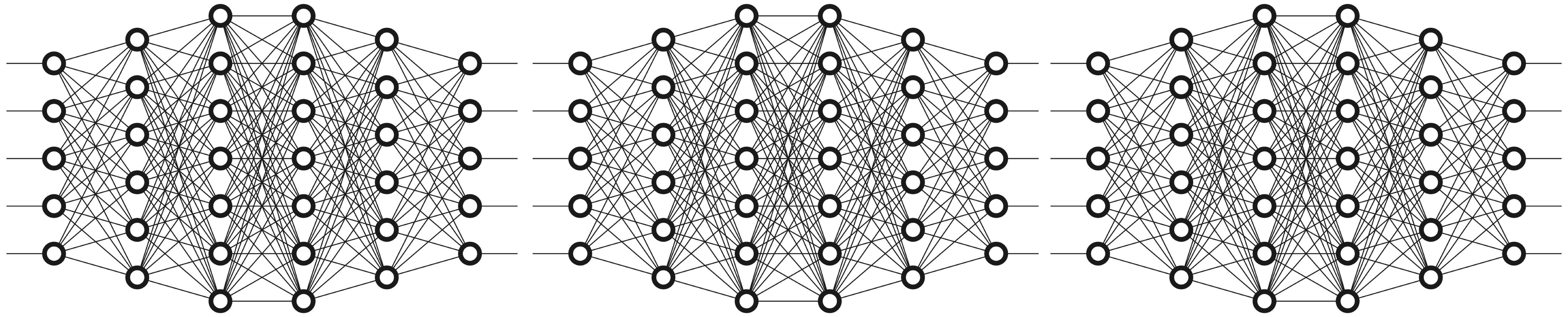
# CPSC 425: Computer Vision



**Lecture 24:** Neural Nets and CNNs (putting it all together)

# **Menu** for Today (**April 7, 2020**)

**Reminders:**

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

— **On-line quiz** due end of the day **today**

— **Material** for **Final Prep** will is available on Canvas (will post Quizzes, Midterm)

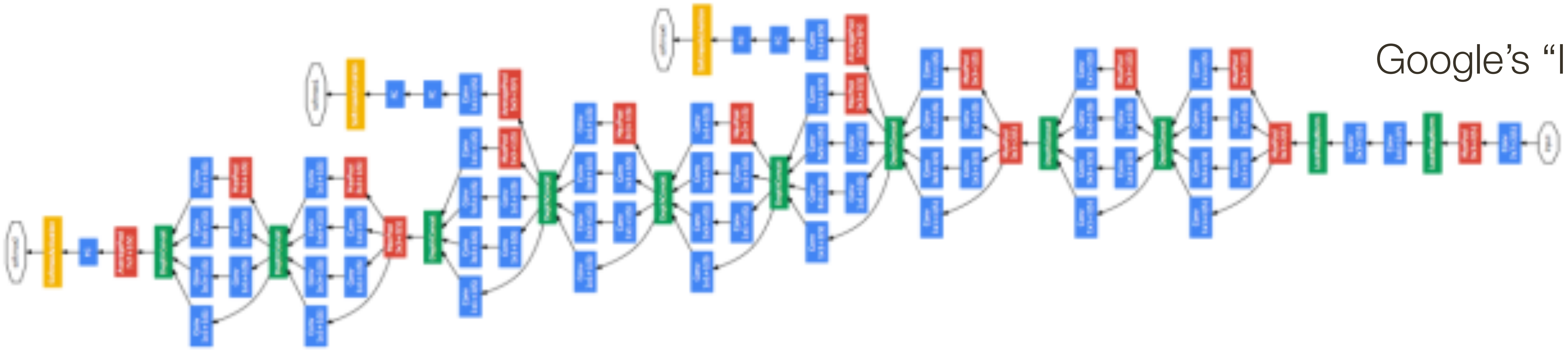
— Will post Final Prep **office hours** today/tomorrow

Please fill out

**Student Evaluations**

(on Canvas)

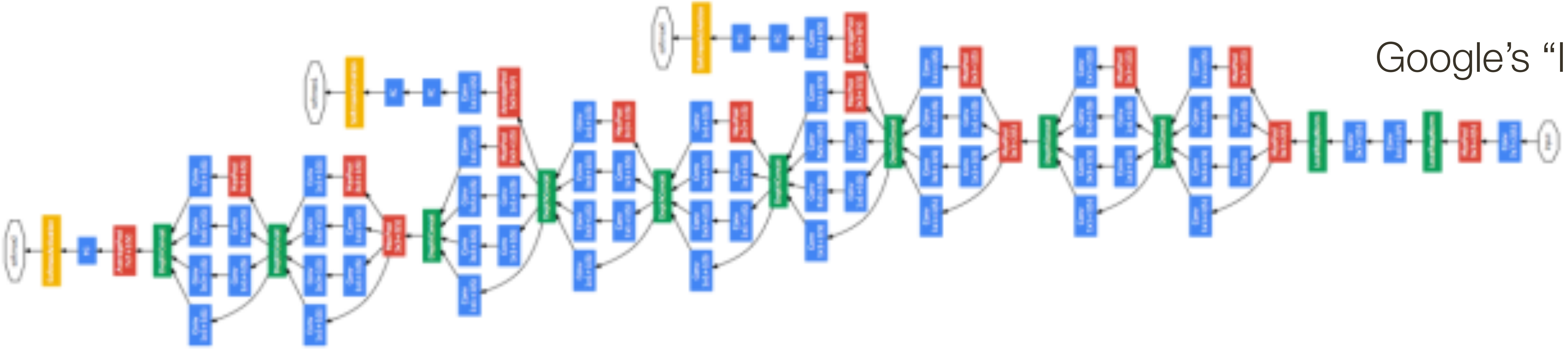
# Deep Learning **Terminology**



Google's "Inception" network

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

# Deep Learning **Terminology**

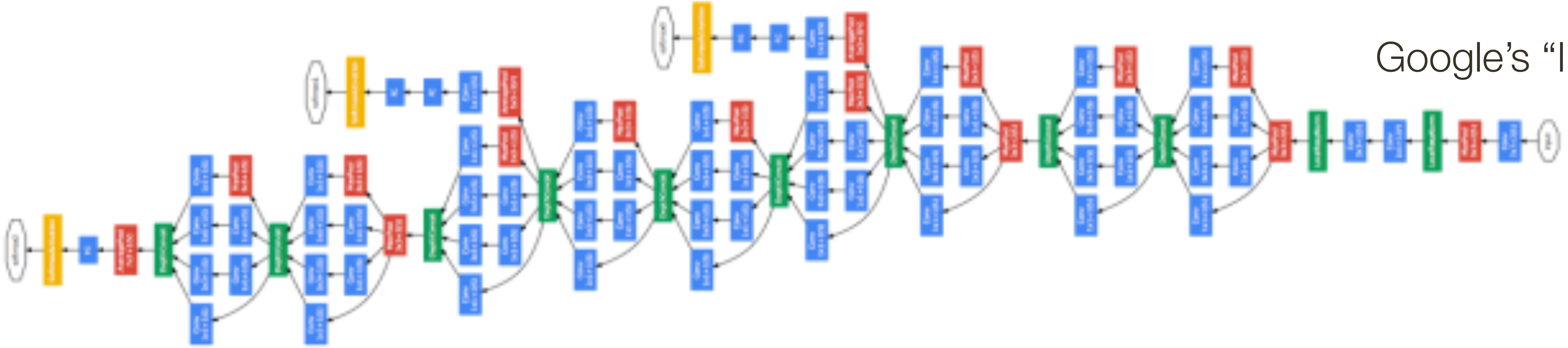


Google's "Inception" network

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

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

# Deep Learning **Terminology**



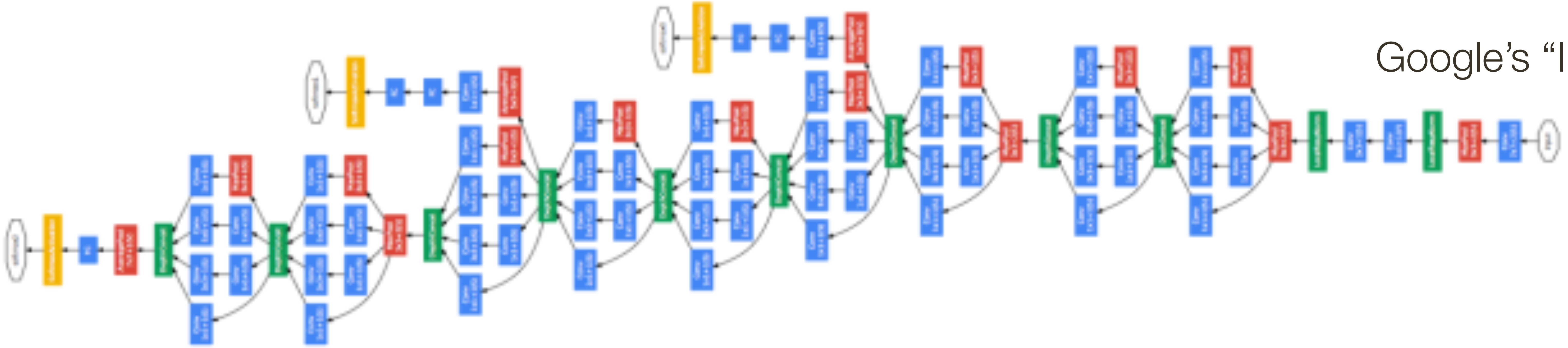
Google's "Inception" network

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

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

deeper = better

# Deep Learning **Terminology**



Google's "Inception" network

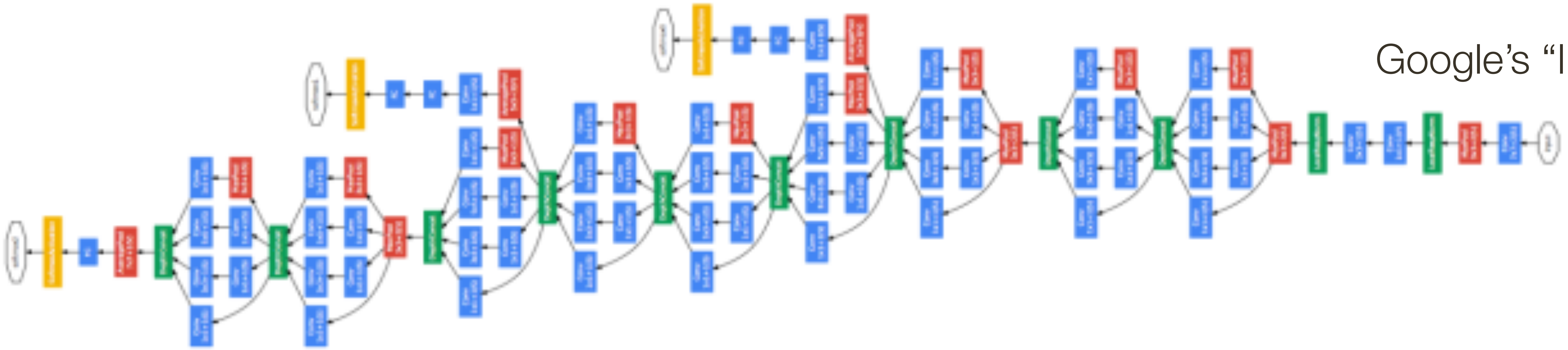
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

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

deeper = better

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

# Deep Learning **Terminology**



Google's "Inception" network

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

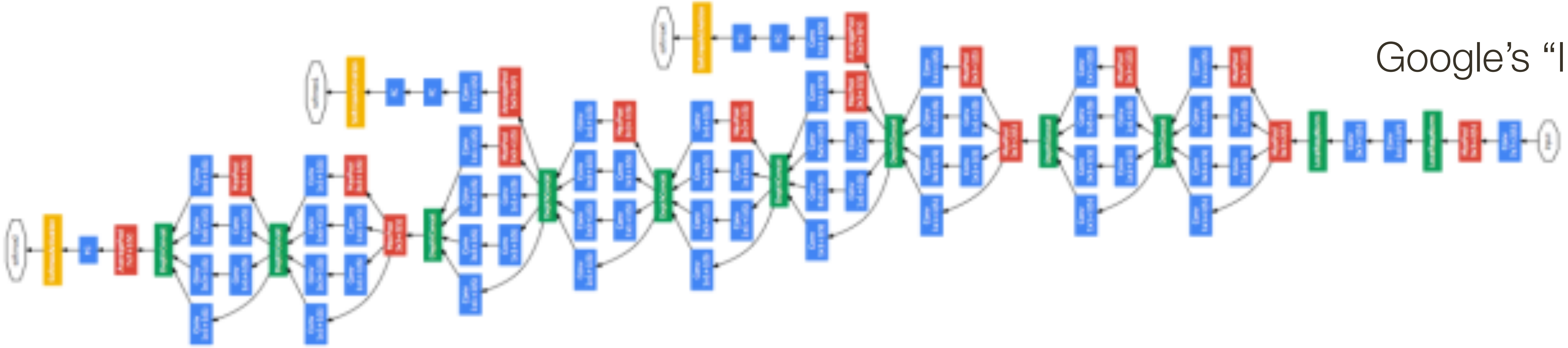
generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

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

requires knowledge of the nature of the problem

# Deep Learning **Terminology**



Google's "Inception" network

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

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

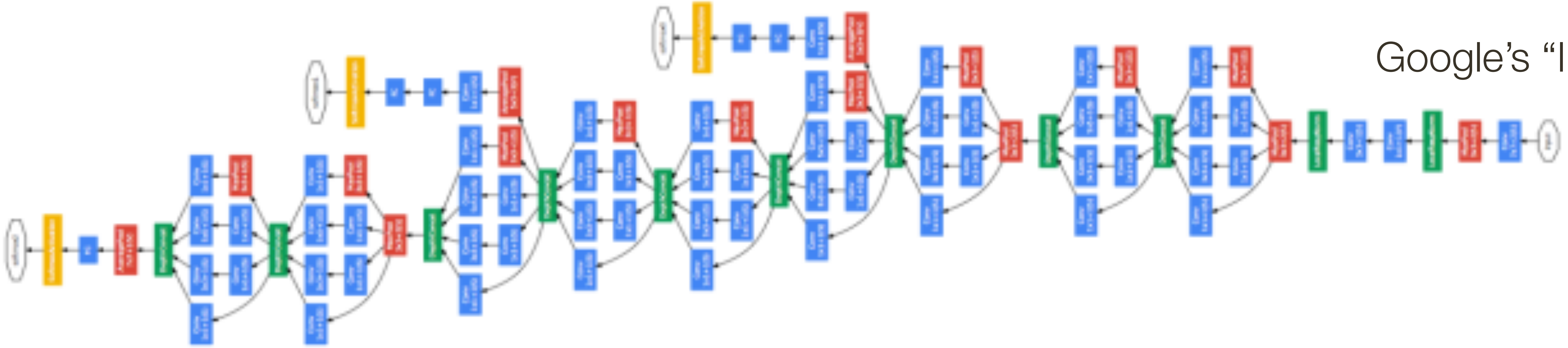
deeper = better

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

requires knowledge of the nature of the problem

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

# Deep Learning **Terminology**



Google's "Inception" network

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

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

deeper = better

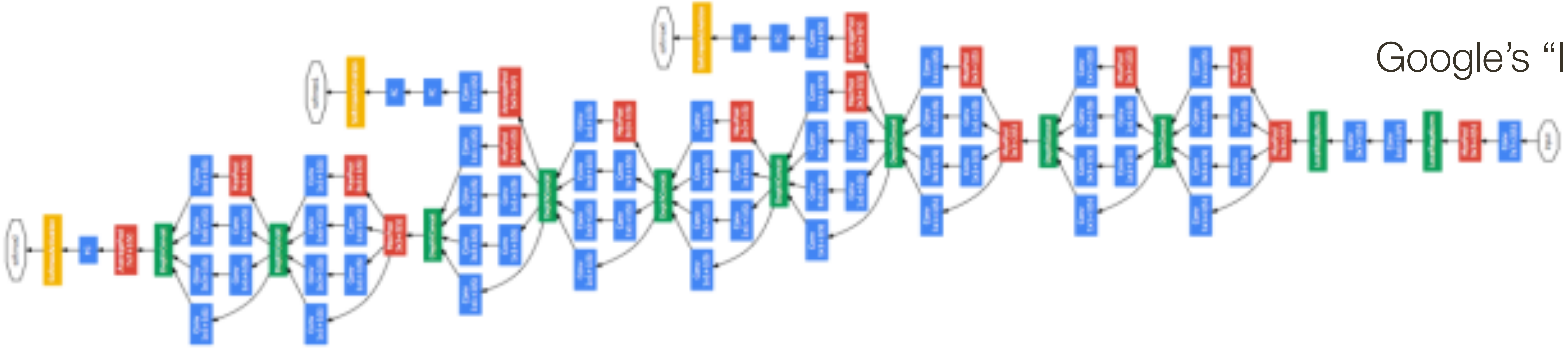
- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

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

optimized using SGD or variants

# Deep Learning **Terminology**



Google's "Inception" network

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

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

deeper = better

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

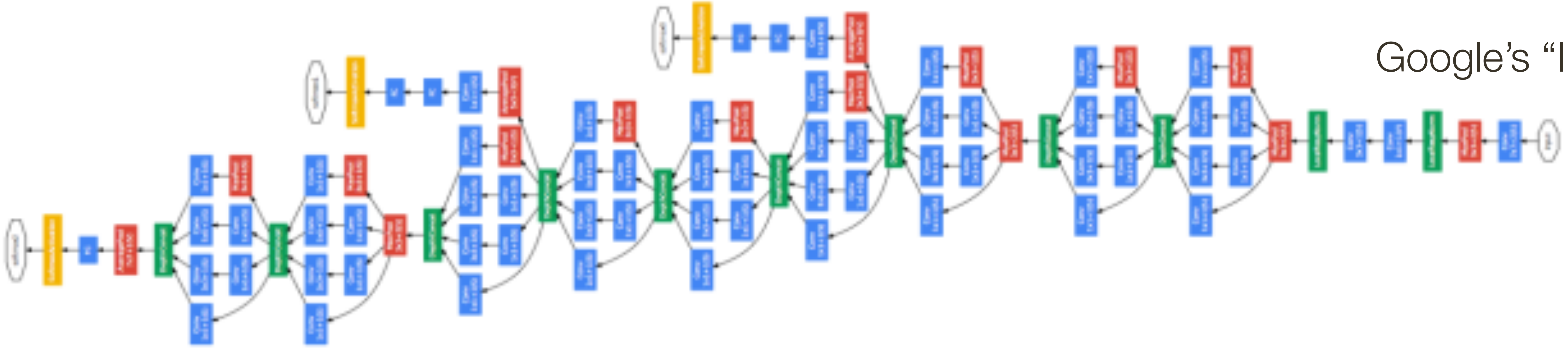
requires knowledge of the nature of the problem

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

optimized using SGD or variants

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

# Deep Learning **Terminology**



Google's "Inception" network

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

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

deeper = better

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

requires knowledge of the nature of the problem

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

optimized using SGD or variants

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

grid search

# Multivariate **Regression**

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

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

# Multivariate **Regression**

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

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

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

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

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

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

# Multivariate **Regression**

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

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

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

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

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

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

**Neural Network** (output): linear layer

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

# Multivariate **Regression**

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

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

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

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

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

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

**Neural Network** (output): linear layer

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

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

# **Binary** Classification (Bernoulli)

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

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

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

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

# **Binary** Classification (Bernoulli)

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

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

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

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

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

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

# **Binary** Classification (Bernoulli)

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

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

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

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

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

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

**Problem**: Not differentiable, probabilistic interpretation maybe desirable

# **Binary** Classification (Bernoulli)

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

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

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

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

**Neural Network** (output): interpret sigmoid output as probability

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

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

# **Binary** Classification (Bernoulli)

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

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

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

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

**Neural Network** (output): interpret sigmoid output as probability

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

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

**Loss:** similarity between two distributions

# **Binary** Classification (Bernoulli)

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

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

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

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

**Neural Network** (output): interpret sigmoid output as probability

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

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

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

# **Binary** Classification (Bernoulli)

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

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

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

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

**Neural Network** (output): interpret sigmoid output as probability

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

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

**Loss:**

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

# **Binary** Classification (Bernoulli)

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

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

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

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

**Neural Network** (output): interpret sigmoid output as probability

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

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

**Loss:**

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

# **Binary** Classification (Bernoulli)

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

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

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

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

# **Binary** Classification (Bernoulli)

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

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

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

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

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

# **Multiclass** Classification (e.g, ImageNet)

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

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

(**one-hot** encoding)

# **Multiclass** Classification (e.g, ImageNet)

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

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

(**one-hot** encoding)

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

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

# **Multiclass** Classification (e.g, ImageNet)

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

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

(**one-hot** encoding)

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

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

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

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

# **Multiclass** Classification (e.g, ImageNet)

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

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

(**one-hot** encoding)

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

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

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

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

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

# **Multiclass** Classification (e.g, ImageNet)

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

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

(**one-hot** encoding)

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

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

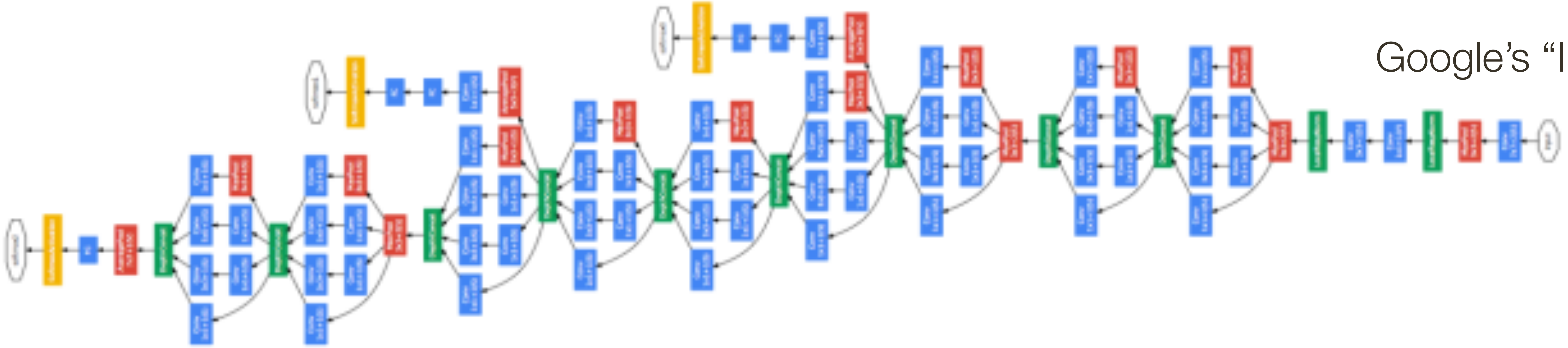
**Neural Network** (output): **softmax** function, where probability of class k is:

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

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

Special case for multi-class single label

# Deep Learning **Terminology**



Google's "Inception" network

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

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

deeper = better

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

requires knowledge of the nature of the problem

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

# Training

**Initialize** parameters of all layers

For a fixed number of iterations or until convergence

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

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

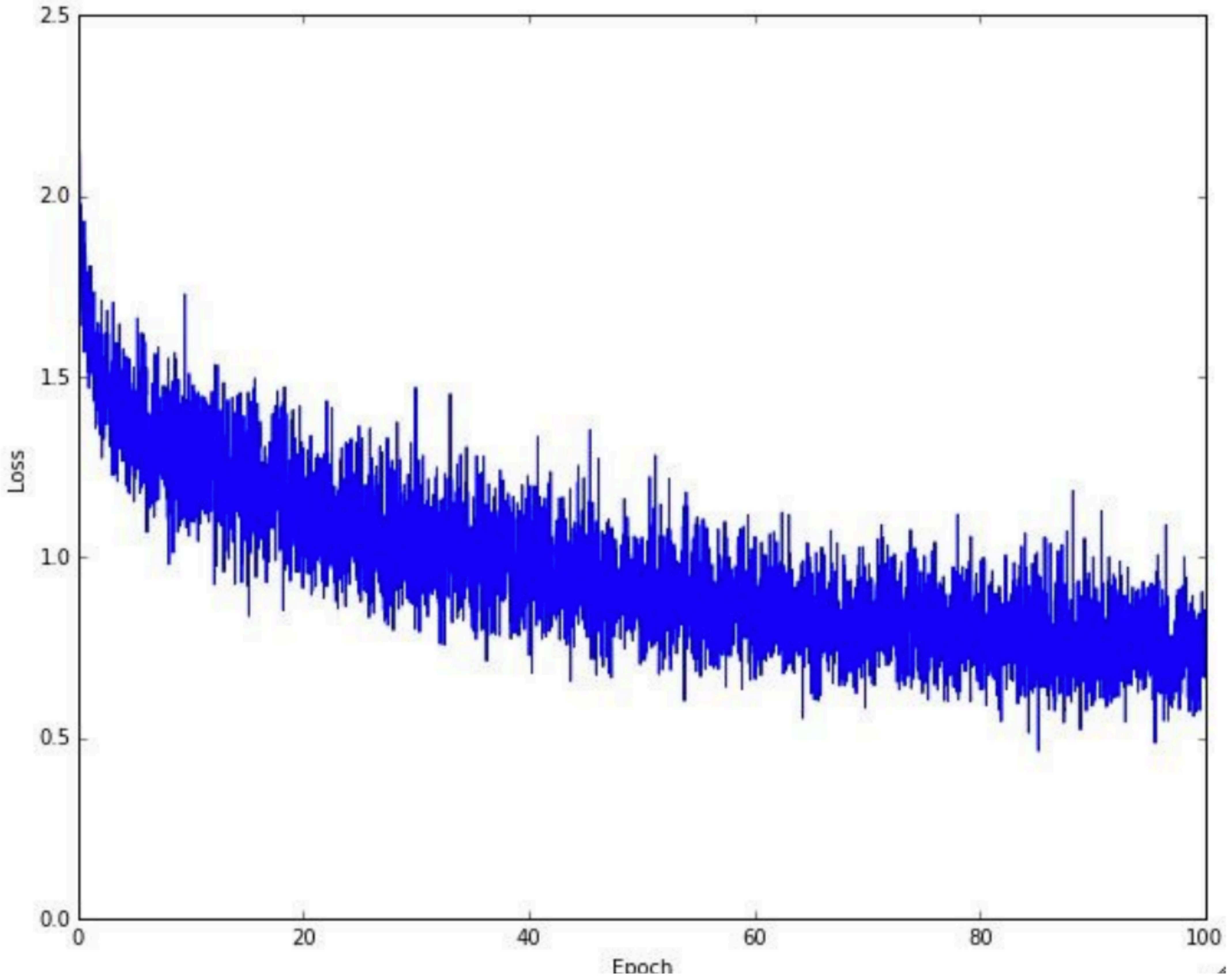
— Compute **backwards** pass to compute the gradient of the loss with respect to each parameter for each example (involves traversing computational graph in reverse order calling backward() on intermediate nodes and composing intermediate gradients — chain rule)

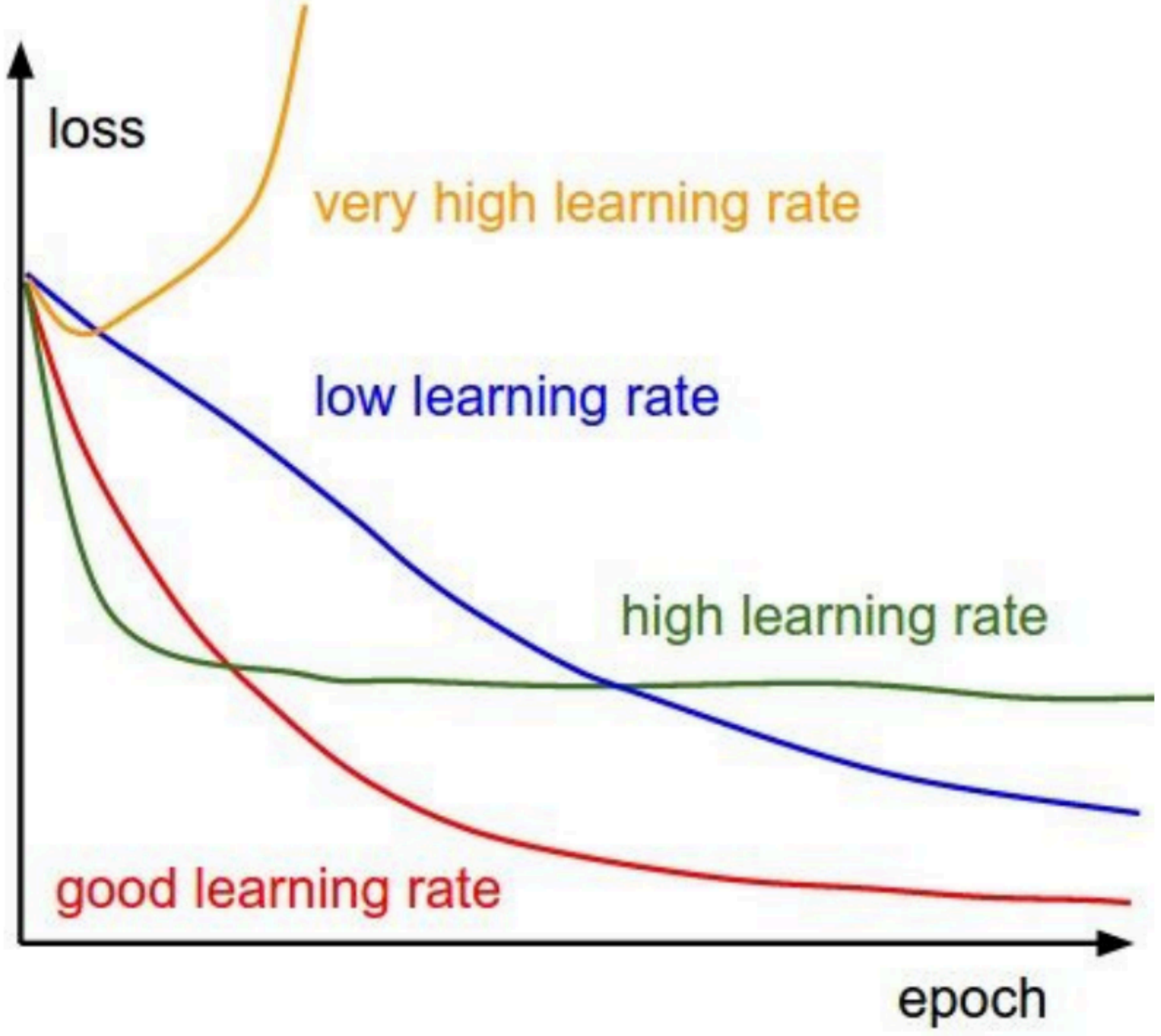
— **Update parameters** of all layers, by taking a step in the negative **average** gradient direction (computed over all examples in the mini-batch)

# **Inference** / Prediction

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

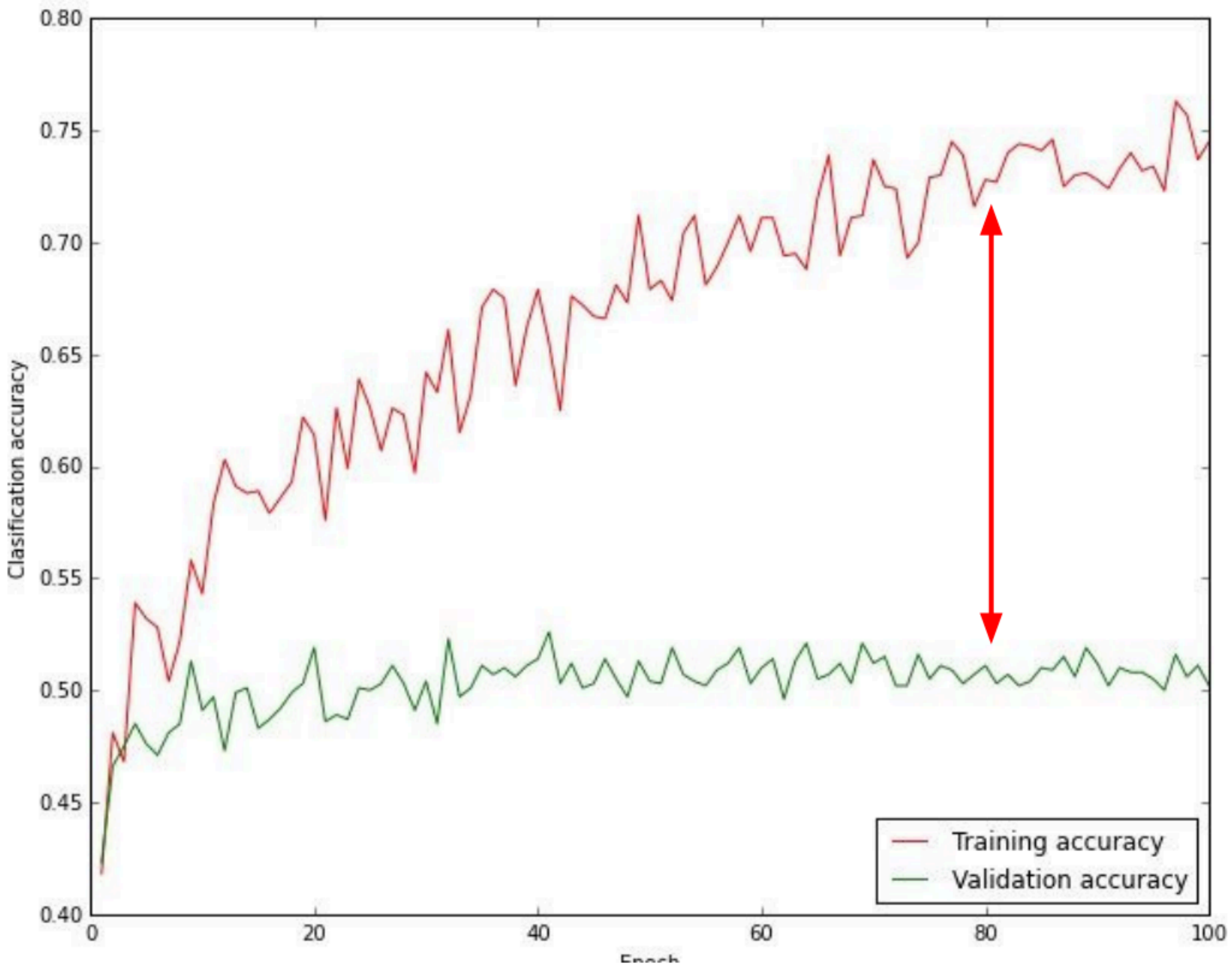
# **Monitoring Learning:** Visualizing the (training) loss




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

# **Monitoring Learning:** Visualizing the (training) loss




Big gap = overfitting

**Solution:** increase regularization

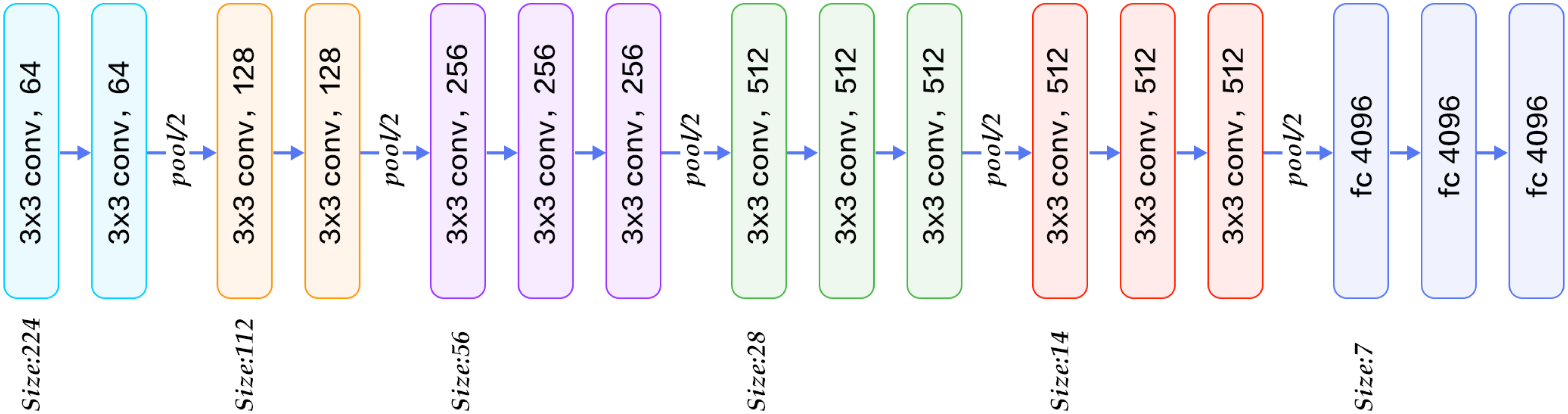
No gap = undercutting

**Solution:** increase model capacity

Small gap = **ideal**

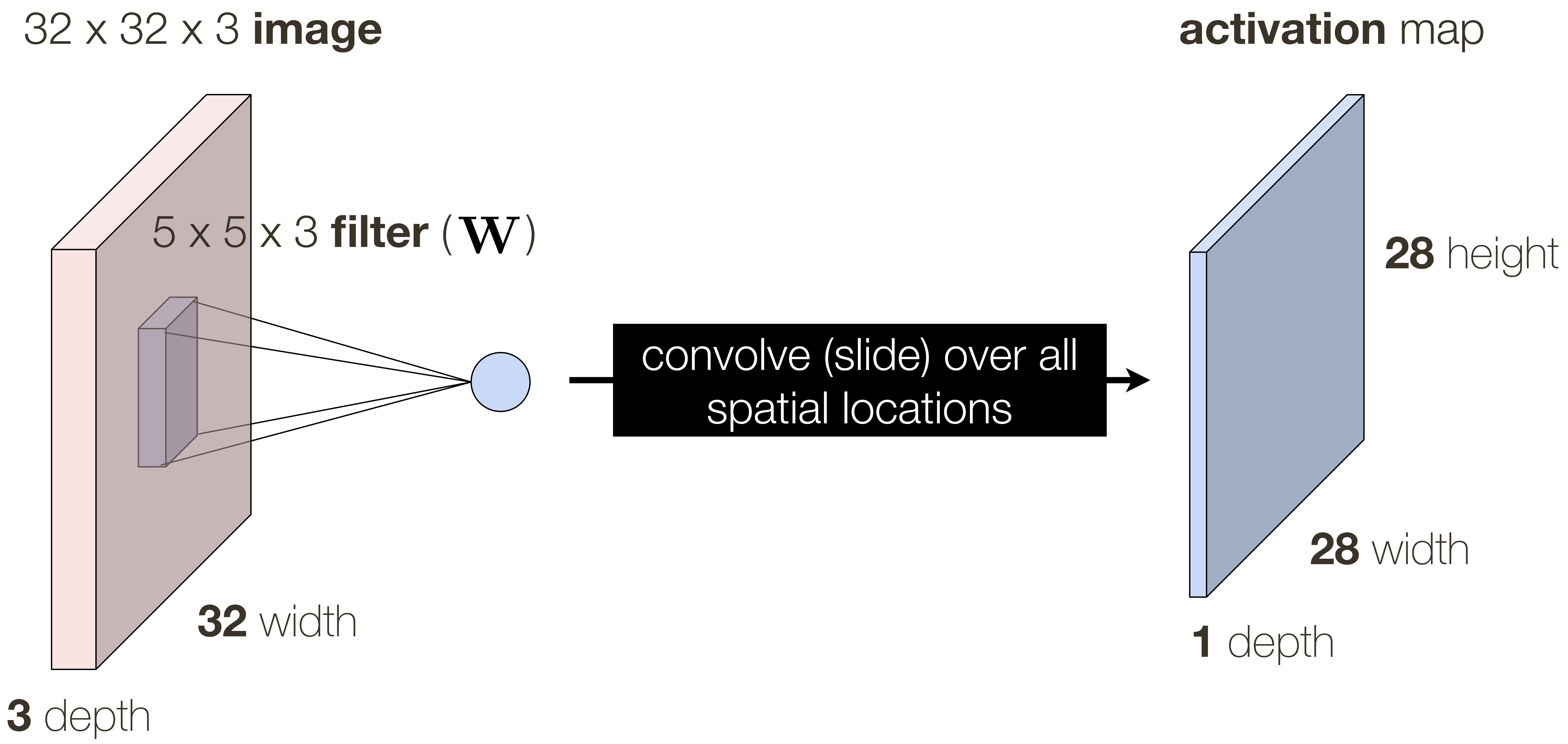
* slide from Li, Karpathy, Johnson's **CS231n at Stanford**

# **Convolutional** Neural Networks



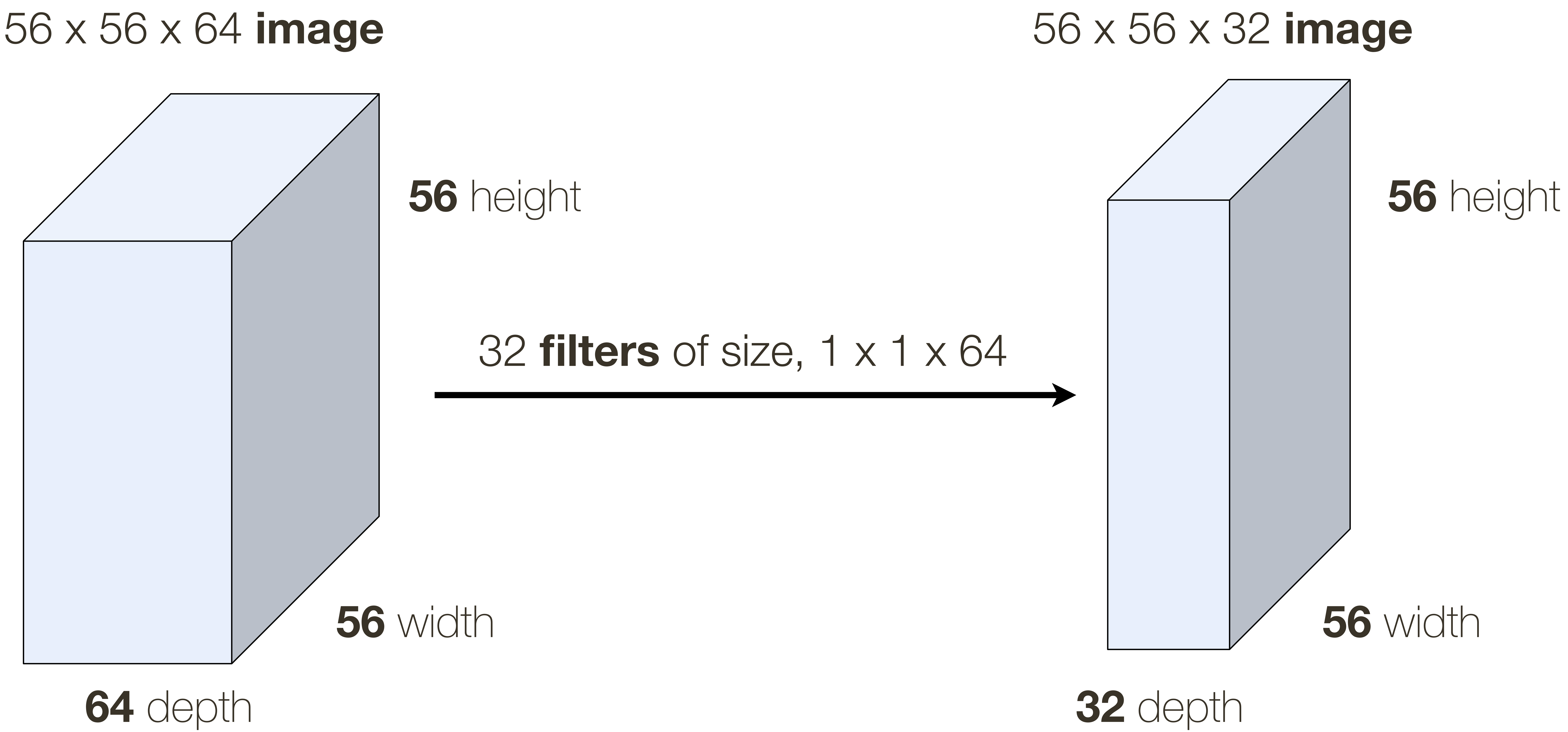

**VGG-16** Network

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



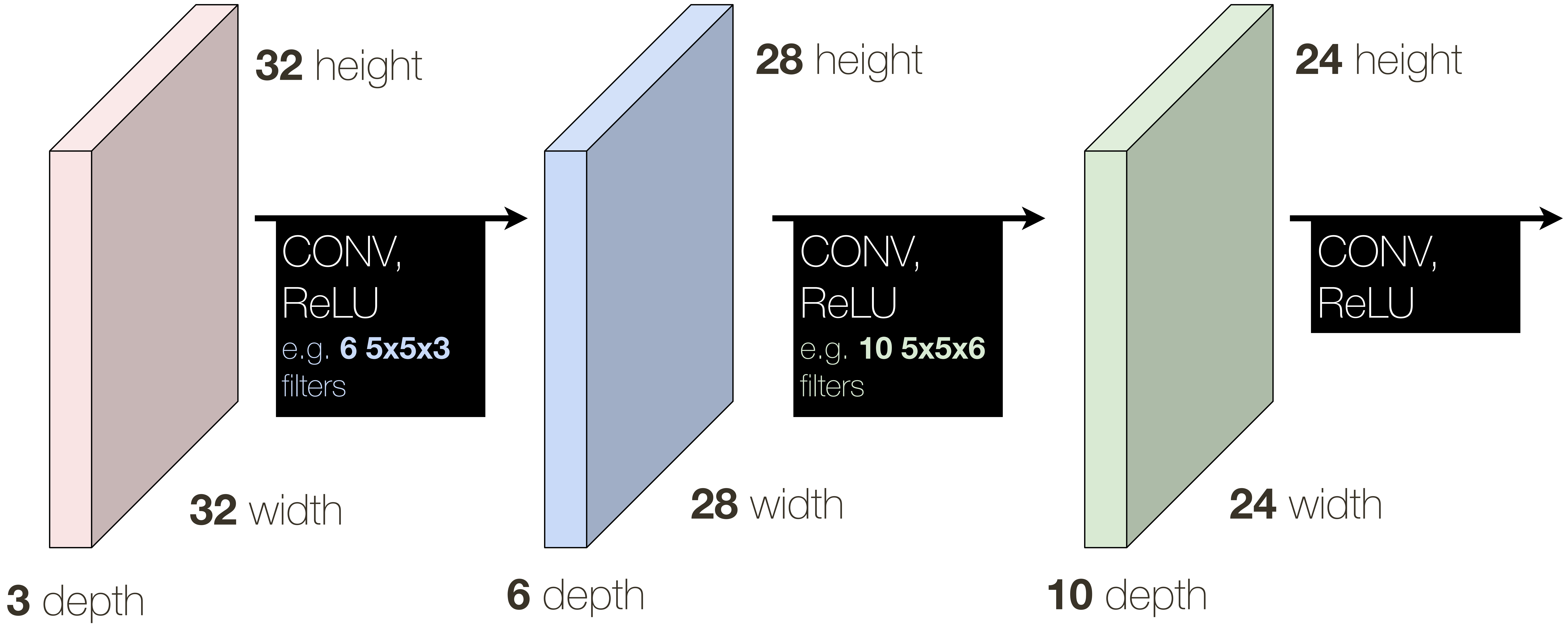

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

# Convolutional Layer: **1x1 convolutions**




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

# **Convolutional** Neural Network (ConvNet)




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

# Convolutional Layer **Summary**

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

# Convolutional Layer **Summary**

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

Requires hyperparameters:

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

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

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

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

# Convolutional Layer **Summary**

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

Requires hyperparameters:

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

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

# Convolutional Layer **Summary**

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

Requires hyperparameters:

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

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

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

# Convolutional Layer **Summary**

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

Requires hyperparameters:

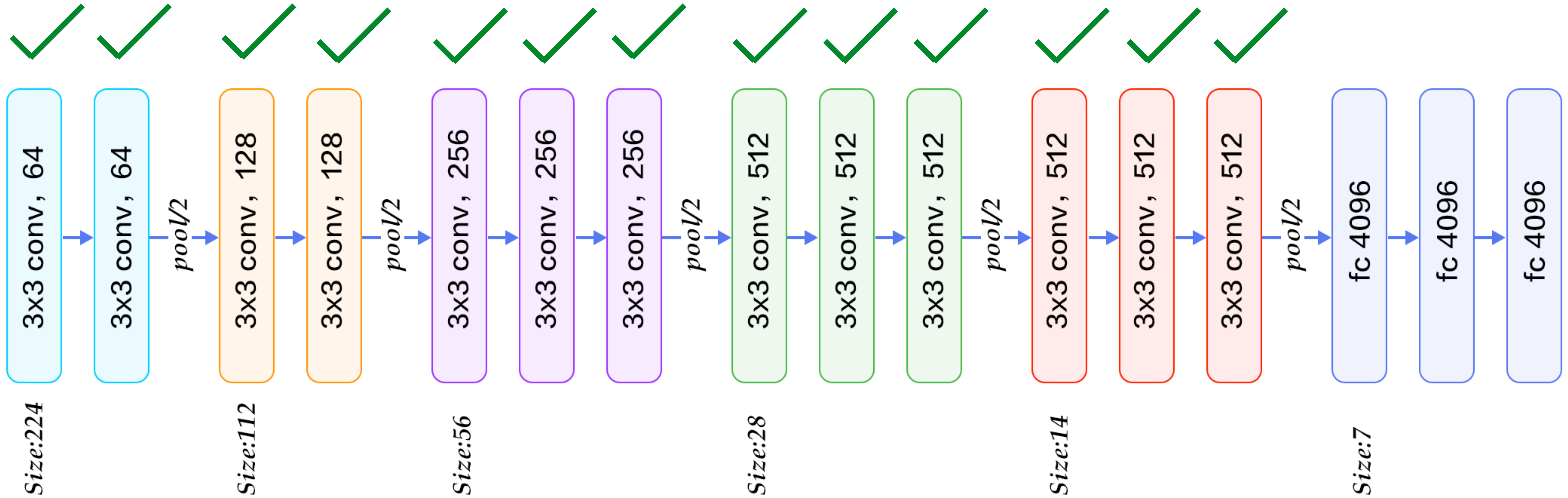
- Number of filters: $K$ (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: $F$ (for a typical networks $F \in \{1, 3, 5, ...\}$)
- Stride of application: $S$ (for a typical network $S \in \{1, 2\}$)
- Zero padding: $P$ (for a typical network $P \in \{0, 1, 2\}$)

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

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

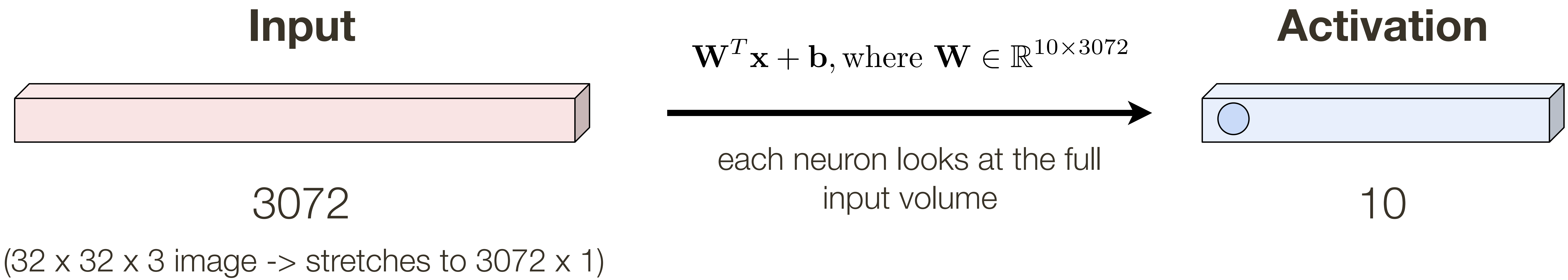
Number of total learnable parameters: $(F \times F \times D_i) \times K + K$

# **Convolutional** Neural Networks

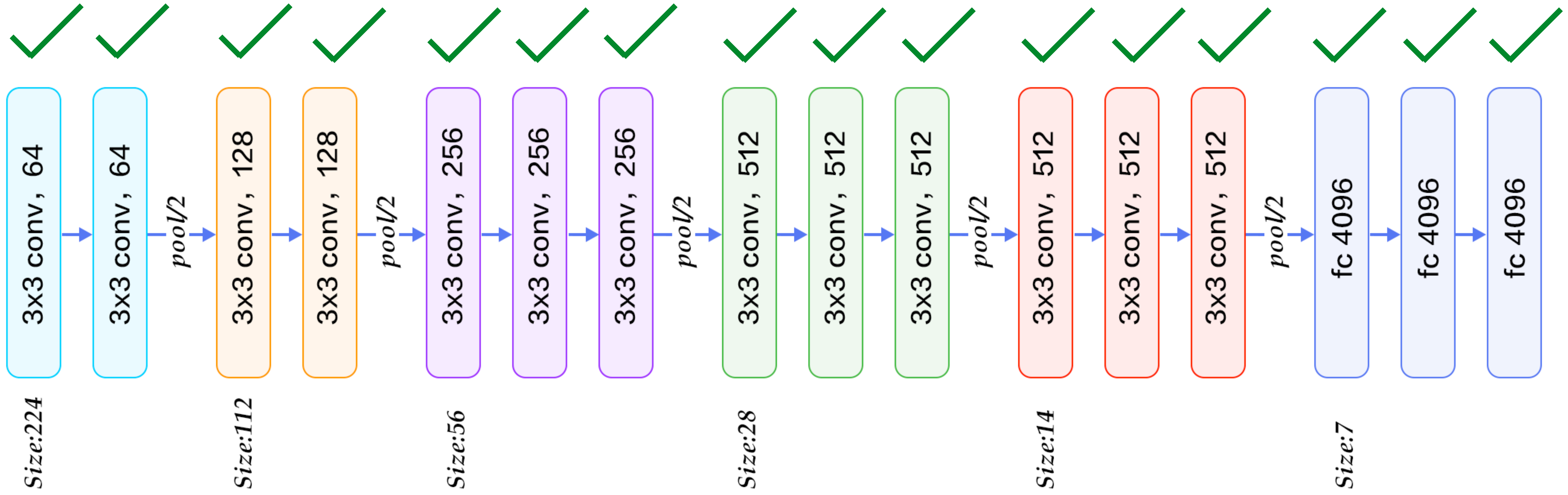



**VGG-16** Network

# **CNNs**: Reminder Fully Connected Layers

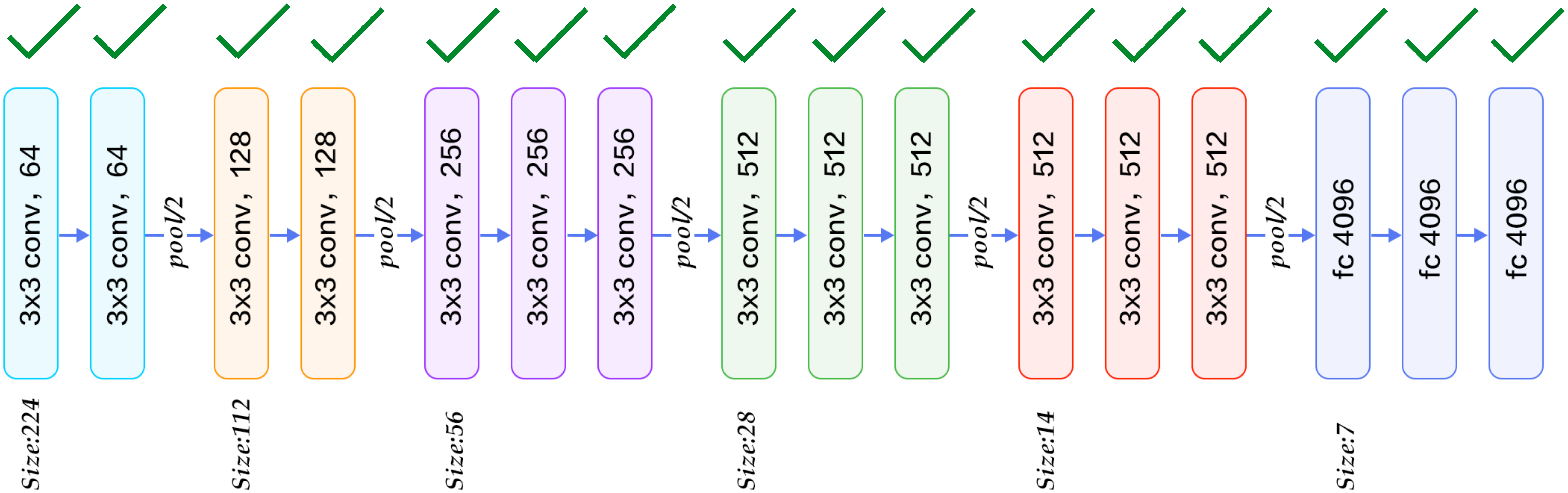


# **Convolutional** Neural Networks



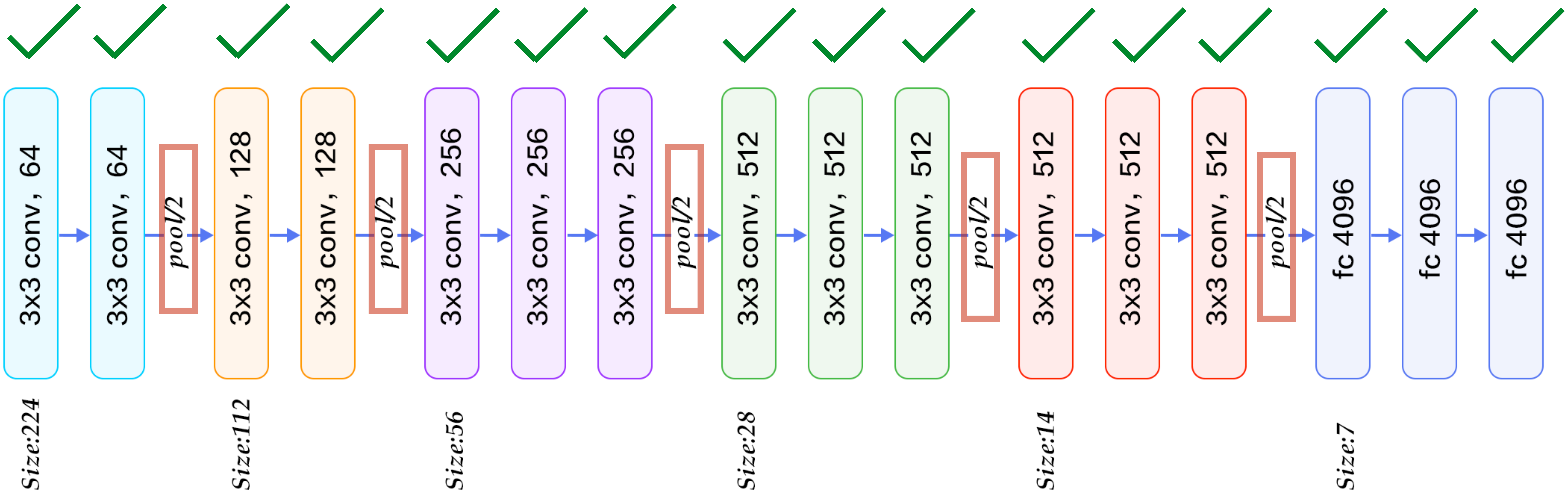

**VGG-16** Network

# **Convolutional** Neural Networks




**VGG-16** Network

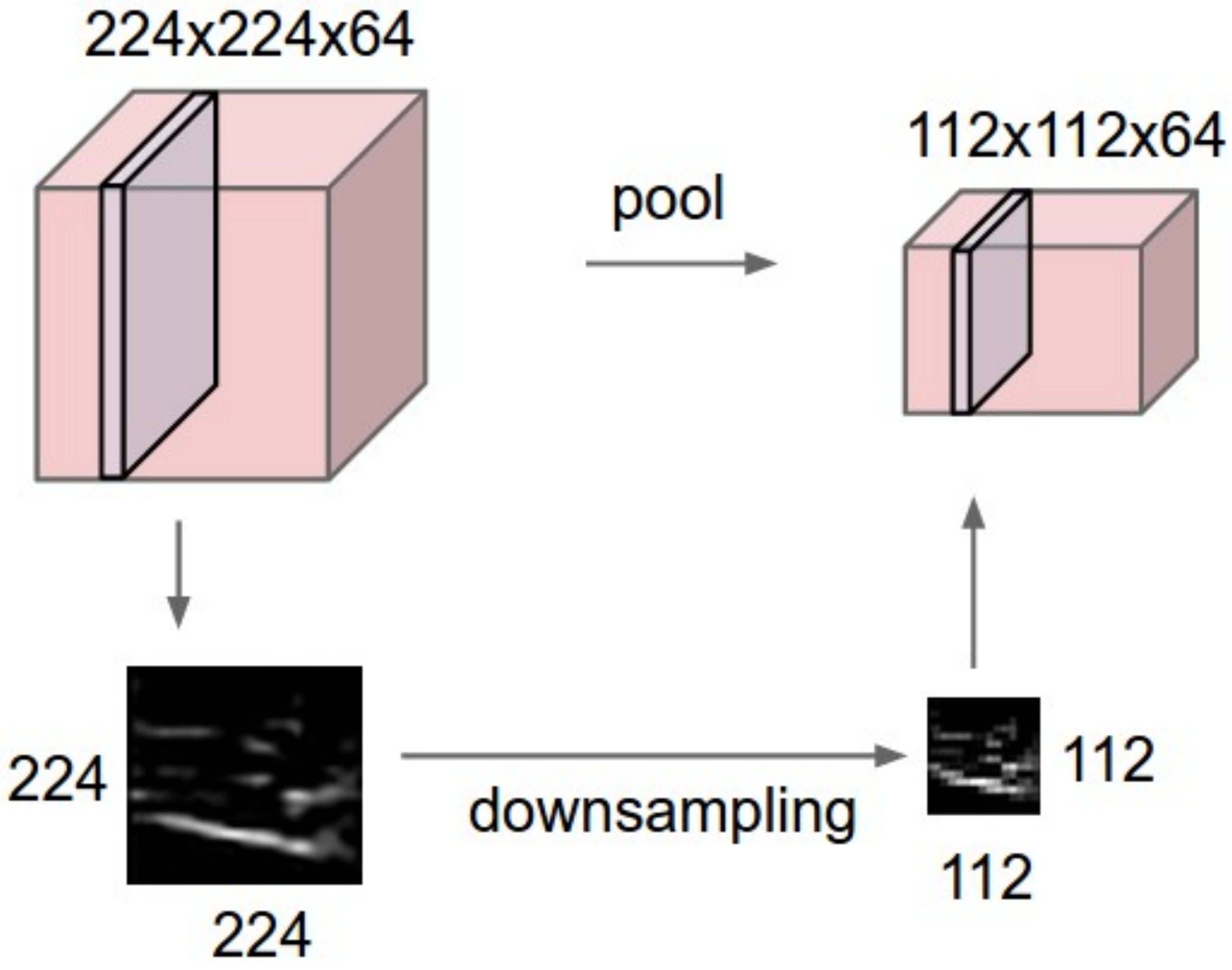
# **Convolutional** Neural Networks




**VGG-16** Network

# **Pooling** Layer

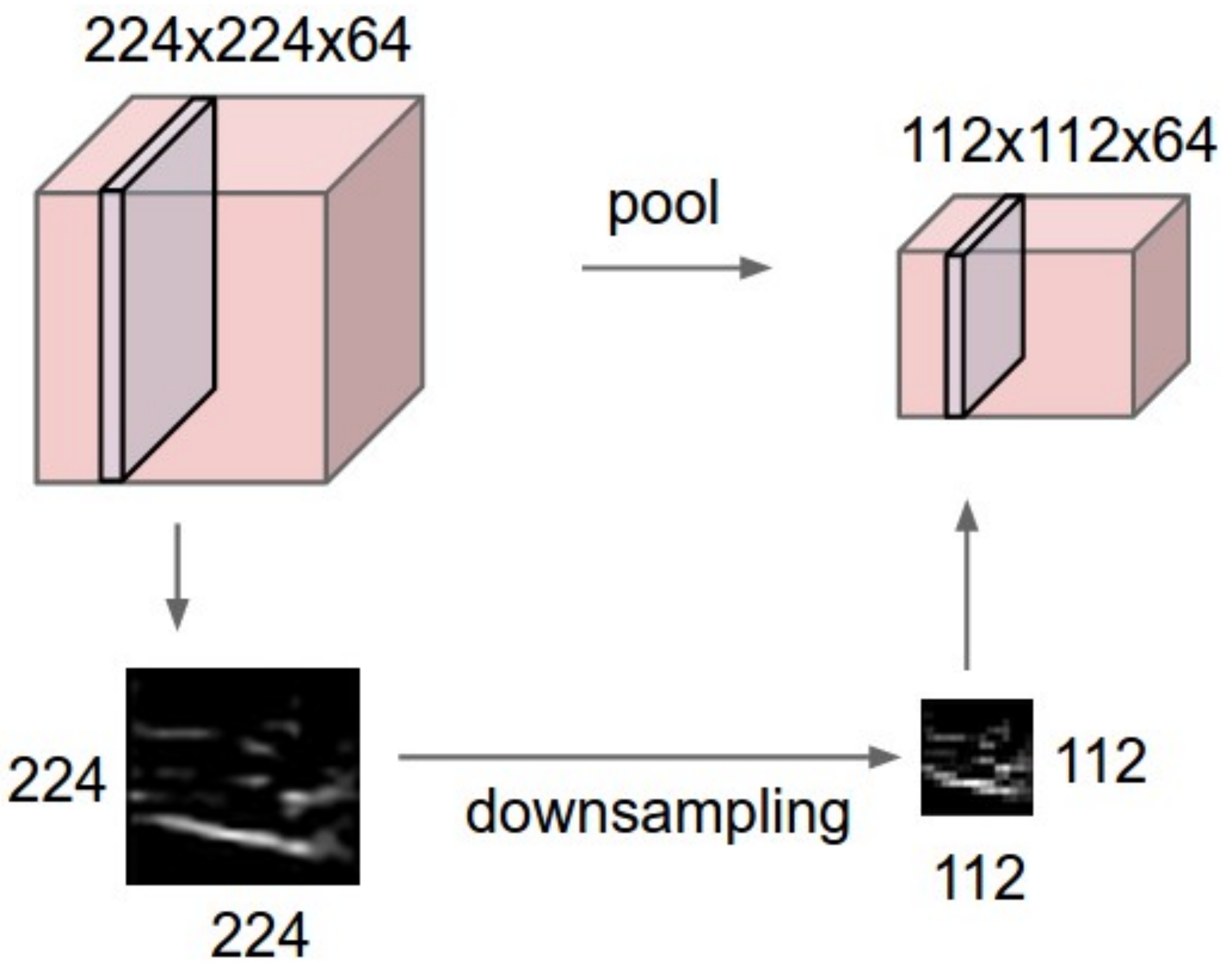
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently




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

# **Pooling** Layer

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

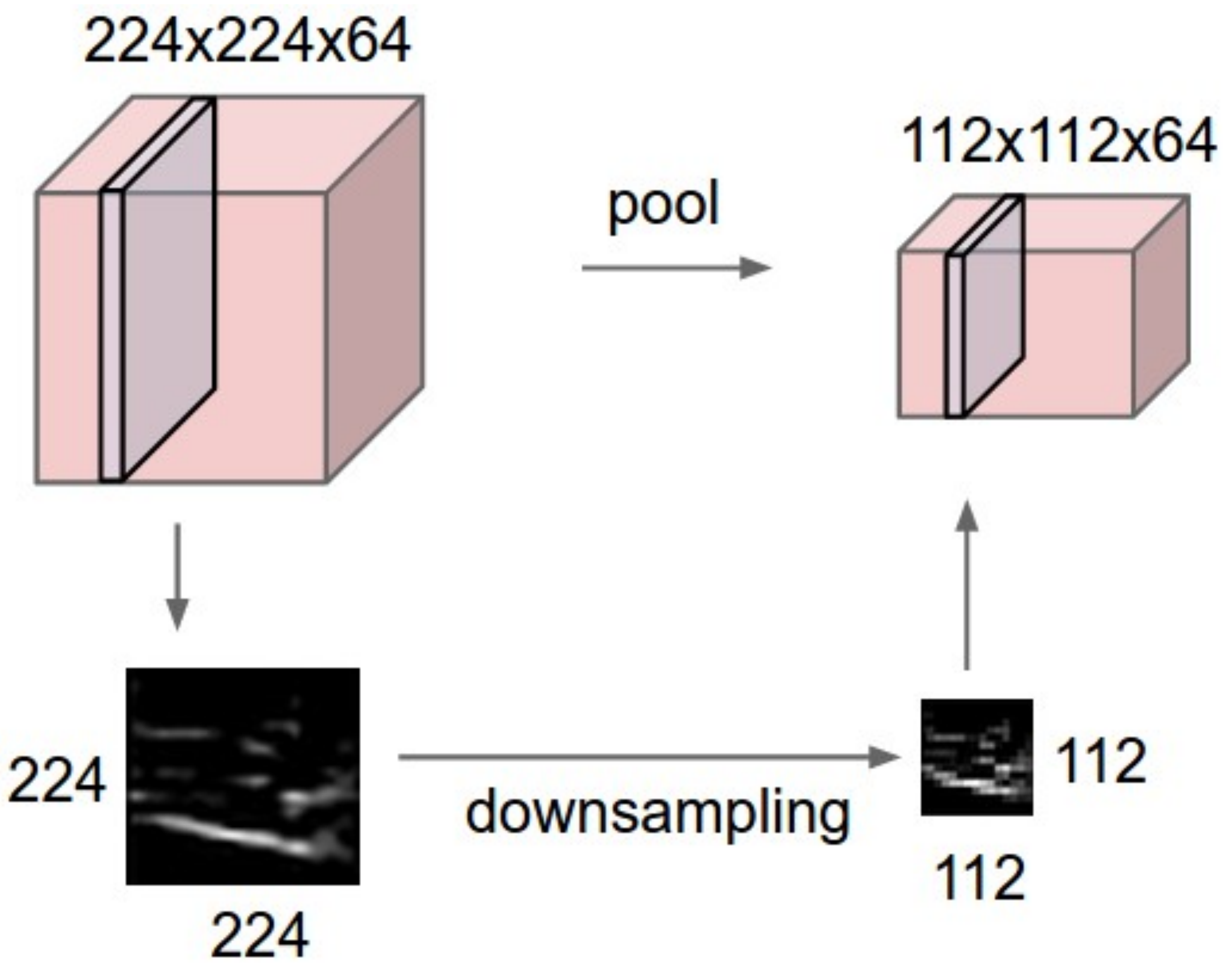



How many **parameters**?

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

# **Pooling** Layer

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




How many **parameters**?

**None**!

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

# Max **Pooling**

activation map

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

max pool with 2 x 2 filter
and stride of 2

| | |
|---|---|
| 6 | 8 |
| 3 | 4 |

# Average **Pooling**

activation map

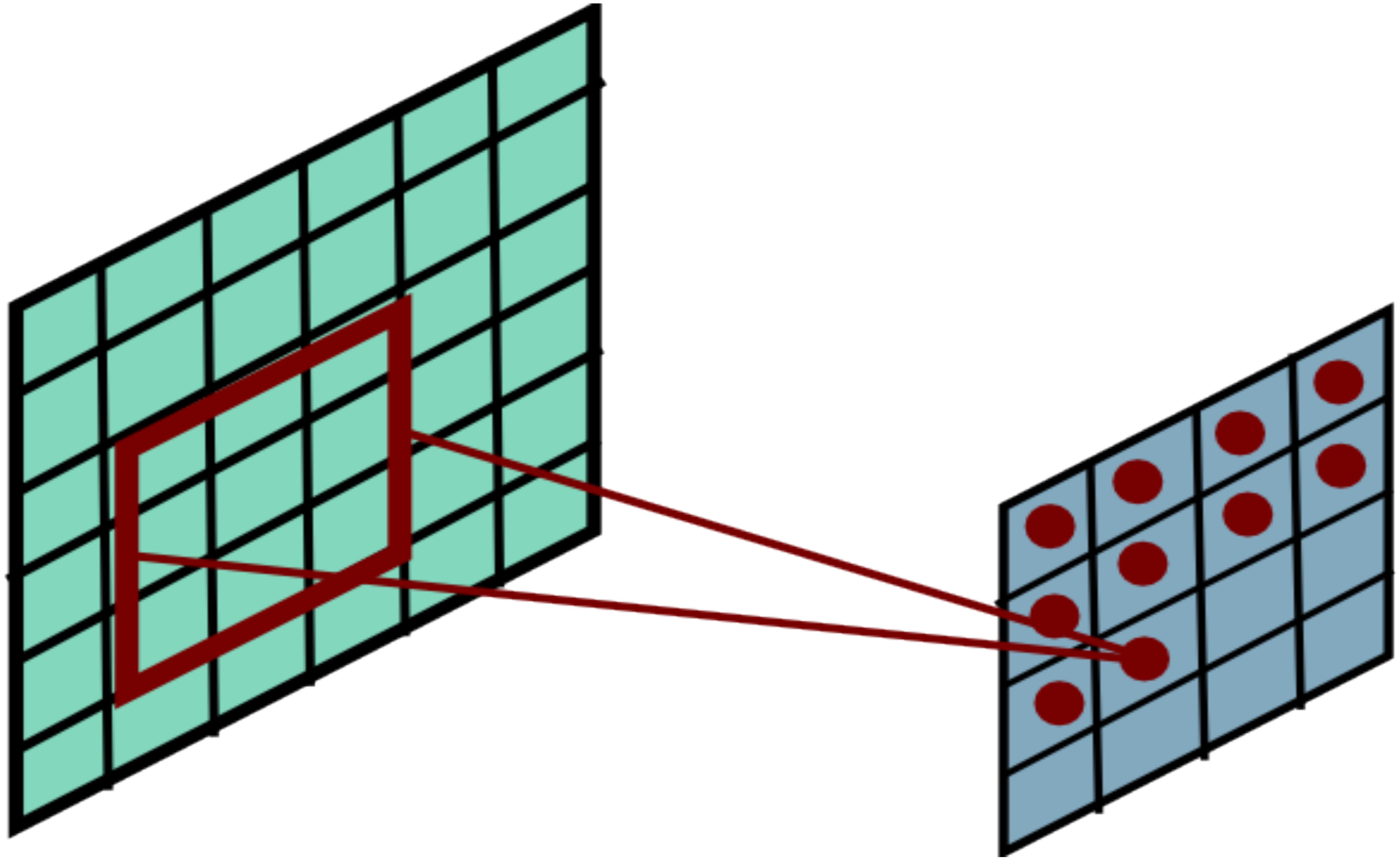
| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 2 | 1 | 0 |
| 1 | 2 | 3 | 4 |

avg pool with 2 x 2 filter
and stride of 2

| | |
|---|---|
| 3.25 | 5.25 |
| 2 | 2 |

# Pooling Layer **Receptive Field**

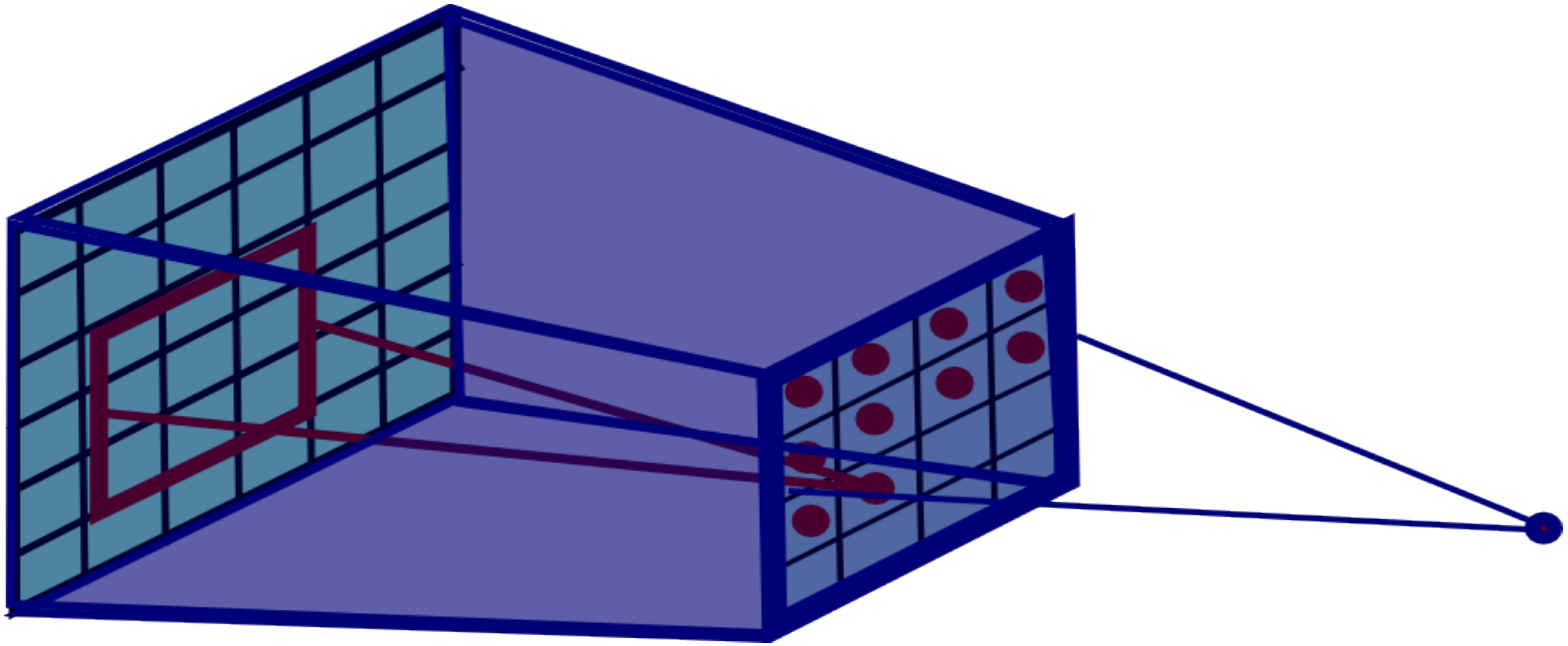
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: **(P+K-1)x(P+K-1)**



* slide from Marc'Aurelio Renzato

# Pooling Layer **Receptive Field**

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



* slide from Marc'Aurelio Renzato

# Pooling Layer **Summary**

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

Requires hyperparameters:

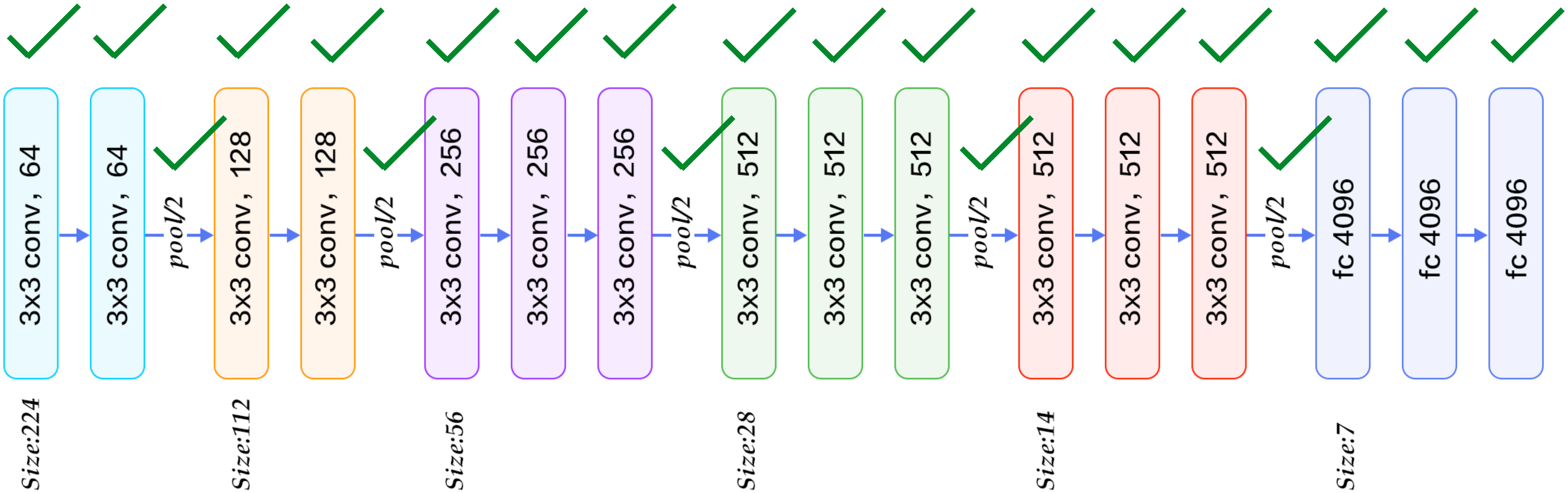
- Spatial extent of filters: $K$
- Stride of application: $F$

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

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

Number of total learnable parameters: 0

# **Convolutional** Neural Networks




**VGG-16** Network

# Computer **Vision Problems**

# Computer **Vision Problems**

Categorization

# Computer **Vision Problems**

Categorization

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

IMAGENET

# Computer **Vision Problems**

## Categorization

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

IMAGENET

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

# Computer **Vision Problems**

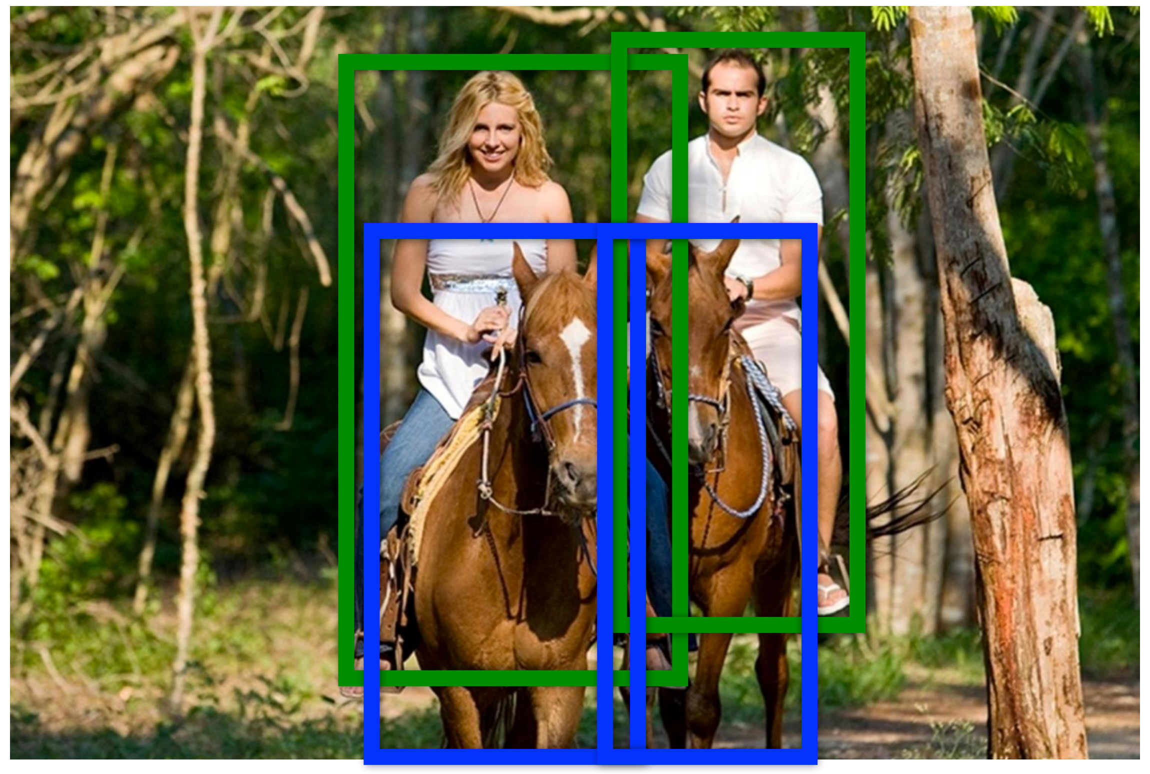
## Categorization

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

IMAGENET

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)

COCO
Common Objects in Context

# Computer **Vision Problems**

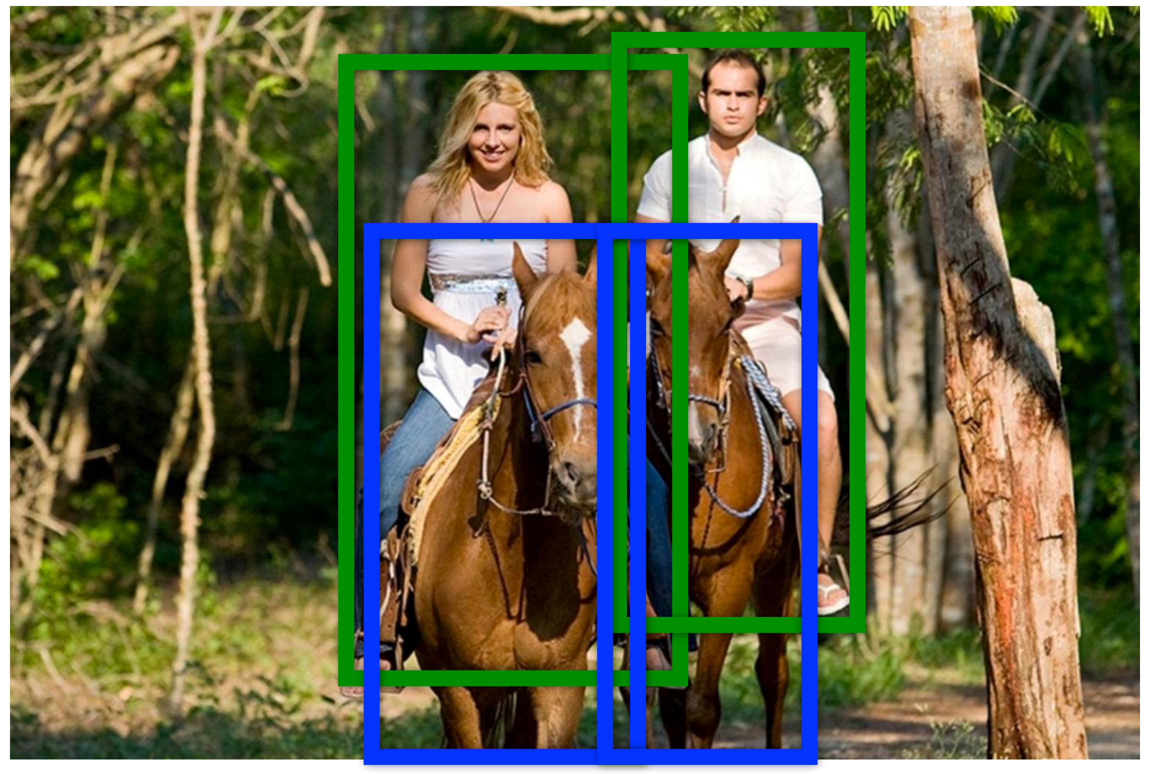
## Categorization

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

IMAGENET

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)

COCO
Common Objects in Context

## Segmentation

Horse
Person

COCO
Common Objects in Context

# Computer **Vision Problems**

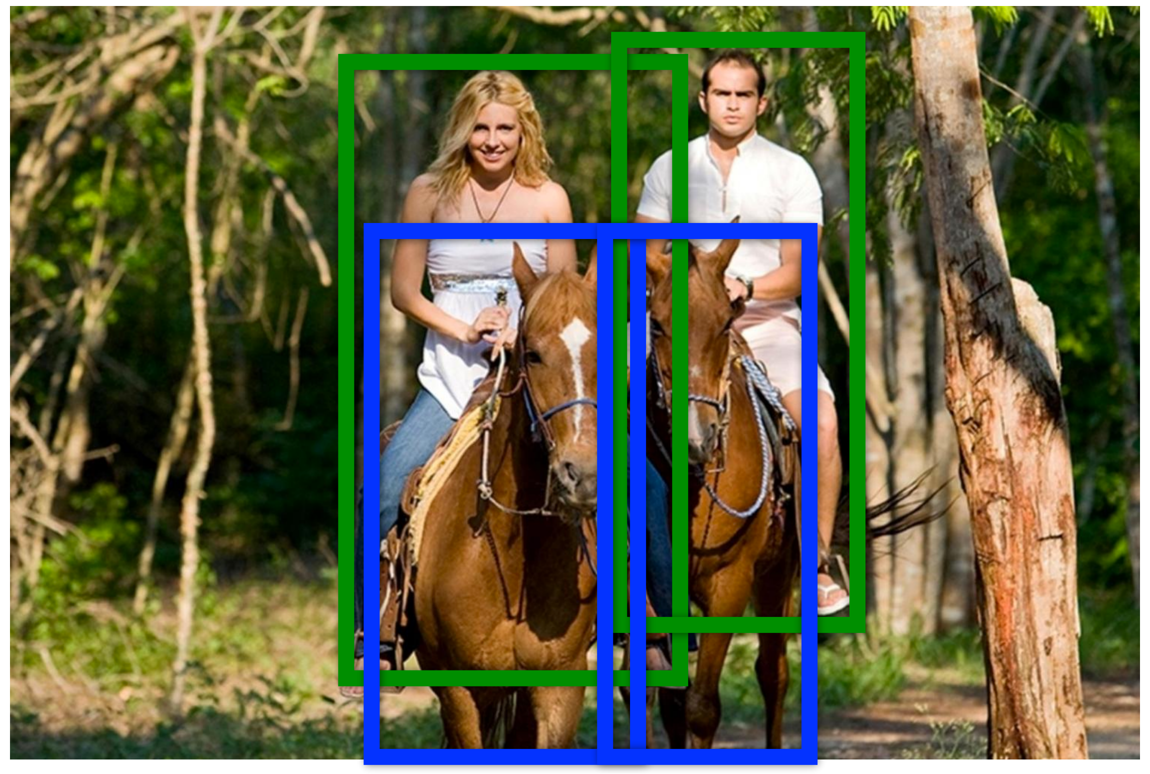
## Categorization

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

IMAGENET

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)

COCO Common Objects in Context

## Segmentation

Horse
Person

COCO Common Objects in Context

## Instance Segmentation

Horse1
Horse2
Person1
Person2

# Computer **Vision Problems**

Categorization

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

IMAGENET

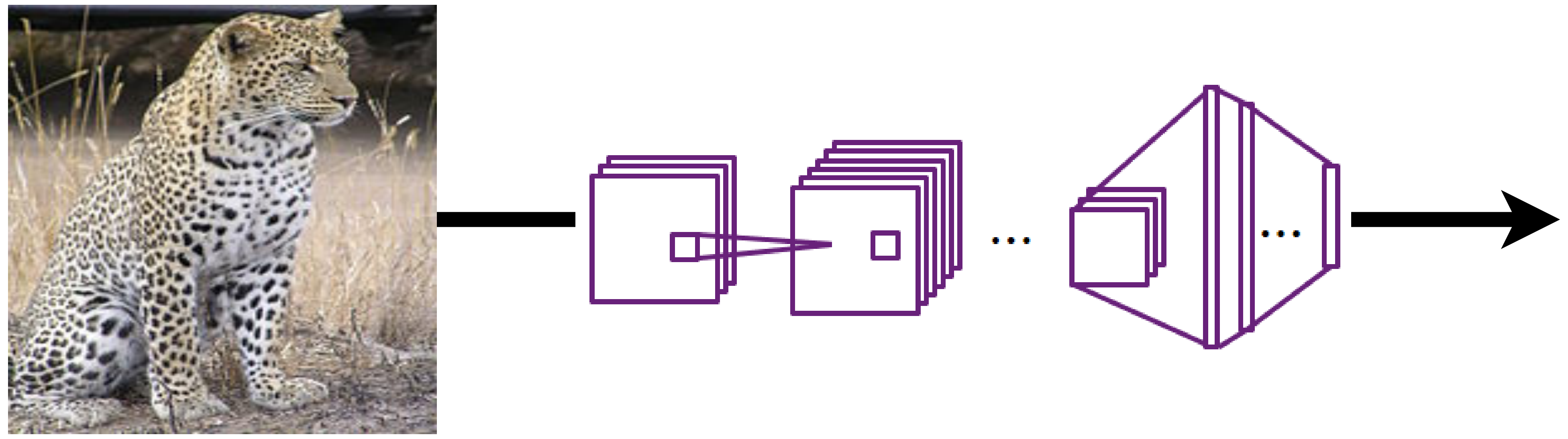
Multi-**label**: **Horse**
Church
Toothbrush
**Person**

# 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

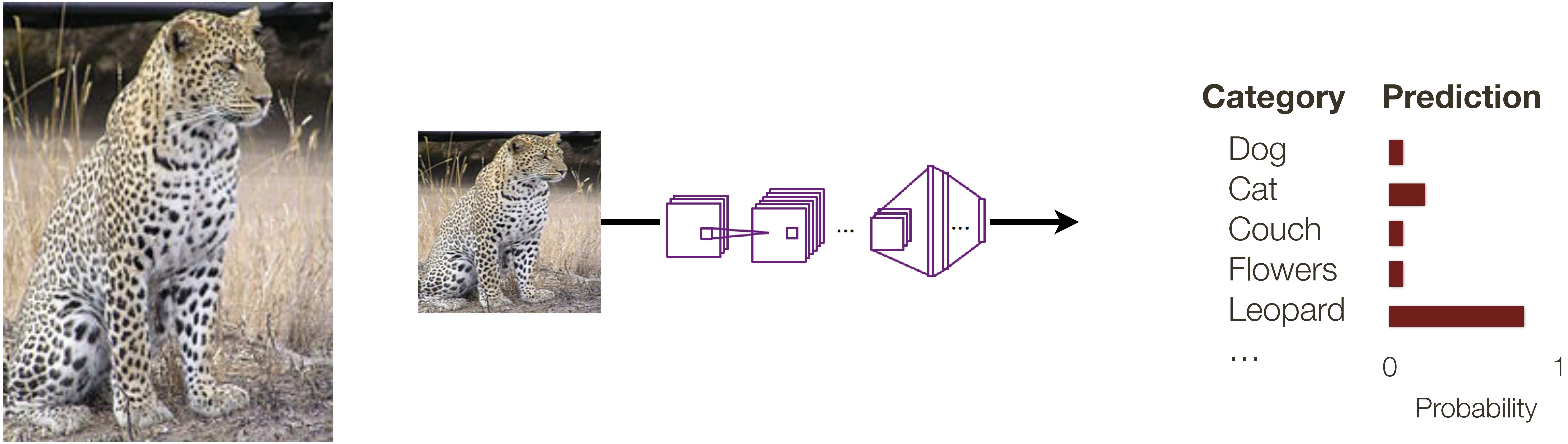
# Object **Classification**



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

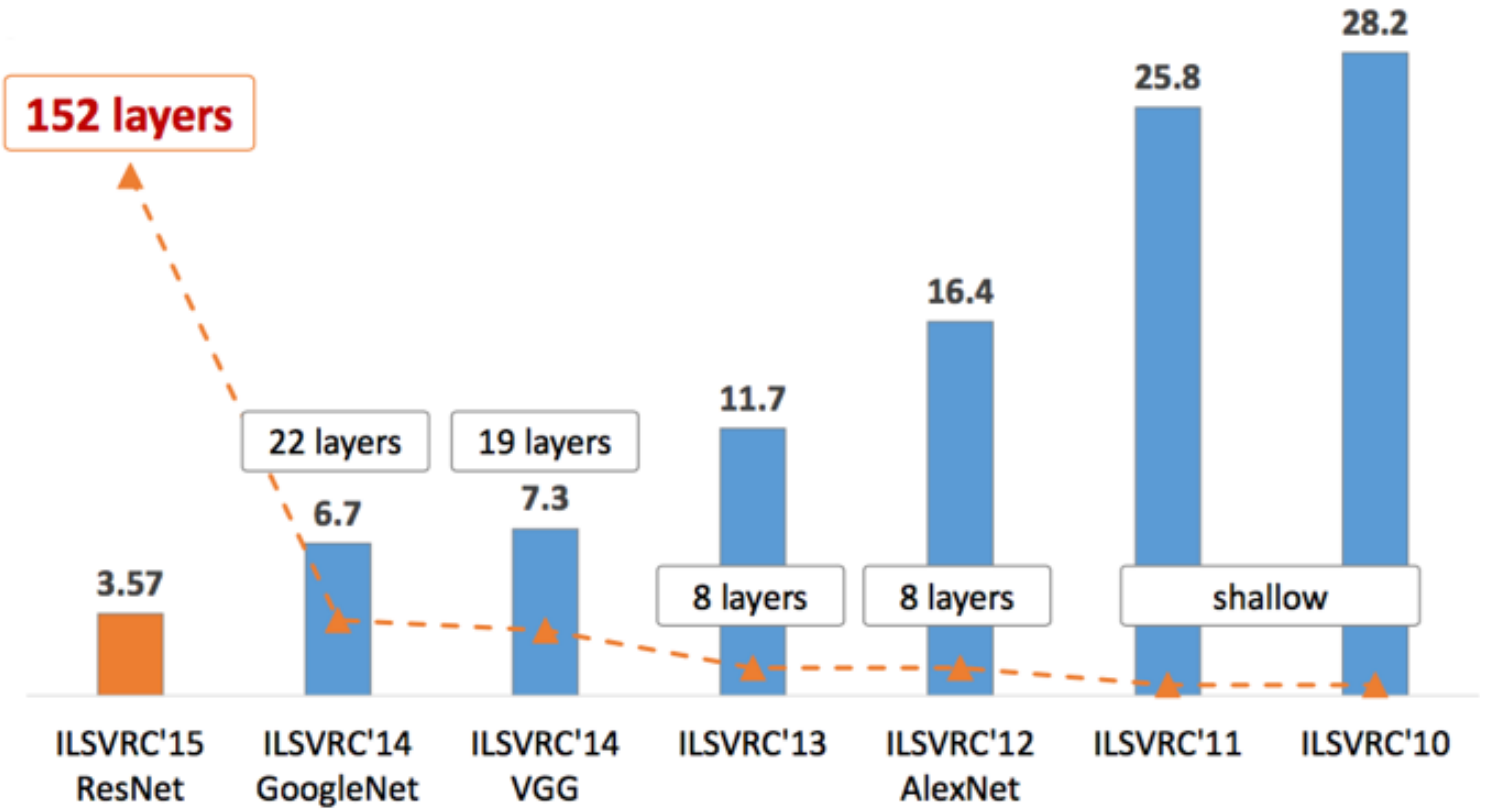
**Problem:** For each image predict which category it belongs to out of a fixed set

# Object **Classification**

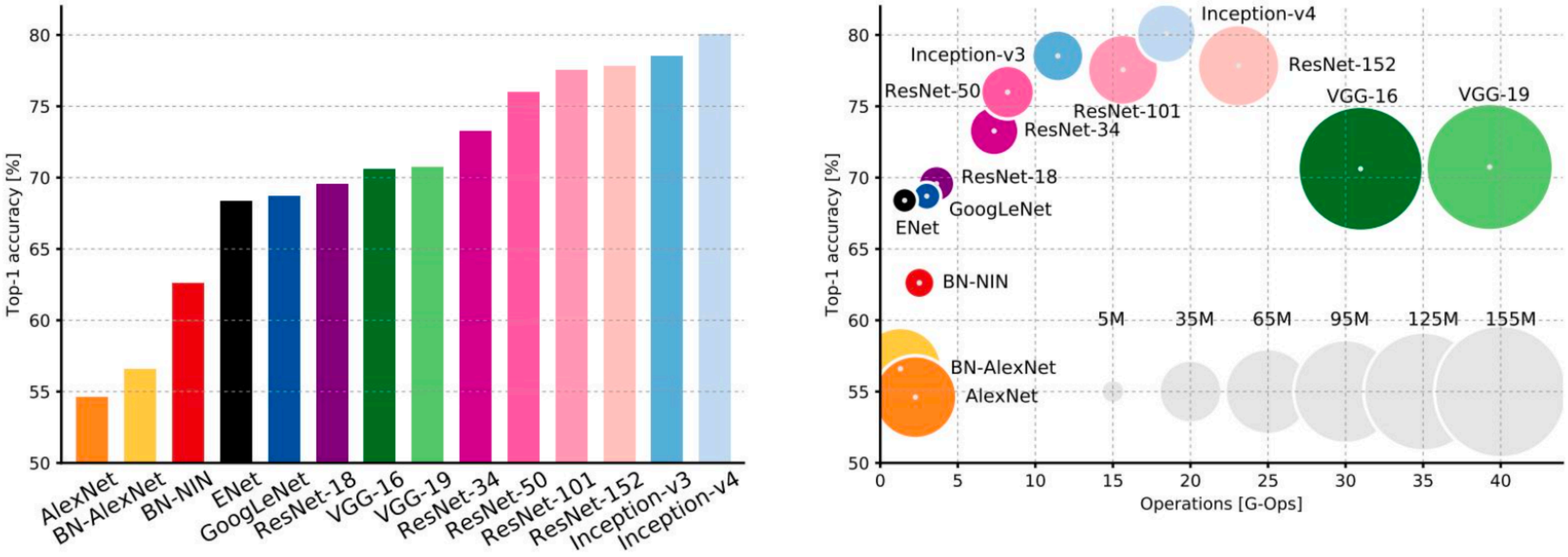



**Problem:** For each image predict which category it belongs to out of a fixed set

# Object **Classification**



# Comparing **Complexity**




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

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

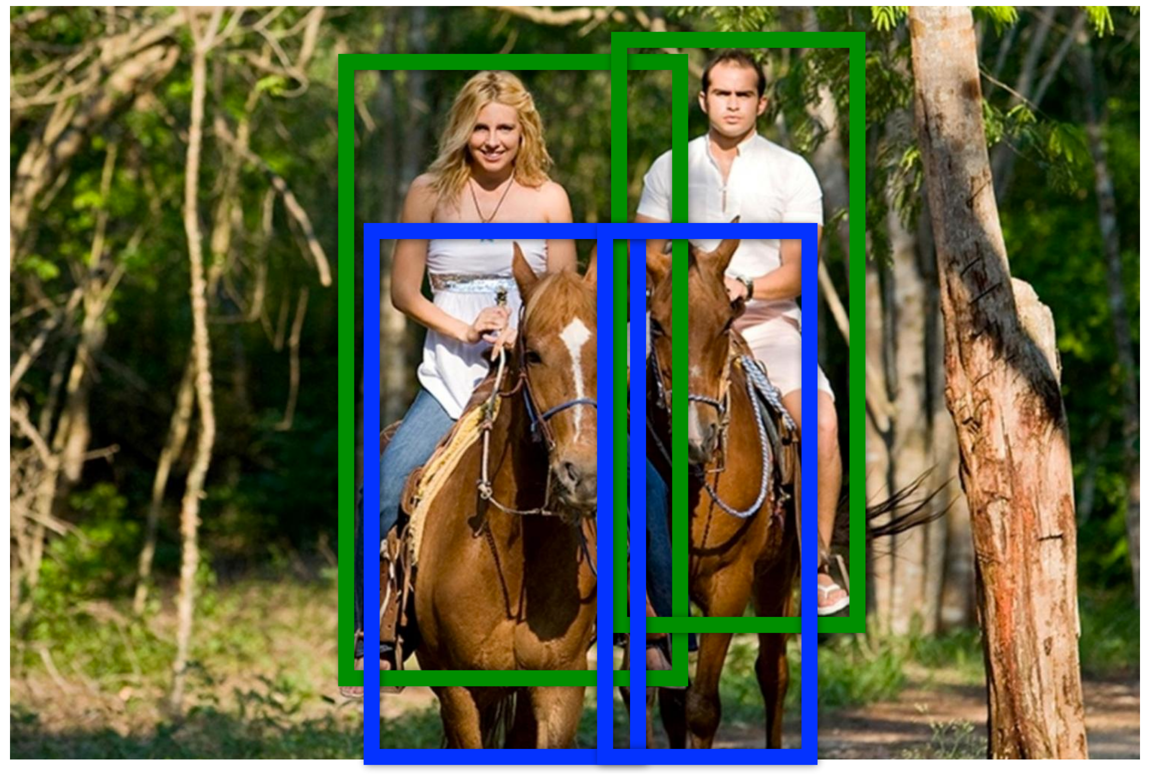
## Categorization

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

IMAGENET

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)

COCO Common Objects in Context

## Segmentation

Horse
Person

COCO Common Objects in Context

## Instance Segmentation

Horse1
Horse2
Person1
Person2

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

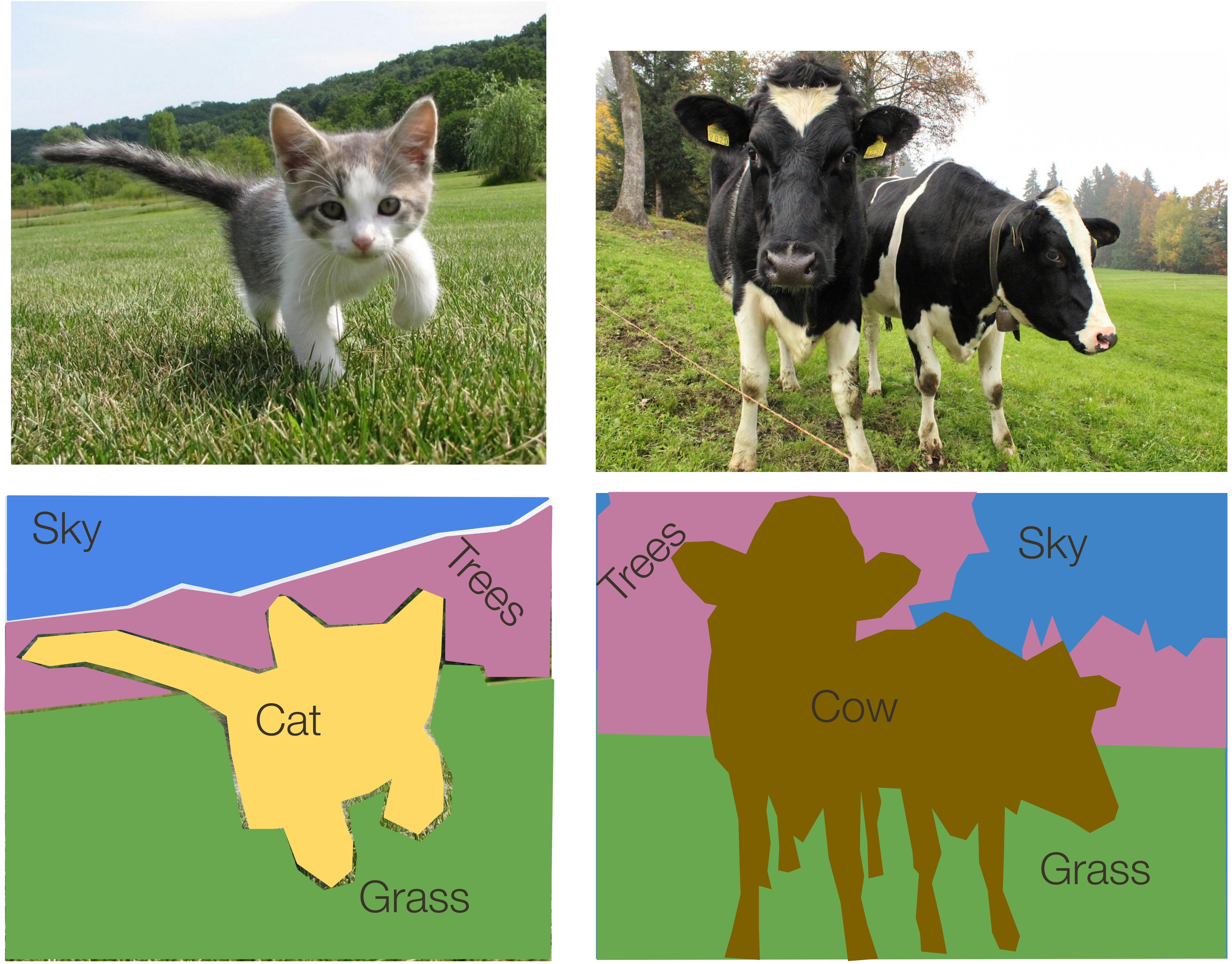
Segmentation

Horse
Person

COCO
Common Objects in Context

# Semantic **Segmentation**

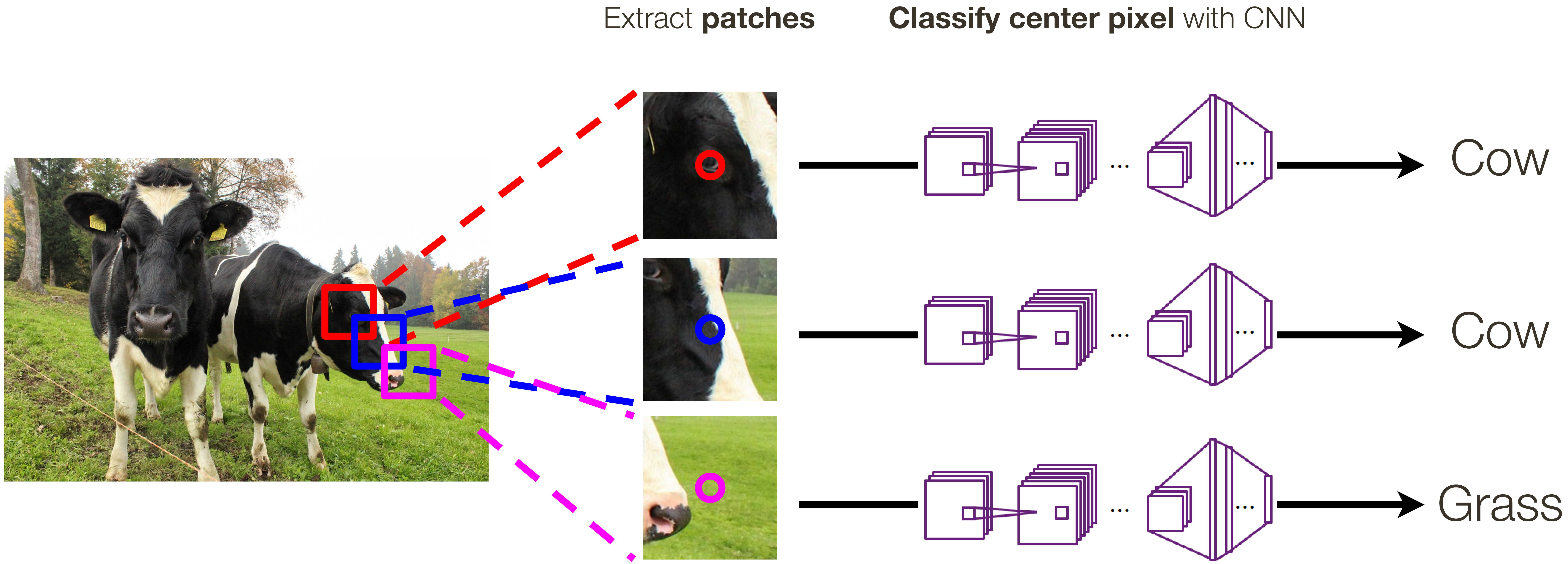
Label **every pixel** with a category label (without differentiating instances)




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

# Semantic **Segmentation:** Sliding Window

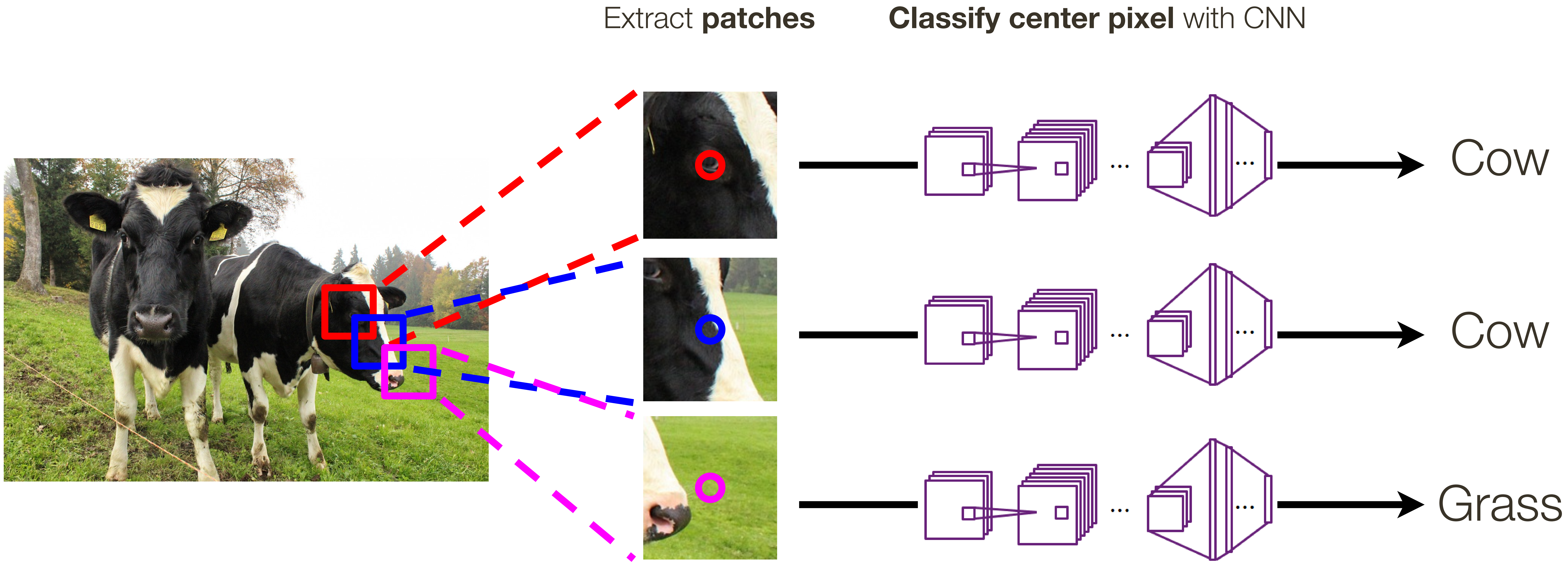
[ Farabet et al, TPAMI 2013 ]
[ Pinheiro et al, ICML 2014 ]




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

# Semantic **Segmentation:** Sliding Window

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

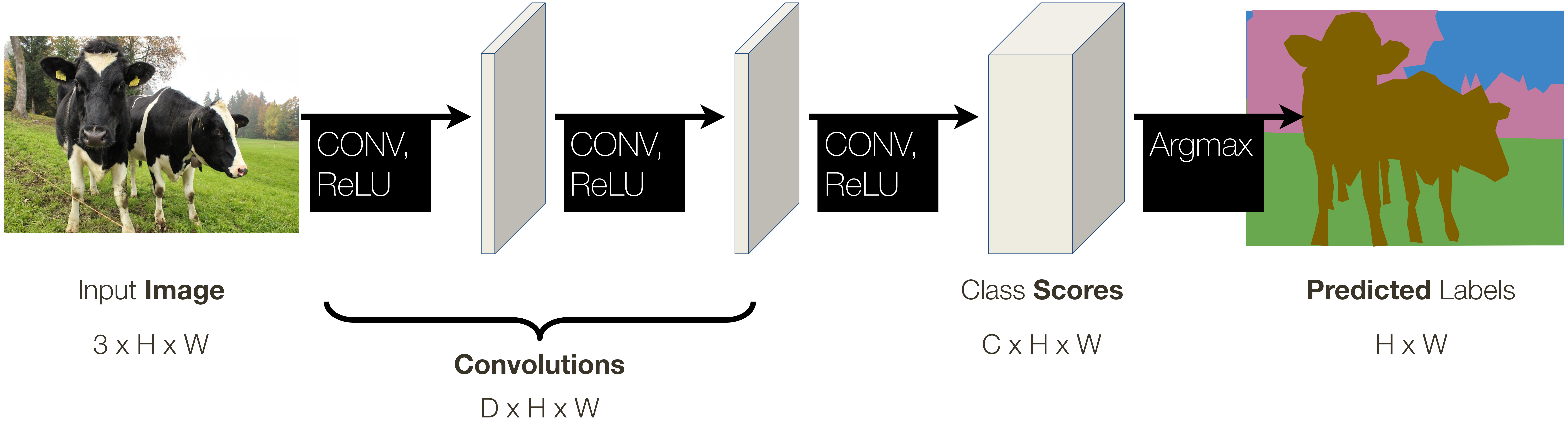



**Problem:** VERY inefficient, no reuse of computations for overlapping patches

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

# Semantic **Segmentation:** Fully Convolutional CNNs

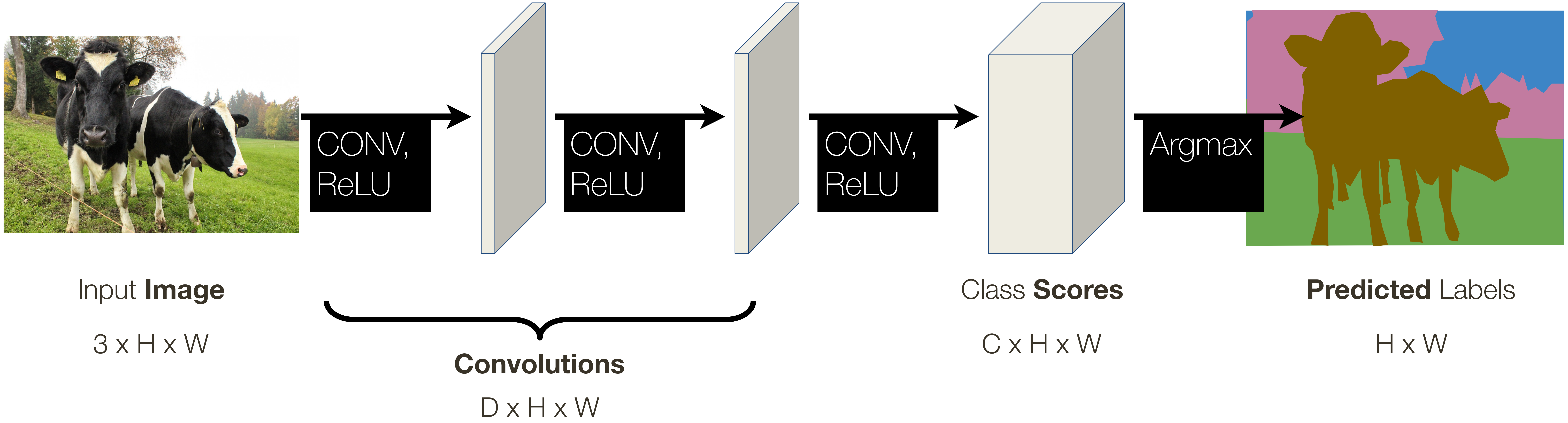
Design a network as a number of convolutional layers to make predictions for all pixels at once!




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

# 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

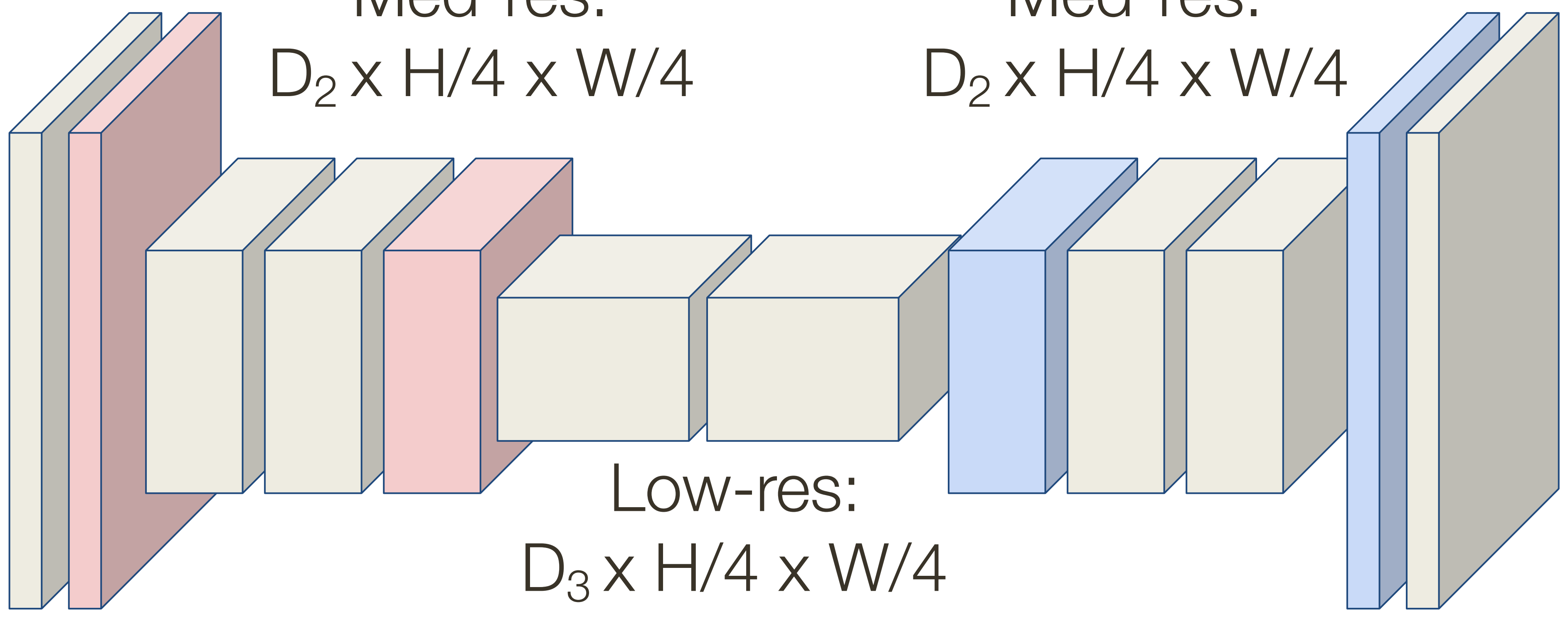
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

# Semantic **Segmentation:** Fully Convolutional CNNs

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

Input **Image**

3 x H x W




**Predicted** Labels

H x W

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

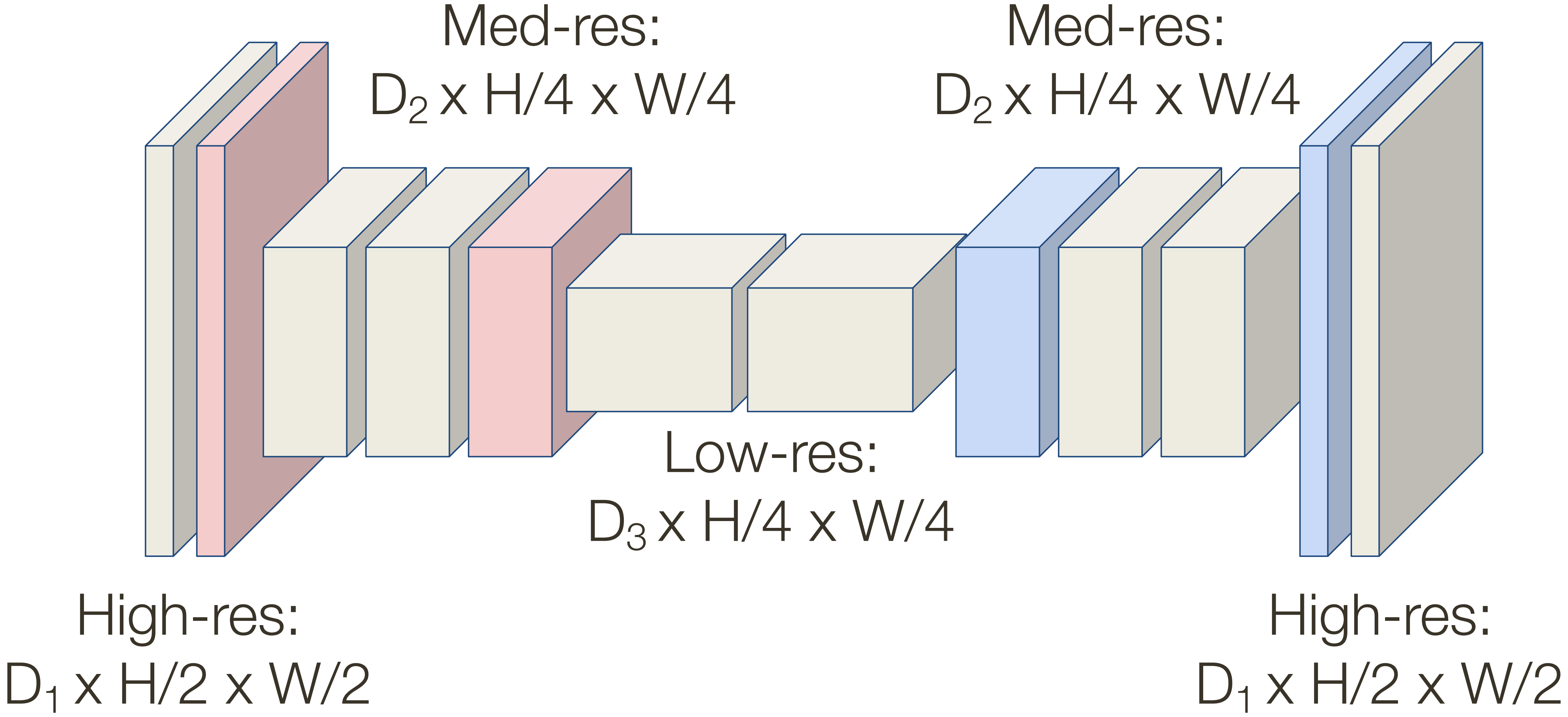
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

# Semantic **Segmentation:** Fully Convolutional CNNs

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

Input **Image**

3 x H x W




**Predicted** Labels

H x W

**Downsampling** = Pooling

**Upsampling** = ???

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

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

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

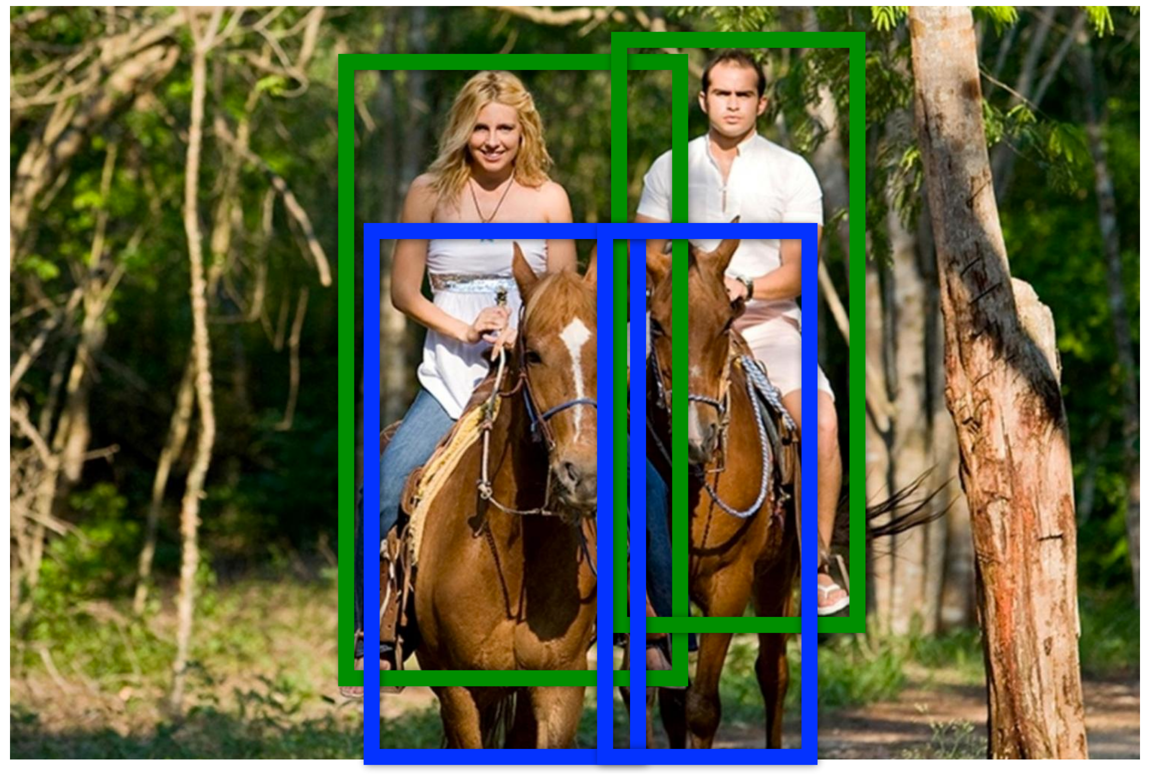
## Categorization

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

IMAGENET

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)

COCO Common Objects in Context

## Segmentation

Horse
Person

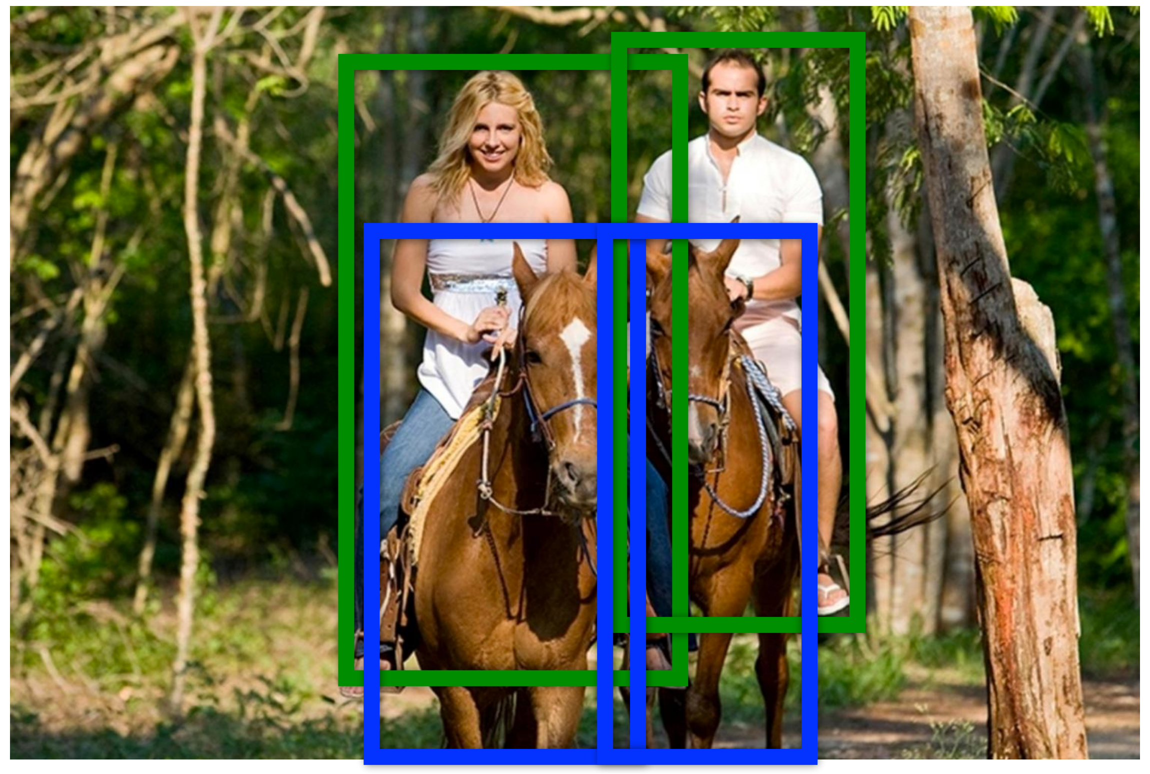
COCO Common Objects in Context

## Instance Segmentation

Horse1
Horse2
Person1
Person2

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

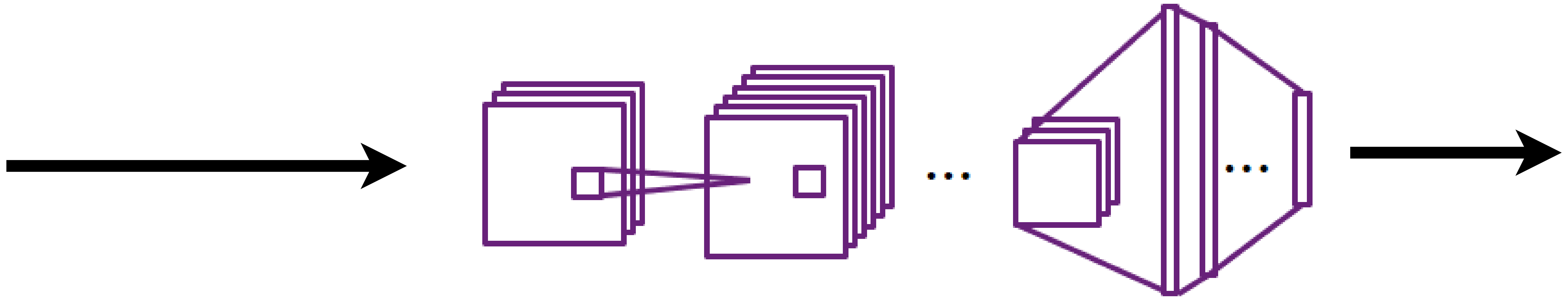
## Detection



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

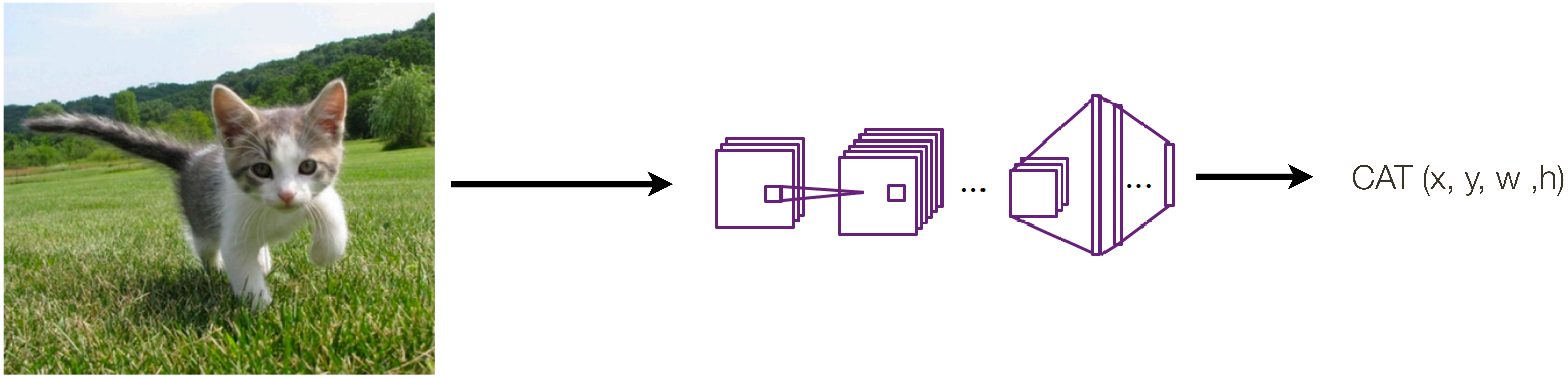
COCO
Common Objects in Context

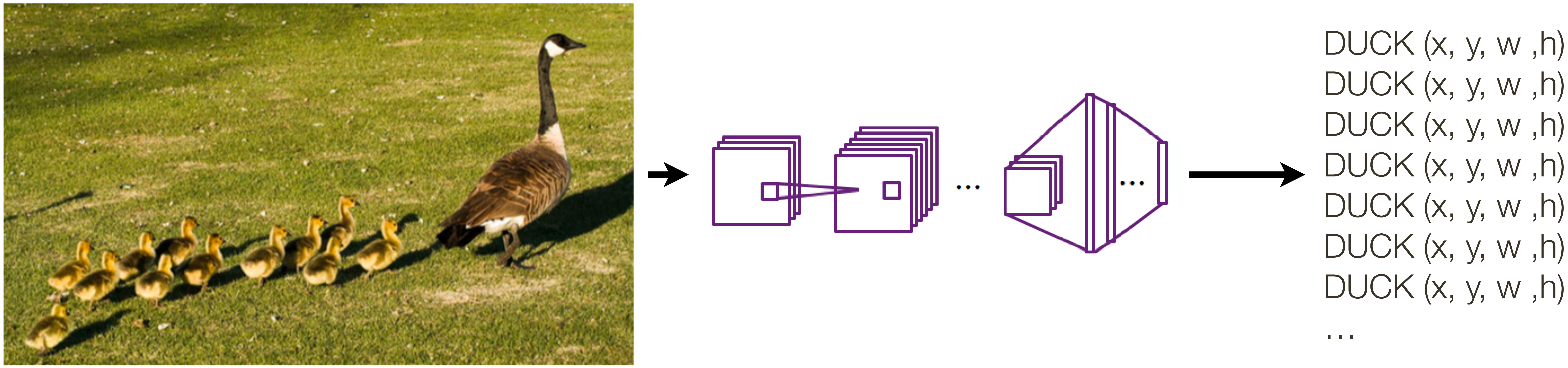
# Object **Detection** as Regression Problem



CAT (x, y, w ,h)

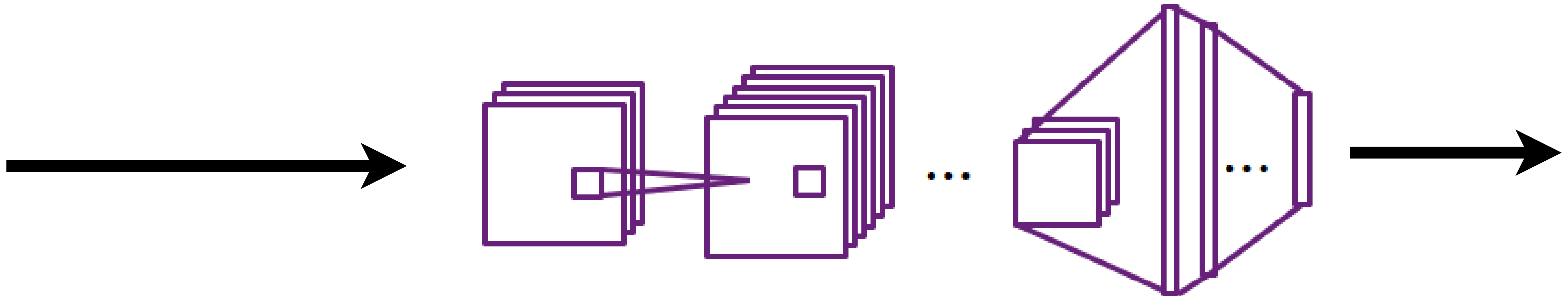
# Object **Detection** as Regression Problem



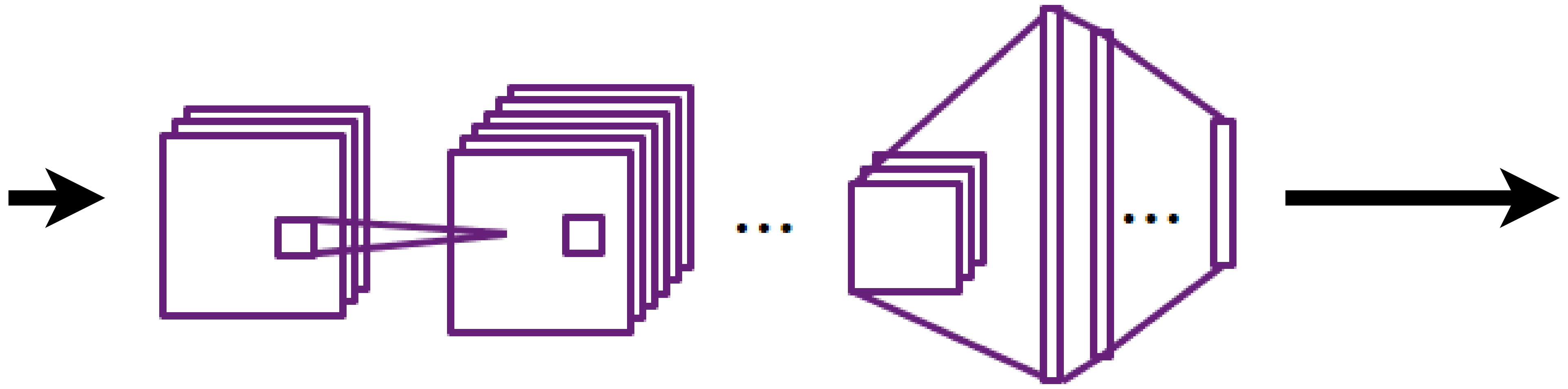

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

# Object **Detection** as Regression Problem



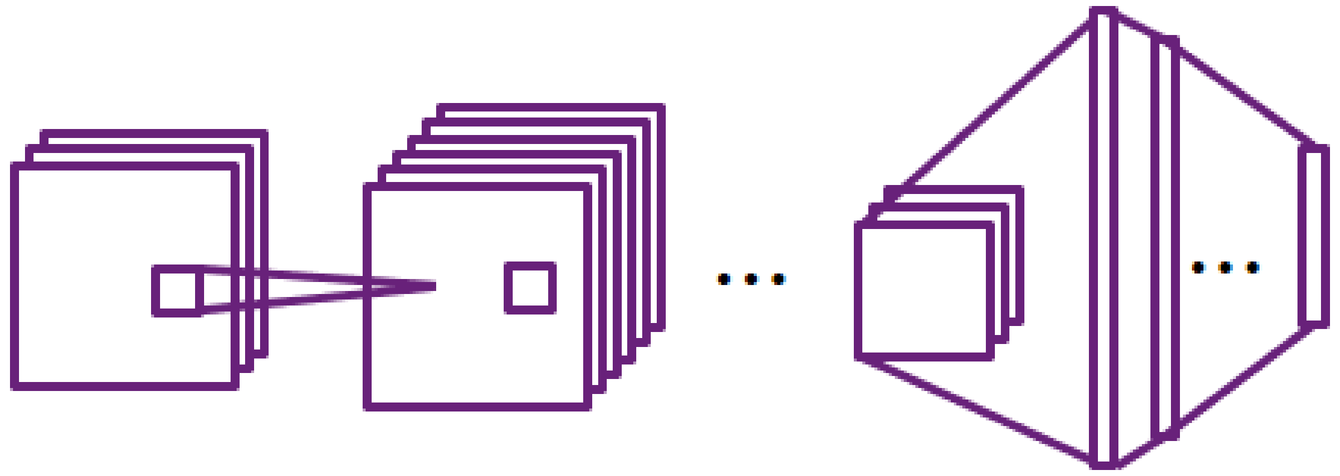
CAT (x, y, w ,h)

**Problem:** each image needs a different number of outputs



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)
…

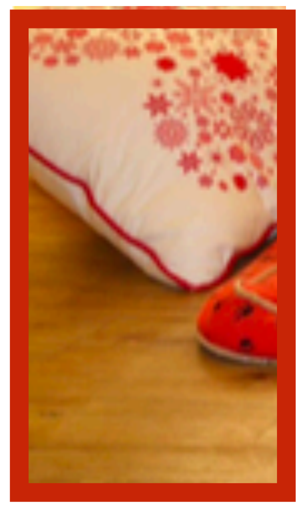
# Object **Detection** as Classification Problem

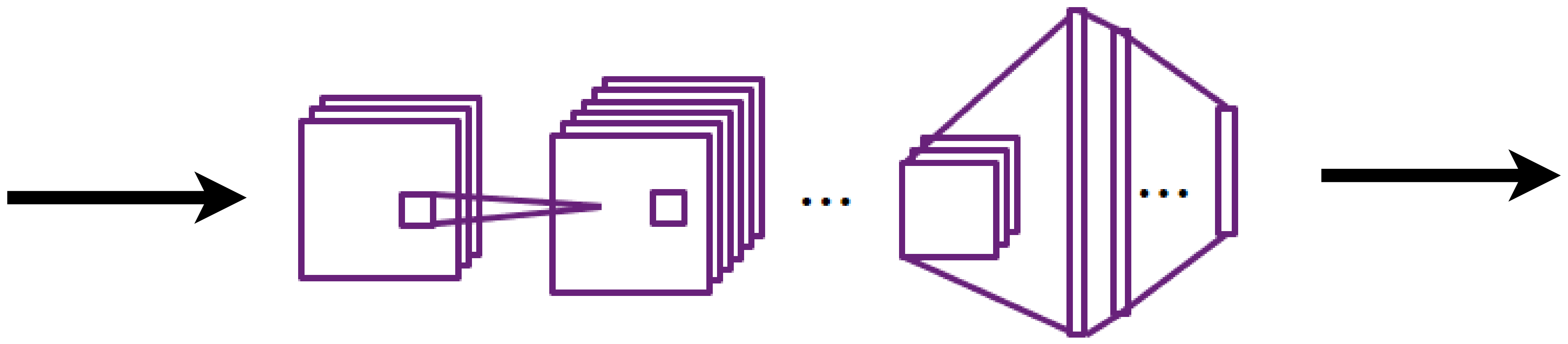


| **Category** | **Prediction** |
|---|---|
| Dog | No |
| Cat | No |
| Couch | No |
| Flowers | No |
| Background | **Yes** |
| … | … |

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

# Object **Detection** as Classification Problem

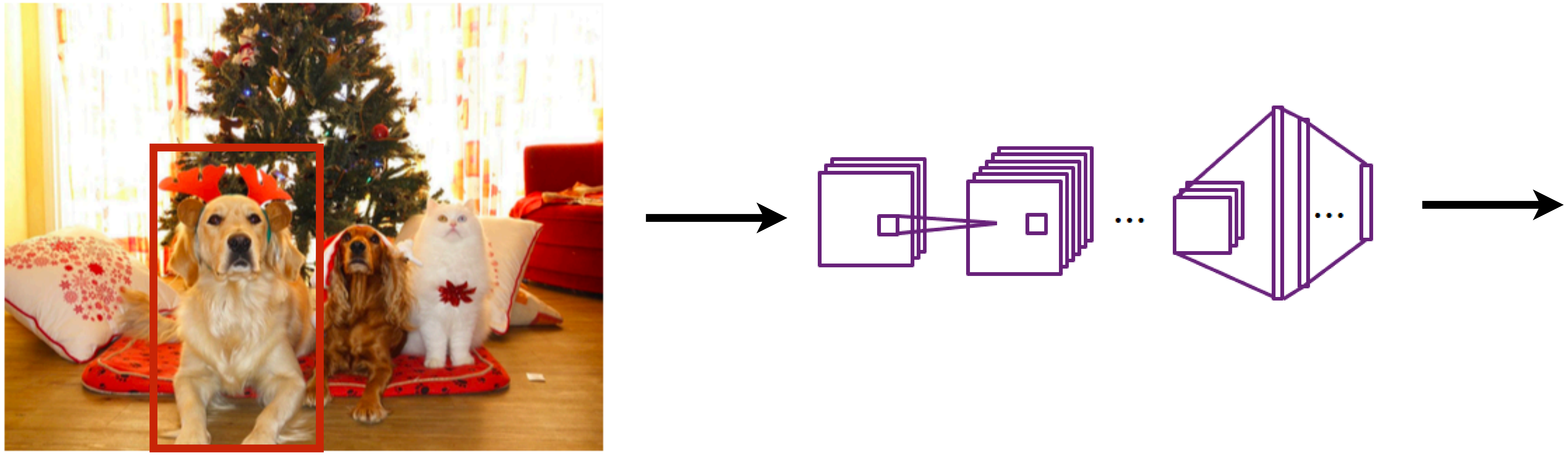




| **Category** | **Prediction** |
|---|---|
| Dog | No |
| Cat | No |
| Couch | No |
| Flowers | No |
| Background | **Yes** |
| … | … |

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

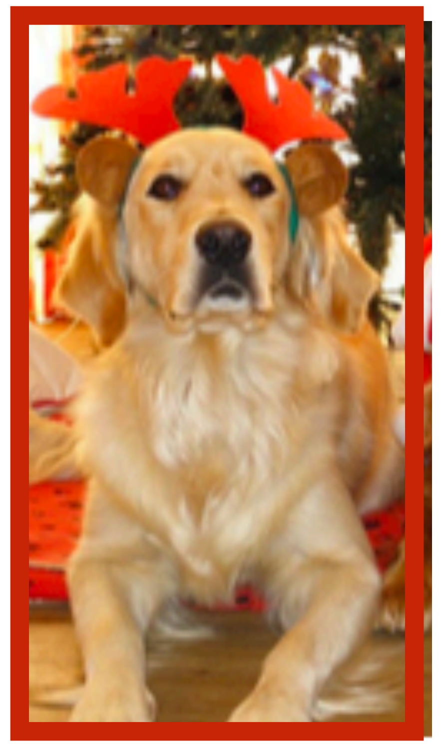
# Object **Detection** as Classification Problem

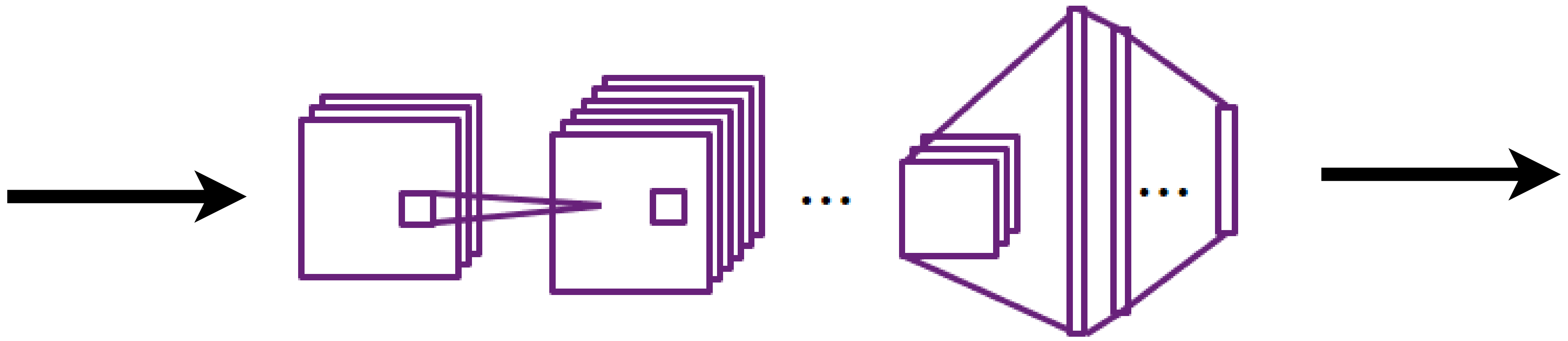


| **Category** | **Prediction** |
|---|---|
| Dog | **Yes** |
| Cat | No |
| Couch | No |
| Flowers | No |
| Background | No |
| … | … |

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

# Object **Detection** as Classification Problem

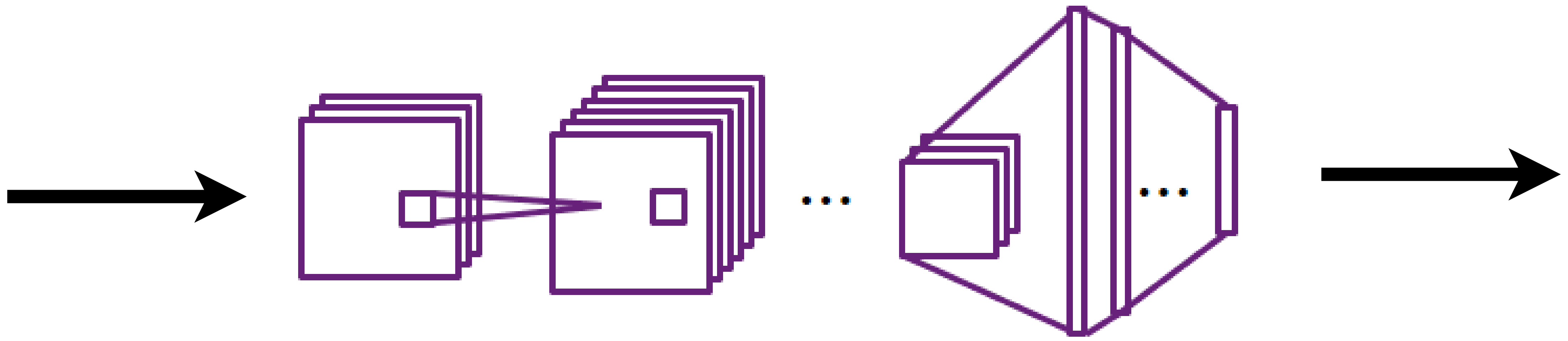





| **Category** | **Prediction** |
| --- | --- |
| Dog | **Yes** |
| Cat | No |
| Couch | No |
| Flowers | No |
| Background | No |
| ... | ... |

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

# Object **Detection** as Classification Problem

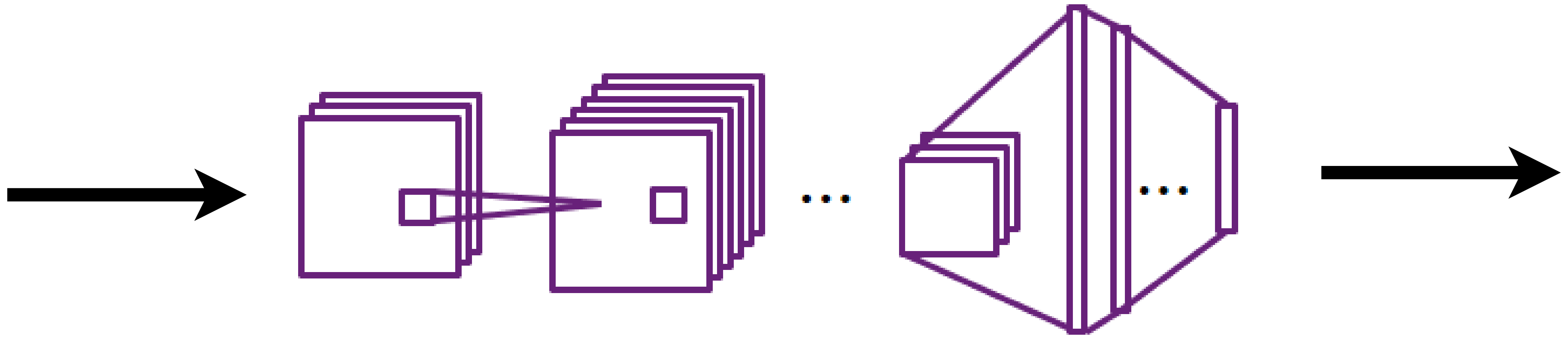


| **Category** | **Prediction** |
|---|---|
| Dog | No |
| Cat | **Yes** |
| Couch | No |
| Flowers | No |
| Background | No |
| … | … |

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)

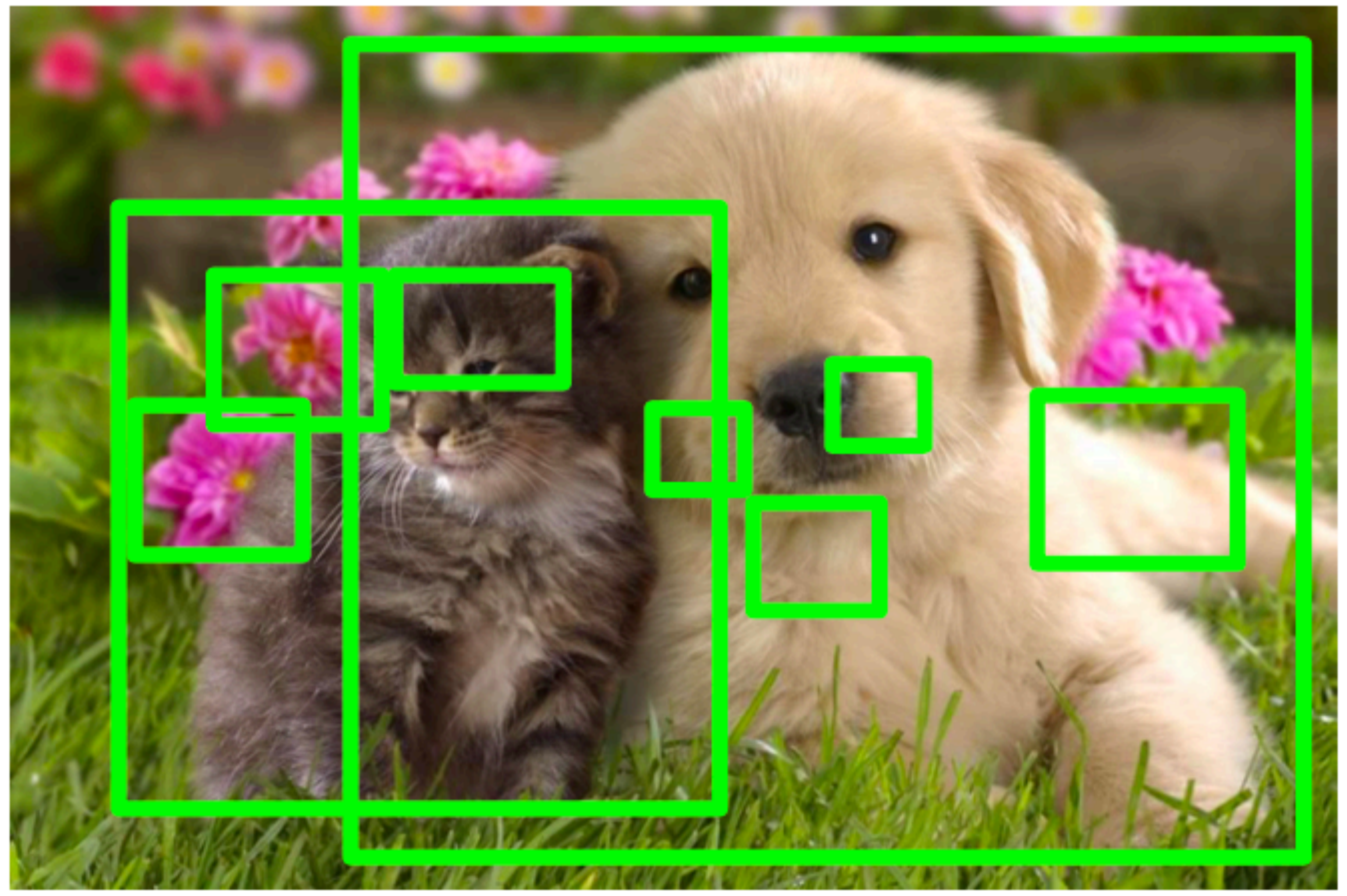
[ Alexe et al, TPAMI 2012 ]
[ Uijkings et al, IJCV 2013 ]
[ Cheng et al, CVPR 2014 ]
[ Zitnick and Dollar, ECCV 2014 ]

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

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

# R-CNN

[ Girshick et al, CVPR 2014 ]

Input **Image**

* image from Ross Girshick

# R-CNN

[ Girshick et al, CVPR 2014 ]

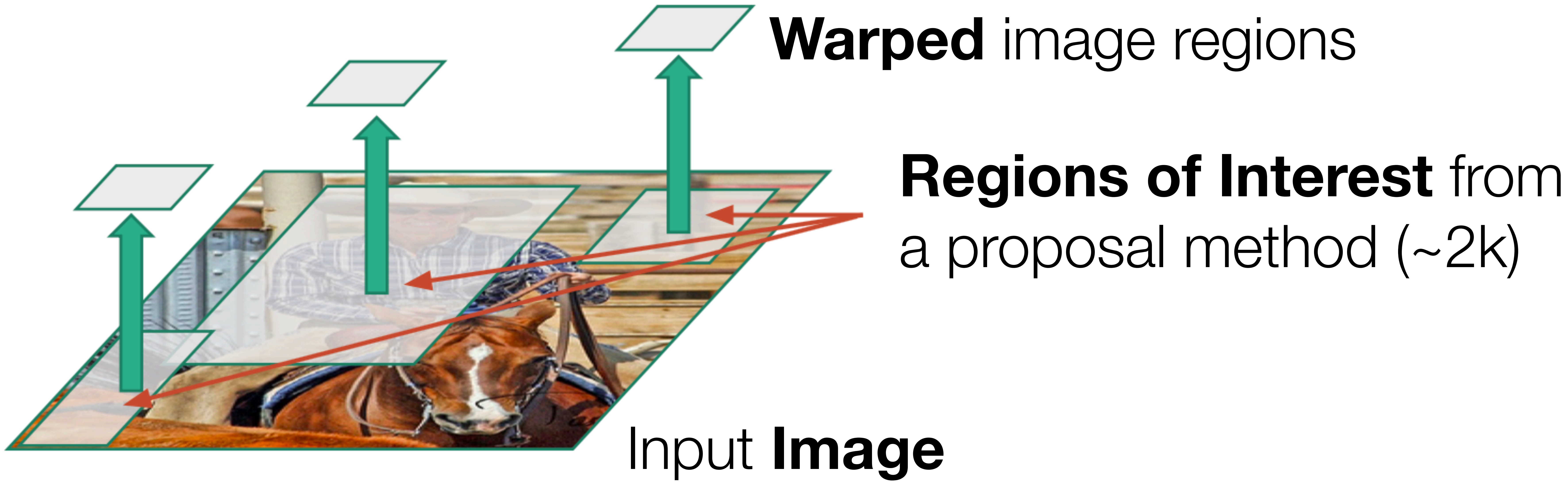
**Regions of Interest** from
a proposal method (~2k)

Input **Image**

* image from Ross Girshick

# R-CNN

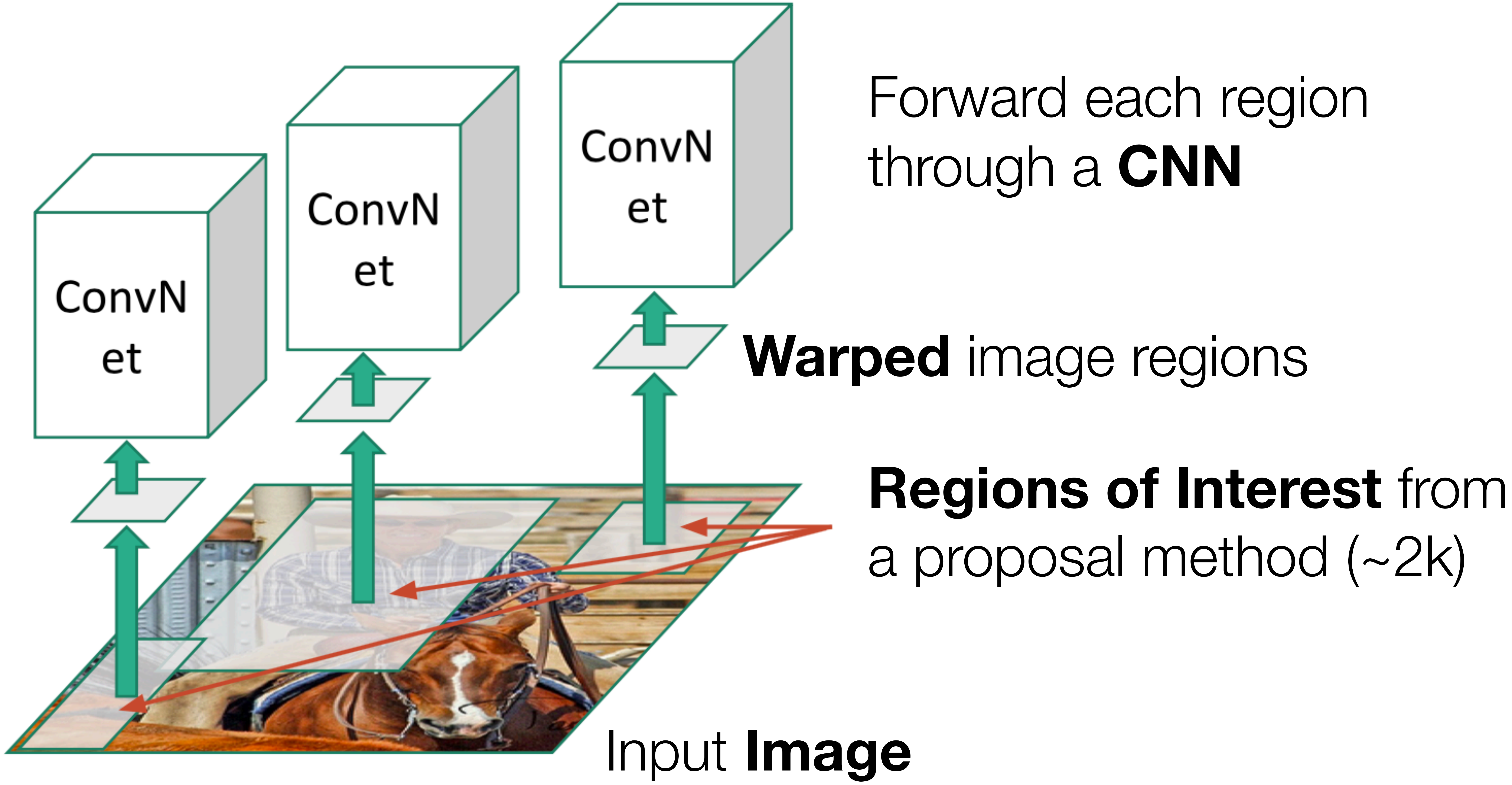
[ Girshick et al, CVPR 2014 ]




* image from Ross Girshick

# R-CNN

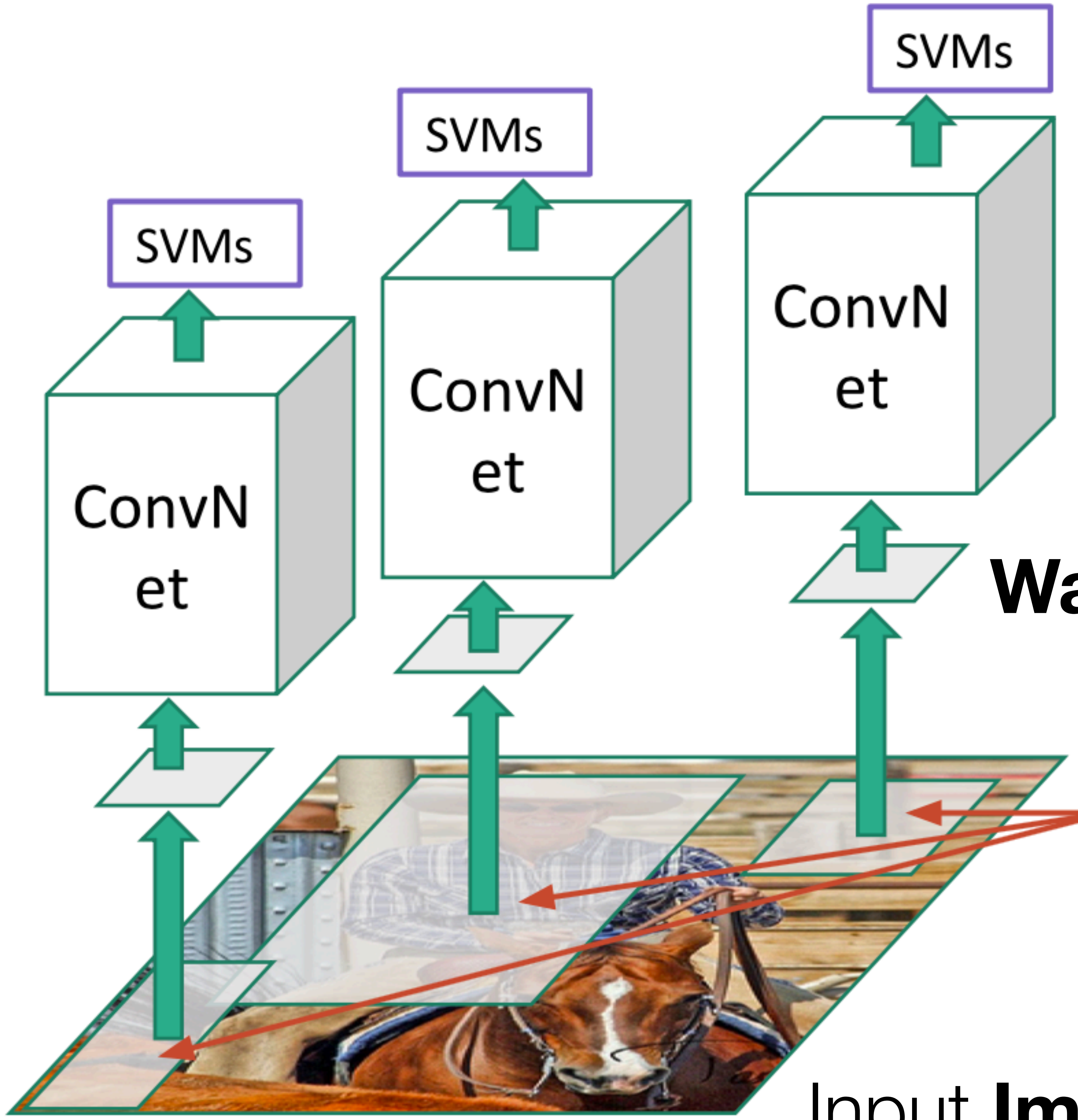
[ Girshick et al, CVPR 2014 ]




* image from Ross Girshick

# R-CNN

[ Girshick et al, CVPR 2014 ]




**Classify** regions with SVM

Forward each region through a **CNN**

**Warped** image regions

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

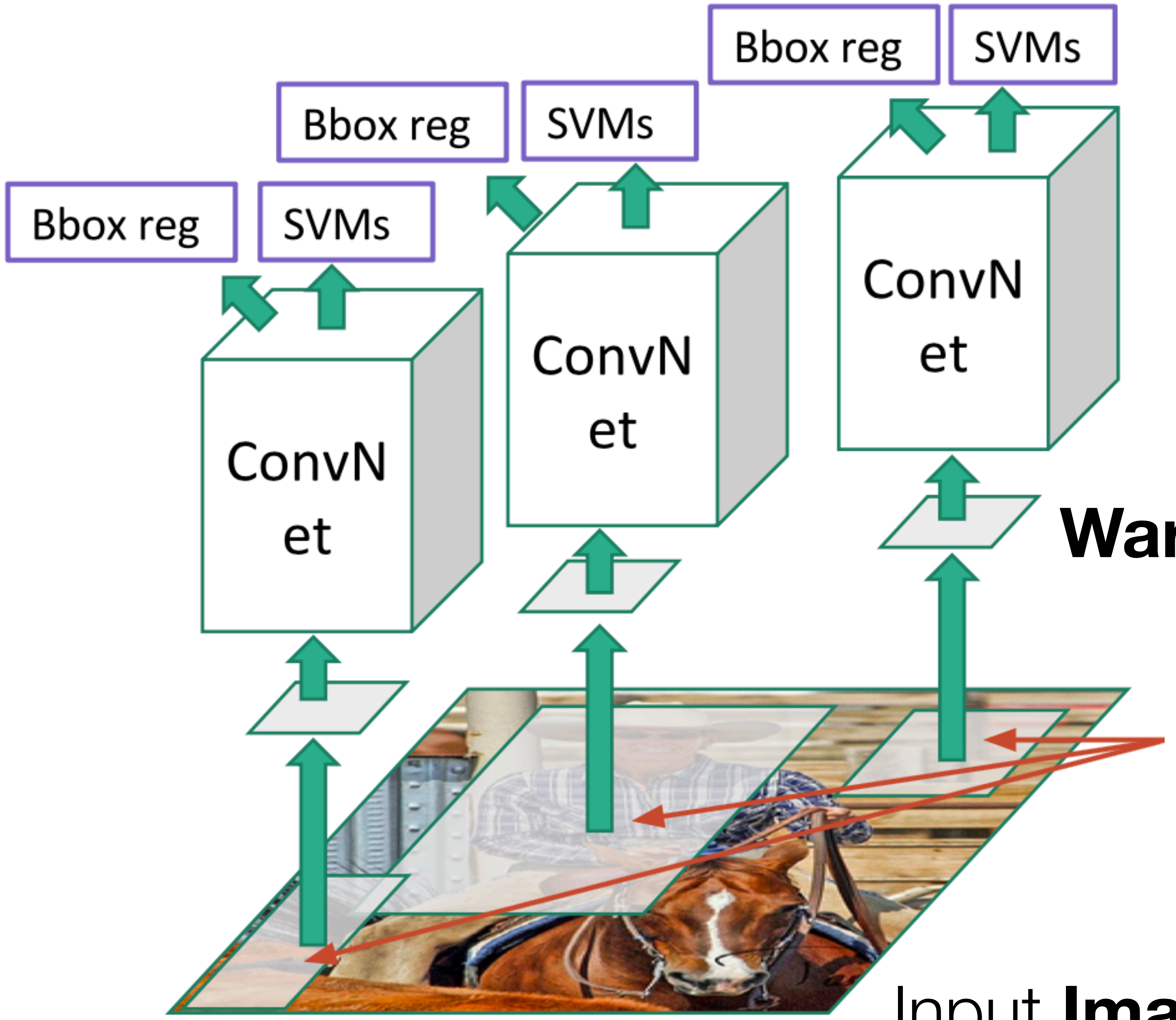
Input **Image**

* image from Ross Girshick

# R-CNN

**Linear Regression** for bounding box offsets

[ Girshick et al, CVPR 2014 ]




**Classify** regions with SVM

Forward each region through a **CNN**

**Warped** image regions

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

Input **Image**

* image from Ross Girshick

# R-CNN

R-CNN (Regions with CNN features) algorithm:

— Extract promising candidate regions using an object proposals algorithm

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

— Input each resized image patch to the convolutional neural network

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

# Summary

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

# Thank you!