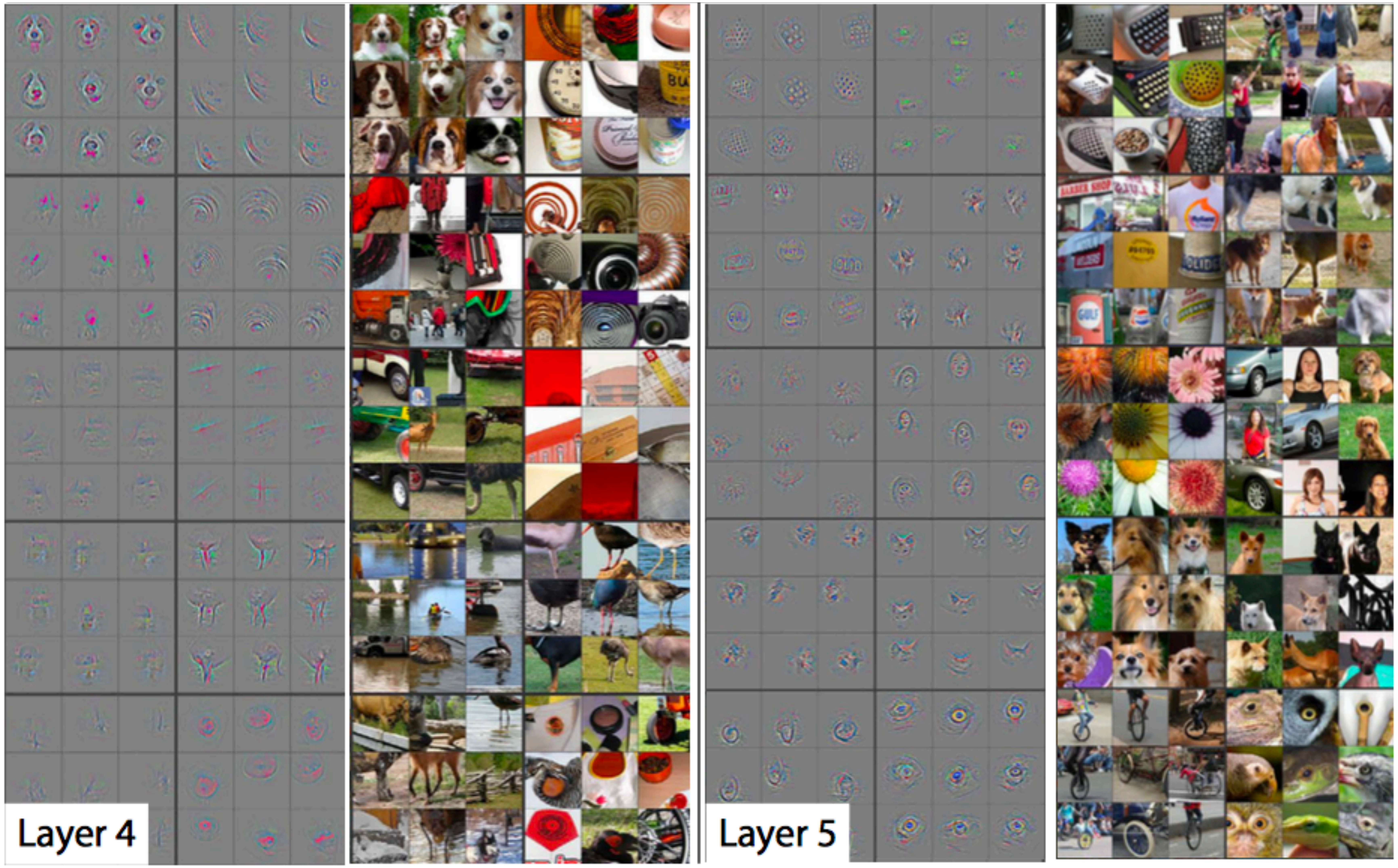
What **filters** do networks learn?





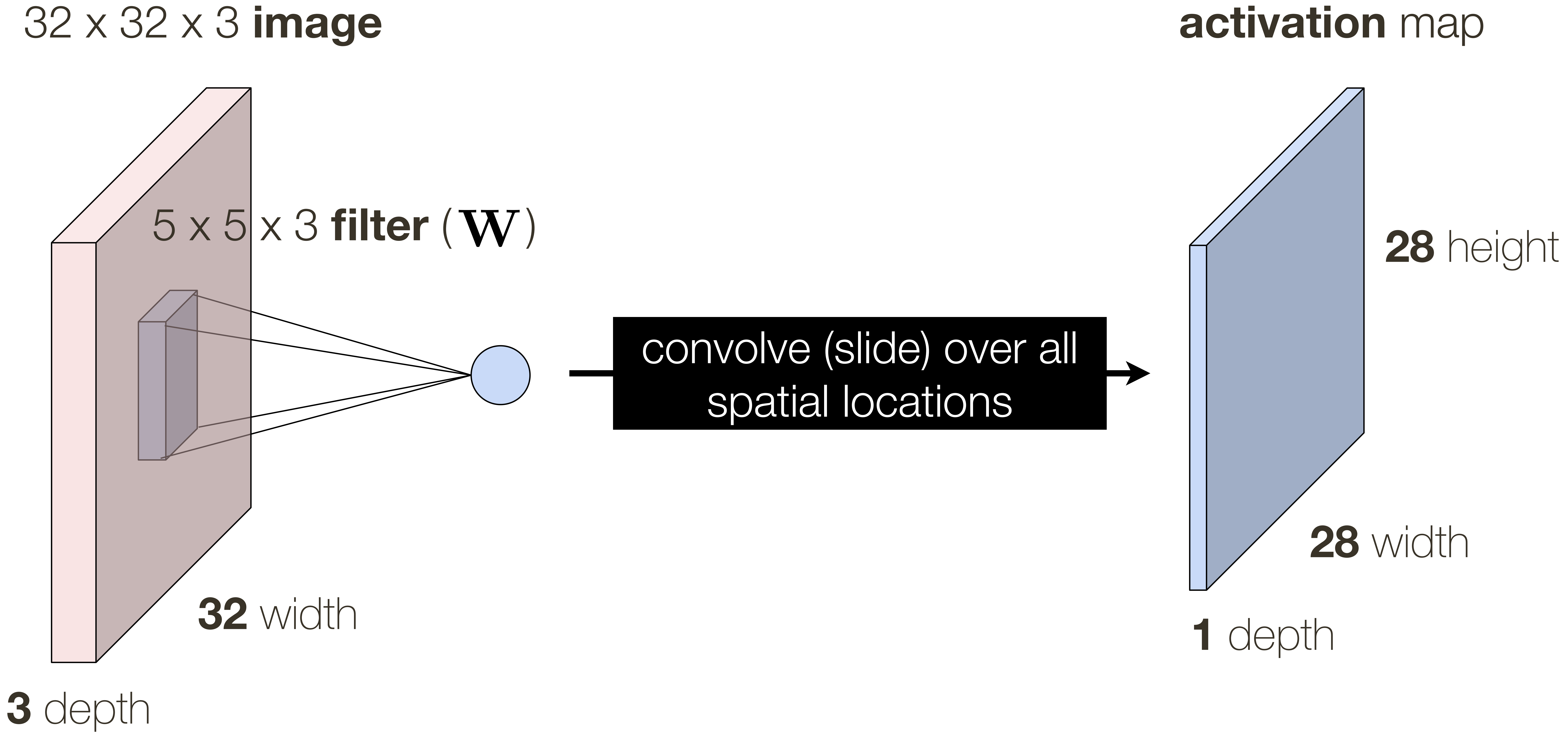
[Zeiler and Fergus, 2013]

What **filters** do networks learn?



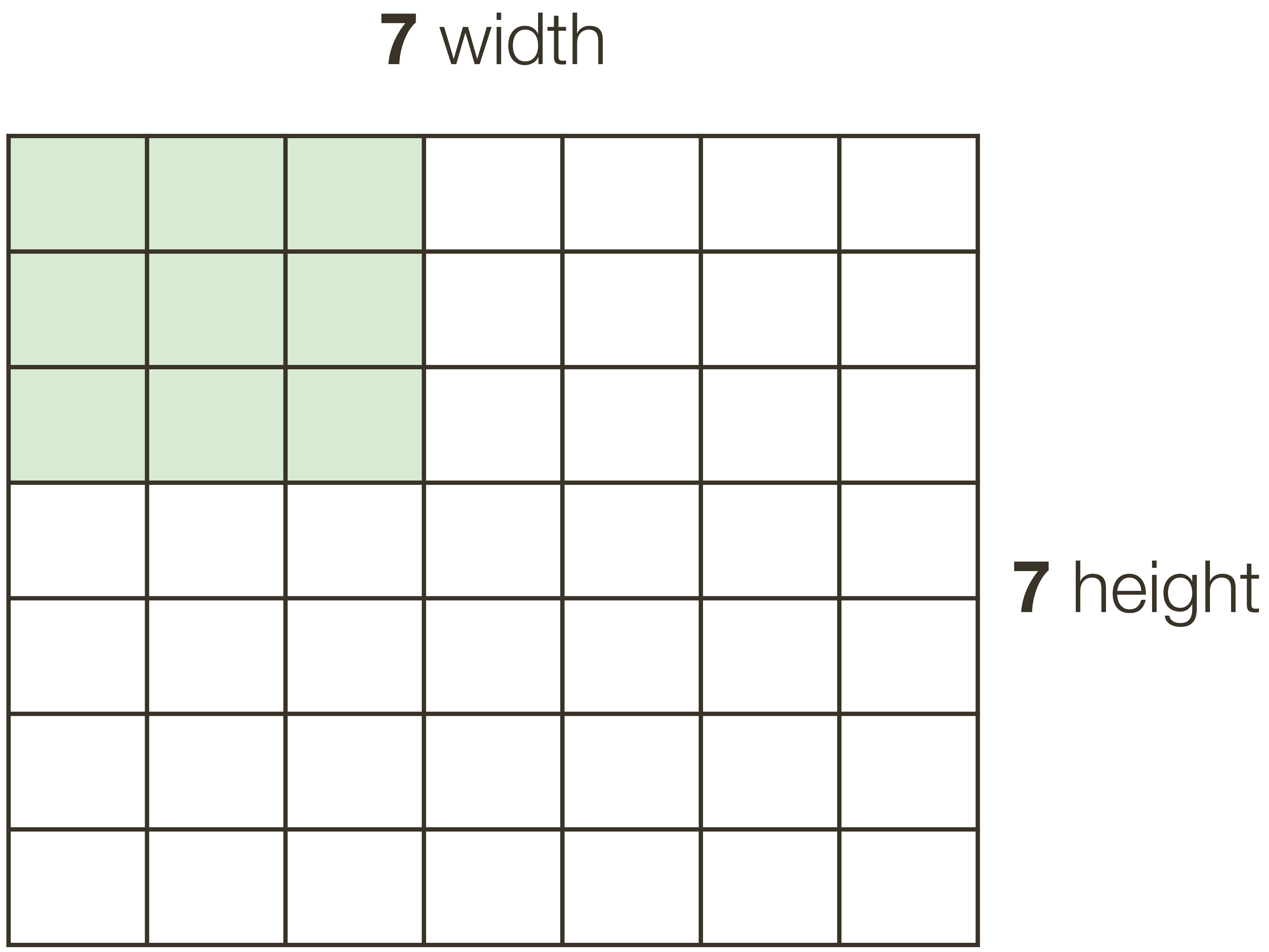
[Zeiler and Fergus, 2013]

Convolutional Layer: Closer Look at **Spatial Dimensions**



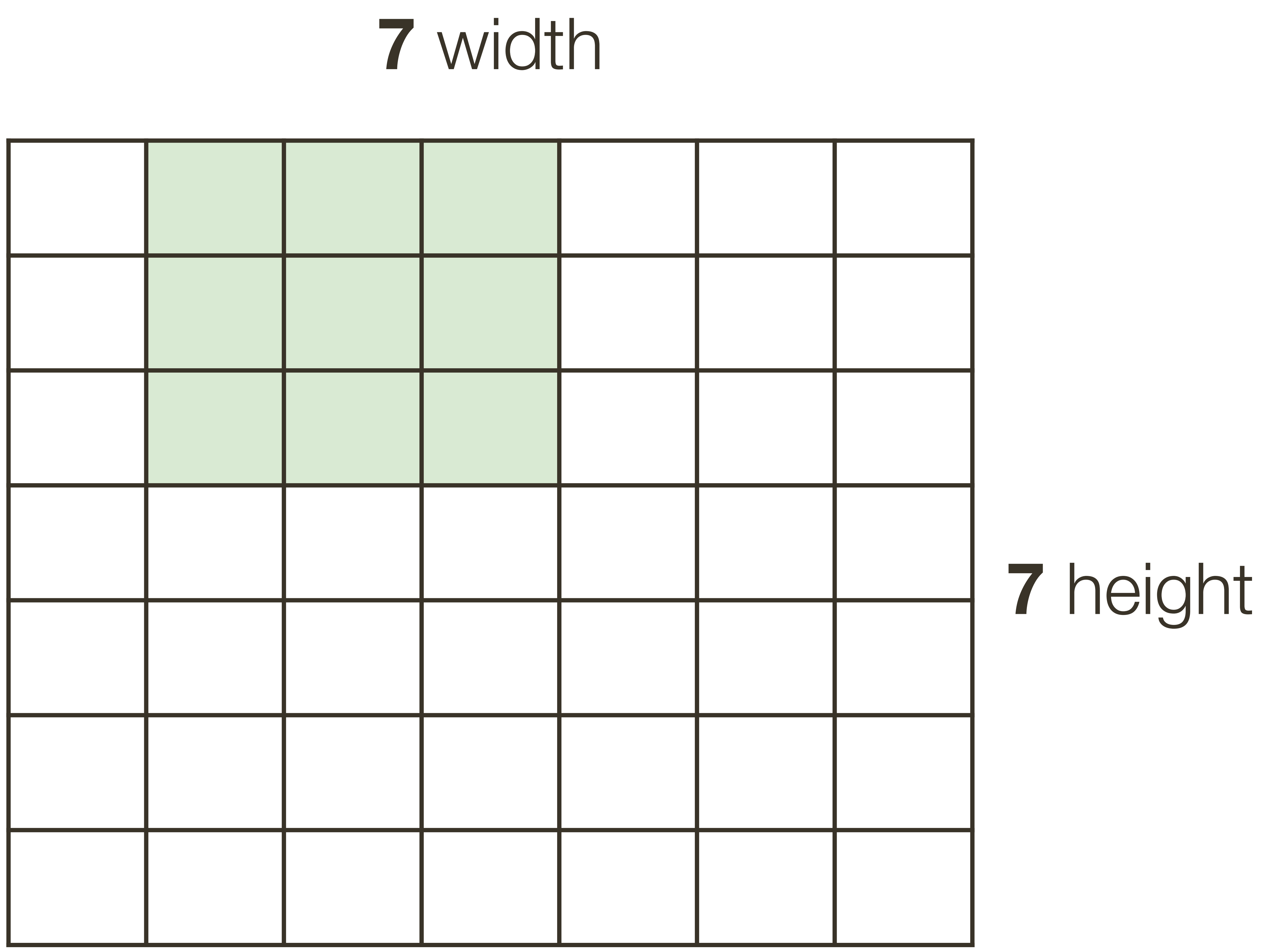
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter

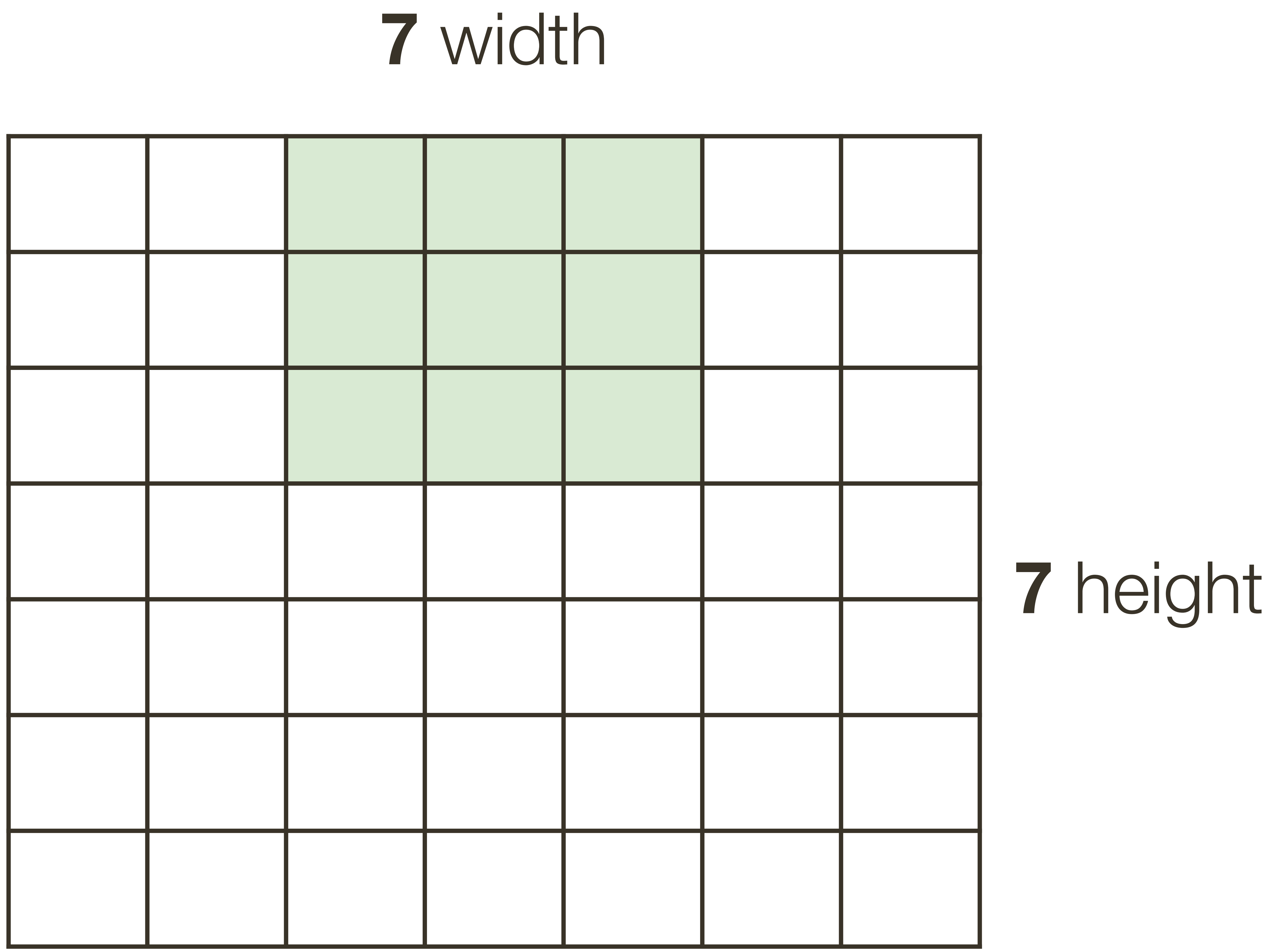
Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

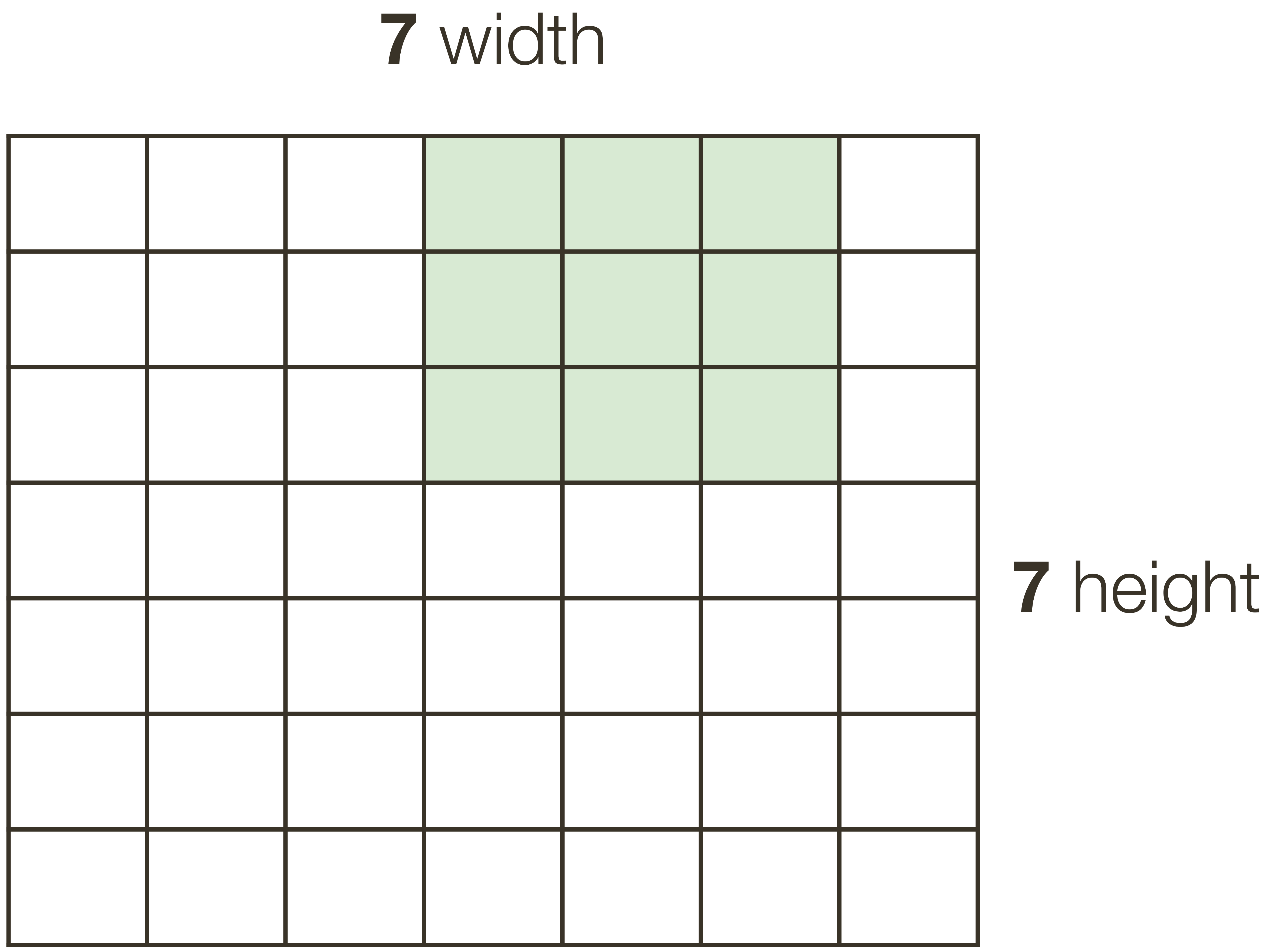
Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

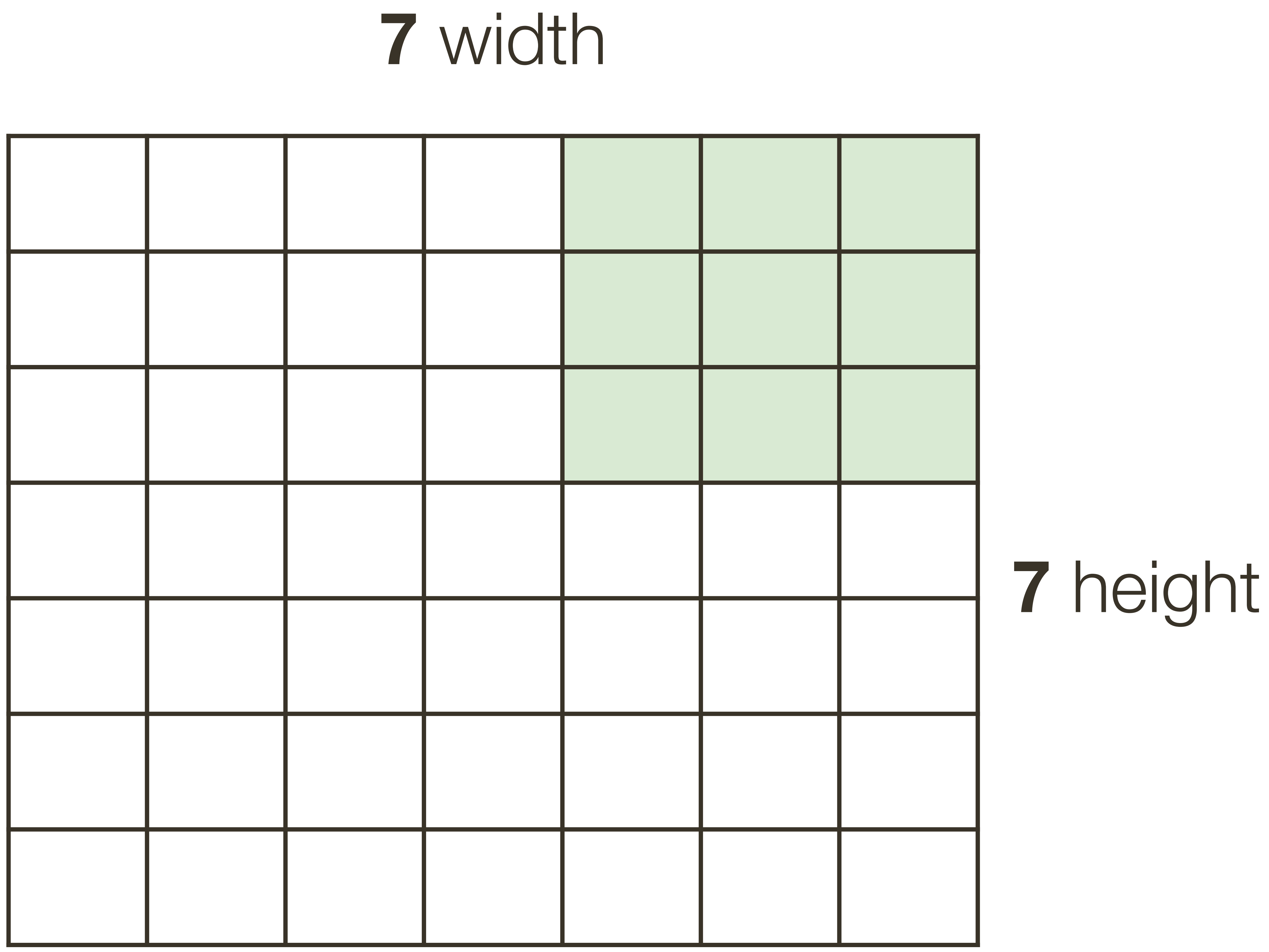
Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter

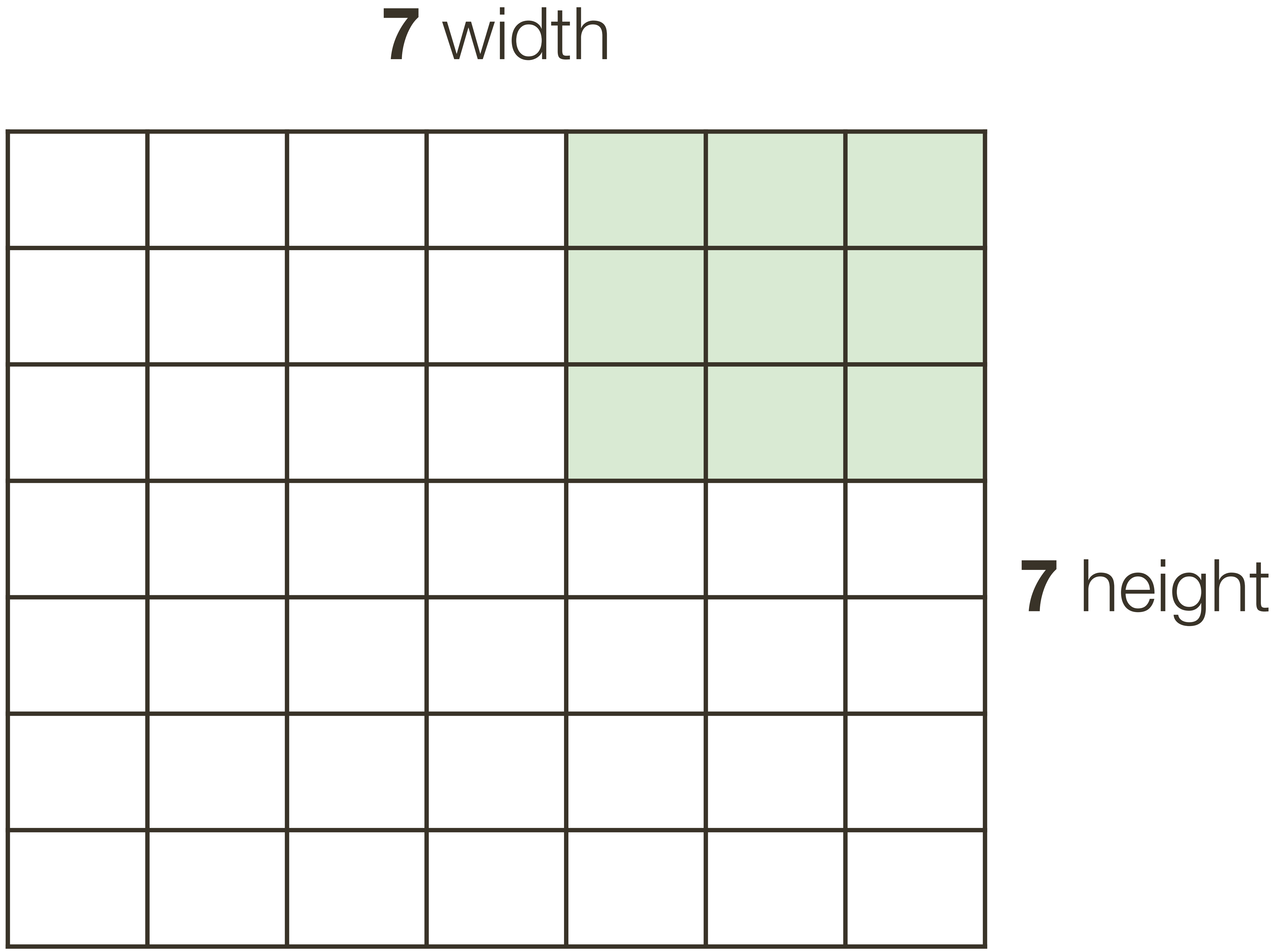
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter

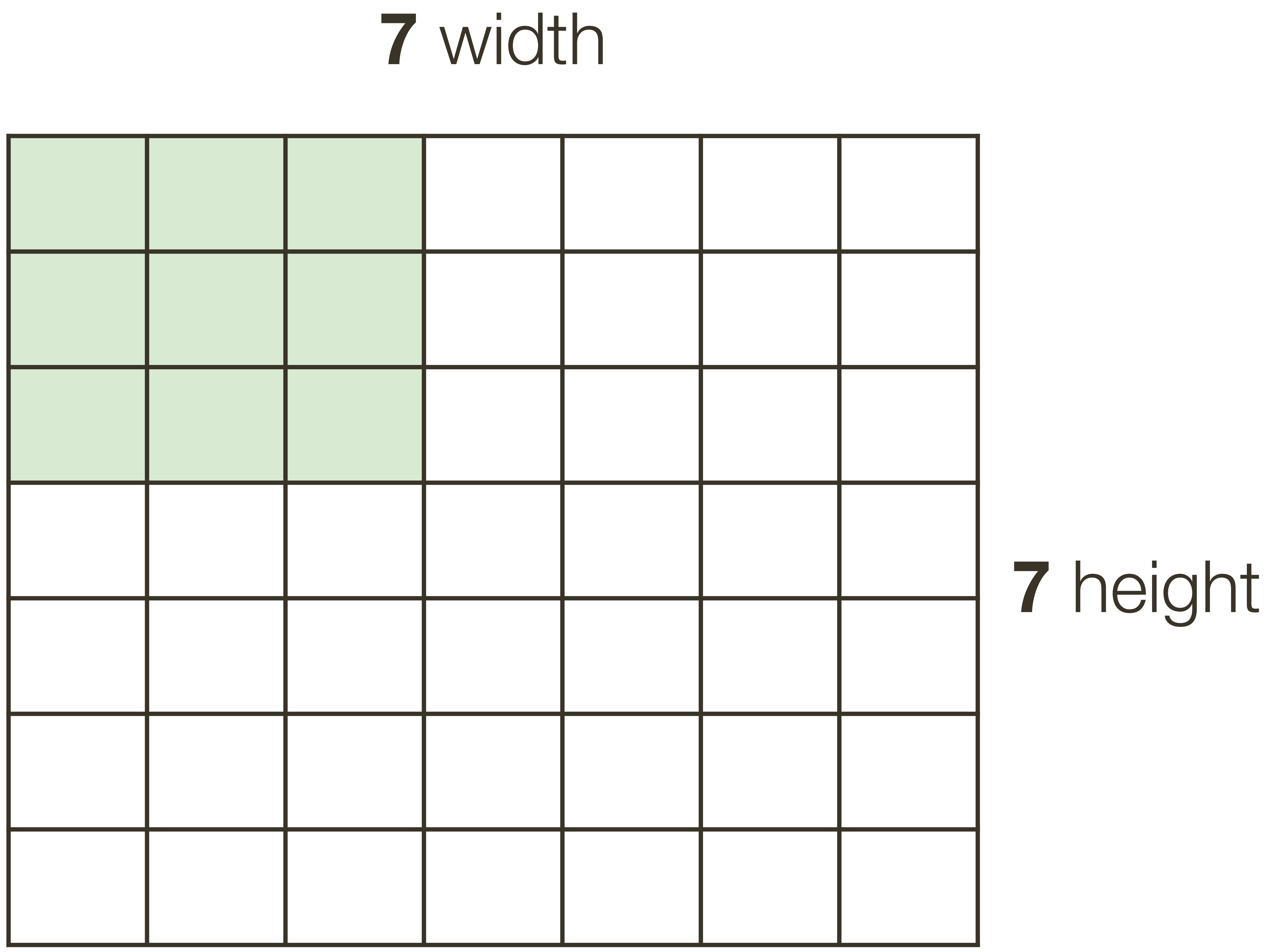
Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter

=> **5 x 5 output**

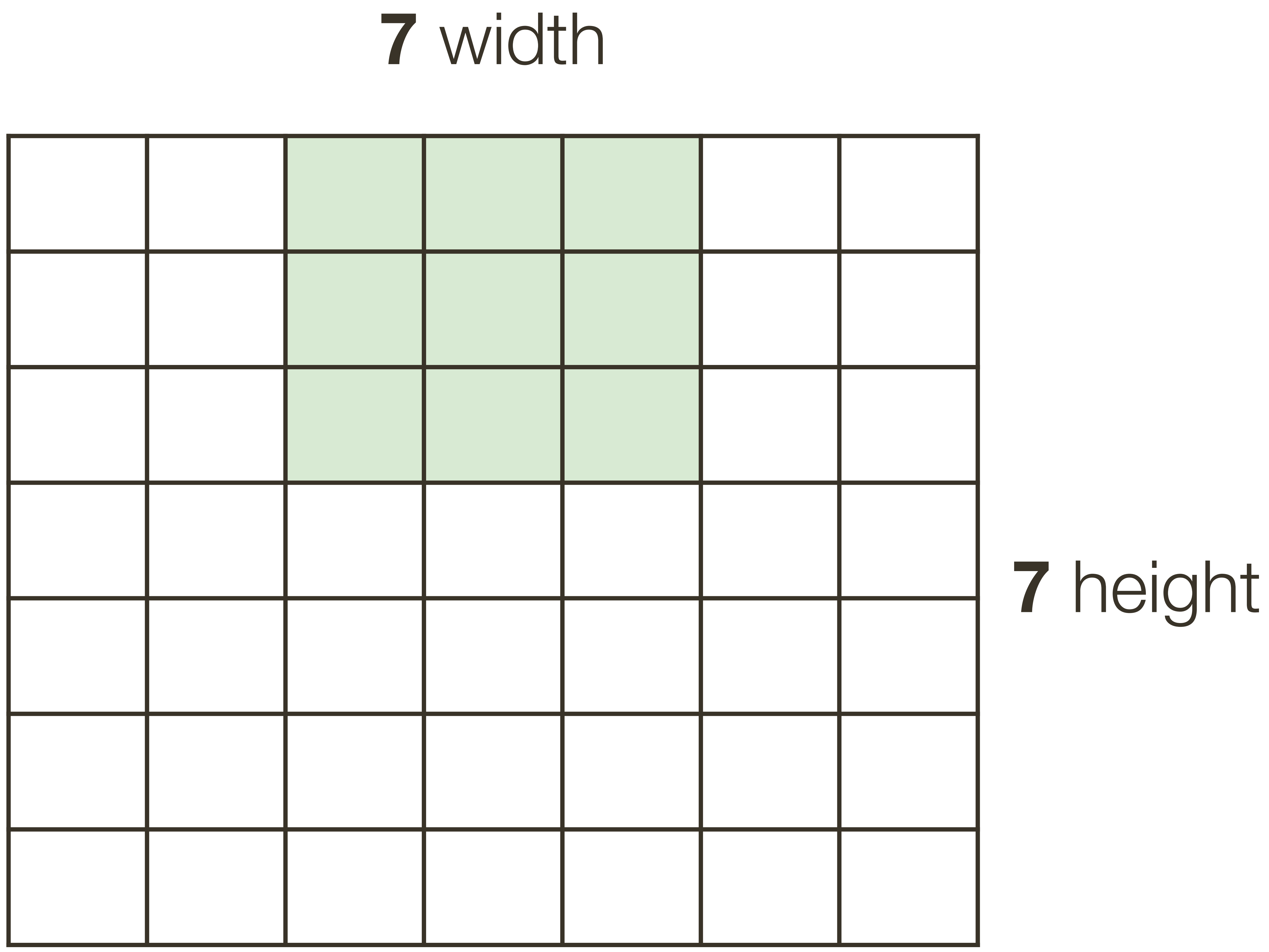
Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 2**)

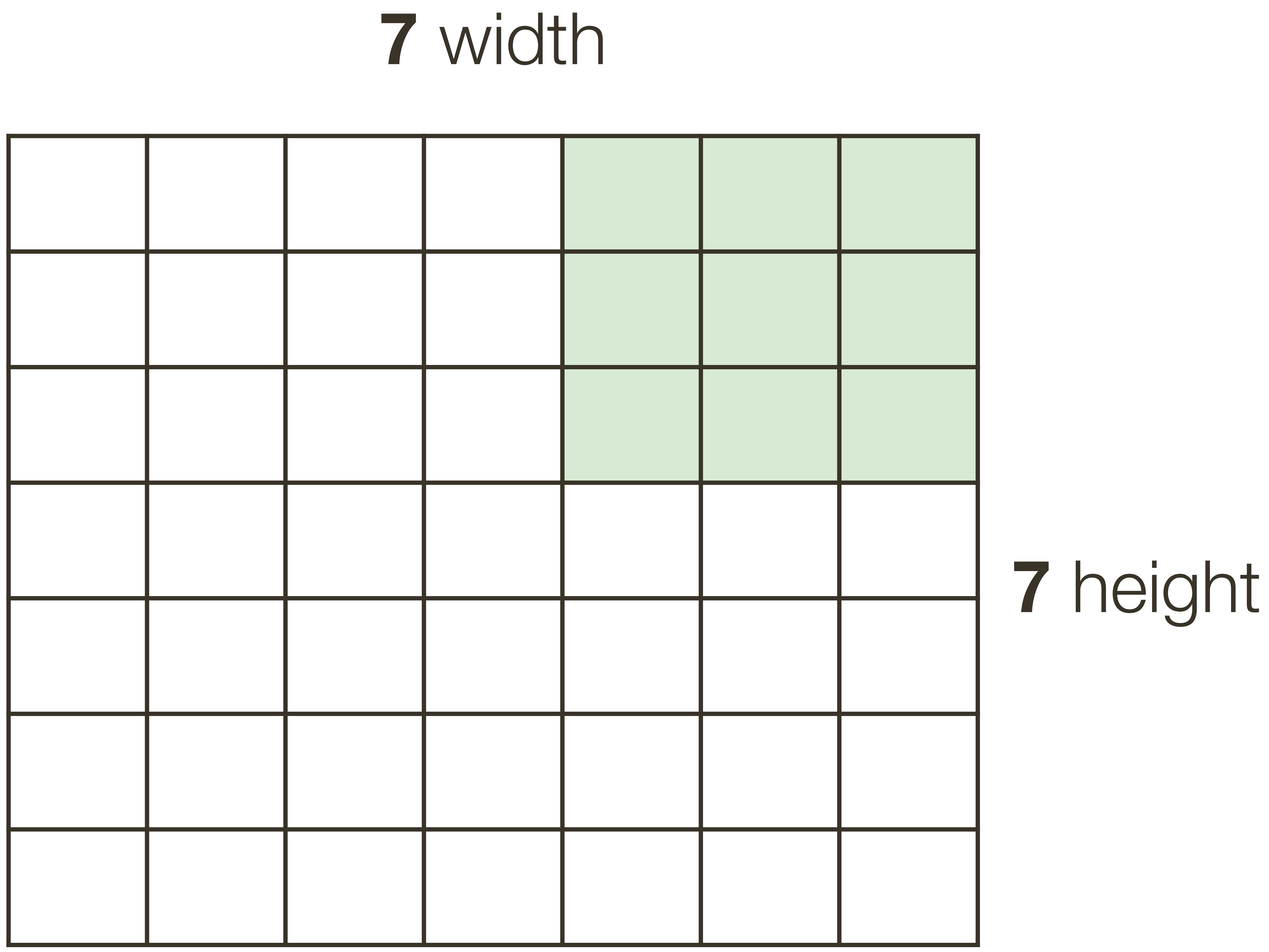
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 2**)

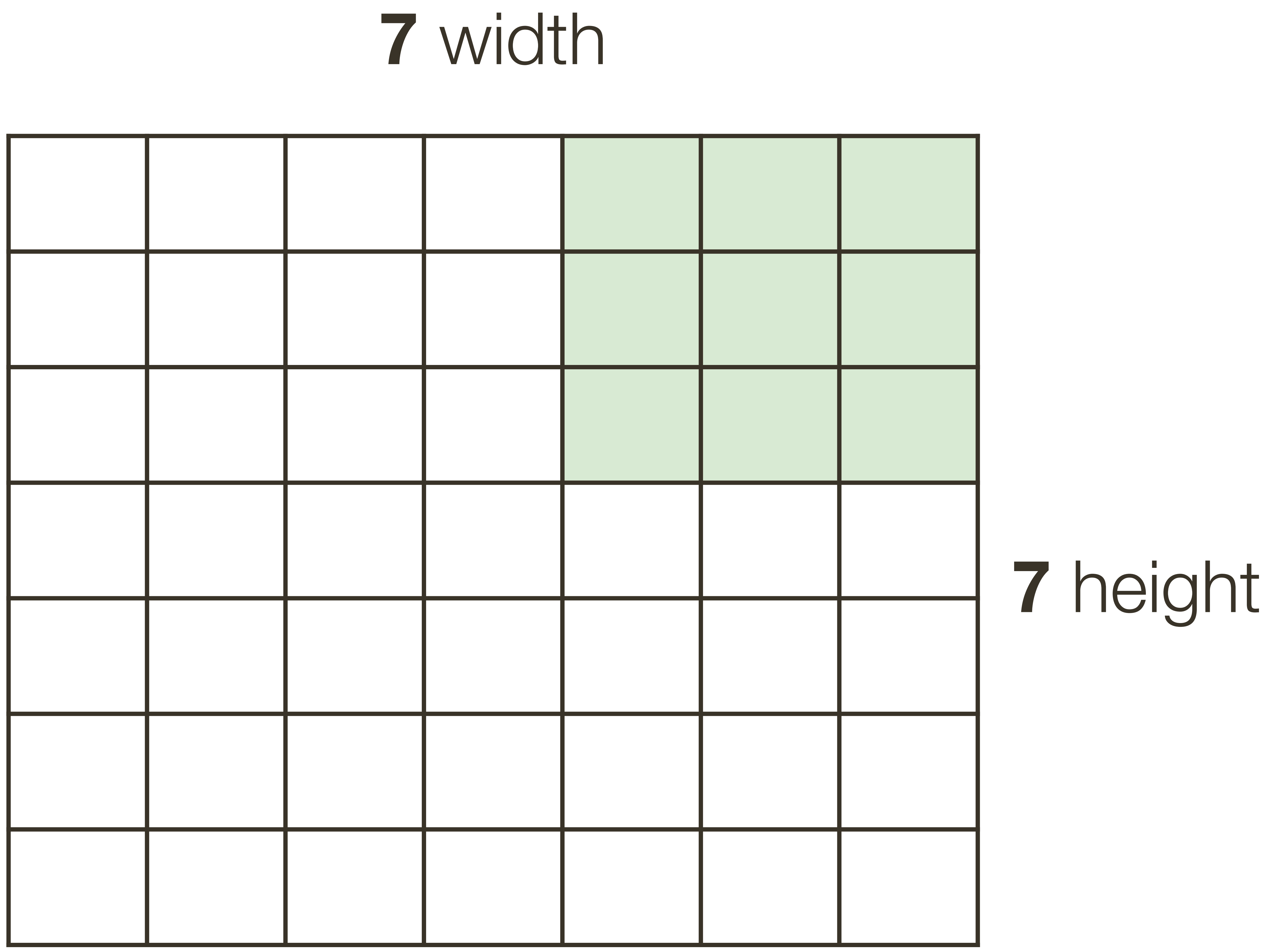
Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 2**)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

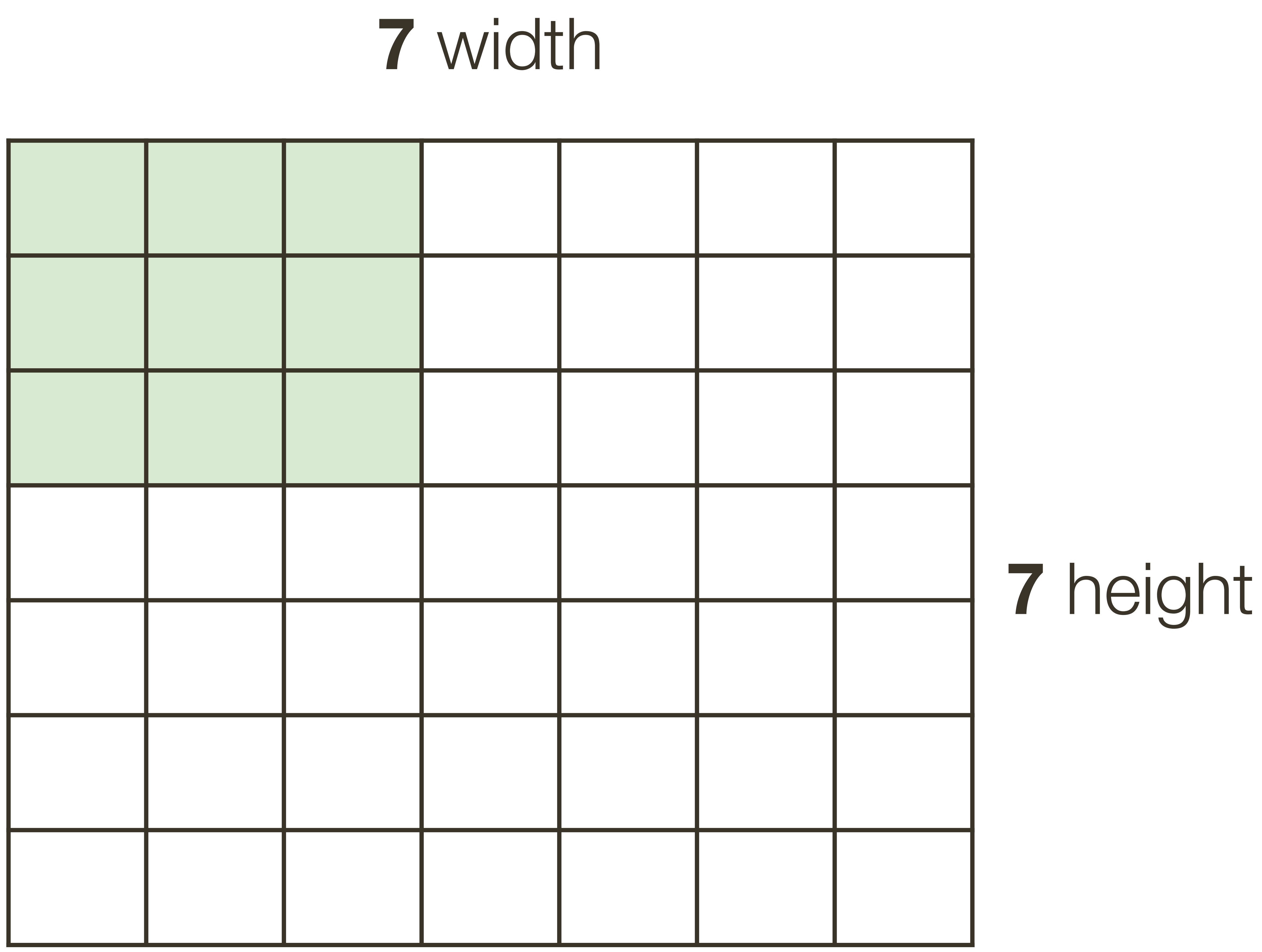
Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 2**)

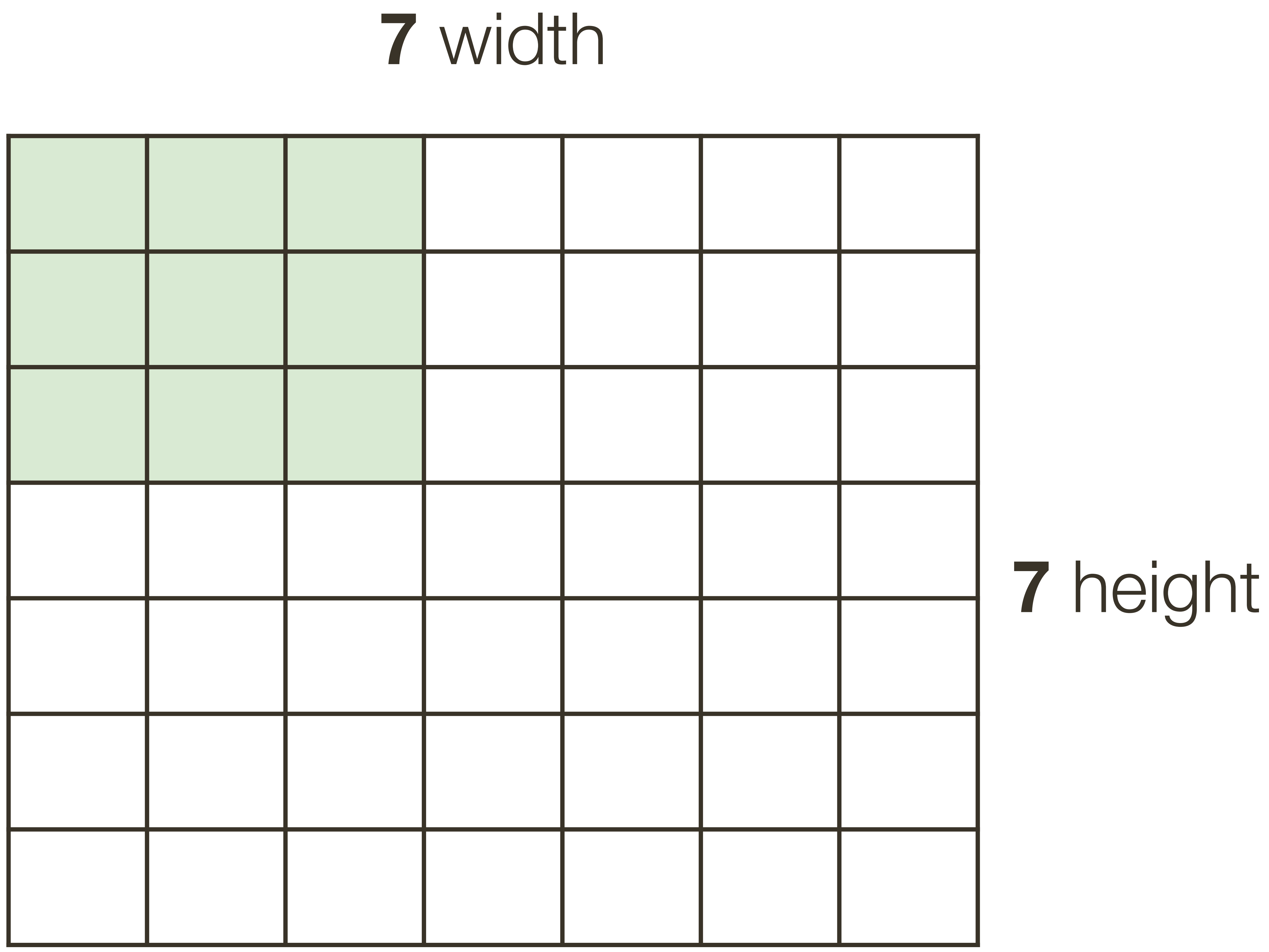
=> **3 x 3 output**

Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 3**)

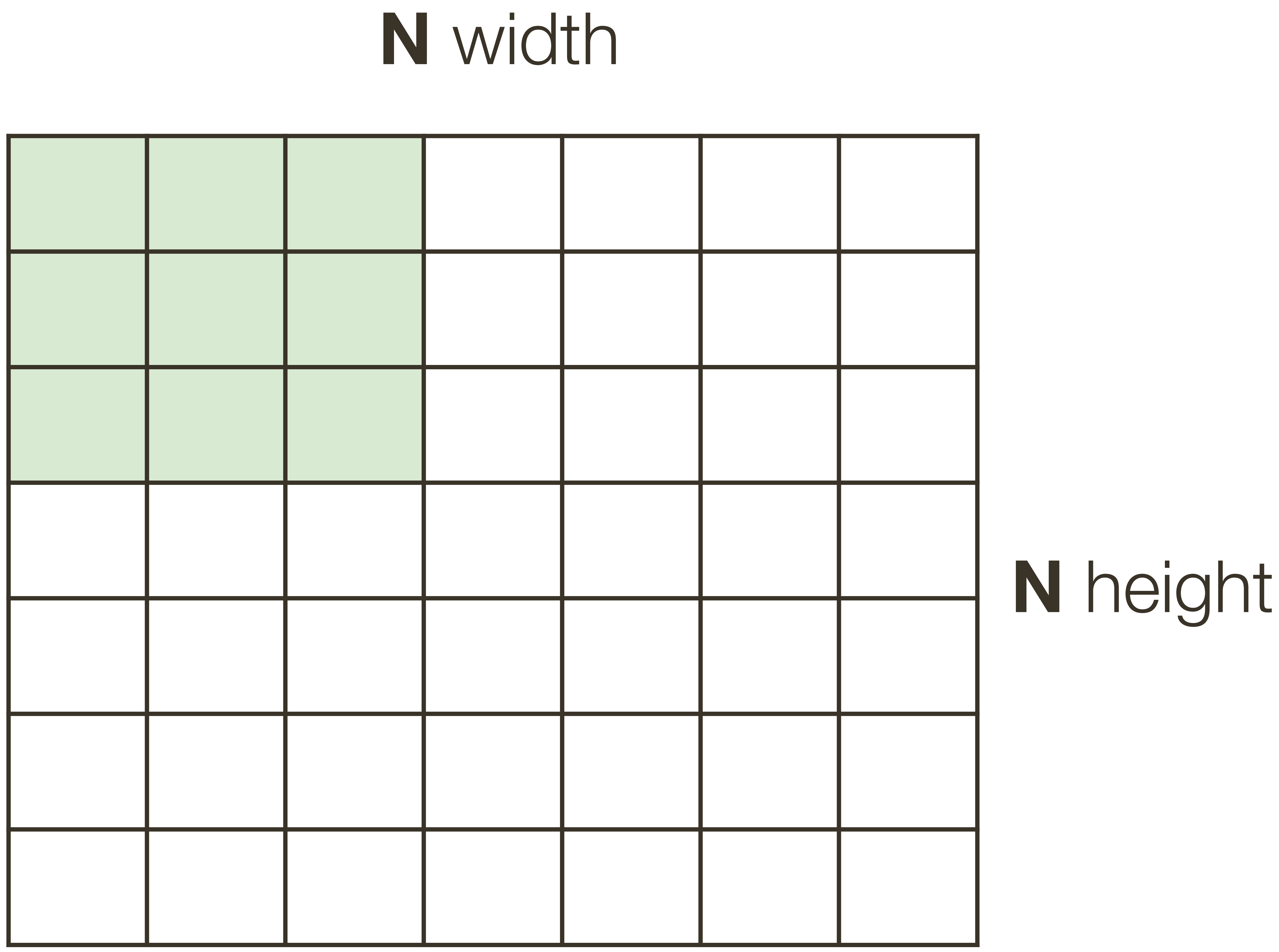
Convolutional Layer: Closer Look at **Spatial Dimensions**



7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 3**)

Does not fit! **Cannot apply** 3 x 3 filter on 7 x 7 image with stride 3

Convolutional Layer: Closer Look at **Spatial Dimensions**

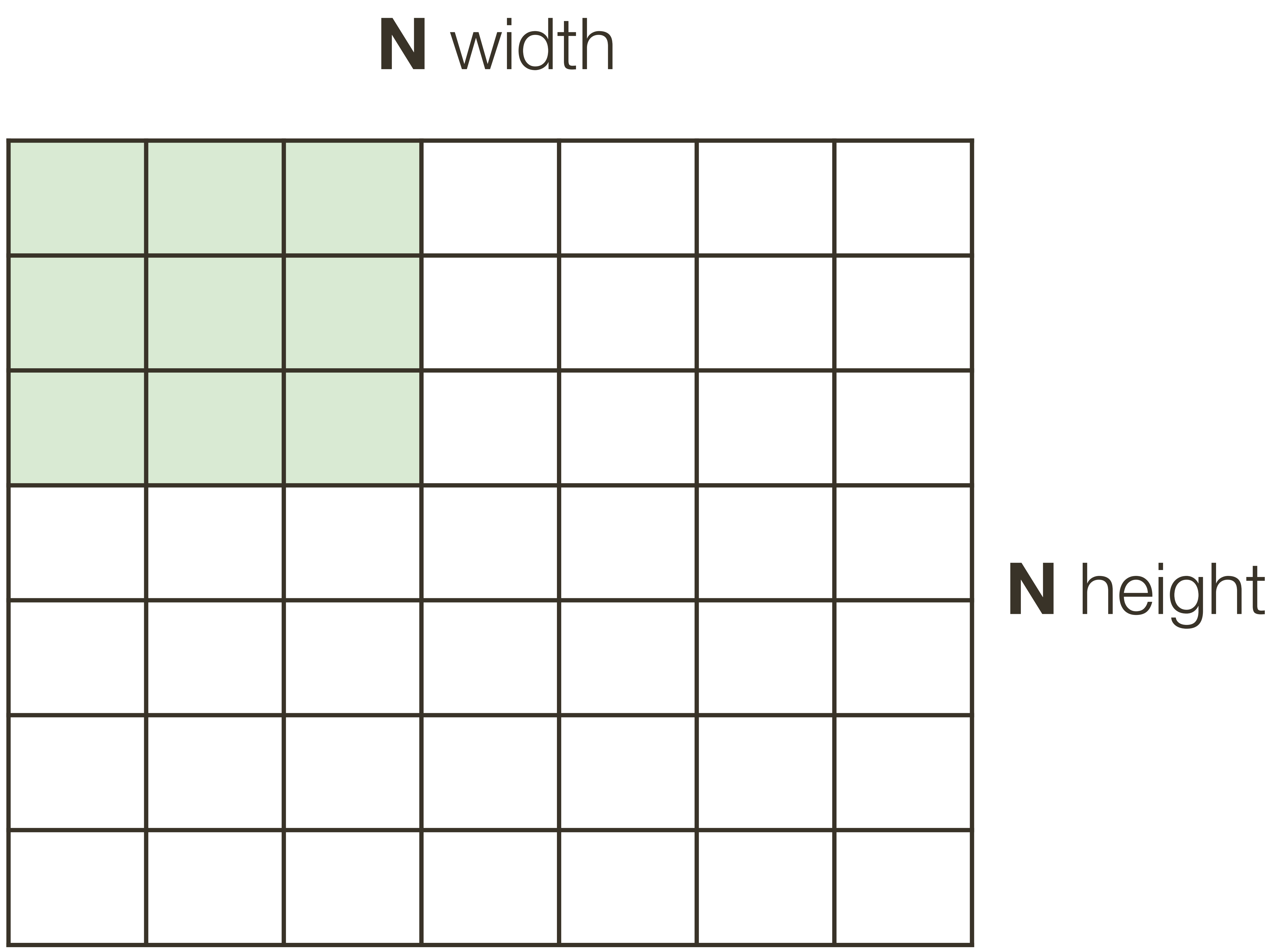


N x N input image (spatially)
F x F filter

Output size: (N-F) / stride + 1

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: Closer Look at **Spatial Dimensions**



N x N input image (spatially)
F x F filter

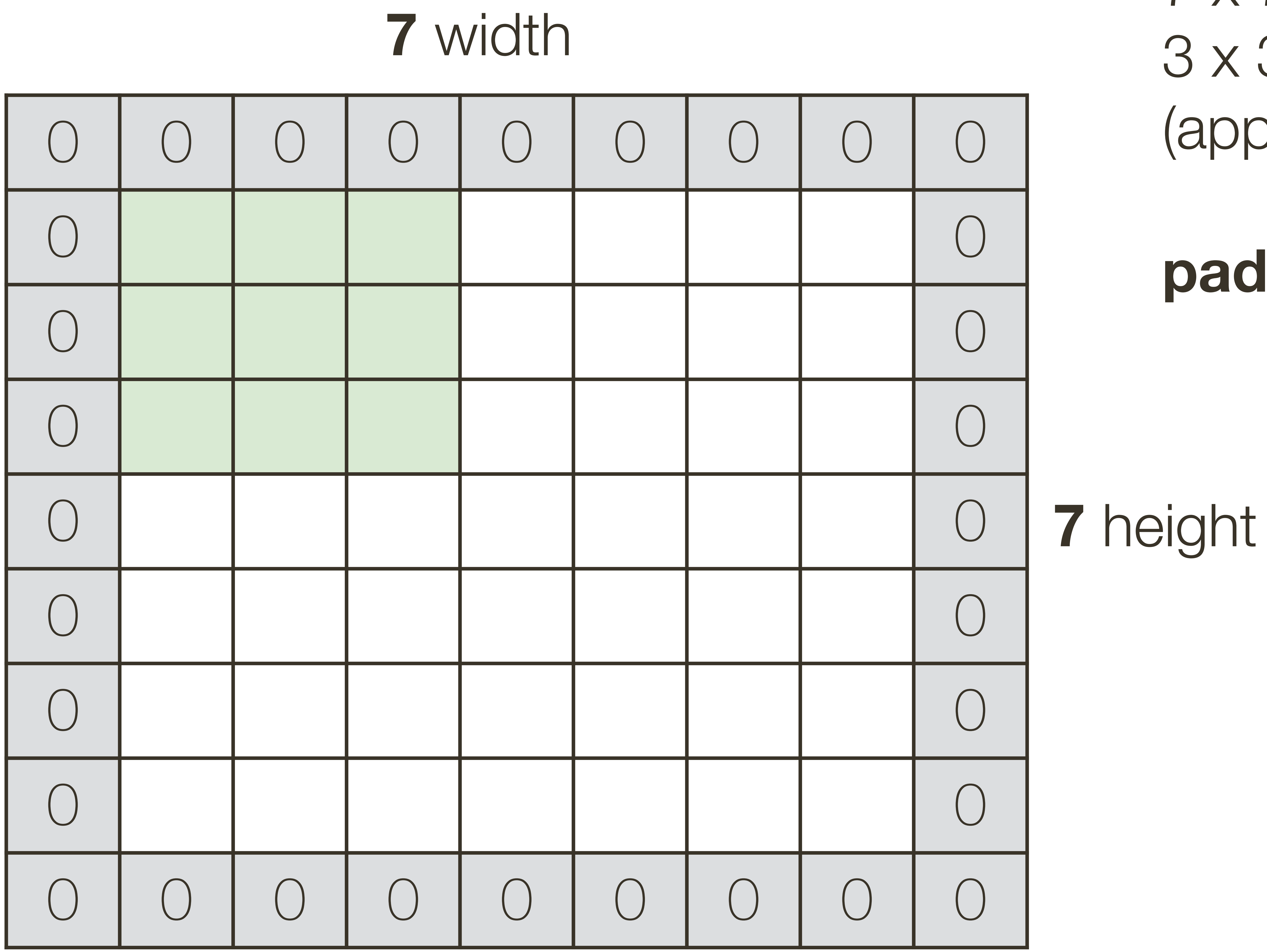
Output size: (N-F) / stride + 1

Example: N = 7, F = 3

stride 1 => (7-3)/1+1 = 5
stride 2 => (7-3)/2+1 = 3
stride 3 => (7-3)/3+1 = **2.33**

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

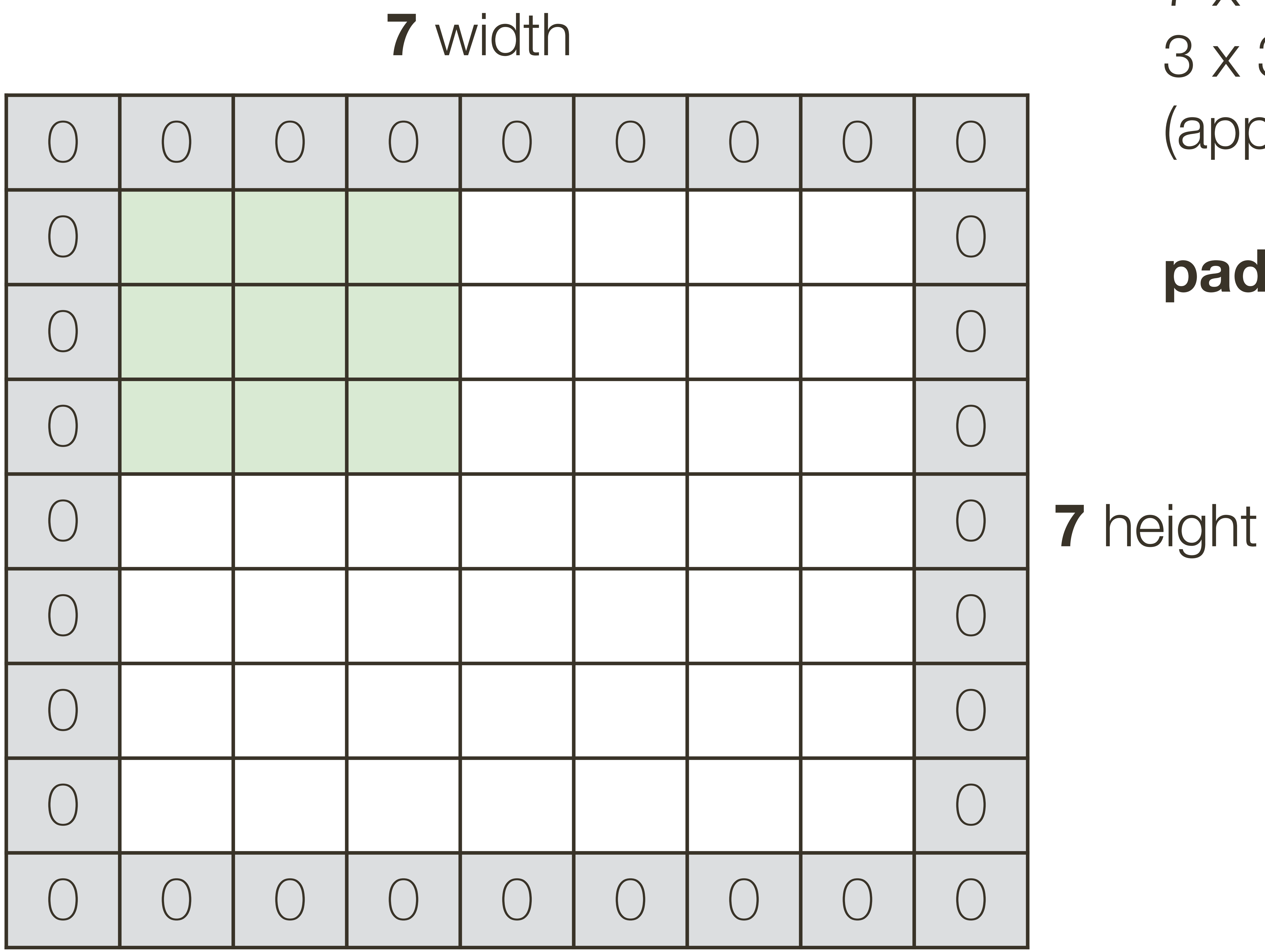
Convolutional Layer: **Border padding**



7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 1**)

pad with 1 pixel border

Convolutional Layer: **Border padding**



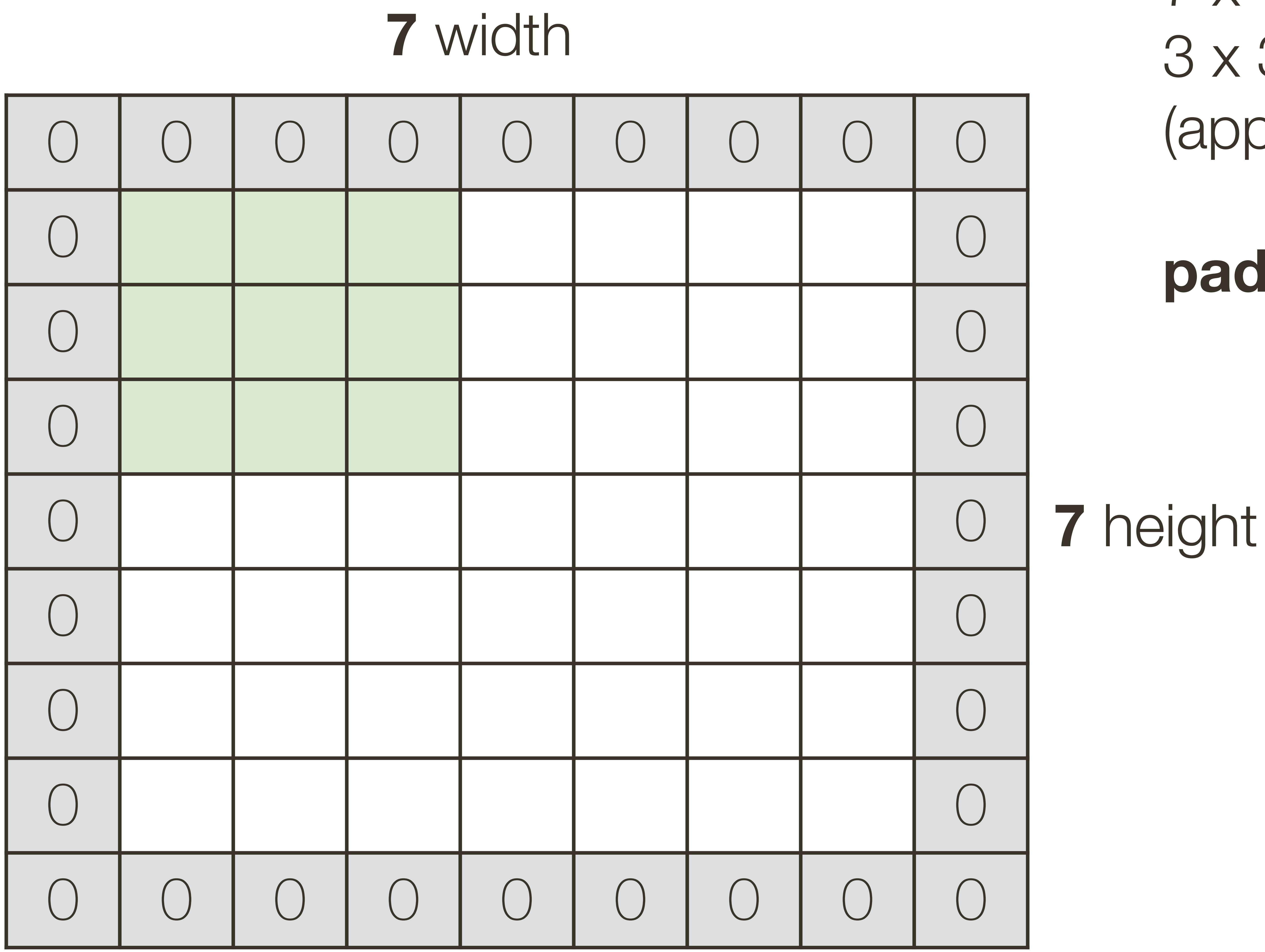
7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 1**)

pad with 1 pixel border

Output size: 7 x 7

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: **Border padding**

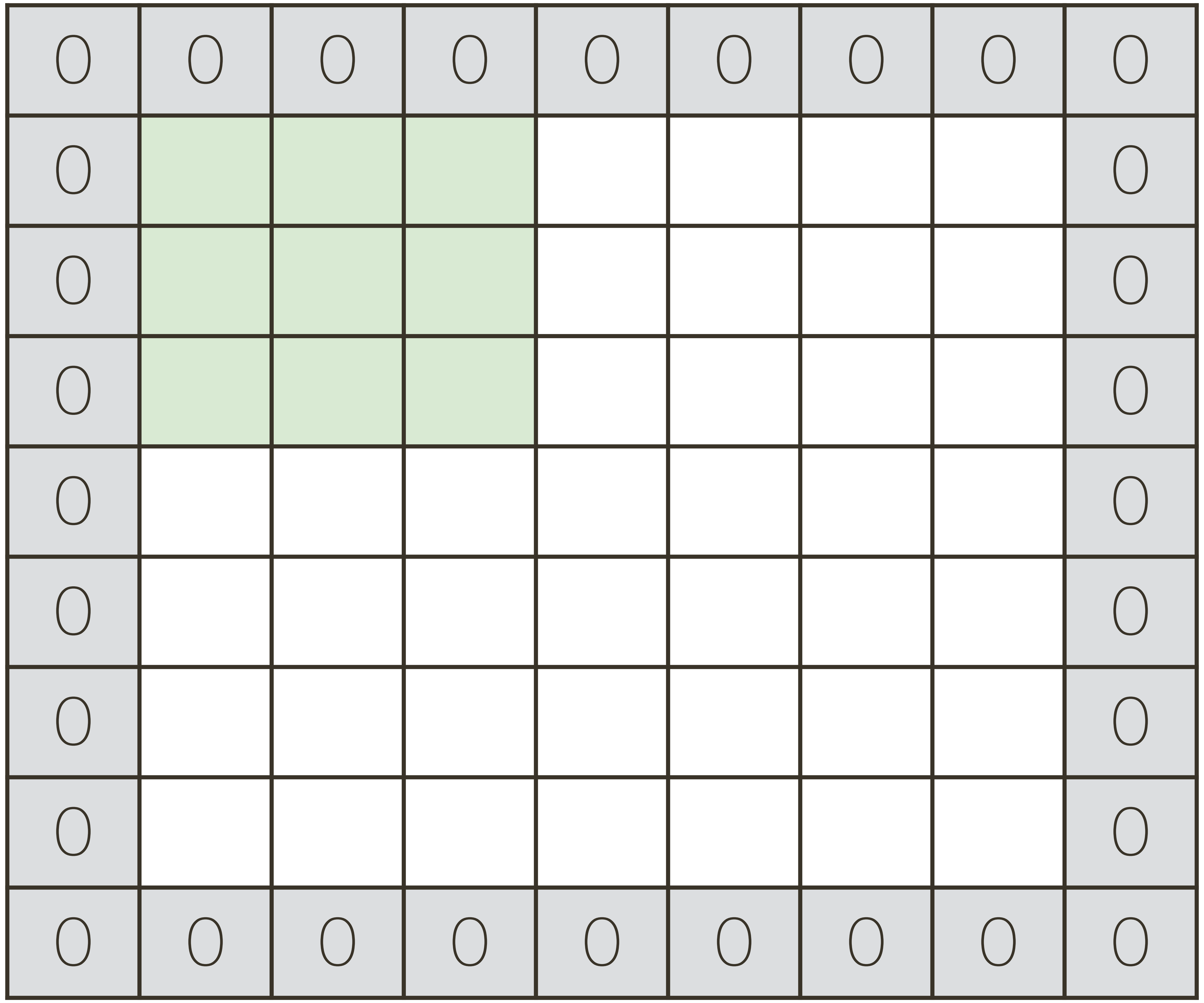


7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 3**)

pad with 1 pixel border

Convolutional Layer: **Border padding**

7 width



7 height

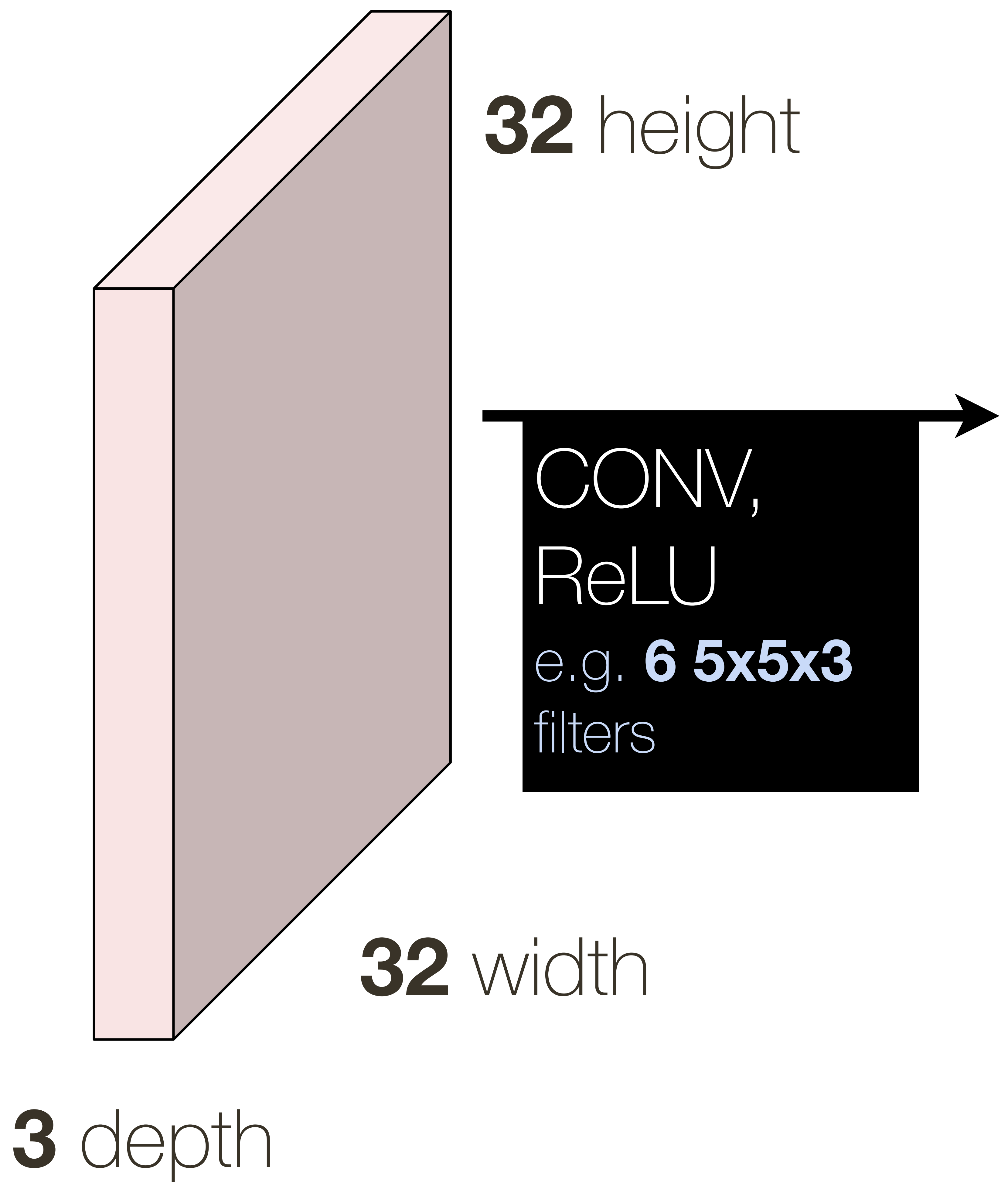
7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 3**)

pad with 1 pixel border

Example: N = 7, F = 3

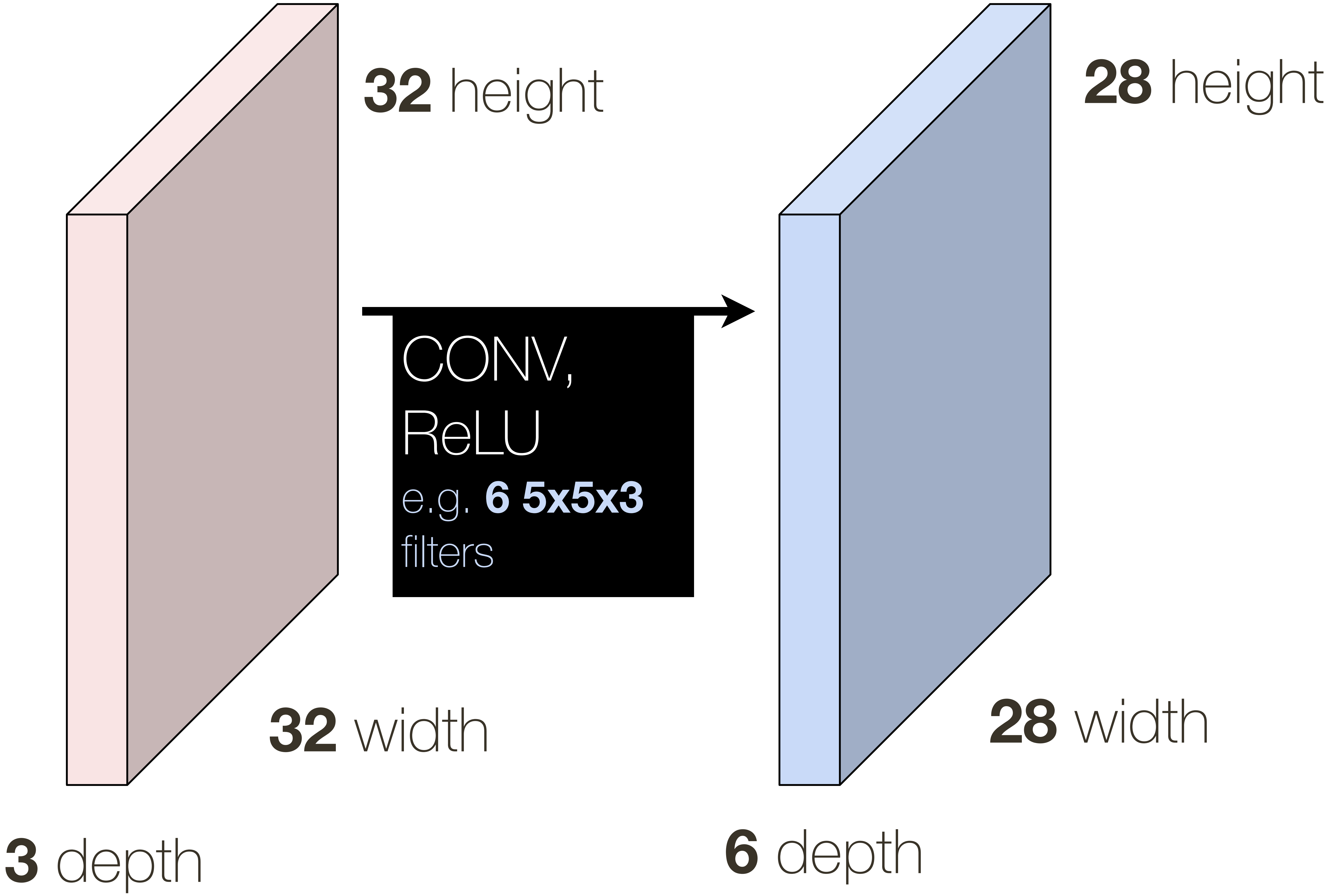
stride 1 => (9-3)/1+1 = 7
stride 2 => (9-3)/2+1 = 4
stride 3 => (9-3)/3+1 = **3**

Convolutional Neural Network (ConvNet)



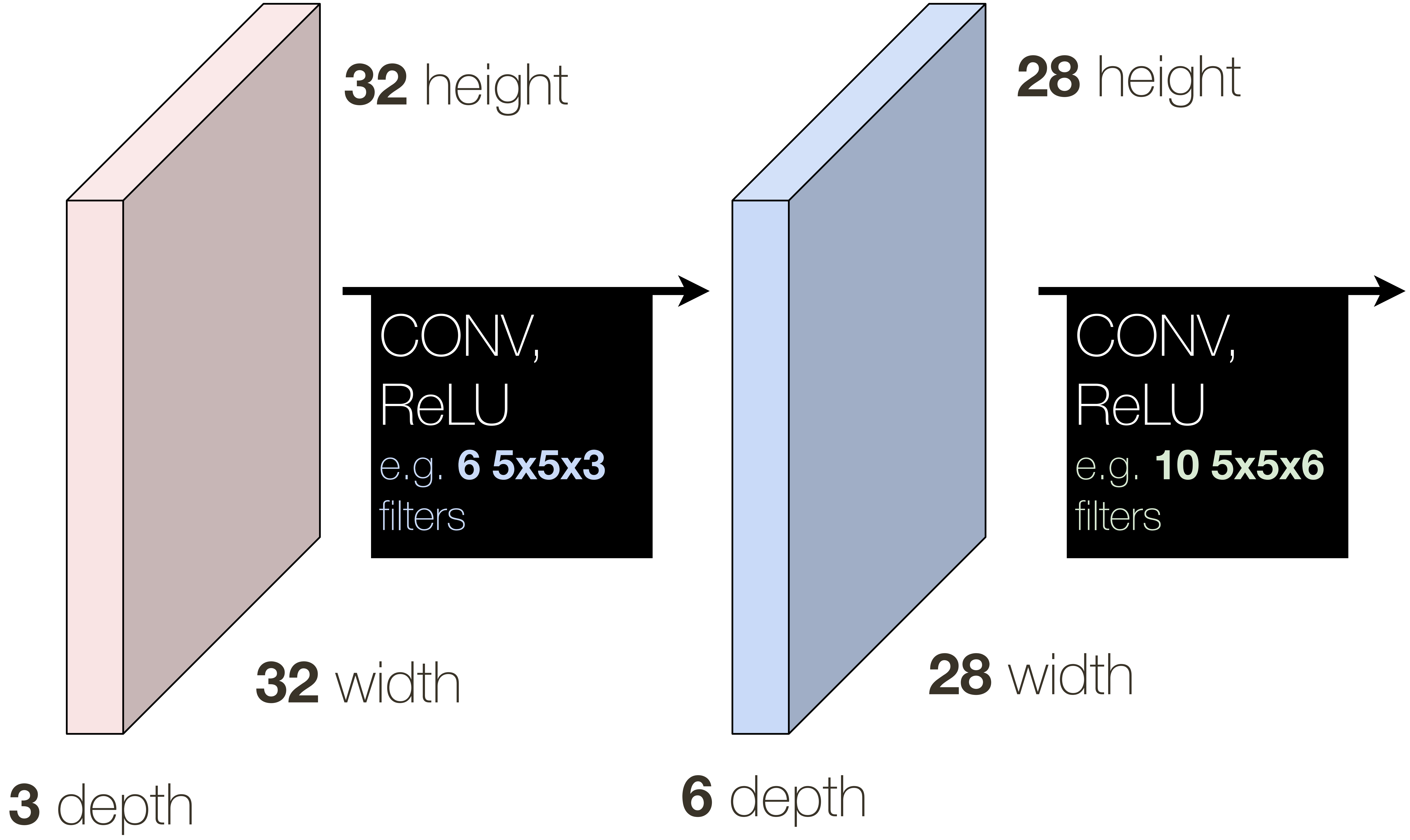
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)

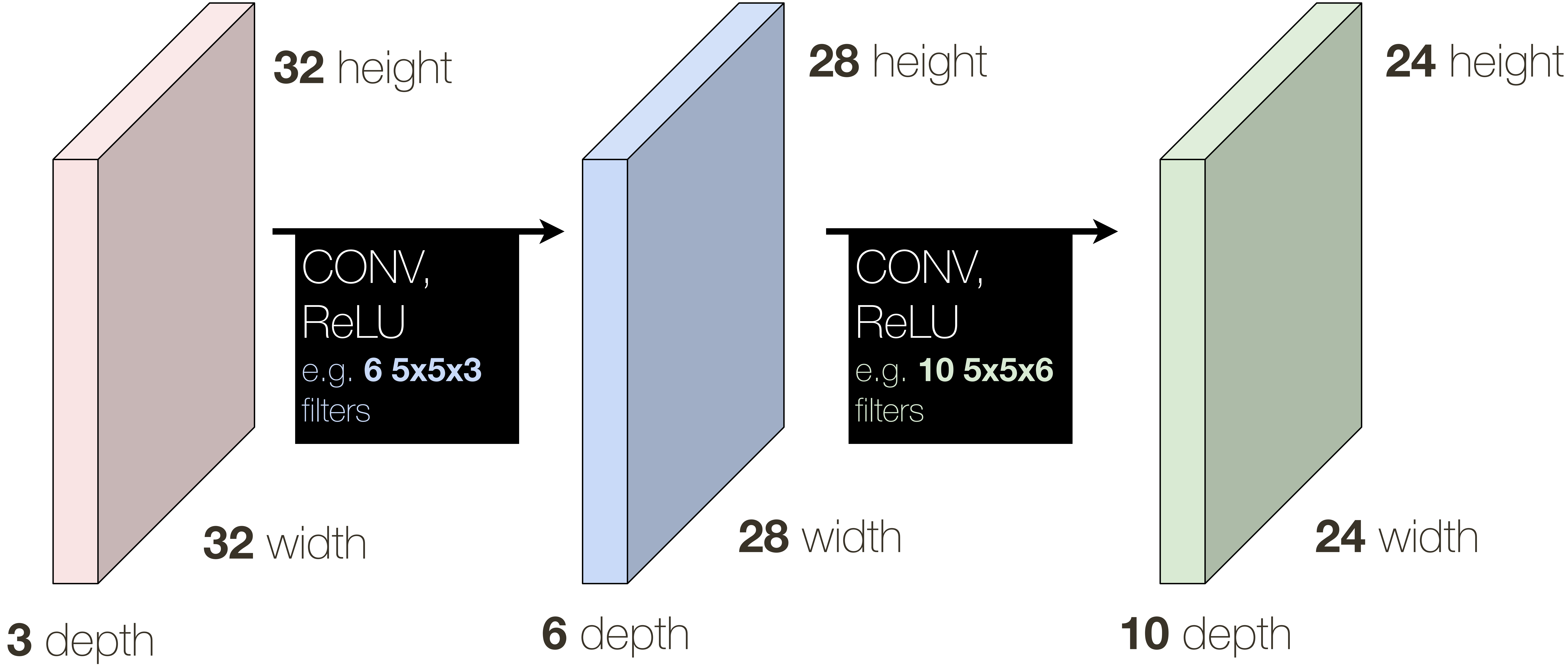


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)

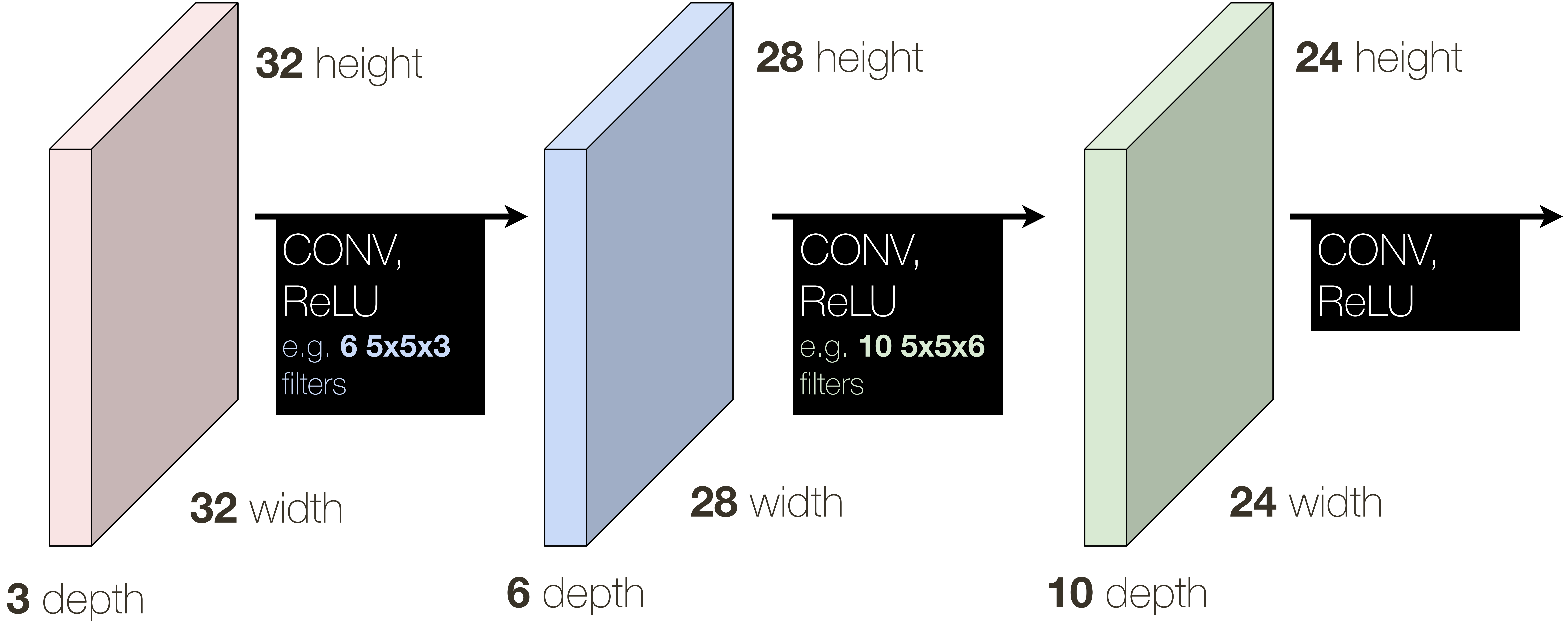


Convolutional Neural Network (ConvNet)



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

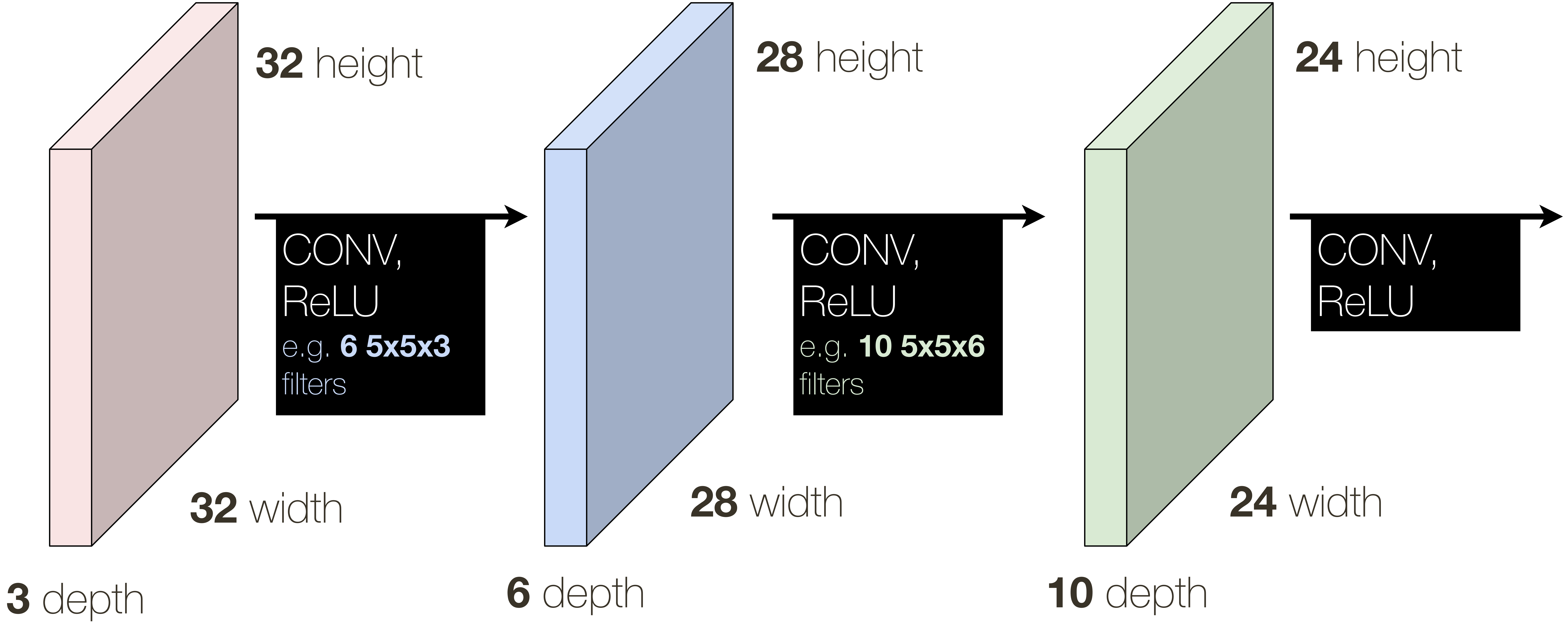
Convolutional Neural Network (ConvNet)



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

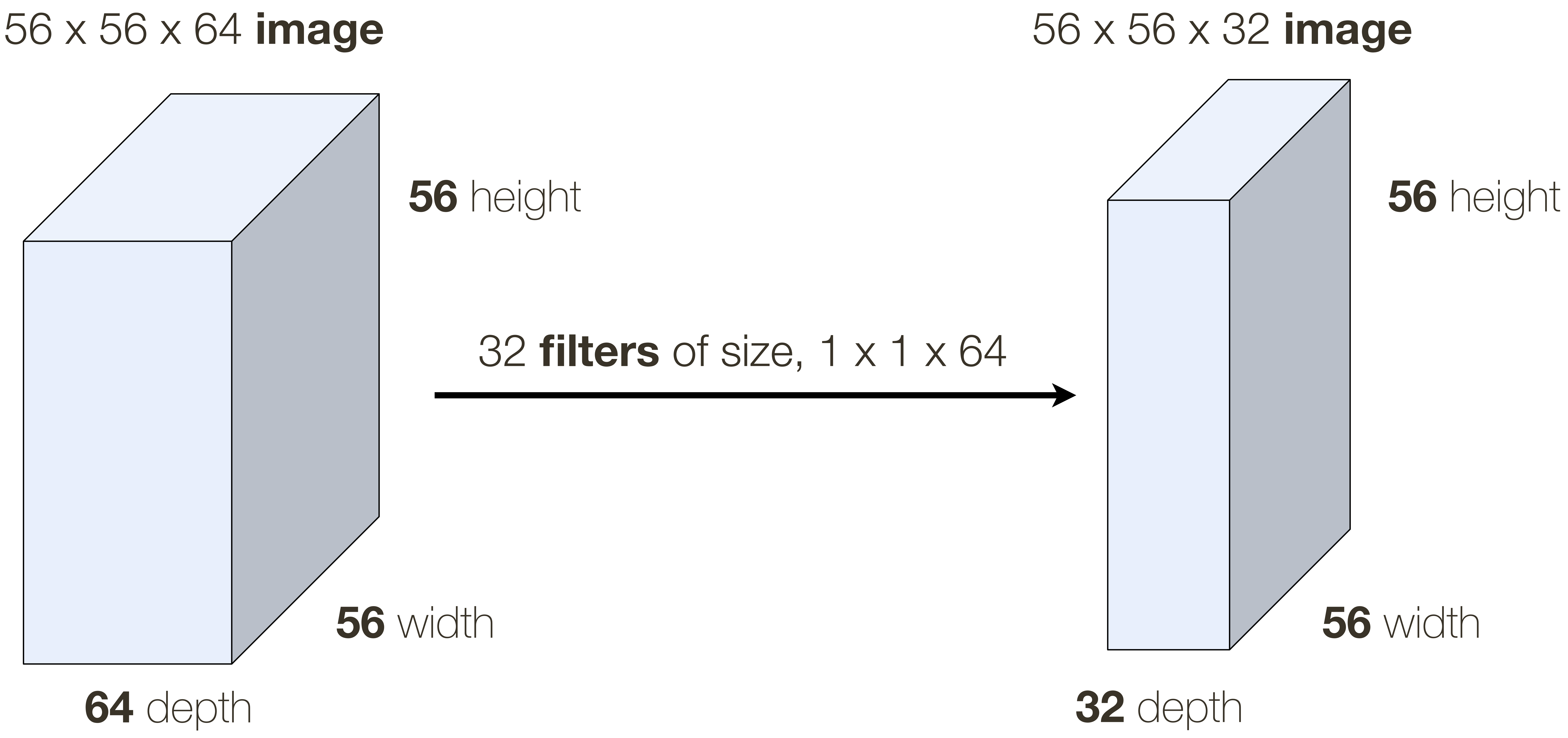
Convolutional Neural Network (ConvNet)

With padding we can achieve no shrinking (32 -> 28 -> 24); shrinking quickly (which happens with larger filters) doesn't work well in practice



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: **1x1 convolutions**



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

— Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)

— Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, ...\}$)

— Stride of application: S (for a typical network $S \in \{1, 2\}$)

— Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

— Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)

— Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, ...\}$)

— Stride of application: S (for a typical network $S \in \{1, 2\}$)

— Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

- Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, ...\}$)
- Stride of application: S (for a typical network $S \in \{1, 2\}$)
- Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$

$$W_o = (W_i - F + 2P)/S + 1 \qquad H_o = (H_i - F + 2P)/S + 1 \qquad D_o = K$$

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

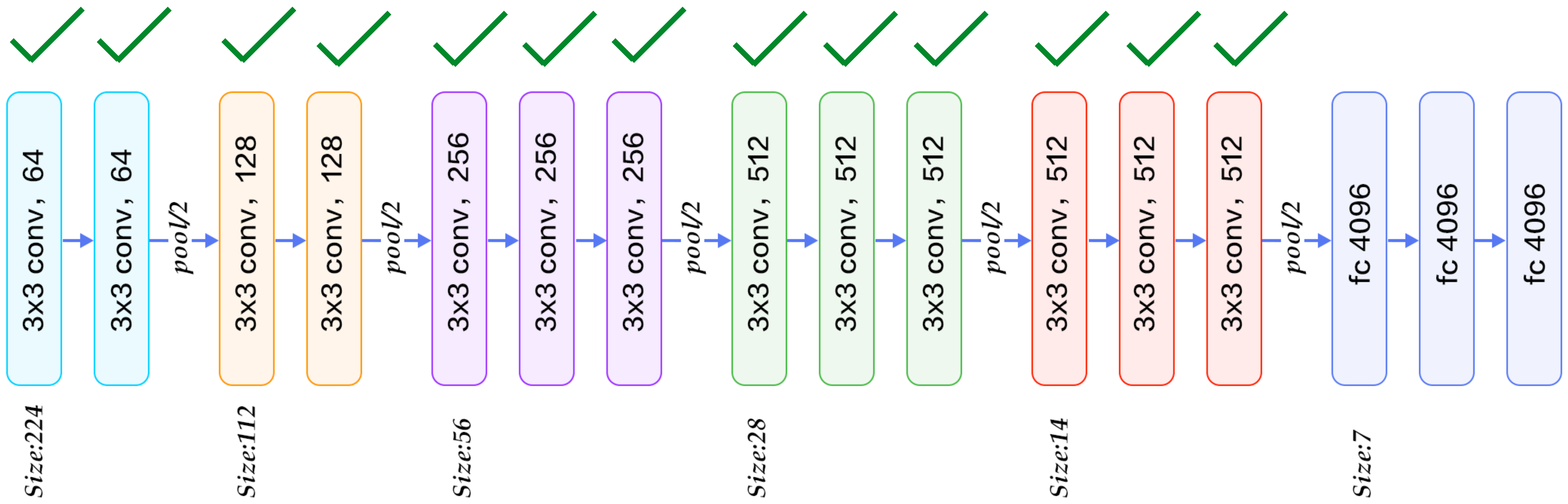
- Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, ...\}$)
- Stride of application: S (for a typical network $S \in \{1, 2\}$)
- Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$

$$W_o = (W_i - F + 2P)/S + 1 \qquad H_o = (H_i - F + 2P)/S + 1 \qquad D_o = K$$

Number of total learnable parameters: $(F \times F \times D_i) \times K + K$

Convolutional Neural Networks



VGG-16 Network

CNNs: Reminder Fully Connected Layers

Input

$$\mathbf{W}^T\mathbf{x} + \mathbf{b}, \text{ where } \mathbf{W} \in \mathbb{R}^{10\times 3072}$$

Activation

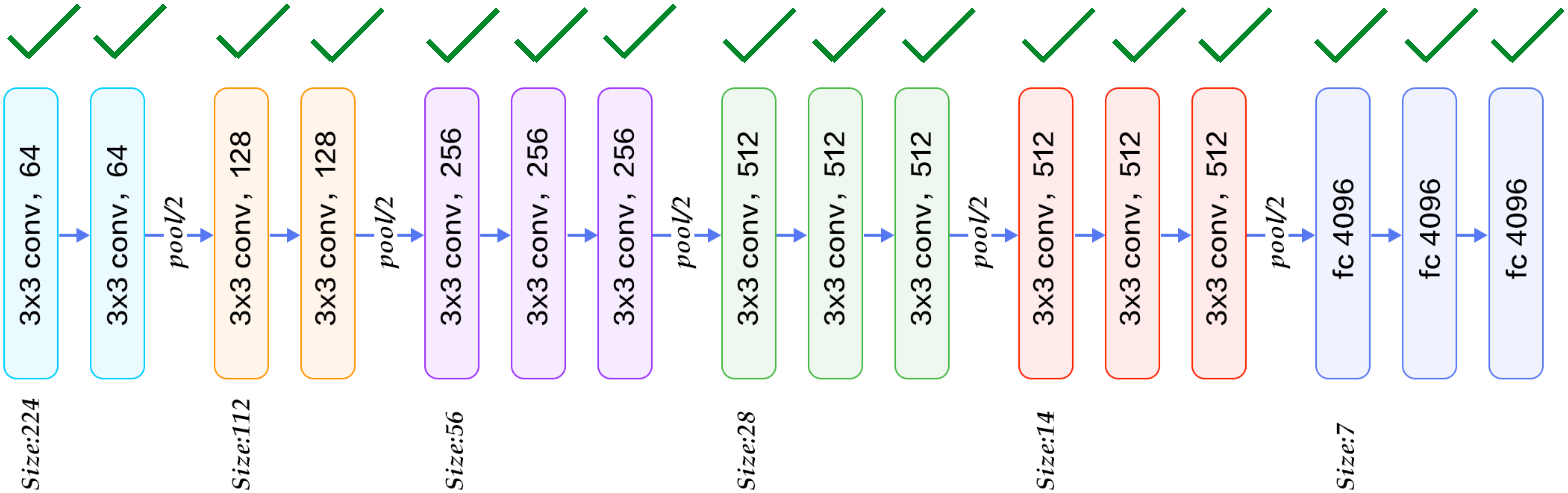
3072

(32 x 32 x 3 image -> stretches to 3072 x 1)

each neuron looks at the full input volume

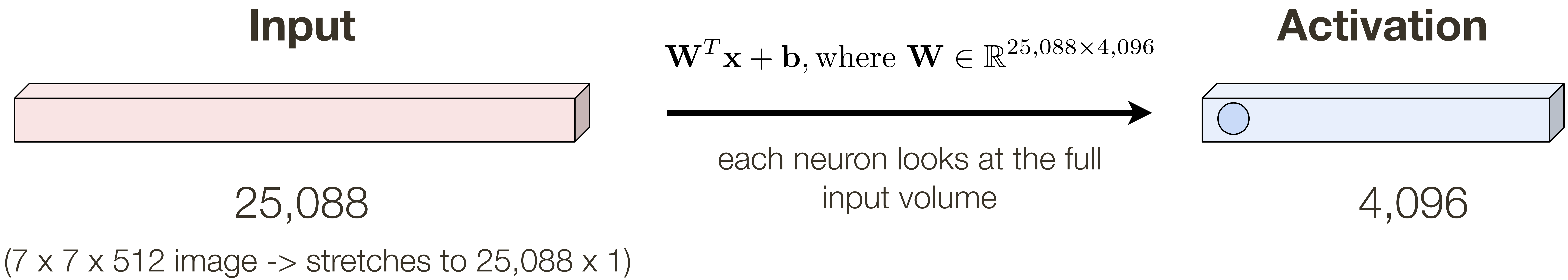
10

Convolutional Neural Networks



VGG-16 Network

CNNs: Reminder Fully Connected Layers



* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

CNNs: Reminder Fully Connected Layers

Input

Activation

$$\mathbf{W}^T\mathbf{x} + \mathbf{b}, \text{where } \mathbf{W} \in \mathbb{R}^{25,088 \times 4,096}$$

each neuron looks at the full input volume

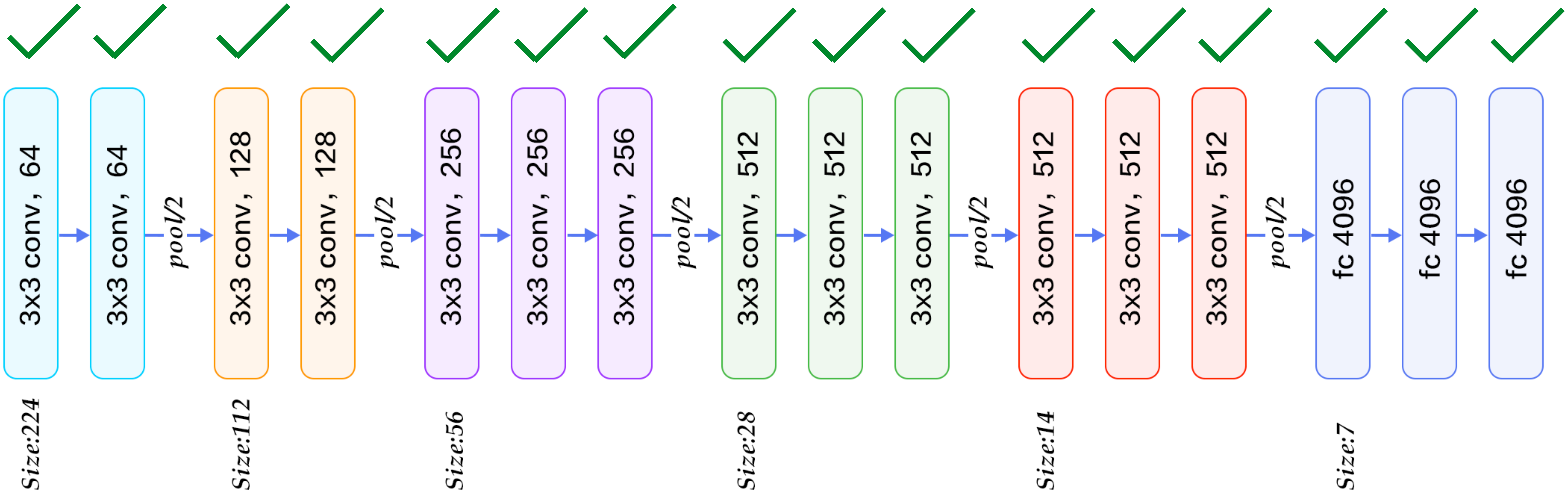
25,088

(7 x 7 x 512 image -> stretches to 25,088 x 1)

4,096

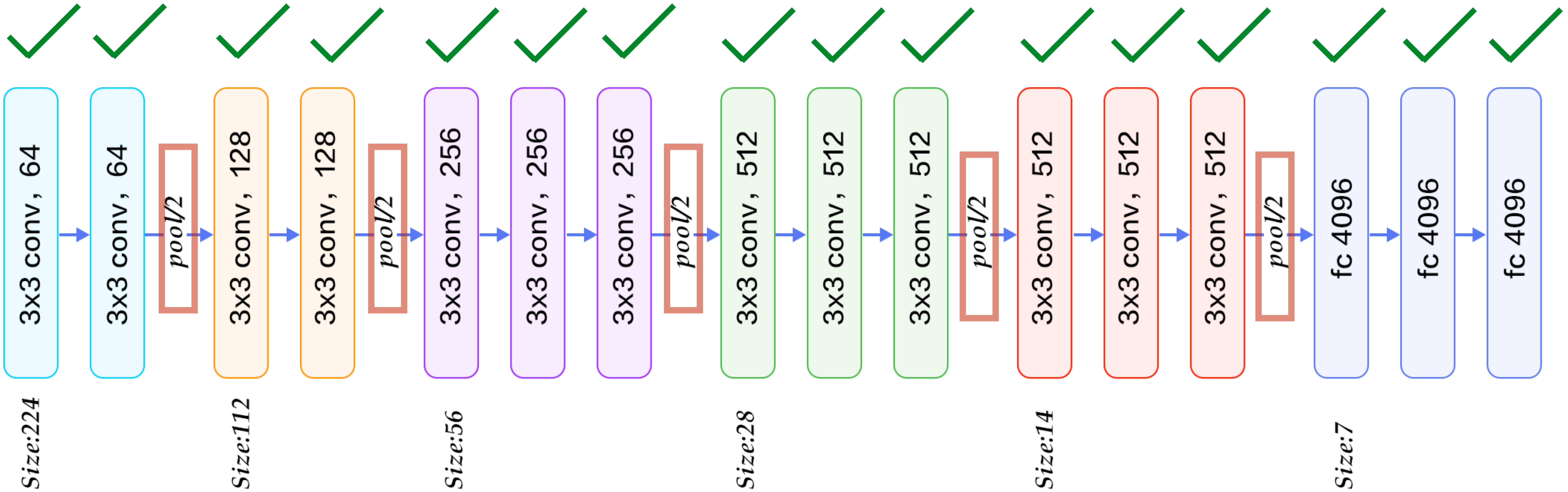
102,760,448 parameters!

Convolutional Neural Networks



VGG-16 Network

Convolutional Neural Networks



VGG-16 Network

Invariance vs. **Equivariance**

Invariance vs. **Equivariance**

Invariance: A mathematical object (or class of mathematical objects) remains unchanged after transformations of certain types applied to the objects

$$f(x) = f(g(x))$$

Invariance vs. **Equivariance**

Invariance: A mathematical object (or class of mathematical objects) remains unchanged after transformations of certain types applied to the objects

$$f(x) = f(g(x))$$

Equivariance: Applying a transformation and then computing the function produces the same result as computing the function and then applying a transformation

$$g(f(x)) = f(g(x))$$

Revisit **Layers** we Learned About

Fully Connected:

Revisit **Layers** we Learned About

Fully Connected:

— **<u>Not</u>** invariant to any transformations

— **<u>Not</u>** equivariant to any transformations

Revisit **Layers** we Learned About

Fully Connected:

— **Not** invariant to any transformations

— **Not** equivariant to any transformations

Convolutional:

Revisit **Layers** we Learned About

Fully Connected:

— **Not** invariant to any transformations

— **Not** equivariant to any transformations

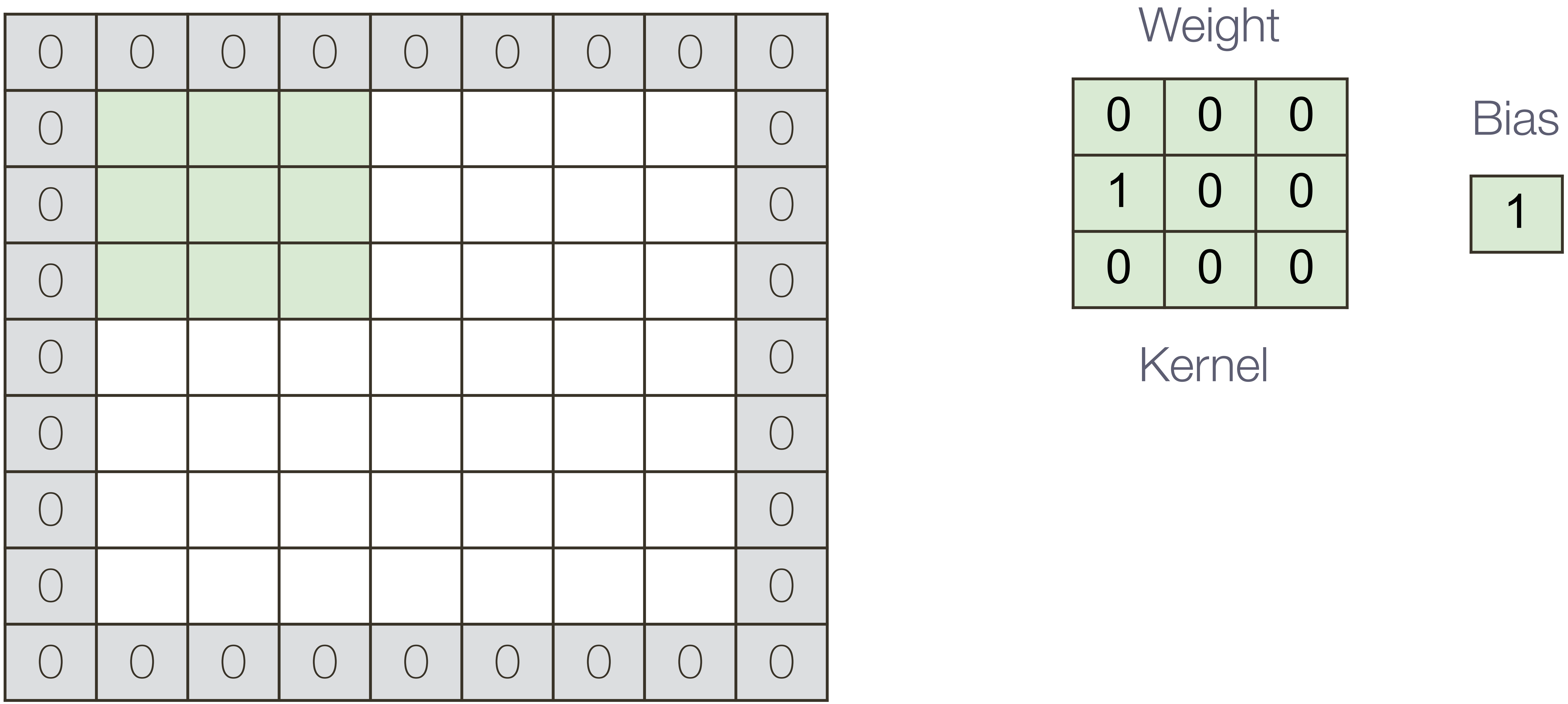
Convolutional:

— **Not** invariant to any transformations

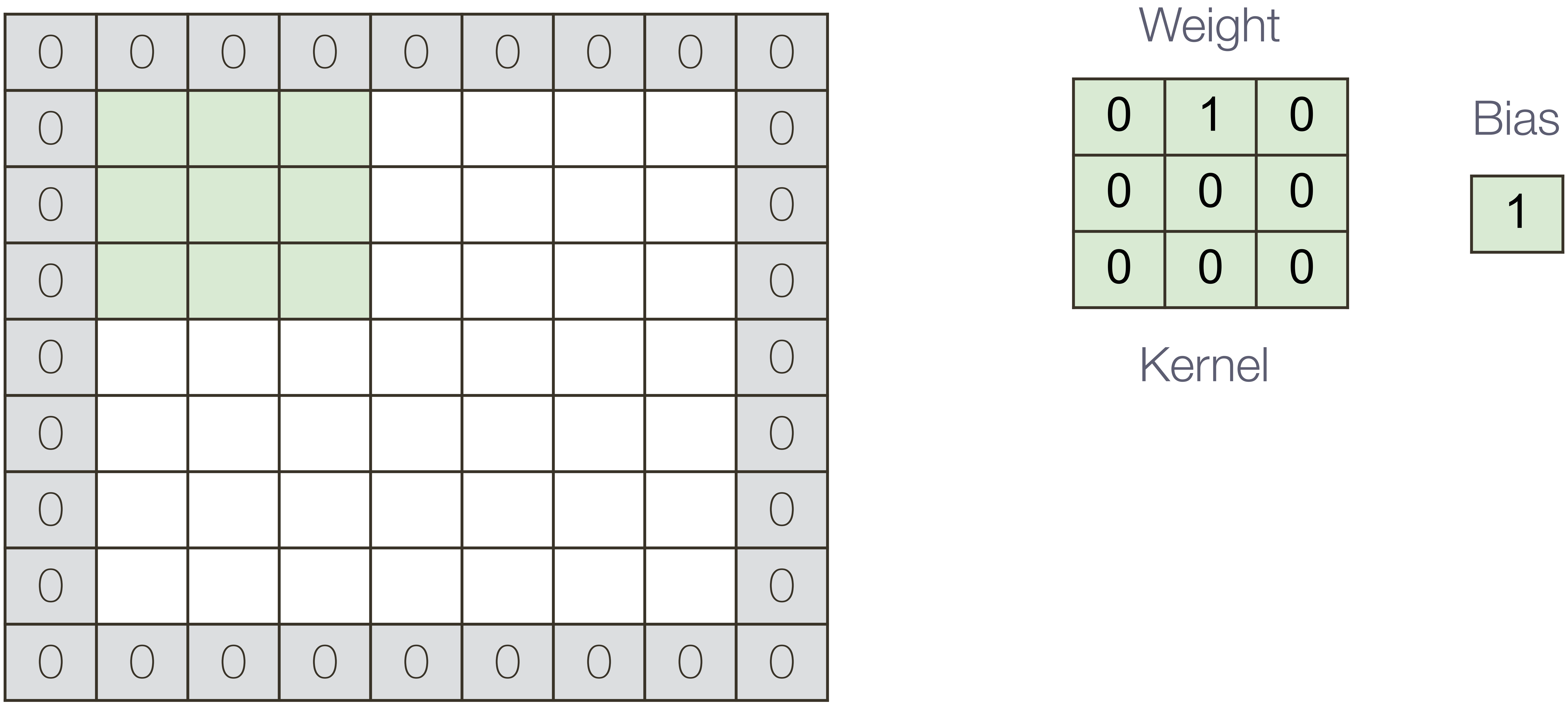
— Convolution is **translation equivariant**

Note: convolution can “learn” not to be equivariant when padding is used.

Revisit **Layers** we Learned About



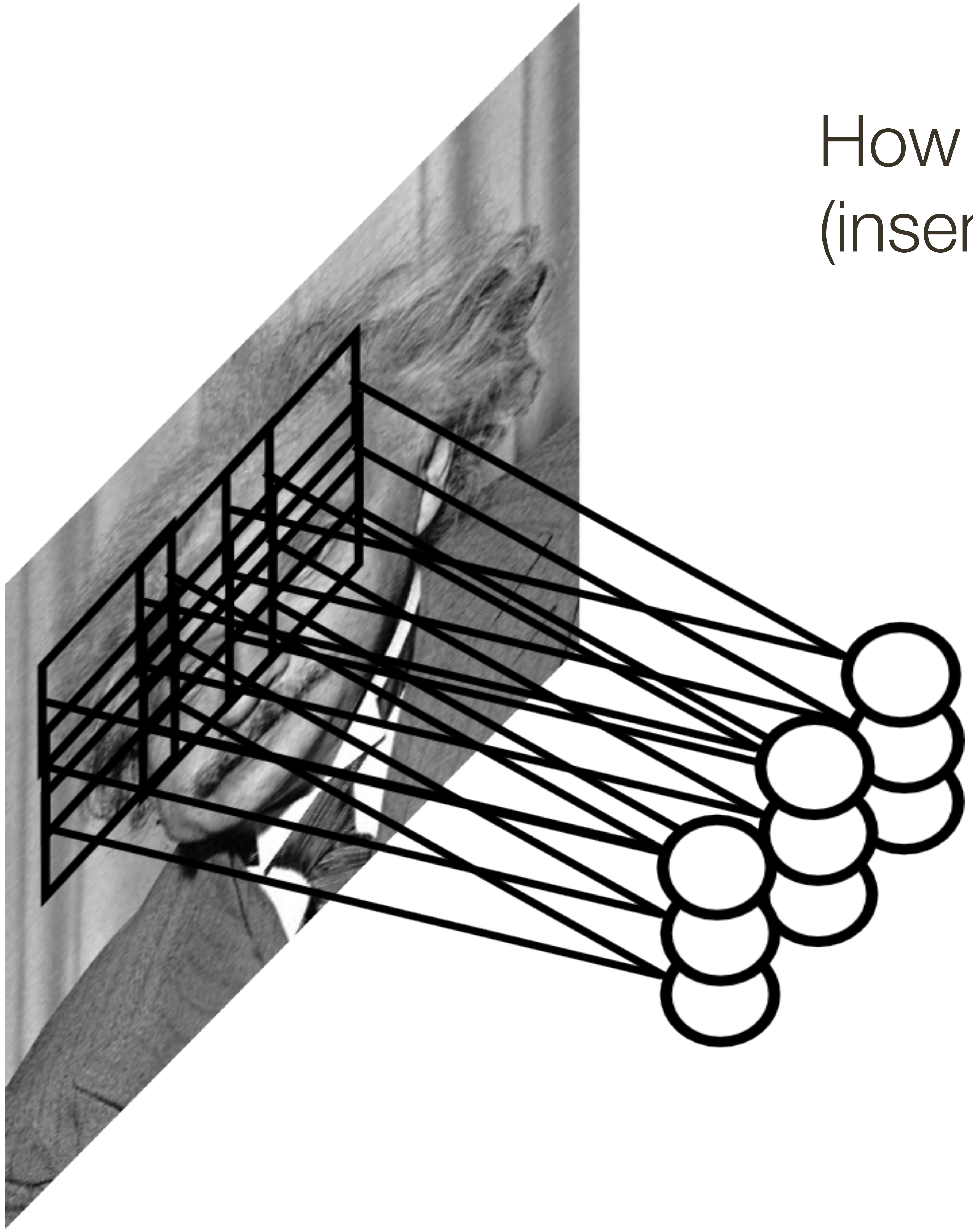
Revisit **Layers** we Learned About



Pooling Layer

Let us assume the filter is an "eye" detector

How can we make detection spatially invariant (insensitive to position of the eye in the image)

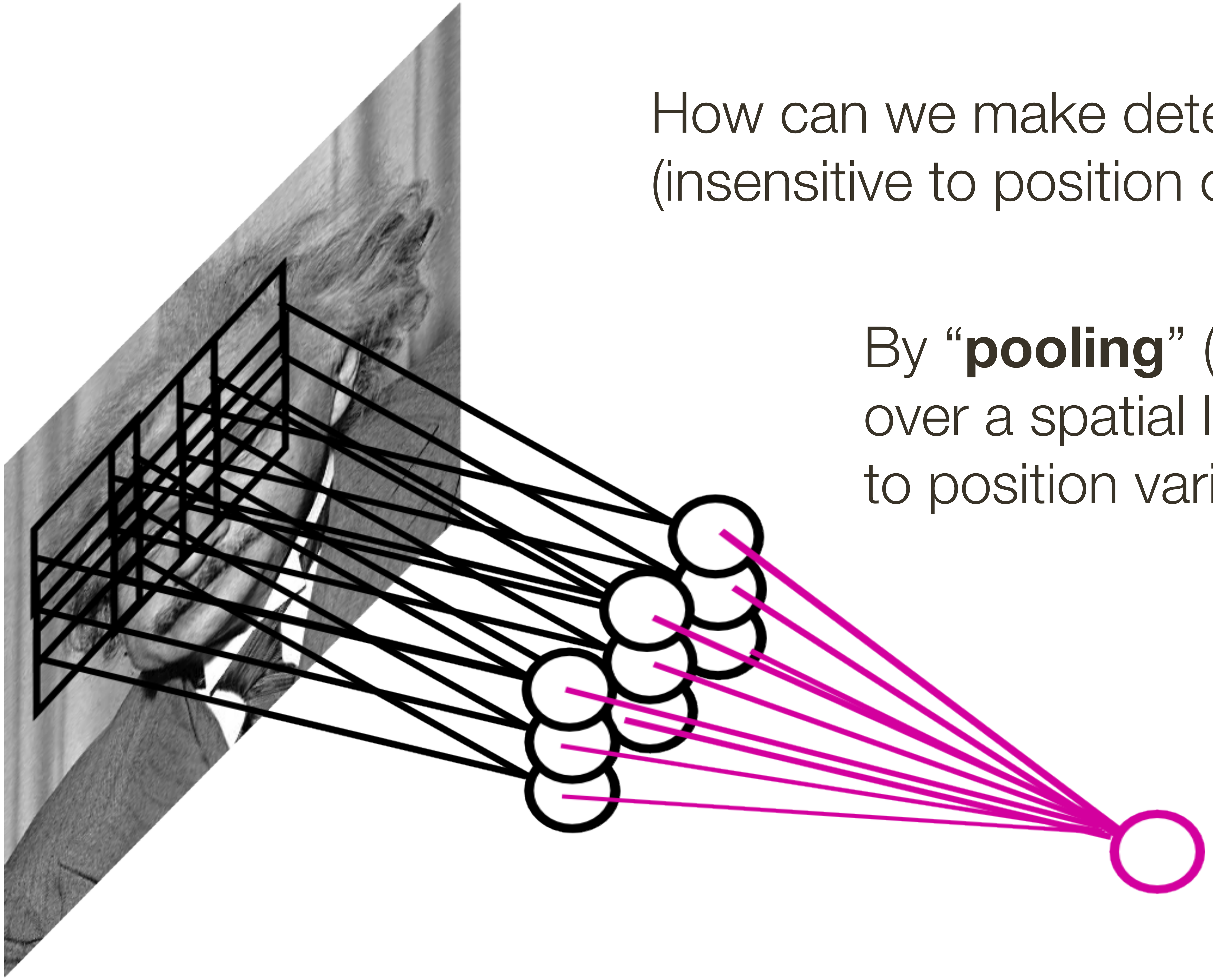


* slide from Marc'Aurelio Renzato

Pooling Layer

Let us assume the filter is an “eye” detector

How can we make detection spatially invariant (insensitive to position of the eye in the image)

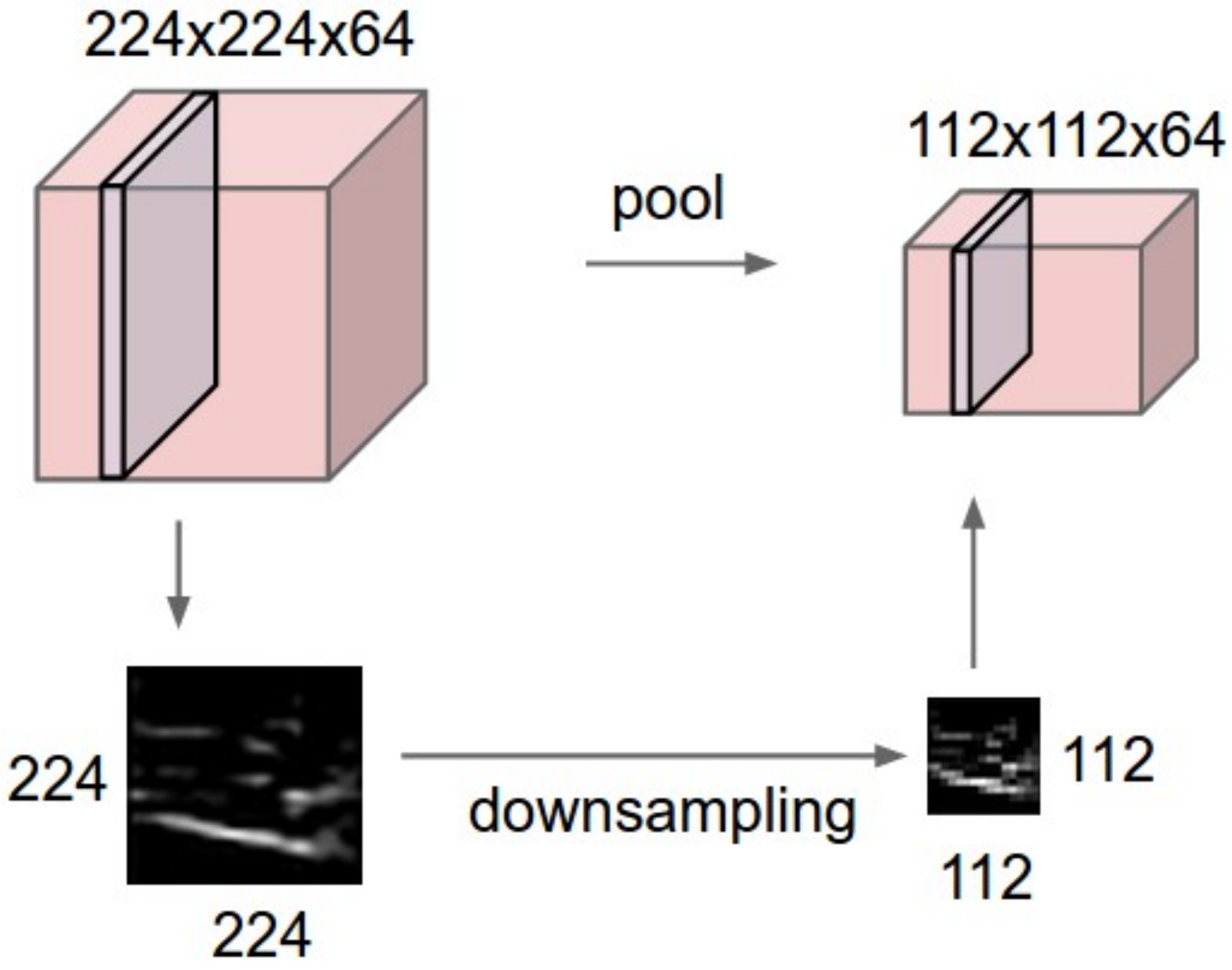


By “**pooling**” (e.g., taking a max) response over a spatial locations we gain robustness to position variations

* slide from Marc’Aurelio Renzato

Pooling Layer

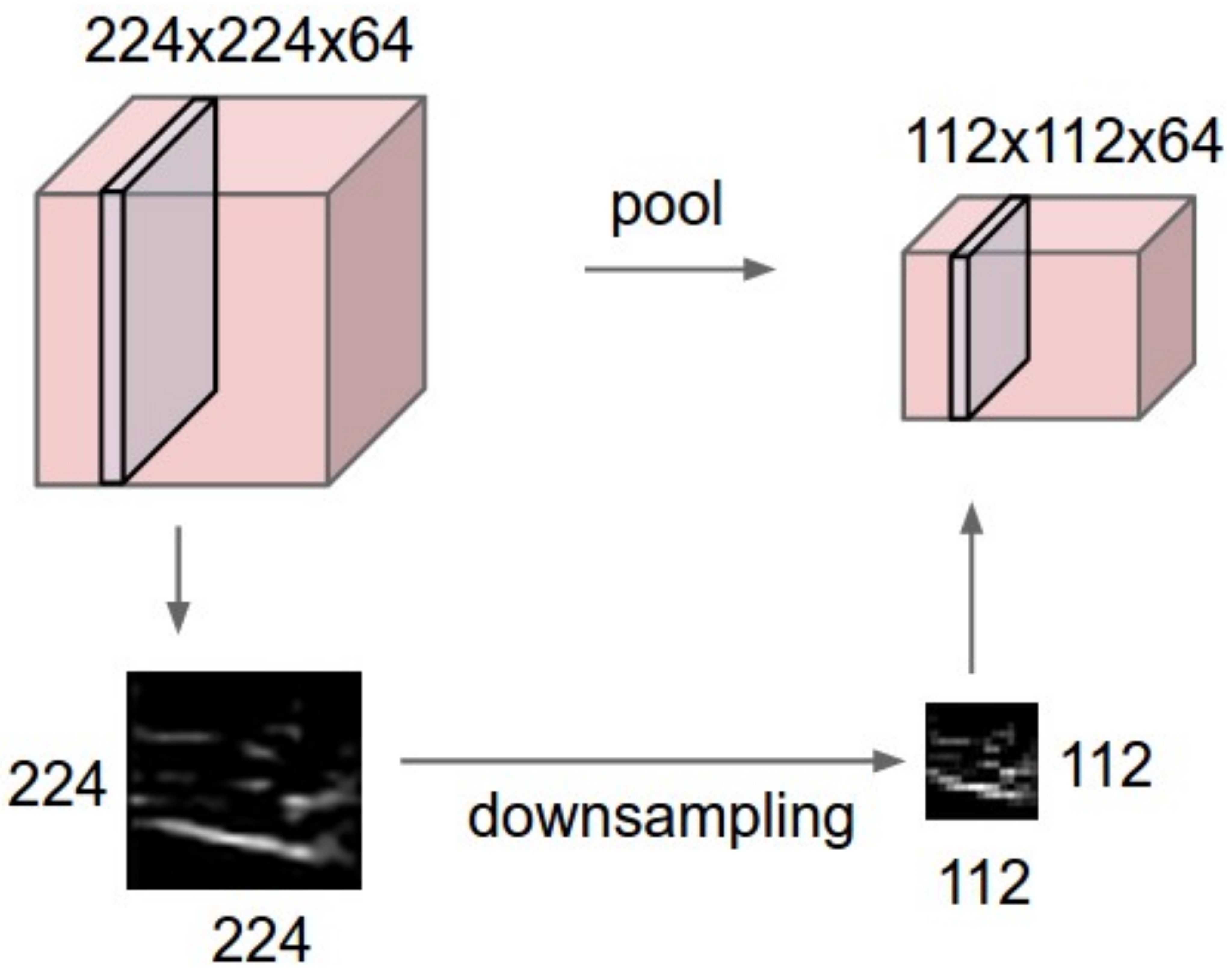
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Pooling Layer

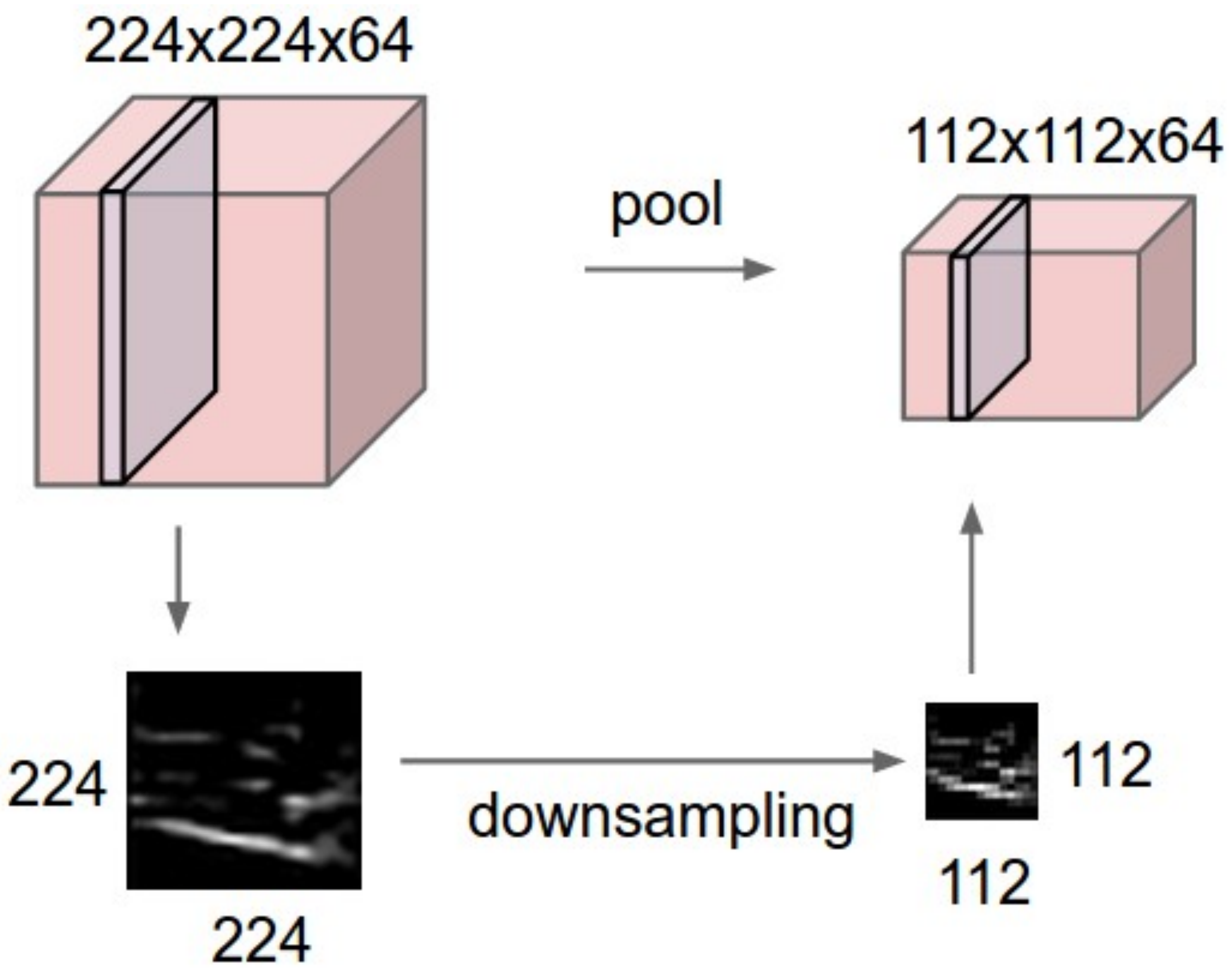
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



How many **parameters**?

Pooling Layer

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



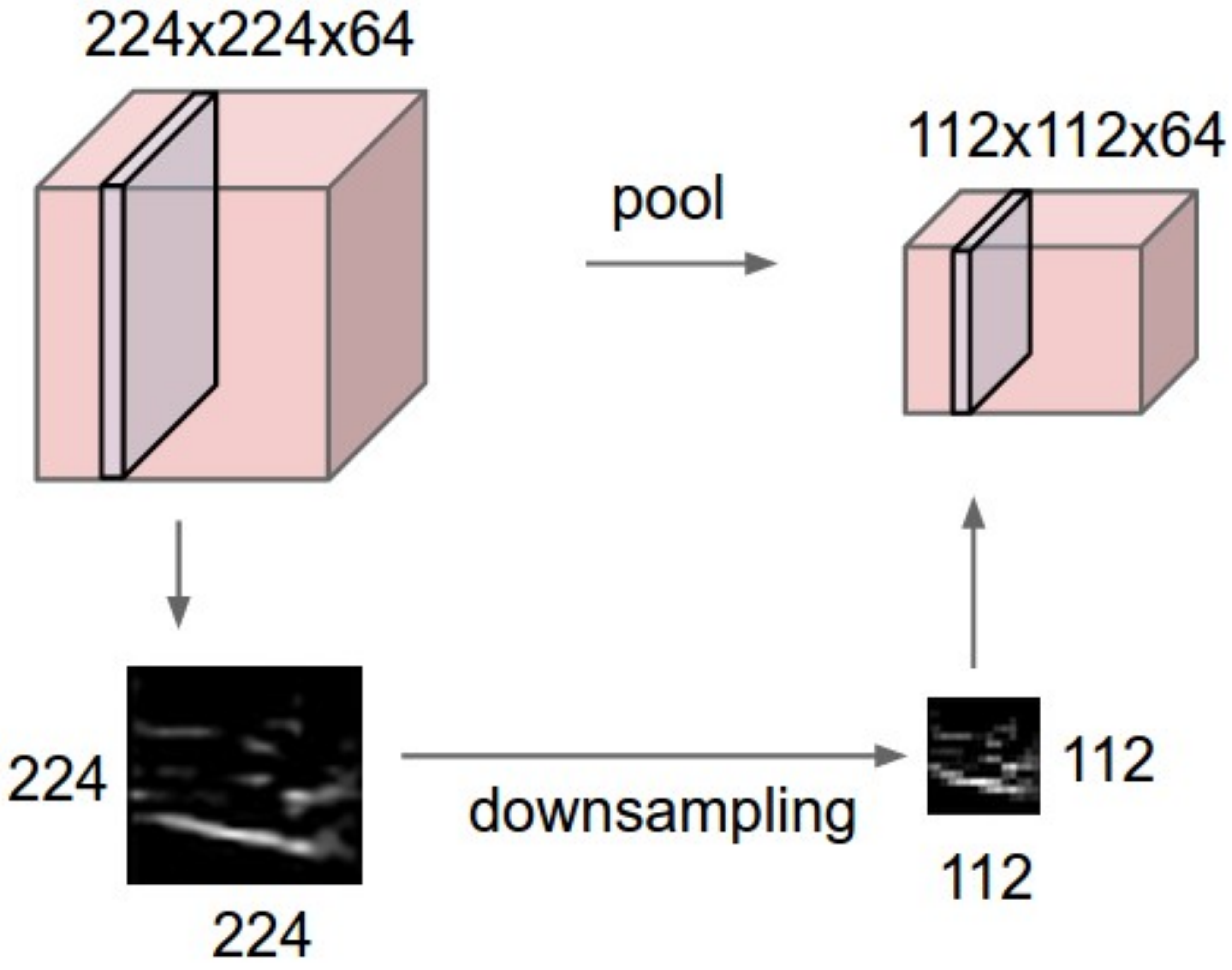
How many **parameters**?

None!

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Pooling Layer

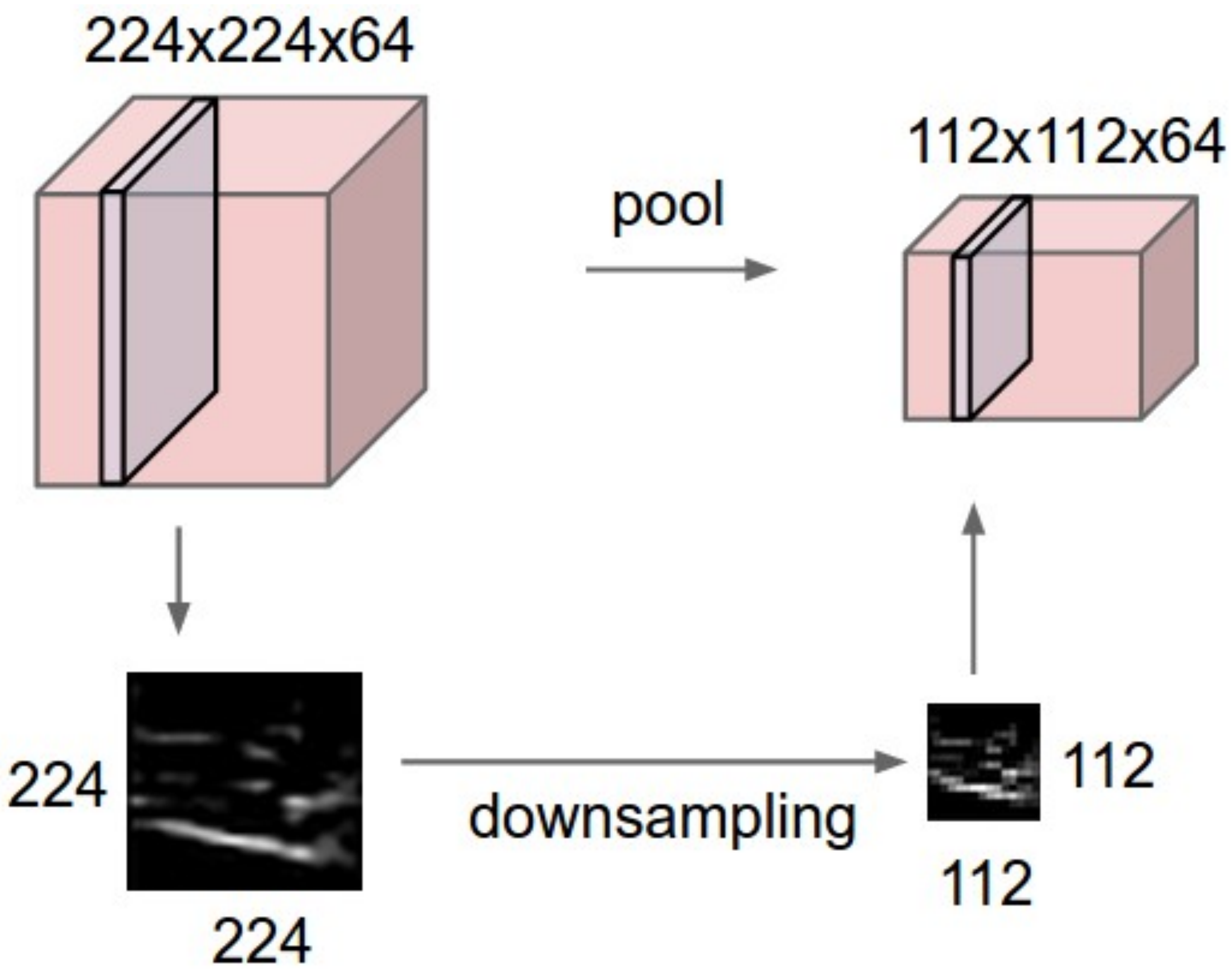
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Pooling Layer

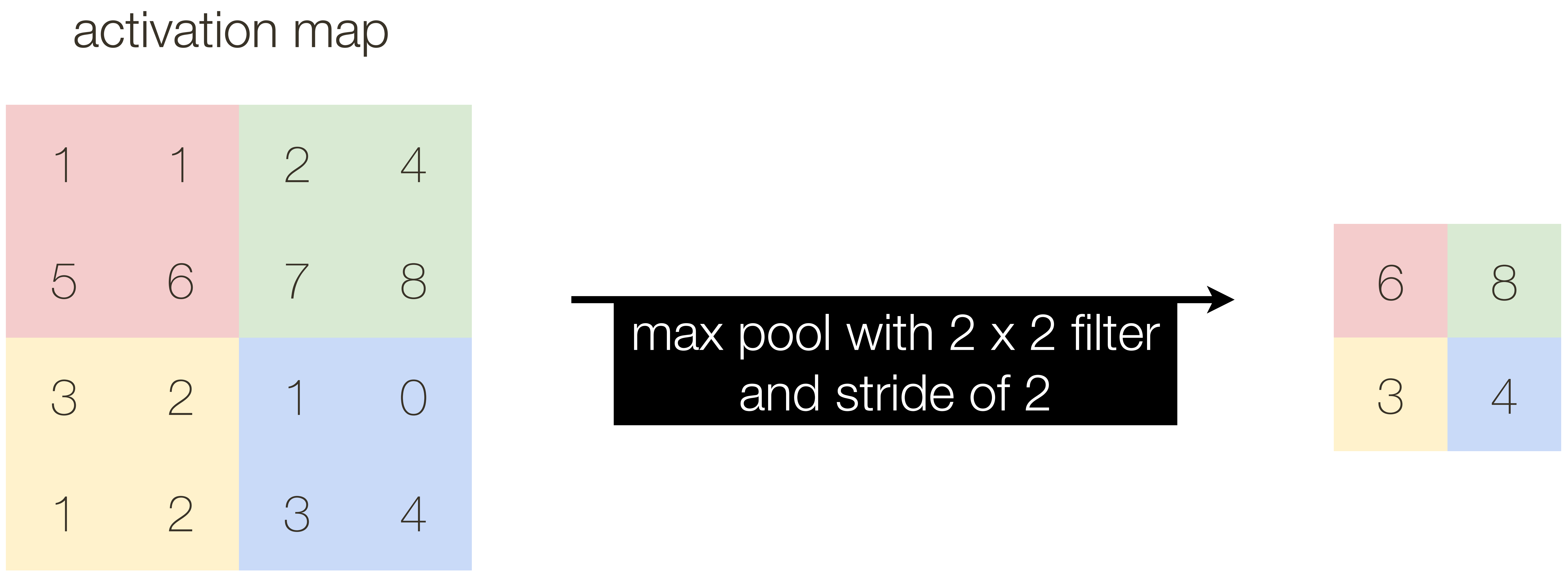
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



How do we implement that in a **computation graph**?

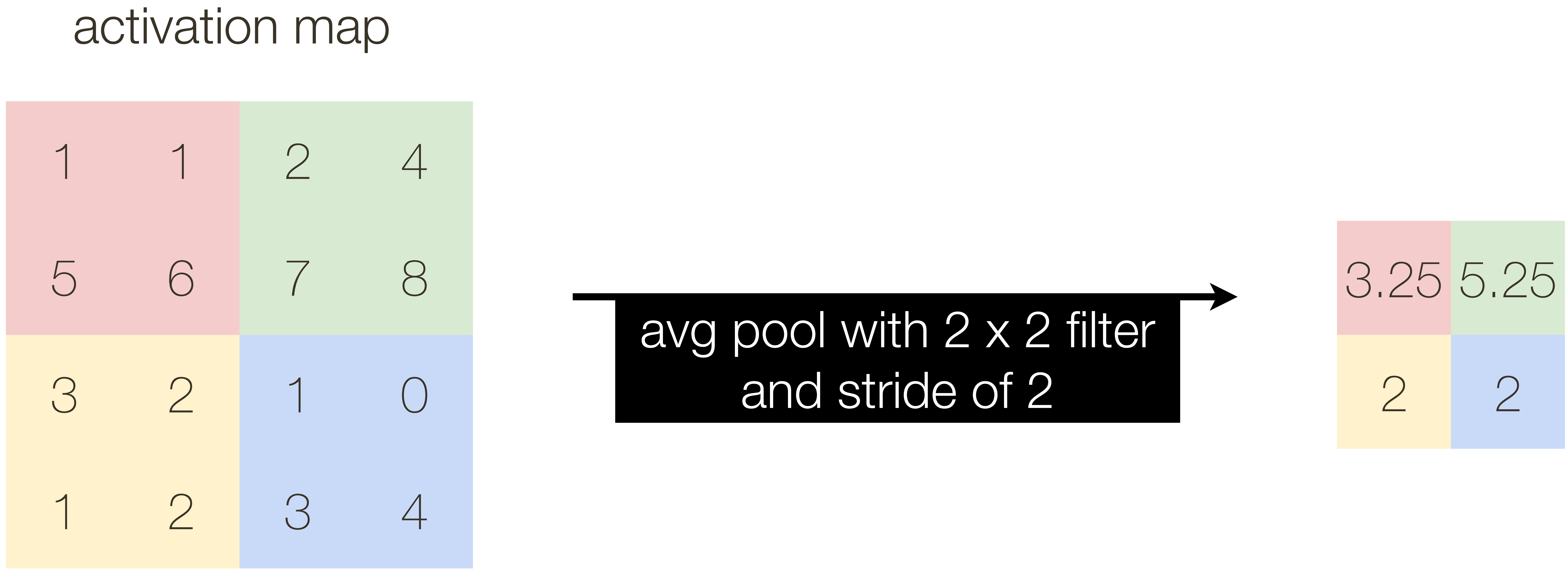
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Max **Pooling**



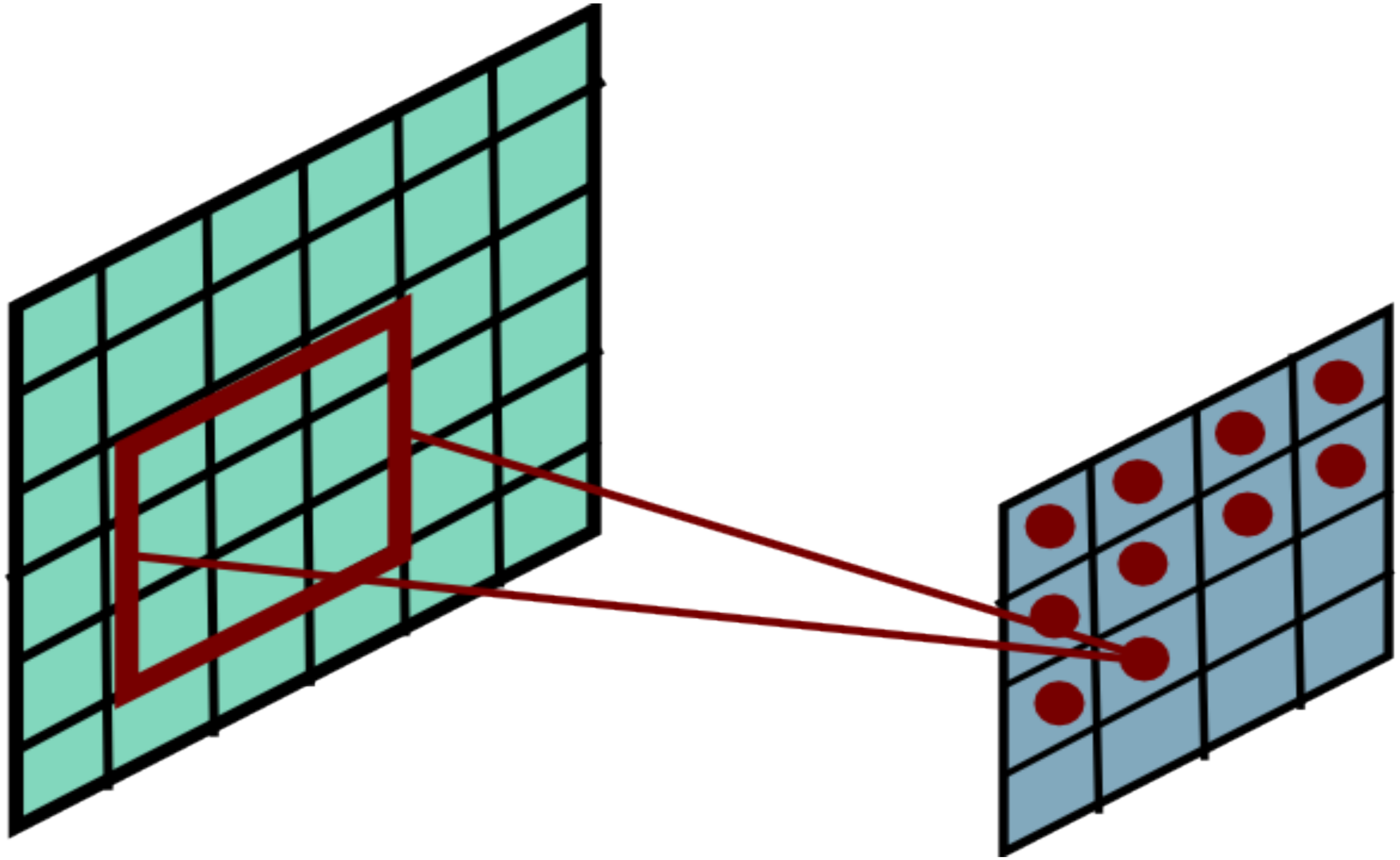
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Average **Pooling**



Pooling Layer **Receptive Field**

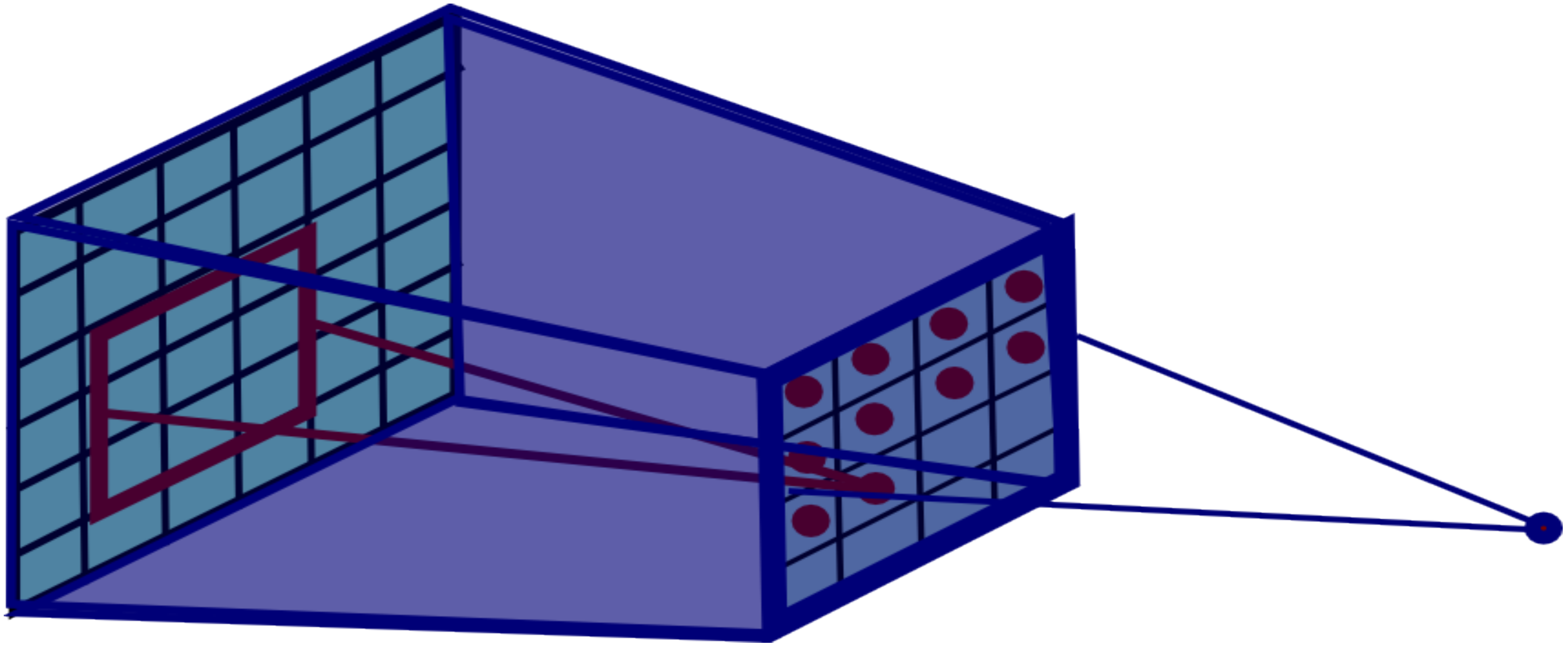
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: **(P+K-1)x(P+K-1)**



* slide from Marc'Aurelio Renzato

Pooling Layer **Receptive Field**

If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: **(P+K-1)x(P+K-1)**



* slide from Marc'Aurelio Renzato

Pooling Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

- Spatial extent of filters: K
- Stride of application: F

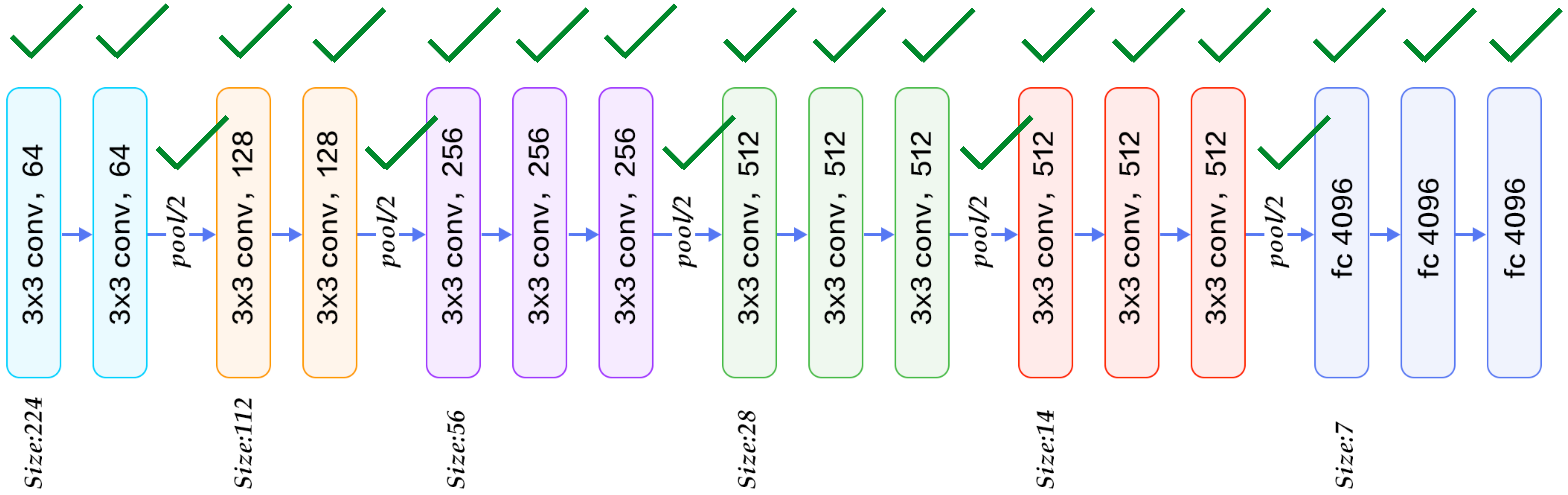
Produces a volume of size: $W_o \times H_o \times D_o$

$W_o = W_i/F$ $\quad$ $H_o = H_i/F$ $\quad$ $D_o = D_i$

Number of total learnable parameters: 0

(you can do padding, but it's a bit trickier)

Convolutional Neural Networks



VGG-16 Network

Improving **Single Model**

Regularization

- L2, L1
- Dropout / Inverted Dropout
- Data augmentation

Improving **Single Model**

Regularization

- L2, L1
- Dropout / Inverted Dropout
- Data augmentation

L2 Regularization: Learn a more (dense) distributed representation

$$R(\mathbf{W}) = ||\mathbf{W}||_2 = \sum_i \sum_j \mathbf{W}_{i,j}^2$$

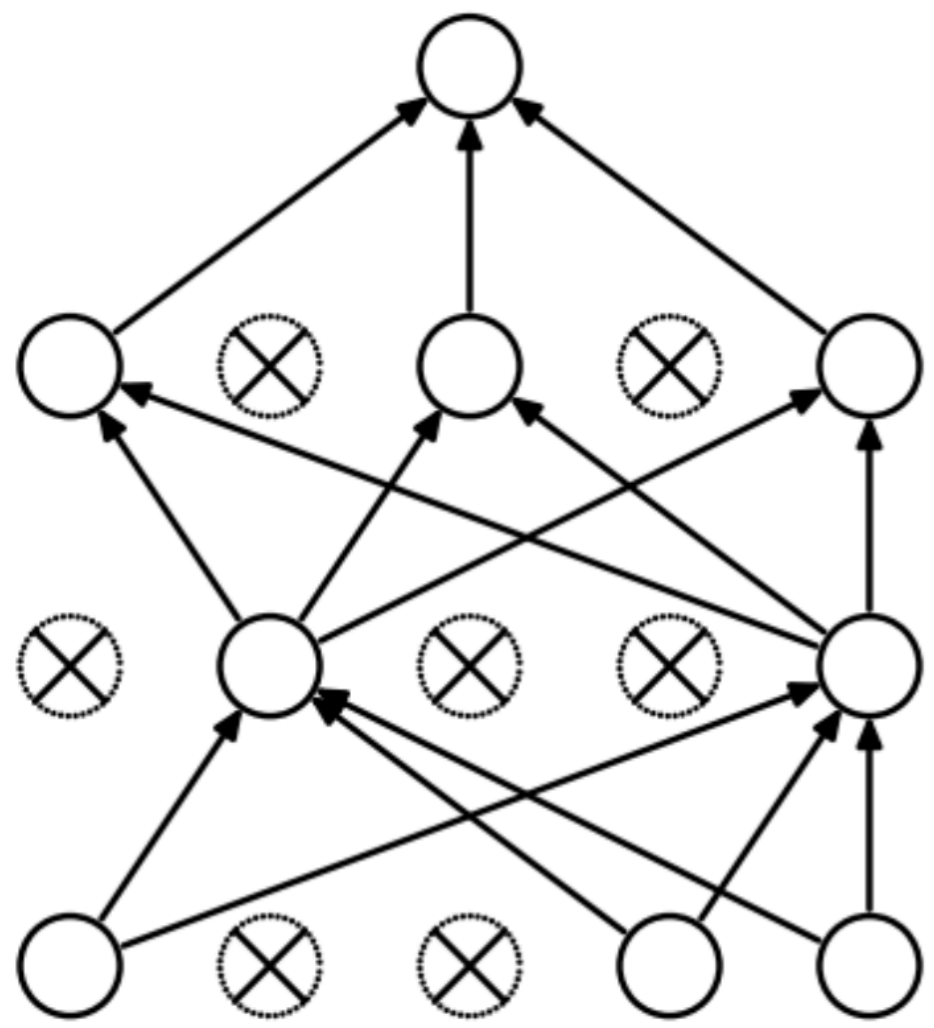
L1 Regularization: Learn a sparse representation (few non-zero wight elements)

$$R(\mathbf{W}) = ||\mathbf{W}||_1 = \sum_i \sum_j |\mathbf{W}_{i,j}|$$

Improving **Single Model**

Regularization

- L2, L1
- Dropout / Inverted Dropout
- Data augmentation



Dropout

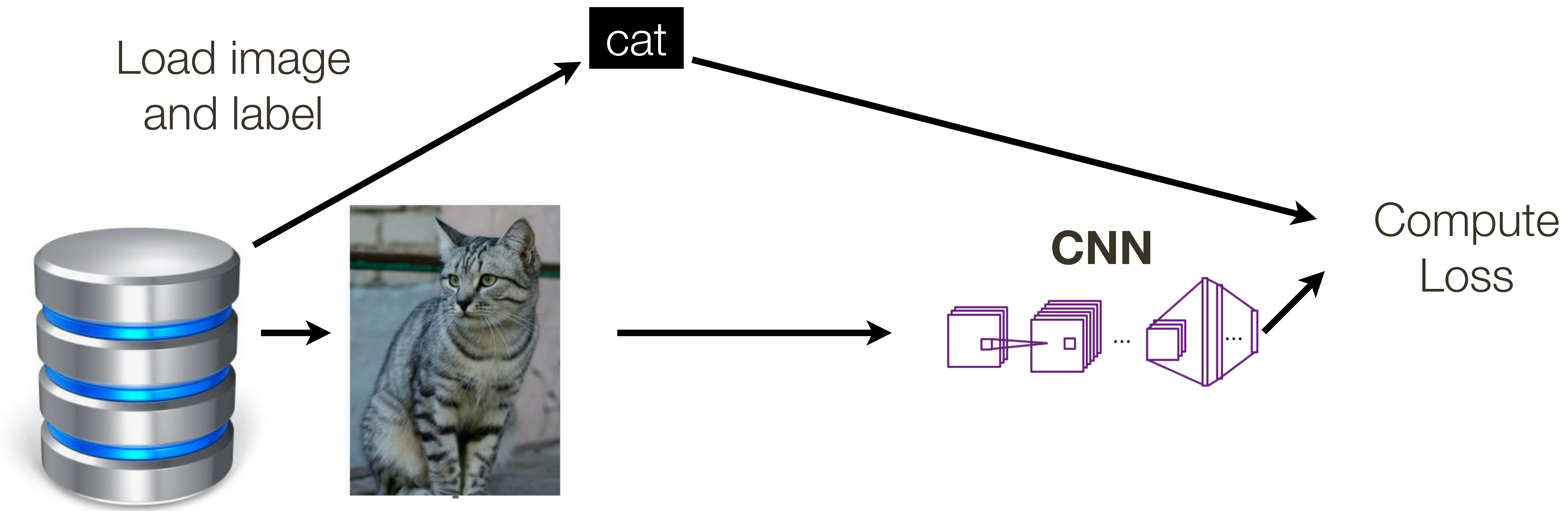
L2 Regularization: Learn a more (dense) distributed representation

$$R(\mathbf{W}) = ||\mathbf{W}||_2 = \sum_i \sum_j \mathbf{W}_{i,j}^2$$

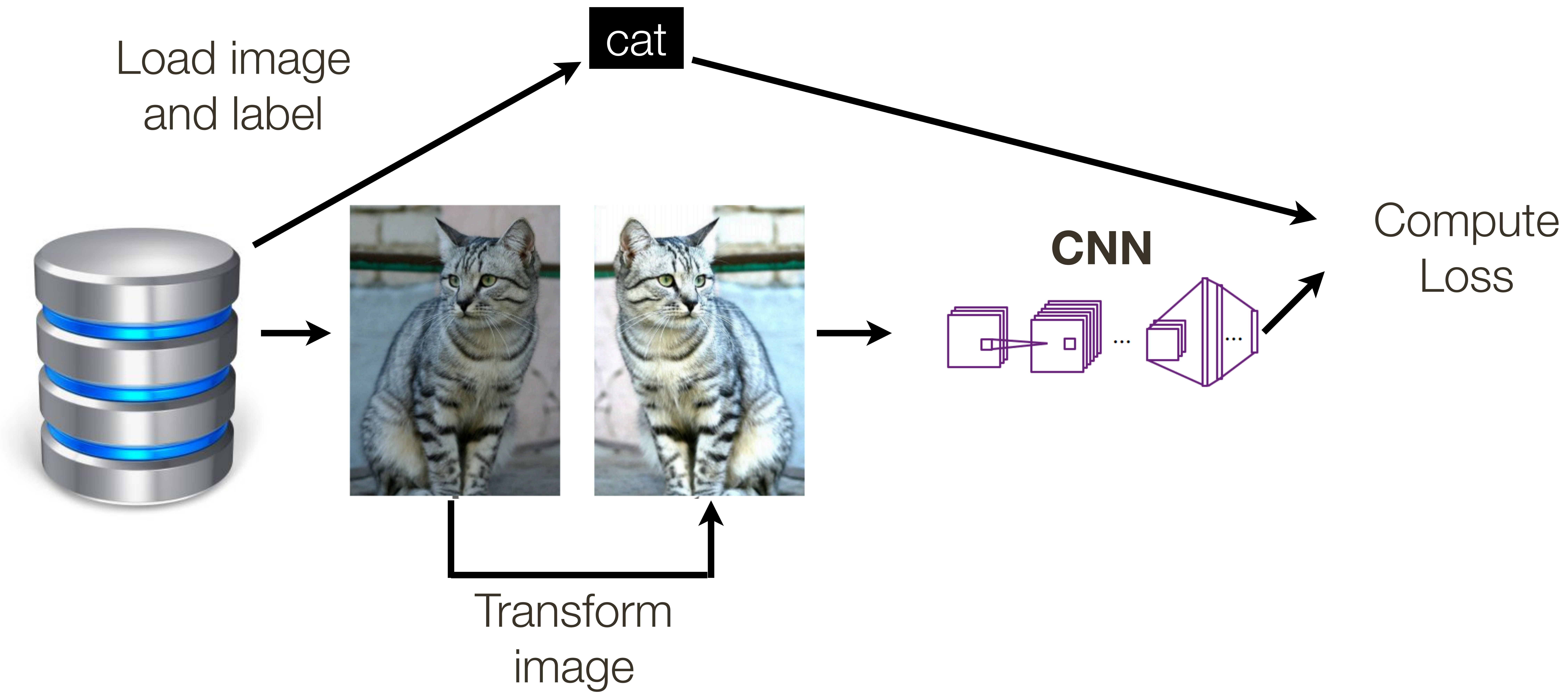
L1 Regularization: Learn a sparse representation (few non-zero wight elements)

$$R(\mathbf{W}) = ||\mathbf{W}||_1 = \sum_i \sum_j |\mathbf{W}_{i,j}|$$

Regularization: Data Augmentation



Regularization: Data Augmentation



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Regularization: Data Augmentation

Horizontal flips

Random crops & scales

Color Jitter

Regularization: Data Augmentation

Horizontal flips | Random crops & scales | Color Jitter

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Regularization: Data Augmentation

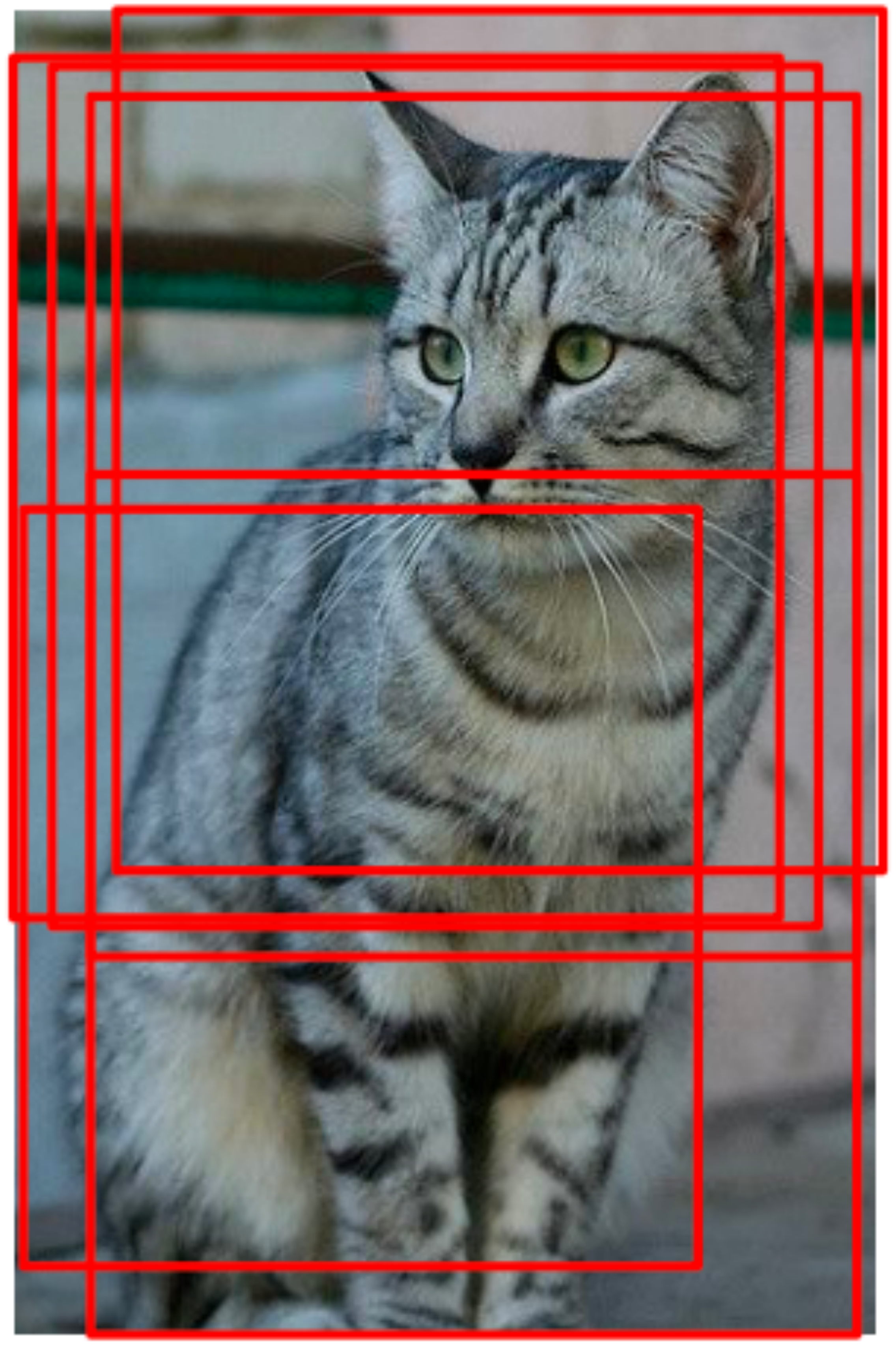
Horizontal flips | **Random crops & scales** | Color Jitter

Training: sample random crops and scales
e.g., ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short size = L
3. Sample random 224x224 patch

Testing: average a fix set of crops
e.g., ResNet:

1. Resize image to 5 scales (224, 256, 384, 480, 640)
2. For each image use 10 224x224 crops: 4 corners + center, + flips



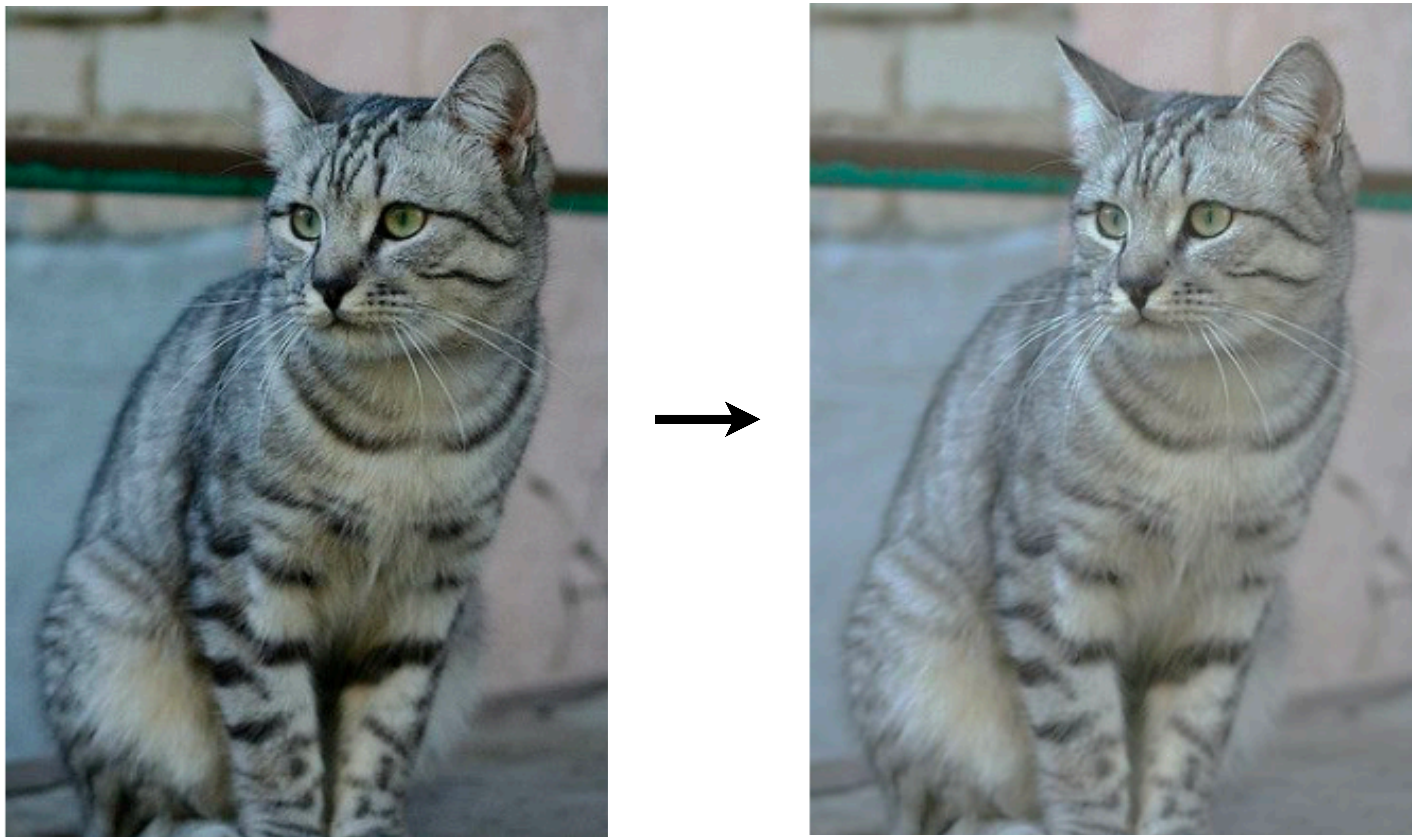
Regularization: Data Augmentation

Horizontal flips

Random crops & scales

Color Jitter

Random perturbations in contrast and brightness



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

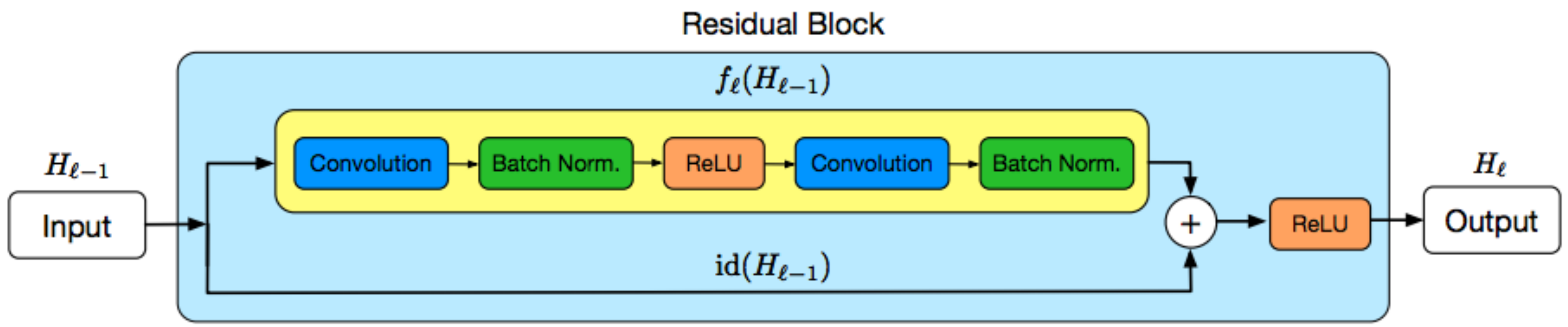
Regularization: Stochastic Depth

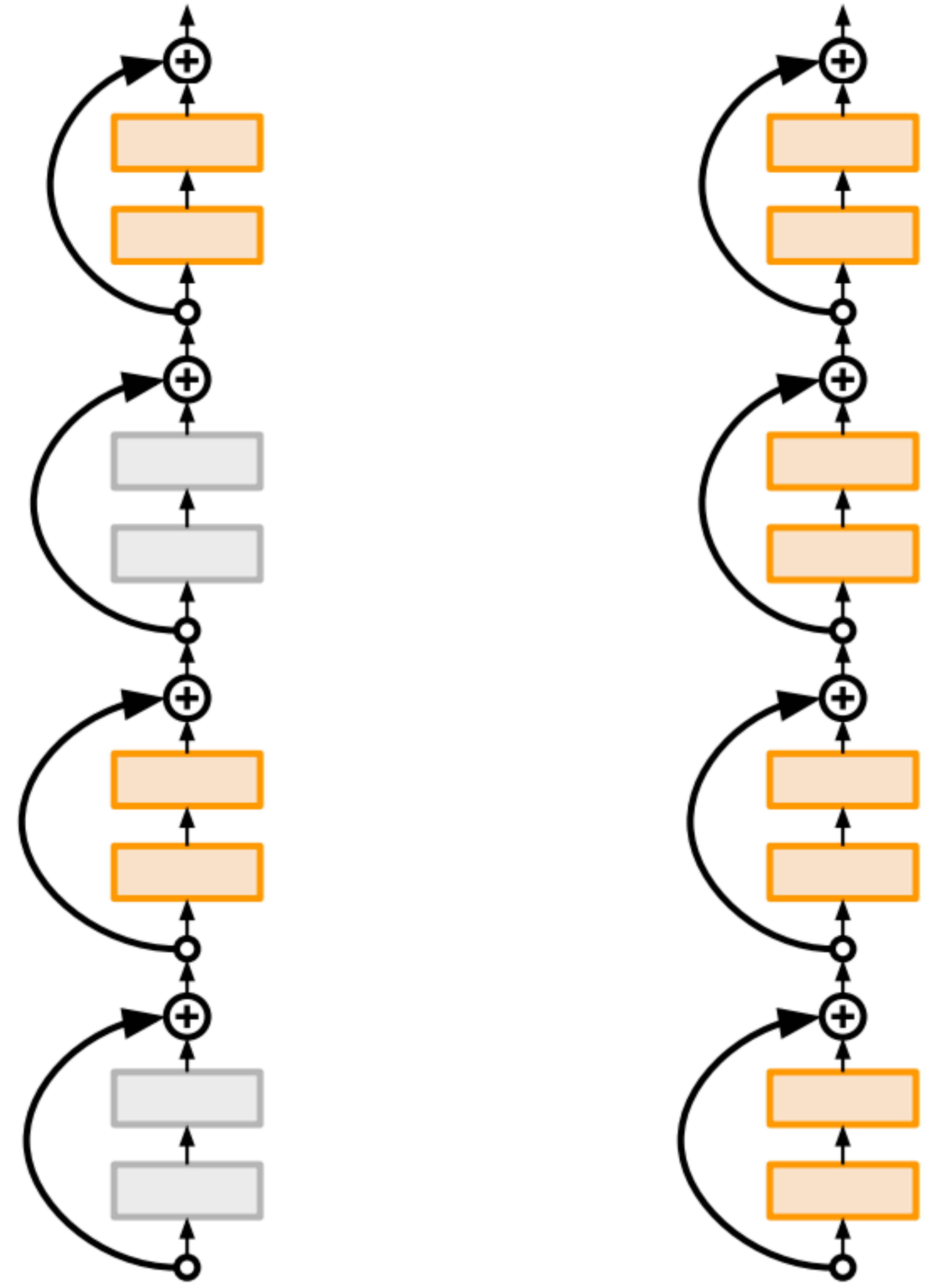
[Huang et al., ECCV 2016]

Effectively "dropout" but for layers

Stochastically with some probability **turn off some layer** (for each batch)

Effectively trains a collection of neural networks





* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

Common "Wisdom": You need a lot of data to train a CNN

Transfer Learning with CNNs

Common "Wisdom": You need a lot of data to train a CNN

Solution: Transfer learning — taking a model trained on the task that has lots of data and adopting it to the task that may not

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

Common “Wisdom”: You need a lot of data to train a CNN

Solution: Transfer learning — taking a model trained on the task that has lots of data and adopting it to the task that may not

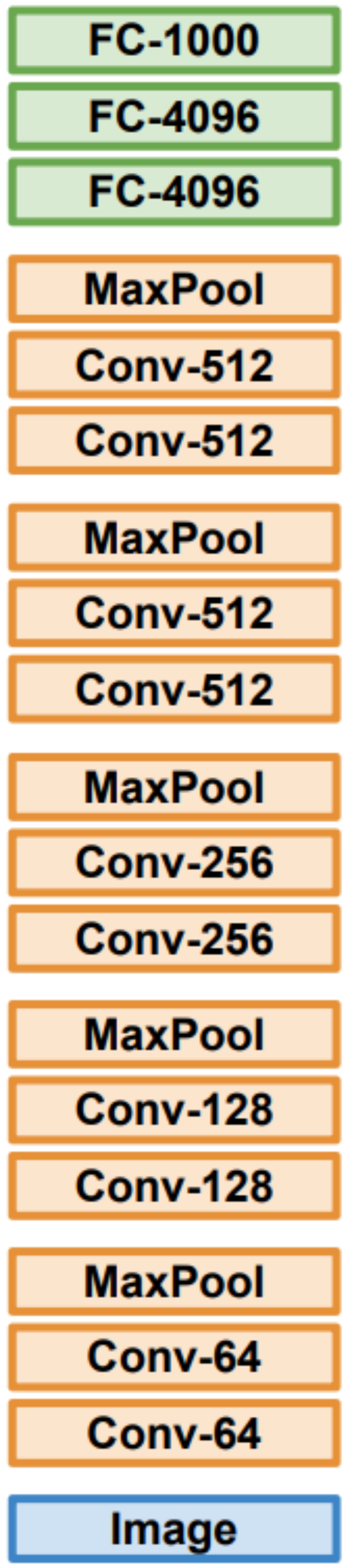
This strategy is PERVASIVE.

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

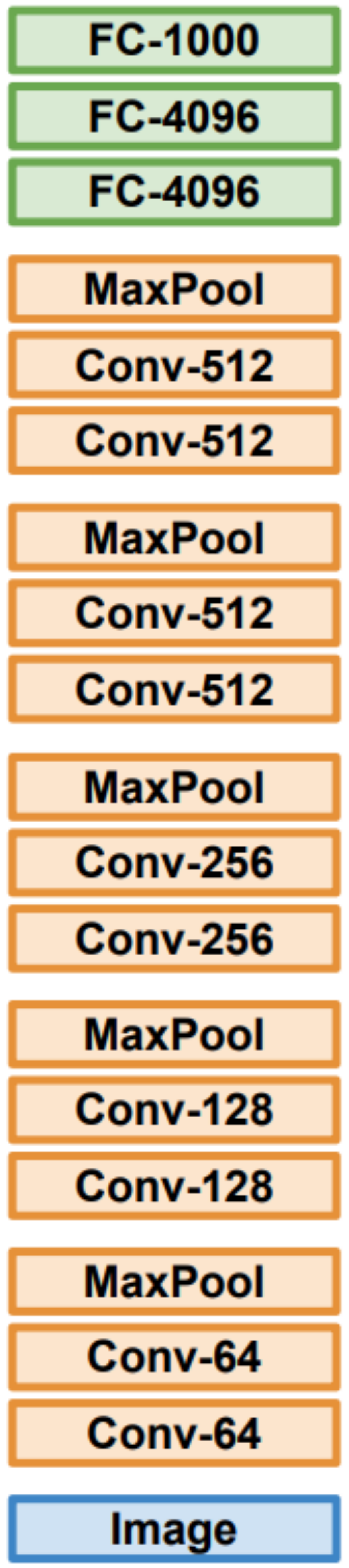
Train on **ImageNet**



Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**

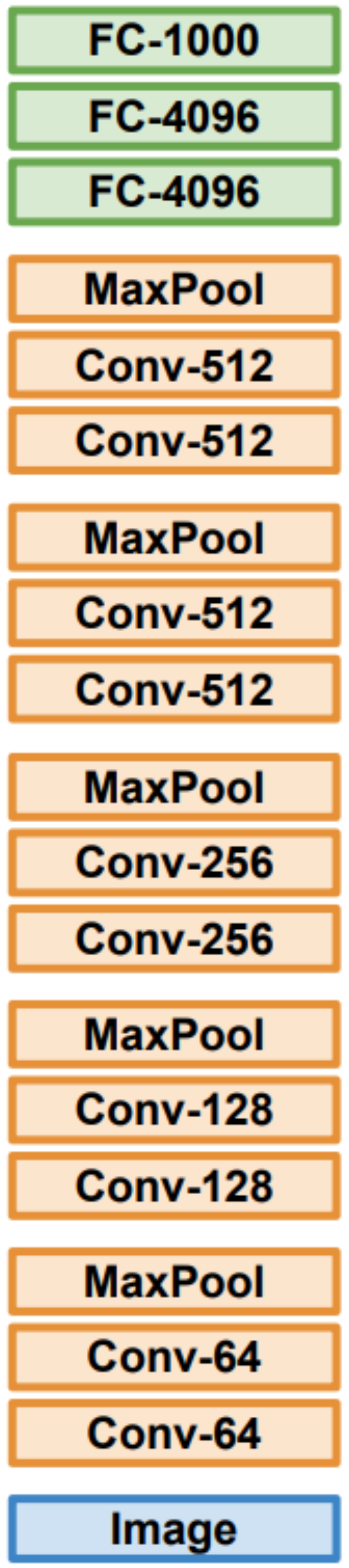


Why on **ImageNet**?

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



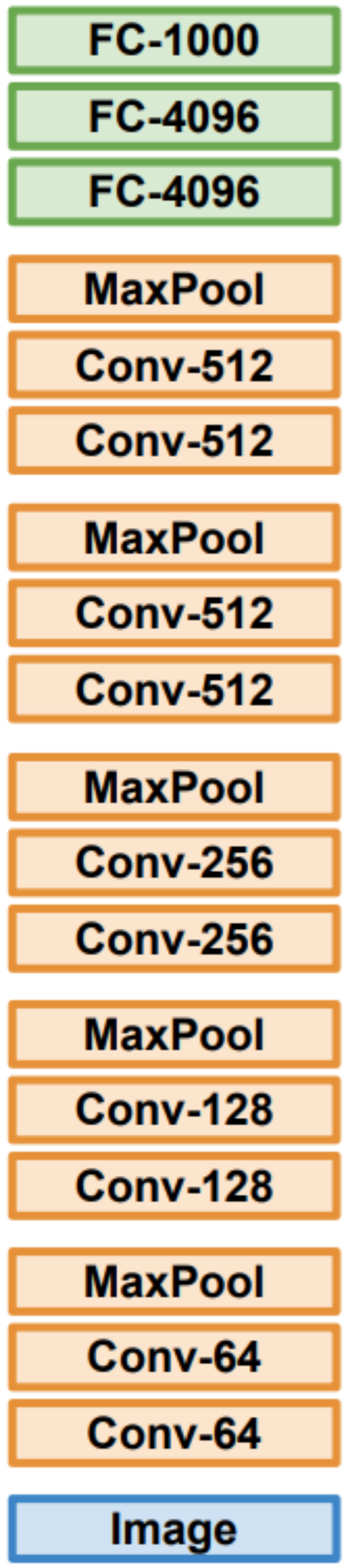
Why on **ImageNet**?

- Convenience, lots of **data**
- We know how to **train these well**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



Why on **ImageNet**?

- Convenience, lots of **data**
- We know how to **train these well**

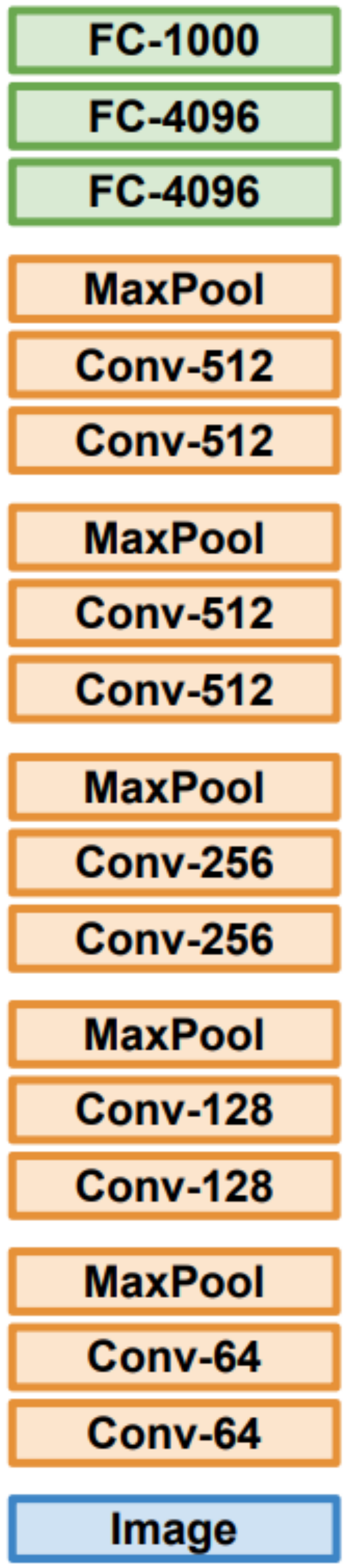
However, for some tasks we would need to start with something else (e.g., videos for optical flow)

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**

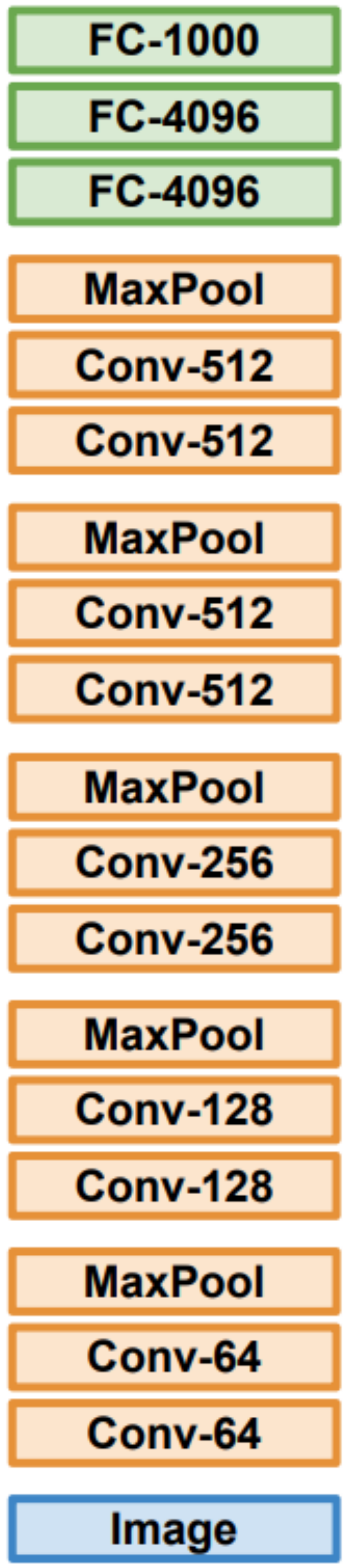


Small dataset with C classes

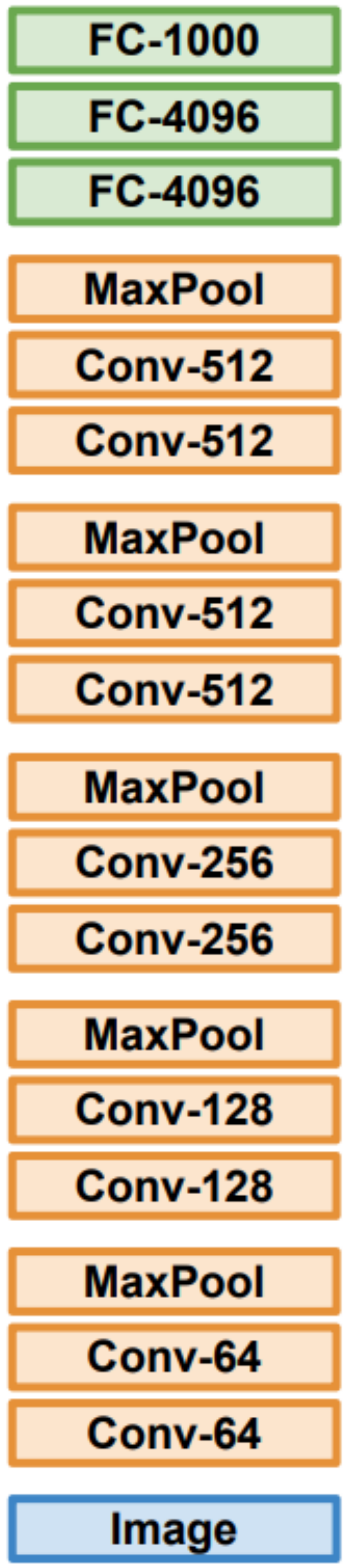
Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



Small dataset with C classes

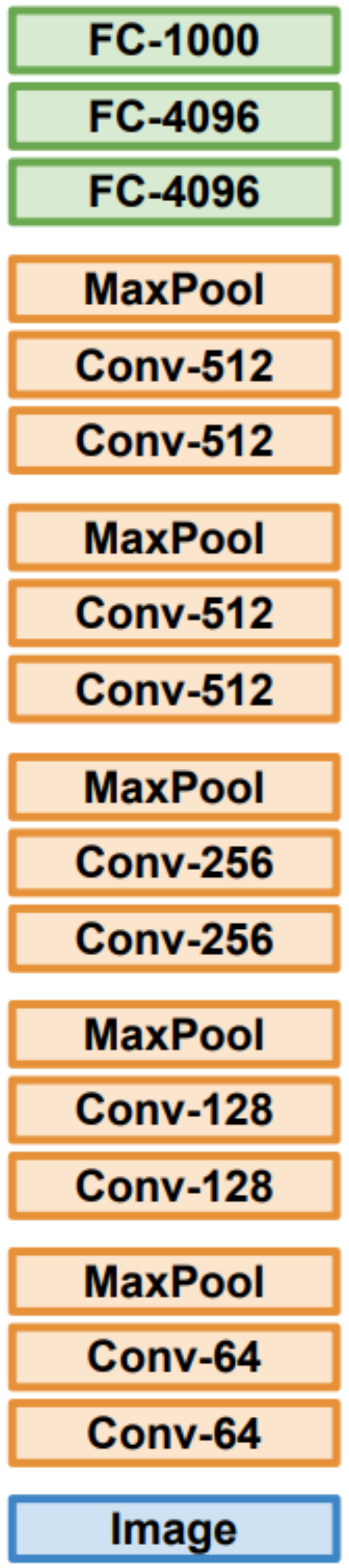


* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

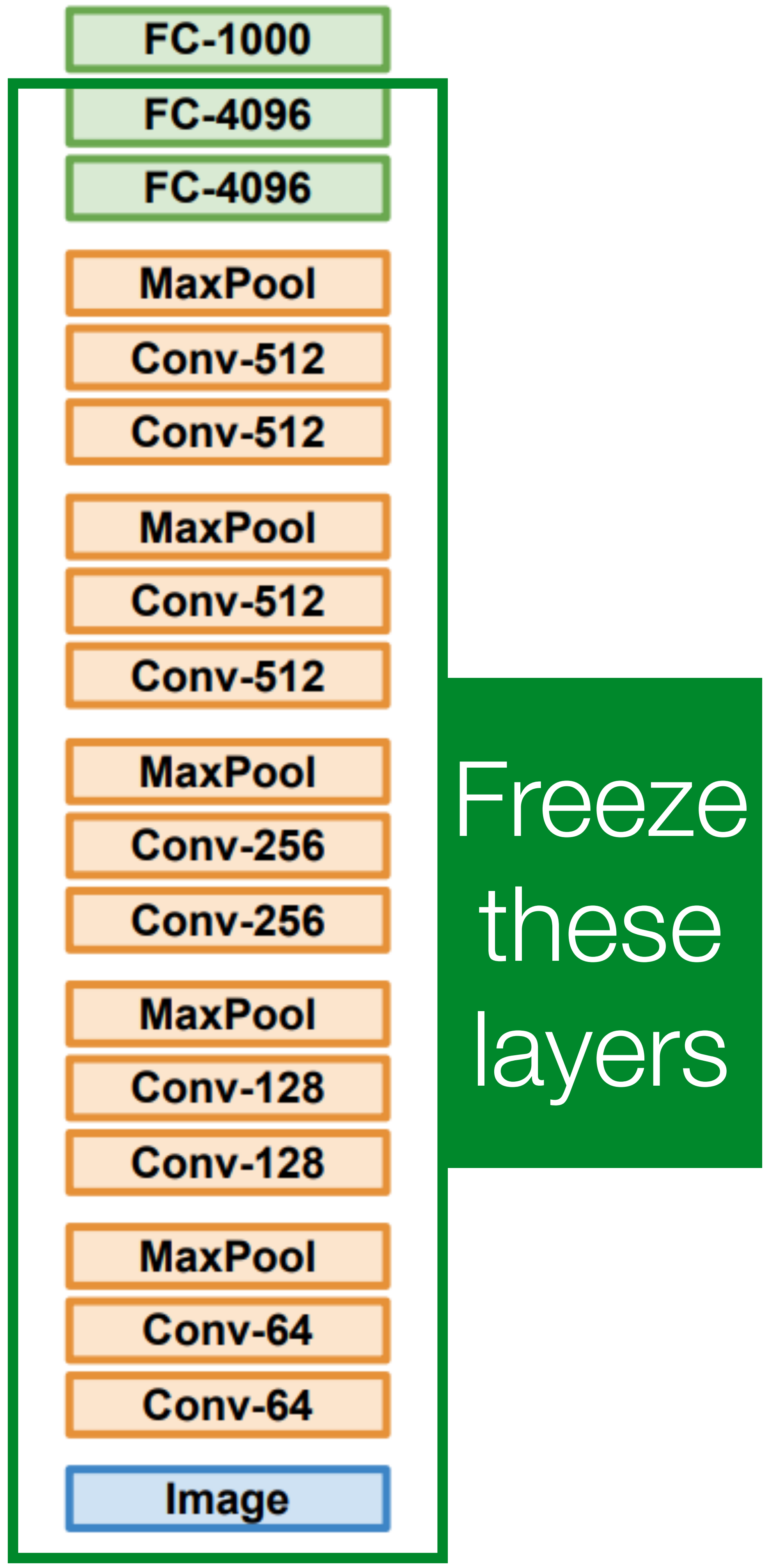
Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**

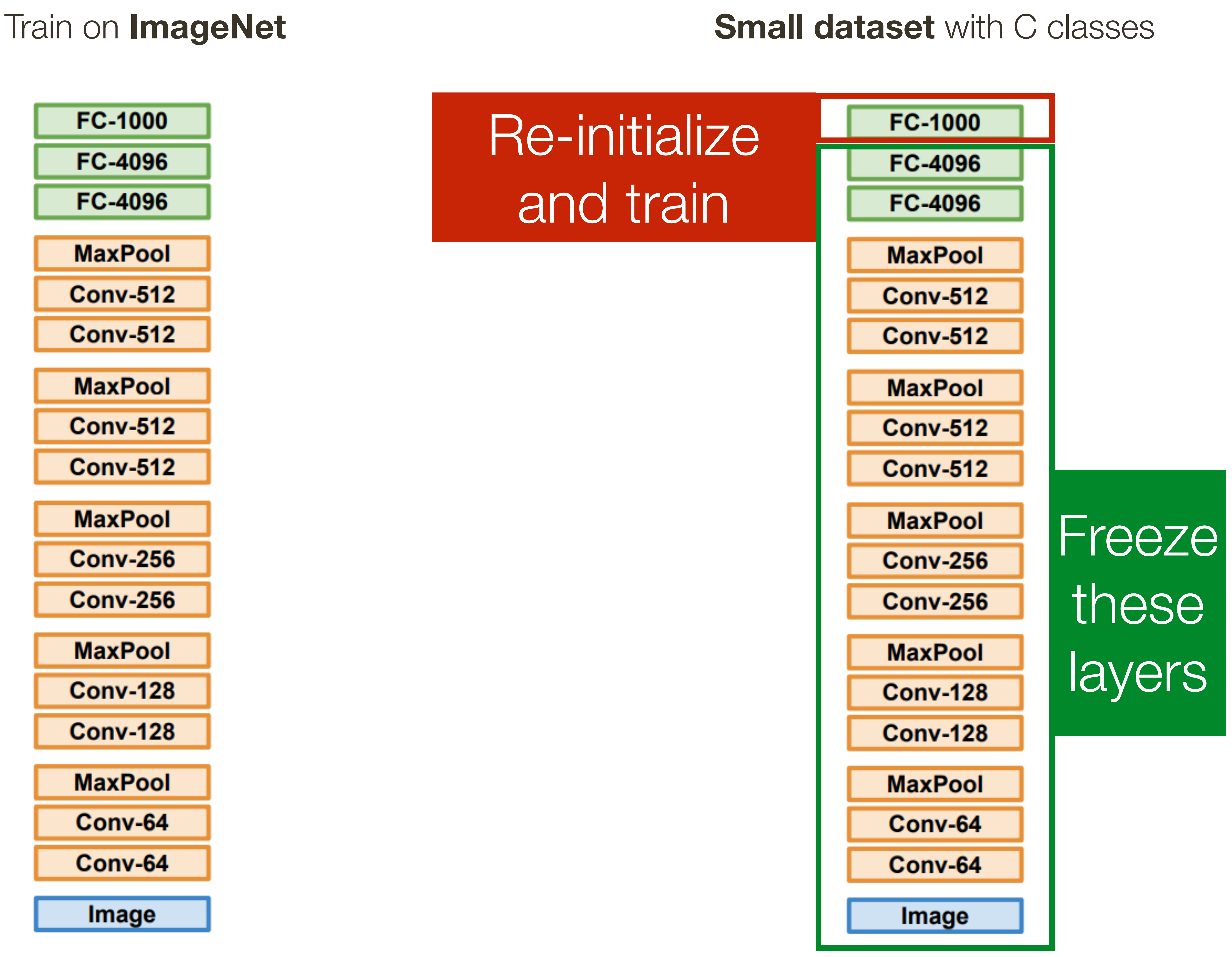


Small dataset with C classes



Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

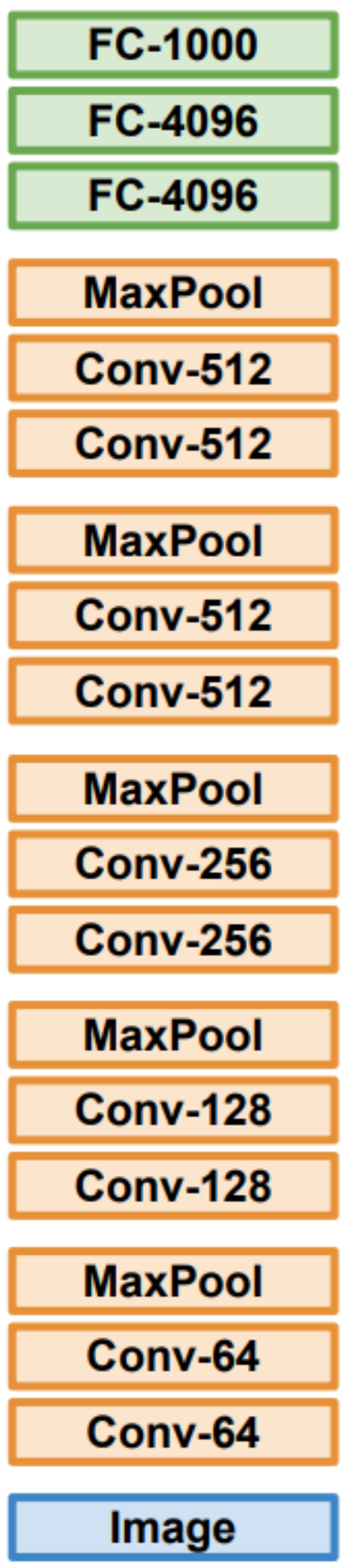


* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

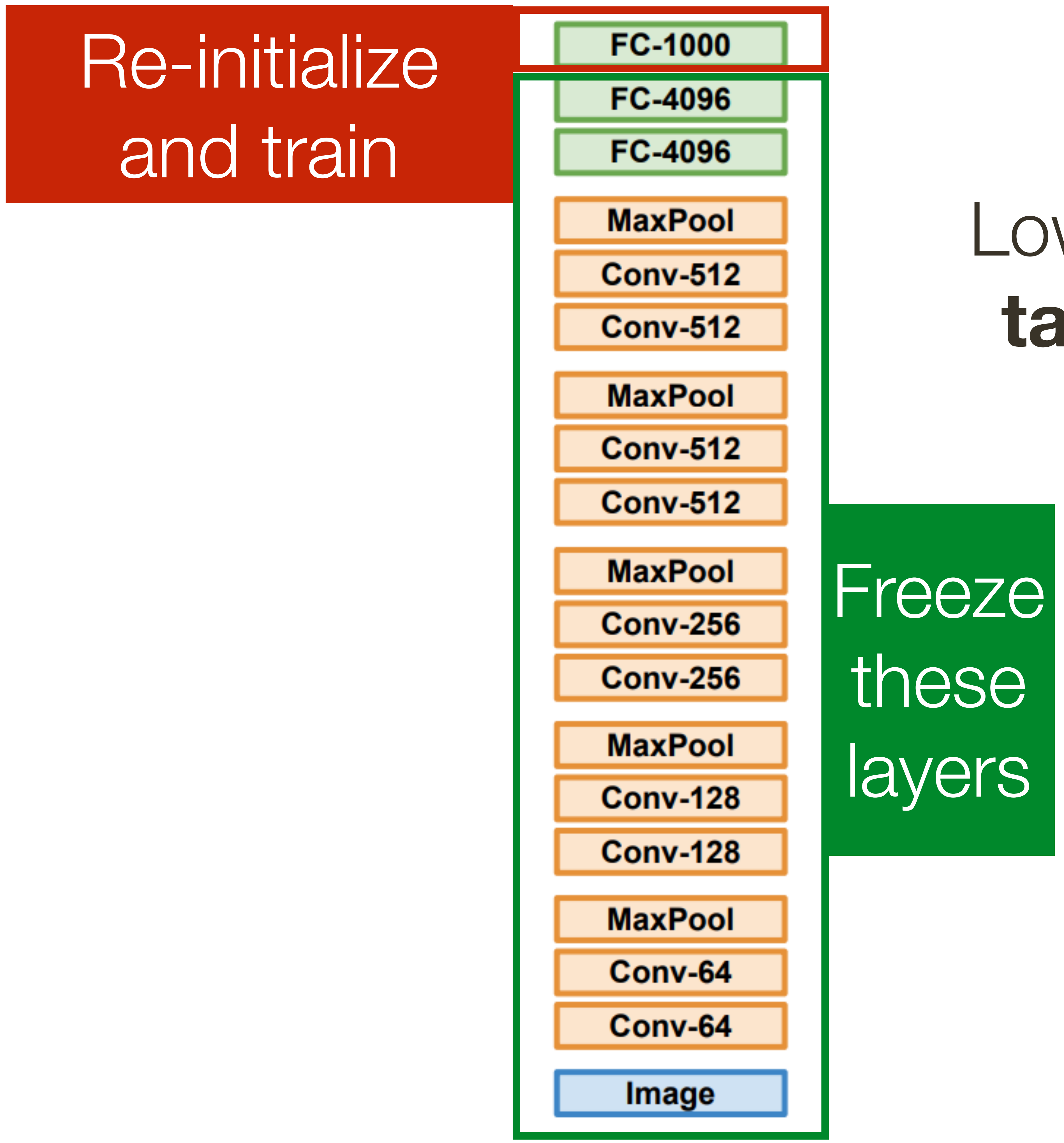
Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



Small dataset with C classes



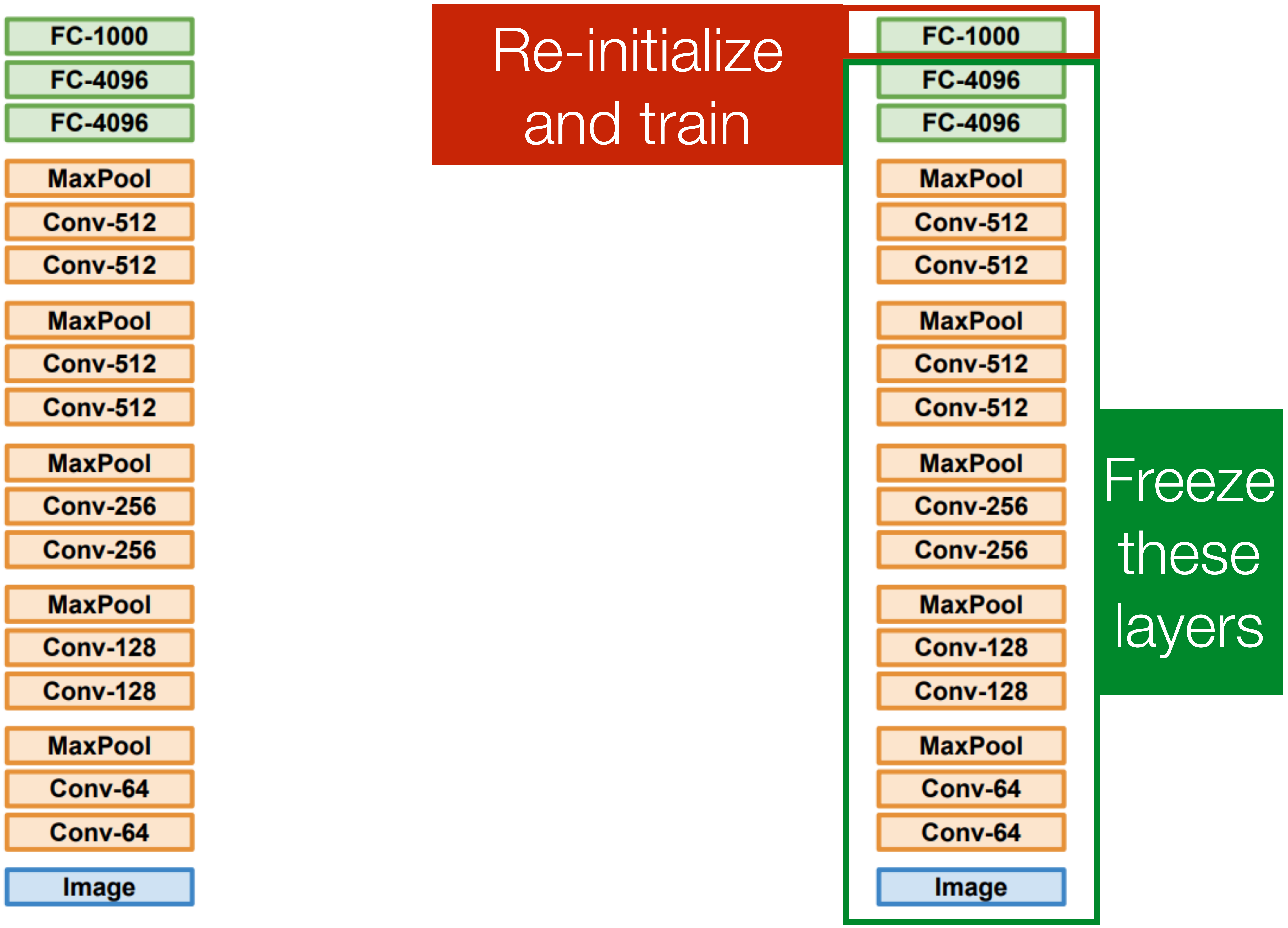
Lower levels of the CNN are at **task independent** anyways

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet** | **Small dataset** with C classes | **Larger dataset**

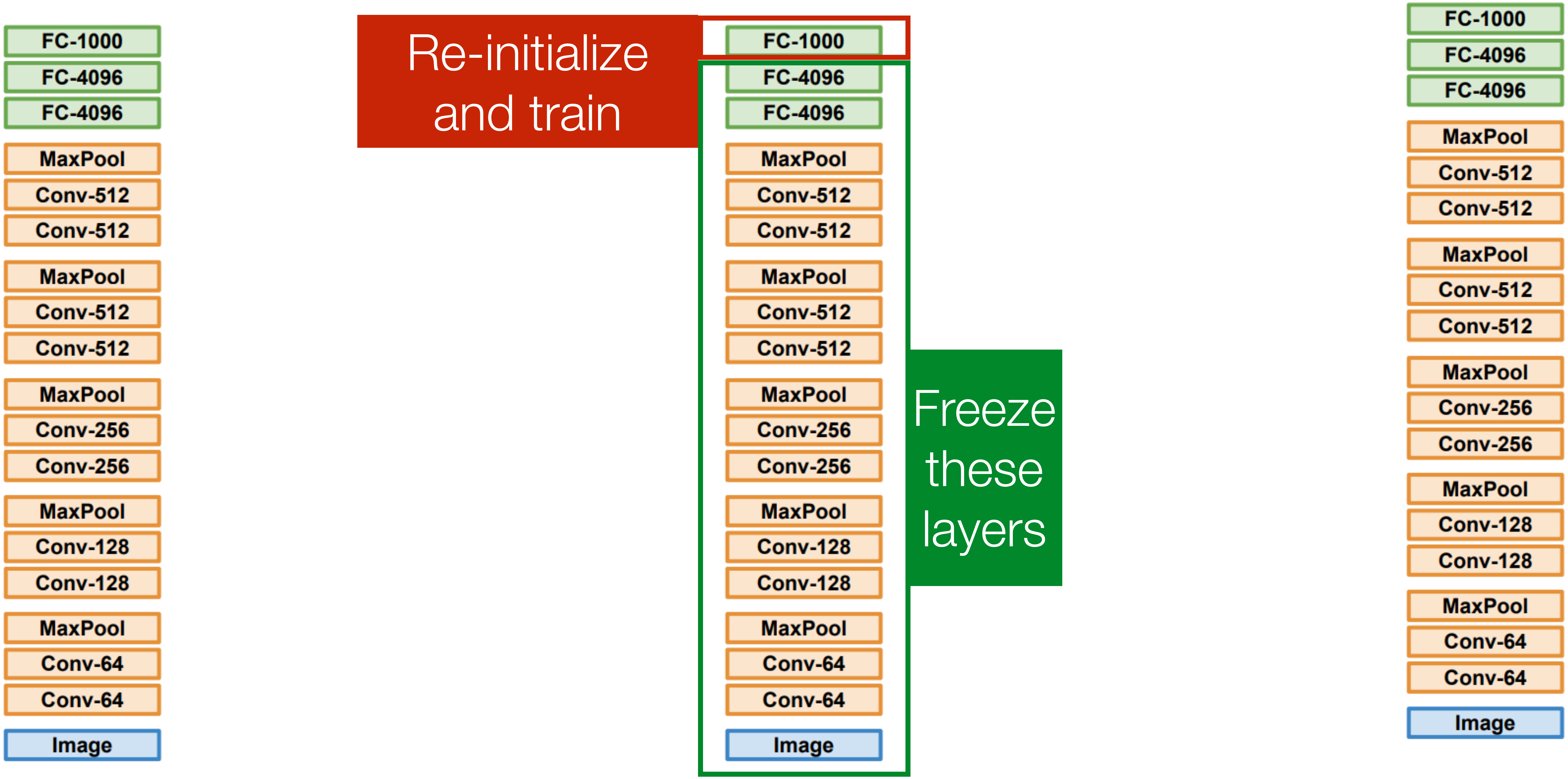


* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

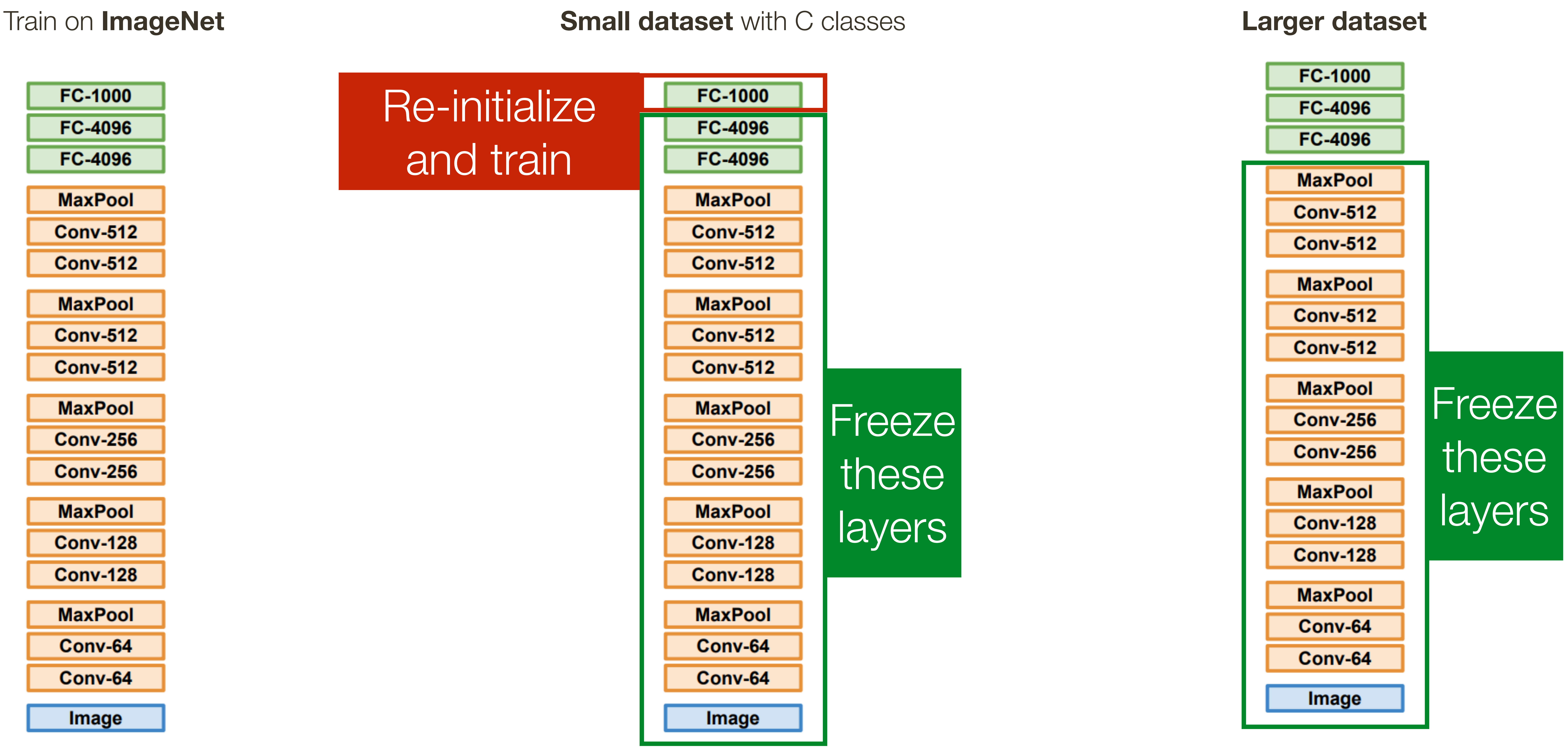
Train on **ImageNet** | **Small dataset** with C classes | **Larger dataset**



* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

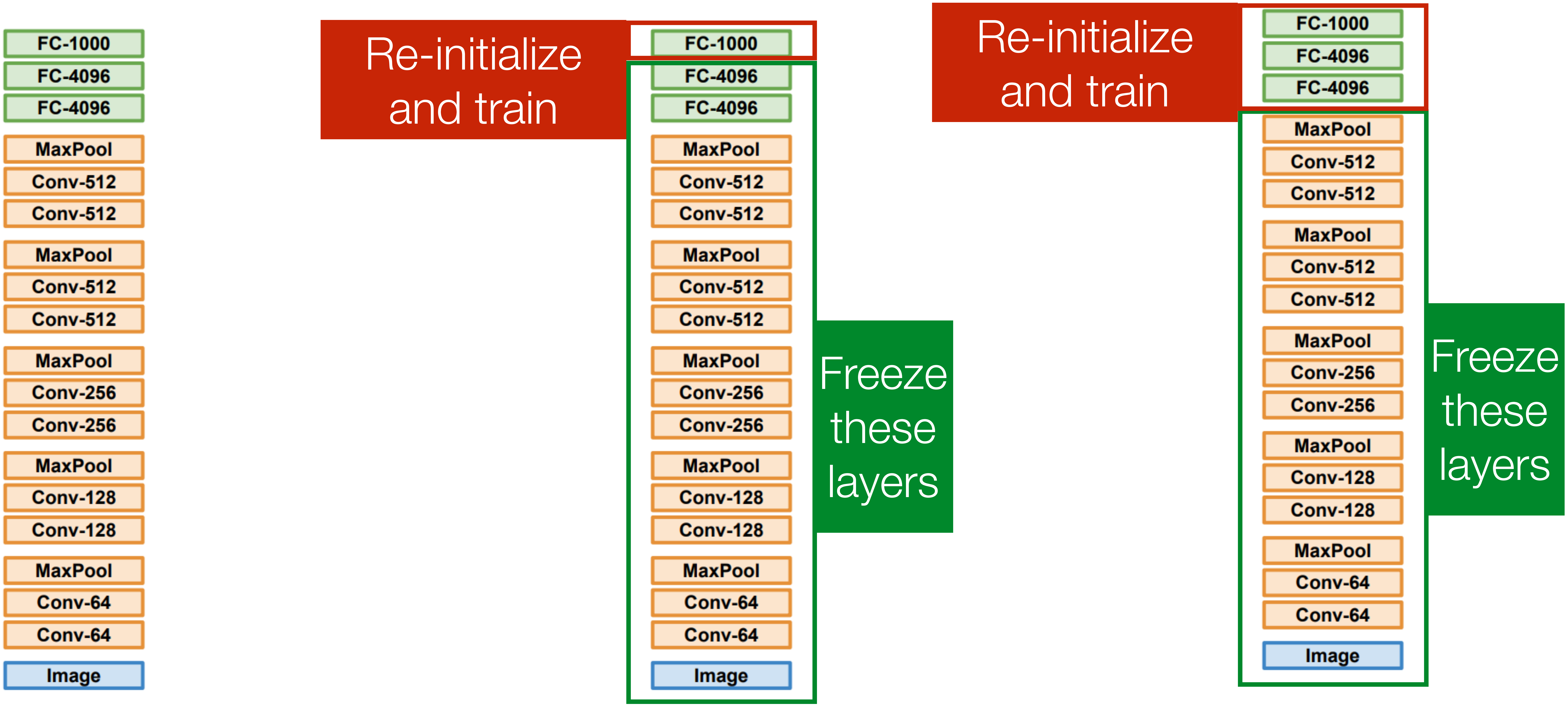


* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

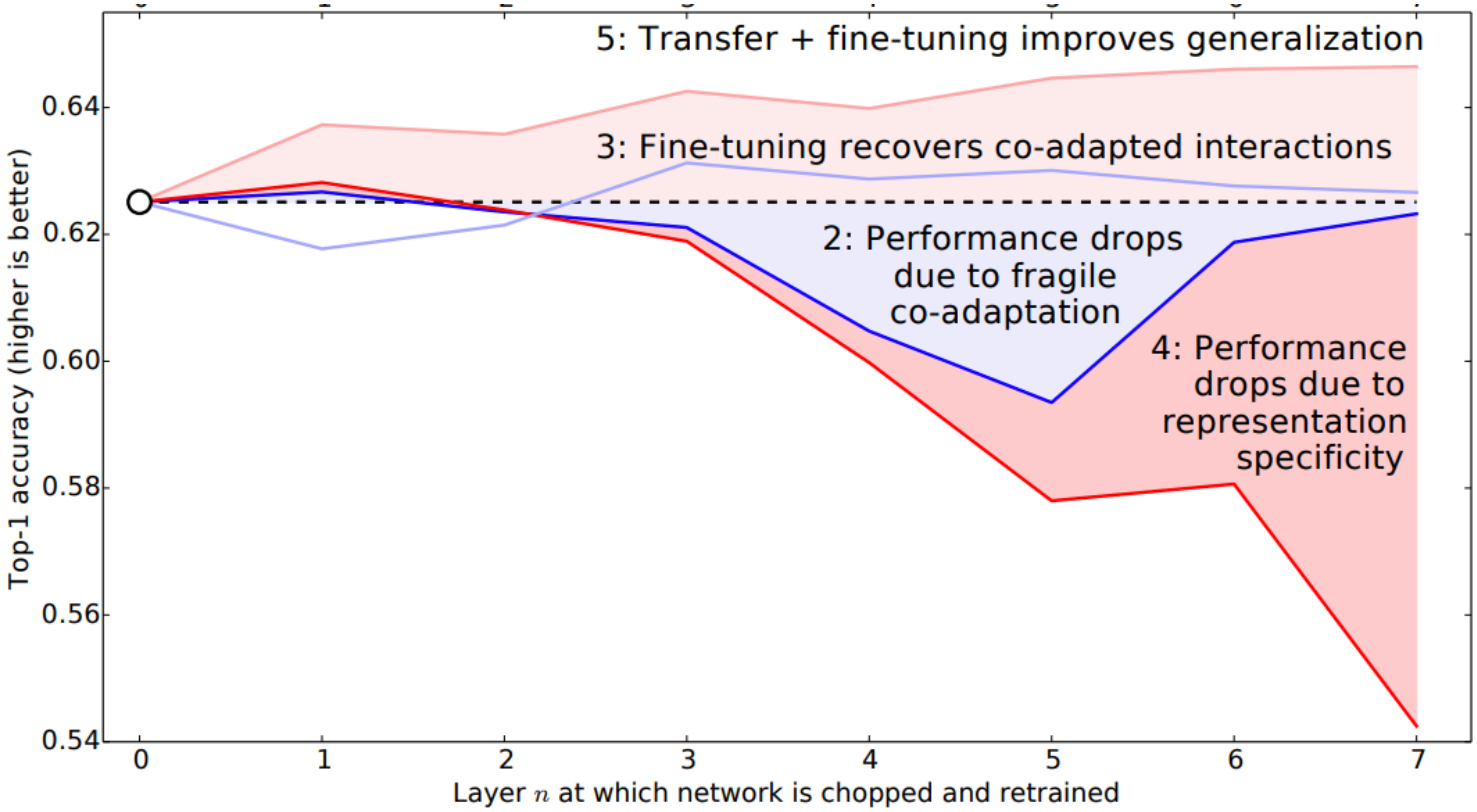
Train on **ImageNet** | **Small dataset** with C classes | **Larger dataset**



* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

Dataset A: 500 classes



Layers fine-tuned

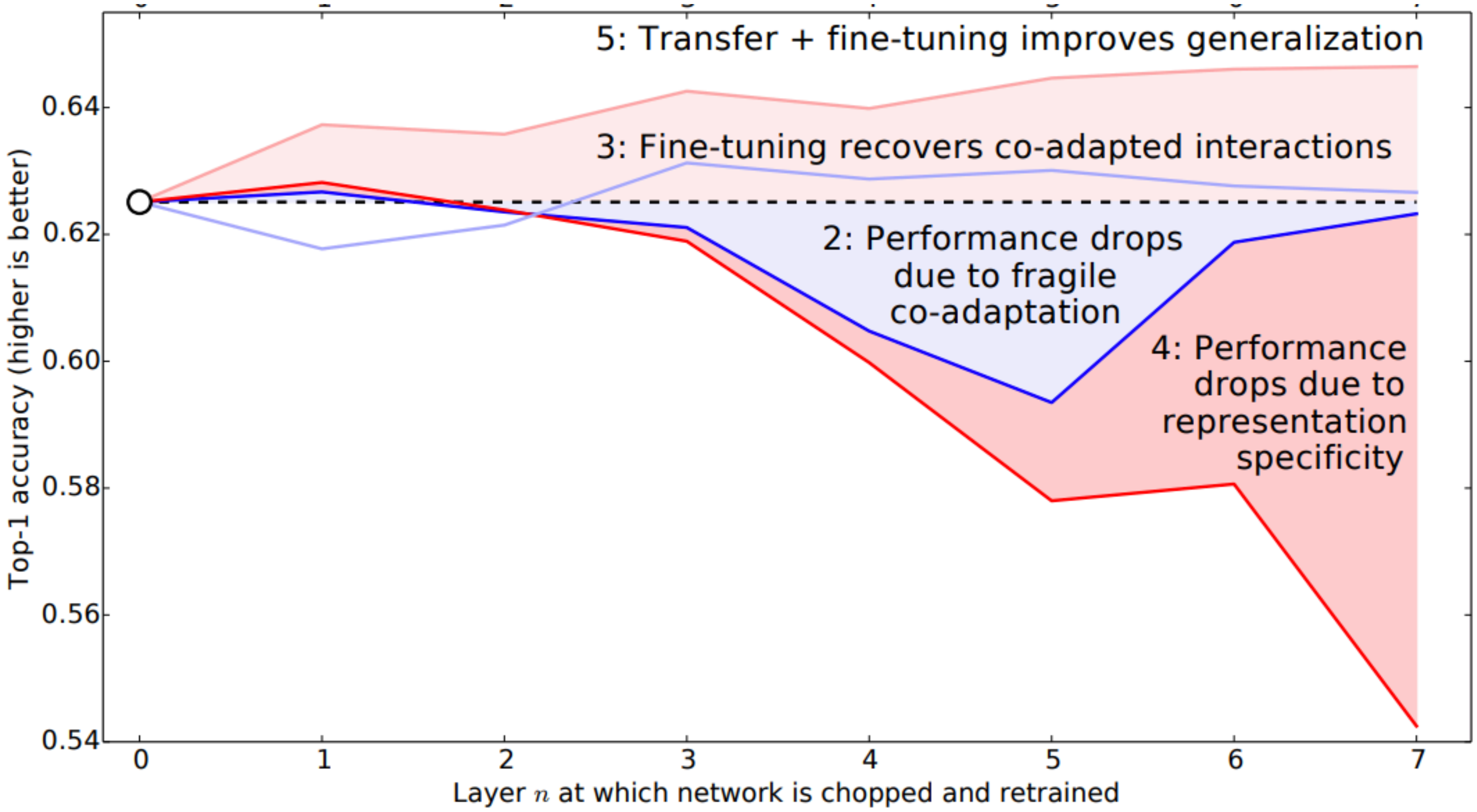
Layers fixed

[Yosinski et al., NIPS 2014]

Transfer Learning with CNNs

Dataset A: 500 classes

Dataset B: (different) 500 classes



Layers fine-tuned

Layers fixed

[Yosinski et al., NIPS 2014]

Layers fine-tuned

Layers fixed

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

~ 2% improved performance in practice

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

~ 2% improved performance in practice

Alternative: Multiple snapshots of the single model during training!

Model **Ensemble**

Training: Train multiple independent models

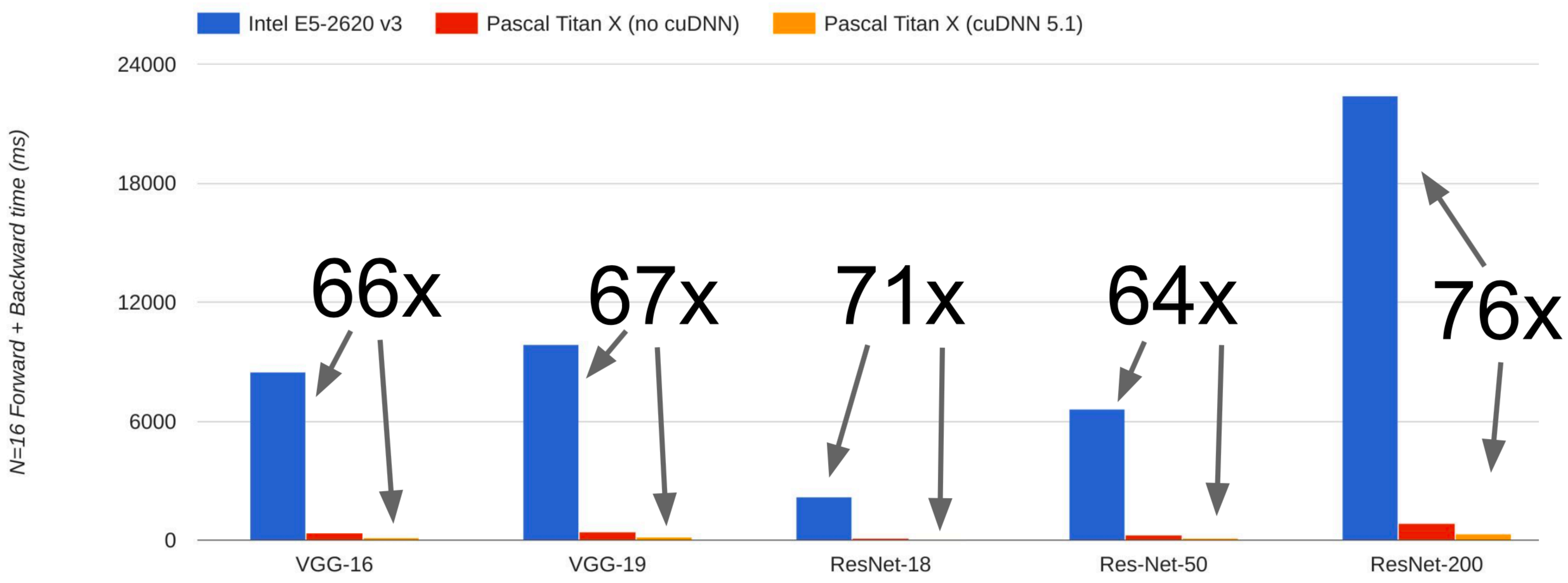
Test: Average their results

~ 2% improved performance in practice

Alternative: Multiple snapshots of the single model during training!

Improvement: Instead of using the actual parameter vector, keep a moving average of the parameter vector and use that at test time (Polyak averaging)

CPU vs. **GPU** (Why do we need Azure?)



Data from https://github.com/jcjohnson/cnn-benchmarks

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Frameworks: Super quick overview

1. Easily **build computational graphs**

2. Easily **compute gradients** in computational graphs

3. **Run it all efficiently** on a GPU (weap cuDNN, cuBLAS, etc.)

Frameworks: Super quick overview

Core DNN Frameworks		
Caffe (UC Berkeley)	Caffe 2 (Facebook)	Puddle (Baidu)
Torch (NYU/Facebook)	**PyTorch** (Facebook)	CNTK (Microsoft)
Theano (U Montreal)	**TensorFlow** (Google)	MXNet (Amazon)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Frameworks: Super quick overview

Core DNN Frameworks		
Caffe (UC Berkeley)	Caffe 2 (Facebook)	Puddle (Baidu)
Torch (NYU/Facebook)	**PyTorch** (Facebook)	CNTK (Microsoft)
Theano (U Montreal)	**TensorFlow** (Google)	MXNet (Amazon)

Wrapper Libraries

Keras
TFLearn
TensorLayer
tf.layers
TF-Slim
tf.contrib.learn
Pretty Tensor

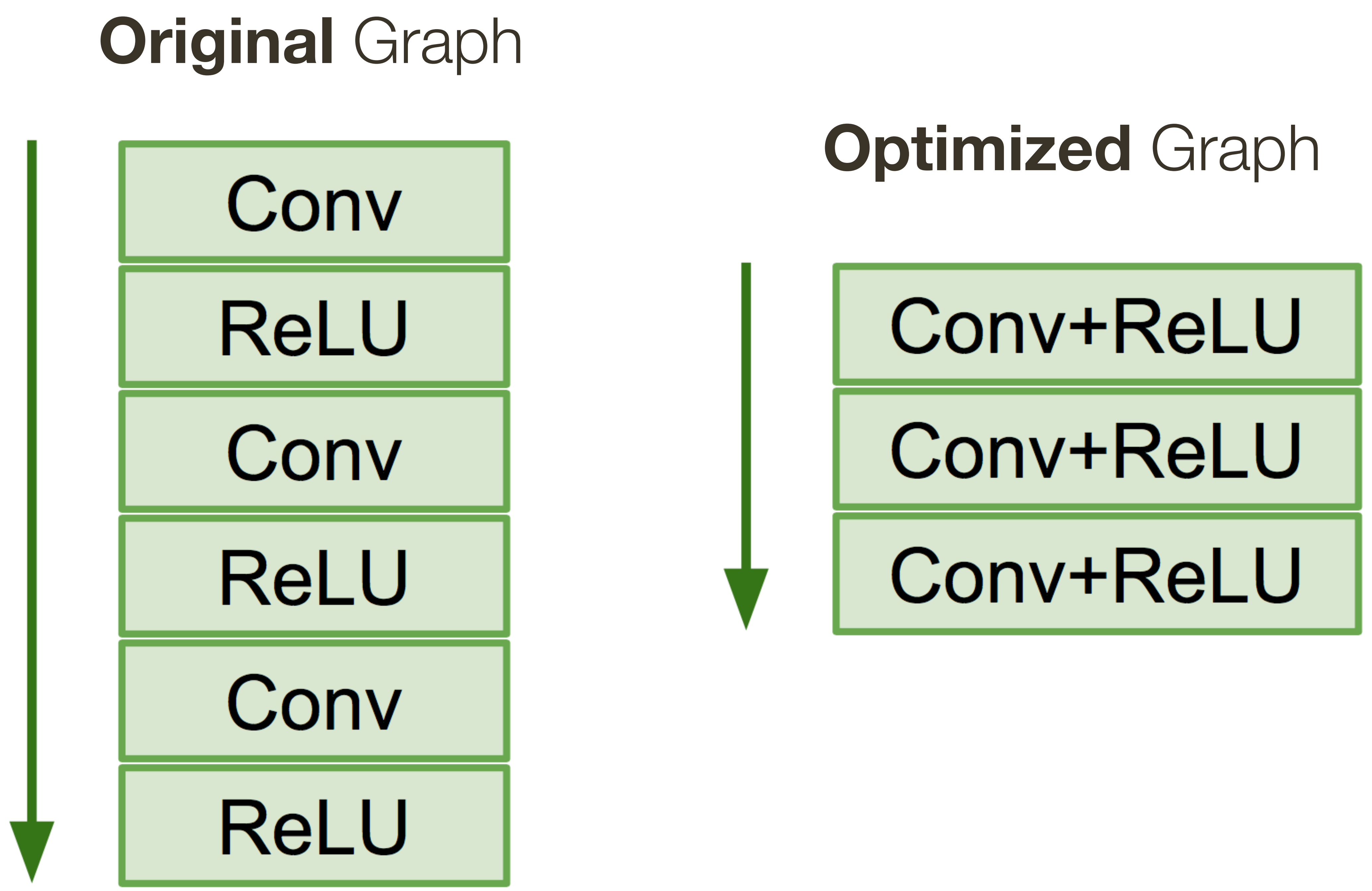
Frameworks: PyTorch vs. TensorFlow (v1)

Dynamic vs. Static computational graphs

Frameworks: PyTorch vs. TensorFlow (v1)

Dynamic vs. **Static** computational graphs

With static graphs, framework can **optimize** the graph for you before it runs!

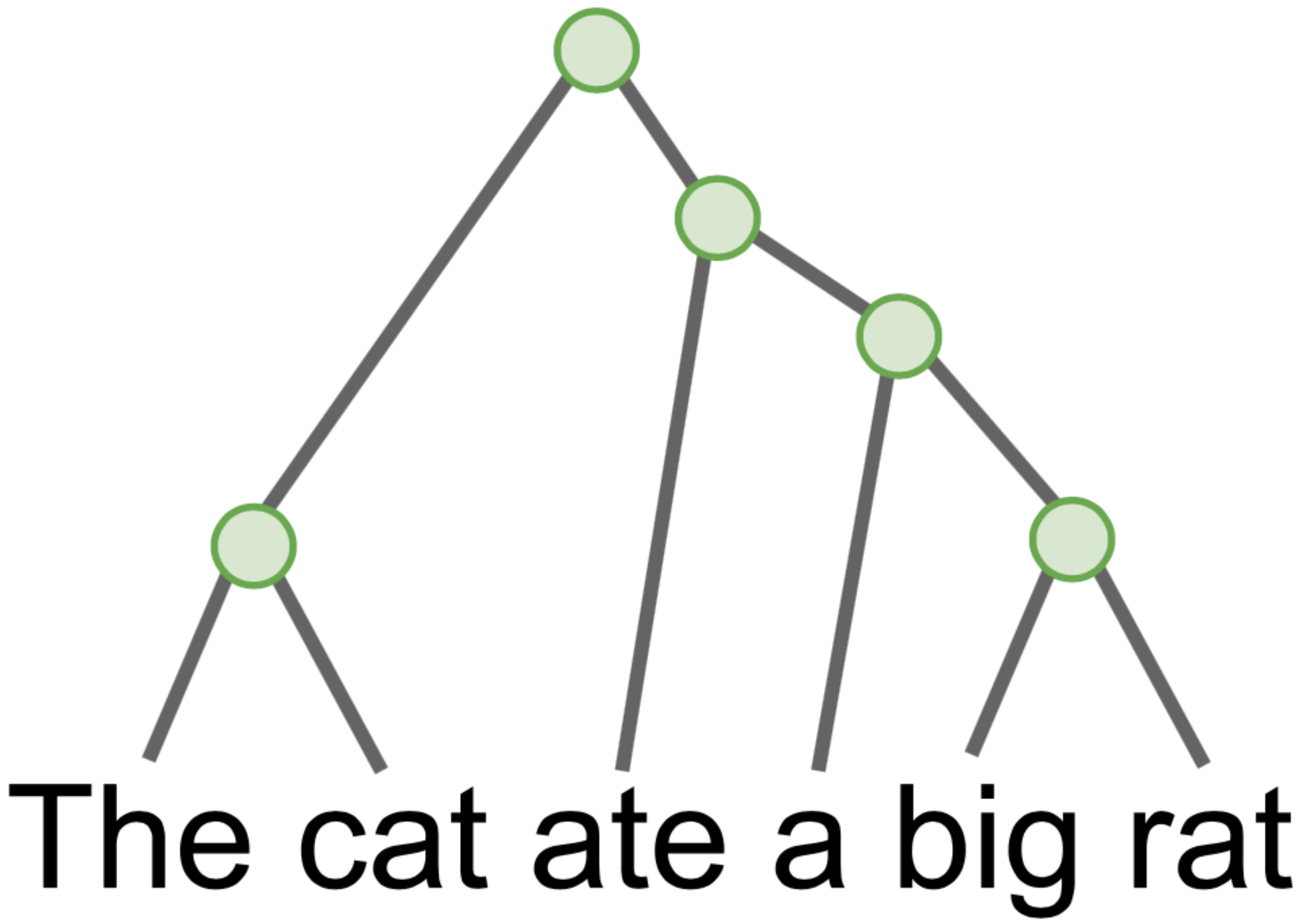


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Frameworks: PyTorch vs. TensorFlow (v1)

Dynamic vs. Static computational graphs

Graph building and execution is intertwined. Graph can be different for every sample.



PyTorch: Three levels of abstraction

Tensor: Imperative ndarray, but runs on GPU

Variable: Node in a computational graph; stores data and gradients

Module: A neural network layer; may store state or learnable weights