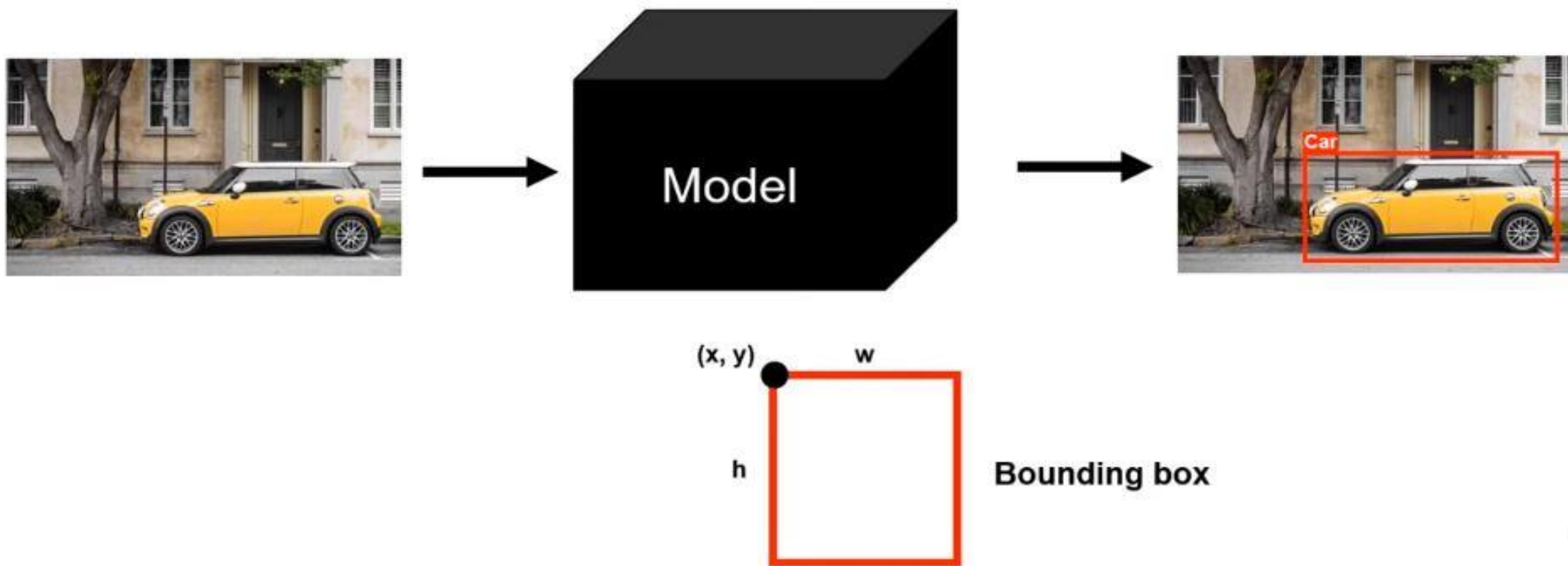


R-CNN

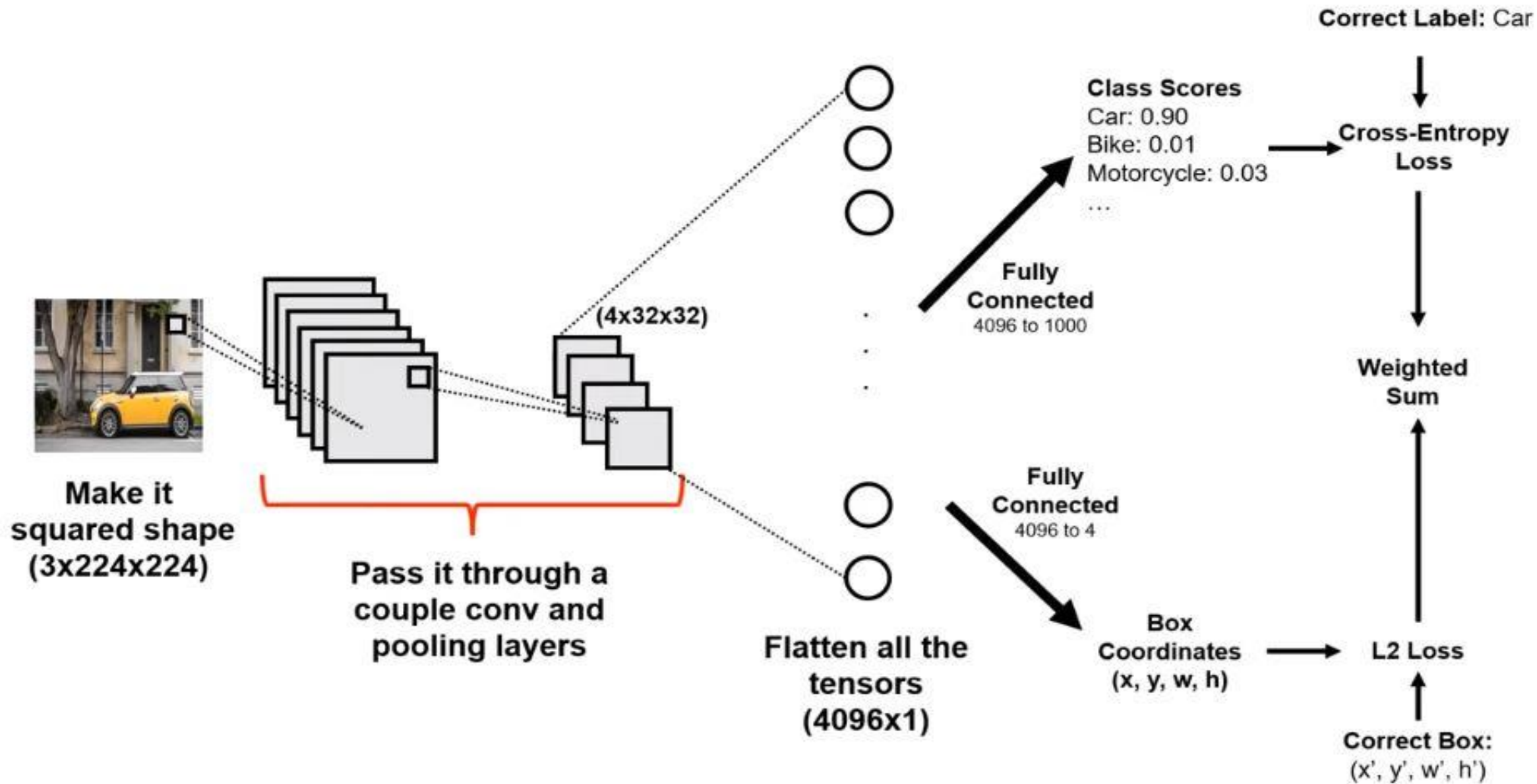


Our Goal

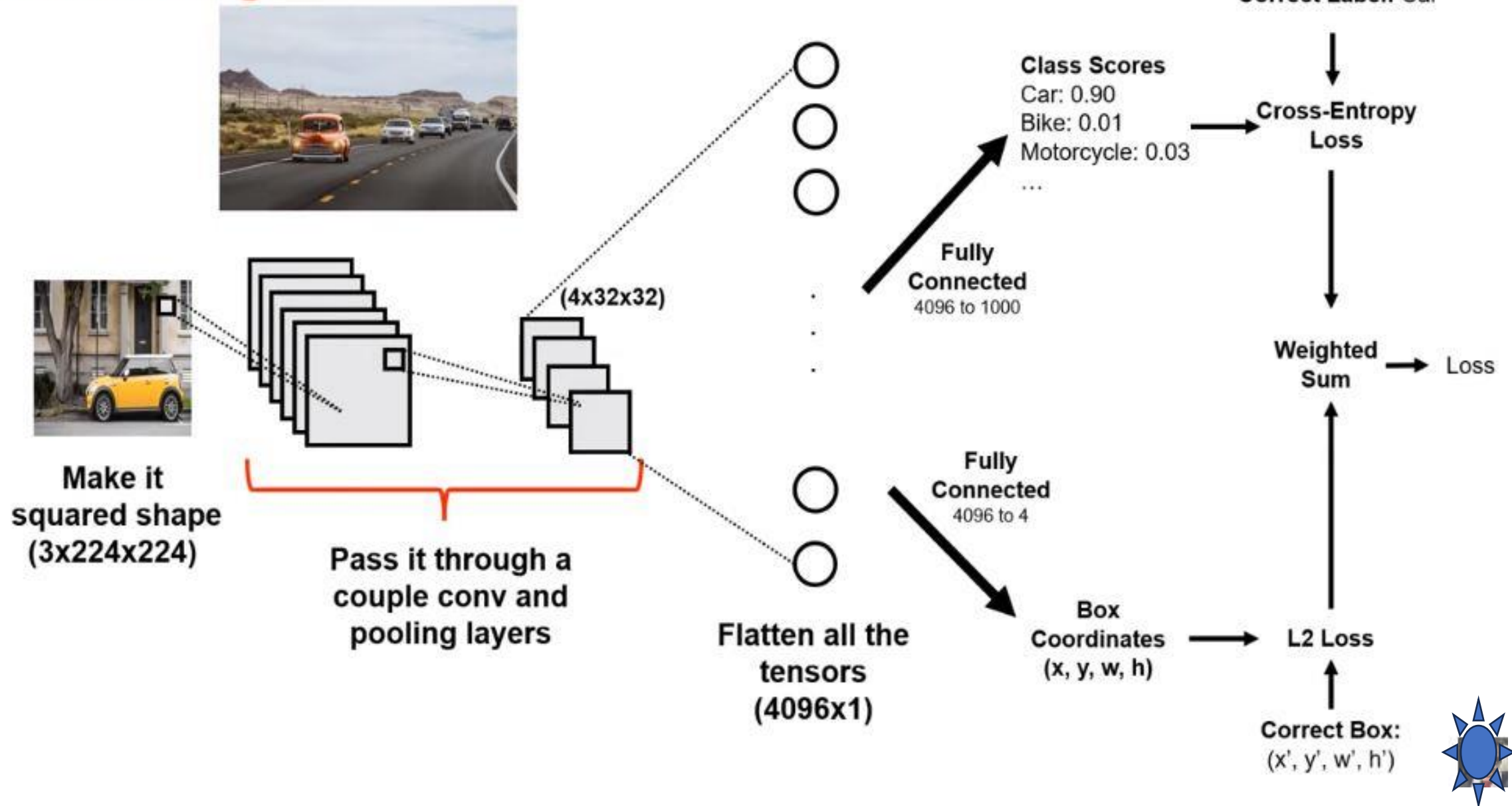


We receive input image





Disadvantage: Only one object can be detected :(



Earliest Approach

Earliest Approach



Sliding Window



Earliest Approach



Classify this region!



Neural Network classifier



$c + 1$ outputs

Car
Motorcycle
Bike
...

Background

Earliest Approach



Classify this region!



Neural Network classifier



Mountain

Earliest Approach



Classify this region!



Neural Network classifier



Car

Earliest Approach

W



H

W



h

Possible Positions:

$$(W - w + 1) * (H - h + 1)$$

CNN as feature extractor

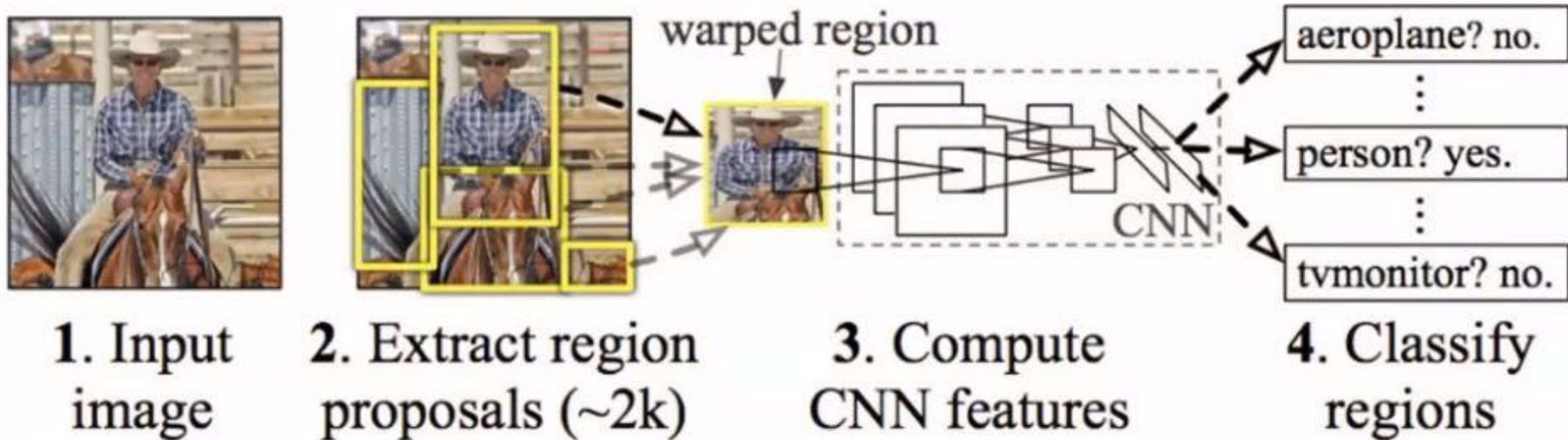
- › What could be the problems?
 - Suppose we have a 600 x 600 image, if sliding window size is 20 x 20, then have $(600-20+1) \times (600-20+1) = \sim 330,000$ windows
 - Sometimes we want to have more accurate results -> multi-scale detection
 - › Resize image
 - › Multi-scale sliding window

Disadvantages

- Very Slow
- Number of Picked windows is very huge
- For CNN classifier, needs to apply convolution to each window content
- Same object will be detected in multiple windows (with different Bounding Boxes)

R-CNN

R-CNN: *Regions with CNN features*



(Girshick, et al., 2014)

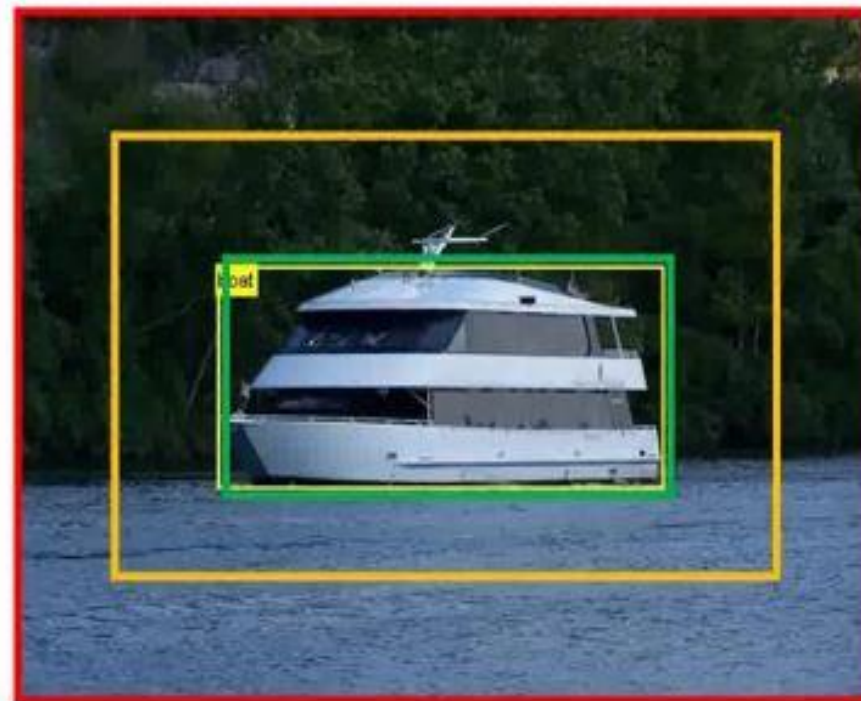
Bounding Box Regression Training

$(x1, y1) = (200, 250)$

$(x2, y2) = (600, 400)$

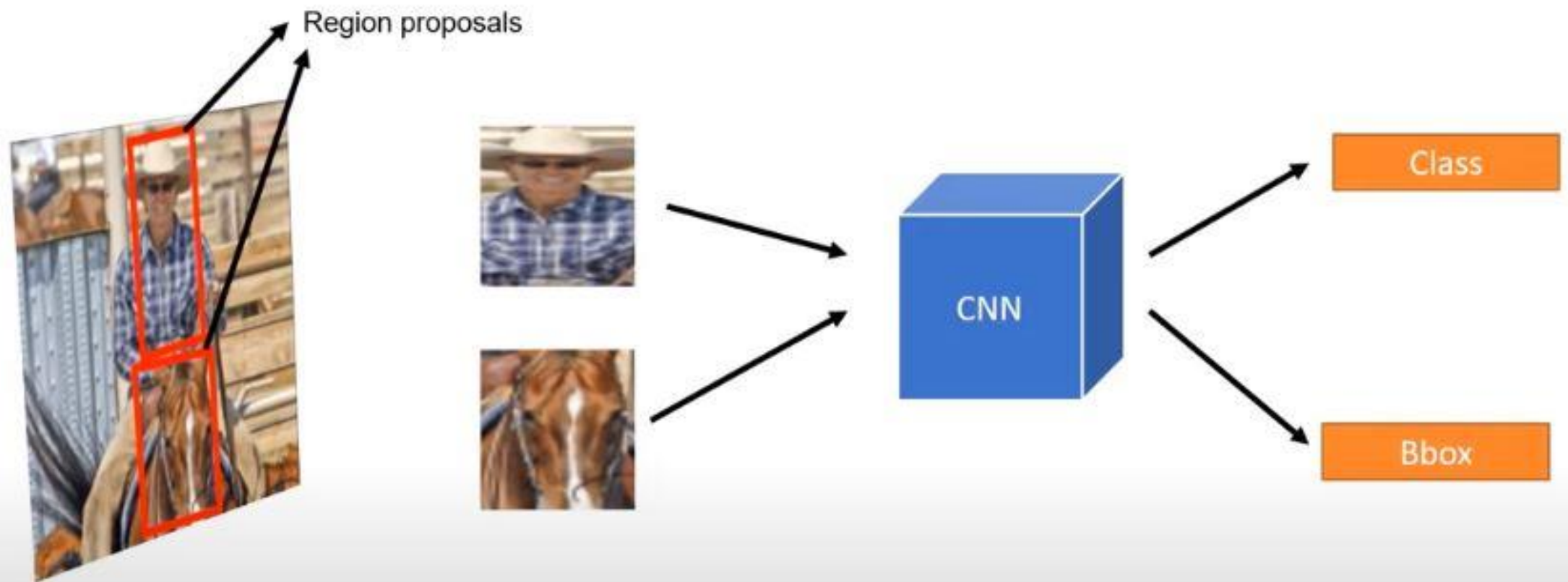
800

600



	x1	y1	x2	y2	L2 Loss				
Expected	200	250	600	400					
Prediction	0	0	800	600	$(200-0)^2$	$(250-0)^2$	$(600-800)^2$	$(400-600)^2$	182500
	100	150	700	450	$(200-100)^2$	$(250-150)^2$	$(600-700)^2$	$(400-450)^2$	32500
	210	245	590	405	$(200-210)^2$	$(250-245)^2$	$(600-590)^2$	$(400-405)^2$	250
	200	250	600	400	$(200-200)^2$	$(250-250)^2$	$(600-600)^2$	$(400-400)^2$	0

R-CNN



R-CNN

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)

Region Based CNN

R-CNN (Region proposal + CNN)

Selective Search



Color

Texture

Circles and Curves

Selective Search (simplified)

- Group based on intensity of the pixels.



Input Image

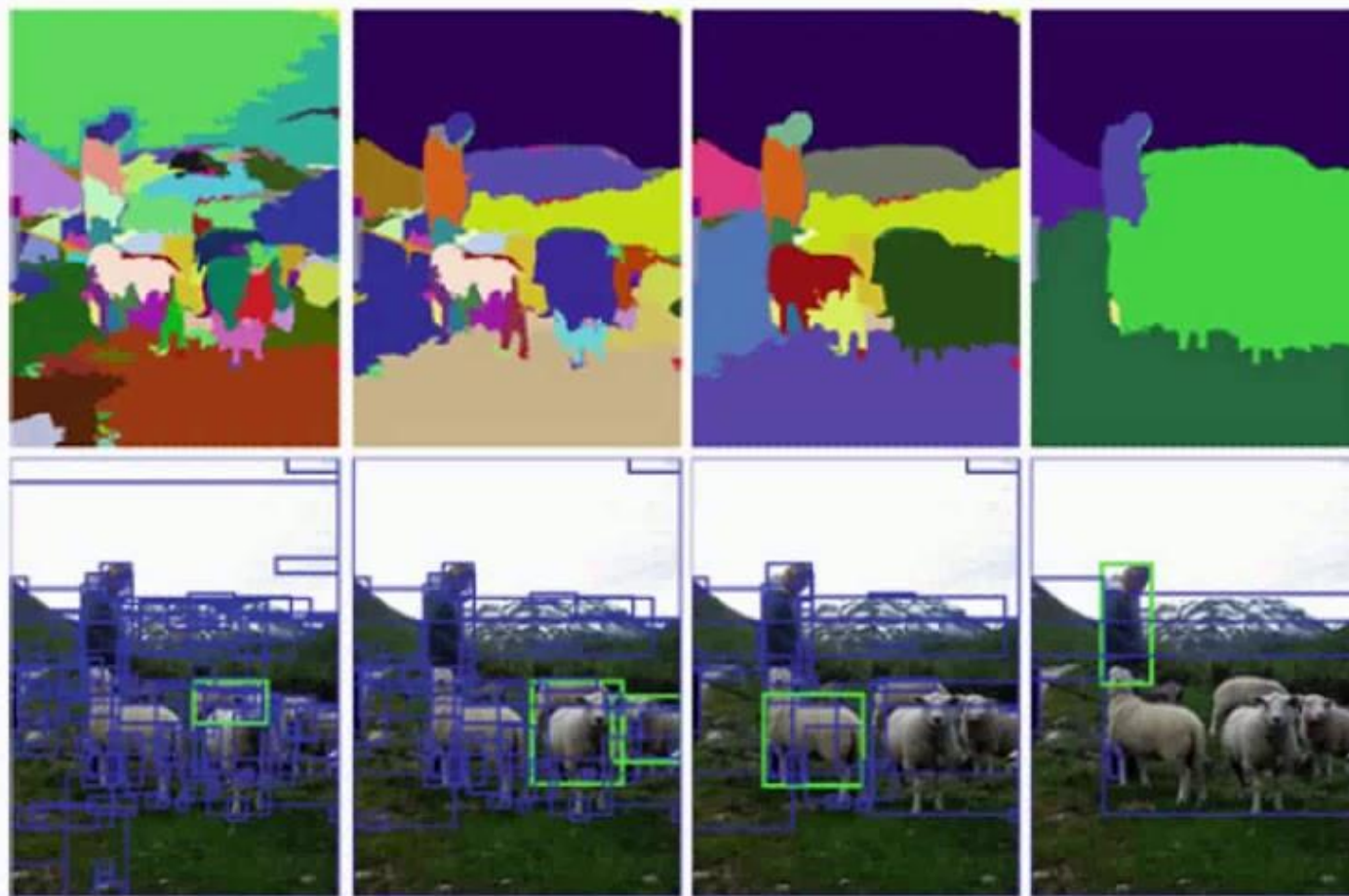


Output Image

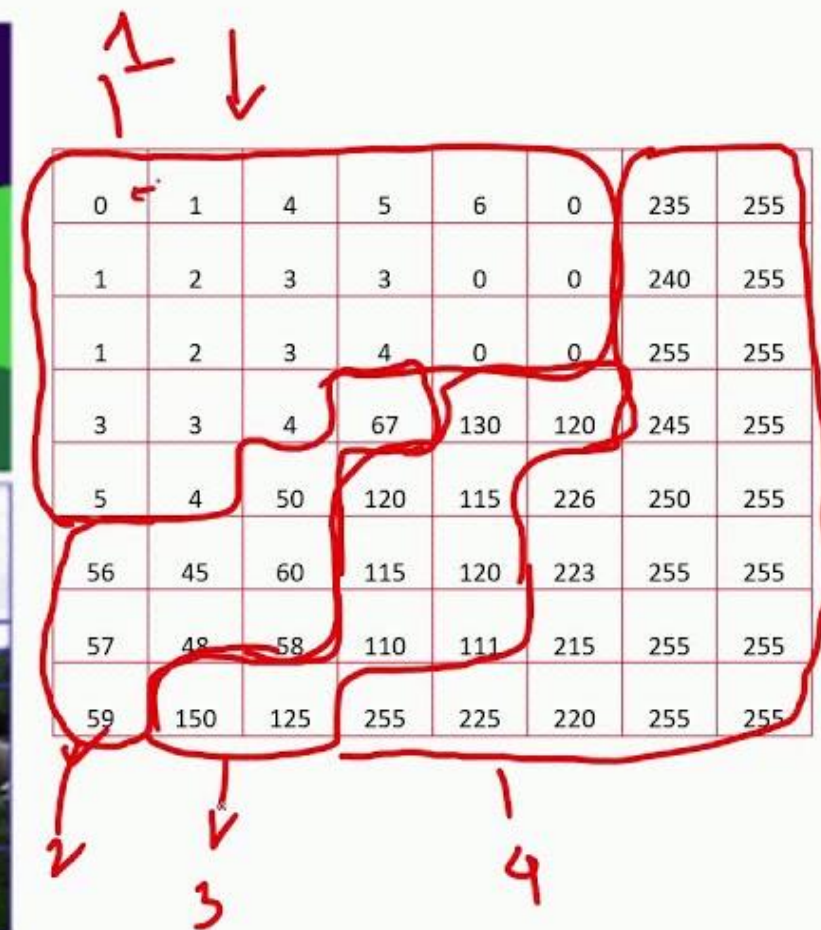
(Chandel, 2017)

- We cannot directly use the segmented image as region proposals!
- Group adjacent segments by similarity.

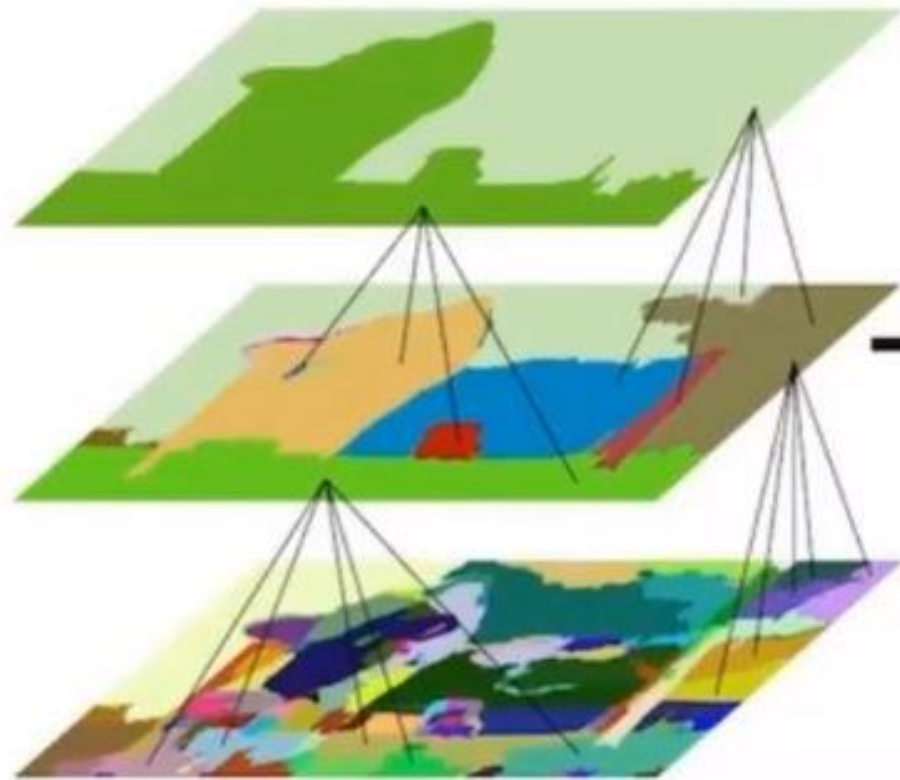
Region Proposal Techniques



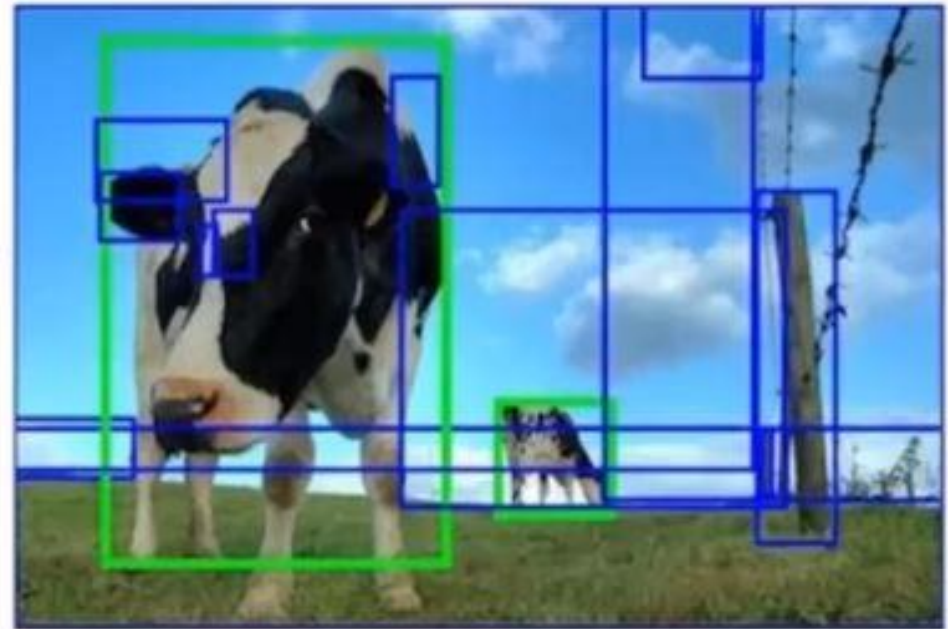
Superpixels Straddling ✓



Selective Search (simplified)

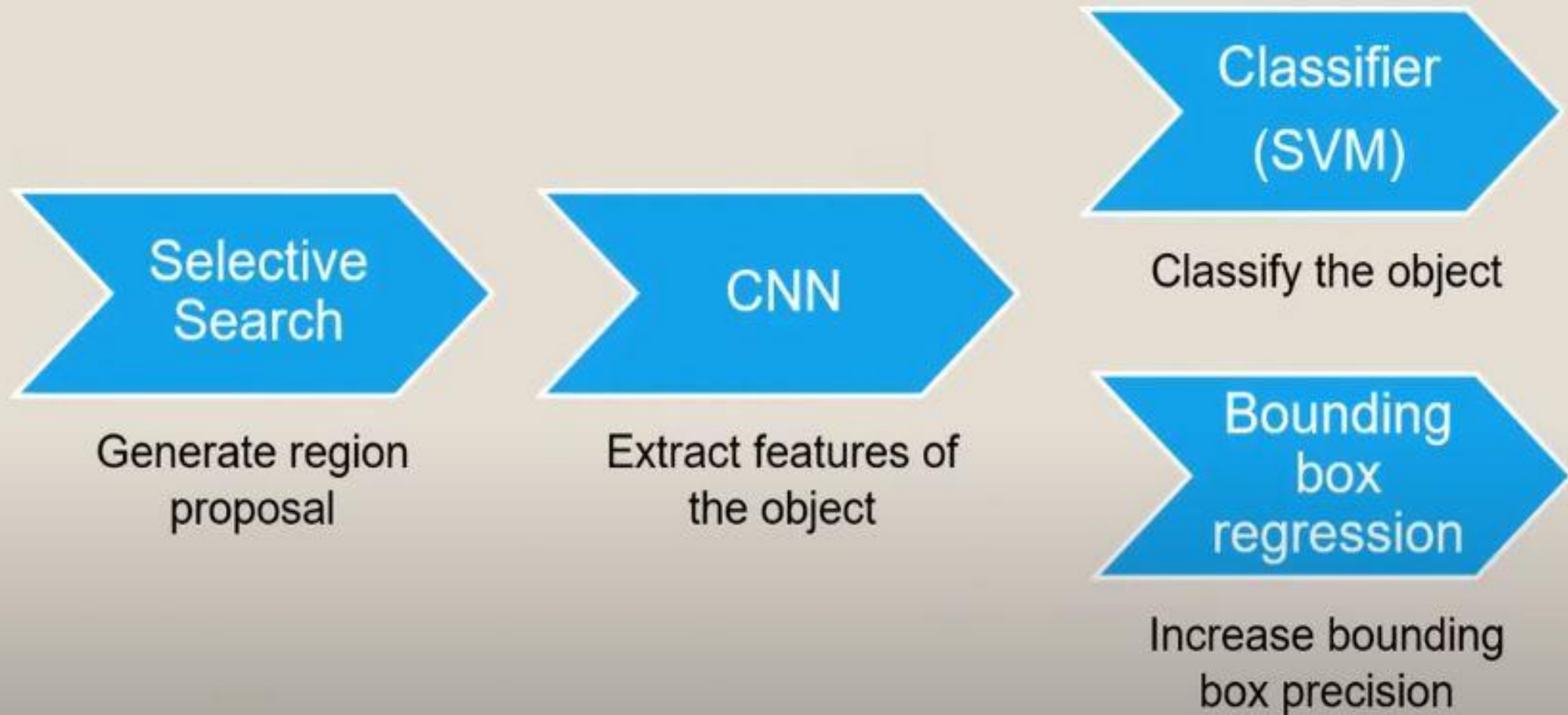


Over-segmented!



(Chandel, 2017)

R-CNN in a Nutshell...



Disadvantages

- Very Slow
- Cropping of proposed regions
- Requires to apply convolution on each proposed image
- Same object will be detected in multiple windows (with different Bounding Boxes) due to NOT-Optimized Proposed regions

Solution

- Save time by doing CNN one time only on the whole input image

Fast R-CNN

Fast R-CNN

Ross Girshick
Microsoft Research
rgg@microsoft.com

Abstract

This paper proposes a Fast Region-based Convolutional Network method (Fast R-CNN) for object detection. Fast R-CNN builds on previous work to efficiently classify object proposals using deep convolutional networks. Compared to previous work, Fast R-CNN employs several innovations to improve training and testing speed while also increasing detection accuracy. Fast R-CNN trains the very deep VGG16 network 9x faster than R-CNN, is 213x faster at test-time, and achieves a higher mAP on PASCAL VOC 2012. Compared to SPPnet, Fast R-CNN trains VGG16 3x faster, tests 10x faster, and is more accurate. Fast R-CNN is implemented in Python and C++ (using Caffe) and is available under the open-source MIT License at <https://github.com/rbgirshick/fast-rcnn>.

1. Introduction

Recently, deep ConvNets [14, 16] have significantly improved image classification [14] and object detection [9, 10] accuracy. Compared to image classification, object detection is a more challenging task that requires more complex methods to solve. Due to this complexity, current approaches (e.g., [9, 11, 19, 25]) train models in multi-stage pipelines that are slow and inelegant.

Complexity arises because detection requires the accurate localization of objects, creating two primary challenges. First, numerous candidate object locations (often called “proposals”) must be processed. Second, these candidates provide only rough localization that must be refined to achieve precise localization. Solutions to these problems often compromise speed, accuracy, or simplicity.

In this paper, we streamline the training process for state-of-the-art ConvNet-based object detectors [9, 11]. We propose a single-stage training algorithm that jointly learns to classify object proposals and refine their spatial locations.

The resulting method can train a very deep detection network (VGG16 [20]) 9x faster than R-CNN [9] and 3x faster than SPPnet [11]. At runtime, the detection network processes images in 0.3s (excluding object proposal time)

while achieving top accuracy on PASCAL VOC 2012 [7] with a mAP of 66% (vs. 62% for R-CNN).¹

1.1. R-CNN and SPPnet

The Region-based Convolutional Network method (R-CNN) [9] achieves excellent object detection accuracy by using a deep ConvNet to classify object proposals. R-CNN, however, has notable drawbacks:

1. **Training is a multi-stage pipeline.** R-CNN first fine-tunes a ConvNet on object proposals using log loss. Then, it fits SVMs to ConvNet features. These SVMs act as object detectors, replacing the softmax classifier learnt by fine-tuning. In the third training stage, bounding-box regressors are learned.
2. **Training is expensive in space and time.** For SVM and bounding-box regressor training, features are extracted from each object proposal in each image and written to disk. With very deep networks, such as VGG16, this process takes 2.5 GPU-days for the 5k images of the VOC07 trainval set. These features require hundreds of gigabytes of storage.
3. **Object detection is slow.** At test-time, features are extracted from each object proposal in each test image. Detection with VGG16 takes 47s / image (on a GPU).

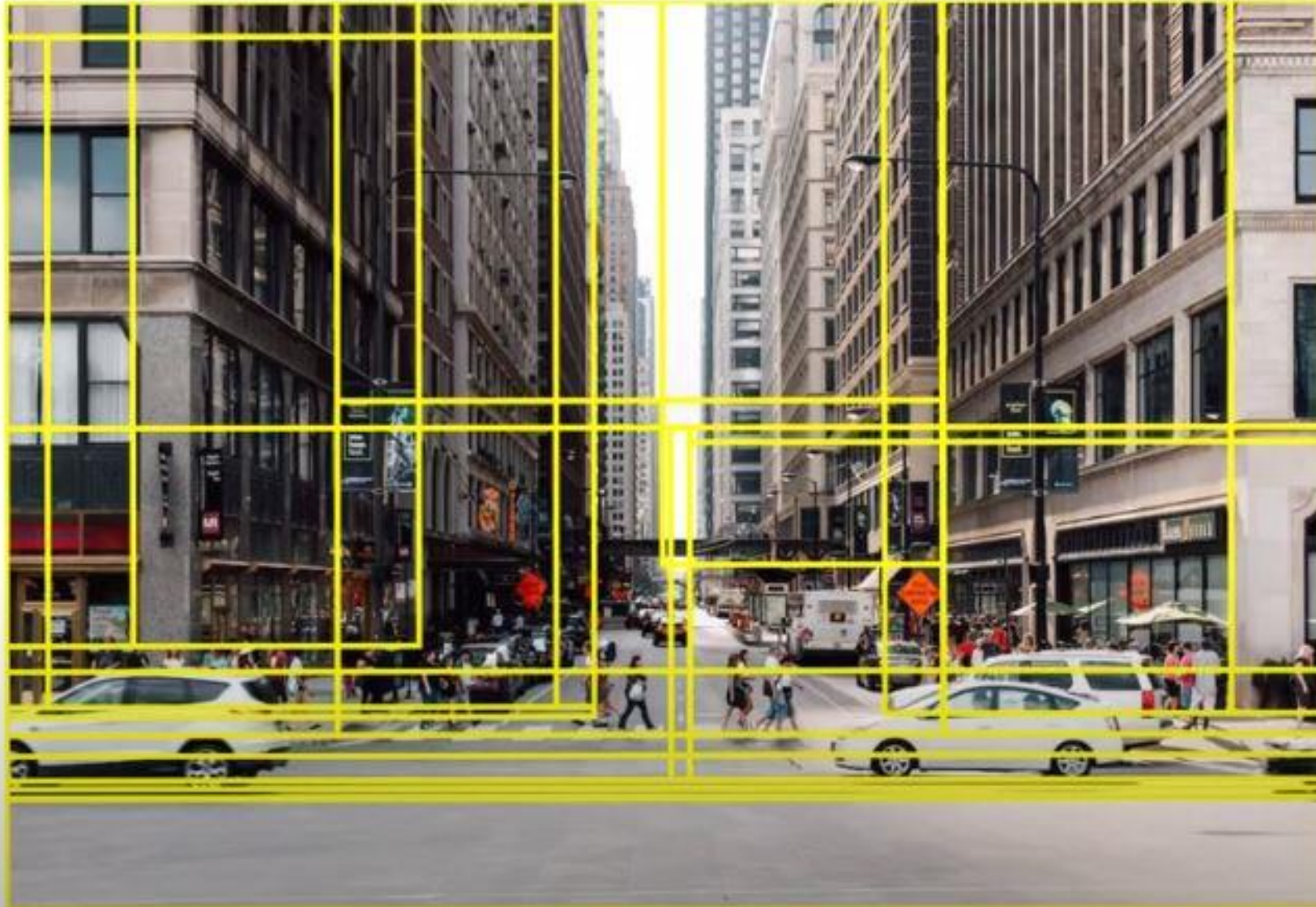
R-CNN is slow because it performs a ConvNet forward pass for each object proposal, without sharing computation. Spatial pyramid pooling networks (SPPnets) [11] were proposed to speed up R-CNN by sharing computation. The SPPnet method computes a convolutional feature map for the entire input image and then classifies each object proposal using a feature vector extracted from the shared feature map. Features are extracted for a proposal by max-pooling the portion of the feature map inside the proposal into a fixed-size output (e.g., 6×6). Multiple output sizes are pooled and then concatenated as in spatial pyramid pooling [15]. SPPnet accelerates R-CNN by 10 to 100x at test time. Training time is also reduced by 3x due to faster proposal feature extraction.

¹ All timings use one Nvidia K40 GPU overclocked to 875 MHz.

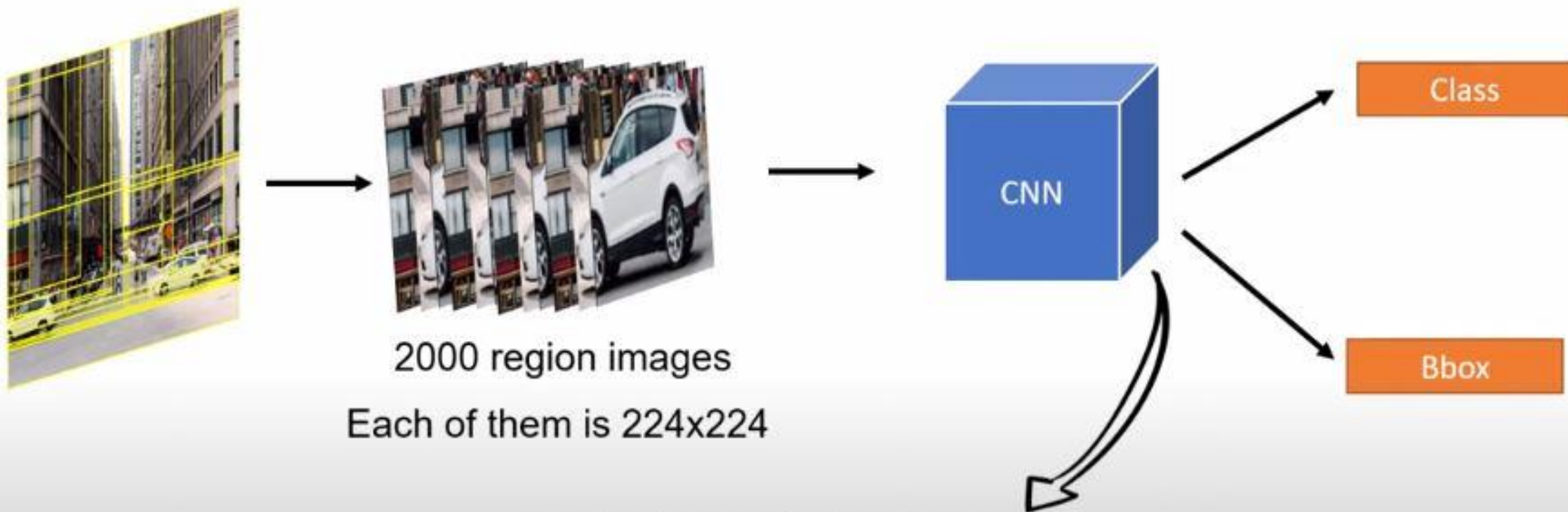
Last time: R-CNN



Last time: R-CNN



Last time: R-CNN



It is just extracting features

Can't we do this at first and
only one time per image?

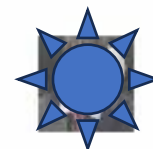
RoI Pooling

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

$$h = 4 \quad w = 5$$

$$H = 2 \quad W = 2$$

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

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

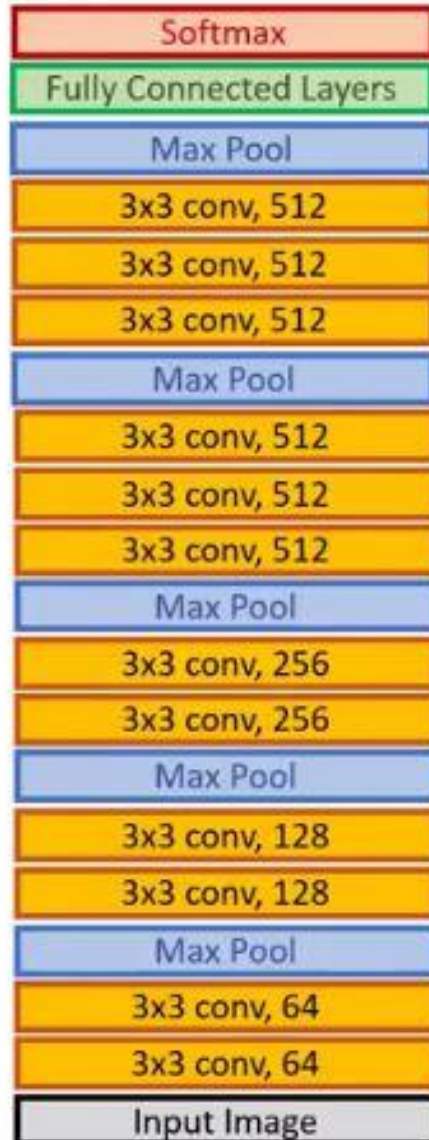
RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard 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].

60	91
96	85



Initializing from pre-trained networks

it undergoes three transformations.

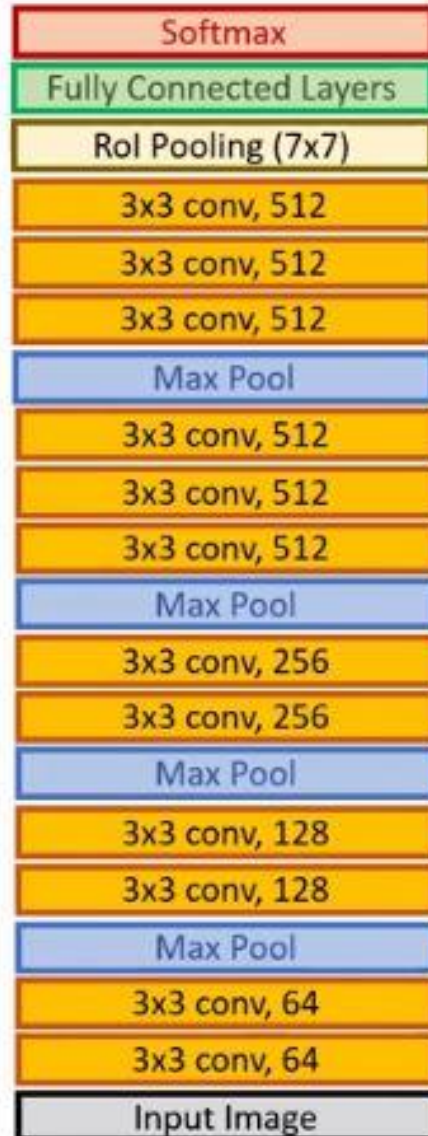


First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (*e.g.*, $H = W = 7$ for VGG16).



Initializing from pre-trained networks

it undergoes three transformations.

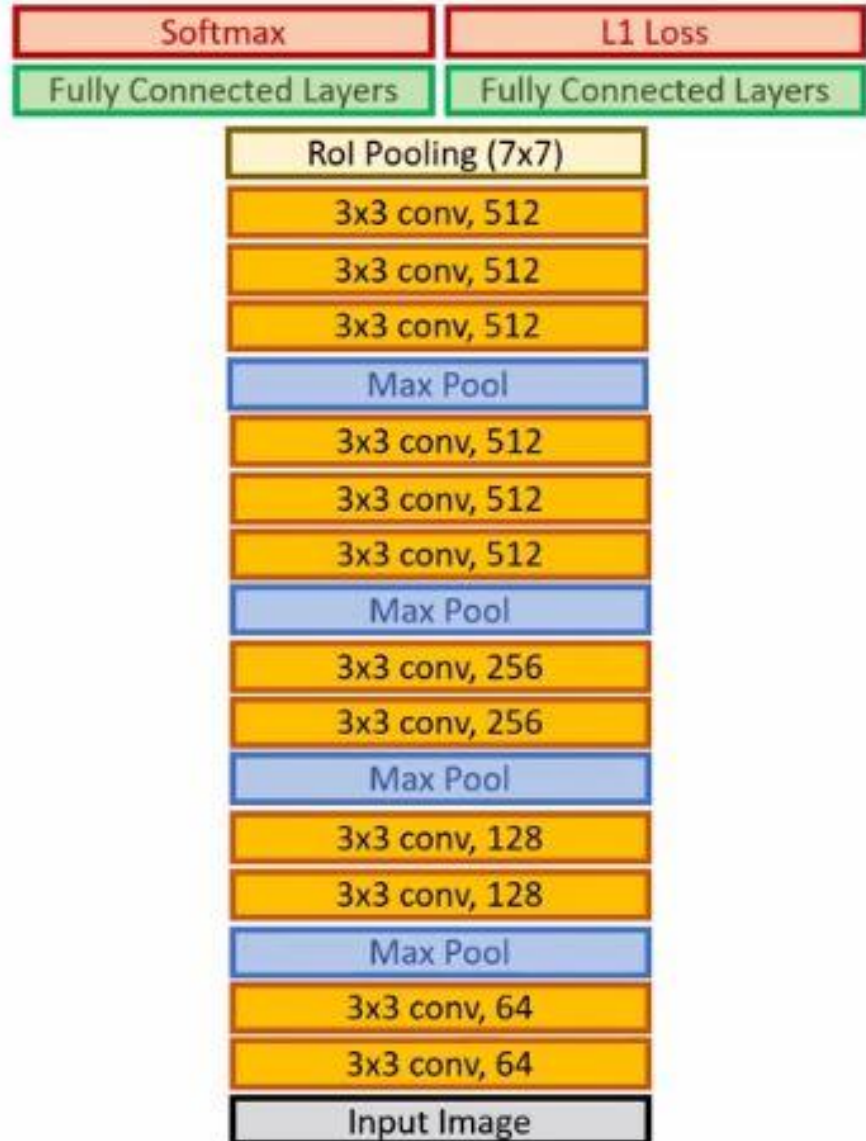


First, the last max pooling layer is replaced by a Rol pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (*e.g.*, $H = W = 7$ for VGG16).



Initializing from pre-trained networks

it undergoes three transformations.



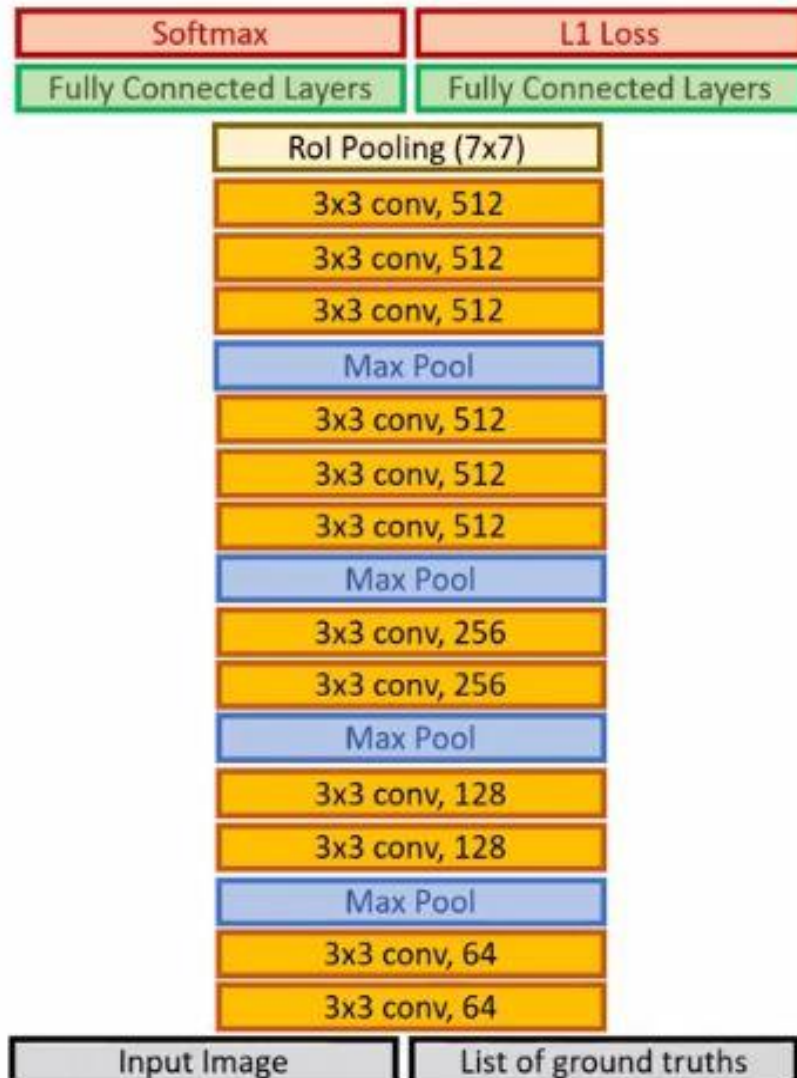
First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (*e.g.*, $H = W = 7$ for VGG16).

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

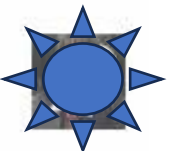
it undergoes three transformations.



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.



Fine-tuning for detection

Mini-batch sampling



Mini-batch sampling

Mini-batch 1



Mini-batch 2



Mini-batch 3



Mini-batch 4



Mini-batch sampling

Mini-batch 2



Mini-batch 3



Mini-batch 4



Mini-batch sampling

Mini-batch 2



Mini-batch 3



Mini-batch 4

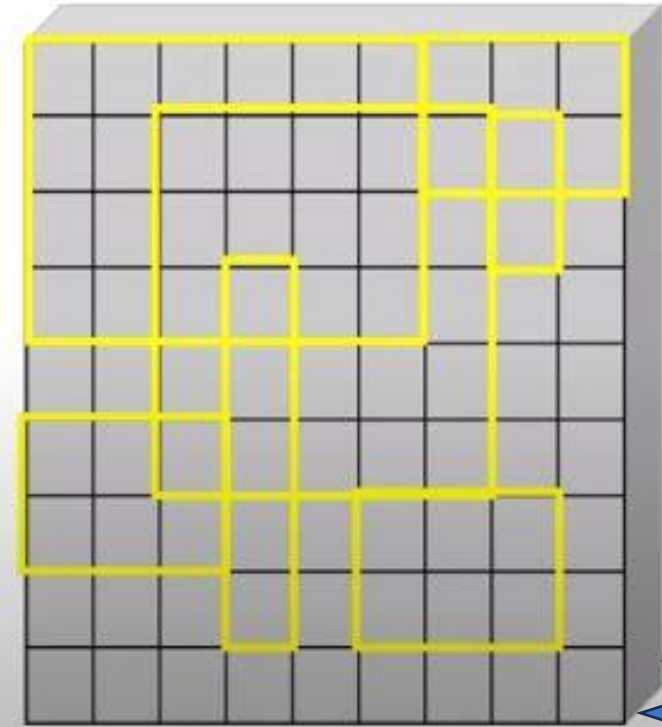
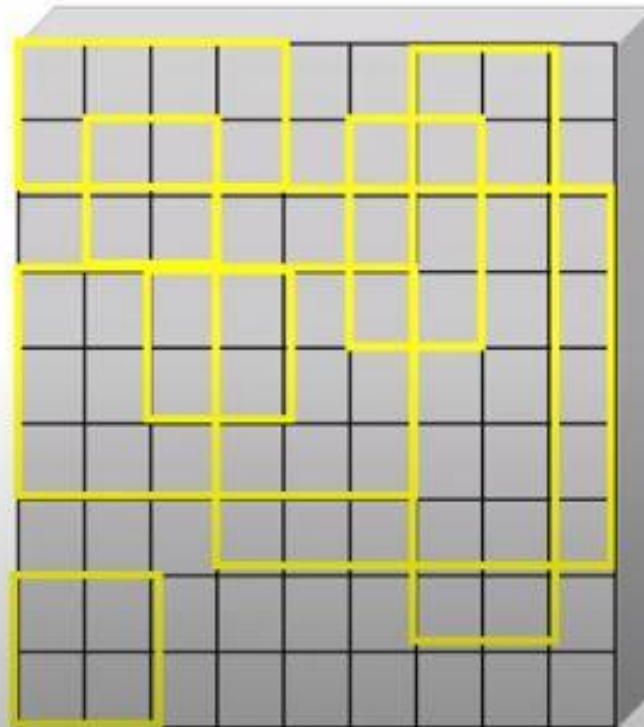


It only selects 64 RoIs from each image!

How???

we take 25% of the RoIs from object proposals that have intersection over union (IoU) overlap with a ground-truth bounding box of at least 0.5.

The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval $[0.1, 0.5)$



Multi-task Loss

Two sibling output layers:

1

$$p = (p_0, \dots, p_K)$$

over $K+1$ categories

Object classes

Background

2

$$t^k = (t_x^k, t_y^k, t_w^k, t_h^k)$$

Ground truths:

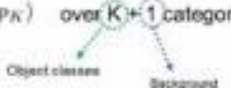
u

v

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v),$$

Multi-task Loss

Two sibling output layers:

1 $p = (p_0, \dots, p_K)$ over $K+1$ categories


2 $t^k = (t_x^k, t_y^k, t_w^k, t_h^k)$

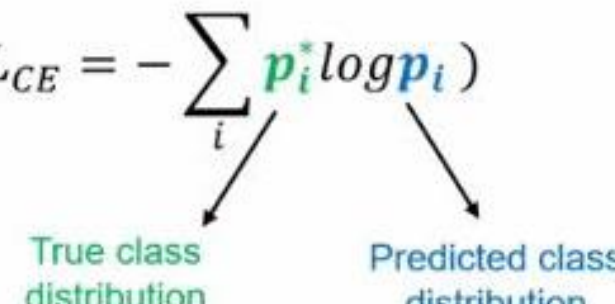
Ground truths:

u

v

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v),$$

● $L_{\text{cls}}(p, u) = -\log p_u$ $(L_{\text{CE}} = -\sum_i \text{True class distribution} \log \text{Predicted class distribution})$



● $[u \geq 1]$ evaluates to 1 when $u \geq 1$ and 0 otherwise.

● $L_{\text{loc}}(t^u, v) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L_1}(t_i^u - v_i),$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$



Scale Invariance

Scale Invariance

Brute force

Image pyramids



Fast R-CNN



output