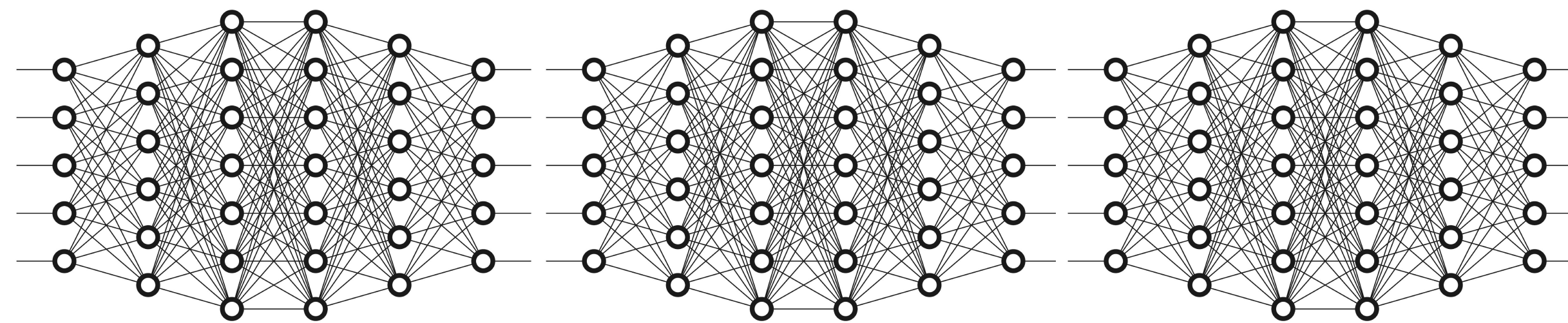




CPSC 425: Computer Vision



Lecture 22: Neural Networks, CNNs (cont.)

Menu for Today

Topics:

- Convolutional Layers
- Pooling Layers
- Classification, Detection

Readings:

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

Reminders:

- **Assignment 6:** Deep Learning is out and due **Thursday**
- **Material** for **Final Prep** will be on Canvas **tonight**
- **Quiz 6** will be available **tonight**

Please fill out
Student Evaluations
(on Canvas)

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



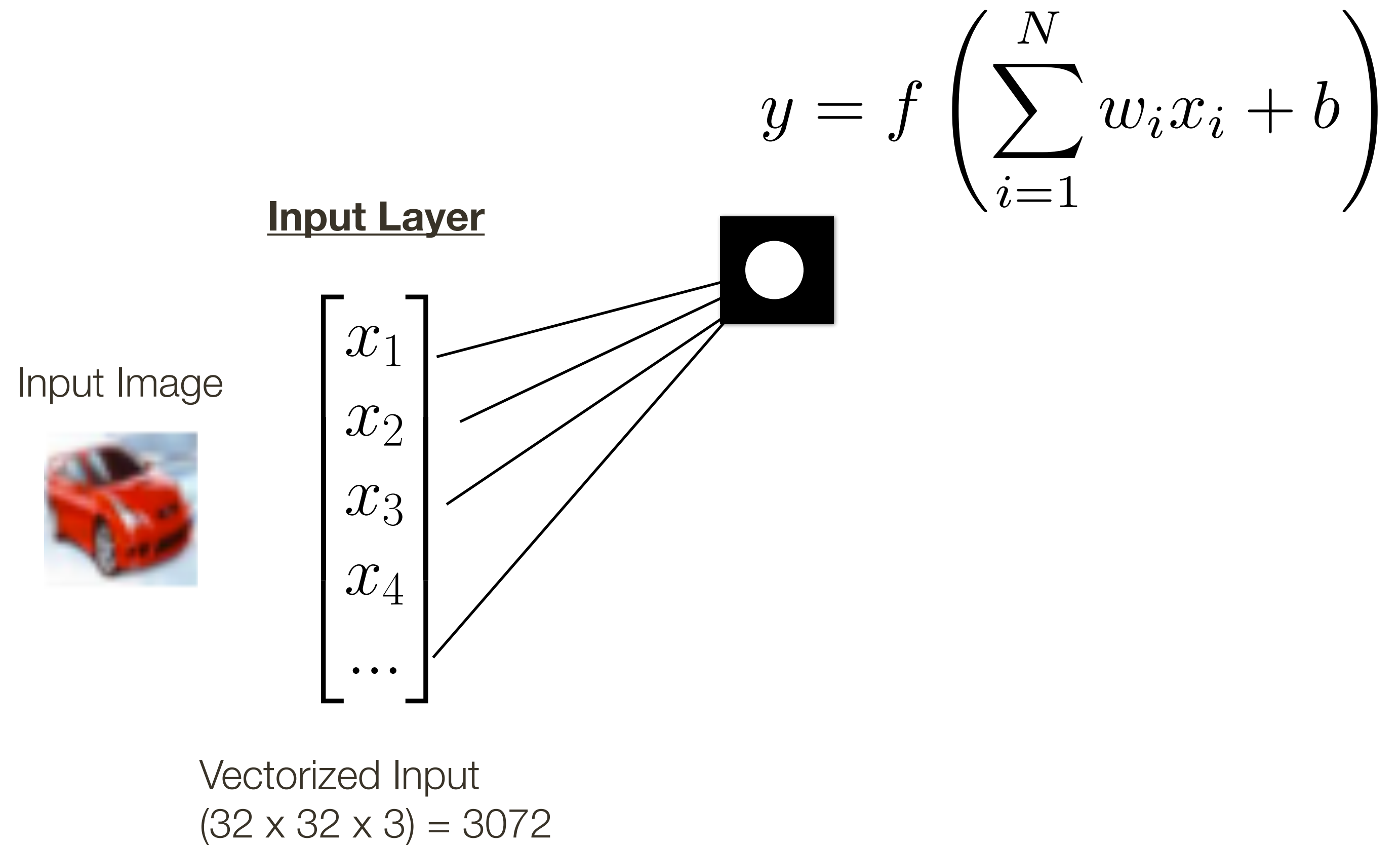
Today's “**fun**” Example: Yolo Object Detector



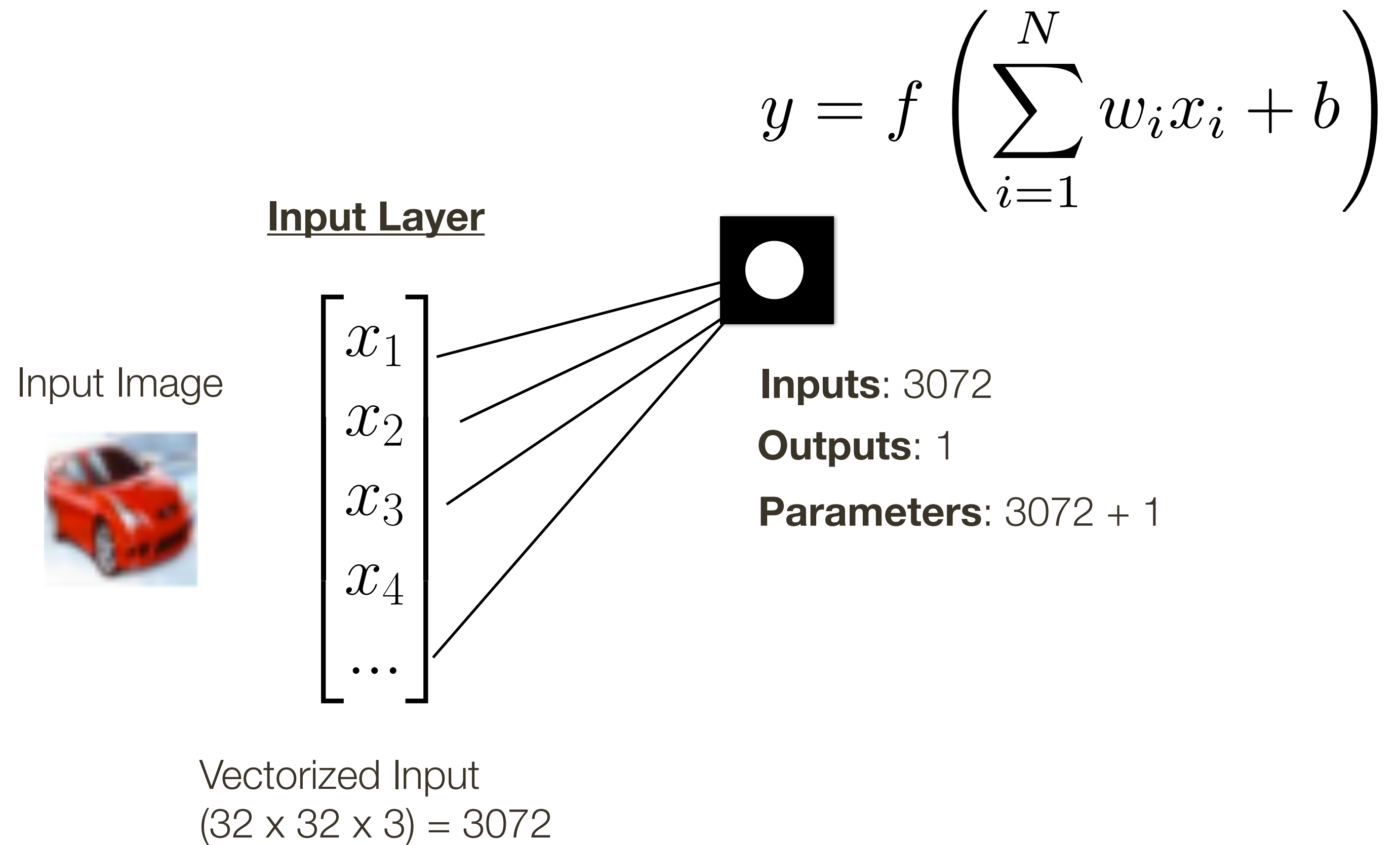


Neural Network: Short Review

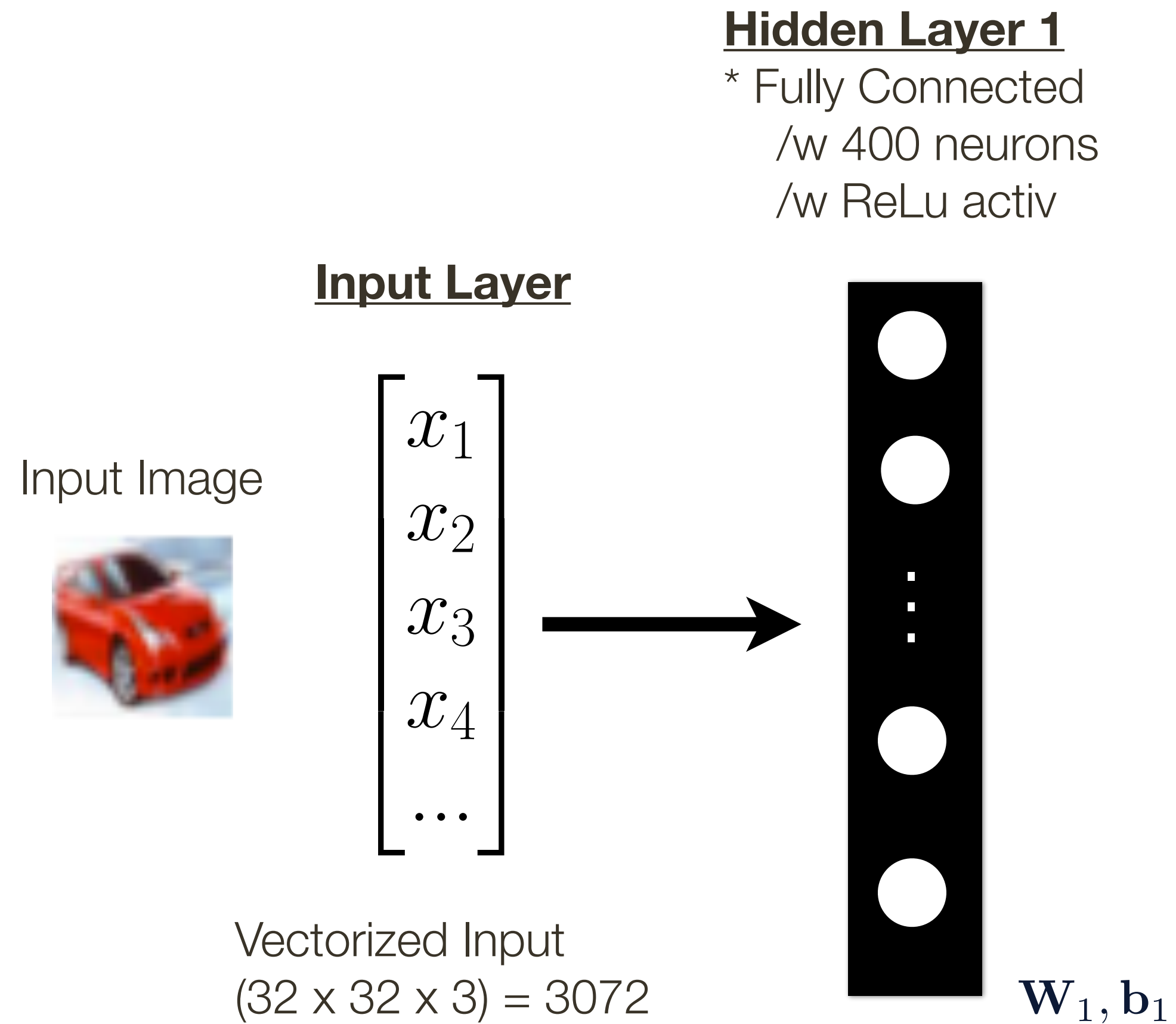
Neural Network: Short Review



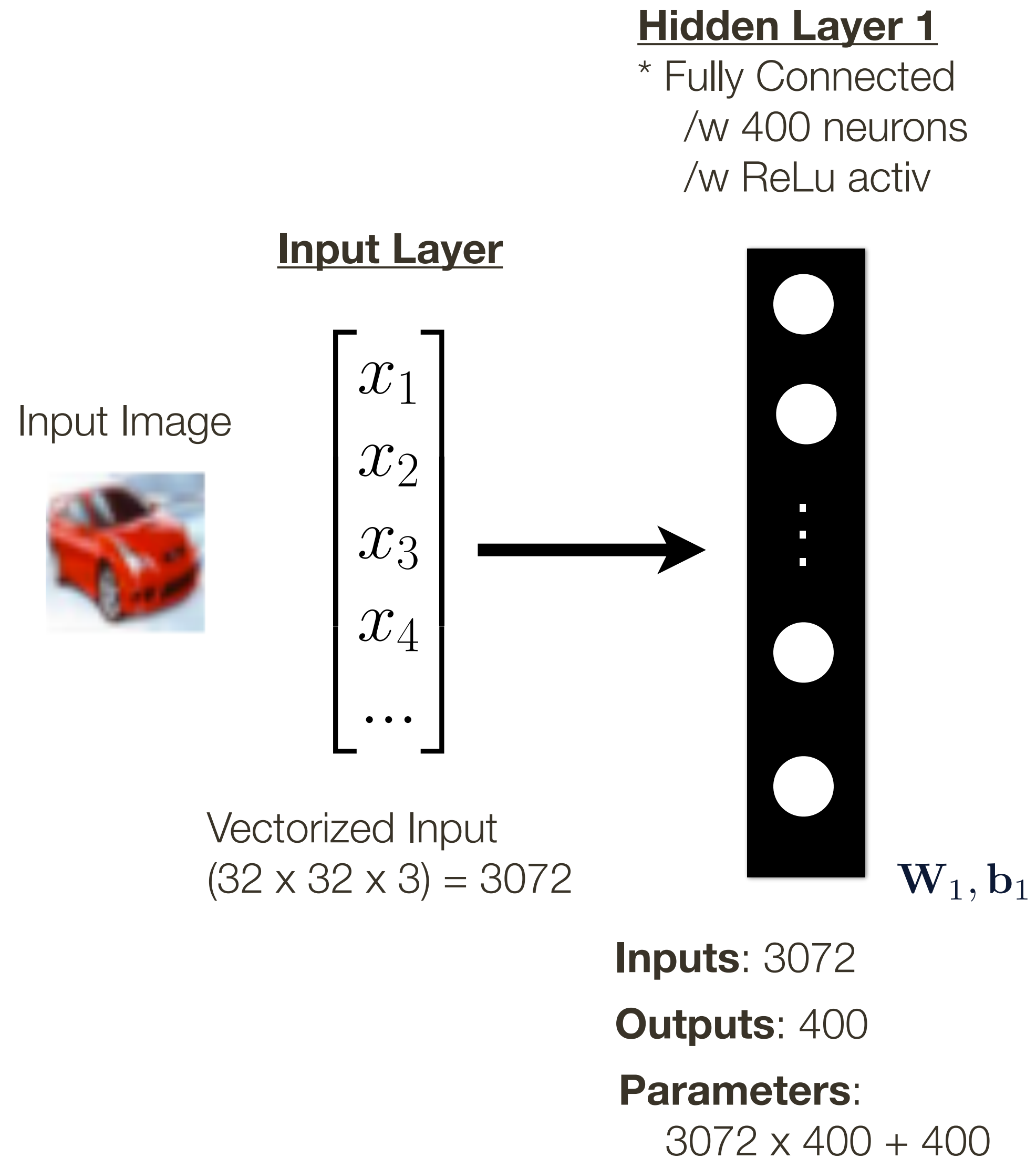
Neural Network: Short Review



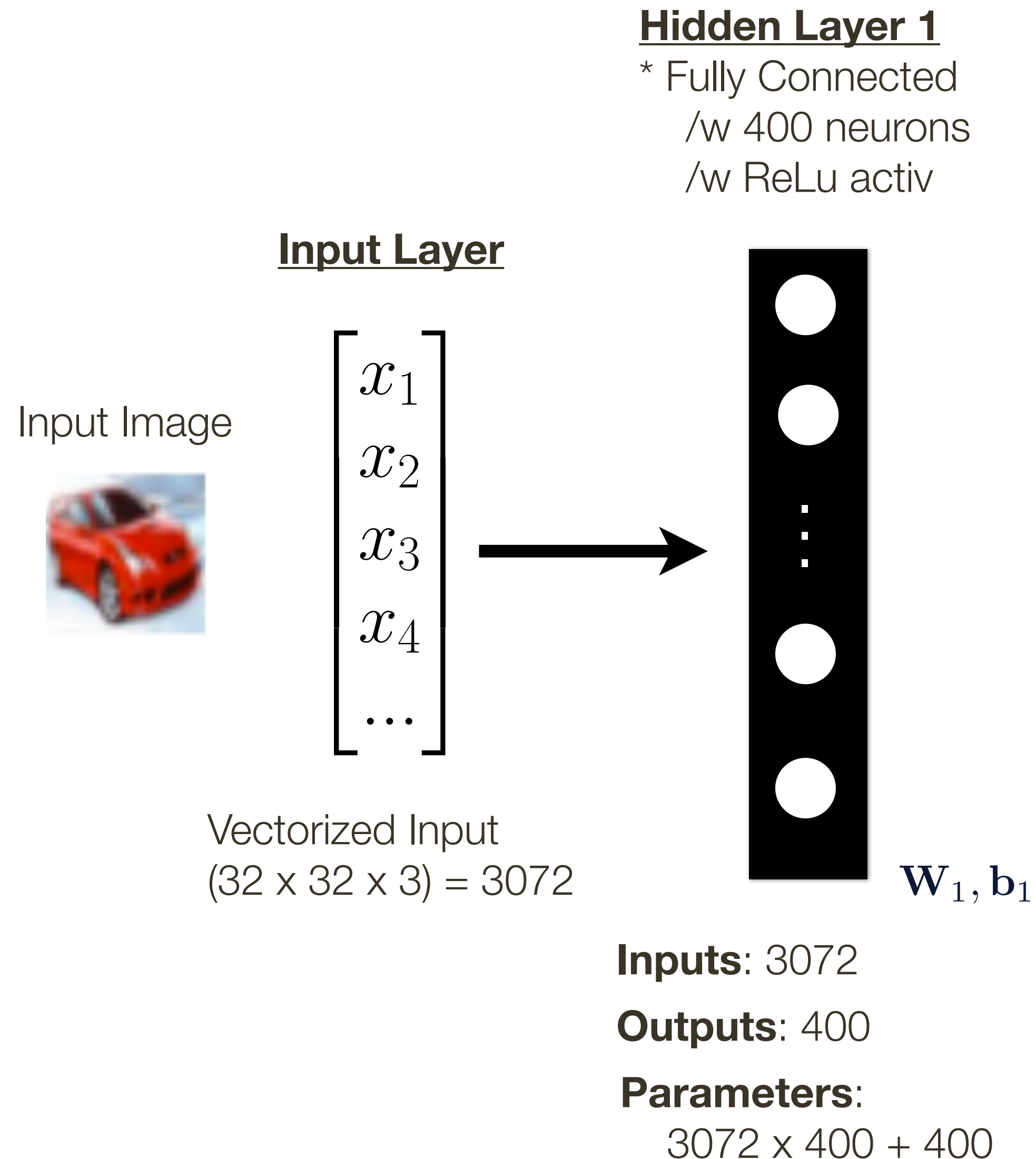
Neural Network: Short Review



Neural Network: Short Review

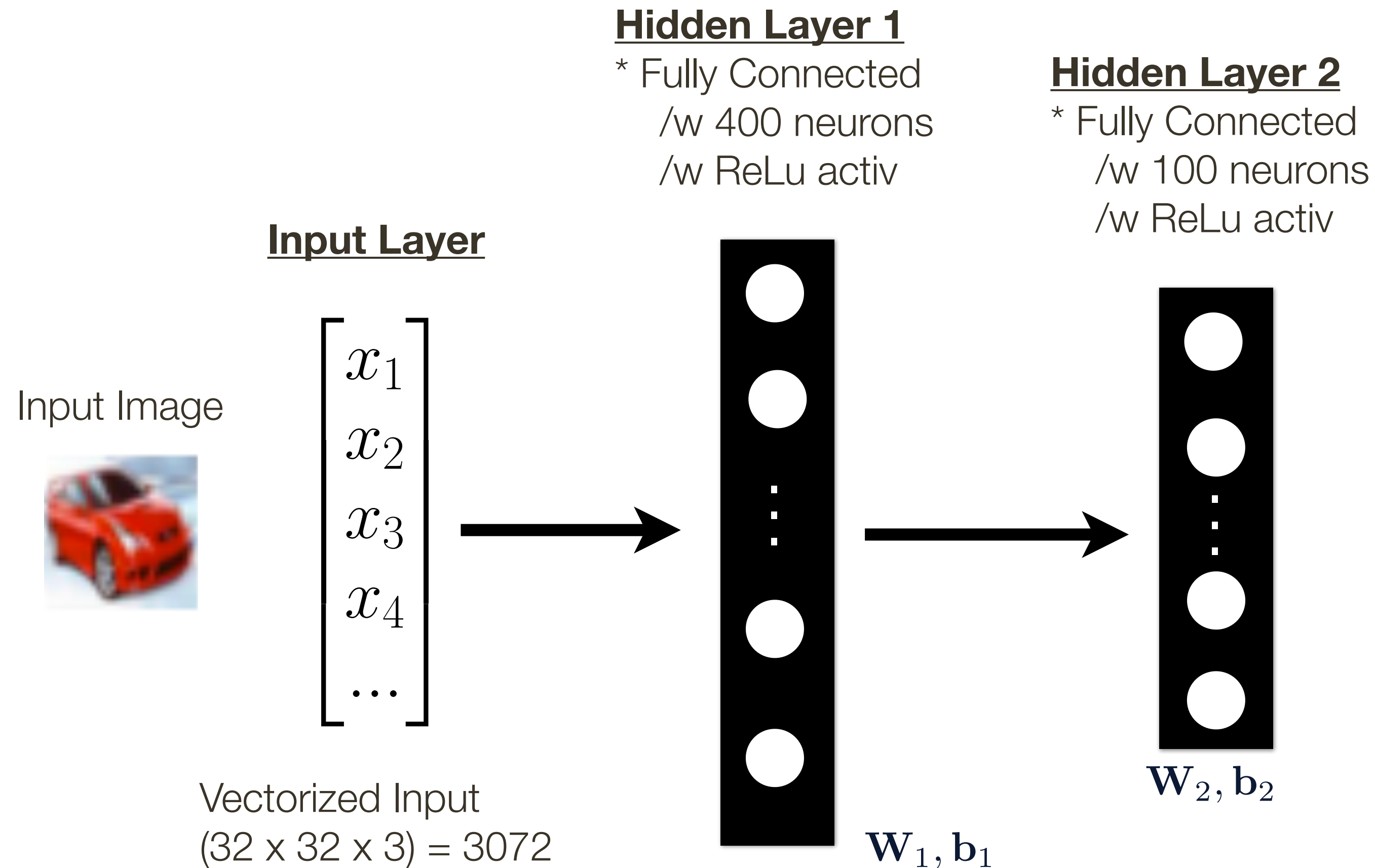


Neural Network: Short Review



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

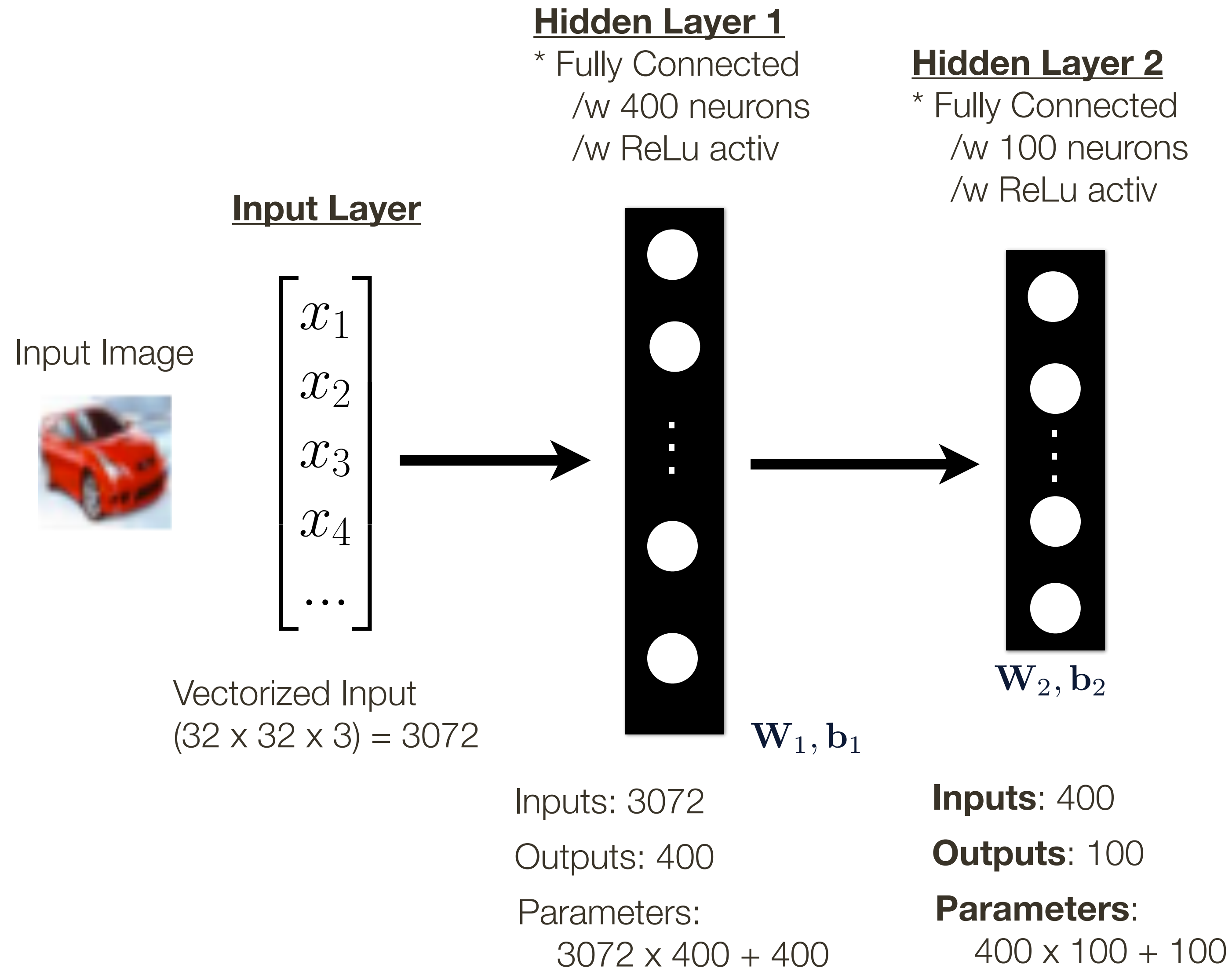
Inputs: 3072

Outputs: 400

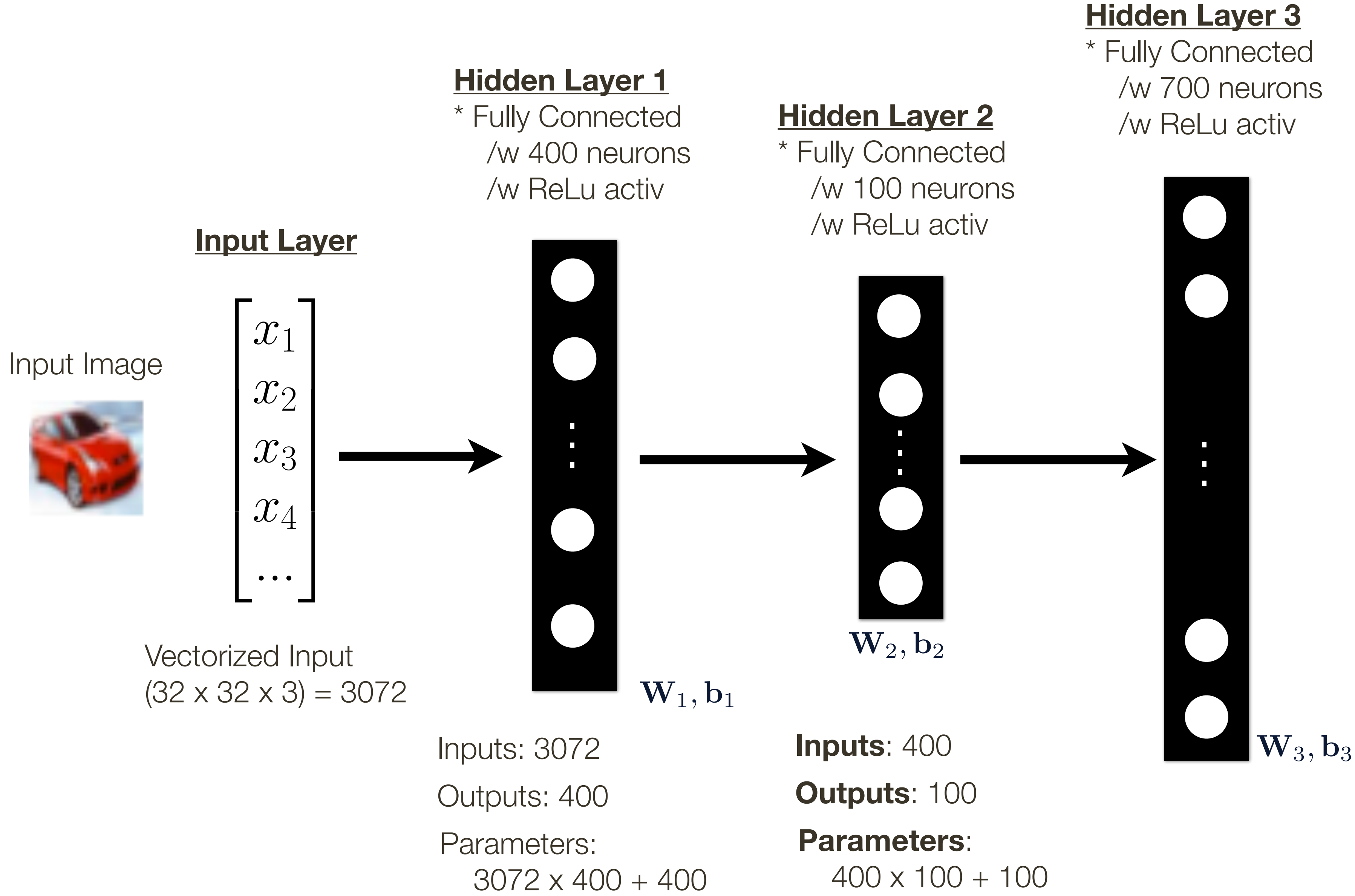
Parameters:

$$3072 \times 400 + 400$$

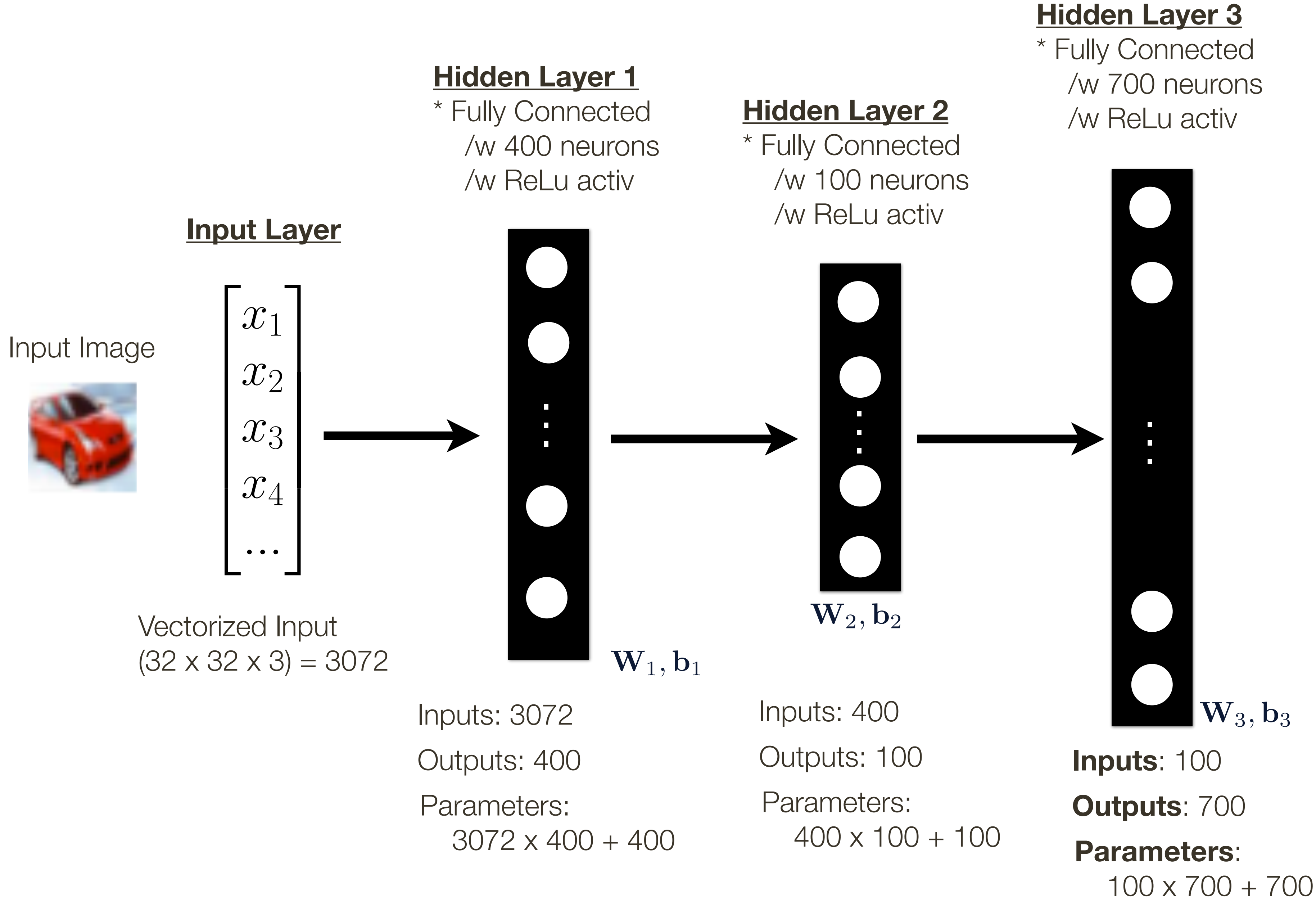
Neural Network: Short Review



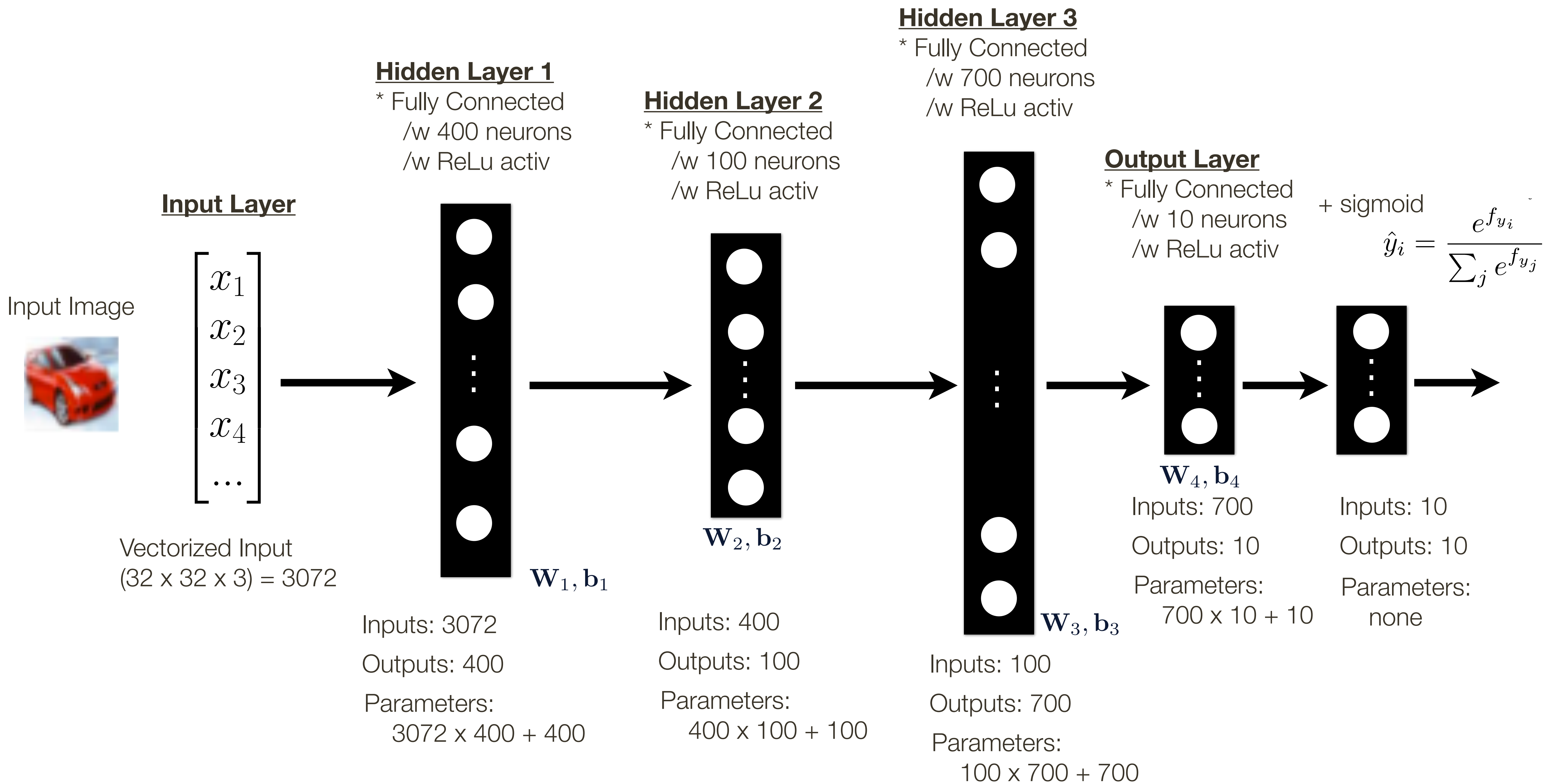
Neural Network: Short Review



Neural Network: Short Review

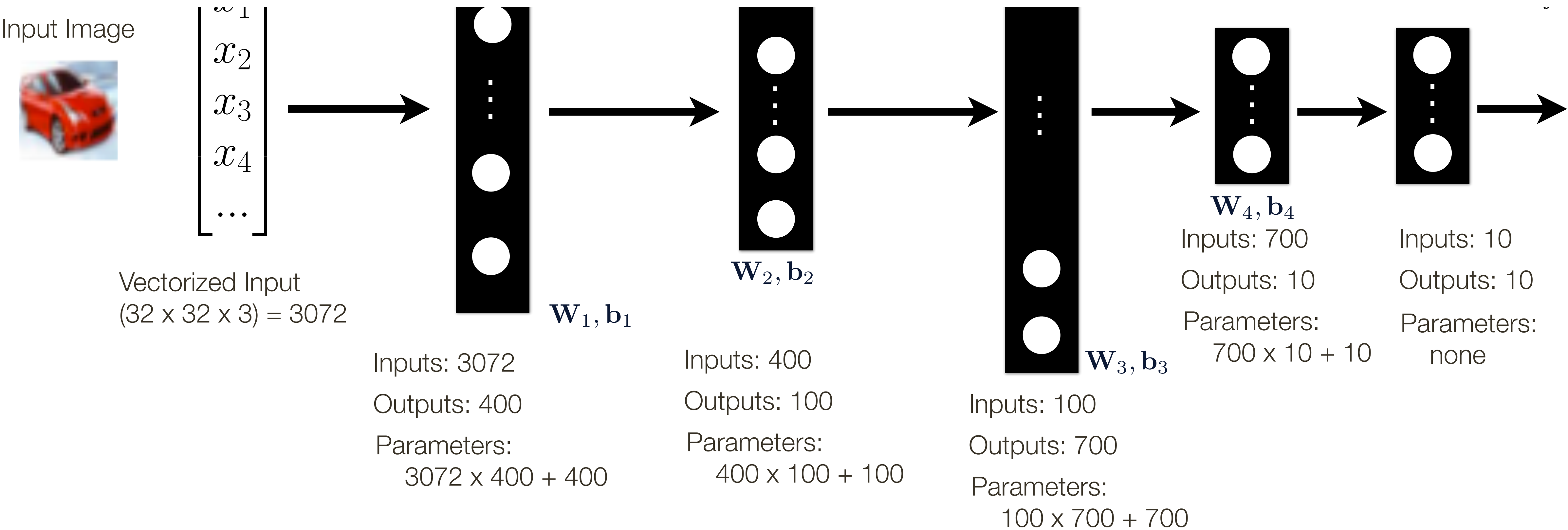


Neural Network: Short Review

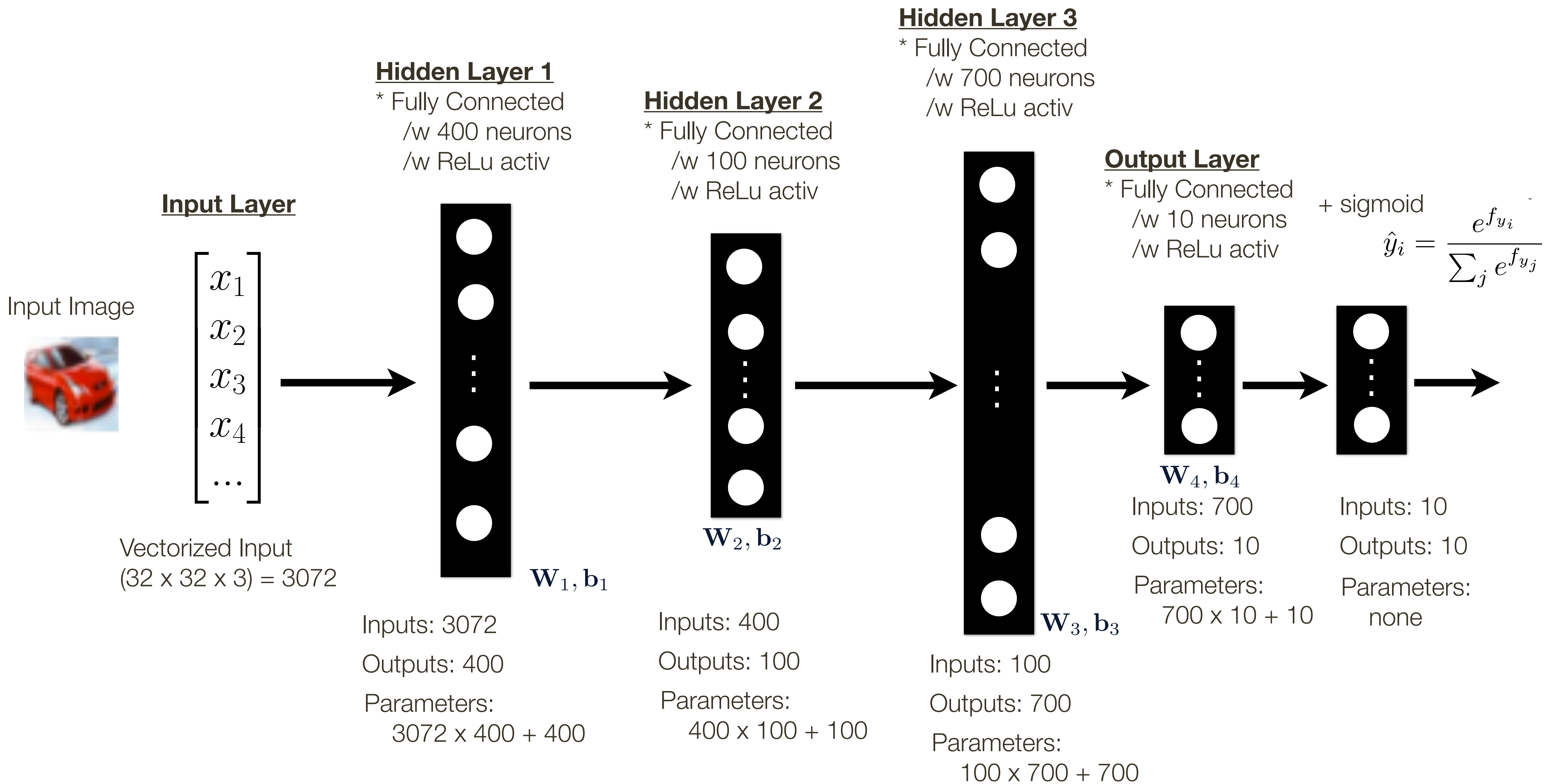


Neural Network: Short Review

This simple neural network has nearly 1.35 million parameters



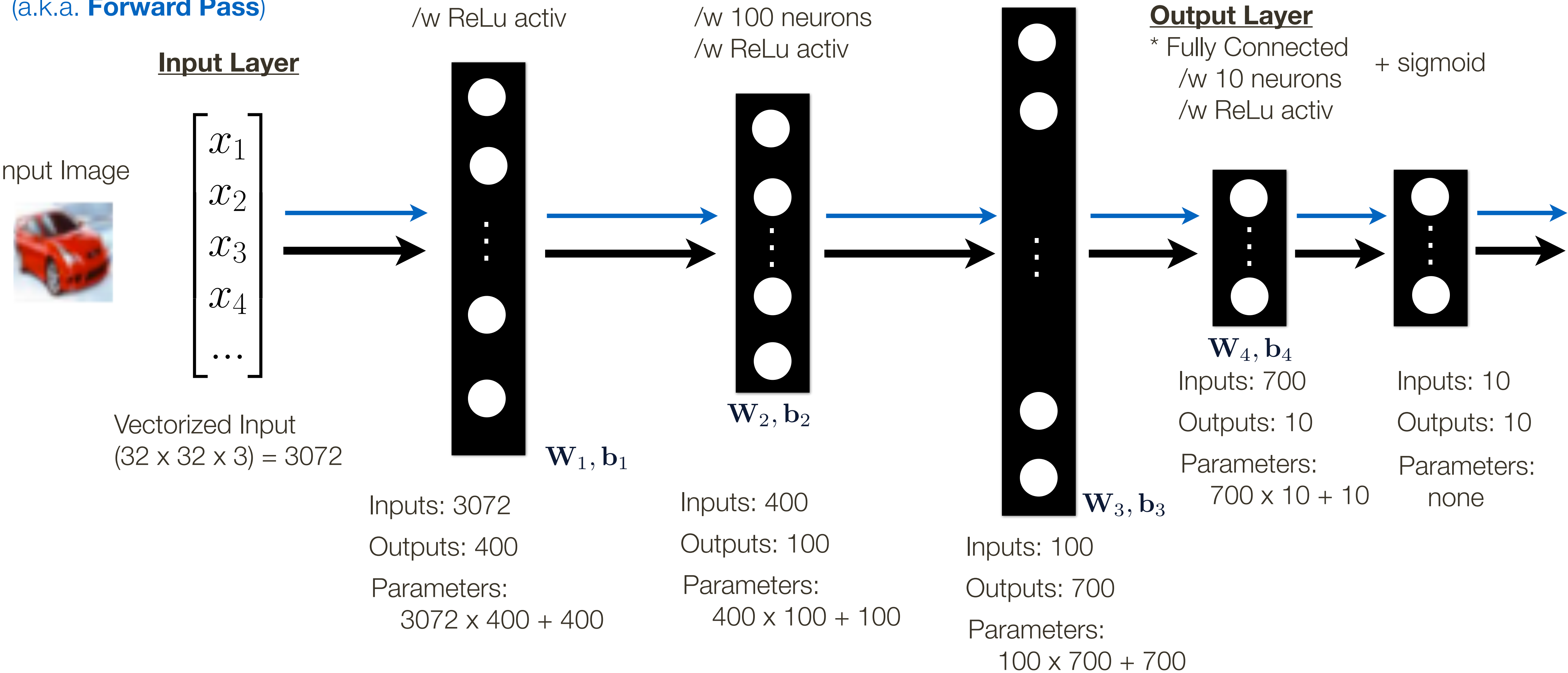
Neural Network: Short Review



Neural Network: Short Review

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

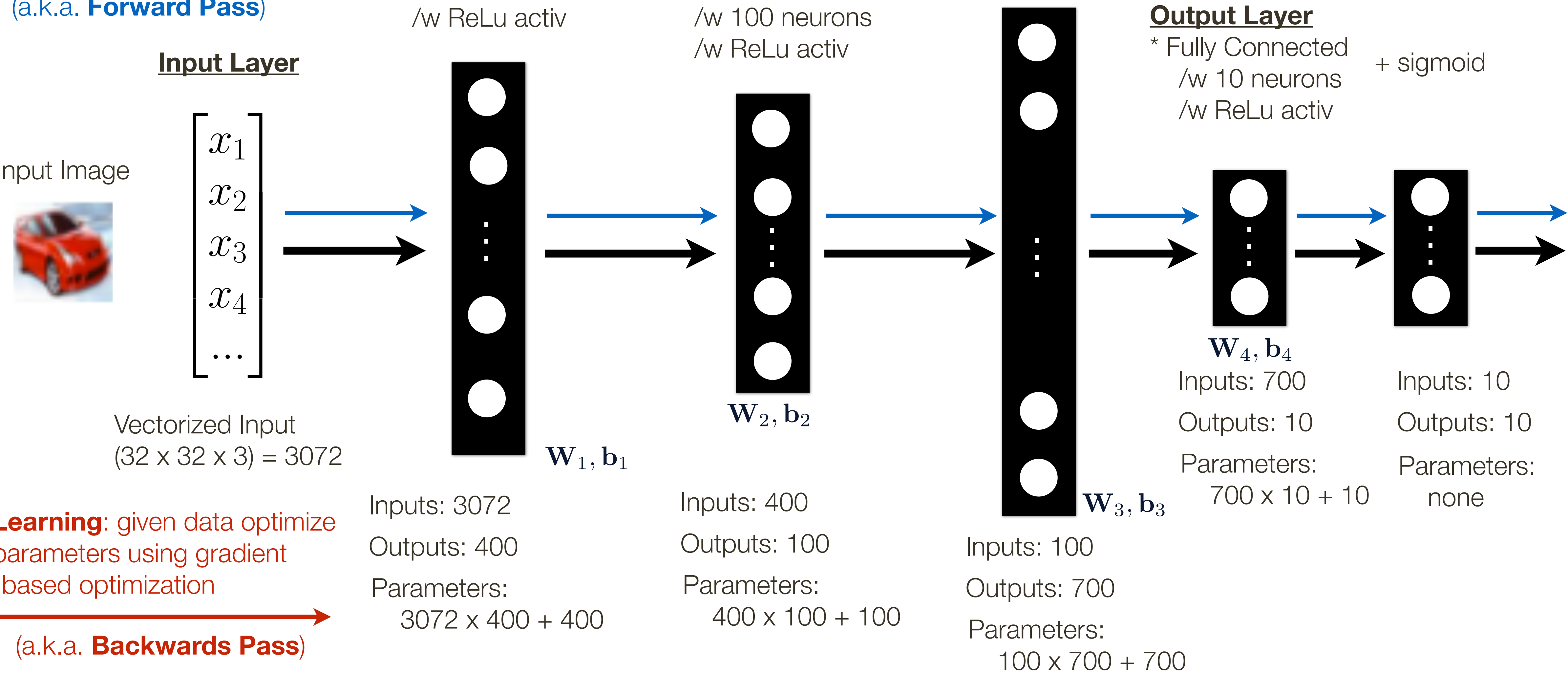
→
(a.k.a. **Forward Pass**)



Neural Network: Short Review

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

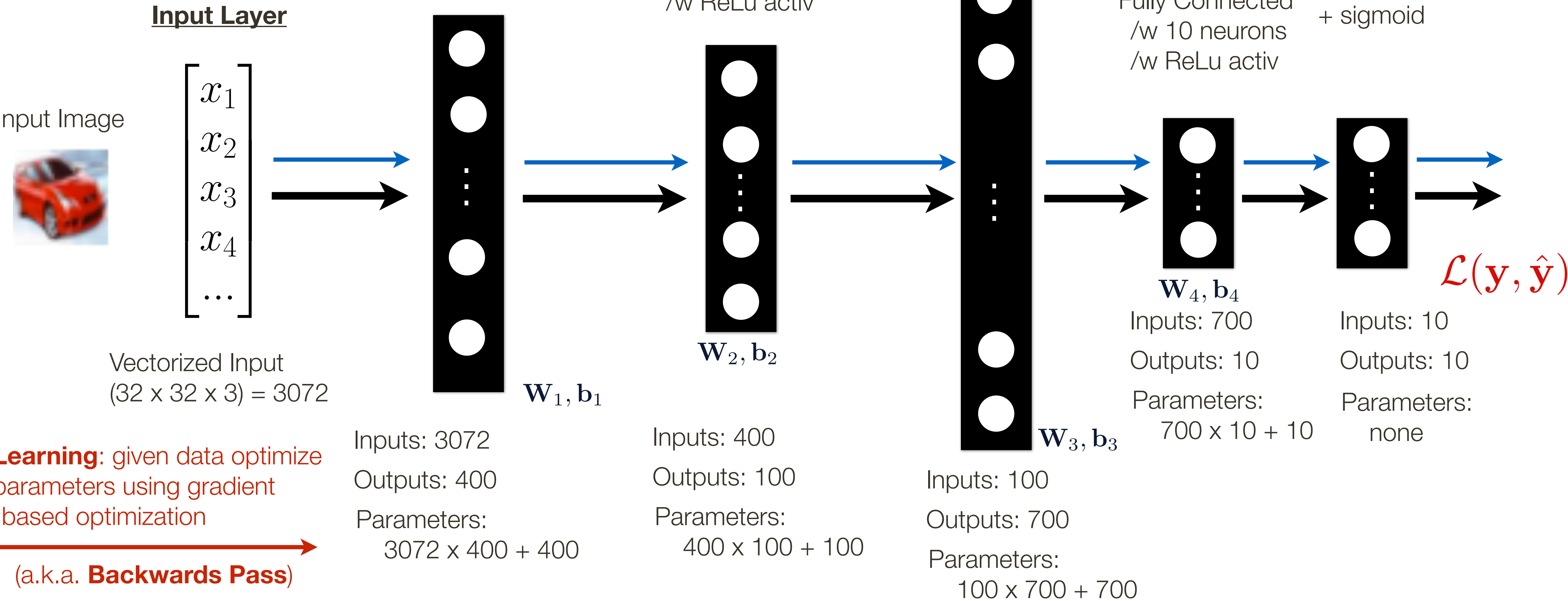
(a.k.a. **Forward Pass**)



Neural Network: Short Review

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

→
(a.k.a. **Forward Pass**)



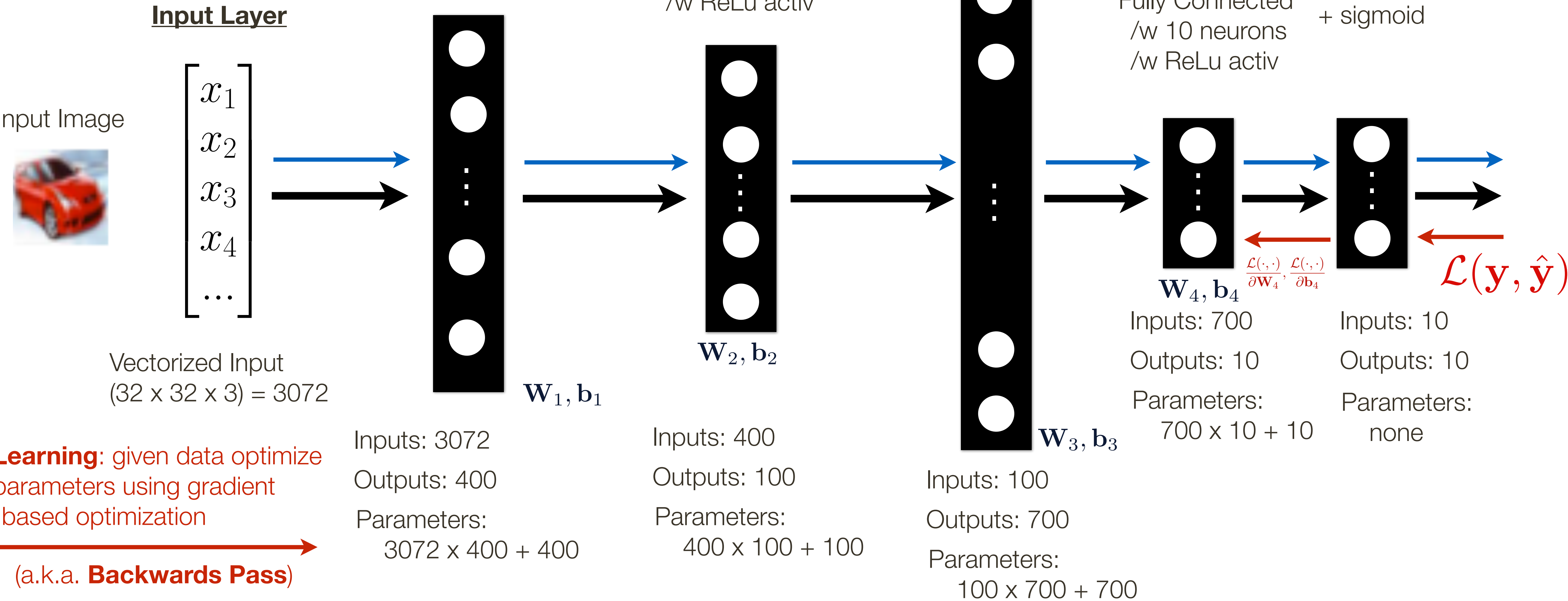
Learning: given data optimize
parameters using gradient
-based optimization

→
(a.k.a. **Backwards Pass**)

Neural Network: Short Review

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

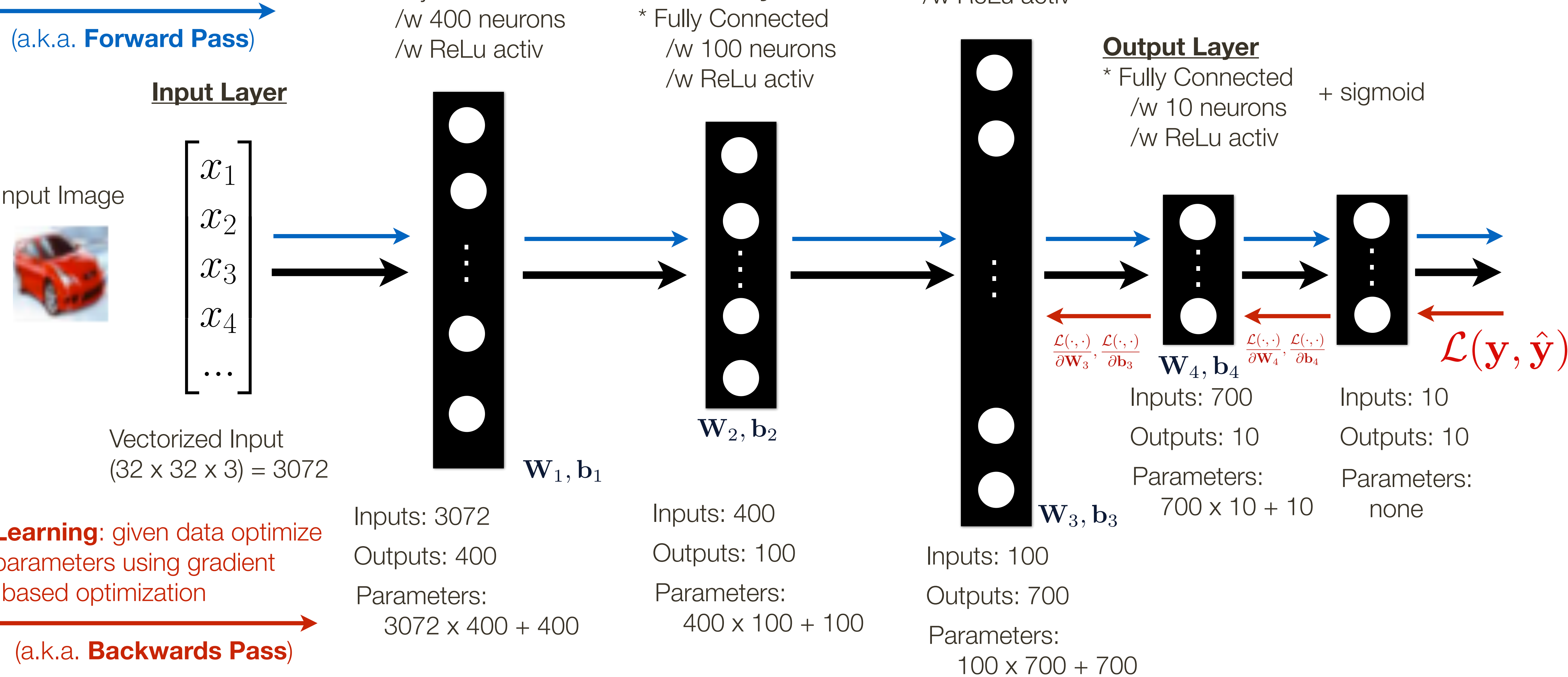
(a.k.a. **Forward Pass**)



Neural Network: Short Review

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

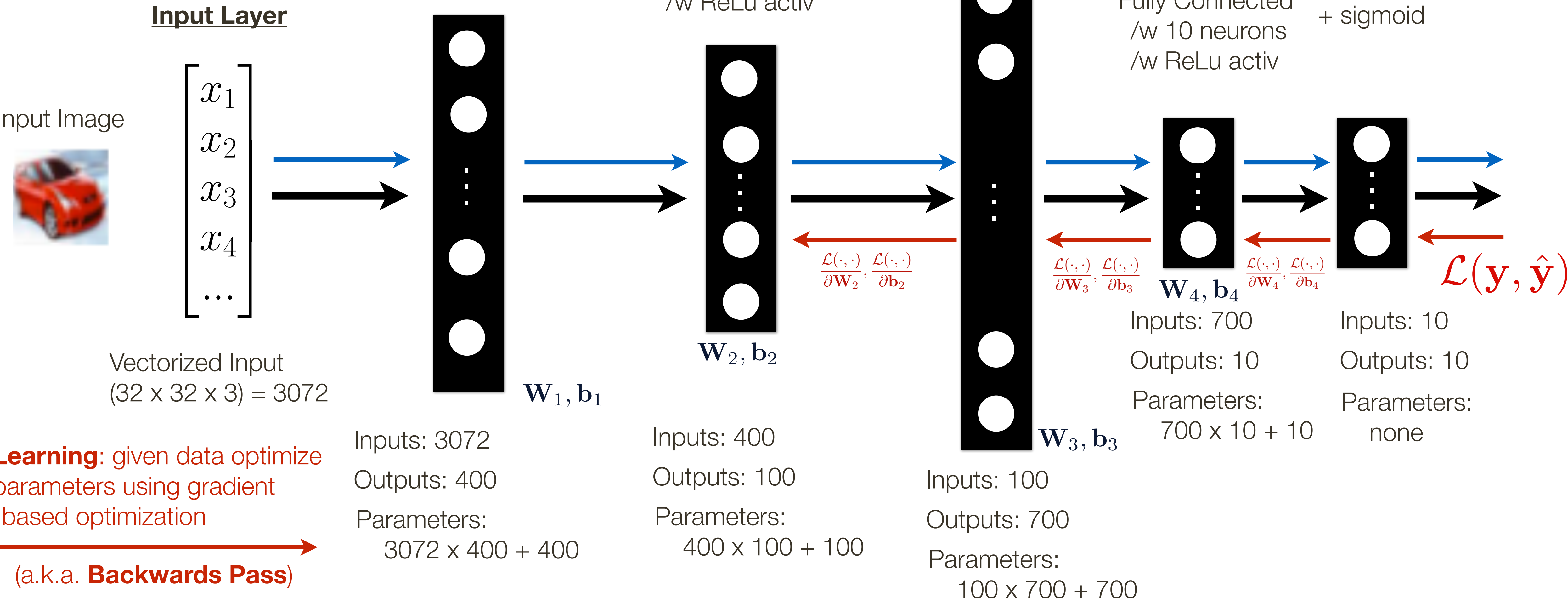
(a.k.a. **Forward Pass**)



Neural Network: Short Review

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

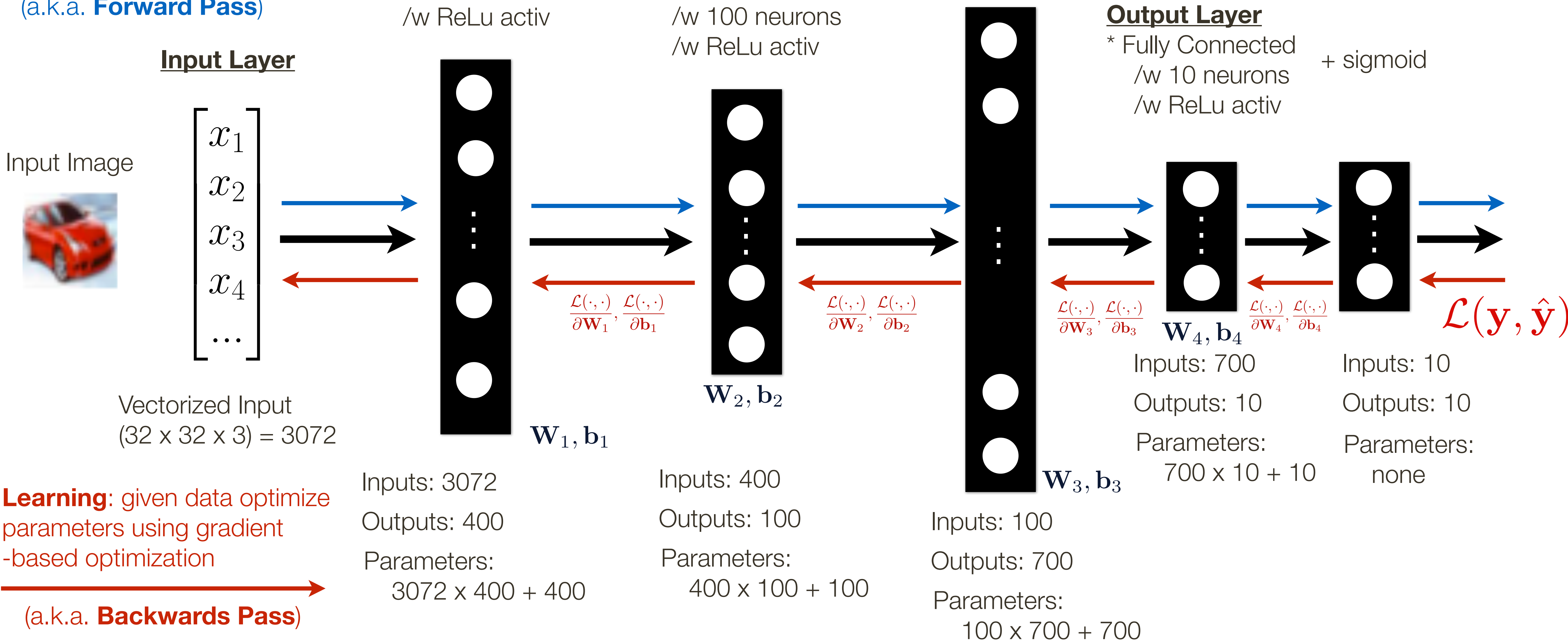
(a.k.a. **Forward Pass**)



Neural Network: Short Review

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

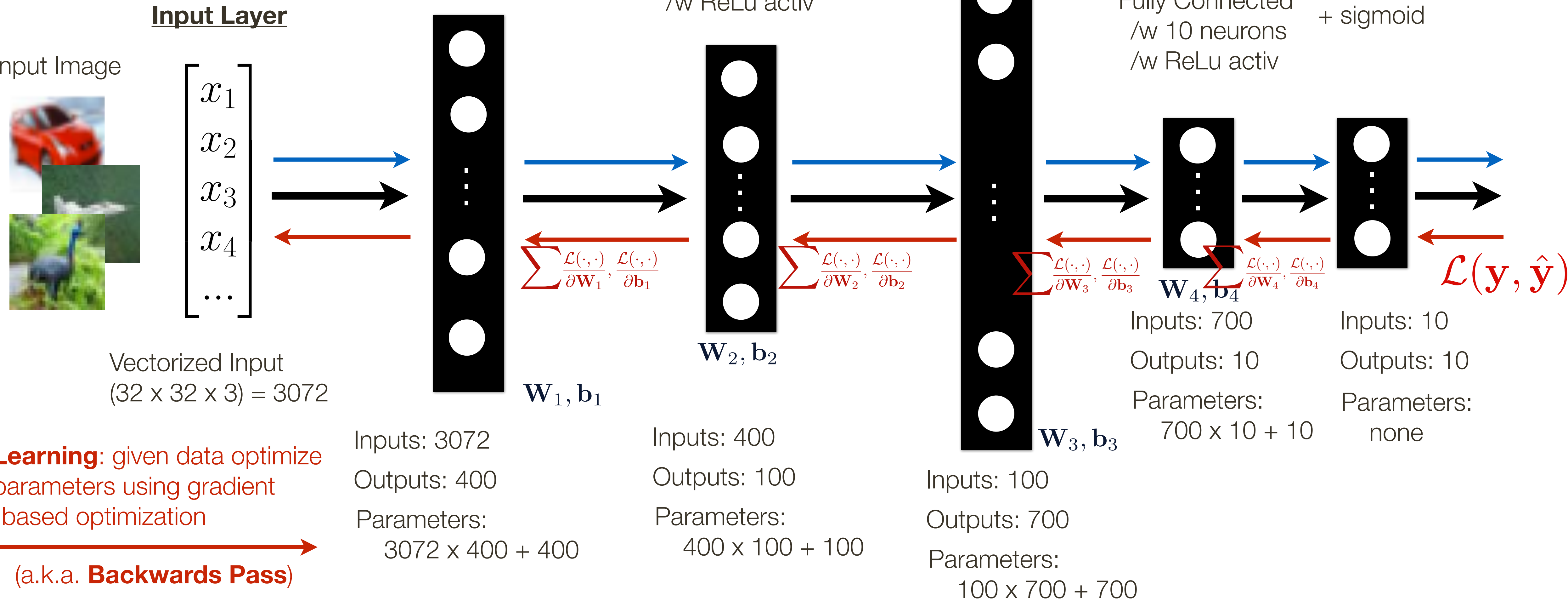
(a.k.a. **Forward Pass**)



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**)

Input Layer

Input Image

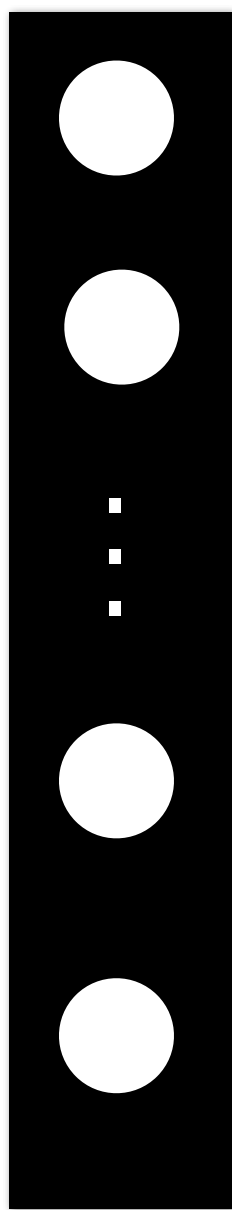


$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \end{bmatrix}$$

Vectorized Input
(32 x 32 x 3) = 3072

Hidden Layer 1

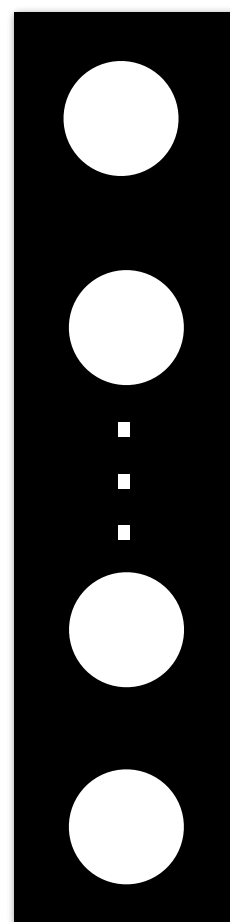
* Fully Connected
/w 400 neurons
/w ReLu activ



Inputs: 3072
Outputs: 400
Parameters:
3072 x 400 + 400

Hidden Layer 2

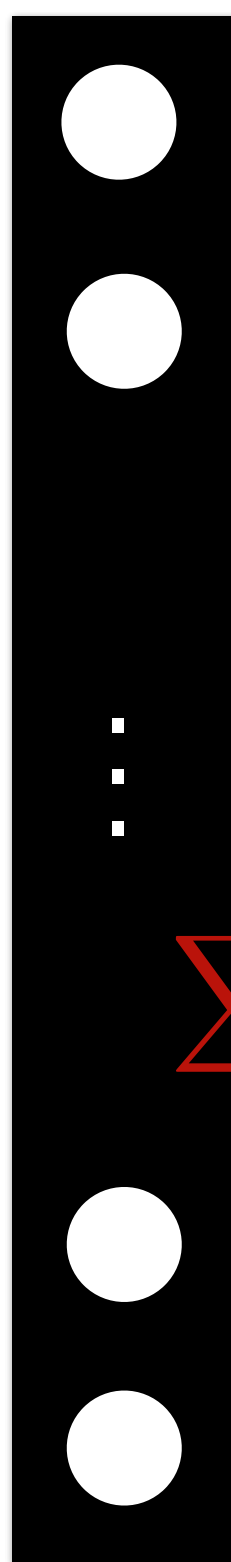
* Fully Connected
/w 100 neurons
/w ReLu activ



Inputs: 400
Outputs: 100
Parameters:
400 x 100 + 100

Hidden Layer 3

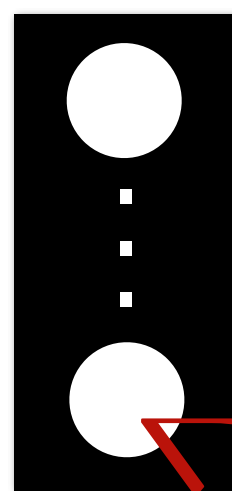
* Fully Connected
/w 700 neurons
/w ReLu activ



Inputs: 100
Outputs: 700
Parameters:
100 x 700 + 700

Output Layer

* Fully Connected
/w 10 neurons
/w ReLu activ + sigmoid



Inputs: 700
Outputs: 10
Parameters:
700 x 10 + 10

Inputs: 10
Outputs: 10
Parameters:
none

$$\sum \frac{\mathcal{L}(\cdot, \cdot)}{\partial \mathbf{W}_1}, \frac{\mathcal{L}(\cdot, \cdot)}{\partial \mathbf{b}_1}$$

$$\sum \frac{\mathcal{L}(\cdot, \cdot)}{\partial \mathbf{W}_2}, \frac{\mathcal{L}(\cdot, \cdot)}{\partial \mathbf{b}_2}$$

$$\sum \frac{\mathcal{L}(\cdot, \cdot)}{\partial \mathbf{W}_3}, \frac{\mathcal{L}(\cdot, \cdot)}{\partial \mathbf{b}_3}$$

$$\sum \frac{\mathcal{L}(\cdot, \cdot)}{\partial \mathbf{W}_4}, \frac{\mathcal{L}(\cdot, \cdot)}{\partial \mathbf{b}_4}$$

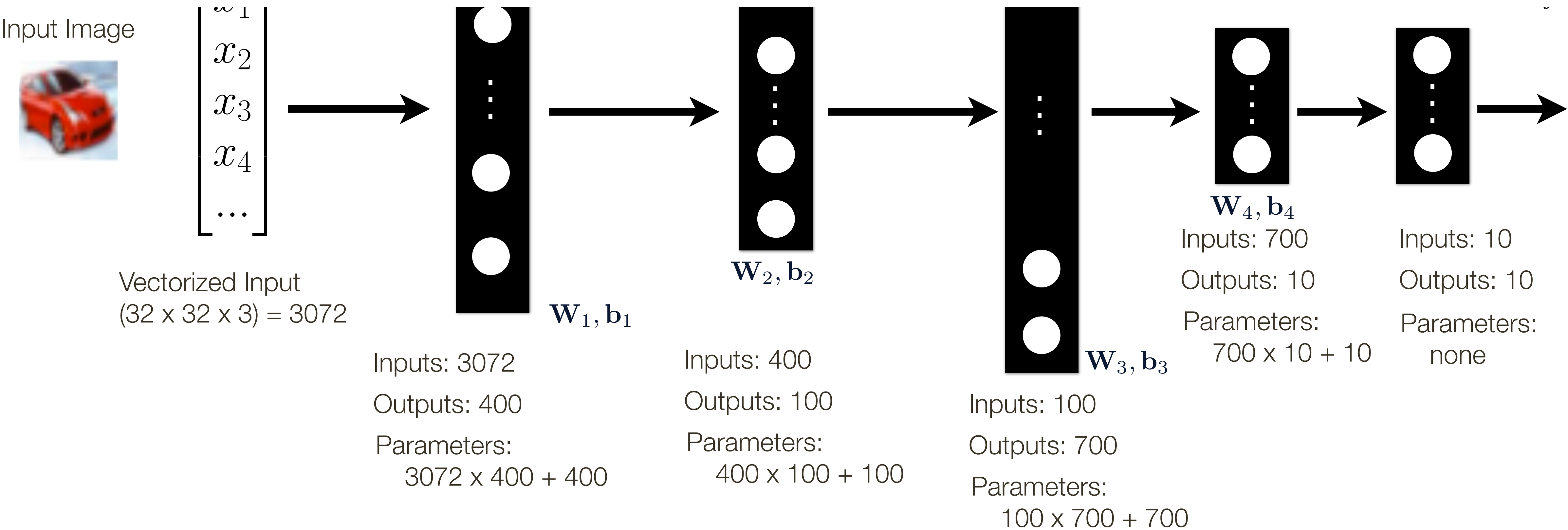
$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$$

Learning: given data optimize parameters using gradient-based optimization

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

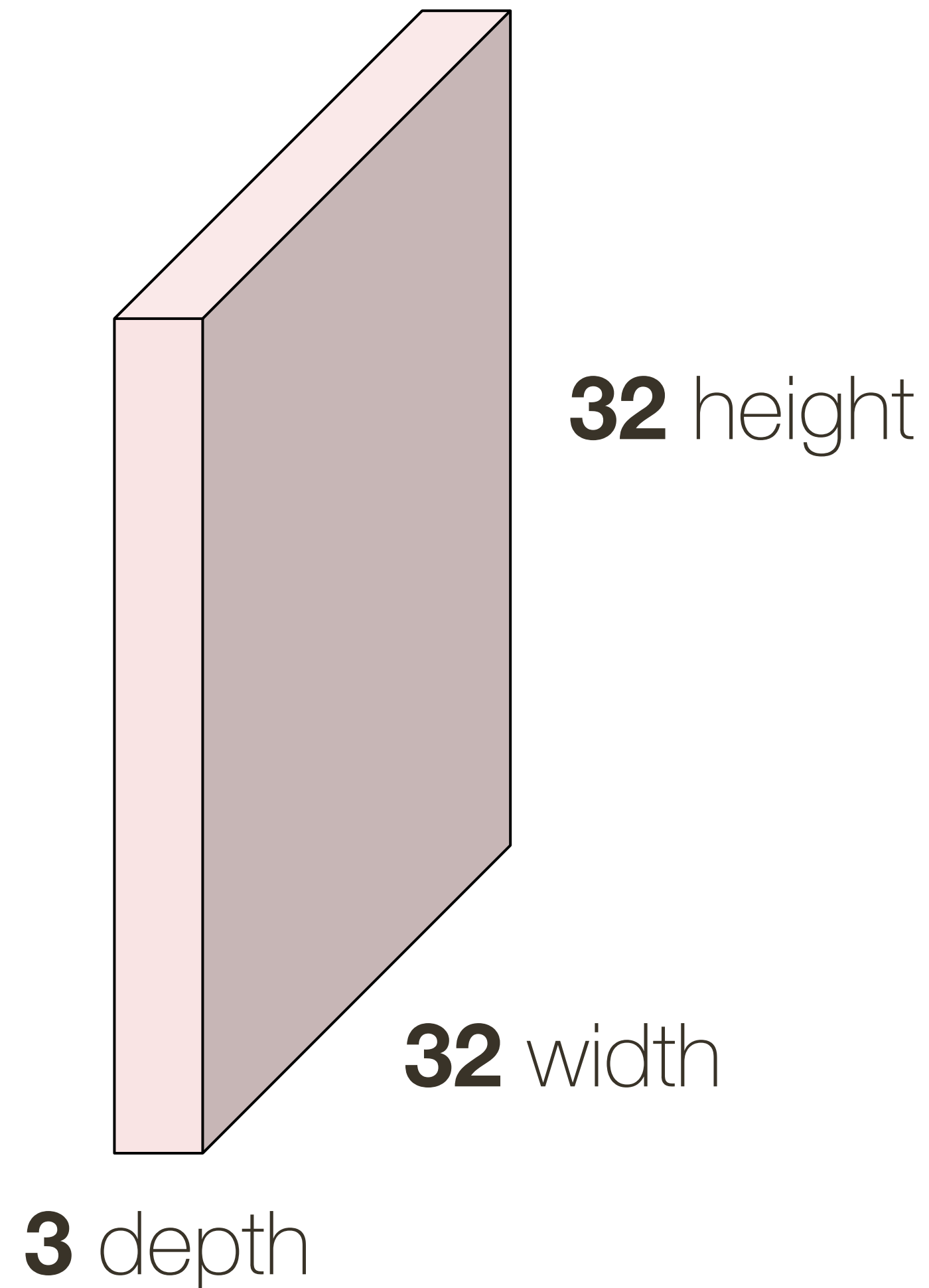
Neural Network: Short Review

This simple neural network has nearly 1.35 million parameters



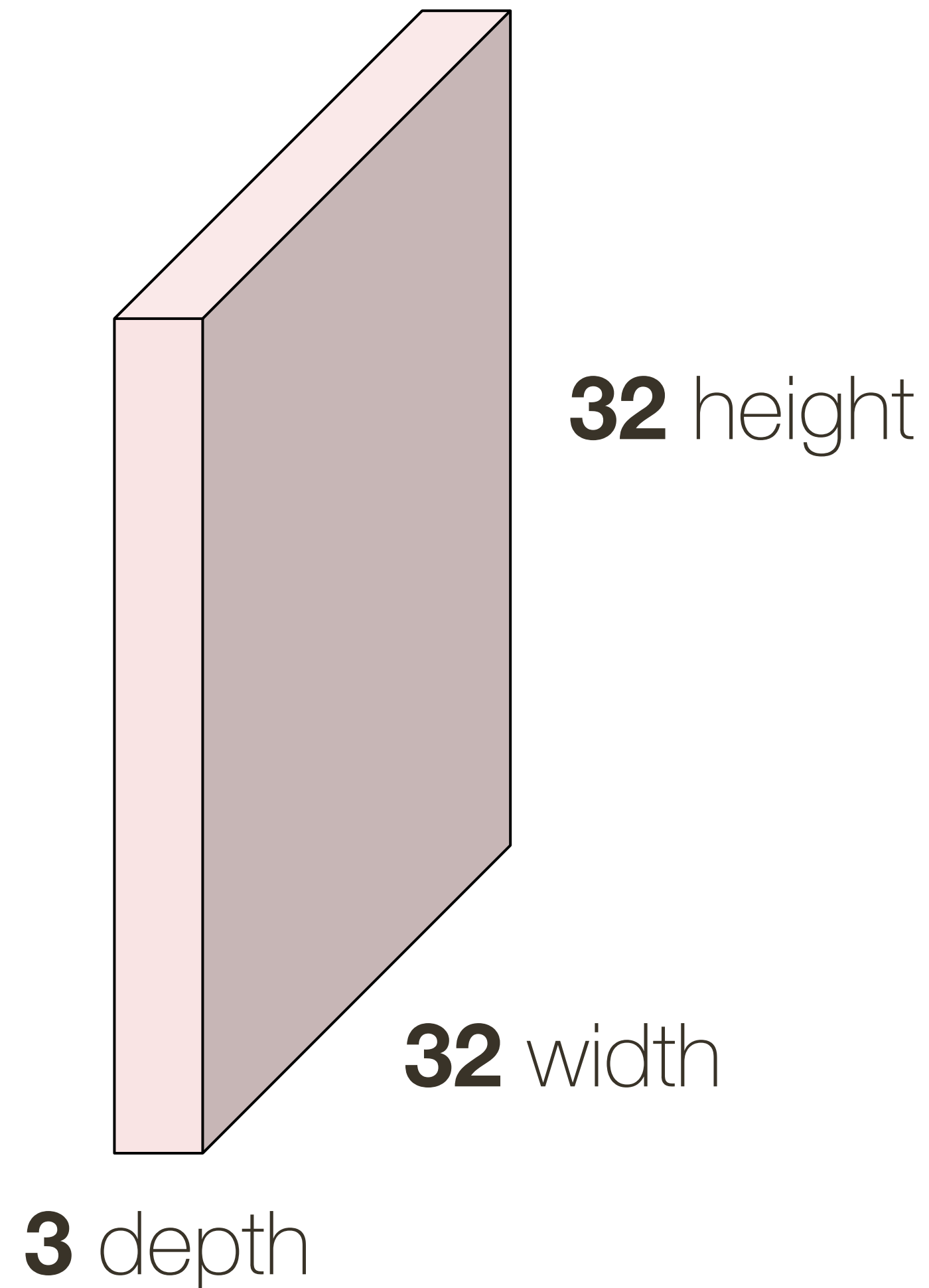
Convolutional Layer

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



Convolutional Layer

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



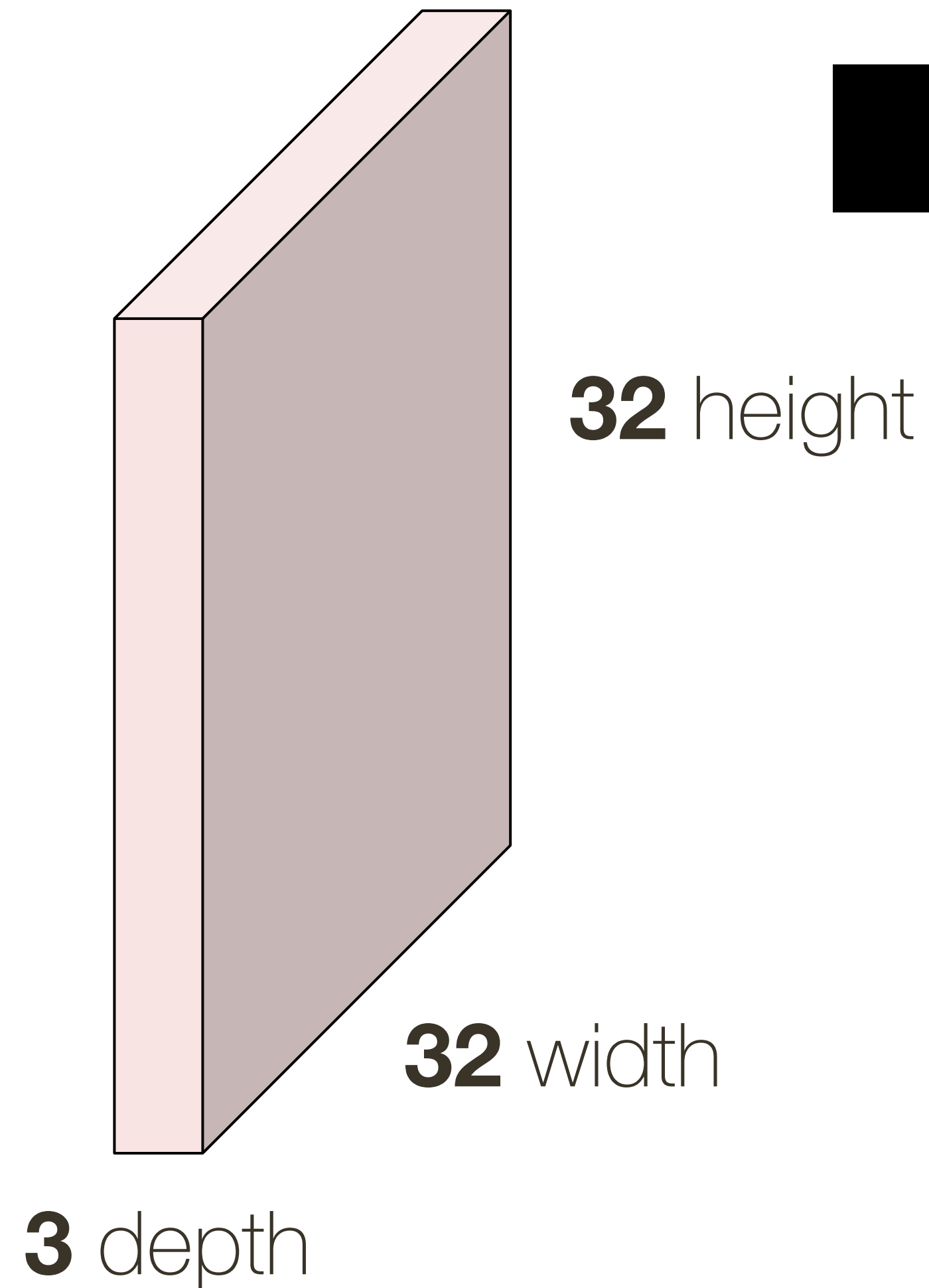
$5 \times 5 \times 3$ **filter**



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

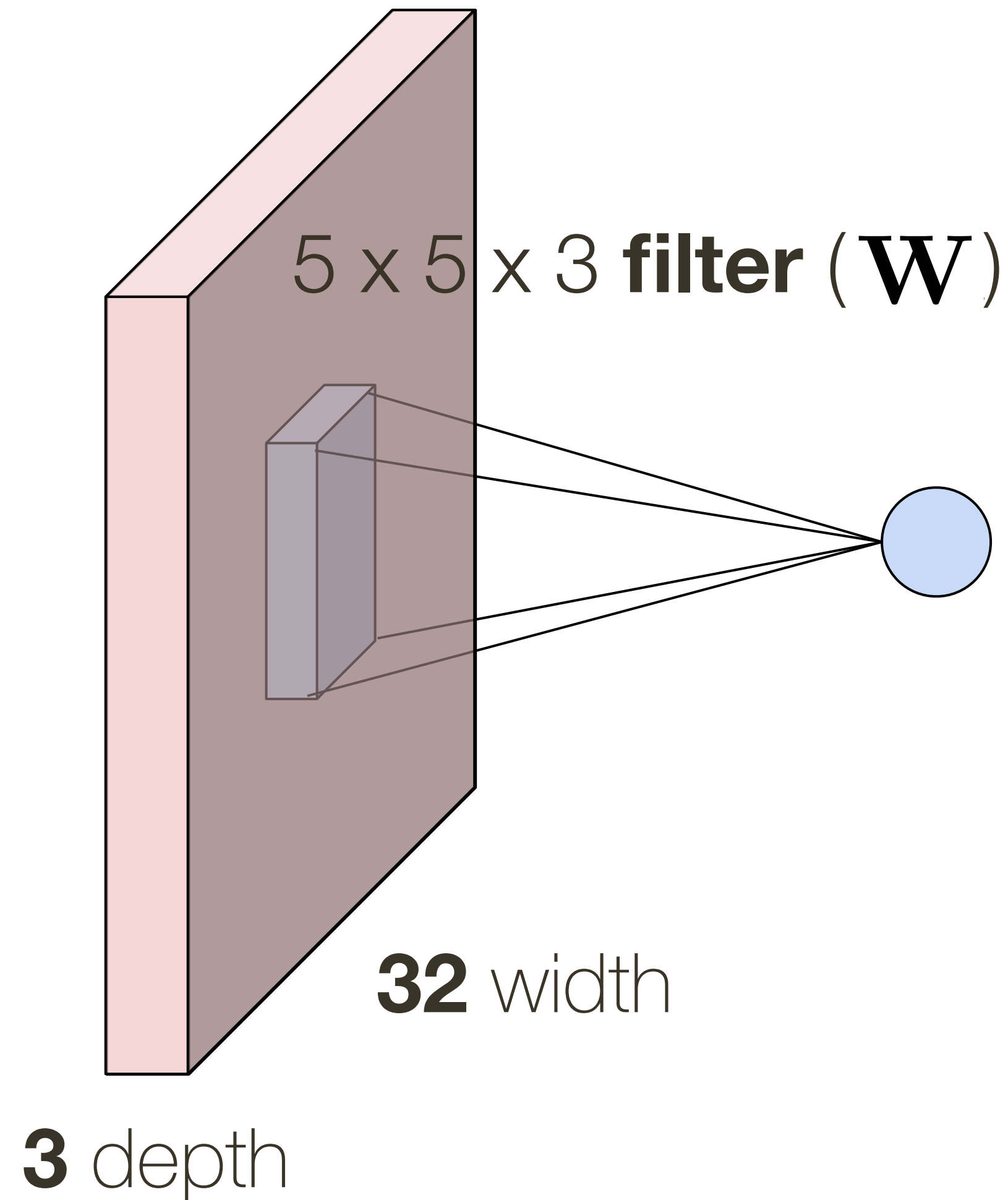
5 x 5 x **3** filter



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

Convolutional Layer

32 x 32 x 3 **image**

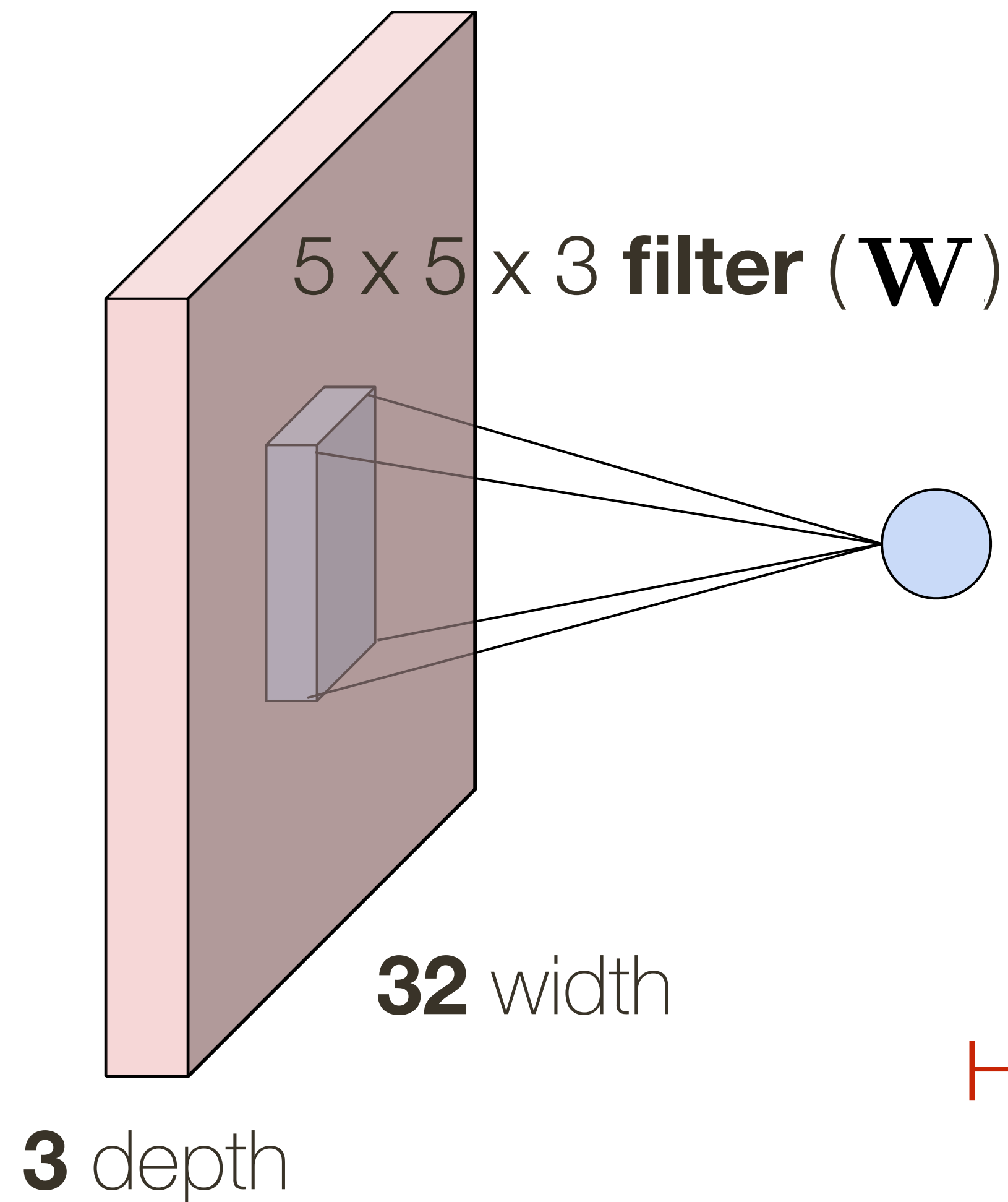


1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

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

Convolutional Layer

32 x 32 x 3 **image**



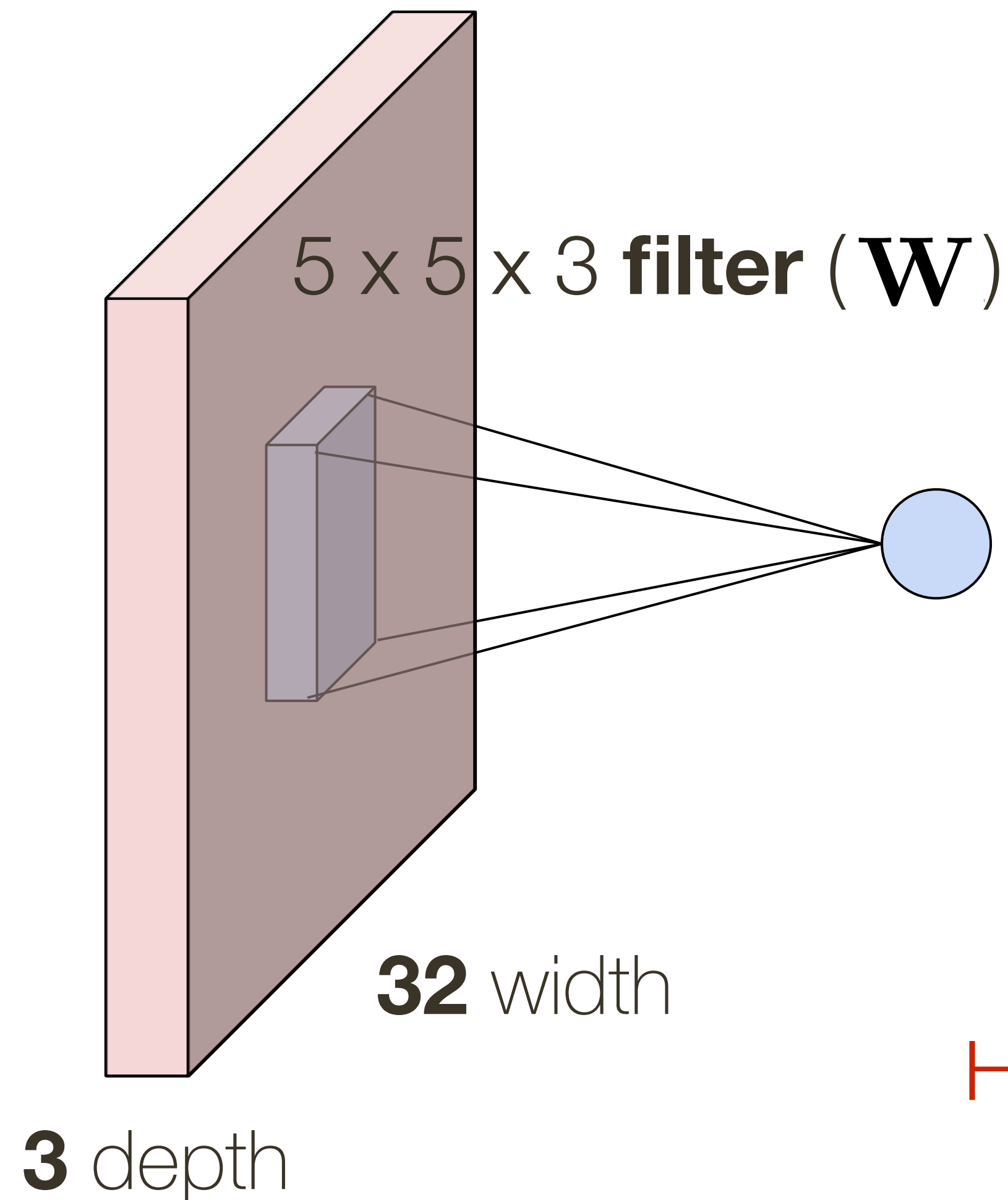
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

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

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

Convolutional Layer

32 x 32 x 3 **image**



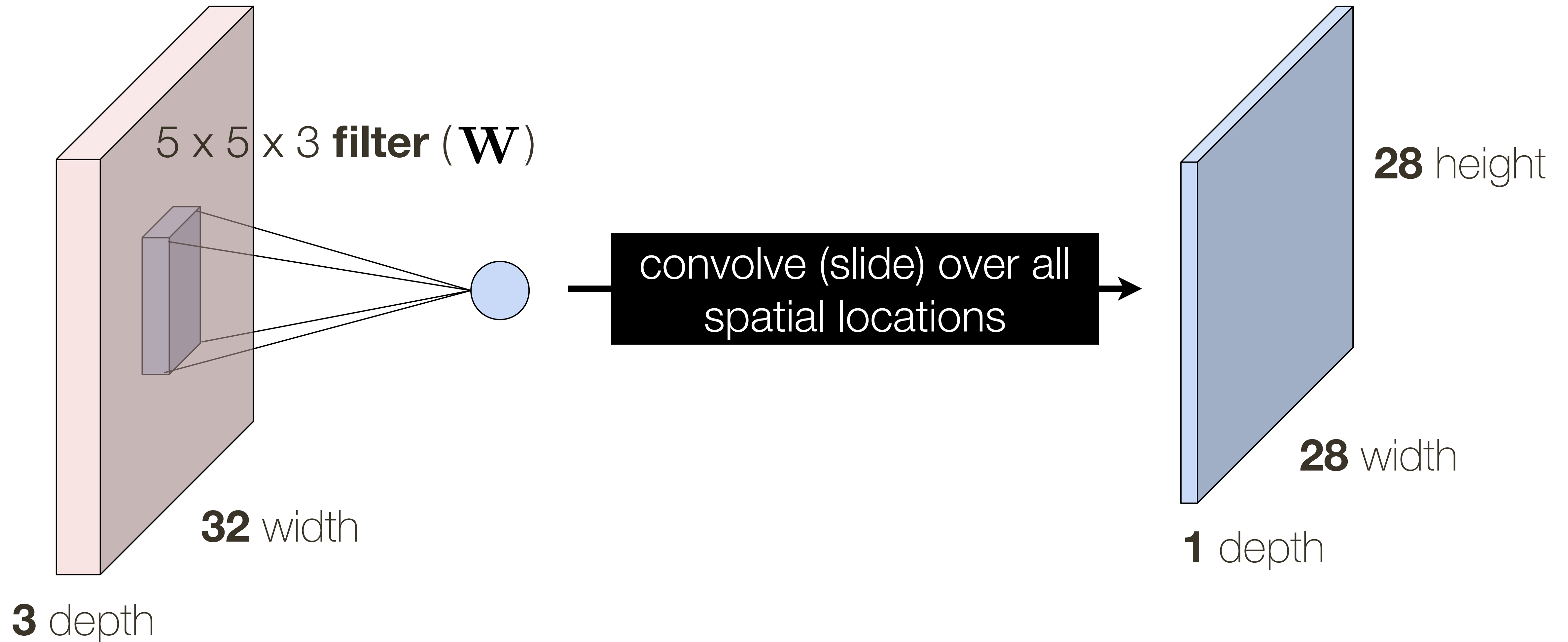
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

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

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

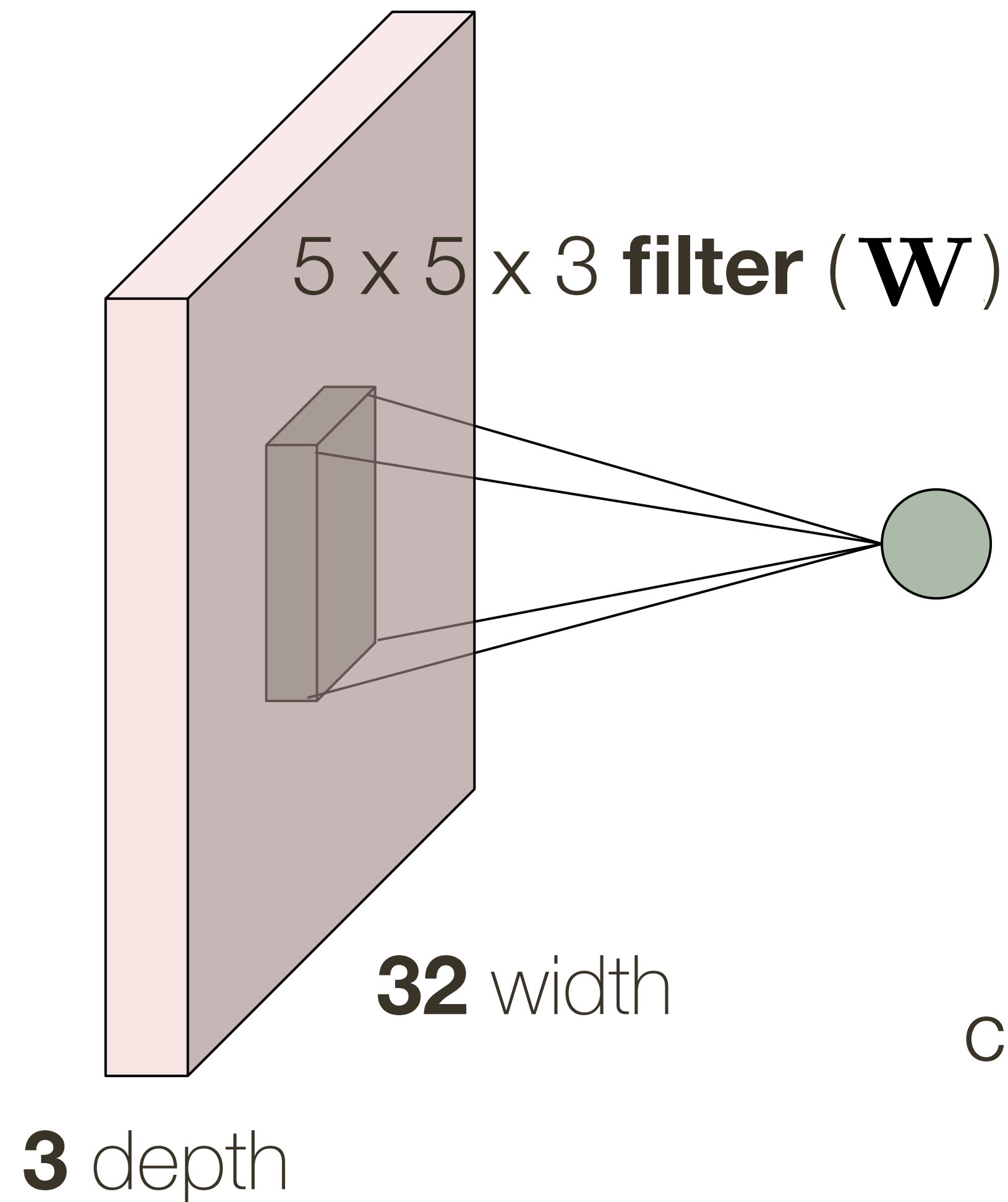
Convolutional Layer

$32 \times 32 \times 3$ image



Convolutional Layer

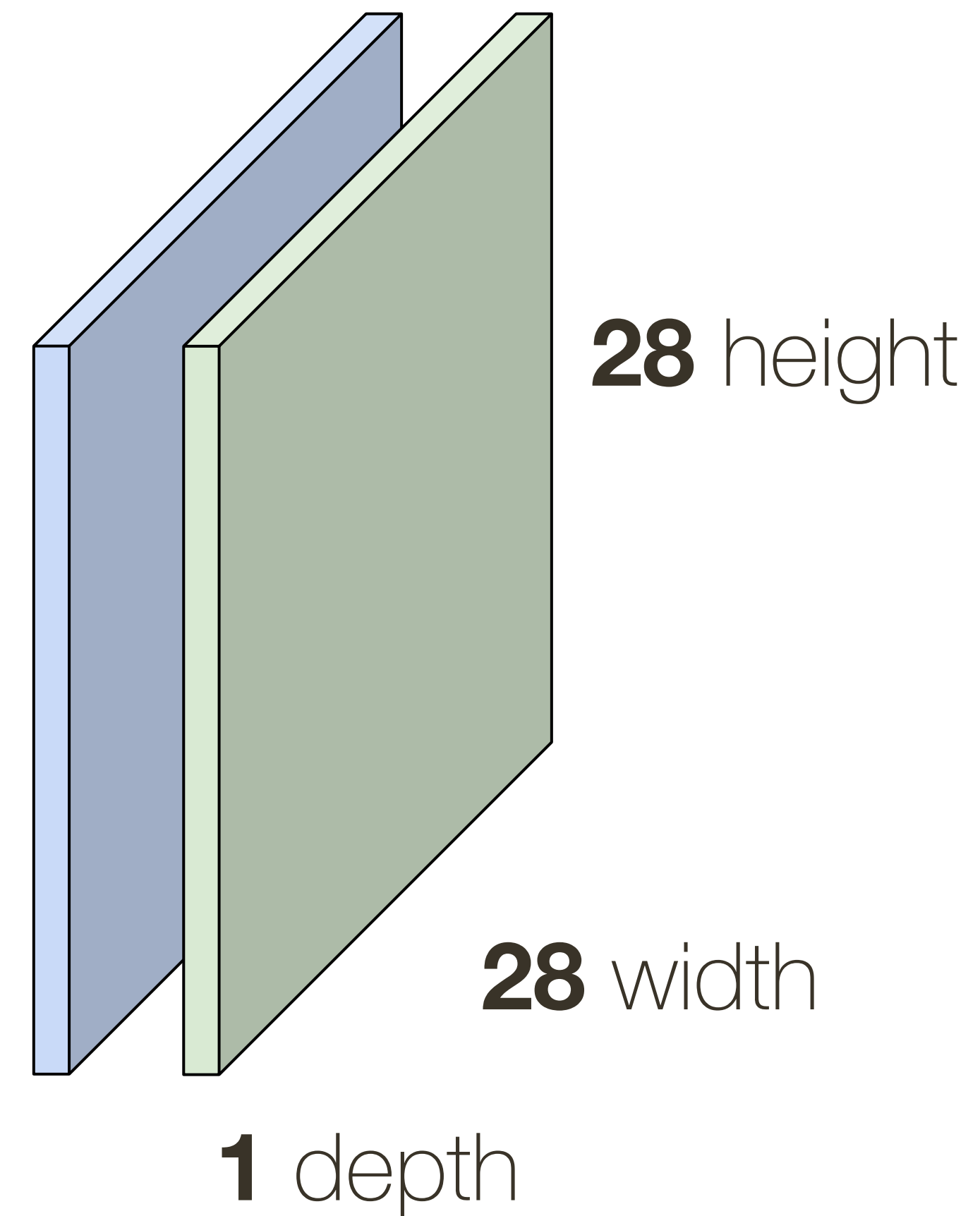
32 x 32 x 3 **image**



convolve (slide) over all
spatial locations

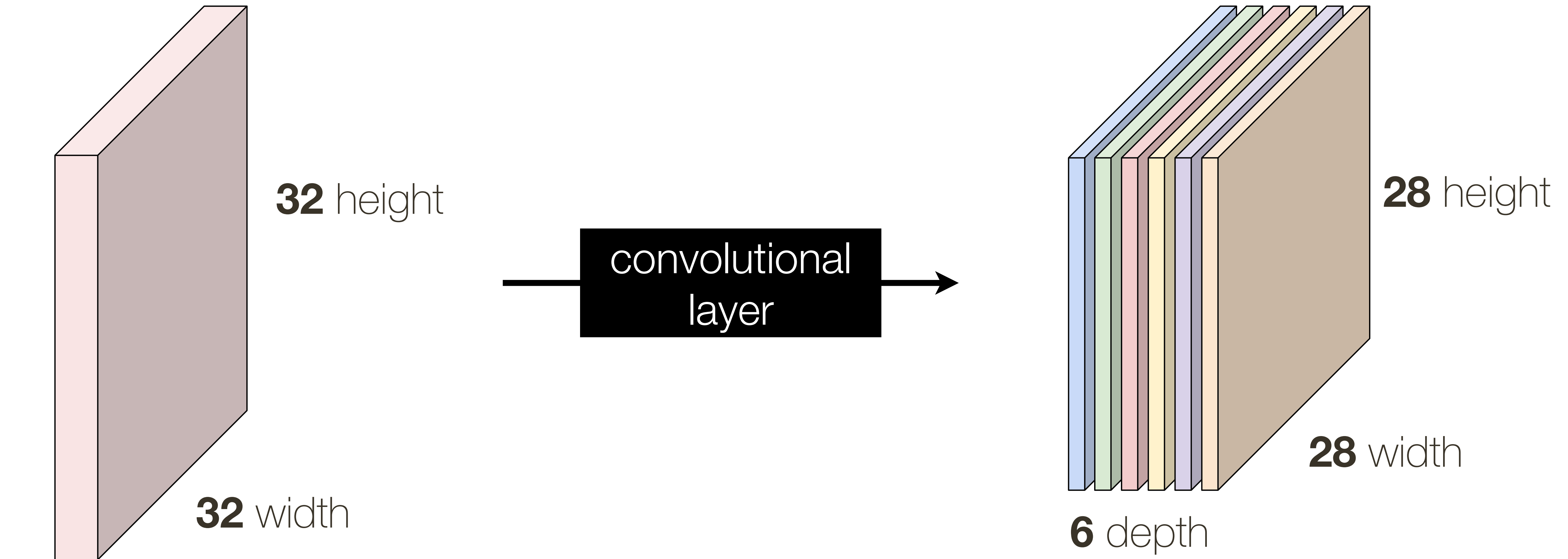
consider another **green** filter

activation map



Convolutional Layer

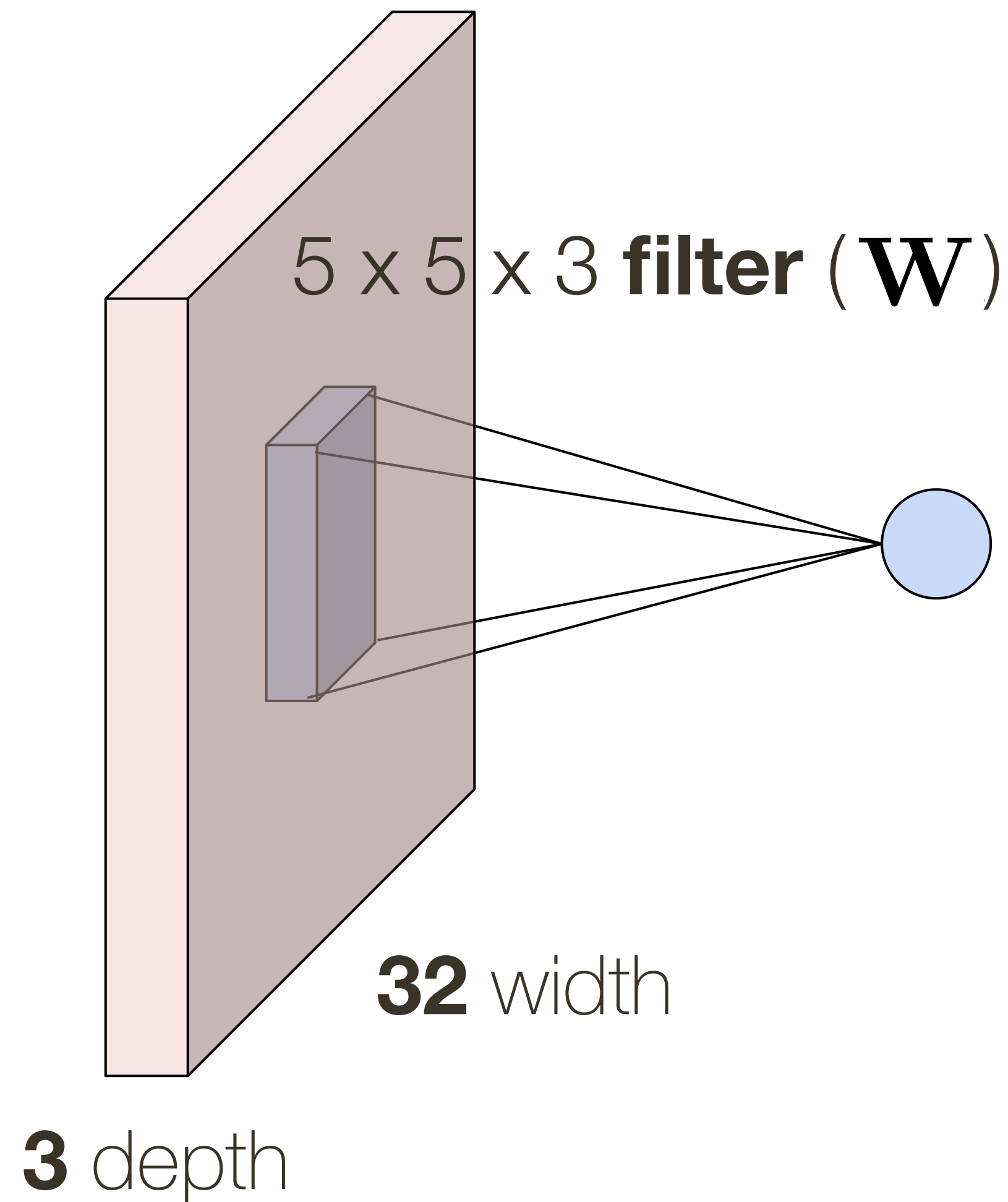
If we have 6 5x5 filter, we'll get 6 separate activation maps: **activation** map



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

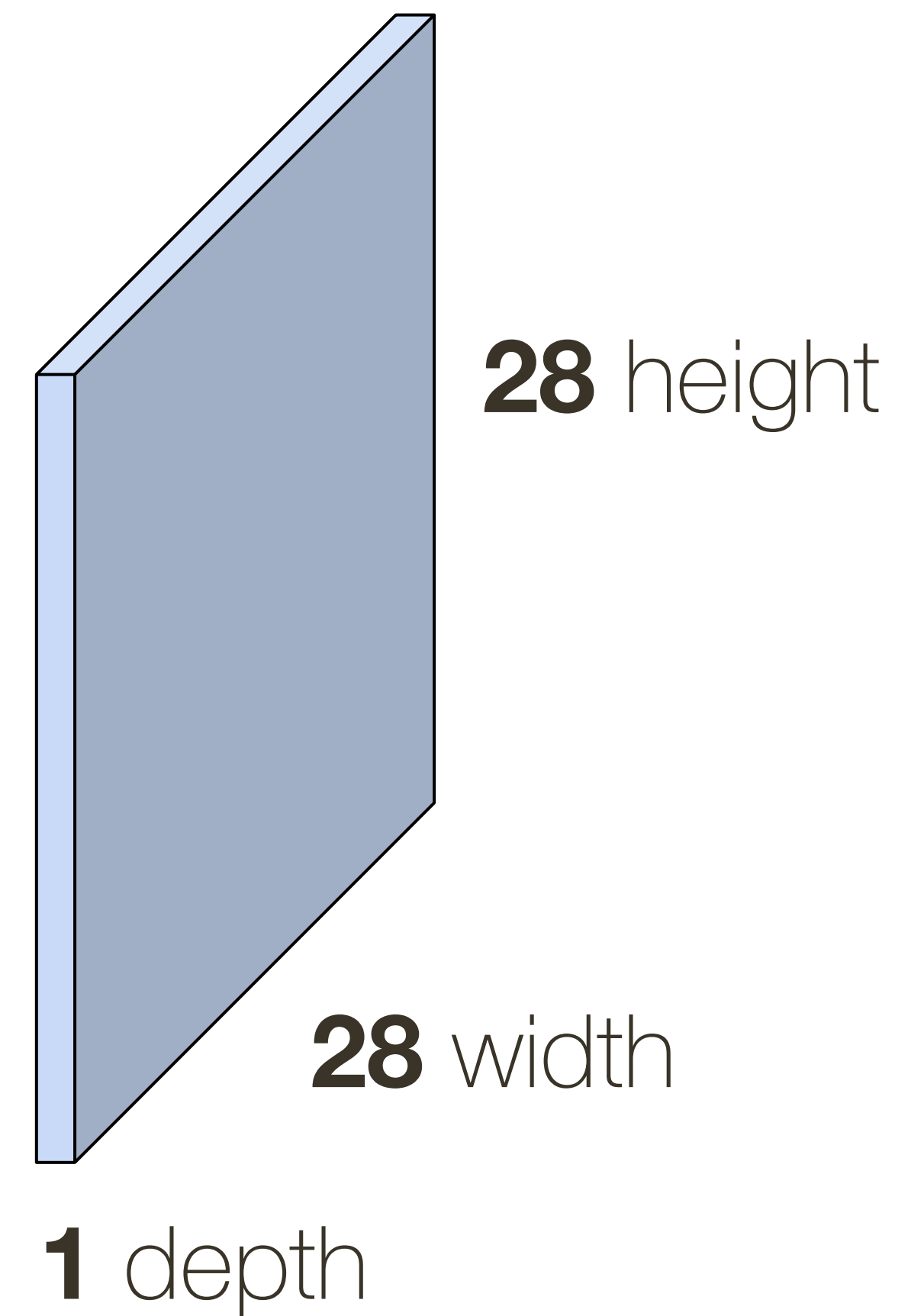
Convolutional Layer: Closer Look at **Spatial Dimensions**

32 x 32 x 3 image

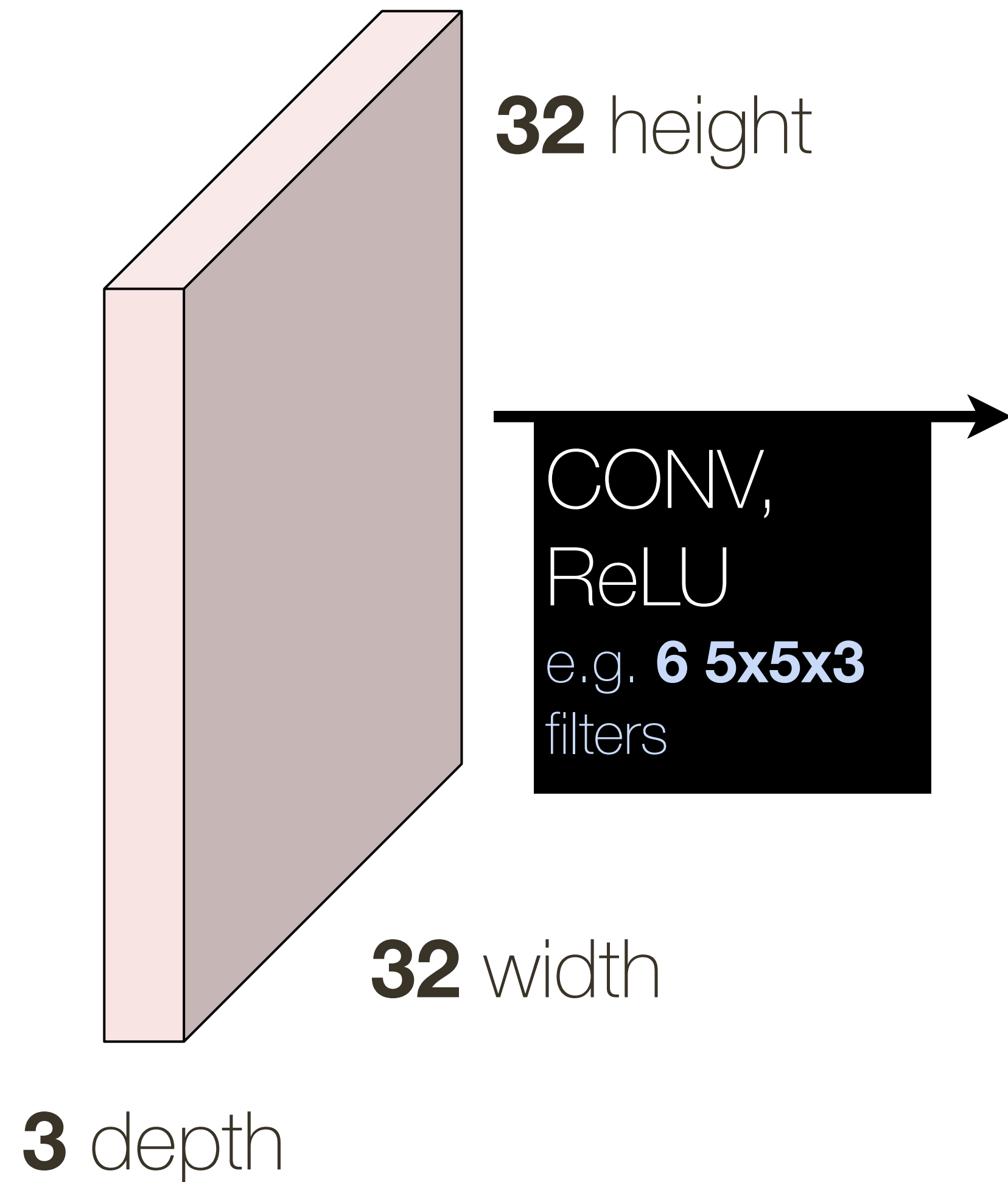


convolve (slide) over all
spatial locations

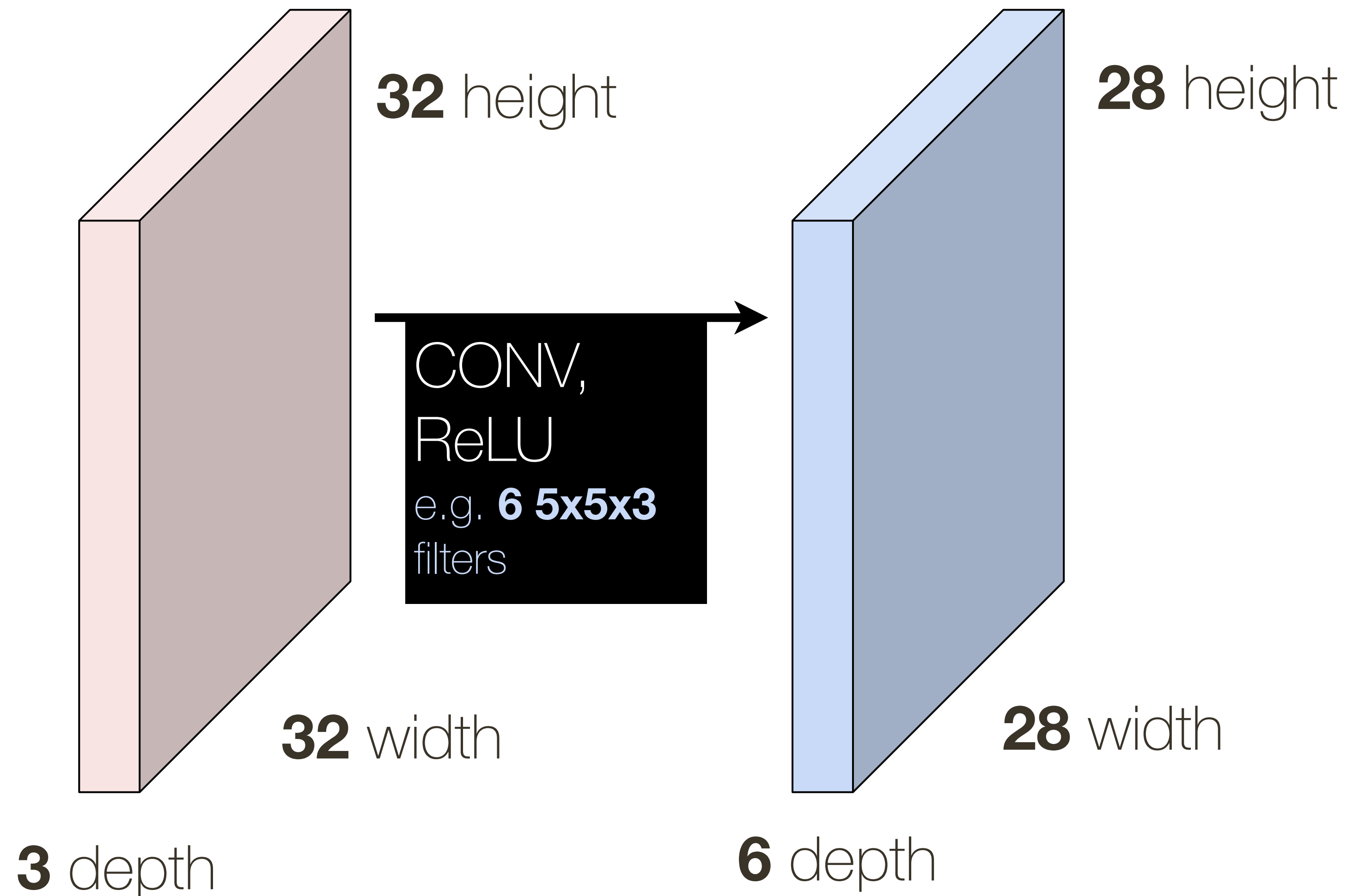
activation map



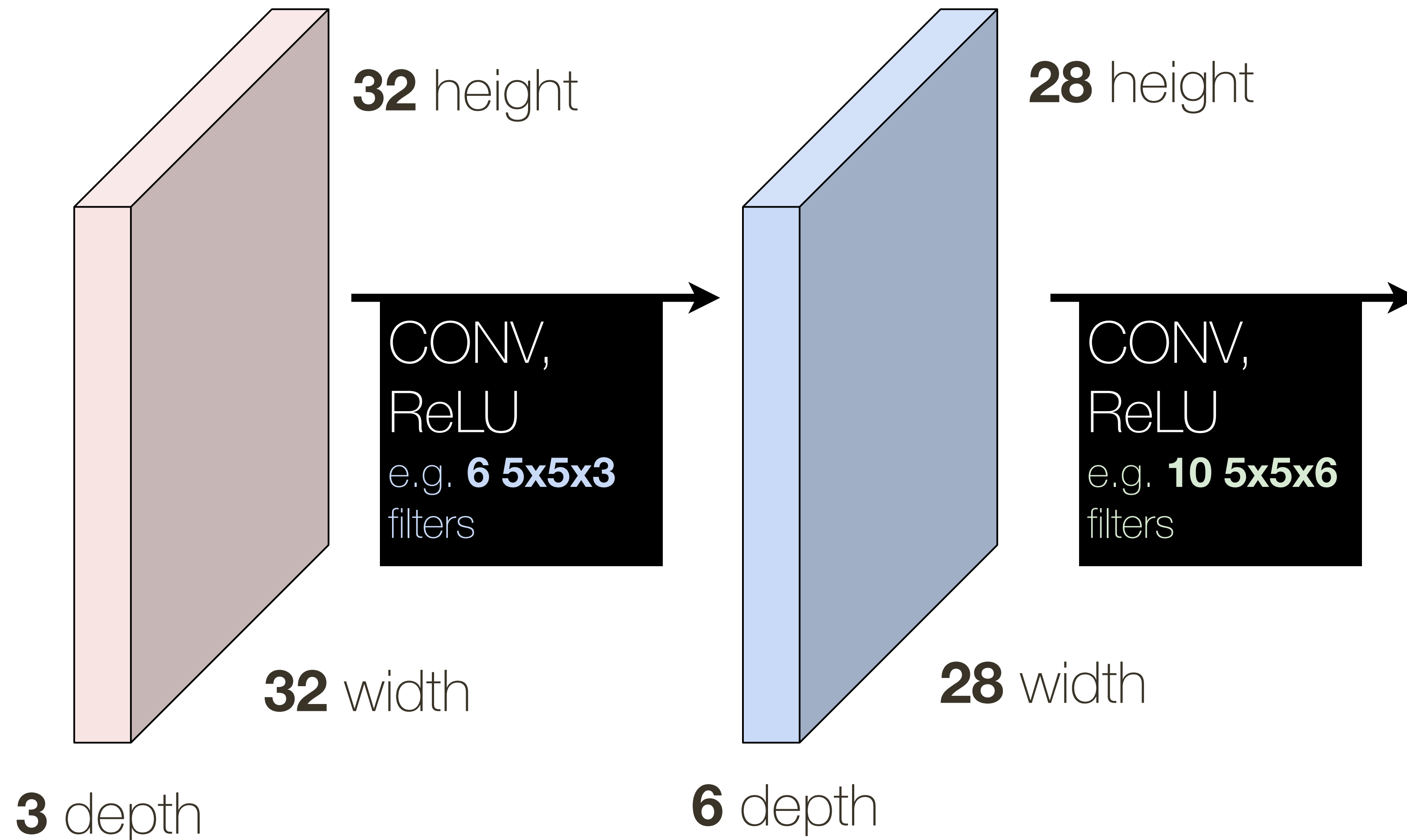
Convolutional Neural Network (ConvNet)



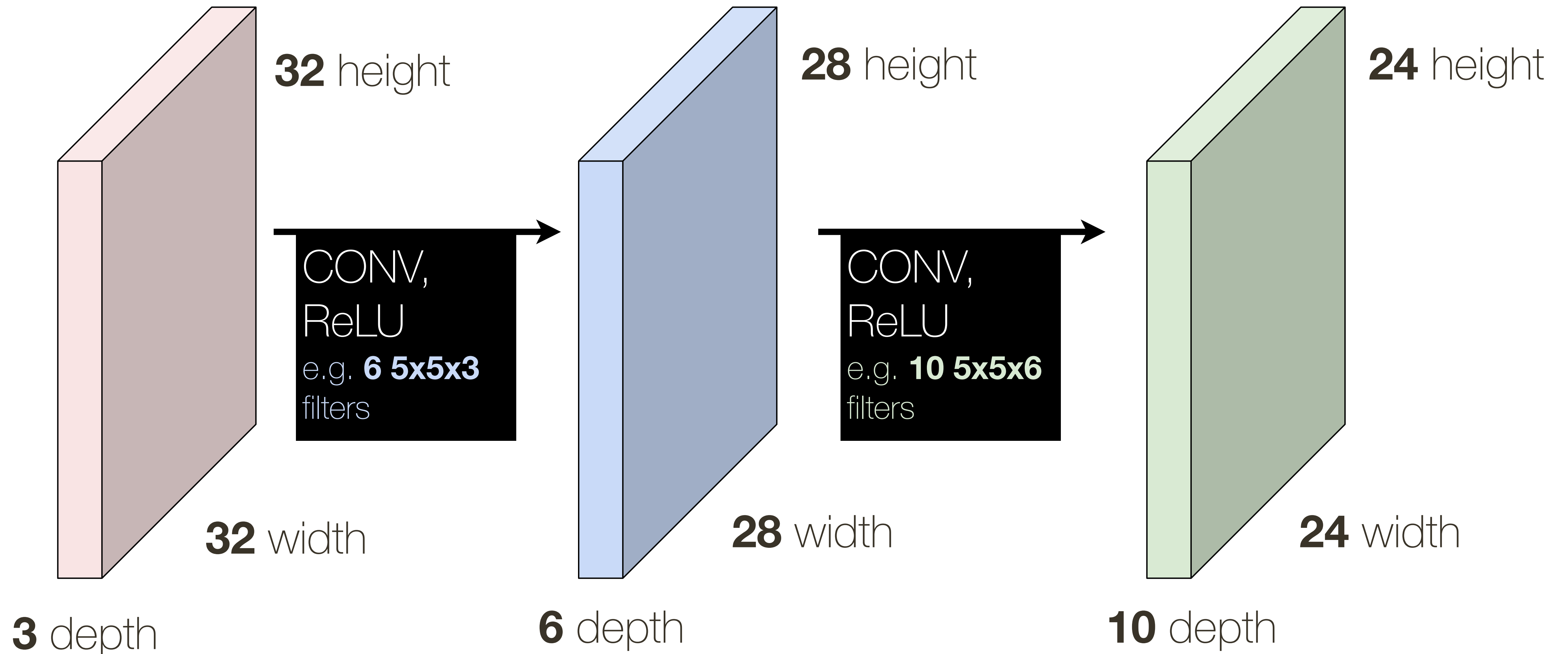
Convolutional Neural Network (ConvNet)



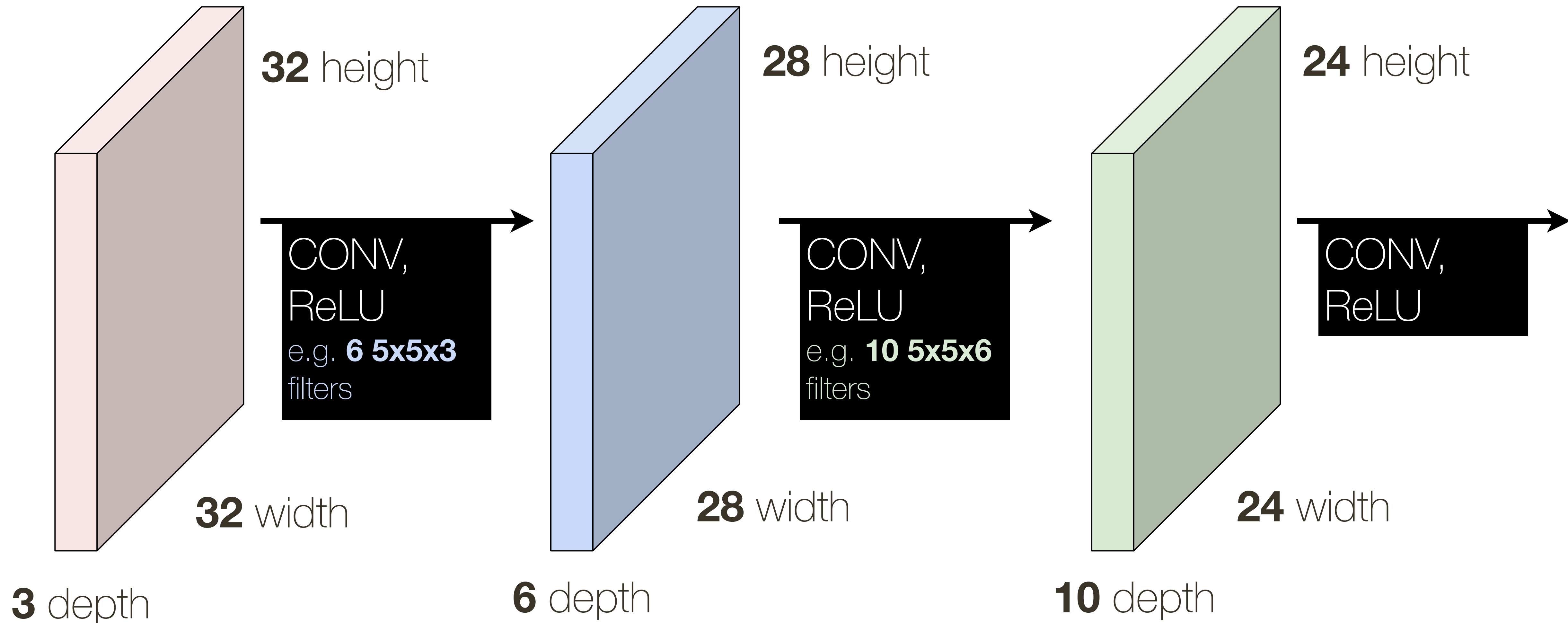
Convolutional Neural Network (ConvNet)



Convolutional Neural Network (ConvNet)

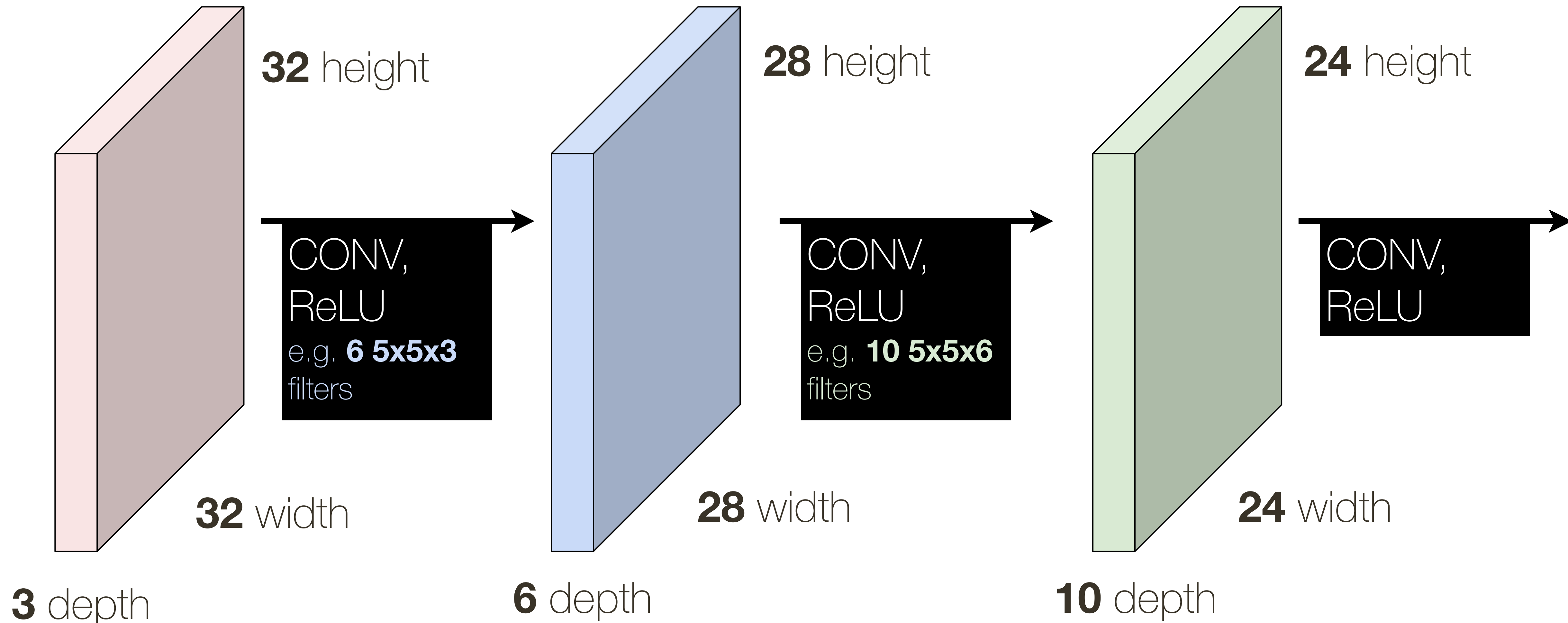


Convolutional Neural Network (ConvNet)



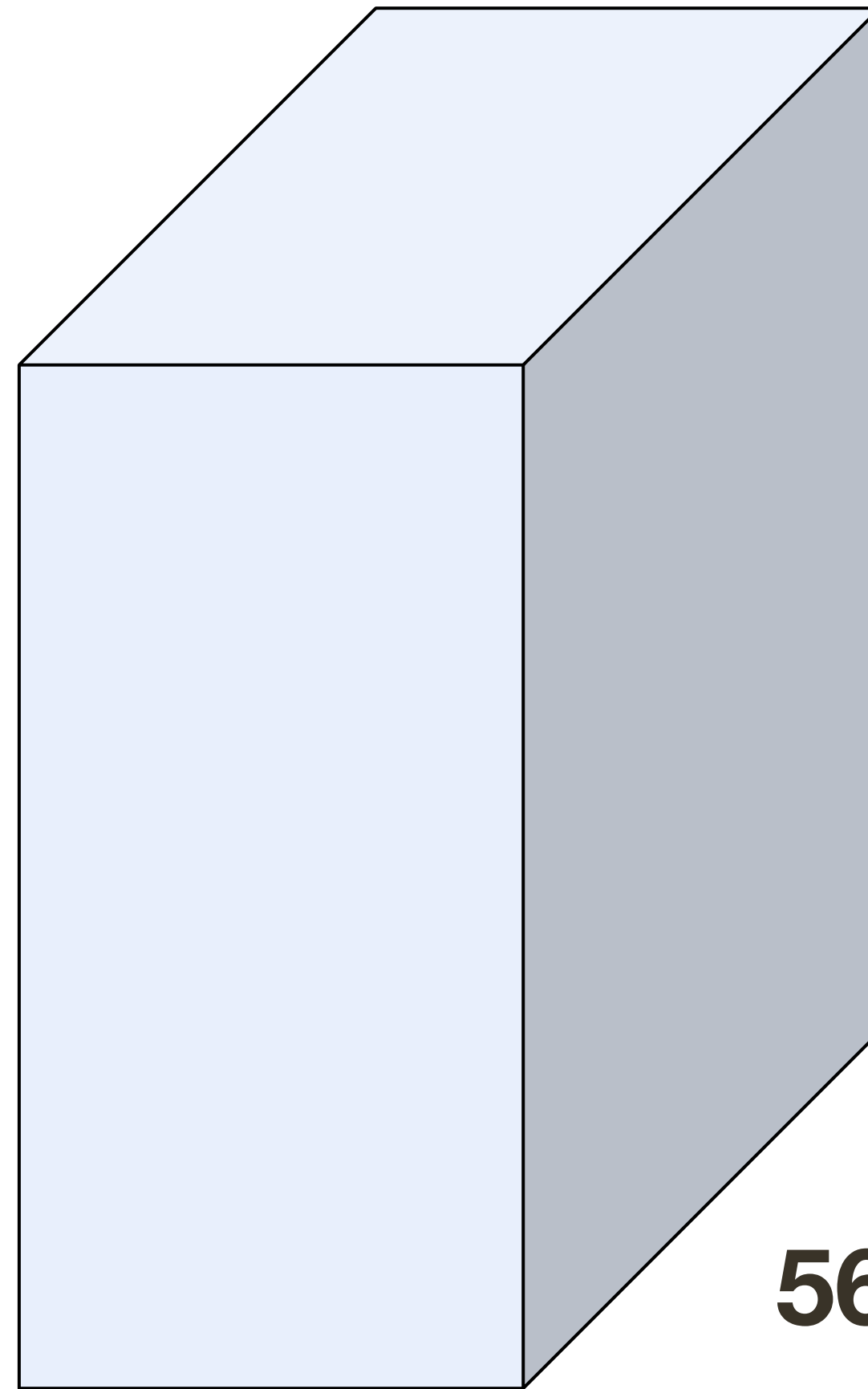
Convolutional Neural Network (ConvNet)

With padding we can achieve no shrinking (32 -> 28 -> 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

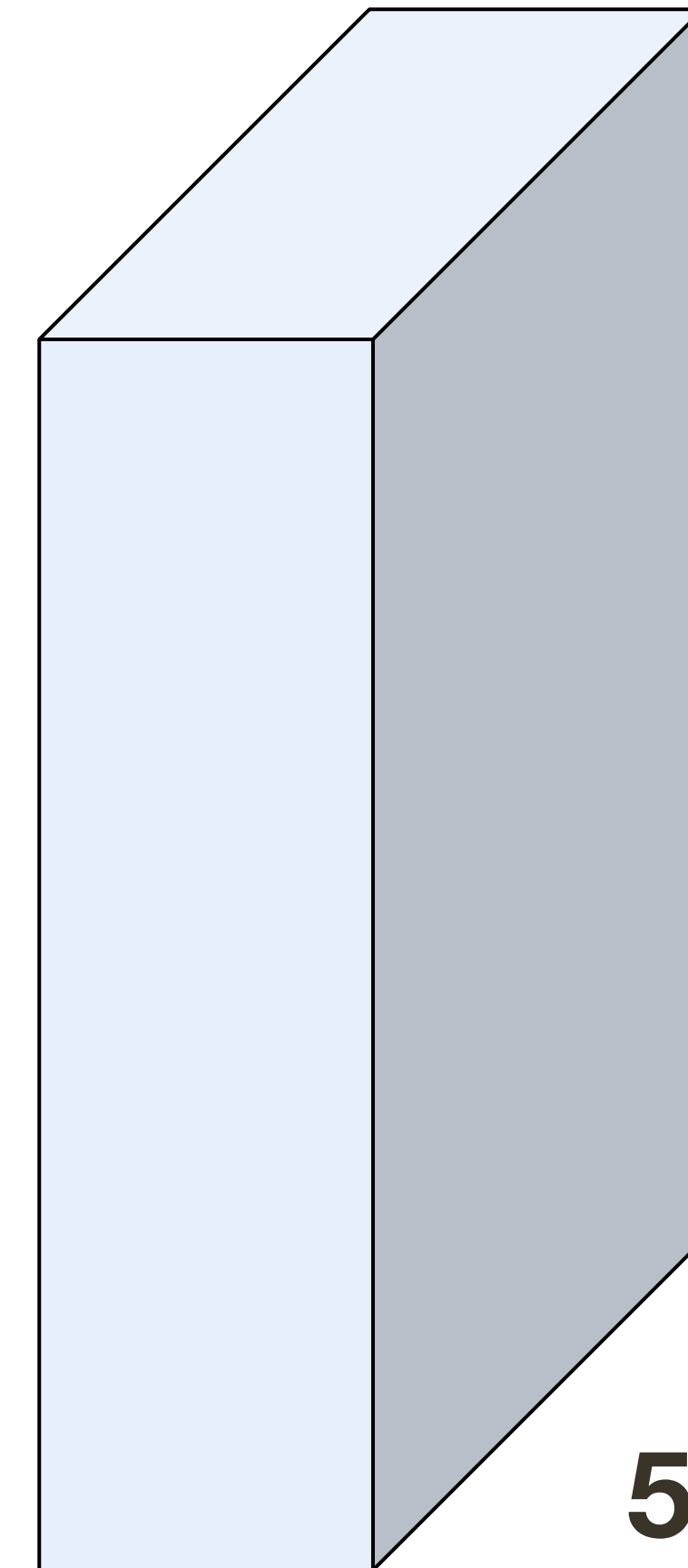
56 width

64 depth

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



56 x 56 x 32 **image**



56 height

56 width

32 depth

Convolutional 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:

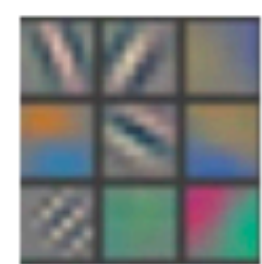
- Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, \dots\}$)
- Stride of application: S (for a typical network $S \in \{1, 2\}$)
- Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

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

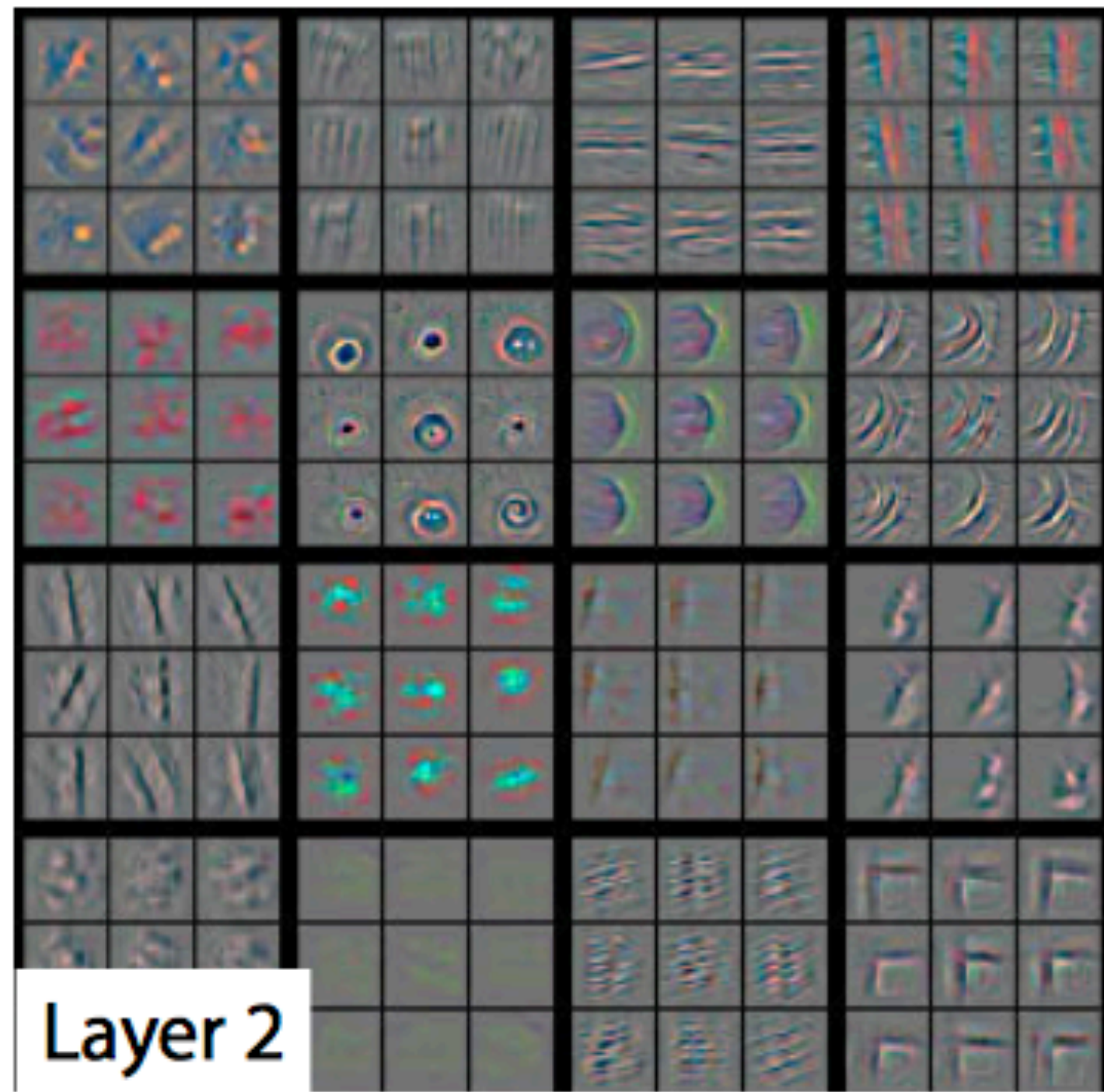
$$W_o = (W_i - F + 2P)/S + 1 \quad H_o = (H_i - F + 2P)/S + 1 \quad D_o = K$$

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

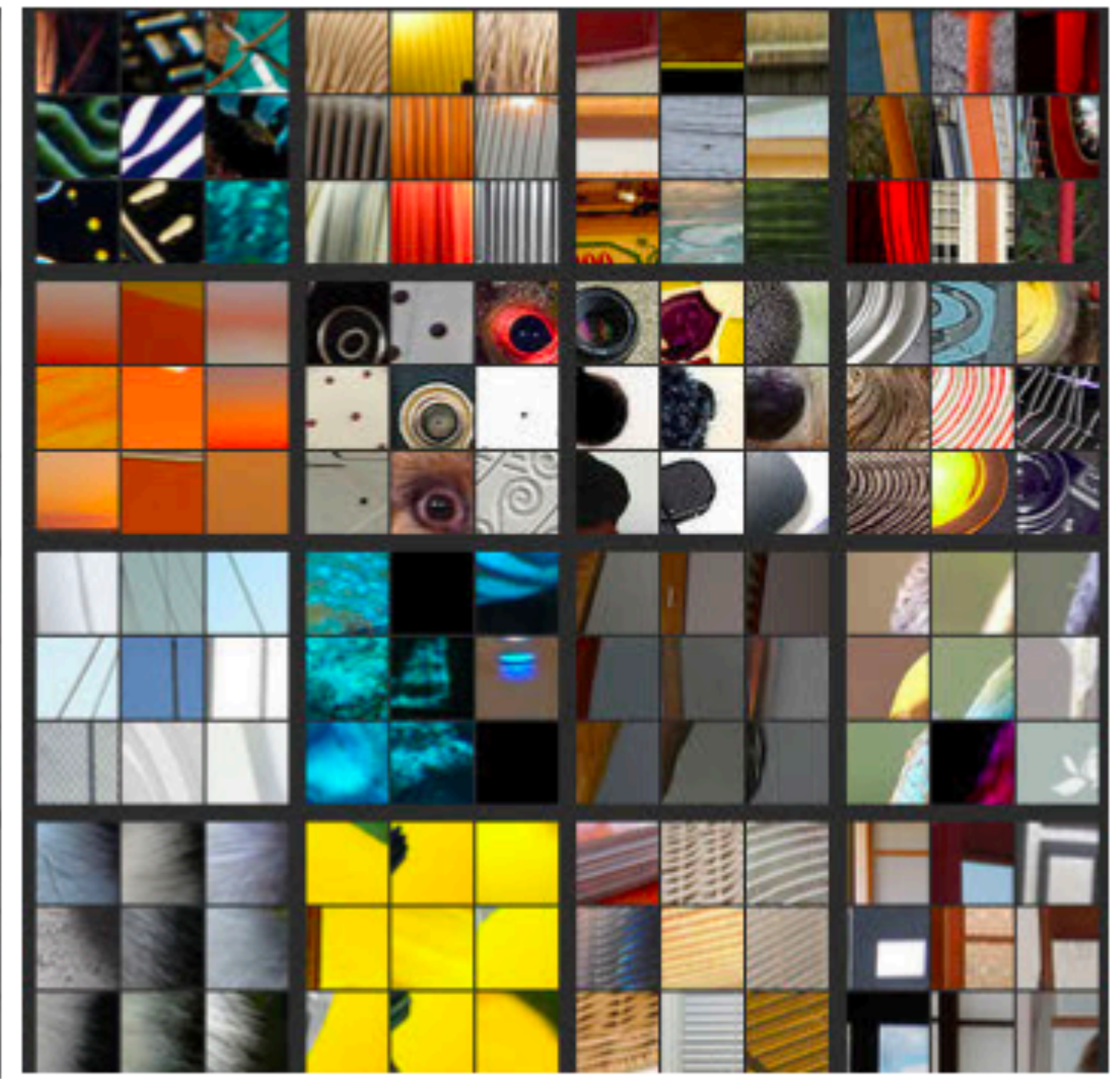
What **filters** do networks learn?



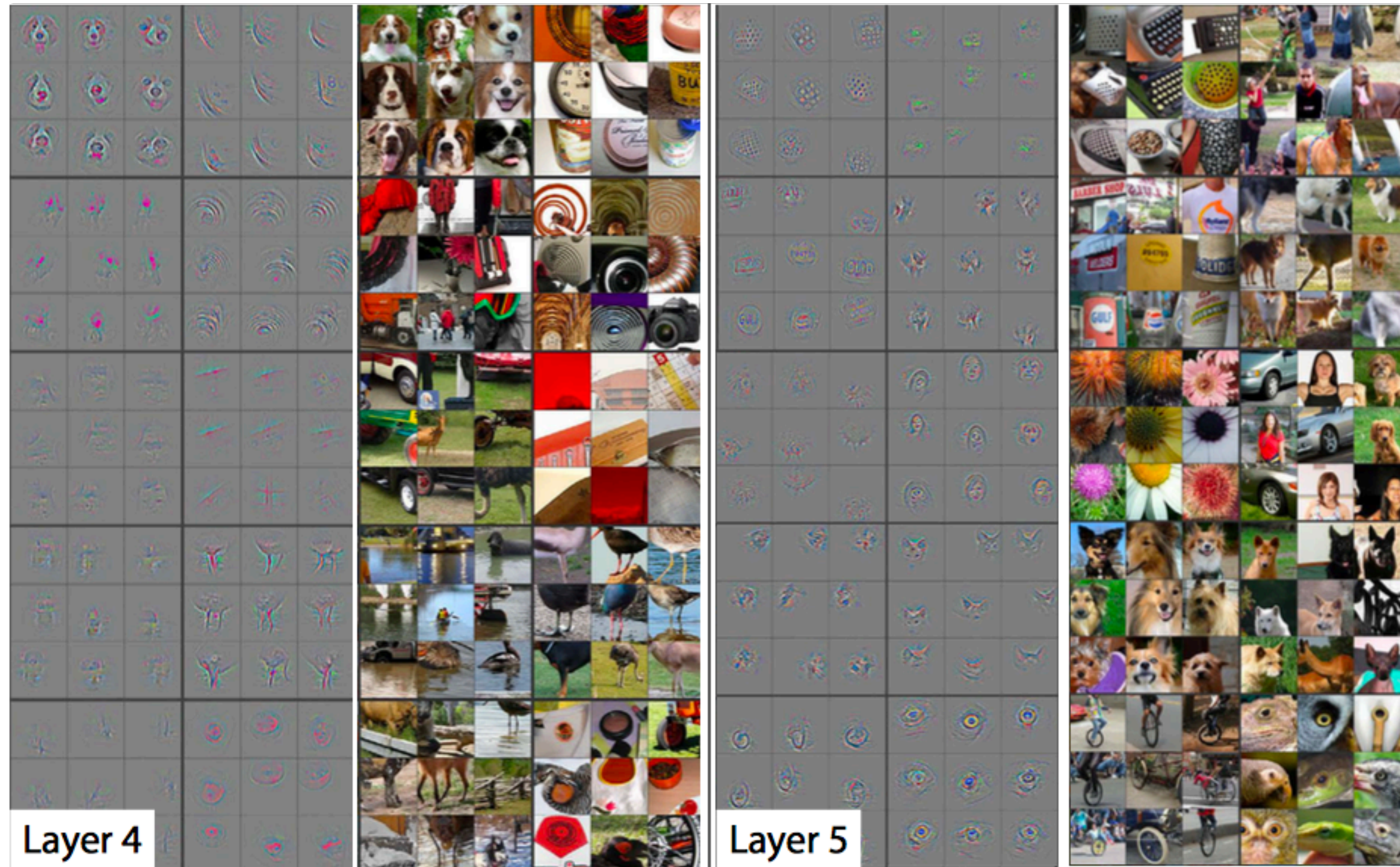
Layer 1



Layer 2



What **filters** do networks learn?

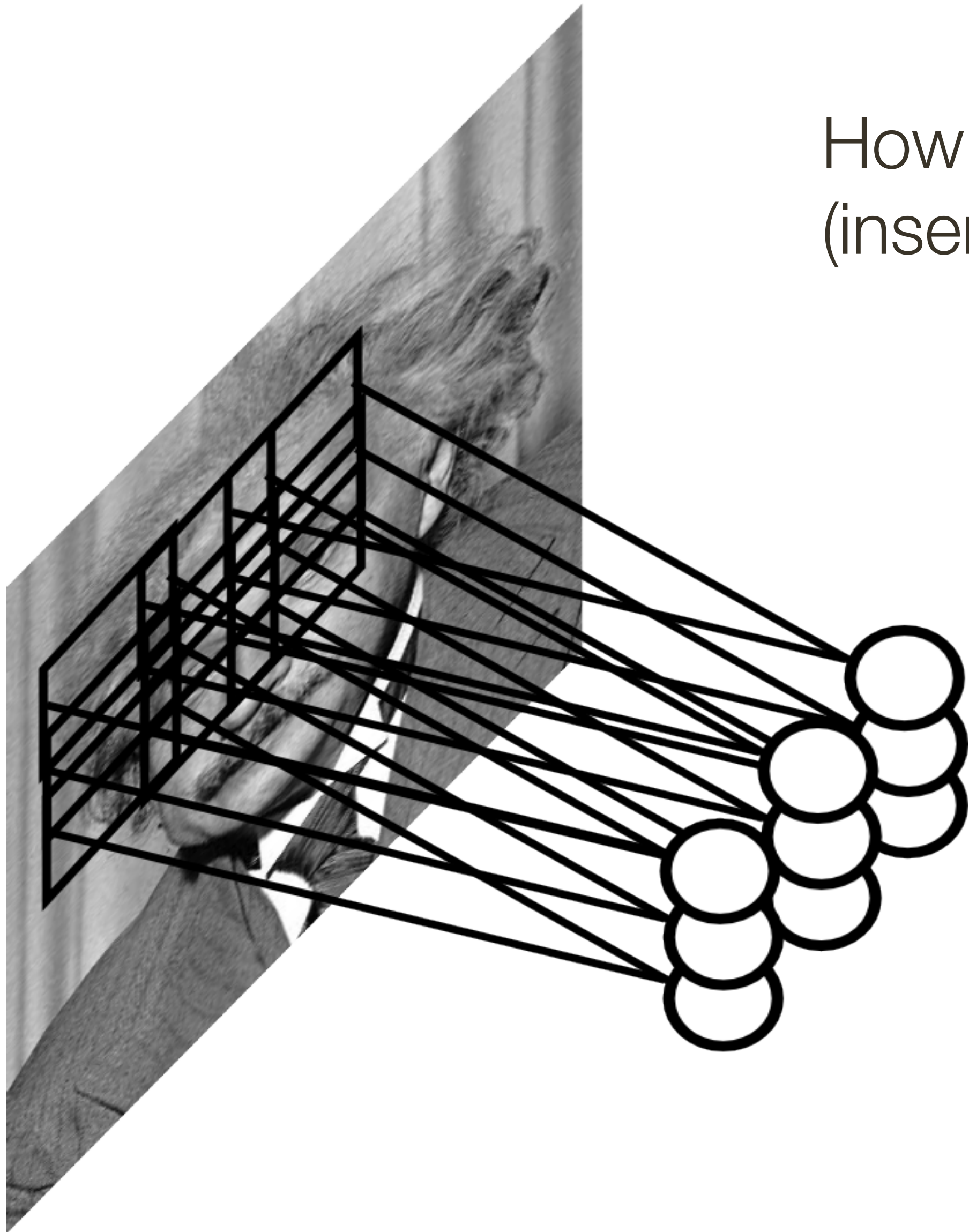


[Zeiler and Fergus, 2013]

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)

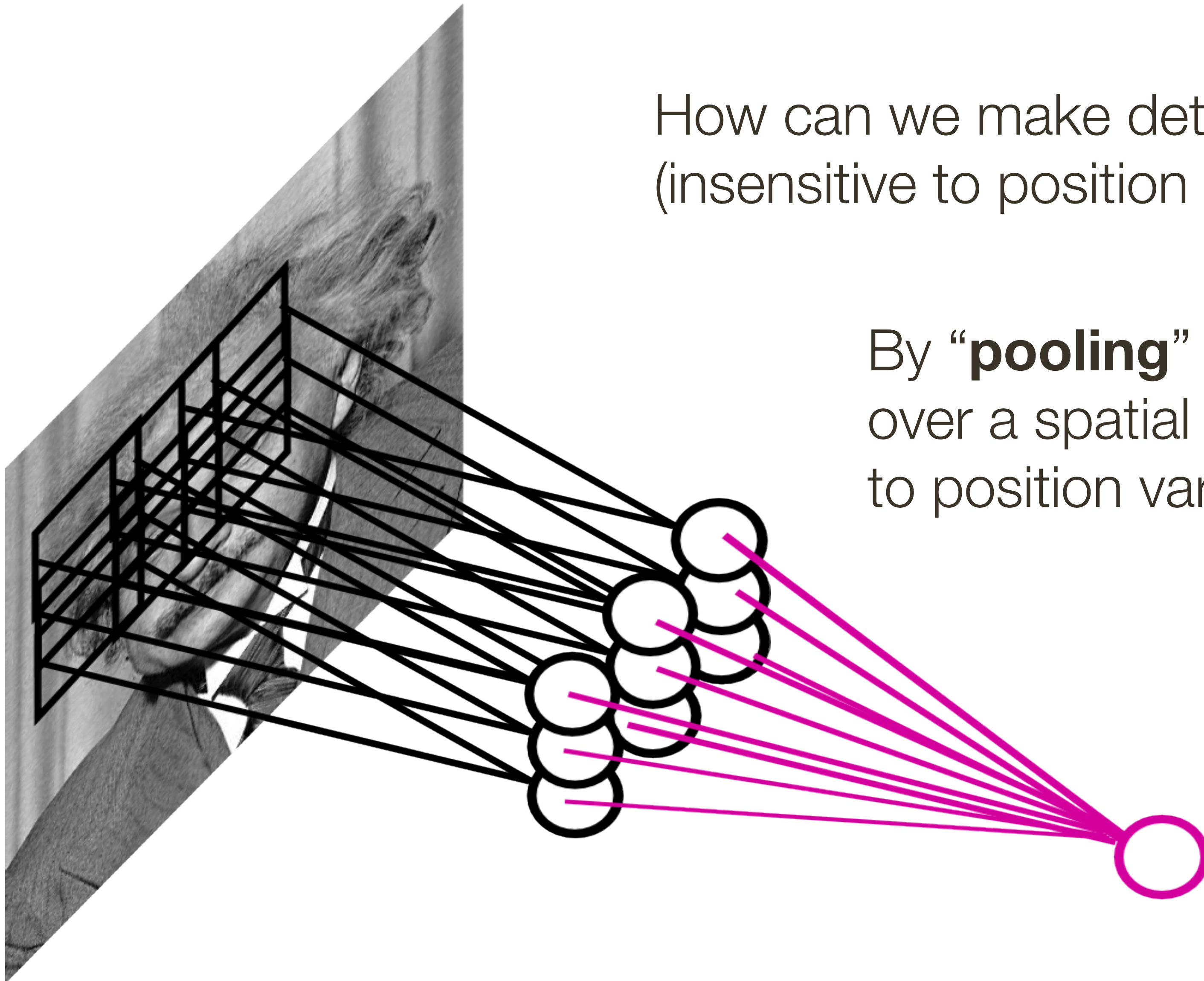


Pooling Layer

Let us assume the filter is an “eye” detector

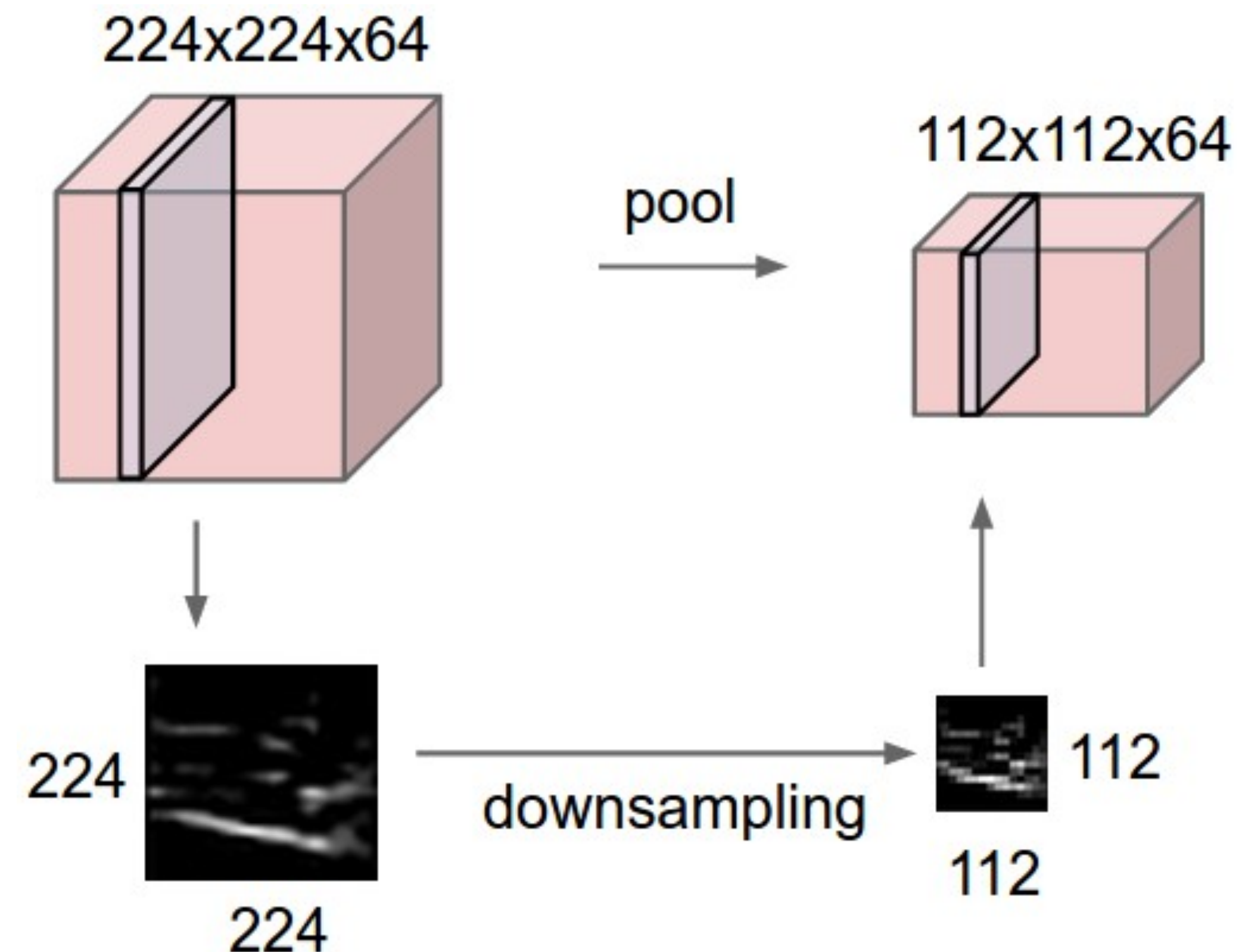
How can we make detection spatially invariant
(insensitive to position of the eye in the image)

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



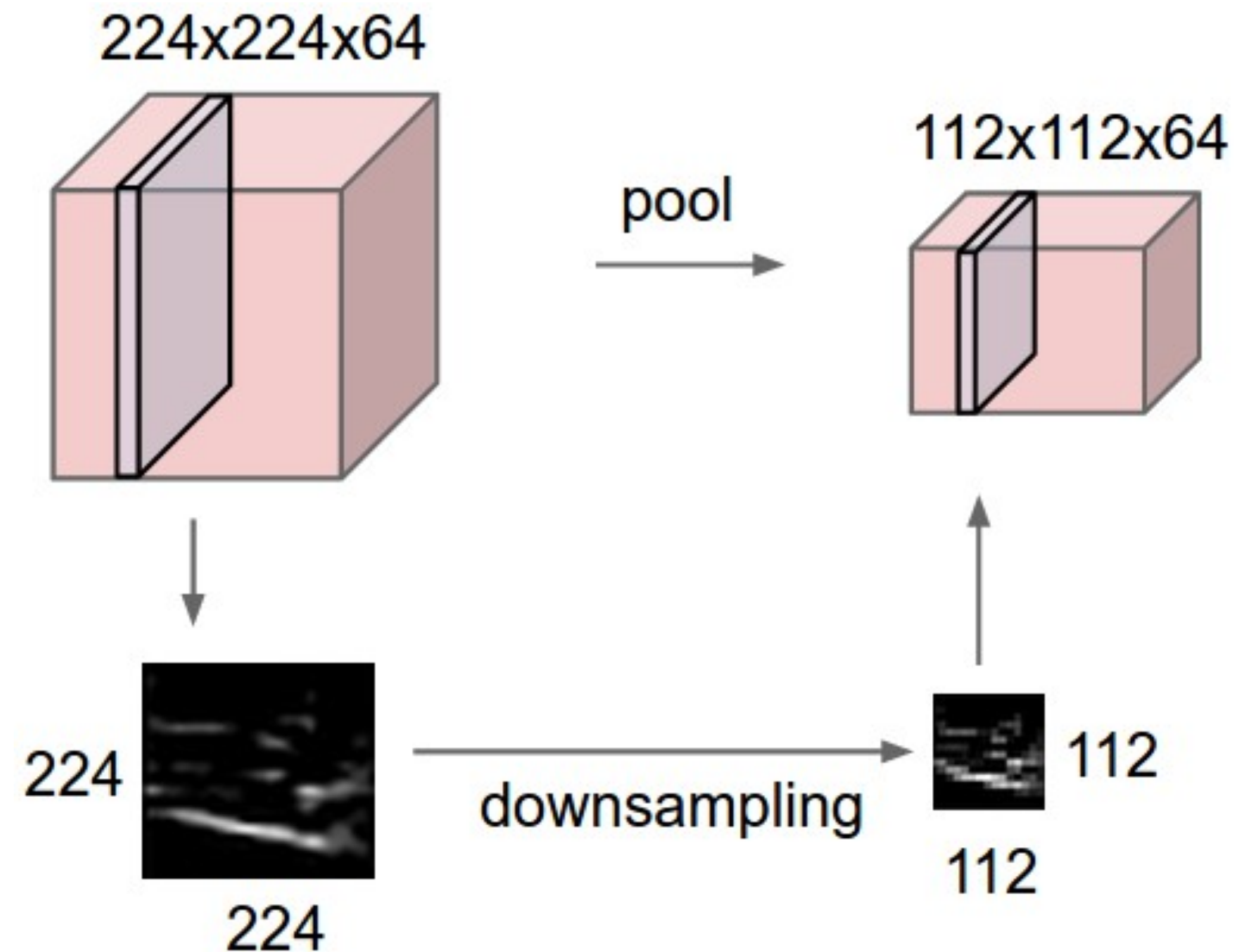
Pooling Layer

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



Pooling Layer

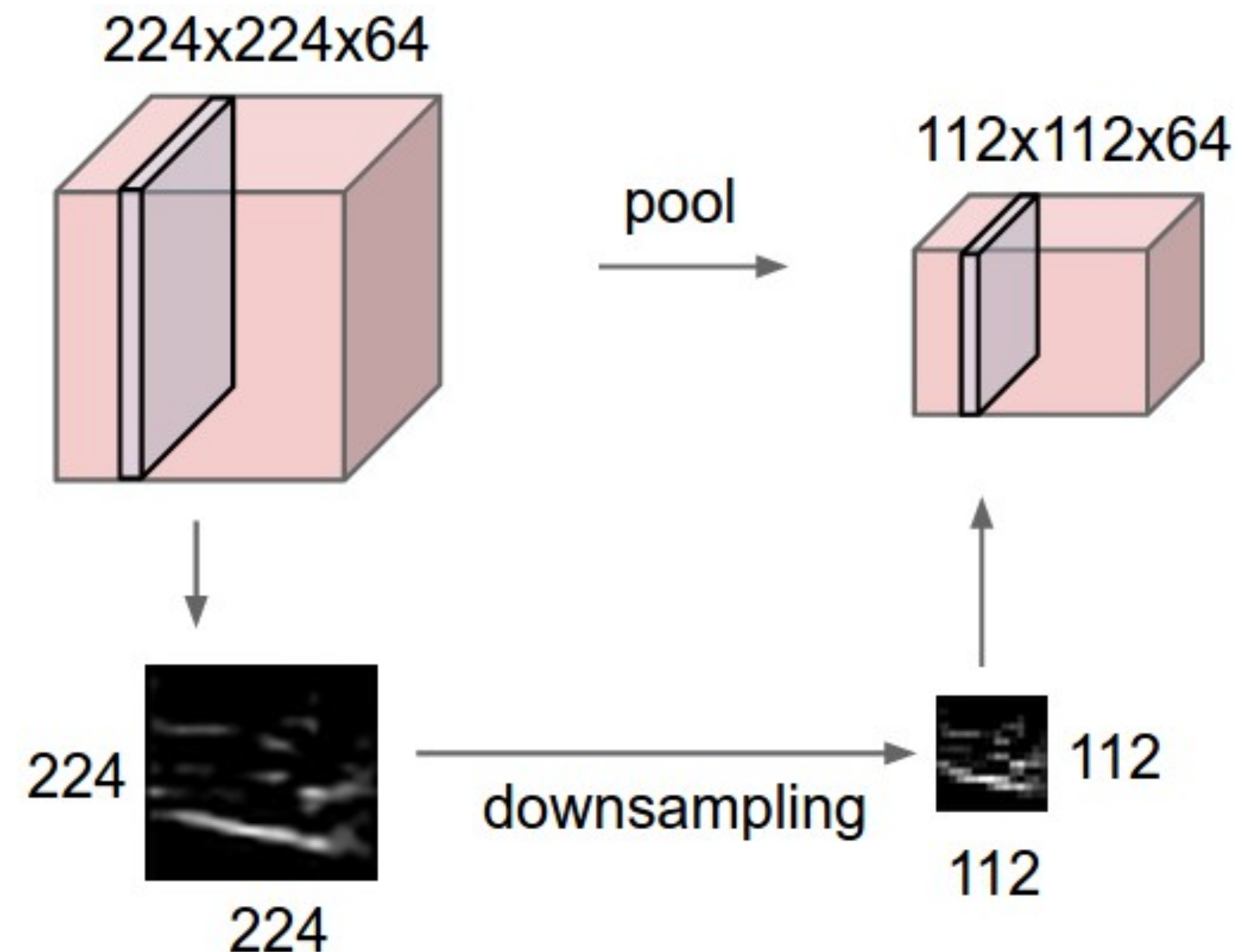
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



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: $W_o \times H_o \times D_o$

$$W_o = W_i / F$$

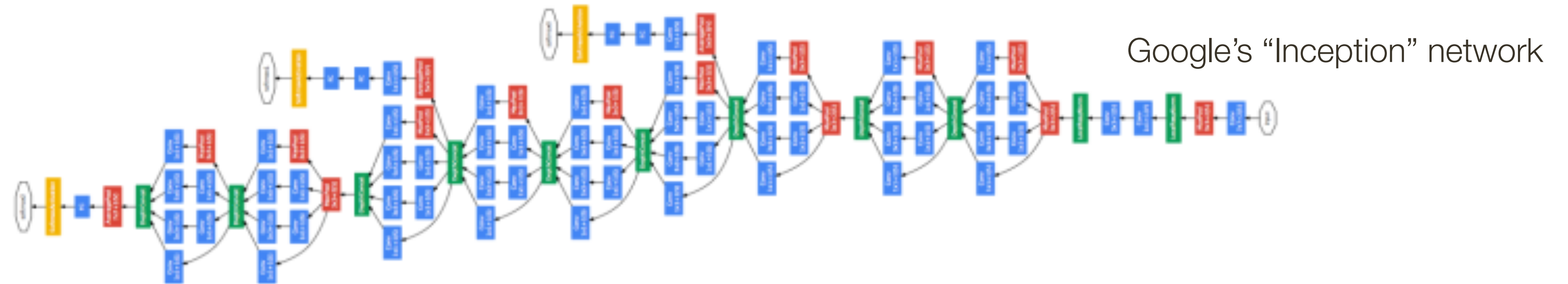
$$H_o = H_i / F$$

$$D_o = D_i$$

Number of total learnable parameters: 0

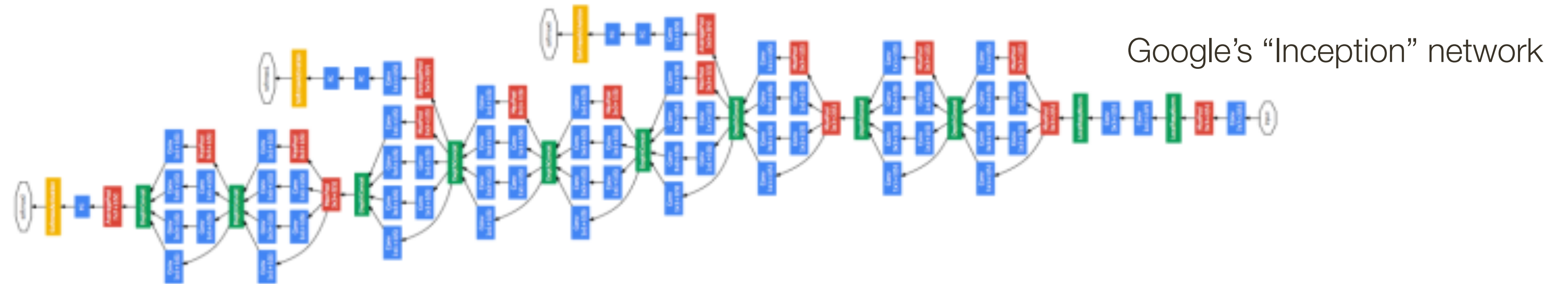
(you can do padding, but it's a bit trickier)

Deep Learning **Terminology**



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

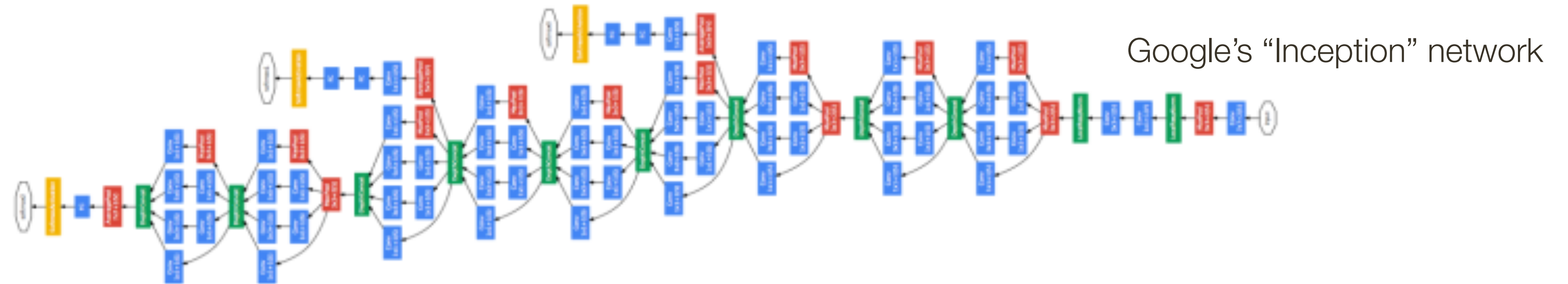
Deep Learning Terminology



- **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

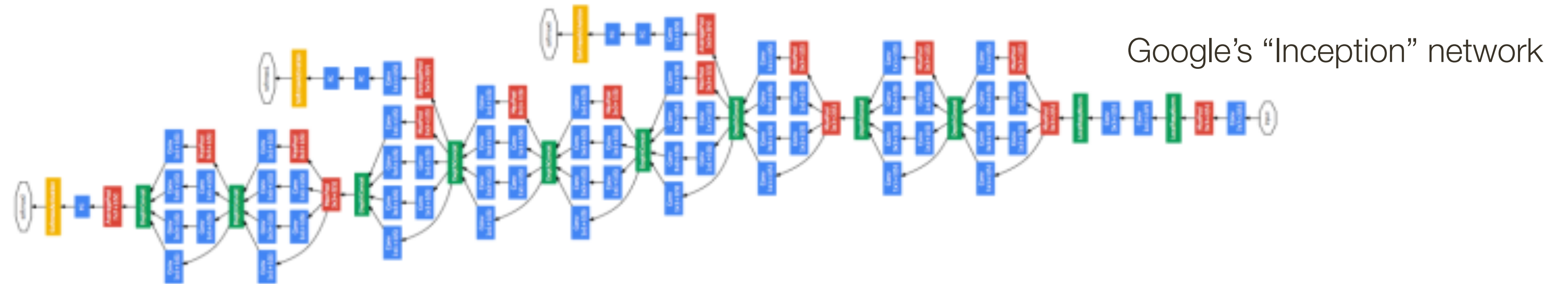


- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

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

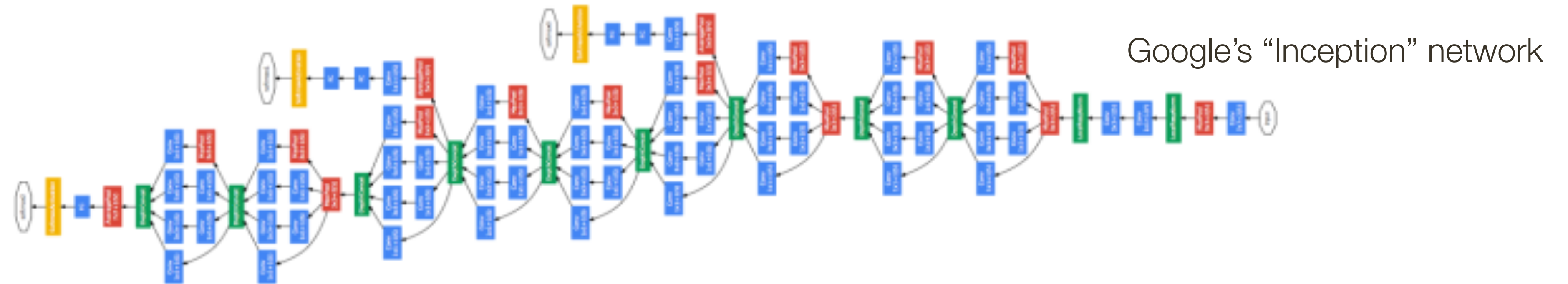
deeper = better

Deep Learning Terminology



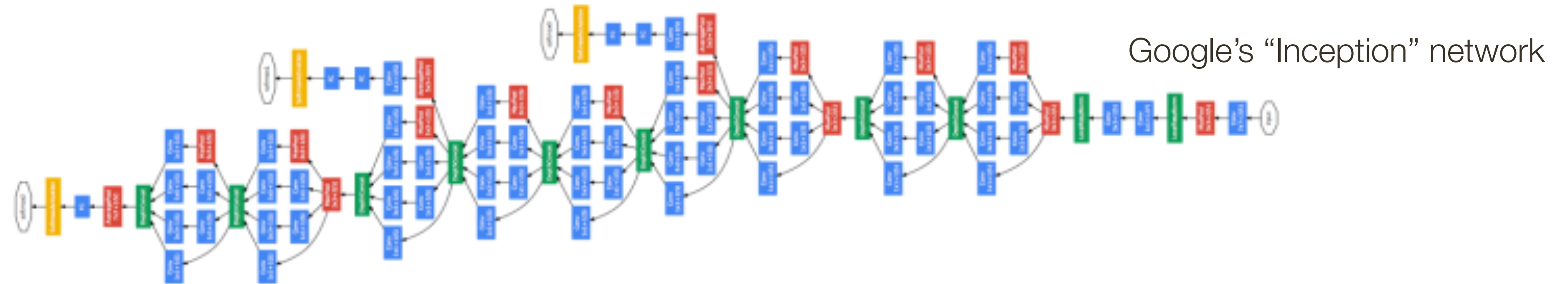
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
 - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
 - deeper = better
- **Loss function:** objective function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

Deep Learning Terminology



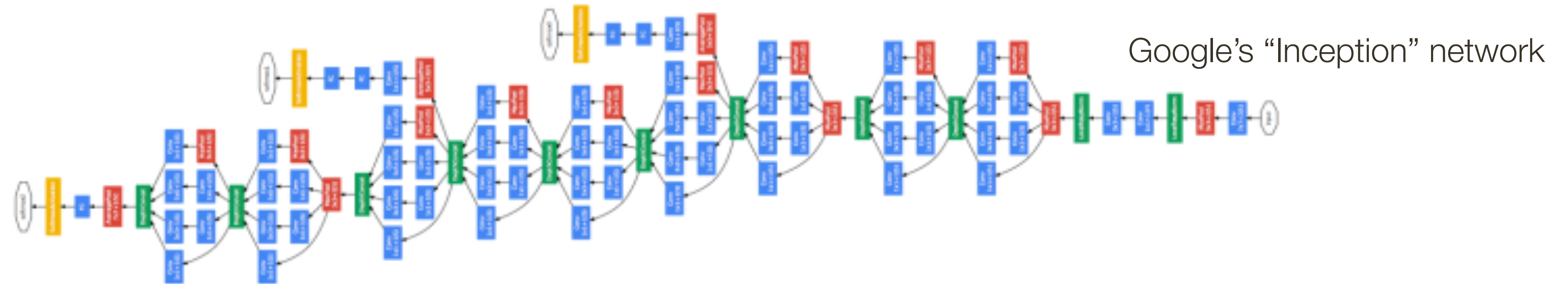
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
 - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
 - deeper = better
- **Loss function:** objective function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

Deep Learning Terminology



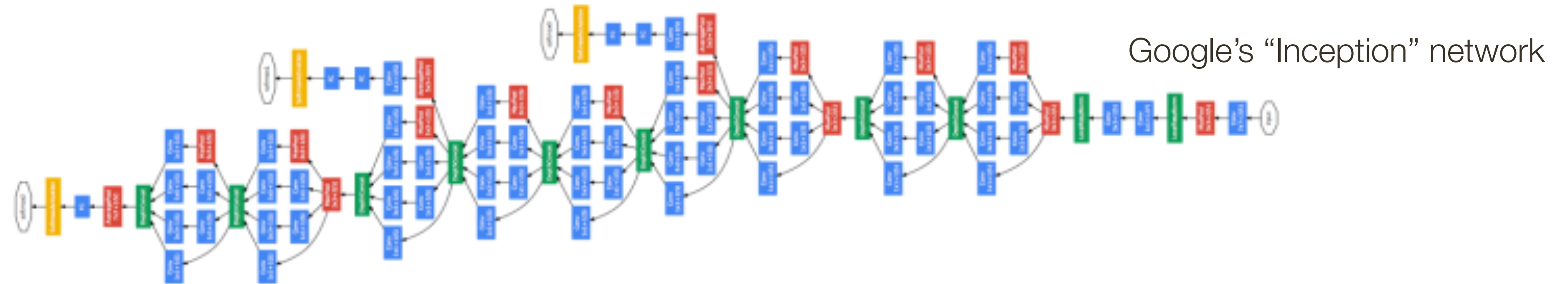
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
 - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
 - deeper = better
- **Loss function:** objective function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



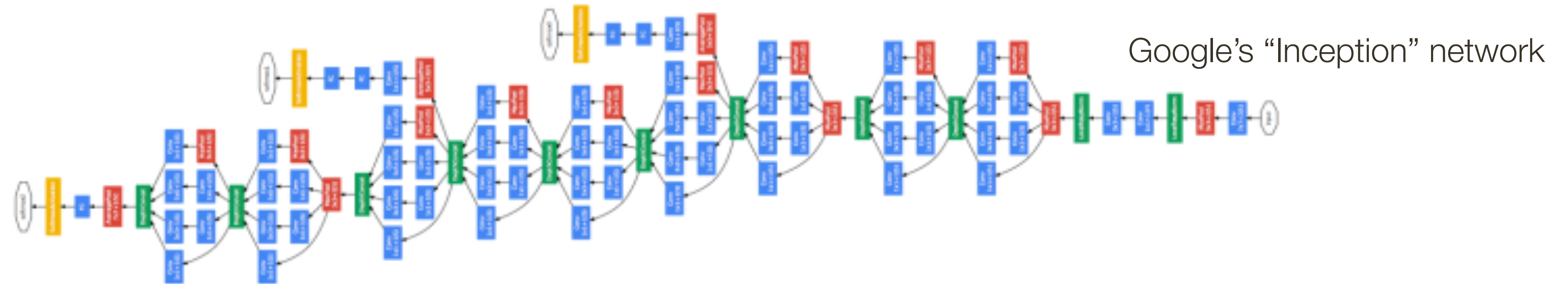
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
 - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
 - deeper = better
- **Loss function:** objective function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

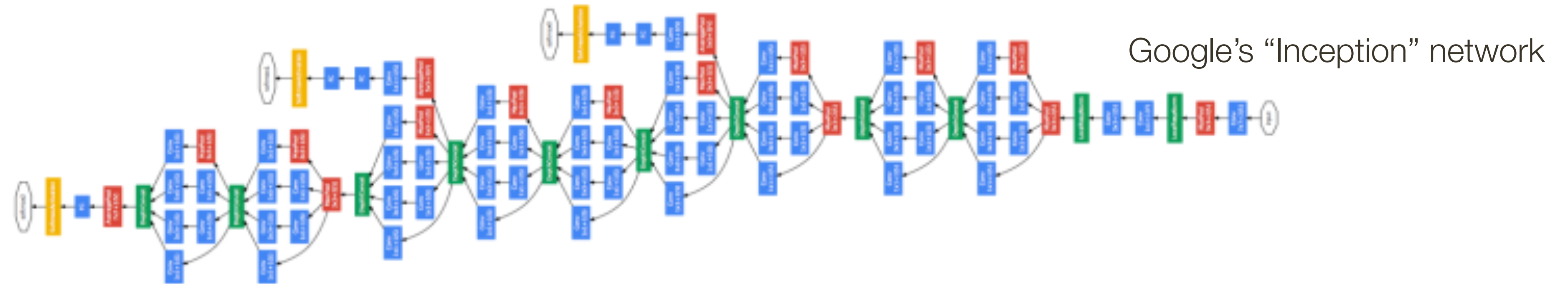
- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

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

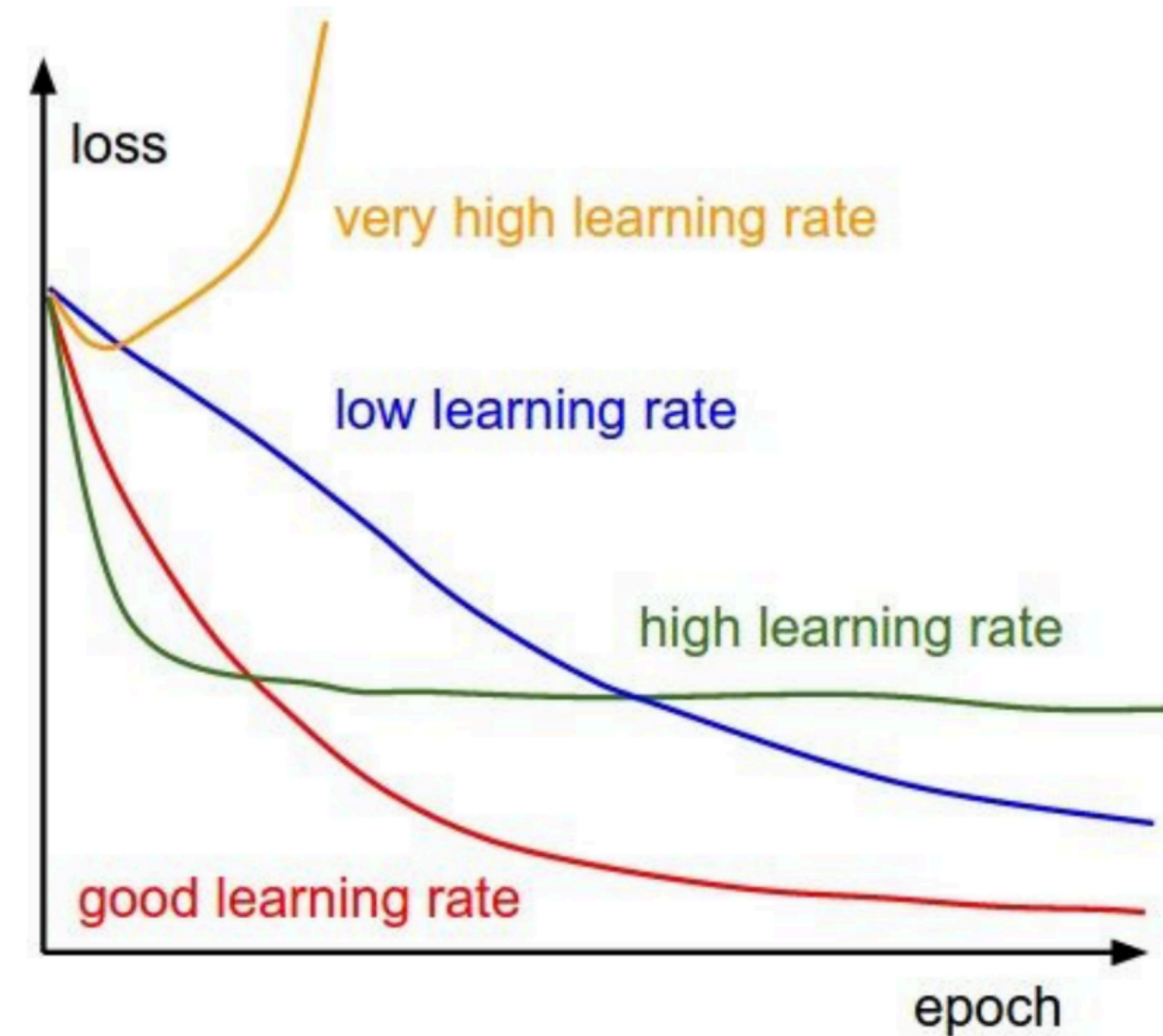
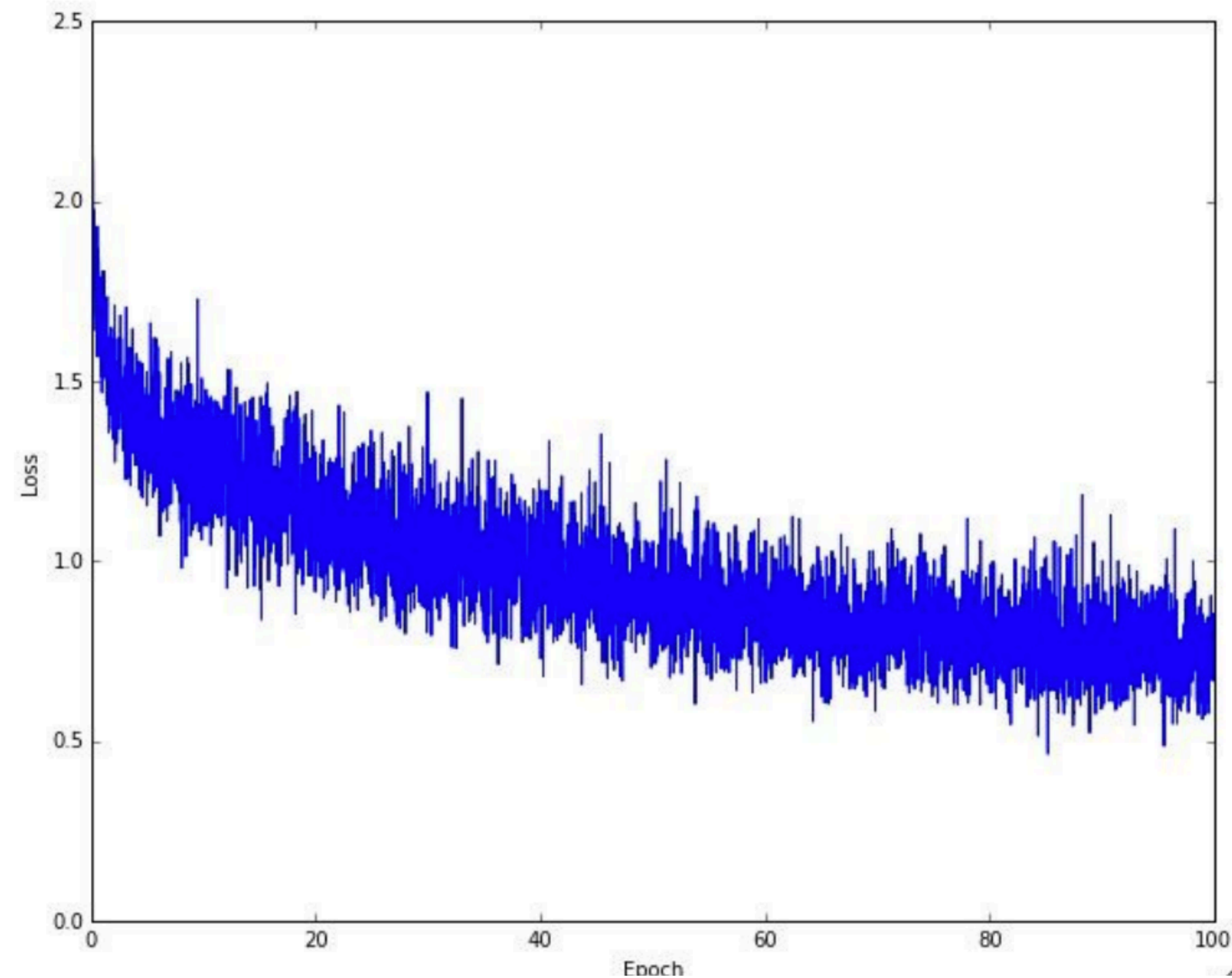
For a fixed number of iterations or until convergence

- 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)

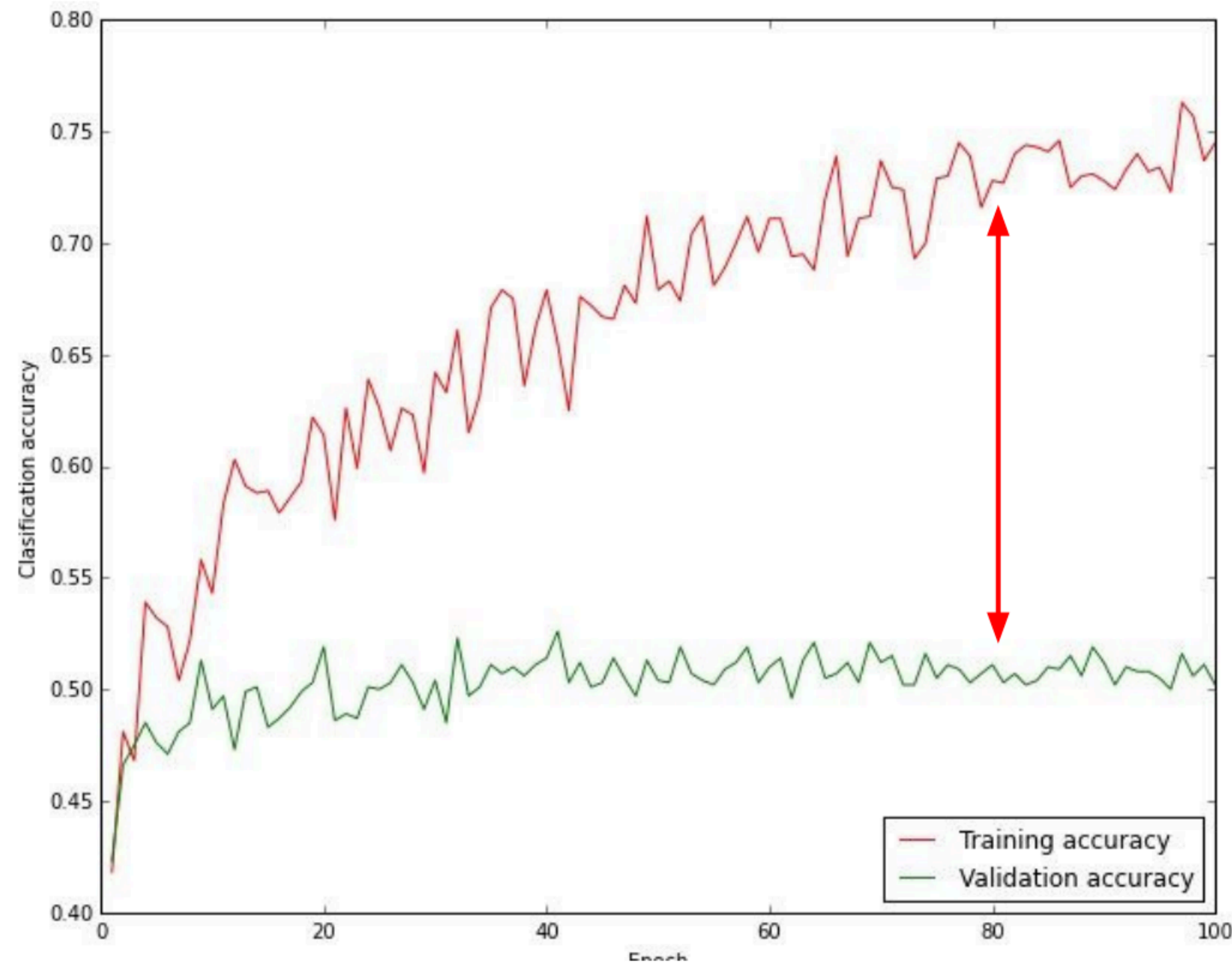
Inference / Prediction

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

Monitoring Learning: Visualizing the (training) loss



Monitoring Learning: Visualizing the (training) loss



Big gap = overfitting

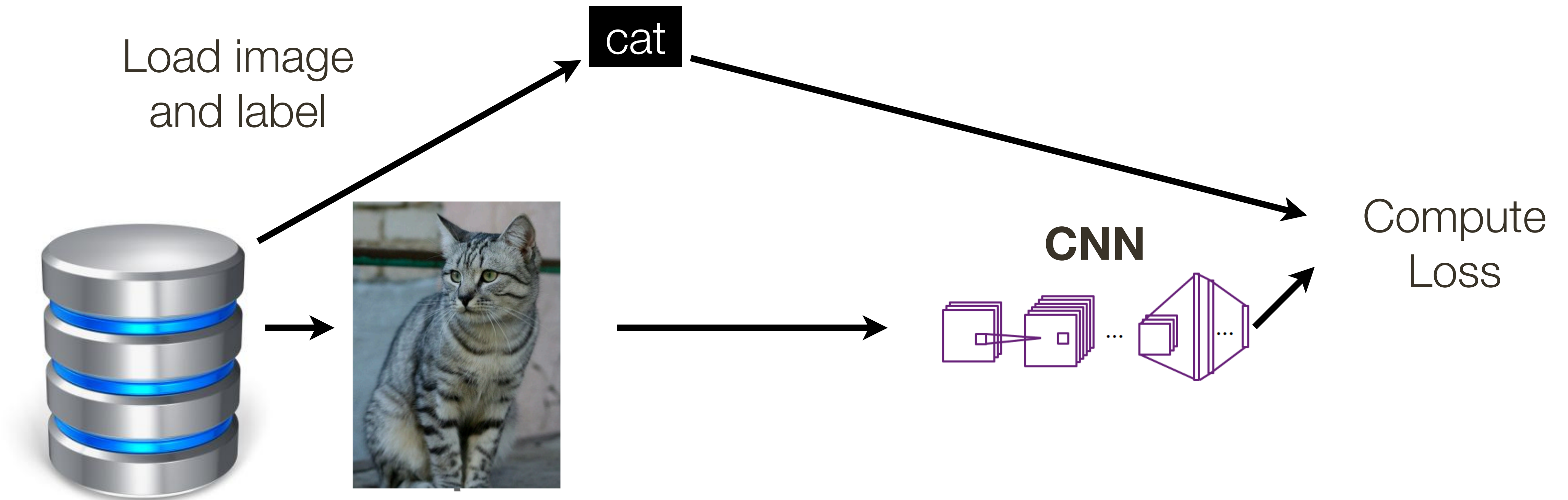
Solution: increase regularization

No gap = undercutting

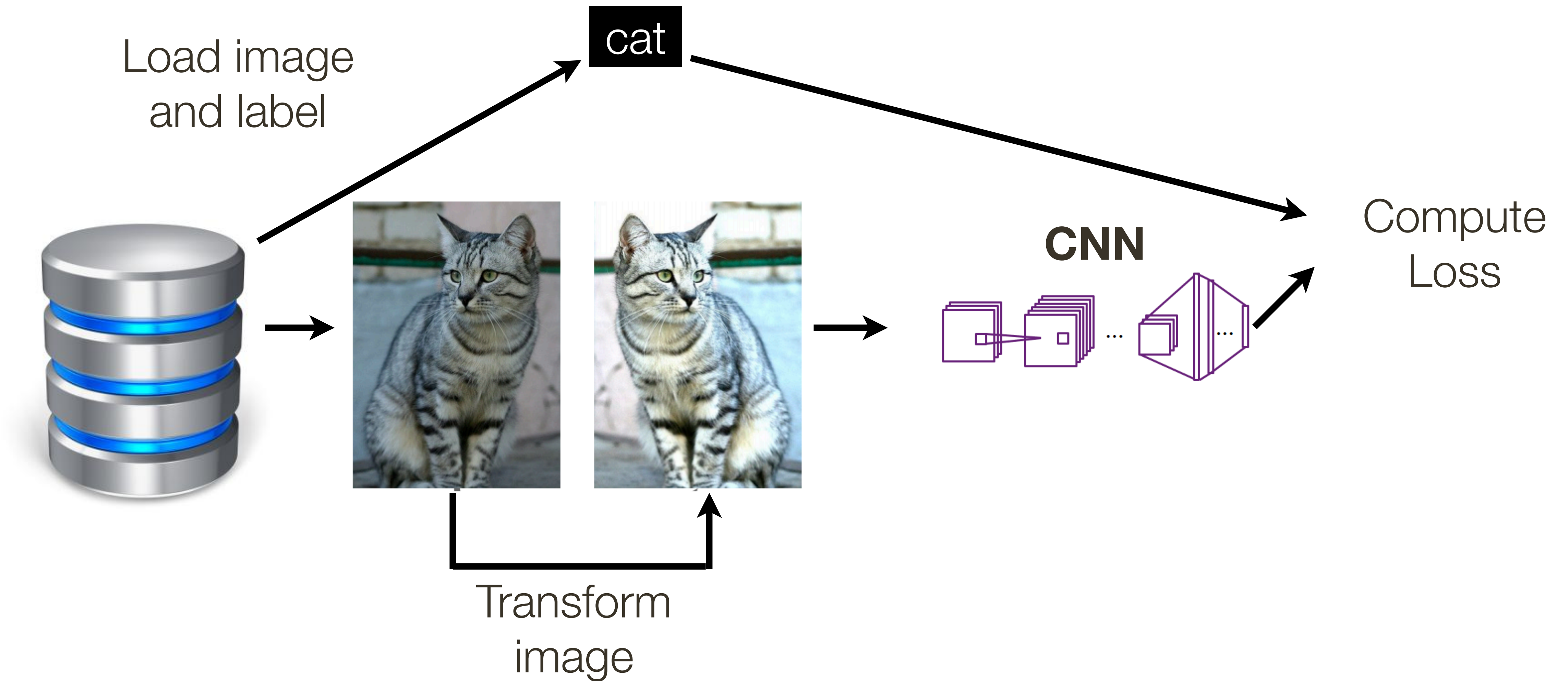
Solution: increase model capacity

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

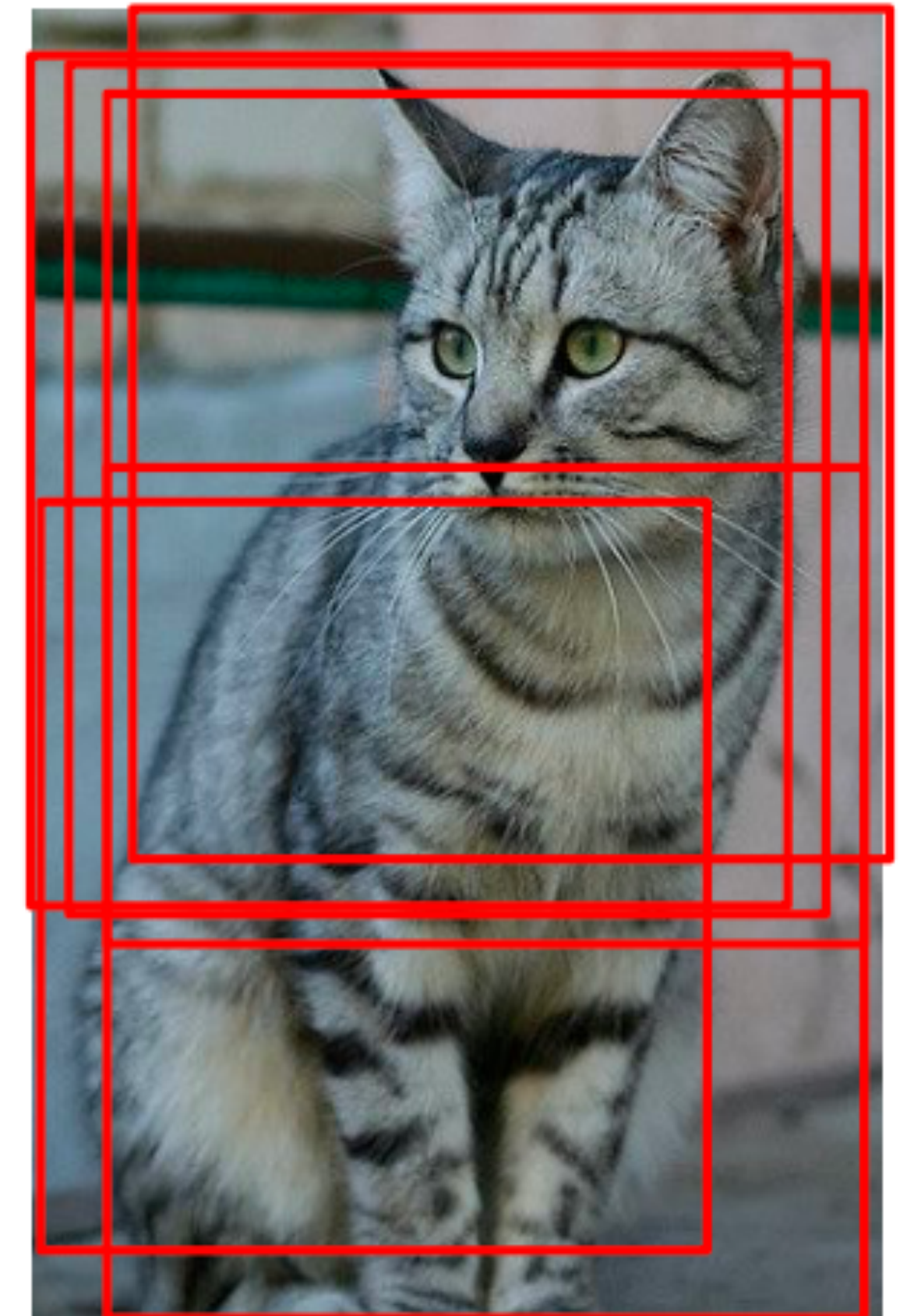
Color Jitter

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



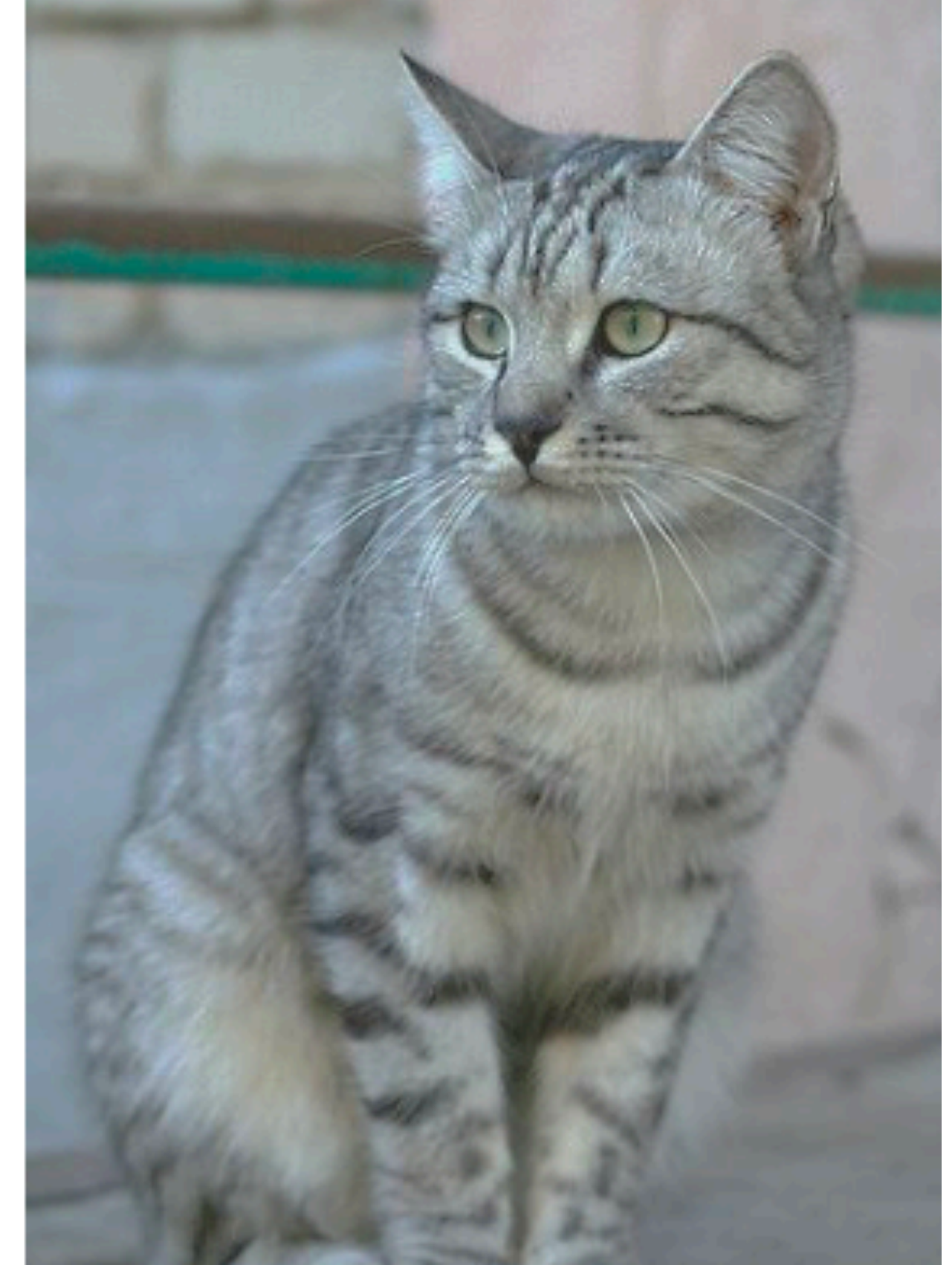
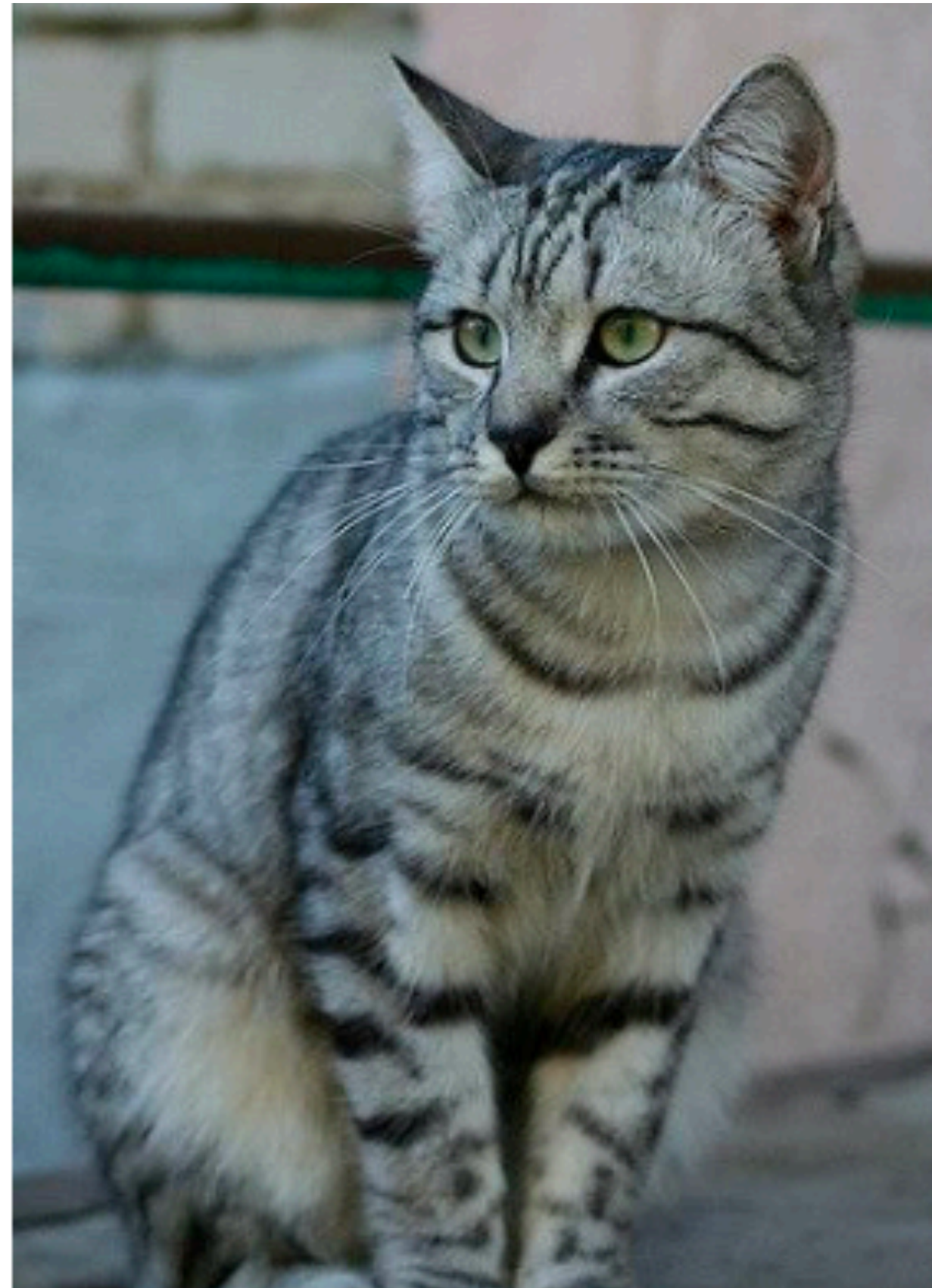
Regularization: Data Augmentation

Horizontal flips

Random crops & scales

Color Jitter

Random perturbations in
contrast and brightness



Activation Function: **Sigmoid**

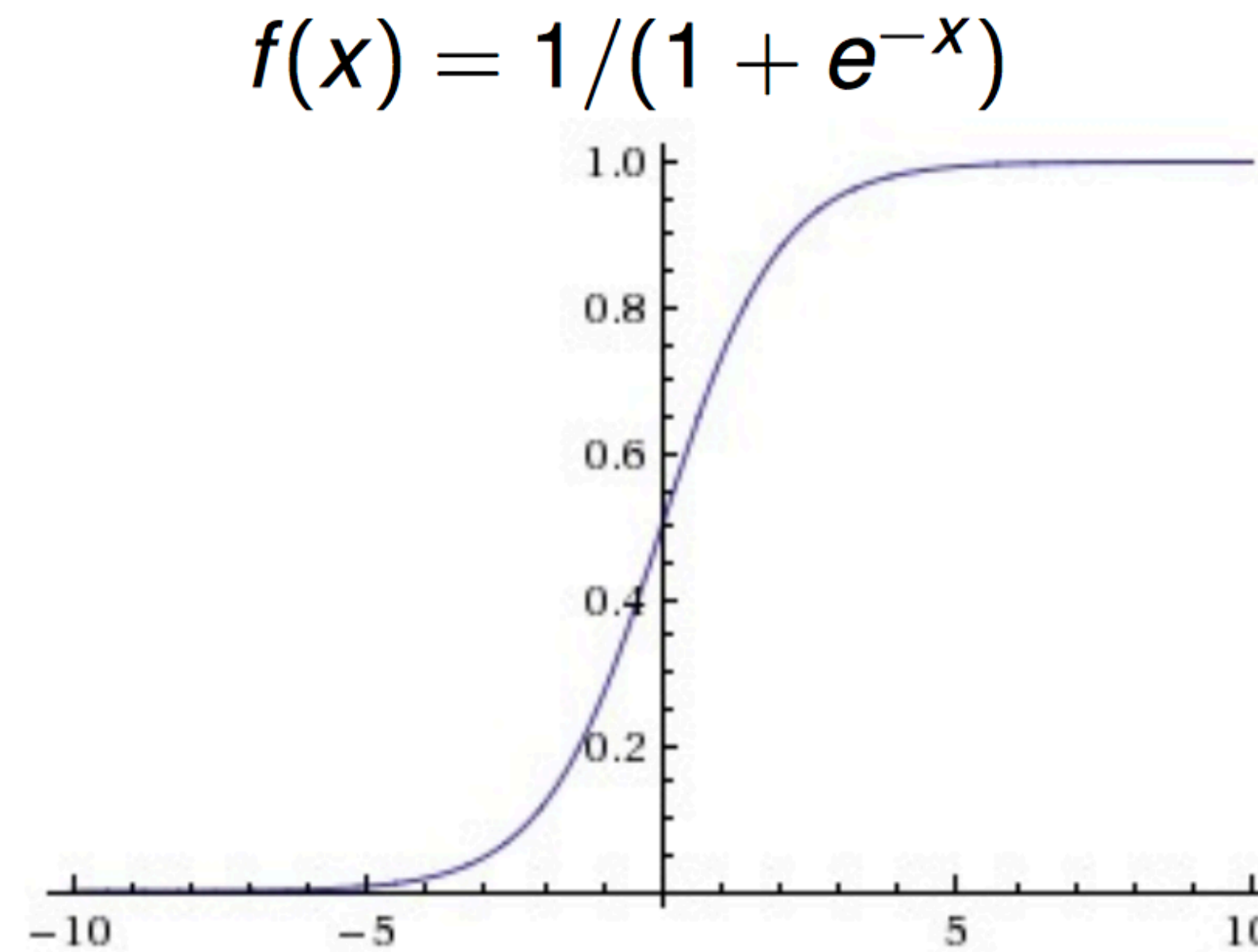


Figure credit: Fei-Fei and Karpathy

Common in many early neural networks

Biological analogy to saturated firing rate of neurons

Maps the input to the range [0, 1]

Activation Function: **ReLU** (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

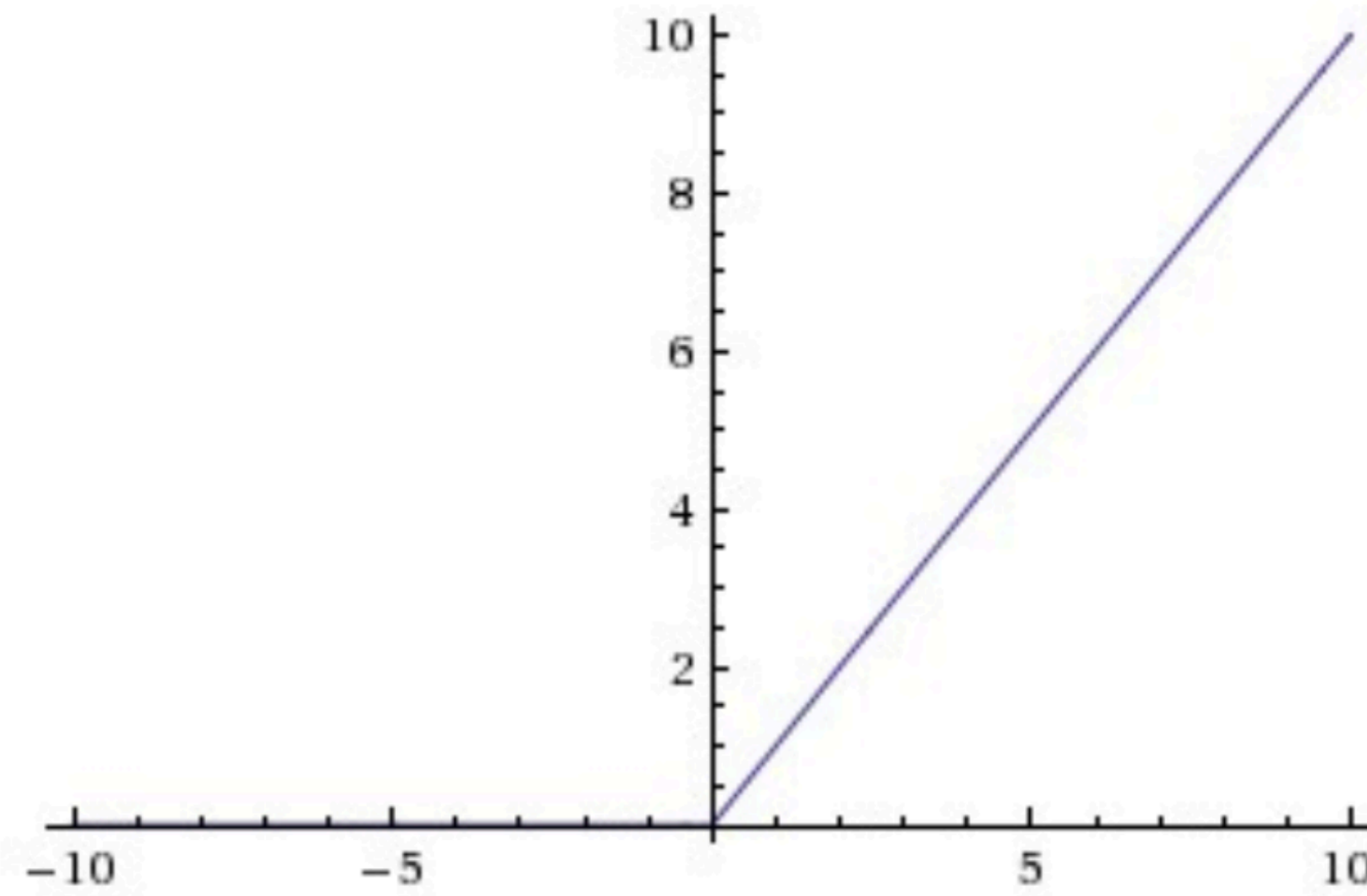


Figure credit: Fei-Fei and Karpathy

Found to accelerate convergence during learning
Used in the most recent neural networks

Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

$$\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

In practice, also learn how to scale and offset:

$$y^{(k)} = \gamma^{(k)} \bar{x}^{(k)} + \beta^{(k)}$$

BN layer parameters

Benefit:

Improves learning (better gradients, higher learning rate, less reliance on initialization)

Typically inserted **before** activation layer

[Ioffe and Szegedy, NIPS 2015]

Regularization: Batch Normalization

Consider what happens at **runtime**, when you are only passing a single sample

$$\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

In practice, also learn how to scale and offset:

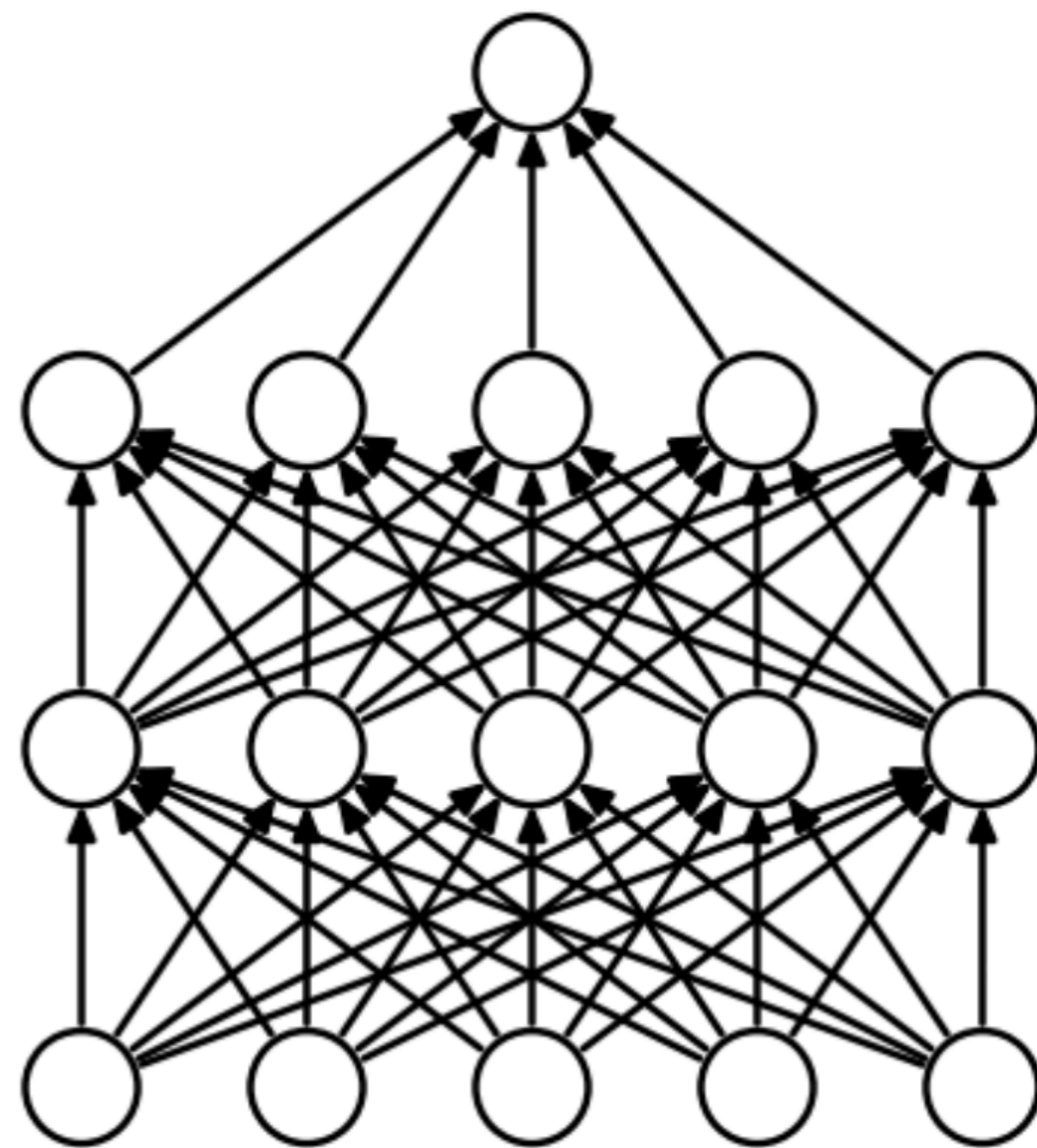
$$y^{(k)} = \gamma^{(k)} \bar{x}^{(k)} + \beta^{(k)}$$

BN layer parameters

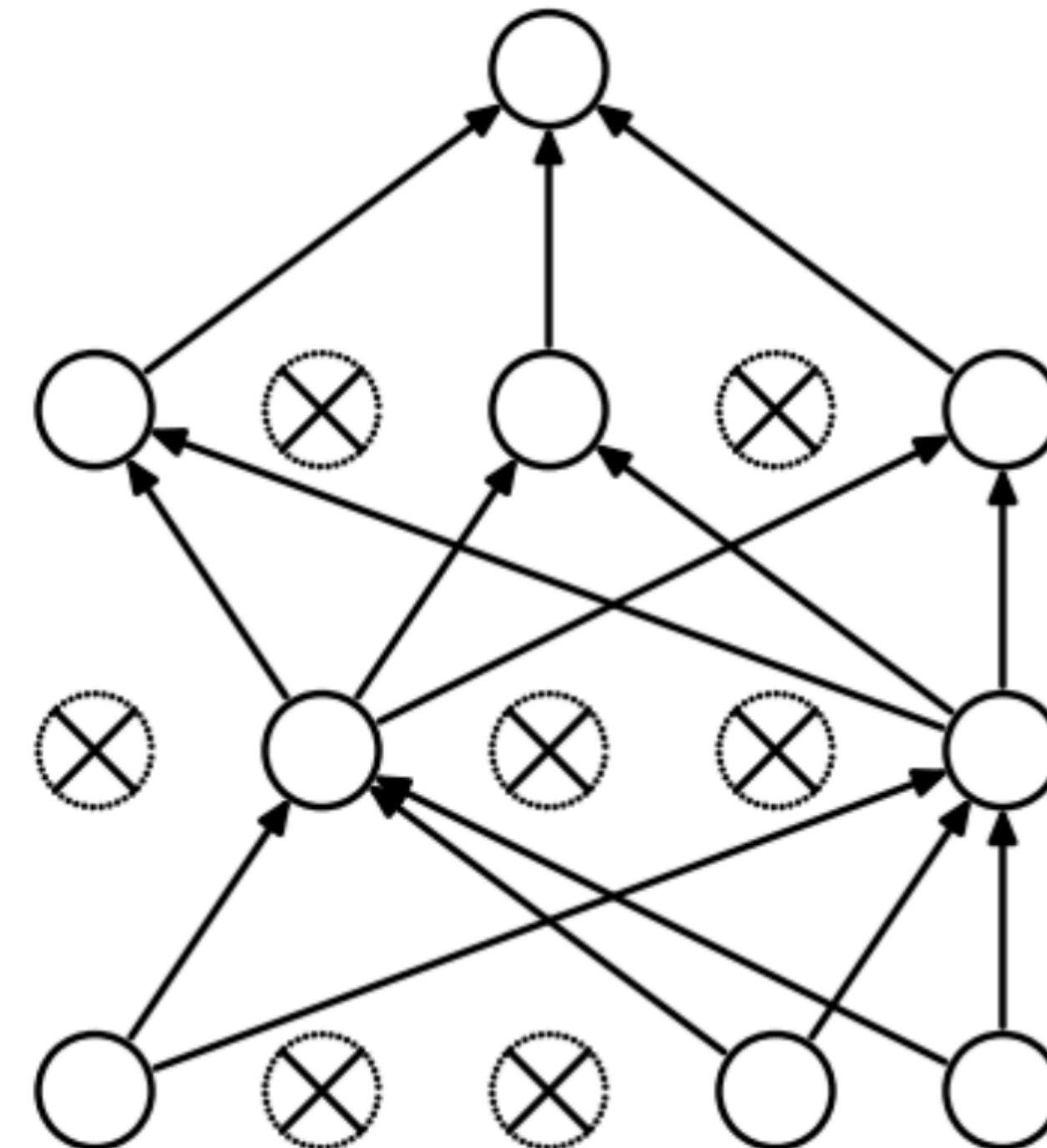
[Ioffe and Szegedy, NIPS 2015]

Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to dropout rate (between 0 to 1)



Standar Neural Network



After Applying **Dropout**

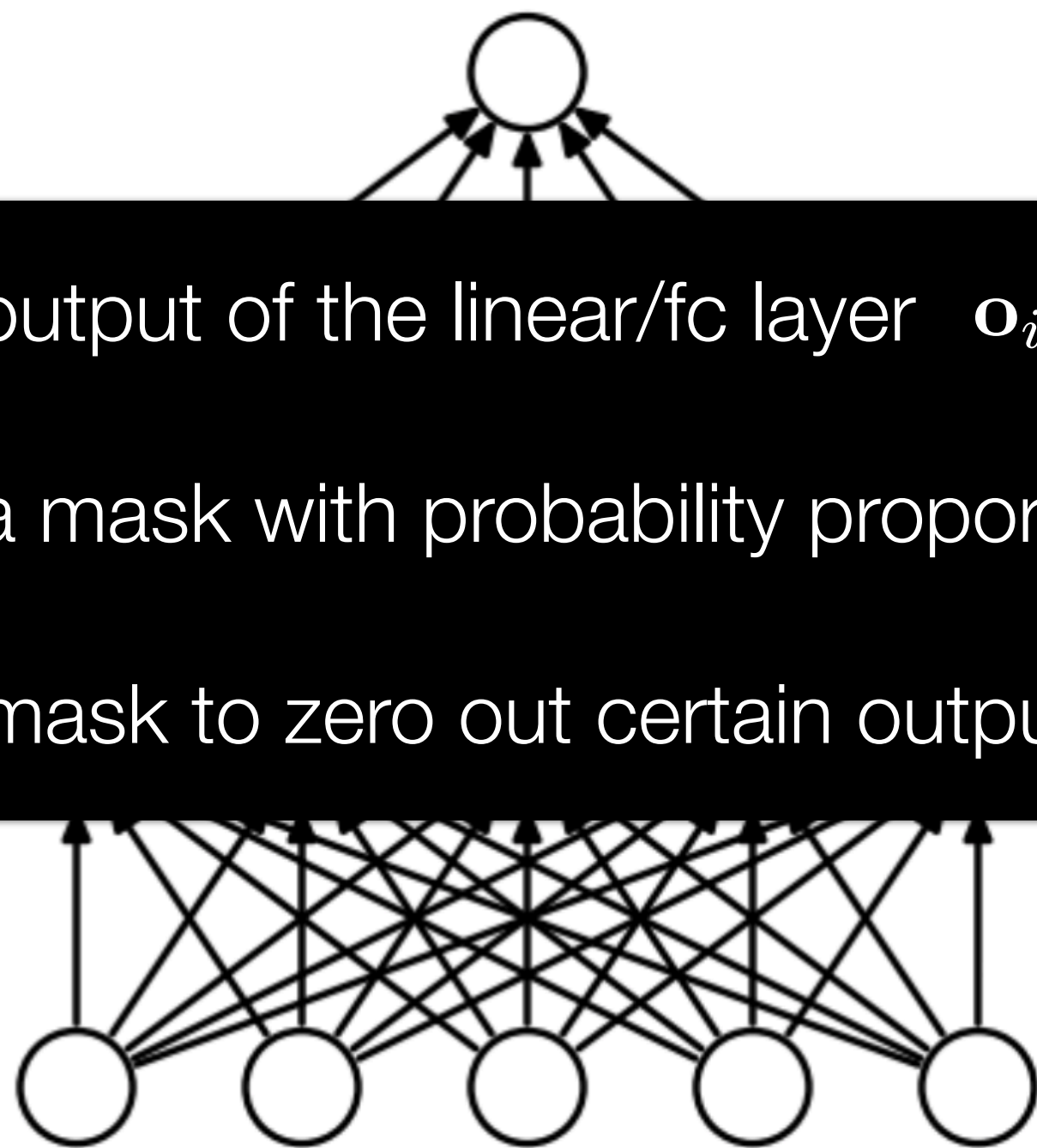
[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

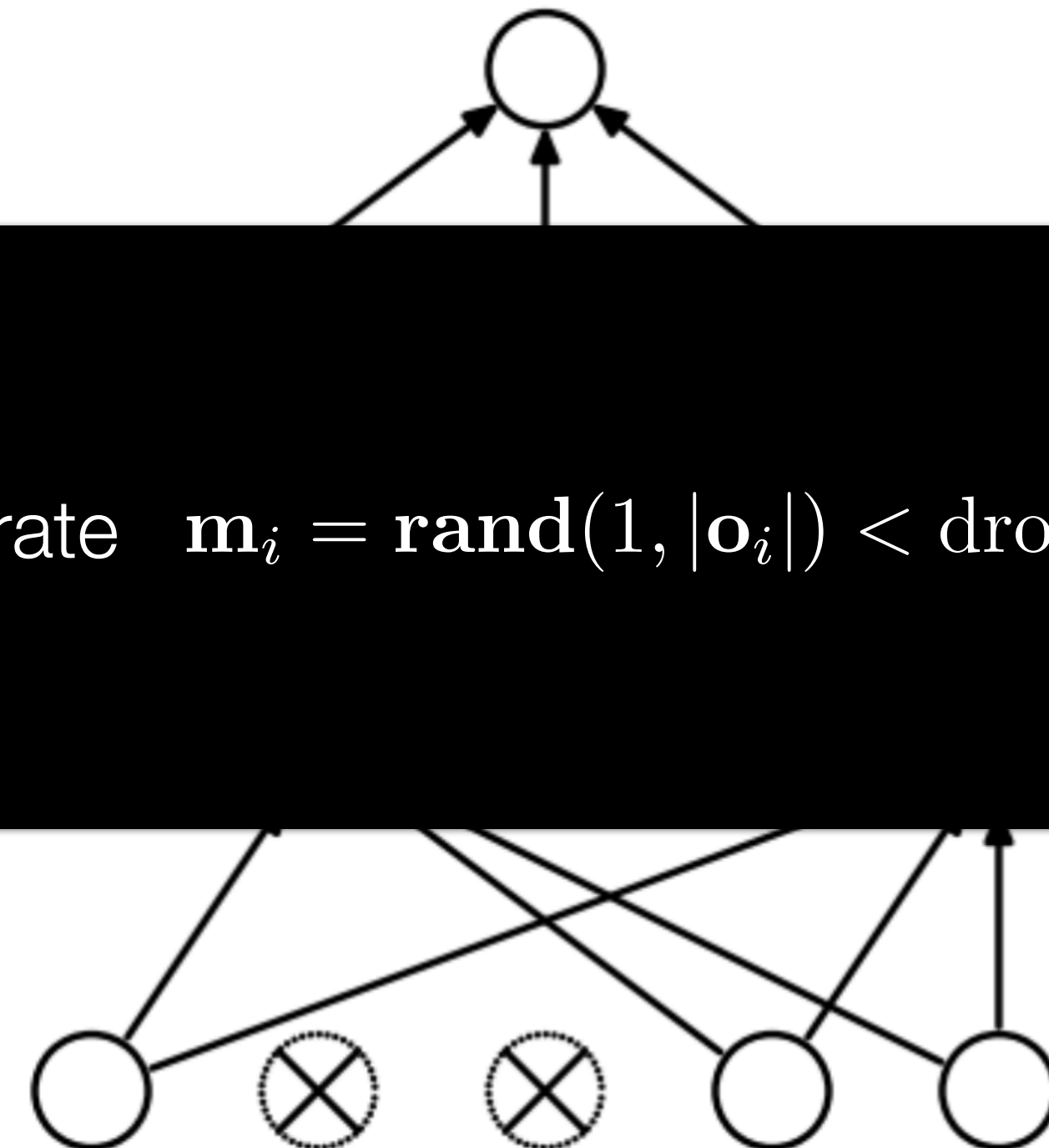
Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to dropout rate (between 0 to 1)

1. Compute output of the linear/fc layer $\mathbf{o}_i = \mathbf{W}_i \cdot \mathbf{x} + \mathbf{b}_i$
2. Compute a mask with probability proportional to dropout rate $\mathbf{m}_i = \mathbf{rand}(1, |\mathbf{o}_i|) < \text{dropout rate}$
3. Apply the mask to zero out certain outputs $\mathbf{o}_i = \mathbf{o}_i \odot \mathbf{m}_i$



Standar Neural Network



After Applying **Dropout**

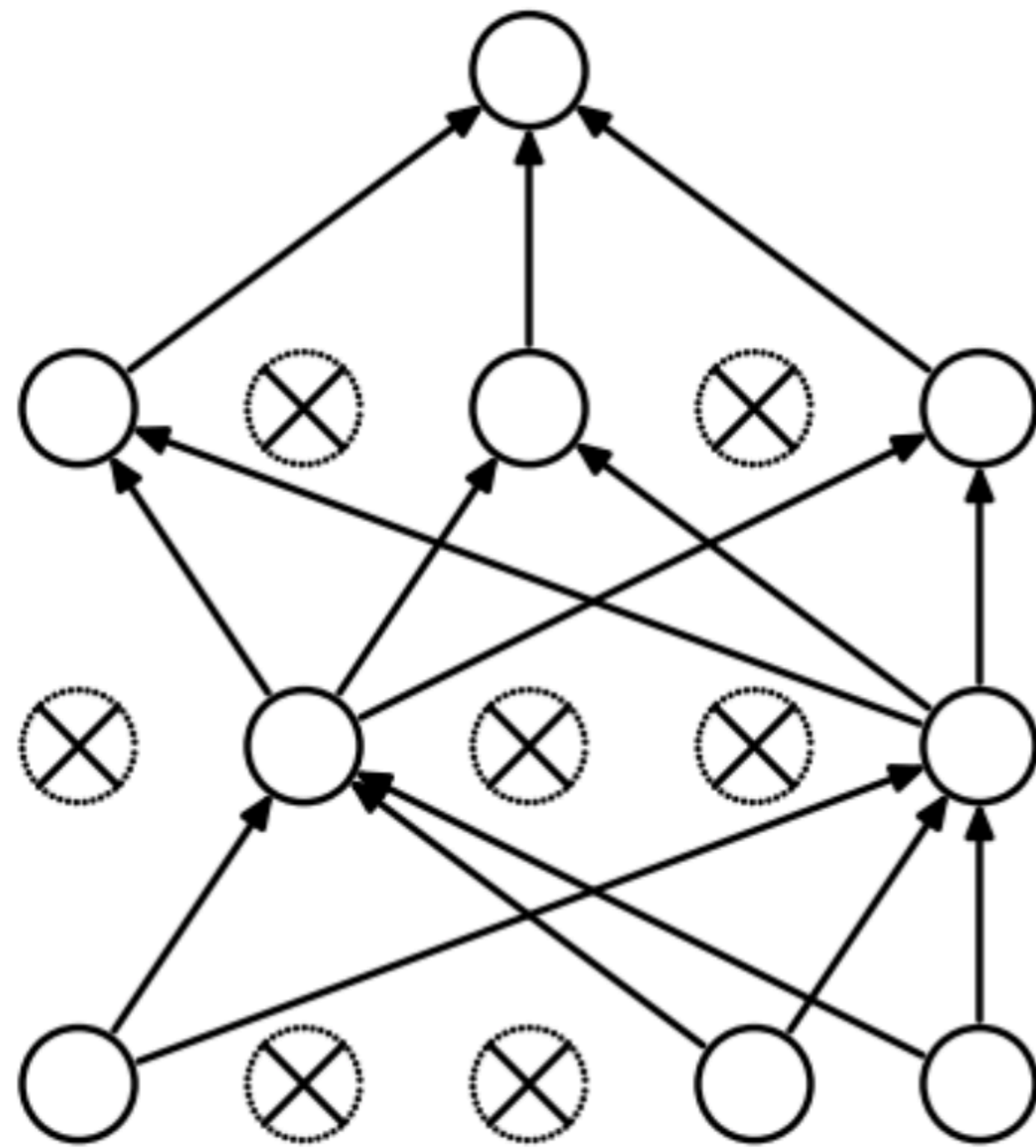
[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to dropout rate (between 0 to 1)

Why is this a good idea?



After Applying **Dropout**

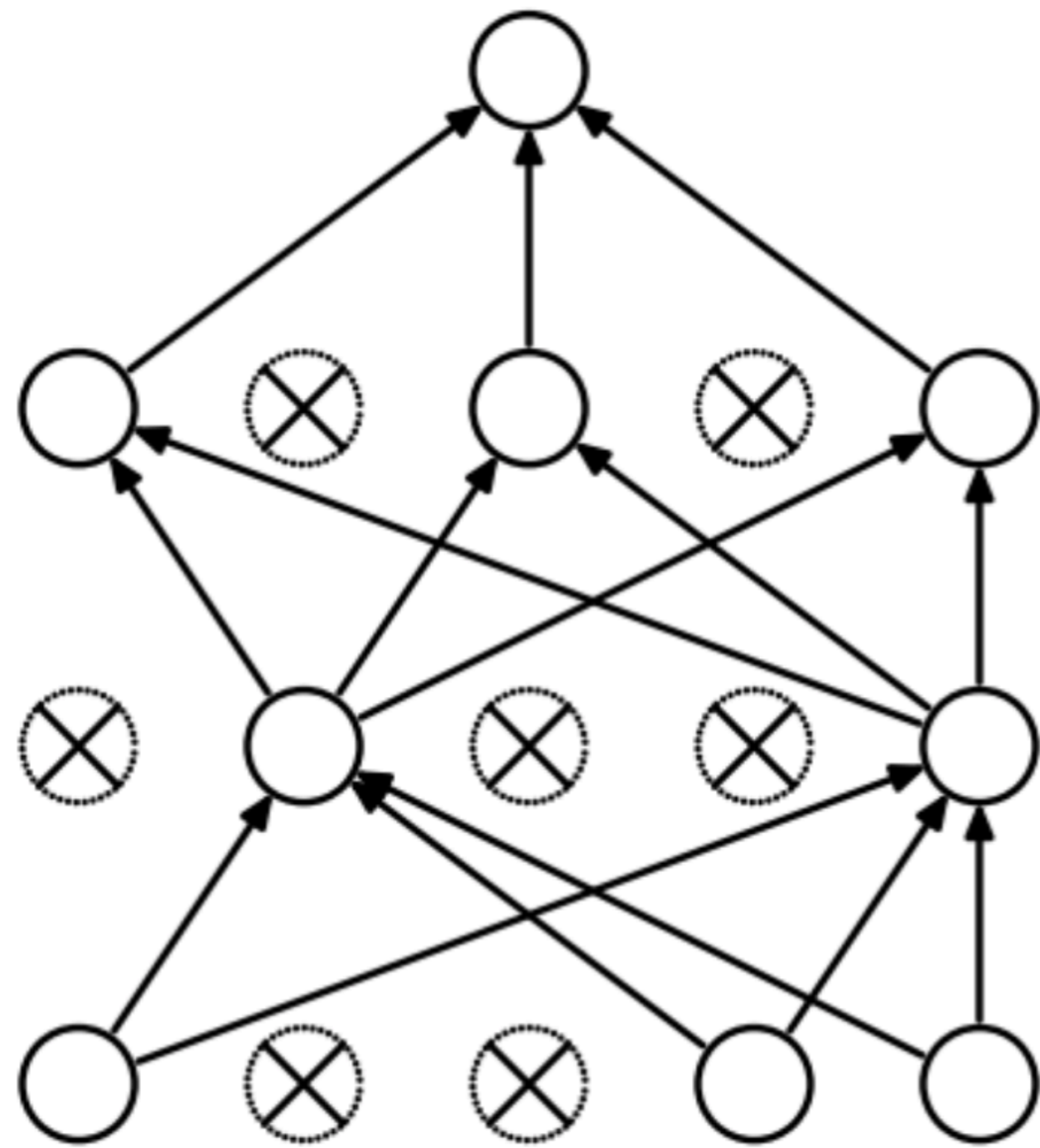
[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout_rate` (between 0 to 1)

Why is this a good idea?



After Applying **Dropout**

Dropout is training an **ensemble of models** that share parameters

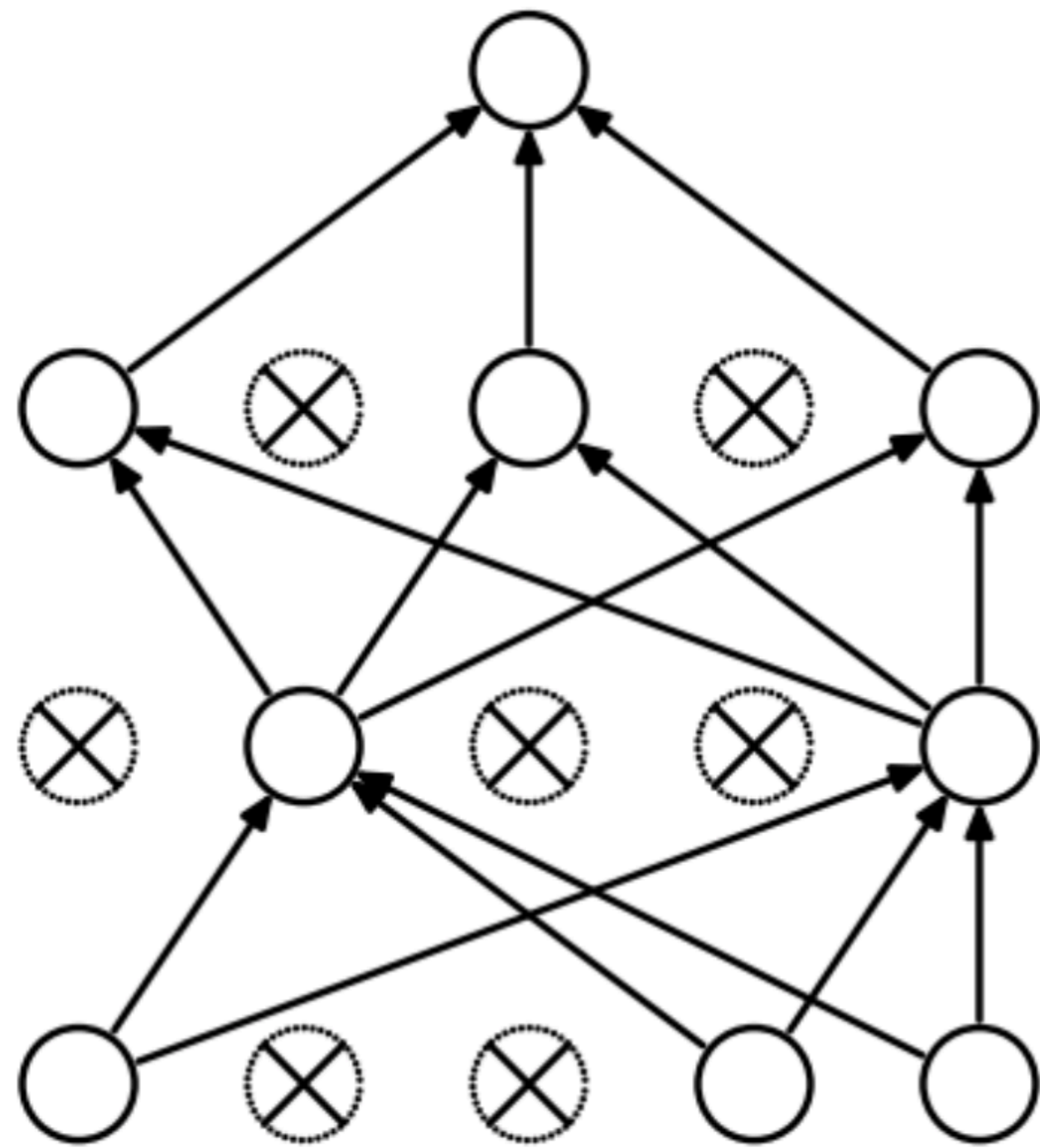
Each binary mask (generated in the forward pass) is one model that is trained on (approximately) one data point

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Regularization: Dropout (at test time)

Randomly **set some neurons to zero** in the forward pass, with probability proportional to dropout rate (between 0 to 1)



After Applying **Dropout**

At test time, **integrate out all the models** in the ensemble

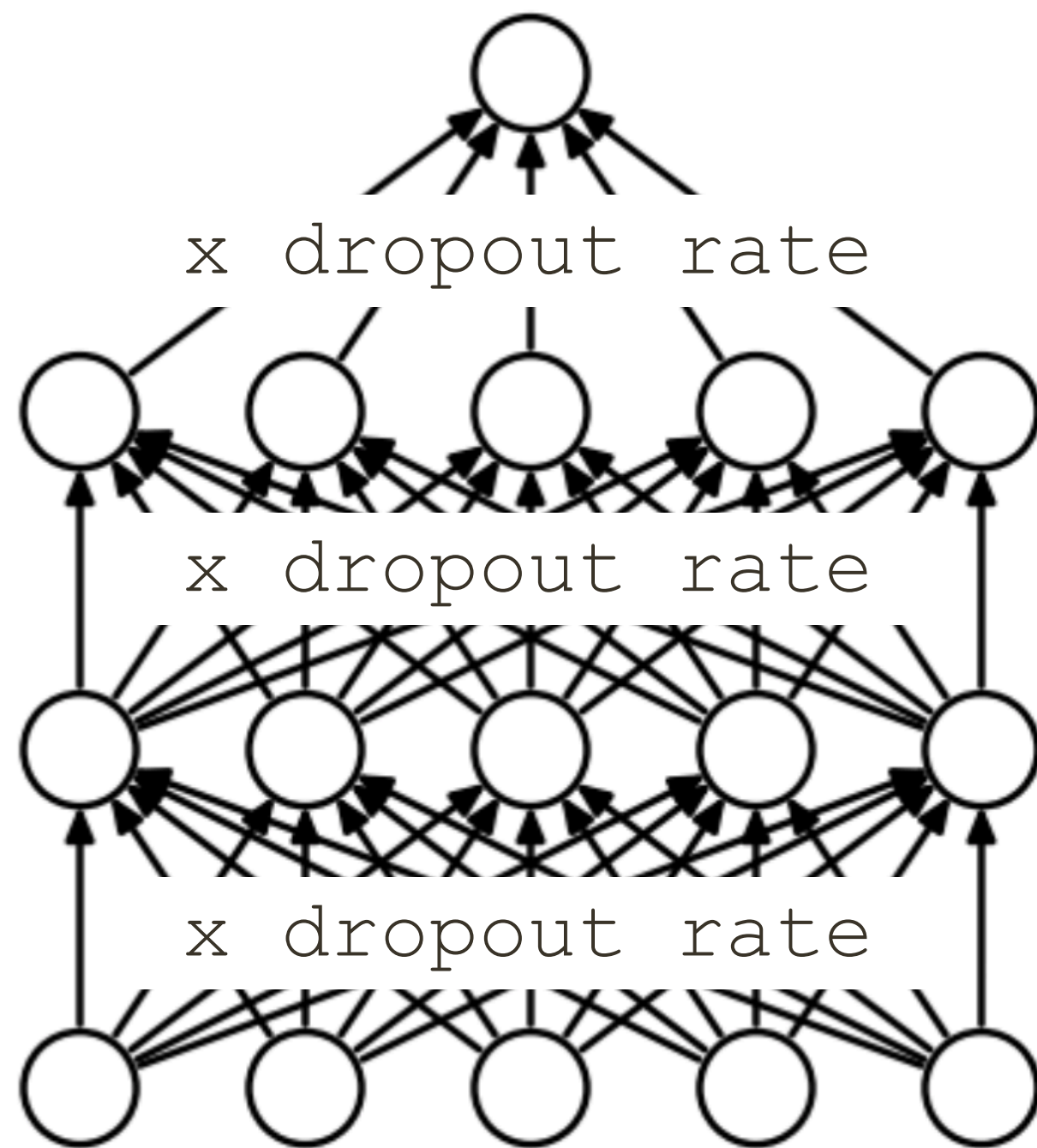
Monte Carlo approximation: many forward passes with different masks and average all predictions

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Regularization: Dropout (at test time)

Randomly **set some neurons to zero** in the forward pass, with probability proportional to dropout rate (between 0 to 1)



At test time, **integrate out all the models** in the ensemble

Monte Carlo approximation: many forward passes with different masks and average all predictions

Equivalent to forward pass with all connections on and **scaling of the outputs** by dropout rate

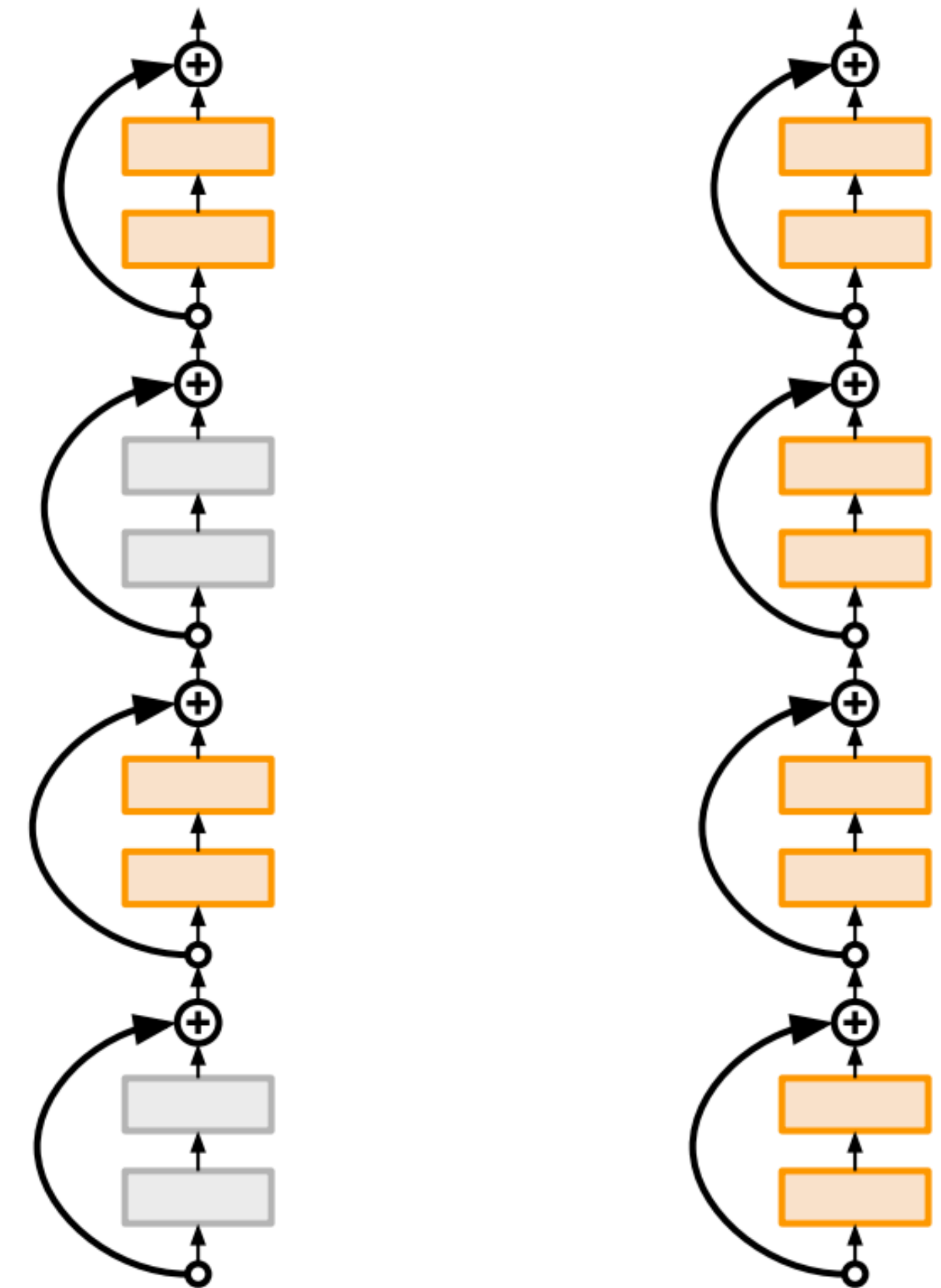
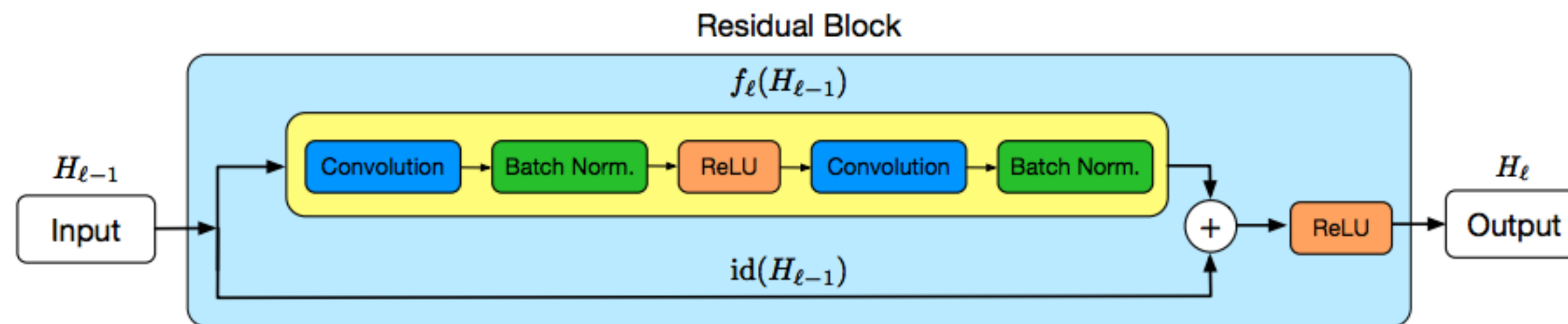
Regularization: Stochastic Depth

[Huang et al., ECCV 2016]

Effectively “dropout” but for layers

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

Effectively trains a collection of neural networks



Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

~ 2% improved performance in practice

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

~ 2% improved performance in practice

Alternative: Multiple snapshots of the single model during training!

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

~ 2% improved performance in practice

Alternative: Multiple snapshots of the single model during training!

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



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

IM  GENET

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

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

Categorization

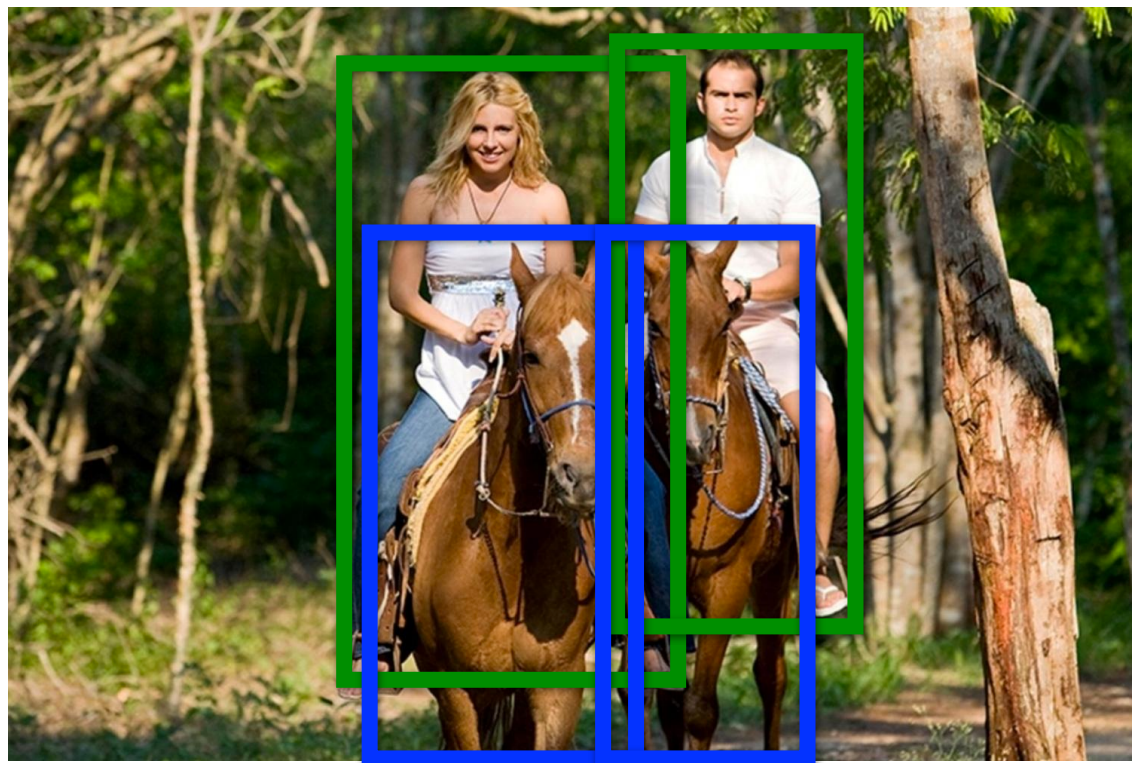


Multi-**class**: Horse
 Church
 Toothbrush
 Person

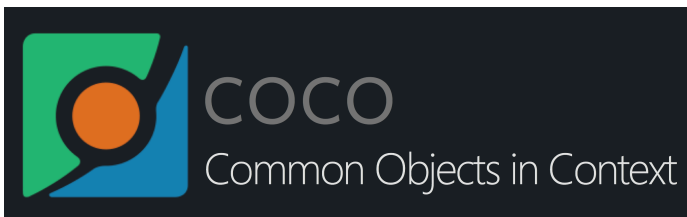


Multi-**label**: **Horse**
 Church
 Toothbrush
 Person

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

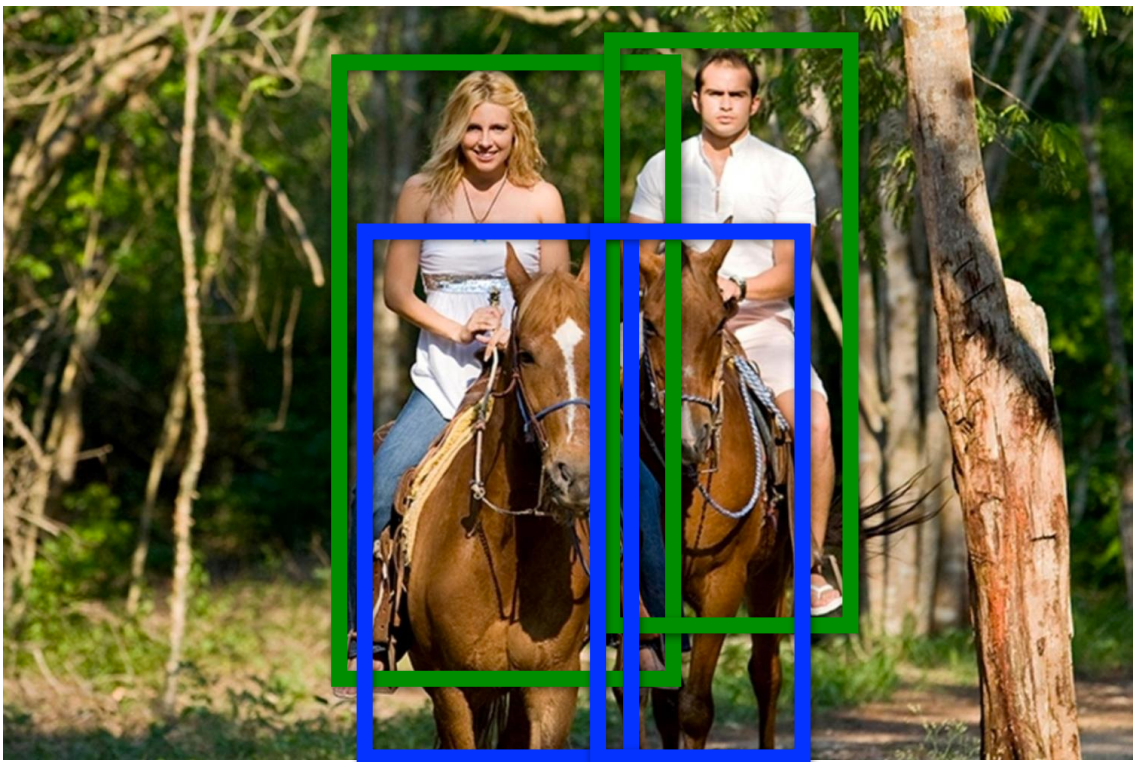


Multi-**class**: Horse
Church
Toothbrush
Person

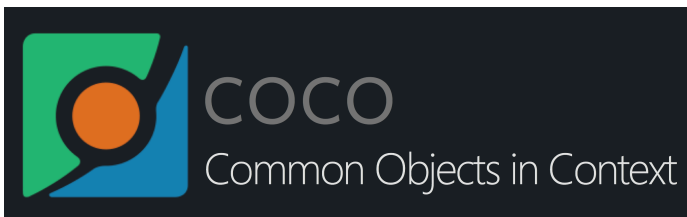


Multi-**label**: **Horse**
Church
Toothbrush
Person

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

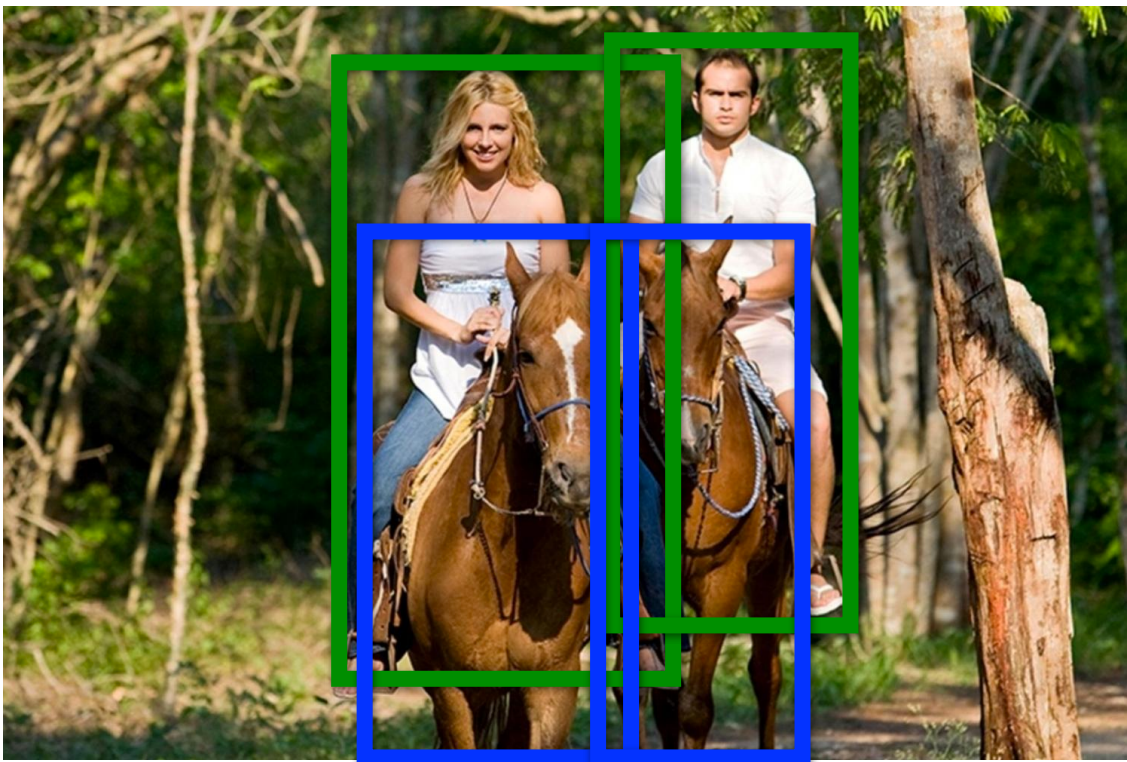


Multi-**class**: Horse
Church
Toothbrush
Person

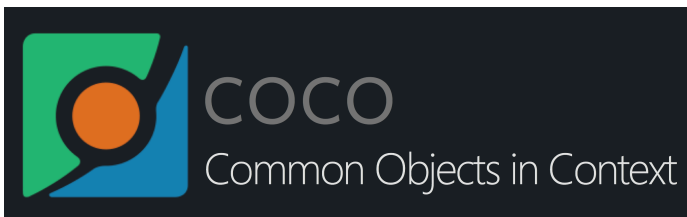


Multi-**label**: **Horse**
Church
Toothbrush
Person

Detection



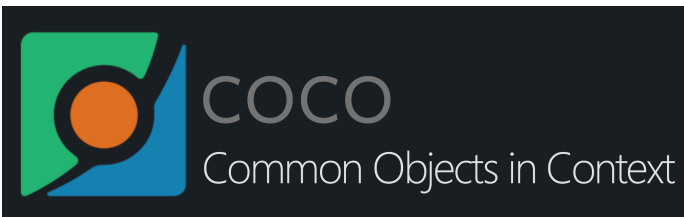
Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)



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

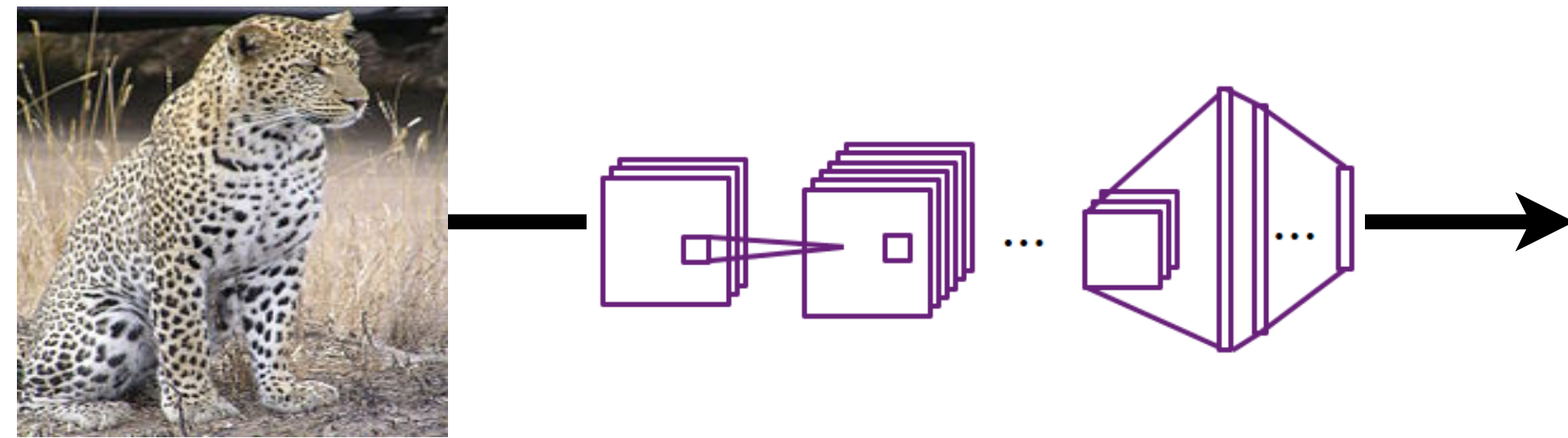
Object **Classification**



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

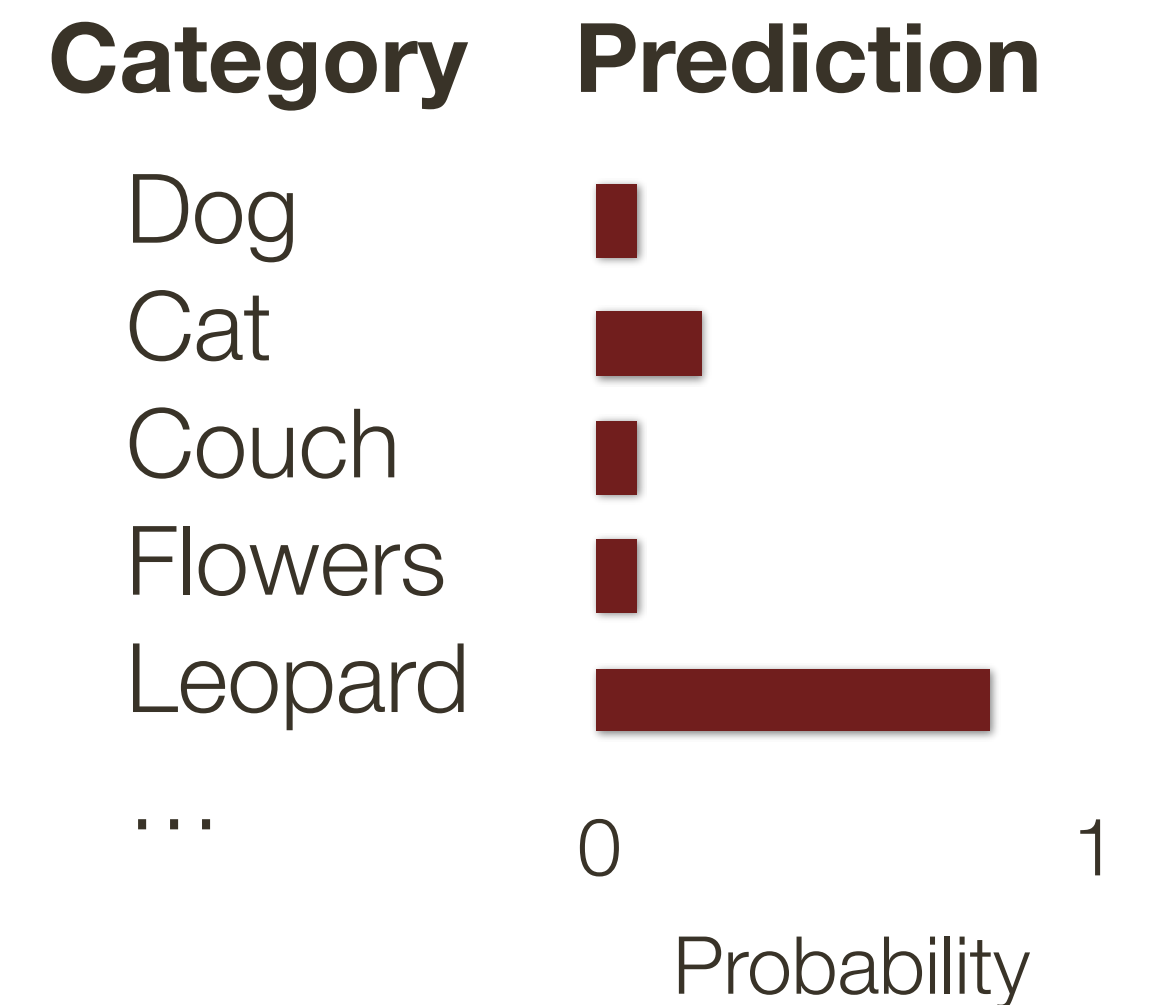
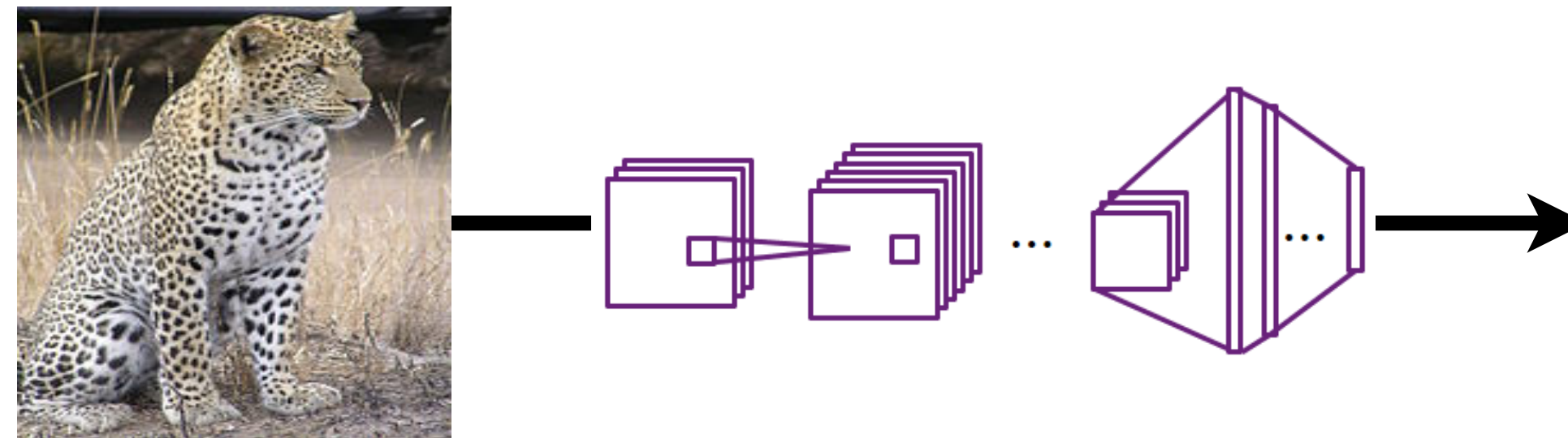
Object **Classification**



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

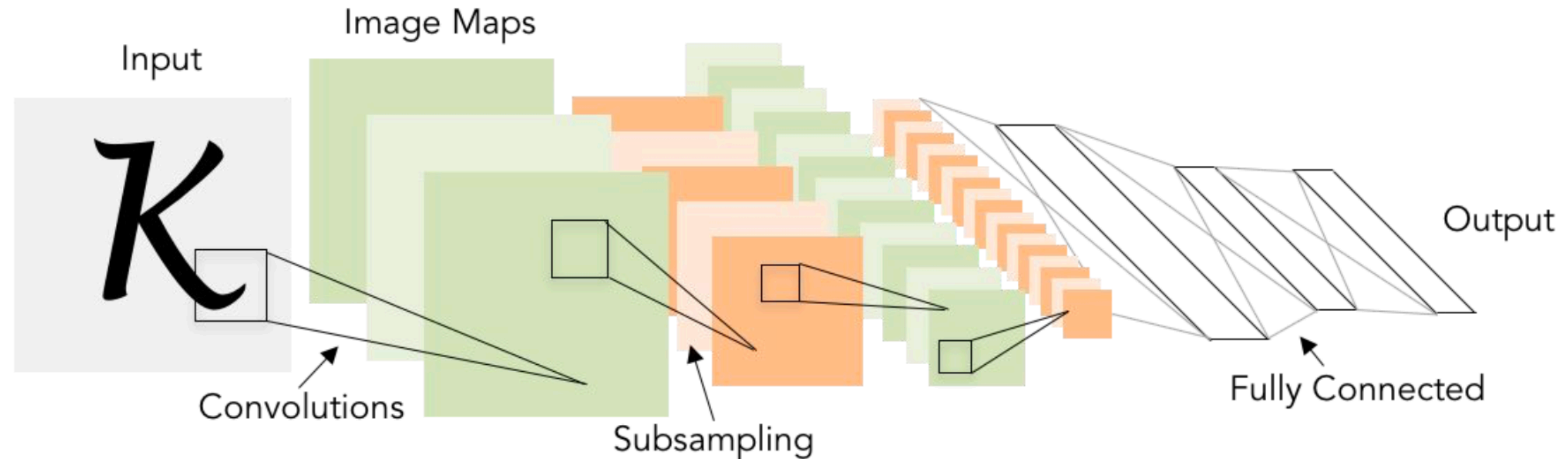
Object Classification



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

CNN Architectures: LeNet-5

[LeCun et al., 1998]



Architecture: CONV $\rightarrow$ POOL $\rightarrow$ CONV $\rightarrow$ POOL $\rightarrow$ FC $\rightarrow$ FC

Conv filters: 5x5, Stride: 1

Pooling: 2x2, Stride: 2

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

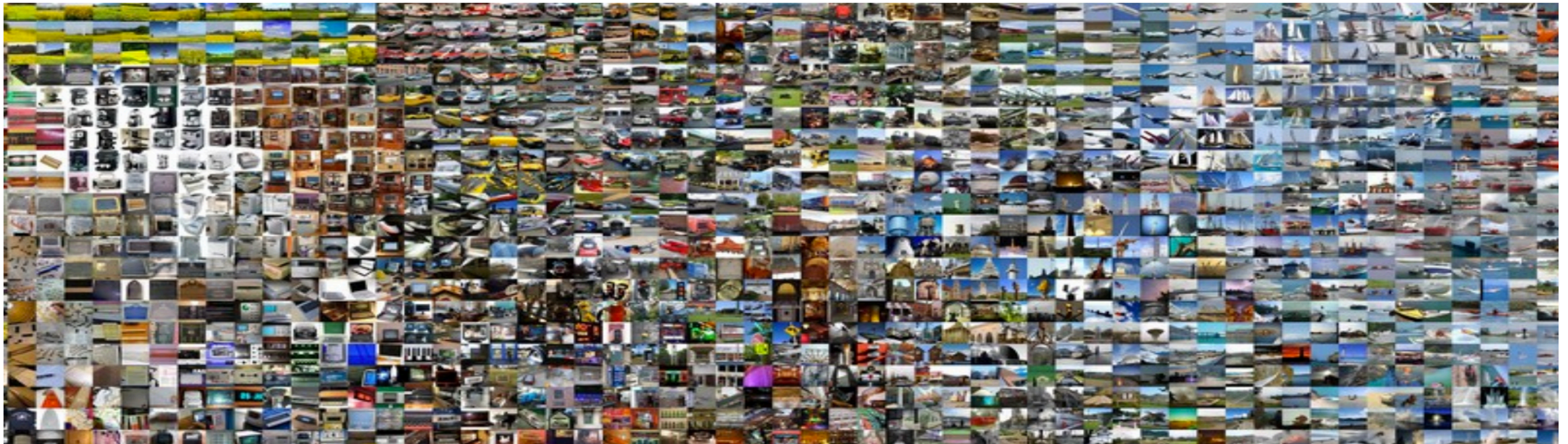
utensil
appliance
tool
musical instrument

Scene

room
geological formation

Food

beverage



ImageNet **Competition** (ILSVRC)

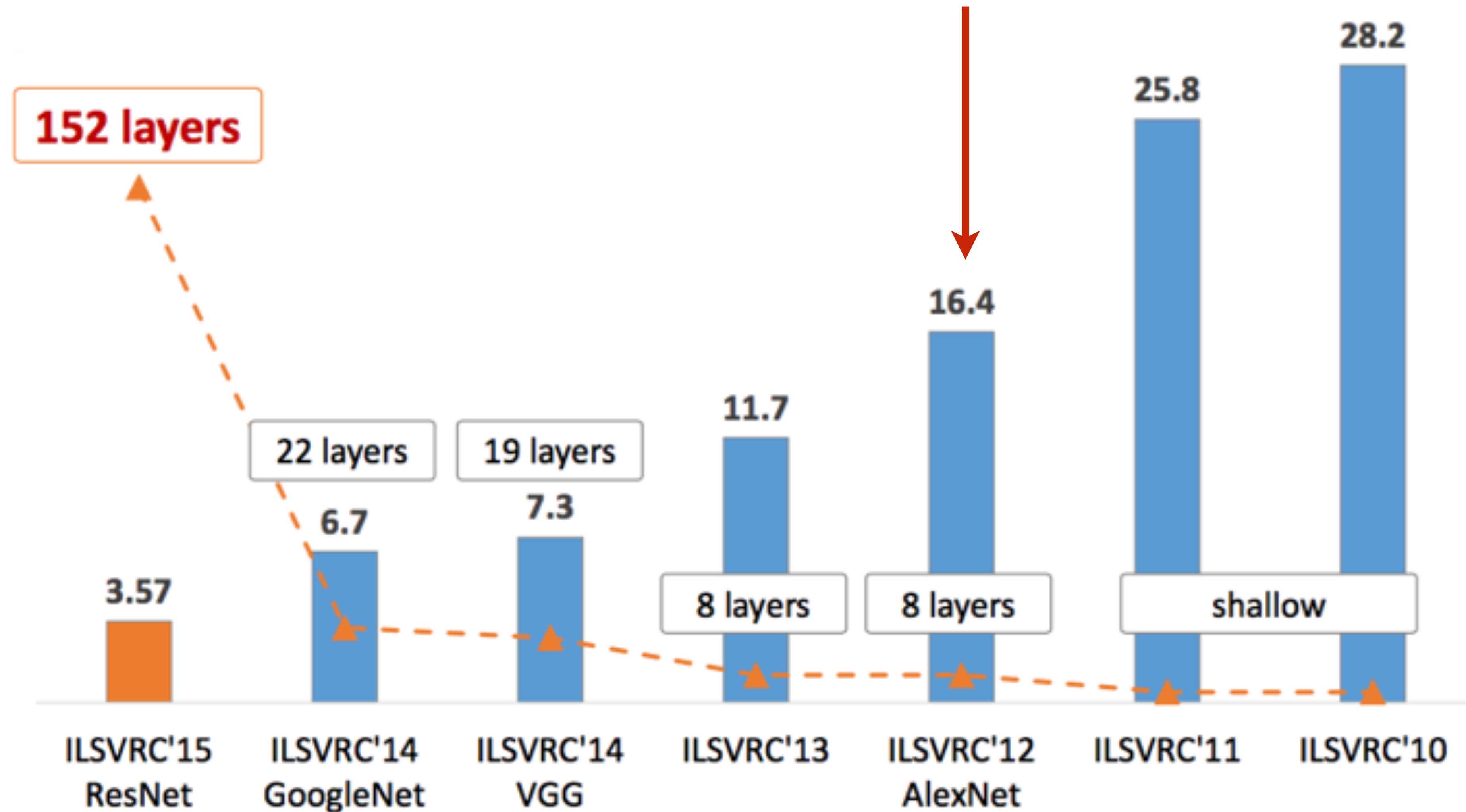
Annual competition of image classification at scale

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

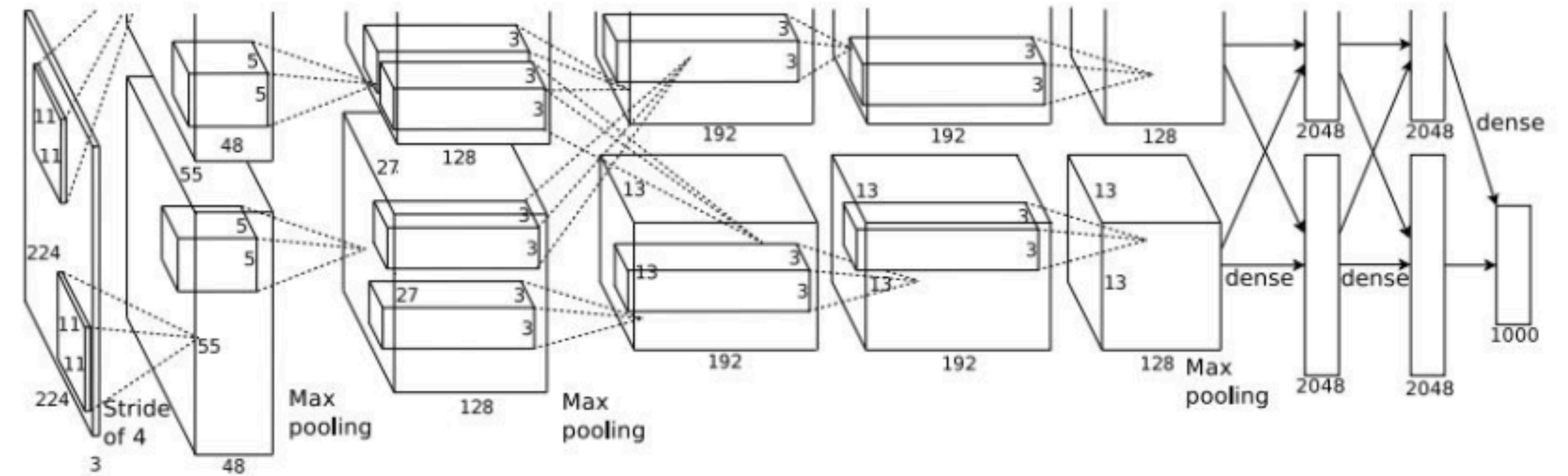
Scoring: need to predict true label within top K (K=5)



ILSVRC winner 2012



AlexNet



Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

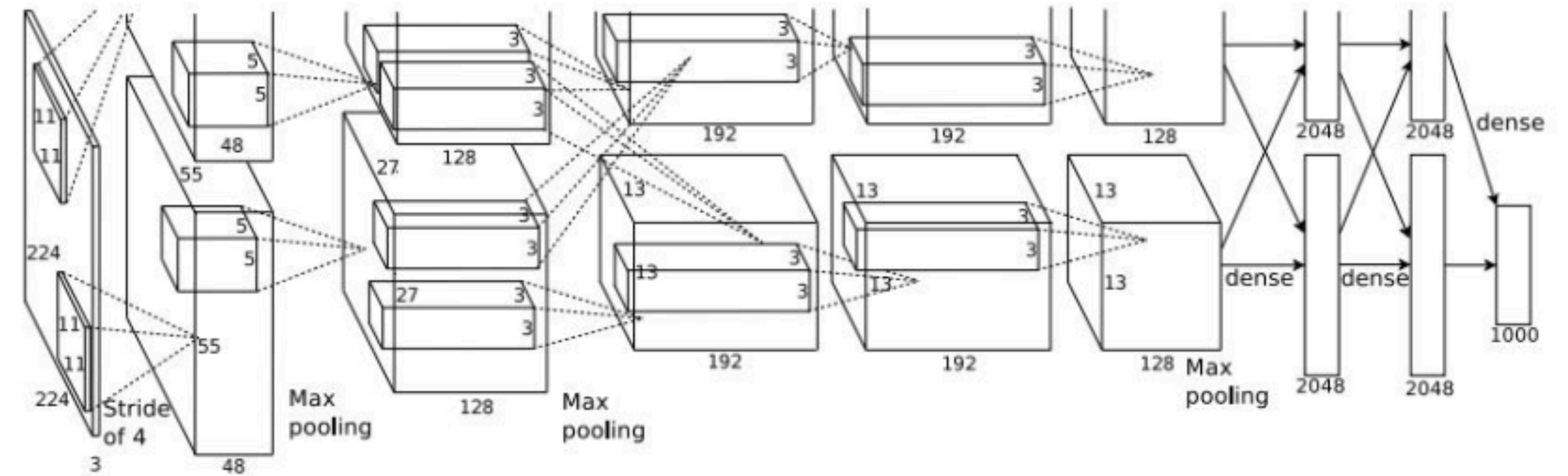
FC7

FC8

[Krizhevsky et al., 2012]

Input: 227 x 227 x 3 images

AlexNet



Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

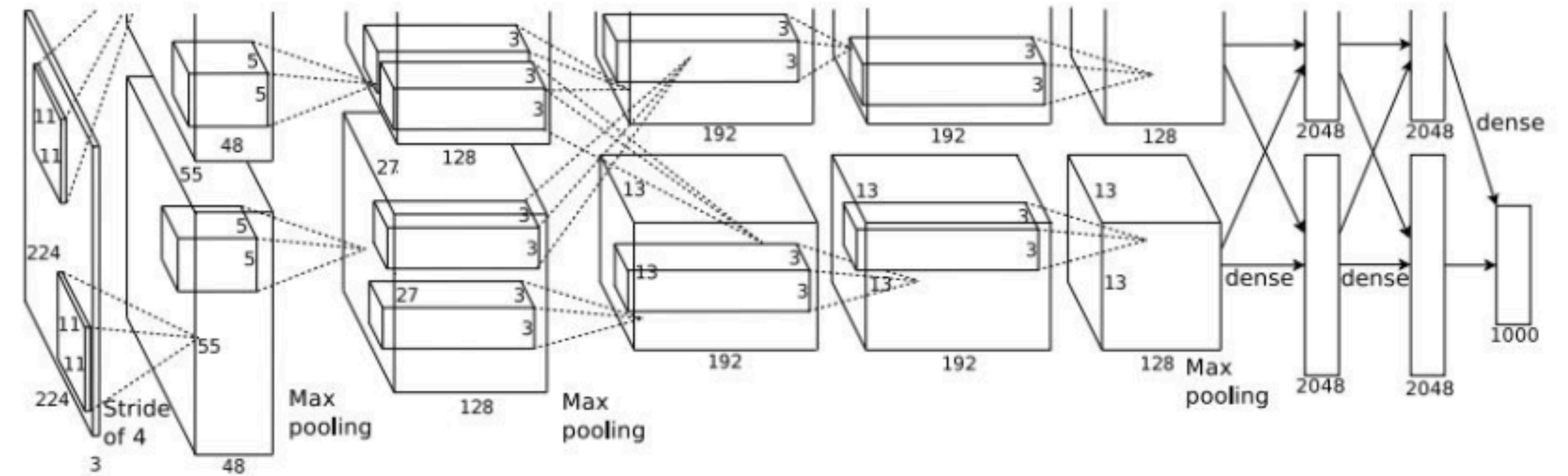
FC8

[Krizhevsky et al., 2012]

Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

AlexNet



Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

FC8

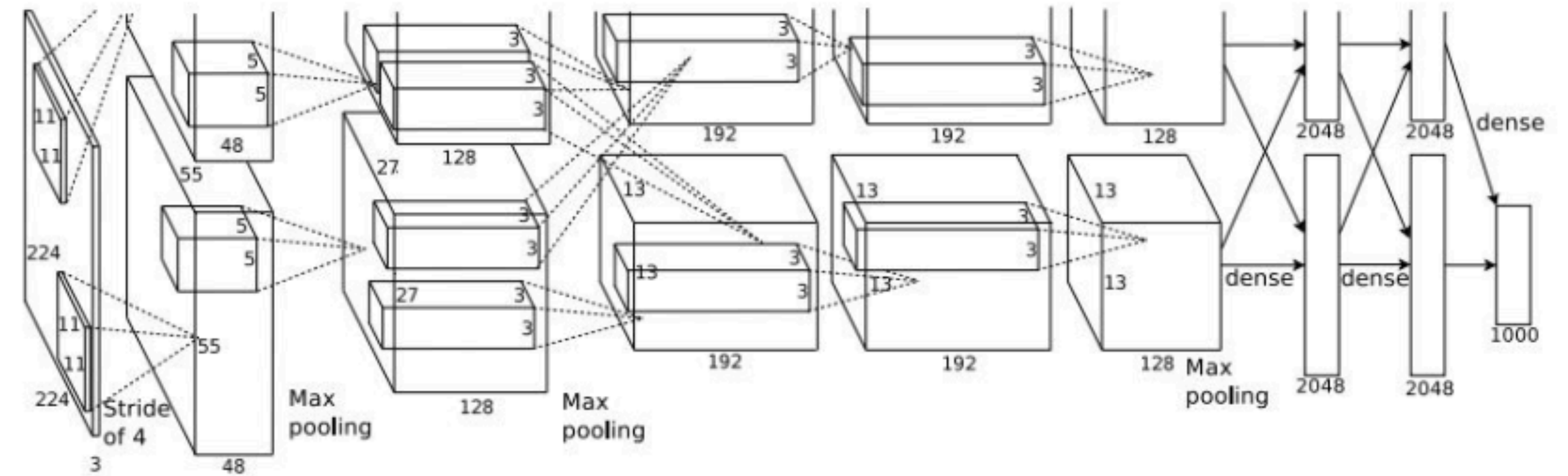
[Krizhevsky et al., 2012]

Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

Output: 55 x 55 x 96

AlexNet



Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

FC8

[Krizhevsky et al., 2012]

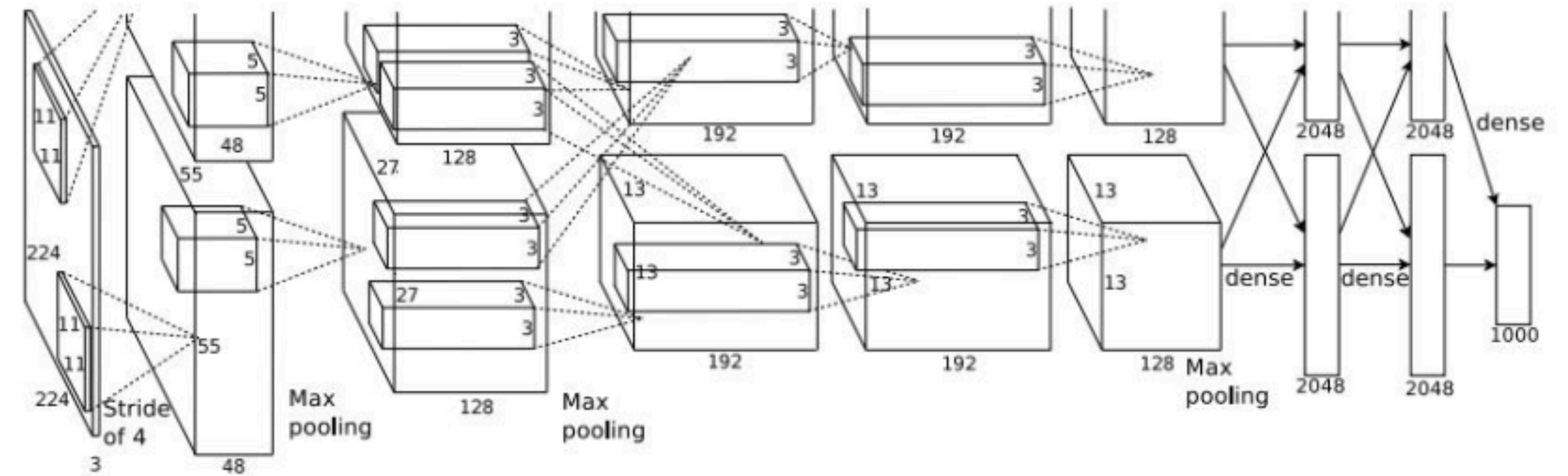
Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

Output: 55 x 55 x 96

Parameters: 35K

AlexNet



[Krizhevsky et al., 2012]

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

FC8

Input: 227 x 227 x 3 images

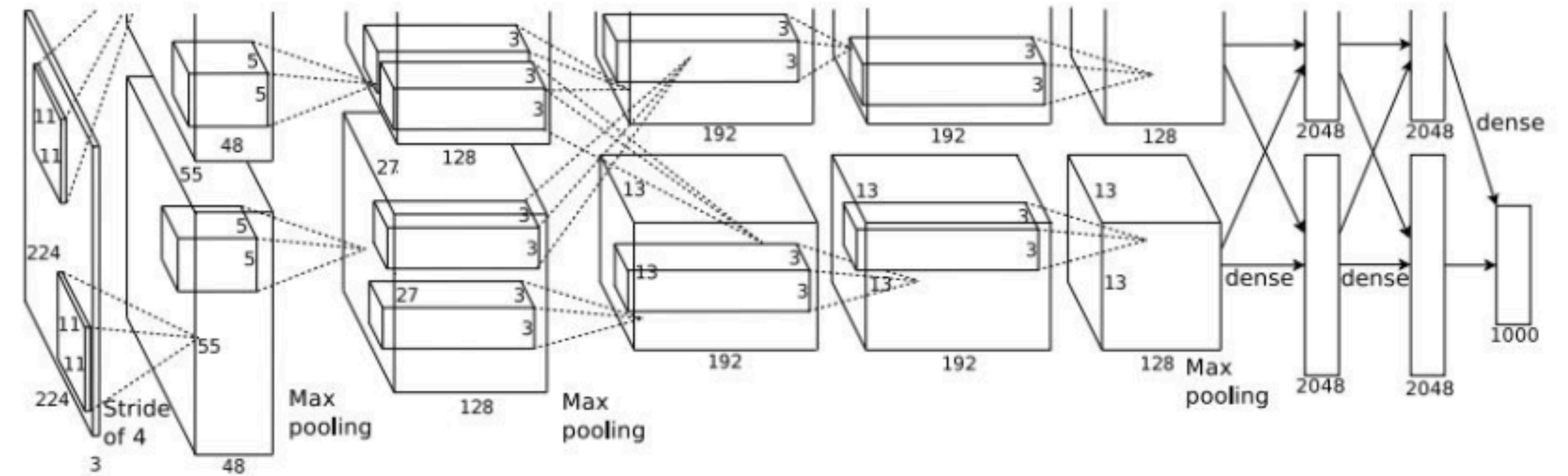
CONV1: 96 11 x 11 filters applied at stride 4

Output: 55 x 55 x 96

Parameters: 35K

MAX POOL1: 96 11 x 11 filters applied at stride 4

AlexNet



[Krizhevsky et al., 2012]

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

FC8

Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

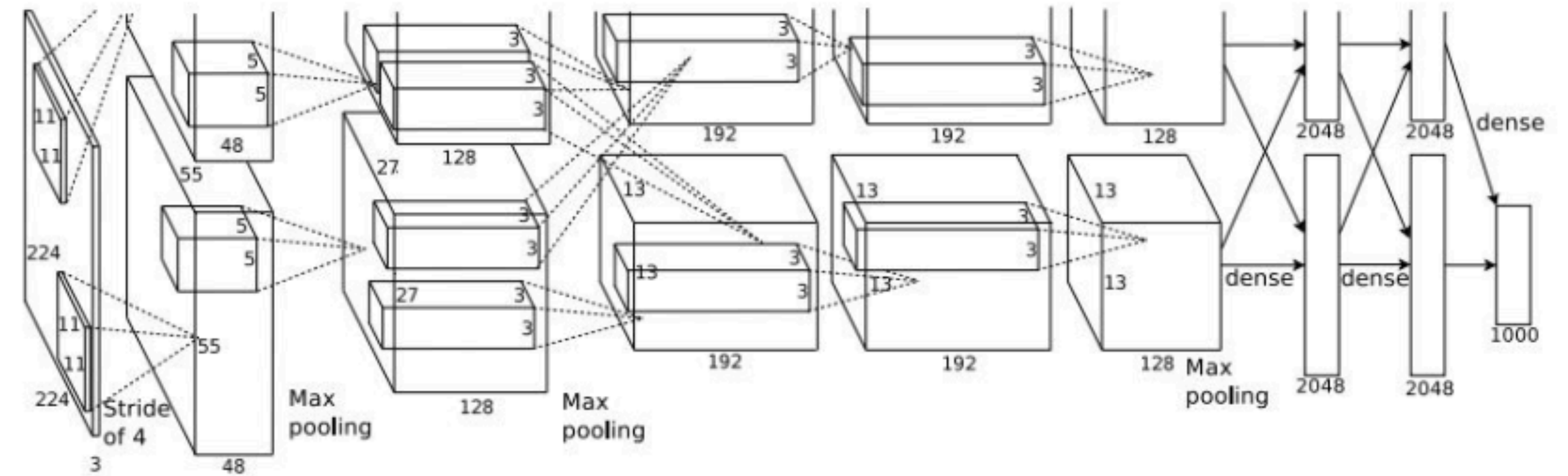
Output: 55 x 55 x 96

Parameters: 35K

MAX POOL1: 96 11 x 11 filters applied at stride 4

Output: 27 x 27 x 96

AlexNet



[Krizhevsky et al., 2012]

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

FC8

Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

Output: 55 x 55 x 96

Parameters: 35K

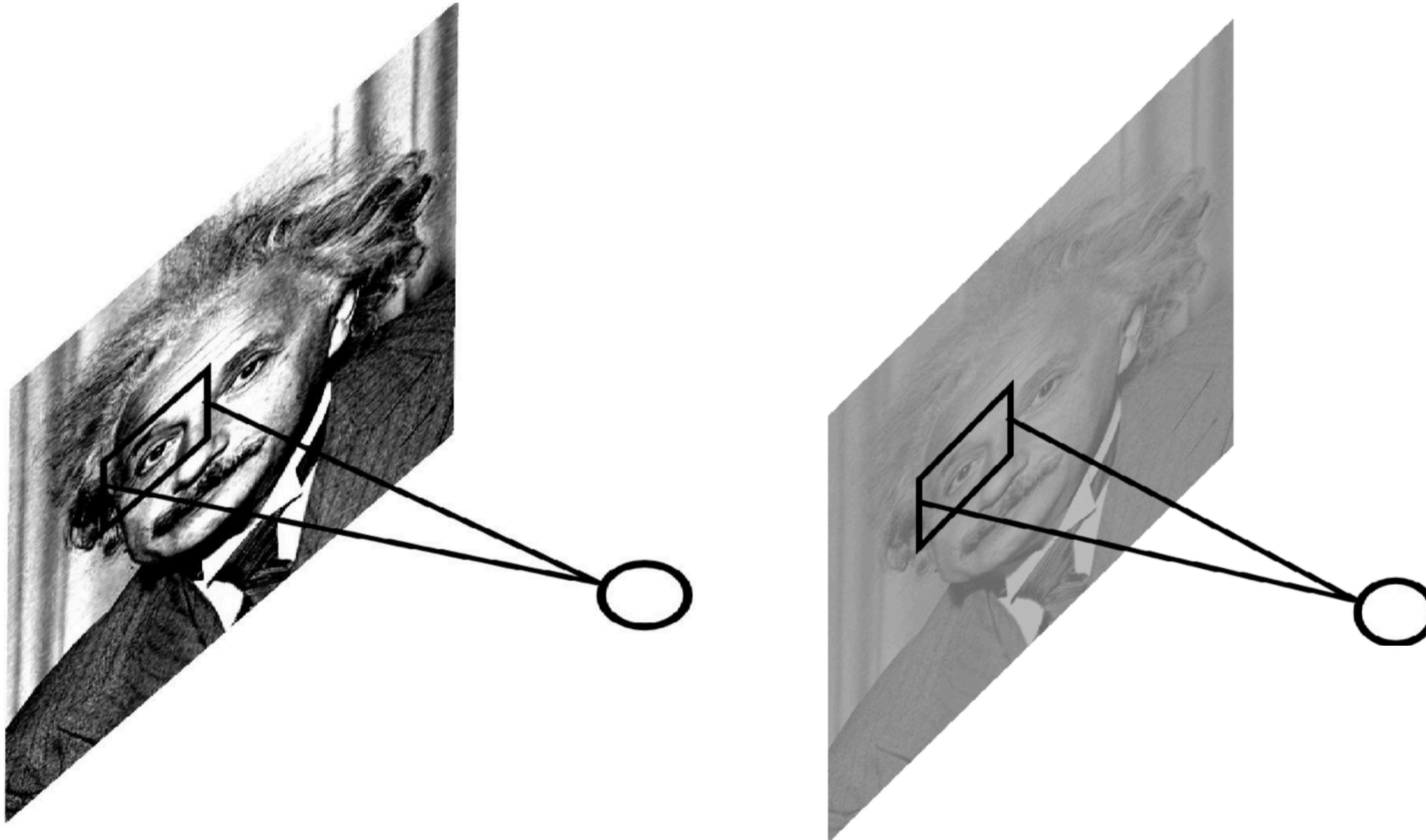
MAX POOL1: 96 11 x 11 filters applied at stride 4

Output: 27 x 27 x 96

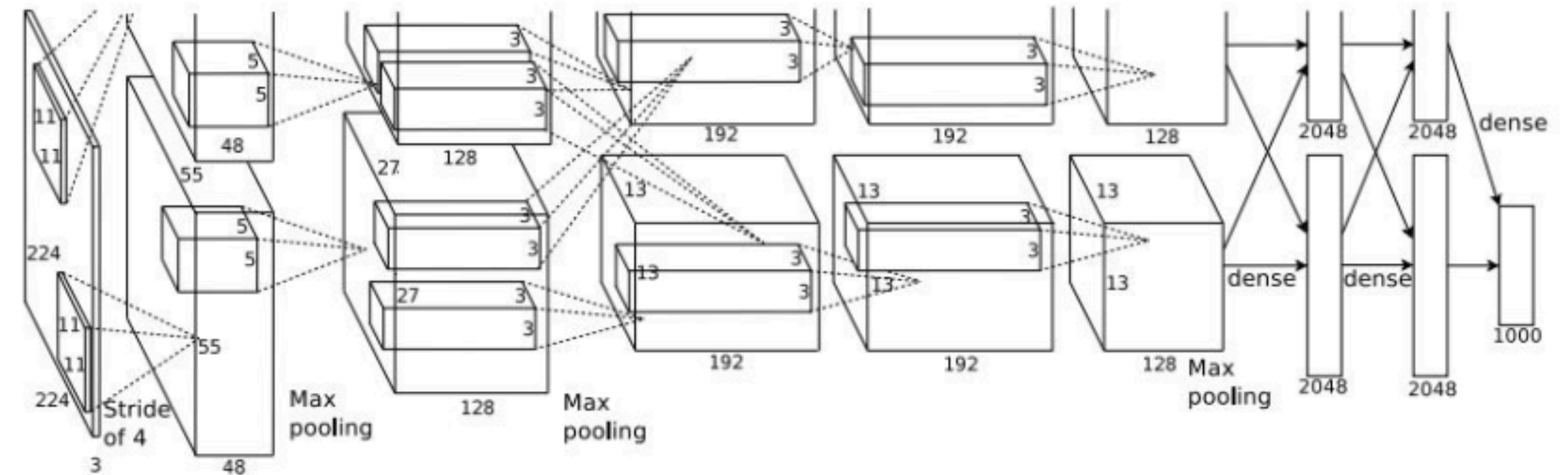
Parameters: 0

Local Contrast Normalization Layer

ensures response is the same in both case (details omitted, no longer popular)



AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

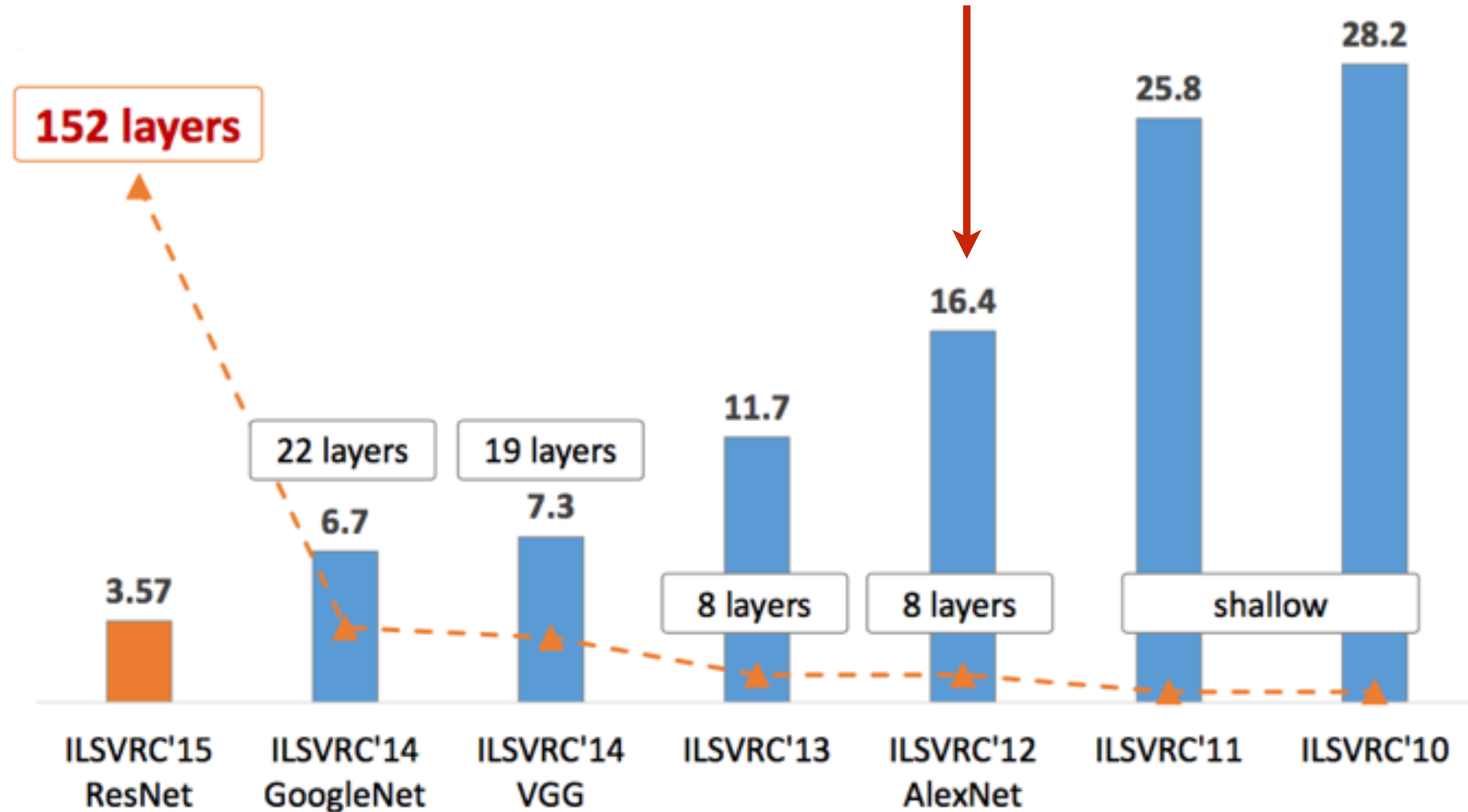
[1000] **FC8**: 1000 neurons (class scores)

[Krizhevsky et al., 2012]

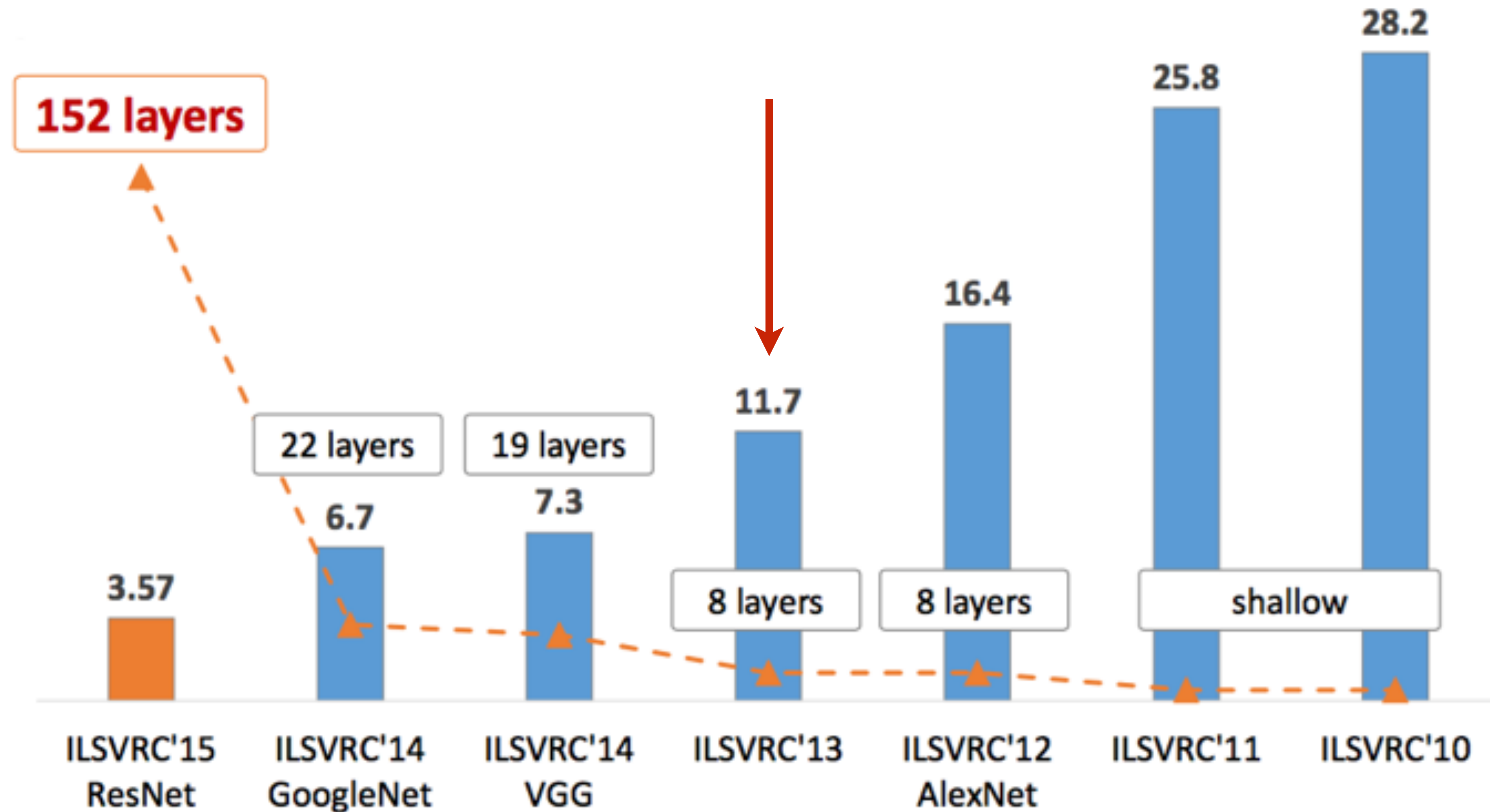
Details / Comments

- First use of ReLU
- Used contrast normalization layers
- Heavy data augmentation
- Dropout of 0.5
- Batch size of 128
- SGD Momentum (0.9)
- Learning rate (1e-2) reduced by 10 manually when validation accuracy plateaus
- L2 weight decay
- 7 CNN ensemble: 18.2% -> 15.4%

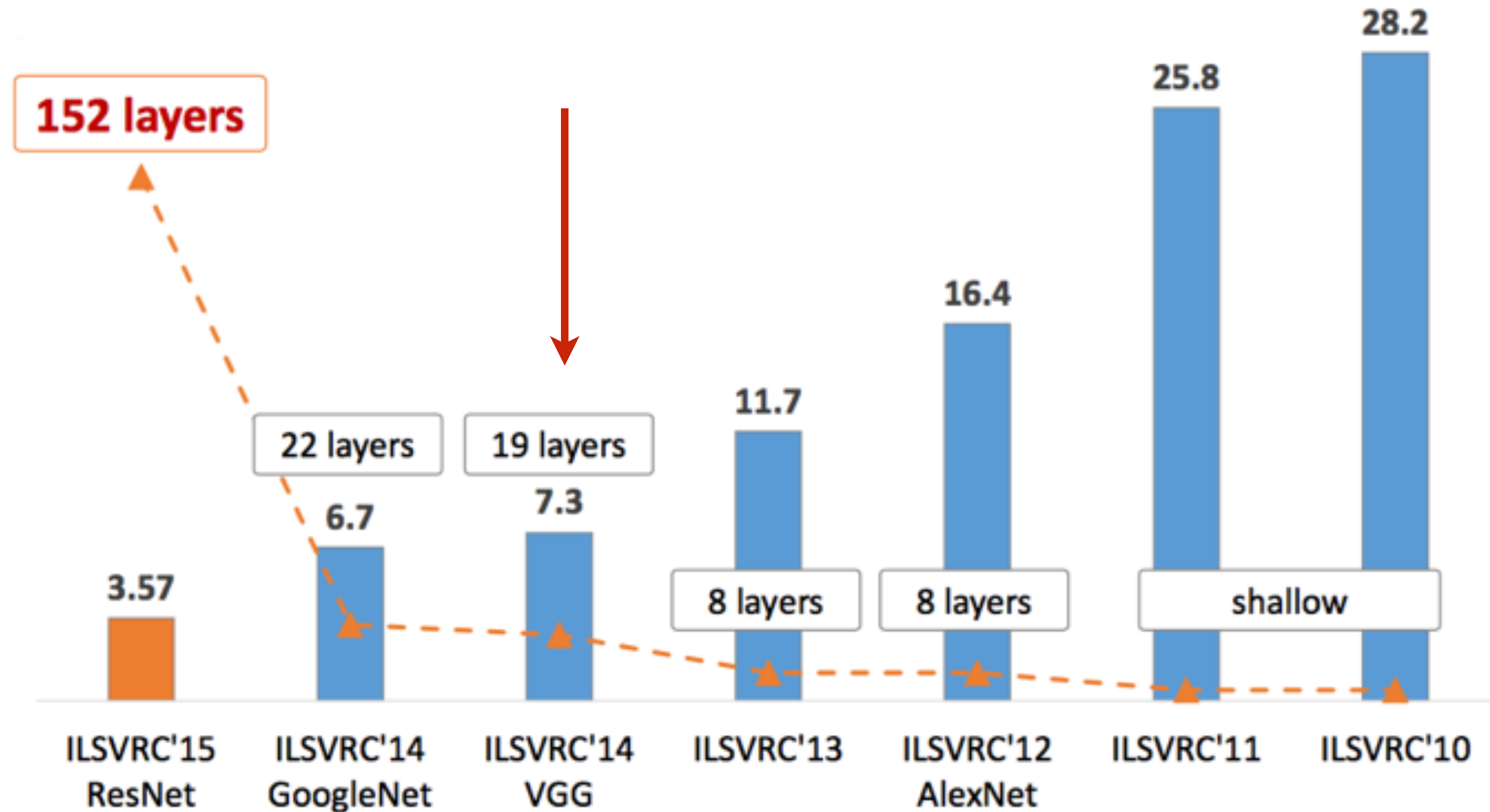
ILSVRC winner 2012



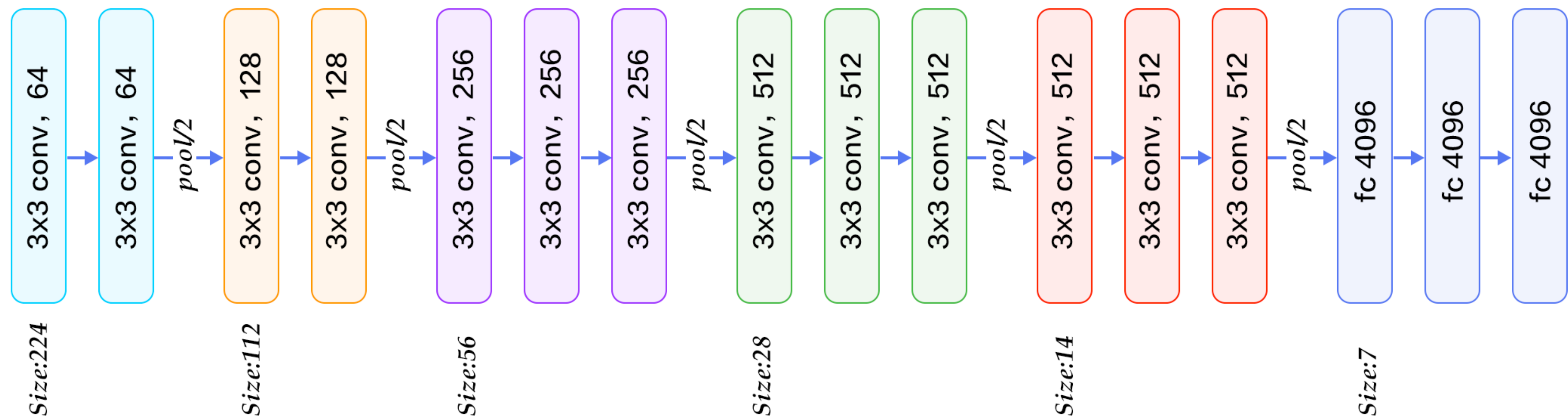
ILSVRC winner 2012



ILSVRC winner 2012

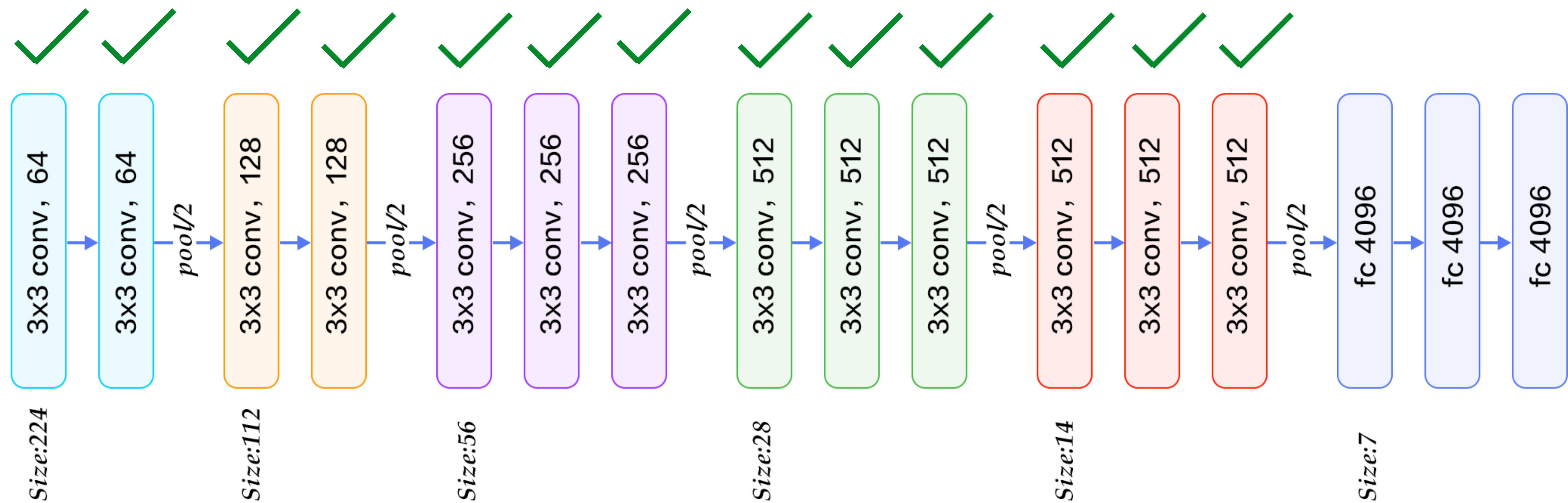


Convolutional Neural Networks



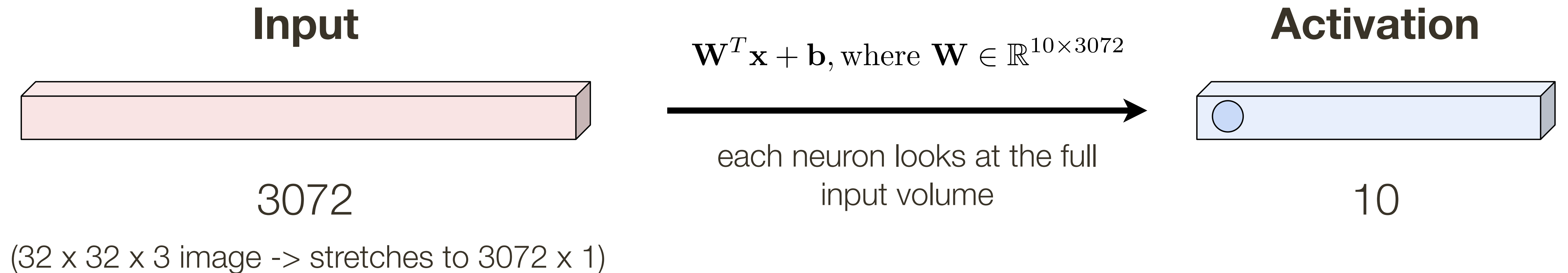
VGG-16 Network

Convolutional Neural Networks

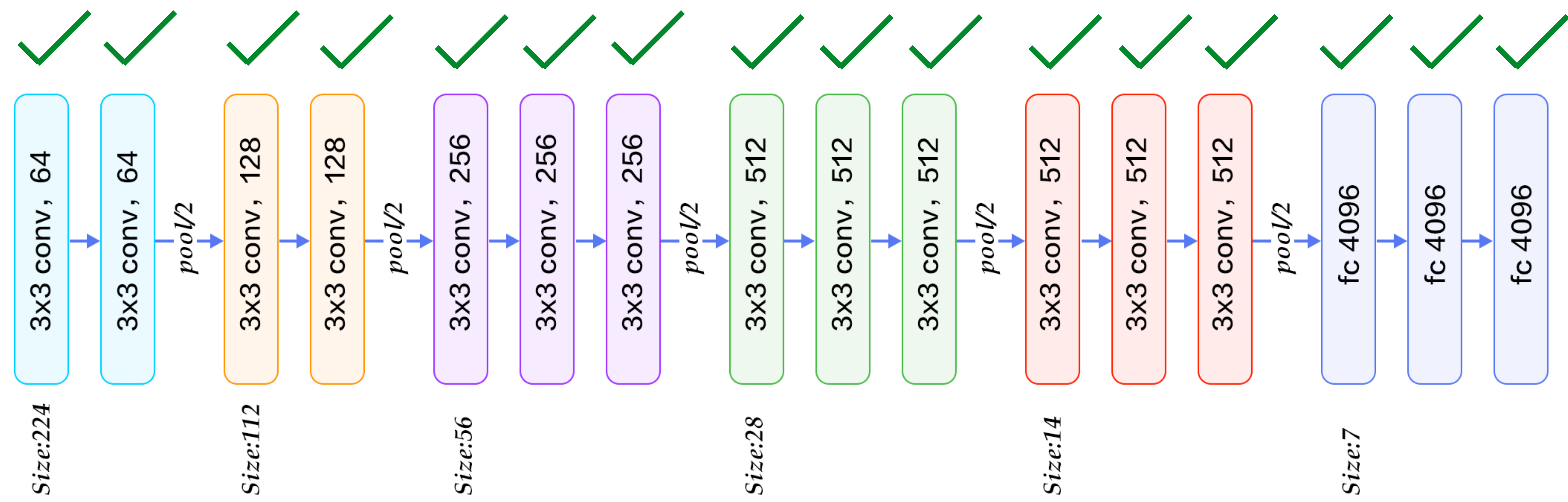


VGG-16 Network

CNNs: Reminder Fully Connected Layers

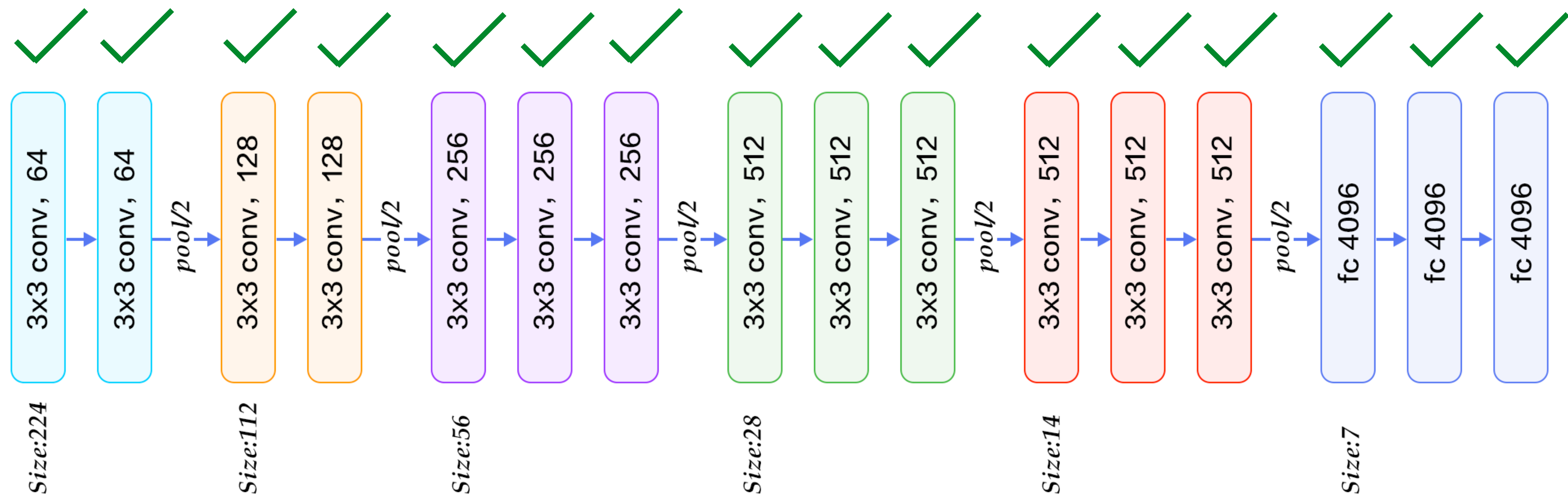


Convolutional Neural Networks



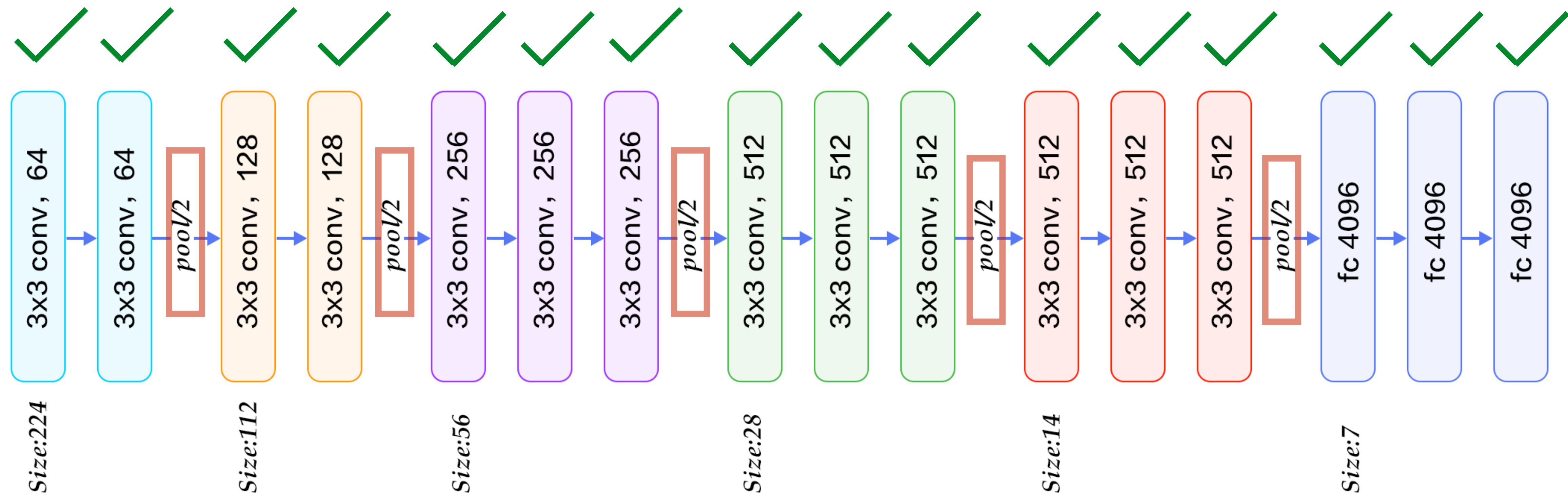
VGG-16 Network

Convolutional Neural Networks



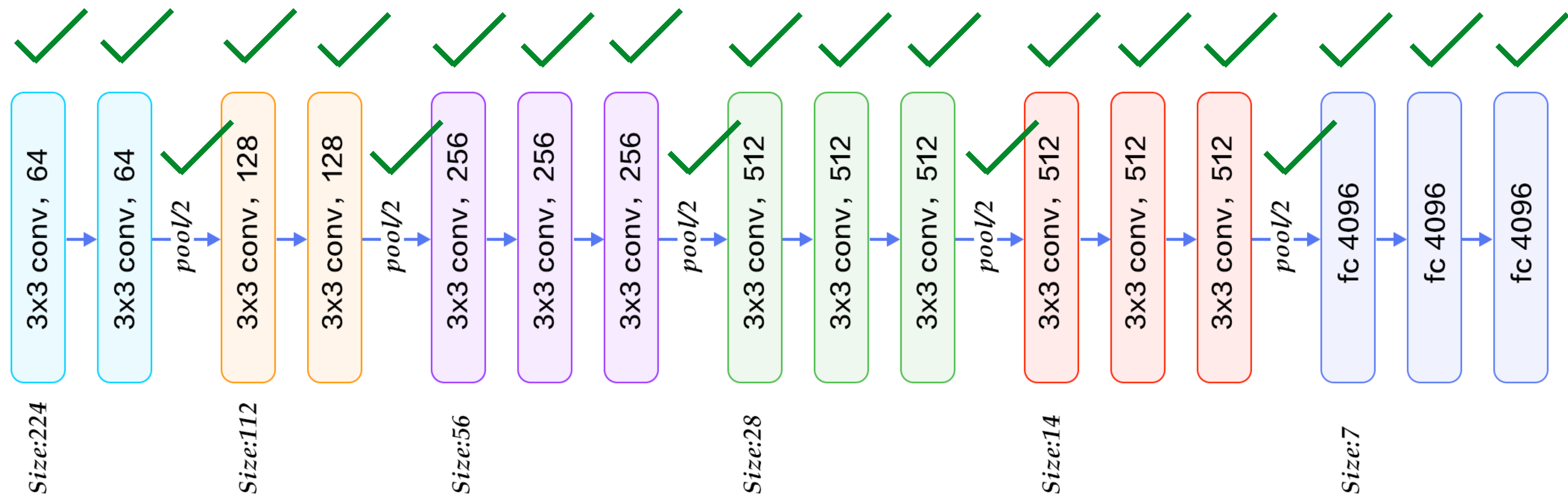
VGG-16 Network

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

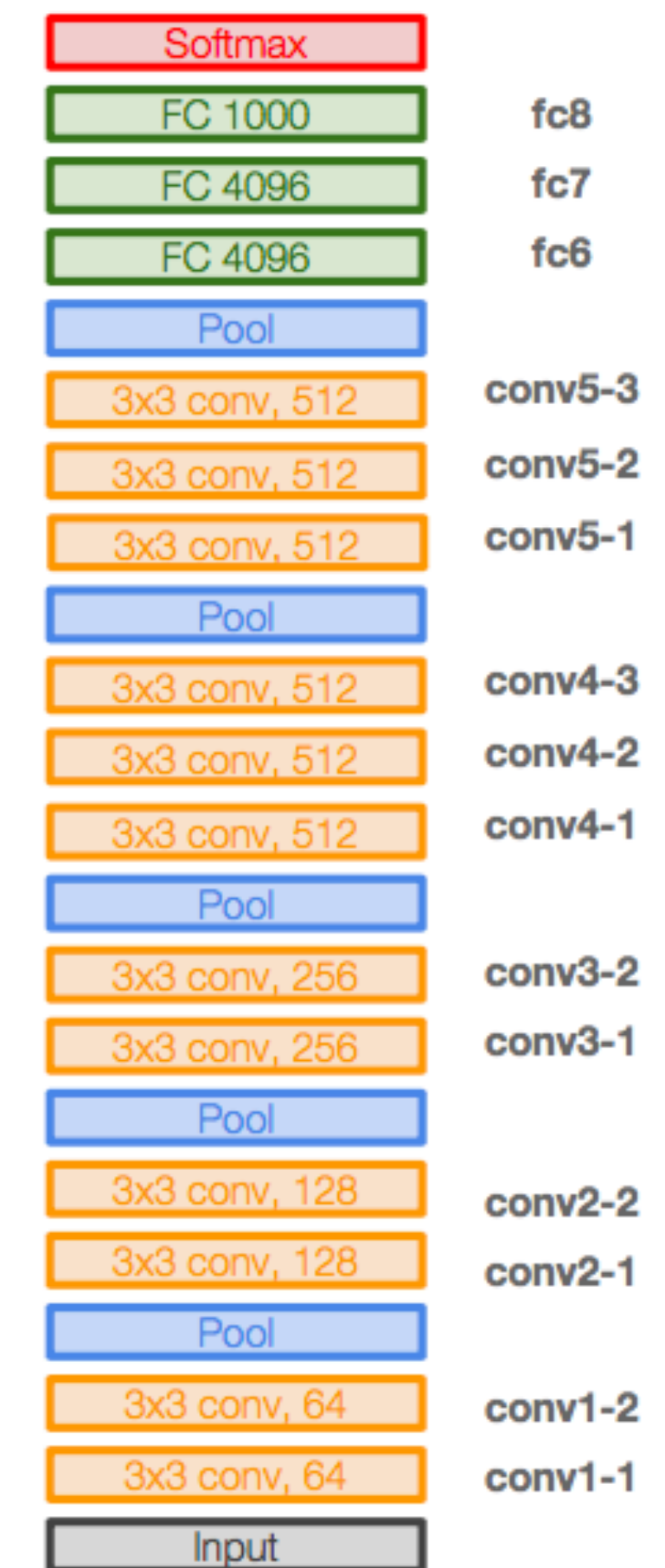
FC: [1x1x4096] memory: 4096 params: $7*7*512*4096 = 102,760,448$

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

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

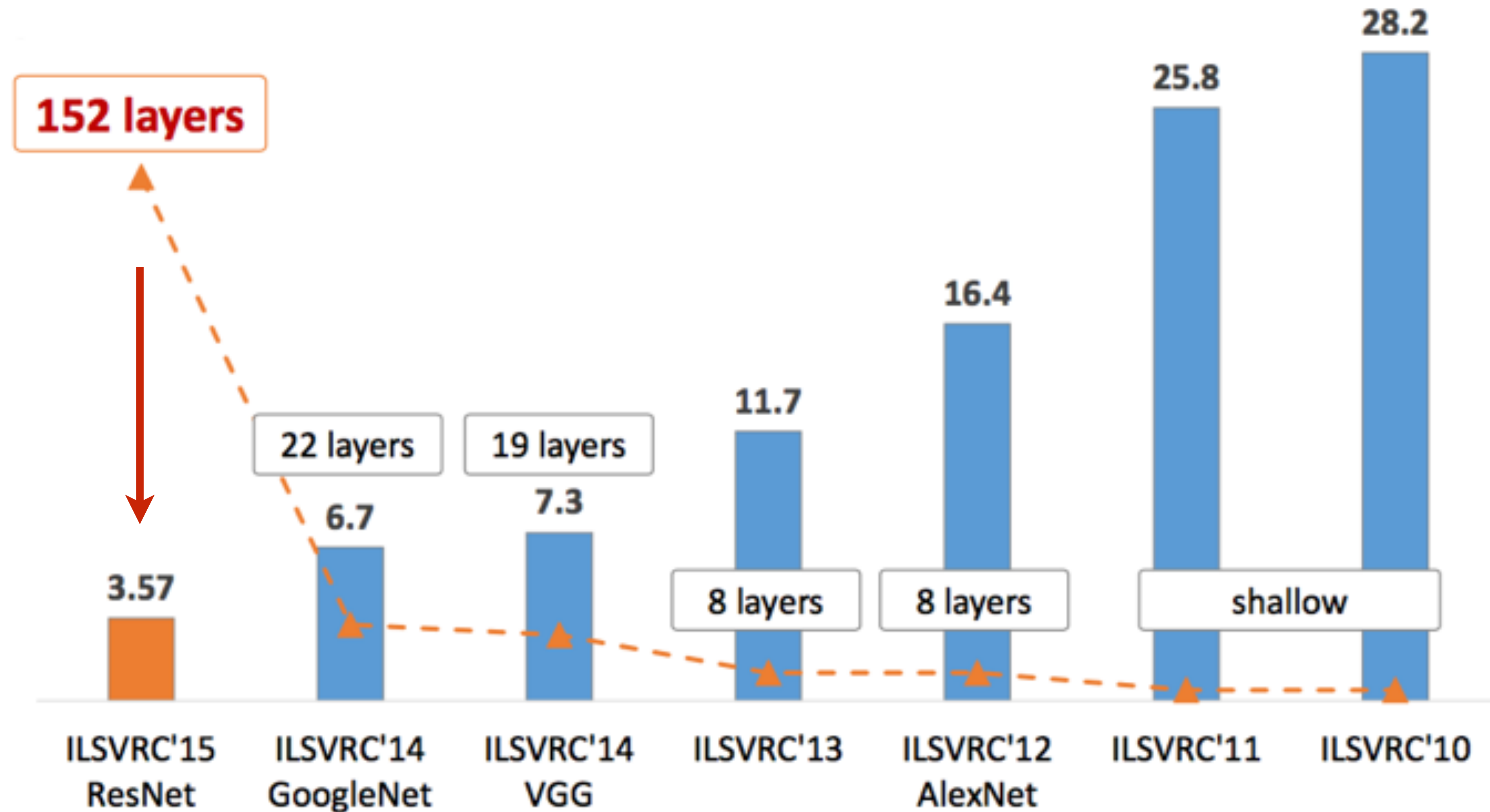
TOTAL memory: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB}$ / image (only forward! $\sim*2$ for bwd)

TOTAL params: 138M parameters



VGG16

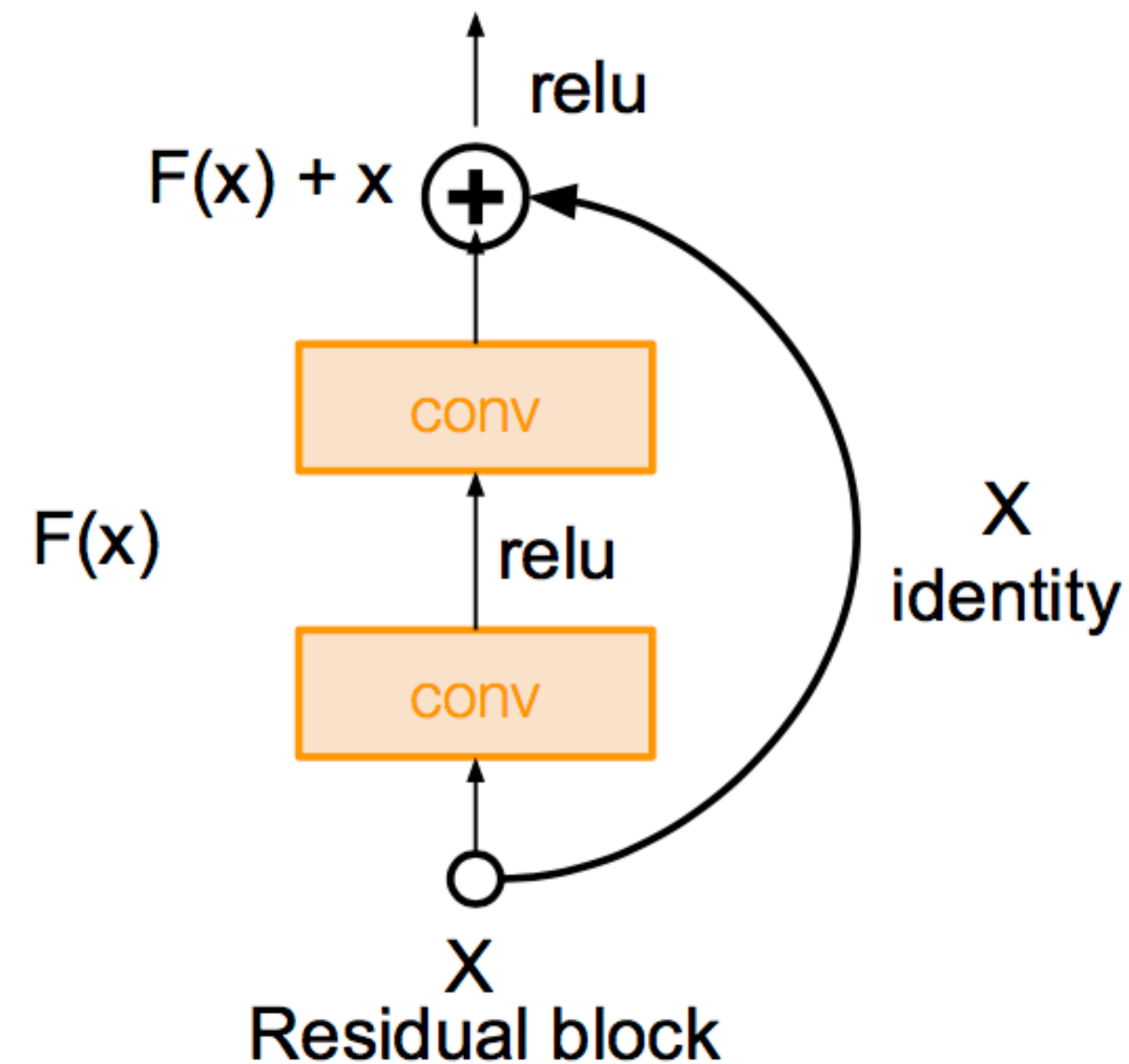
ILSVRC winner 2012



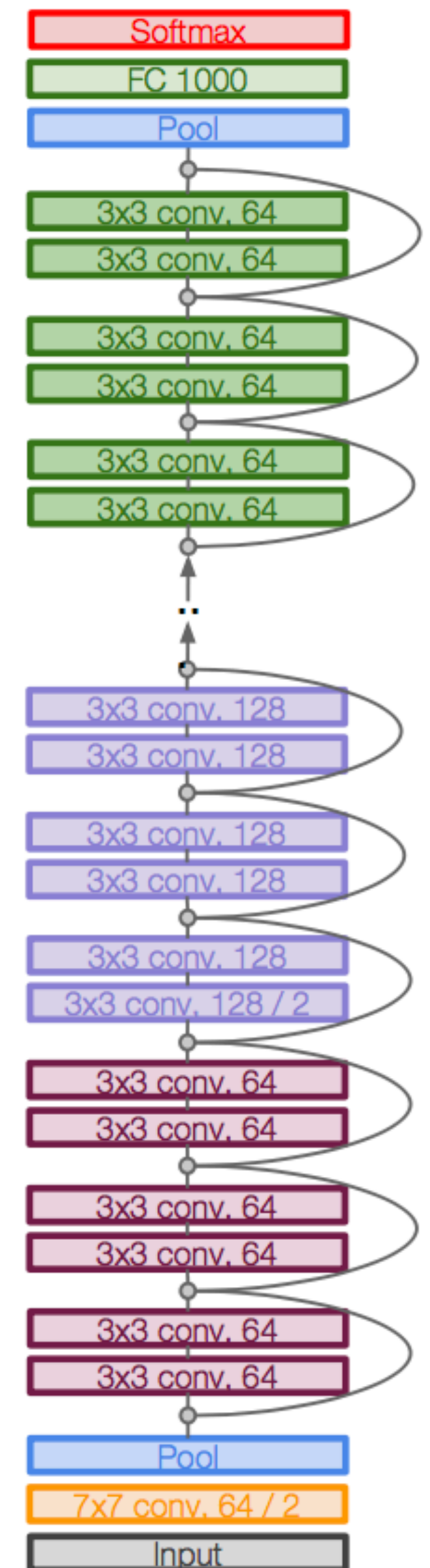
ResNet

even deeper — **152 layers!**

using residual connections



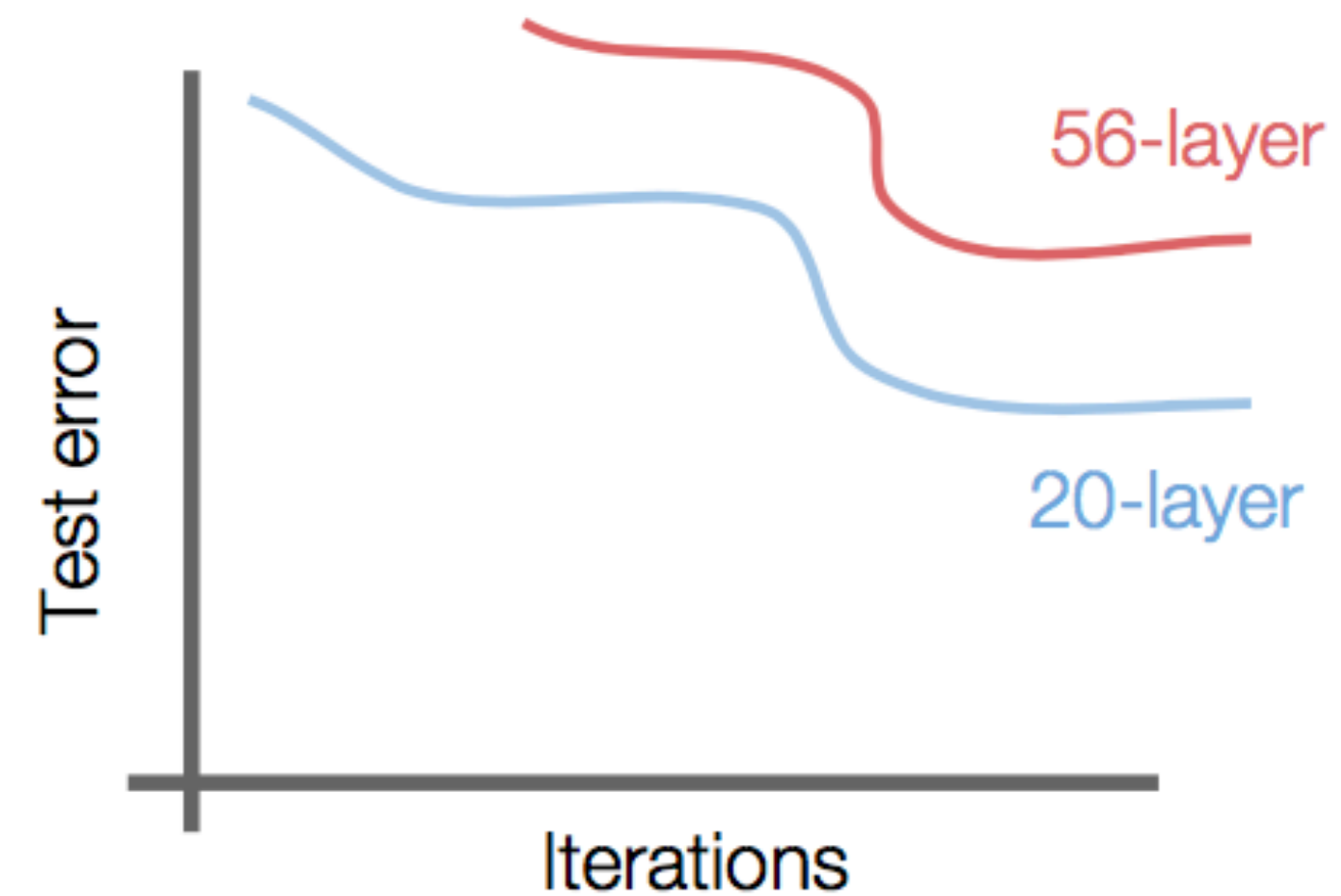
[He et al., 2015]



ResNet: Motivation

[He et al., 2015]

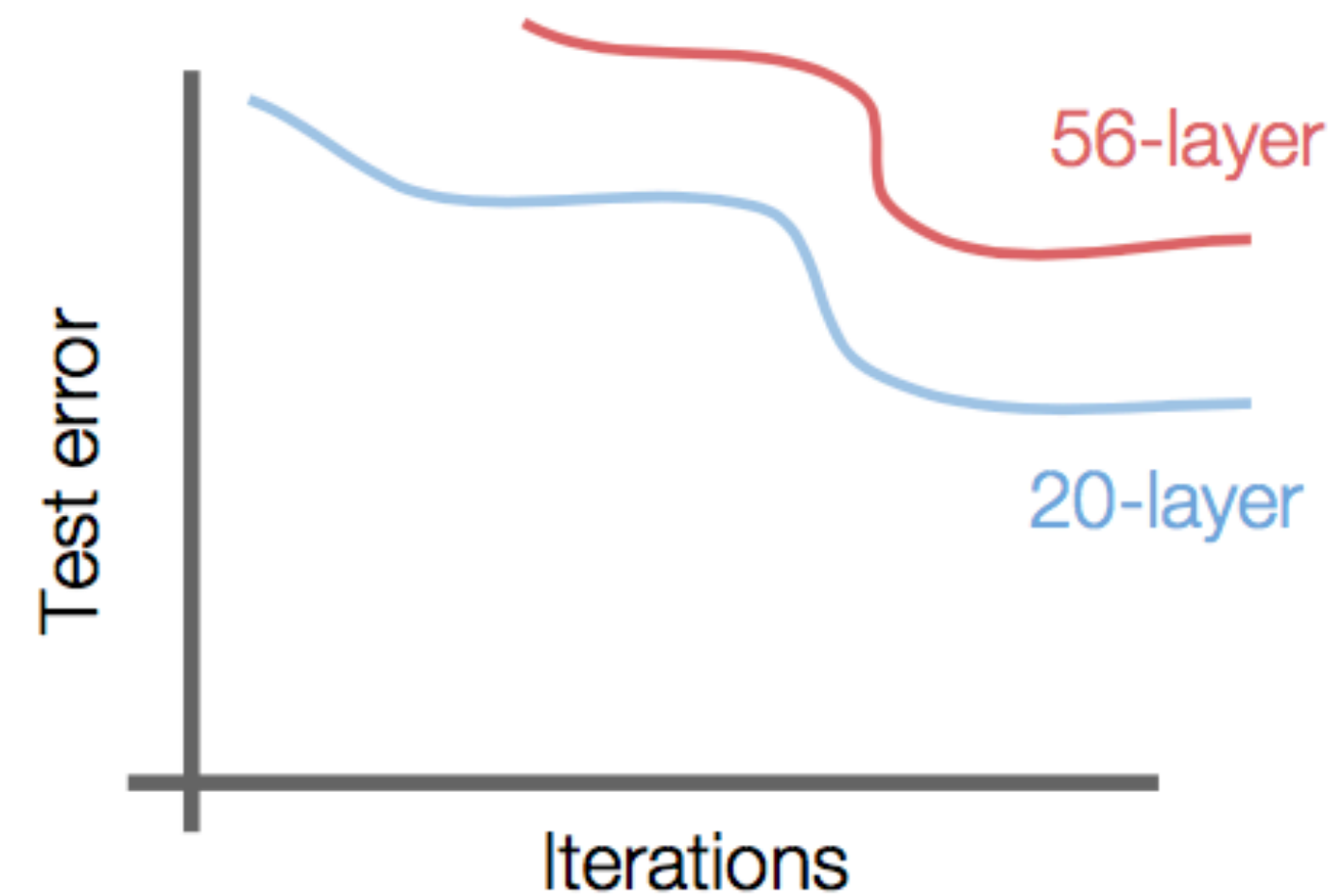
What happens when we continue to stacking deeper layers on a “plain” CNN



ResNet: Motivation

[He et al., 2015]

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

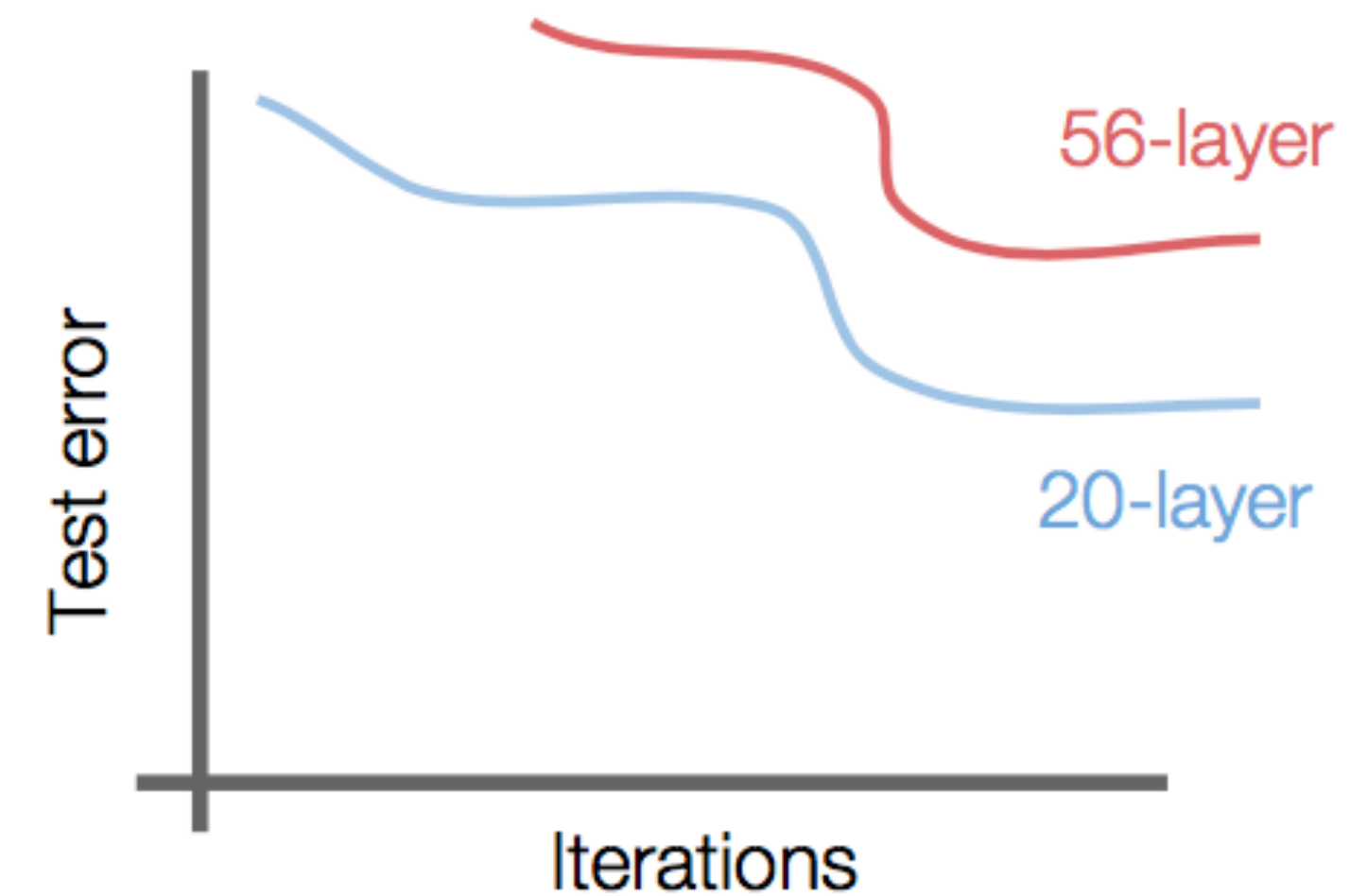
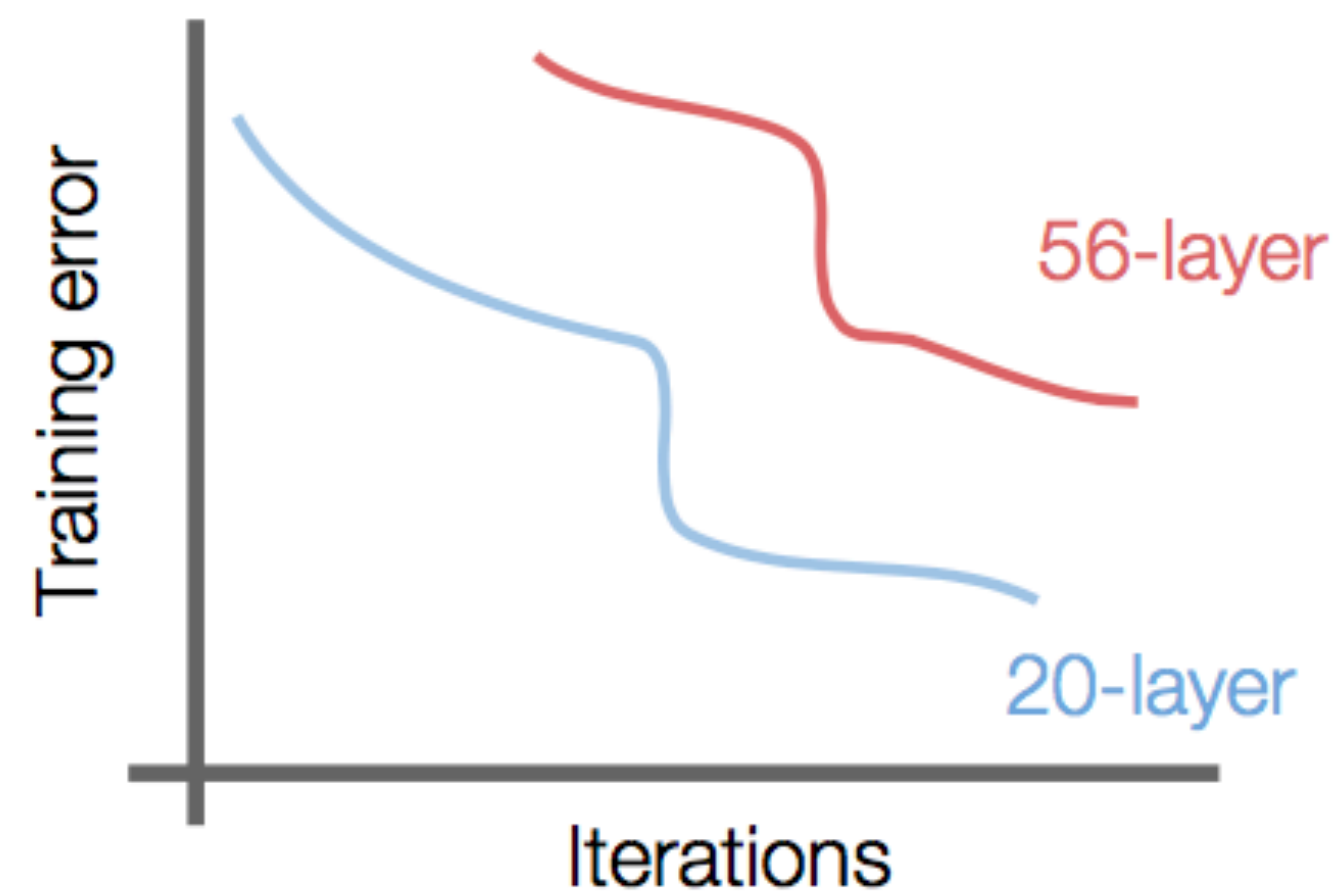


Whats the **problem**?

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

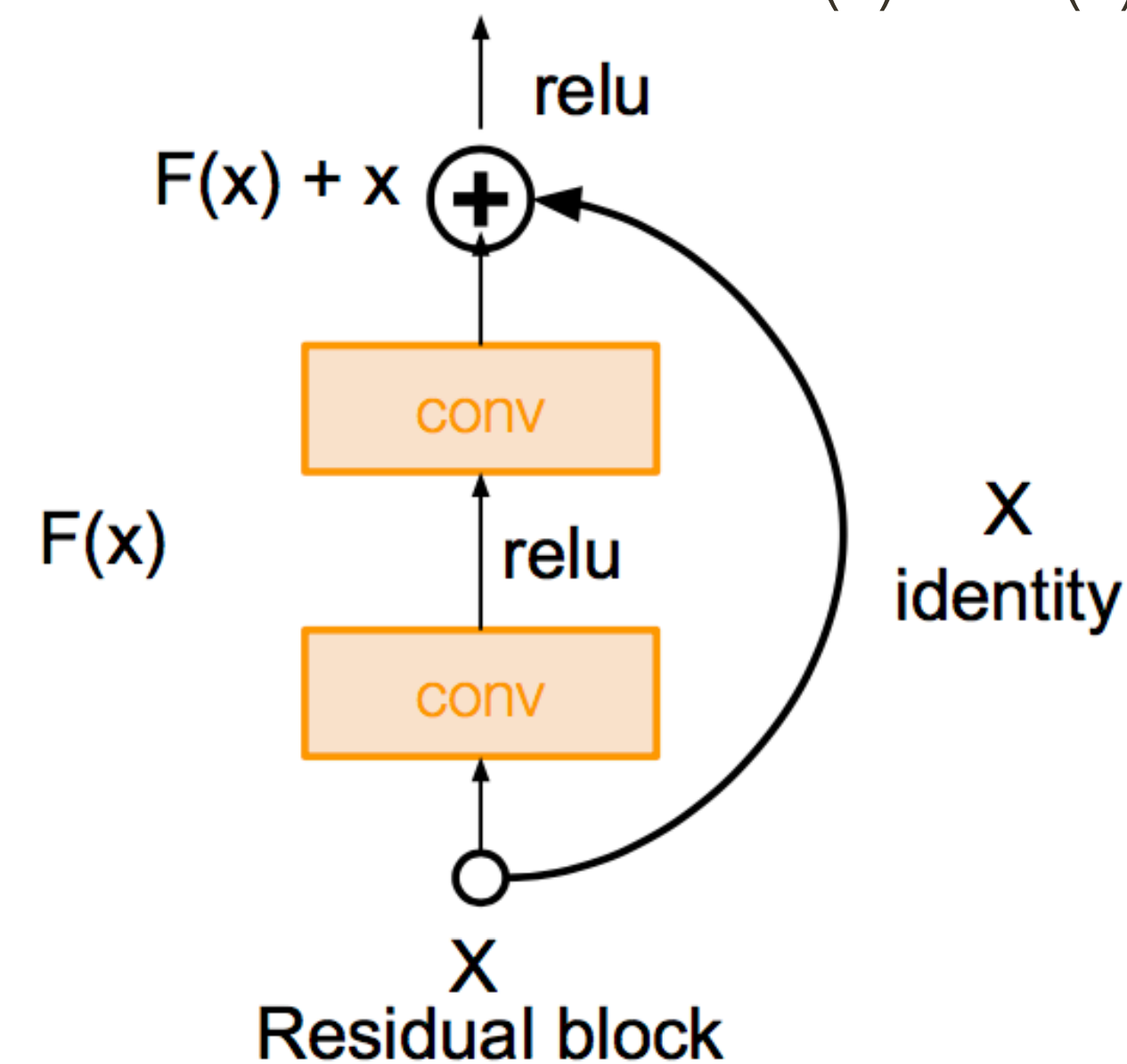
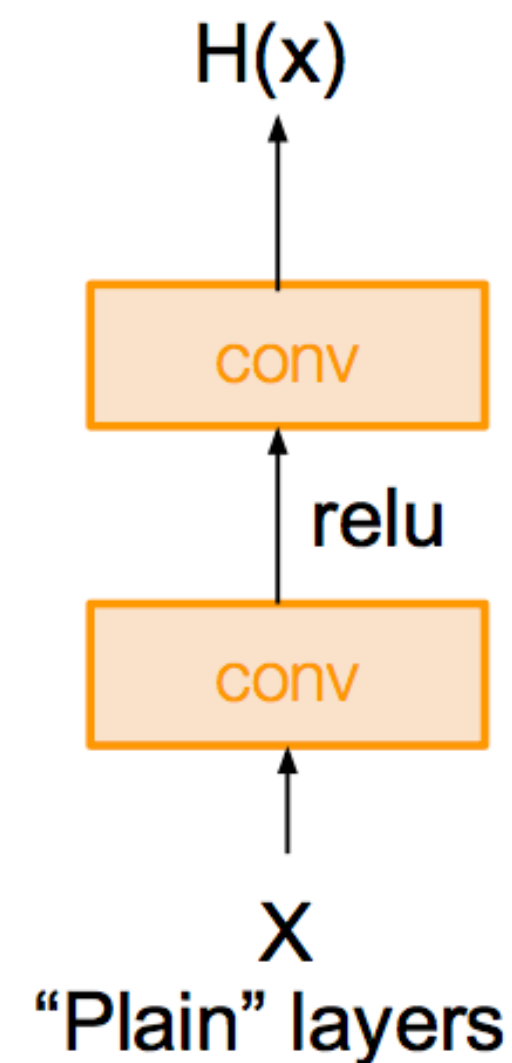
ResNet

[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

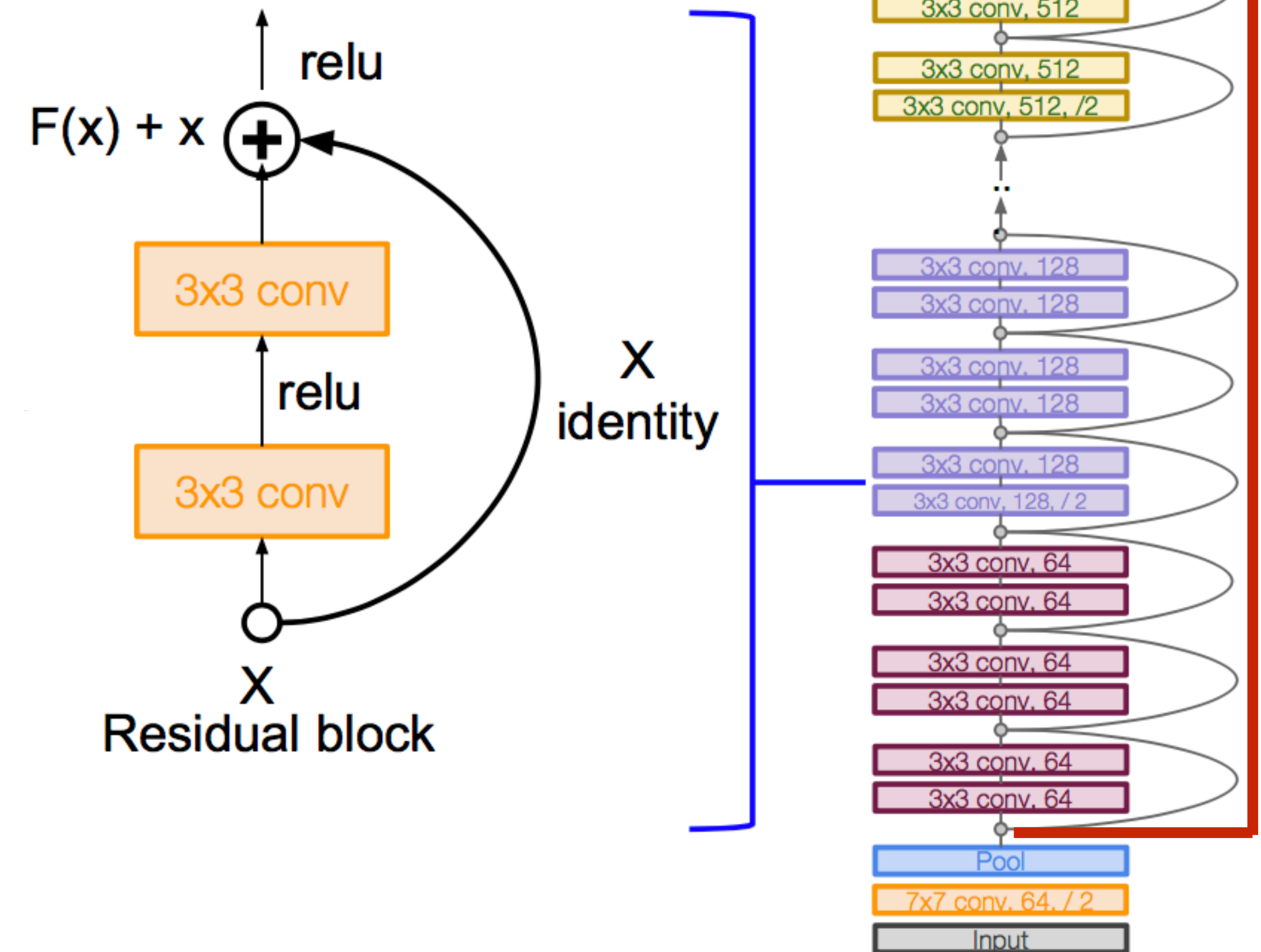


ResNet

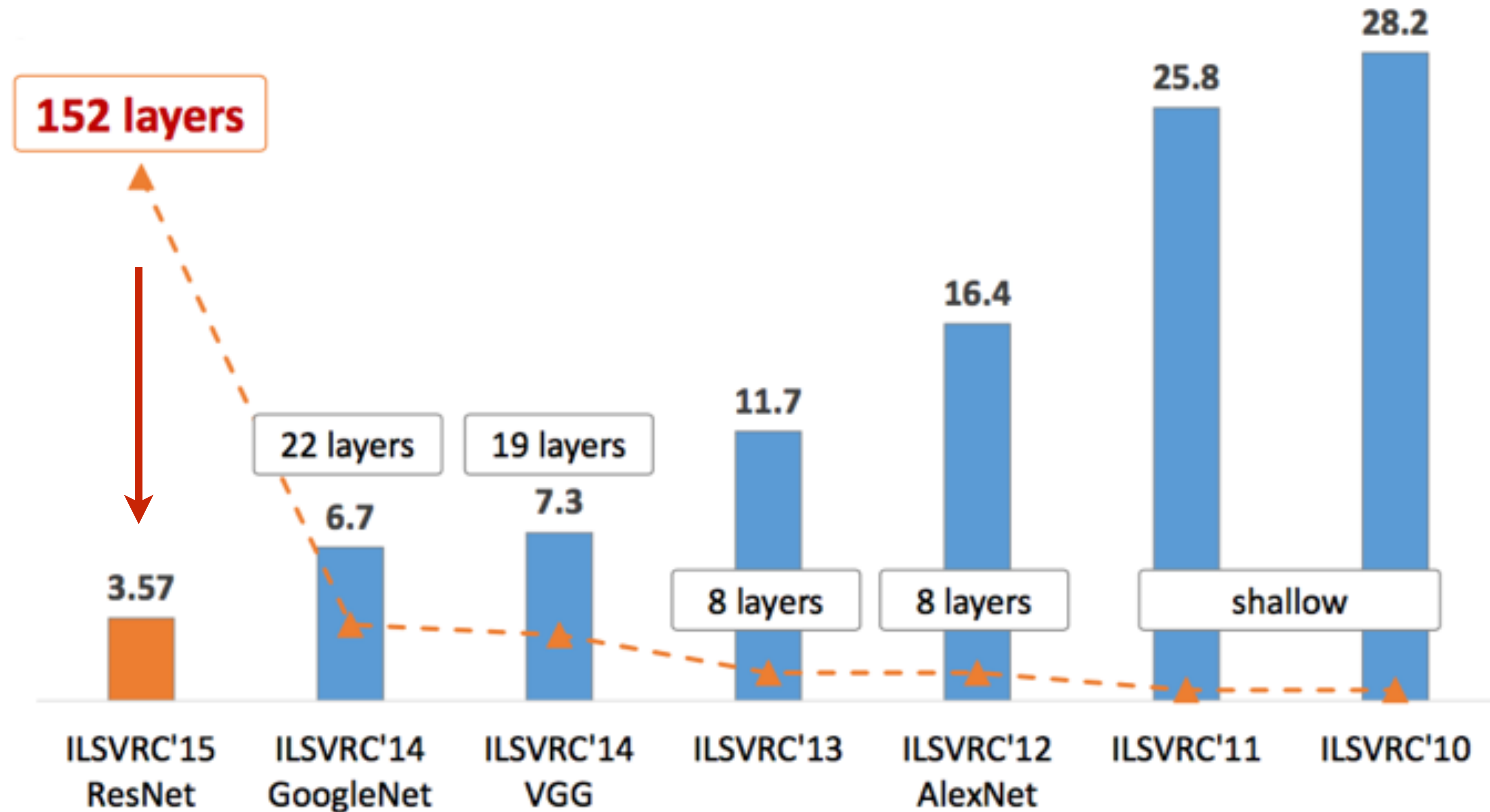
[He et al., 2015]

Full details

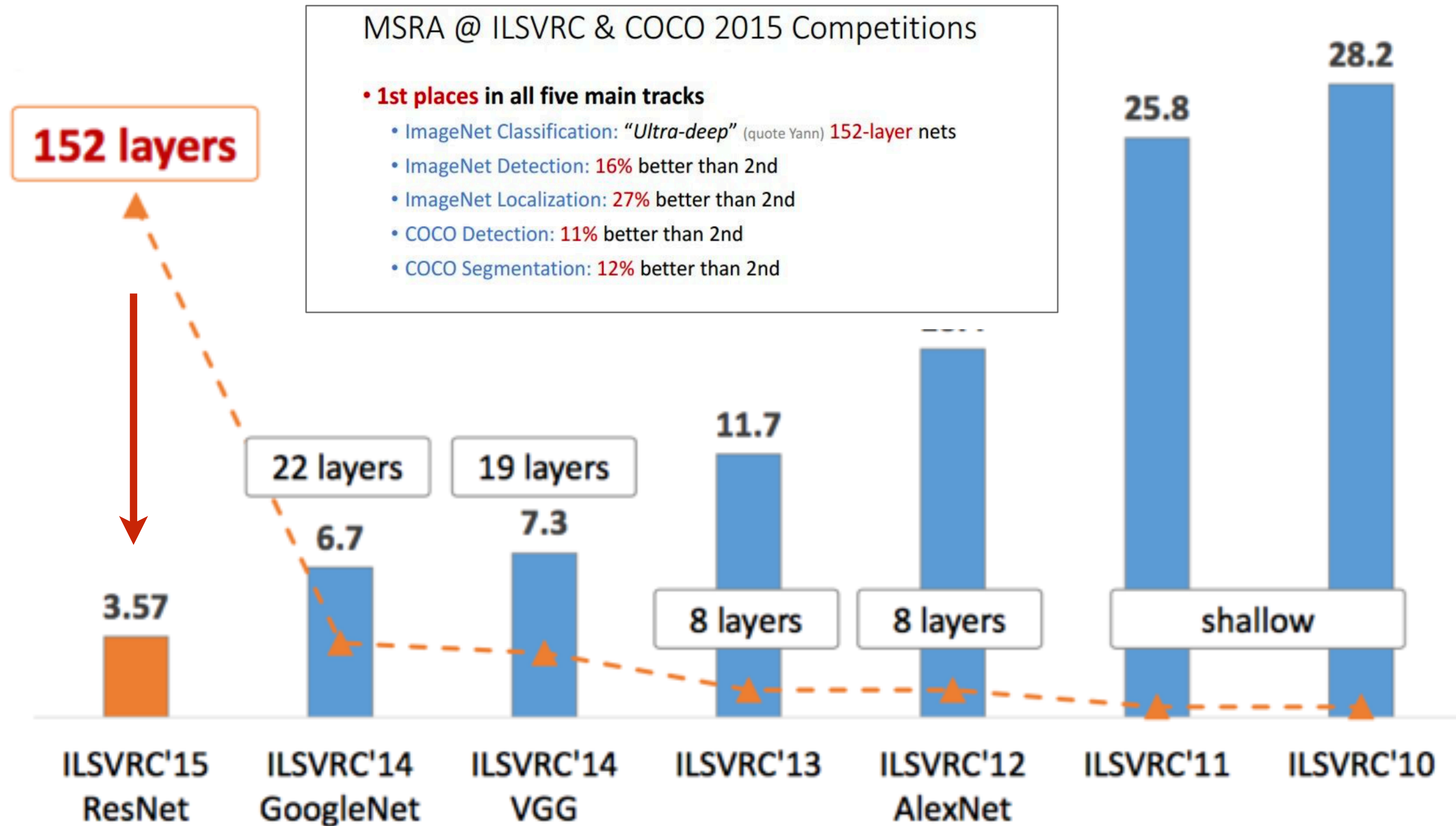
- Stacked **residual blocks**
- Every residual block consists of **two 3x3 filters**
- Periodically double # of filters and downsample spatially using stride of 2
- Additional convolutional layer in the beginning
- **No FC layers** at the end (only FC to output 1000 classes)



ILSVRC winner 2012



ILSVRC winner 2012



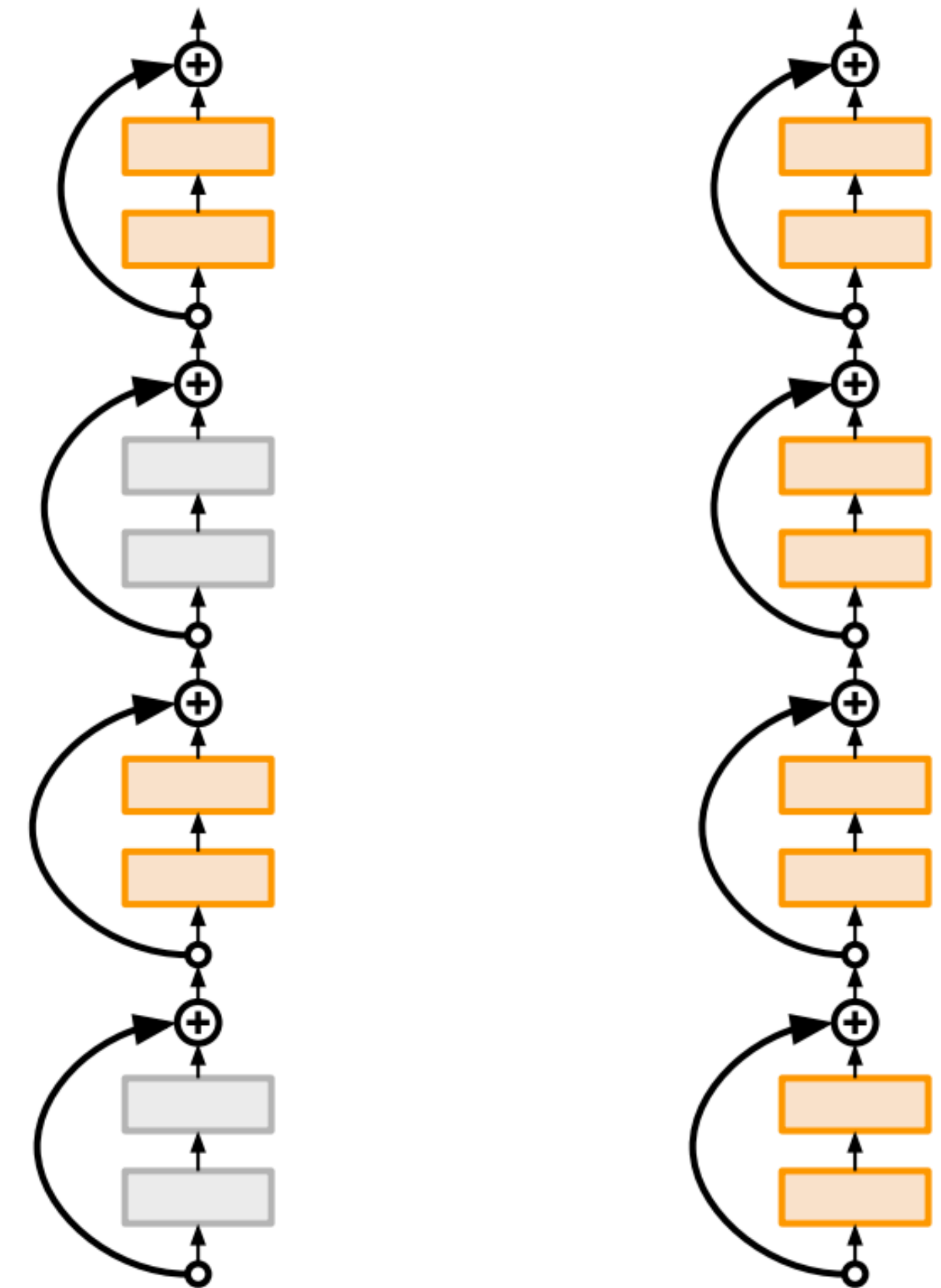
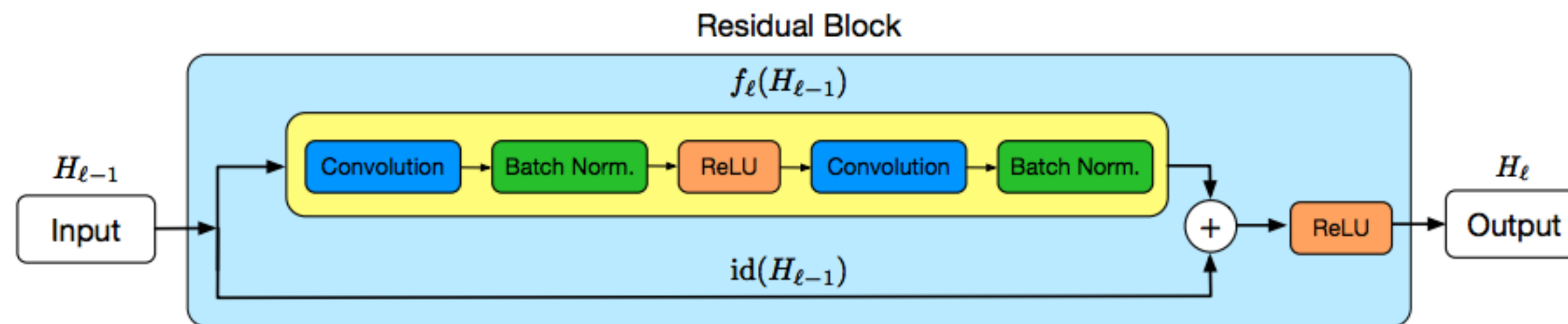
Regularization: Stochastic Depth

[Huang et al., ECCV 2016]

Effectively “dropout” but for layers

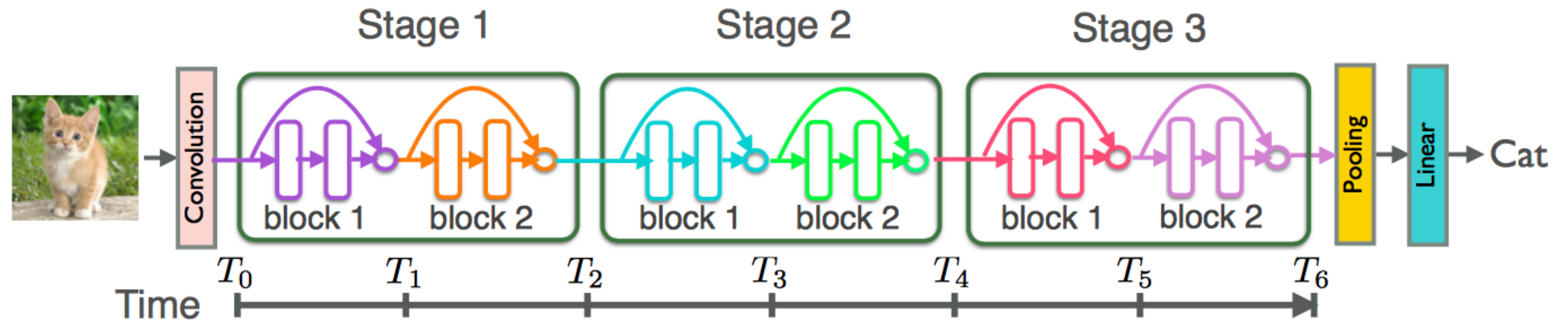
Stochastically with some probability **turn off some layer** (for each batch)

Effectively trains a collection of neural networks



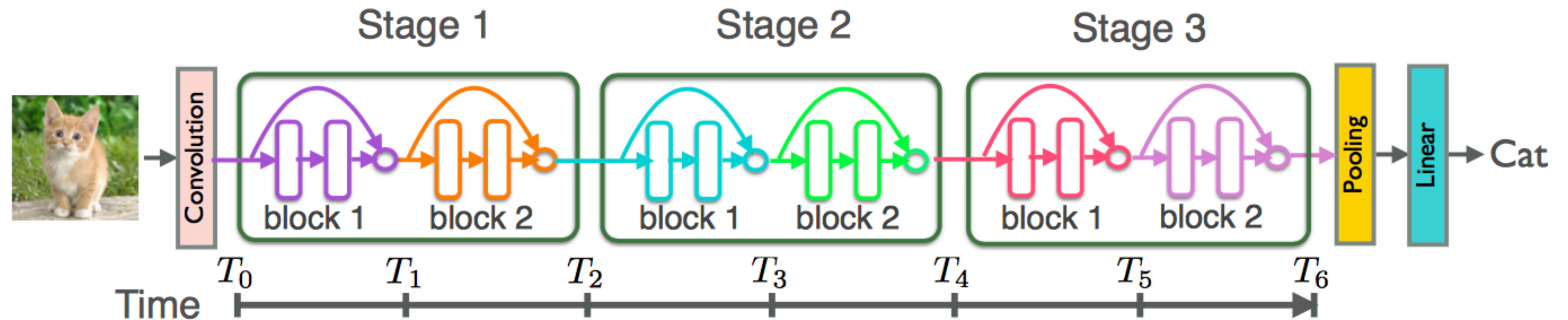
ResNet: A little theory

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

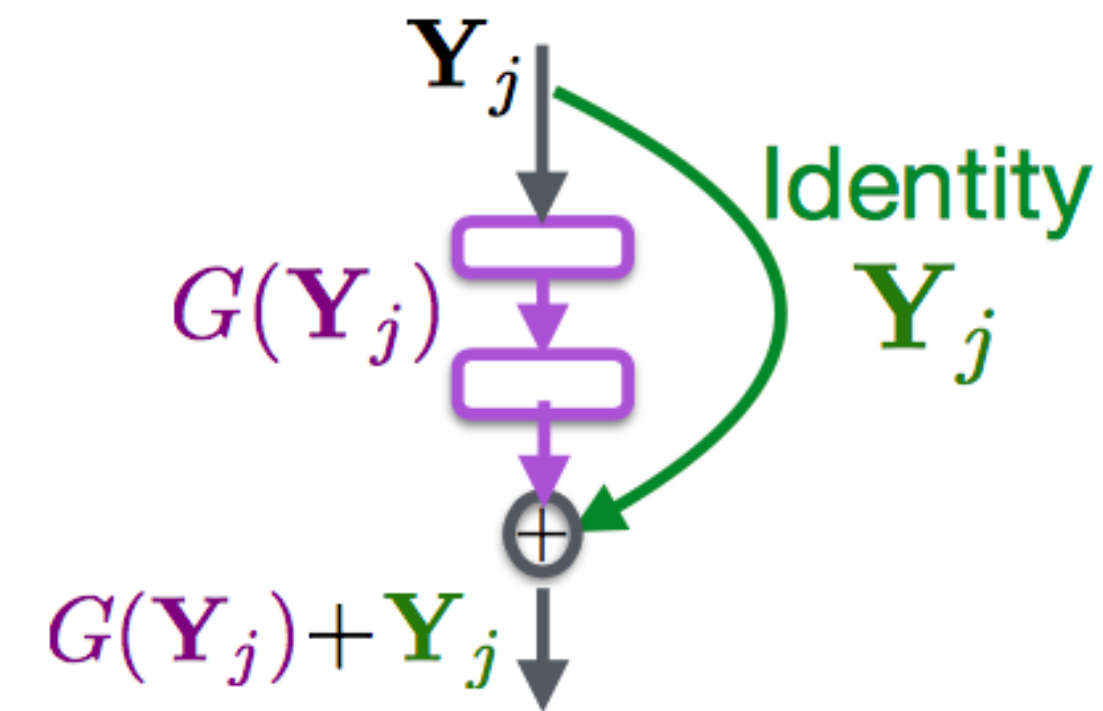


ResNet: A little theory

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

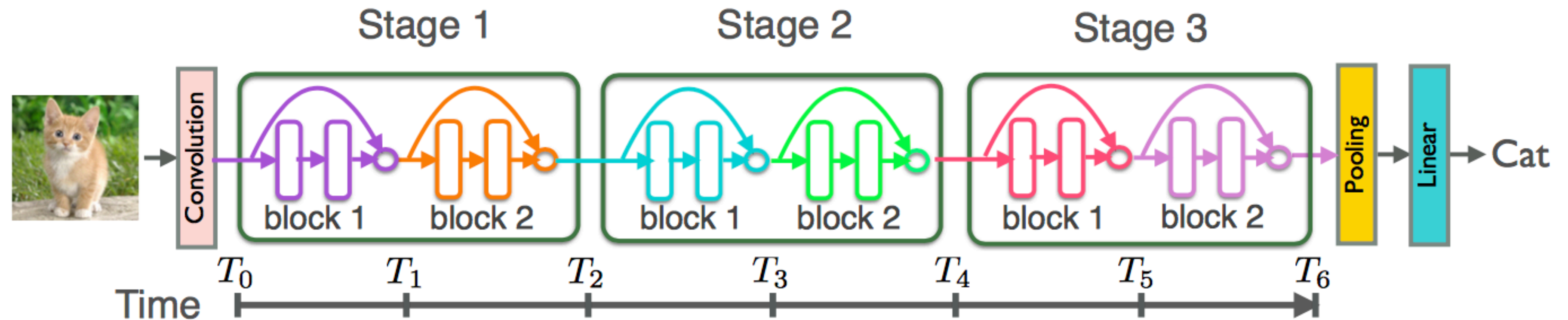


$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + G(\mathbf{Y}_j, \theta_j)$$



ResNet: A little theory

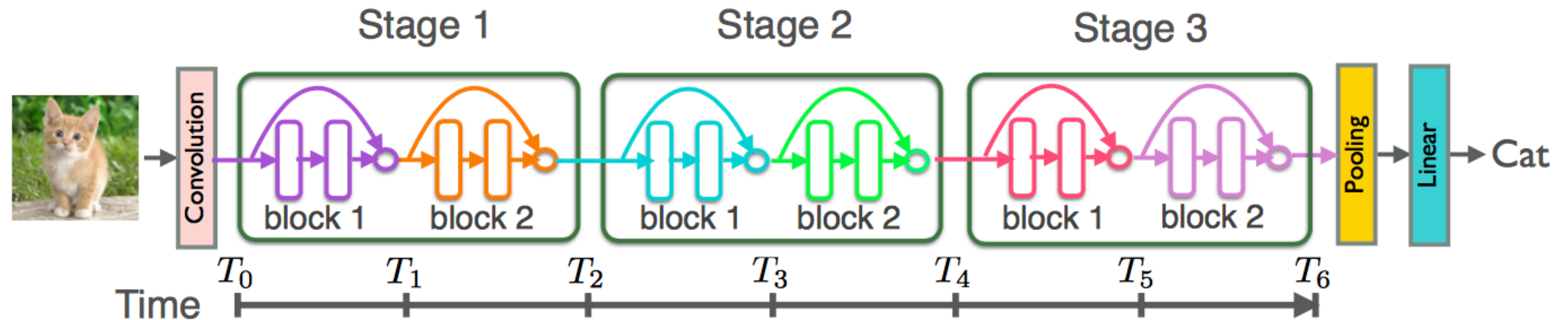
One can view a sequence of outputs from residual layers as a **Dynamical System**



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**



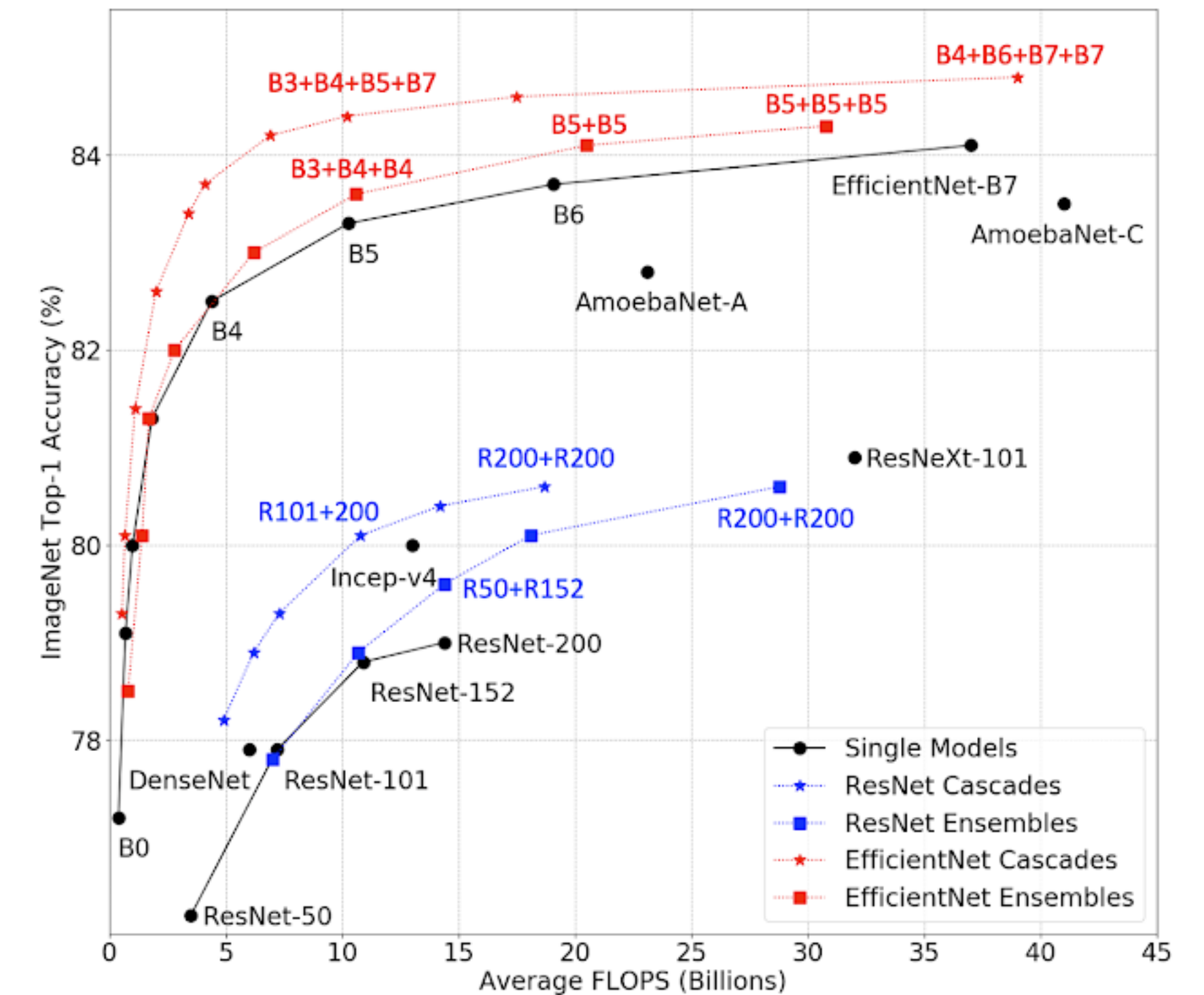
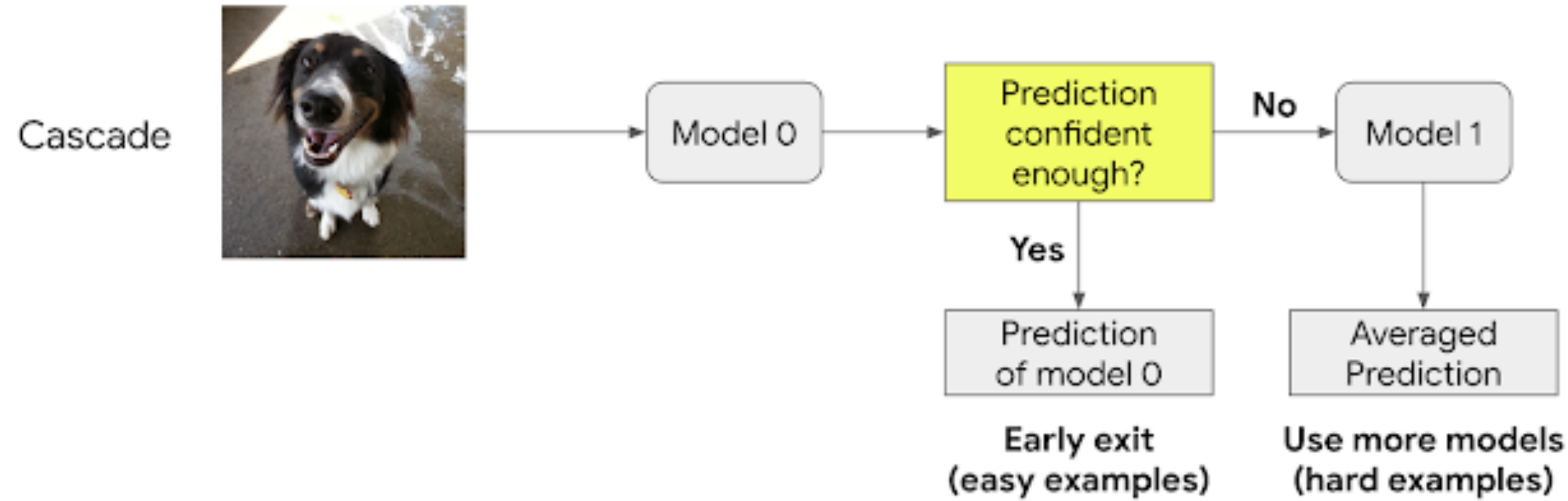
What happens if you take more layers and take smaller steps?

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

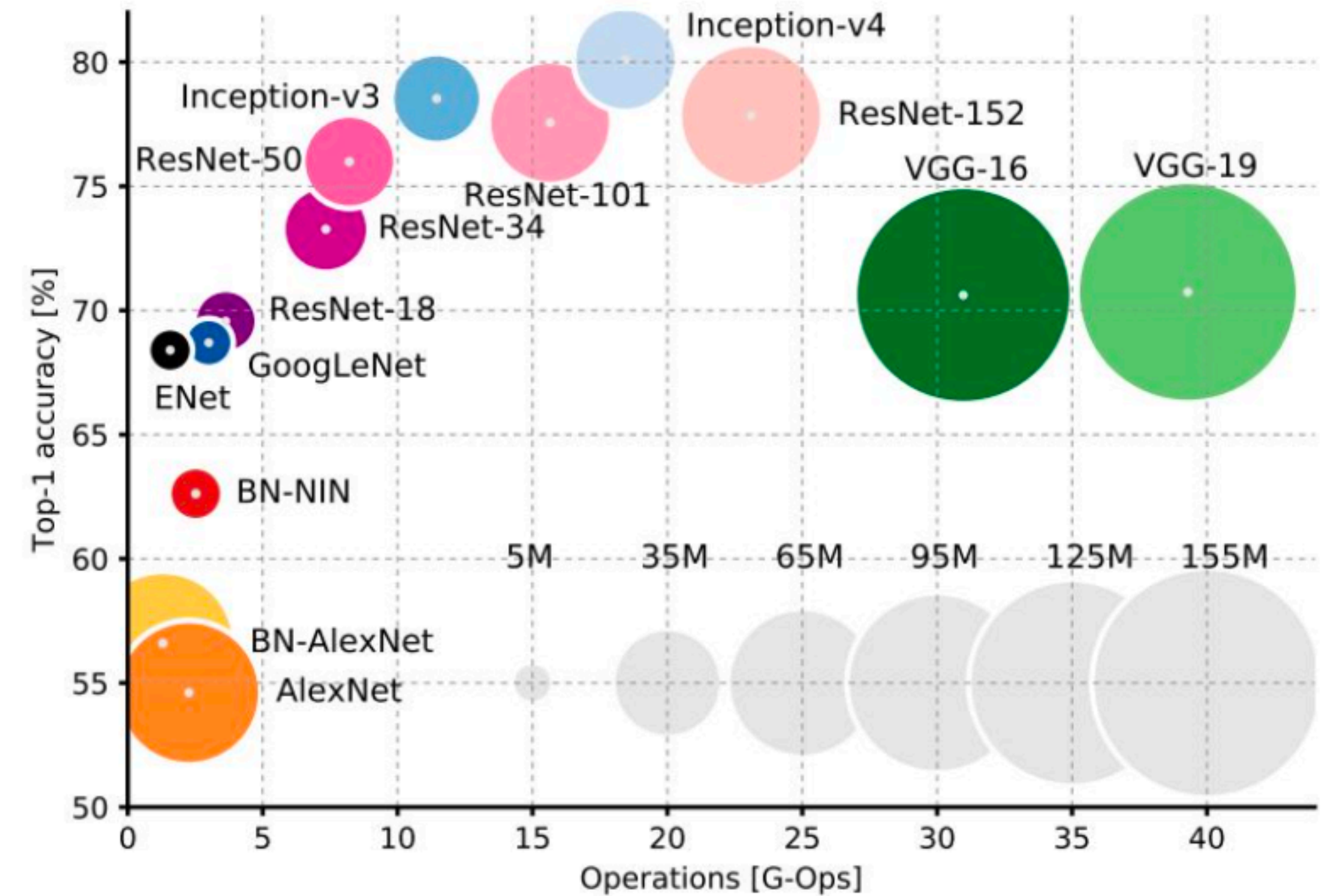
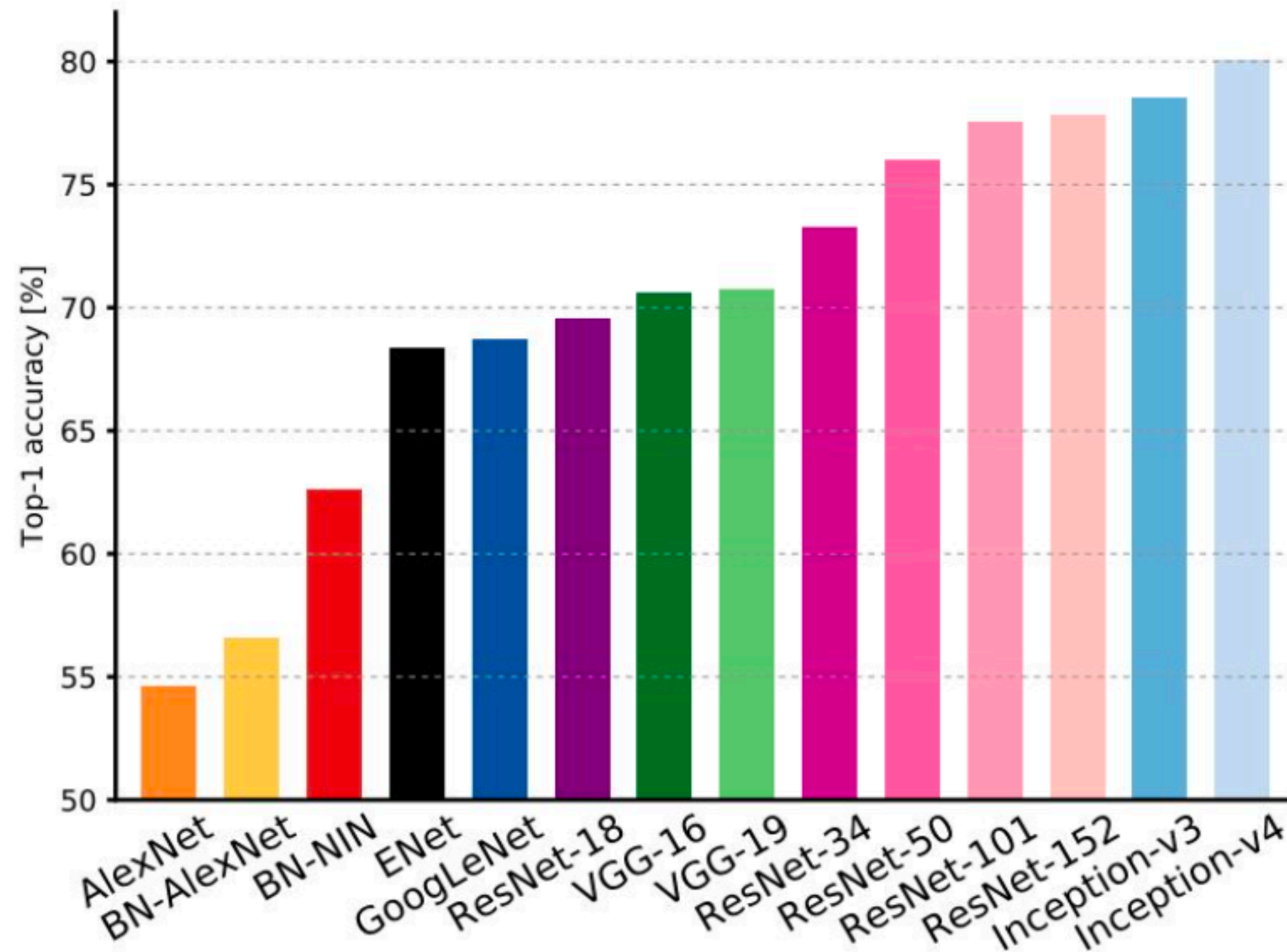
[Chen et al., NIPS 2018 **best paper**]

An Aside: Neural Network Cascades

[Wang et al., ICLR 2022]



Comparing Complexity



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

Computer **Vision Problems** (no language for now)

Categorization

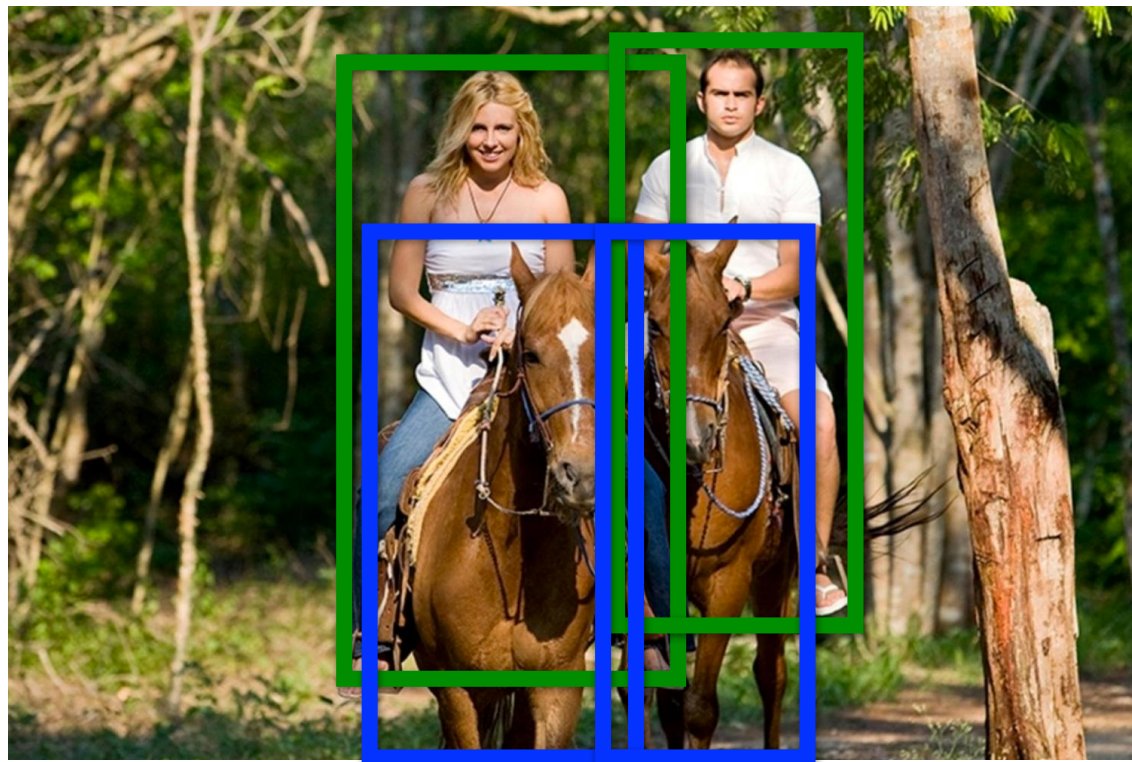


Multi-**class**: Horse
Church
Toothbrush
Person

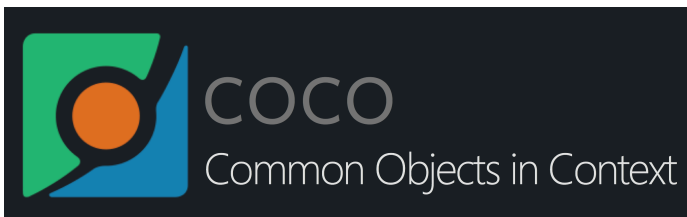


Multi-**label**: **Horse**
Church
Toothbrush
Person

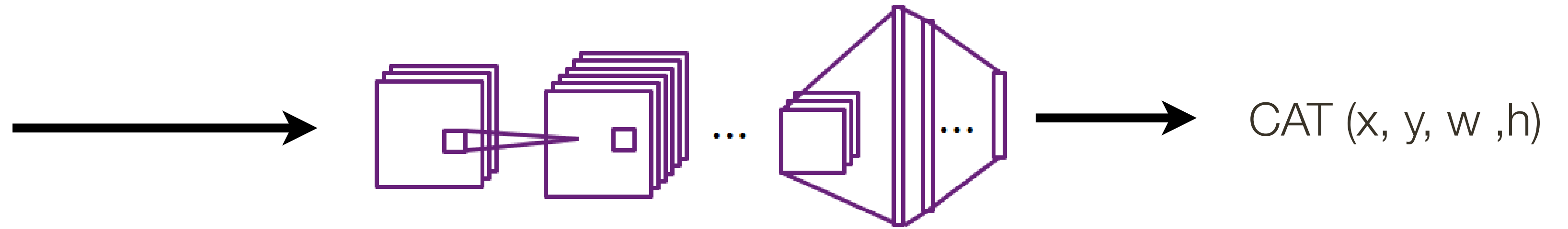
Detection



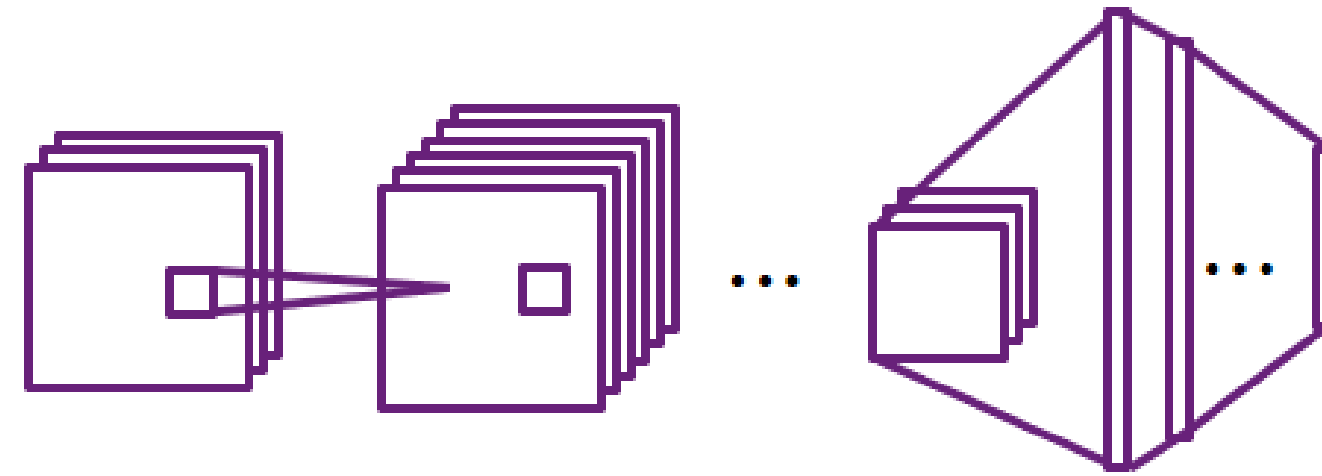
Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)



Object **Detection** as Regression Problem



Object **Detection** as Regression Problem

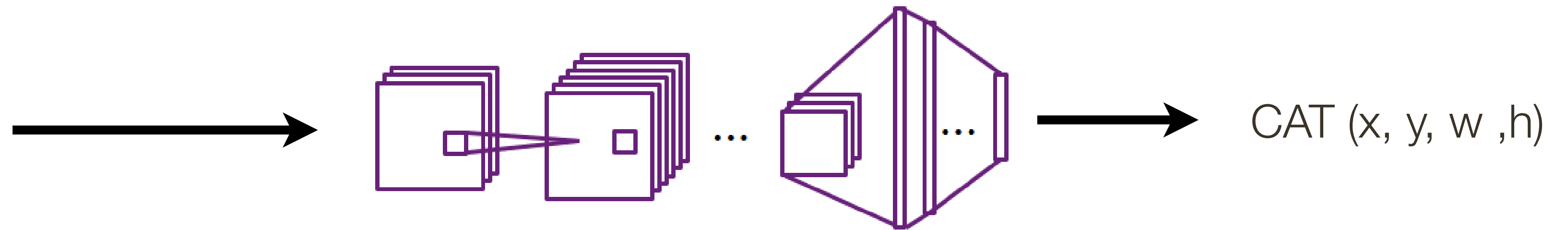


CAT (x, y, w, h)

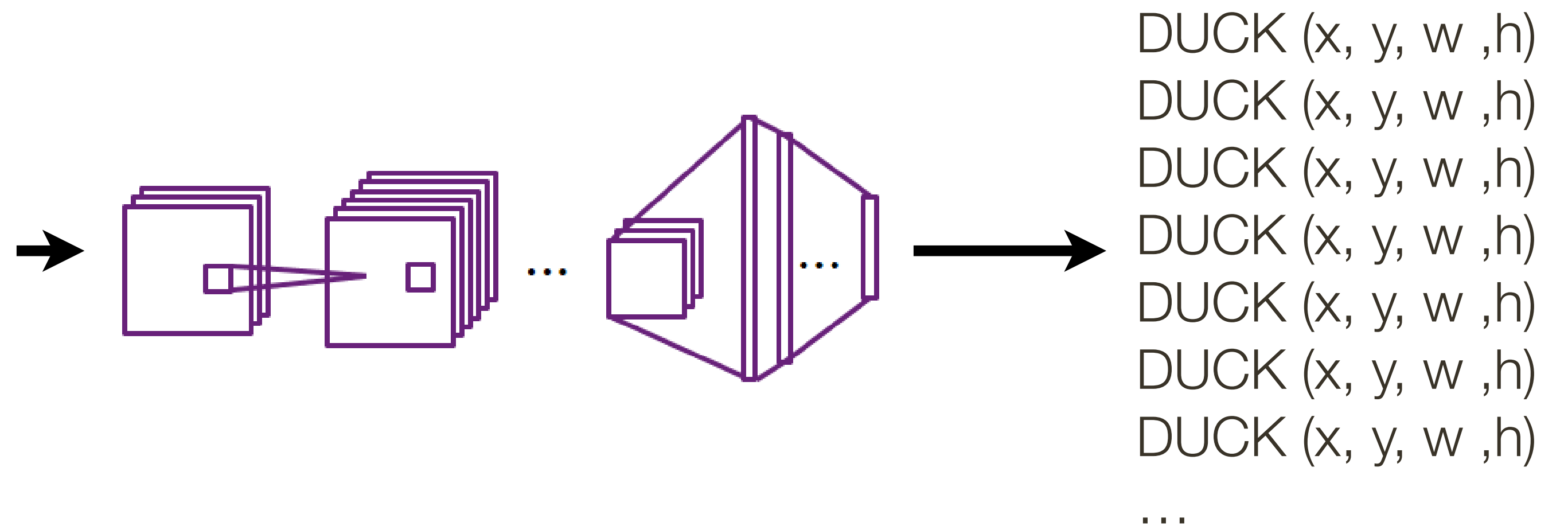


DUCK (x, y, w, h)
DUCK (x, y, w, h)
DUCK (x, y, w, h)
DUCK (x, y, w, h)
DUCK (x, y, w, h)
DUCK (x, y, w, h)
DUCK (x, y, w, h)
DUCK (x, y, w, h)
...

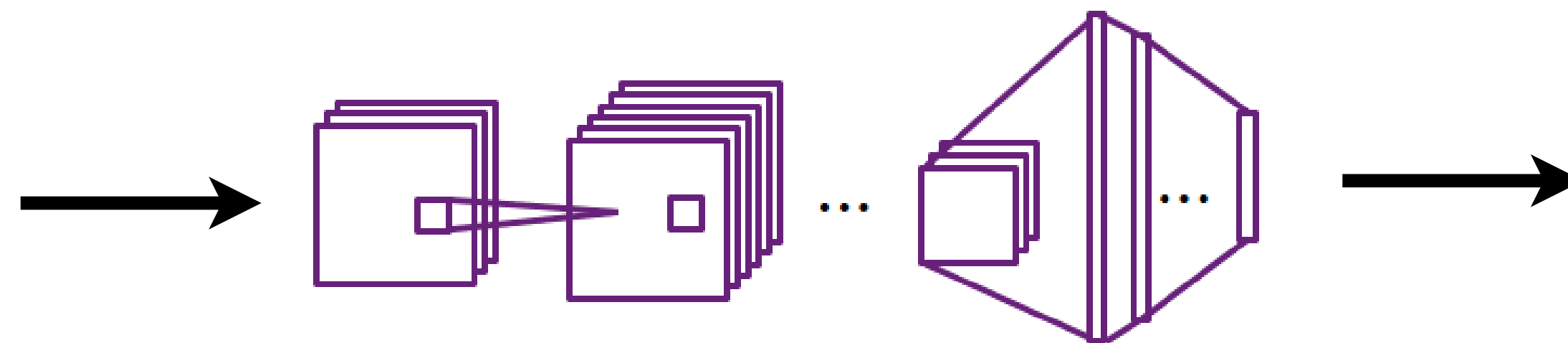
Object **Detection** as Regression Problem



Problem: each image needs a different number of outputs



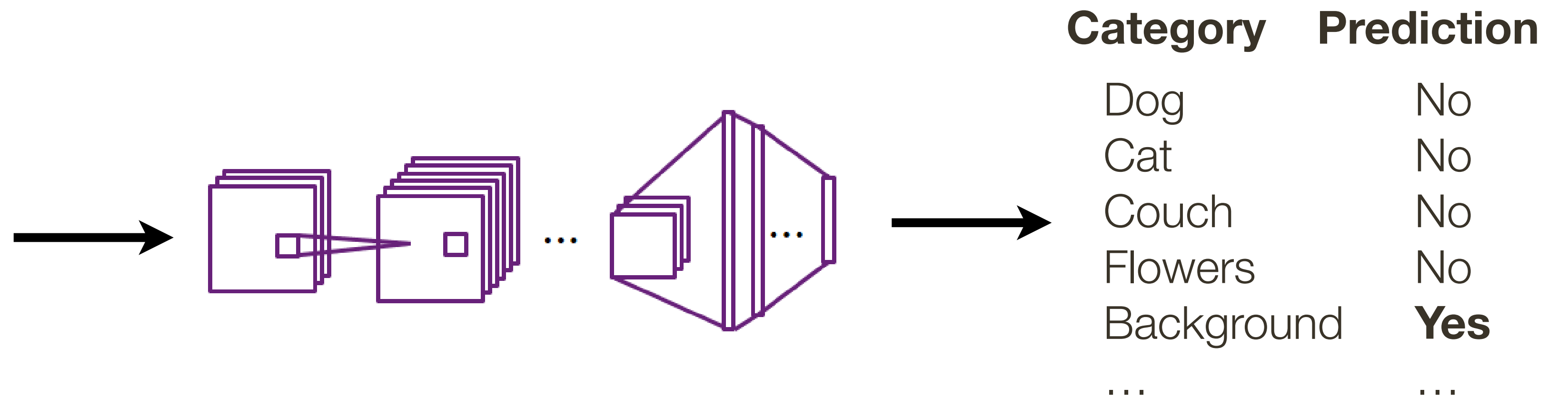
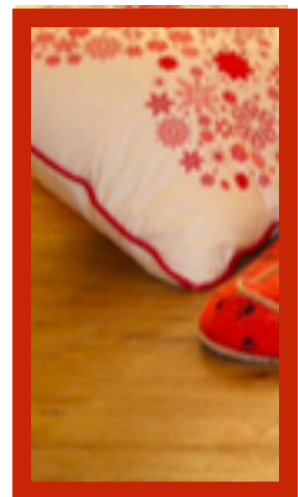
Object **Detection** as Classification Problem



Category	Prediction
Dog	No
Cat	No
Couch	No
Flowers	No
Background	Yes
...	...

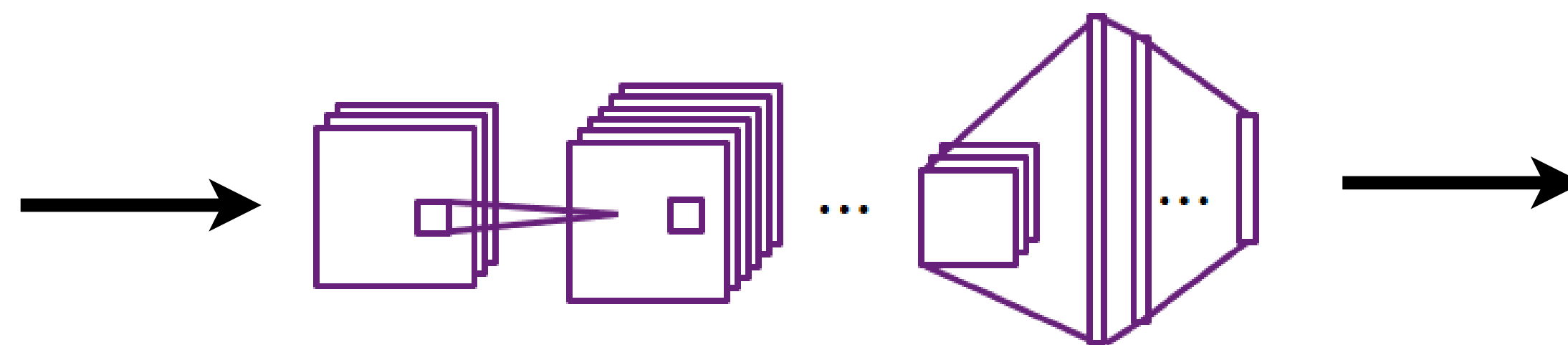
Apply CNN to many different crops in the image and (classification) CNN classifies each patch as object or background

Object **Detection** as Classification Problem



Apply CNN to many different crops in the image and (classification) CNN classifies each patch as object or background

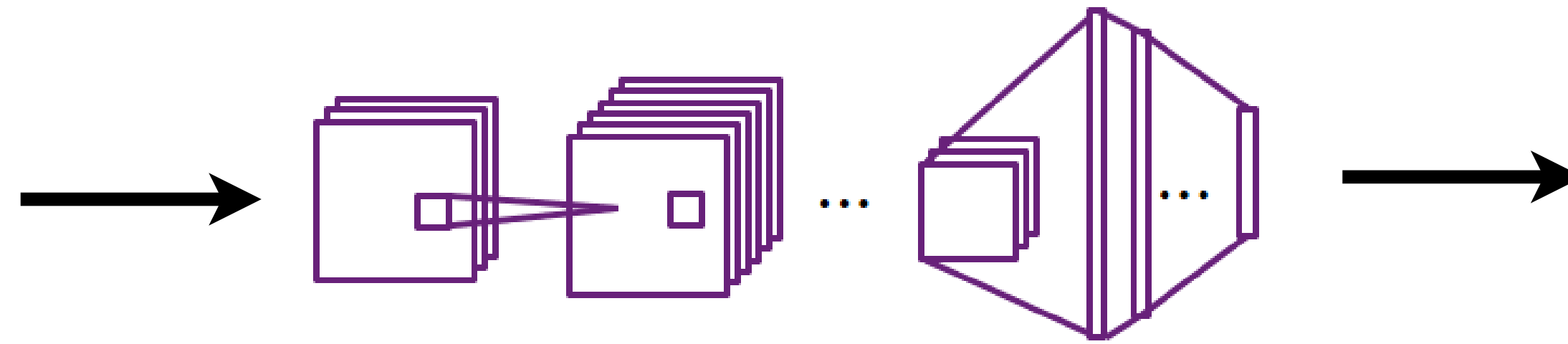
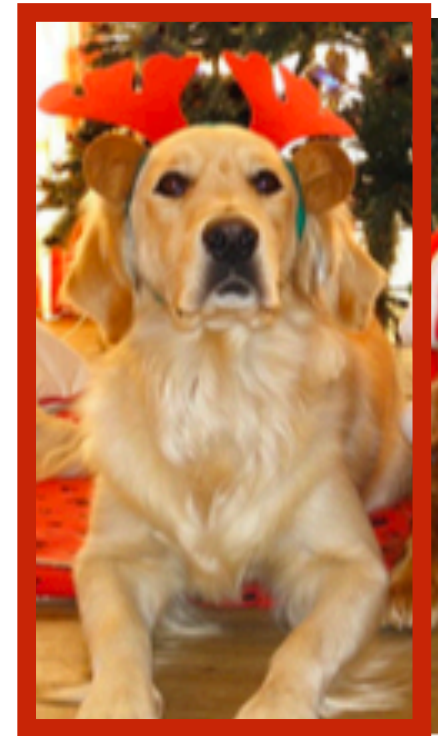
Object **Detection** as Classification Problem



Category	Prediction
Dog	Yes
Cat	No
Couch	No
Flowers	No
Background	No
...	...

Apply CNN to many different crops in the image and (classification) CNN classifies each patch as object or background

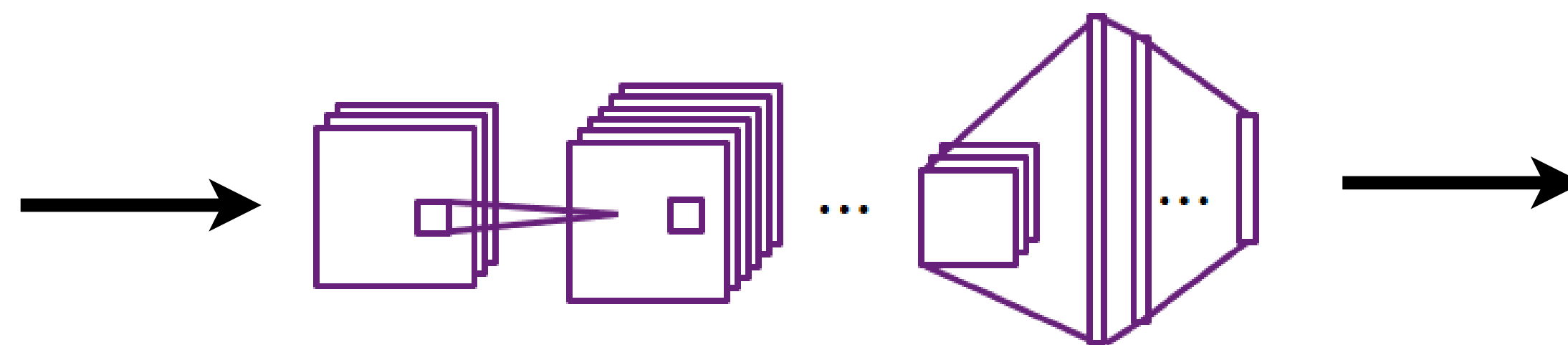
Object **Detection** as Classification Problem



Category	Prediction
Dog	Yes
Cat	No
Couch	No
Flowers	No
Background	No
...	...

Apply CNN to many different crops in the image and (classification) CNN classifies each patch as object or background

Object **Detection** as Classification Problem

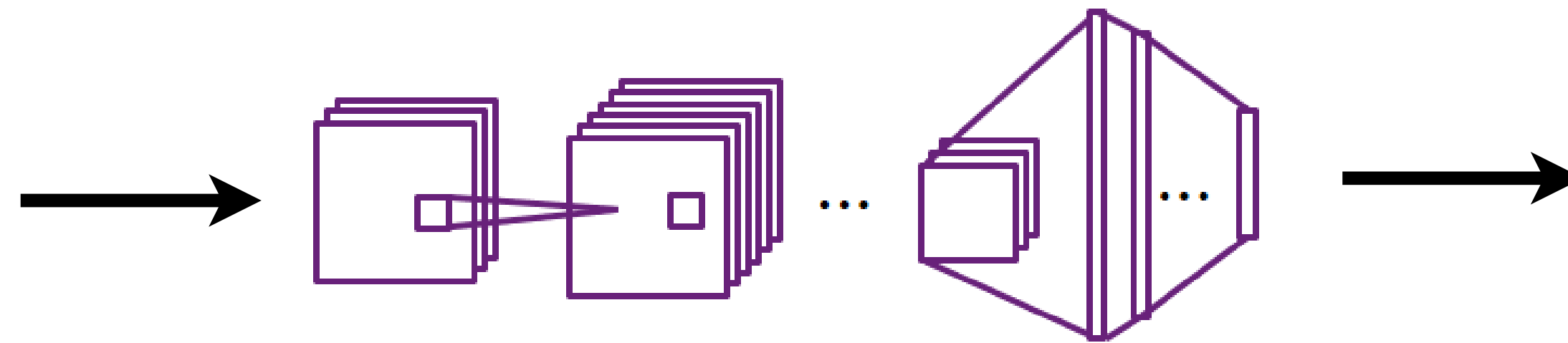


Category	Prediction
Dog	No
Cat	Yes
Couch	No
Flowers	No
Background	No
...	...

Apply CNN to many different crops in the image and (classification) CNN classifies each patch as object or background

Object **Detection** as Classification Problem

Problem: Need to apply CNN to **many** patches in each image



Category	Prediction
Dog	No
Cat	Yes
Couch	No
Flowers	No
Background	No
...	...

Apply CNN to many different crops in the image and (classification) CNN classifies each patch as object or background

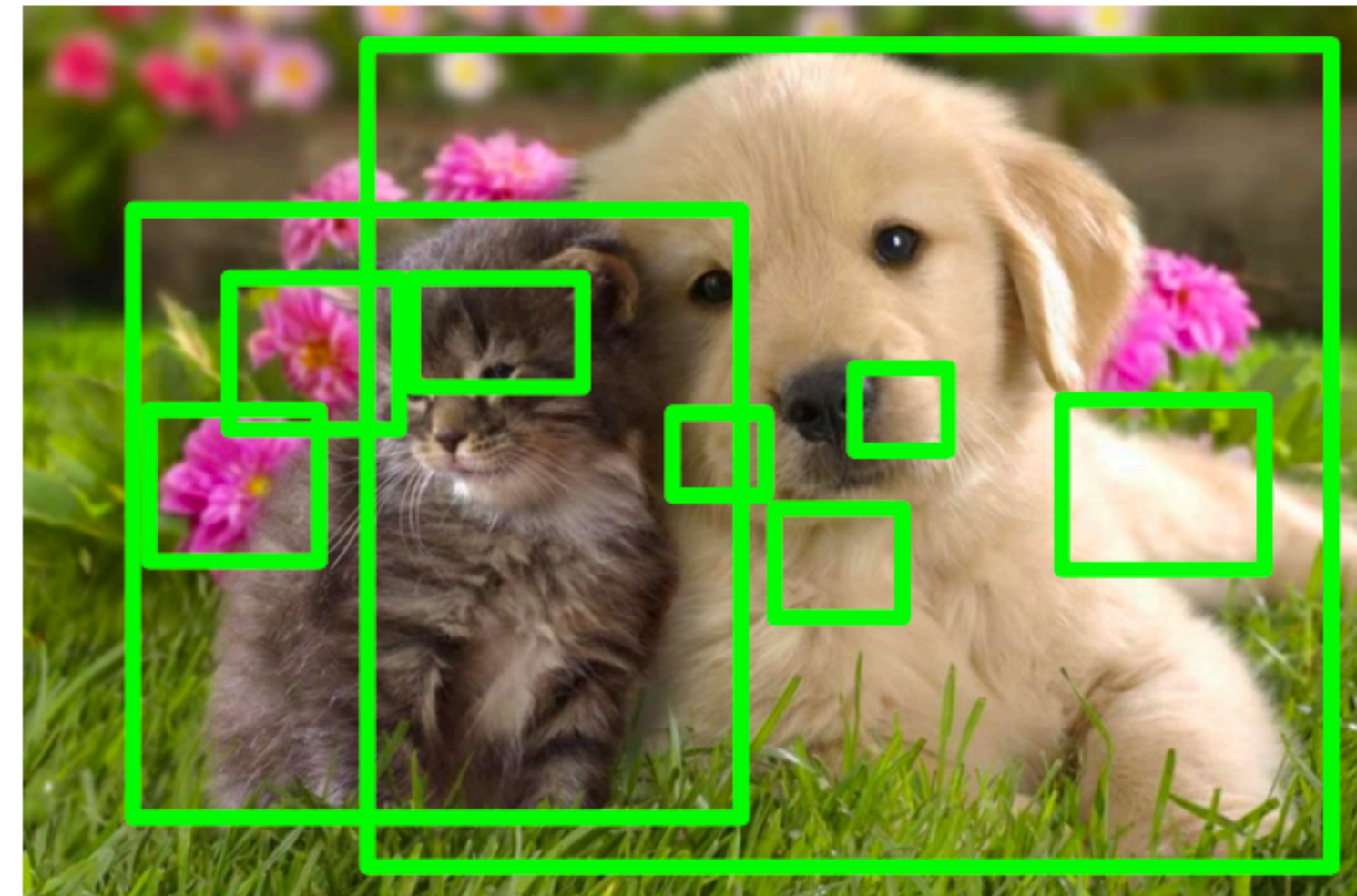
Region Proposals (older idea in vision)

[Alexe et al, TPAMI 2012]
[Ujkings et al, IJCV 2013]
[Cheng et al, CVPR 2014]
[Zitnick and Dollar, ECCV 2014]

Find image **regions that are likely contain objects** (any object at all)

- typically works by looking at histogram distributions, region aspect ratio, closed contours, coherent color

Relatively **fast to run** (Selective Search gives 1000 region proposals in a few seconds on a CPU)



Goal: Get “true” object regions to be in as few top K proposals as possible

R-CNN

[Girshick et al, CVPR 2014]



Input **Image**

* image from Ross Girshick

R-CNN

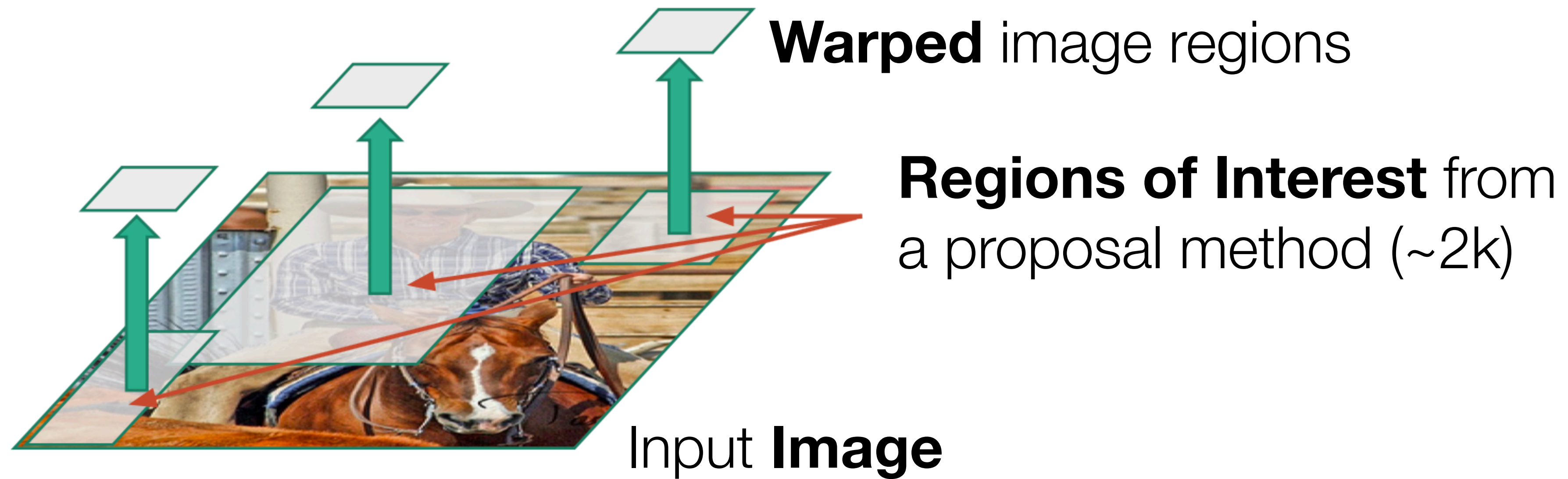
[Girshick et al, CVPR 2014]



* image from Ross Girshick

R-CNN

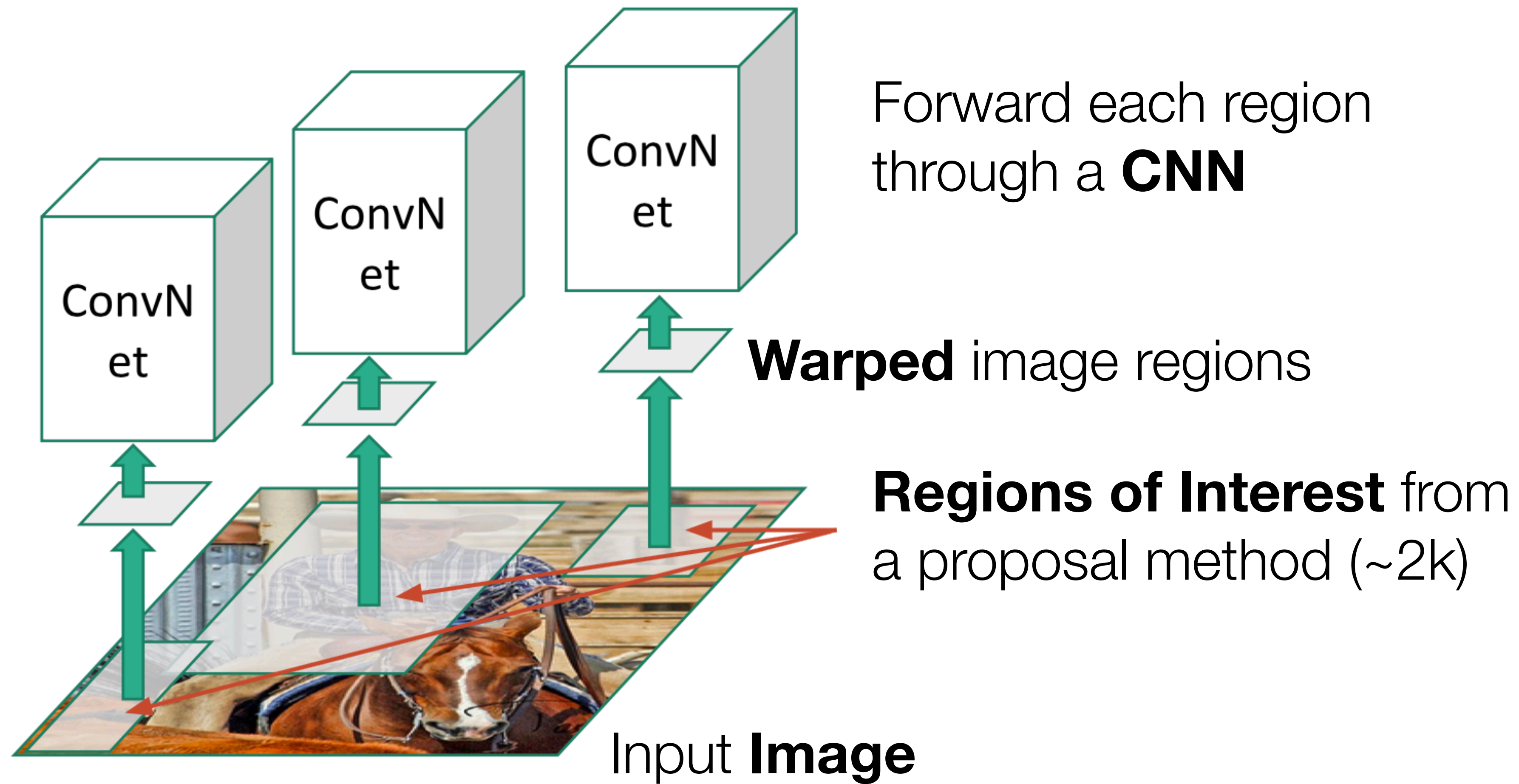
[Girshick et al, CVPR 2014]



* image from Ross Girshick

R-CNN

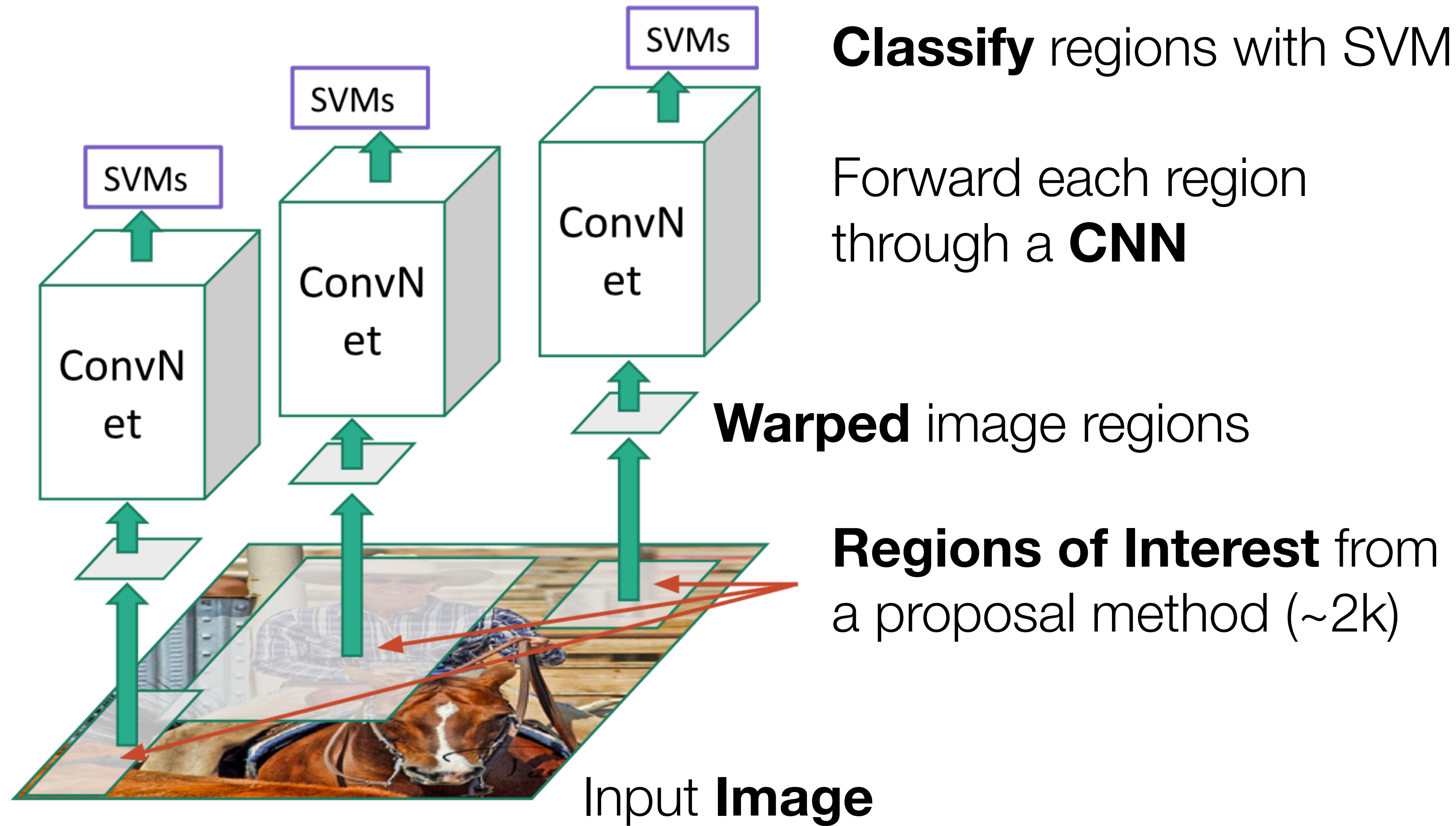
[Girshick et al, CVPR 2014]



* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]

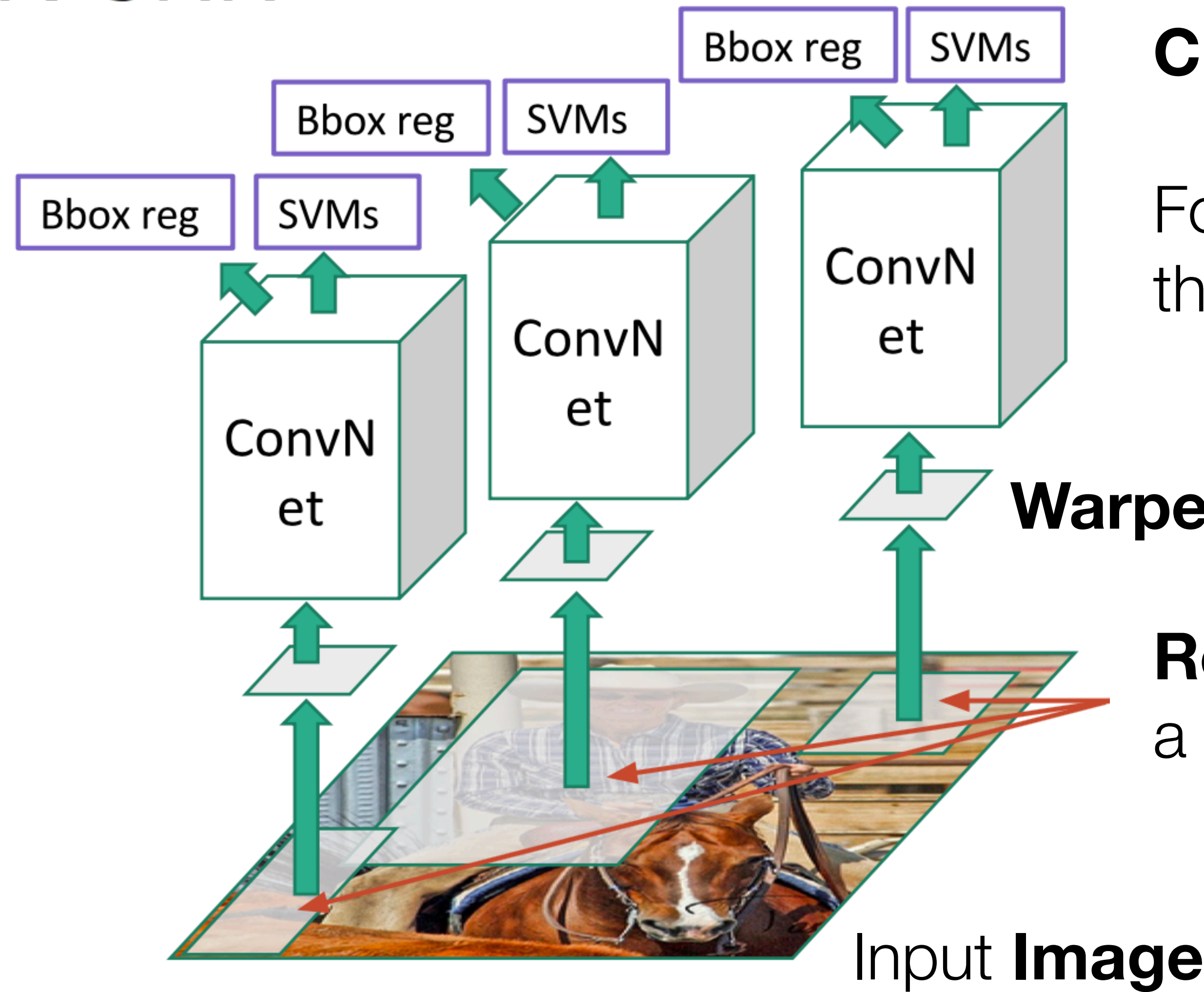


* image from Ross Girshick

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)

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)

Fast **R-CNN**

[Girshick et al, ICCV 2015]

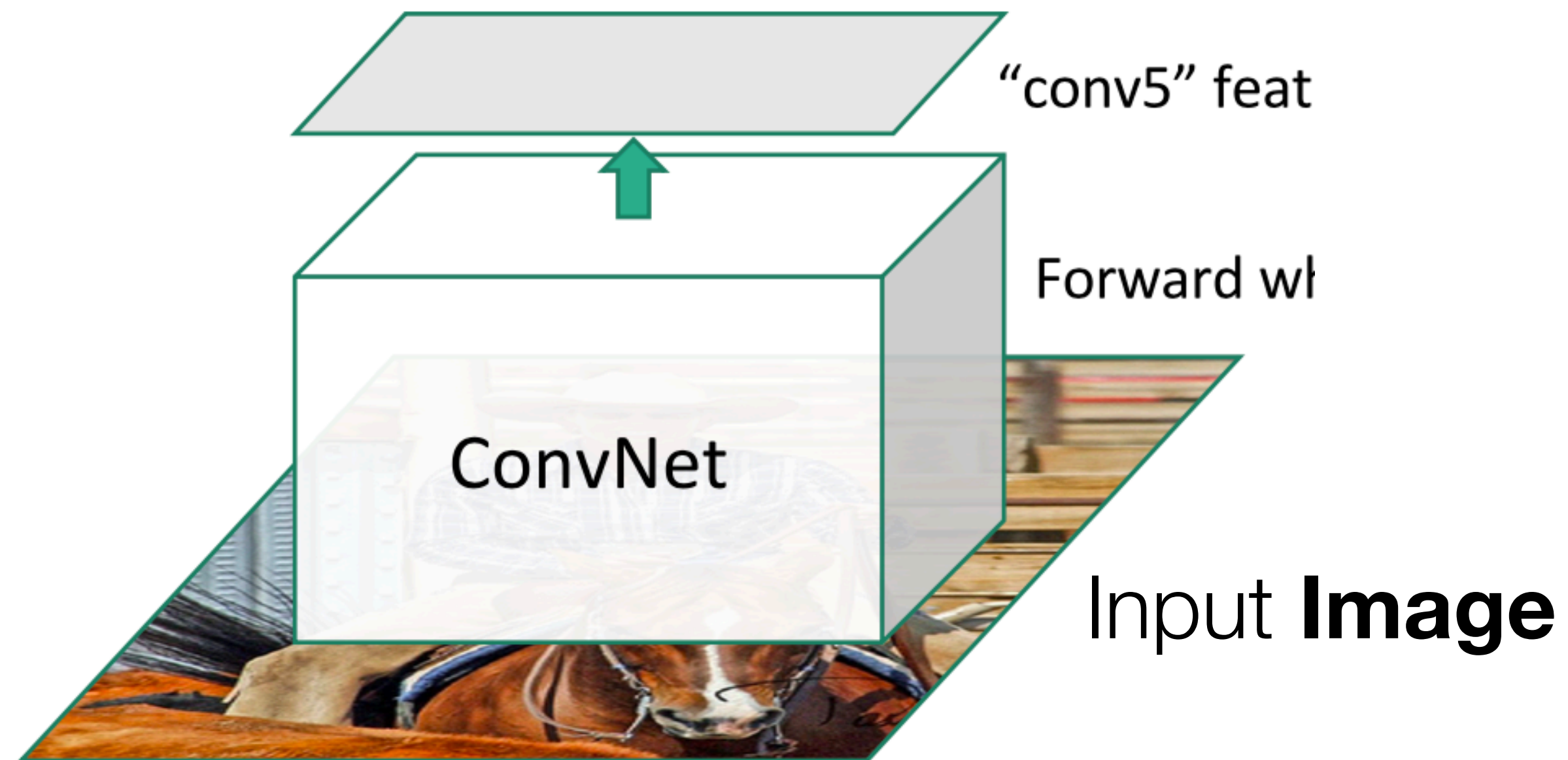


Input **Image**

* image from Ross Girshick

Fast **R-CNN**

[Girshick et al, ICCV 2015]



* image from Ross Girshick

Fast R-CNN

[Girshick et al, ICCV 2015]



* image from Ross Girshick

Fast R-CNN

[Girshick et al, ICCV 2015]

Regions of Interest
from the
proposal
method

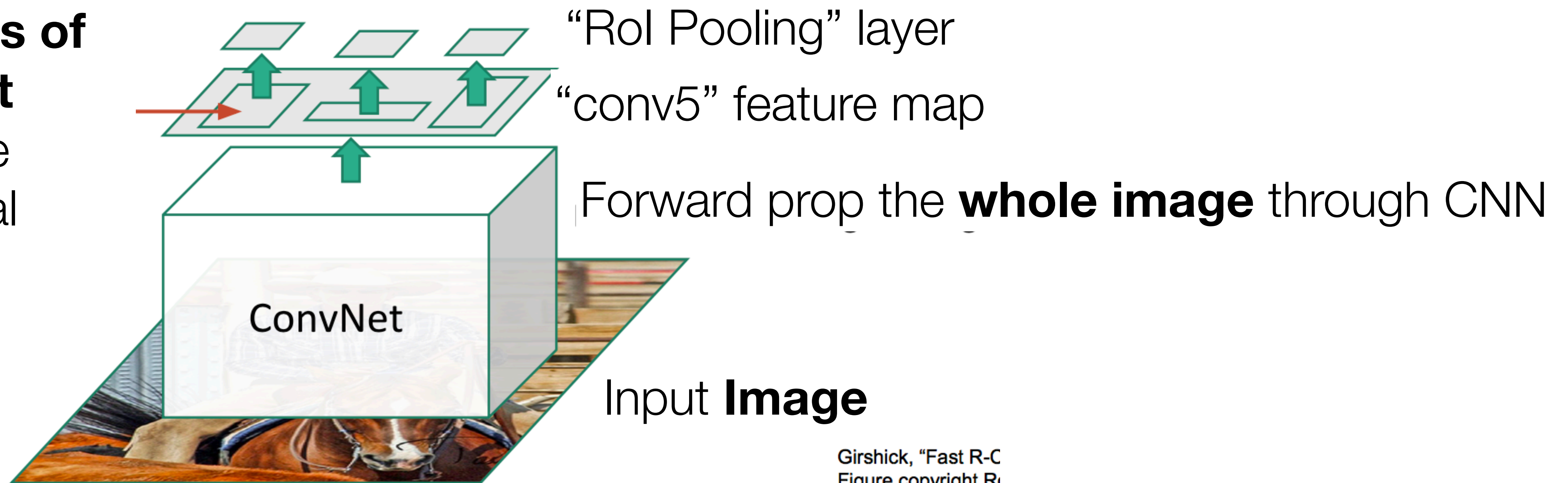


* image from Ross Girshick

Fast R-CNN

[Girshick et al, ICCV 2015]

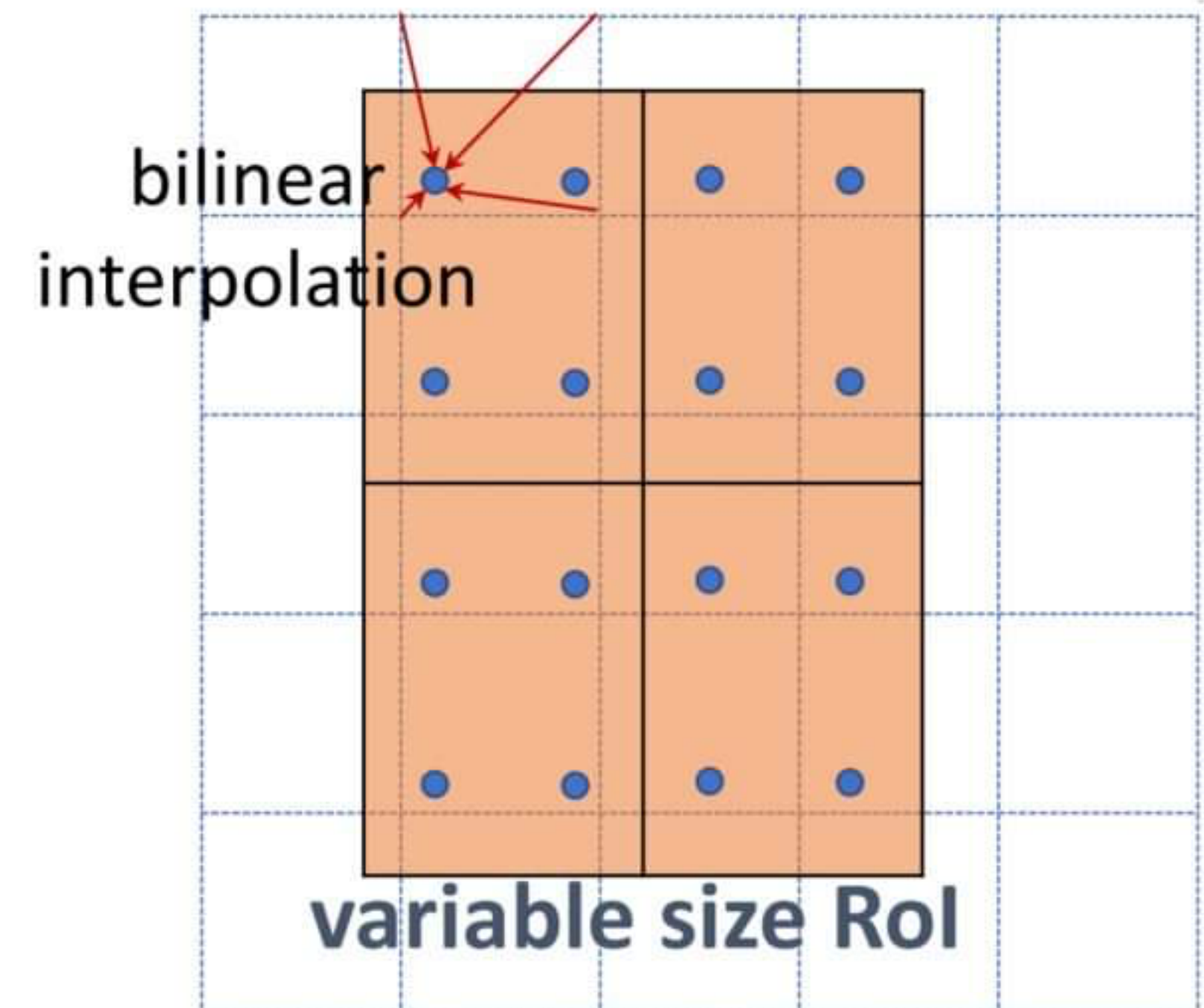
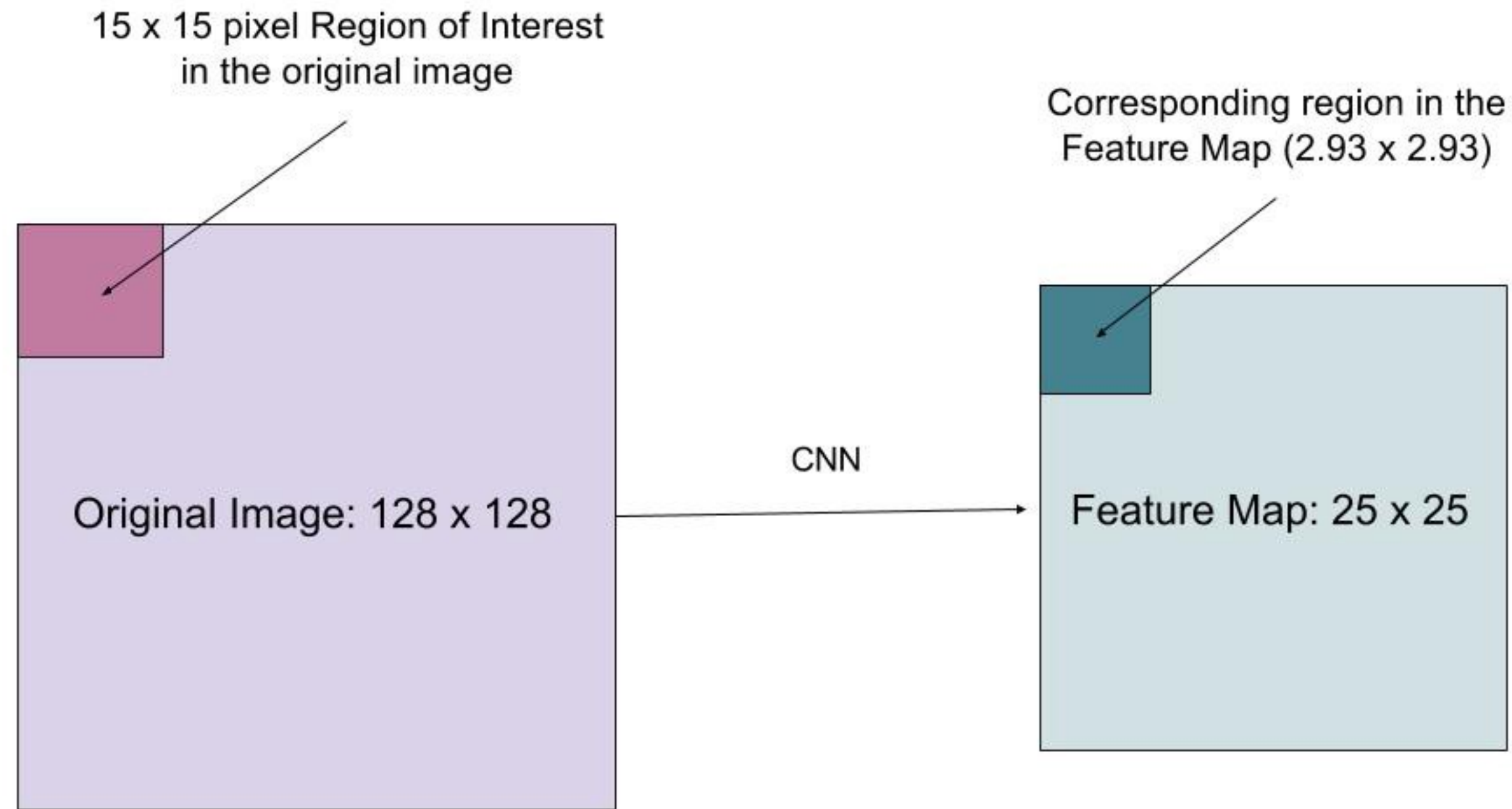
Regions of Interest
from the
proposal
method



Girshick, "Fast R-C
Figure copyright R

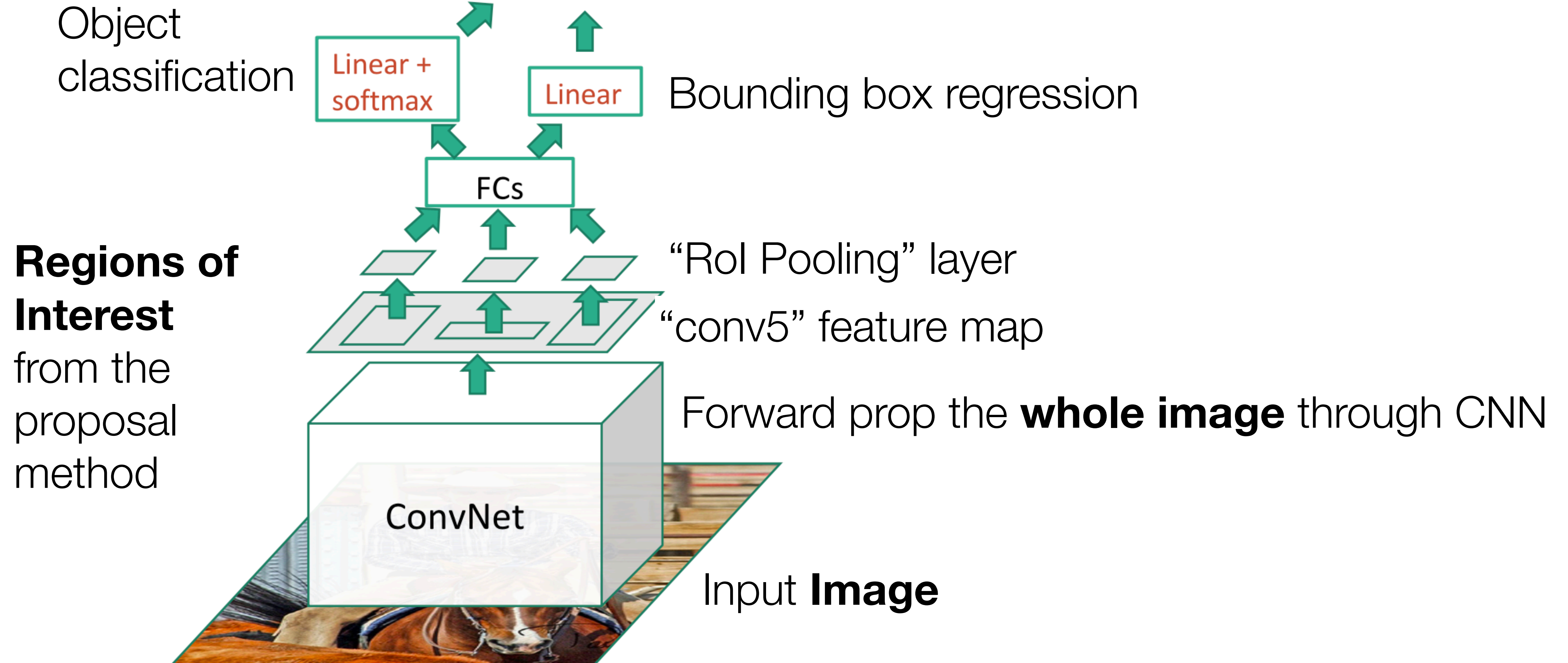
* image from Ross Girshick

RoI Align



Fast R-CNN

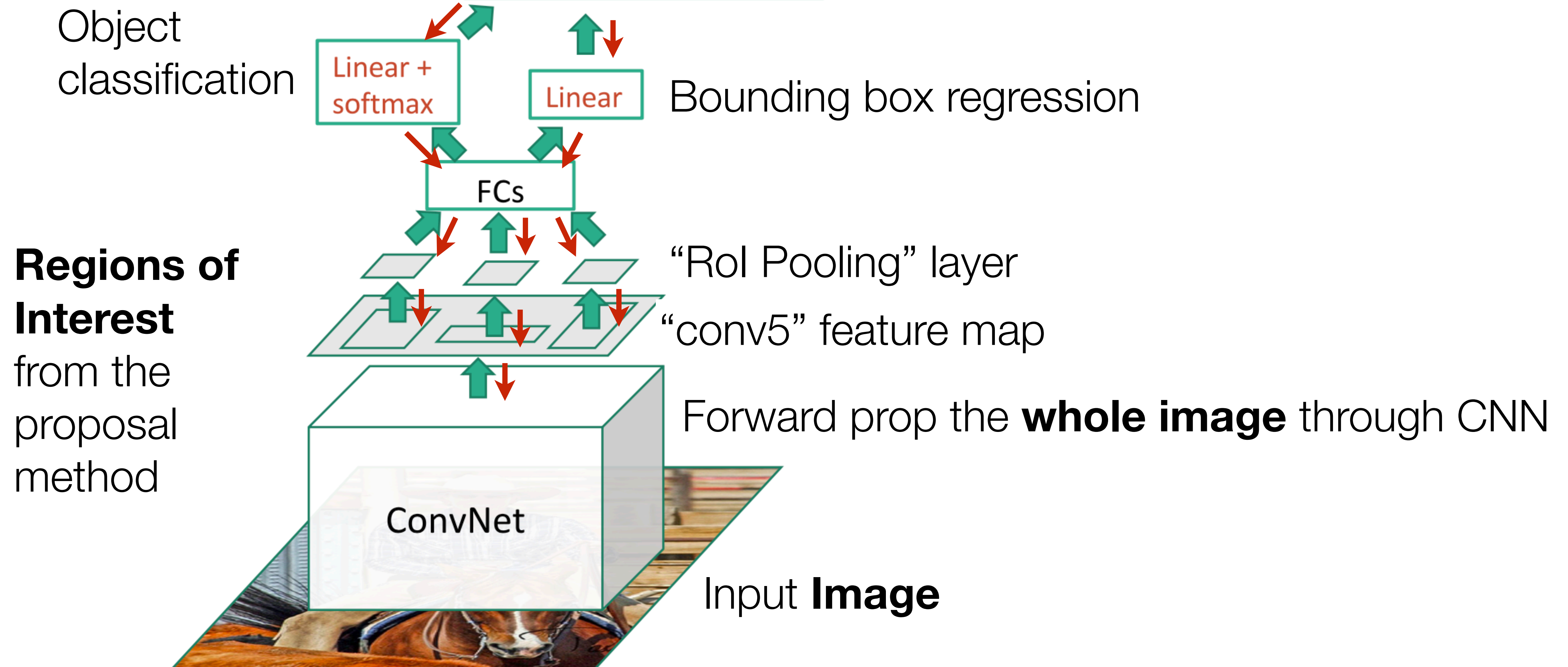
[Girshick et al, ICCV 2015]



* image from Ross Girshick

Fast **R-CNN**: Training

[Girshick et al, ICCV 2015]



* image from Ross Girshick

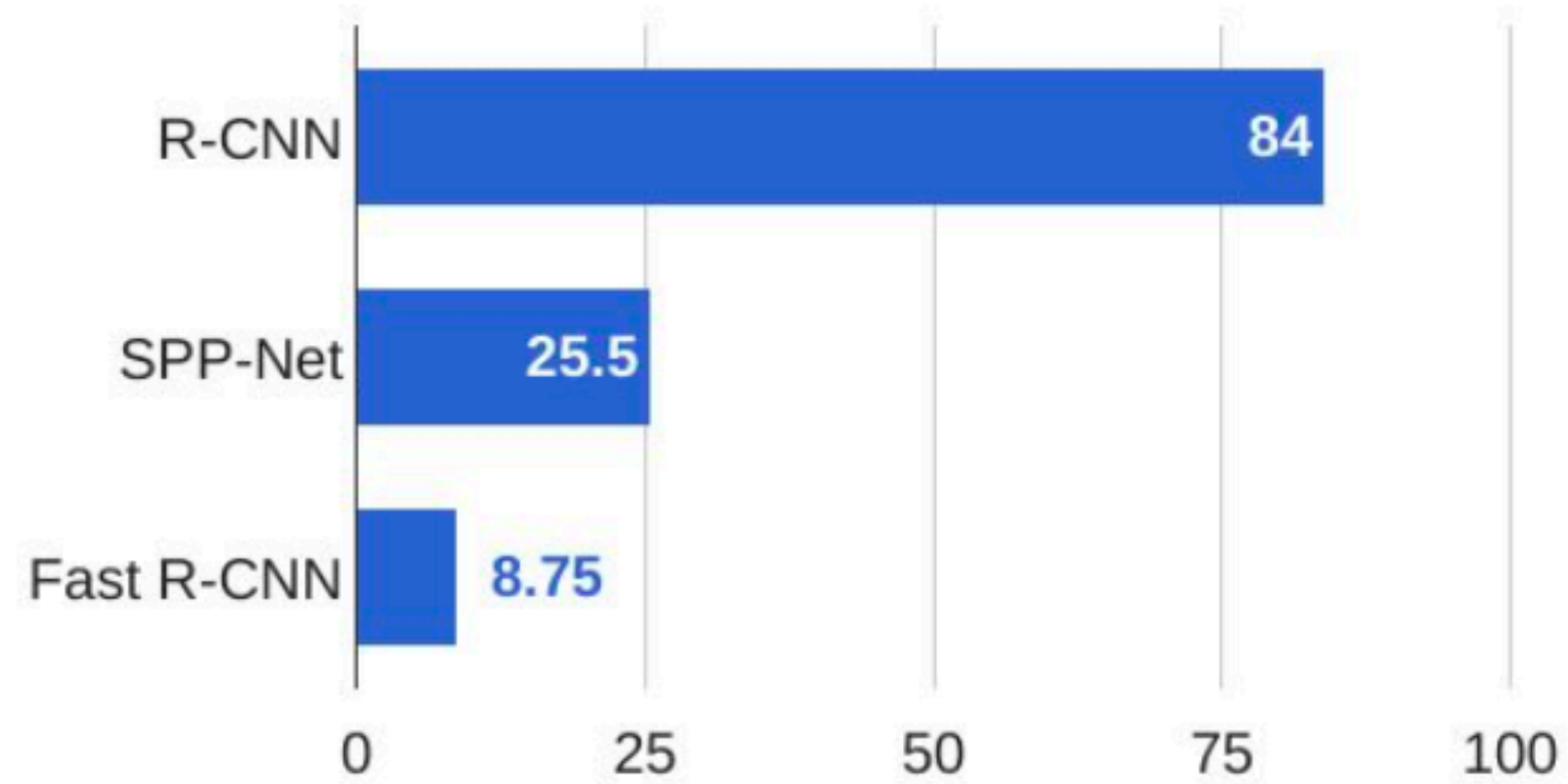
R-CNN vs. SPP vs. Fast R-CNN

[Girshick et al, CVPR 2014]

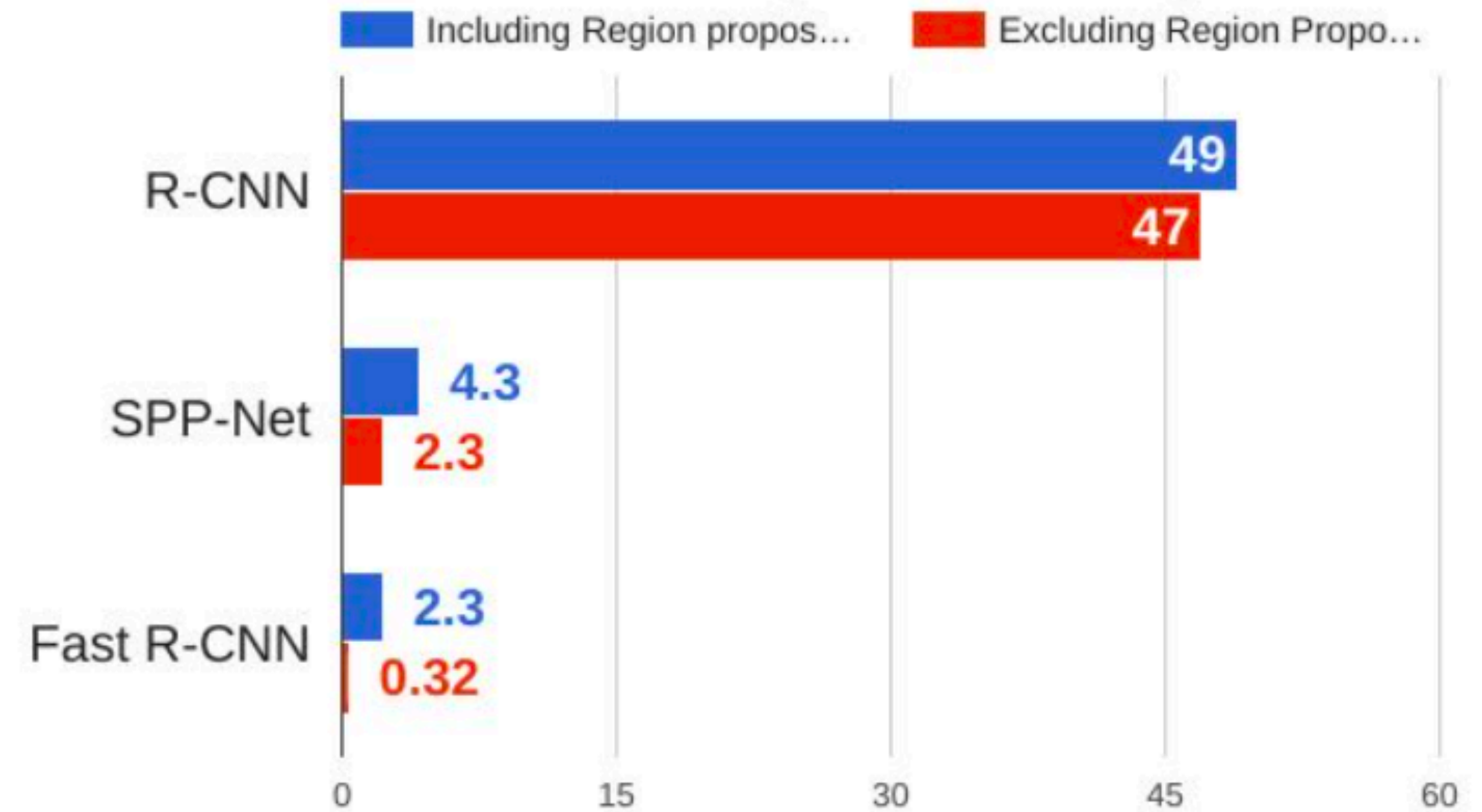
[Girshick et al, ICCV 2015]

[He et al, ECCV 2014]

Training time (Hours)



Test time (seconds)

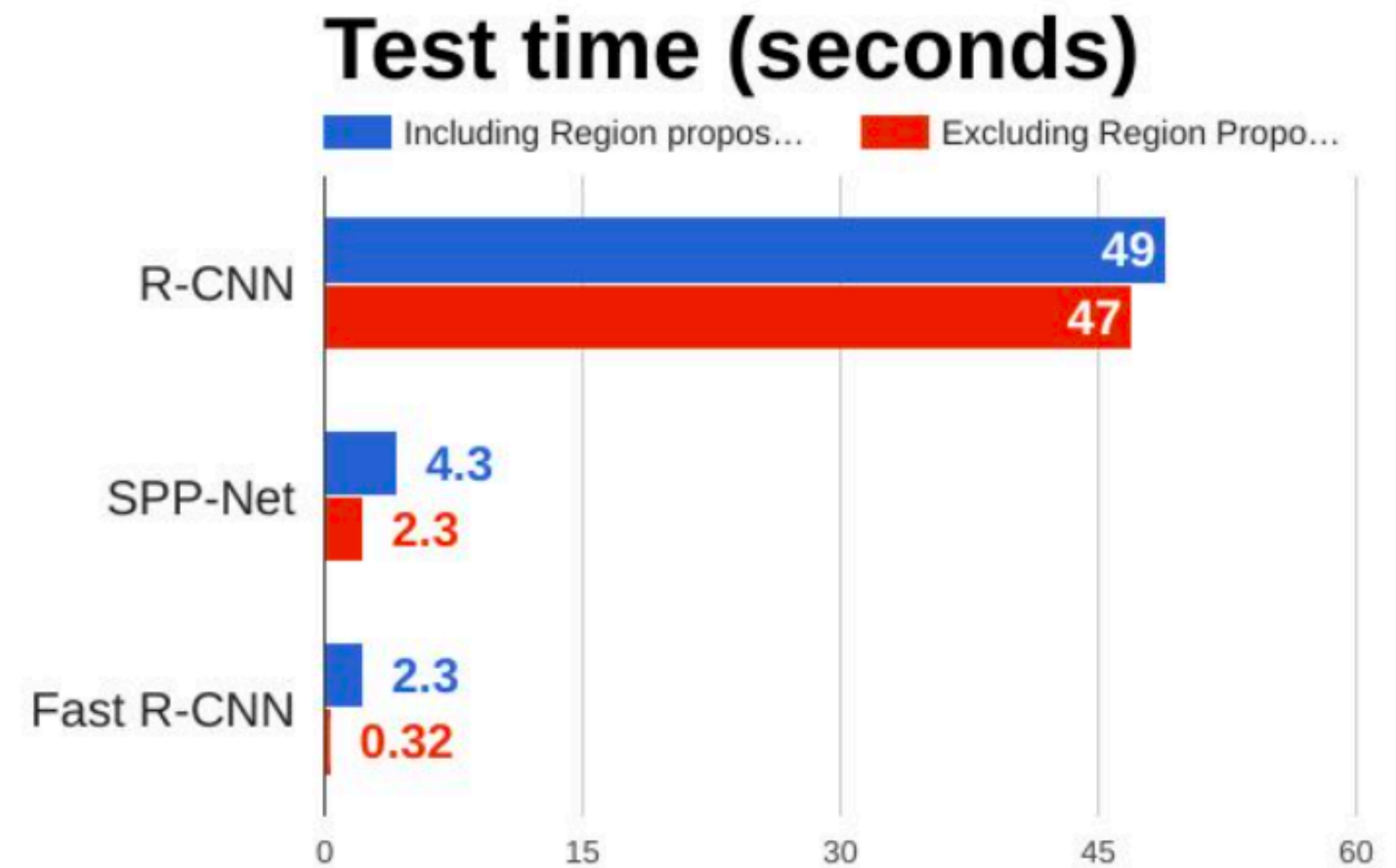
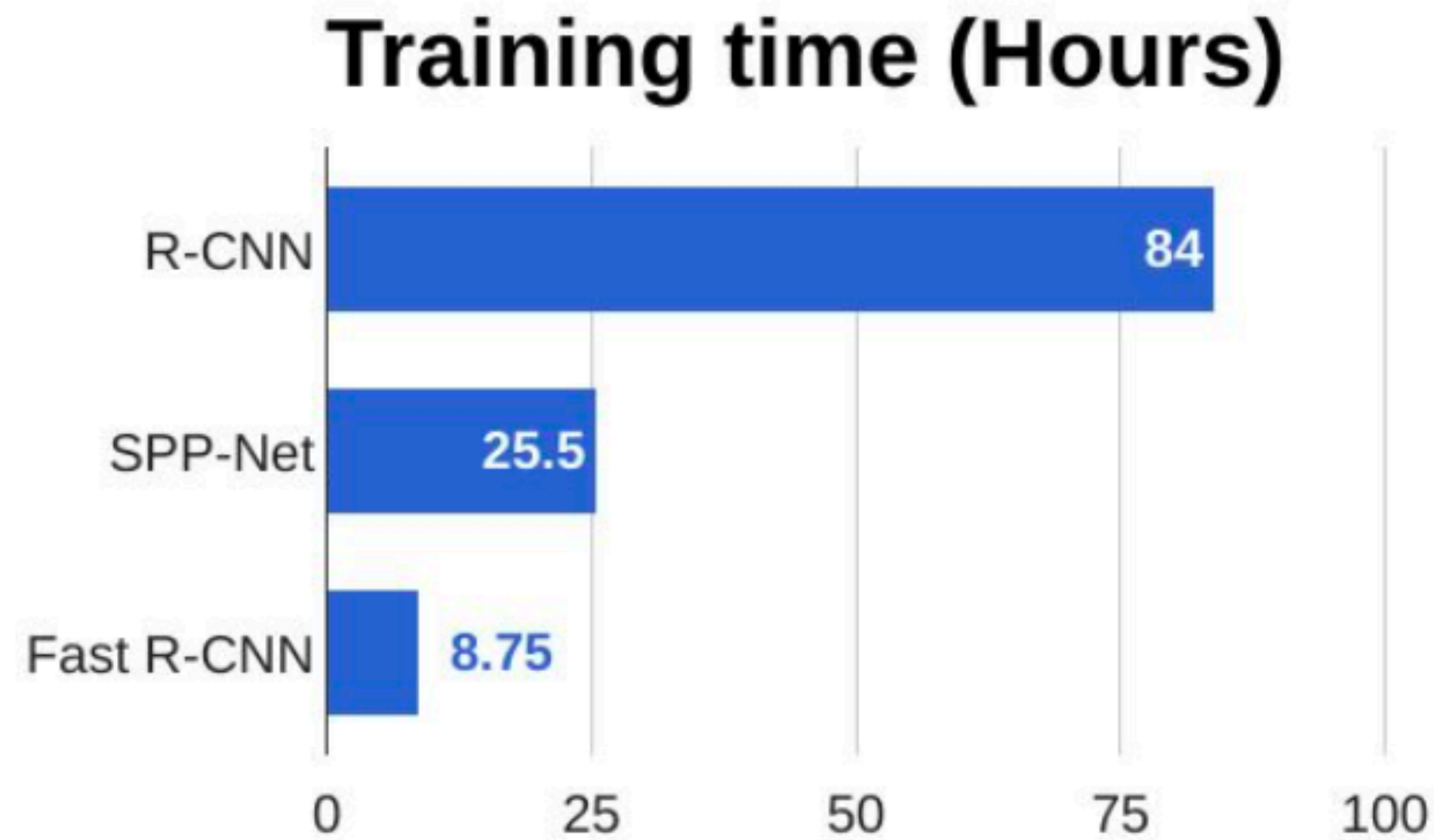


R-CNN vs. SPP vs. Fast R-CNN

[Girshick et al, CVPR 2014]

[Girshick et al, ICCV 2015]

[He et al, ECCV 2014]



Observation: Performance dominated by the region proposals at this point!

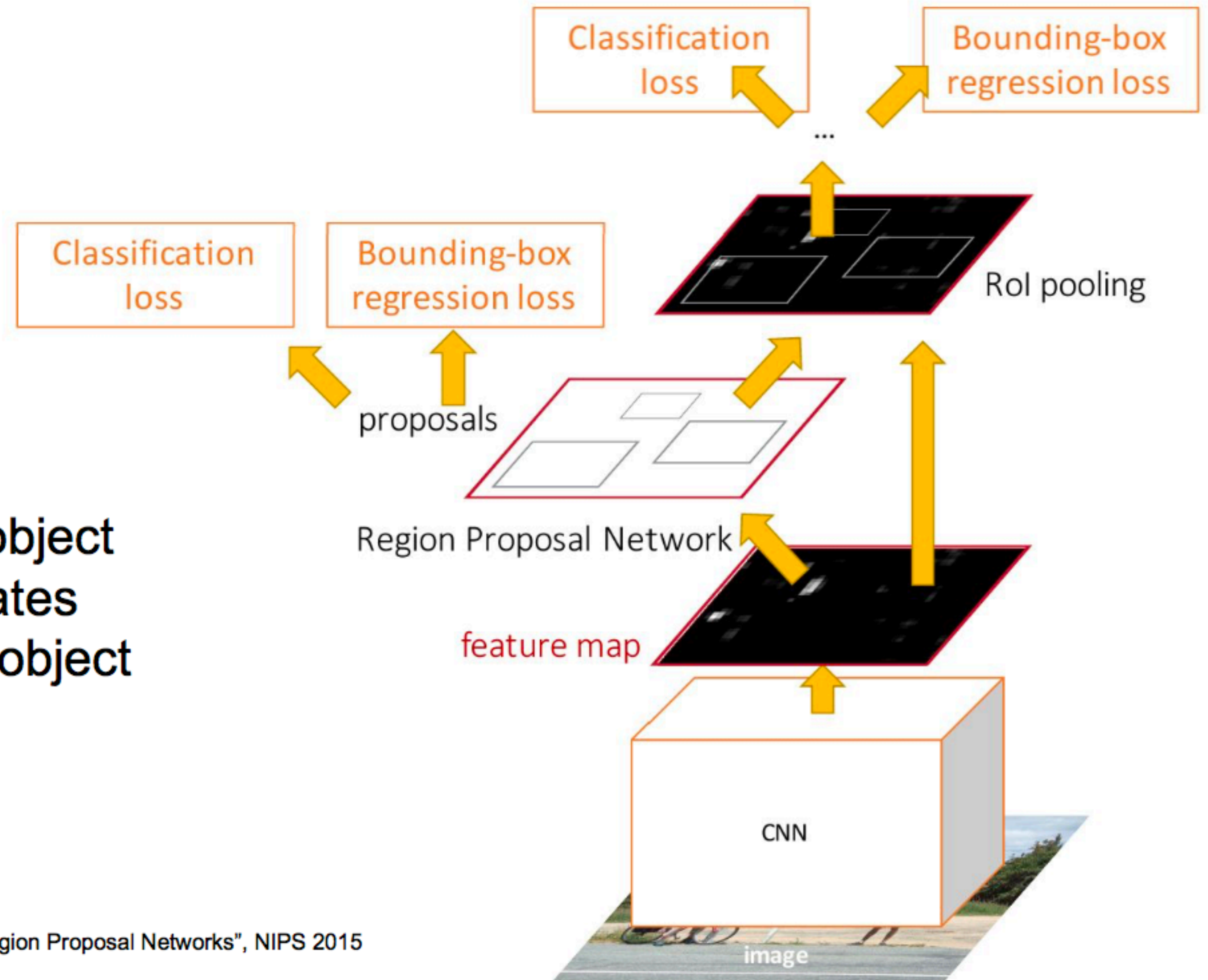
Faster R-CNN

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

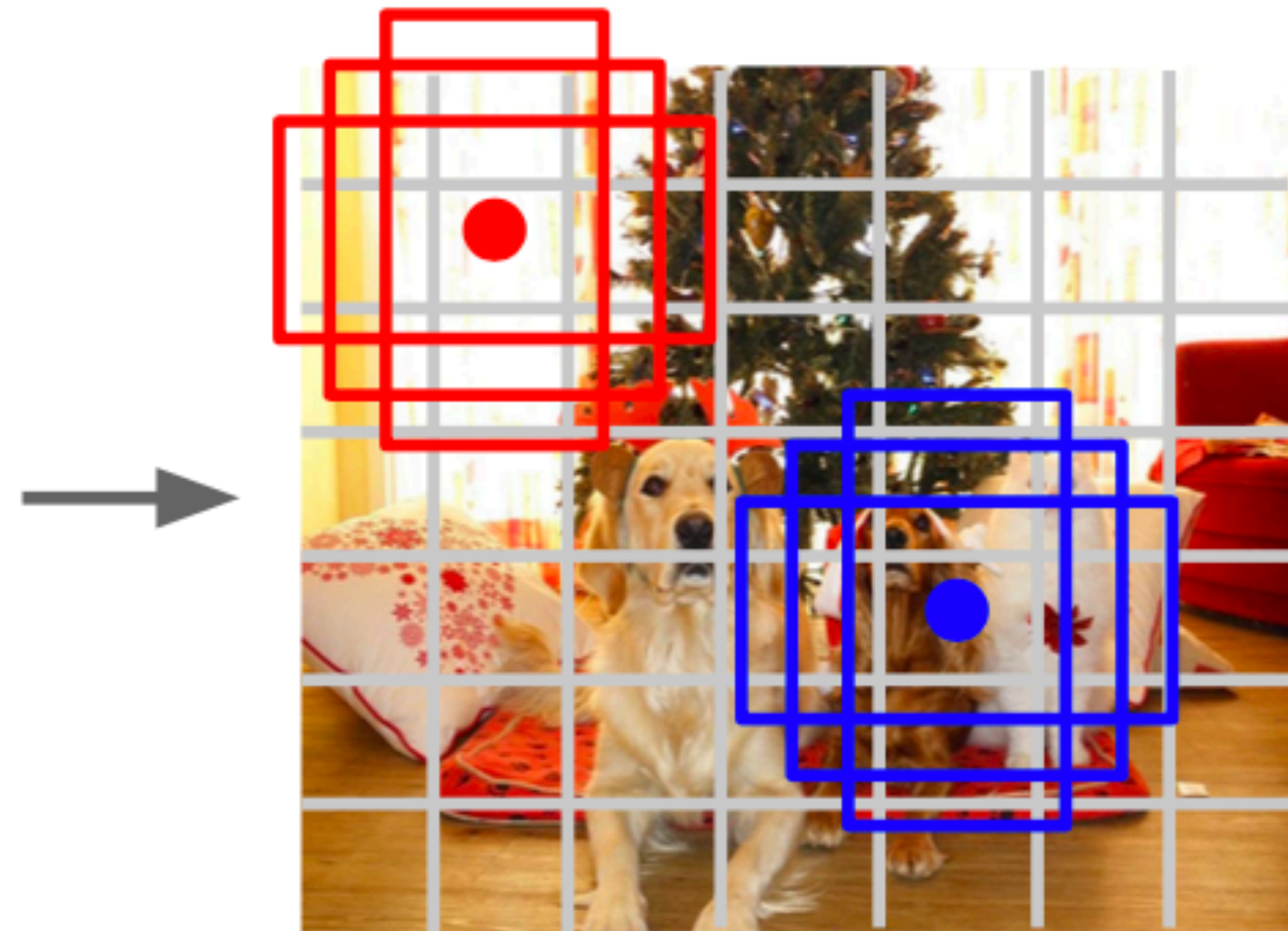


YOLO: You Only Look Once

[Redmon et al, CVPR 2016]



Input image
 $3 \times H \times W$



Divide image into grid
 7×7

Image a set of **base boxes**
centered at each grid cell
Here $B = 3$

Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers:
(dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)

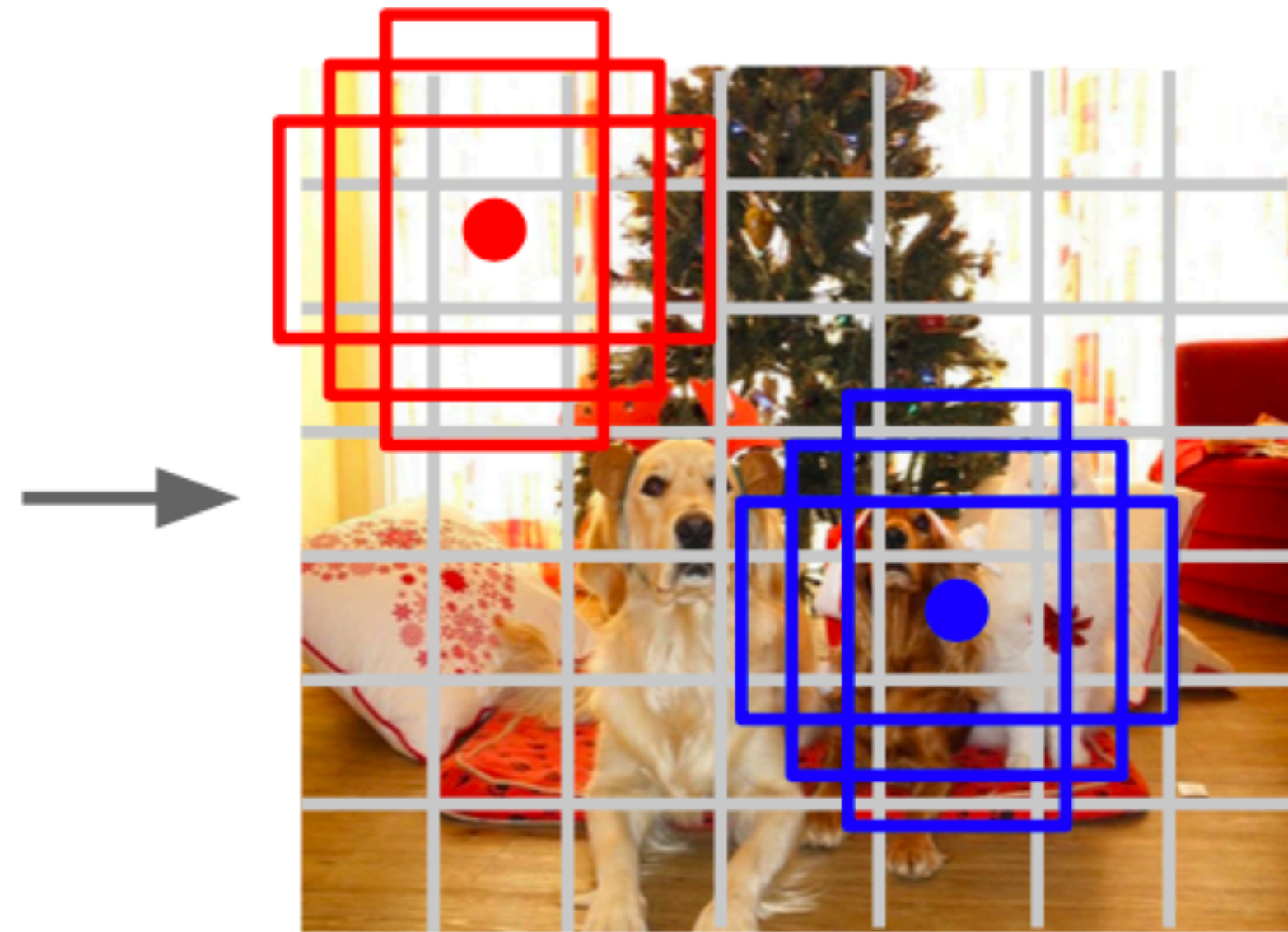
Output:
 $7 \times 7 \times (5 * B + C)$

YOLO: You Only Look Once

[Redmon et al, CVPR 2016]

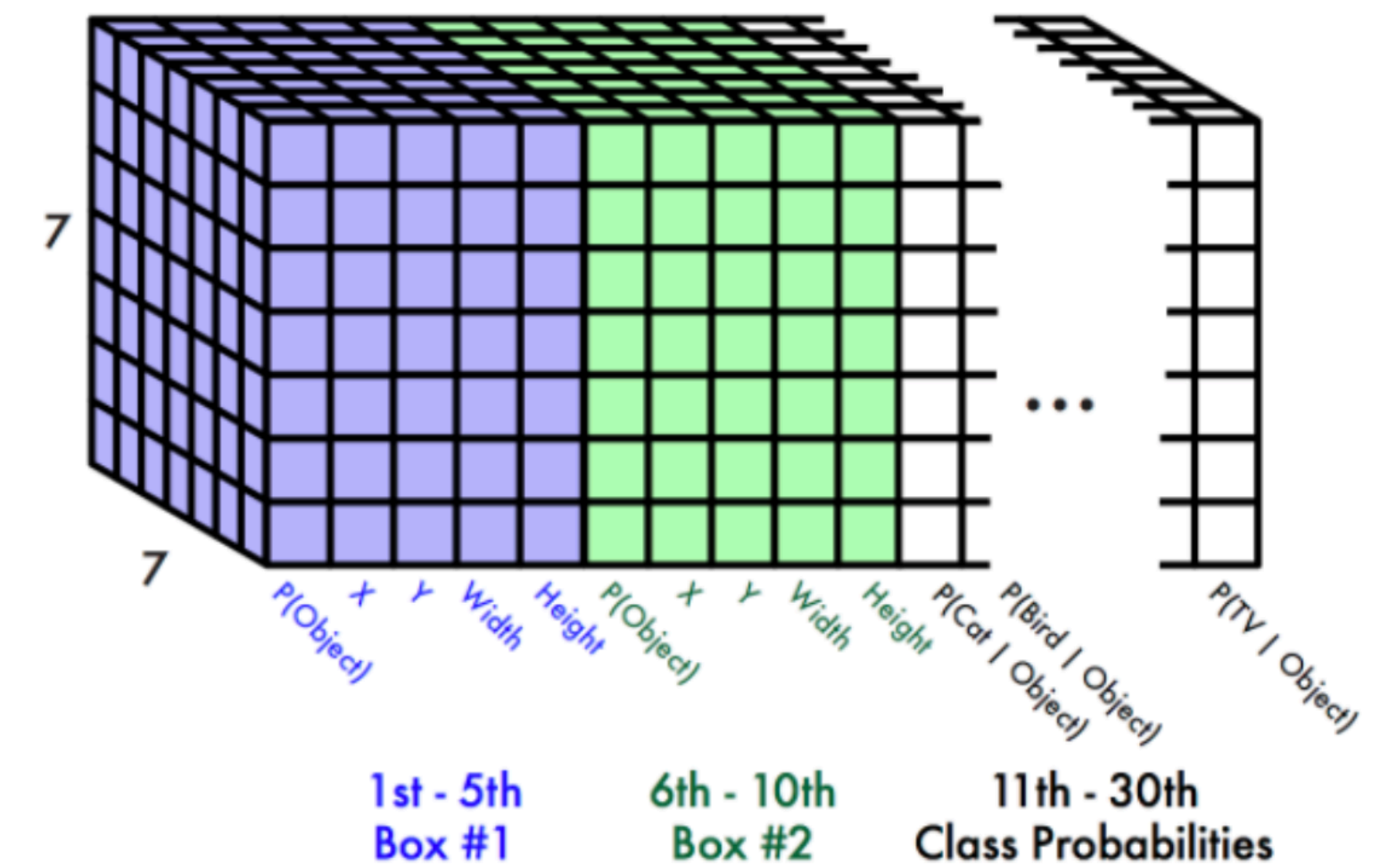


Input image
 $3 \times H \times W$



Divide image into grid
 7×7

Image a set of **base boxes**
centered at each grid cell
Here $B = 3$





YOLO v2

<http://pureddie.com/yolo>



YOLO v2

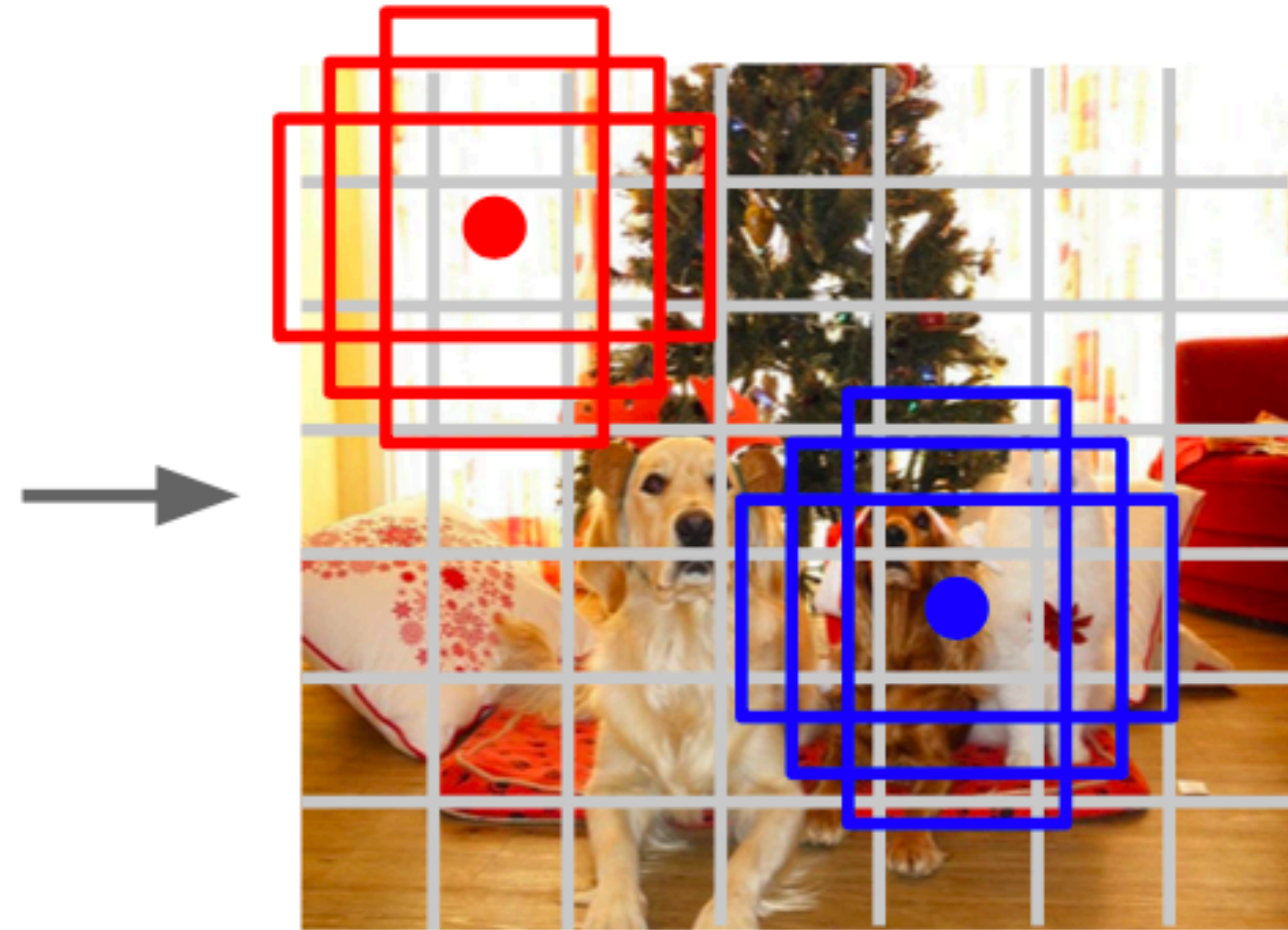
<http://pureddie.com/yolo>

YOLO: You Only Look Once

[Redmon et al, CVPR 2016]

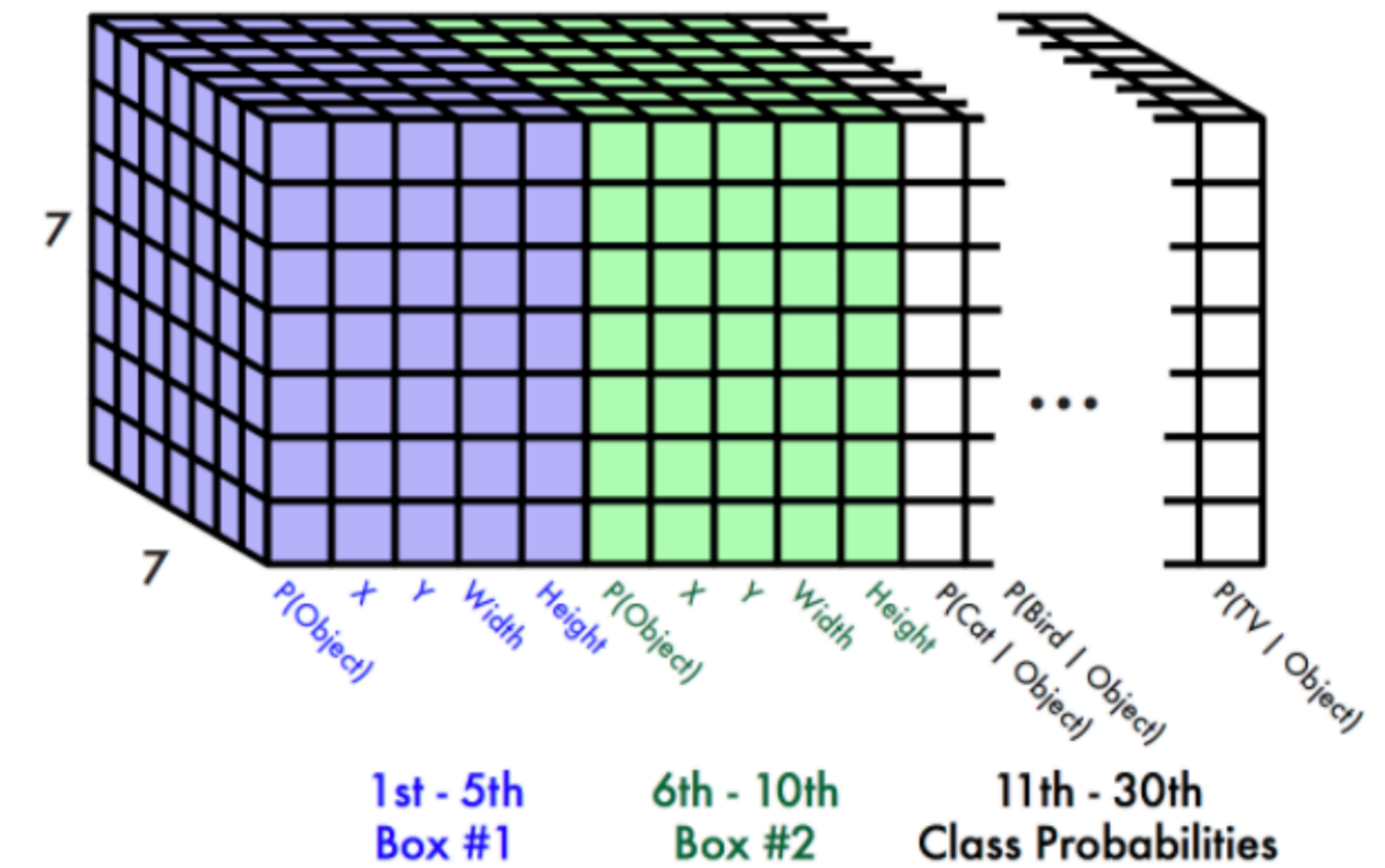


Input image
 $3 \times H \times W$



Divide image into grid
 7×7

Image a set of **base boxes**
centered at each grid cell
Here $B = 3$

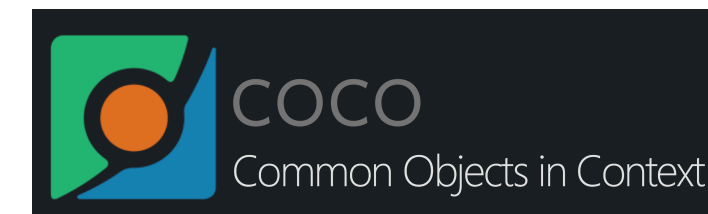


Computer **Vision Problems** (no language for now)

Segmentation

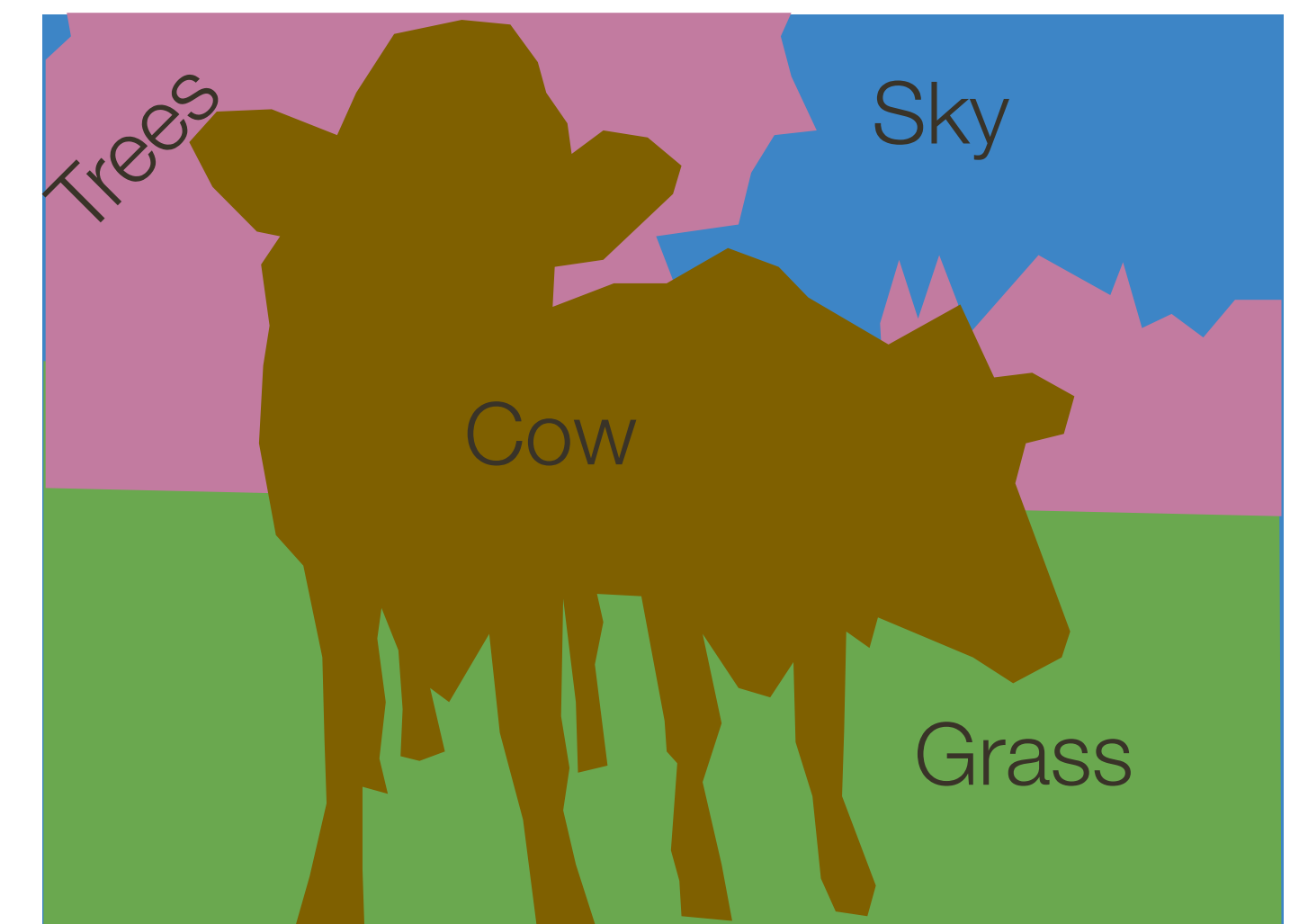
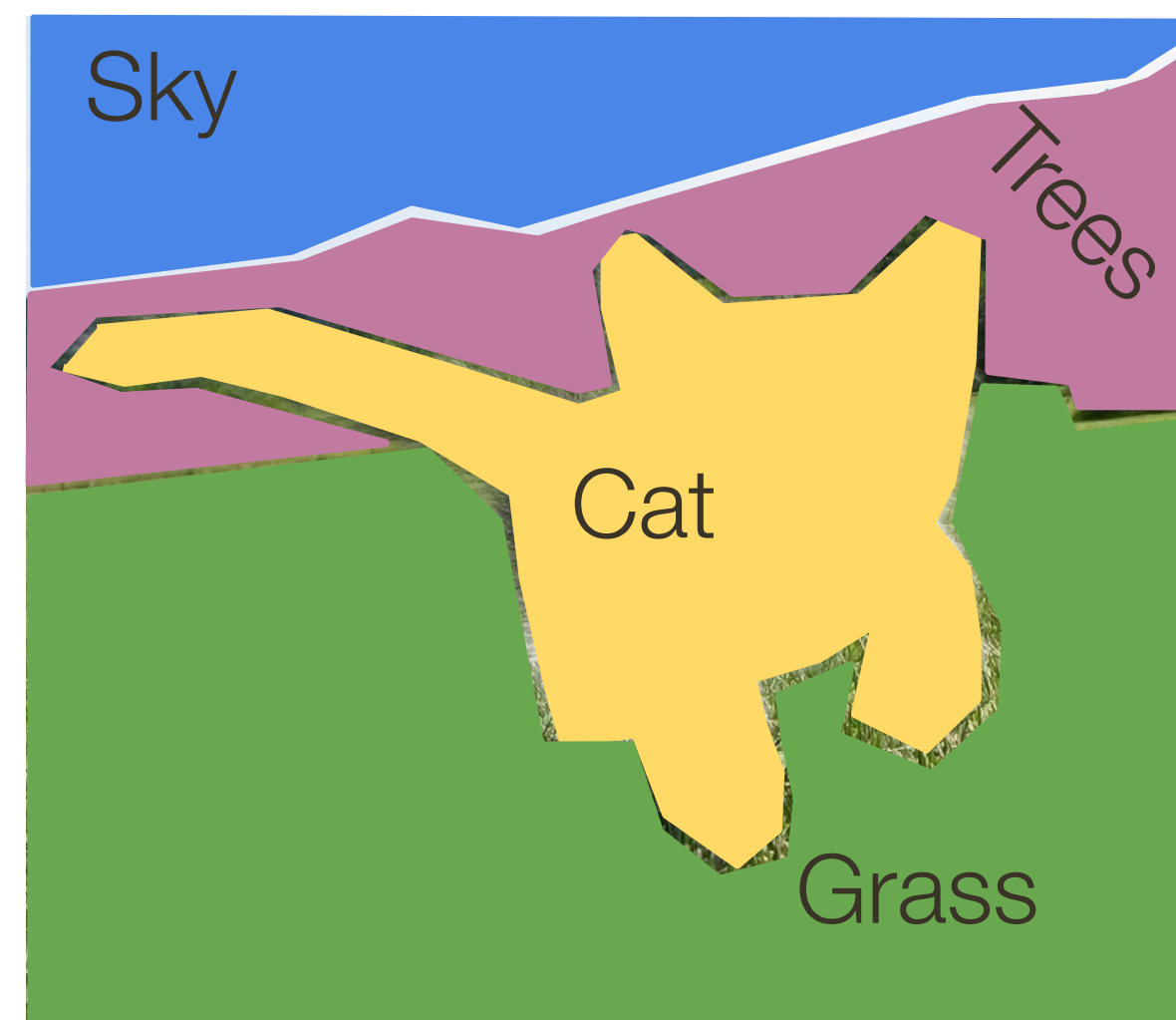


Horse
Person



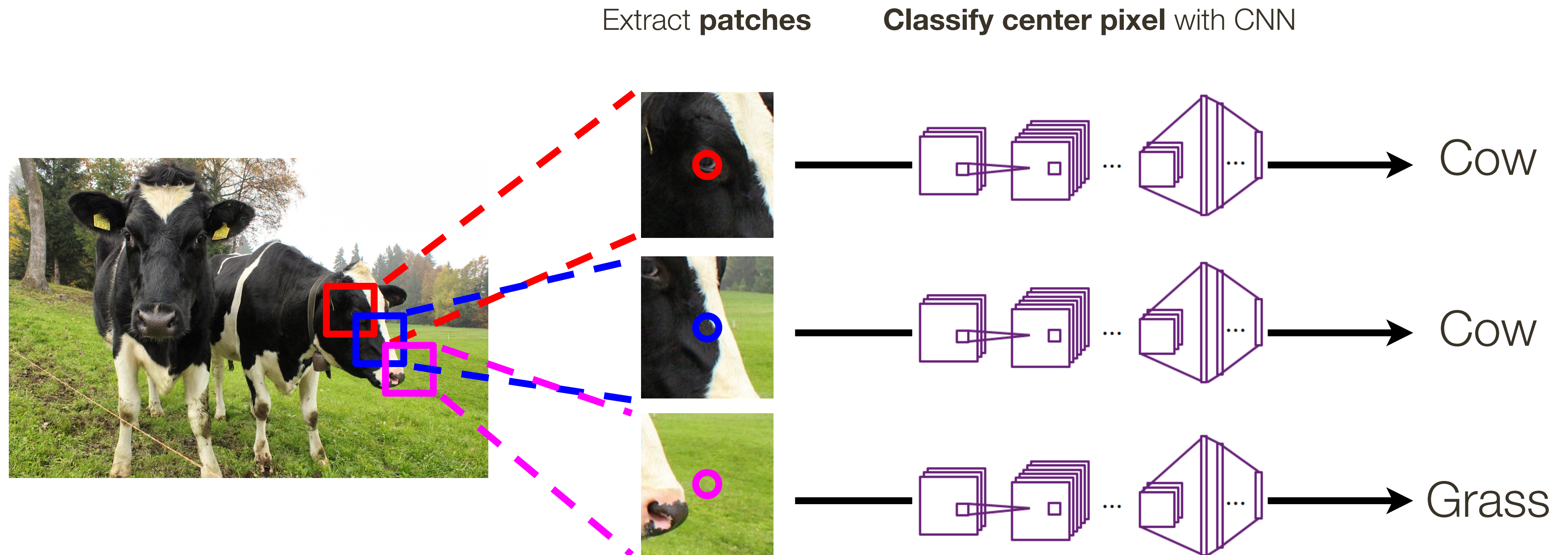
Semantic Segmentation

Label **every pixel** with a category label (without differentiating instances)



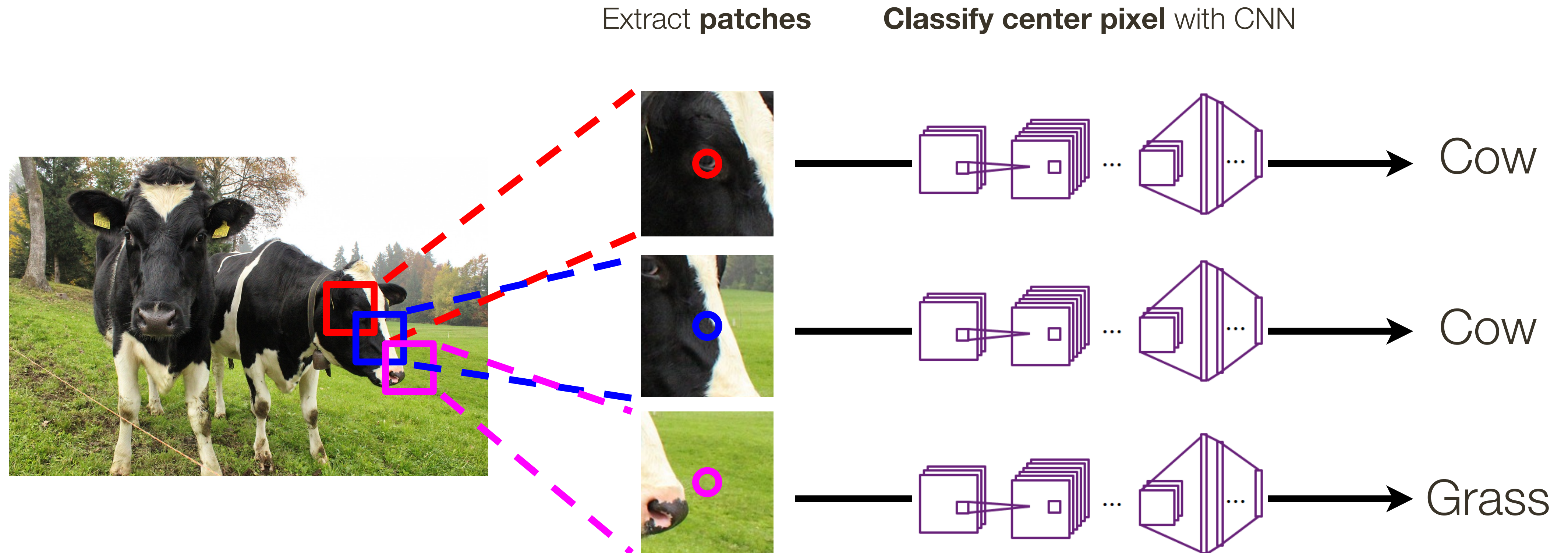
Semantic **Segmentation**: Sliding Window

[Farabet et al, TPAMI 2013]
[Pinheiro et al, ICML 2014]



Semantic Segmentation: Sliding Window

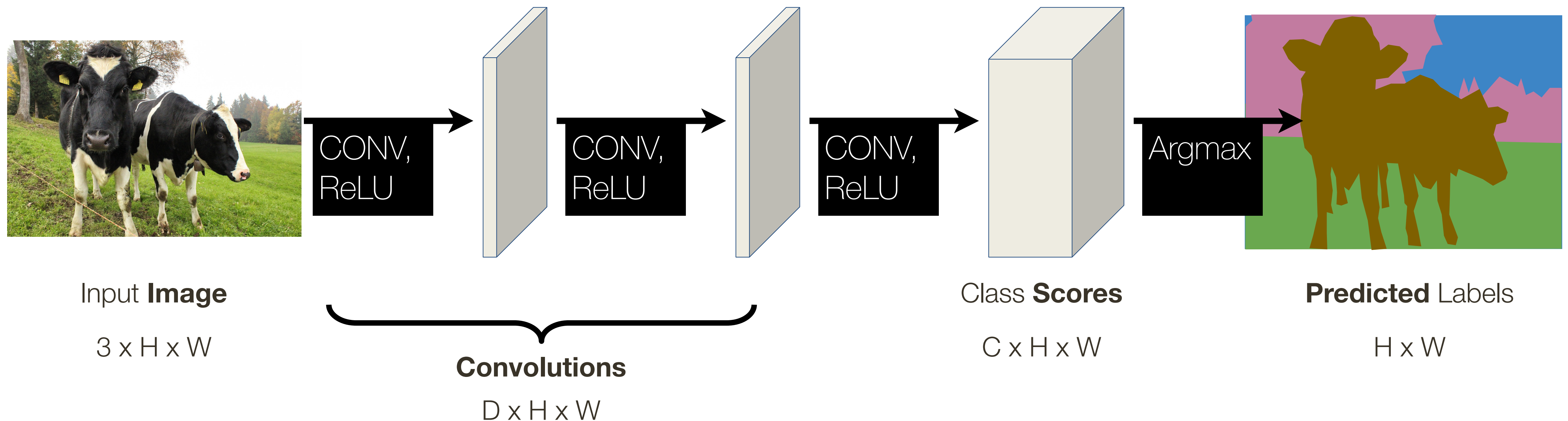
[Farabet et al, TPAMI 2013]
[Pinheiro et al, ICML 2014]



Problem: VERY inefficient, no reuse of computations for overlapping patches

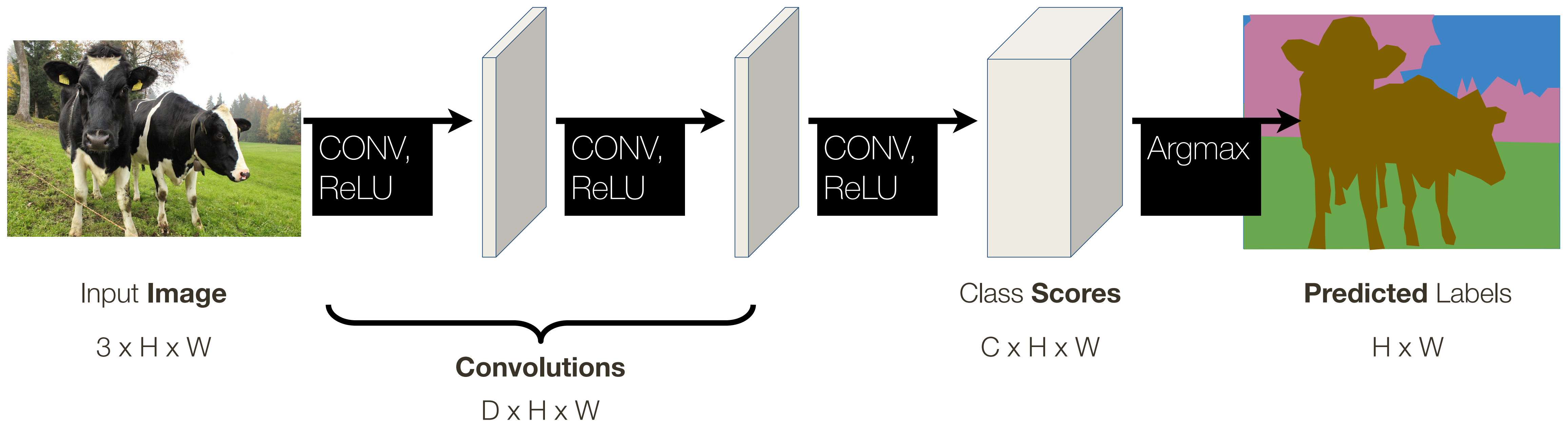
Semantic **Segmentation**: Fully Convolutional CNNs

Design a network as a number of convolutional layers to make predictions for all pixels at once!



Semantic **Segmentation**: Fully Convolutional CNNs

Design a network as a number of convolutional layers to make predictions for all pixels at once!



Problem: Convolutions at the original image scale will be very expensive

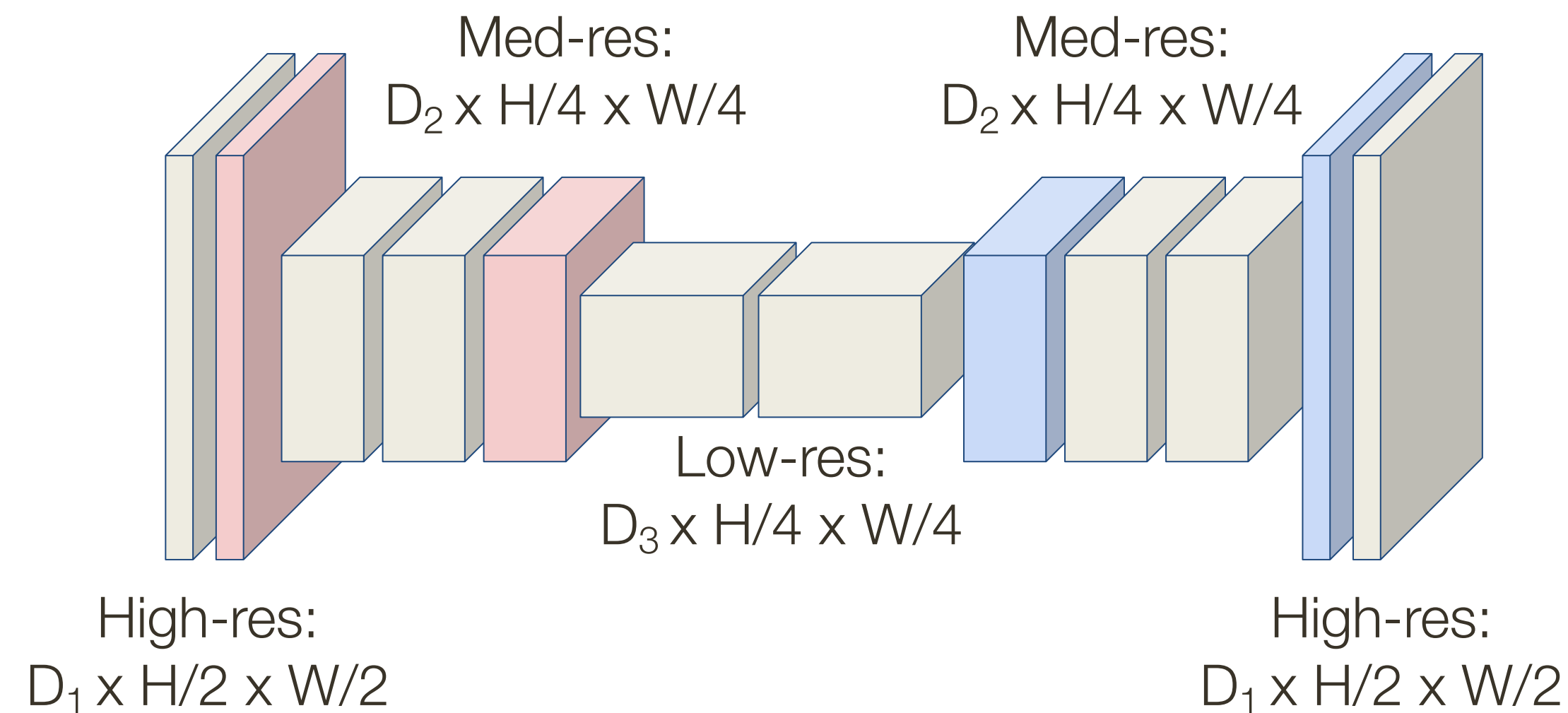
Semantic **Segmentation**: Fully Convolutional CNNs

Design a network as a number of convolutional layers with **downsampling** and **upsampling** inside the network!



Input **Image**

$3 \times H \times W$



Predicted Labels

$H \times W$

[Long et al, CVPR 2015]
[Noh et al, ICCV 2015]

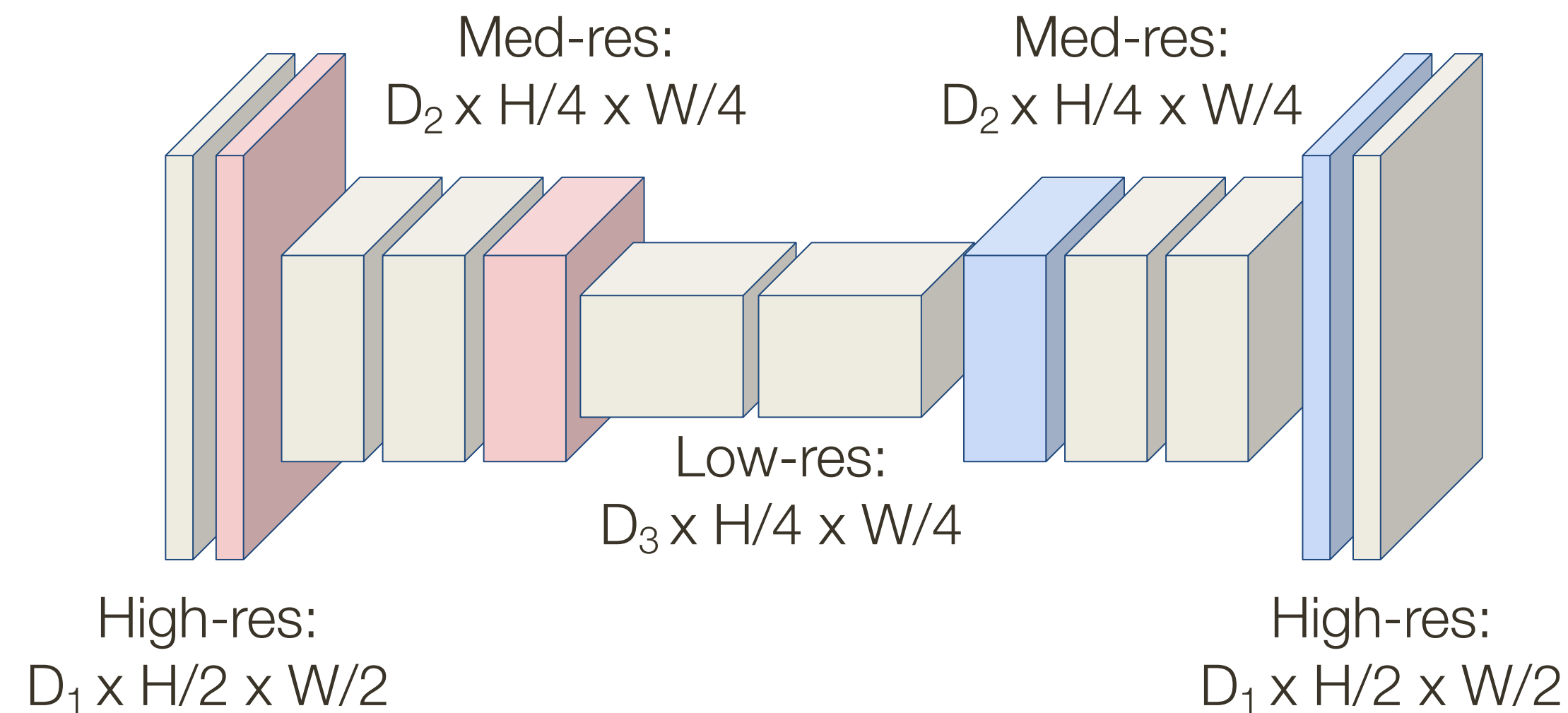
Semantic **Segmentation**: Fully Convolutional CNNs

Design a network as a number of convolutional layers with **downsampling** and **upsampling** inside the network!



Input **Image**

$3 \times H \times W$



Predicted Labels

$H \times W$

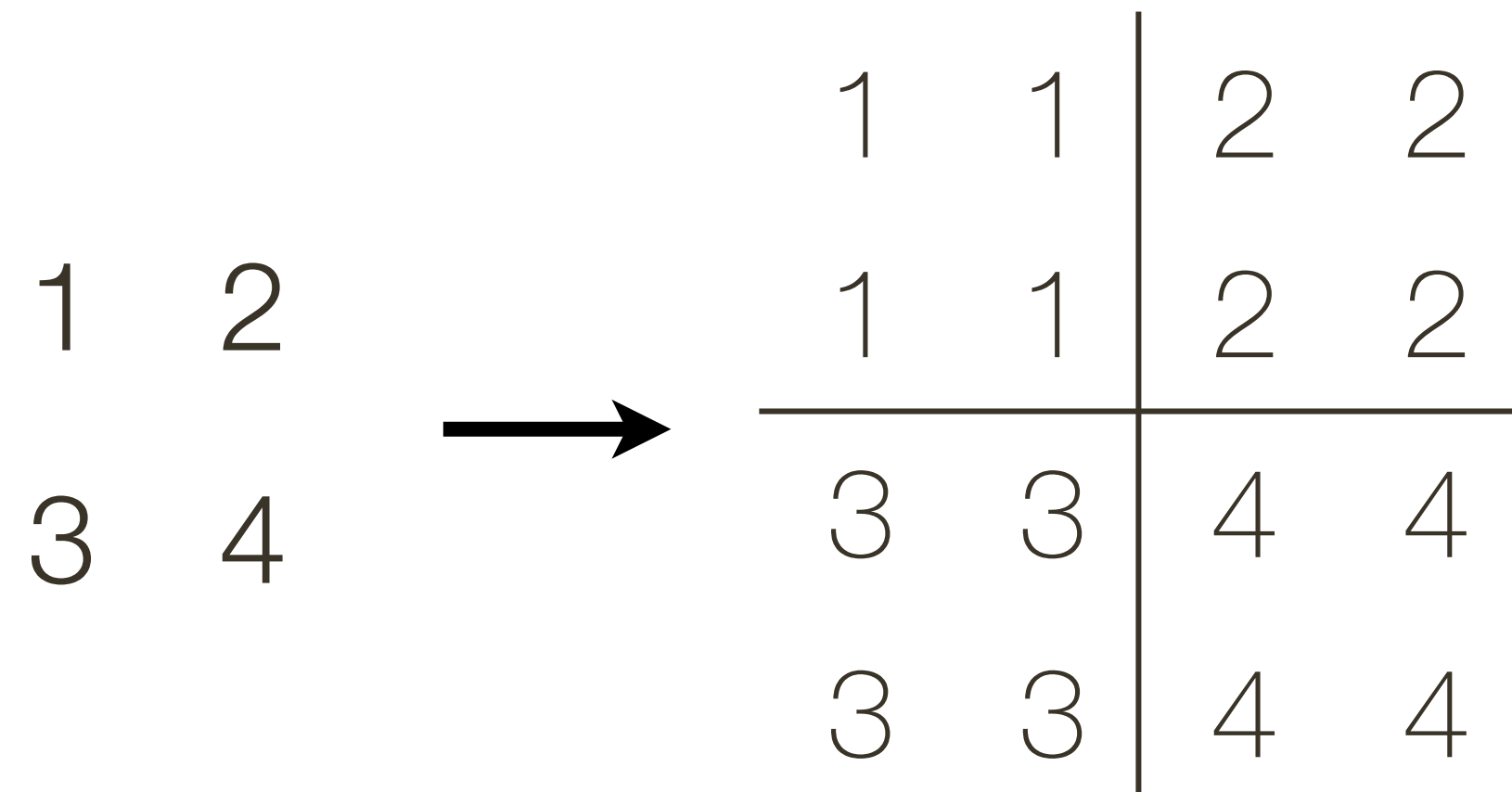
Downsampling = Pooling

Upsampling = ???

[Long et al, CVPR 2015]
[Noh et al, ICCV 2015]

In-network **Up Sampling** (a.k.a “Unpooling”)

Nearest Neighbor

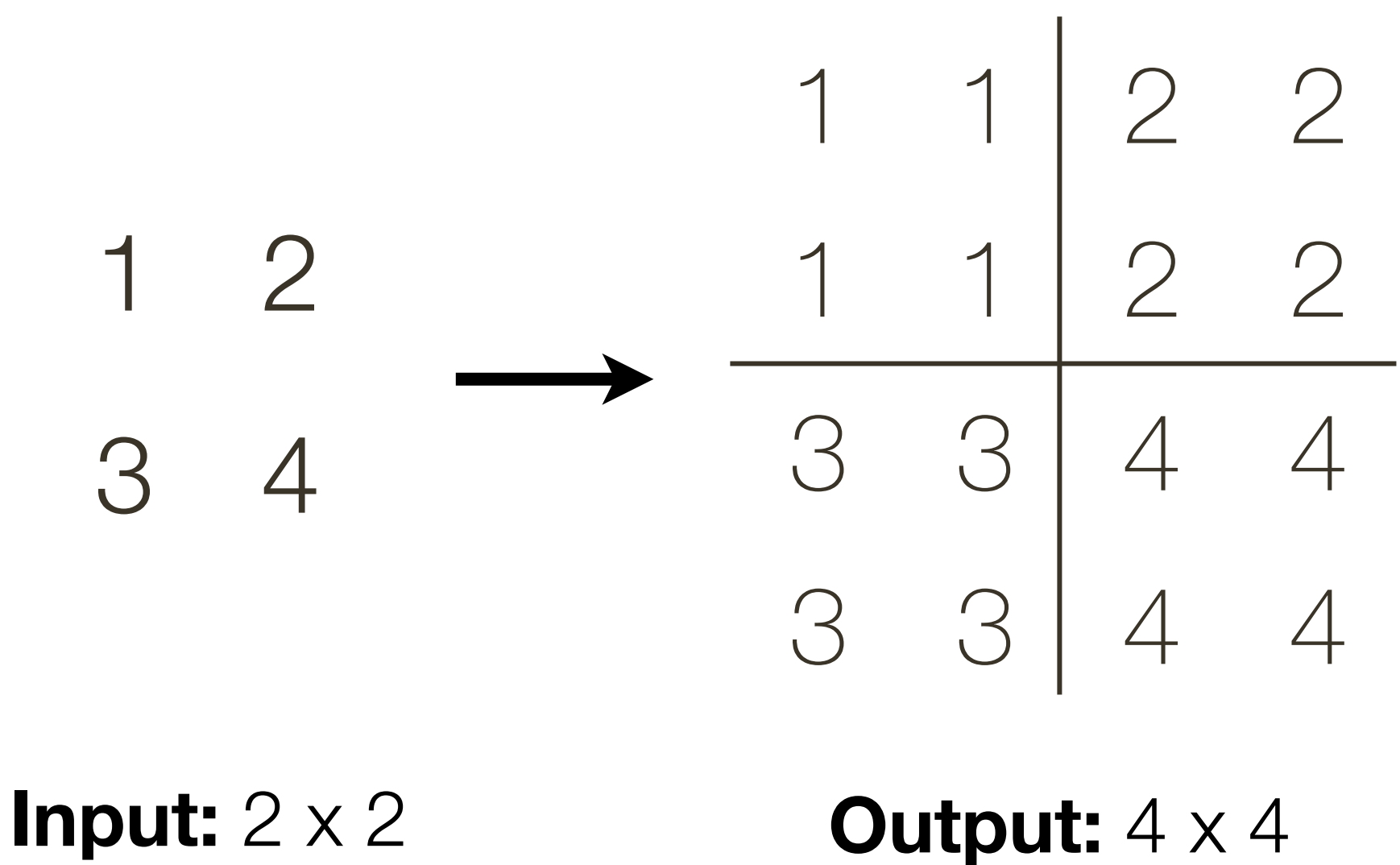


Input: 2 x 2

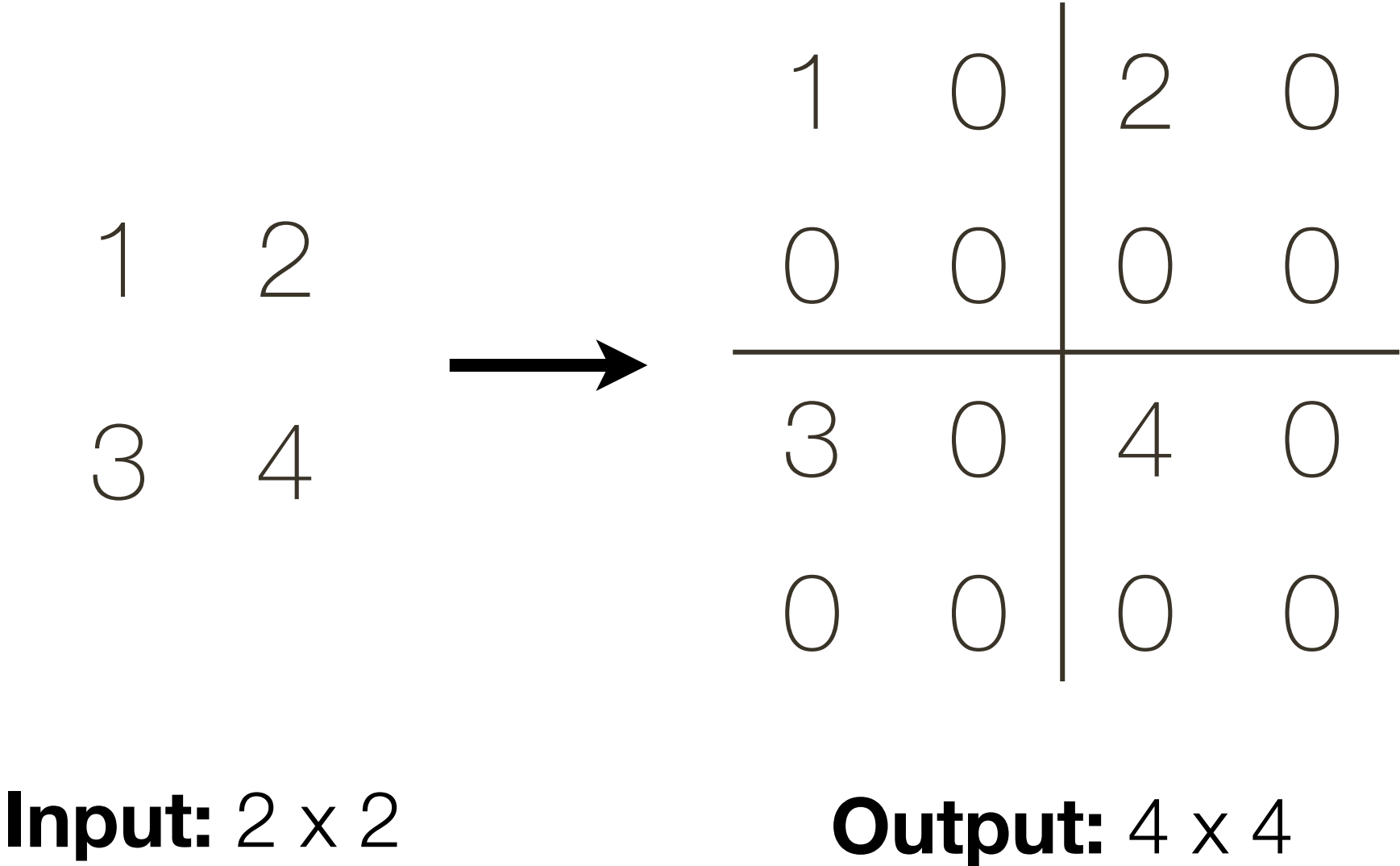
Output: 4 x 4

In-network **Up Sampling** (a.k.a “Unpooling”)

Nearest Neighbor



“Bed of Nails”



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

In-network **Up Sampling**: Max Unpooling

Max Pooling

Remember which element was max!

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

Input: 4 x 4



5	6
7	8

Output: 2 x 2



...

Rest of the network

Max Unpooling

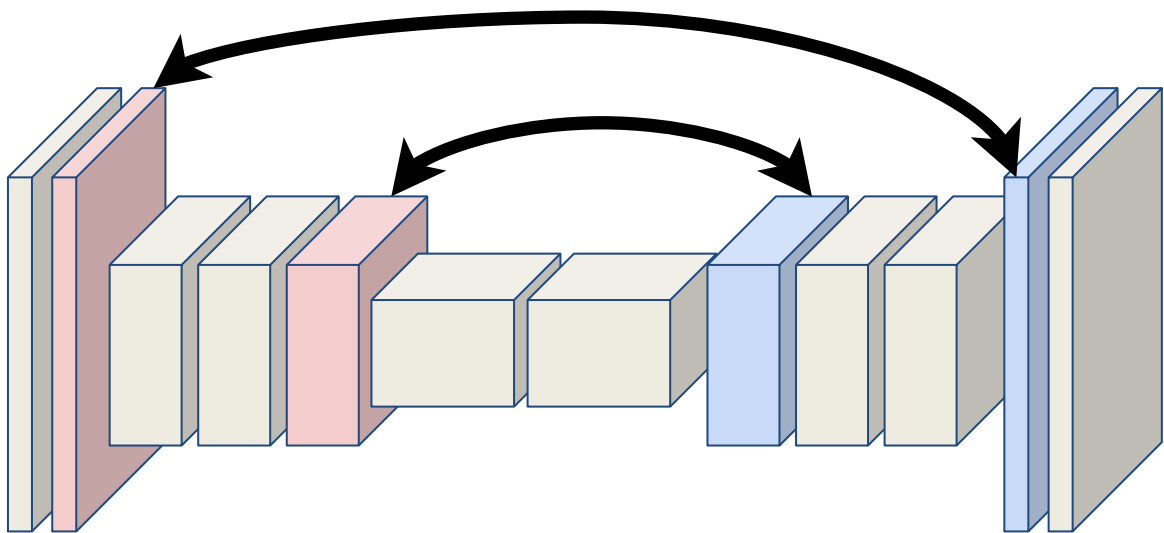
Use positions from pooling layer

1	2
3	4

Input: 2 x 2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

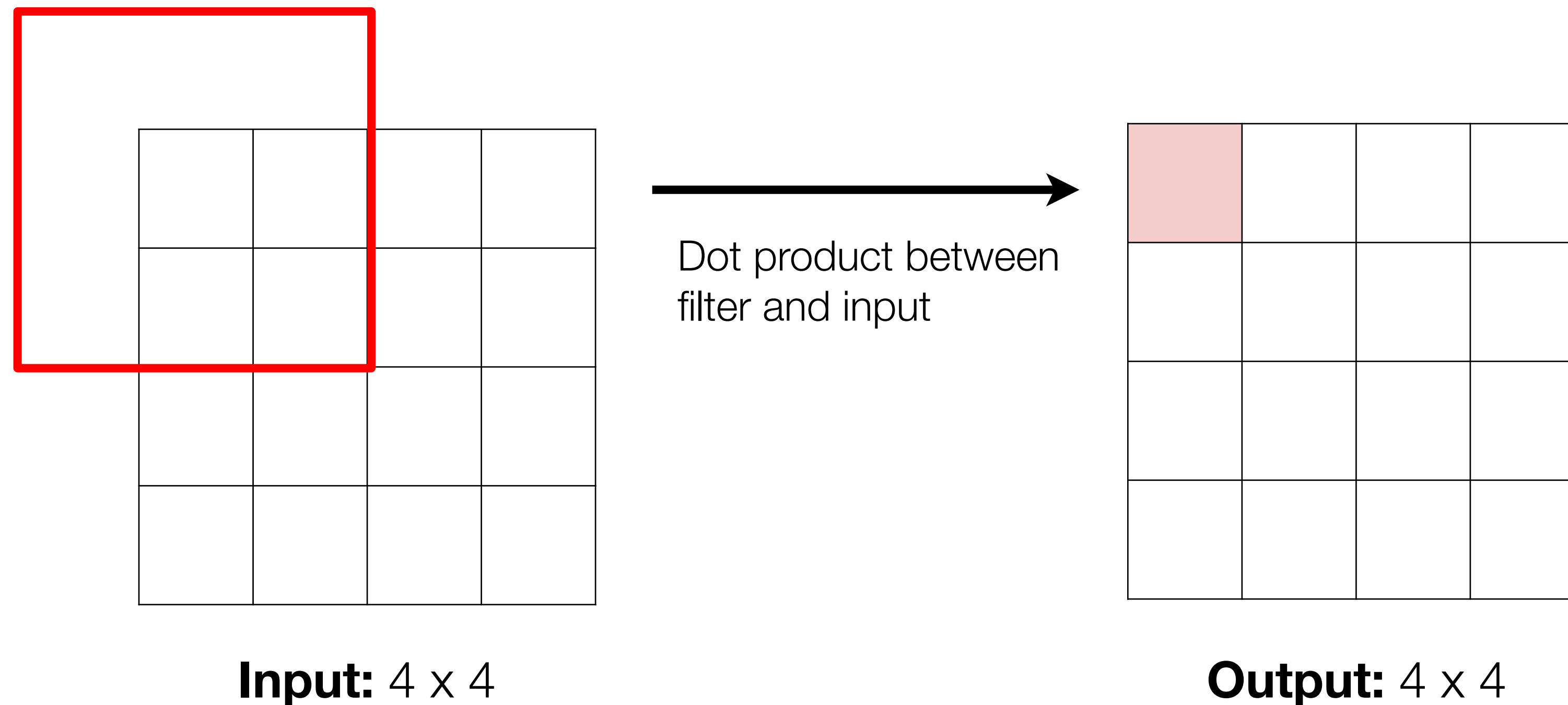
Output: 4 x 4



Corresponding pairs of downsampling and upsampling layers

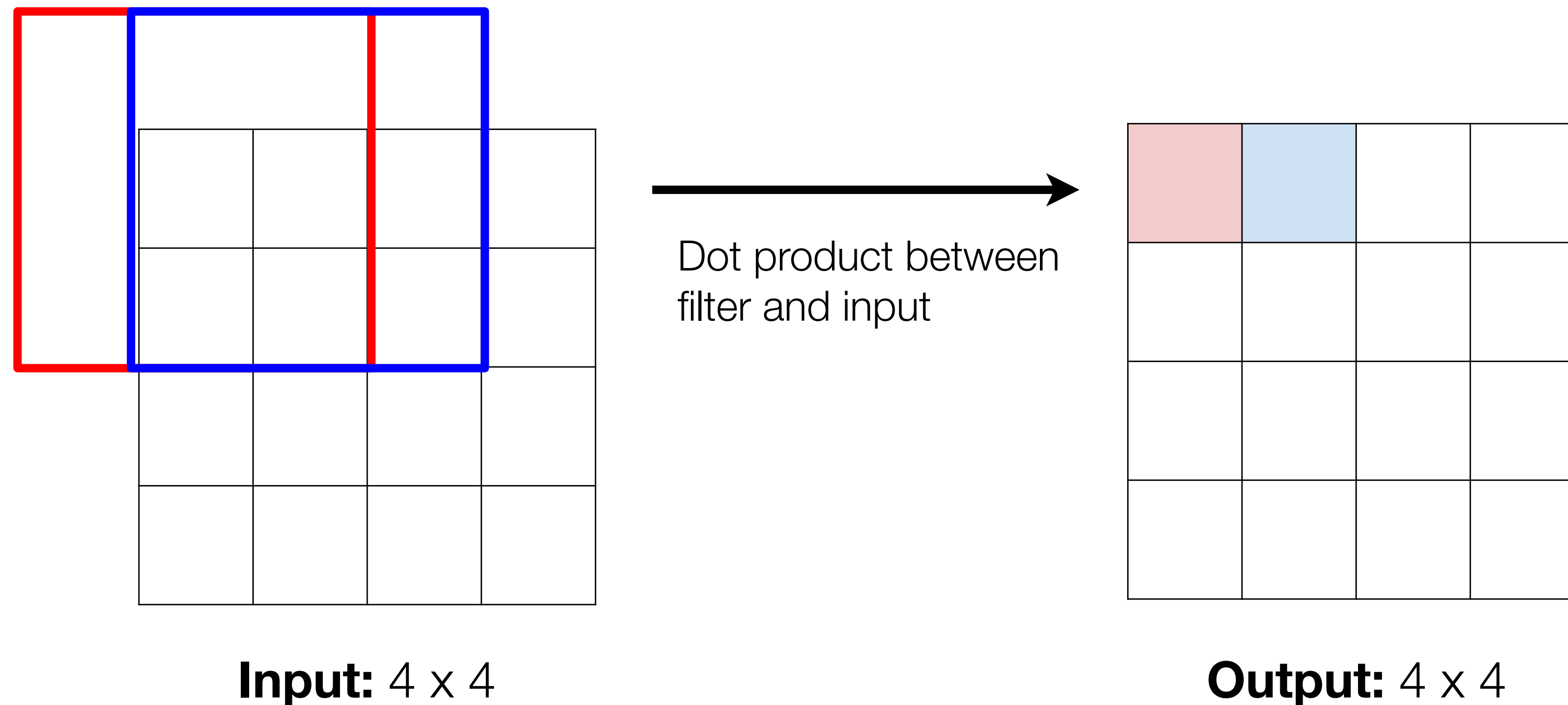
In-network **Up Sampling:** Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1



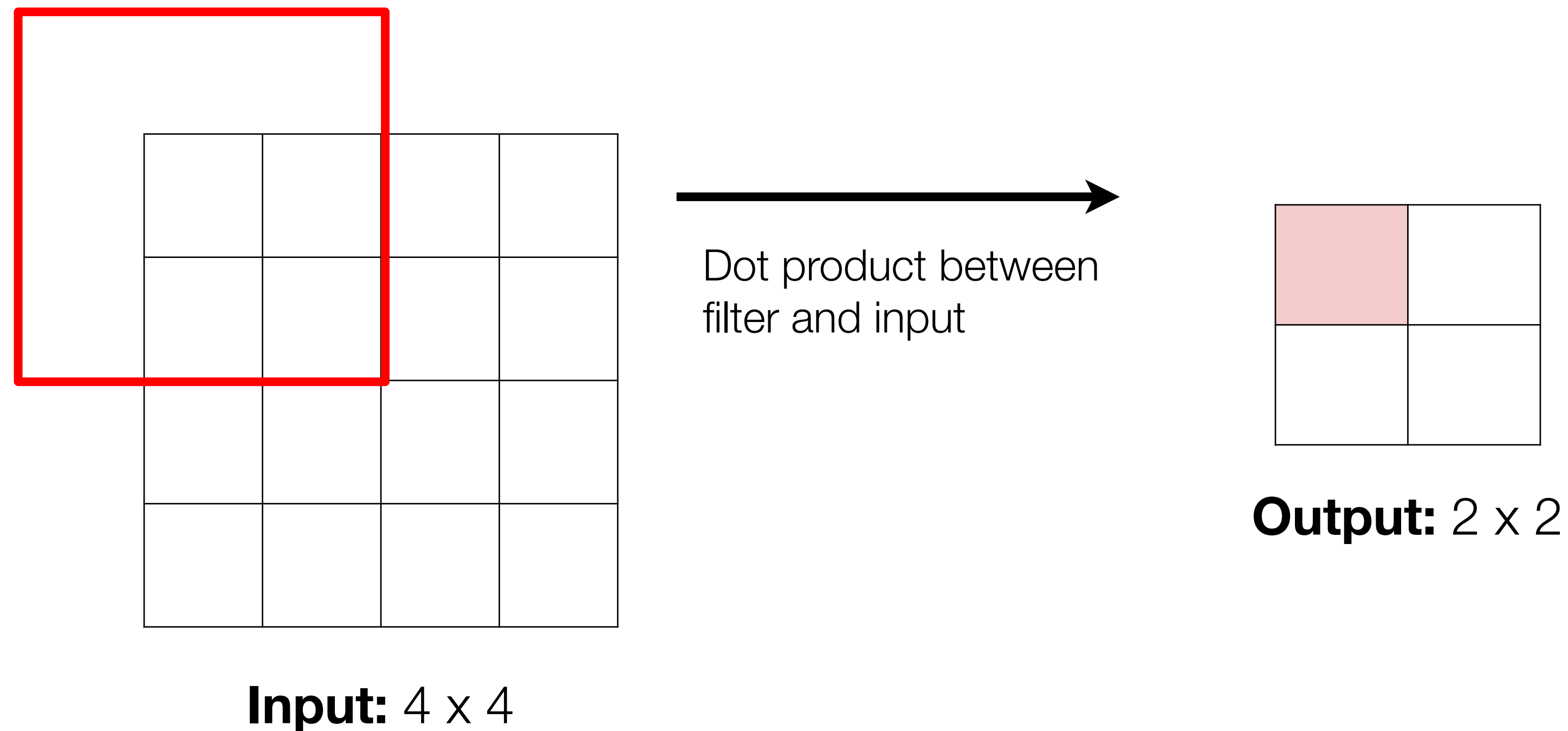
In-network **Up Sampling:** Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1



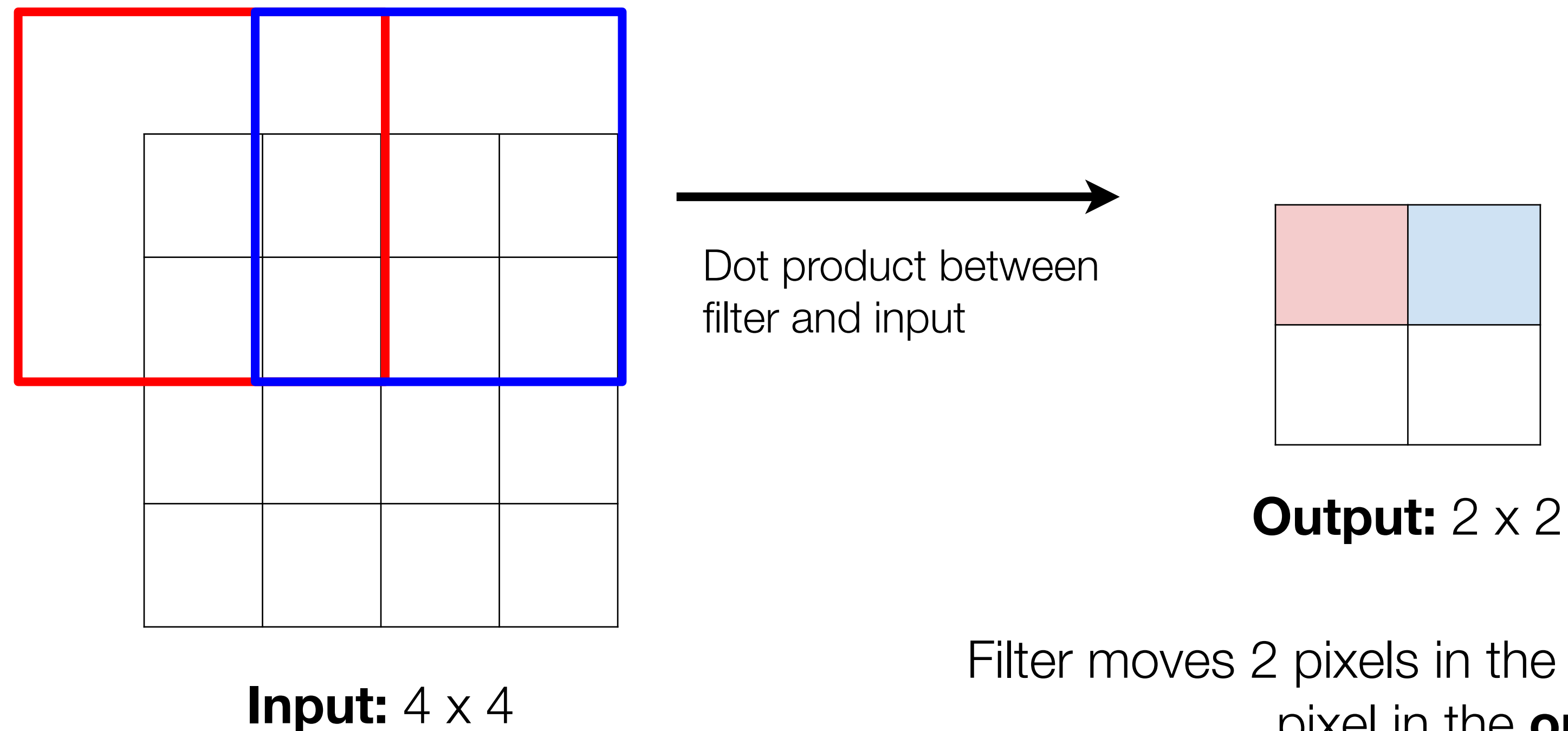
In-network **Up Sampling:** Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



In-network **Up Sampling:** Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Filter moves 2 pixels in the **input** for every one pixel in the **output**

Stride gives ratio in movement in input vs output

In-network **Up Sampling**: Transpose Convolution

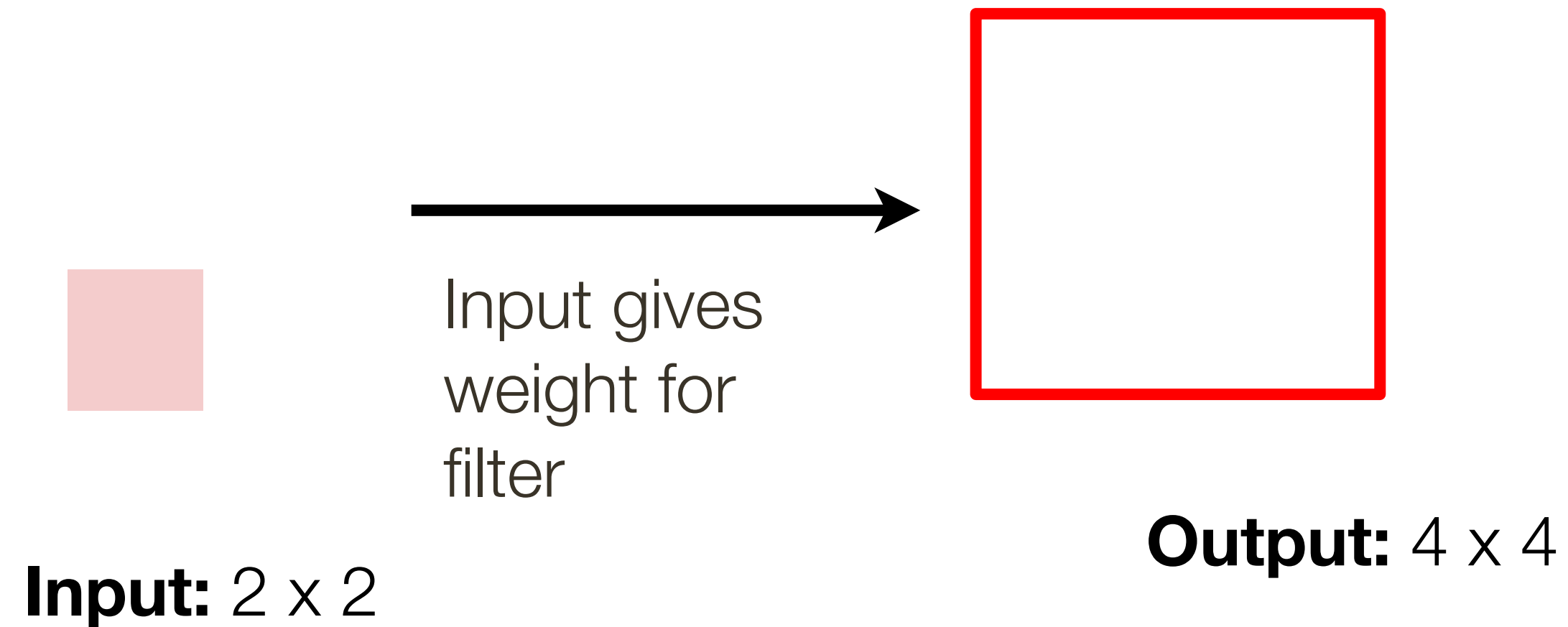
3 x 3 **transpose** convolution, stride 2 pad 1

Input: 2 x 2

Output: 4 x 4

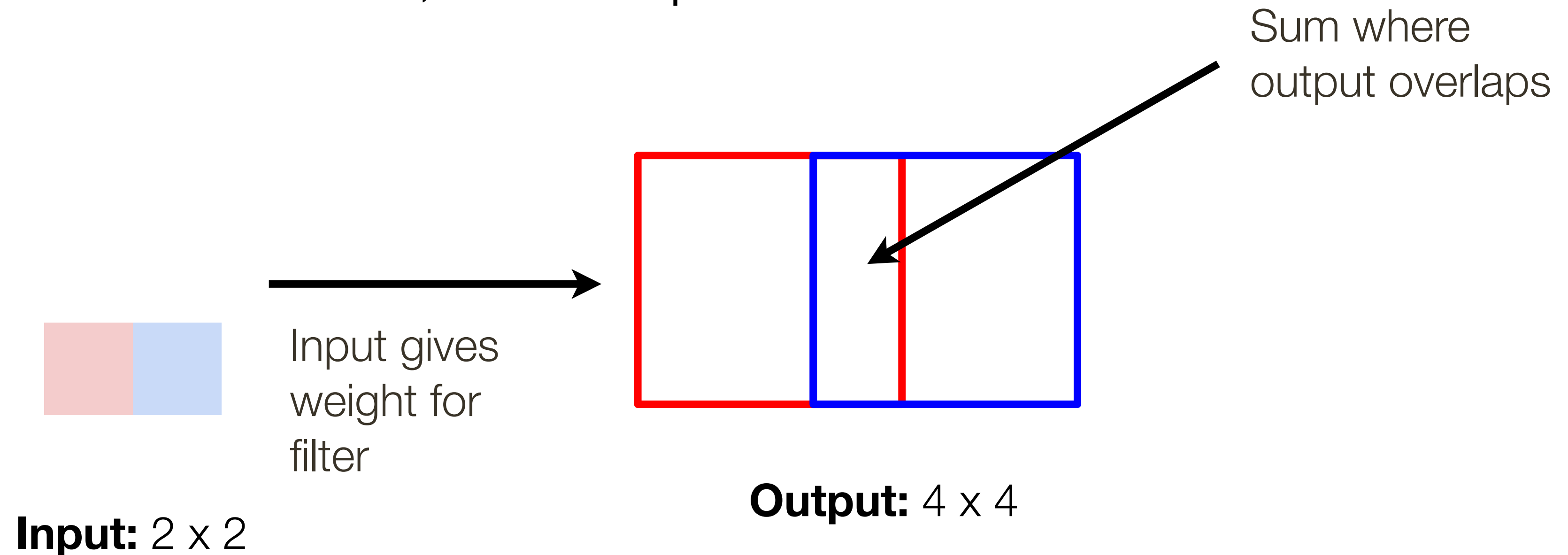
In-network **Up Sampling**: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



In-network **Up Sampling**: Transpose Convolution

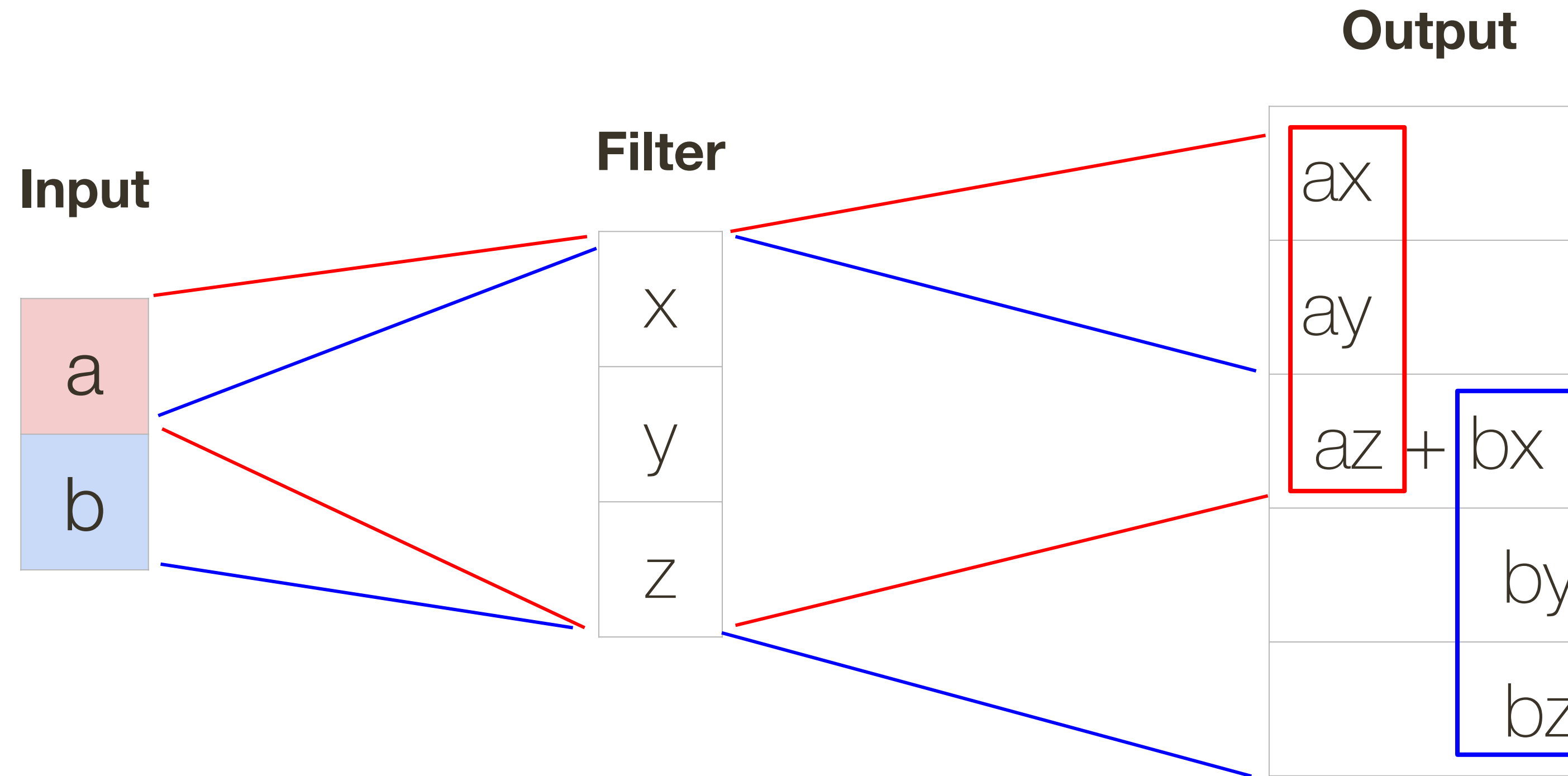
3 x 3 **transpose** convolution, stride 2 pad 1



Filter moves 2 pixels in the **output** for every one pixel in the **input**

Stride gives ratio in movement in output vs input

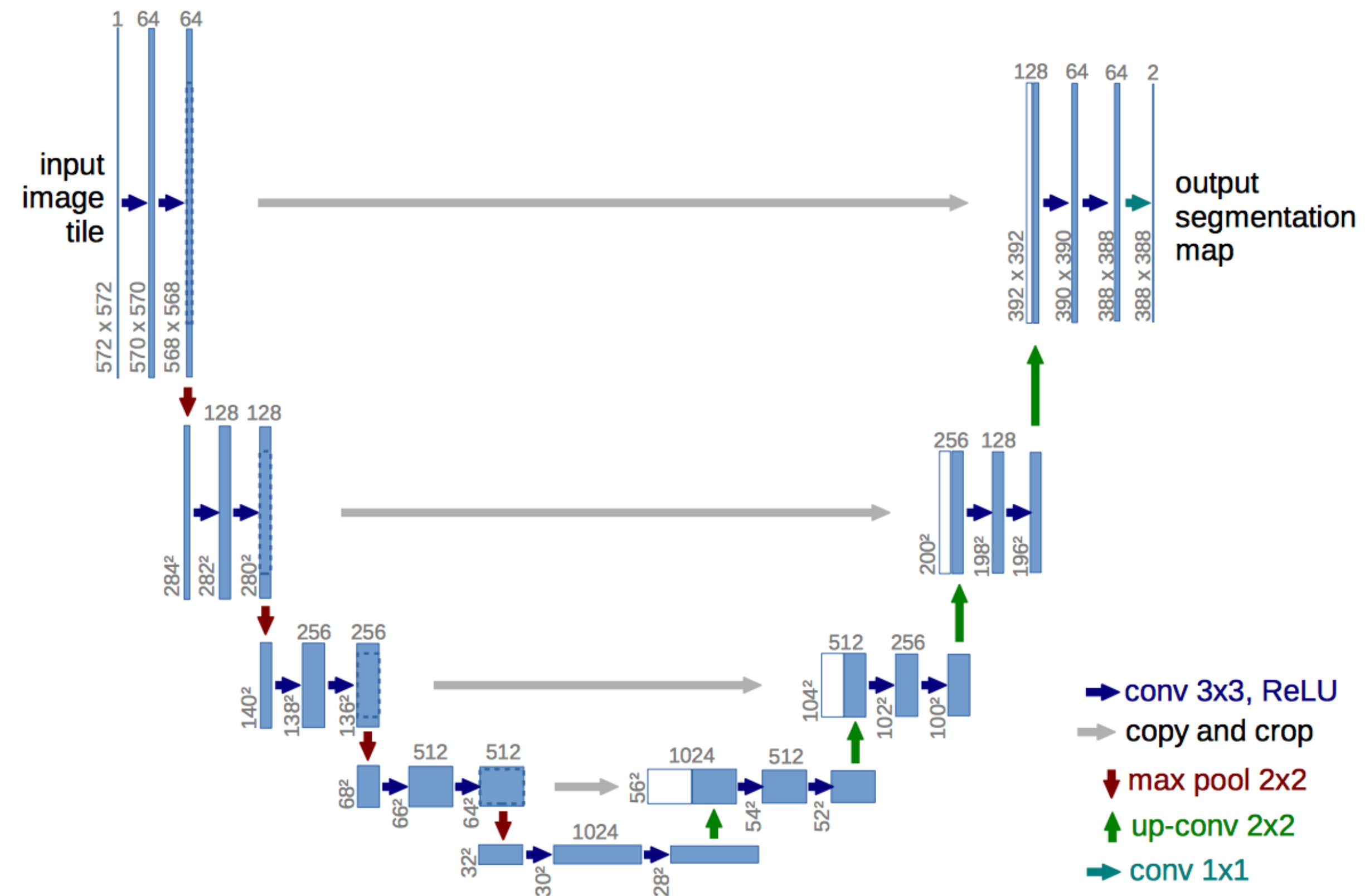
Transpose Convolution: 1-D Example



Output contains copies of the filter weighted multiplied by the input, summing at overlaps in the output

U-Net Architecture

ResNet-like Fully convolutional CNN



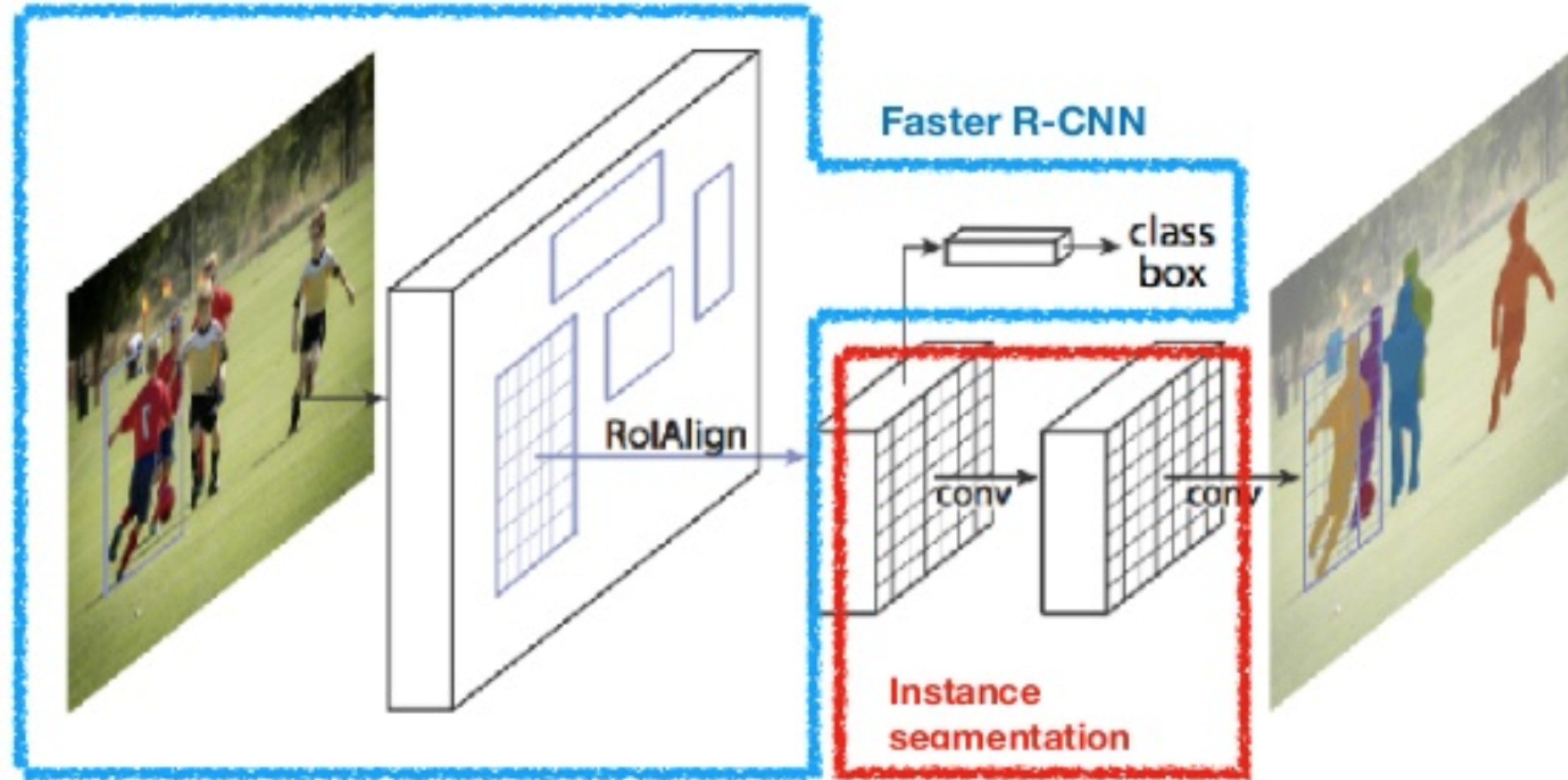
Computer **Vision Problems** (no language for now)

Instance Segmentation



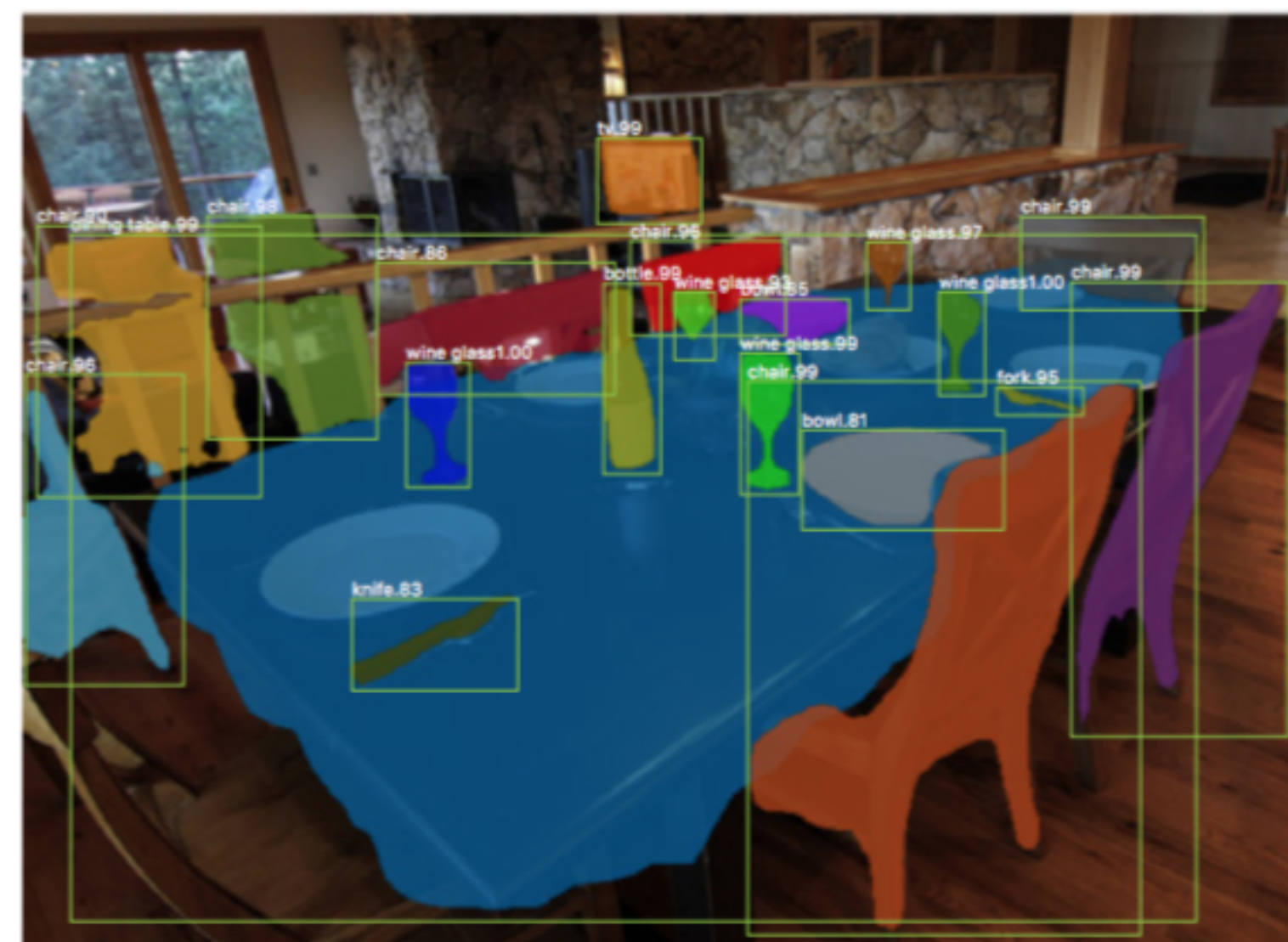
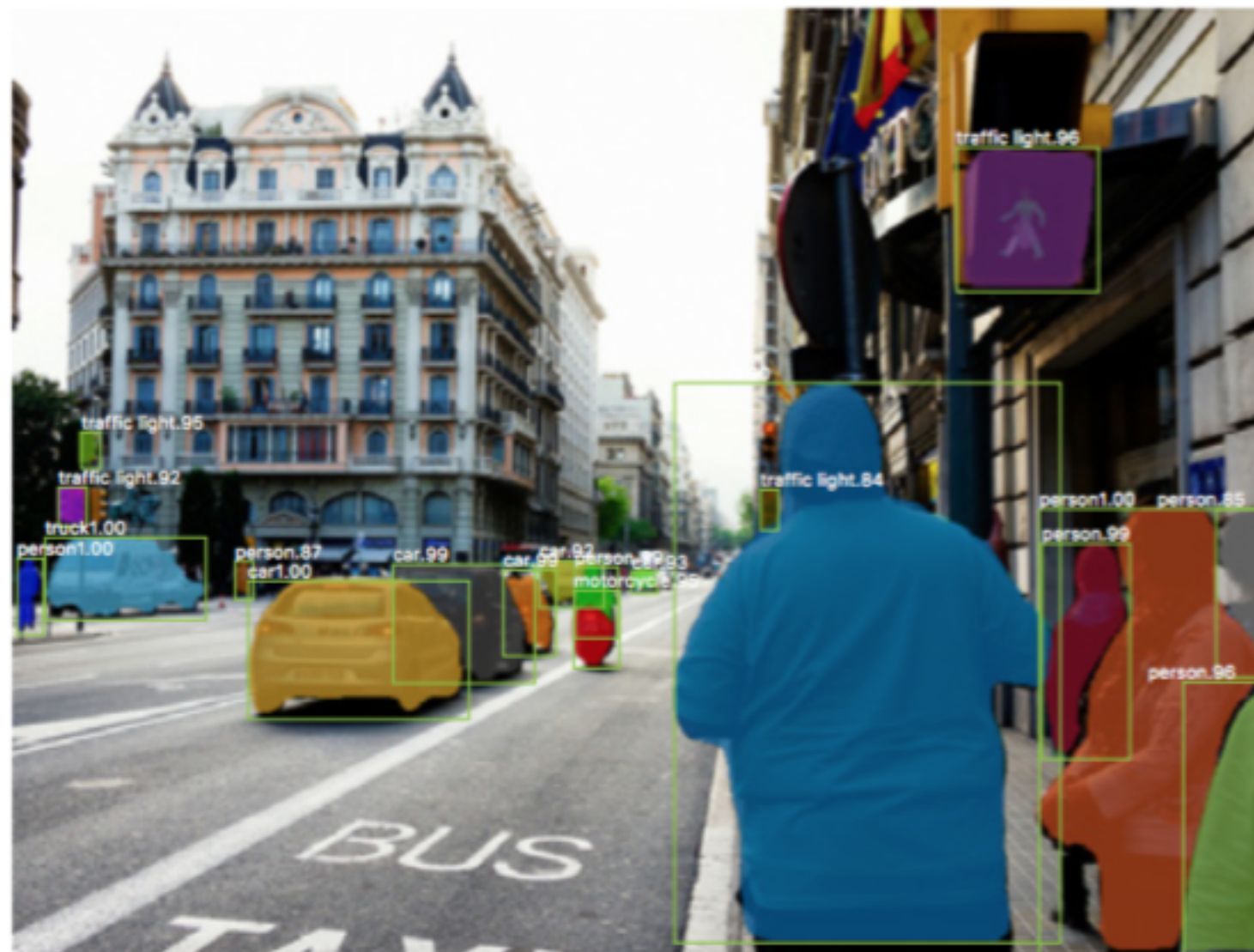
Horse1
Horse2
Person1
Person2

Mask R-CNN



[He et al, 2017]

Mask R-CNN



[He et al, 2017]

Summary

Common types of layers:

1. **Convolutional** Layer
 - Parameters define a set of learnable filters
2. **Pooling** Layer
 - Performs a downsampling along the spatial dimensions
3. **Fully-Connected** Layer
 - As in a regular neural network

Each layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function

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)