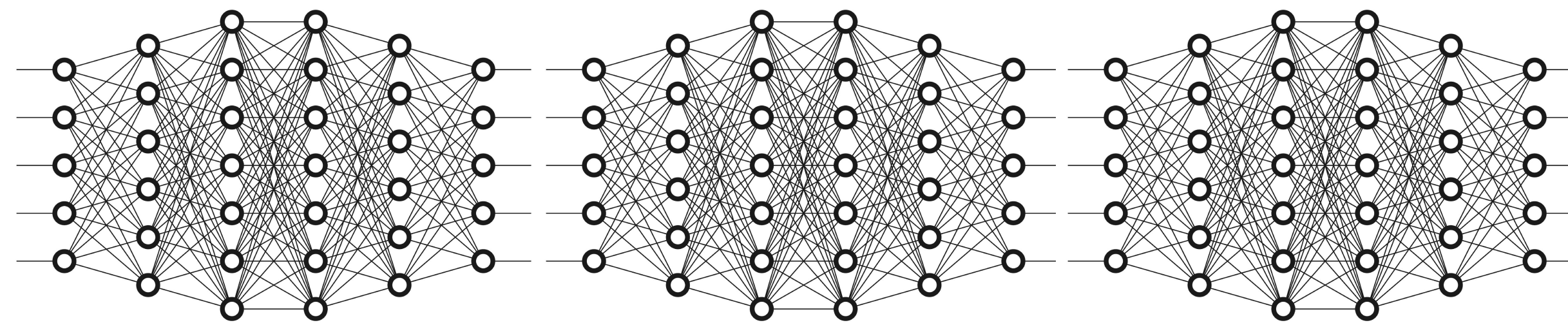




THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 23: Detection, Segmentation

Menu for Today

Topics:

- Classification, Detection, Segmentation
- Attention, Transformers

Readings:

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

Reminders:

- **Assignment 6:** Deep Learning is out and due **tonight**
- **Quiz 6** is due **tonight**

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

Categorization

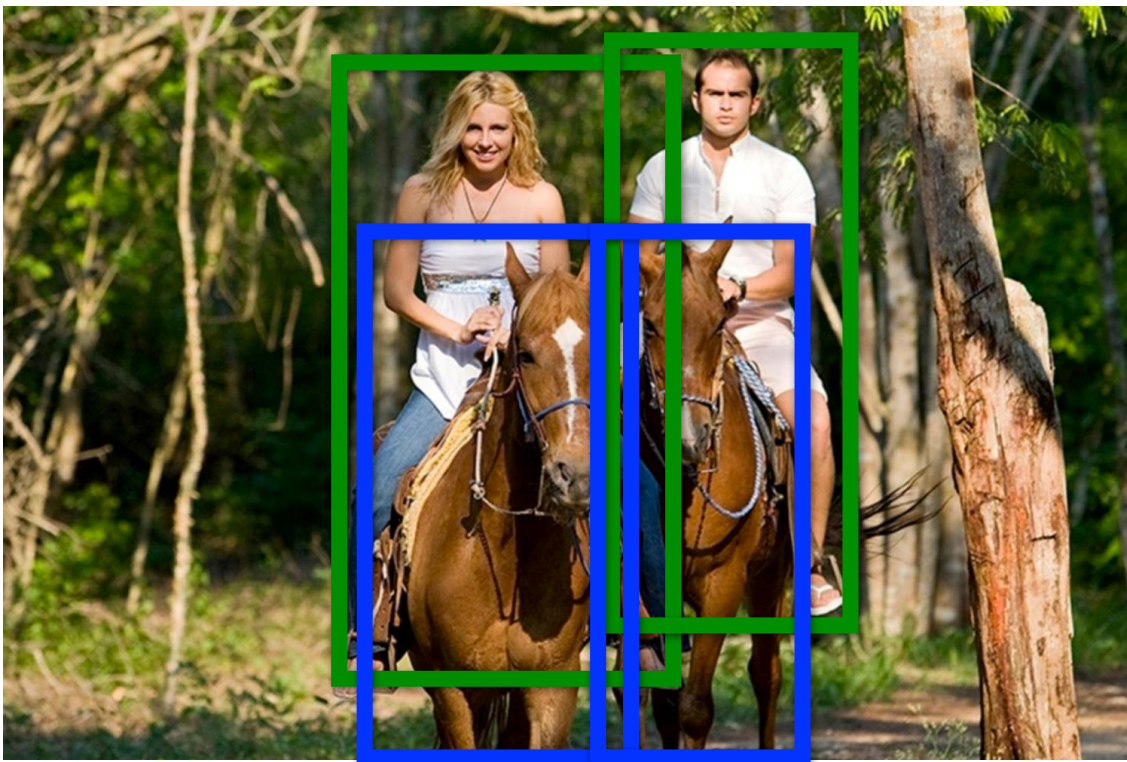


Multi-**class**: Horse
Church
Toothbrush
Person

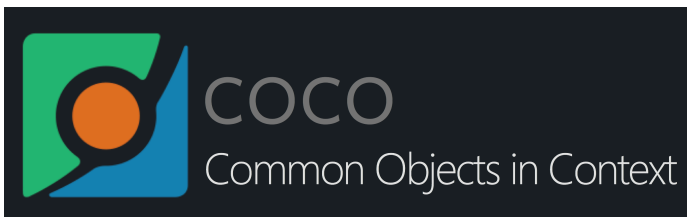


Multi-**label**: **Horse**
Church
Toothbrush
Person

Detection



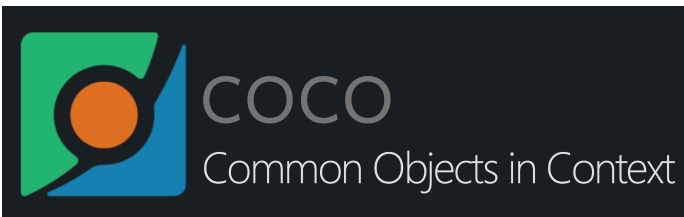
Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)



Segmentation



Horse
Person



Instance Segmentation



Horse1
Horse2
Person1
Person2

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

Categorization

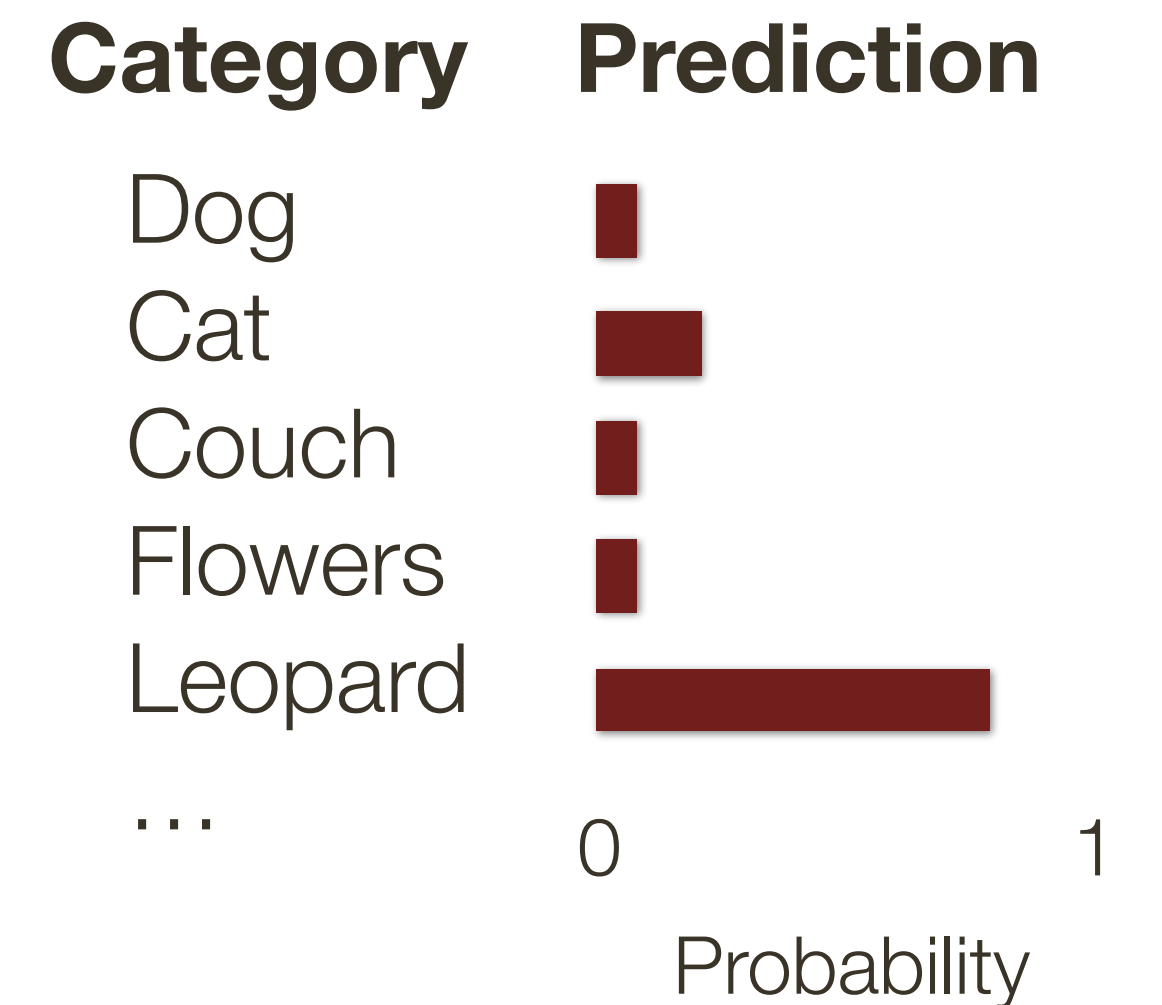
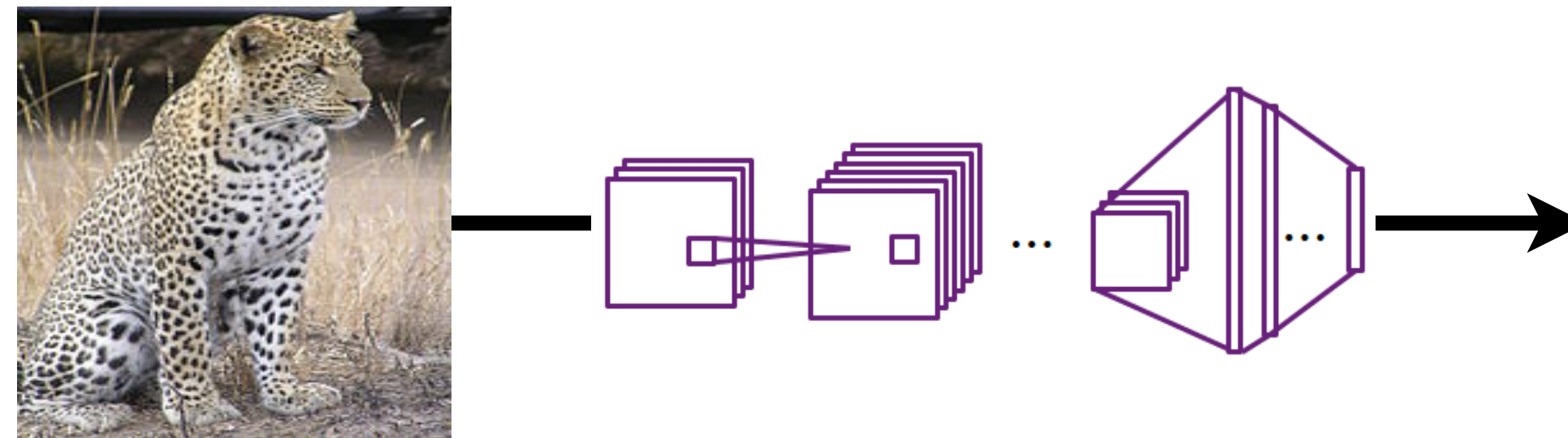


Multi-**class**: Horse
Church
Toothbrush
Person

IMGENET

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

Object Classification



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

ImageNet **Competition** (ILSVRC)

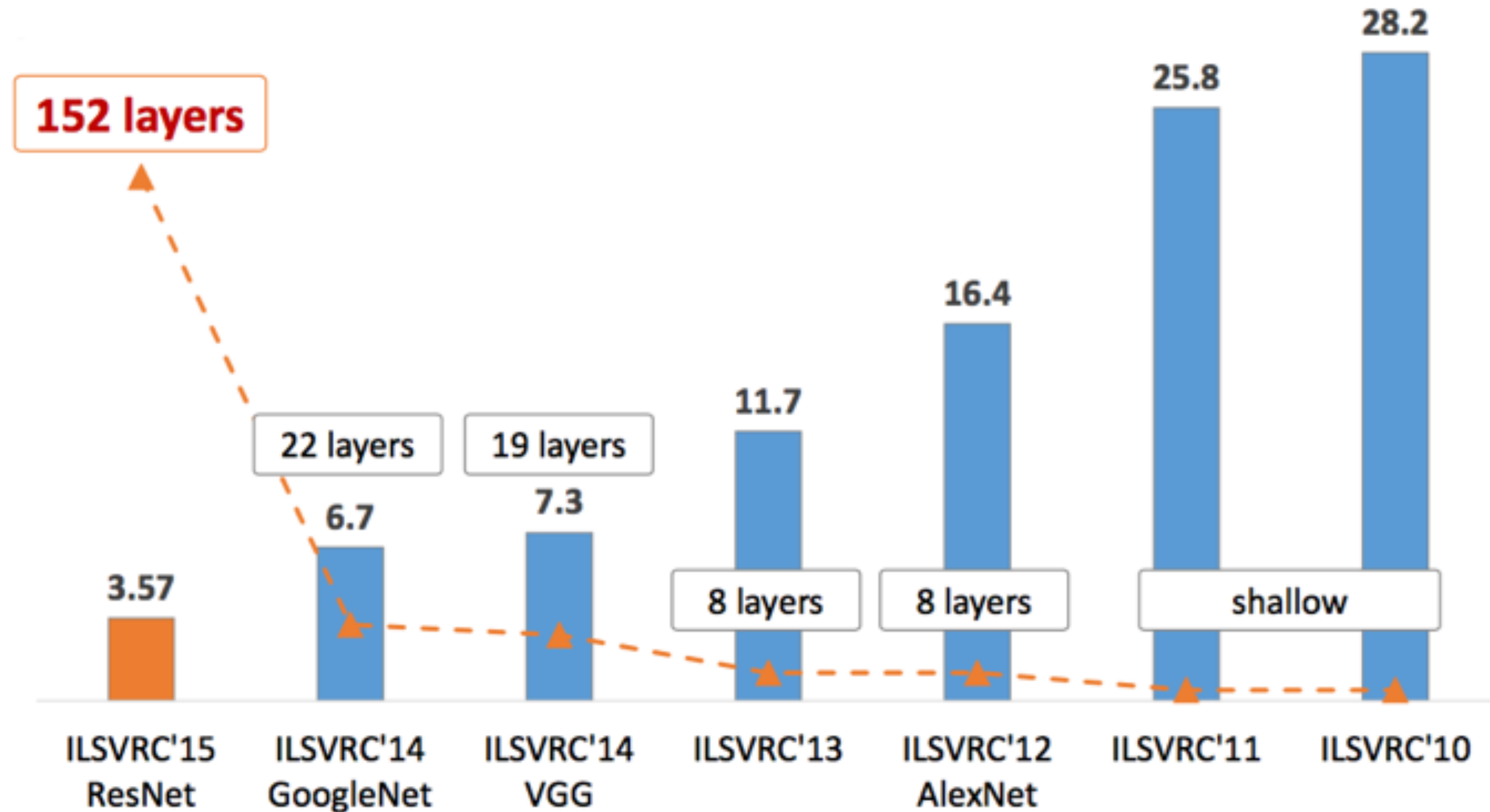
Annual competition of image classification at scale

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

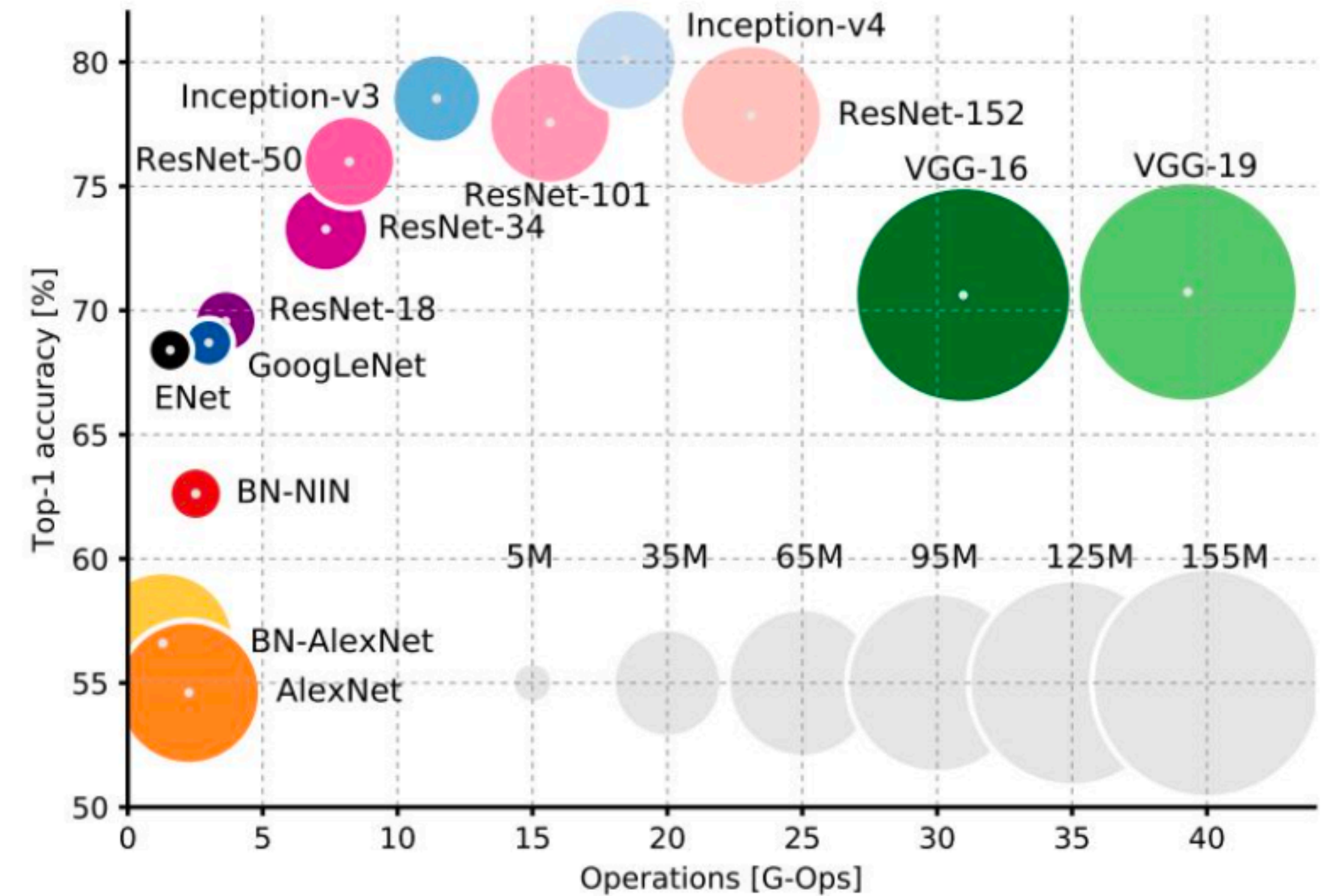
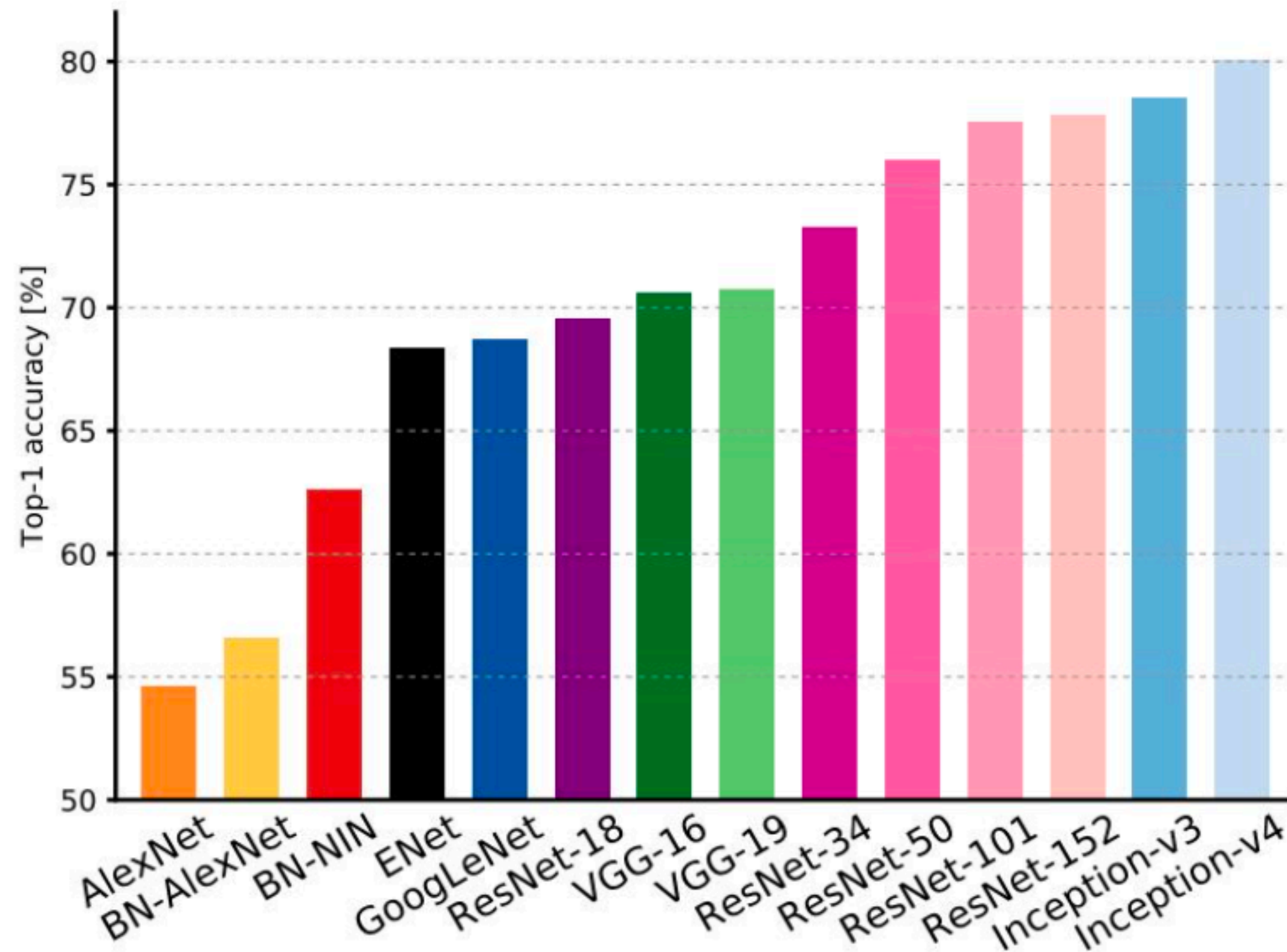
Scoring: need to predict true label within top K (K=5)



ILSVRC winner 2012



Comparing Complexity



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

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

Categorization

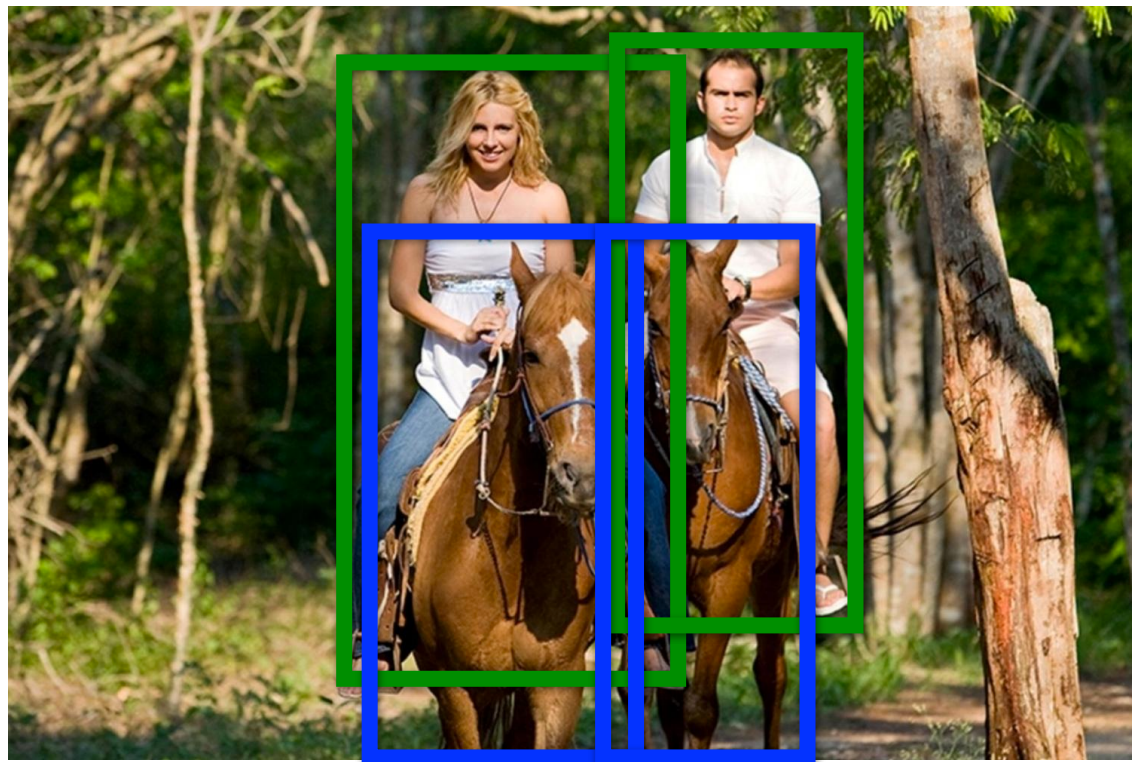


Multi-**class**: Horse
Church
Toothbrush
Person

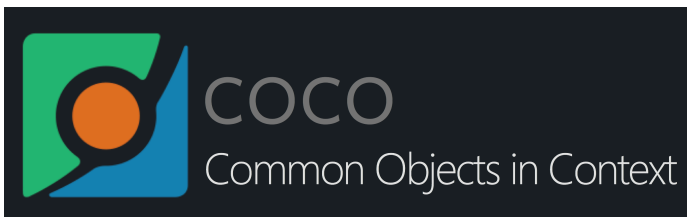


Multi-**label**: **Horse**
Church
Toothbrush
Person

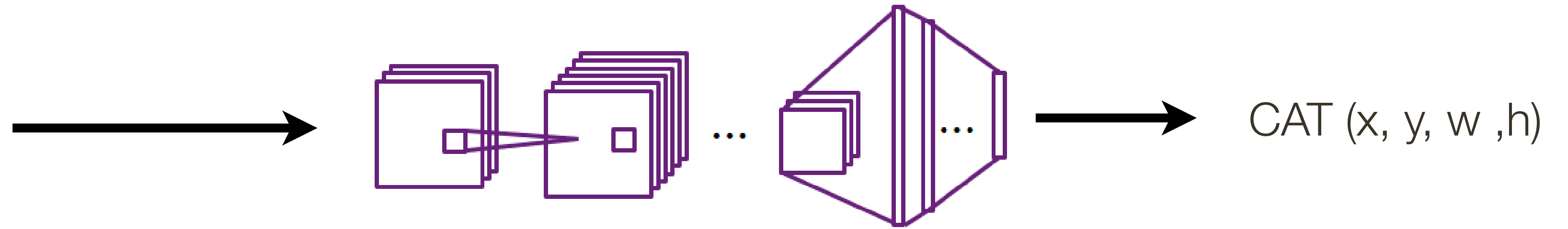
Detection



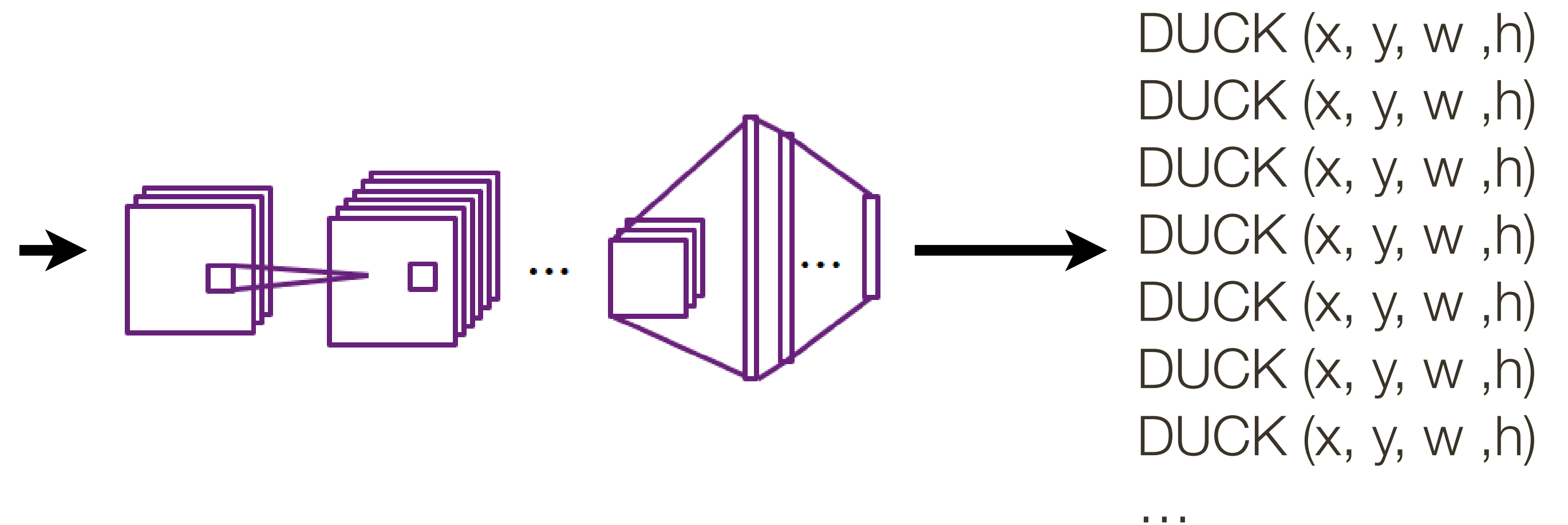
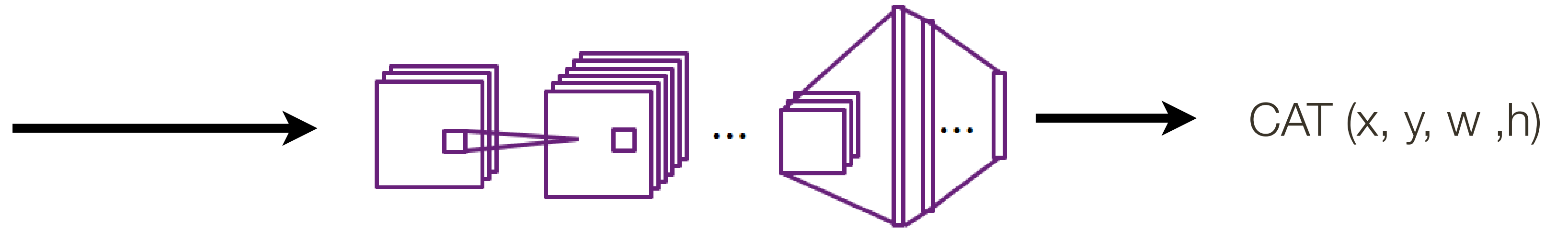
Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)



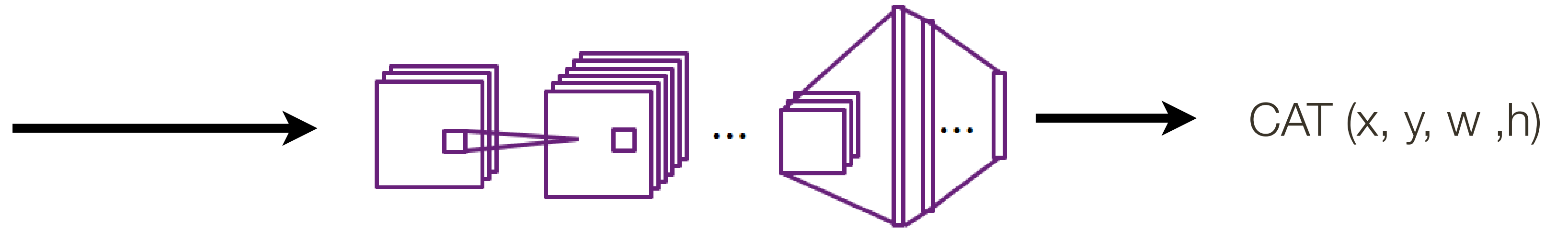
Object **Detection** as Regression Problem



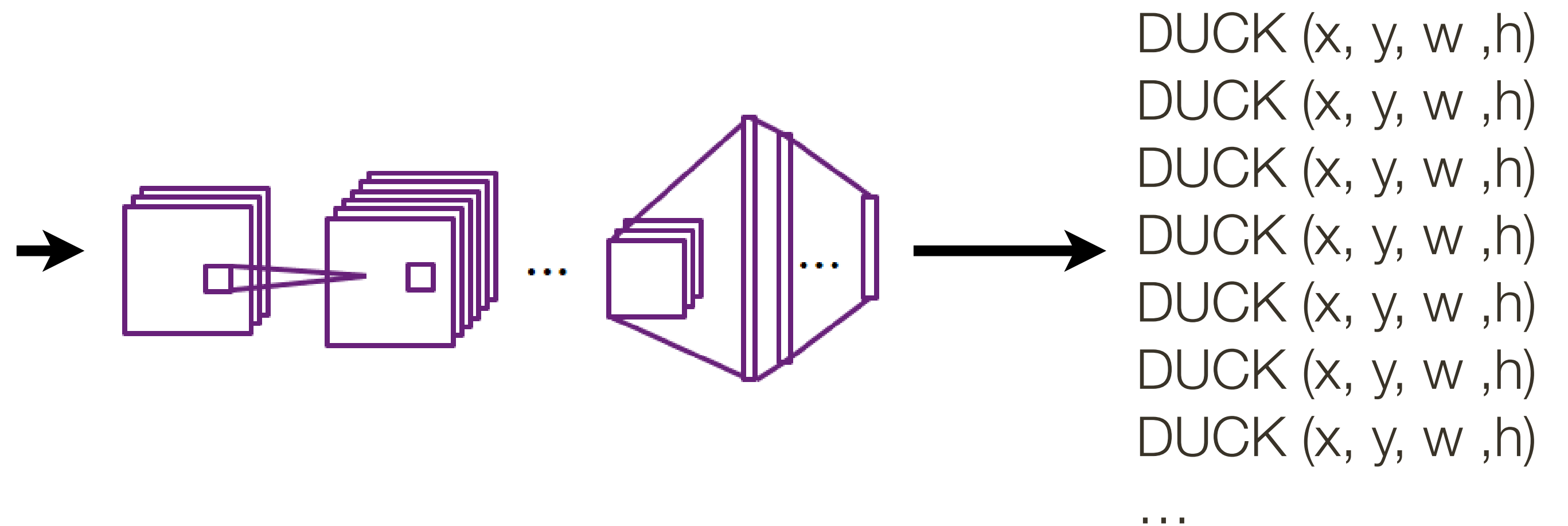
Object **Detection** as Regression Problem



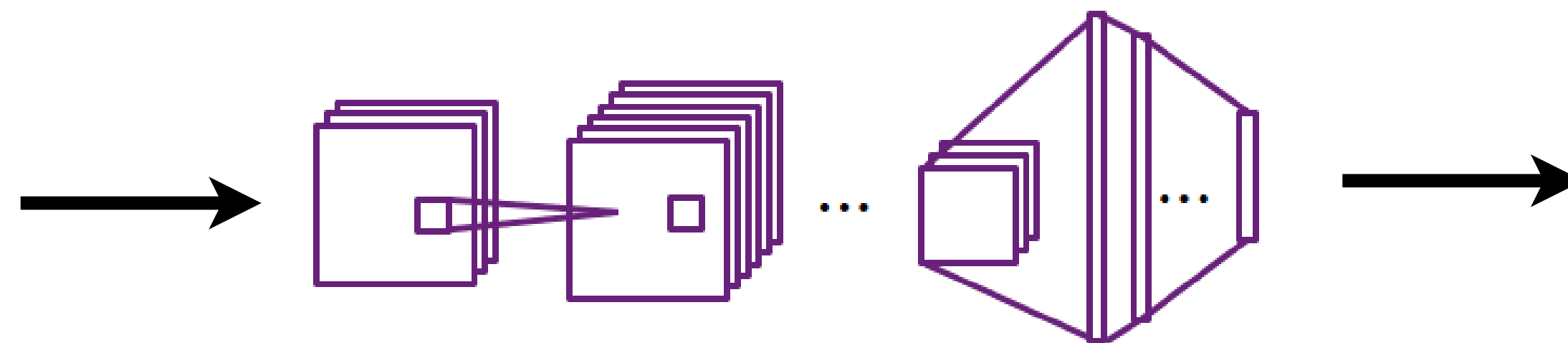
Object **Detection** as Regression Problem



Problem: each image needs a different number of outputs



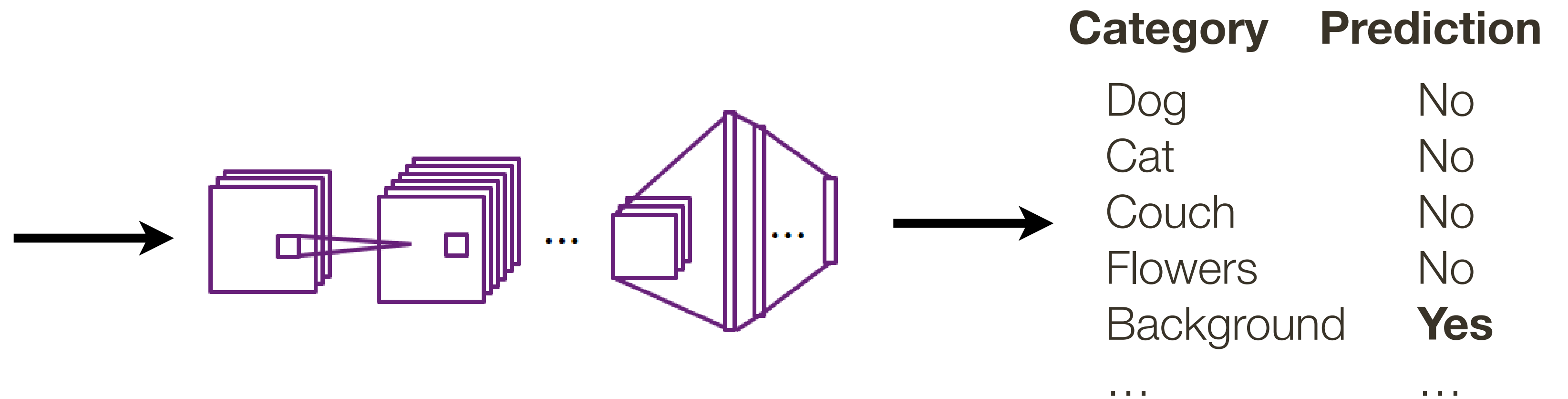
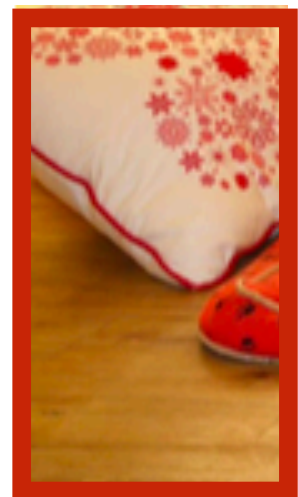
Object **Detection** as Classification Problem



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

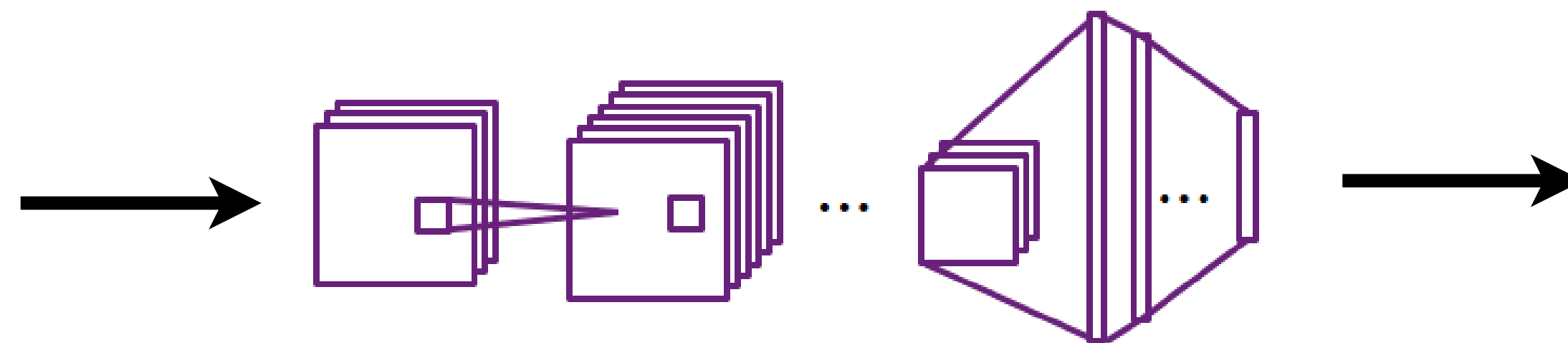
Apply CNN to many different crops in the image and (classification) CNN classifies each patch as object or background

Object **Detection** as Classification Problem



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

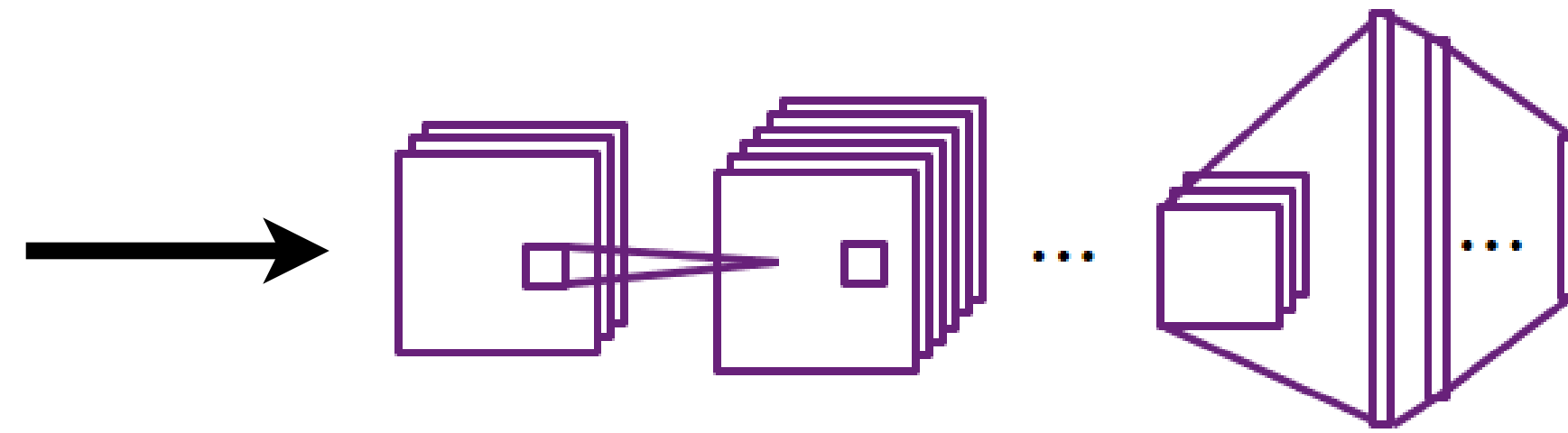
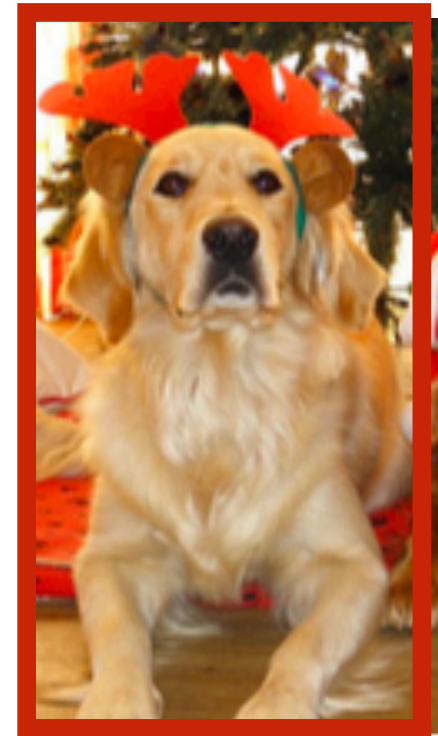
Object **Detection** as Classification Problem



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

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

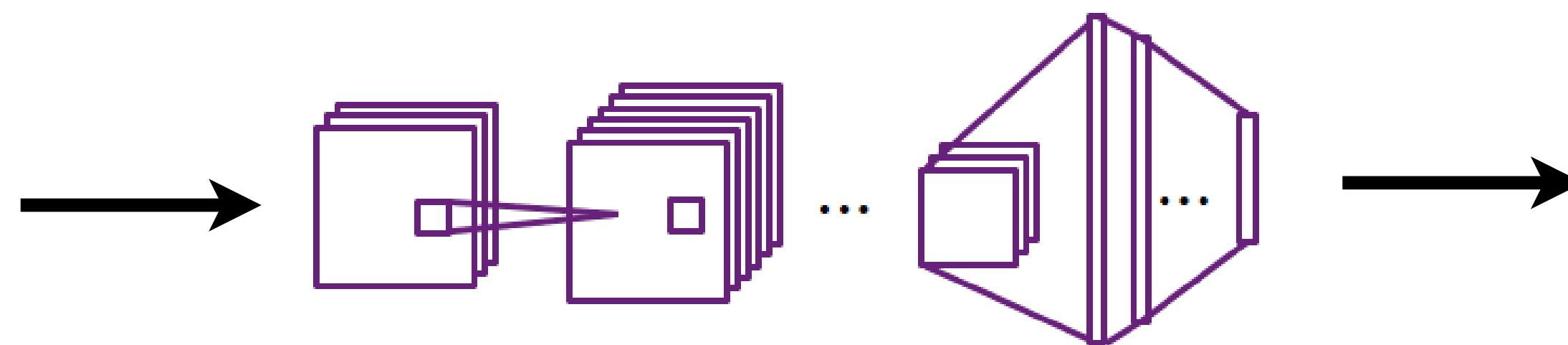
Object **Detection** as Classification Problem



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

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

Object **Detection** as Classification Problem

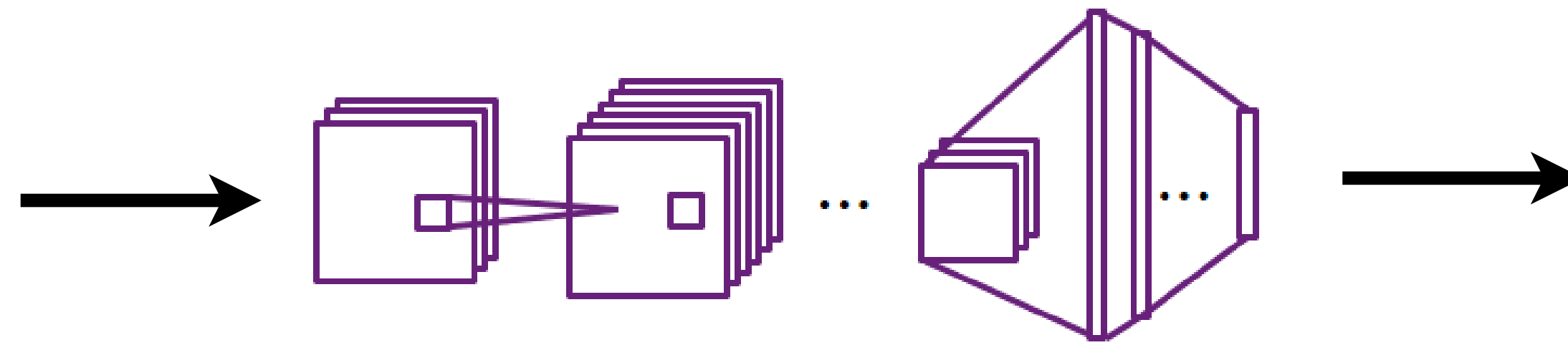


Category	Prediction
Dog	No
Cat	Yes
Couch	No
Flowers	No
Background	No
...	...

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

Object **Detection** as Classification Problem

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



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

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

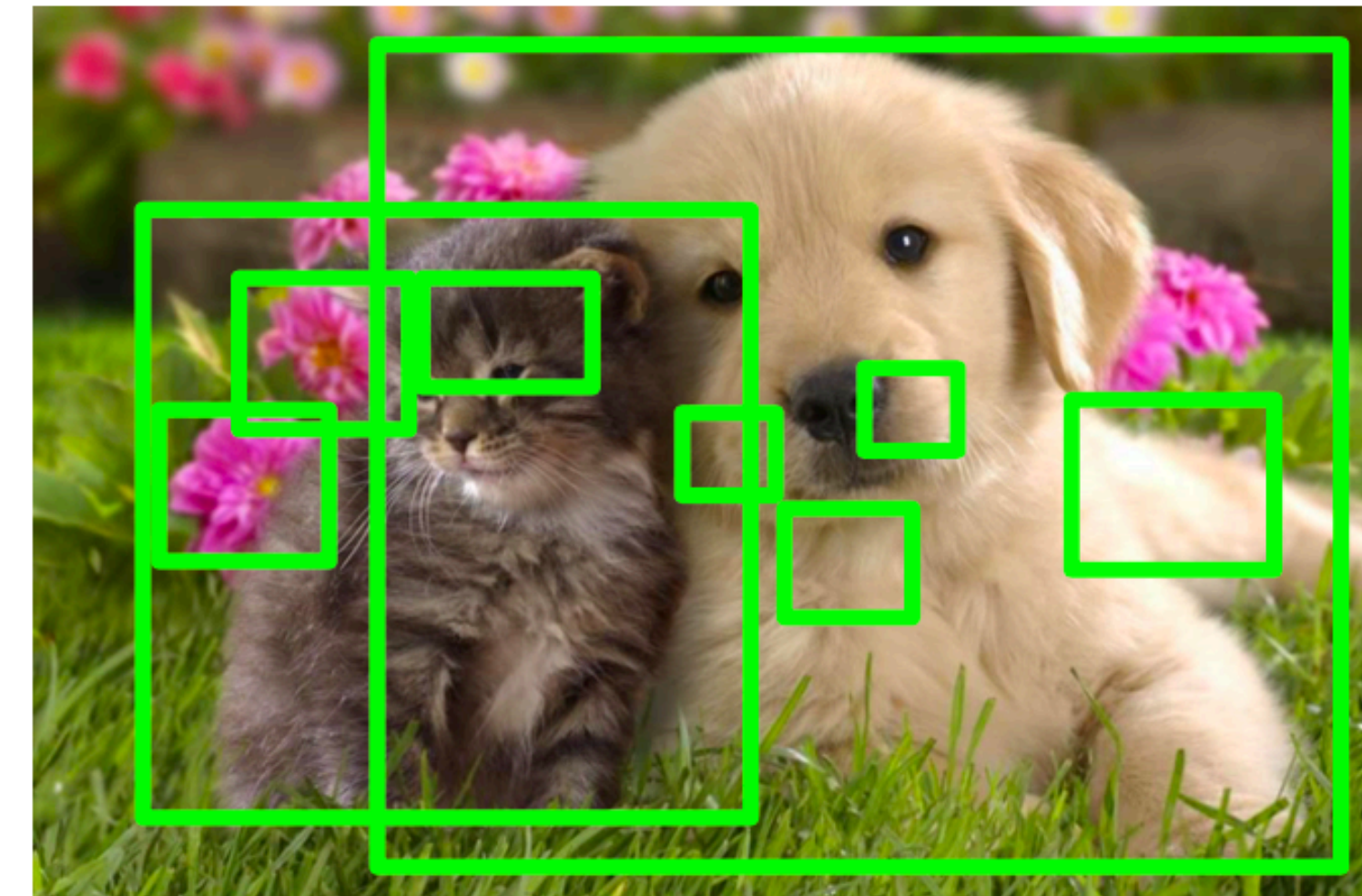
Region Proposals (older idea in vision)

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

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

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

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



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

R-CNN

[Girshick et al, CVPR 2014]



Input **Image**

* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]



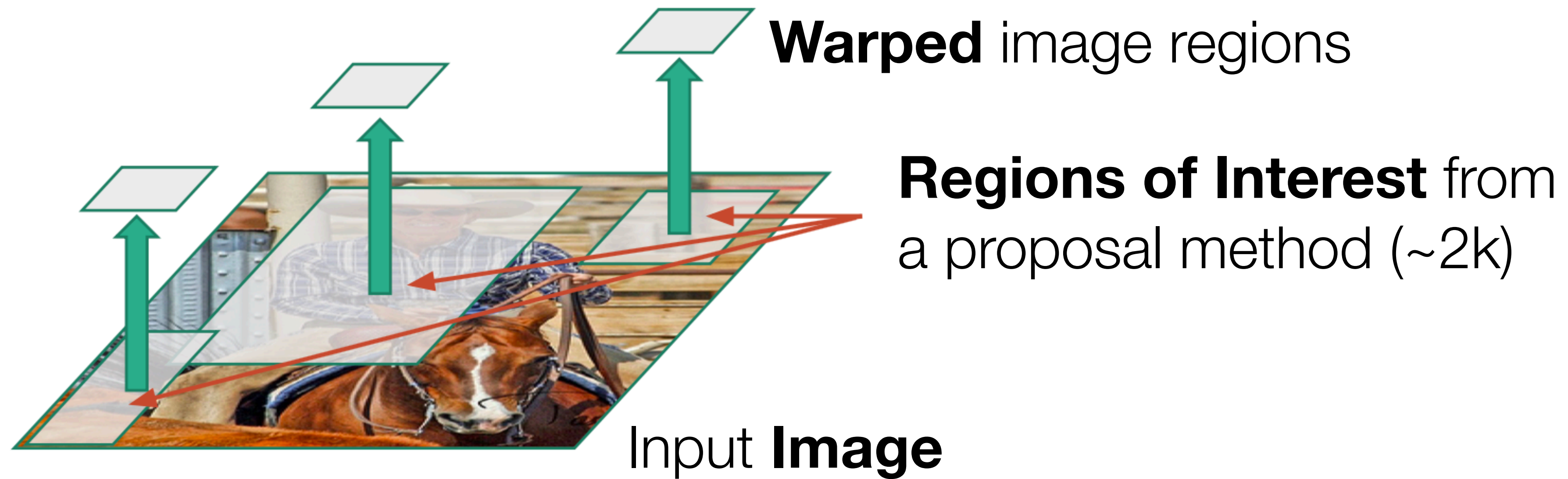
Regions of Interest from
a proposal method (~2k)

Input **Image**

* image from Ross Girshick

R-CNN

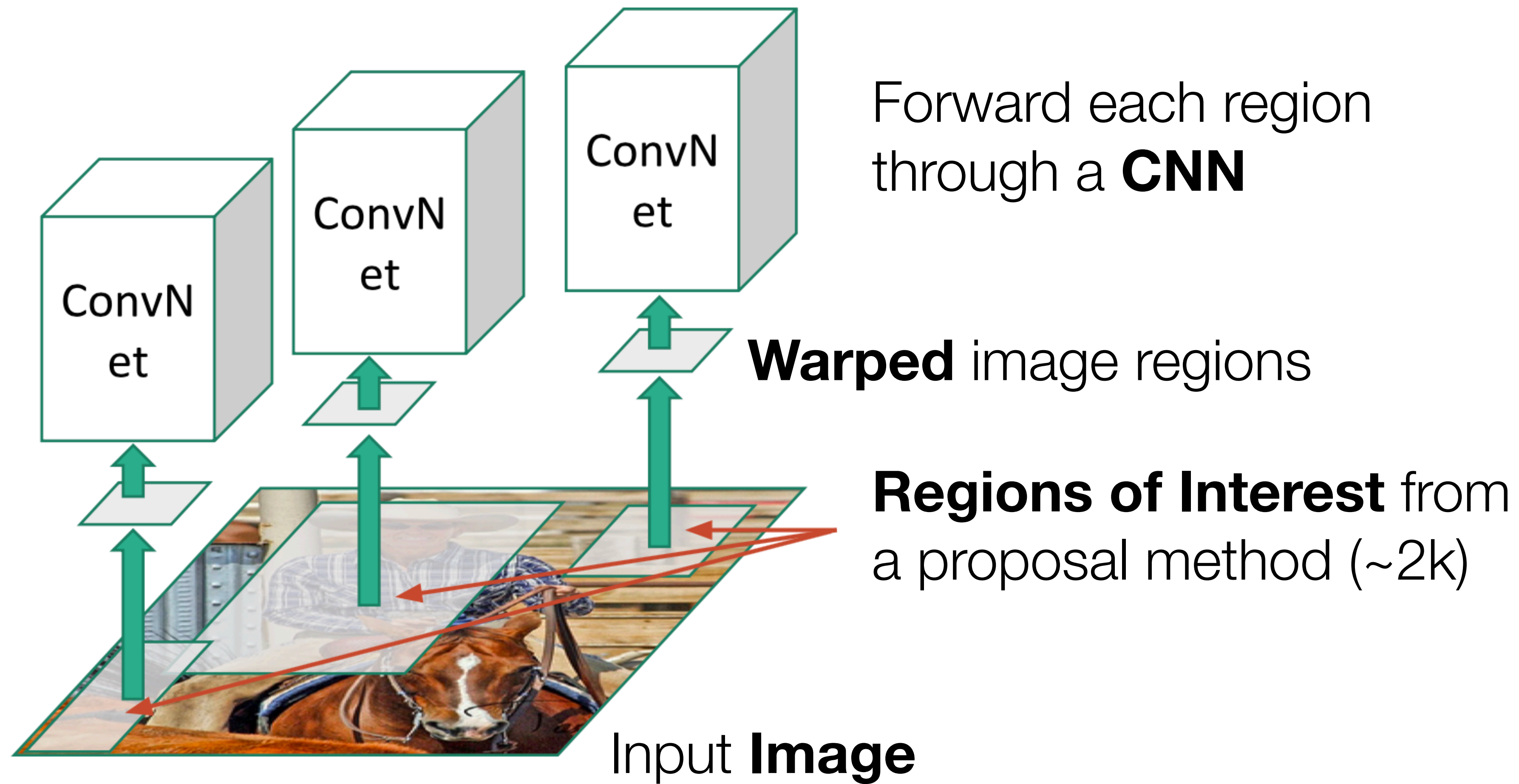
[Girshick et al, CVPR 2014]



* image from Ross Girshick

R-CNN

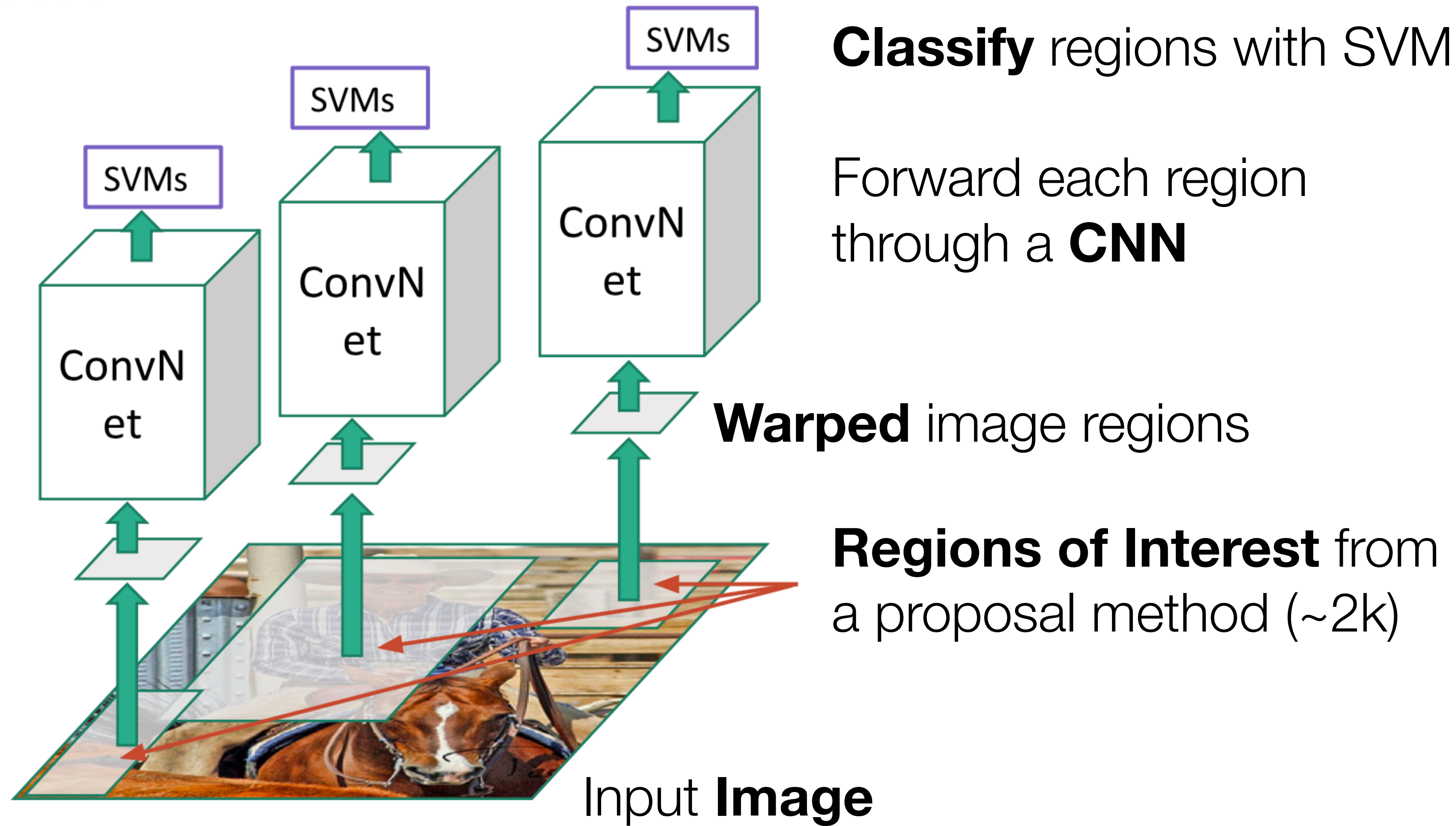
[Girshick et al, CVPR 2014]



* image from Ross Girshick

R-CNN

[Girshick et al, CVPR 2014]

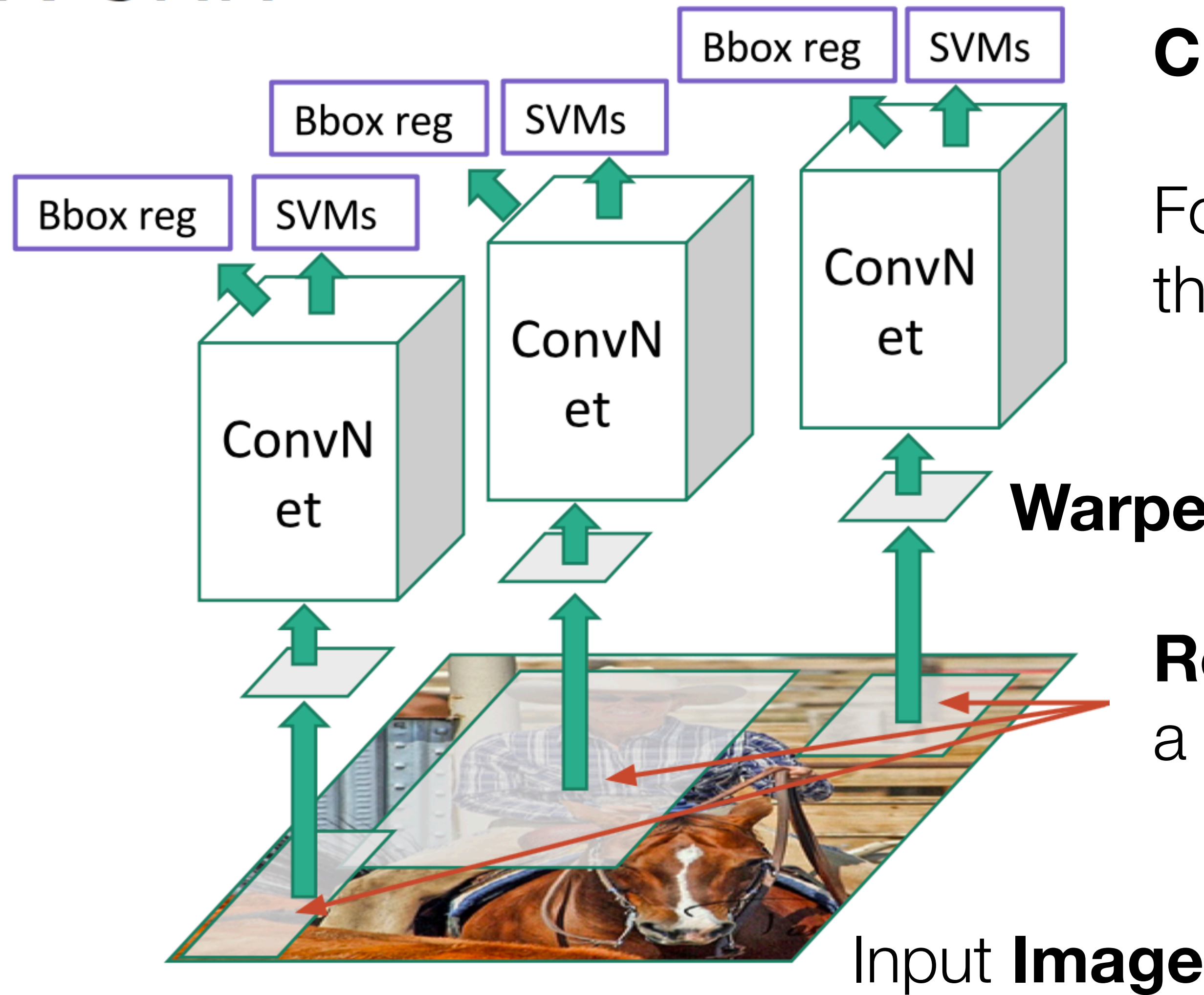


* image from Ross Girshick

R-CNN

Linear Regression for bounding box offsets

[Girshick et al, CVPR 2014]



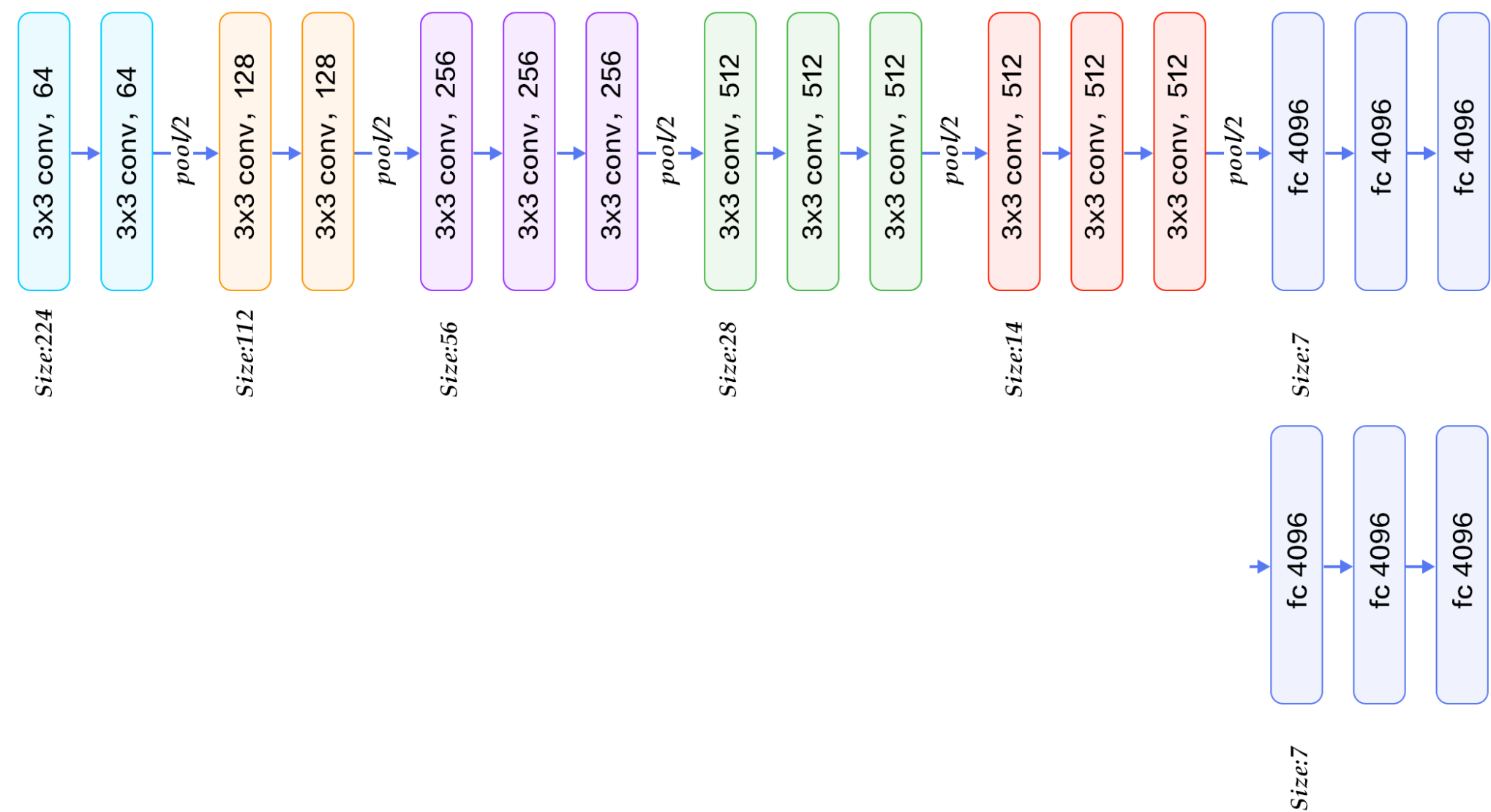
Classify regions with SVM

Forward each region through a **CNN**

Warped image regions

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

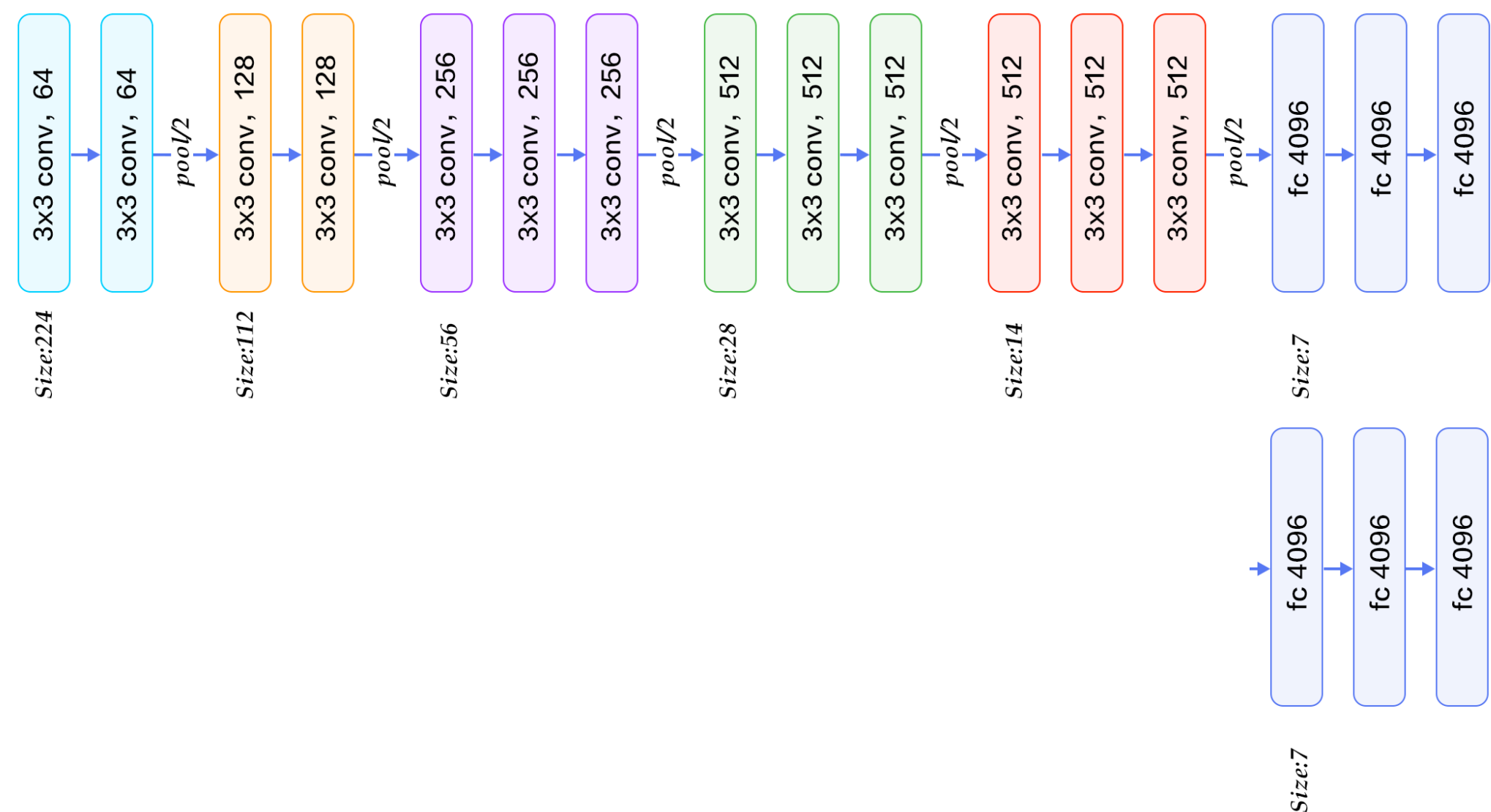
Proposal 1 (x, y, w, h)
(resized to 224 x 224)



Classifier
(4096 x 80 categories + Softmax)

Regressor
(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

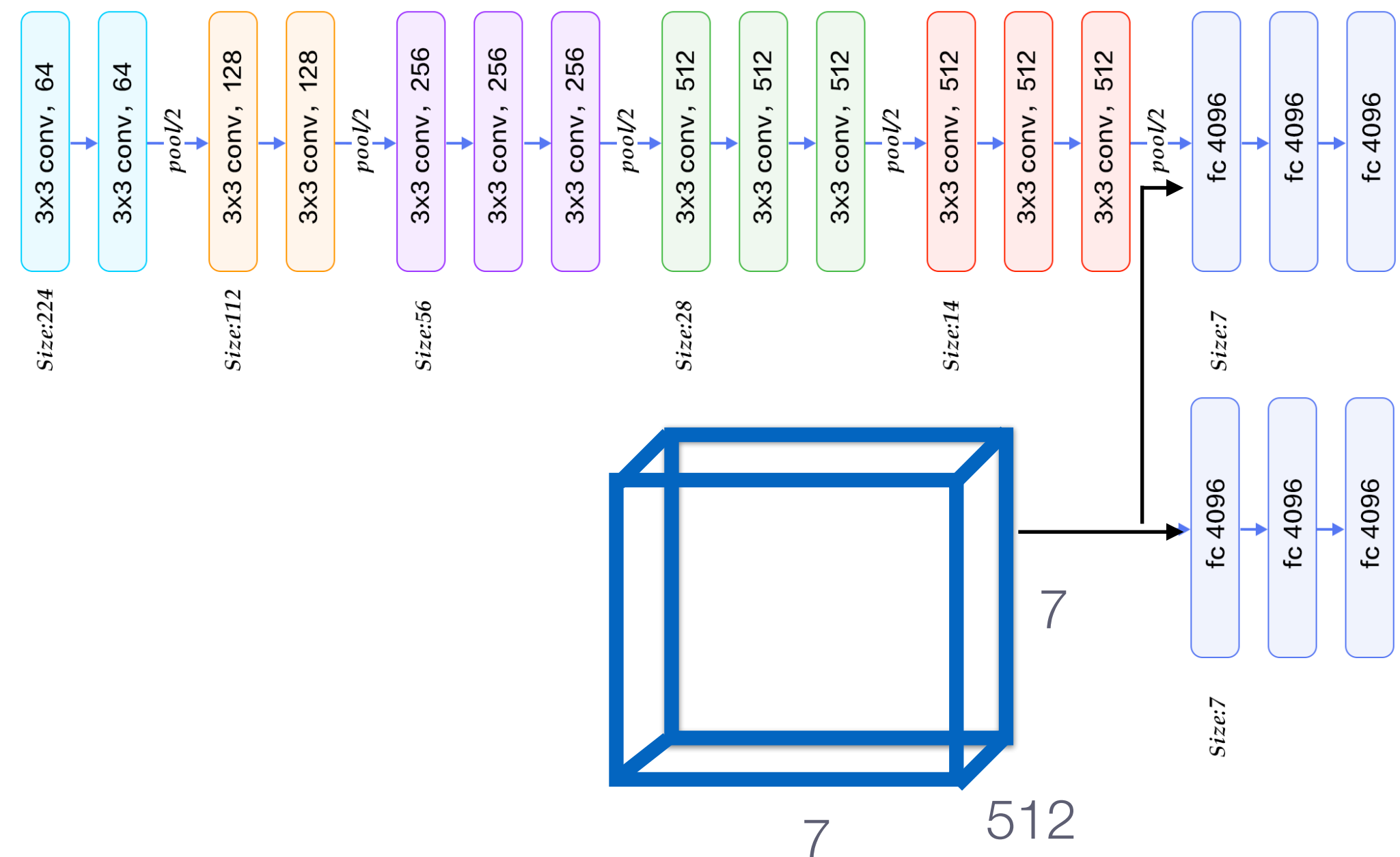
Proposal 2 (x, y, w, h)
(resized to 224 x 224)



Classifier
(4096 x 80 categories + Softmax)

Regressor
(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

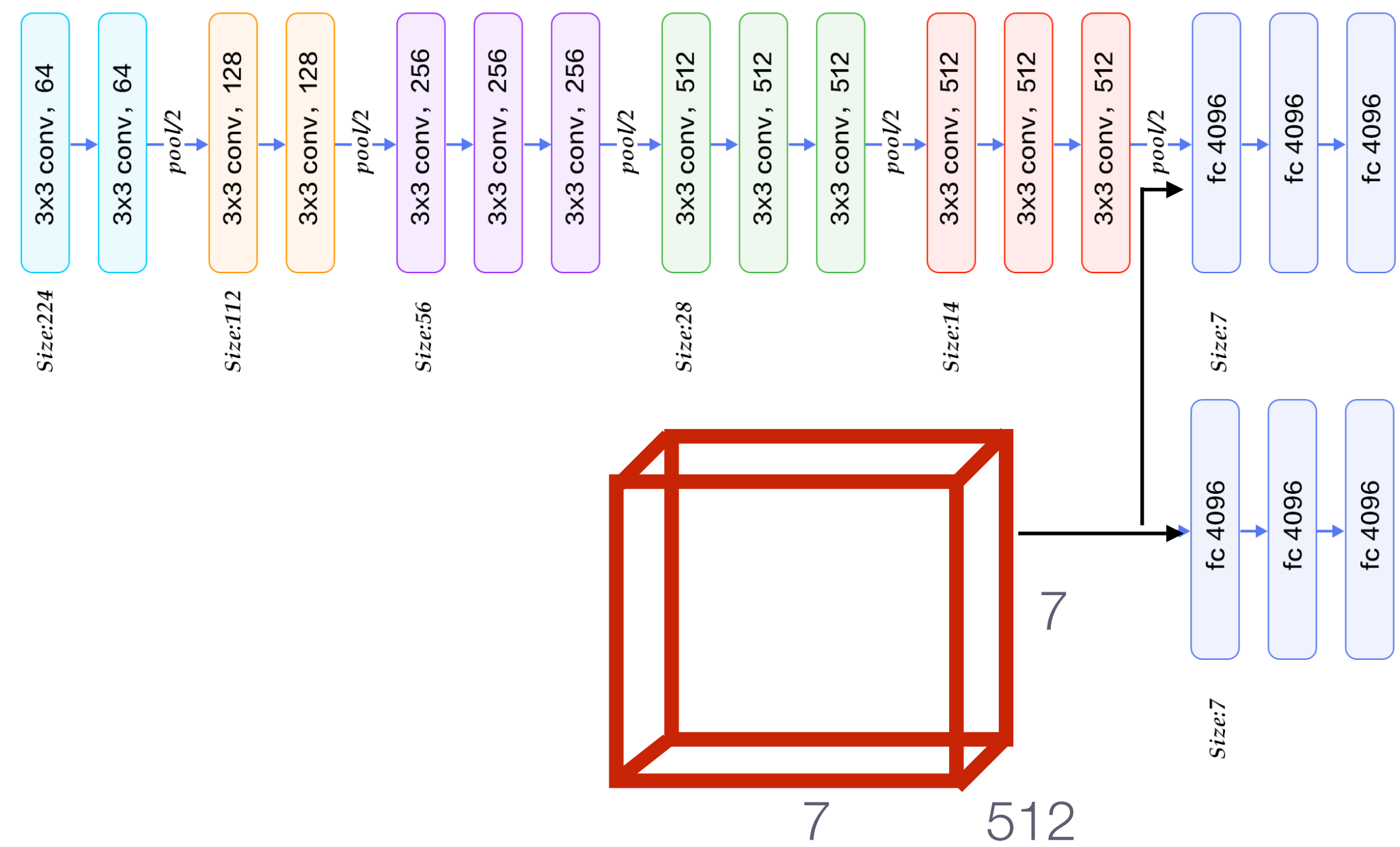
Proposal 1 (x, y, w, h)
(resized to 224 x 224)



Classifier
(4096 x 80 categories + Softmax)

Regressor
(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

Proposal 2 (x, y, w, h)
(resized to 224 x 224)



Classifier
(4096 x 80 categories + Softmax)

Regressor
(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

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**

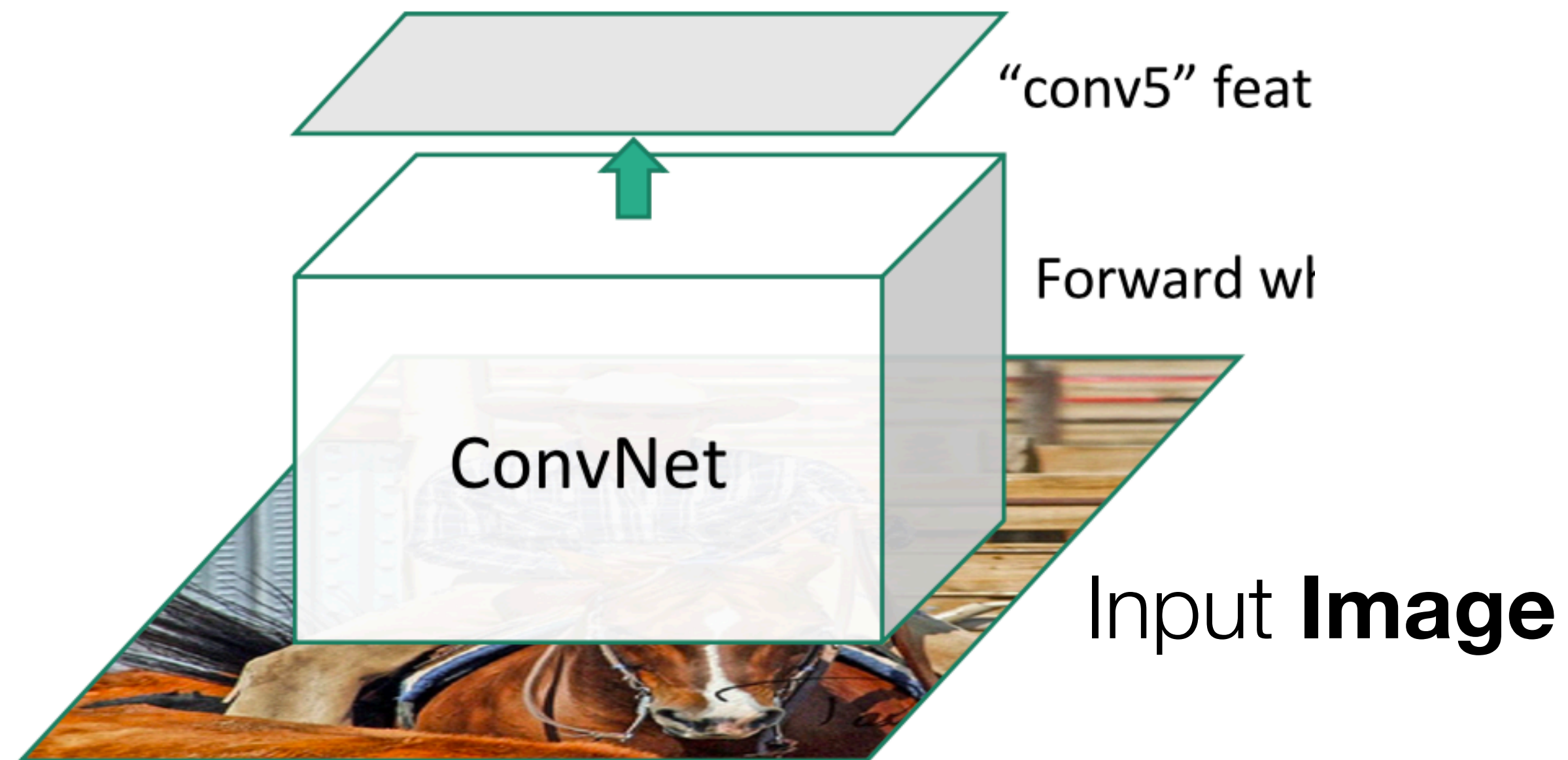
[Girshick et al, ICCV 2015]



Input **Image**

Fast **R-CNN**

[Girshick et al, ICCV 2015]



* image from Ross Girshick

Fast R-CNN

[Girshick et al, ICCV 2015]



* image from Ross Girshick

Fast R-CNN

[Girshick et al, ICCV 2015]

Regions of Interest
from the
proposal
method

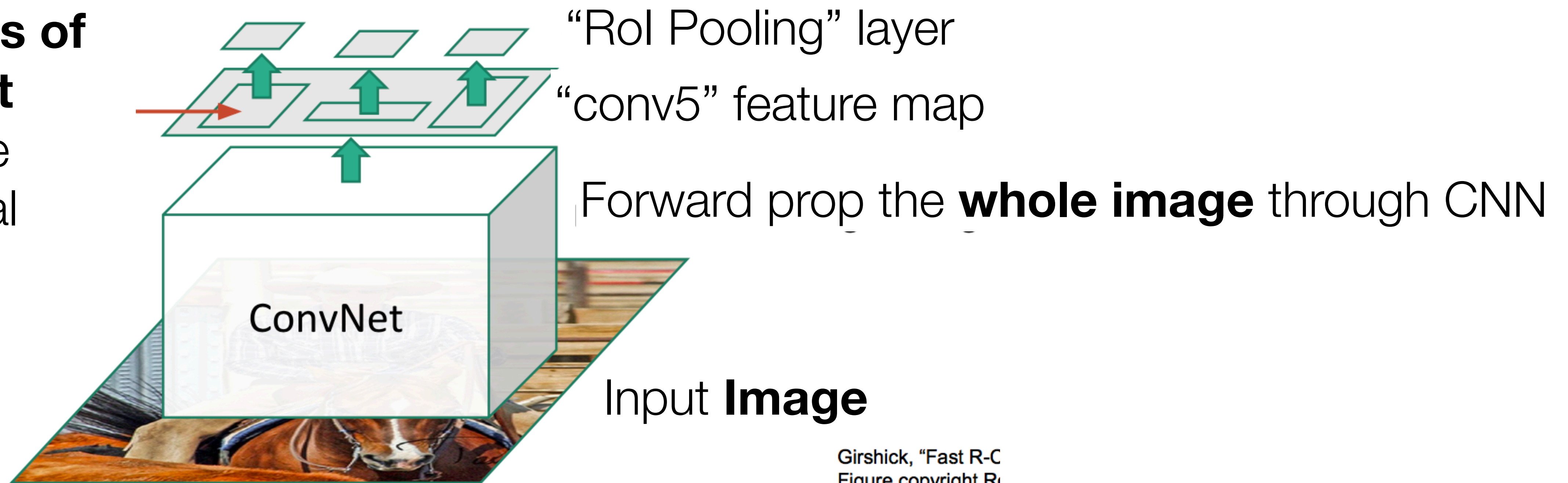


* image from Ross Girshick

Fast R-CNN

[Girshick et al, ICCV 2015]

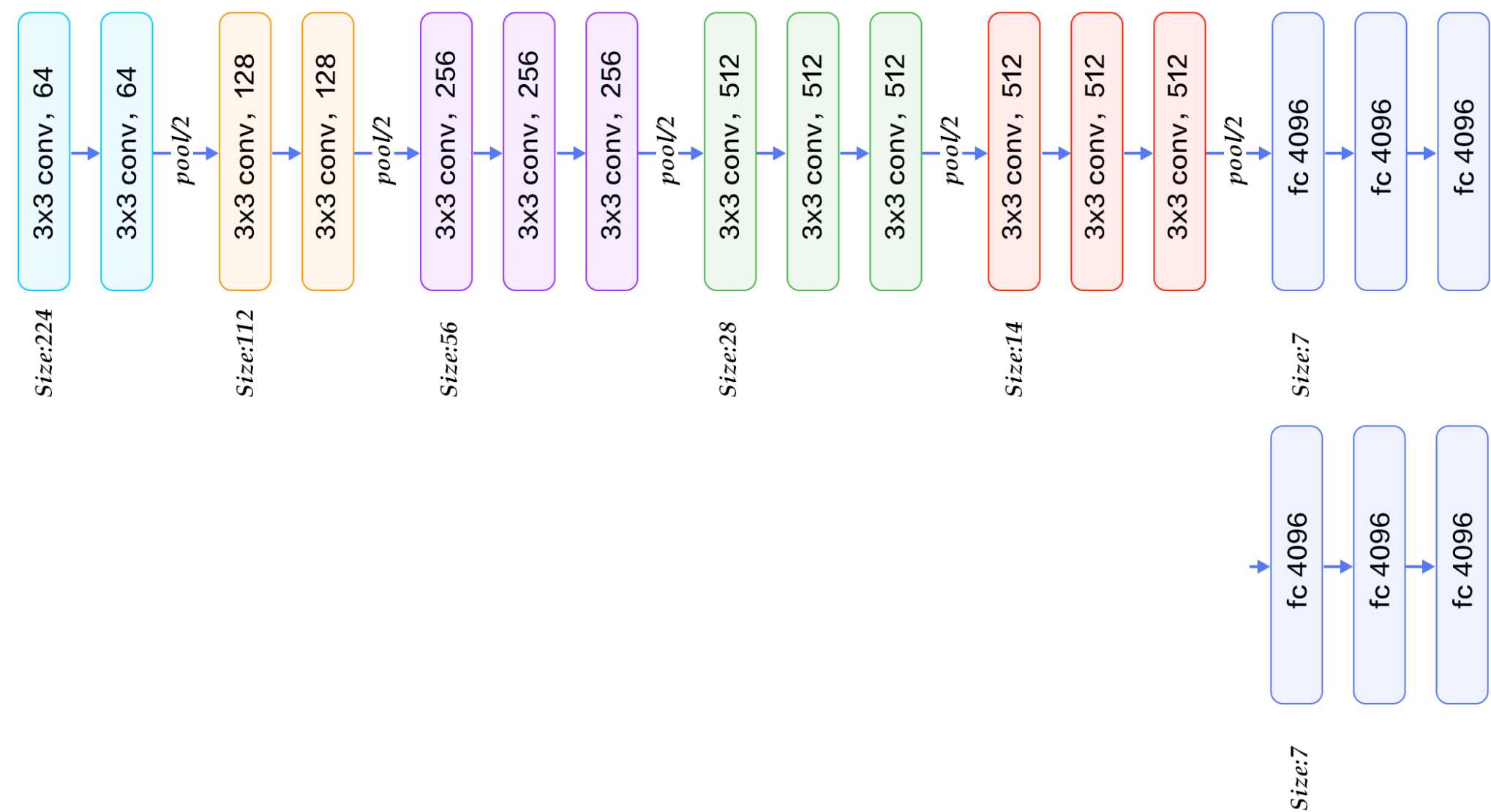
Regions of Interest
from the
proposal
method



Girshick, "Fast R-C
Figure copyright R

* image from Ross Girshick

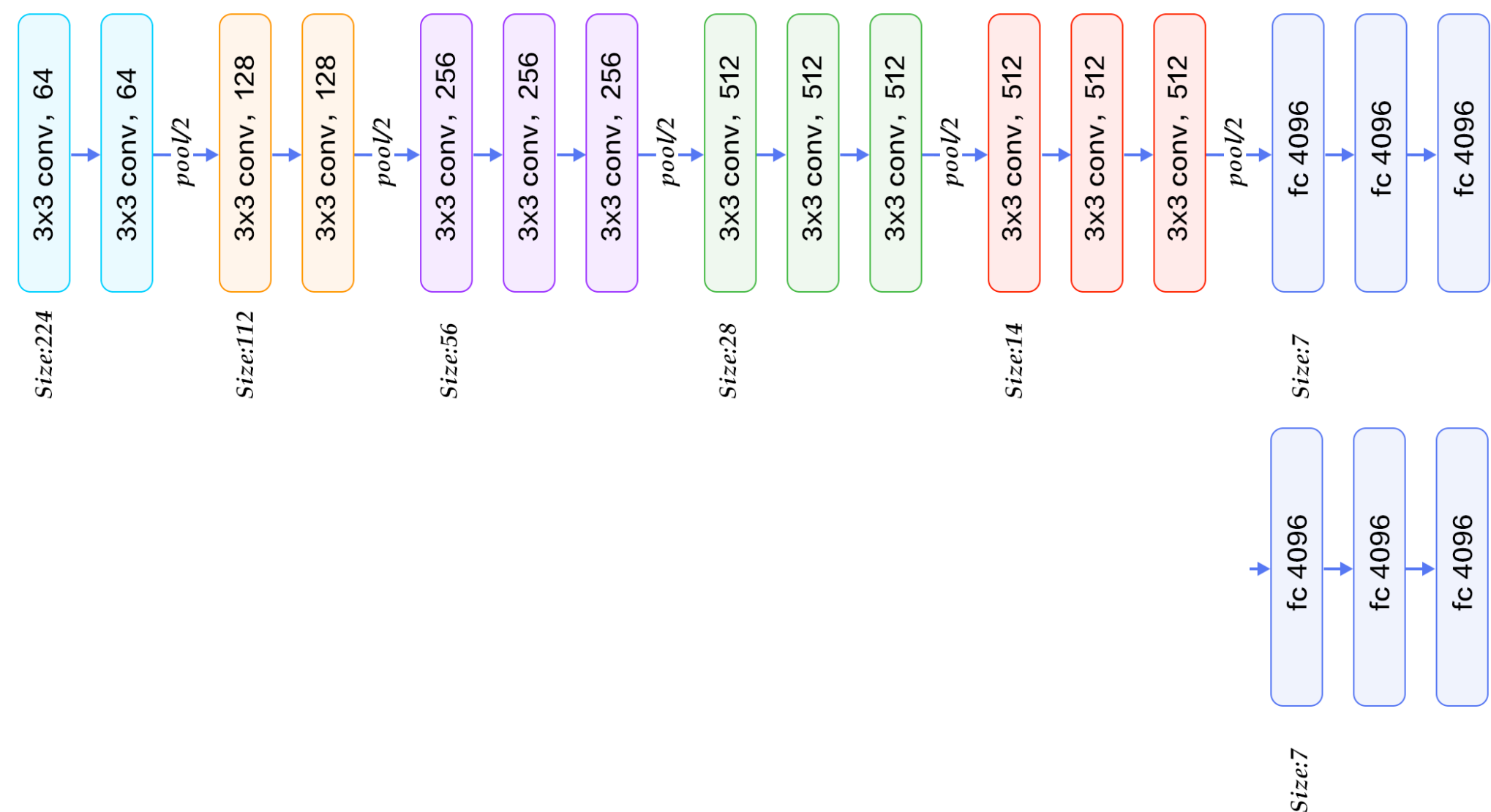
Proposal 1 (x, y, w, h)
(resized to 224 x 224)



Classifier
(4096 x 80 categories + Softmax)

Regressor
(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

Proposal 2 (x, y, w, h)
(resized to 224 x 224)



Classifier
(4096 x 80 categories + Softmax)

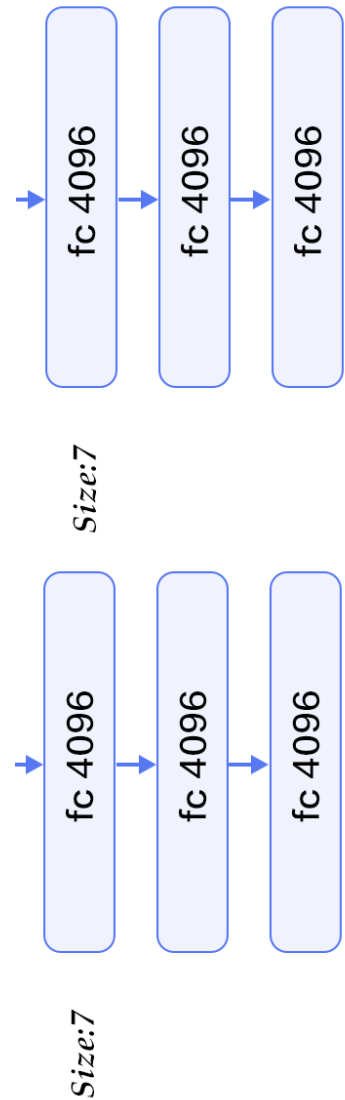
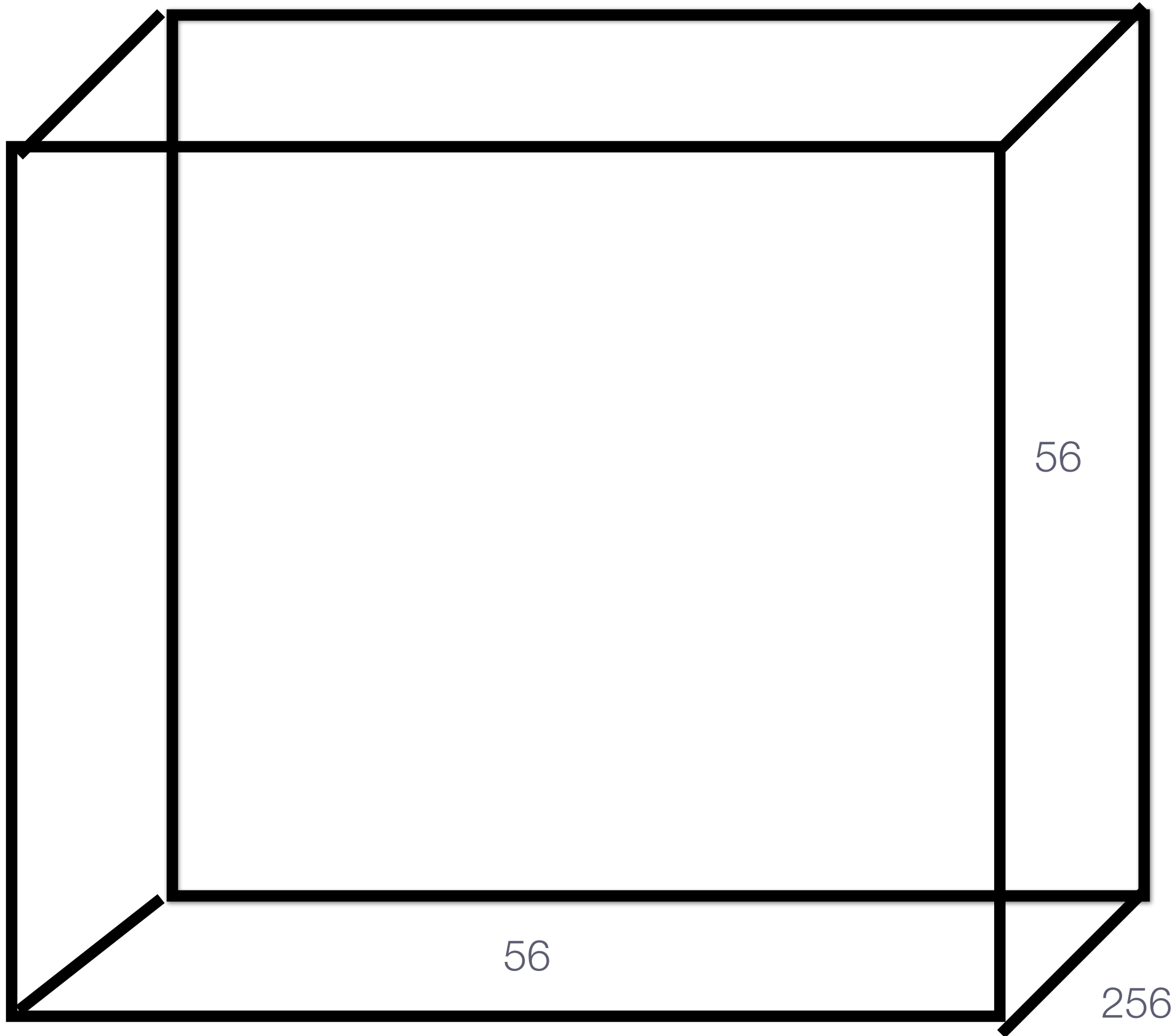
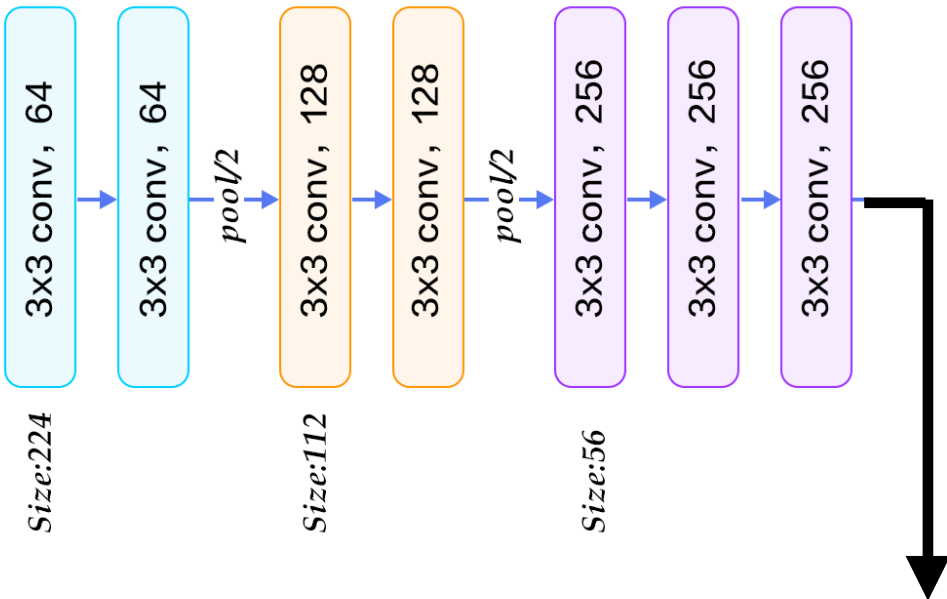
Regressor
(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

RoI Align

Image
(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



Classifier
(4096 x 80 categories + Softmax)

Regressor
(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

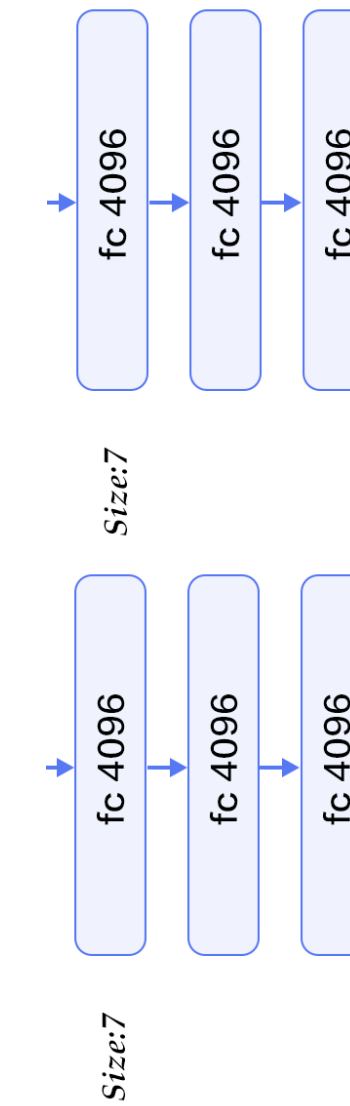
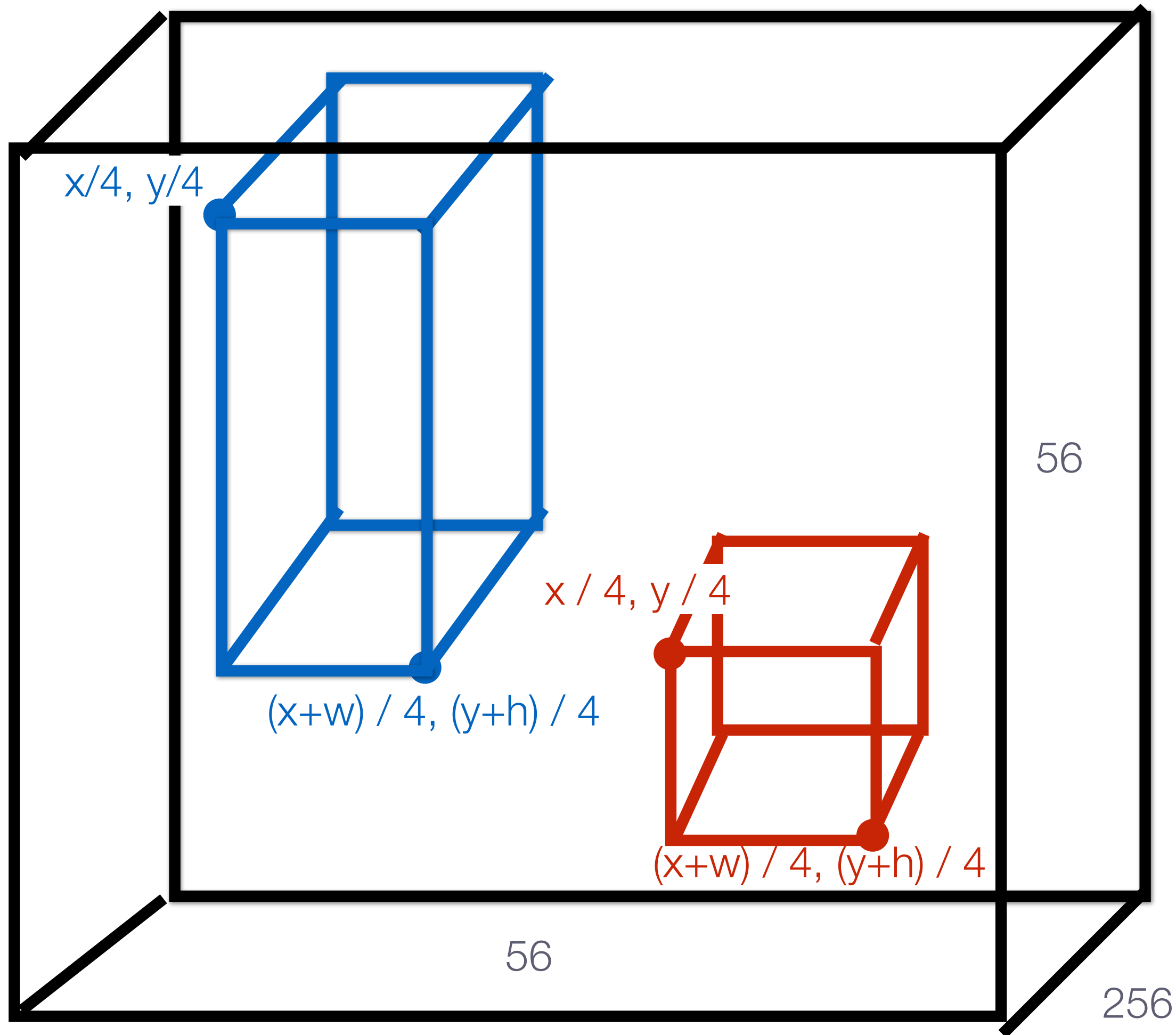
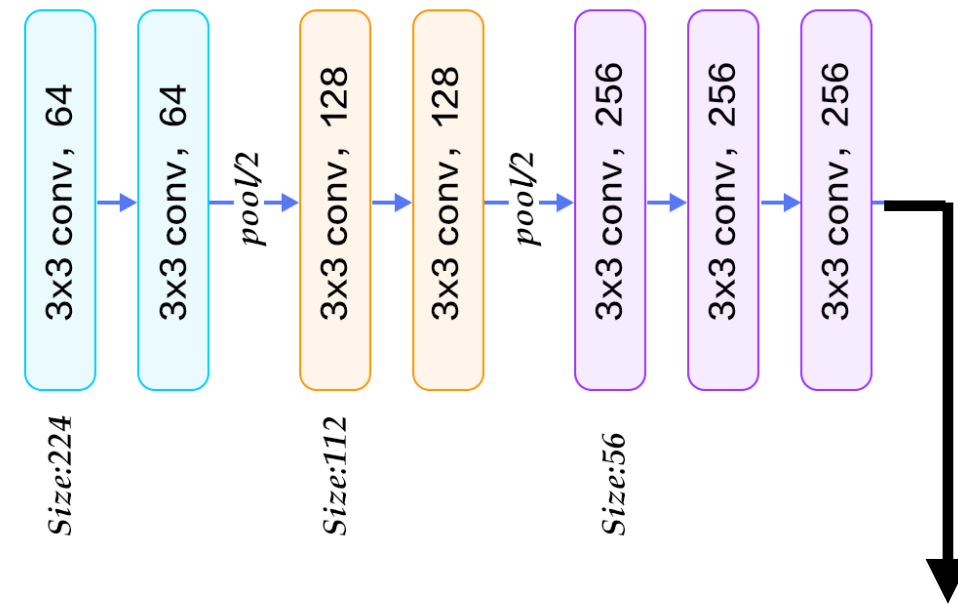
RoI Align

Image

(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



Classifier

(4096 x 80 categories + Softmax)

Regressor

(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

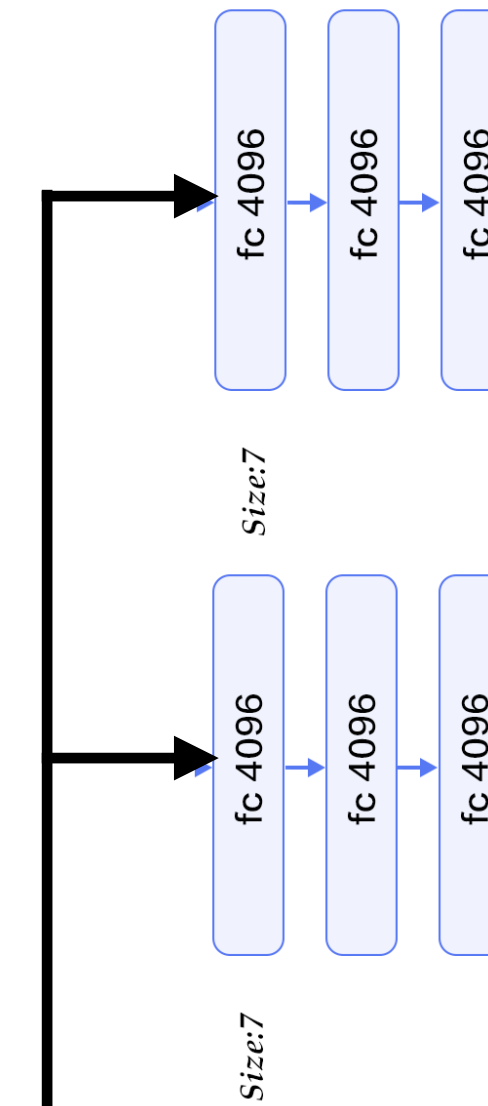
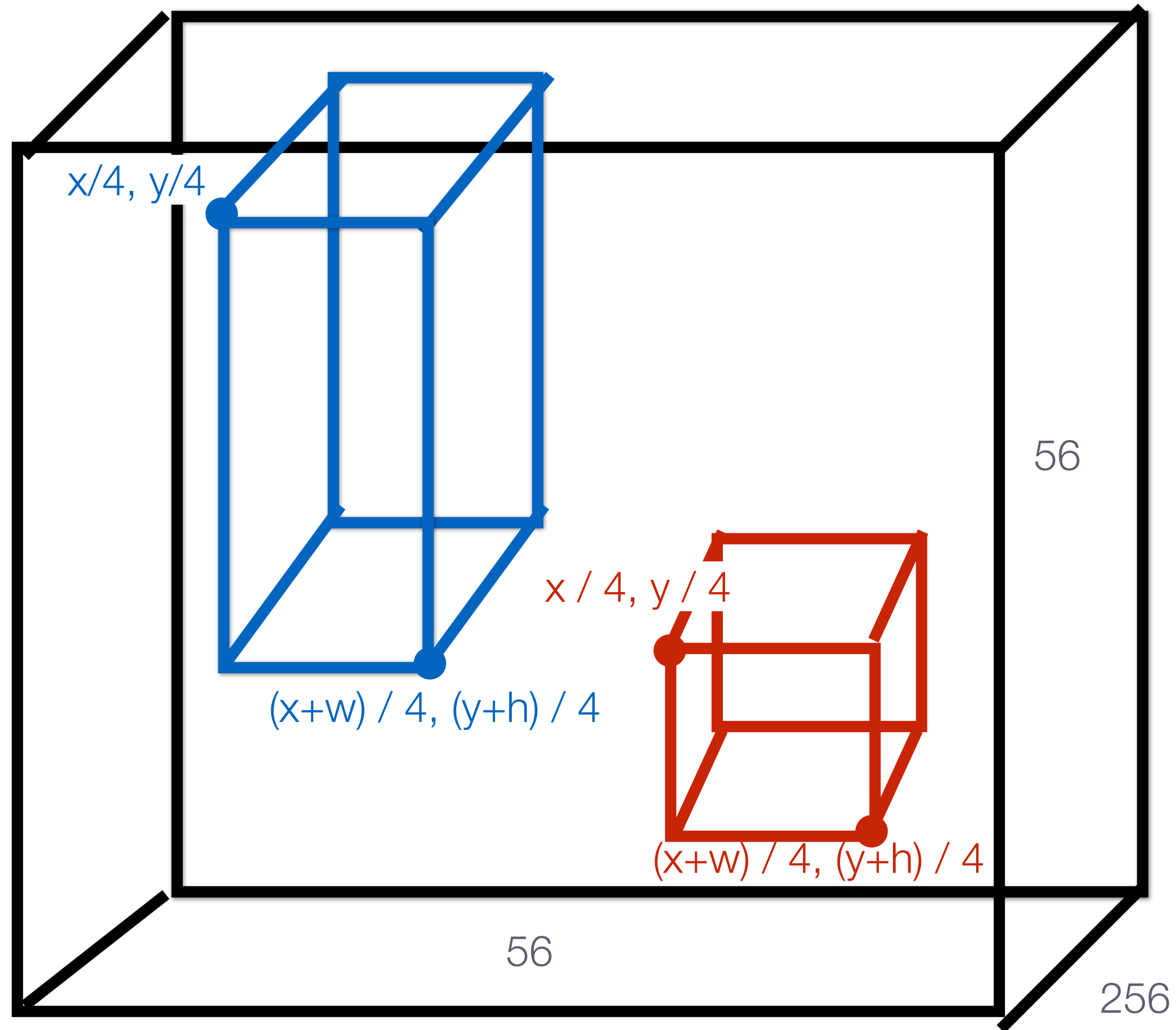
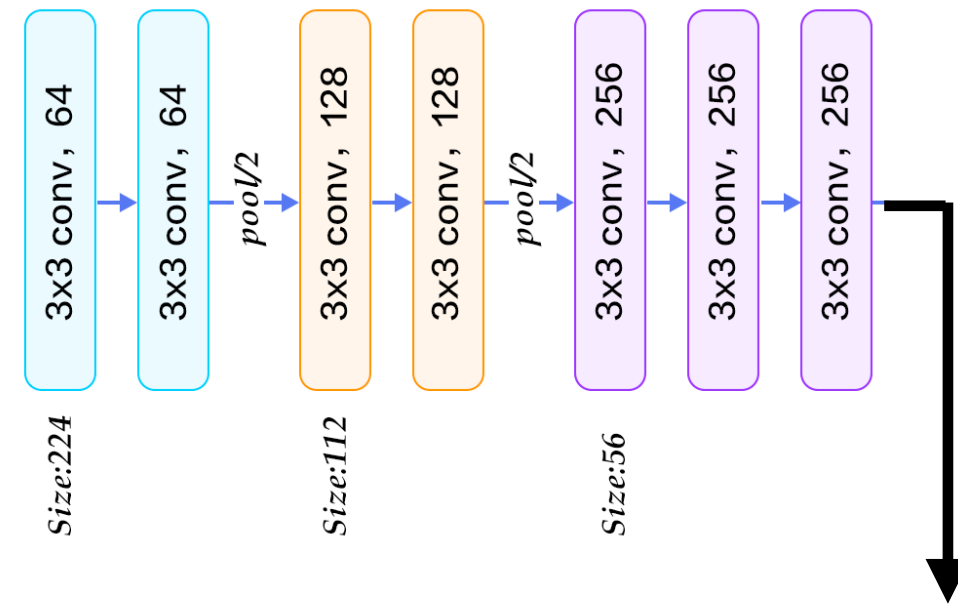
RoI Align

Image

(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



Classifier

(4096 x 80 categories + Softmax)

Regressor

(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h

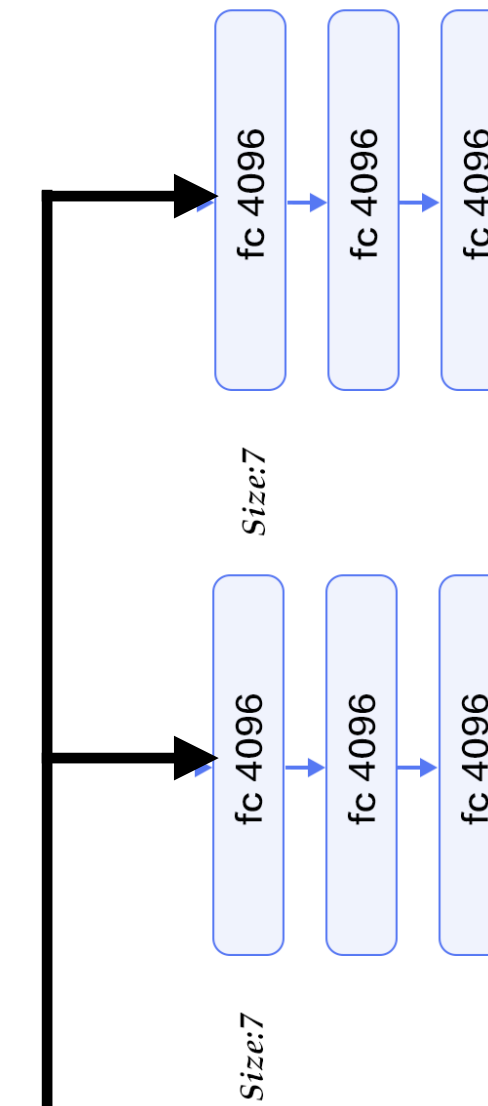
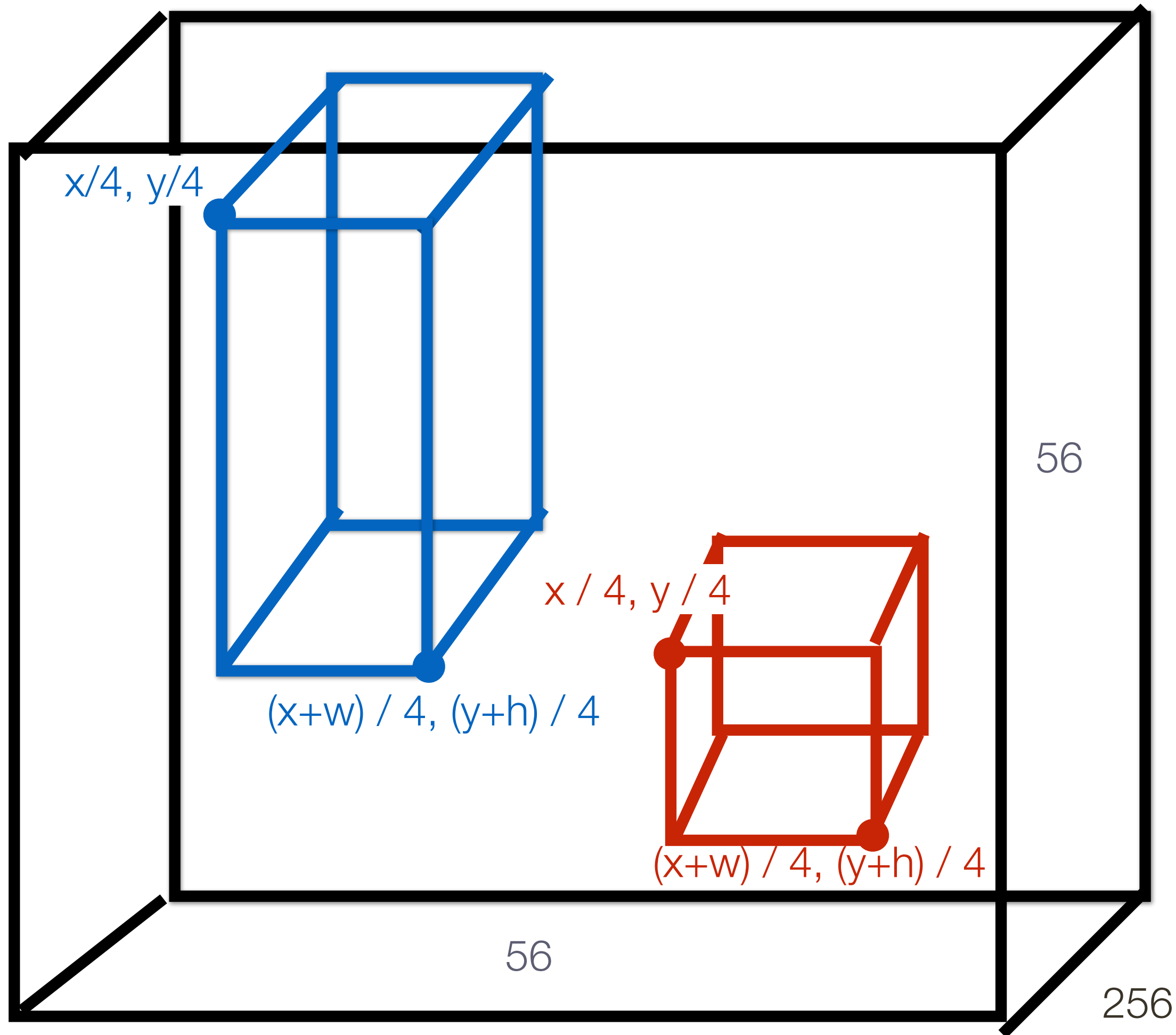
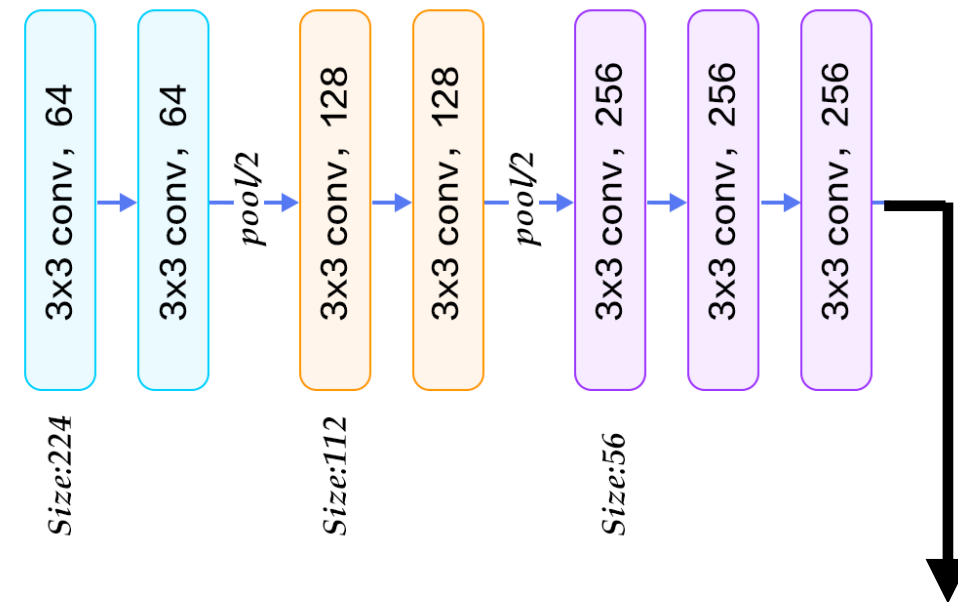
RoI Align

Image

(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)

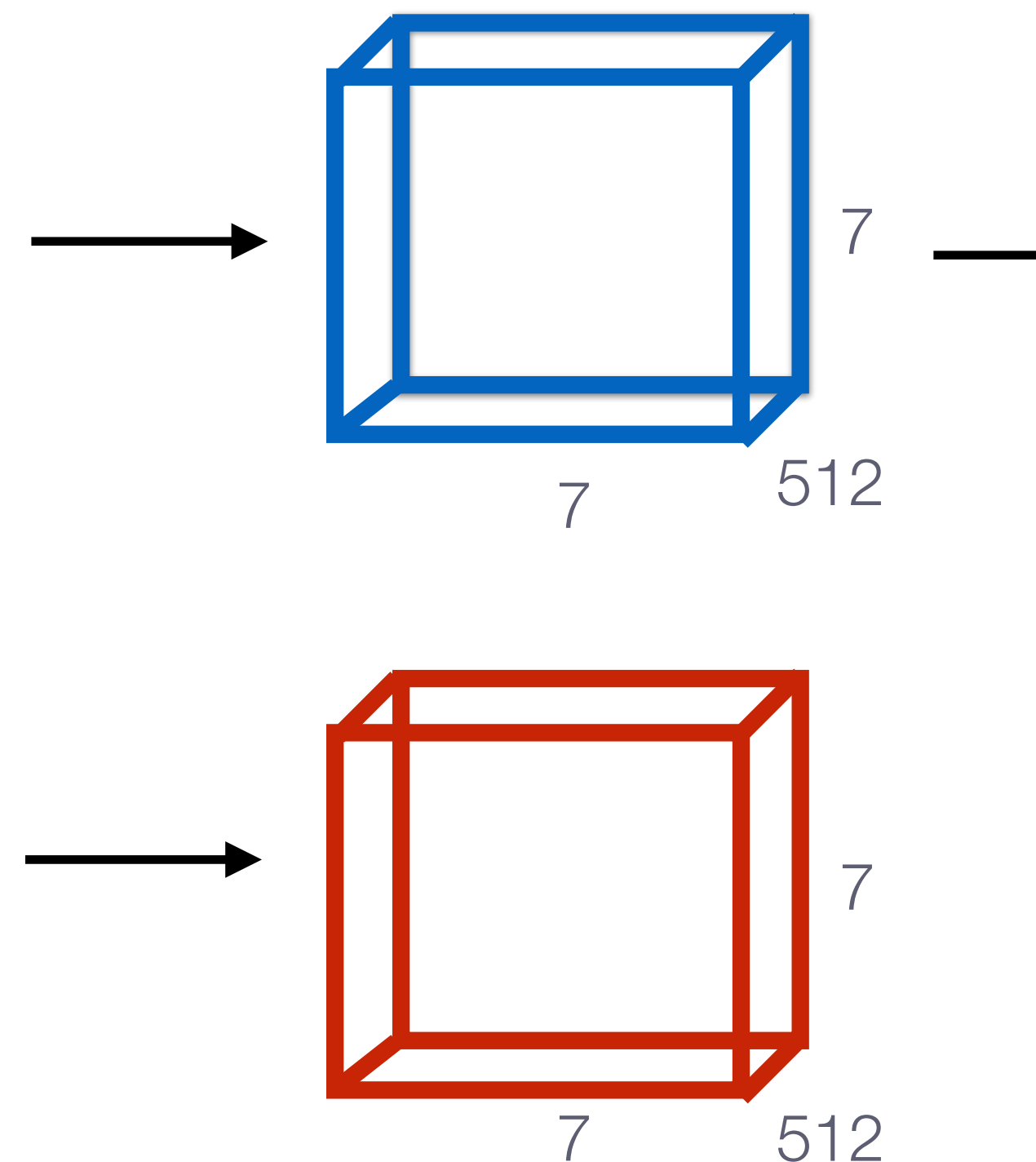


Classifier

(4096 x 80 categories + Softmax)

Regressor

(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h



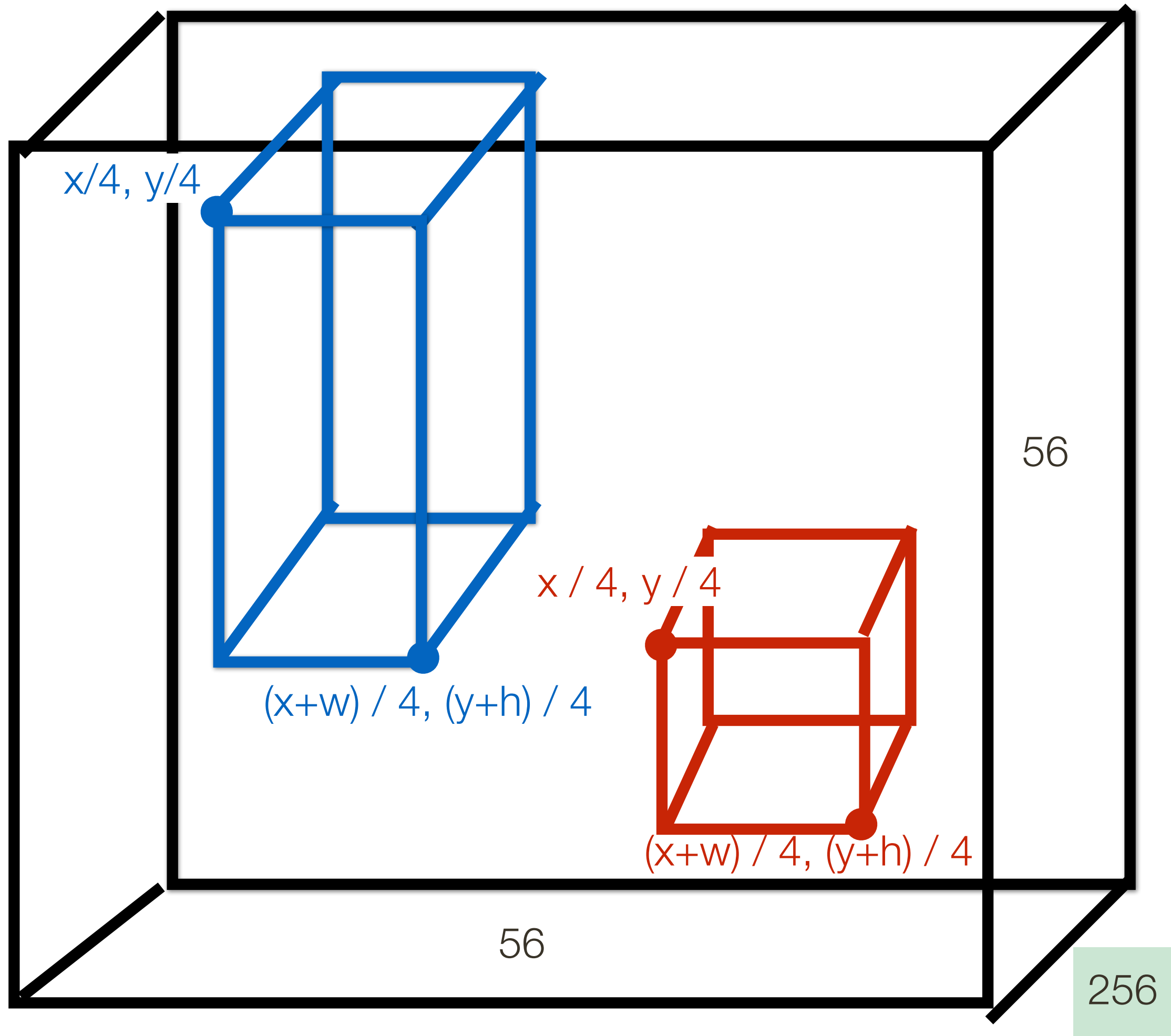
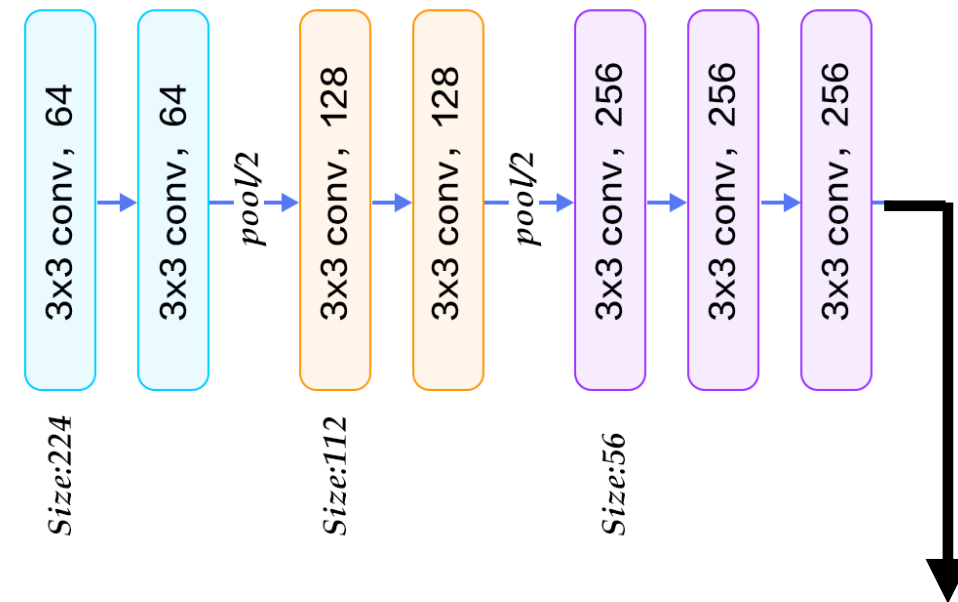
RoI Align

Image

(resized to 224 x 224)

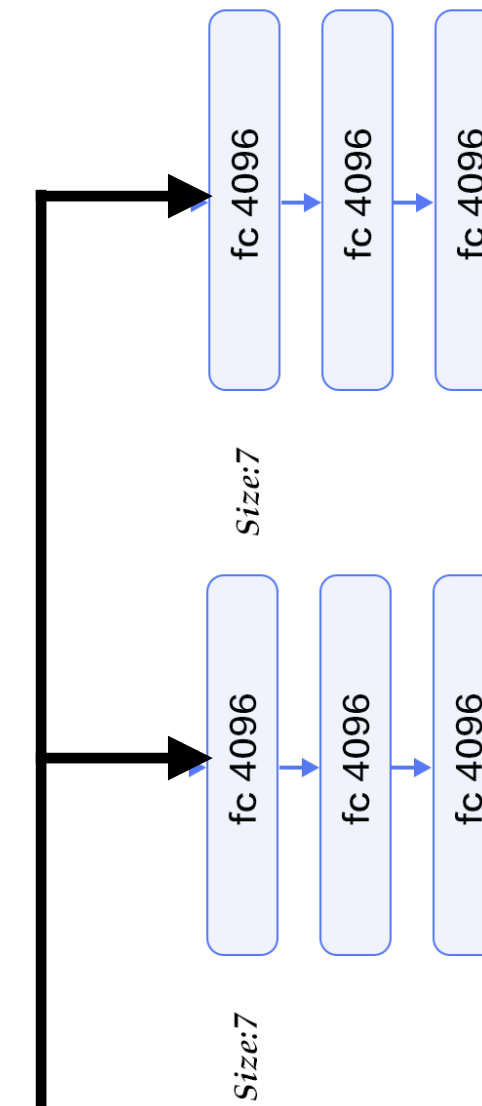
Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



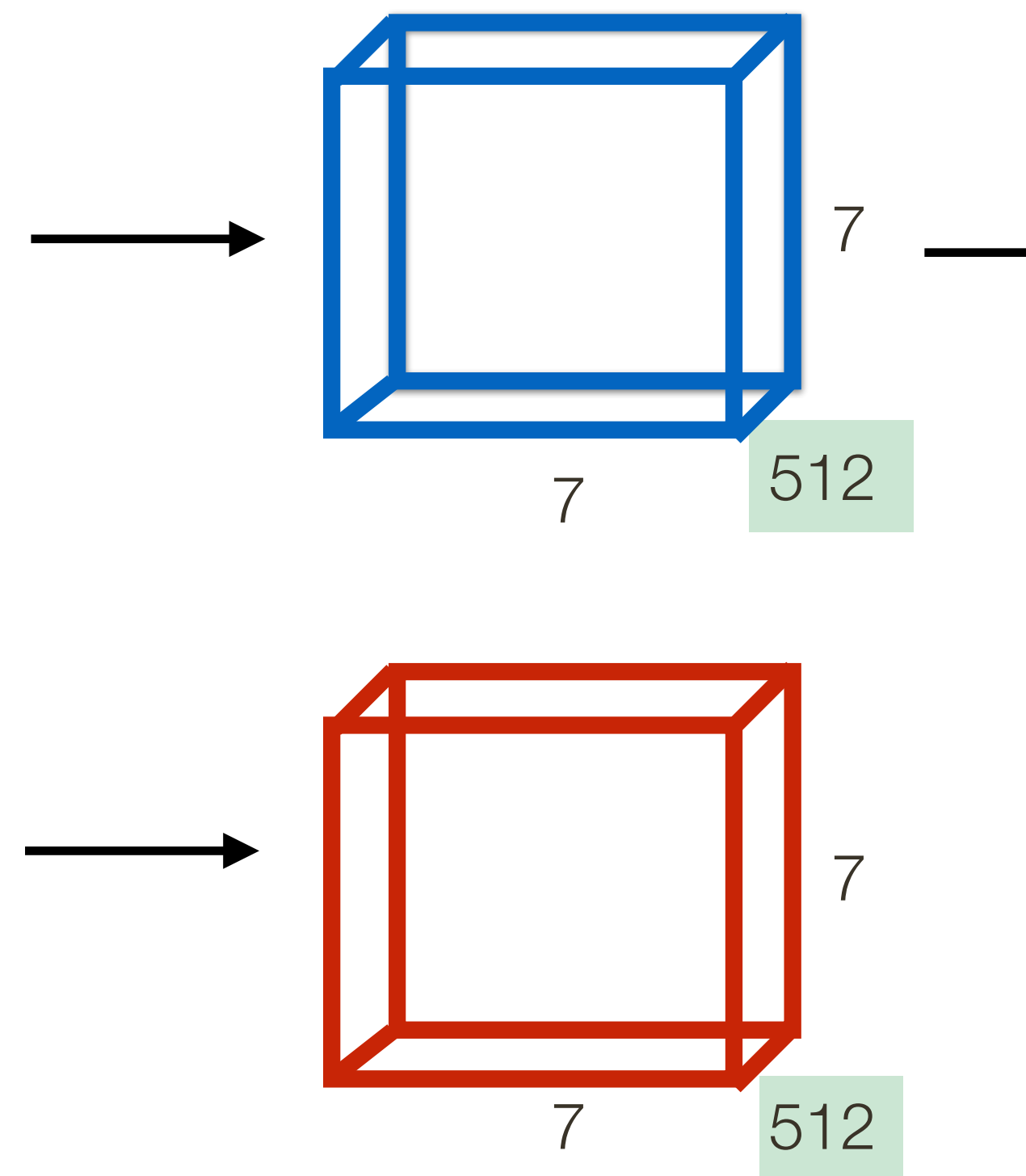
Classifier

(4096 x 80 categories + Softmax)



Regressor

(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h



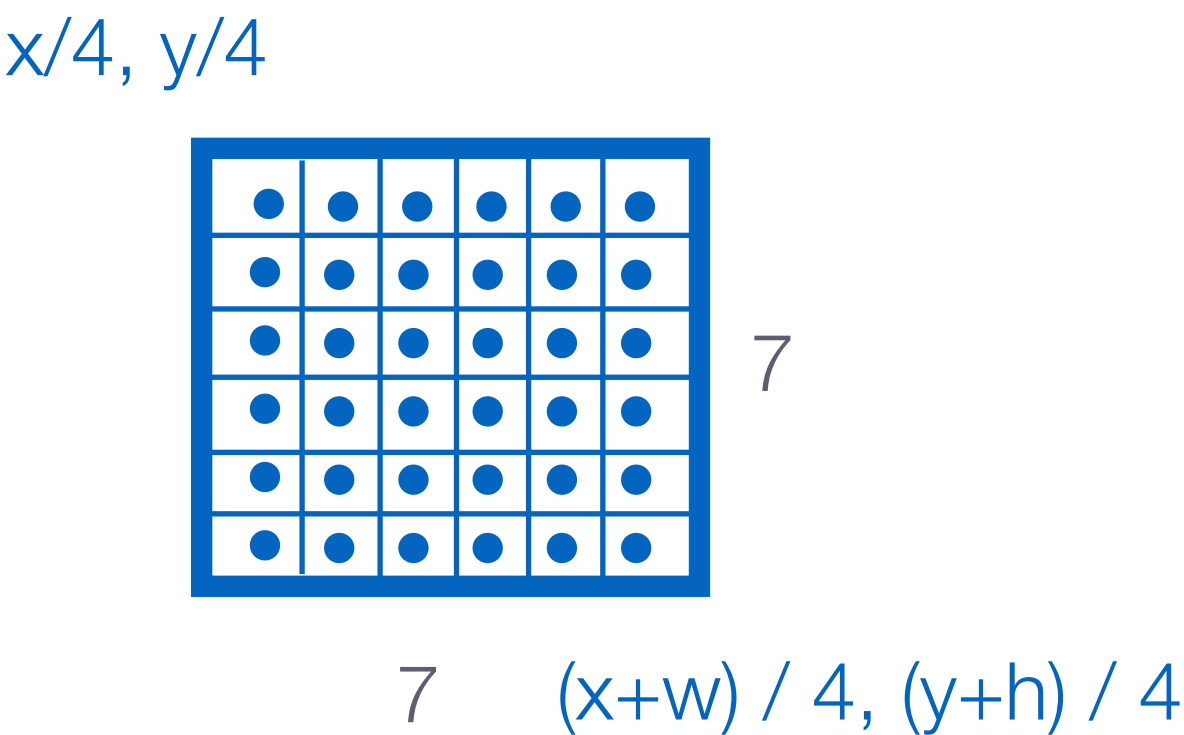
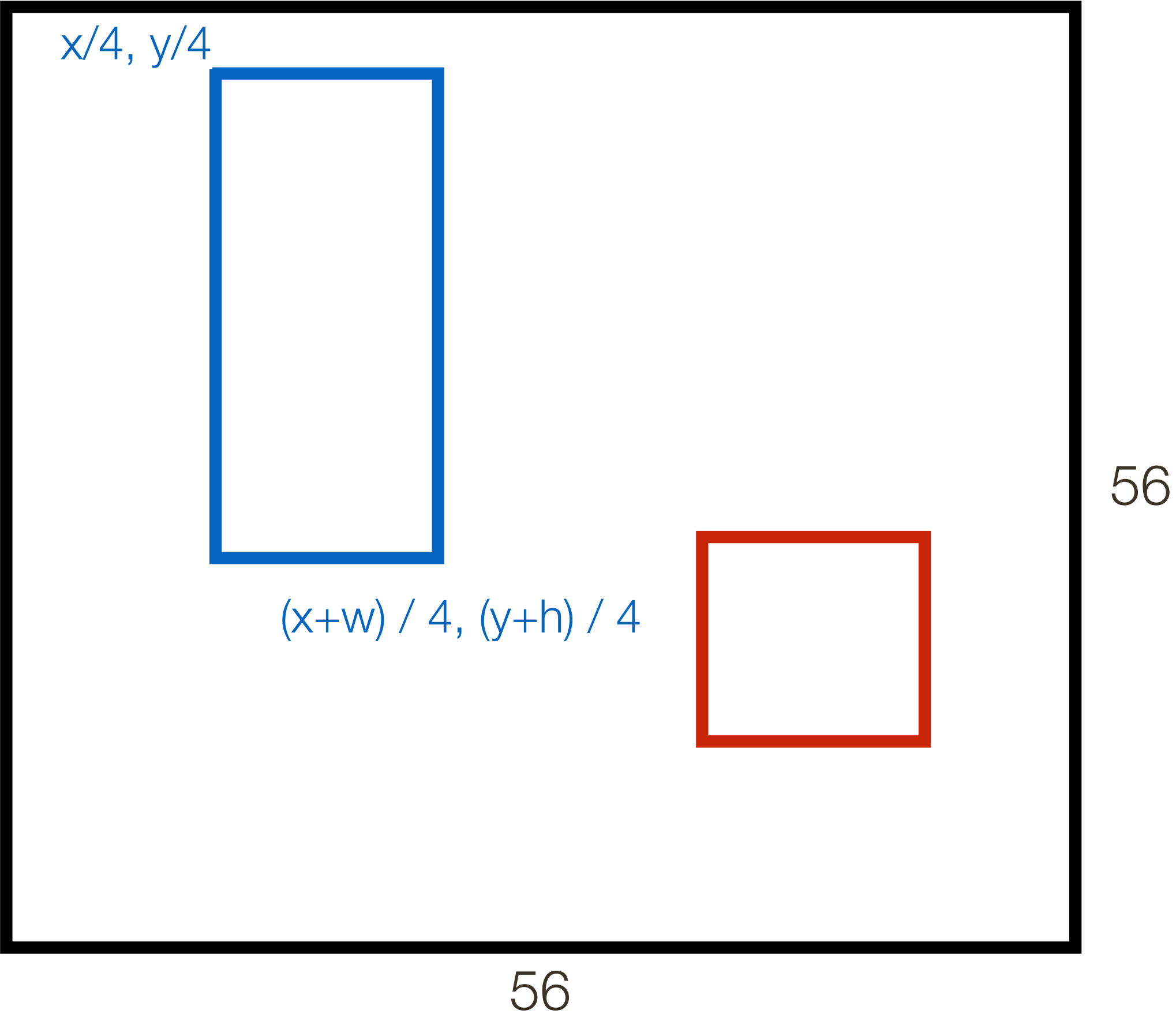
Change in depth /
channel dimension
is easy (1 x 1 conv)

RoI Align

Image
(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)

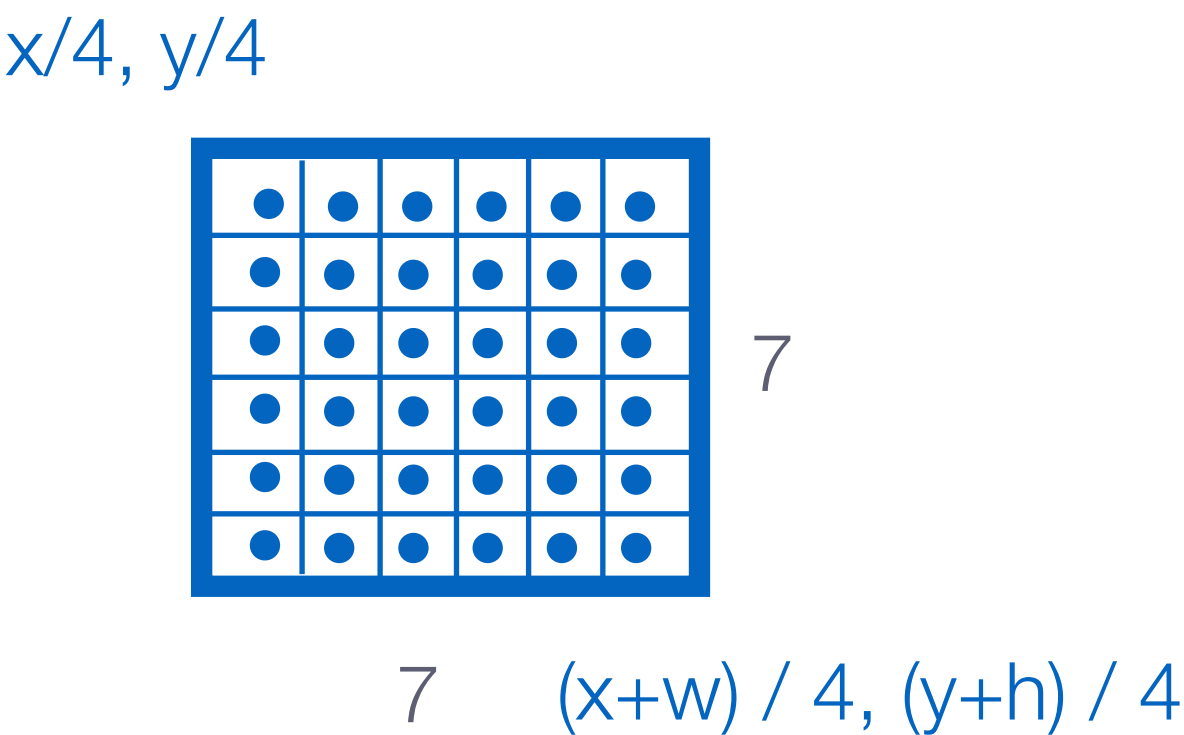
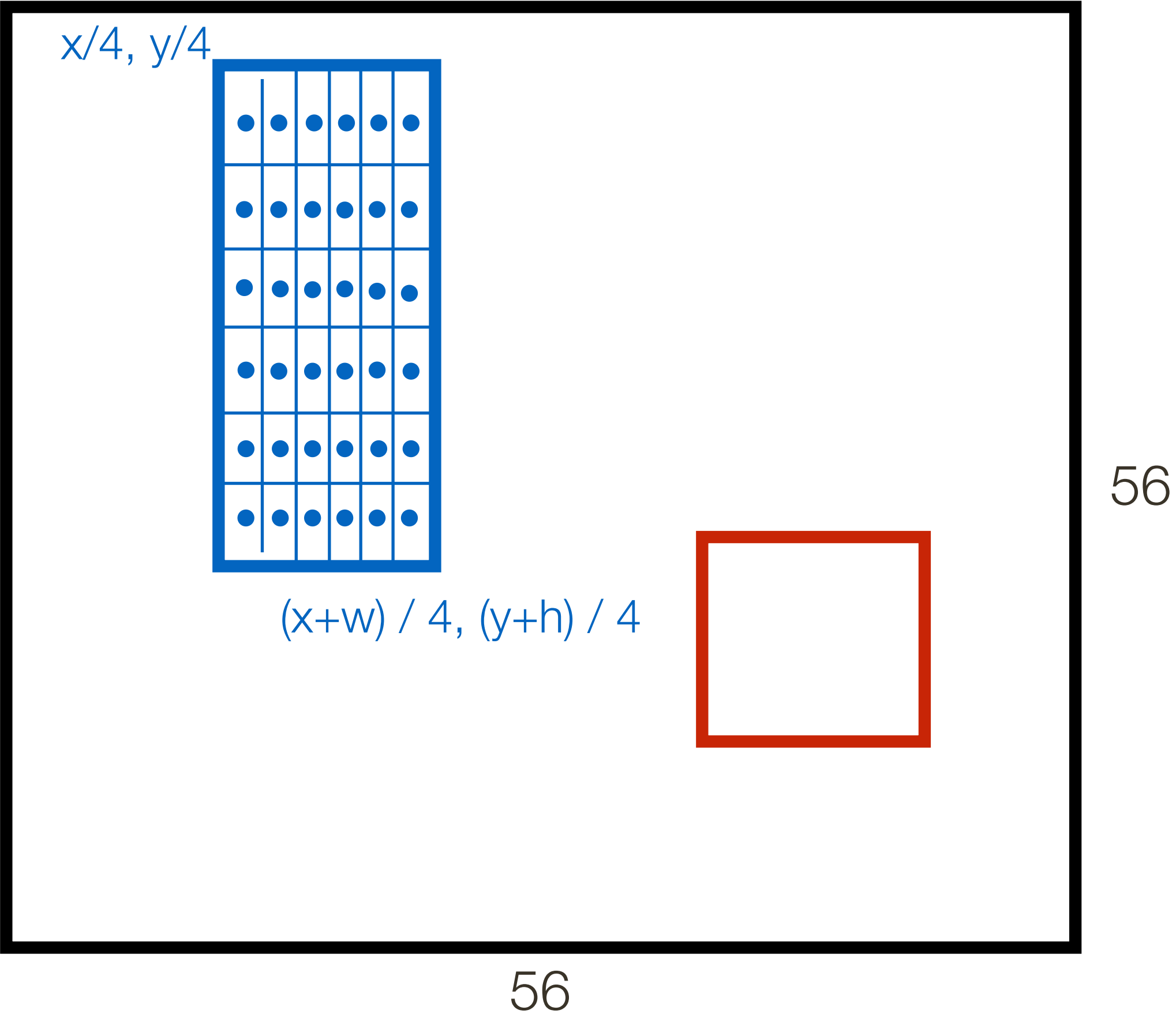


RoI Align

Image
(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



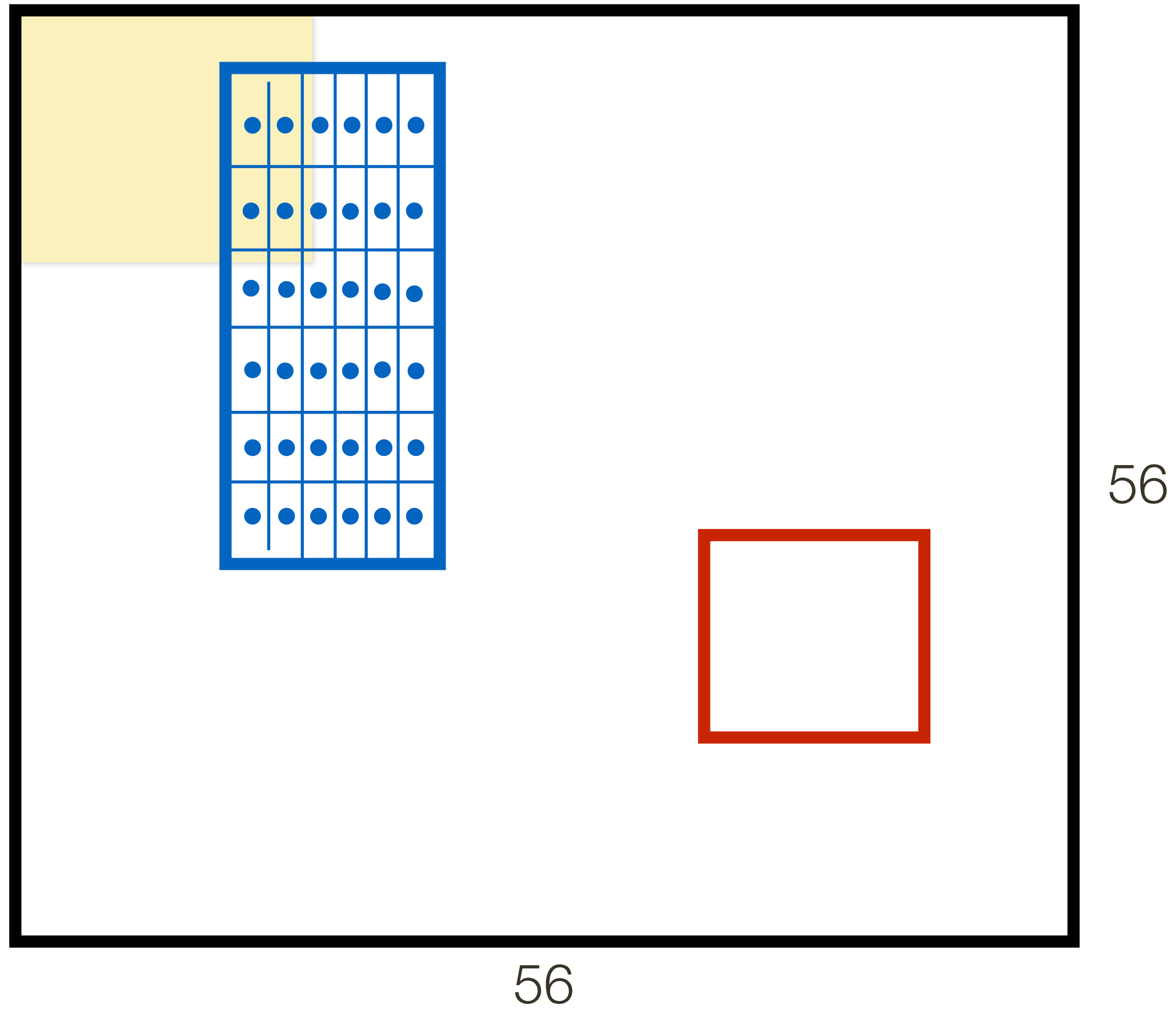
RoI Align

Image

(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)

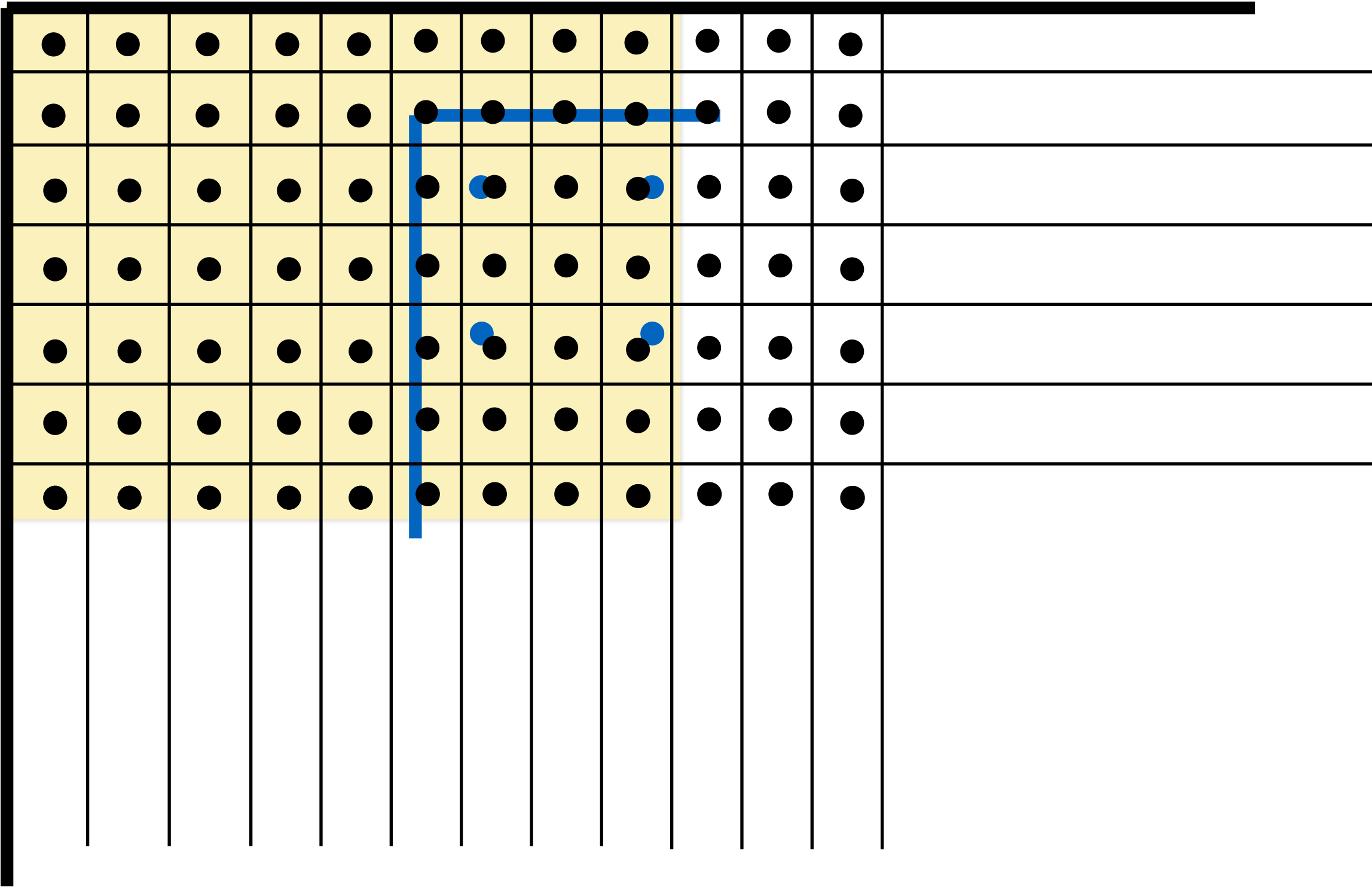
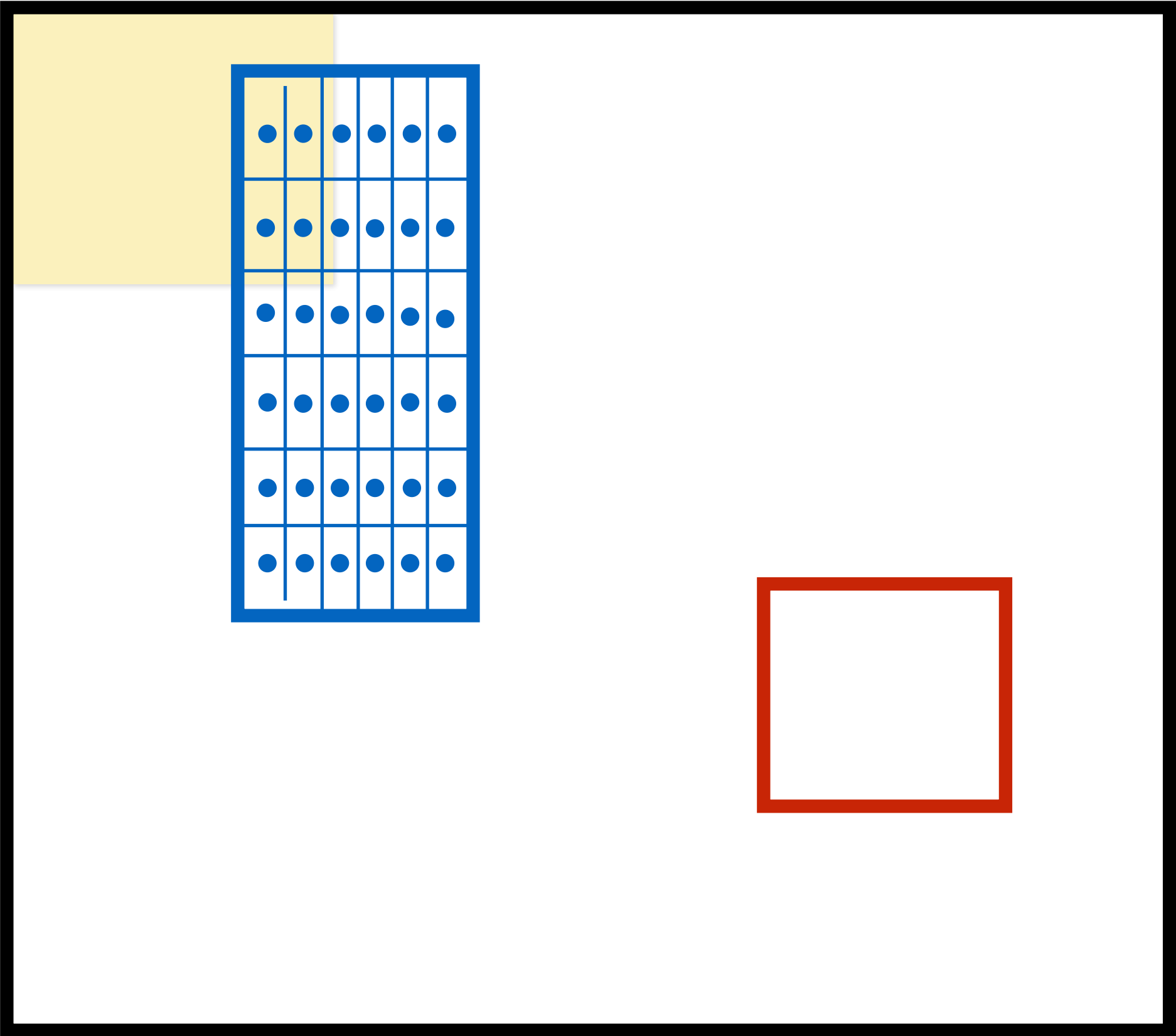


RoI Align

Image
(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



56

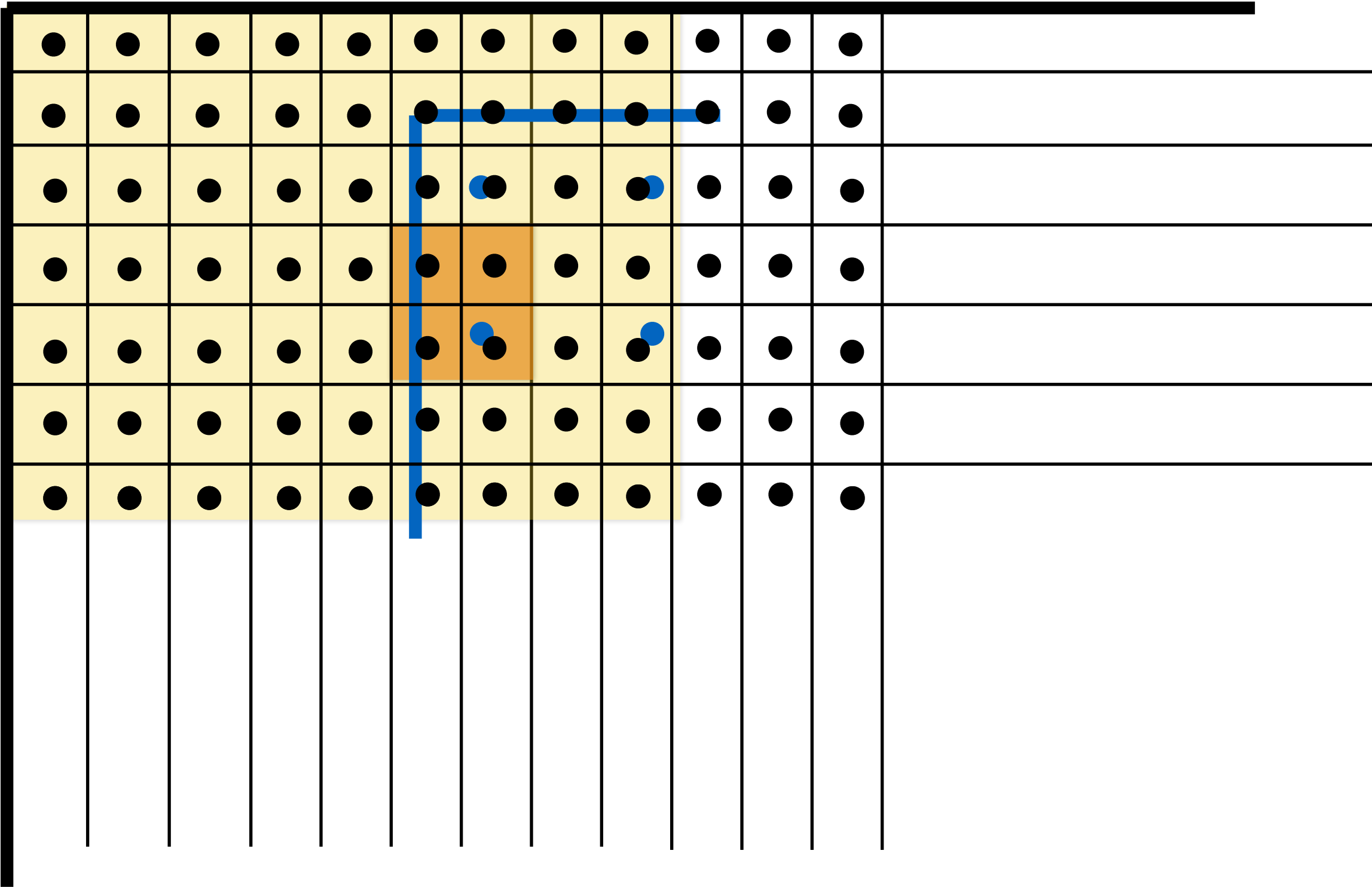
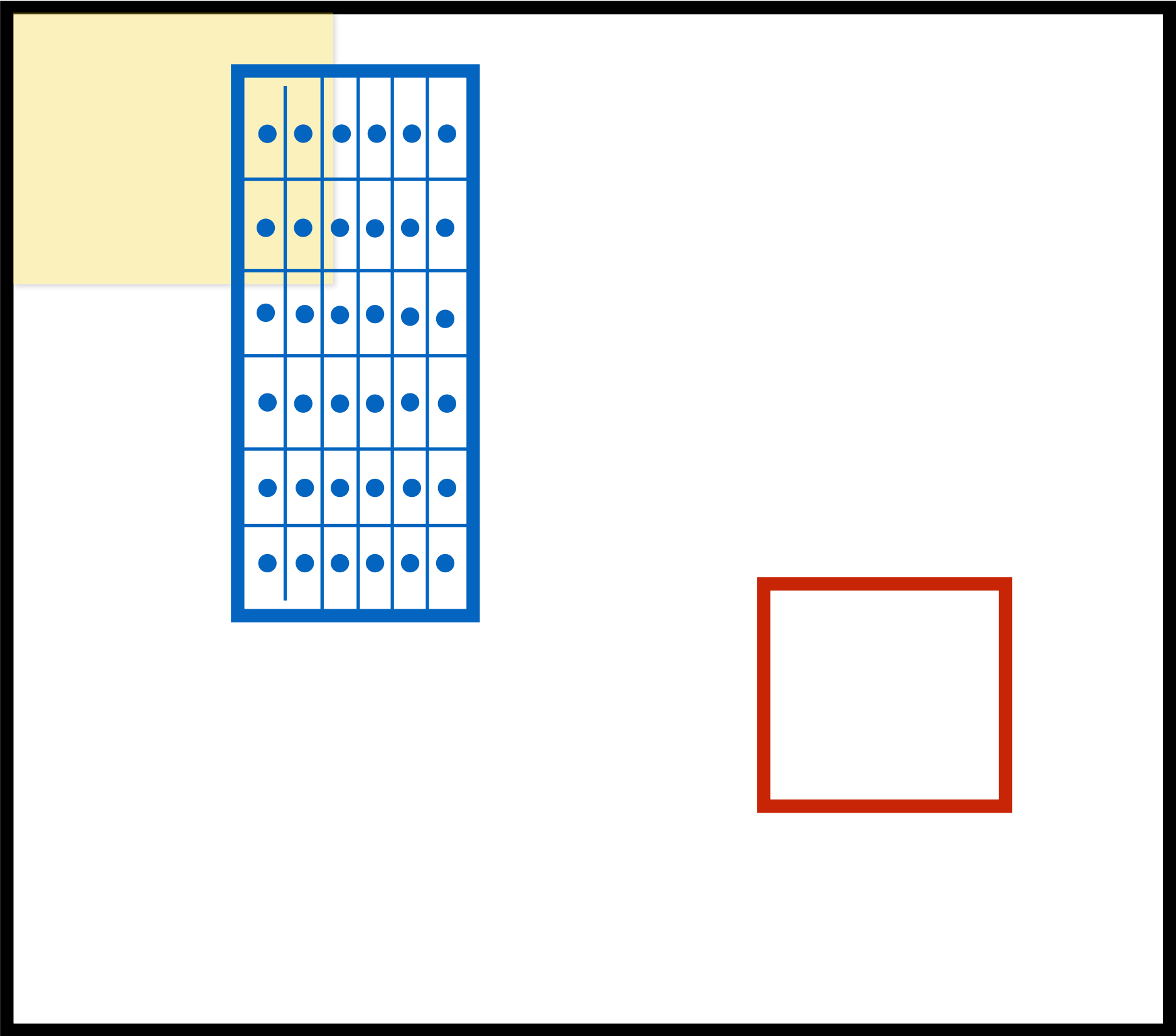
56

RoI Align

Image
(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



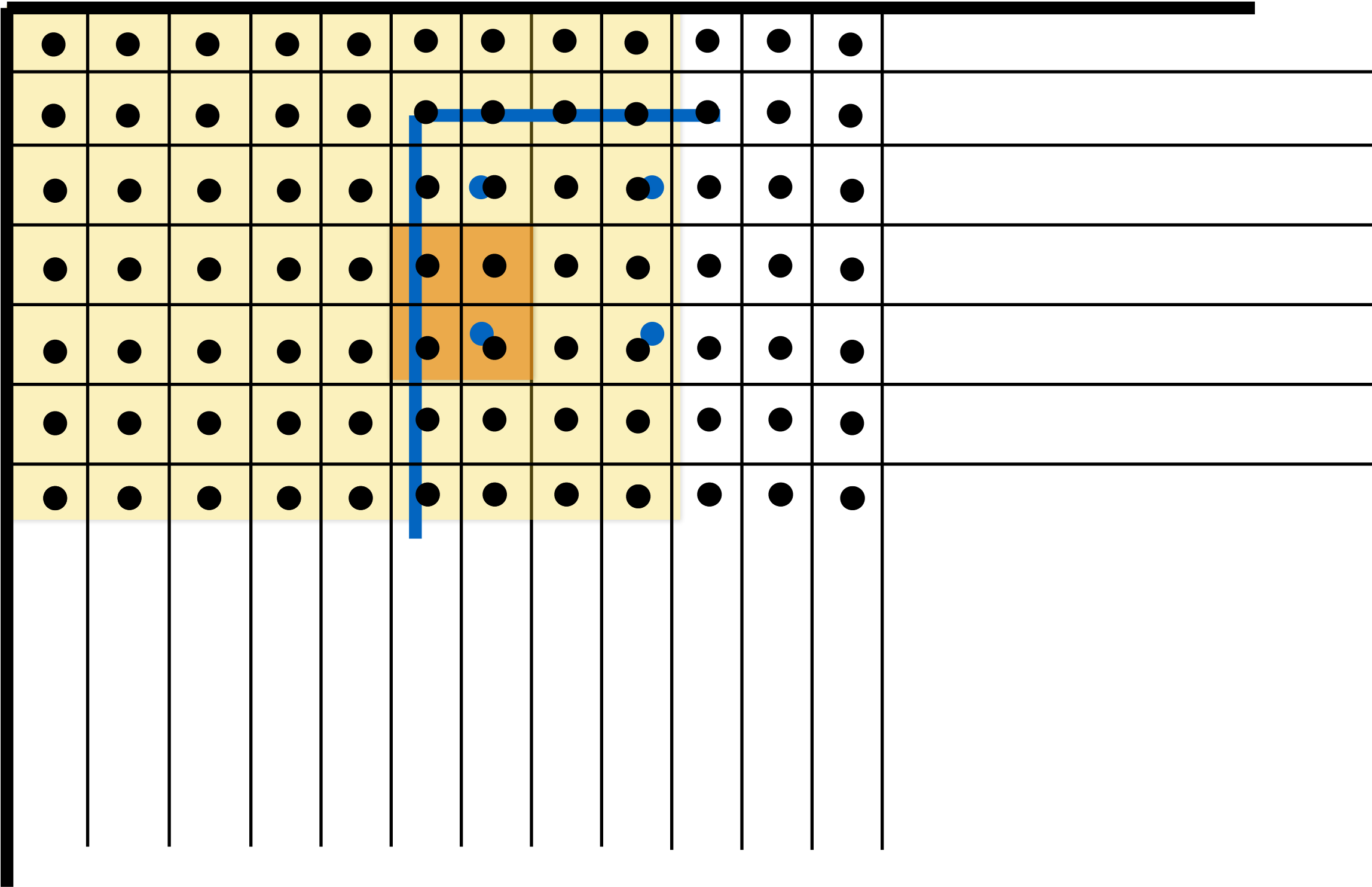
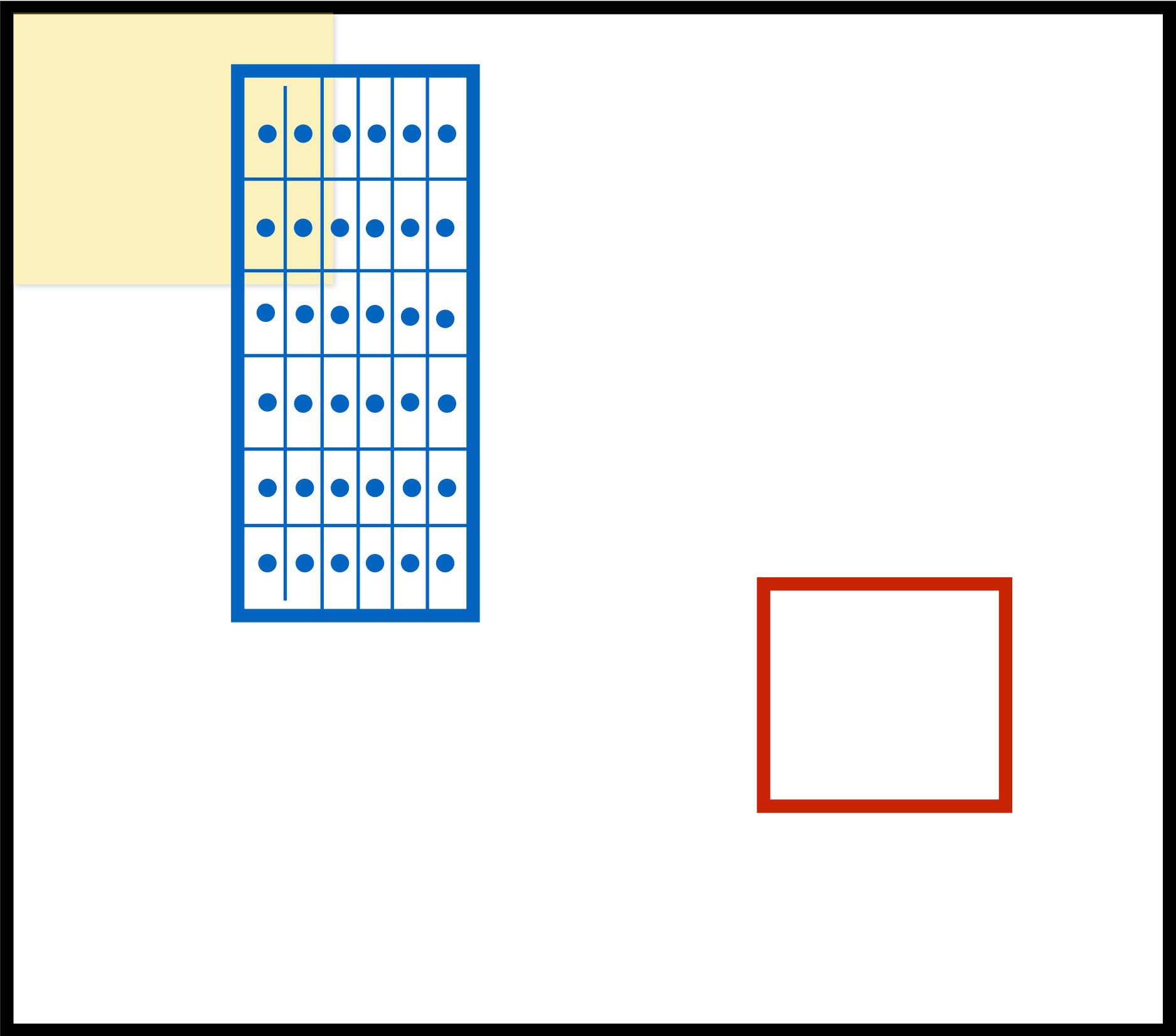
56

RoI Align

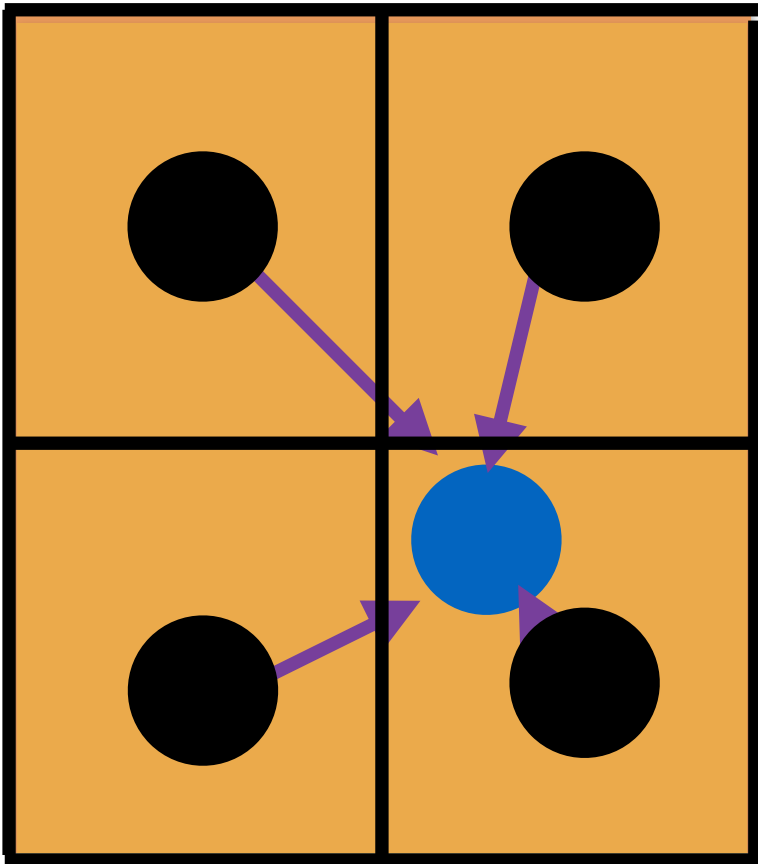
Image
(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



56



56

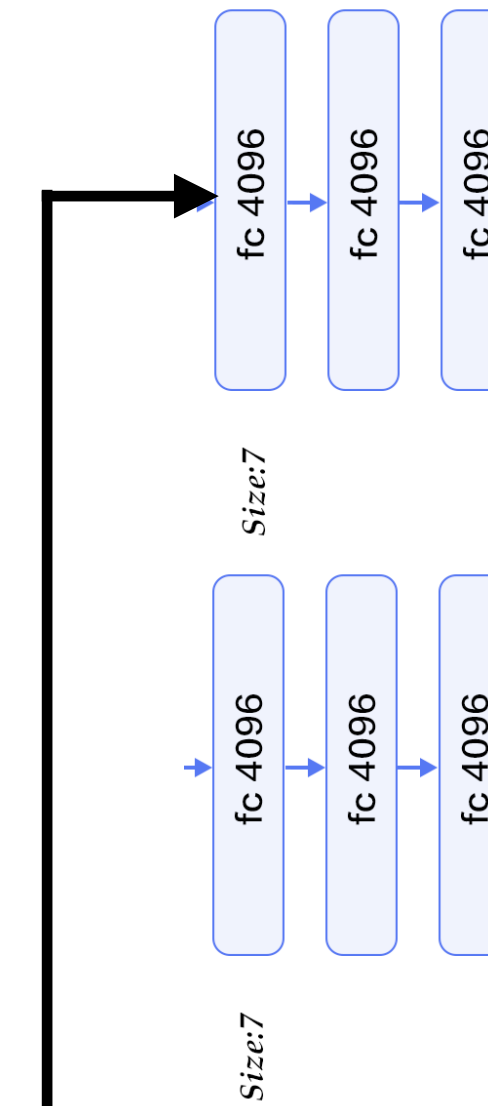
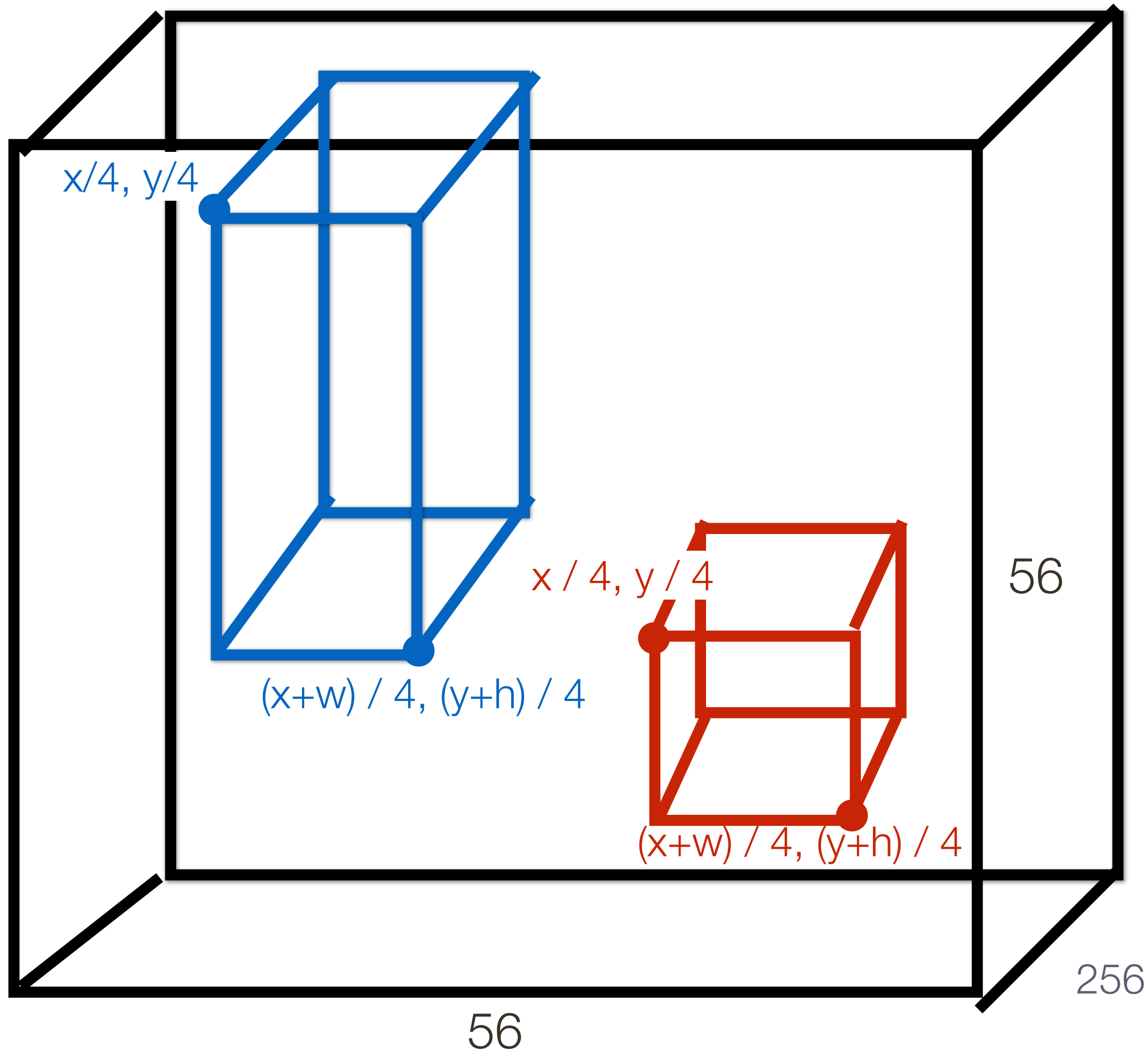
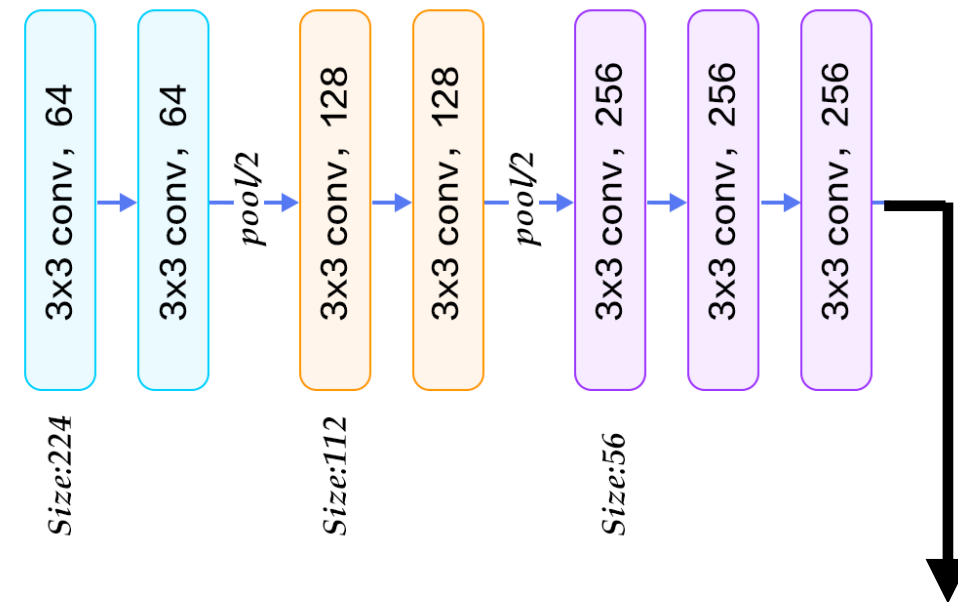
RoI Align

Image

(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)

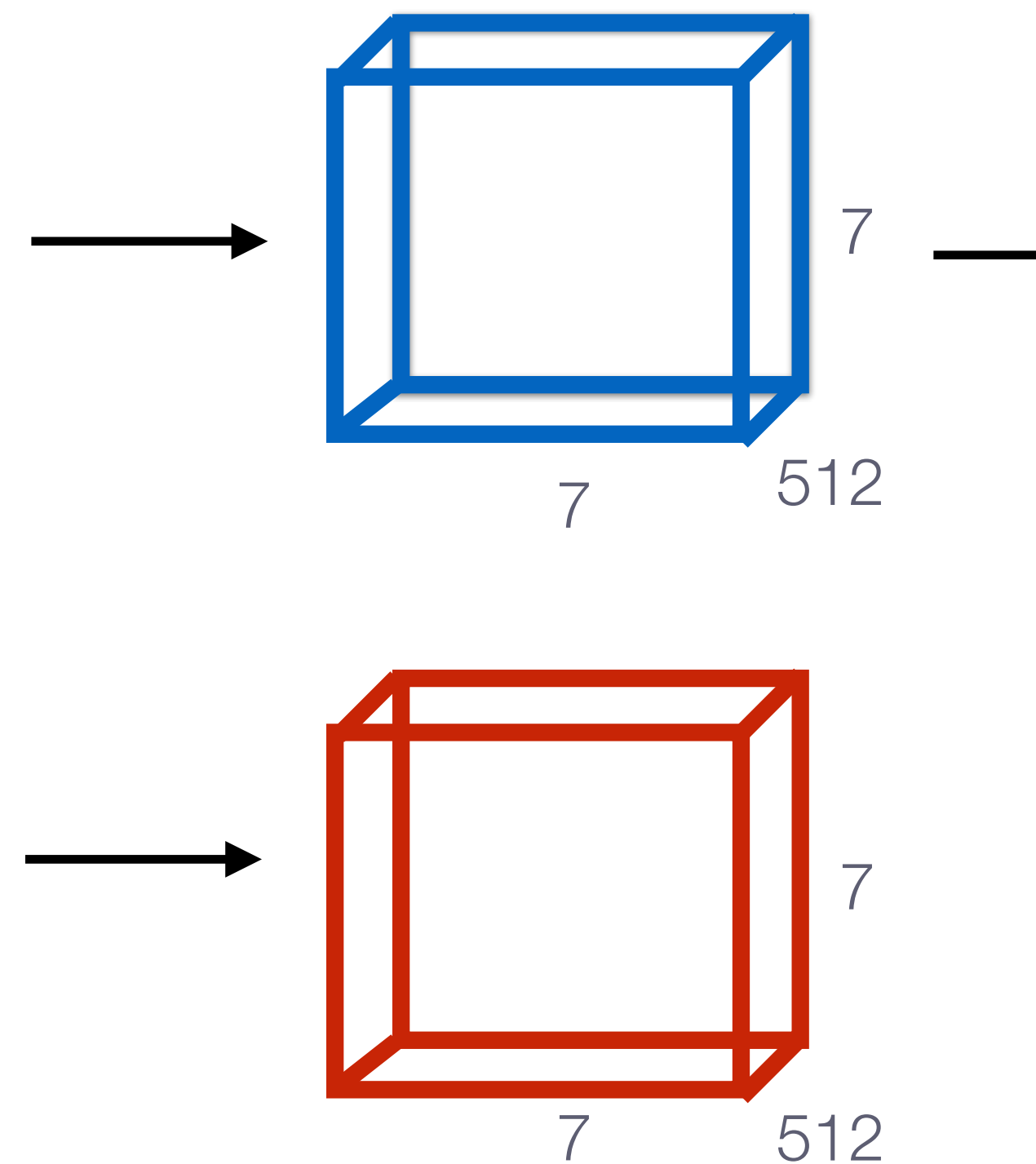


Classifier

(4096 x 80 categories + Softmax)

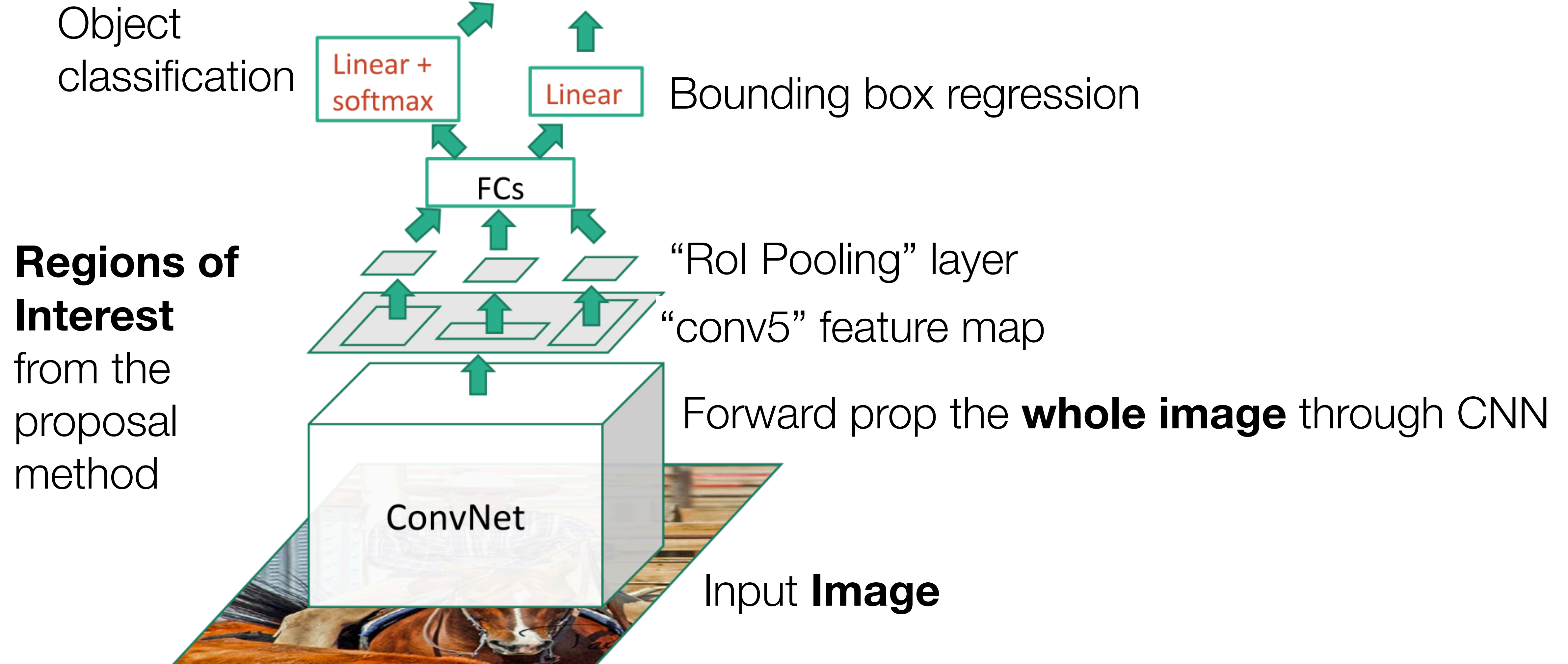
Regressor

(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h



Fast R-CNN

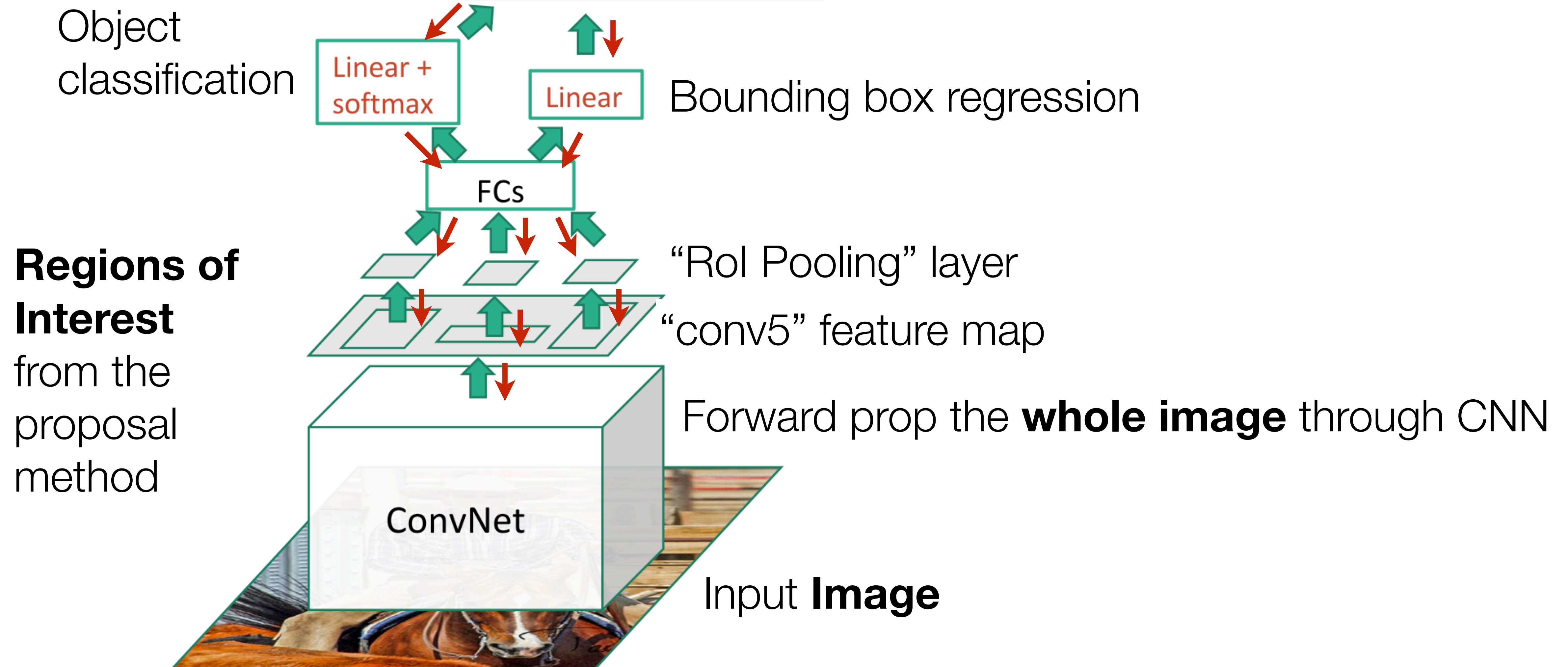
[Girshick et al, ICCV 2015]



* image from Ross Girshick

Fast **R-CNN**: Training

[Girshick et al, ICCV 2015]



* image from Ross Girshick

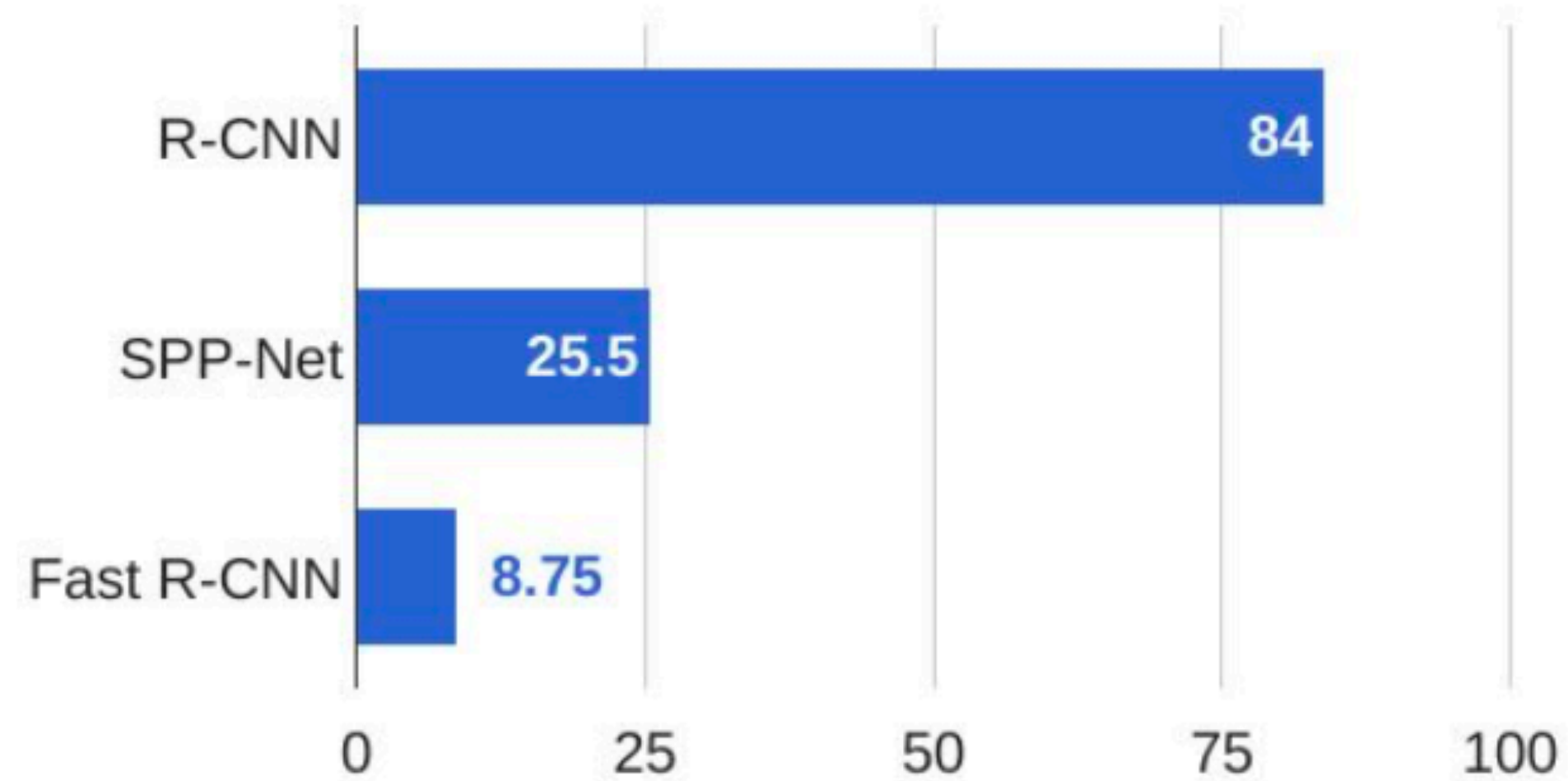
R-CNN vs. SPP vs. Fast R-CNN

[Girshick et al, CVPR 2014]

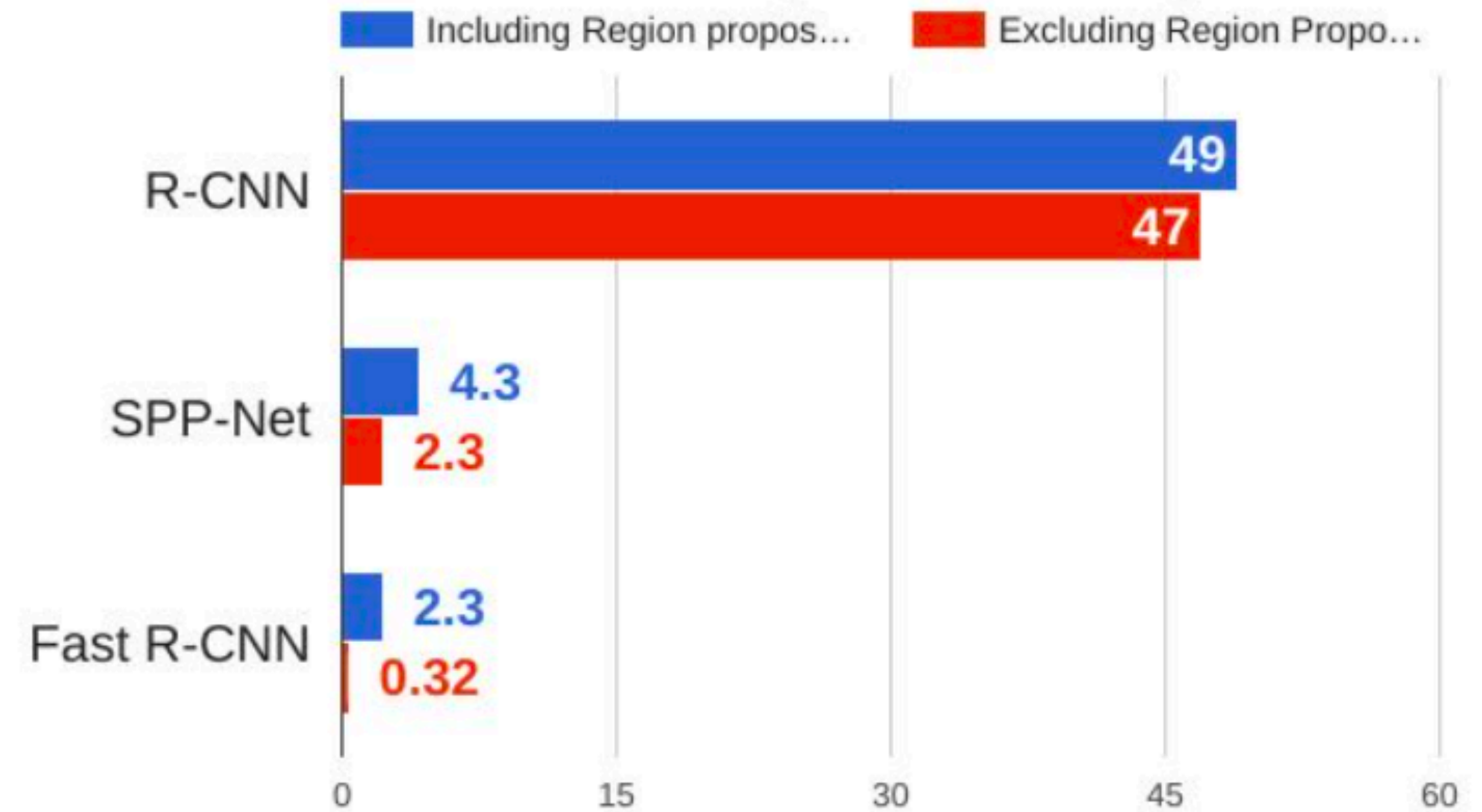
[Girshick et al, ICCV 2015]

[He et al, ECCV 2014]

Training time (Hours)



Test time (seconds)

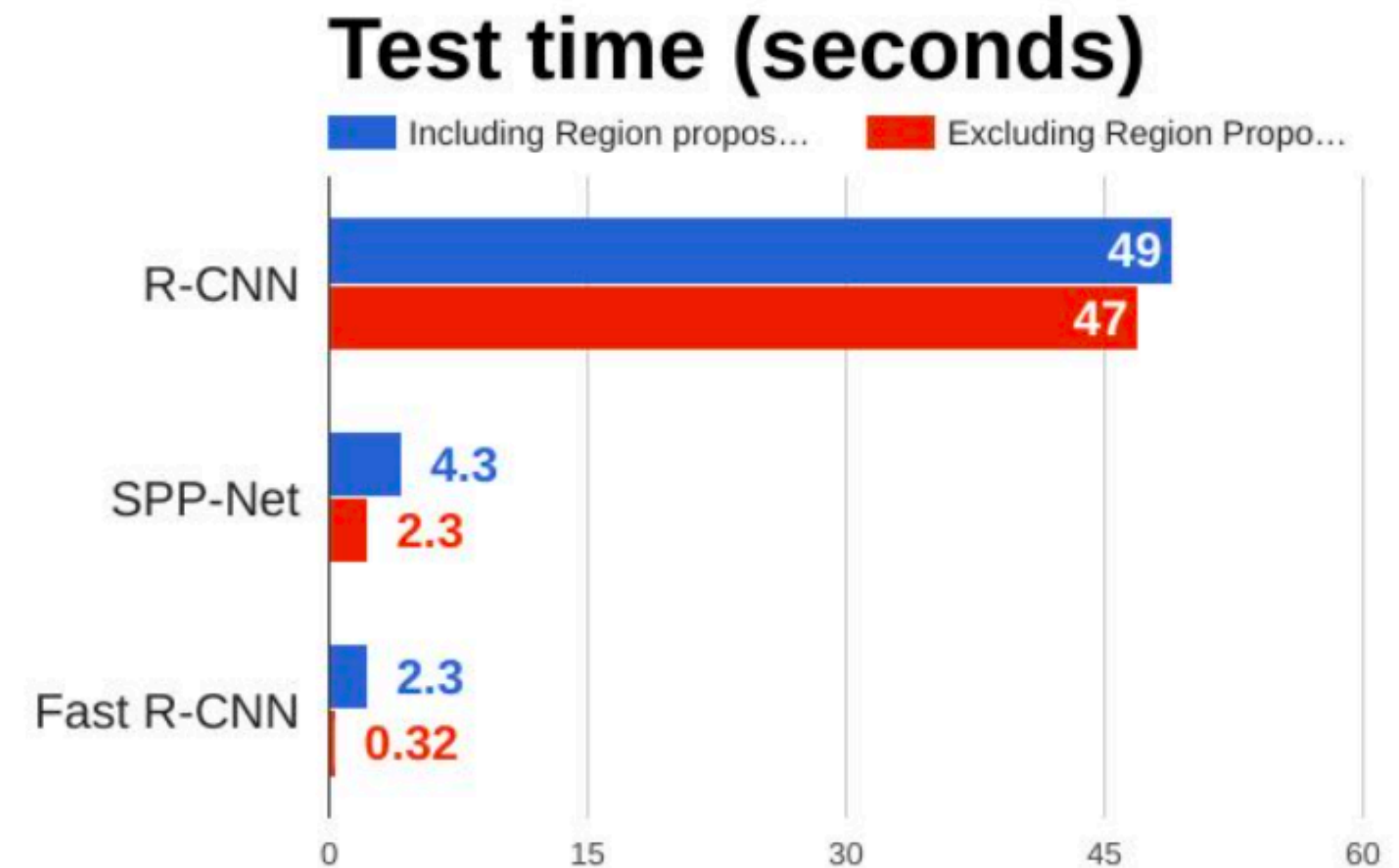
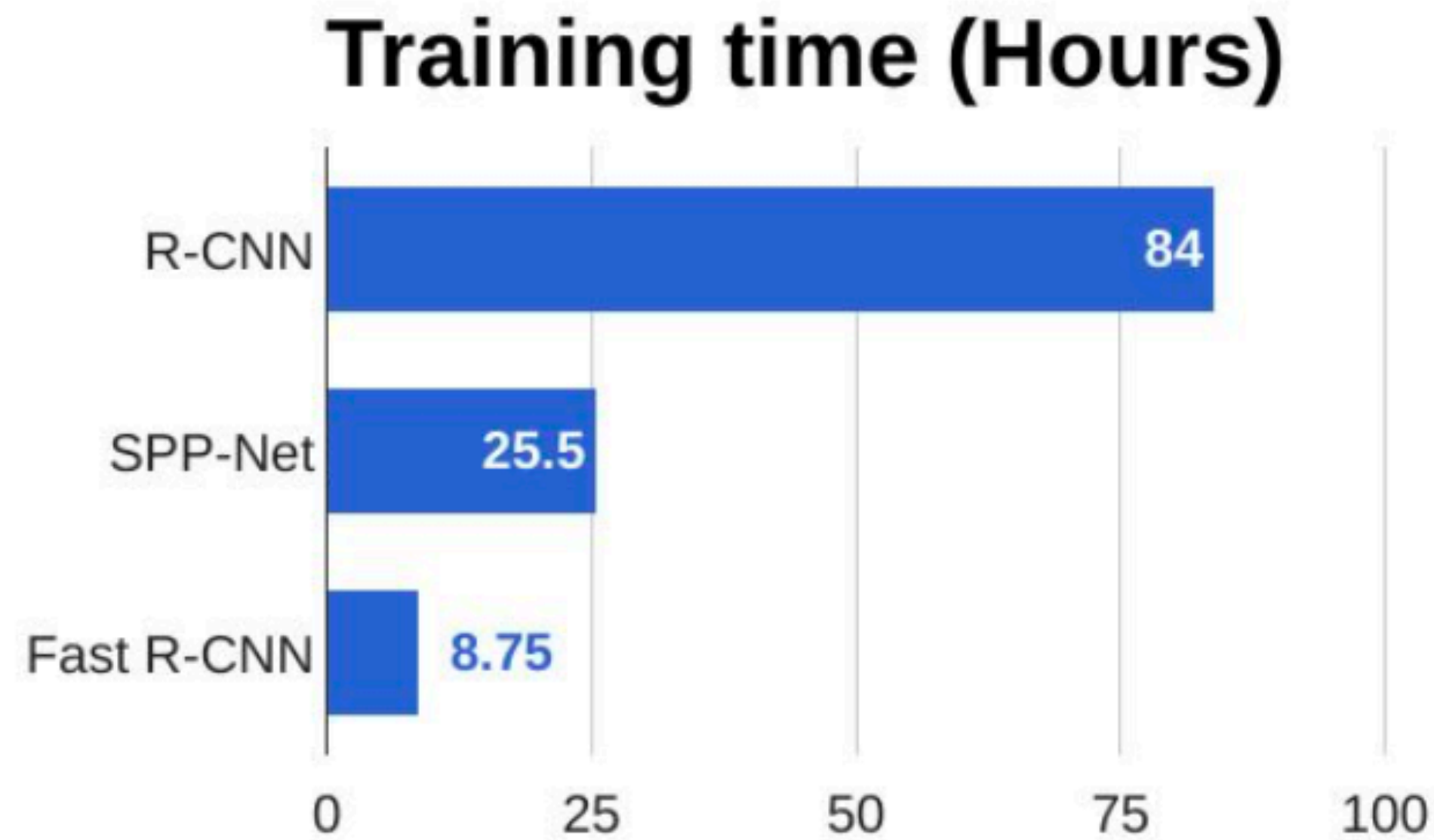


R-CNN vs. SPP vs. Fast R-CNN

[Girshick et al, CVPR 2014]

[Girshick et al, ICCV 2015]

[He et al, ECCV 2014]



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

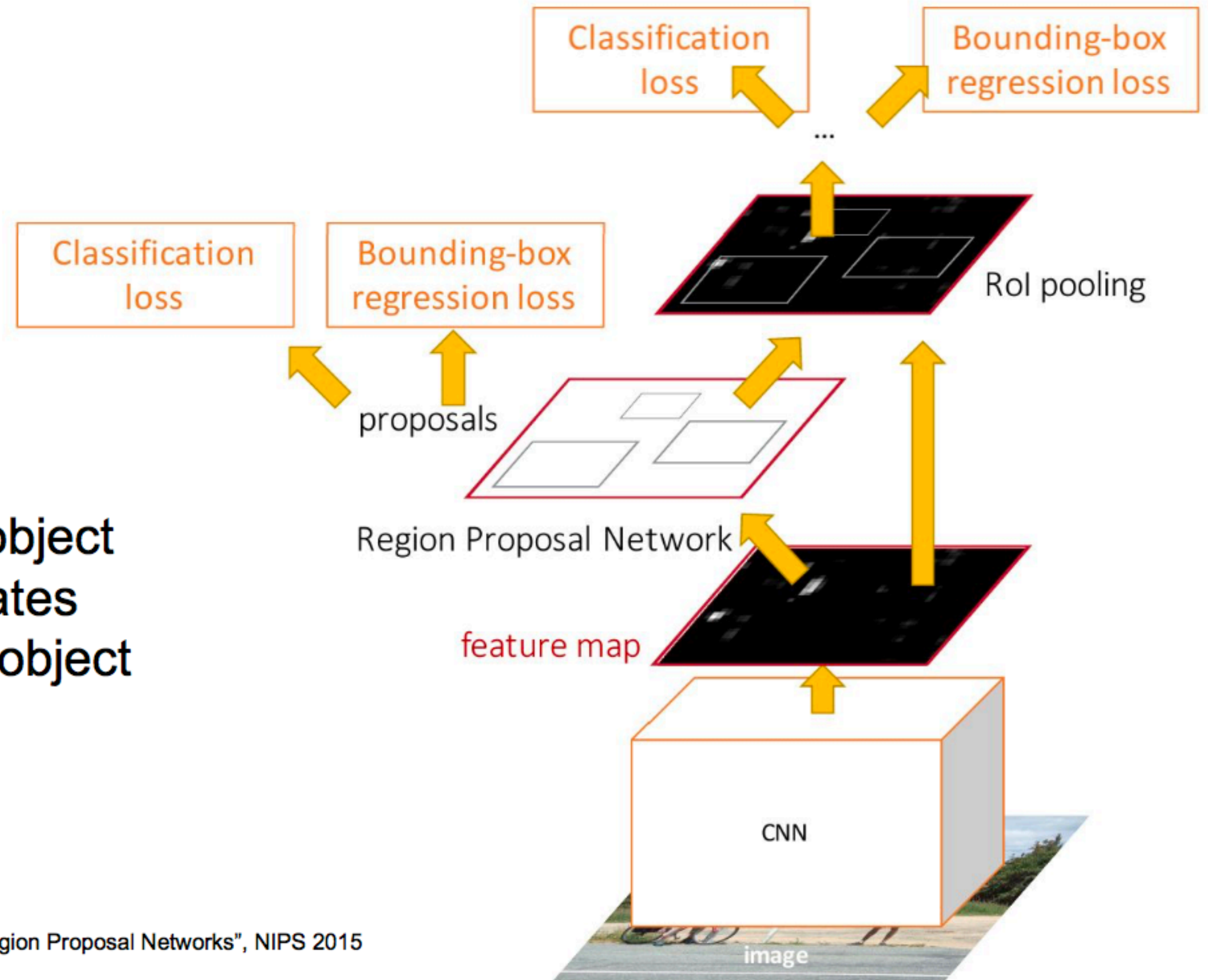
Faster R-CNN

Make CNN do proposals!

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

Jointly train with 4 losses:

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

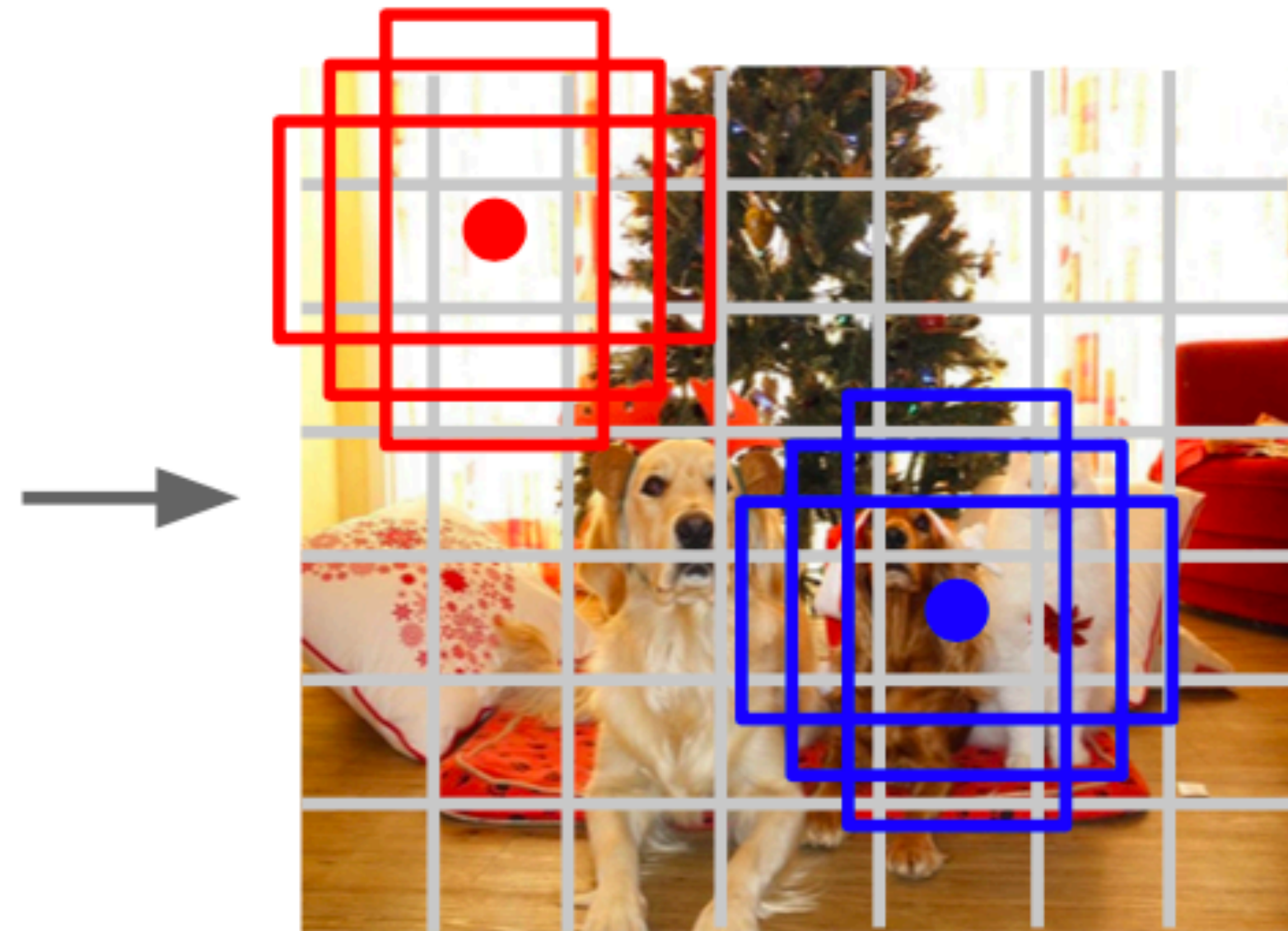


YOLO: You Only Look Once

[Redmon et al, CVPR 2016]



Input image
 $3 \times H \times W$



Divide image into grid
 7×7

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

Within each grid cell:

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

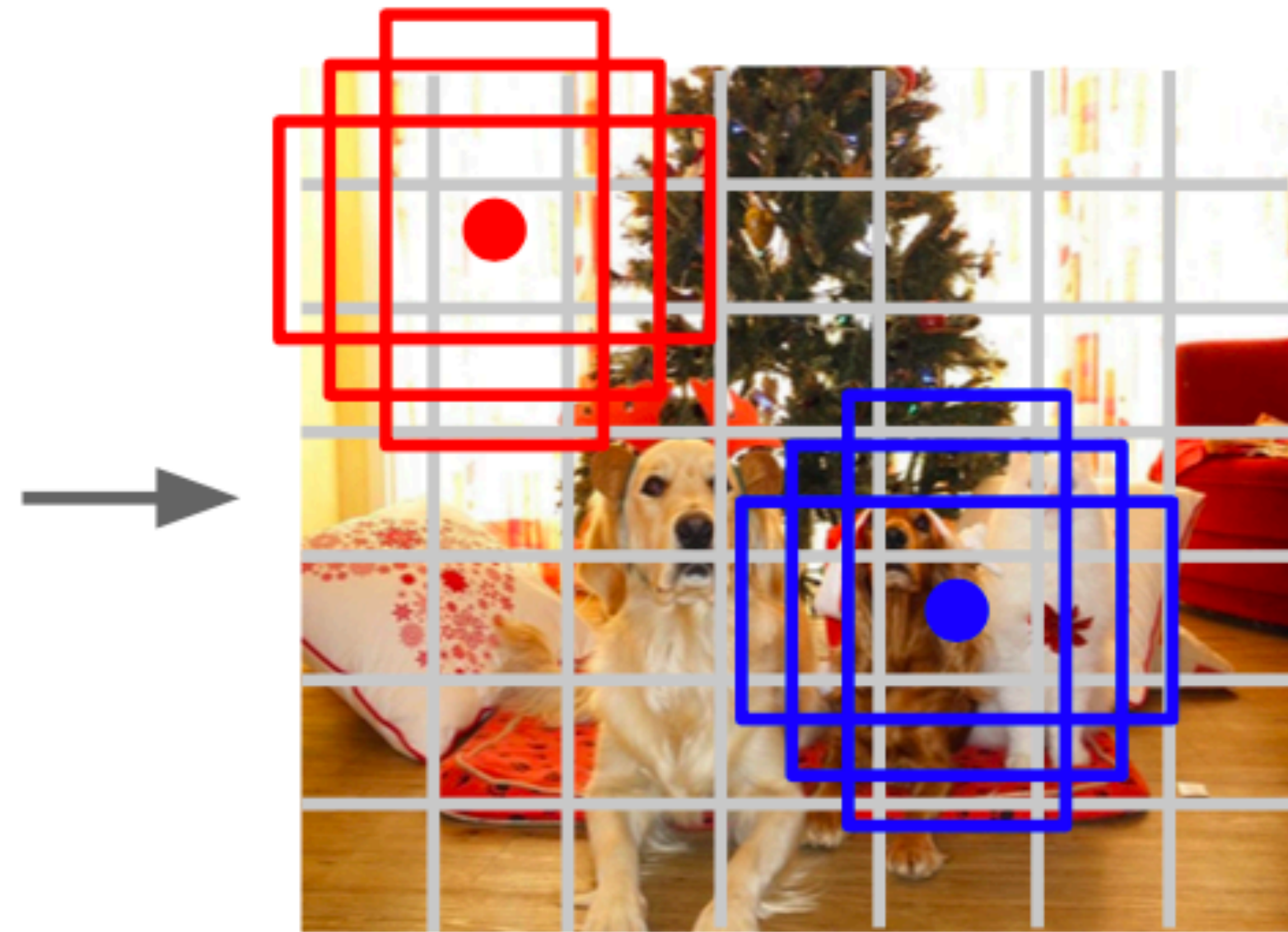
Output:
 $7 \times 7 \times (5 * B + C)$

YOLO: You Only Look Once

[Redmon et al, CVPR 2016]

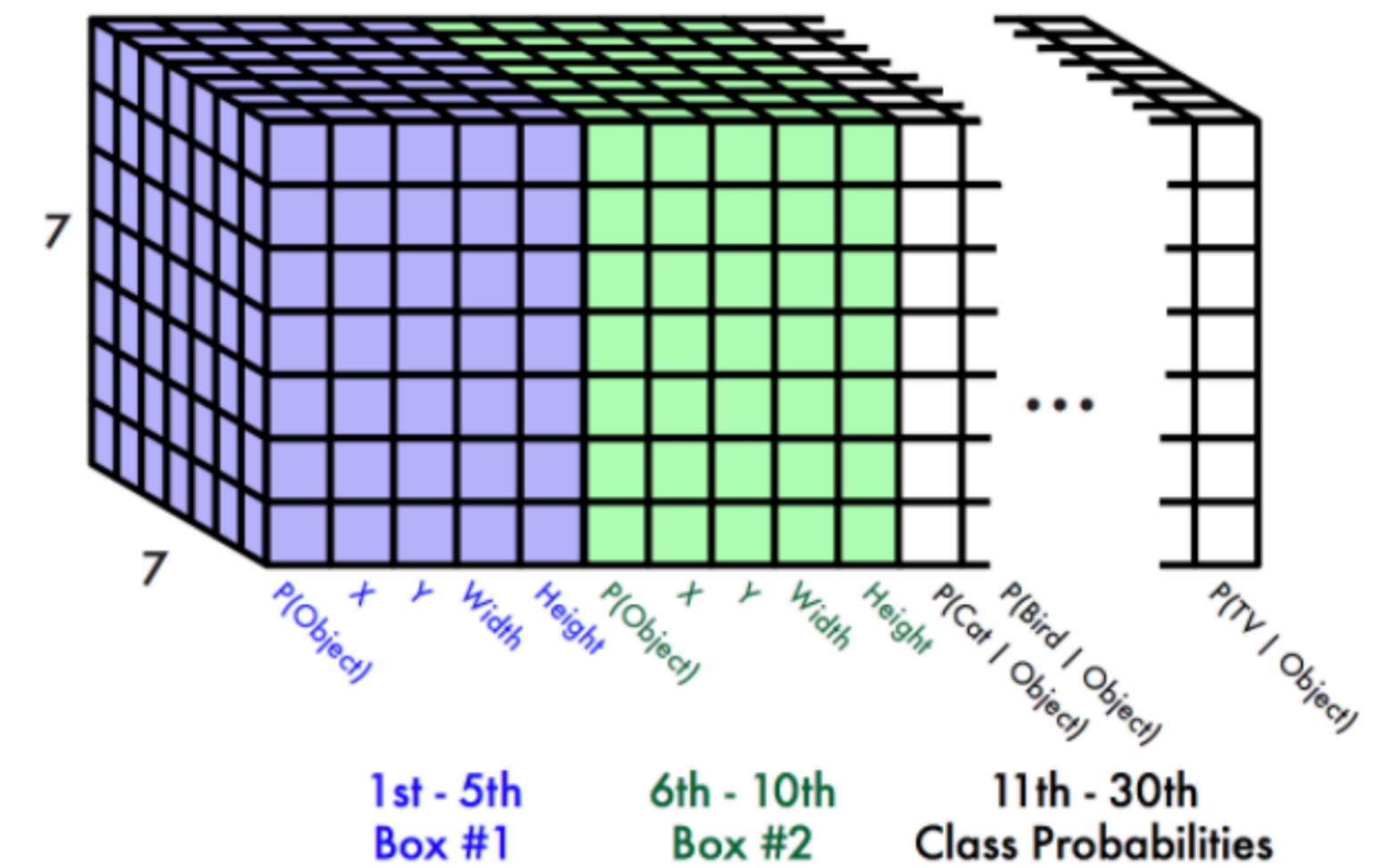


Input image
 $3 \times H \times W$



Divide image into grid
 7×7

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





YOLO v2

<http://pureddie.com/yolo>



YOLO v2

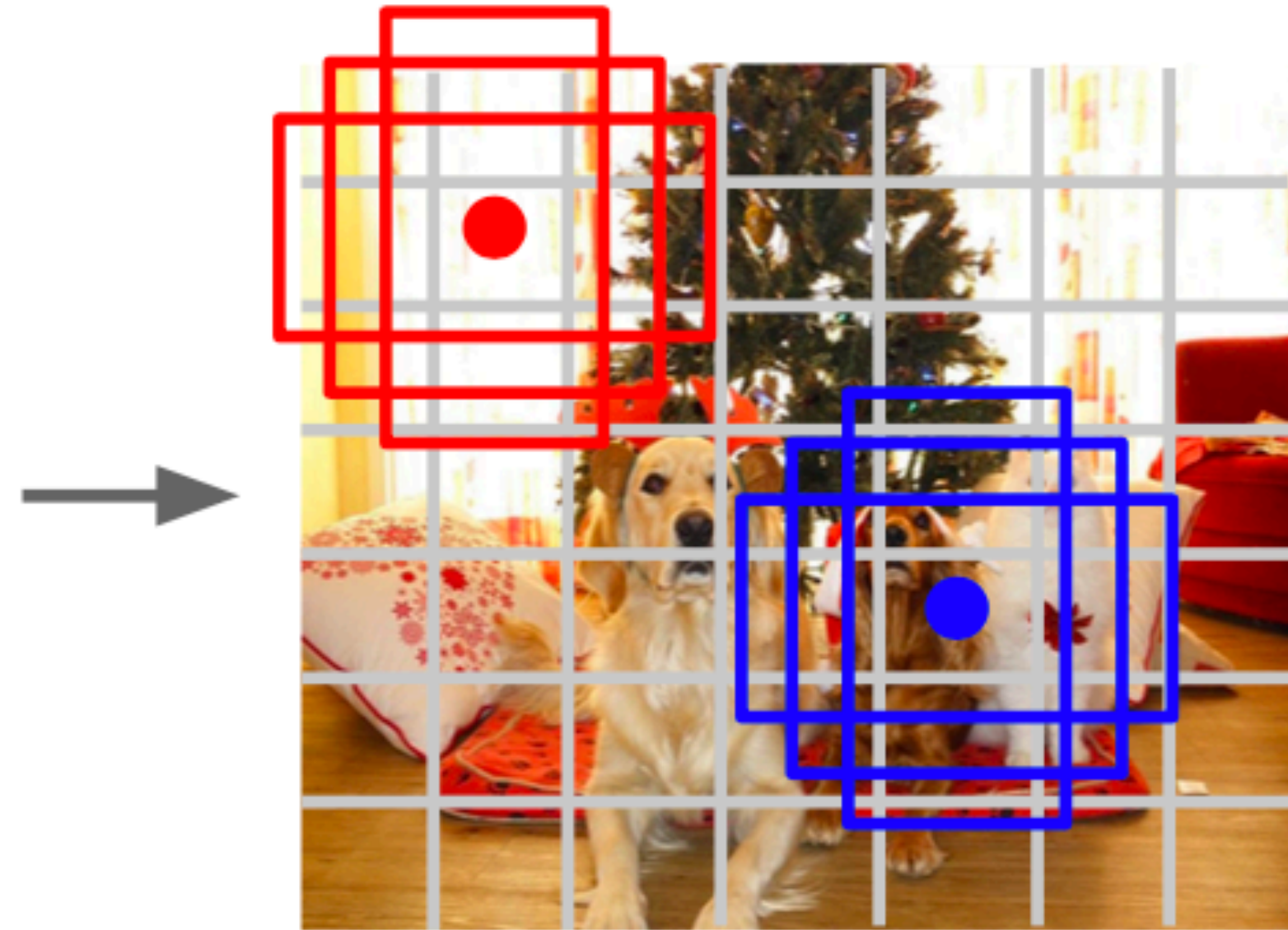
<http://pureddie.com/yolo>

YOLO: You Only Look Once

[Redmon et al, CVPR 2016]

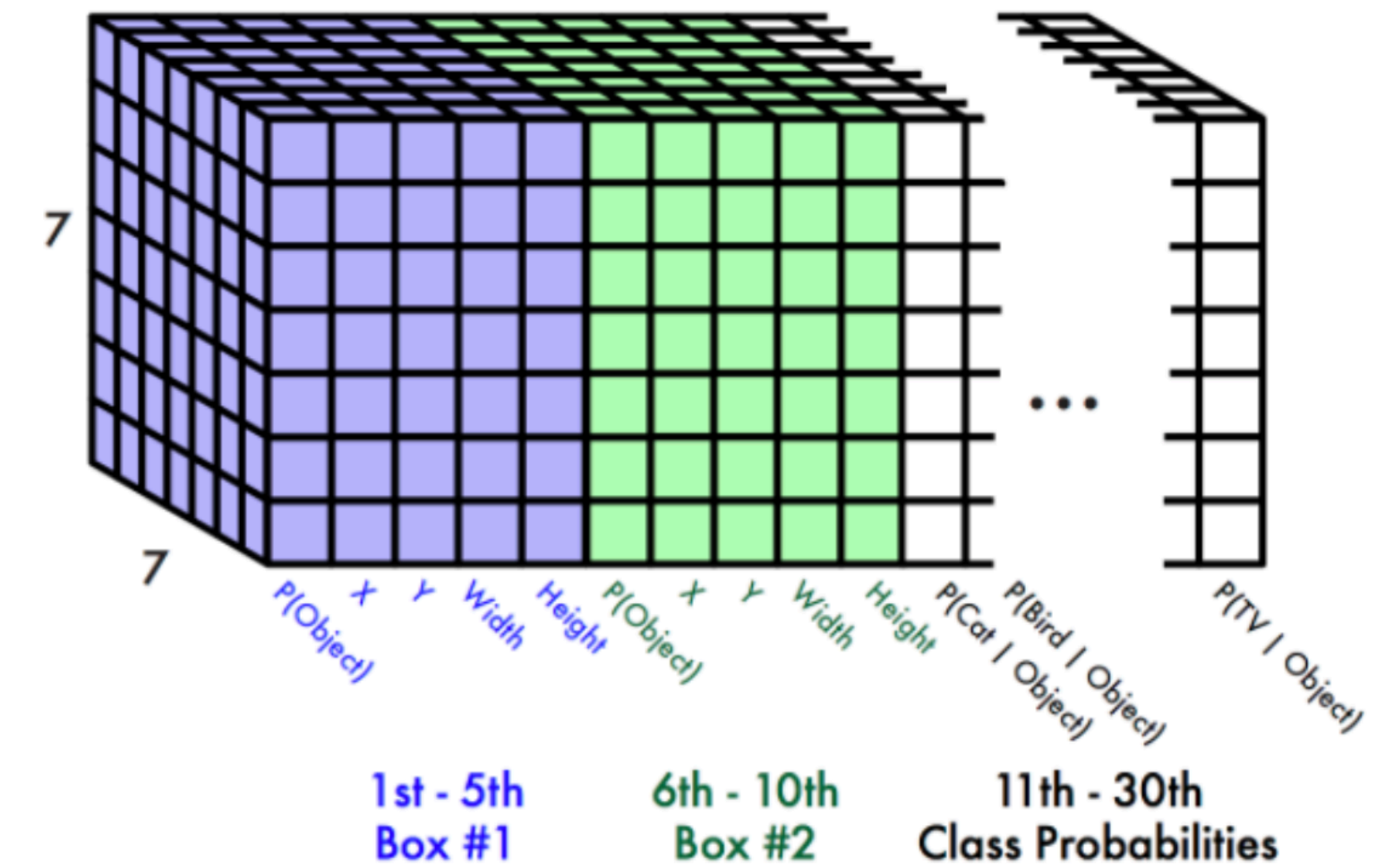


Input image
 $3 \times H \times W$



Divide image into grid
 7×7

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

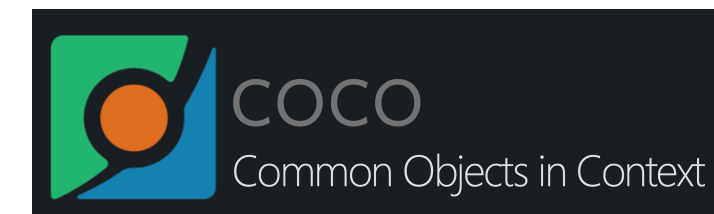


Computer **Vision Problems** (no language for now)

Segmentation

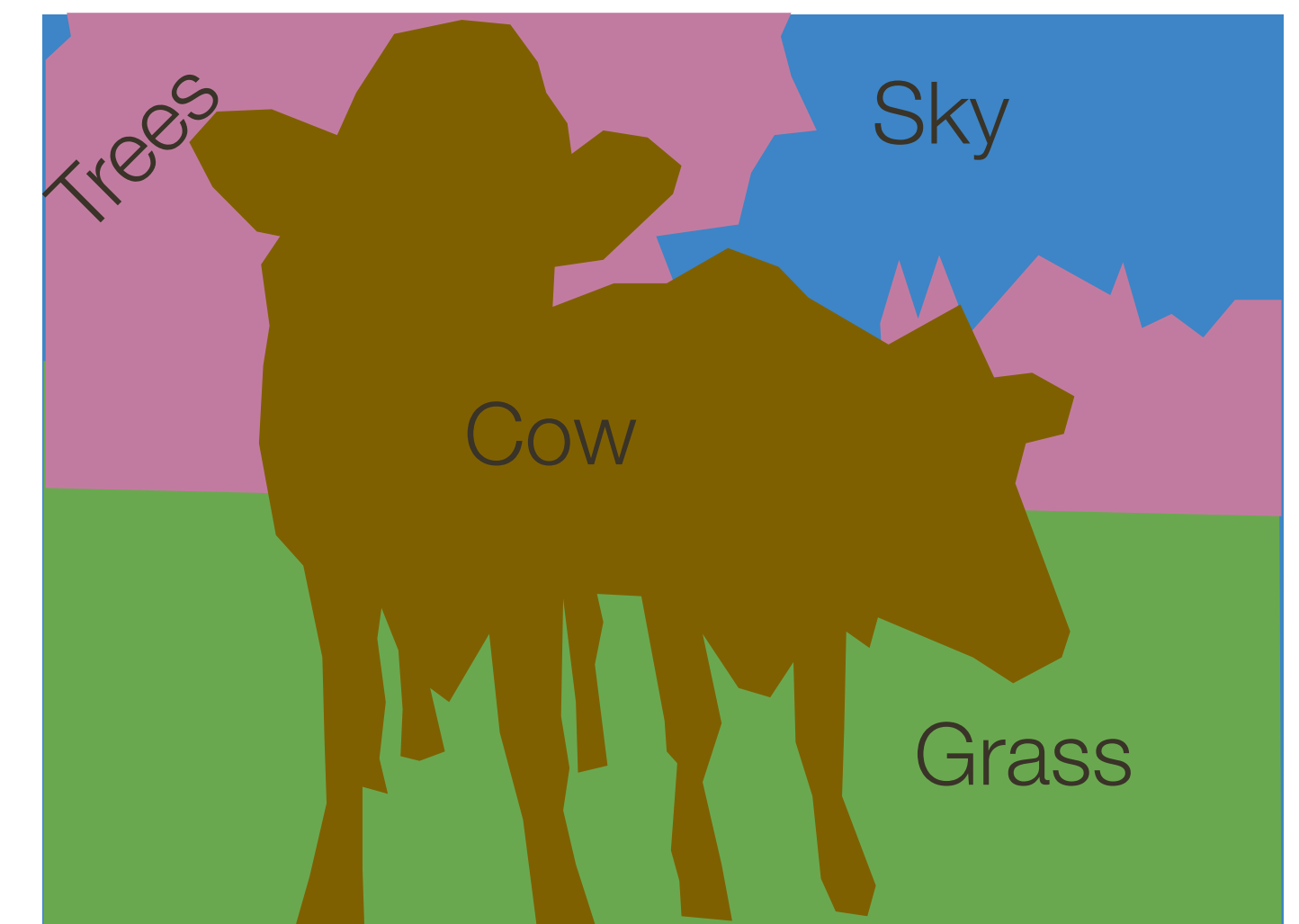
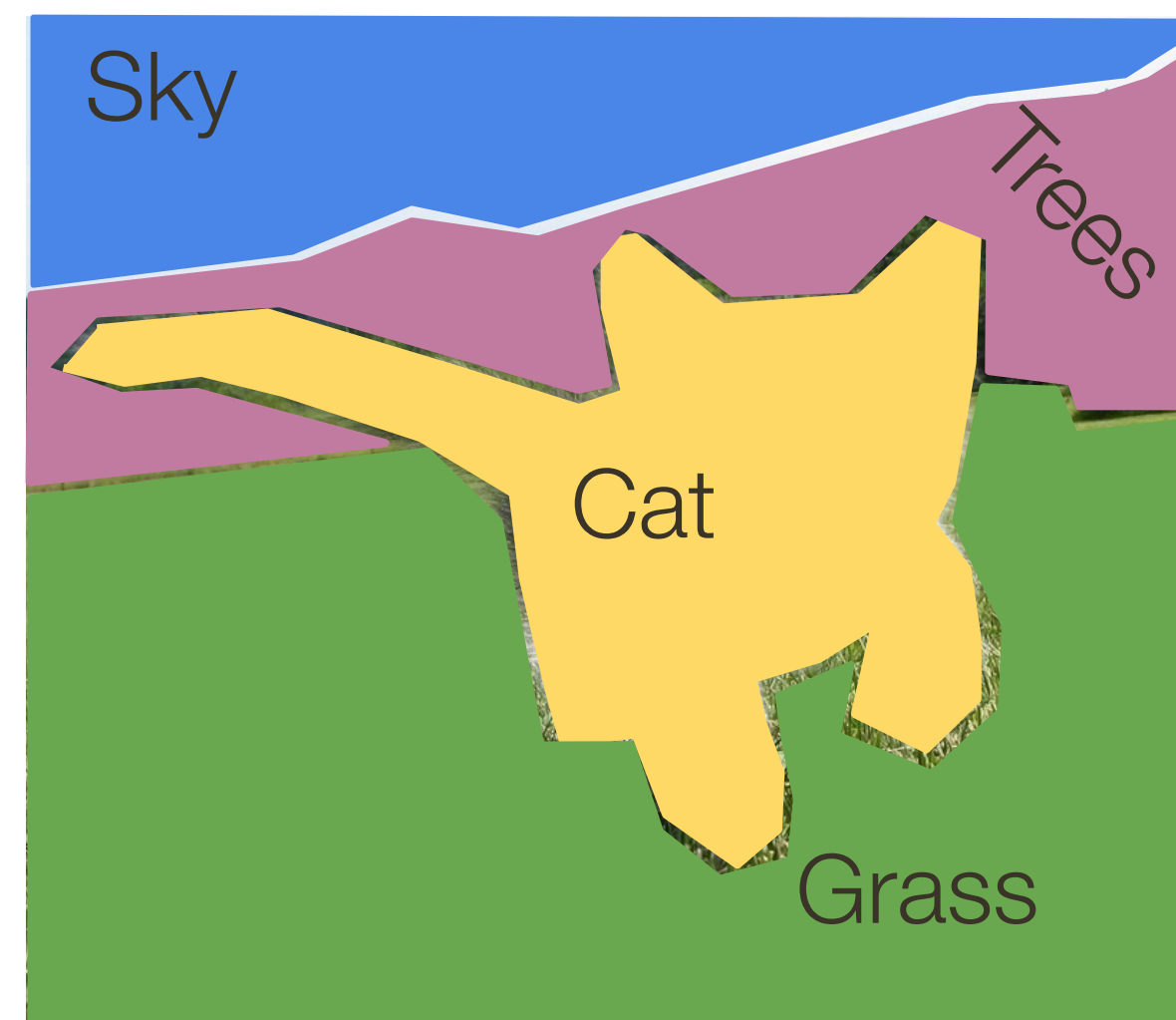


Horse
Person



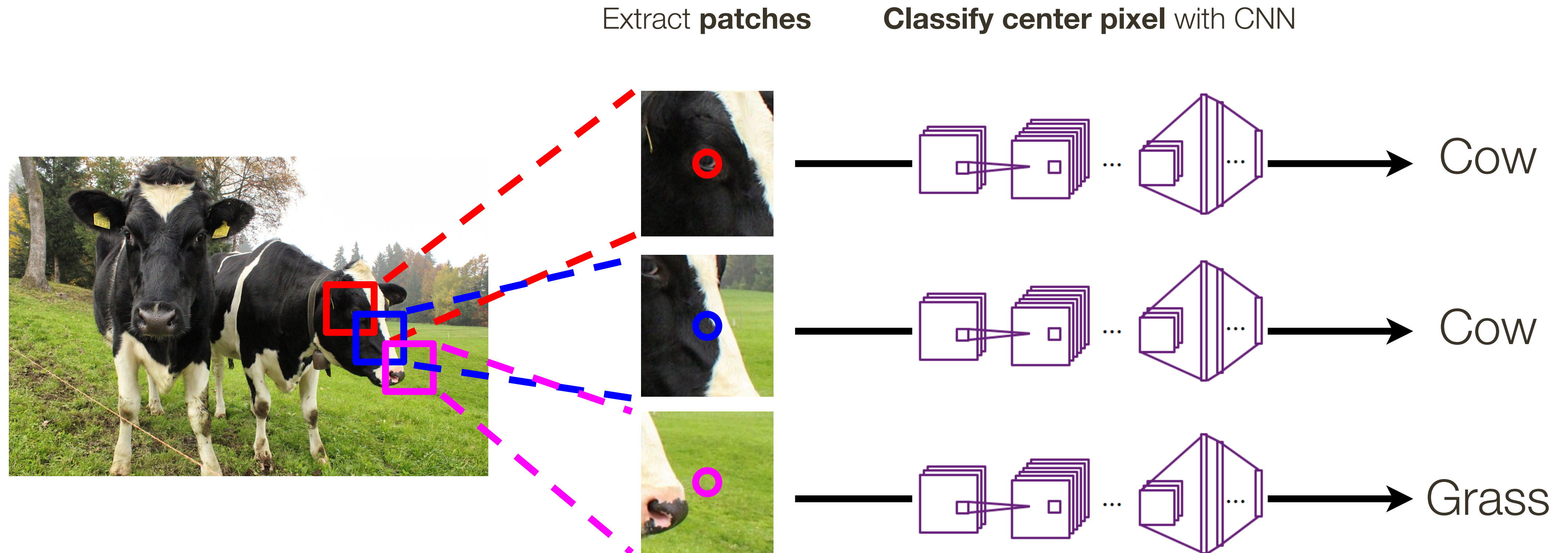
Semantic Segmentation

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



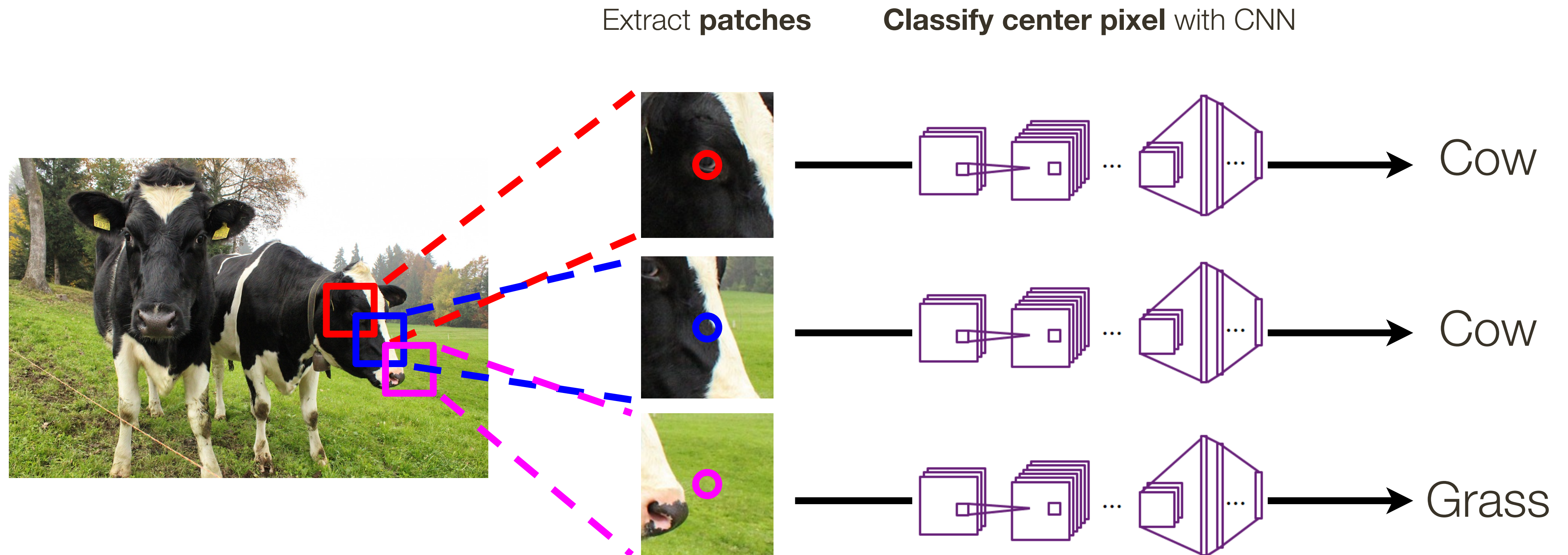
Semantic **Segmentation**: Sliding Window

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



Semantic **Segmentation**: Sliding Window

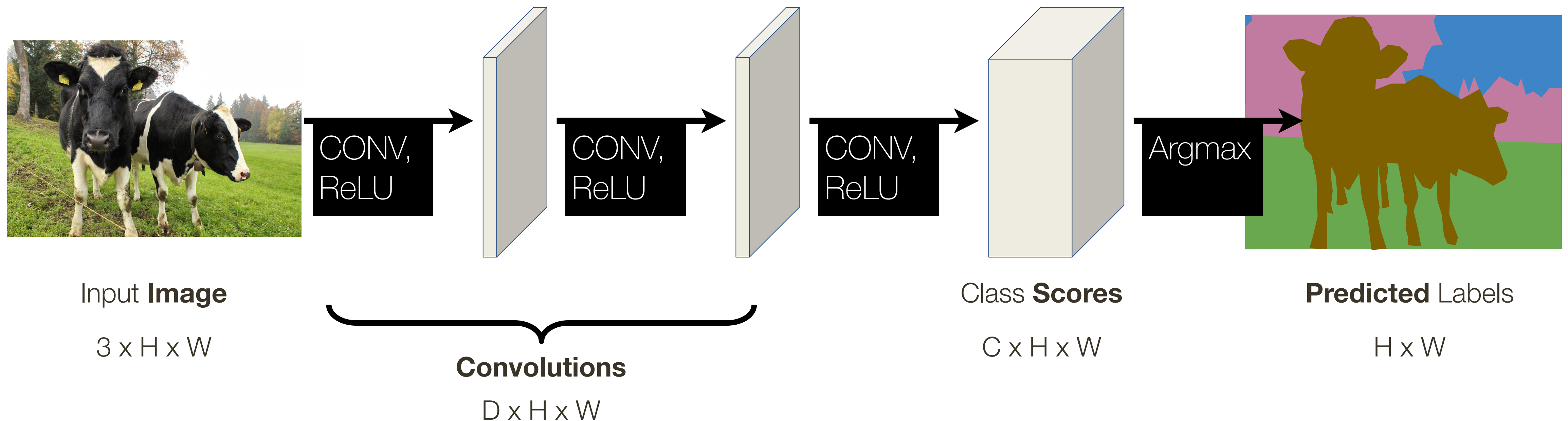
[Farabet et al, TPAMI 2013]
[Pinheiro et al, ICML 2014]



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

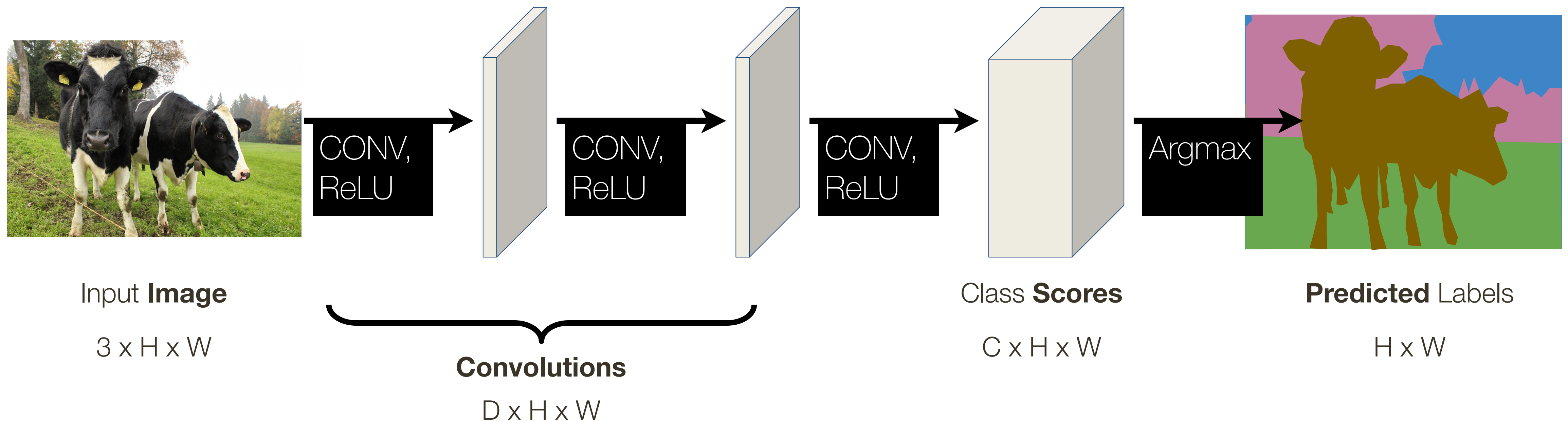
Semantic **Segmentation**: Fully Convolutional CNNs

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



Semantic **Segmentation**: Fully Convolutional CNNs

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



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

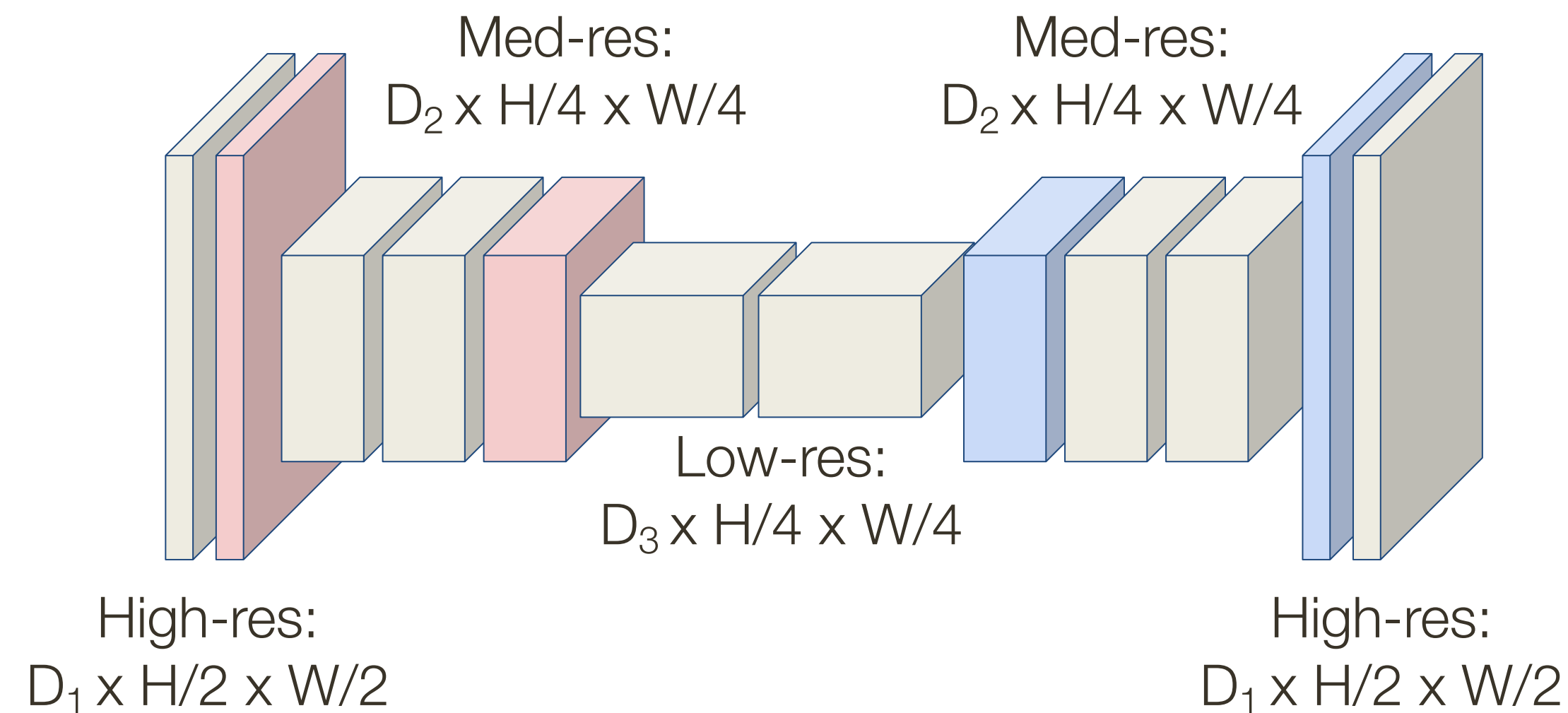
Semantic **Segmentation**: Fully Convolutional CNNs

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



Input **Image**

$3 \times H \times W$



Predicted Labels

$H \times W$

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

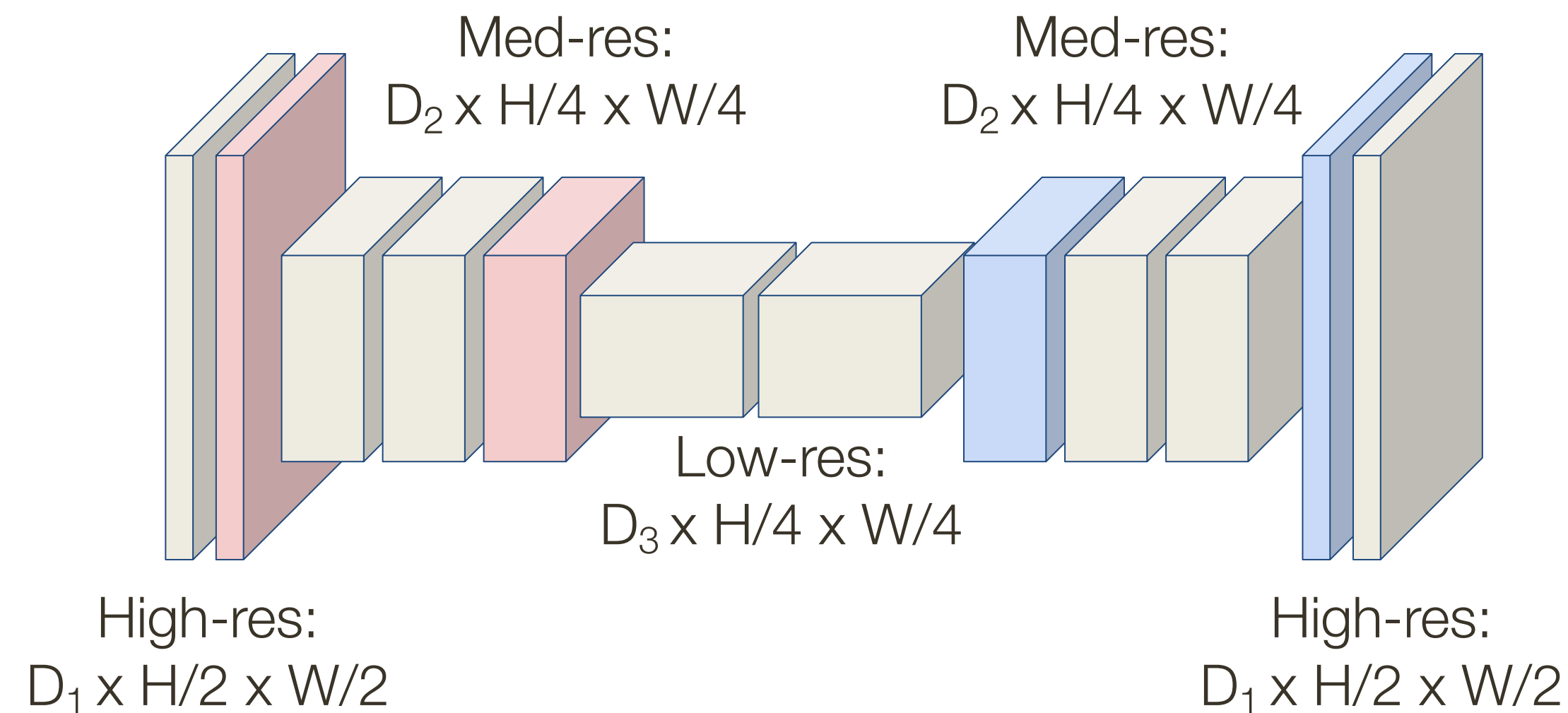
Semantic **Segmentation**: Fully Convolutional CNNs

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



Input **Image**

$3 \times H \times W$



Predicted Labels

$H \times W$

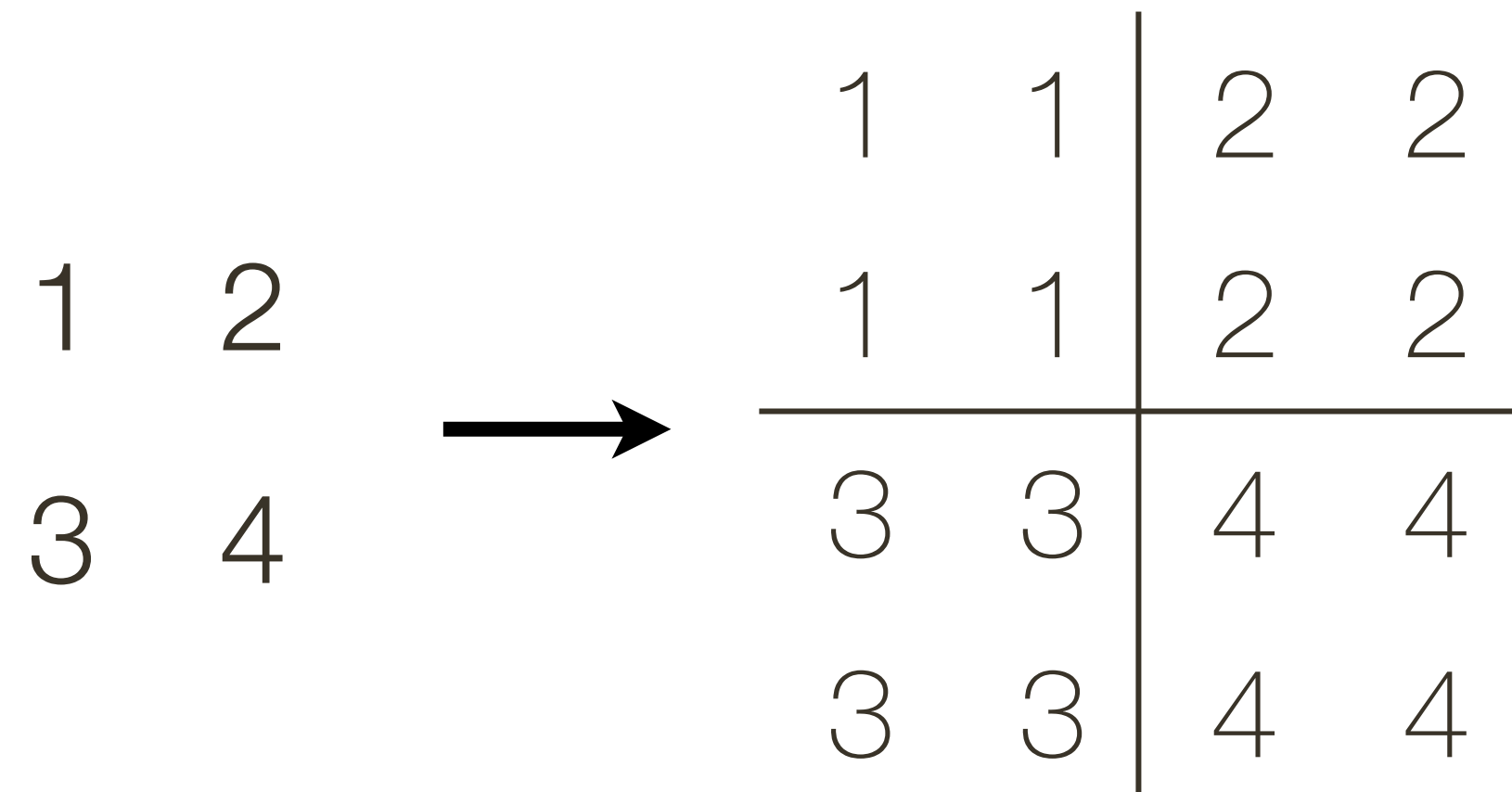
Downsampling = Pooling

Upsampling = ???

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

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

Nearest Neighbor

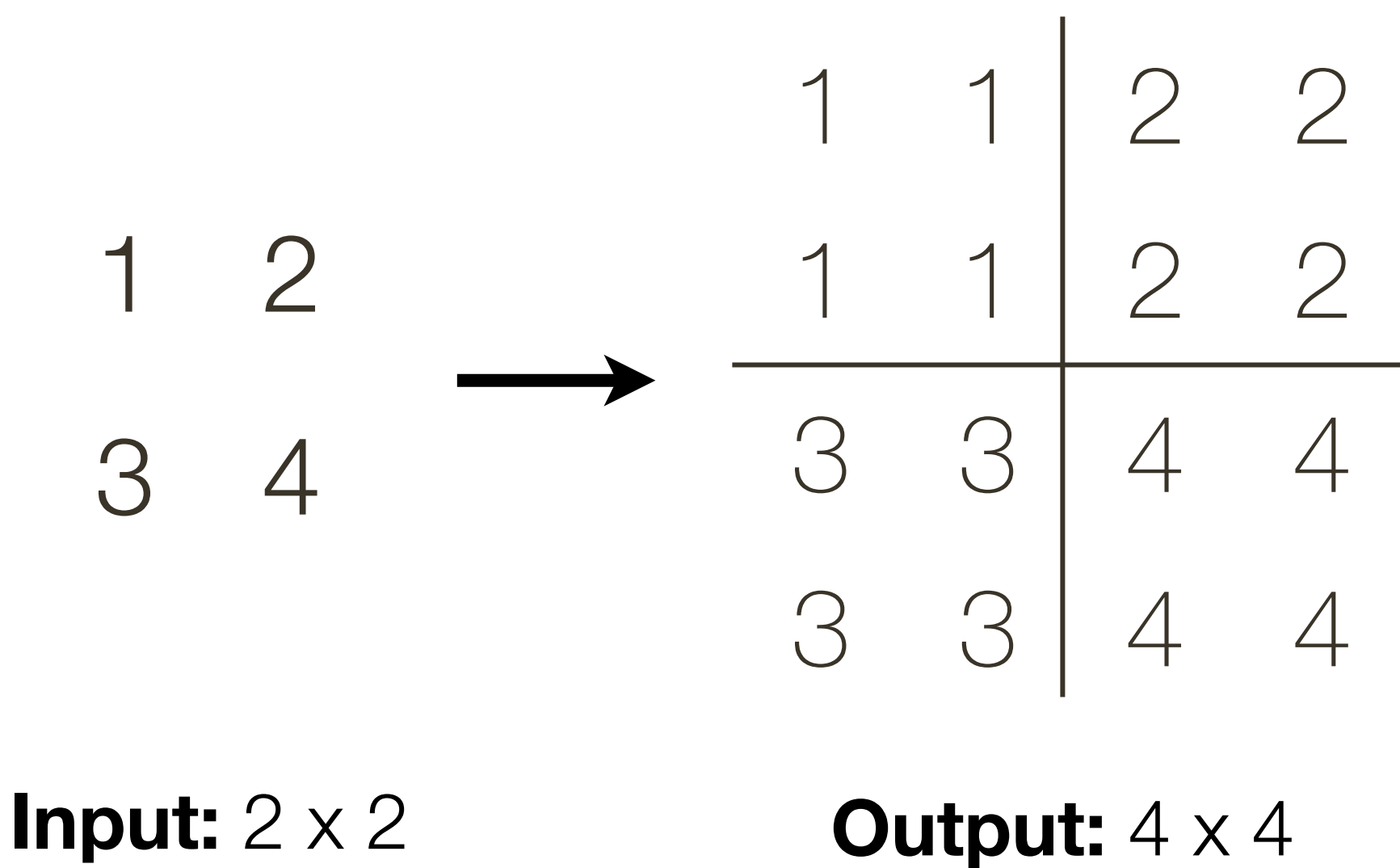


Input: 2 x 2

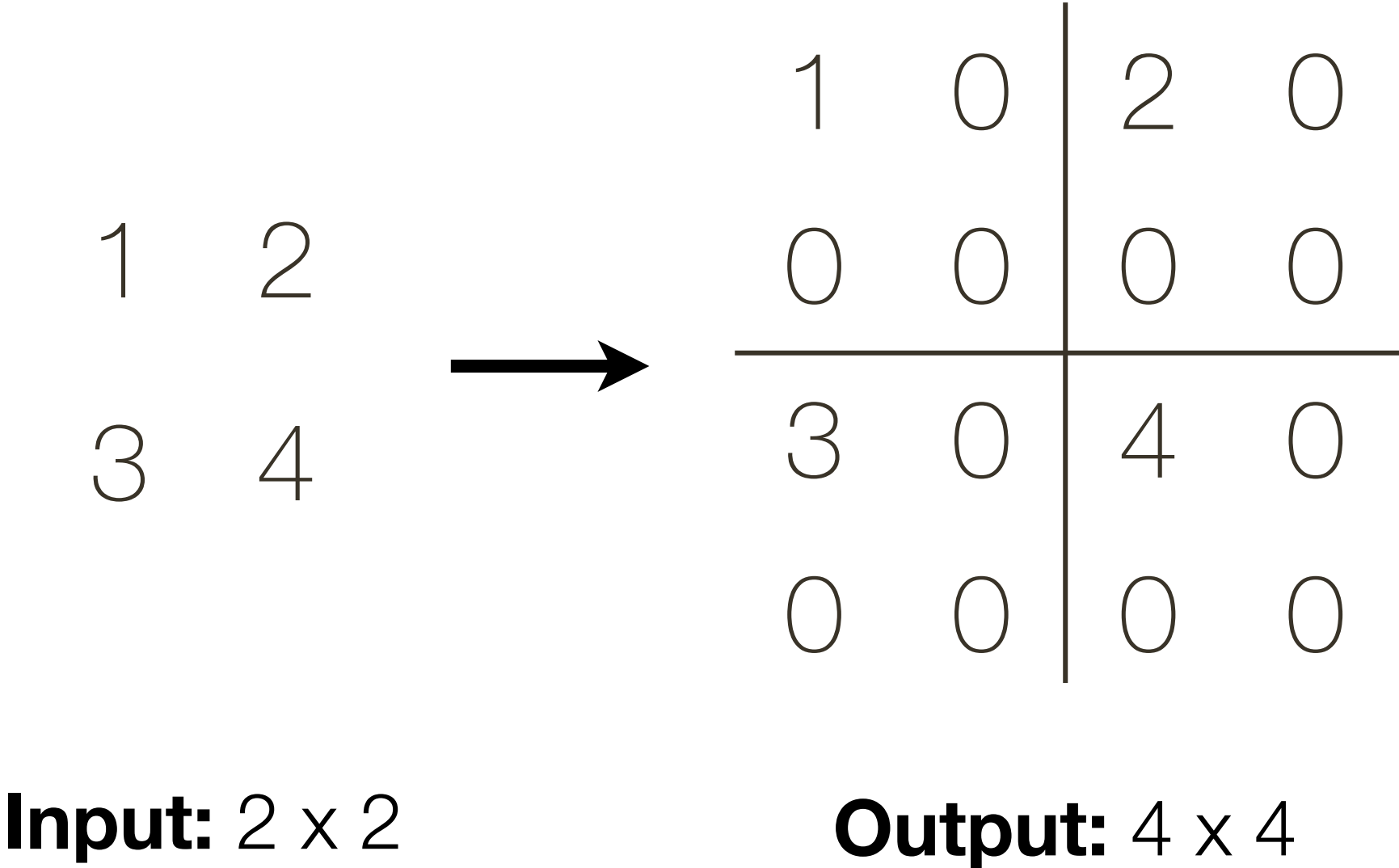
Output: 4 x 4

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

Nearest Neighbor



“Bed of Nails”



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

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

Max Pooling

Remember which element was max!

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

Input: 4 x 4



5	6
7	8

Output: 2 x 2



...

Rest of the network

Max Unpooling

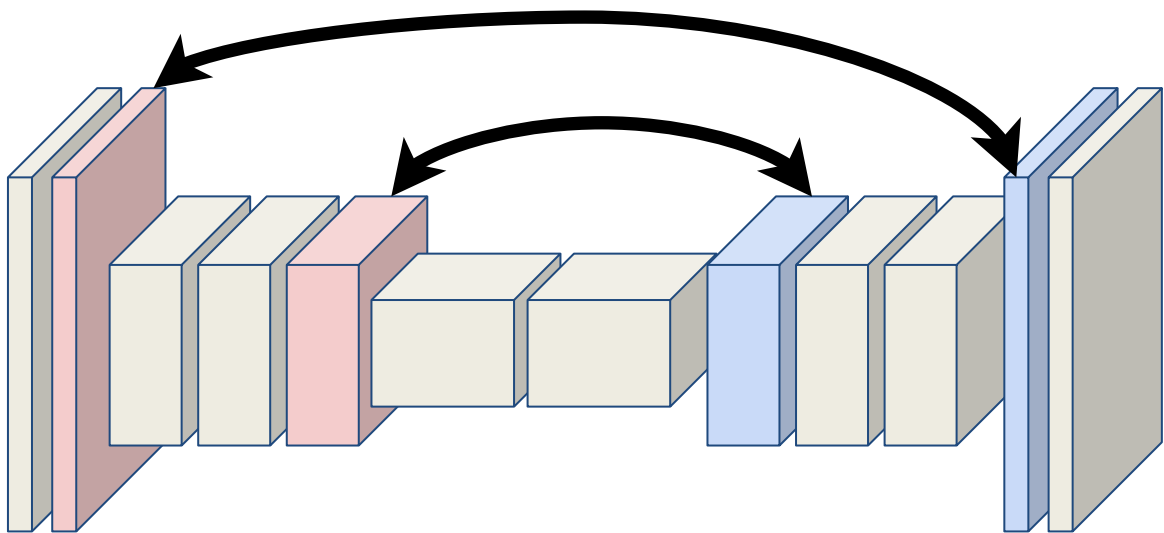
Use positions from pooling layer

1	2
3	4

Input: 2 x 2

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

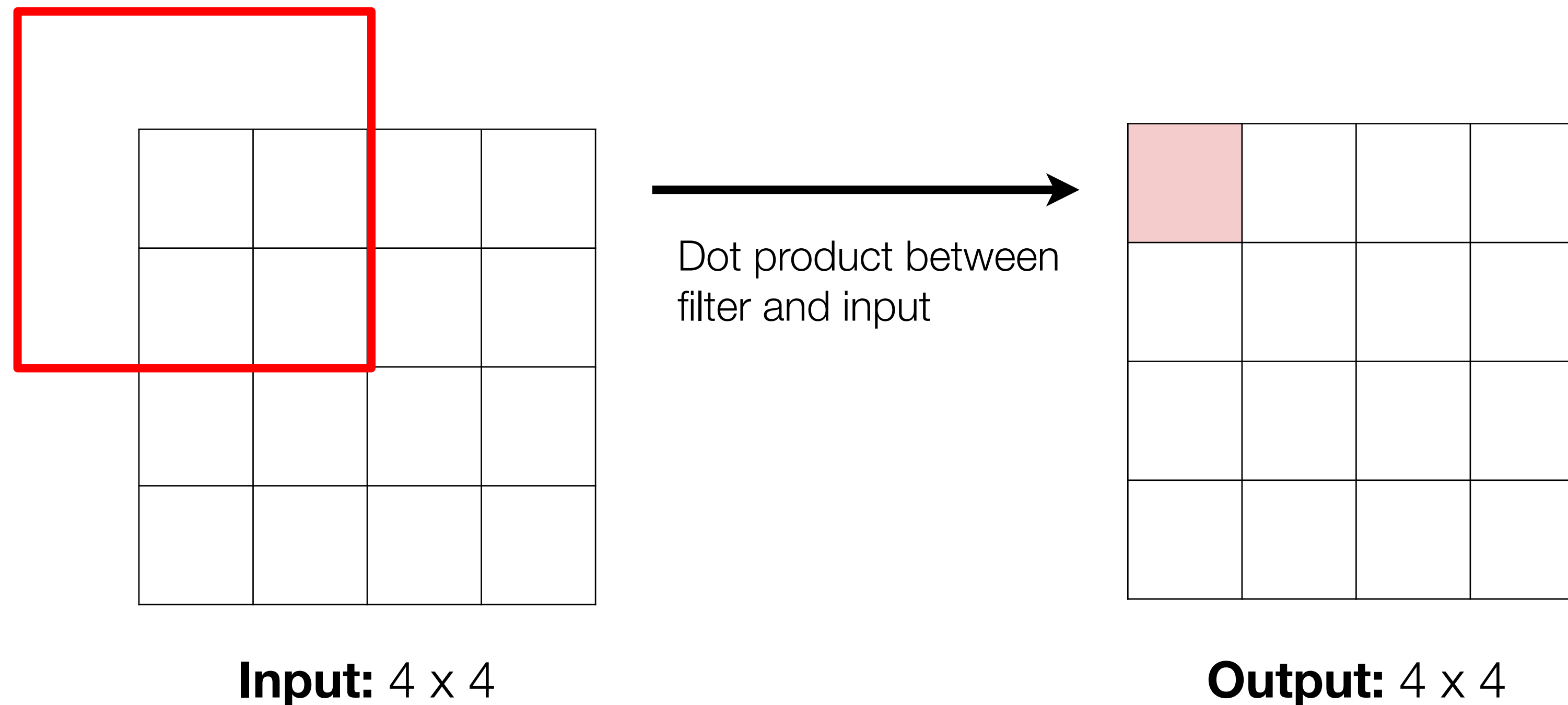
Output: 4 x 4



Corresponding pairs of downsampling and upsampling layers

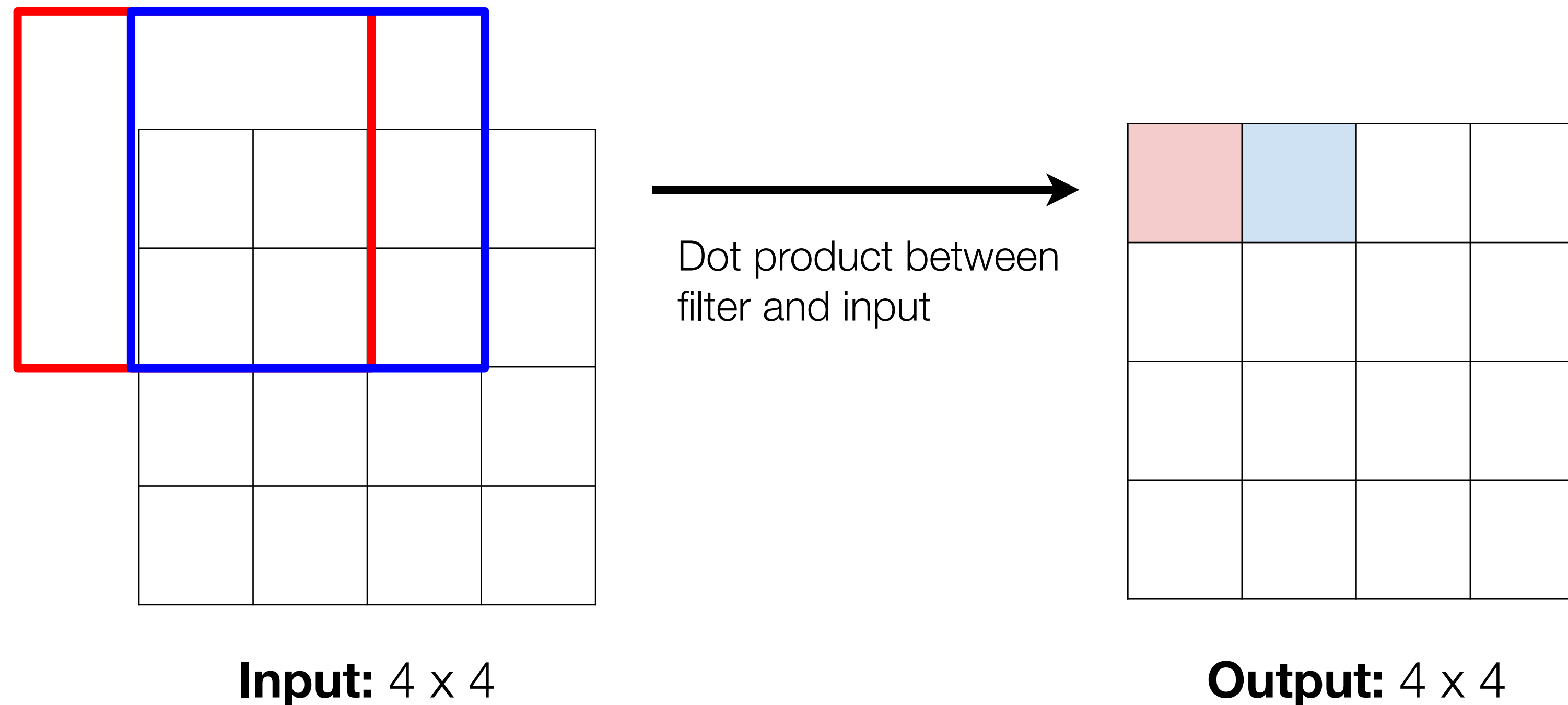
In-network **Up Sampling:** Transpose Convolution

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



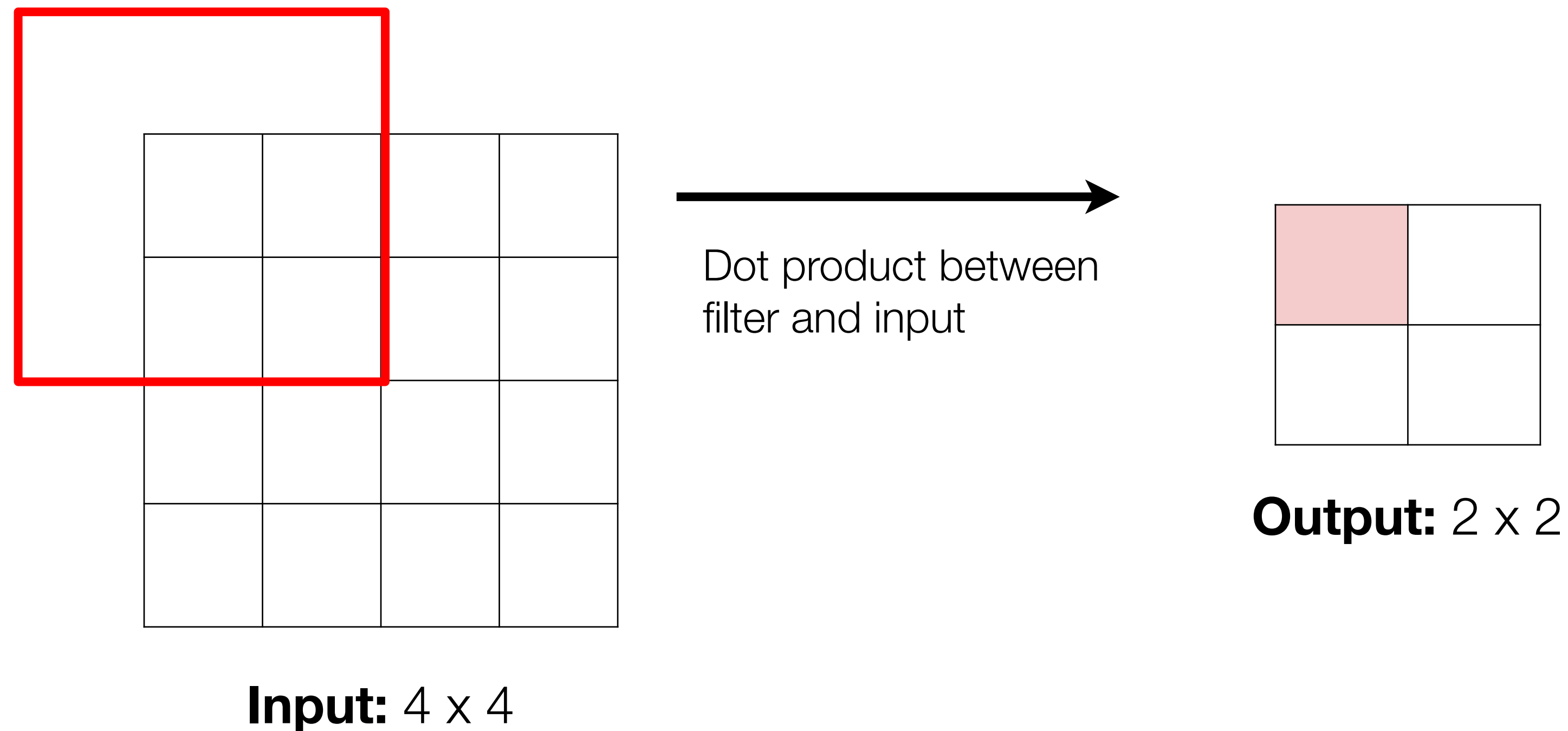
In-network **Up Sampling:** Transpose Convolution

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



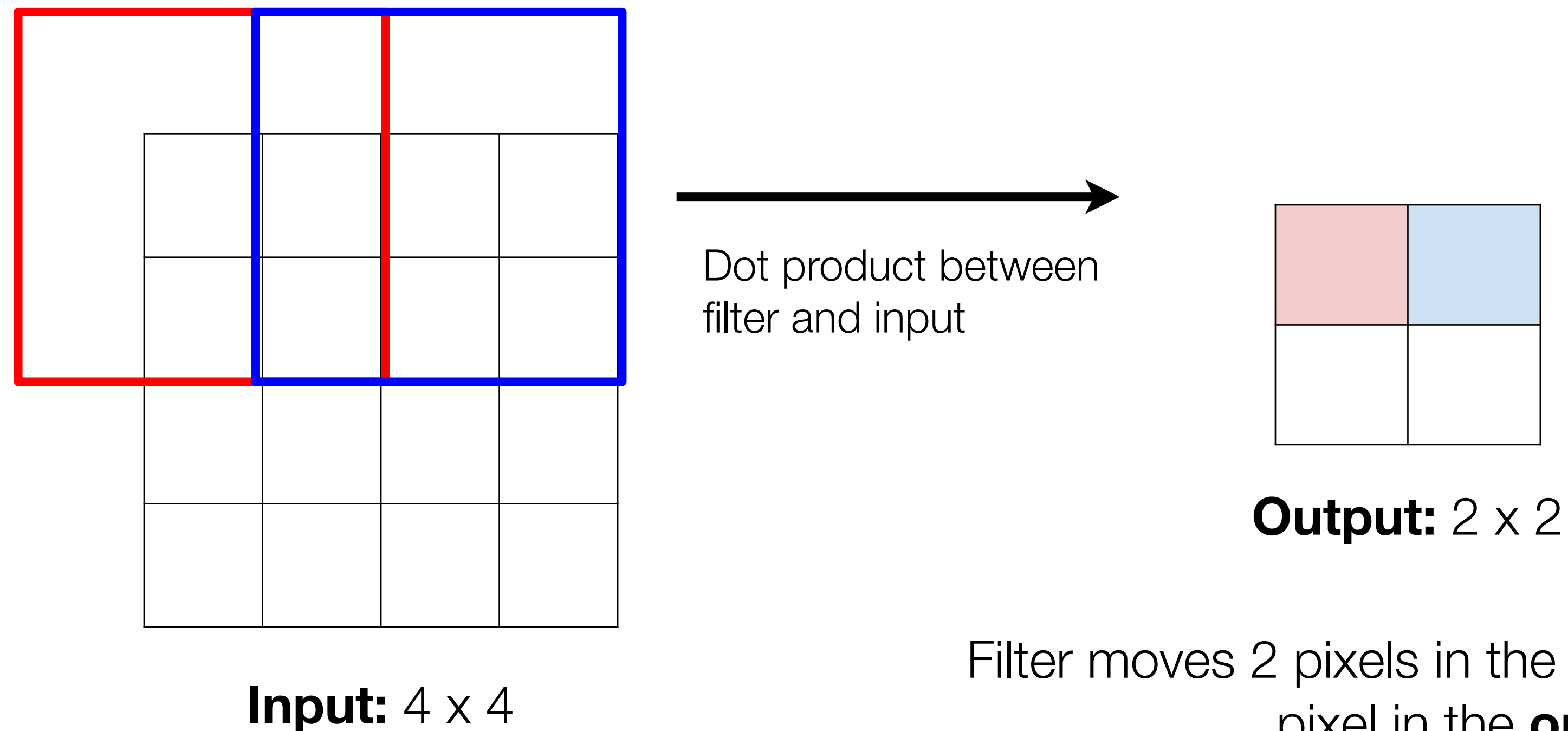
In-network **Up Sampling:** Transpose Convolution

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



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

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



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

Stride gives ratio in movement in input vs output

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

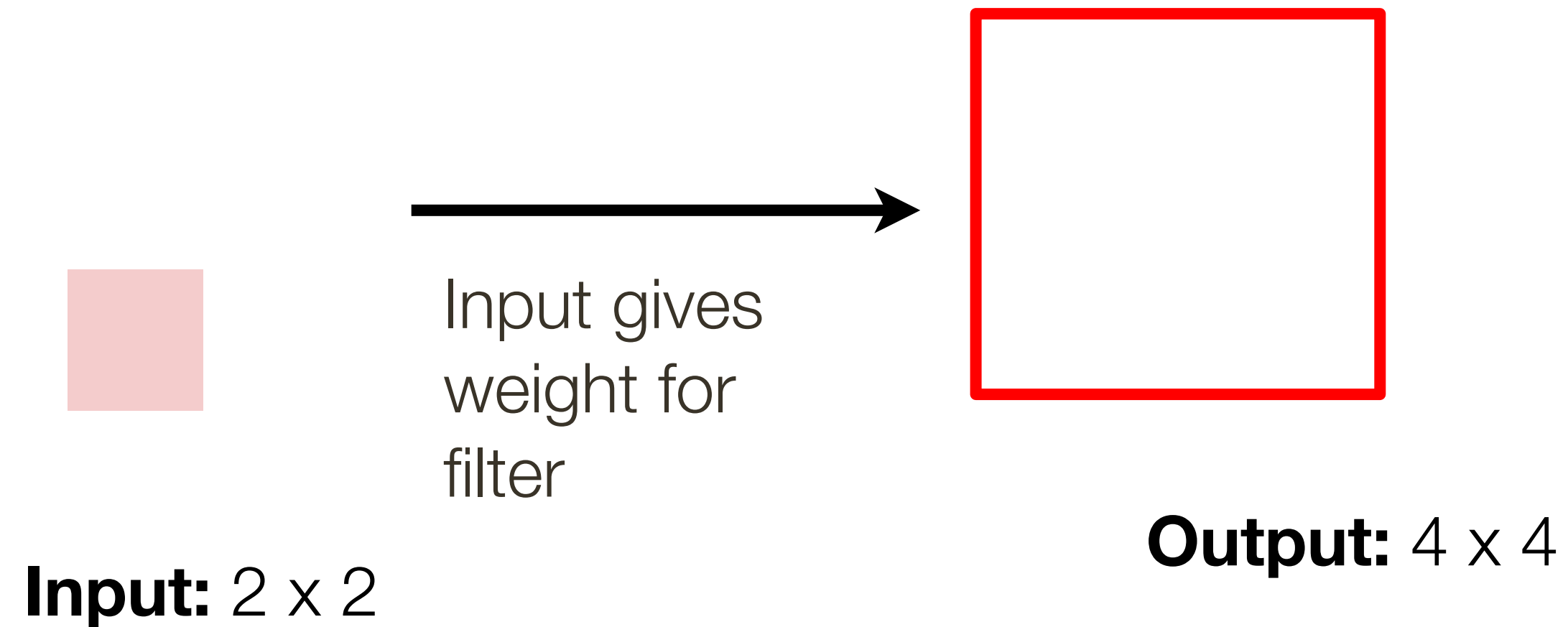
3 x 3 **transpose** convolution, stride 2 pad 1

Input: 2 x 2

Output: 4 x 4

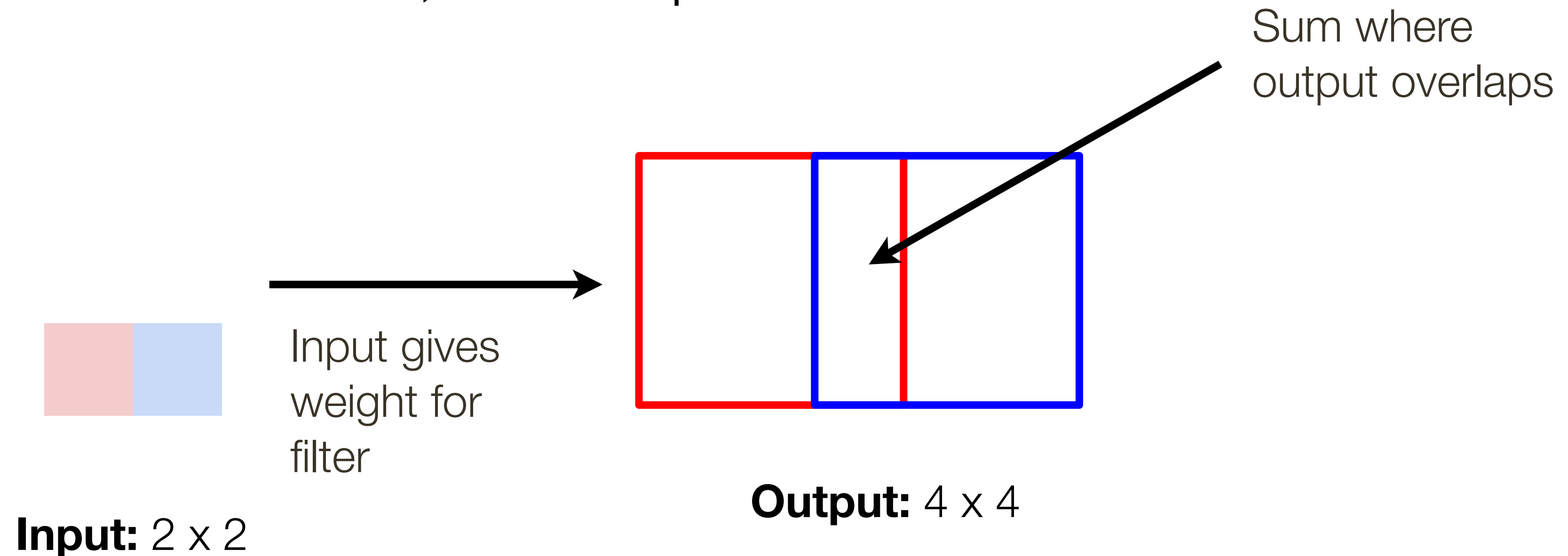
In-network **Up Sampling**: Transpose Convolution

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



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

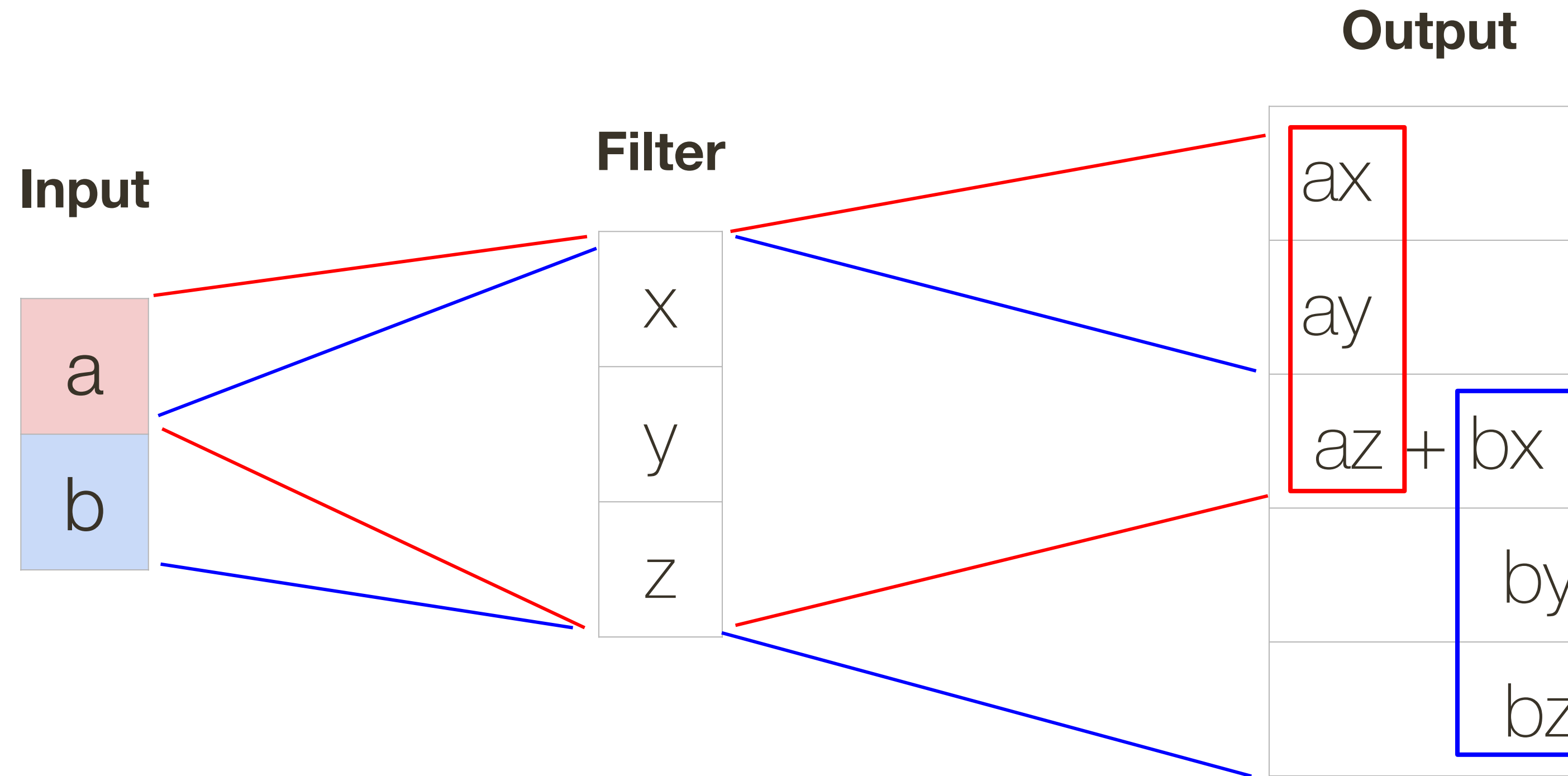
3 x 3 **transpose** convolution, stride 2 pad 1



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

Stride gives ratio in movement in output vs input

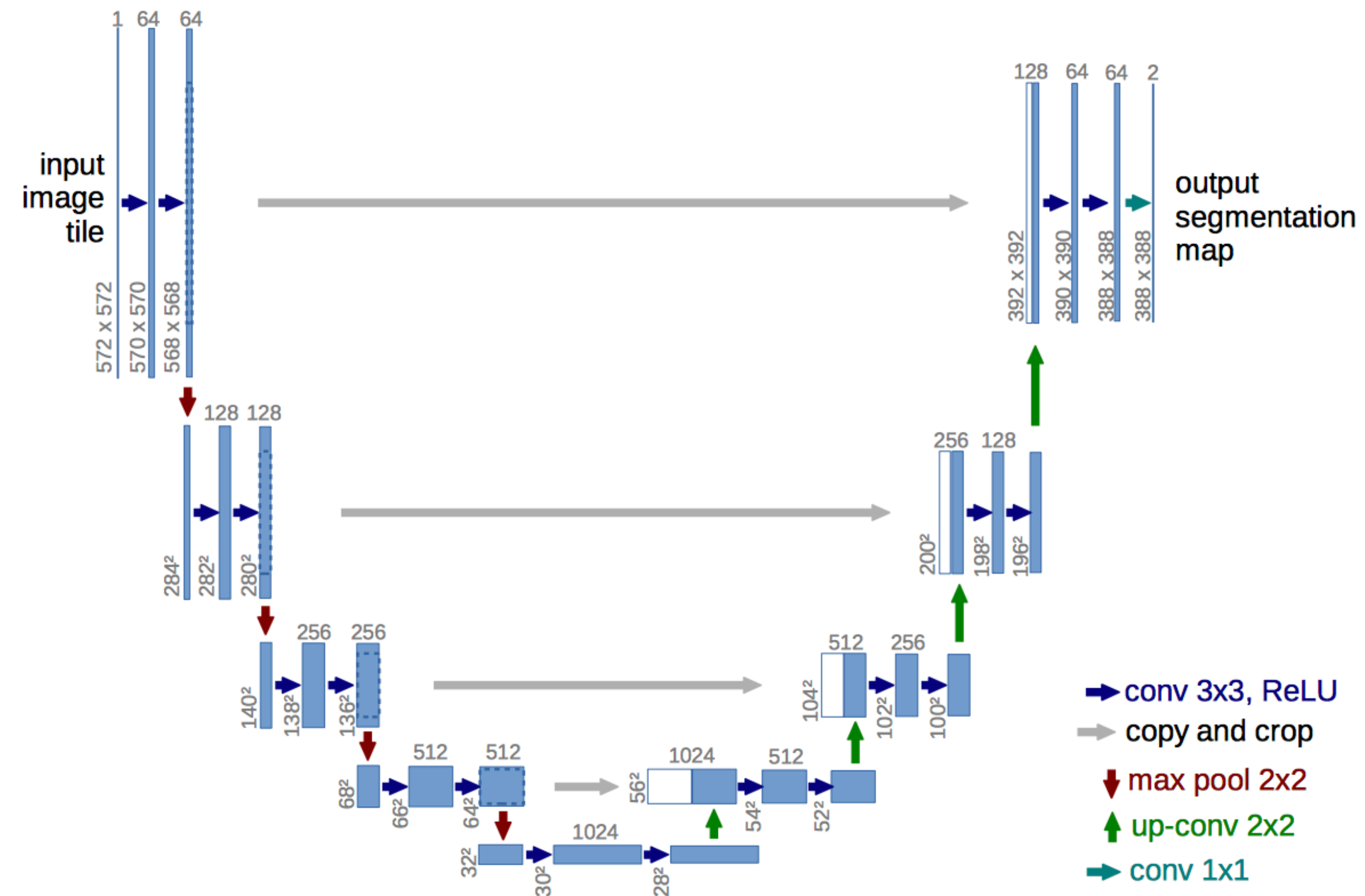
Transpose Convolution: 1-D Example



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

U-Net Architecture

ResNet-like Fully convolutional CNN



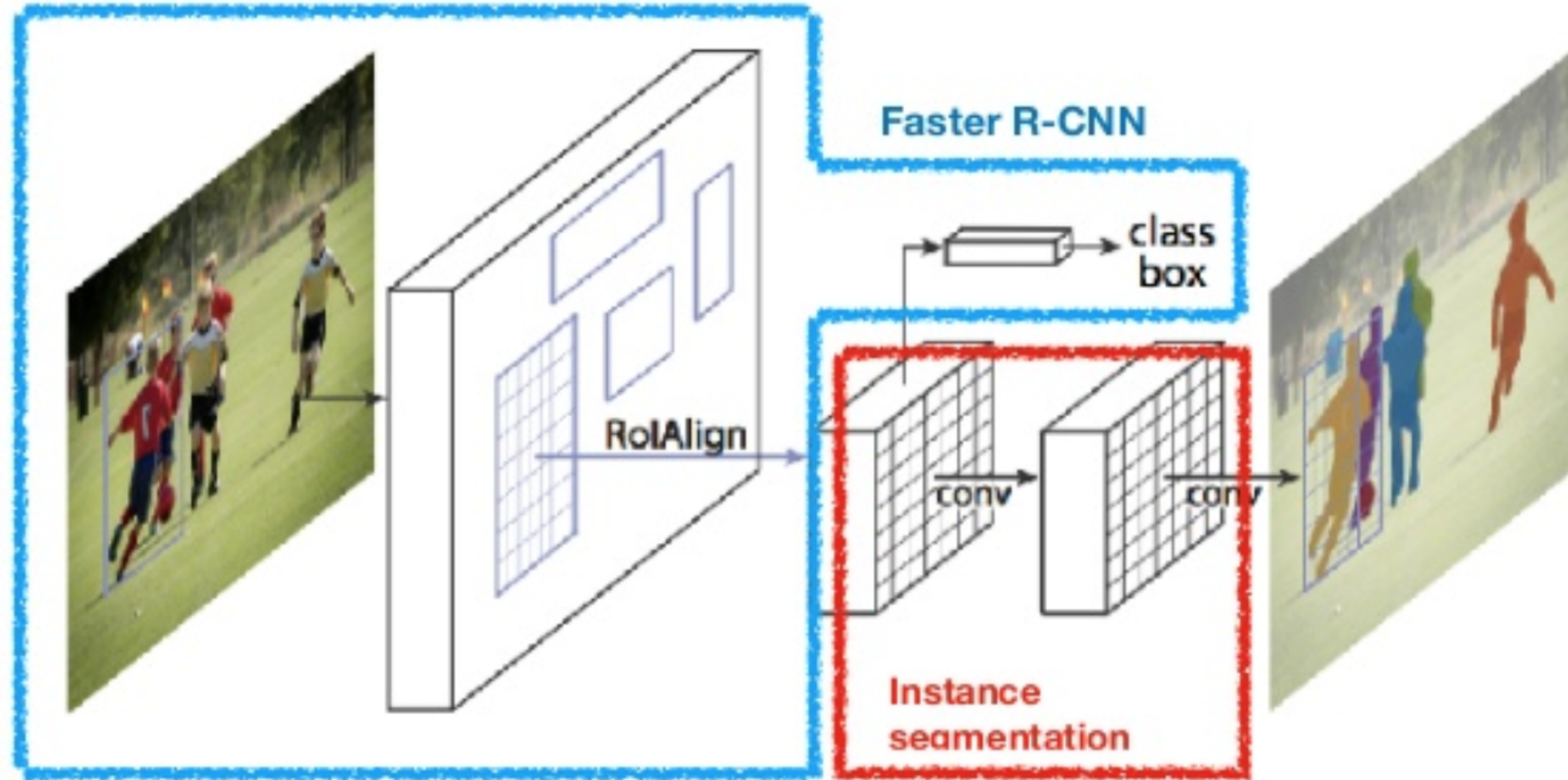
Computer **Vision Problems** (no language for now)

Instance Segmentation



Horse1
Horse2
Person1
Person2

Mask R-CNN



[He et al, 2017]

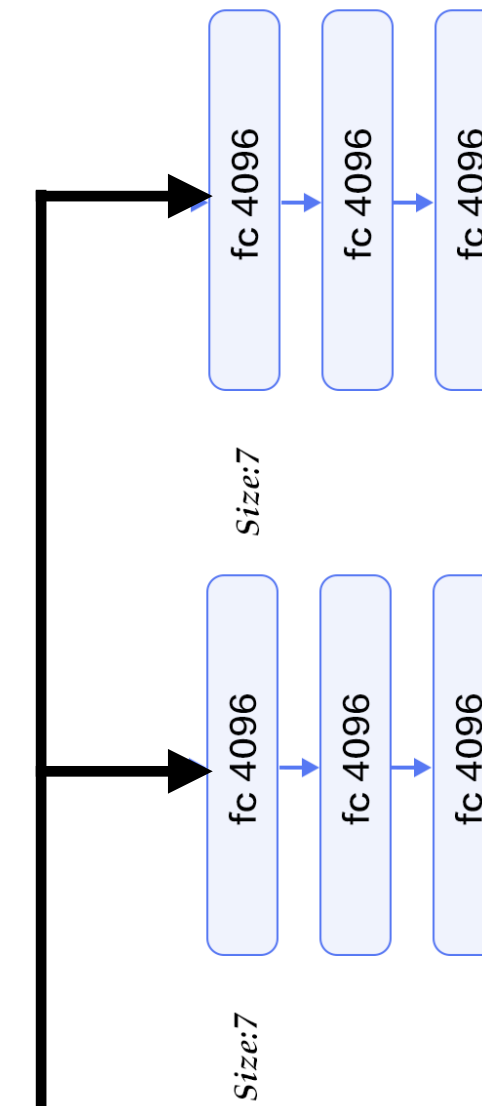
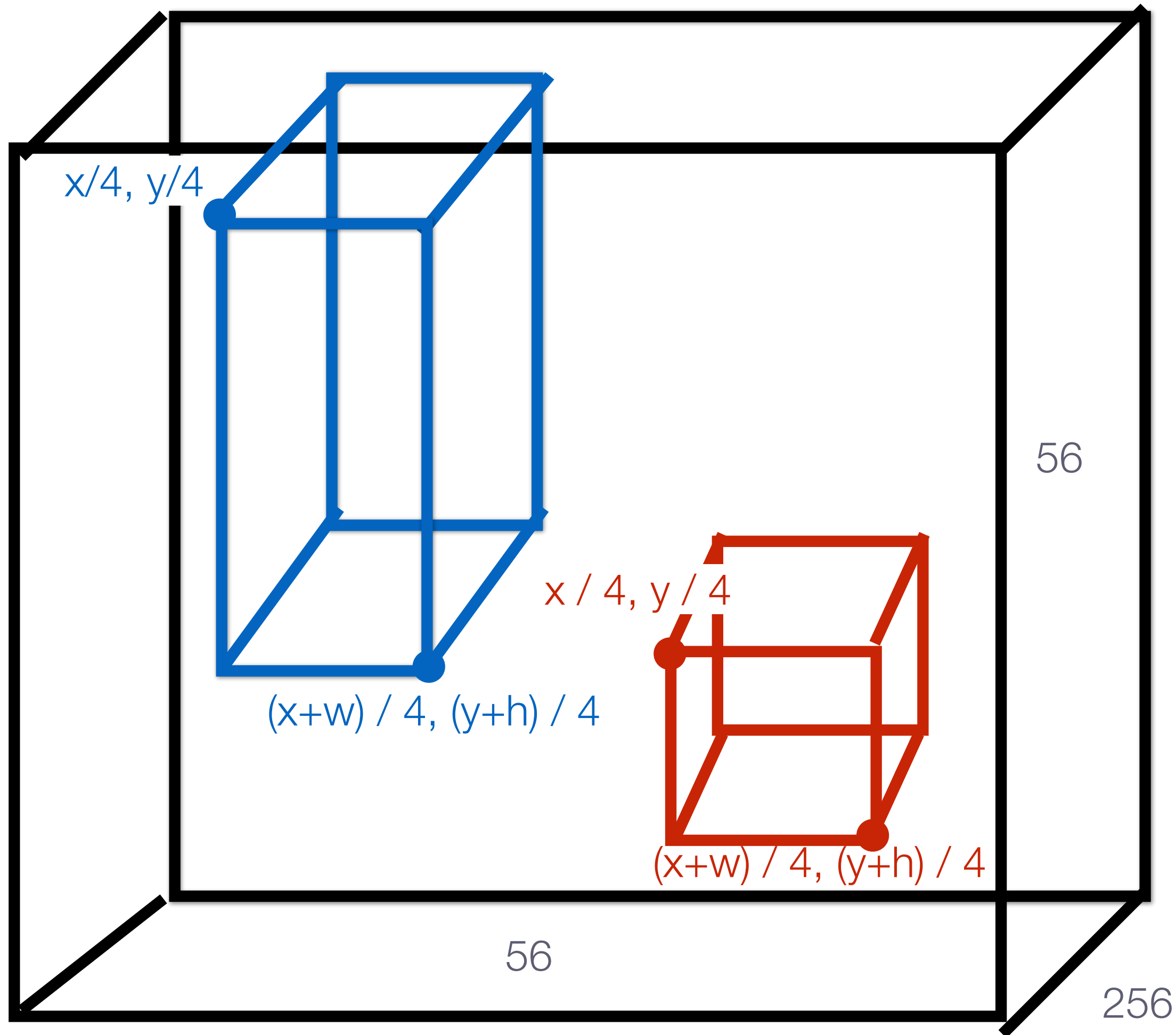
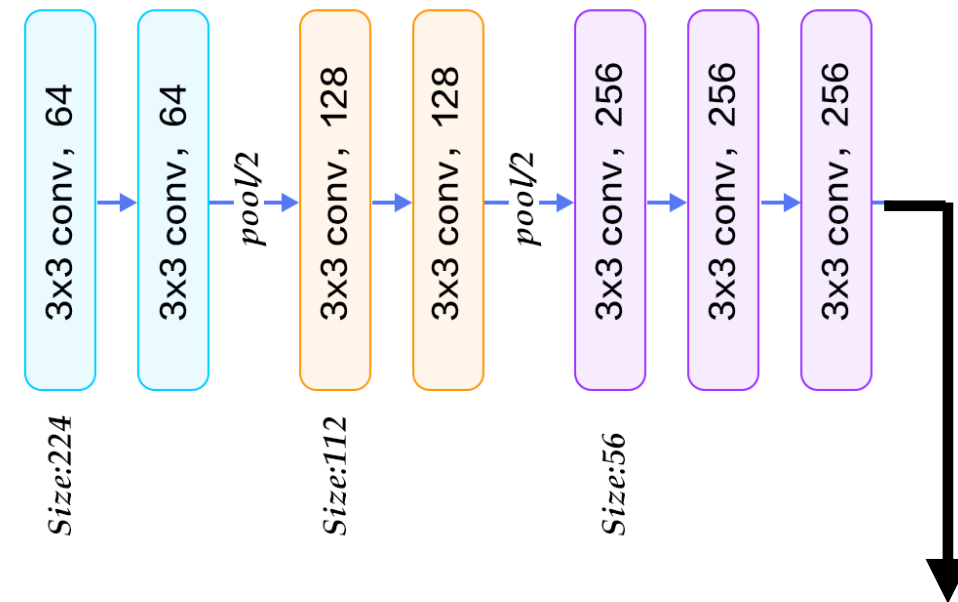
Mask R-CNN

Image

(resized to 224 x 224)

Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)

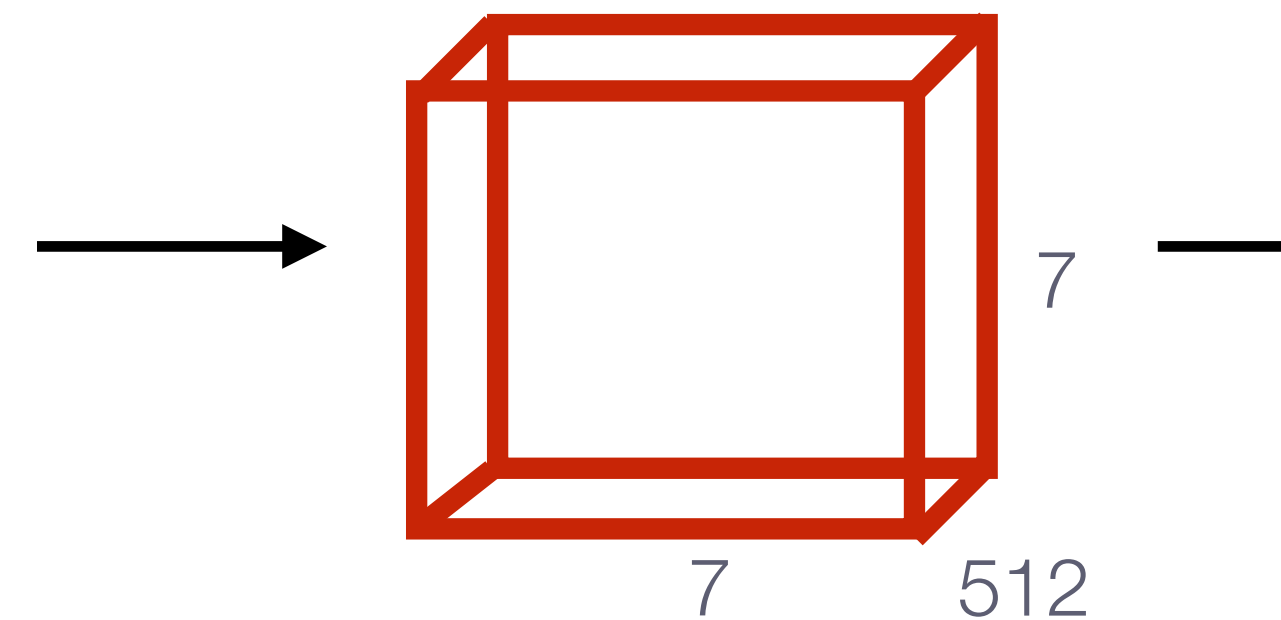


Classifier

(4096 x 80 categories + Softmax)

Regressor

(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h



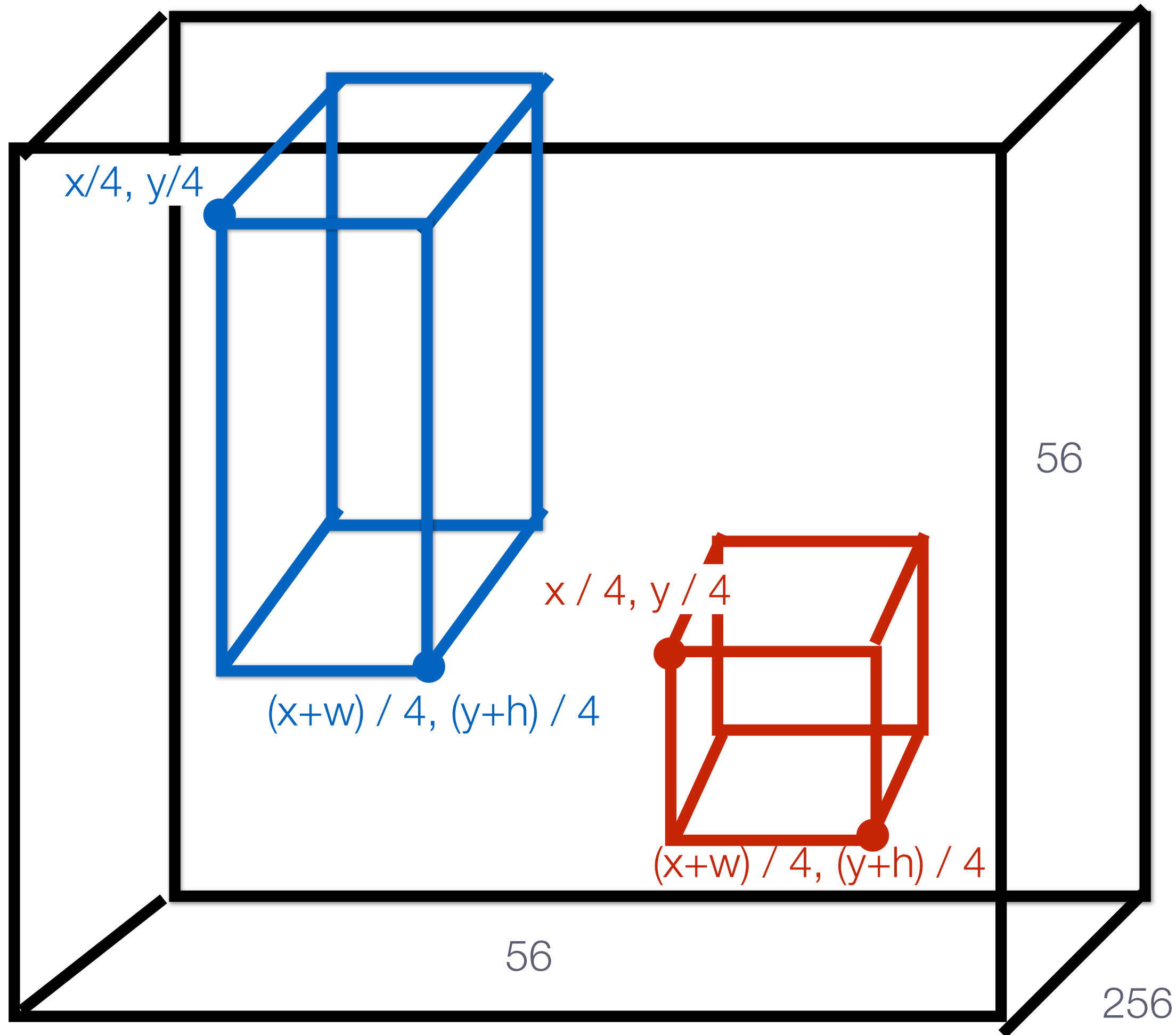
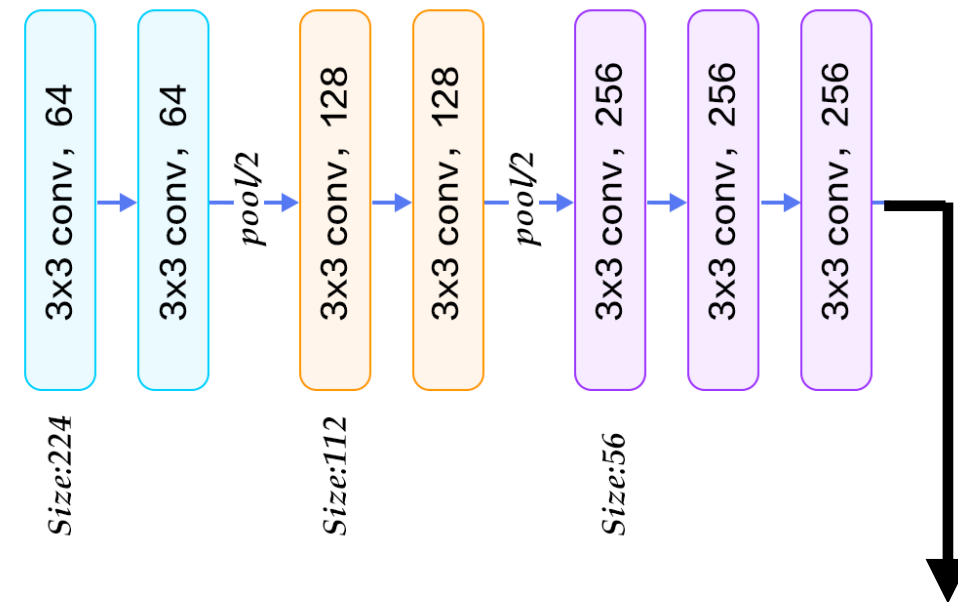
Mask R-CNN

Image

(resized to 224 x 224)

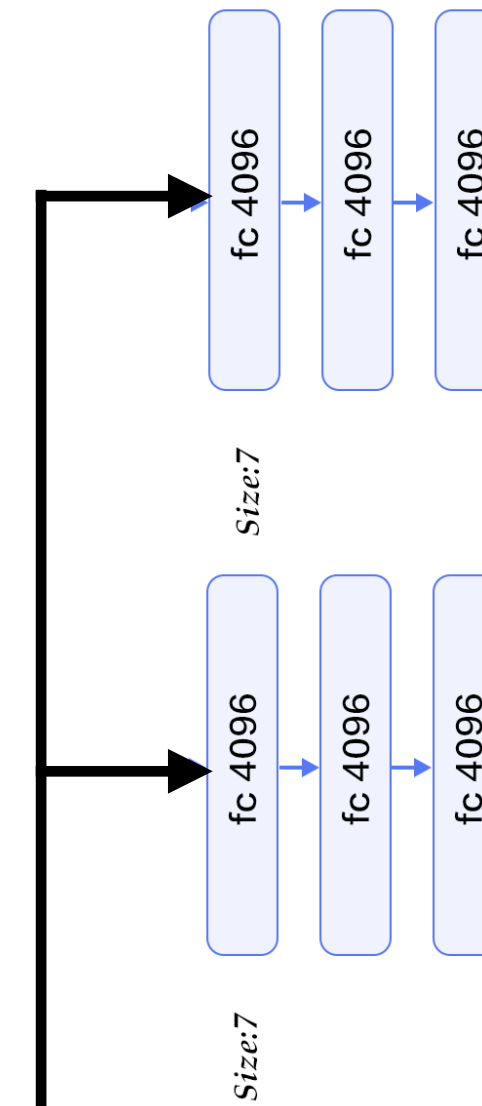
Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



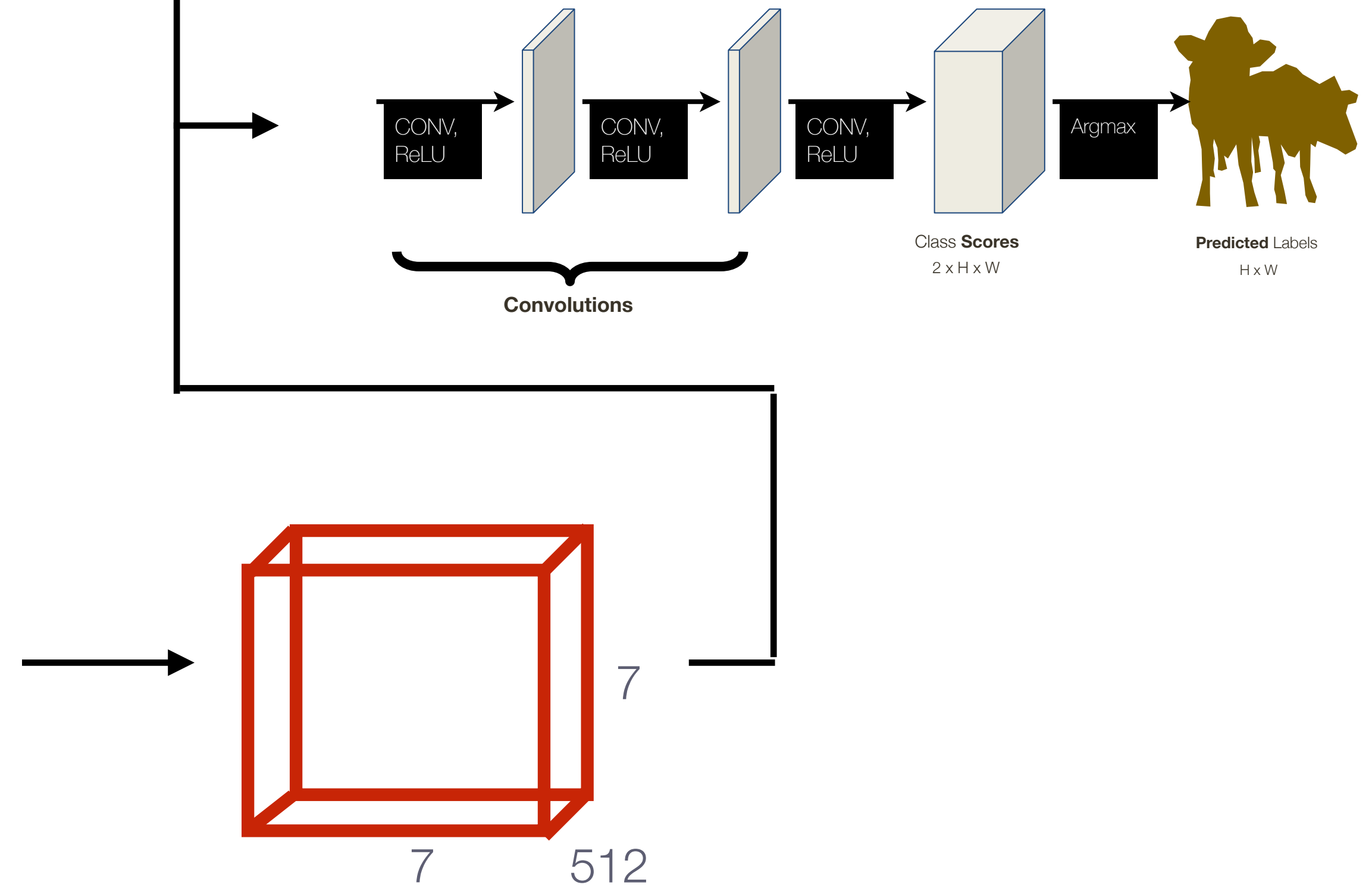
Classifier

(4096 x 80 categories + Softmax)



Regressor

(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h



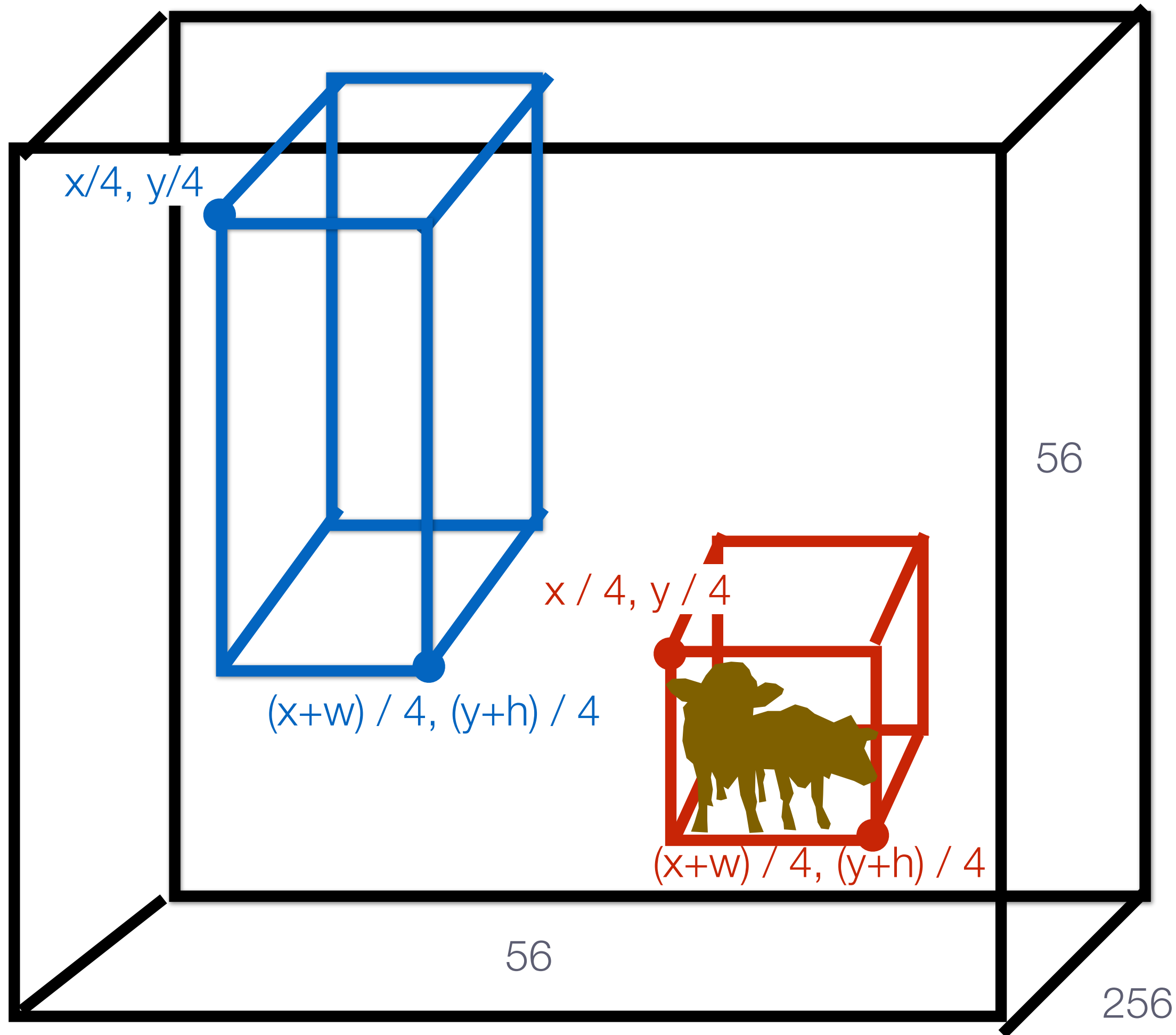
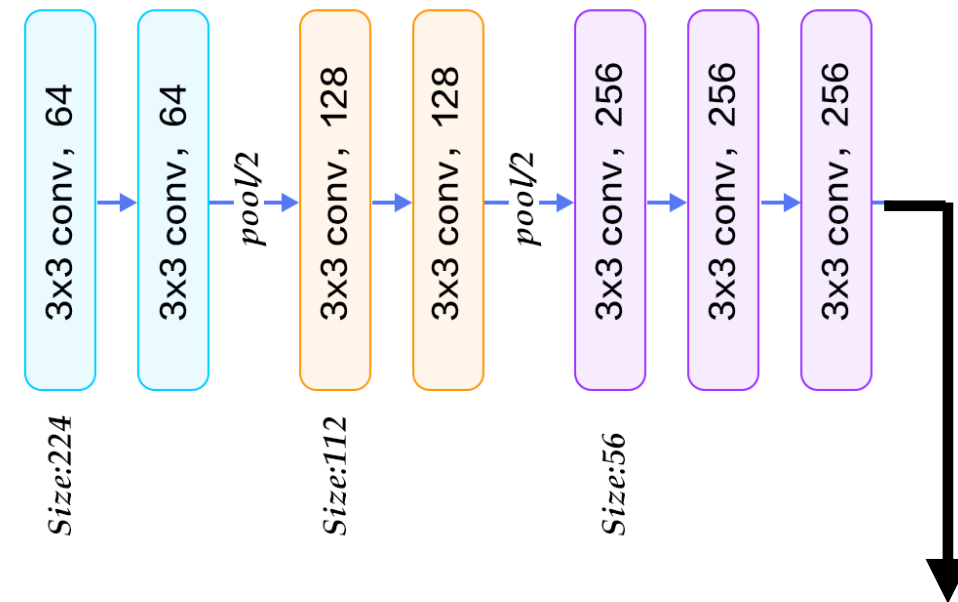
Mask R-CNN

Image

(resized to 224 x 224)

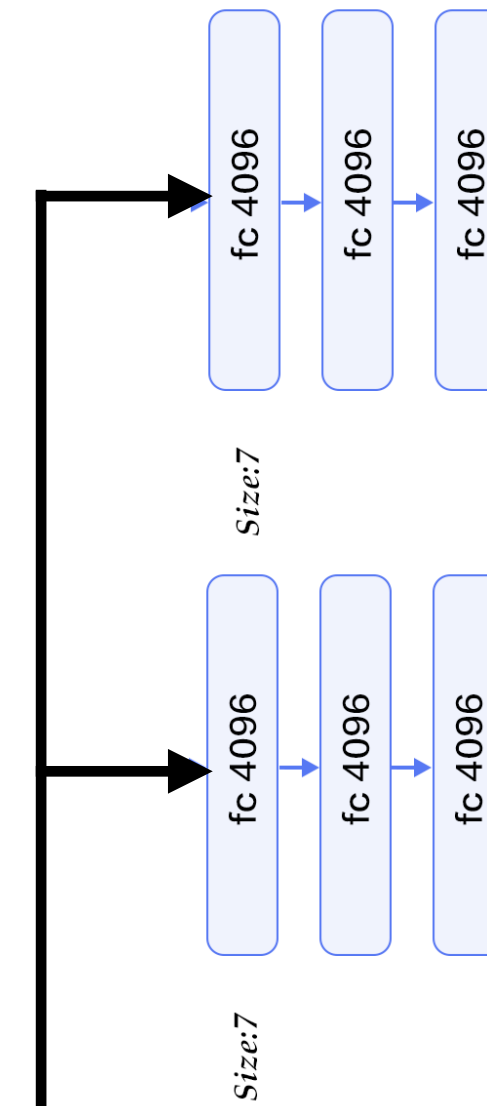
Proposal 1: (x, y, w, h)

Proposal 2: (x, y, w, h)



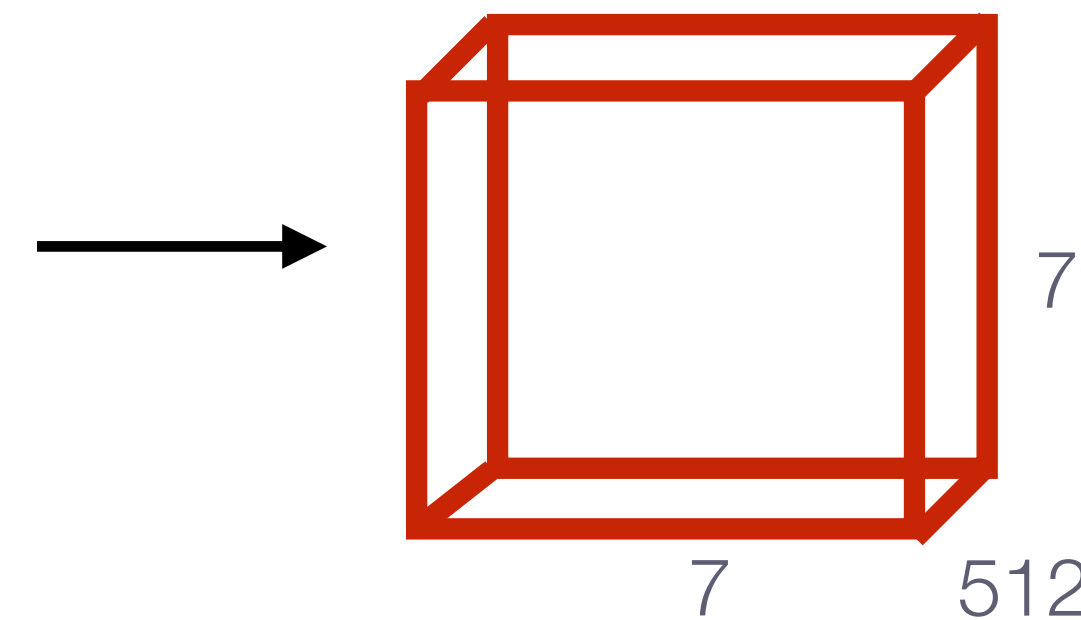
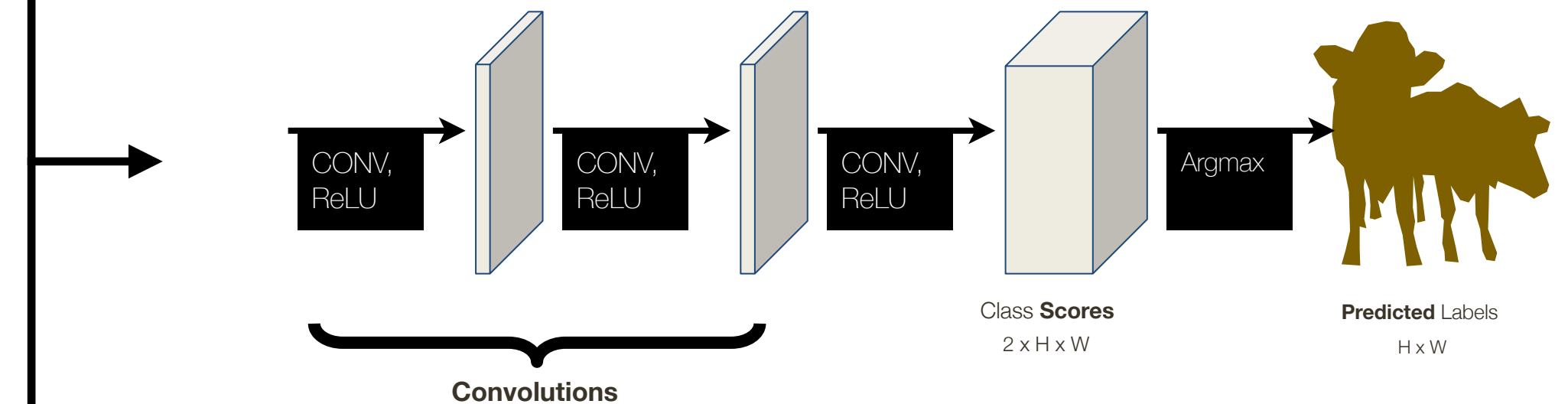
Classifier

(4096 x 80 categories + Softmax)

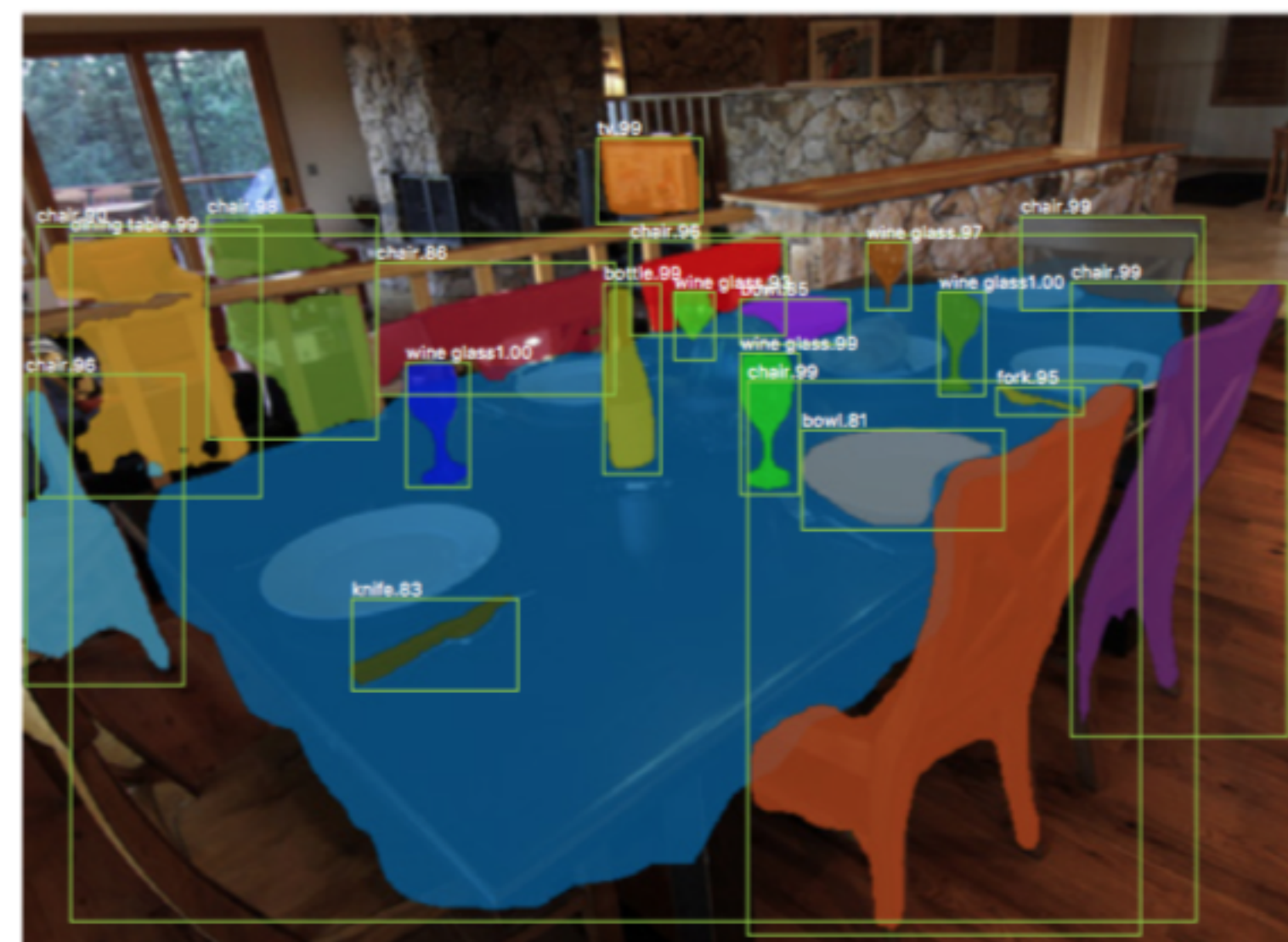
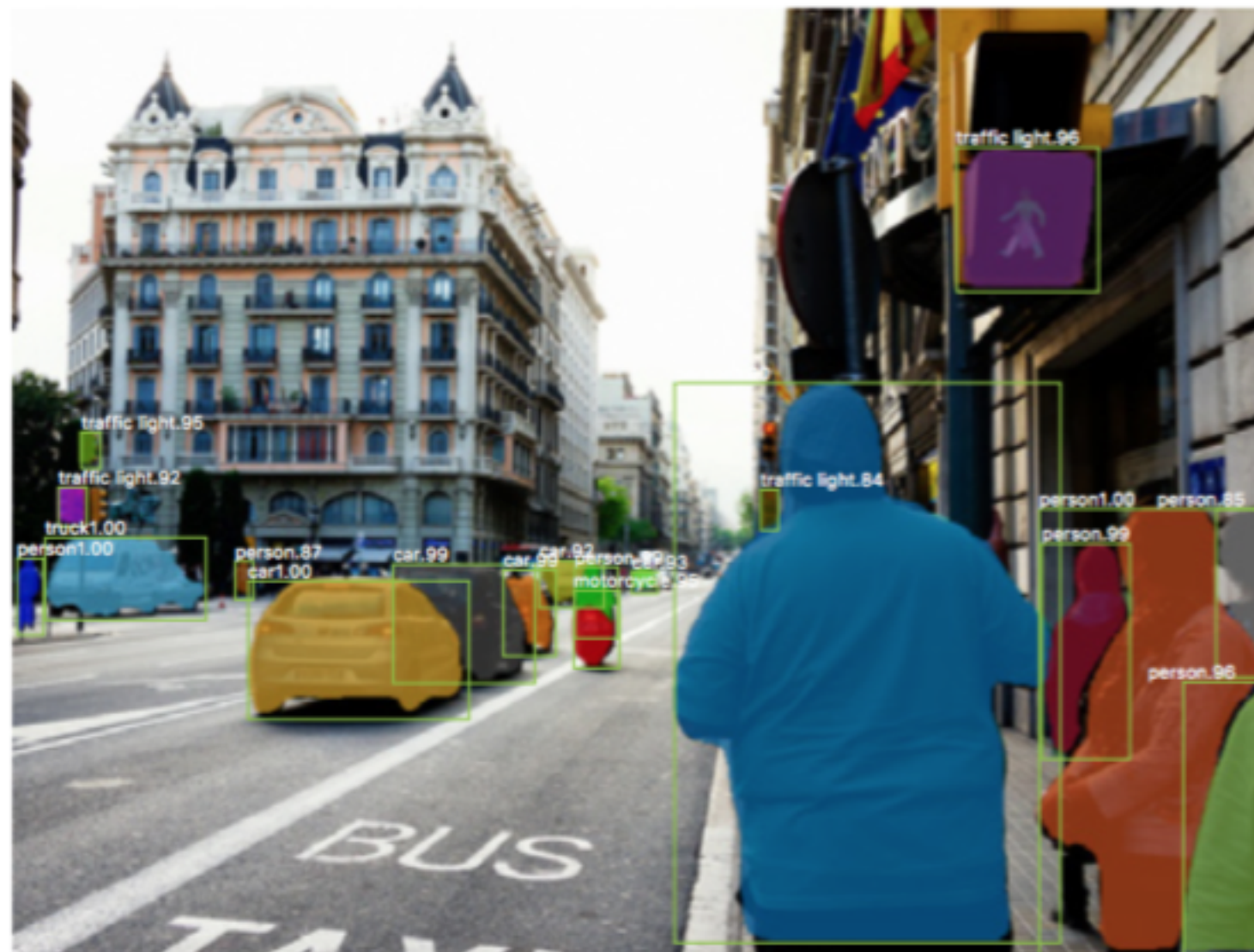


Regressor

(4096 x 4 coord + Linear)
delt x, delta y, delta w, delta h



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 (Concept)



Scientist

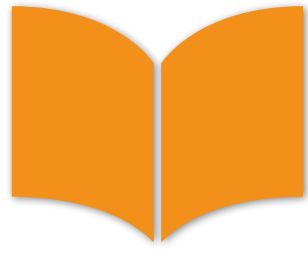


Lawyer

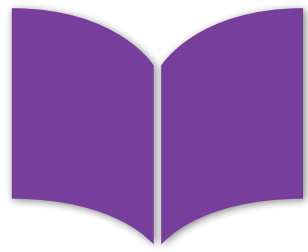


Businessman

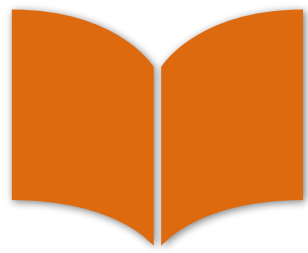
Attention (Concept)



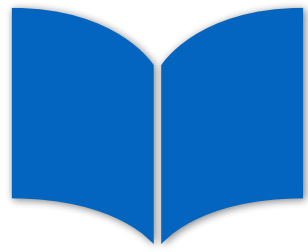
Popular Science



Business Journal



Nature



Law & Order



Scientist

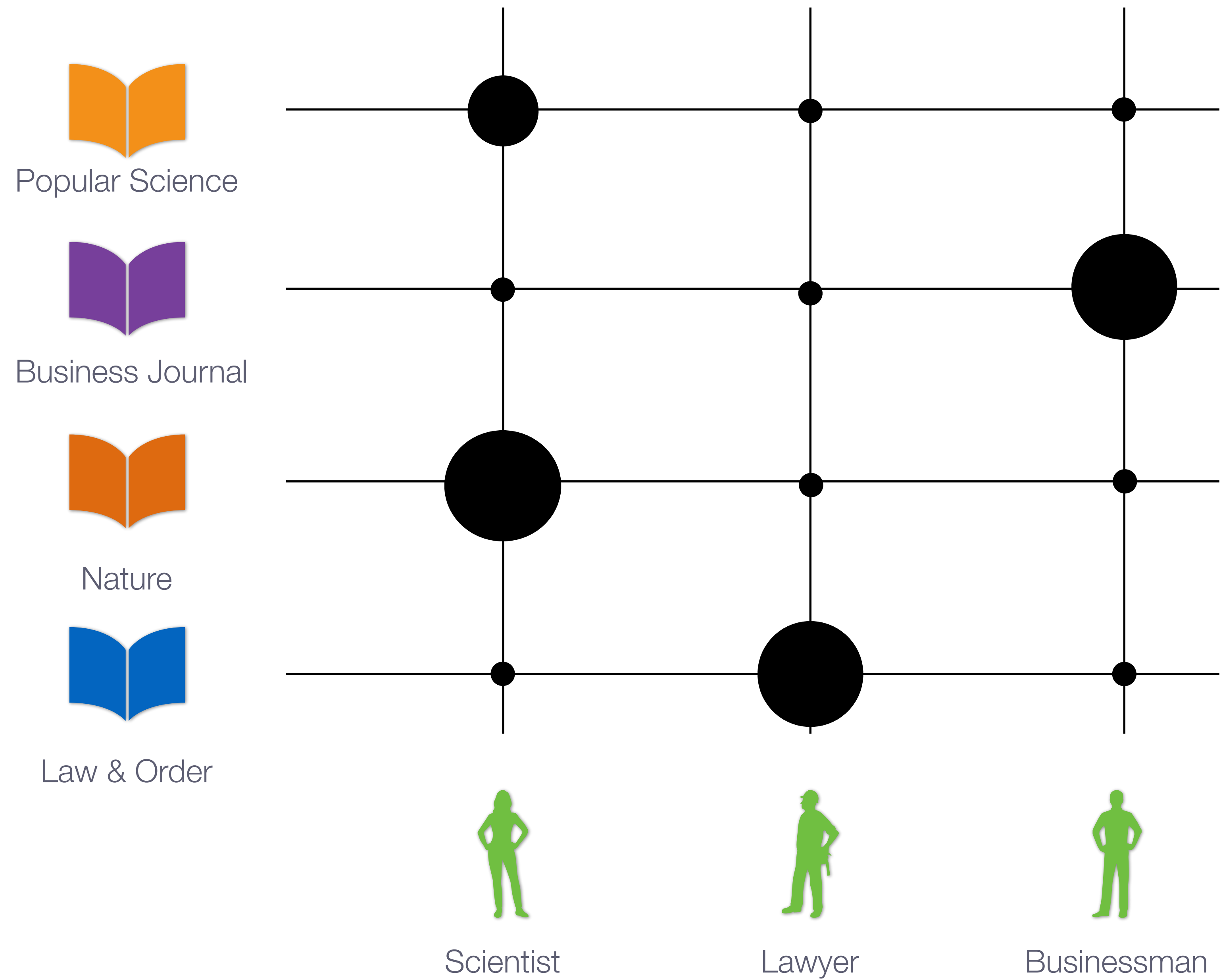


Lawyer

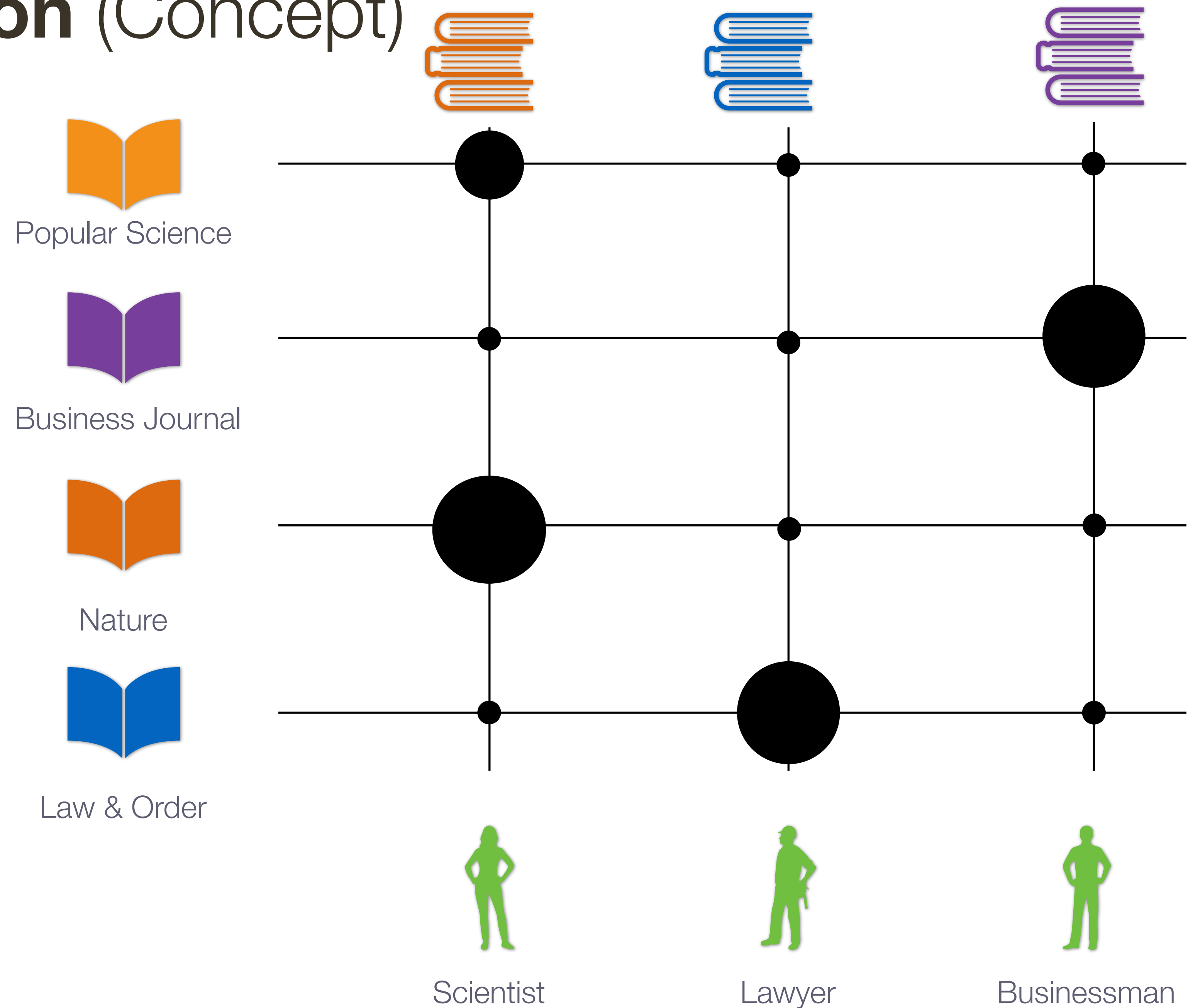


Businessman

Attention (Concept)



Attention (Concept)



The information is largely governed by affinity of people to books and content of the books themselves

Attention Layer

Inputs:

Query vectors: $\mathbf{Q}$ (Shape: $N_Q \times D_Q$)

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: $D_X \times D_V$)

Computation:

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: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

X_1

X_2

X_3

Q_1

Q_2

Q_3

Q_4

Attention Layer

Inputs:

Query vectors: $\mathbf{Q}$ (Shape: $N_Q \times D_Q$)

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: $D_X \times D_V$)

Computation:

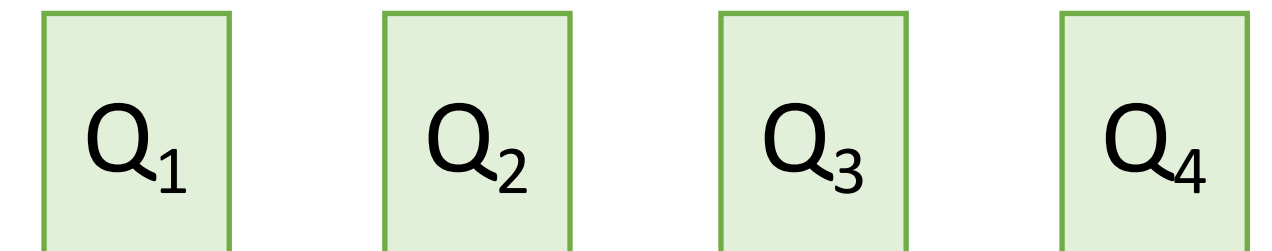
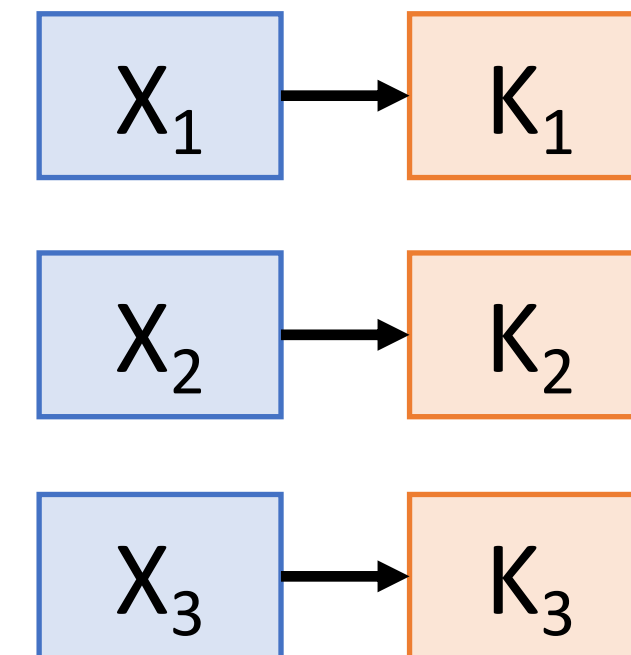
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Attention Layer

Inputs:

Query vectors: $\mathbf{Q}$ (Shape: $N_Q \times D_Q$)

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: $D_X \times D_V$)

Computation:

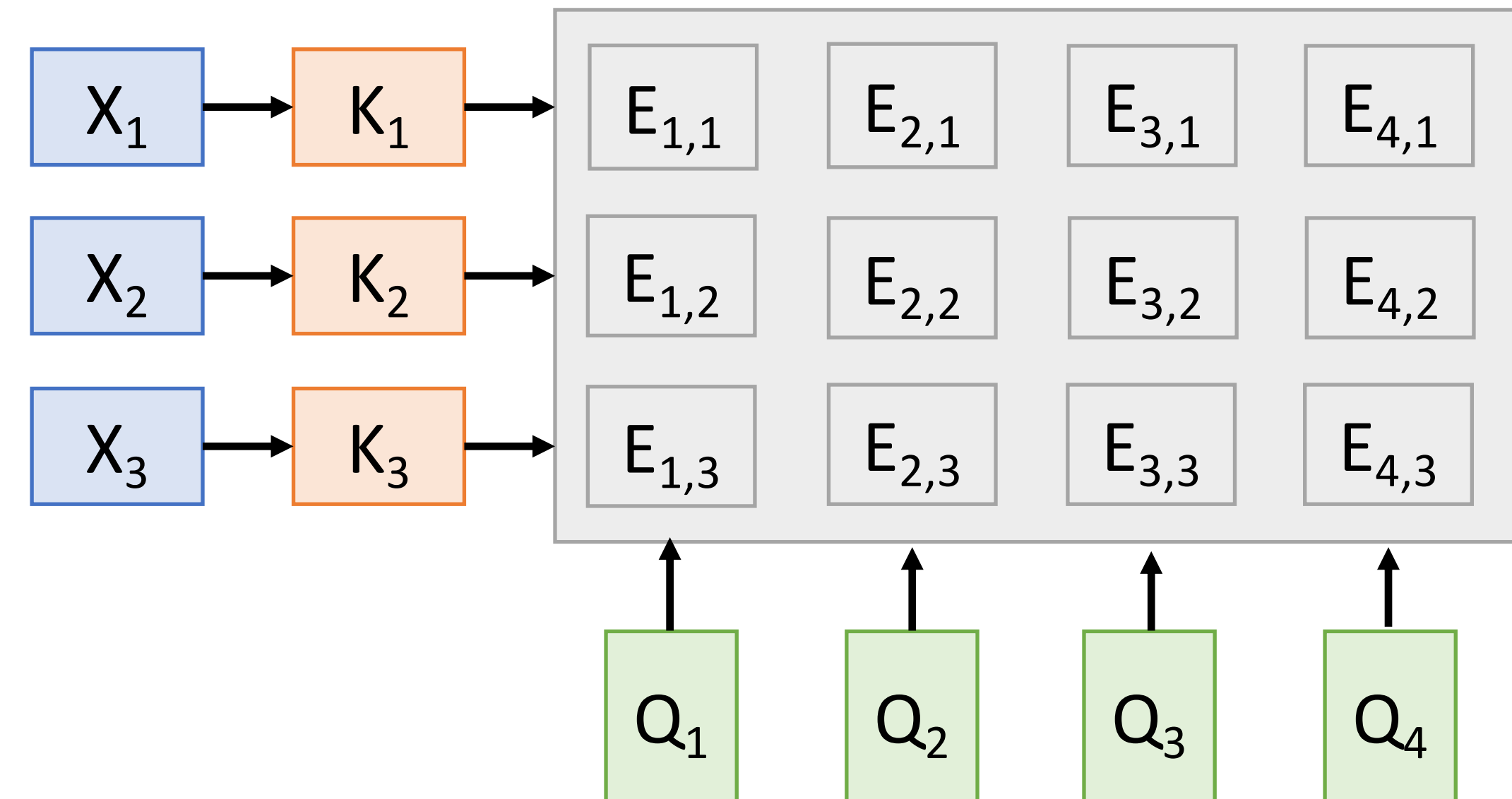
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Attention Layer

Inputs:

Query vectors: $\mathbf{Q}$ (Shape: $N_Q \times D_Q$)

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: $D_X \times D_V$)

Computation:

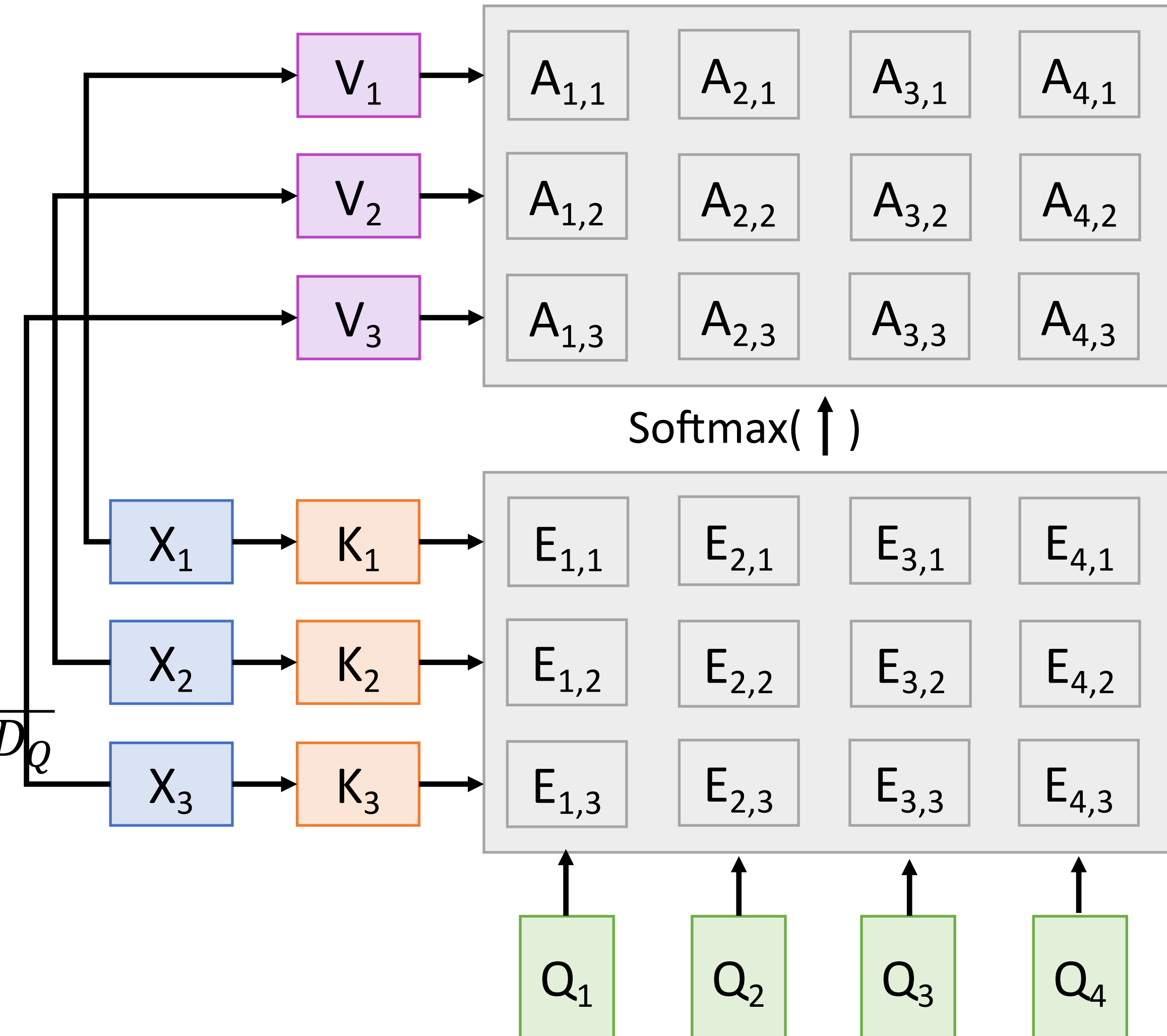
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Attention Layer

Inputs:

Query vectors: $\mathbf{Q}$ (Shape: $N_Q \times D_Q$)

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: $D_X \times D_V$)

Computation:

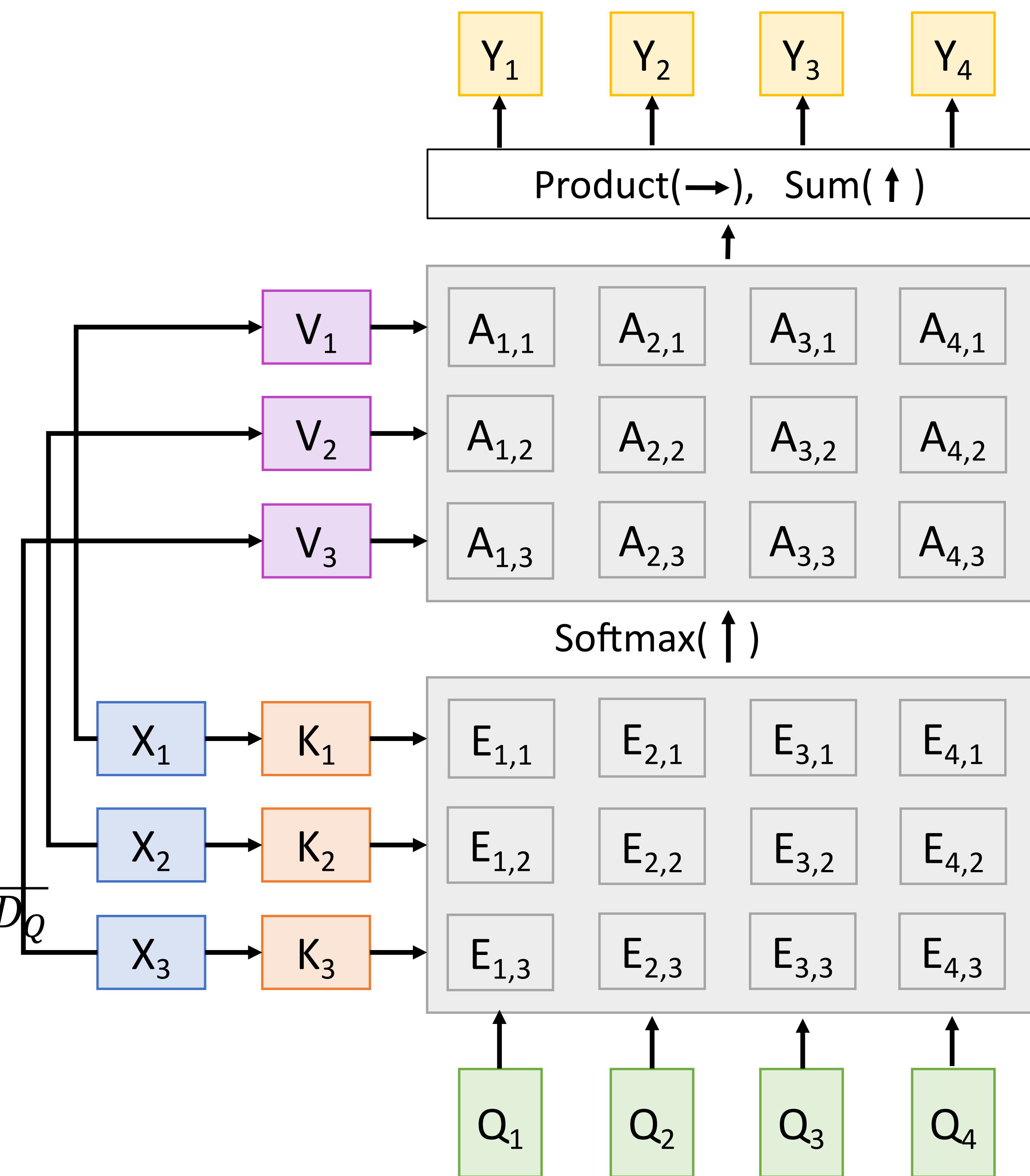
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-attention Layer

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

Inputs:

Query vectors: $\mathbf{Q}$ (Shape: $N_Q \times D_Q$)

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: $D_X \times D_V$)

Computation:

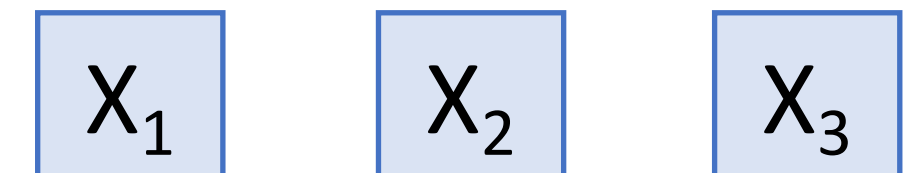
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-attention Layer

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

Inputs:

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: $D_x \times D_Q$)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{XW}_Q$

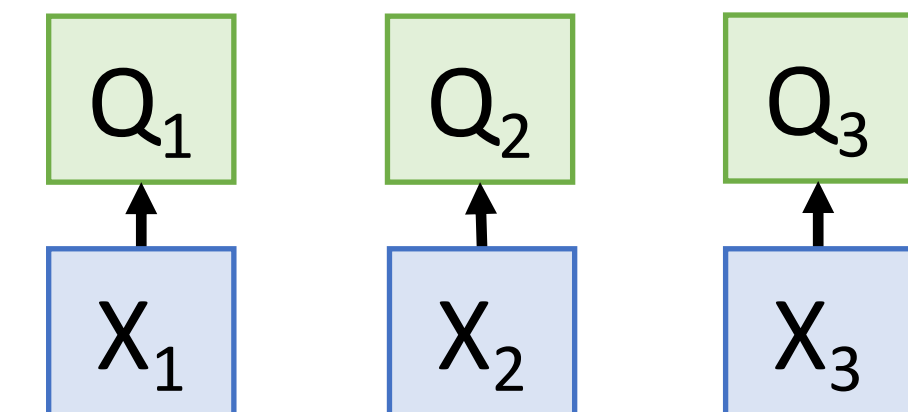
Key vectors: $\mathbf{K} = \mathbf{XW}_k$ (Shape: $N_x \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_v$ (Shape: $N_x \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_x \times N_x$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_x \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-attention Layer

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

Inputs:

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: $D_X \times D_Q$)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{XW}_Q$

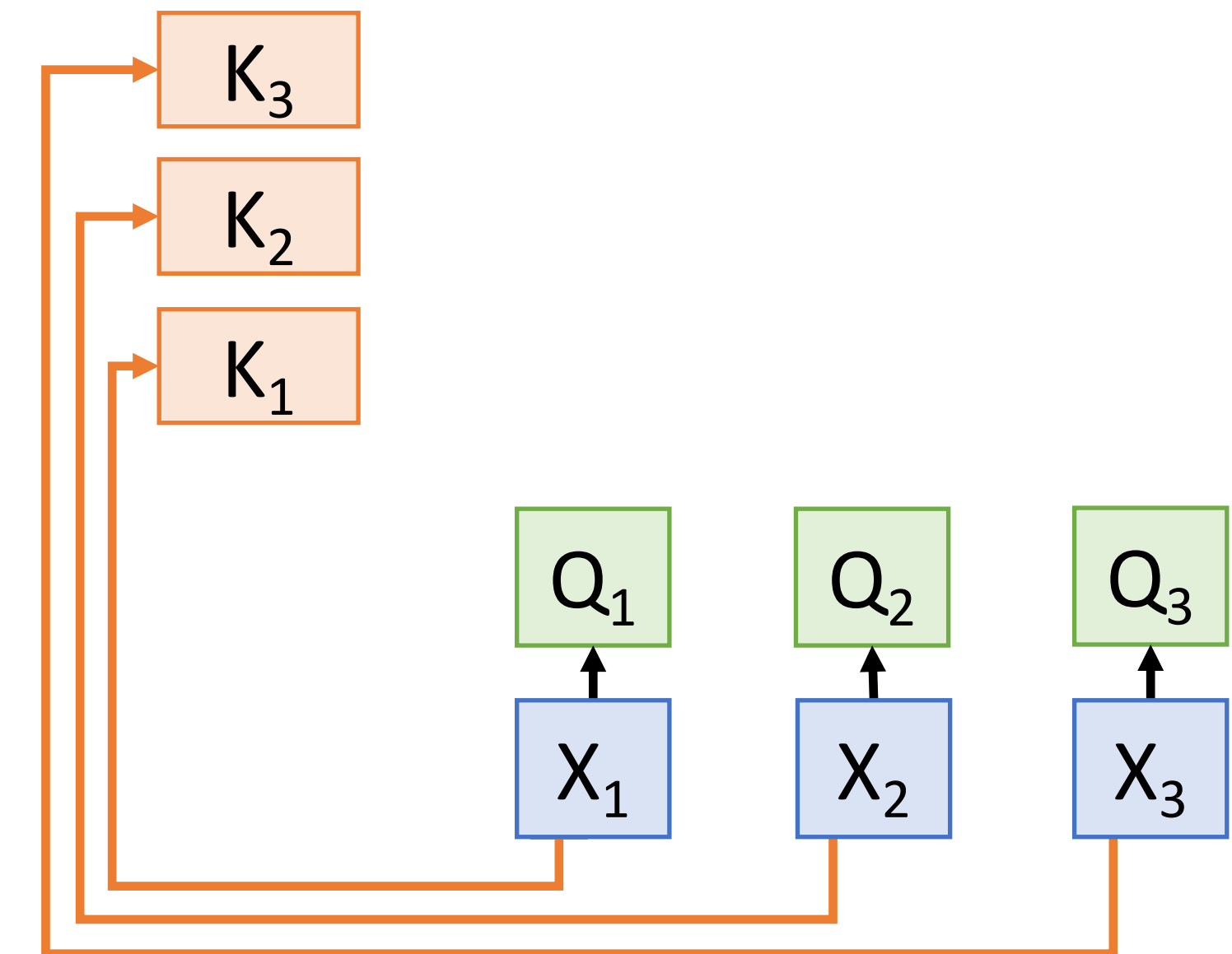
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_X \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-attention Layer

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

Inputs:

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: $D_X \times D_Q$)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{XW}_Q$

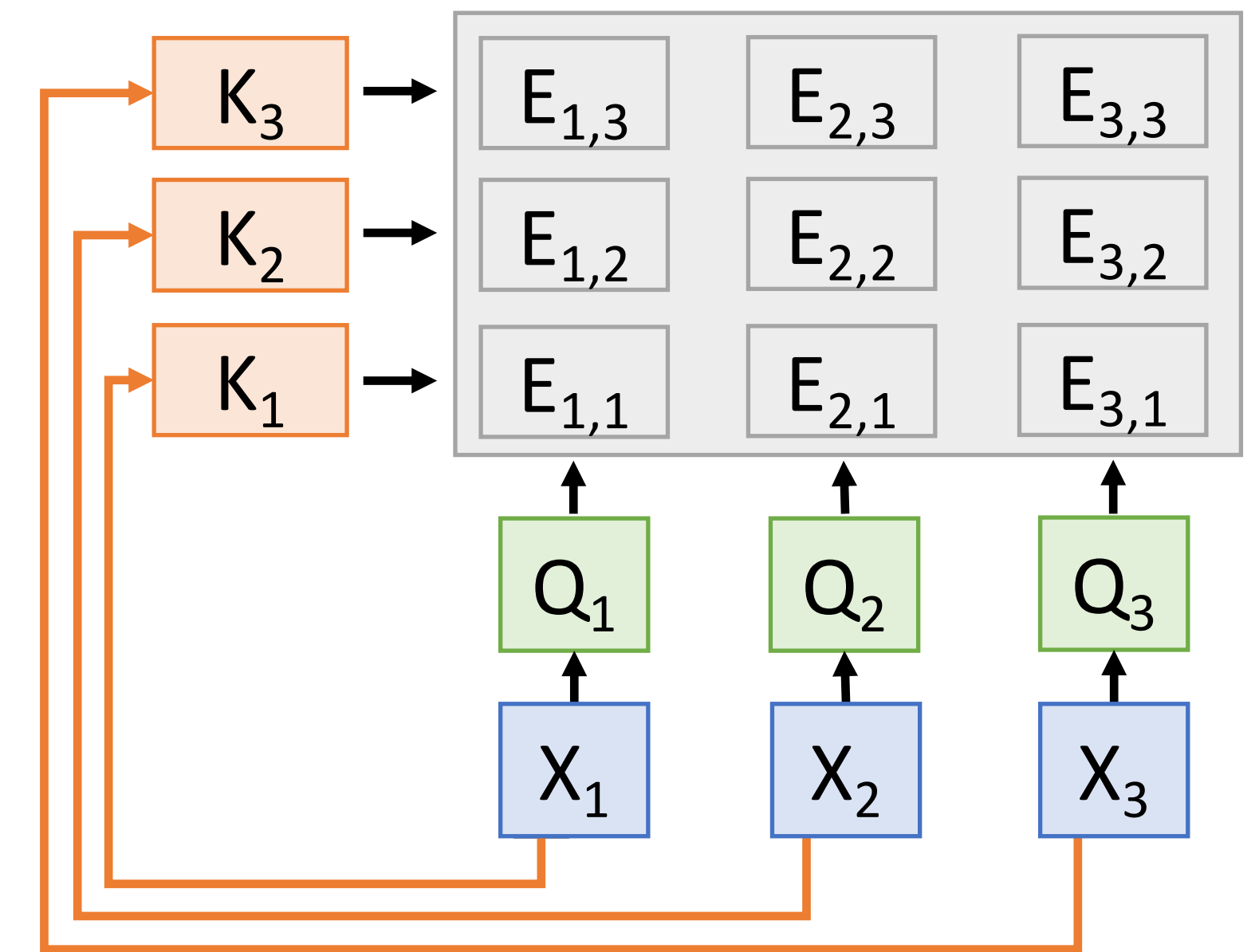
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_X \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-attention Layer

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

Inputs:

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: $D_x \times D_Q$)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$

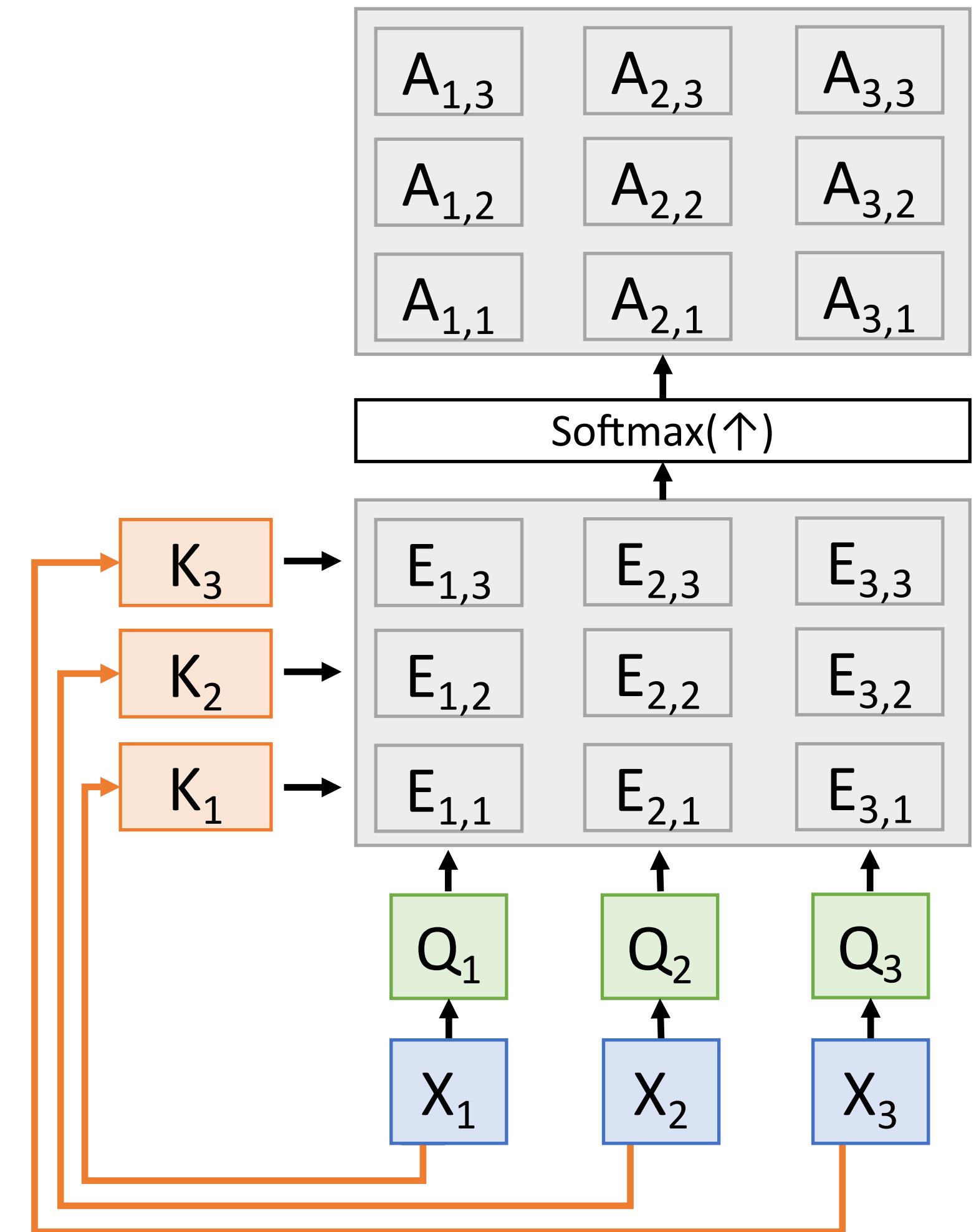
Key vectors: $\mathbf{K} = \mathbf{X}\mathbf{W}_k$ (Shape: $N_x \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{X}\mathbf{W}_v$ (Shape: $N_x \times D_V$)

Similarities: $\mathbf{E} = \mathbf{Q}\mathbf{K}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_x \times N_x$)

Output vectors: $\mathbf{Y} = \mathbf{A}\mathbf{V}$ (Shape: $N_x \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-attention Layer

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

Inputs:

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: $D_X \times D_Q$)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{XW}_Q$

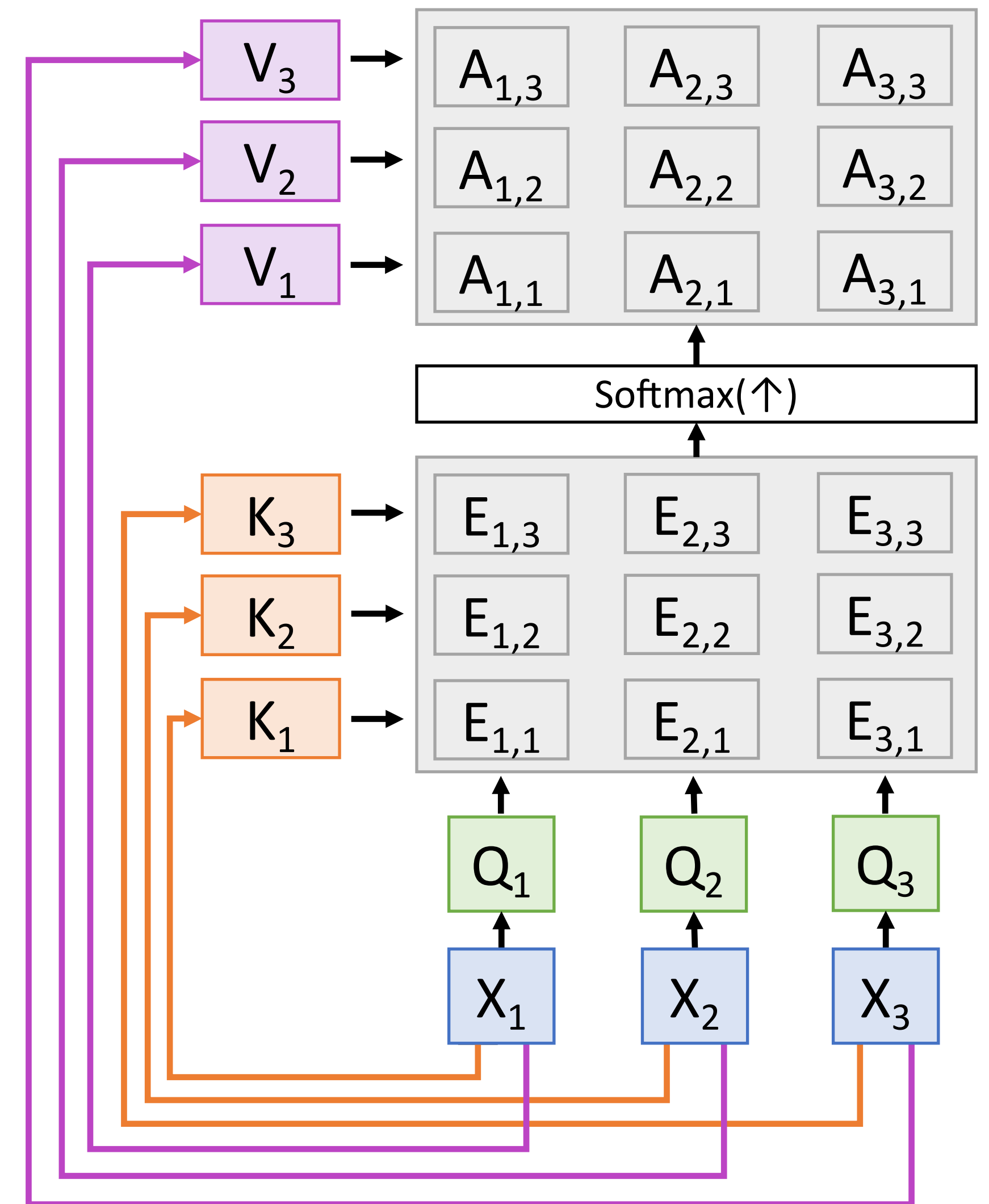
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_X \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-attention Layer

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

Inputs:

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: $D_X \times D_Q$)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{XW}_Q$

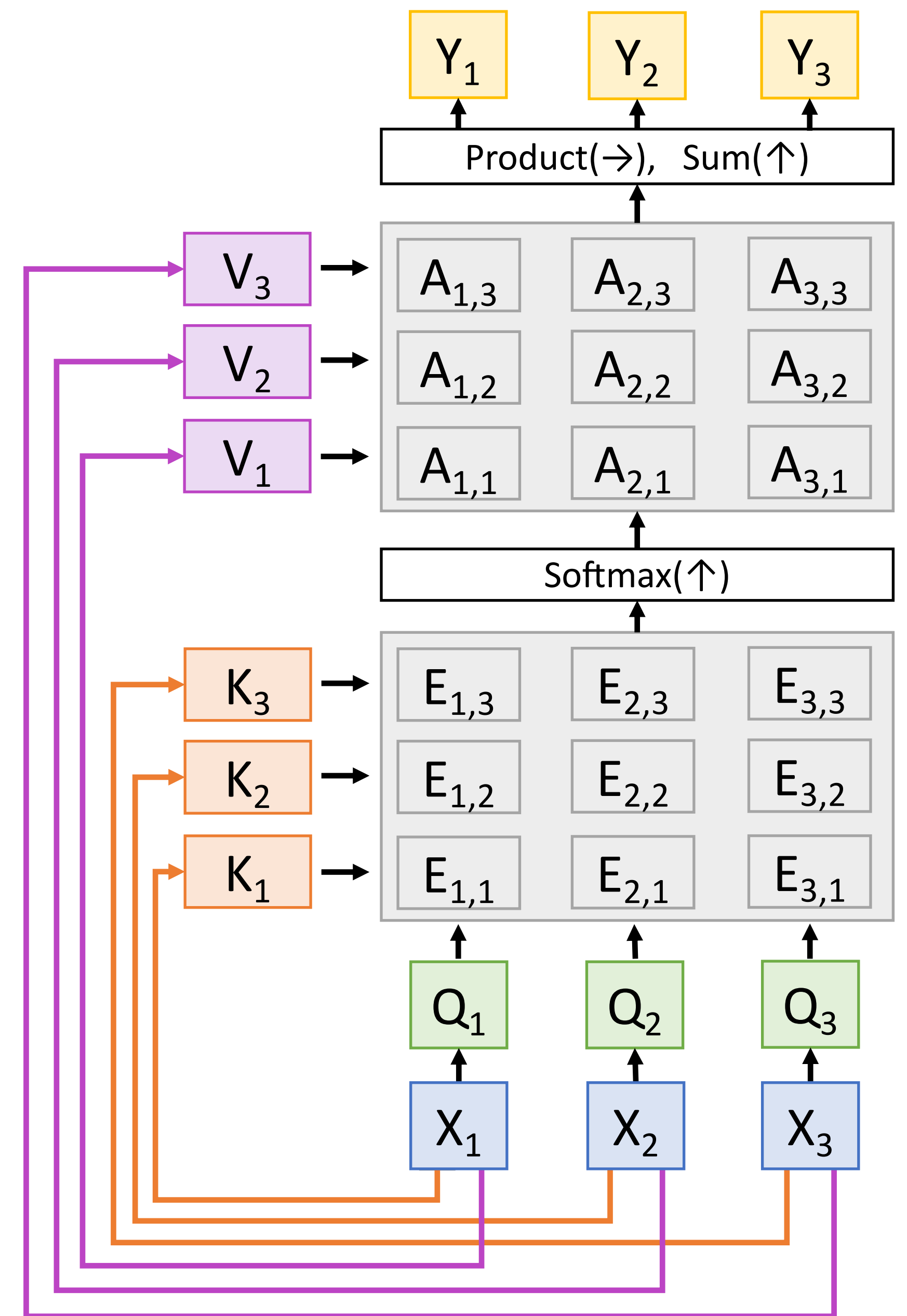
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_X \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-attention Layer

Inputs:

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: $D_x \times D_Q$)

Computation:

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: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_x \times N_x$)

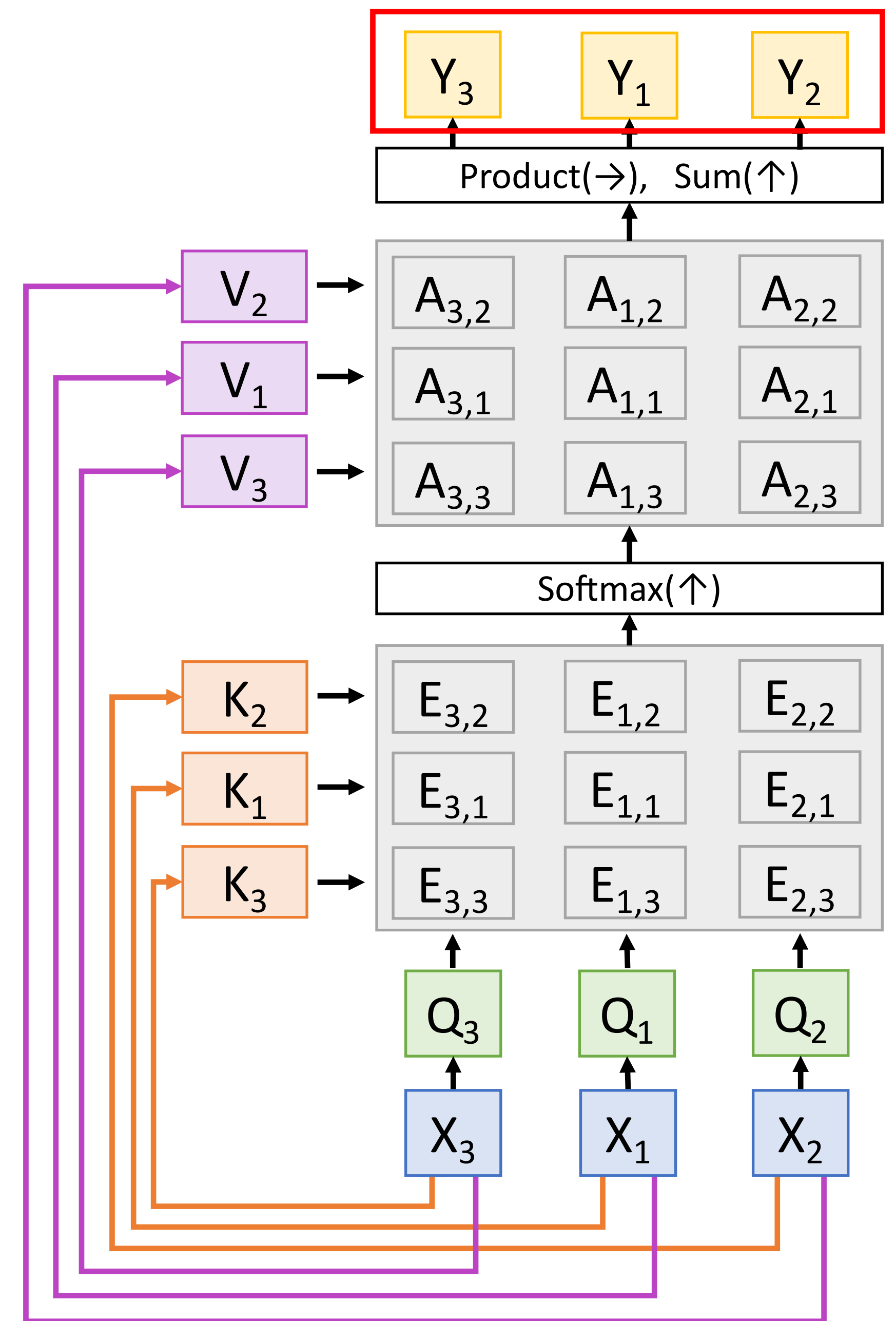
Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_x \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

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



Multi-head Self-attention Layer

Inputs:

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: $D_X \times D_Q$)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{XW}_Q$

Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

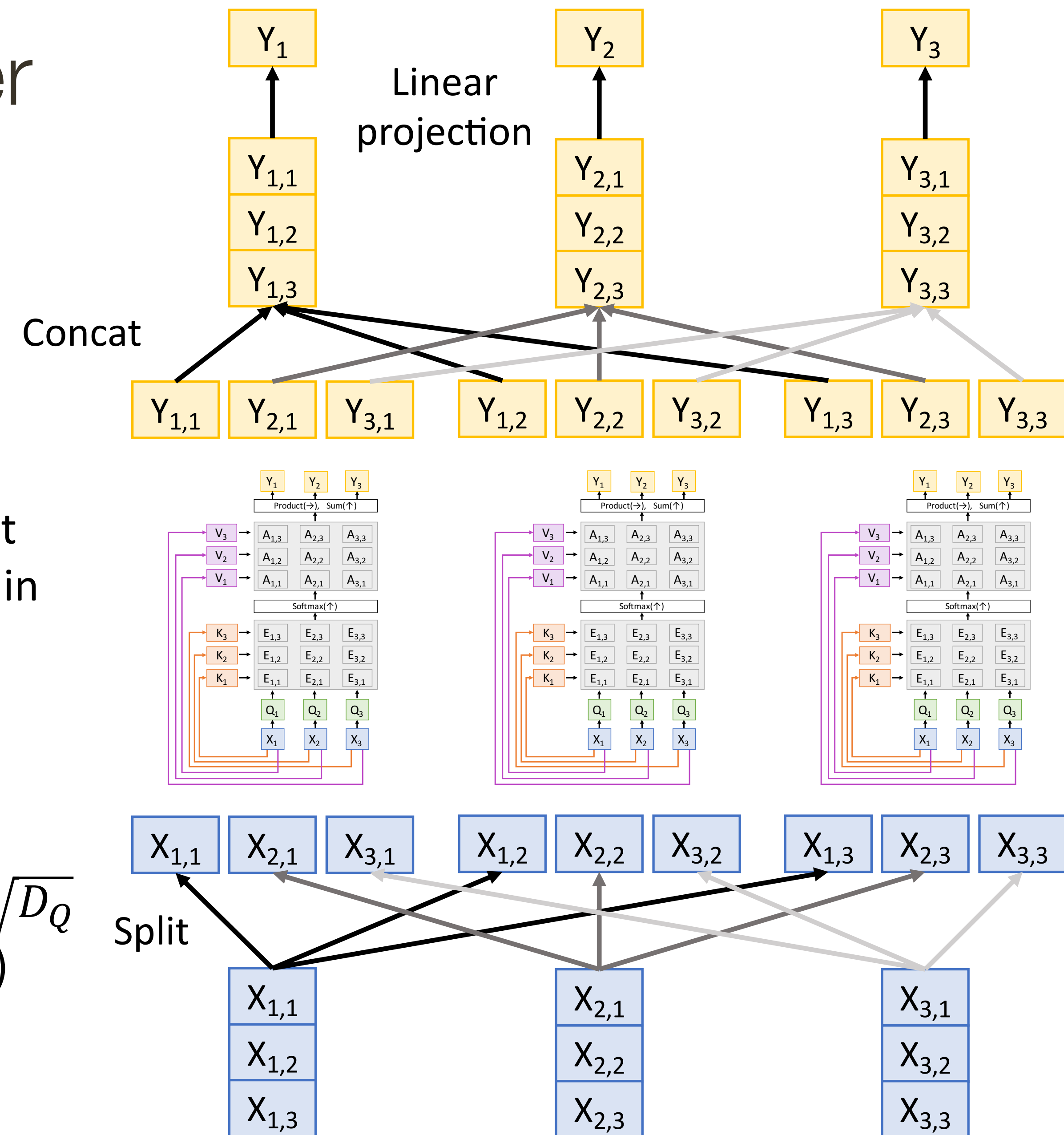
Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_X \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

Use H independent
“Attention Heads” in
parallel

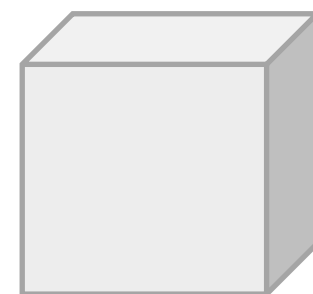
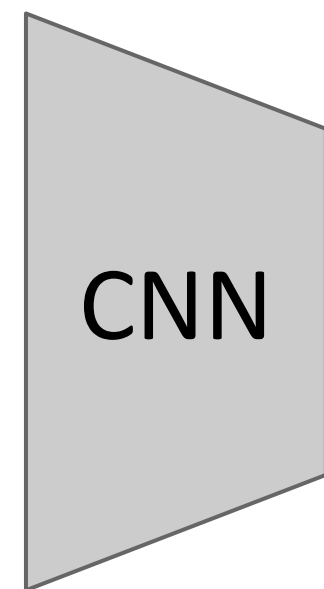


CNN with Self-attention

Input Image

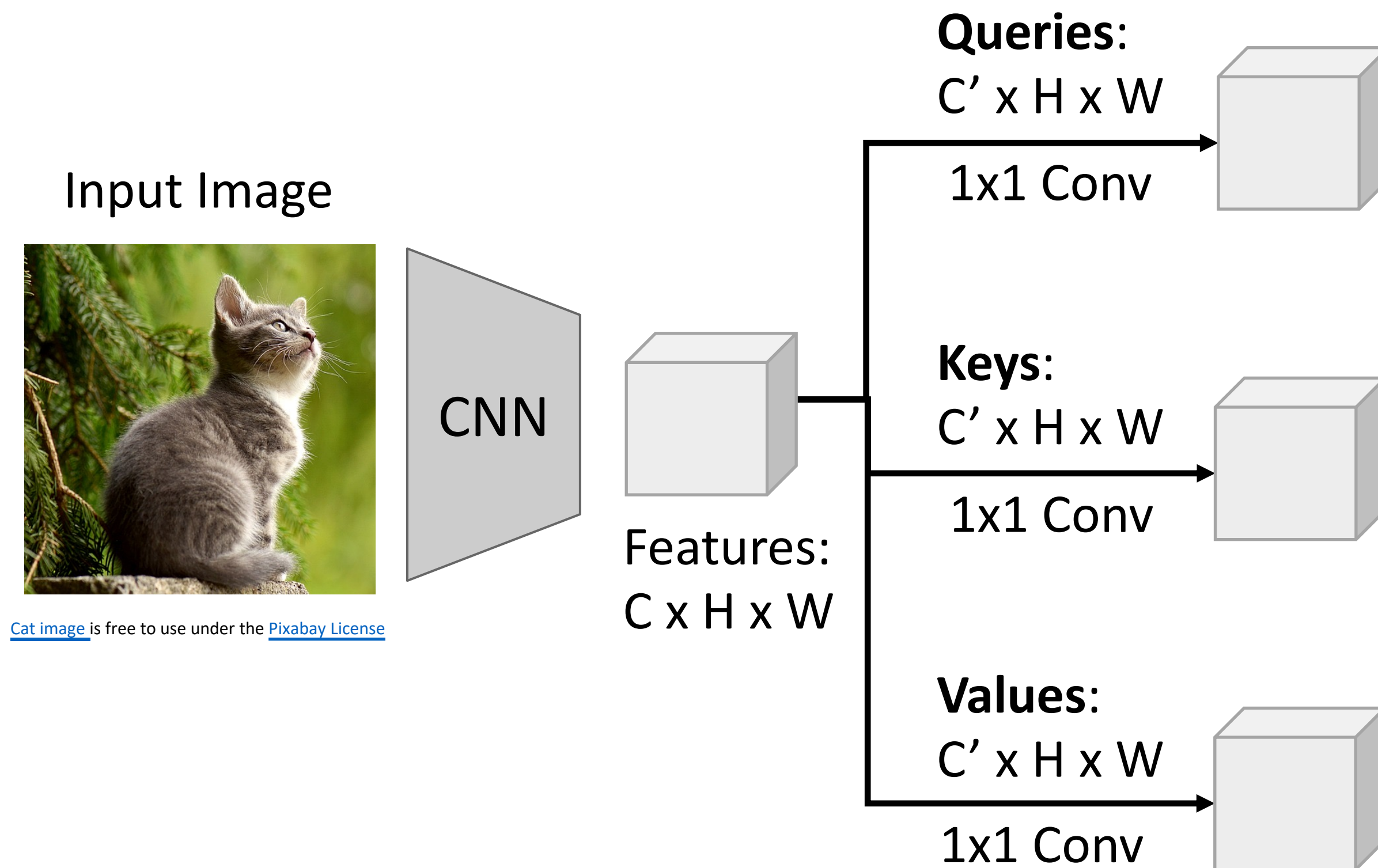


[Cat image](#) is free to use under the [Pixabay License](#)

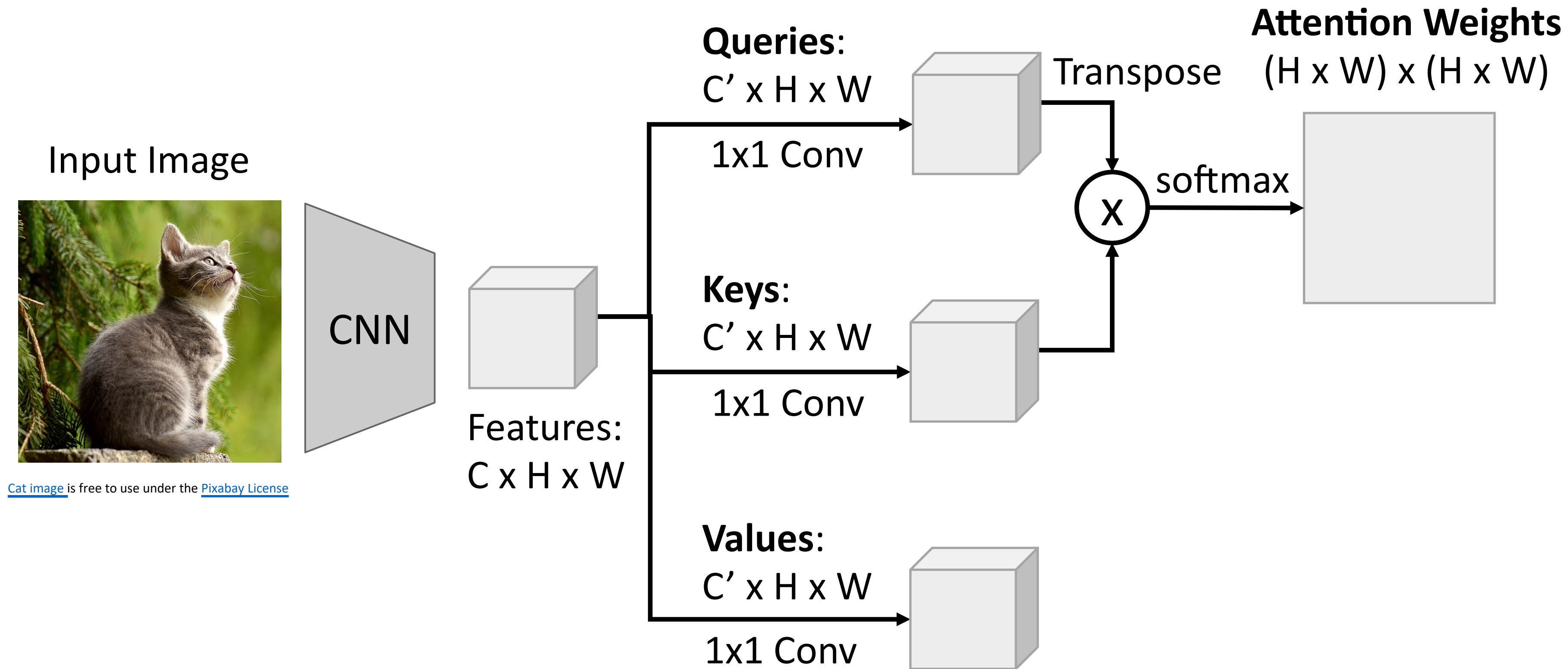


Features:
 $C \times H \times W$

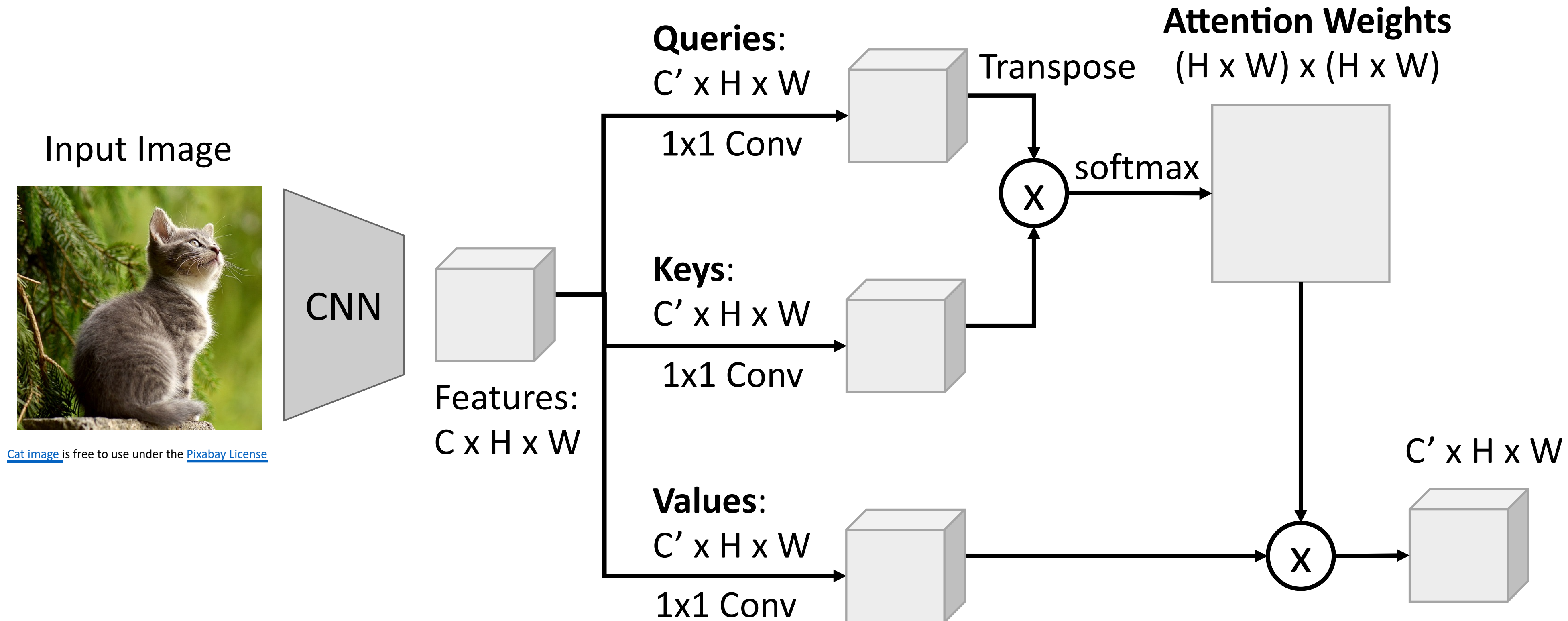
CNN with Self-attention



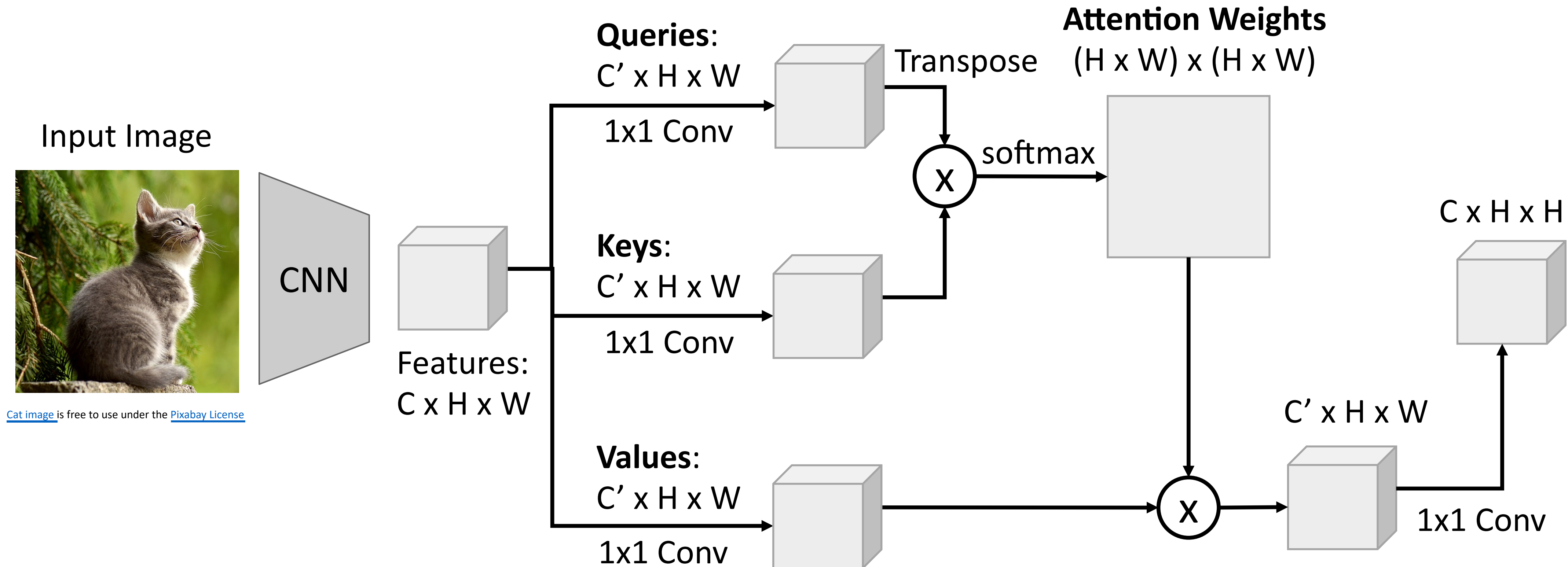
CNN with Self-attention



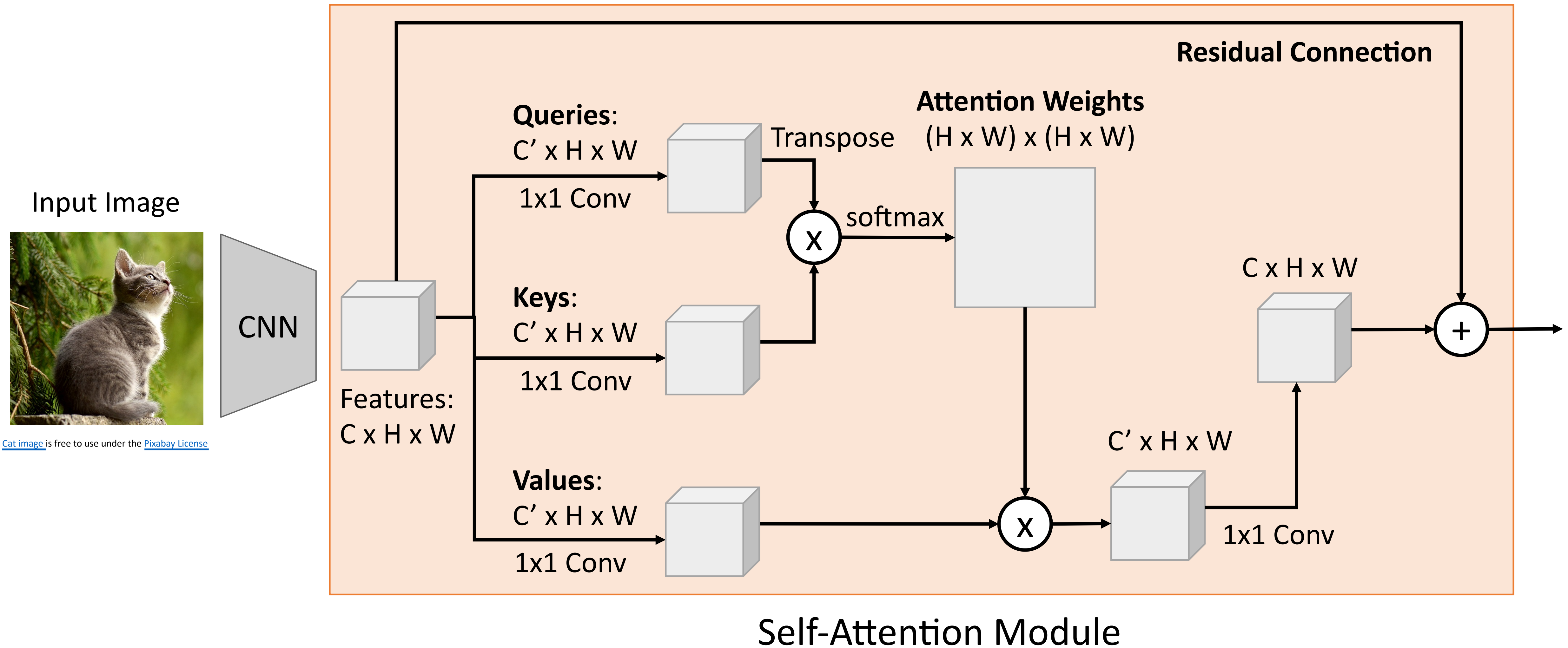
CNN with Self-attention



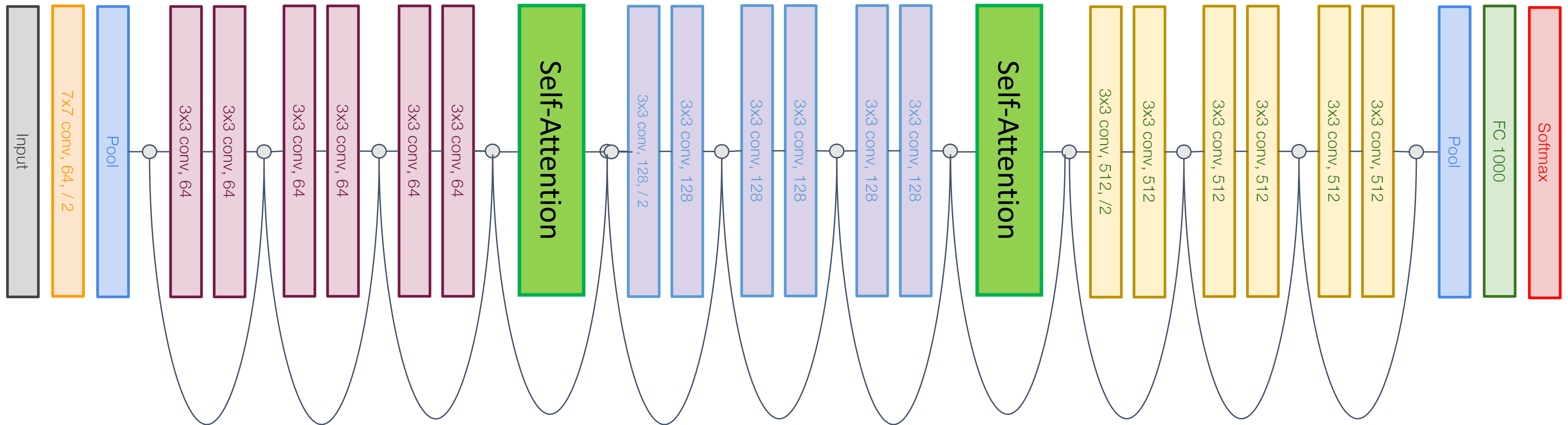
CNN with Self-attention



CNN with Self-attention



Attention with Existing CNNs



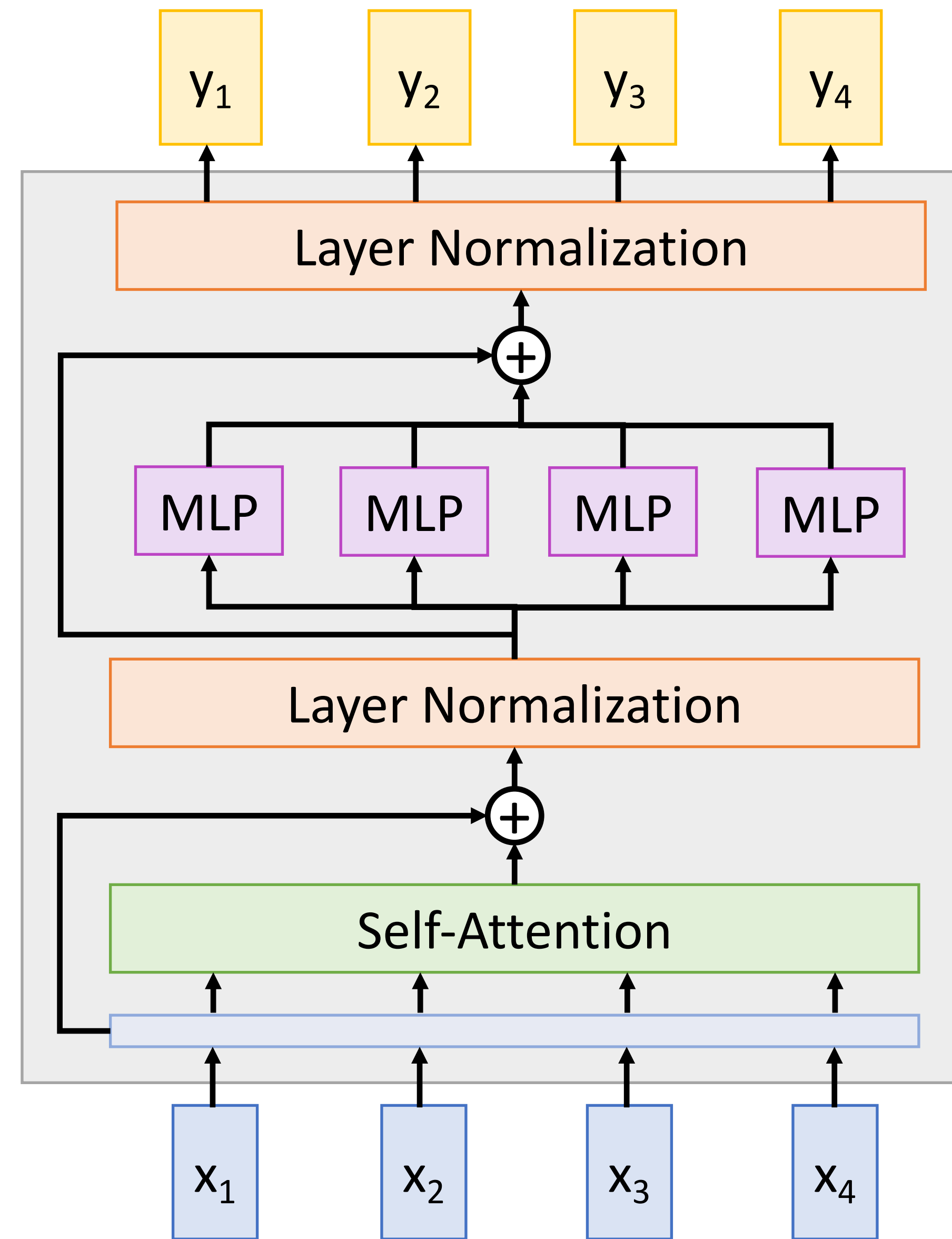
Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Wang et al, "Non-local Neural Networks", CVPR 2018

Transformer

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

Vectors only communicate via (multiheaded) self-attention



Transformer

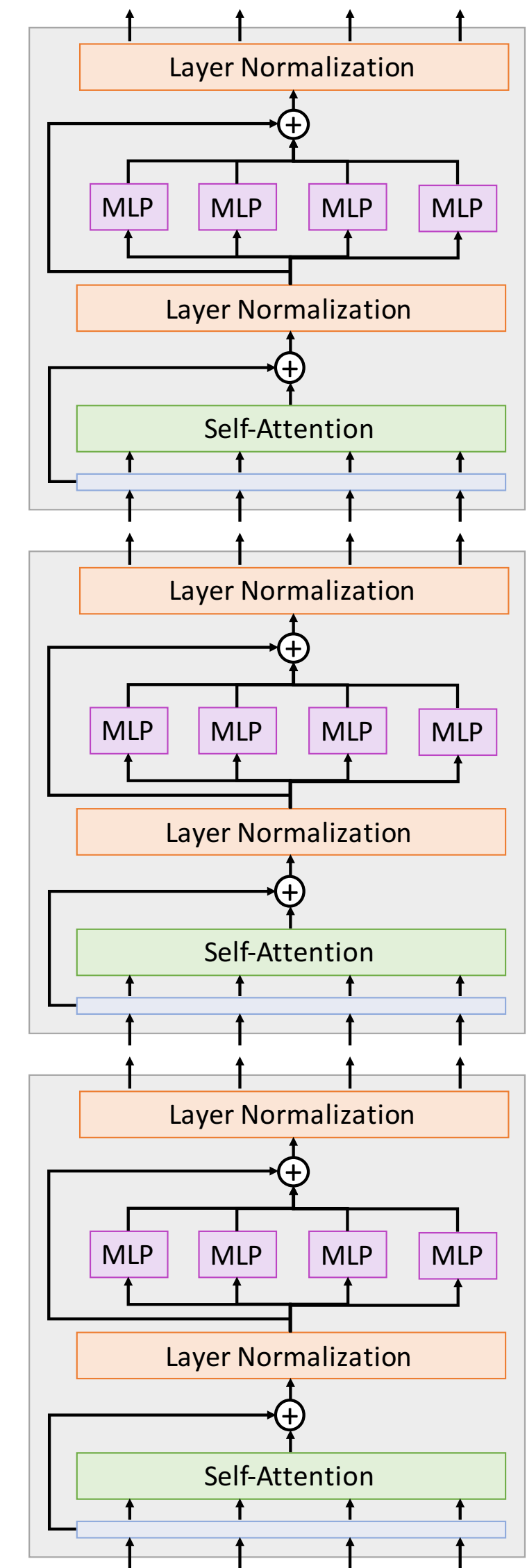
Transformer Block:

Input: Set of vectors x

Output: Set of vectors y

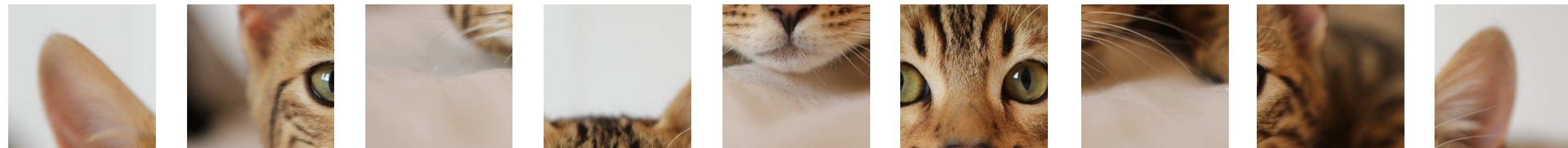
Hyperparameters:

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



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

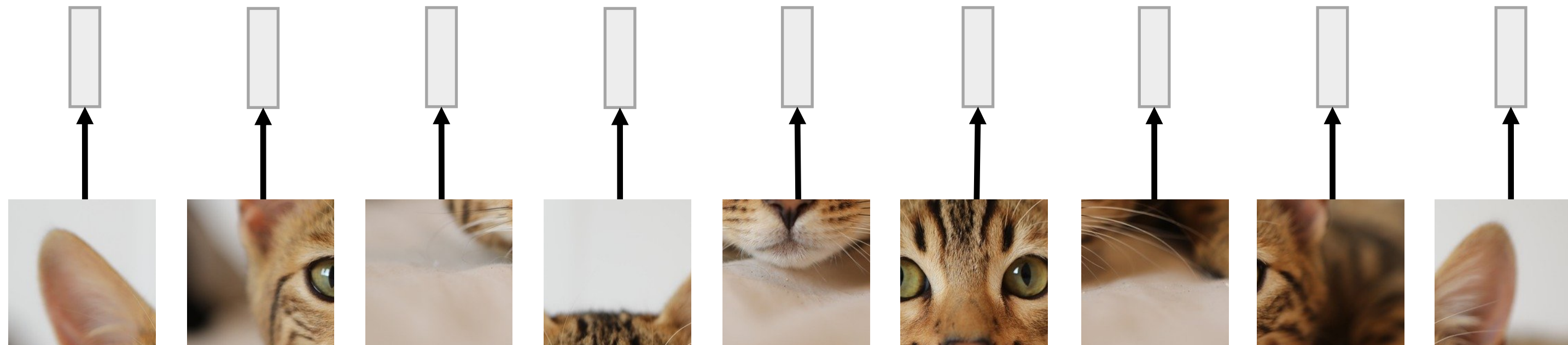
[Cat image](#) is free for commercial
use under a [Pixabay license](#)

[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

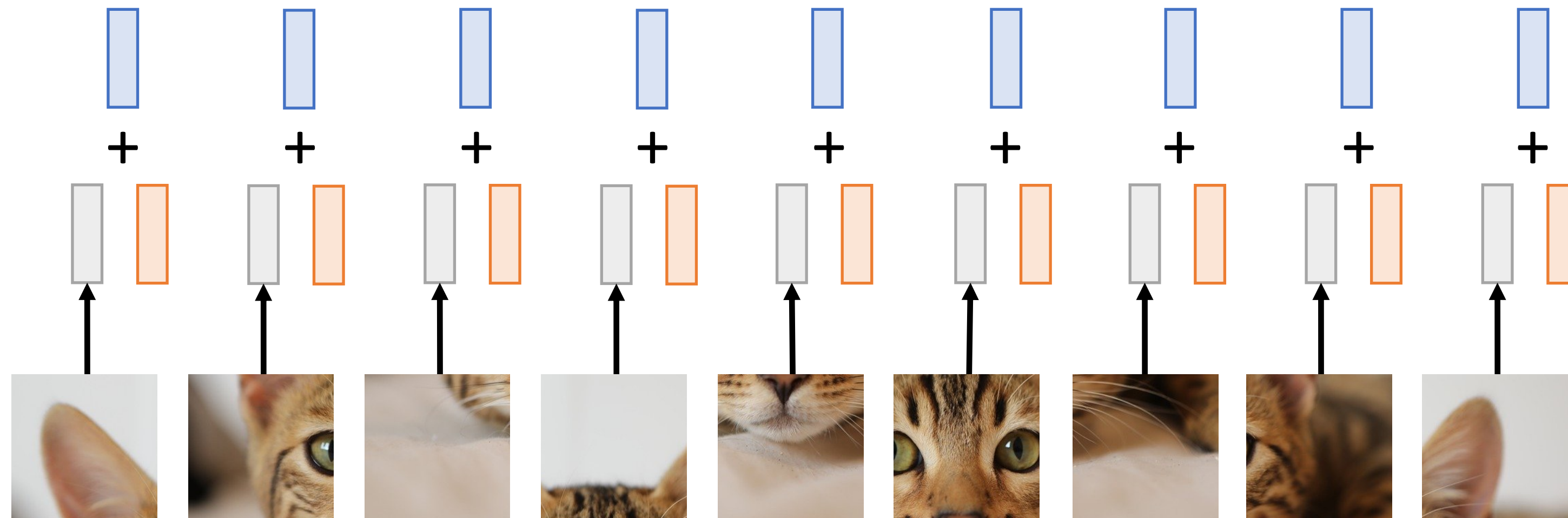
[Cat image](#) is free for commercial
use under a [Pixabay license](#)

Transformer on Image Patches

Add positional
embedding: learned D-
dim vector per position

Linear projection to
D-dimensional vector

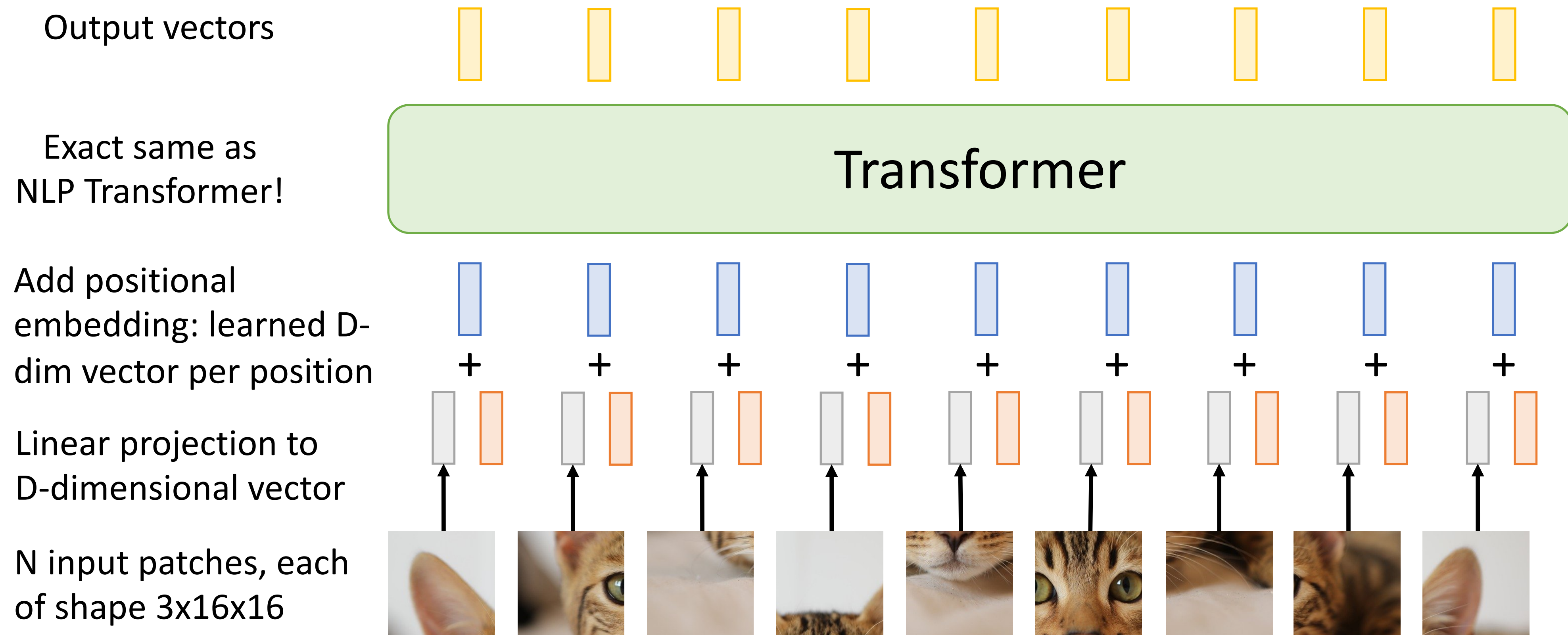
N input patches, each
of shape 3x16x16



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

[Cat image](#) is free for commercial
use under a [Pixabay license](#)

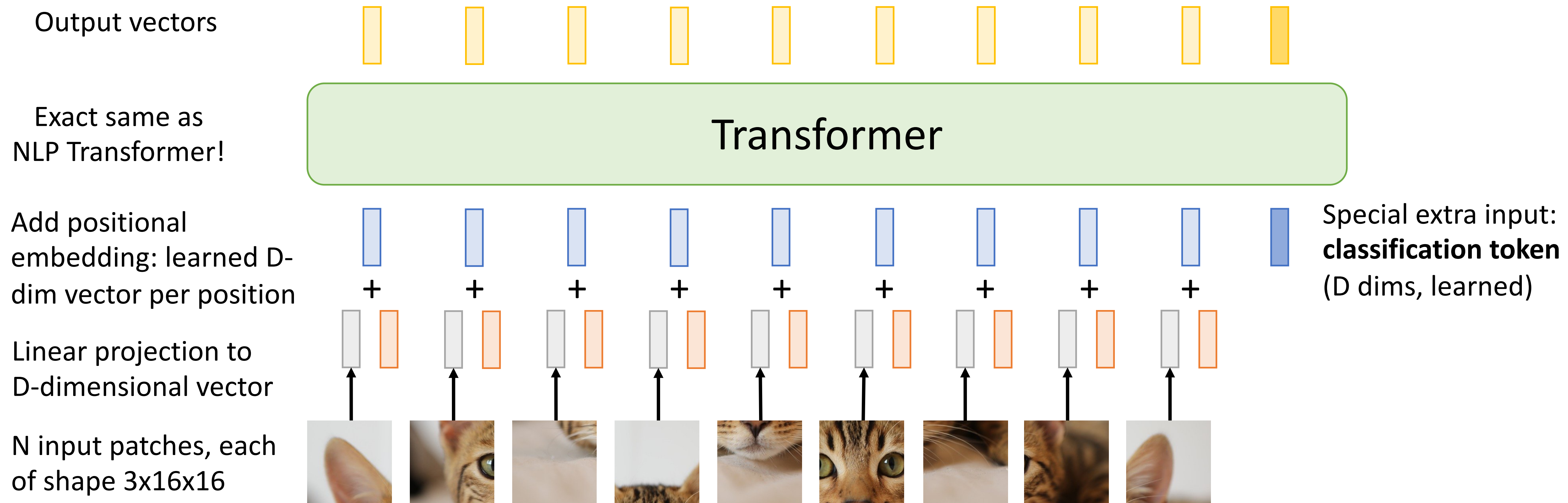
Transformer on Image Patches



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

[Cat image](#) is free for commercial use under a [Pixabay license](#)

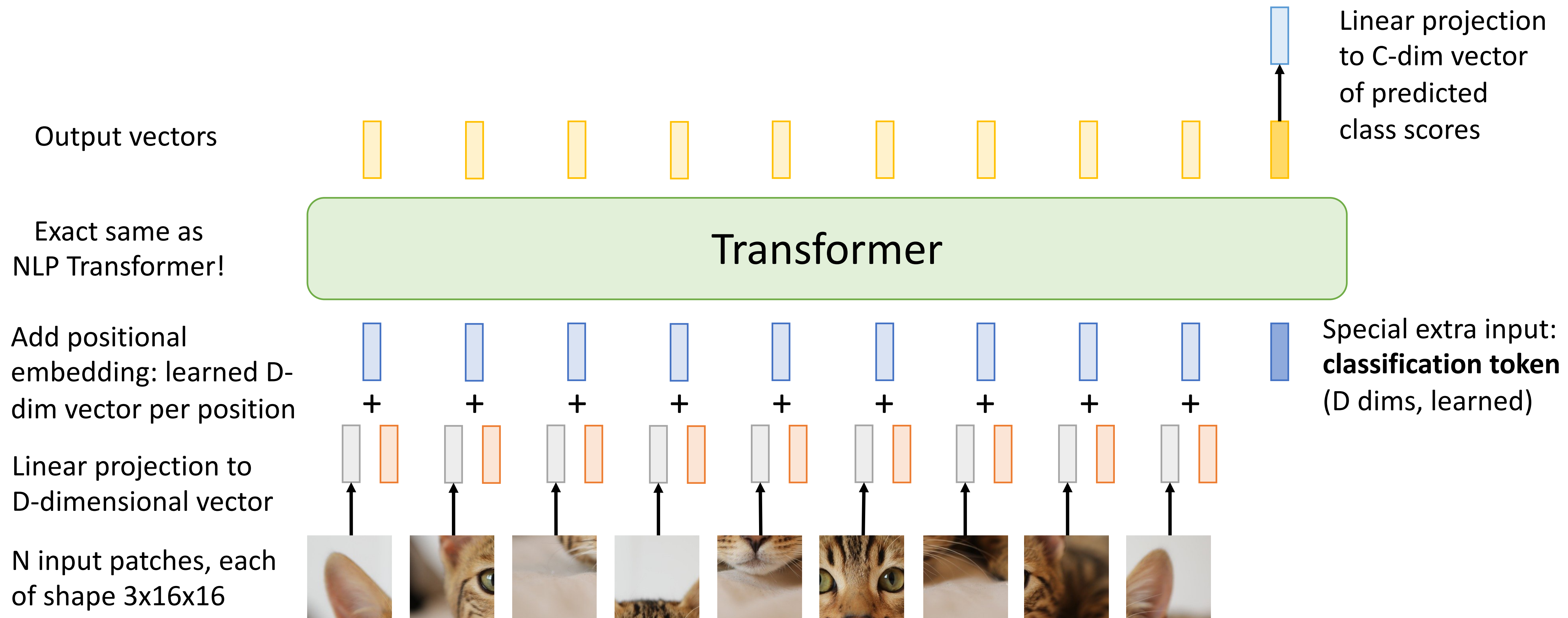
Transformer on Image Patches



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

[Cat image](#) is free for commercial use under a [Pixabay license](#)

Transformer on Image Patches



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

[Cat image](#) is free for commercial use under a [Pixabay license](#)

Vision Transformer (ViT)

Computer vision model
with no convolutions!

Output vectors



Linear projection
to C-dim vector
of predicted
class scores

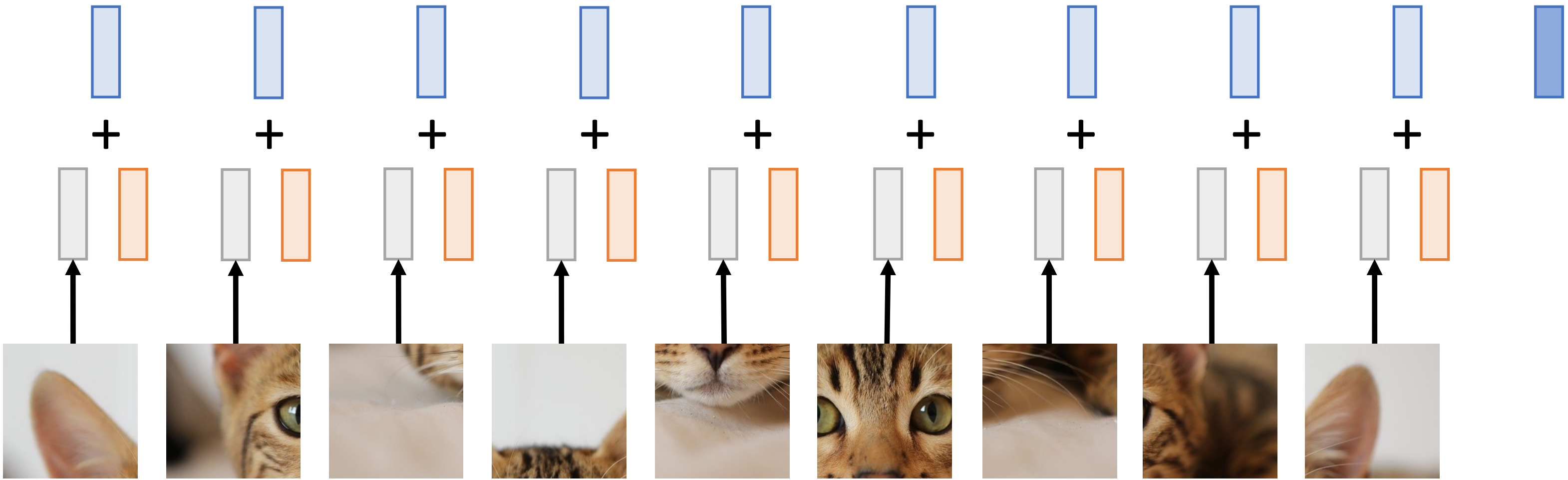
Exact same as
NLP Transformer!



Add positional
embedding: learned D-
dim vector per position

Linear projection to
D-dimensional vector

N input patches, each
of shape 3x16x16



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

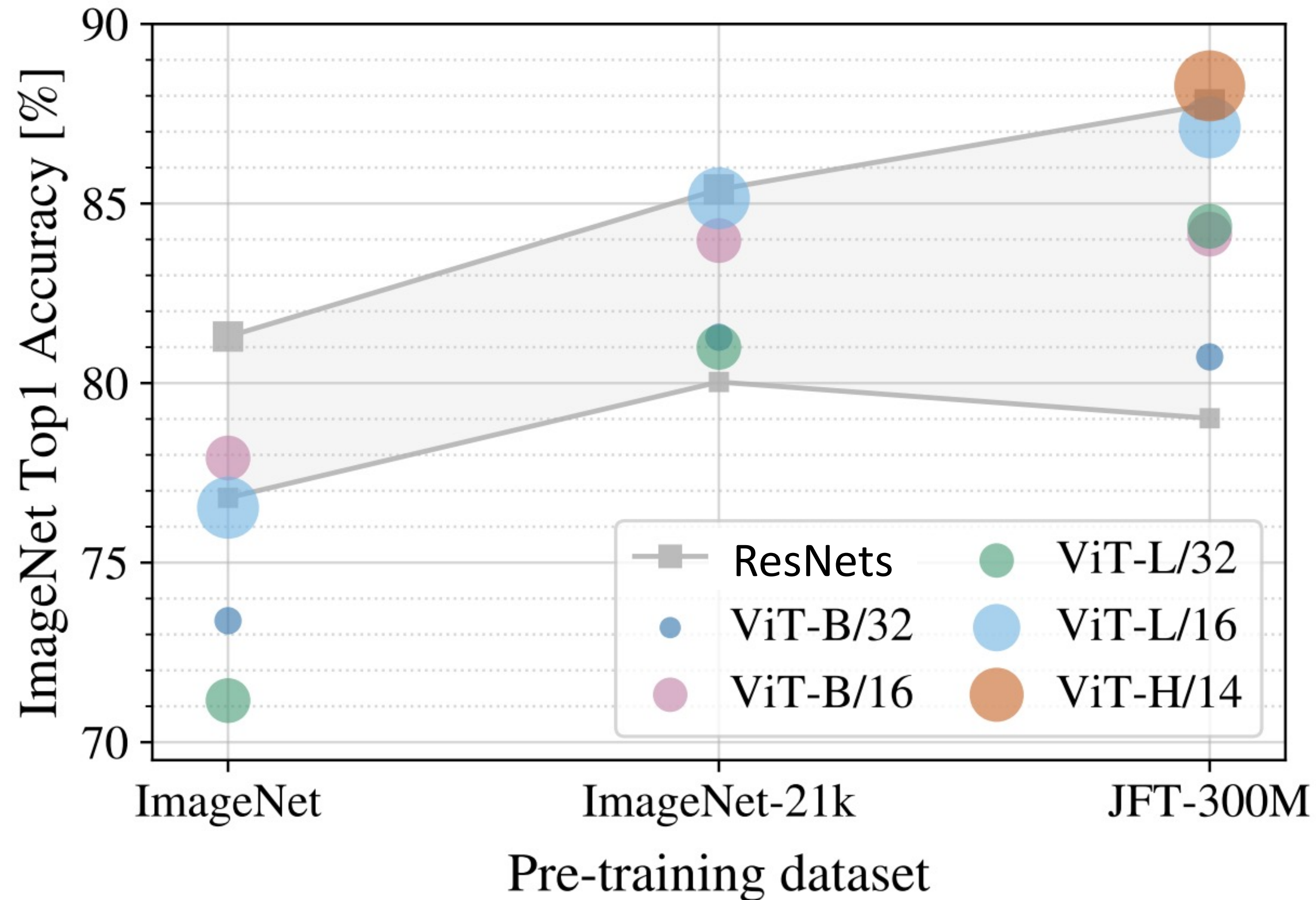
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial
use under a [Pixabay license](#)

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

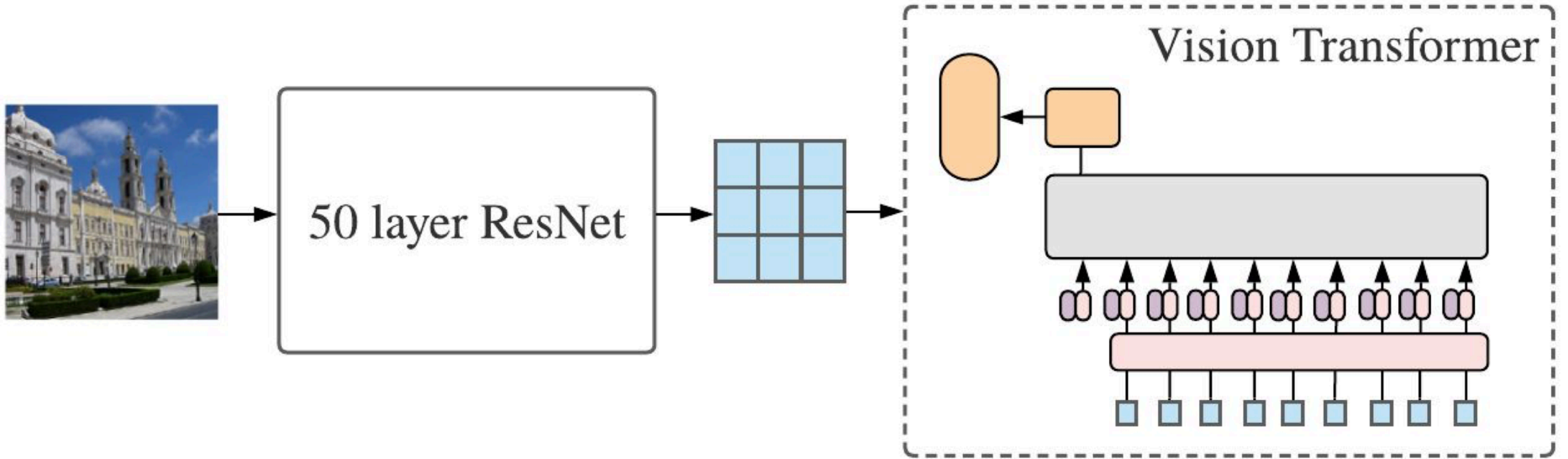


B = Base
L = Large
H = Huge

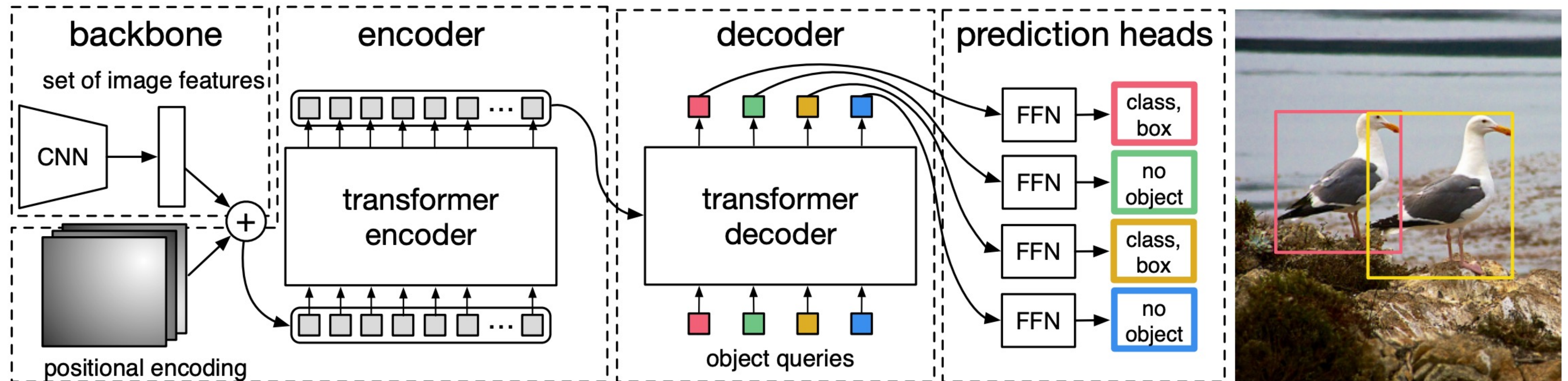
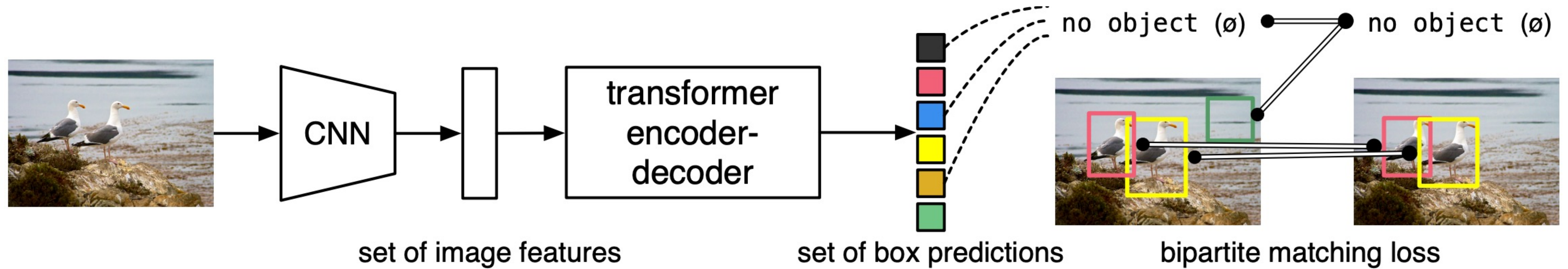
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

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

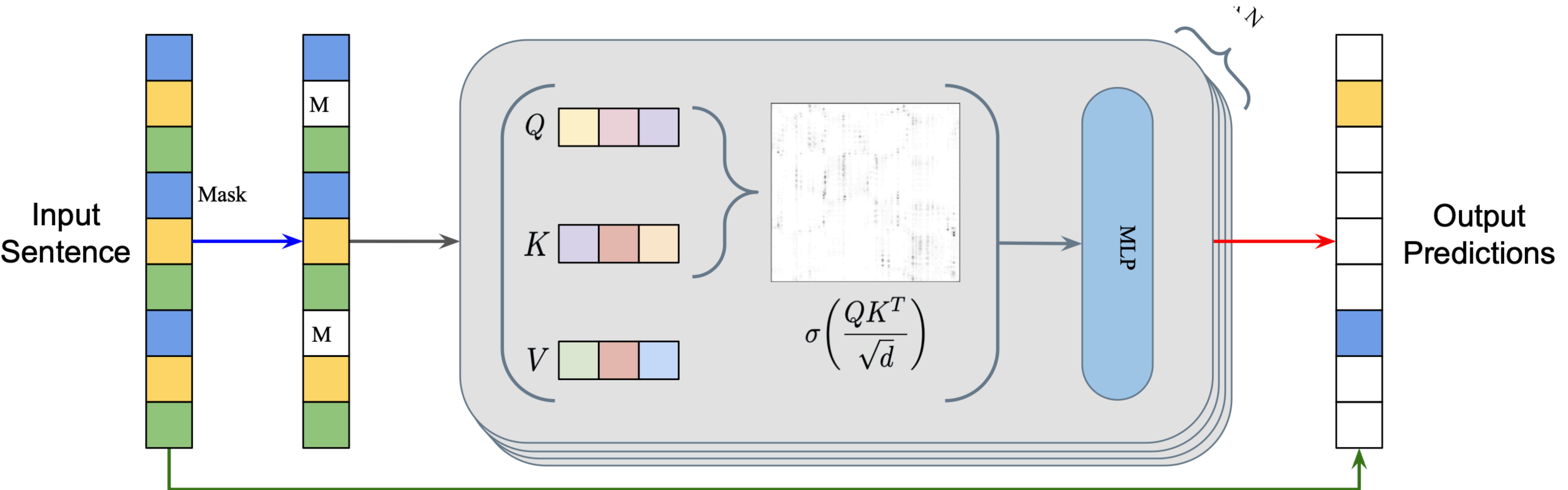
ResNet-ViT Hybrid



Object Detection with Transformers: DETR Model



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



Training: Predict Masked Tokens

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

(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: $\mathbf{X}$ (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: $D_x \times D_Q$)

Computation:

Query vectors: $\mathbf{Q} = \mathbf{XW}_Q$

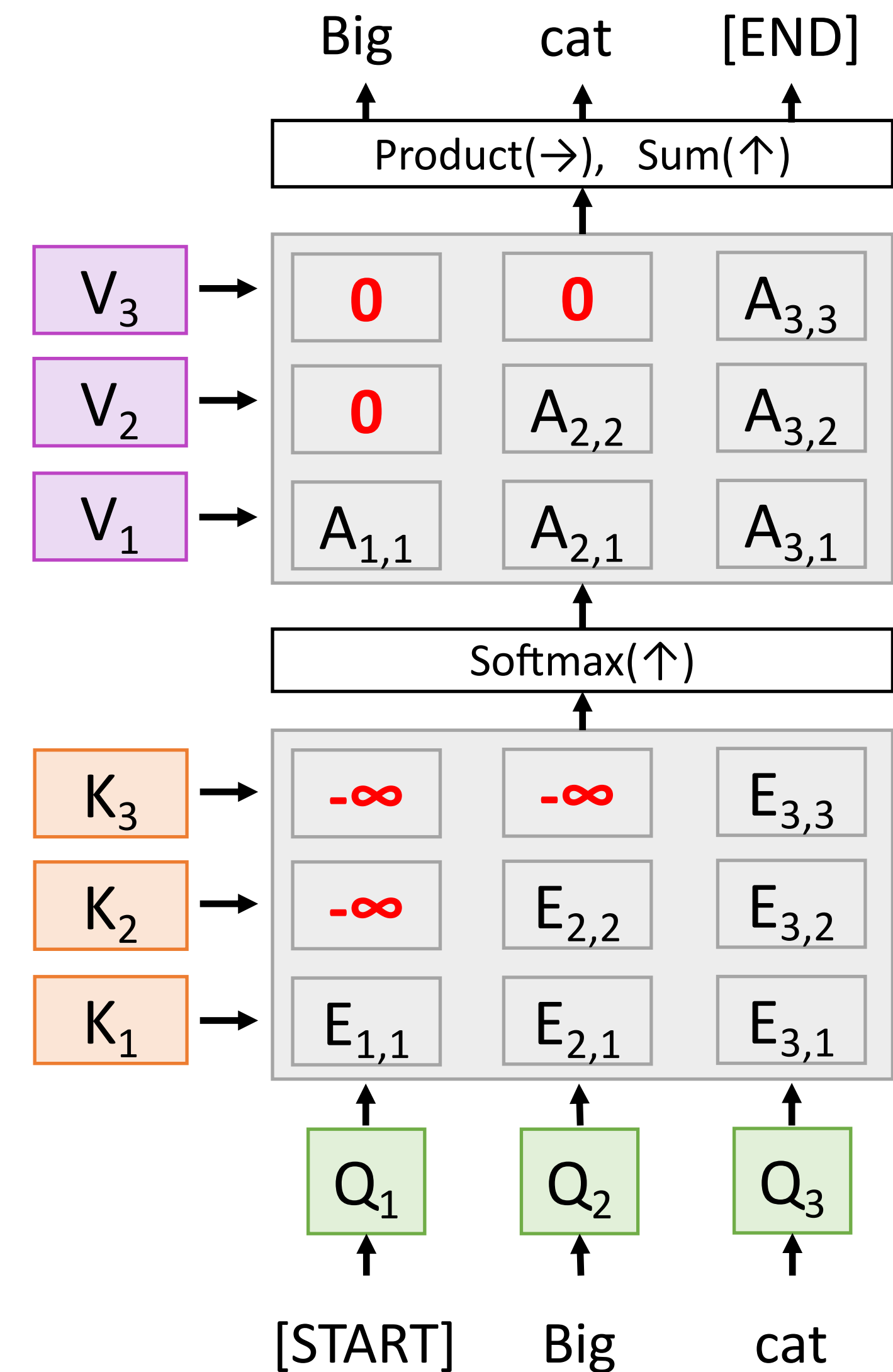
Key vectors: $\mathbf{K} = \mathbf{XW}_k$ (Shape: $N_x \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_v$ (Shape: $N_x \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_x \times N_x$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_x \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Please fill out
Student Evaluations
(on Canvas)