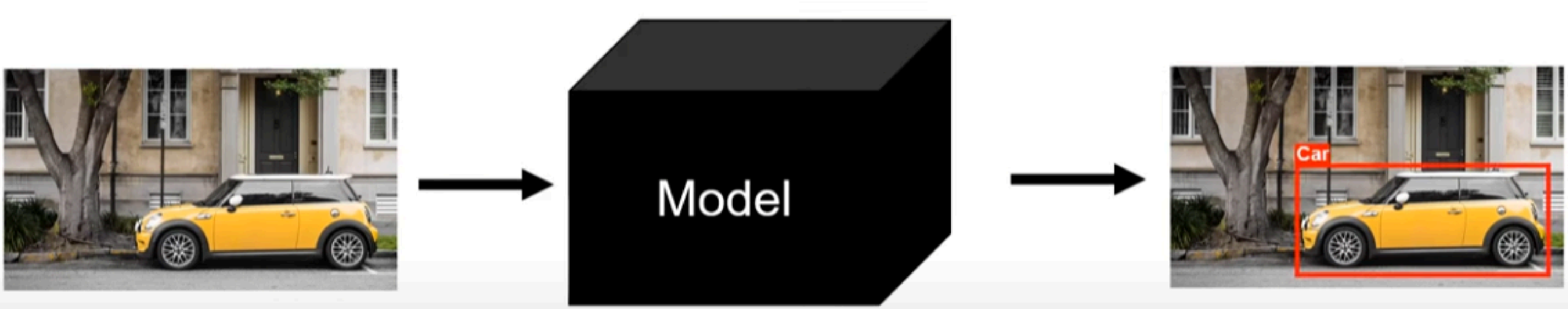
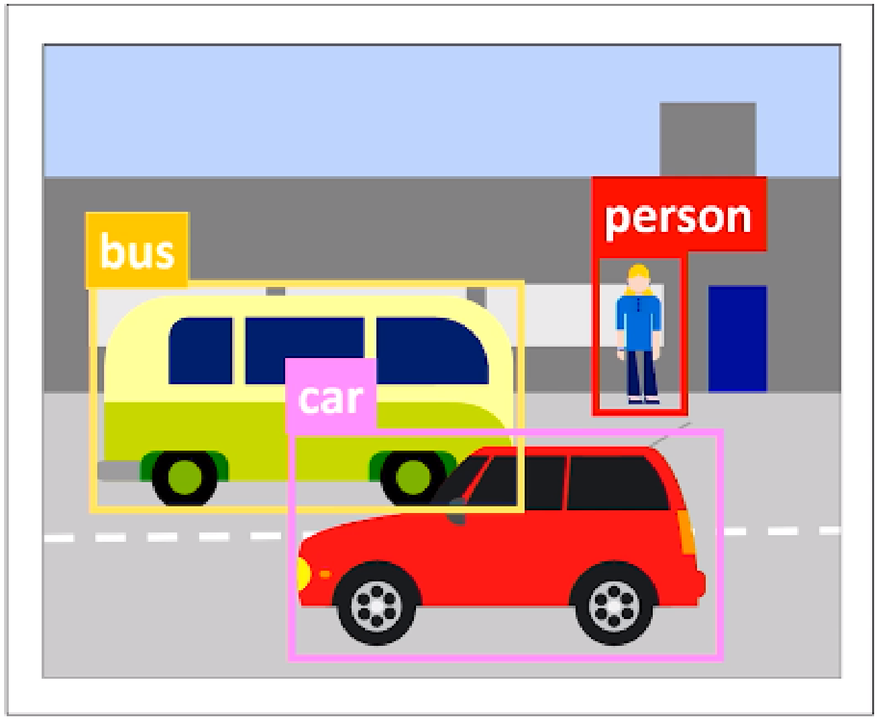
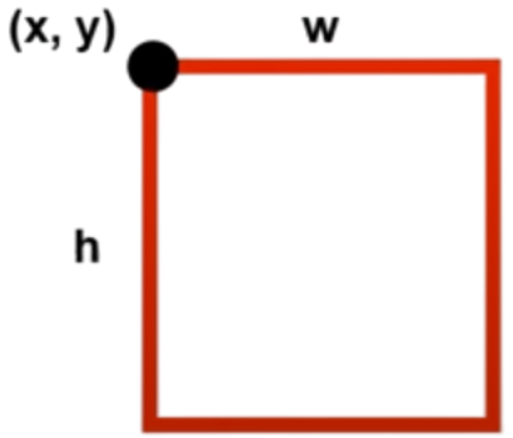
R-CNN & Fast R-CNN
Everything You Need to Know
for
Object Detection

R-CNN:
(Region-based Convolutional Neural Networks)

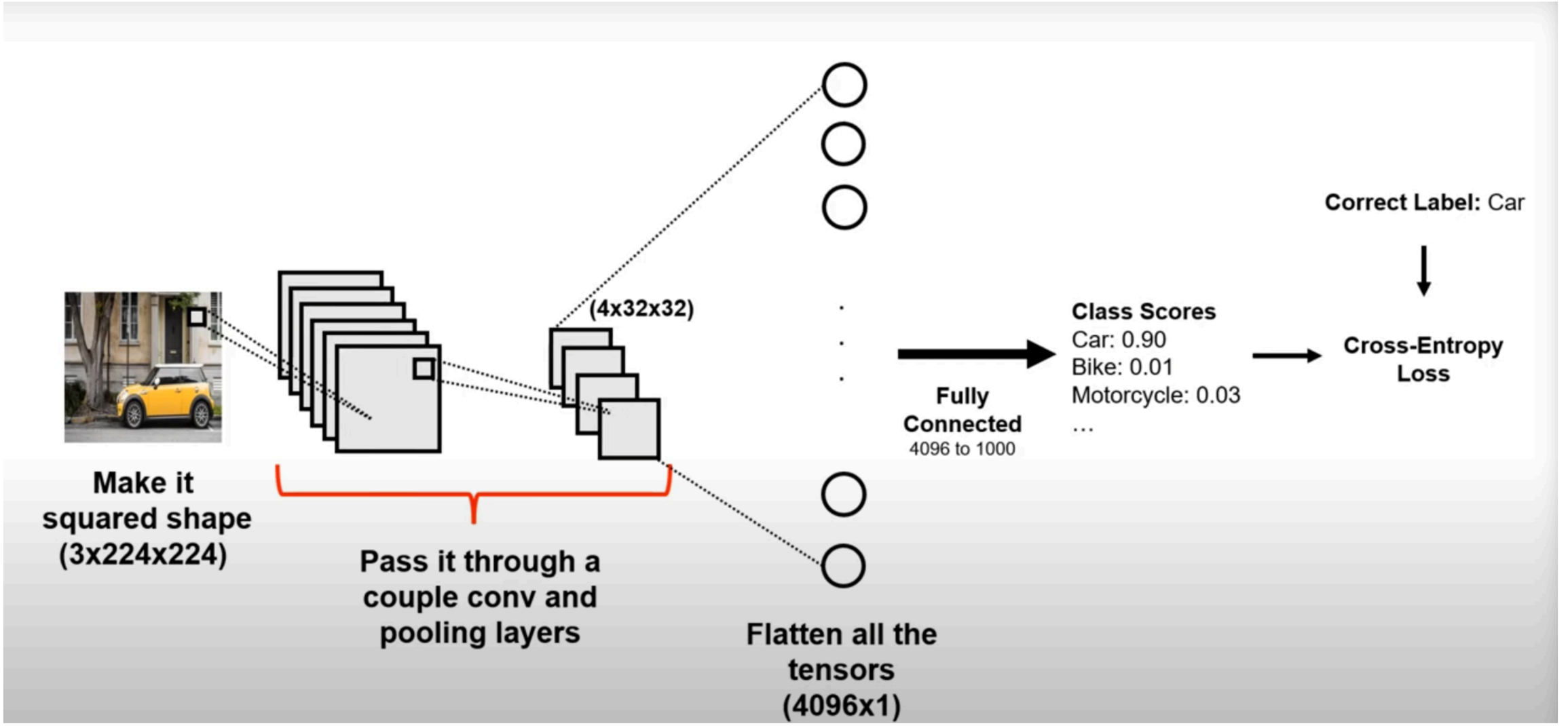
Our Goal in OD

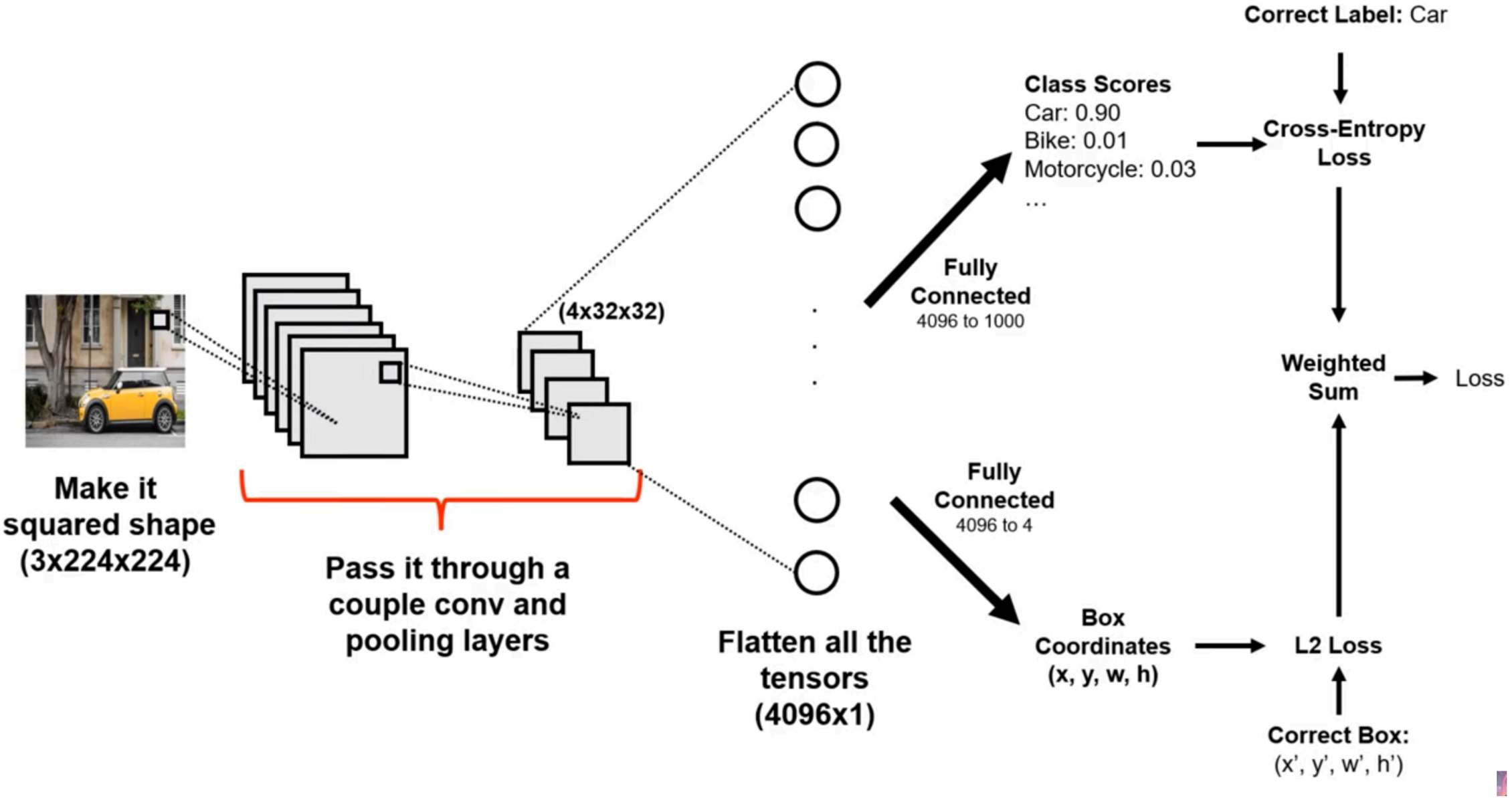




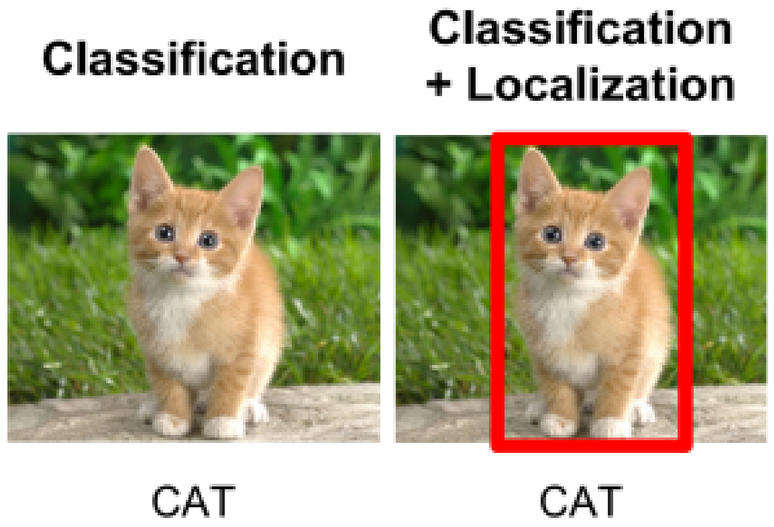


Bounding box


Correct Label: Car
(4x32x32)
Class Scores
Car: 0.90
Bike: 0.01
Motorcycle: 0.03
…
Cross-Entropy Loss
Fully Connected
4096 to 1000
Make it squared shape (3x224x224)
Pass it through a couple conv and pooling layers
Flatten all the tensors (4096x1)


Correct Label: Car
Class Scores
Car: 0.90
Bike: 0.01
Motorcycle: 0.03
…
Cross-Entropy Loss
Fully Connected
4096 to 1000
Weighted Sum
Loss
(4x32x32)
Make it squared shape (3x224x224)
Pass it through a couple conv and pooling layers
Flatten all the tensors (4096x1)
Fully Connected
4096 to 4
Box Coordinates
(x, y, w, h)
L2 Loss
Correct Box:
(x', y', w', h')

Disadvantage : Only one object can be detected :(



Note : This Architecture is used for Object Localization

Old Approach: Sliding Window

Classify this region!

Neural Network classifier

c + 1 outputs

Car
Motorcycle
Bike
…

Background

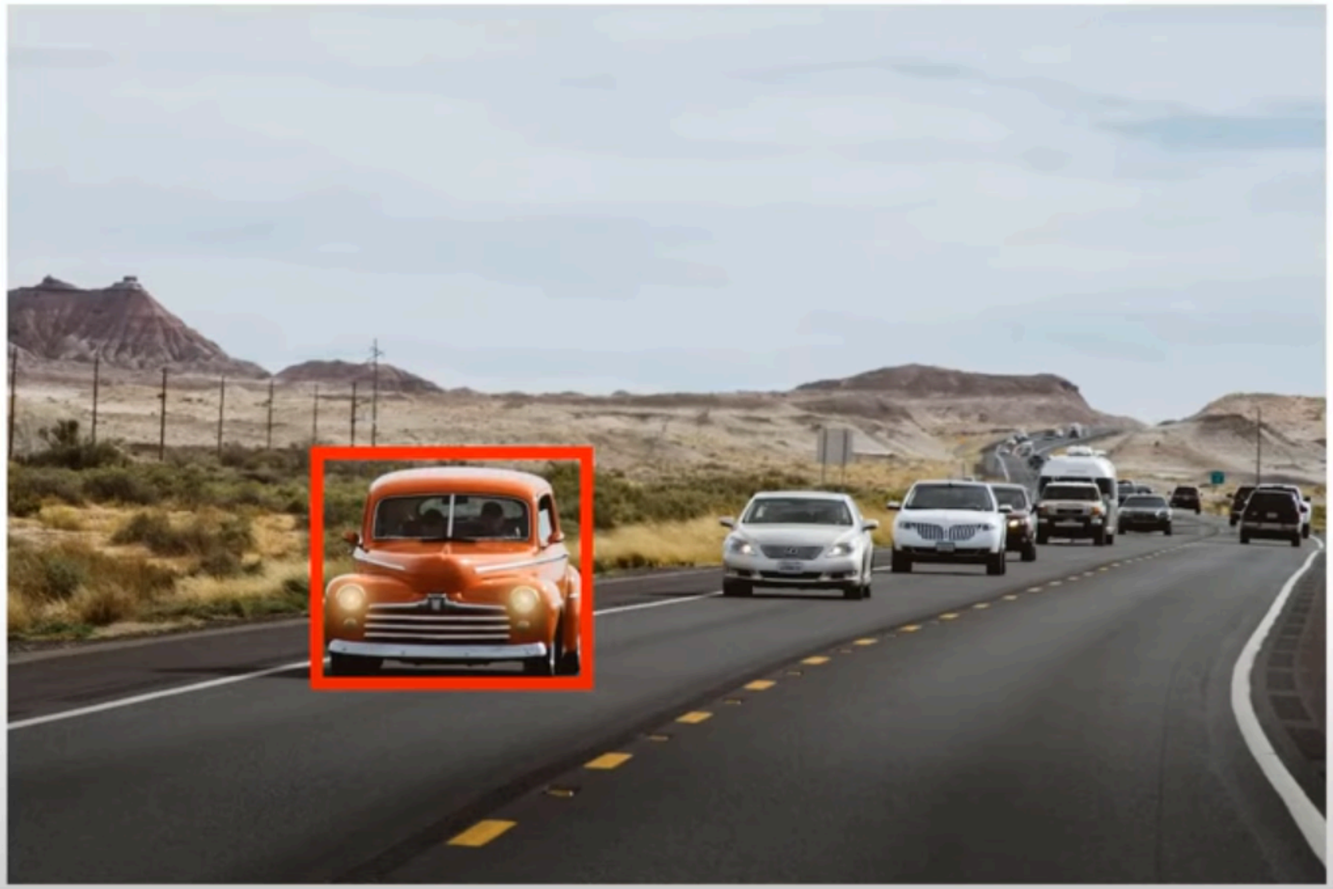
Old Approach: Sliding Window

Classify this region!

Neural Network classifier

Mountain

Old Approach: Sliding Window



Classify this region!

Neural Network classifier

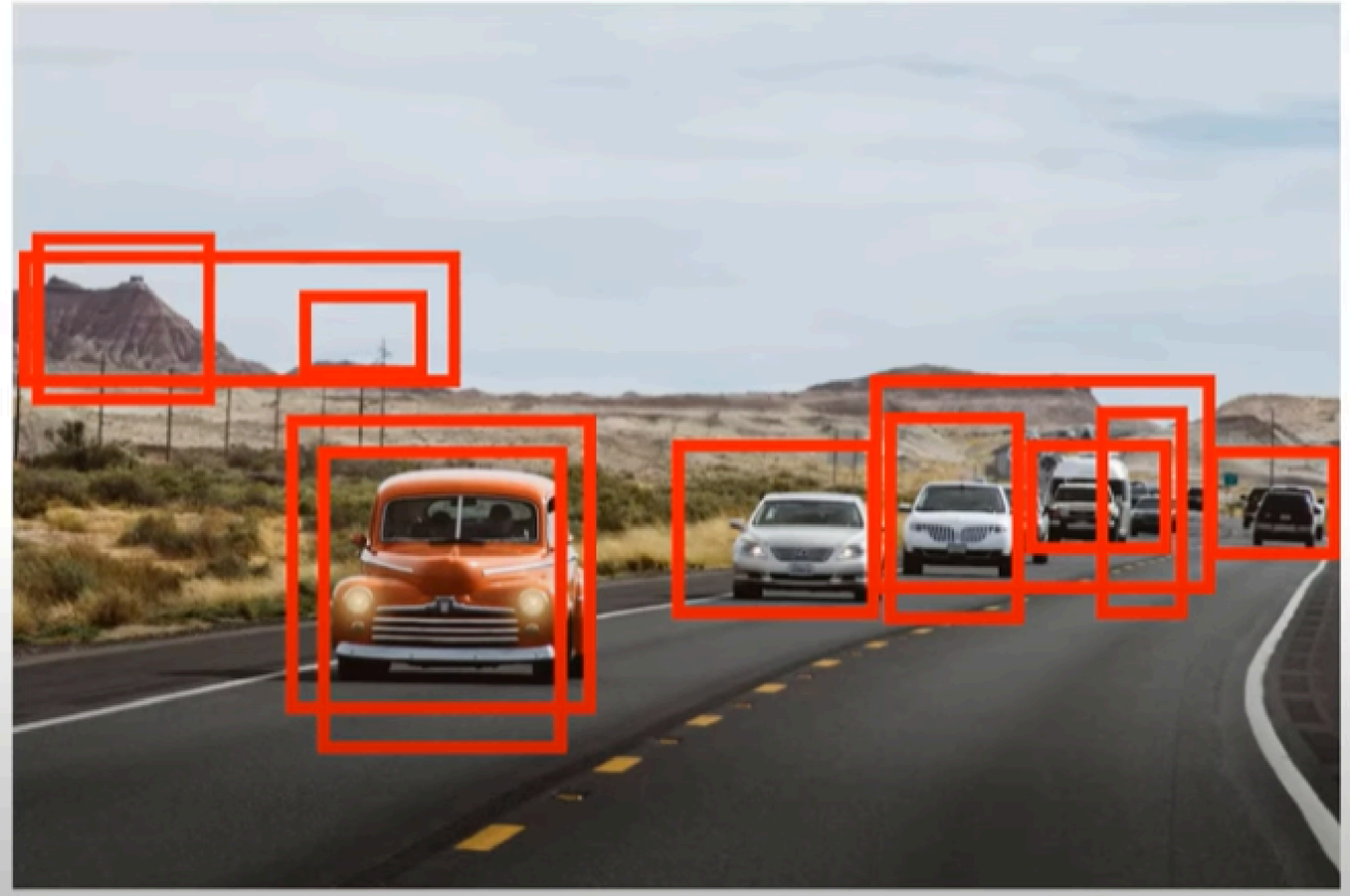
Car

RCNN

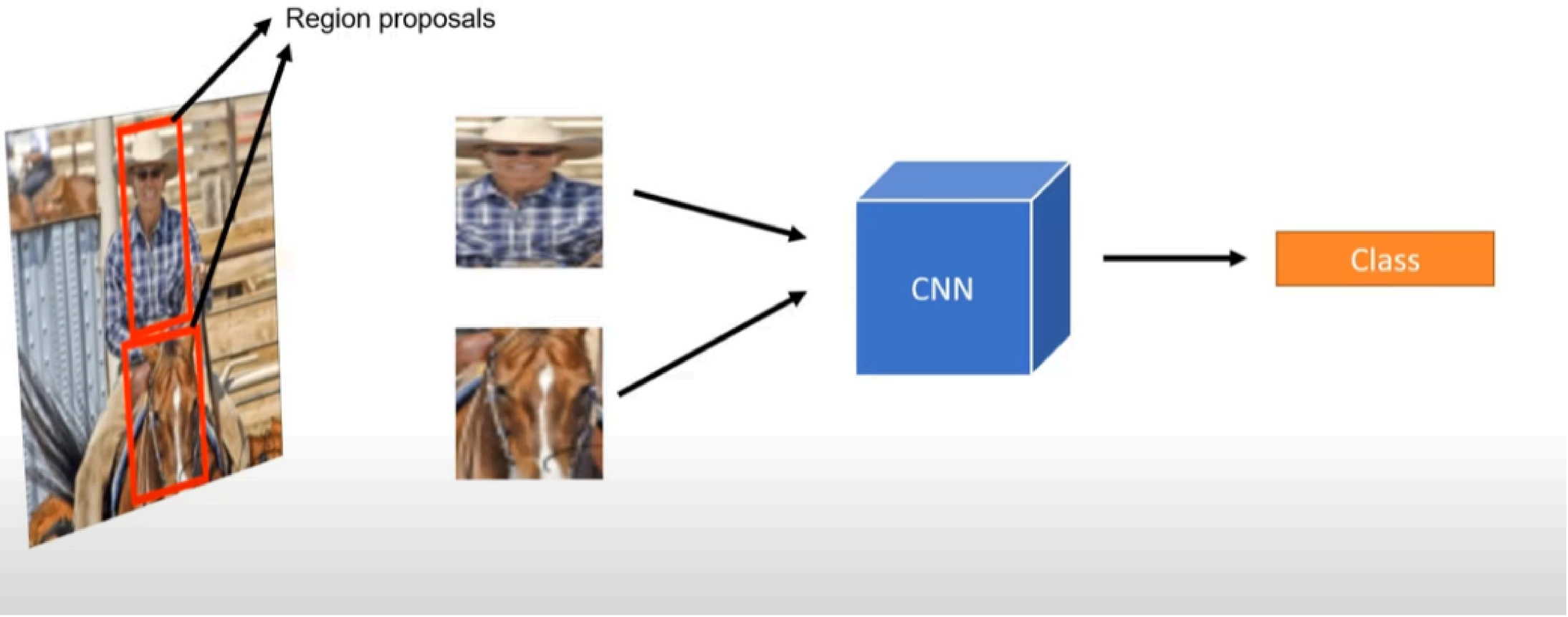
It uses an external algorithm to propose some regions.

Selective Search

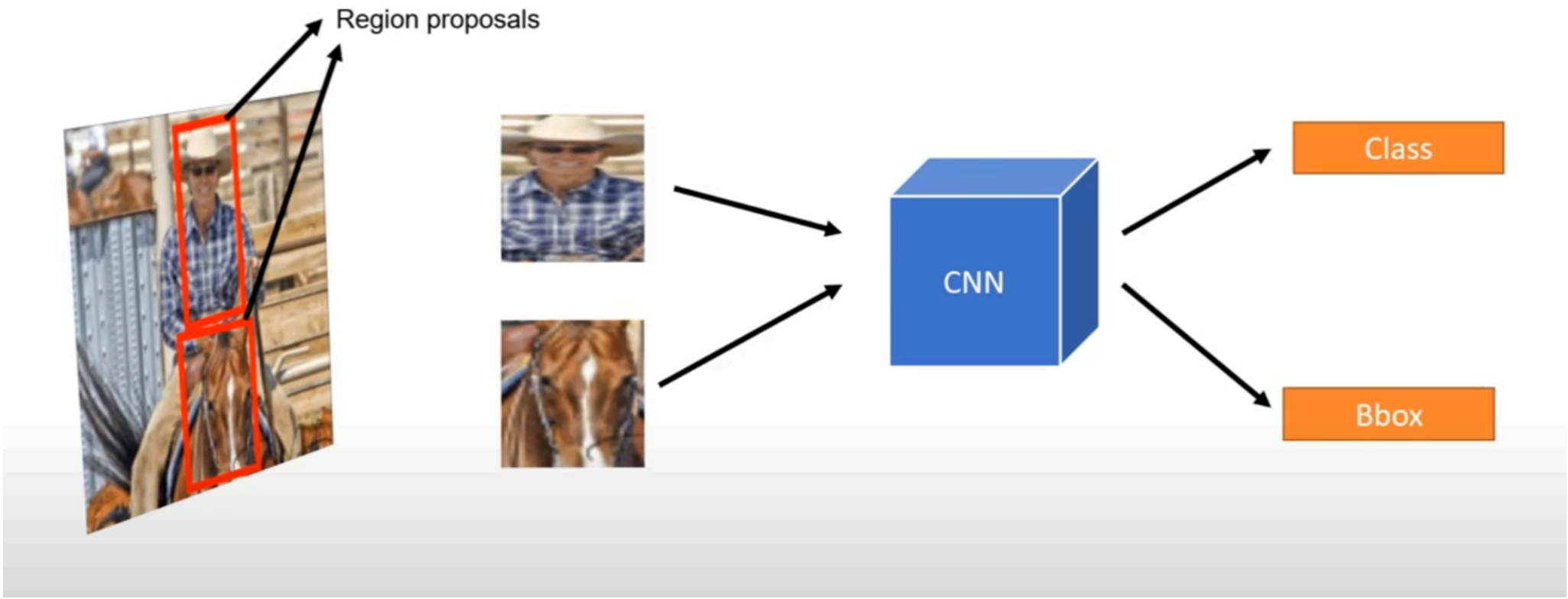
It gives us like 2000 region proposals within one or two seconds



RCNN



RCNN



RCNN

Region proposal: (p_x, p_y, p_h, p_w)

Bbox

Transform: (t_x, t_y, t_h, t_w)

Output: (b_x, b_y, b_h, b_w)

RCNN

Region proposal: (p_x, p_y, p_h, p_w)

Bbox

Transform: (t_x, t_y, t_h, t_w)

Output: (b_x, b_y, b_h, b_w)

Translation:

$b_x = p_x + p_w t_w$ (Horizontal translation)

$b_y = p_y + p_h t_h$ (Vertical translation)

Log-space scale transform:

$b_w = p_w \exp(t_w)$ (Horizontal scale)

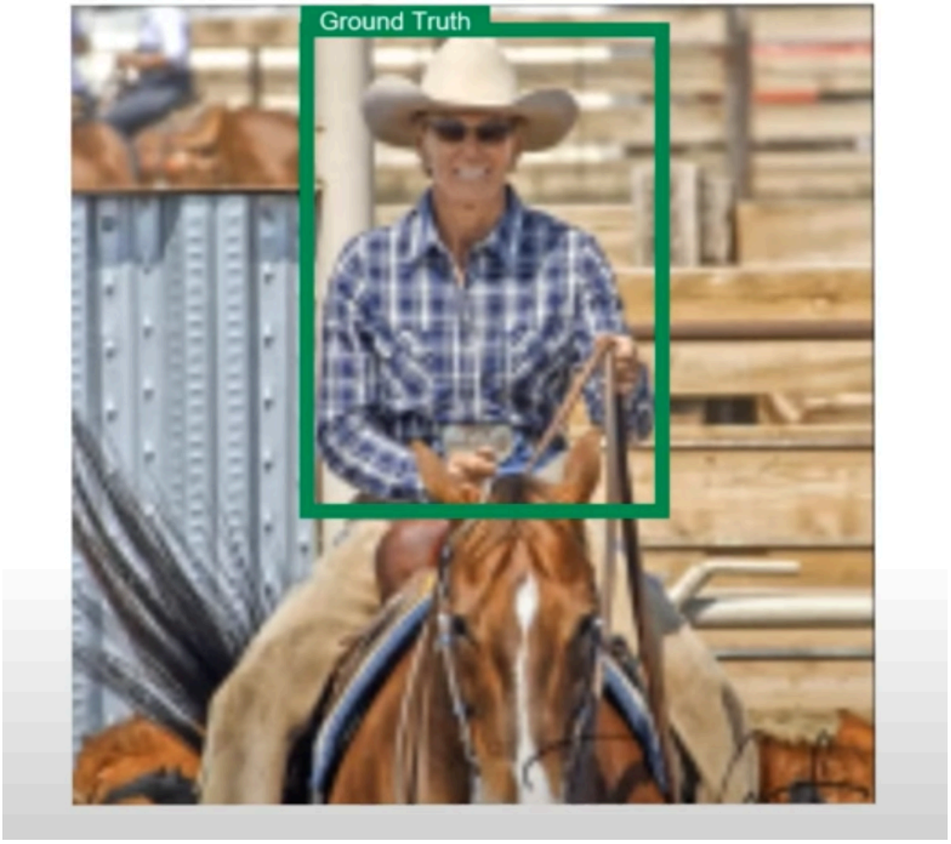
$b_h = p_h \exp(t_h)$ (Vertical scale)

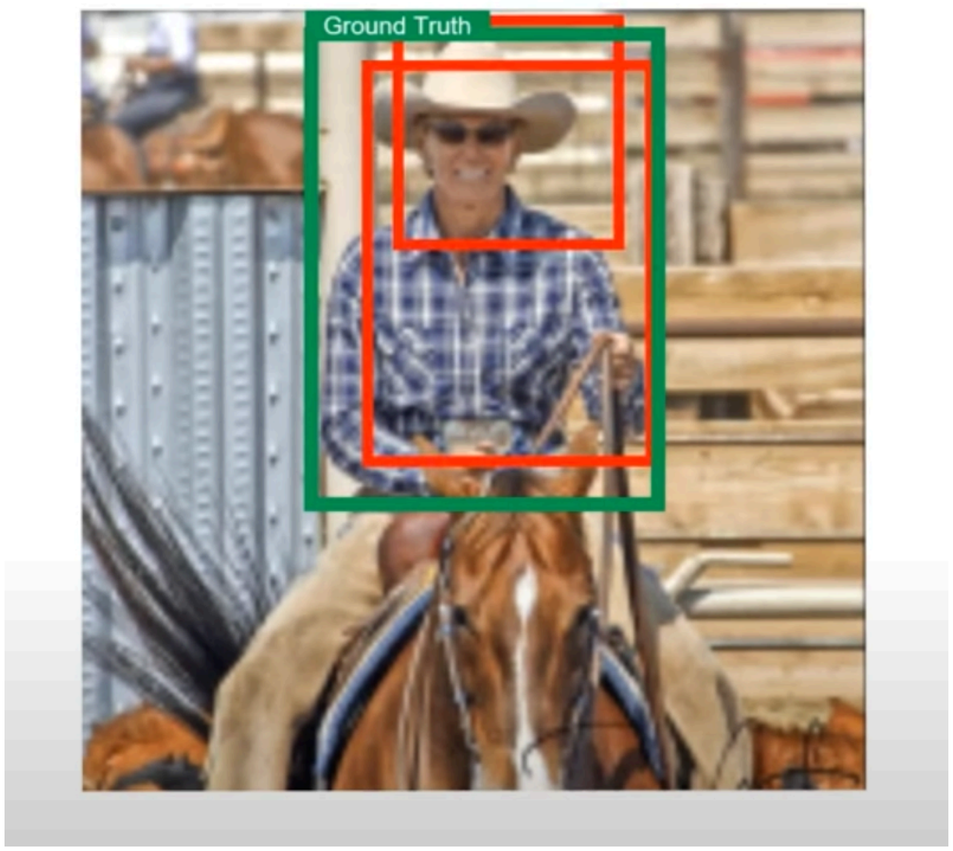
RCNN

What if we output two boxes both pointing to the same object?

Non-Maximum suppression

Non-Maximum suppression

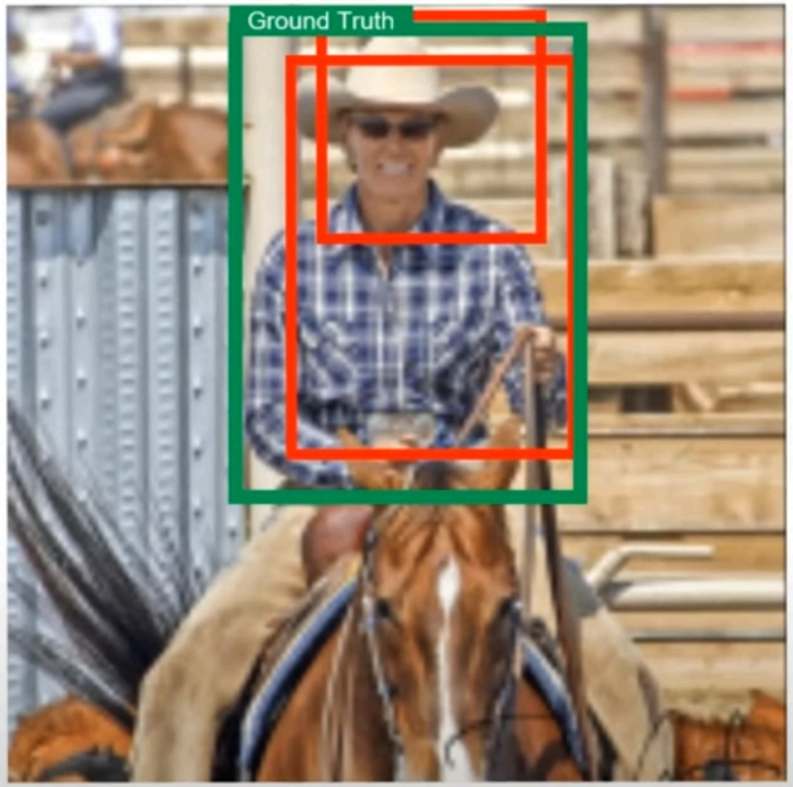




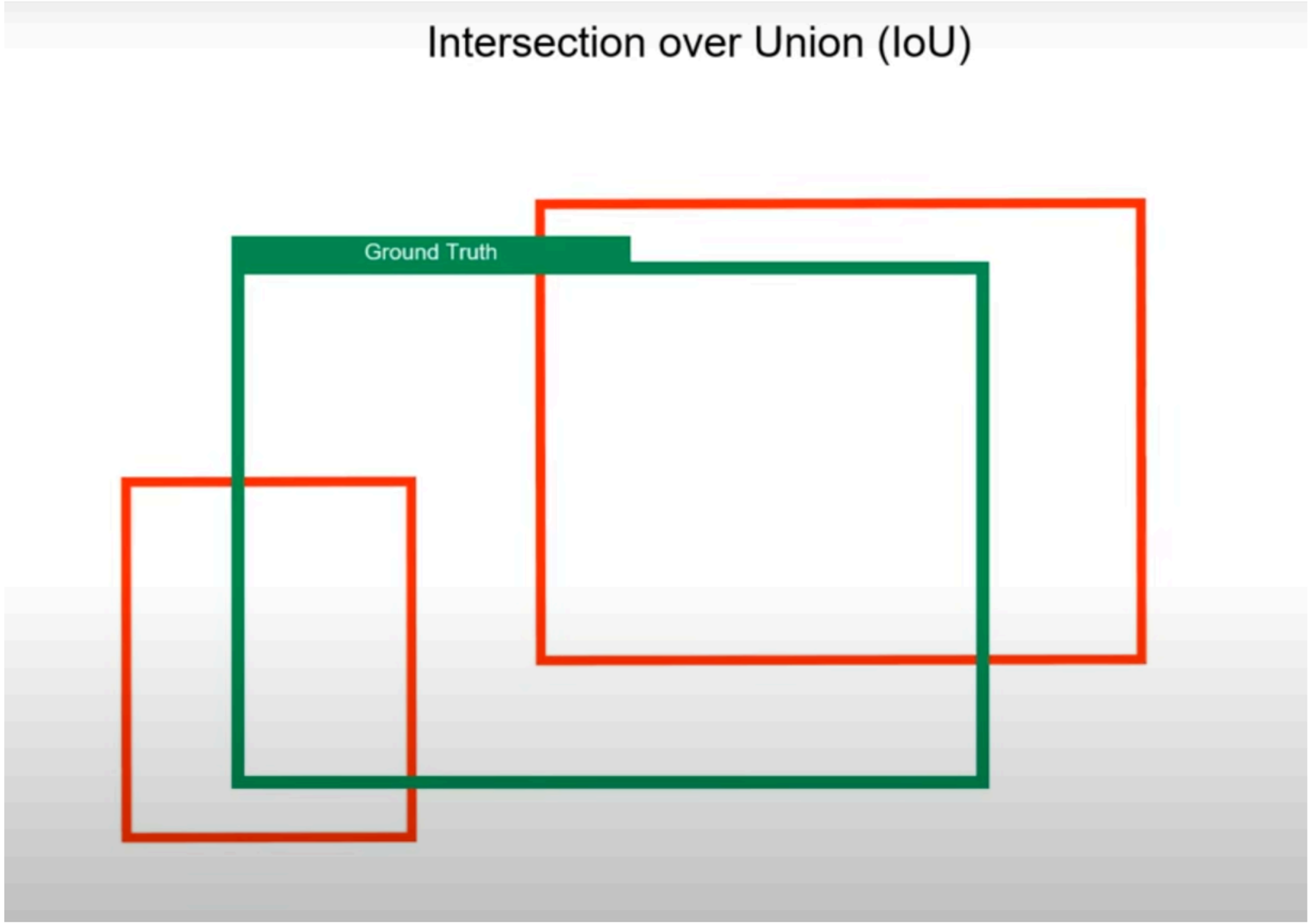
Non-Maximum suppression

Which one to choose?

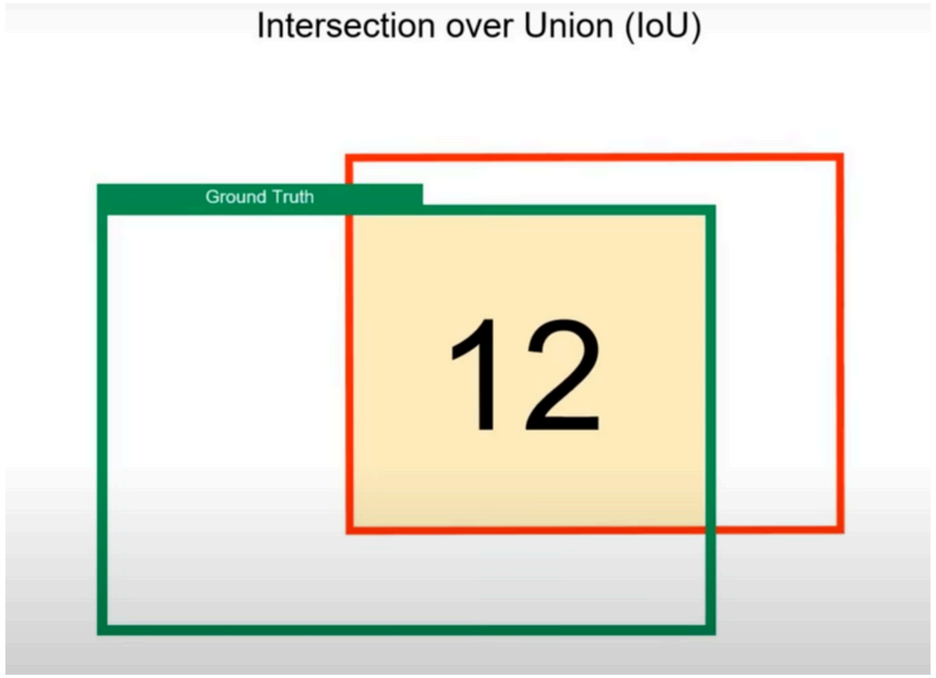
Intuitively: The one that is closer to the ground truth



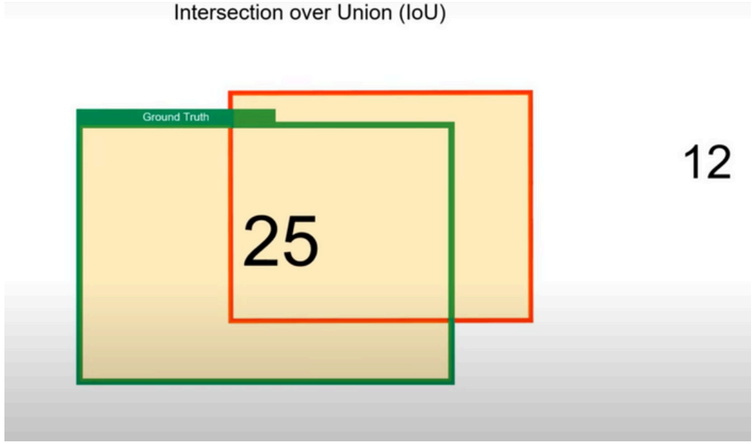
Non-Maximum suppression



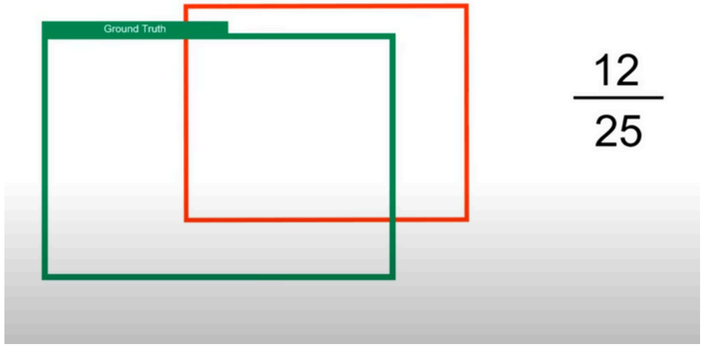
Non-Maximum suppression



Intersection

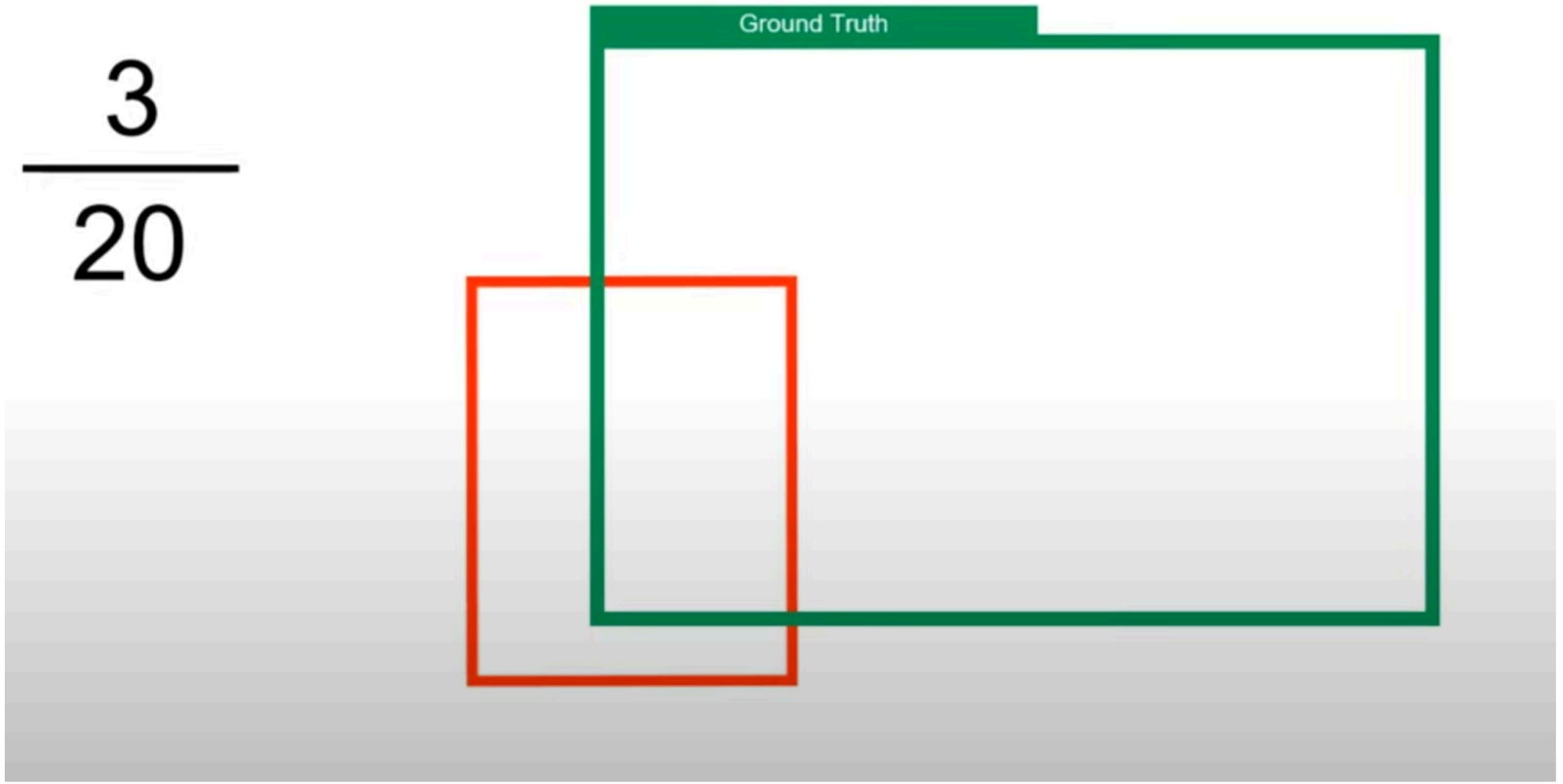


Union

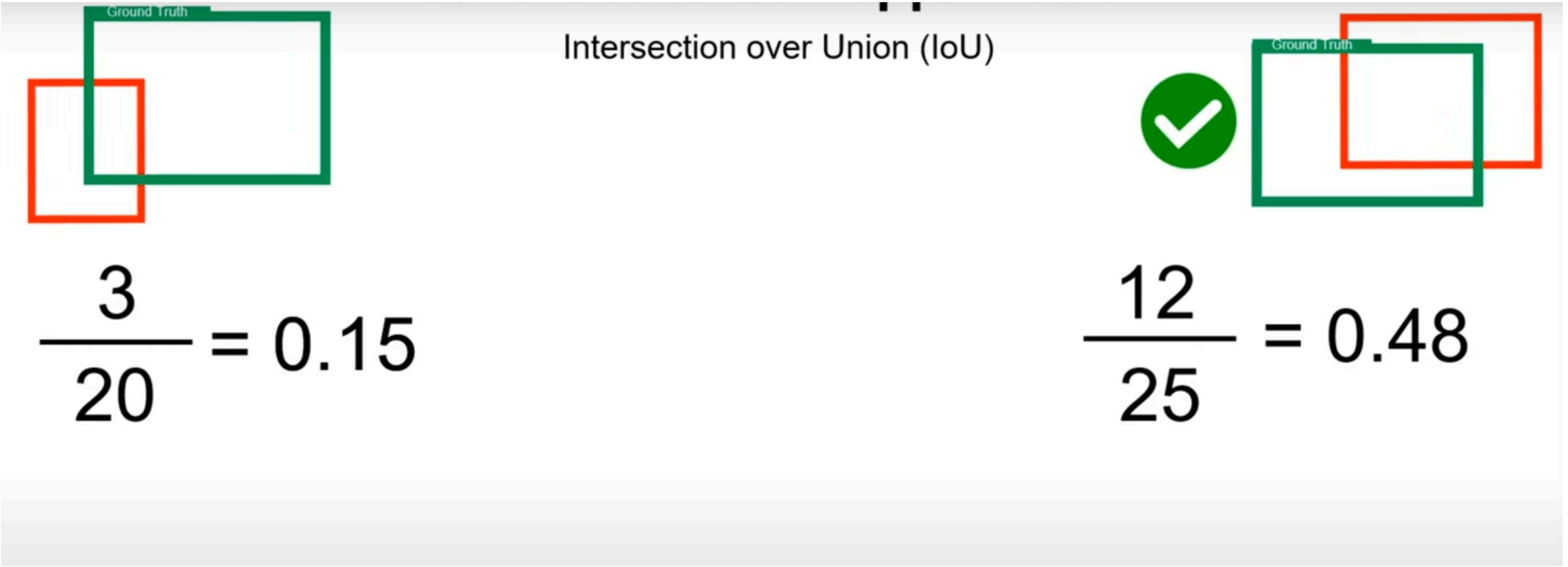


Non-Maximum suppression

$$\frac{3}{20}$$

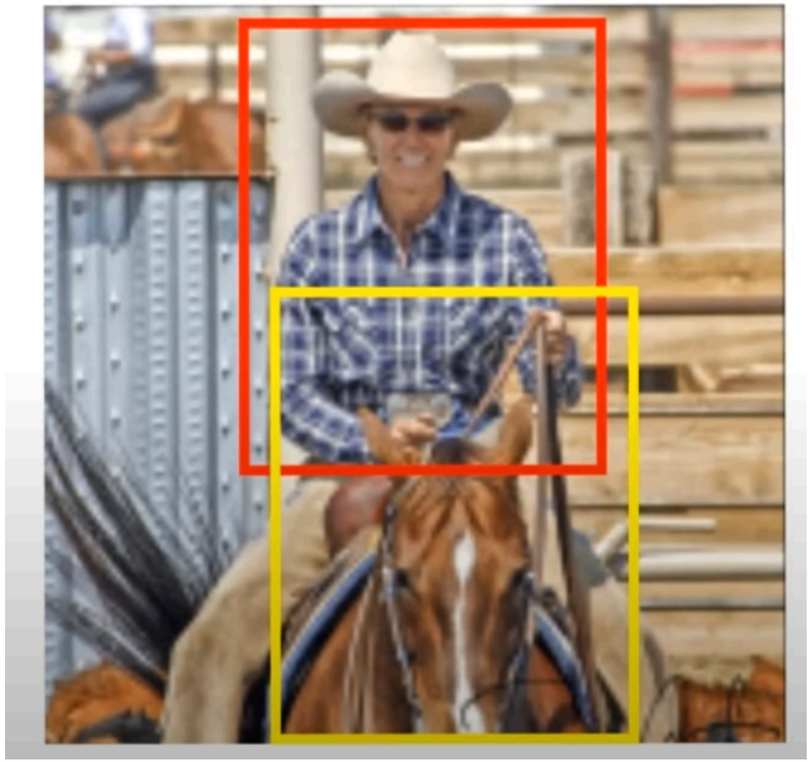


Non-Maximum suppression



Non-Maximum suppression

Note: We use non-max suppression when the object is the same for all of the bounding boxes



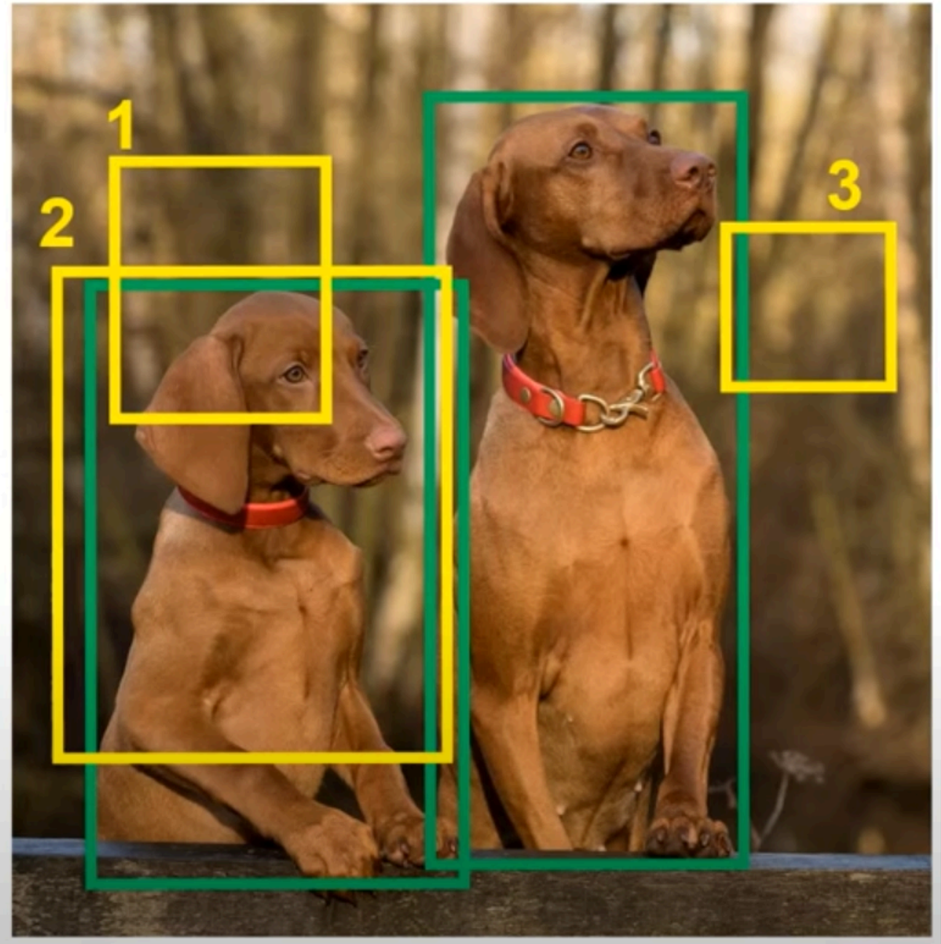
How to Evaluate our Model?

Mean Average Precision(mAP)

Mean Average Precision(mAP)

Which predicted bounding boxes are correct?

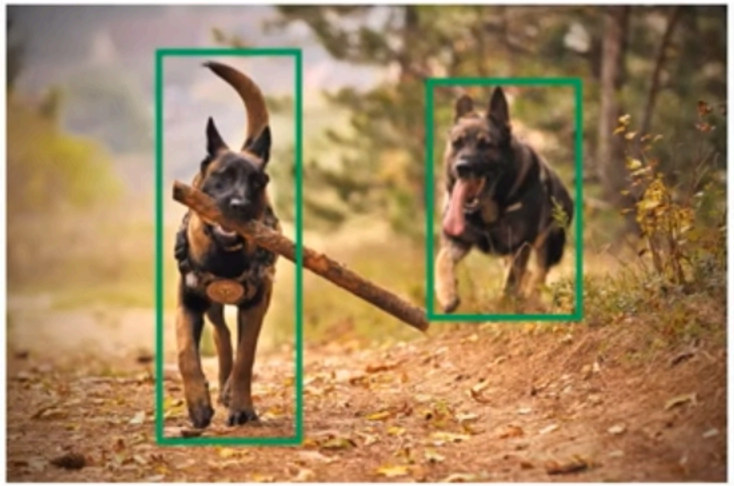
IoU
- >= 0.5 → Correct ✓ — 2
- < 0.5 → Wrong ✗ — 1 3

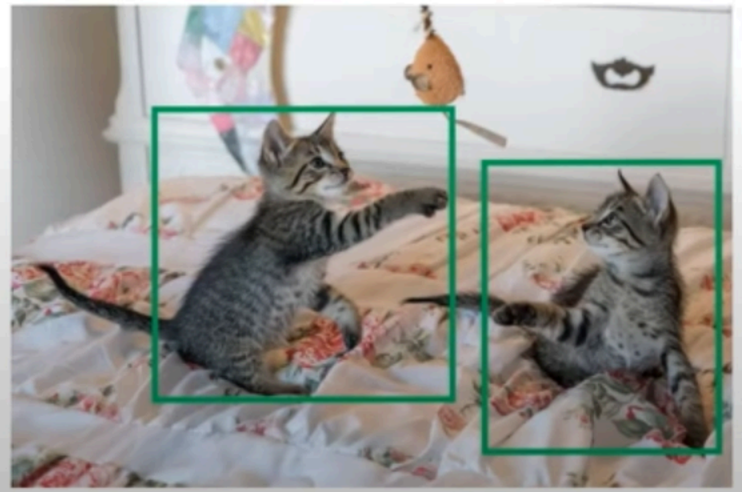


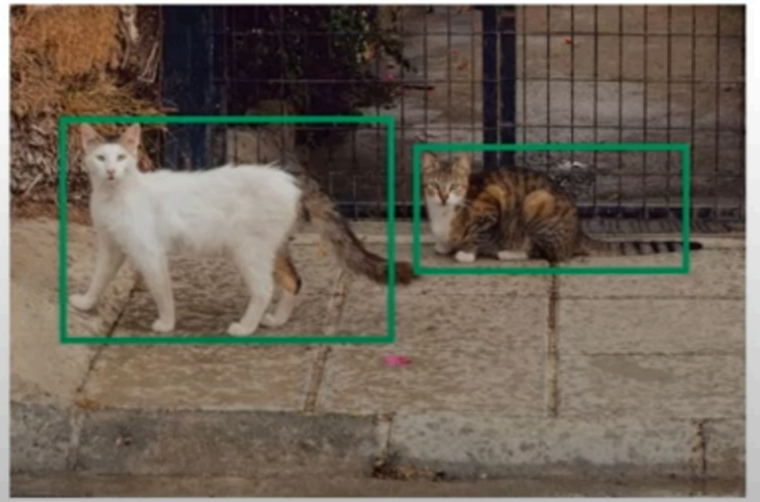
Precision $\frac{1}{3}$

Recall $\frac{1}{2}$

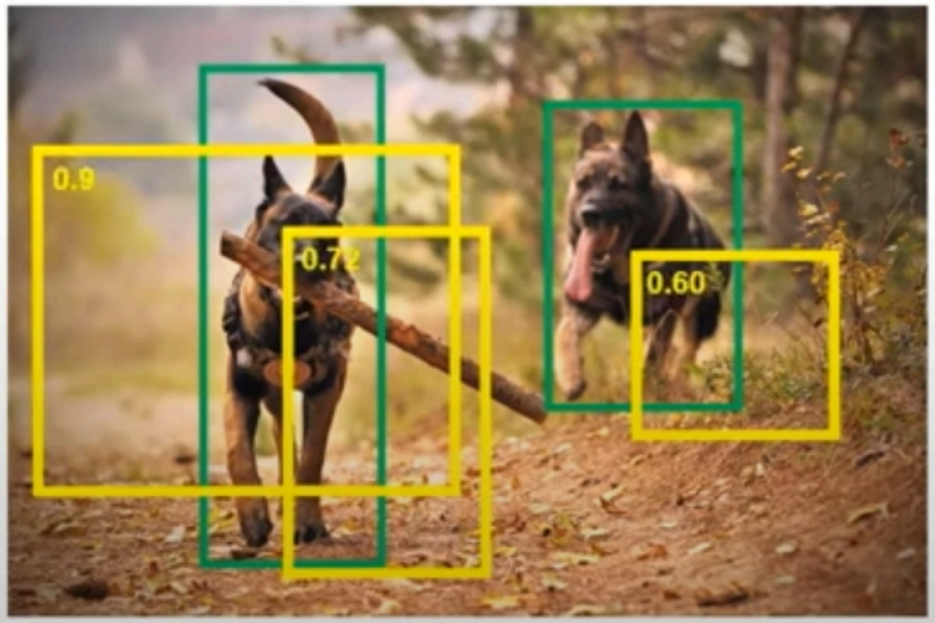
Mean Average Precision(mAP)

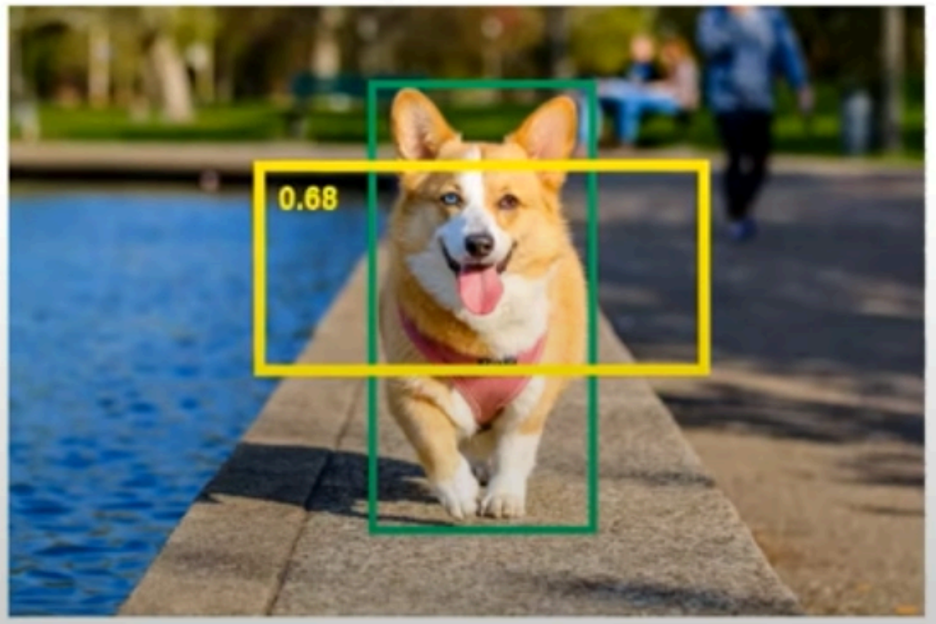




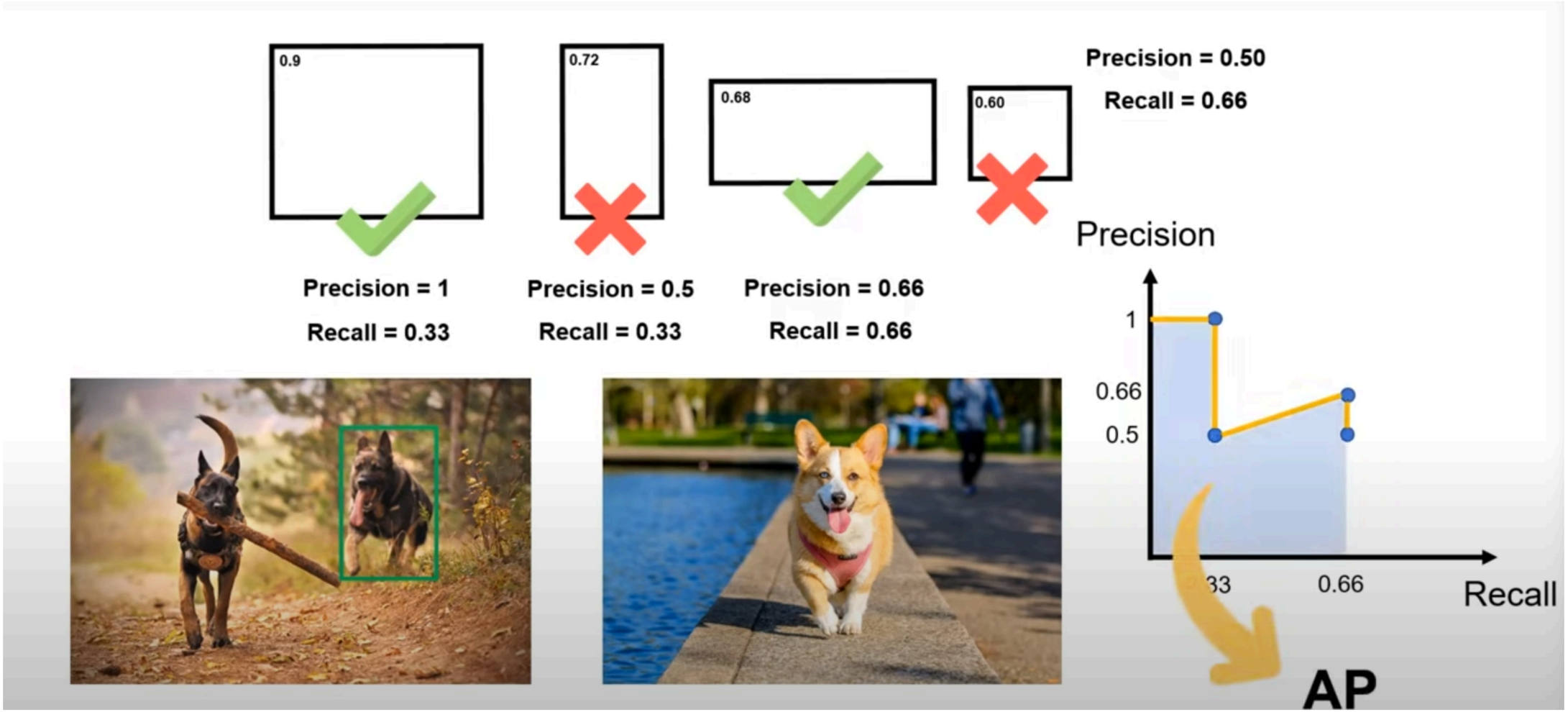


Mean Average Precision(mAP)

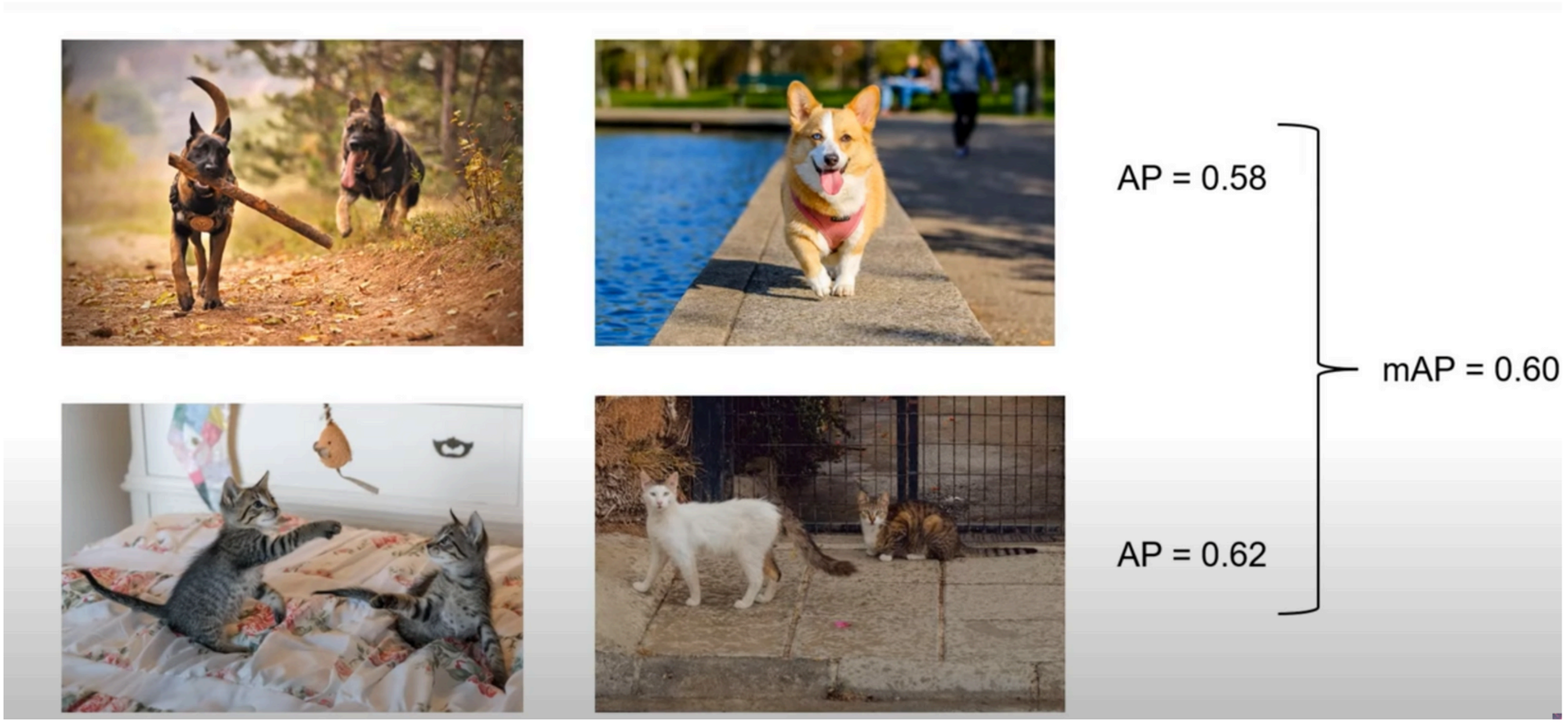




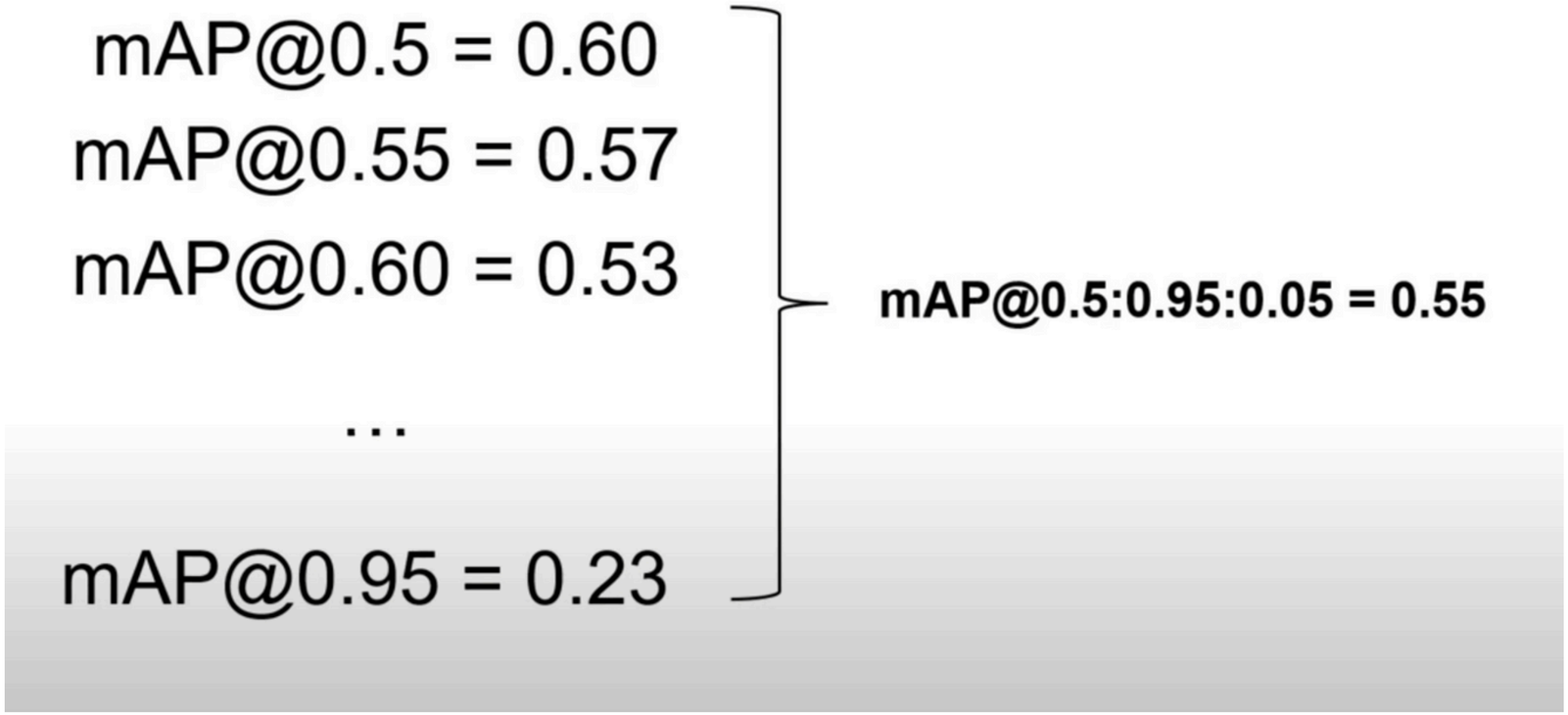
Mean Average Precision(mAP)



Mean Average Precision(mAP)



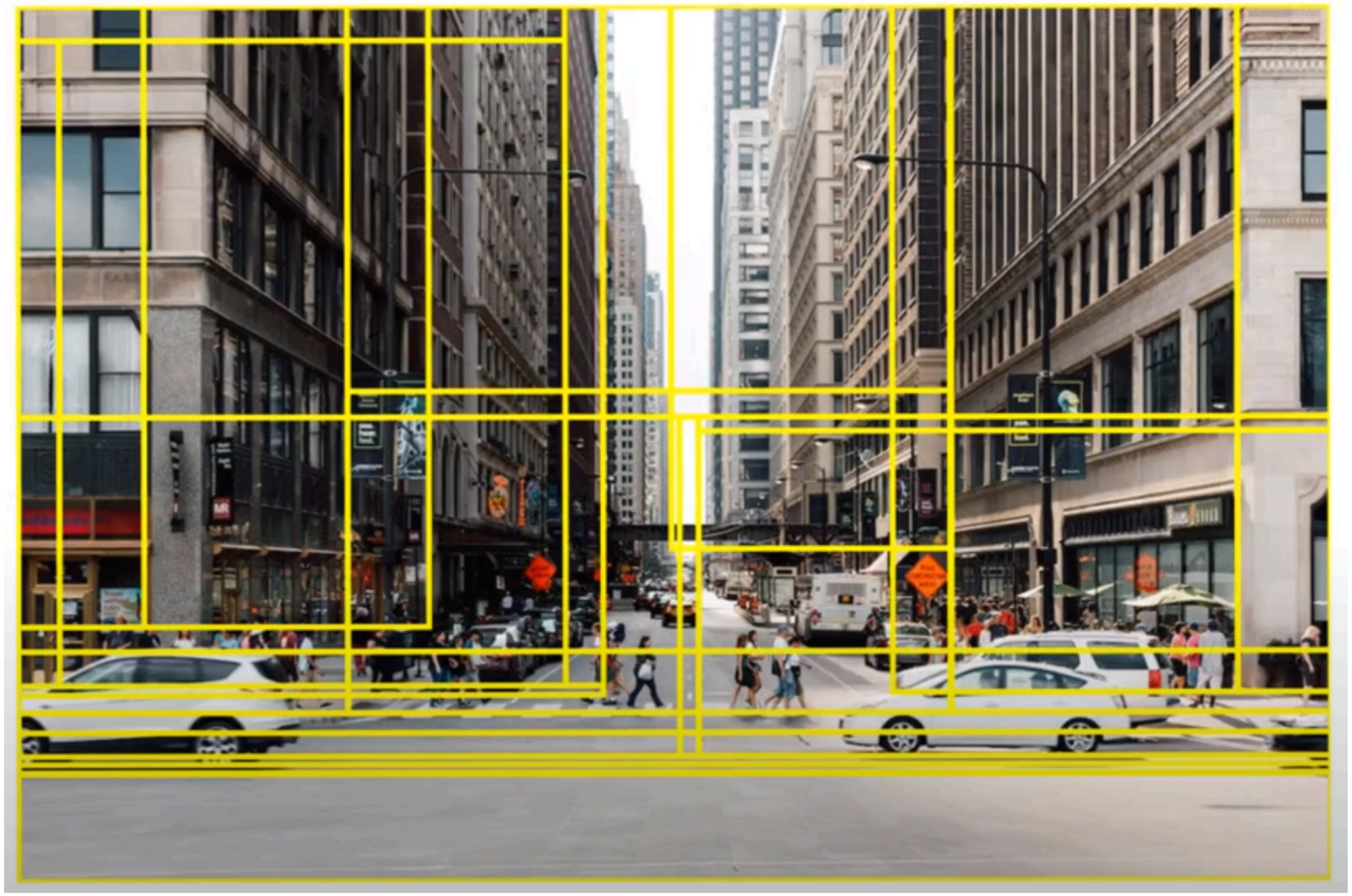
Mean Average Precision(mAP)



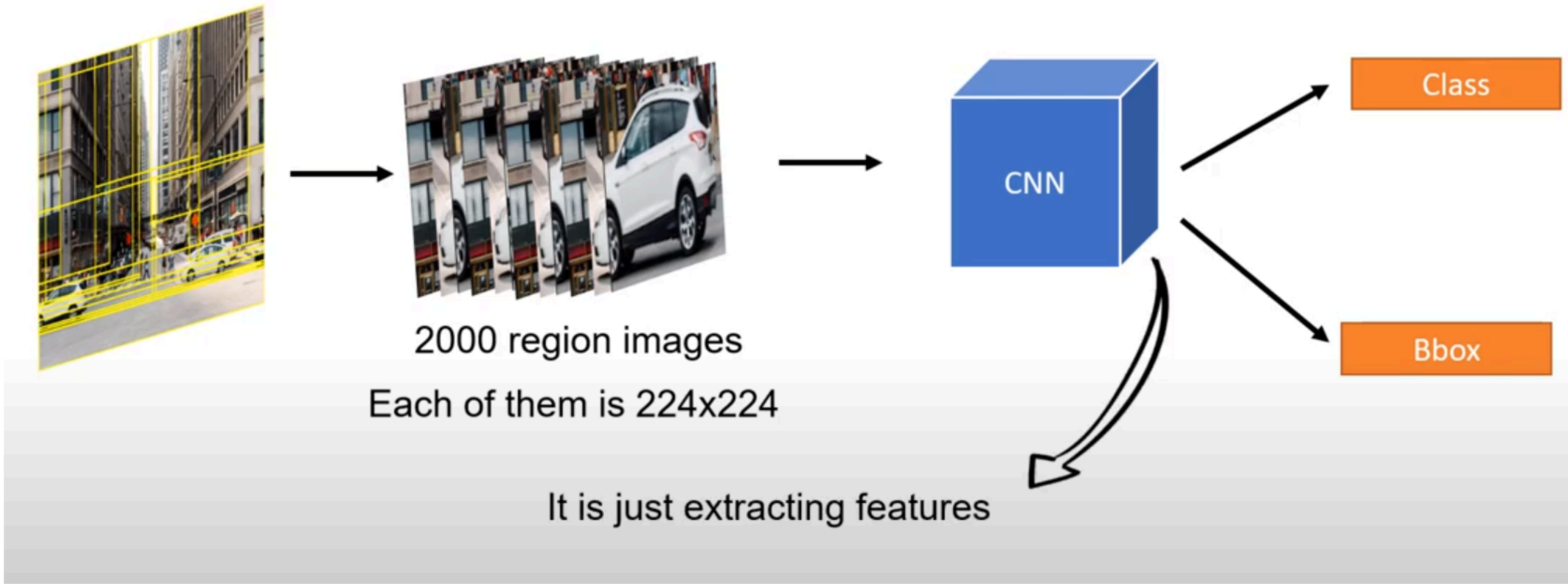
FAST R-CNN

Recap of RCNN

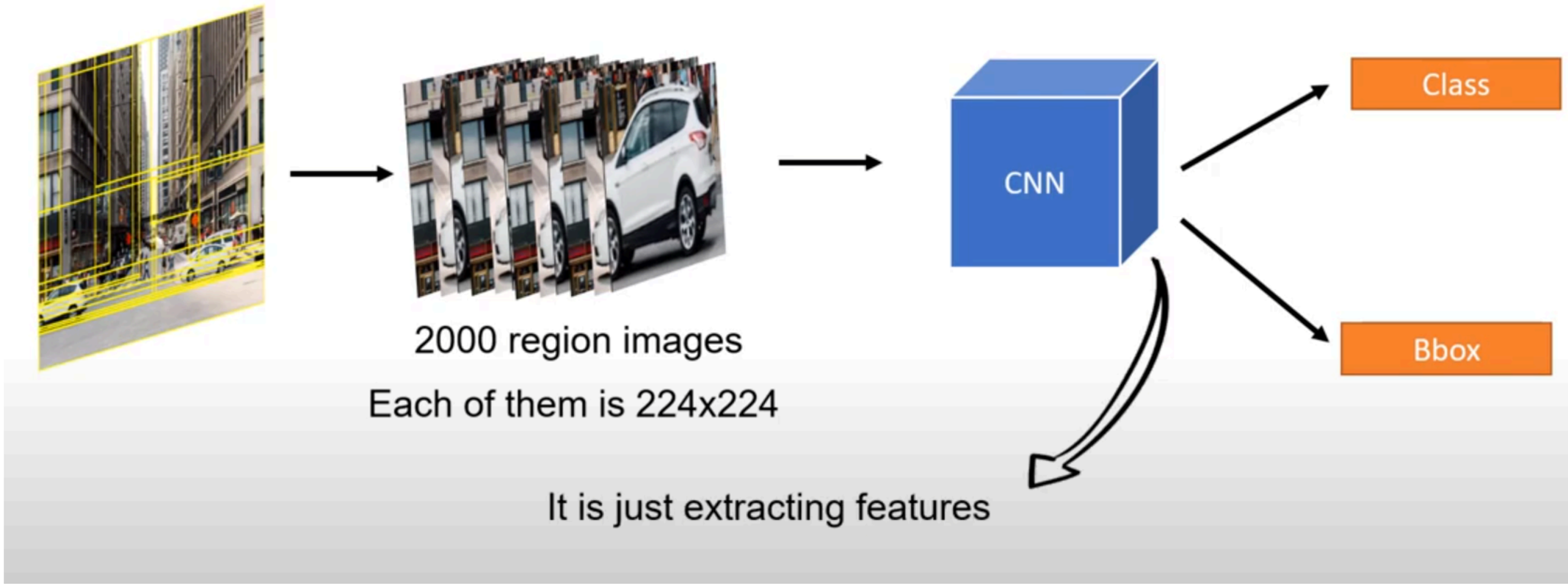
Recap of RCNN



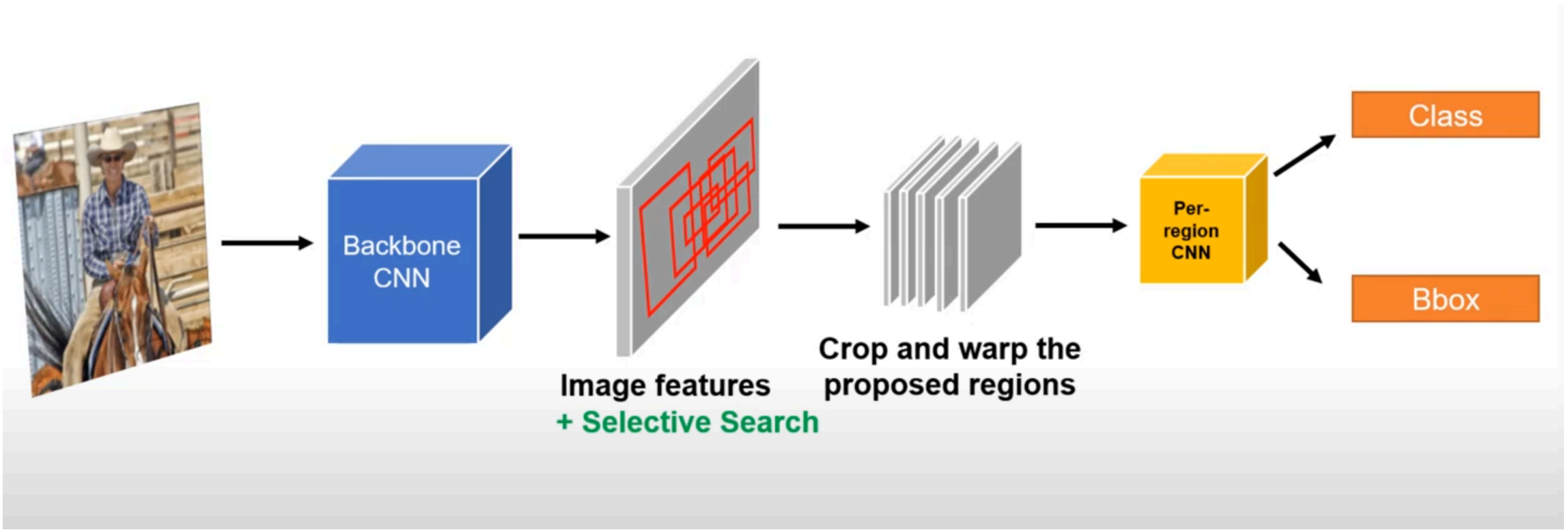
Recap of RCNN



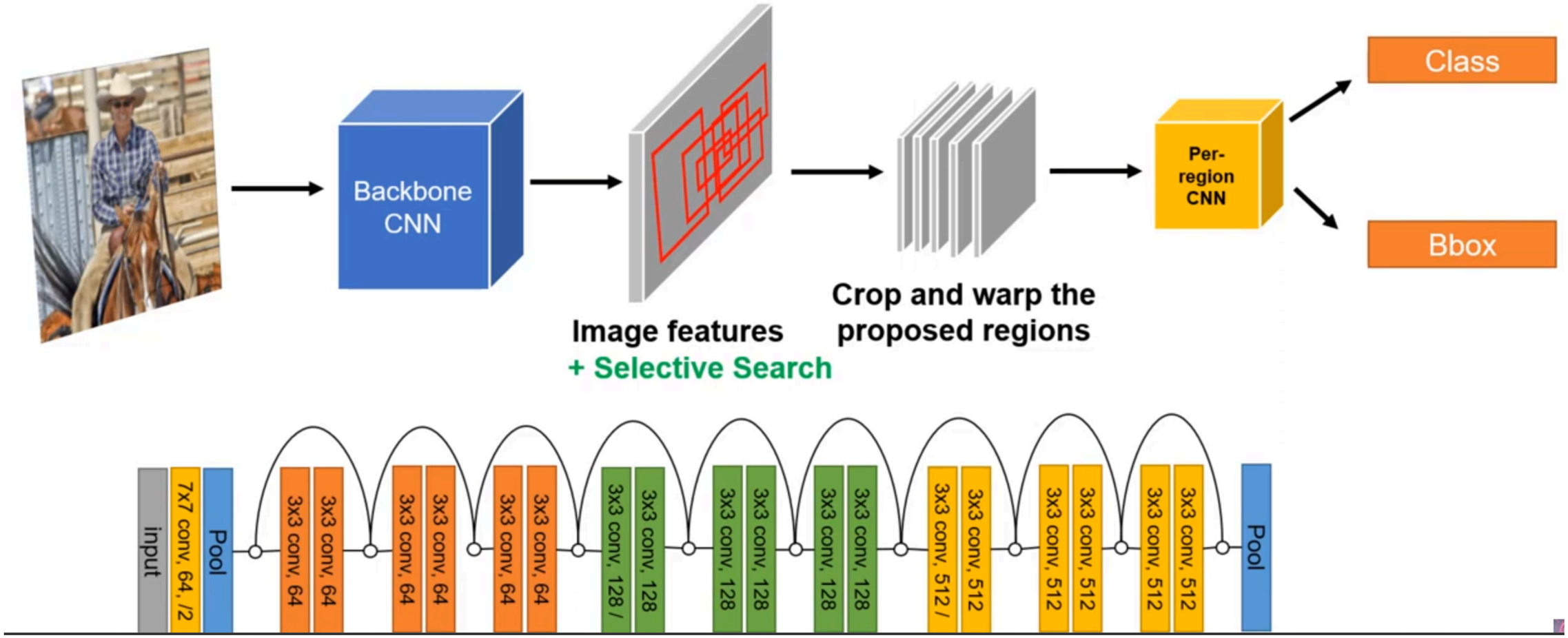
Recap of RCNN



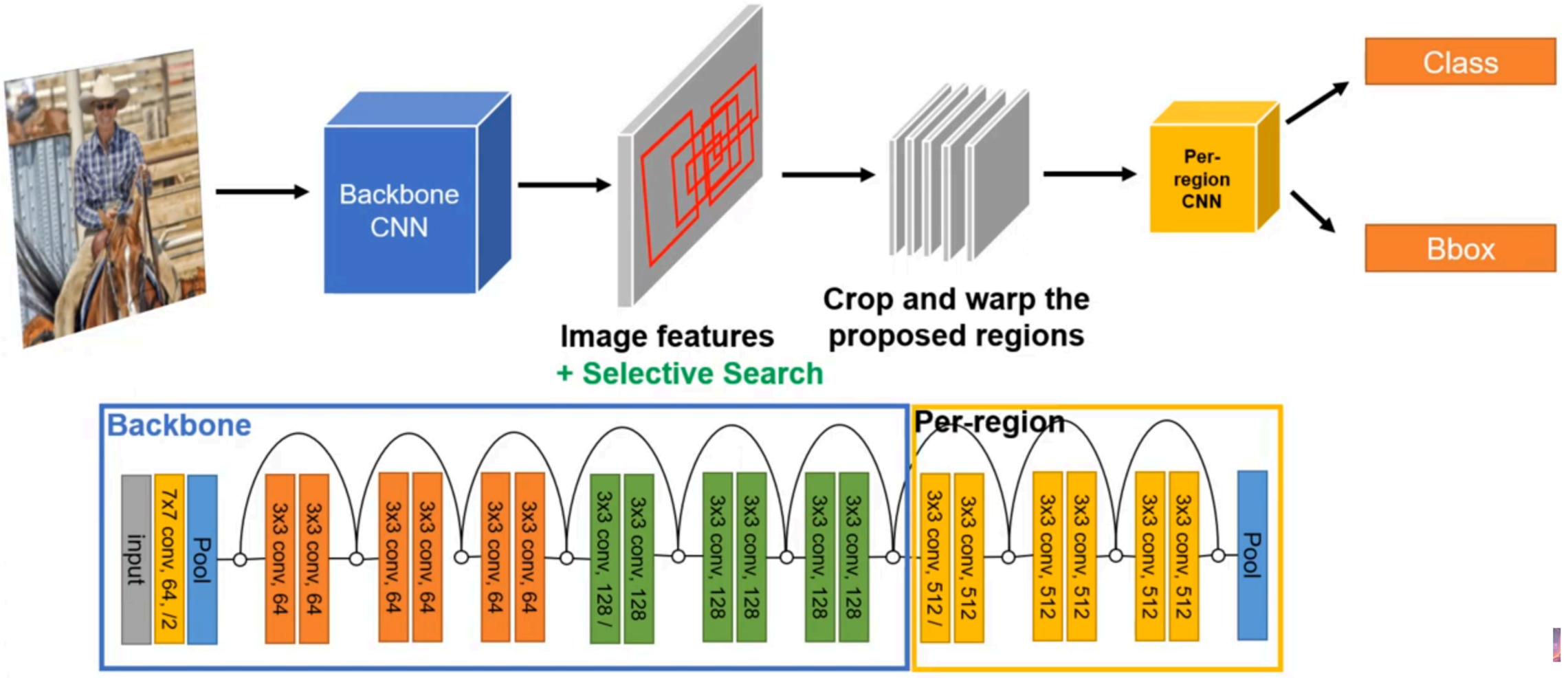
Fast R-CNN


Backbone CNN
Image features
+ Selective Search
Crop and warp the proposed regions
Per-region CNN
Class
Bbox

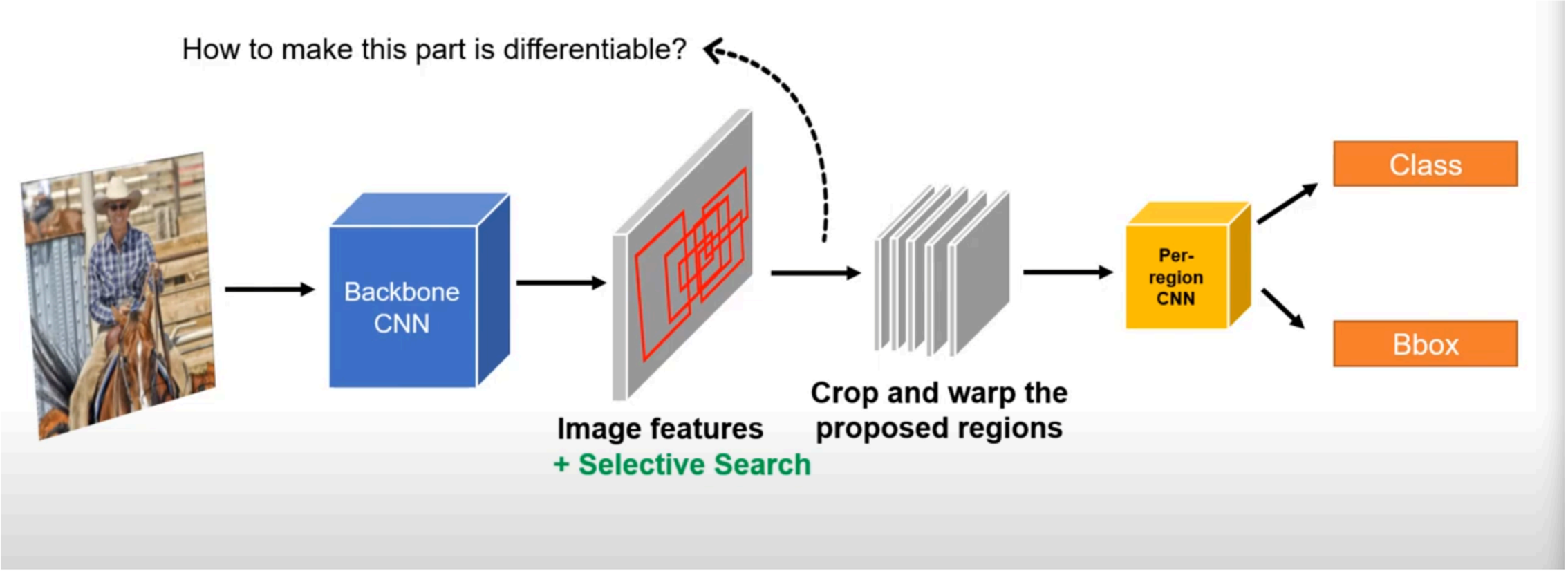
Fast R-CNN


Backbone CNN
Image features
+ Selective Search
Crop and warp the proposed regions
Per-region CNN
Class
Bbox
input
7x7 conv, 64, /2
Pool
3x3 conv, 64
3x3 conv, 128 /
3x3 conv, 128
3x3 conv, 512 /
3x3 conv, 512
Pool

Fast R-CNN

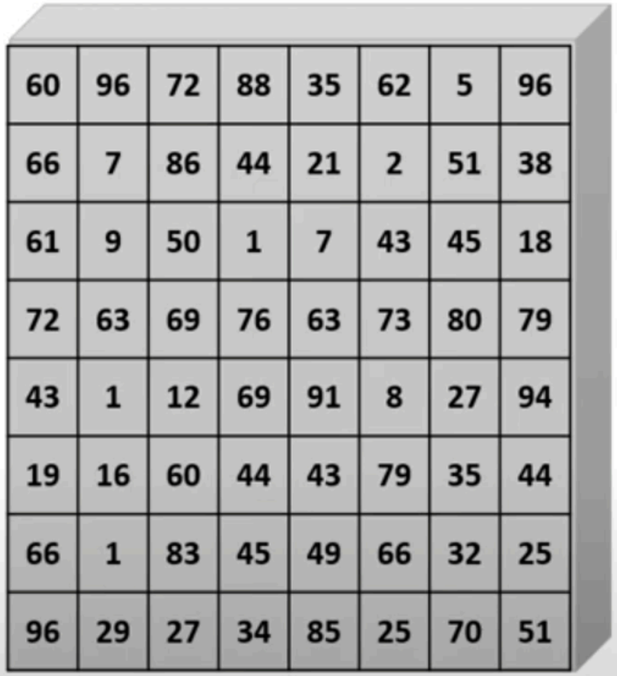

Backbone CNN
Image features
+ Selective Search
Crop and warp the proposed regions
Per-region CNN
Class
Bbox
Backbone
input
7x7 conv, 64, /2
Pool
3x3 conv, 64
3x3 conv, 128 /
3x3 conv, 128
3x3 conv, 512 /
3x3 conv, 512
Per-region
Pool

Fast R-CNN


How to make this part is differentiable?
Backbone CNN
Image features
+ Selective Search
Crop and warp the proposed regions
Per-region CNN
Class
Bbox

ROI Pooling

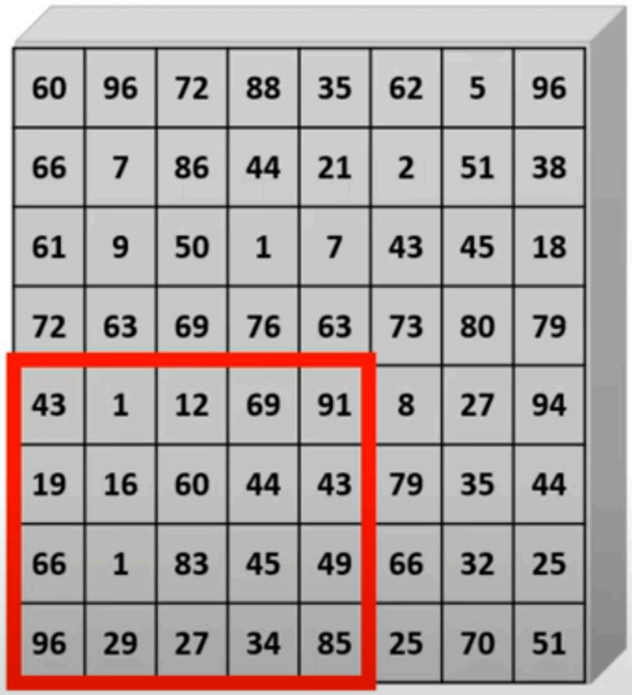
ROI Pooling



60	96	72	88	35	62	5	96
66	7	86	44	21	2	51	38
61	9	50	1	7	43	45	18
72	63	69	76	63	73	80	79
43	1	12	69	91	8	27	94
19	16	60	44	43	79	35	44
66	1	83	45	49	66	32	25
96	29	27	34	85	25	70	51

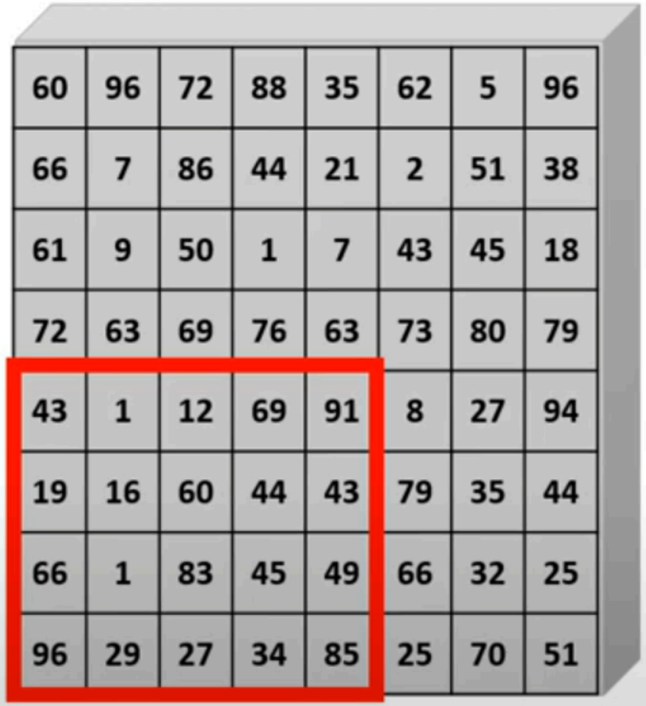
RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

ROI Pooling



RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

ROI Pooling

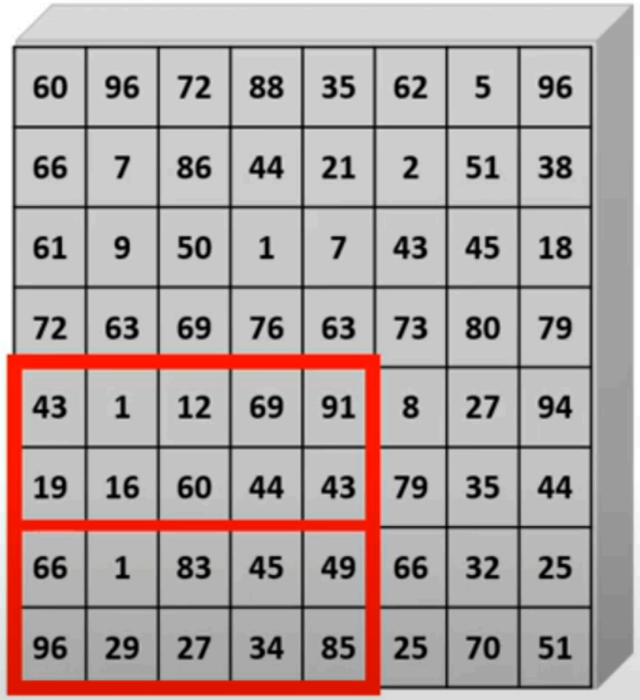


h = 4 w = 5

H = 2 W = 2

RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

ROI Pooling

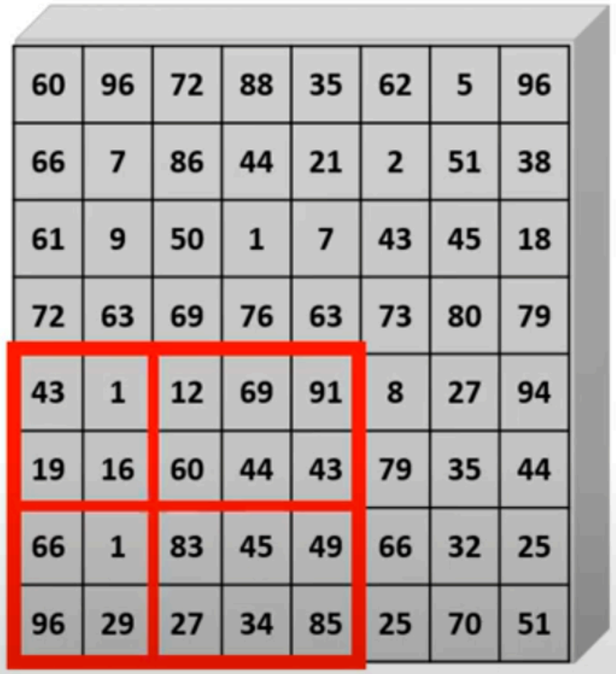


h = 4 w = 5

H = 2 W = 2

RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

ROI Pooling

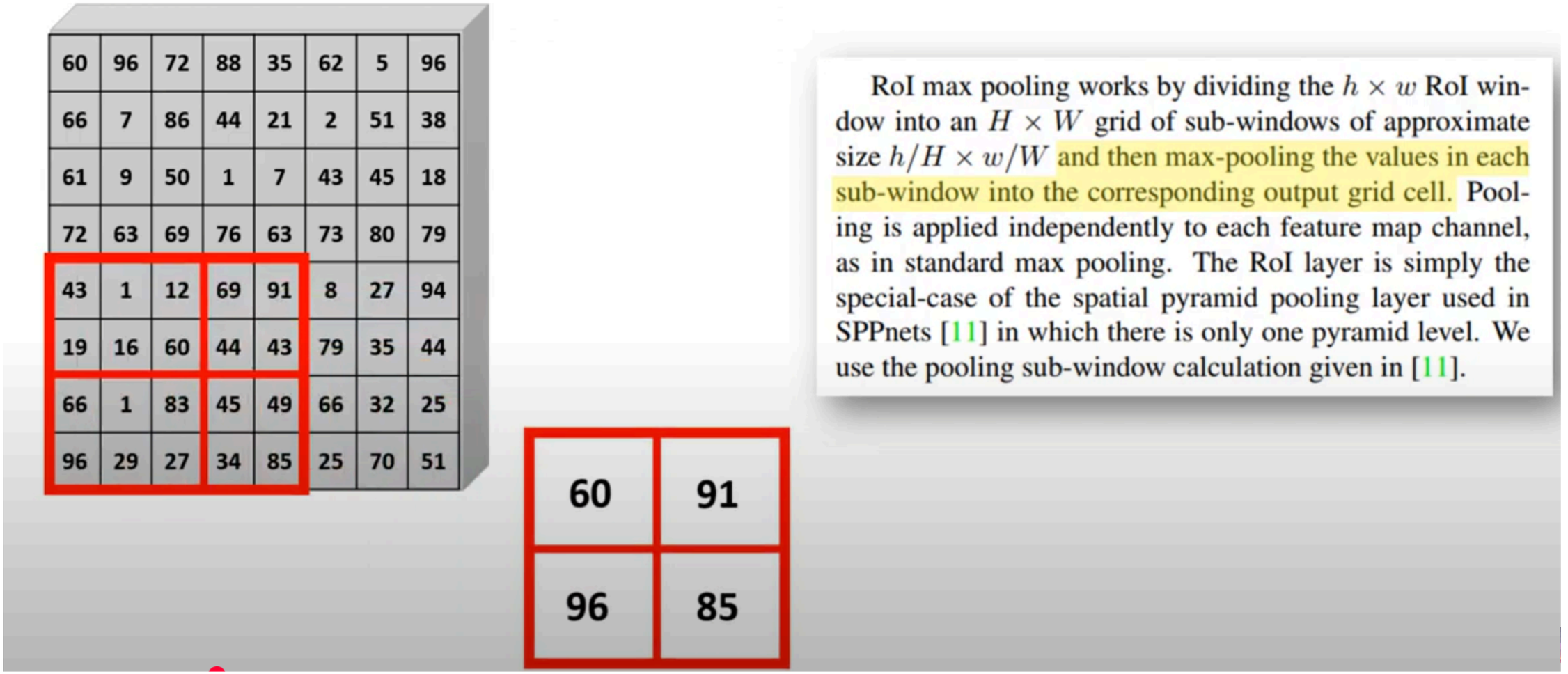


h = 4 w = 5

H = 2 W = 2

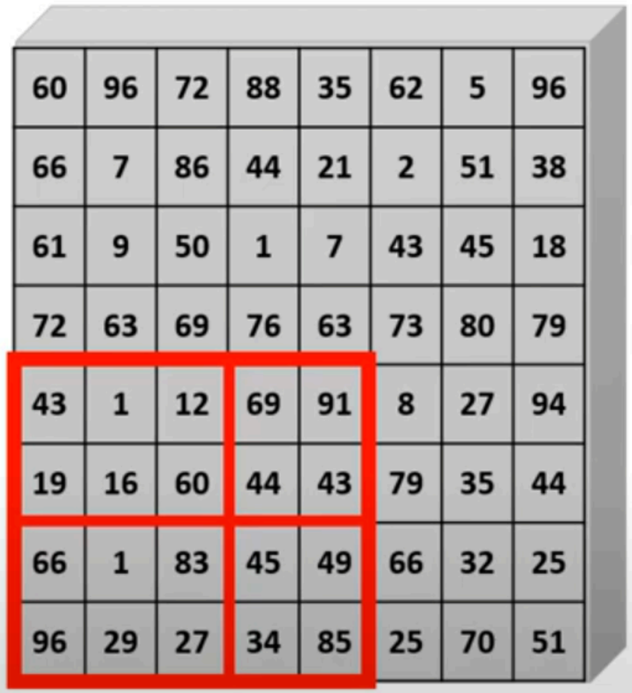
RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

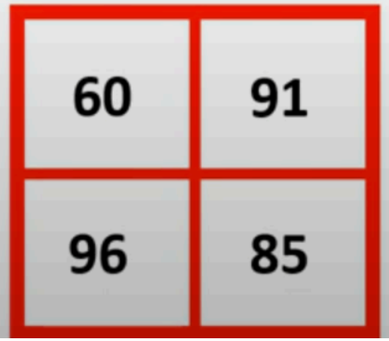
ROI Pooling



RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

ROI Pooling





RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

How to train our model?

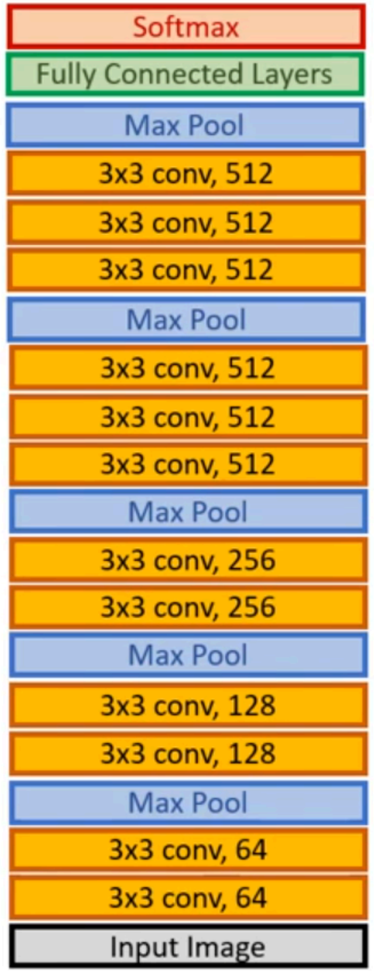
Initializing from pre-trained networks

We experiment with three pre-trained ImageNet [4] networks, each with five max pooling layers and between five and thirteen conv layers (see Section 4.1 for network details)

Our experiments use three pre-trained ImageNet models that are available online.[2] The first is the CaffeNet (essentially AlexNet [14]) from R-CNN [9]. We alternatively refer to this CaffeNet as model **S**, for "small." The second network is VGG_CNN_M_1024 from [3], which has the same depth as **S**, but is wider. We call this network model **M**, for "medium." The final network is the very deep VGG16 model from [20]. Since this model is the largest, we call it model **L**. In this section, all experiments use *single-scale* training and testing ($s = 600$; see Section 5.2 for details).

Initializing from pre-trained networks

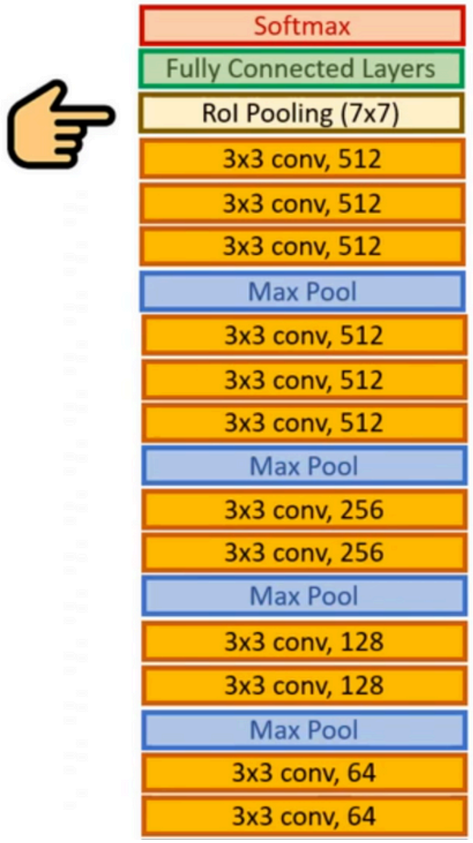
it undergoes three transformations.



First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (*e.g.*, $H = W = 7$ for VGG16).

Initializing from pre-trained networks

it undergoes three transformations.



First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (*e.g.*, $H = W = 7$ for VGG16).

Initializing from pre-trained networks

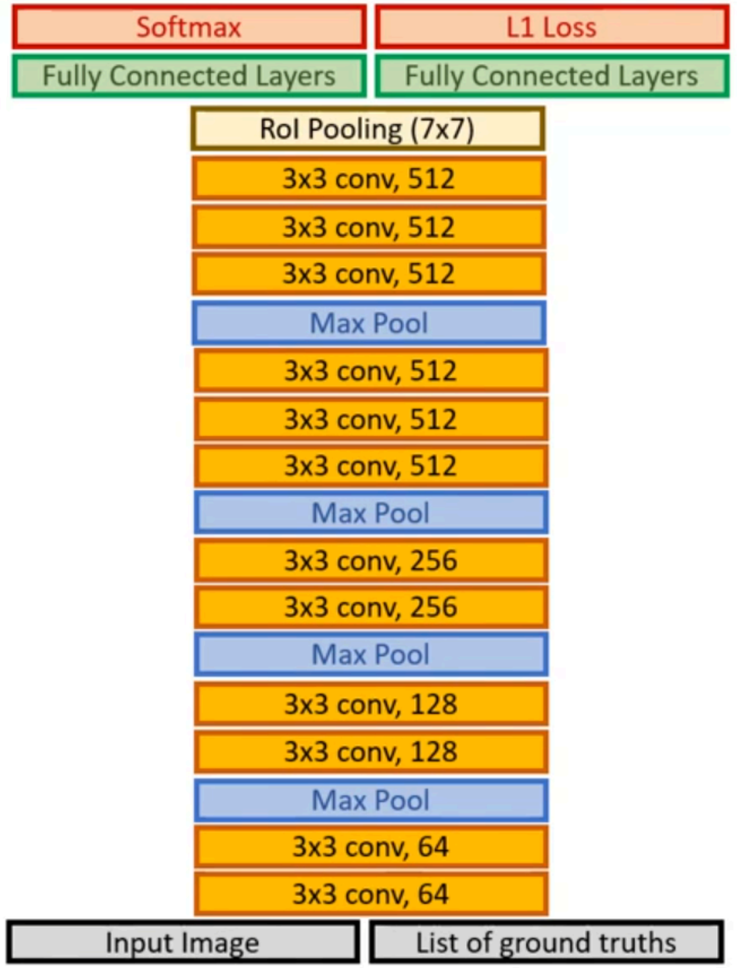
it undergoes three transformations.

First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (*e.g.*, $H = W = 7$ for VGG16).

Second, the network's last fully connected layer and softmax (which were trained for 1000-way ImageNet classification) are replaced with the two sibling layers described earlier (a fully connected layer and softmax over $K+1$ categories and category-specific bounding-box regressors).

Initializing from pre-trained networks

it undergoes three transformations.

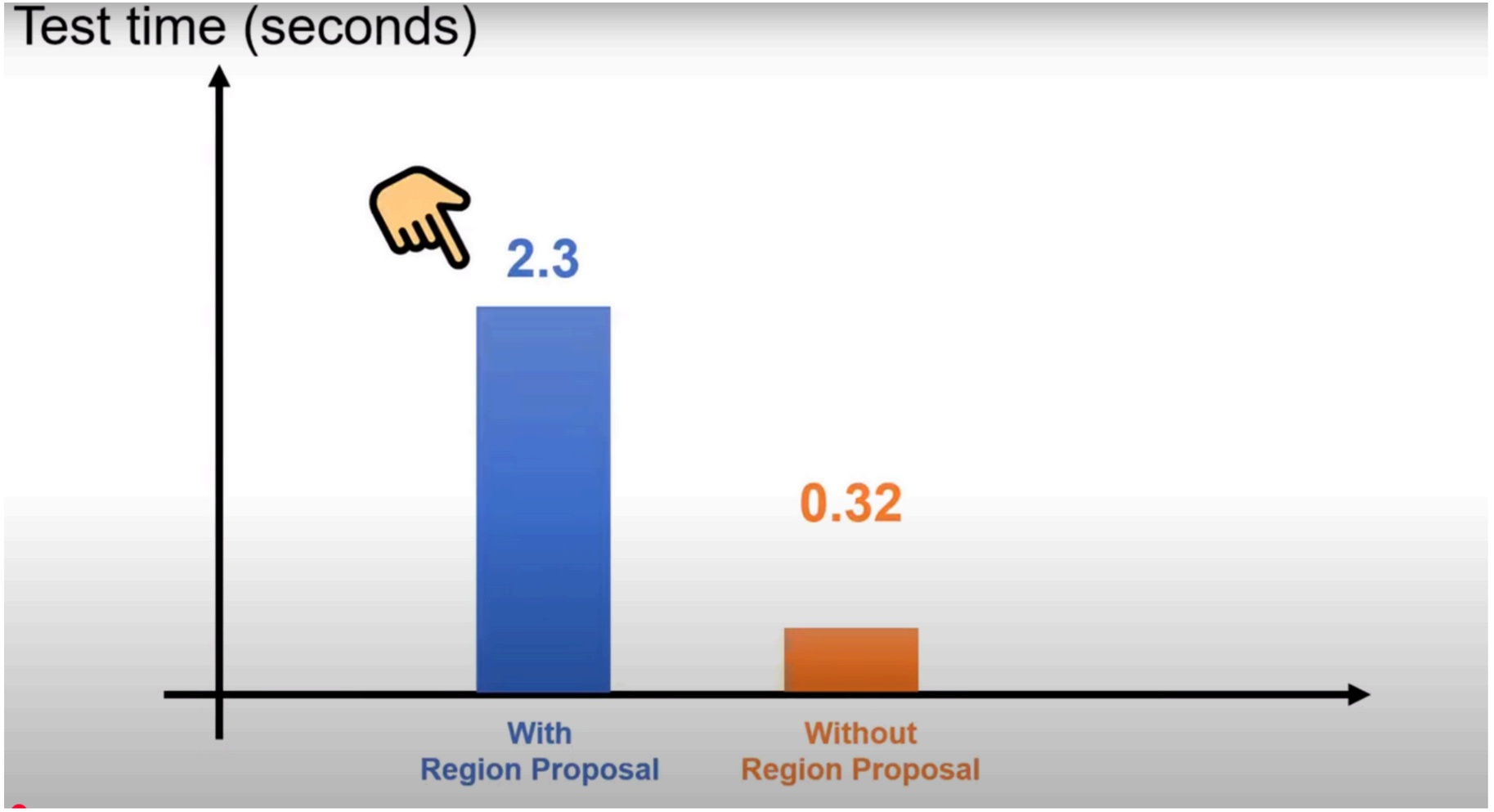


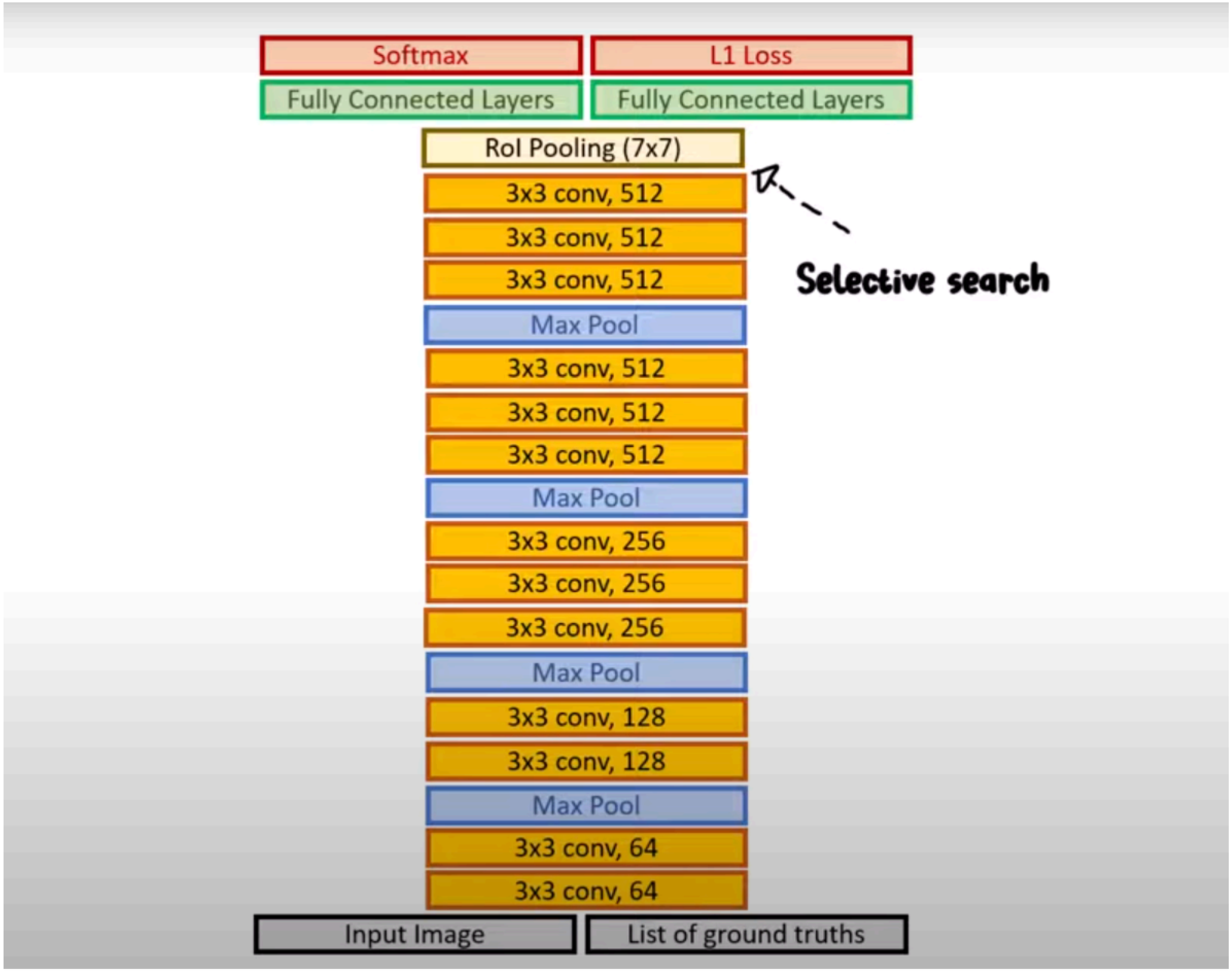
First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (*e.g.*, $H = W = 7$ for VGG16).

Second, the network's last fully connected layer and softmax (which were trained for 1000-way ImageNet classification) are replaced with the two sibling layers described earlier (a fully connected layer and softmax over $K+1$ categories and category-specific bounding-box regressors).

Third, the network is modified to take two data inputs: a list of images and a list of RoIs in those images.

Faster R-CNN


Test time (seconds)
2.3
0.32
With
Region Proposal
Without
Region Proposal


Softmax
L1 Loss
Fully Connected Layers
Fully Connected Layers
RoI Pooling (7x7)
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Max Pool
3x3 conv, 128
3x3 conv, 128
Max Pool
3x3 conv, 64
3x3 conv, 64
Input Image
List of ground truths
Selective search

Softmax | L1 Loss

Fully Connected Layers | Fully Connected Layers

RoI Pooling (7x7)

RPN

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Max Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Max Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Max Pool

3x3 conv, 128

3x3 conv, 128

Max Pool

3x3 conv, 64

3x3 conv, 64

Input Image | List of ground truths

Inside RPN

600

600

Softmax | L1 Loss

Fully Connected Layers | Fully Connected Layers

RoI Pooling (7x7)

RPN

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Max Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Max Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Max Pool

3x3 conv, 128

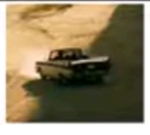
3x3 conv, 128

Max Pool

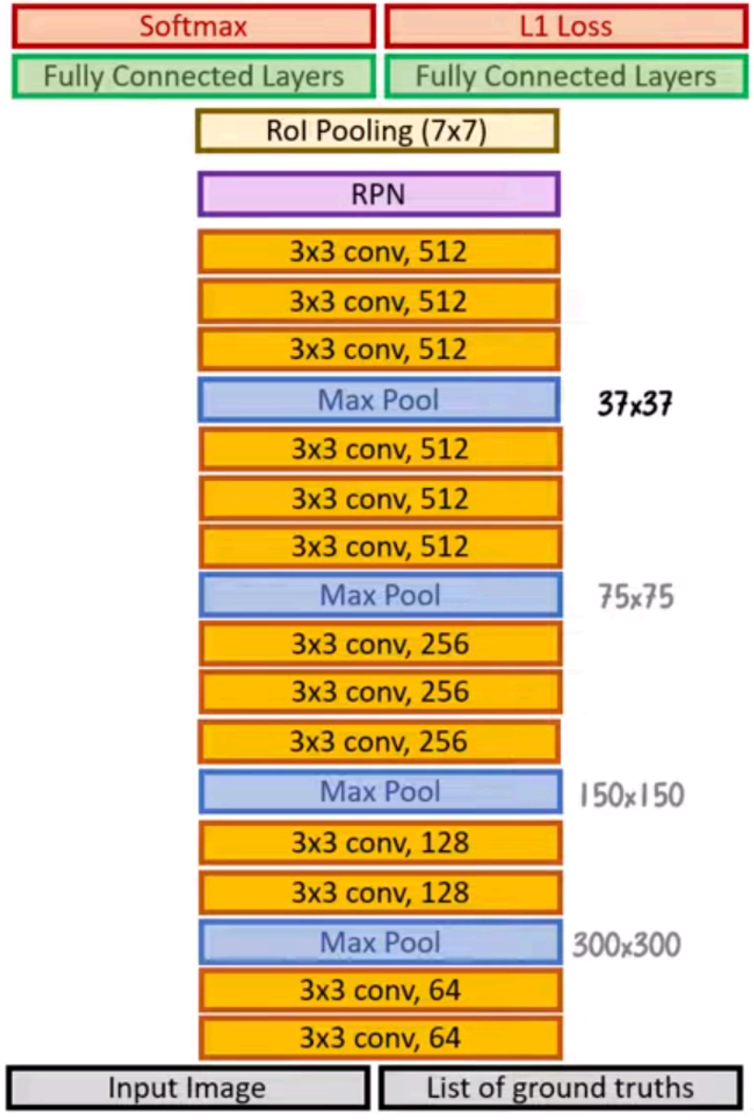
3x3 conv, 64

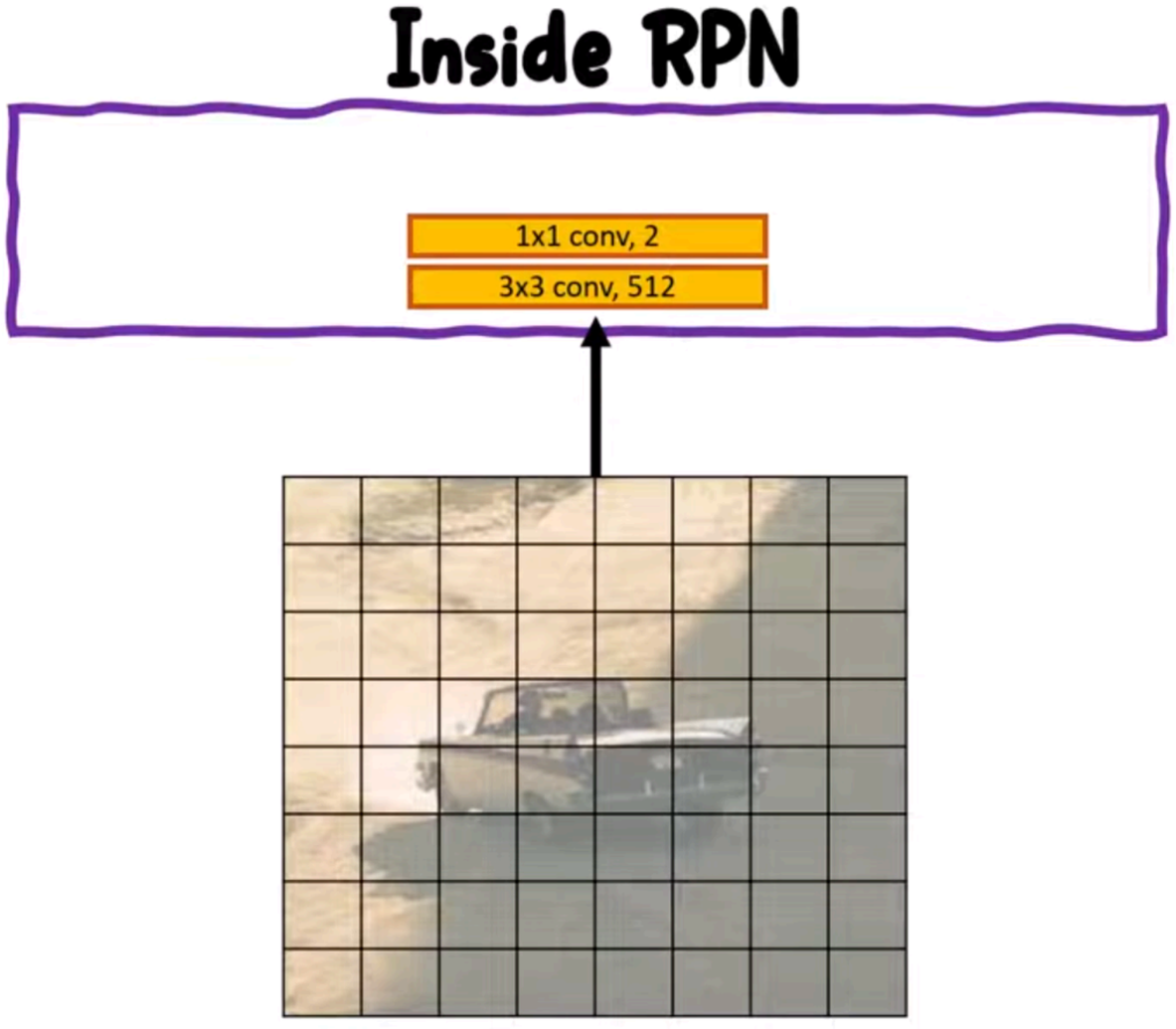
3x3 conv, 64

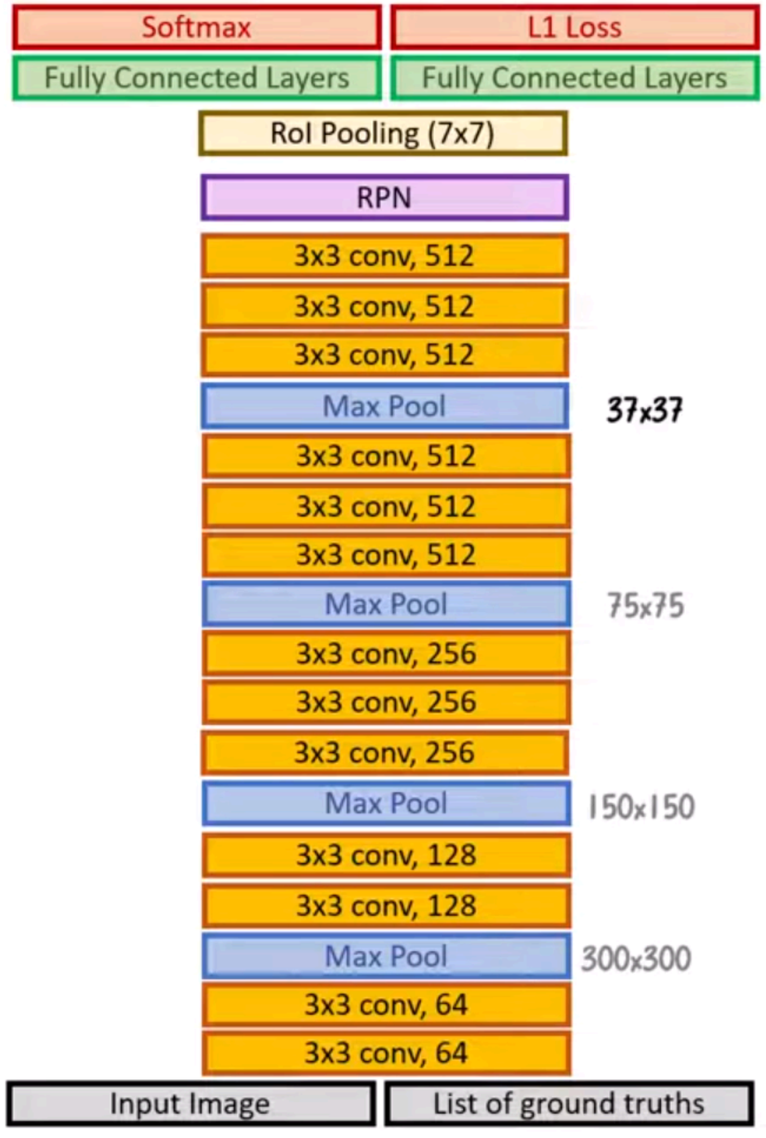
Input Image | List of ground truths

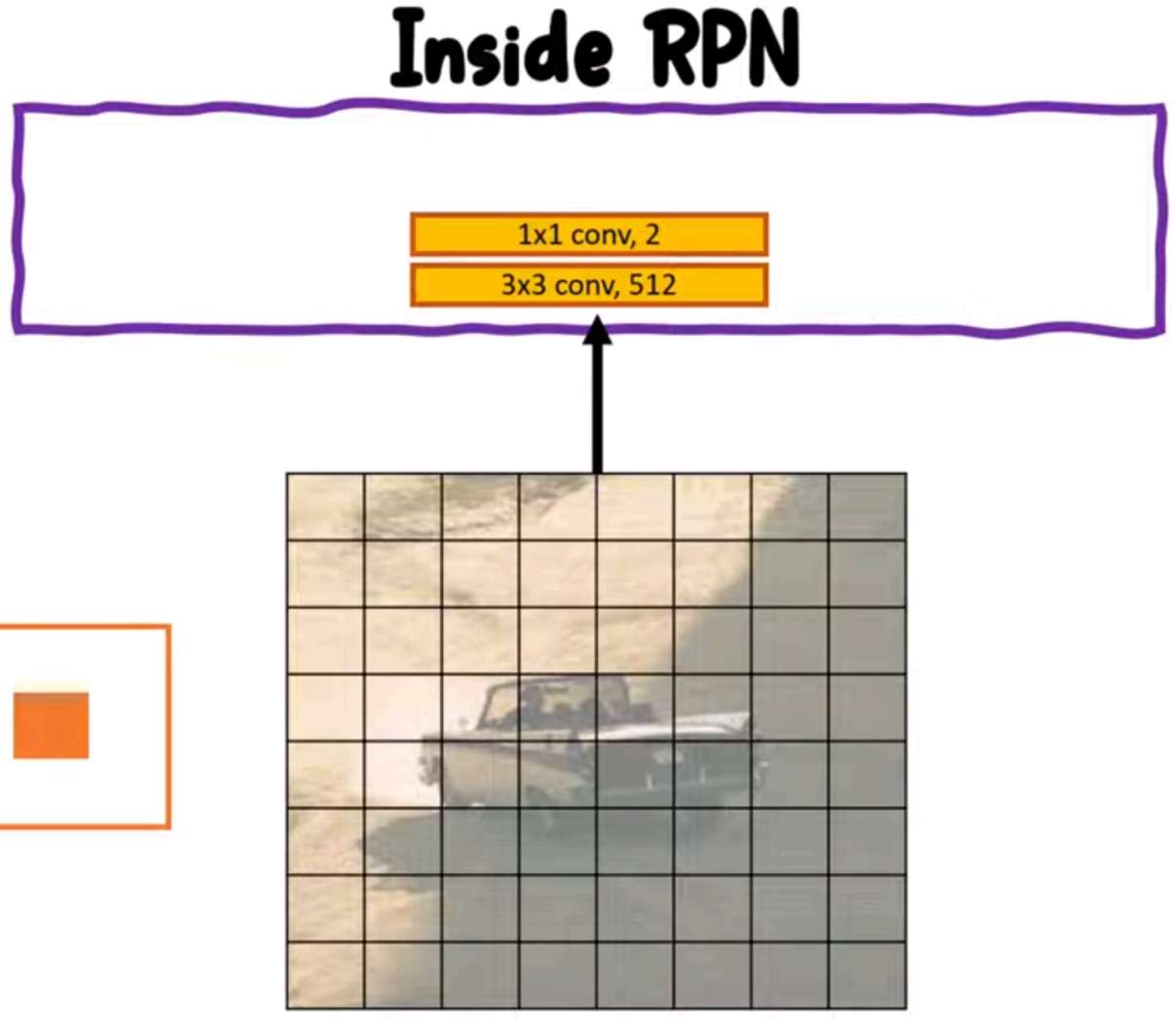


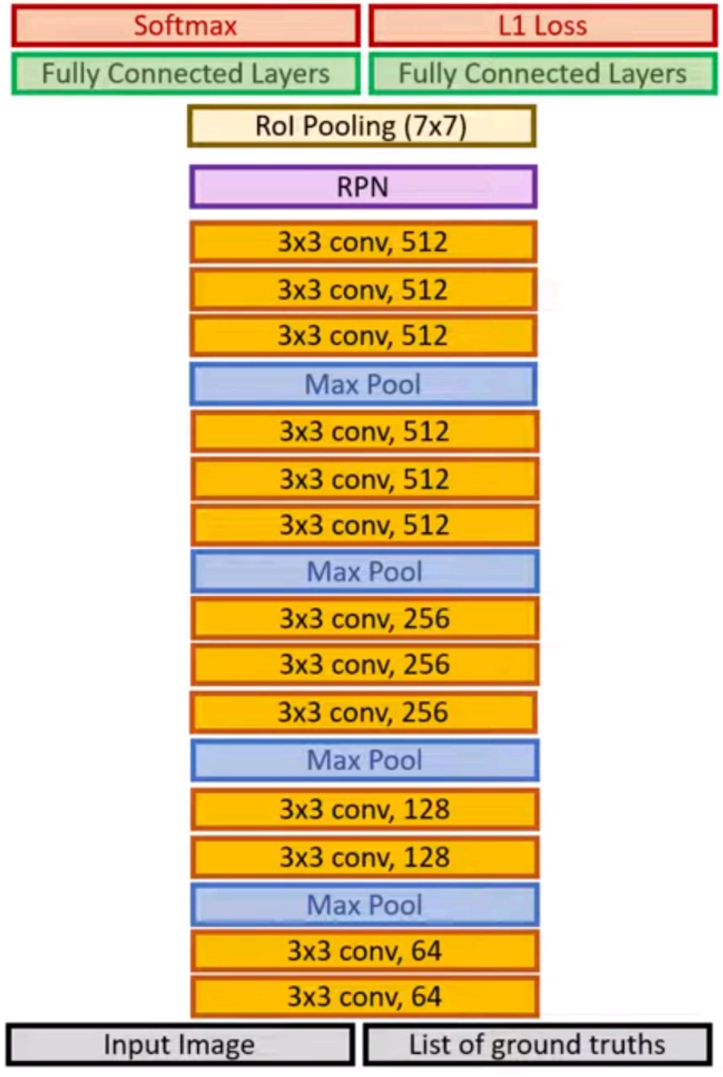
Inside RPN

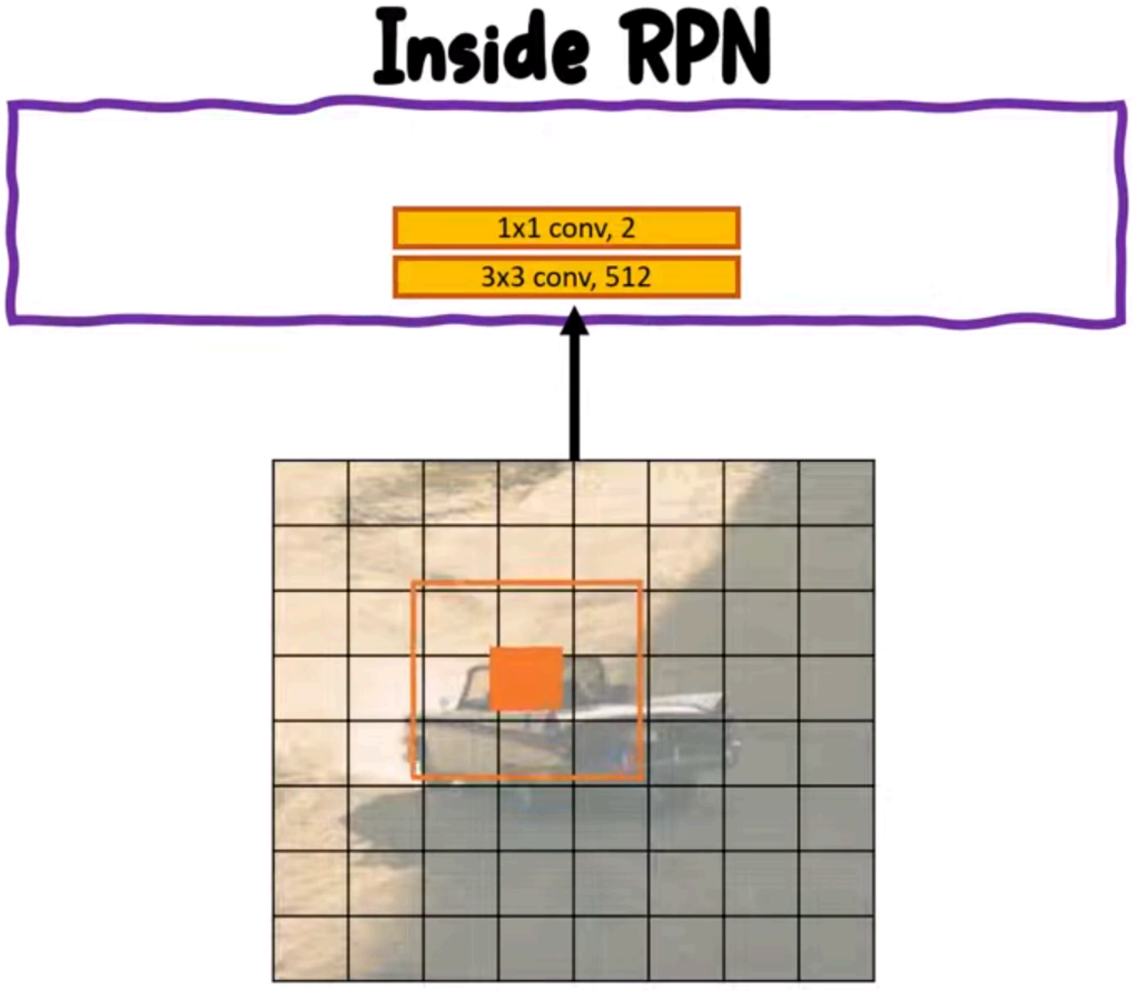

Softmax
L1 Loss
Fully Connected Layers
Fully Connected Layers
RoI Pooling (7x7)
RPN
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
37x37
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
75x75
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Max Pool
150x150
3x3 conv, 128
3x3 conv, 128
Max Pool
300x300
3x3 conv, 64
3x3 conv, 64
Input Image
List of ground truths

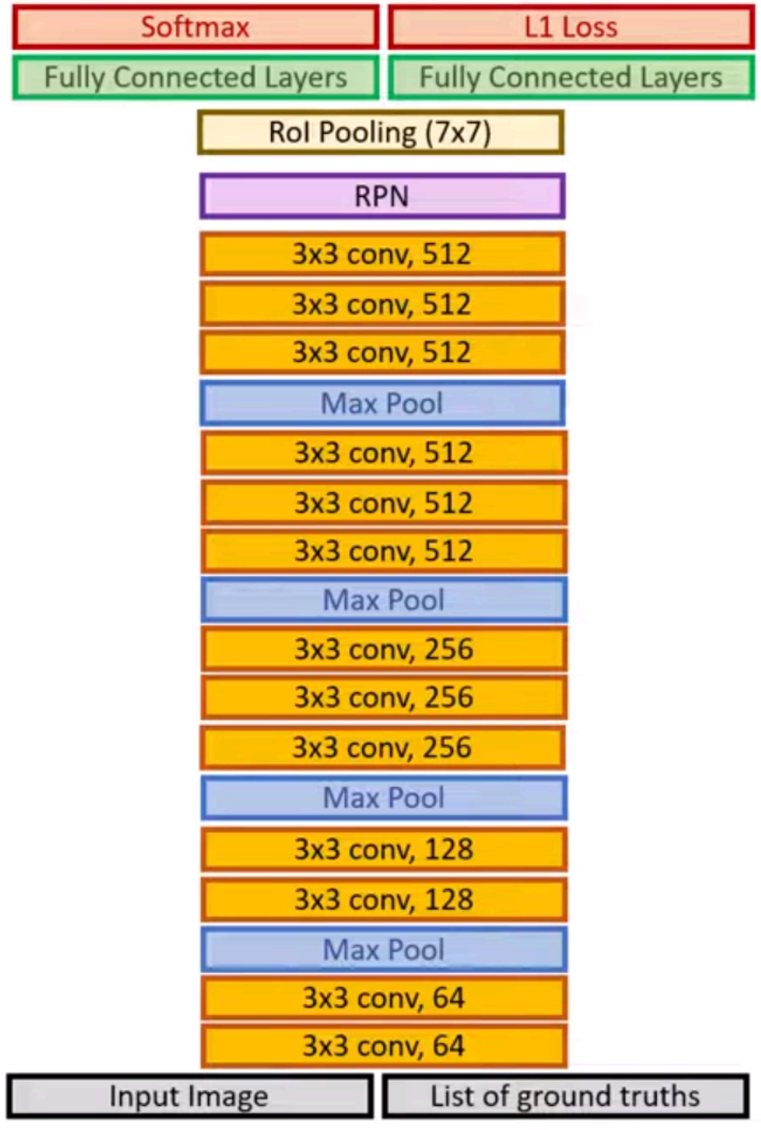

Inside RPN
1x1 conv, 2
3x3 conv, 512

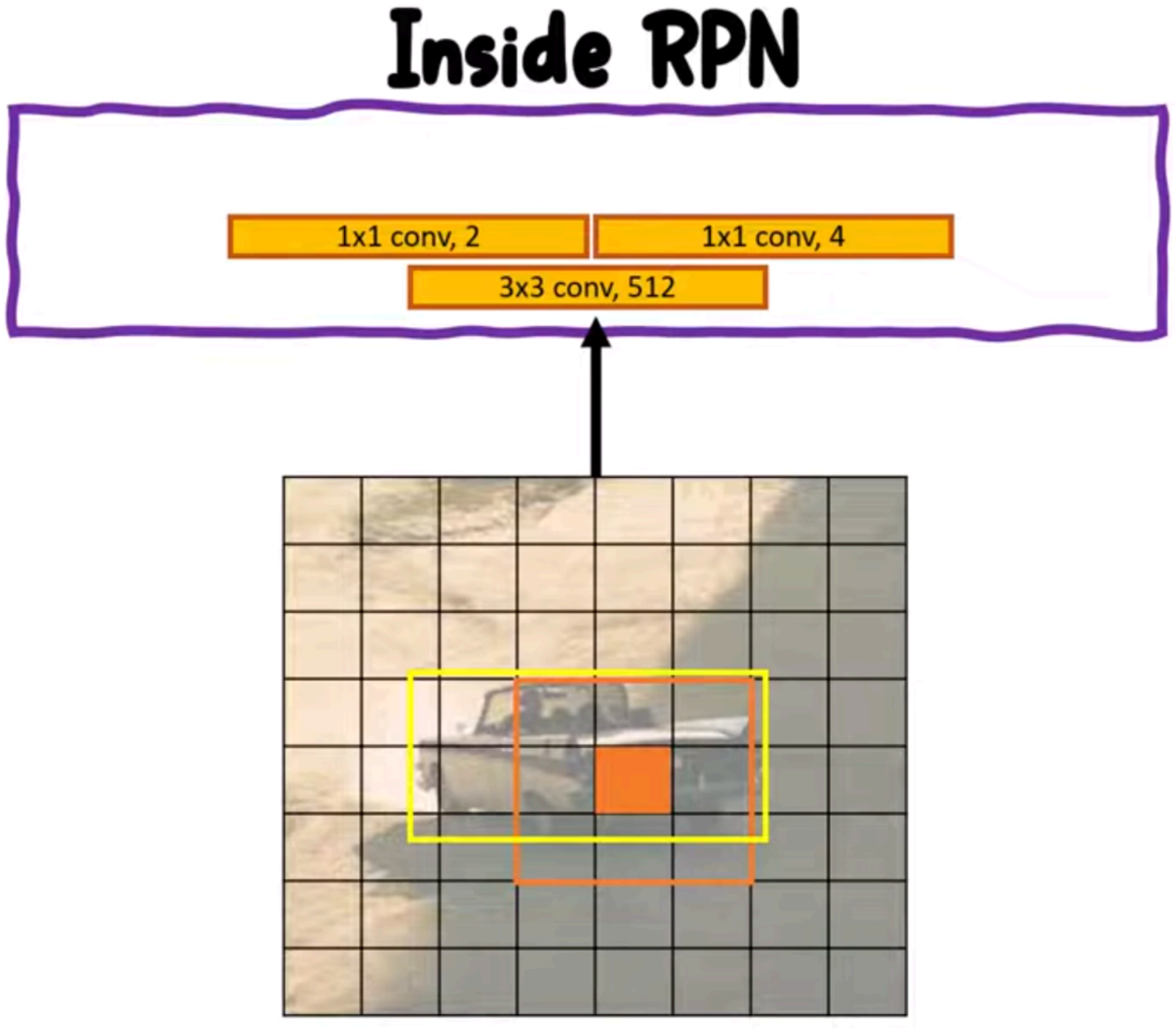

Softmax
L1 Loss
Fully Connected Layers
Fully Connected Layers
RoI Pooling (7x7)
RPN
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
37x37
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
75x75
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Max Pool
150x150
3x3 conv, 128
3x3 conv, 128
Max Pool
300x300
3x3 conv, 64
3x3 conv, 64
Input Image
List of ground truths

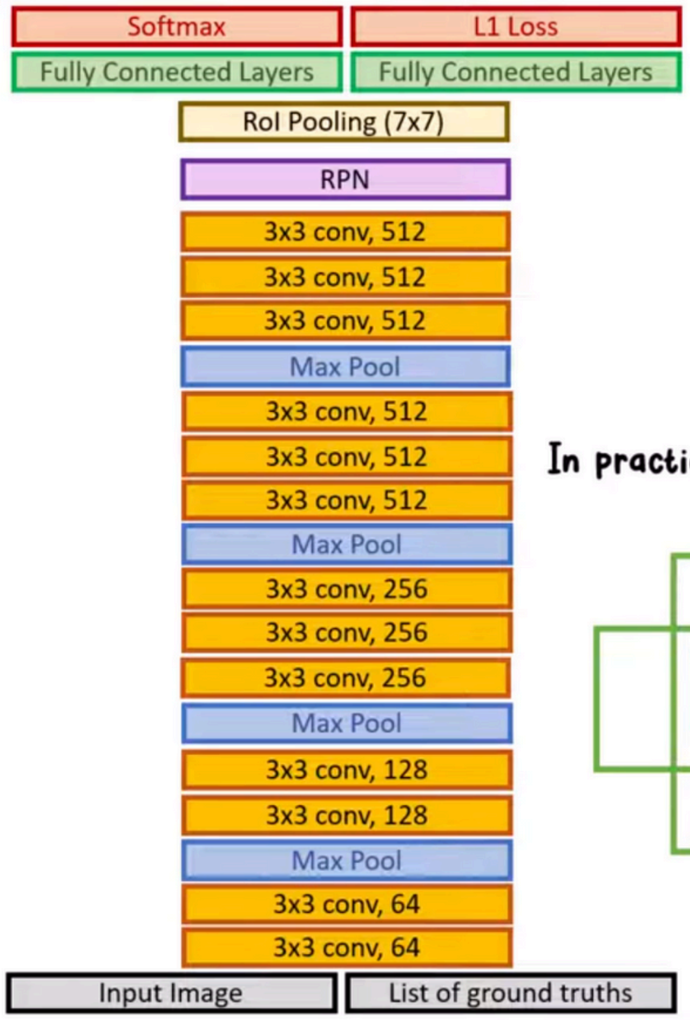

Inside RPN
1x1 conv, 2
3x3 conv, 512

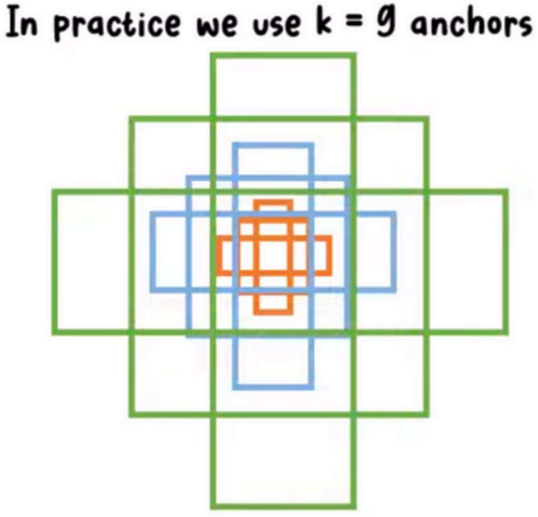

Softmax
L1 Loss
Fully Connected Layers
Fully Connected Layers
RoI Pooling (7x7)
RPN
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Max Pool
3x3 conv, 128
3x3 conv, 128
Max Pool
3x3 conv, 64
3x3 conv, 64
Input Image
List of ground truths

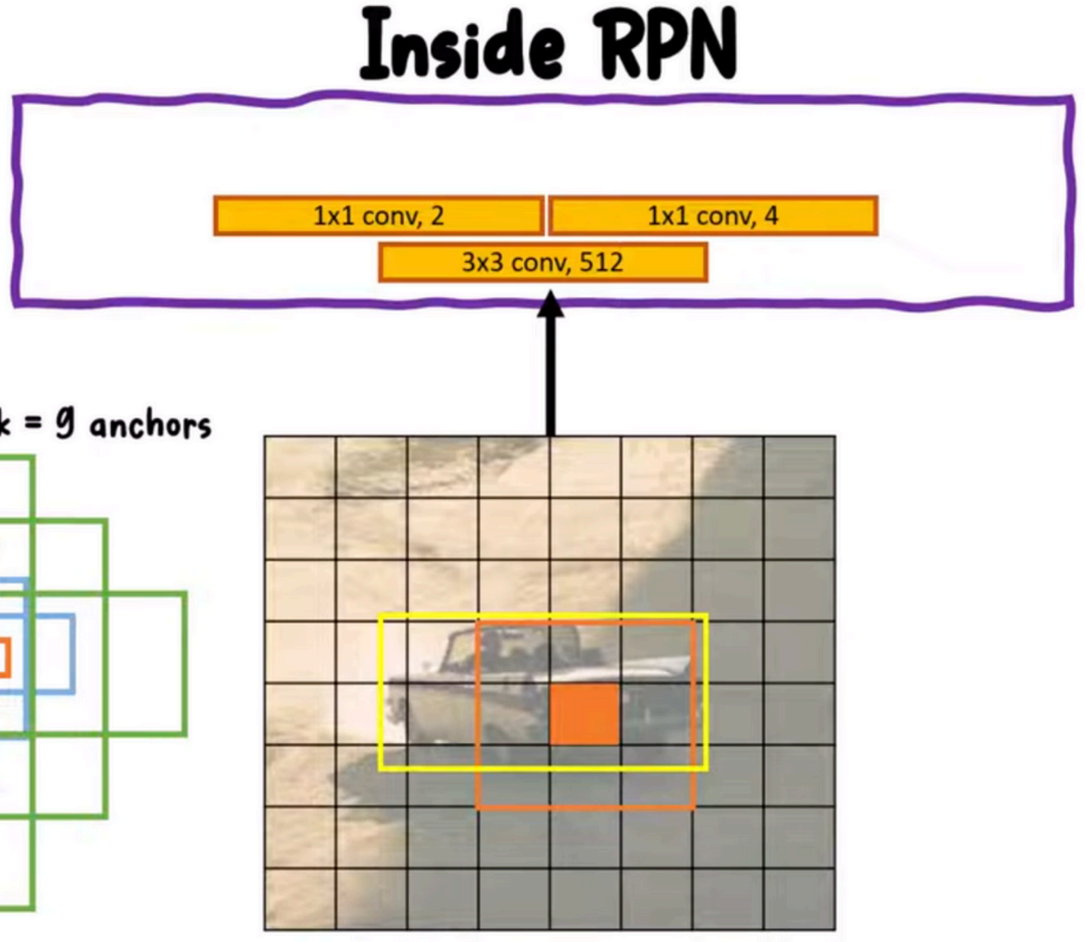

Inside RPN
1x1 conv, 2
3x3 conv, 512


Softmax
L1 Loss
Fully Connected Layers
Fully Connected Layers
RoI Pooling (7x7)
RPN
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Max Pool
3x3 conv, 128
3x3 conv, 128
Max Pool
3x3 conv, 64
3x3 conv, 64
Input Image
List of ground truths


Inside RPN
1x1 conv, 2
1x1 conv, 4
3x3 conv, 512


Softmax
L1 Loss
Fully Connected Layers
Fully Connected Layers
RoI Pooling (7x7)
RPN
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Max Pool
3x3 conv, 128
3x3 conv, 128
Max Pool
3x3 conv, 64
3x3 conv, 64
Input Image
List of ground truths


In practice we use k = 9 anchors


Inside RPN
1x1 conv, 2
1x1 conv, 4
3x3 conv, 512

Softmax | L1 Loss

Fully Connected Layers | Fully Connected Layers

RoI Pooling (7x7)

RPN

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Max Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Max Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Max Pool

3x3 conv, 128

3x3 conv, 128

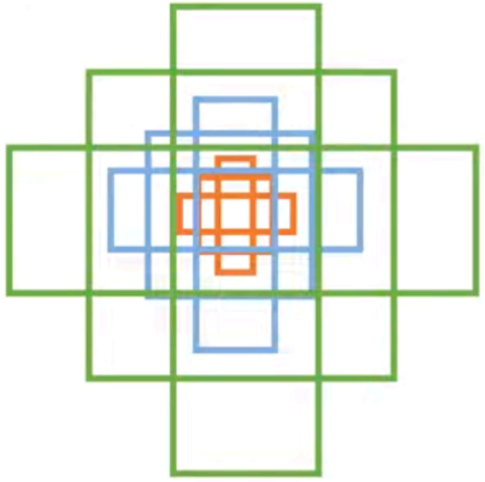
Max Pool

3x3 conv, 64

3x3 conv, 64

Input Image | List of ground truths

In practice we use k = 9 anchors



Inside RPN

1x1 conv, 2*k | 1x1 conv, 4*k

3x3 conv, 512