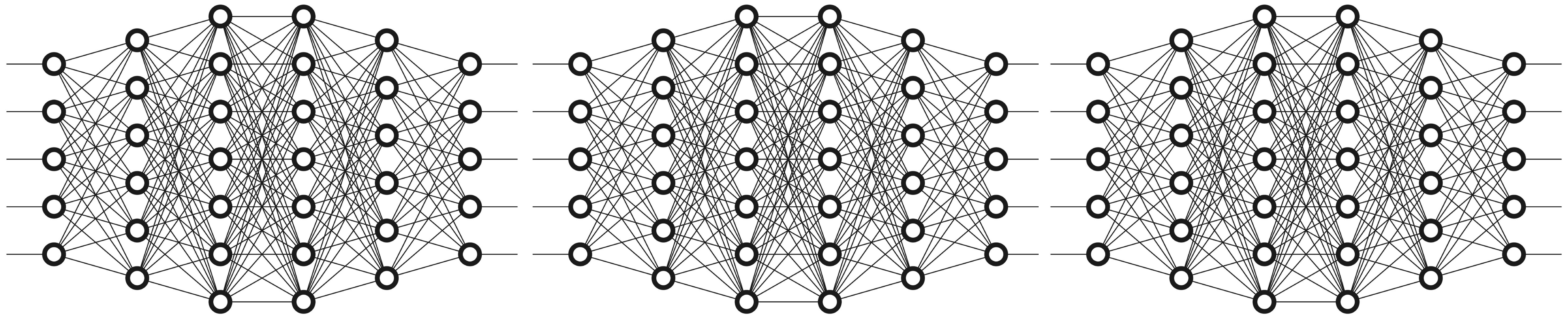
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 23: Neural Networks (cont), CNNs

Menu for Today (**April 2, 2020**)

Topics:

- Backpropagation
- Convolutional Layers
- Pooling Layer
- R-CNN

Redings:

- **Today's** Lecture: N/A
- **Next** Lecture: N/A

Reminders:

- **Assignment 6**: Deep Learning due **Tuesday, April 7th**

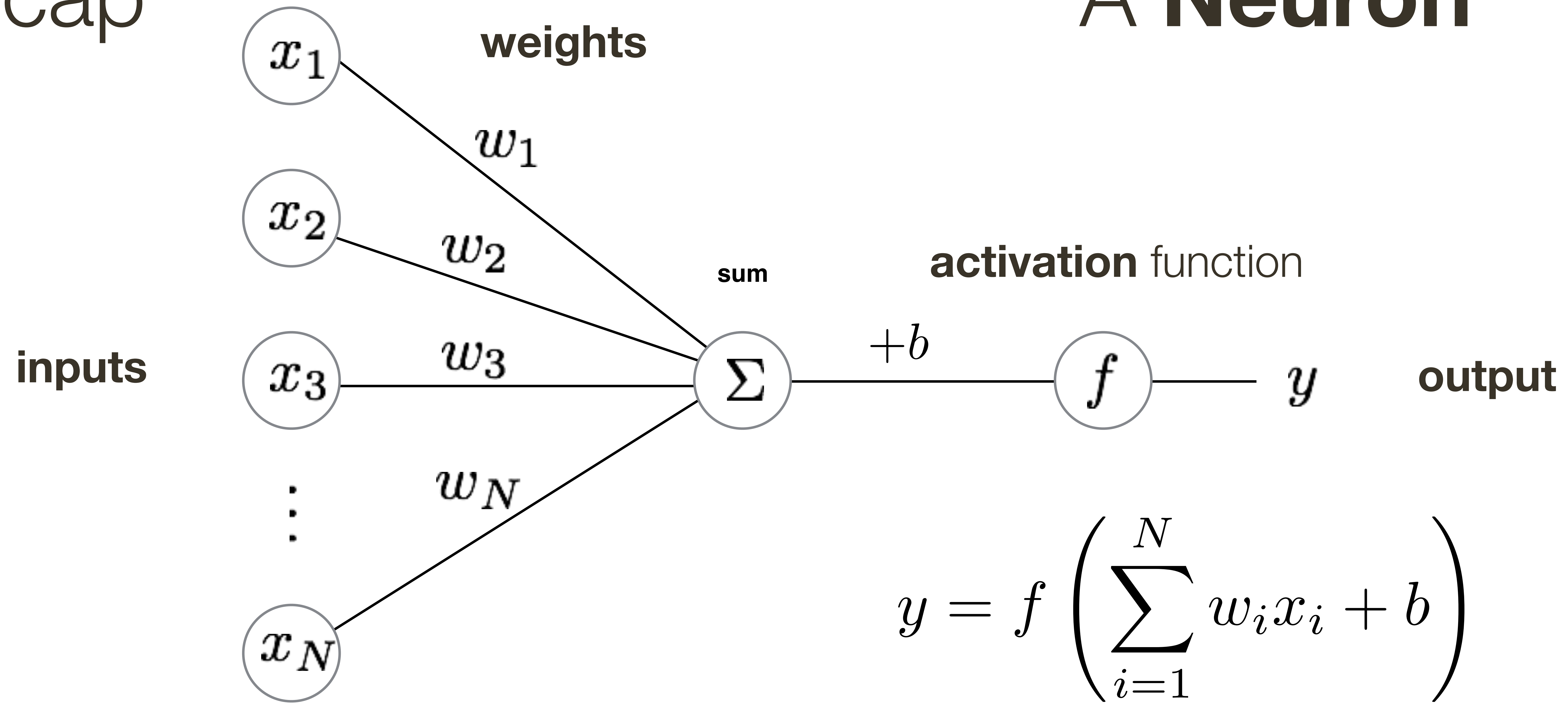
Please fill out

Student Evaluations

(on Canvas)

Lecture 22: Re-cap

A **Neuron**



$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

— The basic unit of computation in a neural network is a neuron.

— A neuron accepts some number of input signals, computes their weighted sum, and applies an **activation function** (or **non-linearity**) to the sum.

— Common activation functions include sigmoid and rectified linear unit (ReLU)

Lecture 22: Re-cap

Neural Network

A neural network comprises neurons connected in an acyclic graph

The outputs of neurons can become inputs to other neurons

Neural networks typically contain multiple layers of neurons

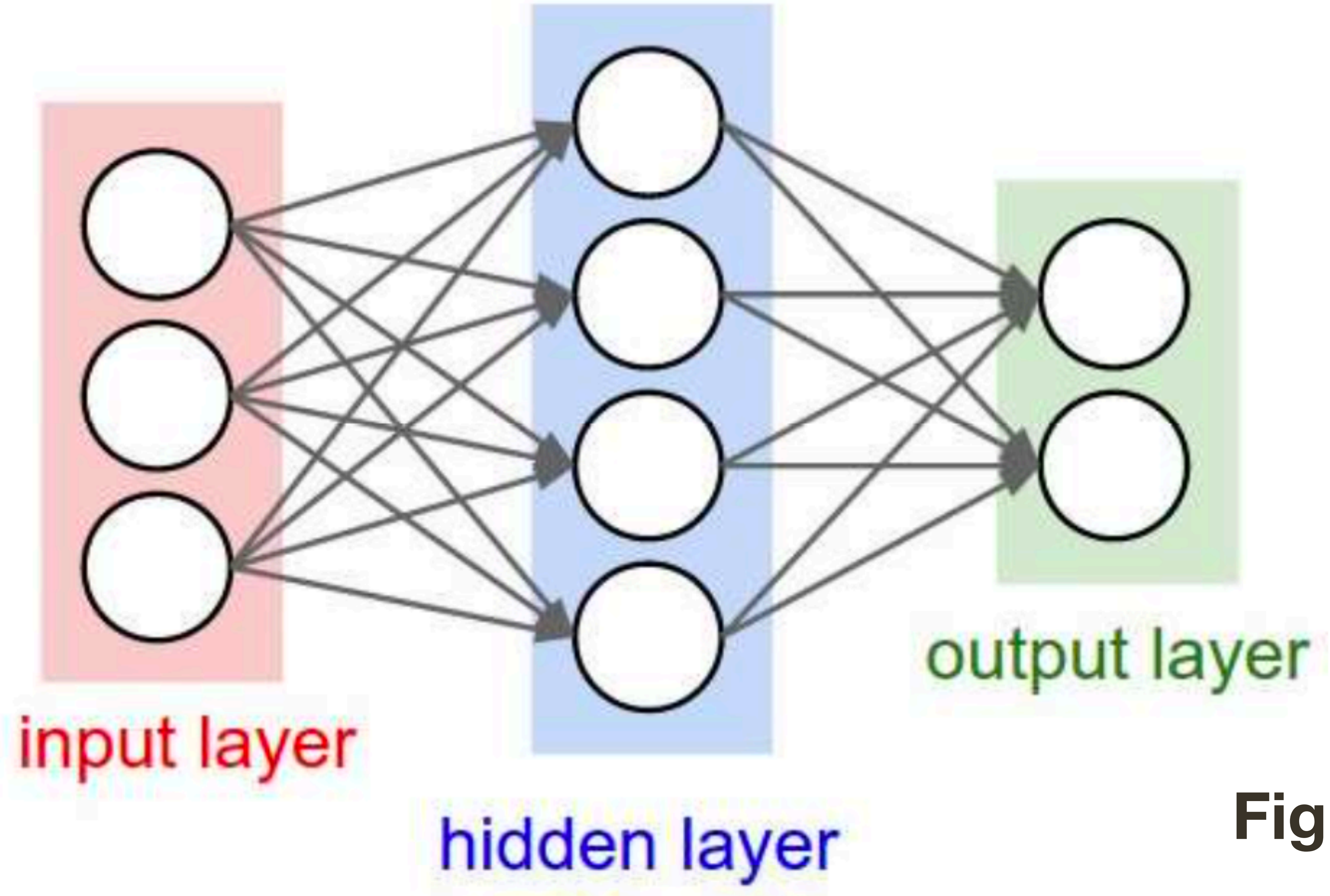


Figure credit: Fei-Fei and Karpathy

Example of a neural network with three inputs, a single hidden layer of four neurons, and an output layer of two neurons

Lecture 22: Re-cap **Neural** Network

Note: each neuron will have its own vector of weights and a bias, its easier to think of all neurons in a layer as a single entity with a matrix of weights (size = number of inputs x number of neurons) and a vector of biases (size = number of neurons)

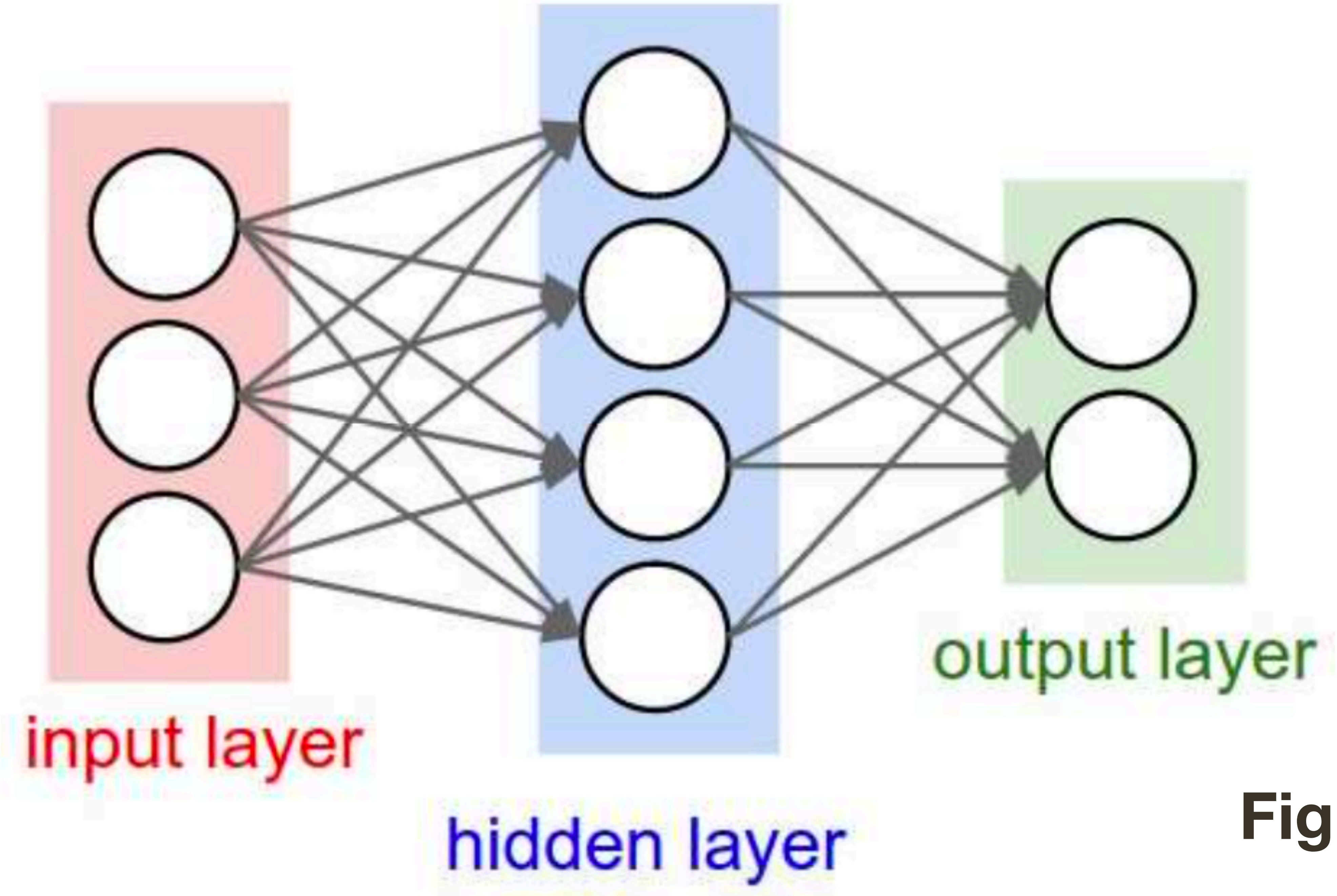


Figure credit: Fei-Fei and Karpathy

Lecture 22: Re-cap

Neural Network

Note: each neuron will have its own vector of weights and a bias, its easier to think of all neurons in a layer as a single entity with a matrix of weights (size = number of inputs x number of neurons) and a vector of biases (size = number of neurons)

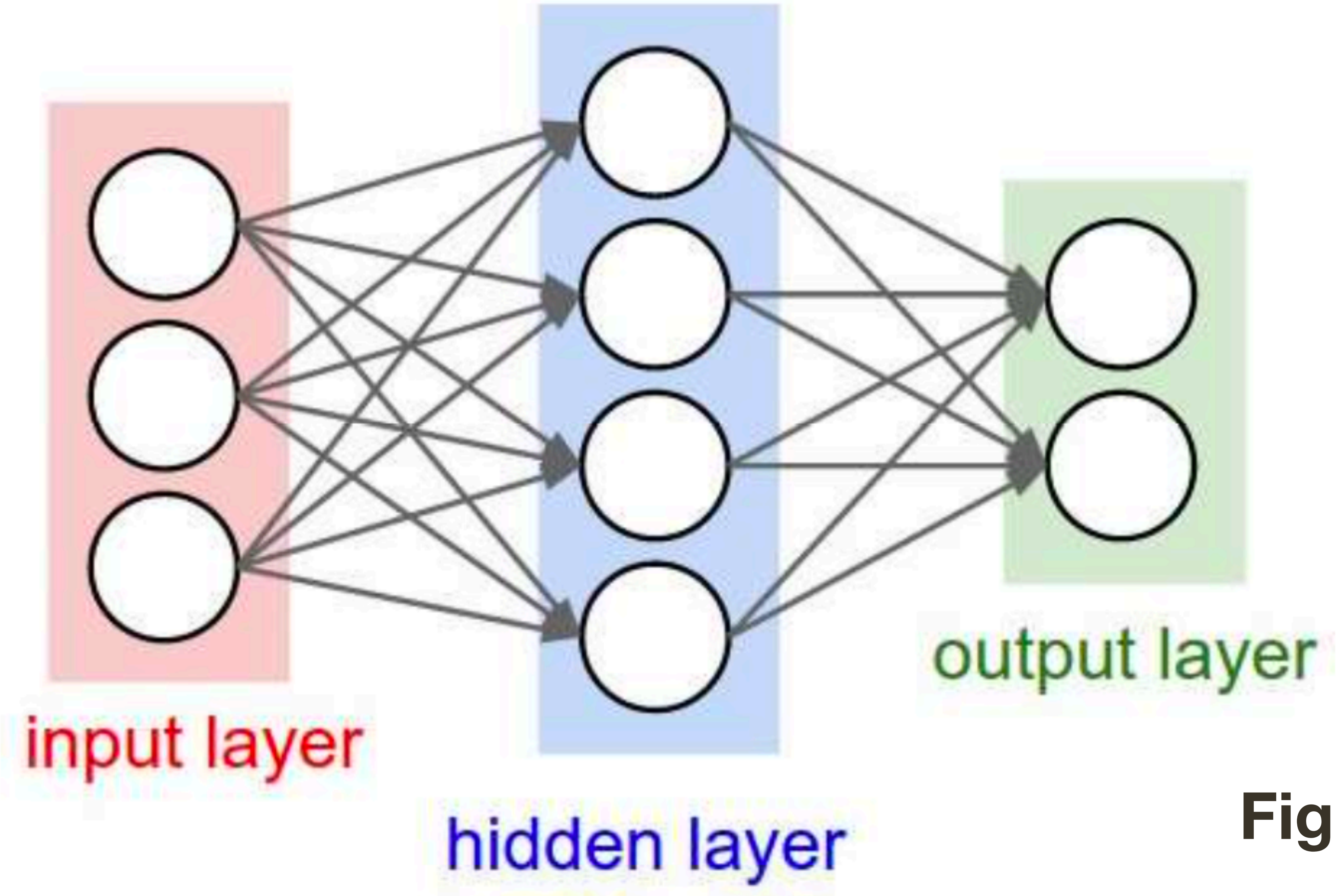
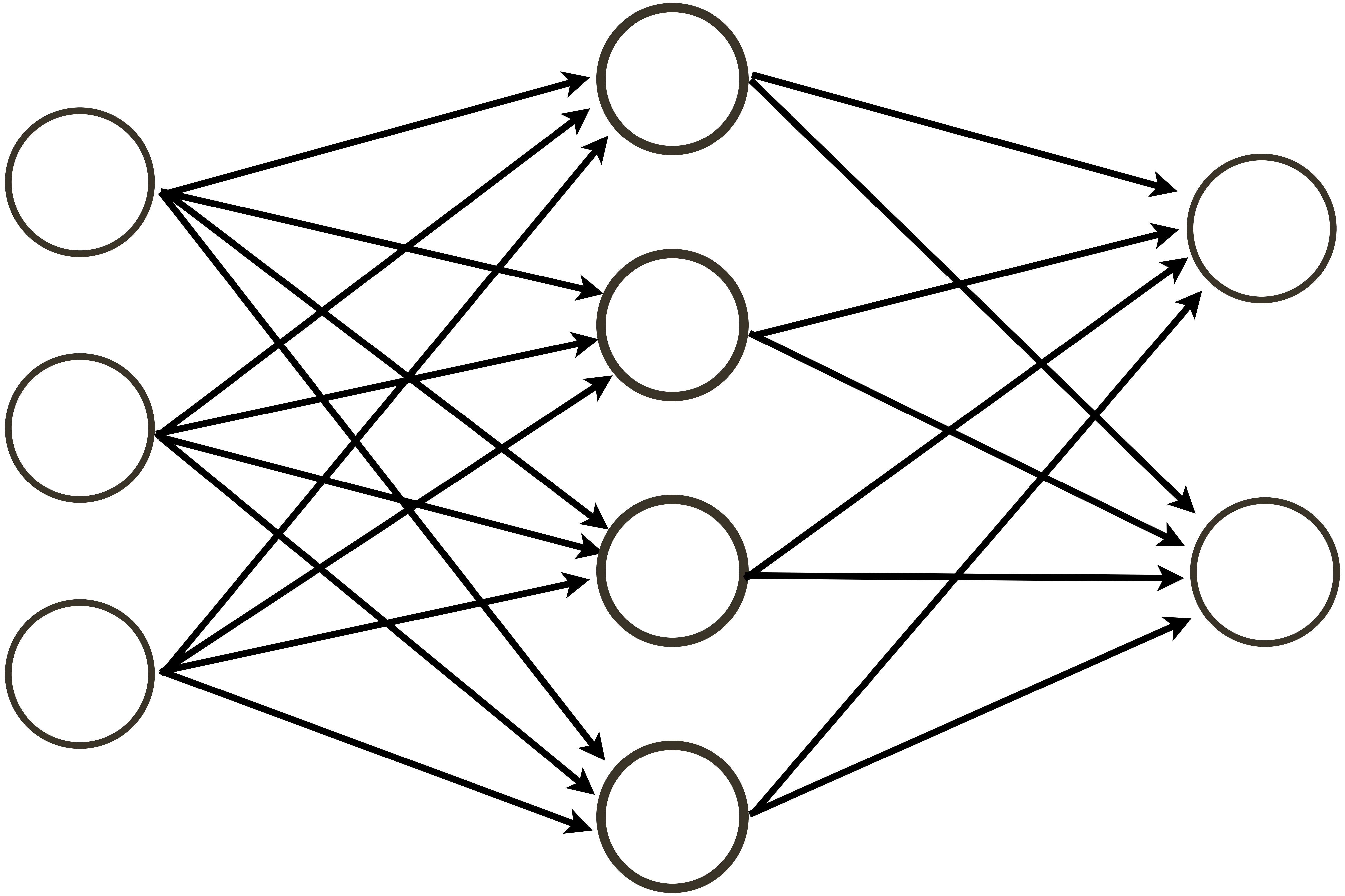


Figure credit: Fei-Fei and Karpathy

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times 4)} \sigma\left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

Activation Function

Why can't we have **linear** activation functions? Why have non-linear activations?



Activation Function

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times 4)}\sigma\left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

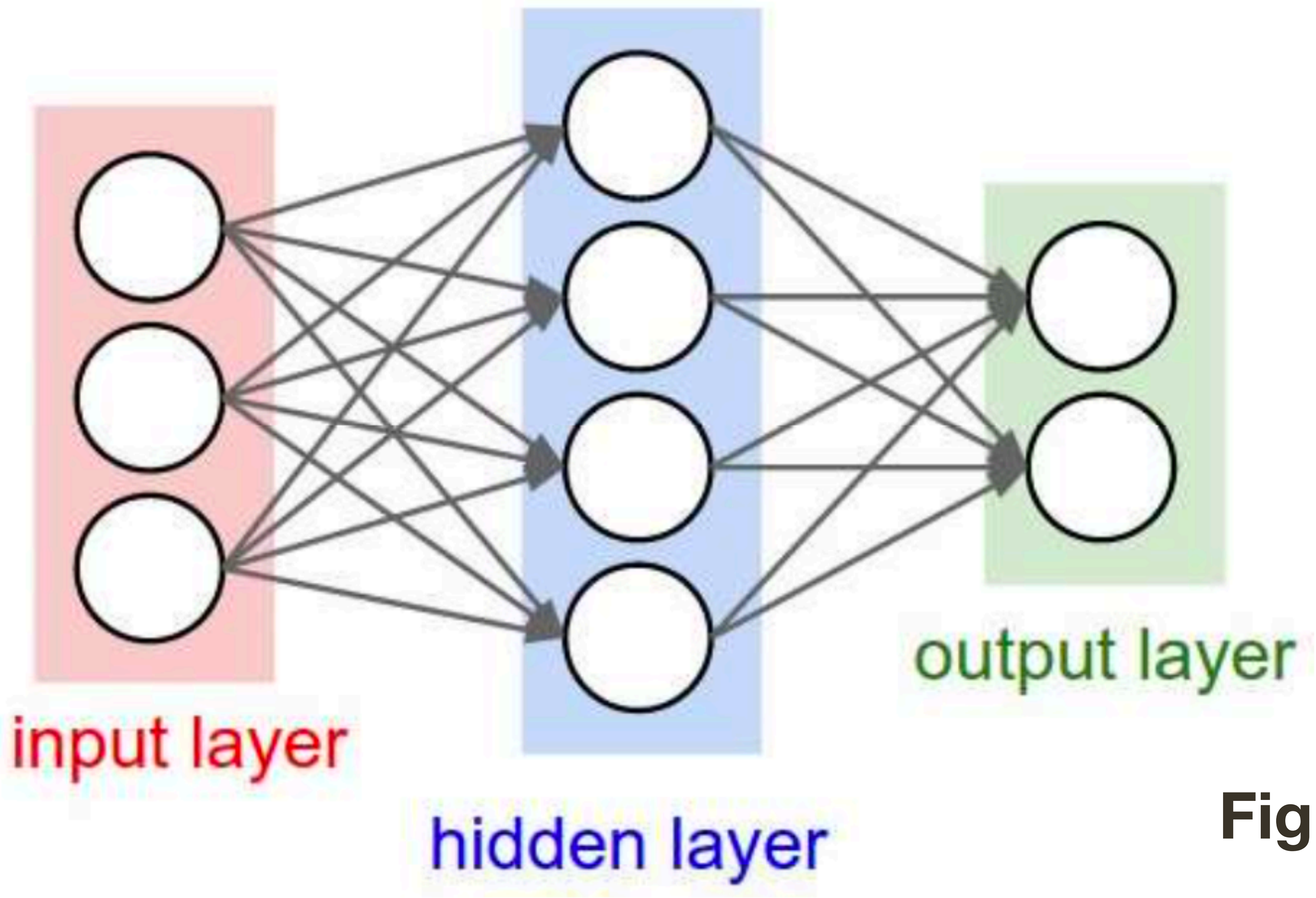


Figure credit: Fei-Fei and Karpathy

Activation Function

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

$$= \mathbf{W}_2^{(2\times4)}\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}$$

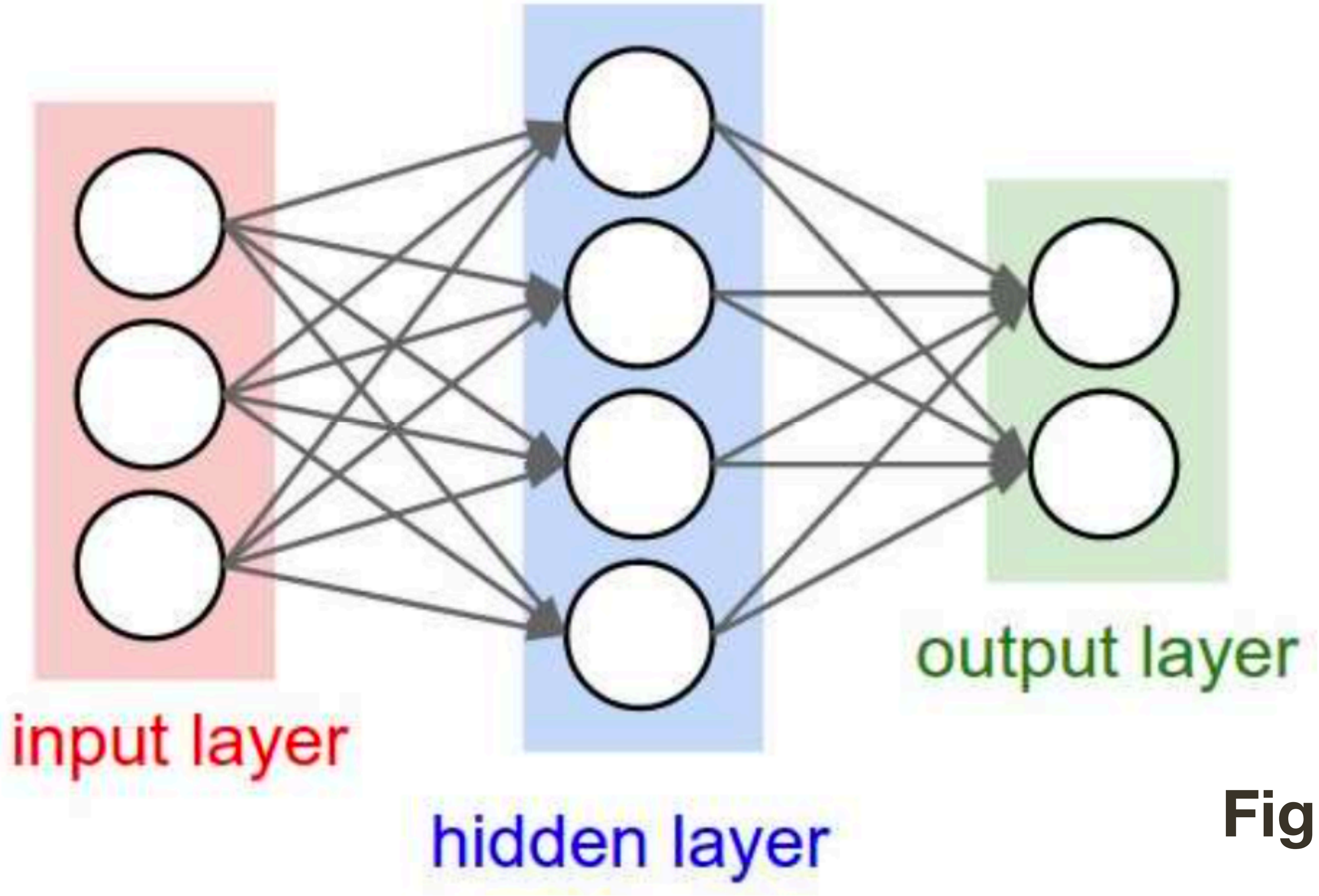


Figure credit: Fei-Fei and Karpathy

Activation Function

$$
\begin{aligned}
\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) &= \sigma\left(\mathbf{W}_2^{(2\times 4)} \sigma\left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right) \\
&= \mathbf{W}_2^{(2\times 4)} \left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)} \\
&= \mathbf{W}_2^{(2\times 4)}\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{W}_2^{(2\times 4)}\mathbf{b}_1^{(4)} + \mathbf{b}_2^{(2)}
\end{aligned}
$$

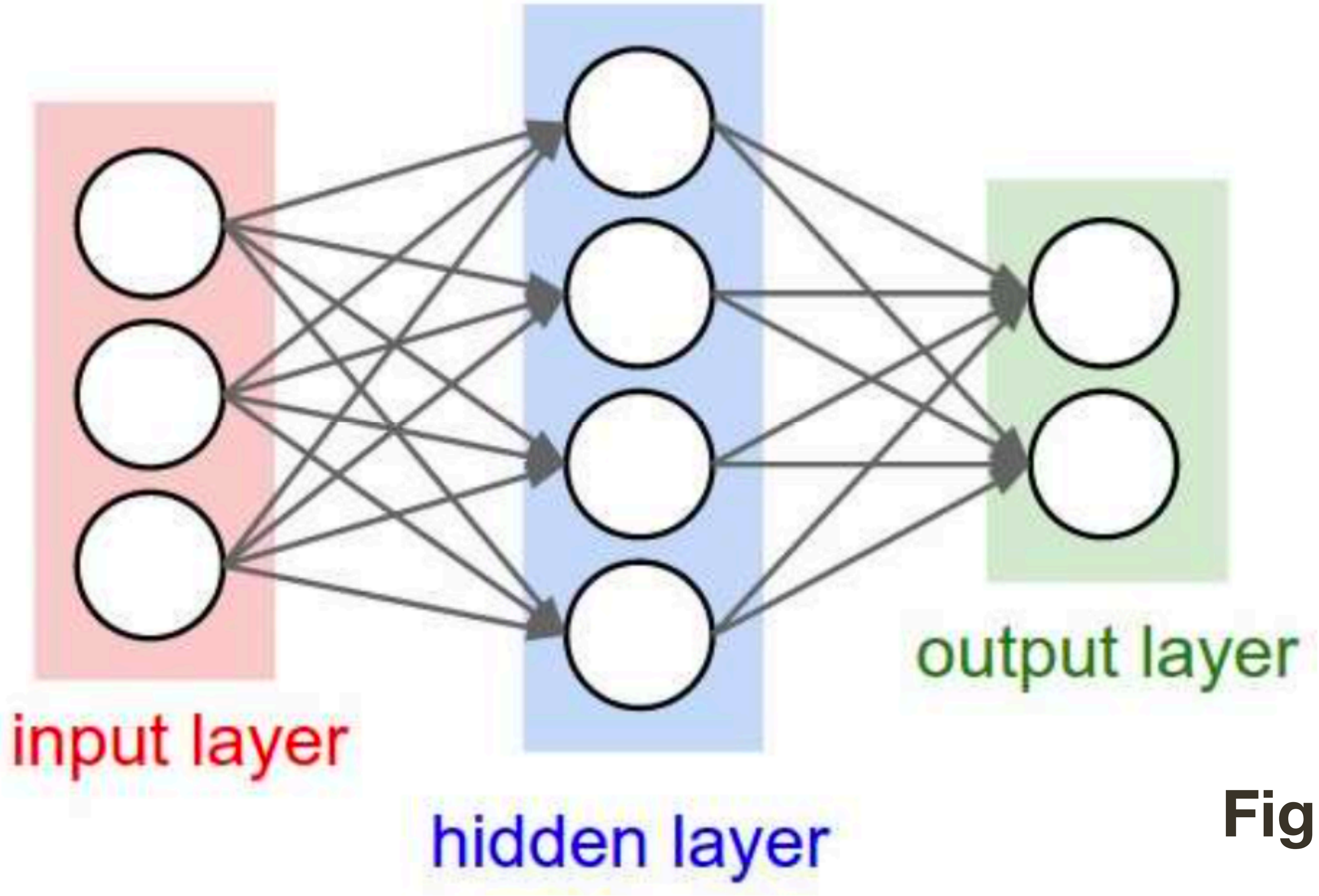


Figure credit: Fei-Fei and Karpathy

Activation Function

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

$$= \mathbf{W}_2^{(2\times4)}\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}$$

$$= \underbrace{\mathbf{W}_2^{(2\times4)}\mathbf{W}_1^{(4\times3)}}_{\mathbf{W}_*^{(2\times3)}}\mathbf{x} + \underbrace{\mathbf{W}_2^{(2\times4)}\mathbf{b}_1^{(4)} + \mathbf{b}_2^{(2)}}_{\mathbf{b}^{(2)}}$$

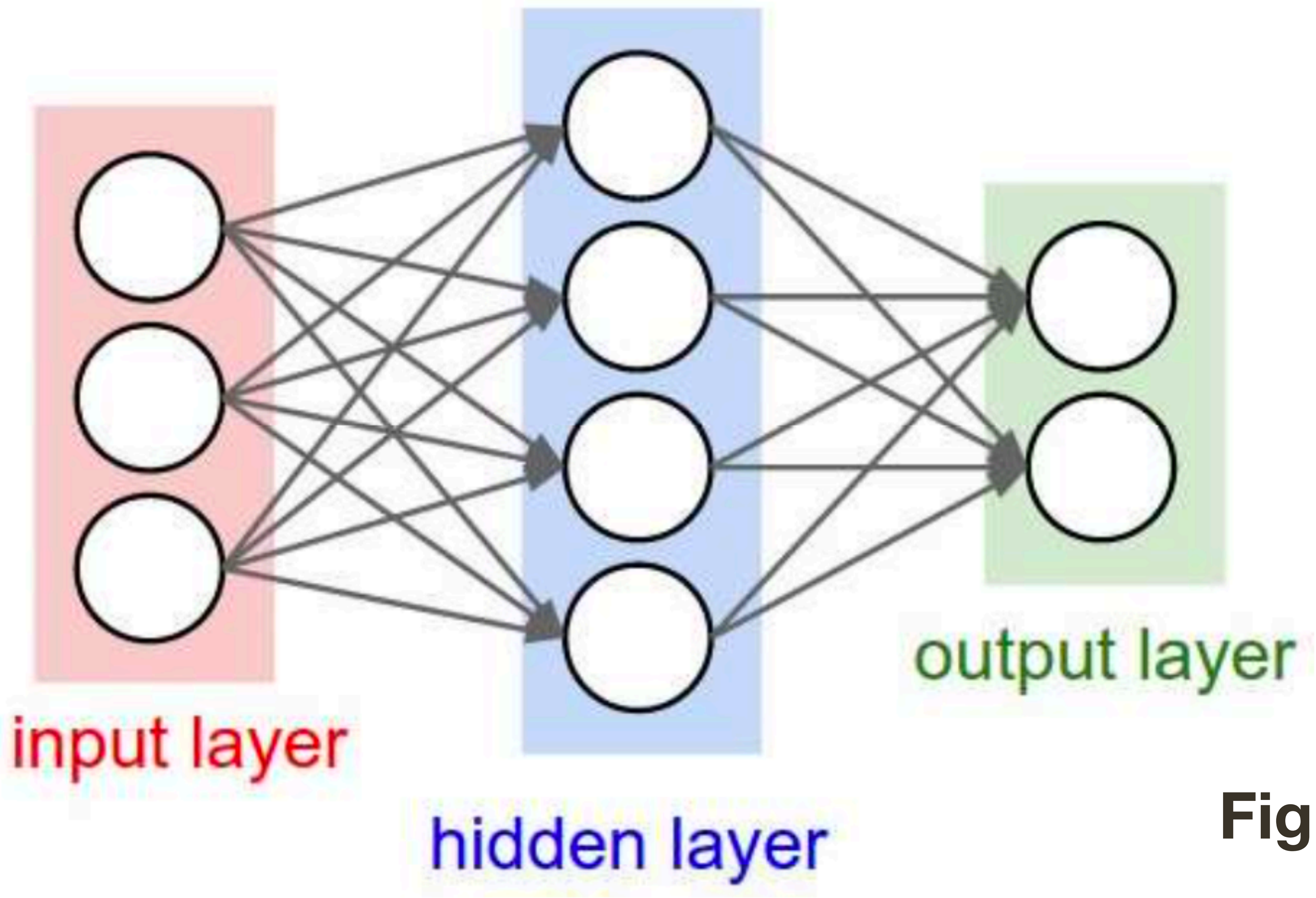


Figure credit: Fei-Fei and Karpathy

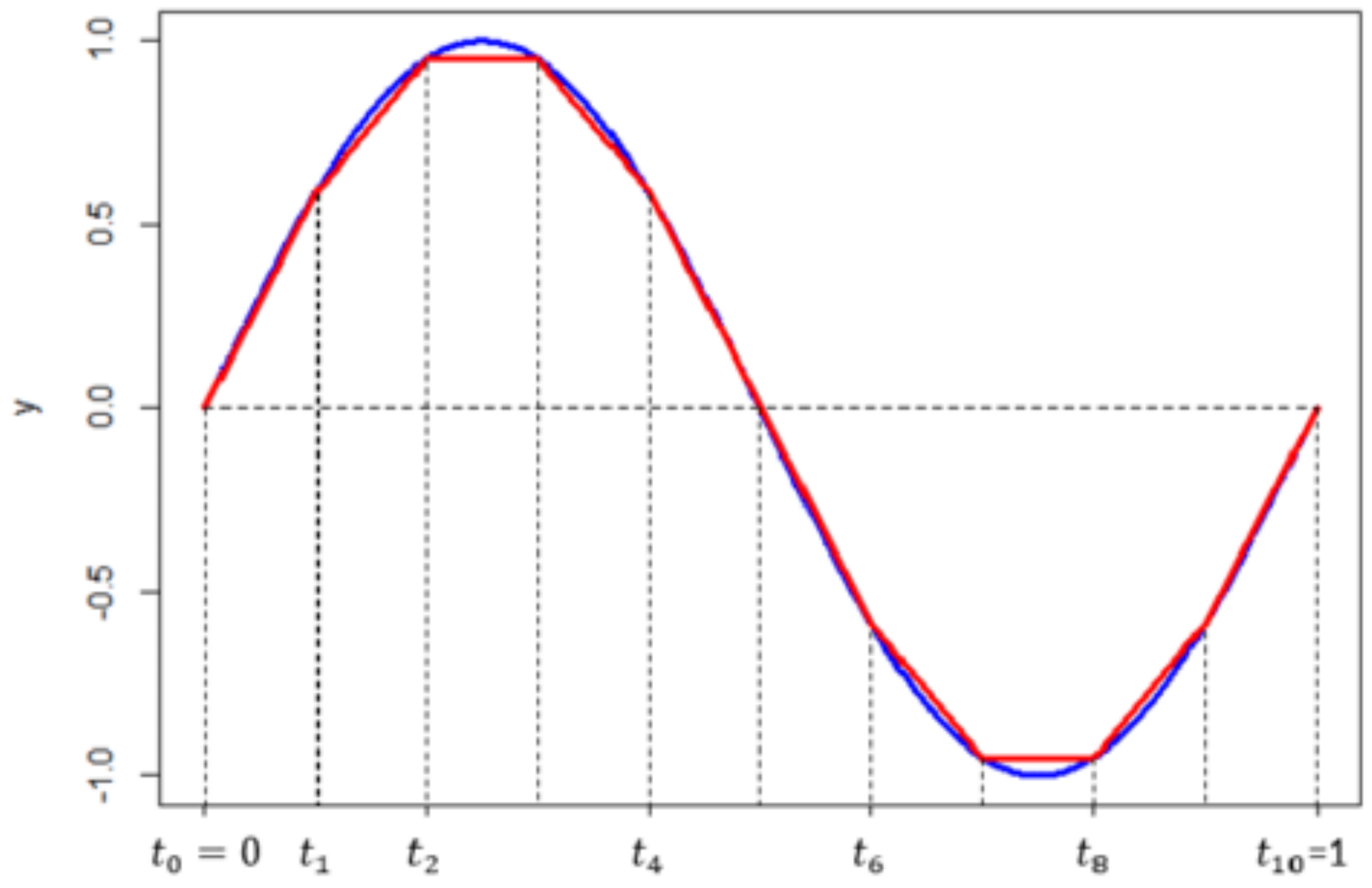
Activation Function

Non-linear activation is required to provably make the Neural Net a **universal function approximator**

Intuition: with ReLU activation, we effectively get a linear spline approximation to any function.

Optimization of neural net parameters = finding slops and transitions of linear pieces

The quality of approximation depends on the number of linear segments



Number of linear segments for large input dimension: $\Omega(2^{\frac{2}{3}Ln})$

Light Theory: Neural Network as Universal Approximator

Universal Approximation Theorem: Single hidden layer can approximate any continuous function with compact support to arbitrary accuracy, when the width goes to infinity.

[Hornik *et al*., 1989]

Universal Approximation Theorem (revised): A network of infinite depth with a hidden layer of size $d+1$ neurons, where d is the dimension of the input space, can approximate any continuous function.

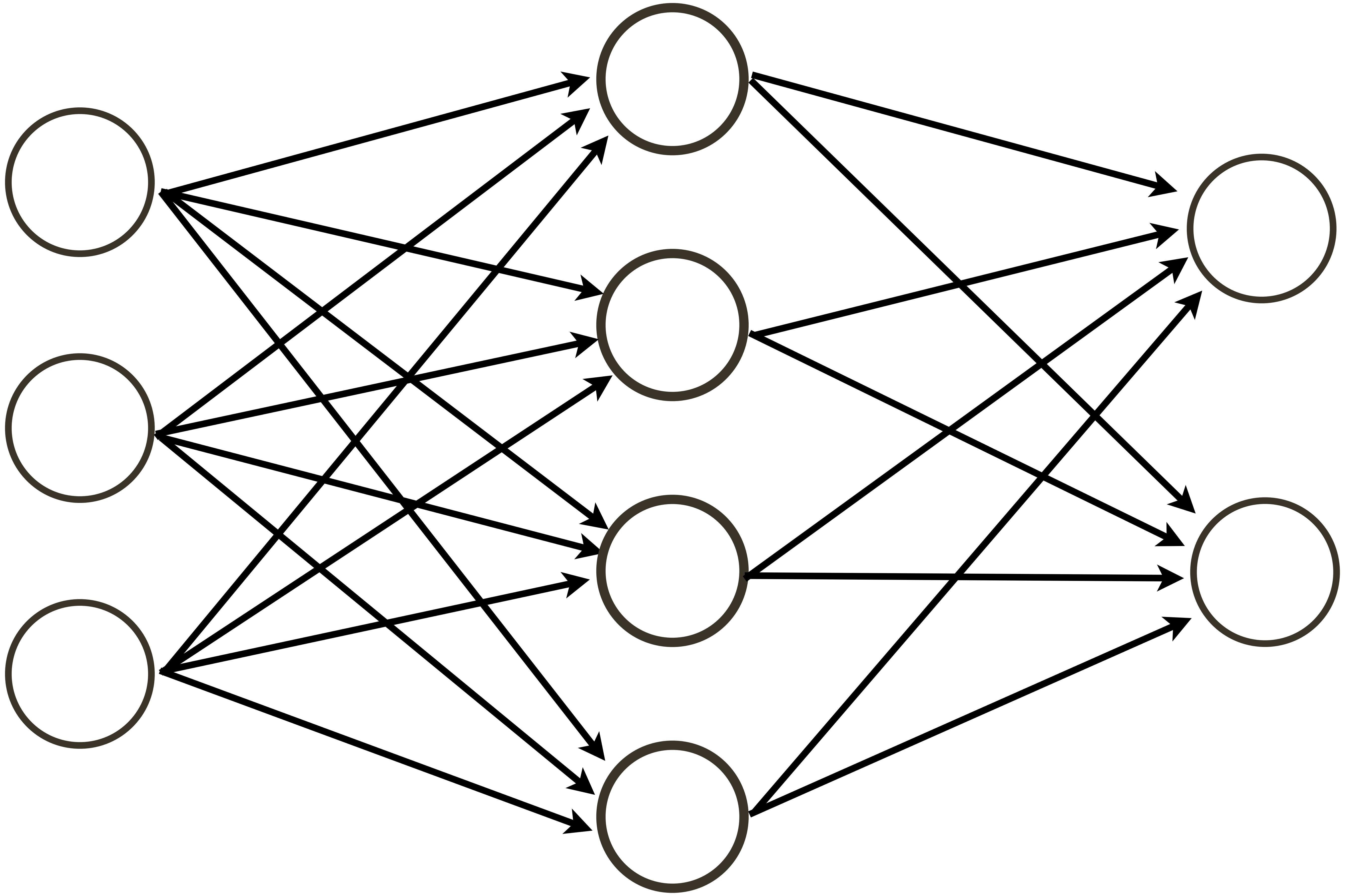
[Lu *et al*., NIPS 2017]

Universal Approximation Theorem (further revised): ResNet with a single hidden unit and infinite depth can approximate any continuous function.

[Lin and Jegelka, NIPS 2018]

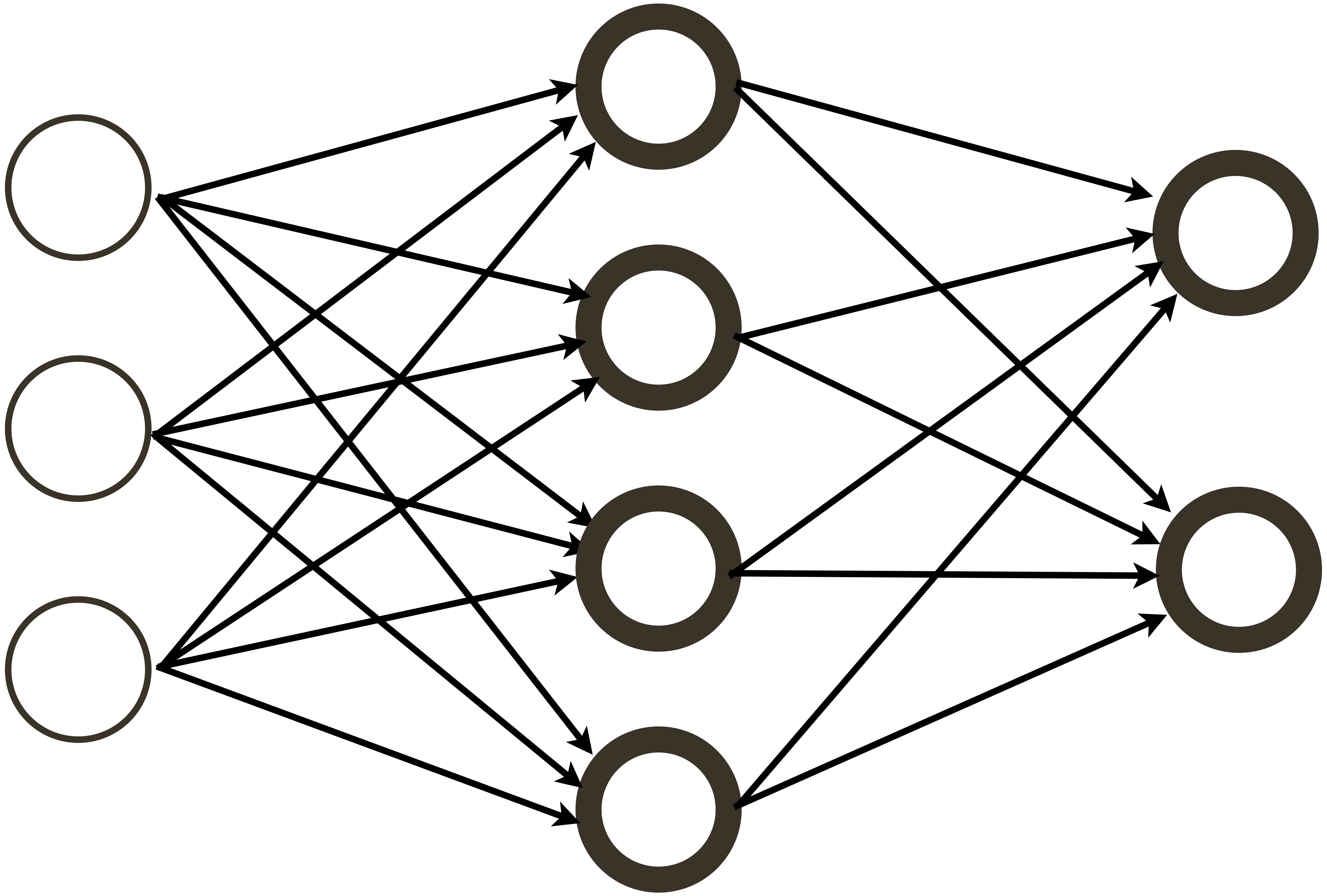
Neural Network

How many neurons?



Neural Network

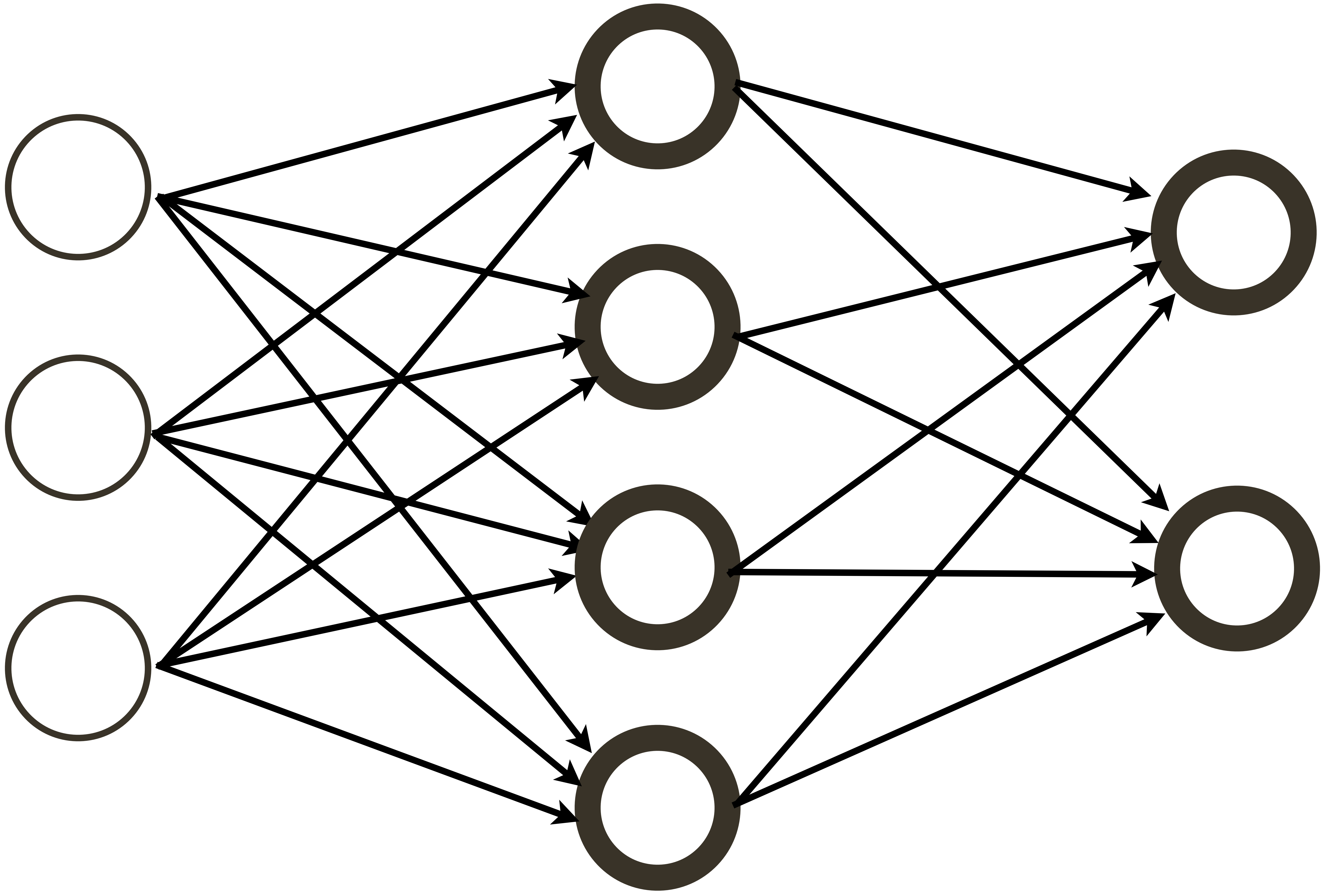
How many neurons?   4+2 = 6



Neural Network

How many neurons? 4+2 = 6

How many weights?

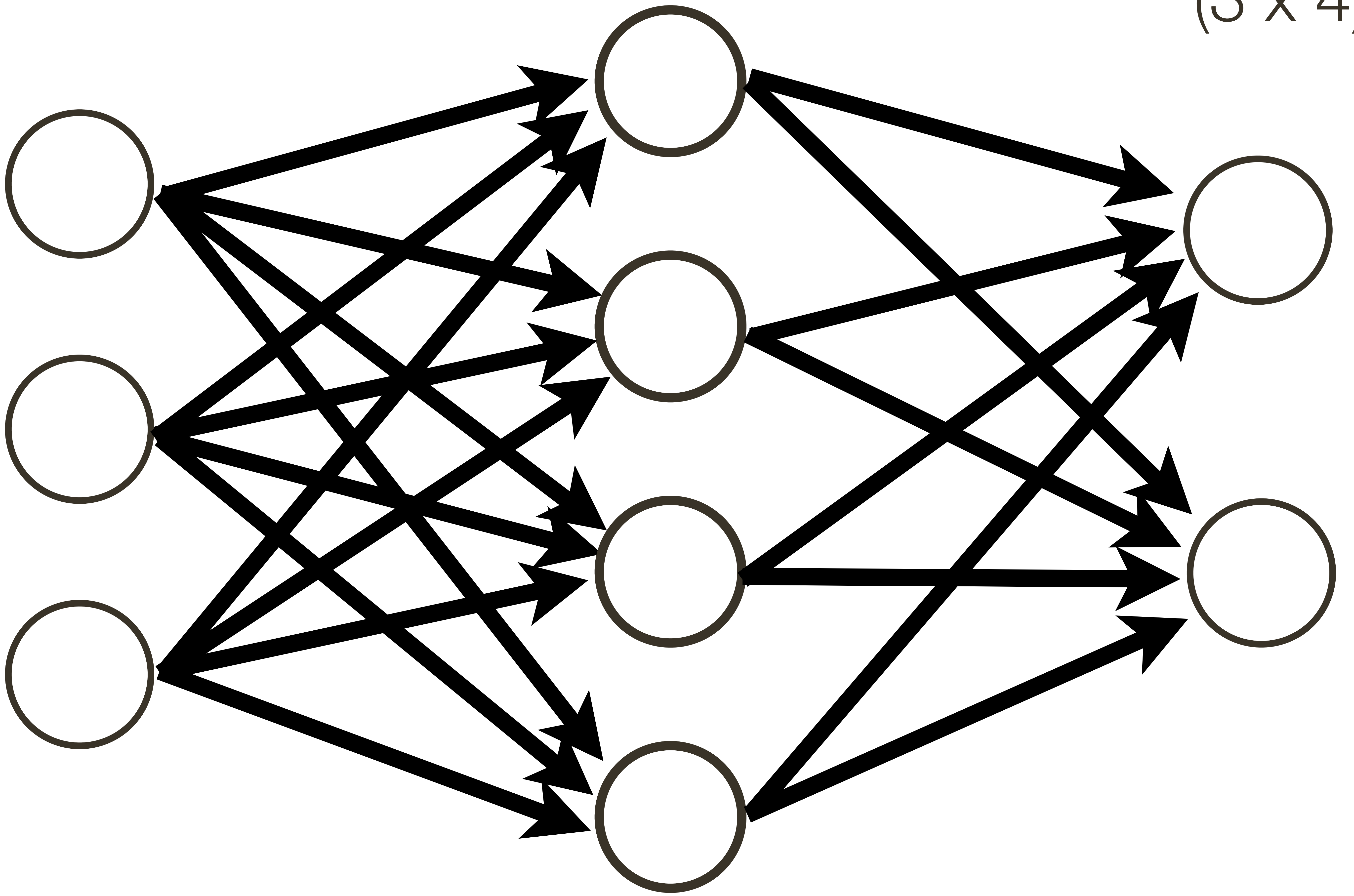


Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20

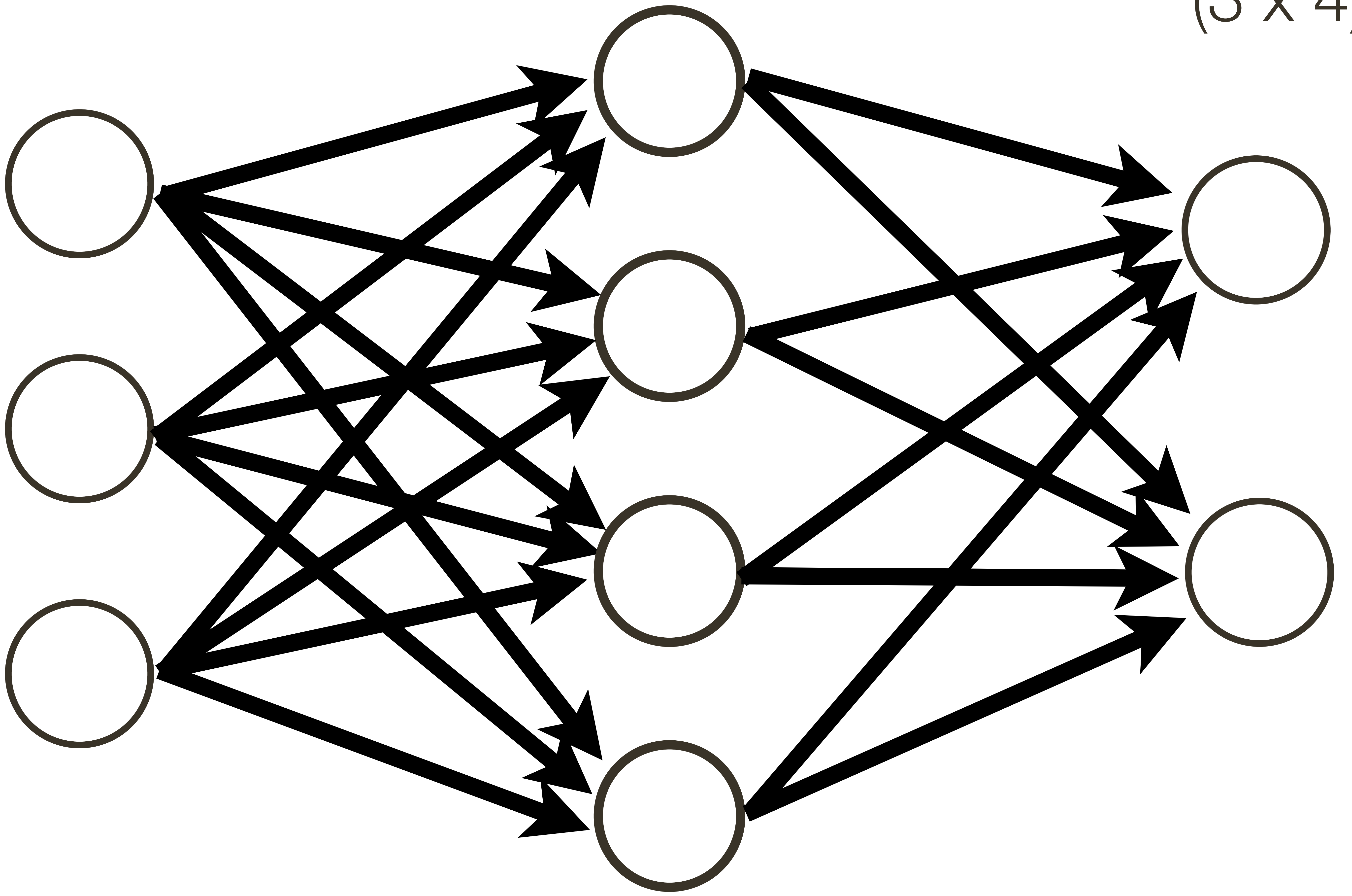


Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20



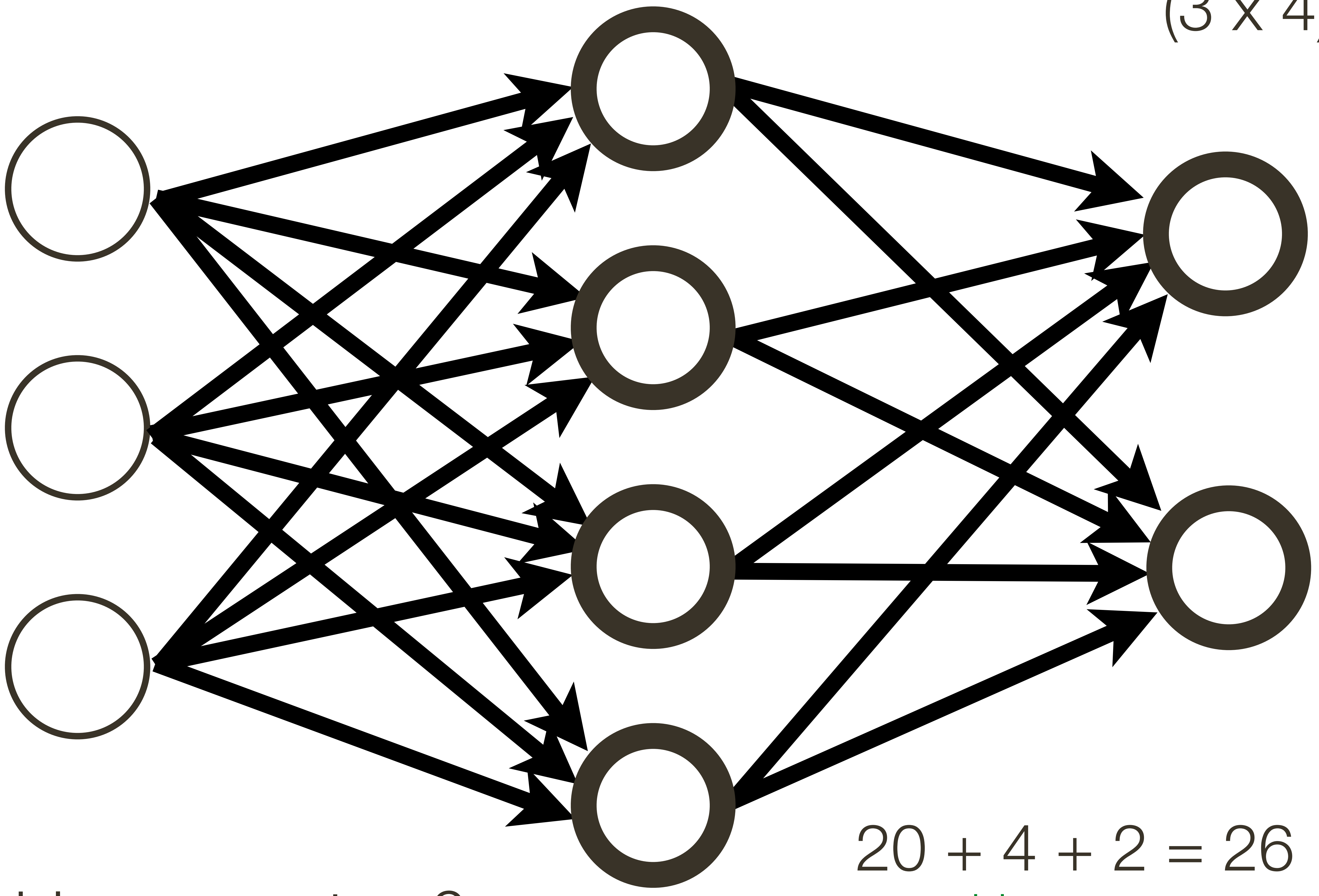
How many learnable parameters?

Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20



How many learnable parameters?

20 + 4 + 2 = 26

bias terms

Neural Networks

Modern **convolutional neural networks** contain 10-20 layers and on the order of 100 million parameters

Training a neural network requires estimating a large number of parameters

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

$$f$$

$$c_1 = -2.85$$

$$c_2 = 0.86$$

$$c_3 = 0.28$$

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

f		
$c_1 = -2.85$	$\xrightarrow{\exp}$	0.058
$c_2 = 0.86$		2.36
$c_3 = 0.28$		1.32

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

$$f$$

f			Normalize to sum to 1	
$c_1 = -2.85$	$\xrightarrow{\exp}$	0.058	$\longrightarrow$	0.016
$c_2 = 0.86$		2.36		0.631
$c_3 = 0.28$		1.32		0.353

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

f				probability of a class
$c_1 = -2.85$	exp →	0.058	Normalize to sum to 1 →	0.016
$c_2 = 0.86$		2.36		0.631
$c_3 = 0.28$		1.32		0.353

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

softmax function
multi-class classifier

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

f			Normalize to sum to 1	probability of a class
$c_1 = -2.85$	exp →	0.058	→	0.016
$c_2 = 0.86$		2.36		0.631
$c_3 = 0.28$		1.32		0.353

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

f			Normalize to sum to 1	probability of a class	
$c_1 = -2.85$	exp →	0.058	→	0.016	
$c_2 = 0.86$		2.36		0.631	$L_i = -\log(0.353) = 1.04$
$c_3 = 0.28$		1.32		0.353	

Backpropagation

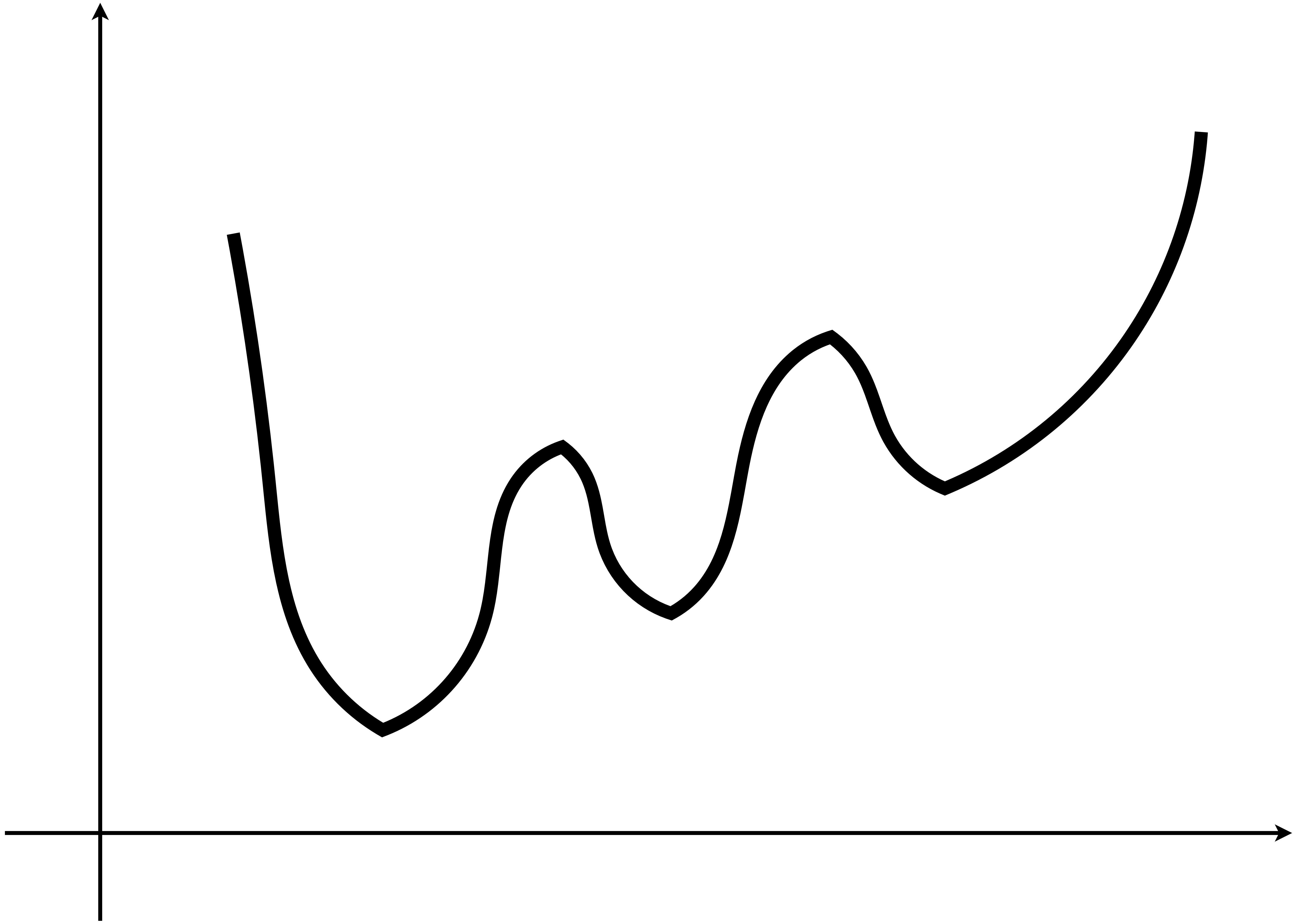
When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

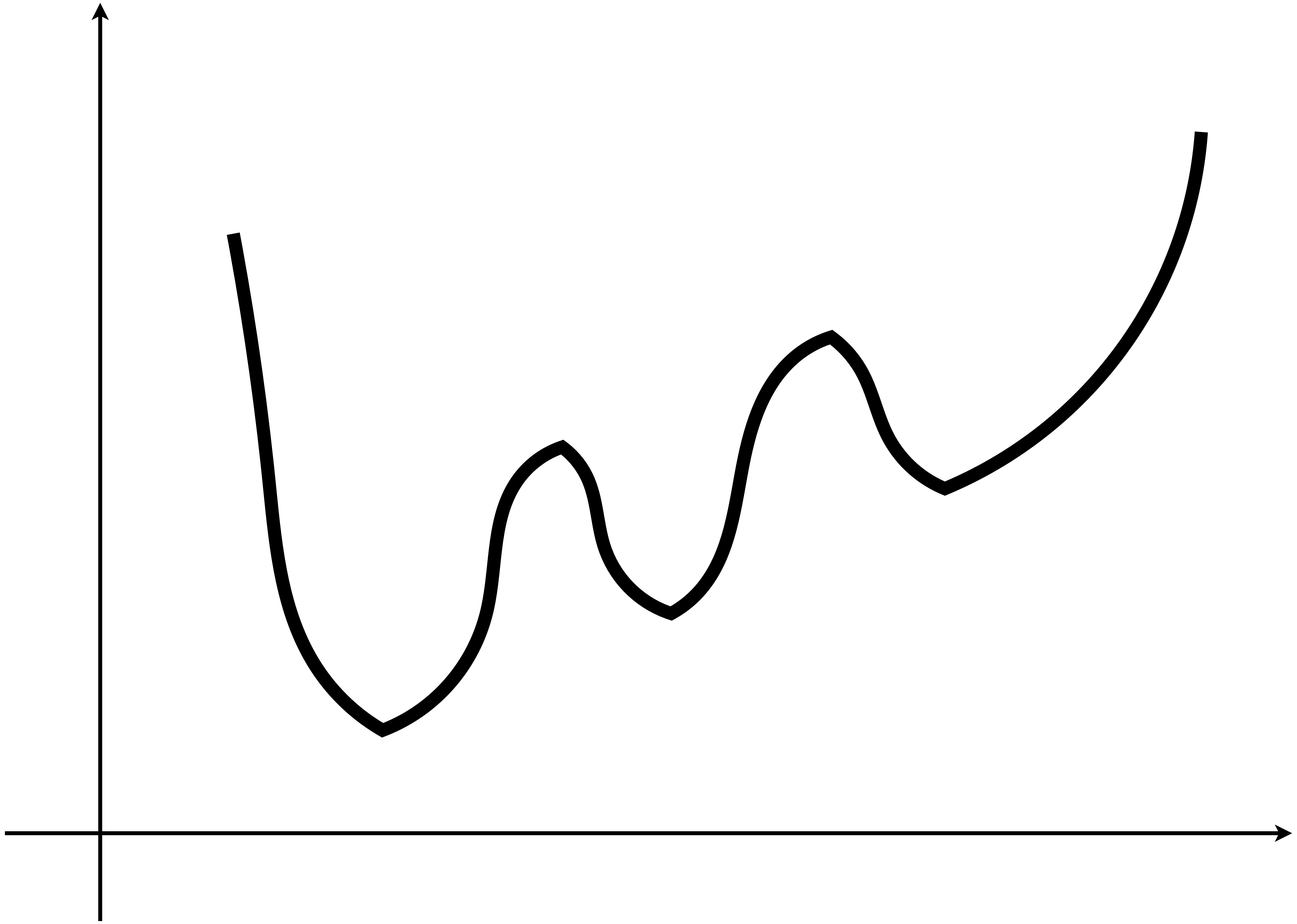
We want to compute the **gradient** of the loss with respect to the network parameters so that we can incrementally adjust the network parameters

Gradient Descent



*slide adopted from V. Ordonex

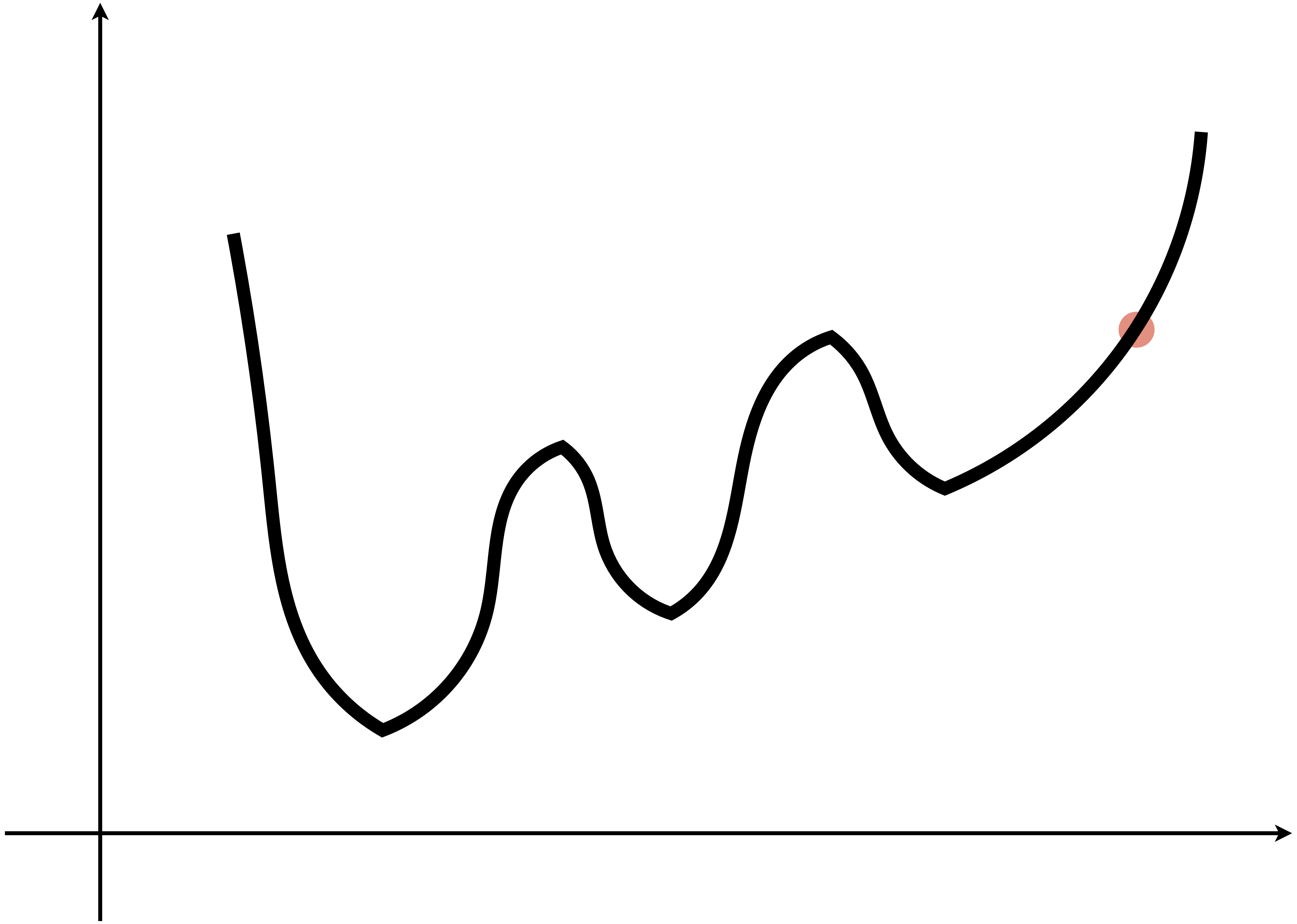
Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

*slide adopted from V. Ordonex

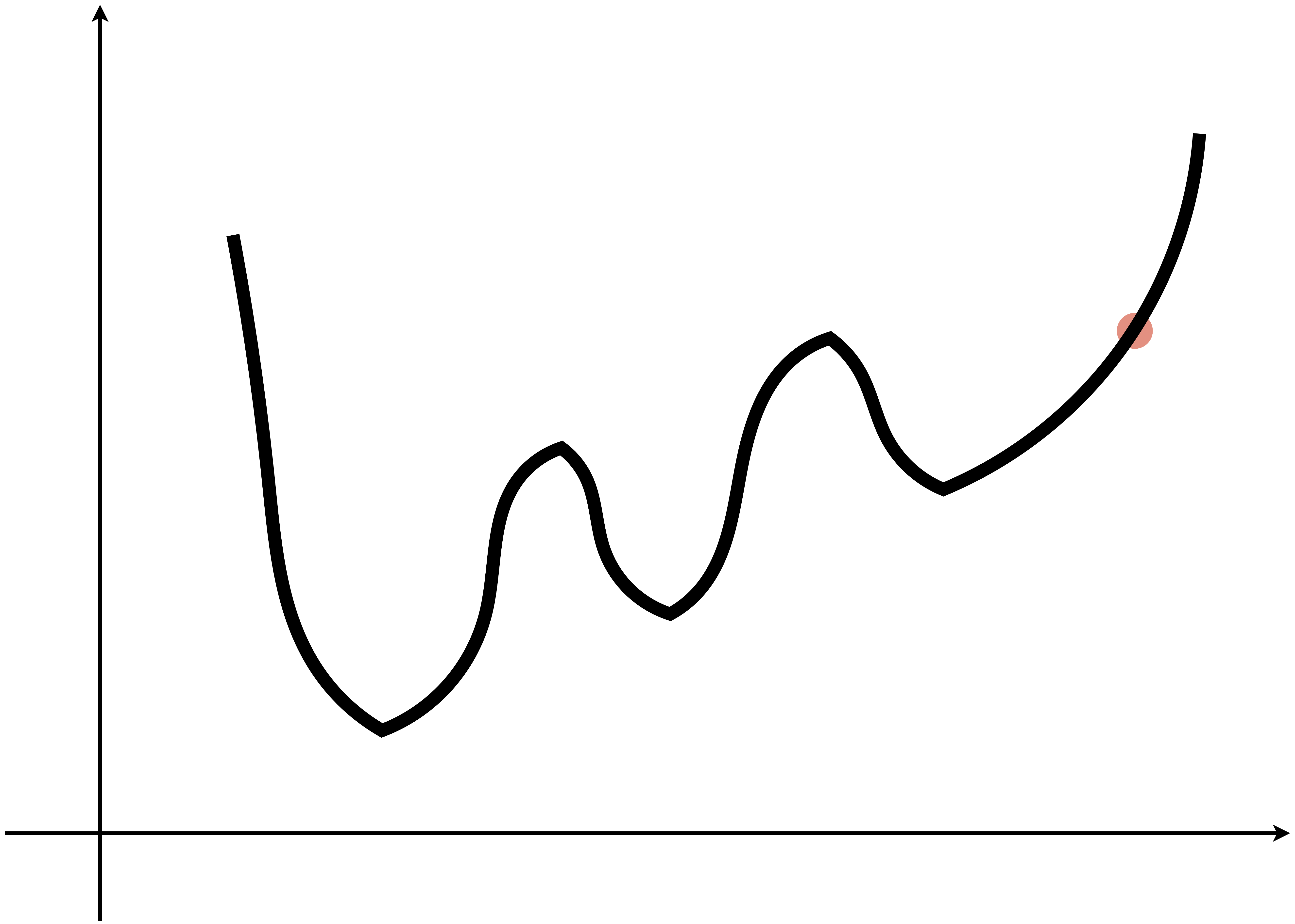
Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

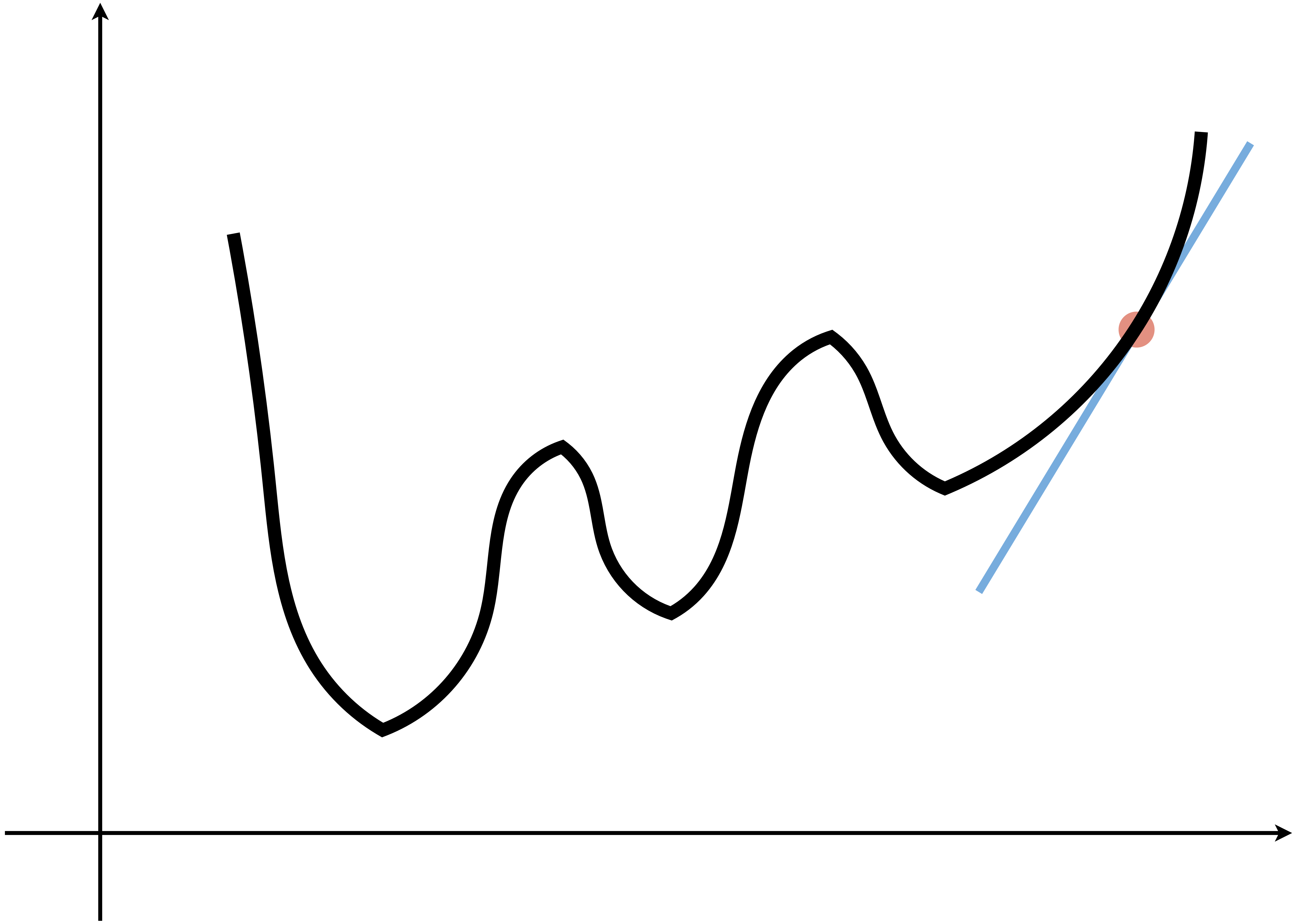
For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

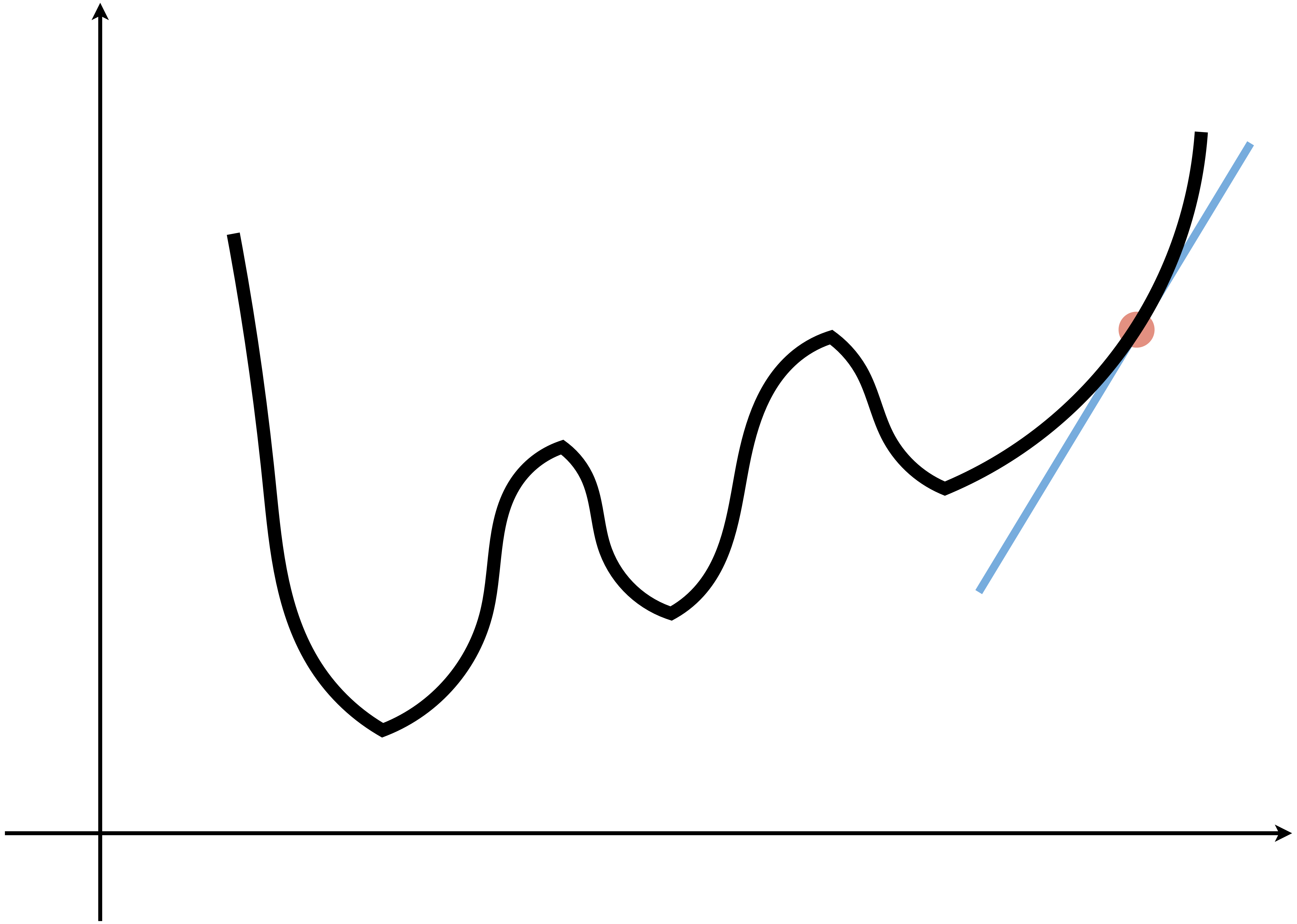
For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

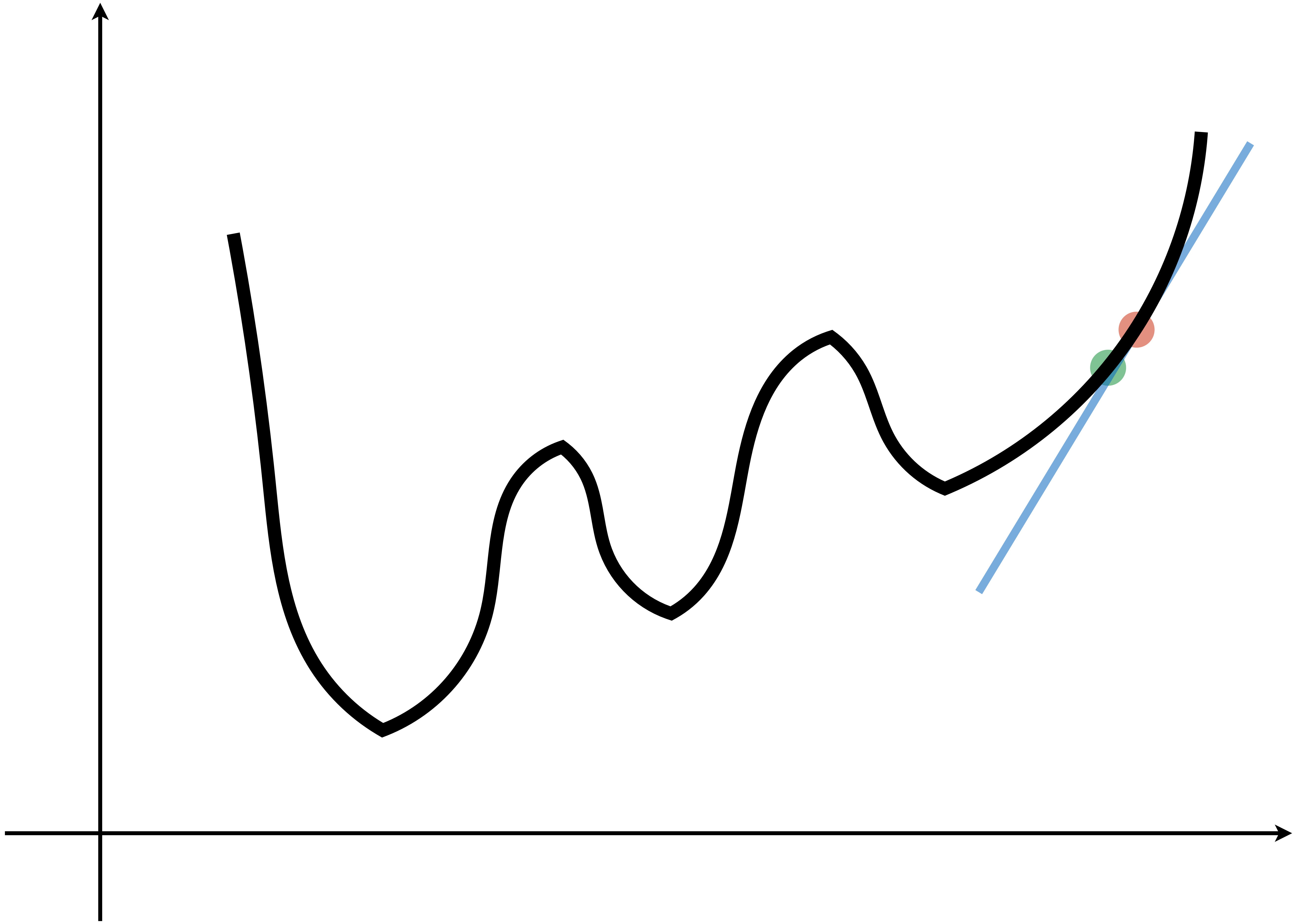
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \left.\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \left.\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

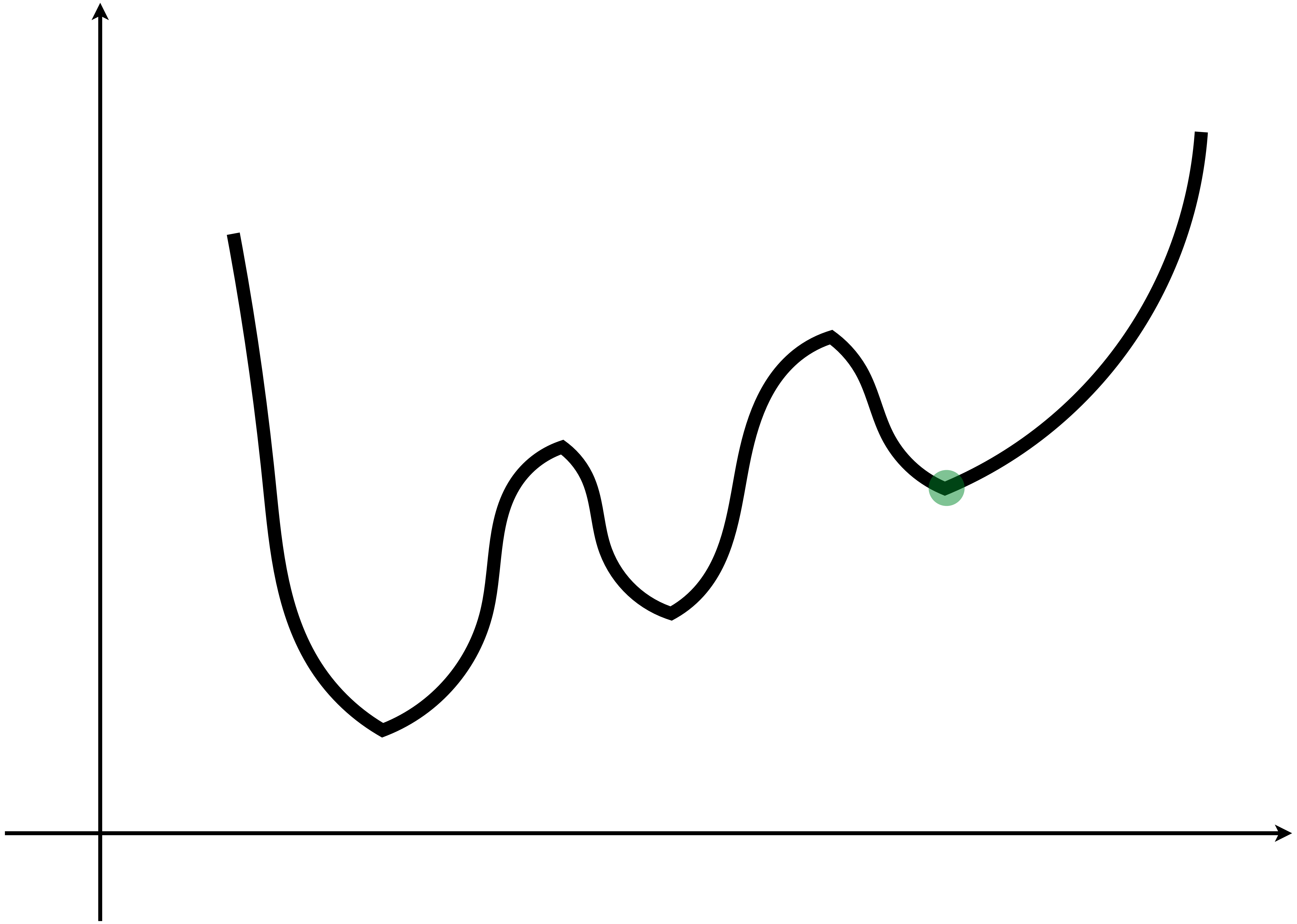
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\Big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\Big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

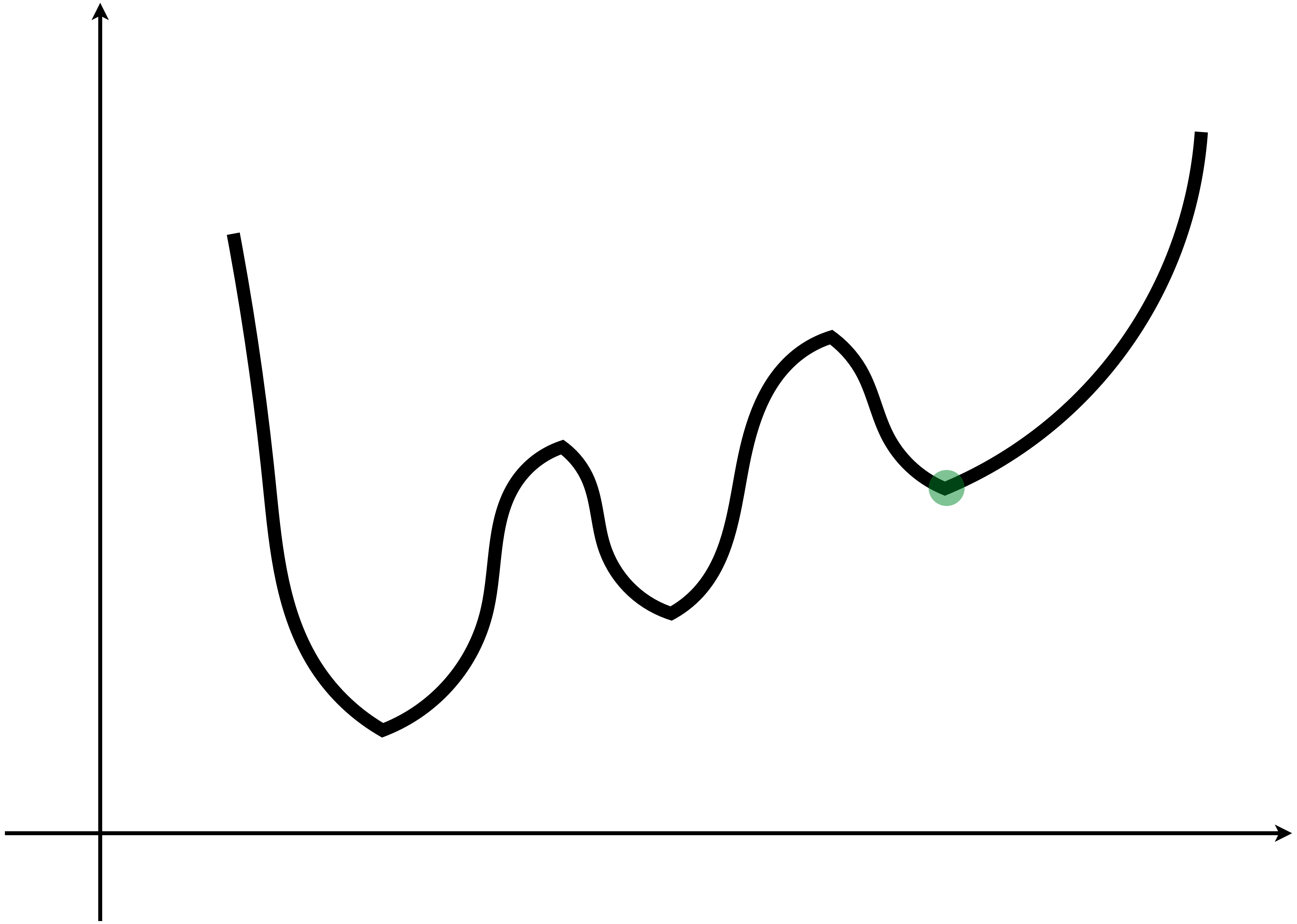
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \left.\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \left.\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})\big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \underline{\lambda} \left. \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}} \right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \underline{\lambda} \left. \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}} \right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

λ - is the learning rate

*slide adopted from V. Ordonex

Loss:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = ||\mathbf{y} - \hat{\mathbf{y}}|| = ||\mathbf{y} - f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)||$$

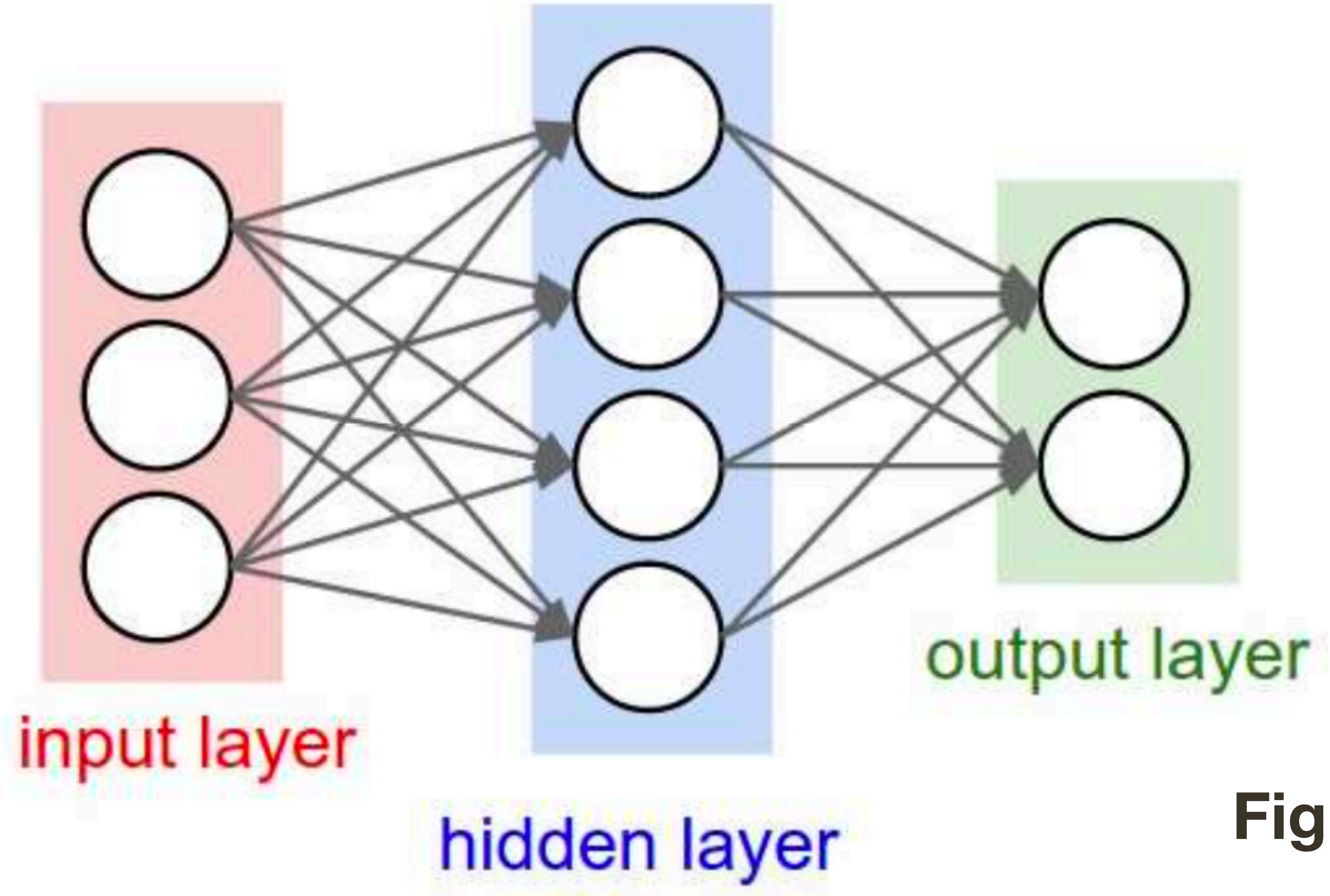


Figure credit: Fei-Fei and Karpathy

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

Re-cap

Neural Network

Loss:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = ||\mathbf{y} - \hat{\mathbf{y}}|| = ||\mathbf{y} - f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)||$$

Gradient Descent

$$\mathbf{W}_{1,i,j} = \mathbf{W}_{1,i,j} - \lambda \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_{1,i,j}}$$

$$\mathbf{b}_{1,i} = \mathbf{b}_{1,i} - \lambda \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}_{1,i}}$$

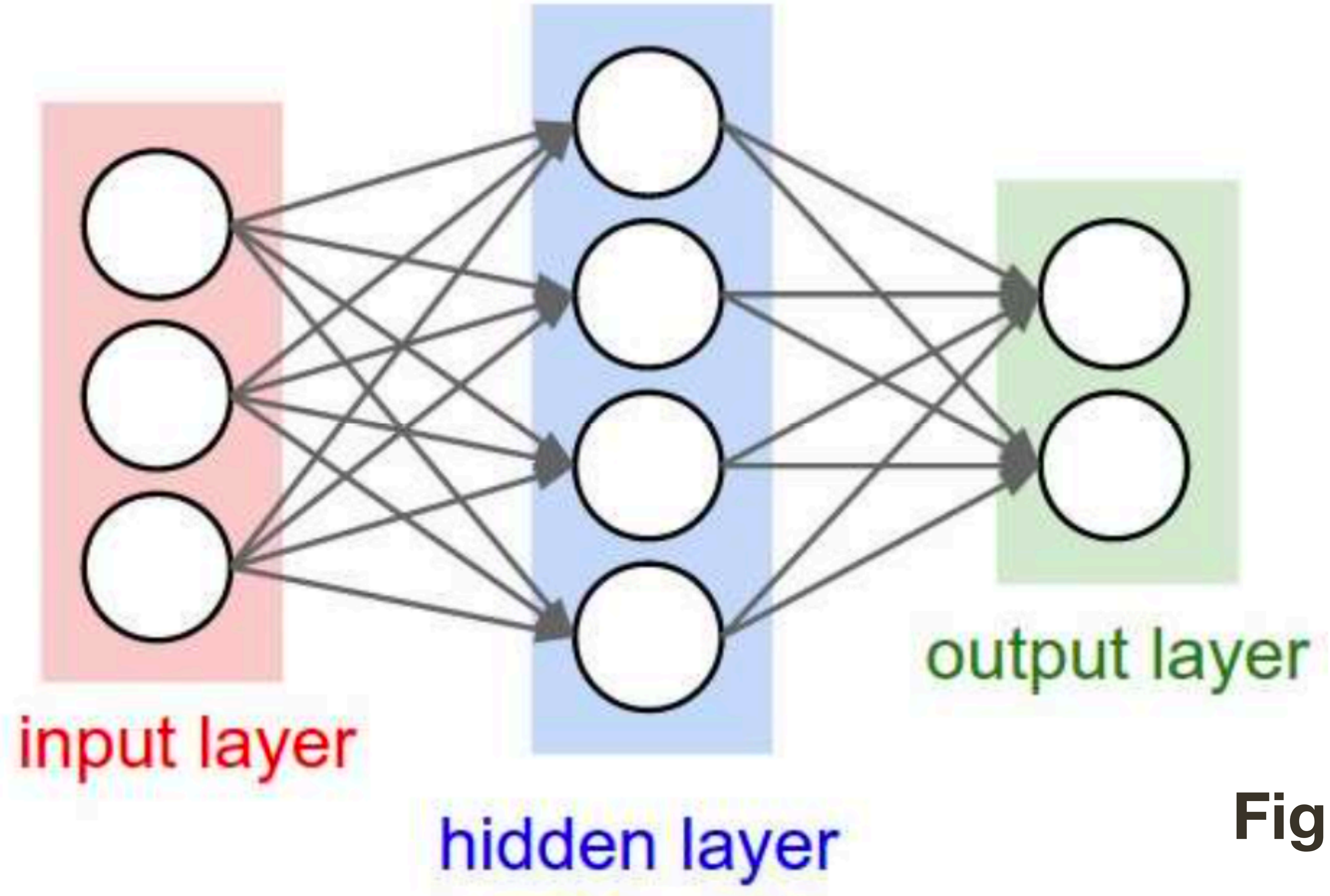


Figure credit: Fei-Fei and Karpathy

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times 4)} \sigma\left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

Backpropagation

The parameters of a neural network are learned using **backpropagation**, which computes gradients via recursive application of the **chain rule** from calculus

Backpropagation

The parameters of a neural network are learned using **backpropagation**, which computes gradients via recursive application of the **chain rule** from calculus

Suppose $f(x, y) = xy$. What is the partial derivative of f with respect to x? What is the partial derivative of f with respect to y?

Backpropagation

The parameters of a neural network are learned using **backpropagation**, which computes gradients via recursive application of the **chain rule** from calculus

Suppose $f(x, y) = xy$. What is the partial derivative of f with respect to x? What is the partial derivative of f with respect to y?

$$\frac{\partial f}{\partial x} = y \qquad\qquad \frac{\partial f}{\partial y} = x$$

Backpropagation

Suppose $f(x, y) = x + y$. What is the partial derivative of f with respect to x? What is the partial derivative of f with respect to y?

Backpropagation

Suppose $f(x, y) = x + y$. What is the partial derivative of f with respect to x? What is the partial derivative of f with respect to y?

$$\frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

Backpropagation

A trickier example: $f(x, y) = \max(x, y)$

Backpropagation

A trickier example: $f(x, y) = \max(x, y)$

$$\frac{\partial f}{\partial x} = \mathbf{1}(x \geq y) \qquad \frac{\partial f}{\partial y} = \mathbf{1}(y \geq x)$$

That is, the (sub)gradient is 1 on the input that is larger, and 0 on the other input

— For example, say x = 4, y = 2. Increasing y by a tiny amount does not change the value of f (f will still be 4), hence the gradient on y is zero.

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Suppose $f(x, y, z) = (x + y)z$. What are the partial derivatives of f with respect to x? y? z?

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Suppose $f(x, y, z) = (x + y)z$. What are the partial derivatives of f with respect to x? y? z?

For illustration we break this expression into $q = x + y$ and $f = qz$. This is a sum and a product, and we have just seen how to compute partial derivatives for these.

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Suppose $f(x, y, z) = (x + y)z$. What are the partial derivatives of f with respect to x? y? z?

For illustration we break this expression into $q = x + y$ and $f = qz$. This is a sum and a product, and we have just seen how to compute partial derivatives for these.

By the chain rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 = z$$

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Suppose $f(x, y, z) = (x + y)z$. What are the partial derivatives of f with respect to x? y? z?

For illustration we break this expression into $q = x + y$ and $f = qz$. This is a sum and a product, and we have just seen how to compute partial derivatives for these.

By the chain rule

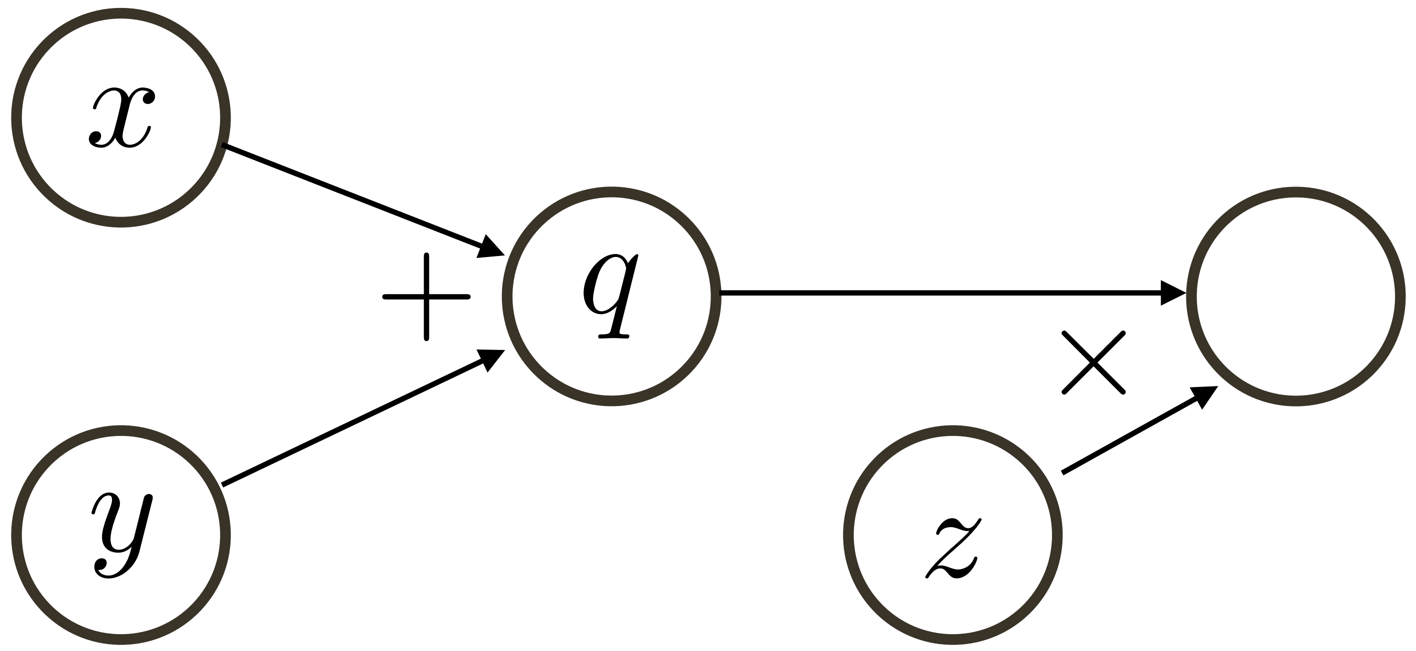
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 = z \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = z \cdot 1 = z \qquad \frac{\partial f}{\partial z} = q$$

Backpropagation

$$f(x, y, z) = (x + y)z$$

Backpropagation

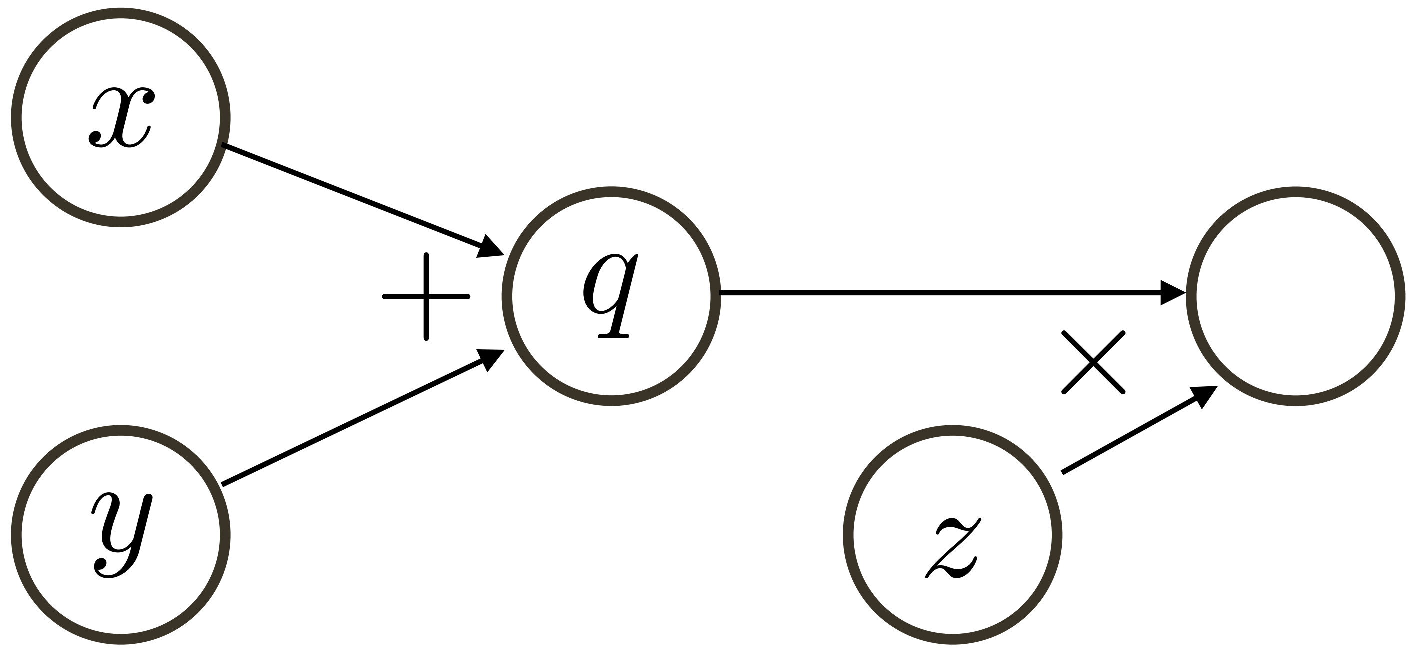
$$f(x, y, z) = (x + y)z$$



Computational graph (a DAG) with variable ordering from topological sort, where each **node** is an input, intermediate, or output variable

Backpropagation

$$f(x, y, z) = (x + y)z$$



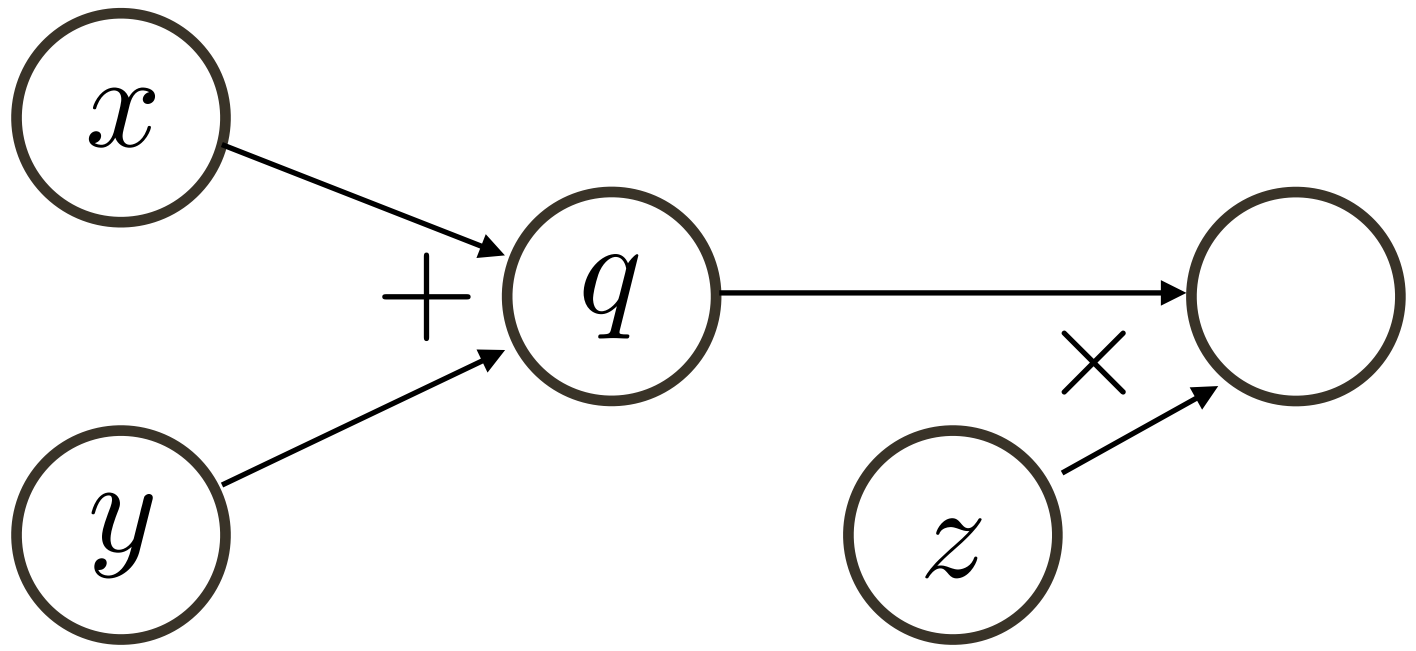
Computational graph (a DAG) with variable ordering from topological sort, where each **node** is an input, intermediate, or output variable

Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

Then: $q = x + y = 3$ $\quad f = qz = -12$ (**forward** pass)

Backpropagation

$$f(x, y, z) = (x + y)z$$



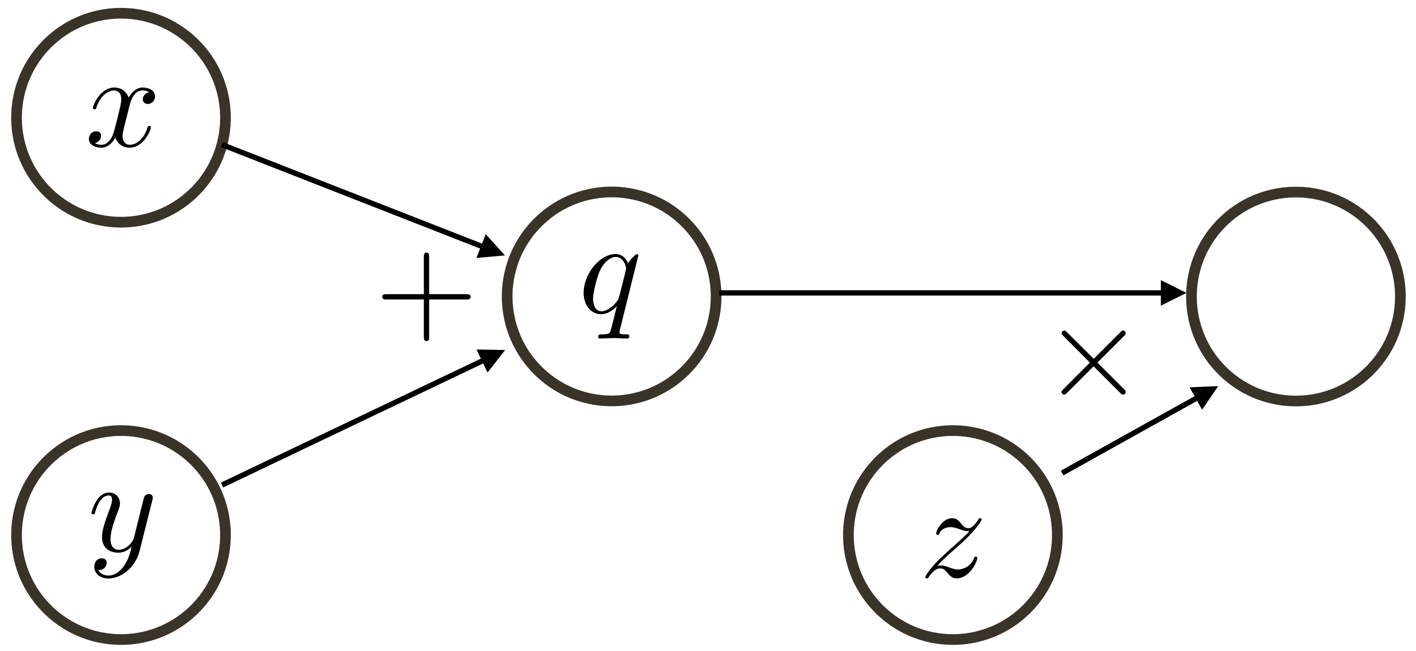
Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

Then: $q = x + y = 3$ $f = qz = -12$ (**forward** pass)

$$\frac{\partial f}{\partial q} = z = -4$$ (**backward** pass)

Backpropagation

$$f(x, y, z) = (x + y)z$$



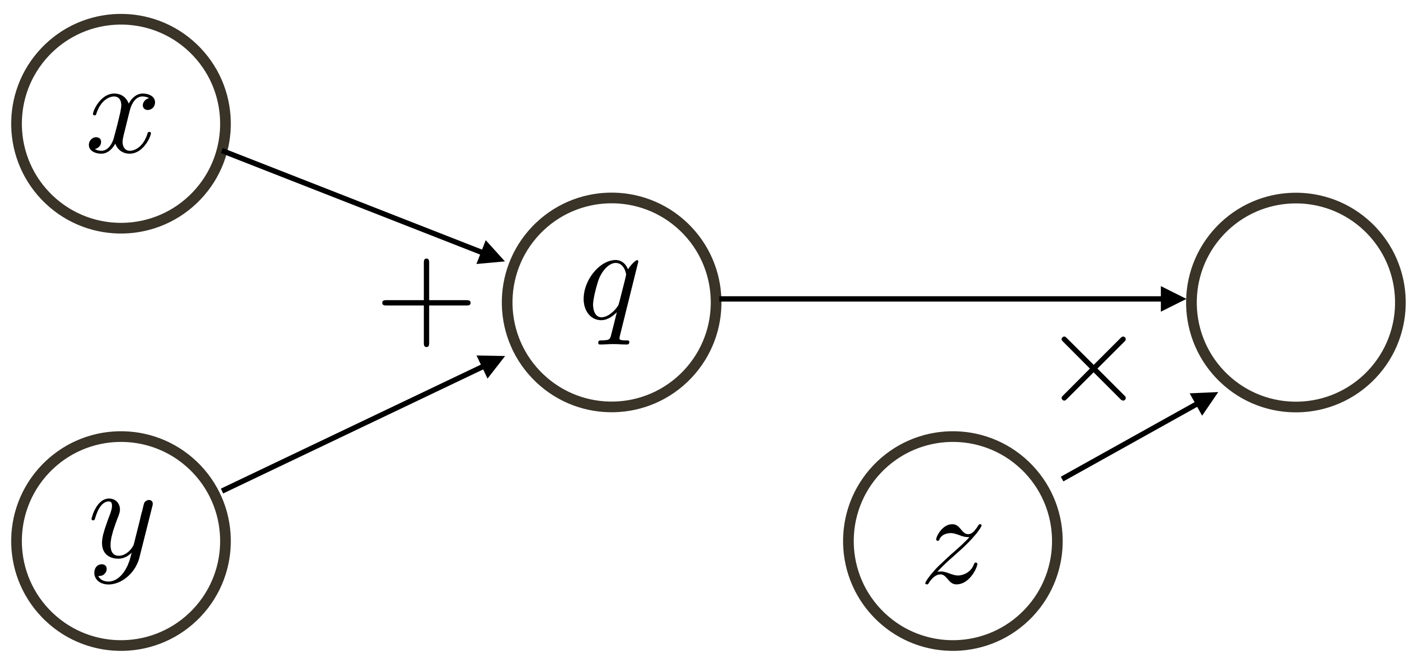
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = \frac{\partial f}{\partial q} \cdot 1$$

Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

Then: $q = x + y = 3$ $\quad f = qz = -12$ (**forward** pass)

$$\frac{\partial f}{\partial q} = z = -4$$ (**backward** pass)

Backpropagation



$$f(x, y, z) = (x + y)z$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = \frac{\partial f}{\partial q} \cdot 1$$

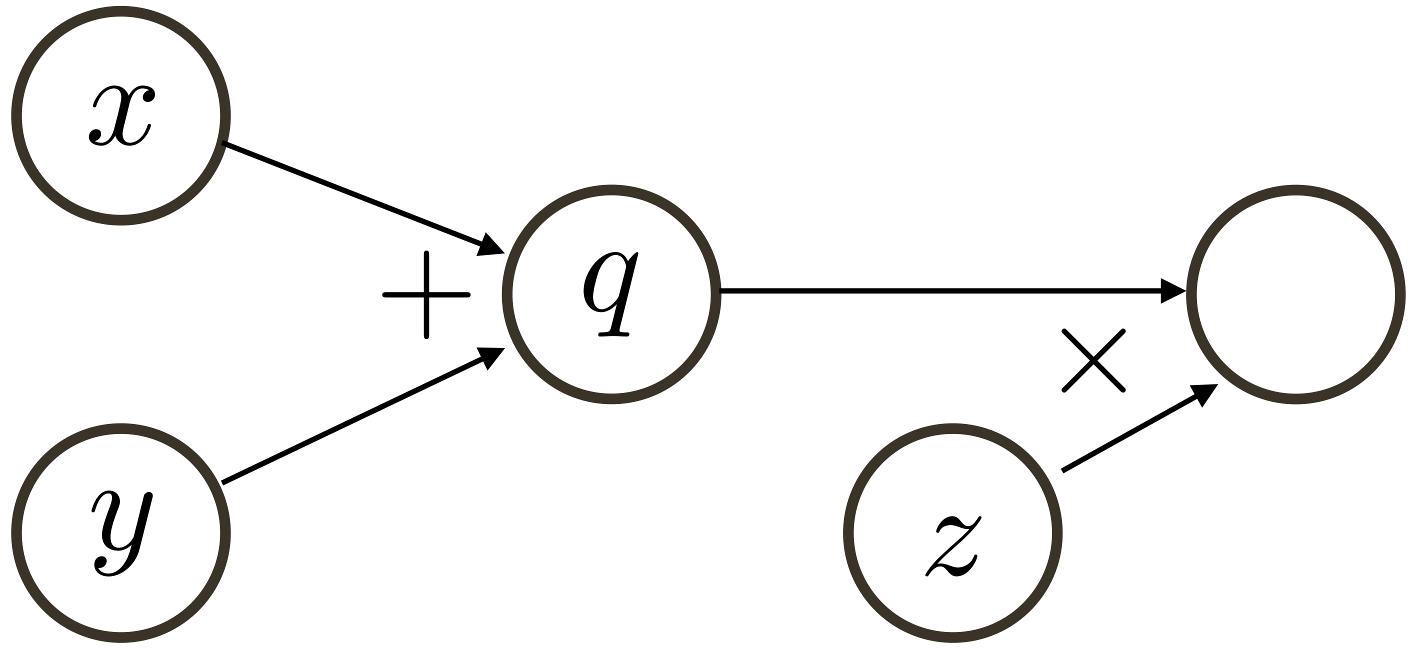
Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

Then: $q = x + y = 3$ $\quad f = qz = -12$ (**forward** pass)

$$\frac{\partial f}{\partial q} = z = -4 \qquad \frac{\partial f}{\partial x} = -4$$

(**backward** pass)

Backpropagation



$$f(x, y, z) = (x + y)z$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = \frac{\partial f}{\partial q} \cdot 1 \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = \frac{\partial f}{\partial q} \cdot 1 \qquad \frac{\partial f}{\partial z} = q$$

Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

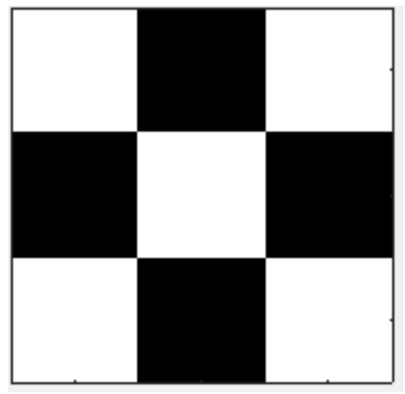
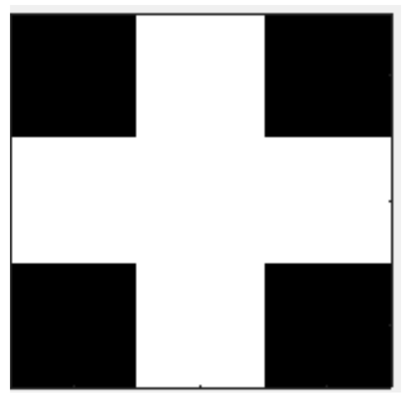
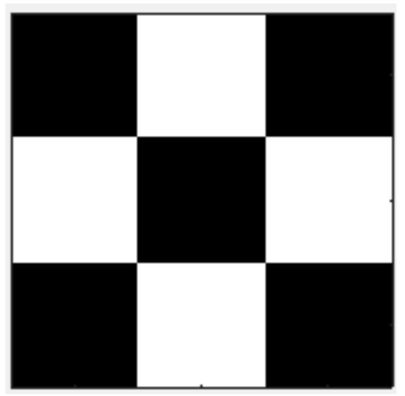
Then: $q = x + y = 3$ $\quad f = qz = -12$ (**forward** pass)

$$\frac{\partial f}{\partial q} = z = -4 \qquad \frac{\partial f}{\partial x} = -4 \qquad \frac{\partial f}{\partial y} = -4 \qquad \frac{\partial f}{\partial z} = 3$$

(**backward** pass)

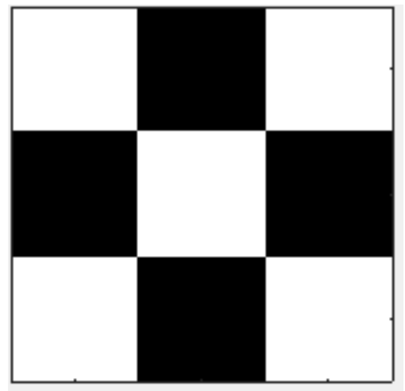
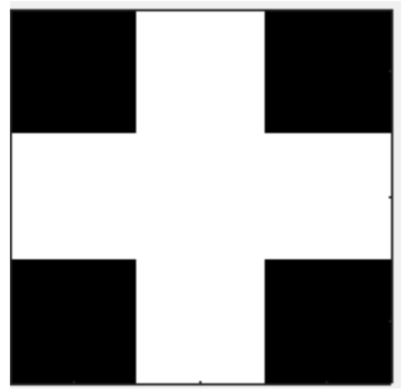
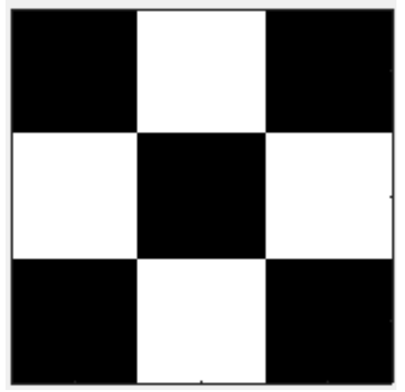
Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

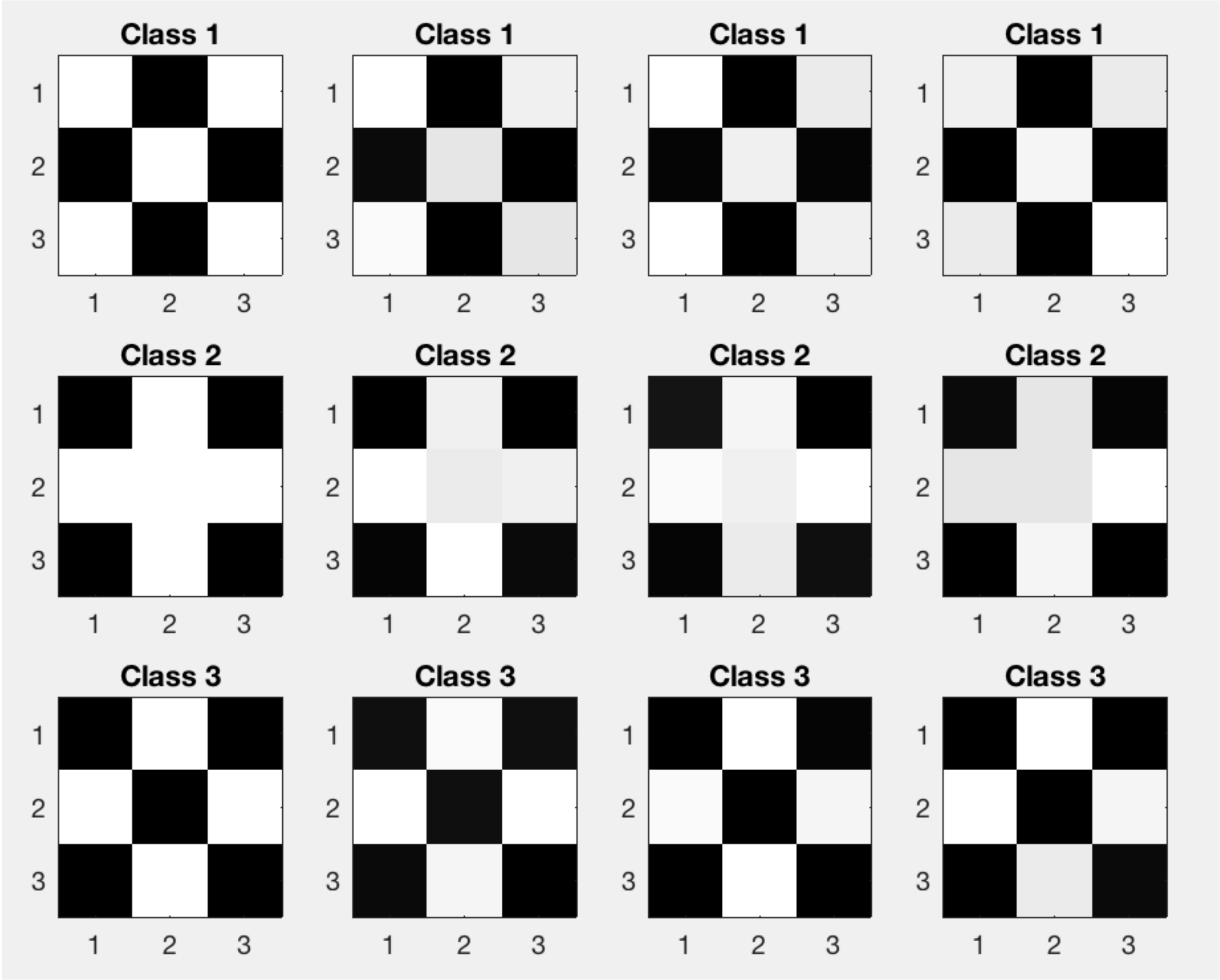




Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

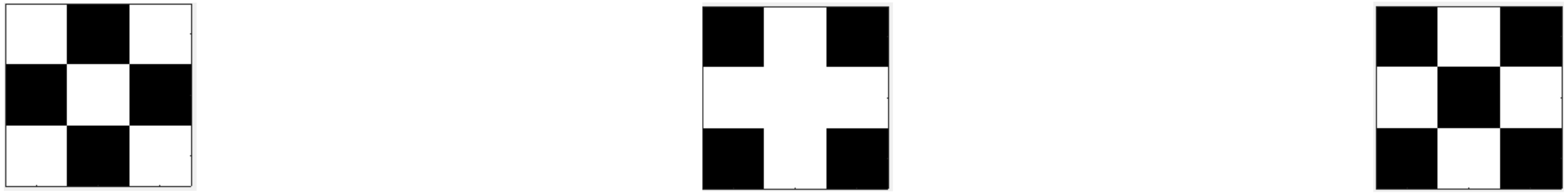




We will need some labeled data



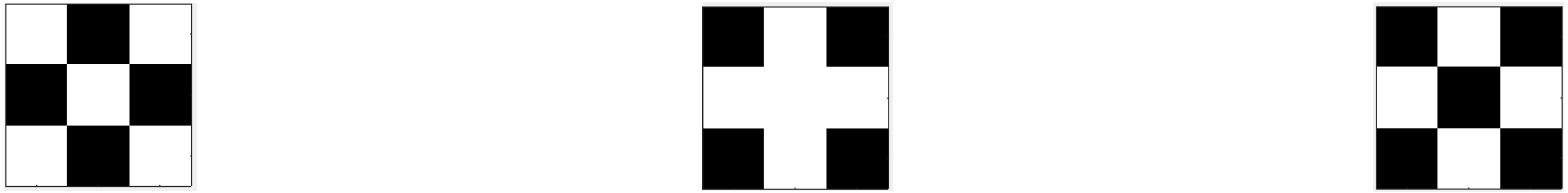
Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



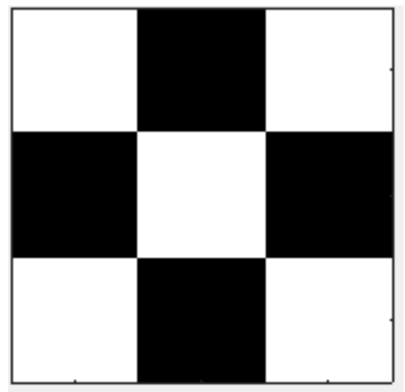
Example: Let's Build (world smallest) Neural Network

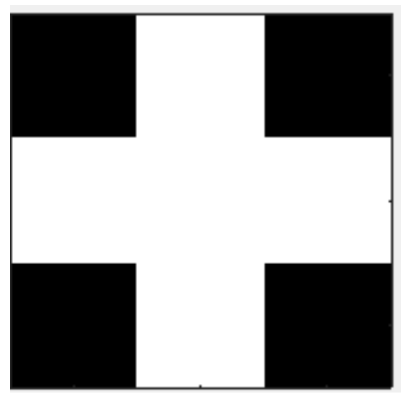
Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

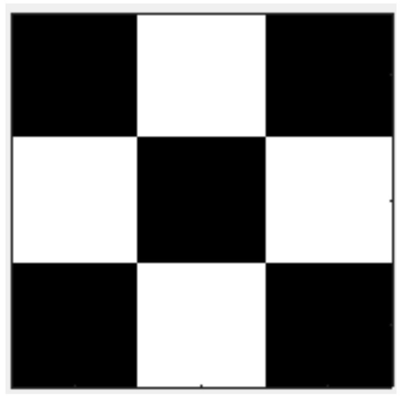


Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

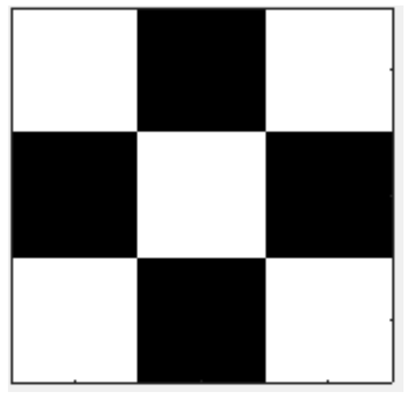
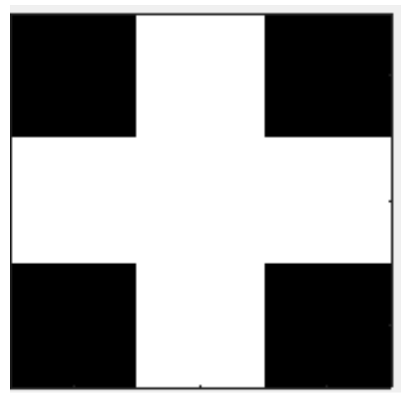
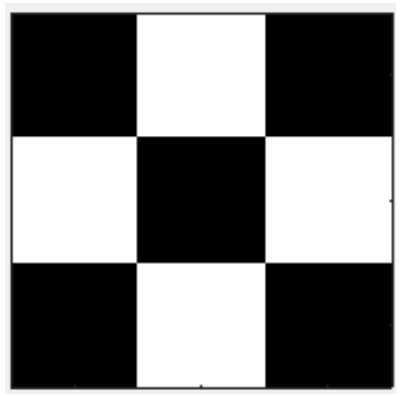






Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

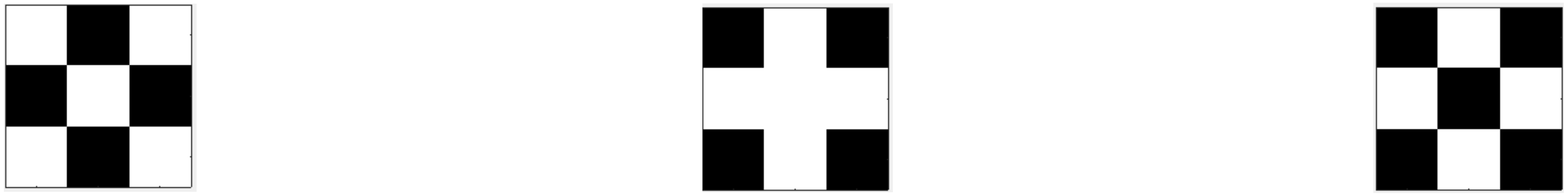




What do we need to do?

First, lets re-formulate the problem

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

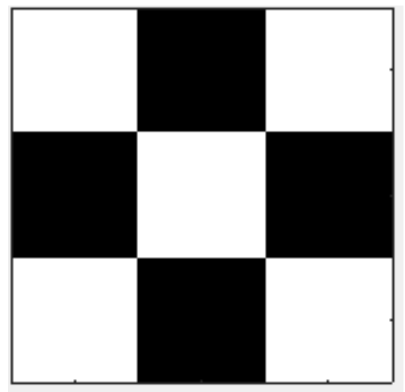


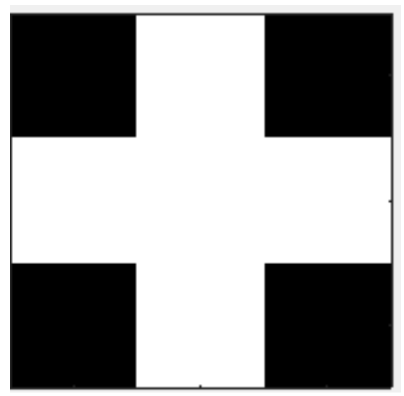
What do we need to do?

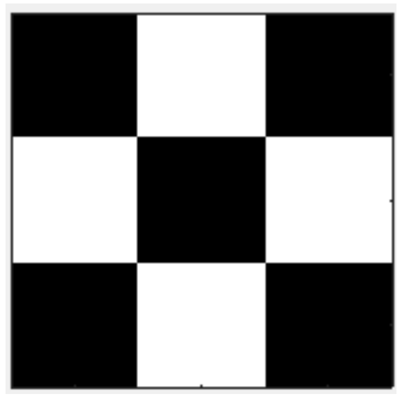
First, lets re-formulate the problem

Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images





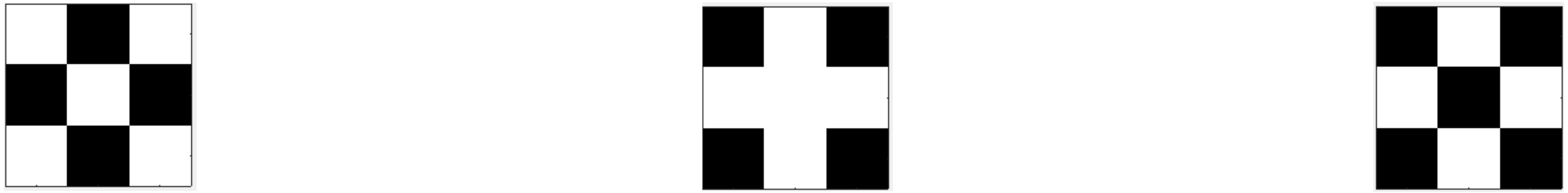


Now, lets build a **network**!

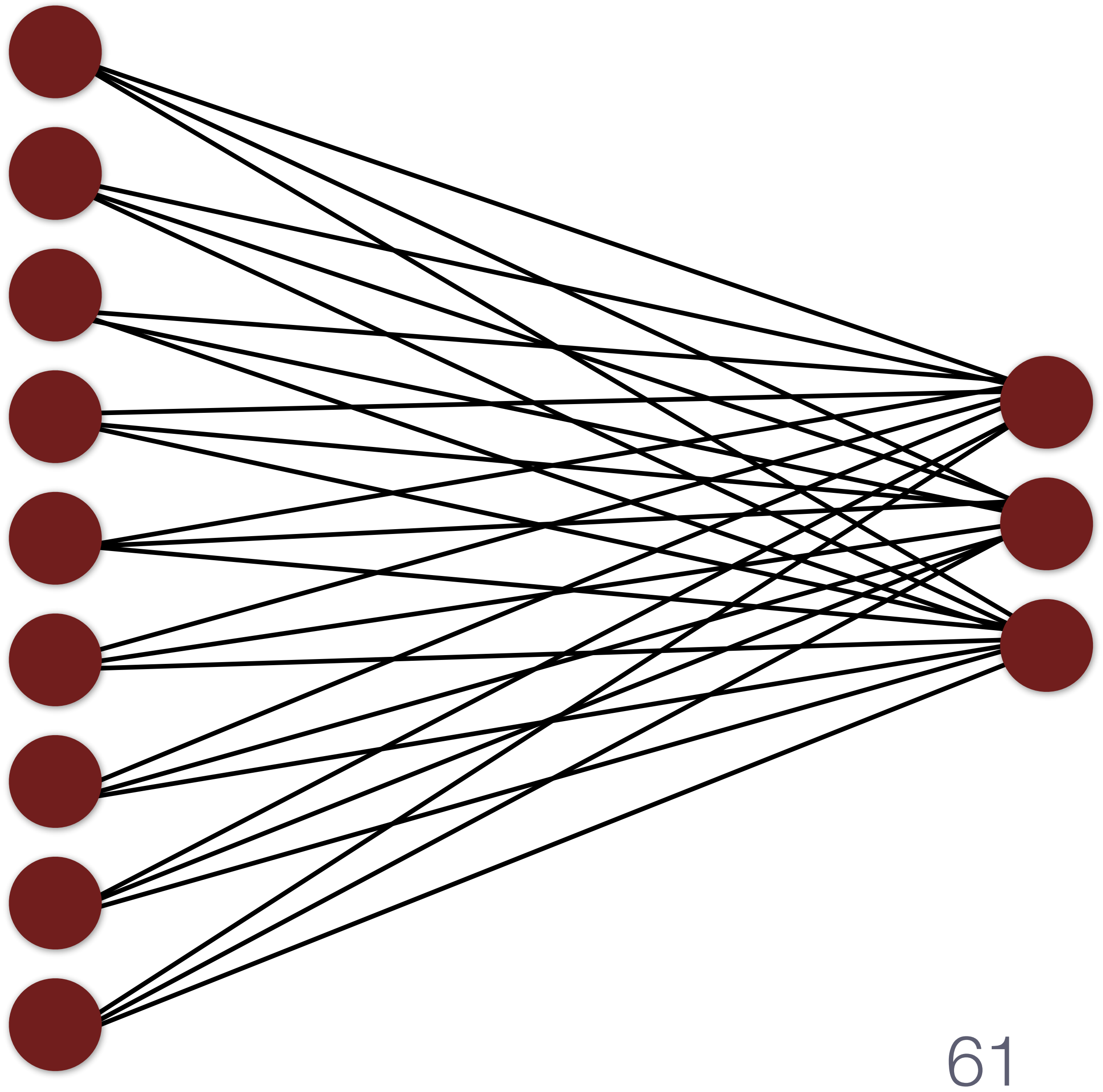
How many inputs should the network have? How neuron outputs?

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



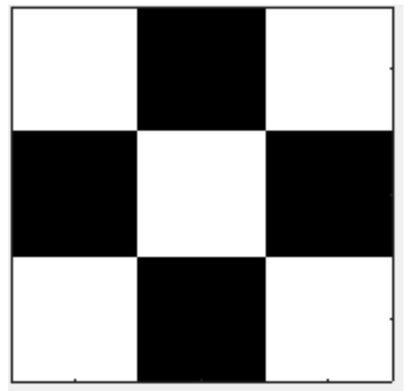
Input Layer **Output** Layer

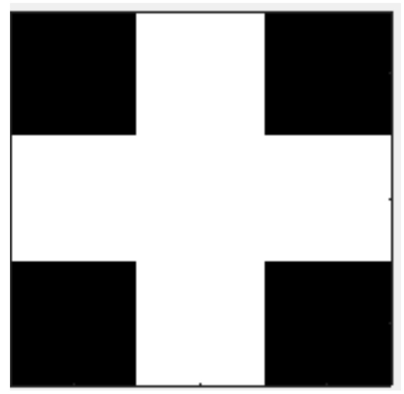


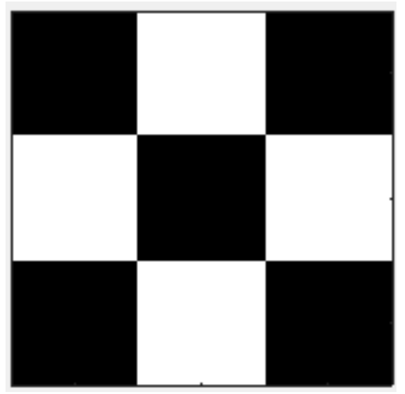
What else is missing for us to train it?

Example: Let's Build (world smallest) Neural Network

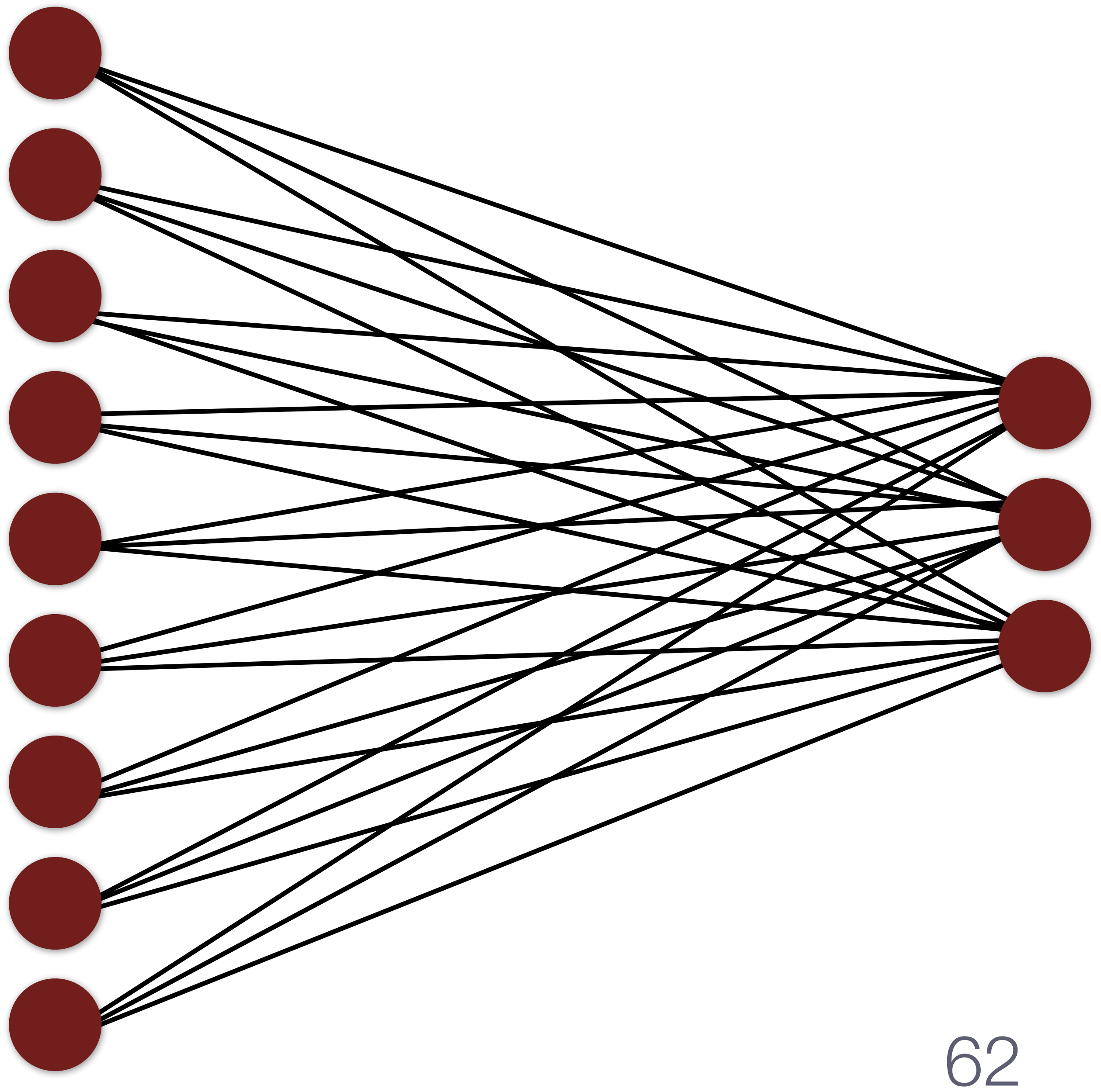
Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images







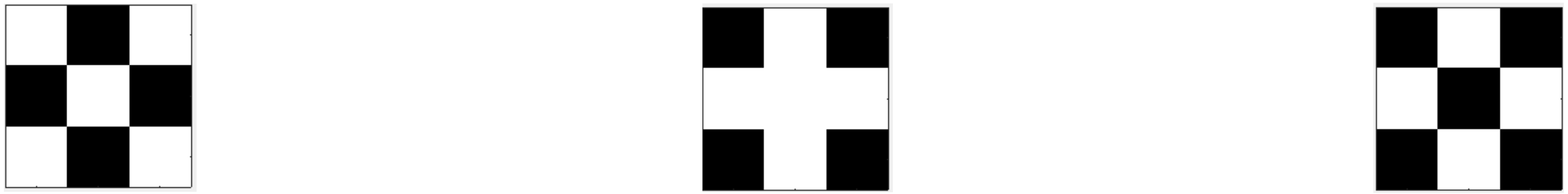
Input Layer **Output** Layer **Loss**



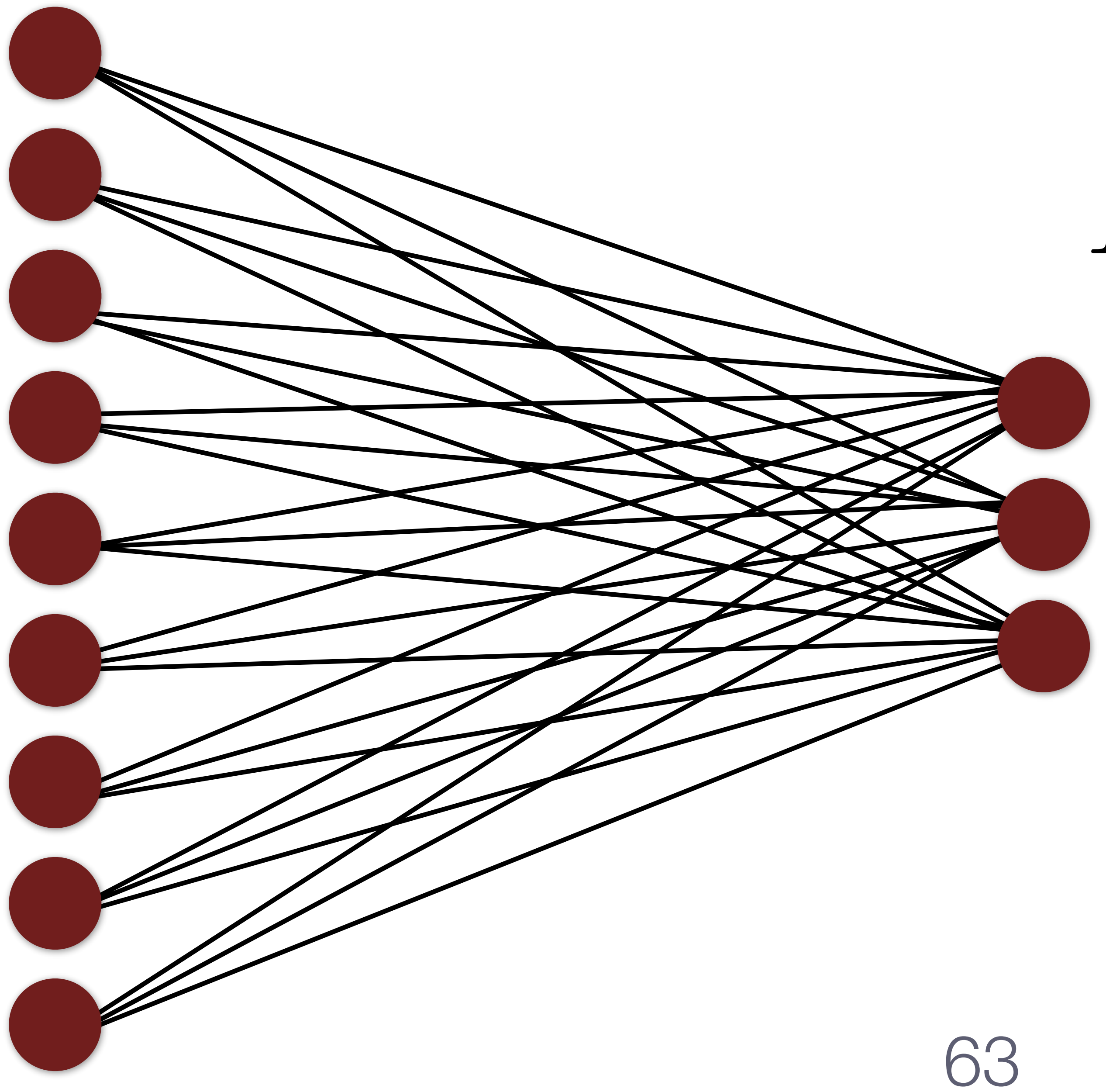
$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

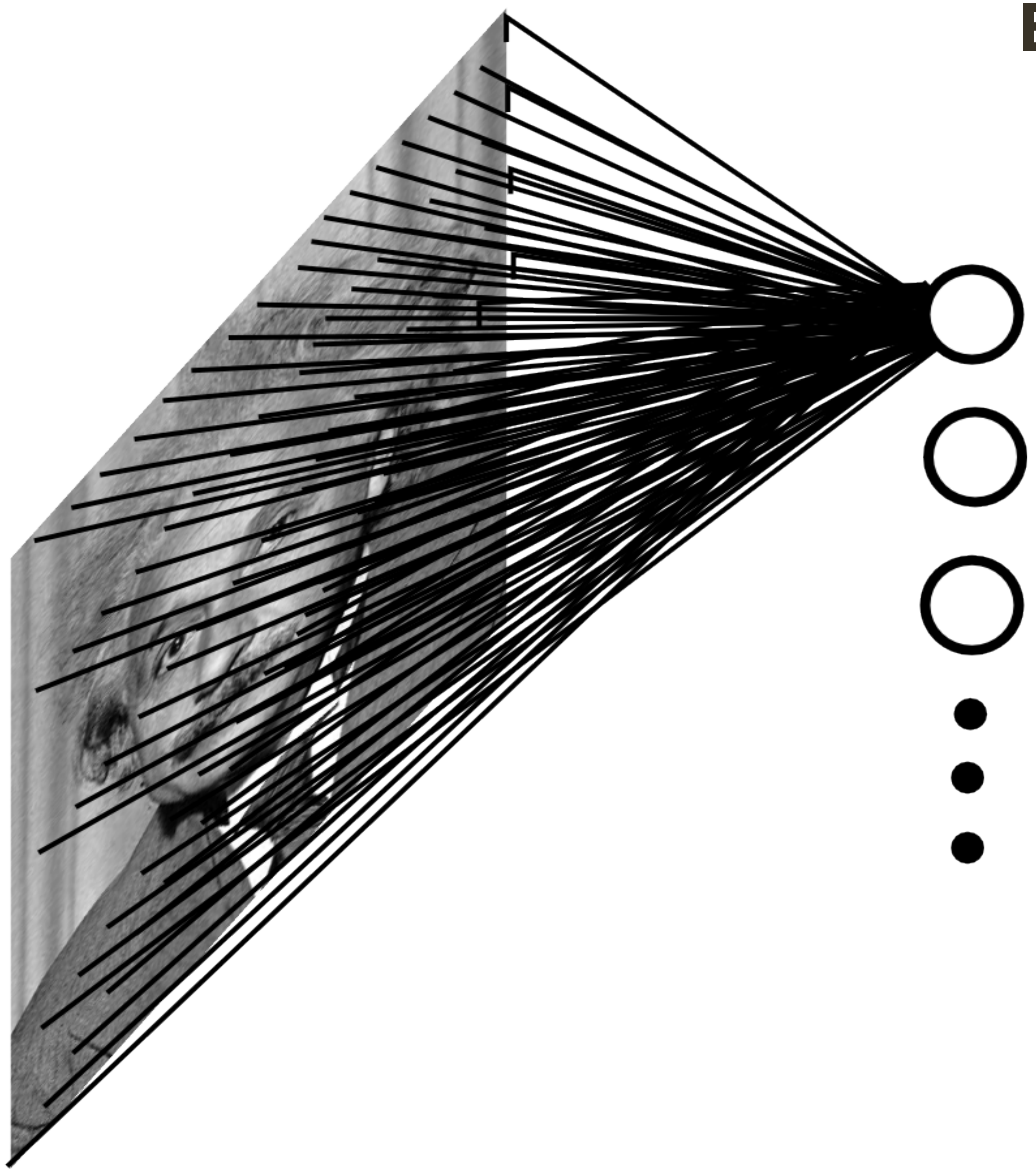


Input Layer **Output** Layer **Loss**



$$L_1 = -log\left(\frac{e^{\sum_{i=1}^{9}\sigma(w_{1,i}x_i+b_1)}}{\sum_{j=1}^{3}e^{\sum_{i=1}^{9}\sigma(w_{1,i}x_i+b_1)}}\right)$$

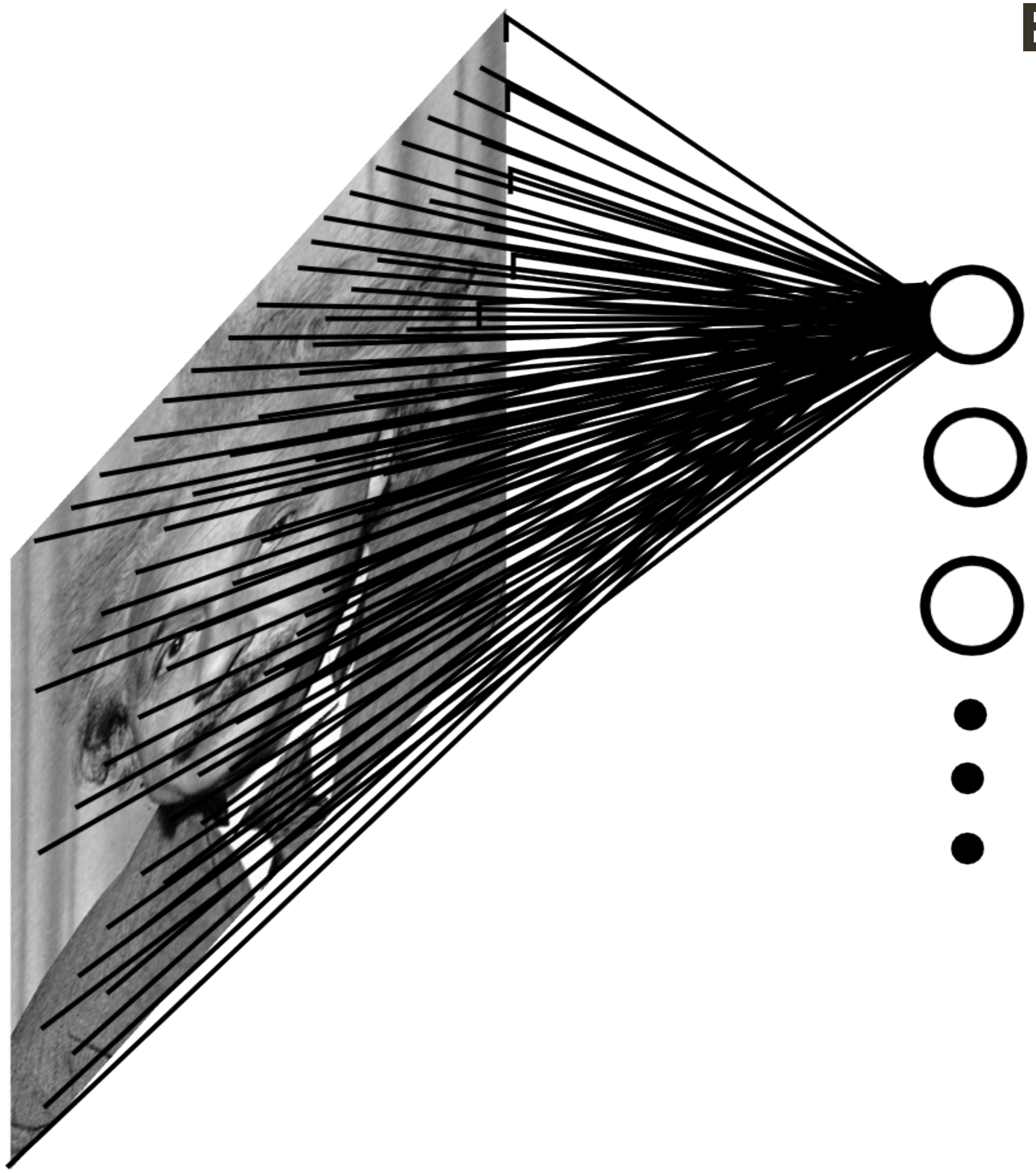
Fully Connected Layer



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

* slide from Marc'Aurelio Renzato

Fully Connected Layer

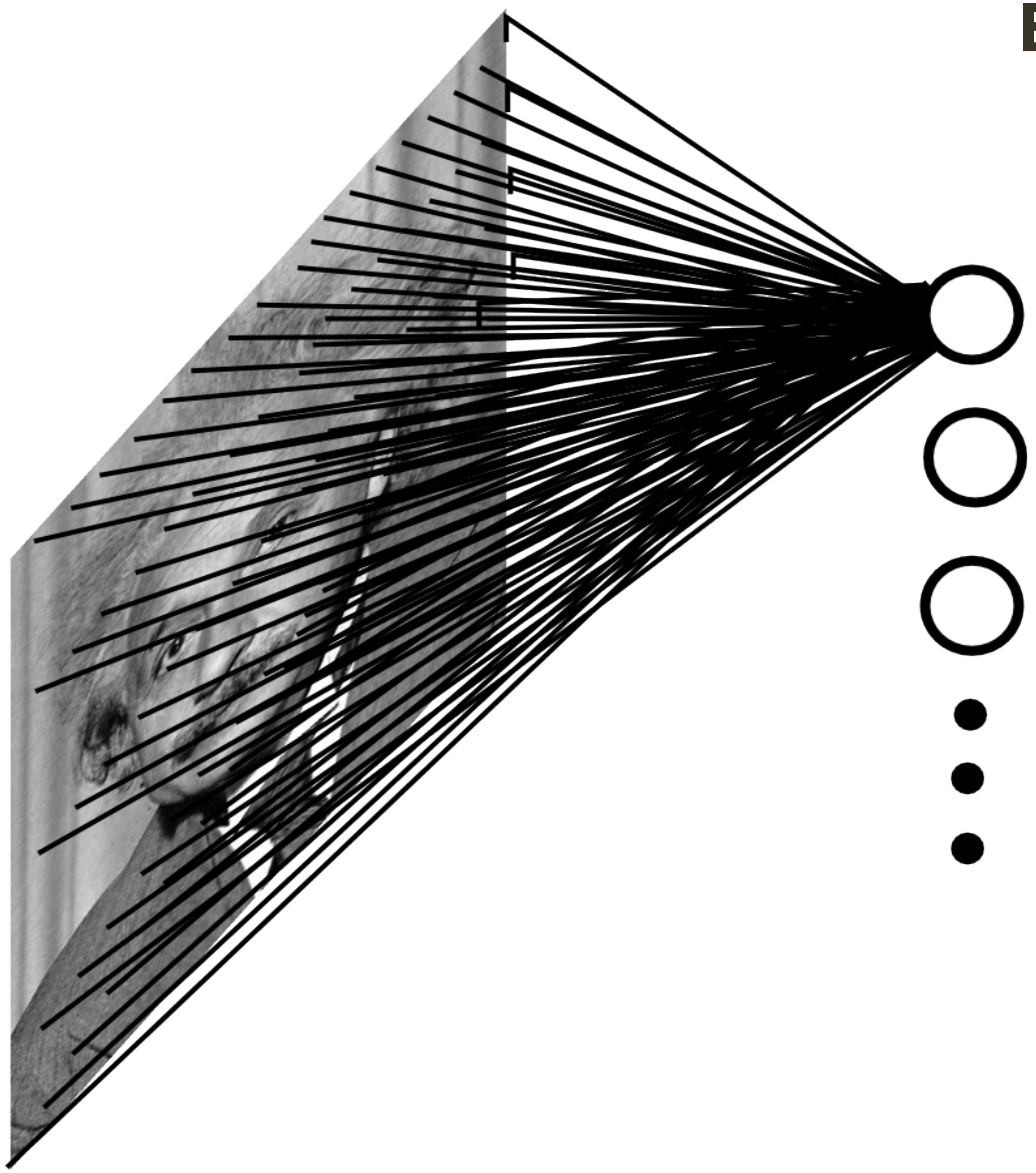


Example: 200 x 200 image (small)
x 40K hidden units

= ~ **2 Billion** parameters (for one layer!)

* slide from Marc'Aurelio Renzato

Fully Connected Layer



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

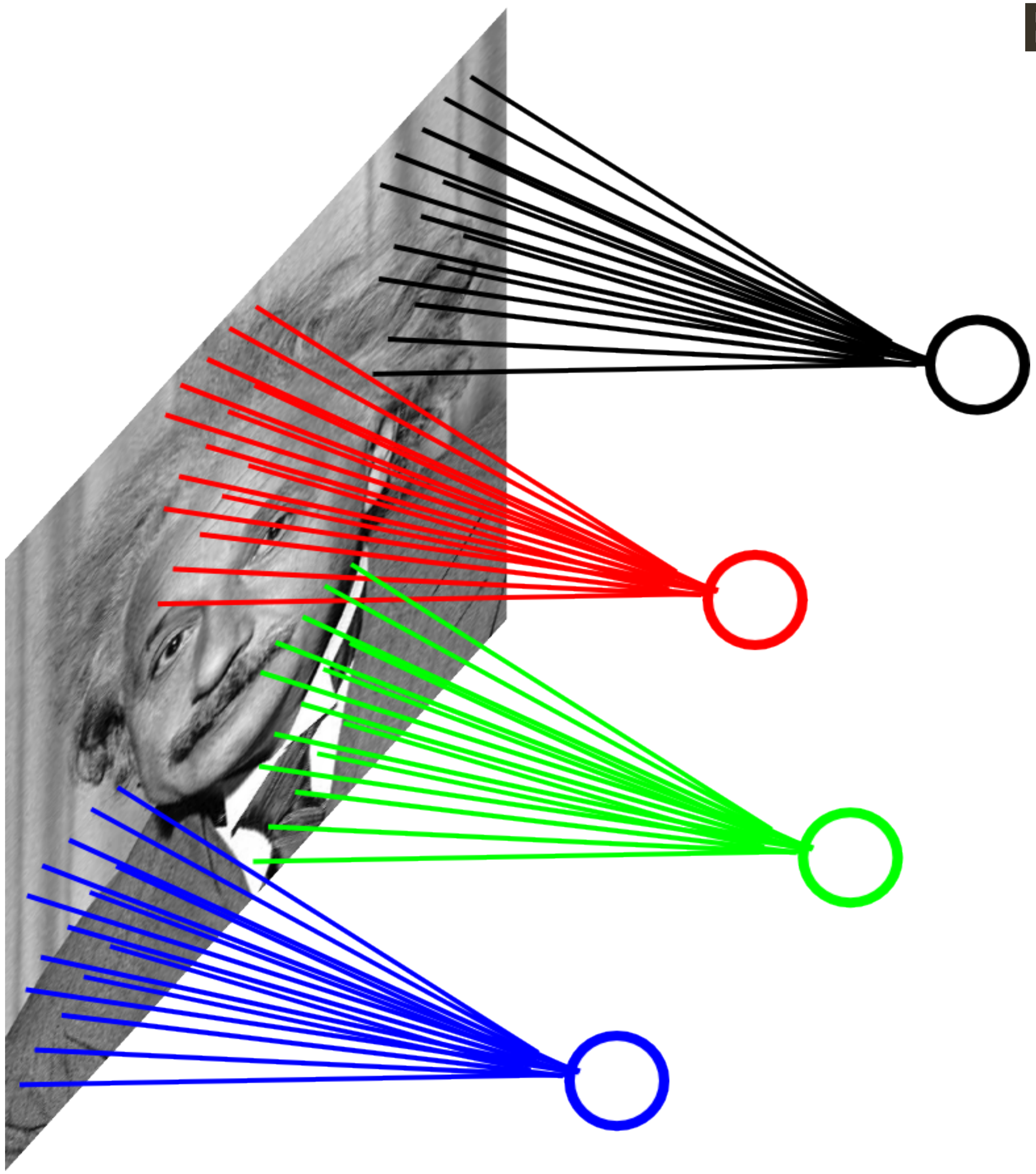
= ~ **2 Billion** parameters (for one layer!)

Spatial correlations are generally local

Waste of resources + we don't have enough data to train networks this large

* slide from Marc'Aurelio Renzato

Locally Connected Layer



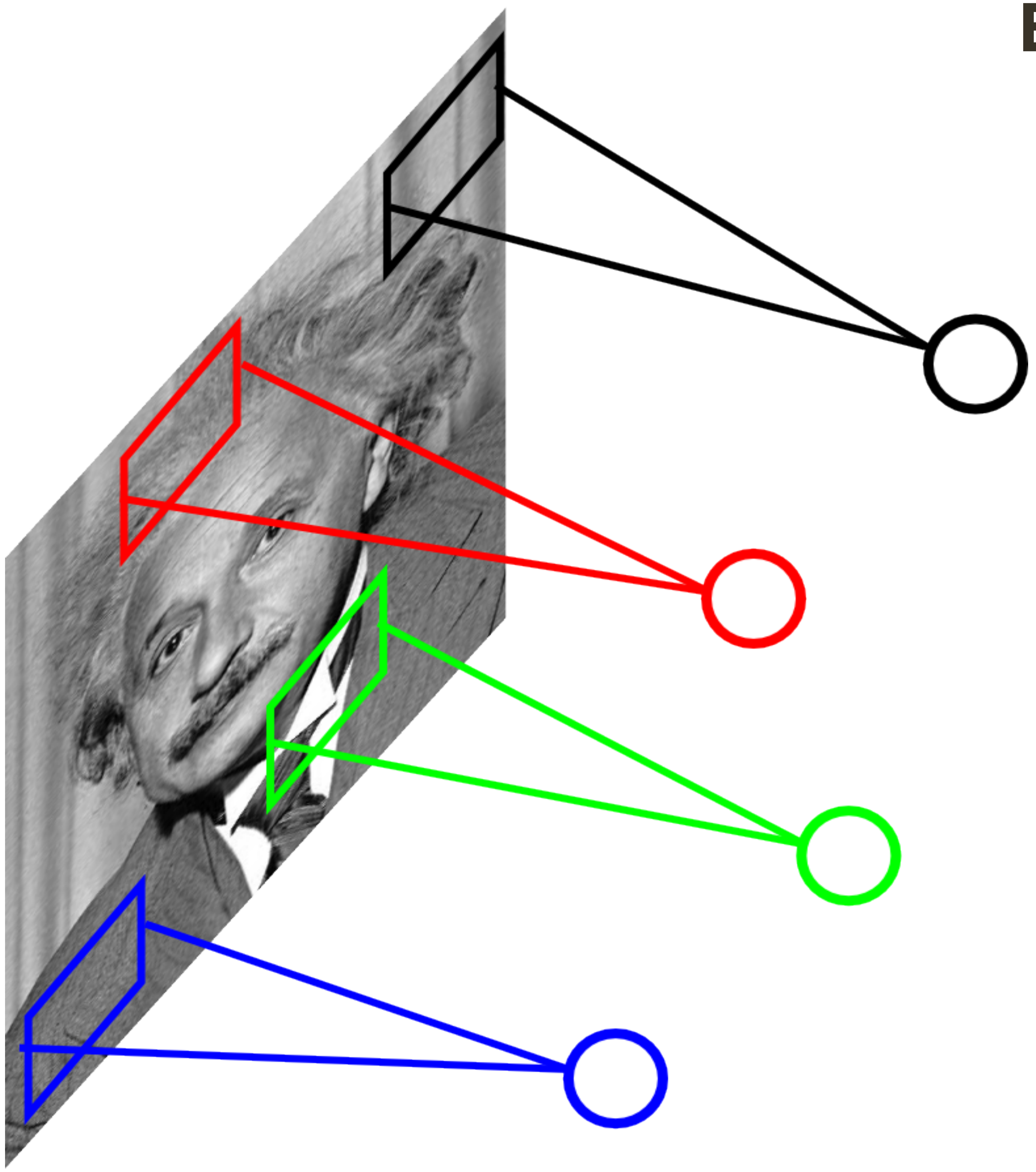
Example: 200 x 200 image (small)
x 40K hidden units

Filter size: 10 x 10

= ~ **4 Million** parameters

* slide from Marc'Aurelio Renzato

Locally Connected Layer



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

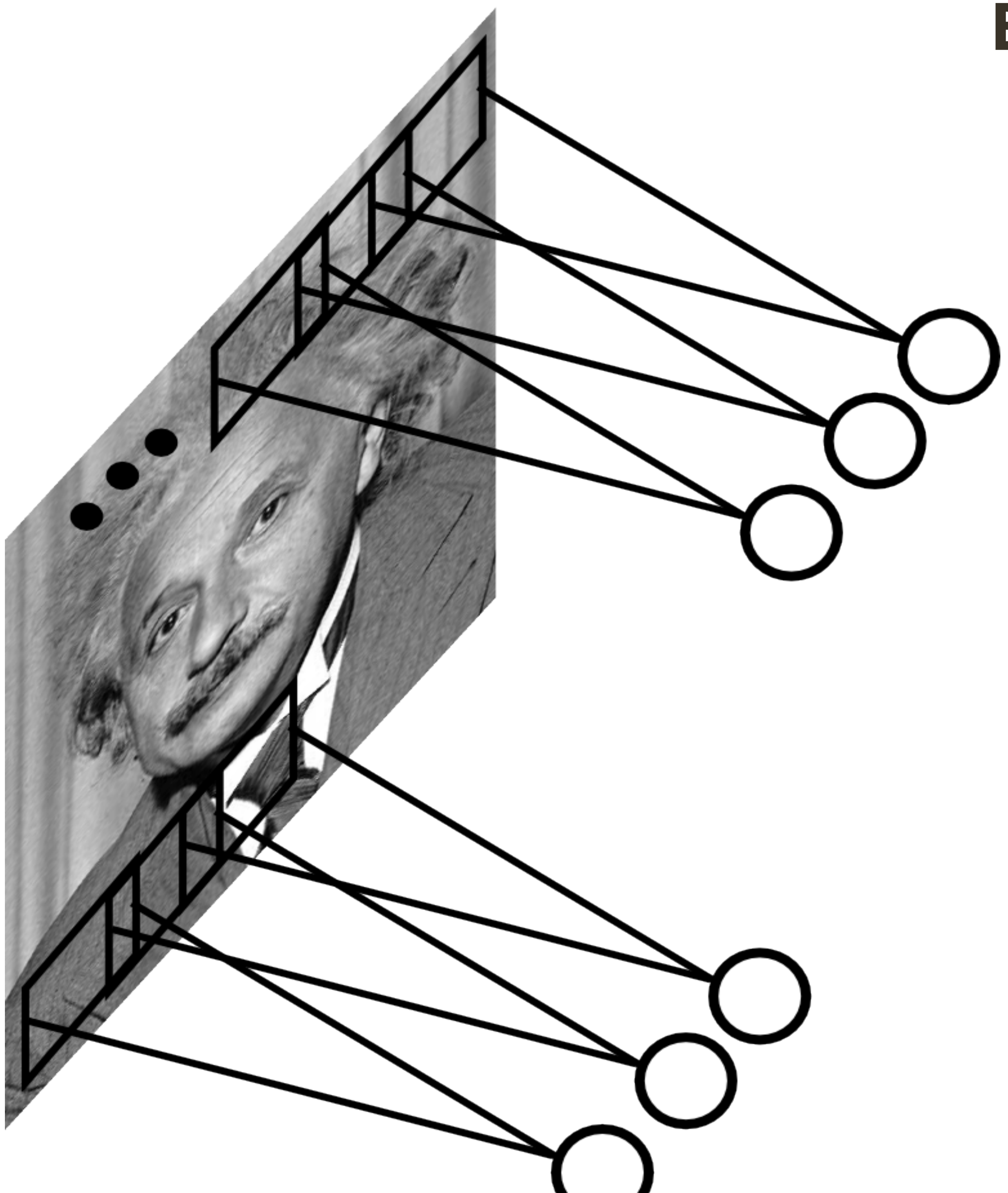
Filter size: 10 x 10

= ~ **4 Million** parameters

Stationarity — statistics is similar at different locations

* slide from Marc'Aurelio Renzato

Convolutional Layer



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

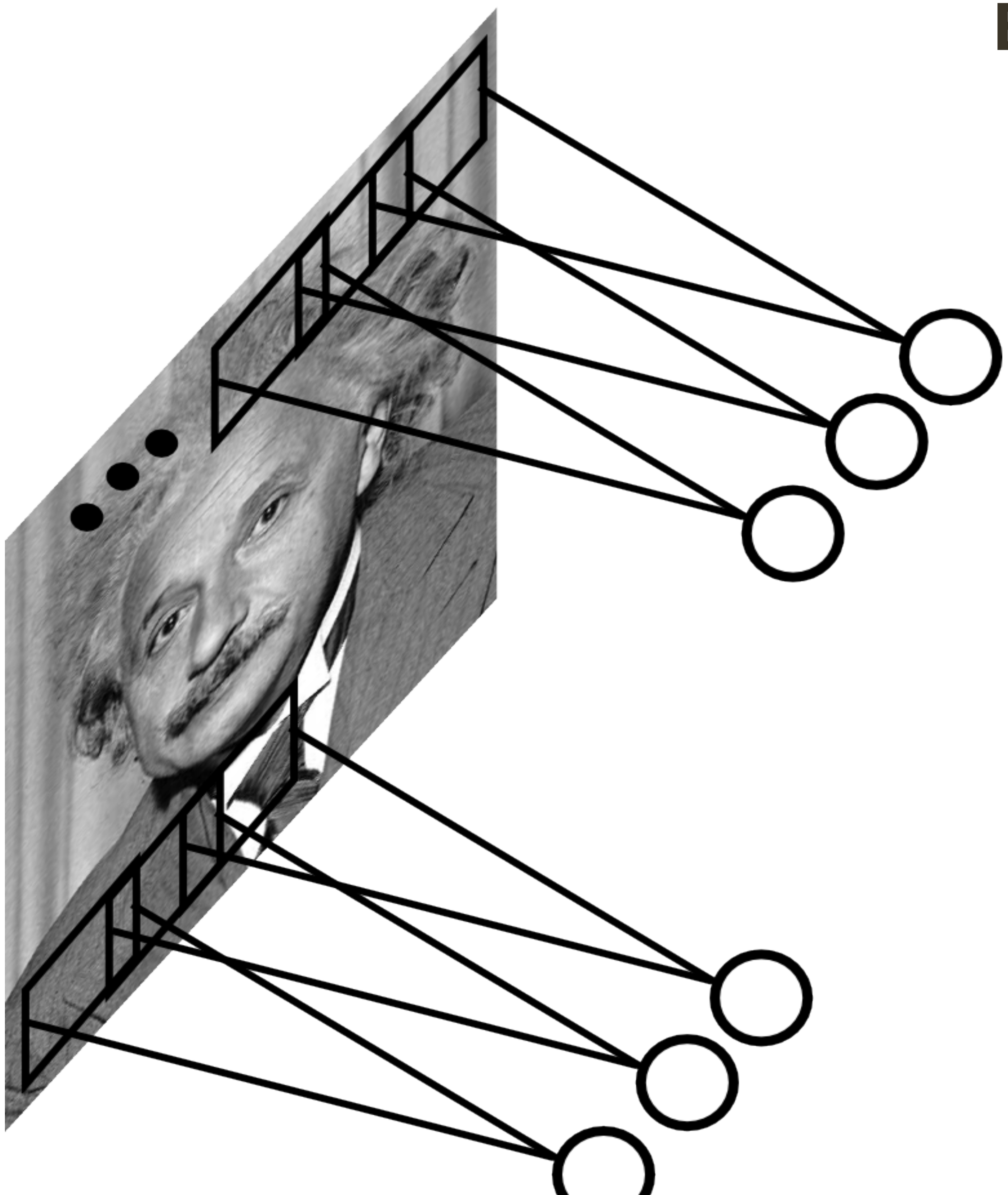
Filter size: 10 x 10

= ~ **4 Million** parameters

Share the same parameters across the locations (assuming input is stationary)

* slide adopted from Marc'Aurelio Renzato

Convolutional Layer



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

Filter size: 10 x 10

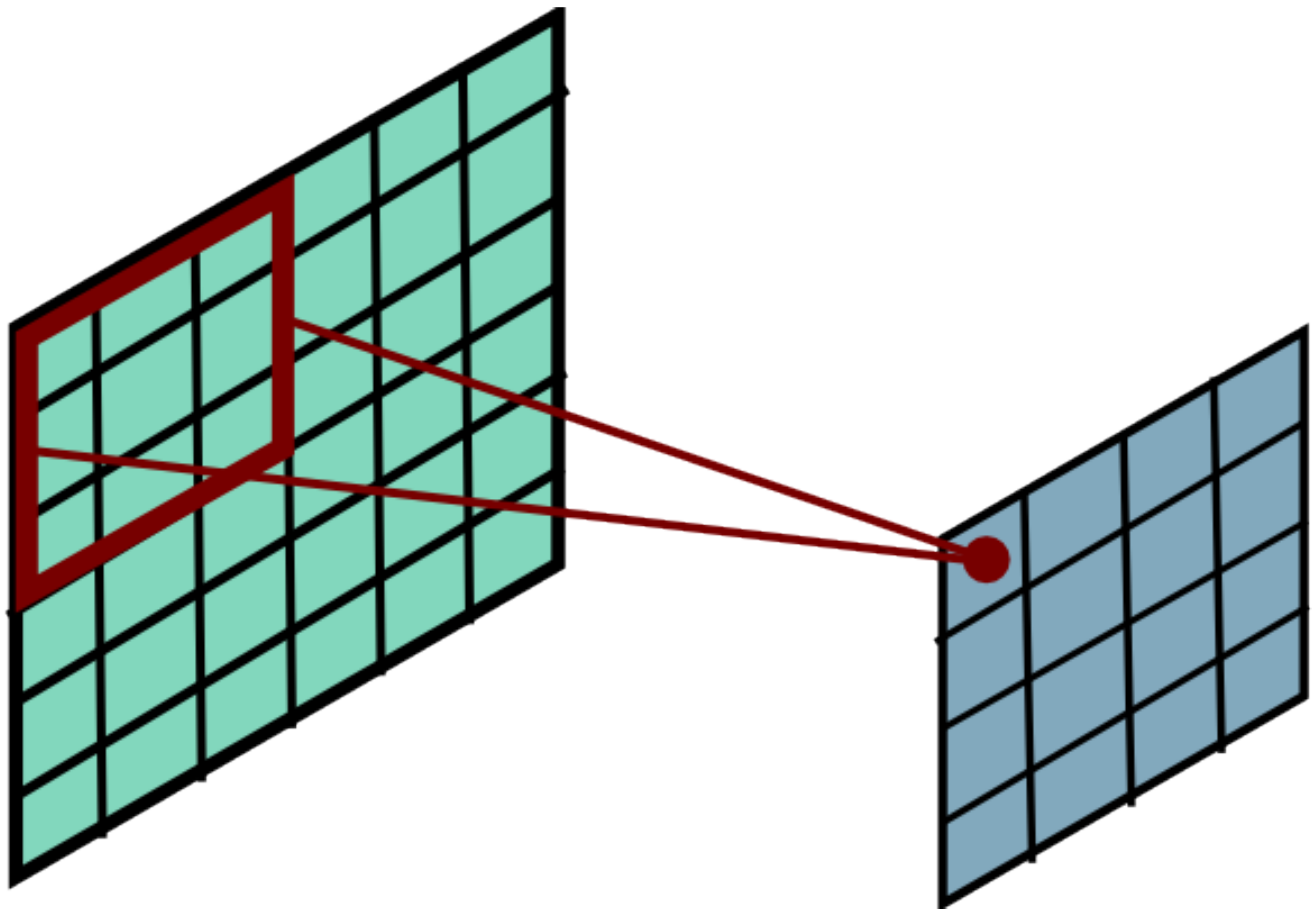
= ~ **~~4 Million~~** parameters

= 100 parameters

Share the same parameters across the locations (assuming input is stationary)

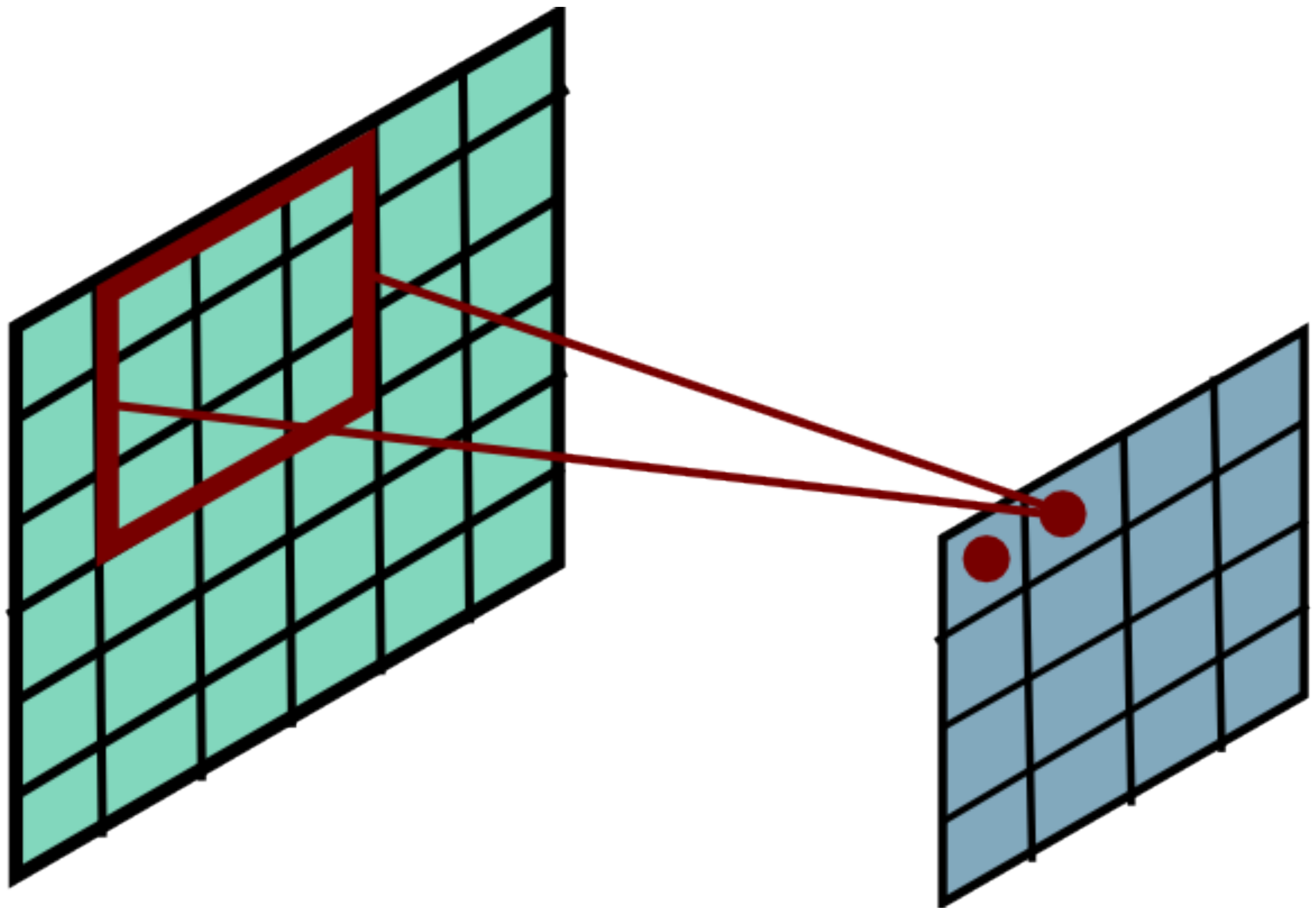
* slide adopted from Marc'Aurelio Renzato

Convolutional Layer



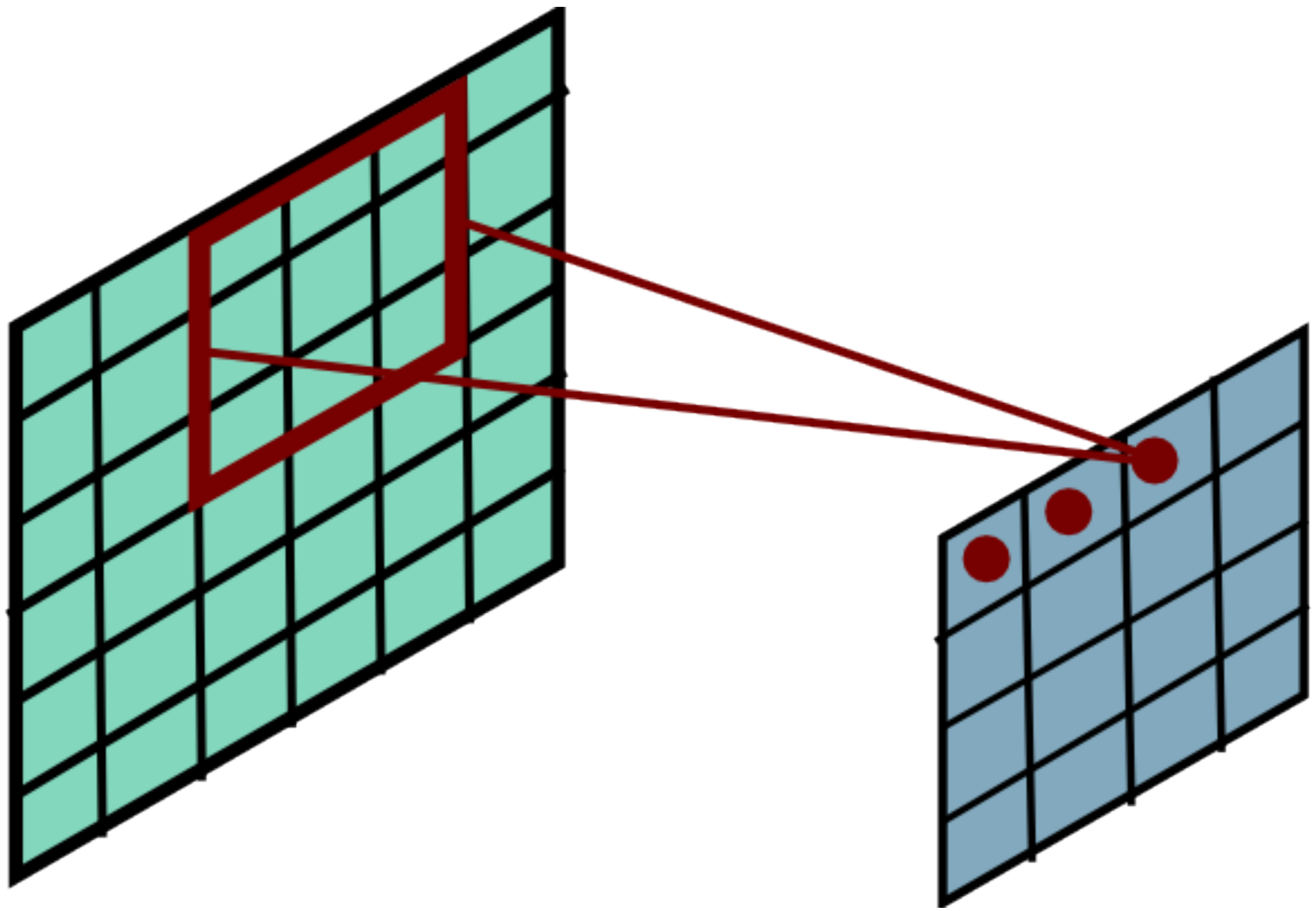
* slide from Marc'Aurelio Renzato

Convolutional Layer



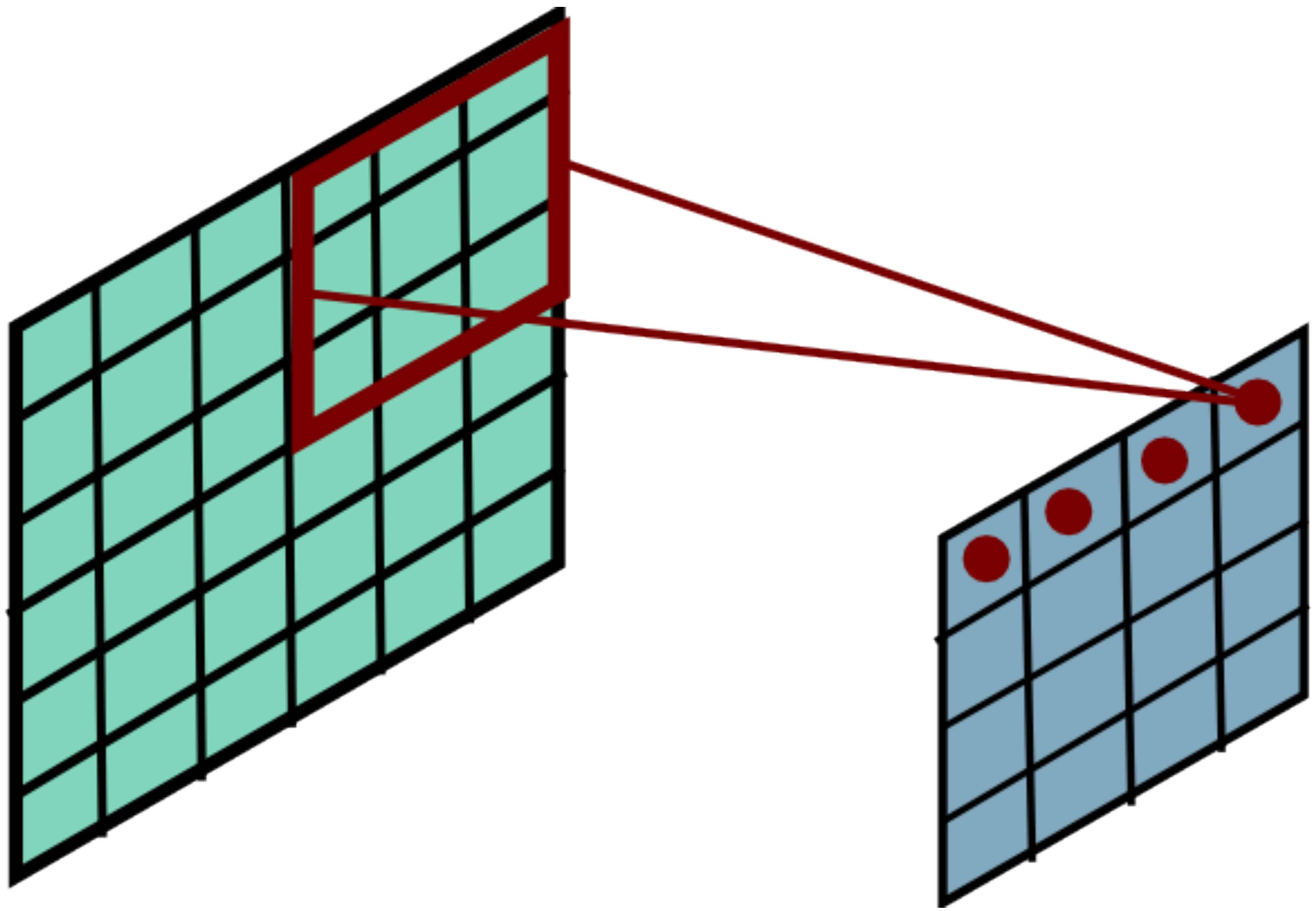
* slide from Marc'Aurelio Renzato

Convolutional Layer



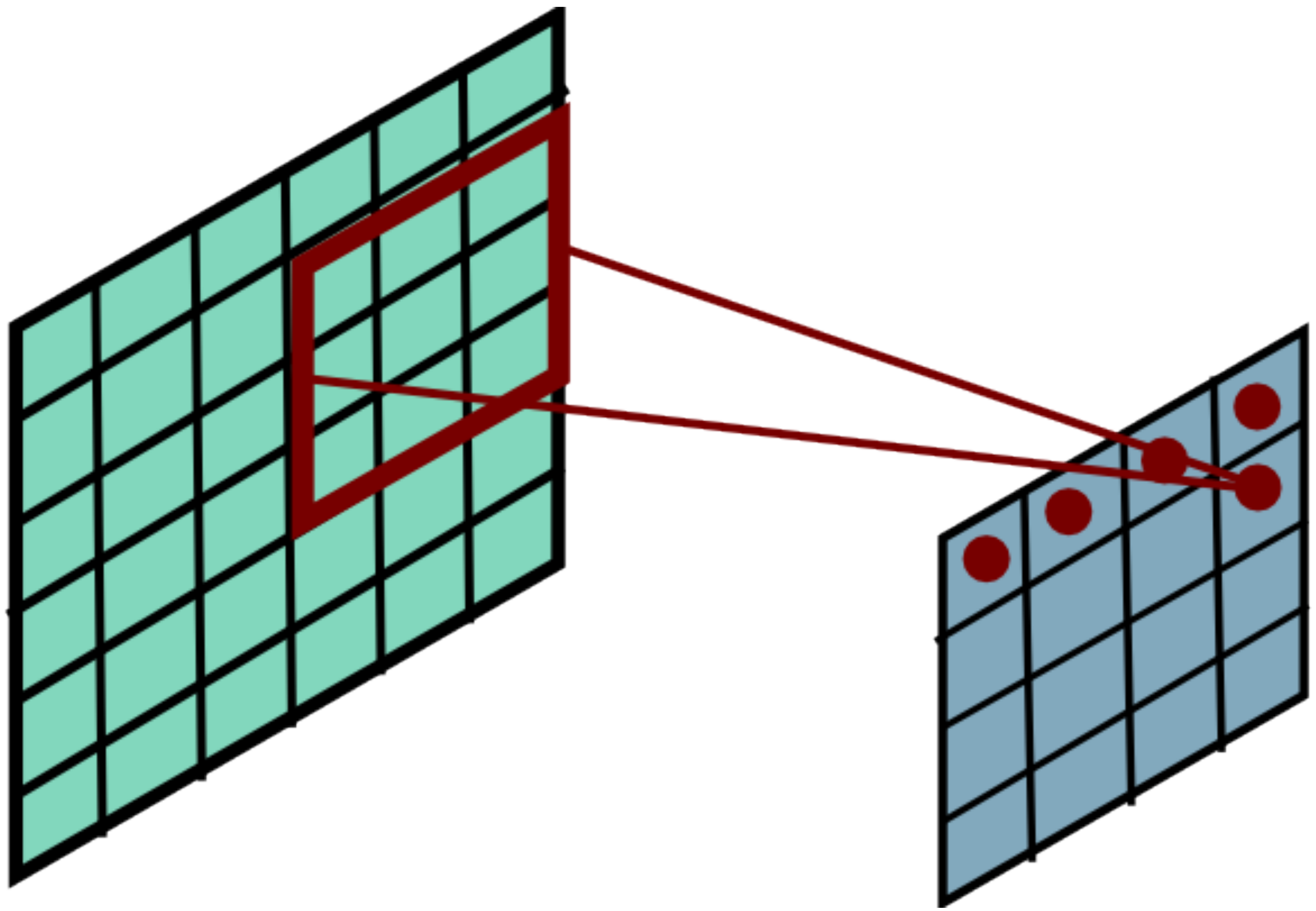
* slide from Marc'Aurelio Renzato

Convolutional Layer



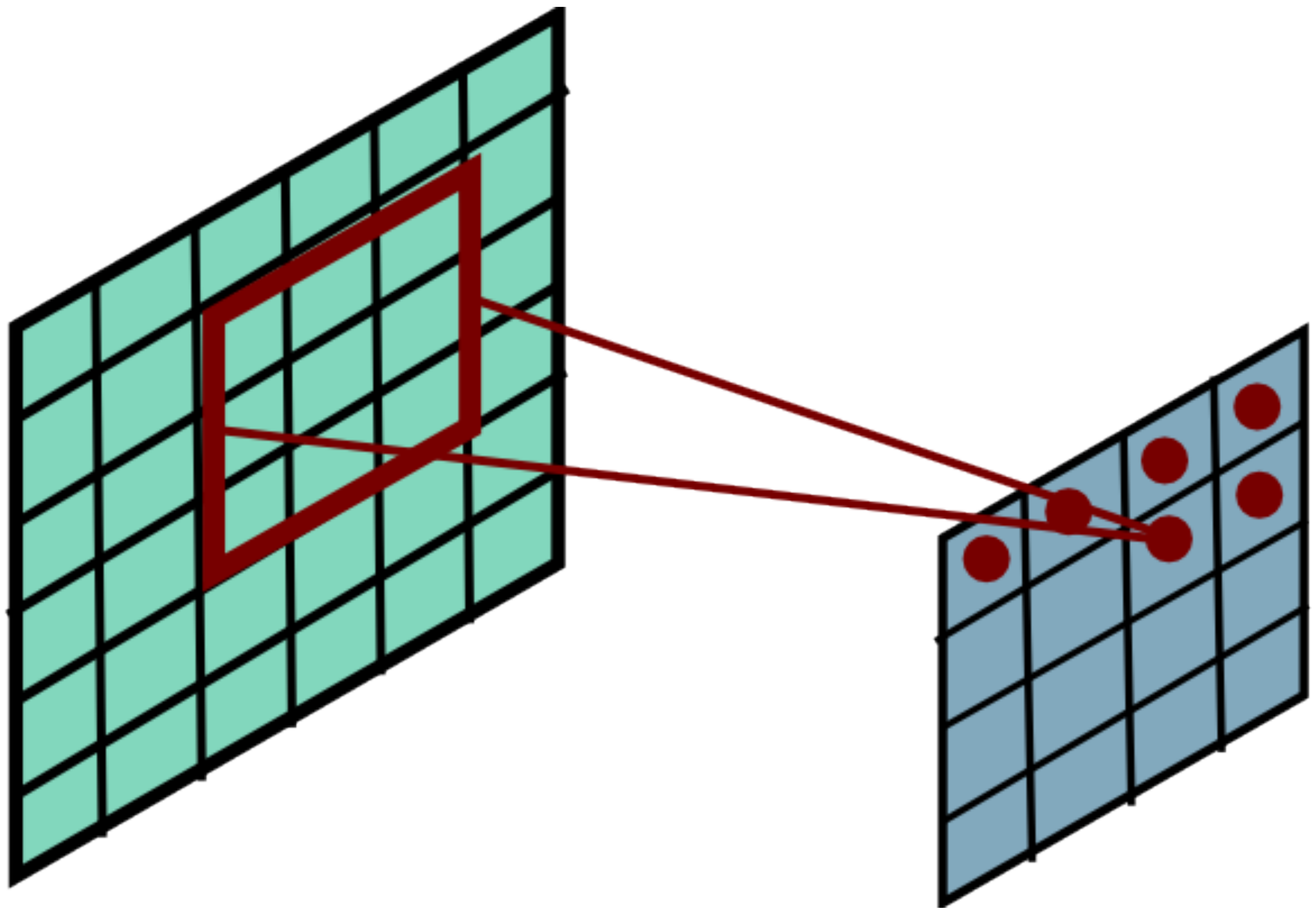
* slide from Marc'Aurelio Renzato

Convolutional Layer



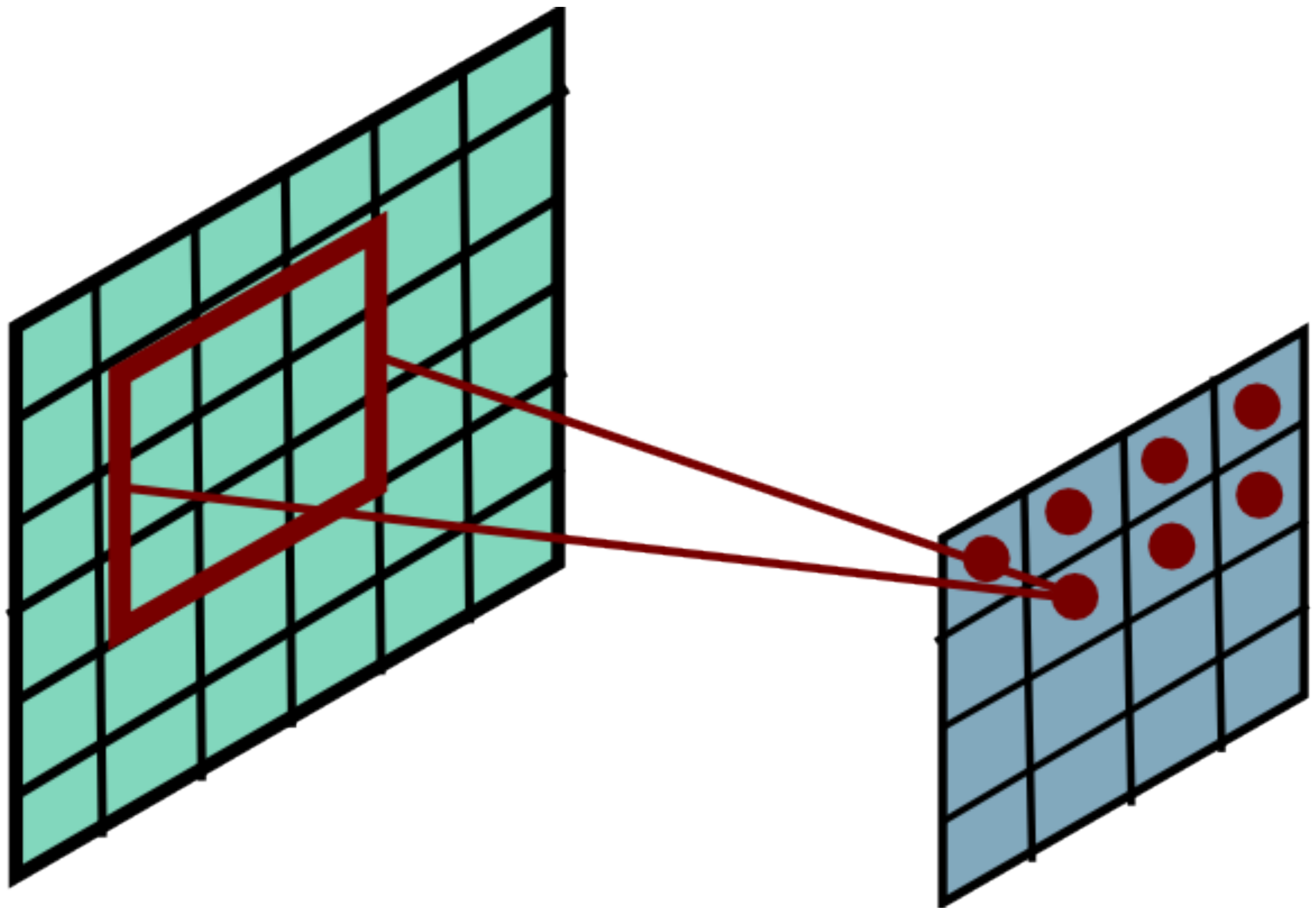
* slide from Marc'Aurelio Renzato

Convolutional Layer



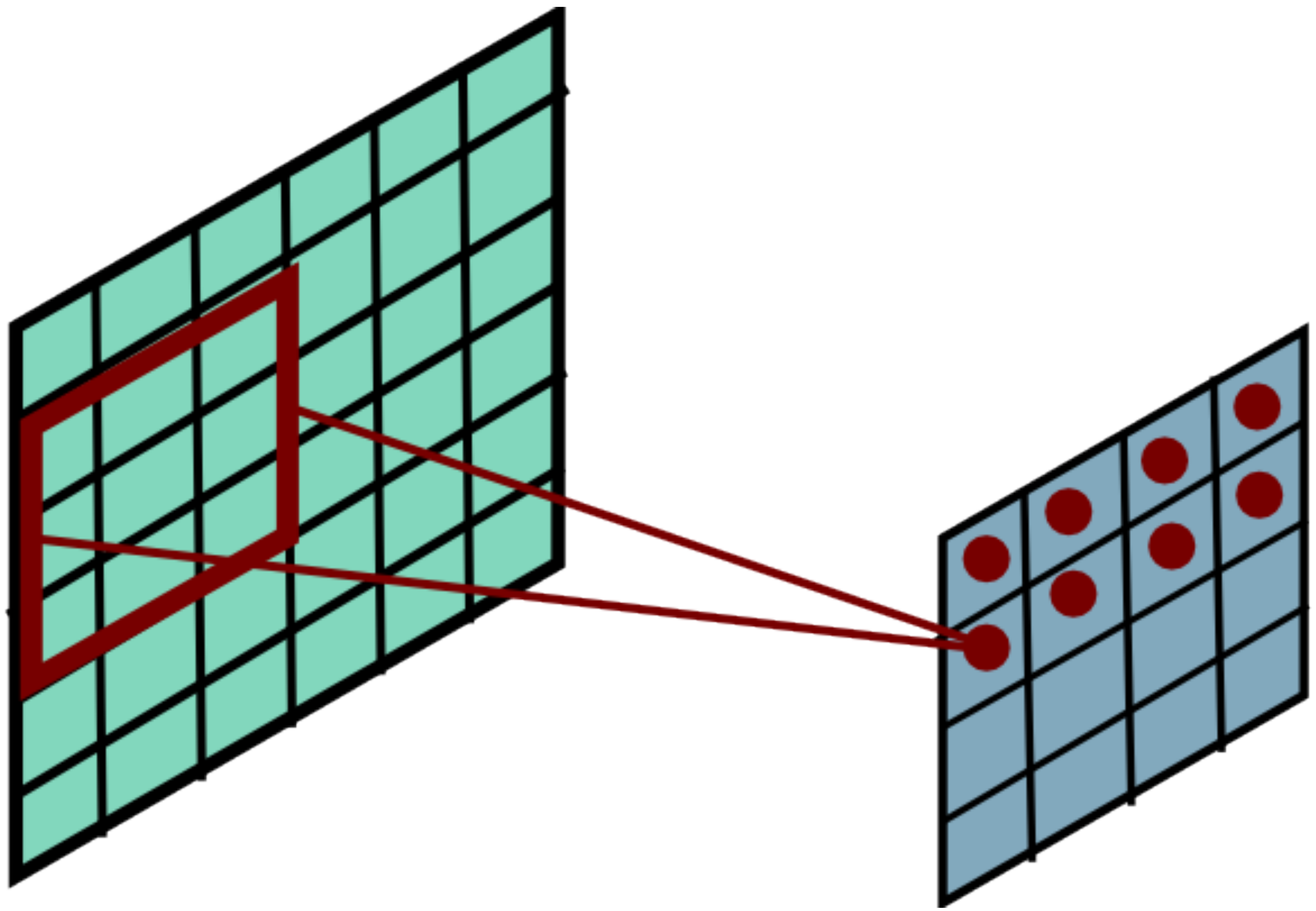
* slide from Marc'Aurelio Renzato

Convolutional Layer



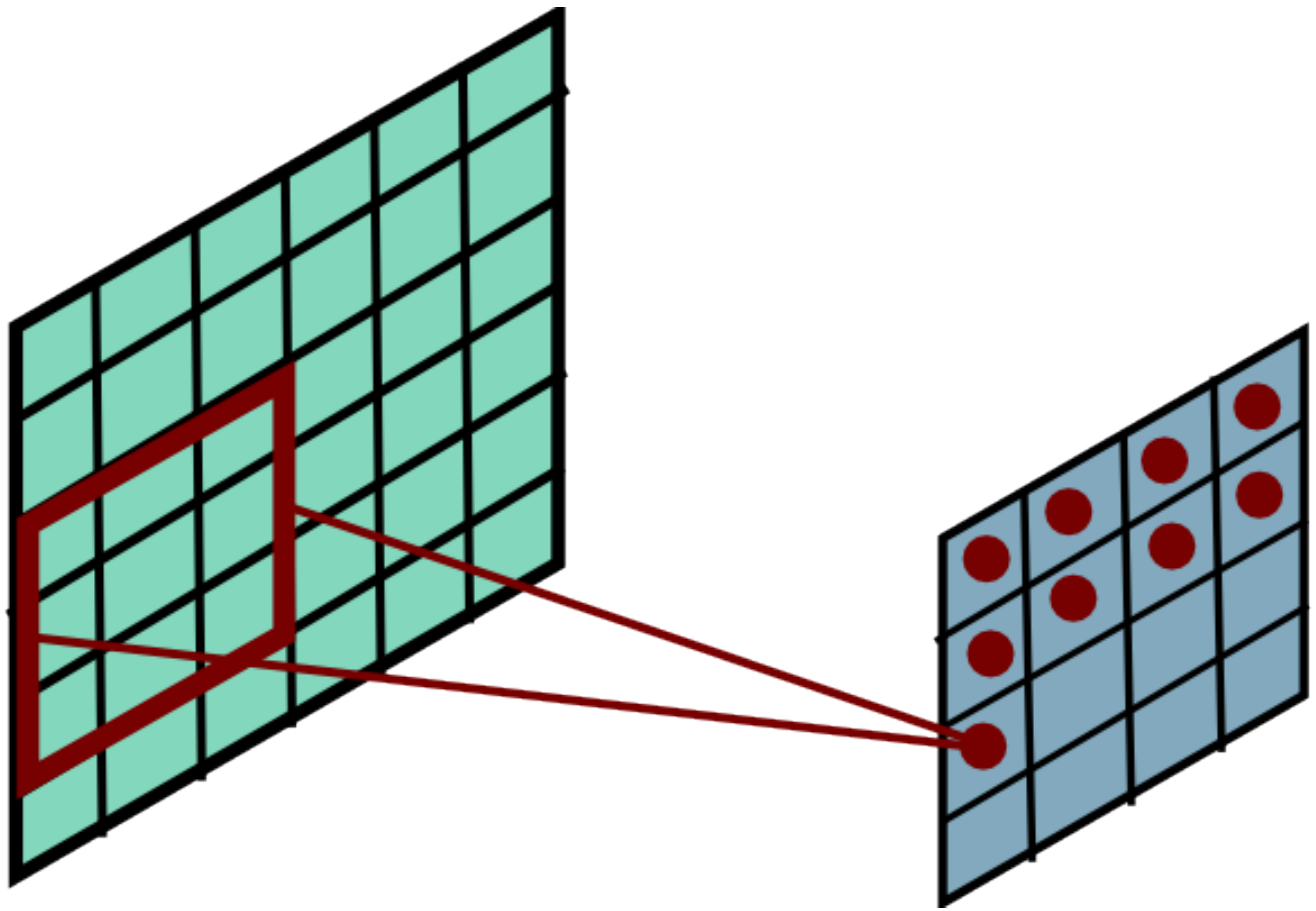
* slide from Marc'Aurelio Renzato

Convolutional Layer



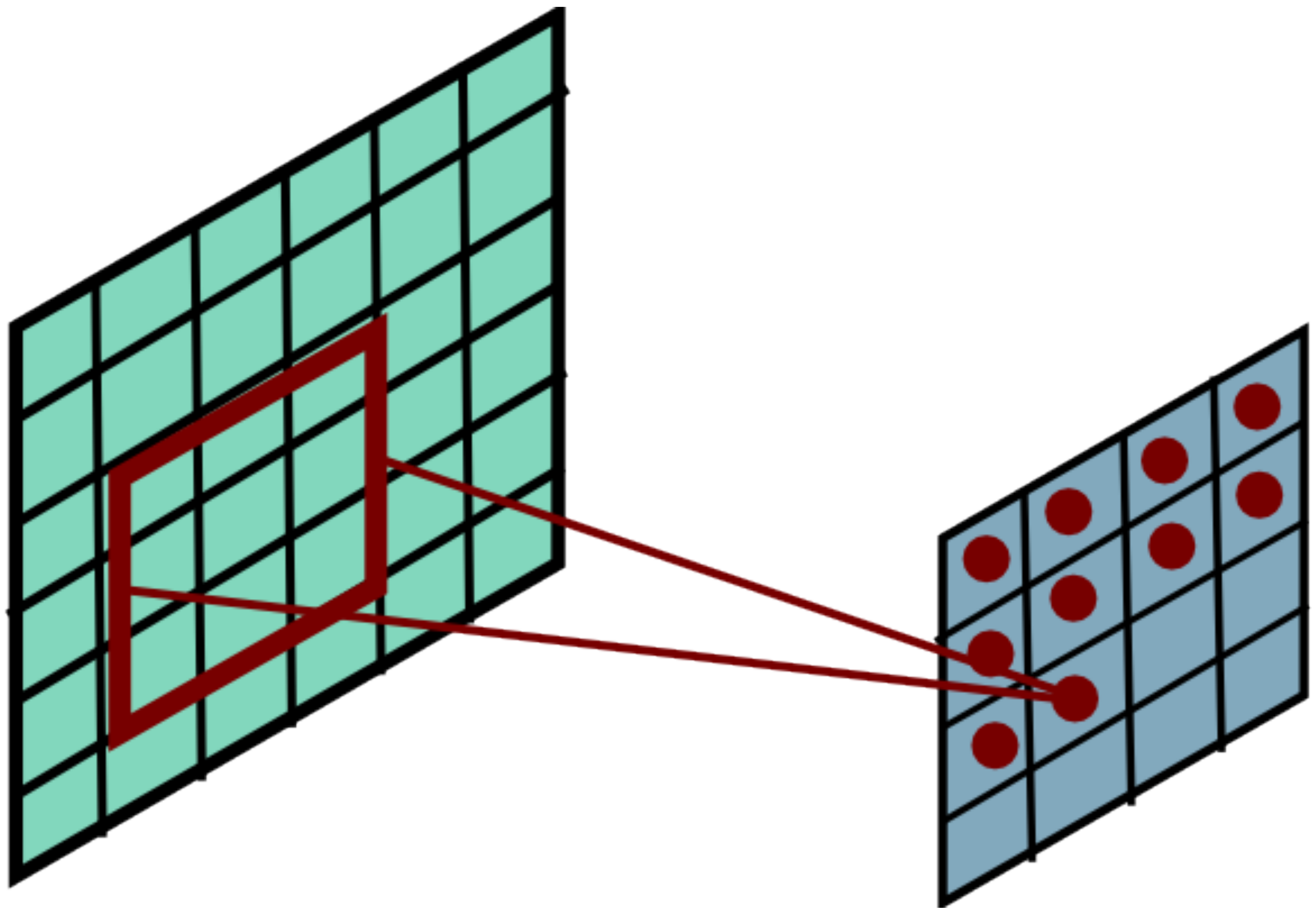
* slide from Marc'Aurelio Renzato

Convolutional Layer



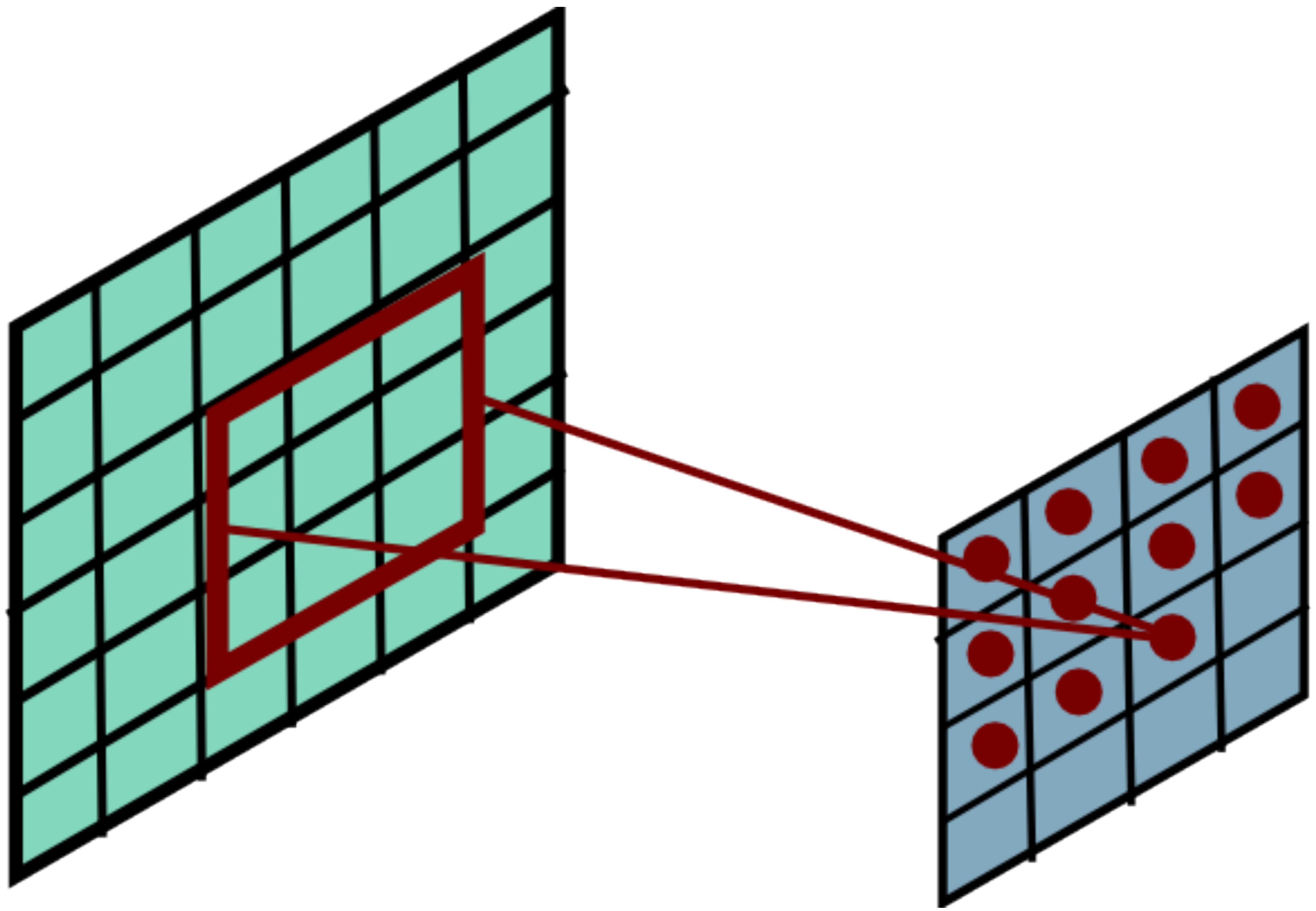
* slide from Marc'Aurelio Renzato

Convolutional Layer



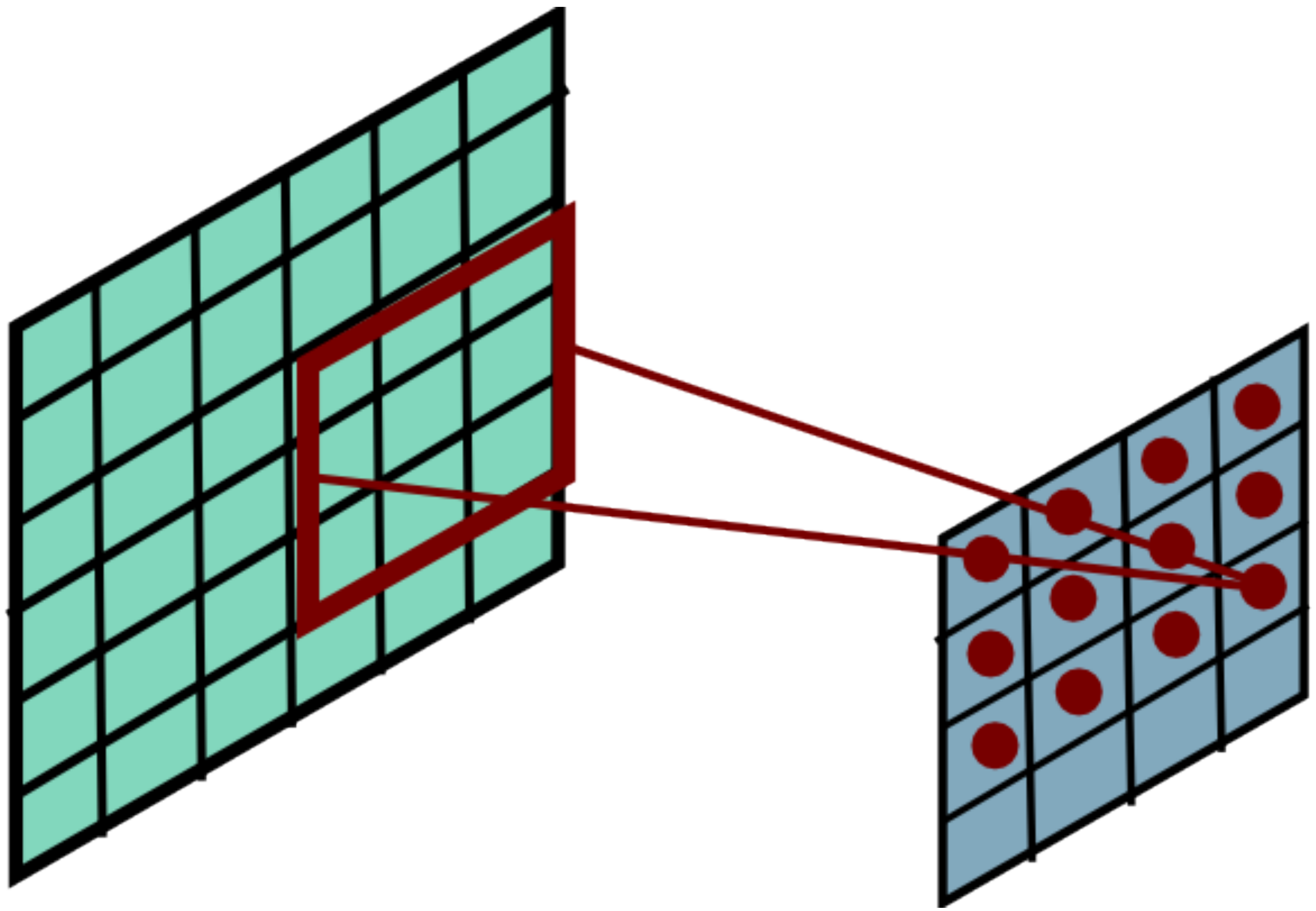
* slide from Marc'Aurelio Renzato

Convolutional Layer



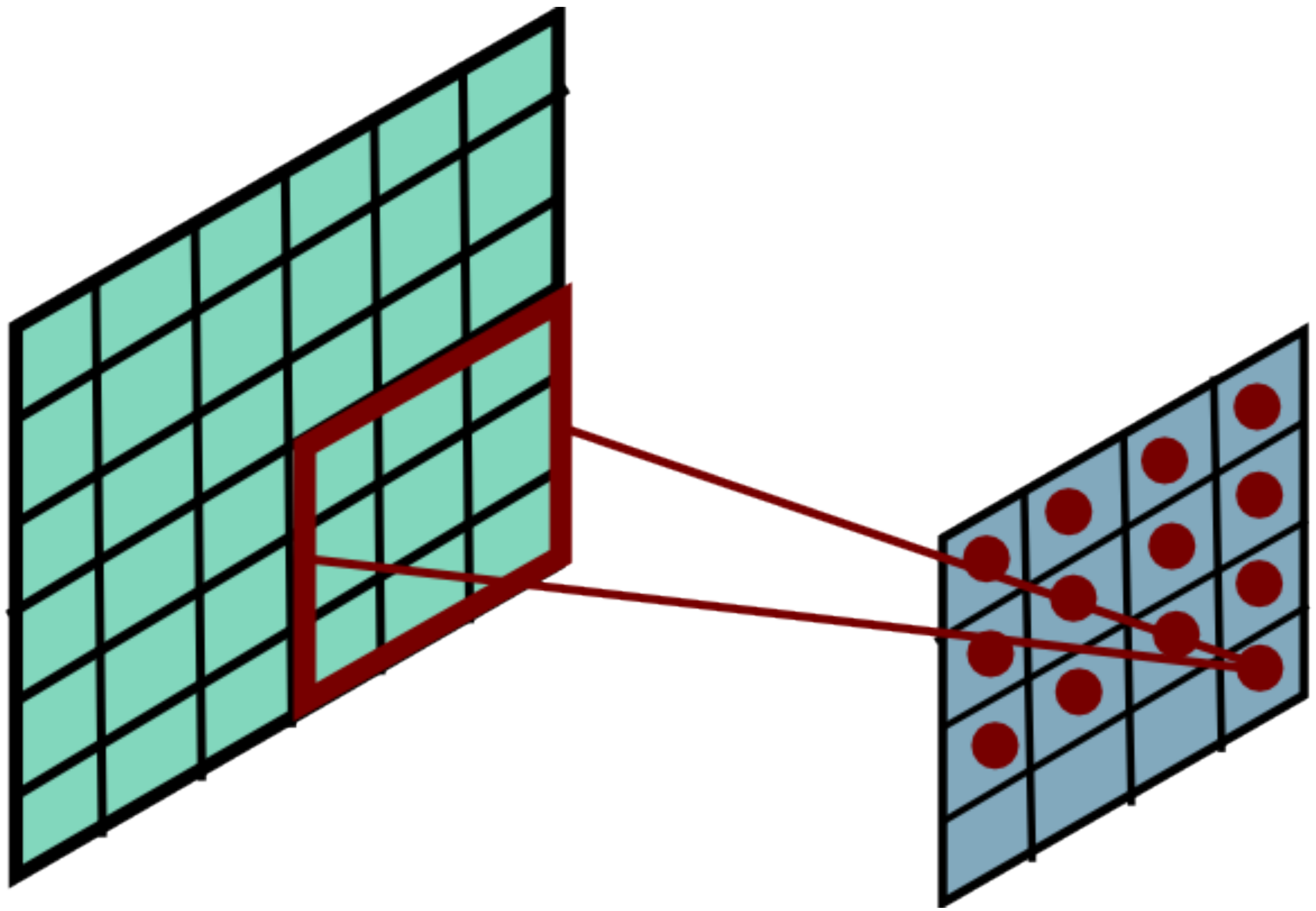
* slide from Marc'Aurelio Renzato

Convolutional Layer



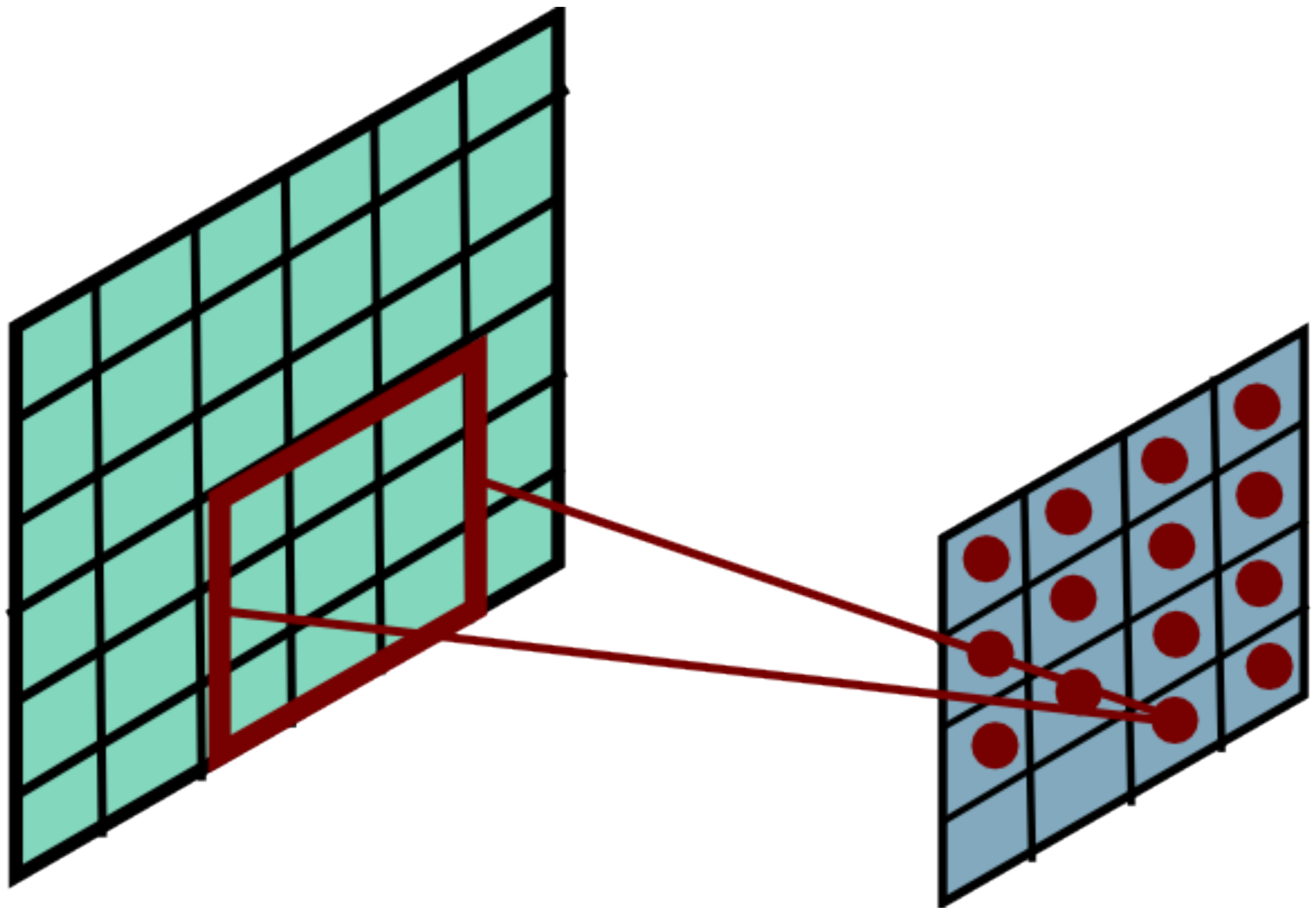
* slide from Marc'Aurelio Renzato

Convolutional Layer



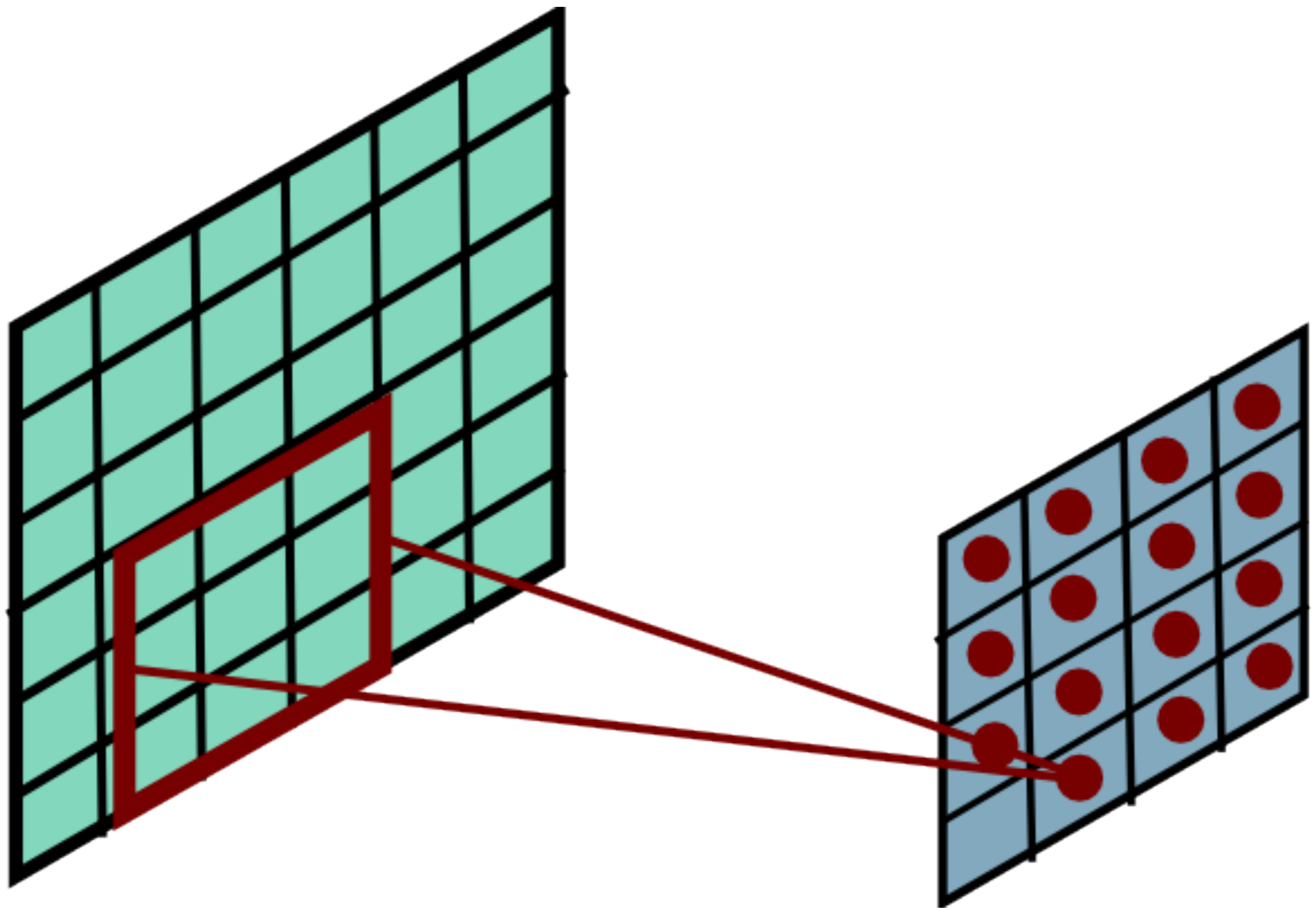
* slide from Marc'Aurelio Renzato

Convolutional Layer



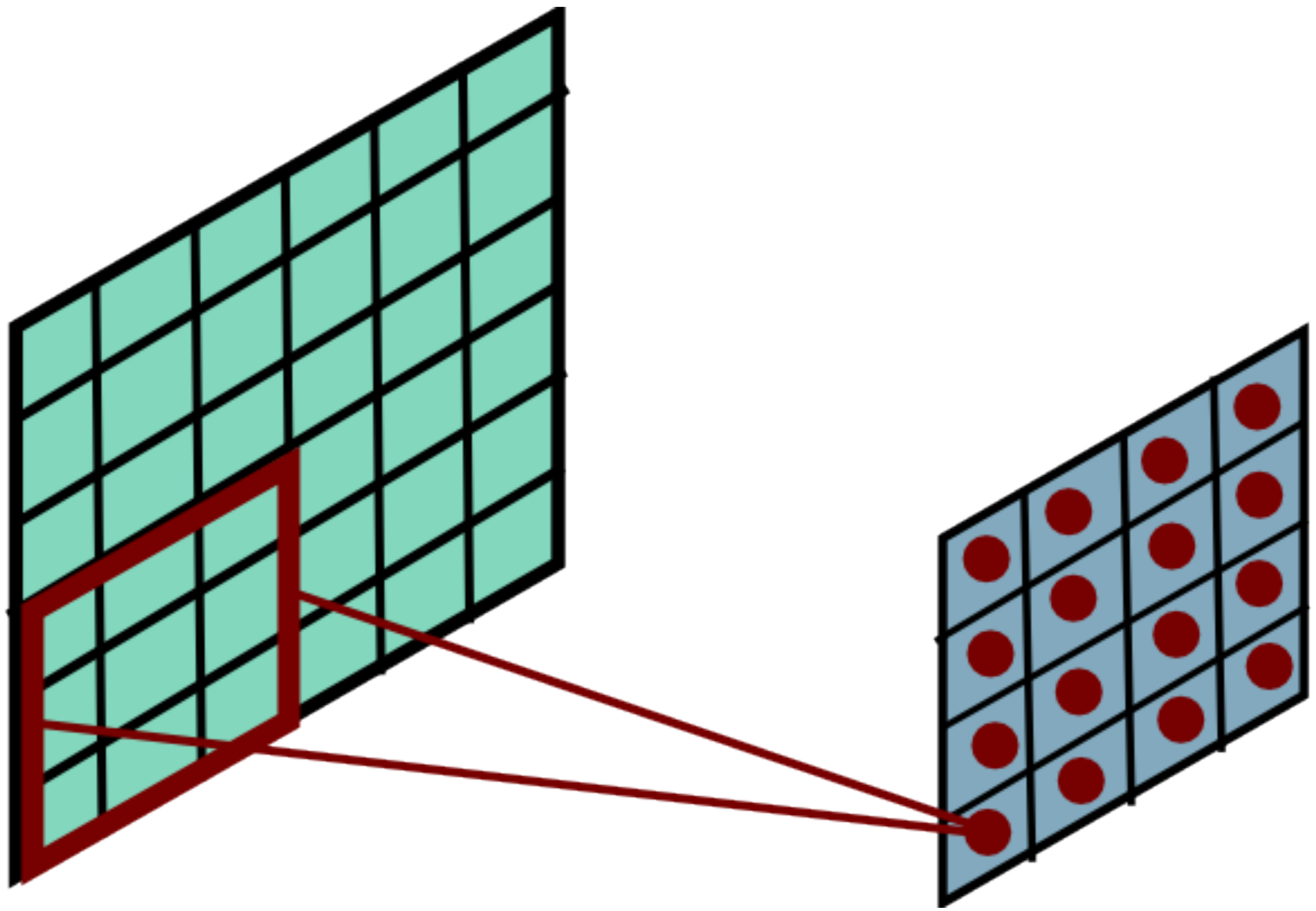
* slide from Marc'Aurelio Renzato

Convolutional Layer



* slide from Marc'Aurelio Renzato

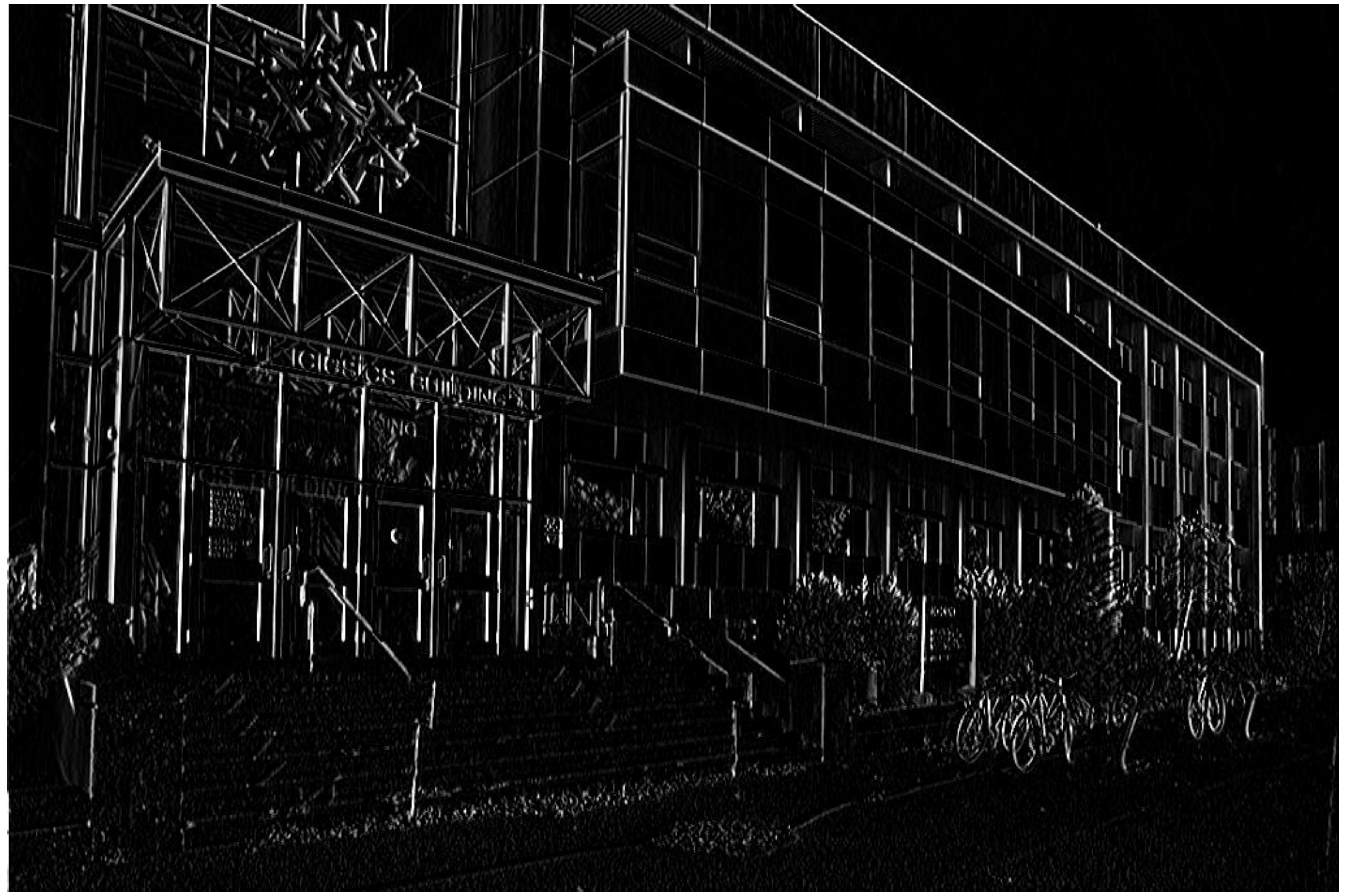
Convolutional Layer



* slide from Marc'Aurelio Renzato

Convolution Layer

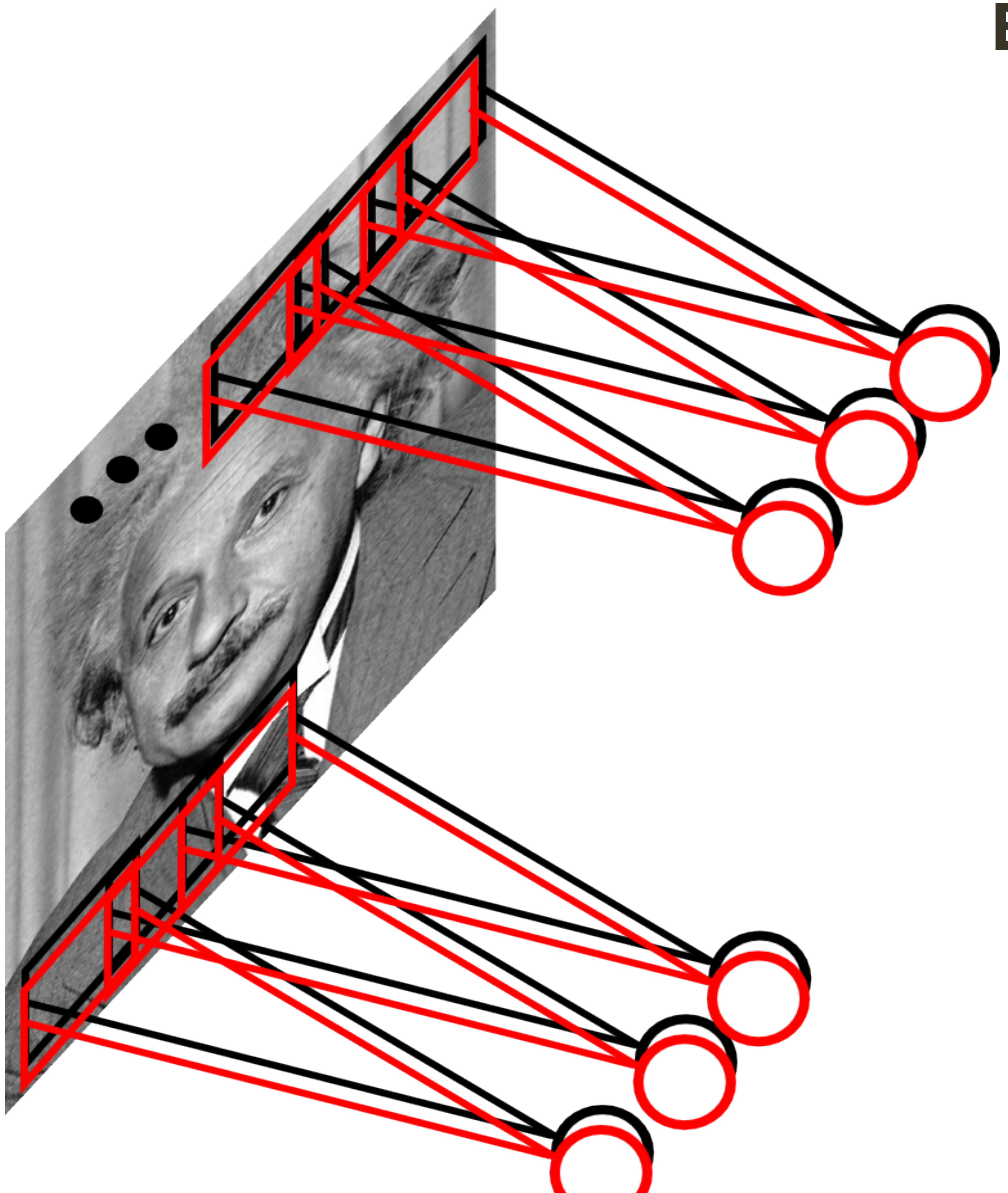
$$\star \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \longrightarrow$$



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

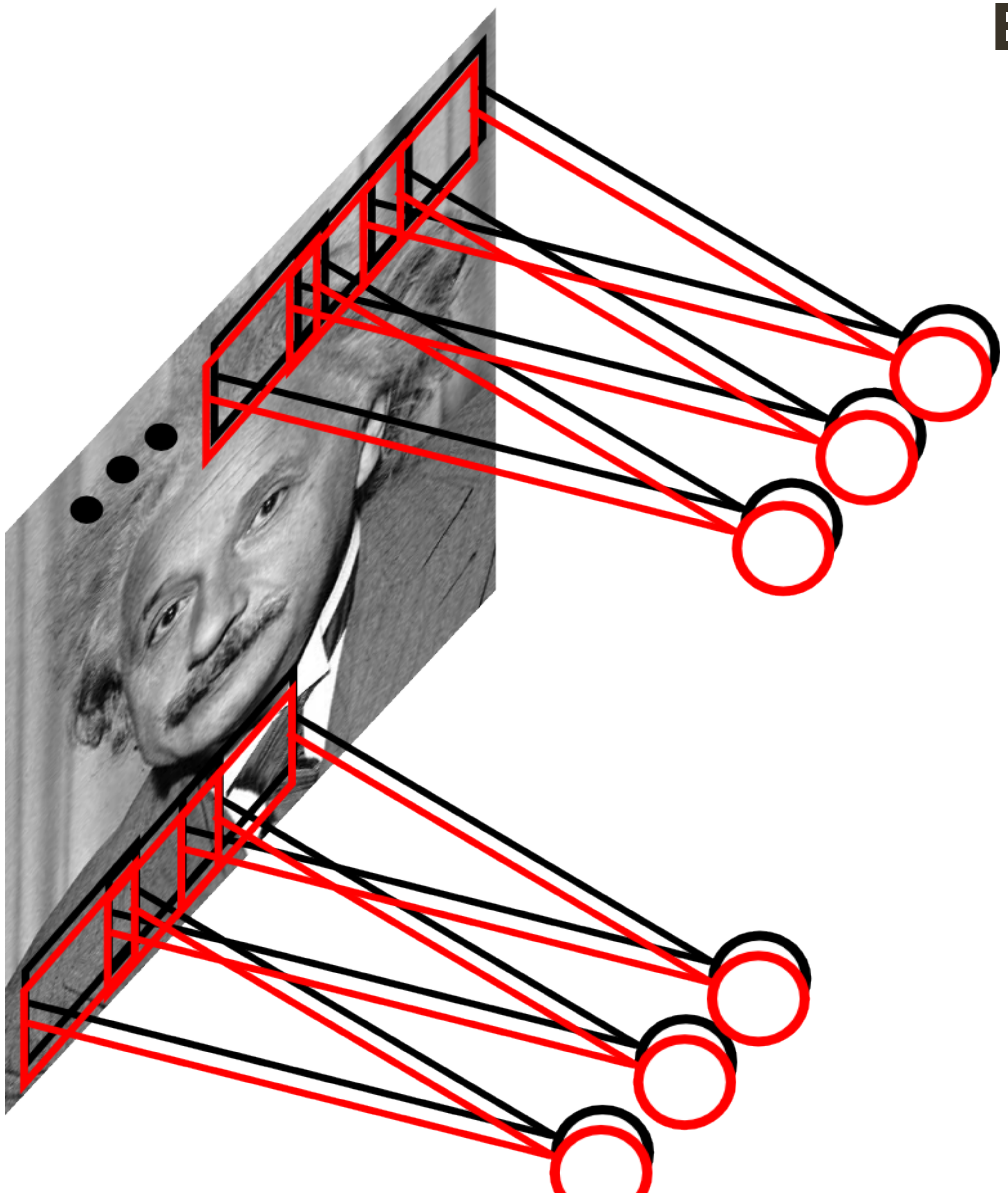
Filter size: 10 x 10

# 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

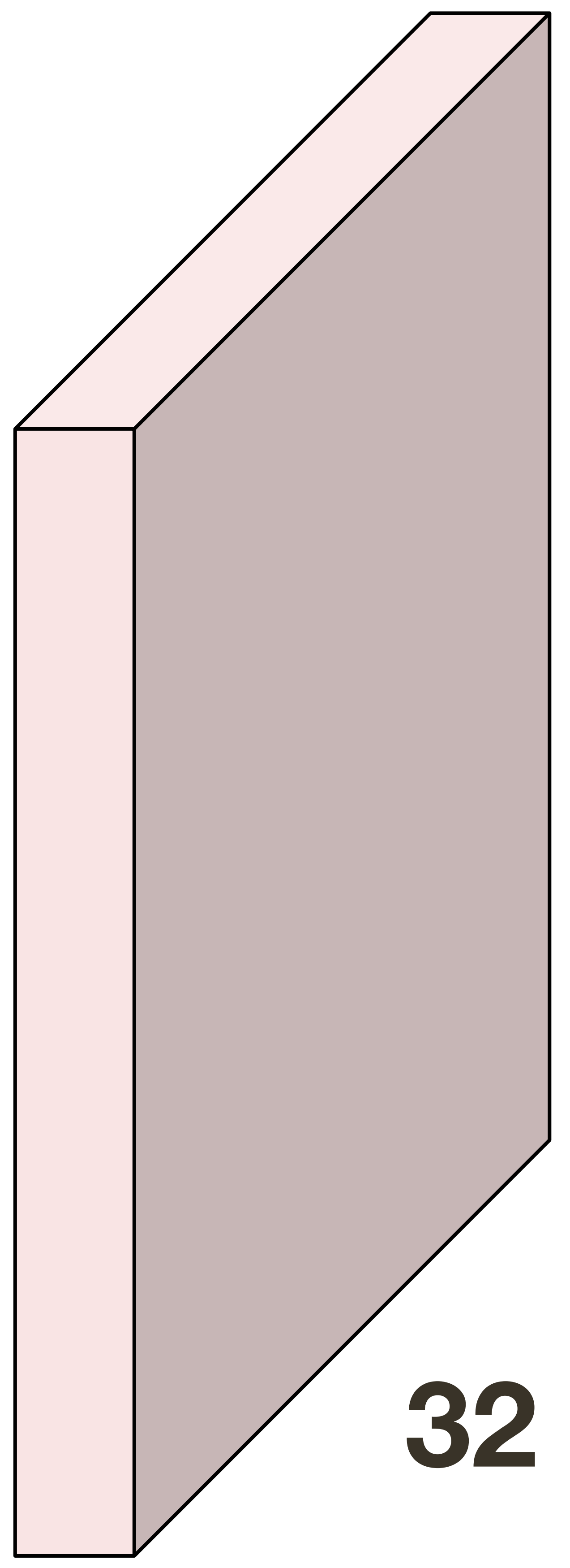
= 2000 parameters

Learn **multiple filters**

* slide from Marc'Aurelio Renzato

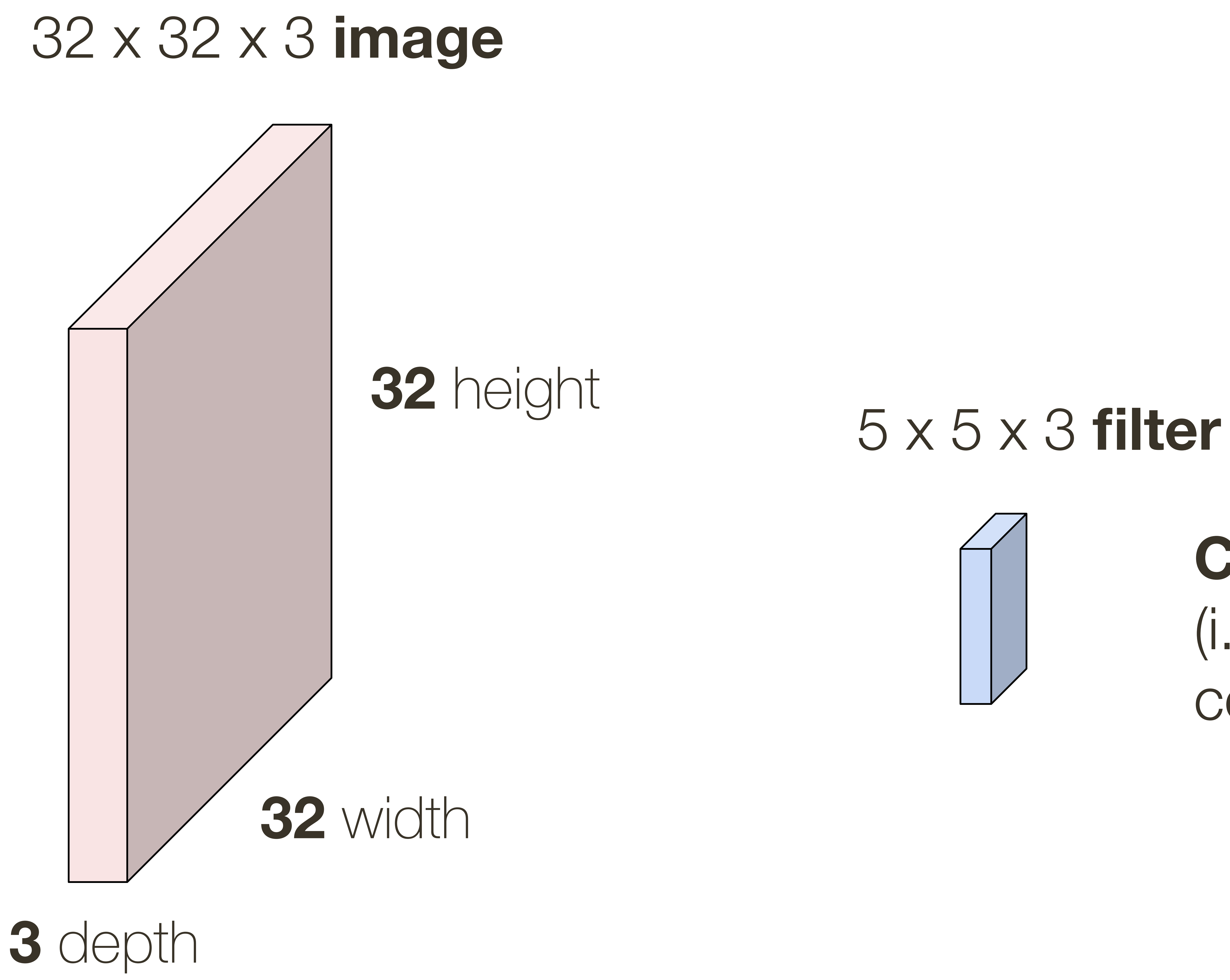
Convolutional Layer

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



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

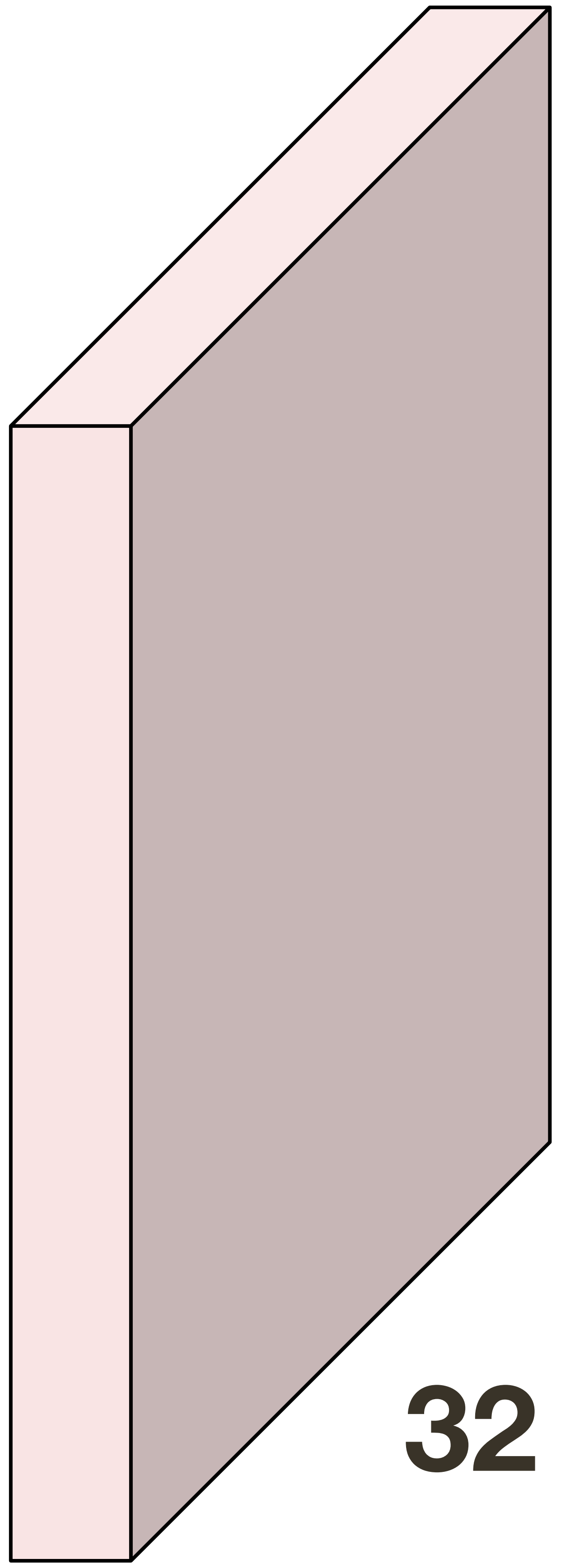
Convolutional Layer



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

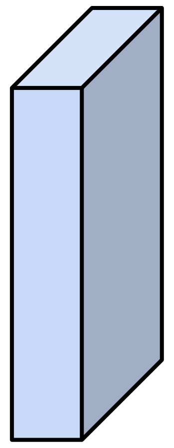
Convolutional Layer

32 x 32 x 3 **image**



Filters always extend the full depth of the input volume

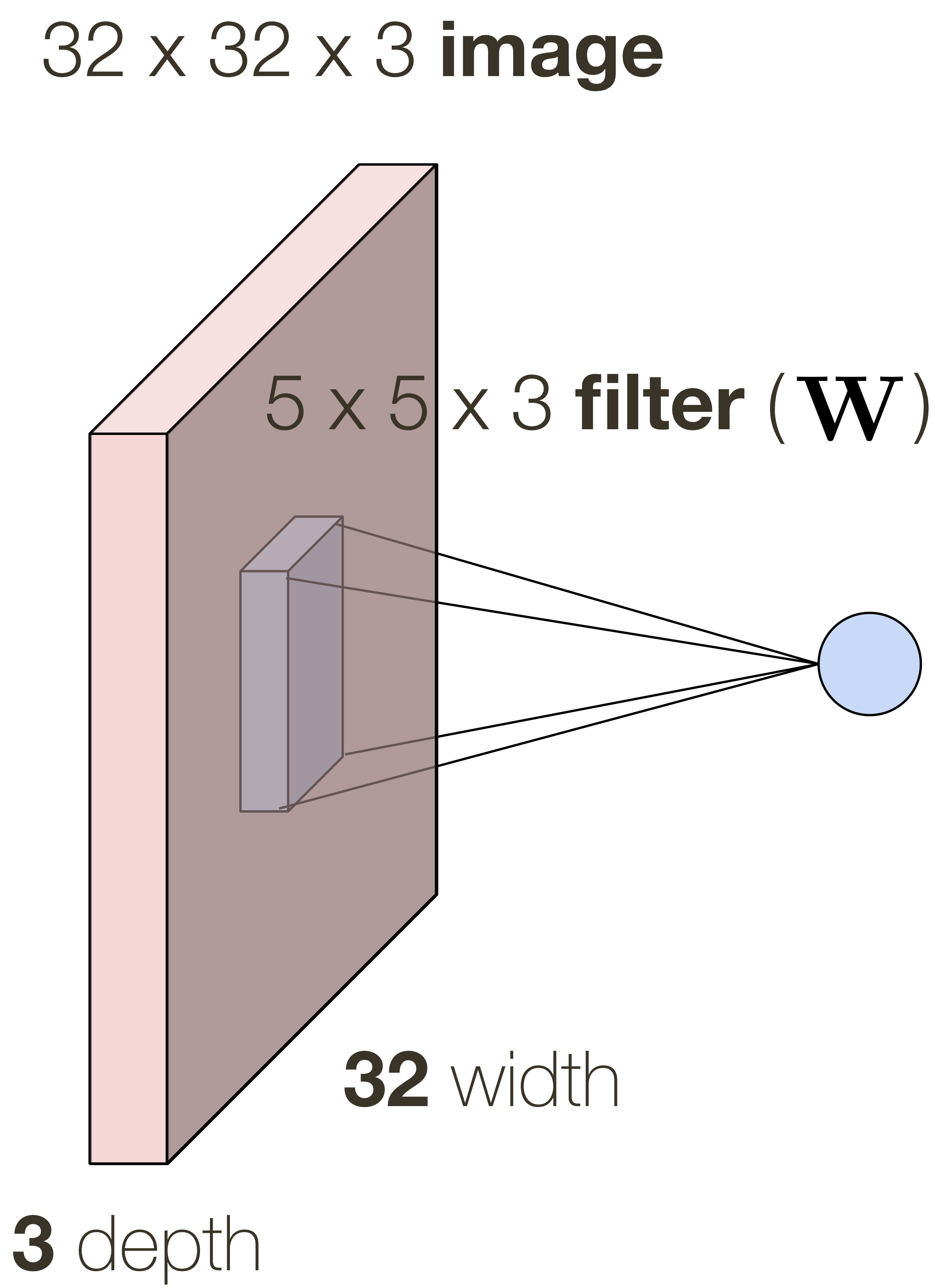
5 x 5 x 3 **filter**



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

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

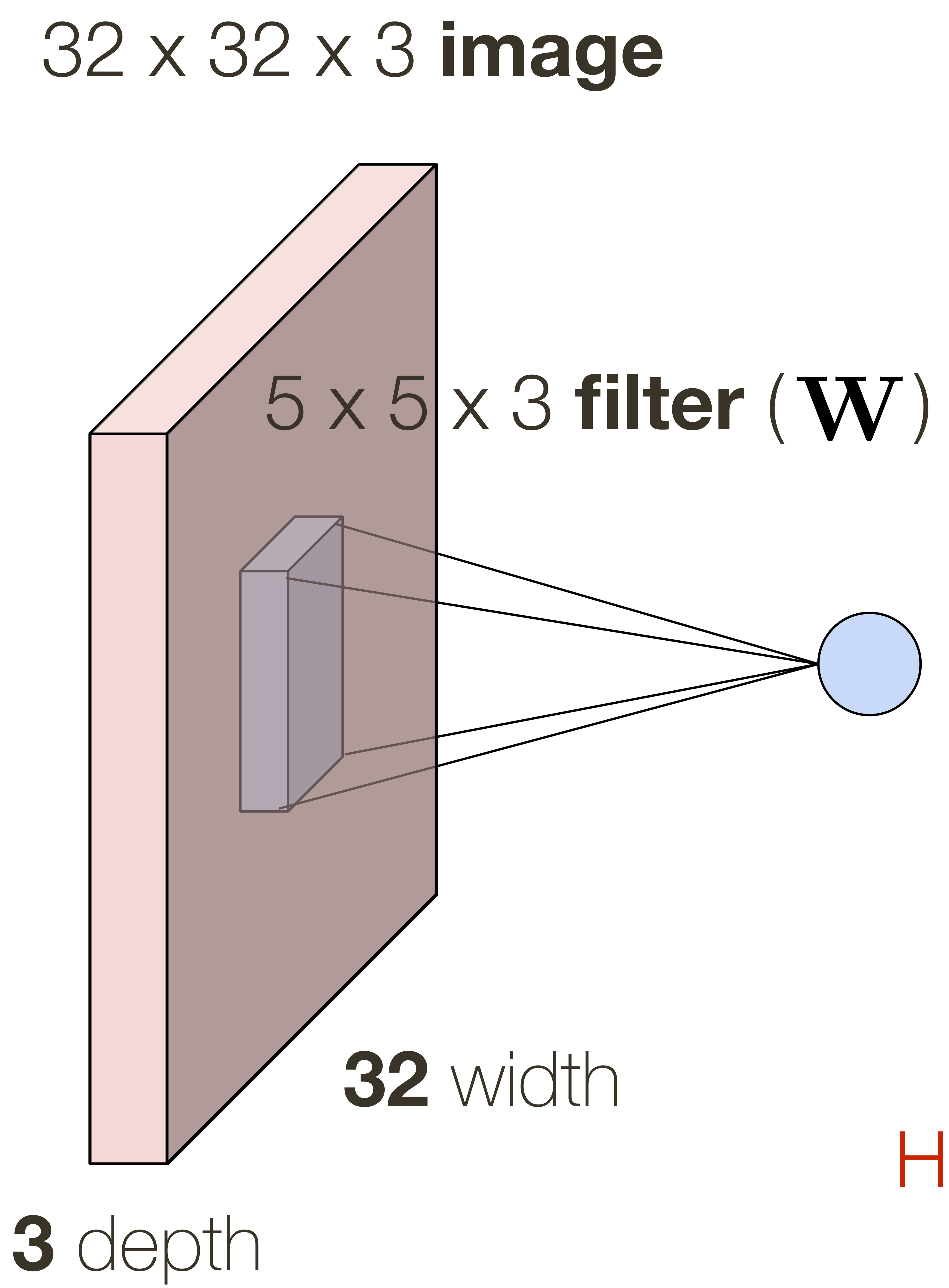
Convolutional Layer



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



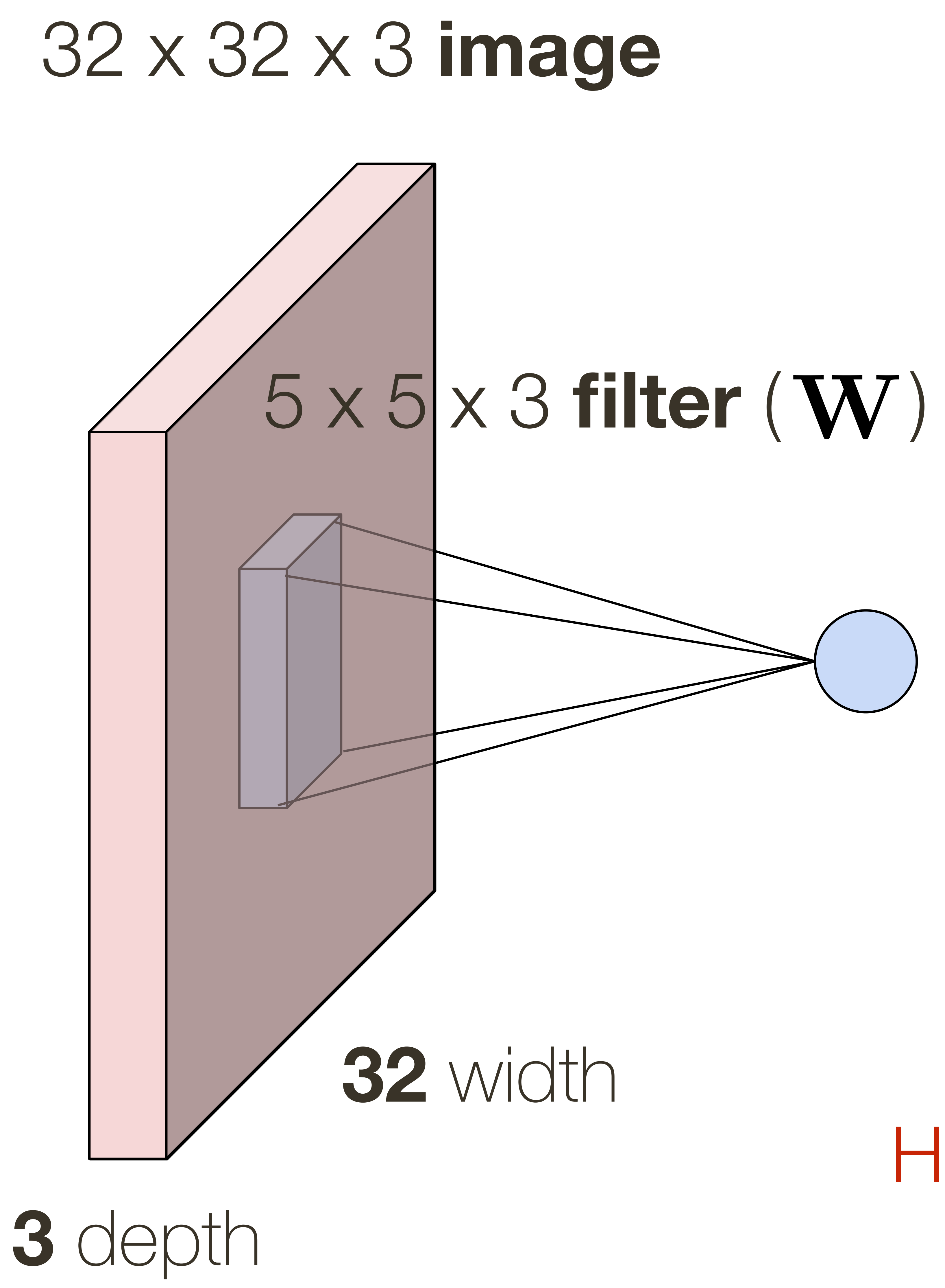
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

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

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

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

Convolutional Layer



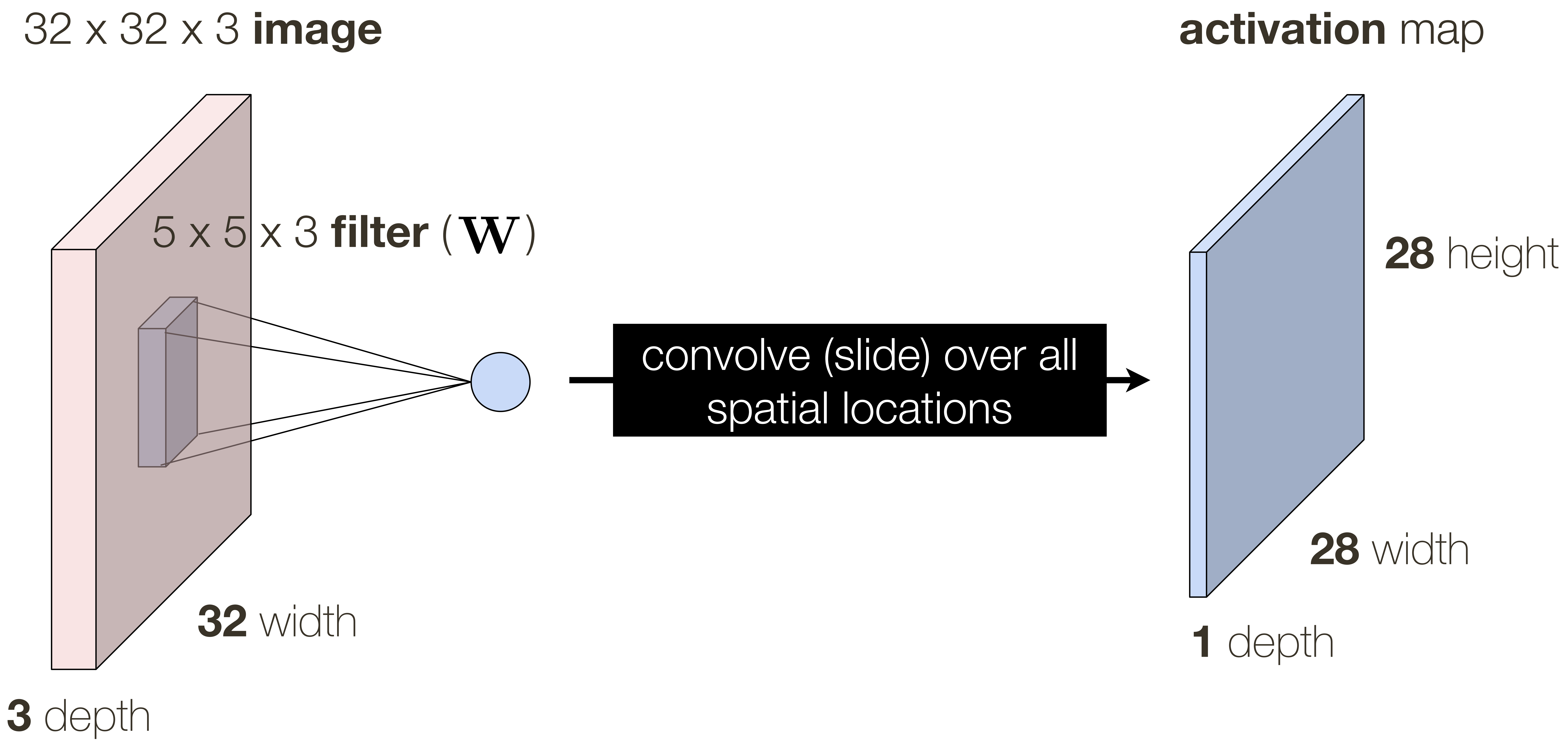
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

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

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

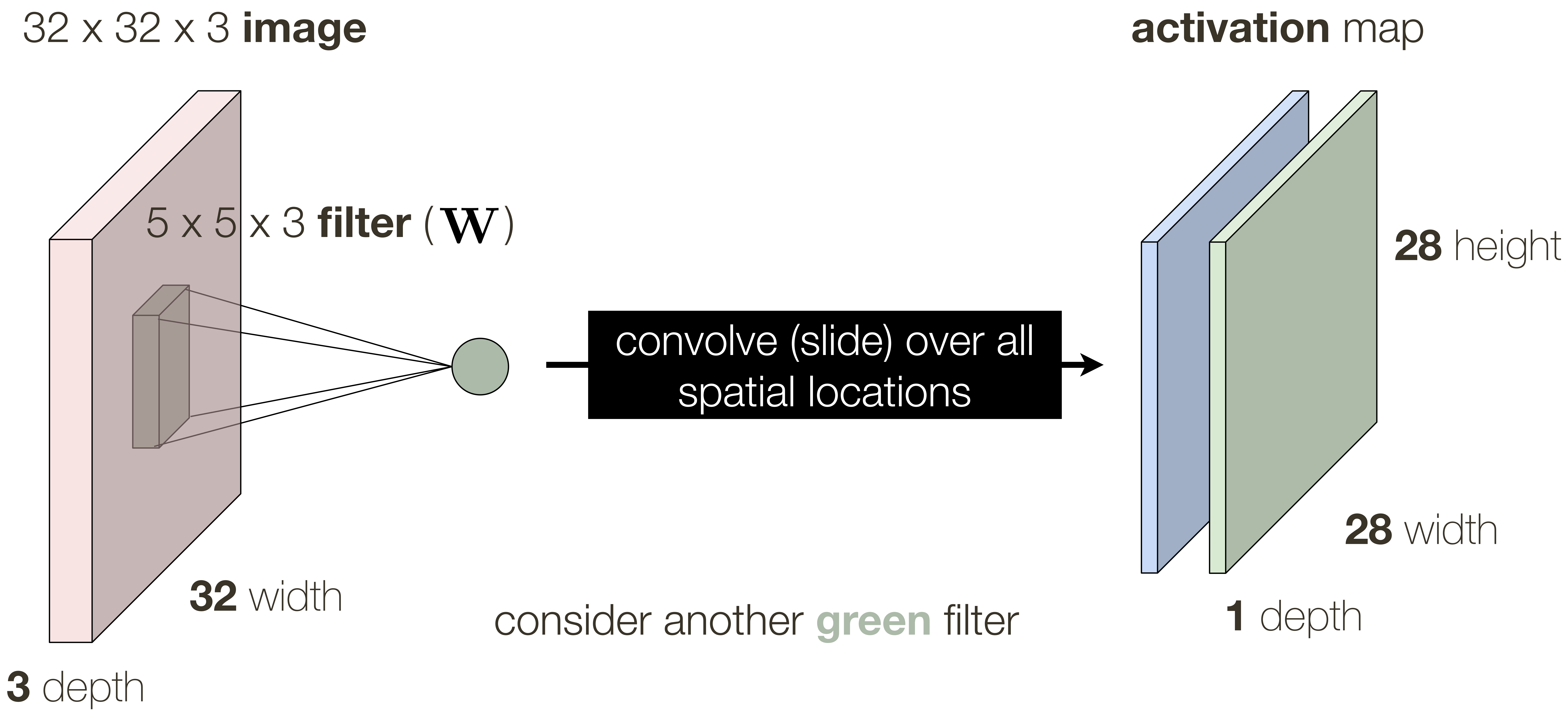
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer



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

Convolutional Layer

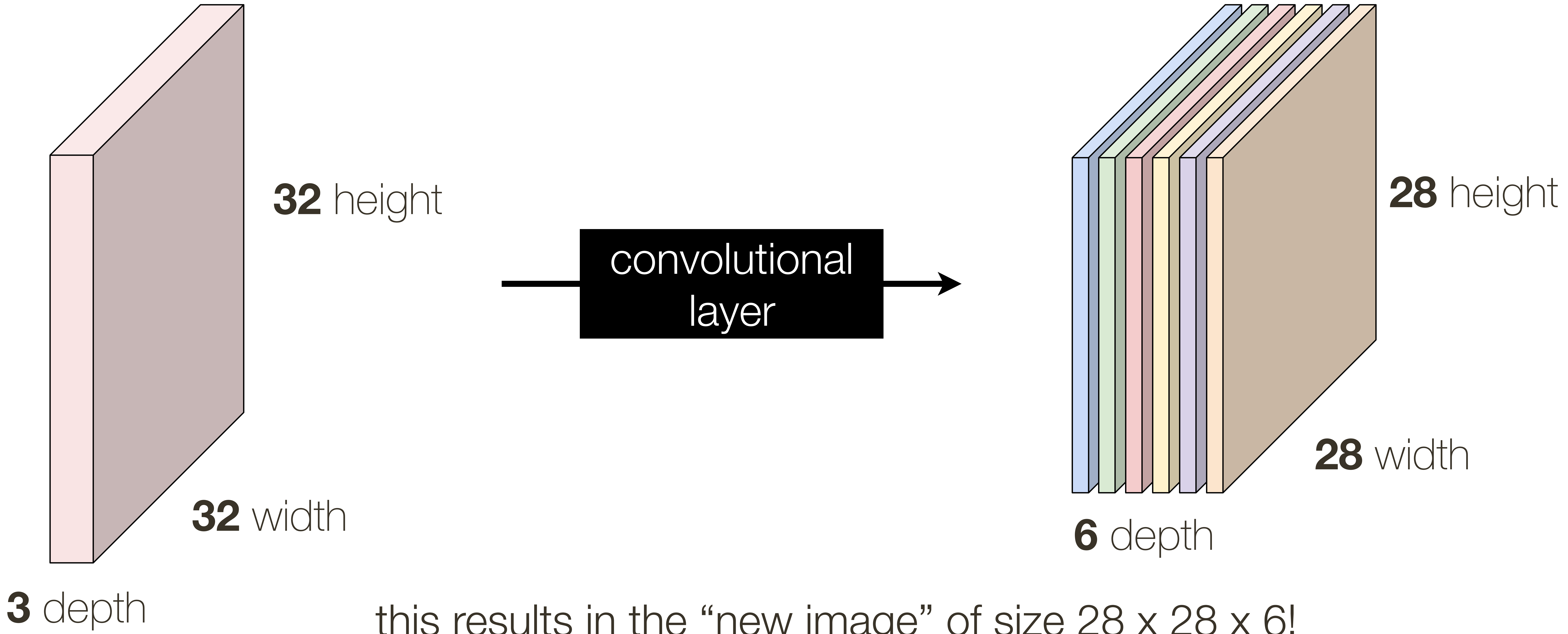


consider another **green** filter

* 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: **activation** map



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

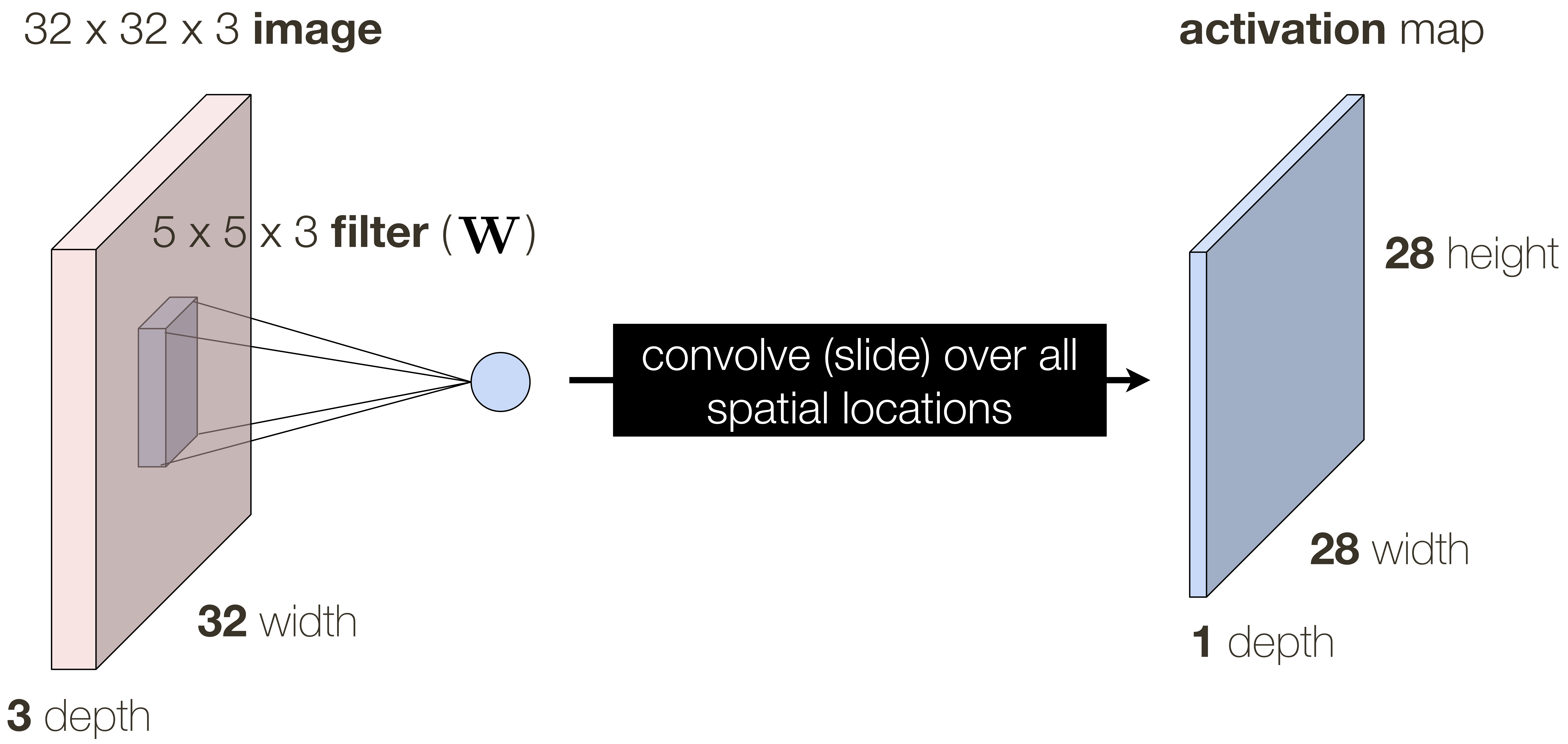
— also affected by zero-padding

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**

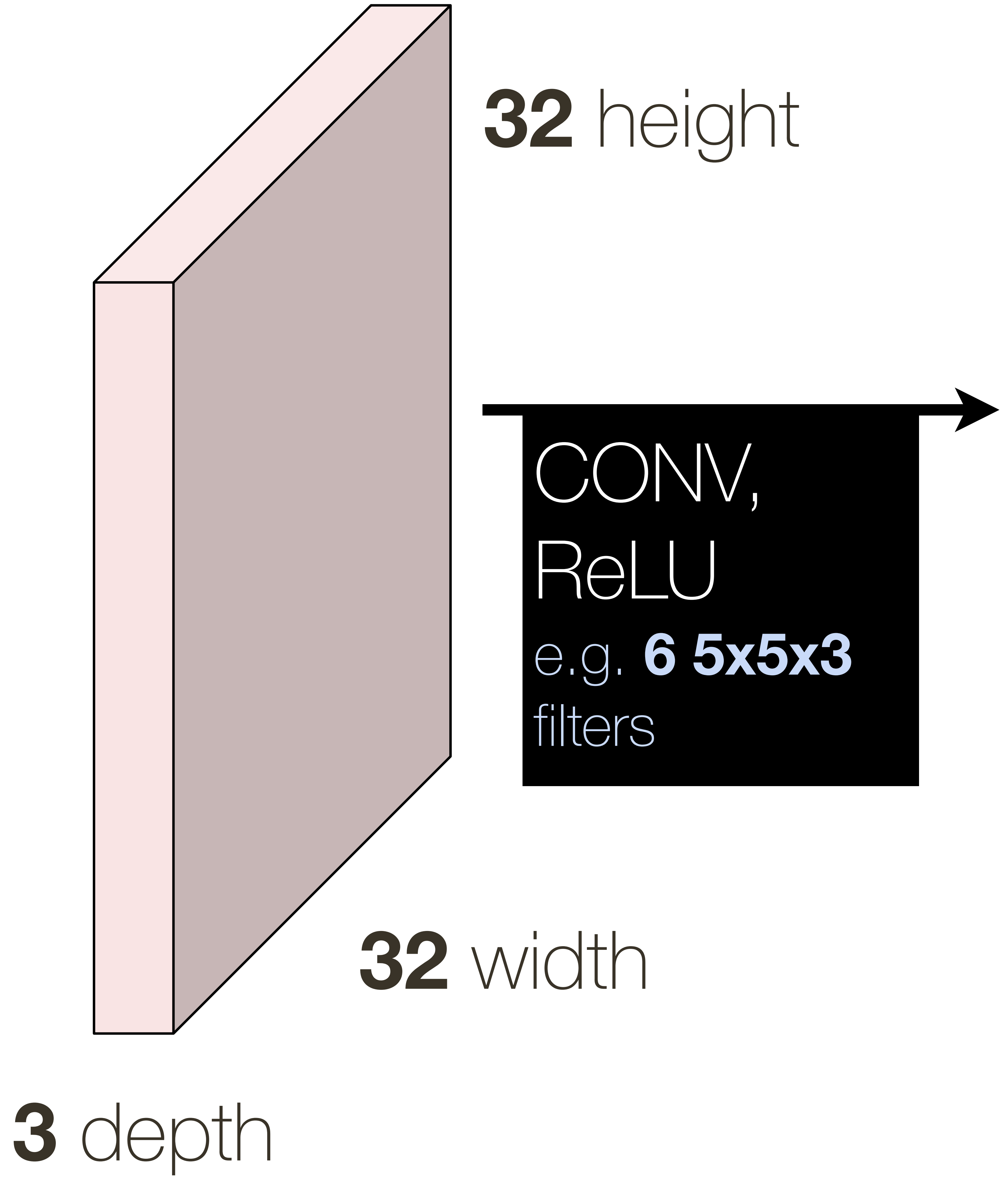
Stride: Controls spatial density. How far apart are depth columns?

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



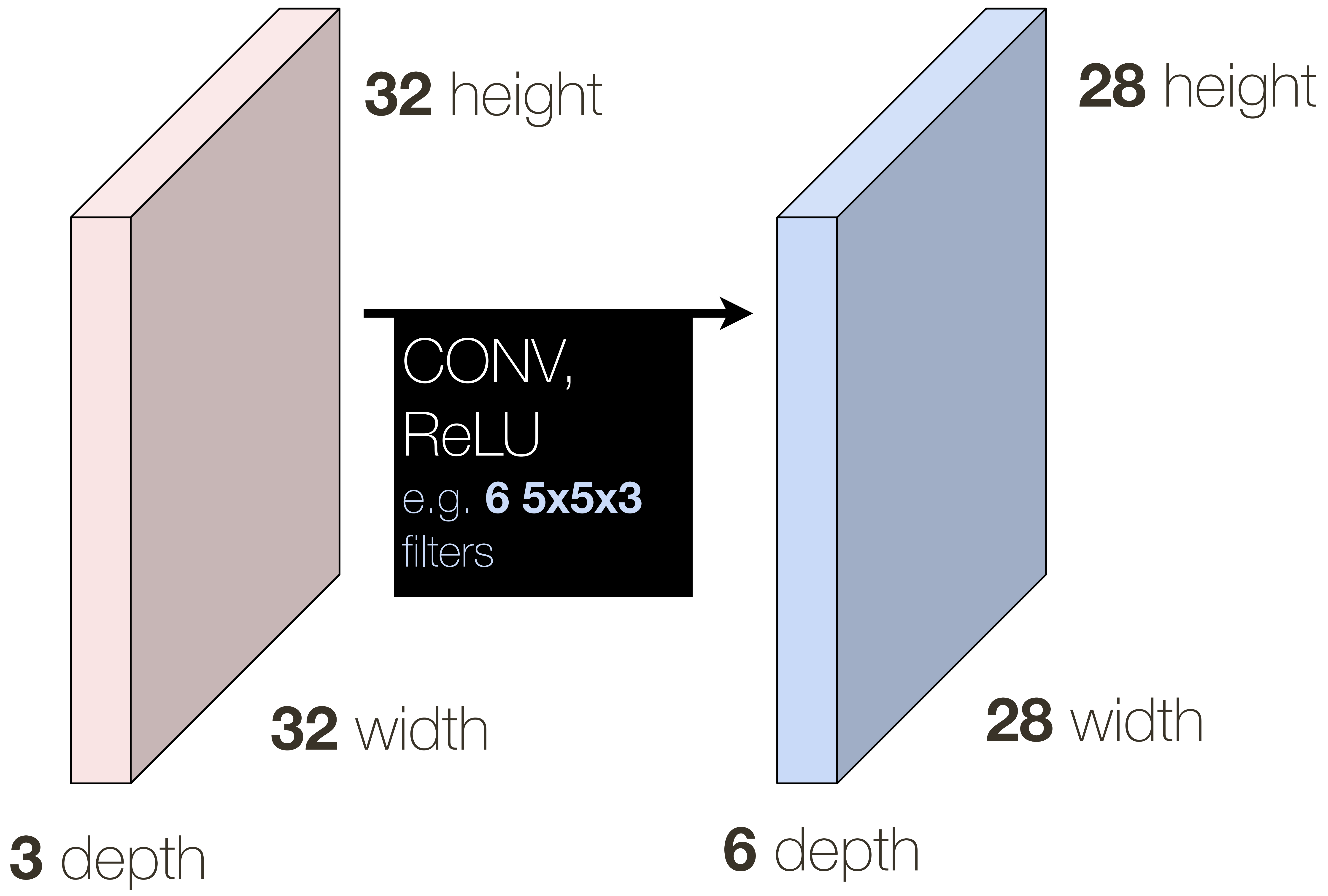
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



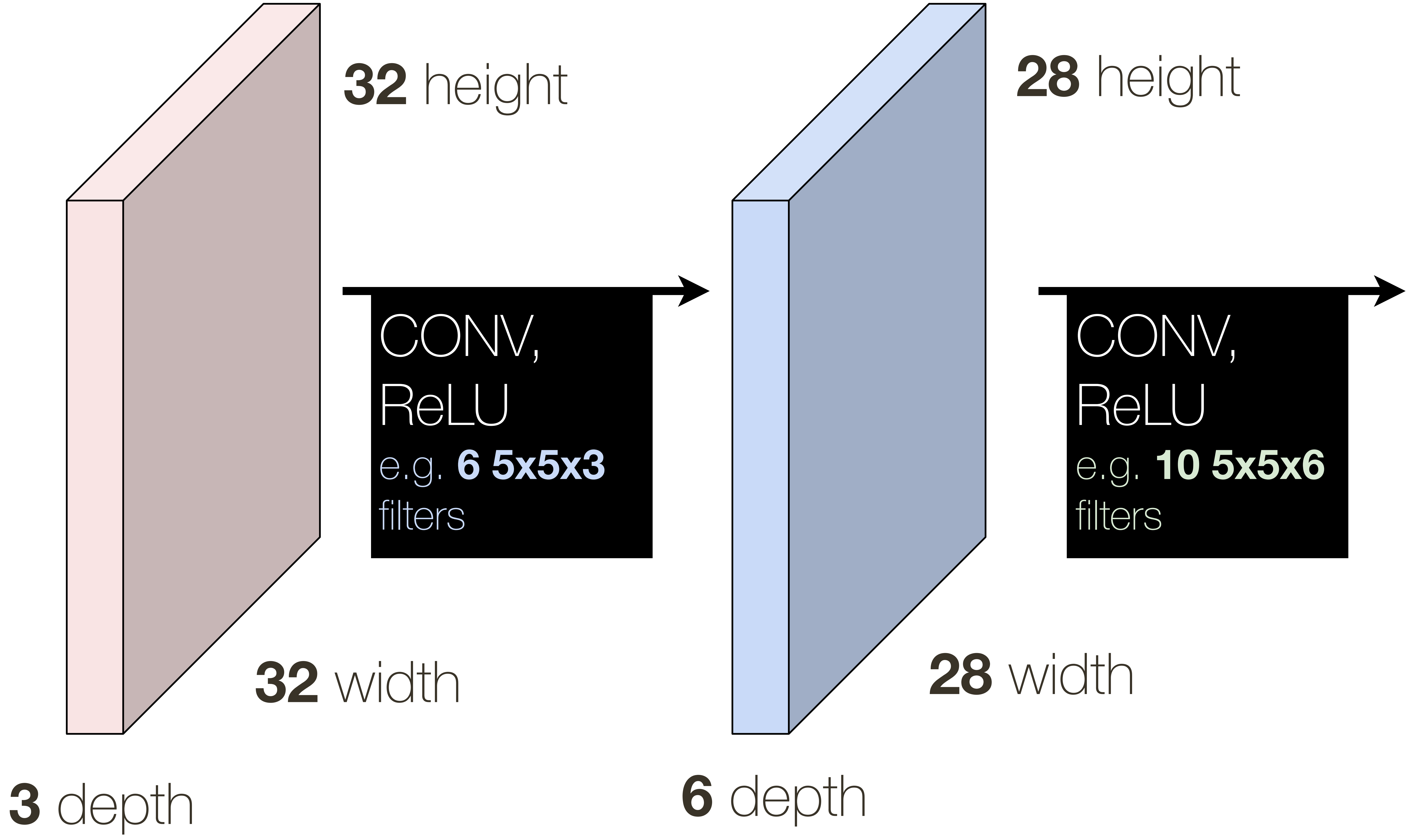
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



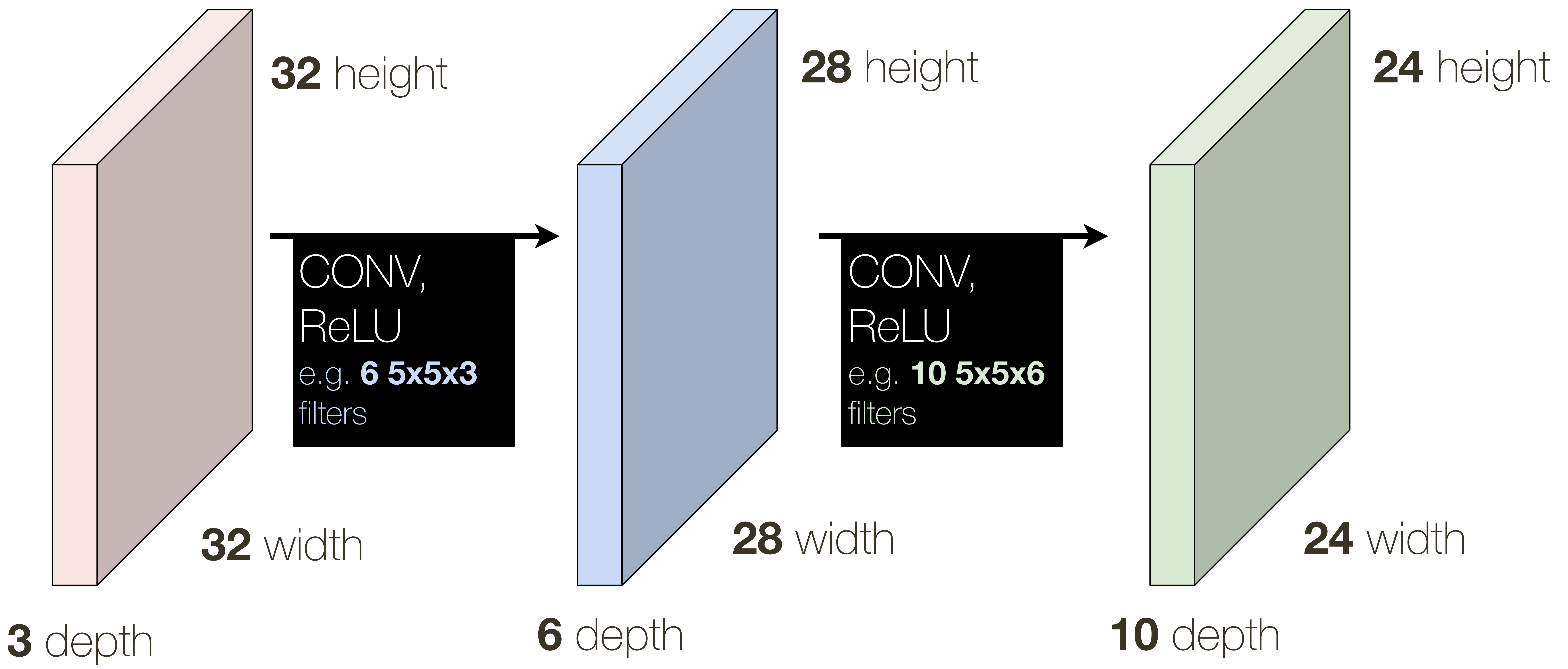
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



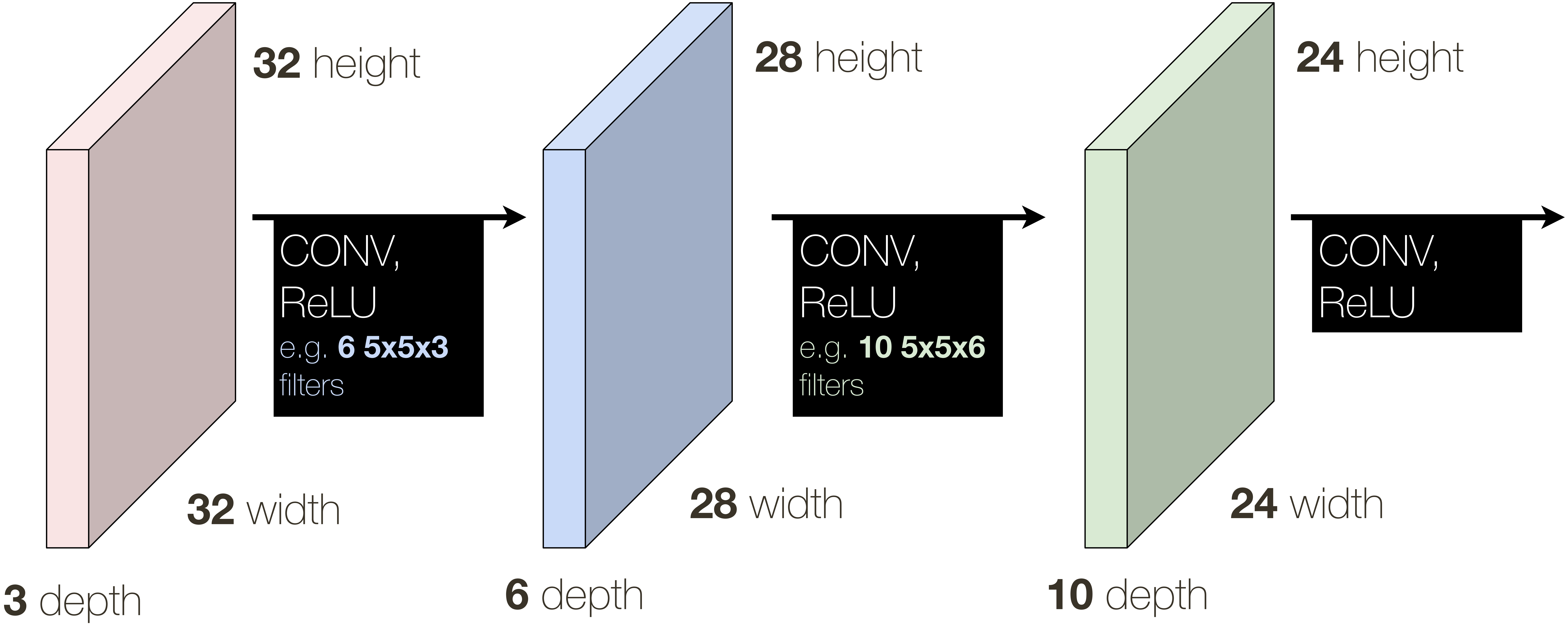
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



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

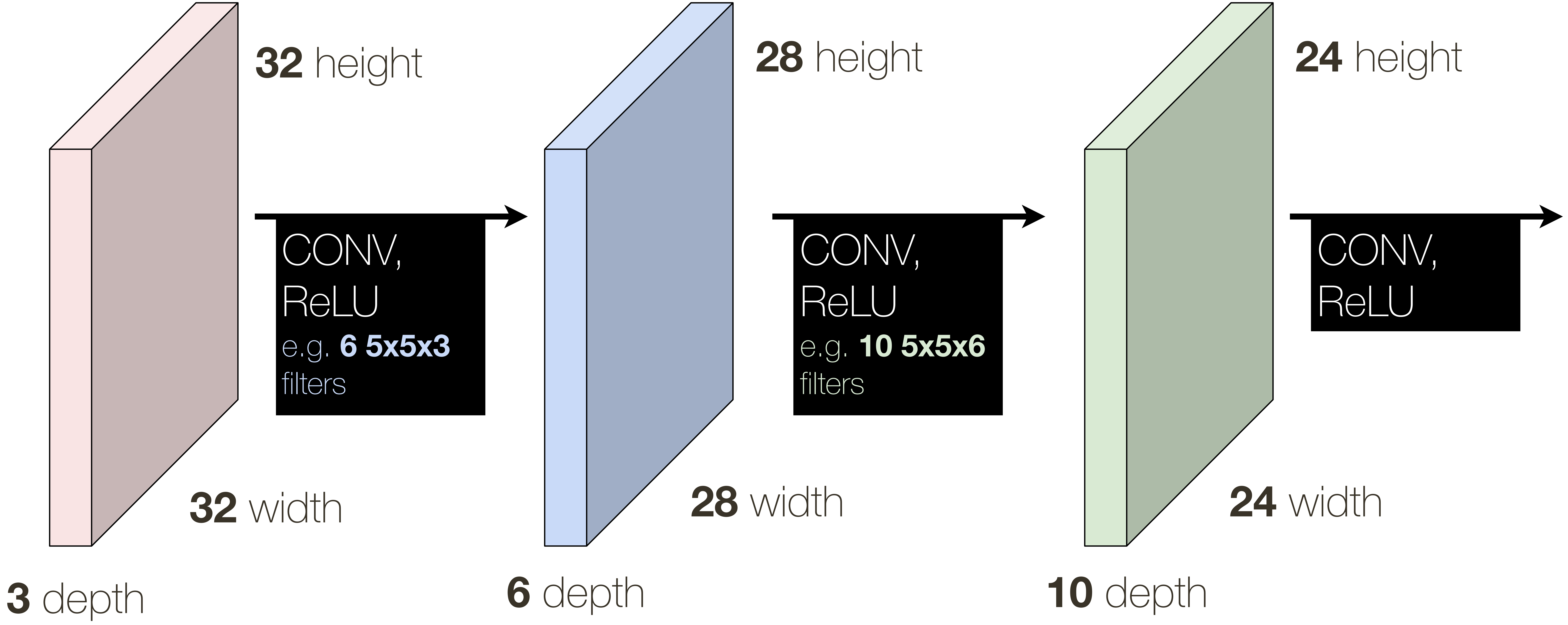
Convolutional Neural Network (ConvNet)



* 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



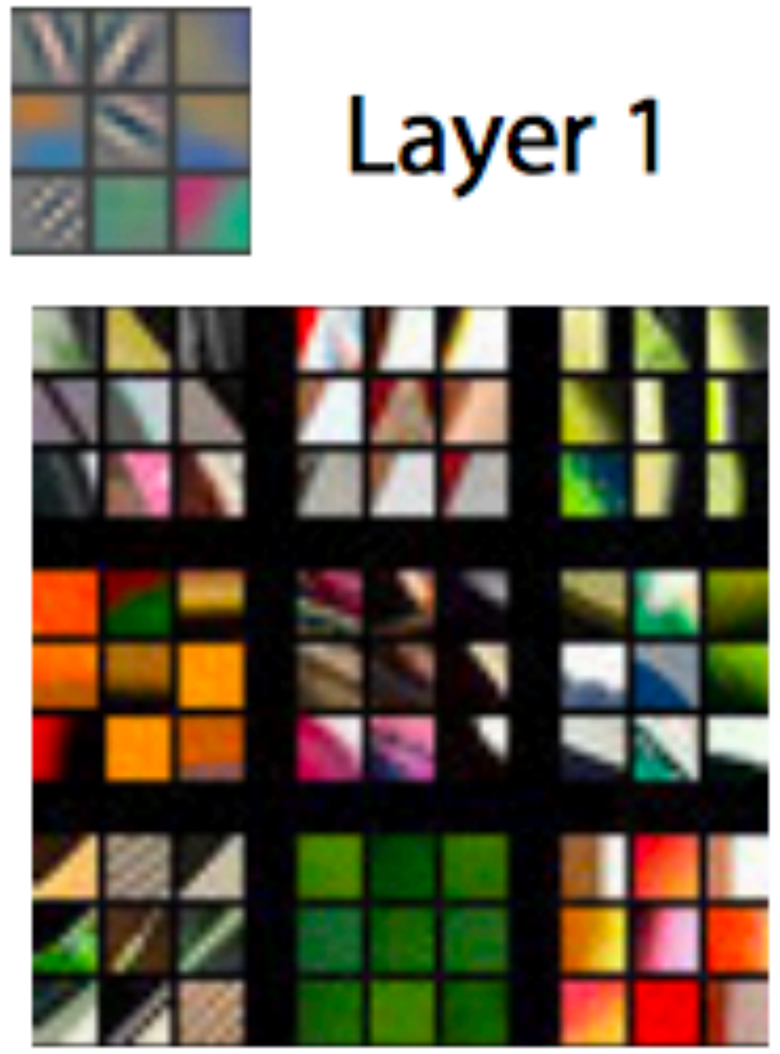
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

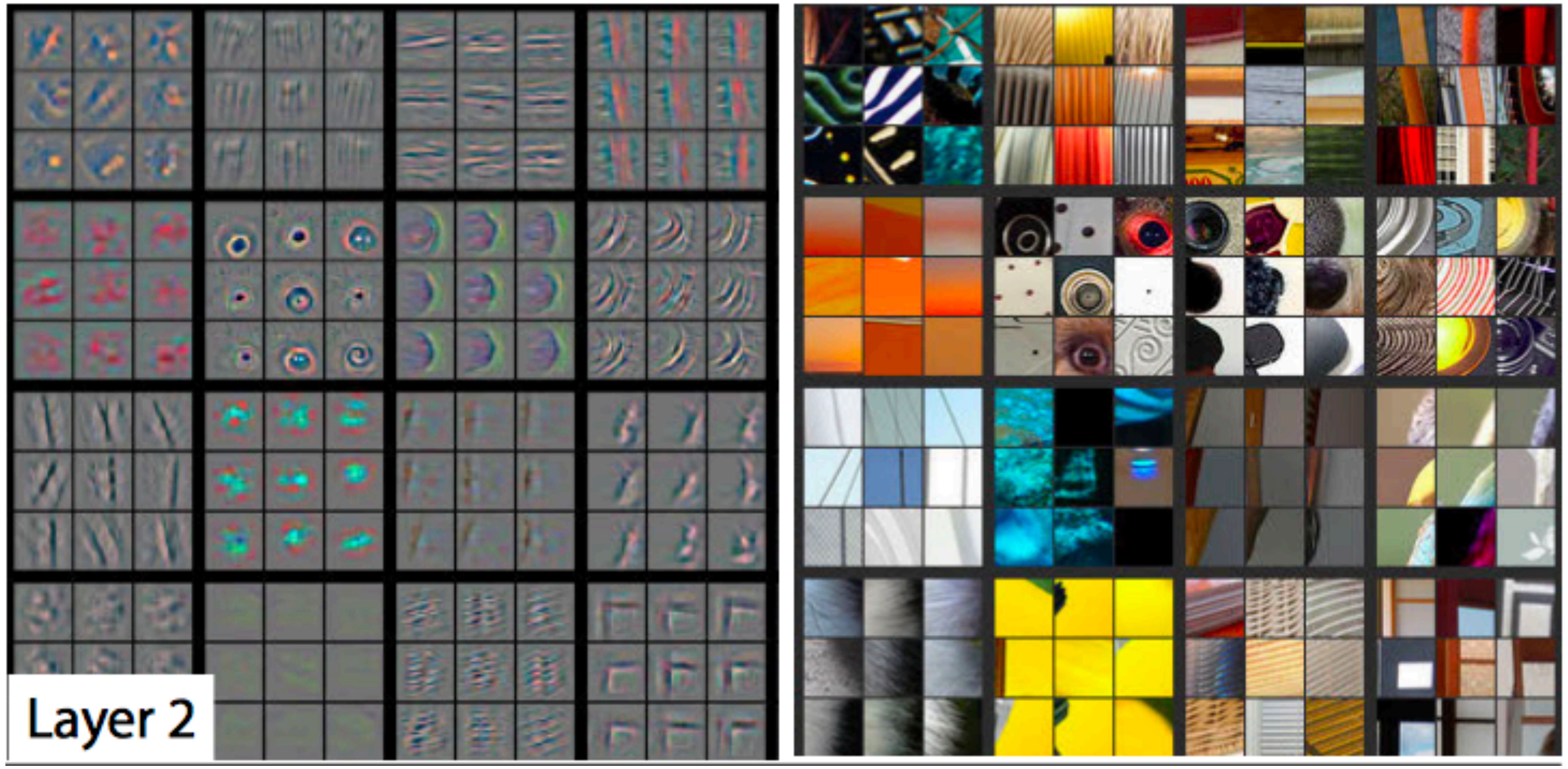
Convolutional Neural Network (ConvNet)

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

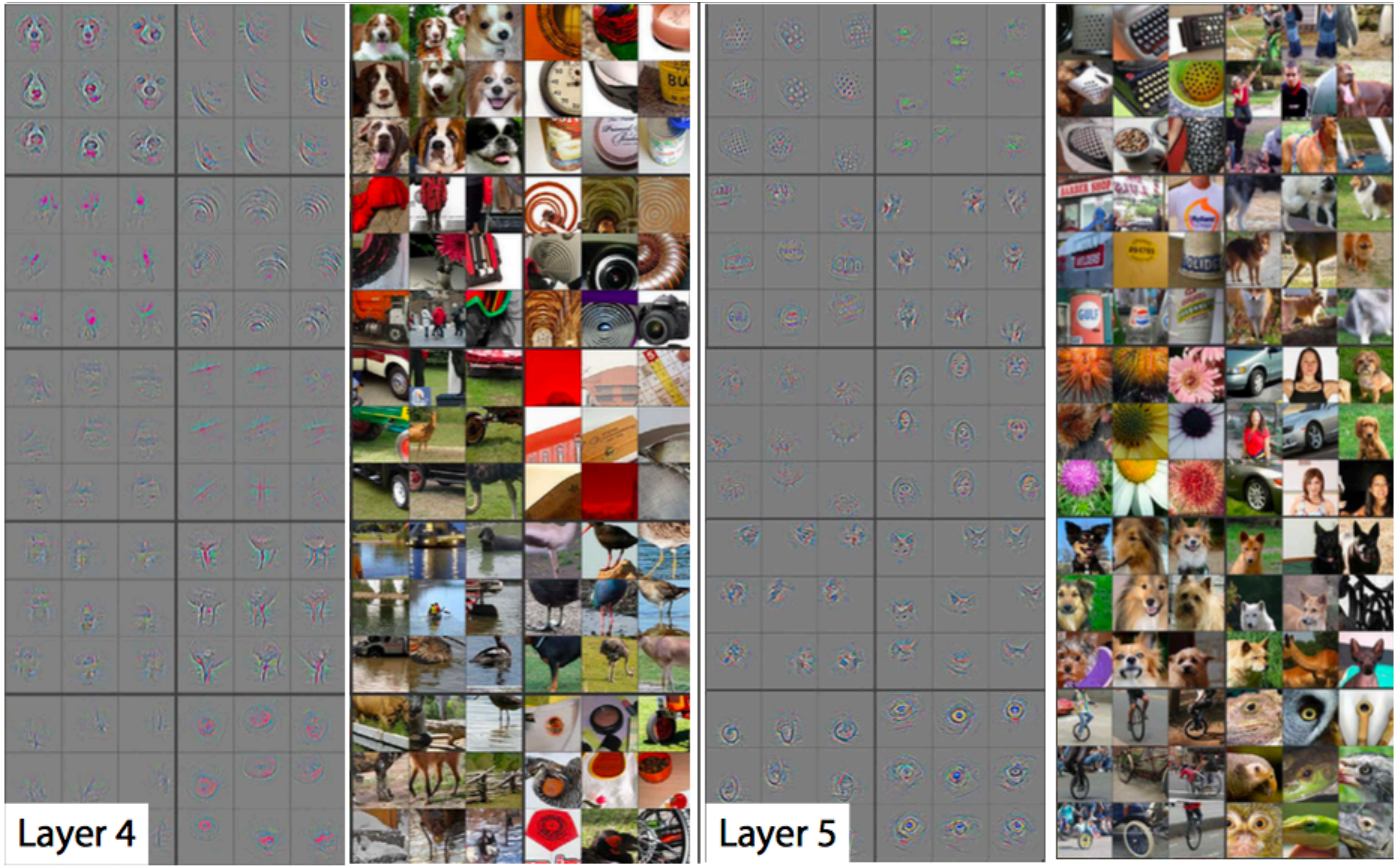
What **filters** do networks learn?





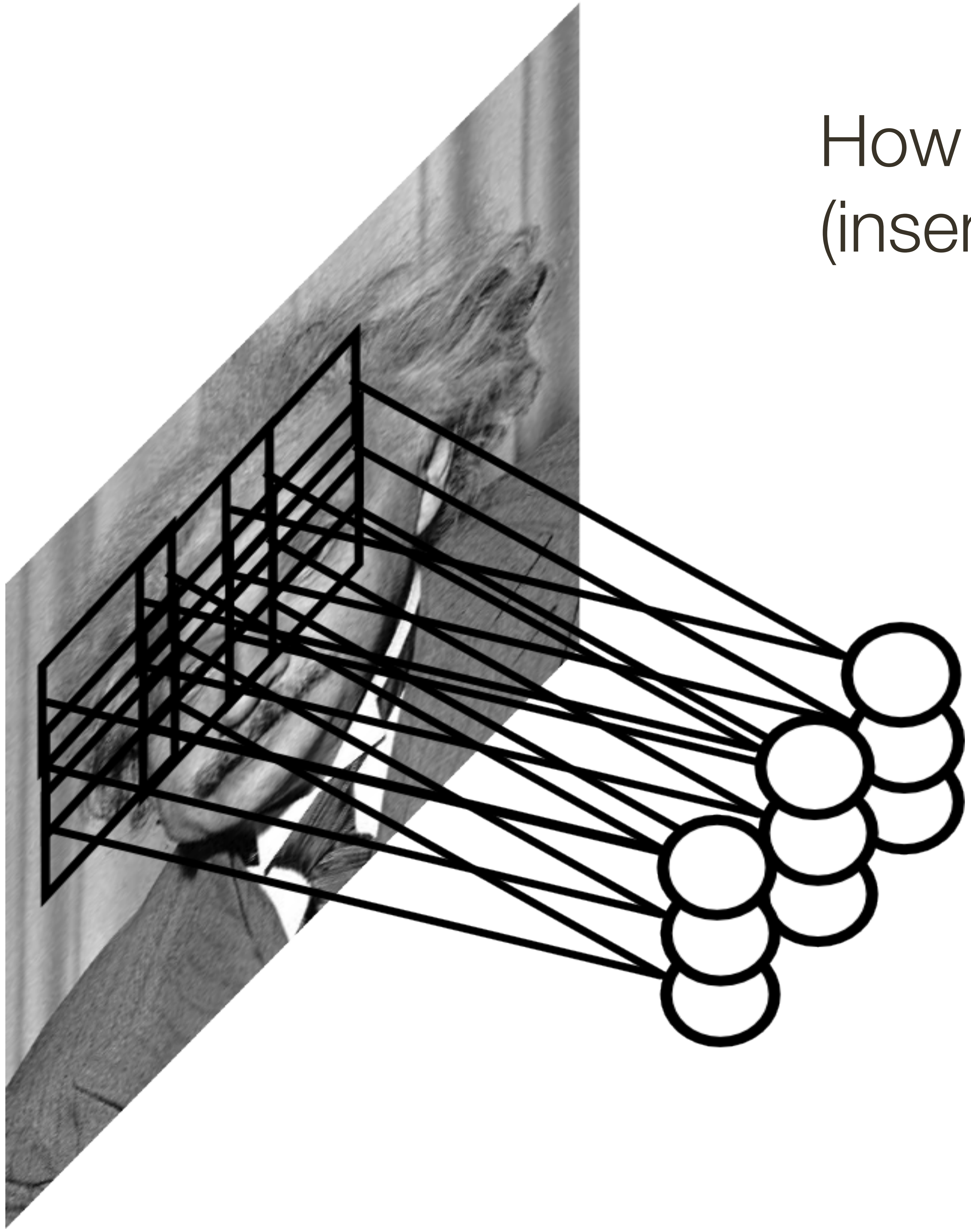
[Zeiler and Fergus, 2013]

What **filters** do networks learn?



[Zeiler and Fergus, 2013]

Pooling Layer

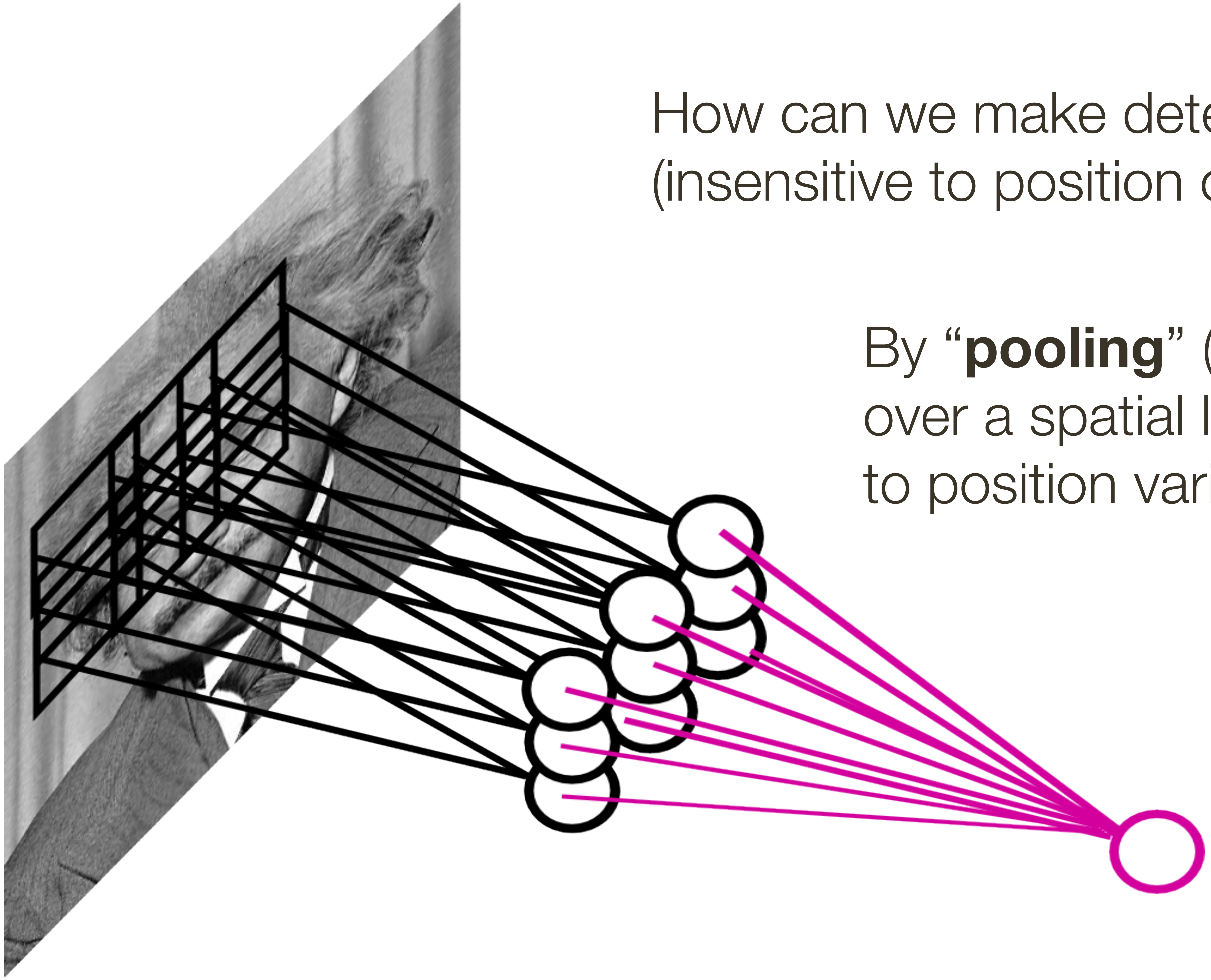


Let us assume the filter is an “eye” detector

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

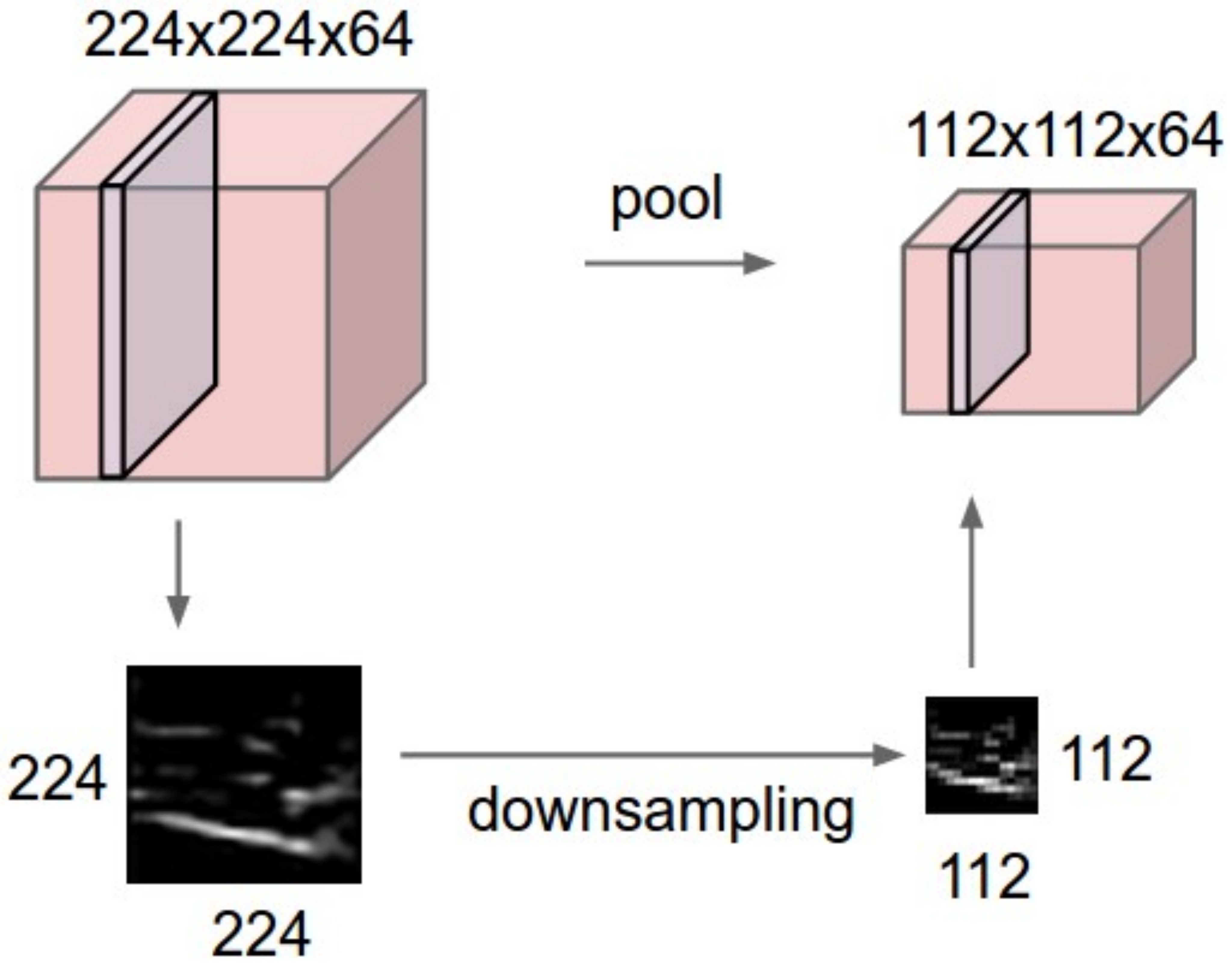
How can we make detection spatially invariant (insensitive to position of the eye in the image)

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

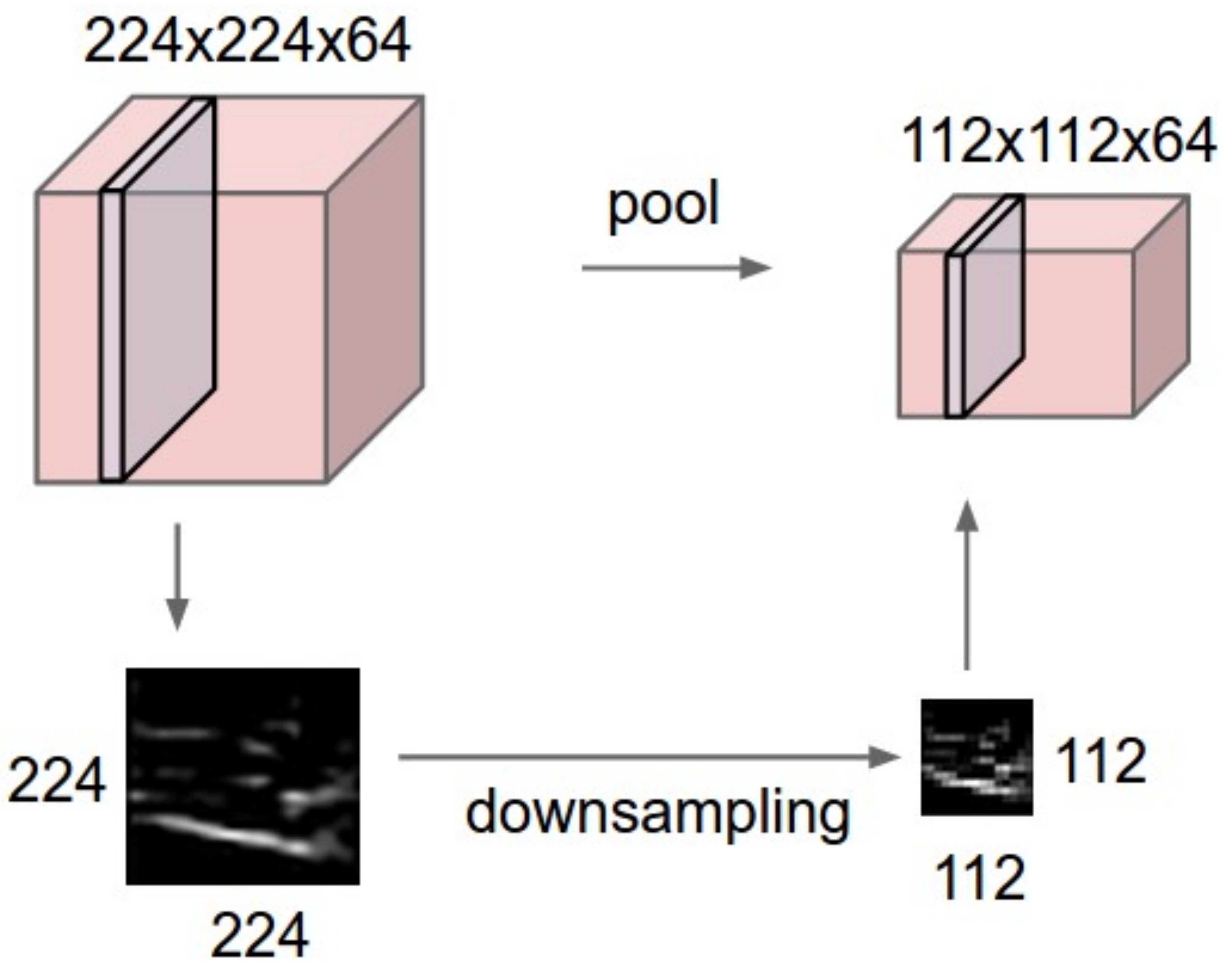
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



* 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

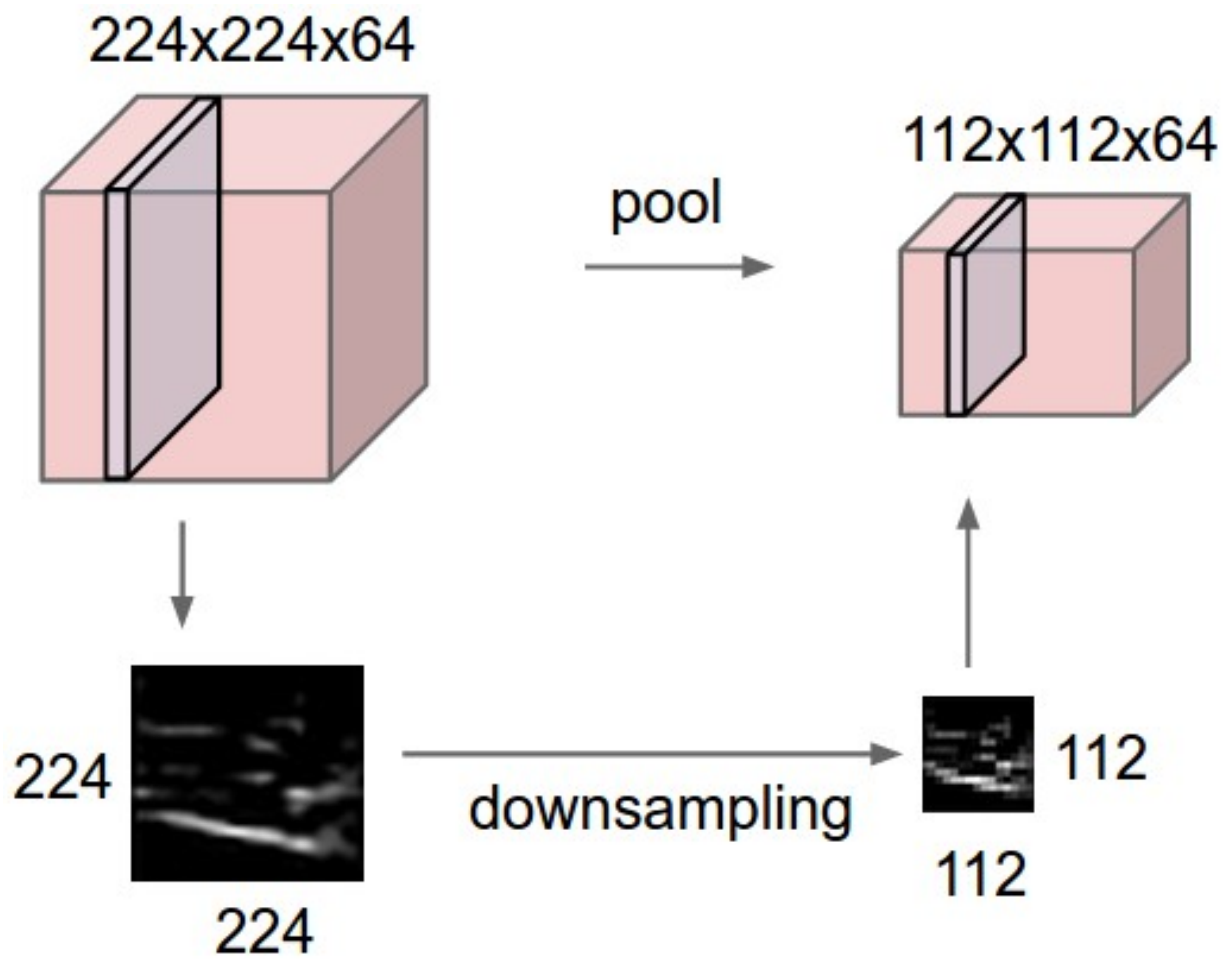


How many **parameters**?

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

Pooling Layer

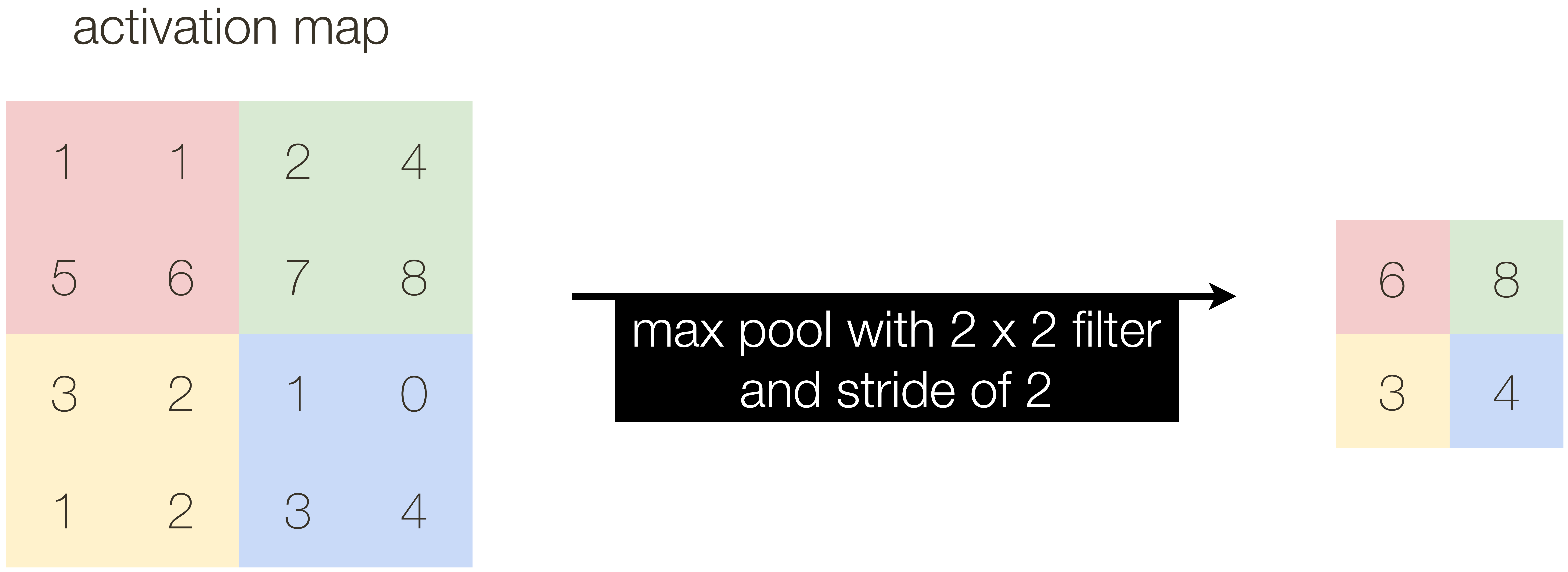
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



How many **parameters**?

None!

Max **Pooling**



* 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

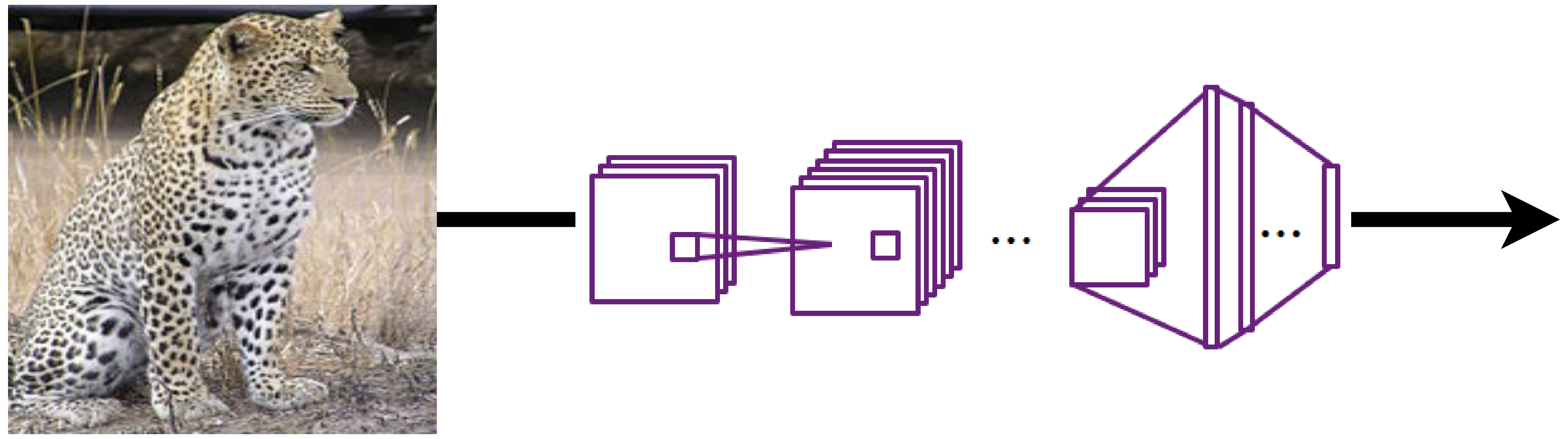
3.25	5.25
2	2

Object **Classification**

Category	Prediction
Dog	No
Cat	No
Couch	No
Flowers	No
Leopard	**Yes**
…	…

Problem: For each image predict which category it belongs to out of a fixed set

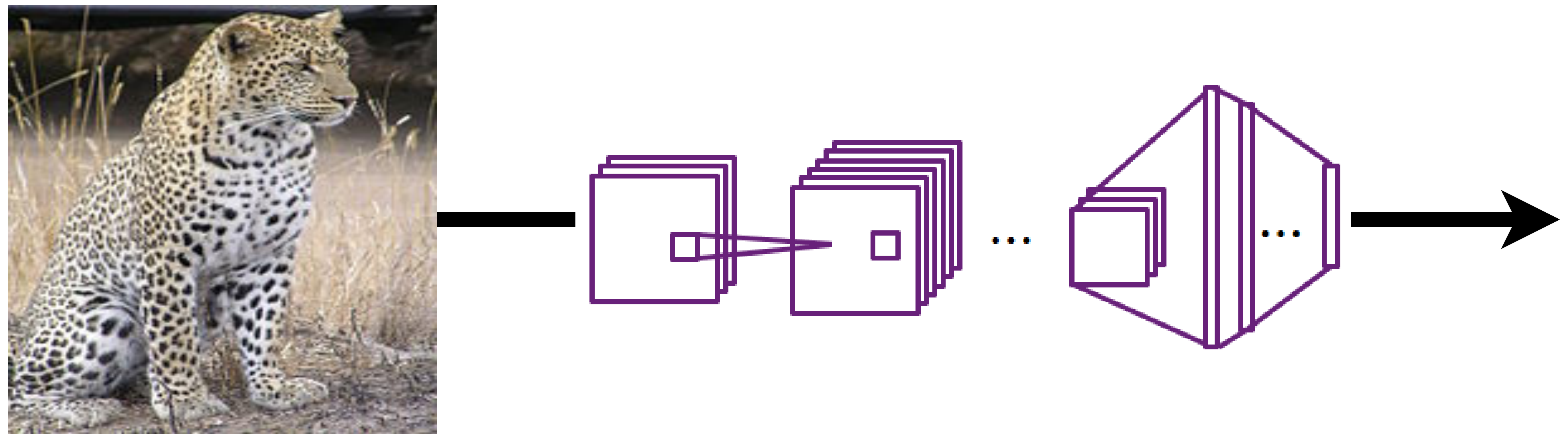
Object **Classification**

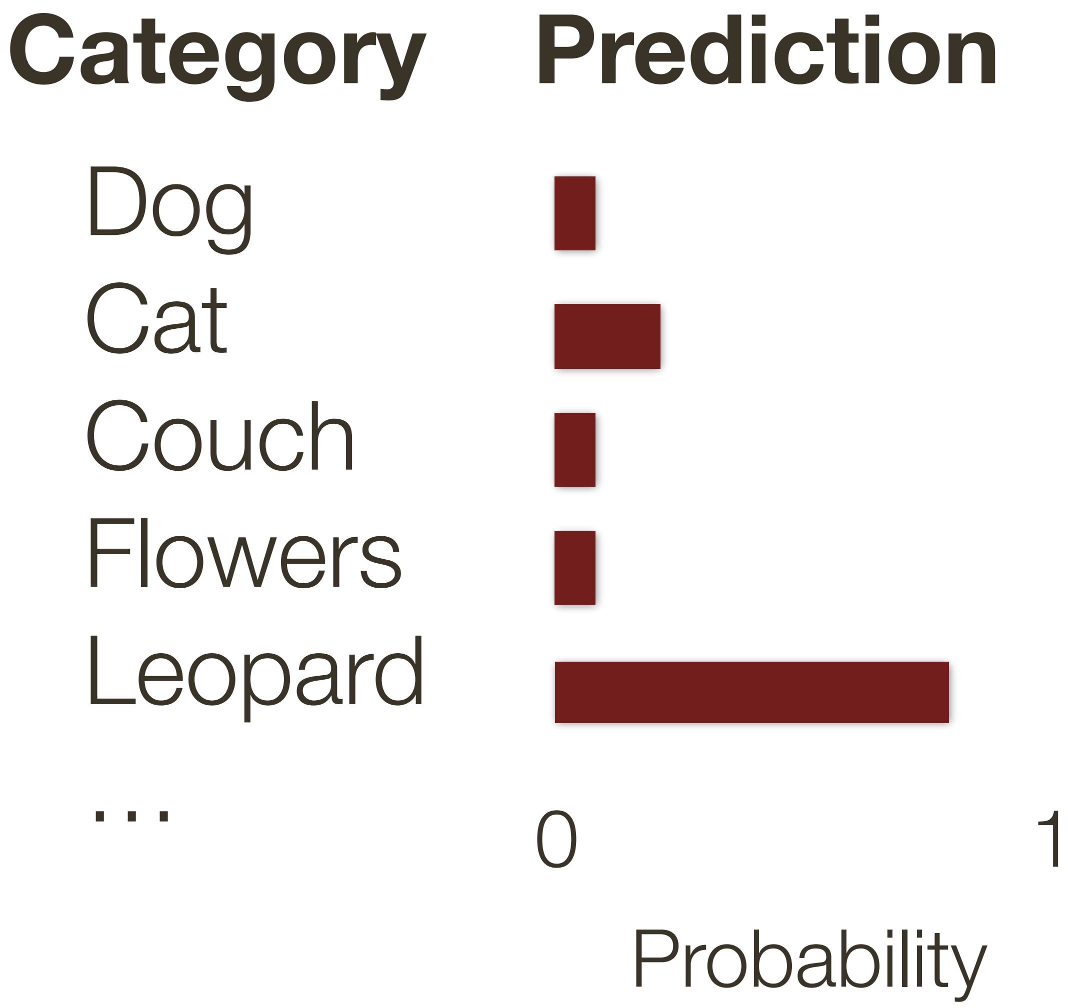


Category	**Prediction**
Dog	No
Cat	No
Couch	No
Flowers	No
Leopard	**Yes**
…	…

Problem: For each image predict which category it belongs to out of a fixed set

Object **Classification**





Problem: For each image predict which category it belongs to out of a fixed set

R-CNN

[Girshick et al, CVPR 2014]

Input **Image**

* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]

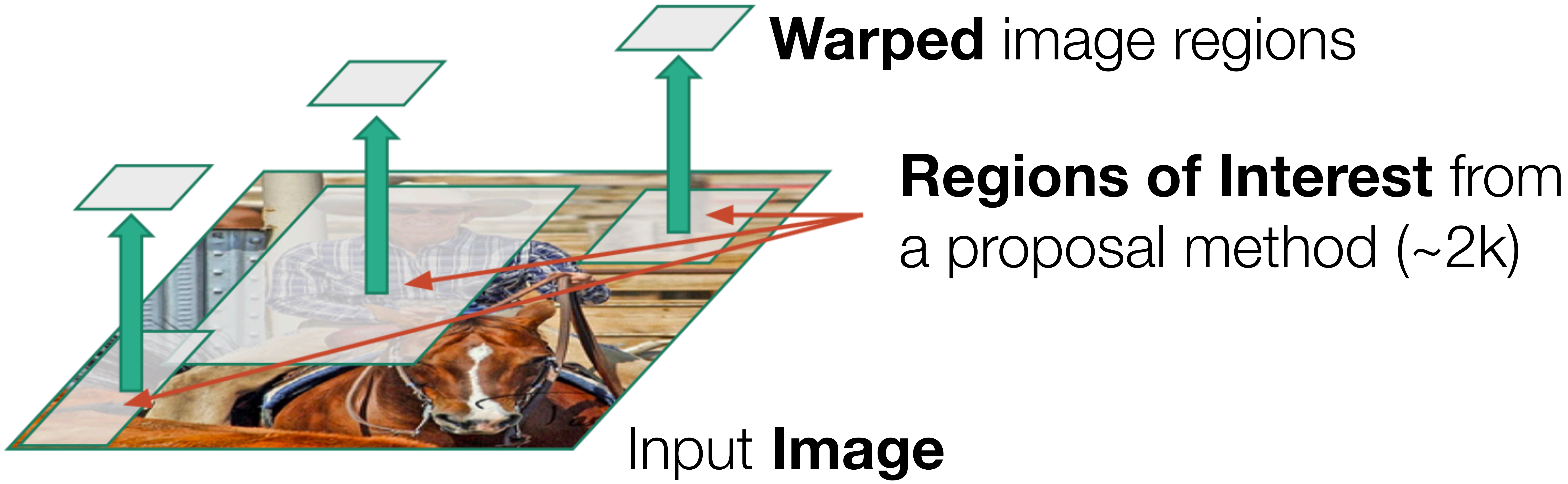
Regions of Interest from a proposal method (~2k)

Input **Image**

* image from Ross Girshick

R-CNN

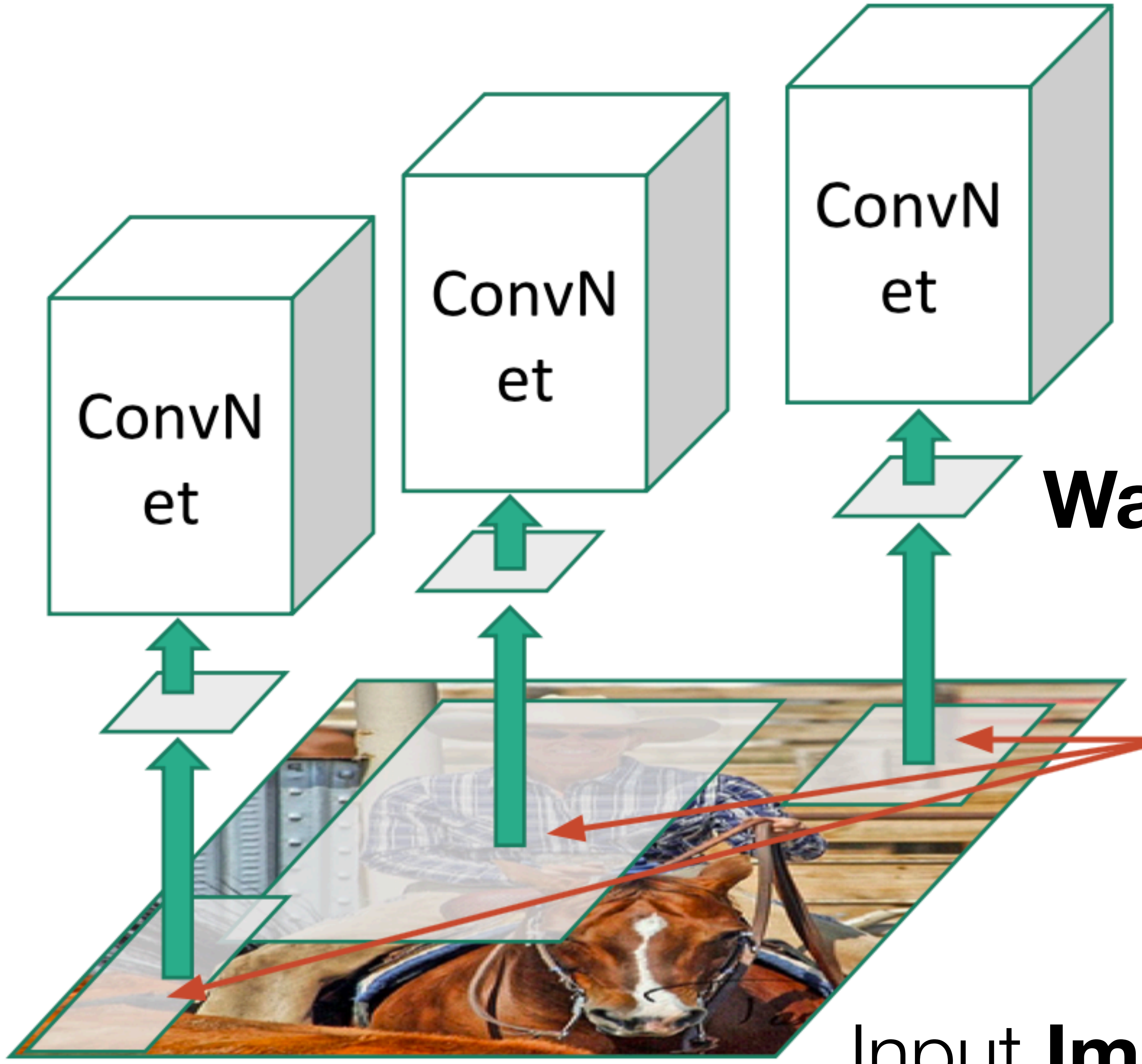
[Girshick et al, CVPR 2014]



* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]



Forward each region through a **CNN**

Warped image regions

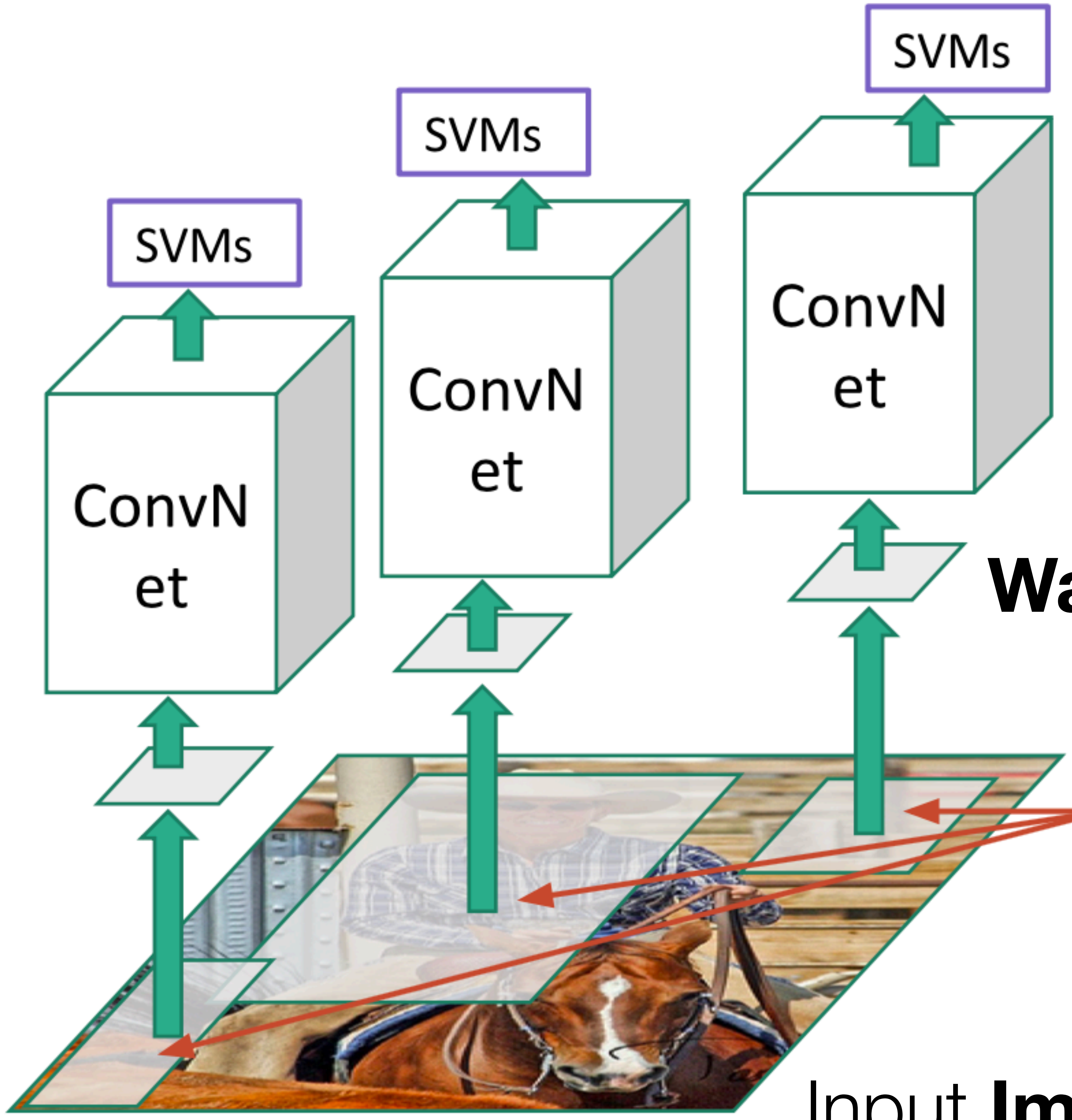
Regions of Interest from a proposal method (~2k)

Input **Image**

* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]



Classify regions with SVM

Forward each region through a **CNN**

Warped image regions

Regions of Interest from a proposal method (~2k)

Input **Image**

* image from Ross Girshick

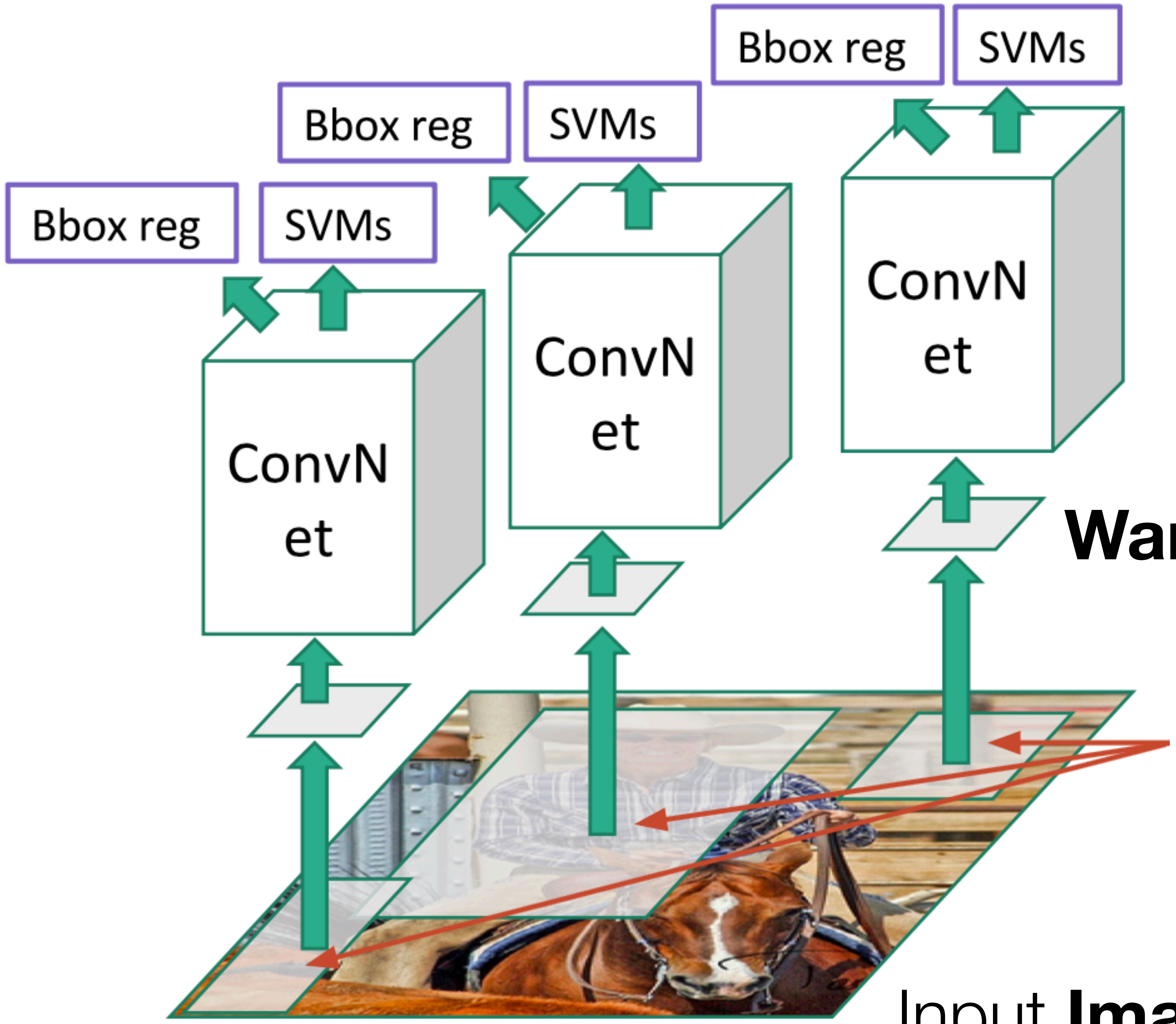
R-CNN

Linear Regression for bounding box offsets

[Girshick et al, CVPR 2014]

Classify regions with SVM

Forward each region through a **CNN**



Warped image regions

Regions of Interest from a proposal method (~2k)

Input Image

* image from Ross Girshick

R-CNN

R-CNN (Regions with CNN features) algorithm:

— Extract promising candidate regions using an object proposals algorithm

— Resize each proposal window to the size of the input layer of a trained convolutional neural network

— Input each resized image patch to the convolutional neural network

Implementation detail: Instead of using the classification scores of the network directly, the output of the final fully-connected layer can be used as an input feature to a trained support vector machine (SVM)

Summary

The parameters of a neural network are learned using **backpropagation**, which computes gradients via recursive application of the chain rule

A **convolutional neural network** assumes inputs are images, and constrains the network architecture to reduce the number of parameters

A **convolutional layer** applies a set of learnable filters

A **pooling layer** performs spatial downsampling

A **fully-connected** layer is the same as in a regular neural network

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

Please fill out

Student Evaluations

(on Canvas)