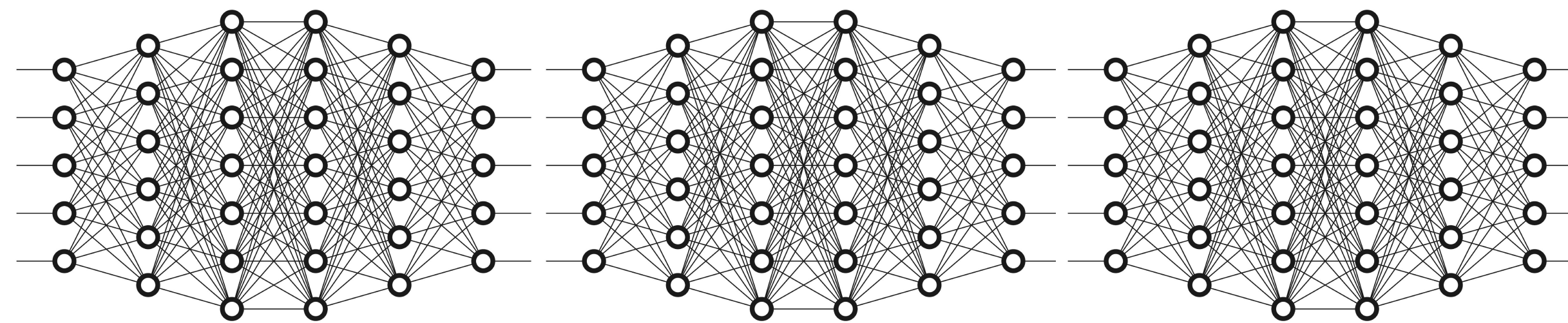




CPSC 425: Computer Vision



Lecture 22: Neural Networks (cont), CNNs

Menu for Today

Topics:

- Convolutional Layers
- Pooling Layers
- Classification, Detection
- Quiz 5 and 6 plan

Readings:

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

Reminders:

- **Assignment 6:** Deep Learning is out and due next **Thursday**
- **Material** for **Final Prep** is on Canvas (some questions we didn't cover)
- **Additional lecture** notes will be available soon

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



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



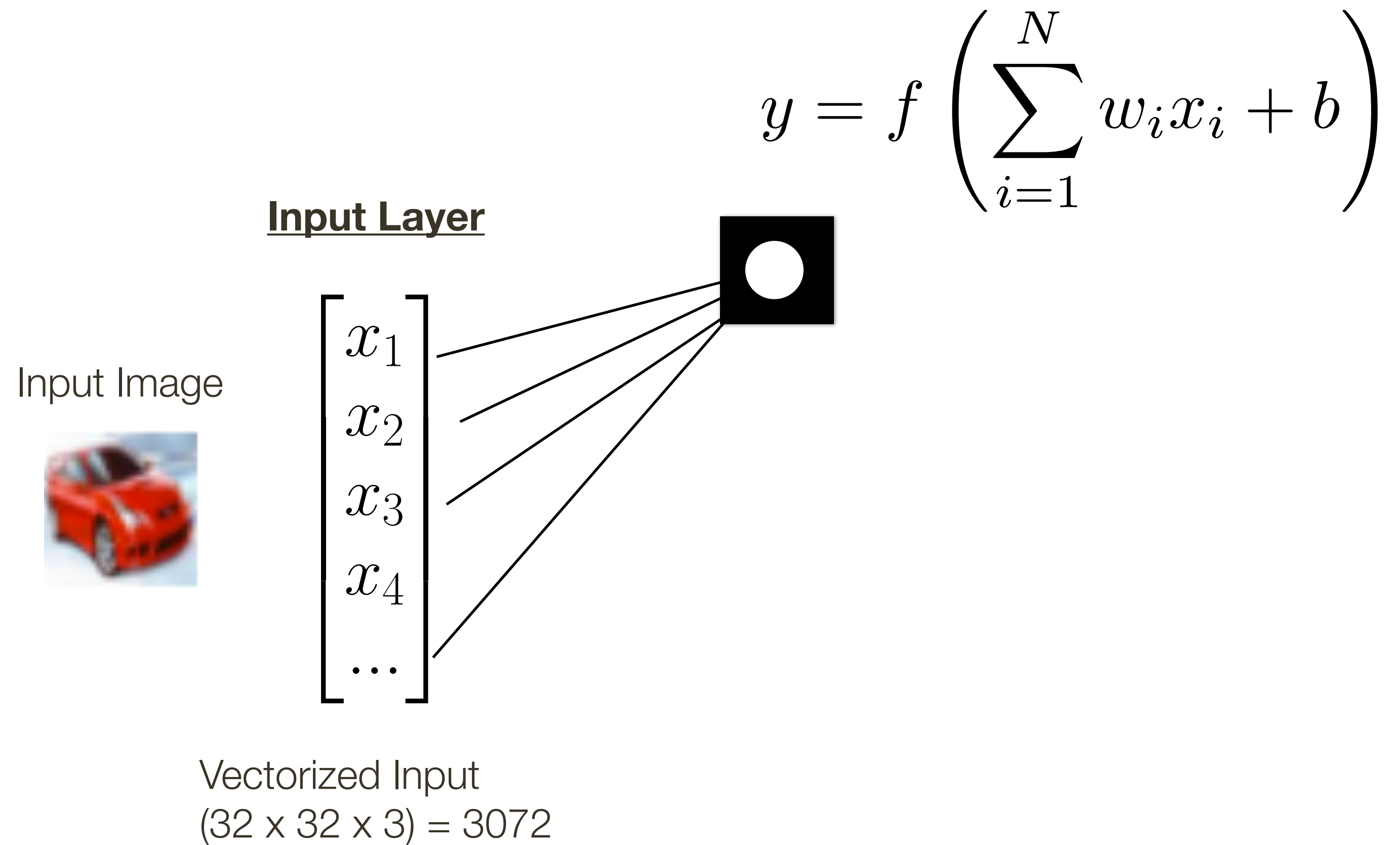
ViperGPT

Dídac Surís* Sachit Menon* Carl Vondrick
Columbia University

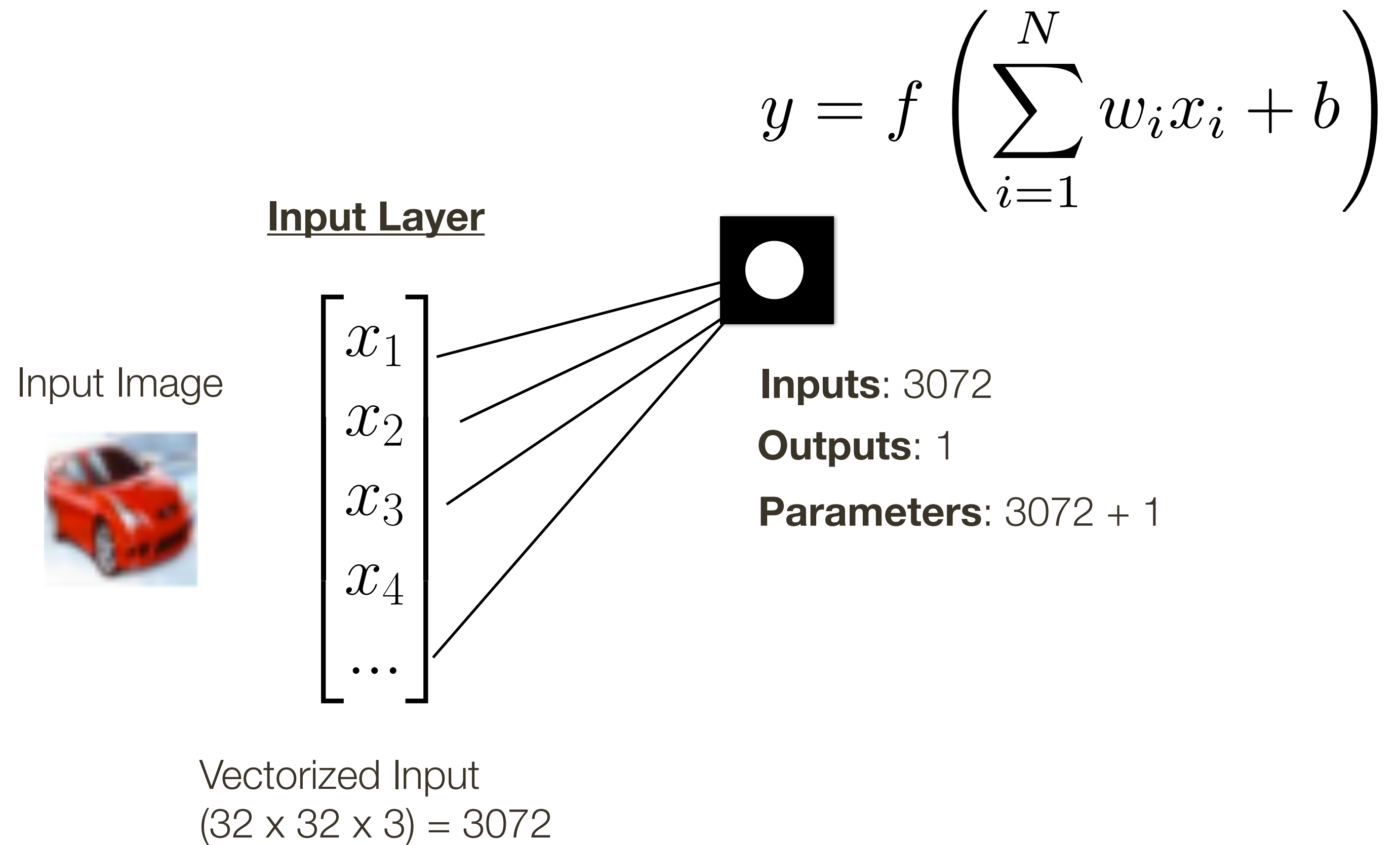
ViperGPT

Dídac Surís* Sachit Menon* Carl Vondrick
Columbia University

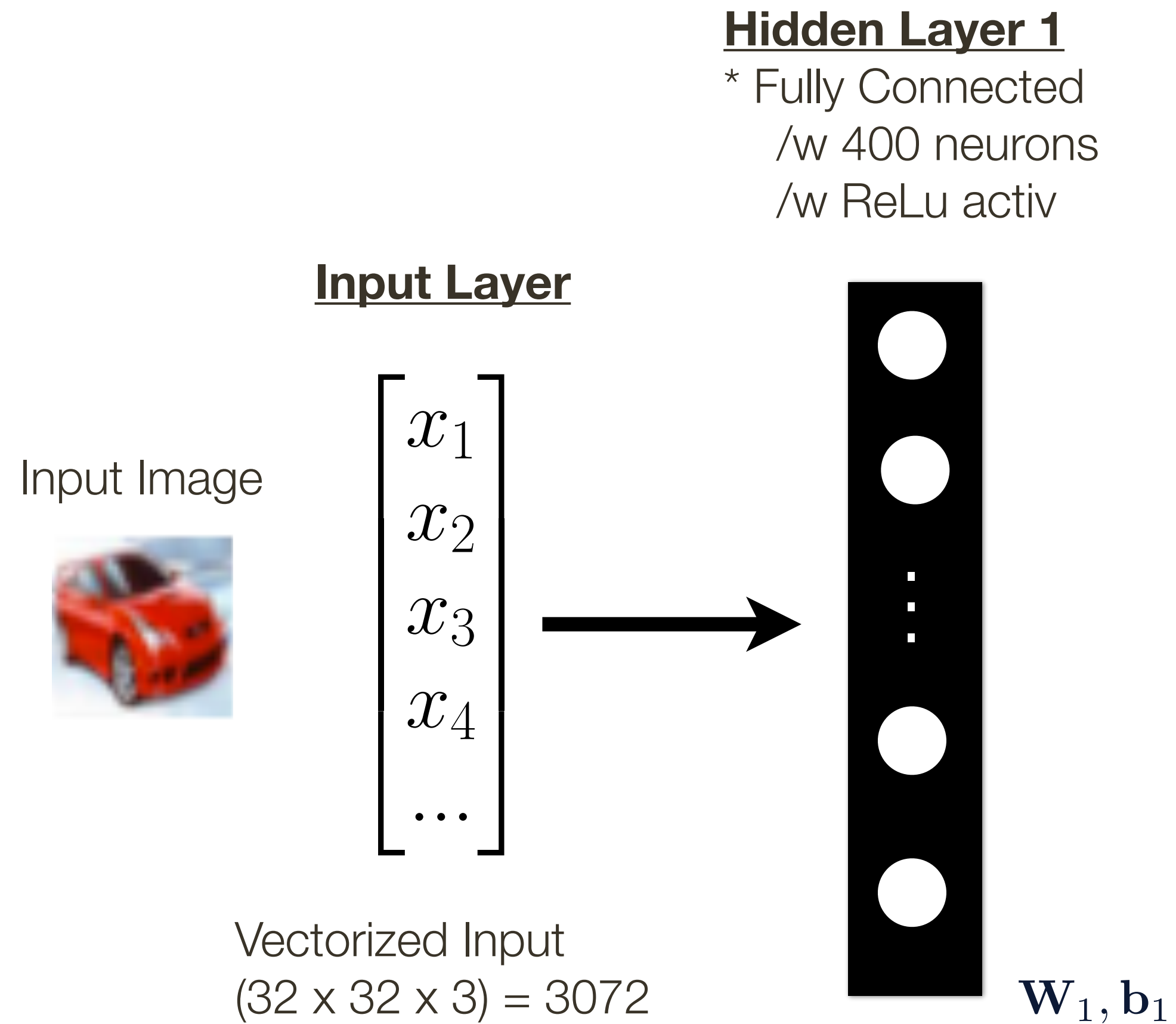
Neural Network: Short Review



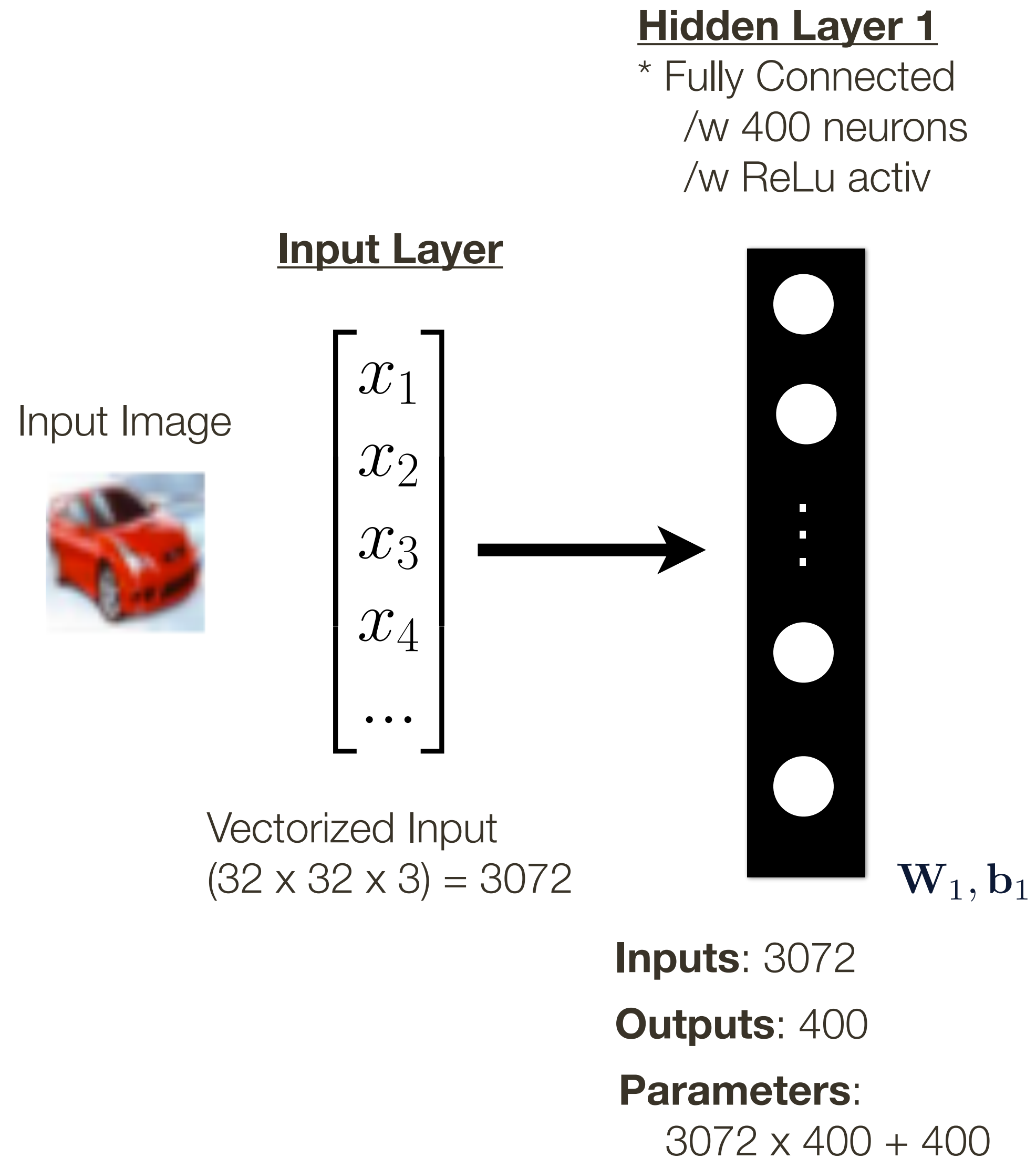
Neural Network: Short Review



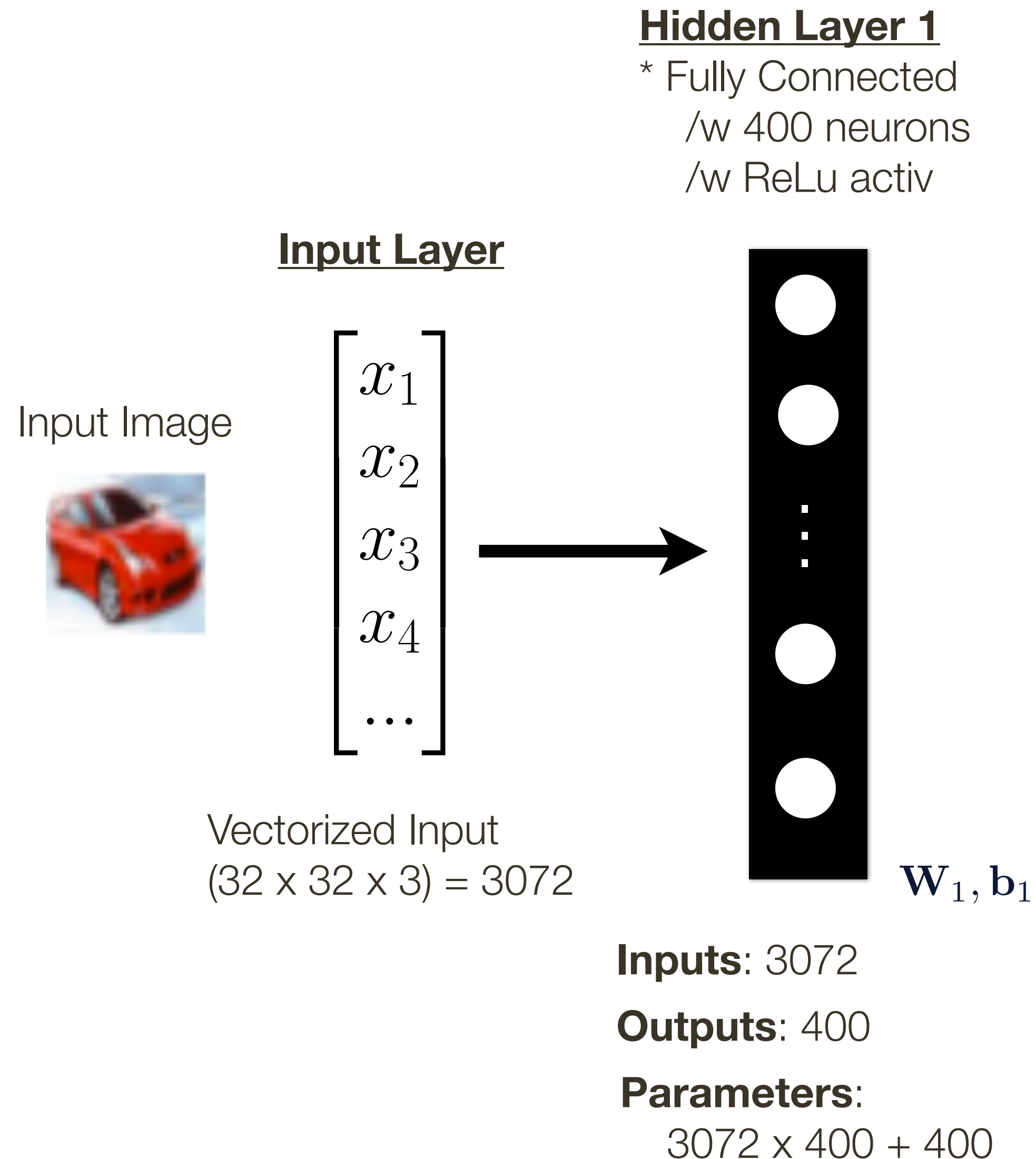
Neural Network: Short Review



Neural Network: Short Review

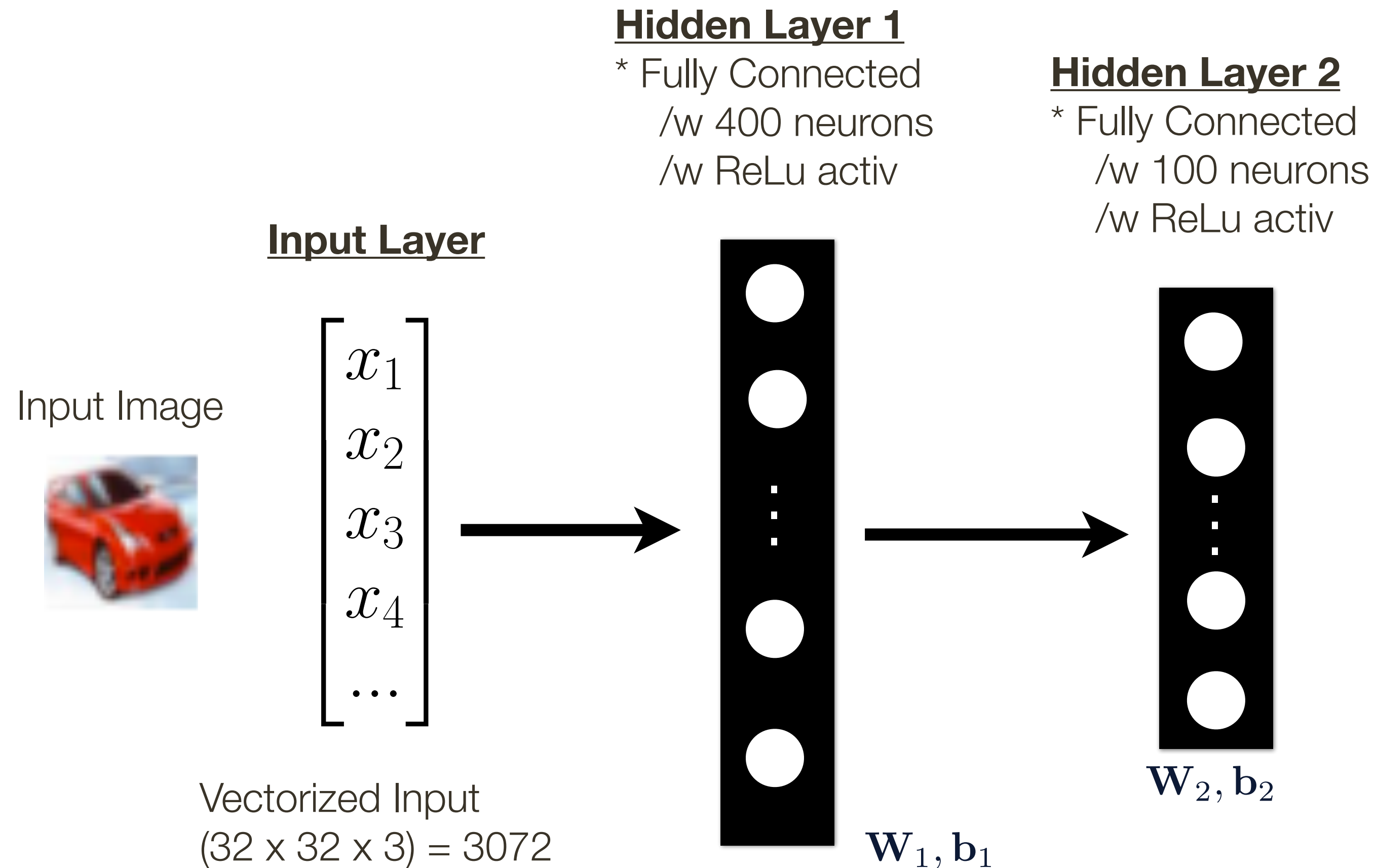


Neural Network: Short Review



Note: All neurons within a layer can be computed in parallel, making computations very efficient (especially on GPUs!, which are designed for parallelism)

Neural Network: Short Review



Note: Across layers computations are sequential

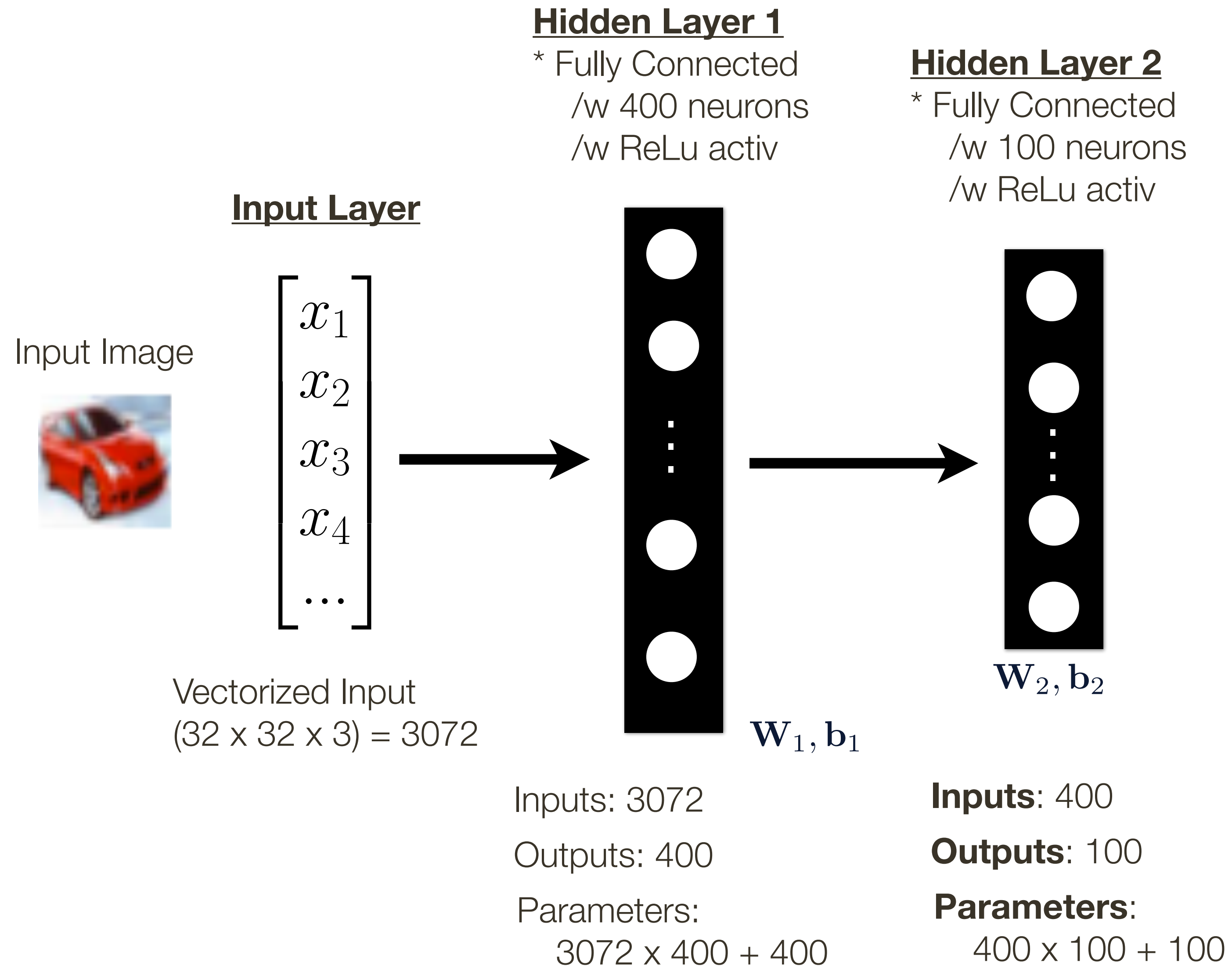
Inputs: 3072

Outputs: 400

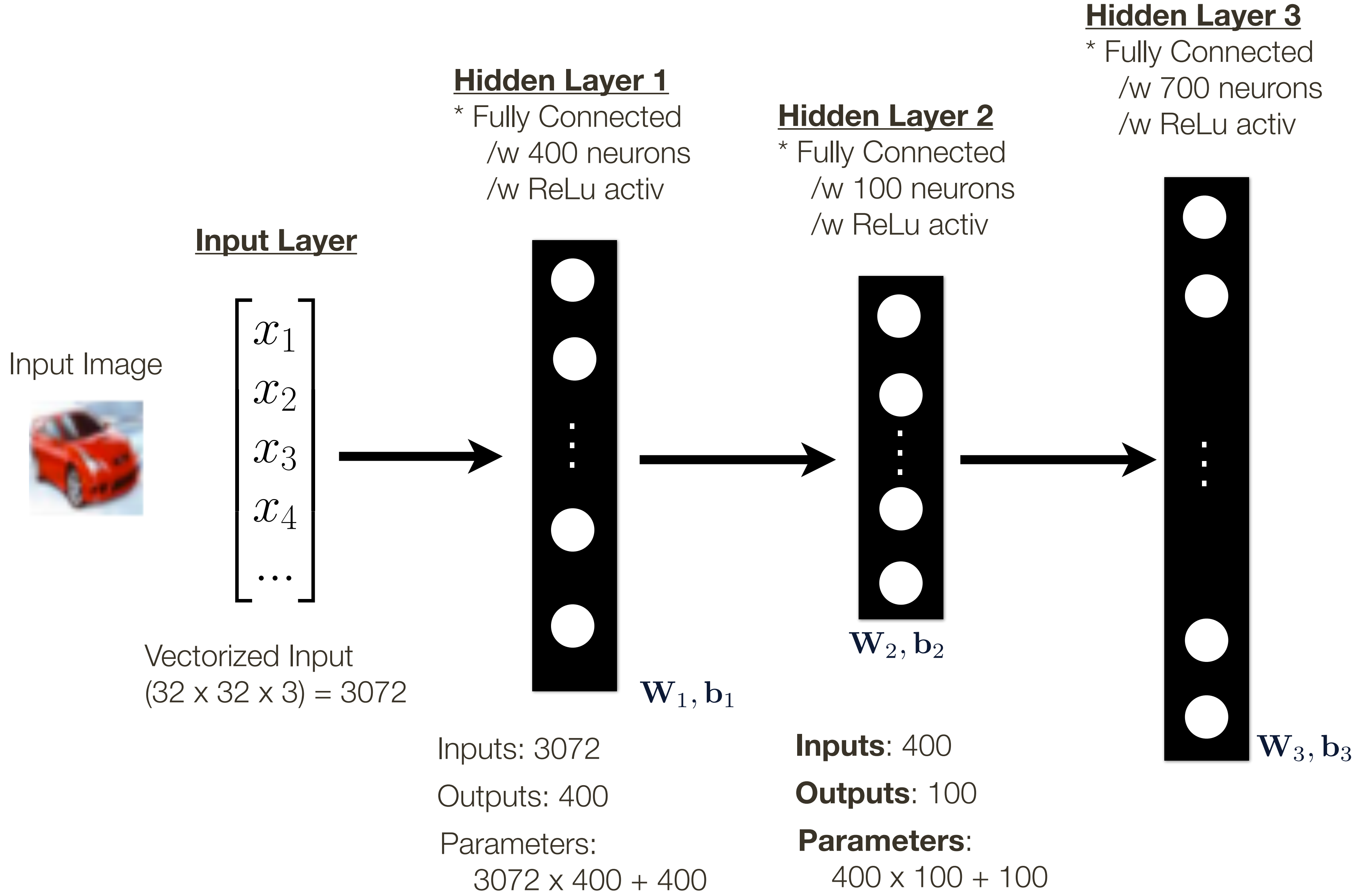
Parameters:

$3072 \times 400 + 400$

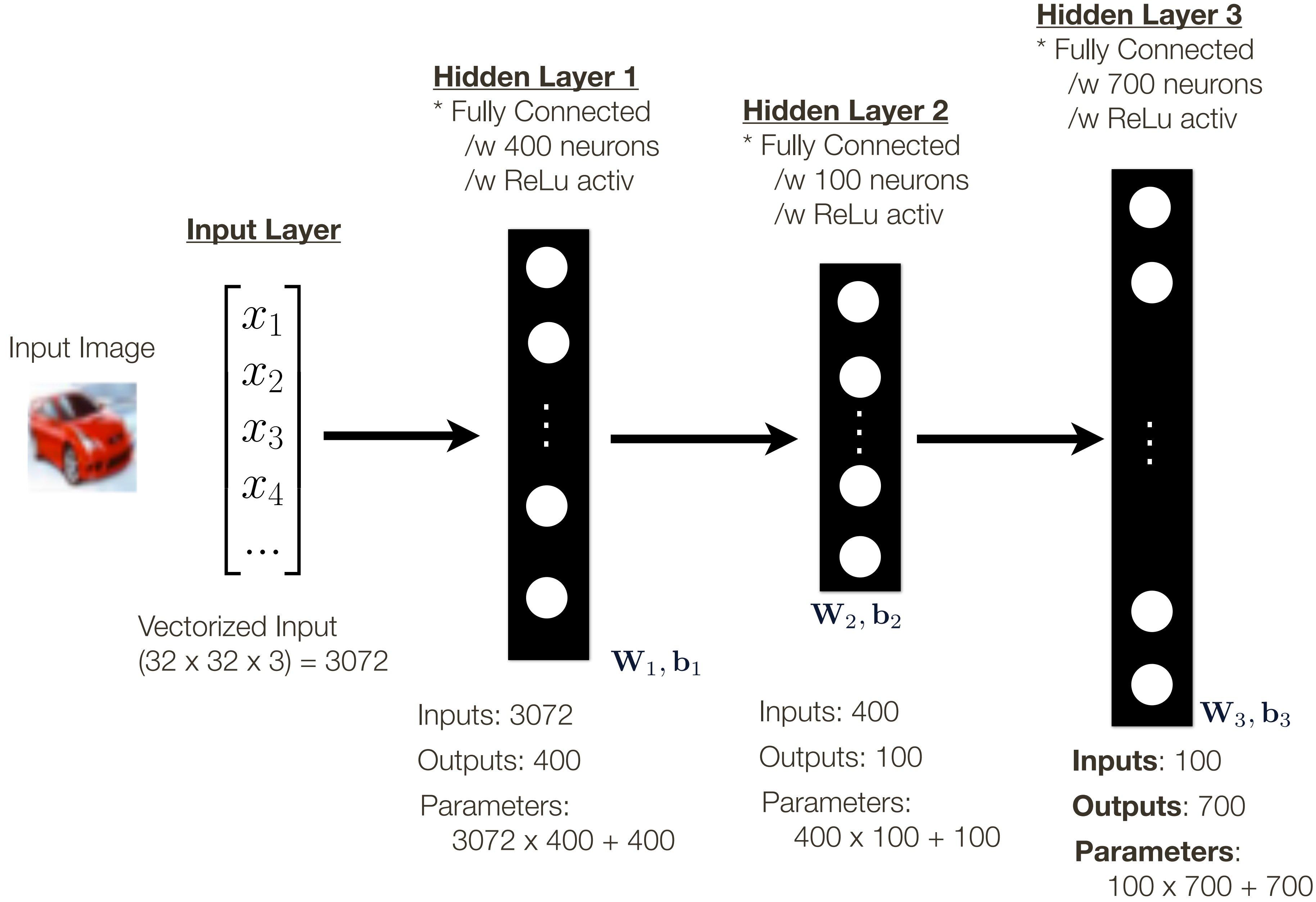
Neural Network: Short Review



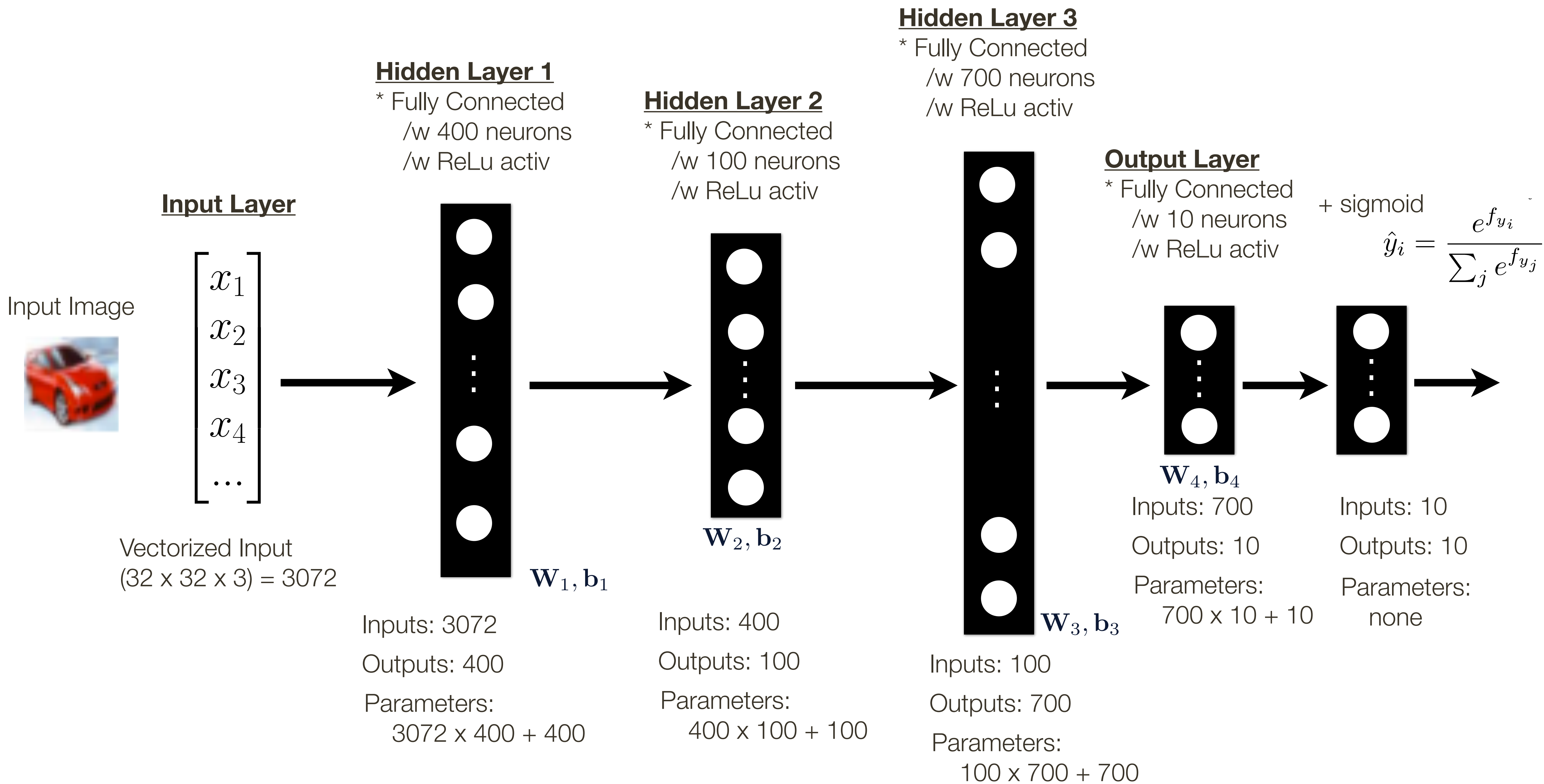
Neural Network: Short Review



Neural Network: Short Review

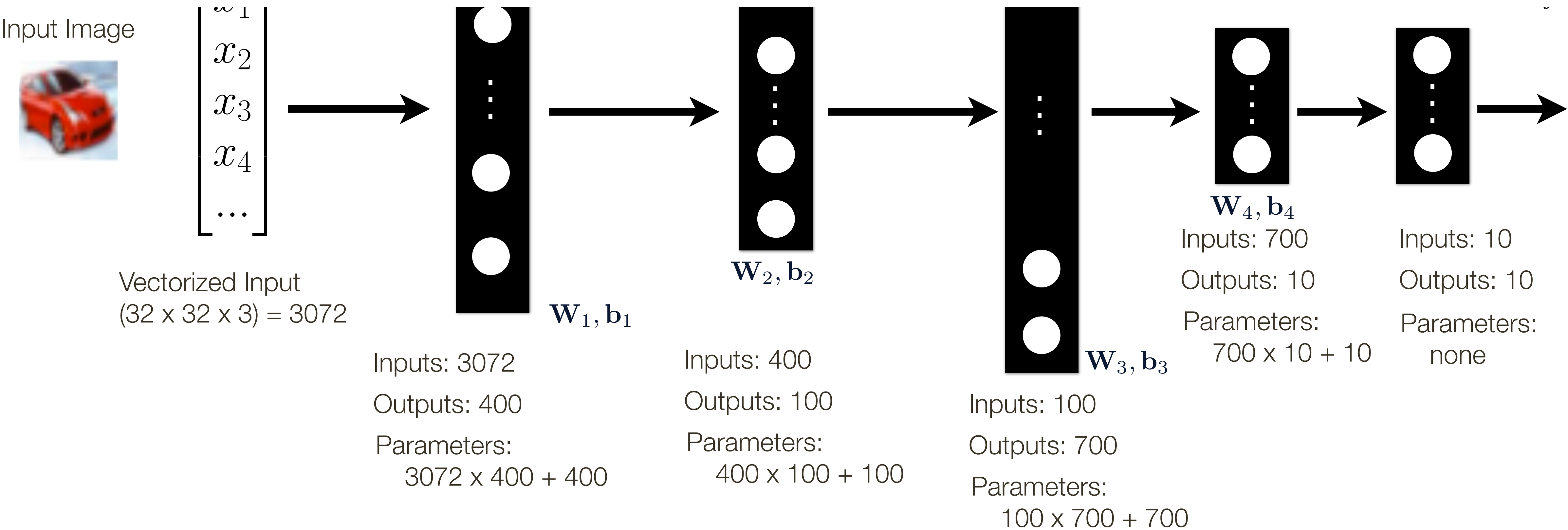


Neural Network: Short Review

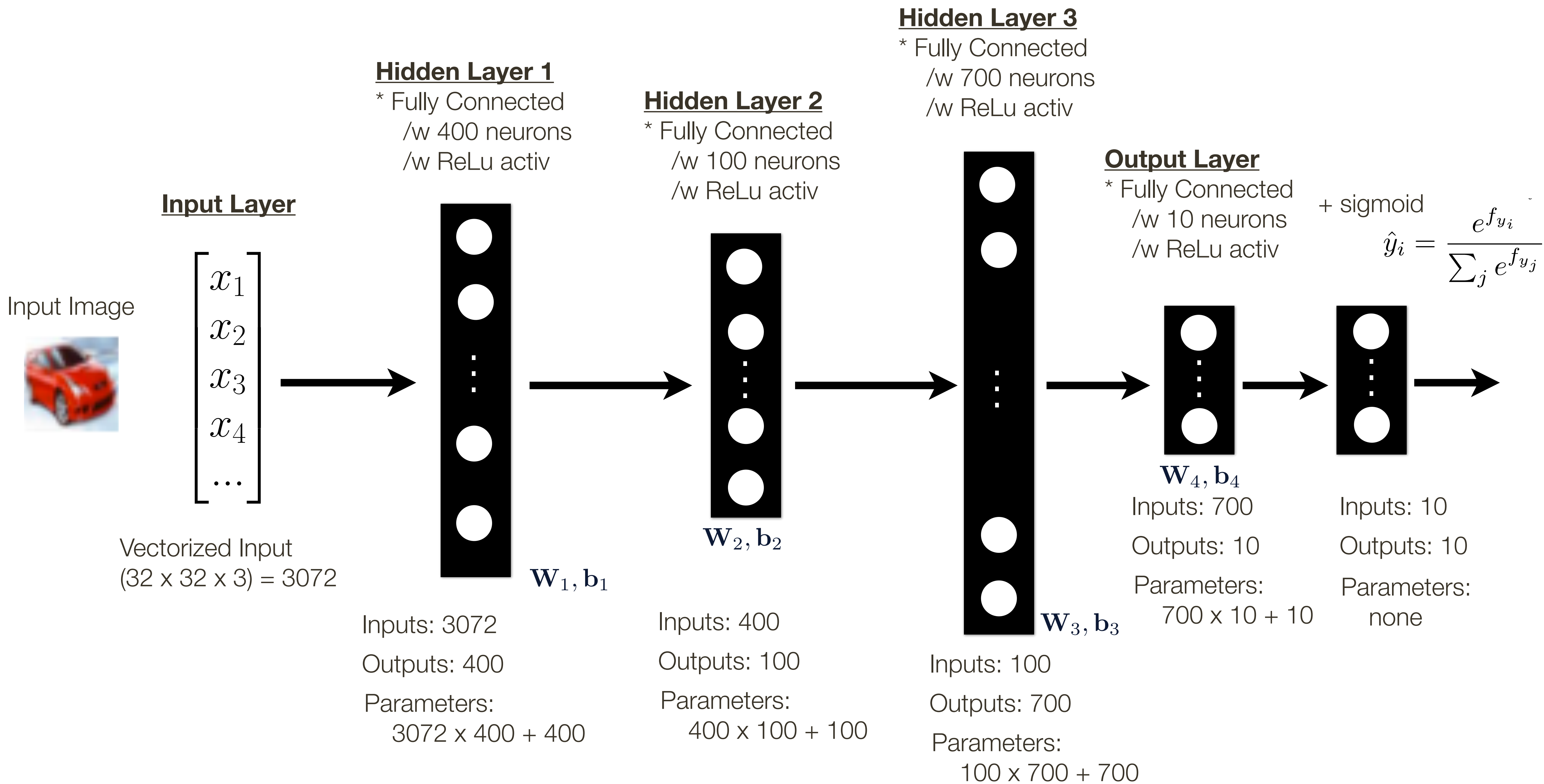


Neural Network: Short Review

This simple neural network has nearly 1.35 million parameters



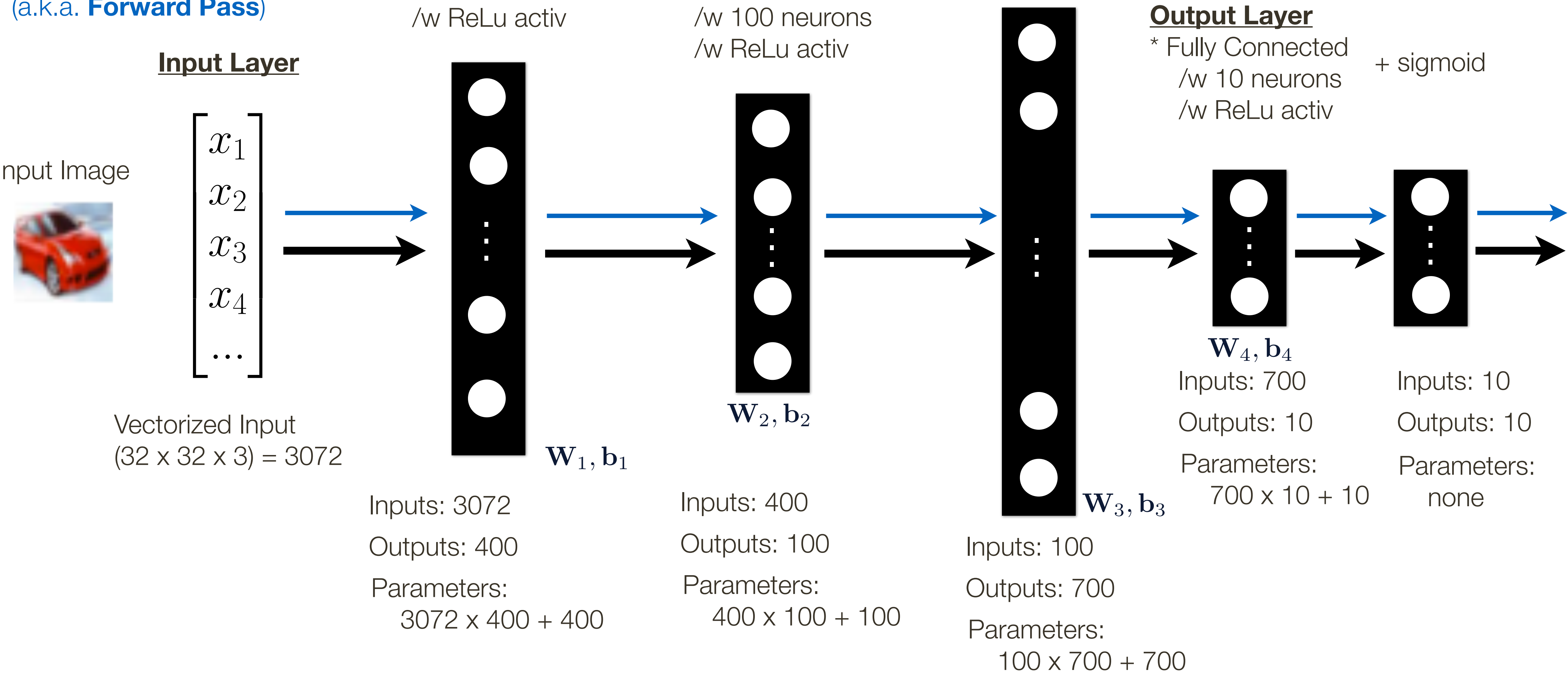
Neural Network: Short Review



Neural Network: Short Review

Inference: given values
for all parameters predict
output (probability)

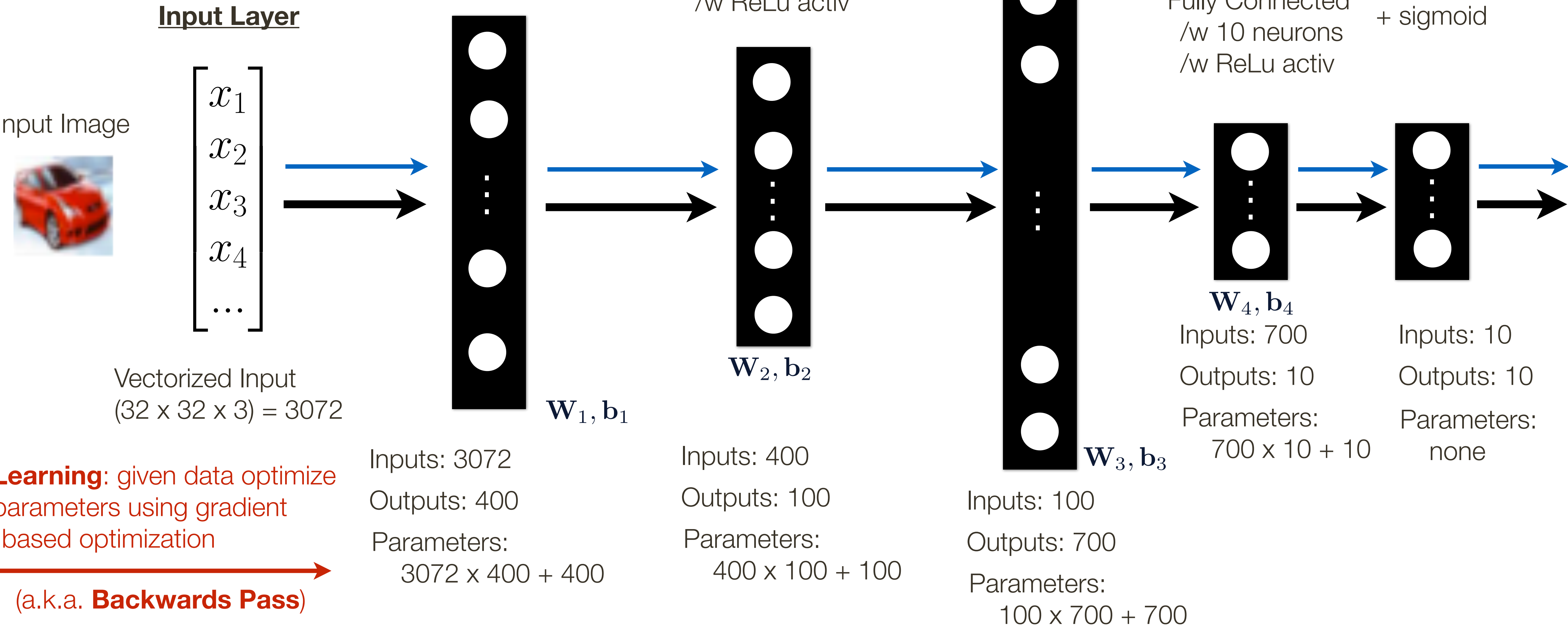
→
(a.k.a. **Forward Pass**)



Neural Network: Short Review

Inference: given values
for all parameters predict
output (probability)

→
(a.k.a. **Forward Pass**)



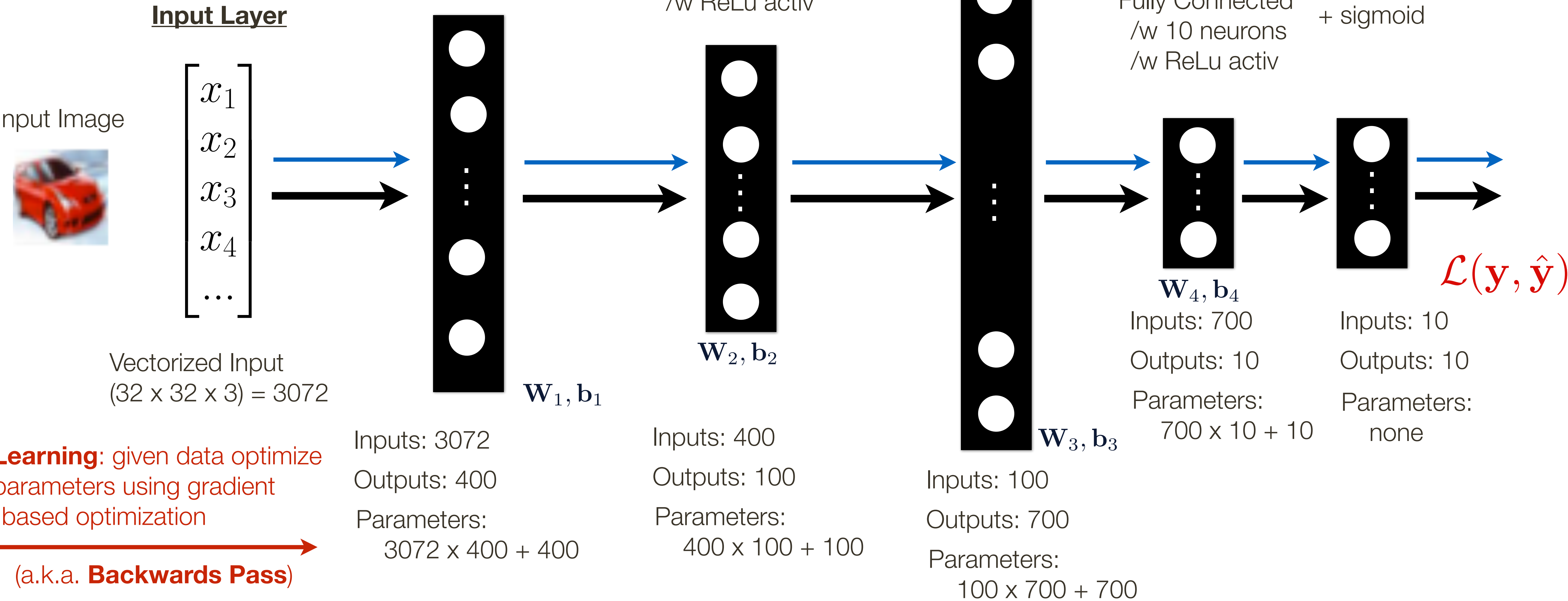
Learning: given data optimize
parameters using gradient
-based optimization

→
(a.k.a. **Backwards Pass**)

Neural Network: Short Review

Inference: given values
for all parameters predict
output (probability)

→
(a.k.a. **Forward Pass**)



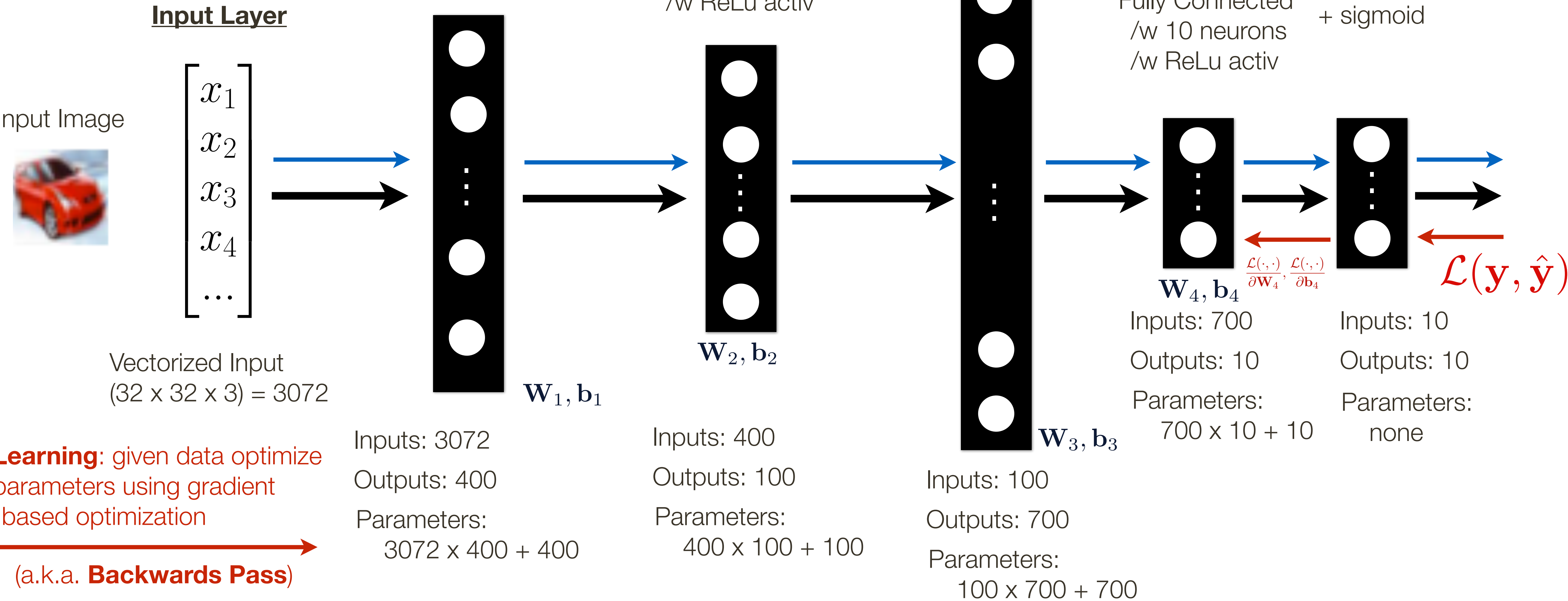
Learning: given data optimize
parameters using gradient
-based optimization

→
(a.k.a. **Backwards Pass**)

Neural Network: Short Review

Inference: given values for all parameters predict output (probability)

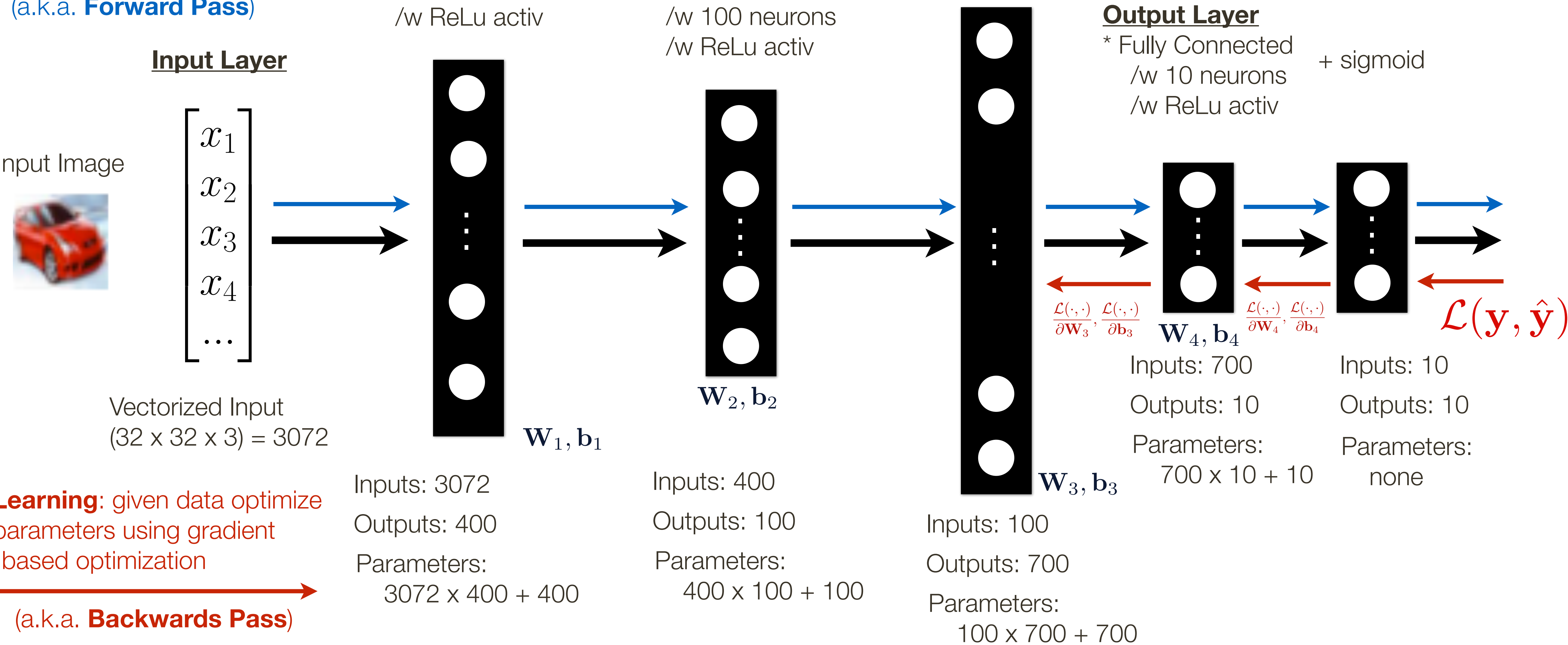
(a.k.a. **Forward Pass**)



Neural Network: Short Review

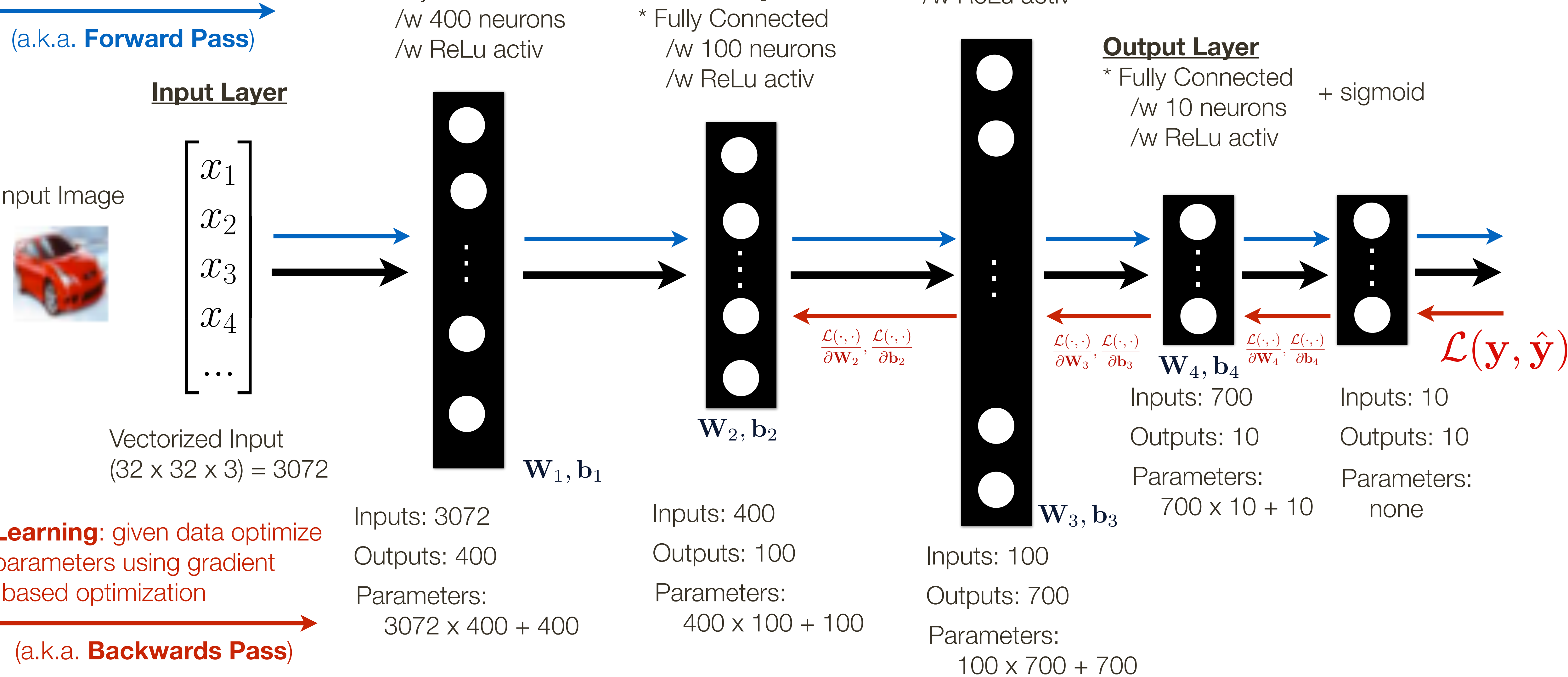
Inference: given values
for all parameters predict
output (probability)

(a.k.a. **Forward Pass**)



Neural Network: Short Review

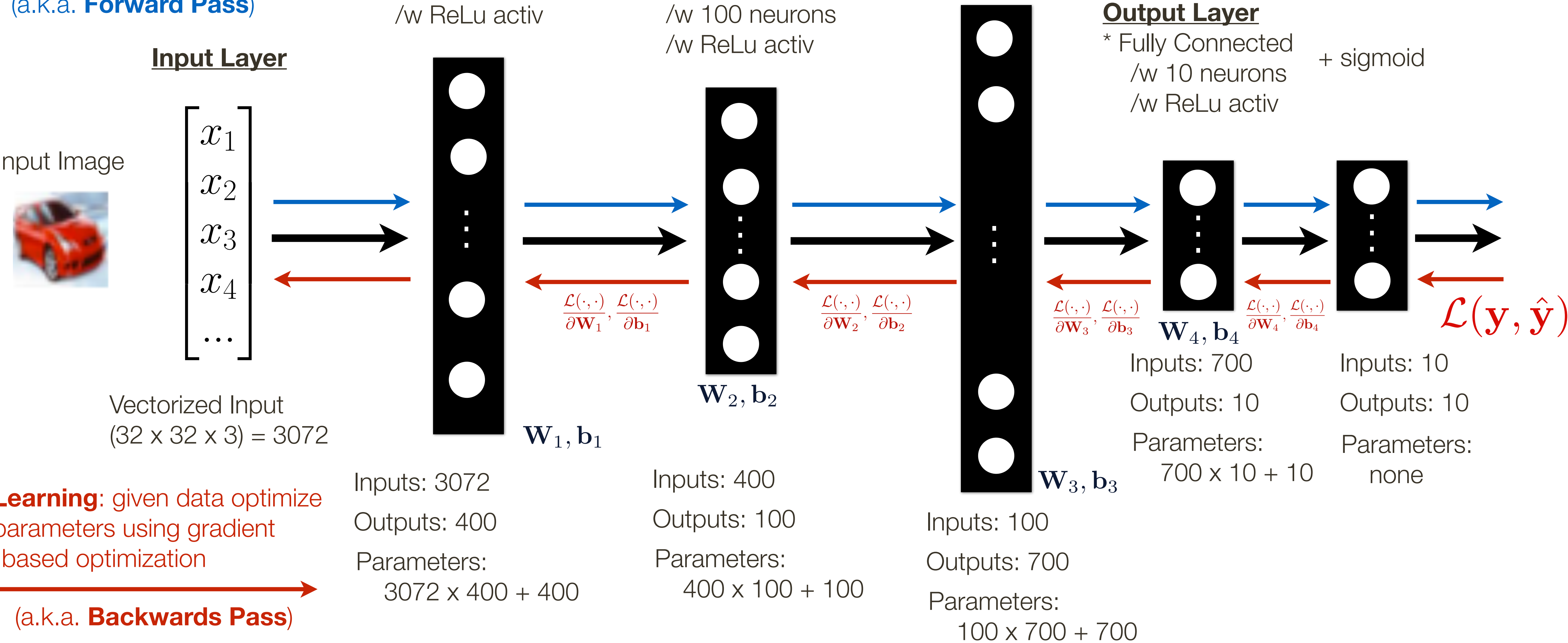
Inference: given values for all parameters predict output (probability)
(a.k.a. **Forward Pass**)



Learning: given data optimize parameters using gradient-based optimization
(a.k.a. **Backwards Pass**)

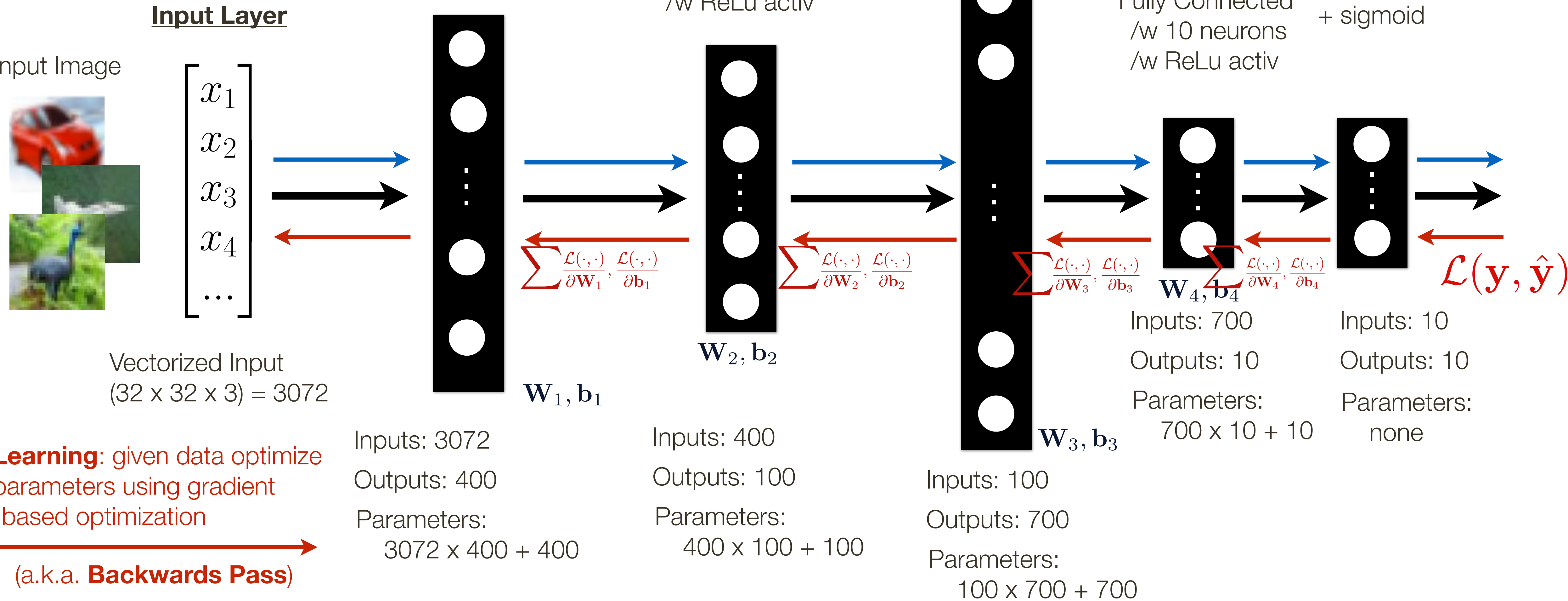
Neural Network: Short Review

Inference: given values for all parameters predict output (probability)
(a.k.a. **Forward Pass**)



Neural Network: Short Review

Inference: given values for all parameters predict output (probability)
(a.k.a. **Forward Pass**)

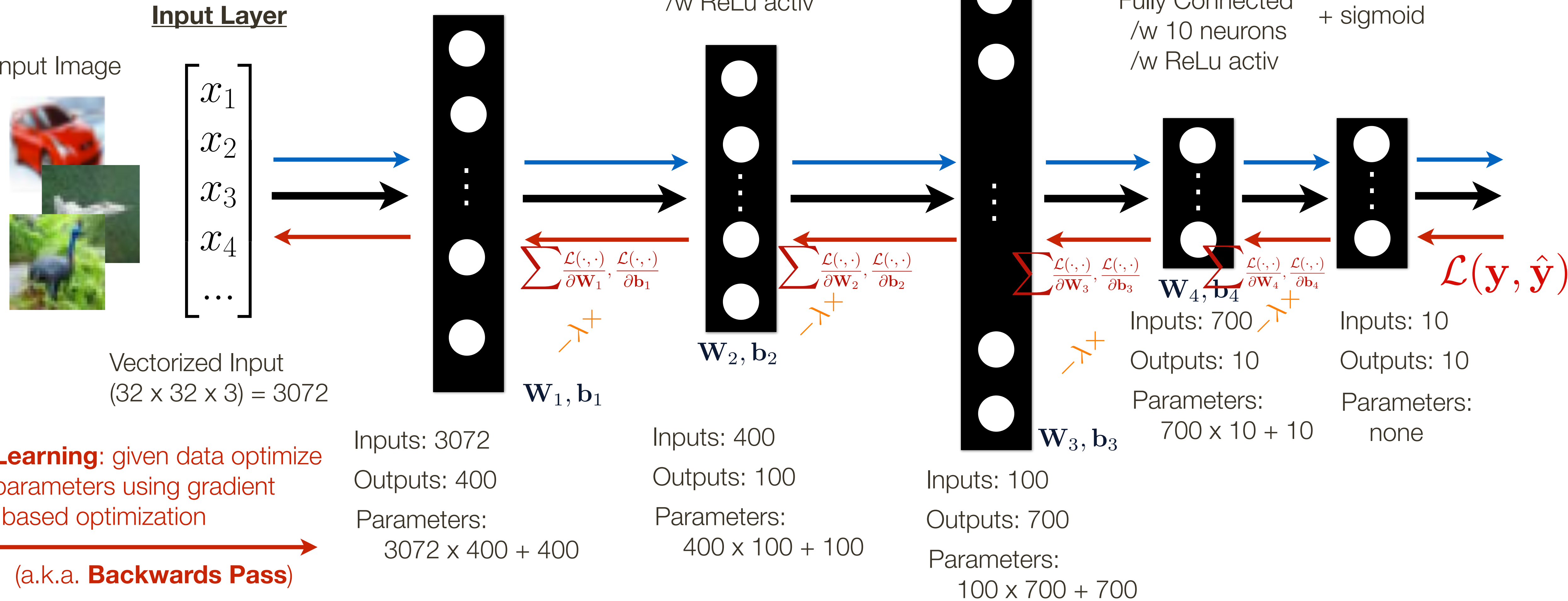


Learning: given data optimize parameters using gradient-based optimization
(a.k.a. **Backwards Pass**)

Neural Network: Short Review

Inference: given values for all parameters predict output (probability)

(a.k.a. **Forward Pass**)

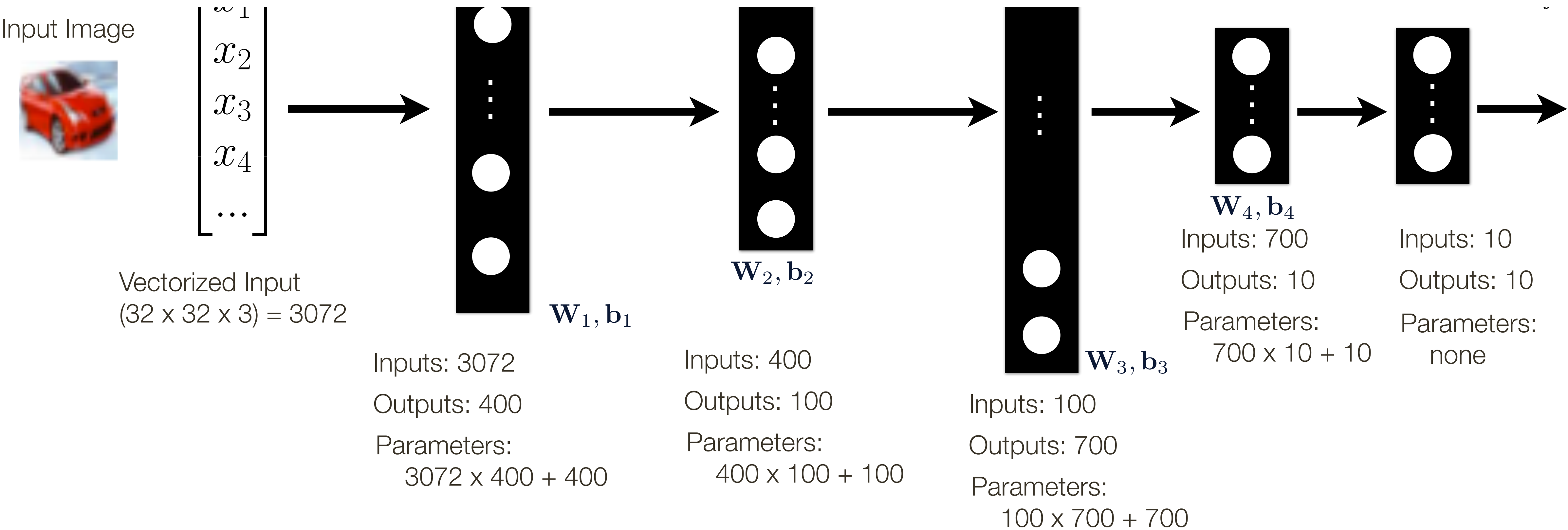


Learning: given data optimize parameters using gradient-based optimization

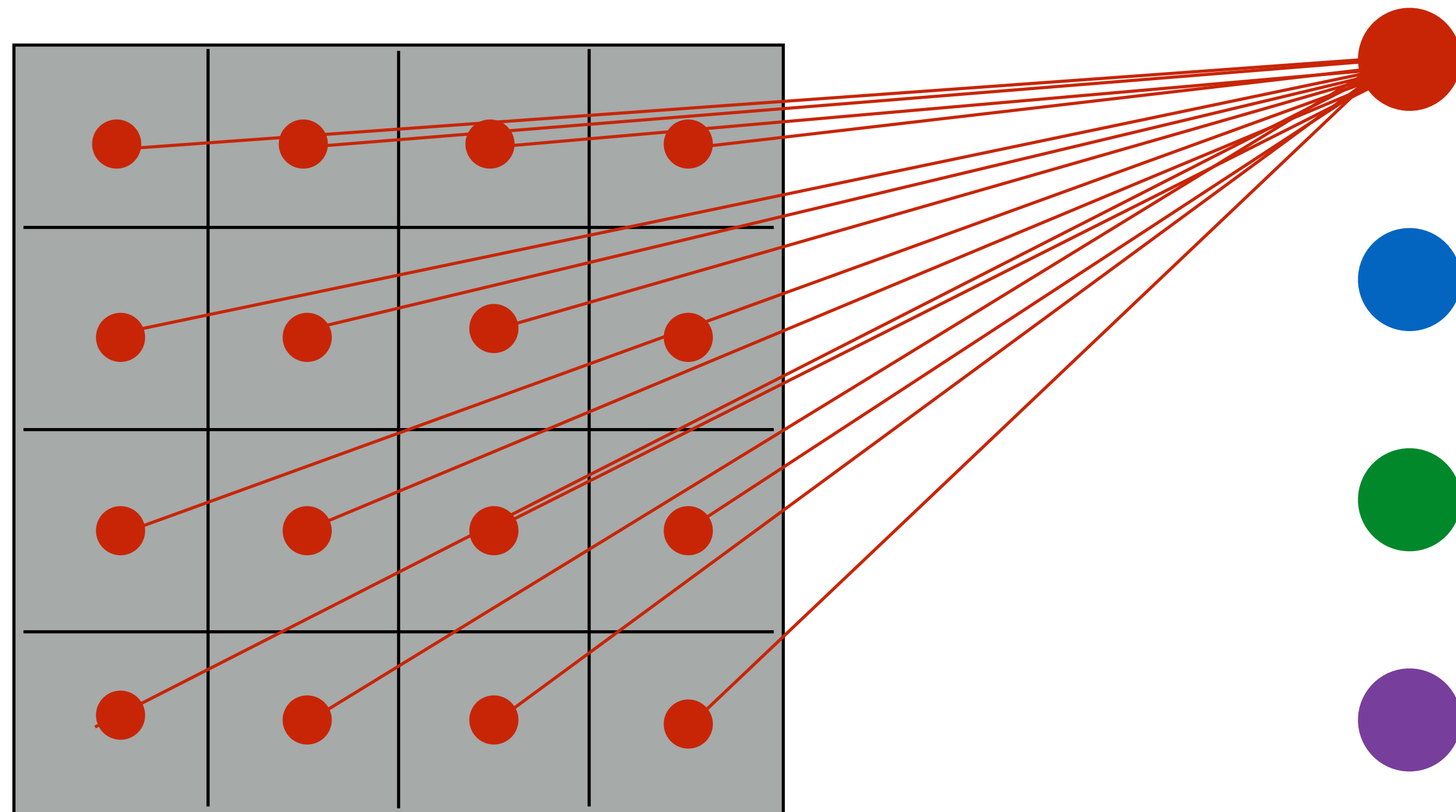
(a.k.a. **Backwards Pass**)

Neural Network: Short Review

This simple neural network has nearly 1.35 million parameters



Fully Connected Layer

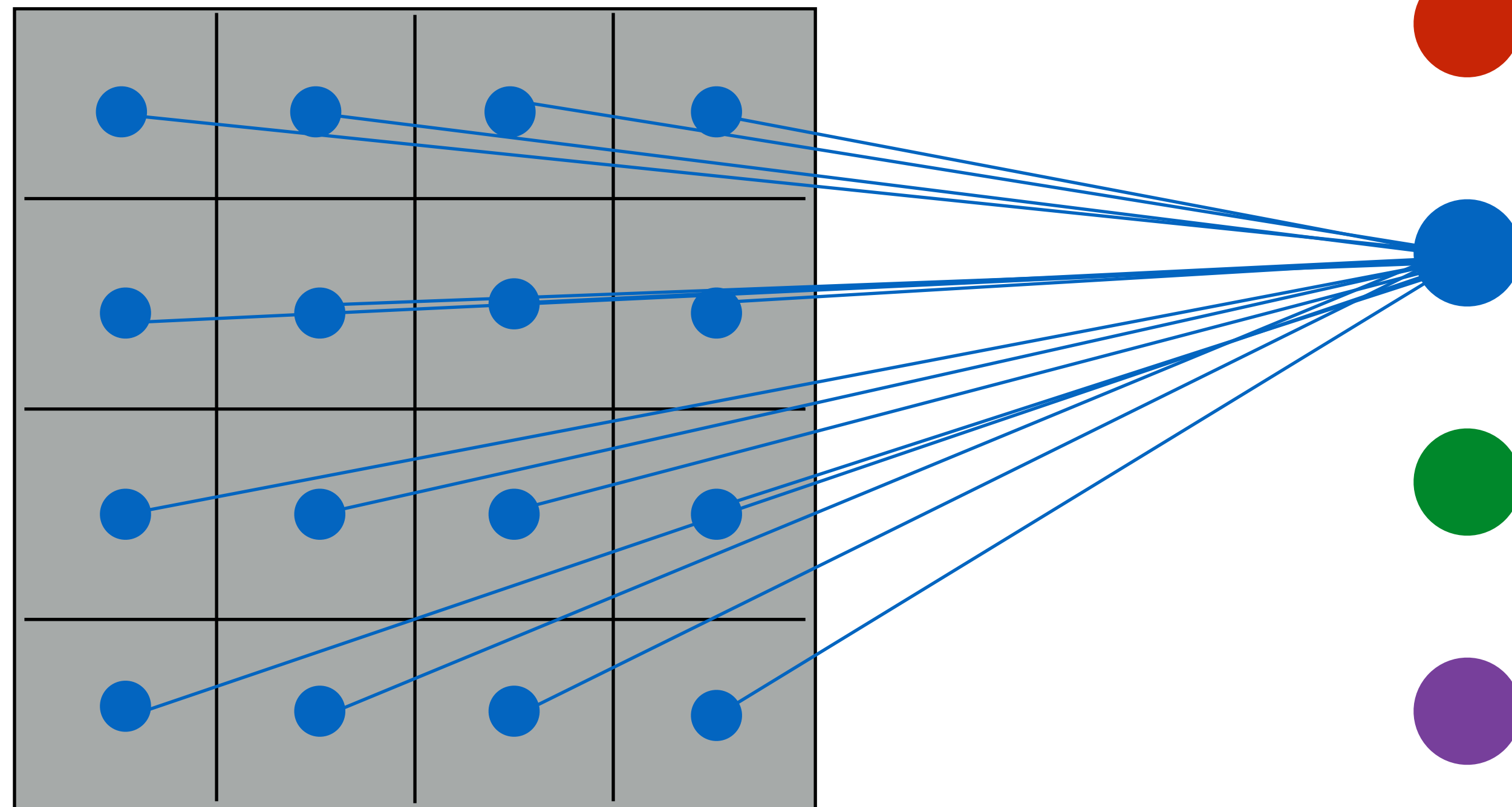


Fully Connected Layer

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

Linear Layer

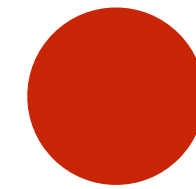
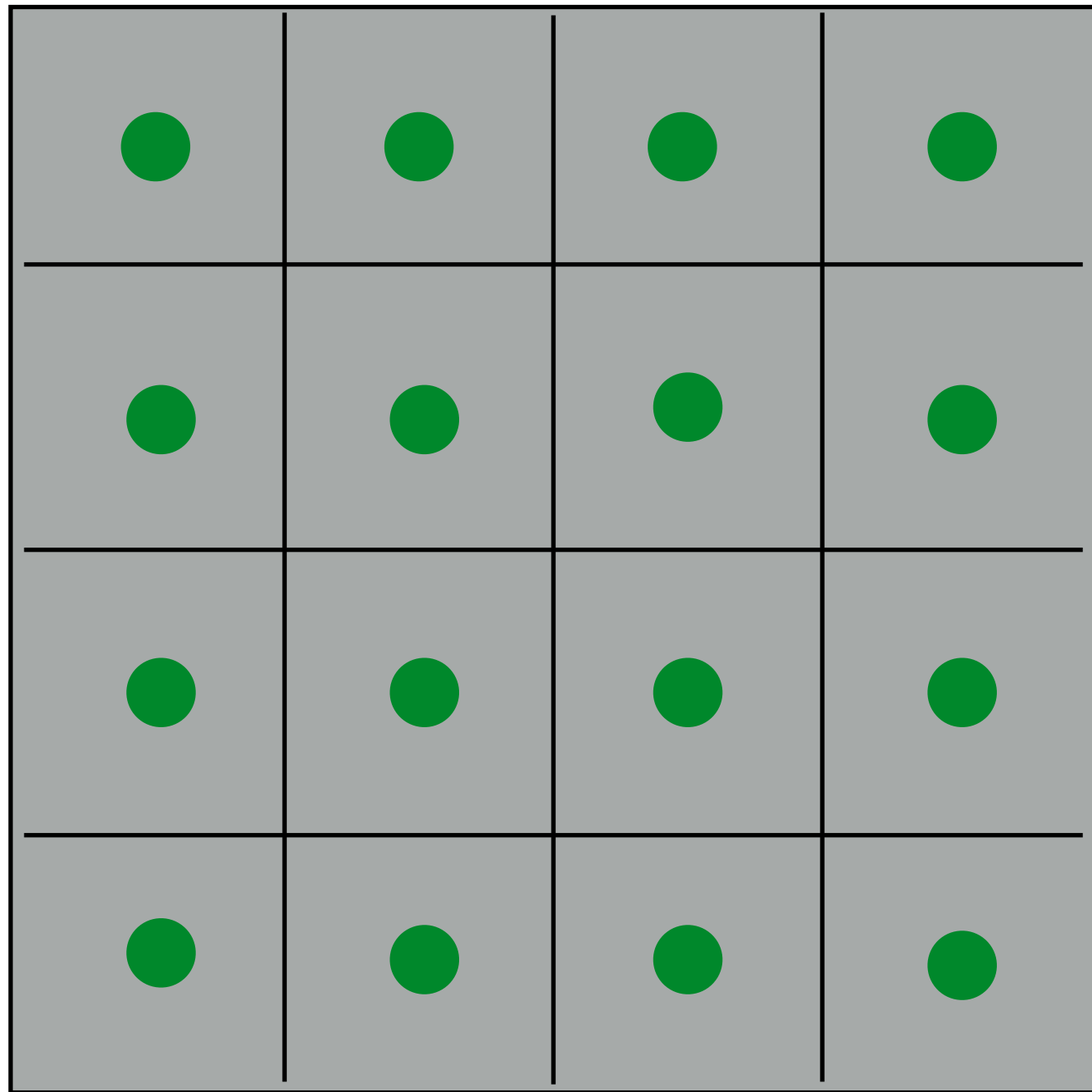
Fully Connected Layer



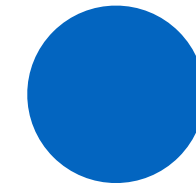
$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{1,i,j} \mathcal{I}(i,j) + b_1 \right)$$

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

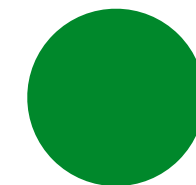
Fully Connected Layer



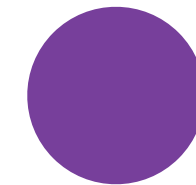
$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{1,i,j} \mathcal{I}(i,j) + b_1 \right)$$



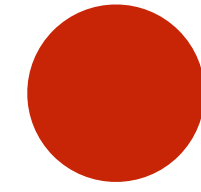
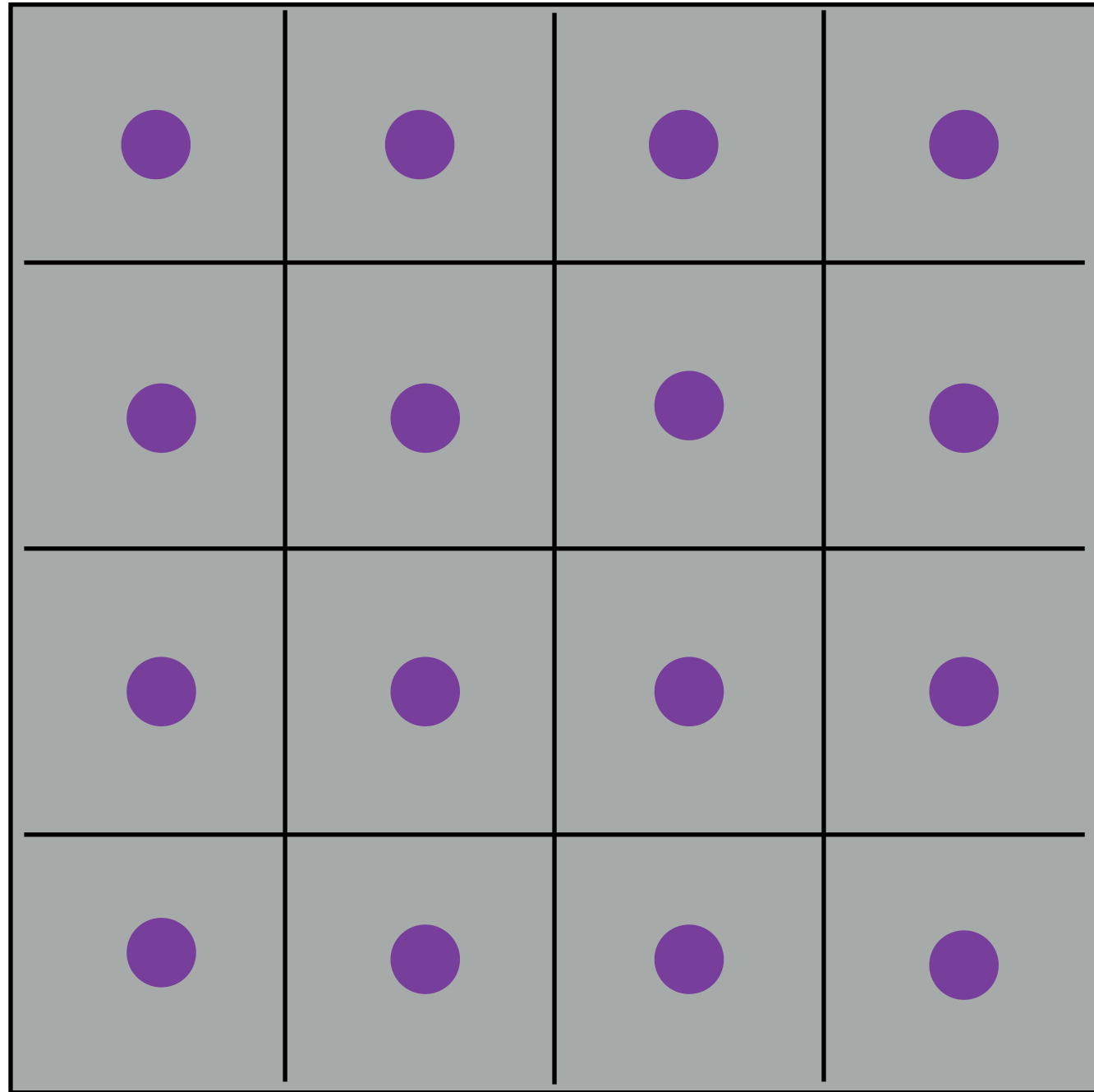
$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{2,i,j} \mathcal{I}(i,j) + b_2 \right)$$



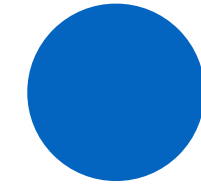
$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{3,i,j} \mathcal{I}(i,j) + b_3 \right)$$



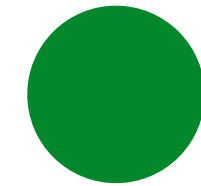
Fully Connected Layer



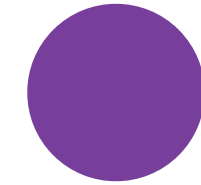
$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{1,i,j} \mathcal{I}(i,j) + b_1 \right)$$



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

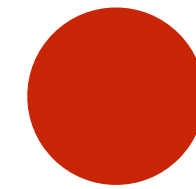
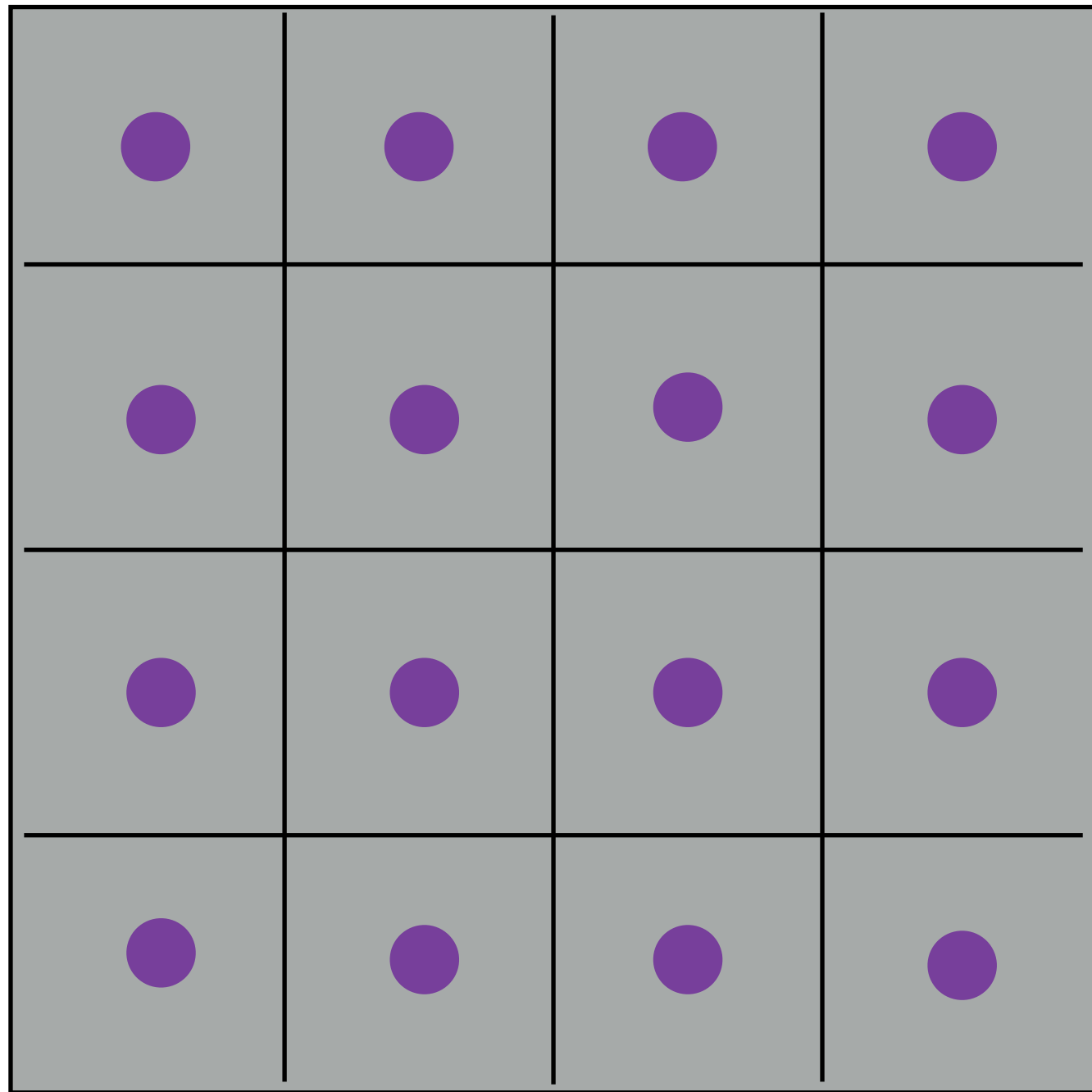


$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{3,i,j} \mathcal{I}(i,j) + b_3 \right)$$



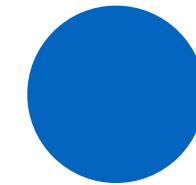
$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{4,i,j} \mathcal{I}(i,j) + b_4 \right)$$

Fully Connected Layer



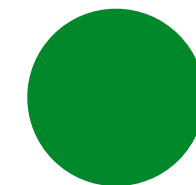
$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{1,i,j} \mathcal{I}(i,j) + b_1 \right)$$

$$4 \times 4 + 1 = 17$$



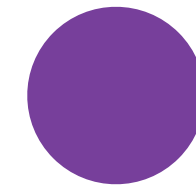
$$\sigma \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{w}_{2,i,j} \mathcal{I}(i,j) + b_2 \right)$$

$$4 \times 4 + 1 = 17$$



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

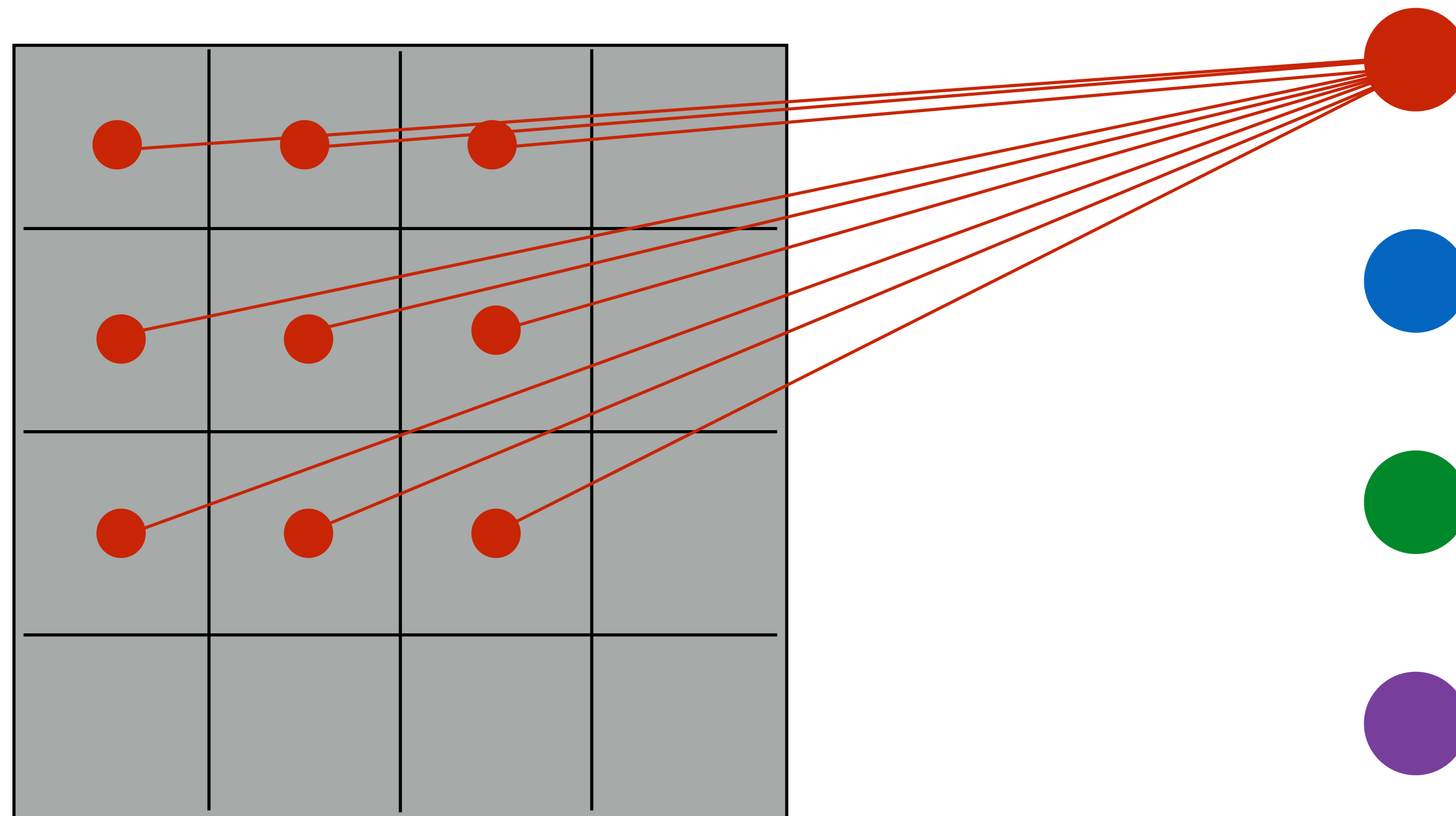
$$4 \times 4 + 1 = 17$$



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

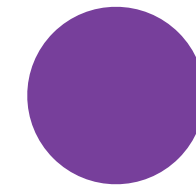
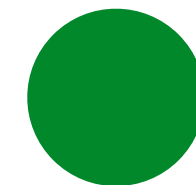
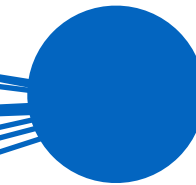
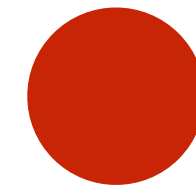
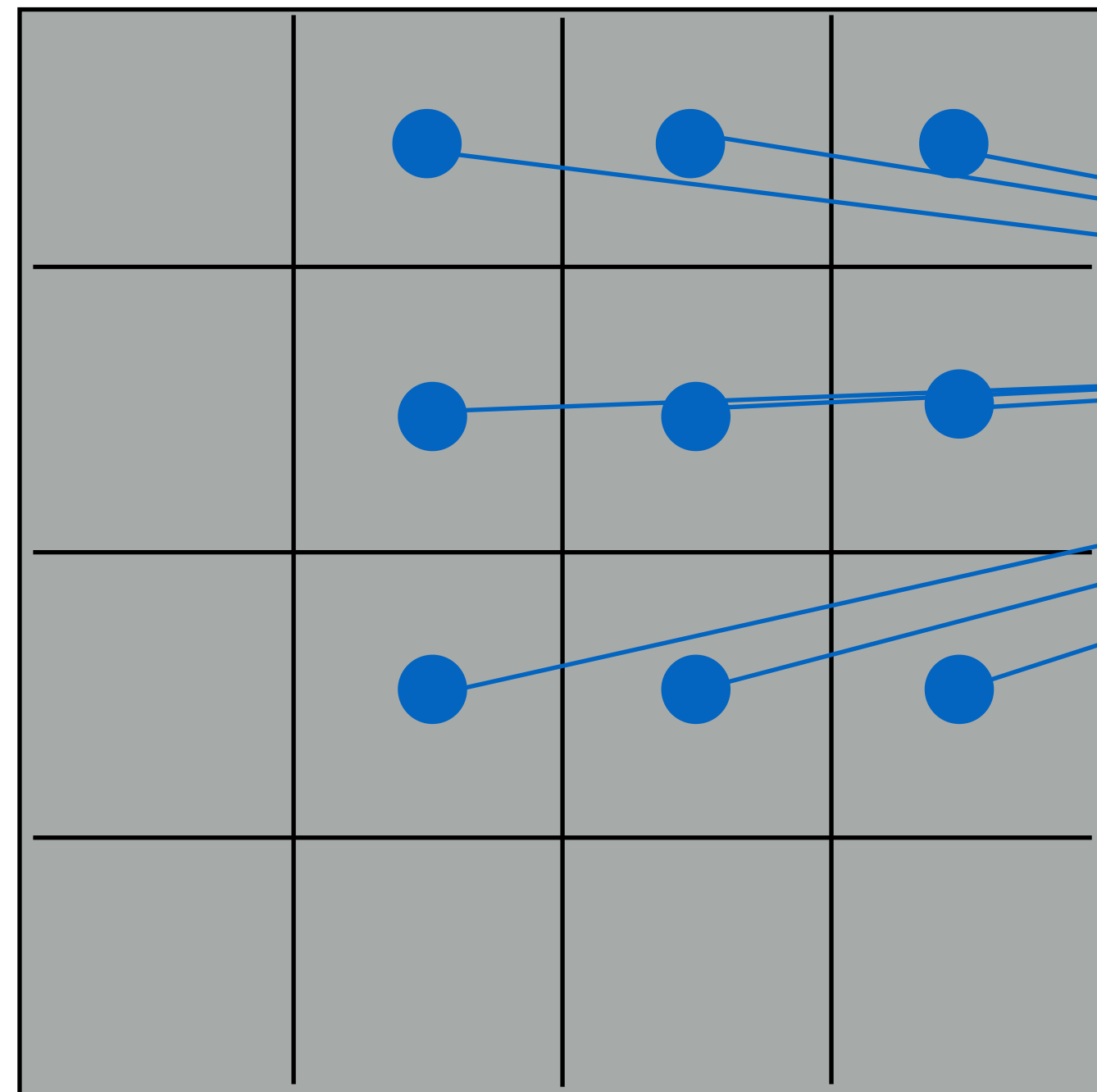
$$4 \times 4 + 1 = 17$$

Locally Connected Layer



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

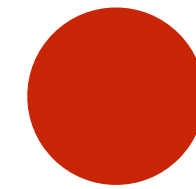
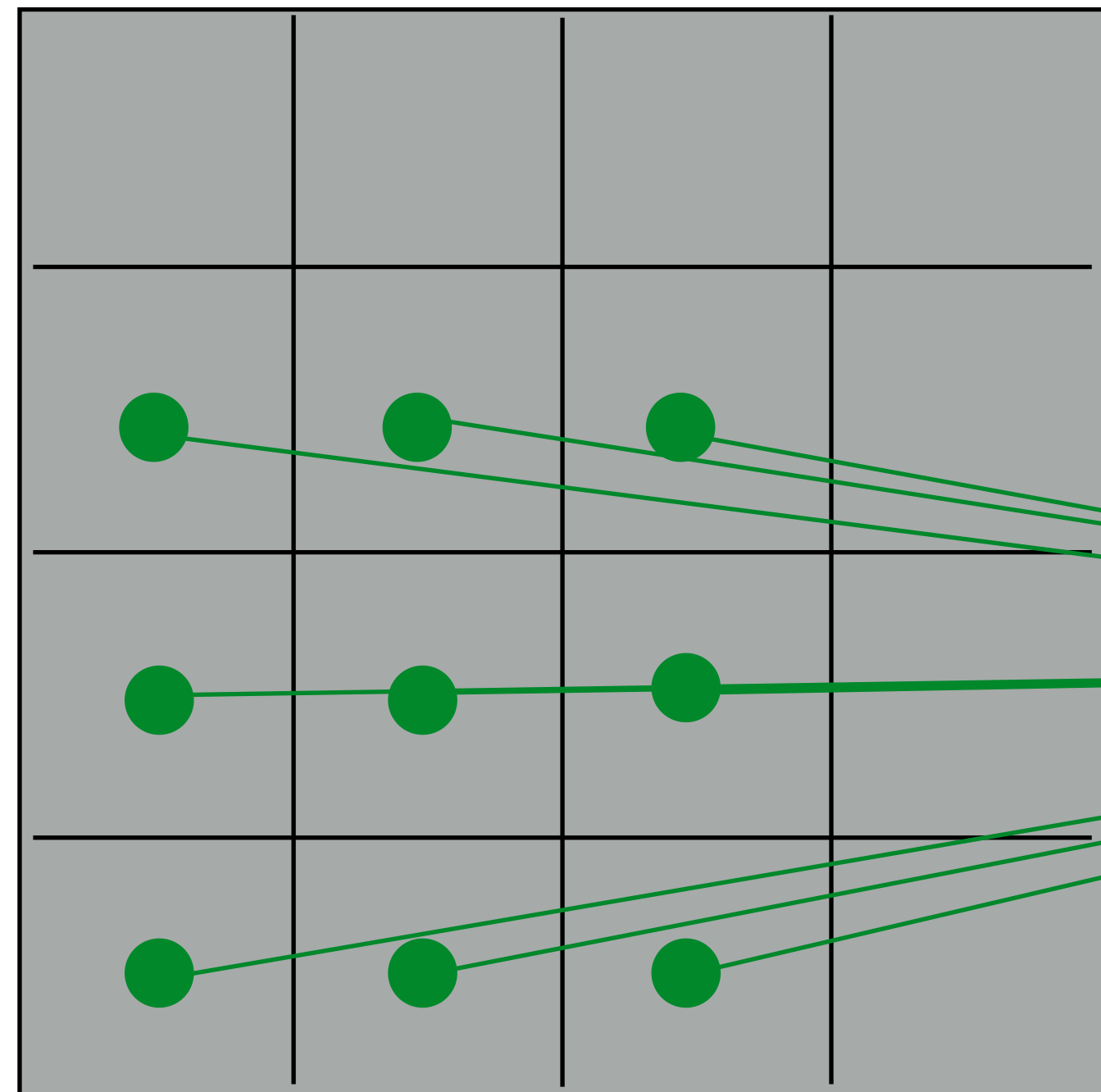
Locally Connected Layer



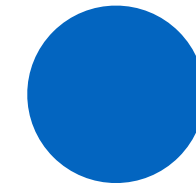
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{1,i,j} \mathcal{I}(i, j) + b_1 \right)$$

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

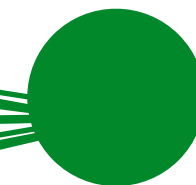
Locally Connected Layer



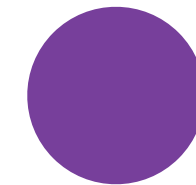
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{1,i,j} \mathcal{I}(i, j) + b_1 \right)$$



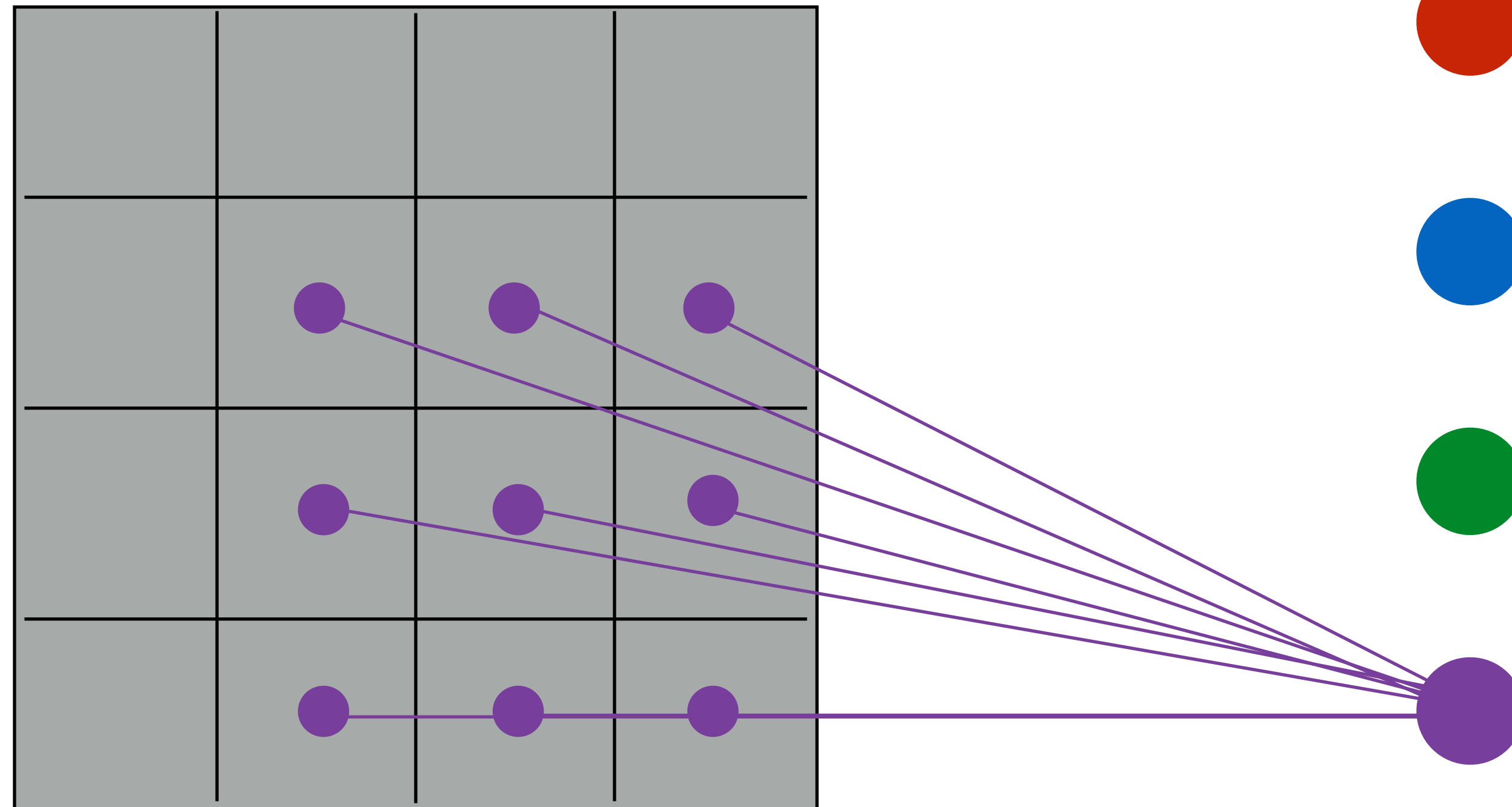
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{2,i,j} \mathcal{I}(i + 1, j) + b_2 \right)$$



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



Locally Connected Layer



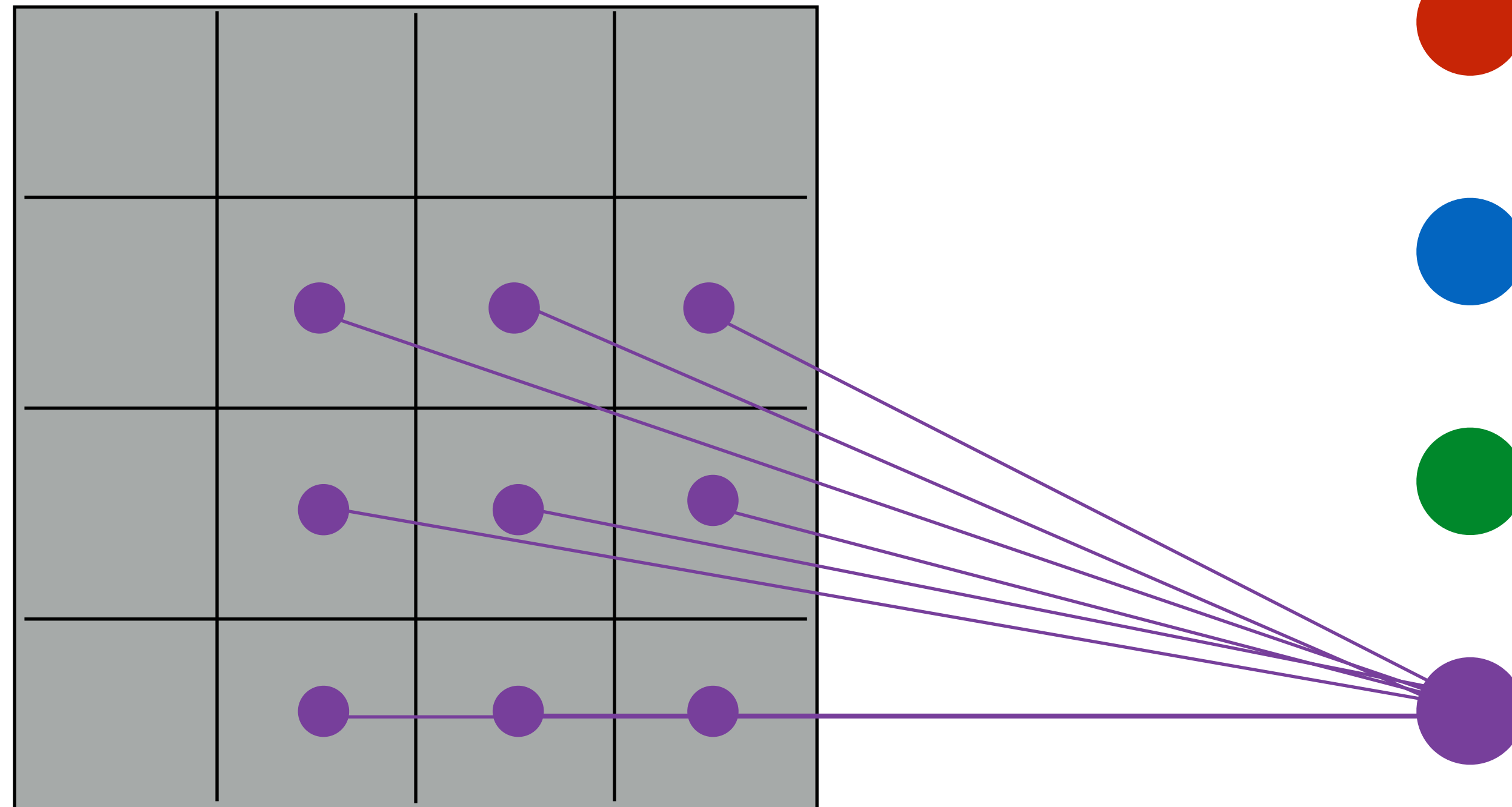
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{1,i,j} \mathcal{I}(i, j) + b_1 \right)$$

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

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

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

Locally Connected Layer



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

$$3 \times 3 + 1 = 10$$

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

$$3 \times 3 + 1 = 10$$

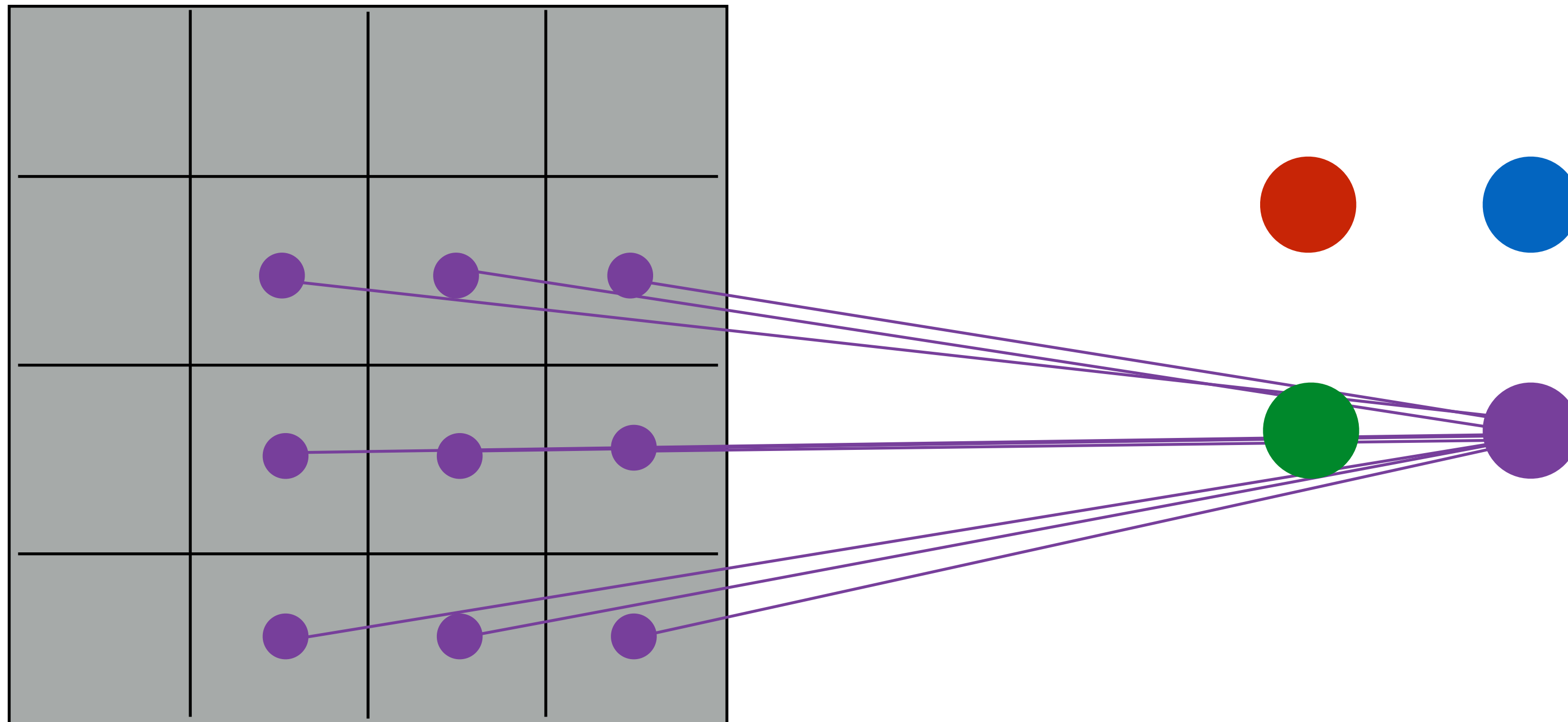
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{3,i,j} \mathcal{I}(i, j + 1) + b_3 \right)$$

$$3 \times 3 + 1 = 10$$

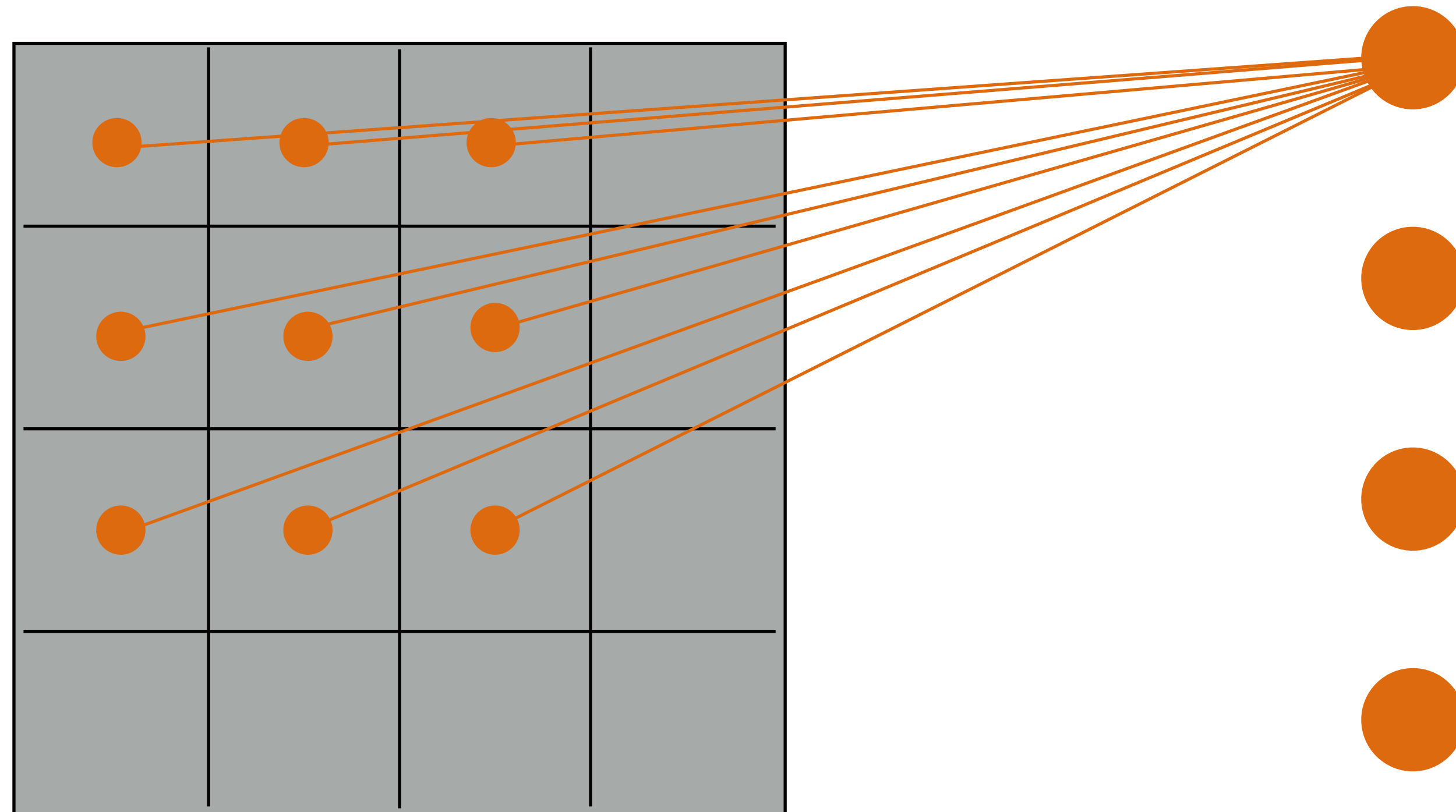
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{4,i,j} \mathcal{I}(i + 1, j + 1) + b_4 \right)$$

$$3 \times 3 + 1 = 10$$

Locally Connected Layer

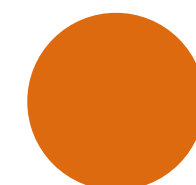
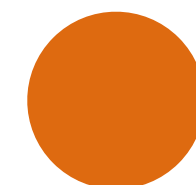
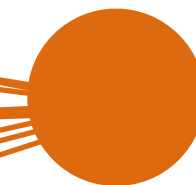
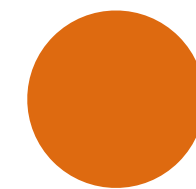
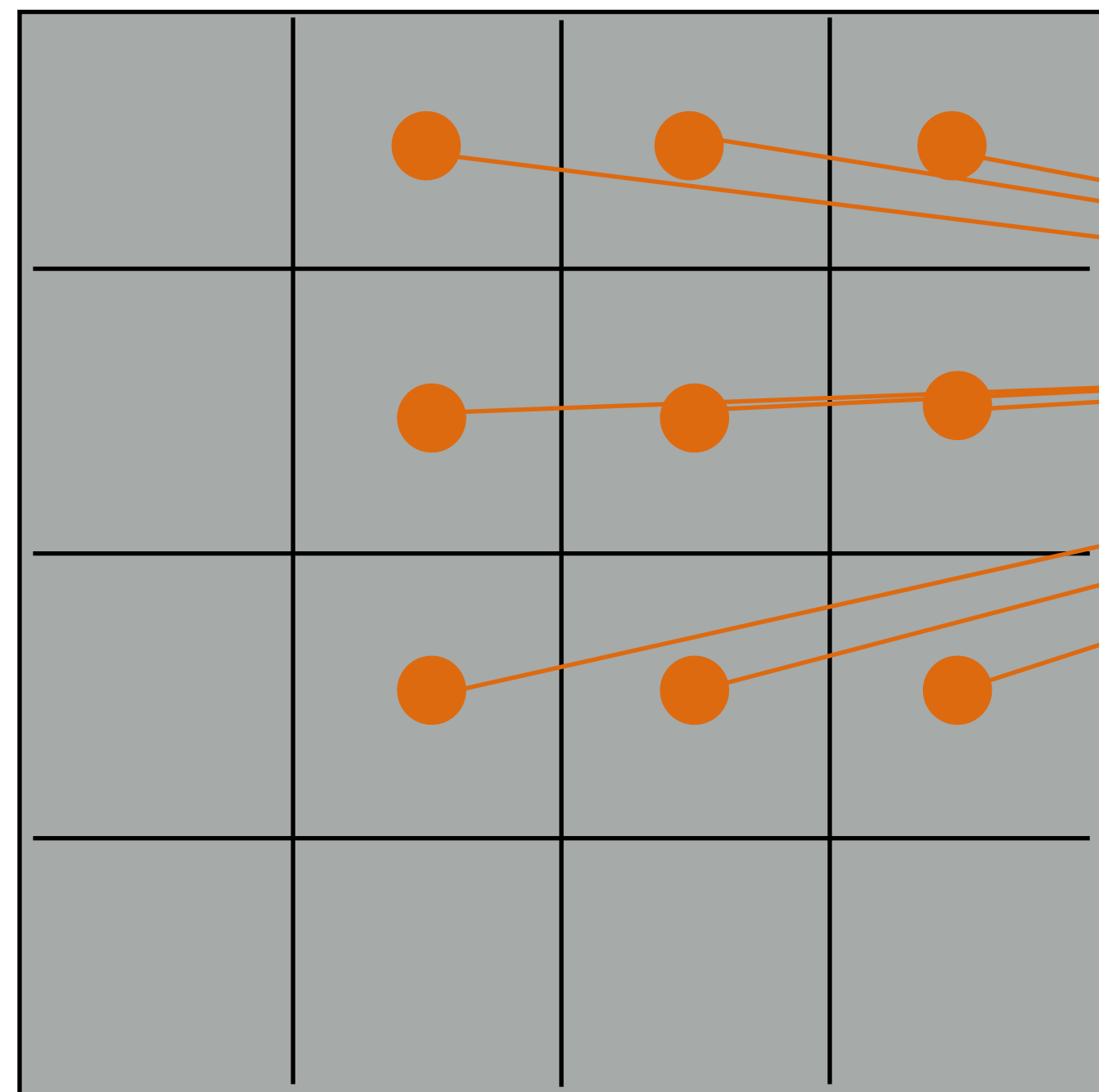


Convolutional Layer



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

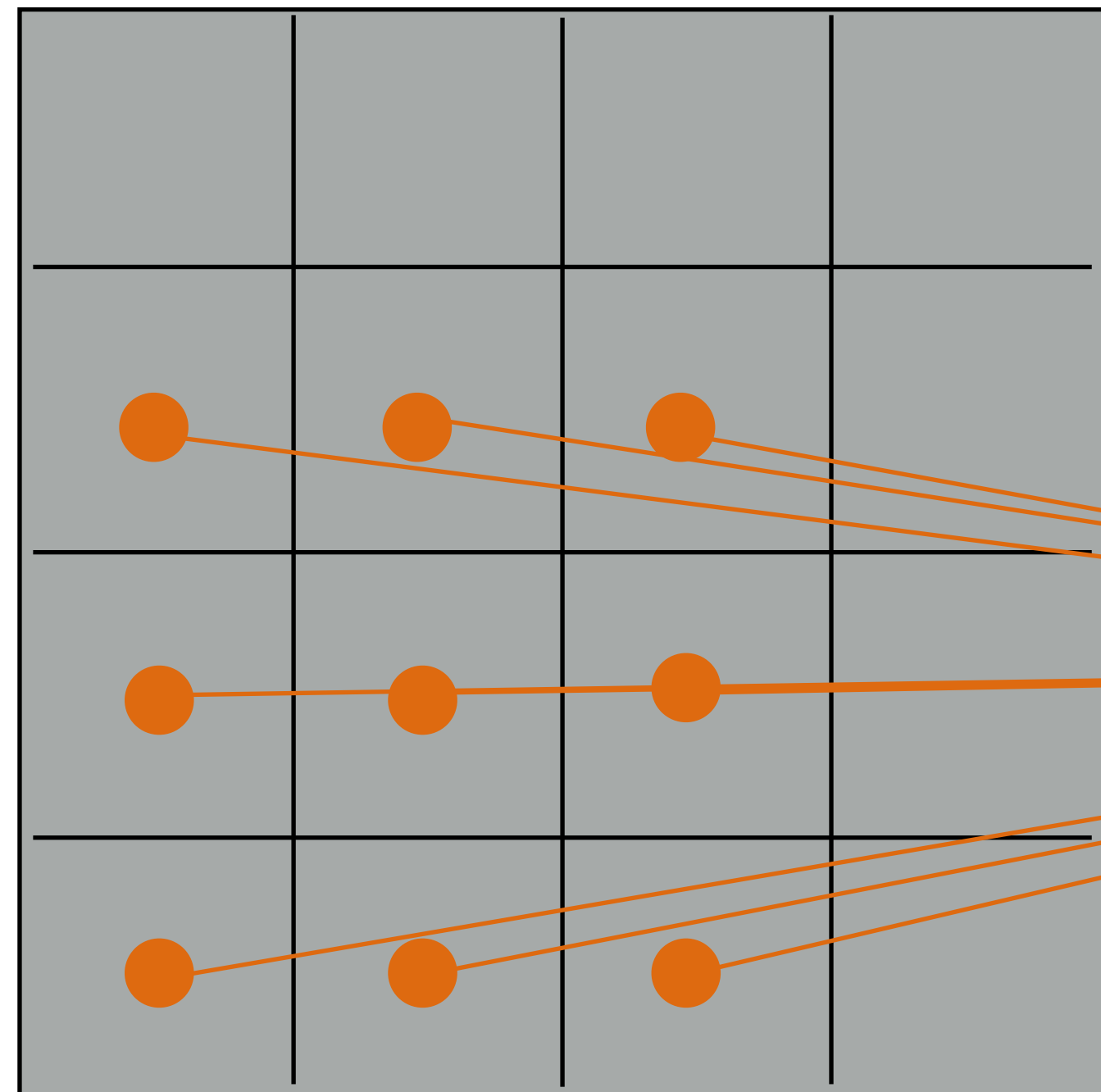
Convolutional Layer



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

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

Convolutional Layer

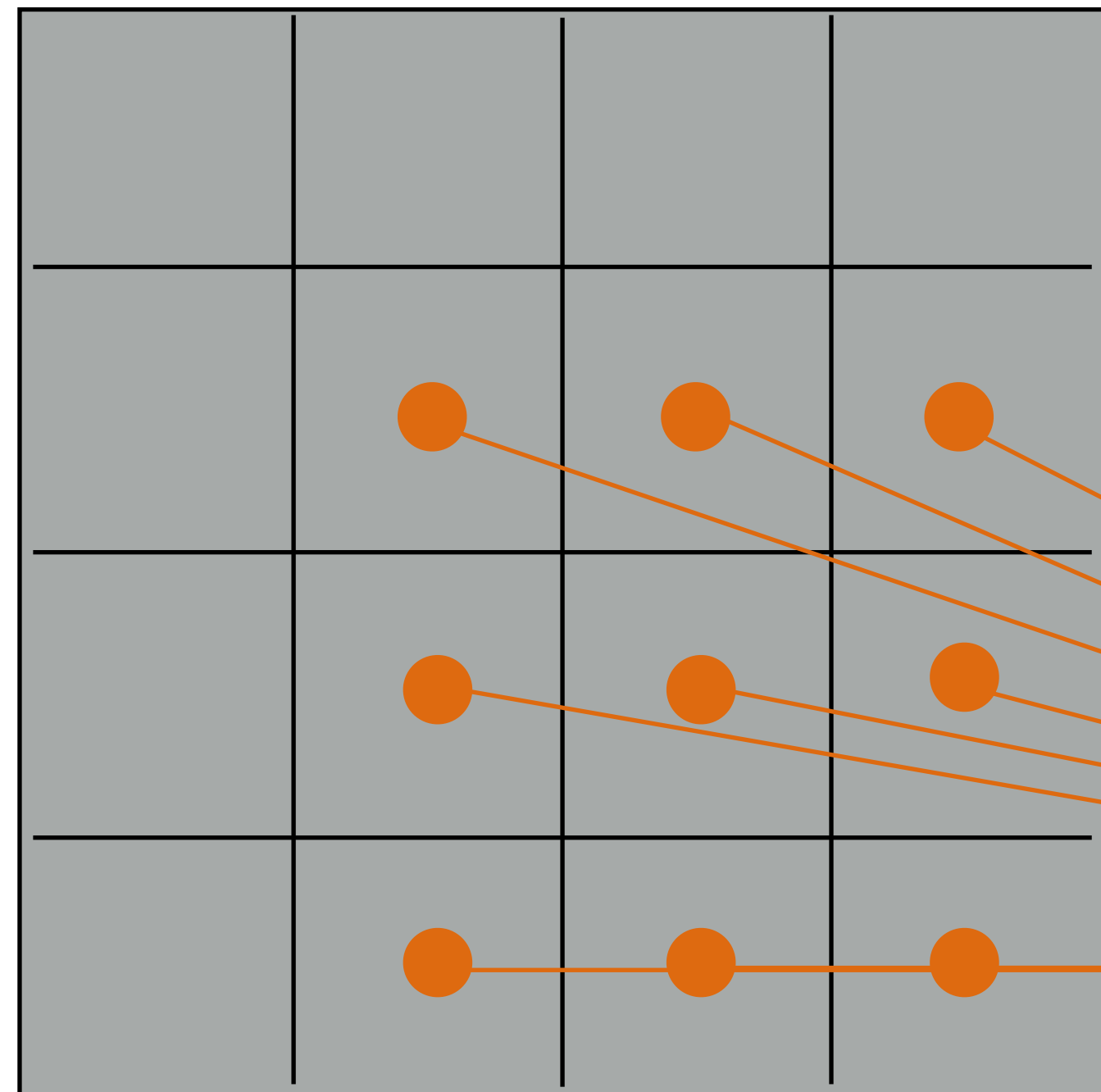


$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i, j) + b \right)$$

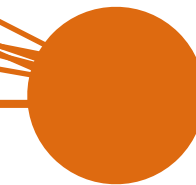
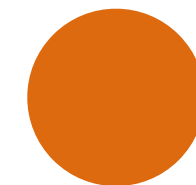
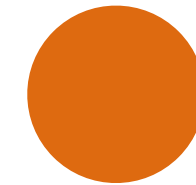
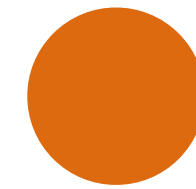
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i + 1, j) + b \right)$$

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

Convolutional Layer



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

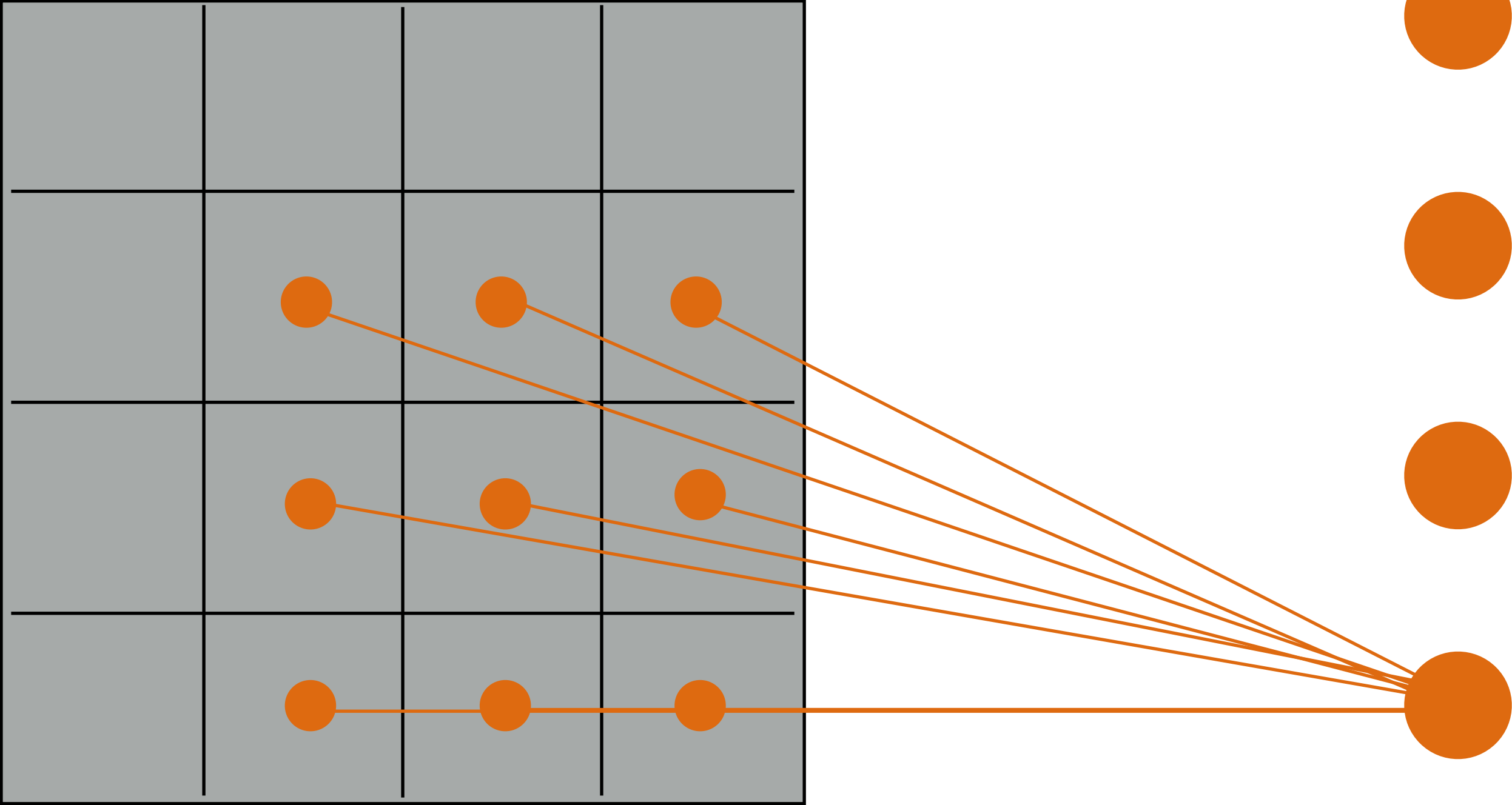


$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i + 1, j) + b \right)$$

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

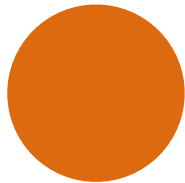
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i + 1, j + 1) + b \right)$$

Convolutional Layer



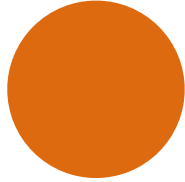
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i, j) + b \right)$$

3 x 3 + 1 = 10



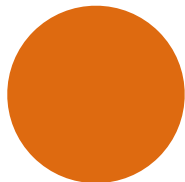
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i + 1, j) + b \right)$$

0 x 0 + 0 = 0



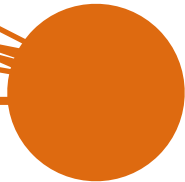
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i, j + 1) + b \right)$$

0 x 0 + 0 = 0

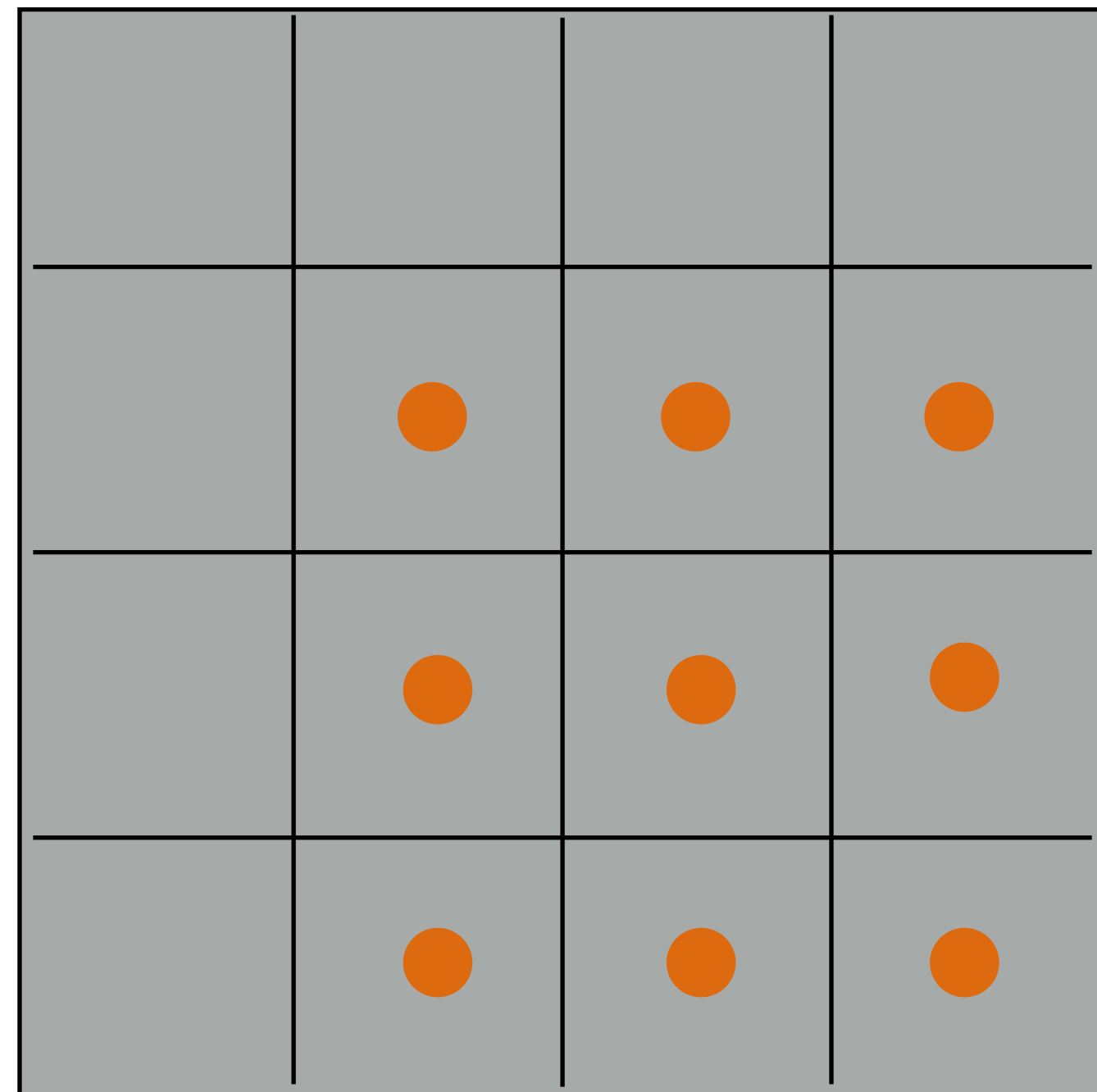


$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i + 1, j + 1) + b \right)$$

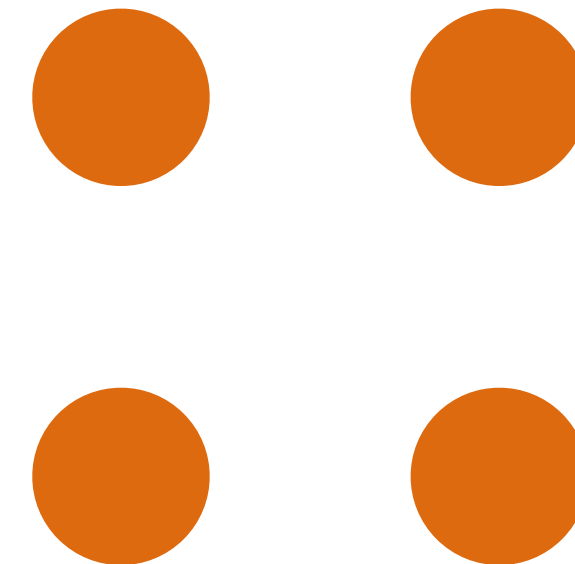
0 x 0 + 0 = 0



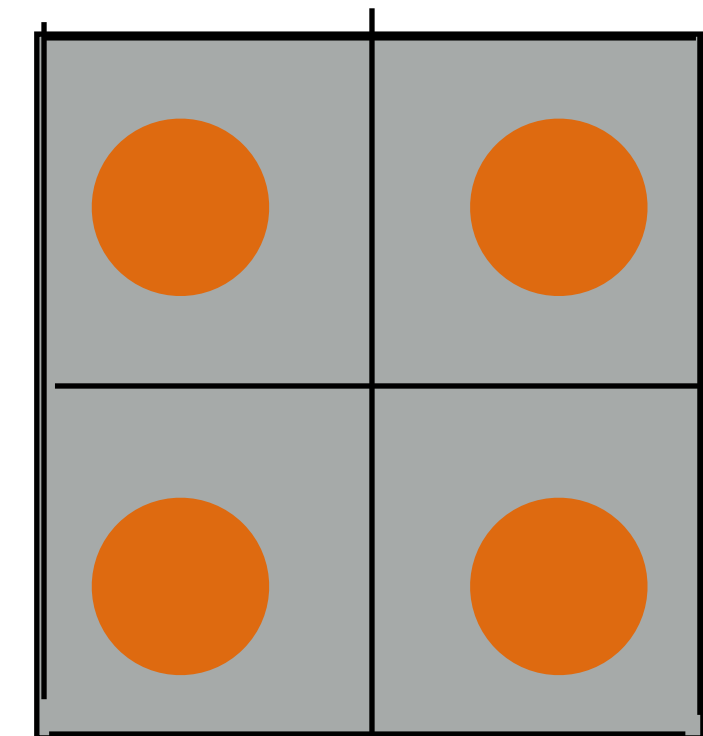
Convolutional Layer: Interpretation #1



Multiple neurons that share weights



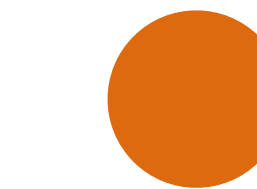
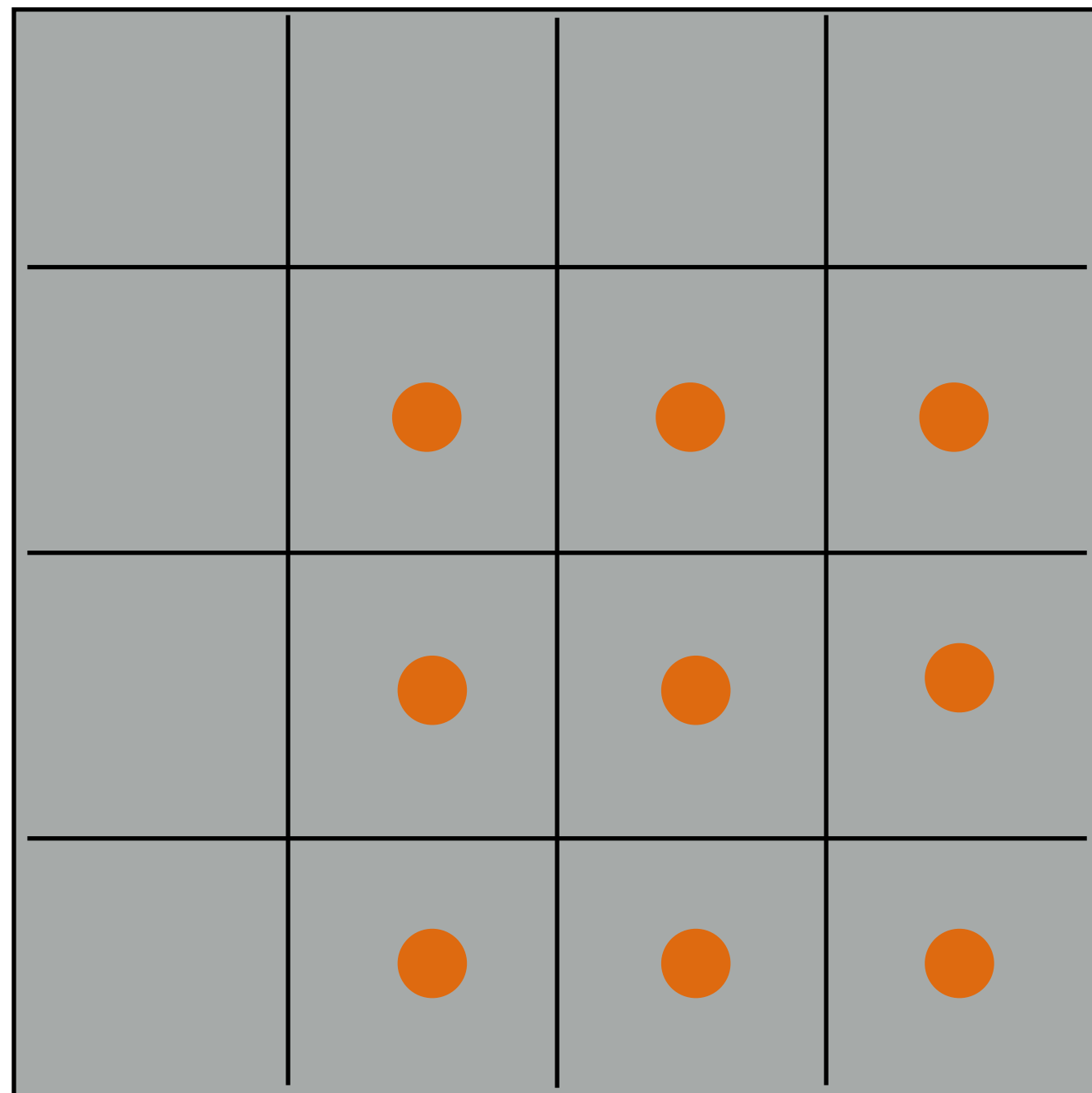
neurons



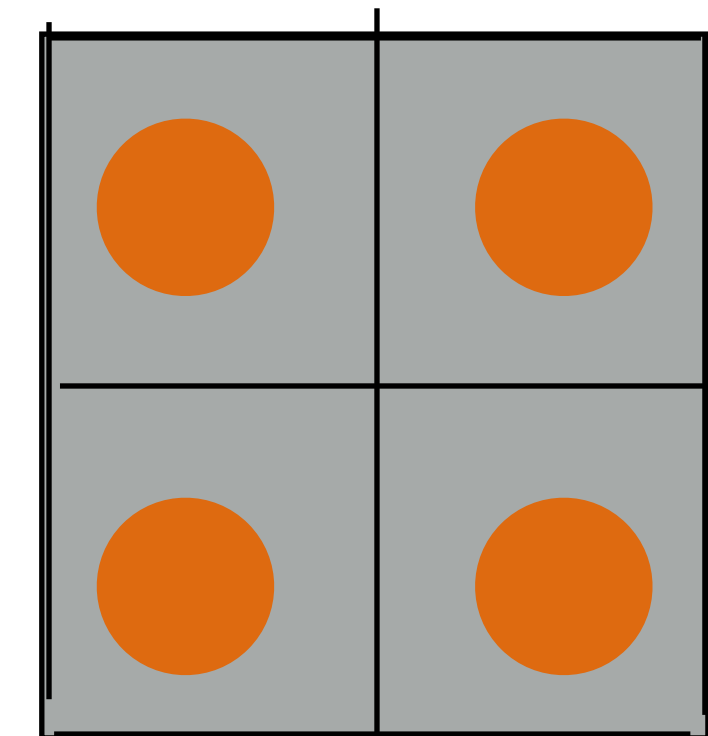
output

Convolutional Layer: Interpretation #2

One neuron applied as convolution (by shifting)



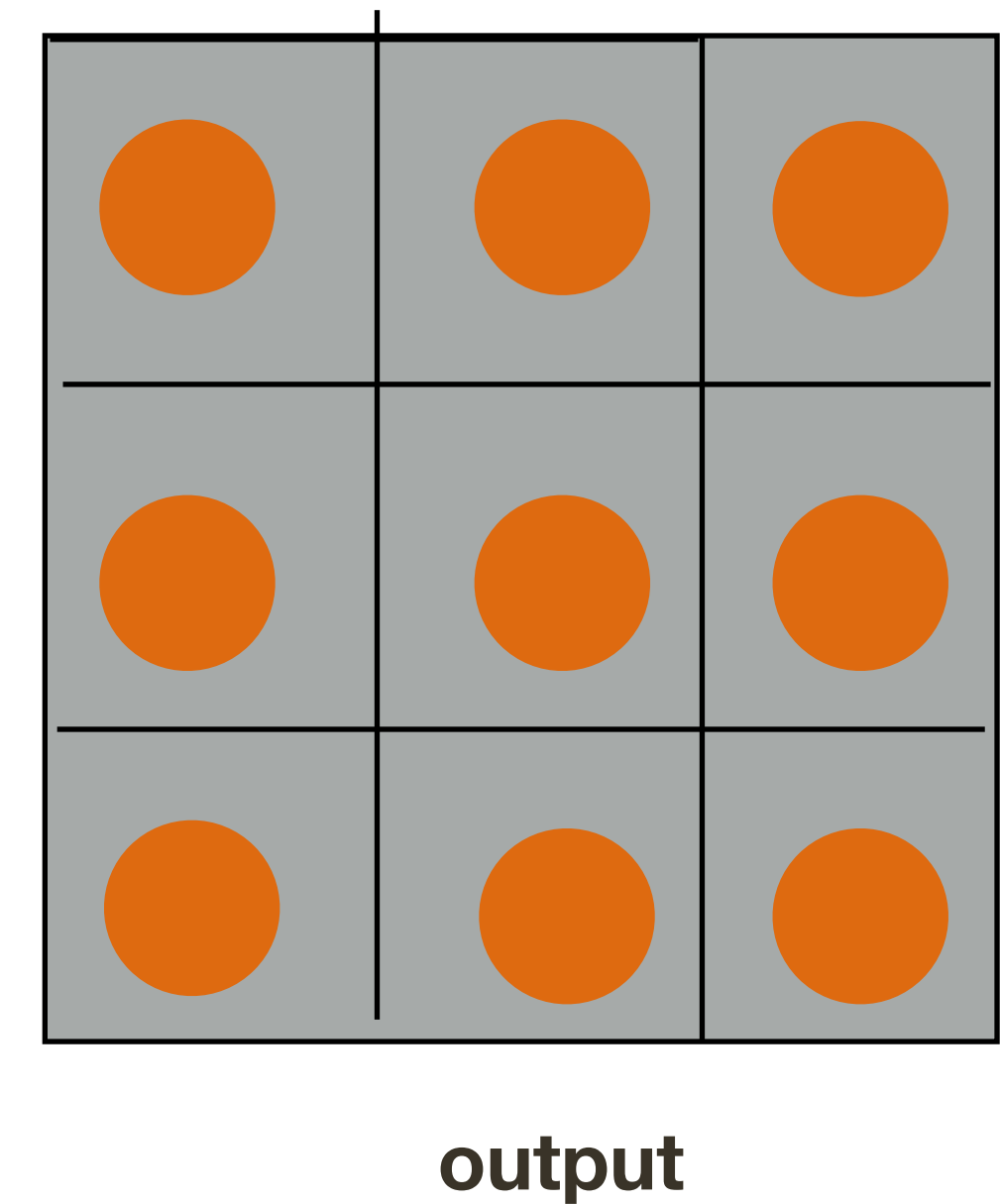
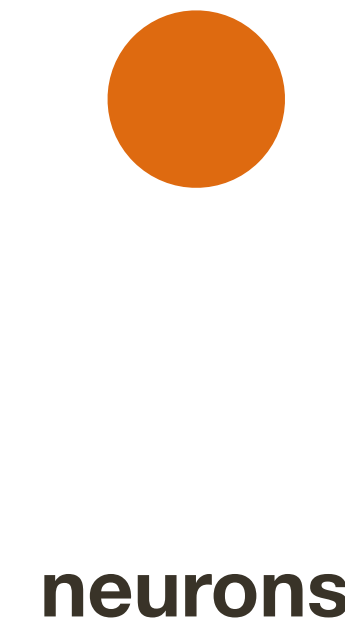
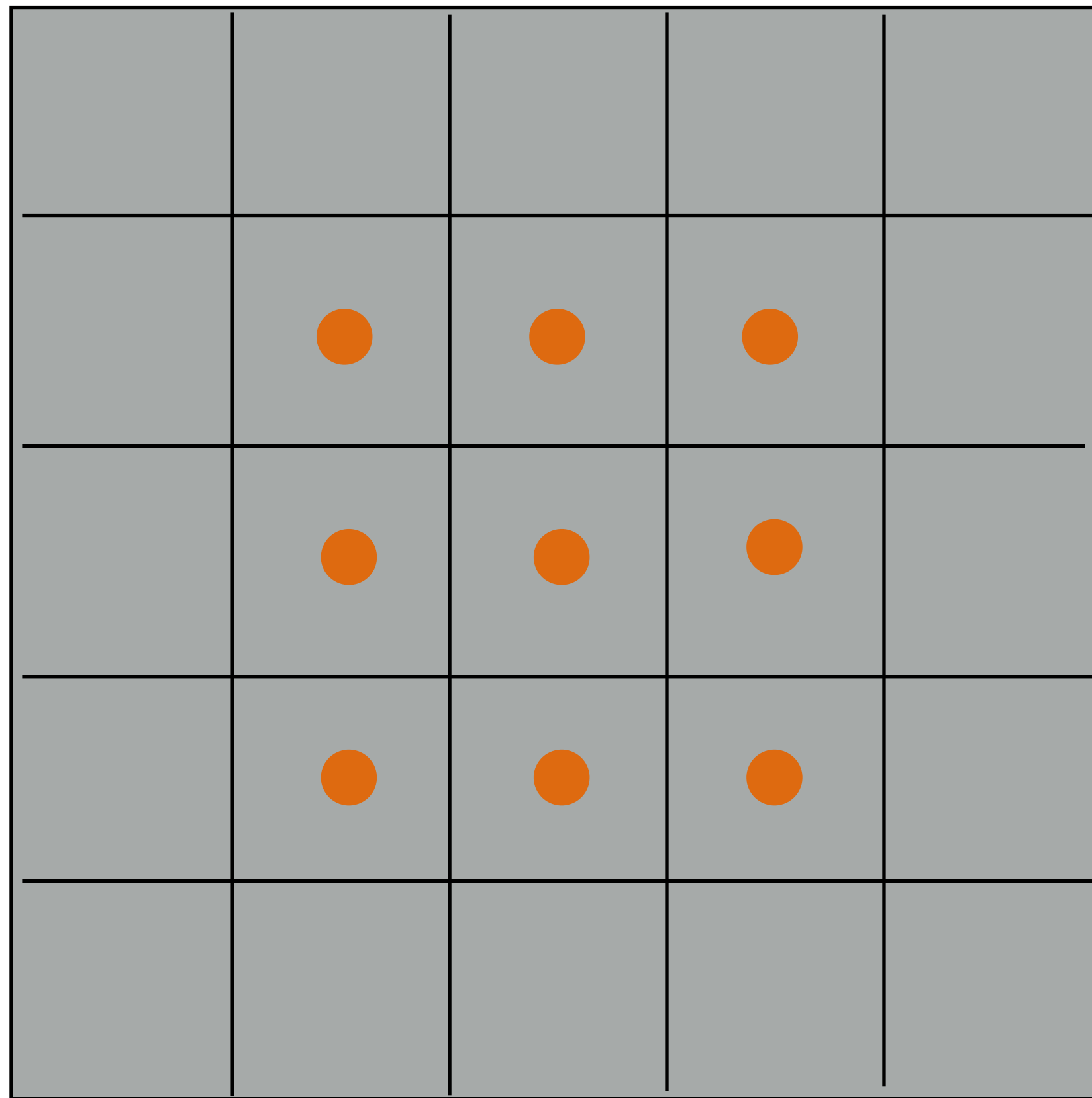
neurons



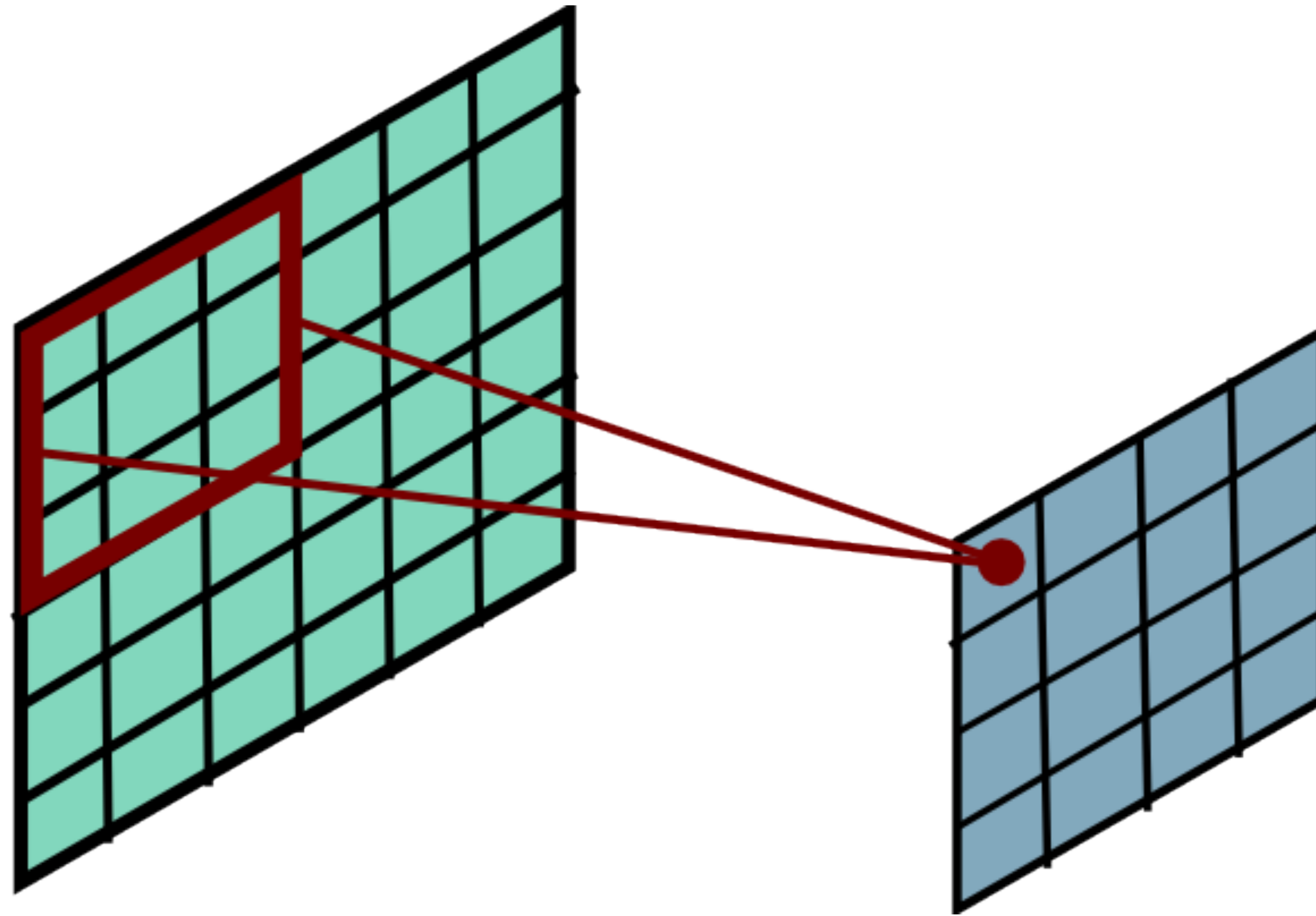
output

Convolutional Layer: Interpretation #2

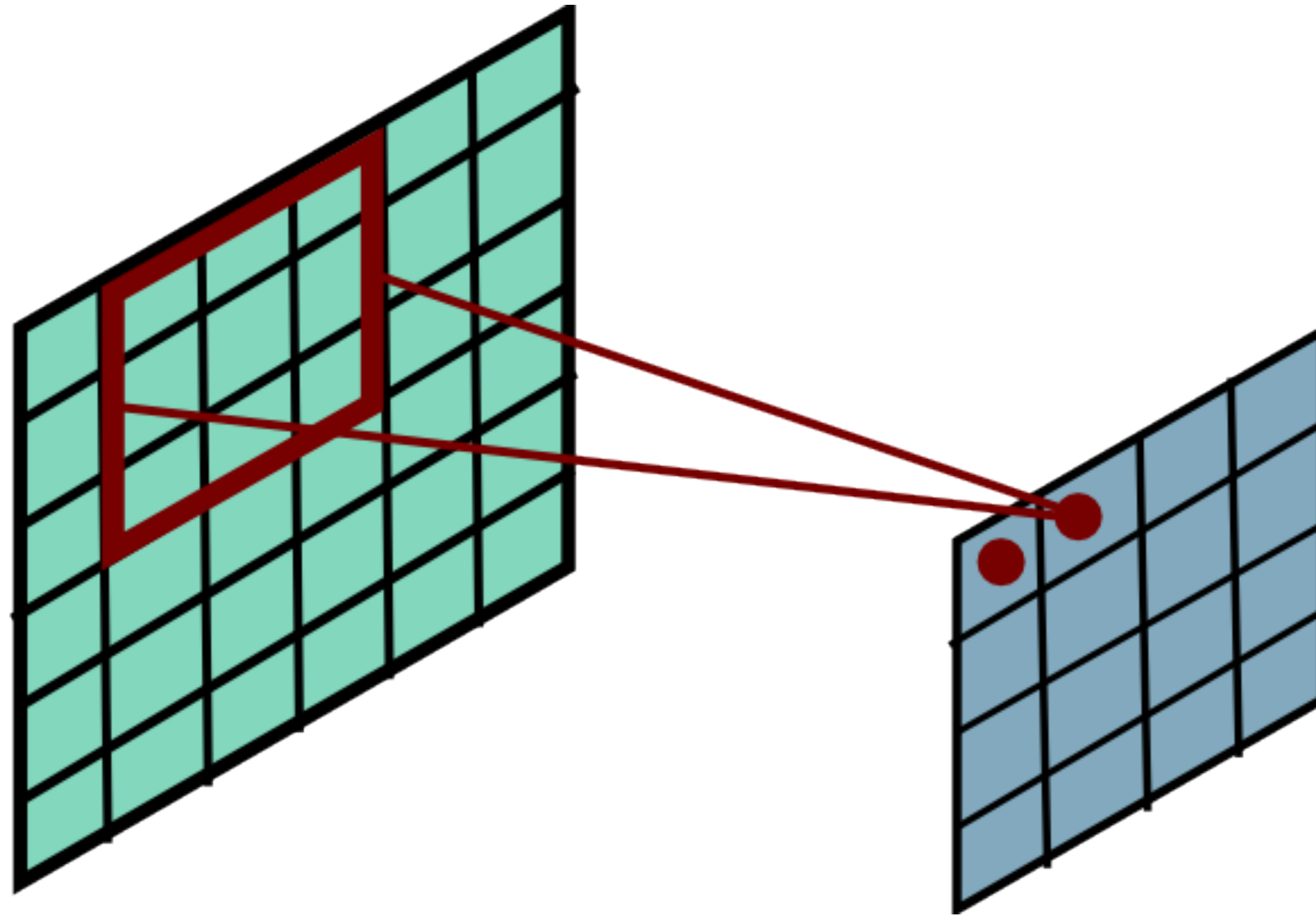
One neuron applied as convolution (by shifting)



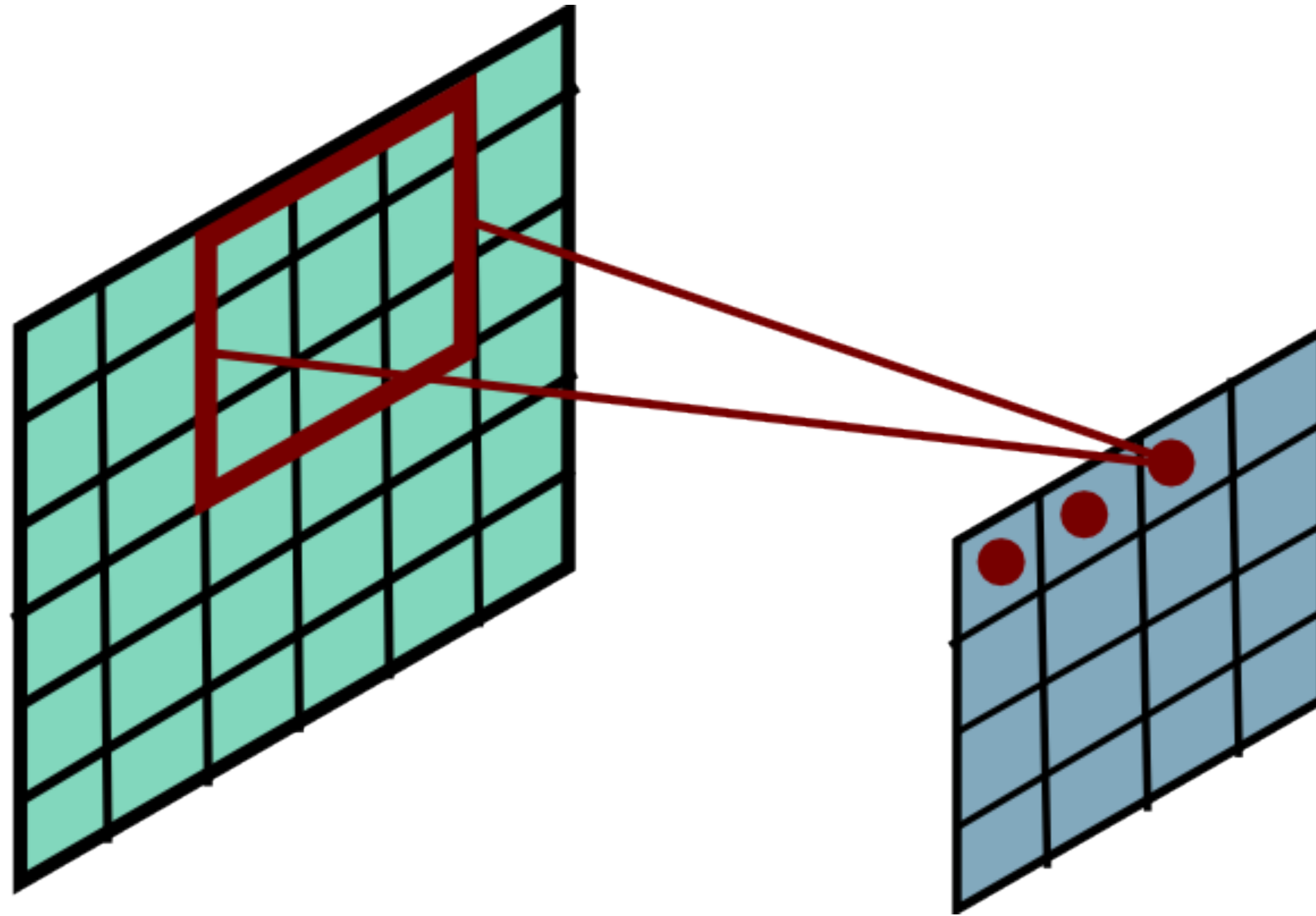
Convolutional Layer



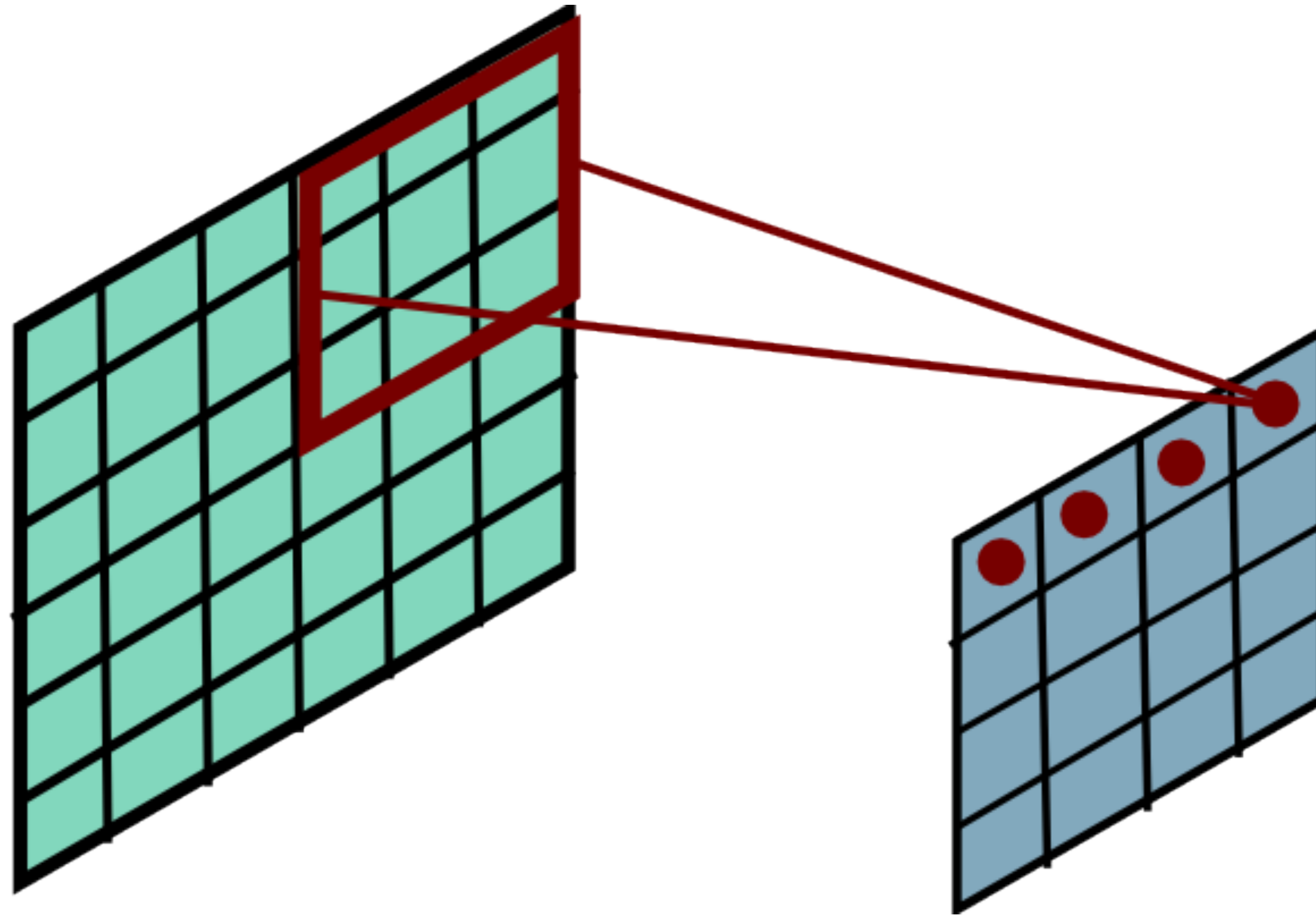
Convolutional Layer



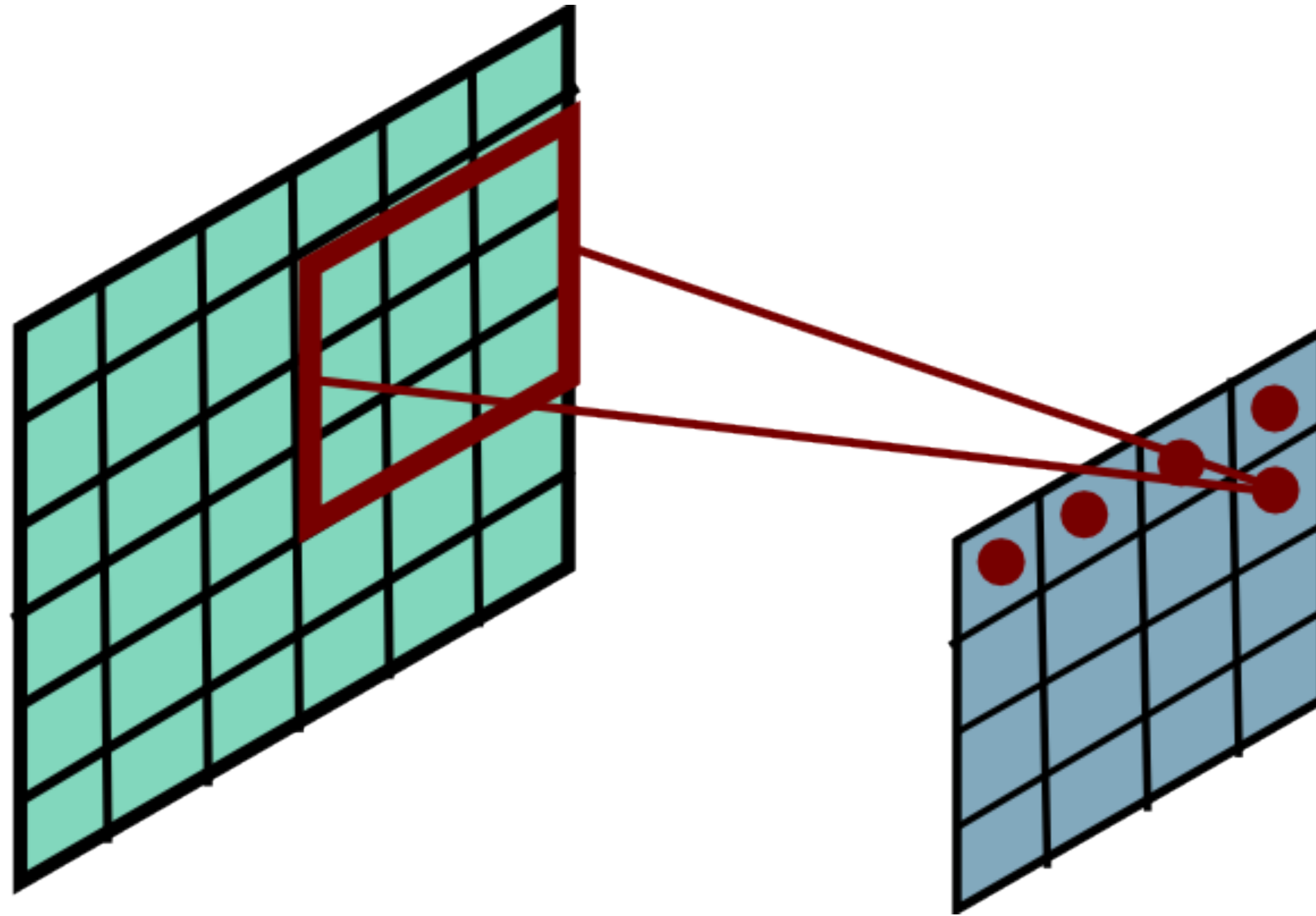
Convolutional Layer



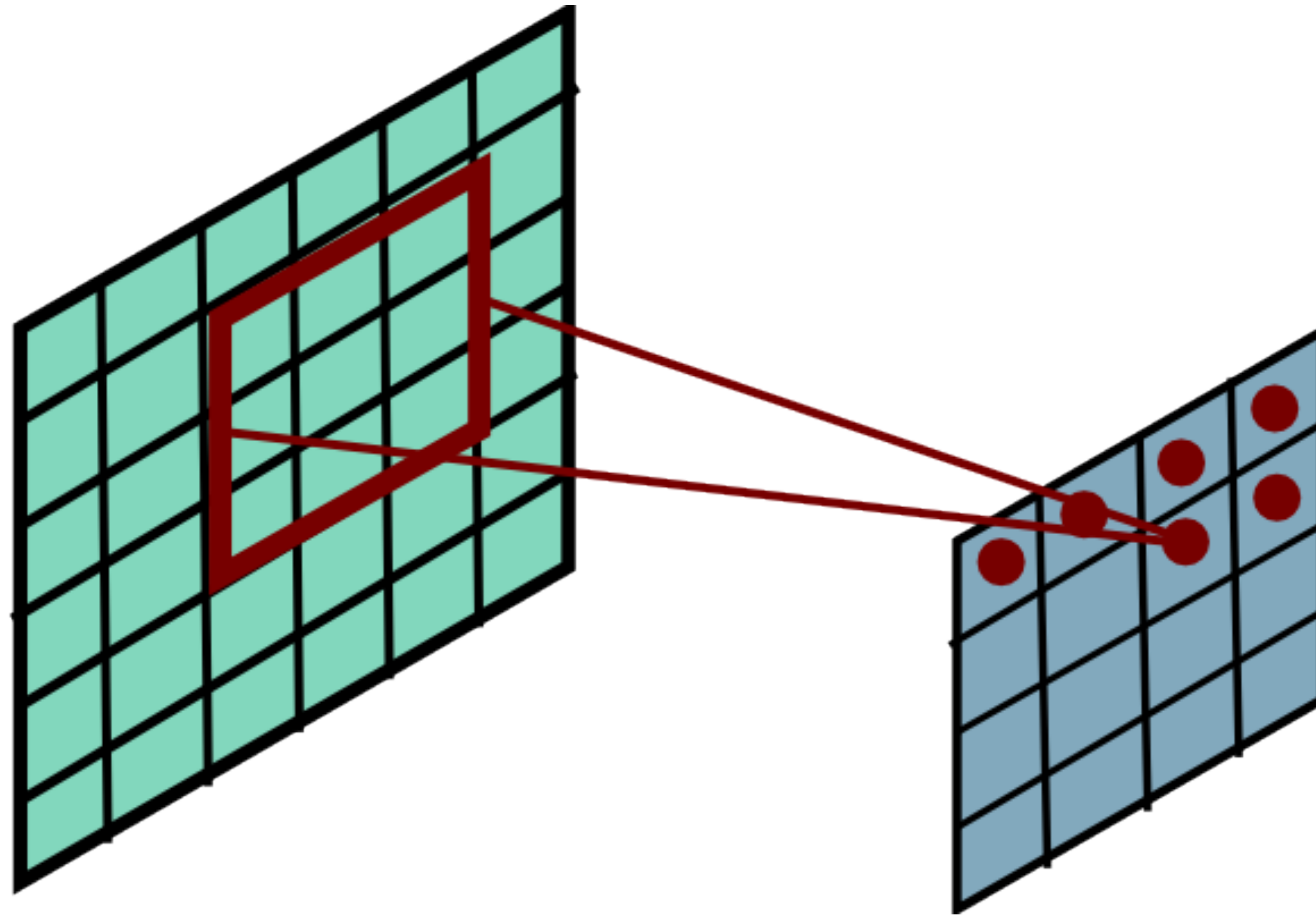
Convolutional Layer



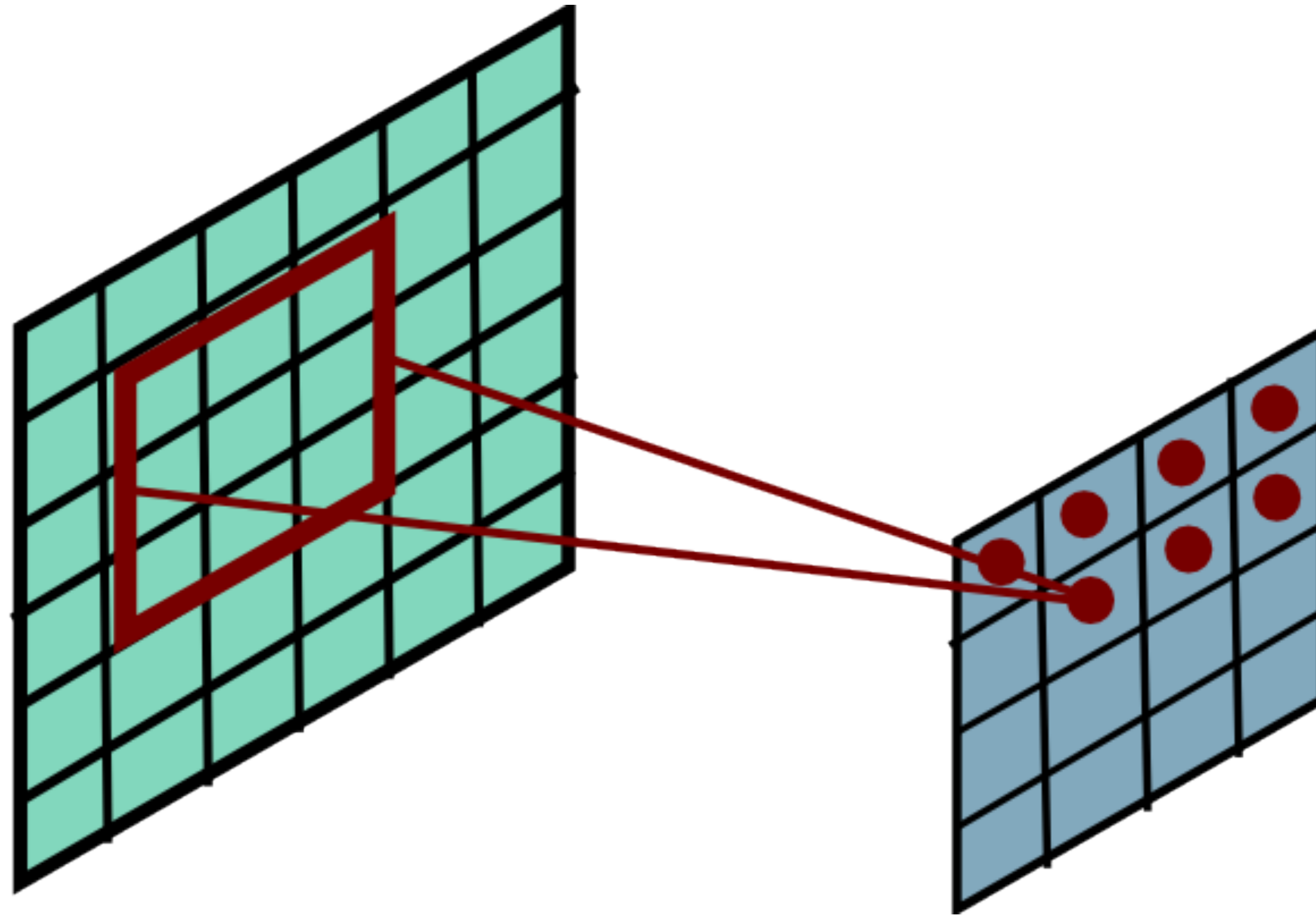
Convolutional Layer



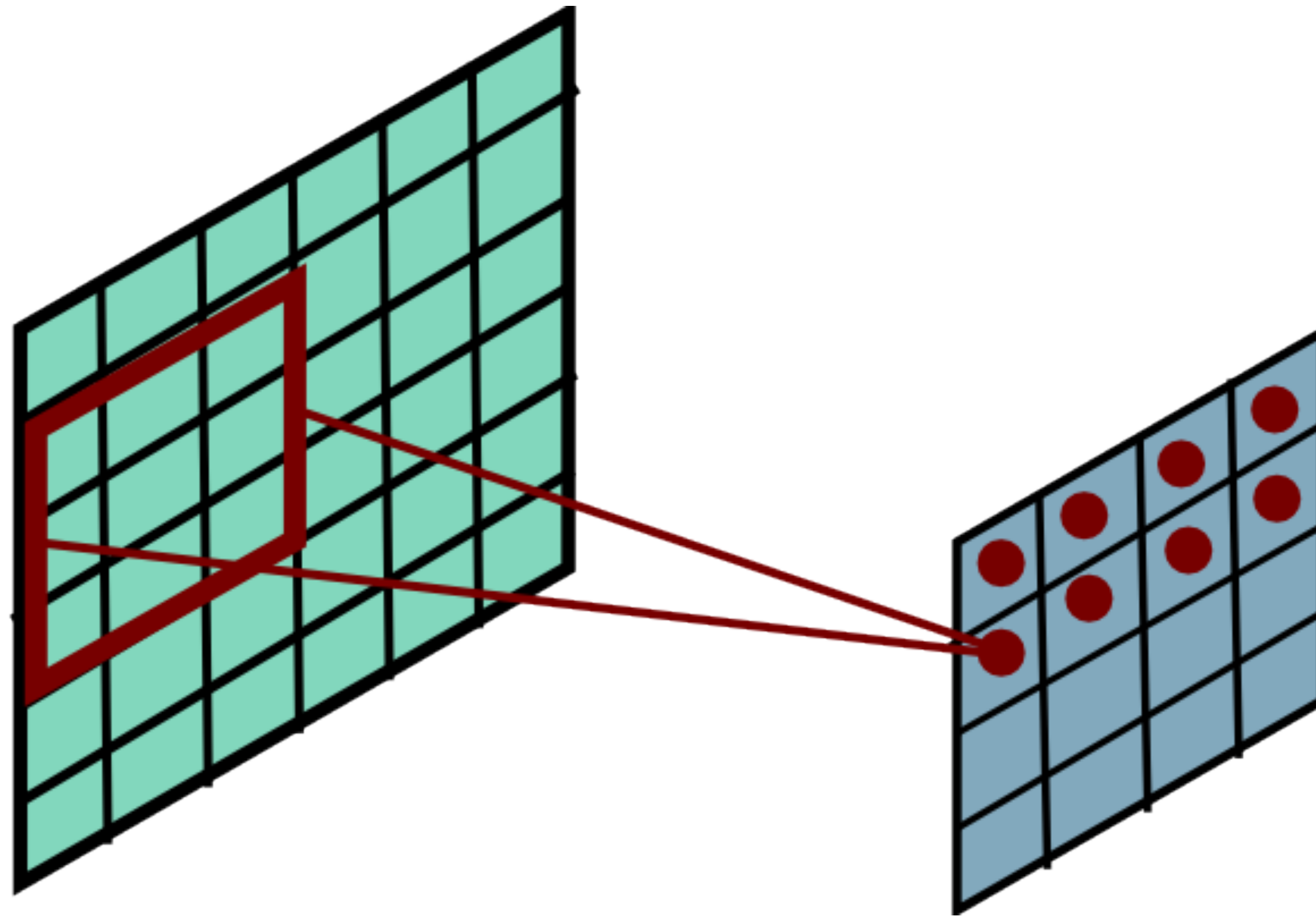
Convolutional Layer



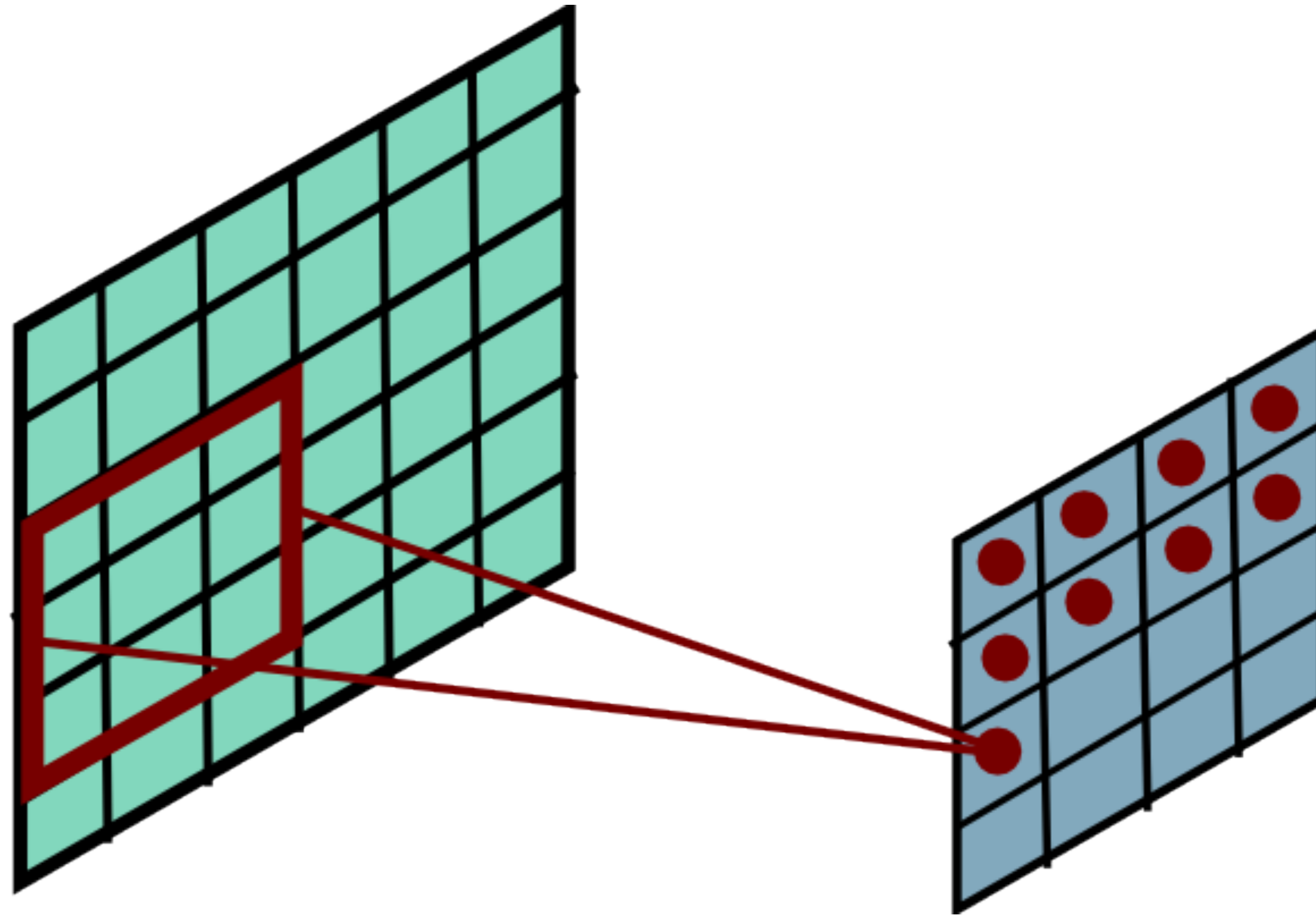
Convolutional Layer



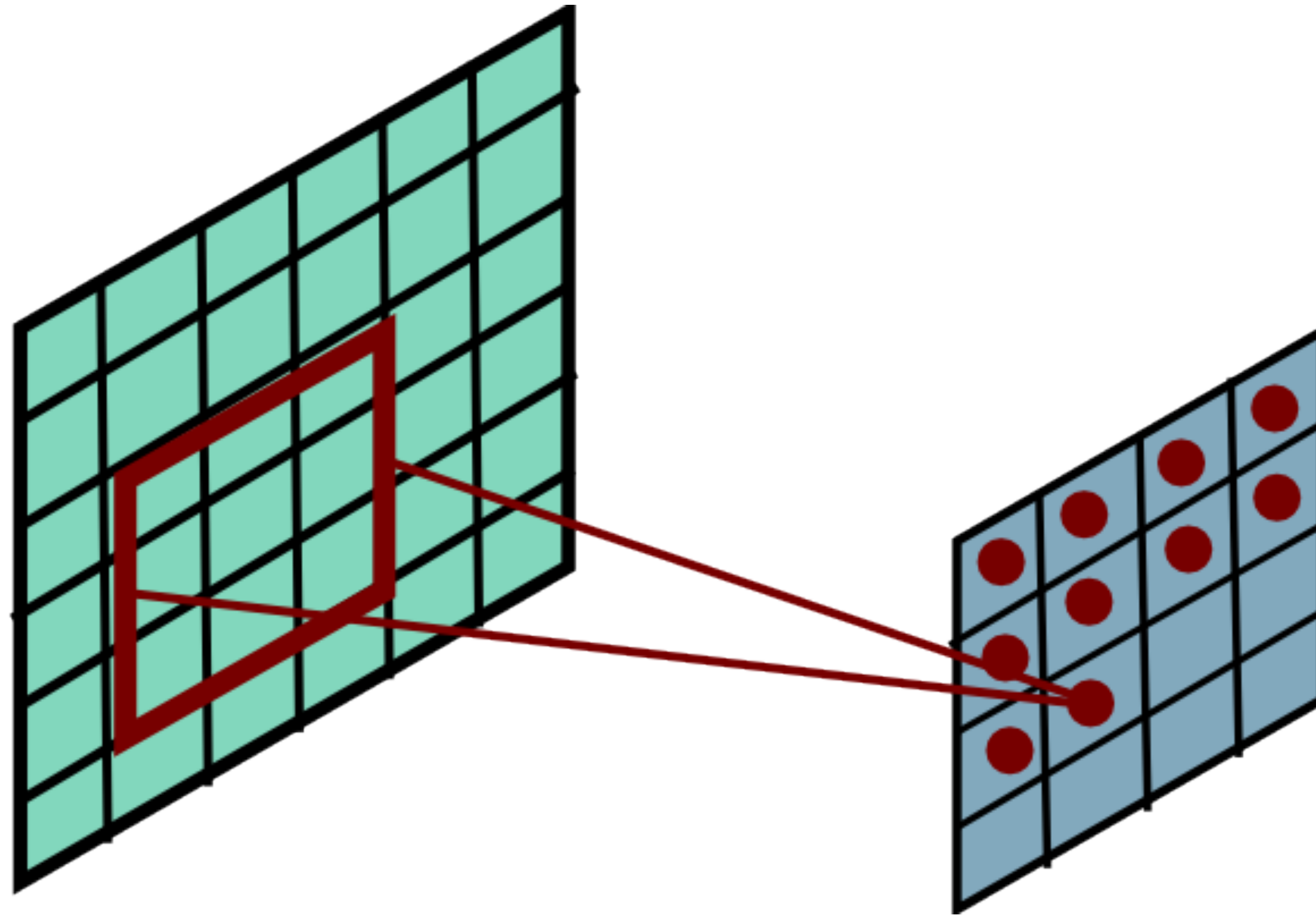
Convolutional Layer



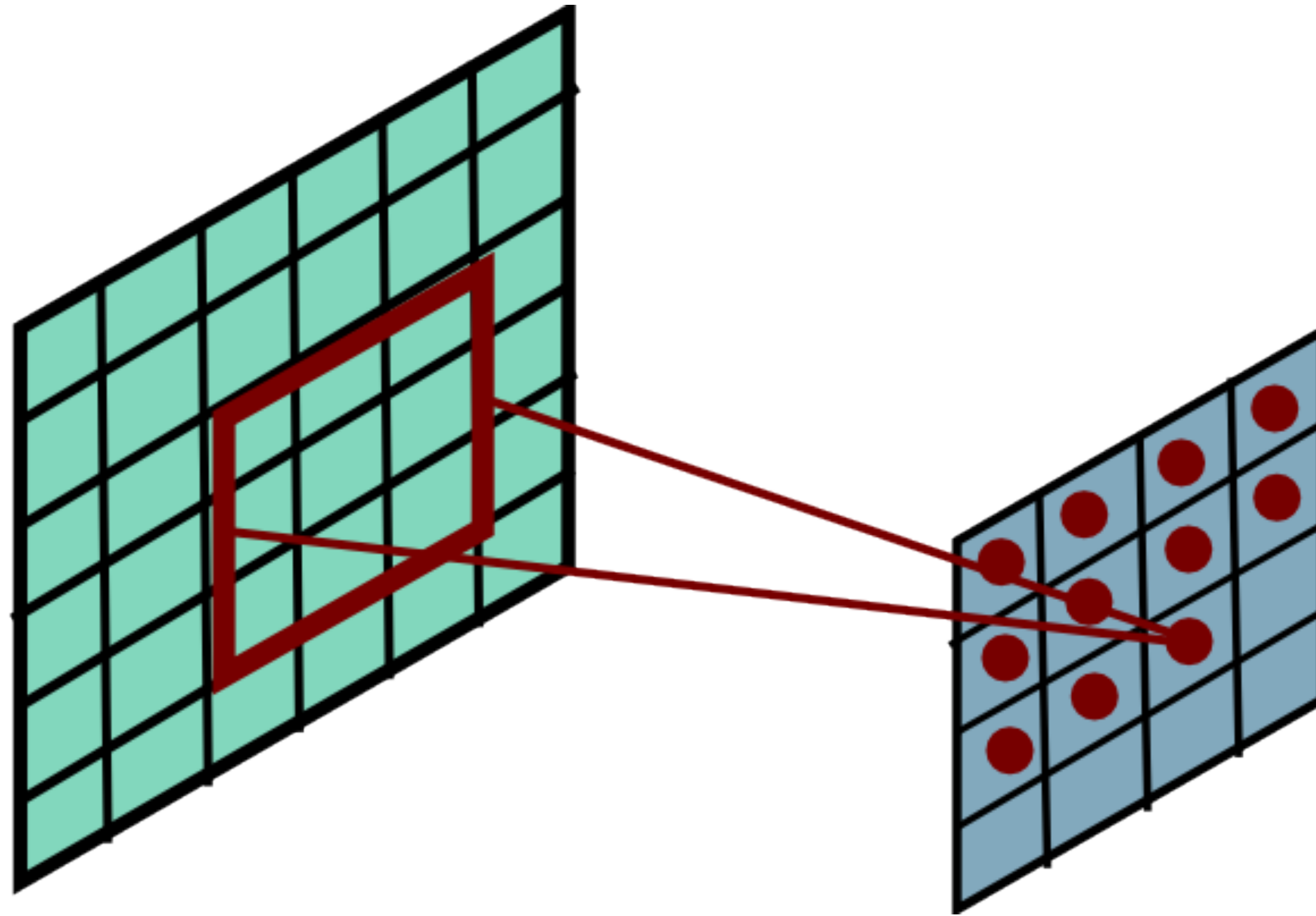
Convolutional Layer



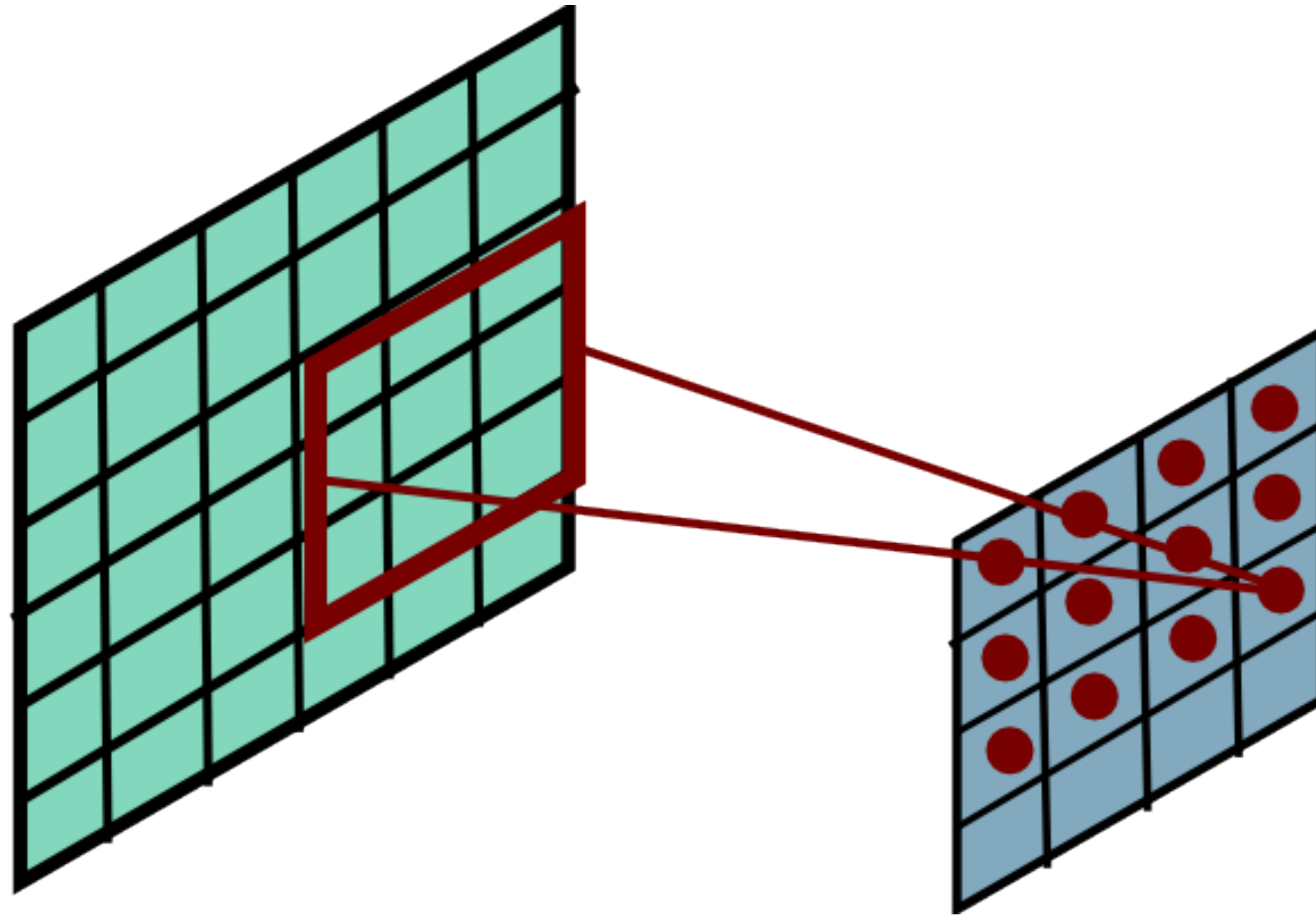
Convolutional Layer



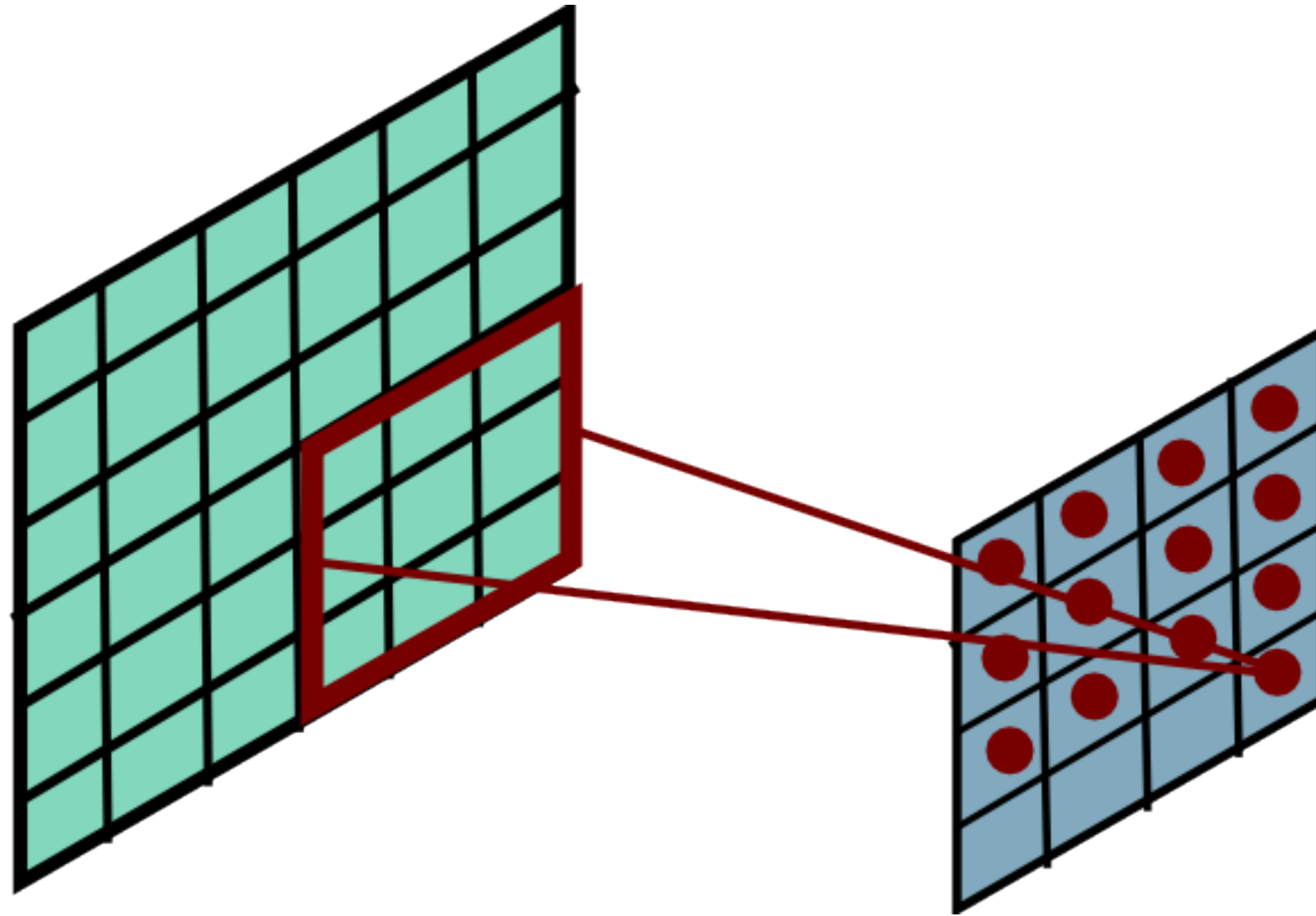
Convolutional Layer



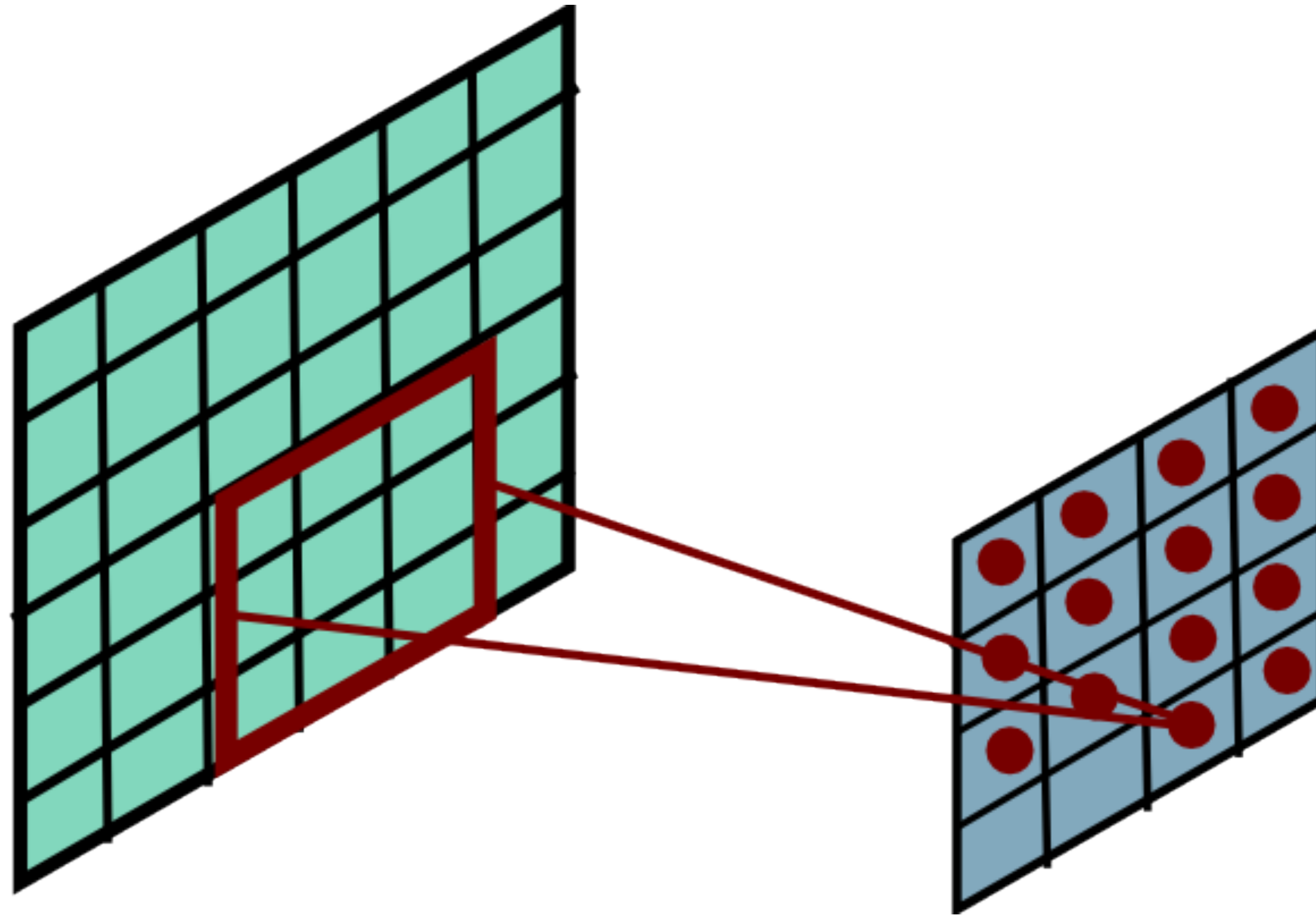
Convolutional Layer



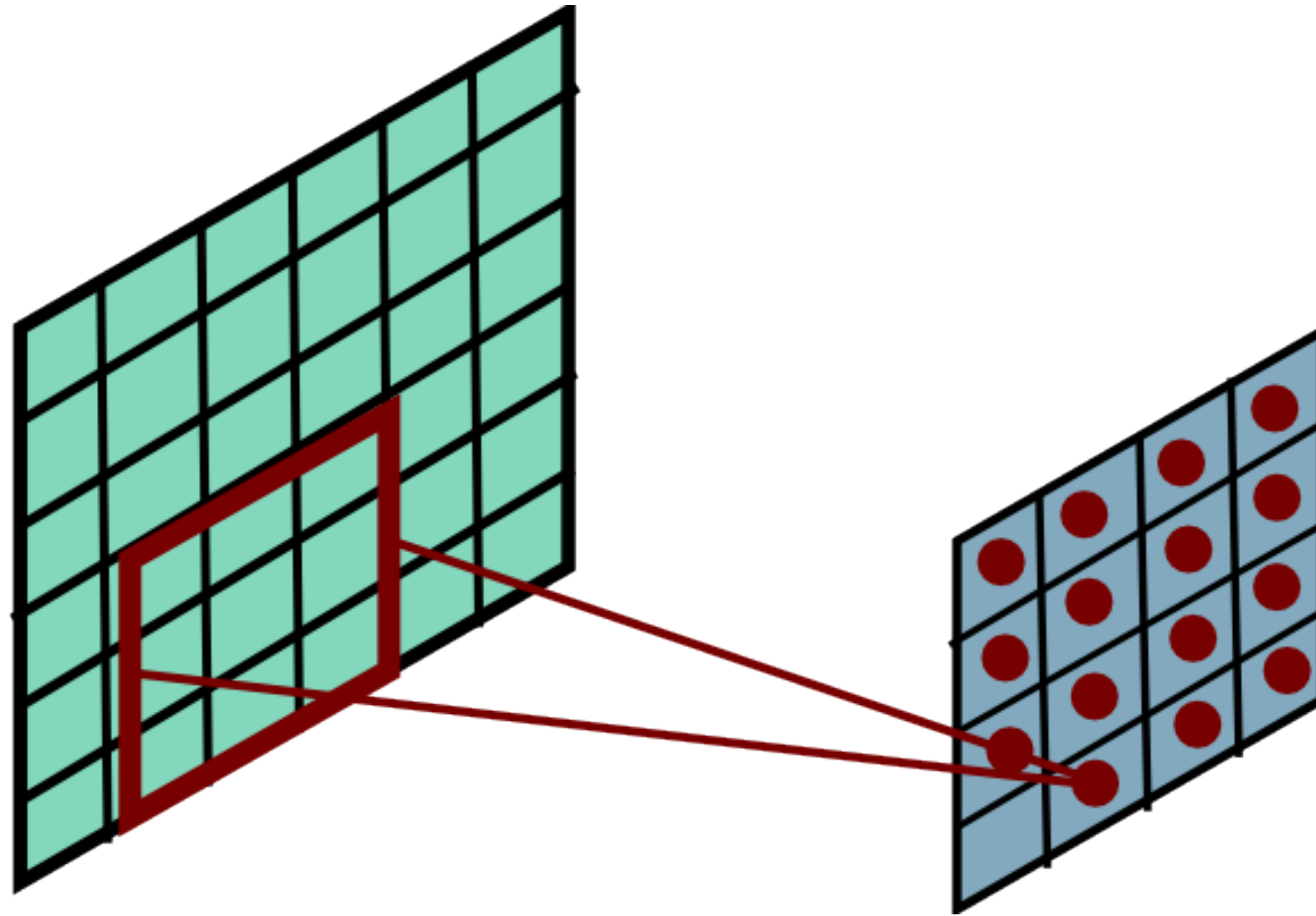
Convolutional Layer



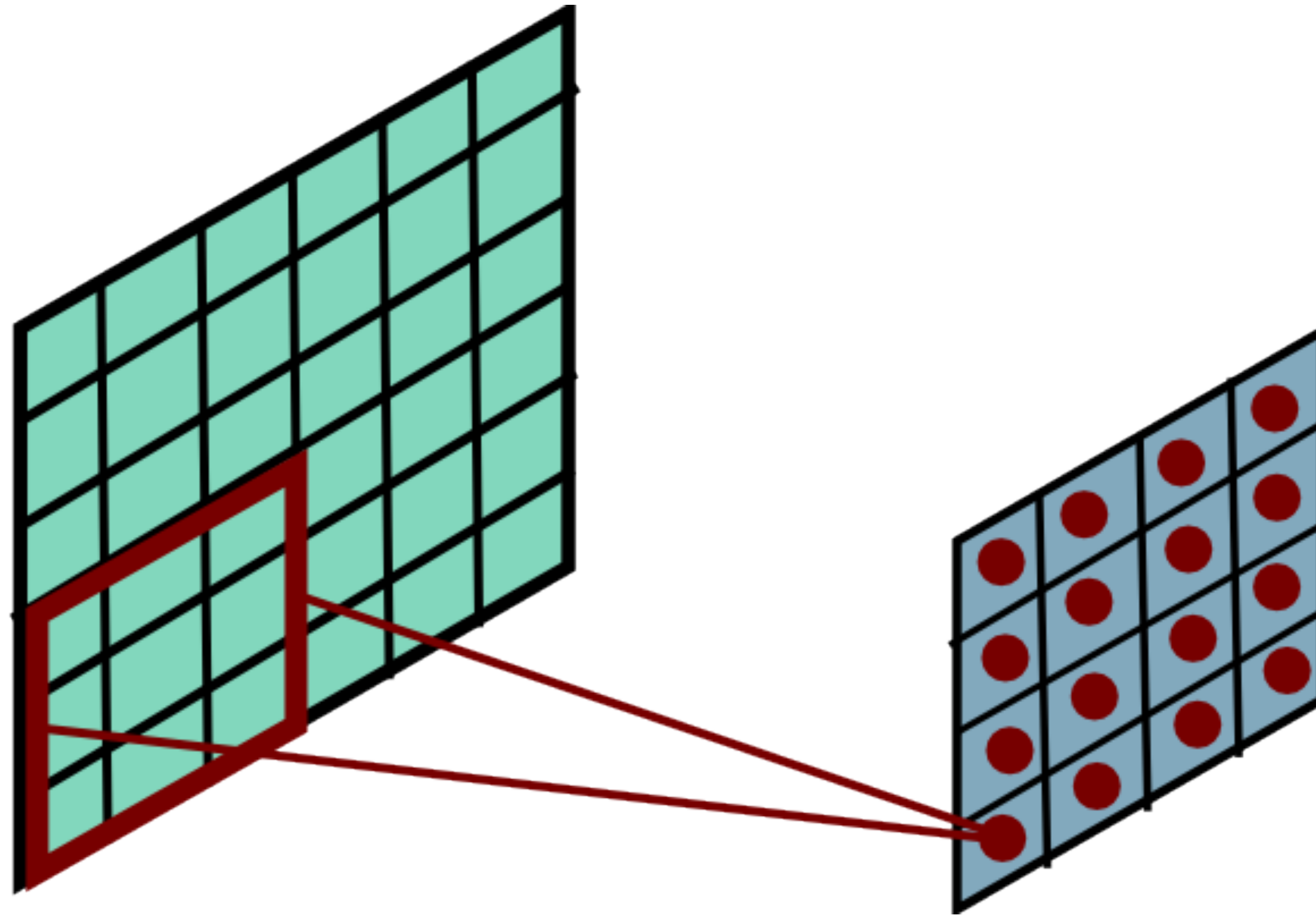
Convolutional Layer



Convolutional Layer

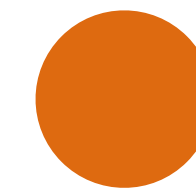
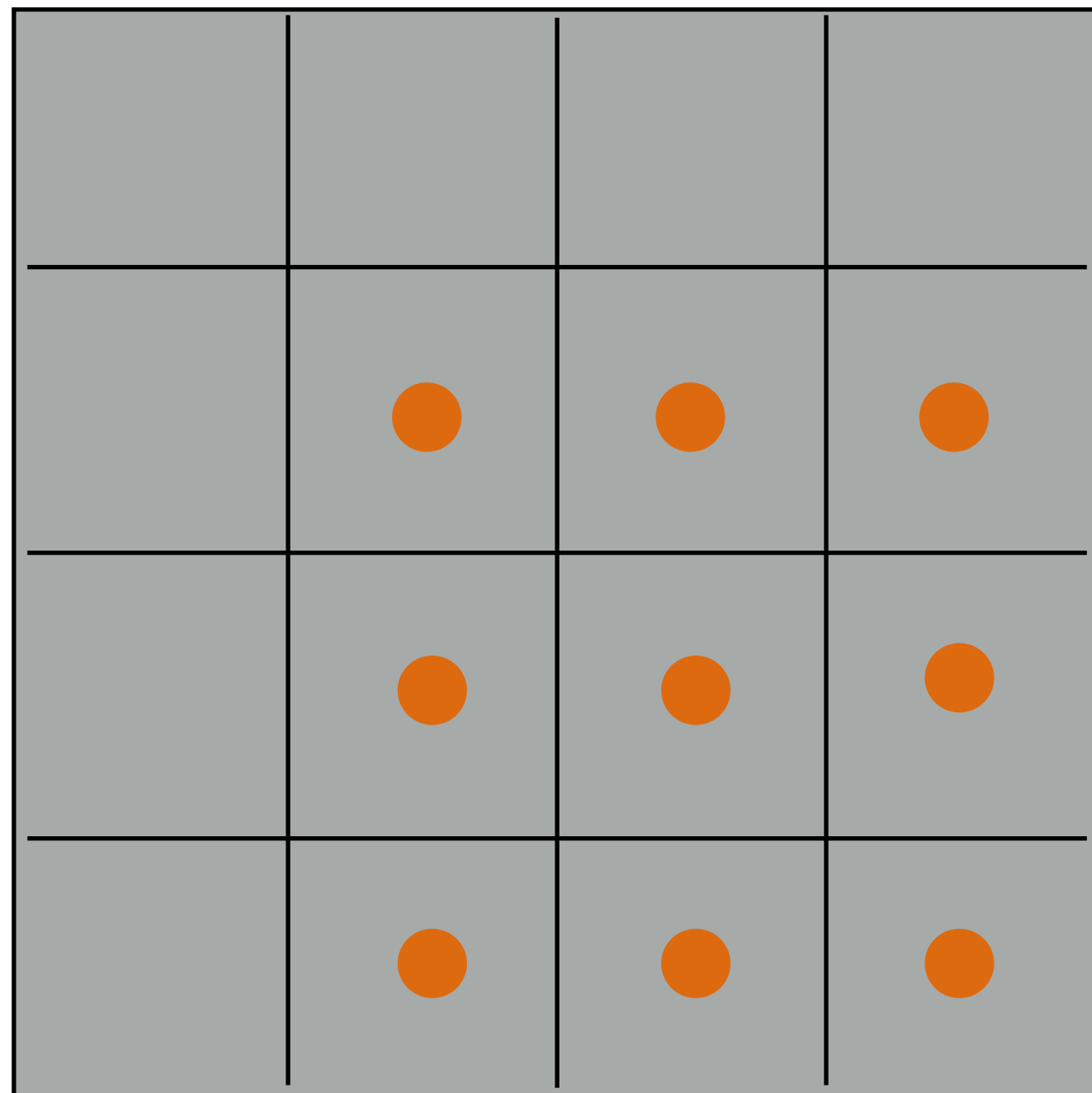


Convolutional Layer

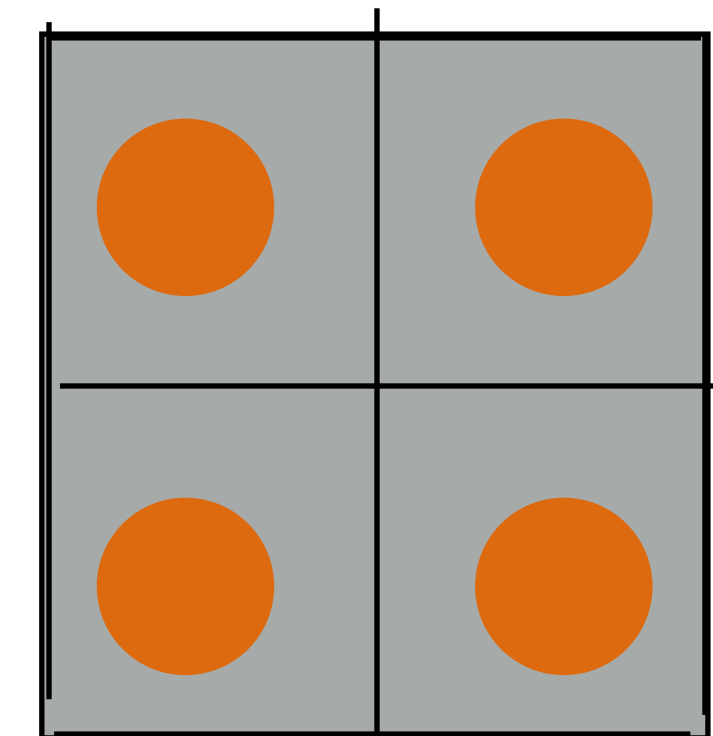


Convolutional Layer: Interpretation #2

One neuron applied as convolution (by shifting)



neurons



output

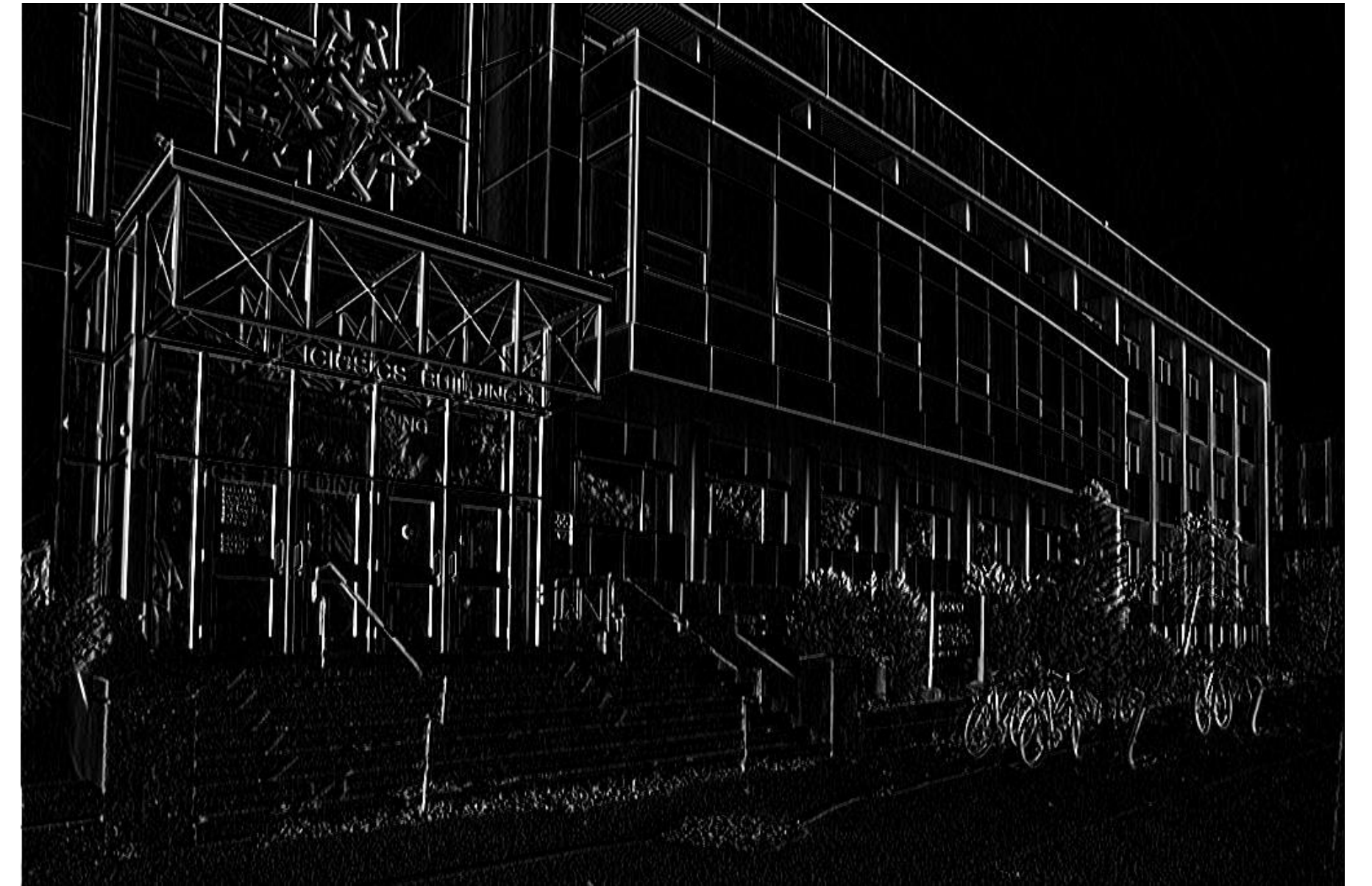
$$\sigma \left(\sum_{i=1}^3 \sum_{j=1}^3 \mathbf{w}_{i,j} \mathcal{I}(i,j) + b \right)$$

Similar to Filter in Normalized Correlation

Convolution Layer



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



Convolution Layer

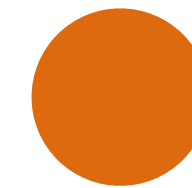
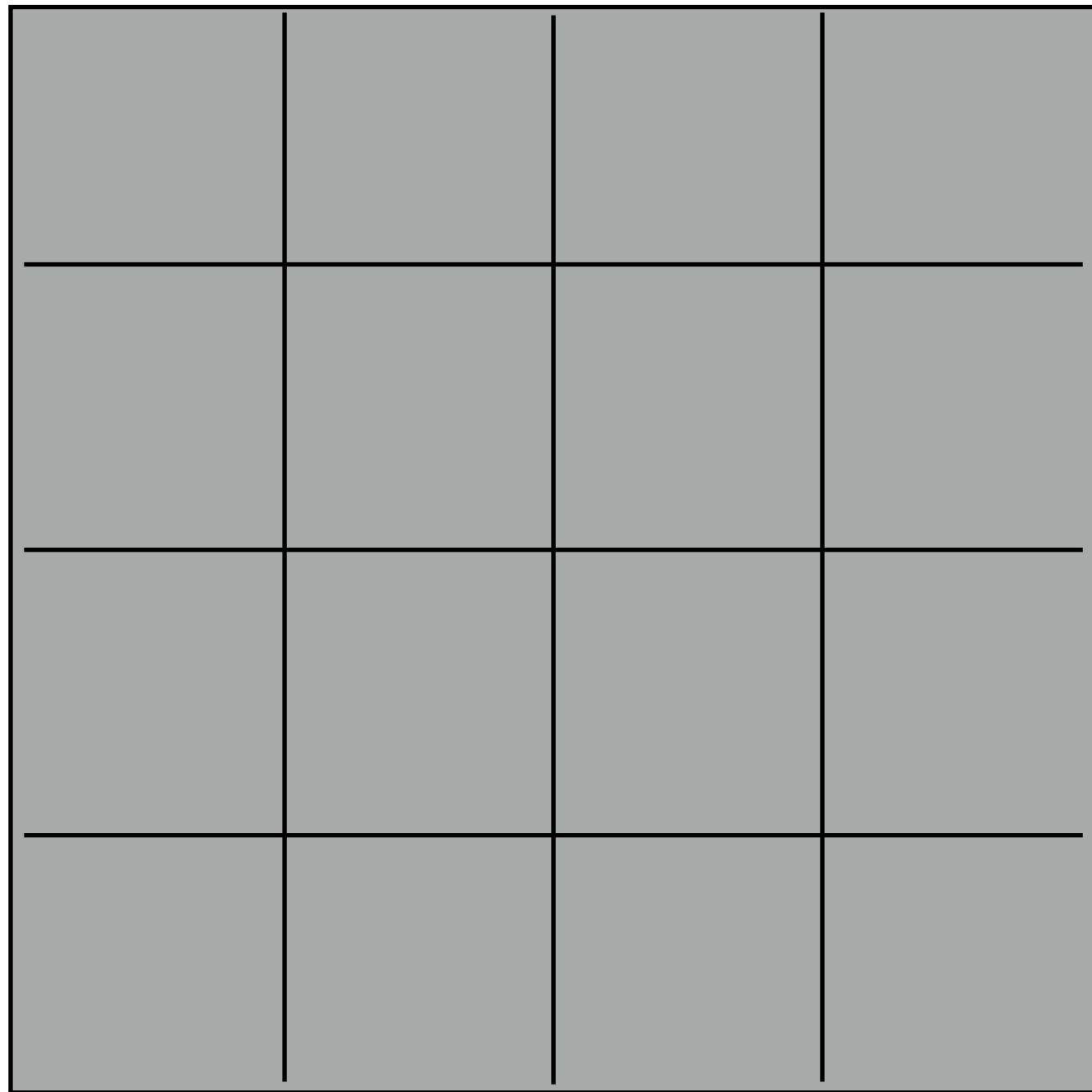


$$\star \begin{bmatrix} 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \end{bmatrix} \rightarrow$$

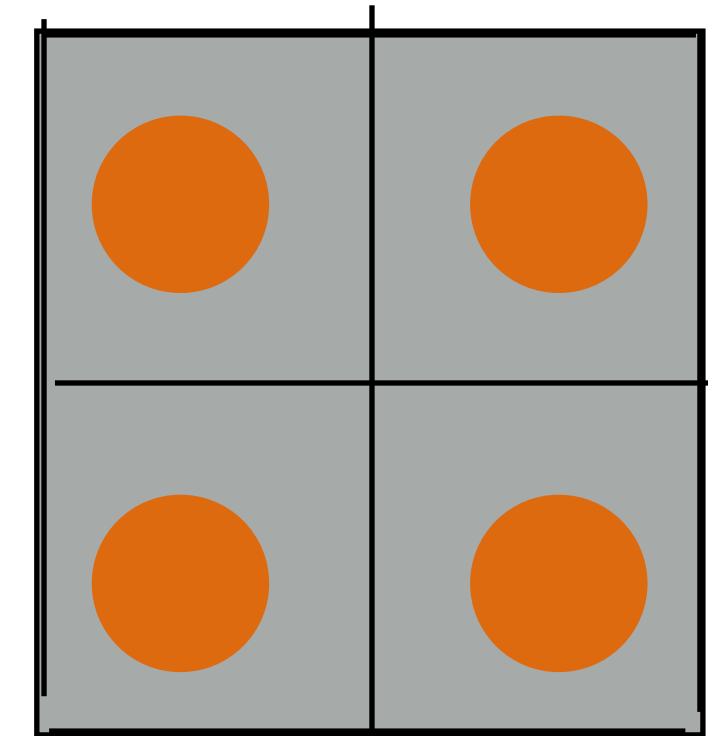


Convolutional Layer: Interpretation #2

One neuron applied as convolution (by shifting)

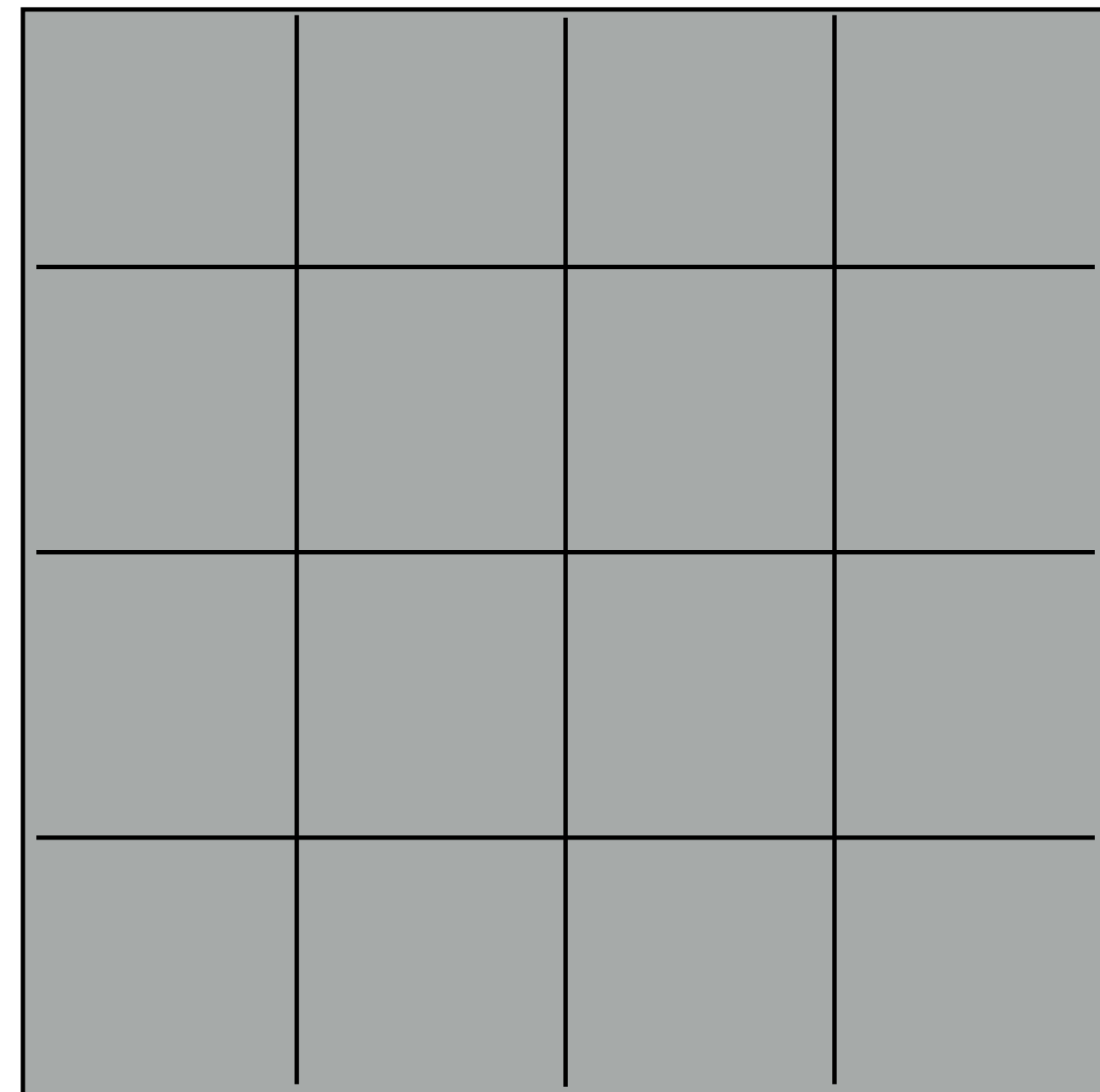


neurons

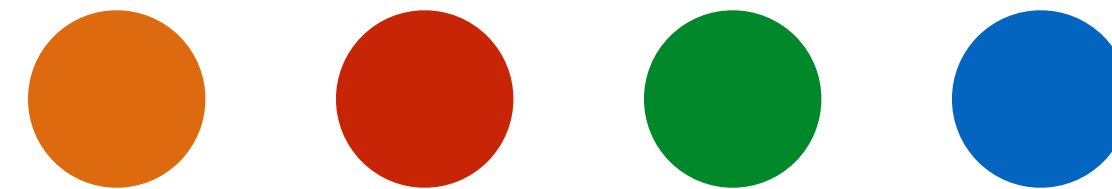


output

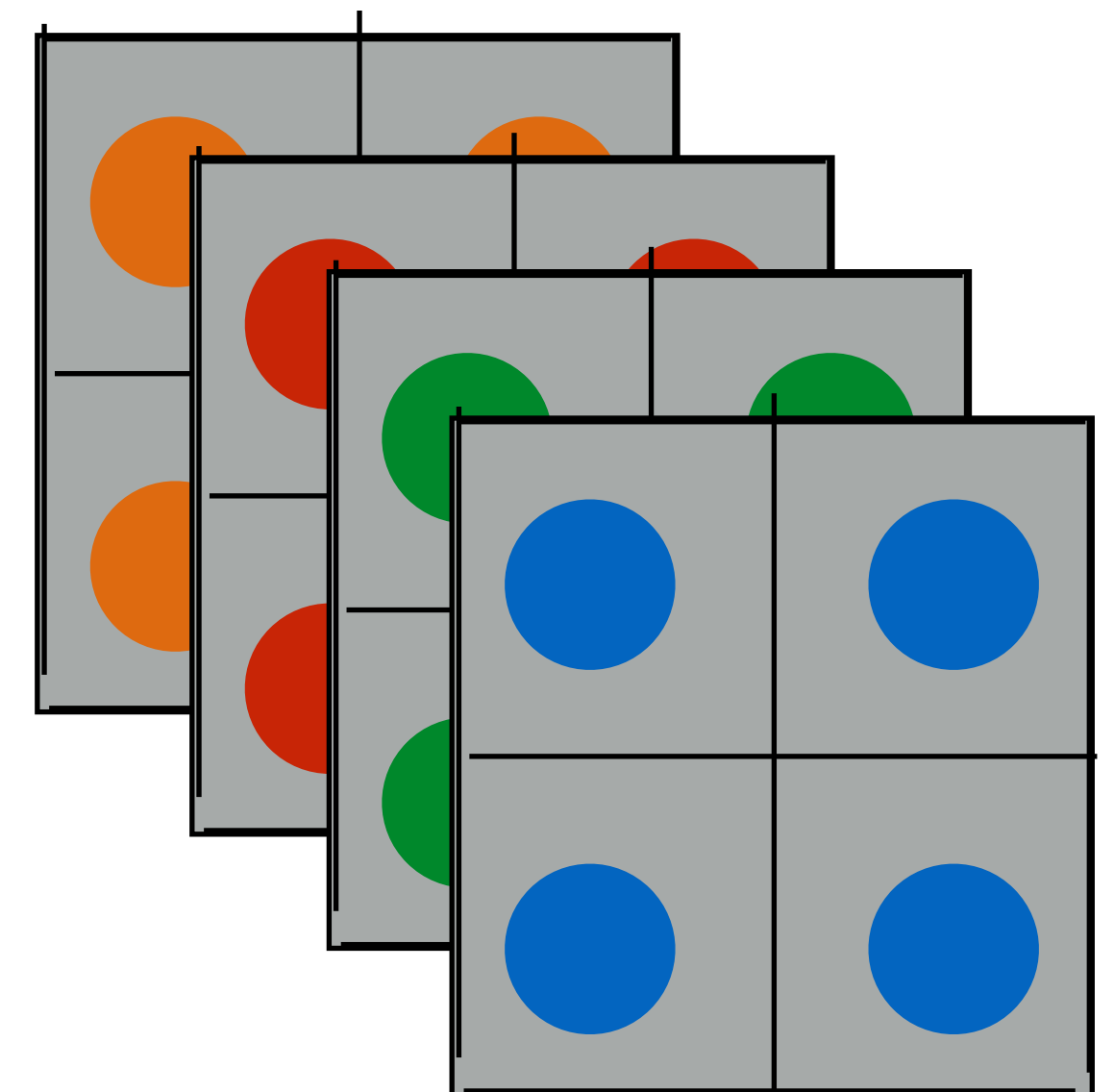
Convolutional Layer: Interpretation #2



One neuron applied as convolution (by shifting)



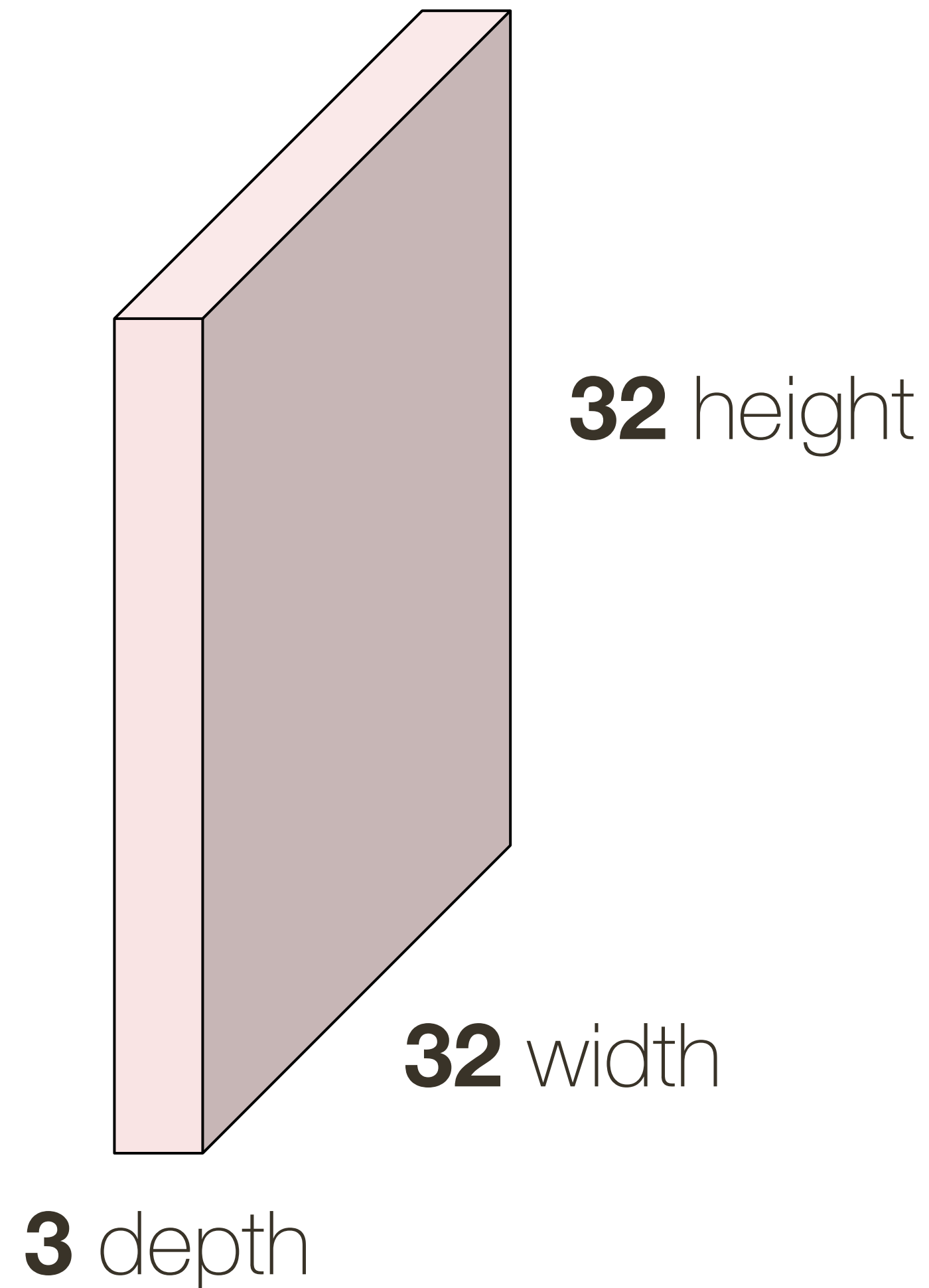
neurons



output

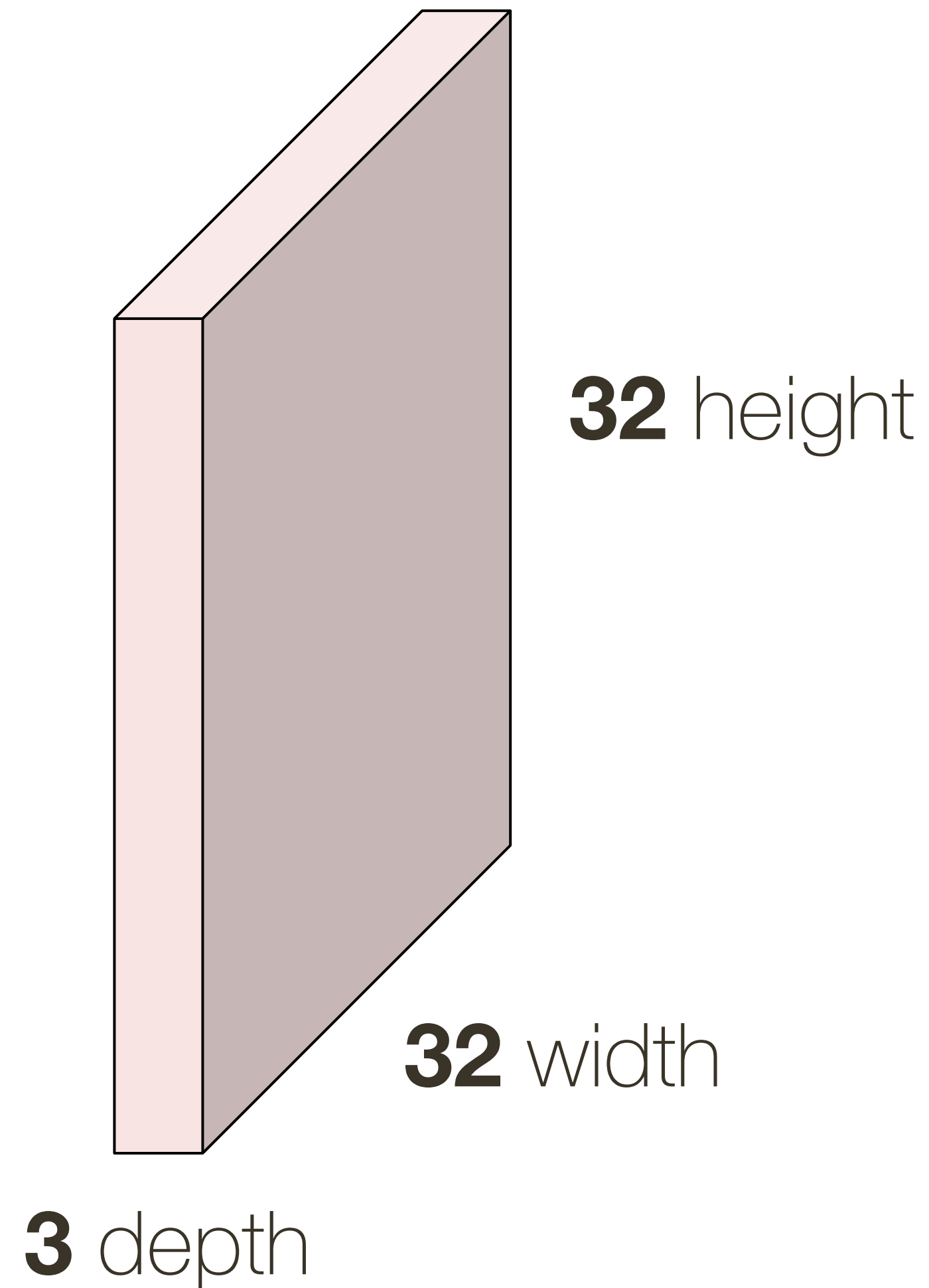
Convolutional Layer

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



Convolutional Layer

$32 \times 32 \times 3$ **image**



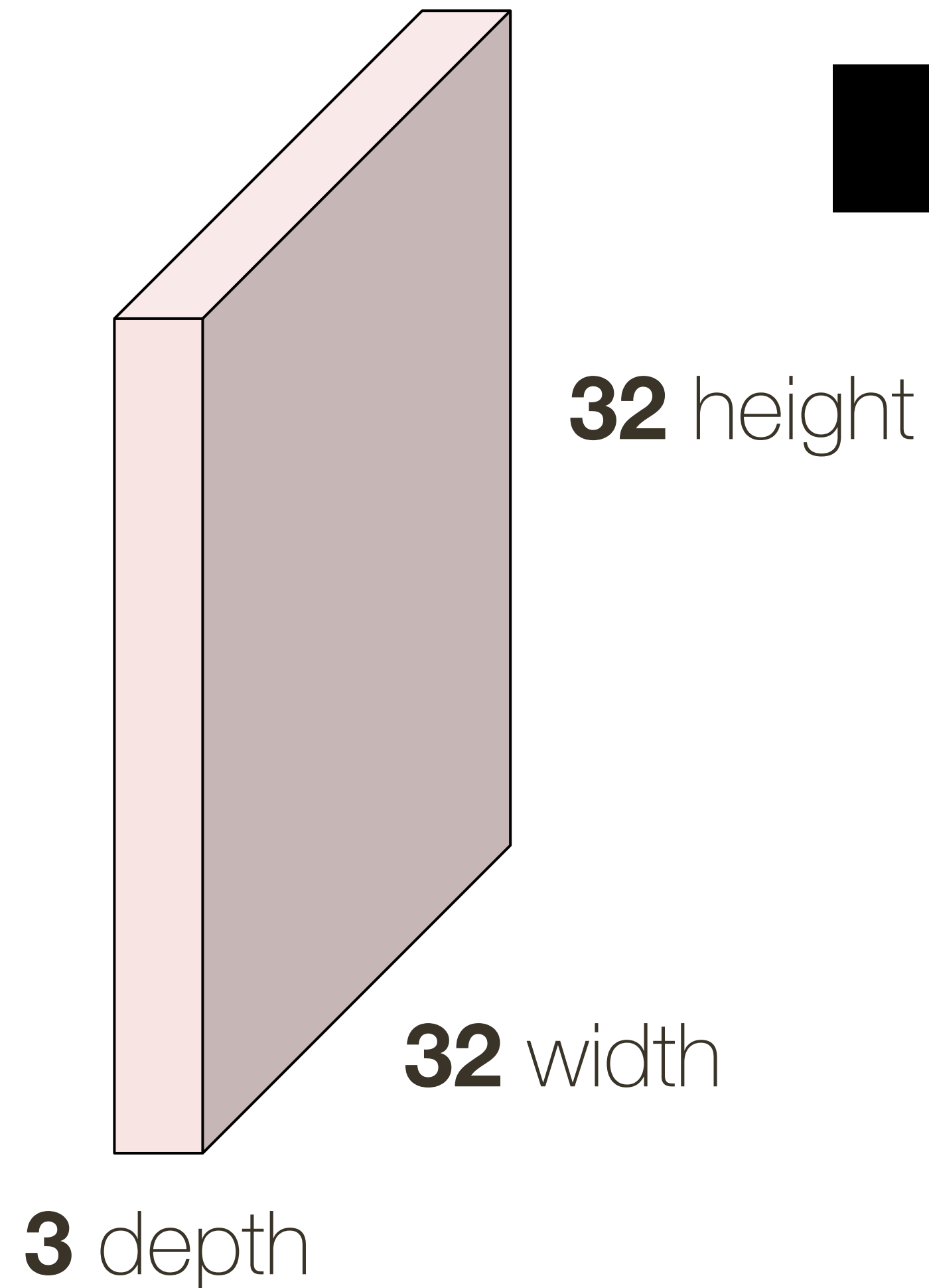
$5 \times 5 \times 3$ **filter**



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

Convolutional Layer

32 x 32 x **3** image



Filters always extend the full depth of the input volume

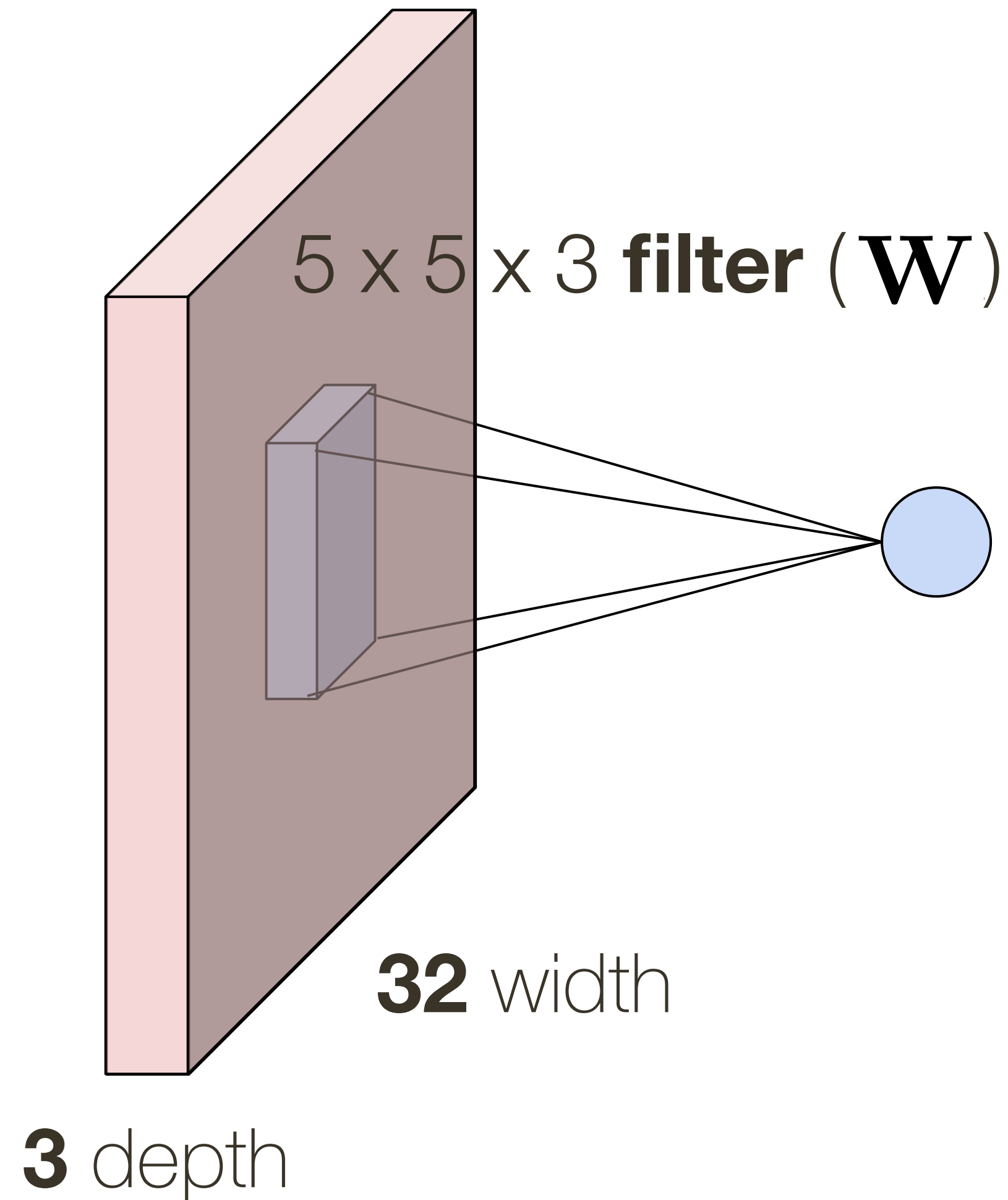
5 x 5 x **3** filter



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

Convolutional Layer

32 x 32 x 3 **image**

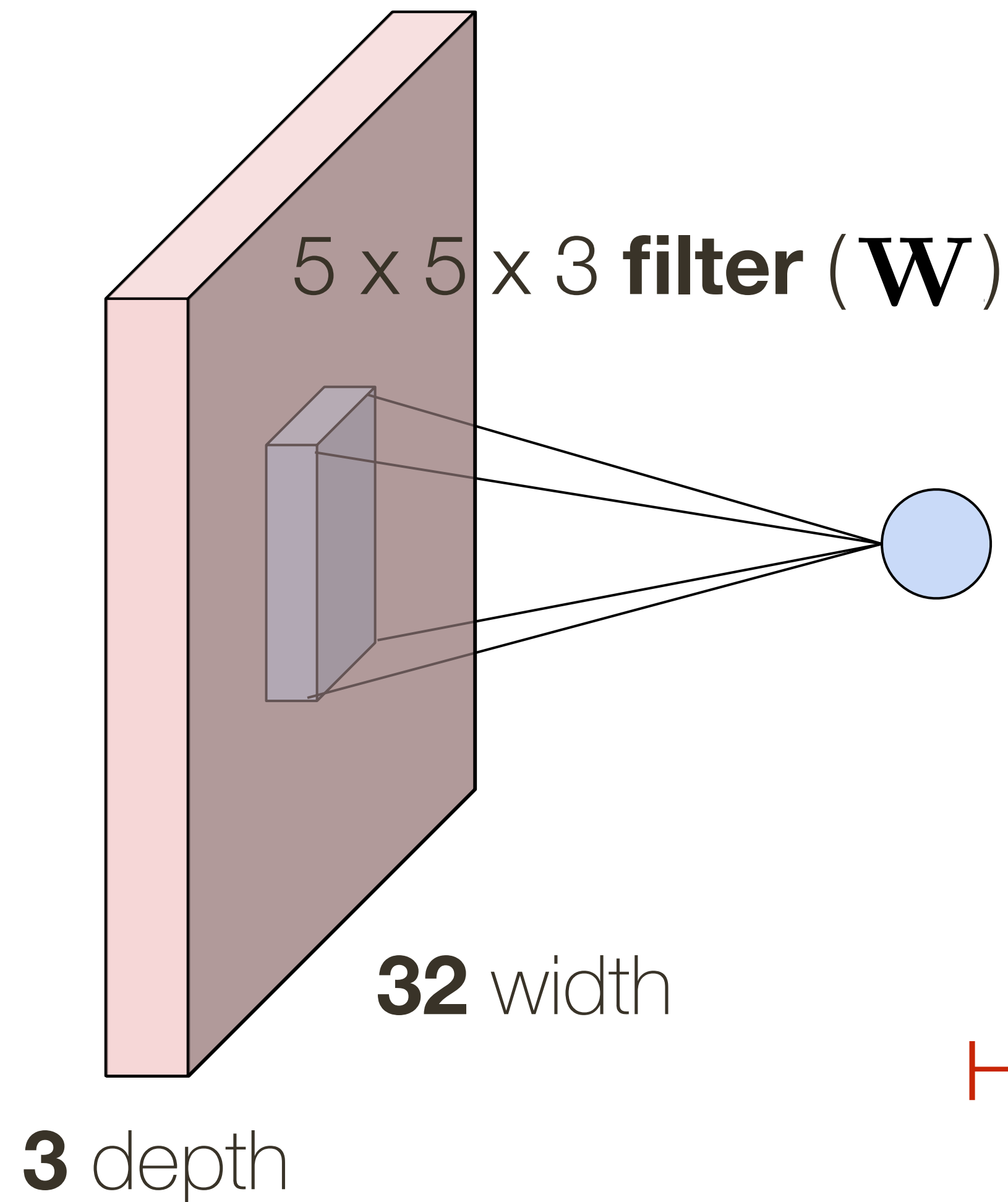


1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

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

Convolutional Layer

32 x 32 x 3 **image**



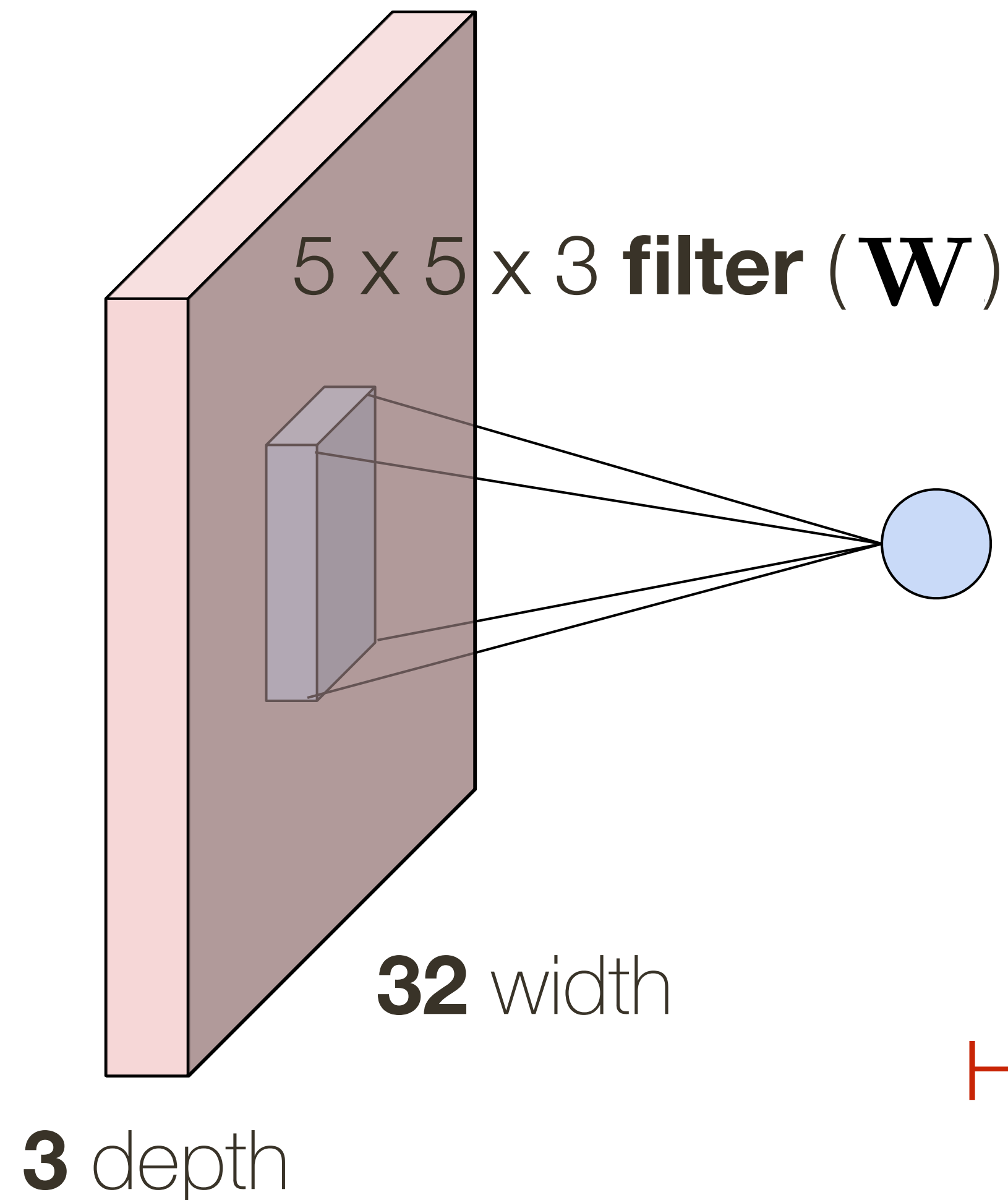
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

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

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

Convolutional Layer

32 x 32 x 3 **image**



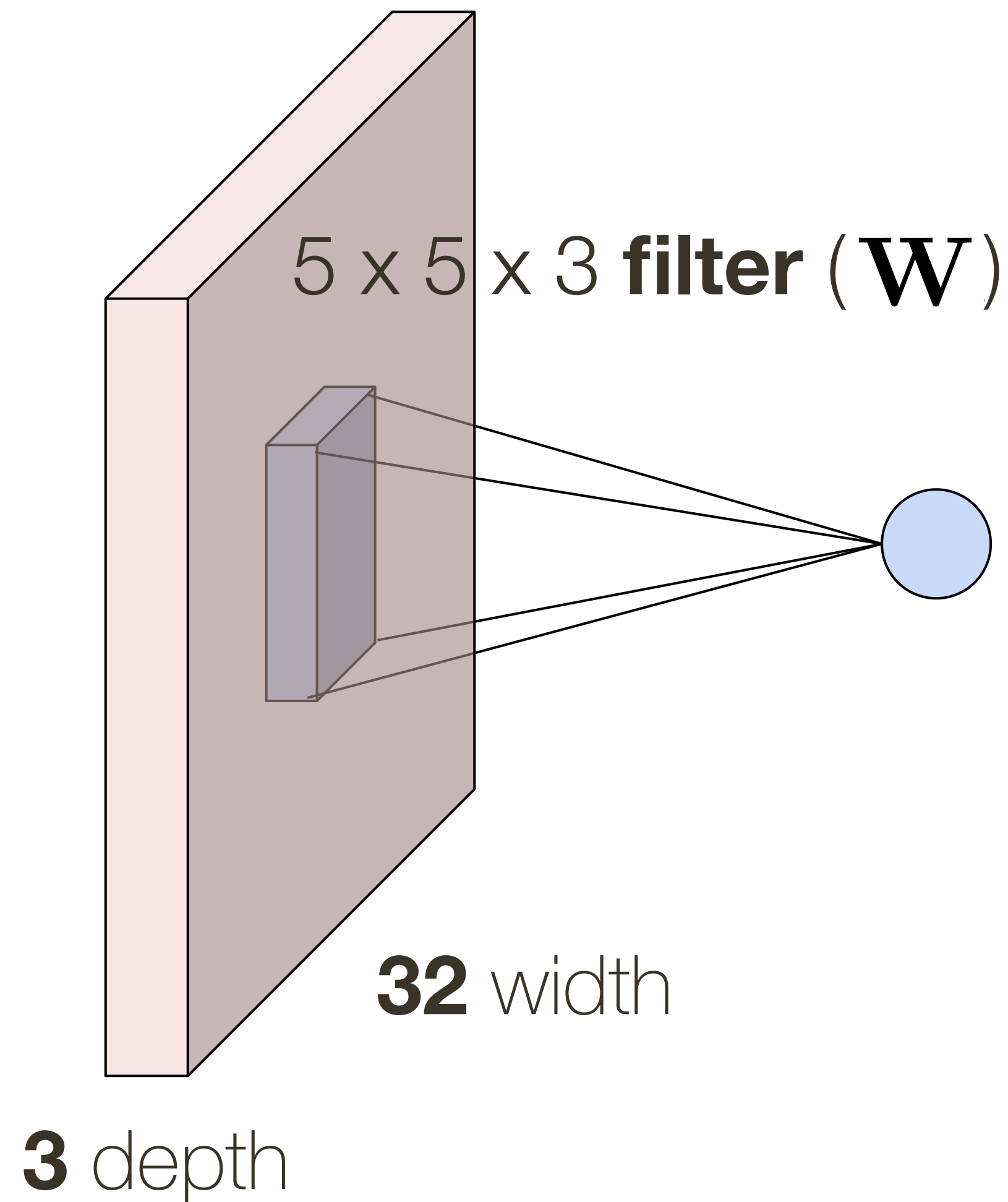
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

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

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

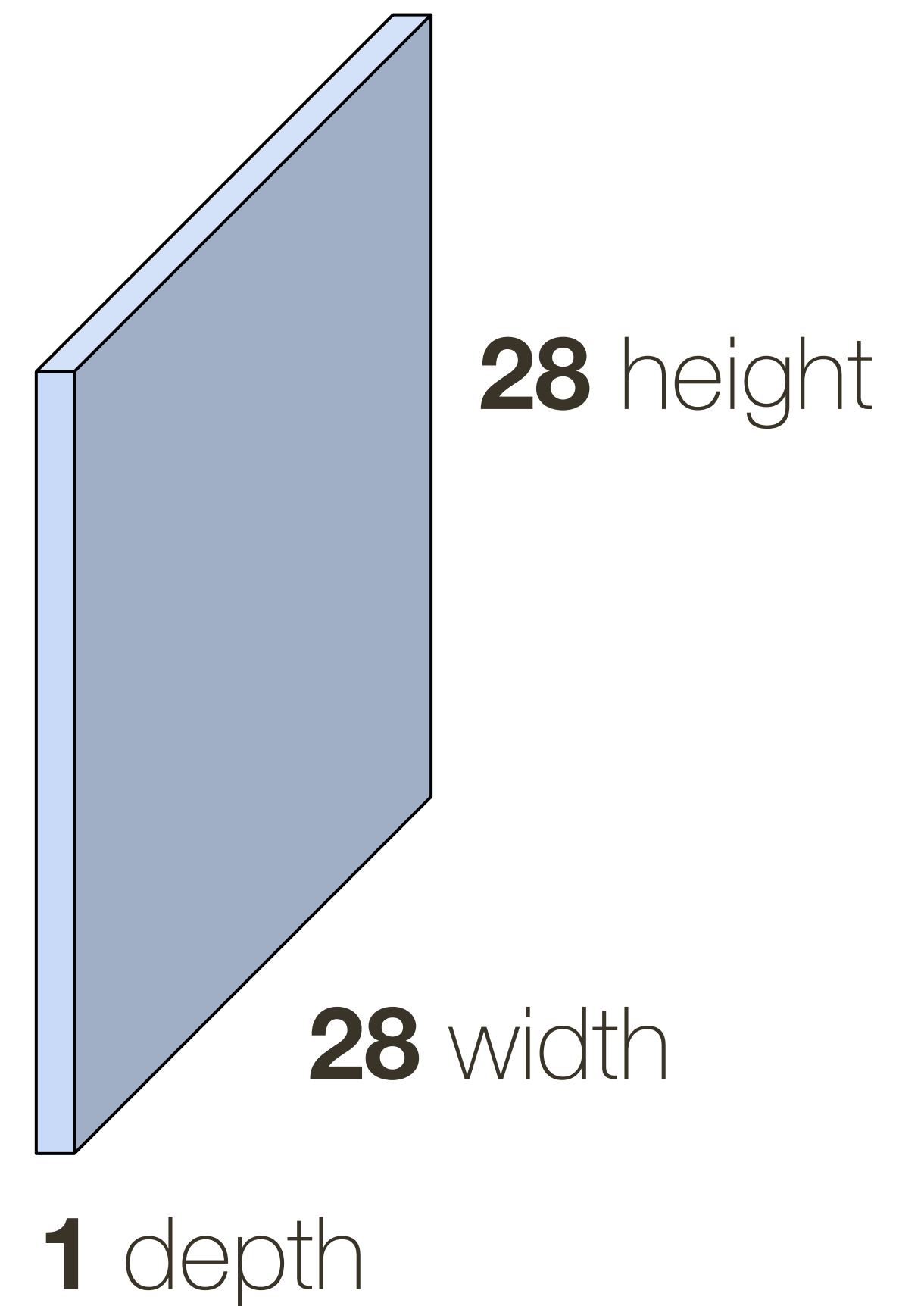
Convolutional Layer

32 x 32 x 3 **image**



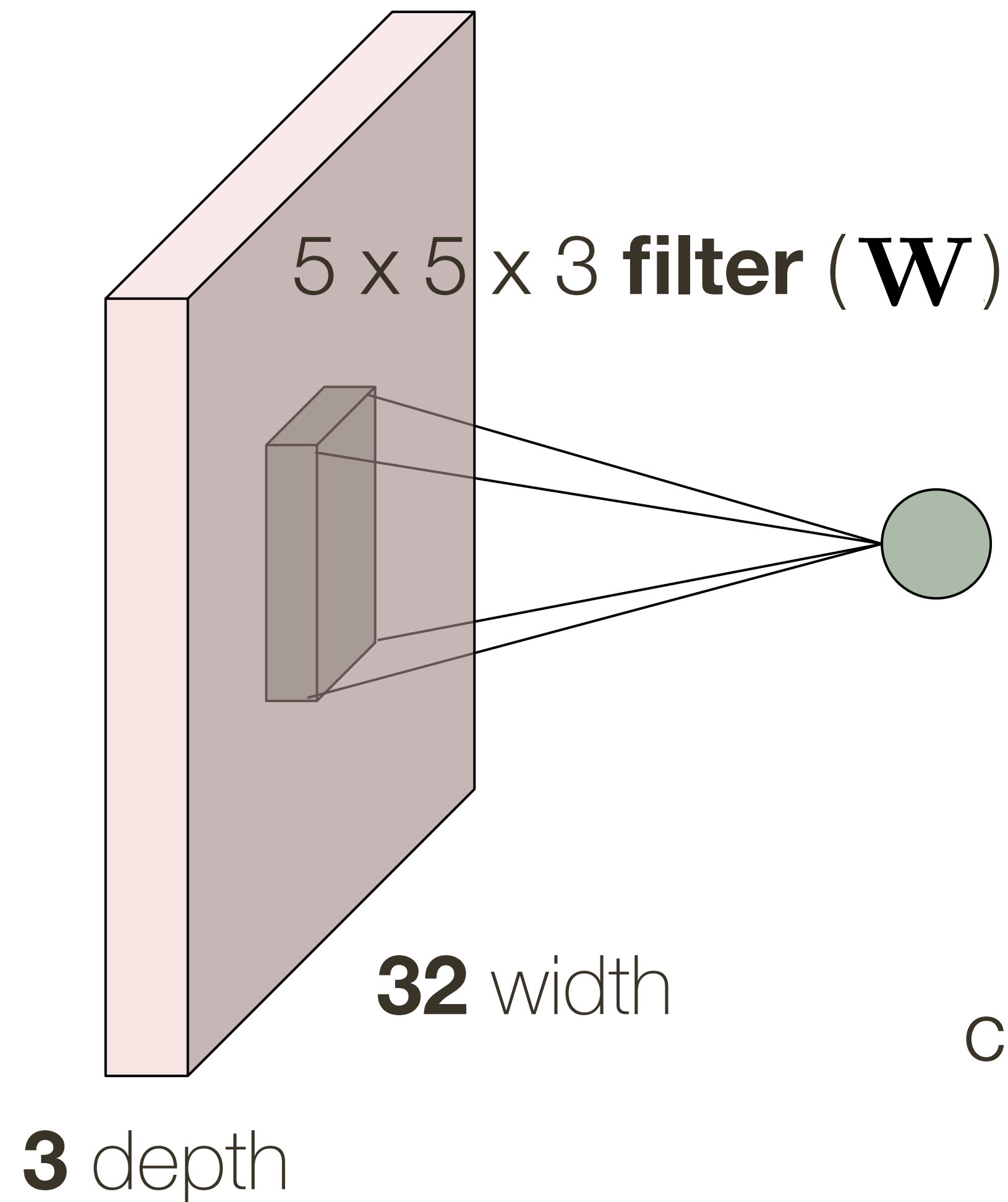
convolve (slide) over all
spatial locations

activation map



Convolutional Layer

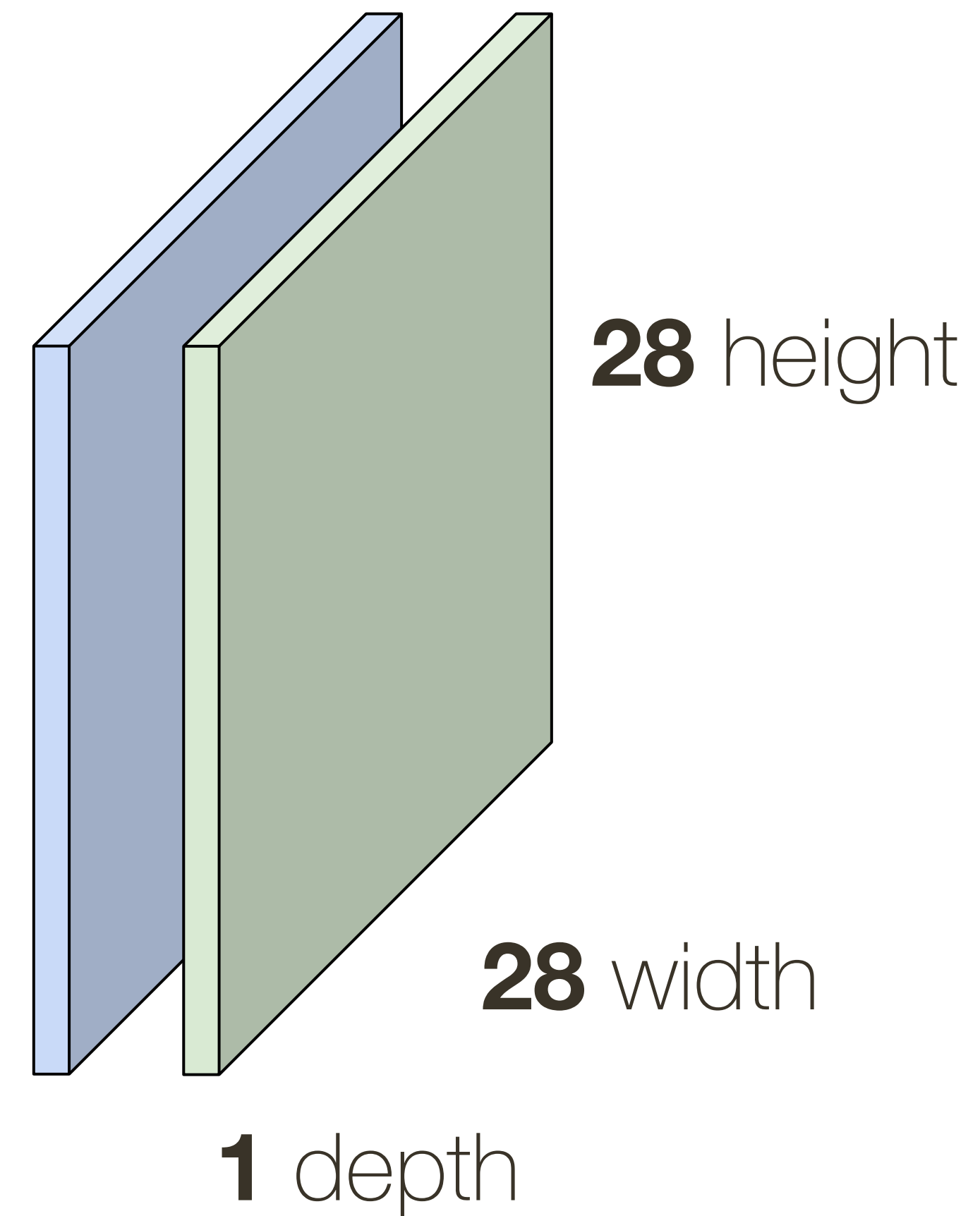
32 x 32 x 3 **image**



convolve (slide) over all
spatial locations

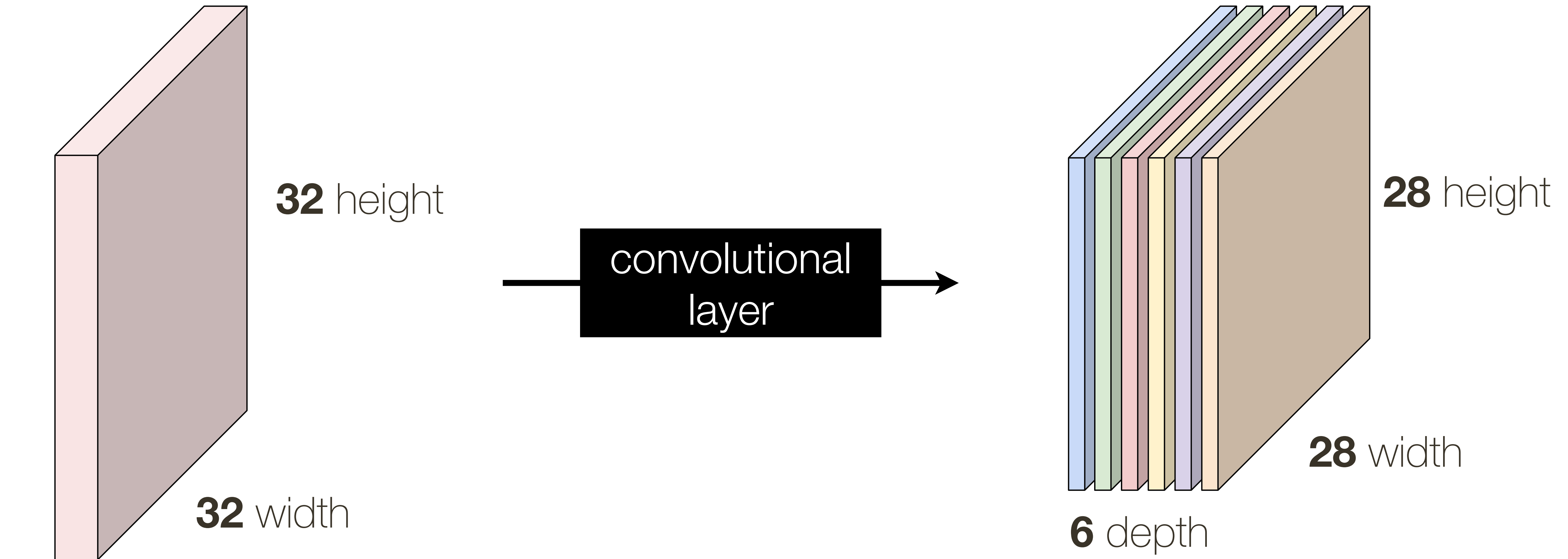
consider another **green** filter

activation map



Convolutional Layer

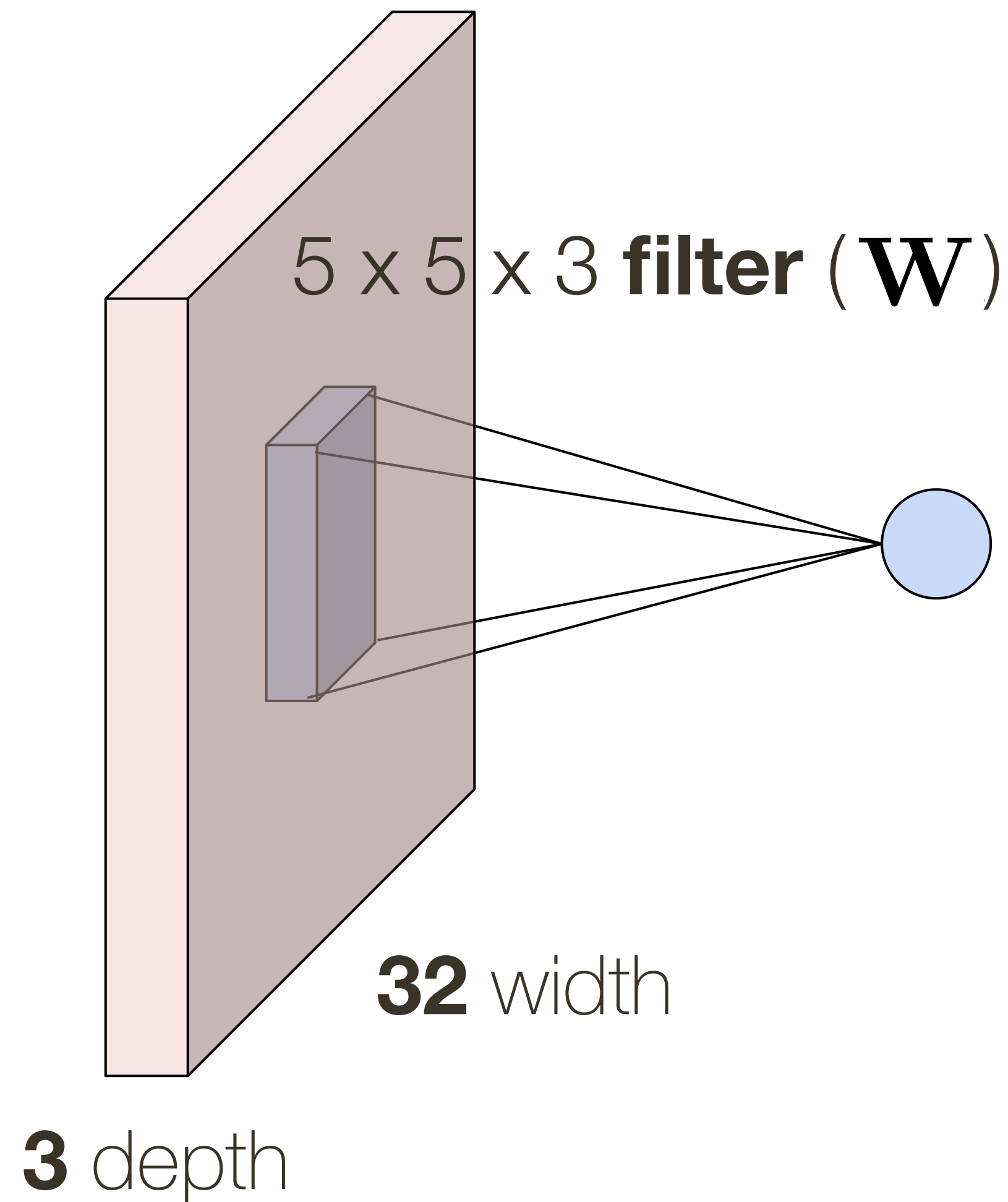
If we have 6 5x5 filter, we'll get 6 separate activation maps: **activation** map



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

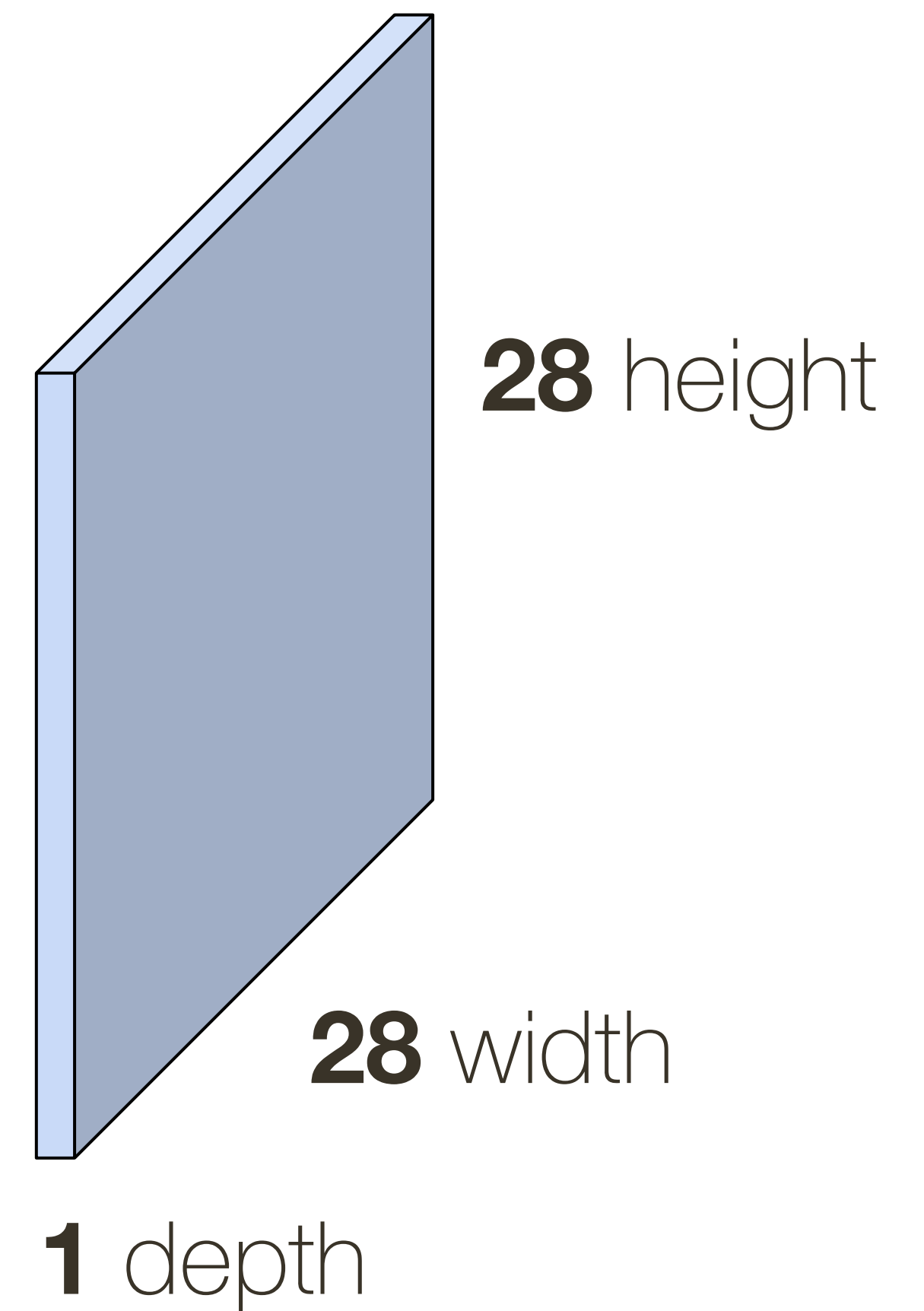
Convolutional Layer: Closer Look at **Spatial Dimensions**

32 x 32 x 3 image

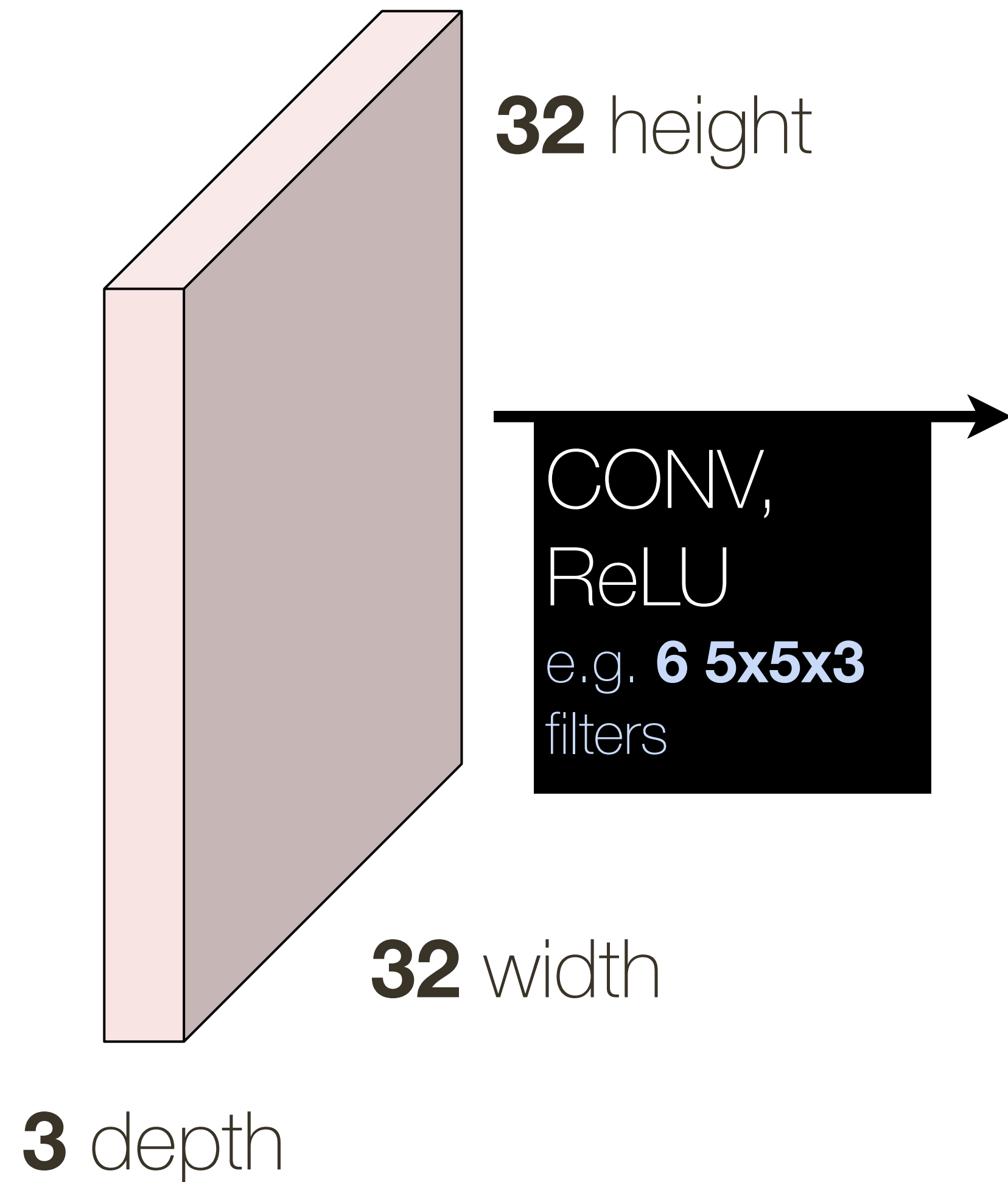


convolve (slide) over all
spatial locations

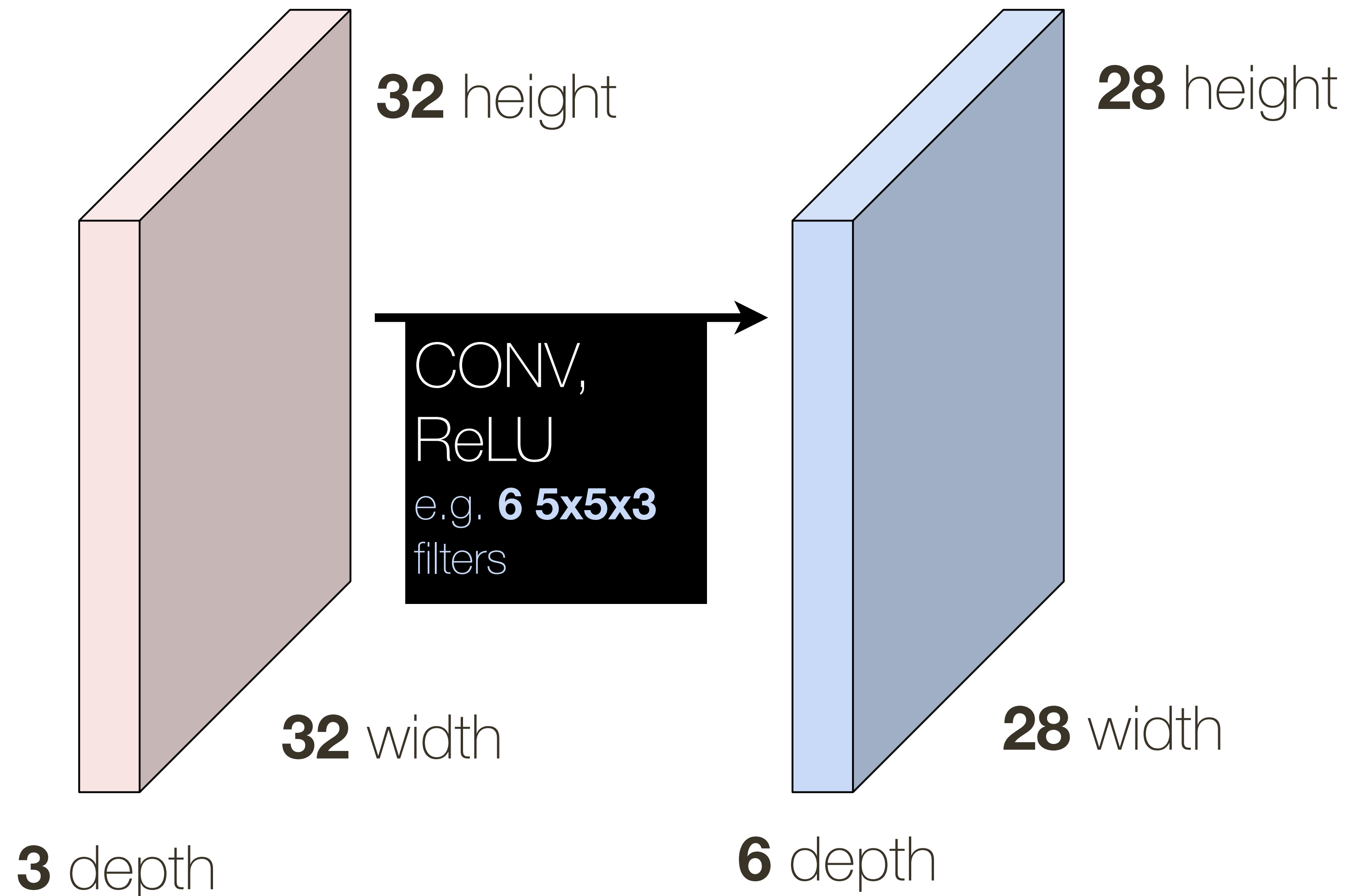
activation map



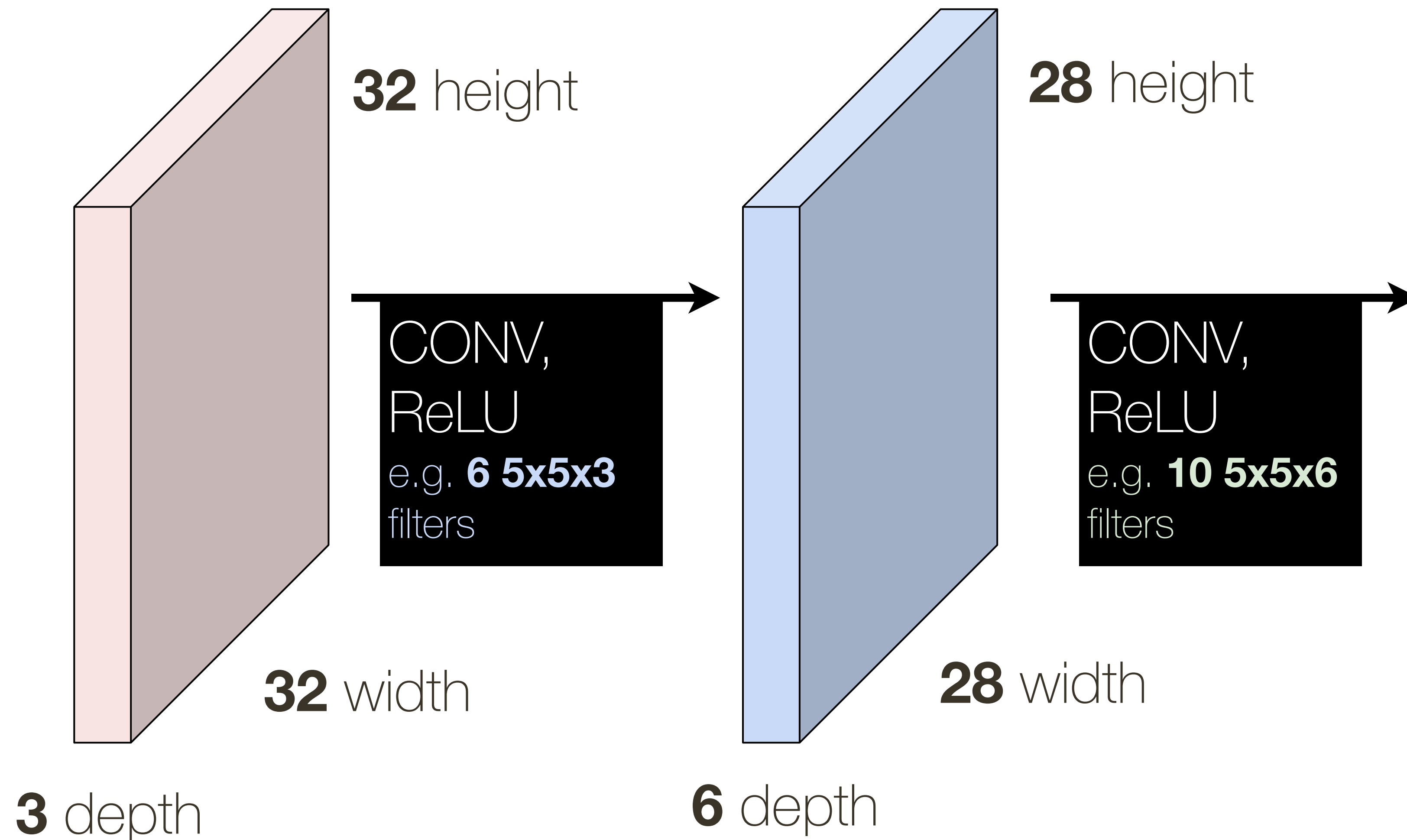
Convolutional Neural Network (ConvNet)



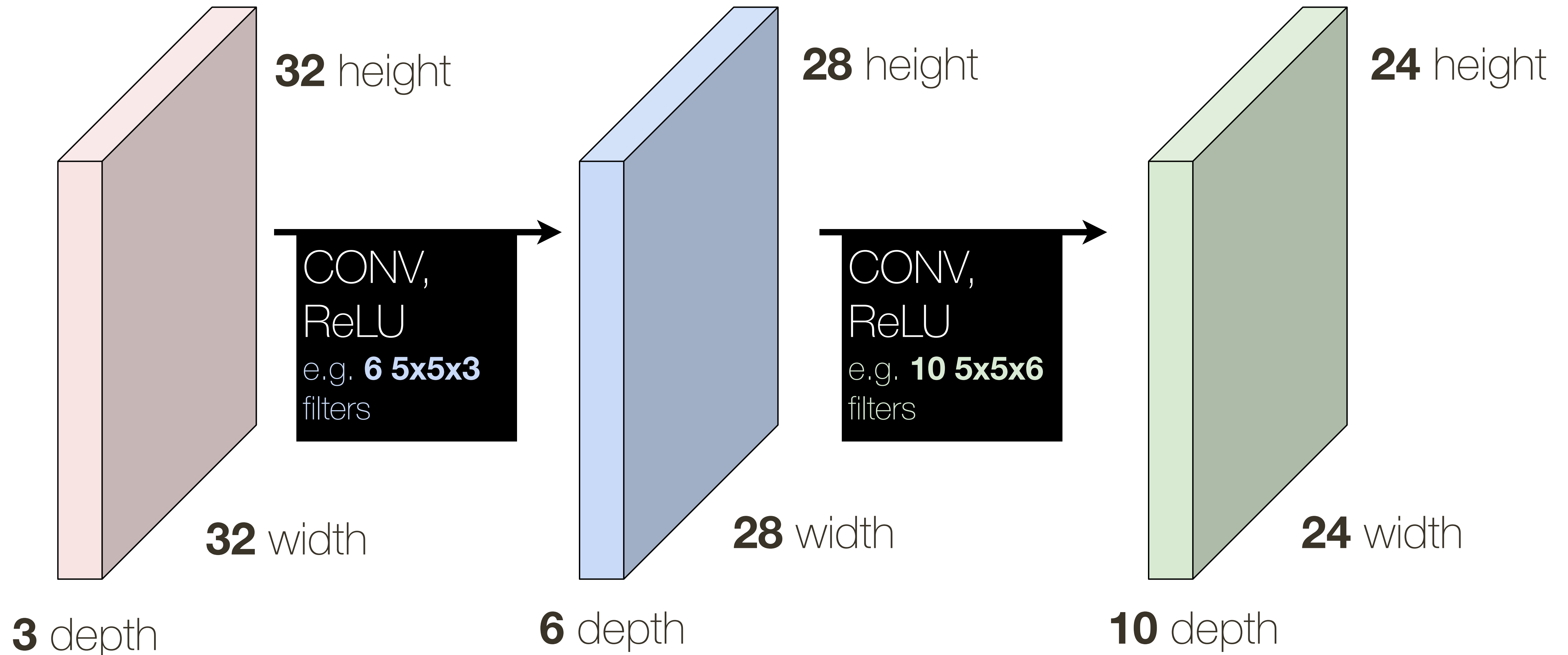
Convolutional Neural Network (ConvNet)



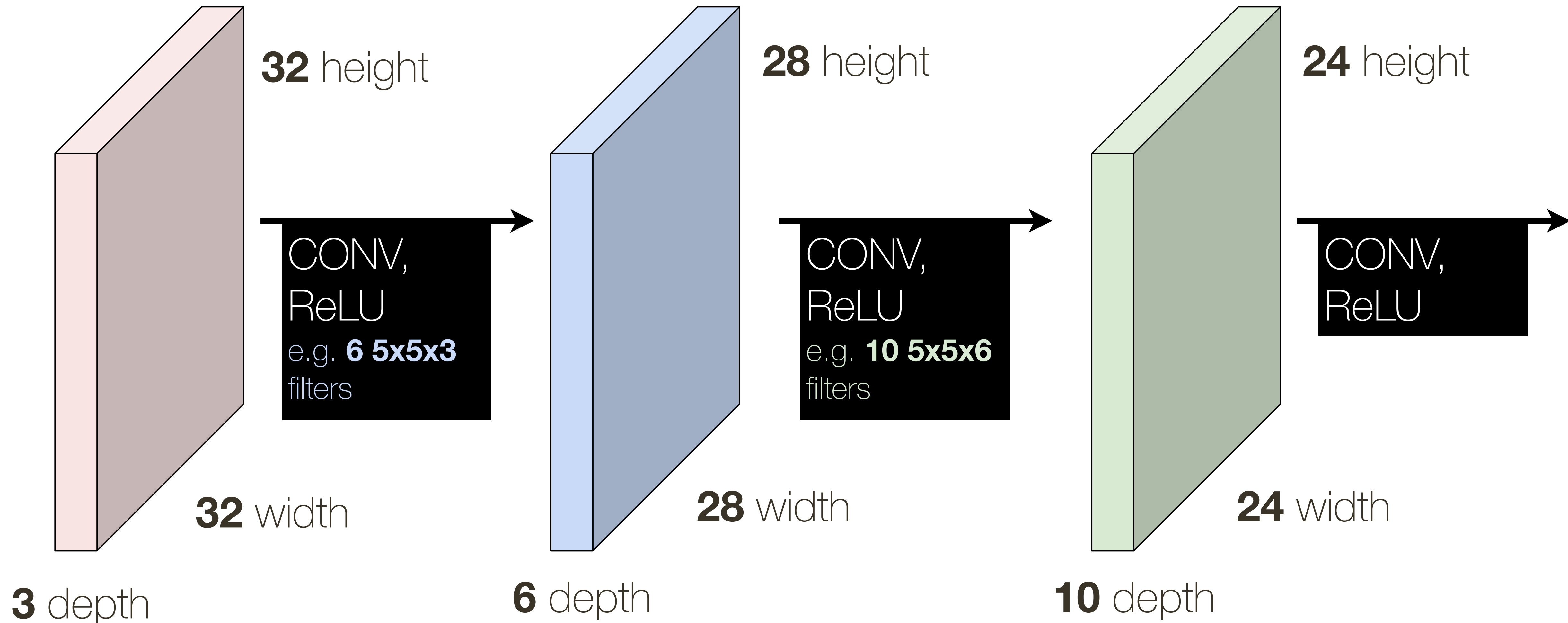
Convolutional Neural Network (ConvNet)



Convolutional Neural Network (ConvNet)

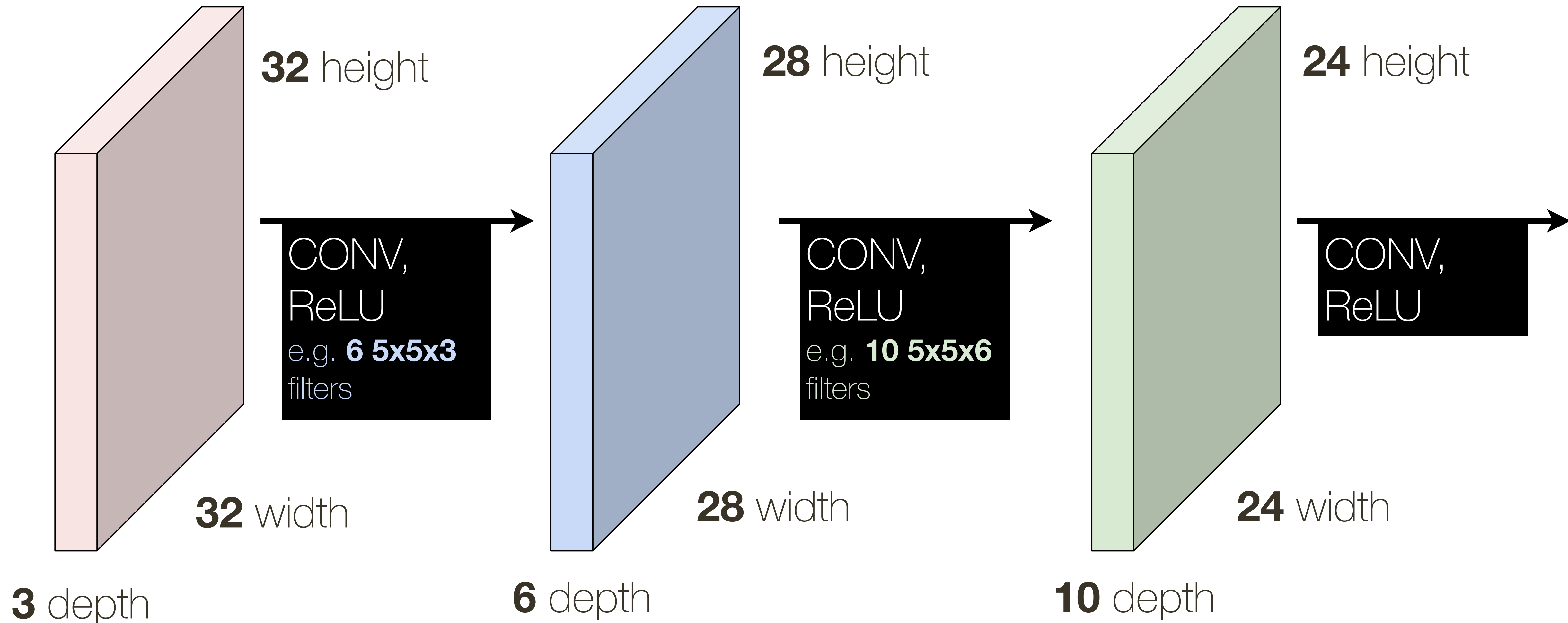


Convolutional Neural Network (ConvNet)



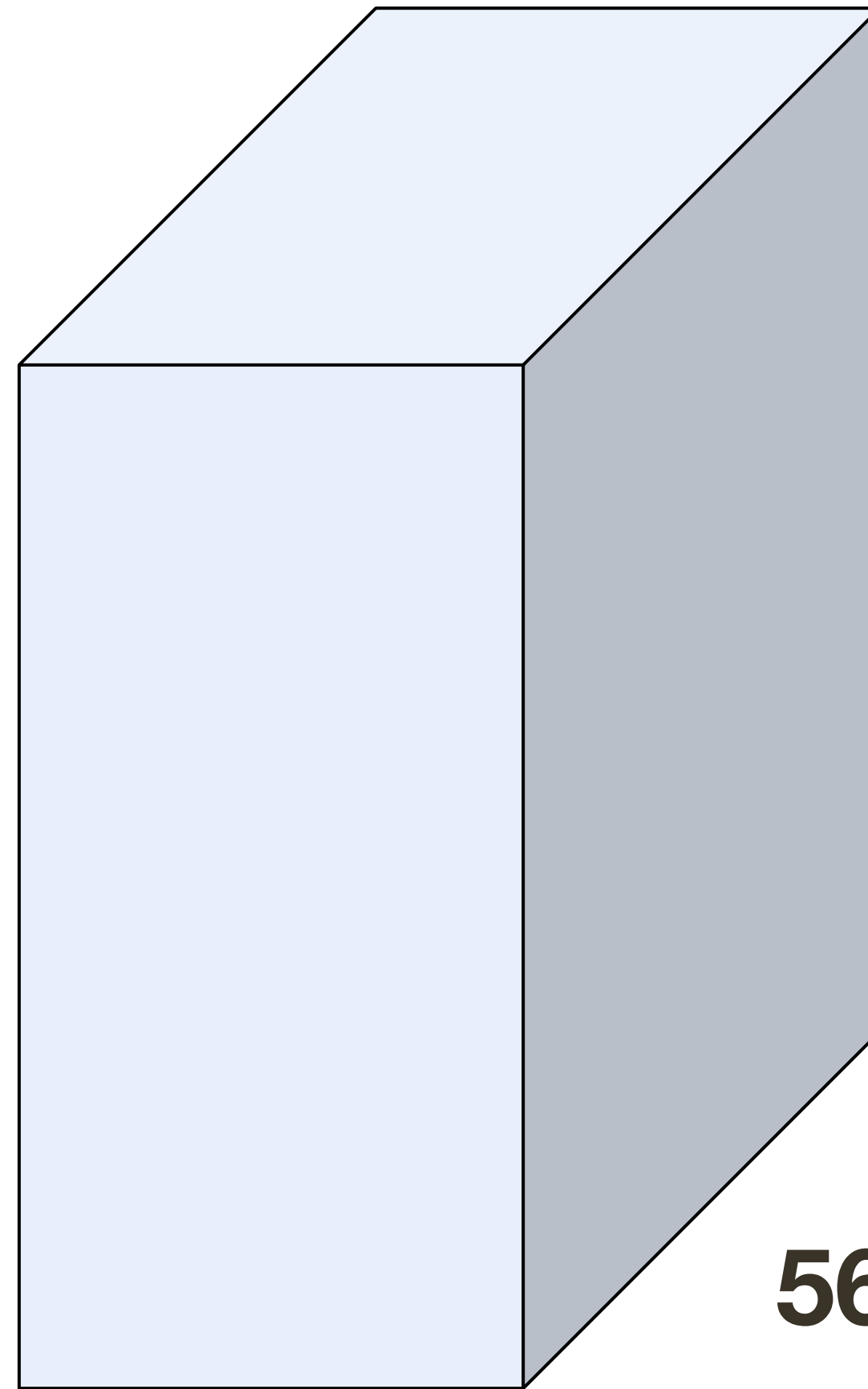
Convolutional Neural Network (ConvNet)

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



Convolutional Layer: **1x1 convolutions**

56 x 56 x 64 **image**



56 height

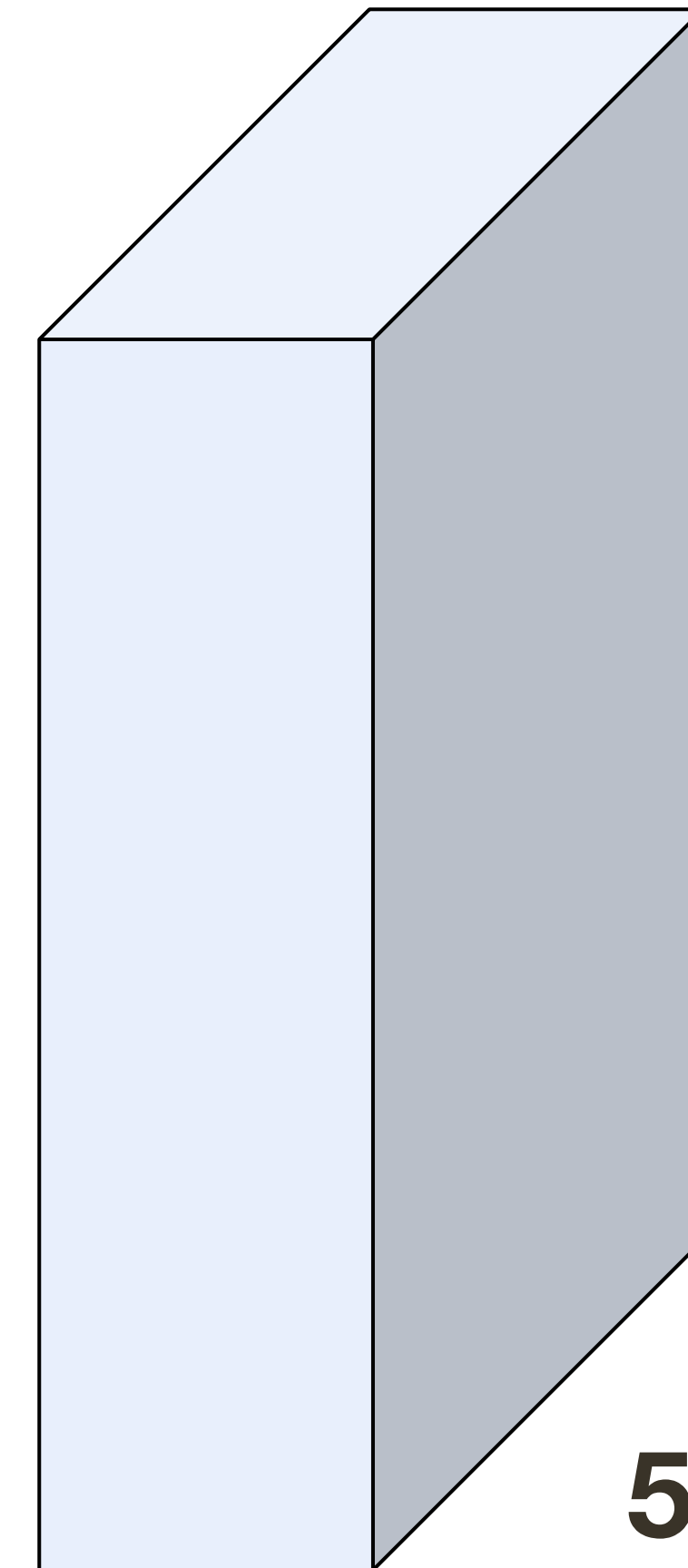
56 width

64 depth

32 **filters** of size, 1 x 1 x 64



56 x 56 x 32 **image**



56 height

56 width

32 depth

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

Convolutional Layer **Summary**

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

Convolutional Layer **Summary**

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

Requires hyperparameters:

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

Convolutional Layer **Summary**

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

Requires hyperparameters:

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

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

Convolutional Layer **Summary**

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

Requires hyperparameters:

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

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

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

Convolutional Layer **Summary**

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

Requires hyperparameters:

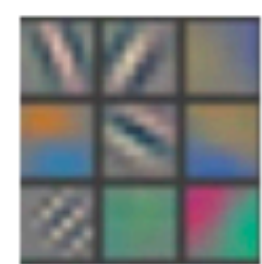
- Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, \dots\}$)
- Stride of application: S (for a typical network $S \in \{1, 2\}$)
- Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

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

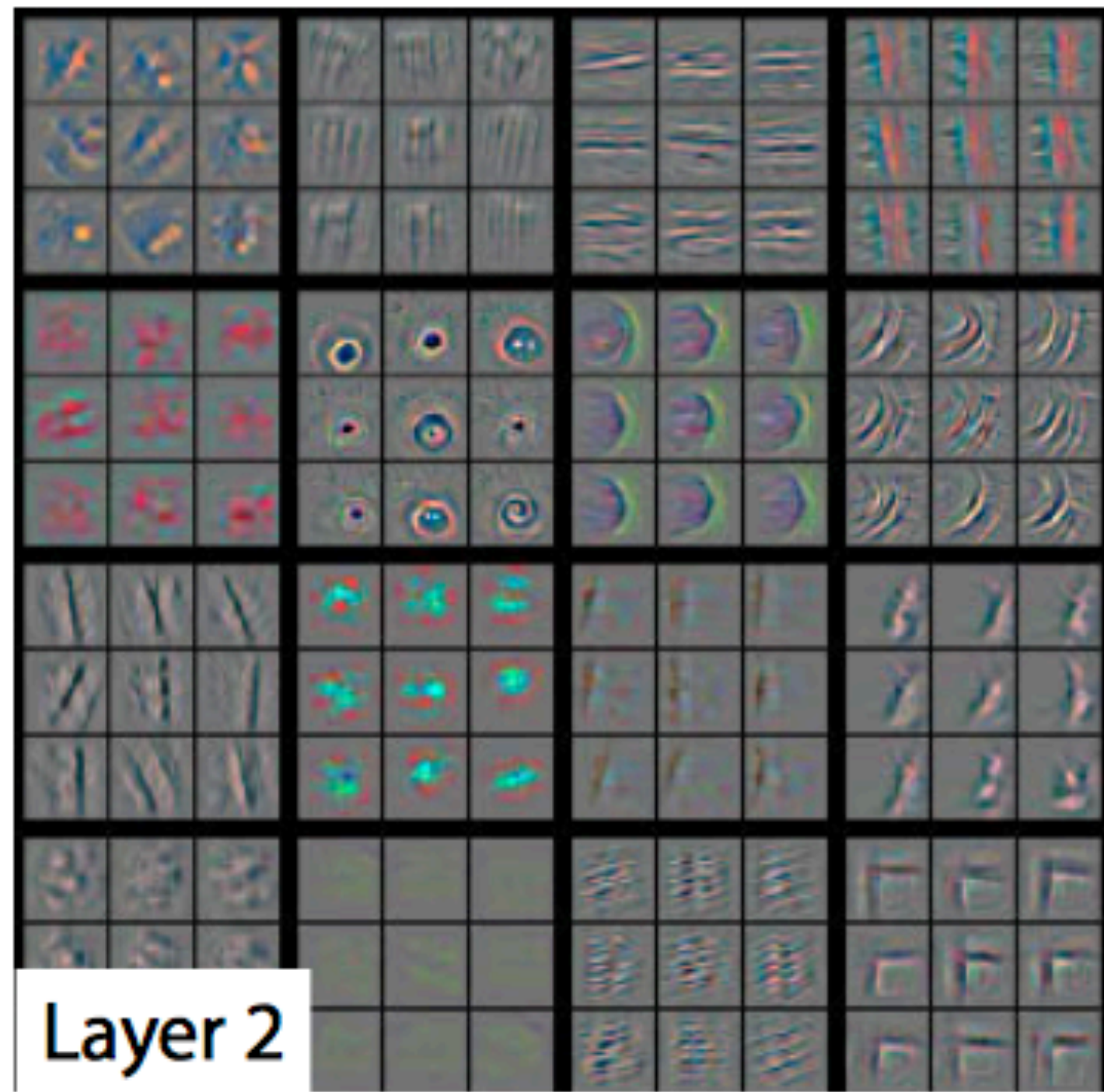
$$W_o = (W_i - F + 2P)/S + 1 \quad H_o = (H_i - F + 2P)/S + 1 \quad D_o = K$$

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

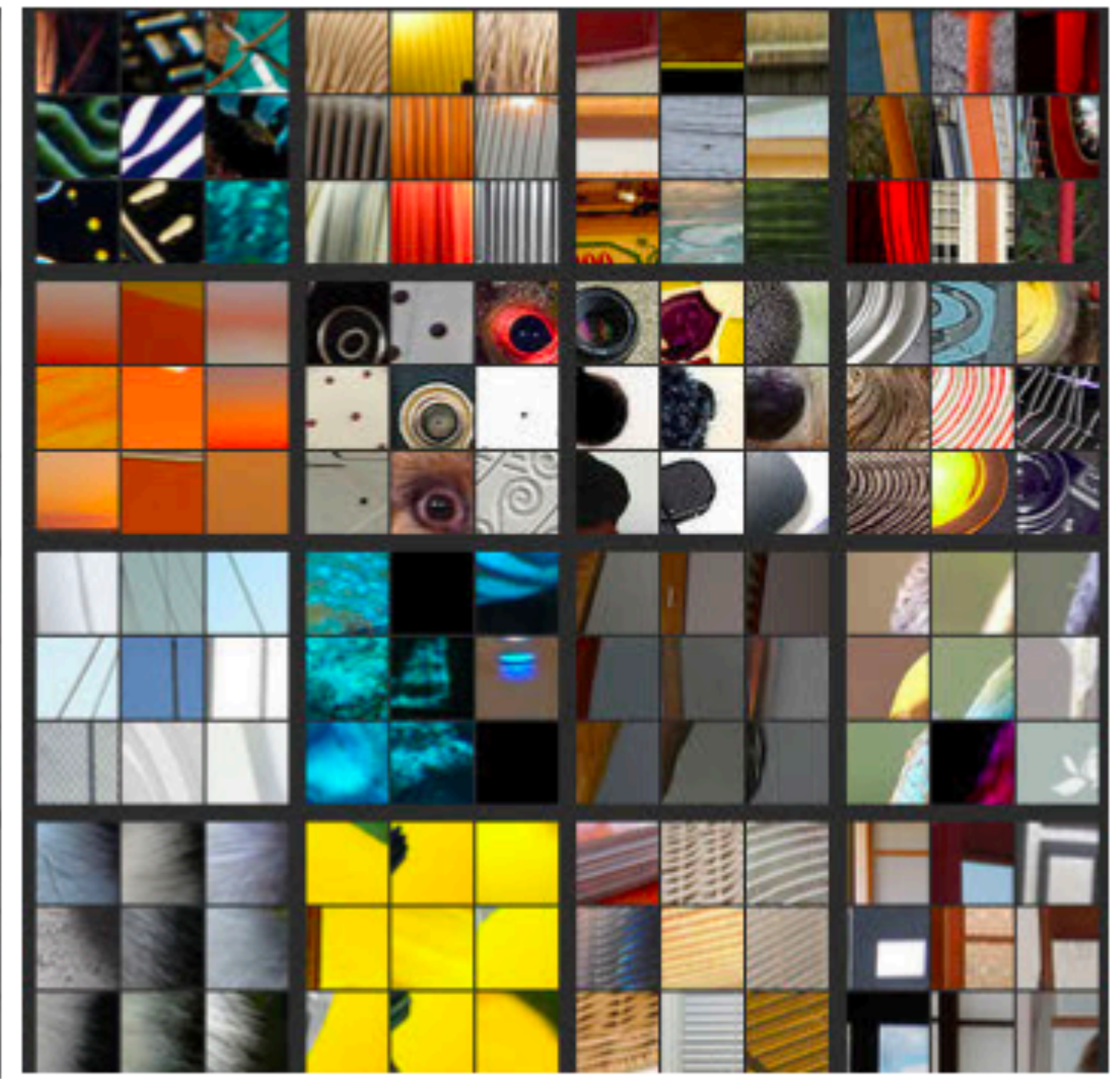
What **filters** do networks learn?



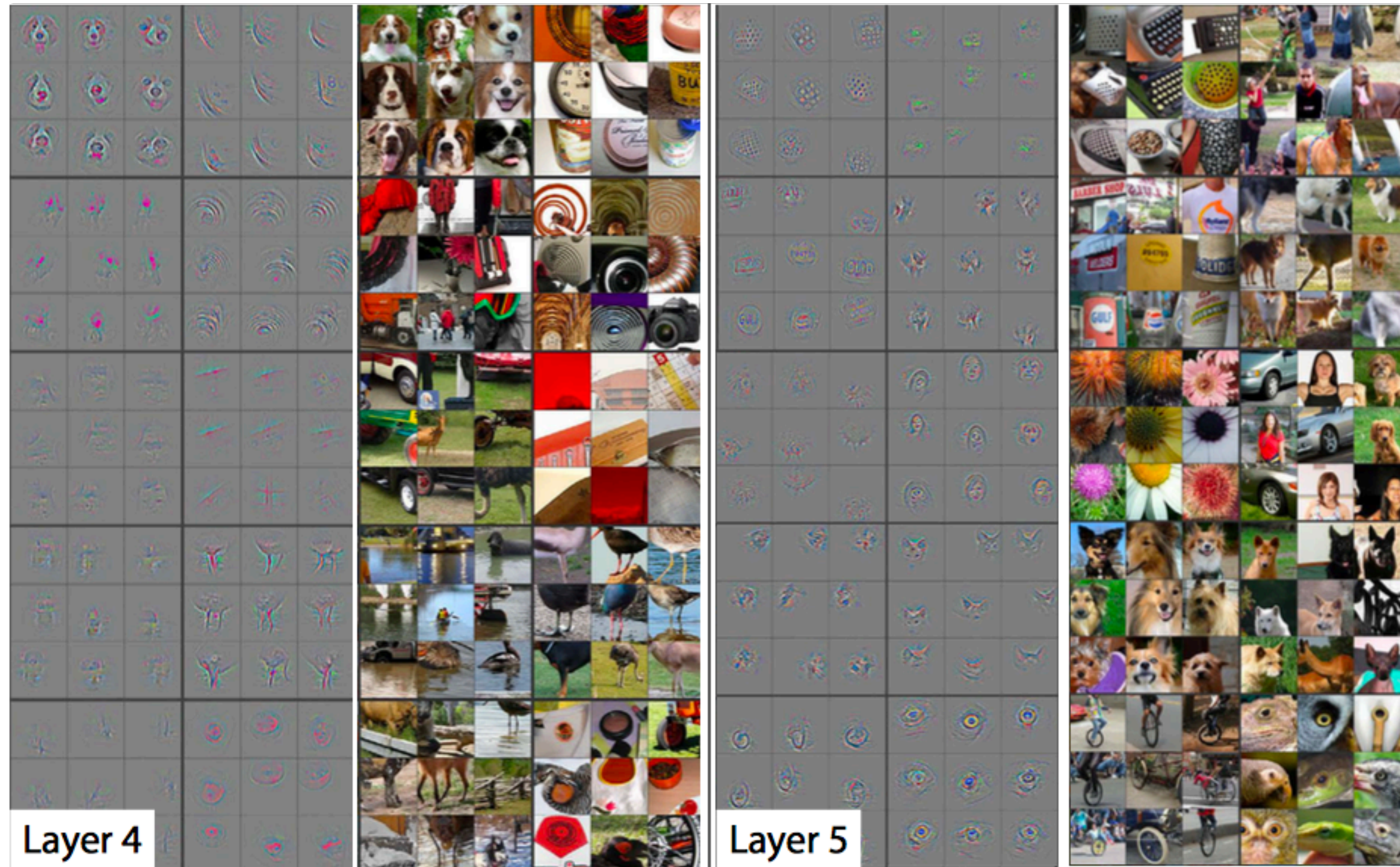
Layer 1



Layer 2



What **filters** do networks learn?

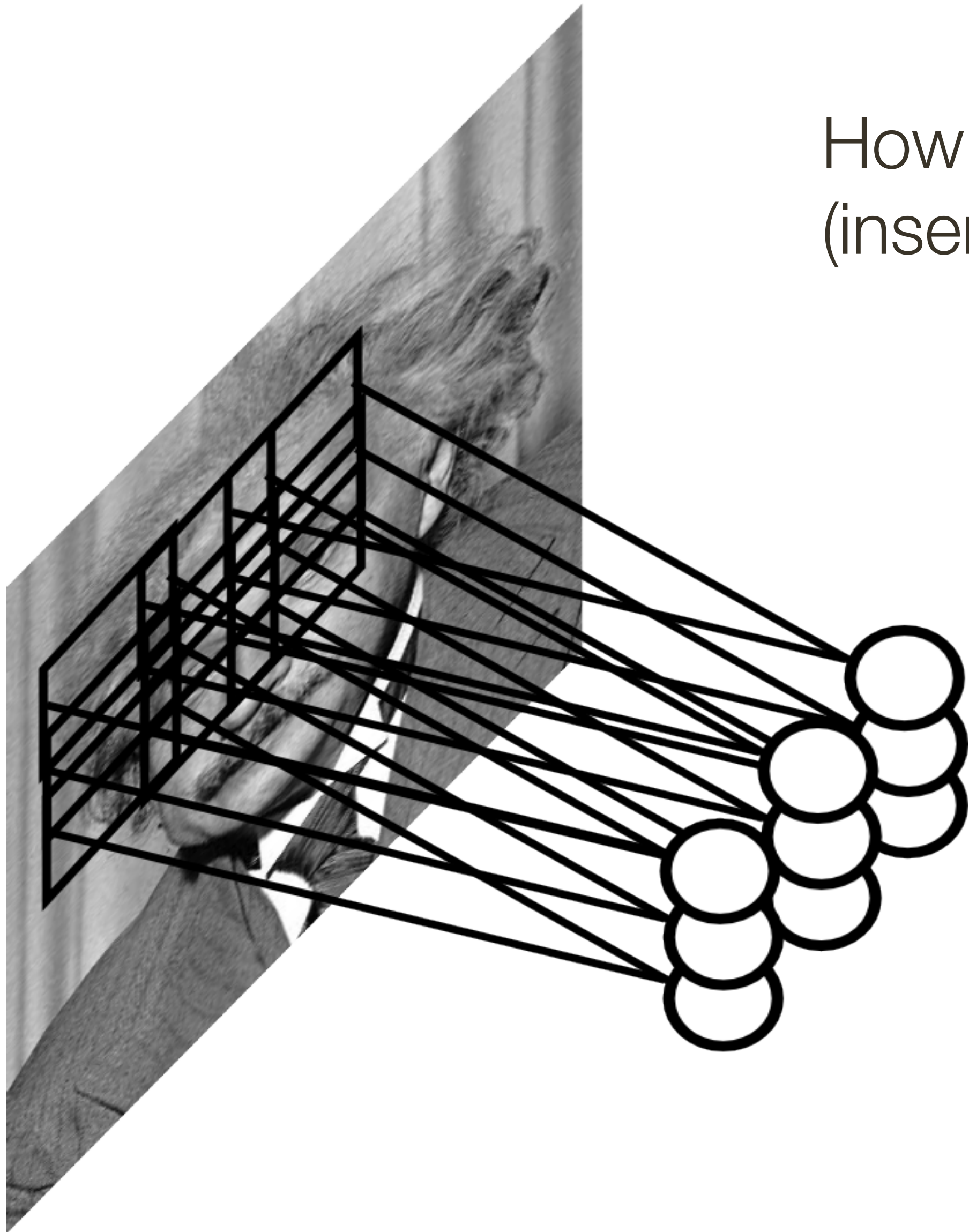


[Zeiler and Fergus, 2013]

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)

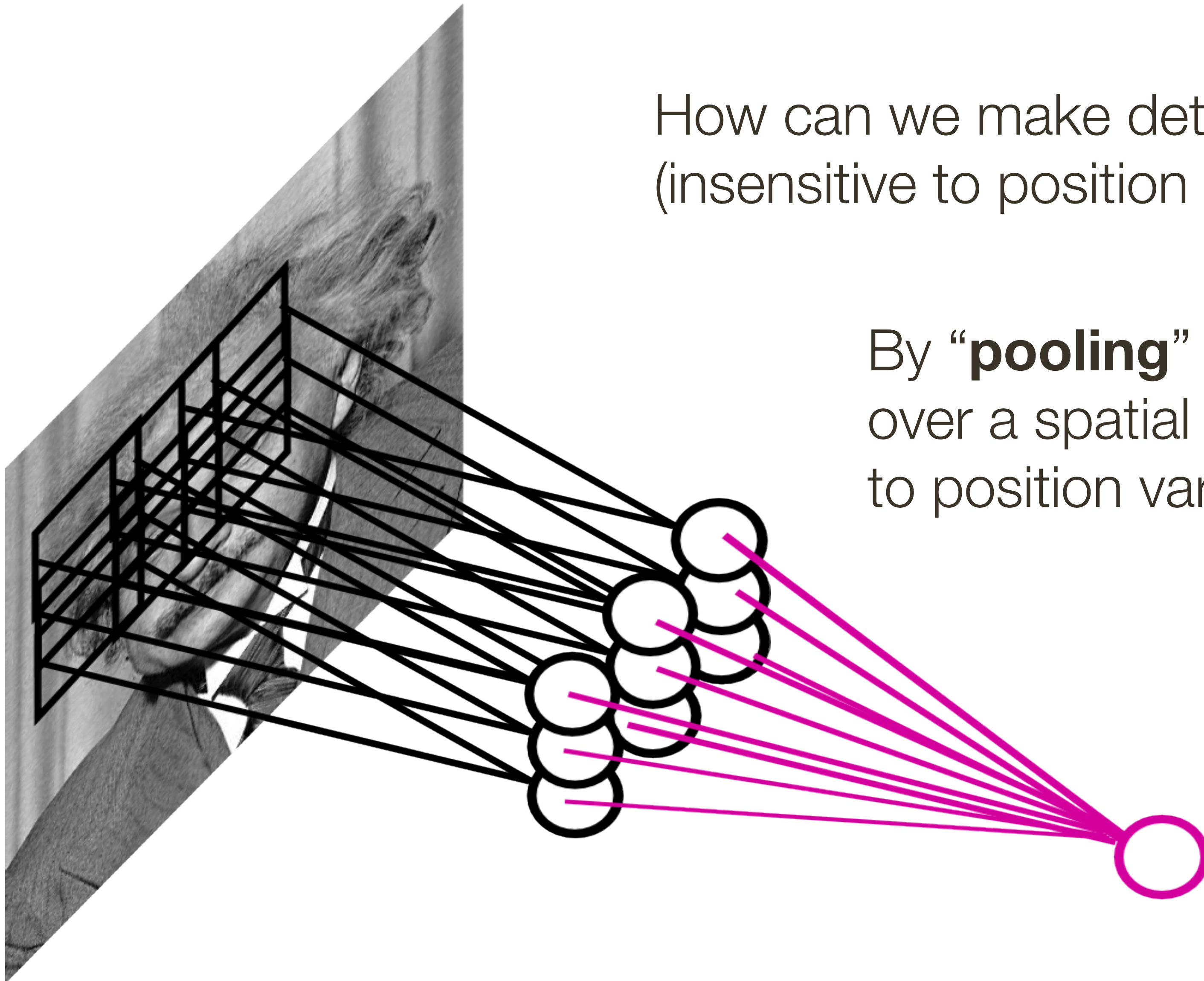


Pooling Layer

Let us assume the filter is an “eye” detector

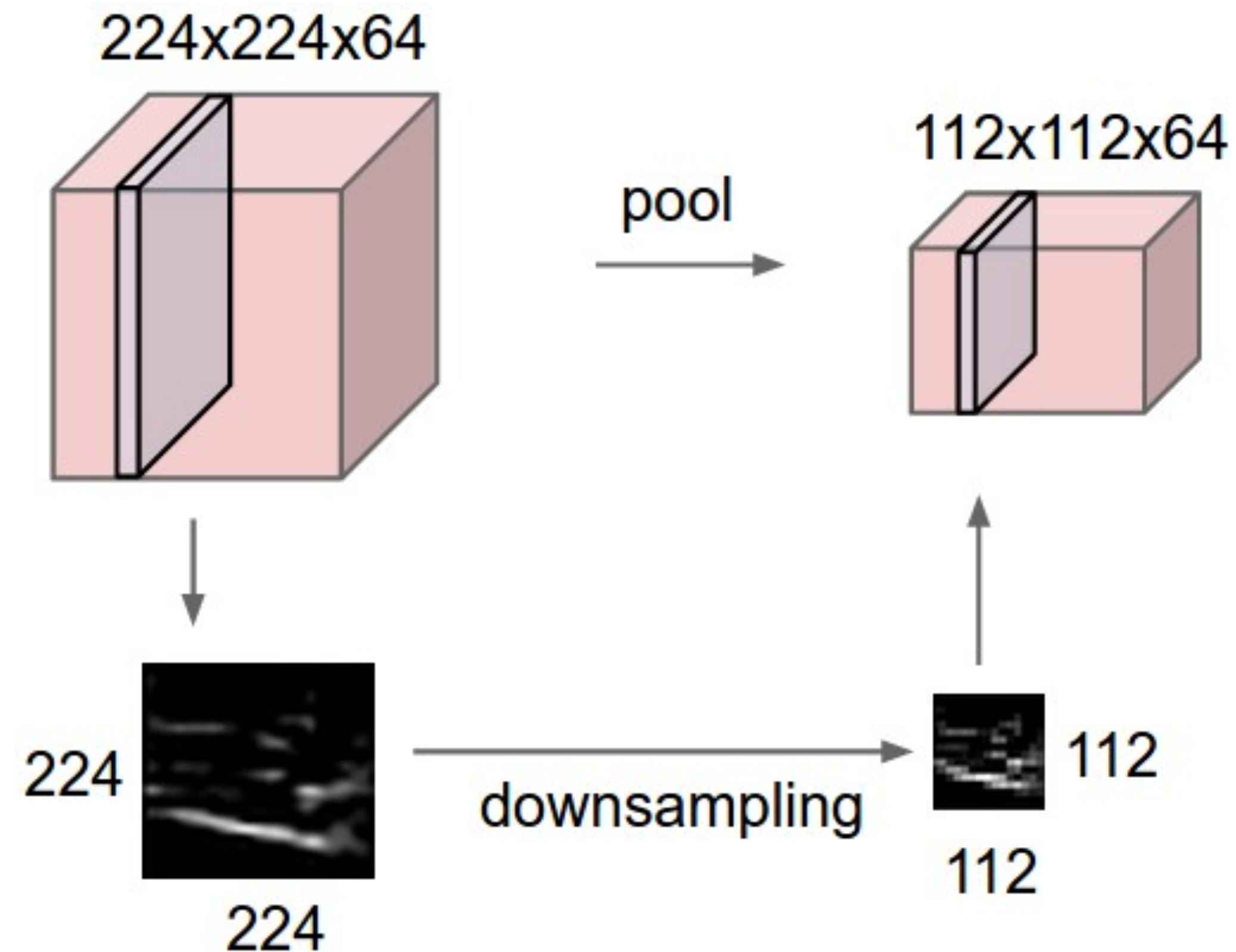
How can we make detection spatially invariant
(insensitive to position of the eye in the image)

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



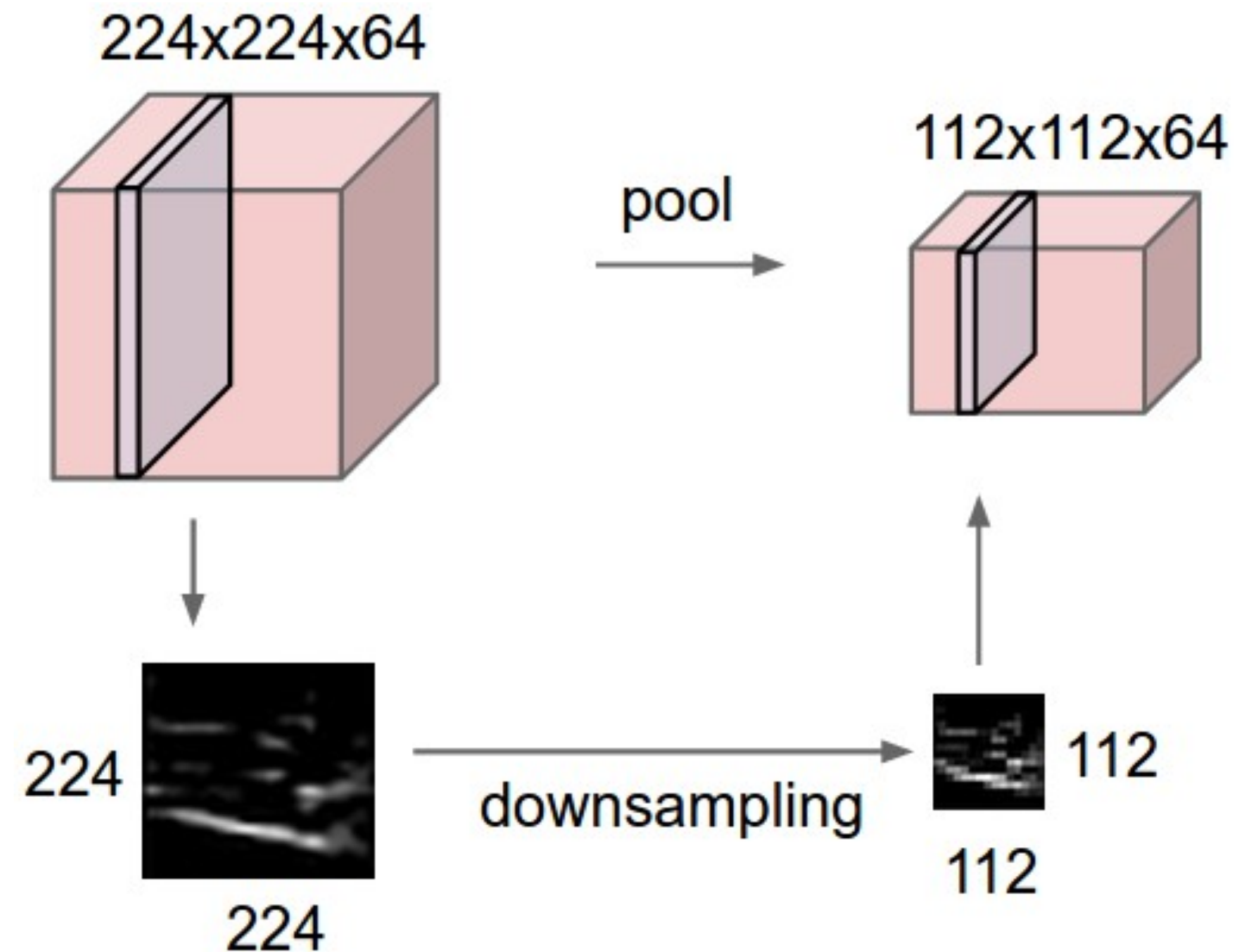
Pooling Layer

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



Pooling Layer

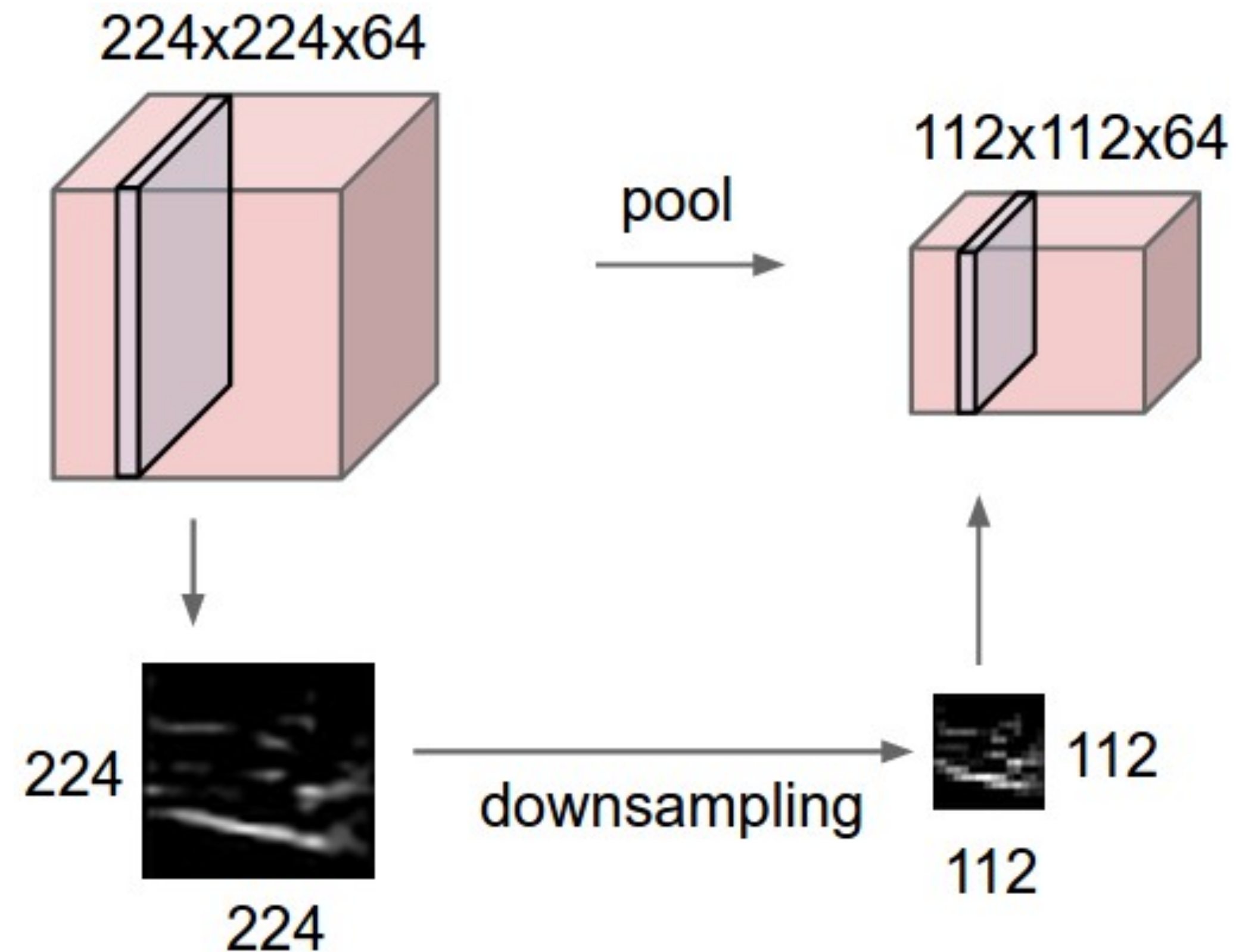
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



How many **parameters**?

Pooling Layer

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



How many **parameters**?

None!

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 **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: $W_o \times H_o \times D_o$

$$W_o = W_i / F$$

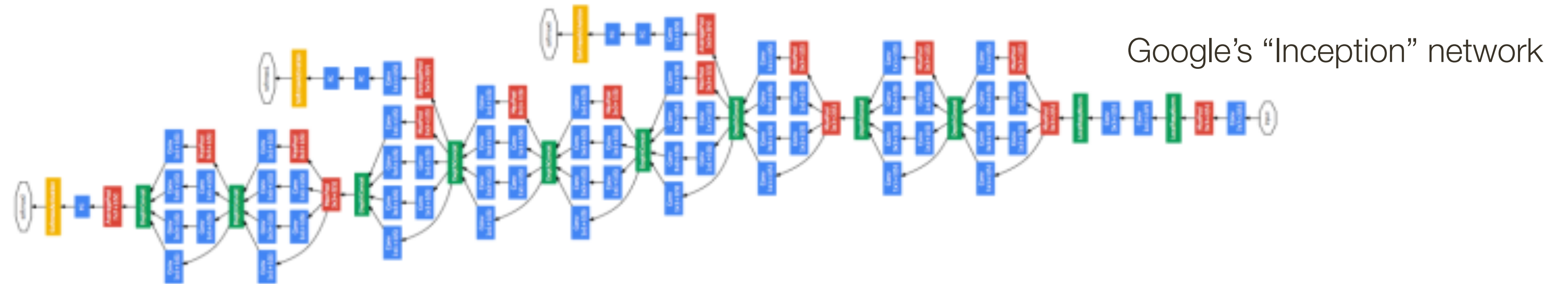
$$H_o = H_i / F$$

$$D_o = D_i$$

Number of total learnable parameters: 0

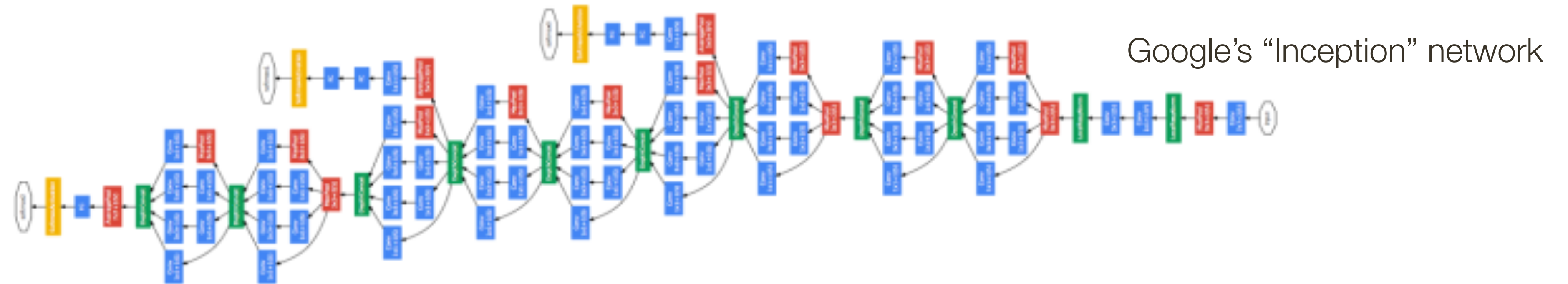
(you can do padding, but it's a bit trickier)

Deep Learning **Terminology**



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

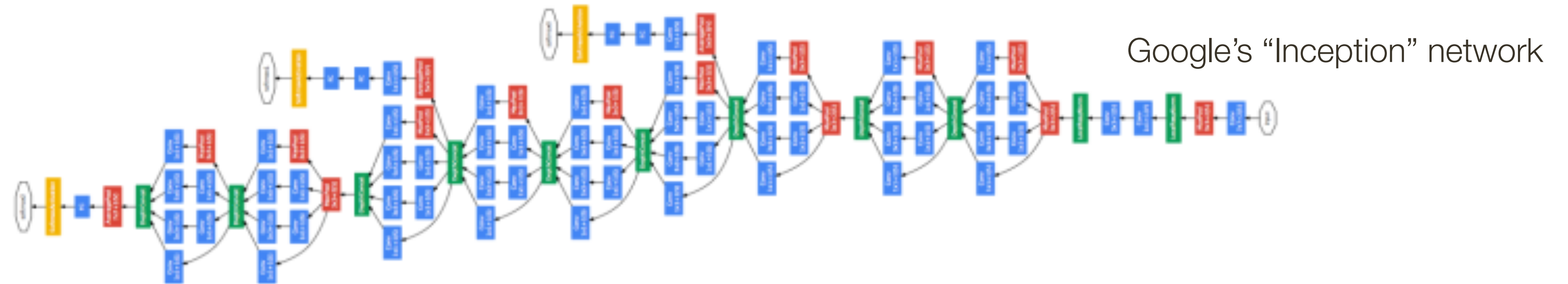
Deep Learning **Terminology**



- **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

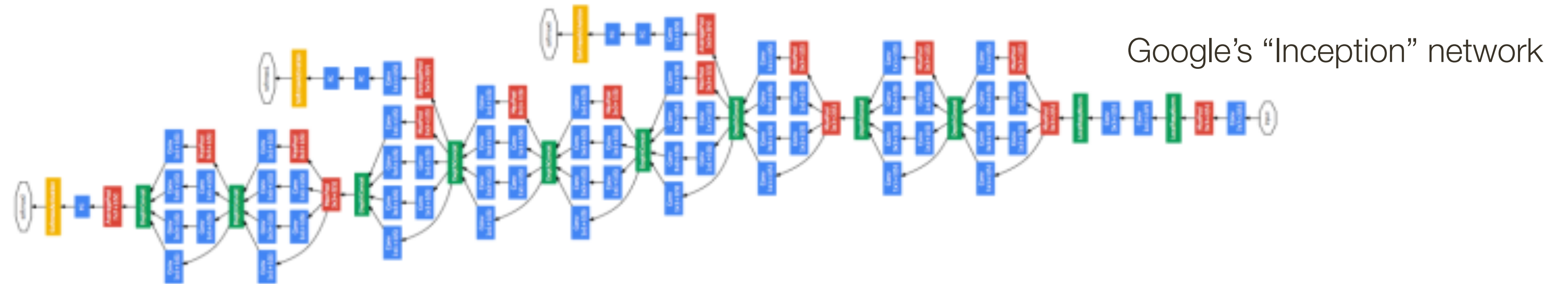


- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

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

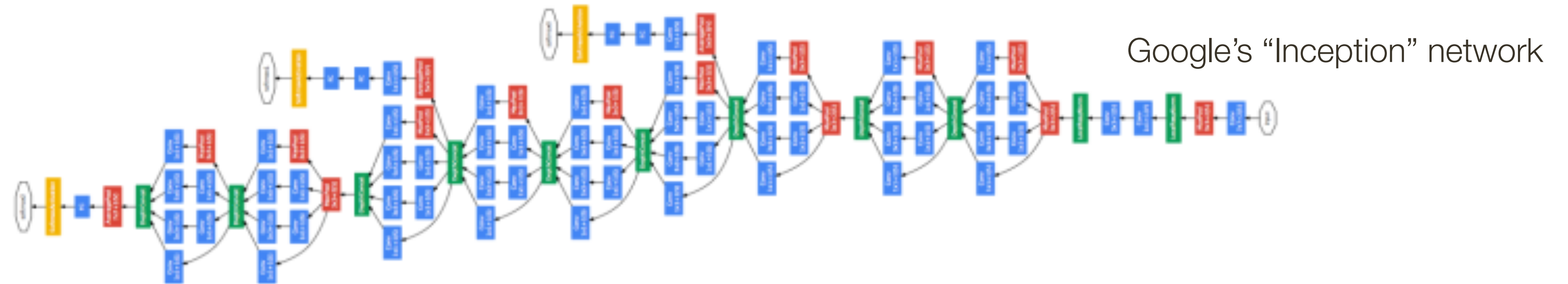
deeper = better

Deep Learning Terminology



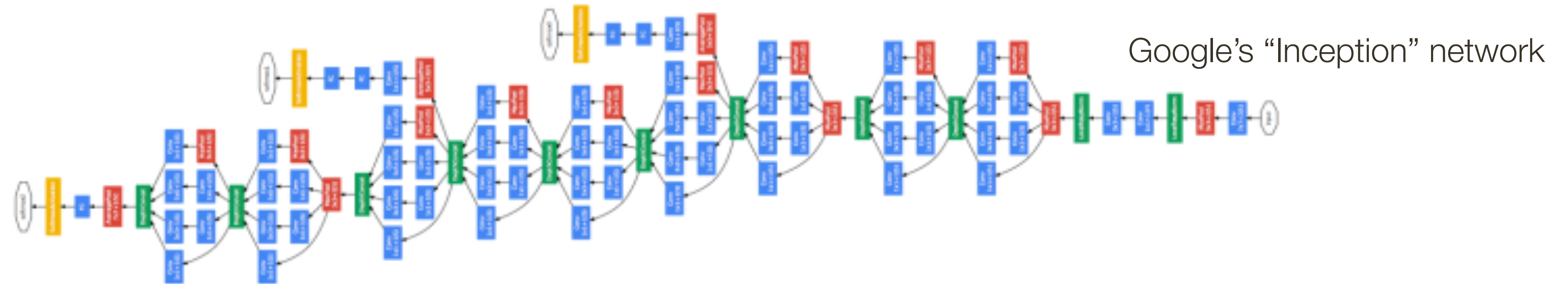
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
 - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
 - deeper = better
- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

Deep Learning Terminology



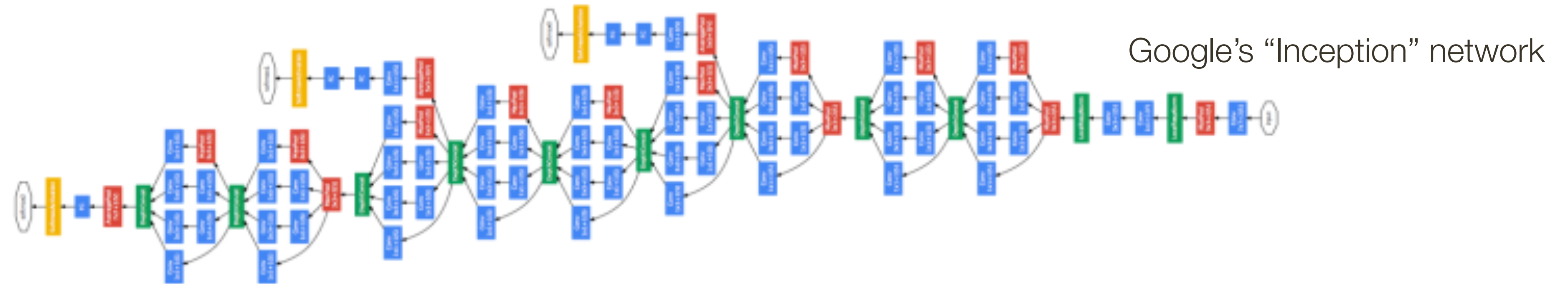
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
 - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
 - deeper = better
- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)
 - requires knowledge of the nature of the problem

Deep Learning Terminology



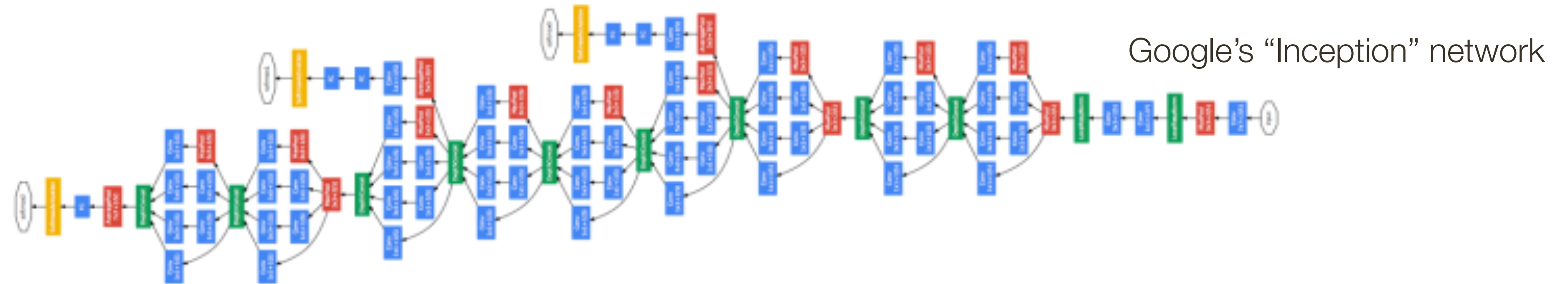
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
 - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
 - deeper = better
- **Loss function:** objective function 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



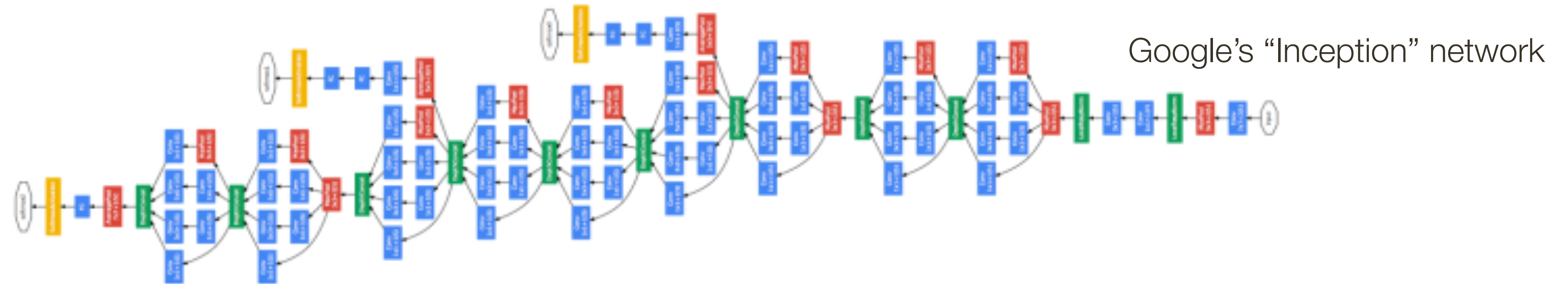
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
 - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
 - deeper = better
- **Loss function:** objective function 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



- **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



- **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

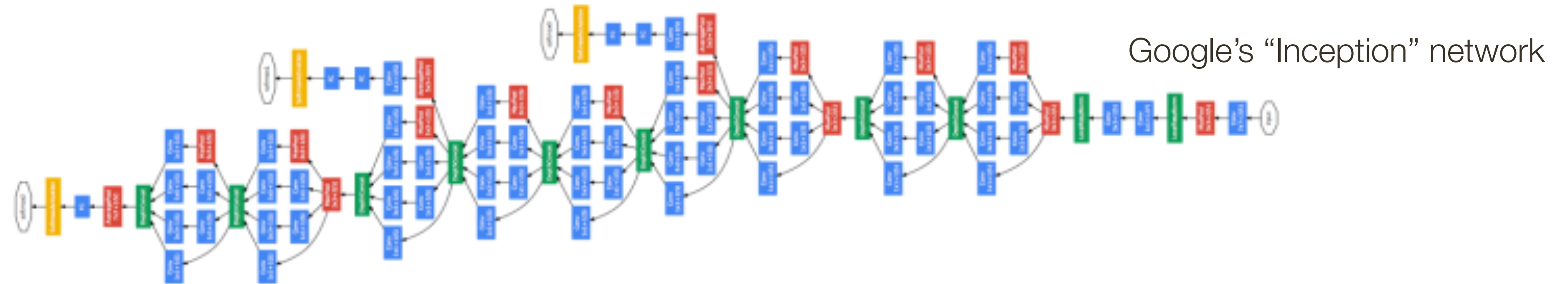
- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

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

Deep Learning Terminology



- **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

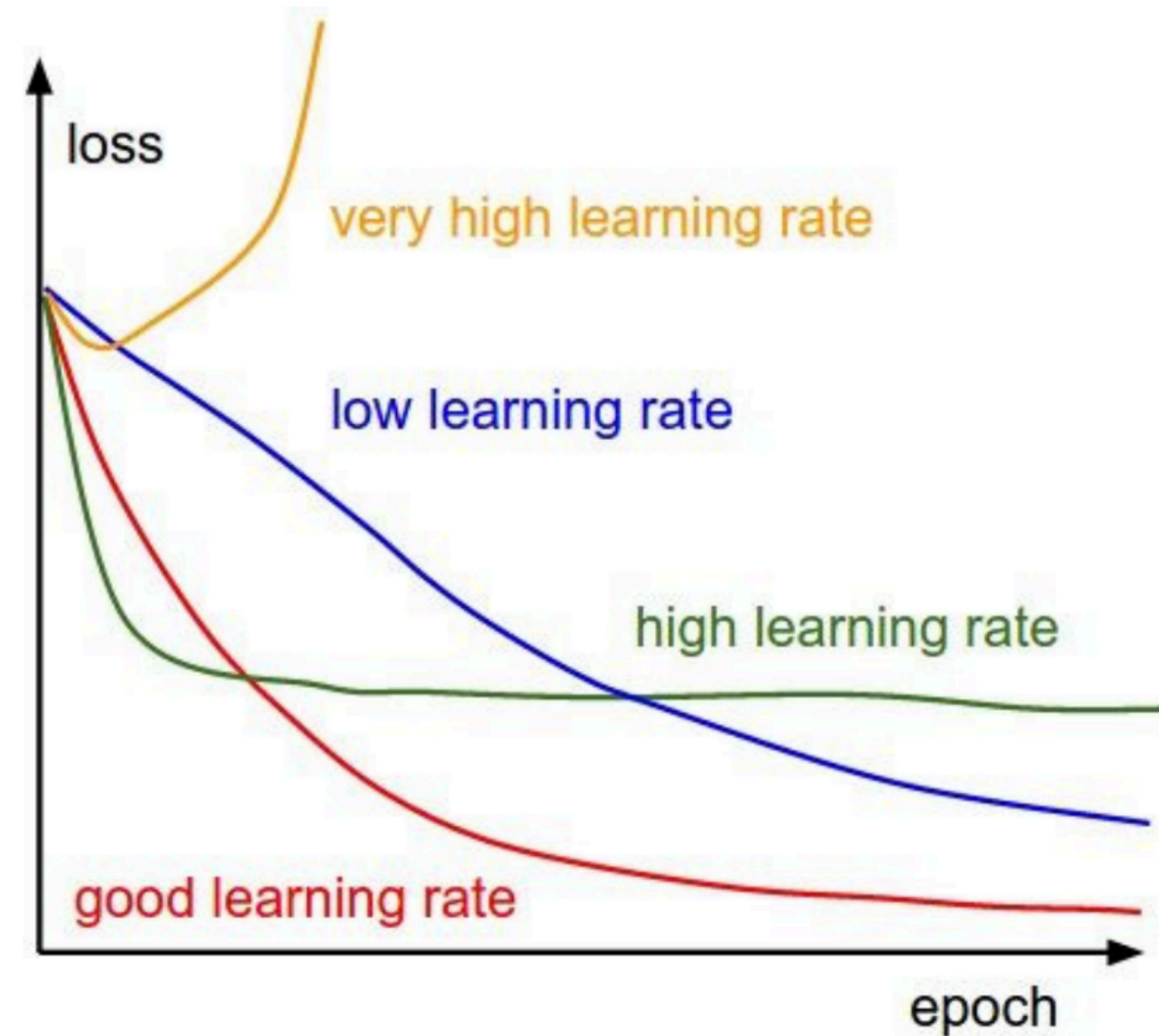
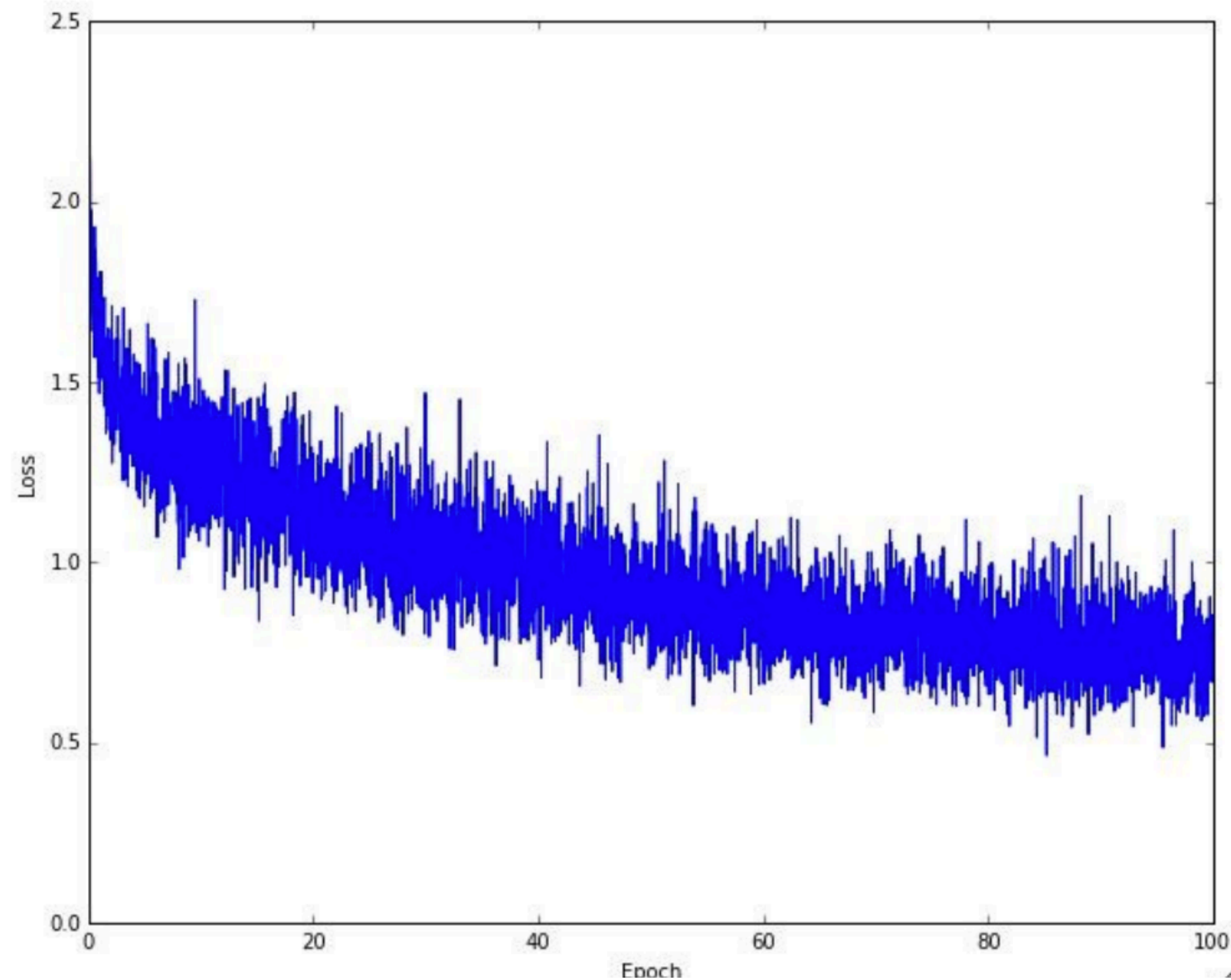
For a fixed number of iterations or until convergence

- 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)

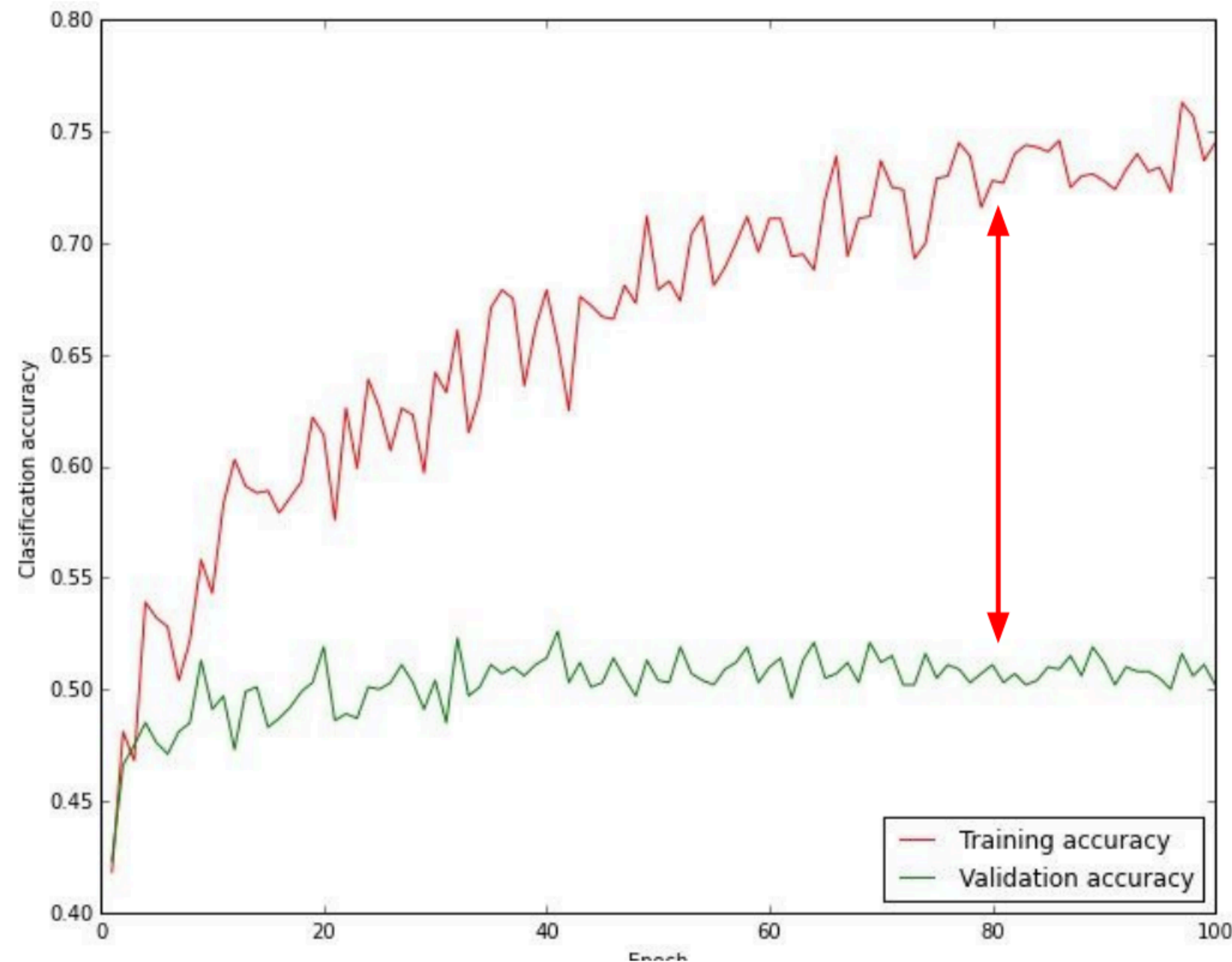
Inference / Prediction

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

Monitoring Learning: Visualizing the (training) loss



Monitoring Learning: Visualizing the (training) loss



Big gap = overfitting

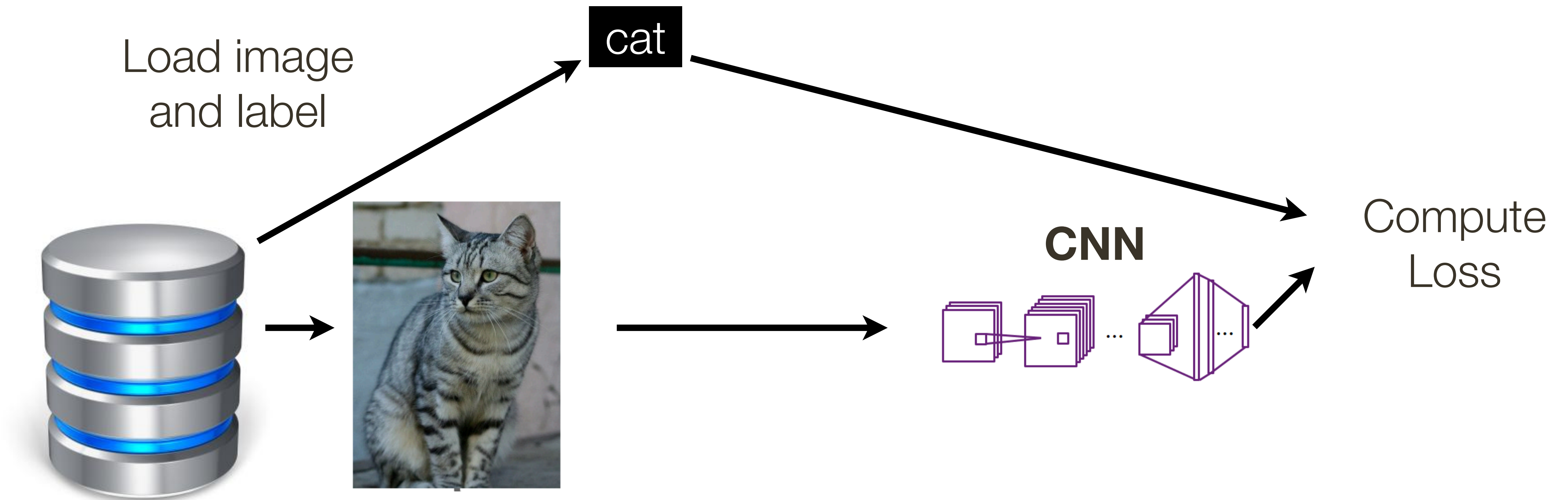
Solution: increase regularization

No gap = undercutting

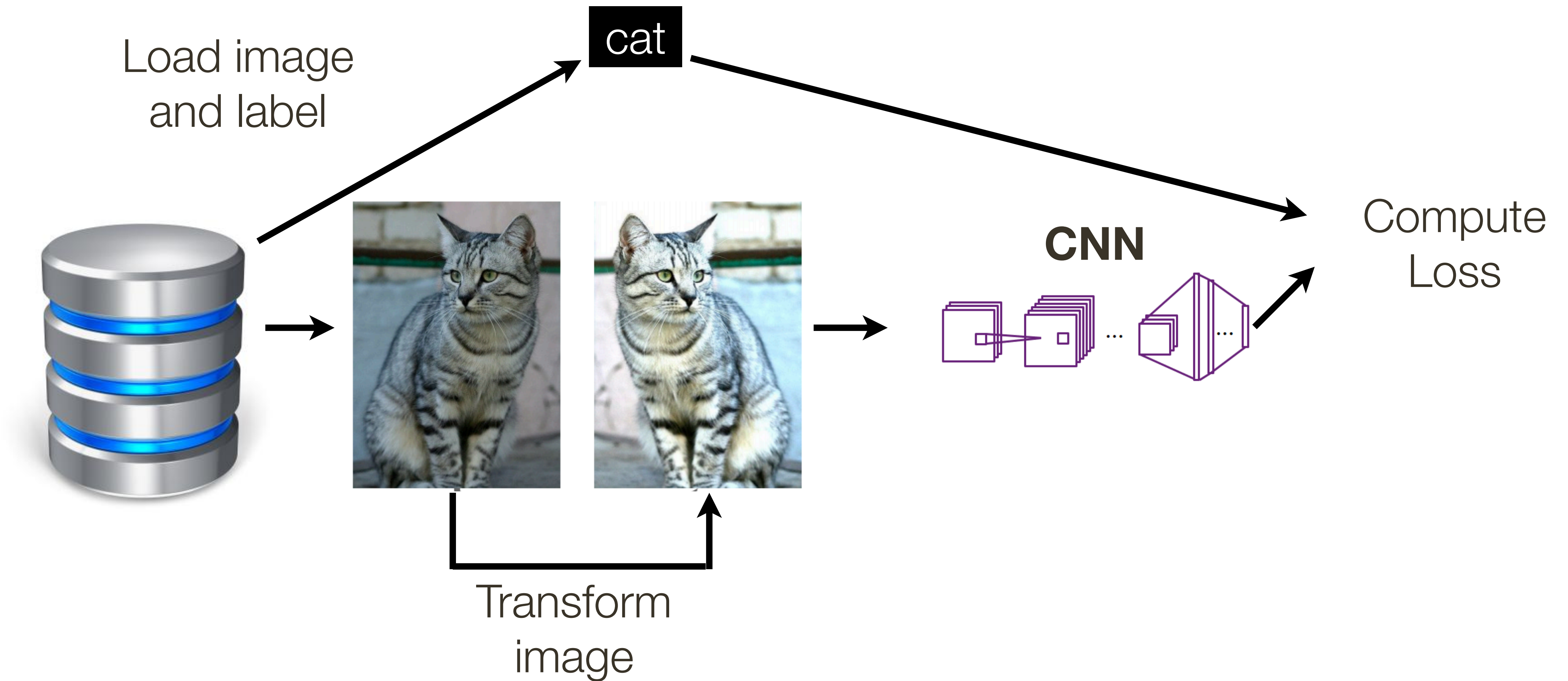
Solution: increase model capacity

Small gap = **ideal**

Regularization: Data Augmentation



Regularization: Data Augmentation



Regularization: Data Augmentation

Horizontal flips

Random crops & scales

Color Jitter

Regularization: Data Augmentation

Horizontal flips

Random crops & scales

Color Jitter



Regularization: Data Augmentation

Horizontal flips

Random crops & scales

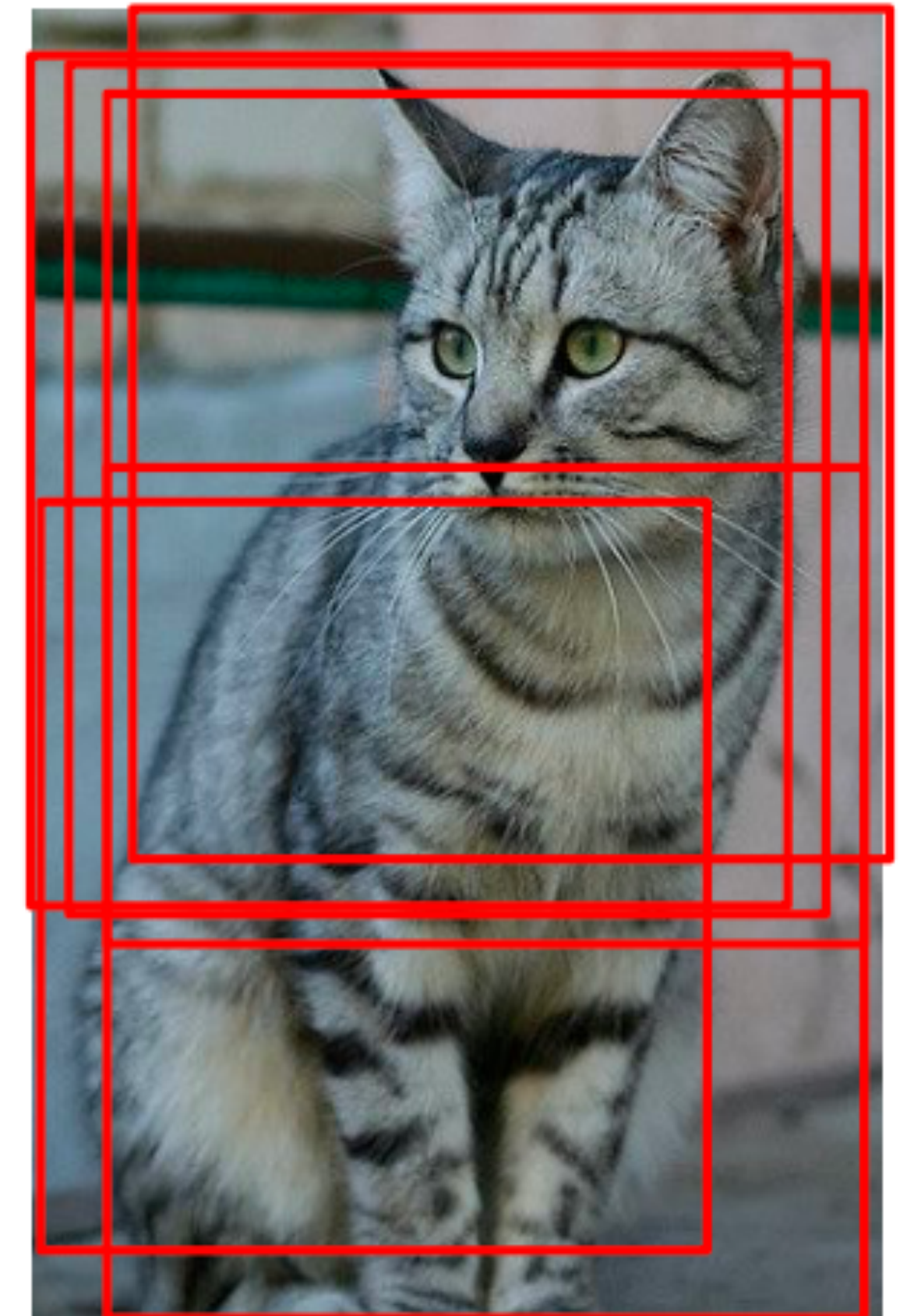
Color Jitter

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

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

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

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



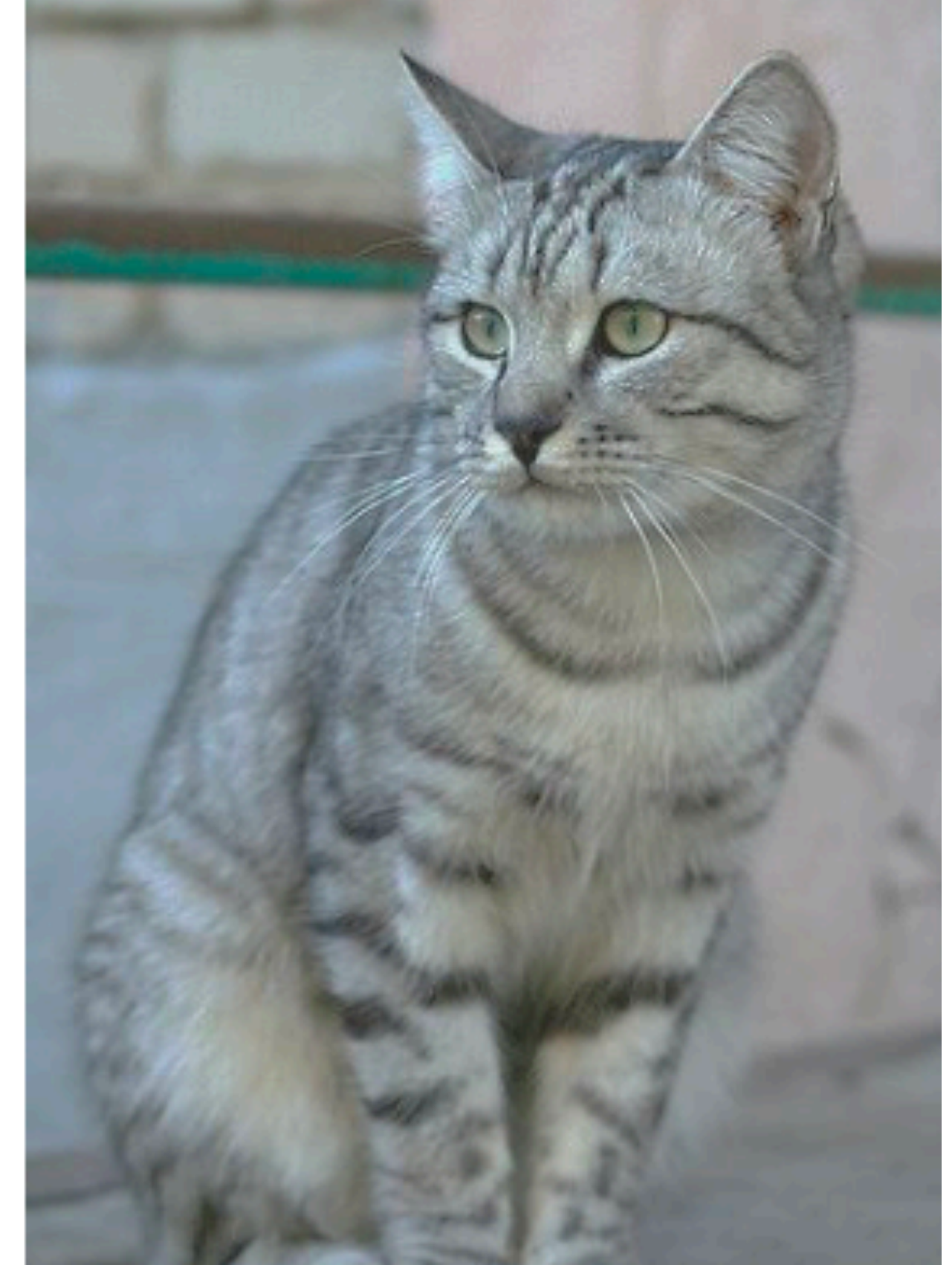
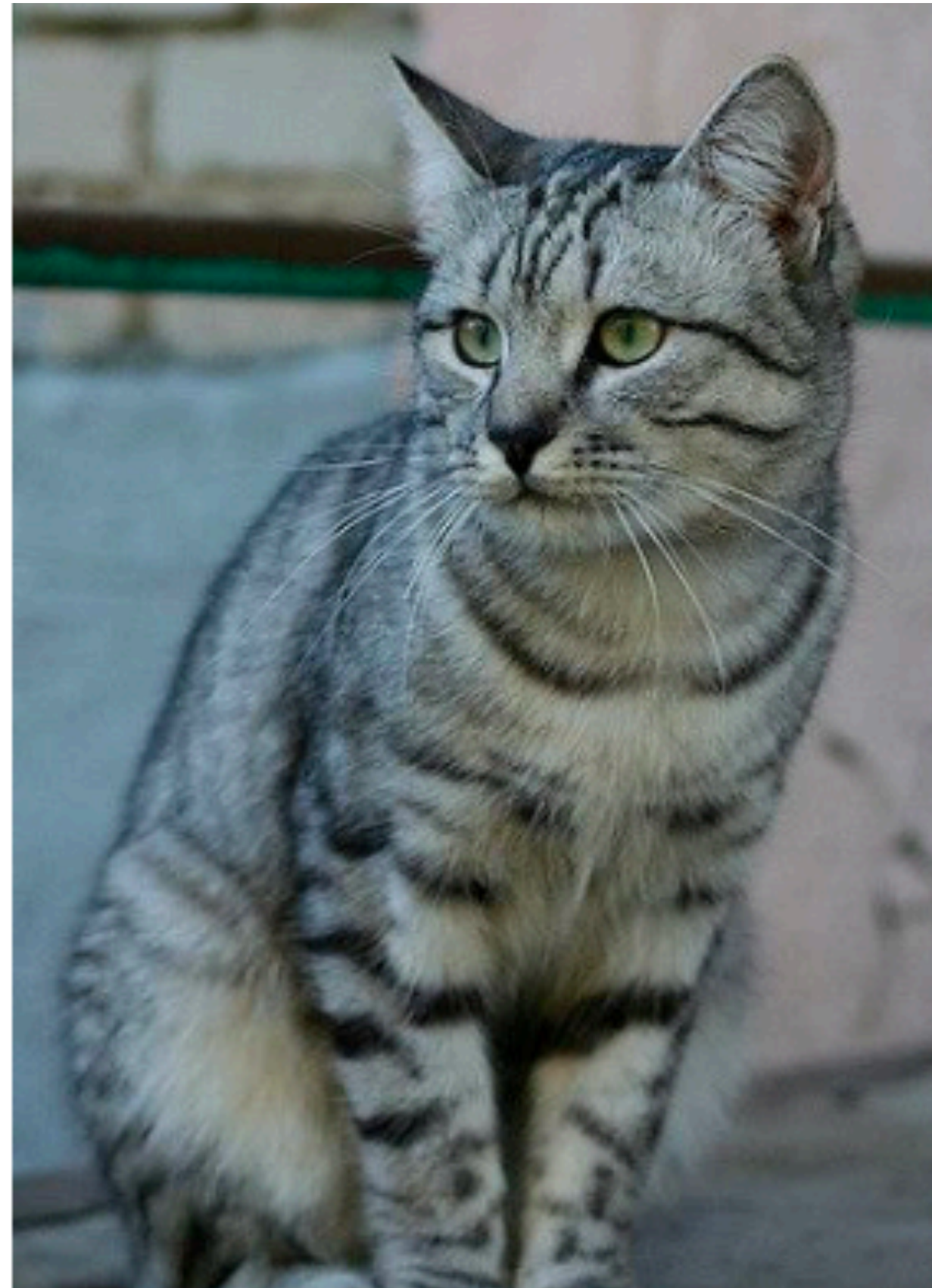
Regularization: Data Augmentation

Horizontal flips

Random crops & scales

Color Jitter

Random perturbations in
contrast and brightness





Computer **Vision Problems**

Computer **Vision Problems**

Categorization



Computer **V**ision **P**roblems

Categorization



Multi-**class**: Horse
Church
Toothbrush
Person

IM  GENET

Computer **V**ision **P**roblems

Categorization



Multi-**class**: Horse
Church
Toothbrush
Person

IMGENET

Multi-**label**: **Horse**
Church
Toothbrush
Person

Computer **V**ision **P**roblems

Categorization

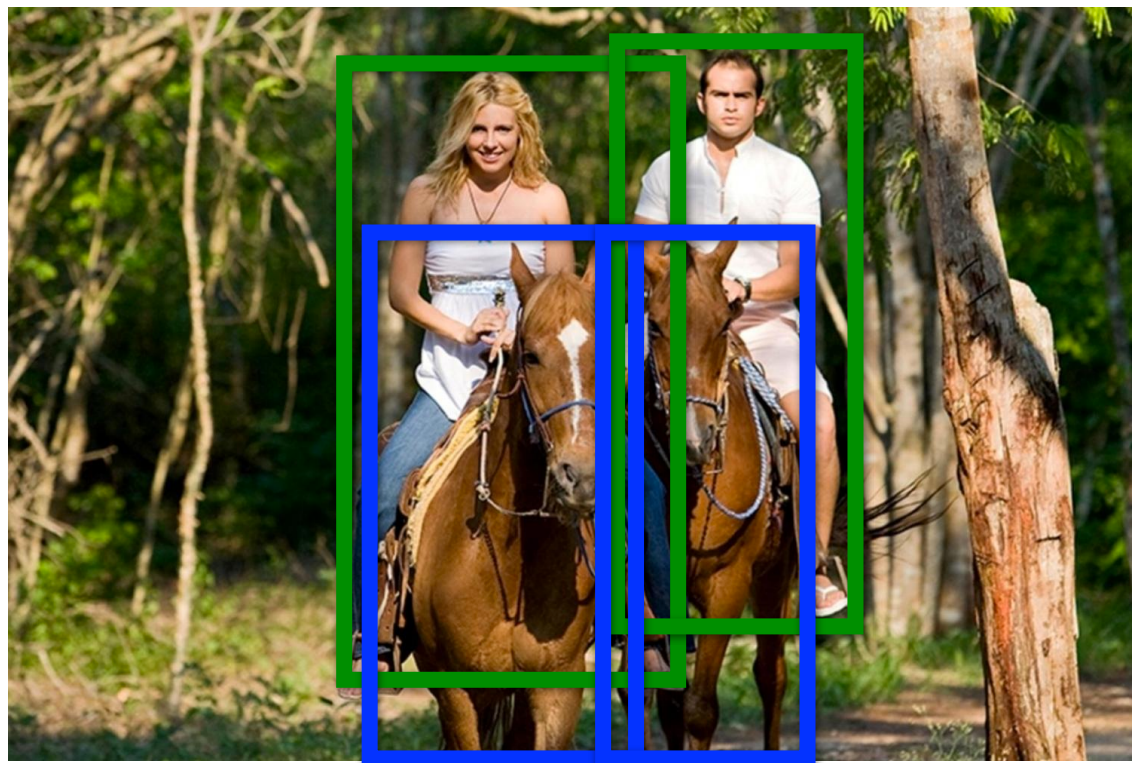


Multi-**class**: Horse
Church
Toothbrush
Person



Multi-**label**: **Horse**
Church
Toothbrush
Person

Detection



Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)



Computer **V**ision **P**roblems

Categorization

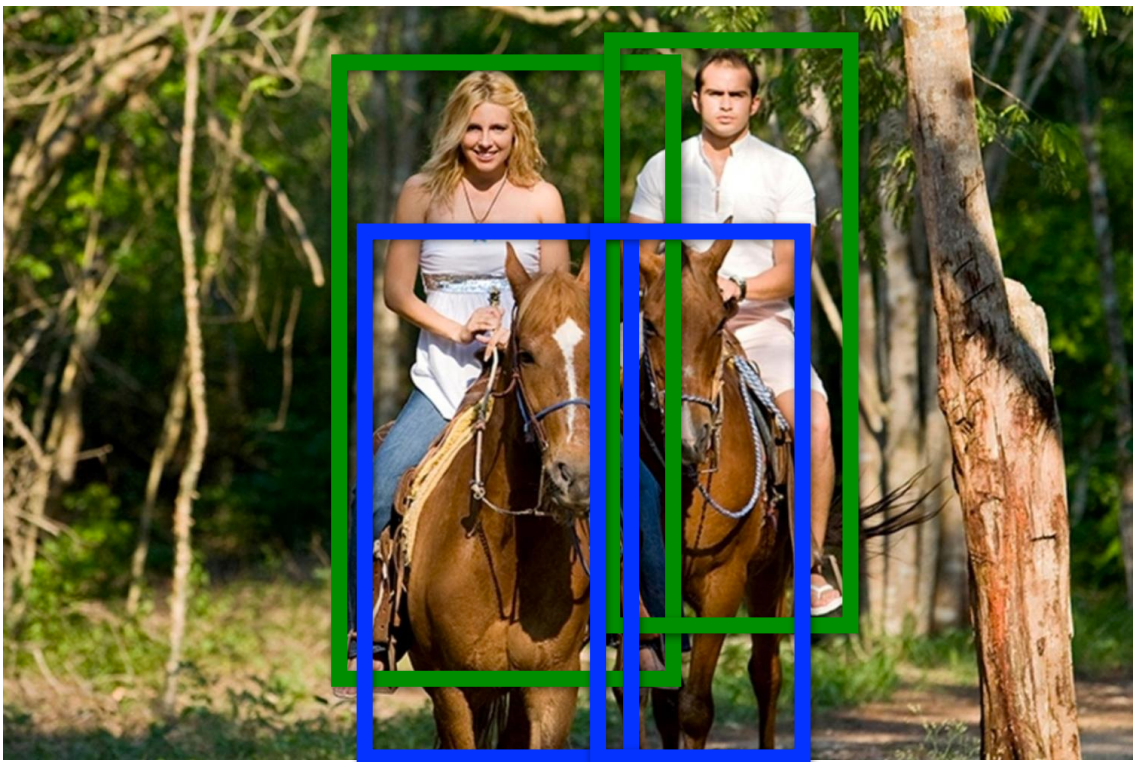


Multi-**class**: Horse
Church
Toothbrush
Person

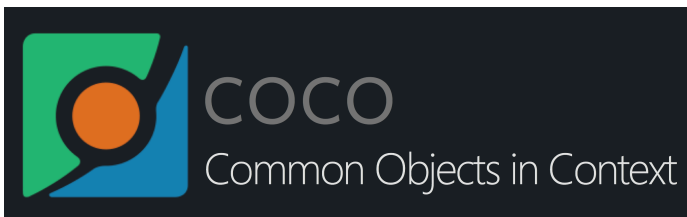


Multi-**label**: **Horse**
Church
Toothbrush
Person

Detection



Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)



Segmentation



Horse
Person



Computer **V**ision **P**roblems

Categorization

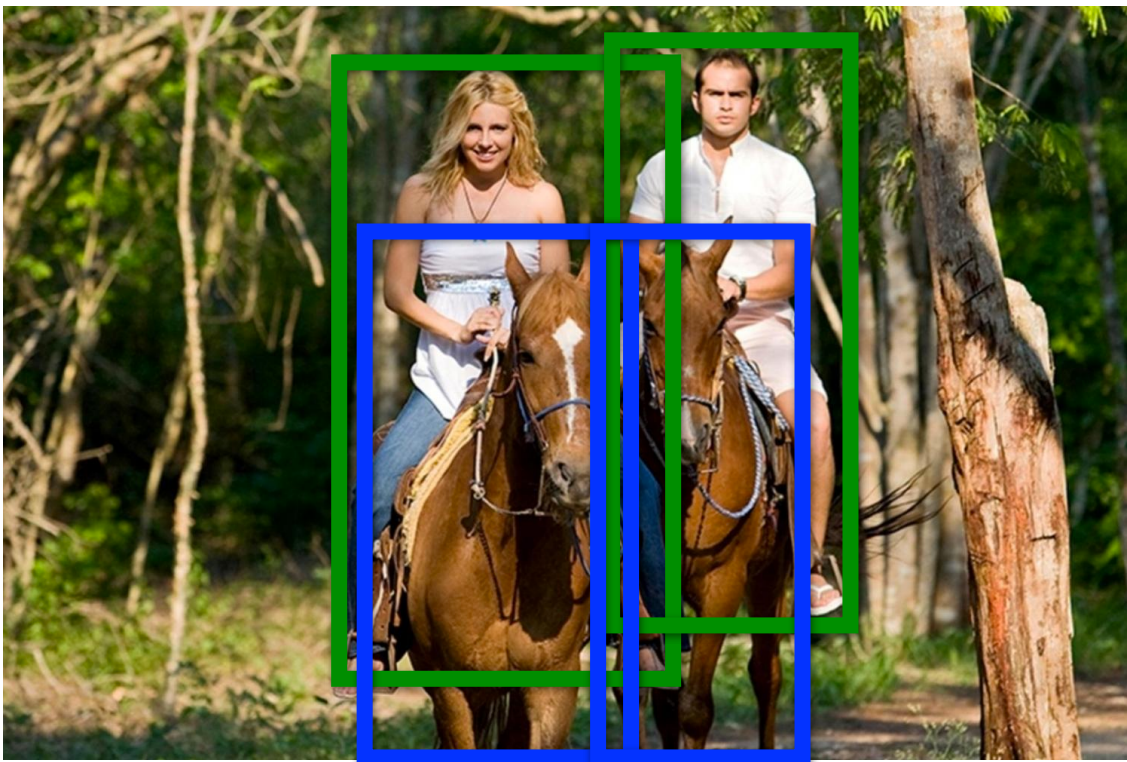


Multi-**class**: Horse
Church
Toothbrush
Person

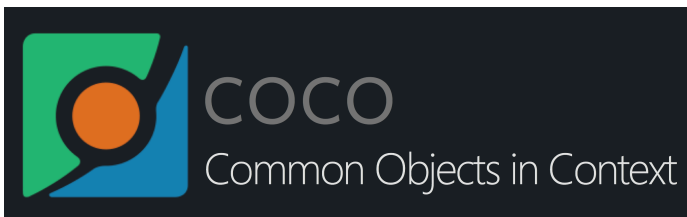


Multi-**label**: **Horse**
Church
Toothbrush
Person

Detection



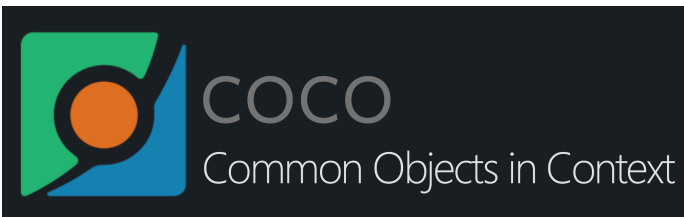
Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)



Segmentation



Horse
Person



Instance Segmentation



Horse1
Horse2
Person1
Person2

Computer **V**ision **P**roblems

Categorization



Multi-**class**: Horse
Church
Toothbrush
Person

IMGENET

Multi-**label**: **Horse**
Church
Toothbrush
Person

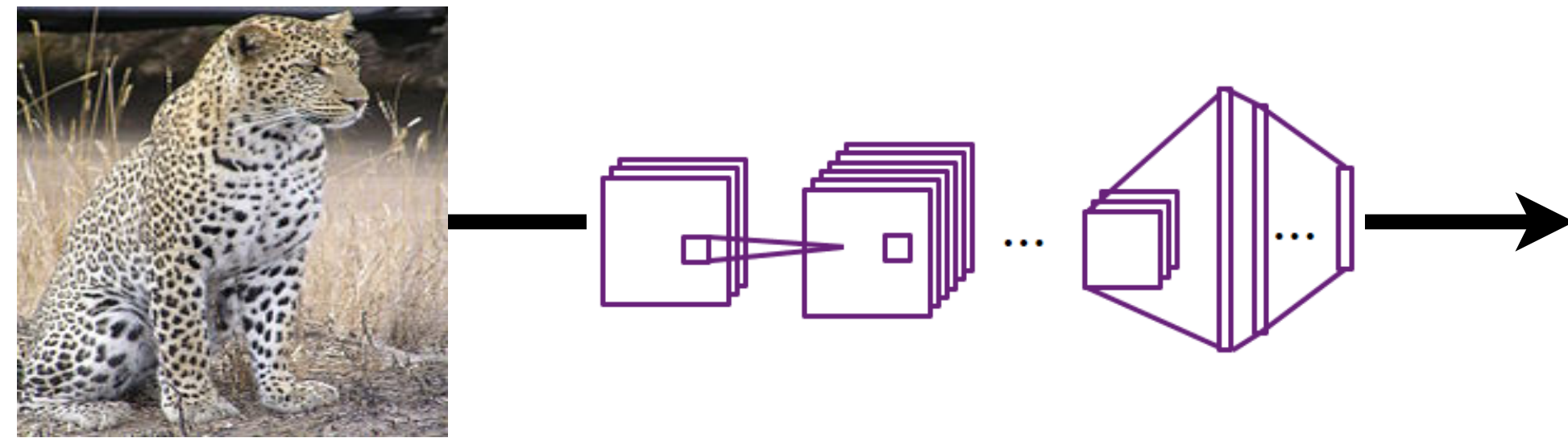
Object **Classification**



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

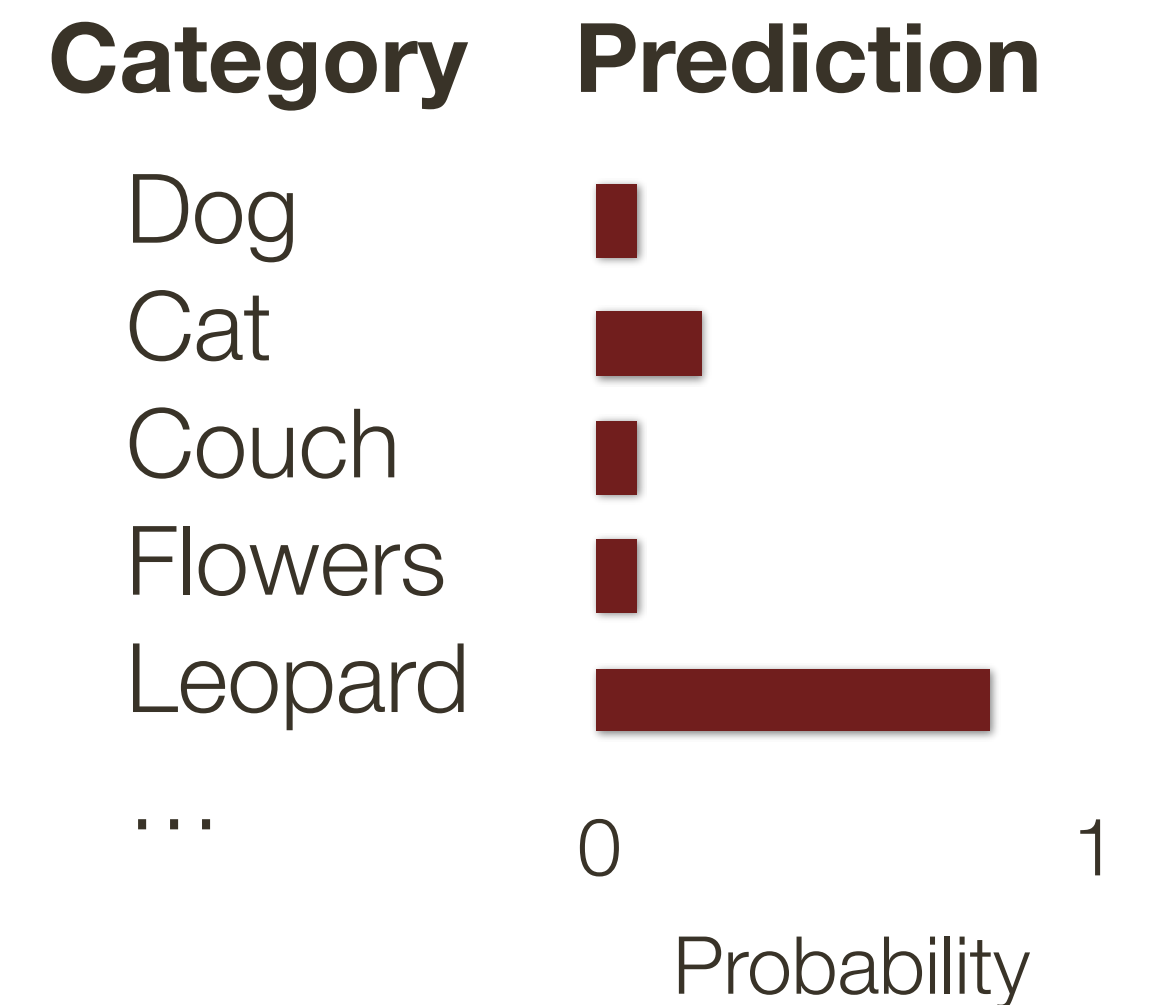
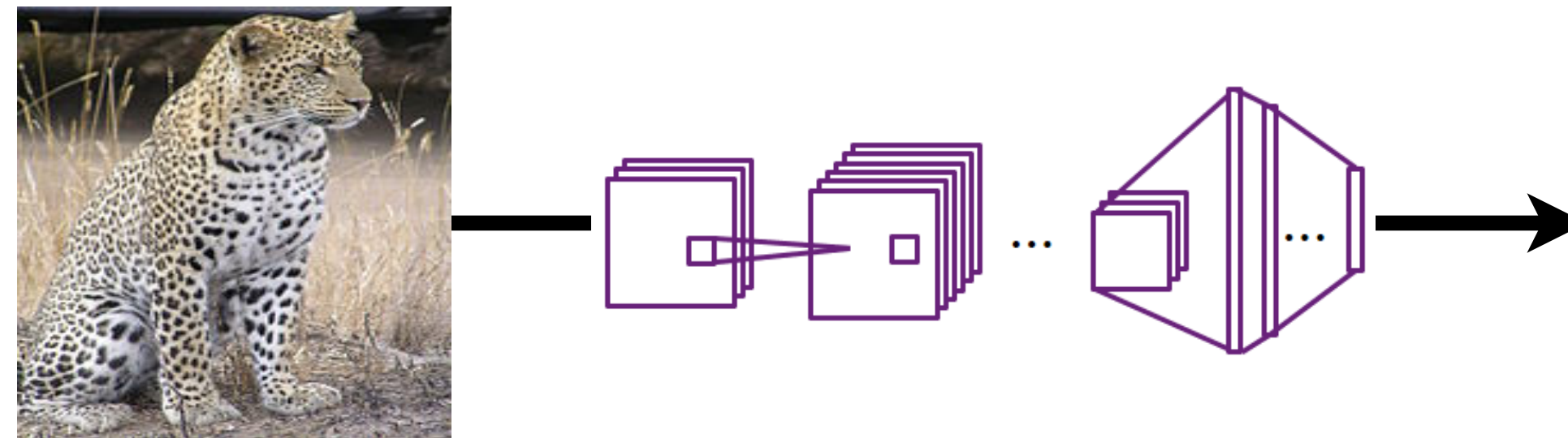
Object **Classification**



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

ImageNet Dataset

Over **14 million** (high resolution) web **images**

Roughly labeled with **22K synset** categories

Labeled on Amazon Mechanical Turk (AMT)

Popular Synsets

Animal

fish
bird
mammal
invertebrate

Plant

tree
flower
vegetable

Activity

sport

Material

fabric

Instrumentation

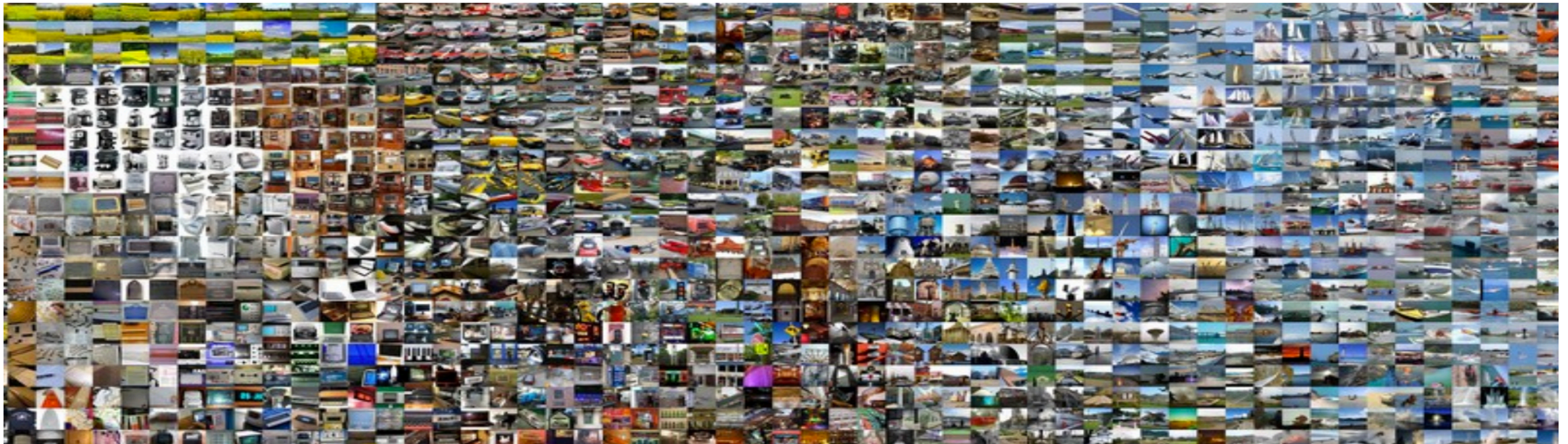
utensil
appliance
tool
musical instrument

Scene

room
geological formation

Food

beverage



ImageNet **Competition** (ILSVRC)

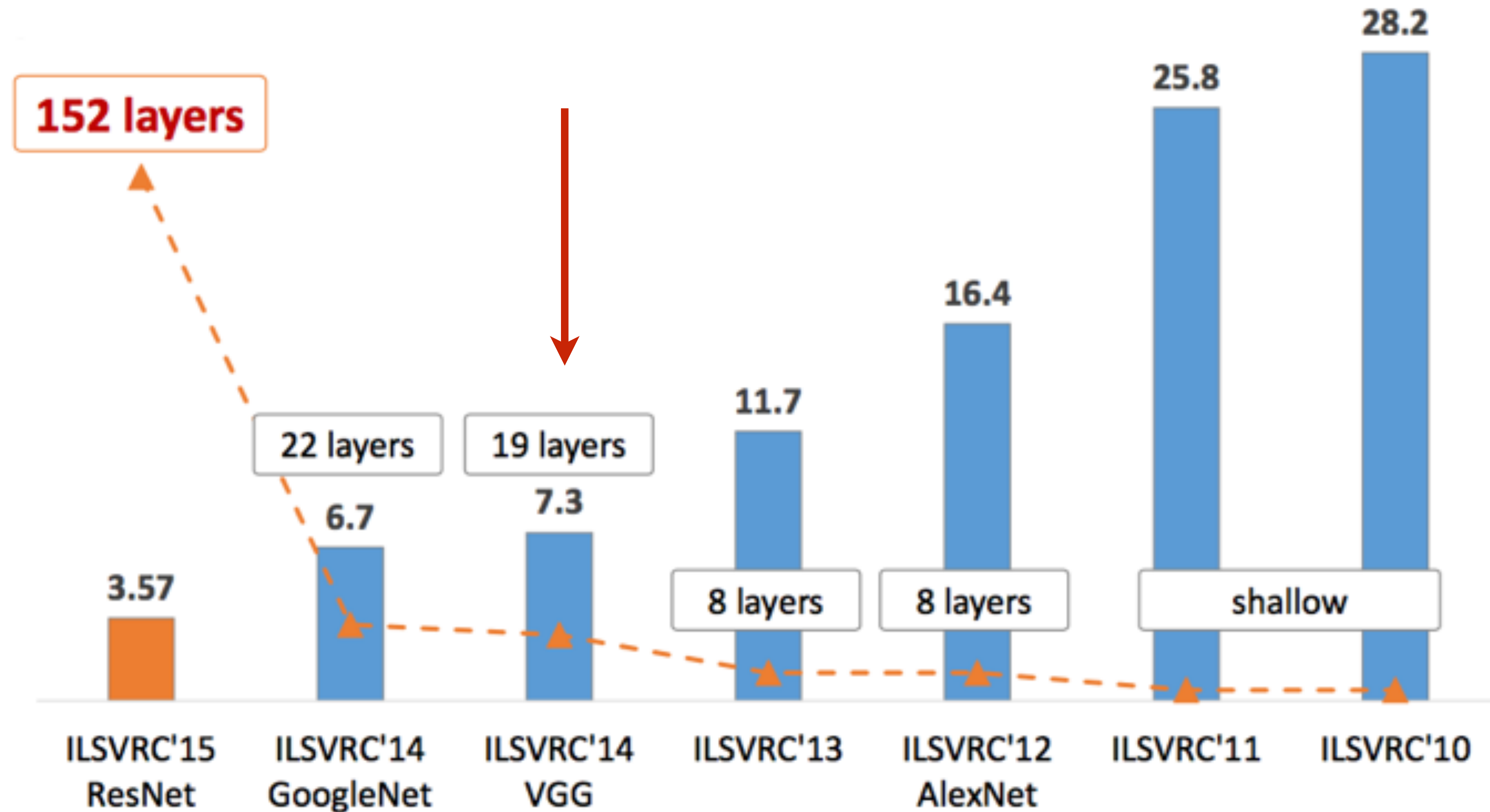
Annual competition of image classification at scale

Focuses on a subset of **1K synset** categories

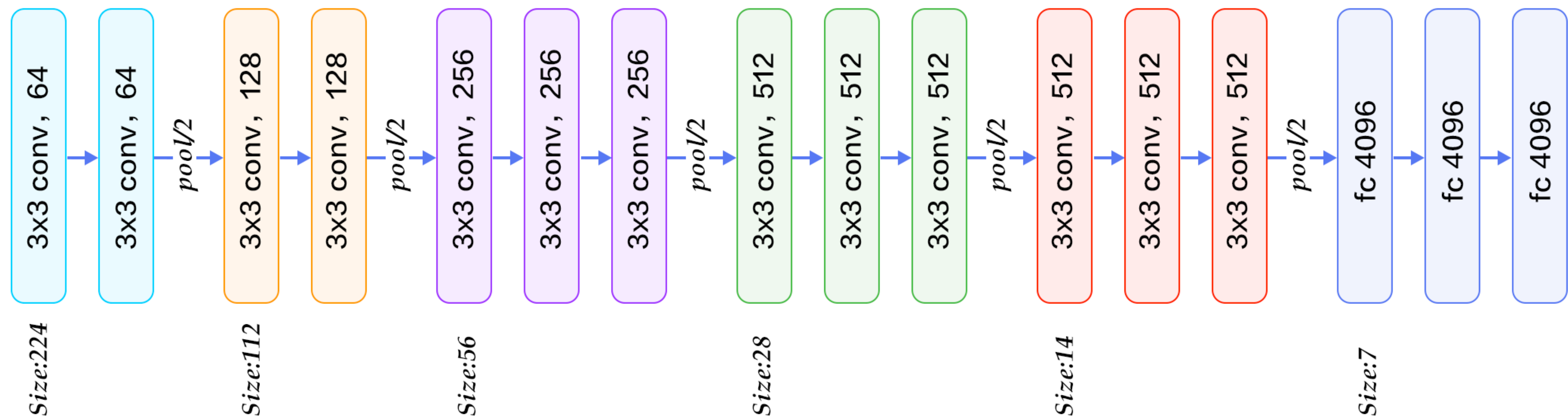
Scoring: need to predict true label within top K (K=5)



ILSVRC winner 2012

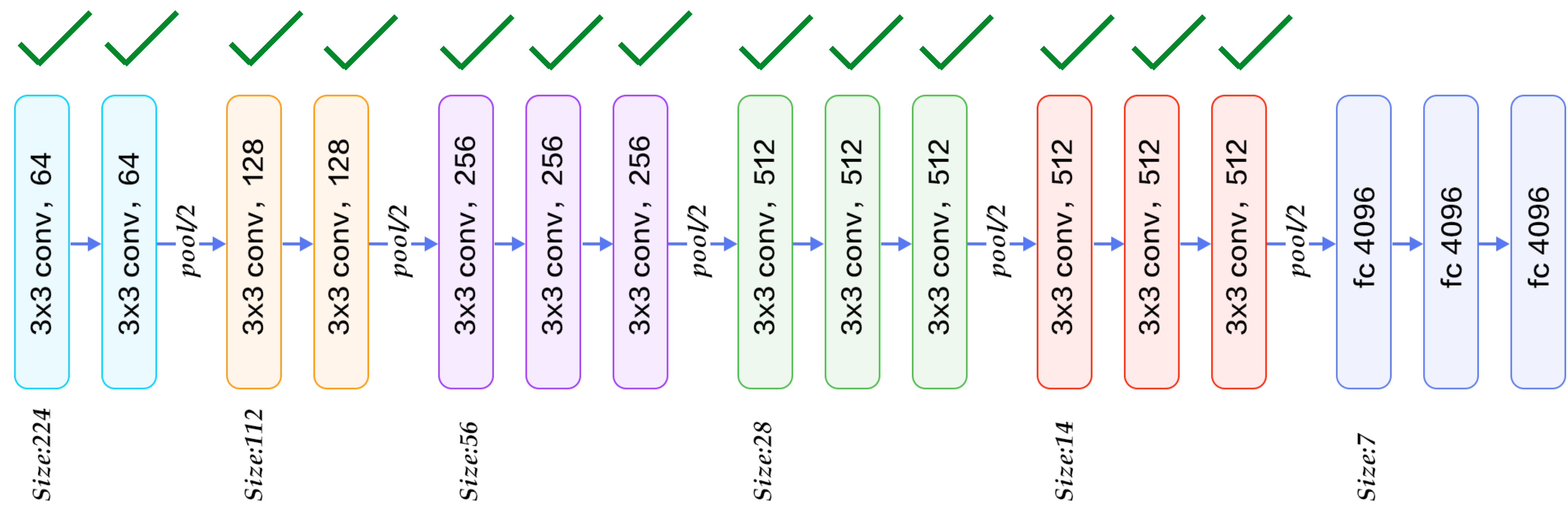


Convolutional Neural Networks



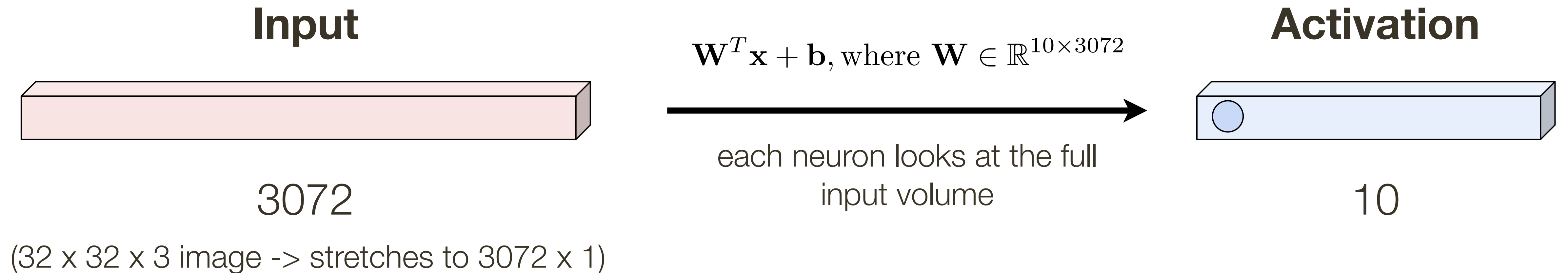
VGG-16 Network

Convolutional Neural Networks

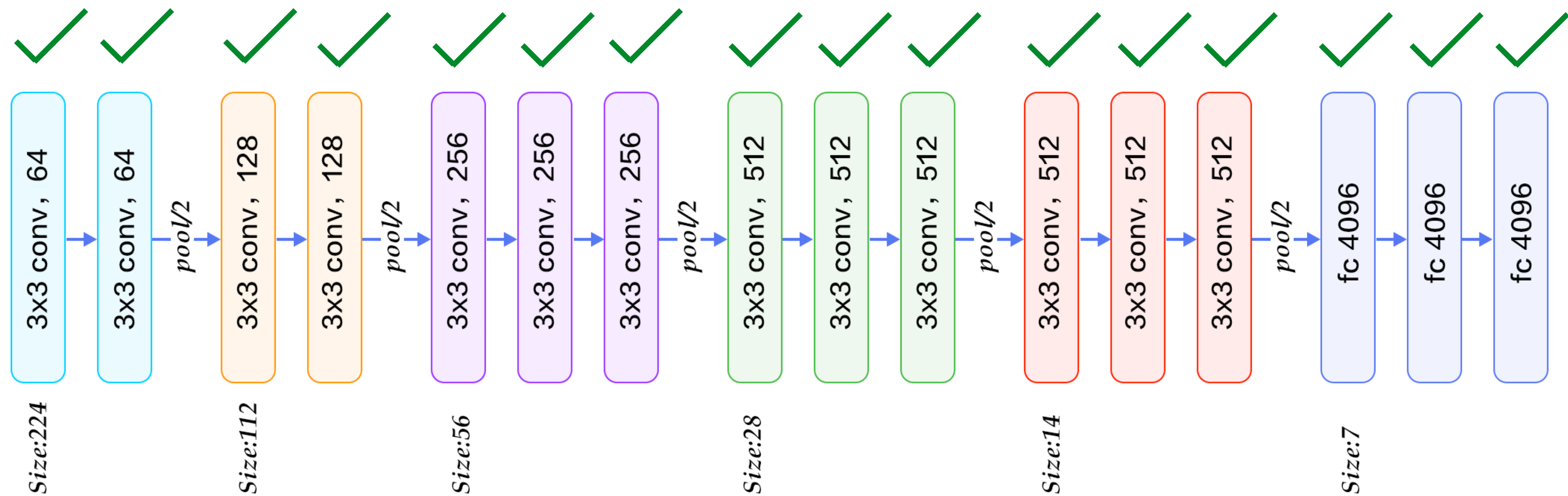


VGG-16 Network

CNNs: Reminder Fully Connected Layers

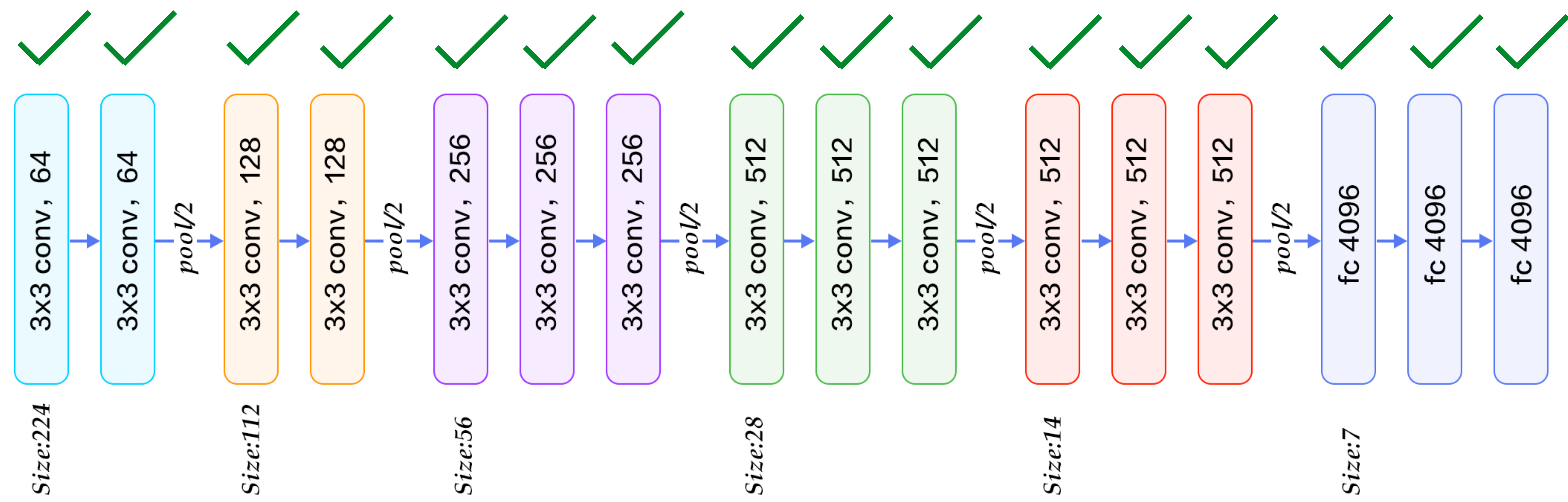


Convolutional Neural Networks



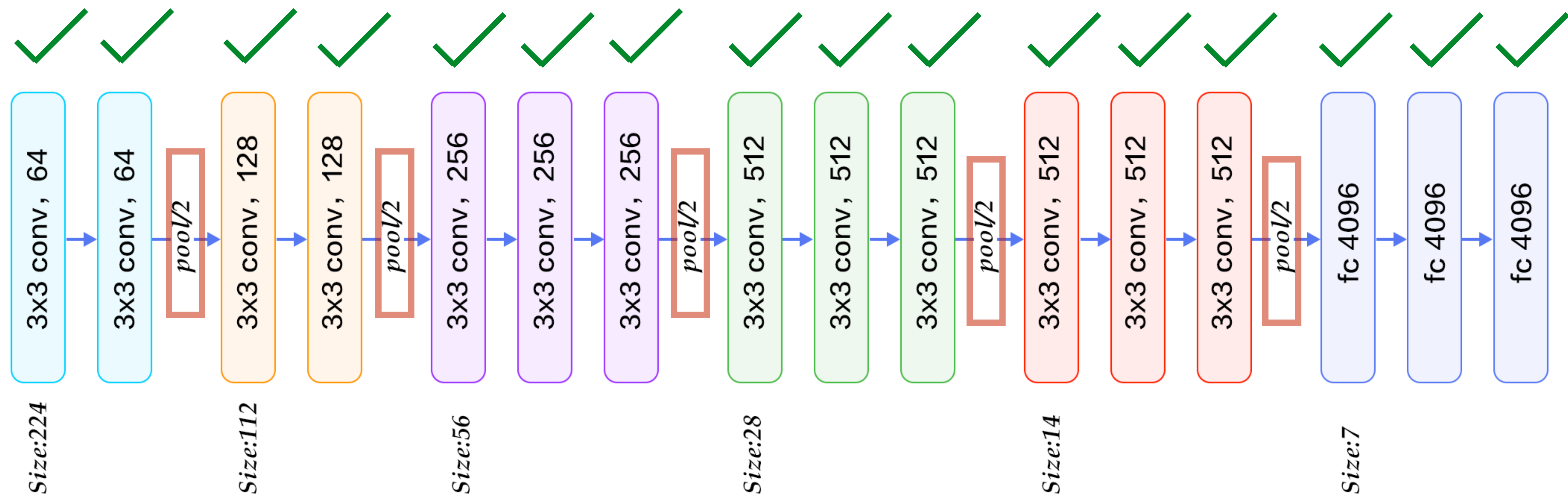
VGG-16 Network

Convolutional Neural Networks



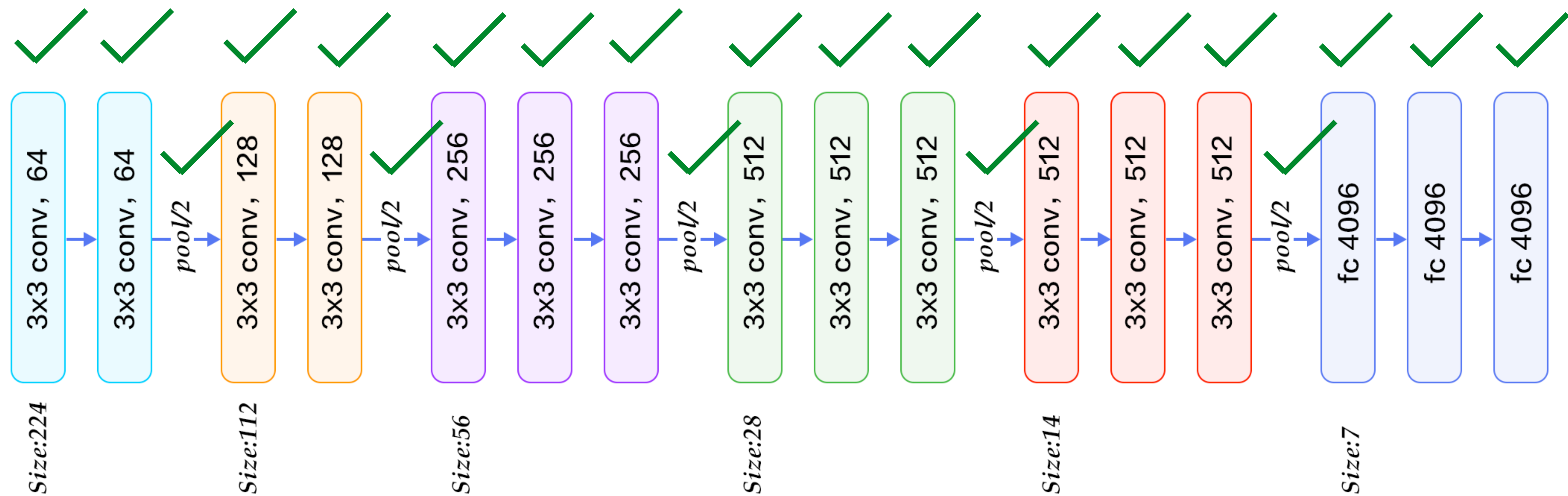
VGG-16 Network

Convolutional Neural Networks



VGG-16 Network

Convolutional Neural Networks



VGG-16 Network

VGG Net

[Simonyan and Zisserman, 2014]

INPUT: [224x224x3] memory: $224*224*3=150\text{K}$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory: $112*112*64=800\text{K}$ params: 0

CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory: $56*56*128=400\text{K}$ params: 0

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory: $28*28*256=200\text{K}$ params: 0

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory: $14*14*512=100\text{K}$ params: 0

CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory: $7*7*512=25\text{K}$ params: 0

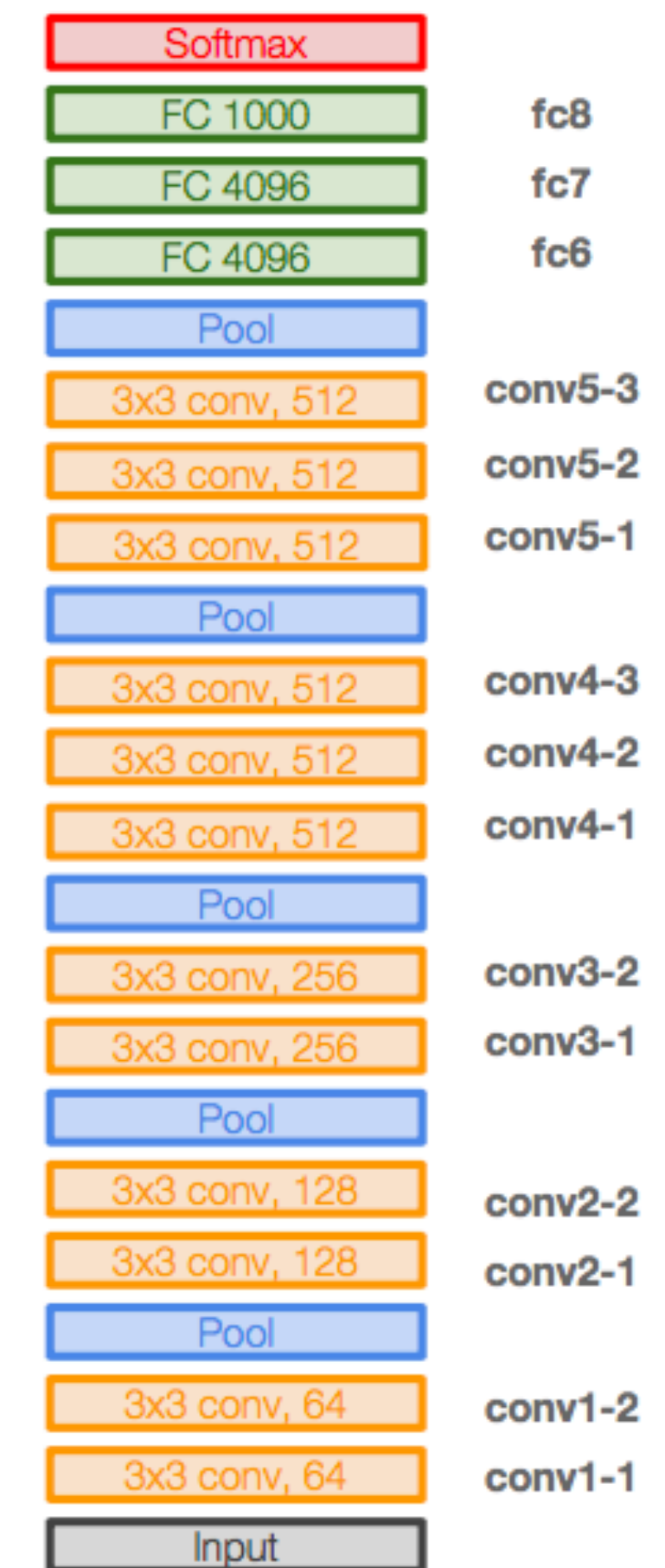
FC: [1x1x4096] memory: 4096 params: $7*7*512*4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096*4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096*1000 = 4,096,000$

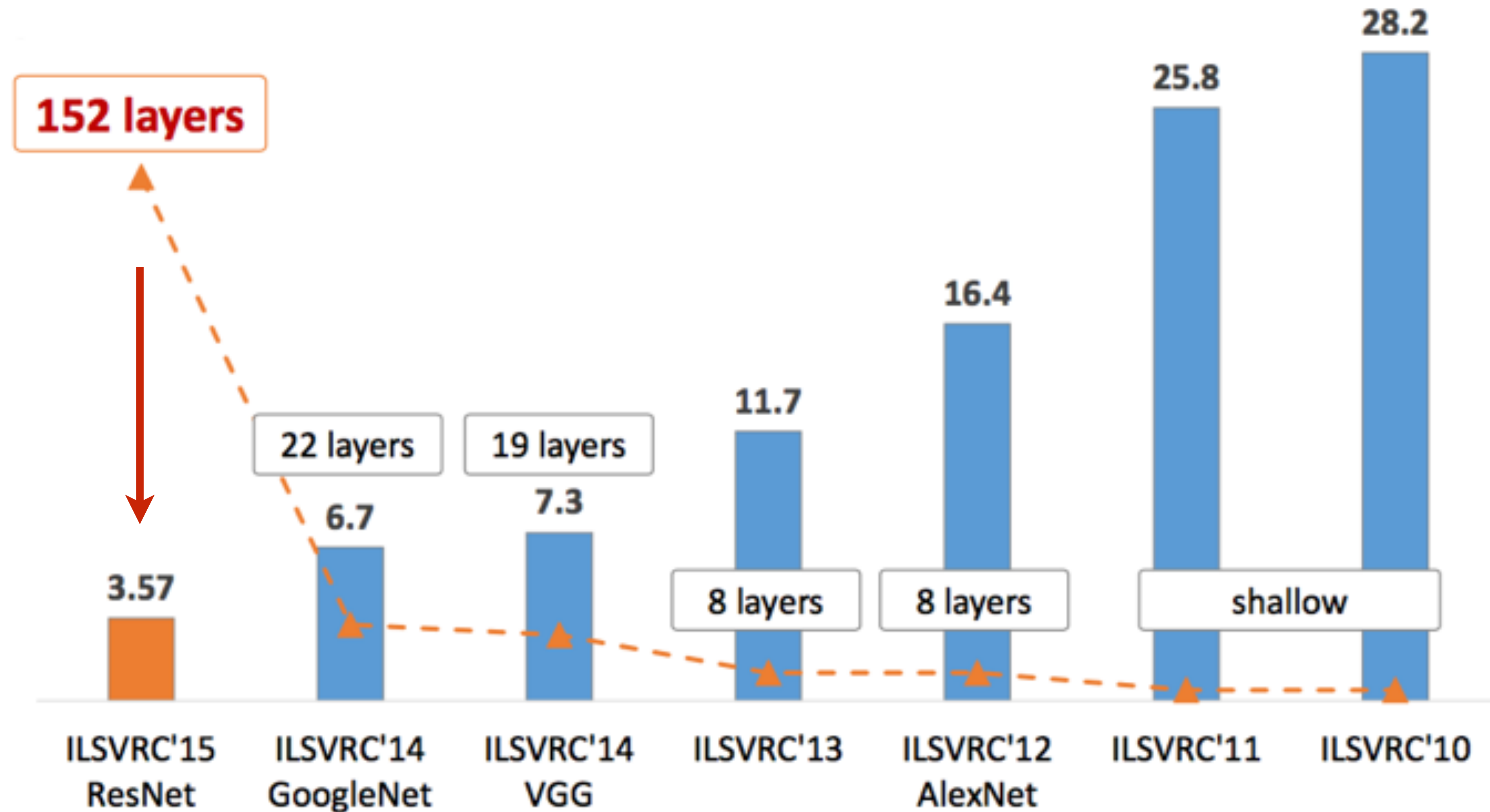
TOTAL memory: $24\text{M} * 4 \text{ bytes} \sim 96\text{MB}$ / image (only forward! ~ 2 for bwd)

TOTAL params: 138M parameters



VGG16

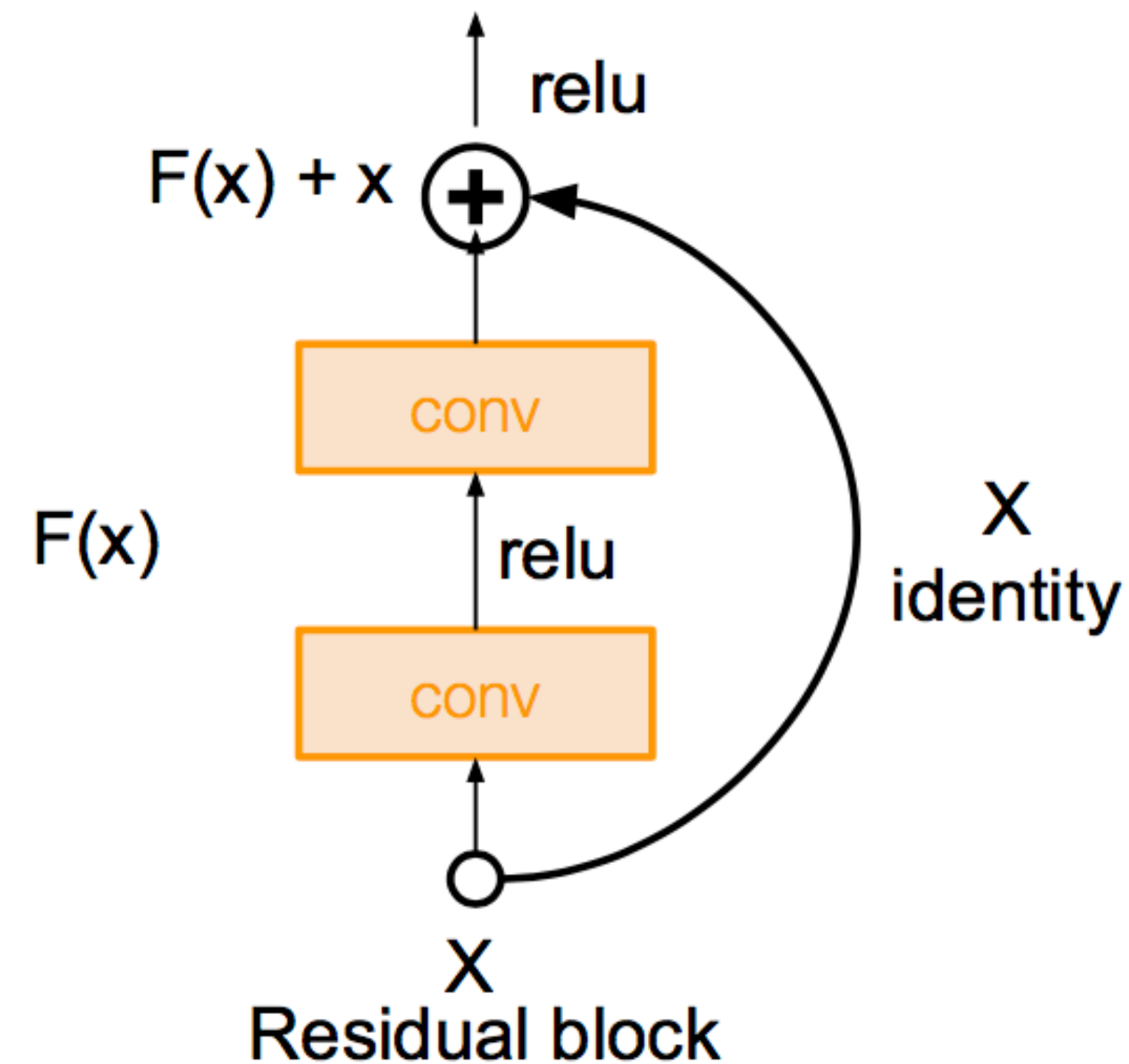
ILSVRC winner 2012



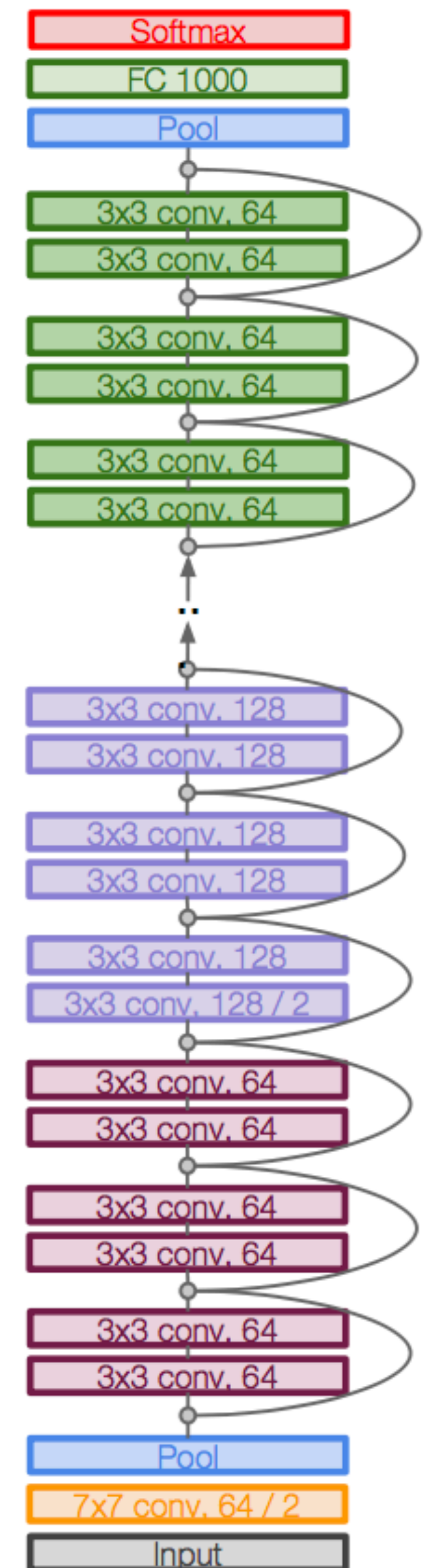
ResNet

even deeper — **152 layers!**

using residual connections



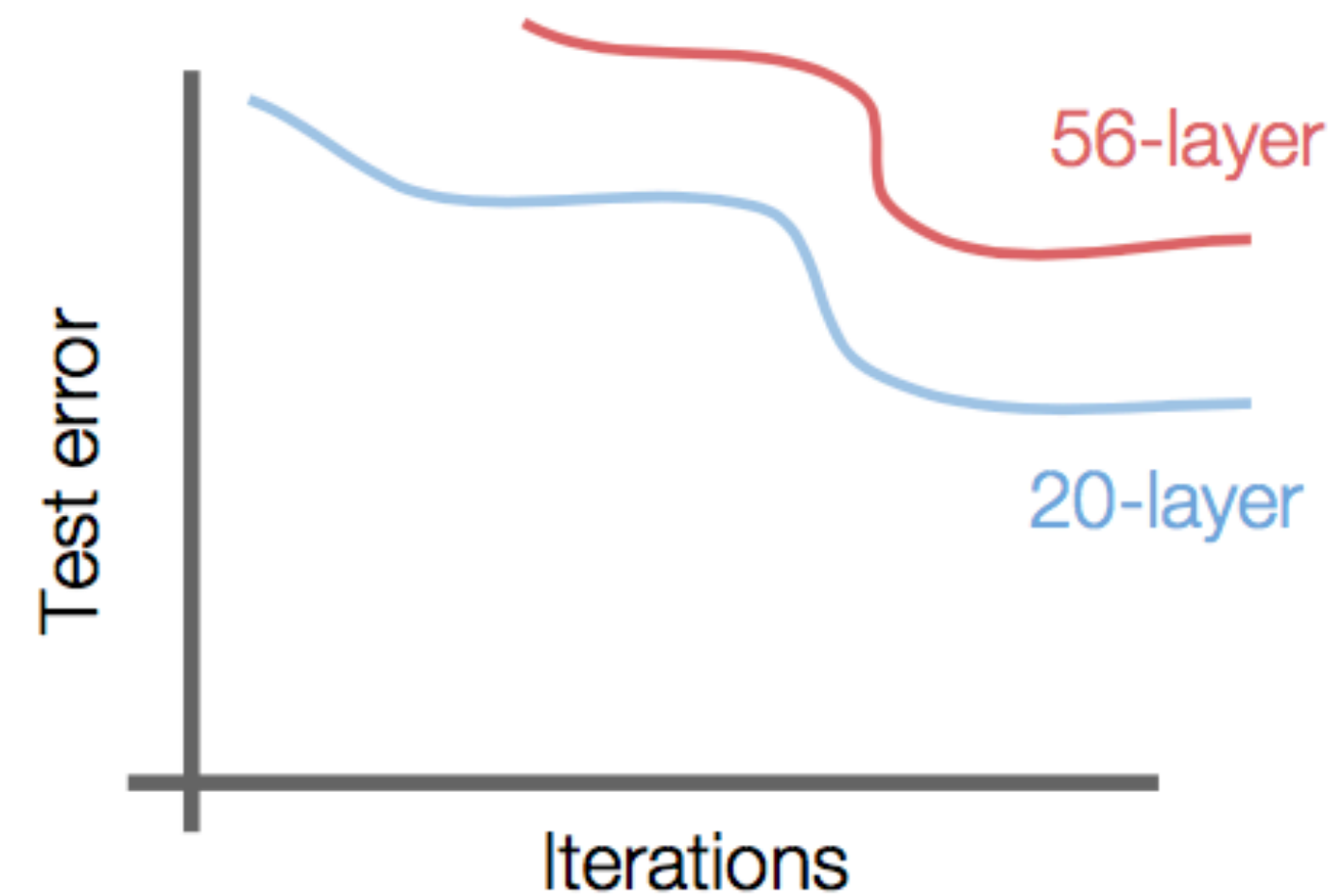
[He et al., 2015]



ResNet: Motivation

[He et al., 2015]

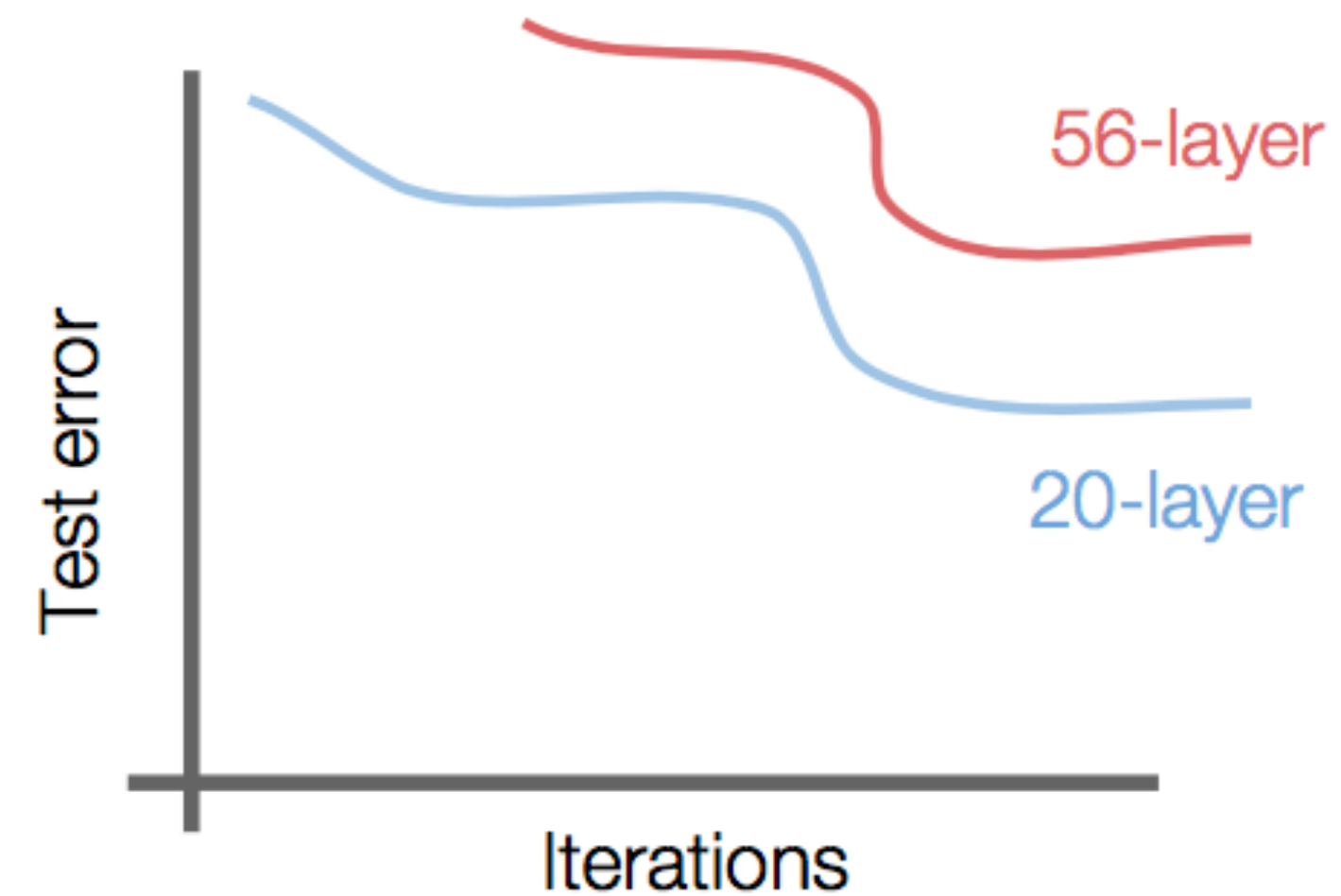
What happens when we continue to stacking deeper layers on a “plain” CNN



ResNet: Motivation

[He et al., 2015]

What happens when we continue to stacking deeper layers on a “plain” CNN

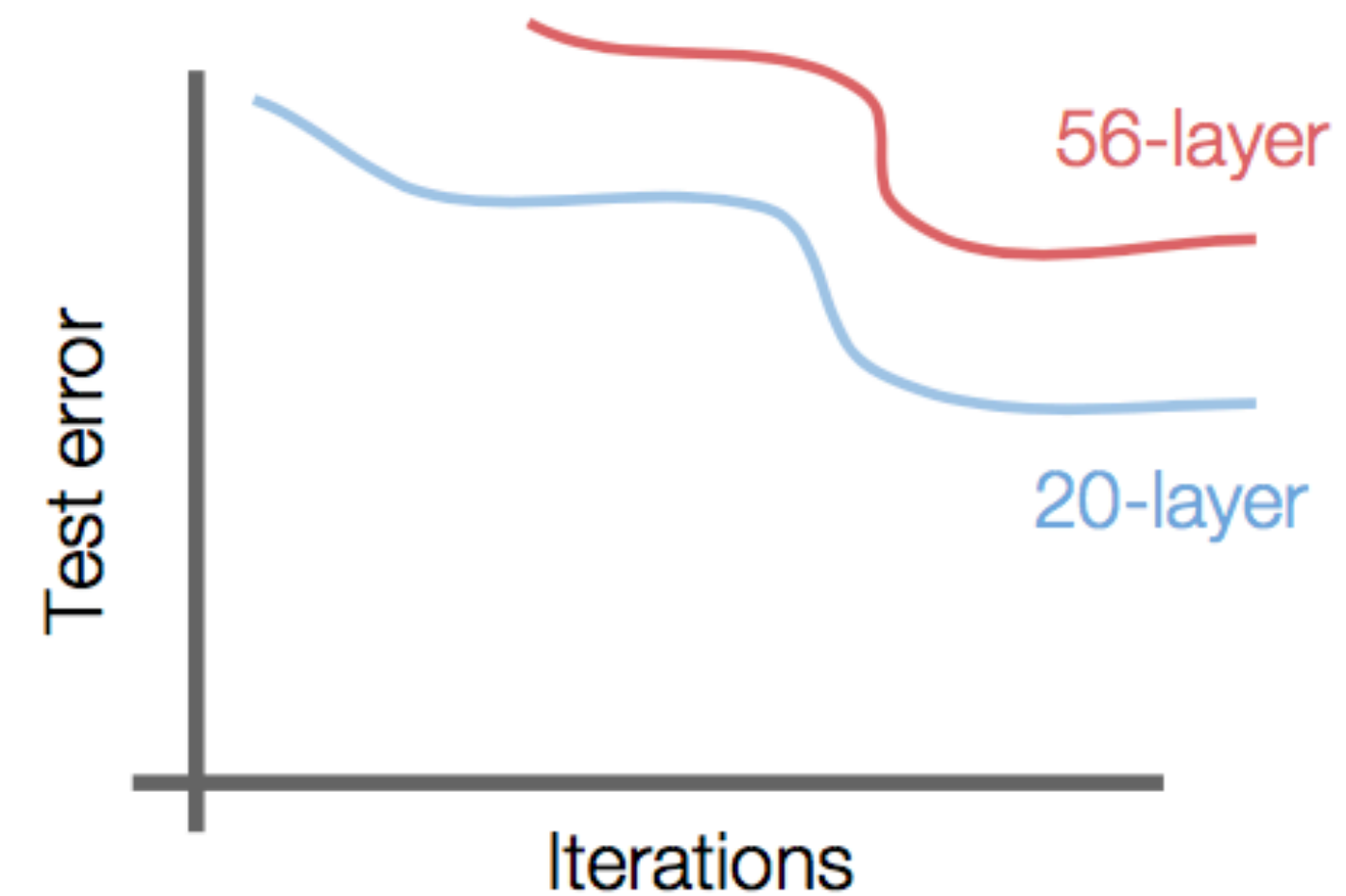
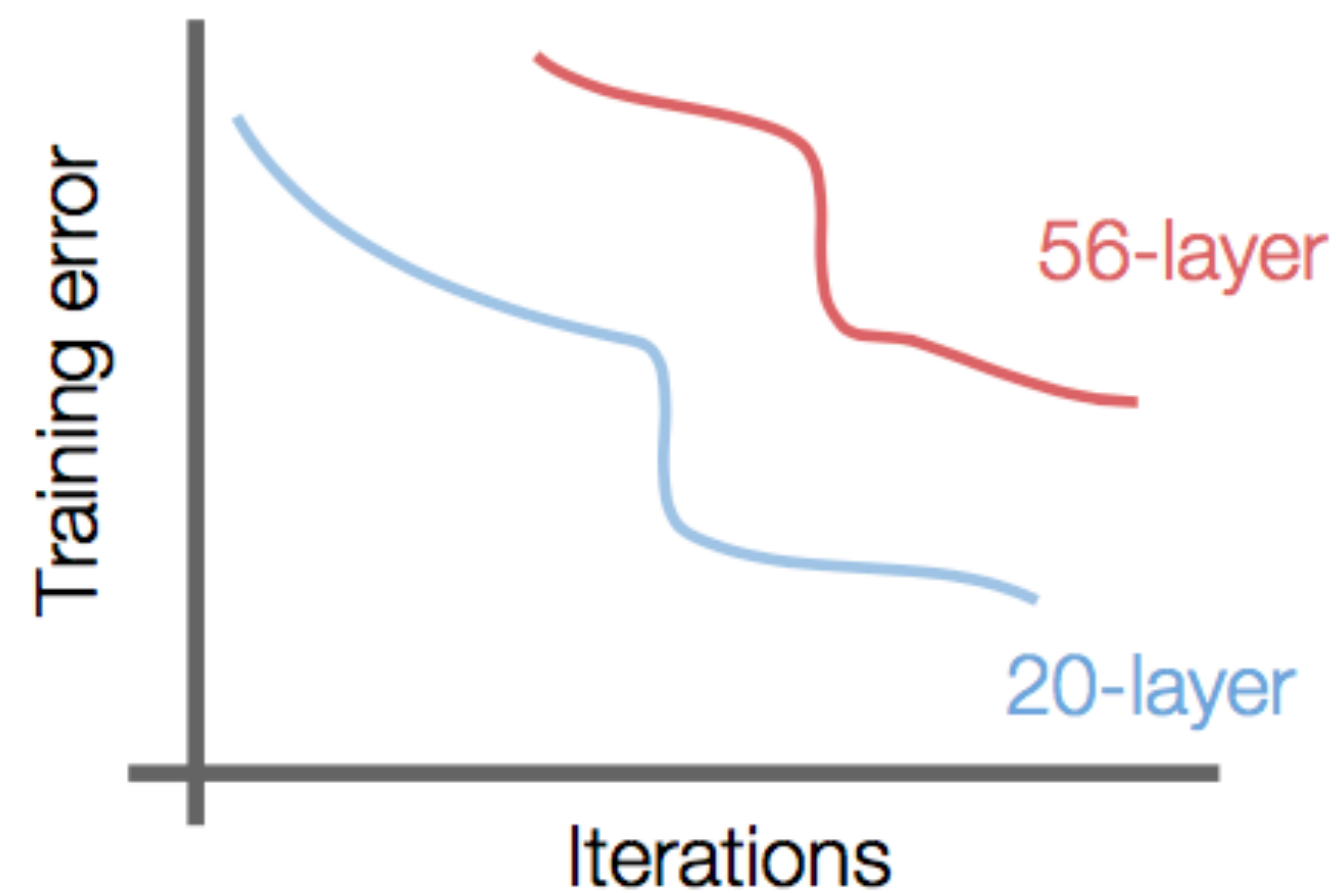


Whats the **problem**?

ResNet: Motivation

[He et al., 2015]

What happens when we continue to stacking deeper layers on a “plain” CNN



Whats the **problem**?

Optimizing **Deep** Neural Networks

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \underbrace{w_2 \sigma'(z_2)}_{< \frac{1}{4}} \underbrace{w_3 \sigma'(z_3)}_{< \frac{1}{4}} \underbrace{w_4 \sigma'(z_4)}_{\text{common terms}} \frac{\partial C}{\partial a_4}$$

This is called **vanishing gradient** problem

- makes deep networks hard to train
- later layers learn faster than earlier ones

$$\frac{\partial C}{\partial b_3} = \sigma'(z_3) \underbrace{w_4 \sigma'(z_4)}_{\text{common terms}} \frac{\partial C}{\partial a_4}$$

ResNet: Motivation

[He et al., 2015]

Hypothesis: deeper models are harder to optimize (optimization problem)

ResNet: Motivation

[He et al., 2015]

Hypothesis: deeper models are harder to optimize (optimization problem)

Observation: the deeper model should (conceptually) perform just as well (e.g., take shallower model and use identity for all remaining layers)

ResNet: Motivation

[He et al., 2015]

Hypothesis: deeper models are harder to optimize (optimization problem)

Observation: the deeper model should (conceptually) perform just as well (e.g., take shallower model and use identity for all remaining layers)

How do we implement this idea in practice

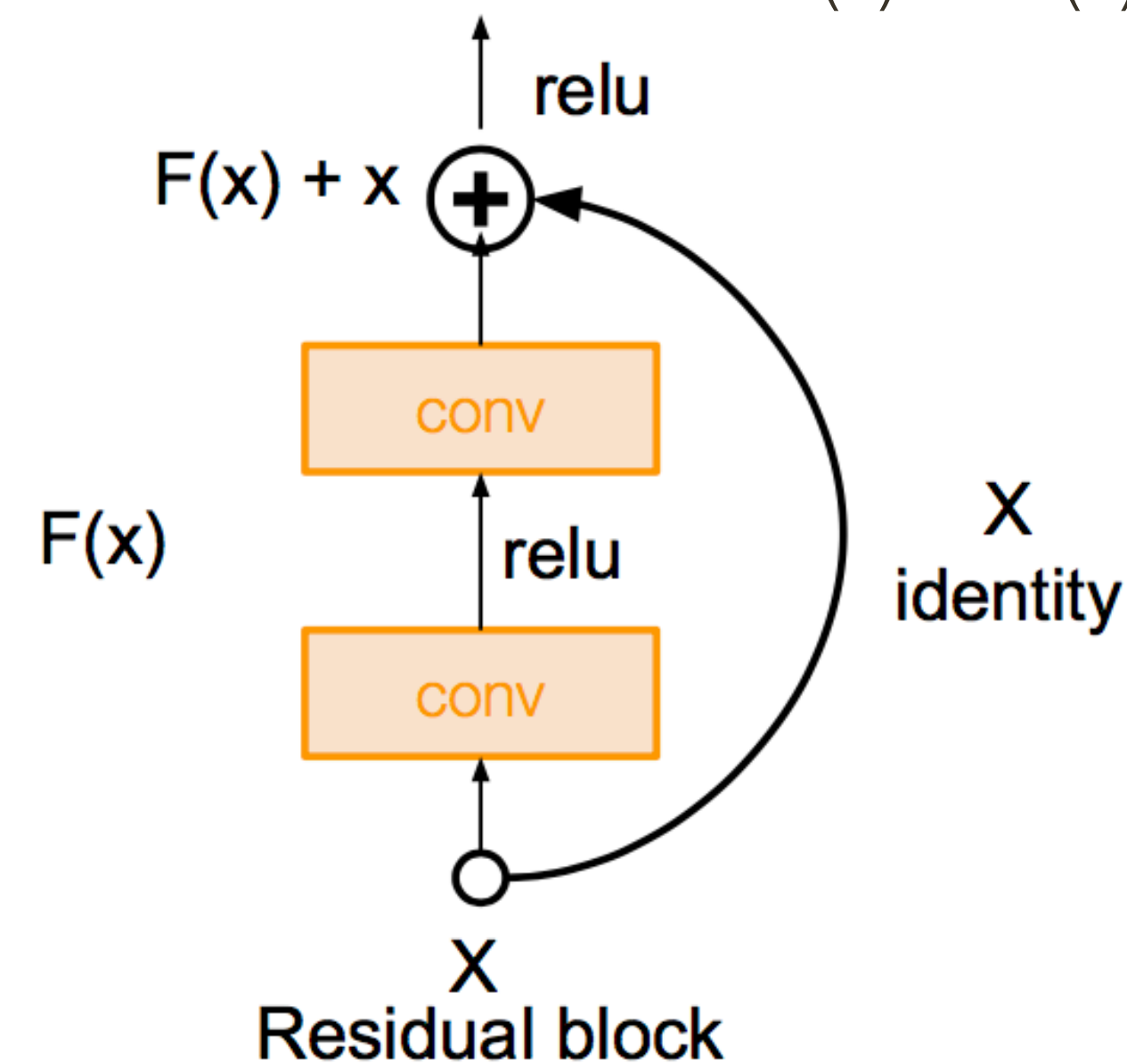
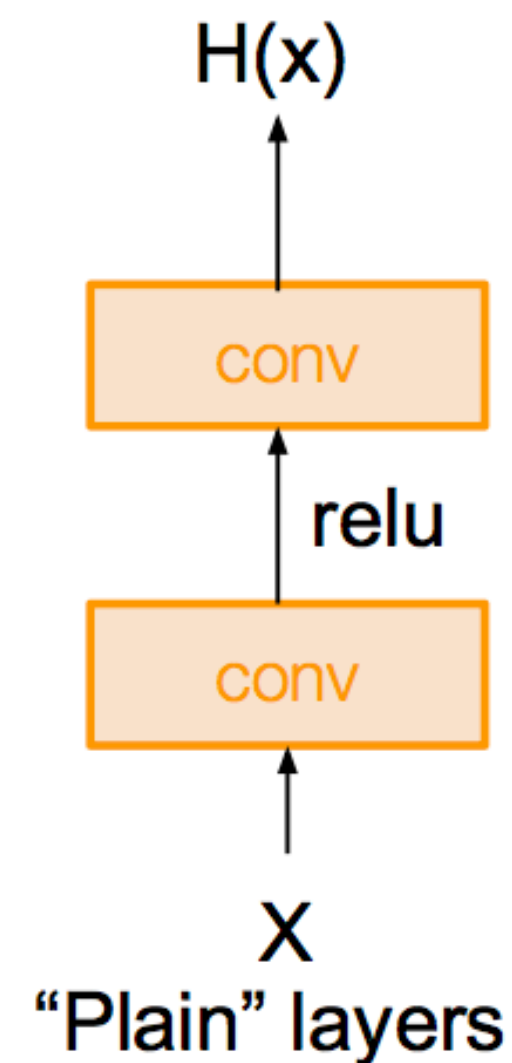
ResNet

[He et al., 2015]

Solution: use network to fit residual mapping instead of directly trying to fit a desired underlying mapping

$$H(x) = F(x) + X$$

Use layers to fit **residual**
 $F(x) = H(x) - X$ instead of $H(x)$ directly

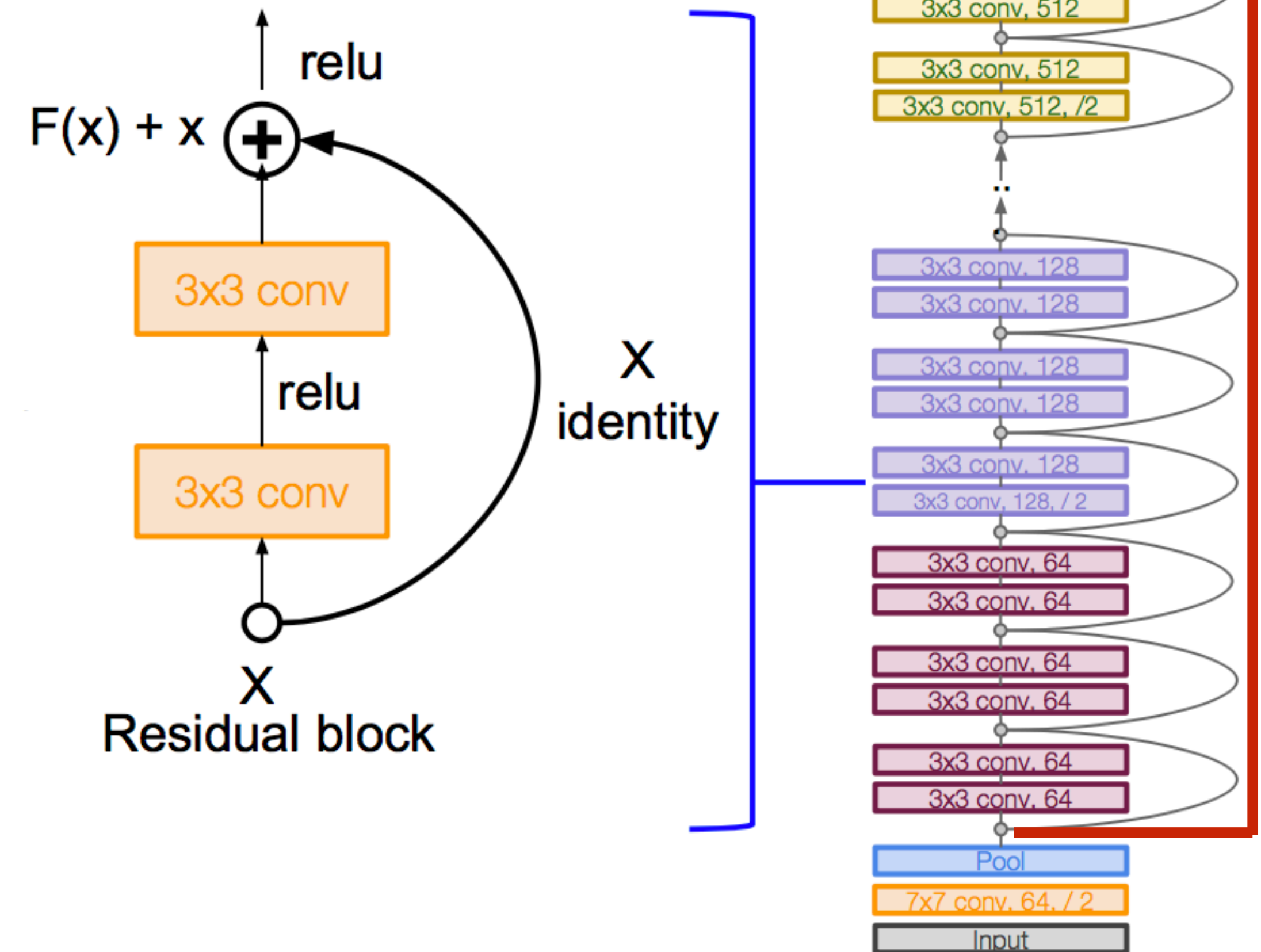


ResNet

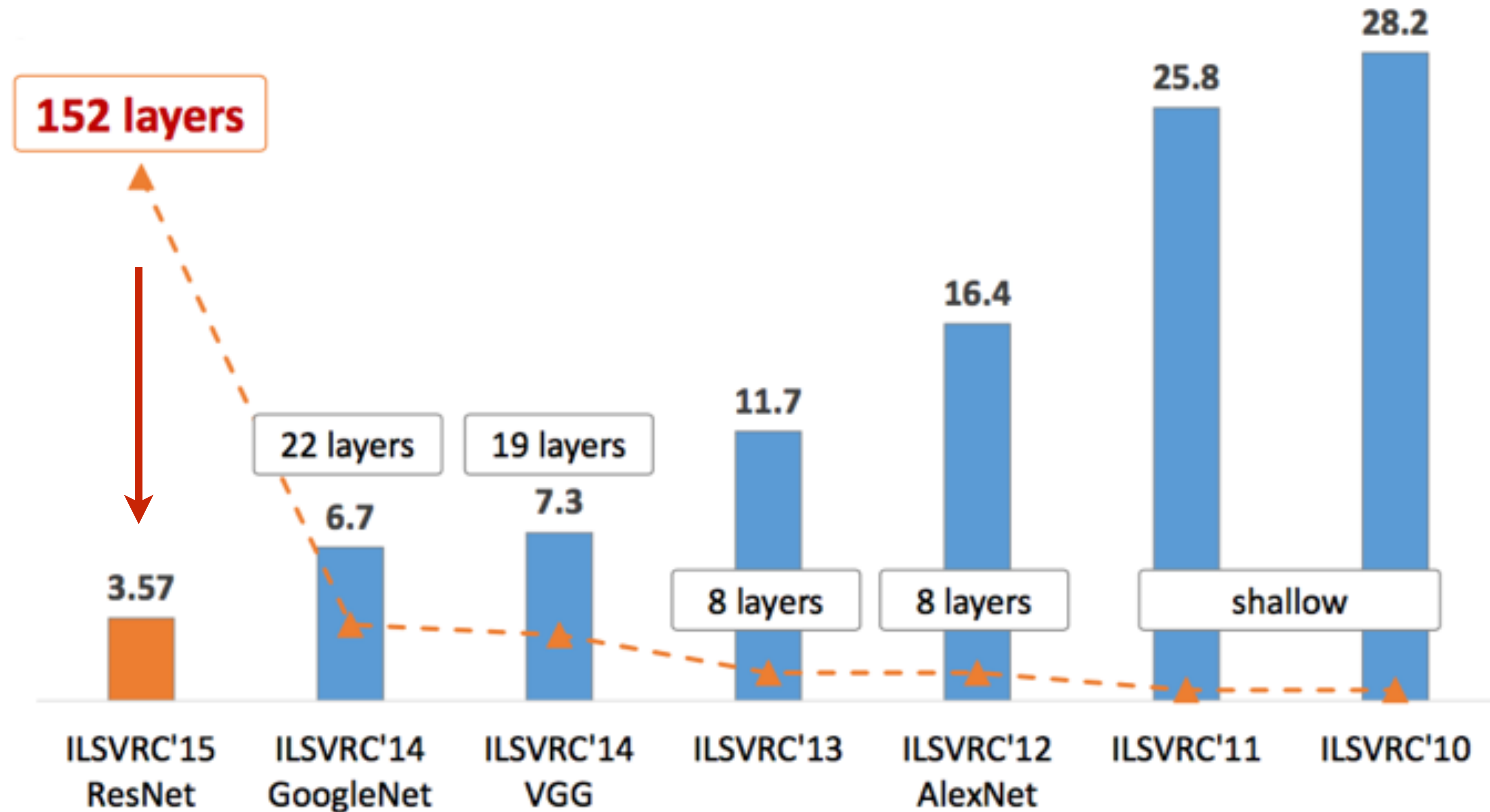
[He et al., 2015]

Full details

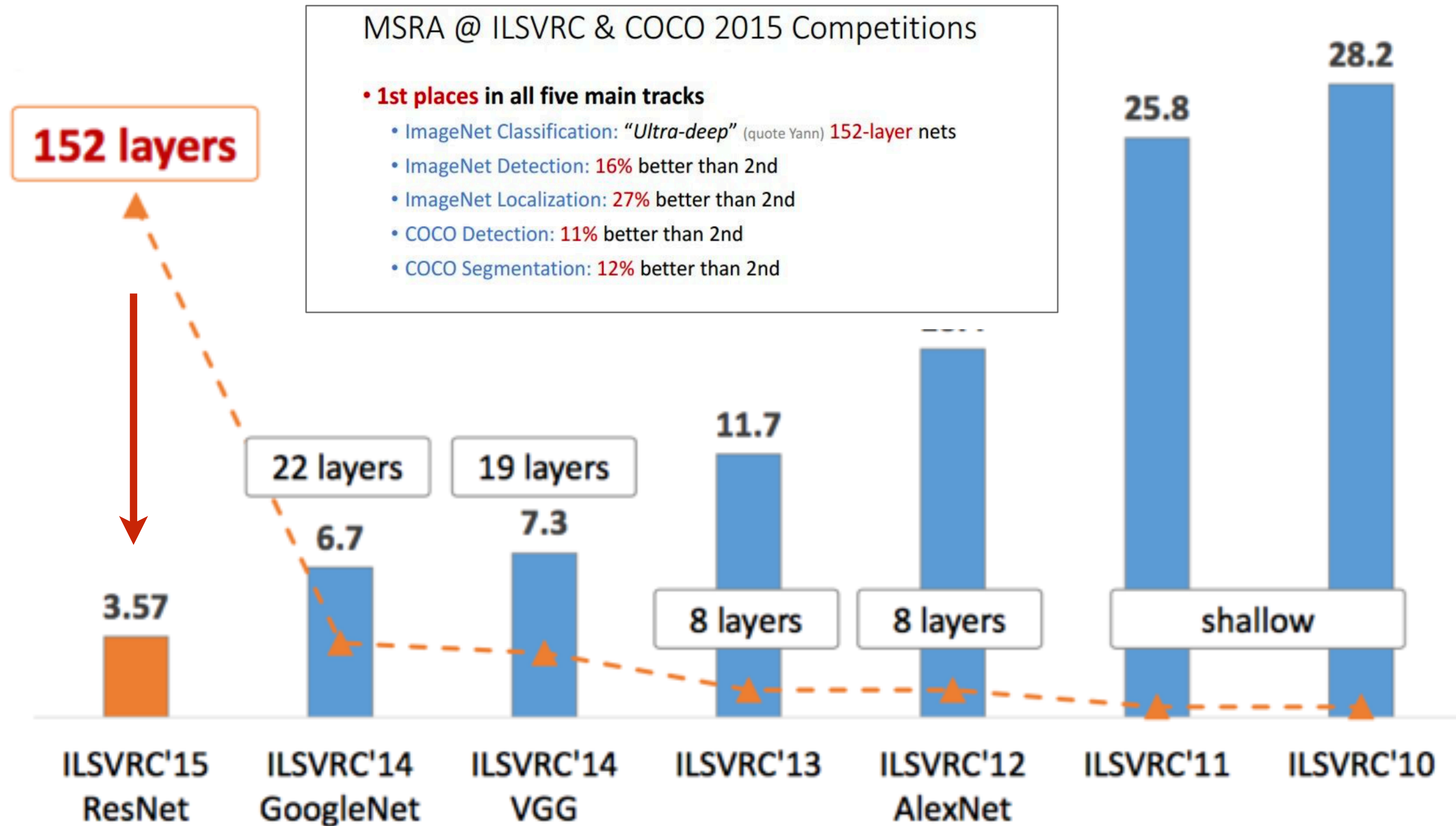
- Stacked **residual blocks**
- Every residual block consists of **two 3x3 filters**
- Periodically double # of filters and downsample spatially using stride of 2
- Additional convolutional layer in the beginning
- **No FC layers** at the end (only FC to output 1000 classes)



ILSVRC winner 2012

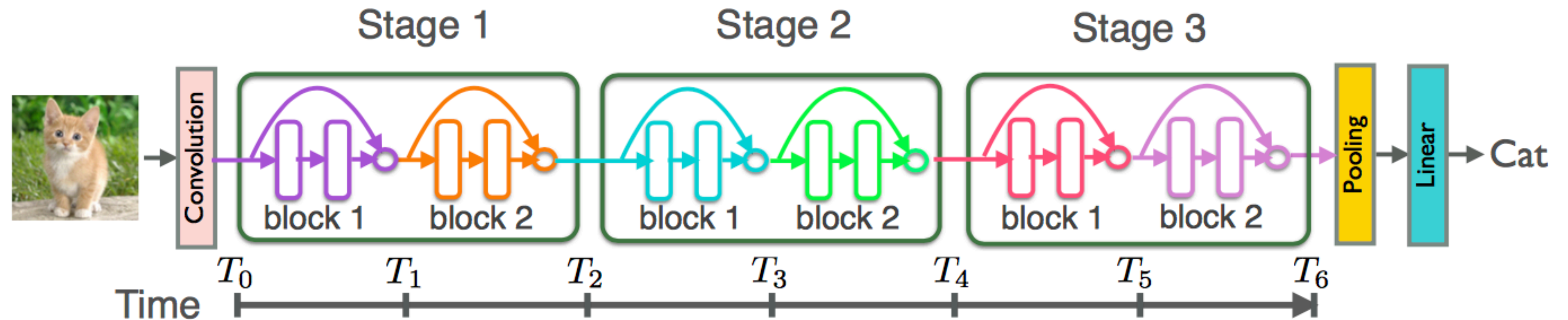


ILSVRC winner 2012



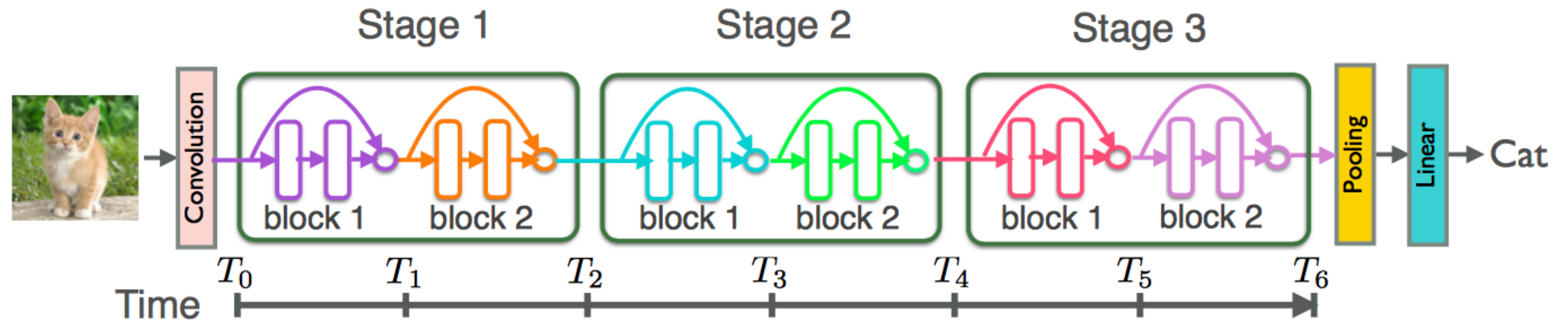
ResNet: A little theory

One can view a sequence of outputs from residual layers as a **Dynamical System**

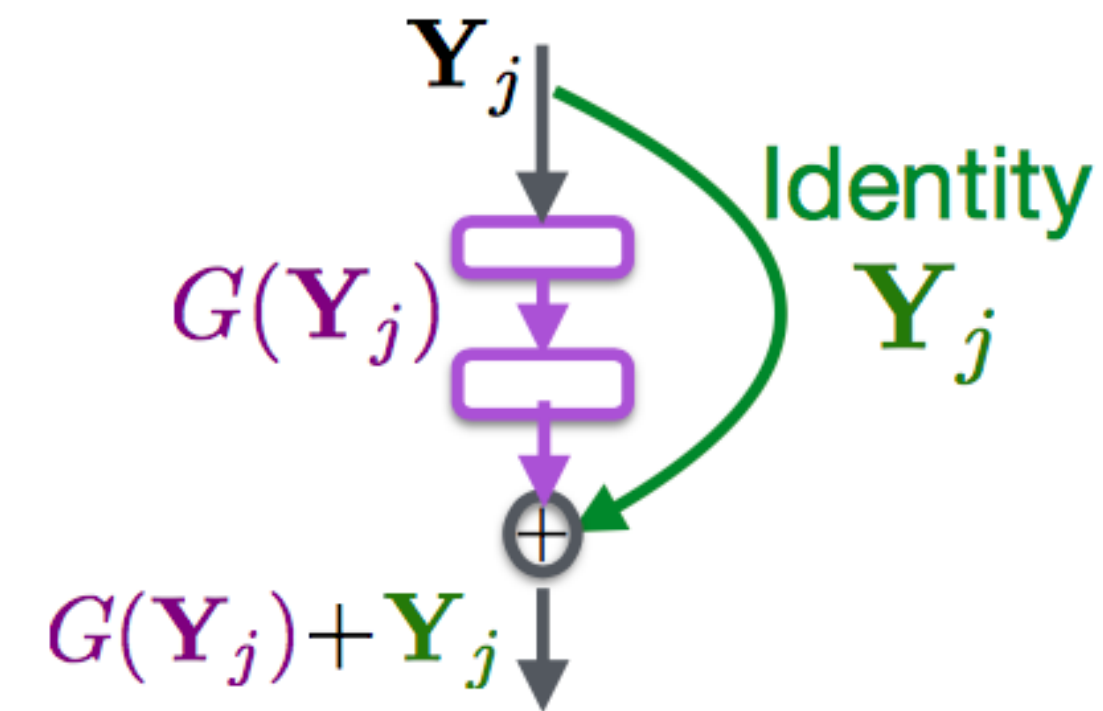


ResNet: A little theory

One can view a sequence of outputs from residual layers as a **Dynamical System**

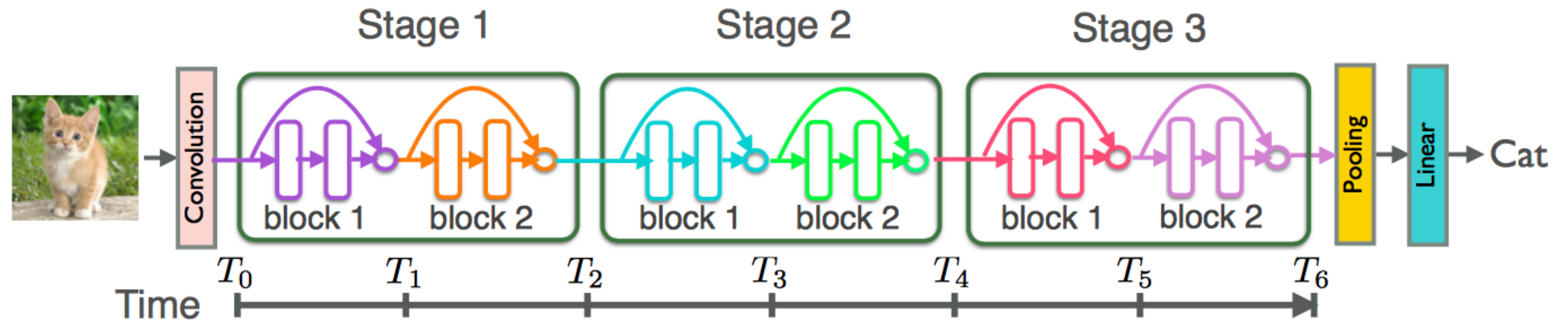


$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + G(\mathbf{Y}_j, \theta_j)$$



ResNet: A little theory

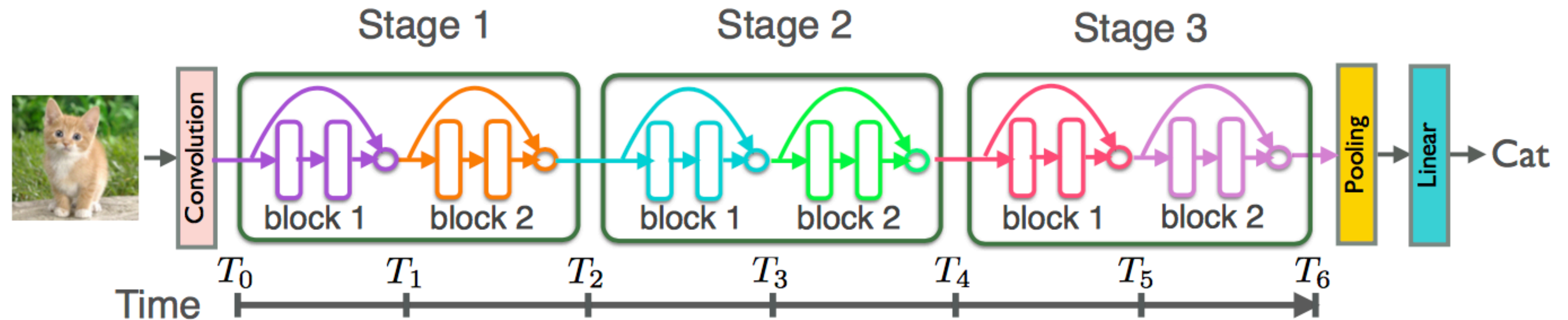
One can view a sequence of outputs from residual layers as a **Dynamical System**



What happens if you take more layers and take smaller steps?

ResNet: A little theory

One can view a sequence of outputs from residual layers as a **Dynamical System**



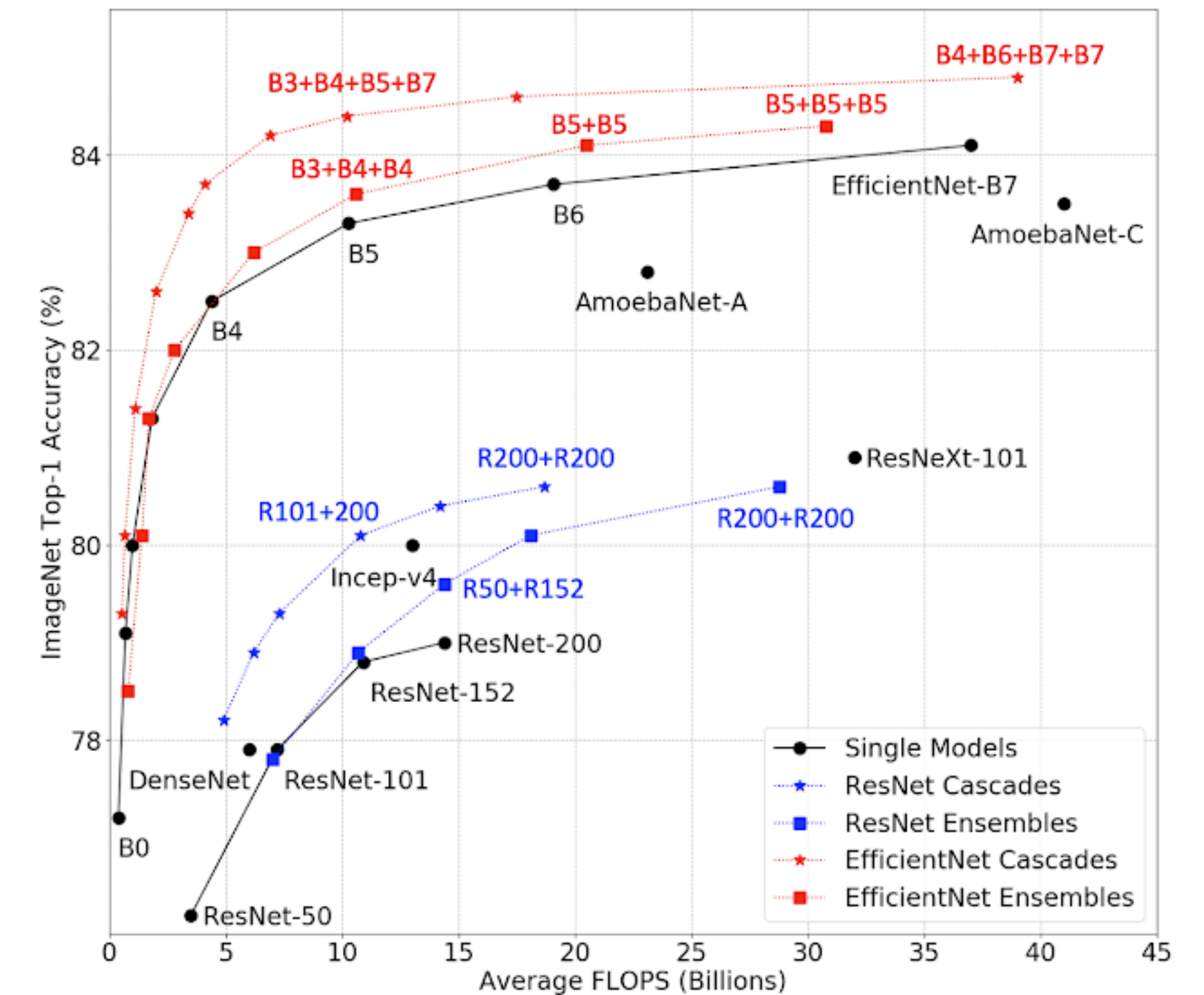
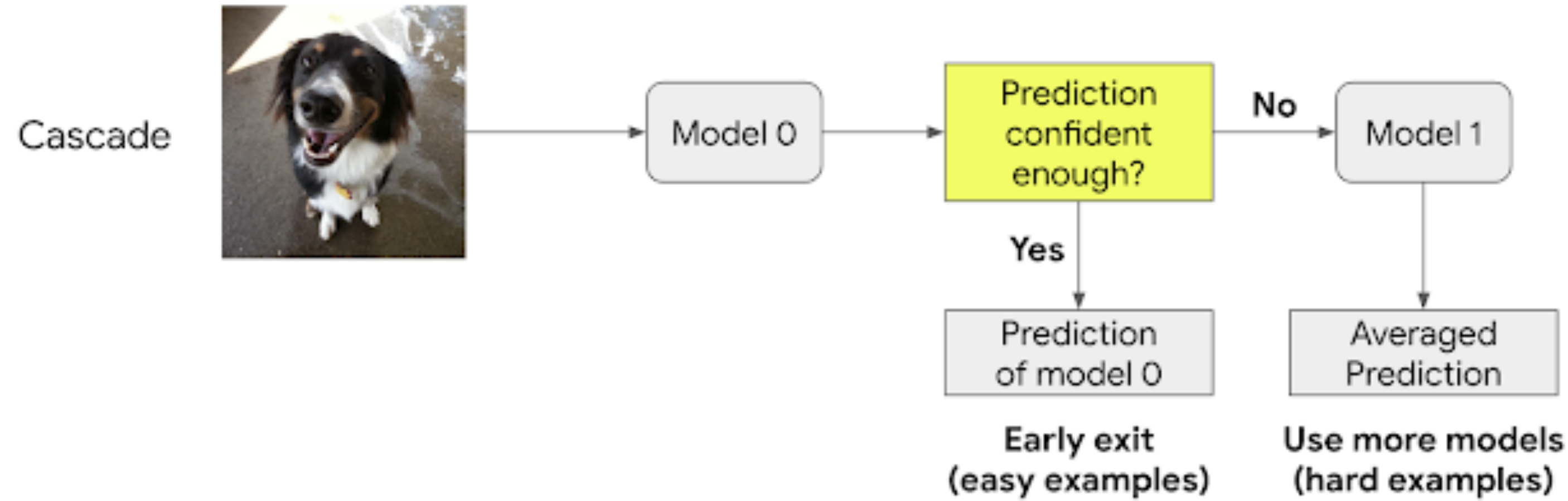
What happens if you take more layers and take smaller steps?

You can actually treat a neural network as an **ODE**:
$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$$

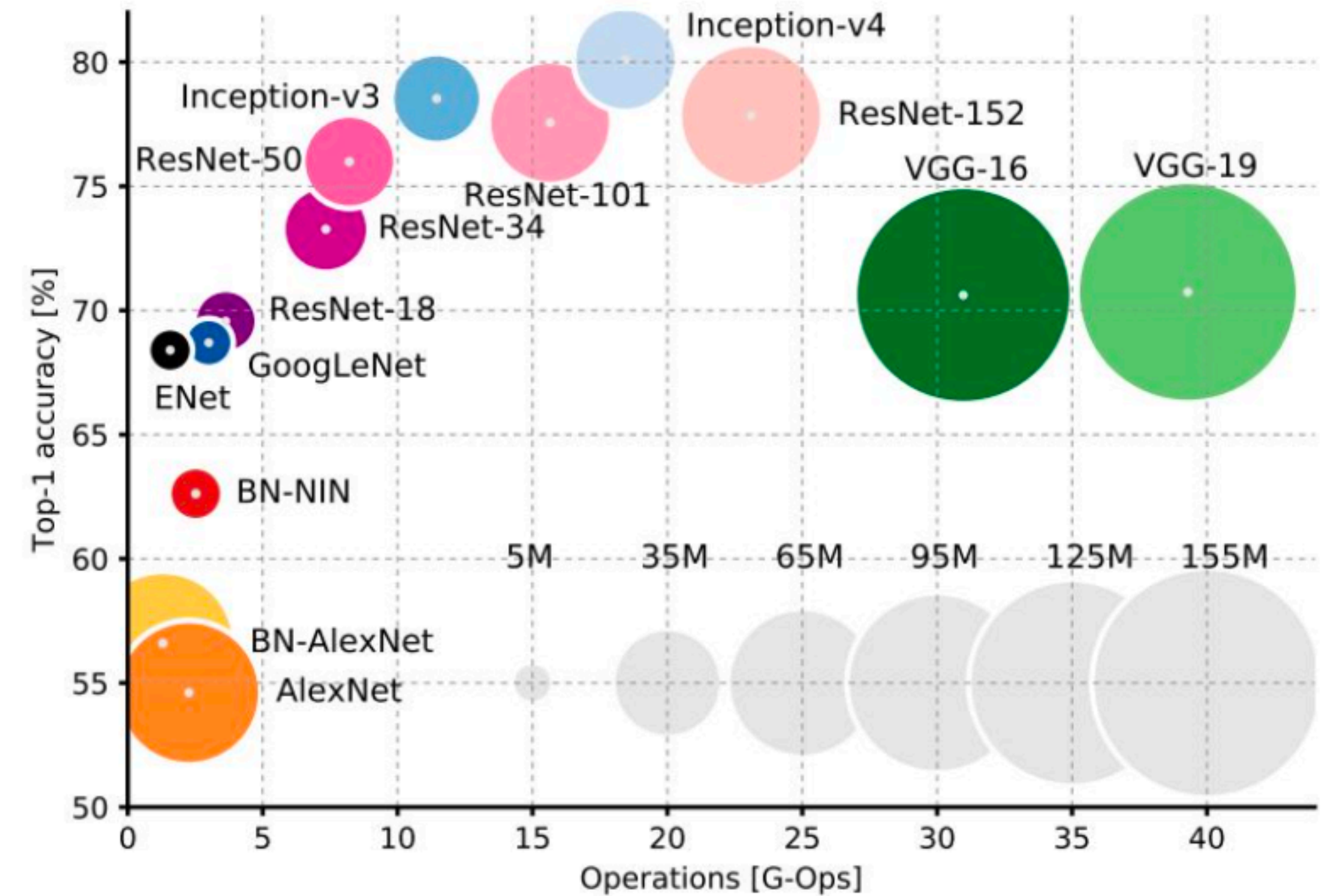
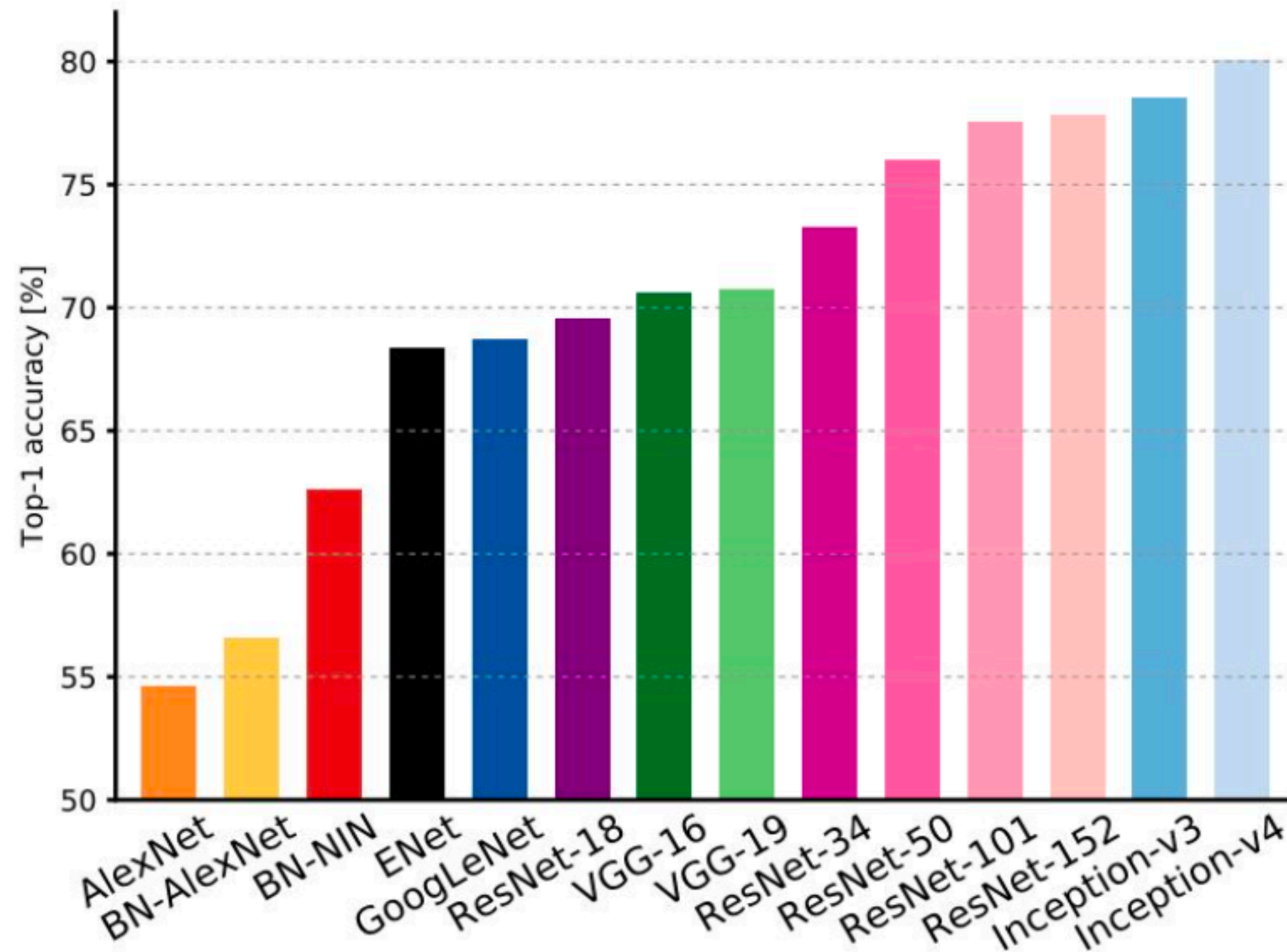
[Chen et al., NIPS 2018 **best paper**]

An Aside: Neural Network Cascades

[Wang et al., ICLR 2022]



Comparing Complexity



An Analysis of Deep Neural Network Models for Practical Applications, 2017.