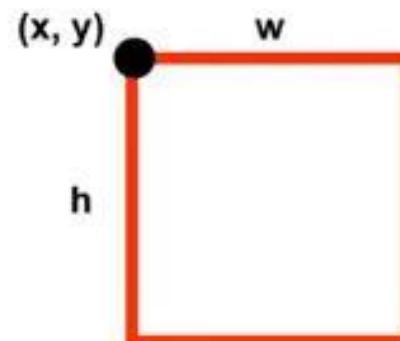


R-CNN



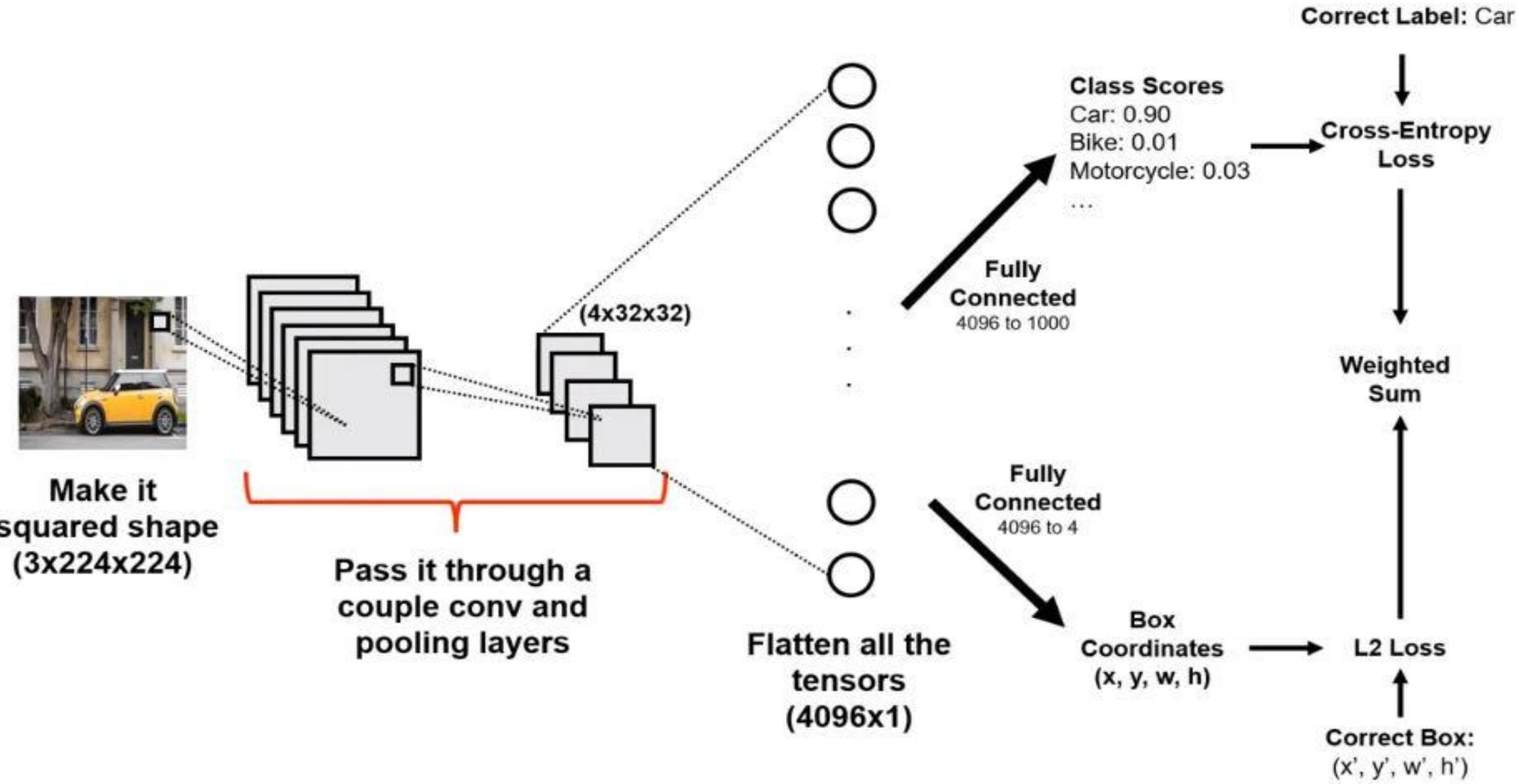
Our Goal



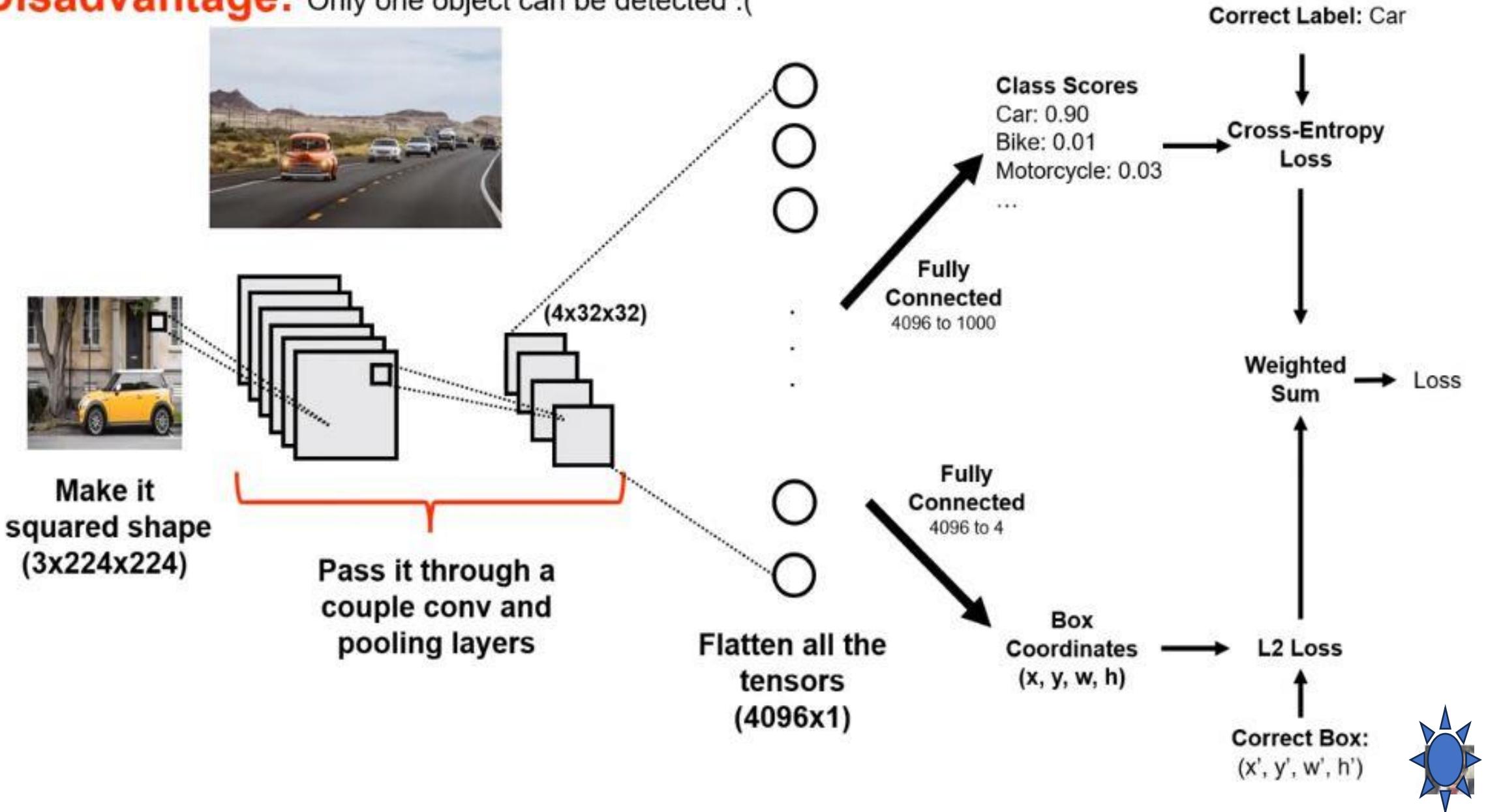
Bounding box

We receive input image





Disadvantage: Only one object can be detected :(



Earliest Approach

Earliest Approach



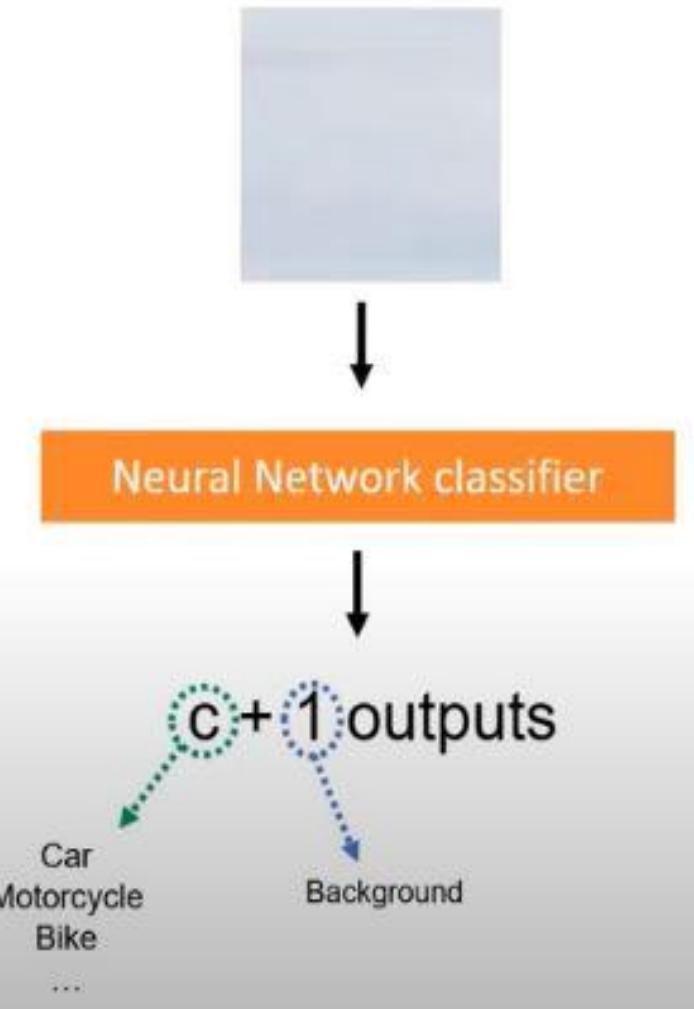
Sliding Window



Earliest Approach



Classify this region!



Earliest Approach



Classify this region!



Neural Network classifier



Mountain

Earliest Approach



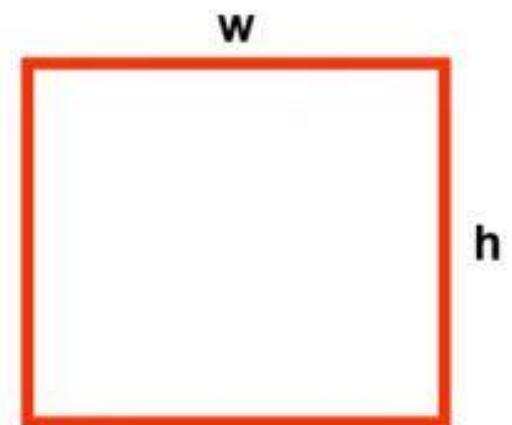
Classify this region!



Neural Network classifier

Car

Earliest Approach



Possible Positions:

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

CNN as feature extractor

- › What could be the problems?
 - Suppose we have a 600×600 image, if sliding window size is 20×20 , 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

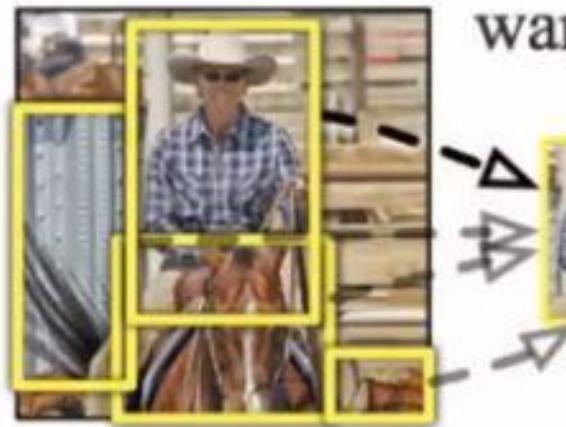
- Very Slow
- Number of Picked windows is very huge
- For CNN classifier, needs to apply convolution to each window content
- Same object will be detected in multiple windows (with different Bounding Boxes)

R-CNN

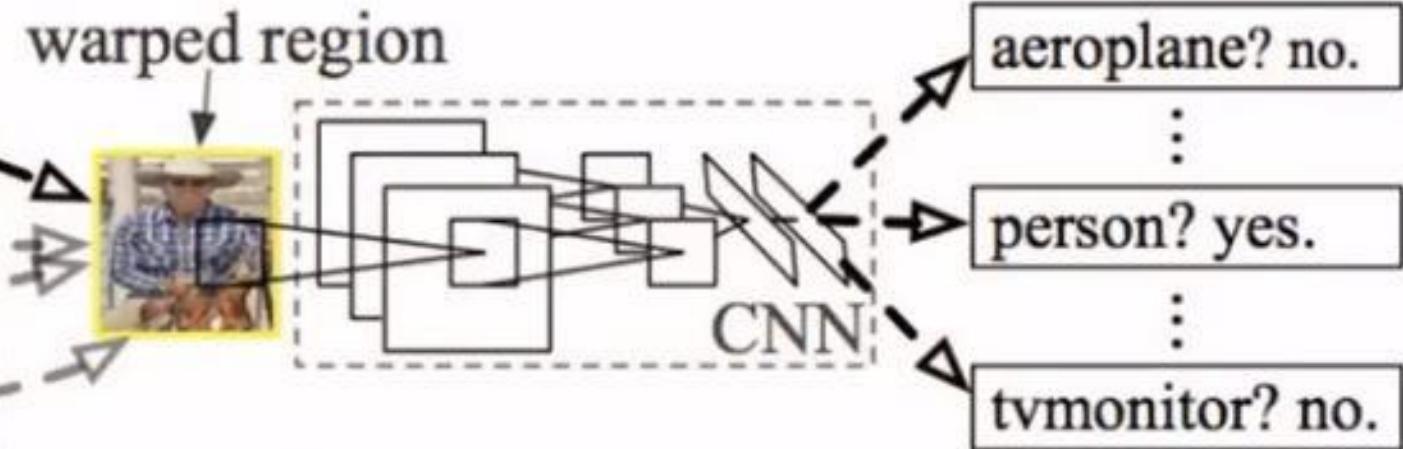
R-CNN: *Regions with CNN features*



1. Input
image



2. Extract region
proposals (~2k)



3. Compute
CNN features

4. Classify
regions

(Girshick, et al., 2014)

Bounding Box Regression Training

$(x_1, y_1) = (200, 250)$
 $(x_2, y_2) = (600, 400)$

800



600

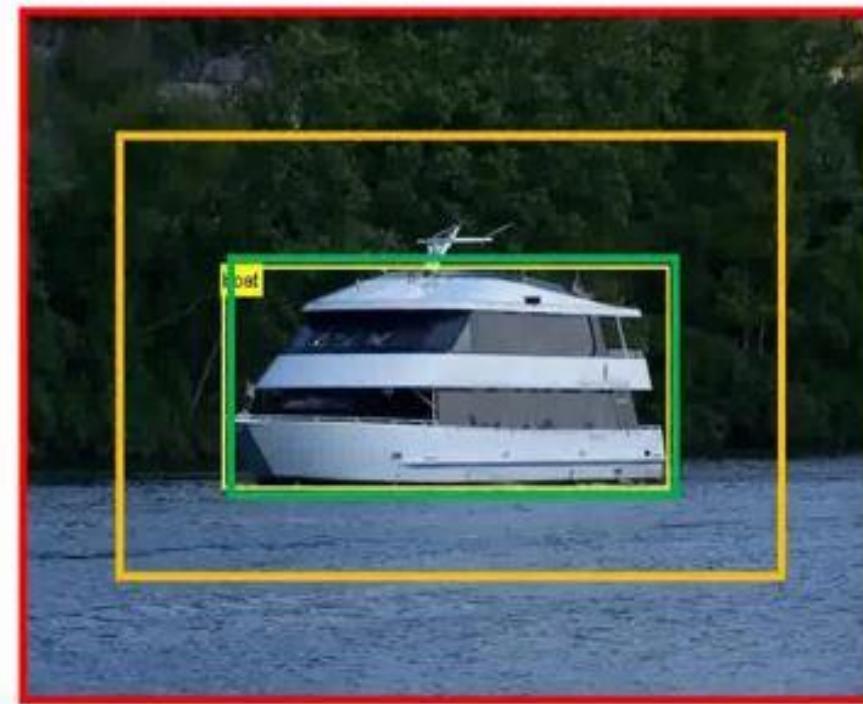
boot

250

200

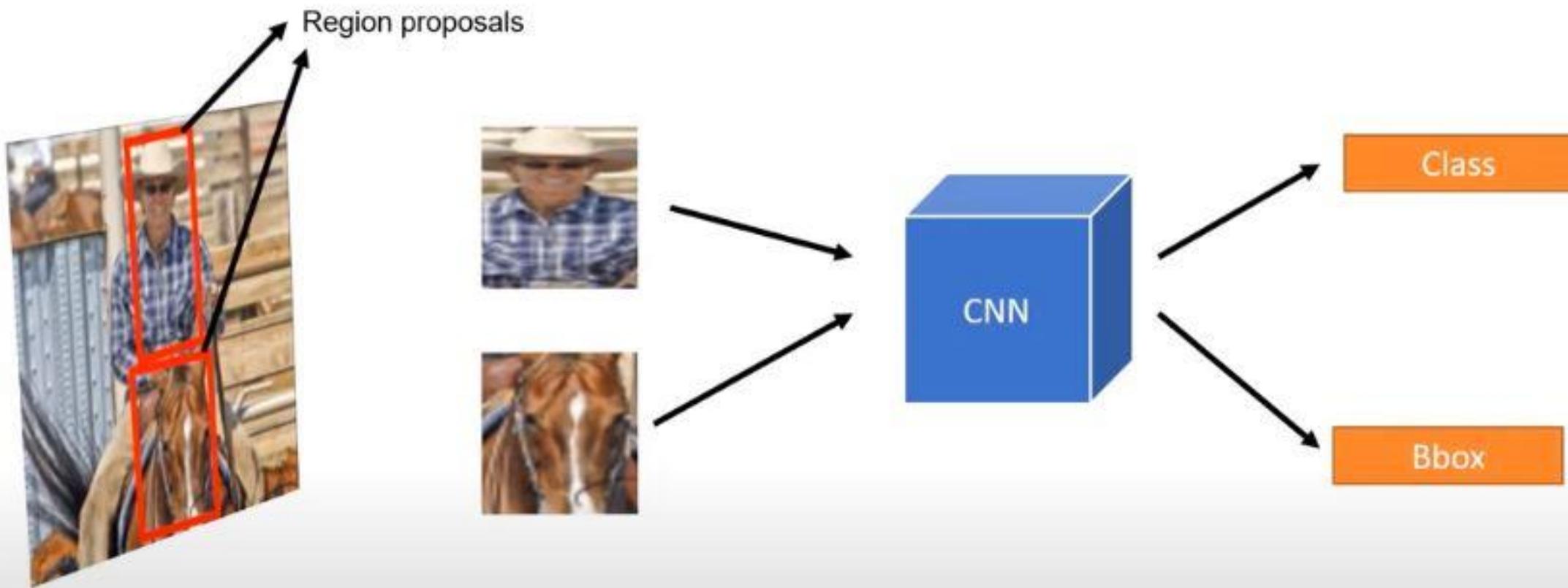
400

600



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

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

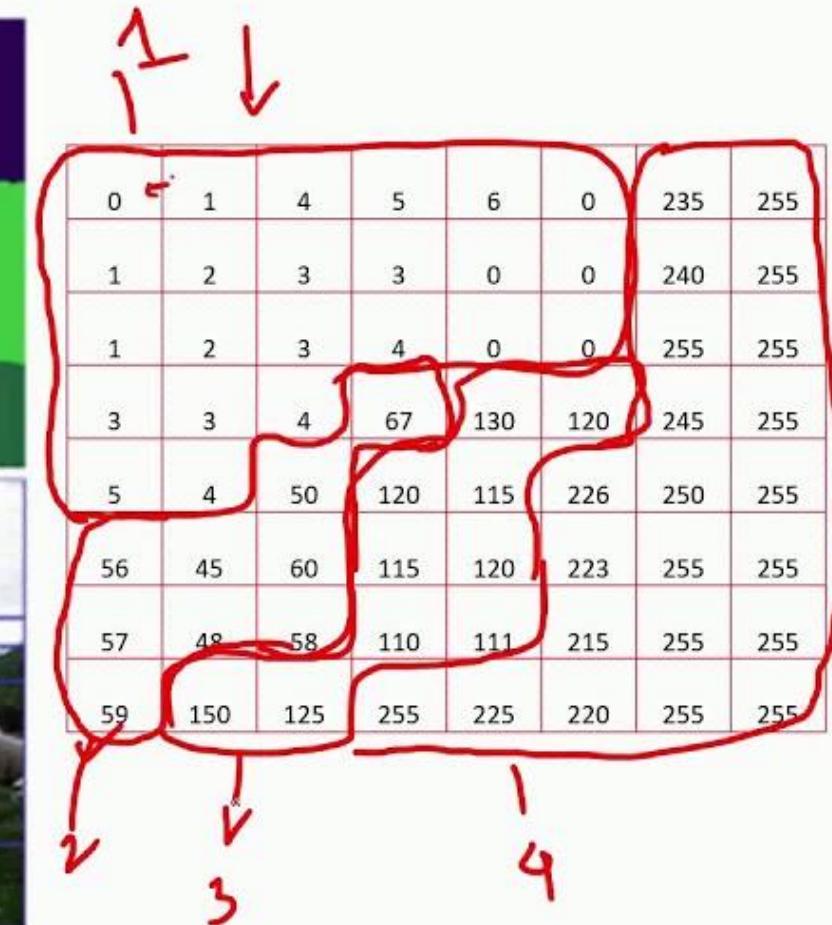
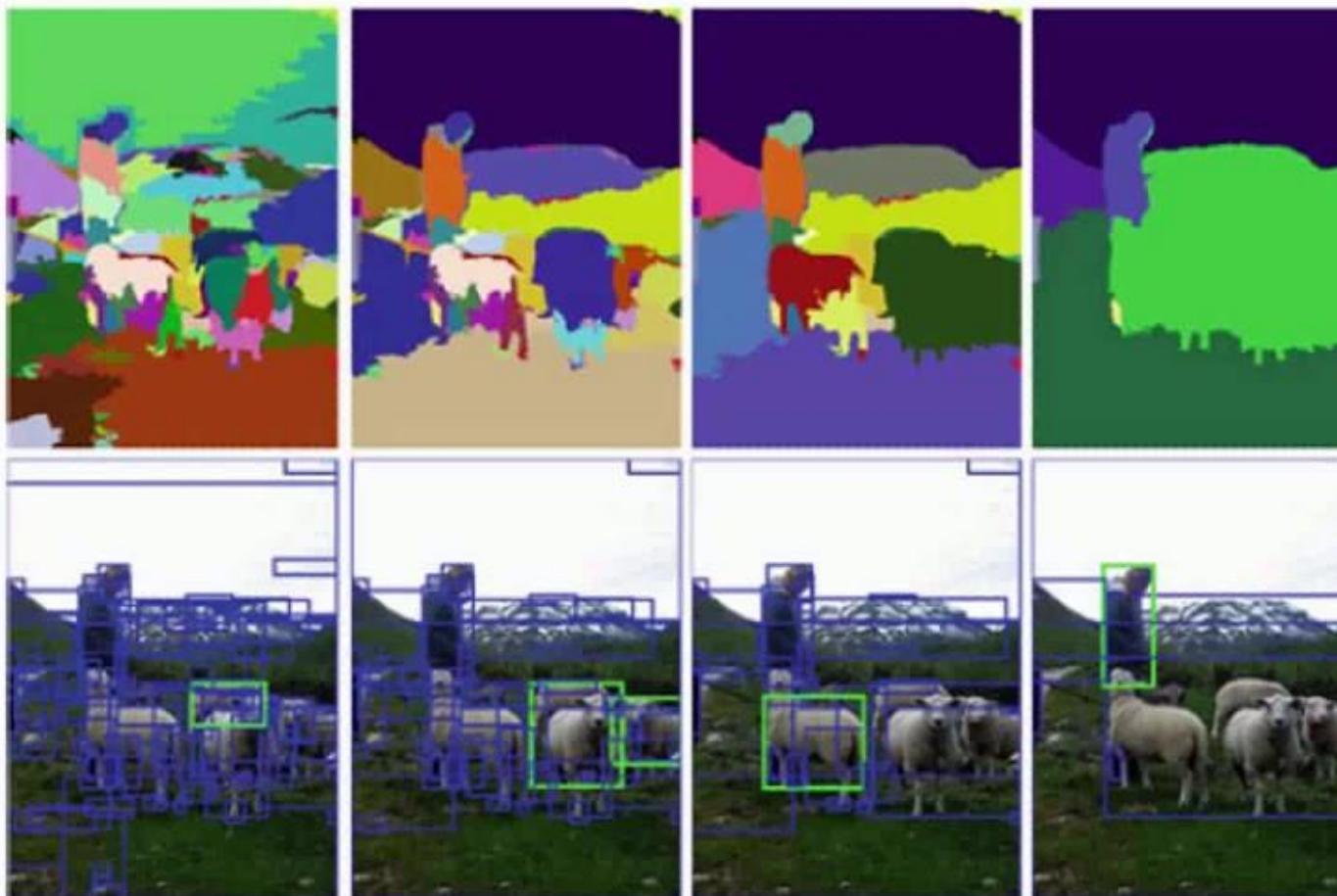


Output Image

(Chadel, 2017)

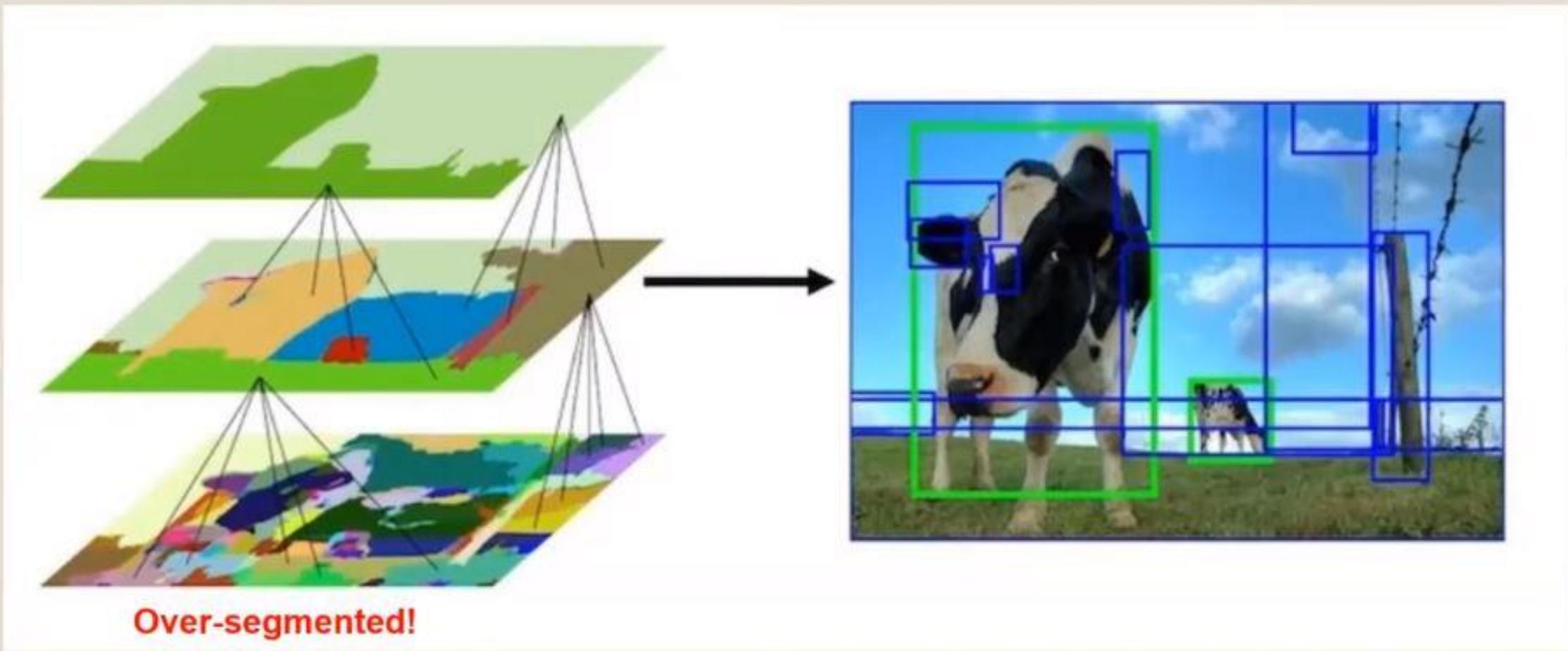
- We cannot directly use the segmented image as region proposals!
- Group adjacent segments by similarity.

Region Proposal Techniques



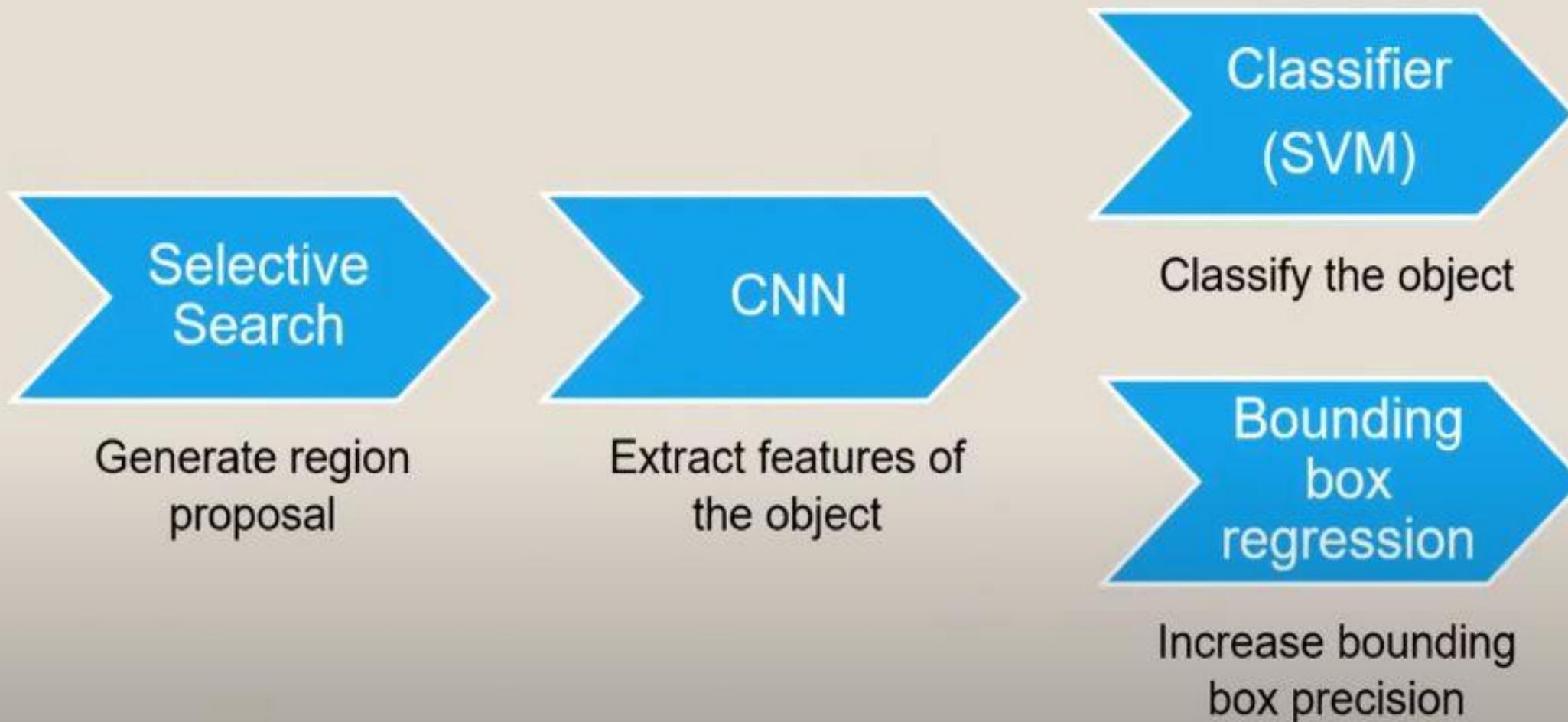
Superpixels Straddling ✓

Selective Search (simplified)



(Chanel, 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
rgb@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, 18] 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, 29]) 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 [28]) 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 [3] 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).

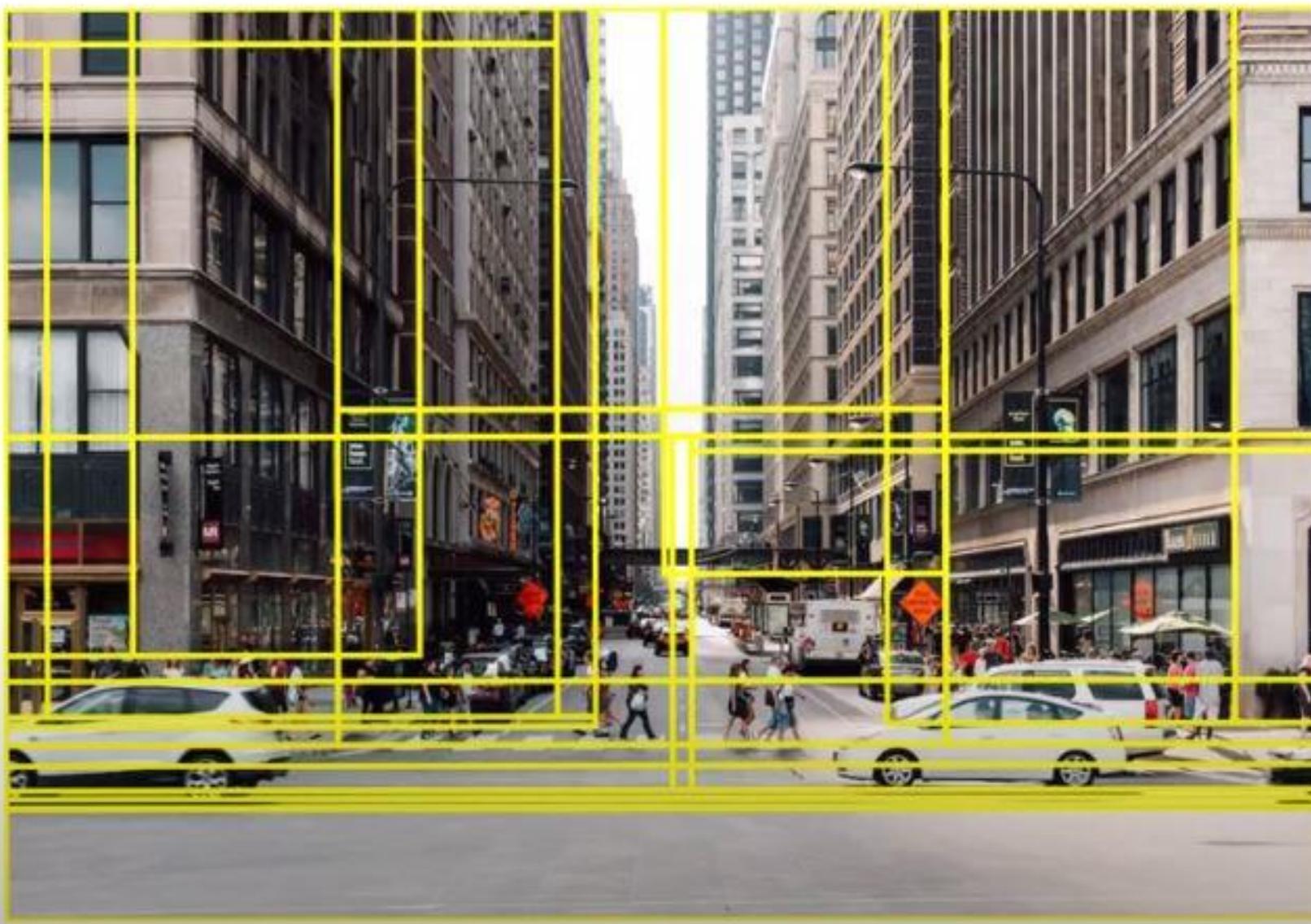
R-CNN is slow because it performs a ConvNet forward pass for each object proposal, without sharing computation. Spatial pyramid pooling networks (SPPnets) [11] were proposed to speed up R-CNN by sharing computation. The SPPnet method computes a convolutional feature map for the entire input image and 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 [13]. SPPnet accelerates R-CNN by 10 to 100x at test-time. Training time is also reduced by 3x due to faster proposal feature extraction.

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

Last time: R-CNN



Last time: R-CNN



Last time: R-CNN



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

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

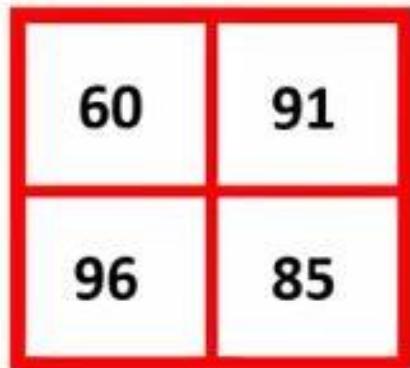
$$H = 2 \quad W = 2$$

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

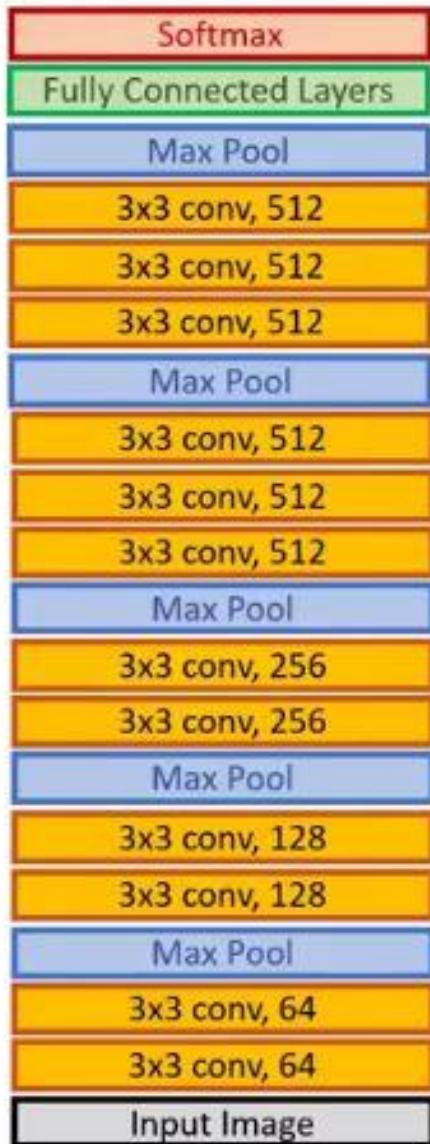


RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard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



Initializing from pre-trained networks

it undergoes three transformations.

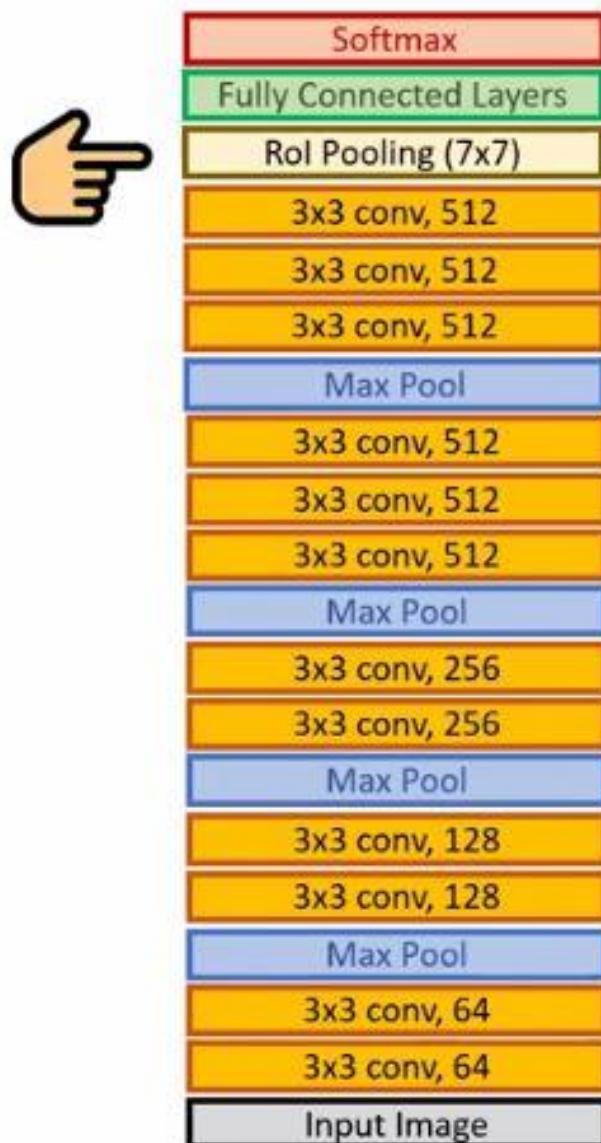


First, the last max pooling layer is replaced by a ROI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (*e.g.*, $H = W = 7$ for VGG16).



Initializing from pre-trained networks

it undergoes three transformations.

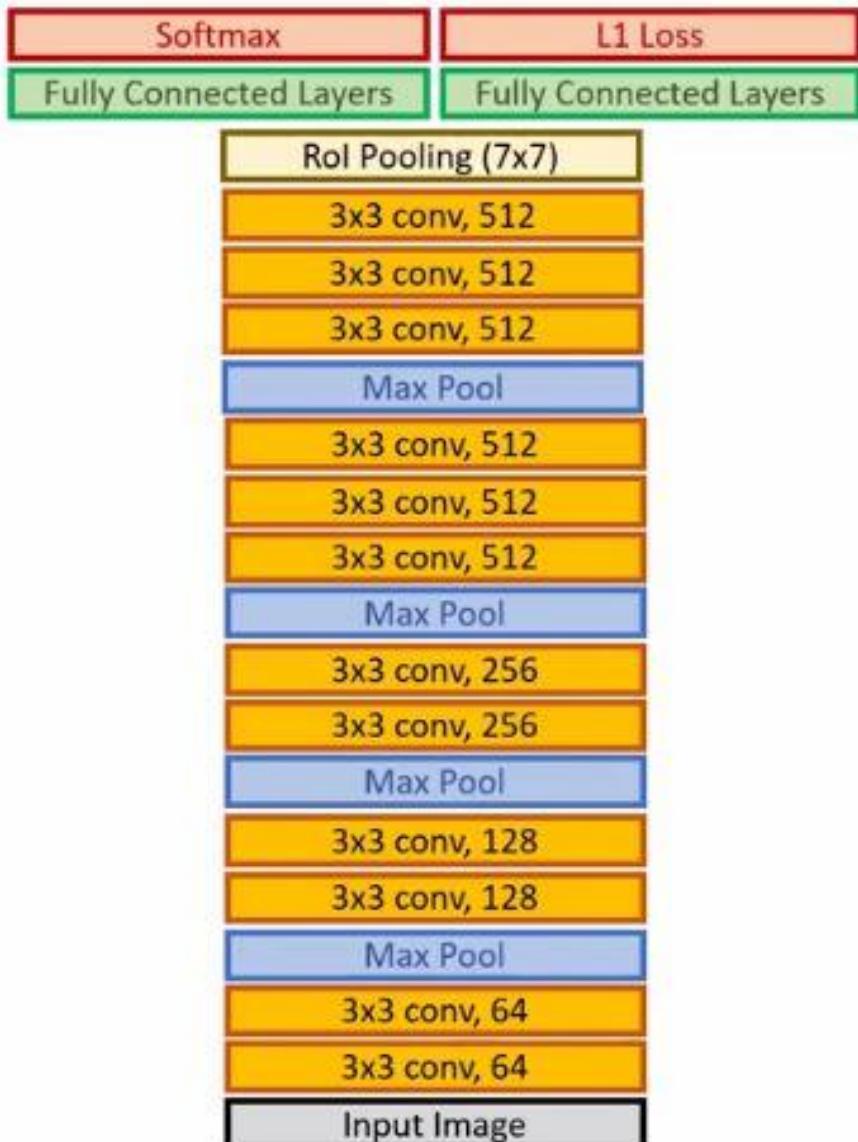


First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (e.g., $H = W = 7$ for VGG16).



Initializing from pre-trained networks

it undergoes three transformations.



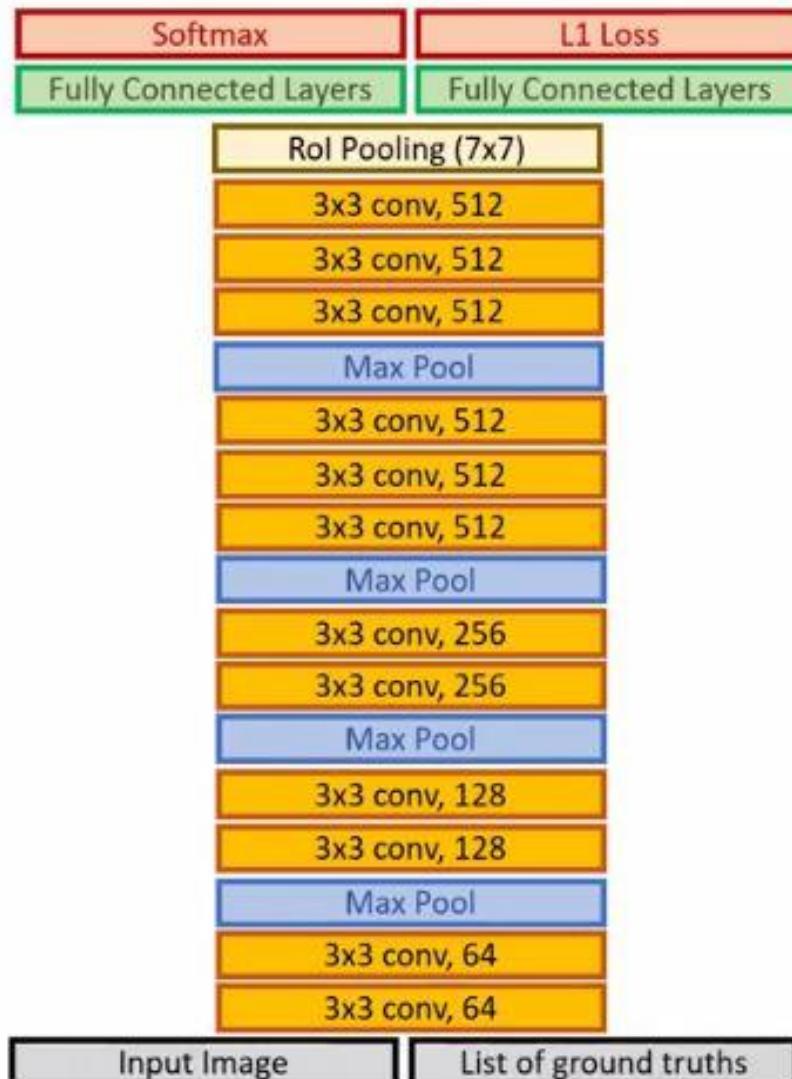
First, the last max pooling layer is replaced by a ROI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (e.g., $H = W = 7$ for VGG16).

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.



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

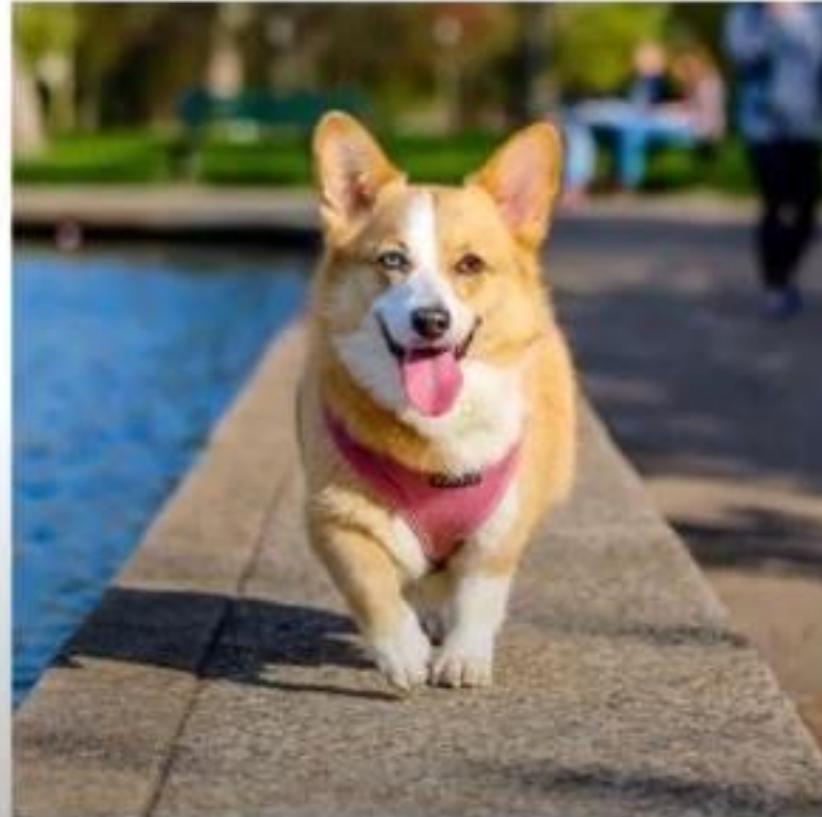
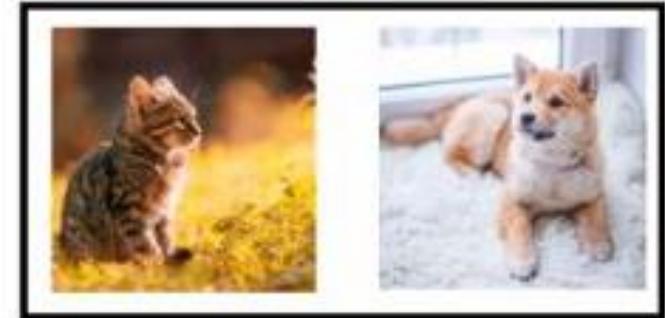
Mini-batch 2



Mini-batch 3



Mini-batch 4



Mini-batch sampling

Mini-batch 2



Mini-batch 3



Mini-batch 4

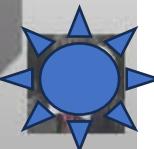
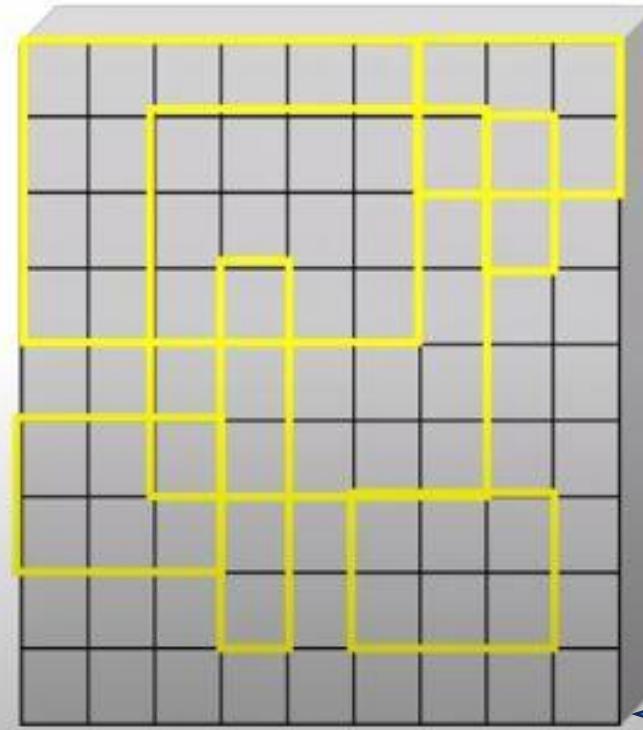
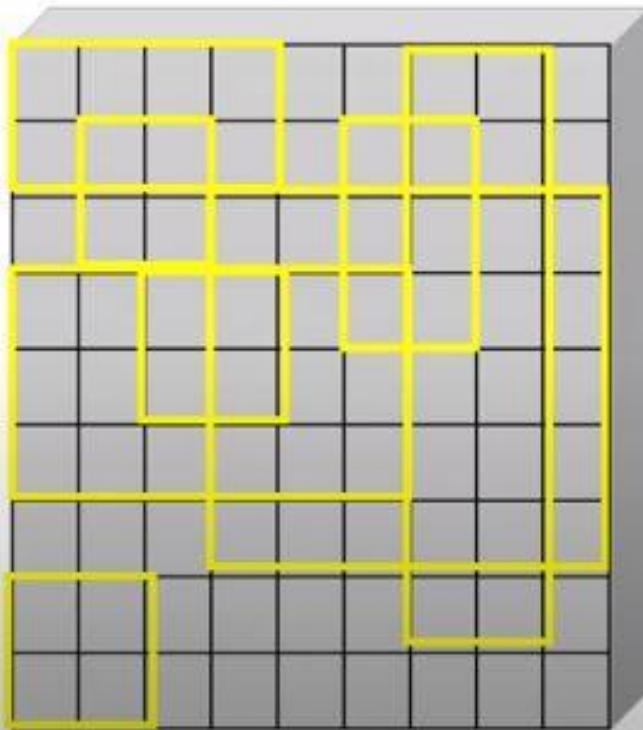


It only selects 64 RoIs from each image!

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

Ground truths:

u

2

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

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

Ground truths:

u

v

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

● $L_{\text{cls}}(p, u) = -\log p_u$

$$(L_{CE} = -\sum_i p_i^* \log p_i)$$

True class distribution

Predicted class distribution

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

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

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

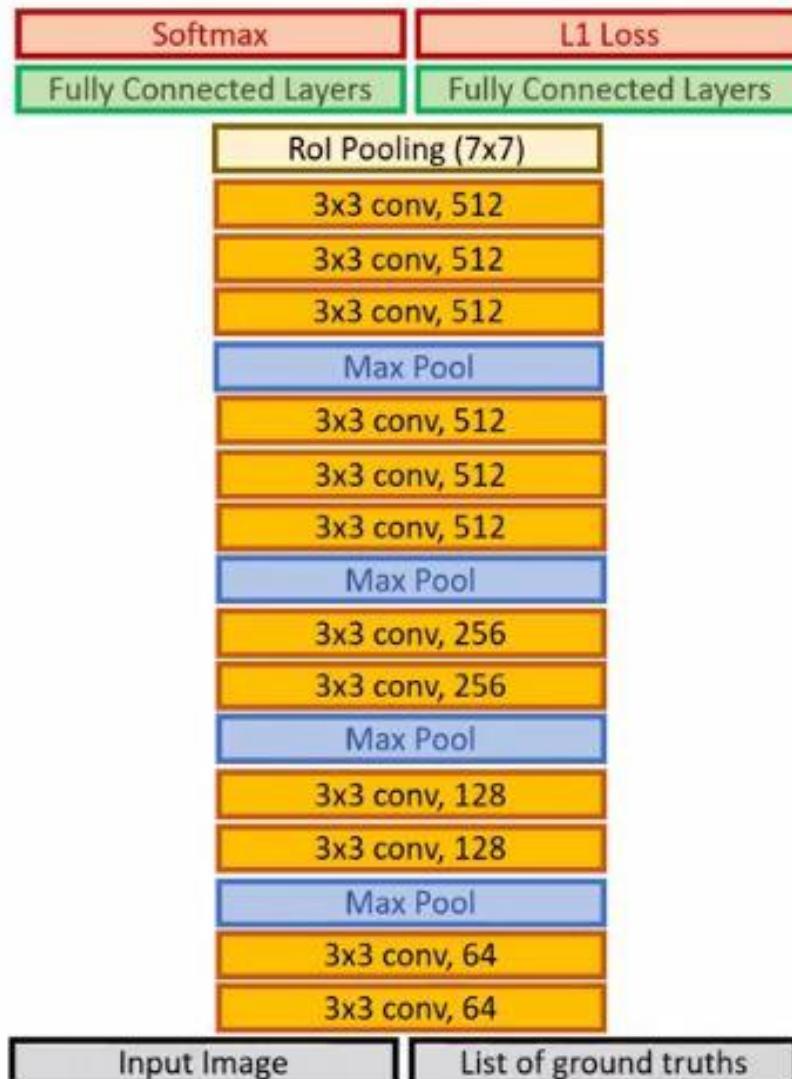
output

- Faster RCNN



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).

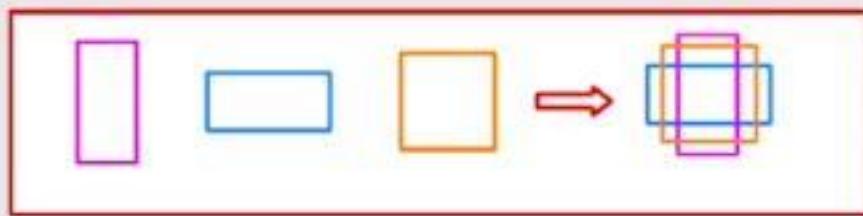
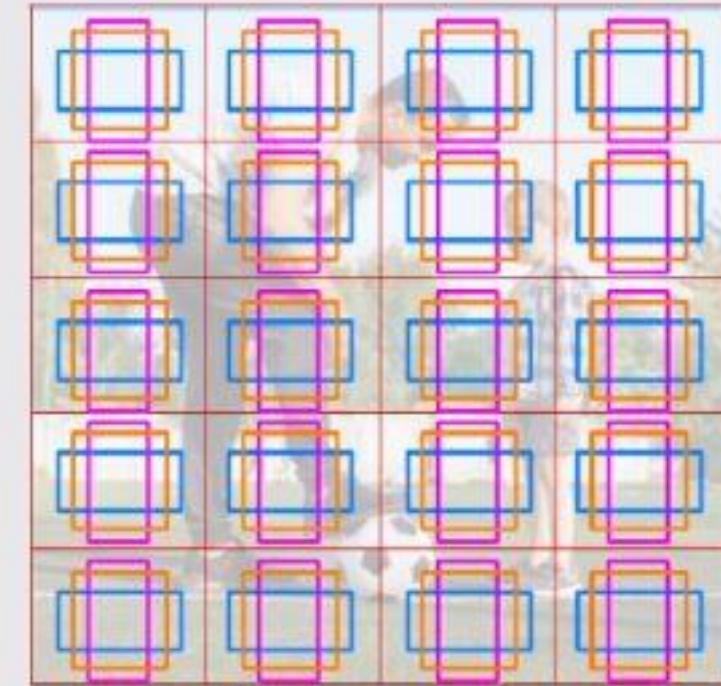
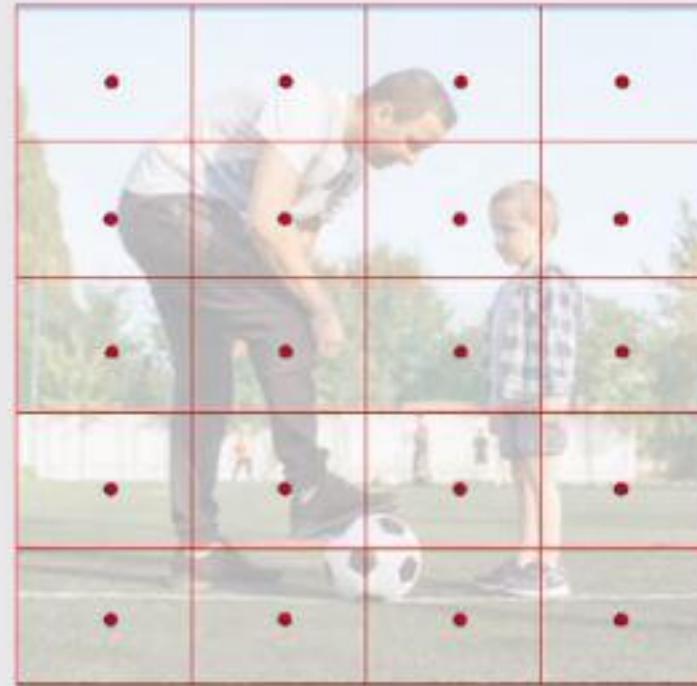
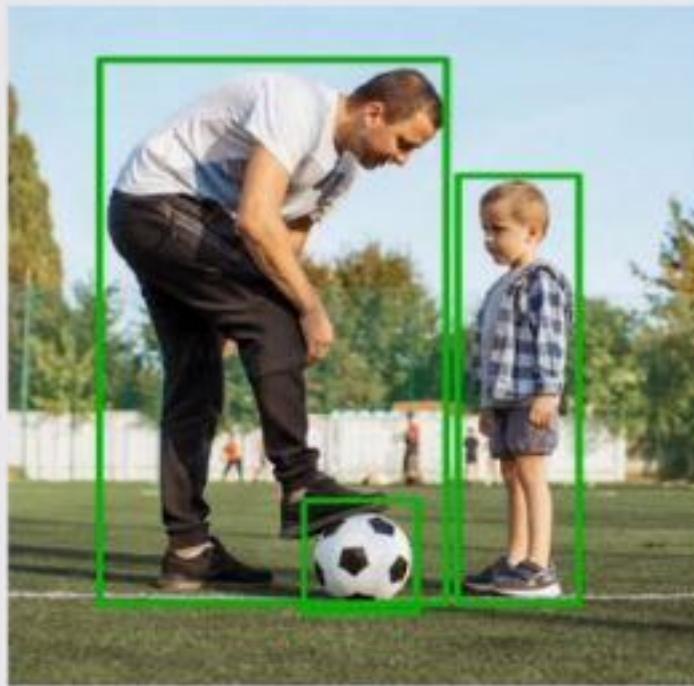
Second, the network's last fully connected layer and softmax (which were trained for 1000-way ImageNet classification) are replaced with the two sibling layers described earlier (a fully connected layer and softmax over $K + 1$ categories and category-specific bounding-box regressors).

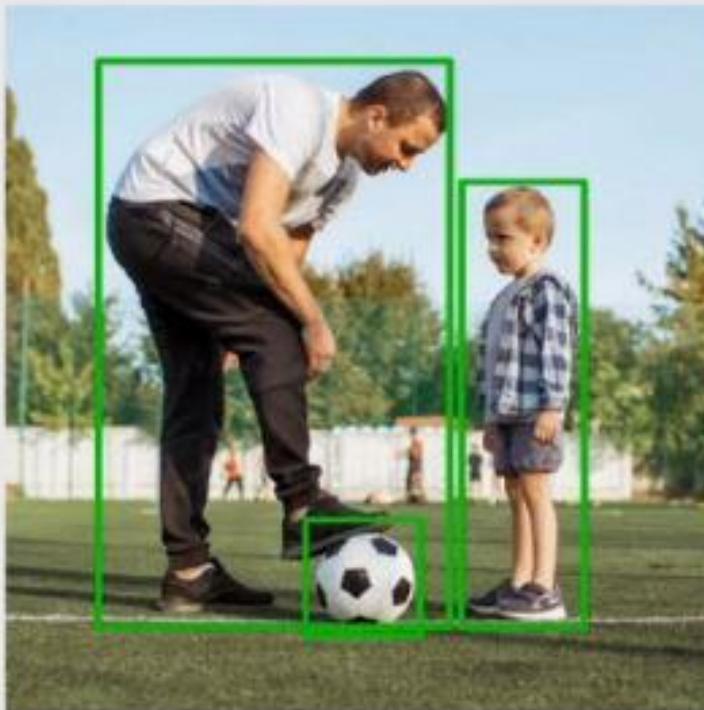
Third, the network is modified to take two data inputs: a list of images and a list of RoIs in those images.



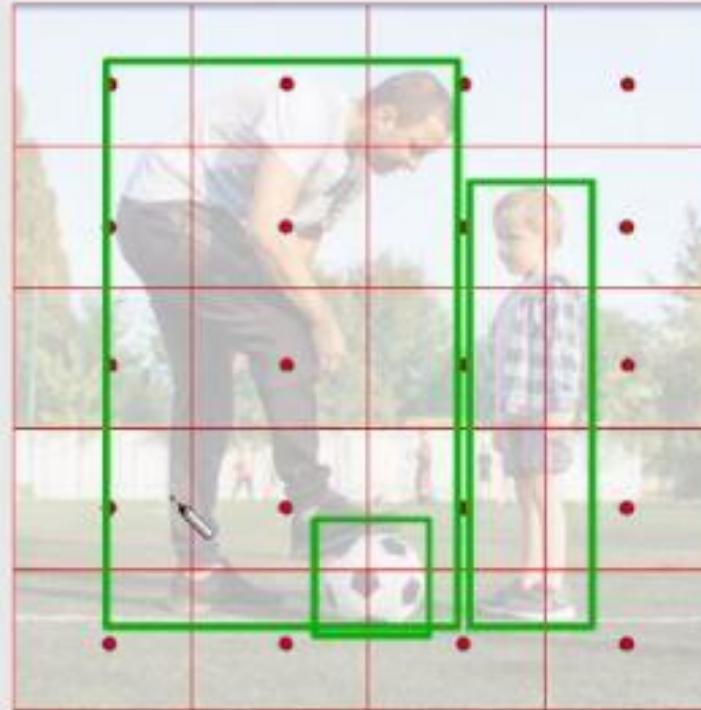
Region Proposal Network

RPN

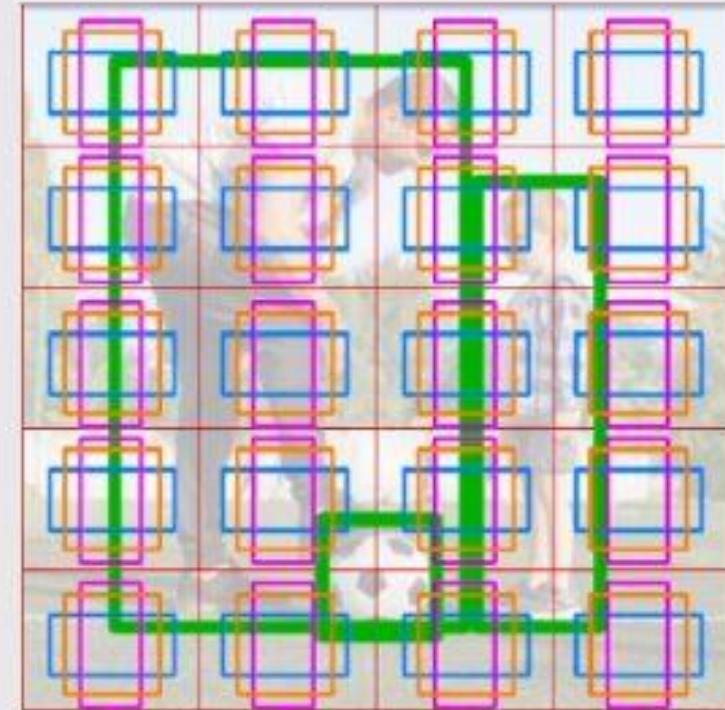




Input Image

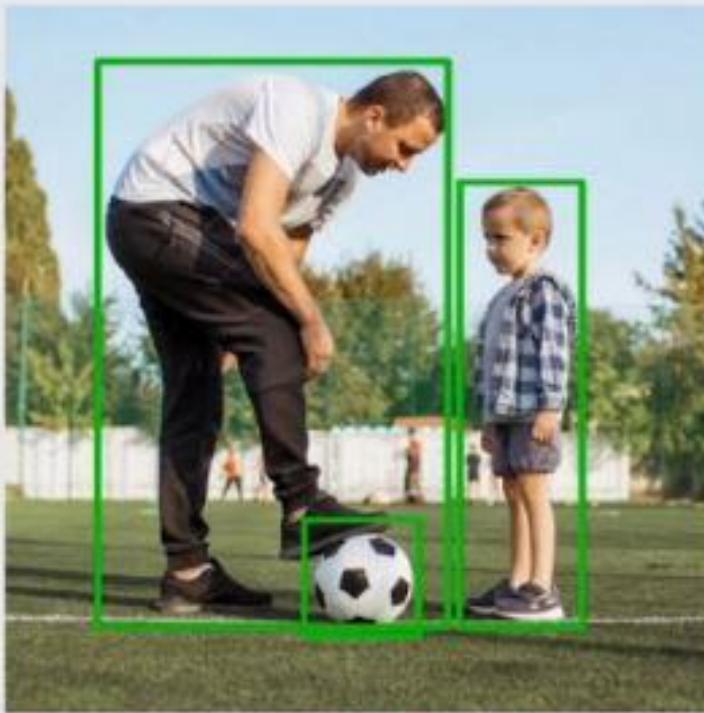


Define Anchor Points

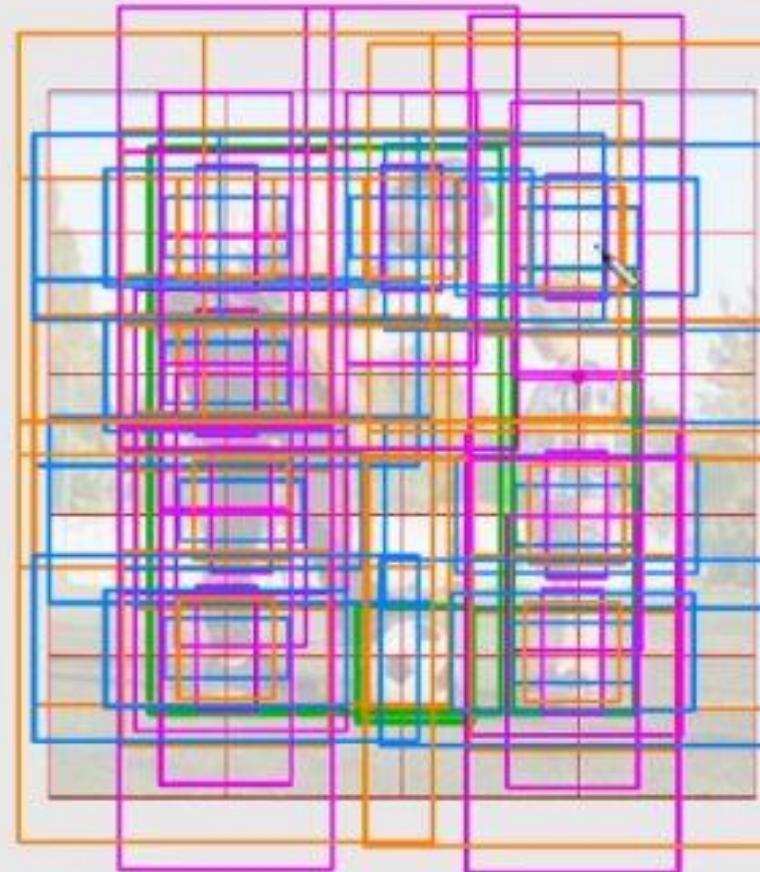


*Define Anchor Boxes
(centered at Anchor Points)*

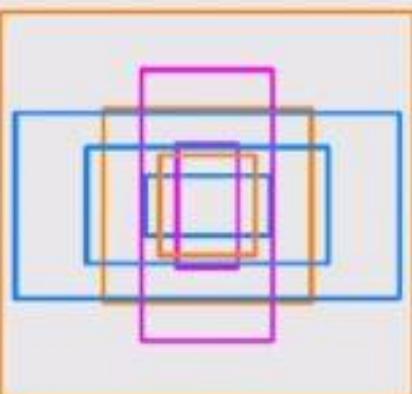
- Selected sizes of Anchor boxes are too small (with respect to objects)
- Only Ball may be detected by anchor boxes
- Other anchor boxes will be assumed as background (IOU with ground truth is small)



*Input Image
with Ground Truth*

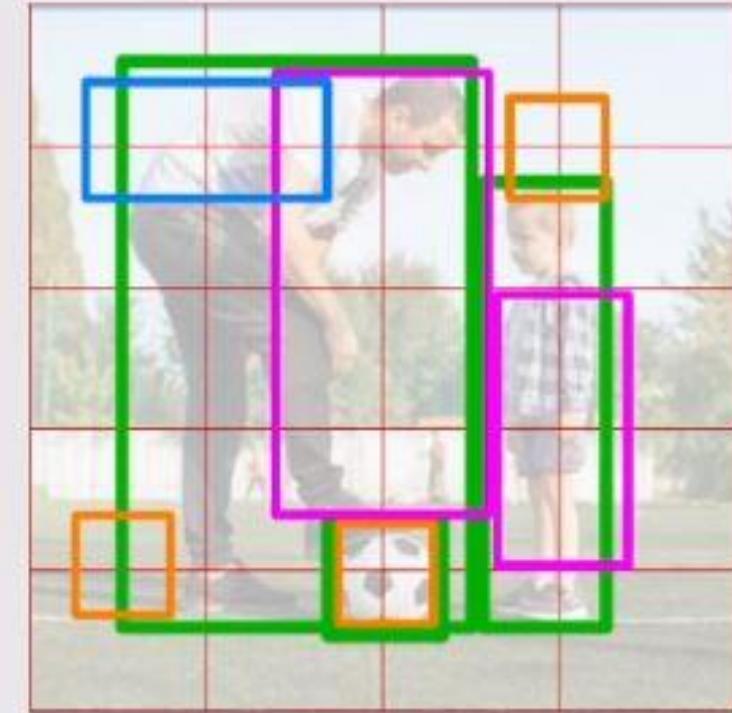


Define Anchor Points



*Define Larger Anchor Boxes
3 scales, 3 ratios (9 anchor boxes)*

Note: Many Overlaps between boxes (No Problem)

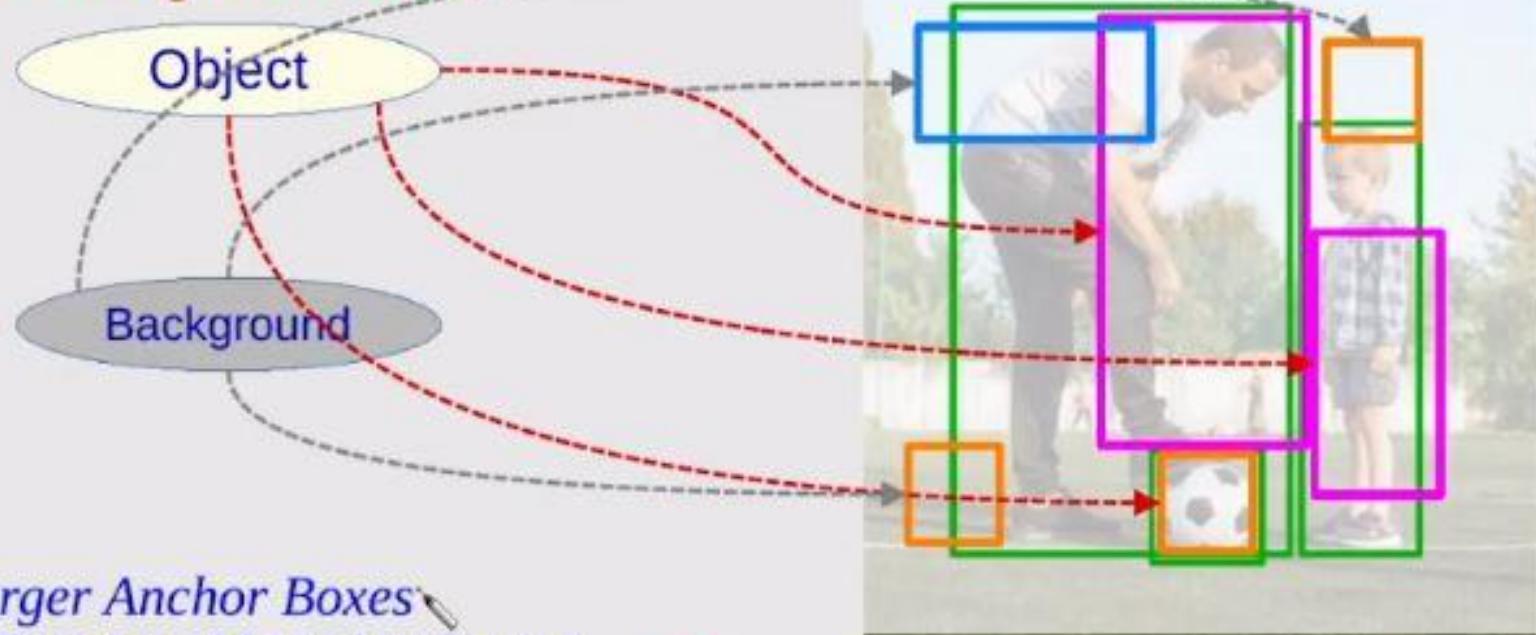


*Example of Some Anchor boxes with
High IOU (Objects)*

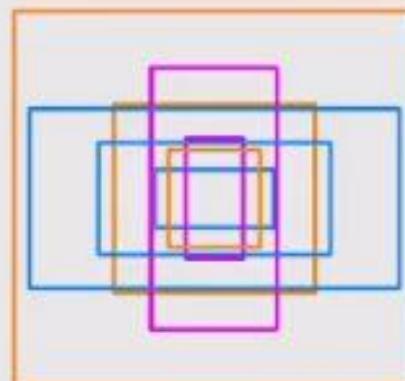
Others with small IOU (Background)

Anchor Boxes classification

Some Anchor boxes with High IOU (Objects)



Some Anchor Boxes with small IOU (Background)



Define Larger Anchor Boxes
3 scales, 3 ratios (9 anchor boxes)

Note: Many Overlaps between boxes
(No Problem)

Number of Anchor Boxes (in the above example):

If K=9 Anchor boxes per anchor point (3 scales * 3 ratios)

$$\# \text{ of anchor points} = 3 * 4 = 12$$

Total Number of Anchor boxes (within Image)= $12 * 9=108$ boxes

Use Different Scales

For Example: 128 x 128 256 x 256 512 x 512

Three ratios for each scale

1:1

2:1

1:2

[2] Anchor Boxes Parameters

Anchor Boxes Parameters

Each Anchor Box is characterized by two vectors:

[1] Content Classification vector:

- Enclose Object
- Part of the Background
- Mix between part of object and part of background

Classification (or *Objectness*) Vector Length=2

[2] Bounding Box offsets vector [w.r.t Ground truth box of its enclosed object]

(for Anchor boxes classified as containing objects ONLY)

- Displacement between both centers
- Ratio between Width and height of Anchor Box and Ground Truth Box

Regressor Vector Length = 4

[1] Classification of Anchor Boxes

Assign class label to each anchor as:

- Anchor Box encloses Object (**Positive** Anchor Box)
- Anchor Box encloses Background (**Negative** Anchor Box)
- Anchor Box encloses part of Object and Part of Background (**Mix** Anchor Box)

Remember:

*Classification of Anchor Boxes is based on IOU Between
Anchor Box and Ground Truth Object Boxes in the image*

[1] Classification of Anchor Boxes

*Classification of Anchor Boxes is based on **IOU** Between
Anchor Box and Ground Truth Object Boxes in the image*

Anchor Box is classified as Positive if:

- 1- Has the **Highest IOU** with one of the ground truth boxes (one of the objects)
- 2- Has **IOU ≥ 0.7** with any of the ground truth boxes (even if not the Highest)

Anchor Box is classified as Negative if:

- 1- Has **IOU ≤ 0.3** with ALL ground truth boxes (all objects)

Anchor Box is classified as Mix (Undetermined) if:

- 1- Neither Positive Nor Negative

[1] Classification of Anchor Boxes

Positive

- Highest IOU with the Father Object $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

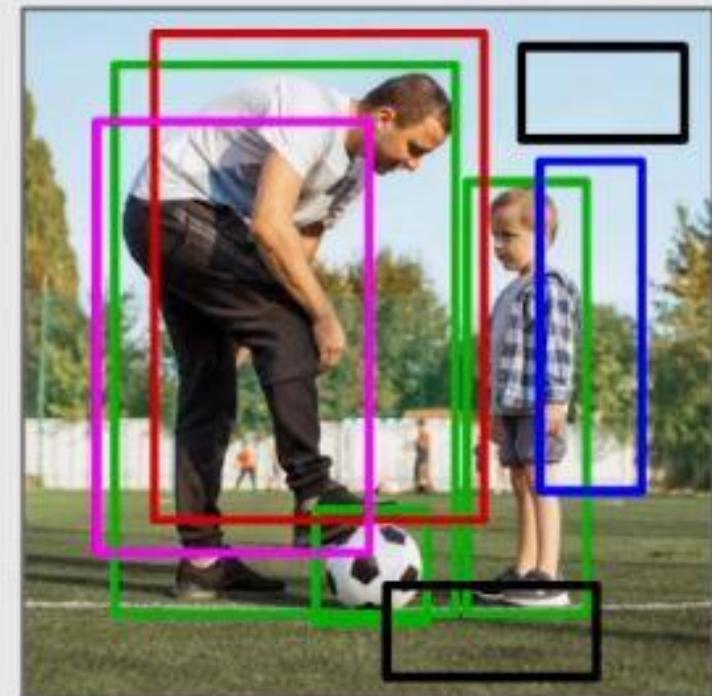
- IOU > 0.7 with the Father Object (but not the highest) ↴

Negative

- IOU < 0.3 with ALL Objects $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

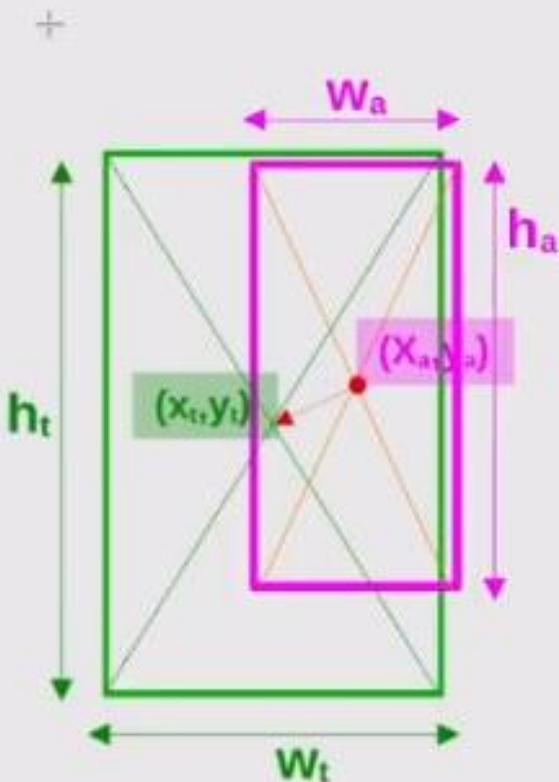
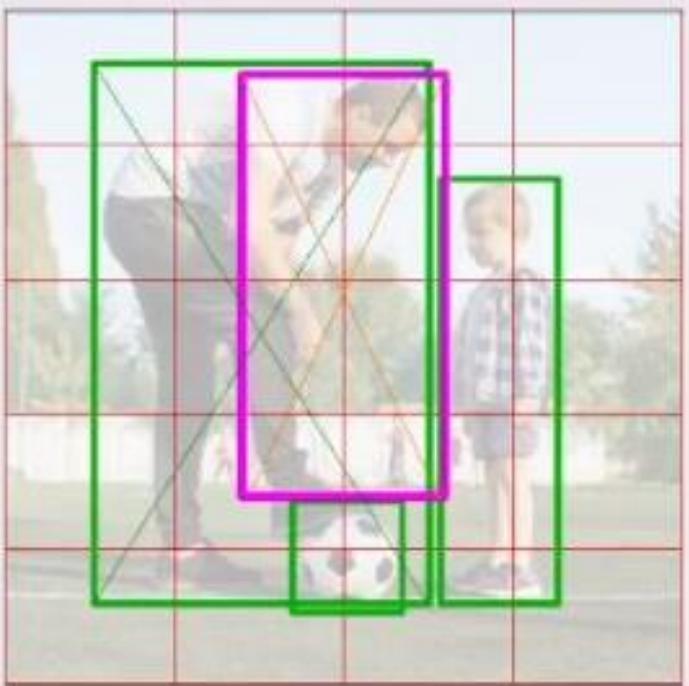
Undetermined (Mixed)

- $0.3 < \text{IOU} < 0.7$ with ALL Objects $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$



Object (Ground Truth)

[2] Regressor Vector of Anchor Box



Regressor Vector
contains $[X, Y, W, H]$
of each Anchor Box

Define Ground Truth Box (target) with

(x_t, y_t) coordinates of the center of object

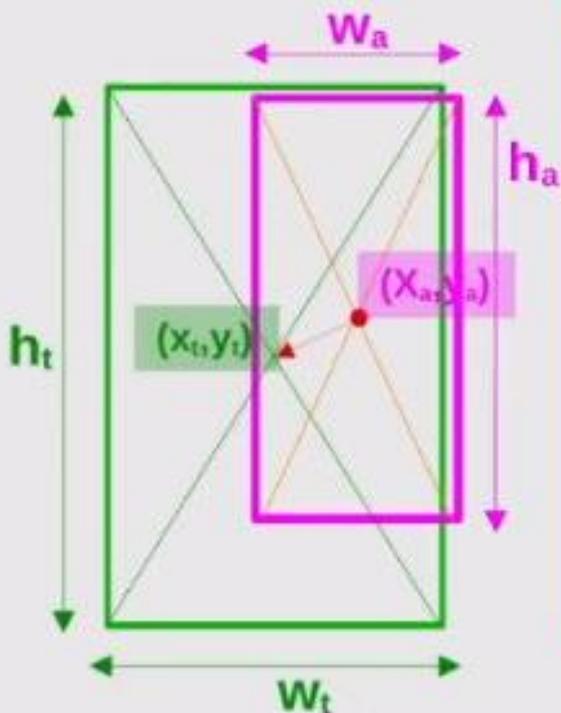
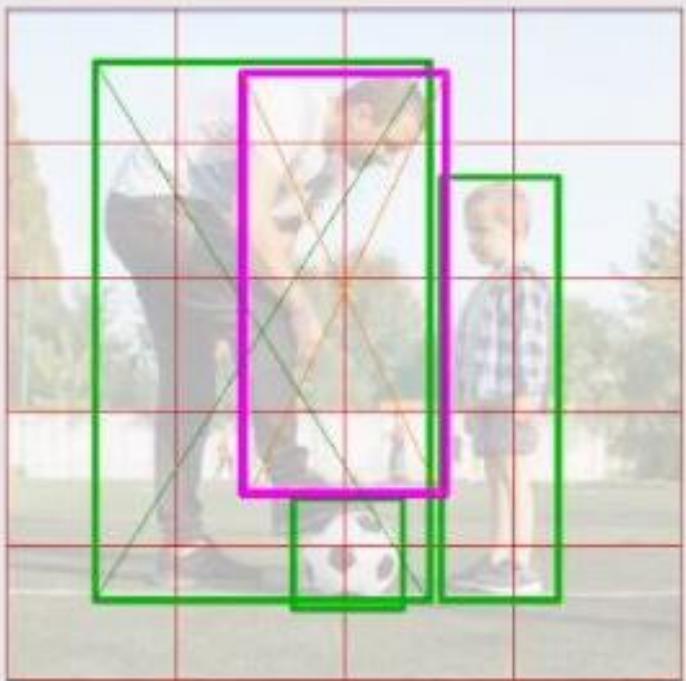
(w_t, h_t) Width and height of Object

Define Anchor Box with

(x_a, y_a) coordinates of the center of box

(w_a, h_a) Width and height of box

[2] Regressor Vector of Anchor Box



To adapt **Anchor Box size** to fit on **Object Truth Box (Target Box)**

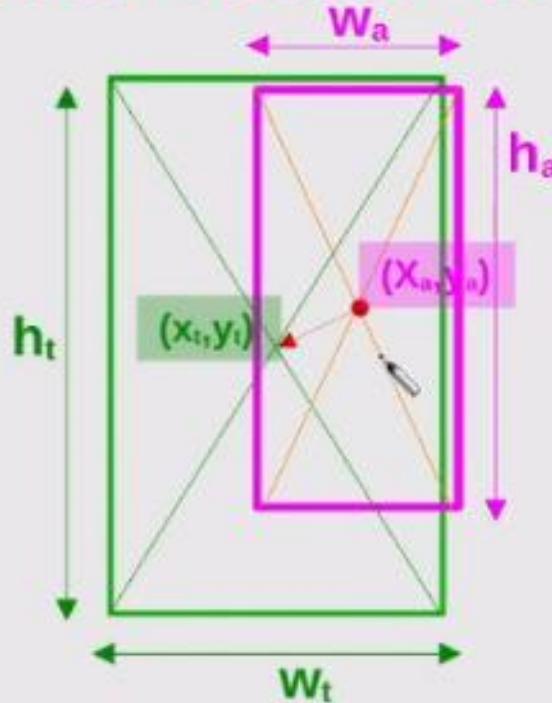
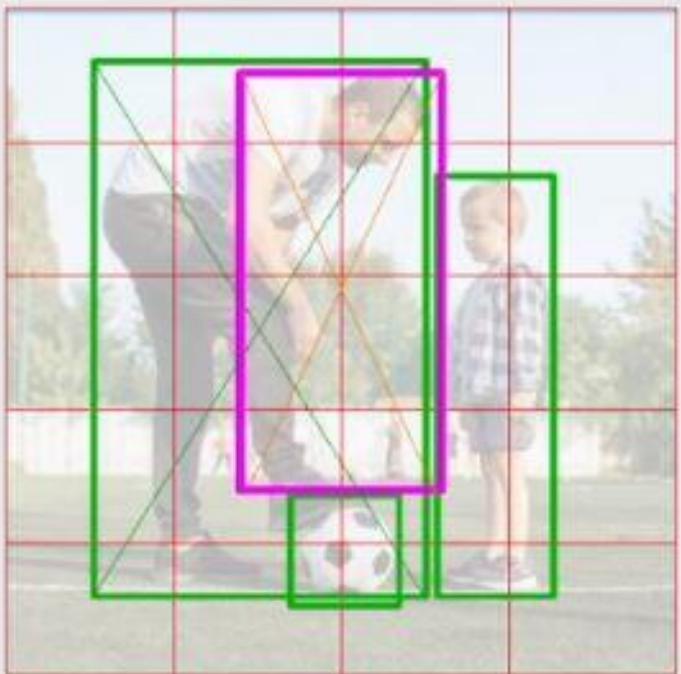
$(x_a, y_a) \rightarrow (x_t, y_t)$
displacement is $(x_t - x_a)$ and $(y_t - y_a)$

$(w_a, h_a) \rightarrow (w_t, h_t)$
Scale difference is w_t/w_a and h_t/h_a

Define Ground Truth Box (target) with
 (x_t, y_t) coordinates of the center of object
 (w_t, h_t) Width and height of Object

Define Anchor Box with
 (x_a, y_a) coordinates of the center of box
 (w_a, h_a) Width and height of box

[2] Regressor Vector of Anchor Box



To adapt Anchor Box size to fit on Object Truth Box (Target Box)

$(x_a, y_a) \rightarrow (x_t, y_t)$
displacement is $(x_t - x_a)$ and $(y_t - y_a)$

$(w_a, h_a) \rightarrow (w_t, h_t)$
Scale difference is w_t/w_a and h_t/h_a

Normalize values in order to be object size independent

$$\text{Use } d_x = (x_t - x_a)/x_a$$

$$d_y = (y_t - y_a)/y_a$$

$$d_w = w_t/w_a$$

$$d_h = h_t/h_a$$



If Anchor Box fit 100% with Object box then

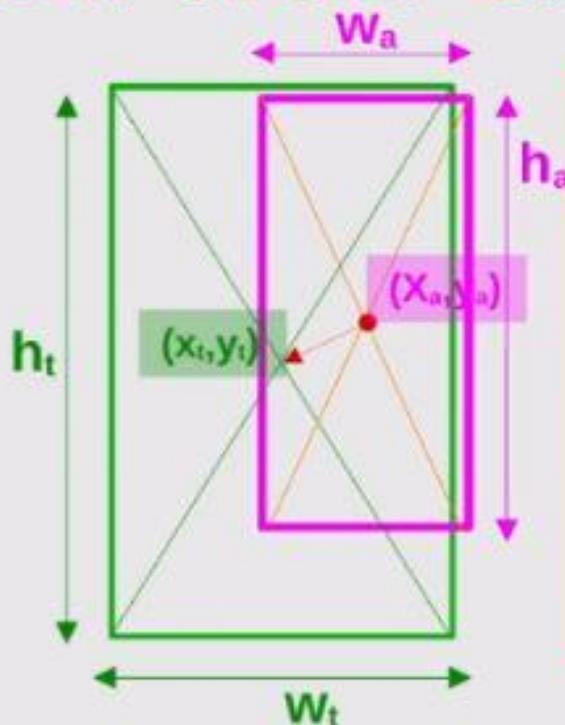
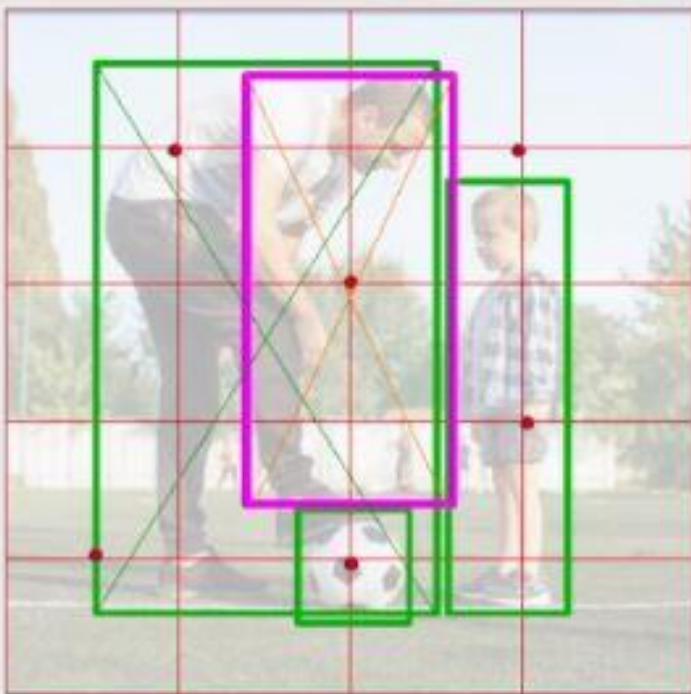
$$d_x = 0$$

$$d_y = 0$$

$$d_w = 1 \quad \dots \dots ??$$

$$d_h = 1 \quad \dots \dots ??$$

[2] Regressor Vector of Anchor Box



To adapt Anchor Box size to fit on Object Truth Box (Target Box)

$(x_a, y_a) \rightarrow (x_t, y_t)$
displacement is $(x_t - x_a)$ and $(y_t - y_a)$

$(w_a, h_a) \rightarrow (w_t, h_t)$
Scale difference is w_t/w_a and h_t/h_a

Normalize values in order to be object size independent

$$\text{Use } d_x = (x_t - x_a)/x_a$$

$$d_y = (y_t - y_a)/y_a$$

$$d_w = w_t/w_a$$

$$d_h = h_t/h_a$$

If Anchor Box fit 100% with Object box then

$$d_x = 0$$

$$d_y = 0$$

$$d_w = 1 \quad \dots \dots ??$$

$$d_h = 1 \quad \dots \dots ??$$

Use Log function

$$d_x = (x_t - x_a)/x_a$$

$$d_y = (y_t - y_a)/y_a$$

$$d_w = \log(w_t/w_a)$$

$$d_h = \log(h_t/h_a)$$

If Anchor Box fit 100% with Object box then

$$d_x = 0$$

$$d_y = 0$$

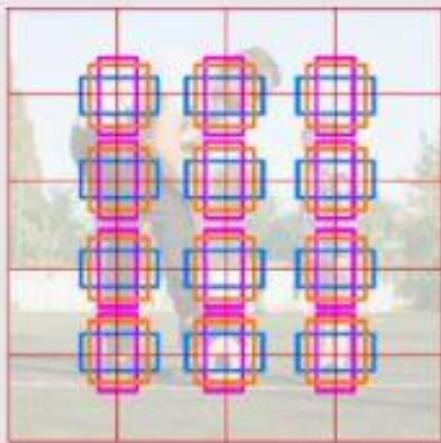
$$d_w = 0$$

$$d_h = 0$$

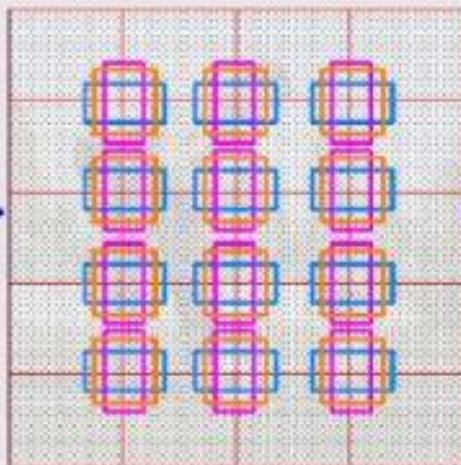
All Values are zeros

[3] RPN Training \ Data Preparation

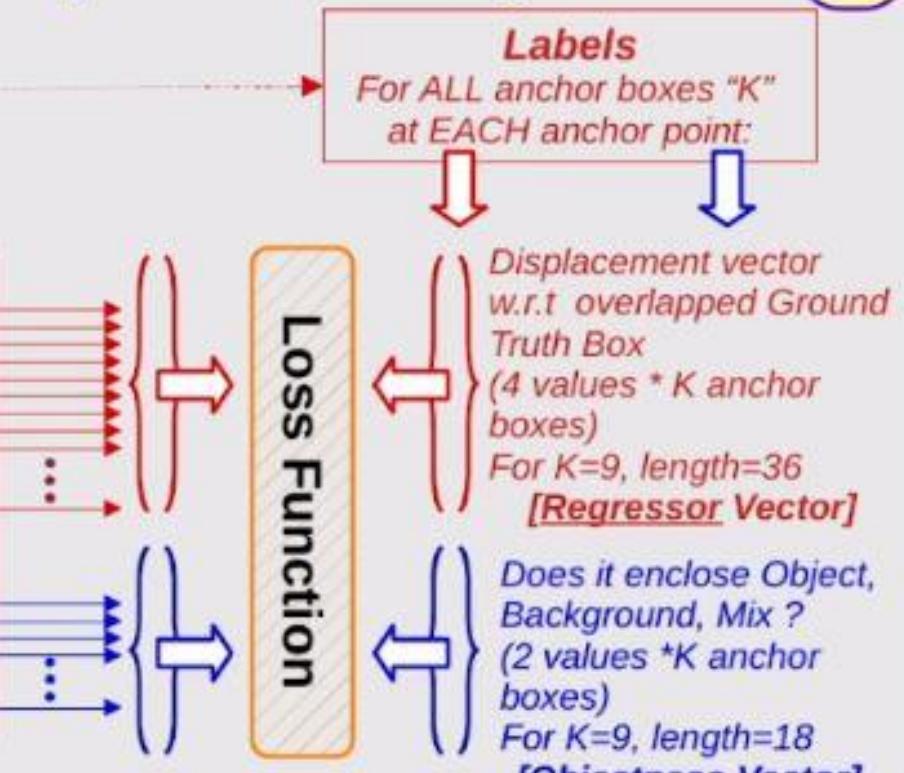
[Training of RPN (*The Complete Picture*)



Pre-Trained CNN Model
e.g. VGG 16



Region Proposal Network (RPN)



Contents of anchor boxes can be extracted directly from the Feature map (instead of using Spacial domain image)

How to Prepare Training Data for RPN

- 1) Get a training dataset that contain images with Annotated objects (bounding boxes)
- 2) Use any pre-trained CNN (e.g. VGG16) to obtain the feature map of the input image
- 3) For each anchor point in the input image, generate coordinates for all "K" anchor boxes (centered at this anchor point)
- 4) Calculate IOU between those anchor boxes and ALL objects ground truth boxes.
- 5) Mark each anchor box as enclosing Object [1 , 0], Background only [0 , 1], or Mix [0 , 0].
This "Objectness" vector length should be "K"*2

How to Prepare Training Data for RPN

- 6) For anchor boxes including objects, calculate the displacement vector (4 values) between the anchor box and corresponding object box.
- 7) For other anchor boxes, fill the values with zeros. (will not be used). This "Regression" vector length should be "K"*4
- 8) Provide RPN with both feature map (as Input) and "Objectness & Regression" vectors (for each anchor point) as Label.

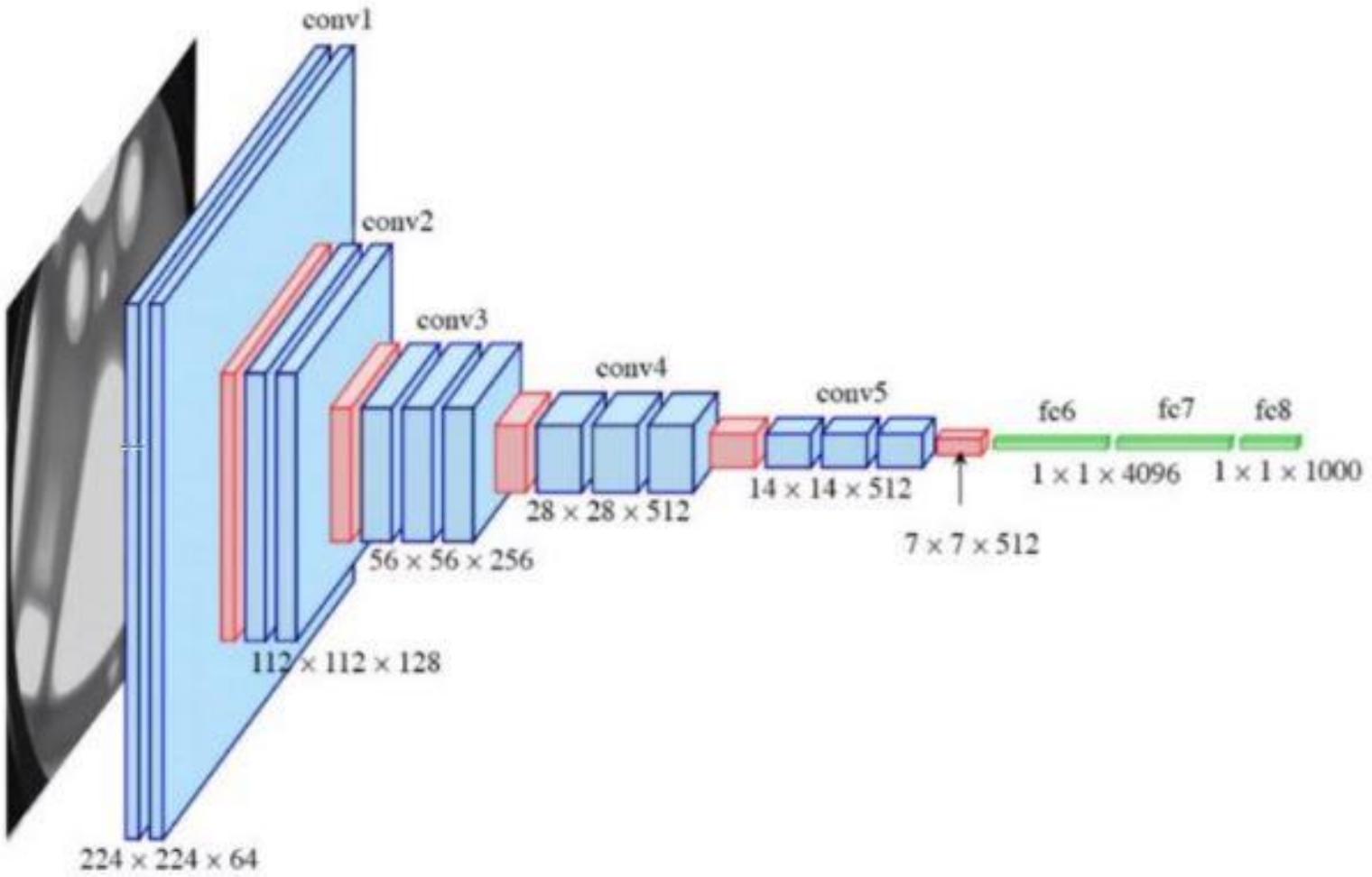
+

[4] Quick Review of VGG 16

(to Generate Feature Map of Input Image)

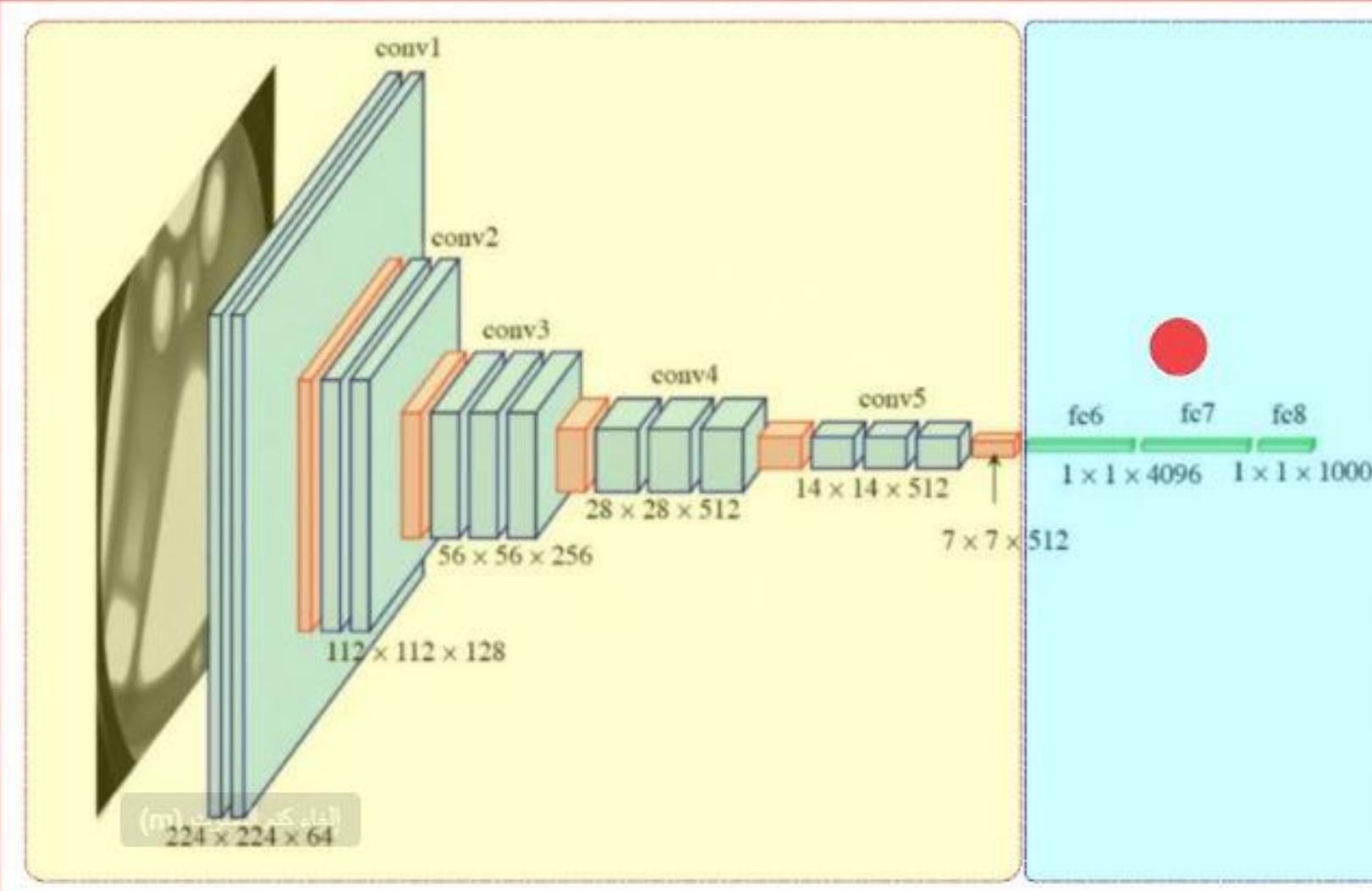
(One may use any other CNN network as AlexNet, ResNet,..)

VGG 16 Architecture



- convolution+ReLU
- max pooling
- fully connected+ReLU

VGG 16 Architecture



Convolution Part of VGG16

Input Image $224 \times 224 \times 3$

Conv1: 64 filters $224 \times 224 \times 64$
Pool $112 \times 112 \times 64$

Conv2: 128 filters $112 \times 112 \times 128$
Pool $56 \times 56 \times 128$

Conv3: 256 filters $56 \times 56 \times 256$
Pool $28 \times 28 \times 256$

Conv4: 512 filters $28 \times 28 \times 512$
Pool $14 \times 14 \times 512$

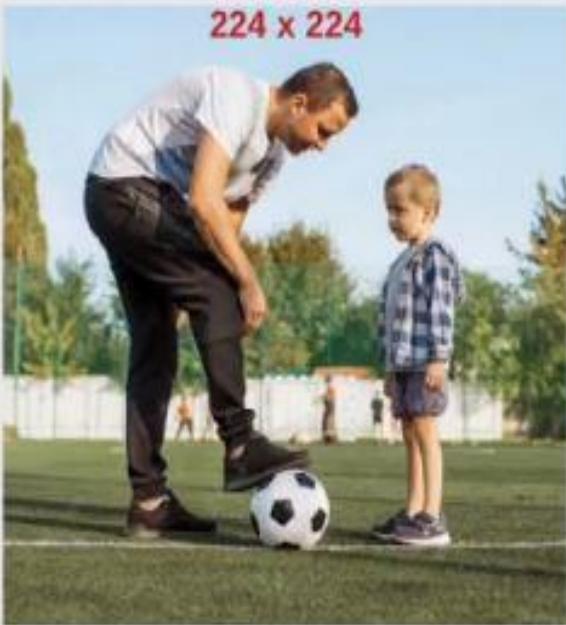
Conv5: 512 filters $14 \times 14 \times 512$
Pool $7 \times 7 \times 512$

- █ convolution+ReLU
- █ max pooling
- █ fully connected+ReLU

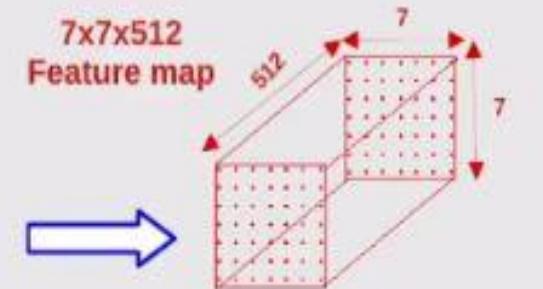
[5] How to Apply 1x1 Convolution

to obtain

Regressor and Objectness Vectors



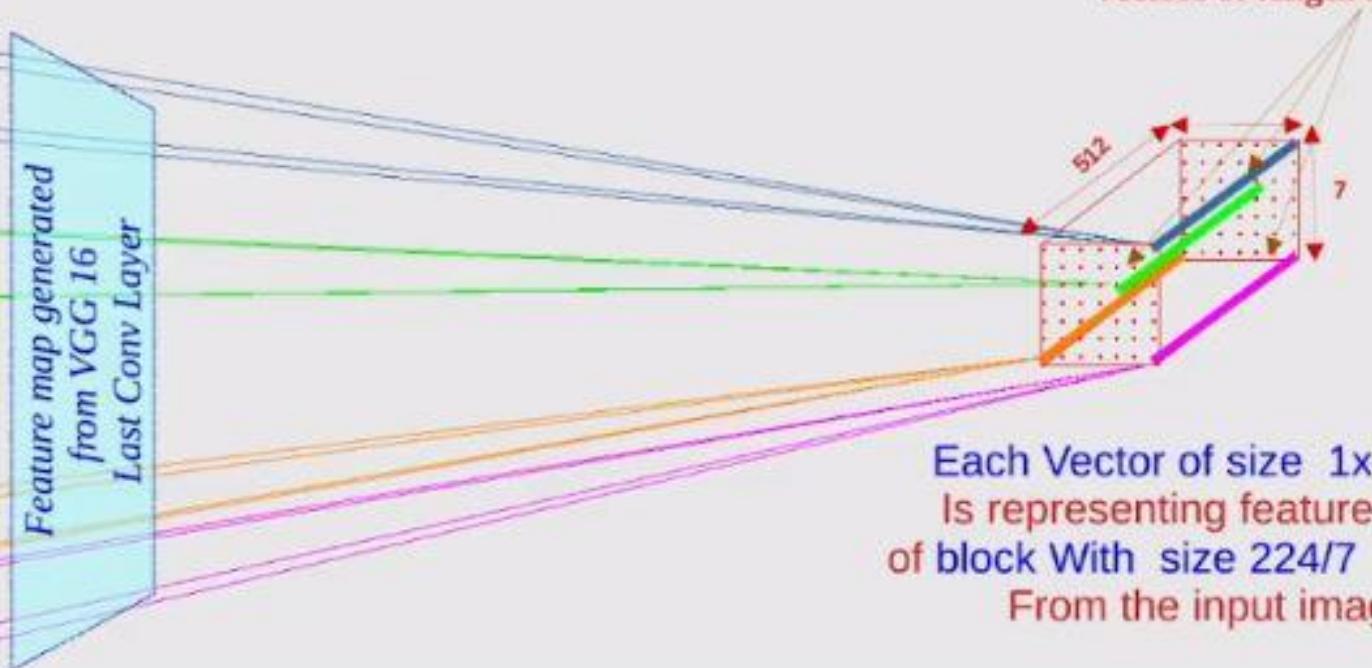
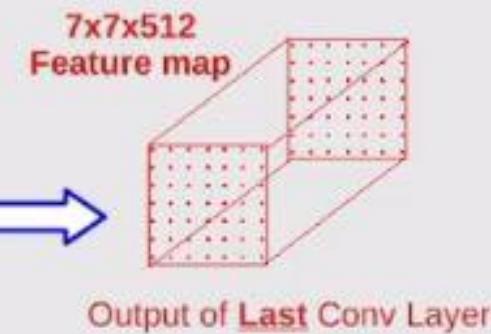
VGG 16
Pre-Trained CNN Model



Output of Last Conv Layer

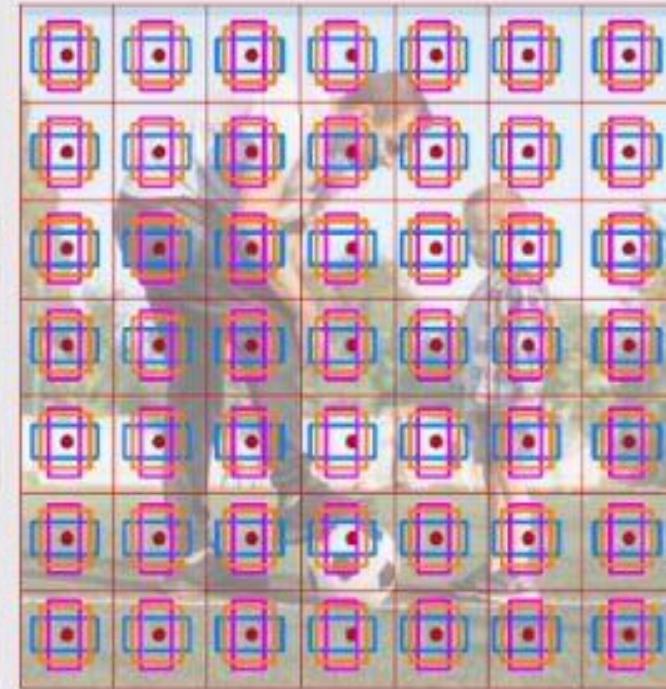


VGG 16
Pre-Trained CNN Model



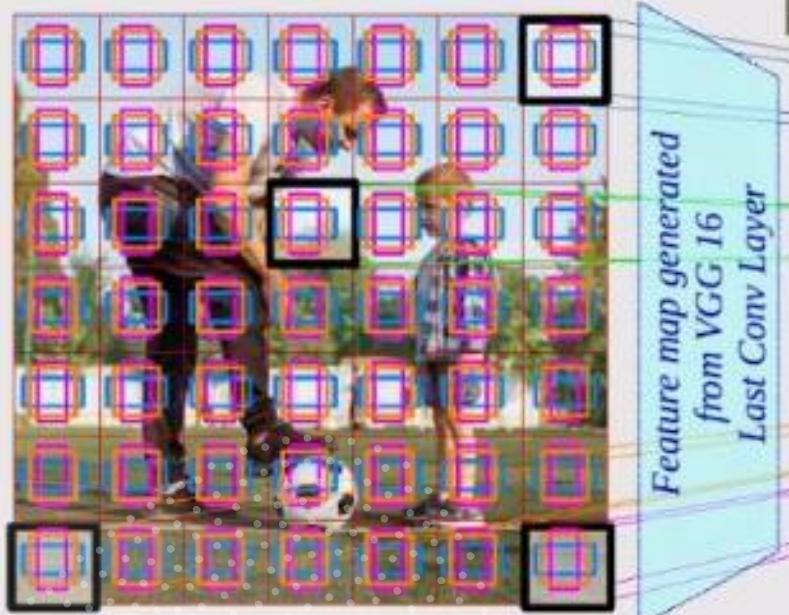
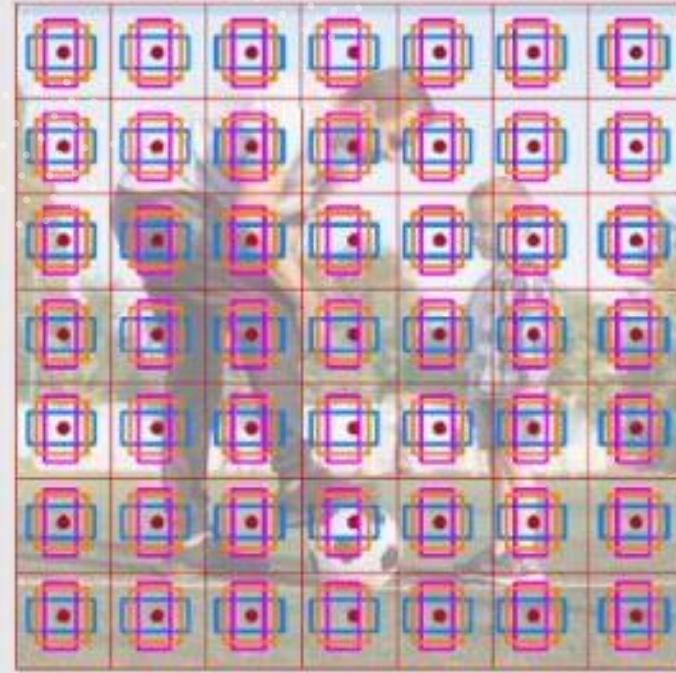
Remember:

- Input Image is divided into Cells (e.g. 7x7 cells as last Conv layer of VGG16)
- Center of each cell is an Anchor Point
- Each anchor point is associated with “K” anchor boxes (K=9 in the paper)



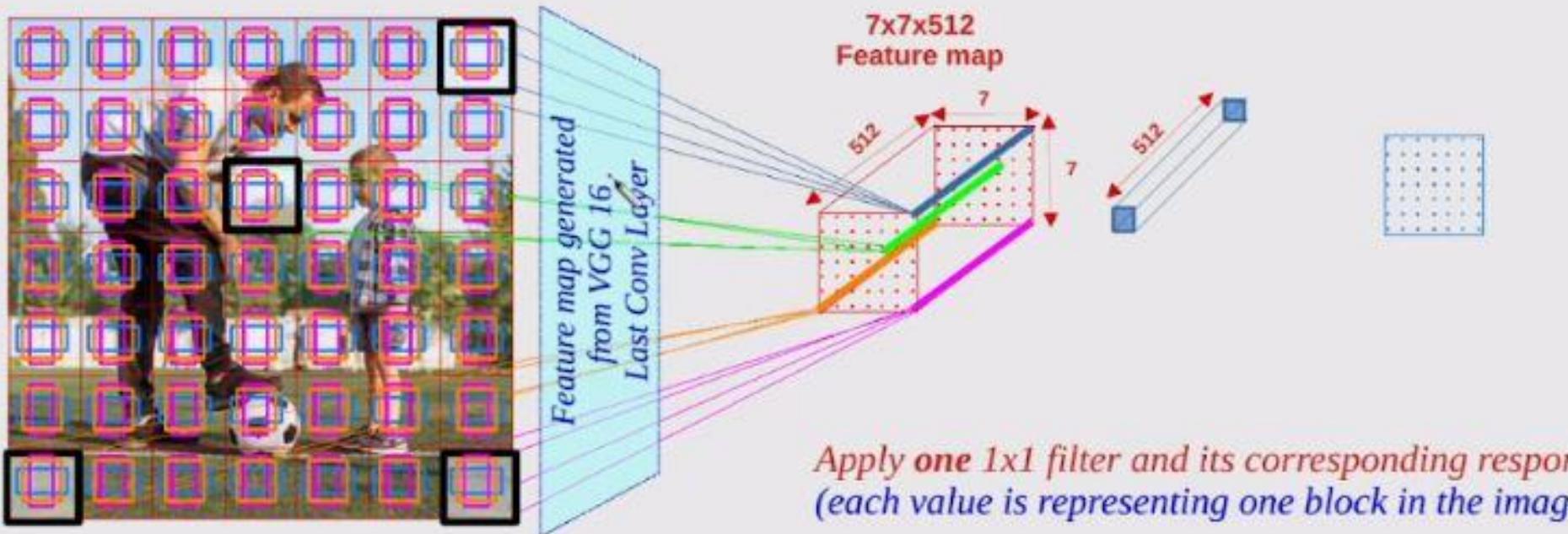
Remember:

- Input Image is divided into Cells (e.g. 7x7 cells as last Conv layer of VGG16)
- Center of each cell is an Anchor Point
- Each anchor point is associated with “K” anchor boxes (K=9 in the paper)

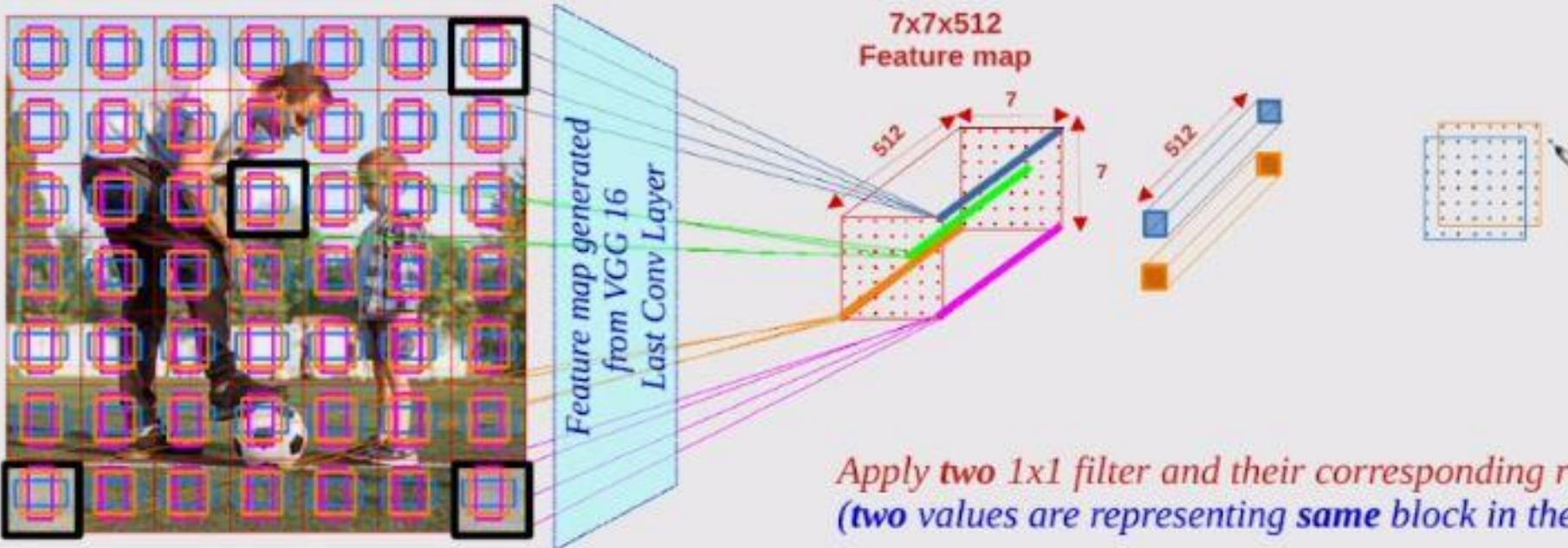


Each Vector of size $1 \times 1 \times 512$
Is representing feature map
Of block covered by “K” Anchor Boxes
(Corresponding to one Anchor Point)

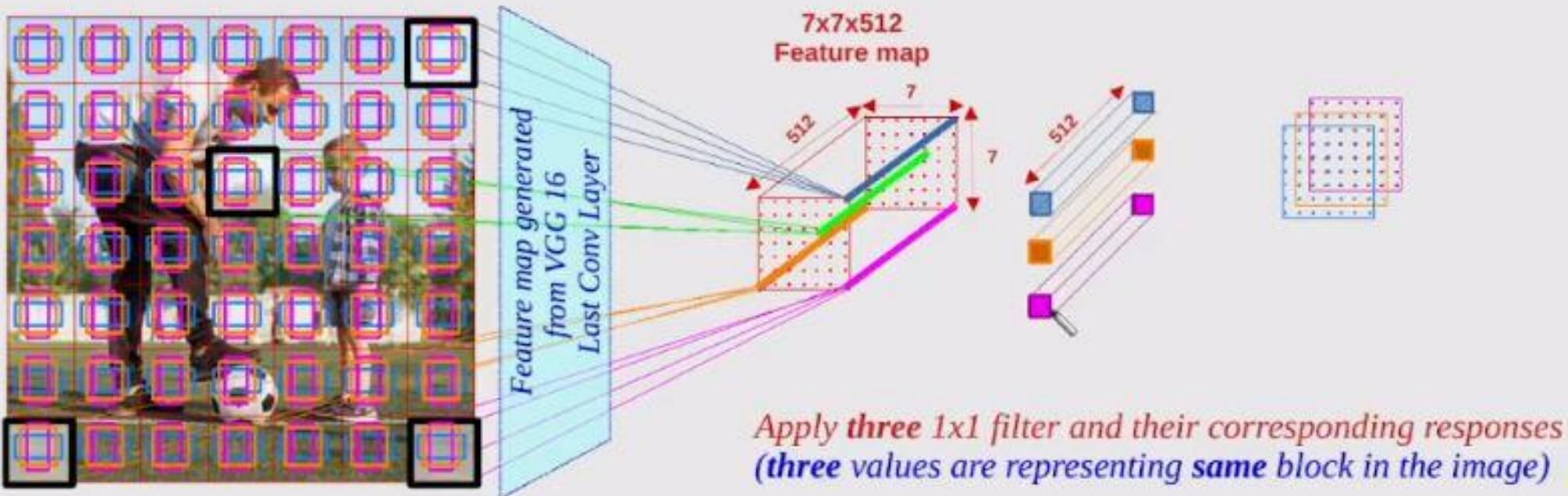
Apply 1x1 Convolution filter(s) *(on the feature map of last VGG Conv layer)*



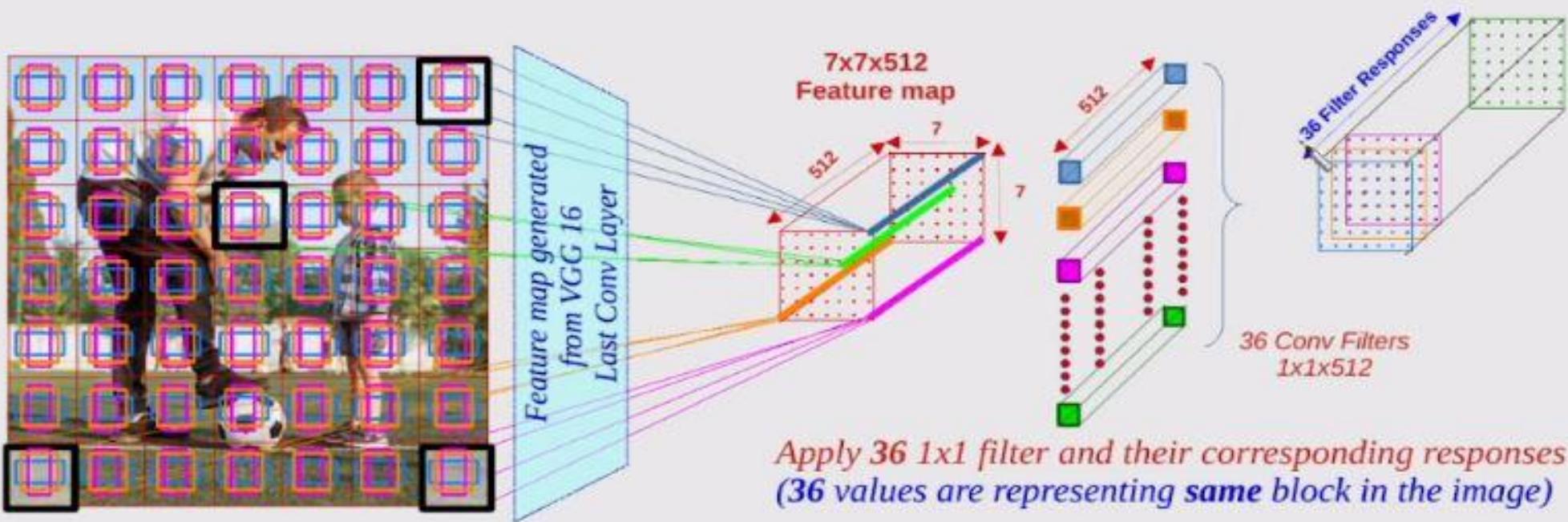
Apply 1x1 Convolution filter(s) *(on the feature map of last VGG Conv layer)*



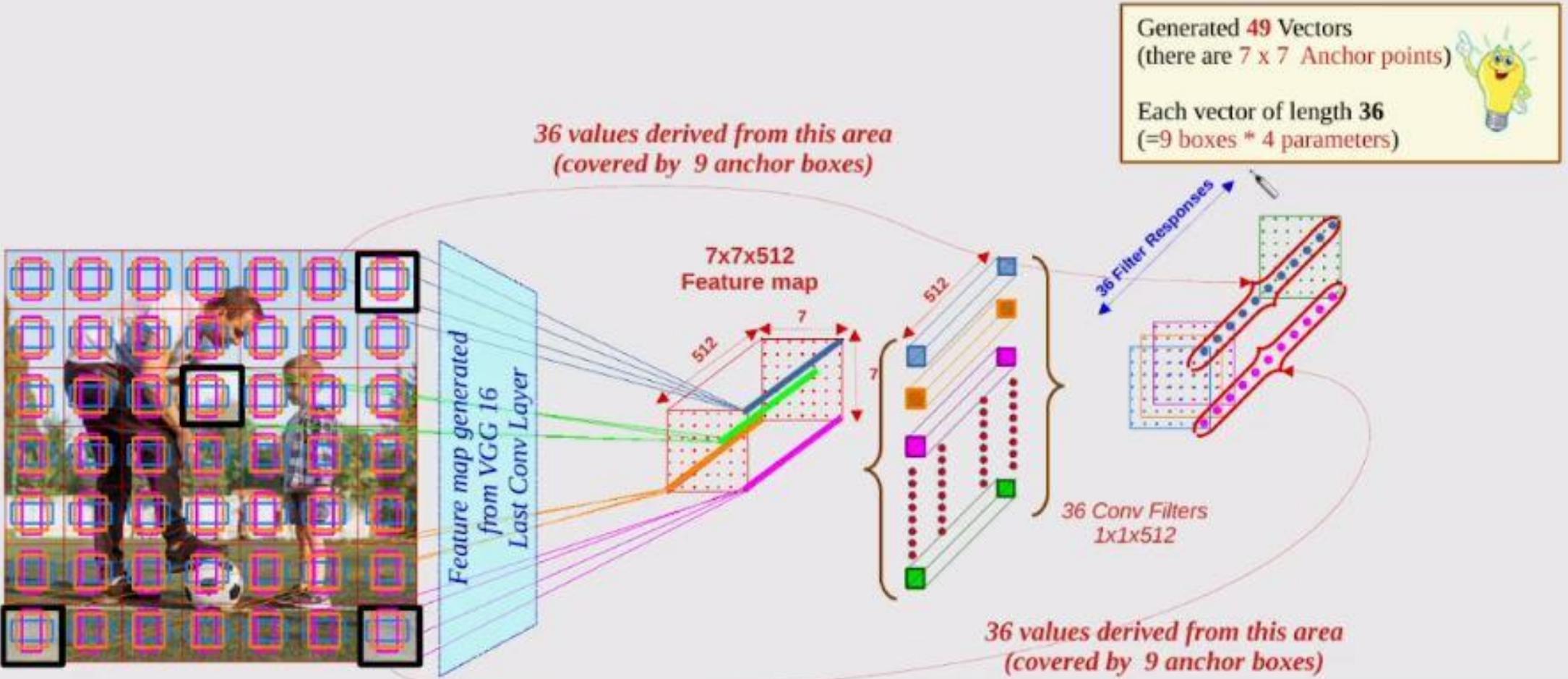
Apply 1x1 Convolution filter(s) *(on the feature map of last VGG Conv layer)*



Apply 1x1 Convolution filter(s) (on the feature map of last VGG Conv layer)



Apply 1x1 Convolution filter(s) (on the feature map of last VGG Conv layer)

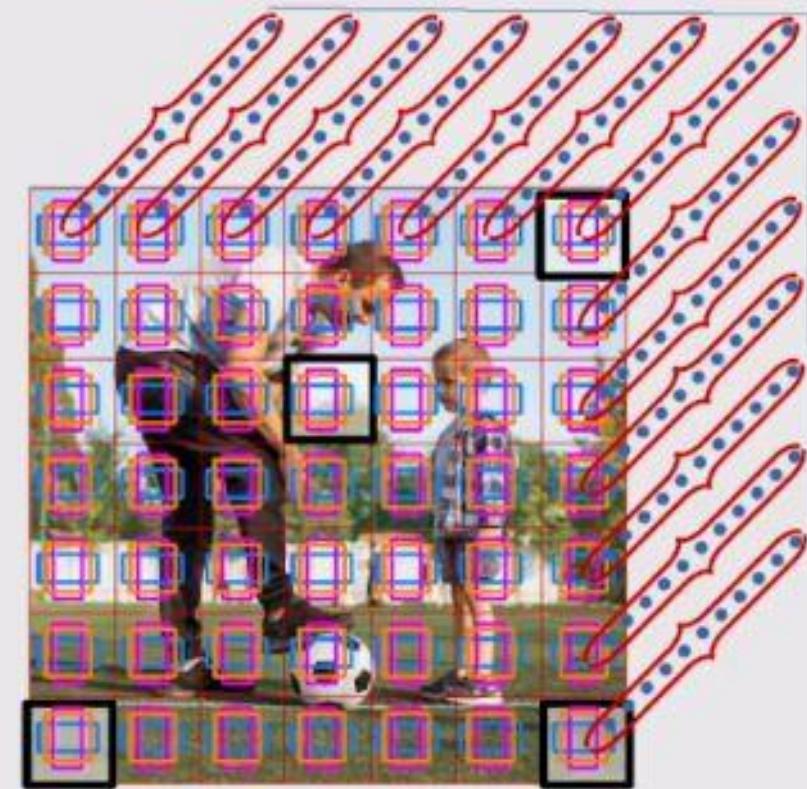
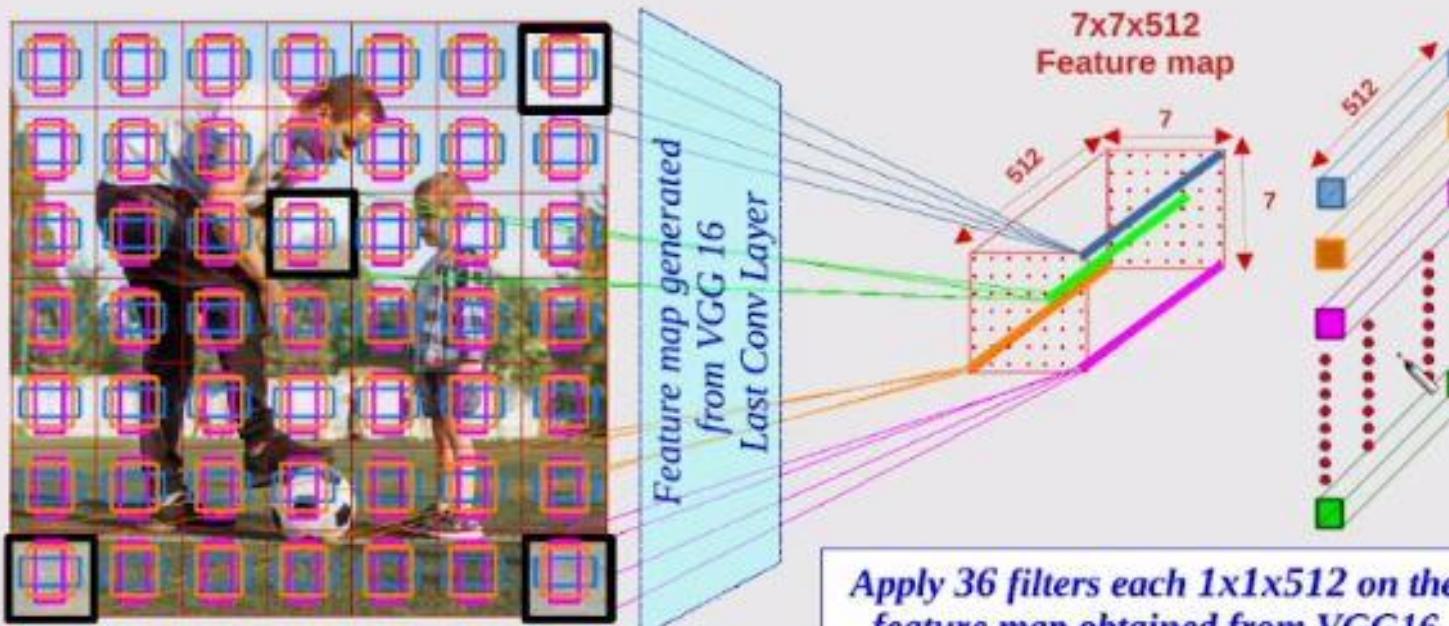


Apply 1x1 Convolution filter(s) (on the feature map of last VGG Conv layer)

Regressor Vectors

4 values per anchor box
36 values for 9 anchor boxes within same center

7x7 vectors each of 36 values
Each vector is derived from one area
(covered by 9 anchor boxes)

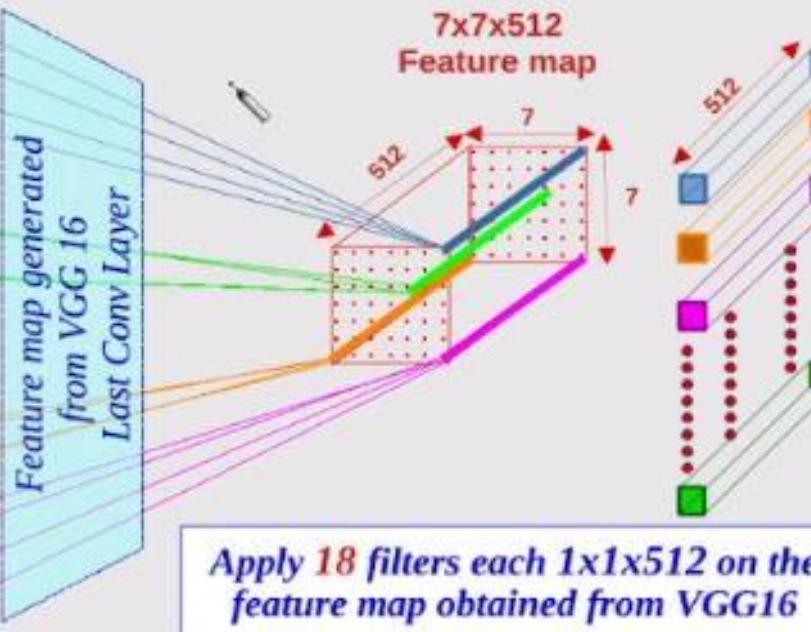
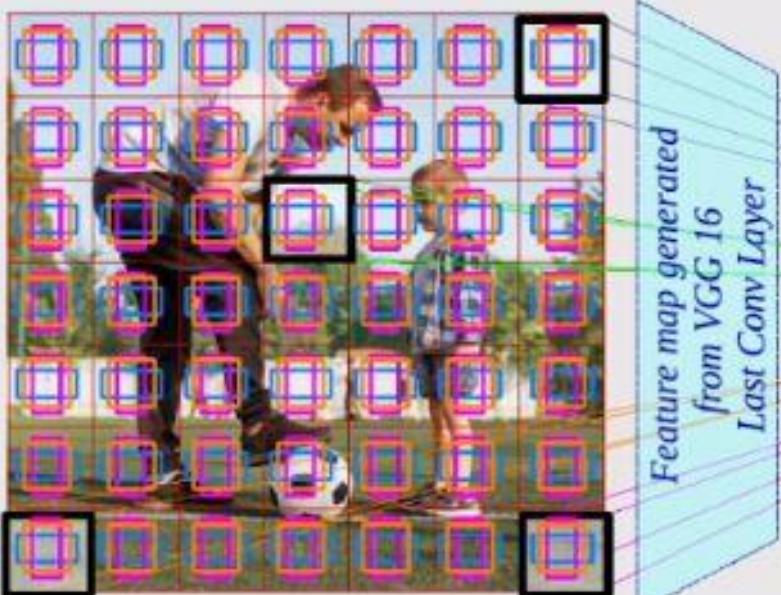


Apply 1x1 Convolution filter(s) (on the feature map of last VGG Conv layer)

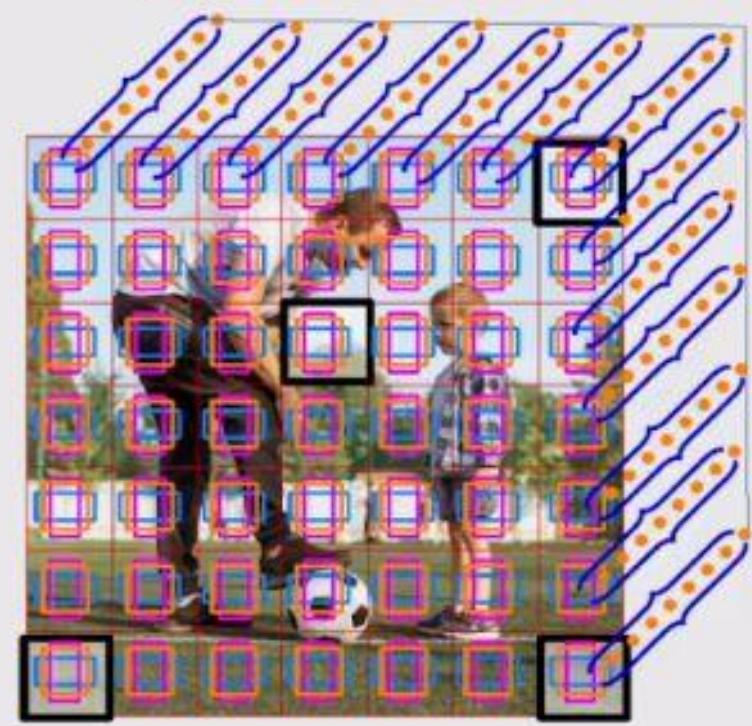
Objectness Vectors

2 values per anchor box

18 values for 9 anchor boxes within same center



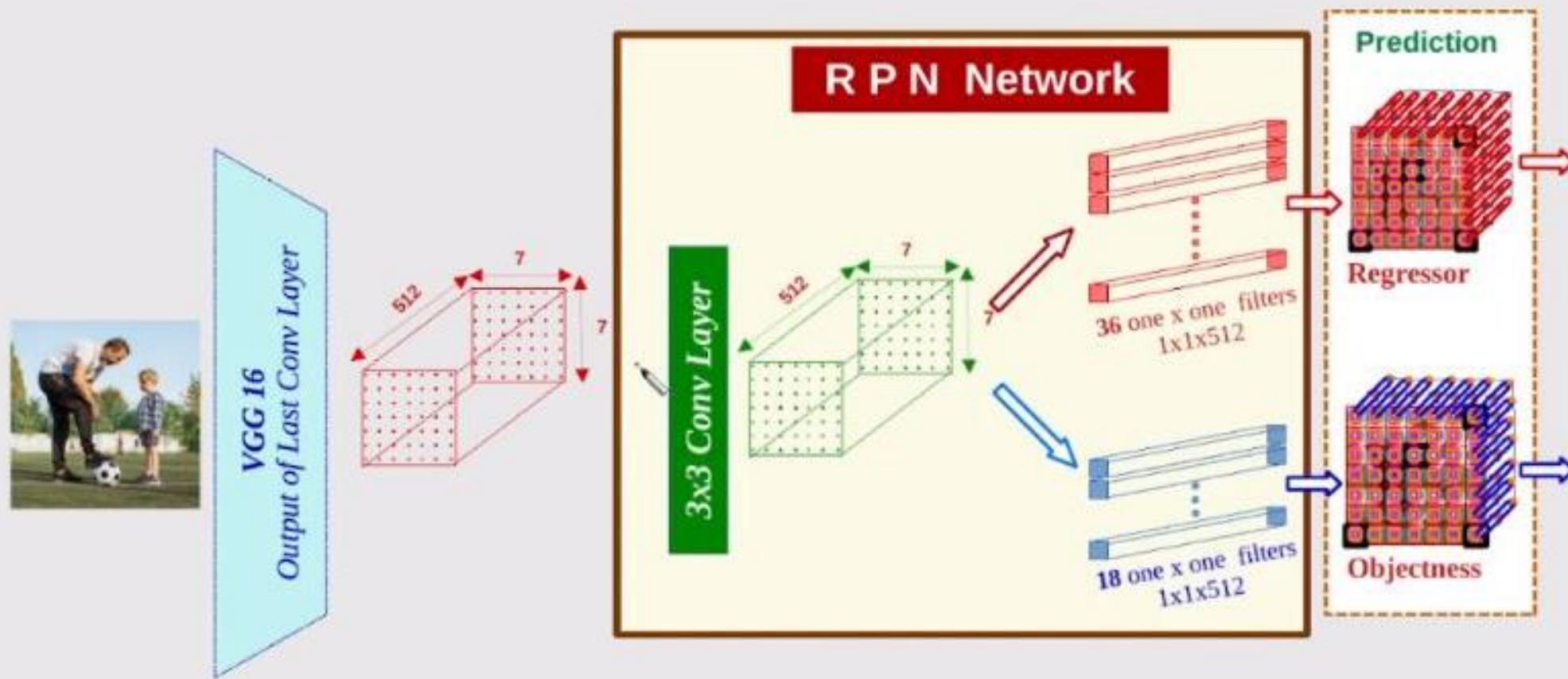
7x7 vectors each of 18 values
Each vector is derived from one area
(covered by 9 anchor boxes)



[6] RPN Architecture

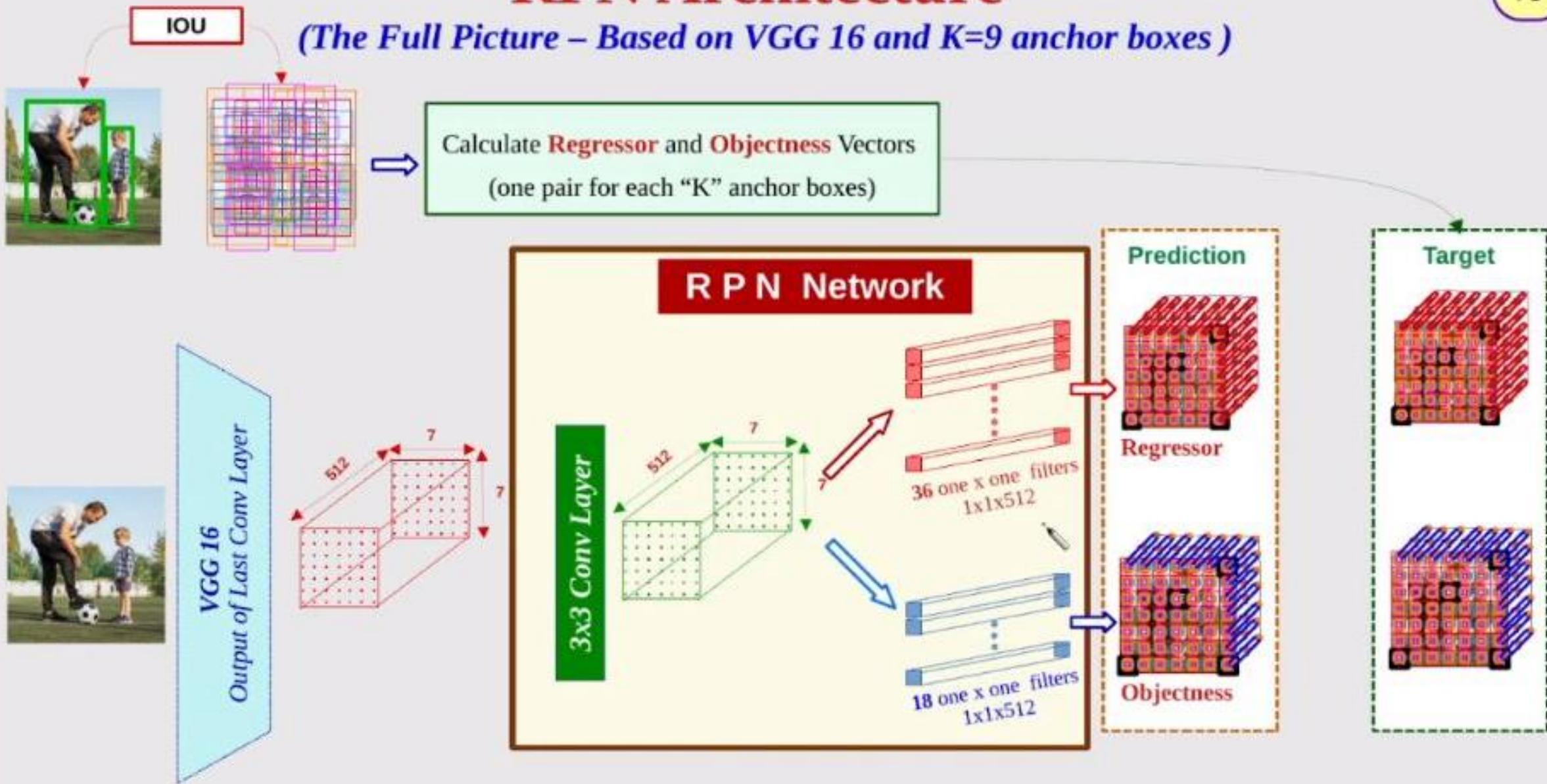
RPN Architecture

(The Full Picture – Based on VGG 16 and K=9 anchor boxes)



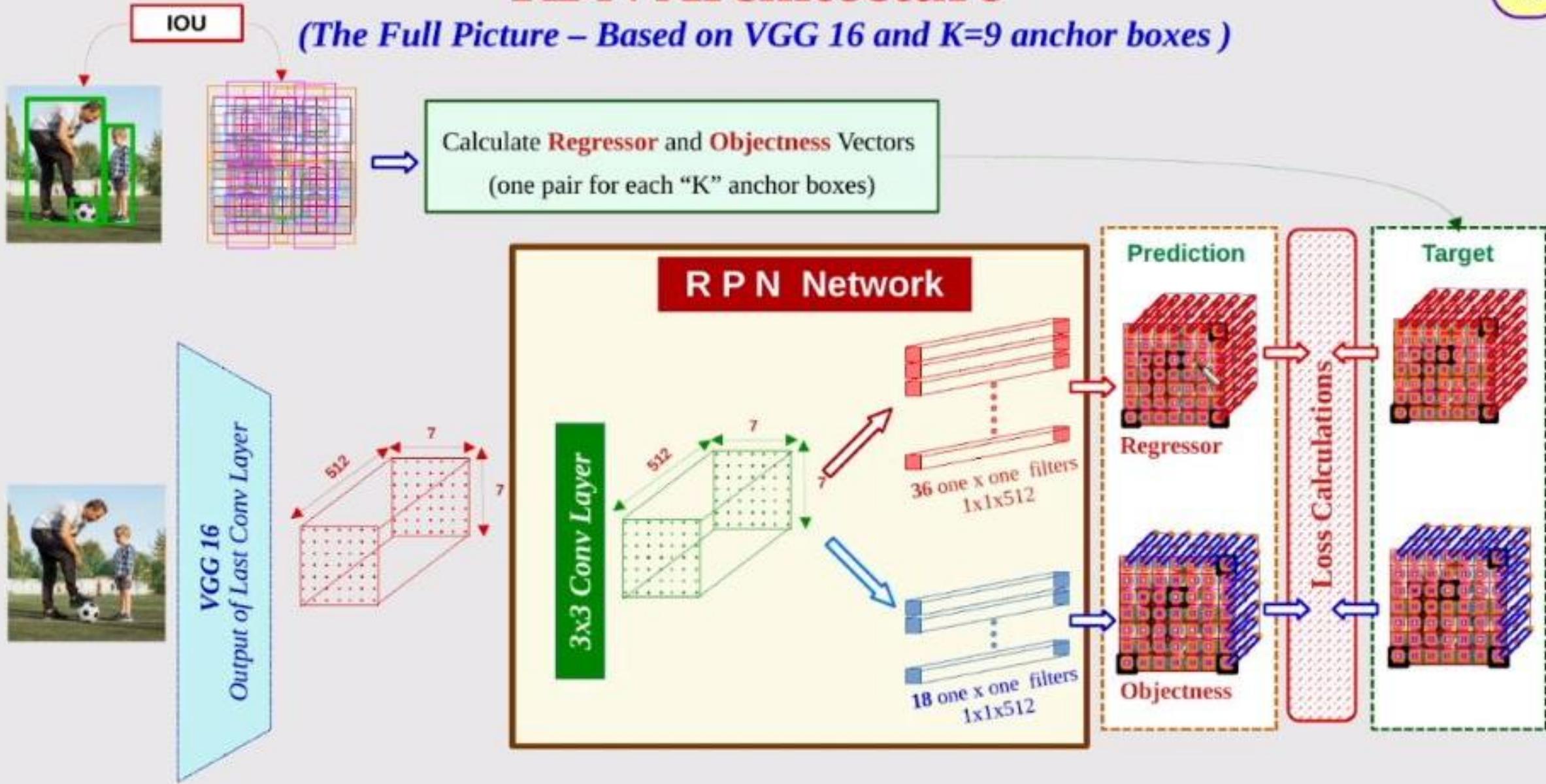
RPN Architecture

(The Full Picture – Based on VGG 16 and K=9 anchor boxes)



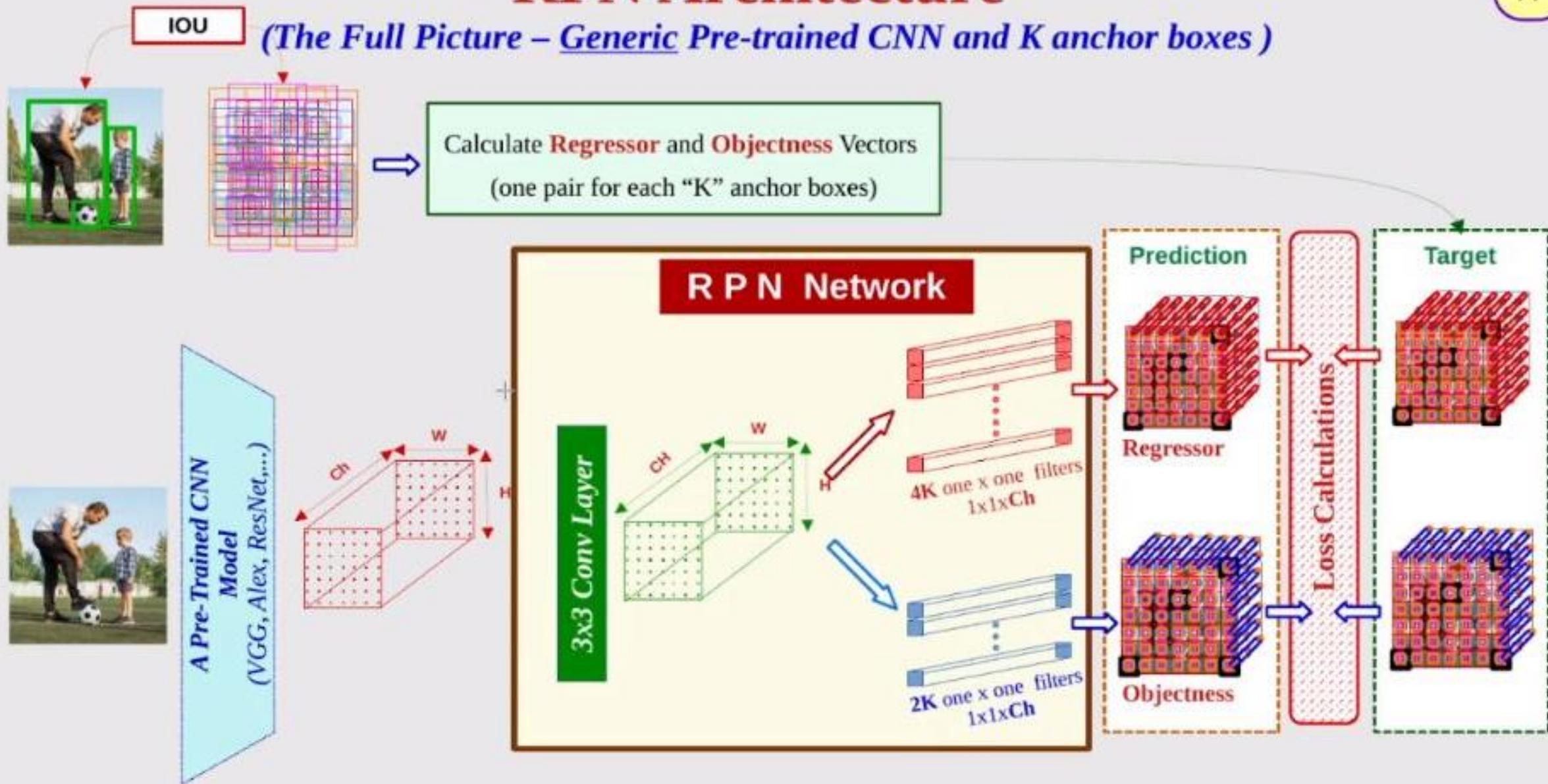
RPN Architecture

(The Full Picture – Based on VGG 16 and K=9 anchor boxes)



RPN Architecture

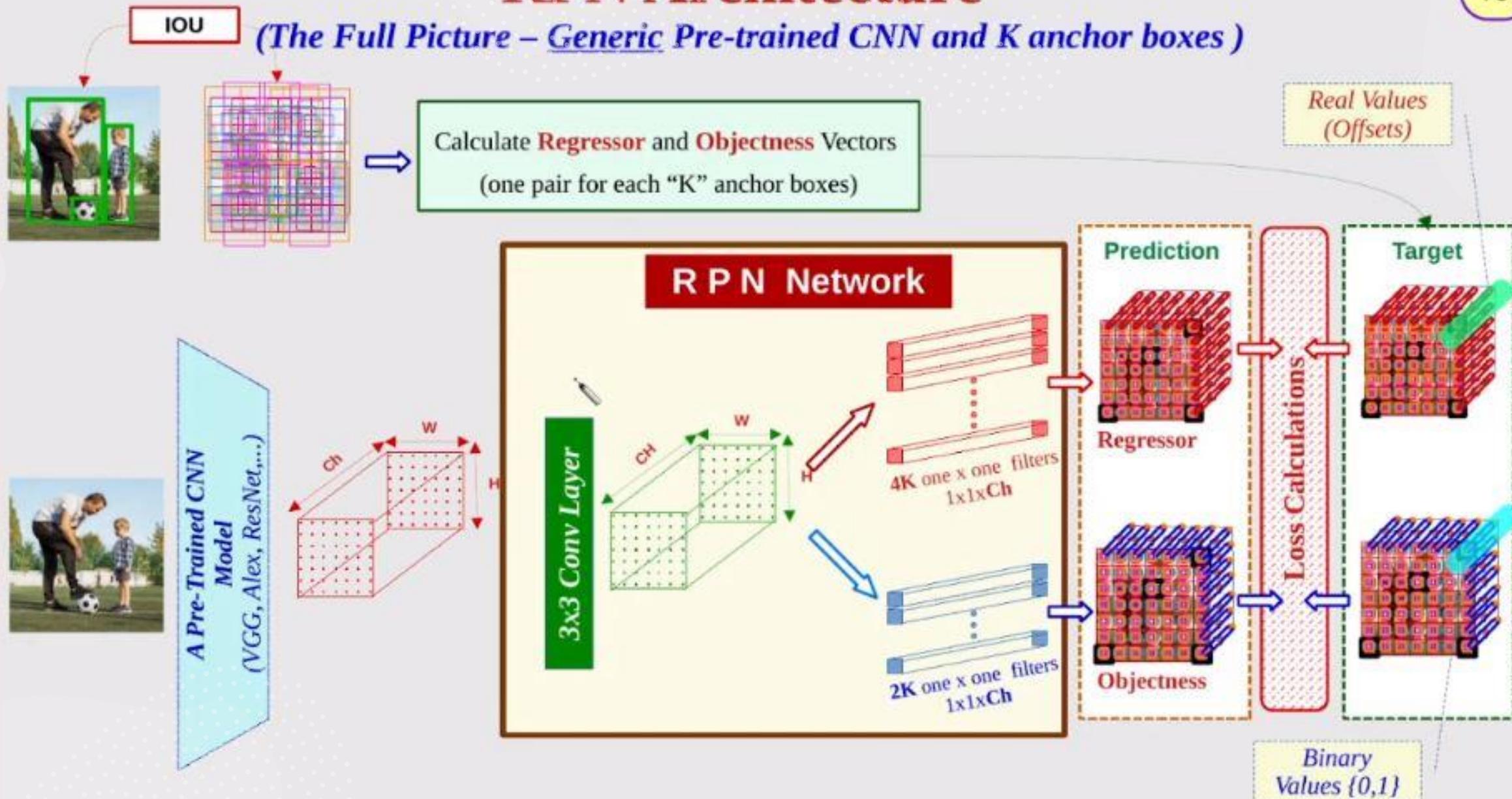
(The Full Picture – Generic Pre-trained CNN and K anchor boxes)



[7] RPN Loss Function

RPN Architecture

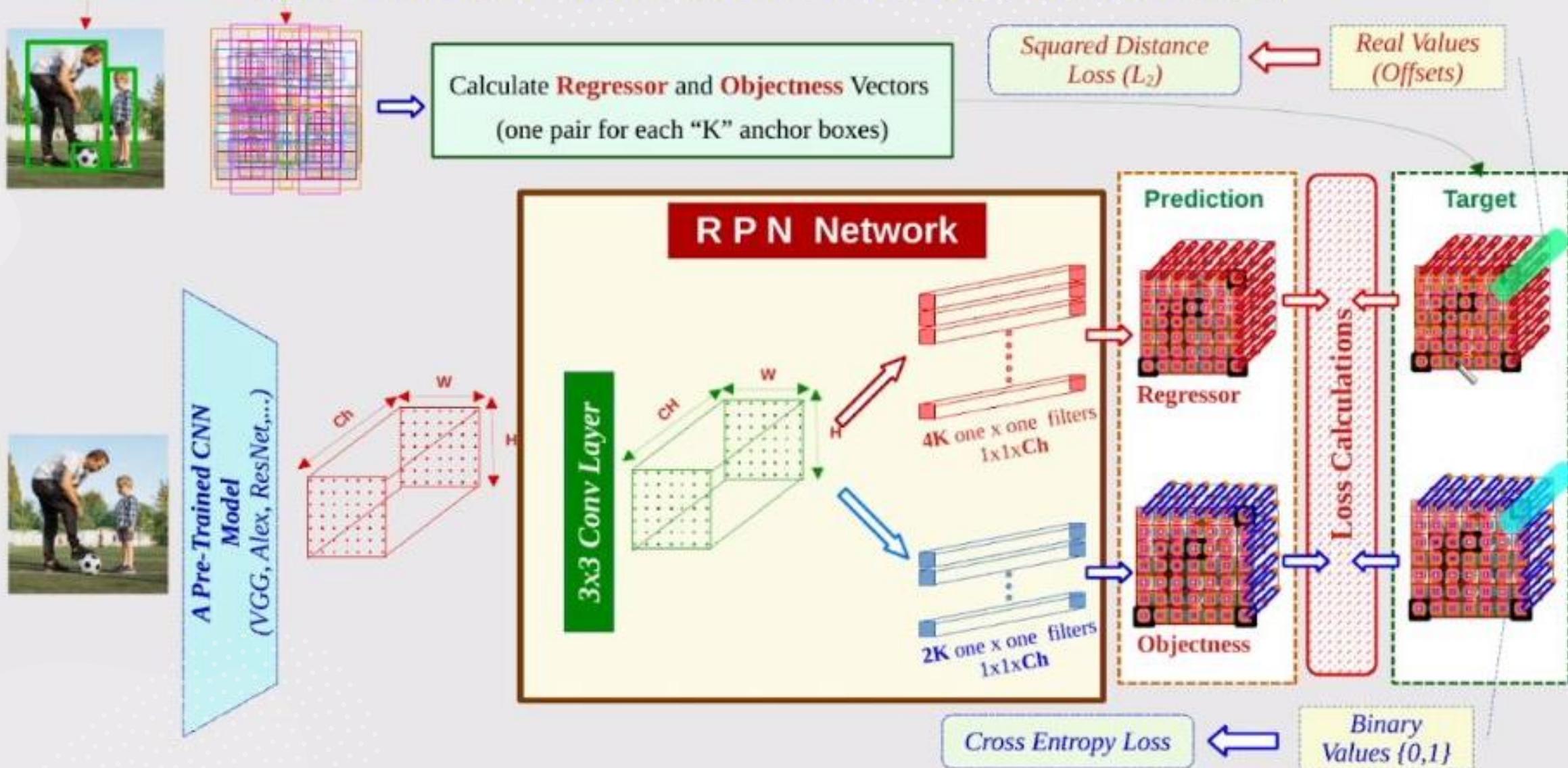
(The Full Picture – Generic Pre-trained CNN and K anchor boxes)



RPN Architecture

IOU

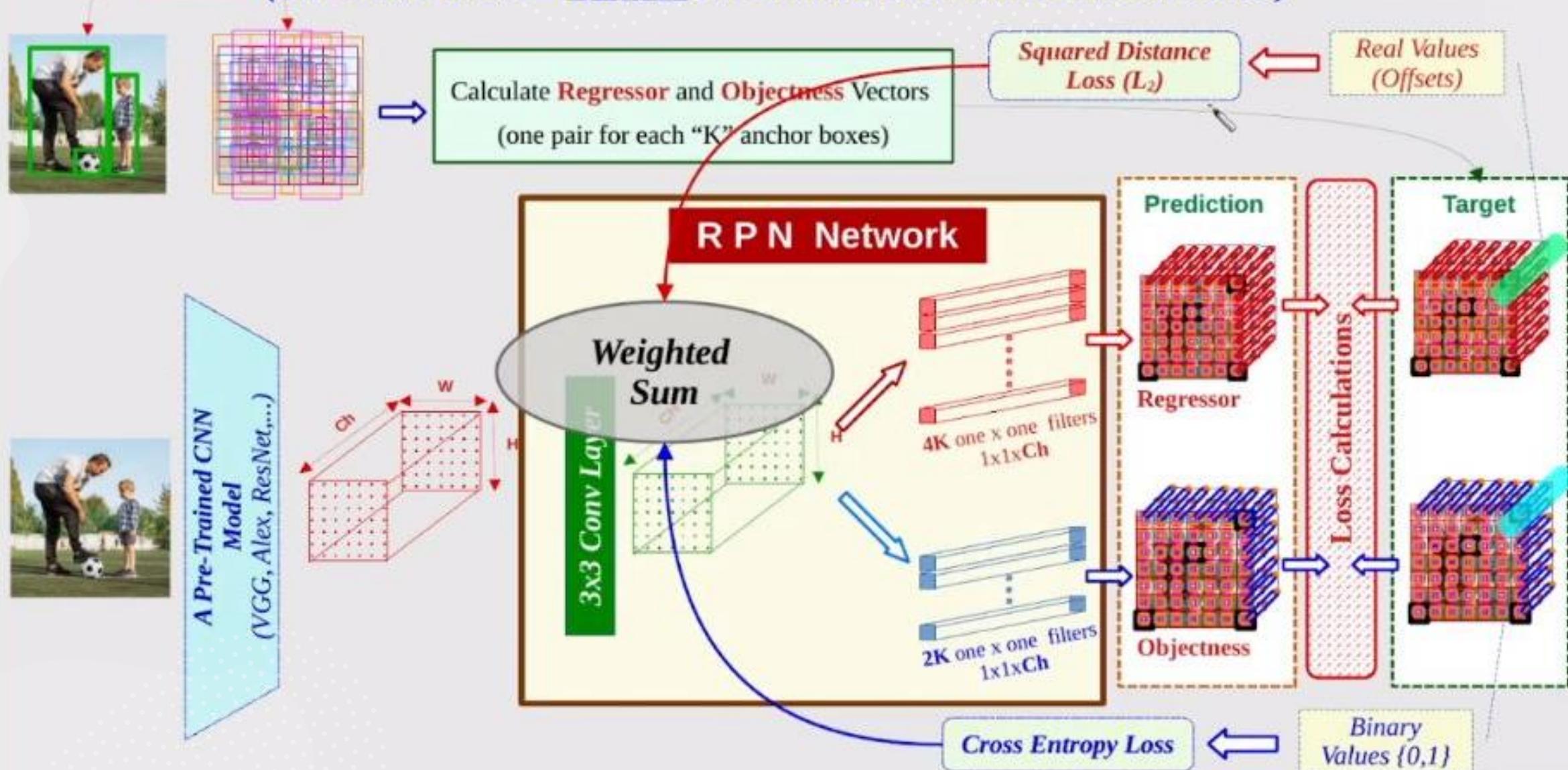
(The Full Picture – Generic Pre-trained CNN and K anchor boxes)



RPN Architecture

IOU

(The Full Picture – Generic Pre-trained CNN and K anchor boxes)



[1] Object Classification Loss (Object / Background)

$$\text{Object classification Loss} = \frac{1}{N_{cls}} \sum_{i=1}^{N_{cls}} L_{cls}(p_i^L, p_i^P)$$

Where

N_{cls} Number of Samples used to calculate classification loss

L_{cls} Classification Loss Function

p_i^L Probabilities Vector of target sample number (i)

{ 1 = Positive (Object) 0 = Negative (Background) }

p_i^P Predicted Probabilities Vector that anchor sample number (i)

{ Prob that it contains an object and it contains Background }

L_{cls} is log Loss (Cross Entropy Loss) since the target vector is one-hot (Probability)

$$L_{cls}(p^L, p^P) = -p^L \log p^P$$

[2] Box Regressor Loss

$$\text{Box Regressor Loss} = \frac{1}{N_{reg}} \sum_{i=1}^N p_i^t L_{reg}(d_i^L, d_i^P)$$

Where

N_{reg} Number of **Positive** Samples used to calculate regression loss

L_{reg} Regressor Loss Function

p_i^t Probability of **target** sample number (i) is Object

Summation over boxes containing **objects ONLY**

d_i^P offsets vector of **anchor** sample number (i) [Predicted]

d_i^L offsets vector of **target** sample number (i) [Label]

$$L_{reg}(d^L, d^P) = L_1(d_x^P - d_x^L) + L_1(d_y^P - d_y^L) + L_1(d_w^P - d_w^L) + L_1(d_h^P - d_h^L)$$

L₁ ???

Remember:
(for label vector)

$$d_x = (x_t - x_a) / x_a$$

$$d_y = (y_t - y_a) / y_a$$

$$d_w = \log(w_t / w_a)$$

$$d_h = \log(h_t / h_a)$$

To retrieve Target Parameters from predicted parameters

$$X_t = d_x x_a + x_a$$

$$Y_t = d_y * y_a + y_a$$

$$w_t = e^{(d_w)} * w_a$$

$$h_t = e^{(d_h)} * h_a$$

Smoothing Function (L_1)

Where L_1 is a *smoothing L_1* function:

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

+

For Small Values of X between {-1,1}

Output is between {-0.5 , 0.5} Loss is reduced

For Large Values of X $\text{Abs}(X) > 1$

output is almost the same (subtract 0.5 for function continuity)

Smoothing Function (L_1)

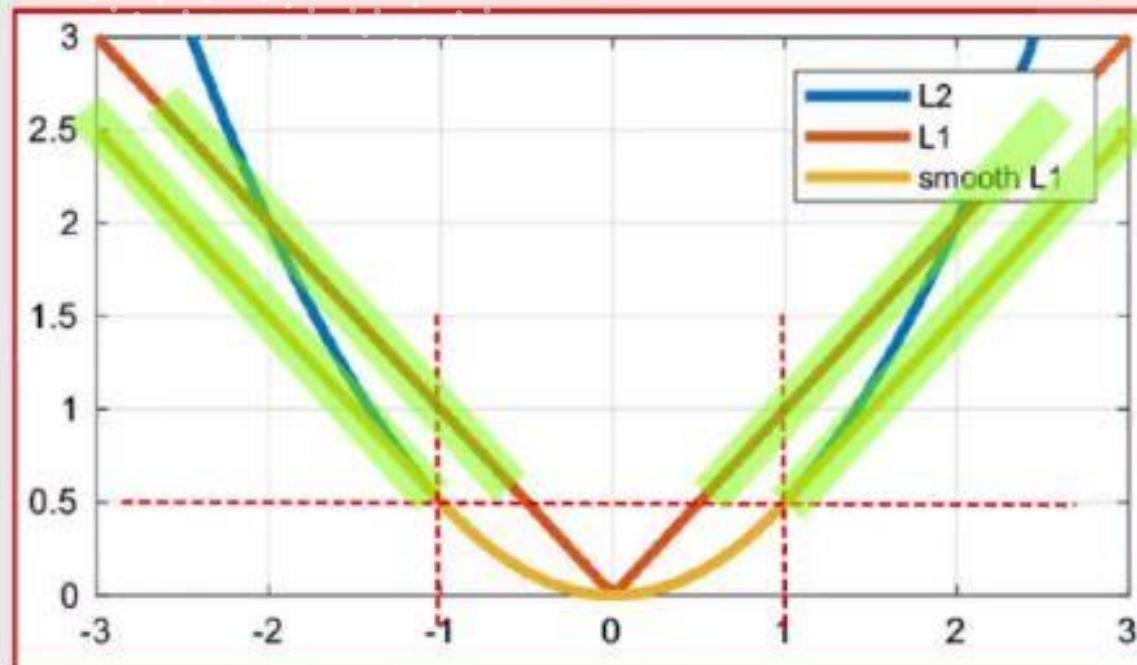
Where L_1 is a smoothing L_1 function:

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

Where:

L_1 is Absolute distance

L_2 is Squared distance



For Small Values of X between $\{-1, 1\}$

Output is between $\{-0.5, 0.5\}$ Loss is reduced

For Large Values of X $Abs(X) > 1$

output is almost the same (subtract 0.5 for function continuity)

Total Loss Function

Loss Function = Object classification Loss + λ^ Box Regressor Loss*

λ is a weight parameter

$$\text{Object classification Loss} = \frac{1}{N_{cls}} \sum_{i=1}^{N_{cls}} L_{cls}(p_i^L, p_i^P)$$

Where:

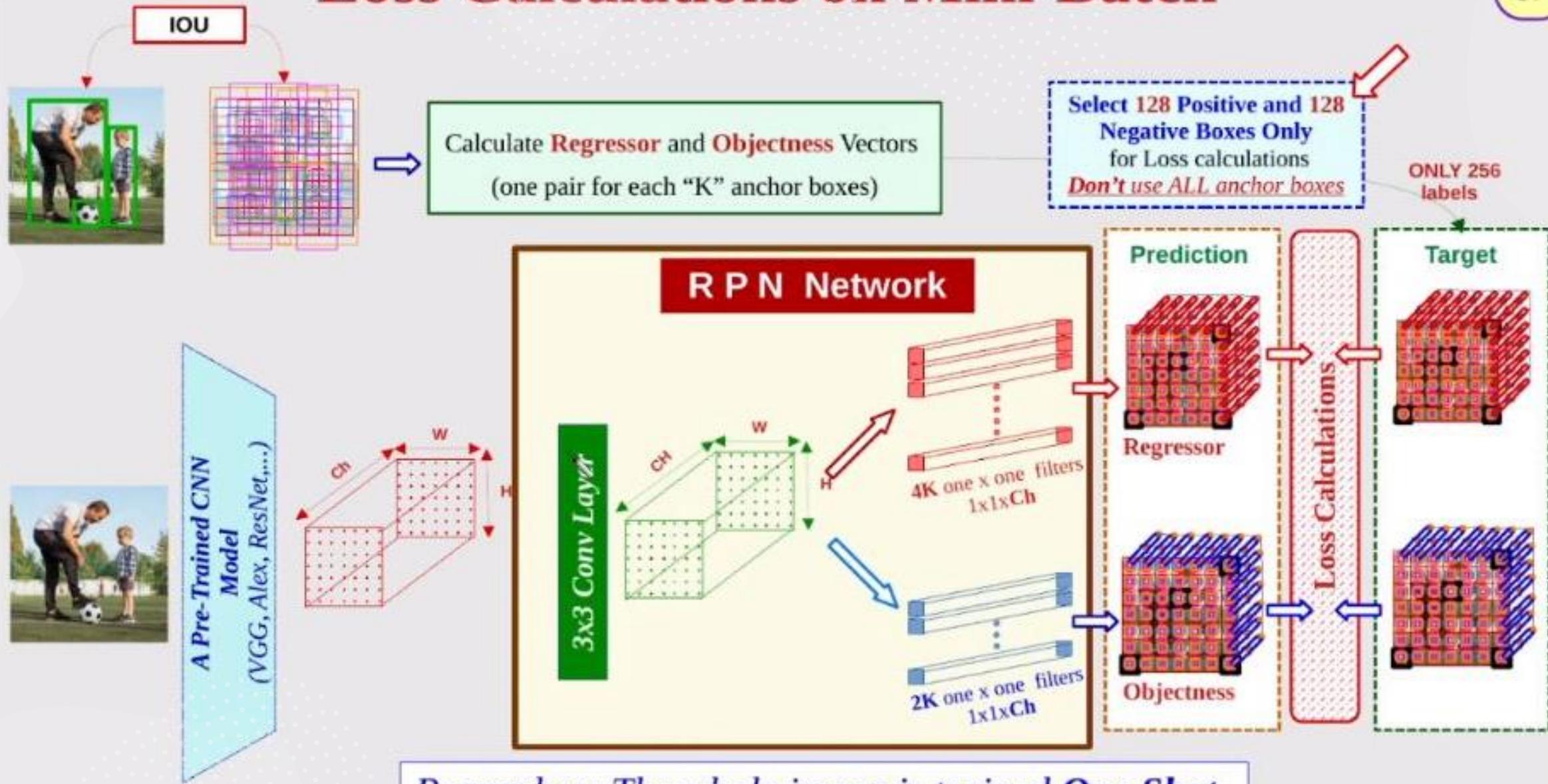
$$L_{cls}(p^L, p^P) = -p^L \log p^P$$

$$\text{Box Regressor Loss} = \frac{1}{N_{reg}} \sum_{i=1}^N p_i^t L_{reg}(d_i^L, d_i^P)$$

Where:

$$L_{reg}(d^L, d^P) = L_1(d_x^P - d_x^L) + L_1(d_y^P - d_y^L) + L_1(d_w^P - d_w^L) + L_1(d_h^P - d_h^L)$$

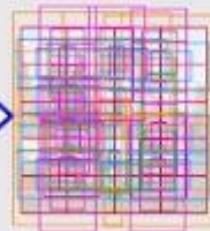
Loss Calculations on Mini-Batch



Final Region Proposals

using RPN and Non-Max Suppression

Obtain Region Proposals (from RPN) (for ALL Anchor Boxes)



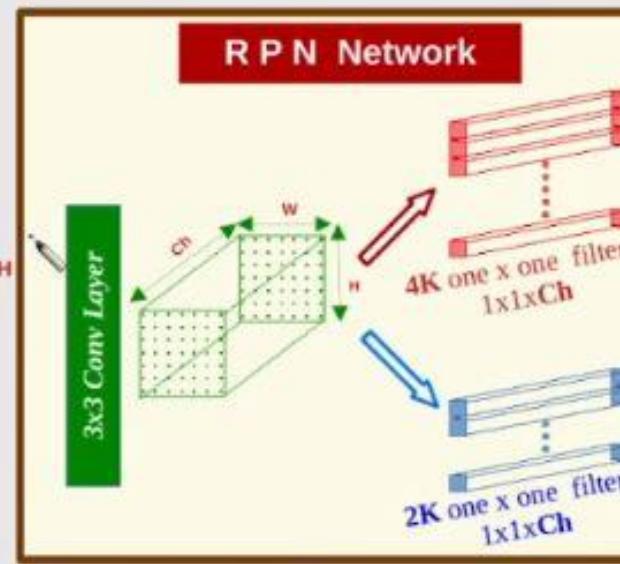
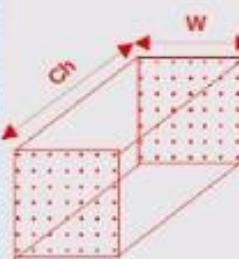
calculate parameters of anchor boxes
 (x_a, y_a, w_a, h_a) for All $W \times H \times K$ boxes
 (7x7*9) for VGG16 example

$$\begin{aligned} X_t &= d_x x_a + x_a \\ Y_t &= d_y * y_a + y_a \\ w_t &= e^{(d_w)} * w_a \\ h_t &= e^{(d_h)} * h_a \end{aligned}$$

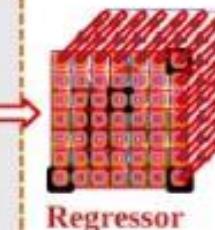
X_t
 Y_t
 W_t
 H_t



A Pre-Trained CNN Model
 (VGG, Alex, ResNet,...)



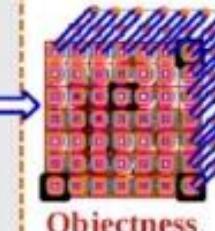
Prediction



Get Predicted regressor vectors for All Anchor boxes (d_x, d_y, d_w, d_h)

P_{obj}
 P_{bg}

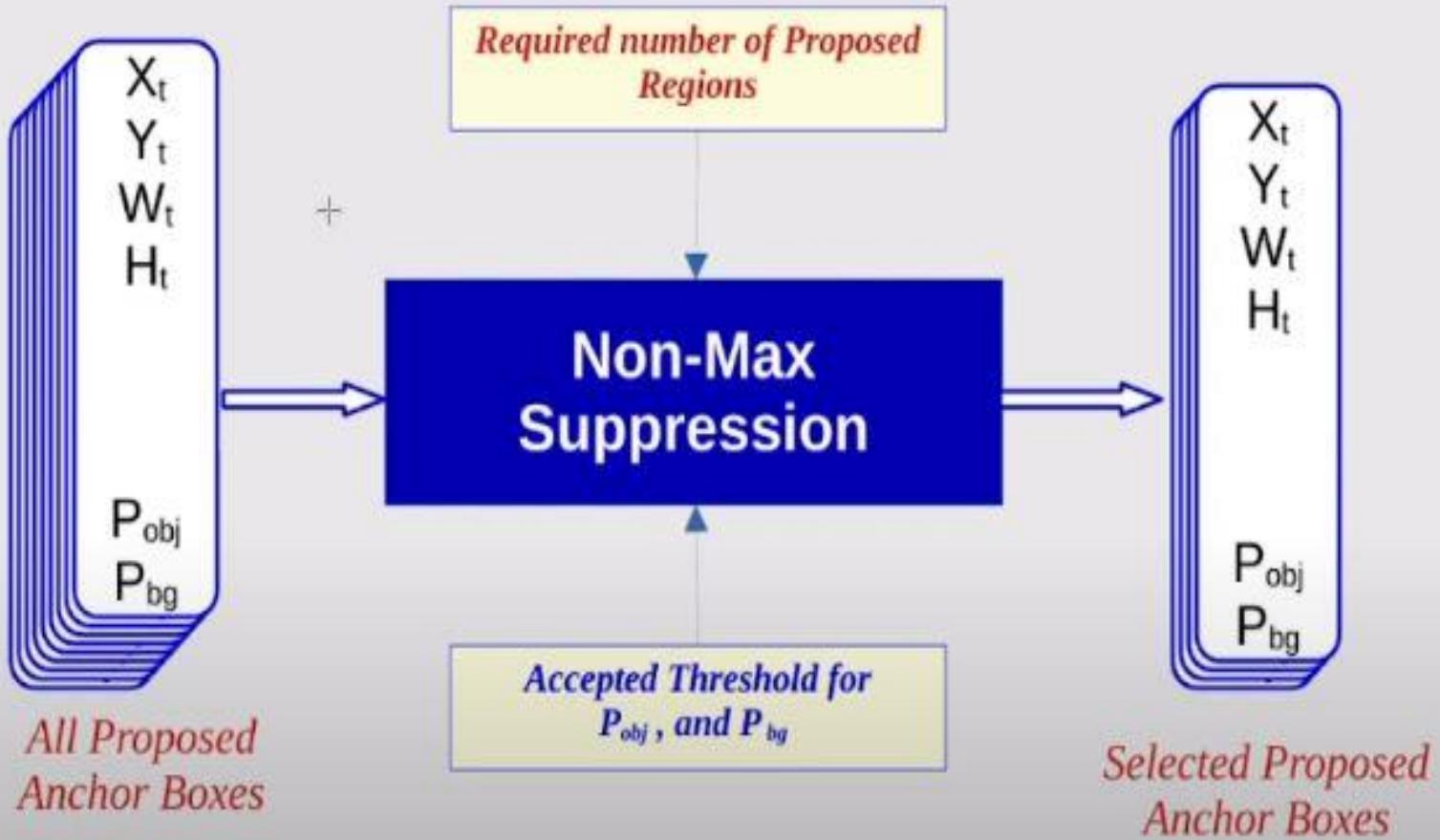
Regressor



Get Predicted Classification vectors for All Anchor boxes (p_{obj}, P_{bg})

Objectness

Filter RPN Region Proposals (Remove highly overlapped Anchor Boxes)



Faster R-CNN

FAST R-CNN + (*Region proposal Network*)

