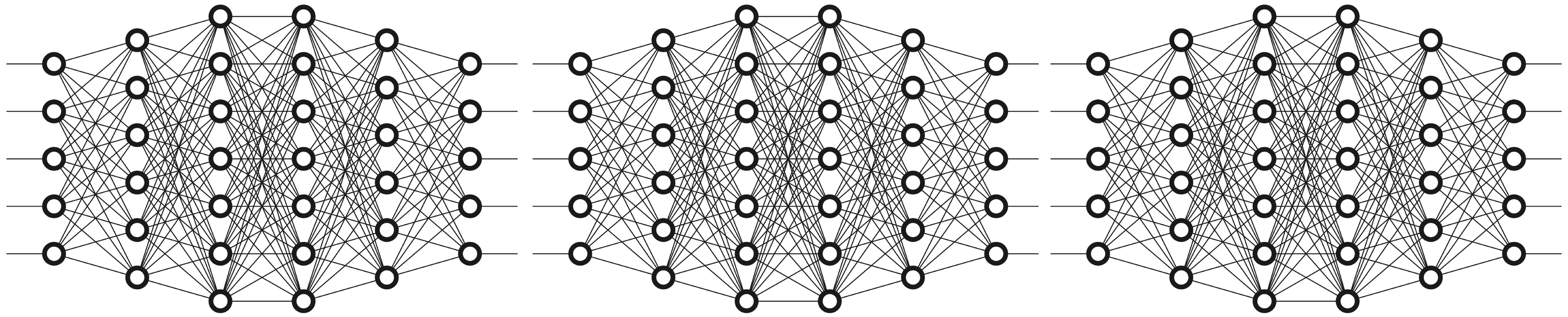
CPSC 425: Computer Vision



Lecture 31: Convolutional Neural Networks

Menu for Today (**November 25, 2020**)

Topics:

— Convolutional Layers

— Pooling Layer

Redings:

— **Today's** Lecture: N/A

— **Next** Lecture: N/A

Reminders:

— **Assignment 6**: Deep Learning due **Wednsday, December 2nd**

Today's "**fun**" Example: Yolo Object Detector

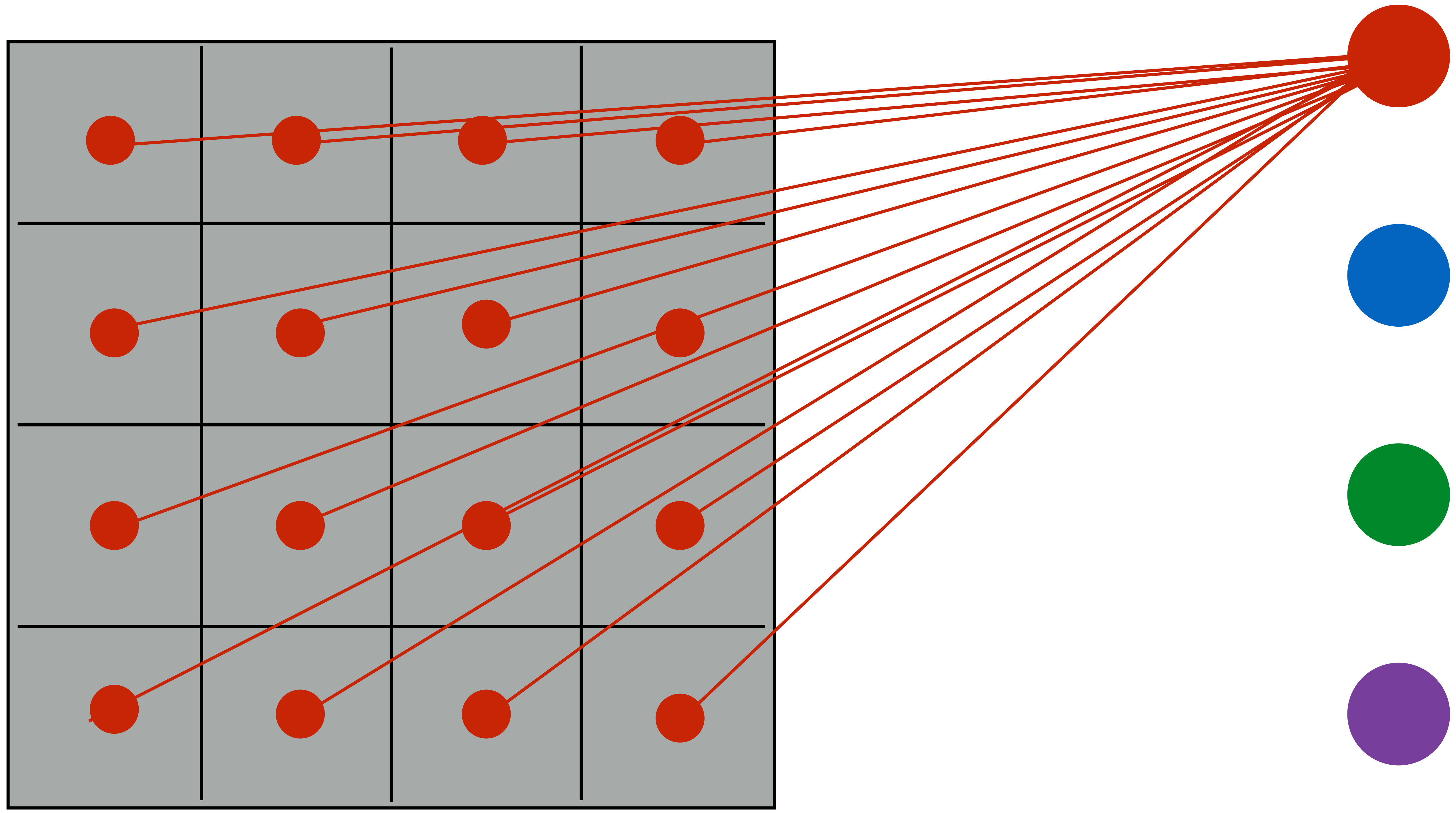
Today's “**fun**” Example: Yolo Object Detector

Fully Connected Layer

Fully Connected Layer

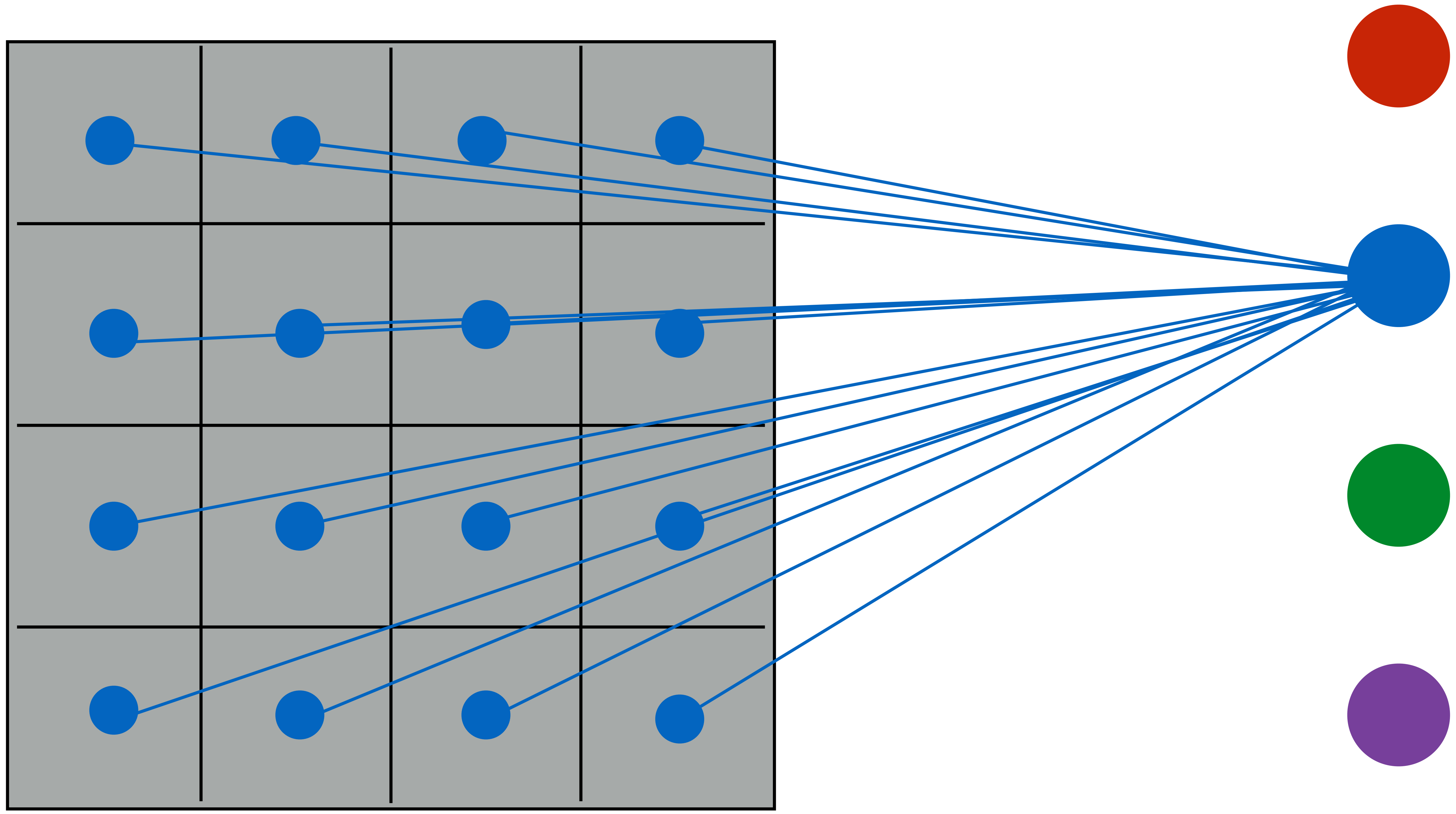
$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

Linear Layer



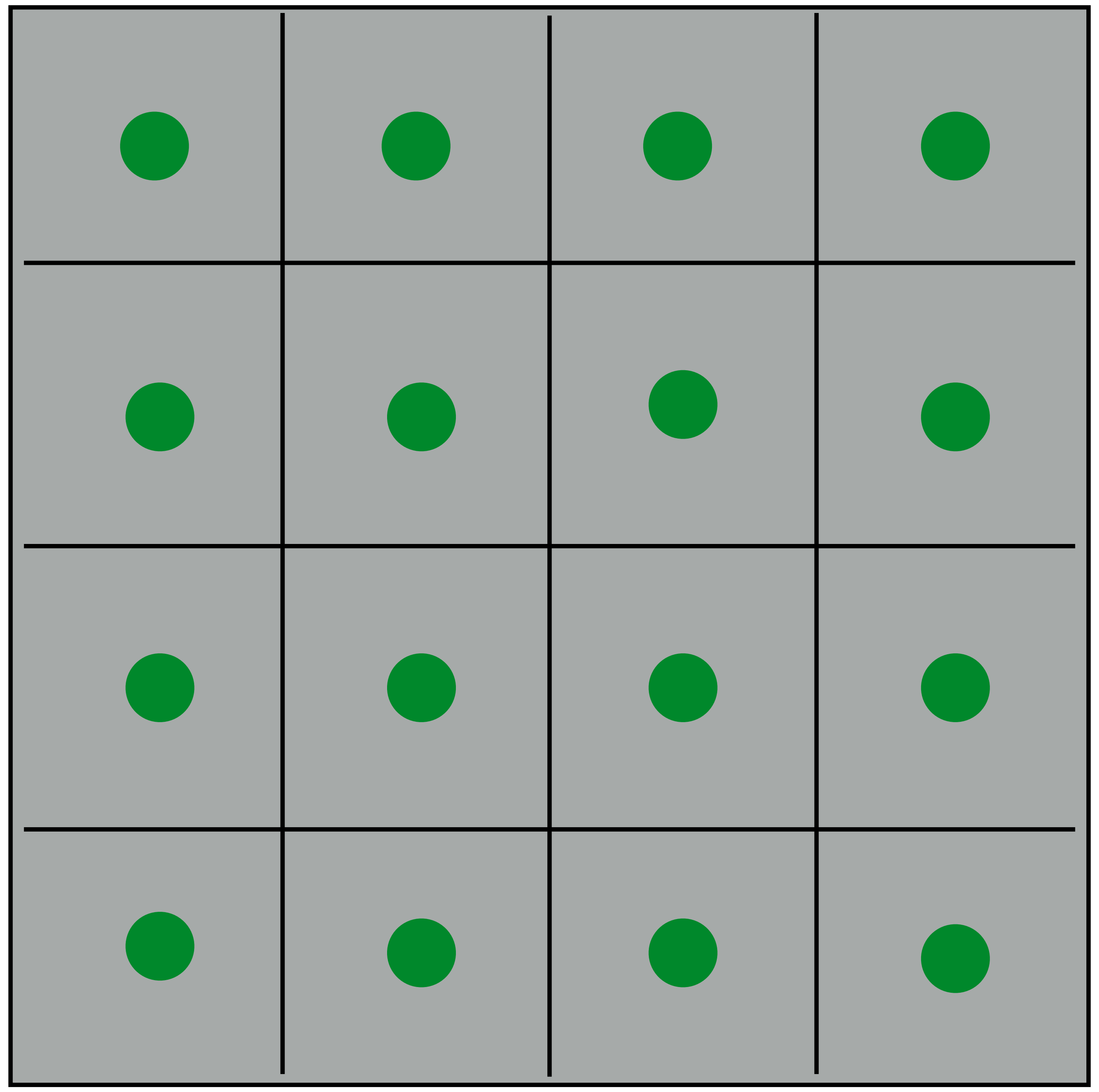
Fully Connected Layer

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$



$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{2,i,j}\mathcal{I}(i,j)+b_2\right)$$

Fully Connected Layer

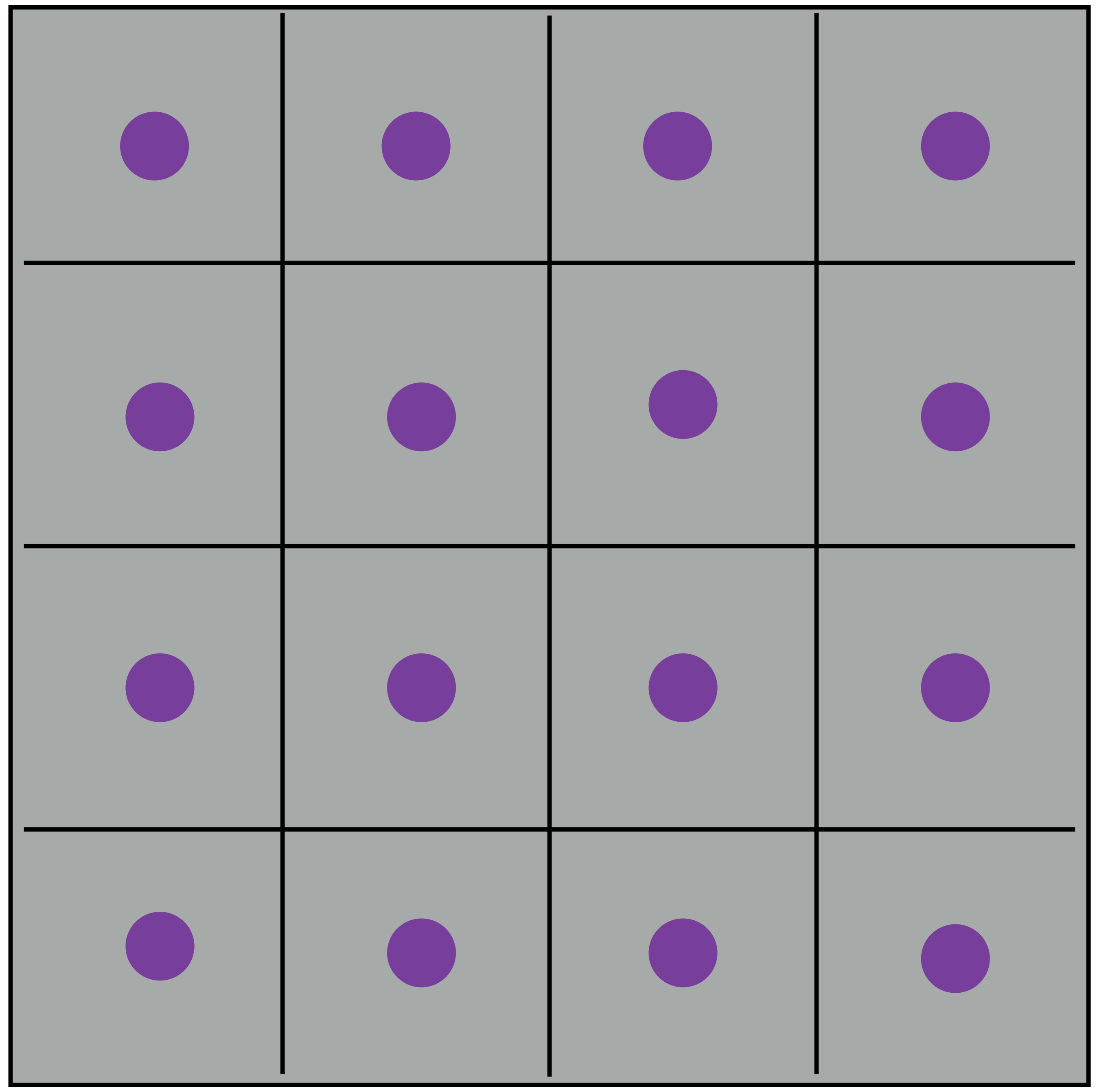


$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{2,i,j}\mathcal{I}(i,j)+b_2\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{3,i,j}\mathcal{I}(i,j)+b_3\right)$$

Fully Connected Layer



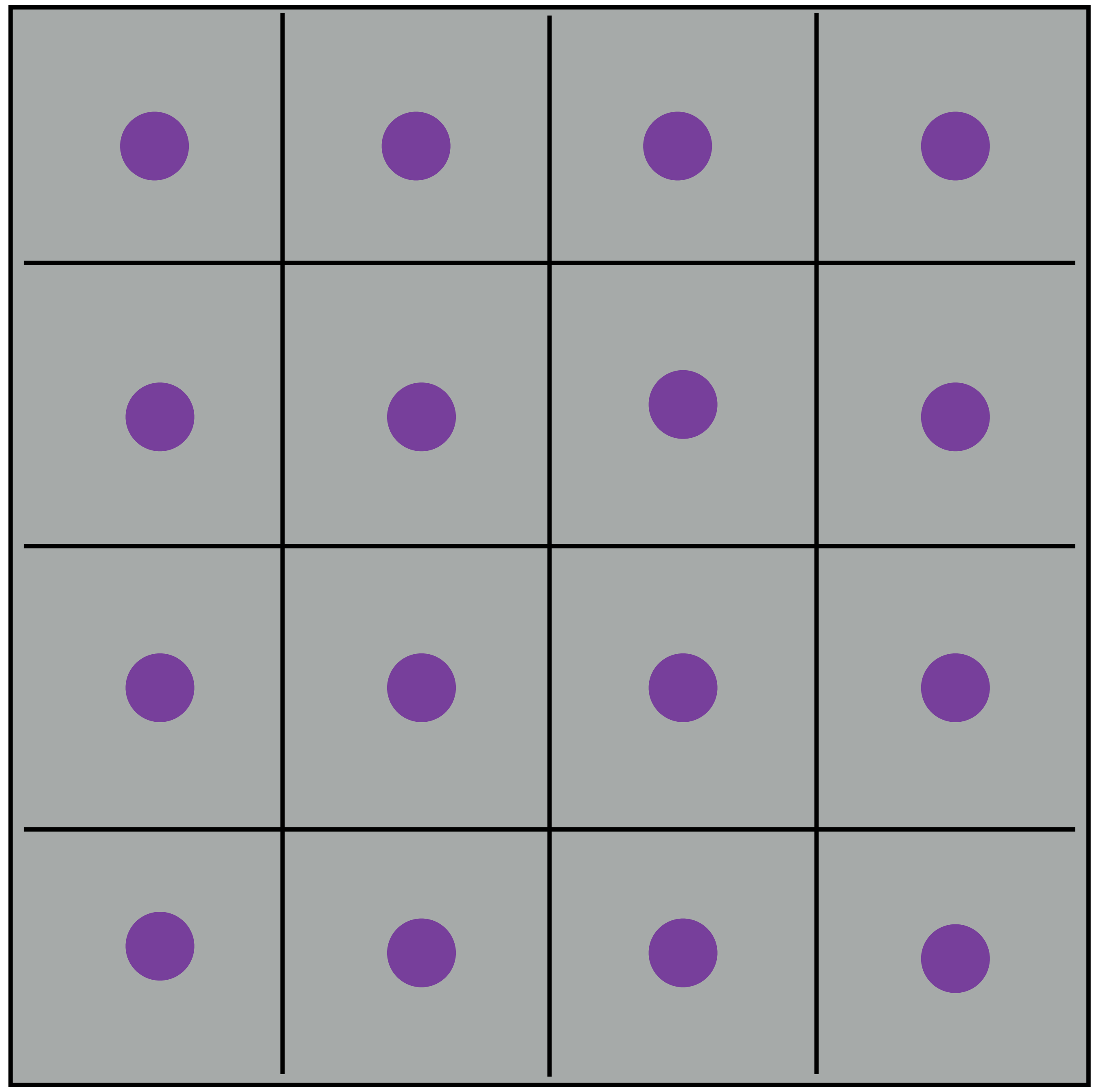
$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{2,i,j}\mathcal{I}(i,j)+b_2\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{3,i,j}\mathcal{I}(i,j)+b_3\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{4,i,j}\mathcal{I}(i,j)+b_4\right)$$

Fully Connected Layer



$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

4 x 4 + 1 = 15

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{2,i,j}\mathcal{I}(i,j)+b_2\right)$$

4 x 4 + 1 = 15

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{3,i,j}\mathcal{I}(i,j)+b_3\right)$$

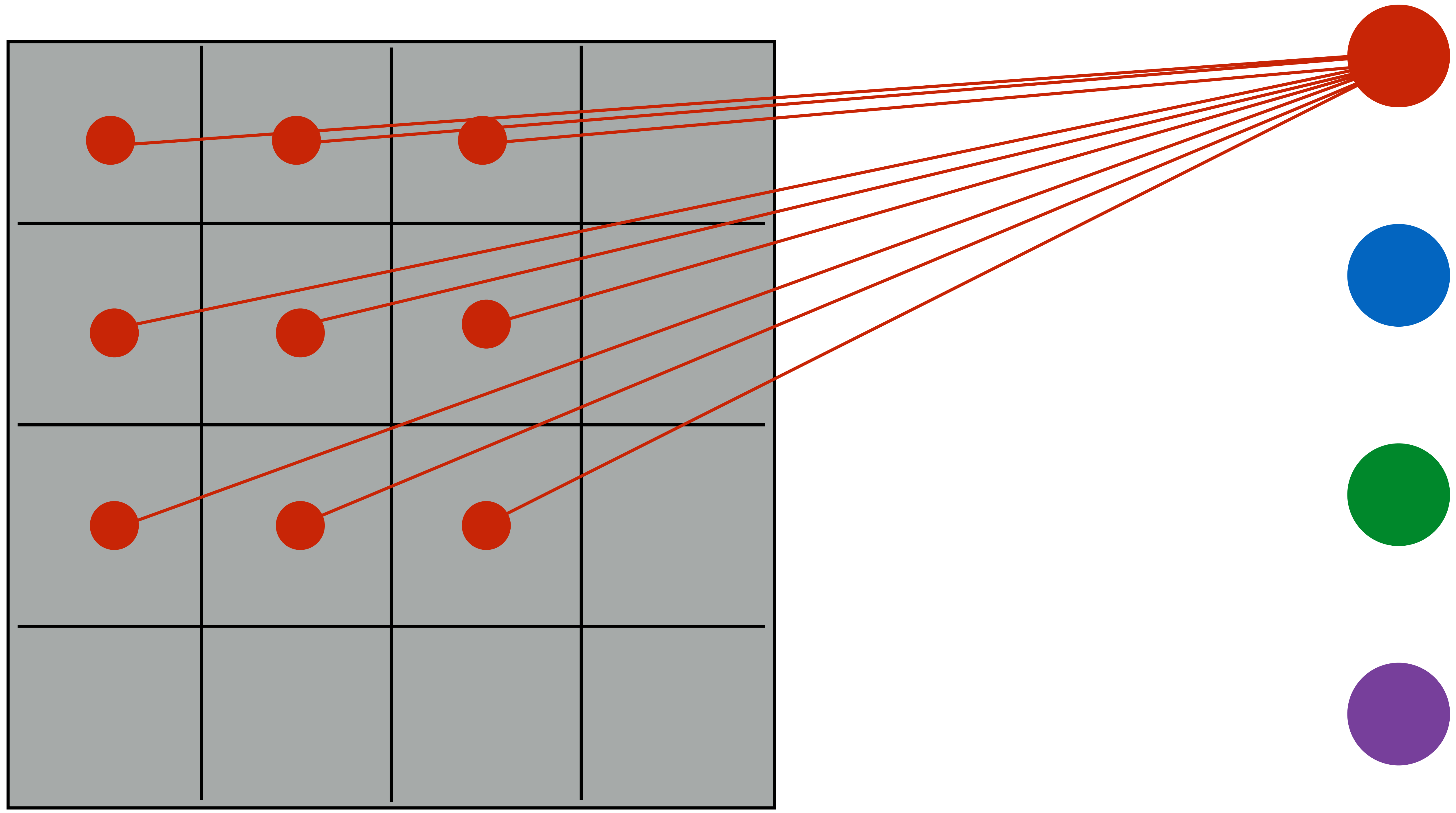
4 x 4 + 1 = 15

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{4,i,j}\mathcal{I}(i,j)+b_4\right)$$

4 x 4 + 1 = 15

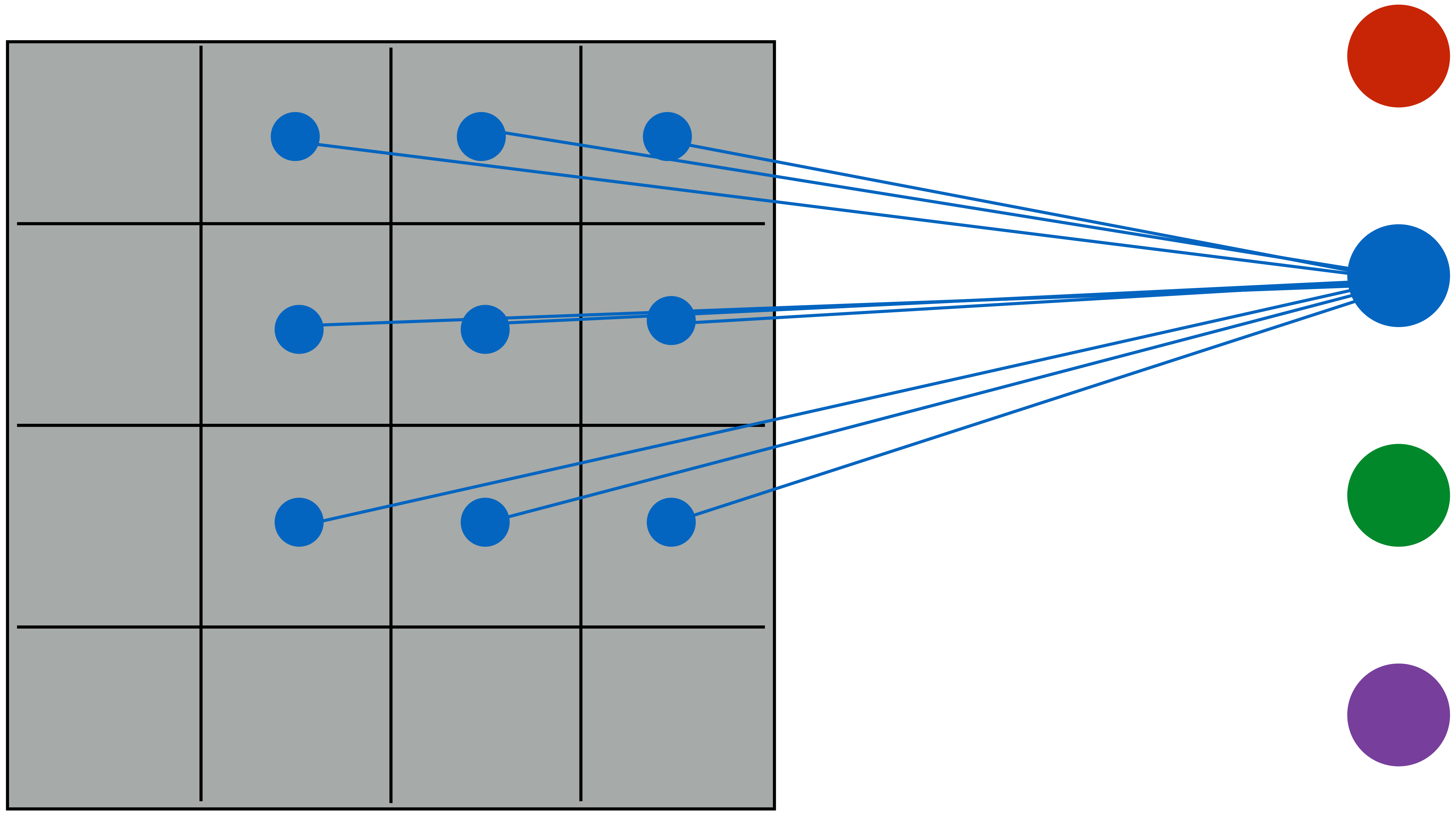
Locally Connected Layer

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$



Locally Connected Layer

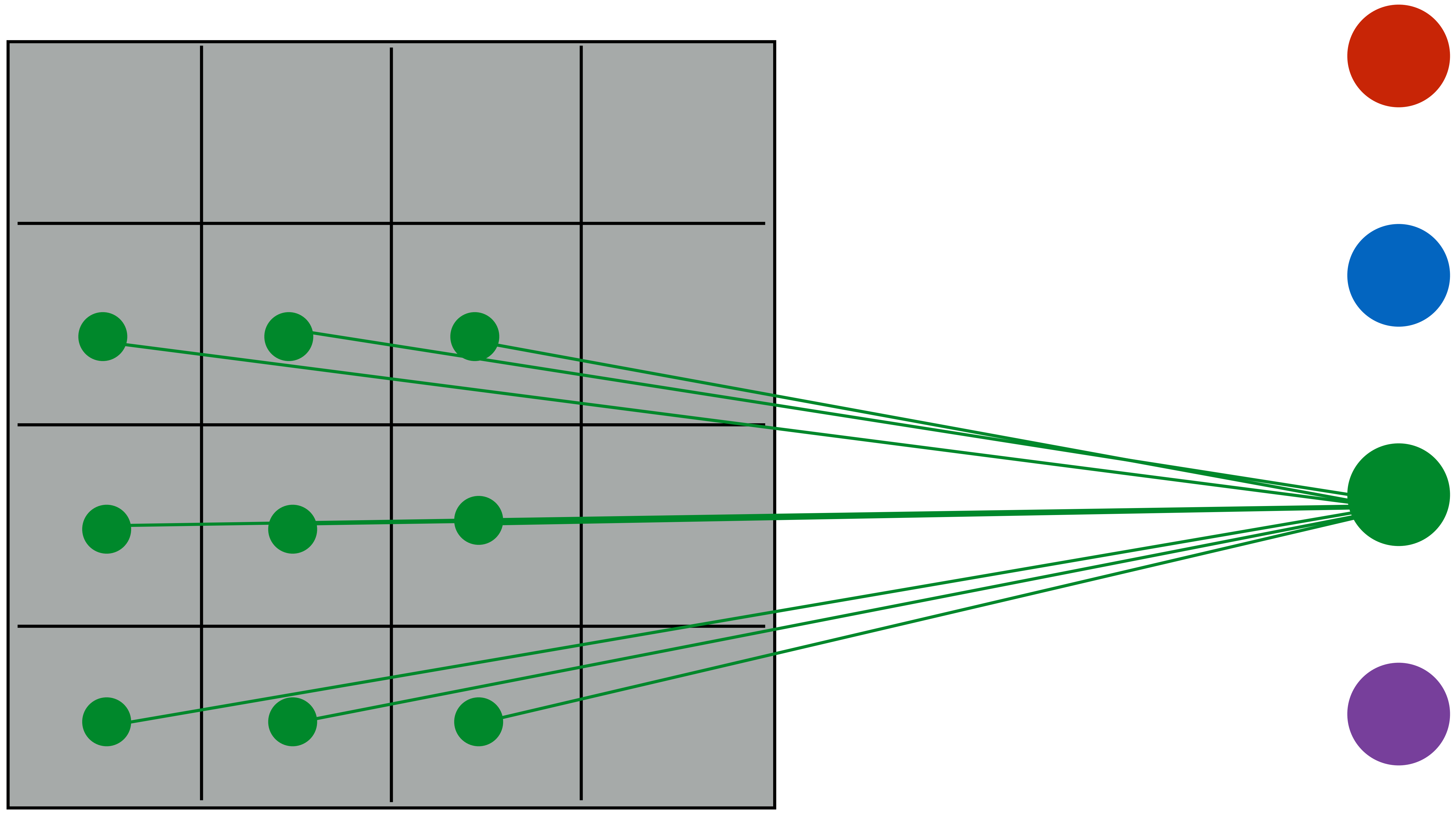
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$



$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{2,i,j}\mathcal{I}(i+1,j)+b_2\right)$$

Locally Connected Layer

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

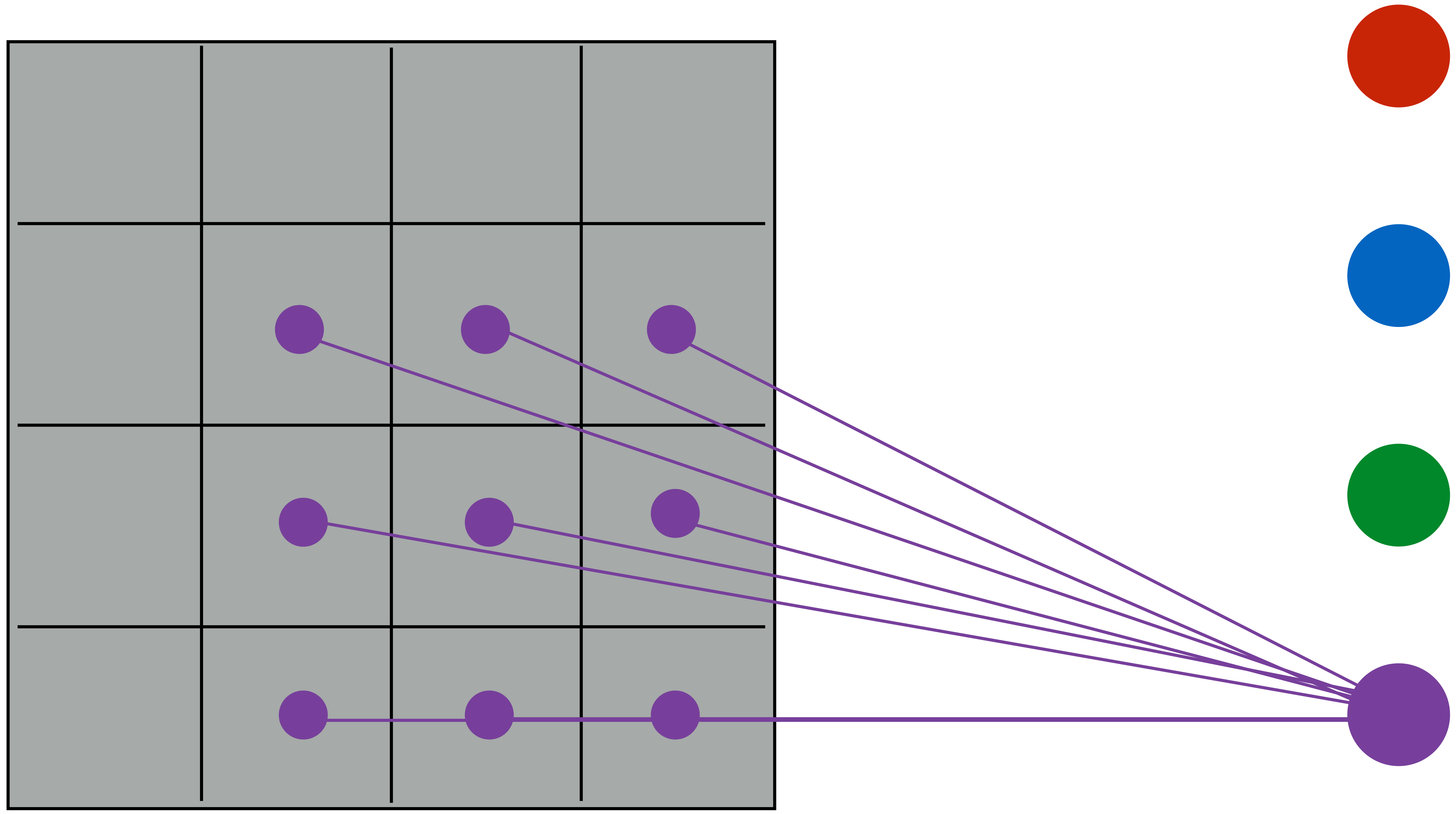


$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{2,i,j}\mathcal{I}(i+1,j)+b_2\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{3,i,j}\mathcal{I}(i,j+1)+b_3\right)$$

Locally Connected Layer

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

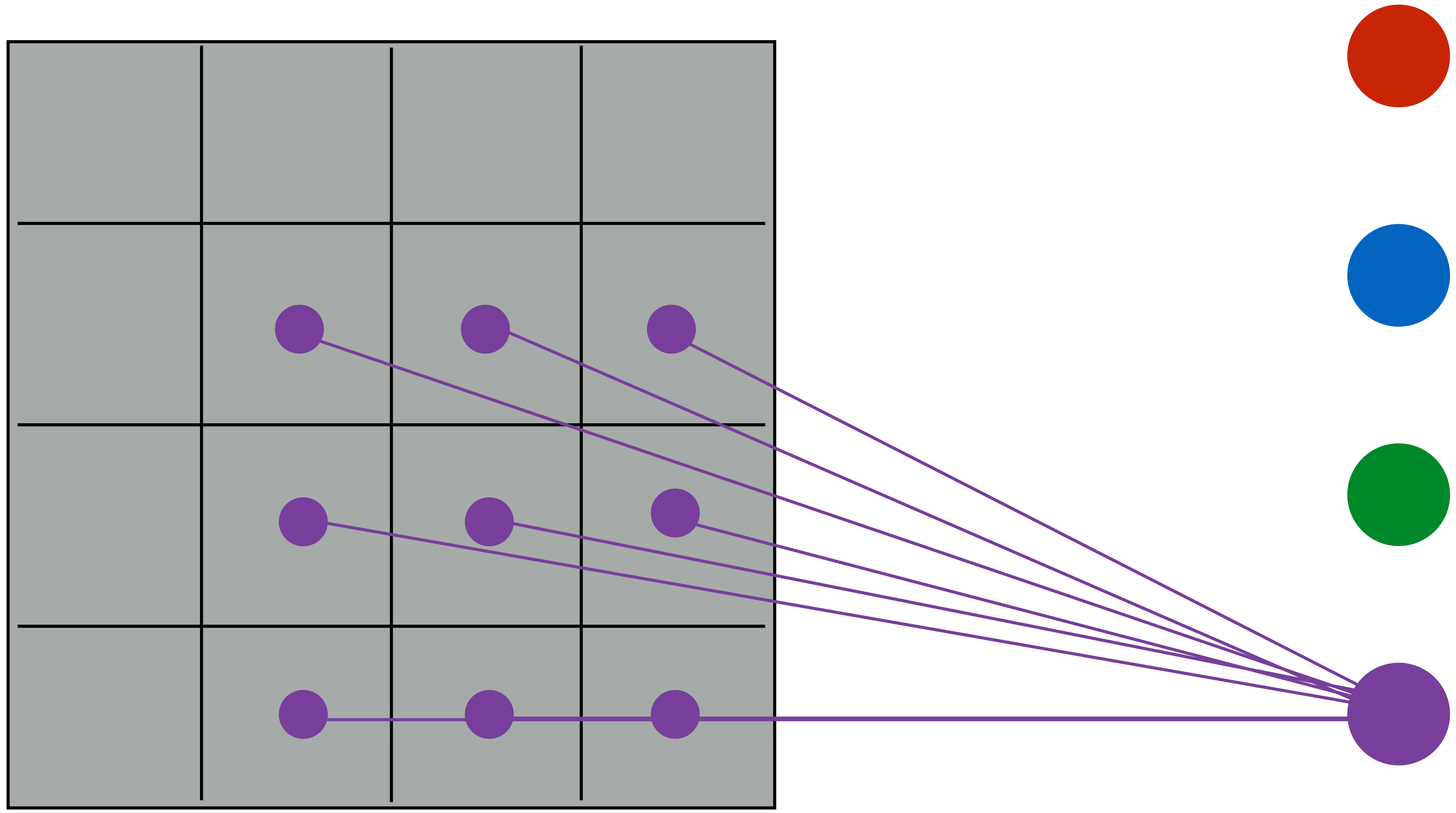


$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{2,i,j}\mathcal{I}(i+1,j)+b_2\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{3,i,j}\mathcal{I}(i,j+1)+b_3\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{4,i,j}\mathcal{I}(i+1,j+1)+b_4\right)$$

Locally Connected Layer



$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

3 x 3 + 1 = 10

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{2,i,j}\mathcal{I}(i+1,j)+b_2\right)$$

3 x 3 + 1 = 10

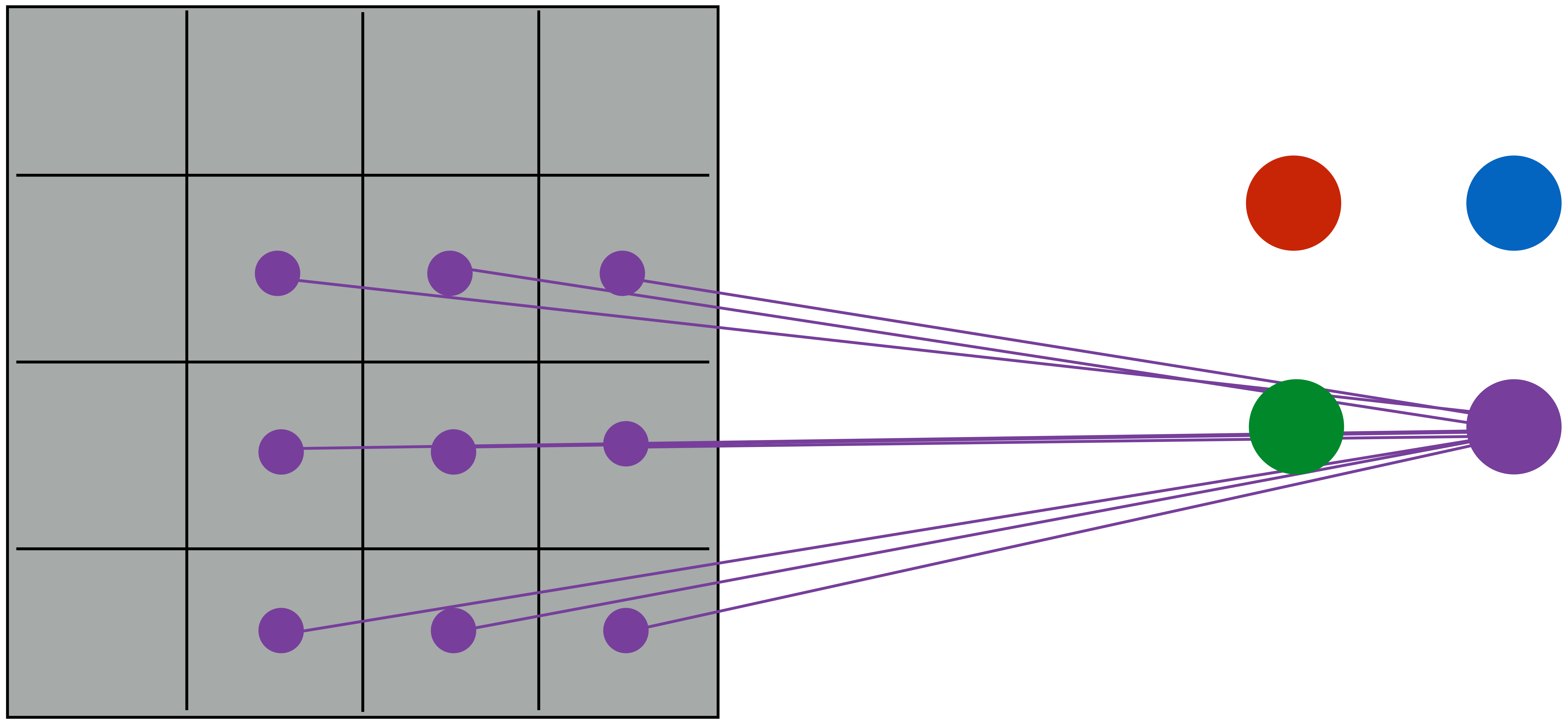
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{3,i,j}\mathcal{I}(i,j+1)+b_3\right)$$

3 x 3 + 1 = 10

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{4,i,j}\mathcal{I}(i+1,j+1)+b_4\right)$$

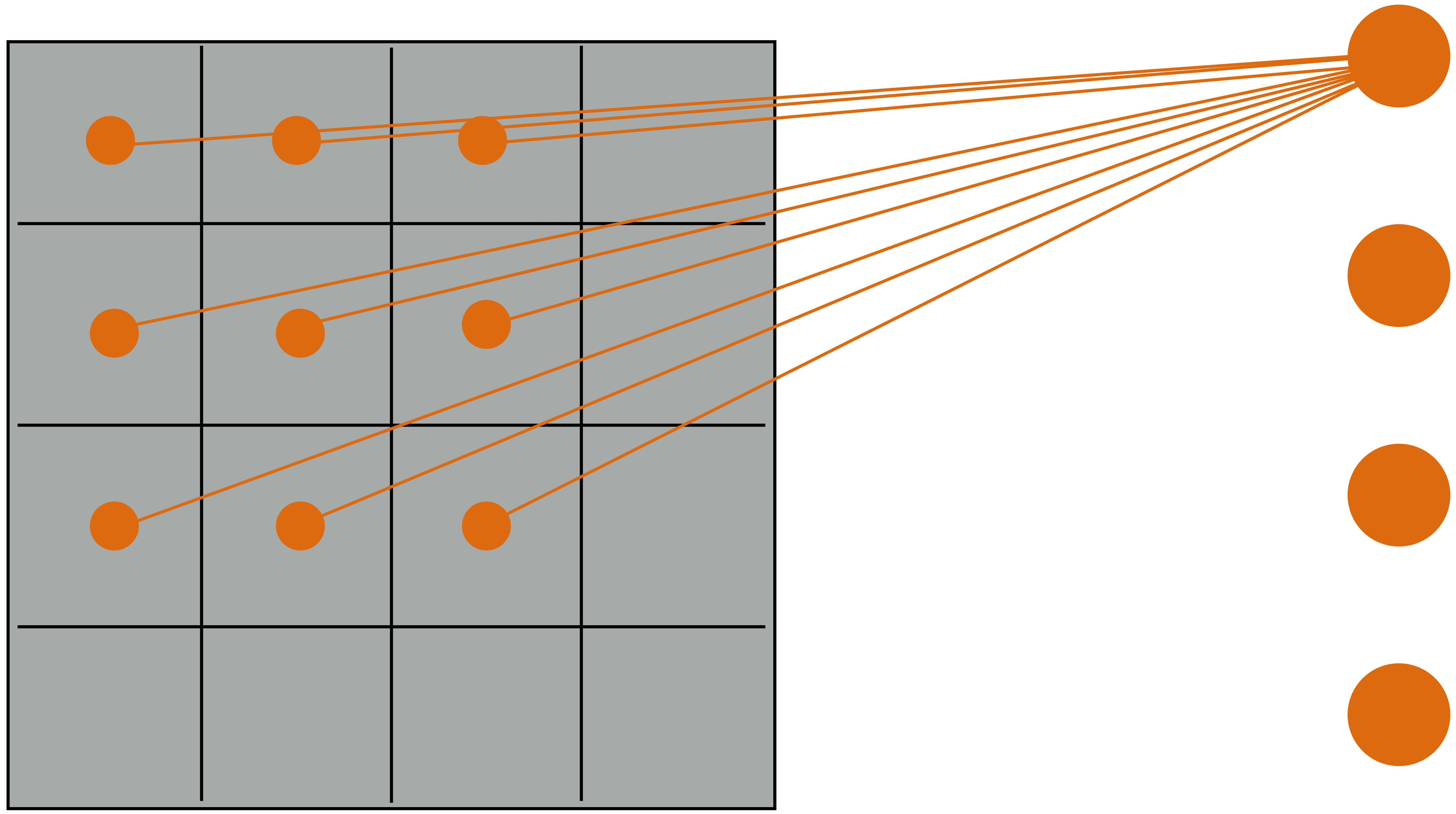
3 x 3 + 1 = 10

Locally Connected Layer



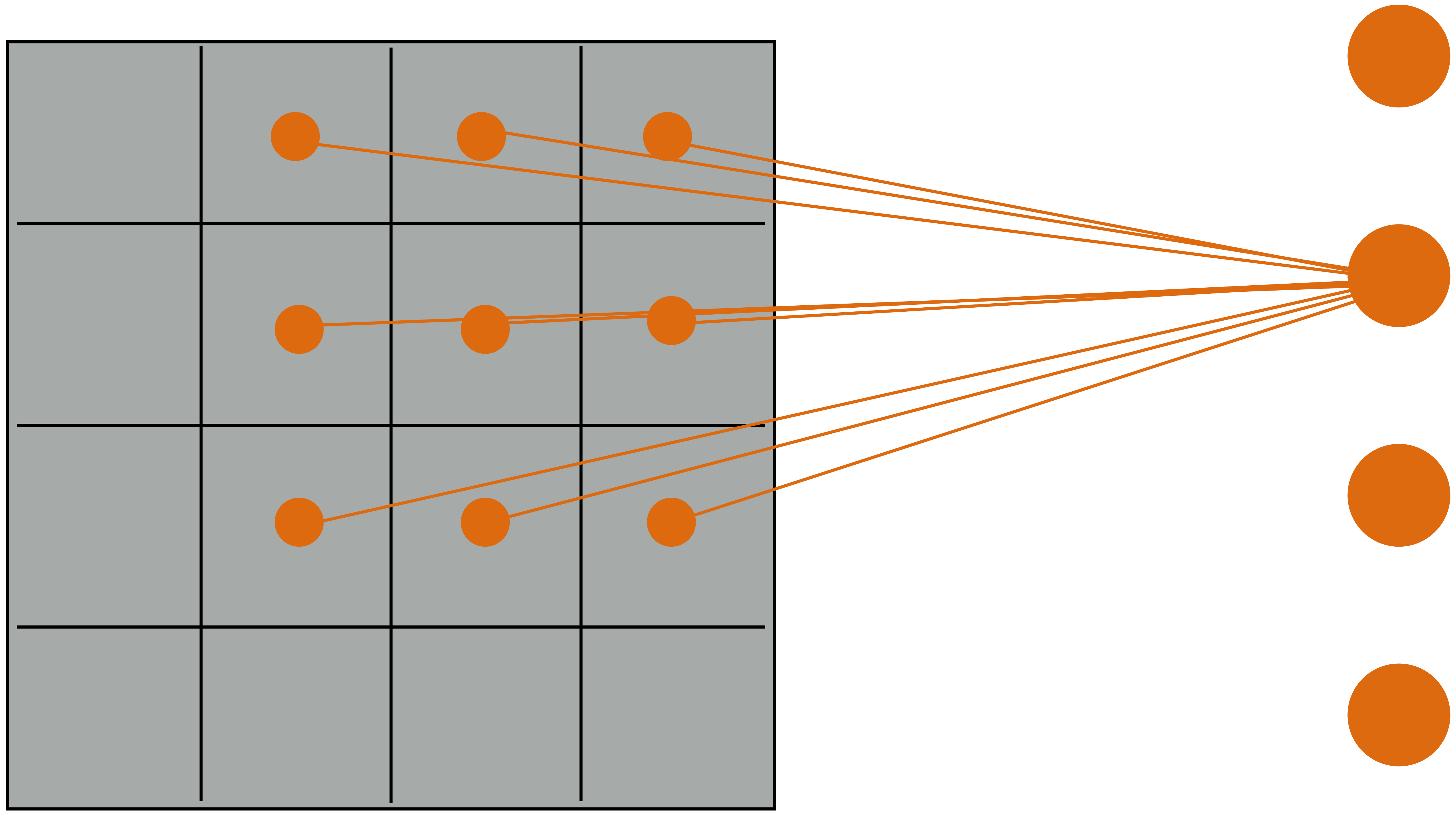
Convolutional Layer

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$



Convolutional Layer

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$



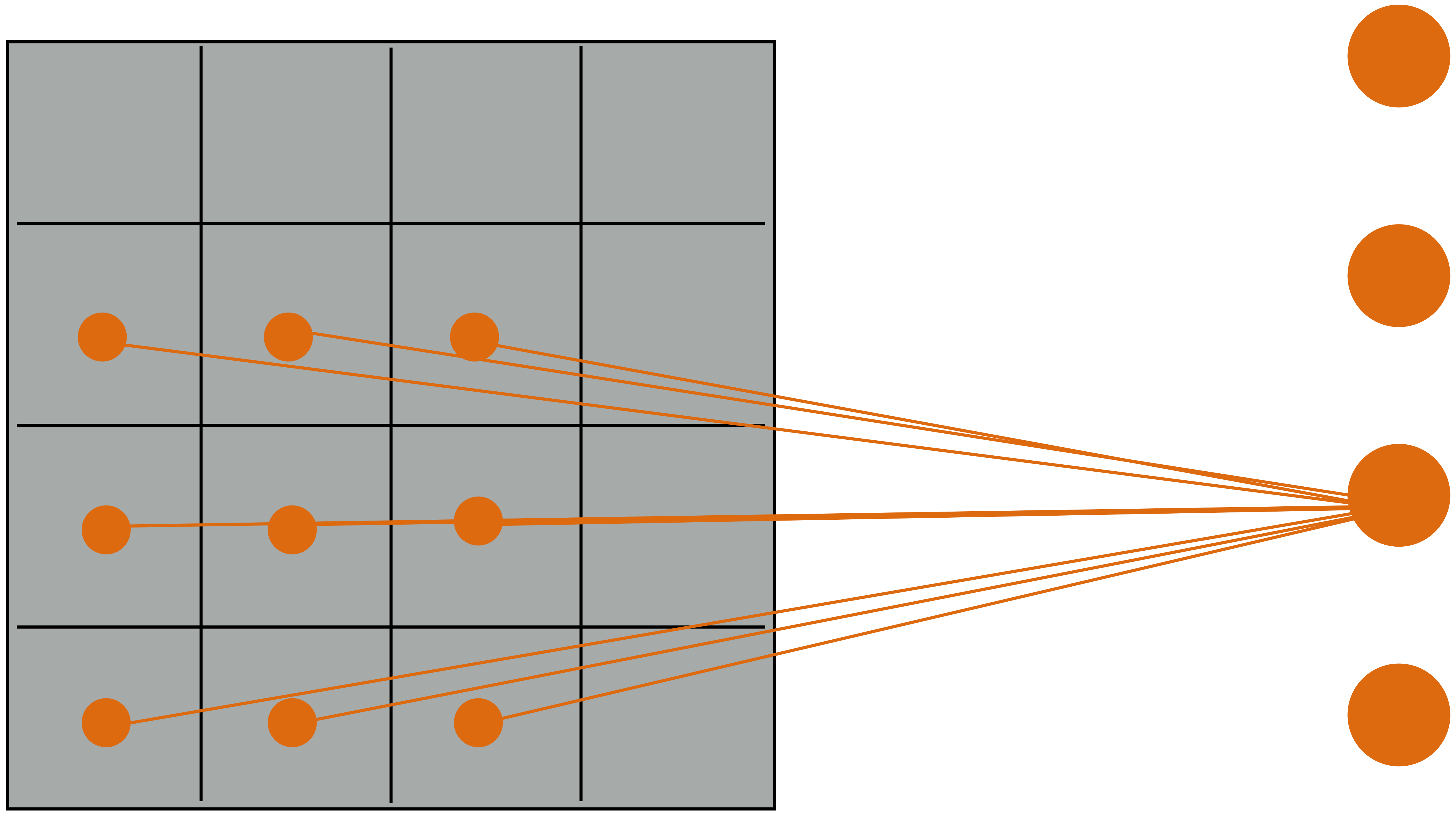
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j)+b\right)$$

Convolutional Layer

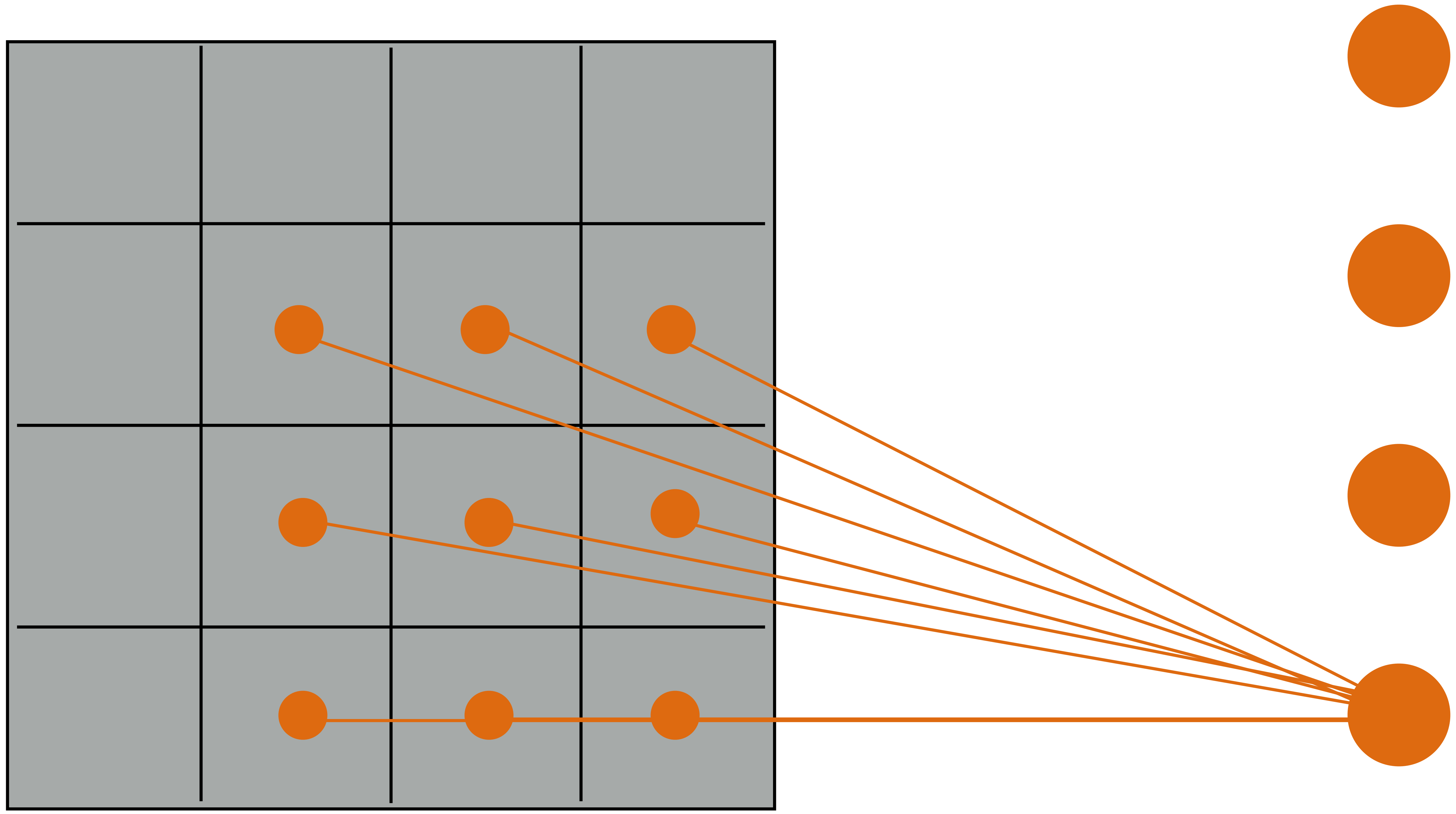
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j+1)+b\right)$$



Convolutional Layer



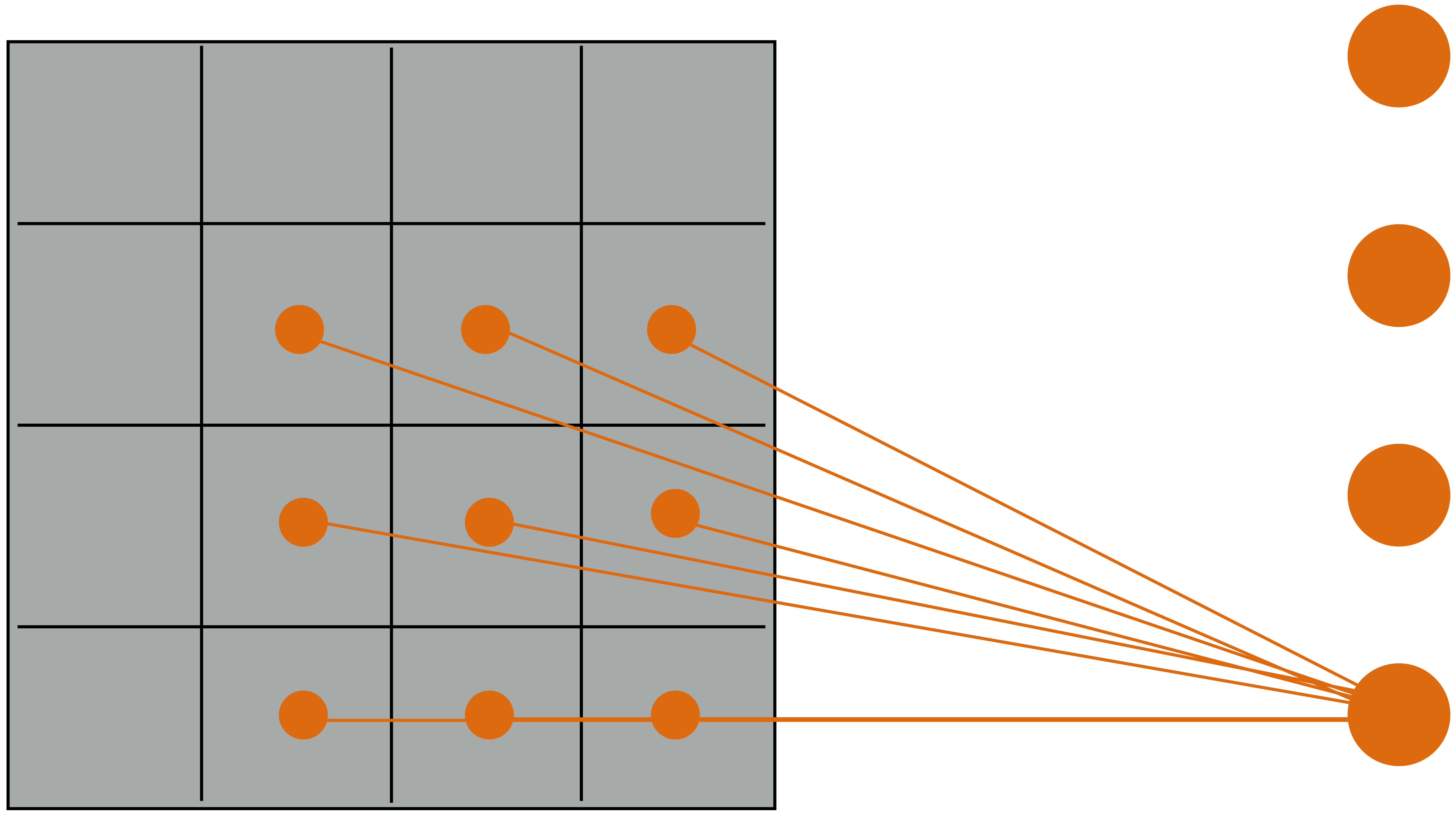
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j+1)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j+1)+b\right)$$

Convolutional Layer



$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$

3 x 3 + 1 = 10

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j)+b\right)$$

0 x 0 + 0 = 0

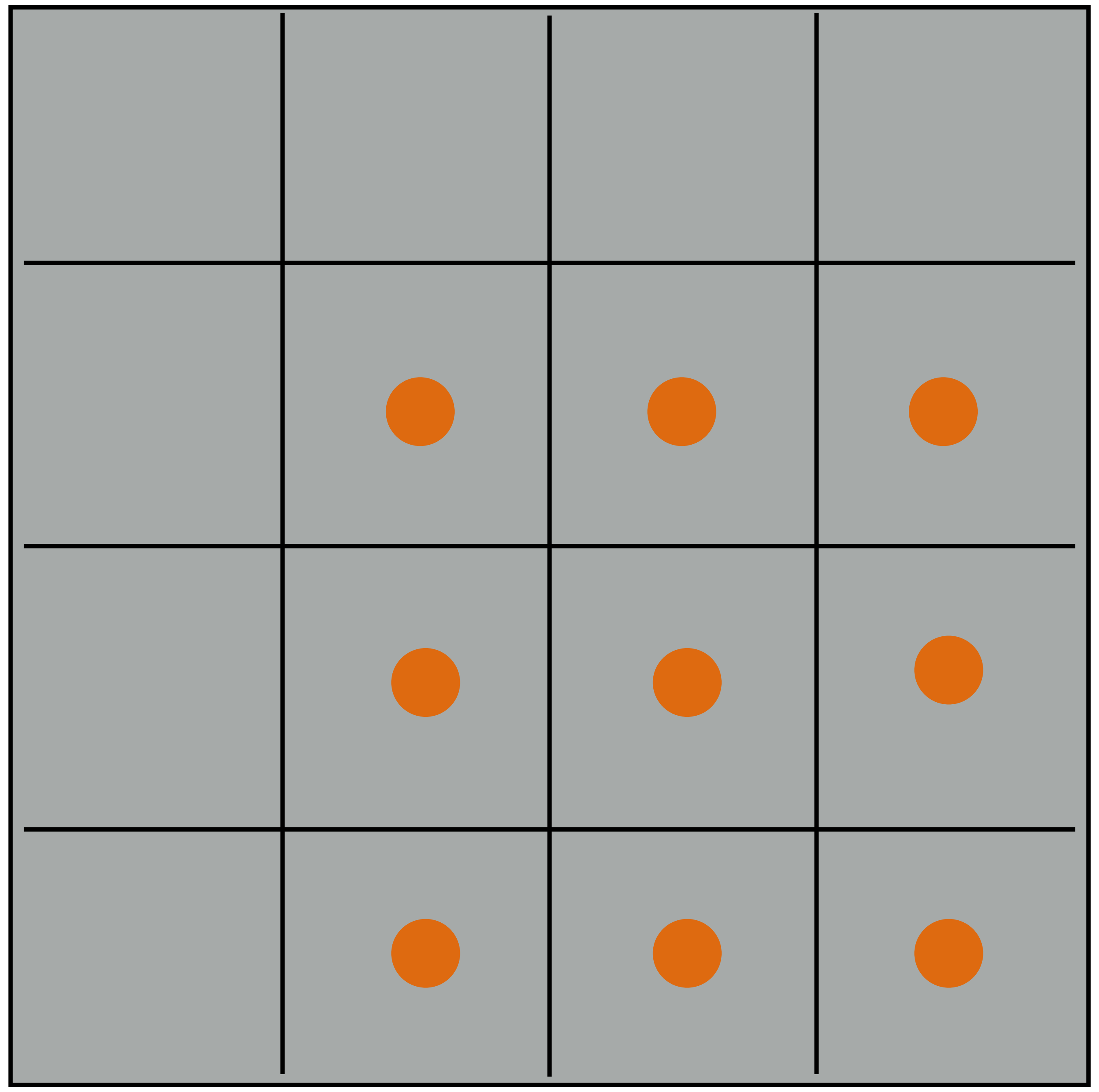
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j+1)+b\right)$$

0 x 0 + 0 = 0

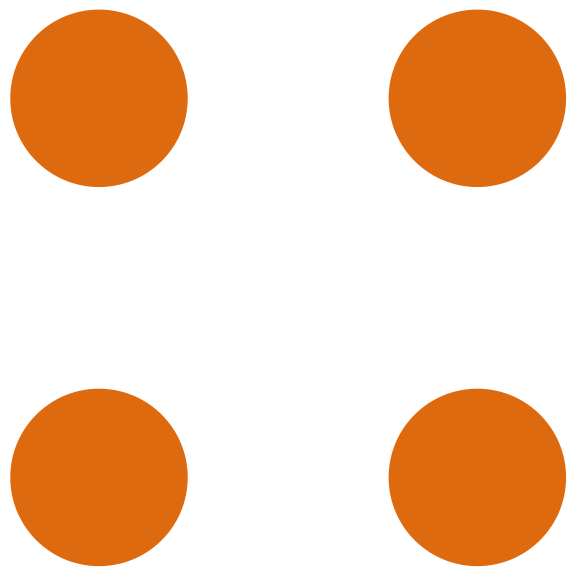
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j+1)+b\right)$$

0 x 0 + 0 = 0

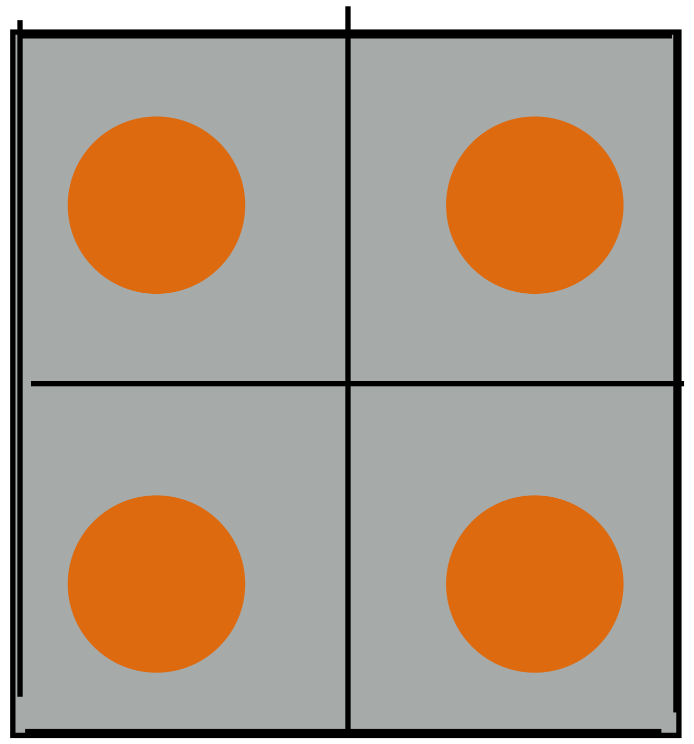
Convolutional Layer: Interpretation #1



Multiple neurons that share weights



neurons



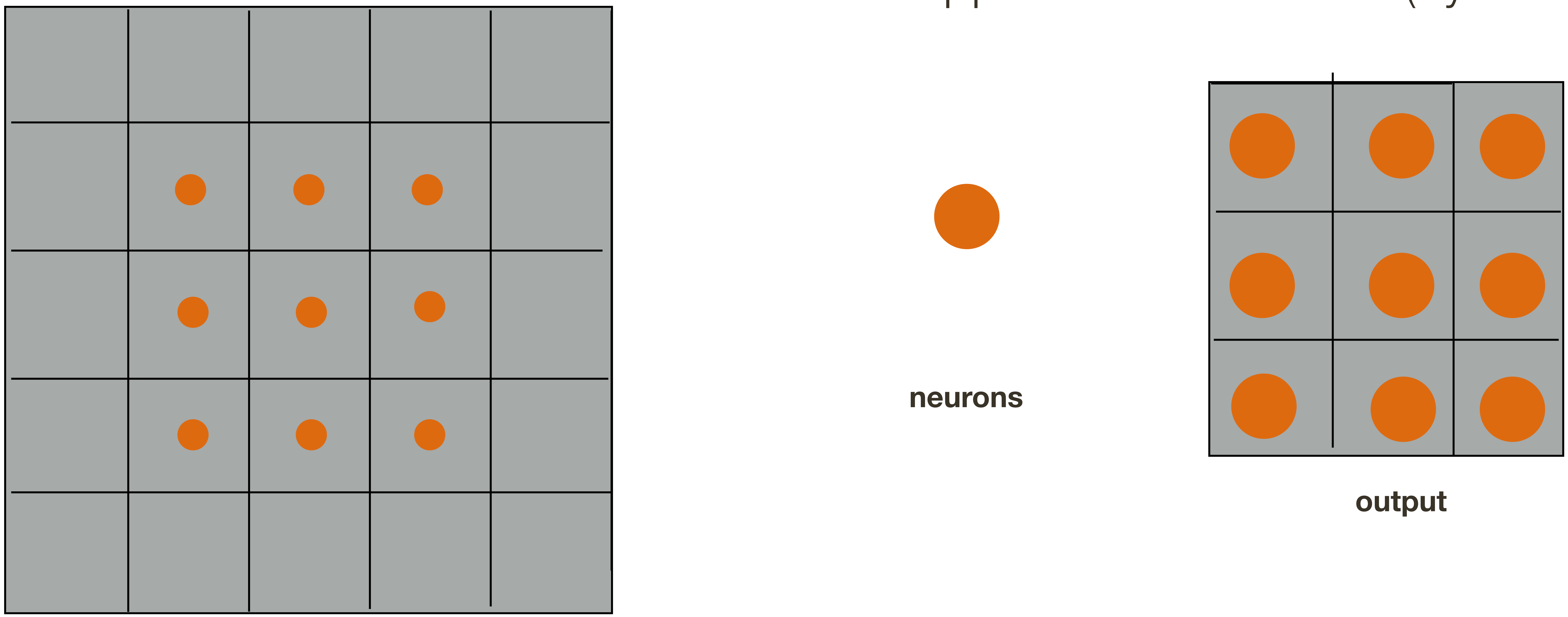
output

Convolutional Layer: Interpretation #2

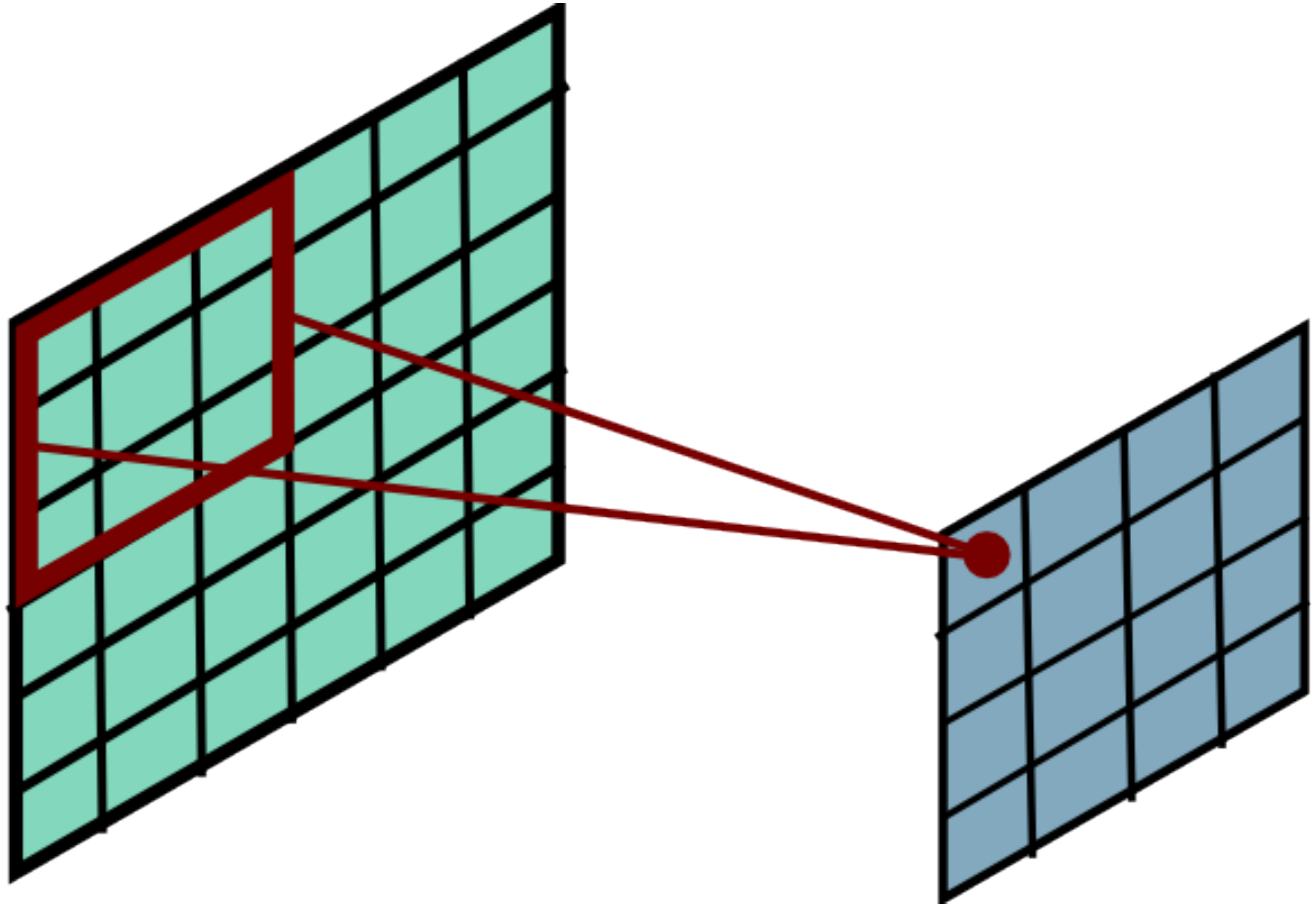
One neuron applied as convolution (by shifting)

Convolutional Layer: Interpretation #2

One neuron applied as convolution (by shifting)

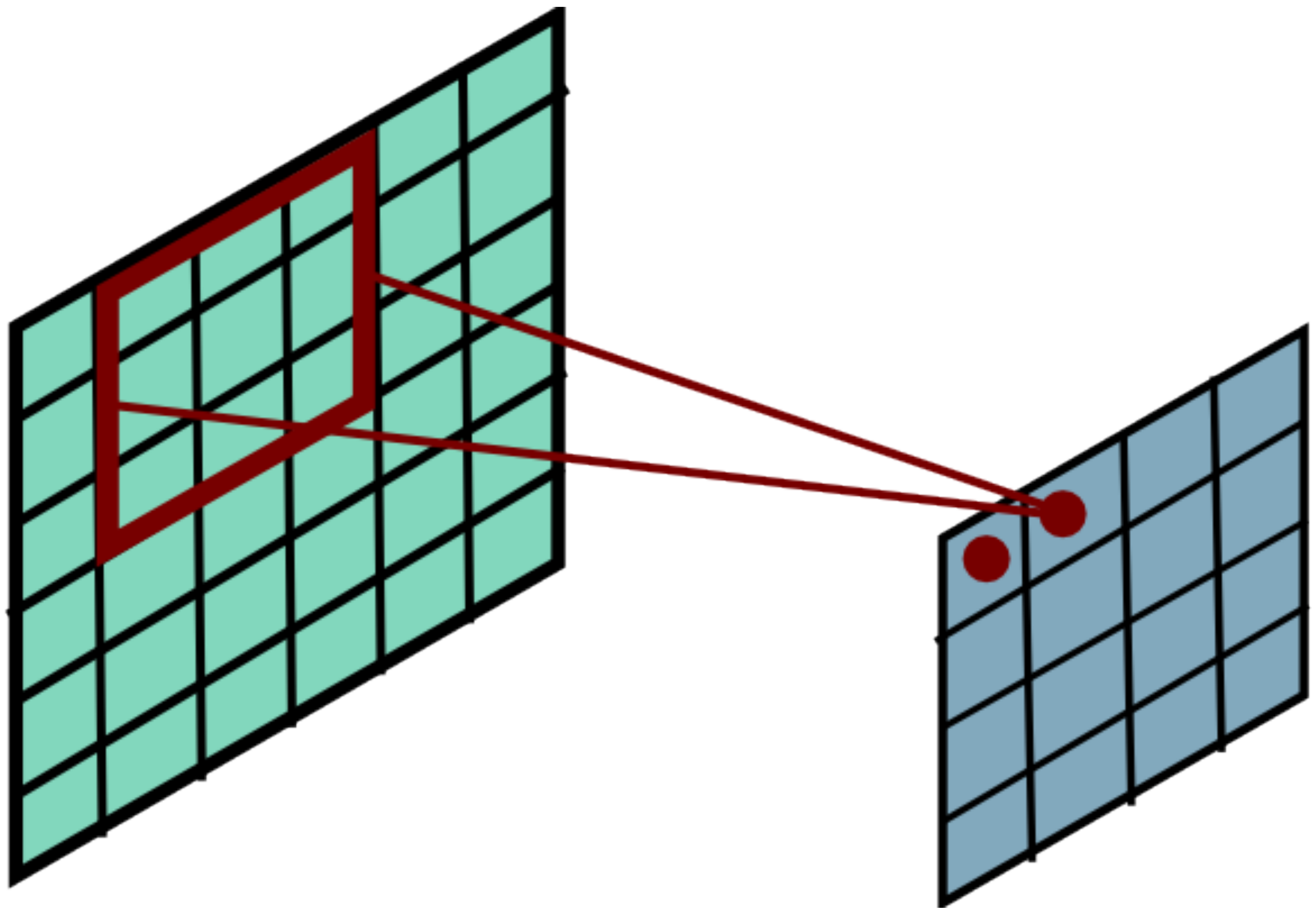


Convolutional Layer



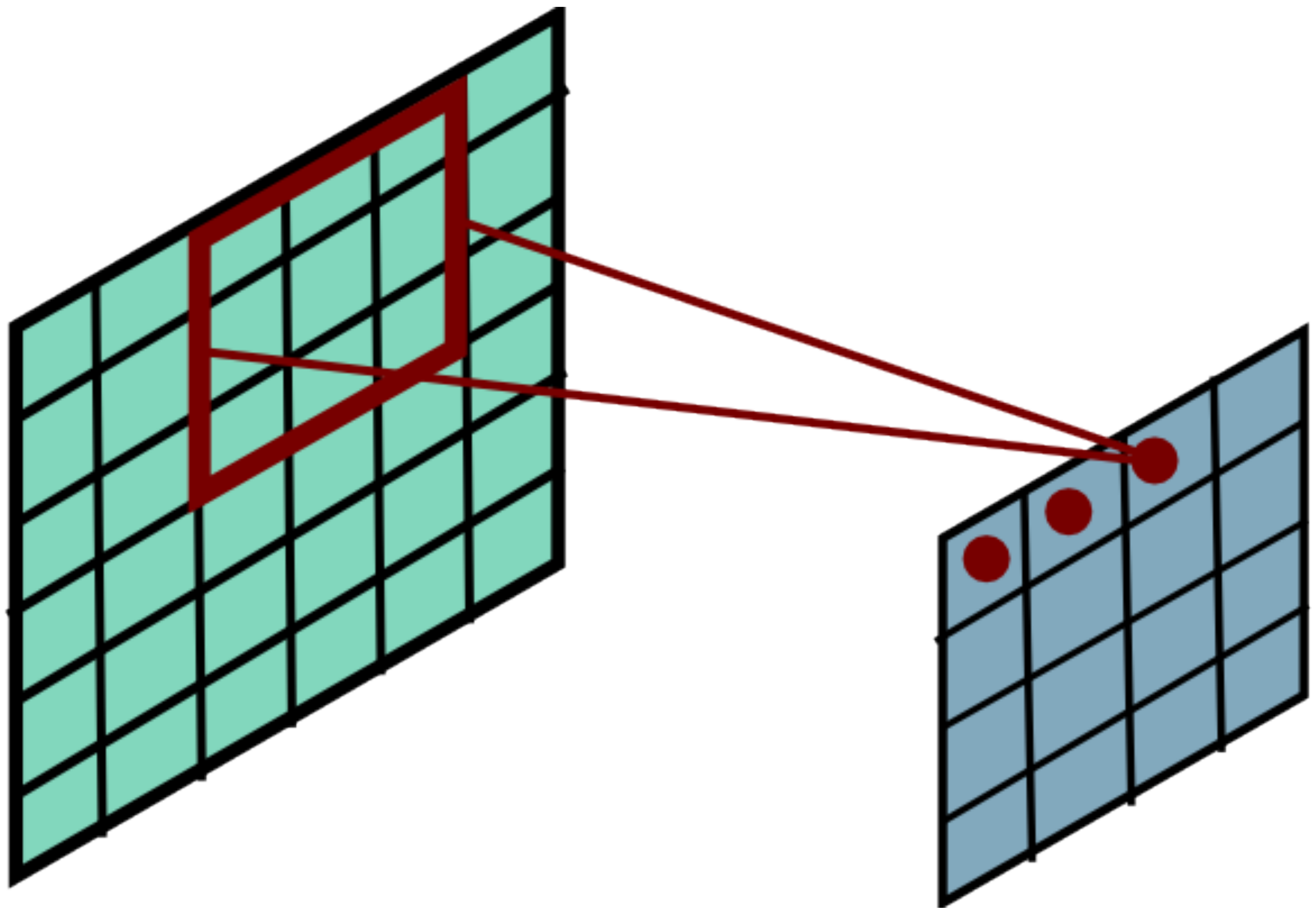
* slide from Marc'Aurelio Renzato

Convolutional Layer



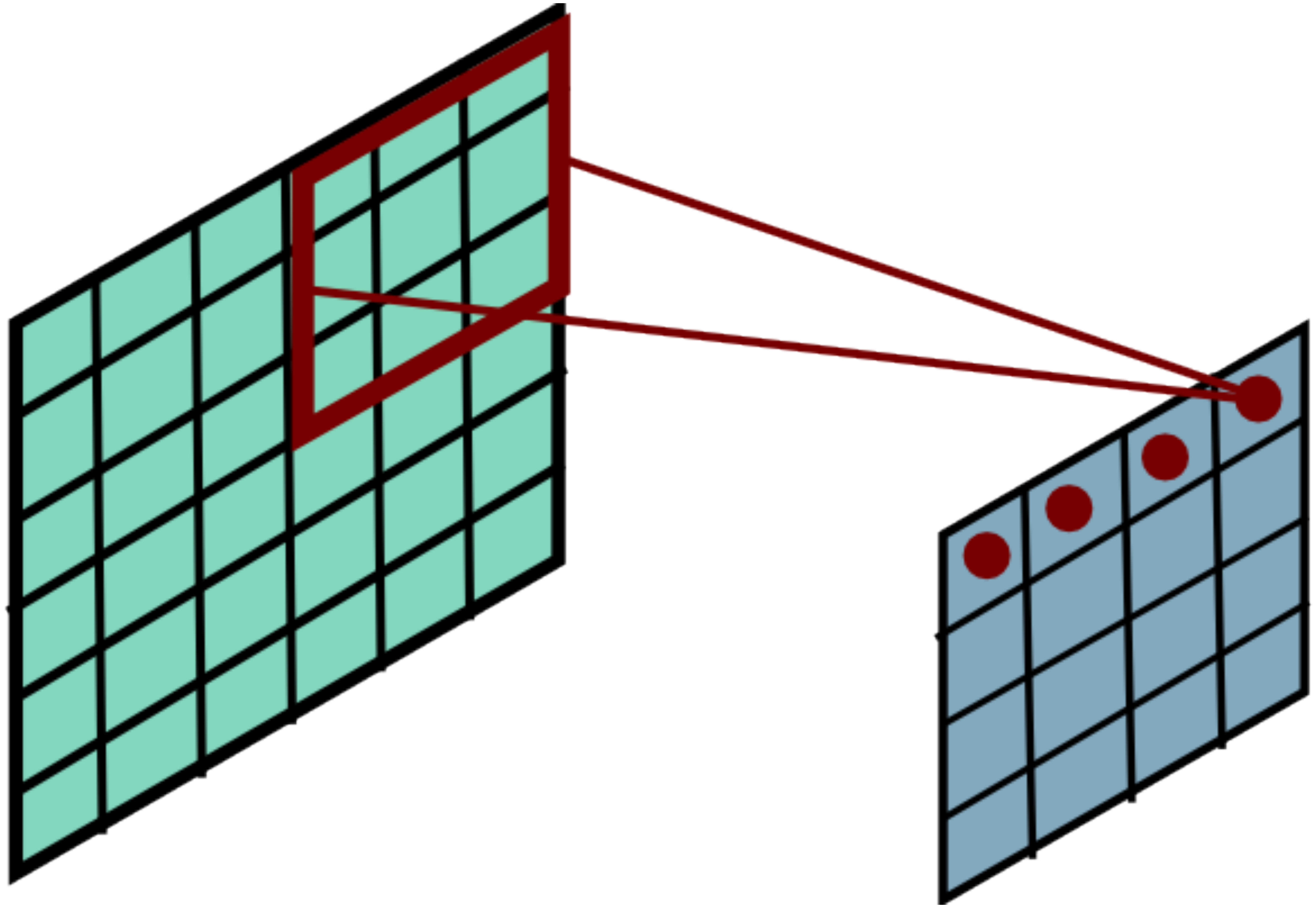
* slide from Marc'Aurelio Renzato

Convolutional Layer



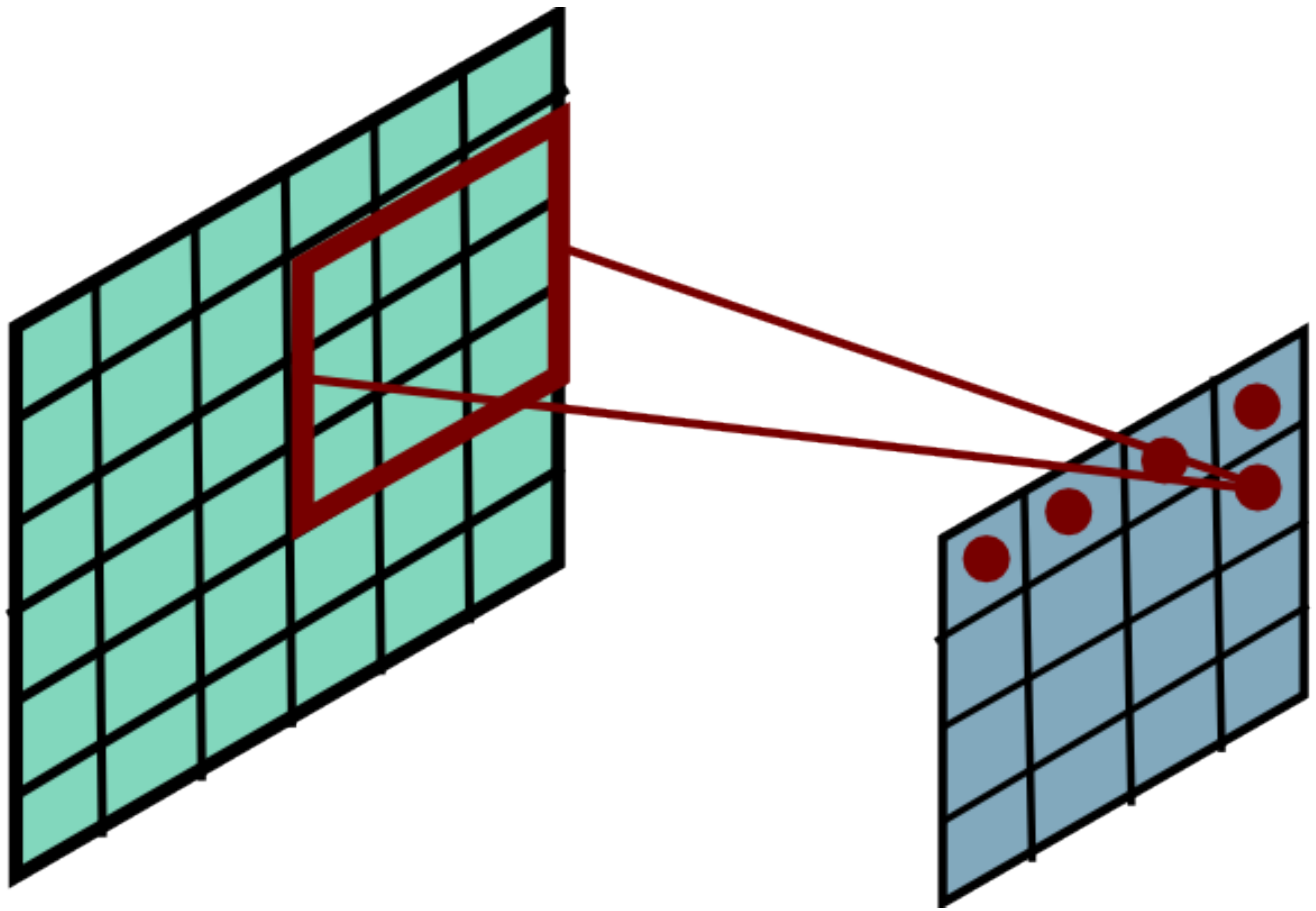
* slide from Marc'Aurelio Renzato

Convolutional Layer



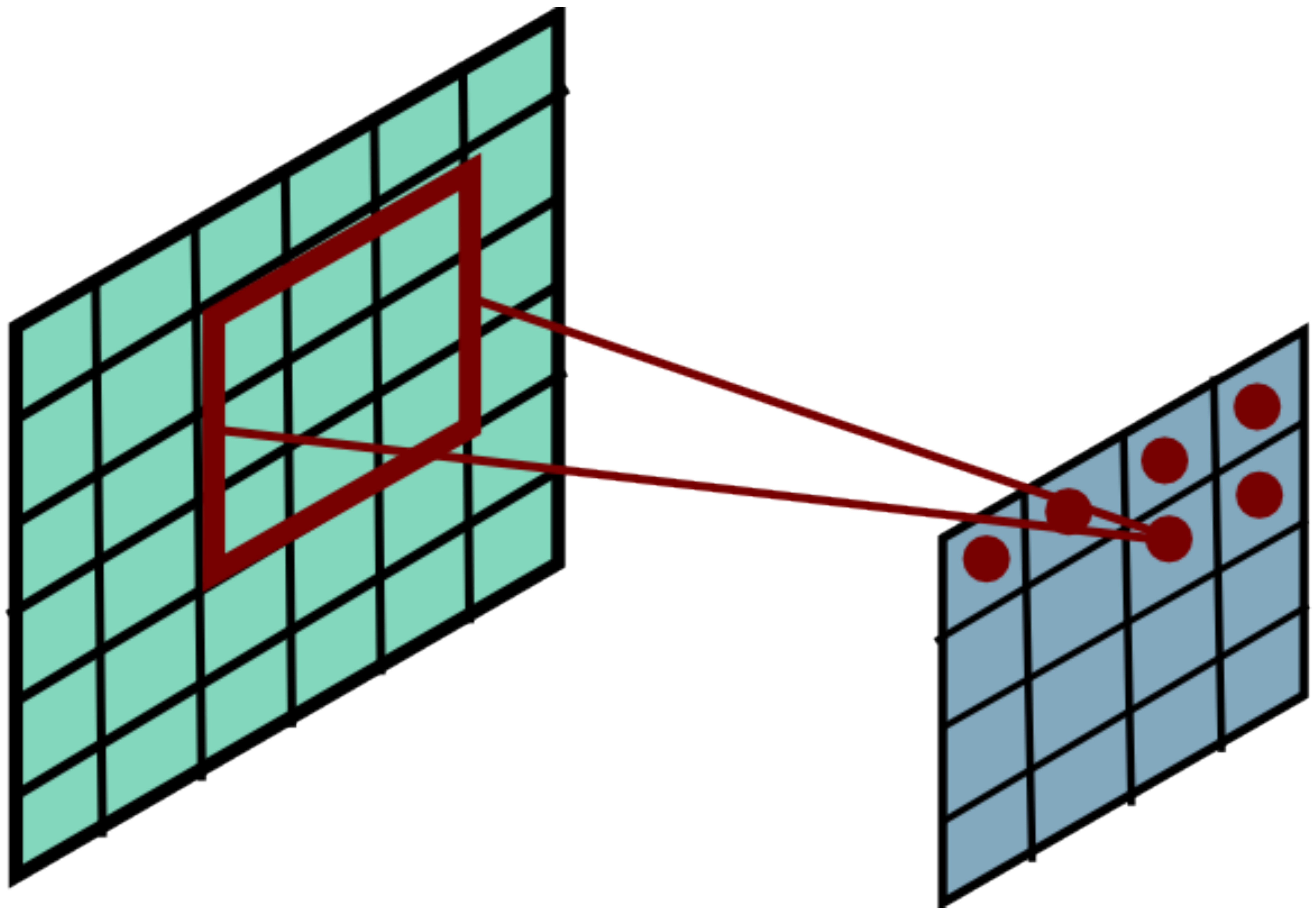
* slide from Marc'Aurelio Renzato

Convolutional Layer



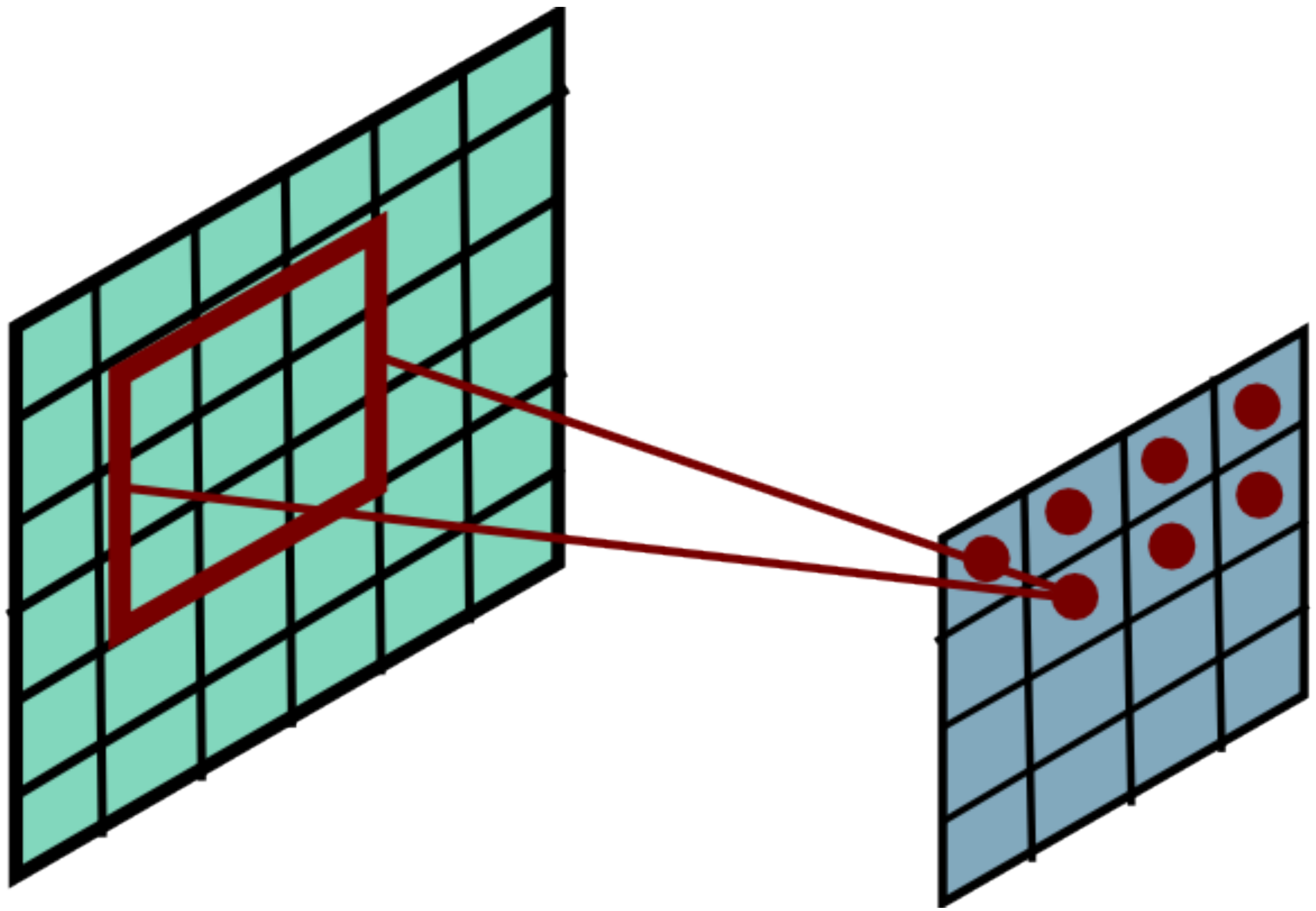
* slide from Marc'Aurelio Renzato

Convolutional Layer



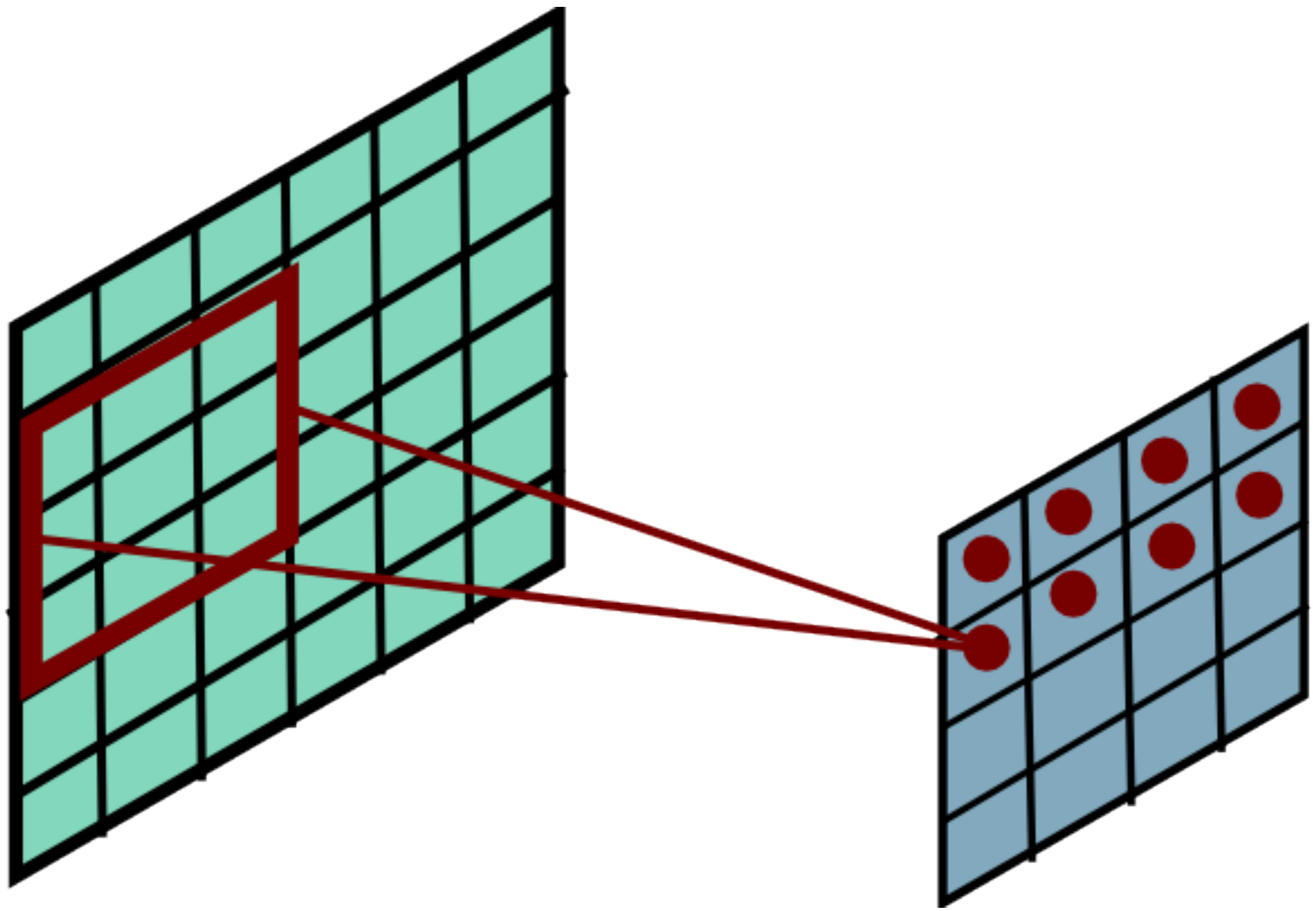
* slide from Marc'Aurelio Renzato

Convolutional Layer



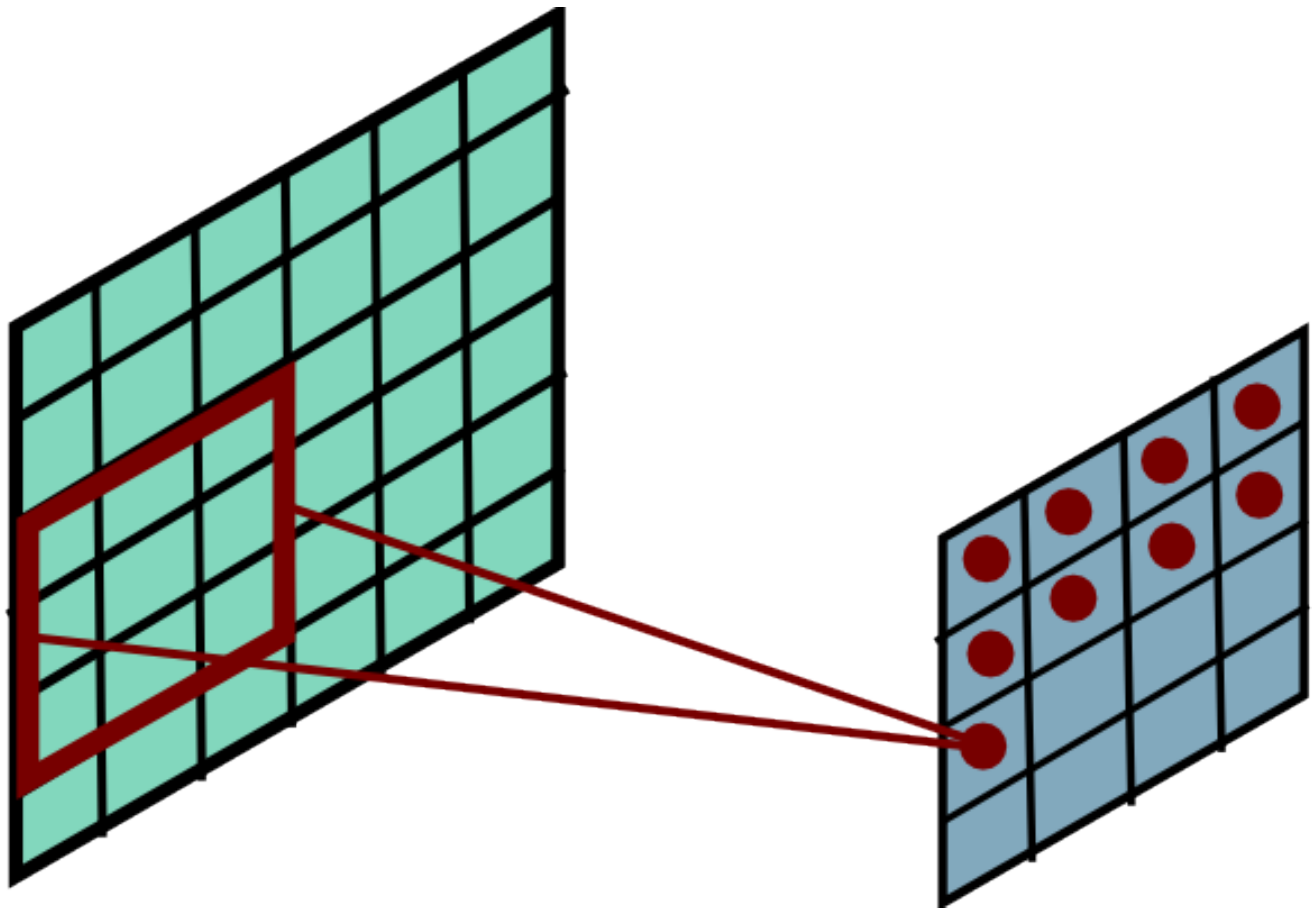
* slide from Marc'Aurelio Renzato

Convolutional Layer



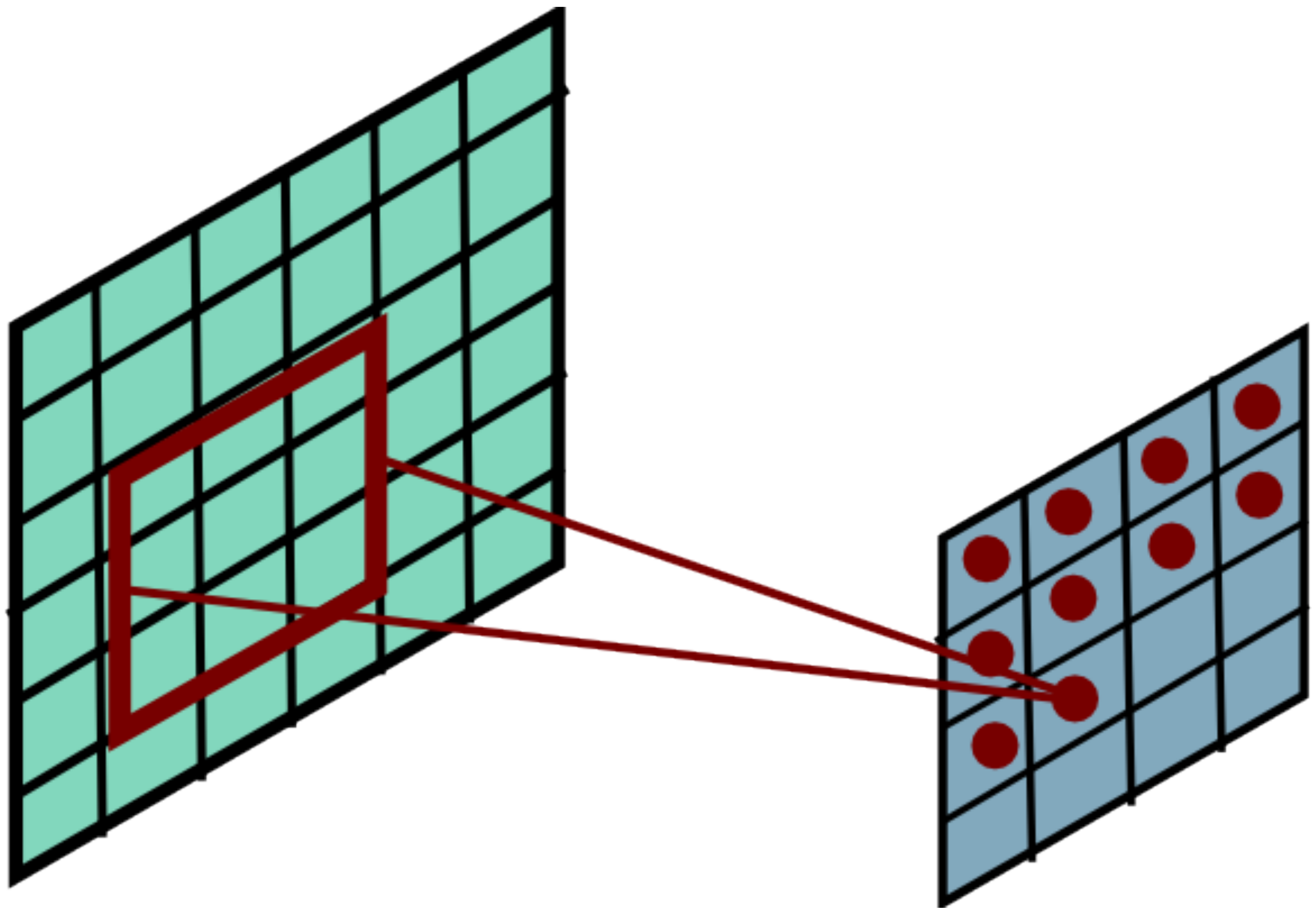
* slide from Marc'Aurelio Renzato

Convolutional Layer



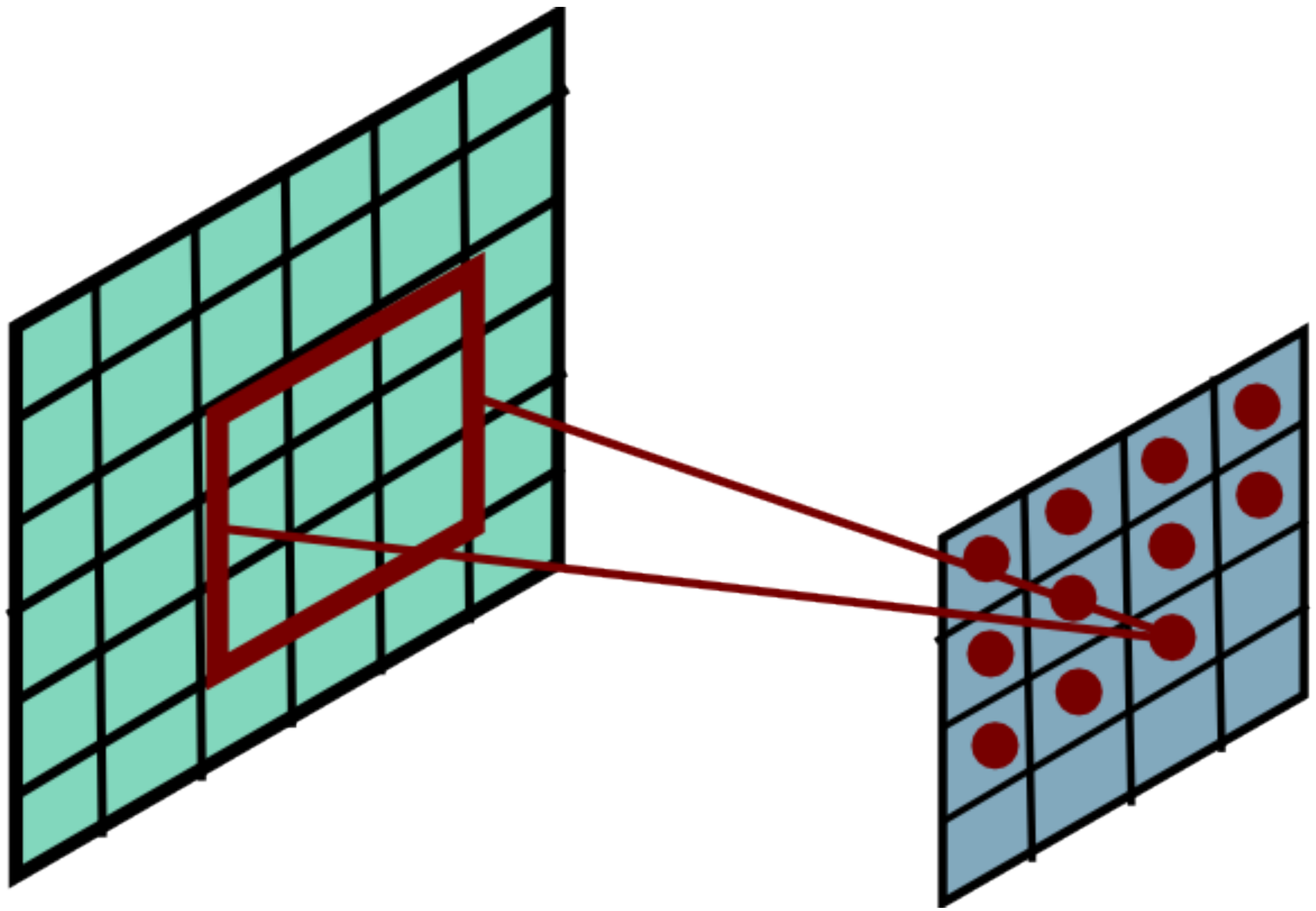
* slide from Marc'Aurelio Renzato

Convolutional Layer



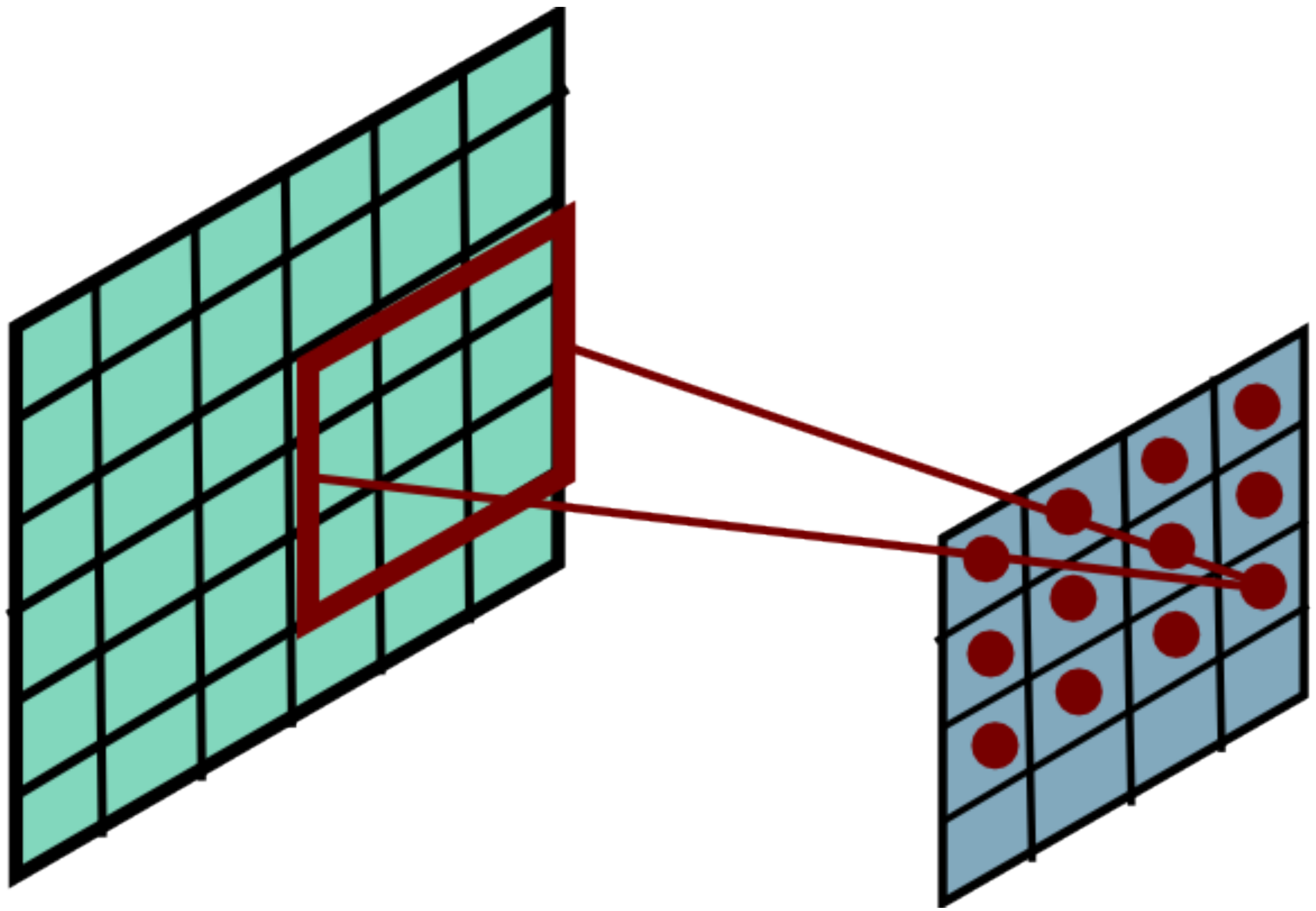
* slide from Marc'Aurelio Renzato

Convolutional Layer



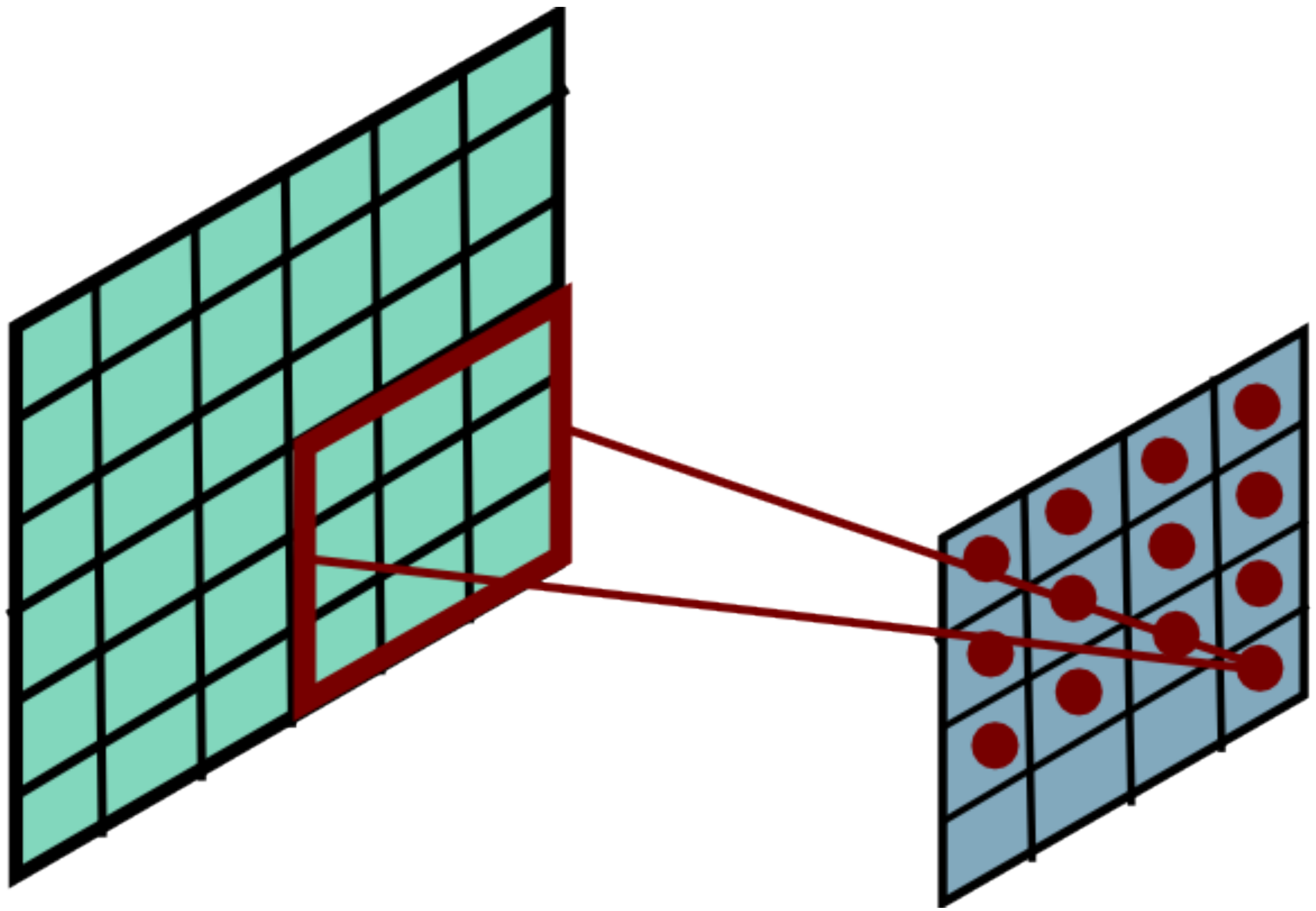
* slide from Marc'Aurelio Renzato

Convolutional Layer



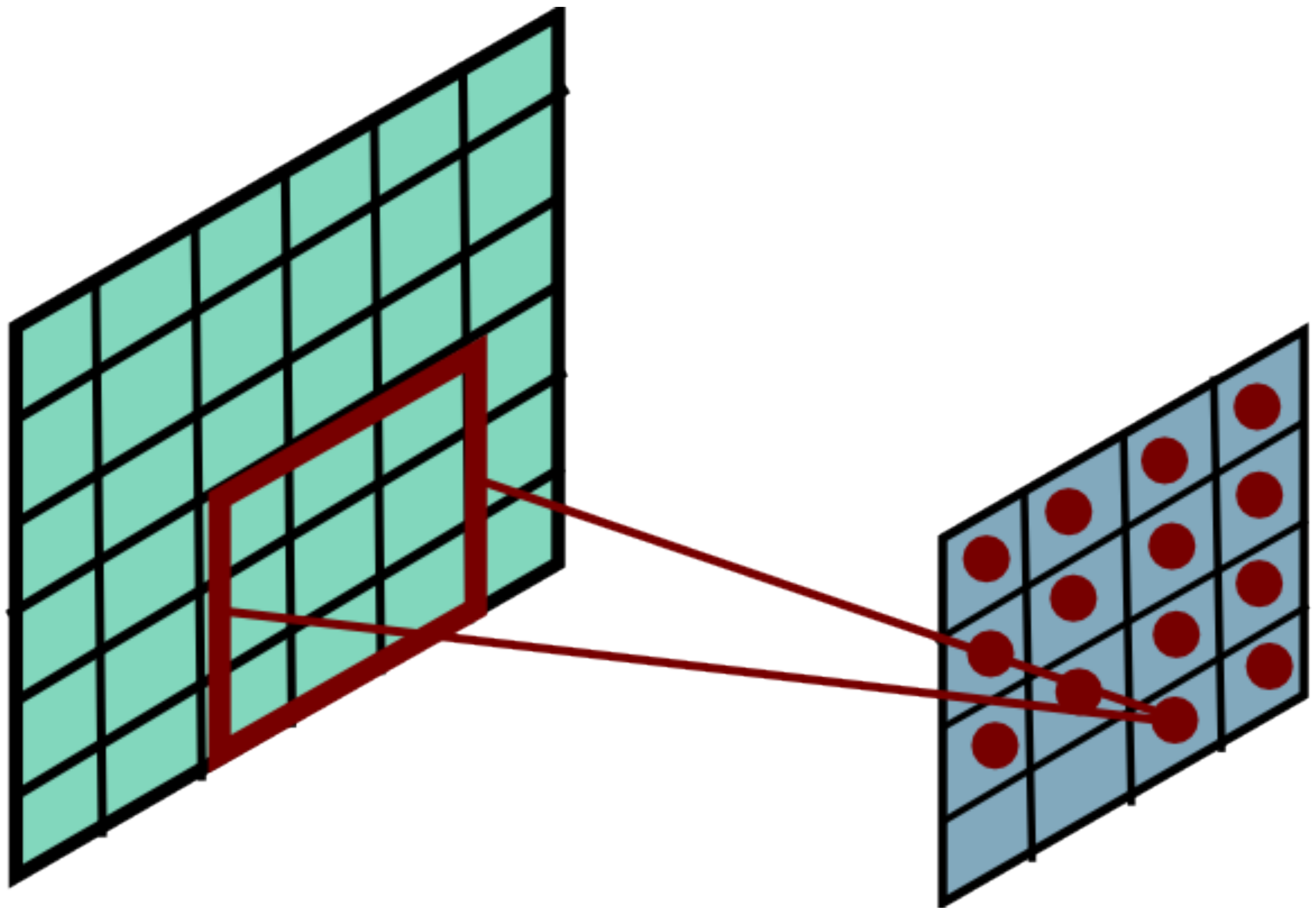
* slide from Marc'Aurelio Renzato

Convolutional Layer



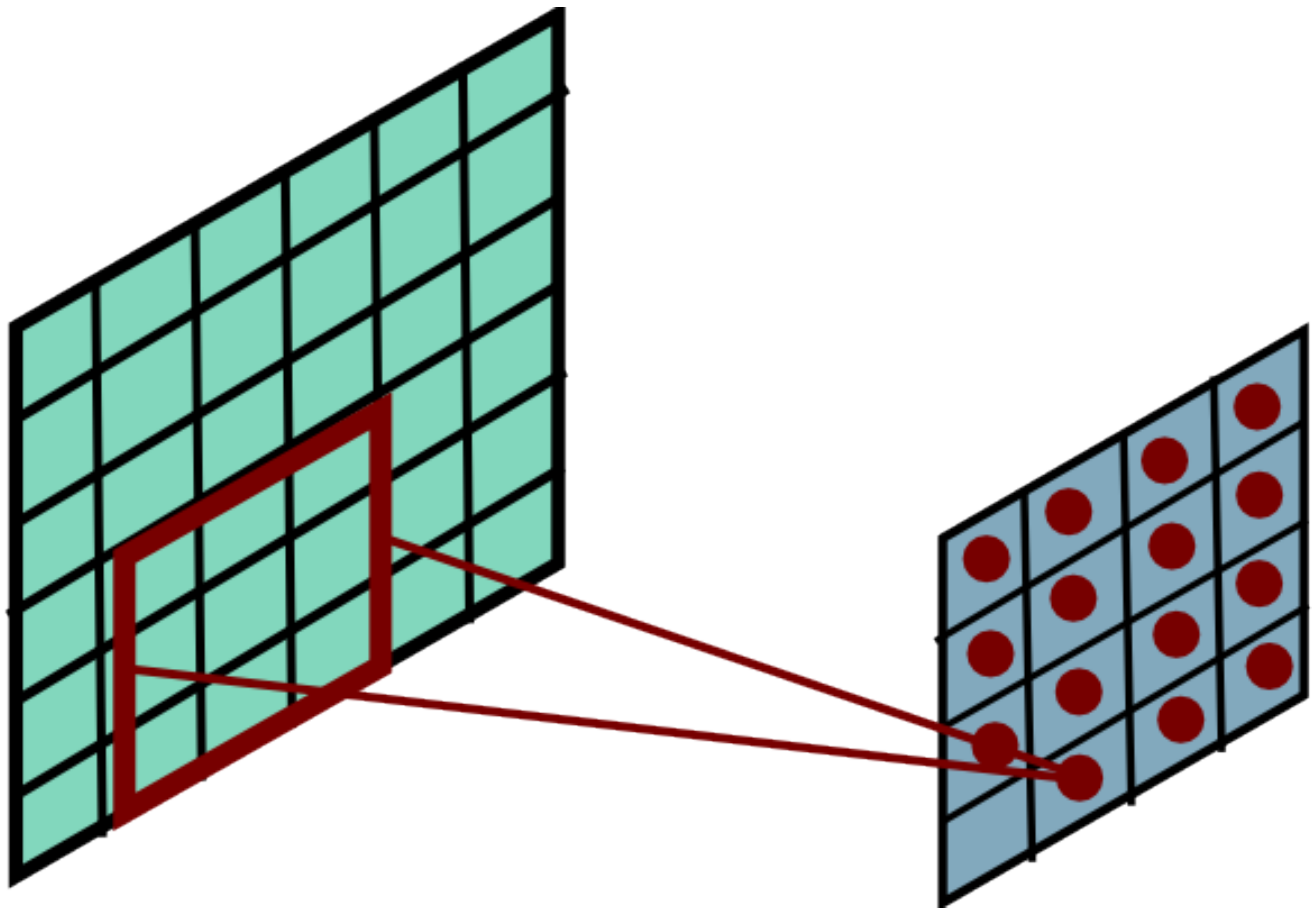
* slide from Marc'Aurelio Renzato

Convolutional Layer



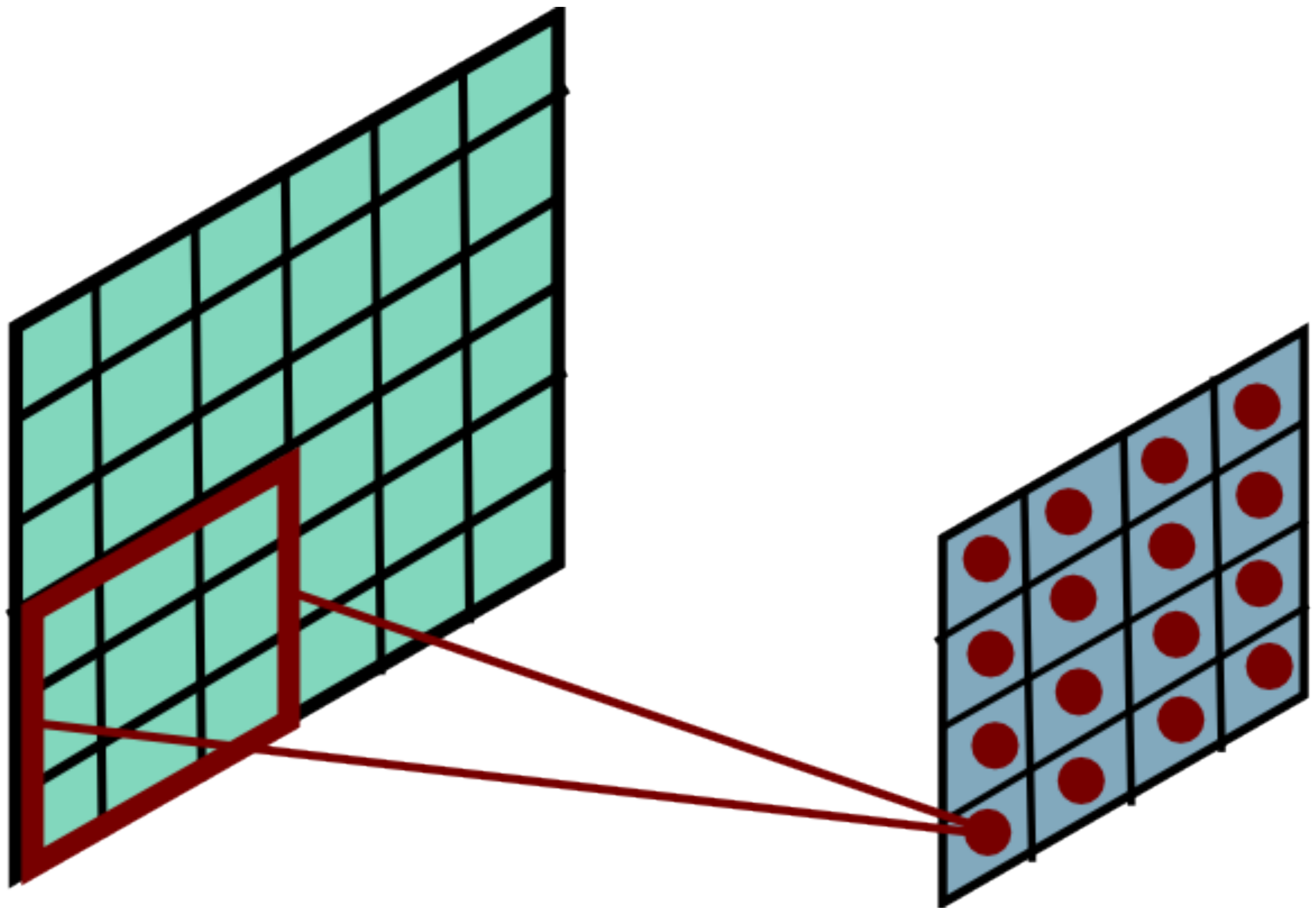
* slide from Marc'Aurelio Renzato

Convolutional Layer



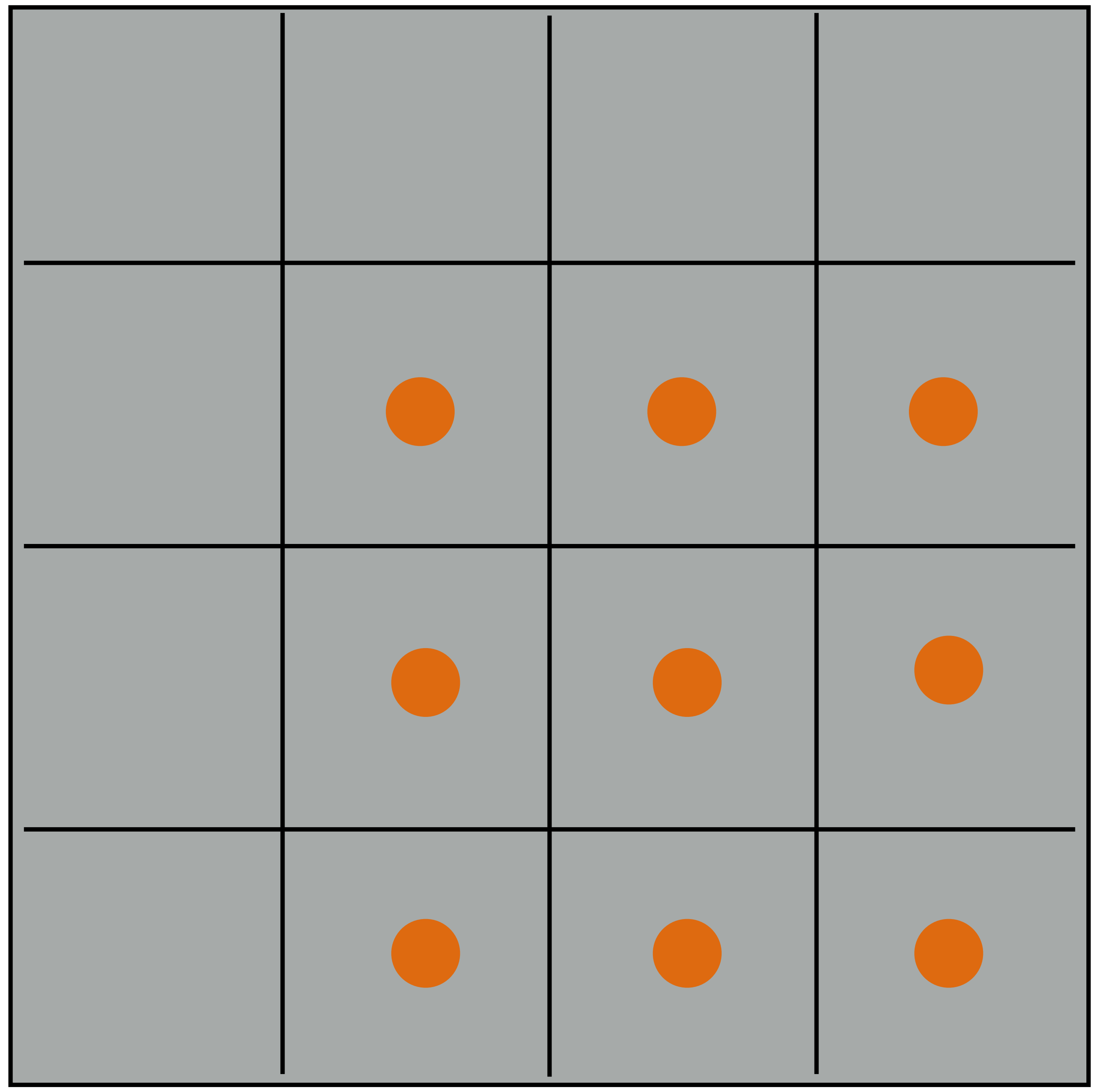
* slide from Marc'Aurelio Renzato

Convolutional Layer

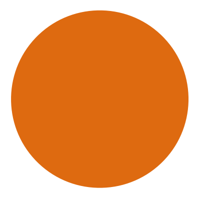


* slide from Marc'Aurelio Renzato

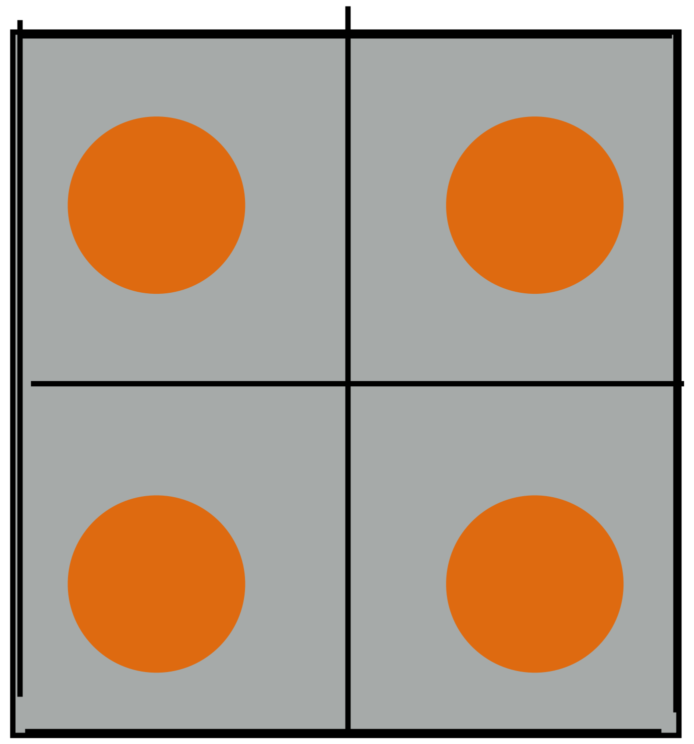
Convolutional Layer: Interpretation #2



One neuron applied as convolution (by shifting)



neurons



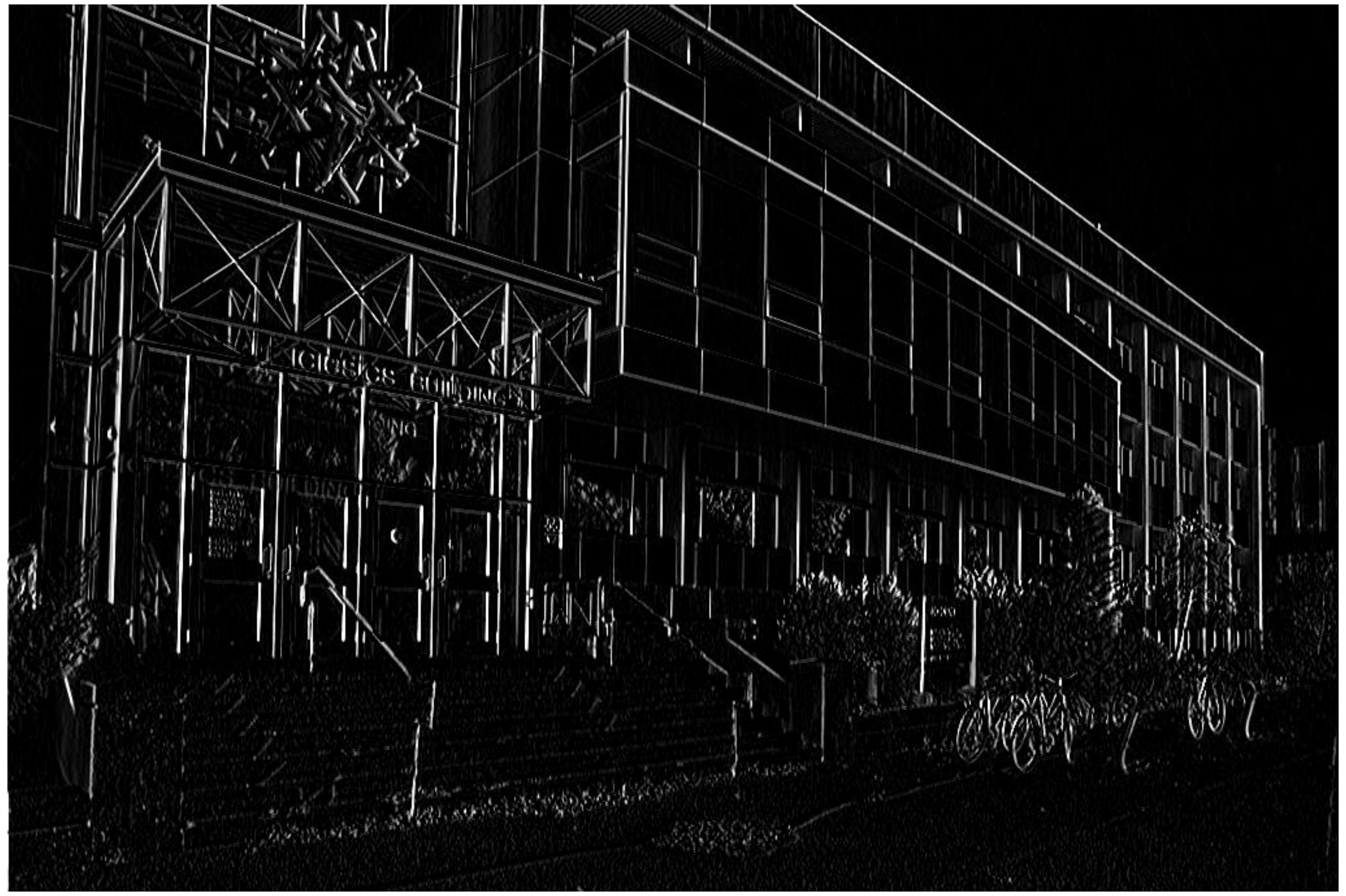
output

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$

Similar to Filter in Convolution / Correlation

Convolution Layer

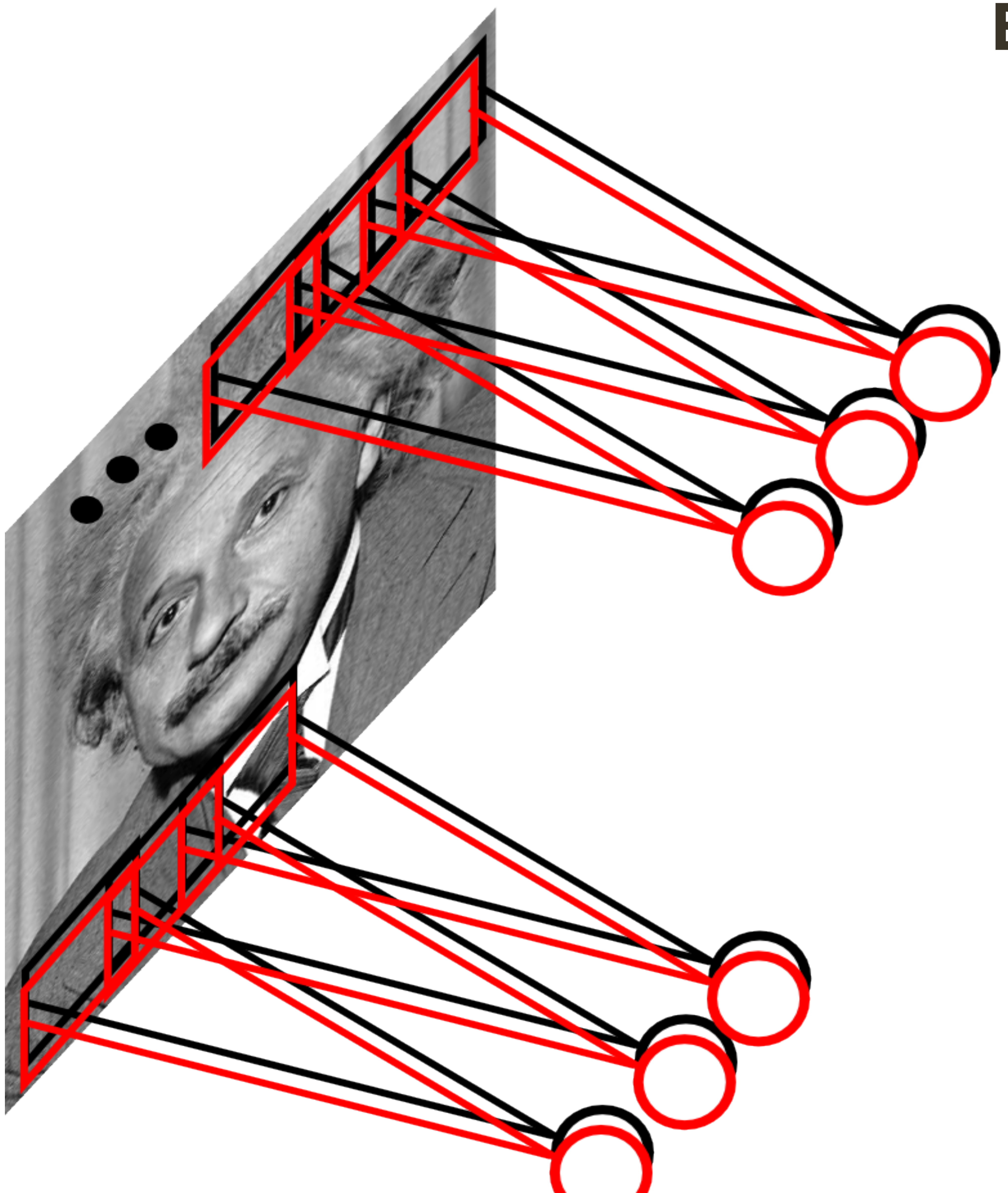
$$\star \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \longrightarrow$$



Convolution Layer

$$\star \begin{bmatrix} 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \end{bmatrix} \rightarrow$$

Convolutional Layer



Example: 200 x 200 image (small)
x 40K hidden units

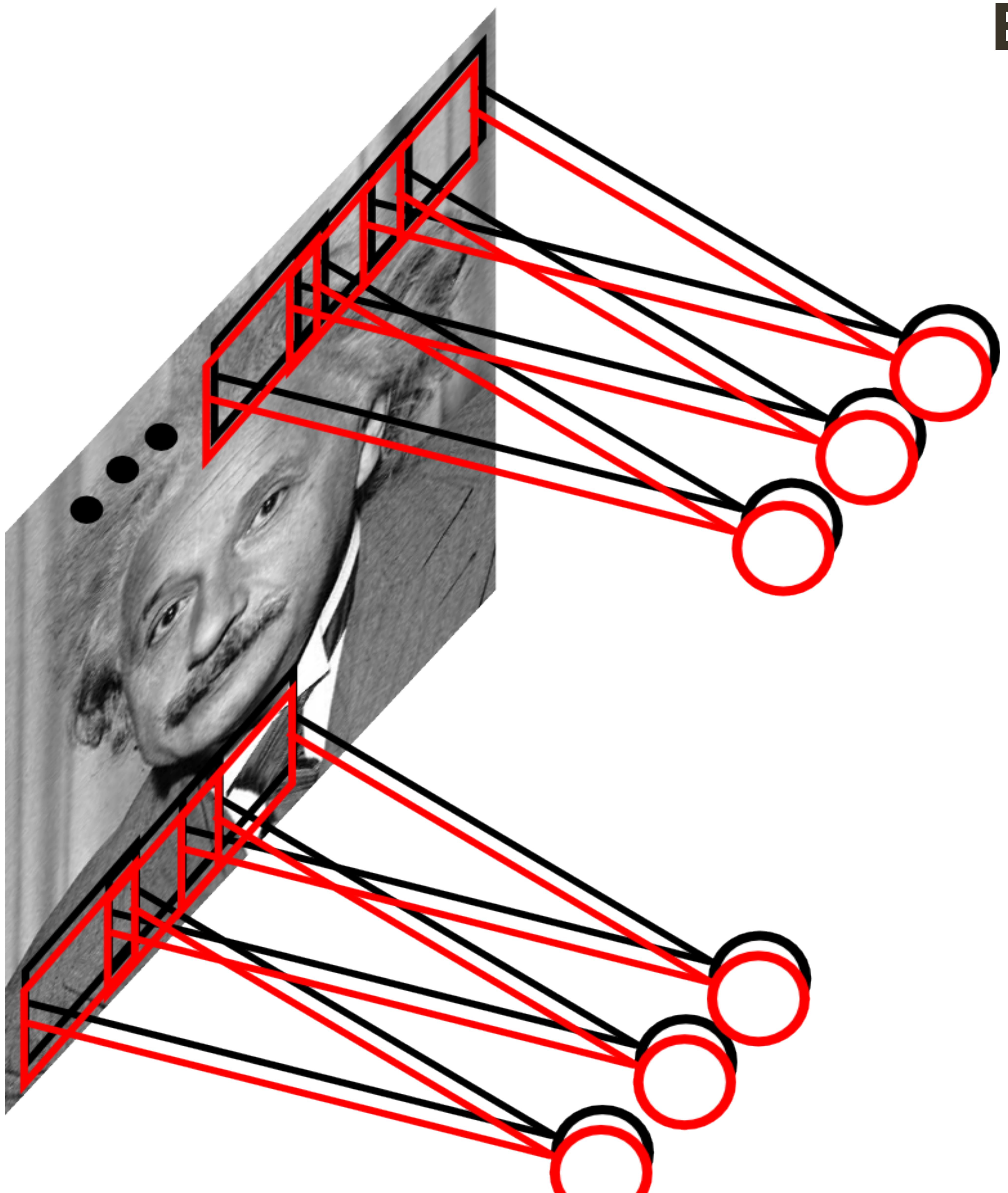
Filter size: 10 x 10

# of filters: 20

Learn **multiple filters**

* slide from Marc'Aurelio Renzato

Convolutional Layer



Example: 200 x 200 image (small)
x 40K hidden units

Filter size: 10 x 10

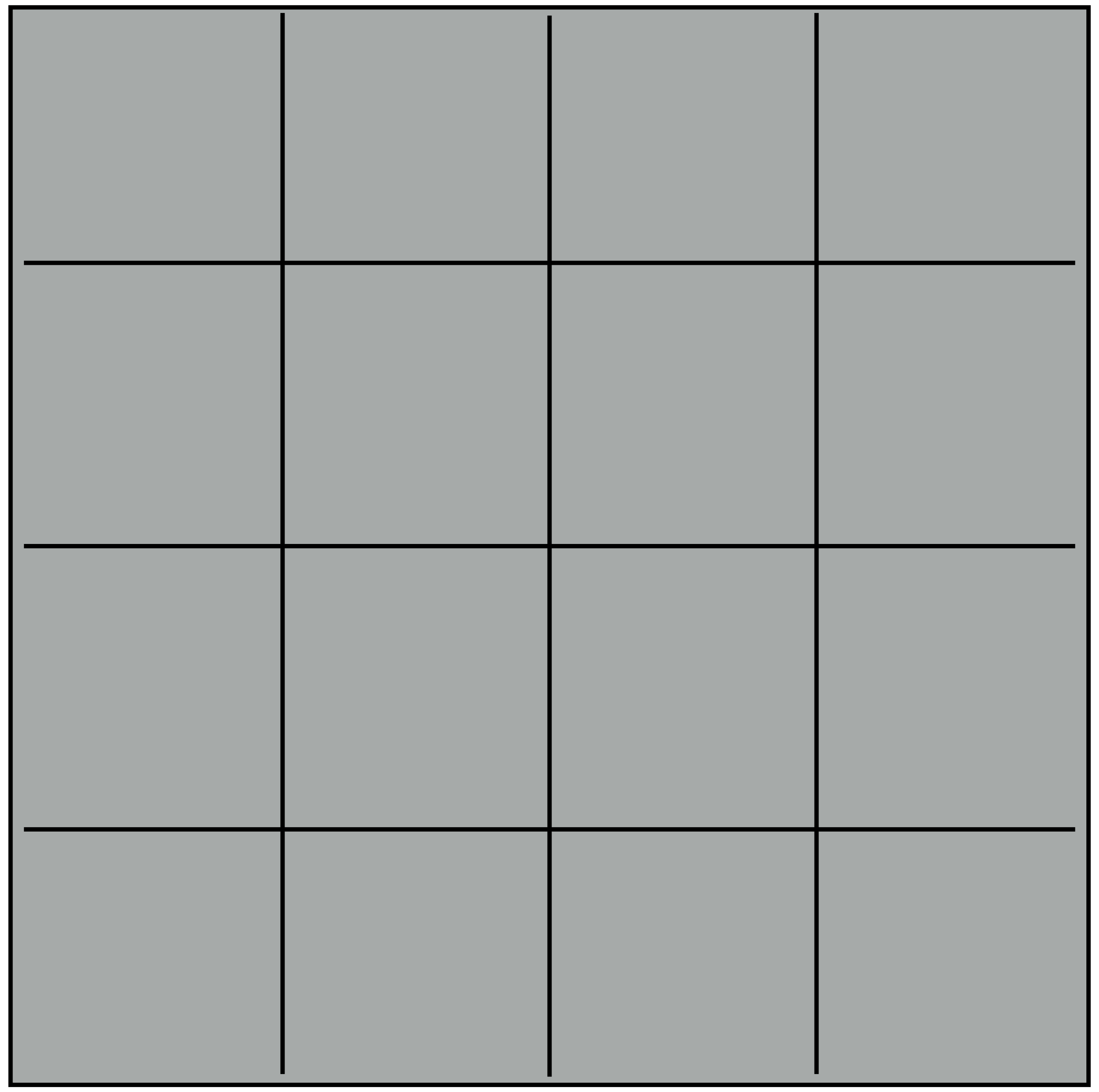
# of filters: 20

= 2000 parameters

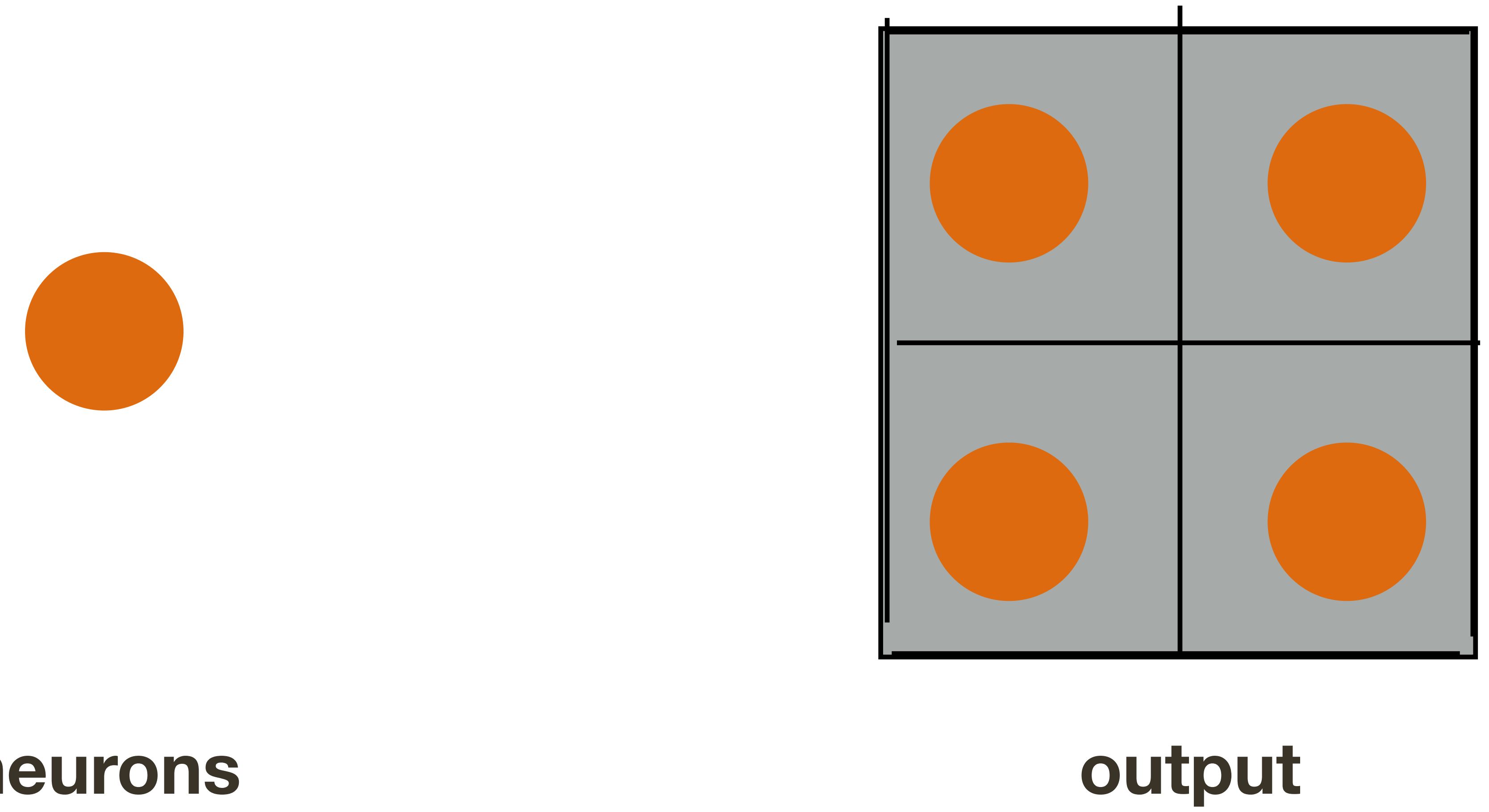
Learn **multiple filters**

* slide from Marc'Aurelio Renzato

Convolutional Layer: Interpretation #2

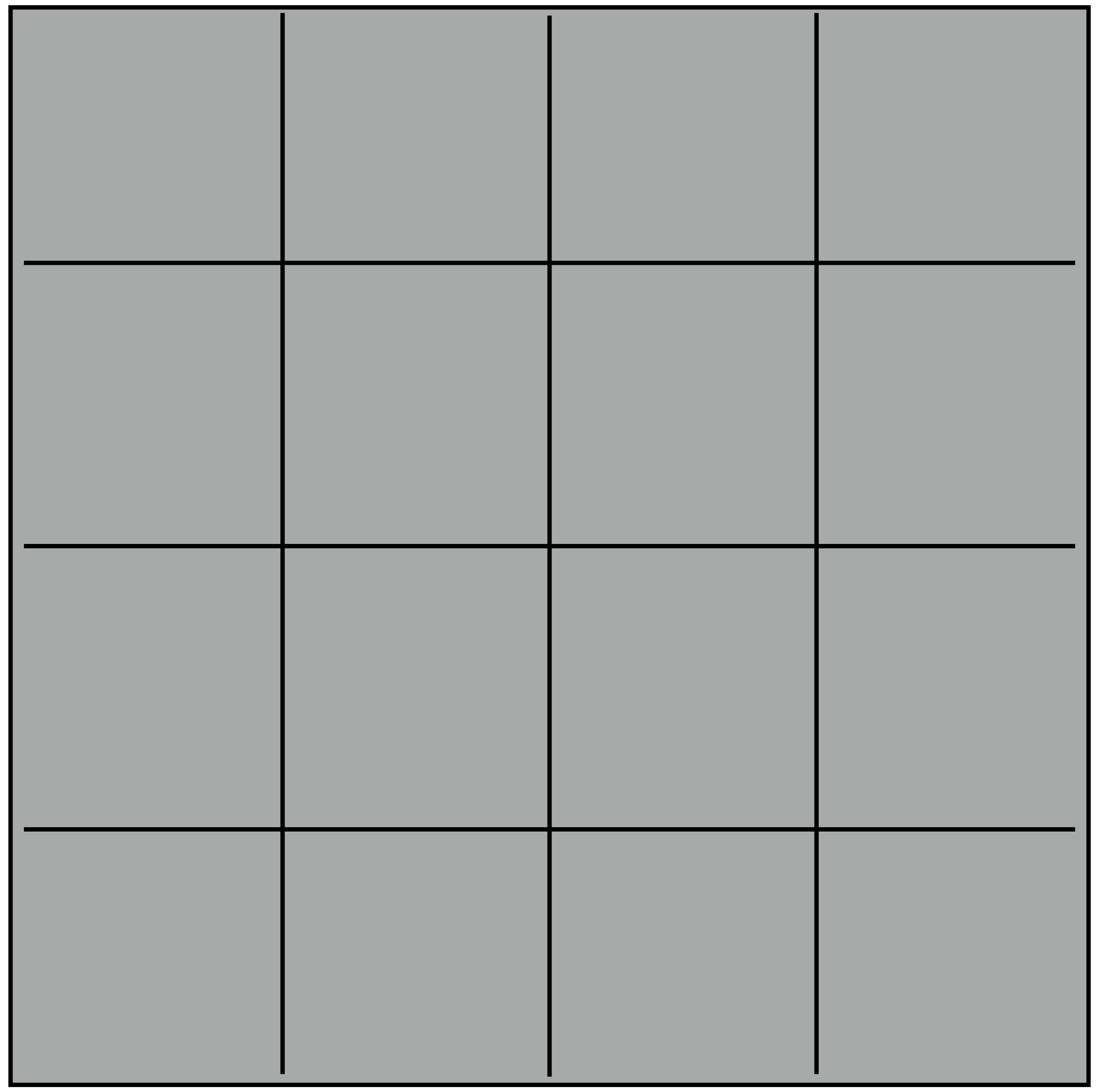


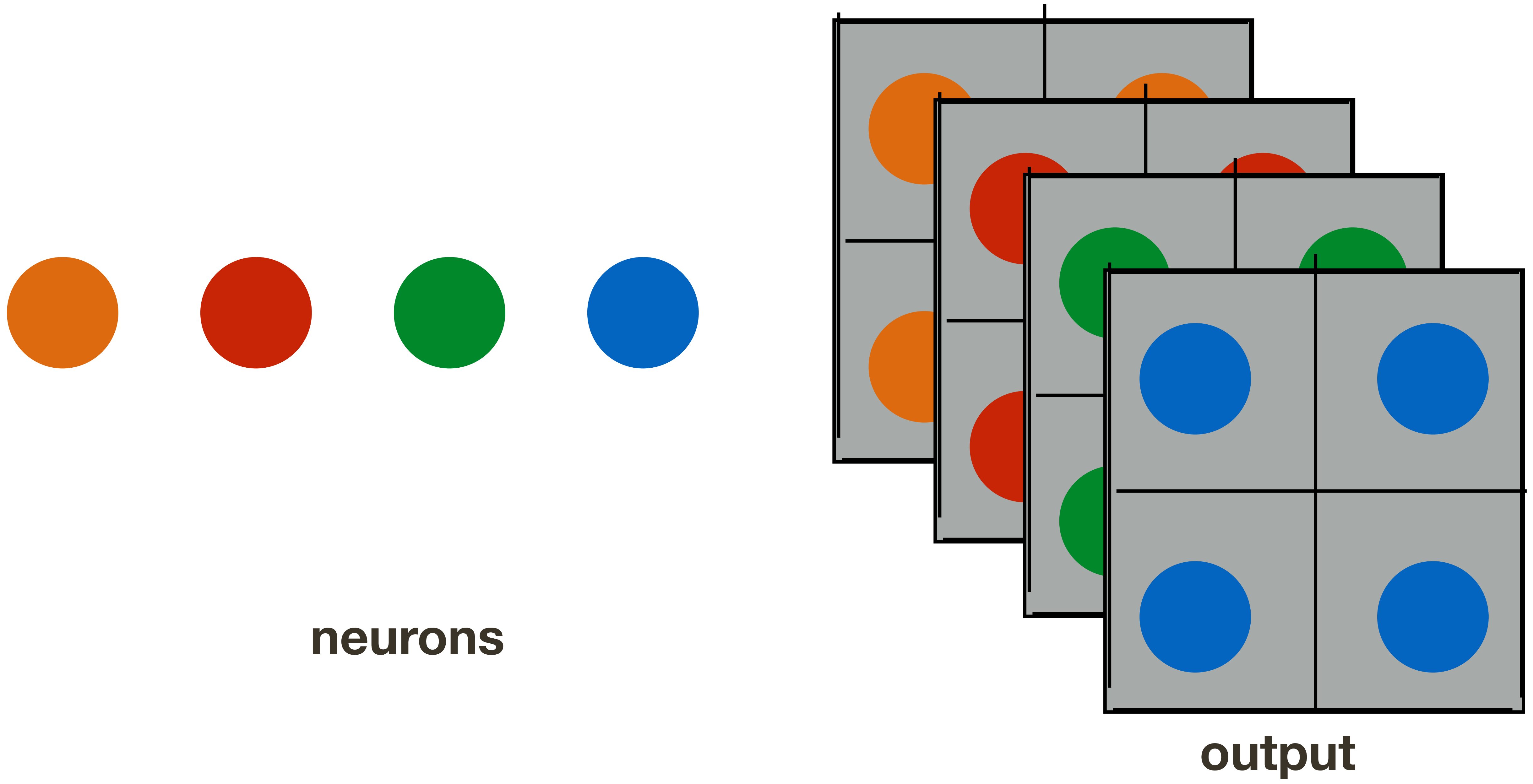
One neuron applied as convolution (by shifting)



Convolutional Layer: Interpretation #2

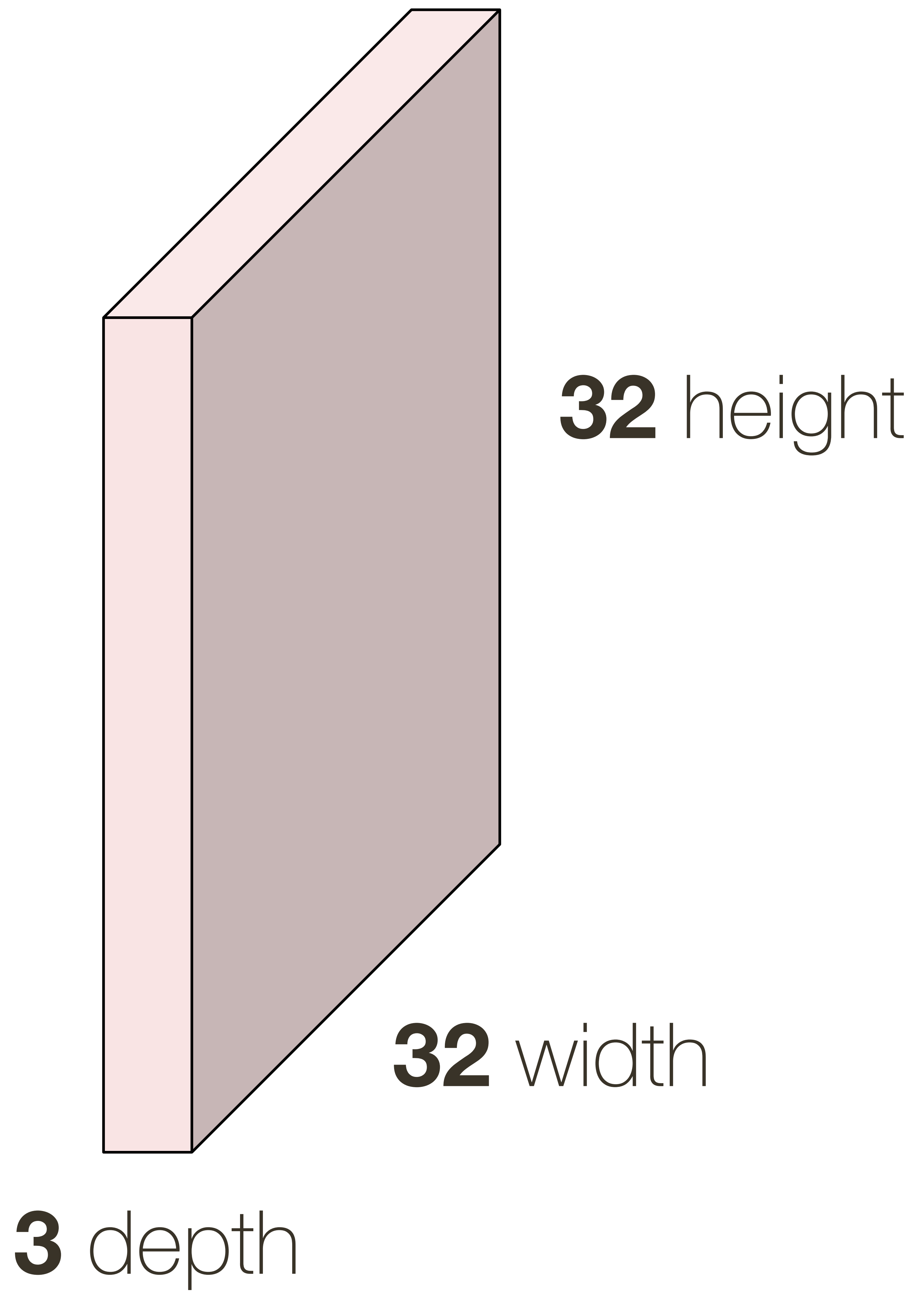
One neuron applied as convolution (by shifting)





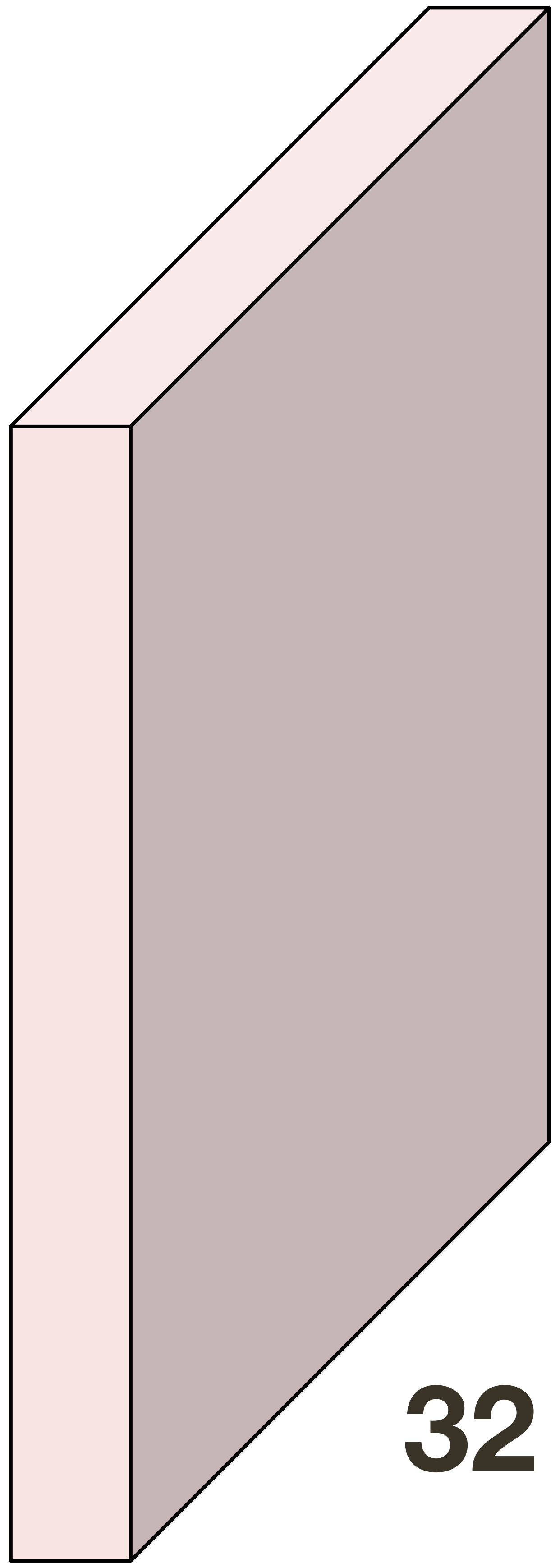
Convolutional Layer

32 x 32 x 3 **image** (note the image preserves spatial structure)



Convolutional Layer

32 x 32 x 3 **image**

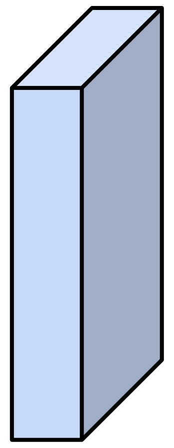


32 height

32 width

3 depth

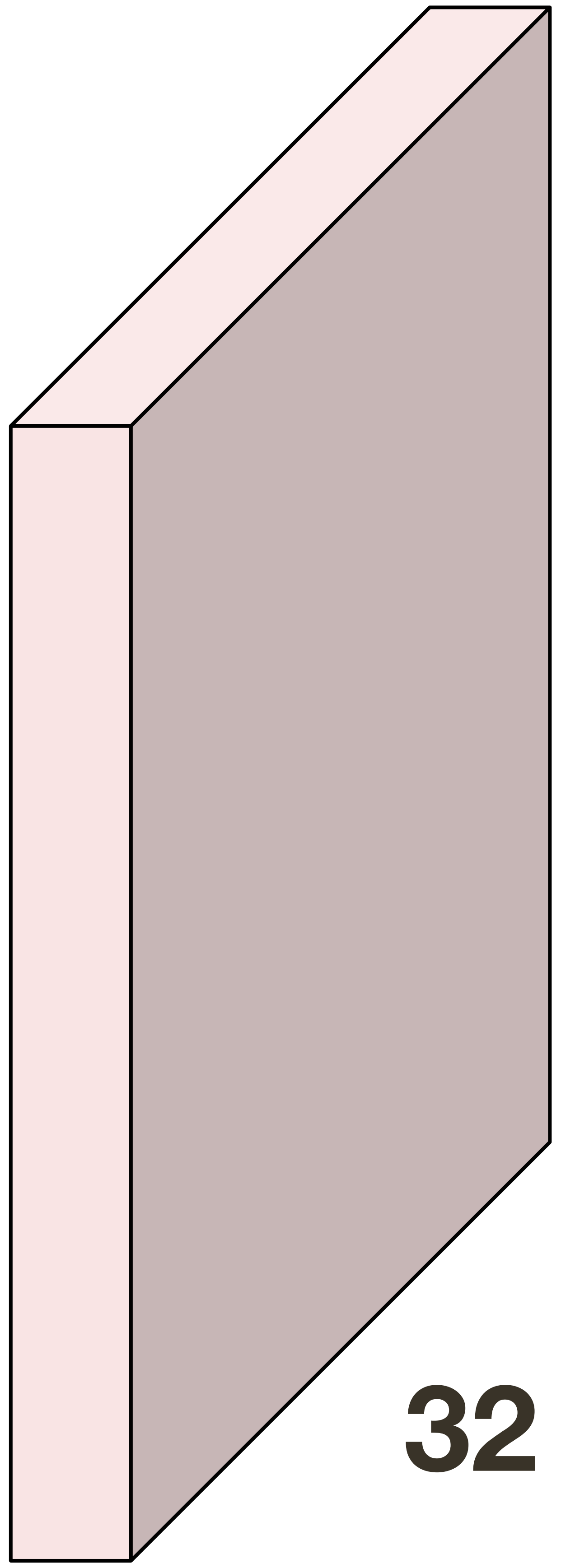
5 x 5 x 3 **filter**



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

Convolutional Layer

32 x 32 x 3 **image**



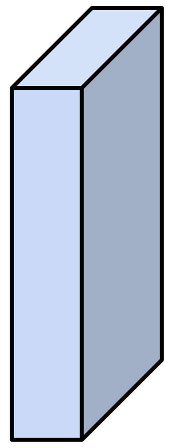
32 height

32 width

3 depth

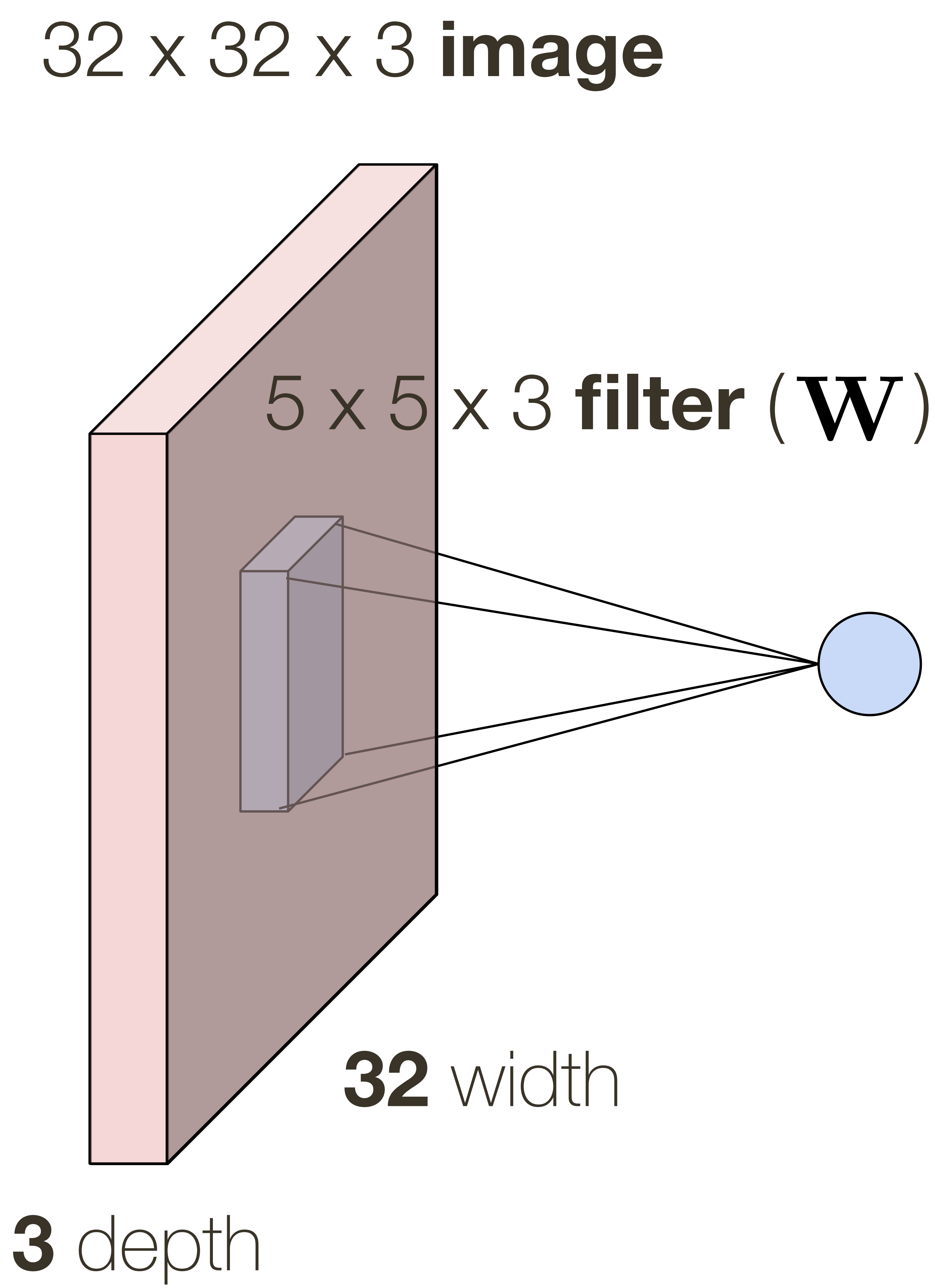
Filters always extend the full depth of the input volume

5 x 5 x 3 **filter**



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

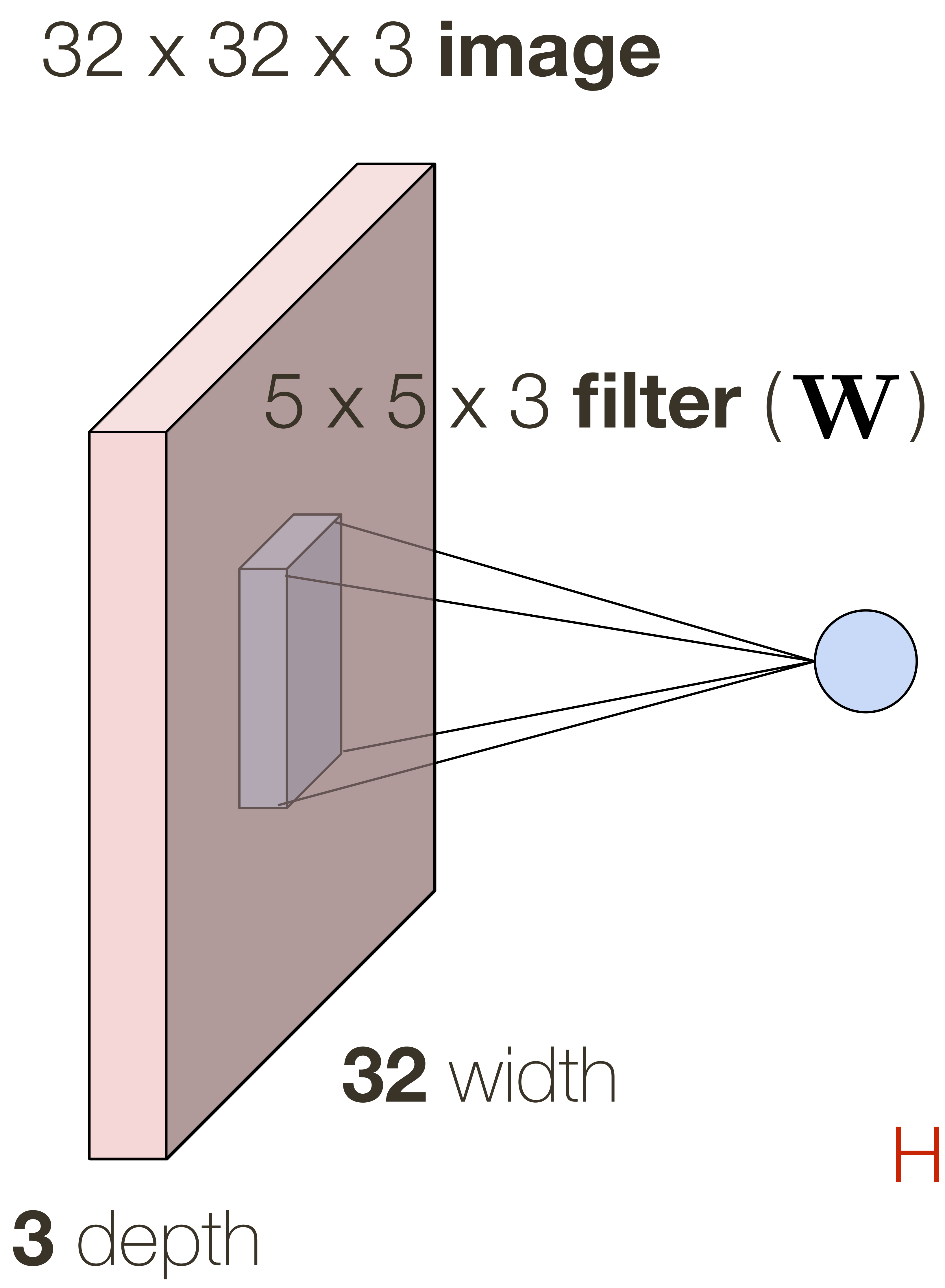
Convolutional Layer



1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

$$\mathbf{W}^T\mathbf{x} + b, \text{where } \mathbf{W}, \mathbf{x} \in \mathbb{R}^{75}$$

Convolutional Layer



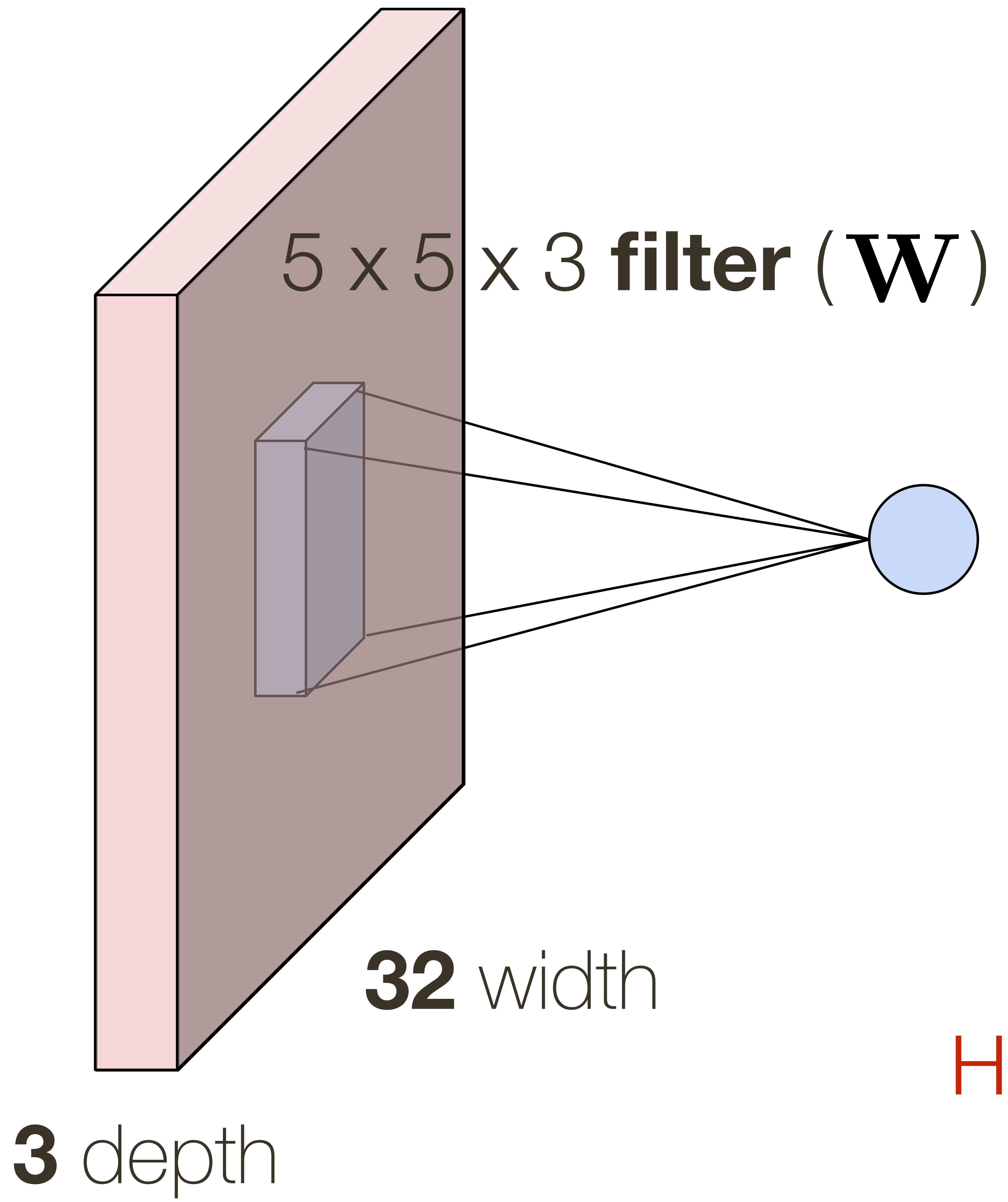
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

$$\mathbf{W}^T\mathbf{x} + b, \text{where } \mathbf{W}, \mathbf{x} \in \mathbb{R}^{75}$$

How many **parameters** does the layer have?

Convolutional Layer

32 x 32 x 3 **image**



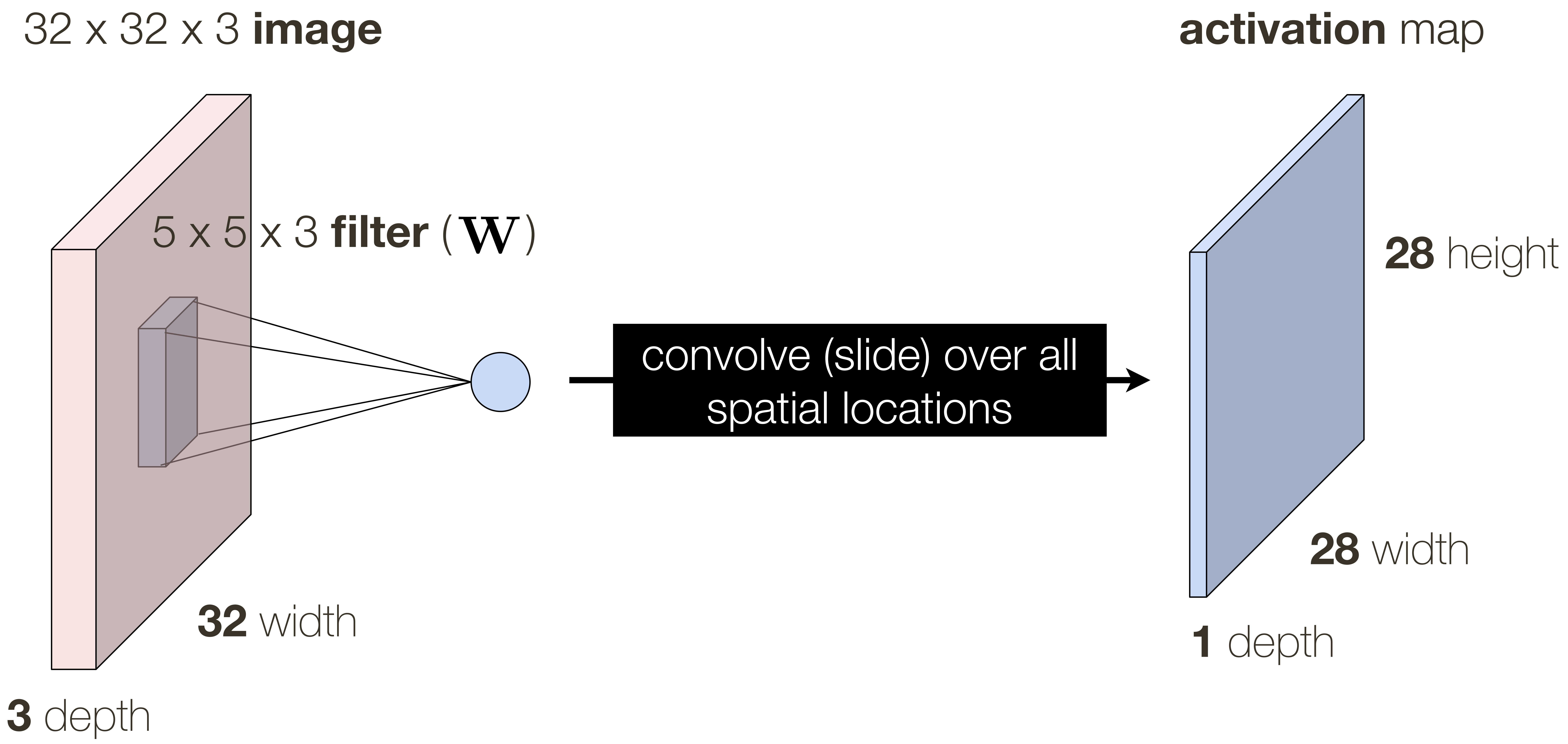
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

$$\mathbf{W}^T\mathbf{x} + b, \text{where } \mathbf{W}, \mathbf{x} \in \mathbb{R}^{75}$$

How many **parameters** does the layer have? **76**

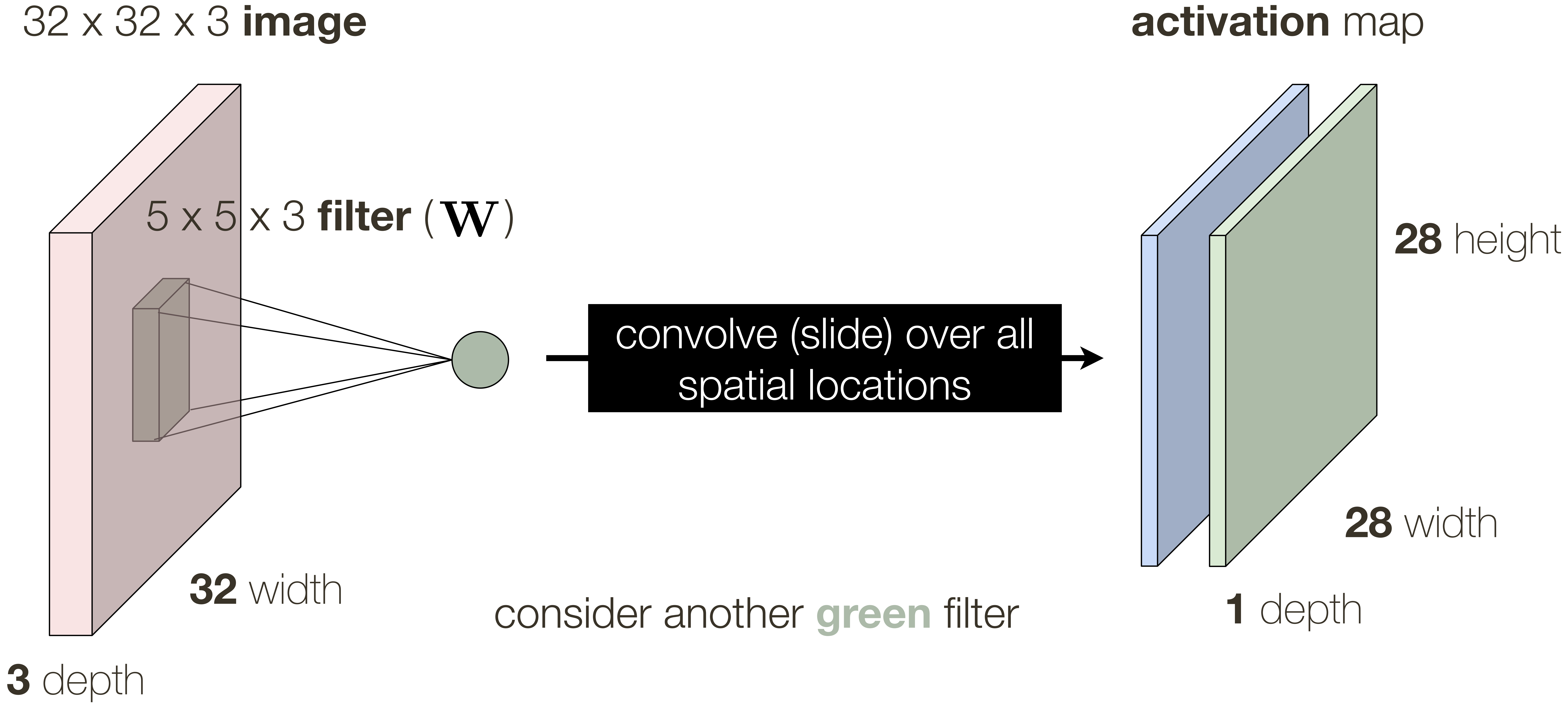
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer



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

Convolutional Layer

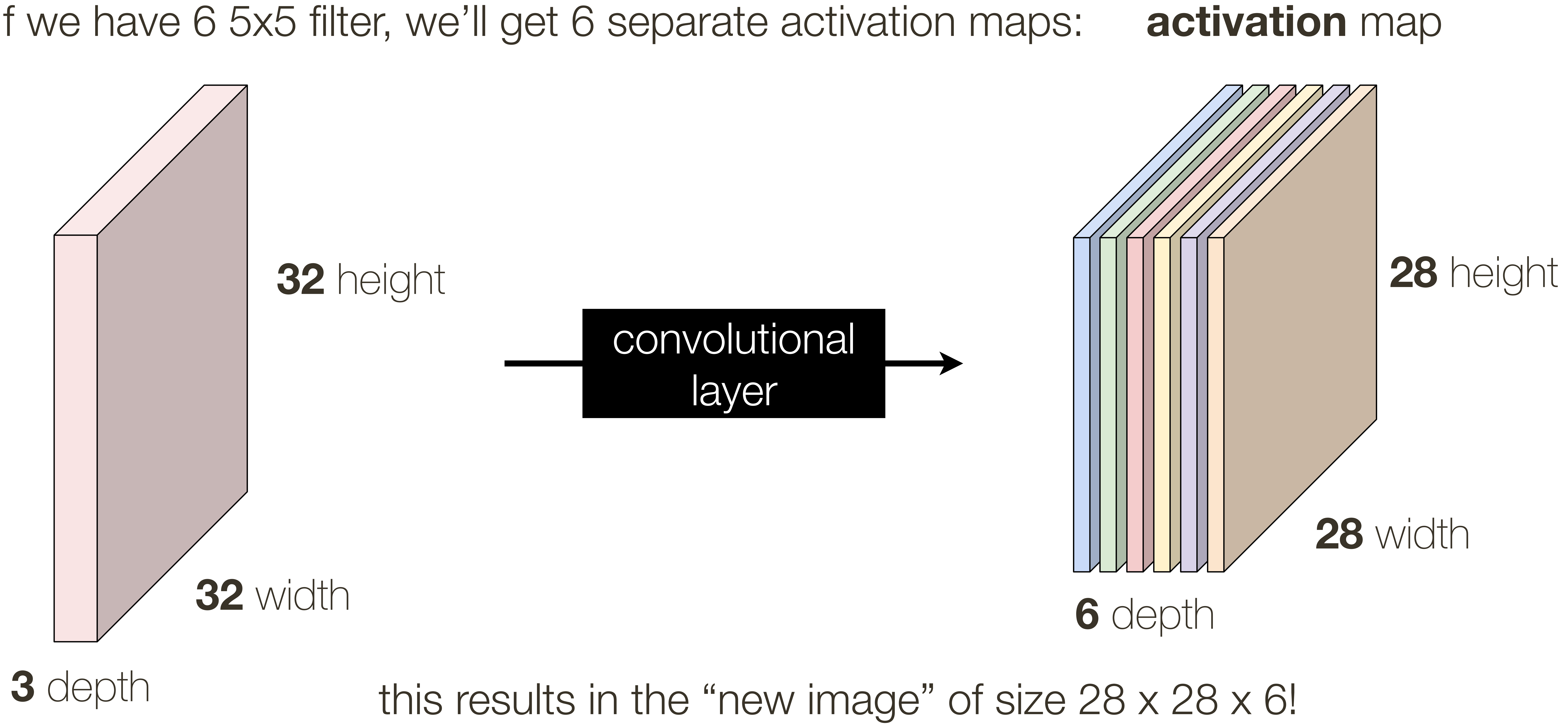


consider another **green** filter

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

Convolutional Layer

If we have 6 5x5 filter, we'll get 6 separate activation maps:



this results in the "new image" of size 28 x 28 x 6!

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

Convolutional Layer

The number of neurons in a layer is determined by depth and stride parameter

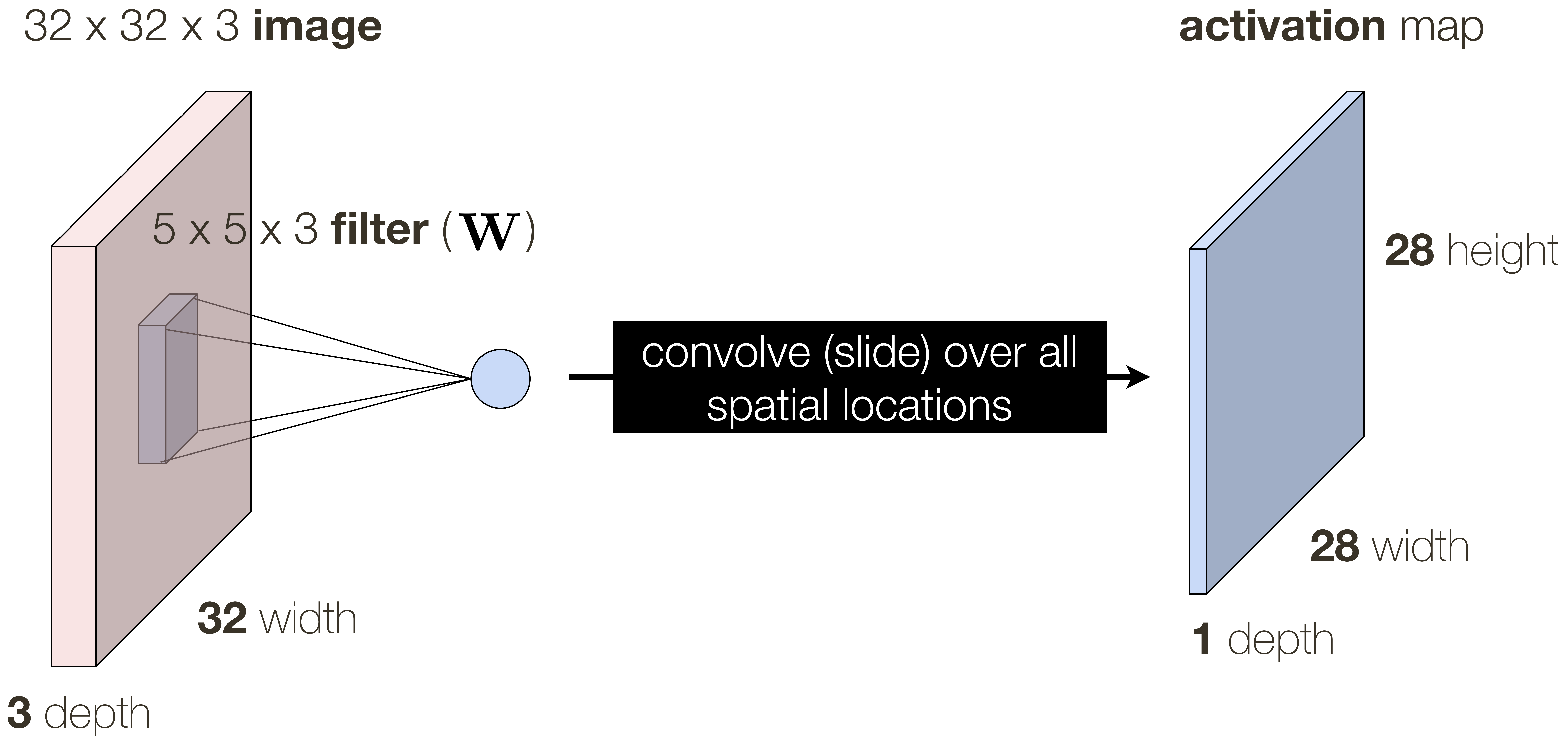
— also affected by zero-padding

Depth: Controls number of neurons that connect to the same region of the input layer

— a set of neurons connected to the same region is called a **depth column**

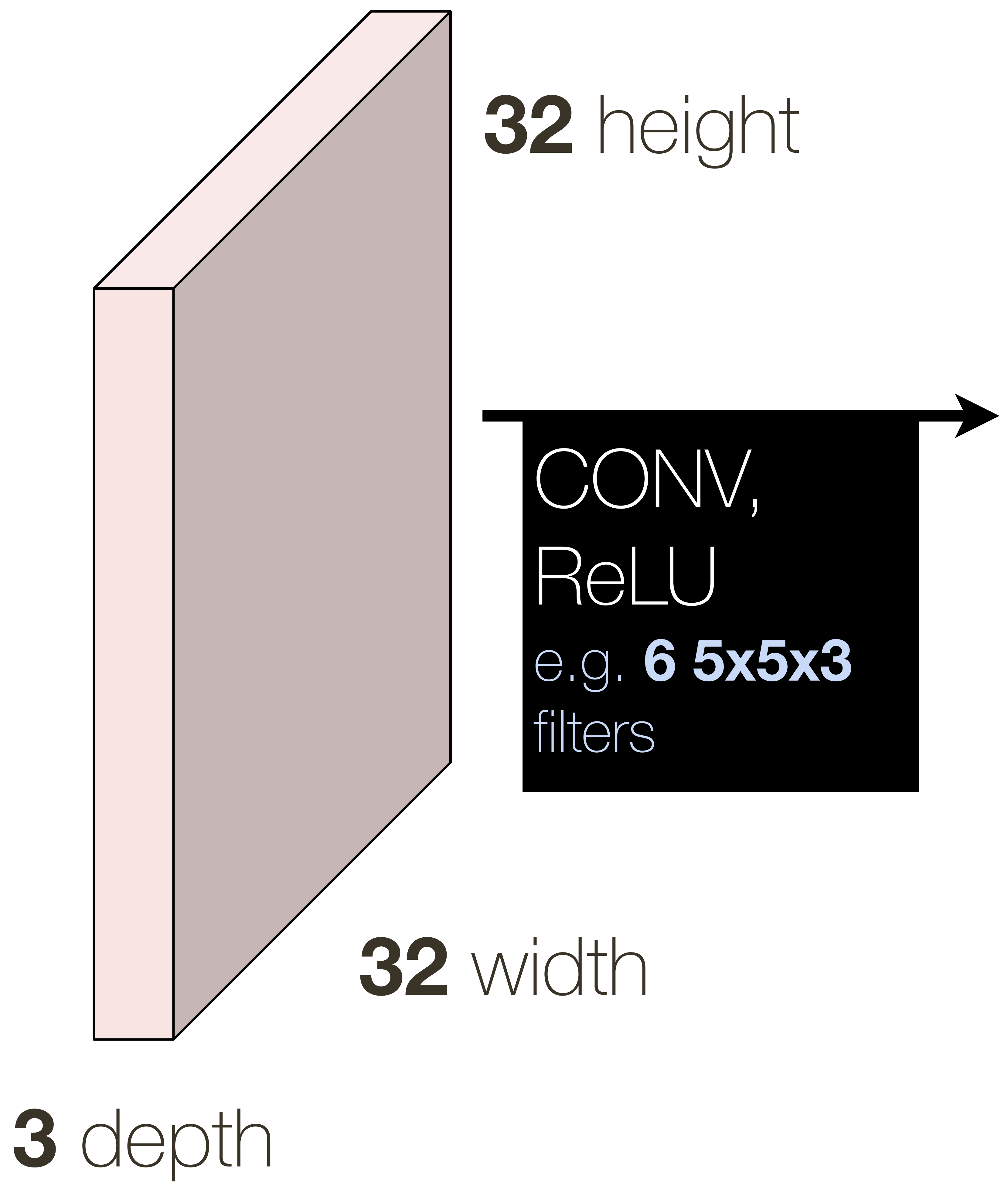
Stride: Controls spatial density. How far apart are depth columns?

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



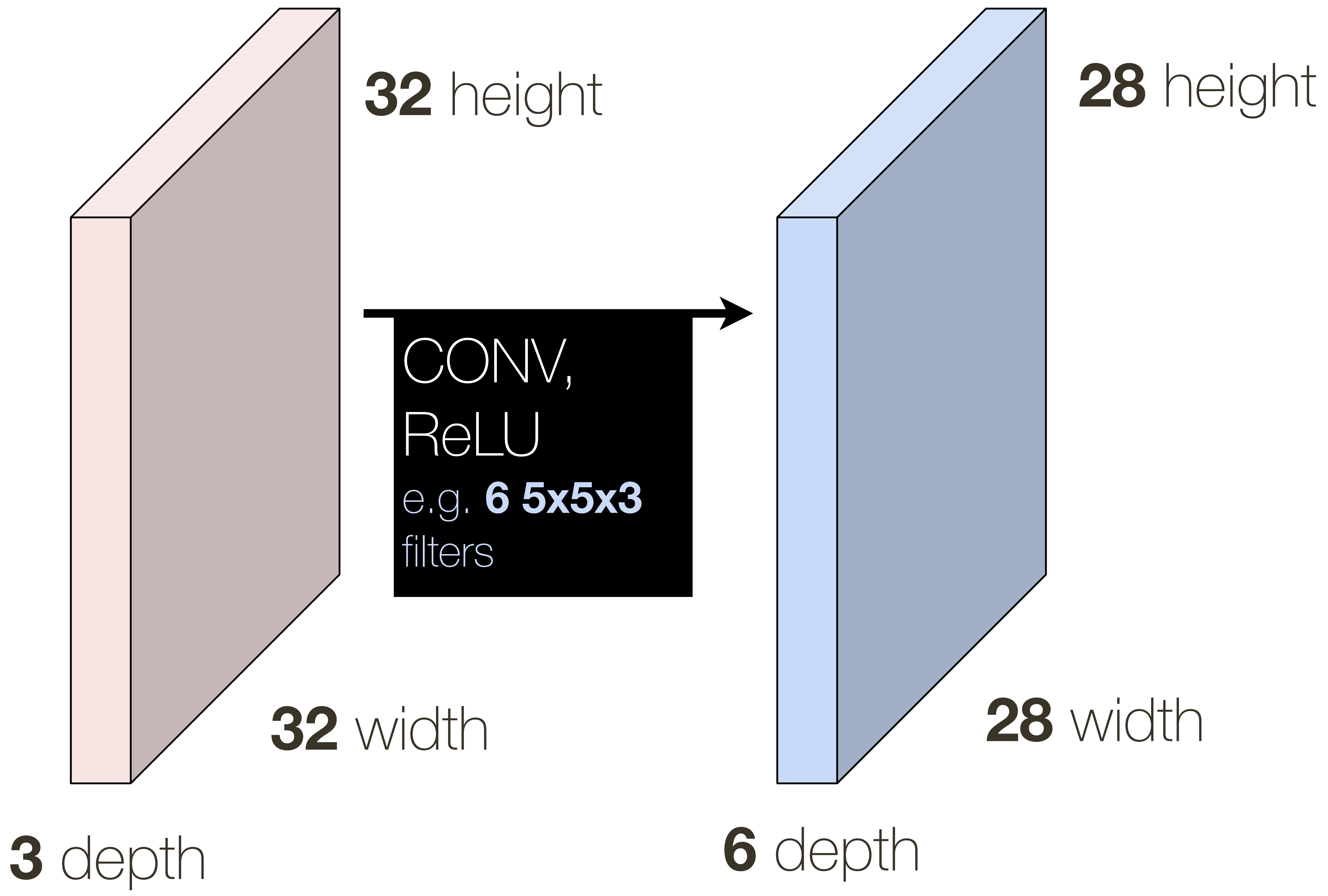
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



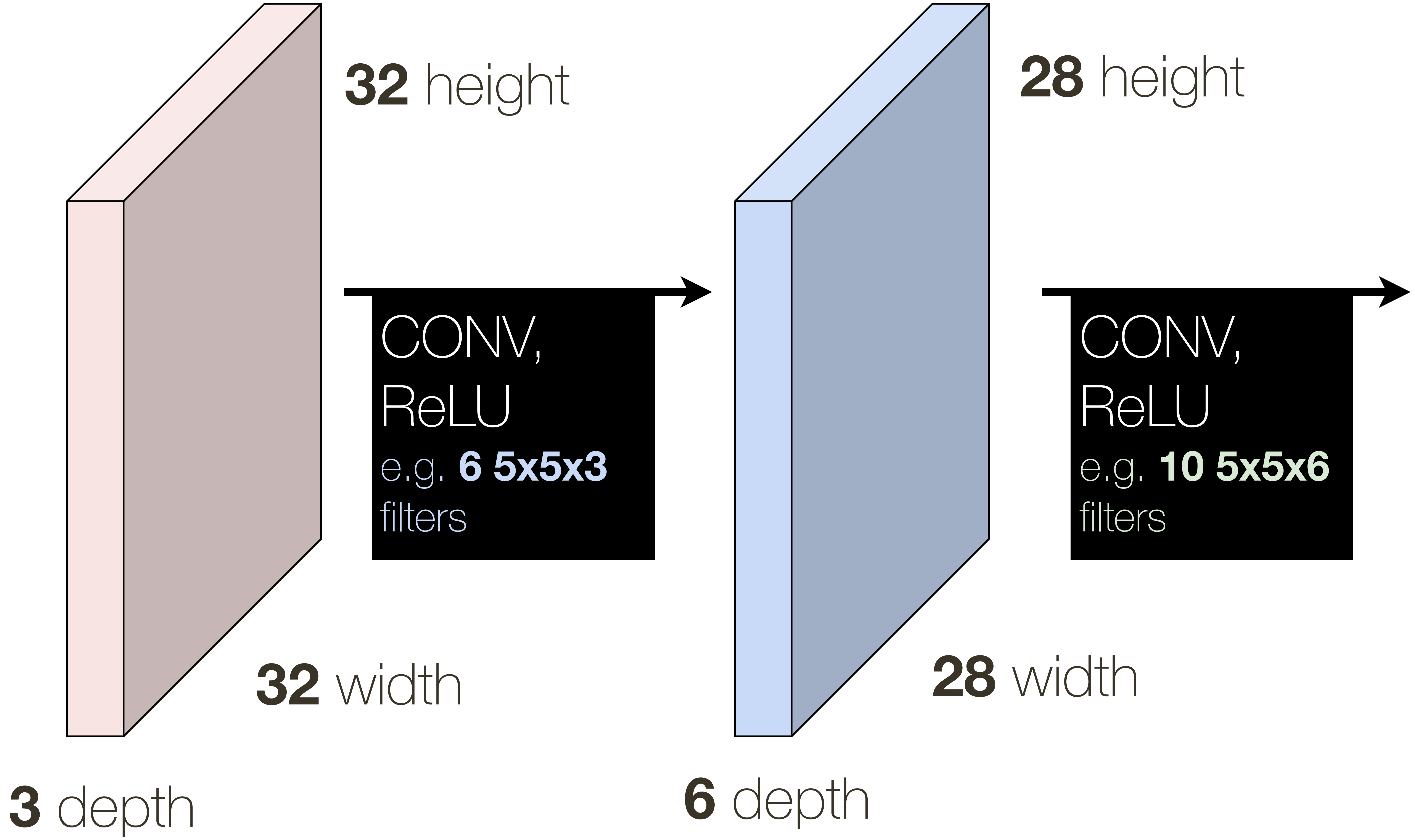
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



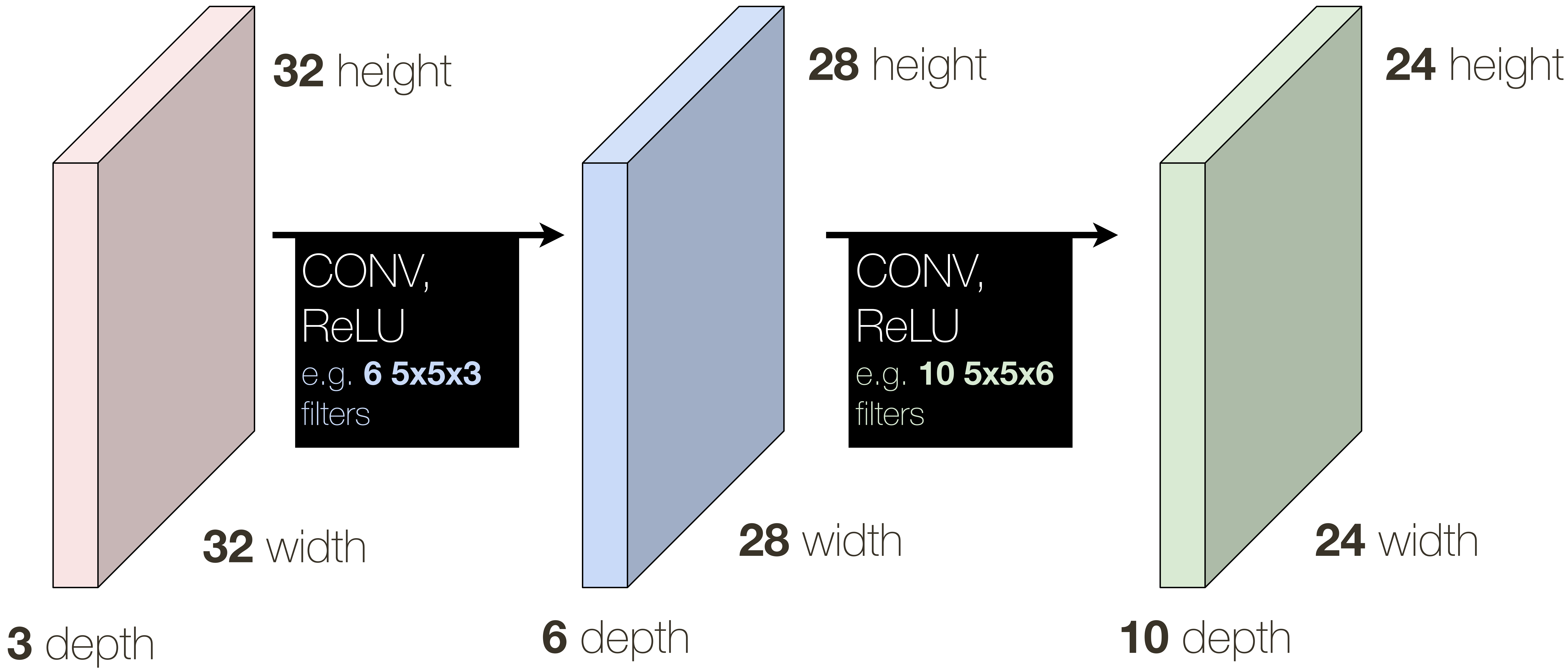
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



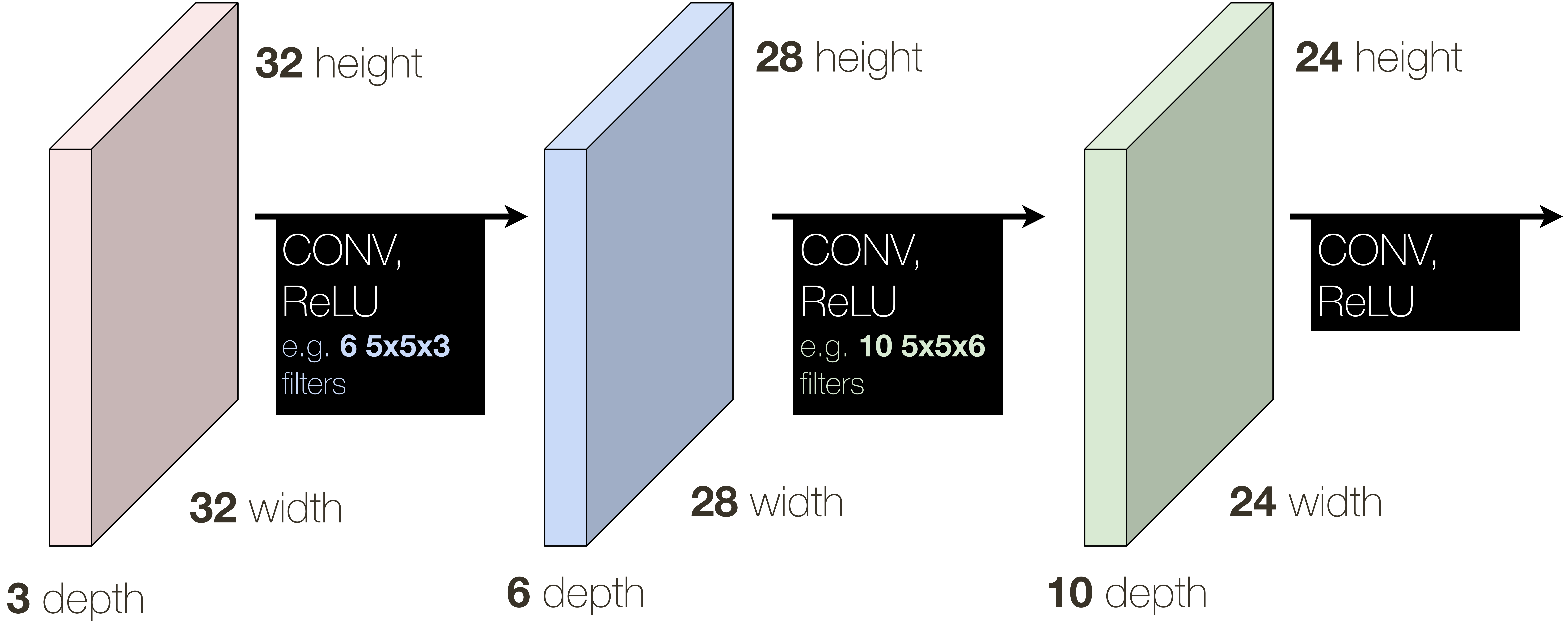
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



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

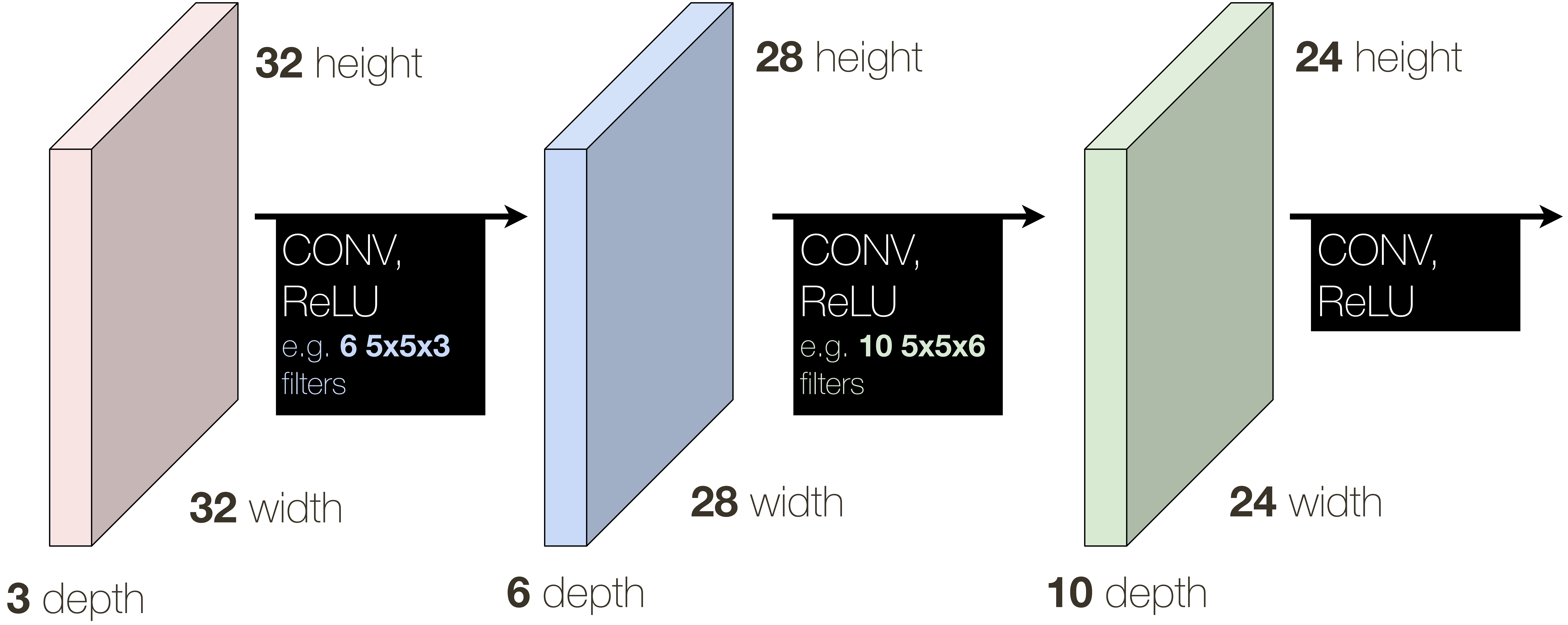
Convolutional Neural Network (ConvNet)



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

Convolutional Neural Network (ConvNet)

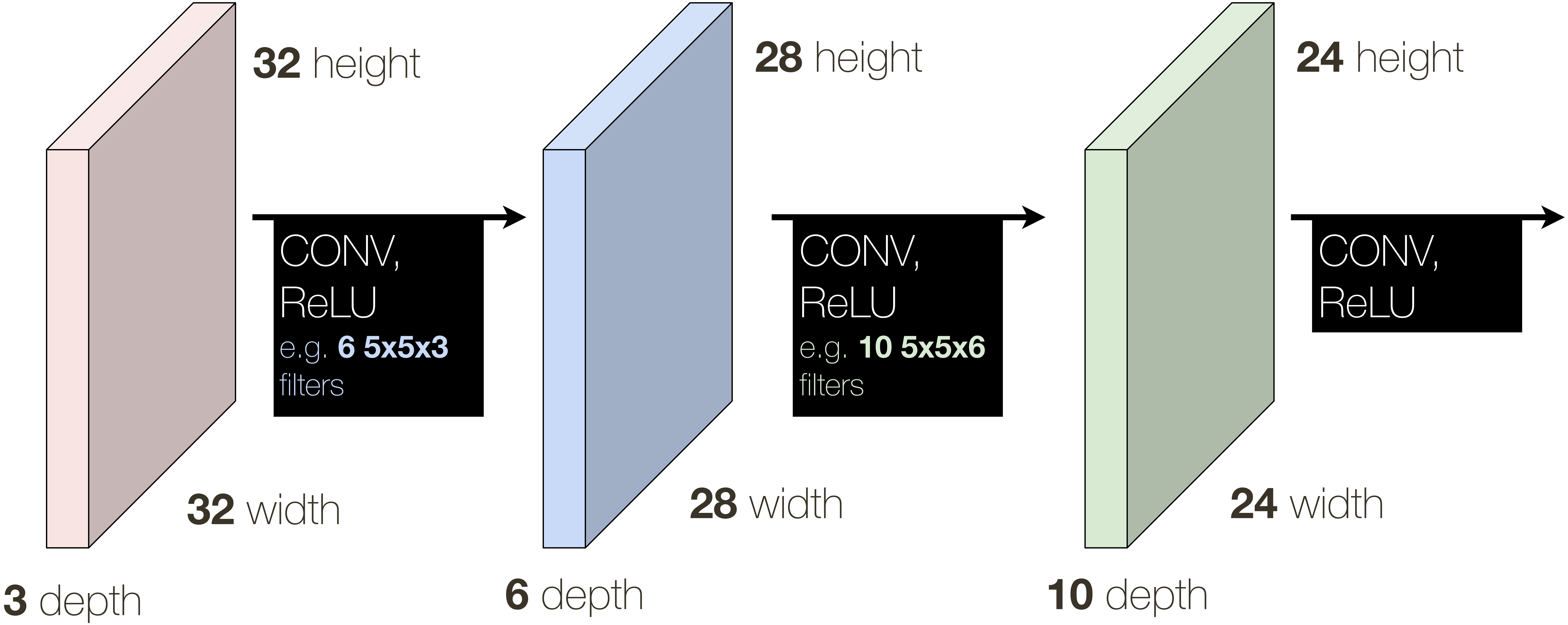
With padding we can achieve no shrinking (32 -> 28 -> 24); shrinking quickly (which happens with larger filters) doesn't work well in practice



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

Receptive Fields

What does a **neuron** in green layer sees?

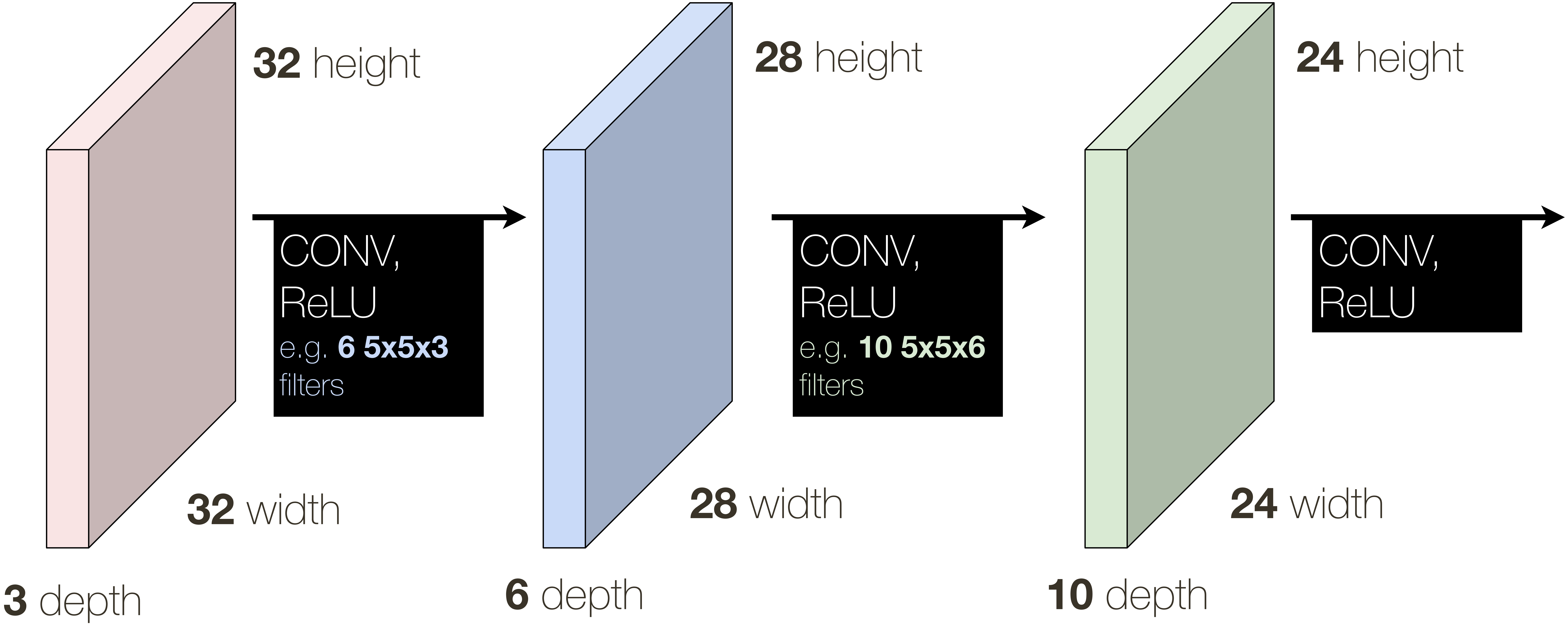


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Receptive Fields

What does a **neuron** in green layer sees?

9 x 9 pixel patch



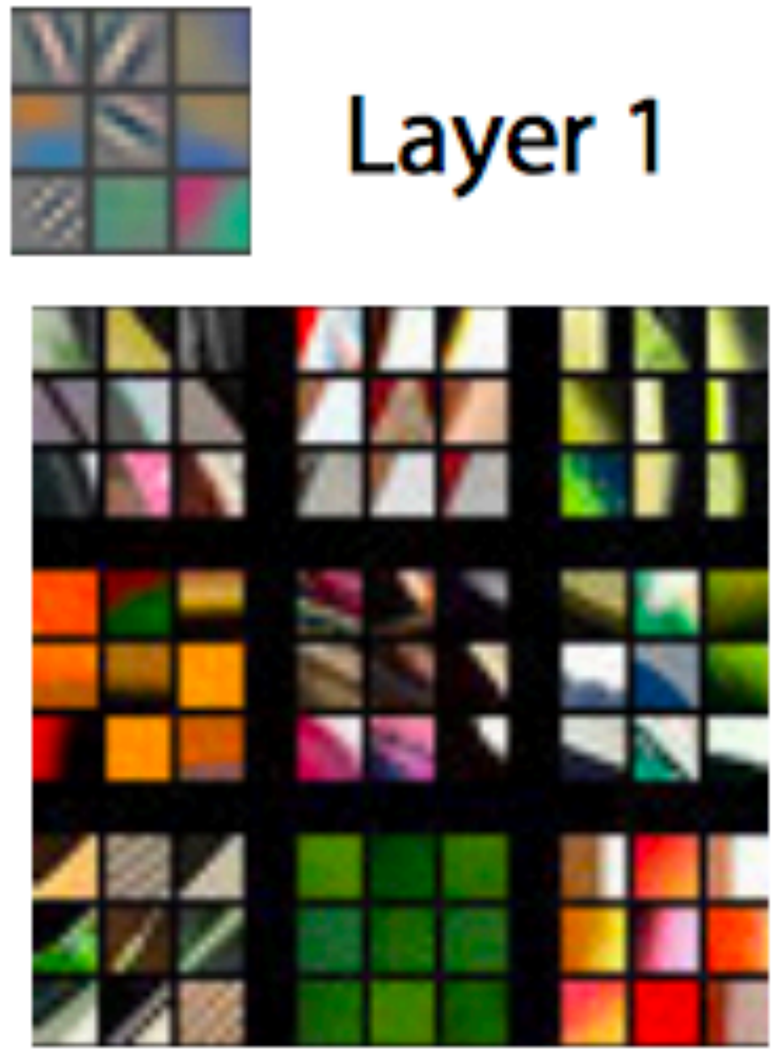
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

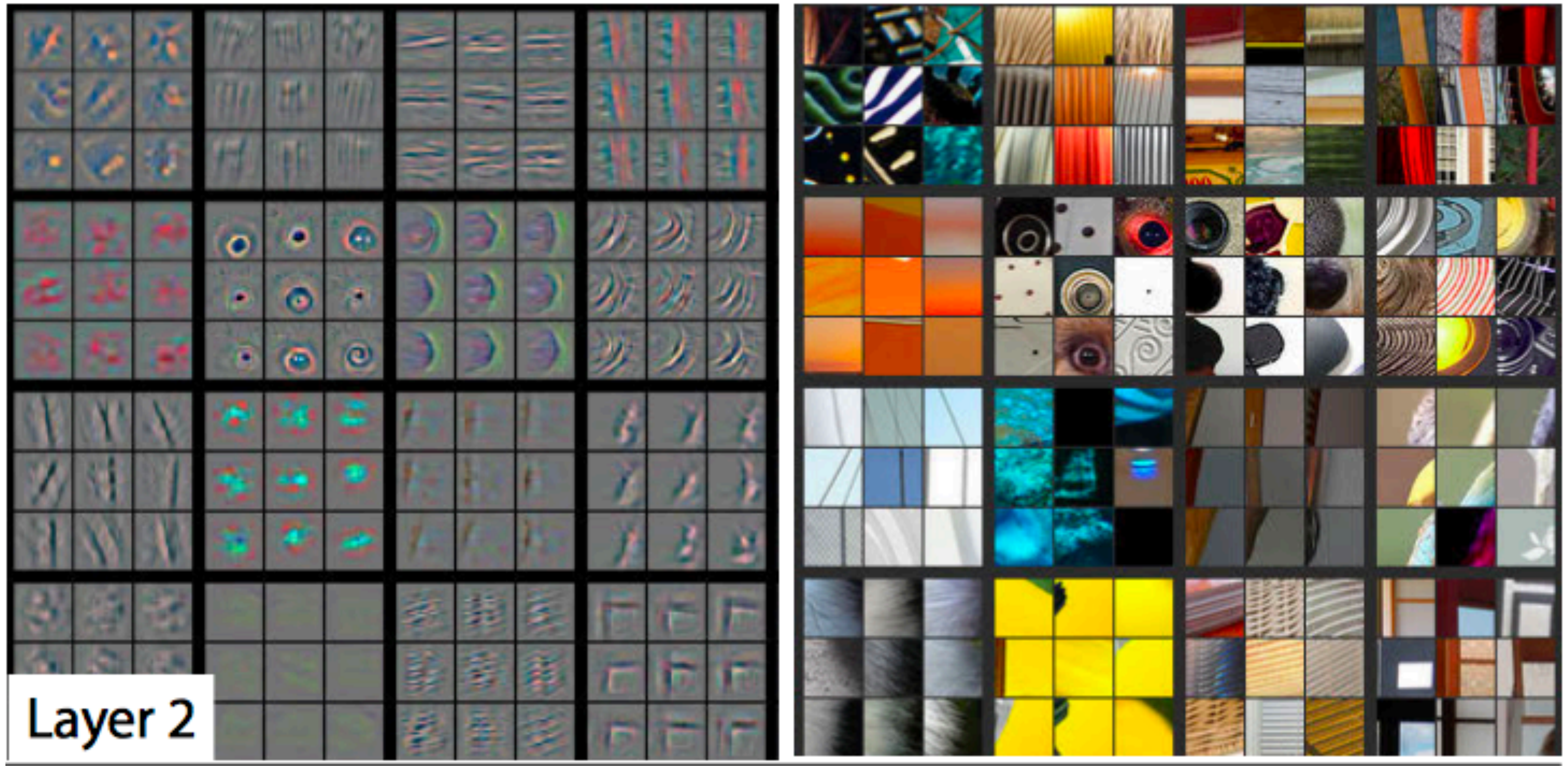
Convolutional Neural Network (ConvNet)

Convolutional neural networks can be seen as learning a hierarchy of filters.

As we go deeper in the network, filters learn and respond to increasingly specialized structures
— The first layers may contain simple orientation filters, middle layers may respond to common substructures, and final layers may respond to entire objects

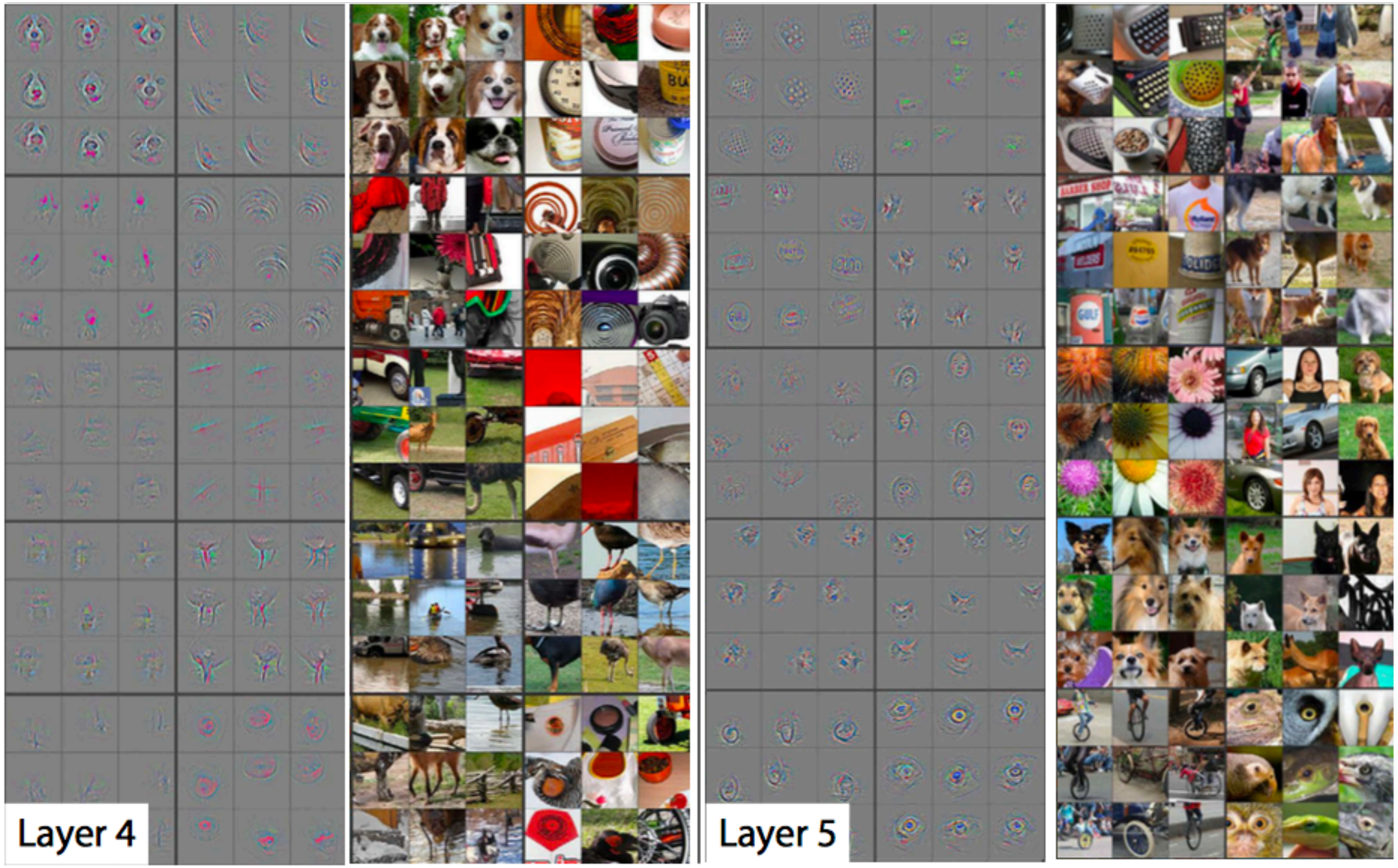
What **filters** do networks learn?





[Zeiler and Fergus, 2013]

What **filters** do networks learn?

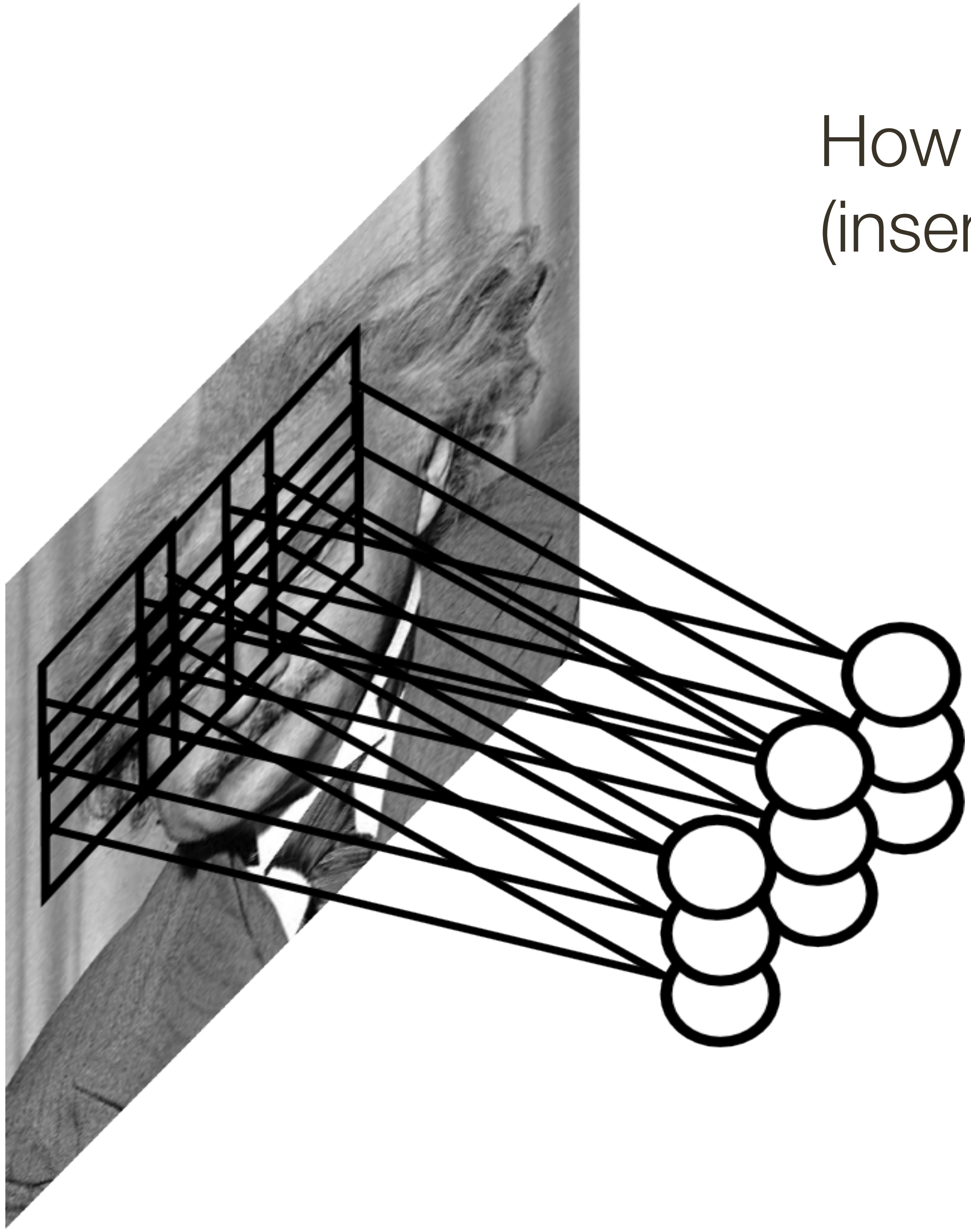


[Zeiler and Fergus, 2013]

Today's "**fun**" Example: Deep Dream — Algorithmic Pareidolia

Today's "**fun**" Example: Deep Dream — Algorithmic Pareidolia

Pooling Layer

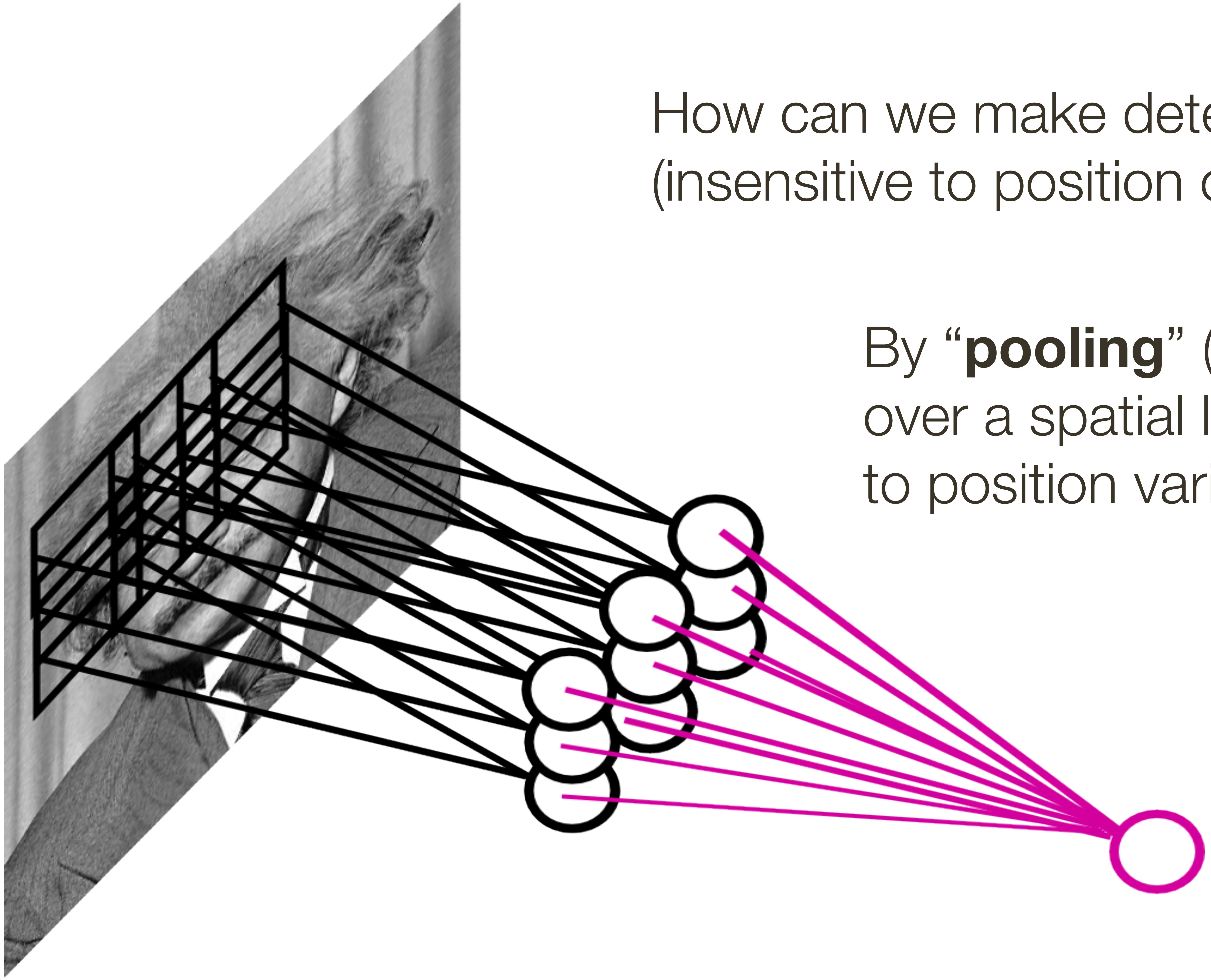


Let us assume the filter is an “eye” detector

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

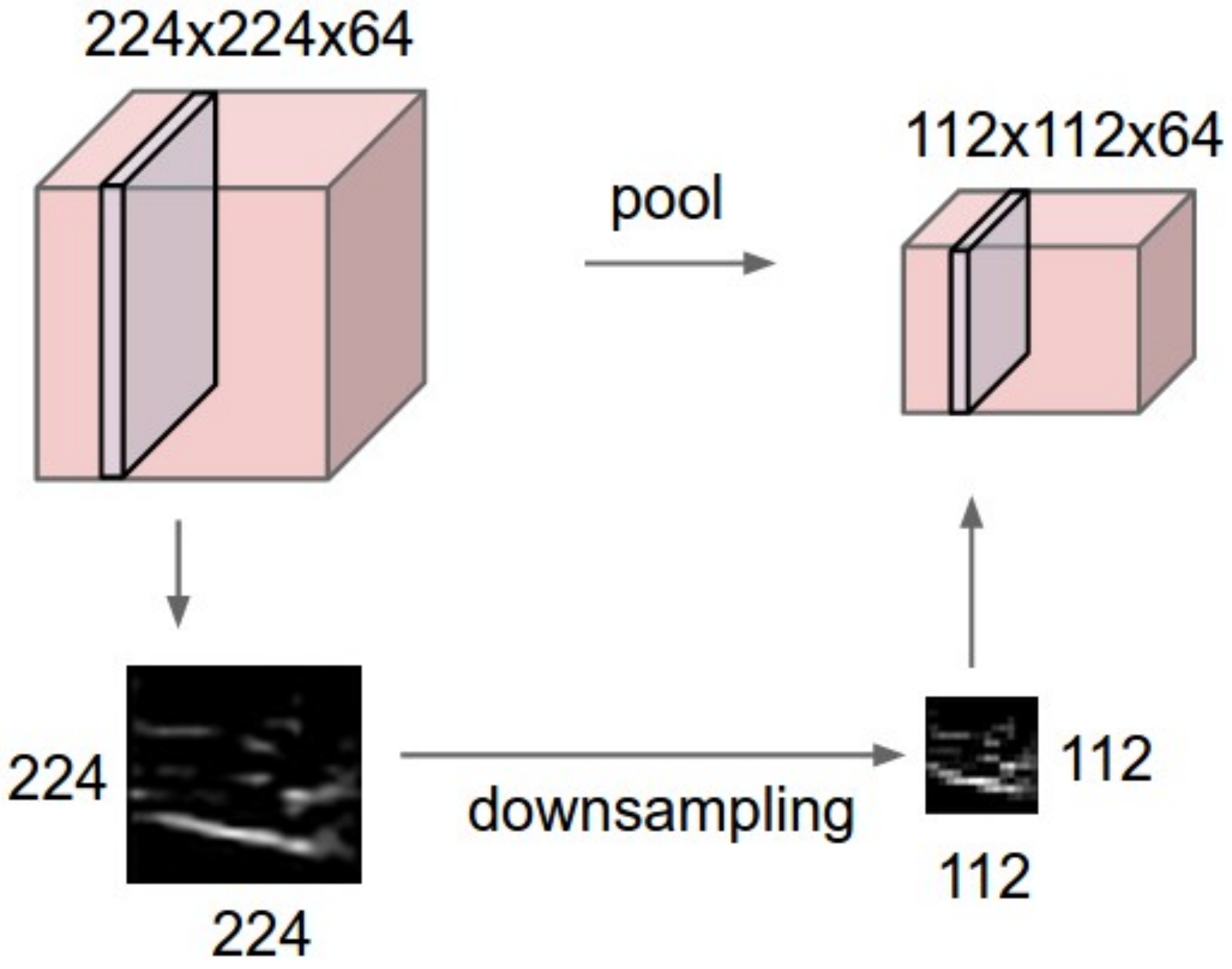
How can we make detection spatially invariant (insensitive to position of the eye in the image)

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

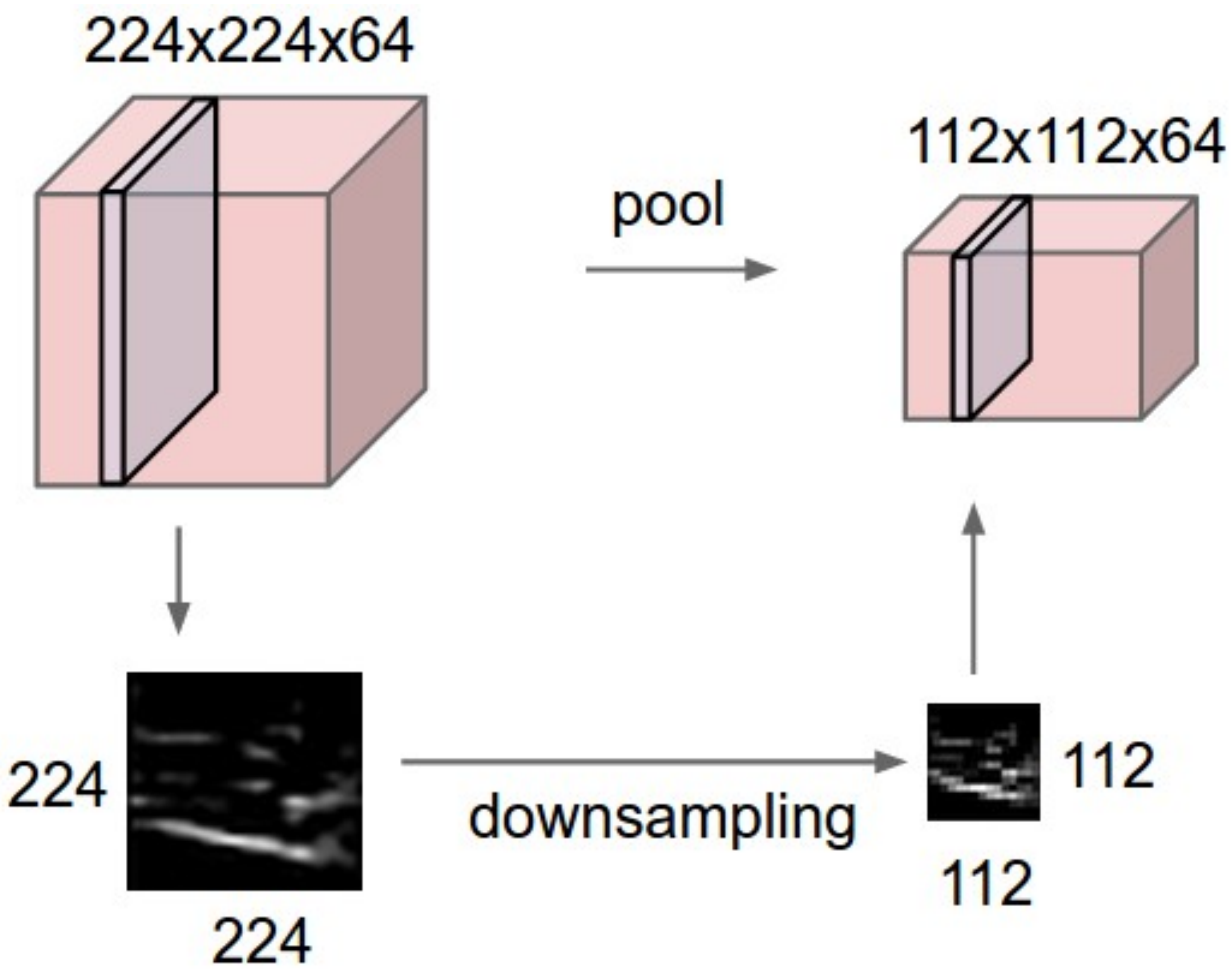
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



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

Pooling Layer

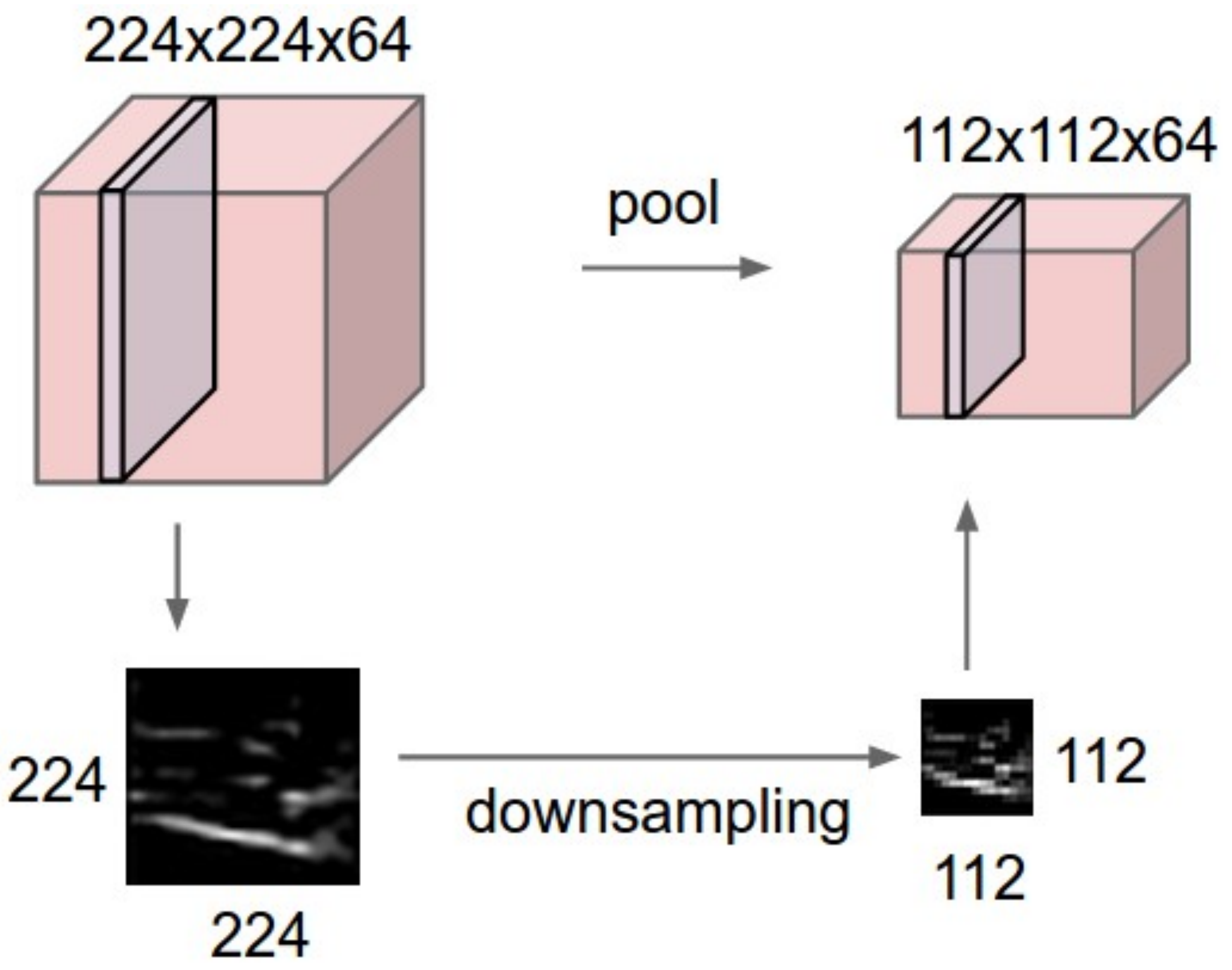
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



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**

Max **Pooling**

activation map

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

Average **Pooling**

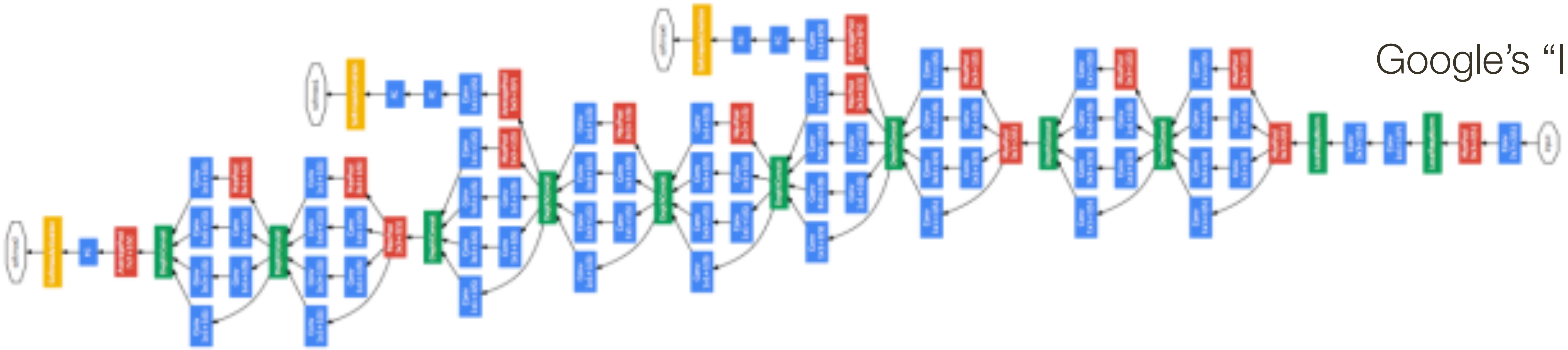
activation map

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

avg pool with 2 x 2 filter
and stride of 2

3.25	5.25
2	2

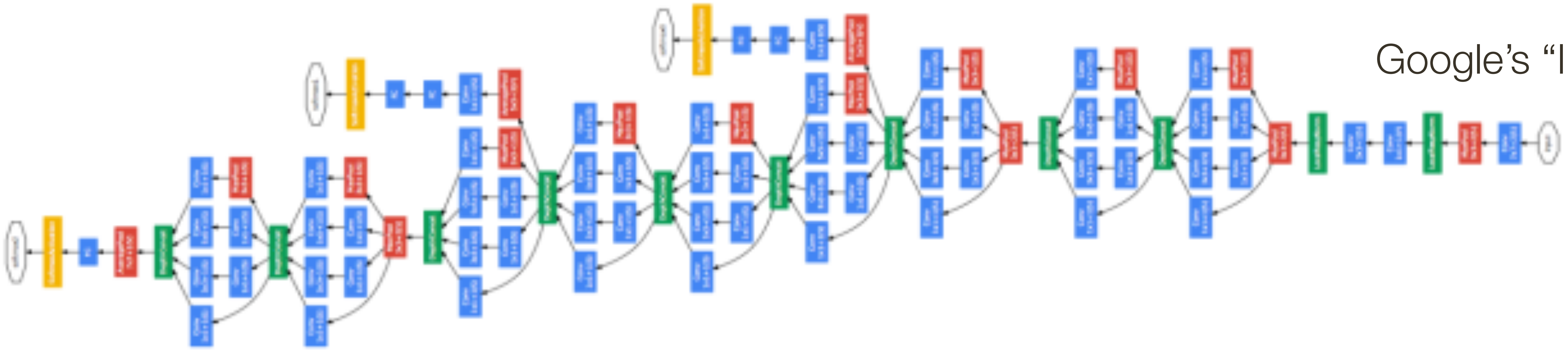
Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

Deep Learning **Terminology**

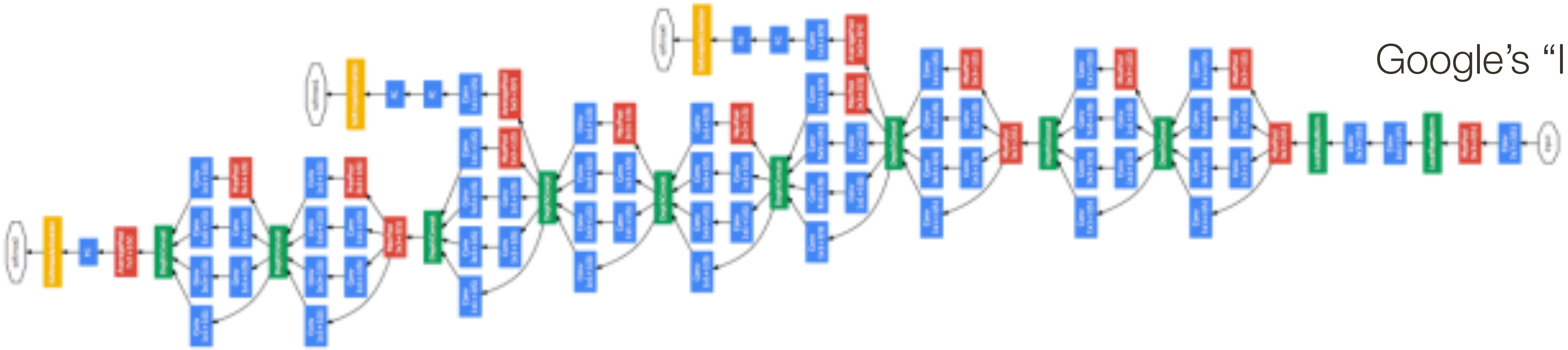


Google's “Inception” network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

Deep Learning **Terminology**



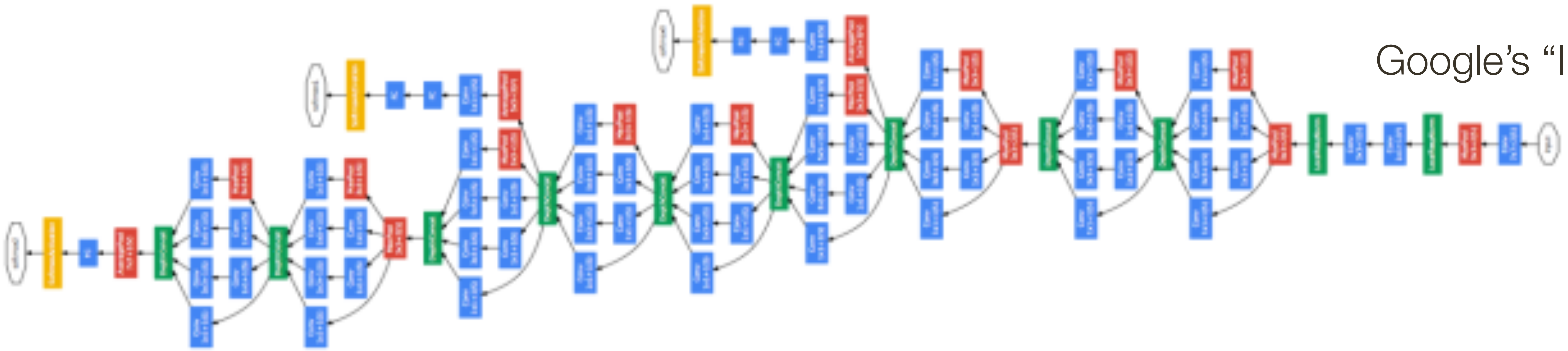
Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

Deep Learning **Terminology**



Google's "Inception" network

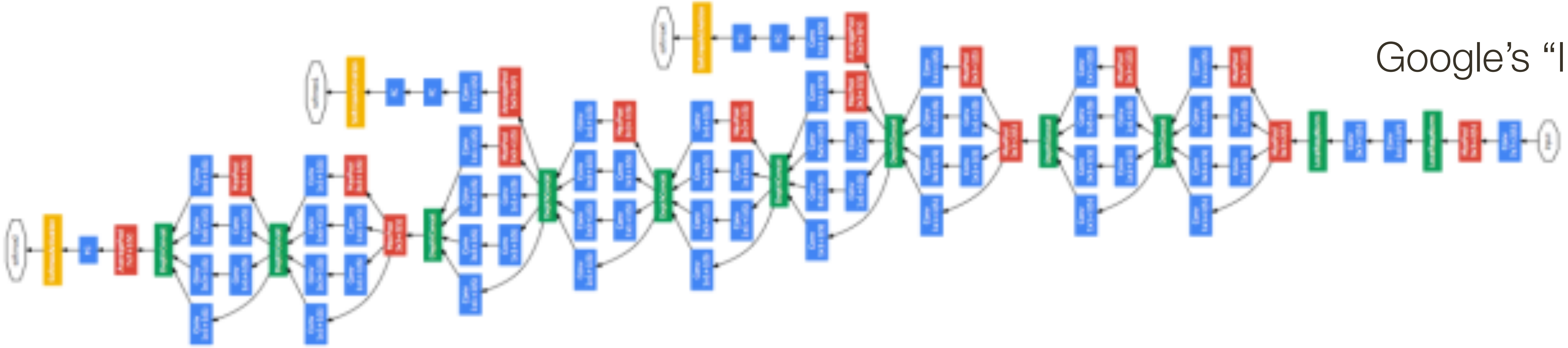
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

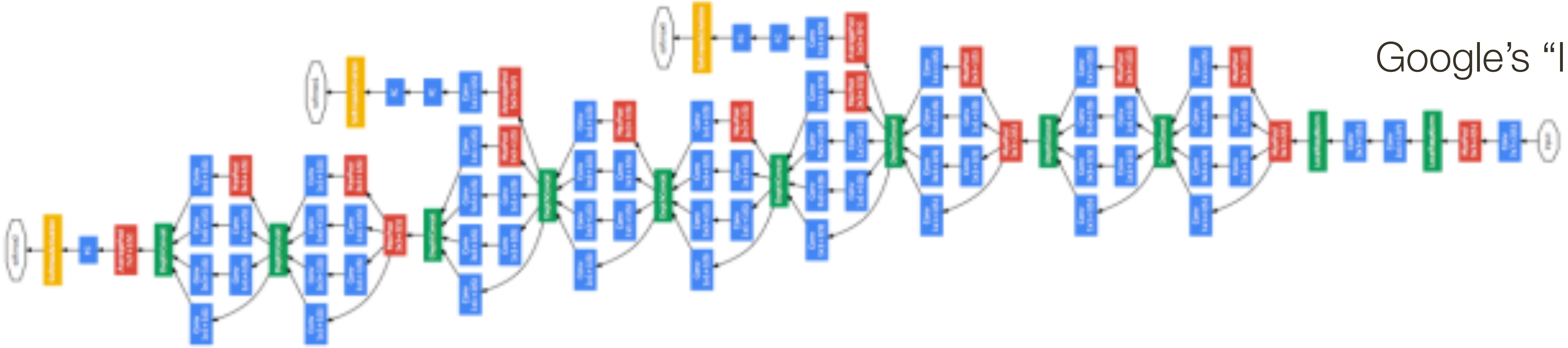
generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

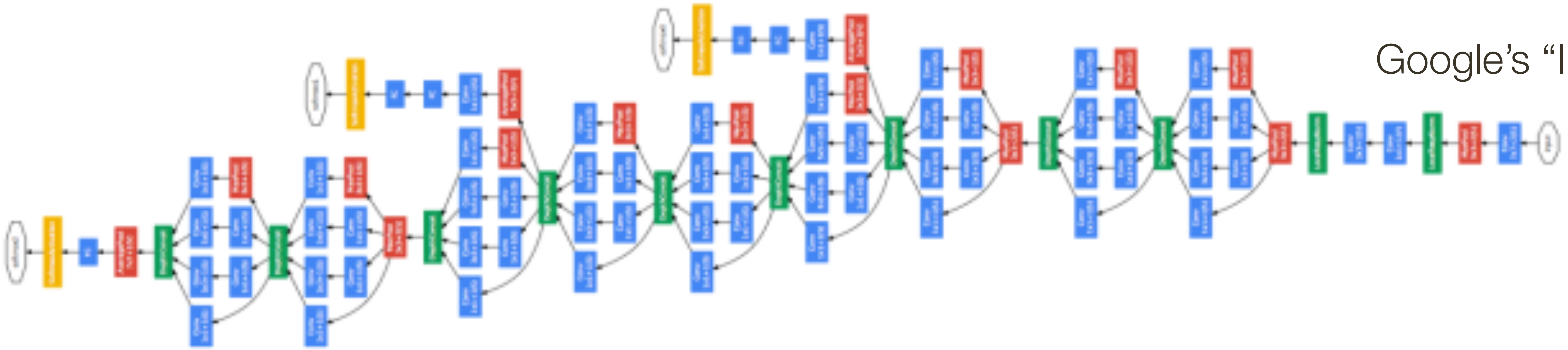
deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

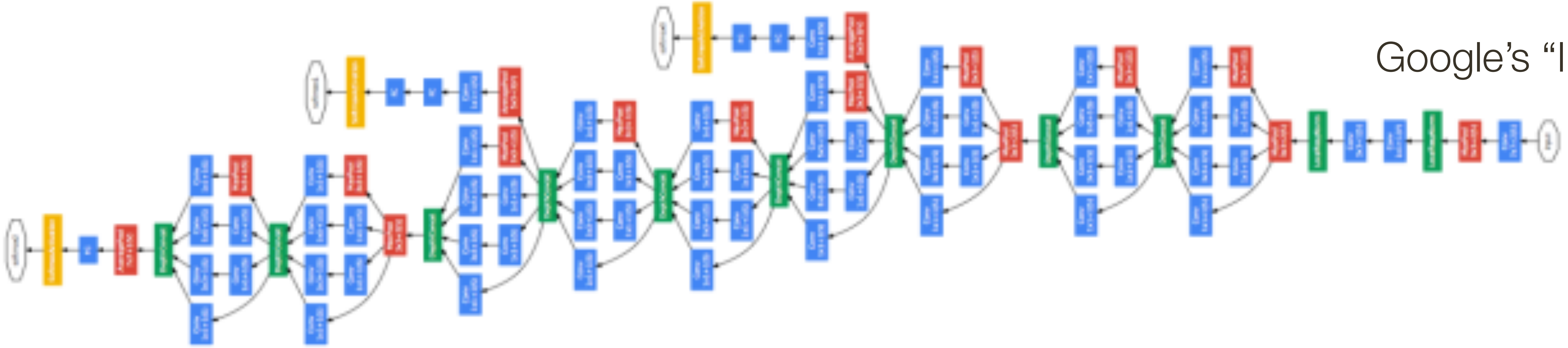
- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

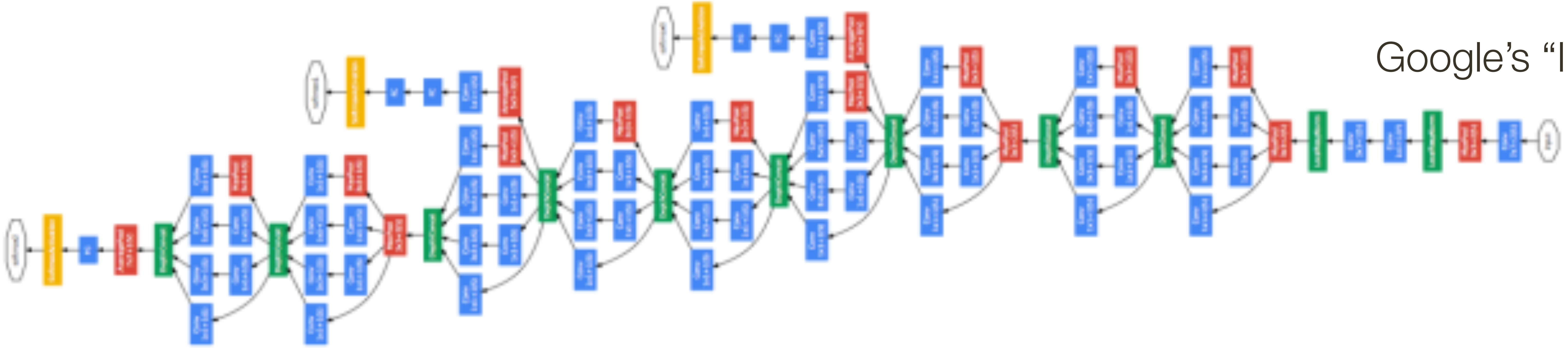
requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

grid search

Multivariate **Regression**

Input:

Value of all stocks at closing of NASDAQ today (3,300 stocks)

Output:

Value of Microsoft, Google, Apple stock at opening tomorrow

Multivariate **Regression**

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Value of all stocks at closing of NASDAQ today (3,300 stocks)

Output: output vector $\mathbf{y} \in \mathbb{R}^m$

Value of Microsoft, Google, Apple stock at opening tomorrow

Multivariate **Regression**

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: output vector $\mathbf{y} \in \mathbb{R}^m$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

with **Tanh** activations: $-\mathbf{1} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Multivariate **Regression**

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: output vector $\mathbf{y} \in \mathbb{R}^m$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}^k$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

with **Tanh** activations: $-\mathbf{1} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Neural Network (output): linear layer

$$\hat{\mathbf{y}} = g(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W} f(\mathbf{x}; \Theta) + \mathbf{b} : \mathbb{R}^k \to \mathbb{R}^m$$

Multivariate **Regression**

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: output vector $\mathbf{y} \in \mathbb{R}^m$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

with **Tanh** activations: $-\mathbf{1} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Neural Network (output): linear layer

$$\hat{\mathbf{y}} = g(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W} f(\mathbf{x}; \Theta) + \mathbf{b} : \mathbb{R}^k \rightarrow \mathbb{R}^m$$

Loss: $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = ||\mathbf{y} - \hat{\mathbf{y}}||^2$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): threshold hidden output (which is a sigmoid)

$$\hat{y} = 1[f(\mathbf{x}; \Theta) > 0.5]$$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): threshold hidden output (which is a sigmoid)

$$\hat{y} = 1[f(\mathbf{x}; \Theta) > 0.5]$$

Problem: Not differentiable, probabilistic interpretation maybe desirable

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): interpret sigmoid output as probability

$$p(y = 1) = f(\mathbf{x}; \Theta)$$

can interpret the score as the log-odds of $y = 1$ (a.k.a. the **logits**)

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): interpret sigmoid output as probability

$$p(y = 1) = f(\mathbf{x}; \Theta)$$

can interpret the score as the log-odds of $y = 1$ (a.k.a. the **logits**)

Loss: similarity between two distributions

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): interpret sigmoid output as probability

$$p(y = 1) = f(\mathbf{x}; \Theta)$$

can interpret the score as the log-odds of $y = 1$ (a.k.a. the **logits**)

Loss: $\mathcal{L}(y, \hat{y}) = -y \log[f(\mathbf{x}; \Theta)] - (1 - y) \log[1 - f(\mathbf{x}; \Theta)]$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): interpret sigmoid output as probability

$$p(y = 1) = f(\mathbf{x}; \Theta)$$

can interpret the score as the log-odds of $y = 1$ (a.k.a. the **logits**)

Loss:

$$\mathcal{L}(y, \hat{y}) = \begin{cases} -log[1 - f(\mathbf{x}; \Theta)] & y = 0 \\ -log[f(\mathbf{x}; \Theta)] & y = 1 \end{cases}$$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): interpret sigmoid output as probability

$$p(y = 1) = f(\mathbf{x}; \Theta)$$

Minimizing this **loss** is the same as maximizing **log likelihood** of data

Loss:

$$\mathcal{L}(y, \hat{y}) = \begin{cases} -log[1 - f(\mathbf{x}; \Theta)] & y = 0 \\ -log[f(\mathbf{x}; \Theta)] & y = 1 \end{cases}$$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}^k$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Neural Network (output): linear layer with one neuron and sigmoid activation

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0, 1\}^m$

(**one-hot** encoding)

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0, 1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0, 1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Neural Network (output): **softmax** function, where probability of class k is:

$$p(\mathbf{y}_k = 1) = \frac{\exp\left[f(\mathbf{x}; \Theta)_i\right]}{\sum_{j=1}^{C} \exp\left[f(\mathbf{x}; \Theta)_j\right]}$$

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0,1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x};\Theta) : \mathbb{R}^n \to \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x};\Theta)$

Neural Network (output): **softmax** function, where probability of class k is:

$$p(\mathbf{y}_k = 1) = \frac{\exp\left[f(\mathbf{x};\Theta)_i\right]}{\sum_{j=1}^{C} \exp\left[f(\mathbf{x};\Theta)_j\right]}$$

Loss: $\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = H(\mathbf{y},\hat{\mathbf{y}}) = -\sum_i \mathbf{y}_i \log \hat{\mathbf{y}}_i$

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0,1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x};\Theta) : \mathbb{R}^n \to \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x};\Theta)$

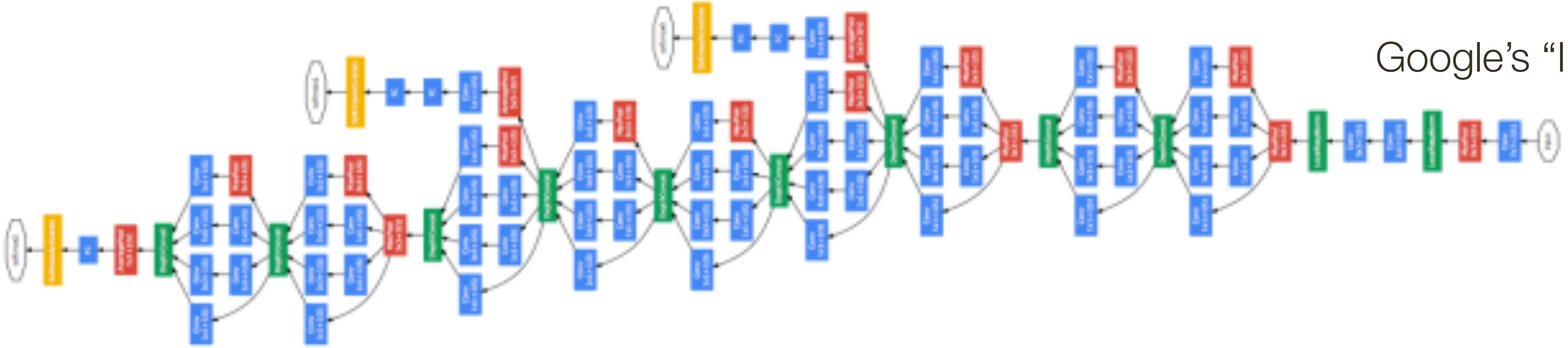
Neural Network (output): **softmax** function, where probability of class k is:

$$p(\mathbf{y}_k = 1) = \frac{\exp\left[f(\mathbf{x};\Theta)_i\right]}{\sum_{j=1}^{C} \exp\left[f(\mathbf{x};\Theta)_j\right]}$$

Loss: $$\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = H(\mathbf{y},\hat{\mathbf{y}}) = -\sum_i \mathbf{y}_i \log \hat{\mathbf{y}}_i = -\log \hat{\mathbf{y}}_i$$

Special case for multi-class single label

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

Specification of neural architecture will define a **computational** graph.

Training

Initialize parameters of all layers

For a fixed number of iterations or until convergence

— Form **mini-batch** of examples (randomly chosen from a training dataset)

— Compute **forward** pass to make predictions for every example and compute the loss (this involves recursively calling forward() for each intermediate layer along computational graph)

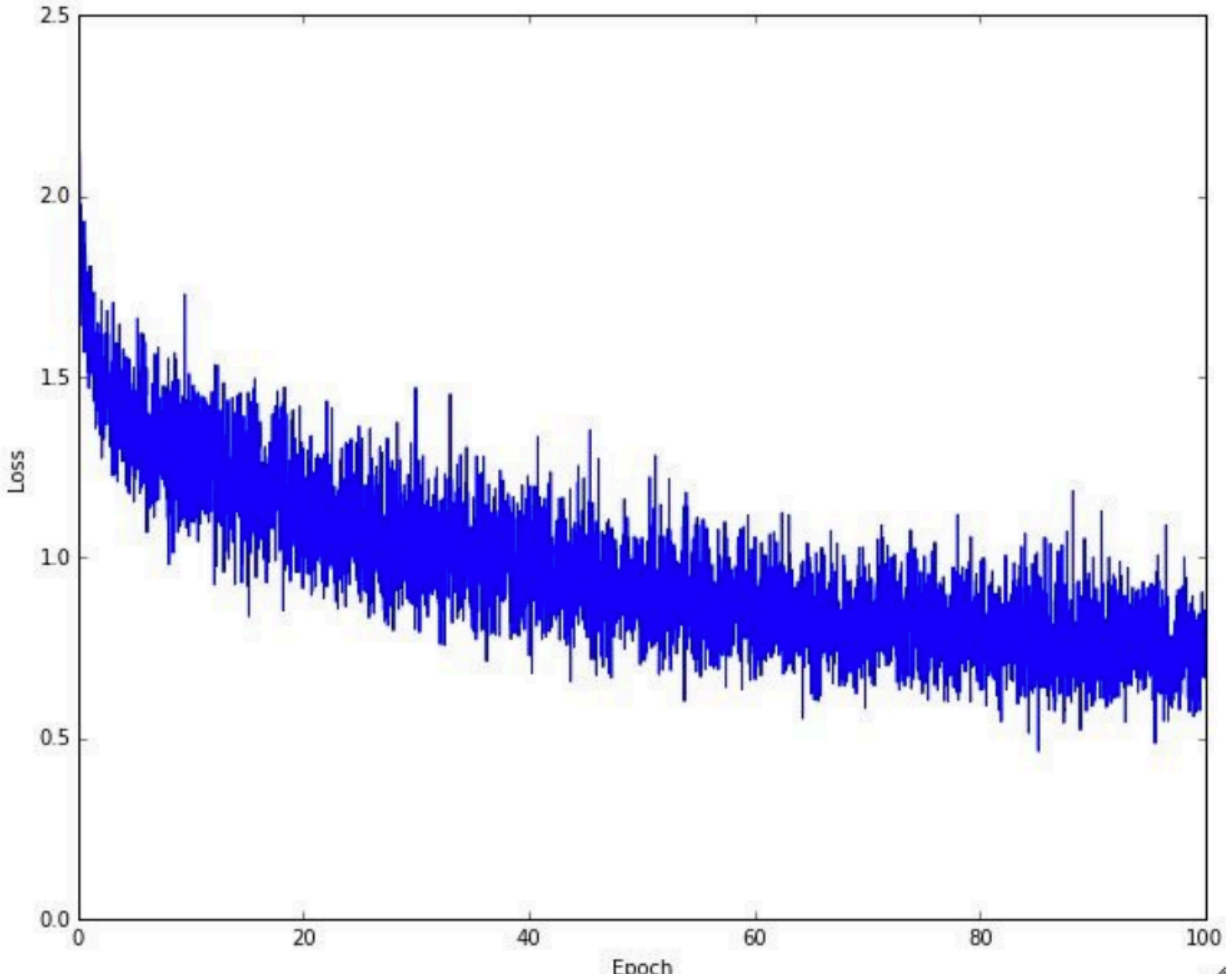
— Compute **backwards** pass to compute the gradient of the loss with respect to each parameter for each example (involves traversing computational graph in reverse order calling backward() on intermediate nodes and composing intermediate gradients — chain rule)

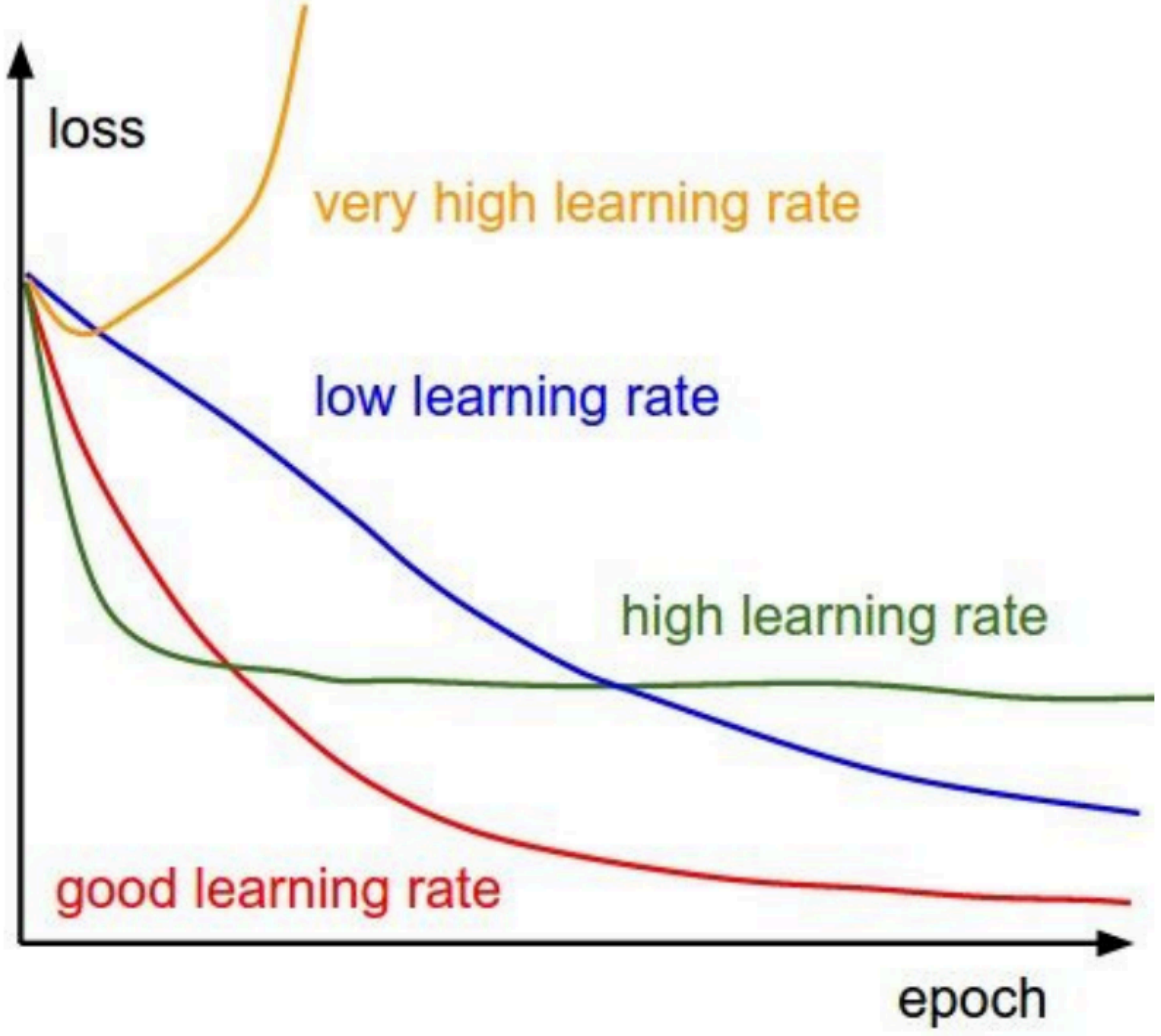
— **Update parameters** of all layers, by taking a step in the negative **average** gradient direction (computed over all examples in the mini-batch)

Inference / Prediction

Compute **forward** pass with **optimized** parameters on test examples

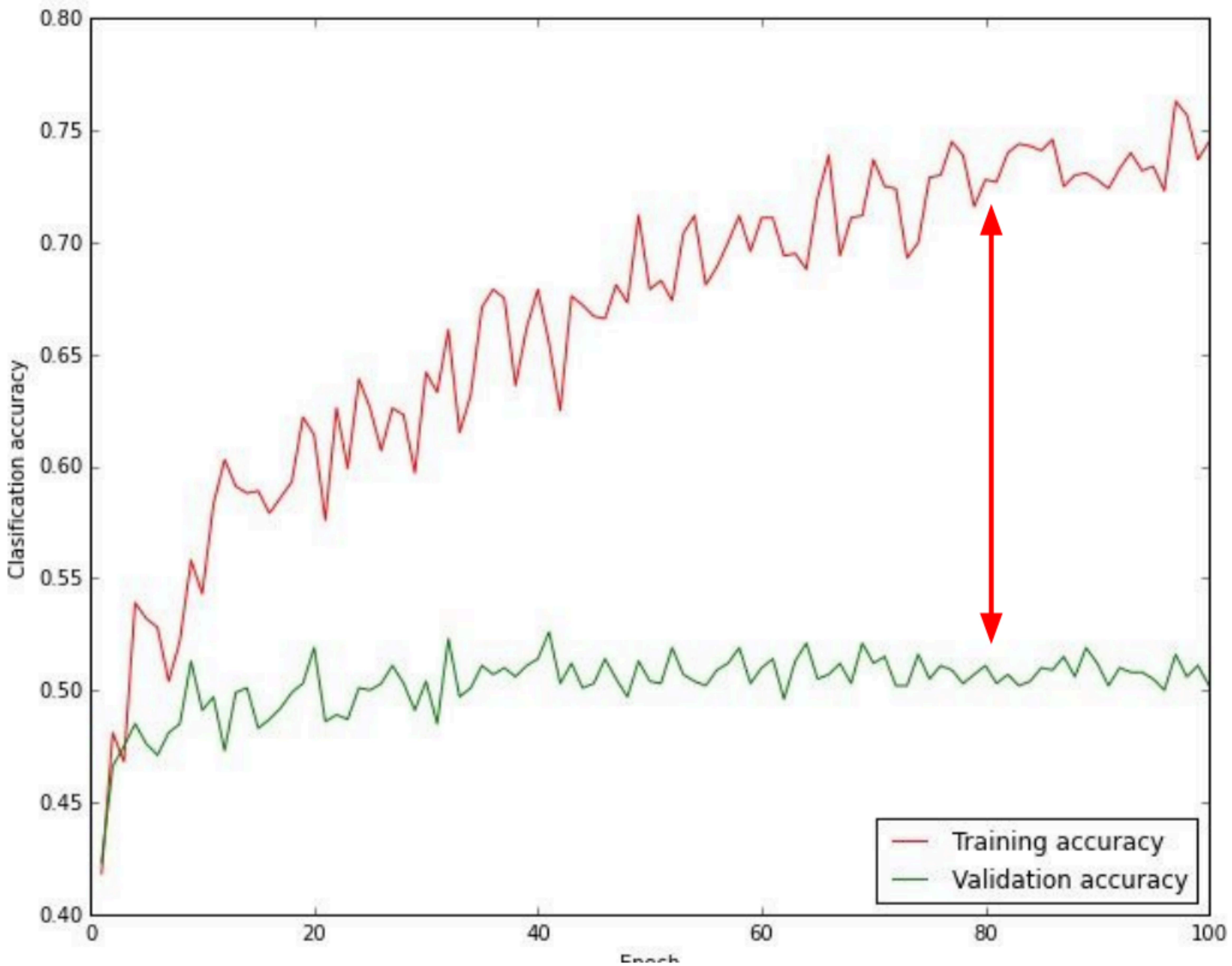
Monitoring Learning: Visualizing the (training) loss





* slide from Li, Karpathy, Johnson's **CS231n at Stanford**

Monitoring Learning: Visualizing the (training) loss



Big gap = overfitting

Solution: increase regularization

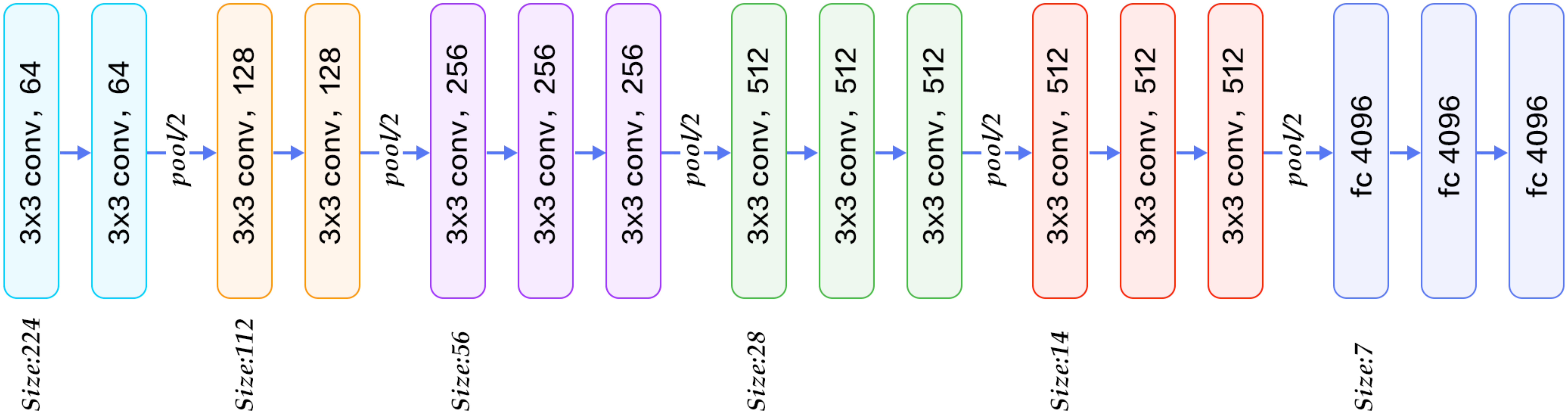
No gap = undercutting

Solution: increase model capacity

Small gap = **ideal**

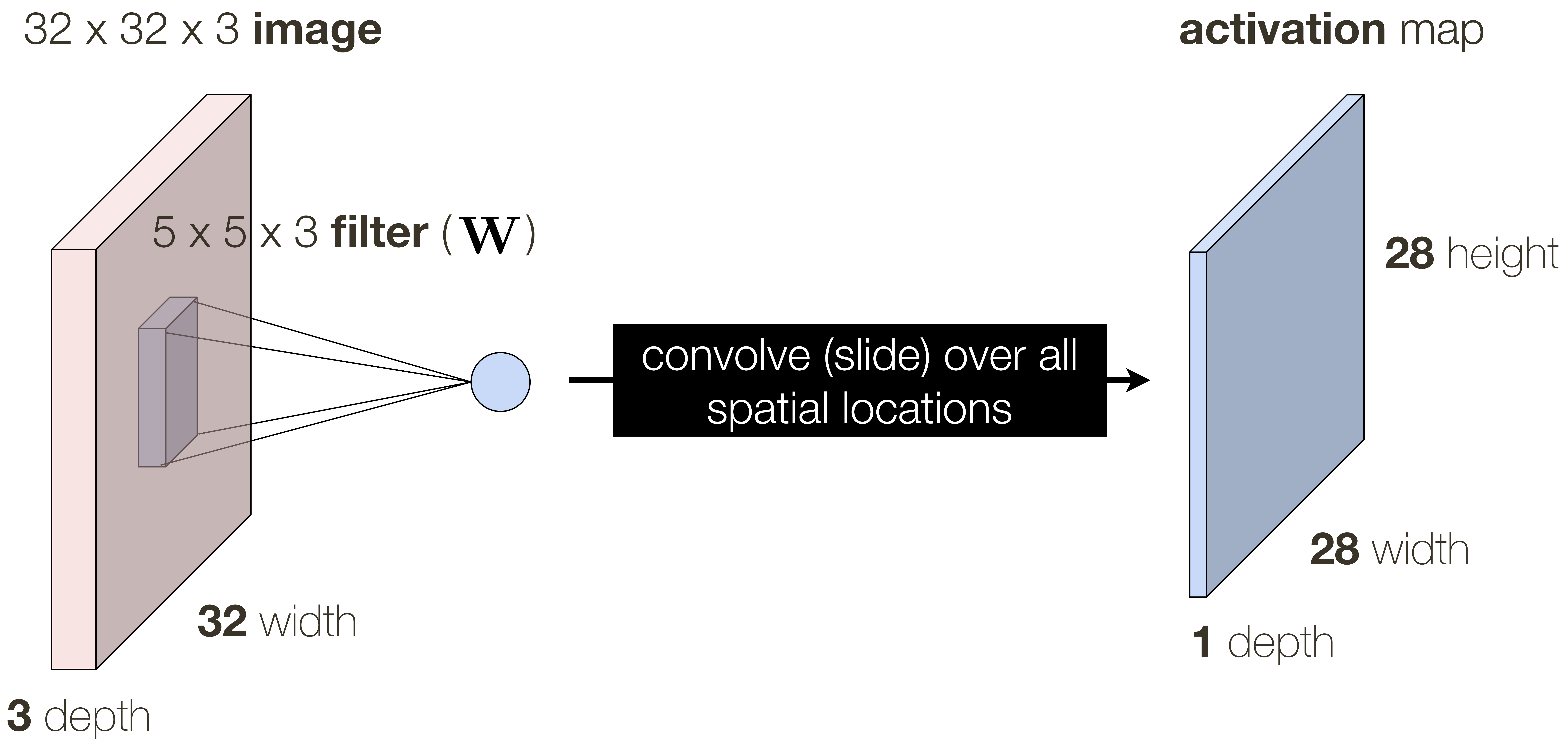
* slide from Li, Karpathy, Johnson's **CS231n at Stanford**

Convolutional Neural Networks



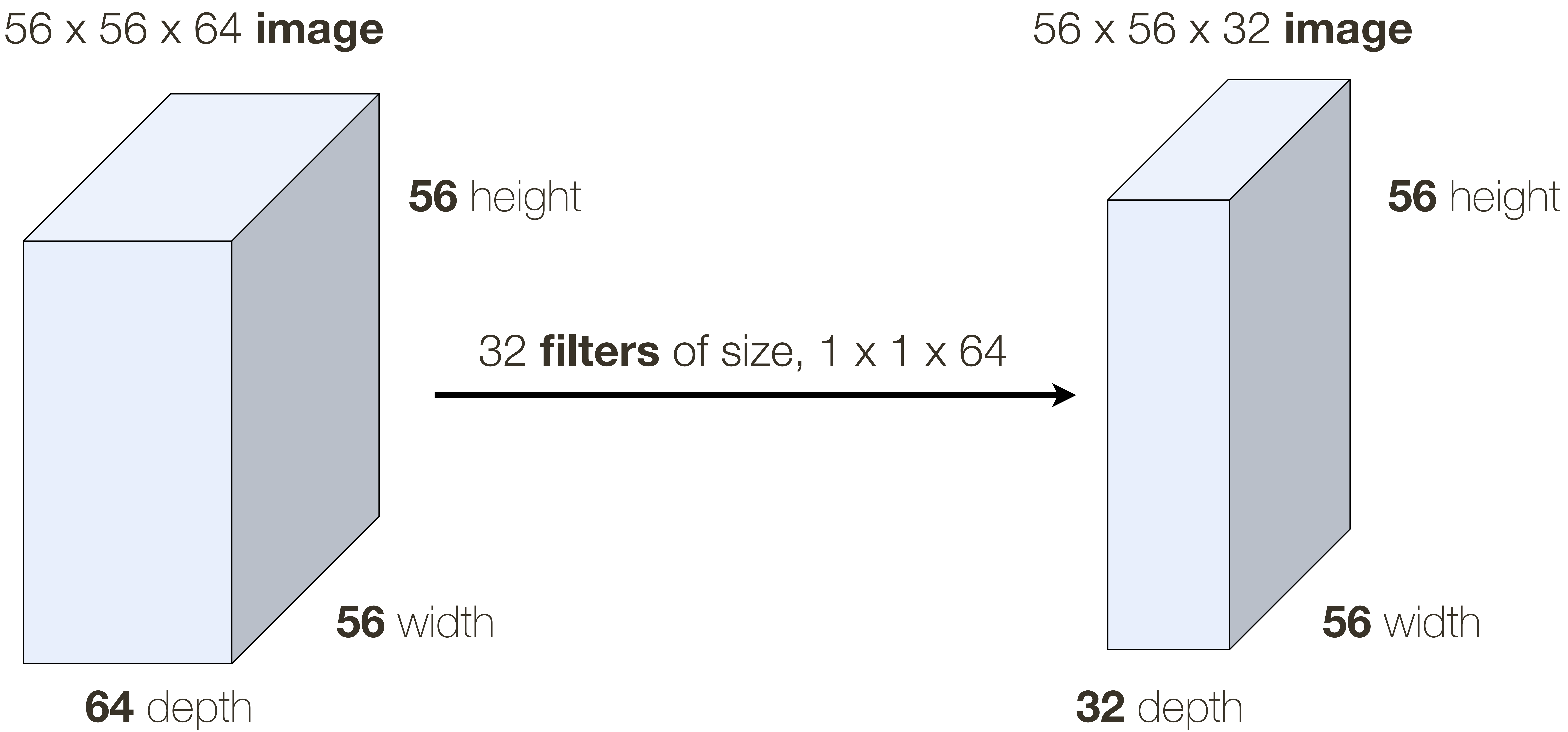
VGG-16 Network

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



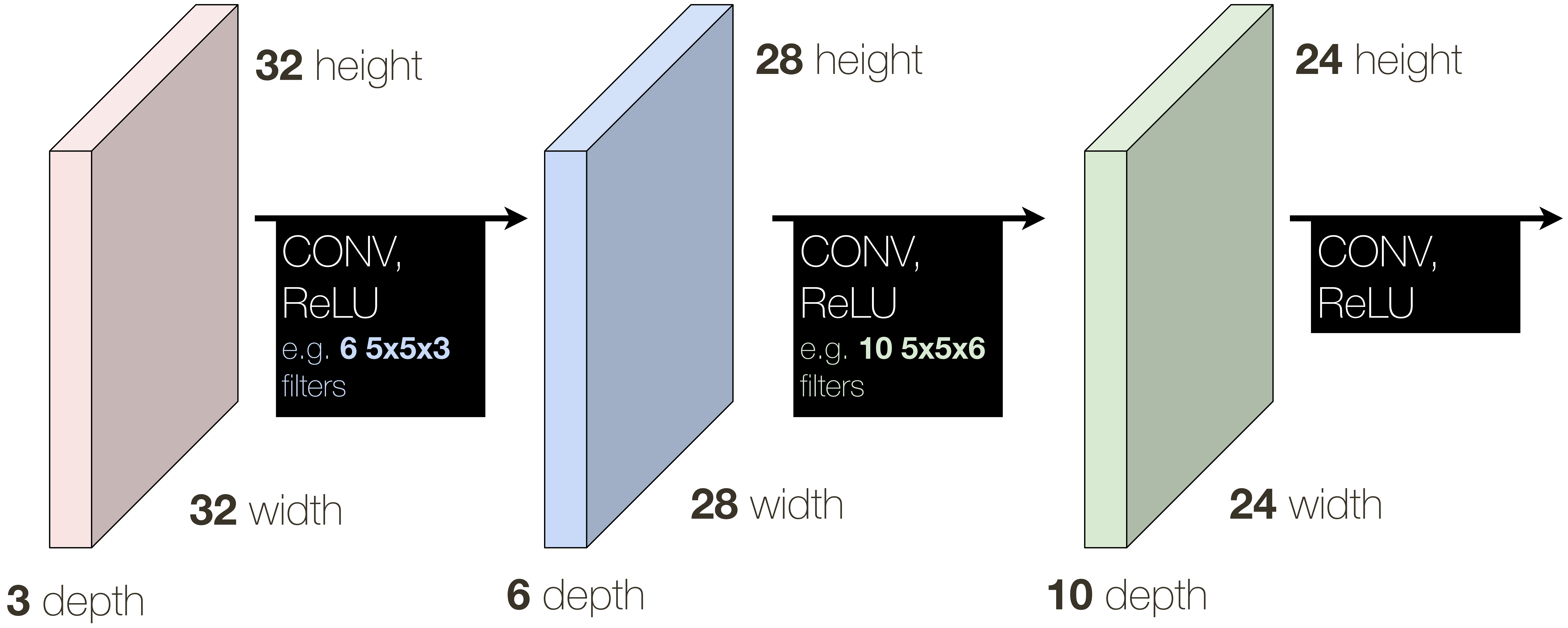
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: **1x1 convolutions**



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

Convolutional Neural Network (ConvNet)



* 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: $W_i \times H_i \times D_i$ (for mini-batch $N \times W_i \times H_i \times D_i$)

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$ (for mini-batch $N \times W_i \times H_i \times D_i$)

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: $W_i \times H_i \times D_i$ (for mini-batch $N \times W_i \times H_i \times D_i$)

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: $W_o \times H_o \times D_o$ (for mini-batch $N \times W_o \times H_o \times D_o$)

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$ (for mini-batch $N \times W_i \times H_i \times D_i$)

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: $W_o \times H_o \times D_o$ (for mini-batch $N \times W_o \times H_o \times D_o$)

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

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$ (for mini-batch $N \times W_i \times H_i \times D_i$)

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 $S \in \{1, 2\}$)

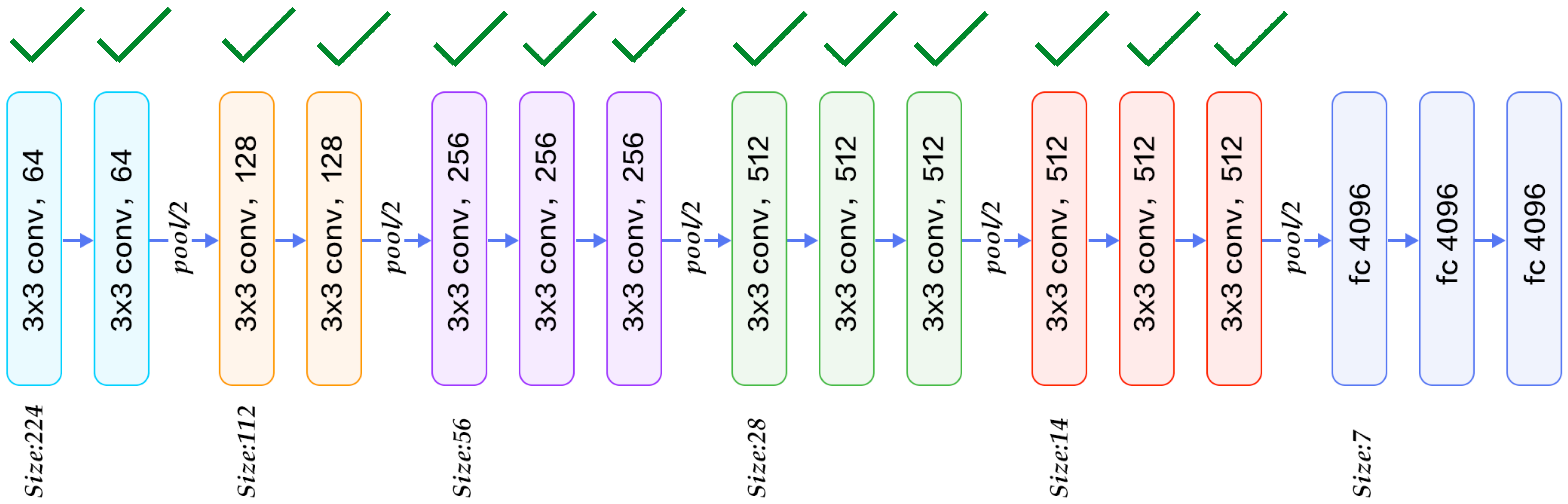
— Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$ (for mini-batch $N \times W_o \times H_o \times D_o$)

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

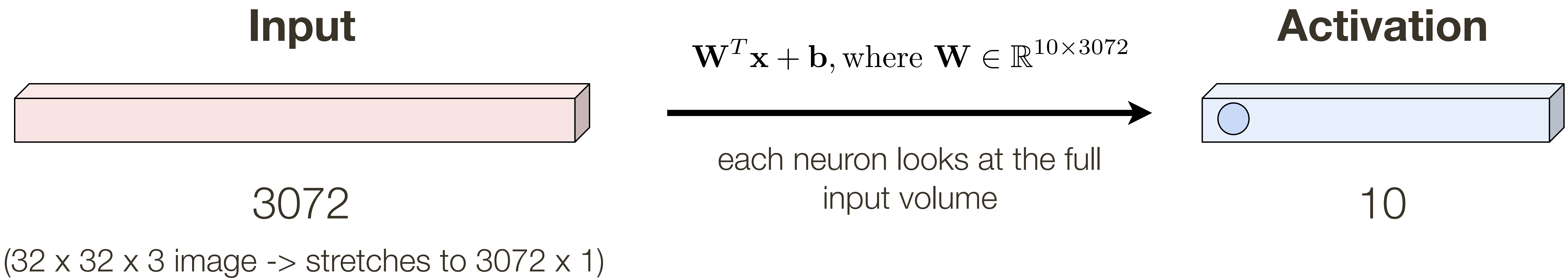
Number of total learnable parameters: $(F \times F \times D_i) \times K + K$

Convolutional Neural Networks



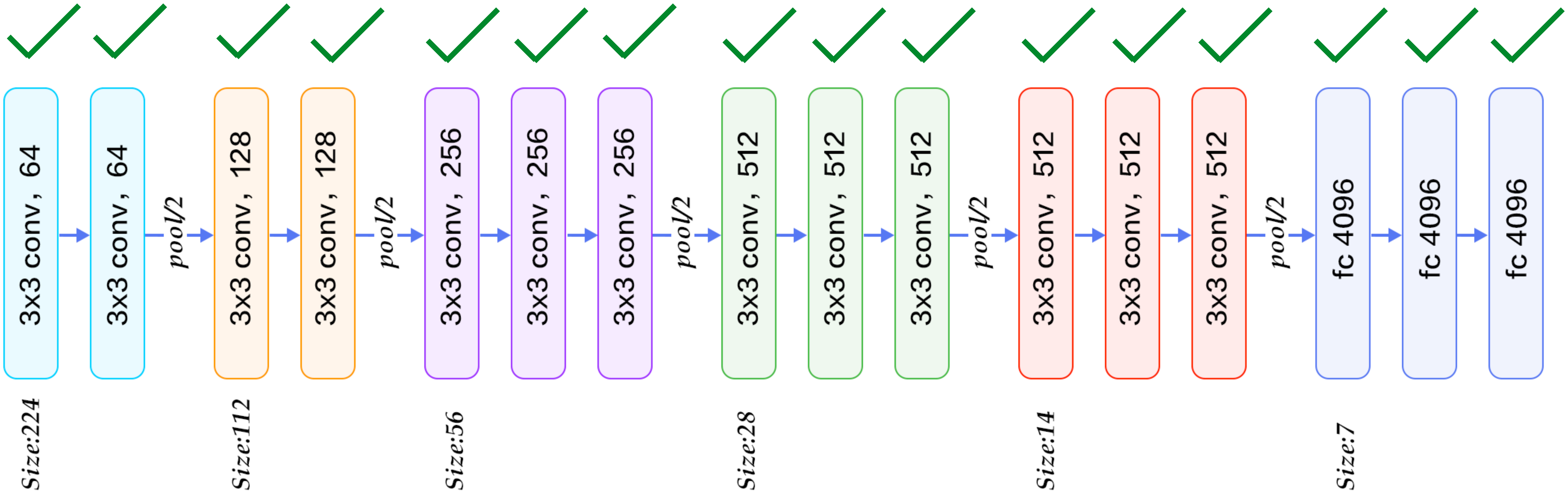
VGG-16 Network

CNNs: Reminder Fully Connected Layers



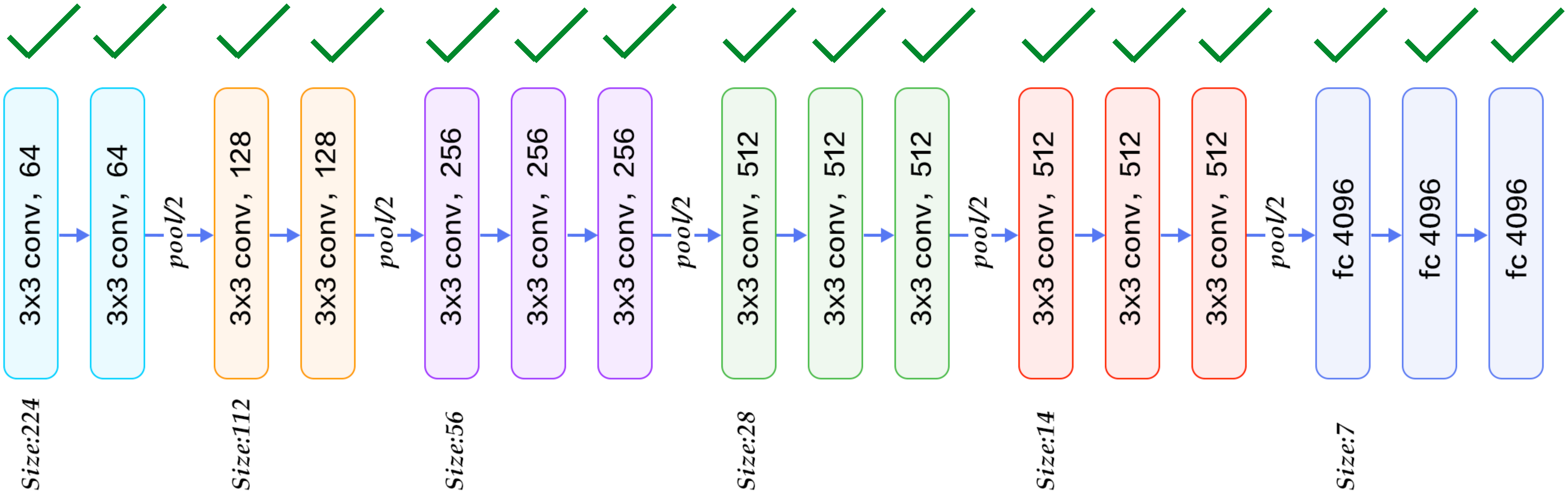
* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Networks



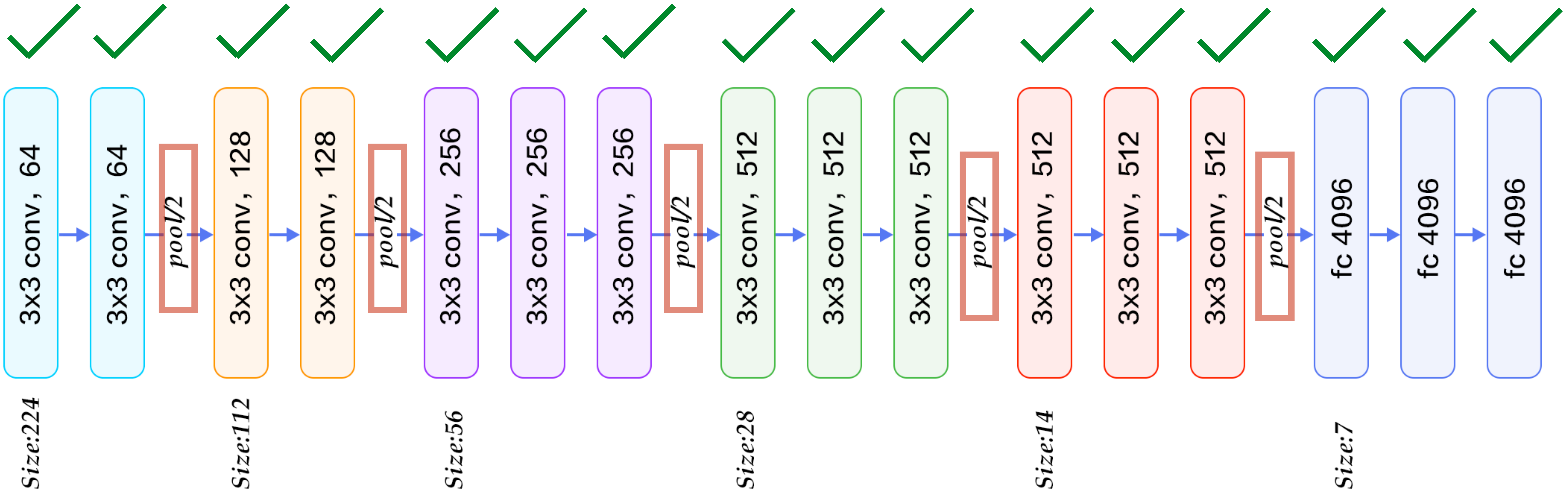
VGG-16 Network

Convolutional Neural Networks



VGG-16 Network

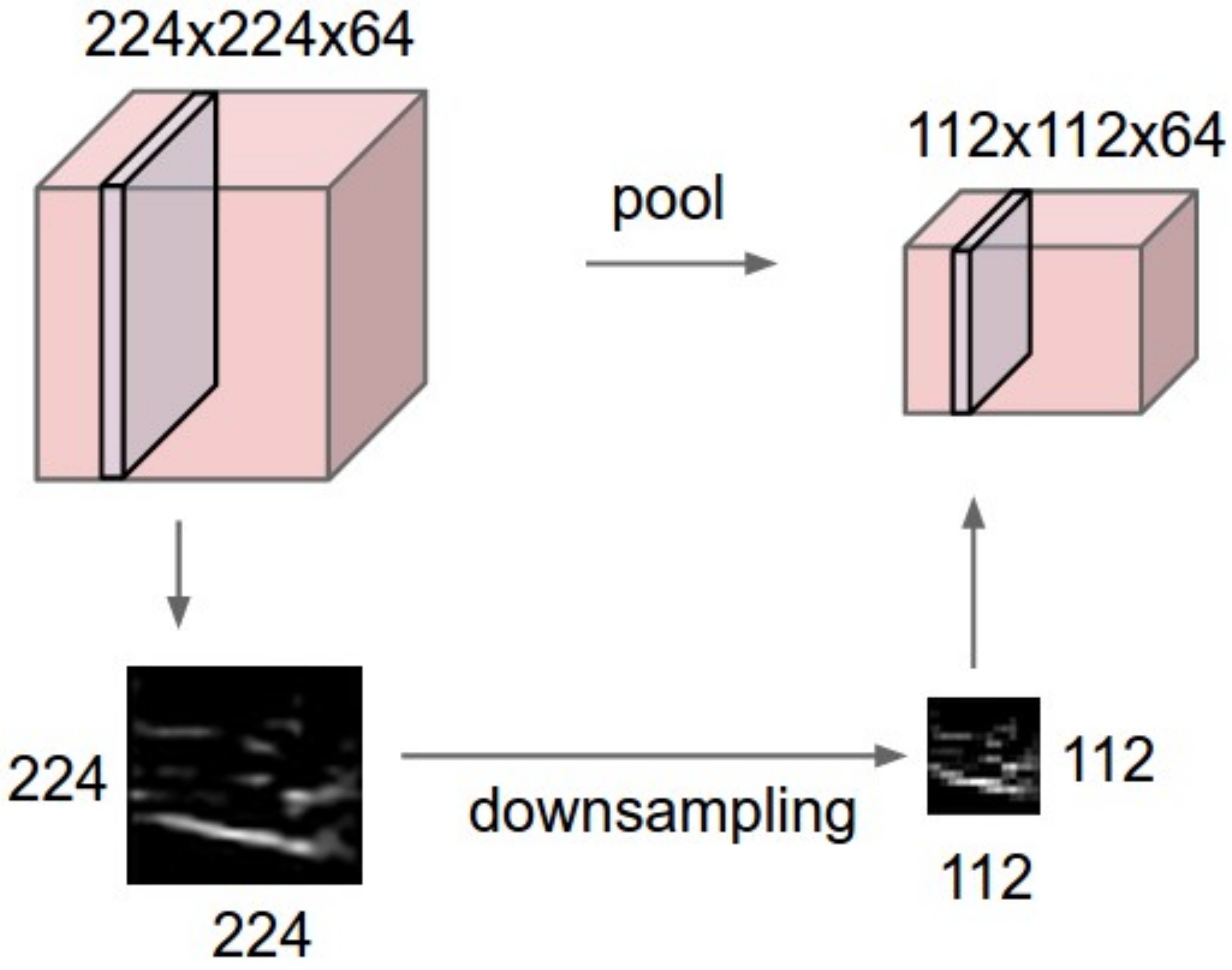
Convolutional Neural Networks



VGG-16 Network

Pooling Layer

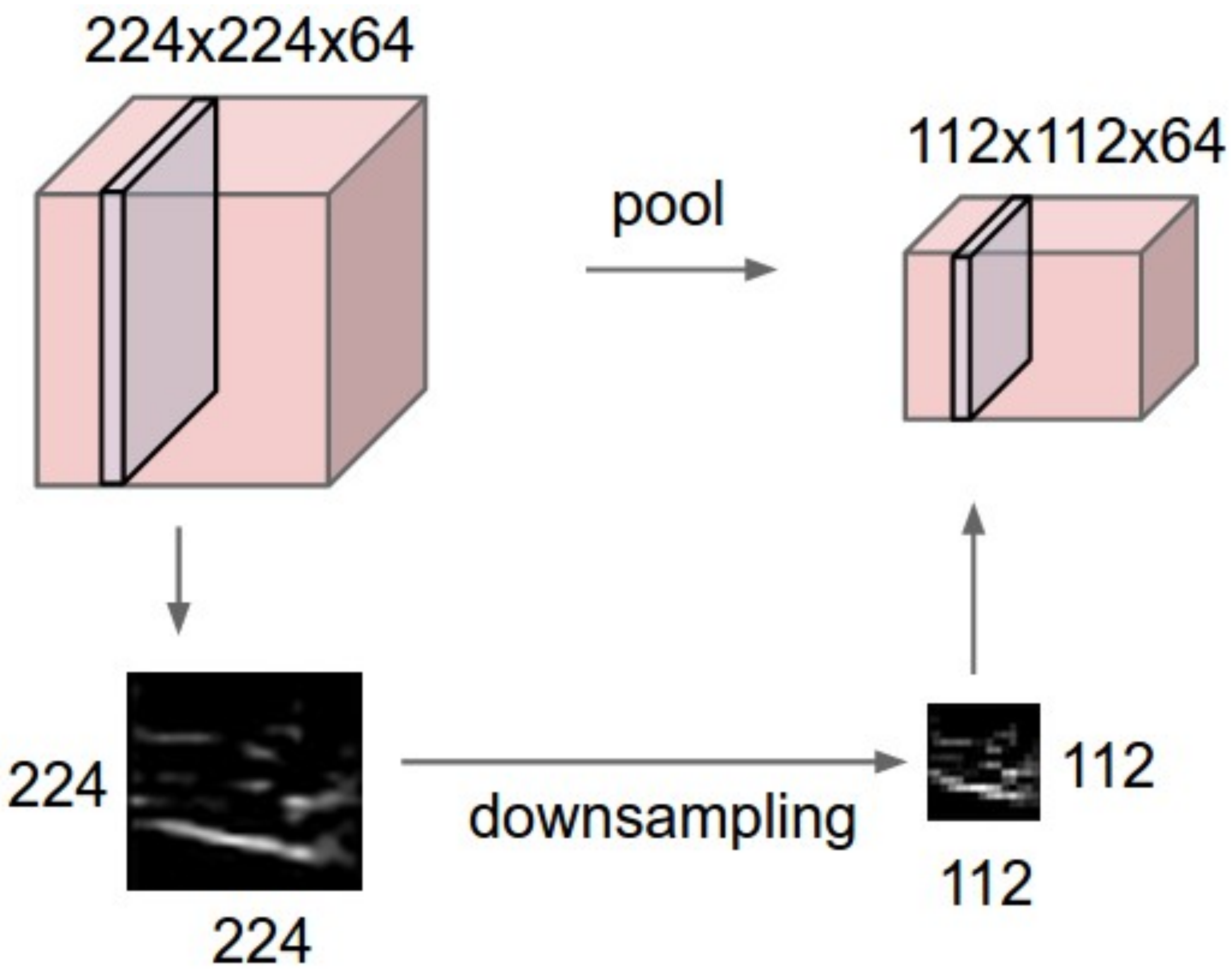
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



* 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

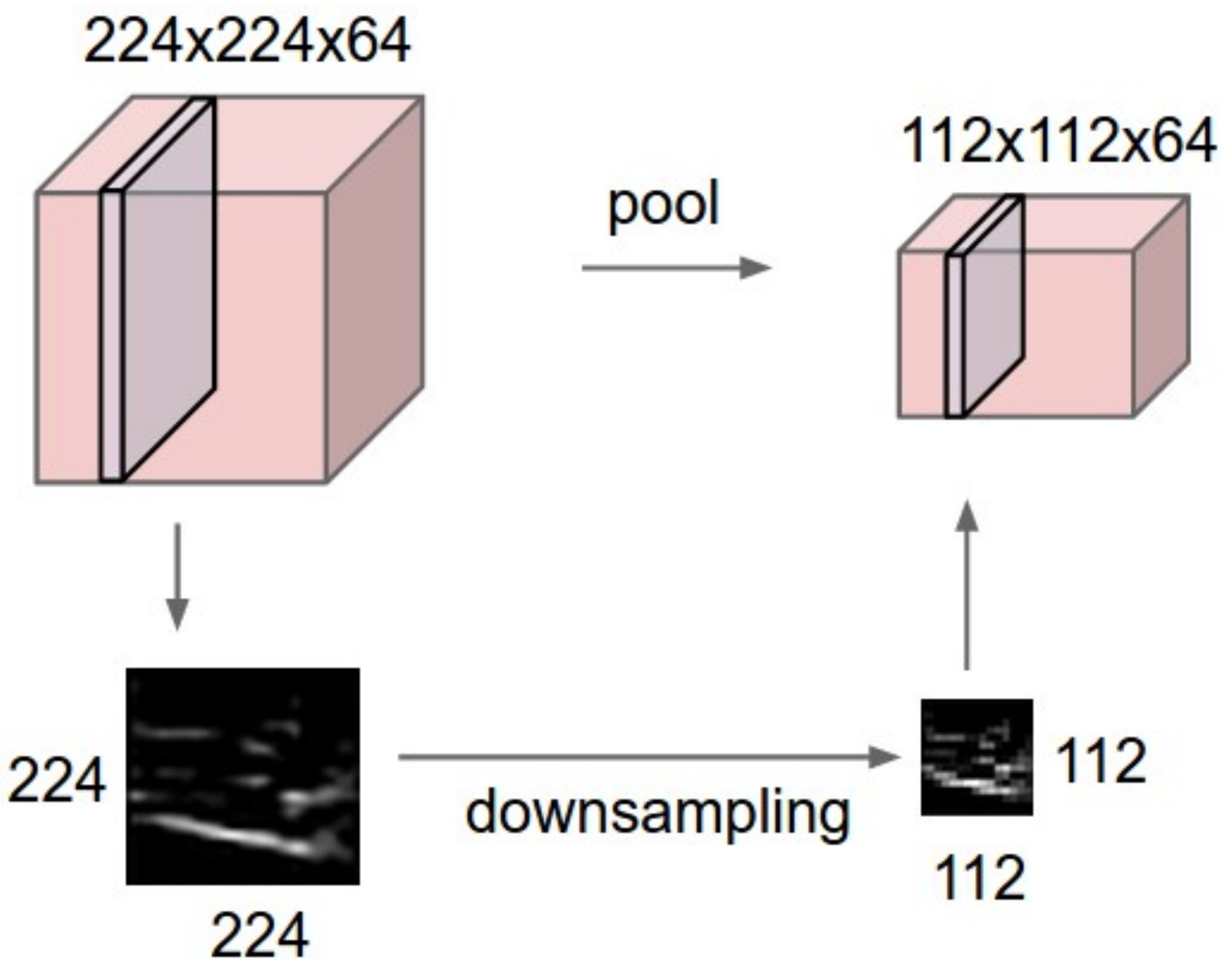


How many **parameters**?

* 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

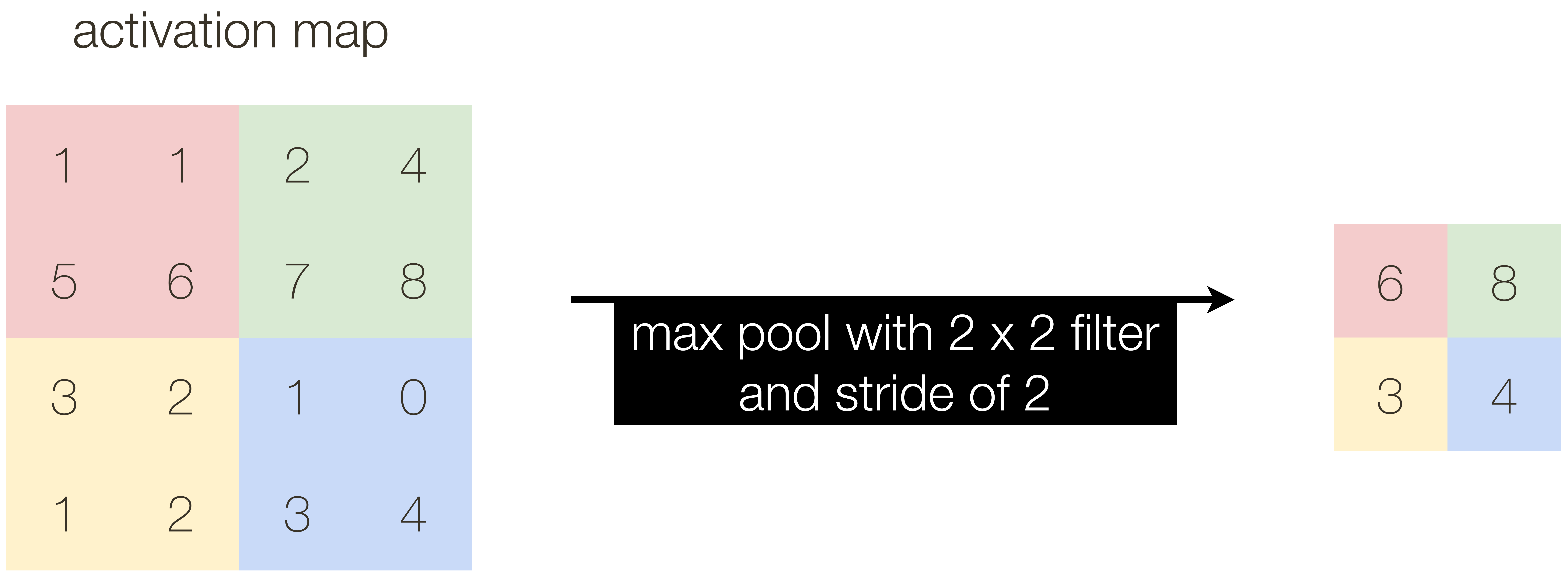


How many **parameters**?

None!

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

Max **Pooling**


activation map
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2 x 2 filter
and stride of 2
6 8
3 4

Average **Pooling**

activation map

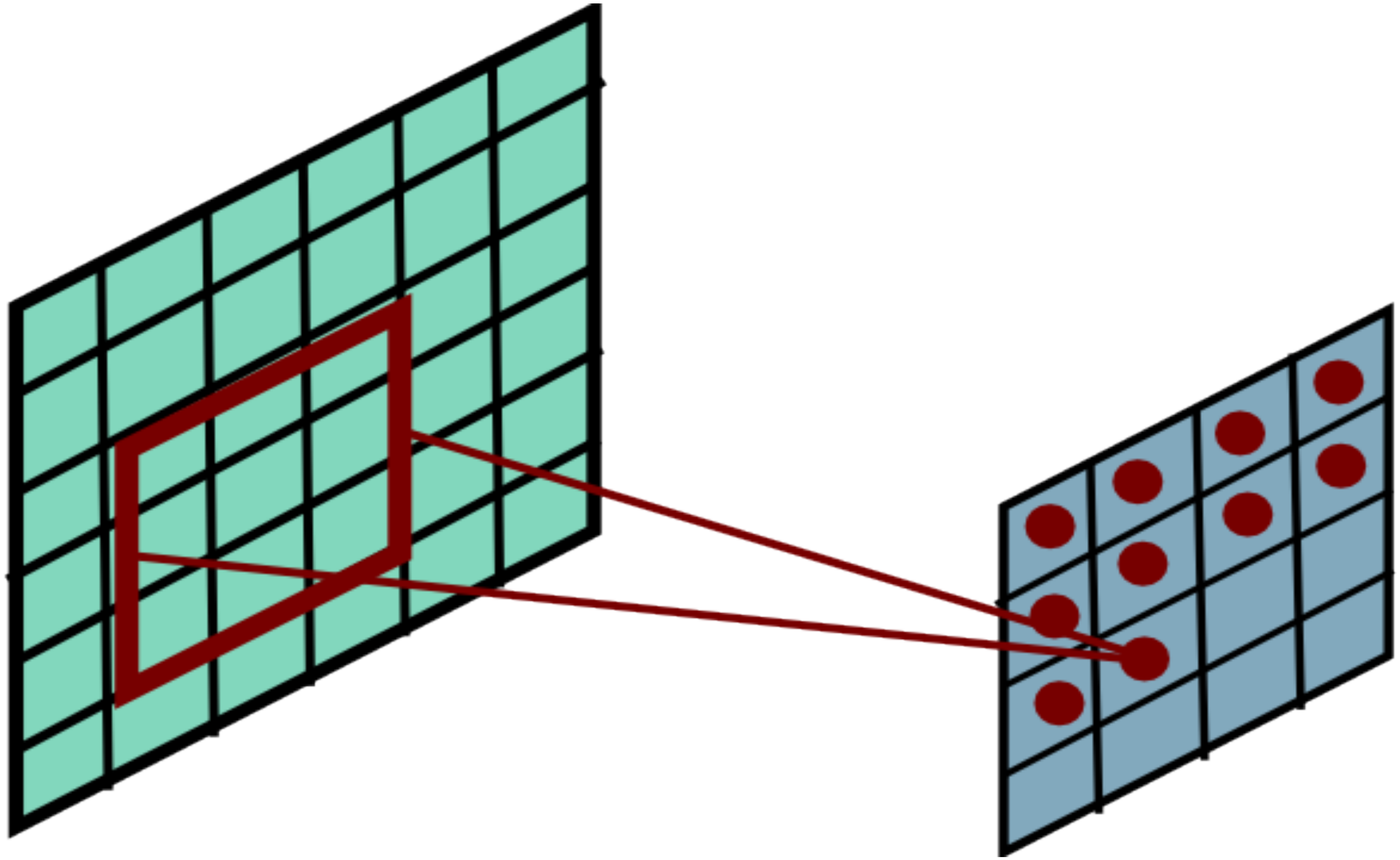
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

avg pool with 2 x 2 filter
and stride of 2

3.25	5.25
2	2

Pooling Layer **Receptive Field**

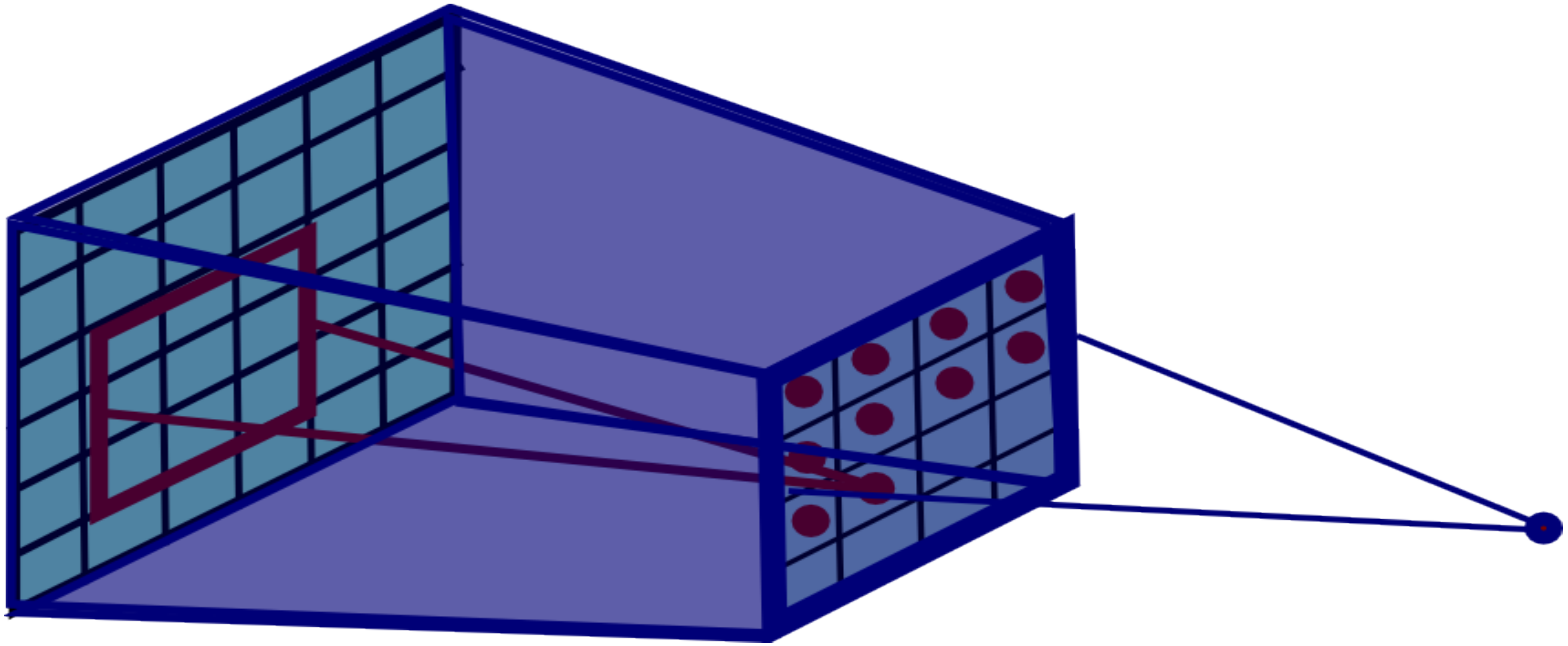
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: **(P+K-1)x(P+K-1)**



* 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:

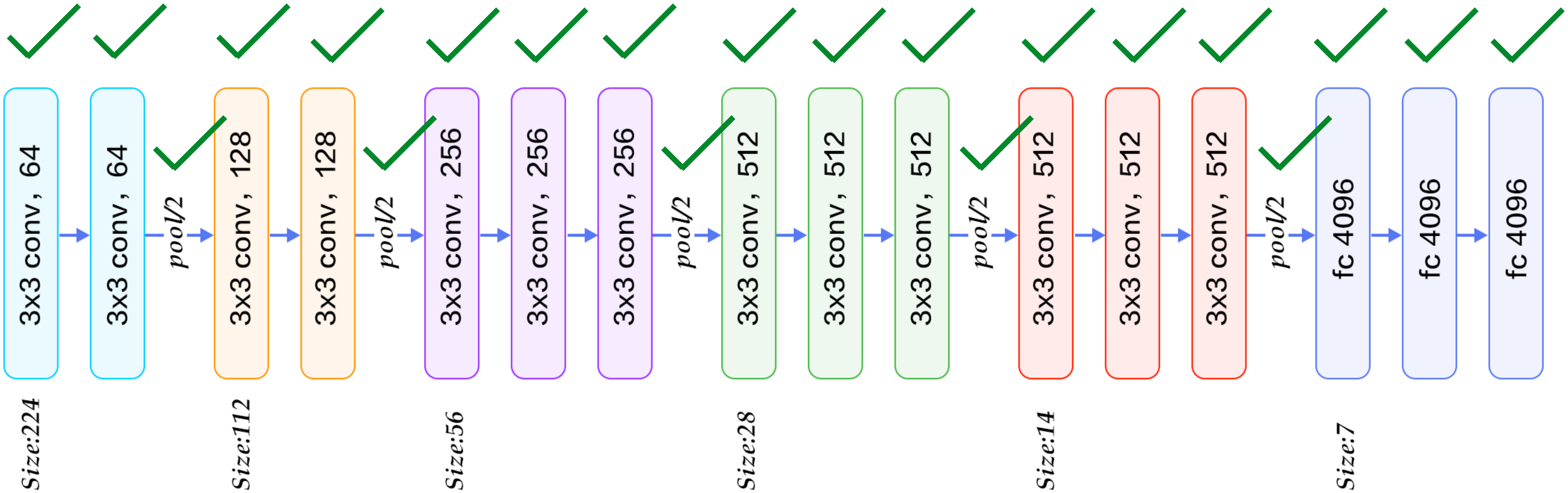
- Spatial extent of filters: K
- Stride of application: F

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

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

Number of total learnable parameters: 0

Convolutional Neural Networks



VGG-16 Network