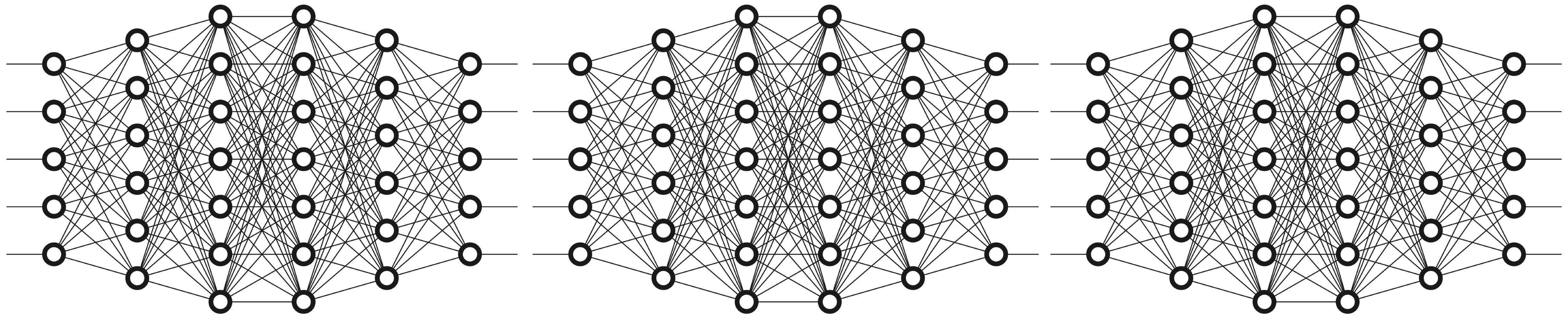
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 23: Detection, Segmentation

Menu for Today

Topics:

— Classification, Detection, Segmentation

— Attention, Transformers

Redings:

— **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** is due **Thursday**

Computer **Vision Problems**

Categorization

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

IMAGENET

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

Computer **Vision Problems**

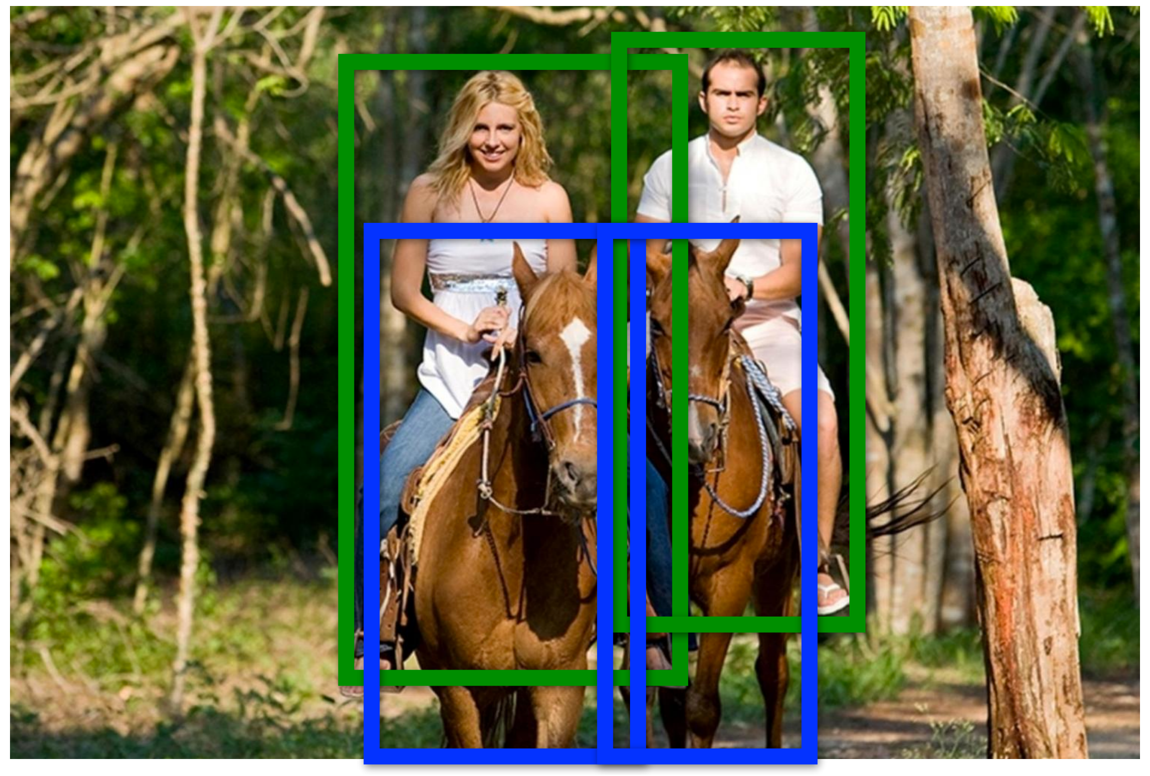
Categorization

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**

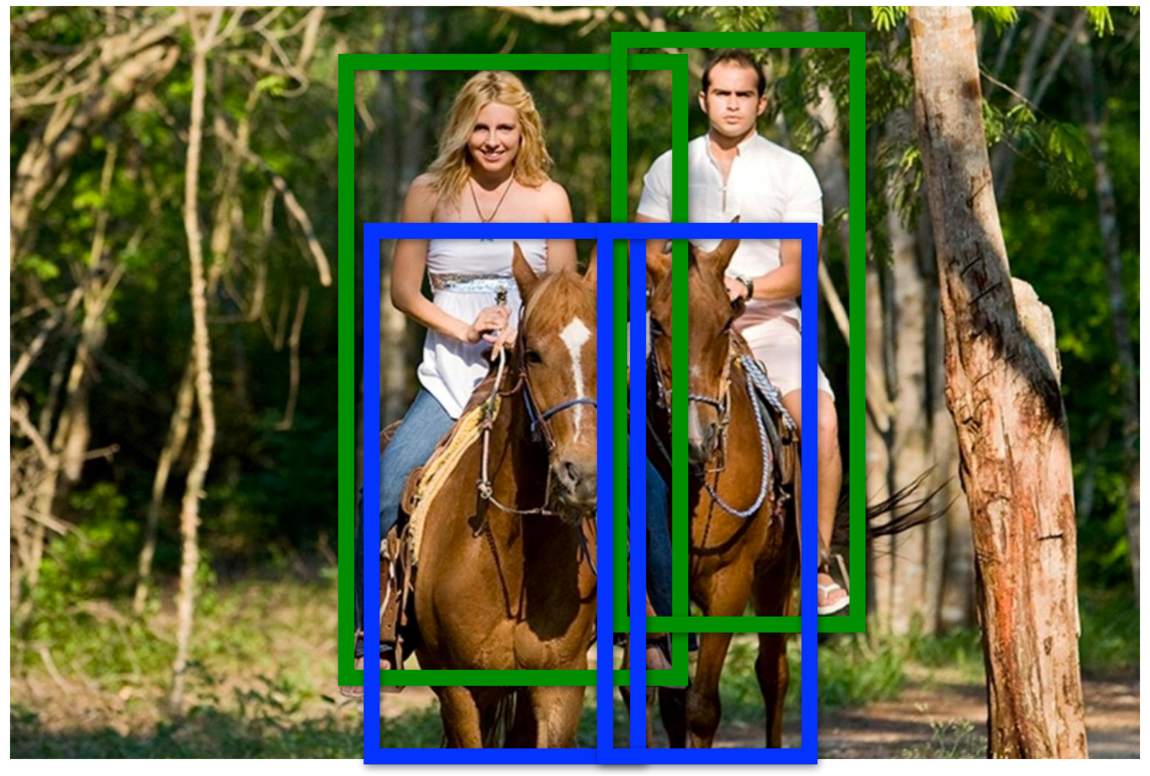
Categorization

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**

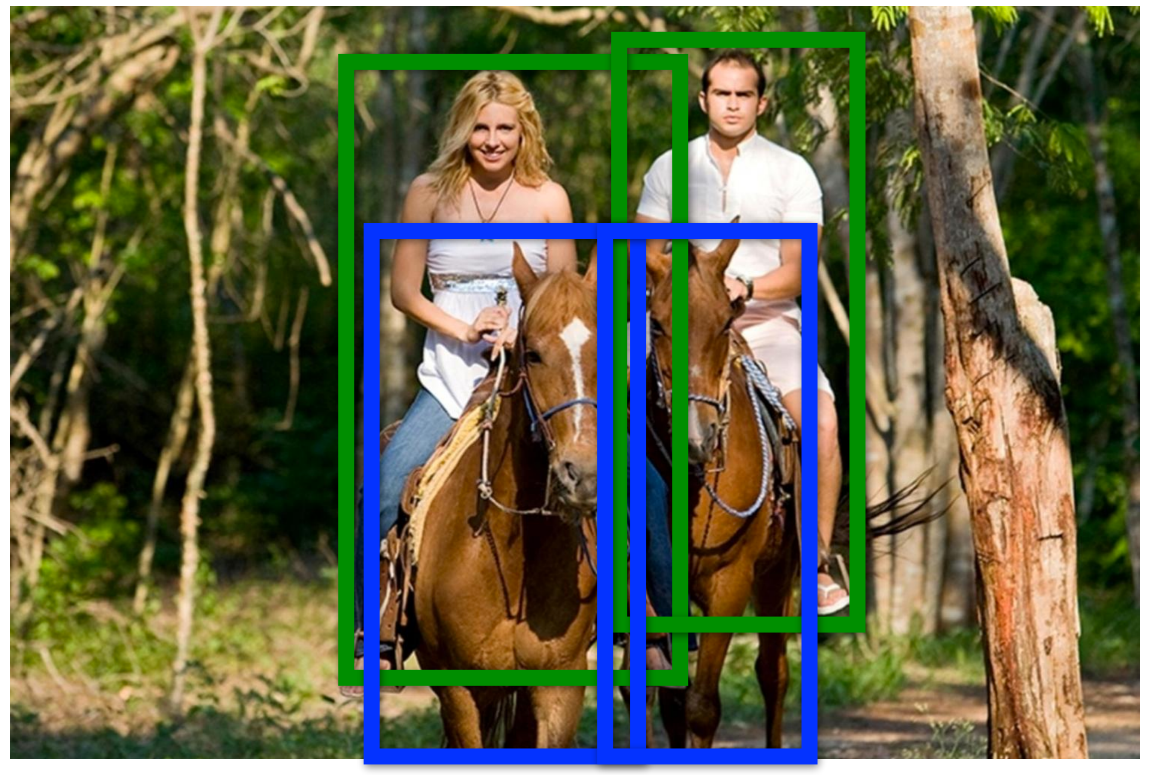
Categorization

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

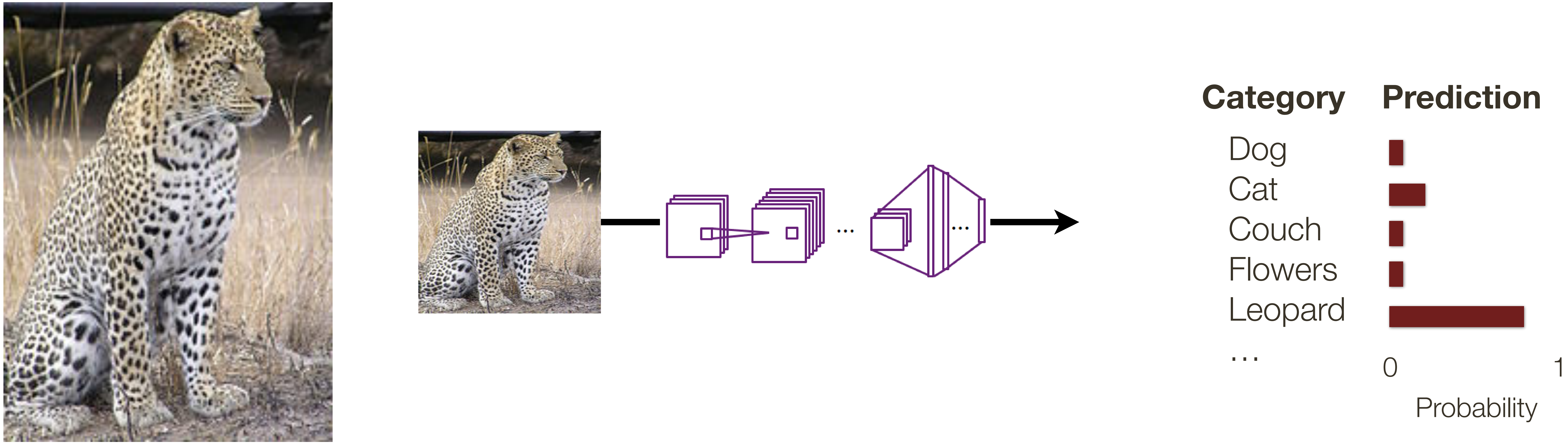
Horse
Person

COCO Common Objects in Context

Instance Segmentation

Horse1
Horse2
Person1
Person2

Object **Classification**



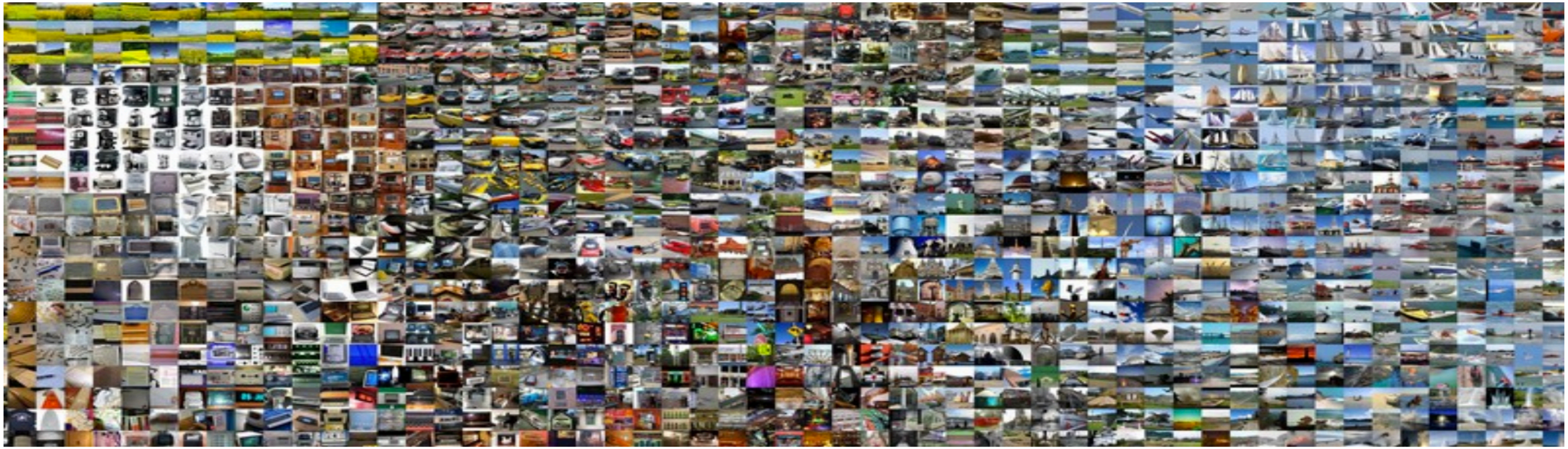
Problem: For each image predict which category it belongs to out of a fixed set

ImageNet **Competition** (ILSVRC)

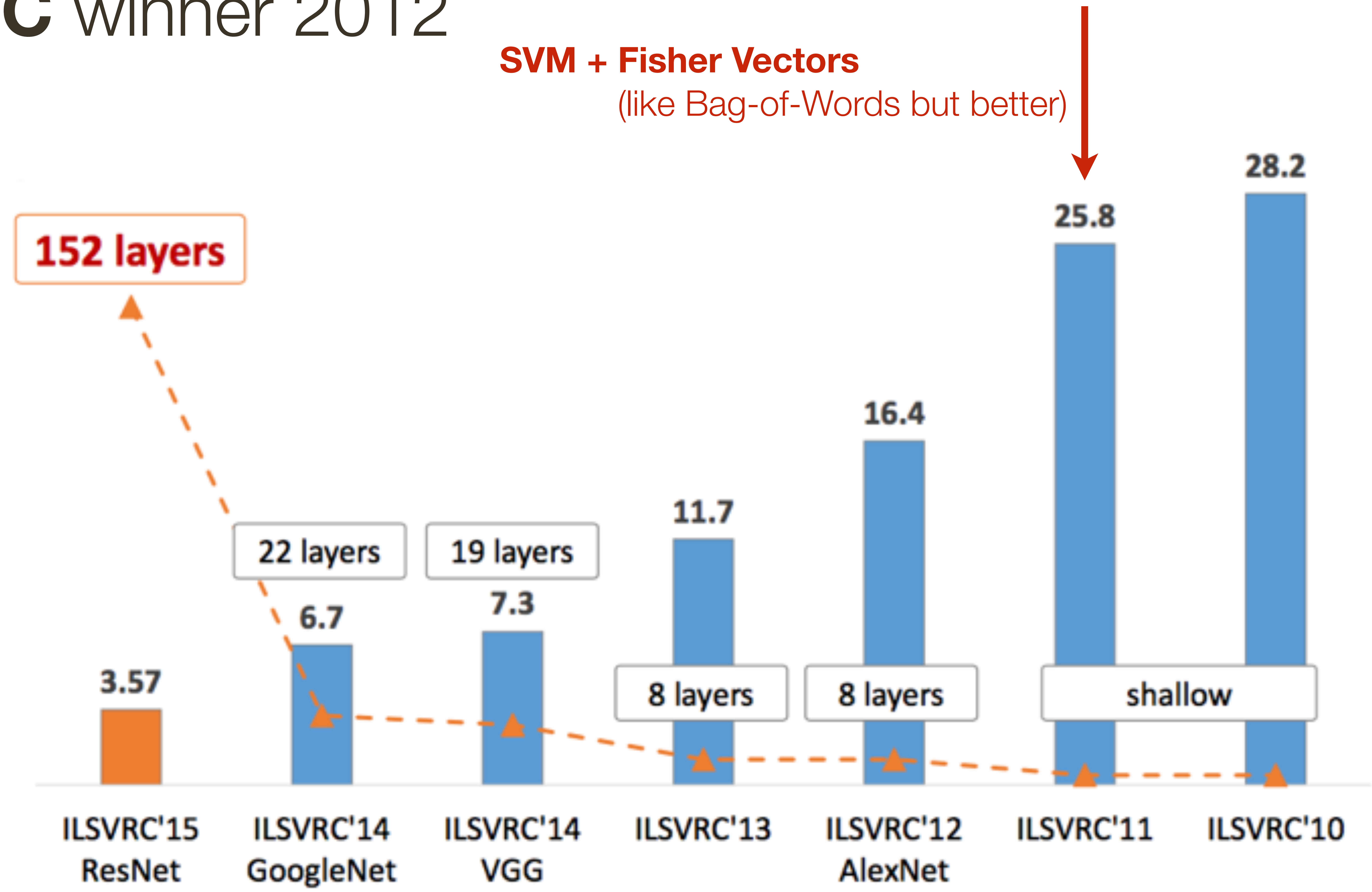
Annual competition of image classification at scale

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

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

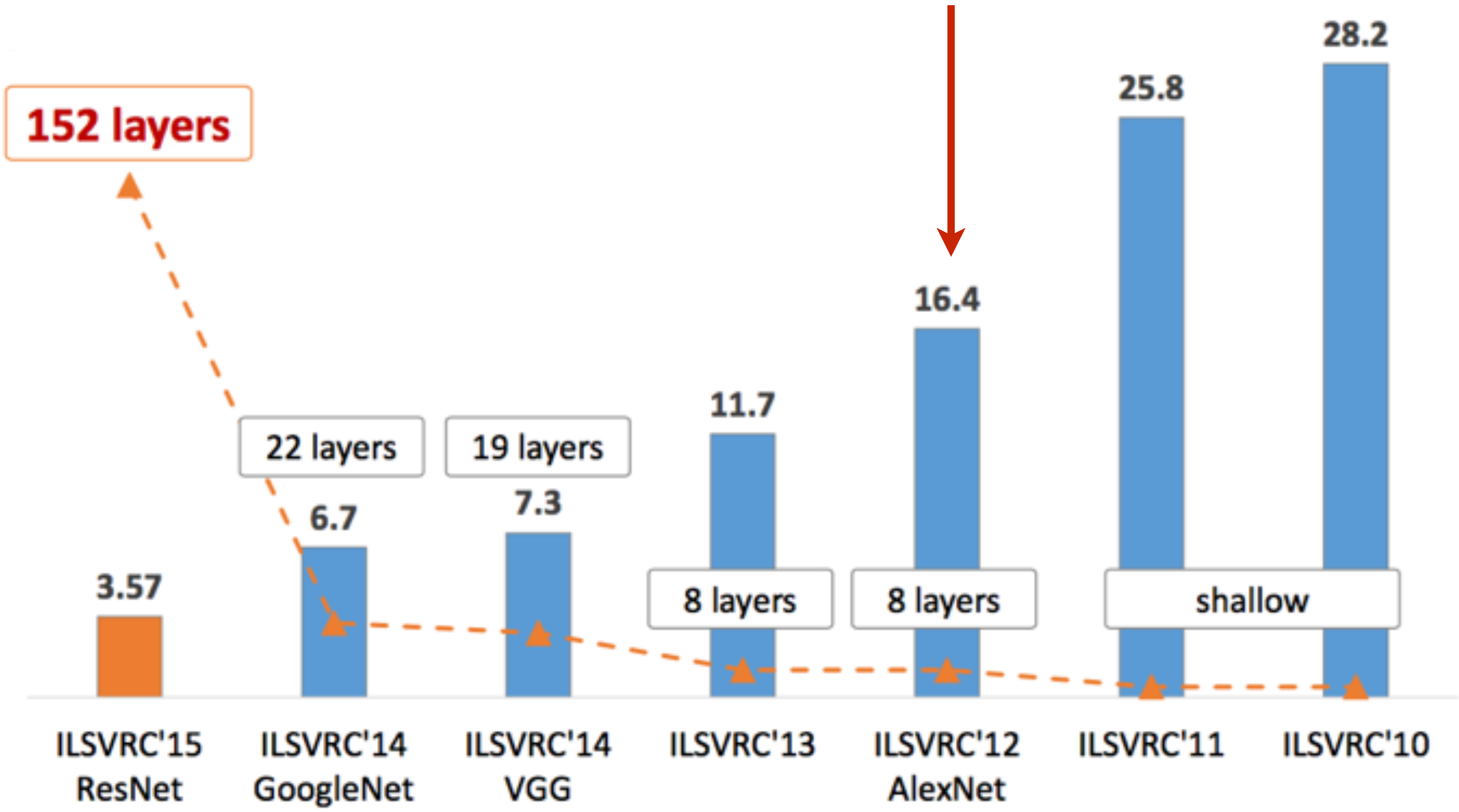


ILSVRC winner 2012



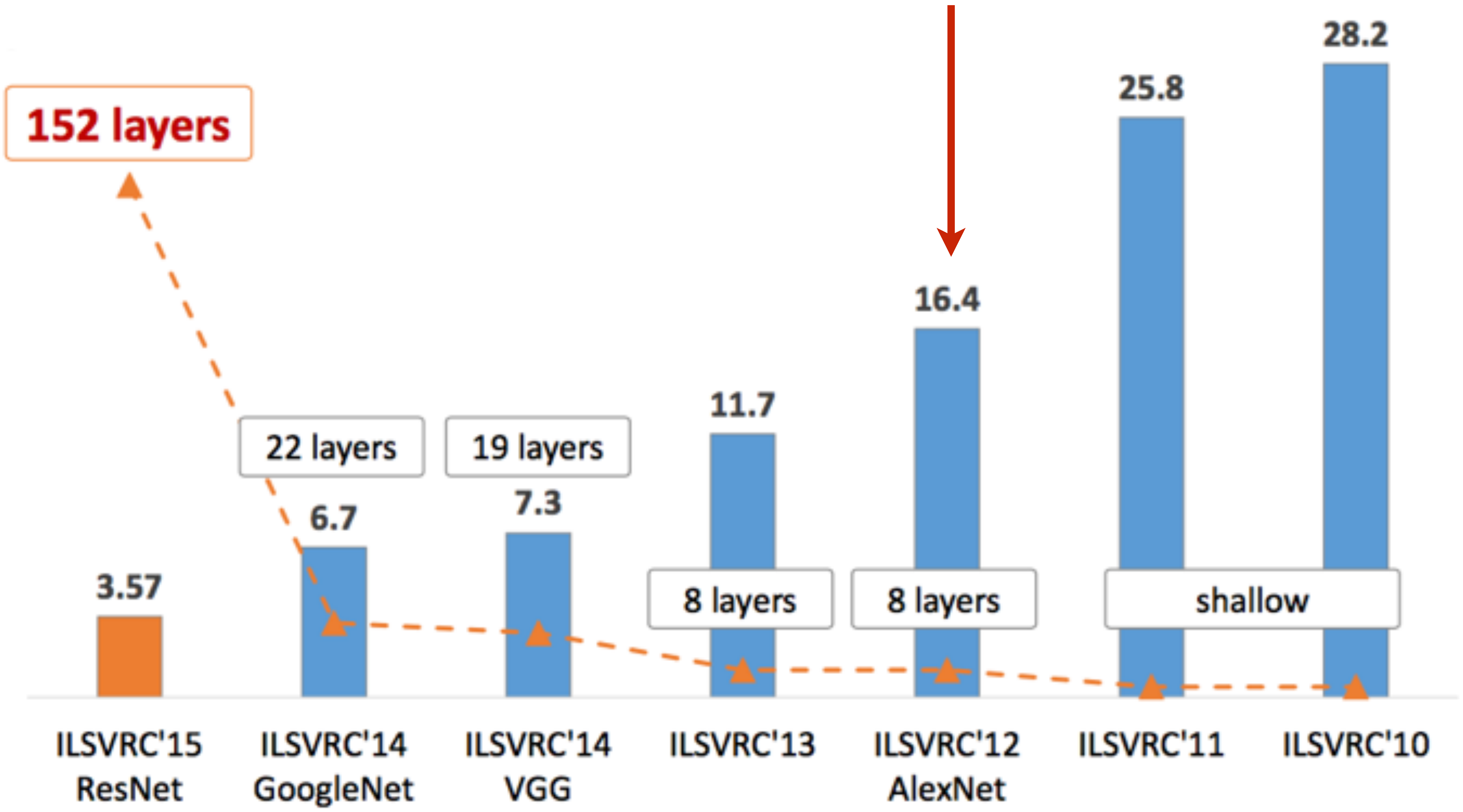
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

ILSVRC winner 2012



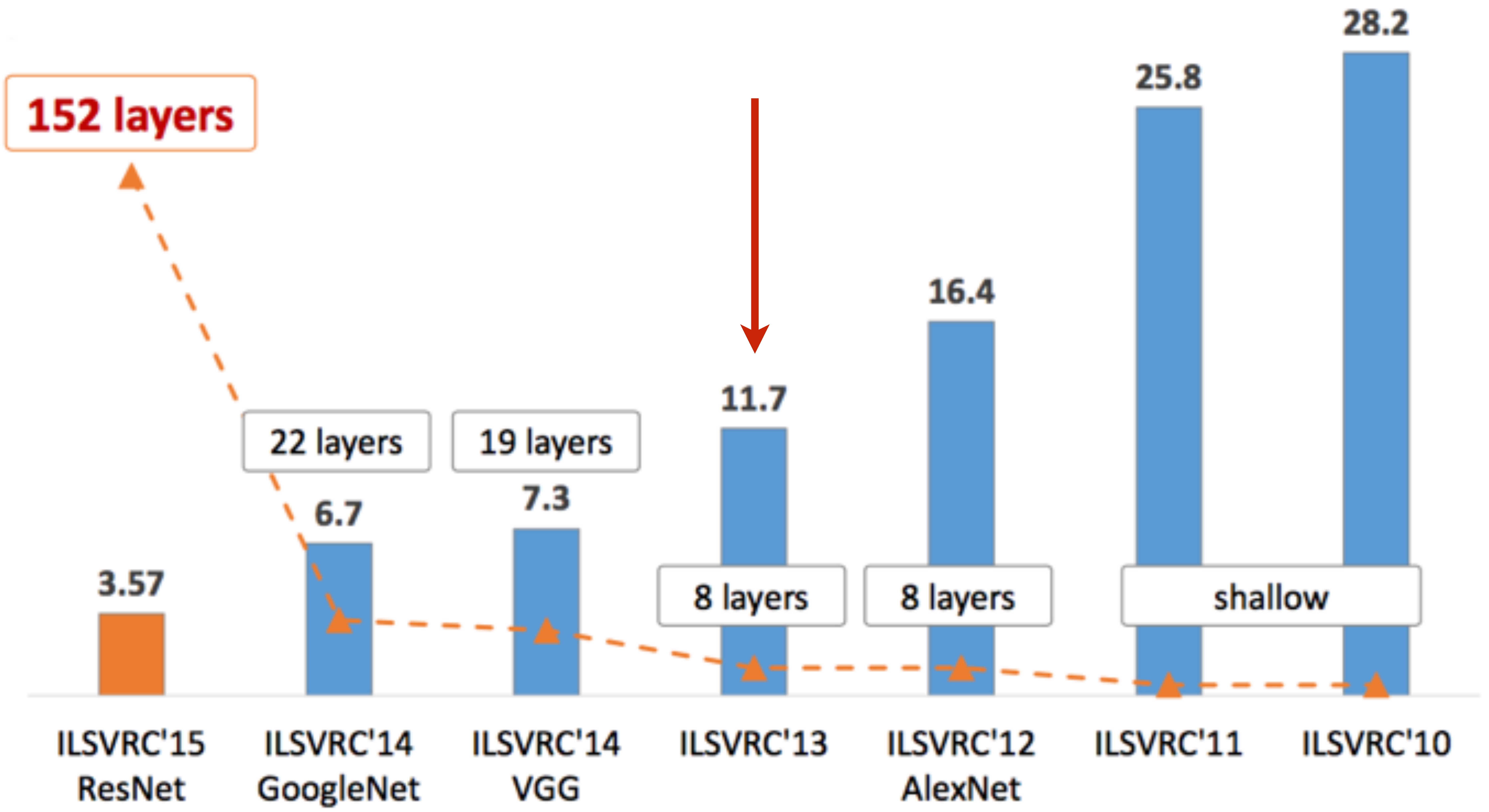
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

ILSVRC winner 2012



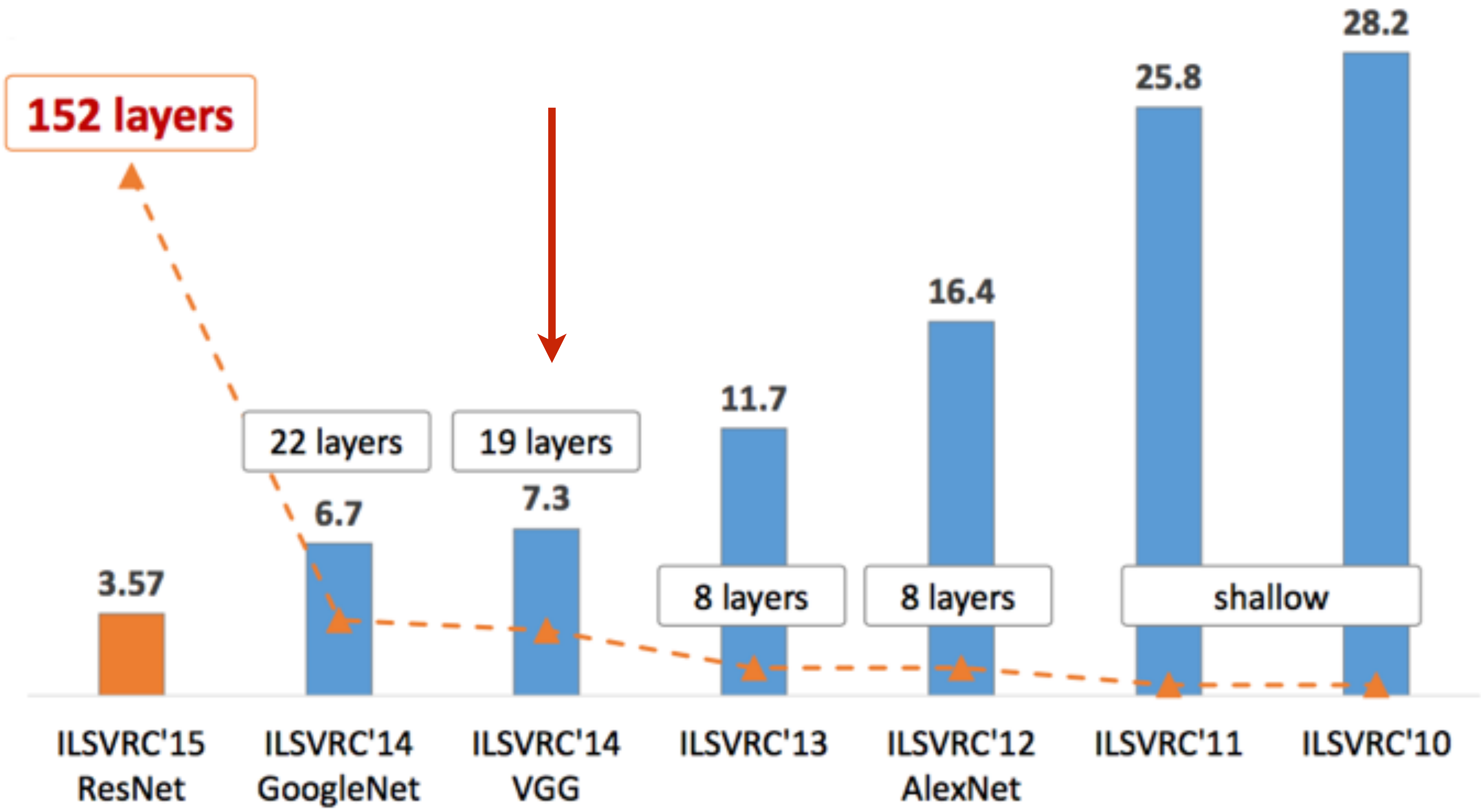
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

ILSVRC winner 2012



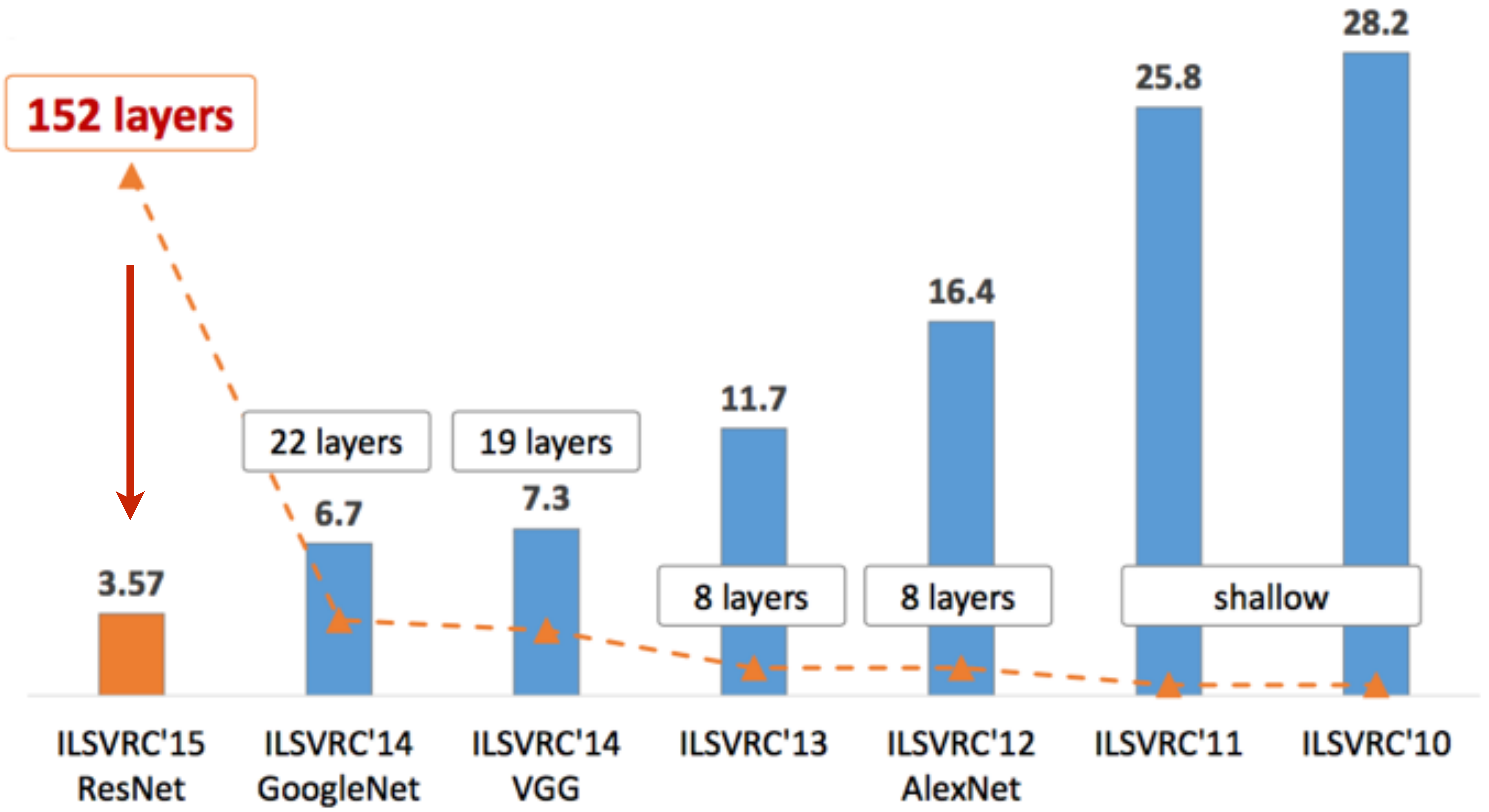
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

ILSVRC winner 2012



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

ILSVRC winner 2012



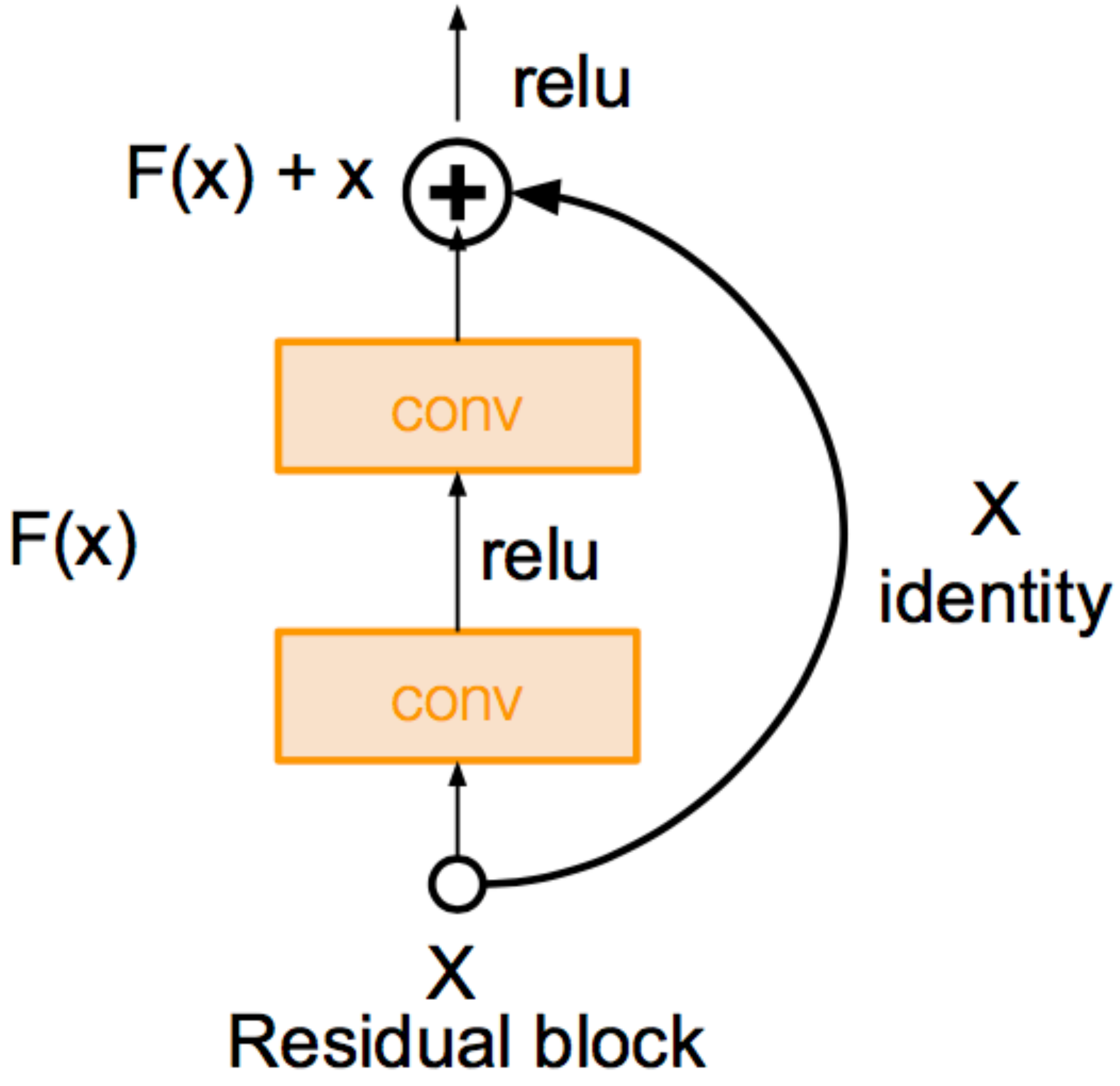
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

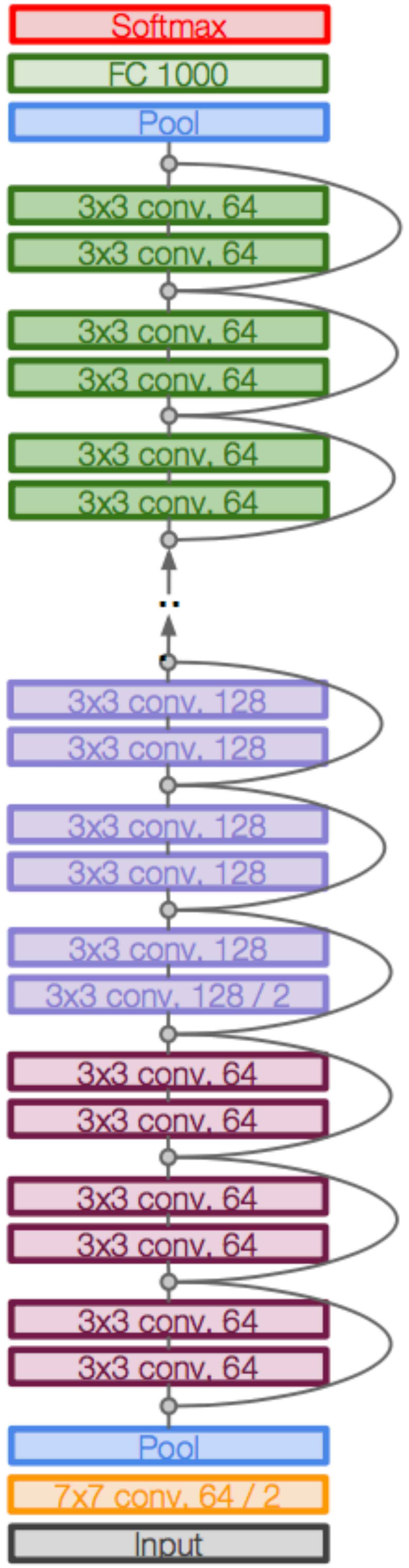
ResNet

[He et al., 2015]

even deeper — **152 layers**!

using residual connections



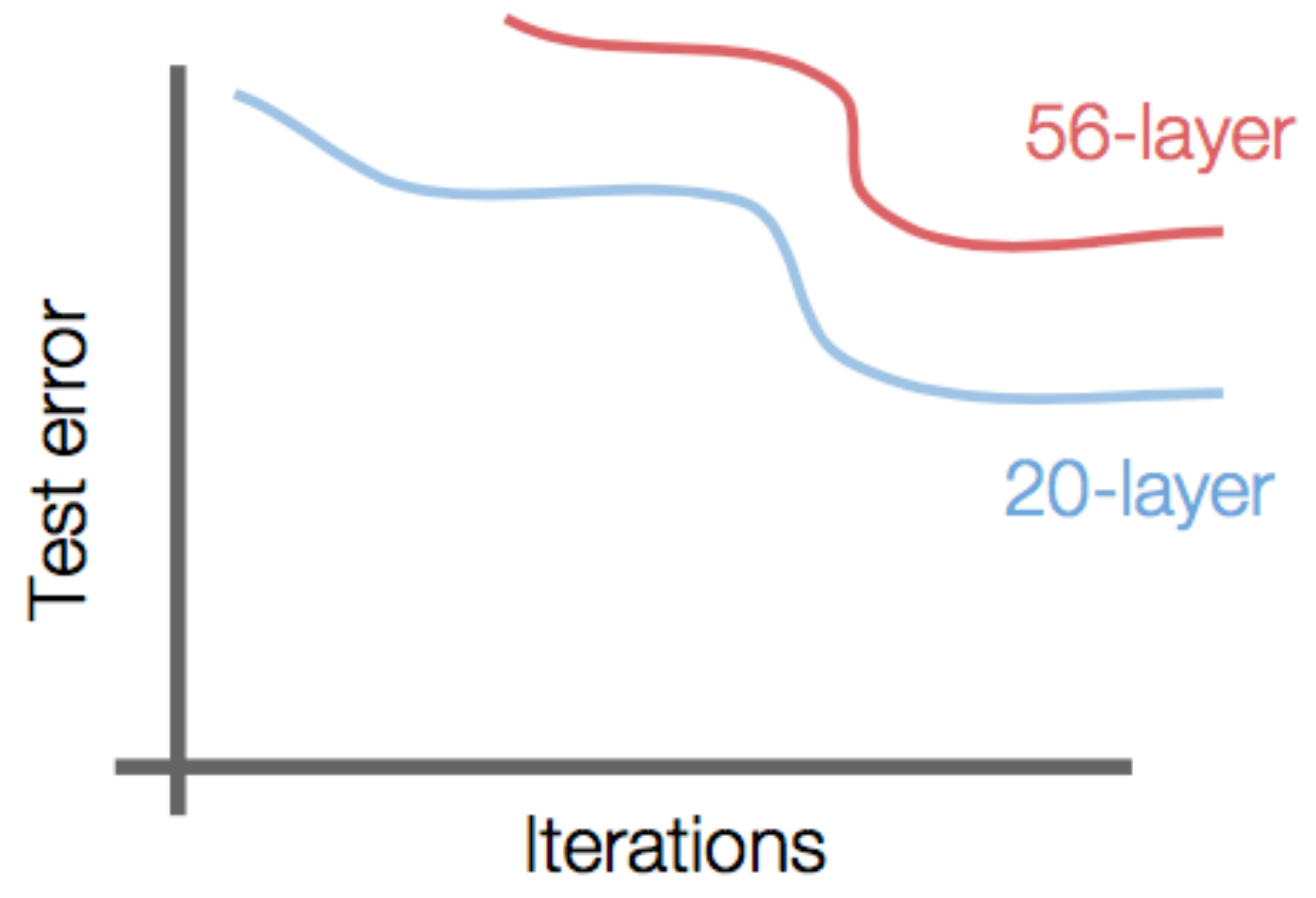


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

ResNet: Motivation

[He et al., 2015]

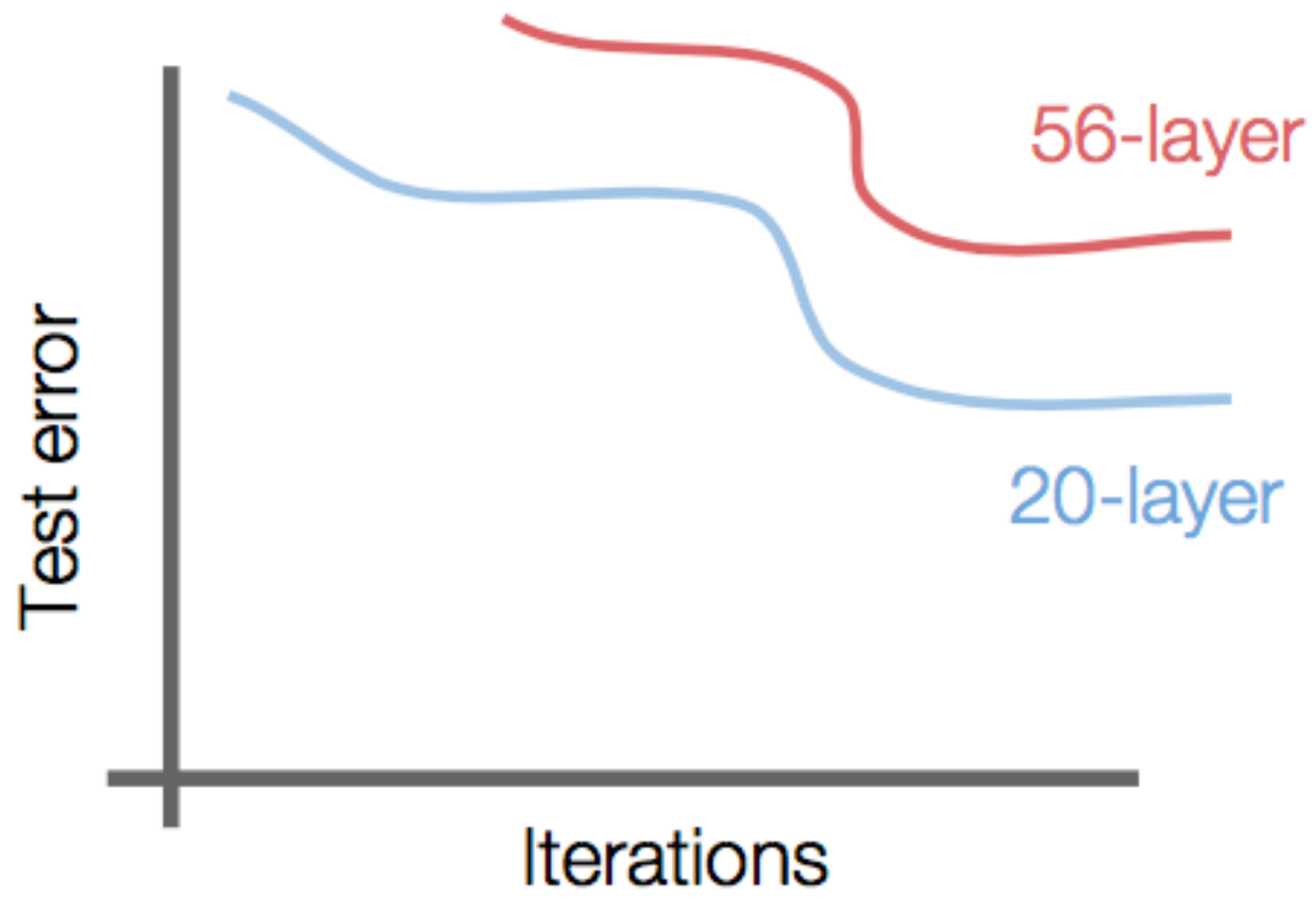
What happens when we continue to stacking deeper layers on a "plain" CNN



ResNet: Motivation

[He et al., 2015]

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



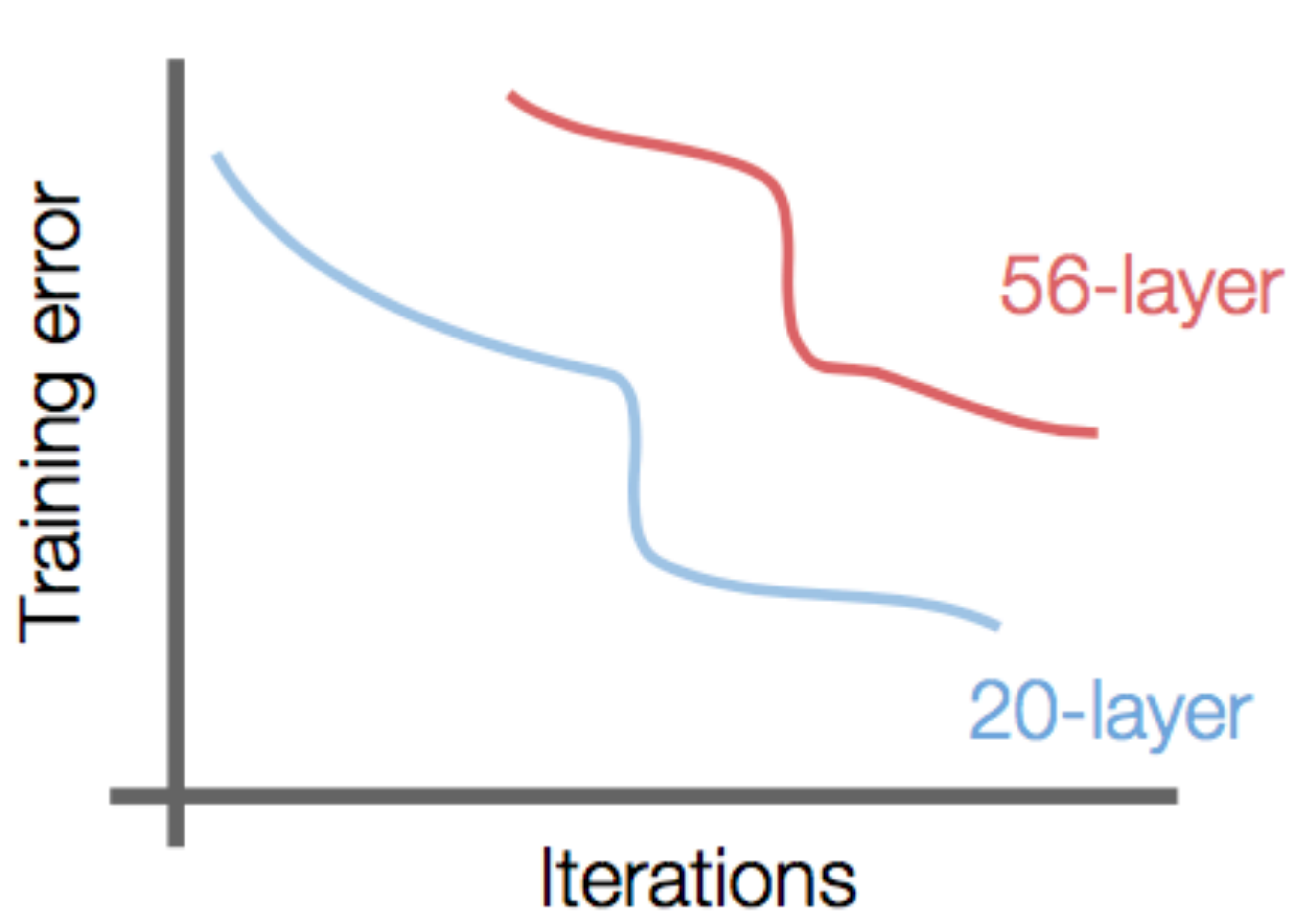
Whats the **problem**?

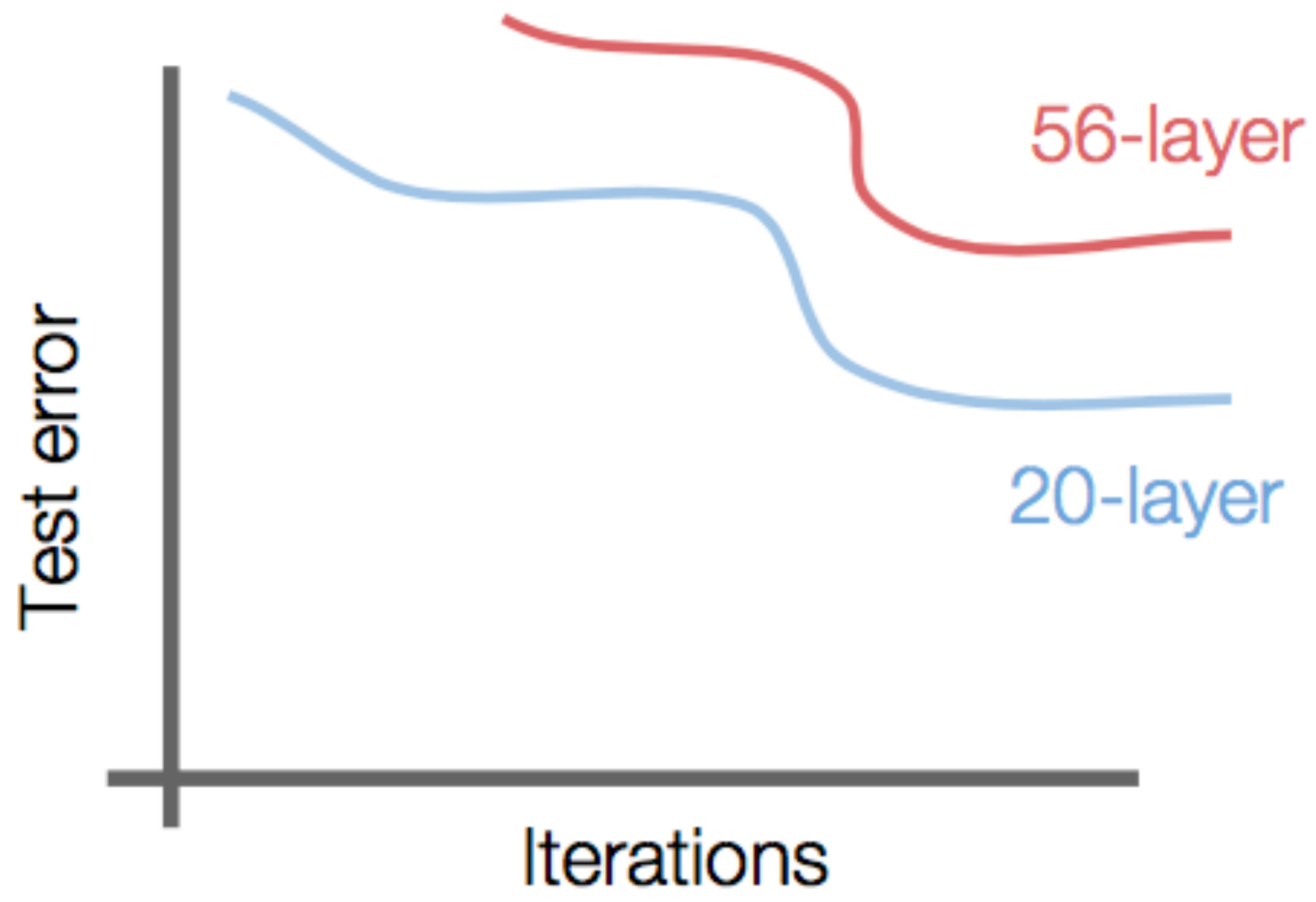
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

ResNet: Motivation

[He et al., 2015]

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

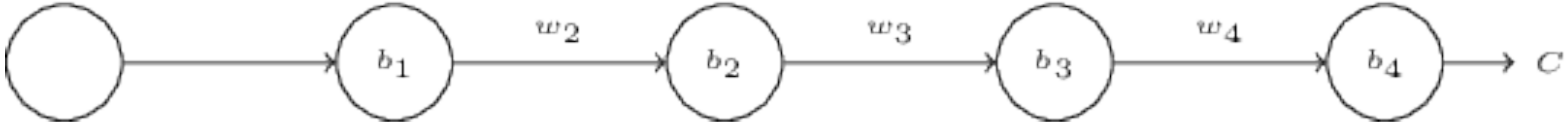




Whats the **problem**?

Optimizing **Deep** Neural Networks

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



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

common terms

$$\frac{\partial C}{\partial b_3} = \sigma'(z_3)\overbrace{w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}}$$

This is called vanishing gradient problem

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

Source: http://neuralnetworksanddeeplearning.com/chap5.html

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

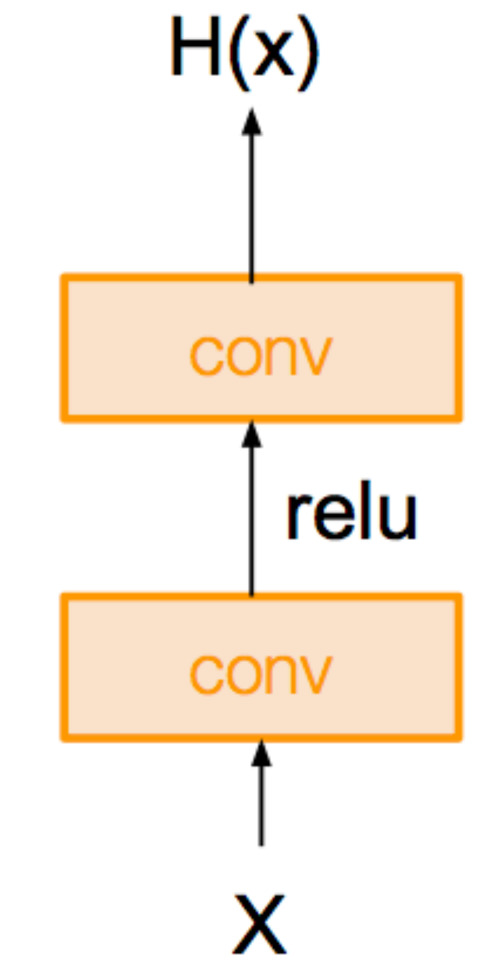
ResNet

[He et al., 2015]

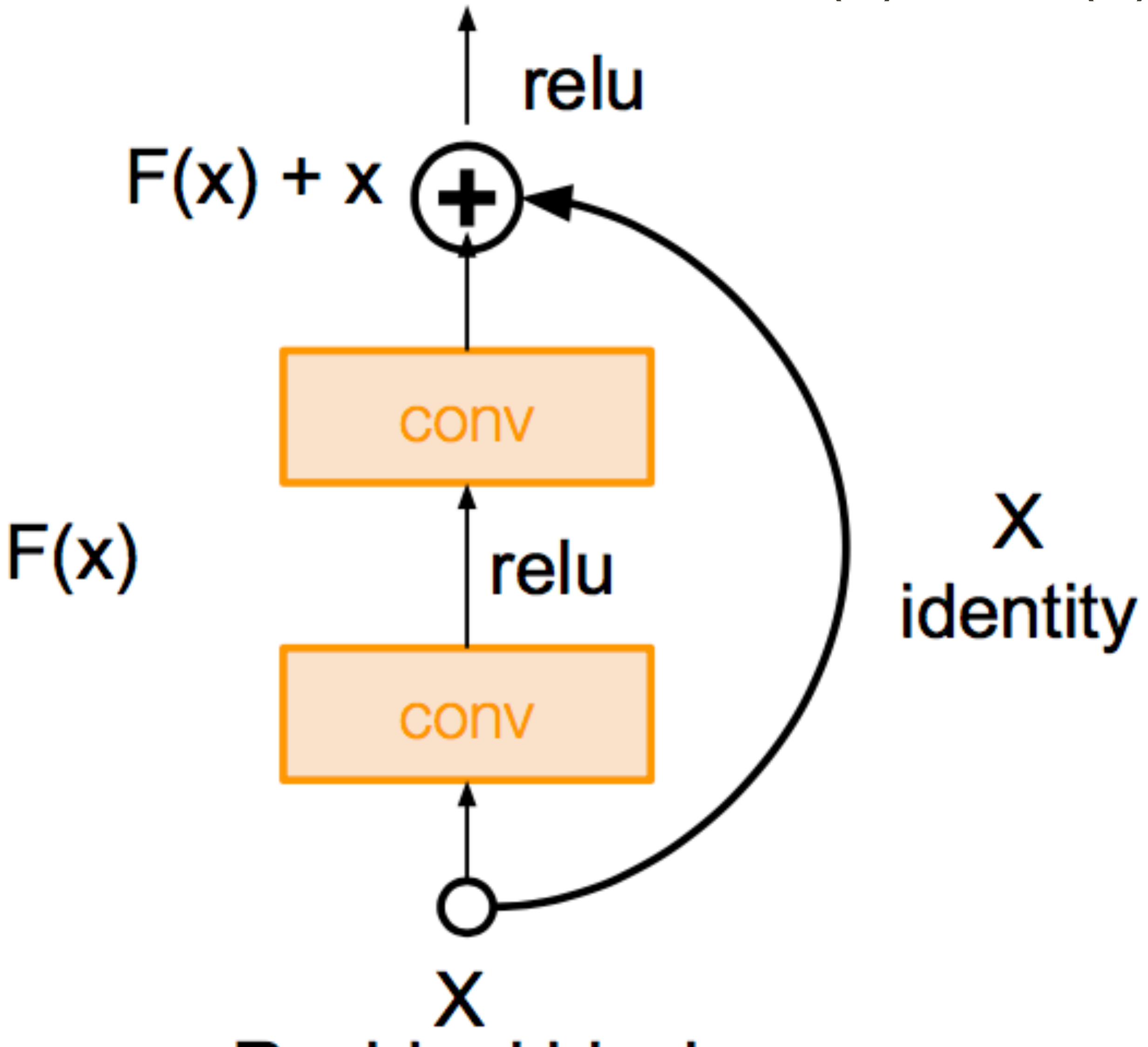
Solution: use network to fit residual mapping instead of directly trying to fit a desired underlying mapping

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

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



"Plain" layers



Residual block

ResNet

[He et al., 2015]

Full details

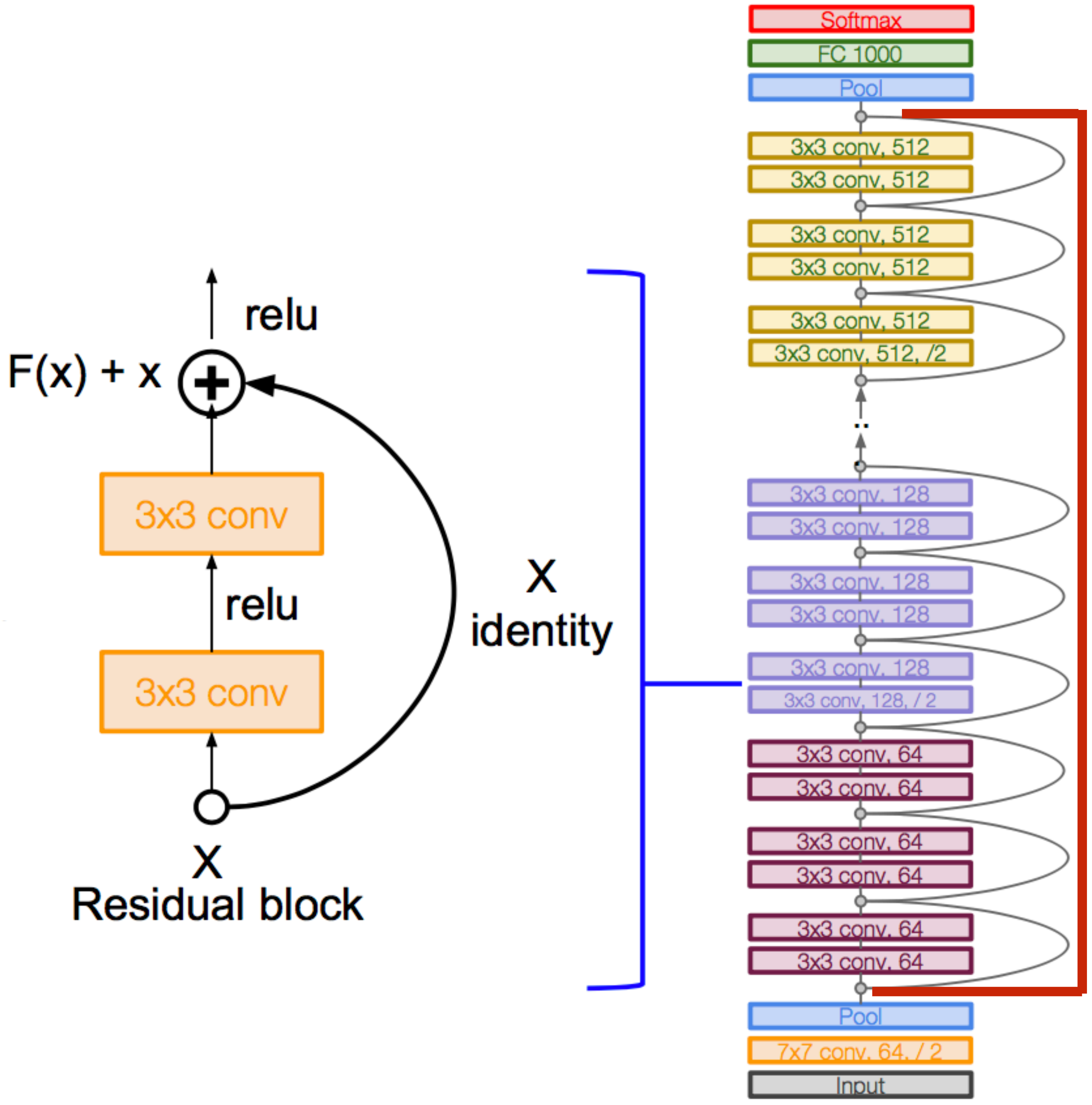
— Stacked **residual blocks**

— Every residual block consists of **two 3x3 filters**

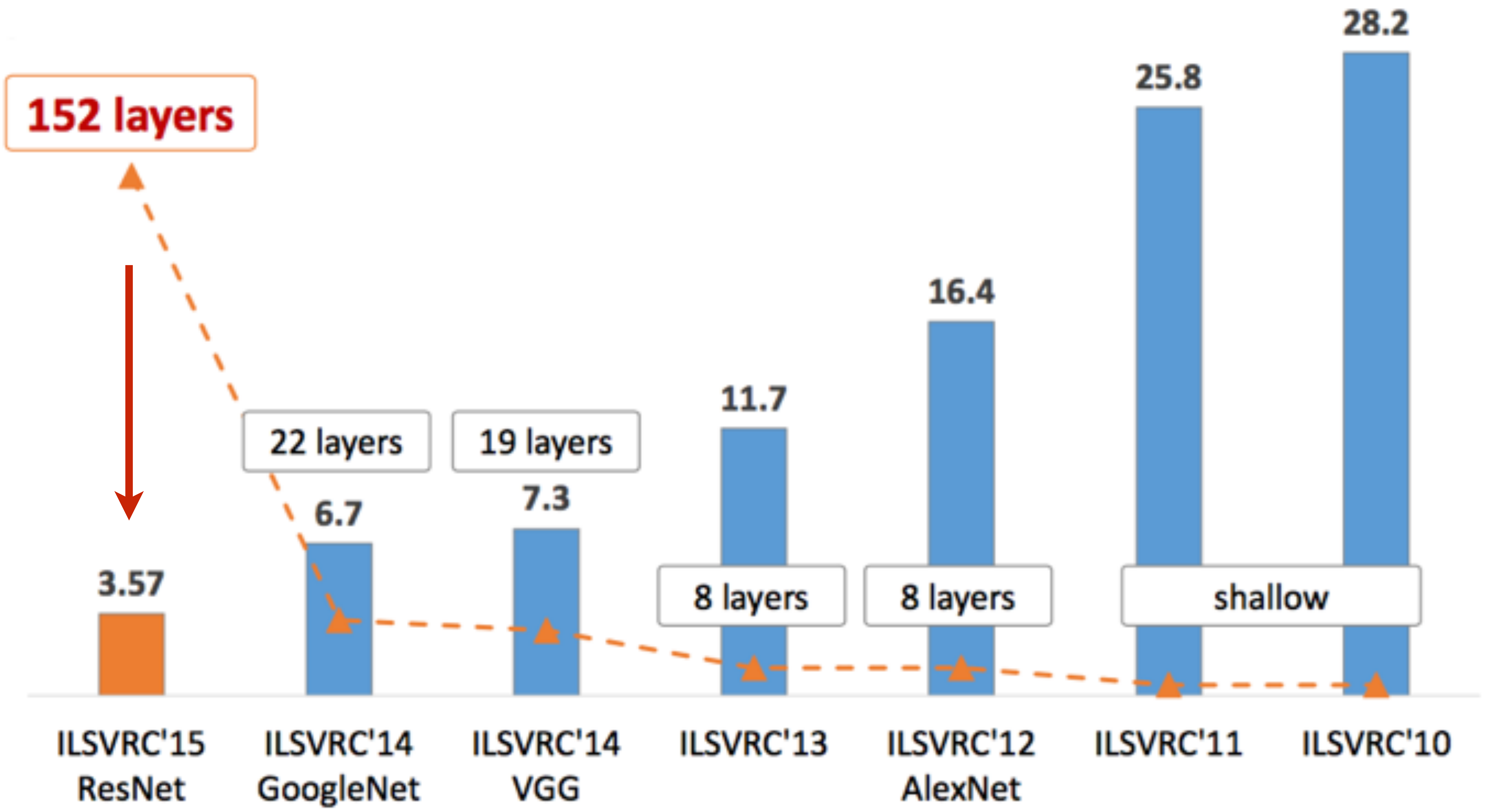
— Periodically double # of filters and downsample spatially using stride of 2

— Additional convolutional layer in the beginning

— **No FC layers** at the end (only FC to output 1000 classes)

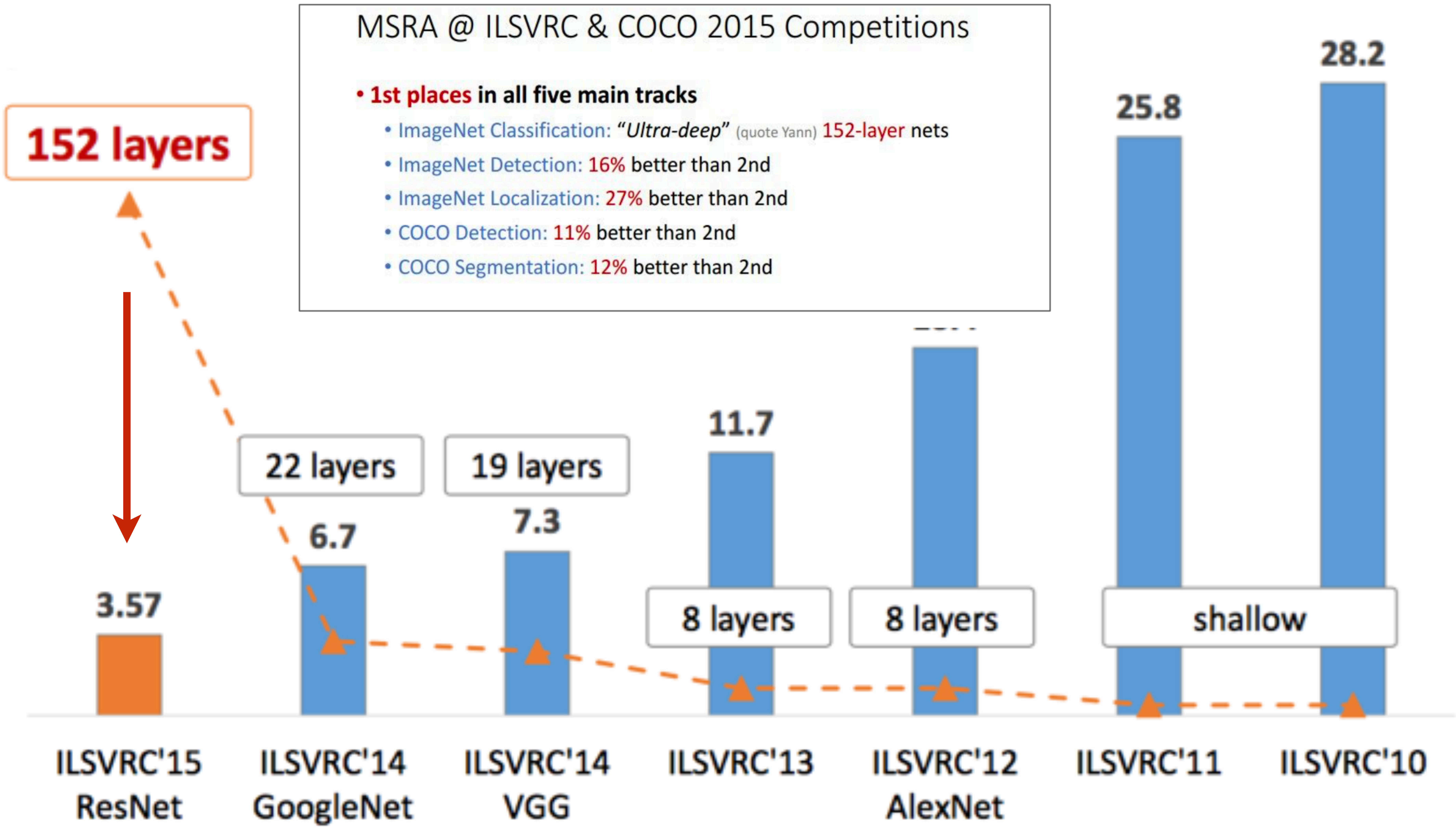


ILSVRC winner 2012



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

ILSVRC winner 2012



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

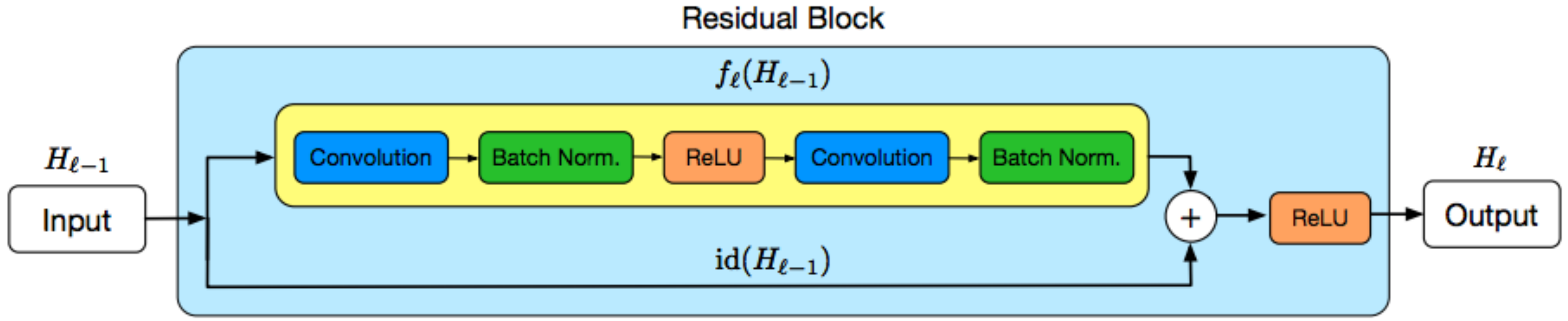
Regularization: Stochastic Depth

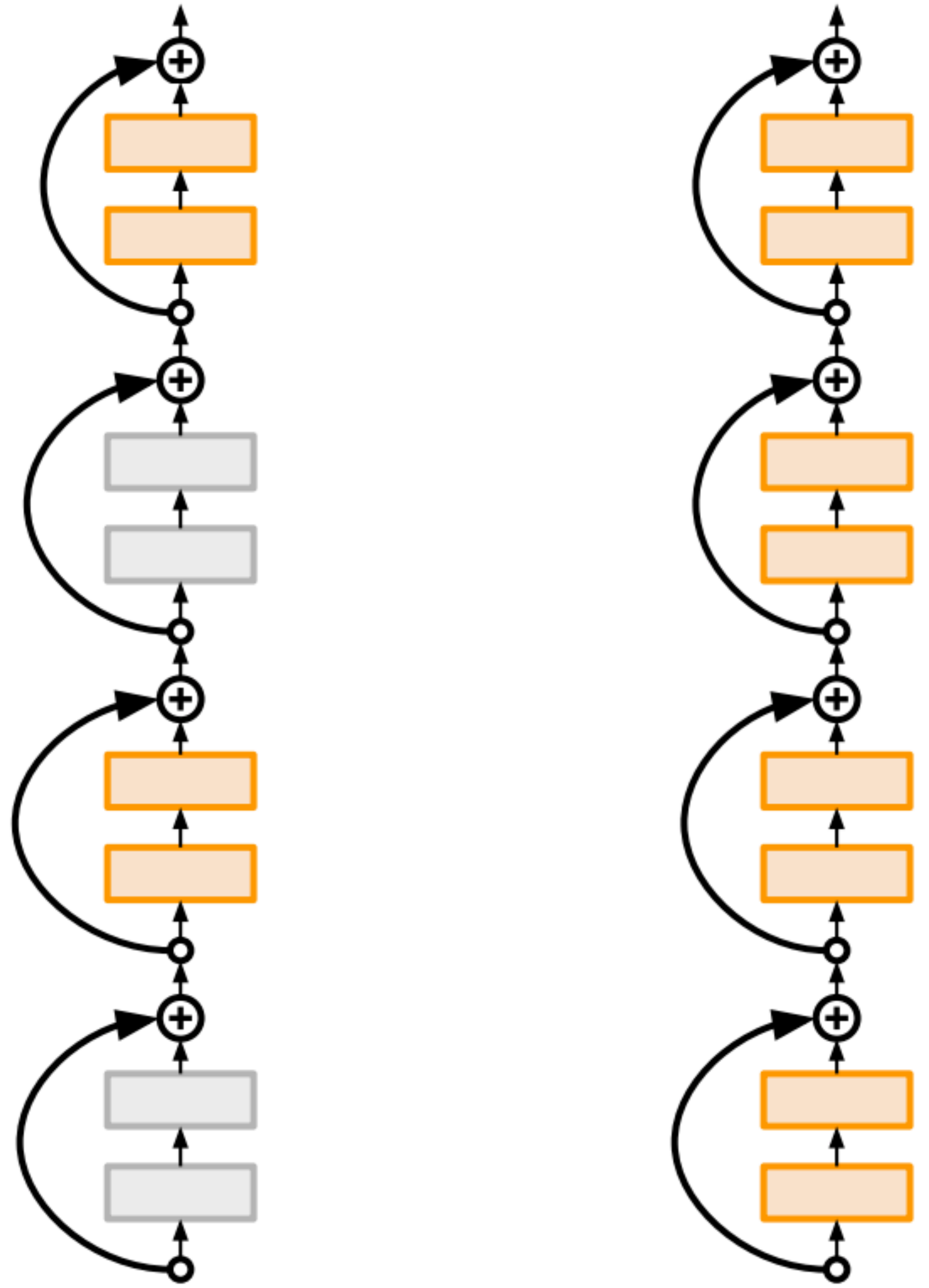
[Huang et al., ECCV 2016]

Effectively "dropout" but for layers

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

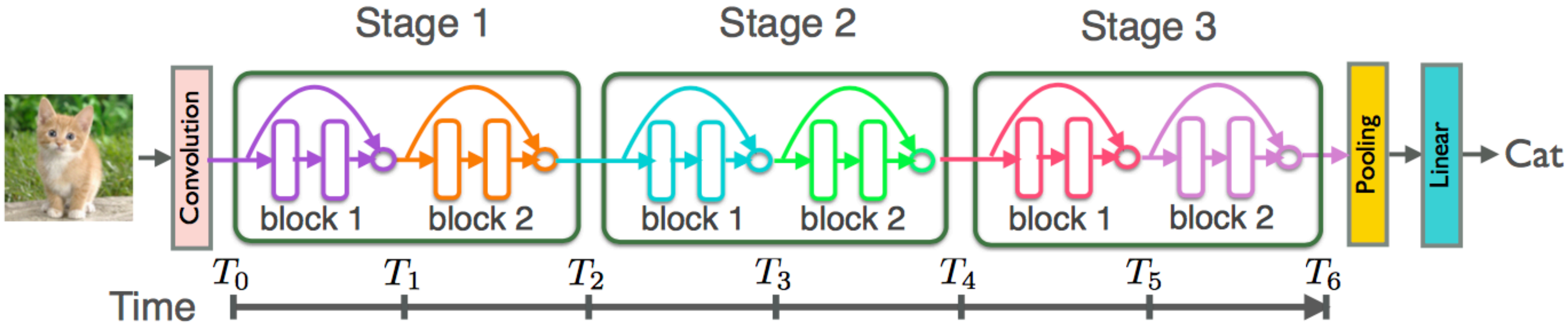
Effectively trains a collection of neural networks





ResNet: A little theory

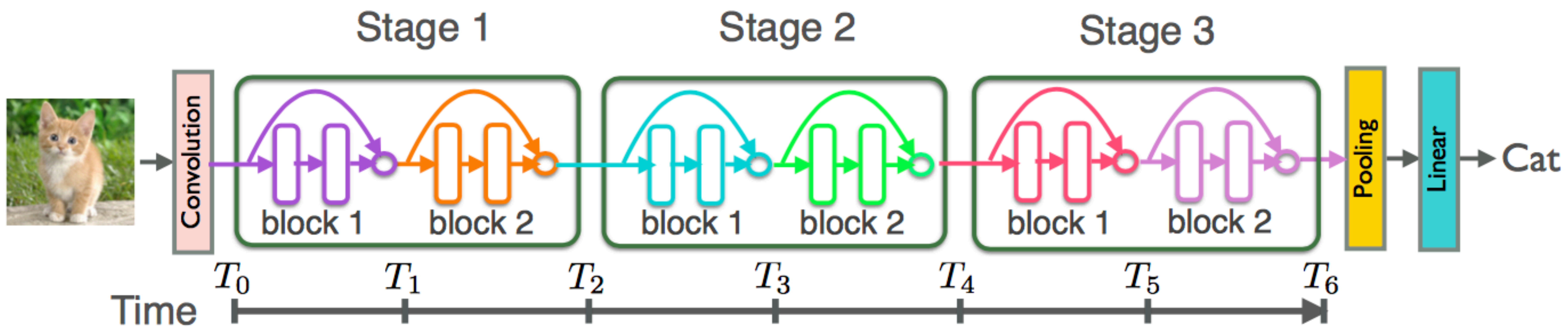
One can view a sequence of outputs from residual layers as a **Dynamical System**



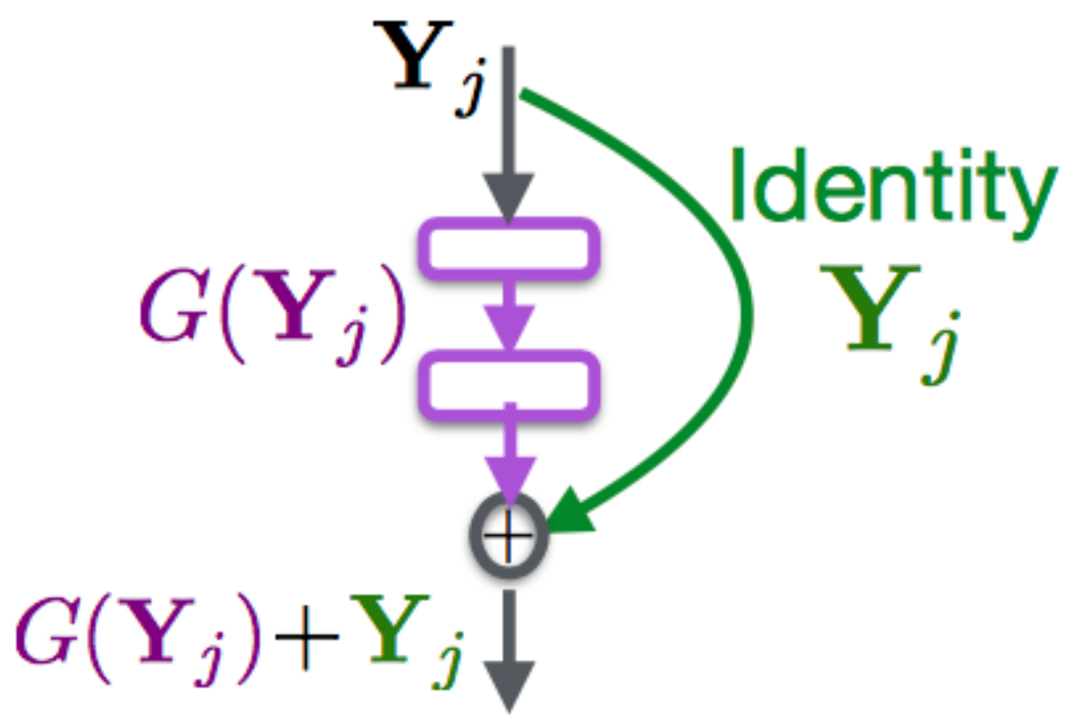
[Cheng et al., ICLR 2018]

ResNet: A little theory

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



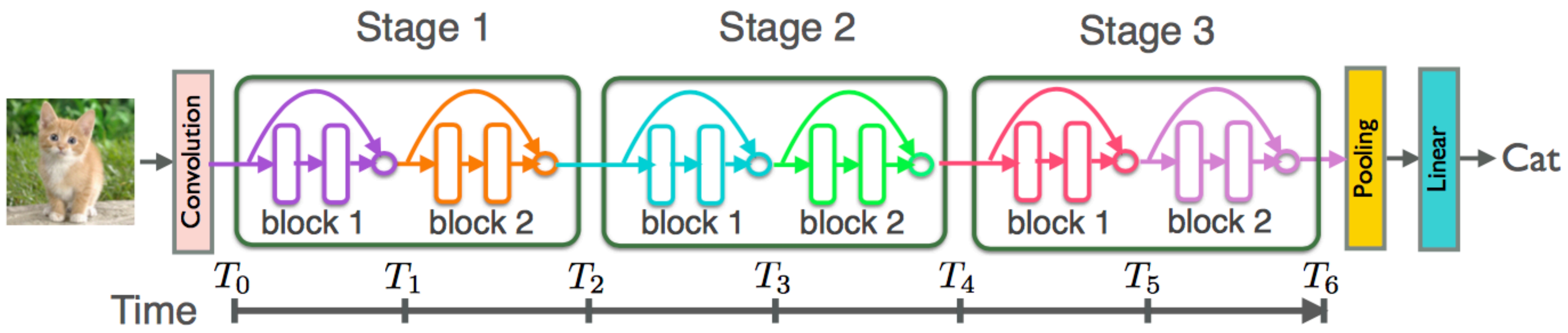
$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + G(\mathbf{Y}_j, \boldsymbol{\theta}_j)$$



[Cheng et al., ICLR 2018]

ResNet: A little theory

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

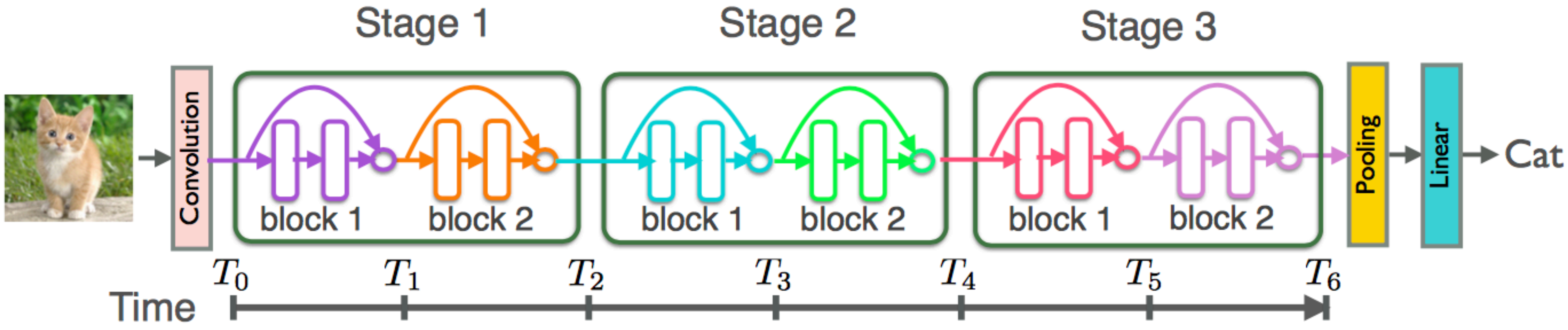


What happens if you take more layers and take smaller steps?

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

ResNet: A little theory

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



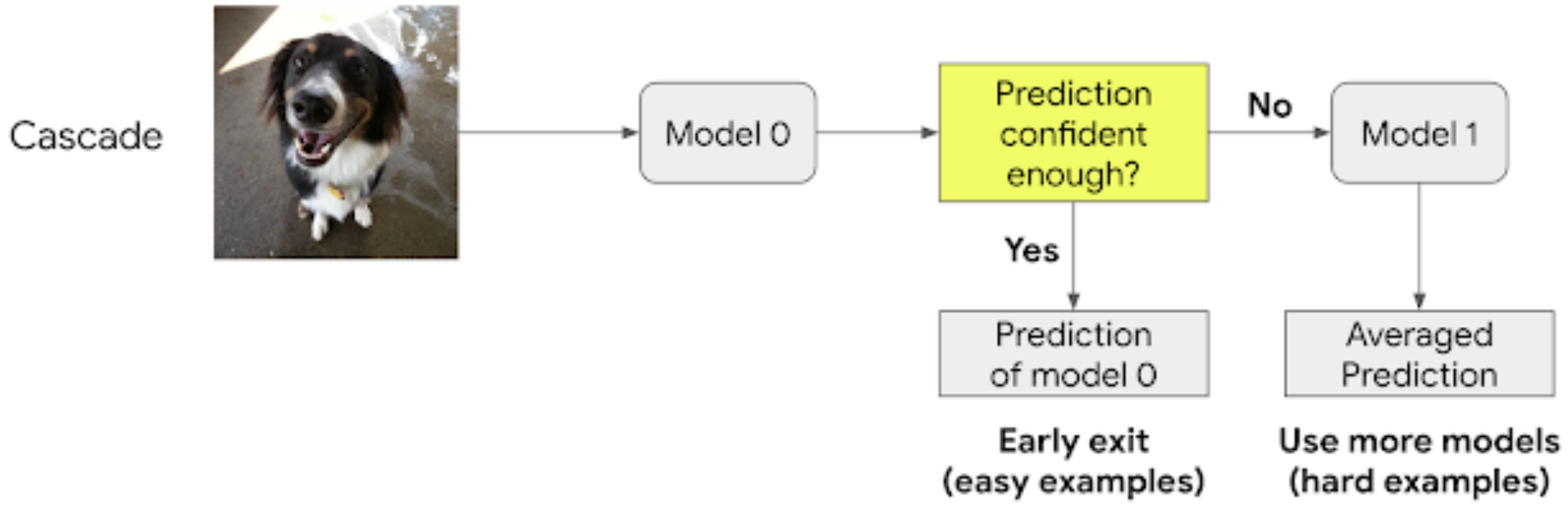
What happens if you take more layers and take smaller steps?

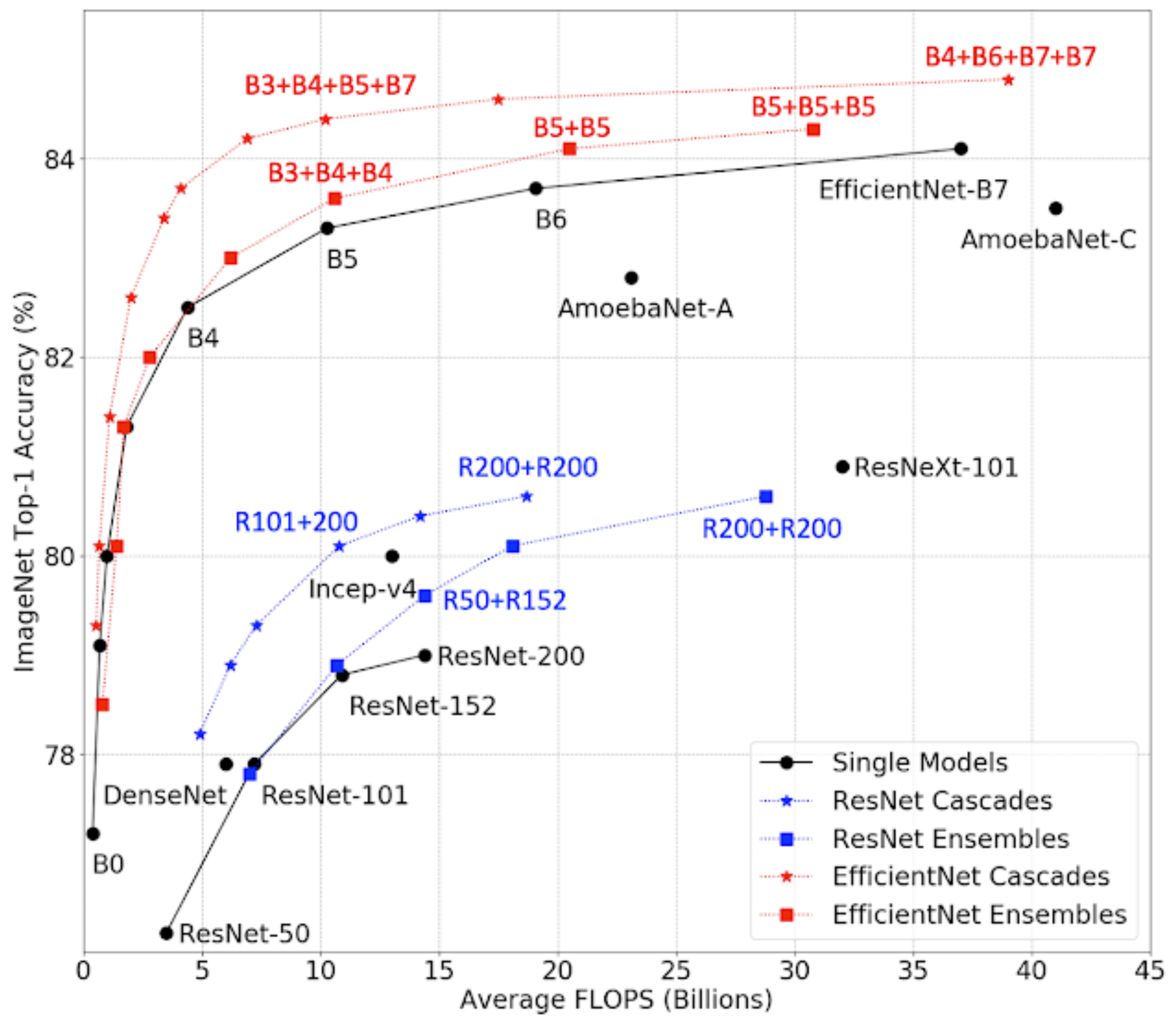
You can actually treat a neural network as an **ODE**: $$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$$

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

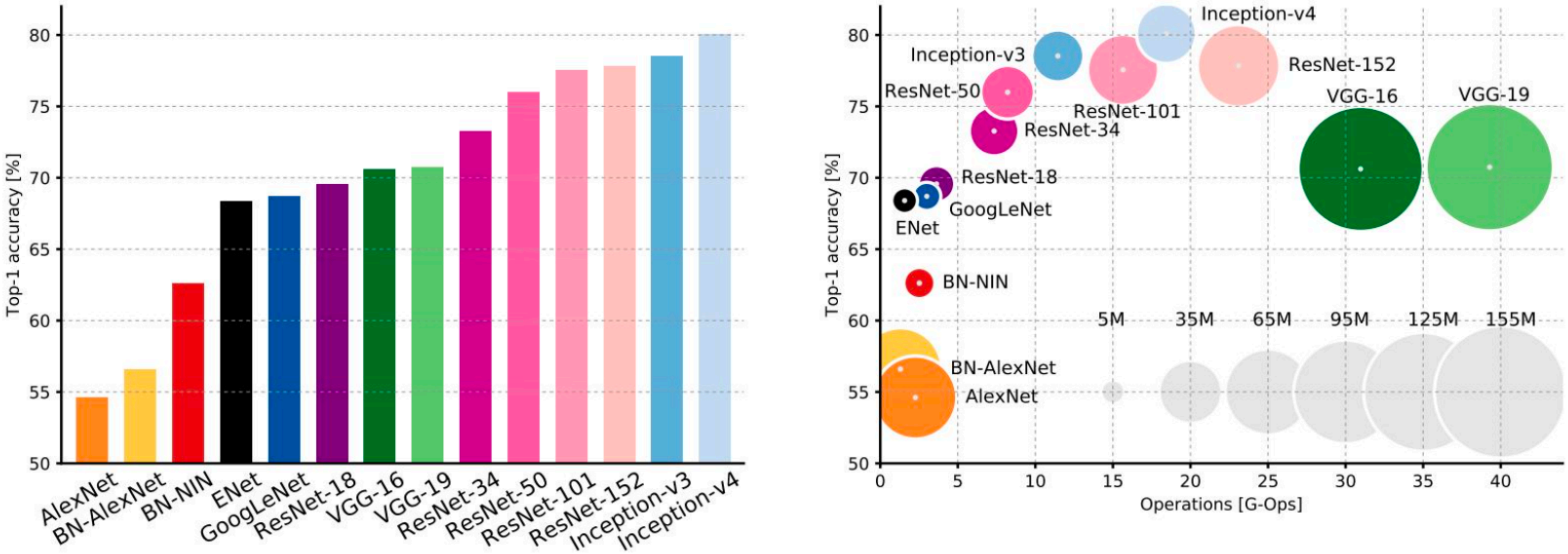
An Aside: Neural Network **Cascades**

[Wang et al., ICLR 2022]





Comparing **Complexity**



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

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

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

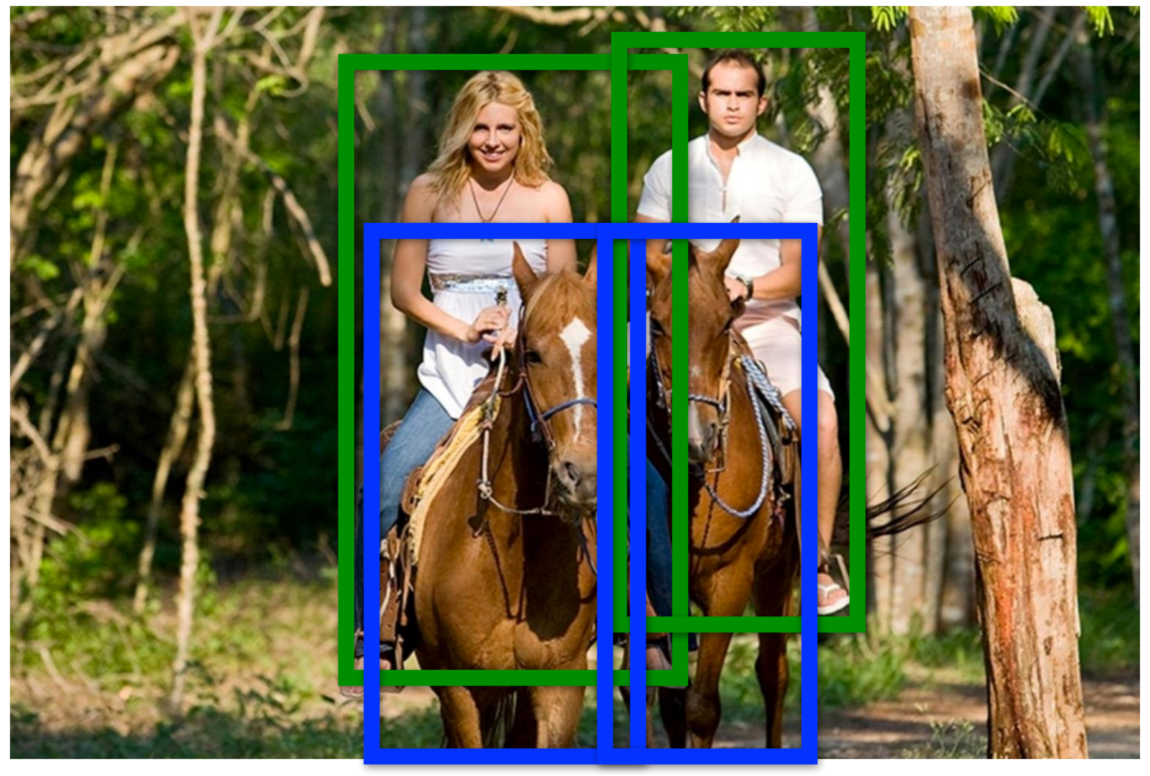
Categorization

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

IMAGENET

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

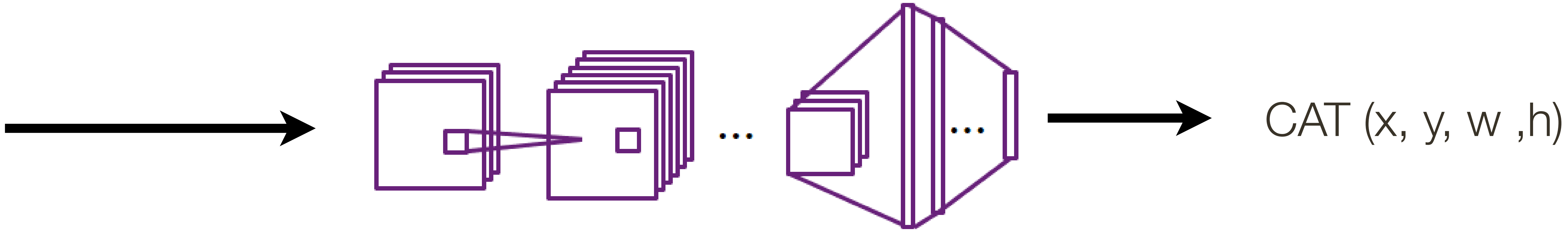
Detection



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

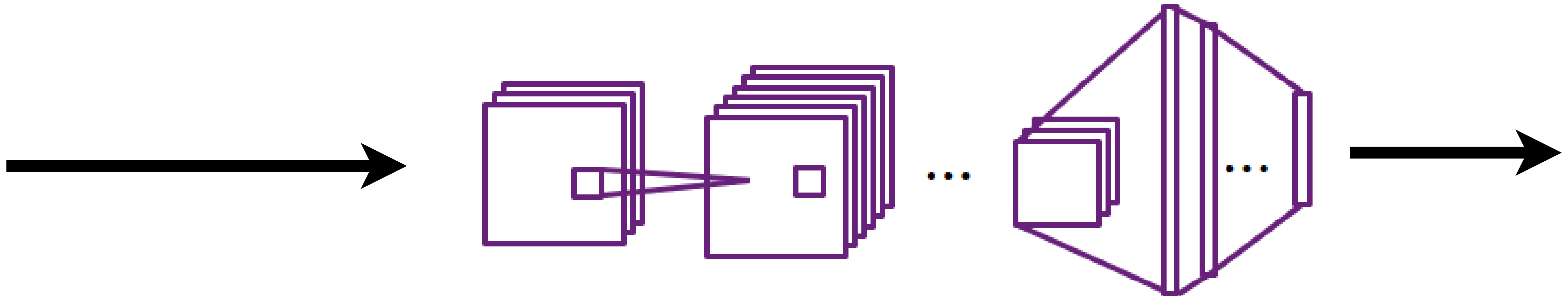
COCO
Common Objects in Context

Object **Detection** as Regression Problem

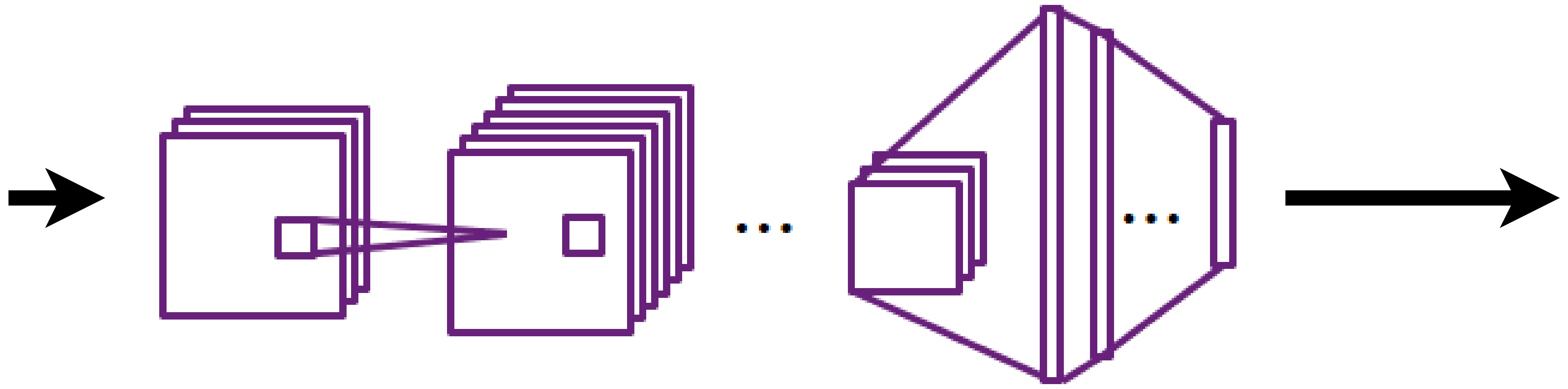


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Object **Detection** as Regression Problem



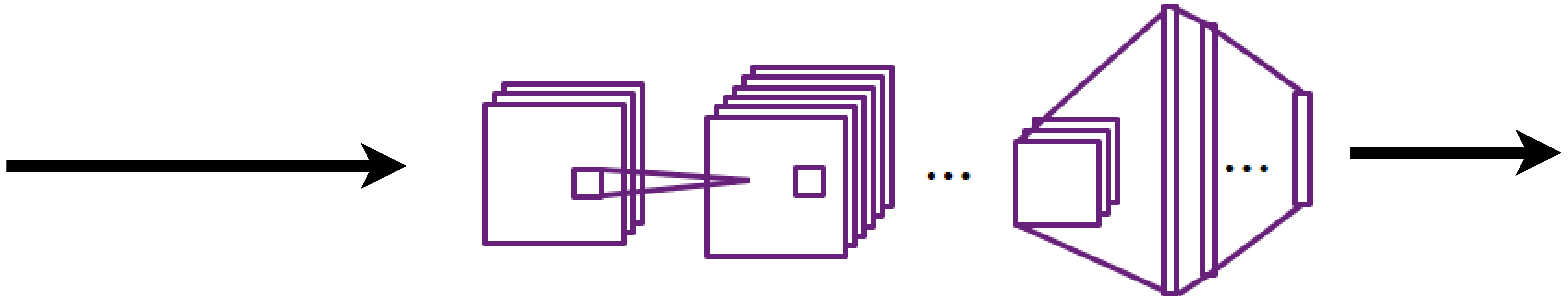
CAT (x, y, w ,h)



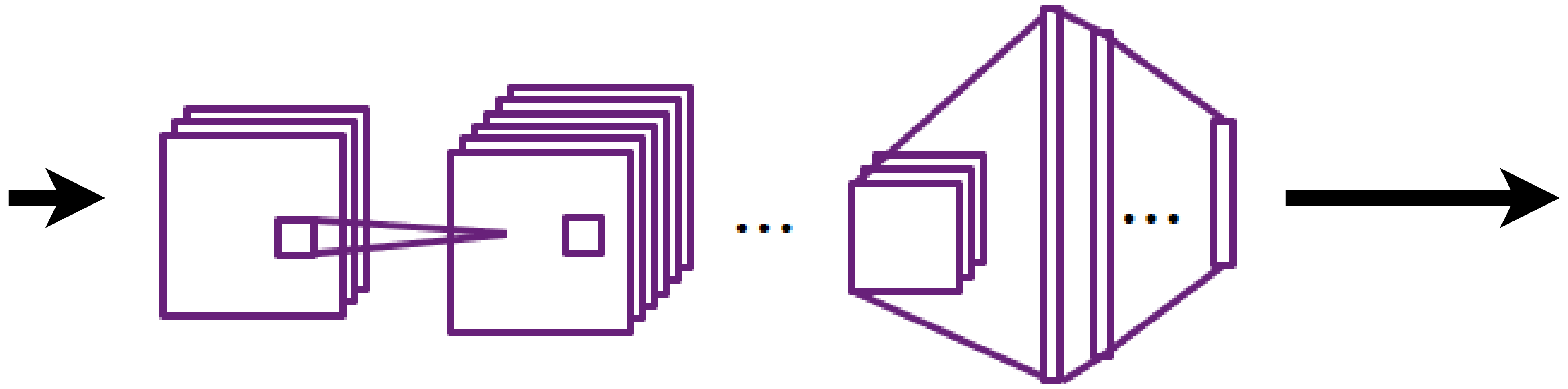
DUCK (x, y, w ,h)
DUCK (x, y, w ,h)
DUCK (x, y, w ,h)
DUCK (x, y, w ,h)
DUCK (x, y, w ,h)
DUCK (x, y, w ,h)
DUCK (x, y, w ,h)
…

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

Object **Detection** as Regression Problem

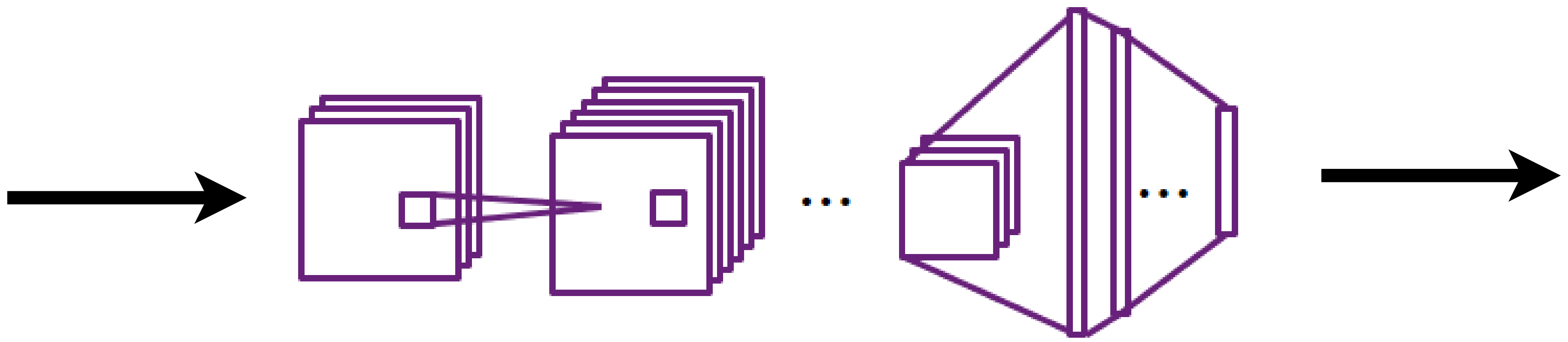


Problem: each image needs a different number of outputs



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

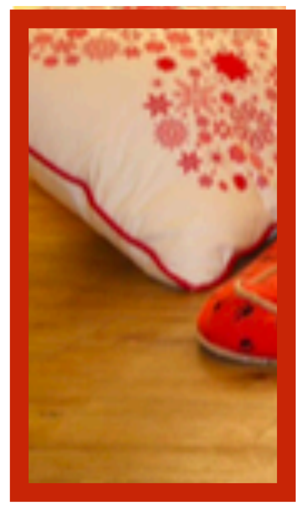
Object **Detection** as Classification Problem

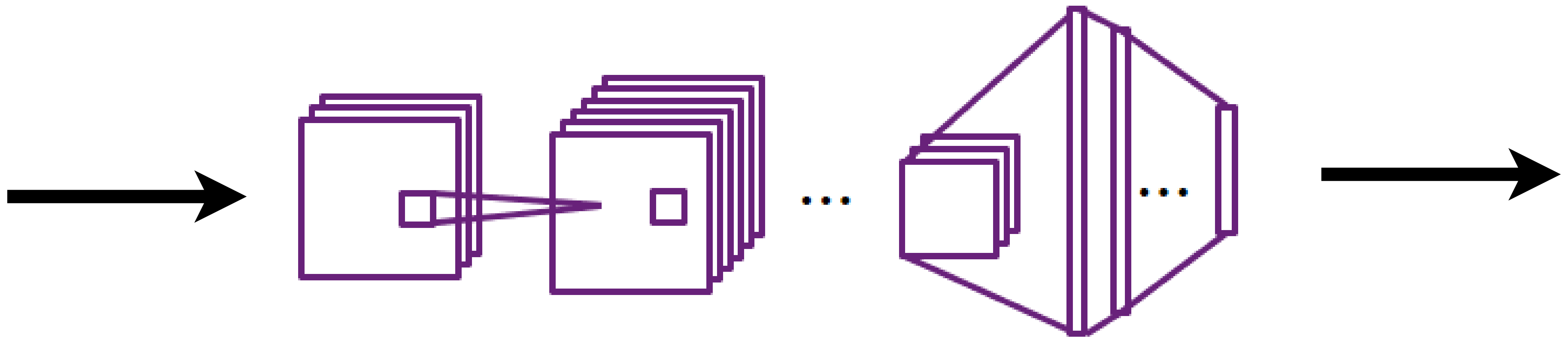


Category	**Prediction**
Dog	No
Cat	No
Couch	No
Flowers	No
Background	**Yes**
…	…

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

Object **Detection** as Classification Problem



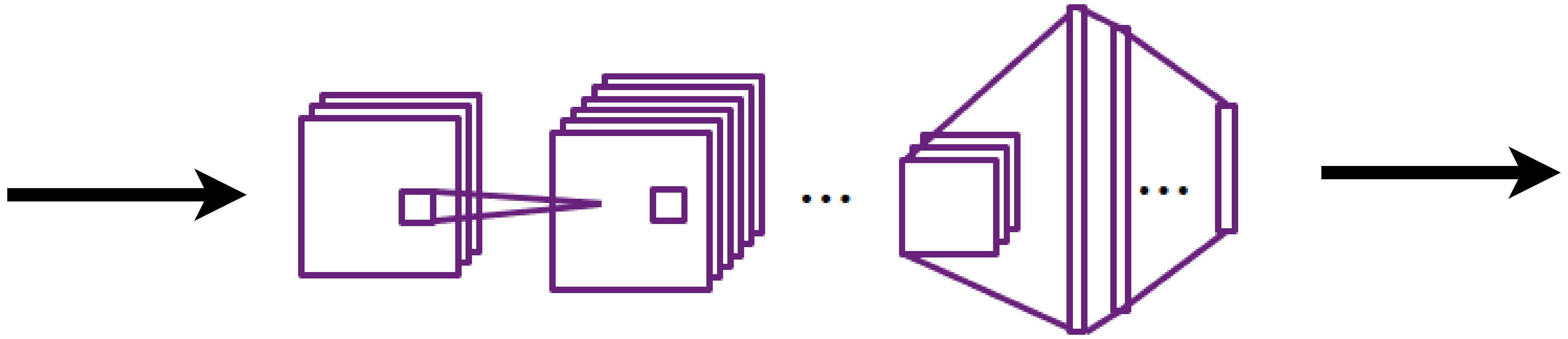


Category	Prediction
Dog	No
Cat	No
Couch	No
Flowers	No
Background	**Yes**
…	…

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

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

Object **Detection** as Classification Problem

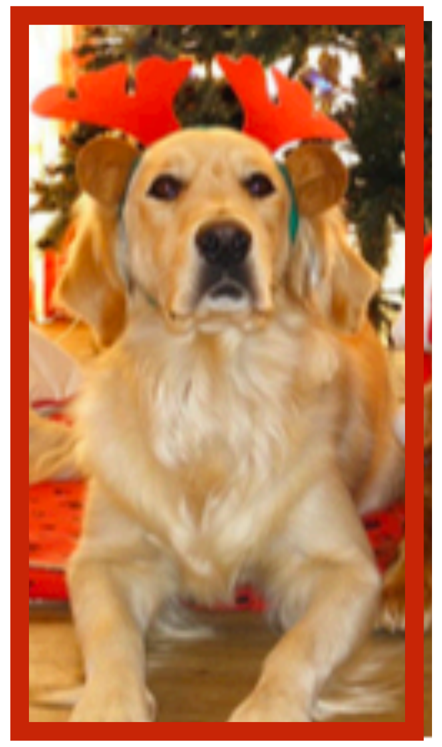


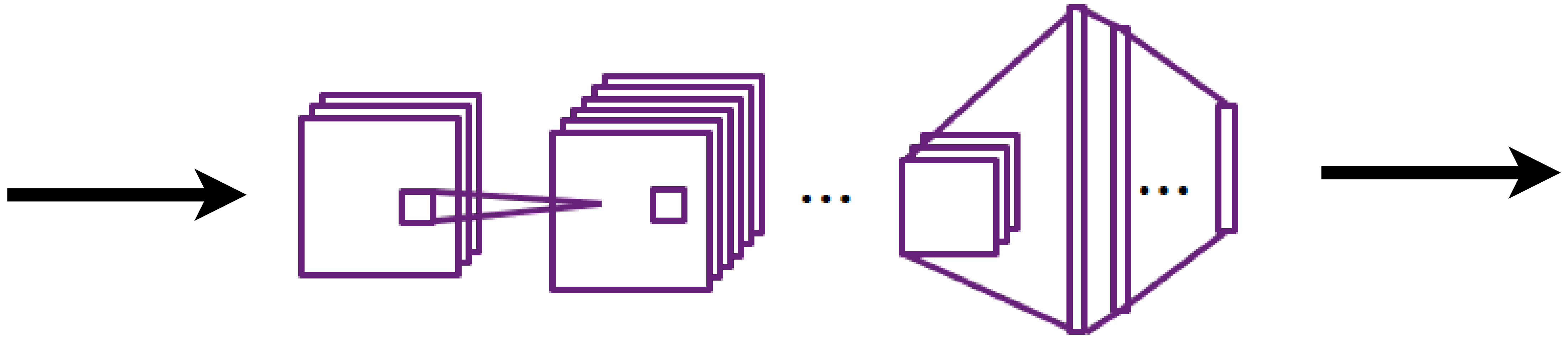
Category	**Prediction**
Dog	**Yes**
Cat	No
Couch	No
Flowers	No
Background	No
…	…

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

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

Object **Detection** as Classification Problem

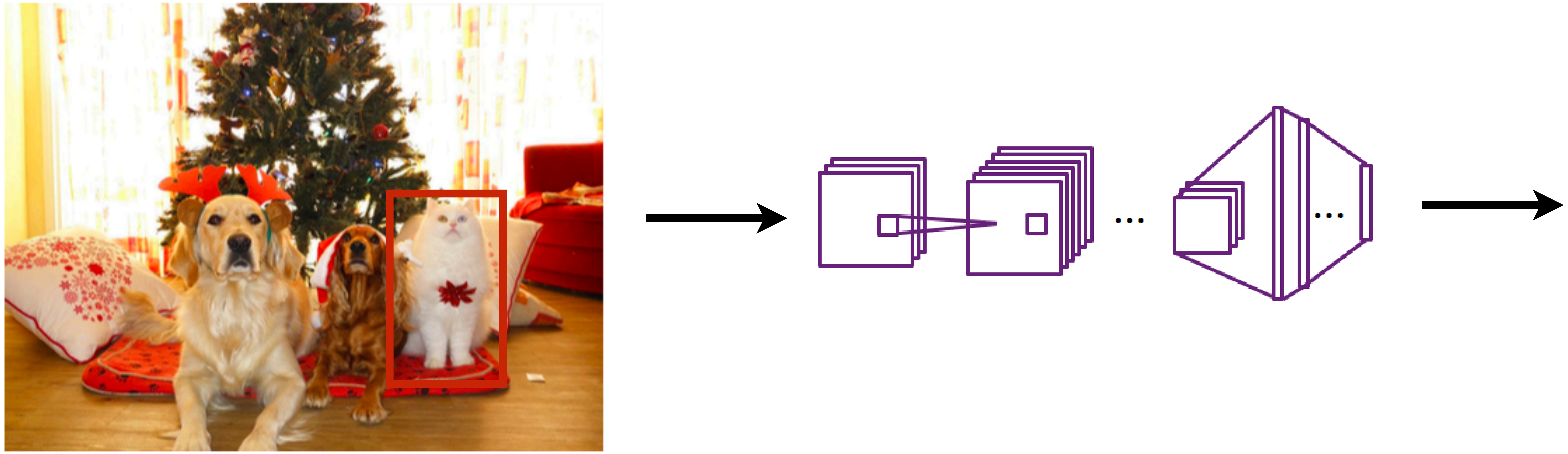




Category	**Prediction**
Dog	**Yes**
Cat	No
Couch	No
Flowers	No
Background	No
…	…

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

Object **Detection** as Classification Problem

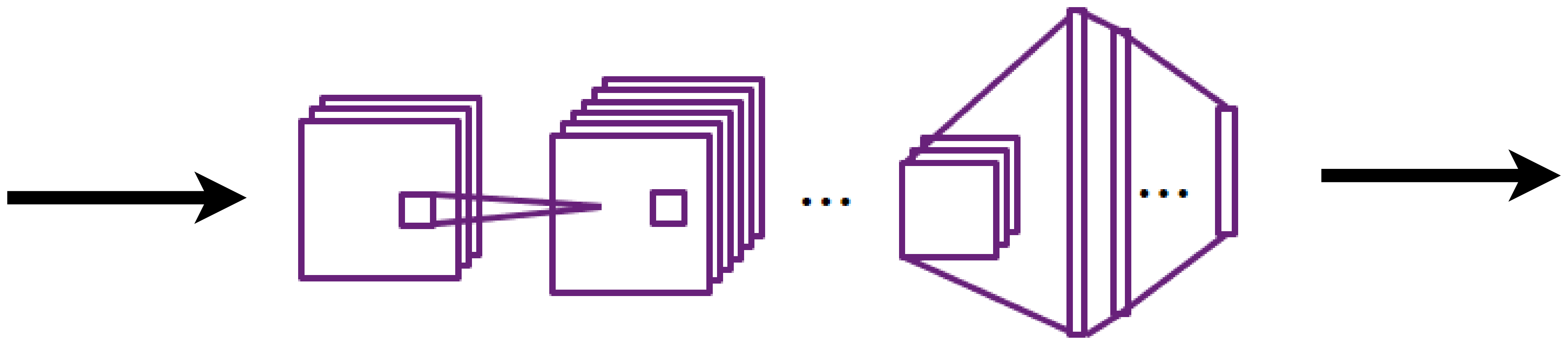


Category	**Prediction**
Dog	No
Cat	**Yes**
Couch	No
Flowers	No
Background	No
…	…

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

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

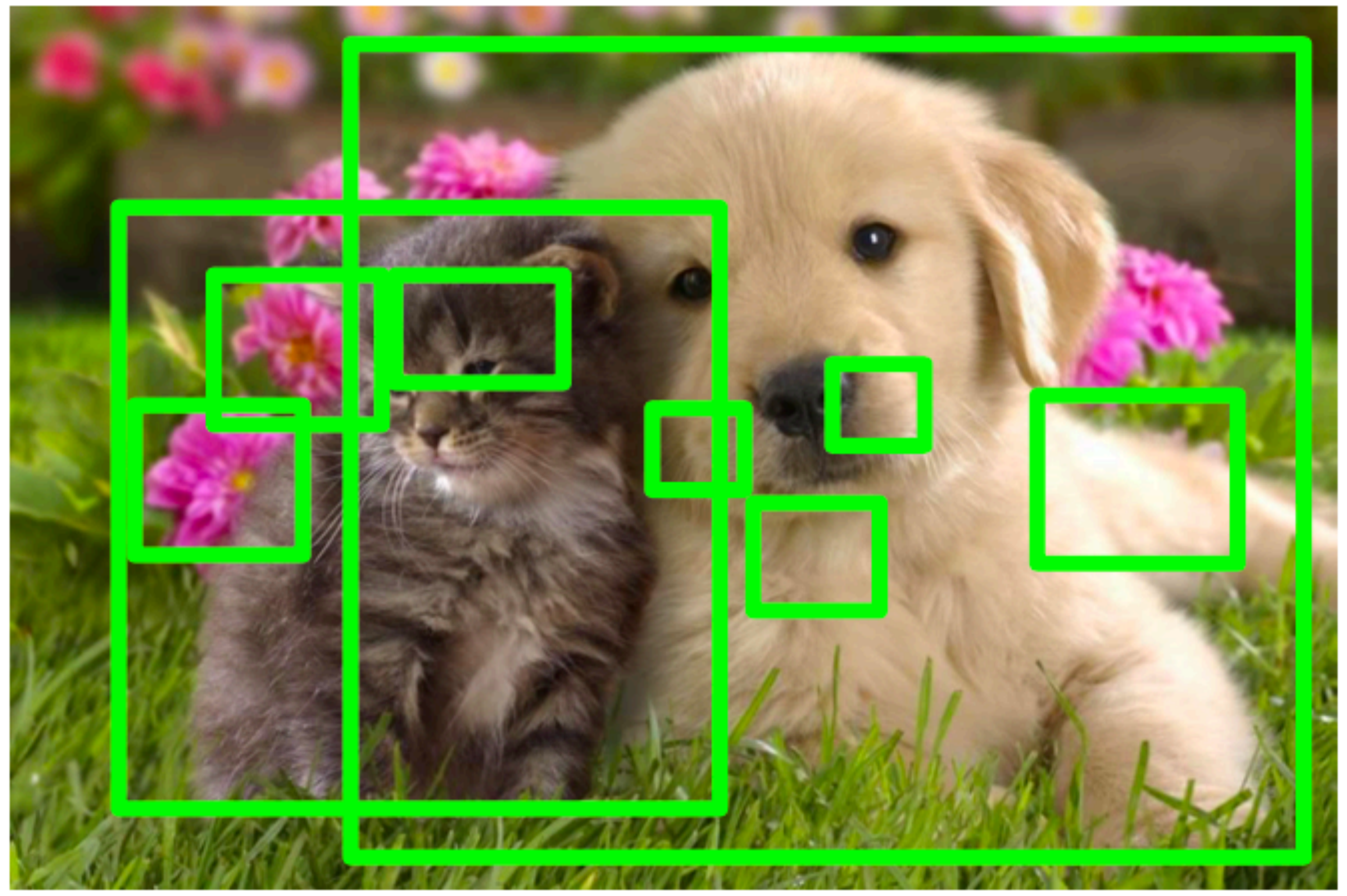
Region Proposals (older idea in vision)

[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

R-CNN

[Girshick et al, CVPR 2014]

Input **Image**

* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]

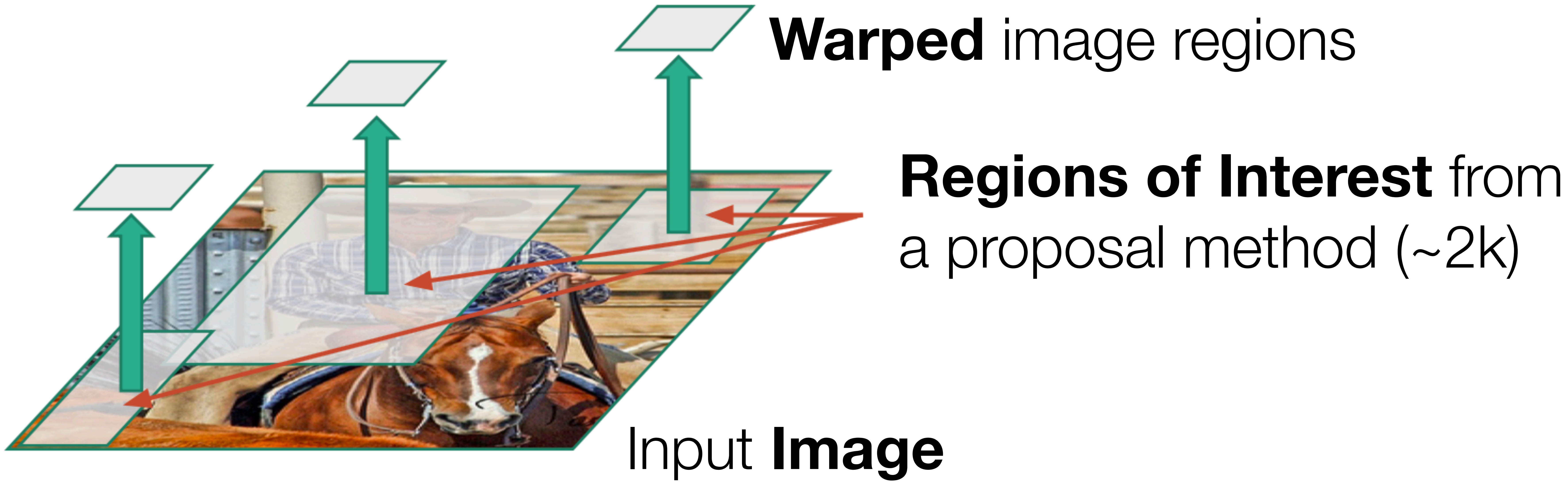
Regions of Interest from a proposal method (~2k)

Input **Image**

* image from Ross Girshick

R-CNN

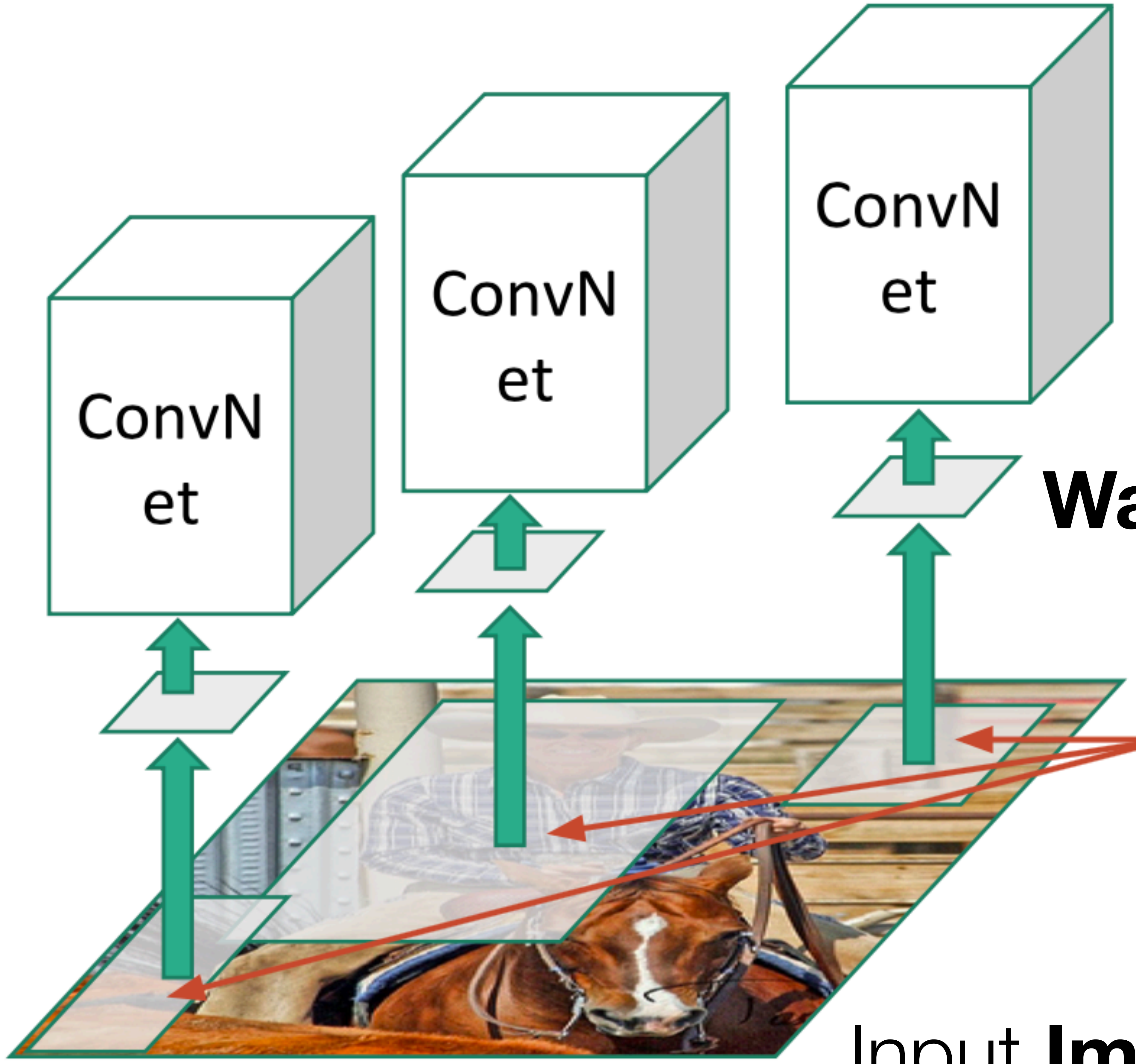
[Girshick et al, CVPR 2014]



* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]



Forward each region through a **CNN**

Warped image regions

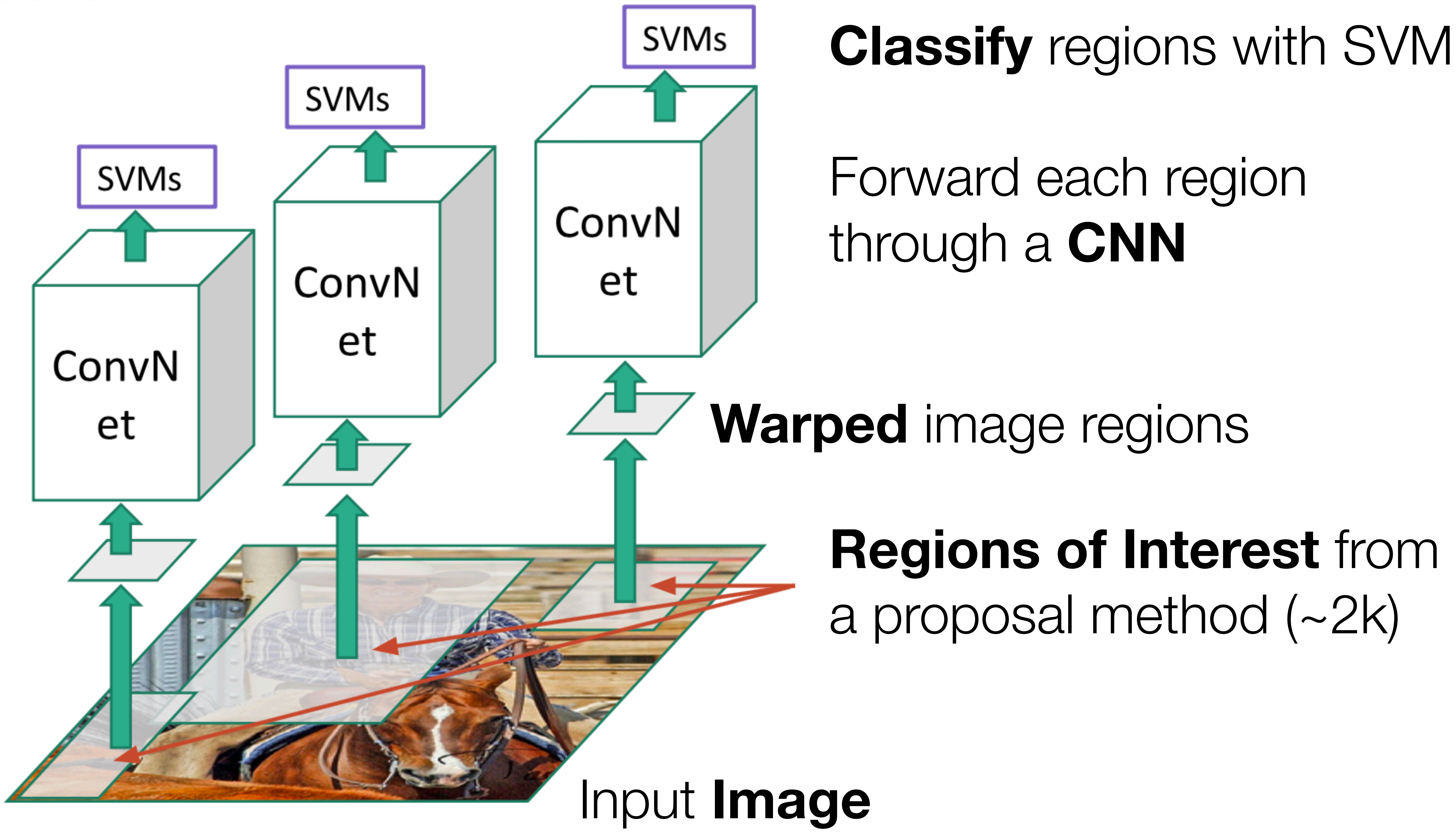
Regions of Interest from a proposal method (~2k)

Input **Image**

* image from Ross Girshick

R-CNN

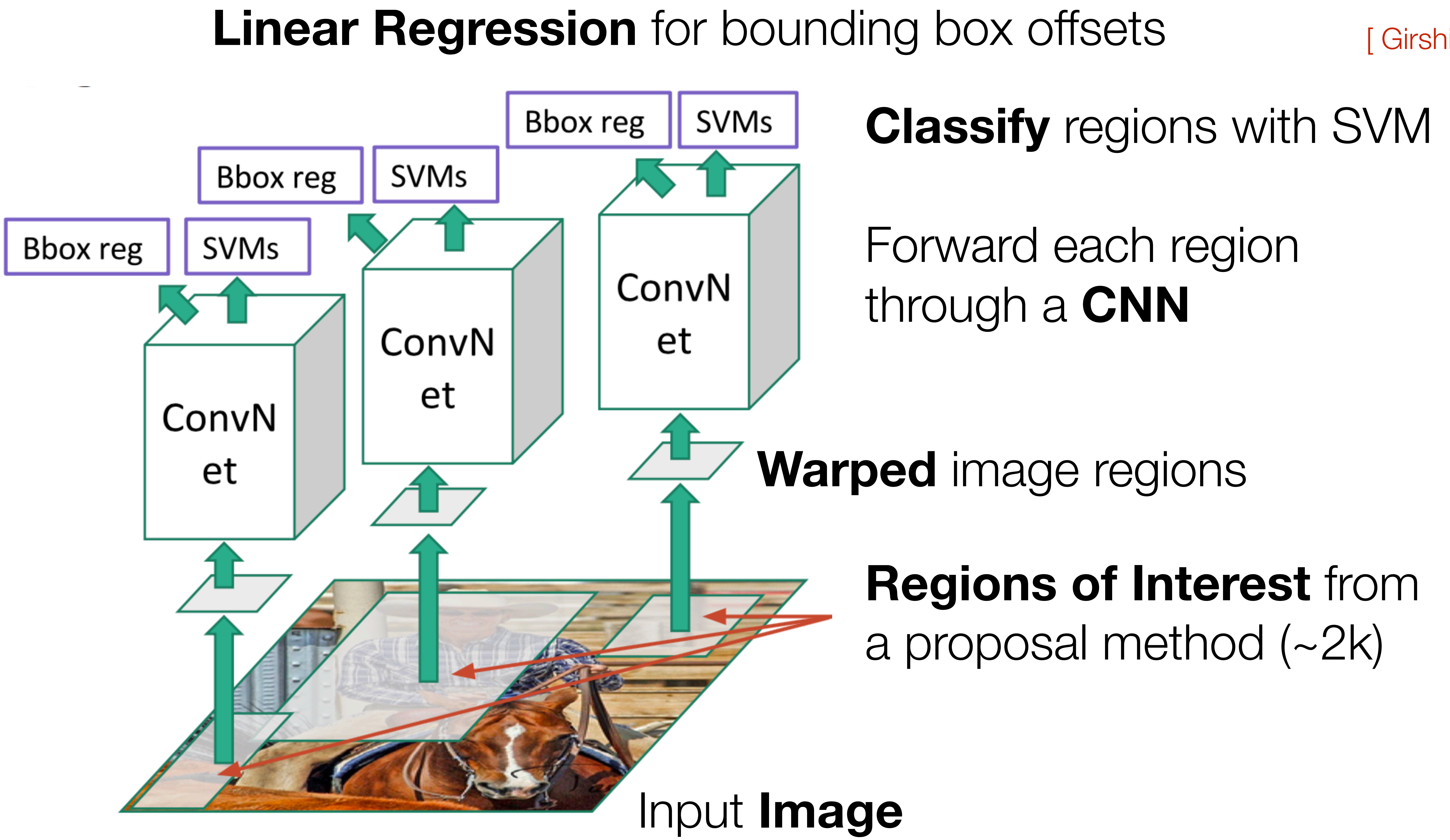
[Girshick et al, CVPR 2014]



* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]



* 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)

Fast **R-CNN**

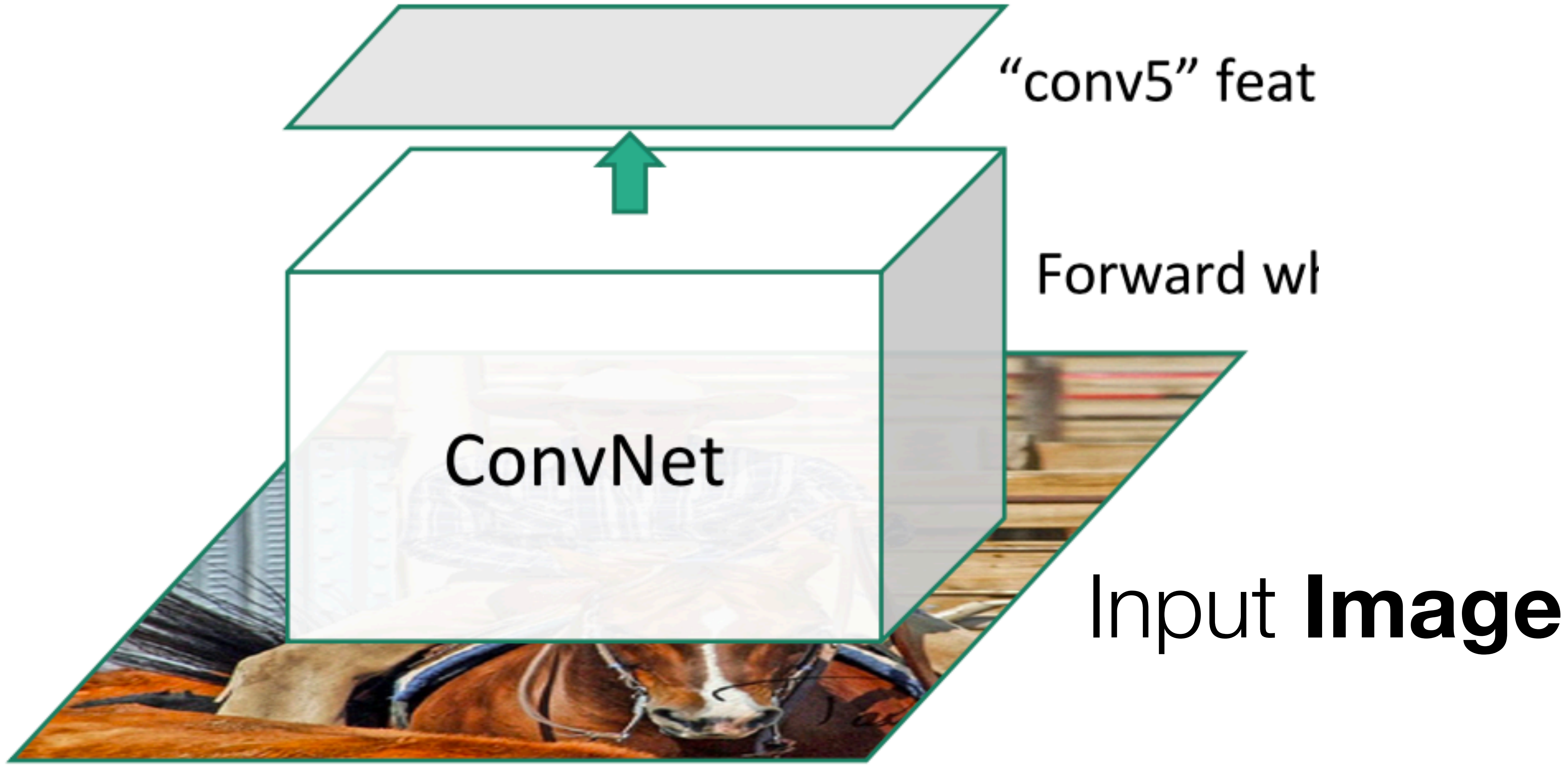
[Girshick et al, ICCV 2015]

Input **Image**

* image from Ross Girshick

Fast **R-CNN**

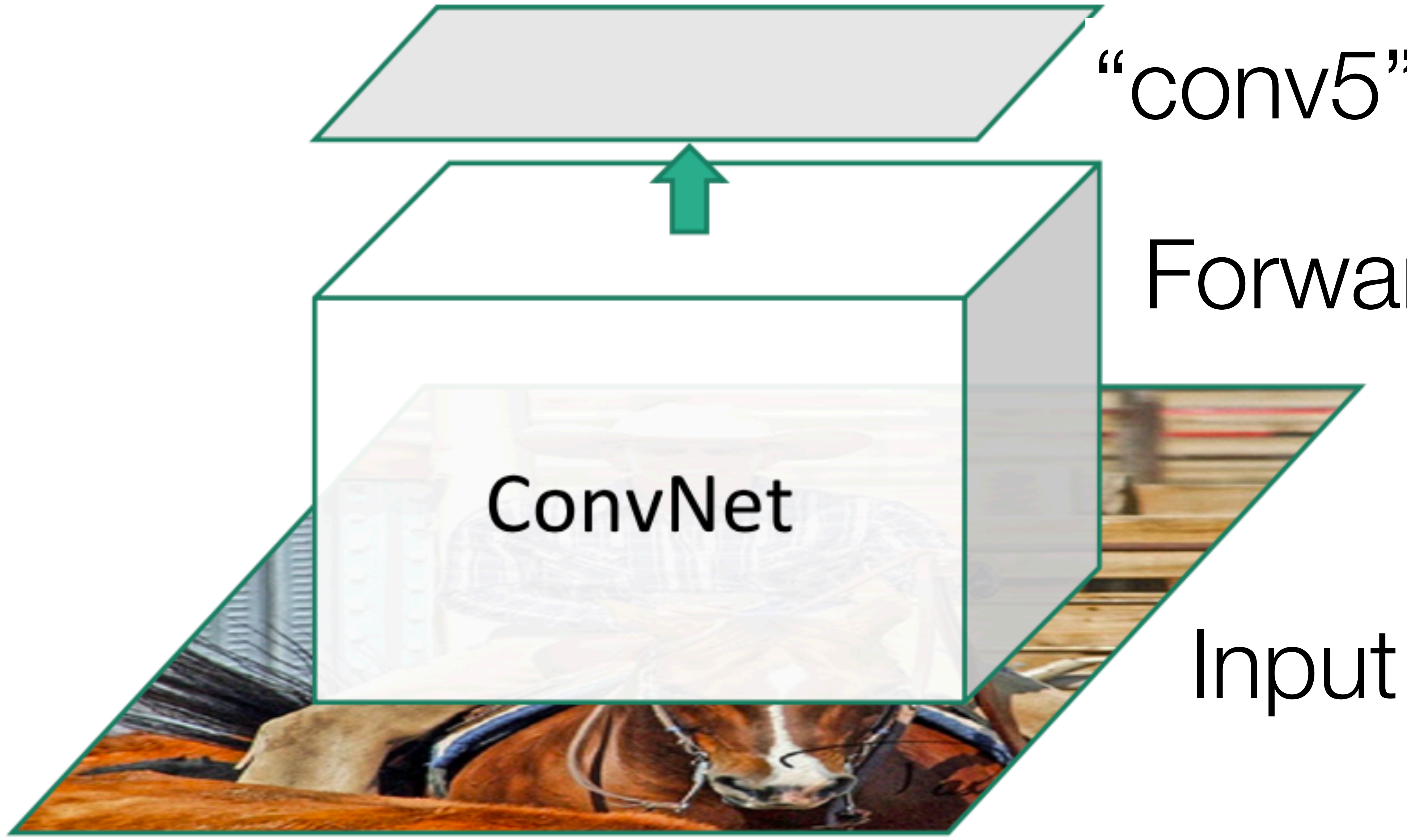
[Girshick et al, ICCV 2015]



* image from Ross Girshick

Fast **R-CNN**

[Girshick et al, ICCV 2015]



"conv5" feature map

Forward prop the **whole image** through CNN

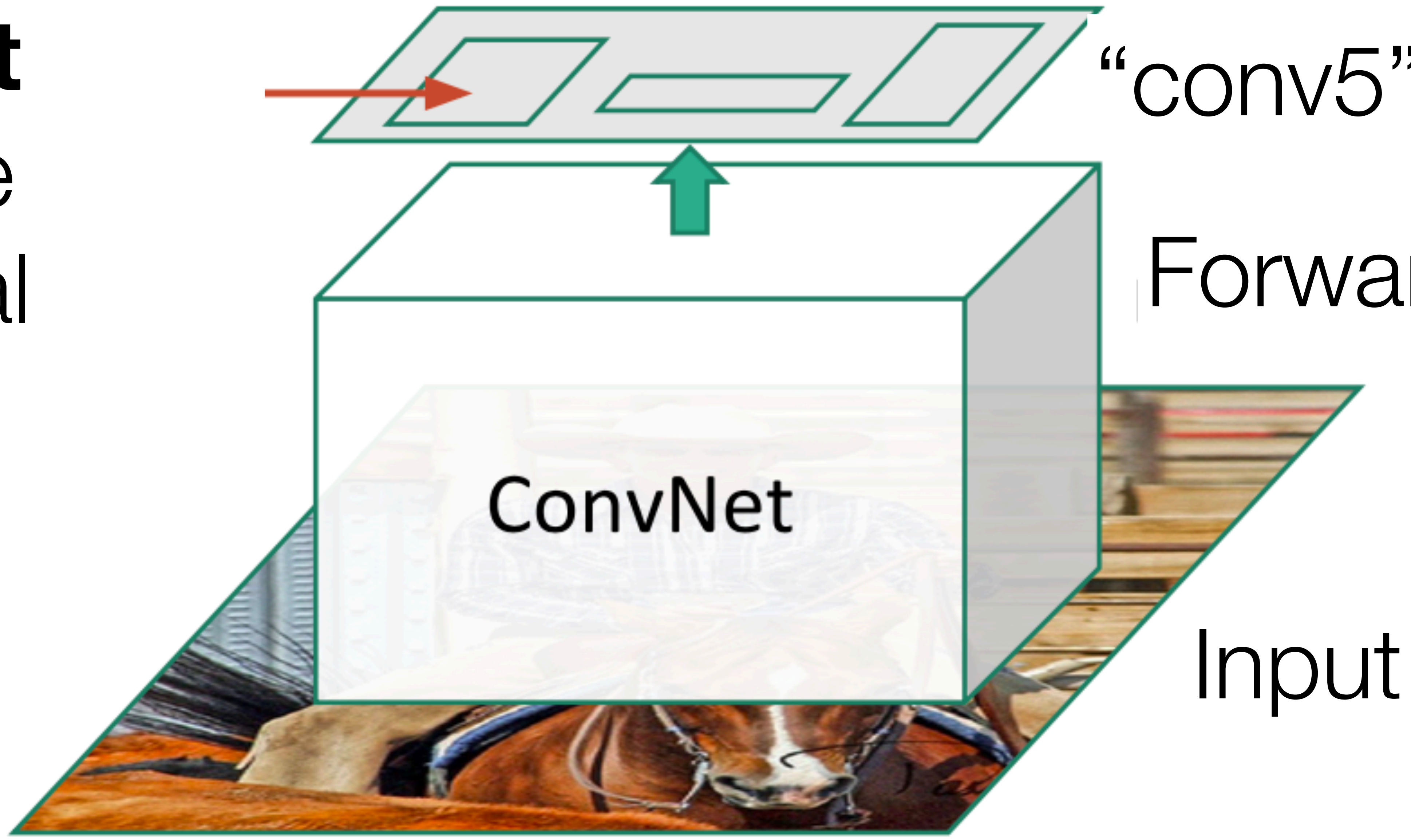
Input **Image**

* image from Ross Girshick

Fast **R-CNN**

[Girshick et al, ICCV 2015]

Regions of Interest from the proposal method



"conv5" feature map

Forward prop the **whole image** through CNN

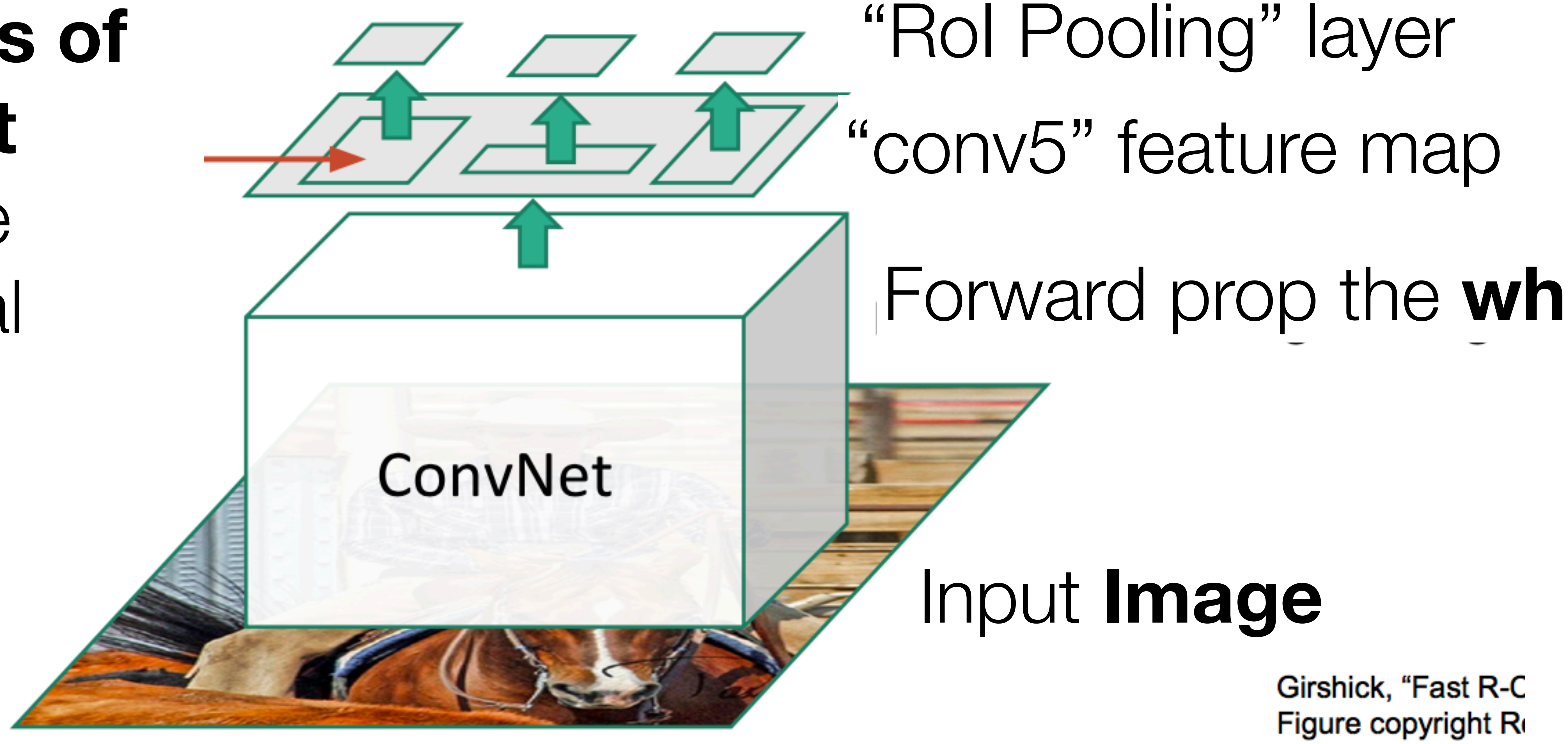
Input **Image**

* image from Ross Girshick

Fast **R-CNN**

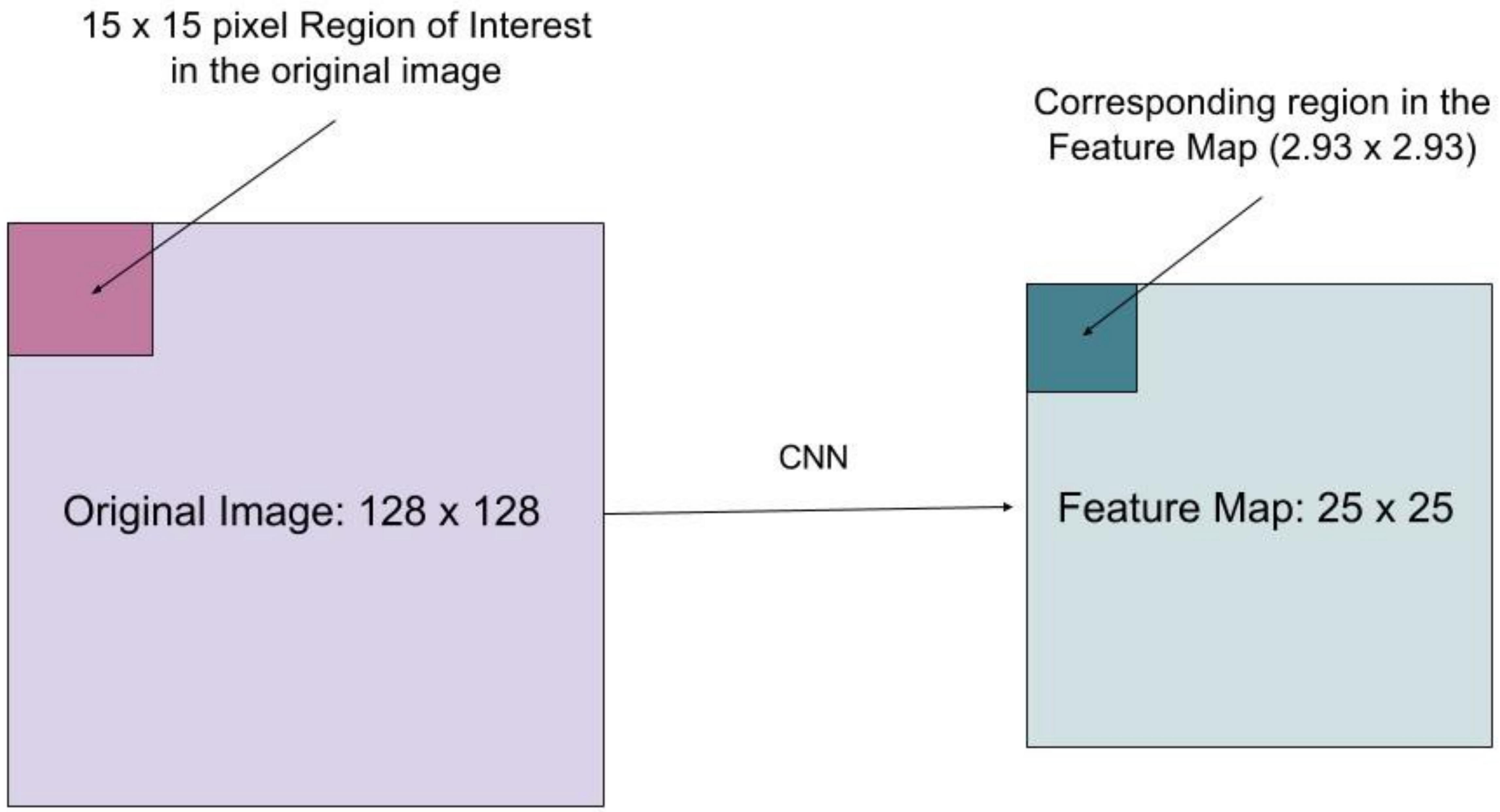
[Girshick et al, ICCV 2015]

Regions of Interest from the proposal method

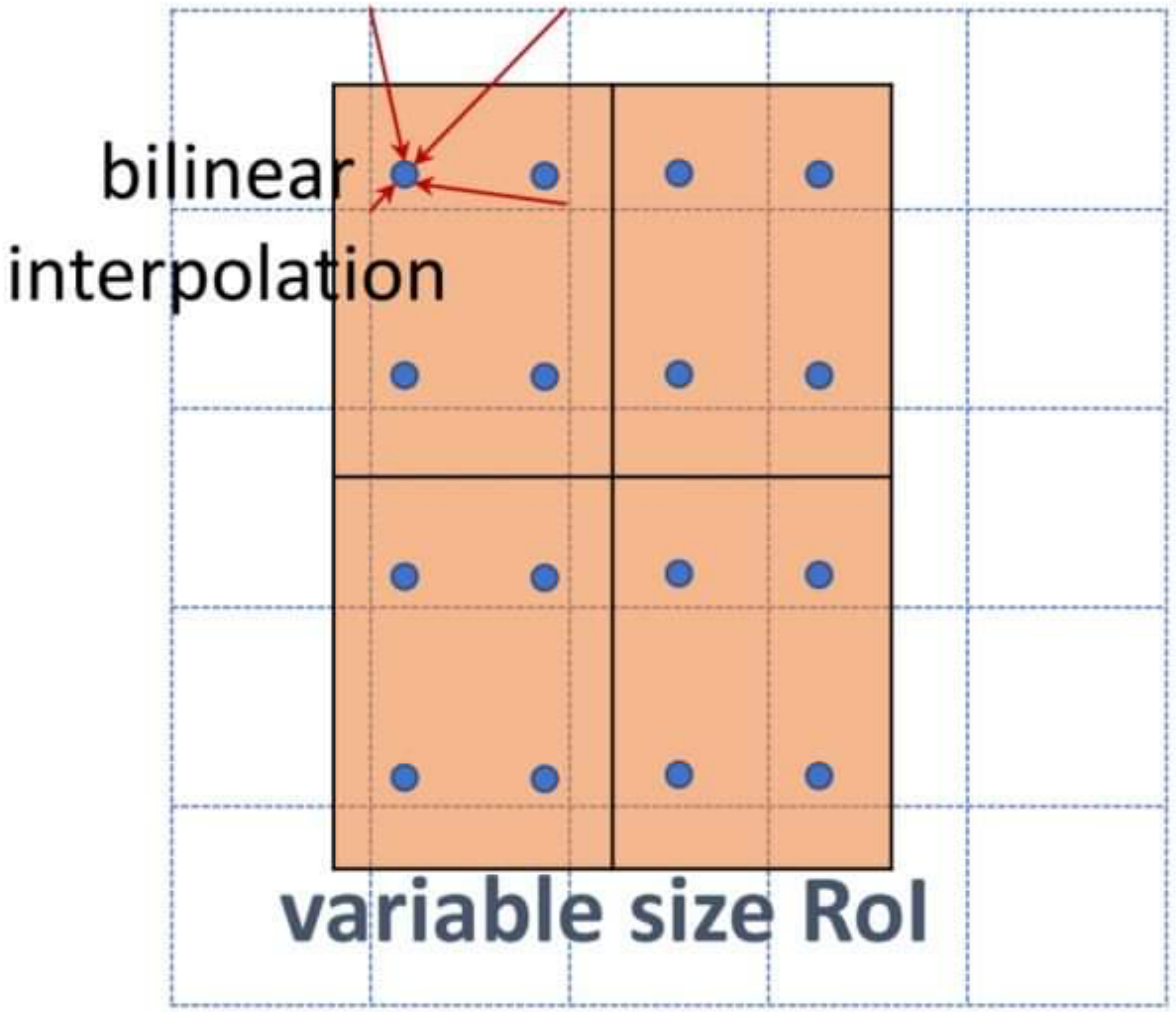


* image from Ross Girshick

RoI Align



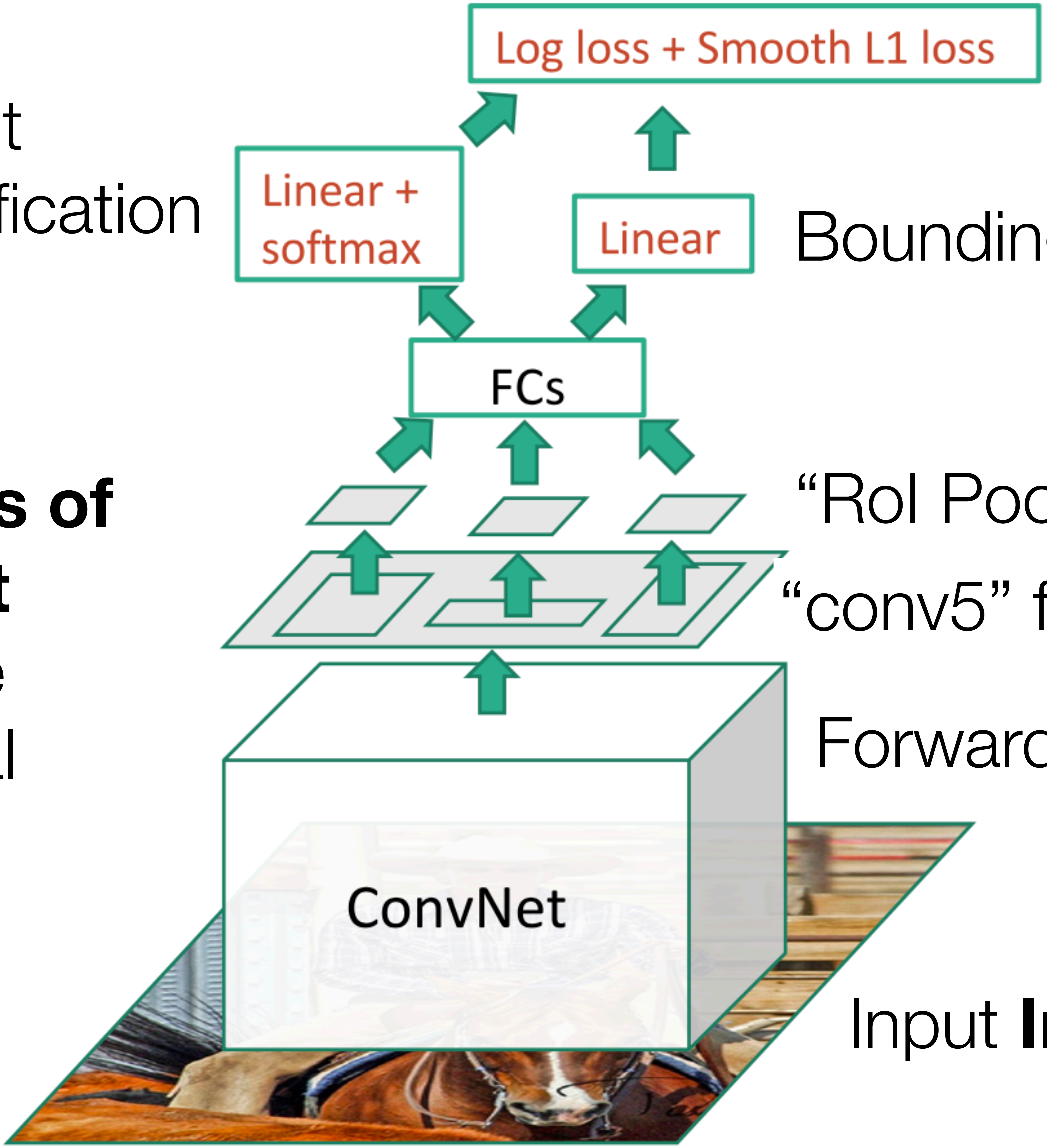
15 x 15 pixel Region of Interest in the original image
Original Image: 128 x 128
CNN
Corresponding region in the Feature Map (2.93 x 2.93)
Feature Map: 25 x 25



bilinear interpolation
variable size RoI

Fast **R-CNN**

[Girshick et al, ICCV 2015]



Multi-task loss

Object classification

Bounding box regression

"RoI Pooling" layer

Regions of Interest from the proposal method

"conv5" feature map

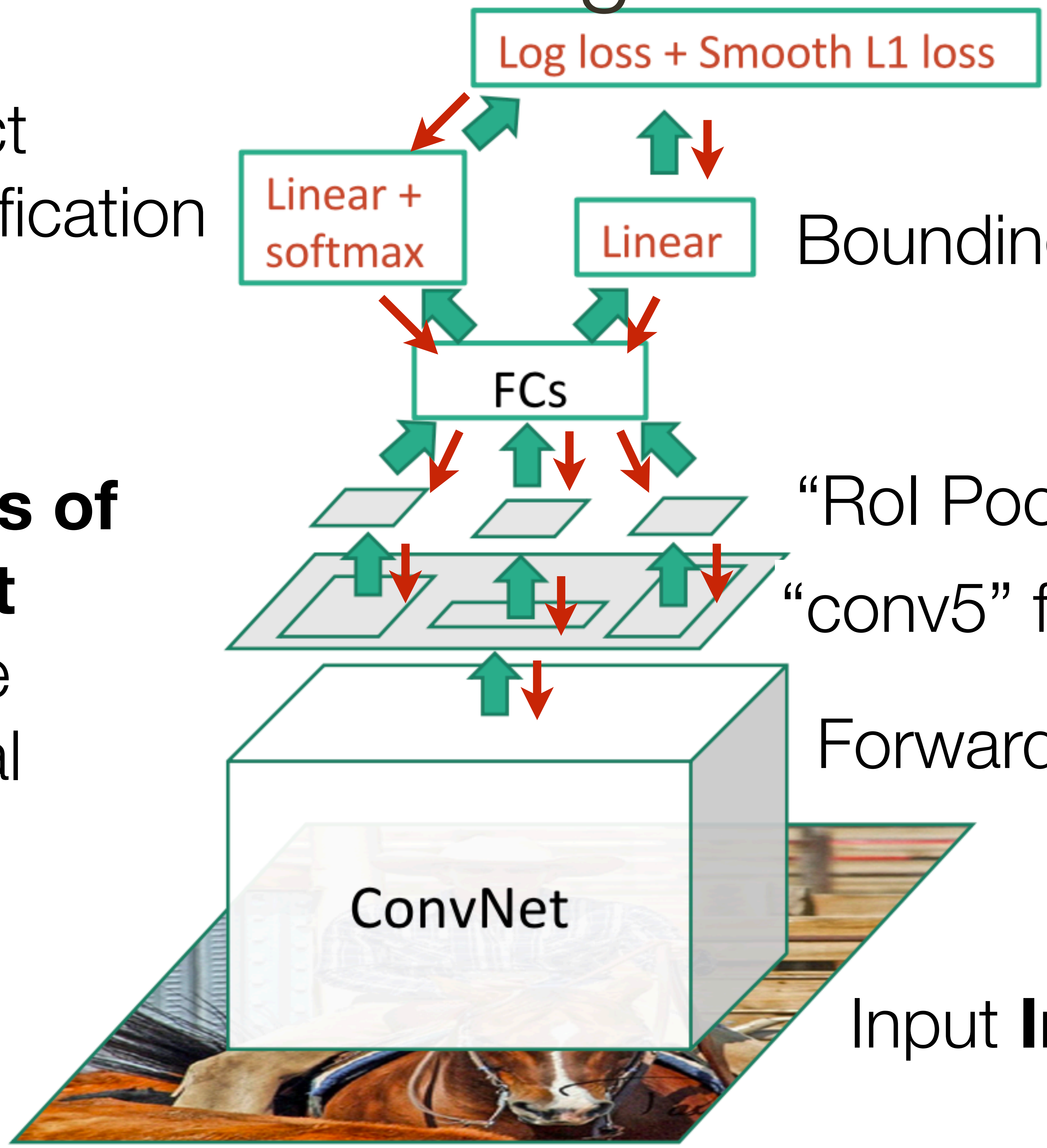
Forward prop the **whole image** through CNN

Input **Image**

* image from Ross Girshick

Fast **R-CNN:** Training

[Girshick et al, ICCV 2015]



Multi-task loss

Object classification

Bounding box regression

“RoI Pooling” layer

“conv5” feature map

Regions of Interest from the proposal method

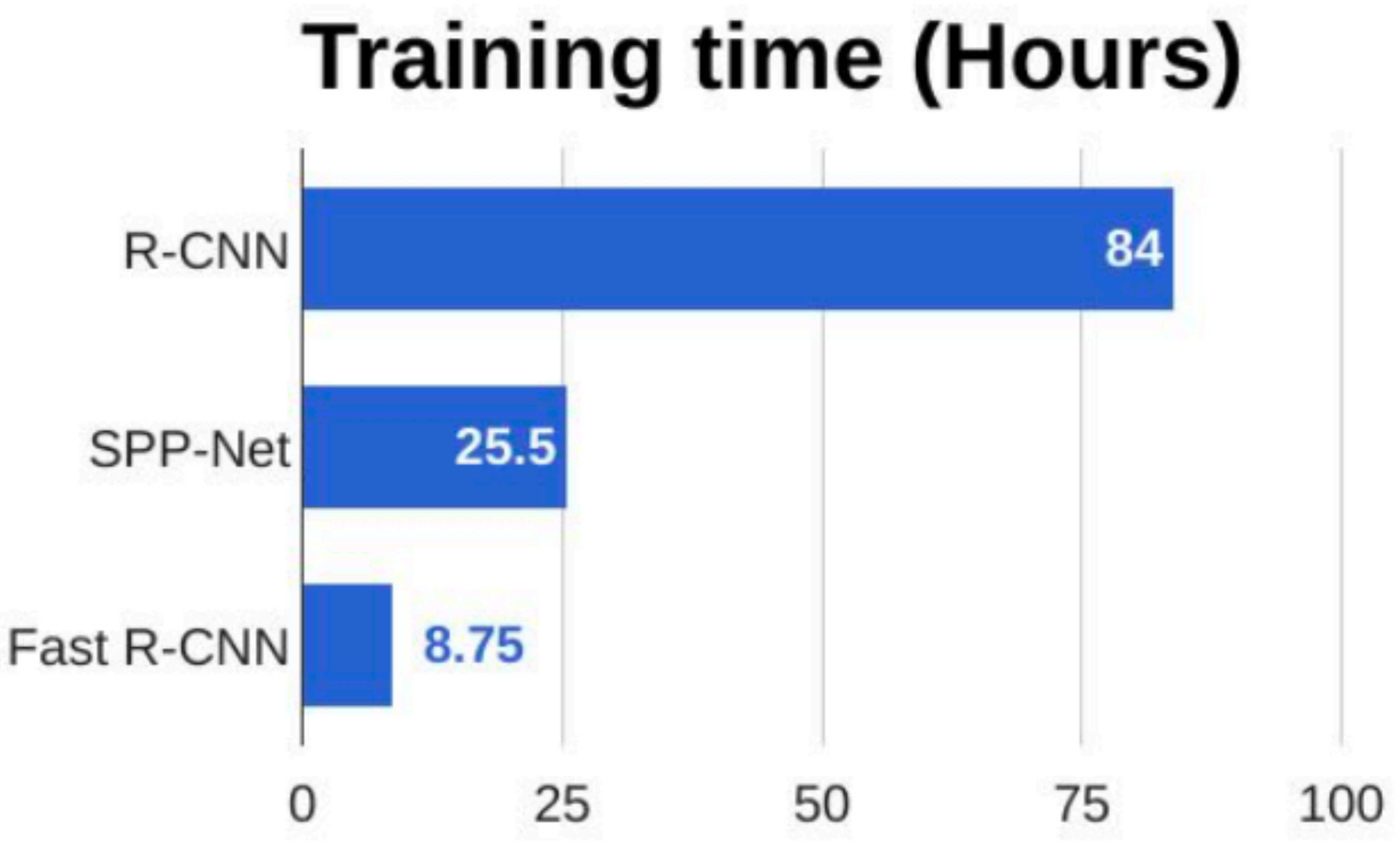
Forward prop the **whole image** through CNN

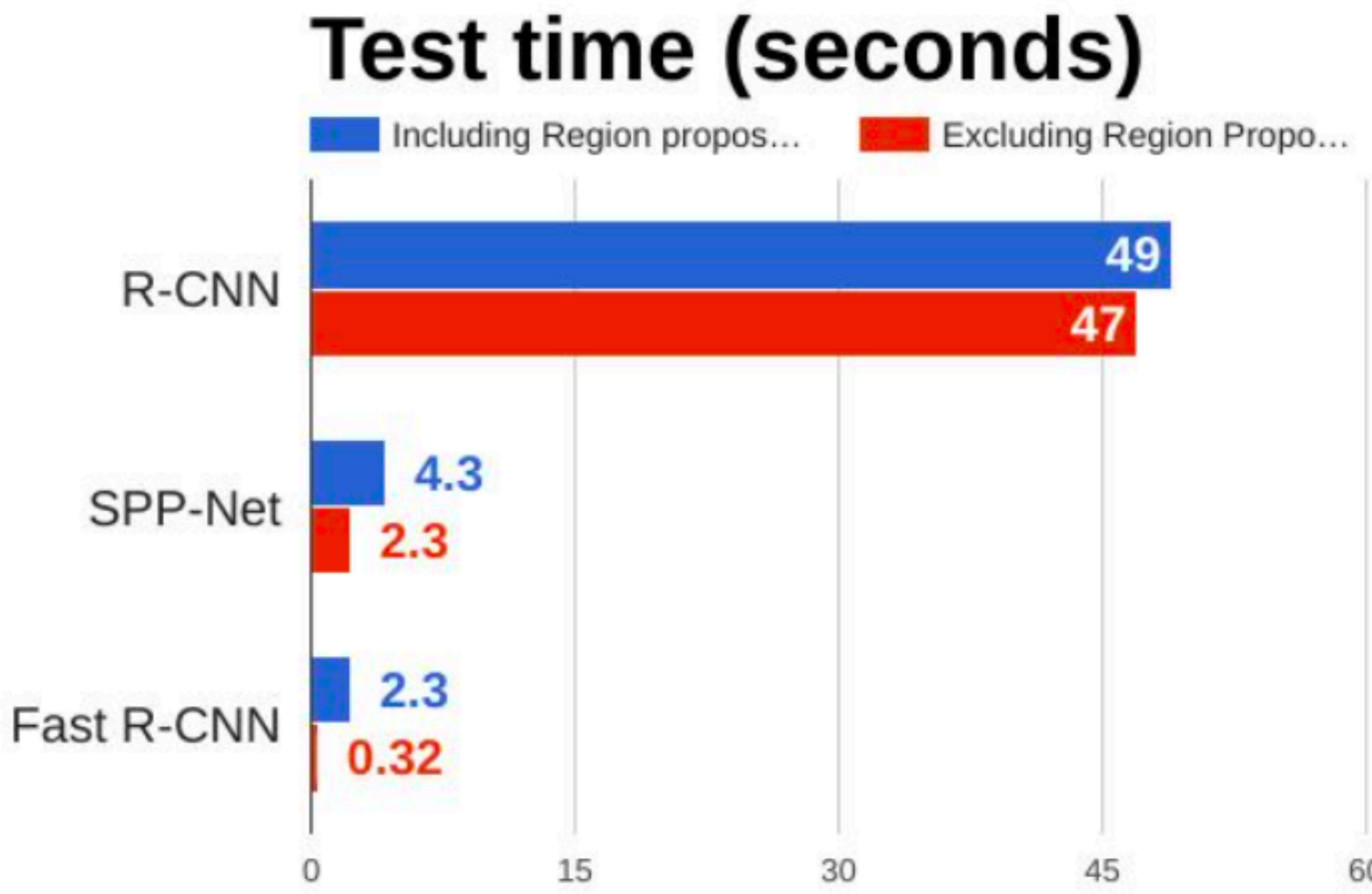
Input **Image**

* image from Ross Girshick

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

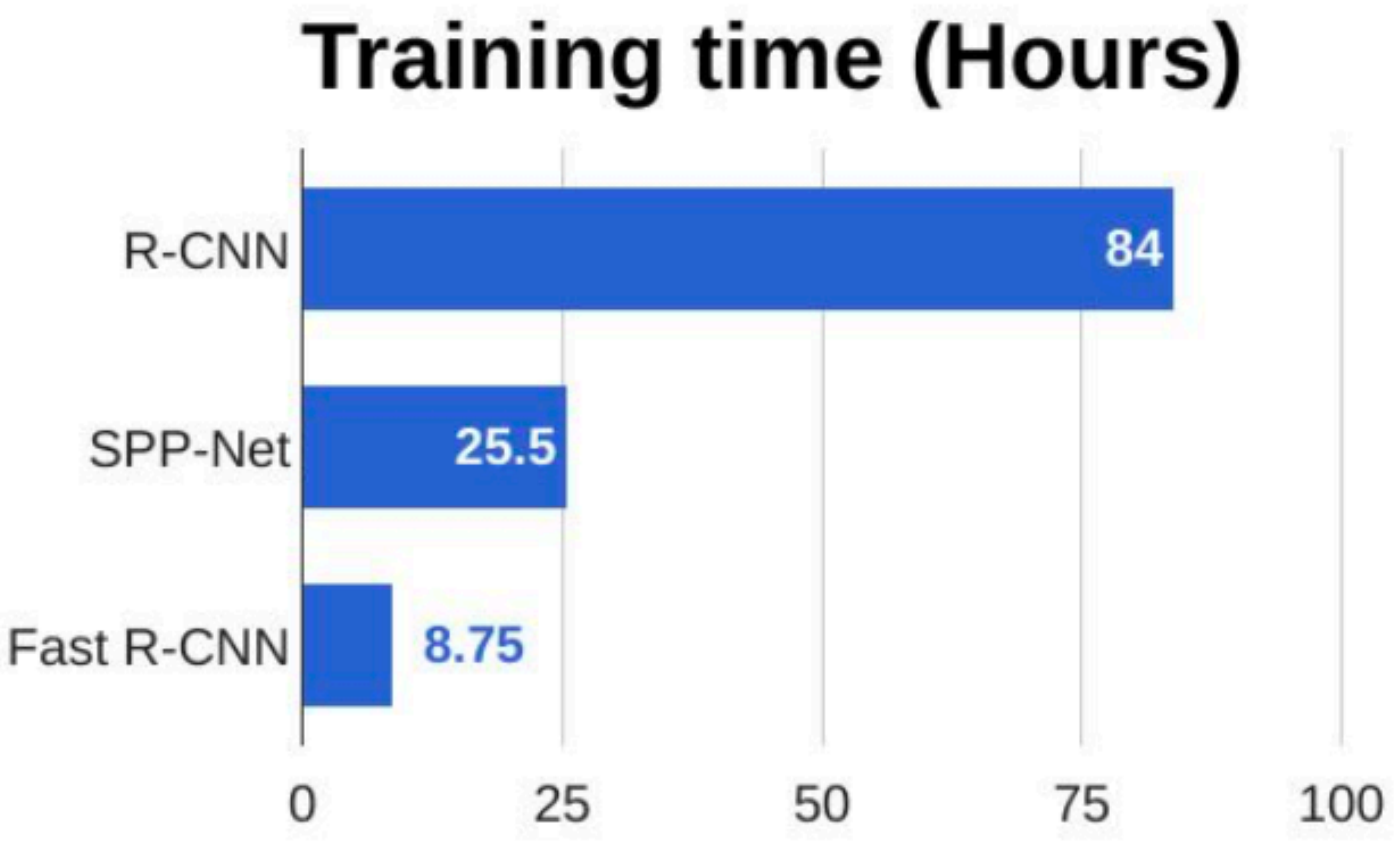
[Girshick et al, CVPR 2014]
[Girshick et al, ICCV 2015]
[He et al, ECCV 2014]

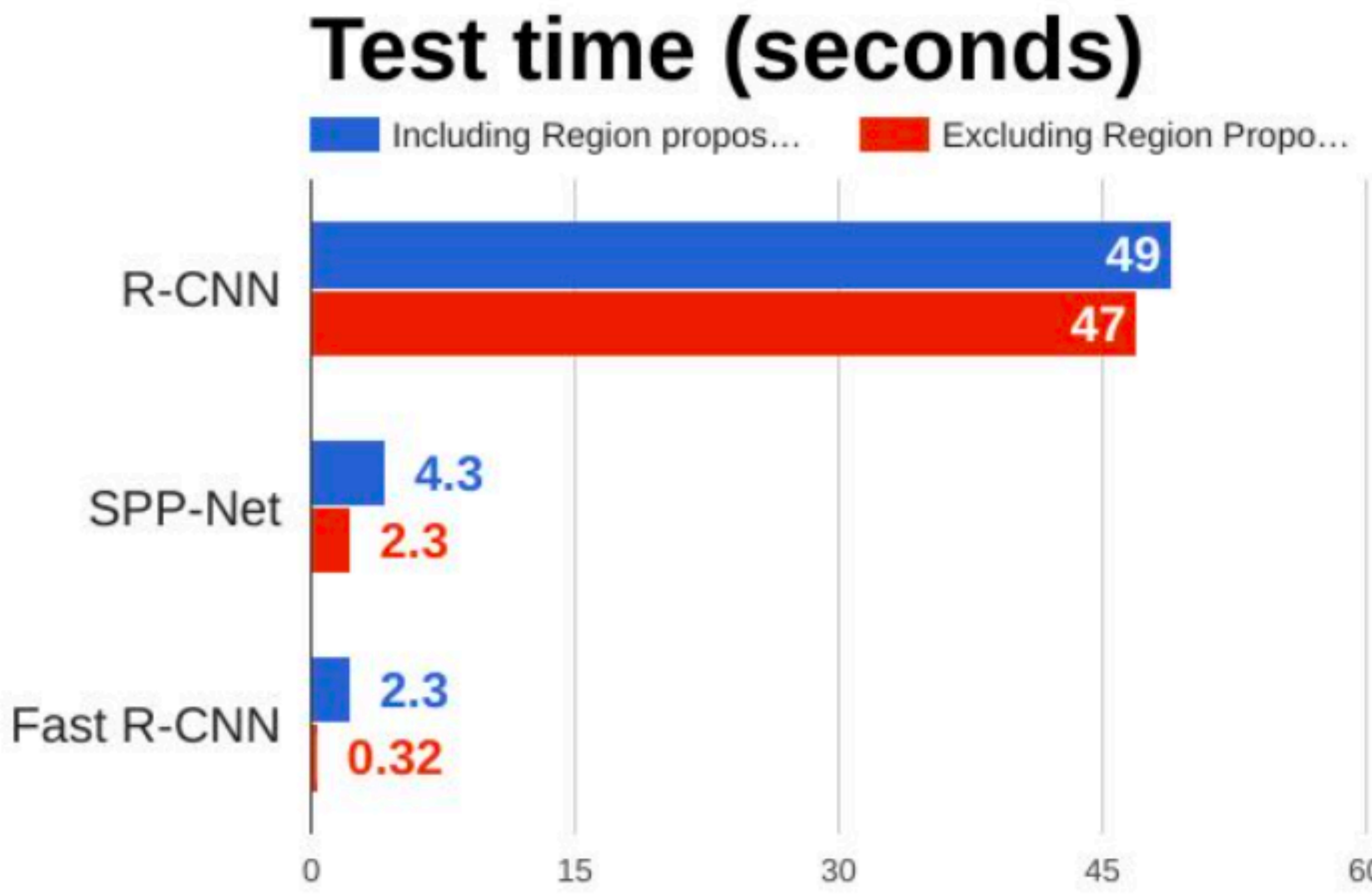




R-CNN vs. SPP vs. Fast R-CNN

[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!

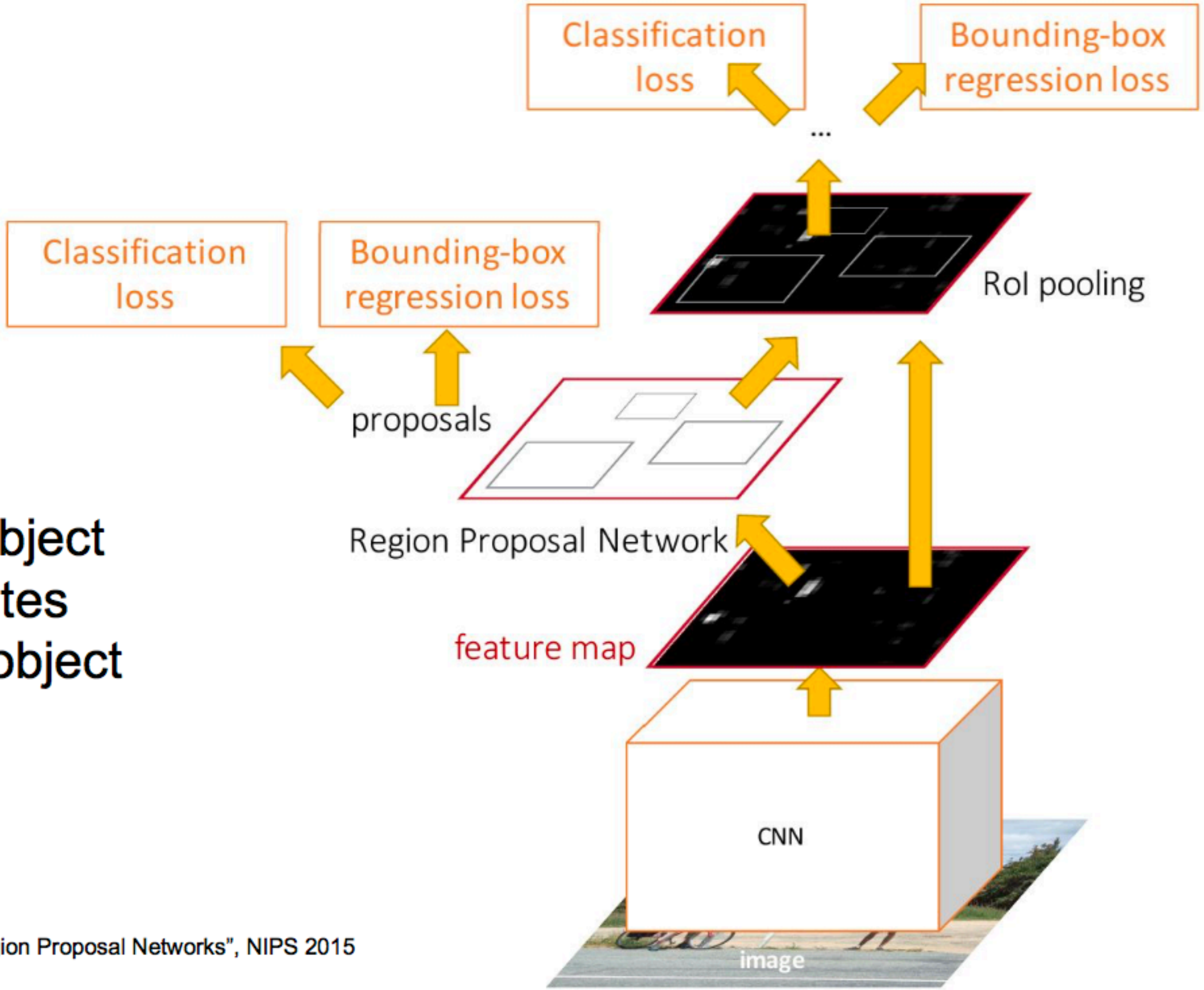
Faster **R-CNN**

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



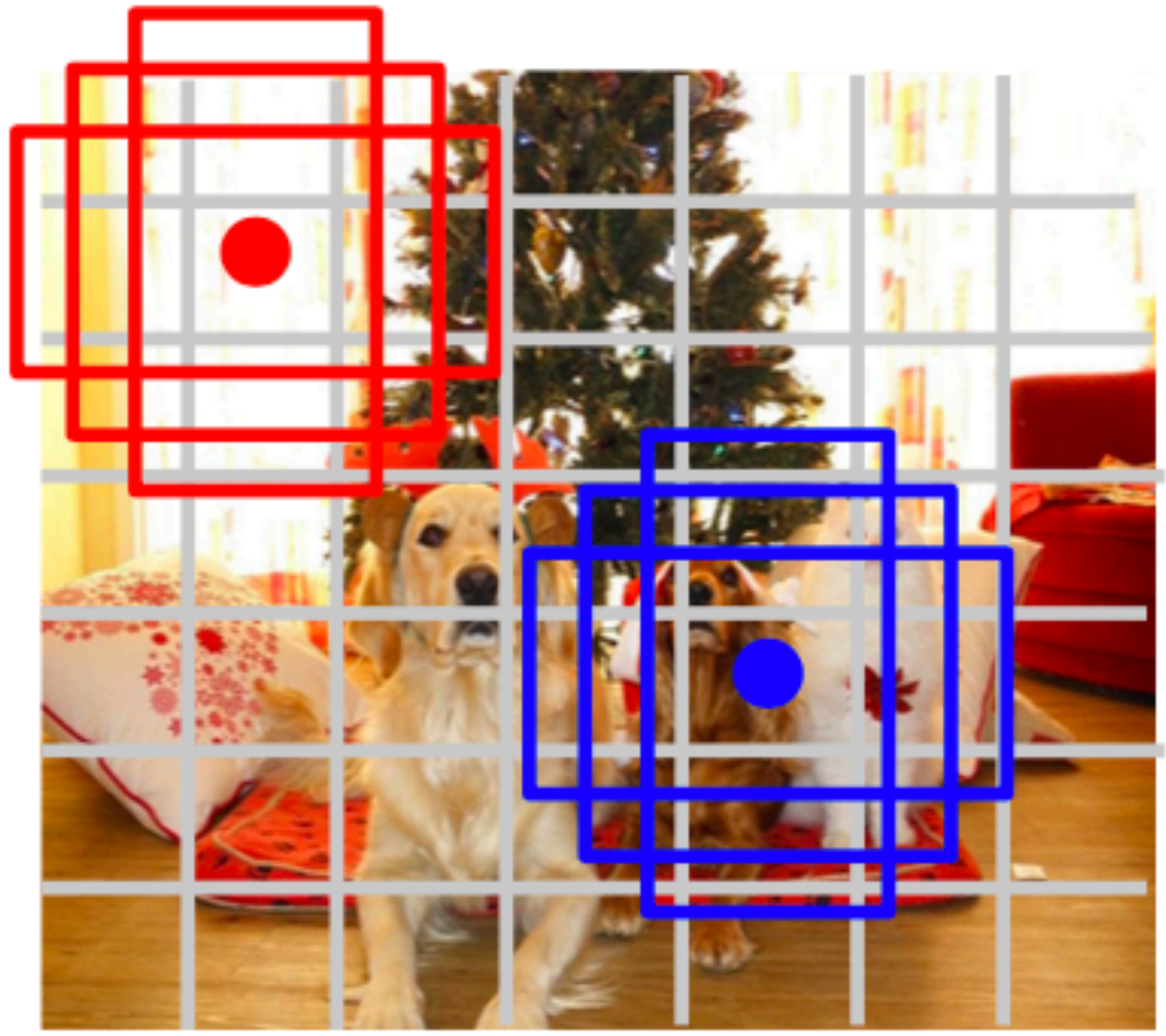
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

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

YOLO: You Only Look Once

[Redmon et al, CVPR 2016]

Input image
3 x H x W



Divide image into grid
7 x 7

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

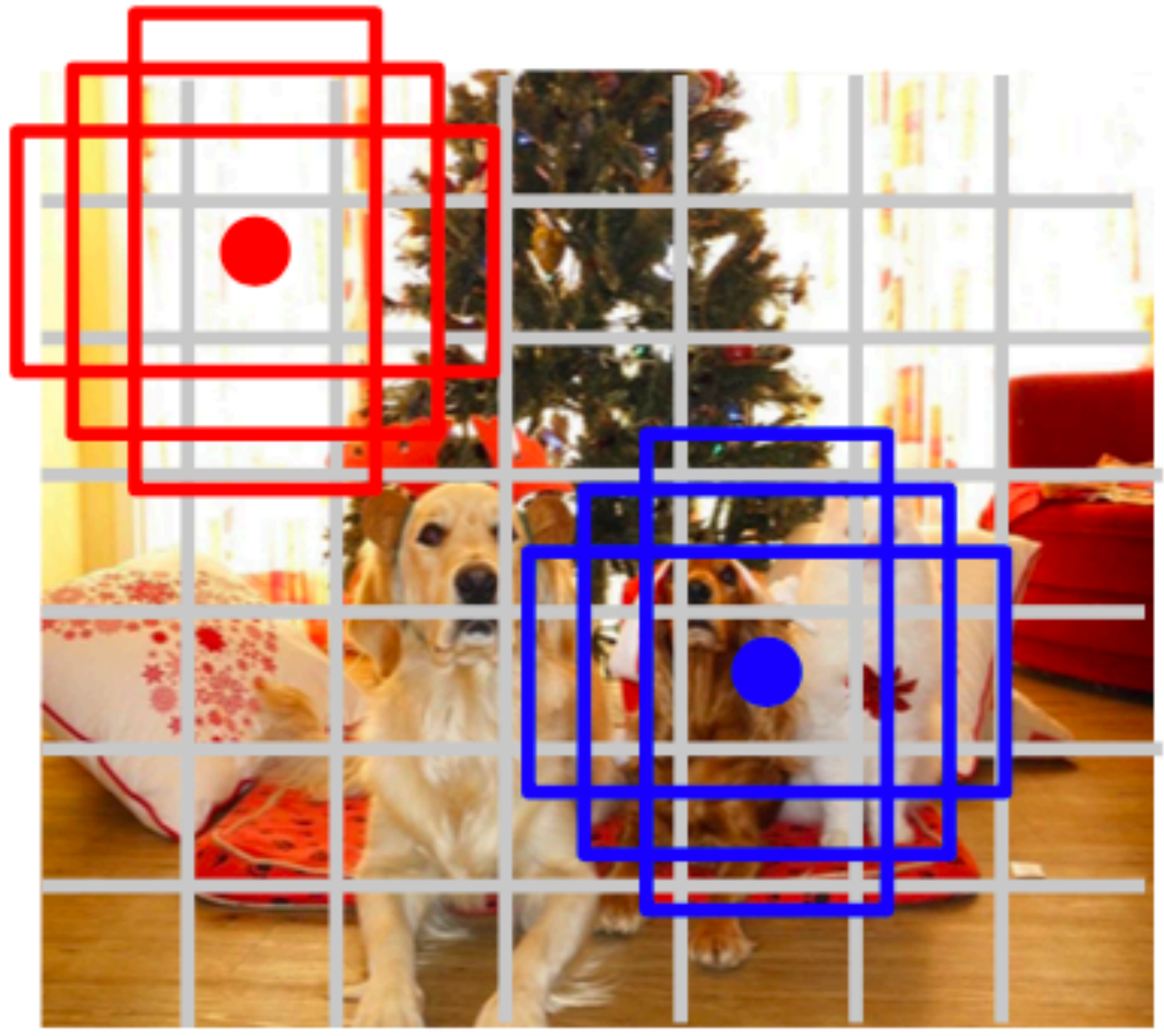
Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers:
  (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)

Output:
7 x 7 x (5 * B + C)

YOLO: You Only Look Once

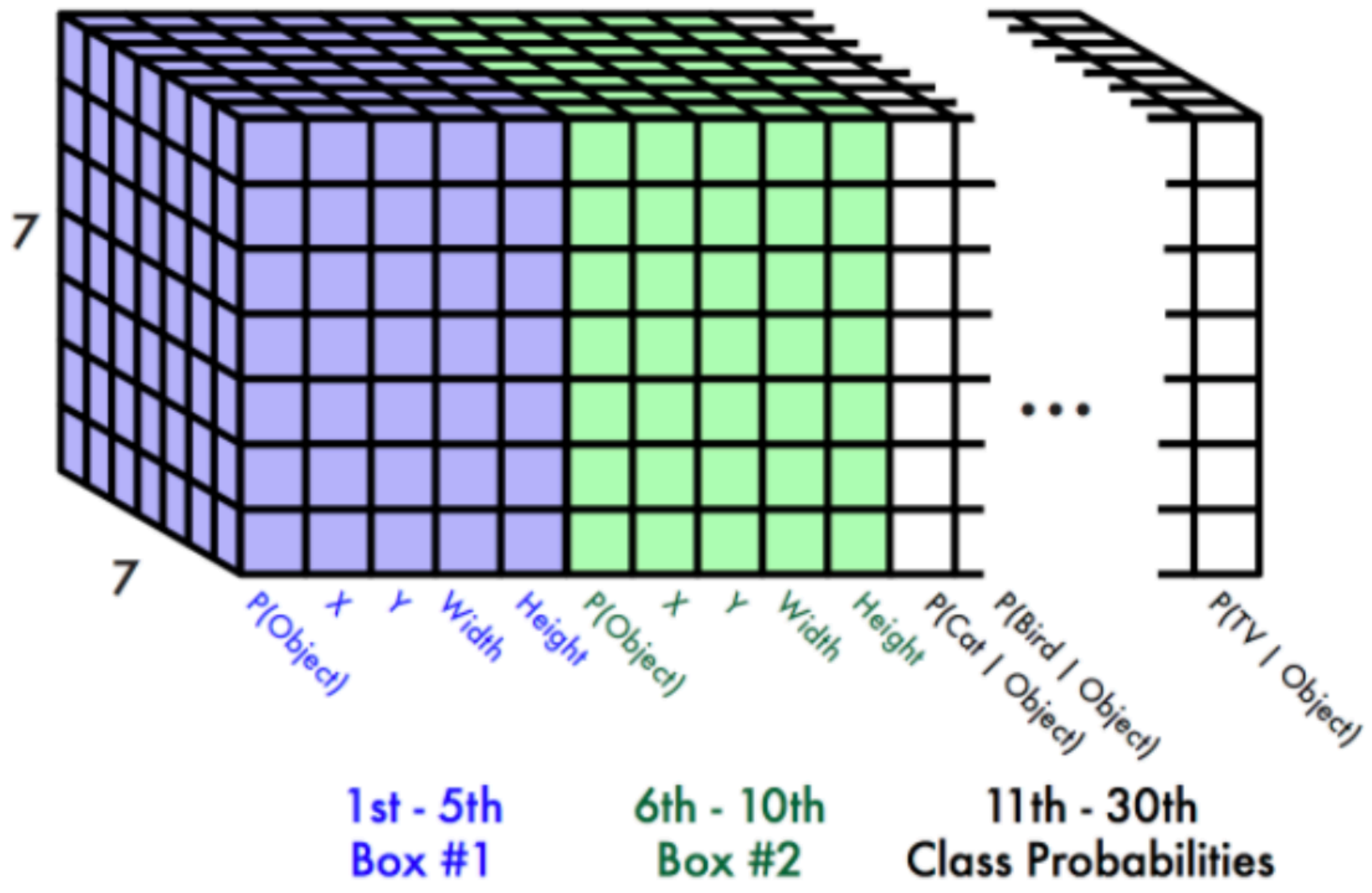
[Redmon et al, CVPR 2016]

Input image
3 x H x W



Divide image into grid
7 x 7

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



YOLO v2

http://pjreddie.com/yolo

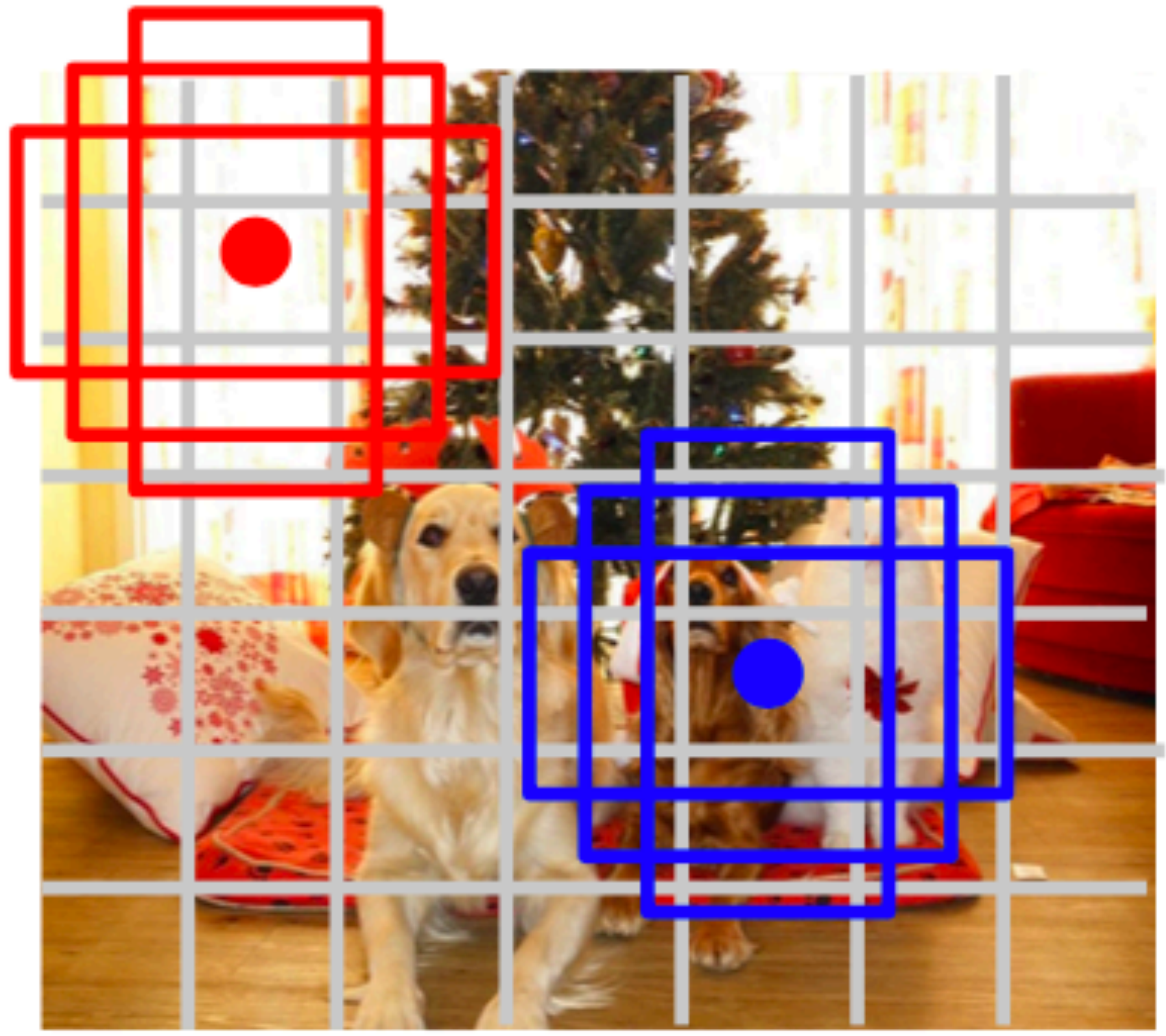
YOLO v2

http://pjreddie.com/yolo

YOLO: You Only Look Once

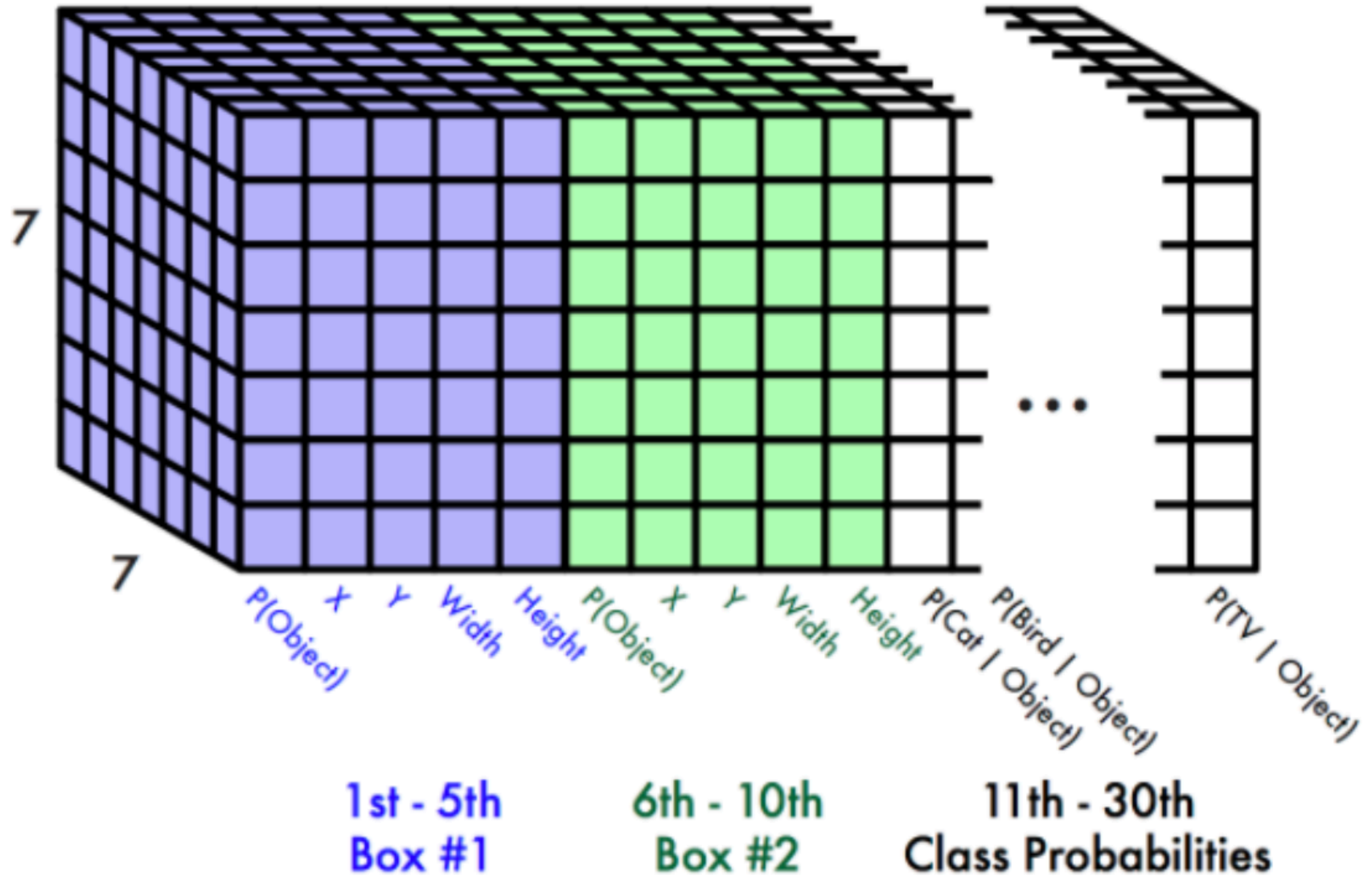
[Redmon et al, CVPR 2016]

Input image
3 x H x W



Divide image into grid
7 x 7

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



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

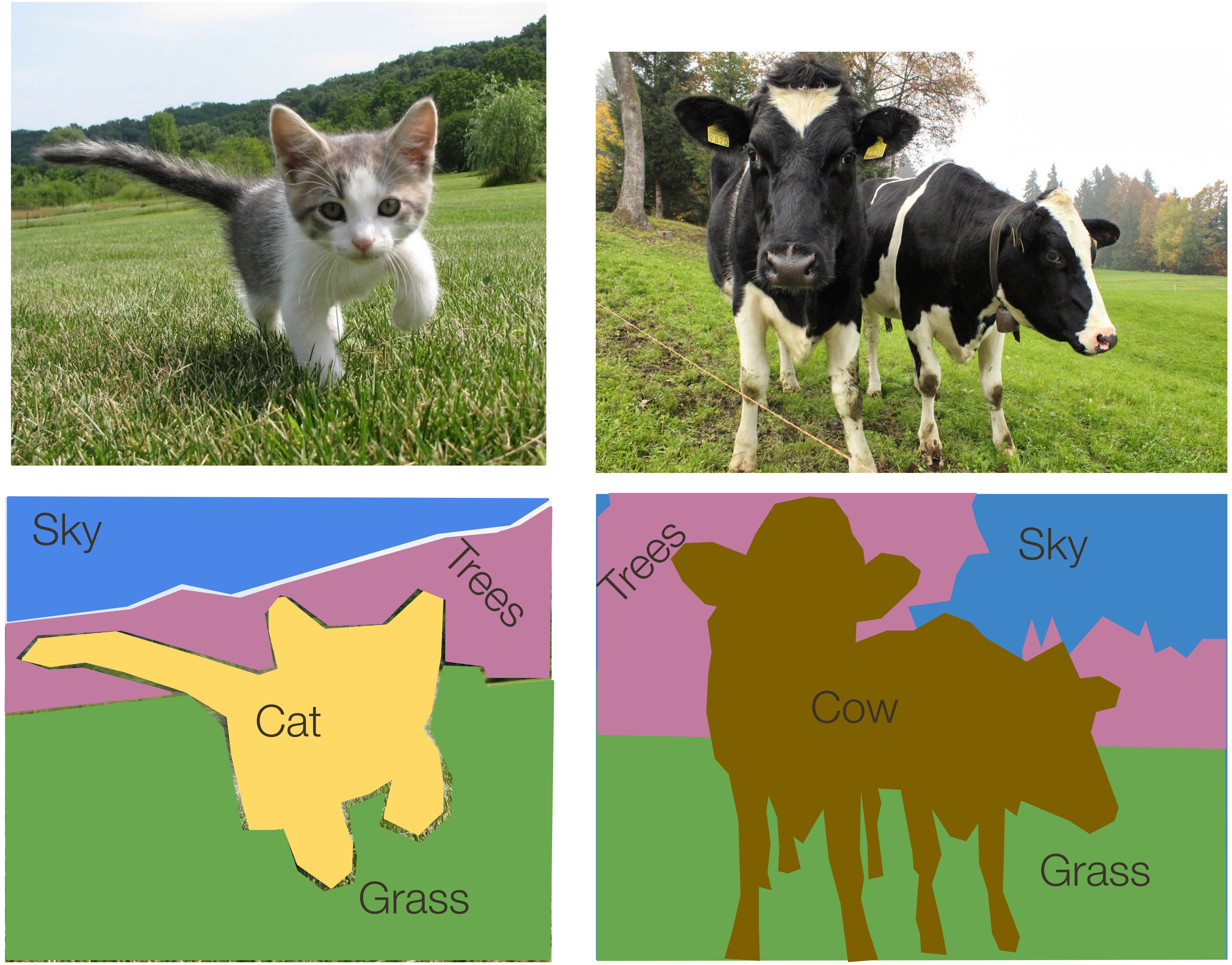
Segmentation

Horse
Person

COCO Common Objects in Context

Semantic **Segmentation**

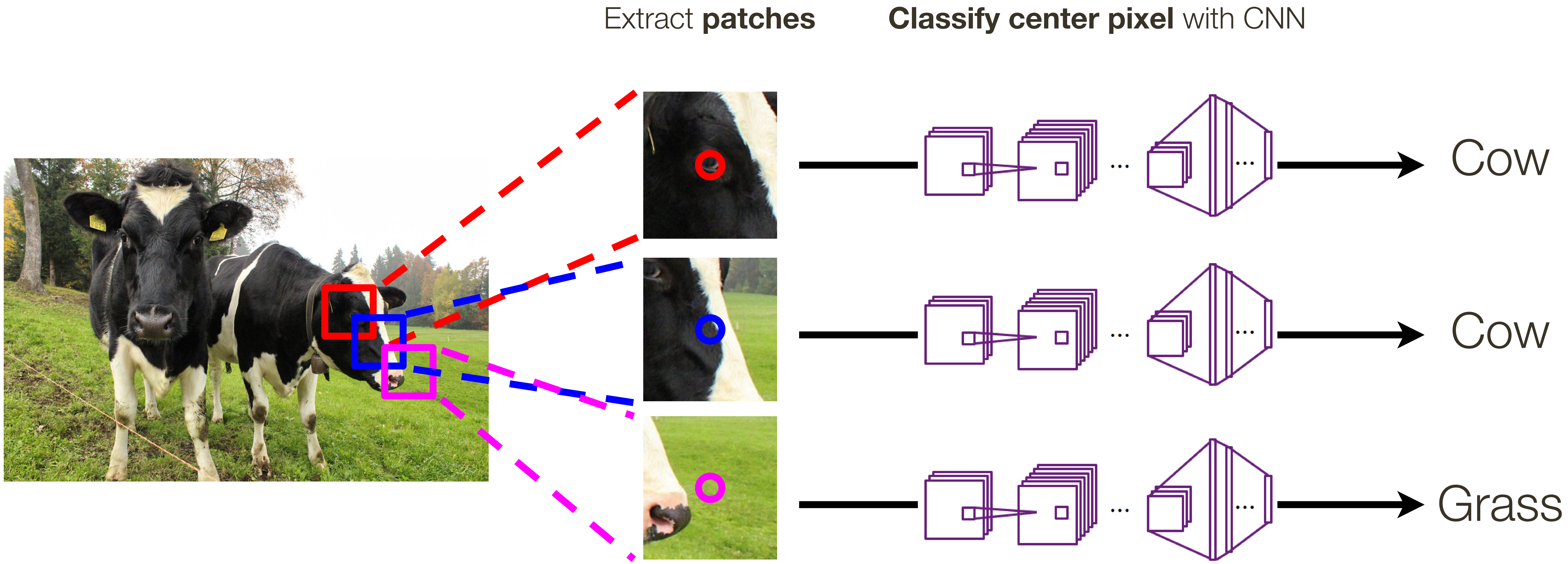
Label **every pixel** with a category label (without differentiating instances)



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

Semantic **Segmentation:** Sliding Window

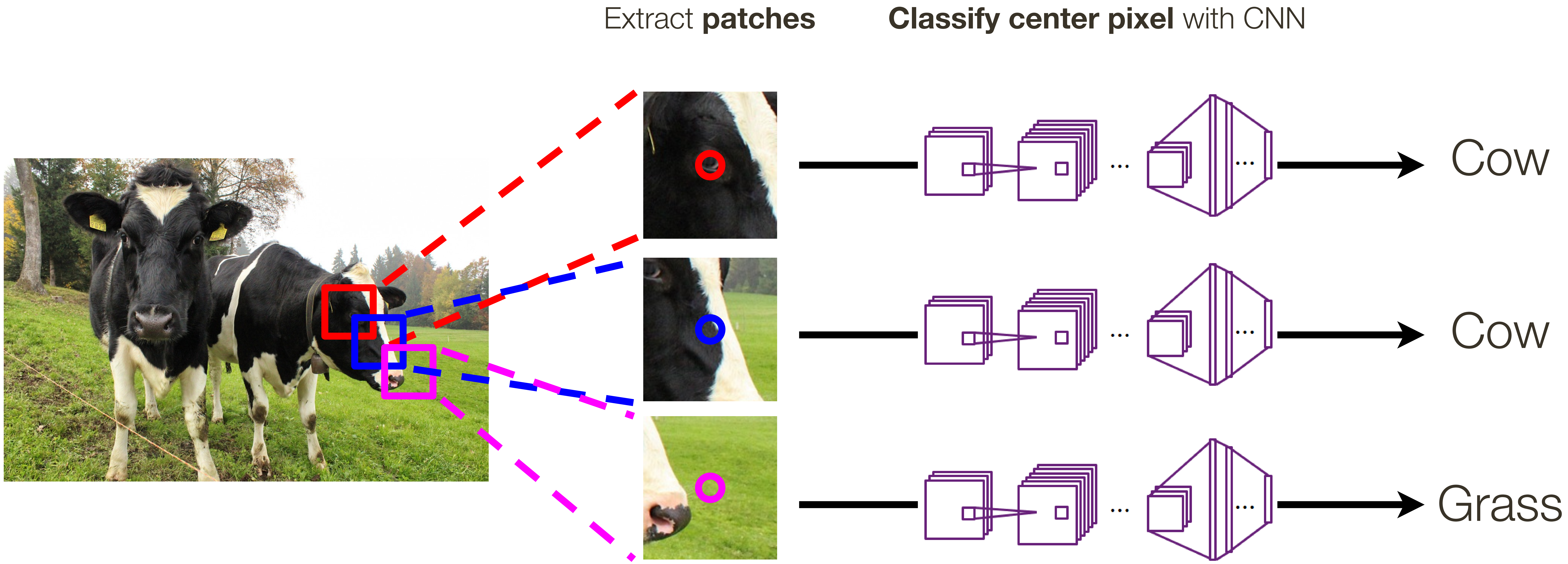
[Farabet et al, TPAMI 2013]
[Pinheiro et al, ICML 2014]



* 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]

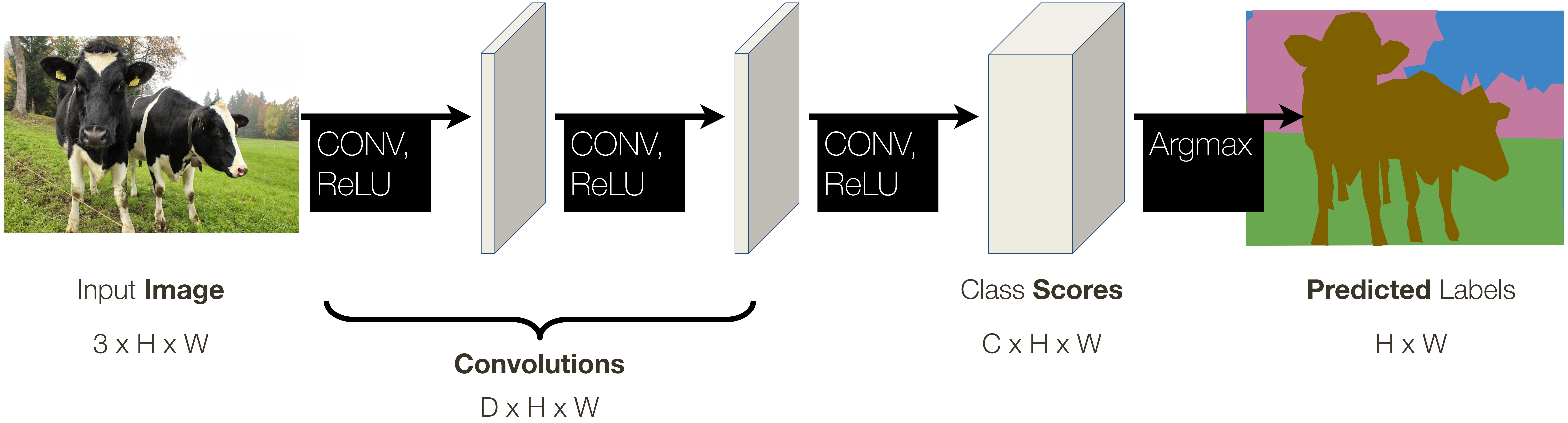


Problem: VERY inefficient, no reuse of computations for overlapping patches

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

Semantic **Segmentation:** Fully Convolutional CNNs

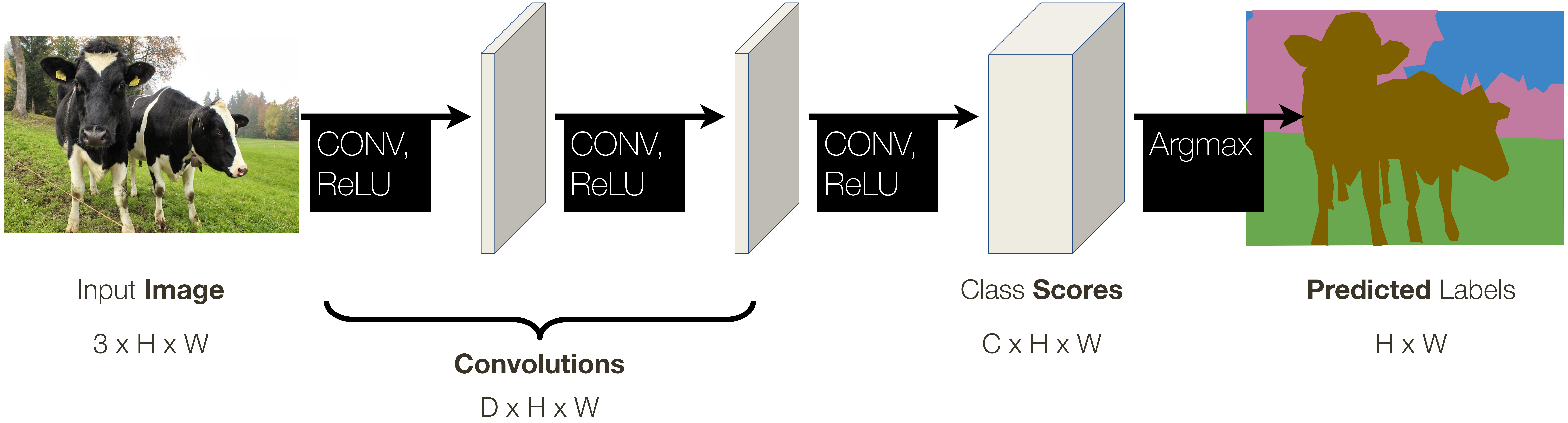
Design a network as a number of convolutional layers to make predictions for all pixels at once!



* 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

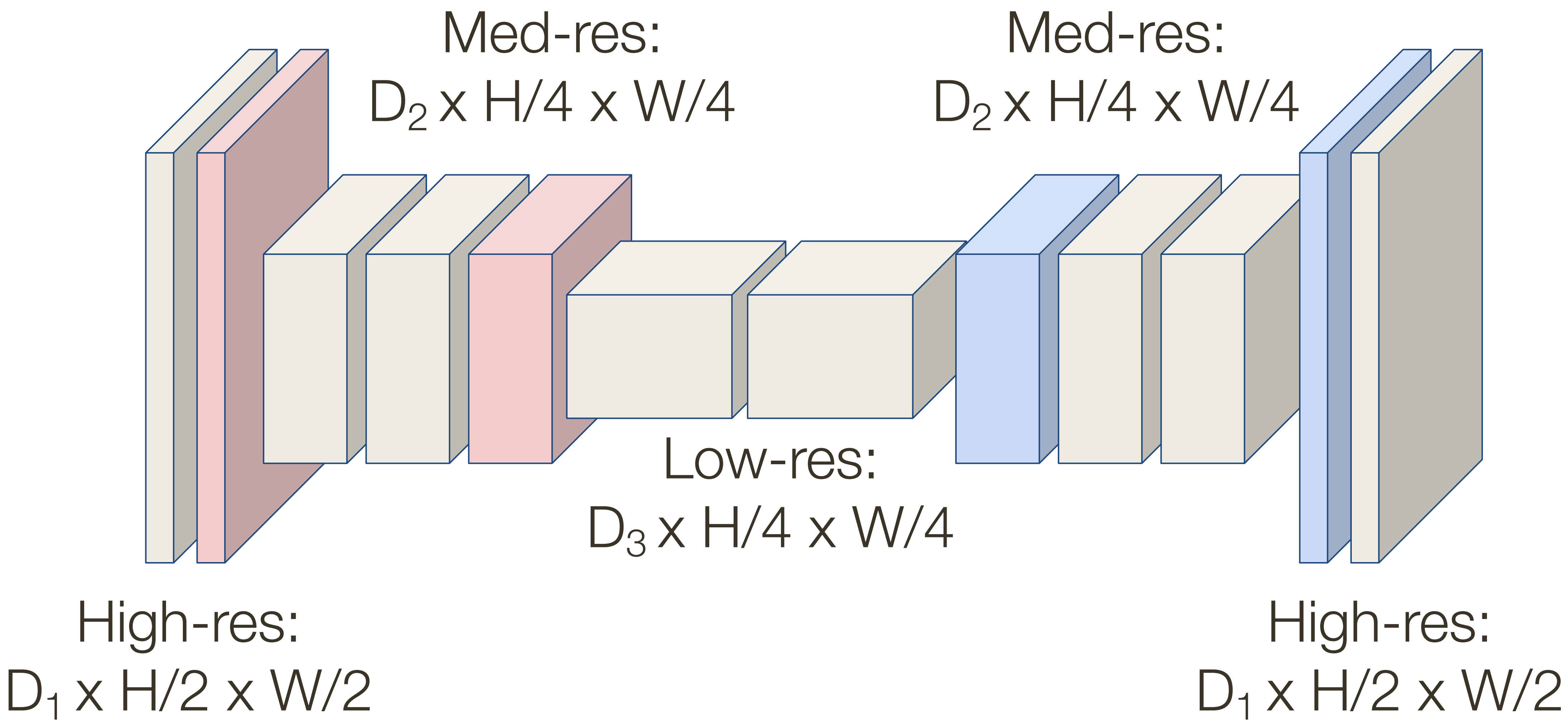
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

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]

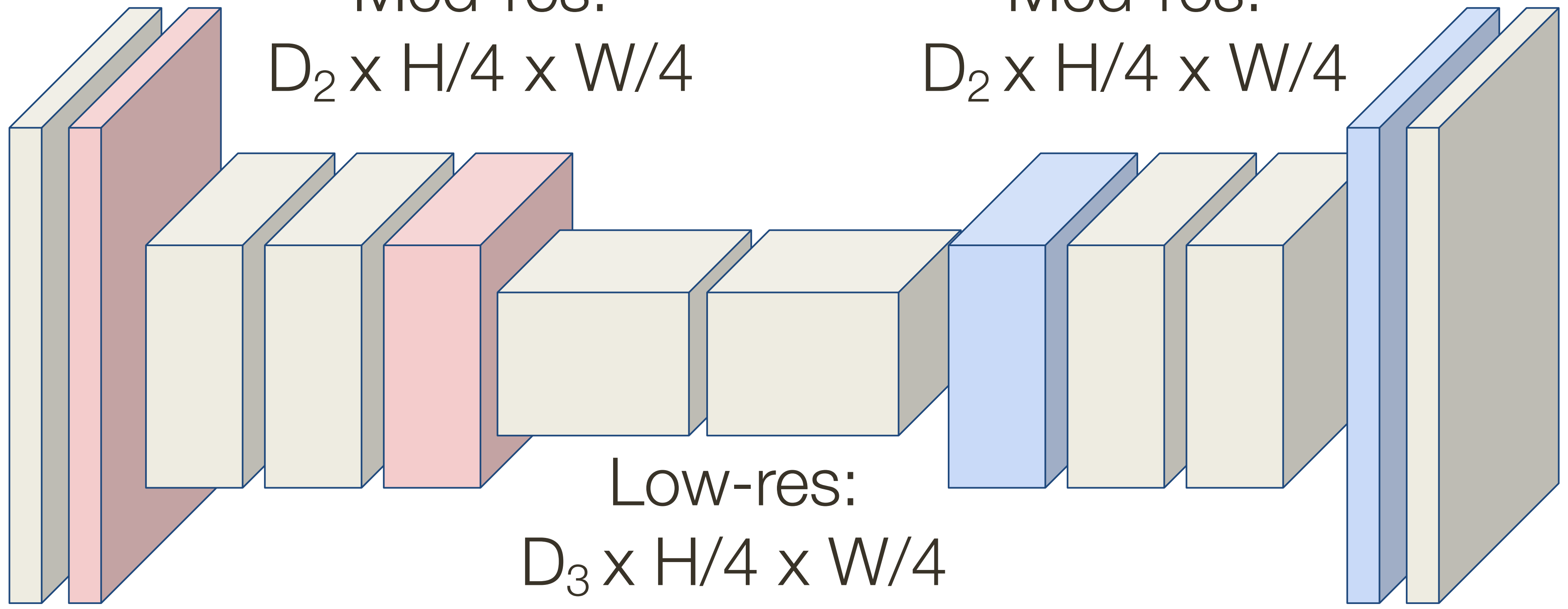
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

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



Med-res: D_2 x H/4 x W/4

Med-res: D_2 x H/4 x W/4

Low-res: D_3 x H/4 x W/4

High-res: D_1 x H/2 x W/2

High-res: D_1 x H/2 x W/2

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**

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

Nearest Neighbor

$$\begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array} \longrightarrow \begin{array}{cc|cc} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ \hline 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{array}$$

Input: 2 x 2 **Output:** 4 x 4

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

Nearest Neighbor

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \longrightarrow \left.\begin{array}{cc|cc} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ \hline 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{array}\right.$$

Input: 2 x 2 **Output:** 4 x 4

"Bed of Nails"

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \longrightarrow \left.\begin{array}{cc|cc} 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 3 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right.$$

Input: 2 x 2 **Output:** 4 x 4

* 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

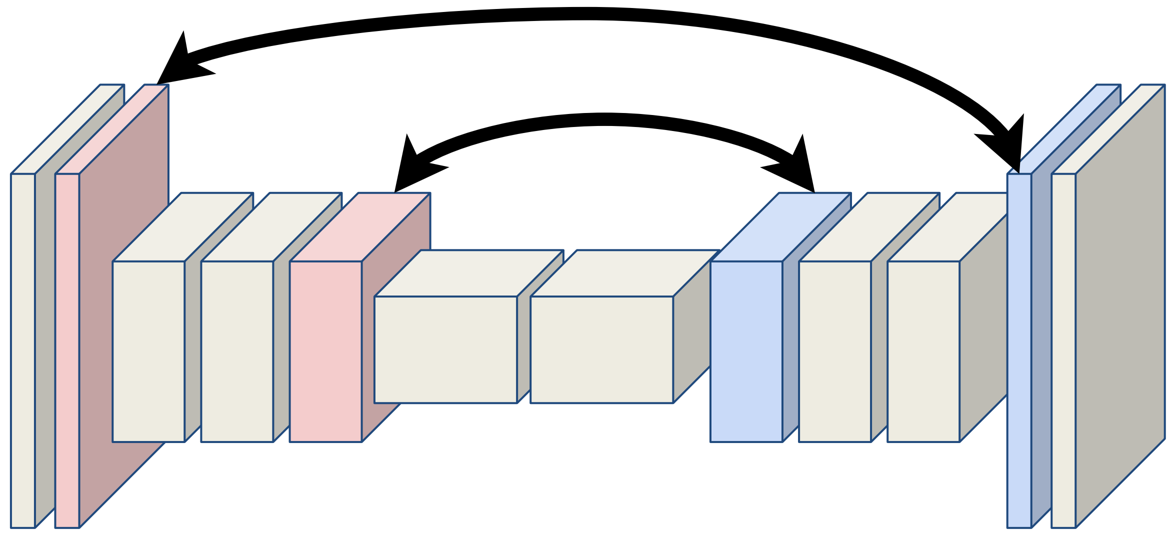
Max Unpooling
Use positions from pooling layer

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

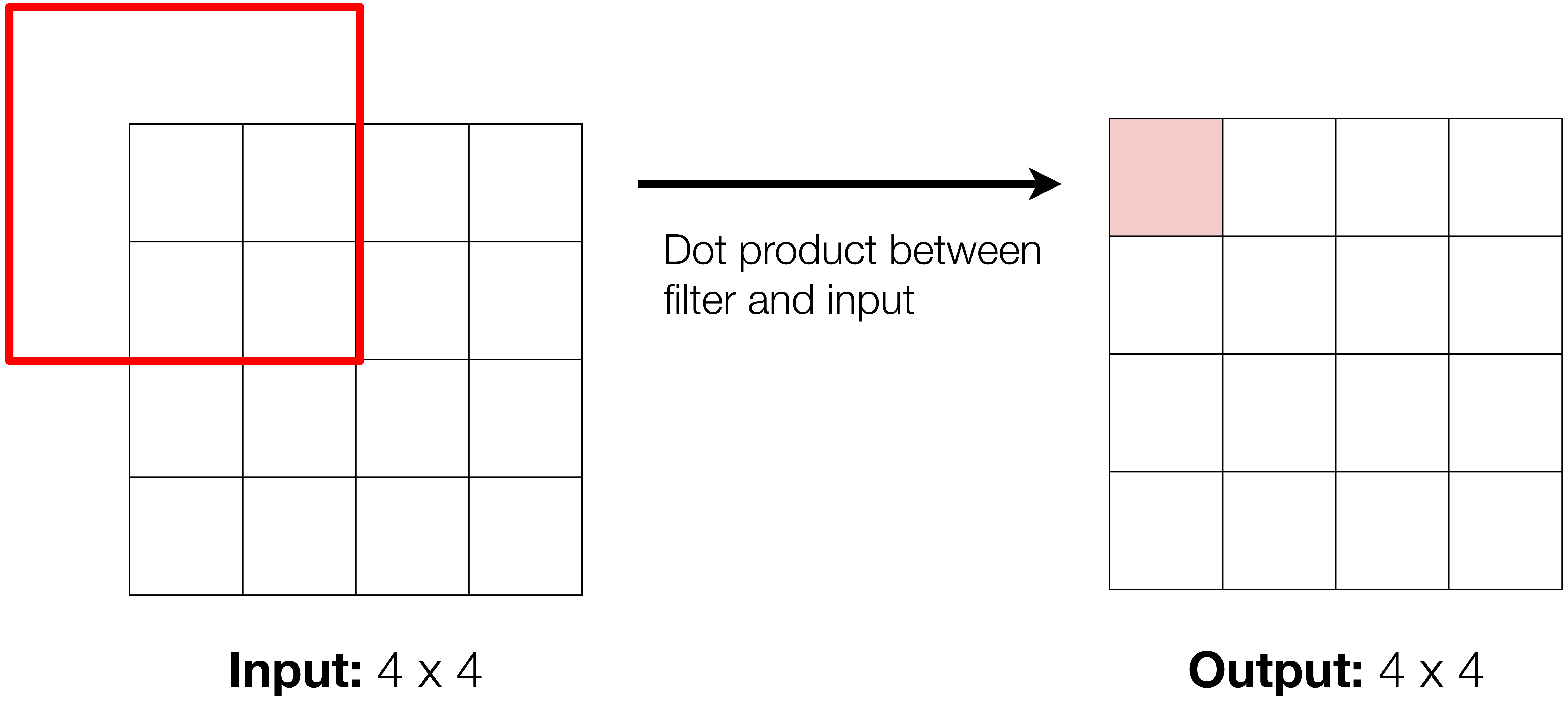


Corresponding pairs of downsampling and upsampling layers

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

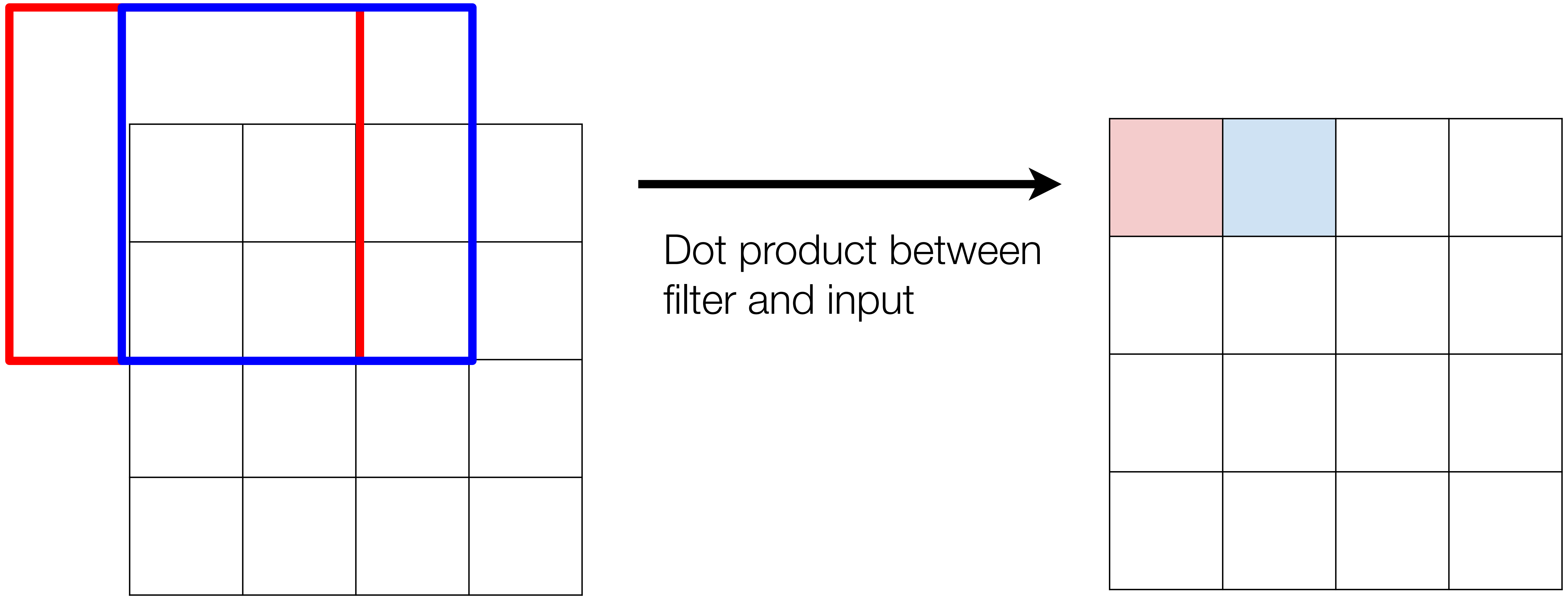
In-network **Up Sampling:** Transpose Convolution

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

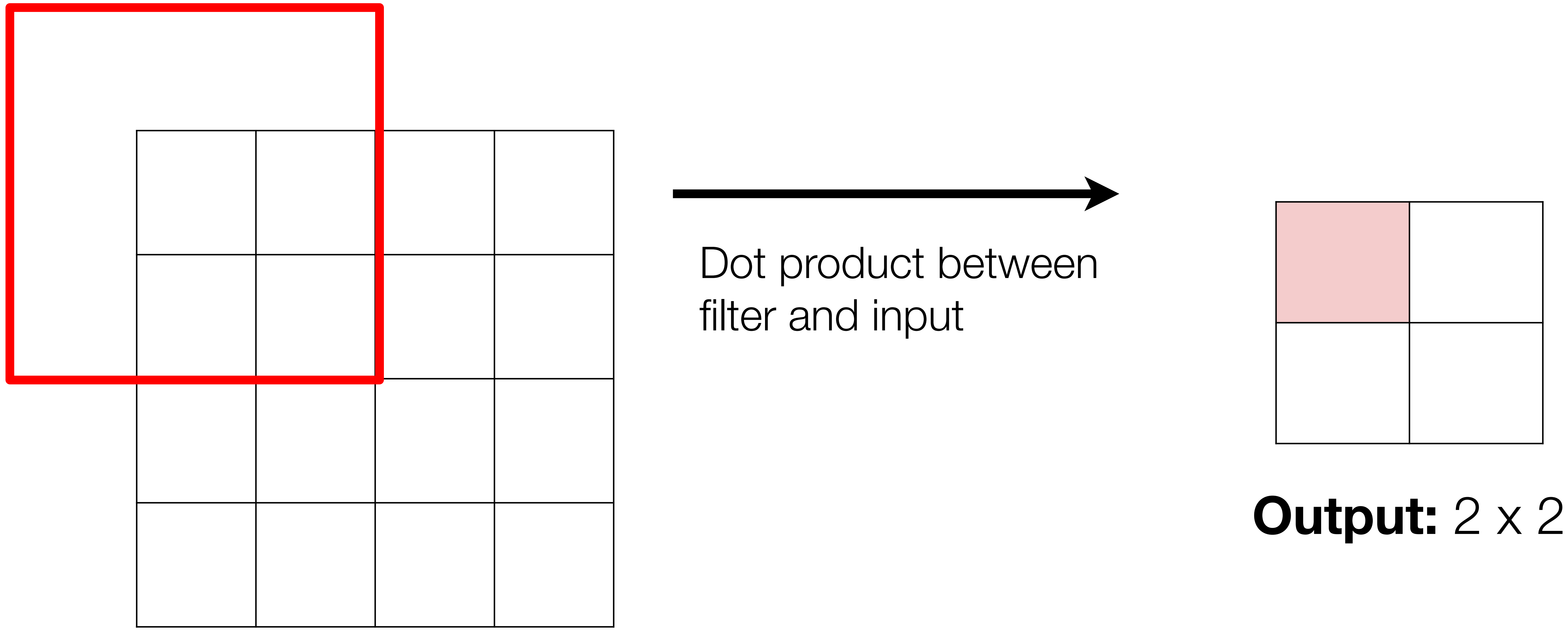


Input: 4 x 4

Output: 4 x 4

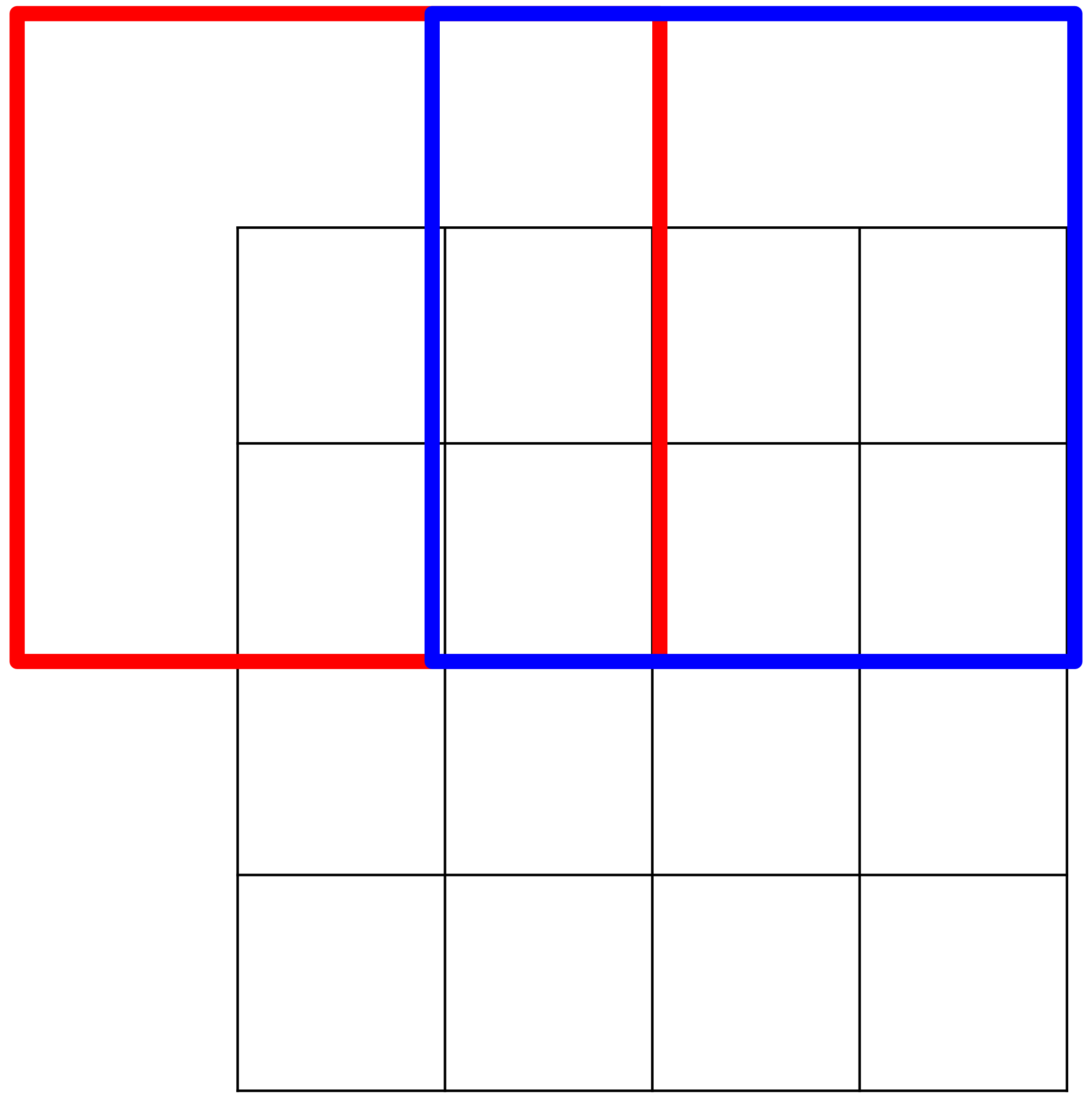
In-network **Up Sampling:** Transpose Convolution

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

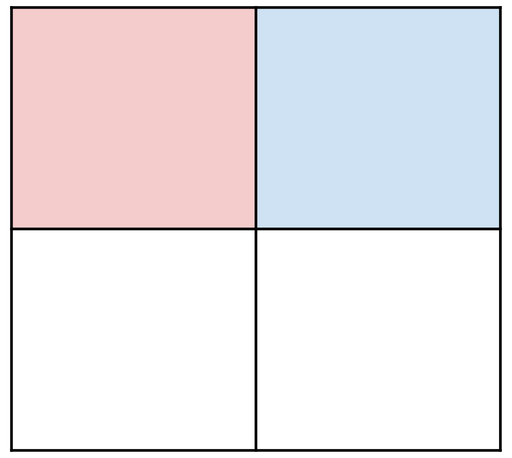


In-network **Up Sampling:** Transpose Convolution

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



Dot product between filter and input



Output: 2 x 2

Input: 4 x 4

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

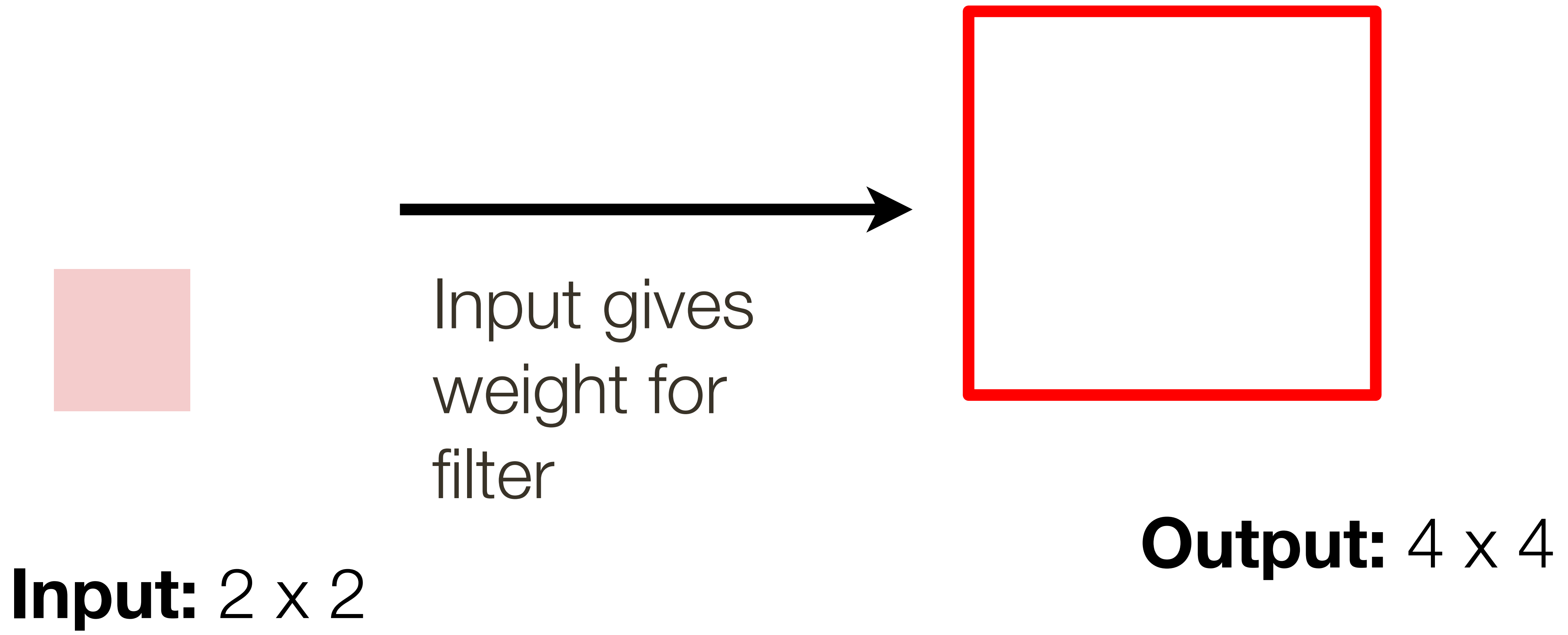
3 x 3 **transpose** convolution, stride 2 pad 1

Input: 2 x 2

Output: 4 x 4

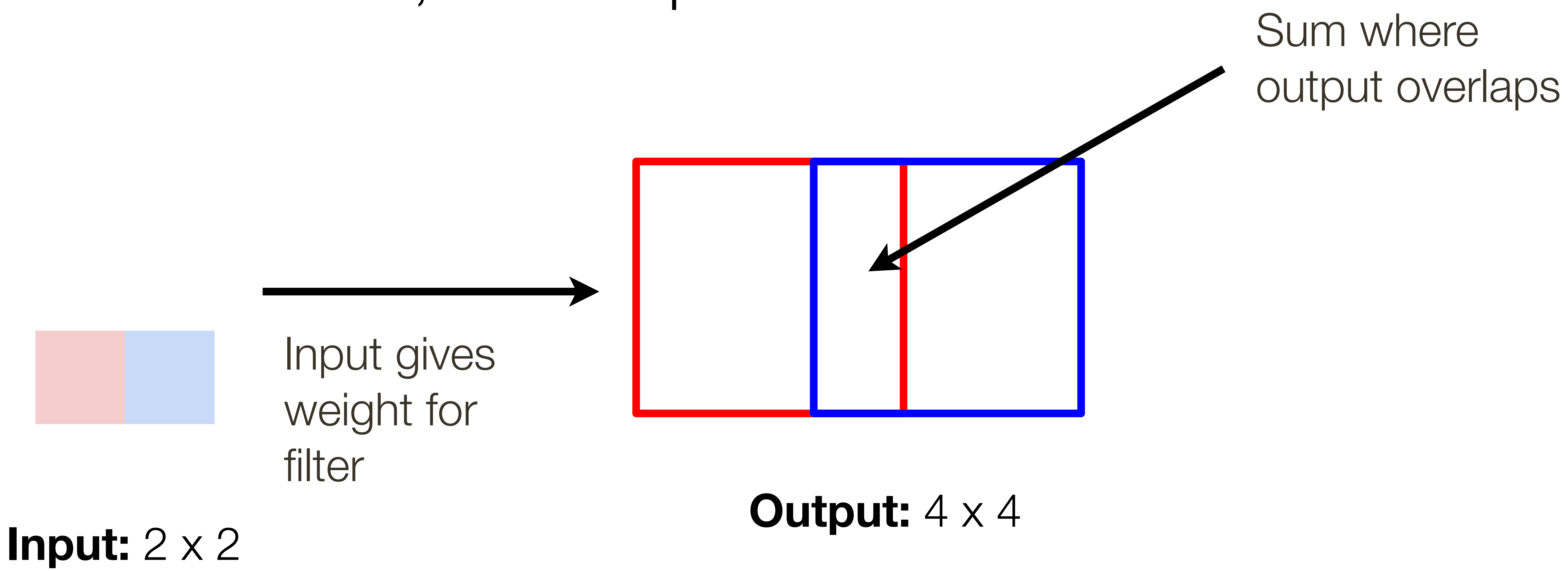
In-network **Up Sampling:** Transpose Convolution

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



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

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

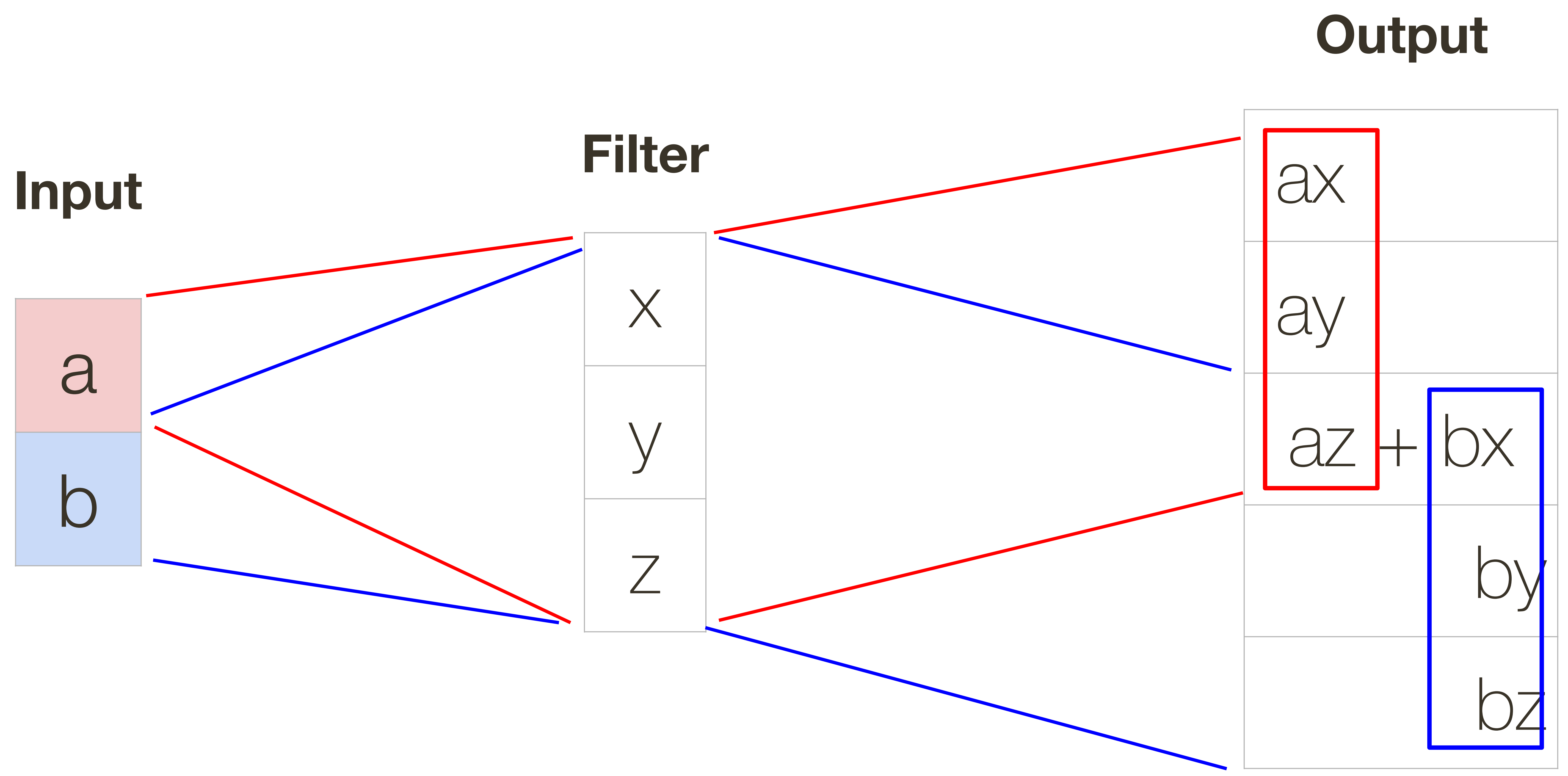


Filter moves 2 pixels in the **output** for every one pixel in the **input**

Stride gives ratio in movement in output vs input

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

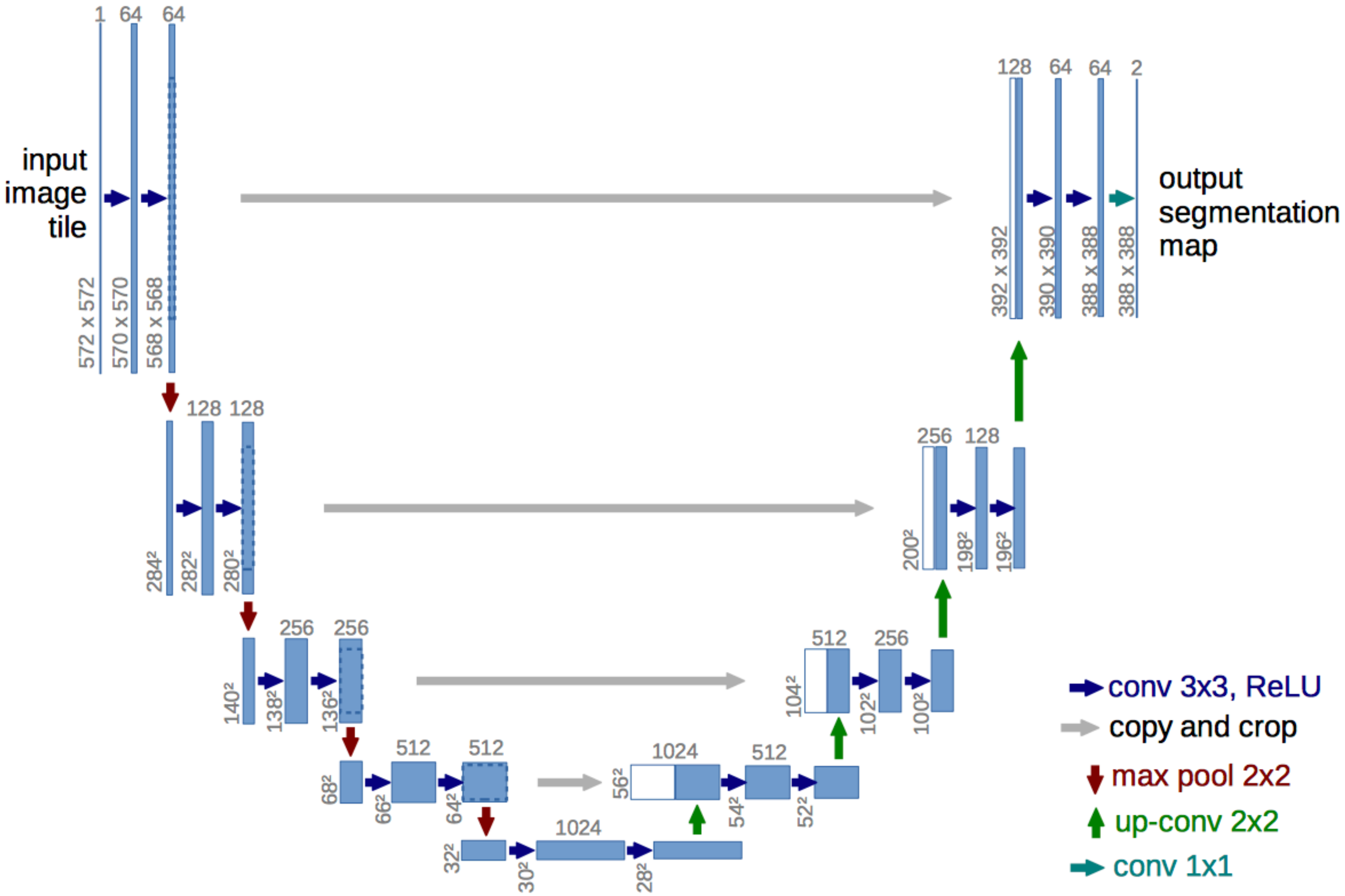
Transpose Convolution: 1-D Example



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



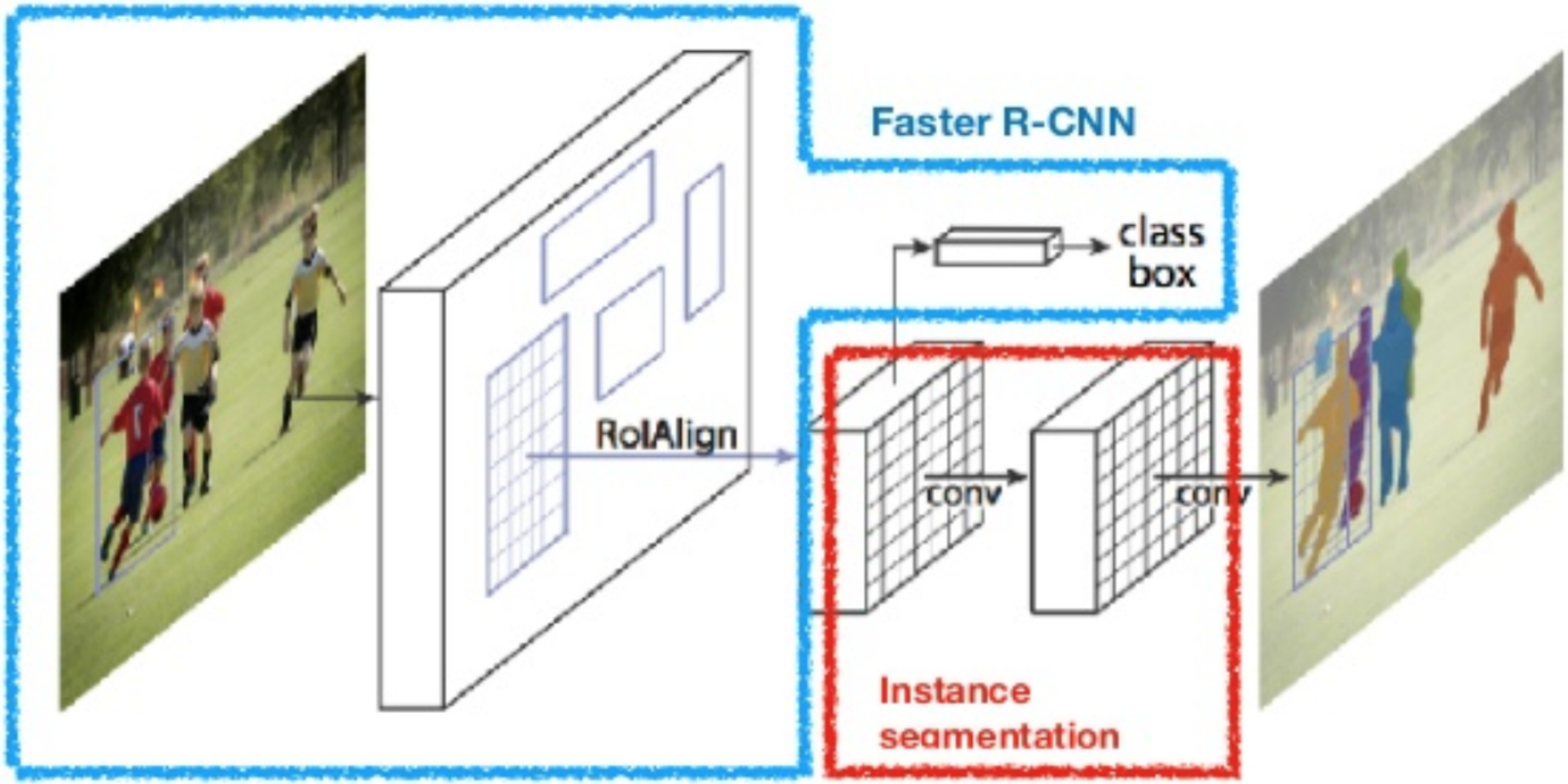
[Ronneberger et al, CVPR 2015]

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

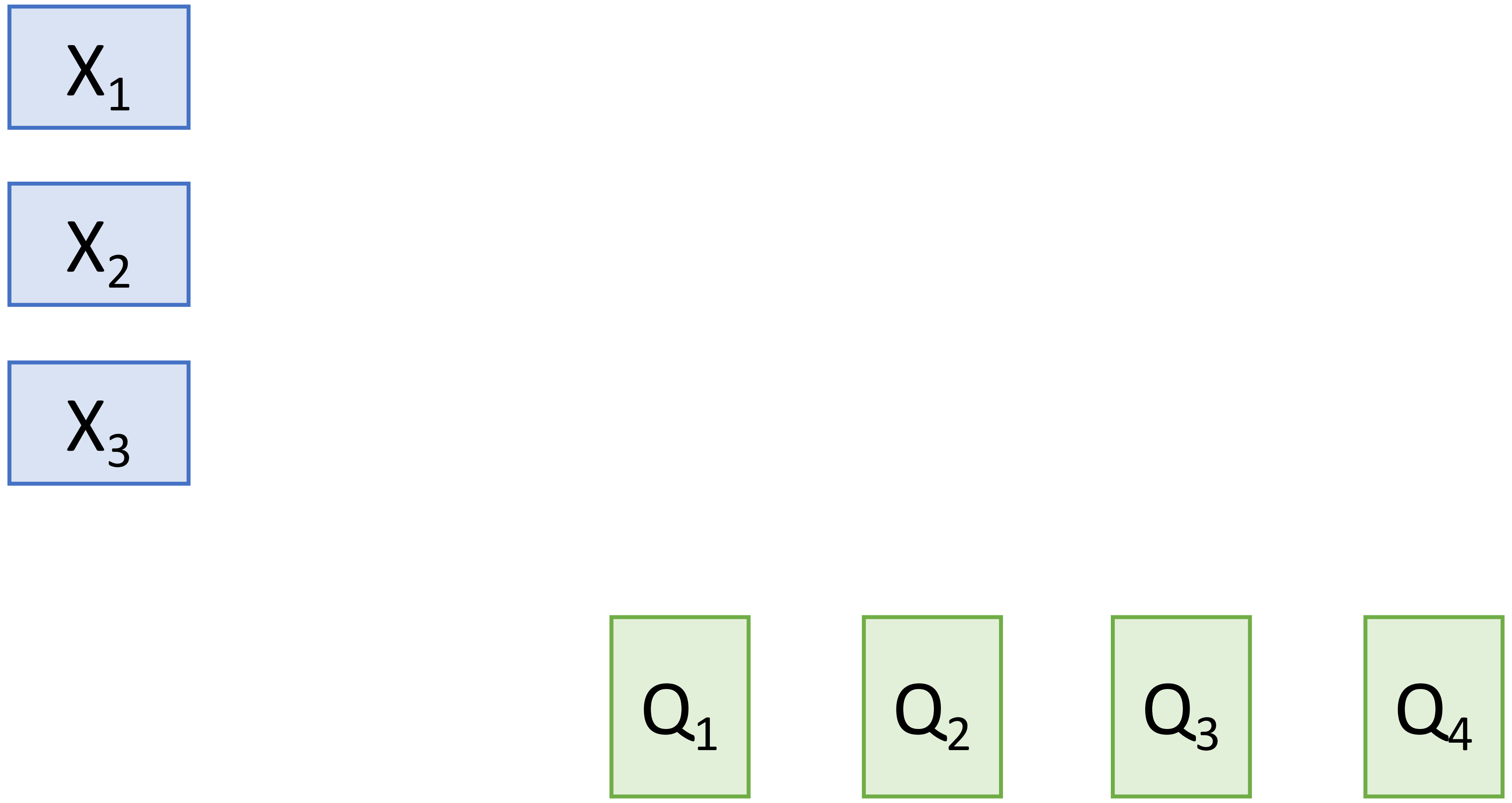
A **fully-connected** layer is the same as in a regular neural network

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

Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

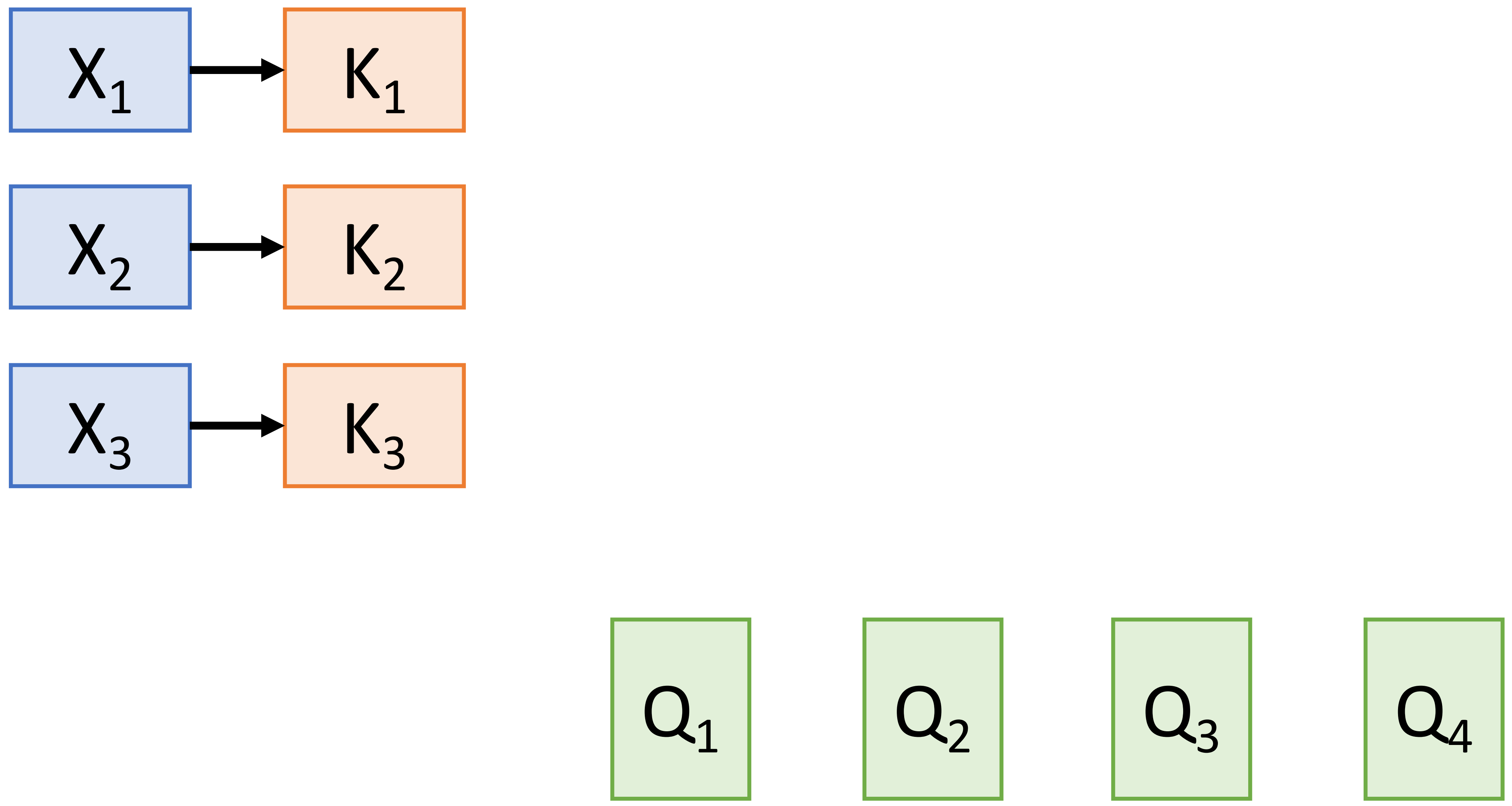


[slide from Justin Johnson, U Michigan]

Attention Layer

Inputs:
Query vectors: $\mathbf{Q}$ (Shape: N_Q x D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

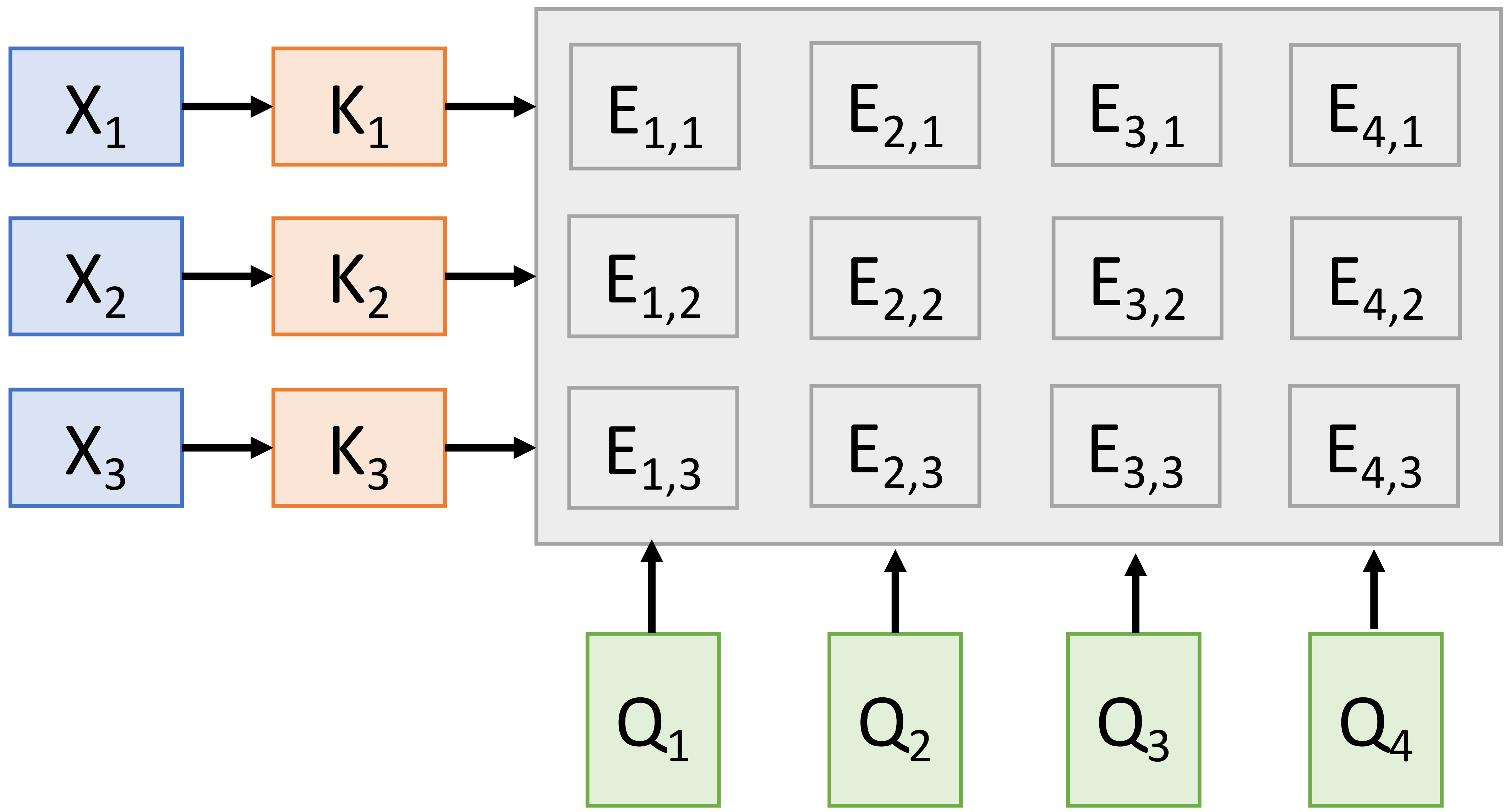
Computation:
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

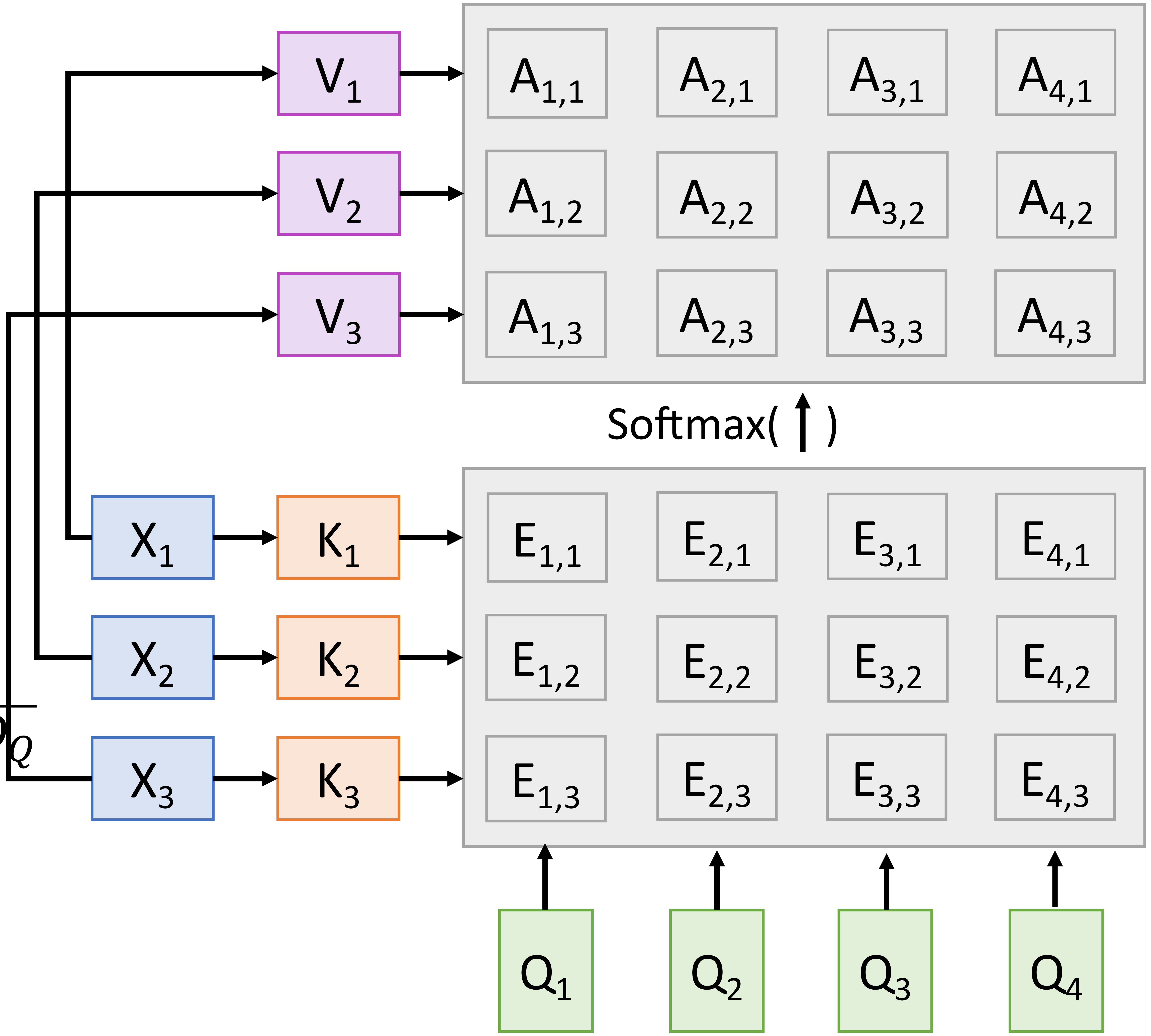


[slide from Justin Johnson, U Michigan]

Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

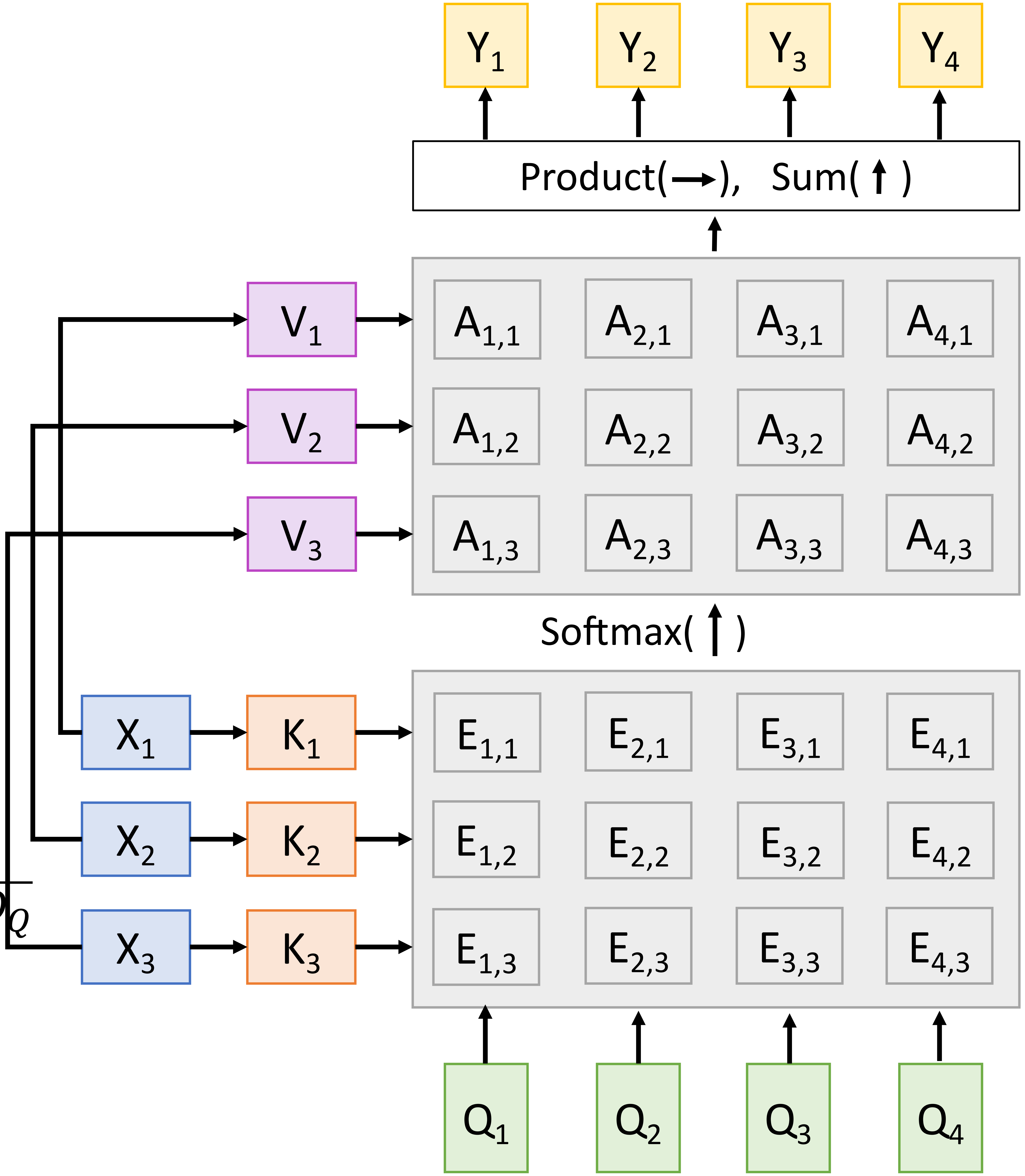


[slide from Justin Johnson, U Michigan]

Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = AV$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$



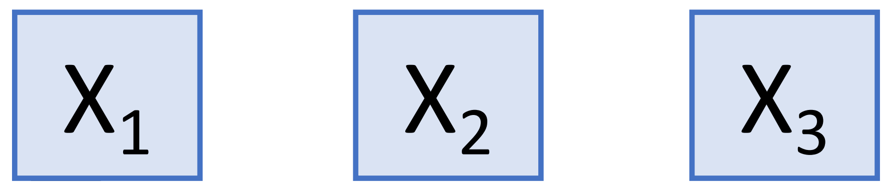
[slide from Justin Johnson, U Michigan]

Self-attention Layer

One **query** per **input vector**

Inputs:
Query vectors: $\mathbf{Q}$ (Shape: N_Q x D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



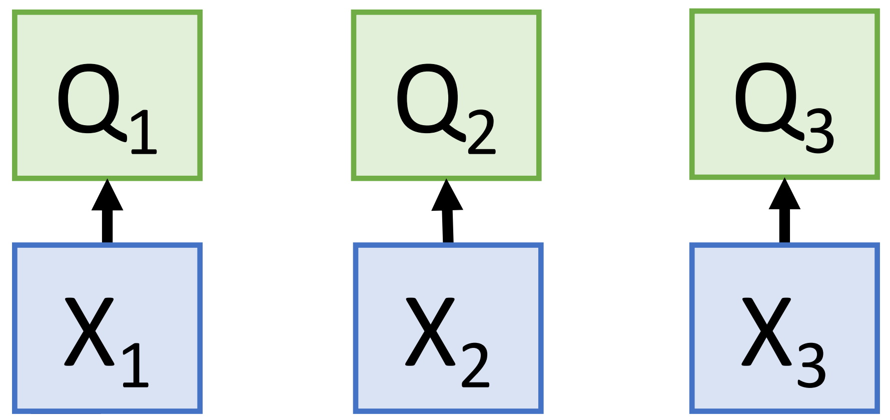
[slide from Justin Johnson, U Michigan]

Self-attention Layer

One **query** per **input vector**

<u>Inputs</u>:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

<u>Computation</u>:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$



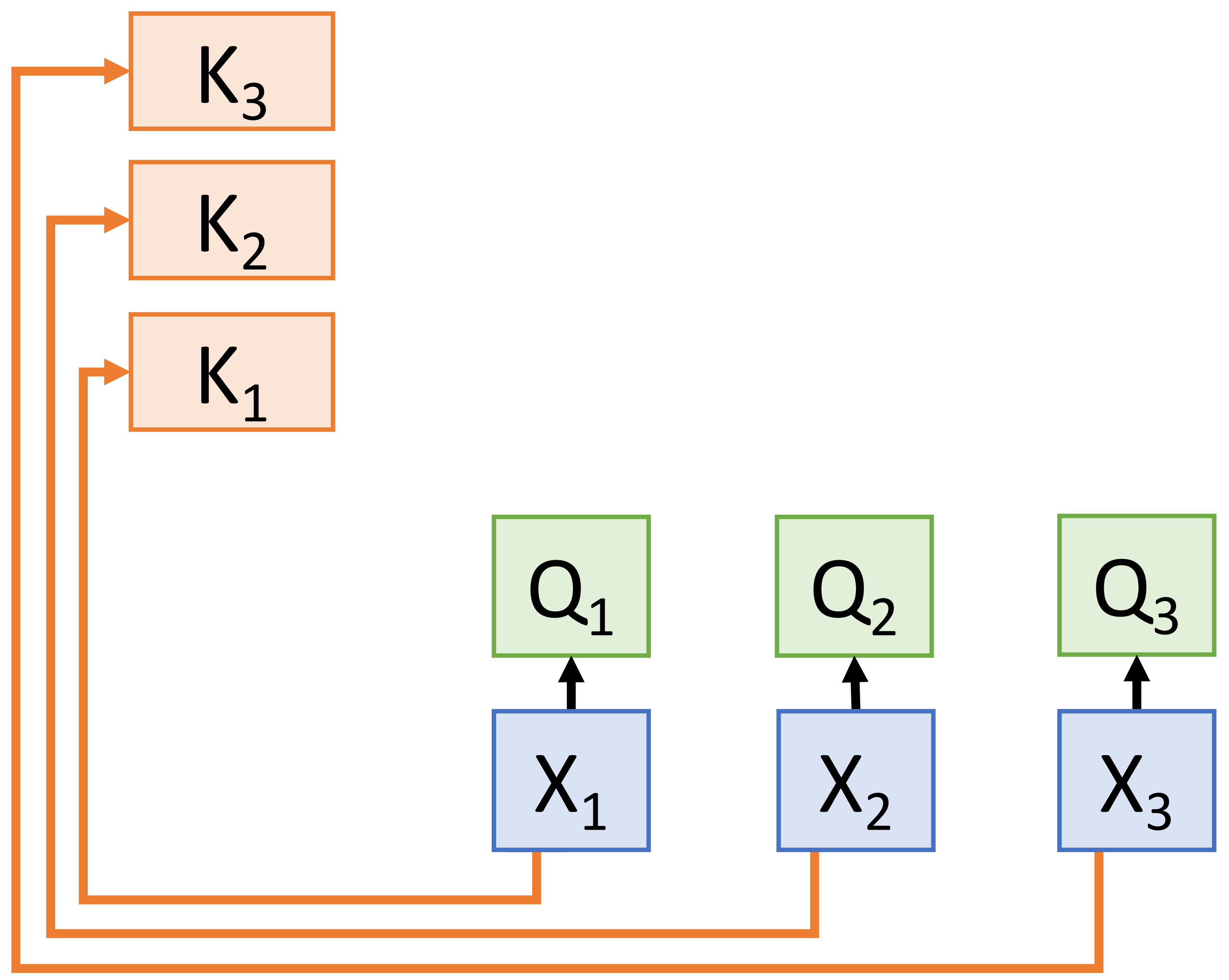
[slide from Justin Johnson, U Michigan]

Self-attention Layer

One **query** per **input vector**

Inputs:
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



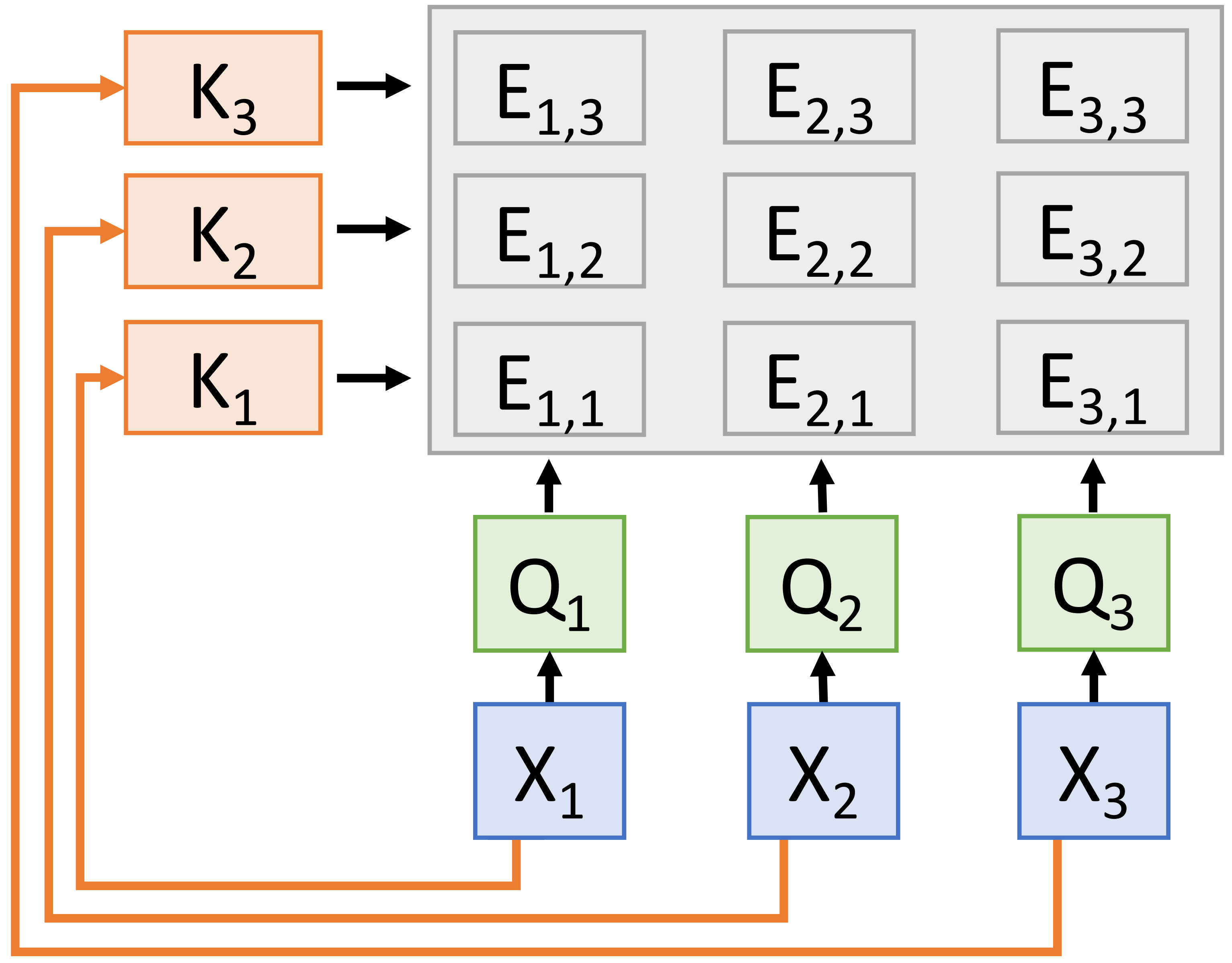
[slide from Justin Johnson, U Michigan]

Self-attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



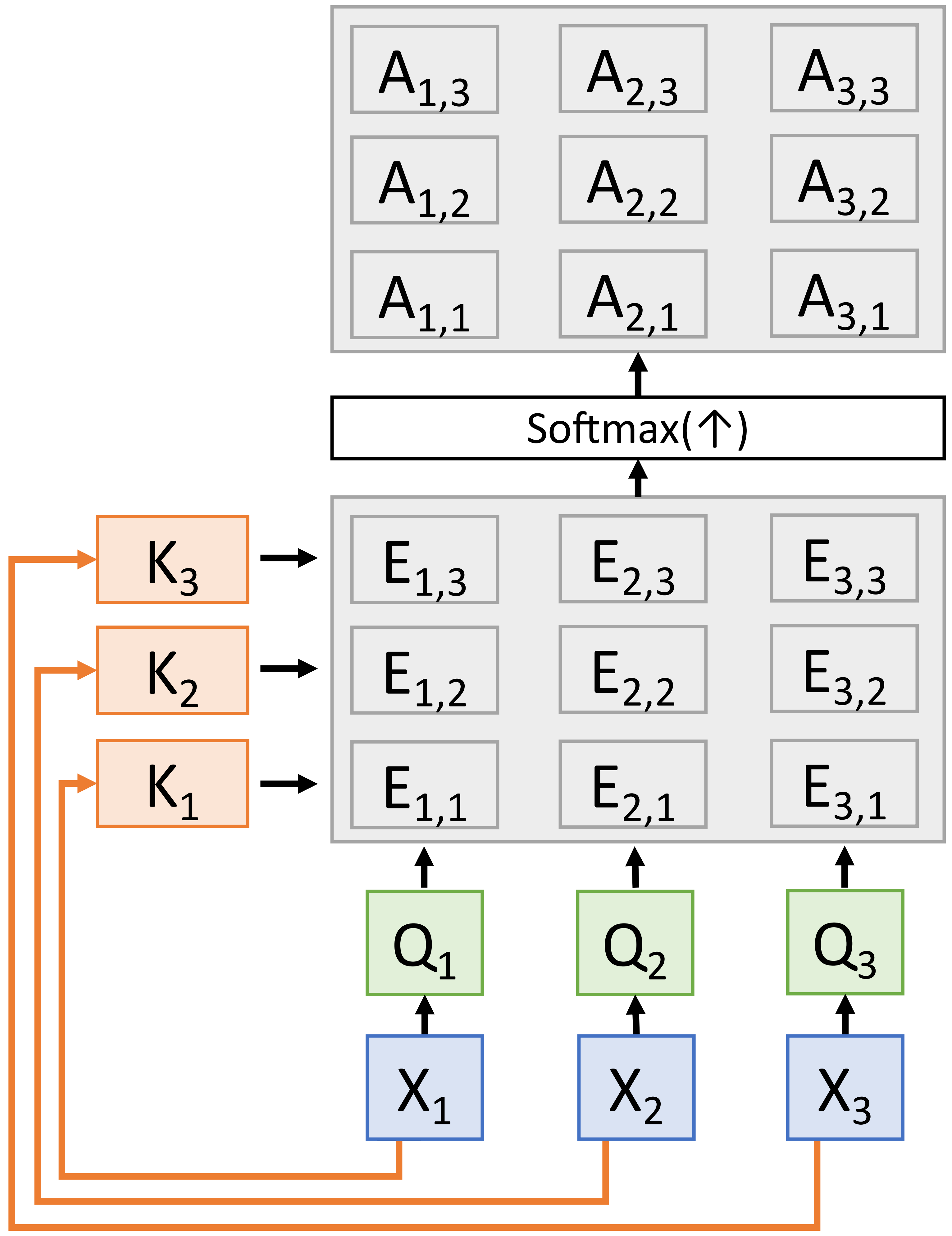
[slide from Justin Johnson, U Michigan]

Self-attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$



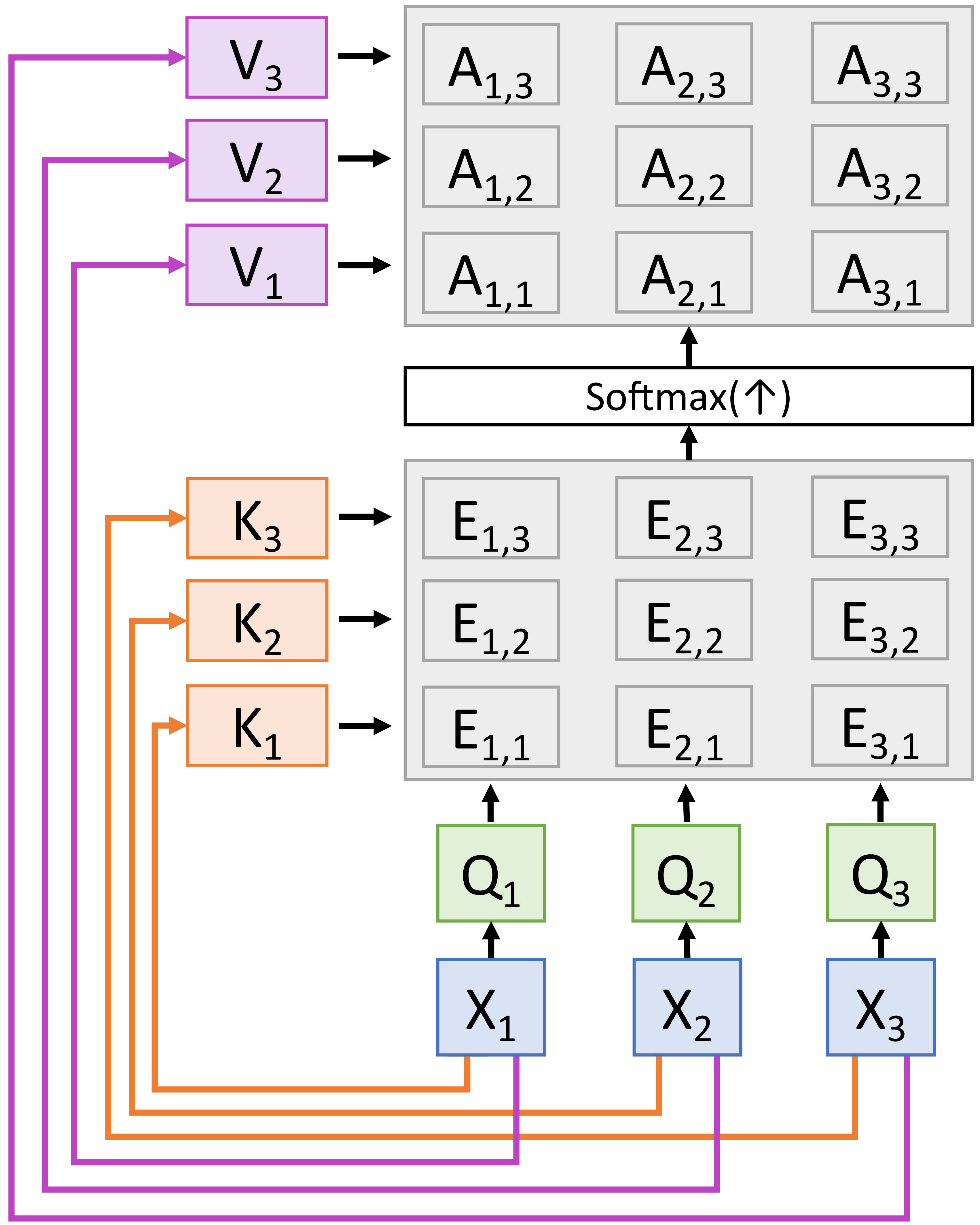
[slide from Justin Johnson, U Michigan]

Self-attention Layer

One query per input vector

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

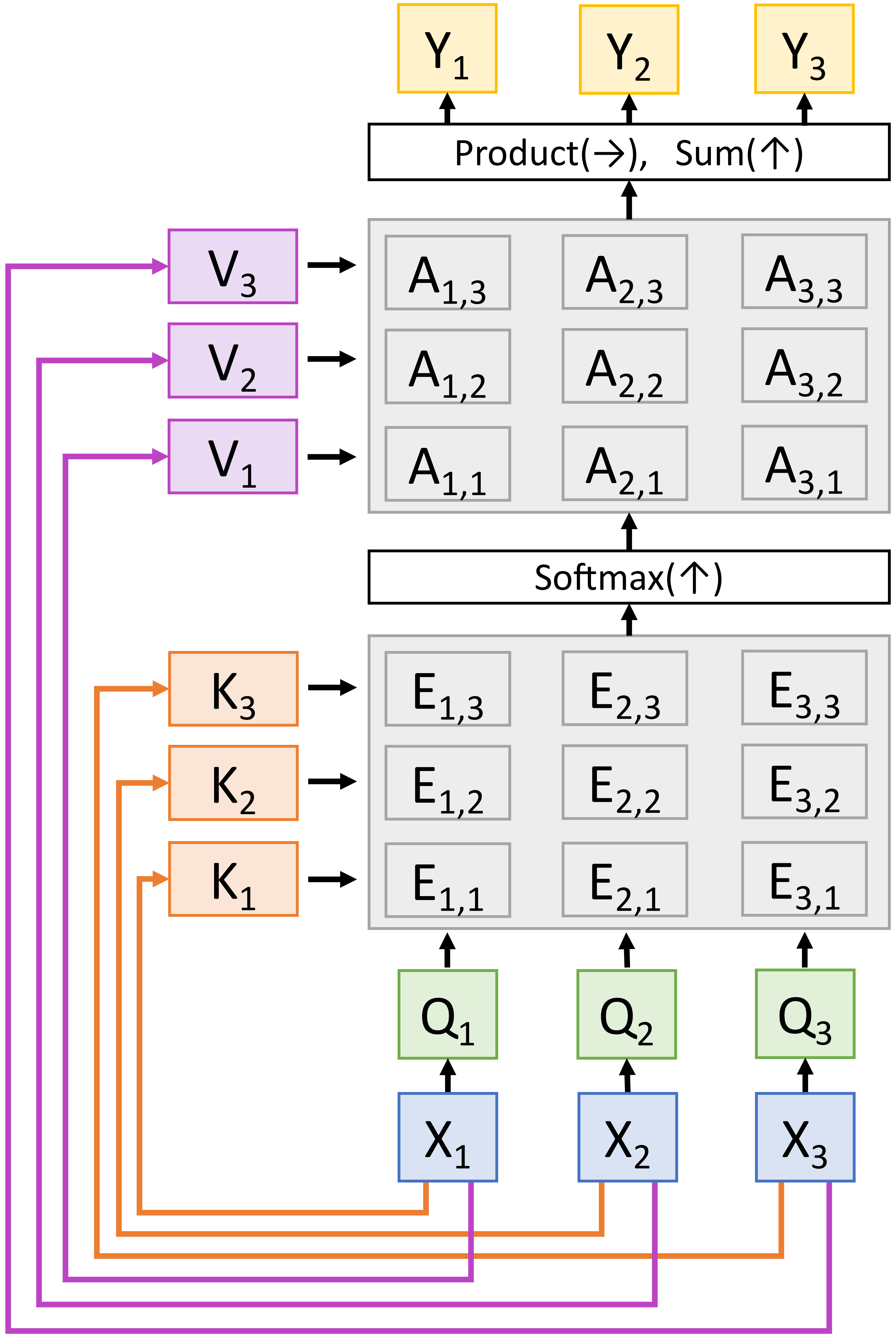


Self-attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



[slide from Justin Johnson, U Michigan]

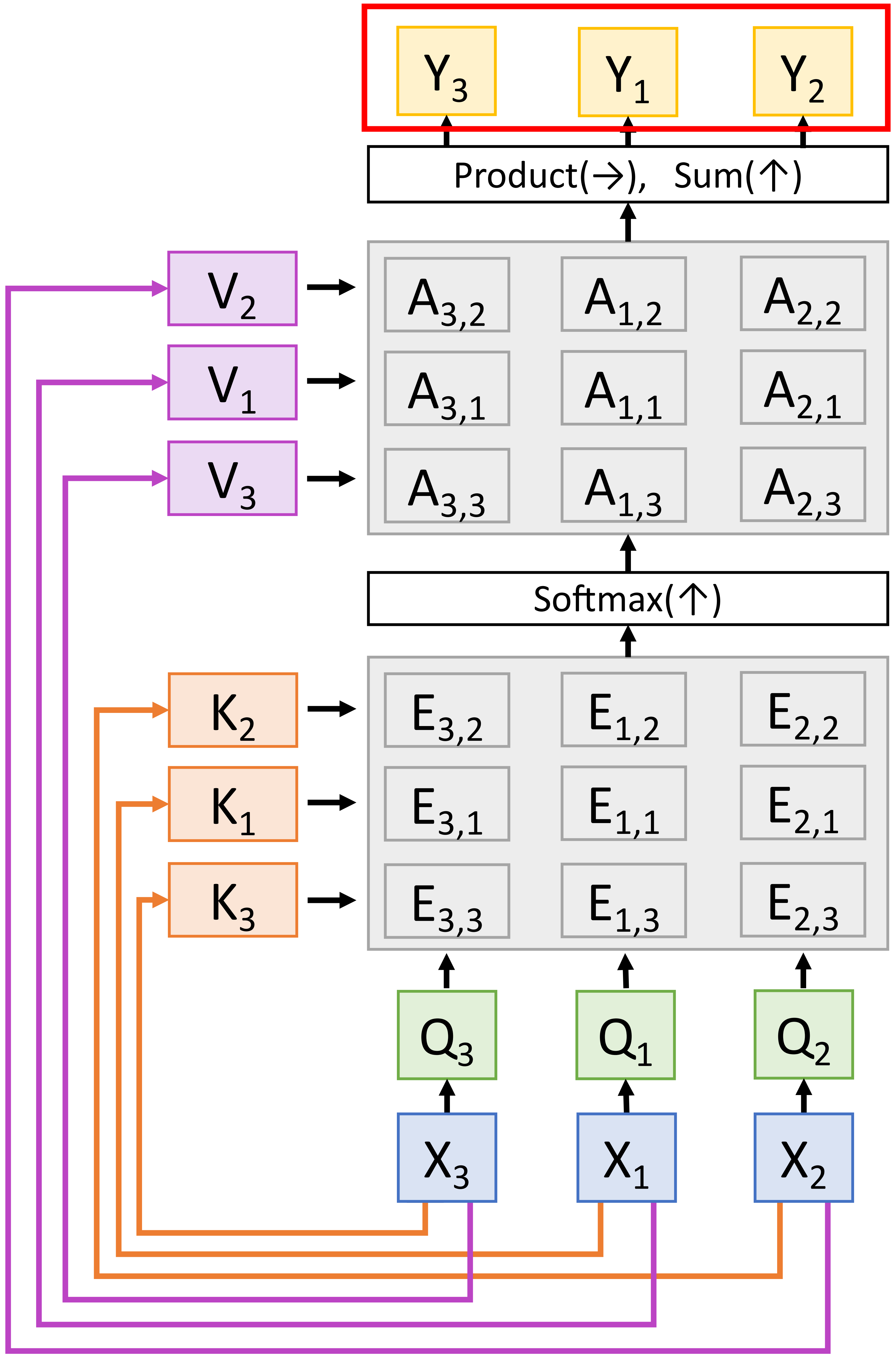
Self-attention Layer

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Self-attention layer is **Permutation Equivariant** $f(s(x)) = s(f(x))$

Self-Attention layer works on **sets** of vectors



<u>Inputs</u>:
Input vectors: **X** (Shape: $N_X \times D_X$)
Key matrix: $\mathbf{W_K}$ (Shape: $D_X \times D_Q$)
Value matrix: $\mathbf{W_V}$ (Shape: $D_X \times D_V$)
Query matrix: $\mathbf{W_Q}$ (Shape: $D_X \times D_Q$)

<u>Computation</u>:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: $N_X \times D_Q$)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: $N_X \times D_V$)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

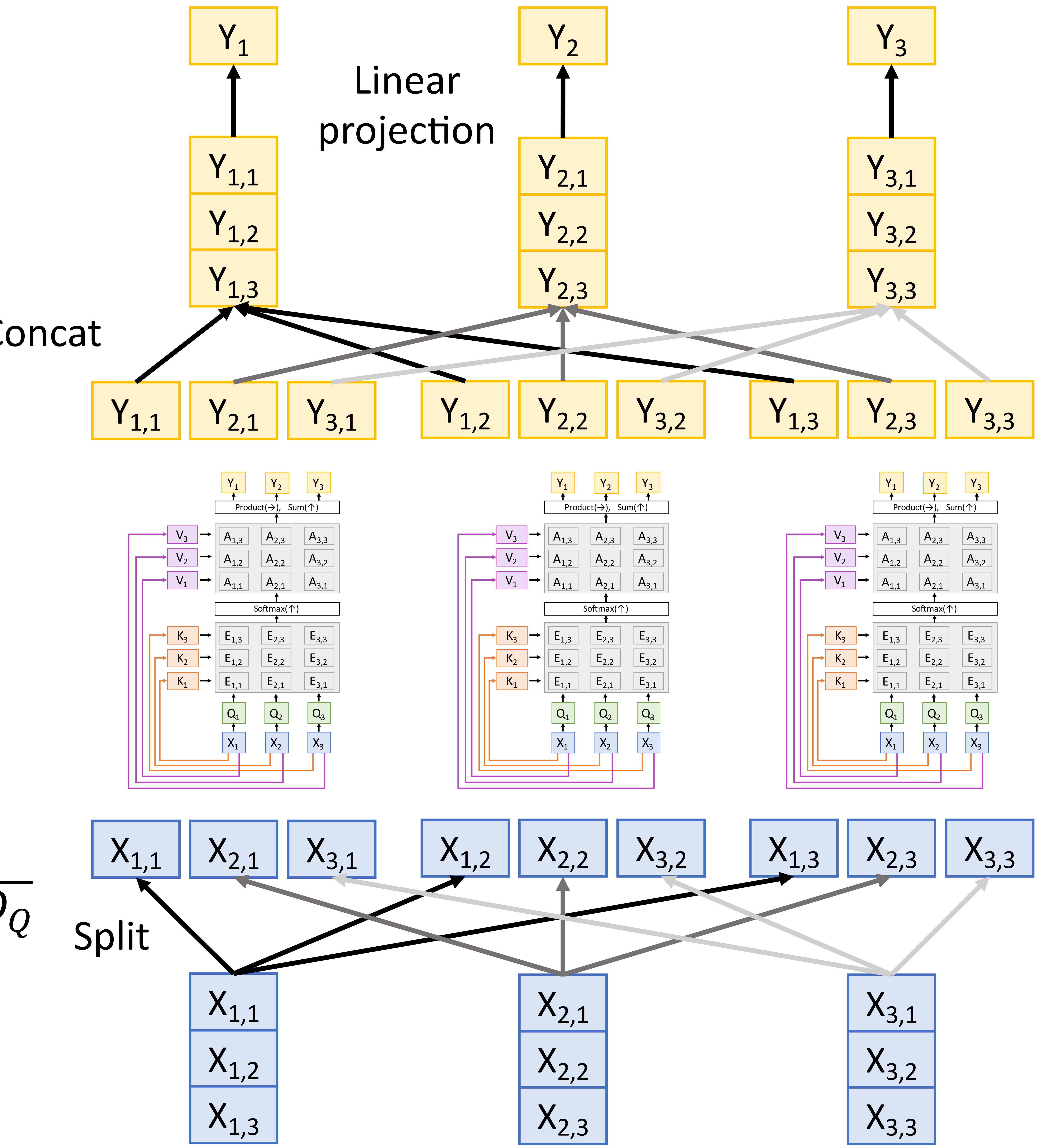
[slide from Justin Johnson, U Michigan]

Multi-head Self-attention Layer

<u>Inputs</u>:
Input vectors: $\mathbf{X}$ (Shape: $N_X \times D_X$)
Key matrix: $\mathbf{W_K}$ (Shape: $D_X \times D_Q$)
Value matrix: $\mathbf{W_V}$ (Shape: $D_X \times D_V$)
Query matrix: $\mathbf{W_Q}$ (Shape: $D_X \times D_Q$)

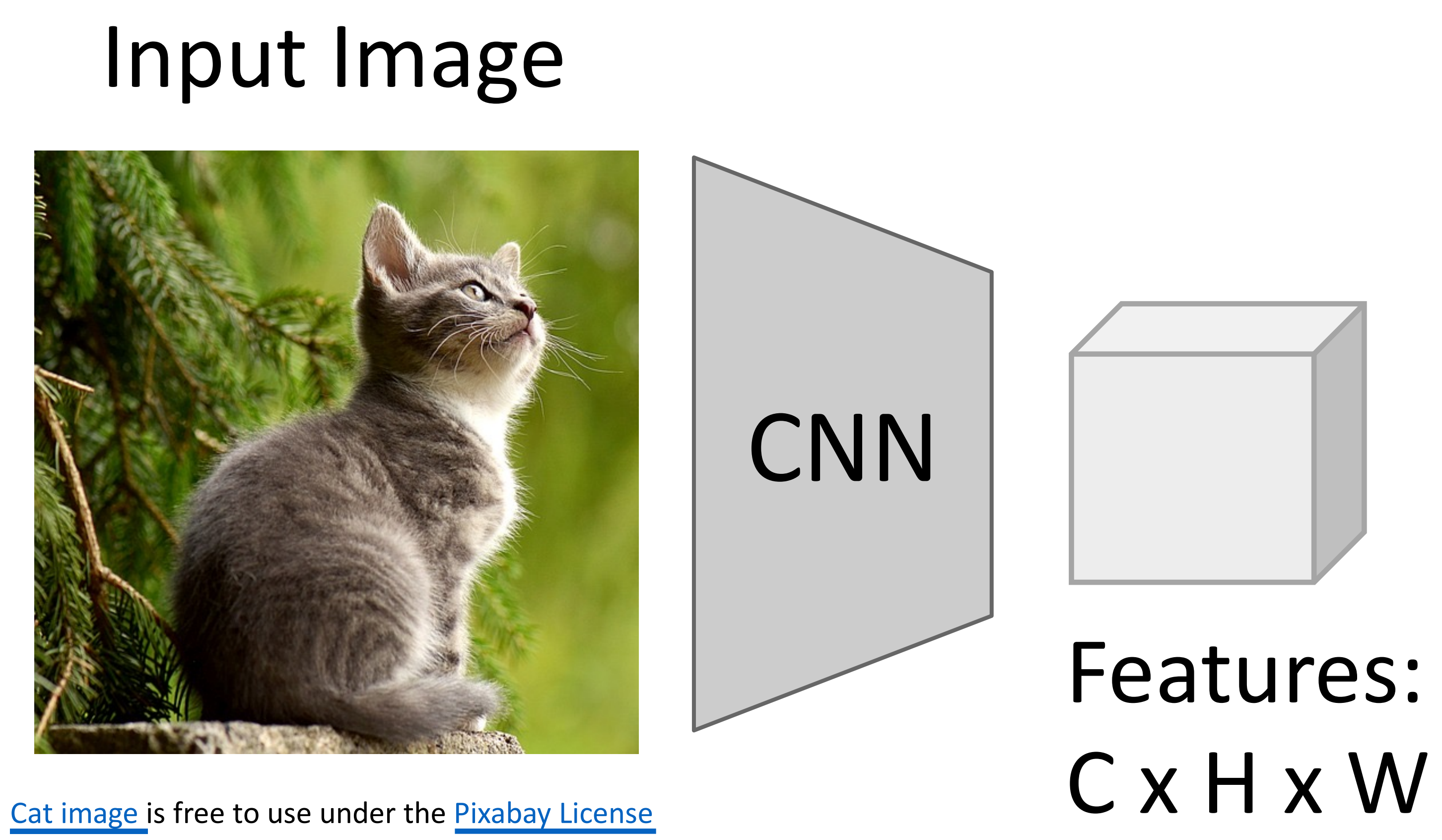
<u>Computation</u>:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: $N_X \times D_Q$)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: $N_X \times D_V$)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

Use H independent
"Attention Heads" in
parallel



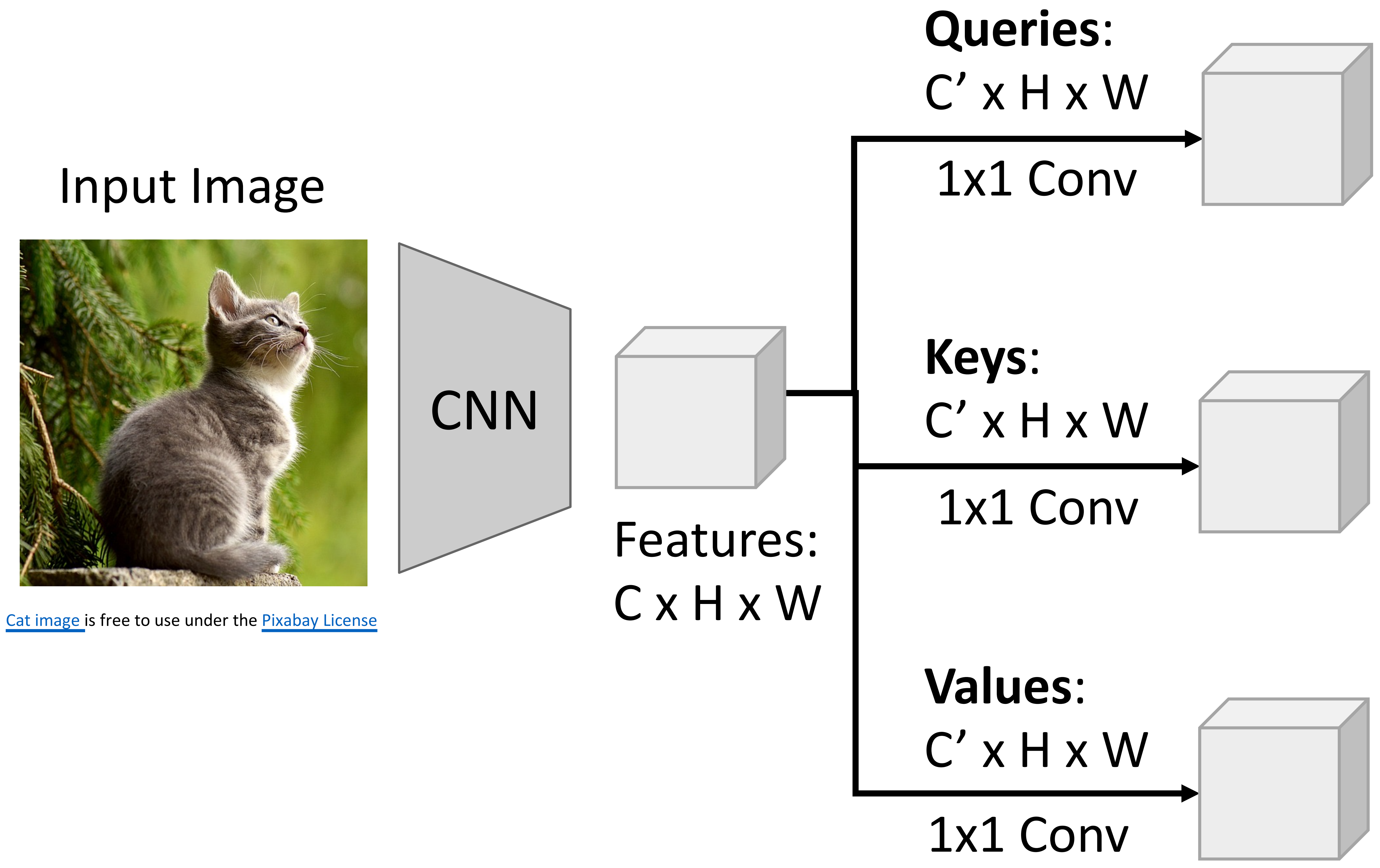
[slide from Justin Johnson, U Michigan]

CNN with **Self-attention**



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

CNN with Self-attention

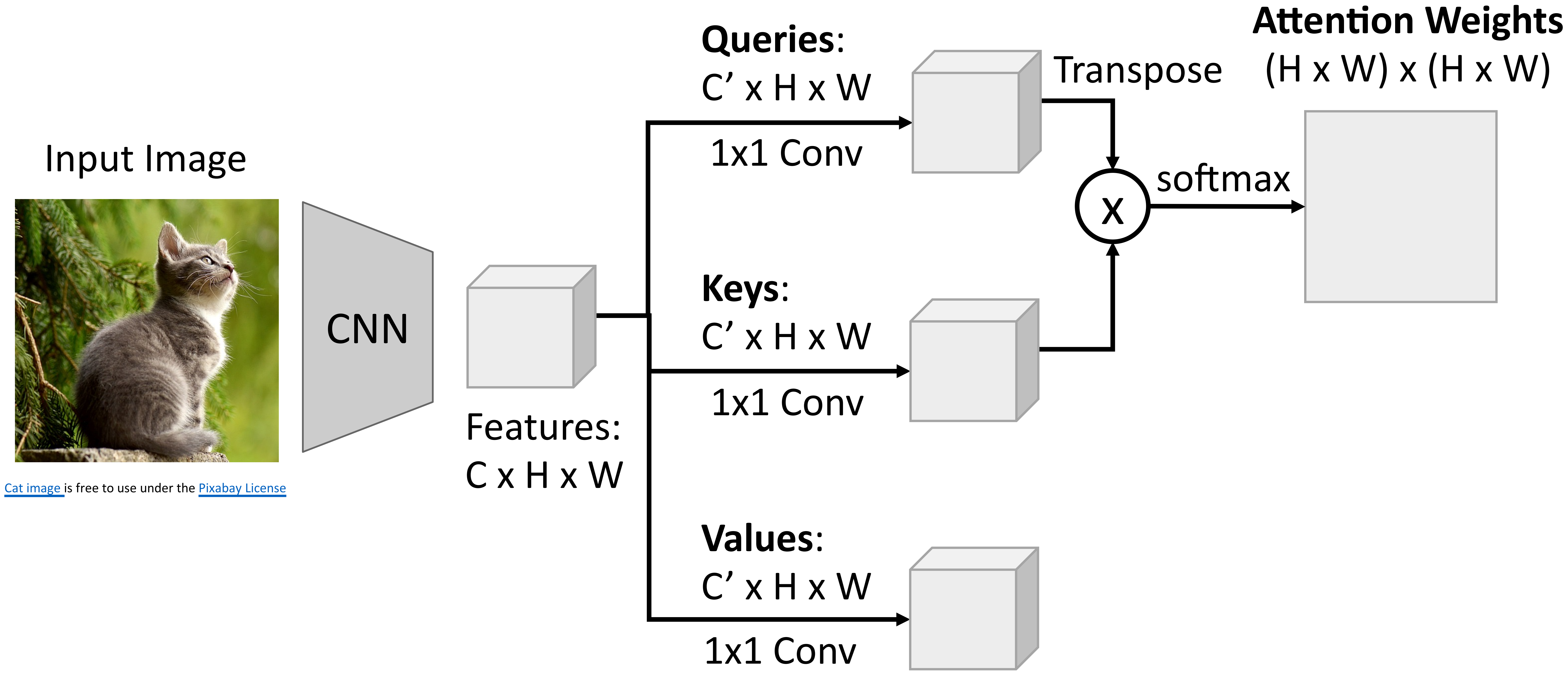


Cat image is free to use under the Pixabay License

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

[slide from Justin Johnson, U Michigan]

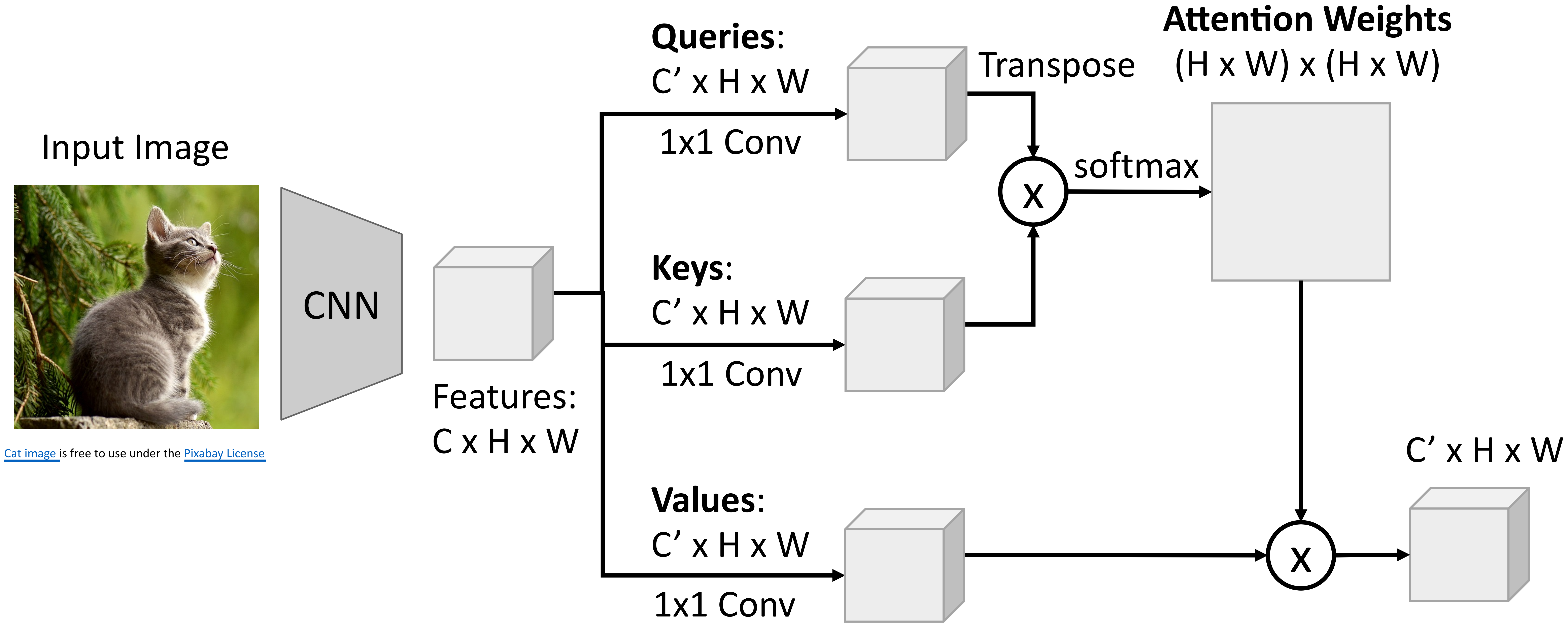
CNN with Self-attention



Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

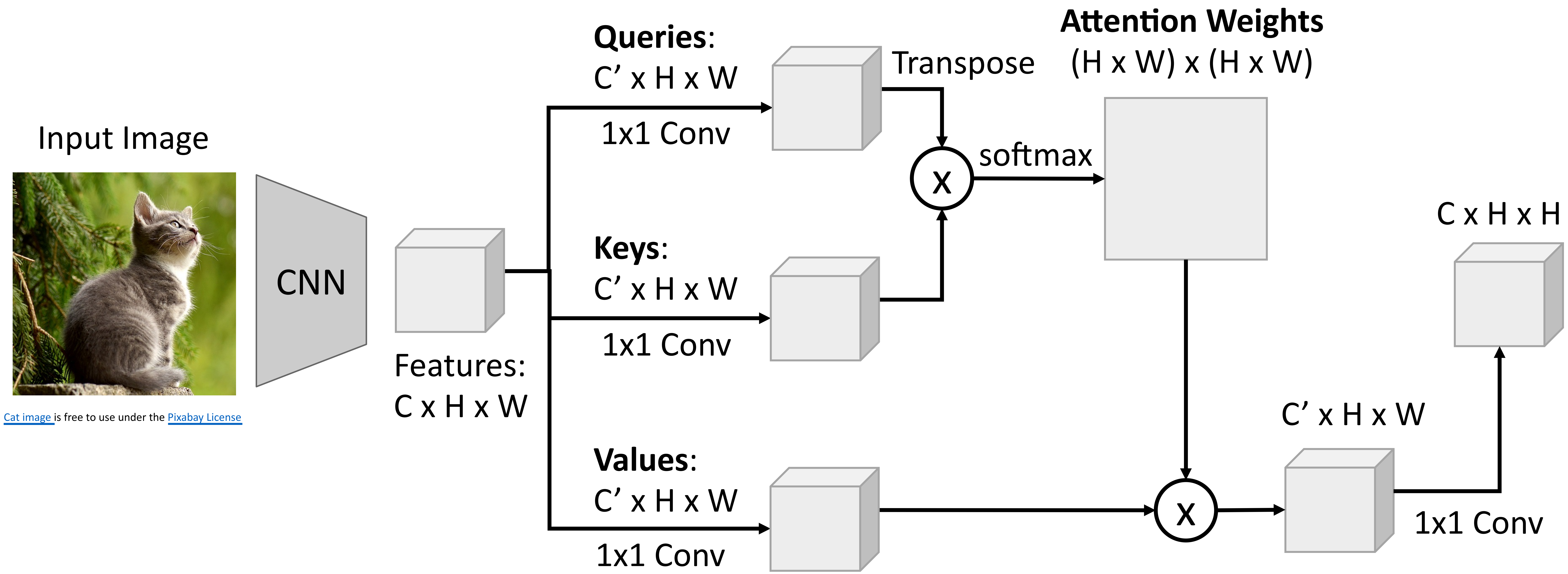
[slide from Justin Johnson, U Michigan]

CNN with Self-attention



Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

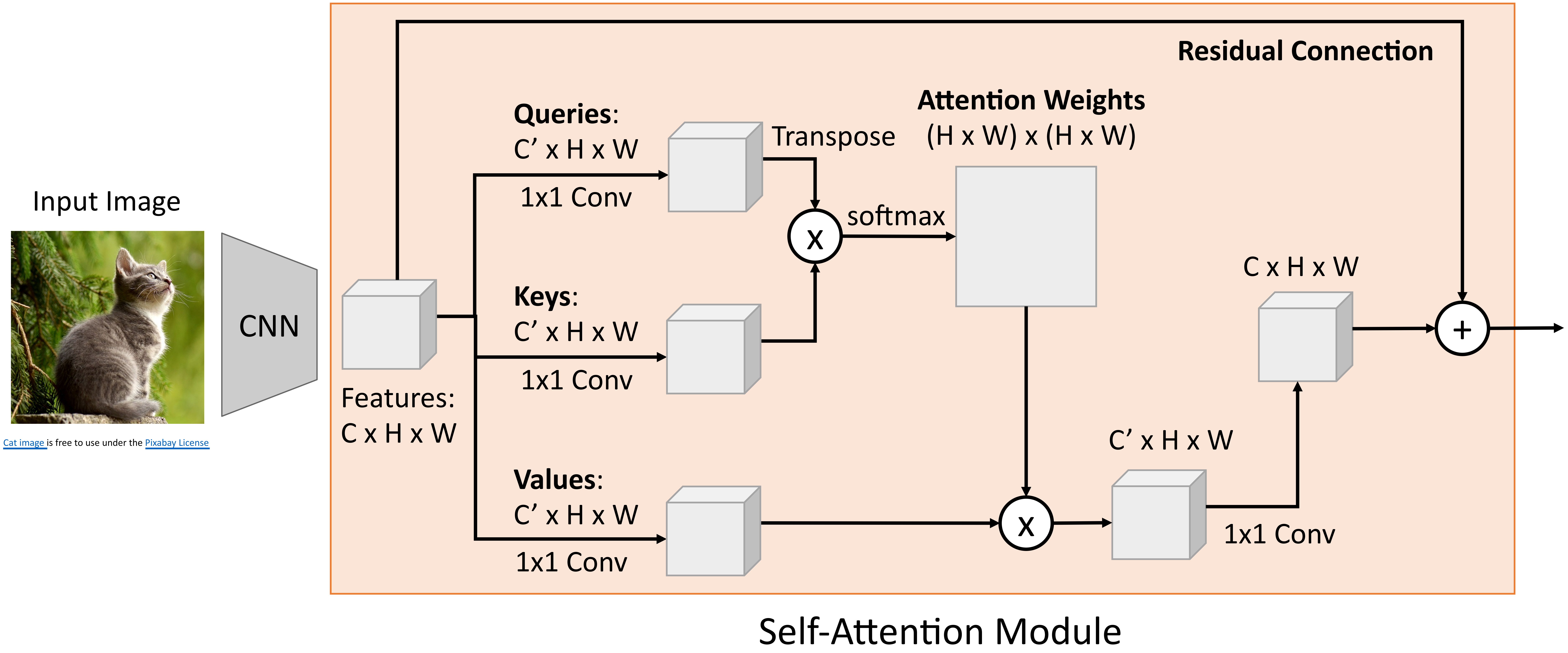
CNN with Self-attention



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

[slide from Justin Johnson, U Michigan]

CNN with Self-attention

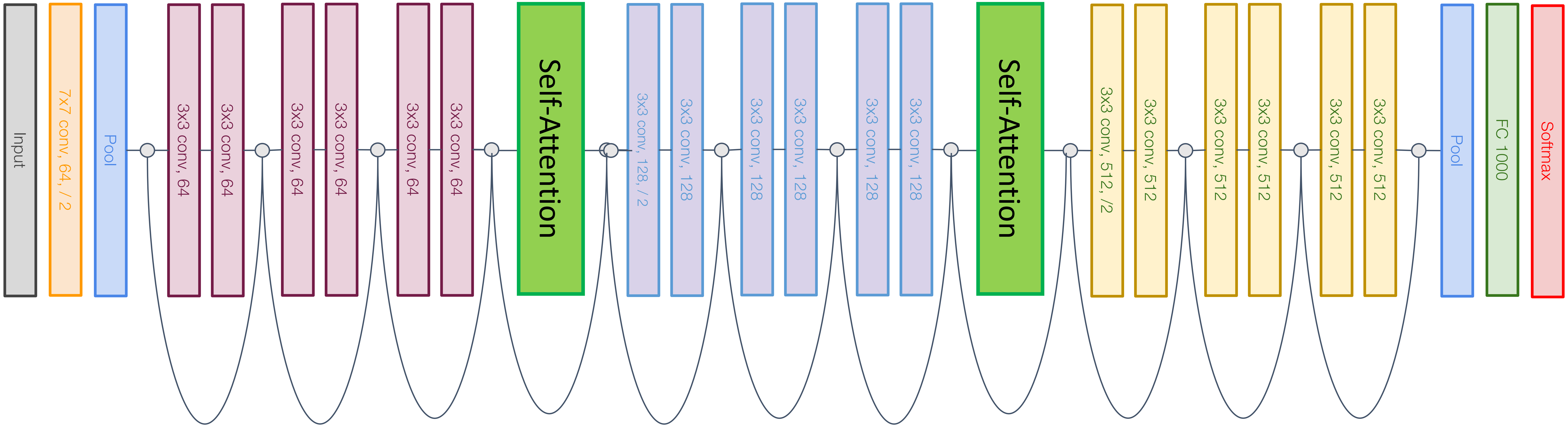


Cat image is free to use under the Pixabay License

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

[slide from Justin Johnson, U Michigan]

Attention with Existing CNNs



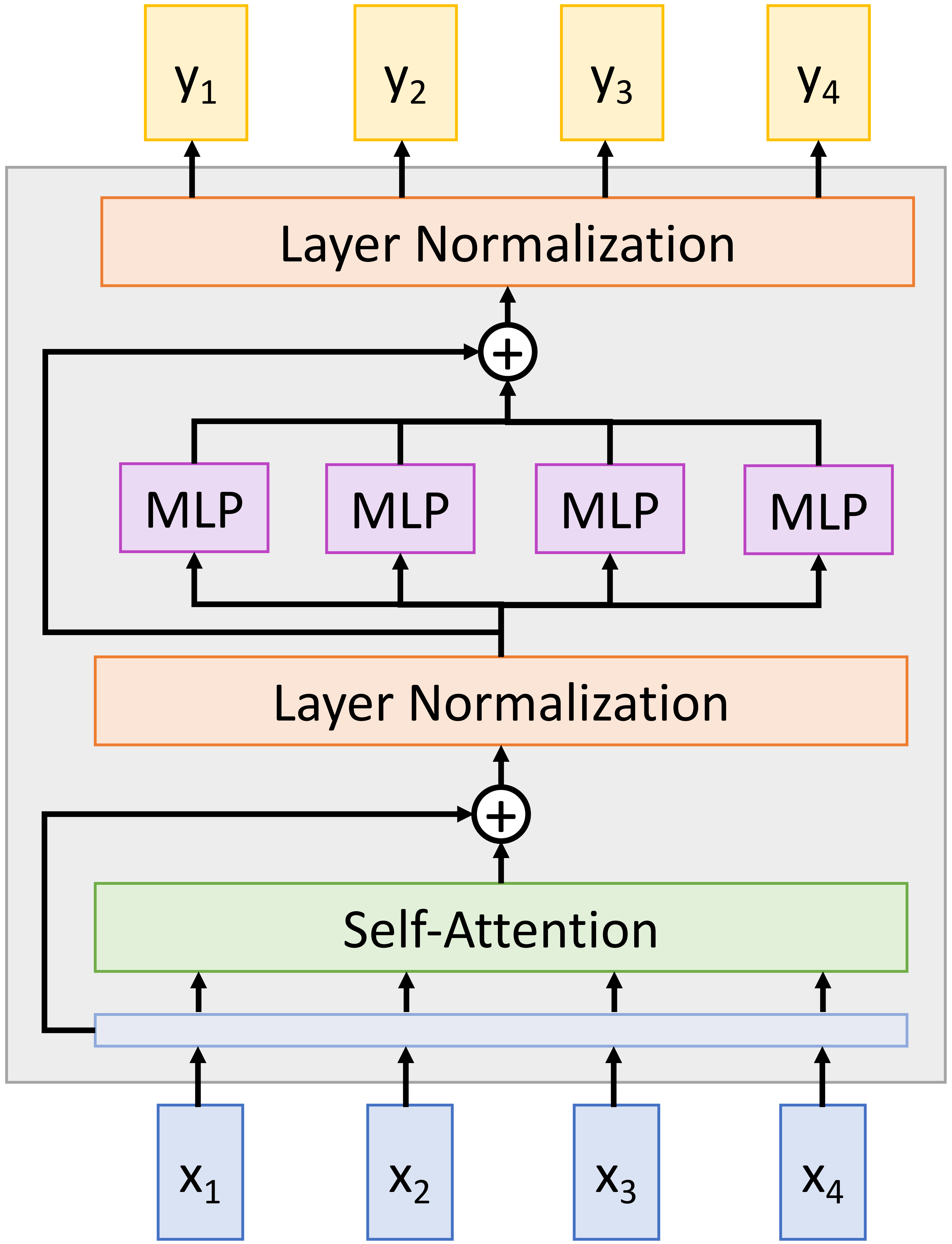
Zhang et al, ”Self-Attention Generative Adversarial Networks”, ICML 2018
Wang et al, ”Non-local Neural Networks”, CVPR 2018

[slide from Justin Johnson, U Michigan]

Transformer

Transfomer block inputs a set of vectors, outputs a set of vectors.

Vectors only communicate via (multiheaded) self-attention



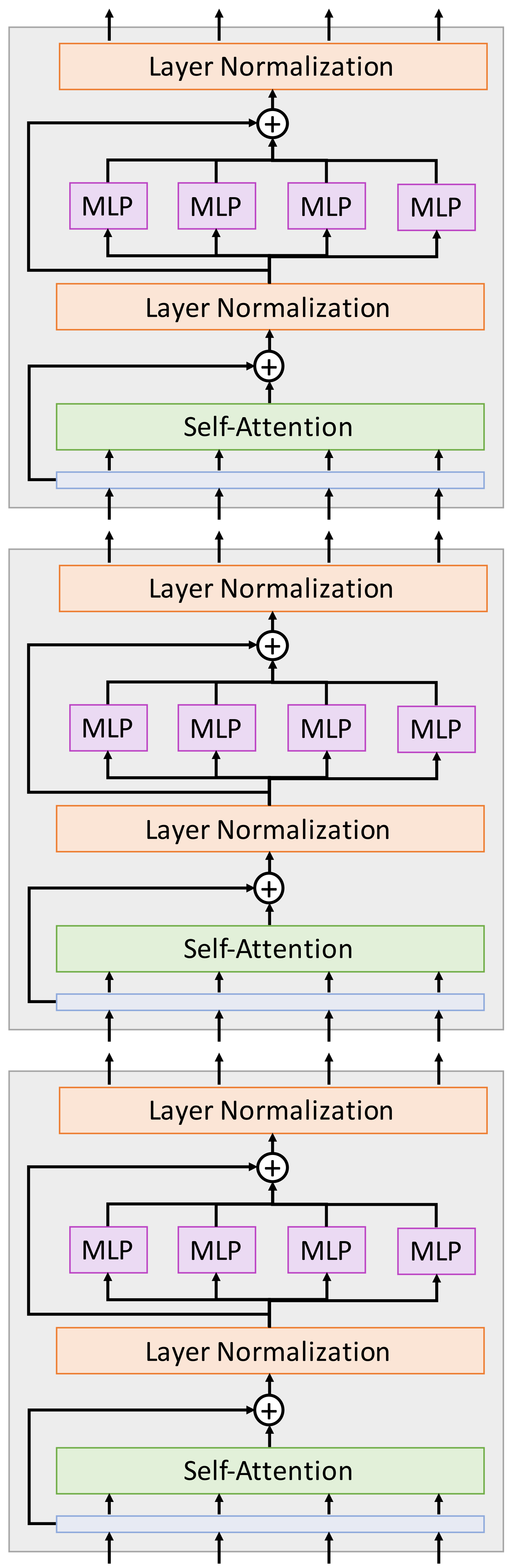
Vaswani et al, "Attention is all you need", NeurIPS 2017

[slide from Justin Johnson, U Michigan]

Transformer

<u>**Transformer Block:**</u>
Input: Set of vectors x
Output: Set of vectors y

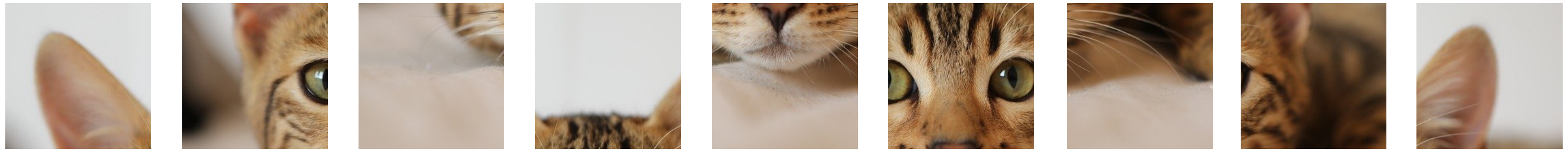
Hyperparameters:
- Number of blocks
- Number of heads per block
- Width (channels per head, FFN width)



Vaswani et al, "Attention is all you need", NeurIPS 2017

Transformer on Image Patches

N input patches, each of shape 3x16x16



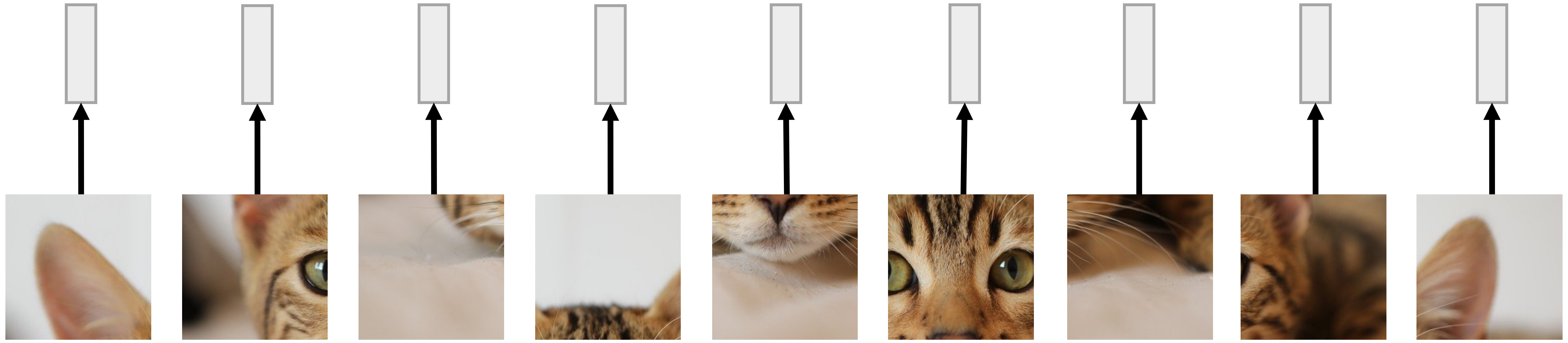
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

[slide from Justin Johnson, U Michigan]

Transformer on Image Patches

Linear projection to D-dimensional vector

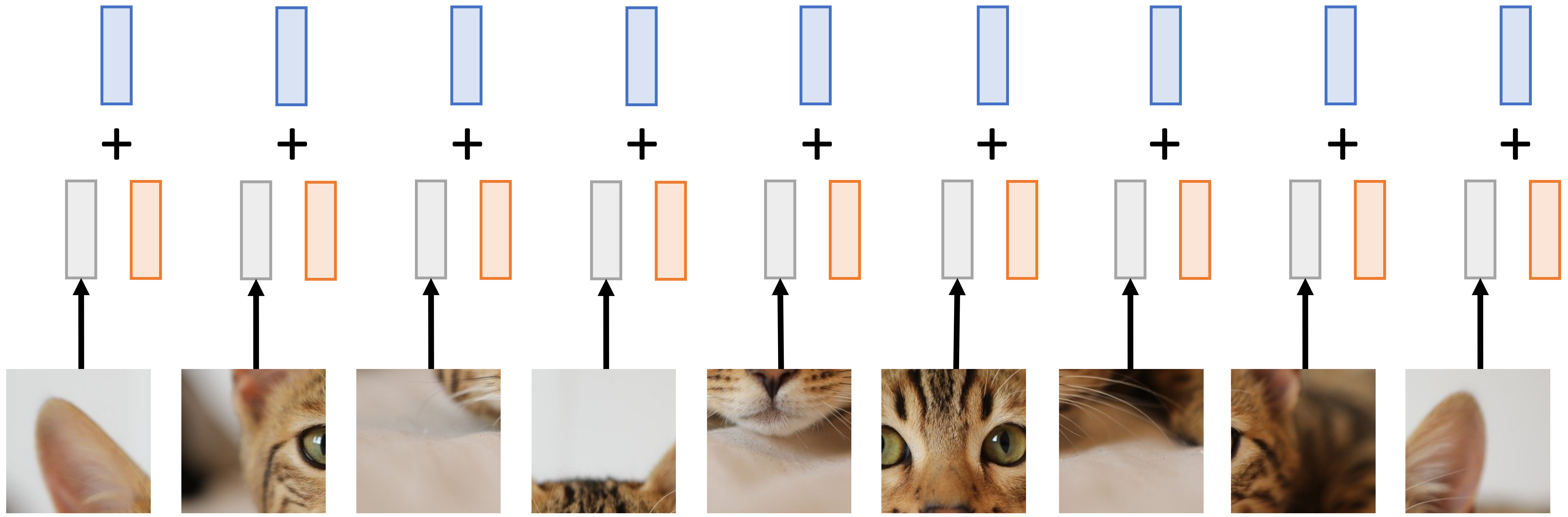
N input patches, each of shape 3x16x16



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[slide from Justin Johnson, U Michigan]

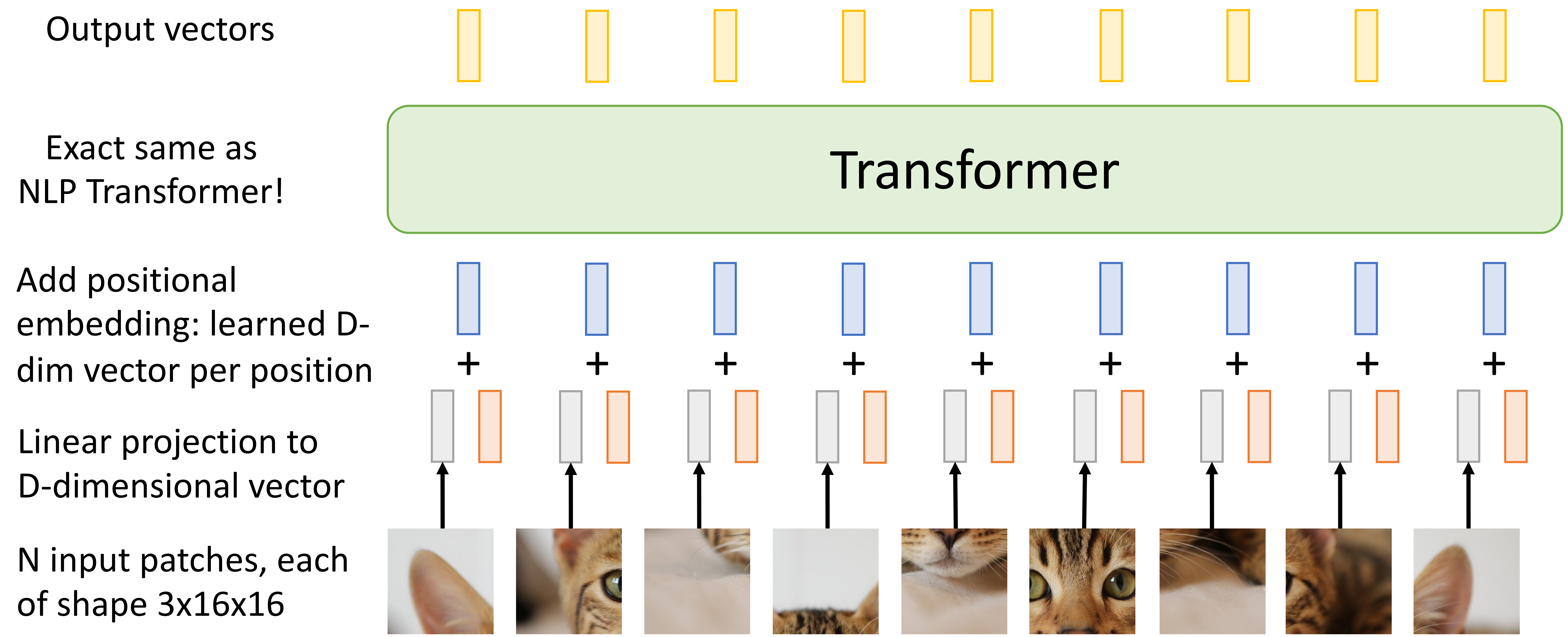
Transformer on Image Patches



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[slide from Justin Johnson, U Michigan]

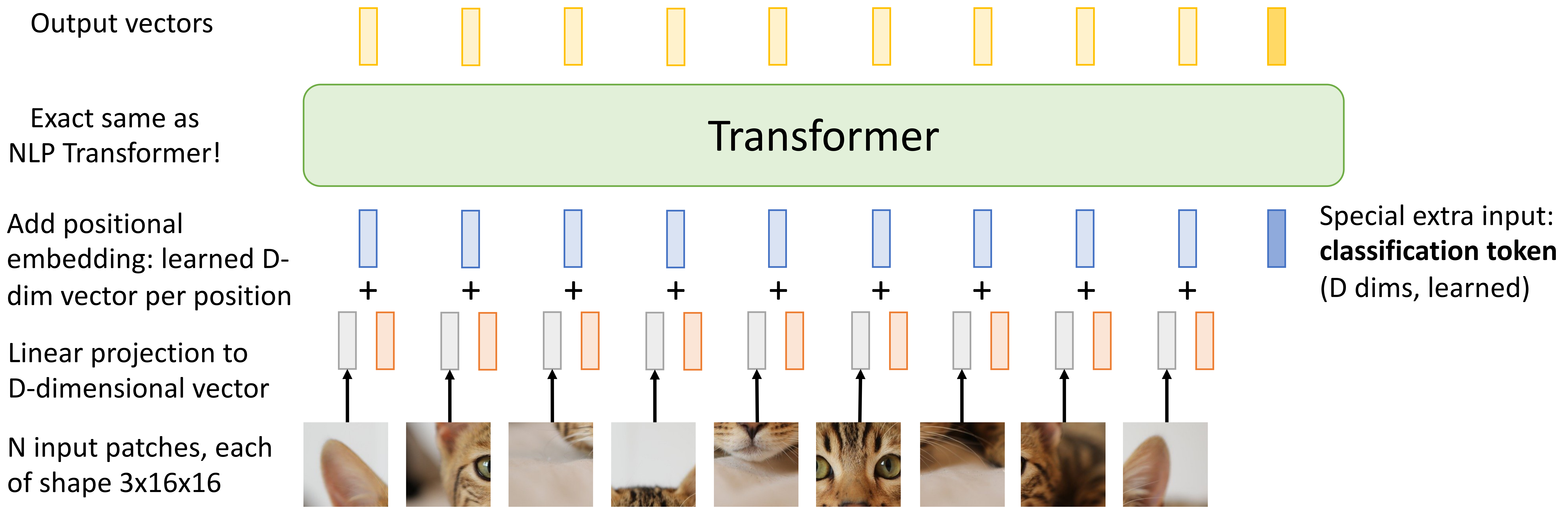
Transformer on Image Patches



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[slide from Justin Johnson, U Michigan]

Transformer on Image Patches



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[slide from Justin Johnson, U Michigan]

Transformer on Image Patches

Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

[slide from Justin Johnson, U Michigan]

Vision Transformer (ViT)

Computer vision model
with no convolutions!

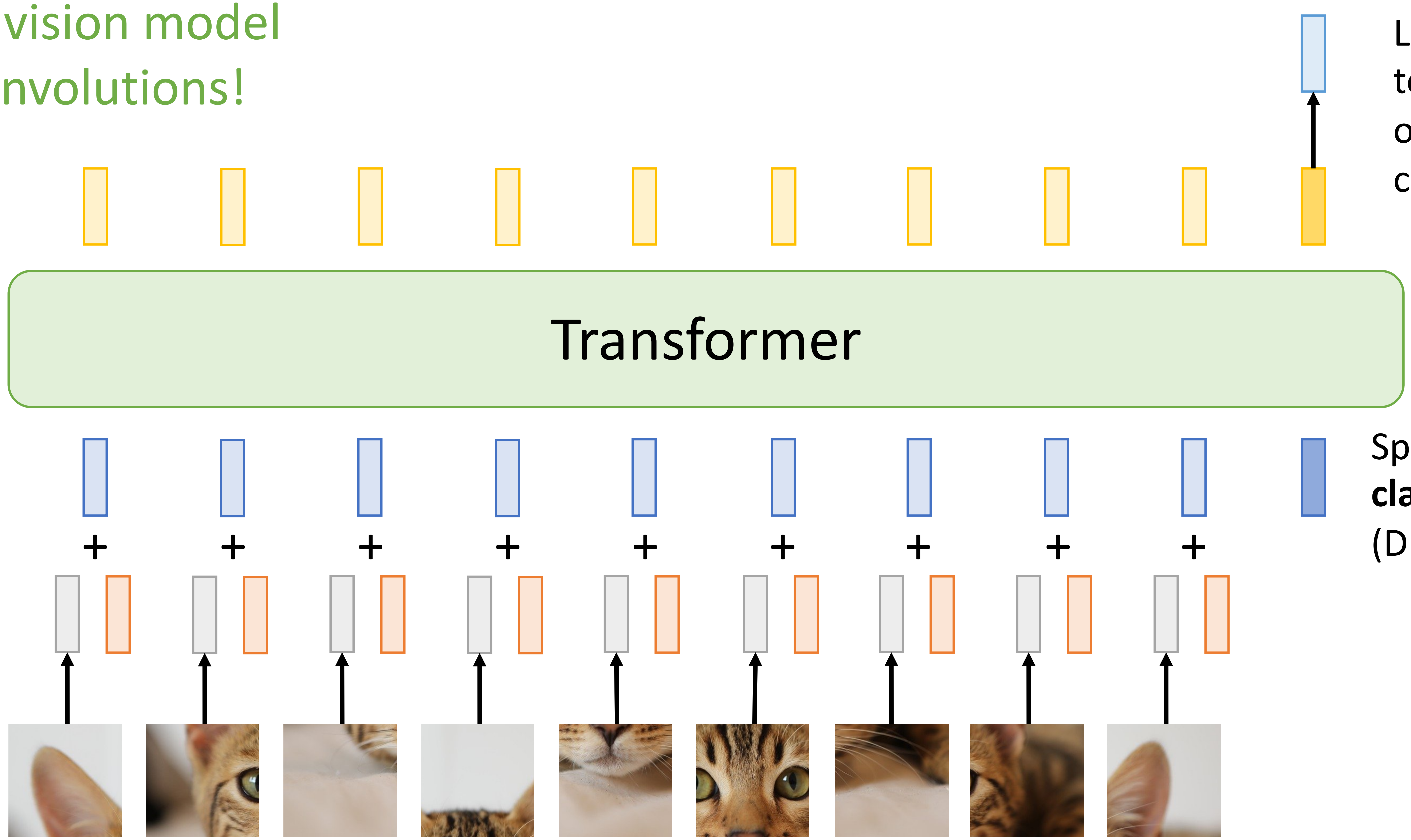
Output vectors

Exact same as
NLP Transformer!

Add positional
embedding: learned D-
dim vector per position

Linear projection to
D-dimensional vector

N input patches, each
of shape 3x16x16



Linear projection
to C-dim vector
of predicted
class scores

Special extra input:
classification token
(D dims, learned)

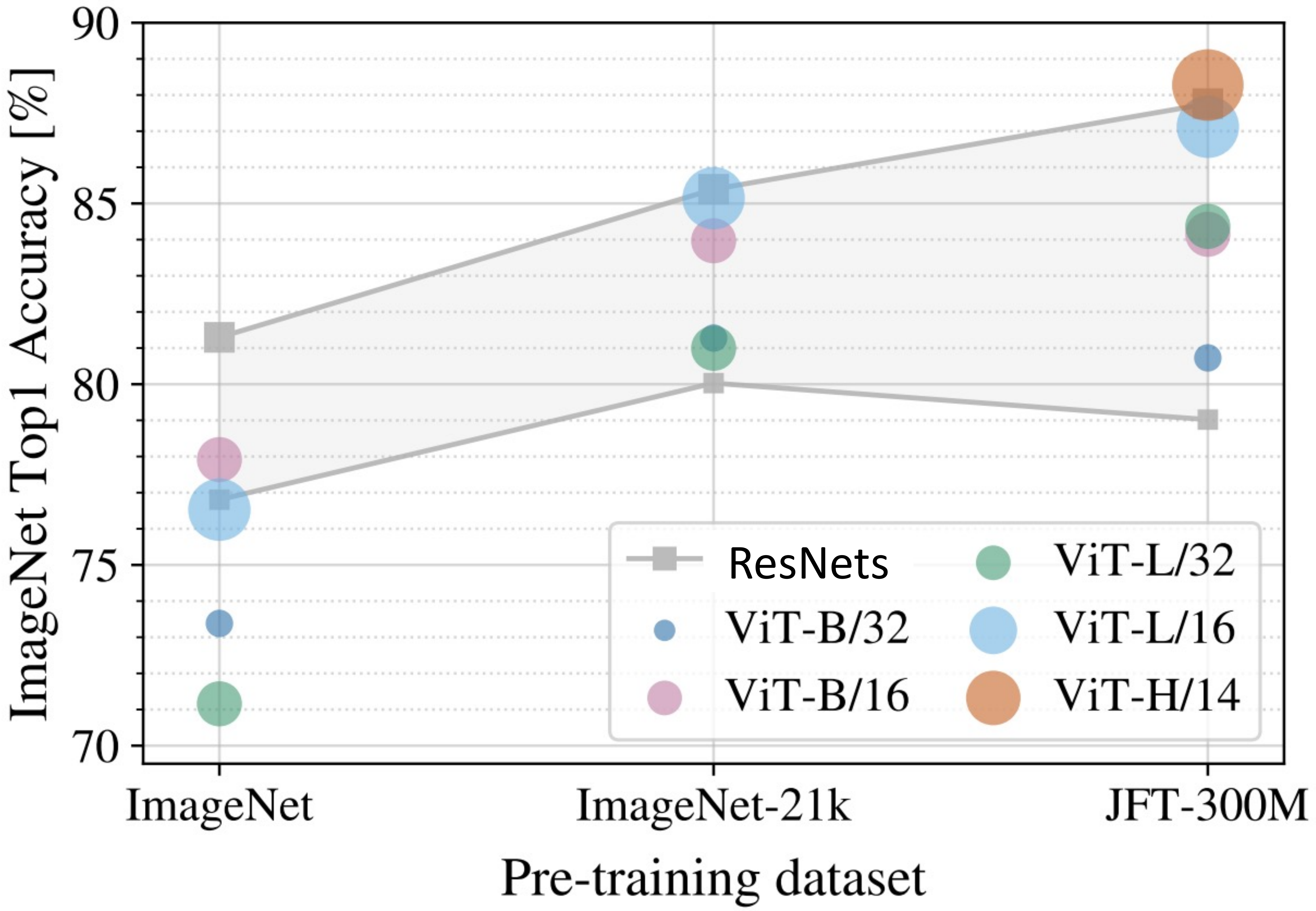
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[slide from Justin Johnson, U Michigan]

Vision Transformer (ViT) vs. **ResNet**

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



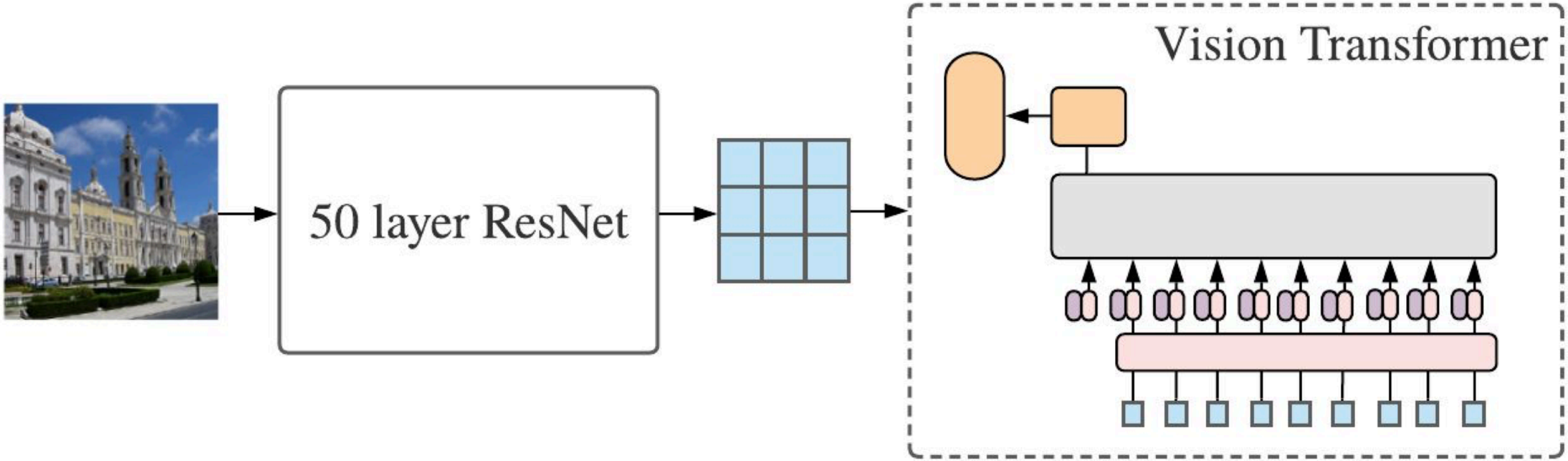
B = Base
L = Large
H = Huge

/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

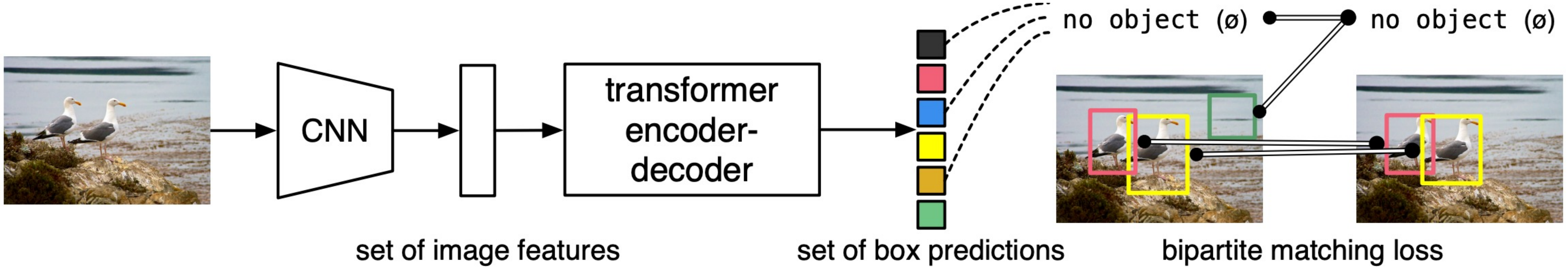
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

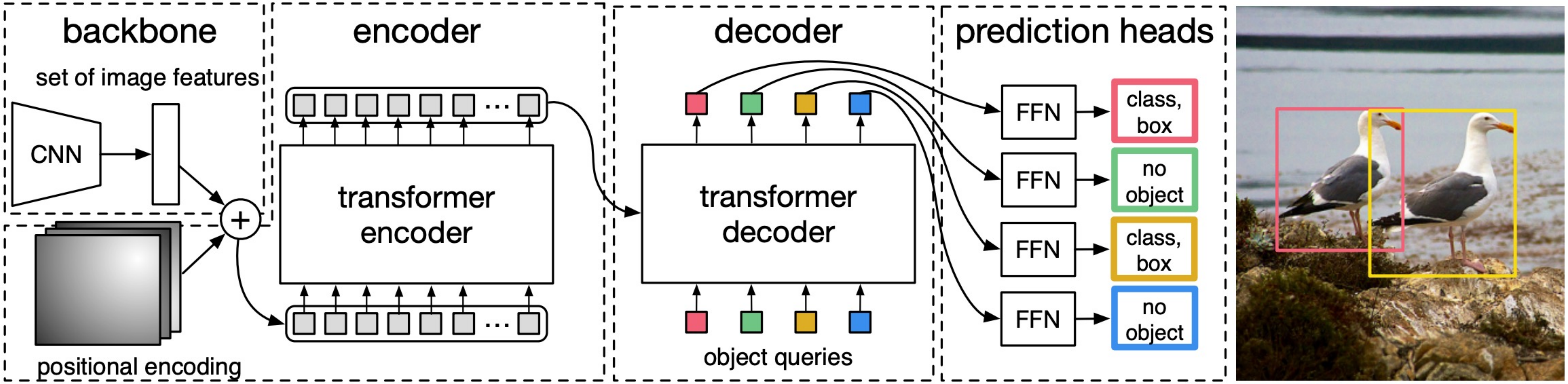
[slide from Justin Johnson, U Michigan]

ResNet-ViT Hybrid



Object Detection with **Transformers**: DETR Model





Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

[slide from Justin Johnson, U Michigan]

Masked Modeling with Transformers (BERT, GPT, etc.)

Training: Predict Masked Tokens

$$\mathcal{L}_{\text{MLM}}(X;\theta) = \mathop{\mathbb{E}}_{x\sim X}\ \mathop{\mathbb{E}}_{\text{mask}} \sum_{i\in\text{mask}} \log p(x_i | x_{j\notin\text{mask}};\theta)$$

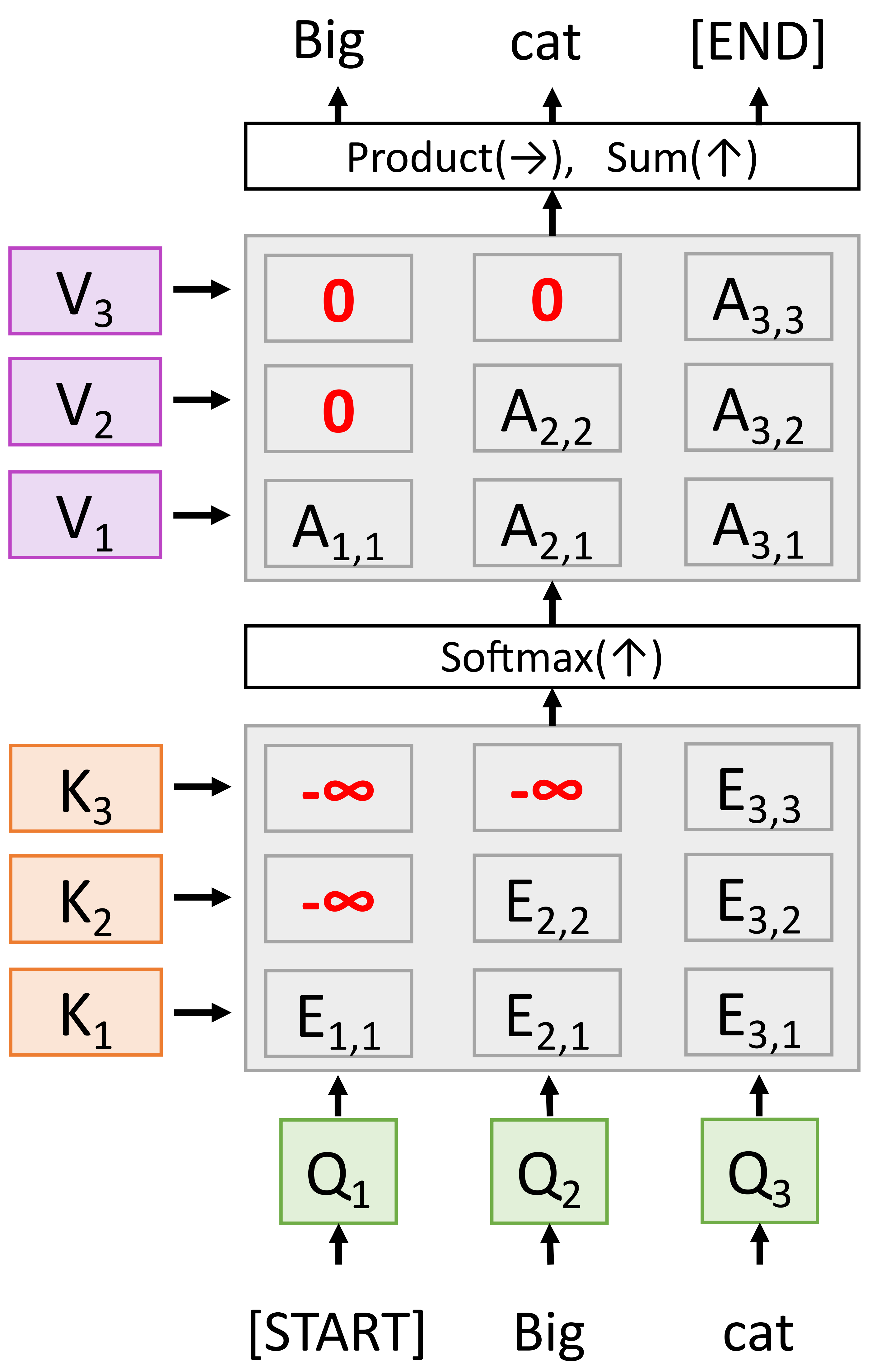
(mask 15% at a time)

Masked Self-Attention Layer

Don't let vectors "look ahead" in the sequence
Used for language modeling (predict next word)

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Please fill out

Student Evaluations

(on Canvas)