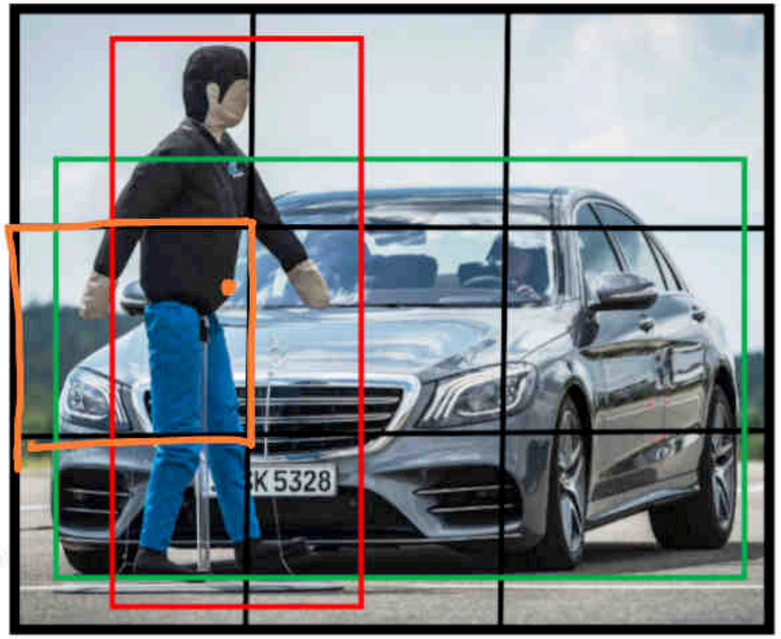
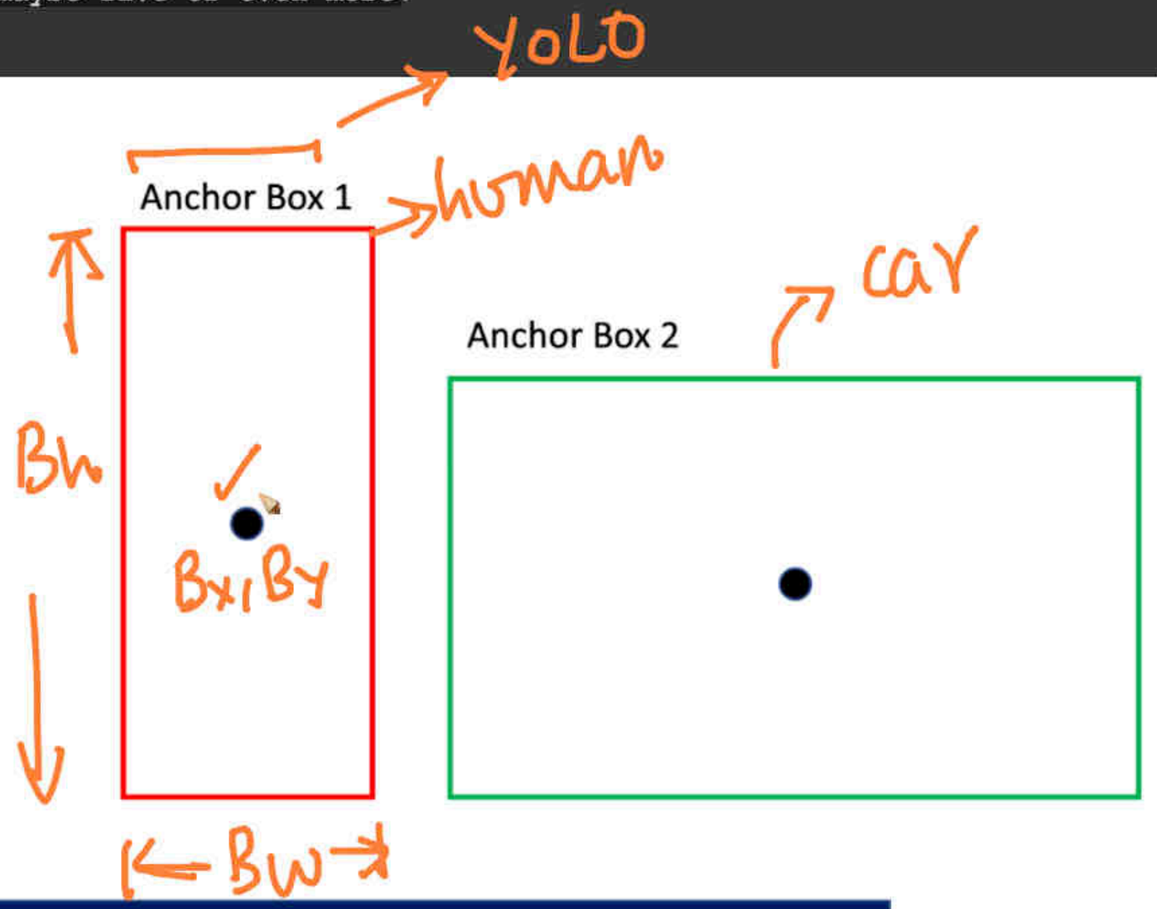
Agenda:

- Anchor boxes (concept)
- R-CNN, Fast RCNN, Faster RCNN (region-CNN) → 2stage
- YOLO - 1 stage → SOTA algo
  V1 ... V7

- And in general, you might use more anchor boxes, maybe five or even more.

Example: Overlapping Objects



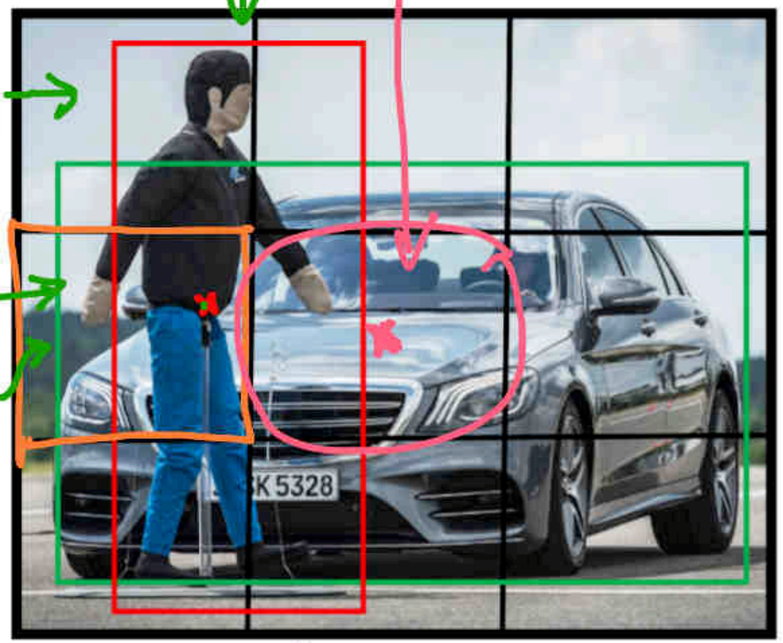


Output (Y) :

P_c	B_x	B_y	B_h	B_w	C_1	C_2	C_3	P_c	B_x	B_y	B_h	B_w	C_1	C_2	C_3
1	B_x	B_y	B_h	B_w	1	0	0	1	?	?	?	?	0	1	0

Example: Overlapping Objects

GRID → a way to represent
- 4 params of BBox
- prob of presence of any O.I
- label of the object in the BBox



Anchor Box 1

Anchor Box 2

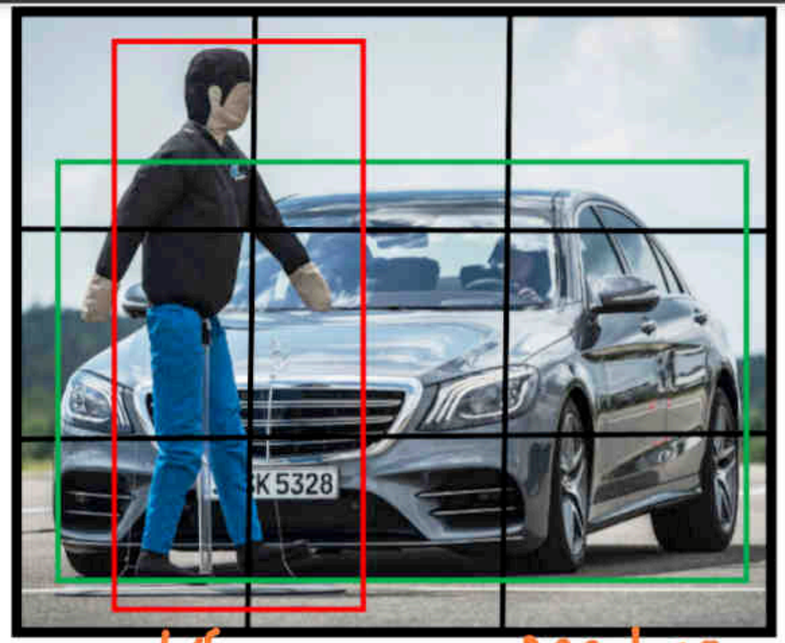
Bh

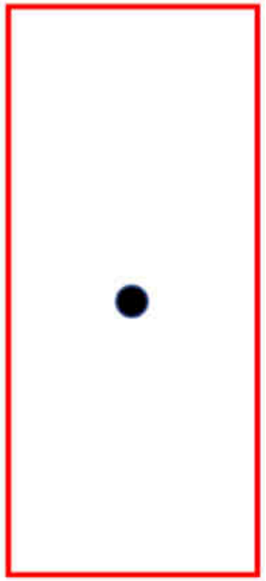
Bx, By

Bw

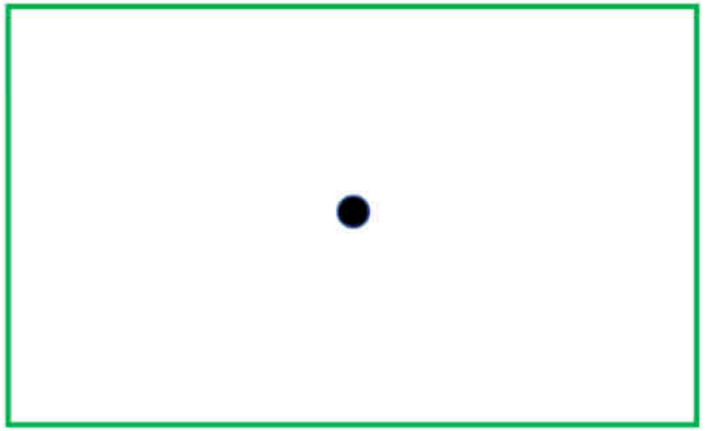
Output (Y) :

P_c	B_x	B_y	B_h	B_w	C_1	C_2	C_3	P_c	B_x	B_y	B_h	B_w	C_1	C_2	C_3
1	B_x	B_y	B_h	B_w	1	0	0	1	B_x	B_y	B_h	B_w	0	1	0





Anchor Box 2



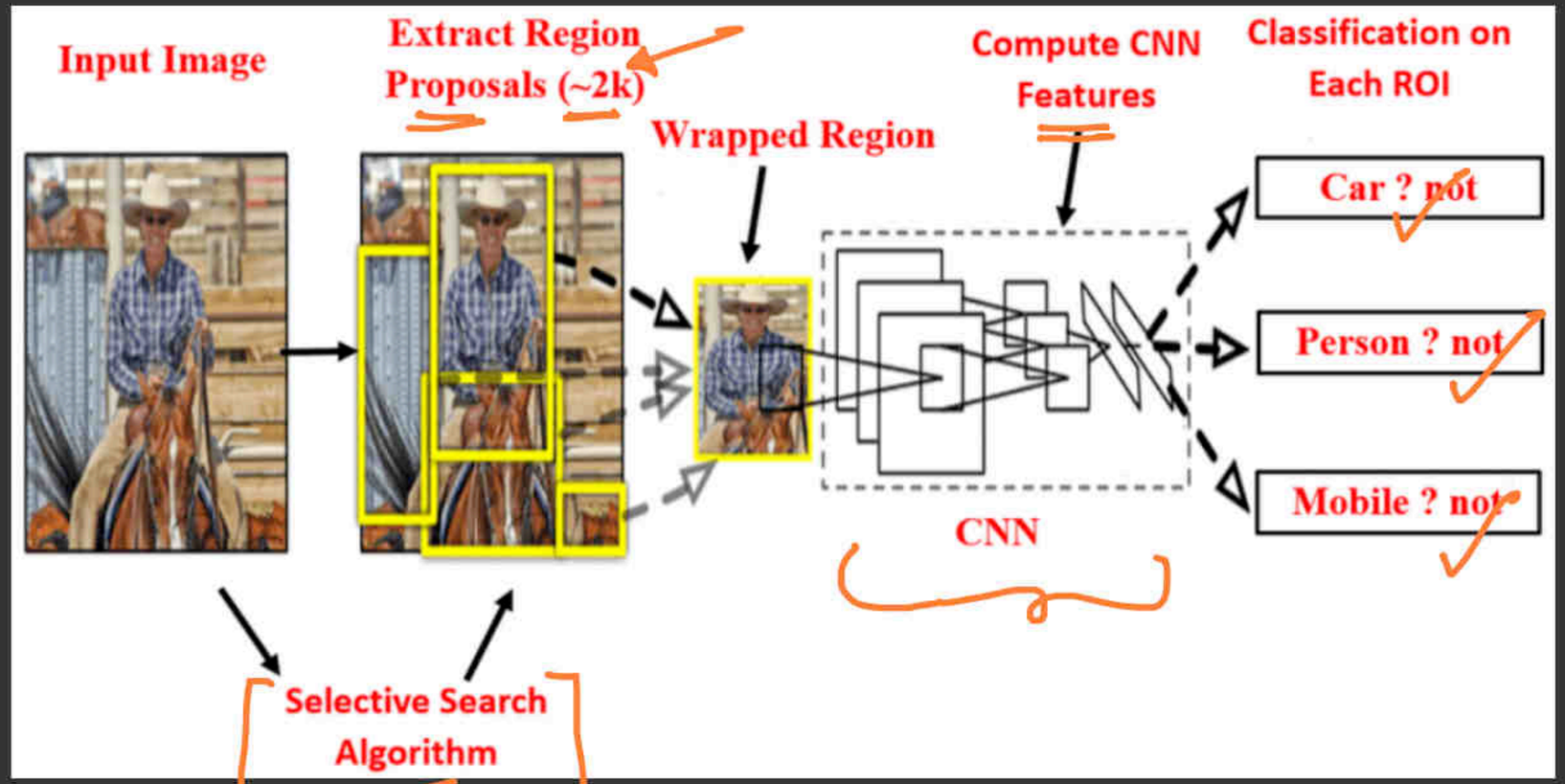
log loss regloss CE

Output (Y) :

P_c	B_x	B_y	B_h	B_w	C_1	C_2	C_3	P_c	B_x	B_y	B_h	B_w	C_1	C_2	C_3
1	B_x	B_y	B_h	B_w	1	0	0	1	B_x	B_y	B_h	B_w	0	1	0

- In the above image Mid point of both `Person` and `car` lies in the same Grid Cell.
- We have already Pre-defined two bounding b

2. Then at second stage, the Regions proposed by first stage are classified into different classes.



Nov 2013

2 stages

→ external algo

↓

regions

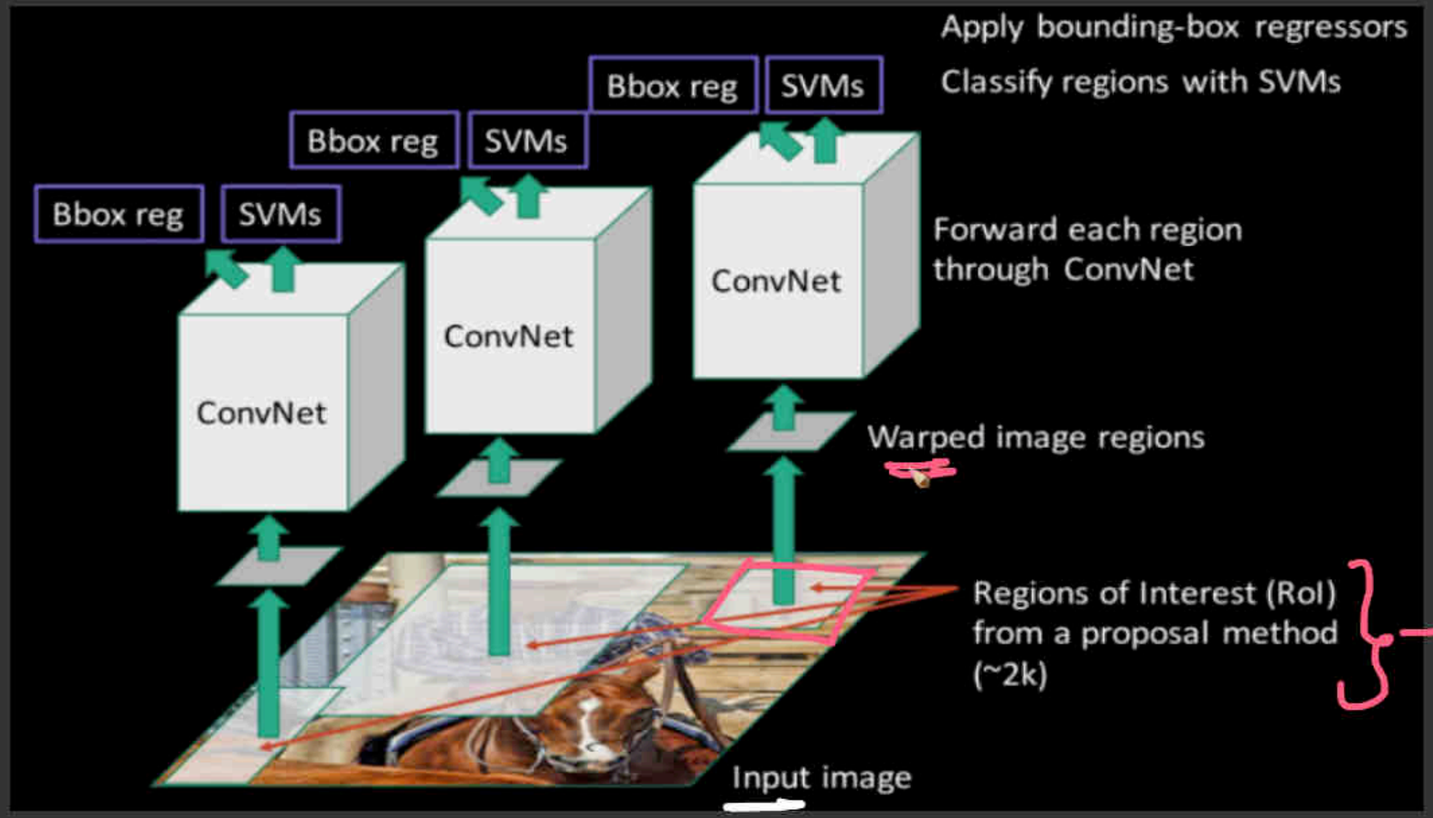
→ ∀ region CNN → feature map

RCNN: Rich feature hierarchies for accurate object detection

https://arxiv.org/abs/1311.2524

RCNN: Rich feature hierarchies for accurate object detection

https://arxiv.org/abs/1311.2524

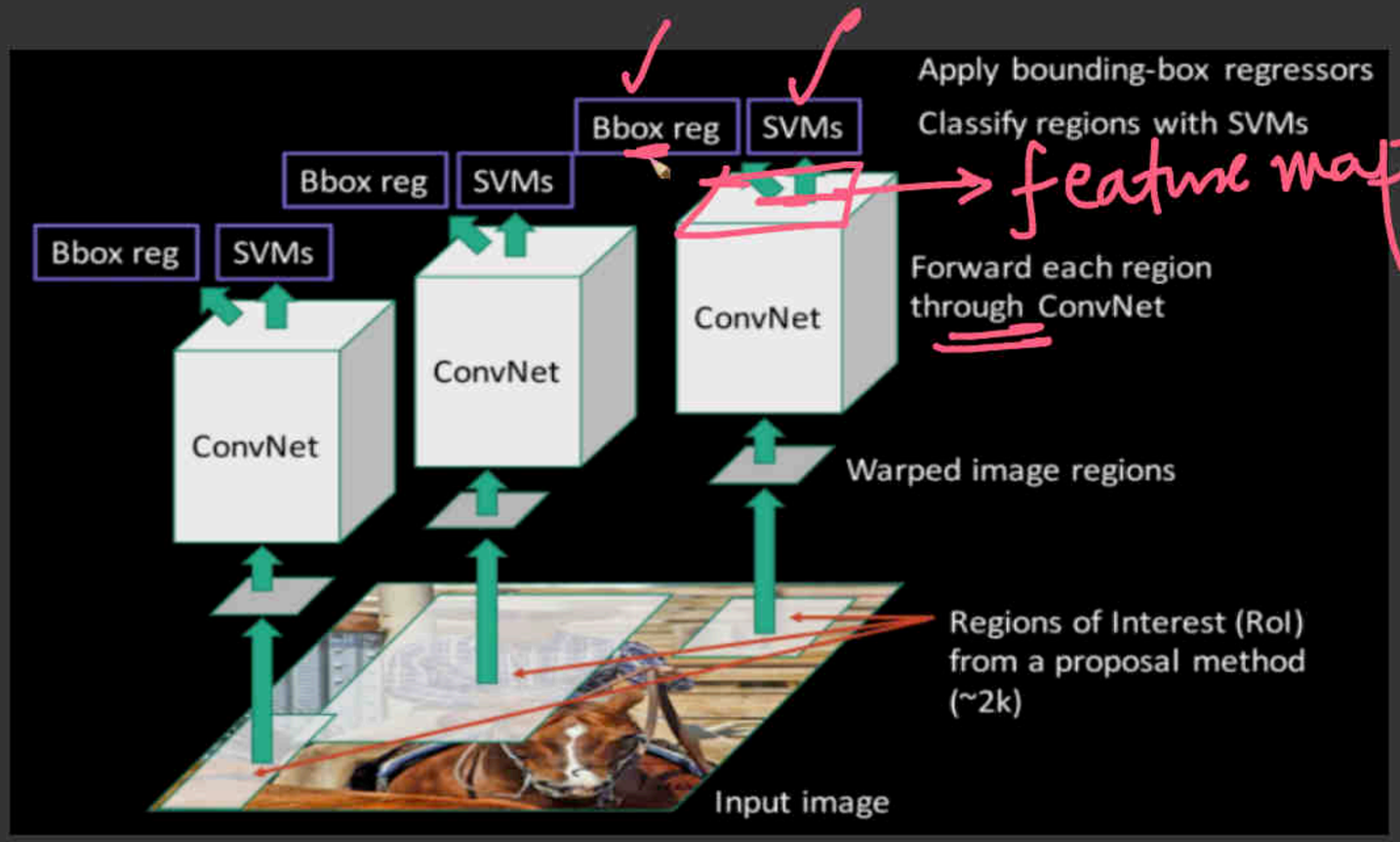


hierarchical clustering (sel. search)

An overview of the R-CNN algorithm can be found in above figure, which can be dividing into following steps:

RCNN: Rich feature hierarchies for accurate object detection

https://arxiv.org/abs/1311.2524

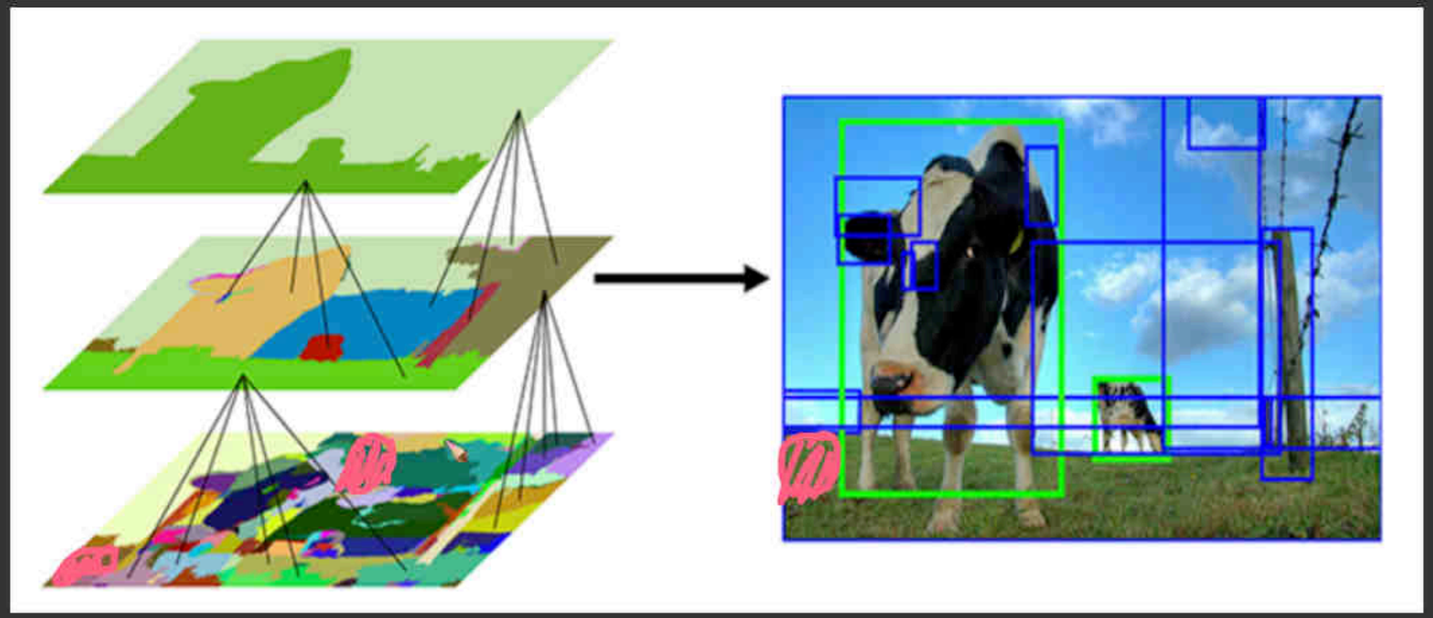


An overview of the R-CNN algorithm can be found in above figure, which can be dividing into following steps:

2. Group adjacent segments based on similarity
3. Go to step 1 until a predefined number of bounding boxes are reached(typically 2000)

Advantage: Selective Search is far more computationally efficient than exhaustively computing image pyramids and sliding windows

https://learnopencv.com/selective-search-for-object-detection-cpp-python/



Is it possible to obtain end-to-end deep learning-based object detection?

(a) (b)

Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

whose power of discovering parts or objects is left unevaluated. In this work, we use multiple complementary strategies to deal with as many image conditions as possible. We include the locations generated using [3] in our evaluation.

2.3 Other Sampling

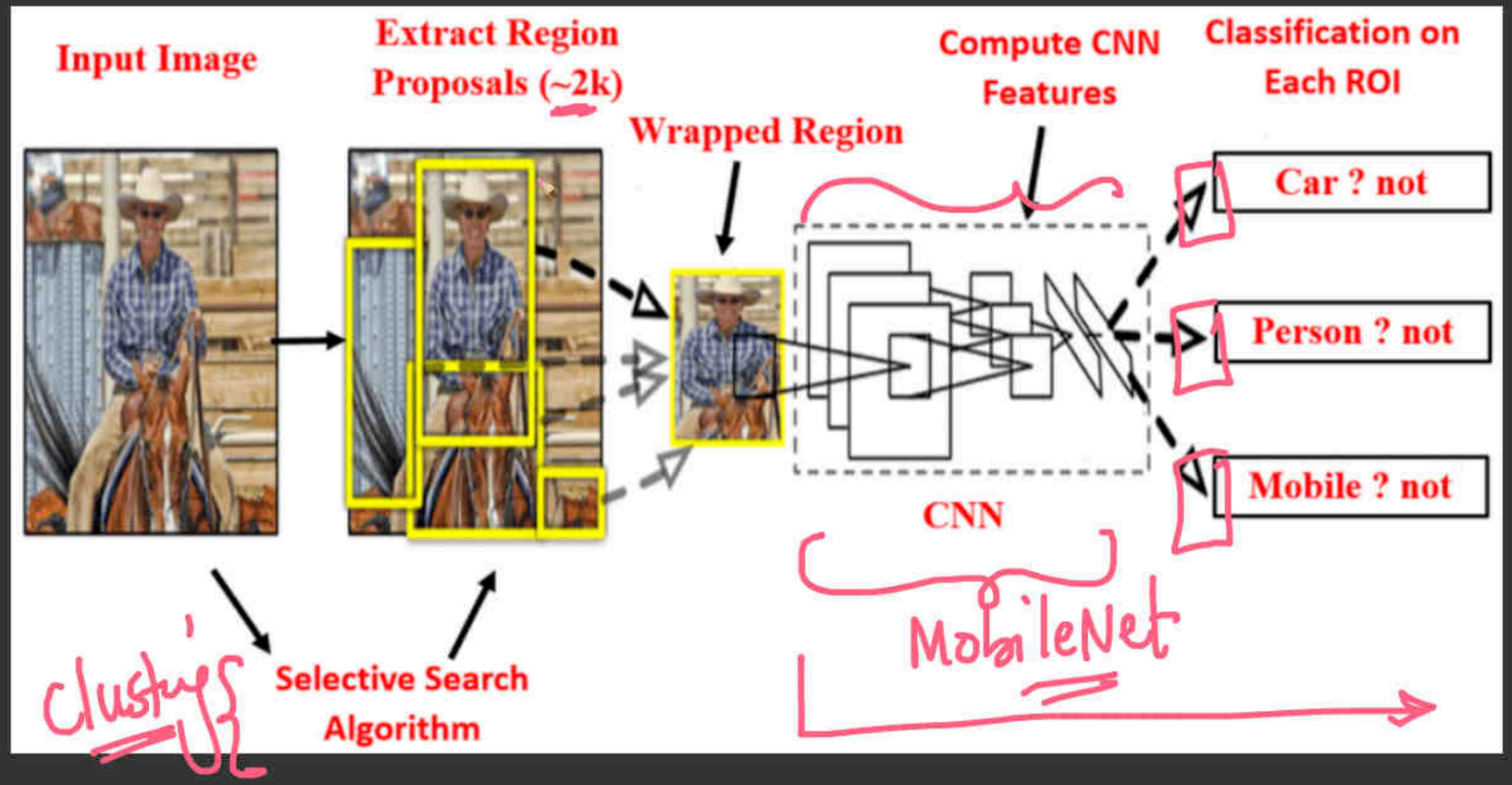
Alexe *et al.* [2] address the problem of the large sampling space

3 Selective Search

In this section we detail our selective search algorithm for object recognition and present a variety of diversification strategies to deal with as many image conditions as possible. A selective search algorithm is subject to the following design considerations:

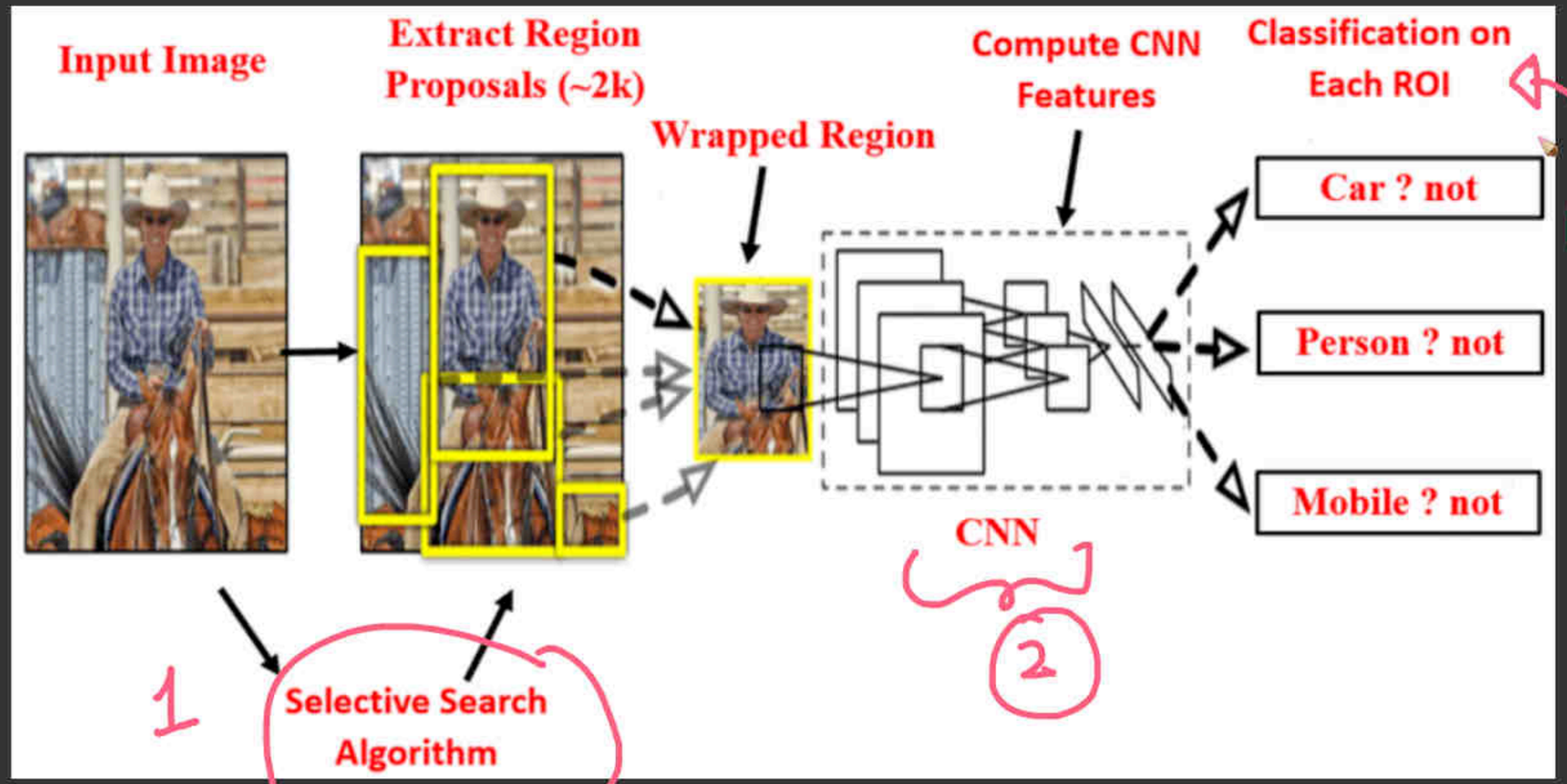
Capture All Scales. Objects can occur at any scale within the image. Unlike clear boundaries search all object scales have to be taken into account, as illustrated in Figure

1. At first stage, Propose some Regions from the image where ther is a chance of Object present.
2. Then at second stage, the Regions proposed by first stage are classified into different classes.



RCNN: Rich feature hierarchies for accurate object detection

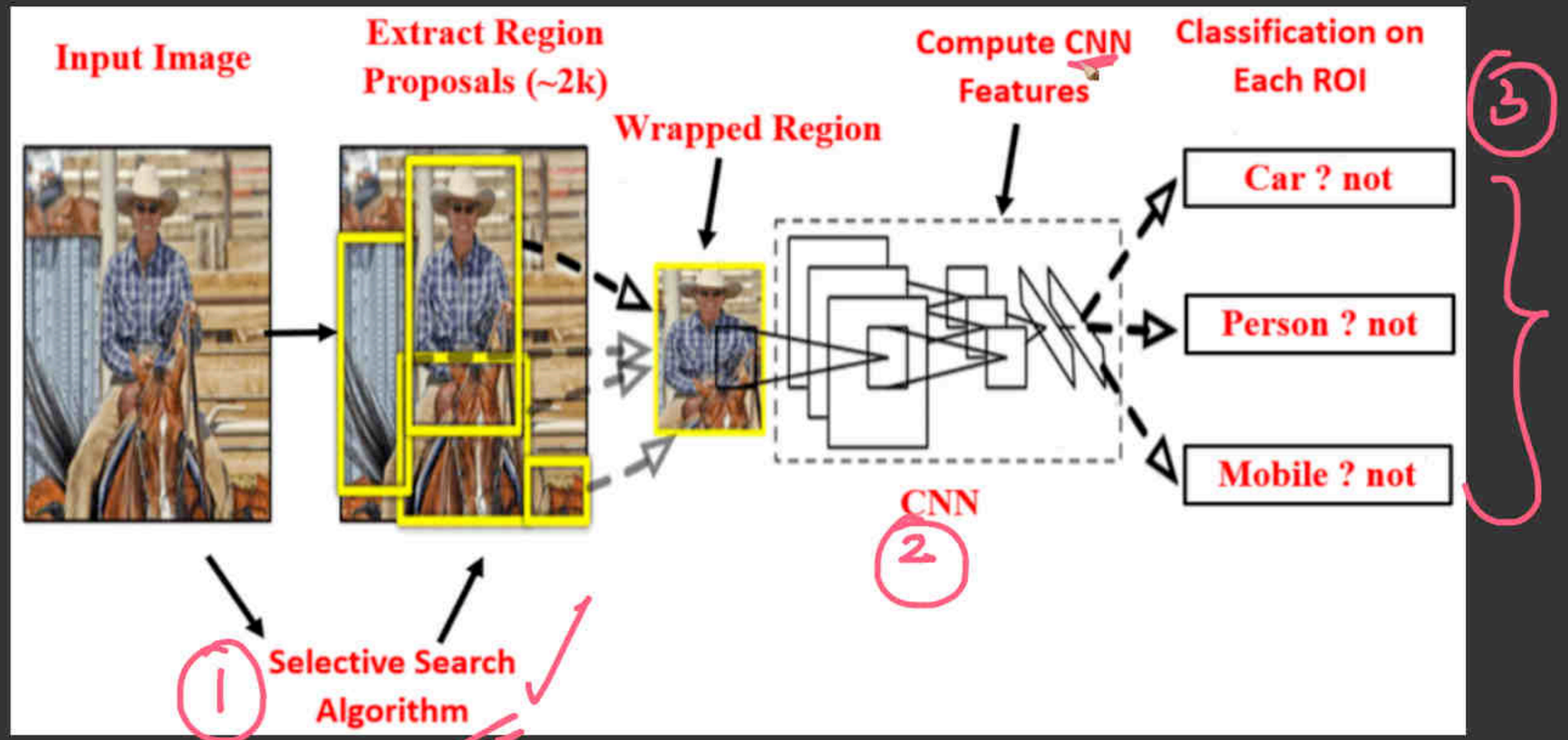
https://arxiv.org/abs/1311.2524



RCNN: Rich feature hierarchies for accurate object detection

https://arxiv.org/abs/1311.2524

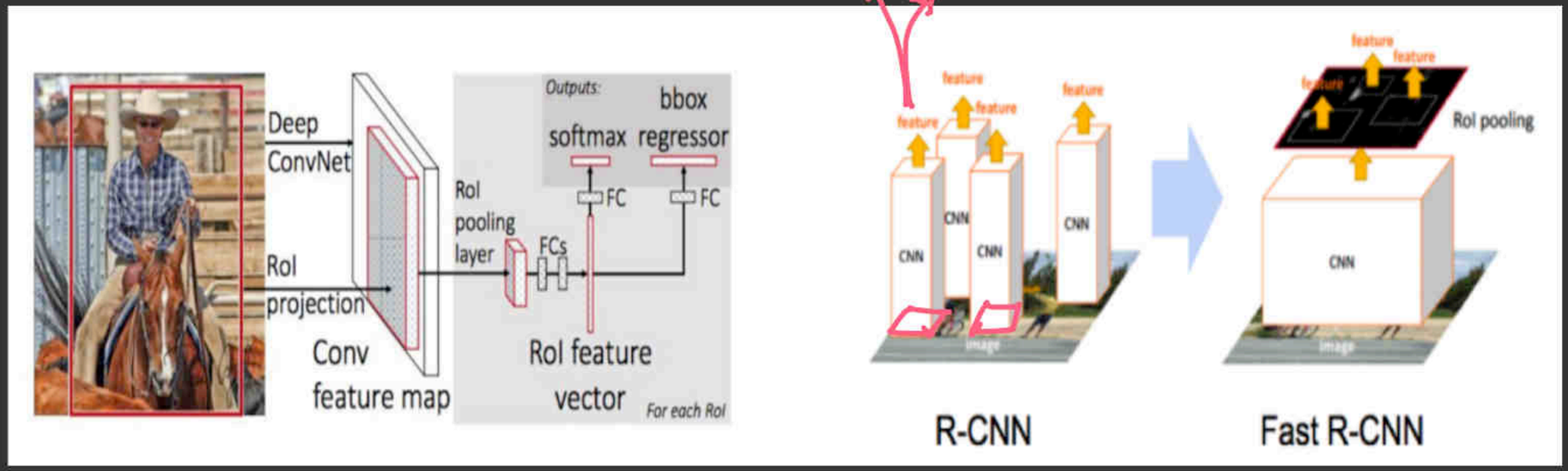
2. Then at second stage, the Regions proposed by first stage are classified into different classes.



RCNN: Rich feature hierarchies for accurate object detection

https://arxiv.org/abs/1311.2524

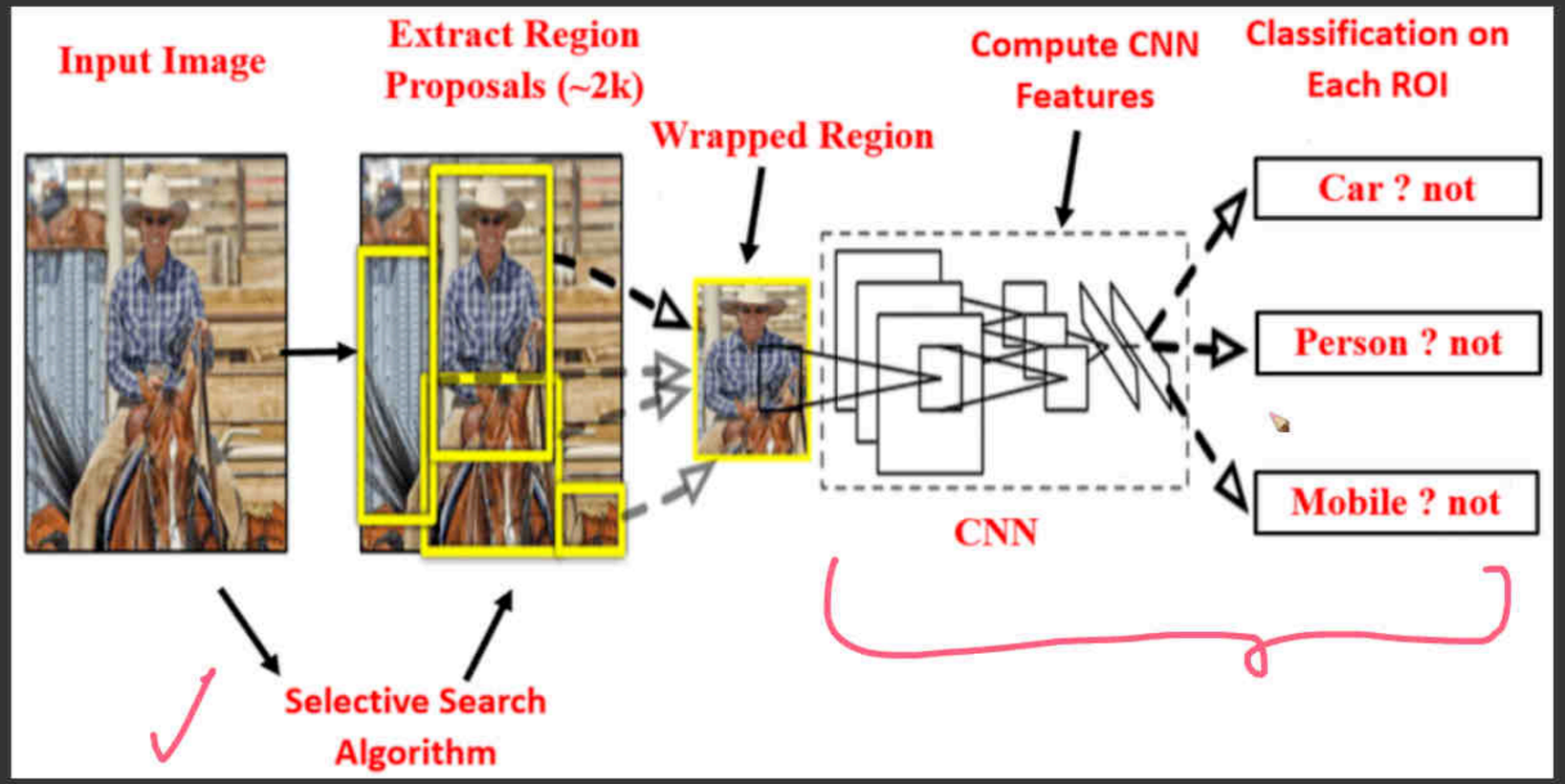
Fast R-CNN: https://arxiv.org/abs/1504.08083



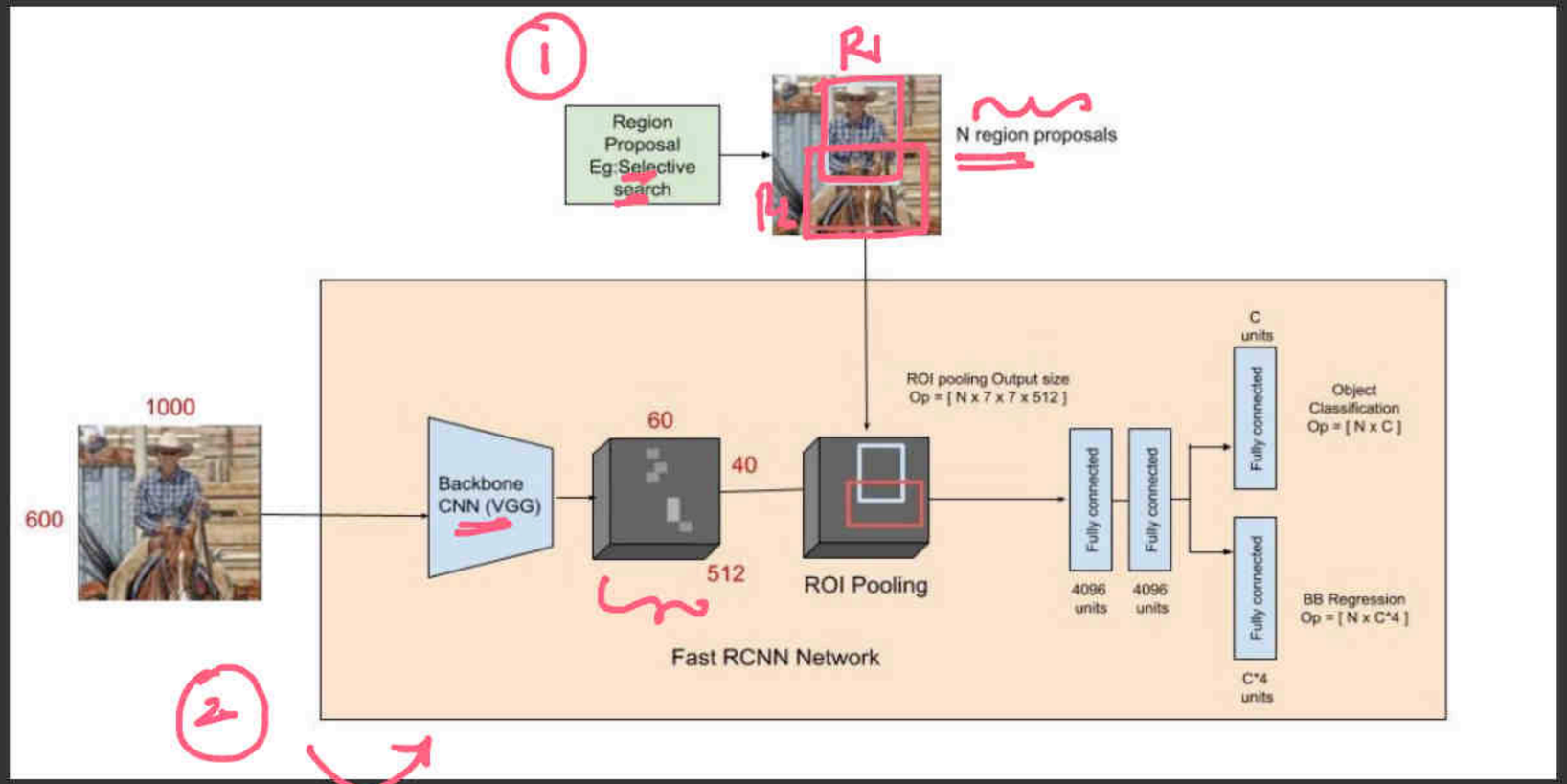
- In the **Fast R-CNN architecture**, we still use **Selective Search to obtain our proposals**, but now we apply **ROI Pooling by extracting a fixed-size window from the feature map** and `using these features to obtain the final class label and bounding box`.

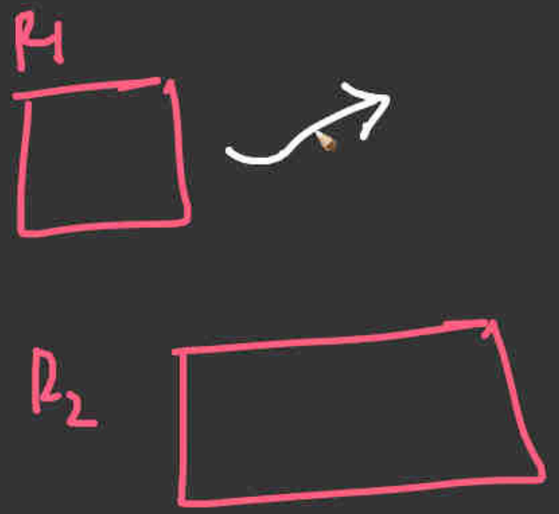
Let's see the Fast R-CNN Architecture in detail

1. At first stage, Propose some Regions from the image where ther is a chance of Object present.
2. Then at second stage, the Regions proposed by first stage are classified into different classes.



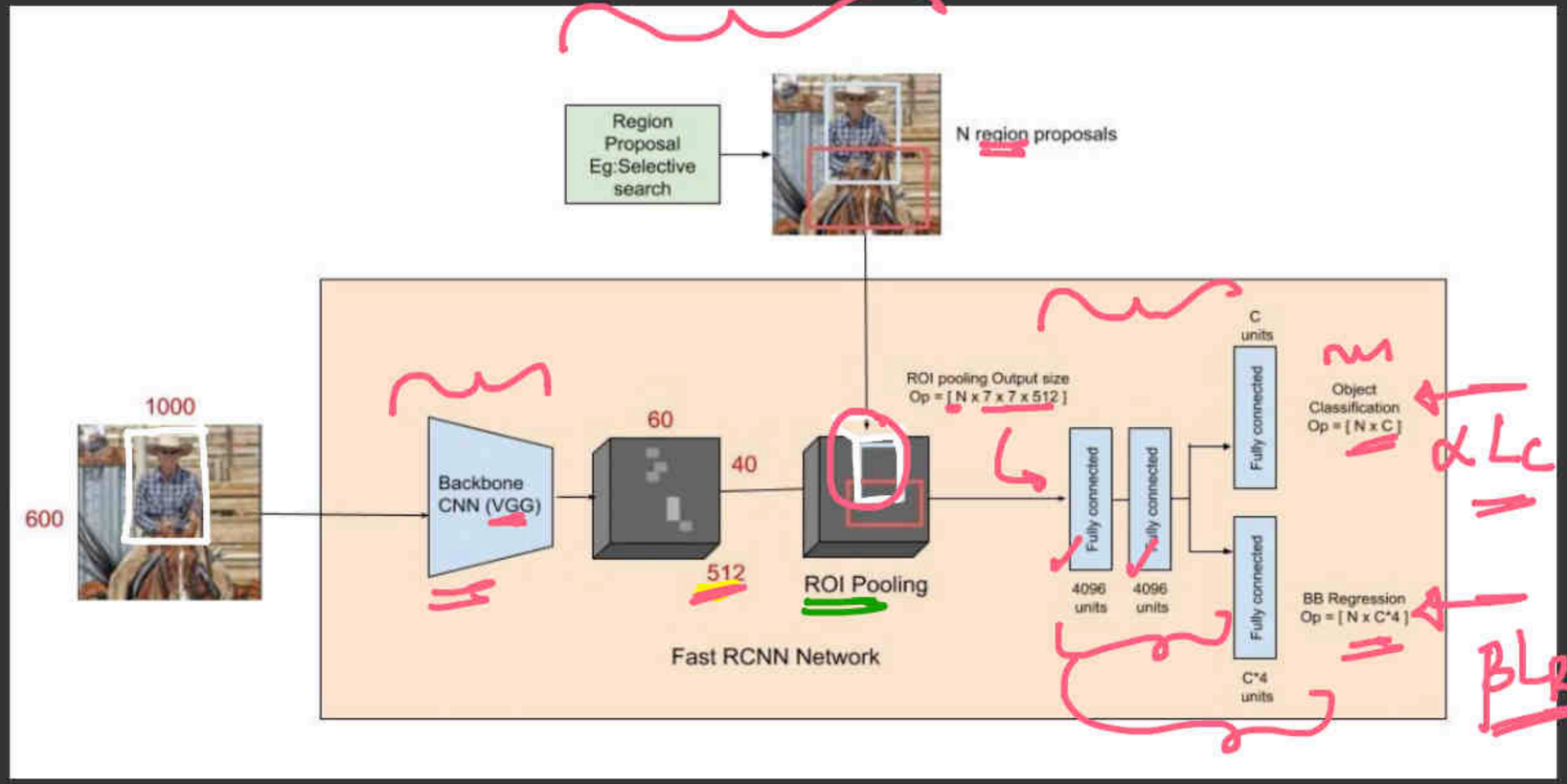
RCNN: Rich feature hierarchies for accurate object detection

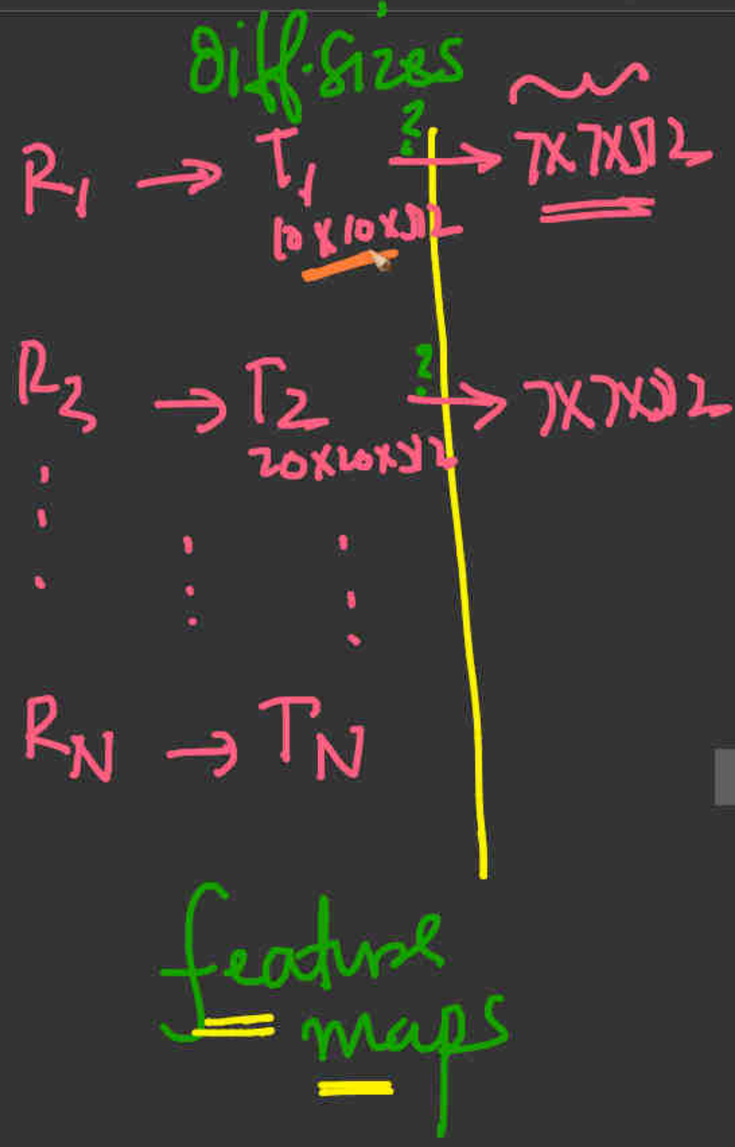




The primary benefit here is that the network is now, effectively, end-to-end trainable:

1. `We input an image and associated ground-truth bounding boxes`.
2. **Extract the feature map**
3. Apply **ROI pooling** and obtain **the ROI feature vector**.
4. And finally use two sets of fully-connected l

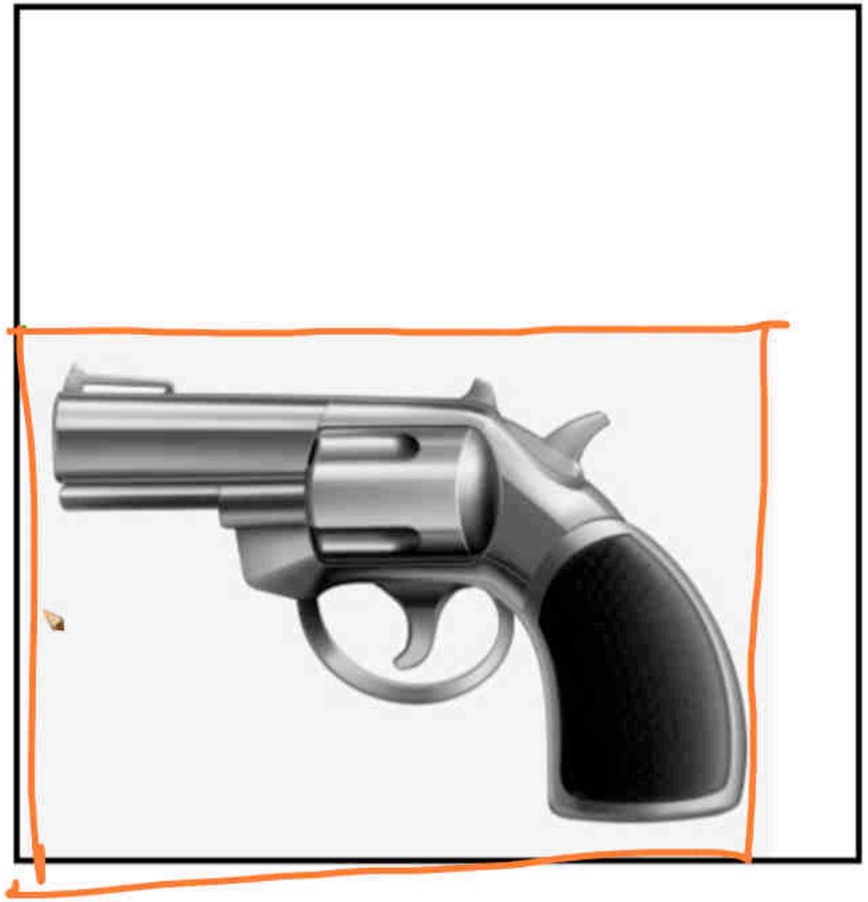




The primary benefit here is that the network is now, effectively, end-to-end trainable:

1. `We input an image and associated ground-truth bounding boxes.`
2. **Extract the feature map**
3. Apply **ROI pooling** and obtain **the ROI feature vector**.
4. And finally use two sets of fully-connected l

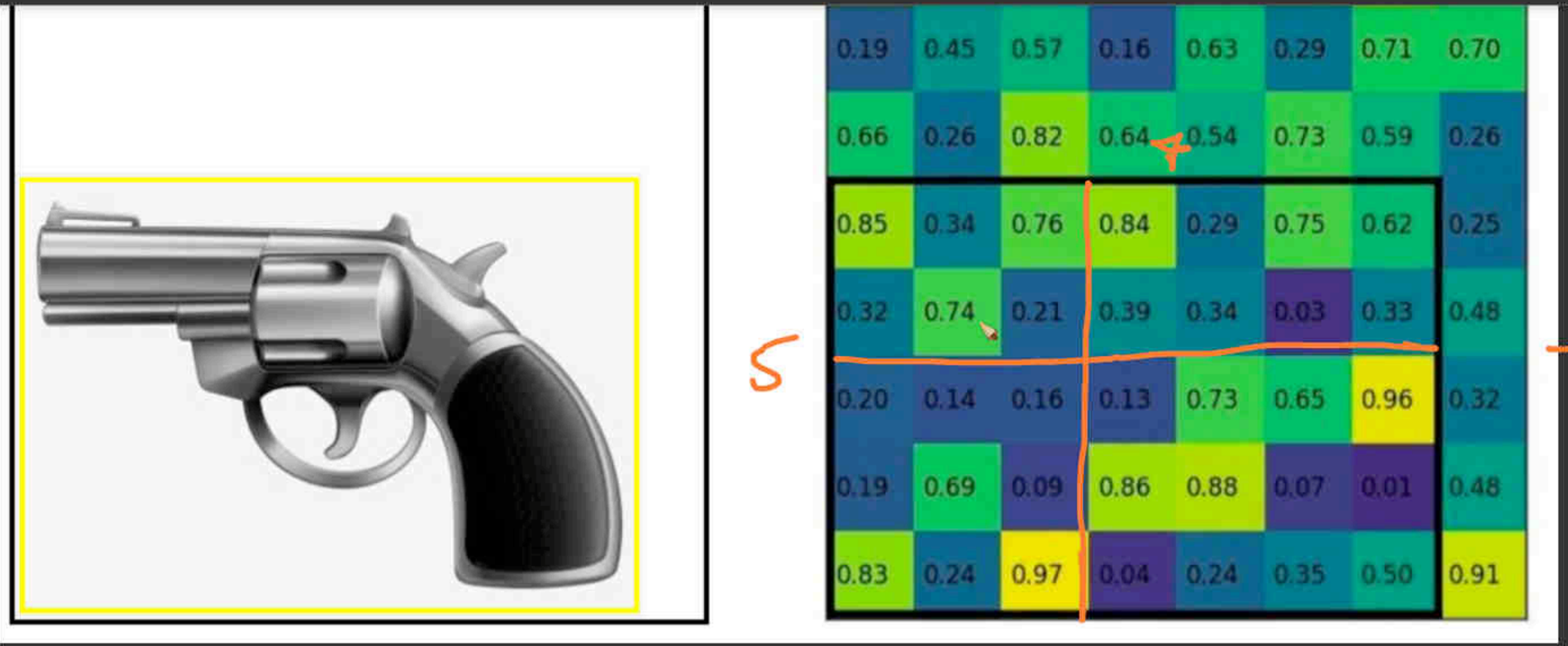
Input Image



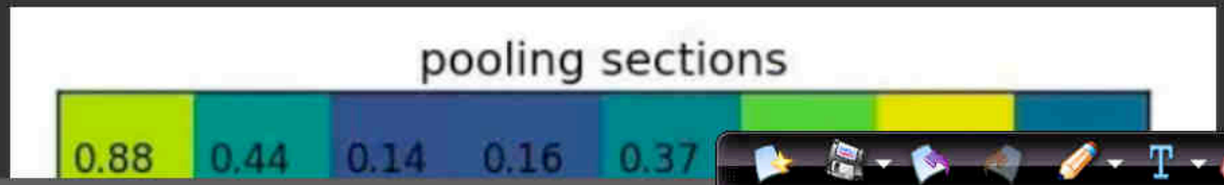
Input feature map

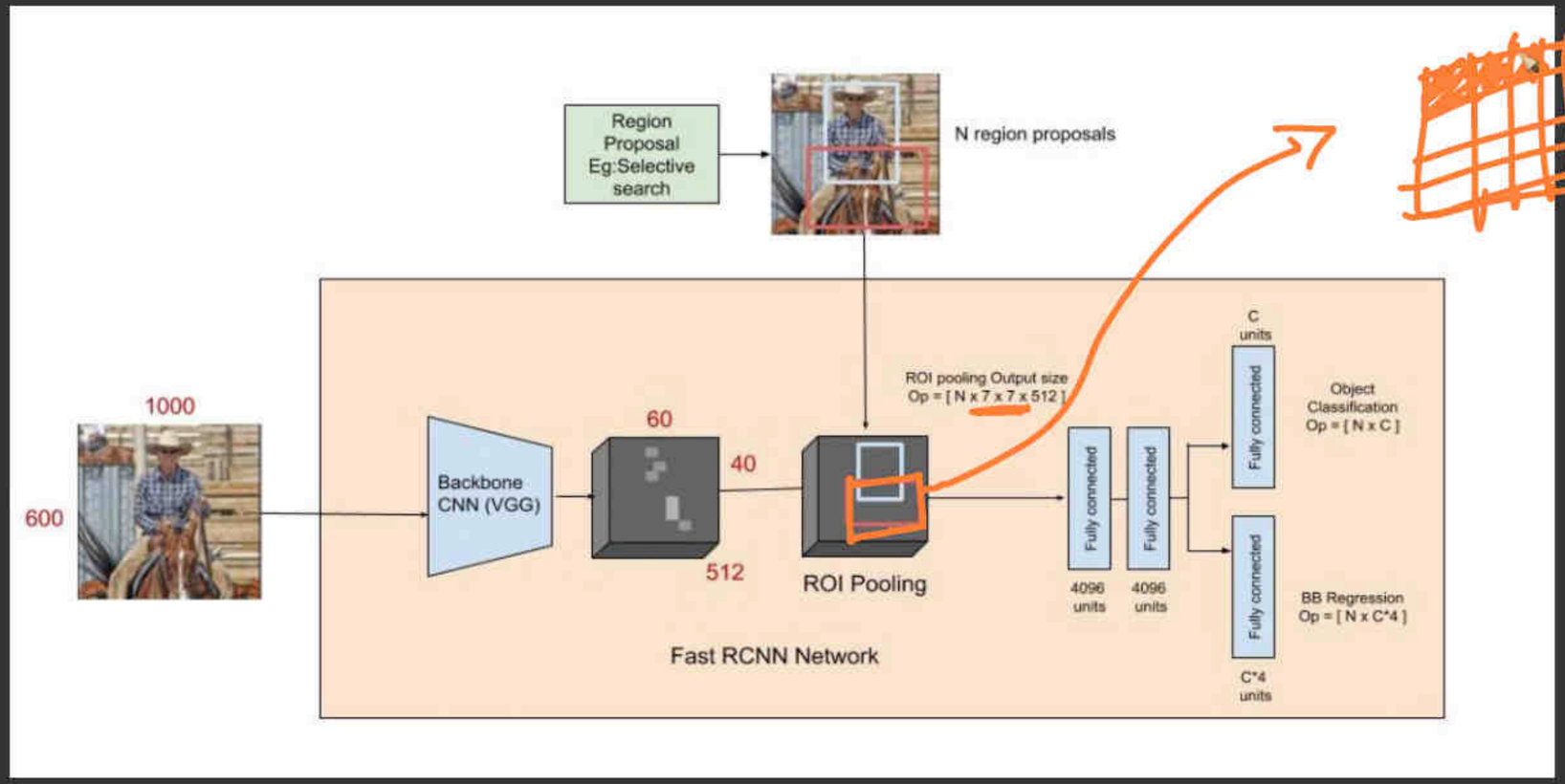
0.88	0.44	0.14	0.16	0.37	0.77	0.96	0.27
0.19	0.45	0.57	0.16	0.63	0.29	0.71	0.70
0.66	0.26	0.82	0.64	0.54	0.73	0.59	0.26
0.85	0.34	0.76	0.84	0.29	0.75	0.62	0.25
0.32	0.74	0.21	0.39	0.34	0.03	0.33	0.48
0.20	0.14	0.16	0.13	0.73	0.65	0.96	0.32
0.19	0.69	0.09	0.86	0.88	0.07	0.01	0.48
0.83	0.24	0.97	0.04	0.24	0.35	0.50	0.91

To Include: Corresponding GUN Image



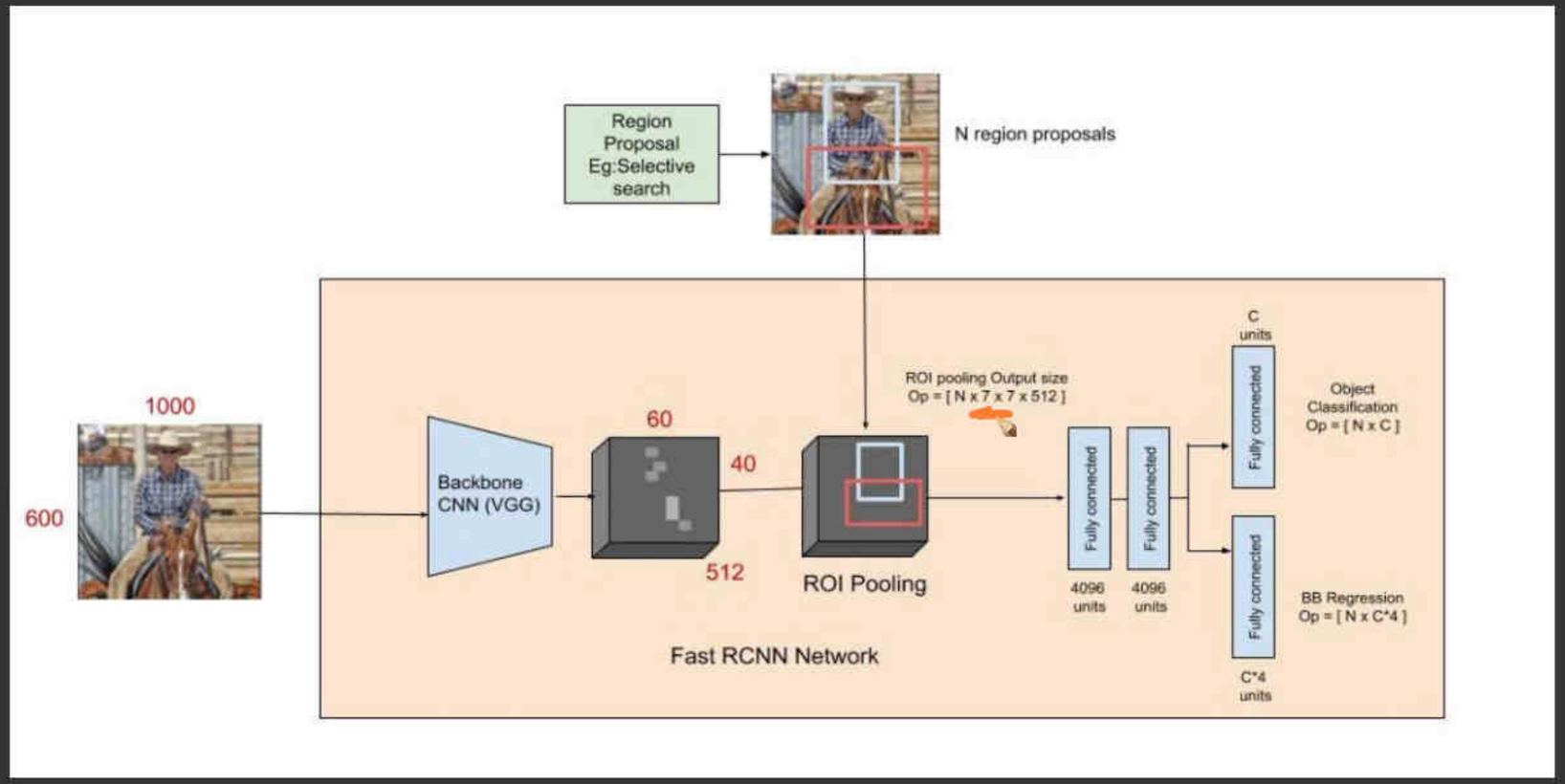
Normally, there'd be multiple feature maps and multiple proposals for each of them, but we're keeping things simple for the example. By dividing it into (2×2) sections (because the output size is 2×2) we get:





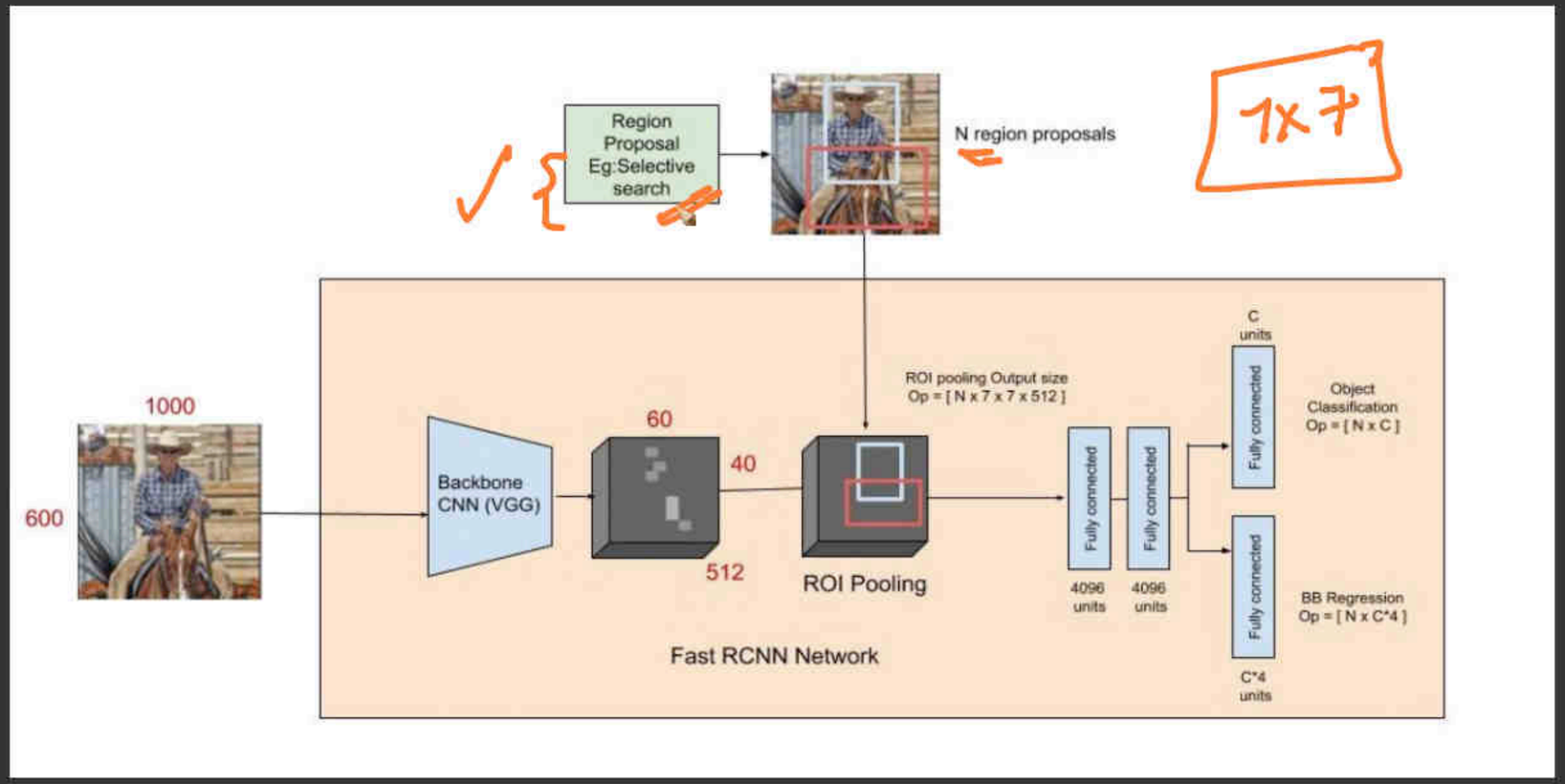
The primary benefit here is that the network is now, effectively, end-to-end trainable:

1. `We input an image and associated ground-truth bounding boxes.`
2. **Extract the feature map**
3. Apply **ROI pooling** and obtain **the ROI feature vector**.
4. And finally use two sets of fully-connected l



The primary benefit here is that the network is now, effectively, end-to-end trainable:

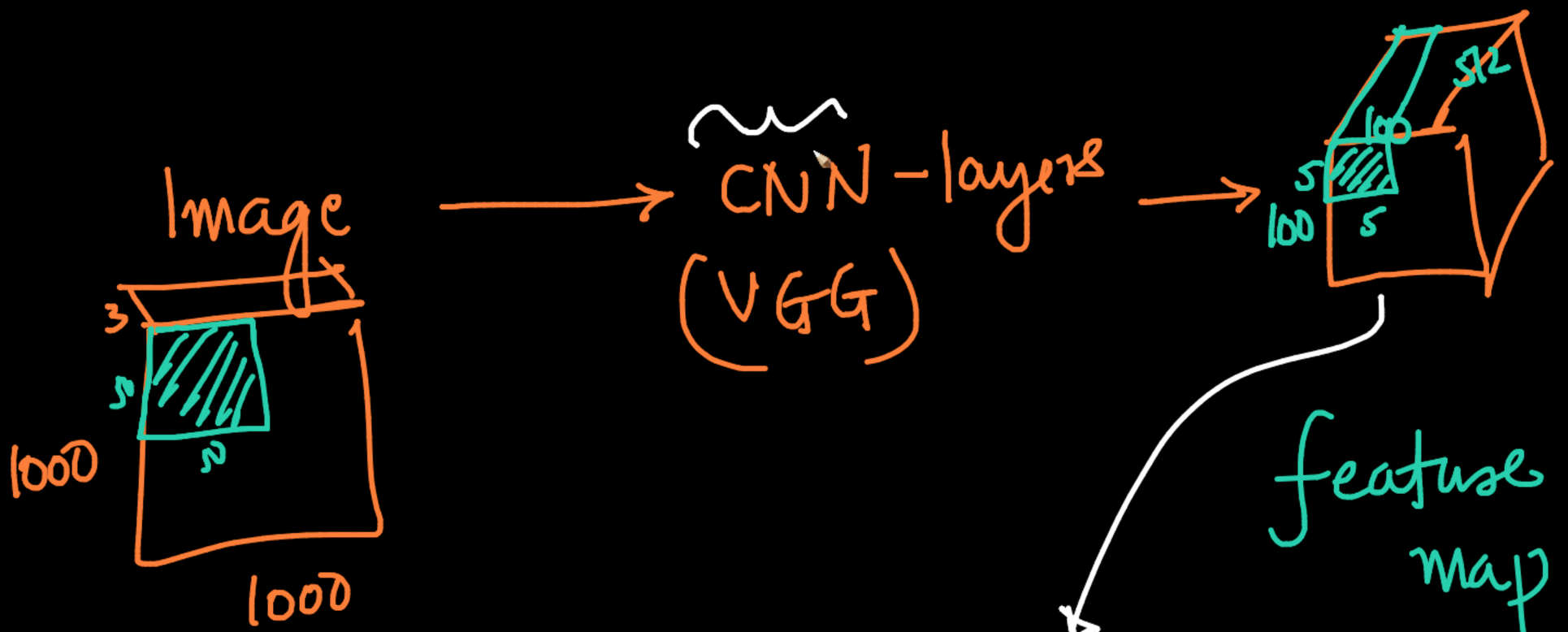
1. `We input an image and associated ground-truth bounding boxes.`
2. **Extract the feature map**
3. Apply **ROI pooling** and obtain **the ROI feature vector**.
4. And finally use two sets of fully-connected l

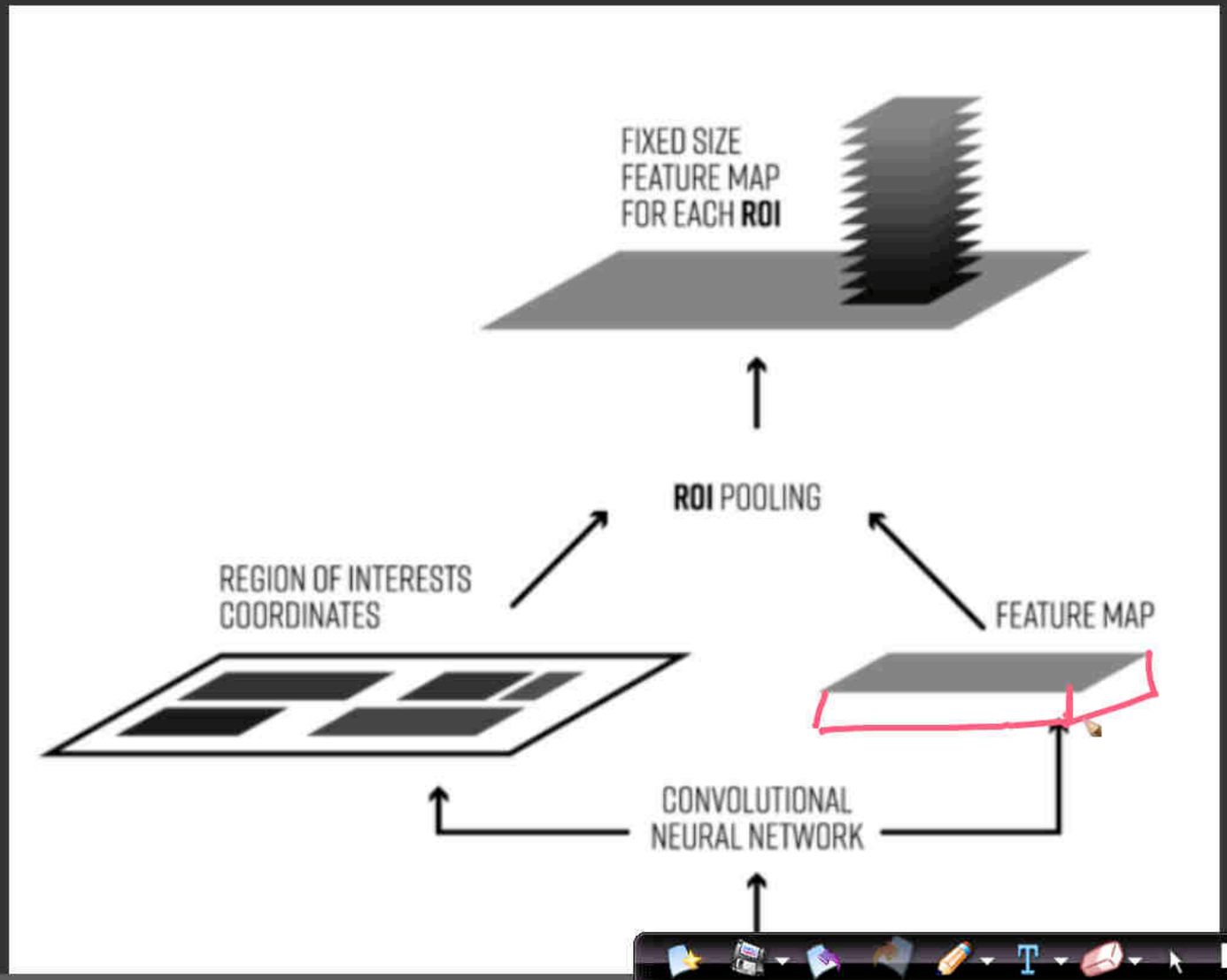


The primary benefit here is that the network is now, effectively, end-to-end trainable:

1. `We input an image and associated ground-truth bounding boxes.`
2. **Extract the feature map**
3. Apply **ROI pooling** and obtain **the ROI feature vector**.
4. And finally use two sets of fully-connected l

Q




FIXED SIZE
FEATURE MAP
FOR EACH ROI
ROI POOLING
REGION OF INTERESTS
COORDINATES
FEATURE MAP
CONVOLUTIONAL
NEURAL NETWORK

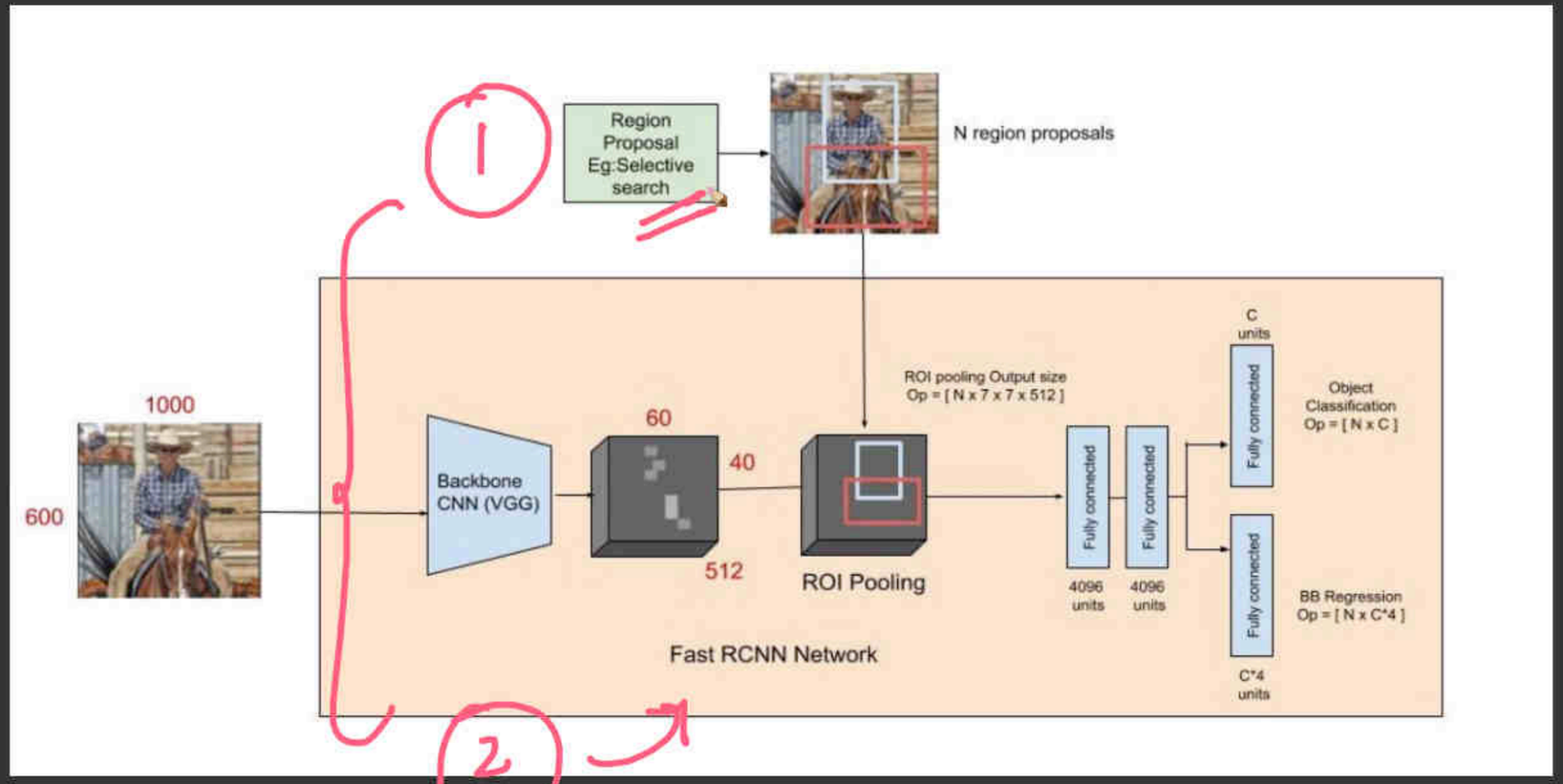
Input Image

Input feature map

0.88	0.44	0.14	0.16	0.37	0.77	0.96	0.27
0.19	0.45	0.57	0.16	0.63	0.29	0.71	0.70
0.66	0.26	0.82	0.64	0.54	0.73	0.59	0.26
0.85	0.34	0.76	0.84	0.29	0.75	0.62	0.25
0.32	0.74	0.21	0.39	0.34	0.03	0.33	0.48
0.20	0.14	0.16	0.13	0.73	0.65	0.96	0.32
0.19	0.69	0.09	0.86	0.88	0.07	0.01	0.48
0.83	0.24	0.97	0.04	0.24	0.35	0.50	0.91

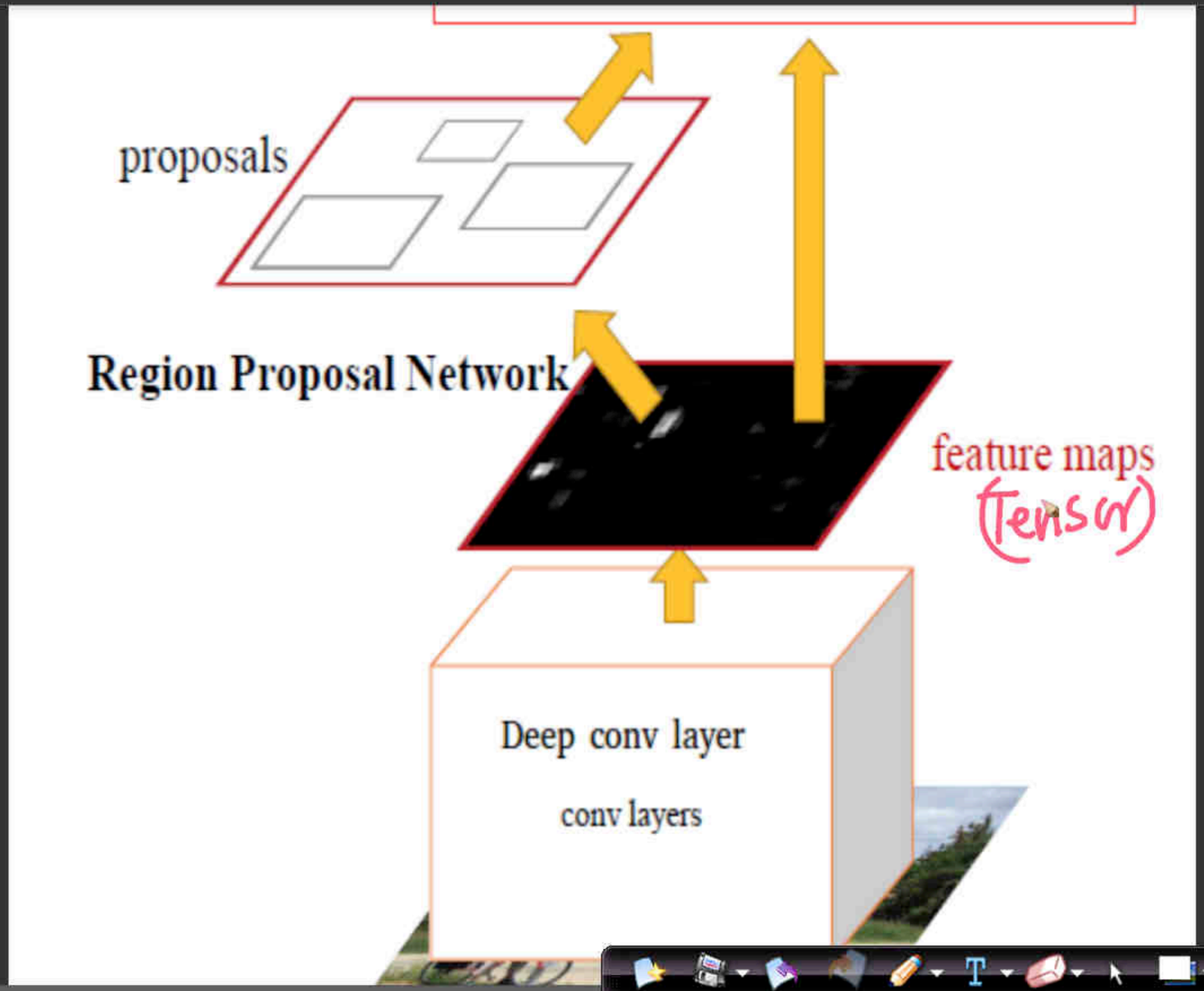
To Include: Corresponding GUN Image

- We'll discuss ROI Pooling in more detail later, but for the time being, understand that **ROI Pooling operates over the feature map extracted from the CNN and extracts a fixed-size window from it.**



The primary benefit here is that the network is now, effectively, end-to-end trainable:

1. We input an image and associated gr


proposals
Region Proposal Network
feature maps
(Tensor)
Deep conv layer
conv layers

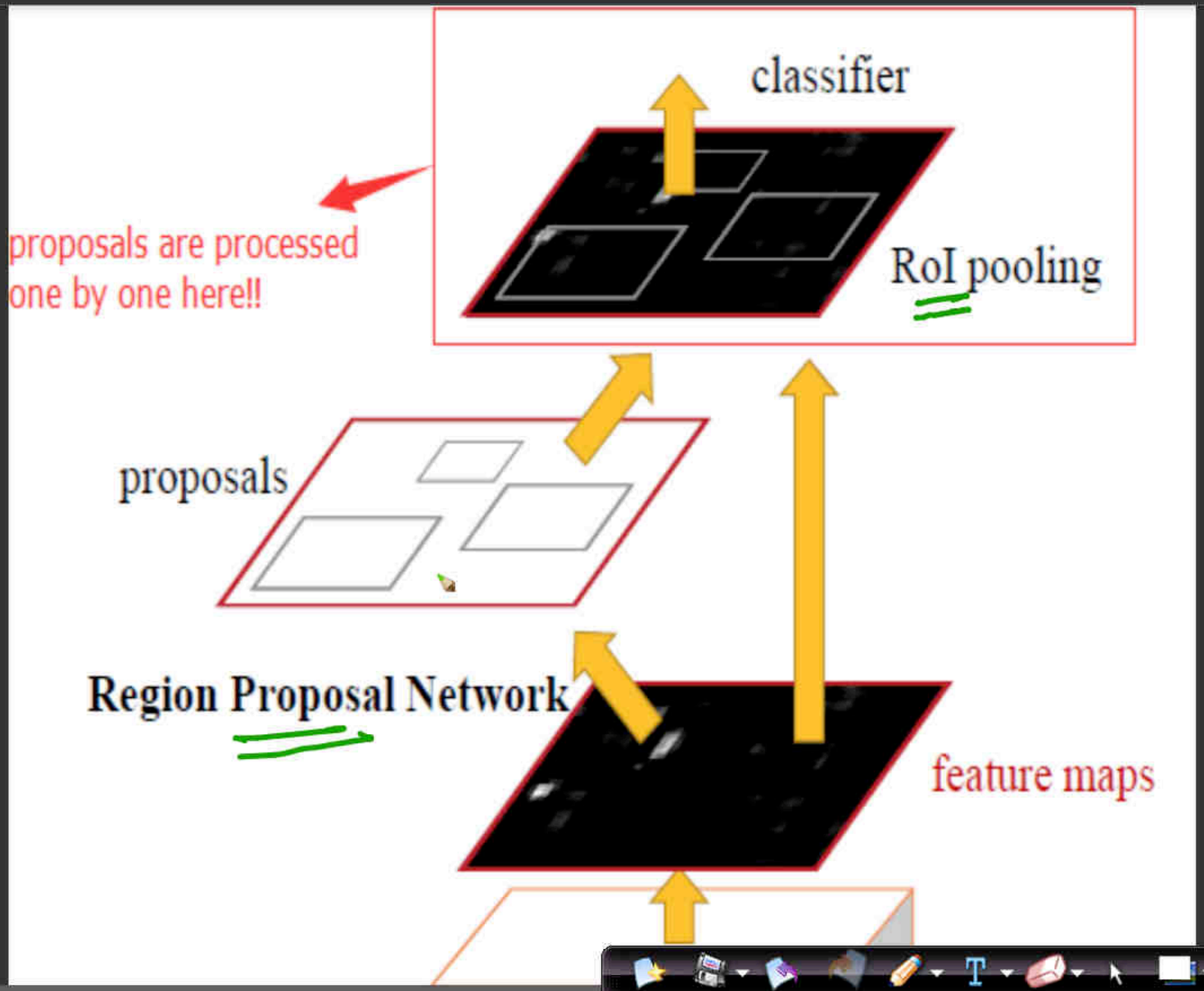


Image → CNN layers

↓

Feature map (T)

RPN(?)

RoI pooling

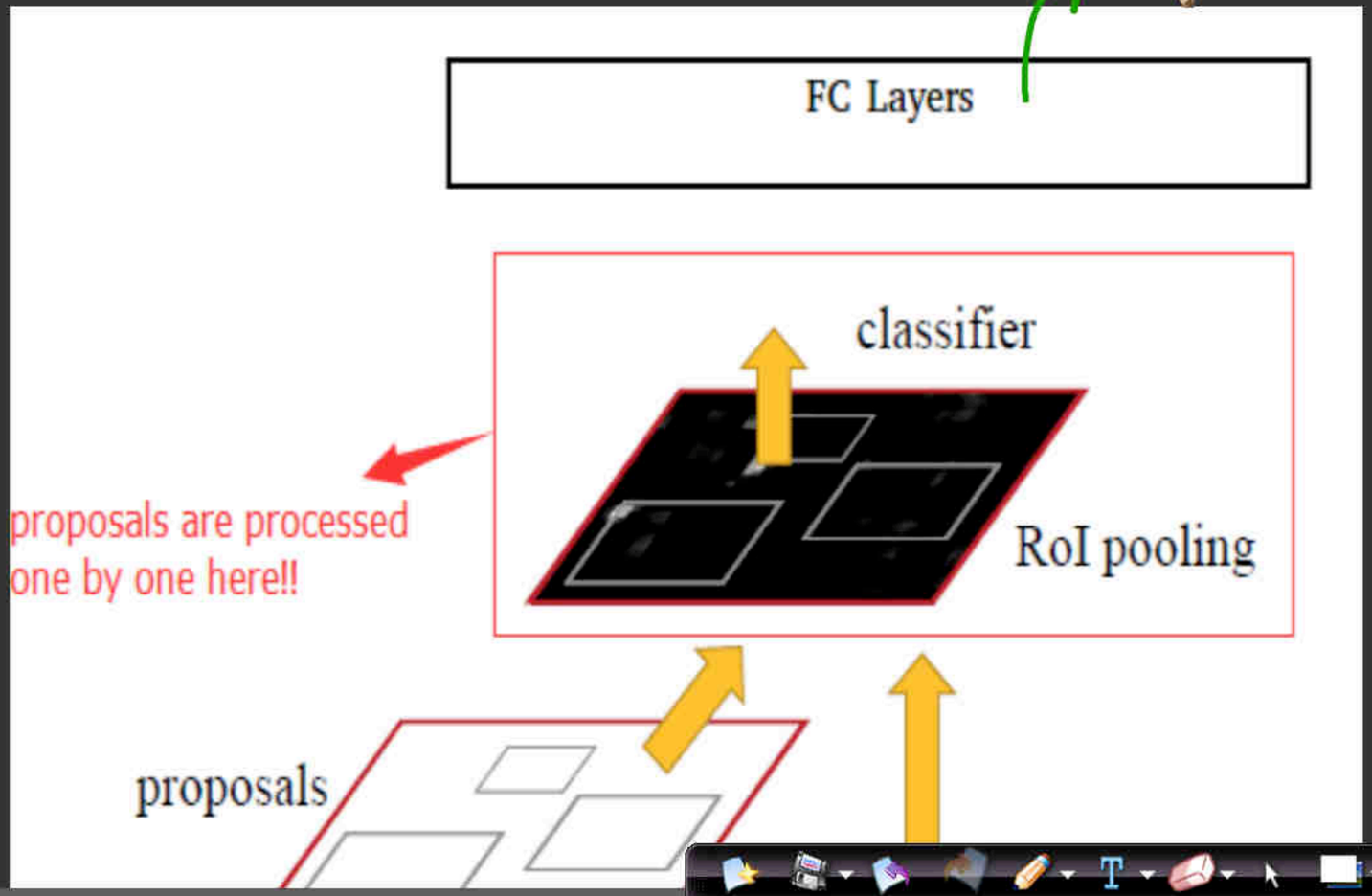
Chrome File Edit View History Bookmarks Profiles Tab Window Help

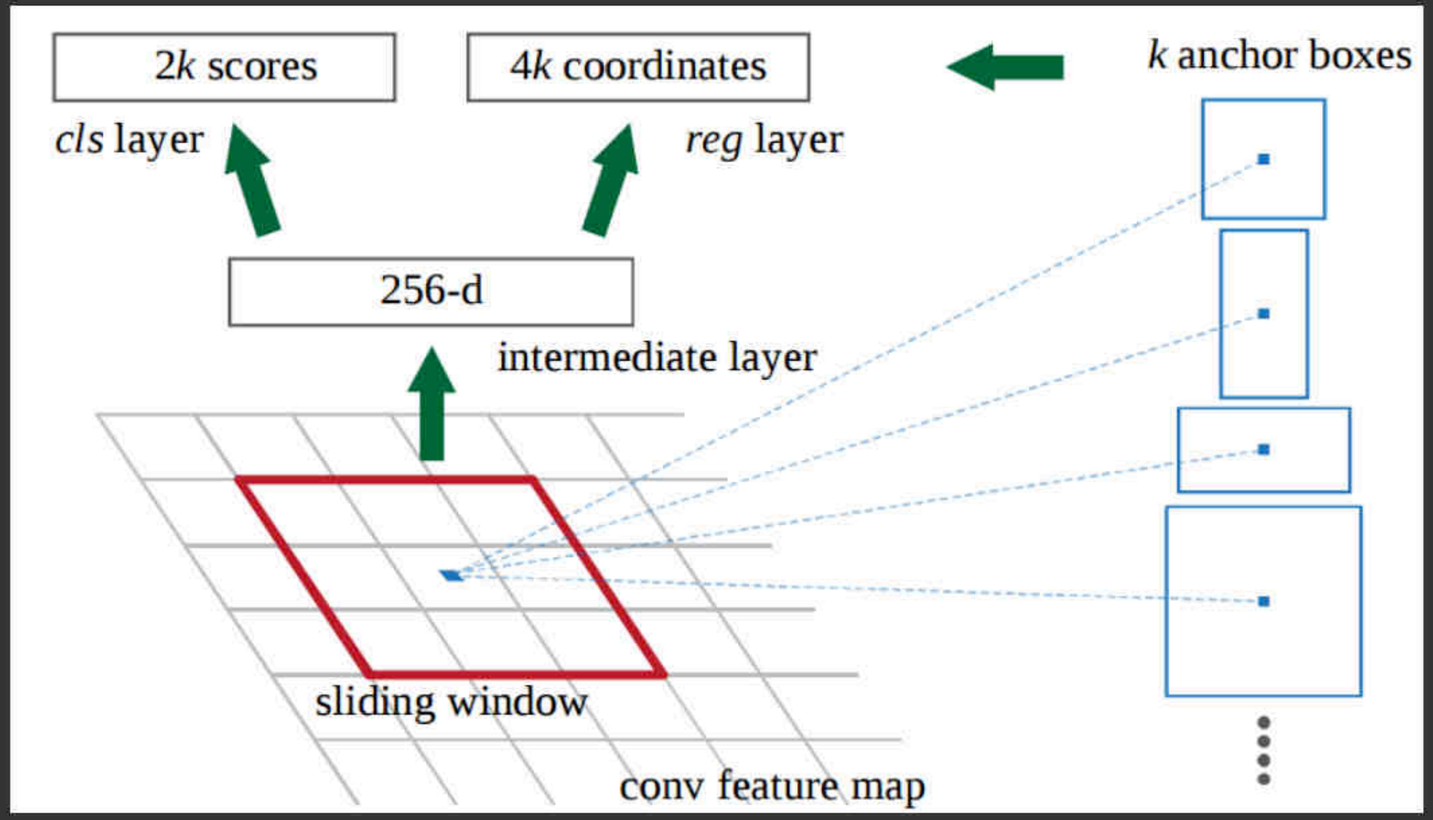
colab.research.google.com/drive/1MTvGvZPDhYaHhf3fkCyfgyFpaHqqAQO1#scrollTo=F3e5Jus83oh8

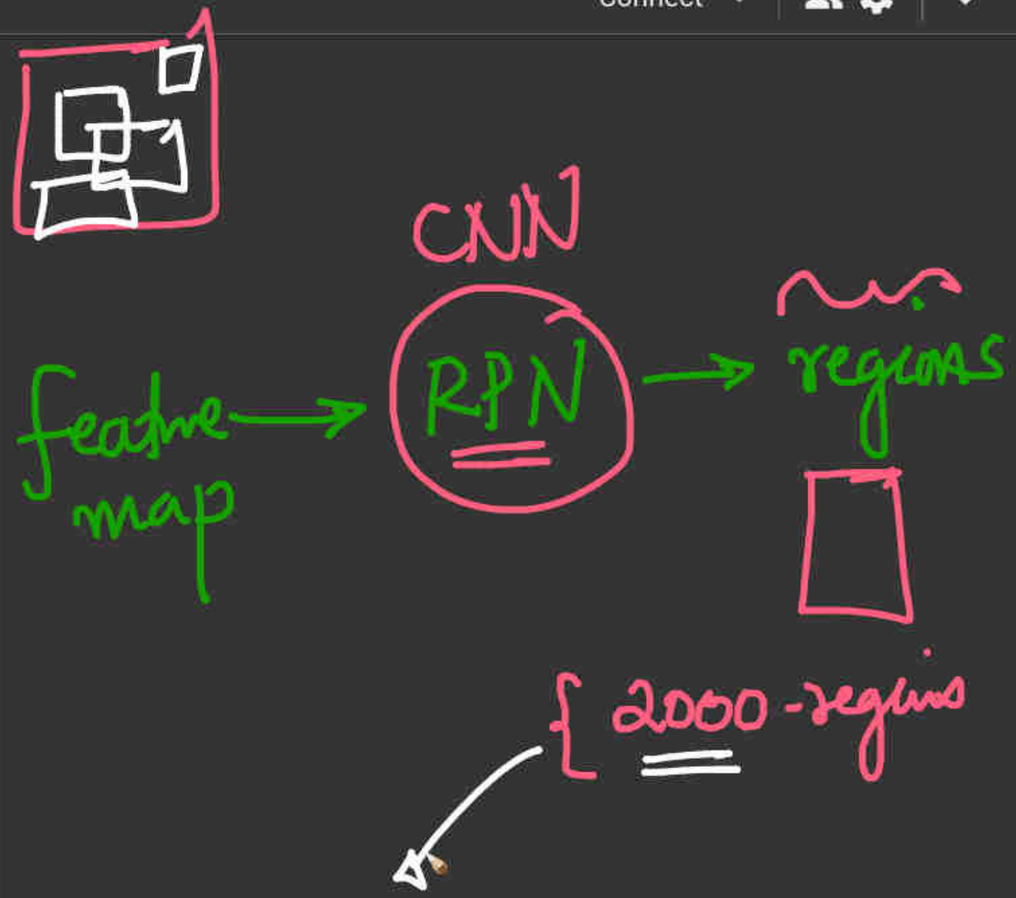
+ Code + Text Connect

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

https://arxiv.org/pdf/1506.01497.pdf







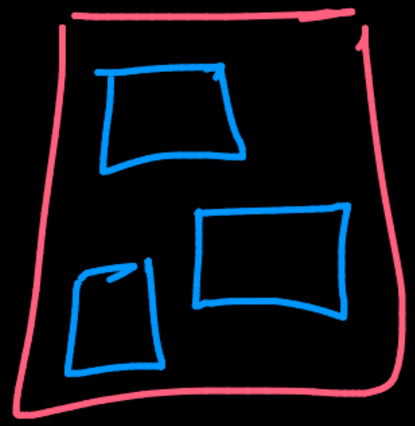
Loss function of Regional Proposal Network is the sum of classification (cls) and regression (reg) loss.

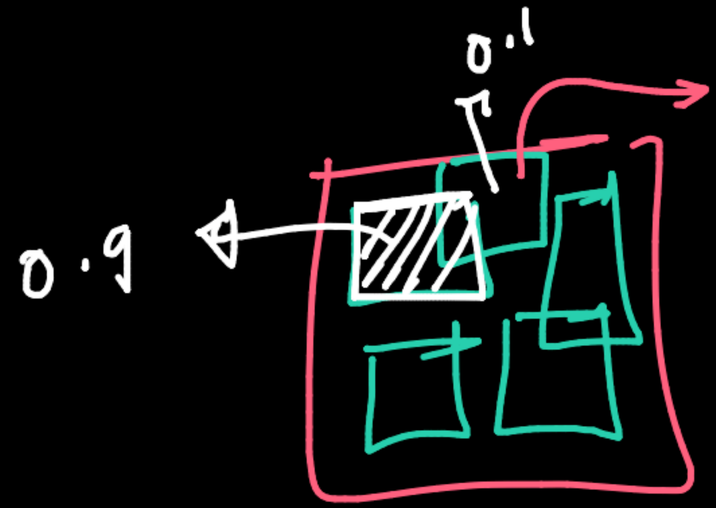
<< Check Equation (1) in the research paper: https://arxiv.org/pdf/1506.01497.pdf >>

feature map → Conv. layers (RPN)

Conv. layers → K boxes × 4 → reg loss

Conv. layers → K → log. loss of each of the K boxes





second condition may find no positive sample. We assign a negative label to a non-positive anchor if its IoU ratio is lower than 0.3 for all ground-truth boxes. Anchors that are neither positive nor negative do not contribute to the training objective.

With these definitions, we minimize an objective function following the multi-task loss in Fast R-CNN [2]. Our loss function for an image is defined as:

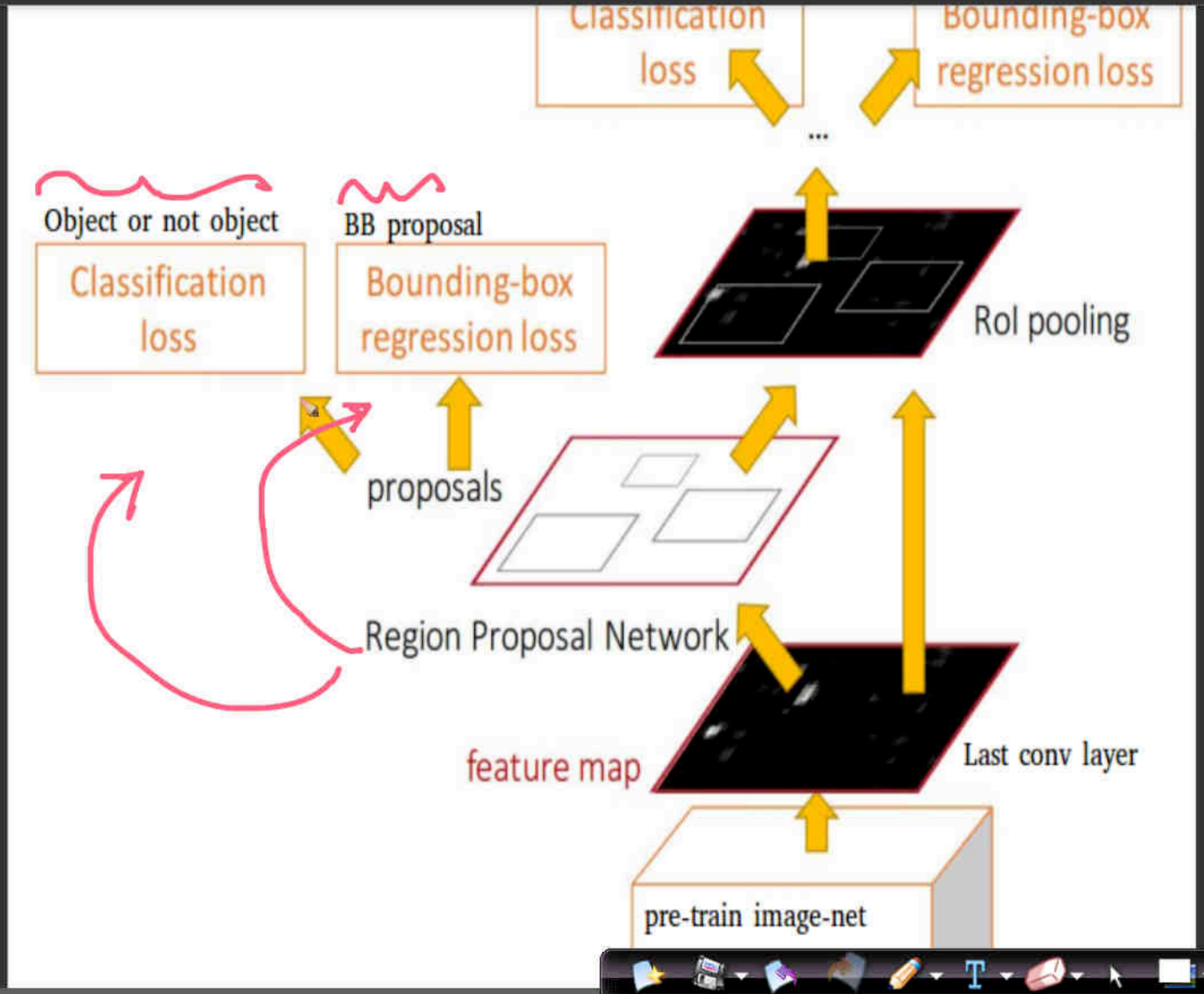
$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*). \quad (1)$$

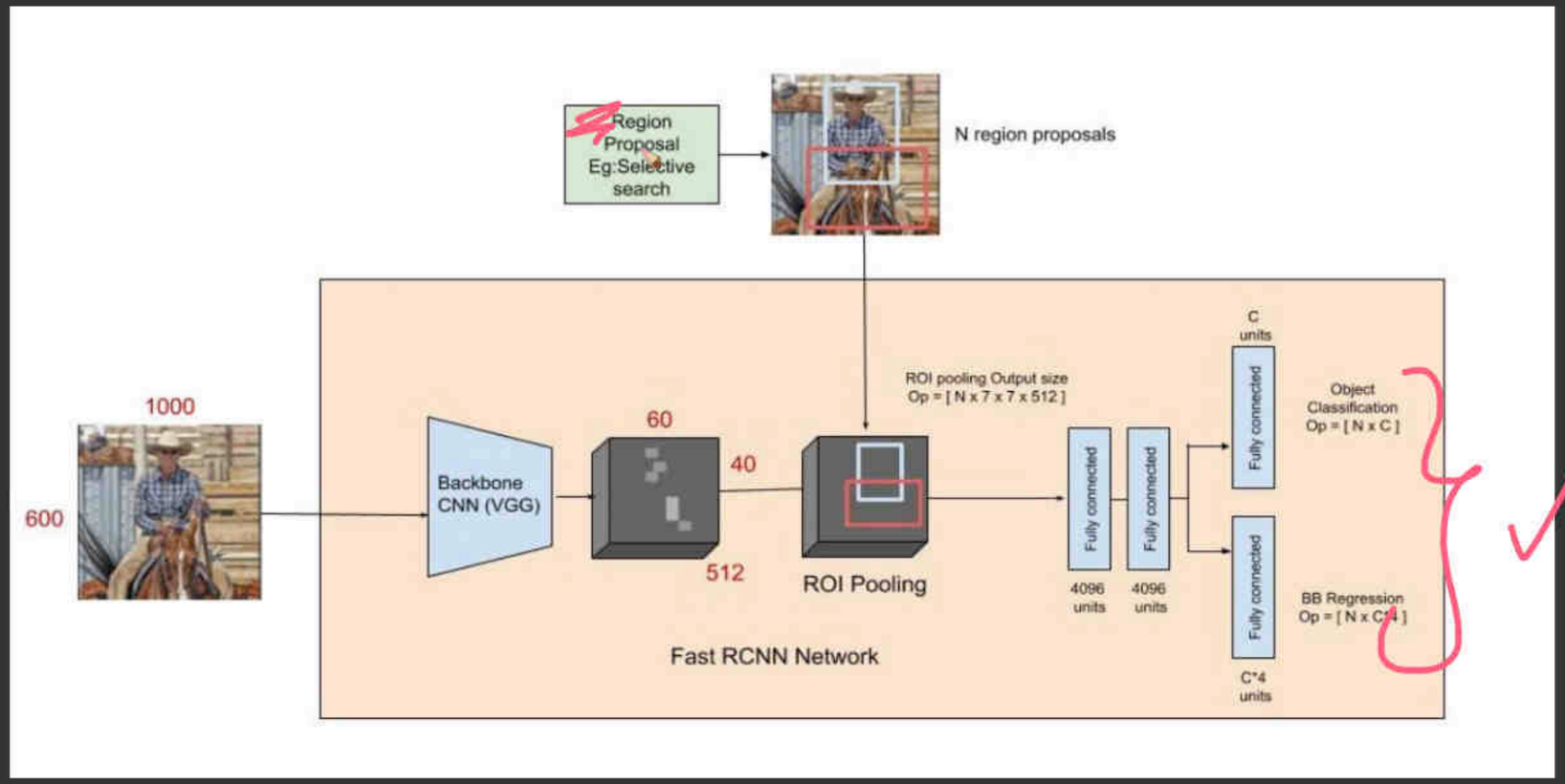
Here, i is the index of an anchor in a mini-batch and p_i is the predicted probability of anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive, and is 0 if the anchor is negative. t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box, and t_i^* is that of the ground-truth box associated with a positive anchor. The classification loss L_{cls} is log l
(object *vs.* not object). For the re

[2], bounding-box regression is performed on features pooled from *arbitrarily* sized RoIs, and the regression weights are *shared* by all region sizes. In our formulation, the features used for regression are of the *same* spatial size (3×3) on the feature maps. To account for varying sizes, a set of k bounding-box regressors are learned. Each regressor is responsible for one scale and one aspect ratio, and the k regressors do *not* share weights. As such, it is still possible to predict boxes of various sizes even though the features are of a fixed size/scale, thanks to the design of anchors.

3.1.3 Training RPNs

The RPN can be trained end-to-end by back-propagation and stochastic gradient descent (SGD) [35]. We follow the "image-centric" sampling strategy from [2] to train this network. Each mini-batch arises from a single image that contains many positive and negative example anchors. It is possible to optimize for the loss functions of all anchors, but this will bias towards negative samples as they are dominate.
ple 256 anchors in an image
to compute the loss function of a mini-batch, where

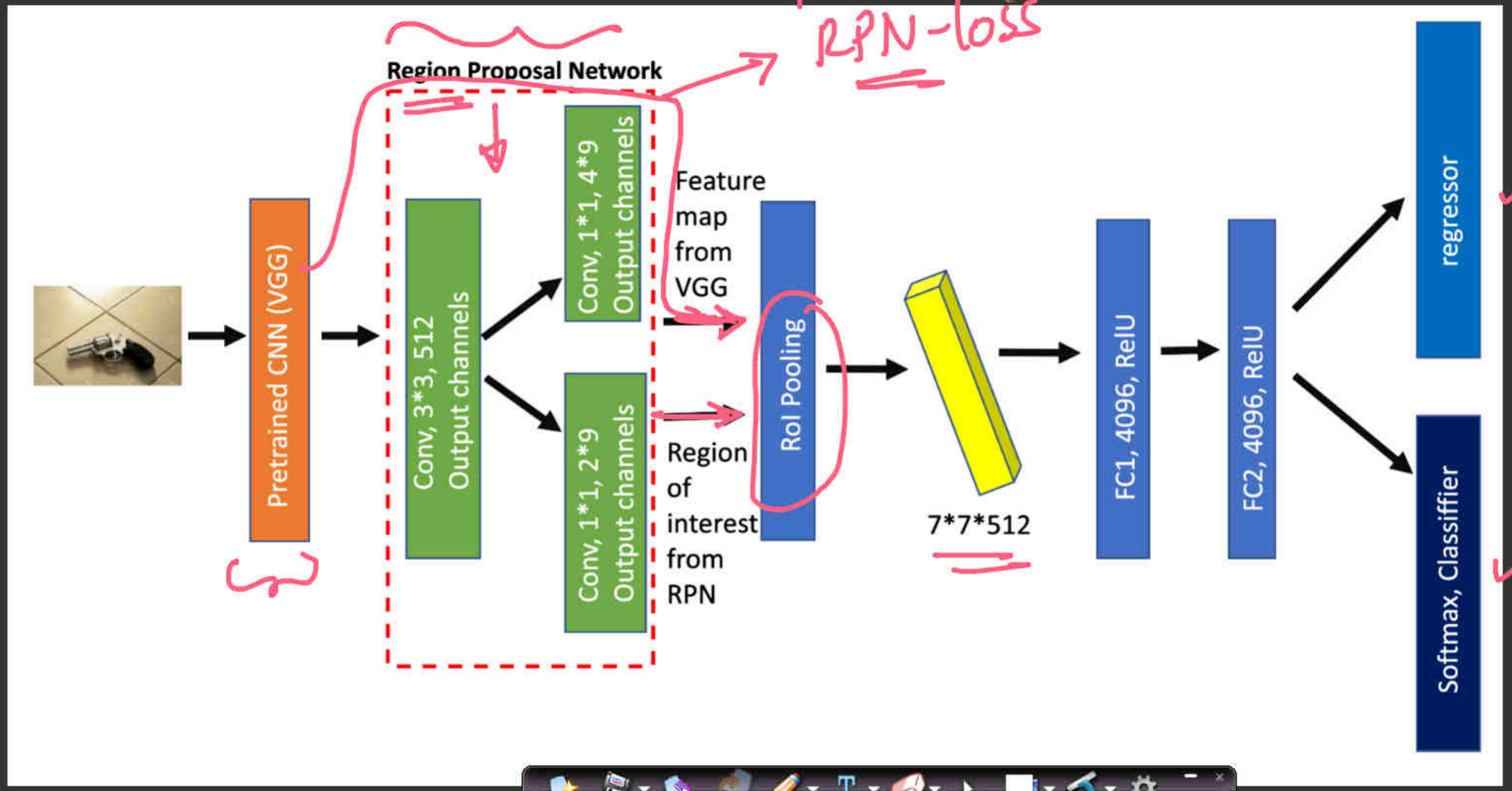

Classification loss
Bounding-box regression loss
...
Object or not object
BB proposal
Classification loss
Bounding-box regression loss
RoI pooling
proposals
Region Proposal Network
feature map
Last conv layer
pre-train image-net

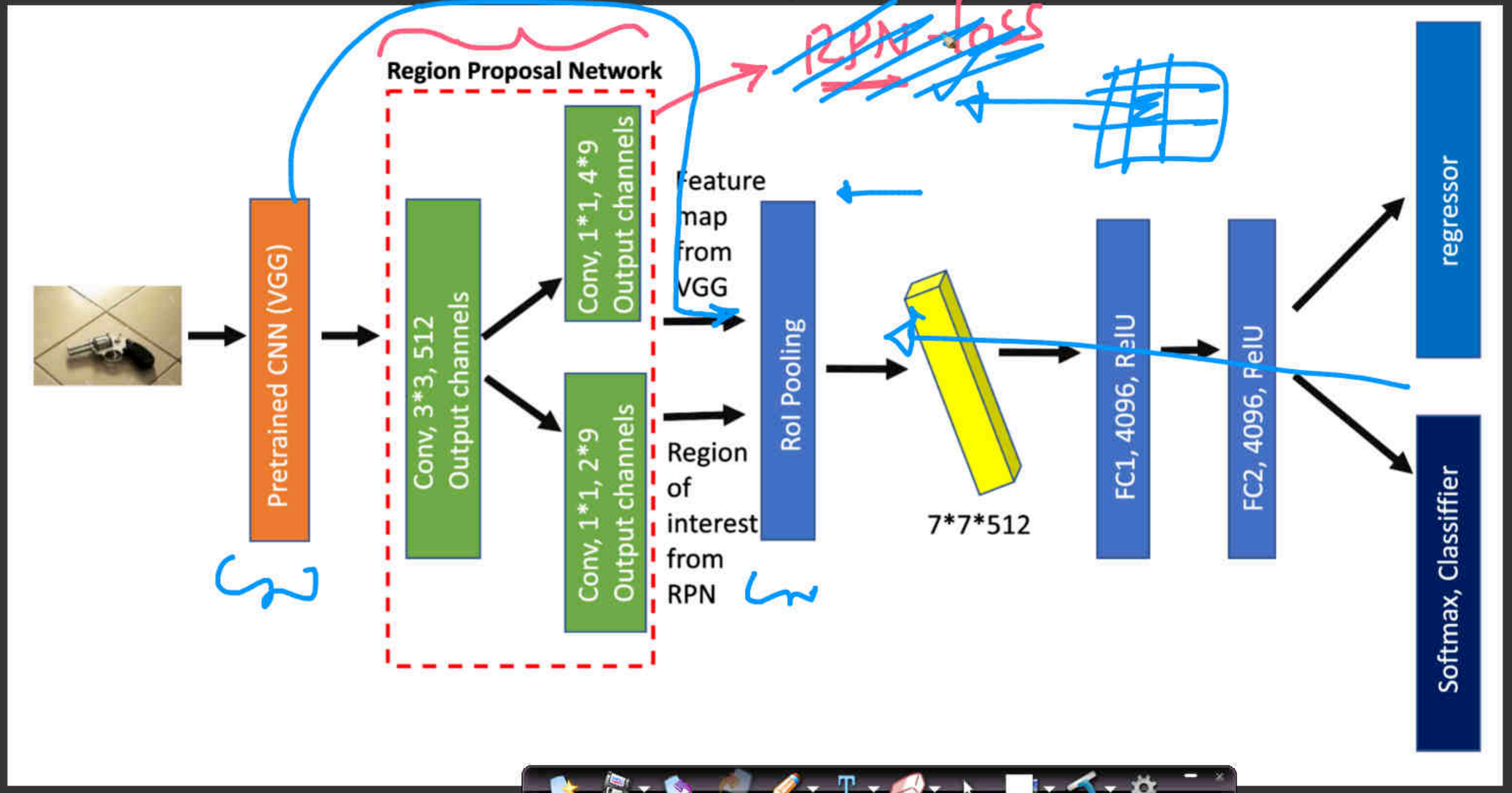


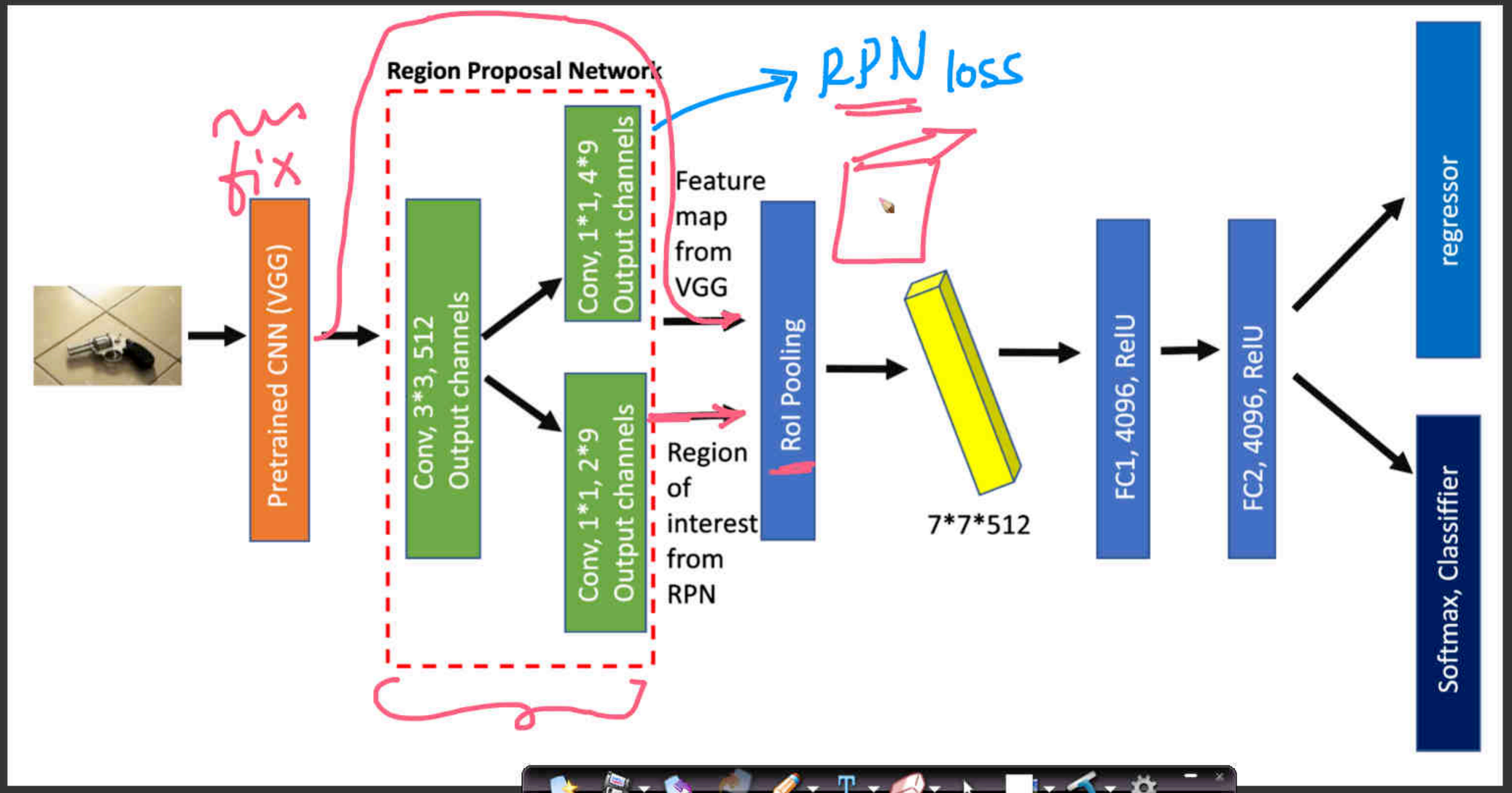
Faster RCNN

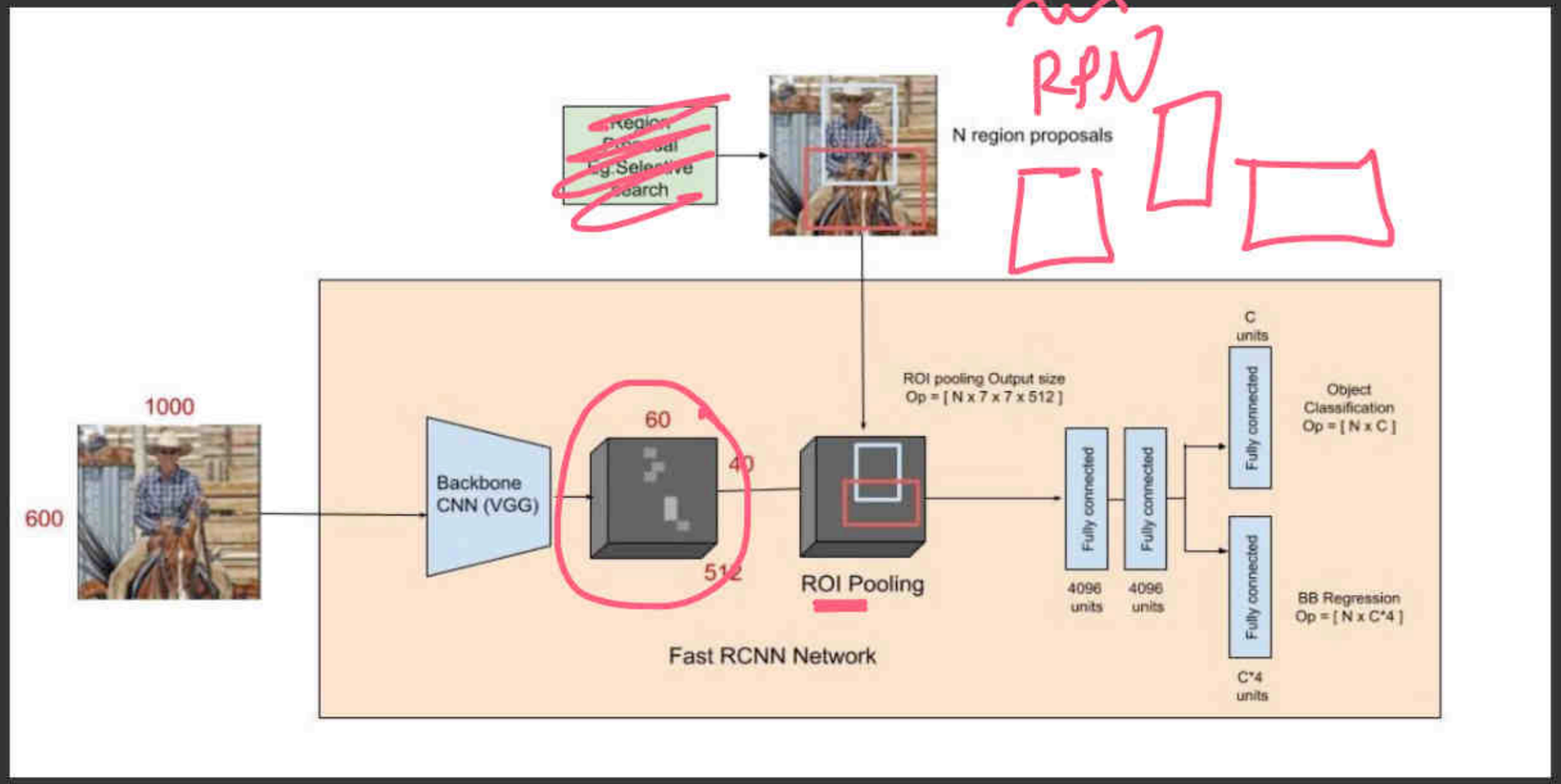
The primary benefit here is that the network is now, effectively, end-to-end trainable:

1. `We input an image and associated ground-truth bounding boxes`.
2. **Extract the feature map**
3. Apply **ROI pooling** and obtain **the ROI feature vector**
4. And finally use two sets of fully connected l




Region Proposal Network
RPN loss
Pretrained CNN (VGG)
Conv, 3*3, 512 Output channels
Conv, 1*1, 4*9 Output channels
Conv, 1*1, 2*9 Output channels
Feature map from VGG
Region of interest from RPN
RoI Pooling
7*7*512
FC1, 4096, ReLU
FC2, 4096, ReLU
regressor
Softmax, Classifier

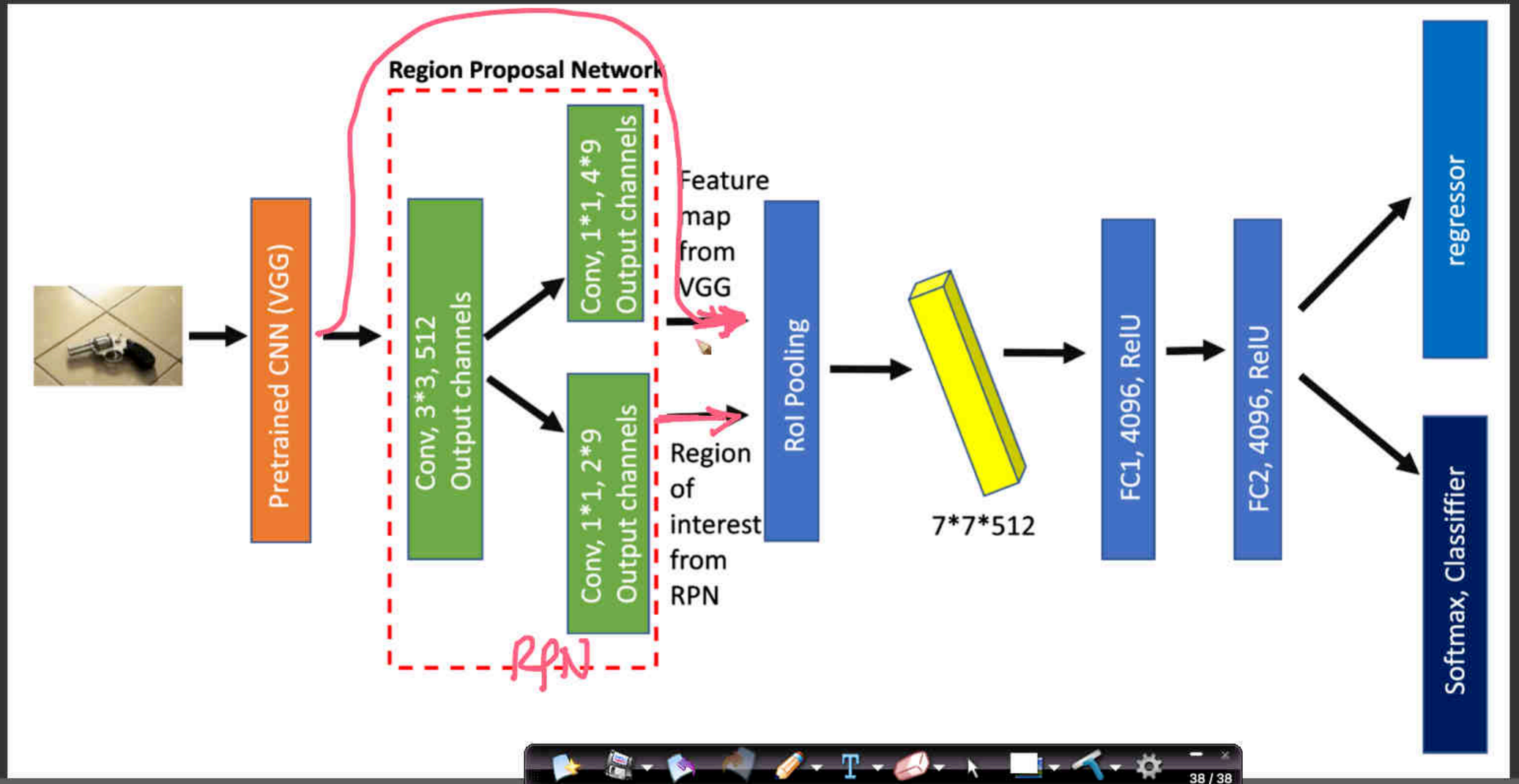

Region Proposal Network
RPN loss
fix
Pretrained CNN (VGG)
Conv, 3*3, 512 Output channels
Conv, 1*1, 4*9 Output channels
Conv, 1*1, 2*9 Output channels
Feature map from VGG
Region of interest from RPN
RoI Pooling
7*7*512
FC1, 4096, ReIU
FC2, 4096, ReIU
regressor
Softmax, Classifier



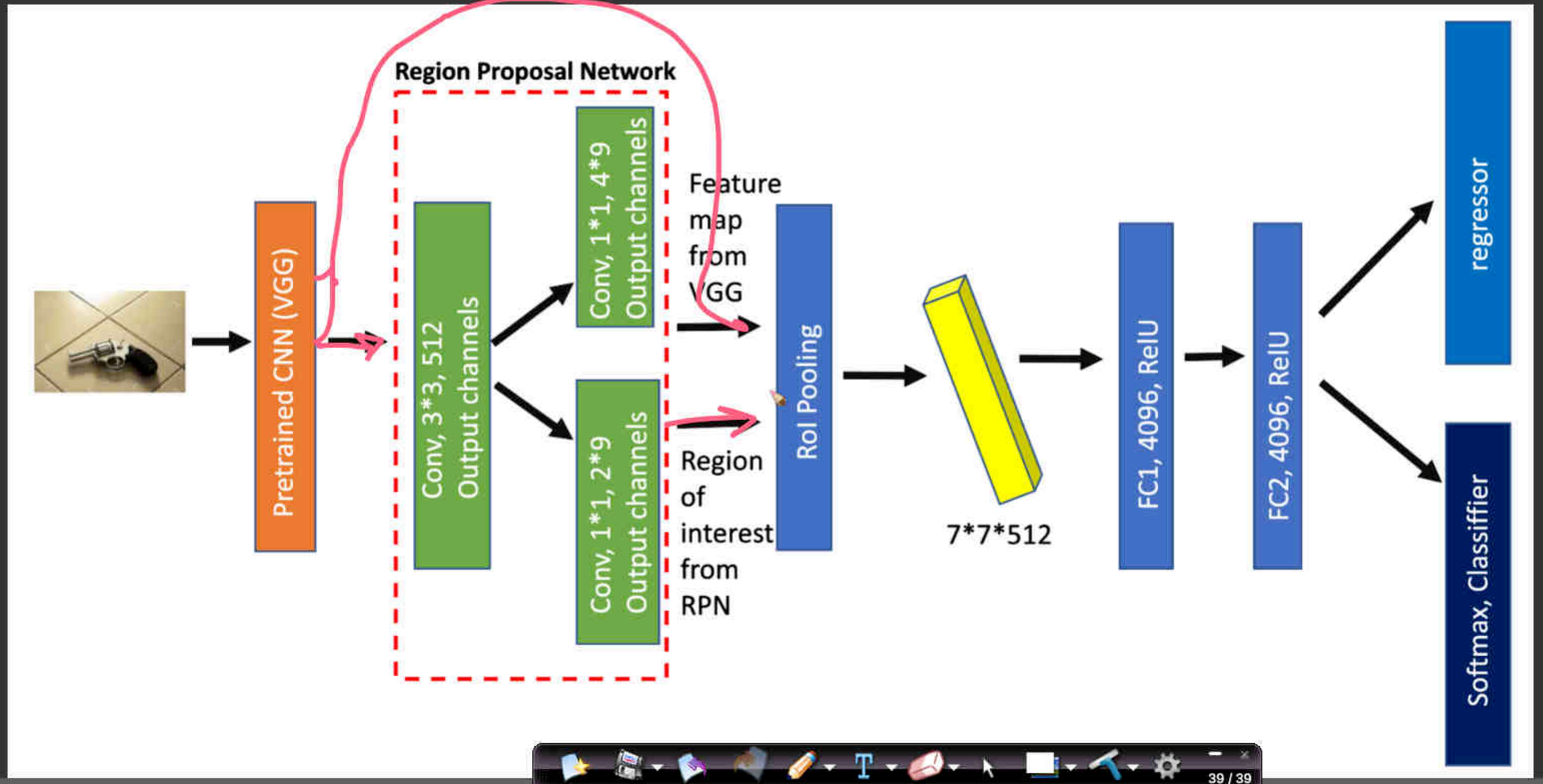
The primary benefit here is that the network is now, effectively, end-to-end trainable:

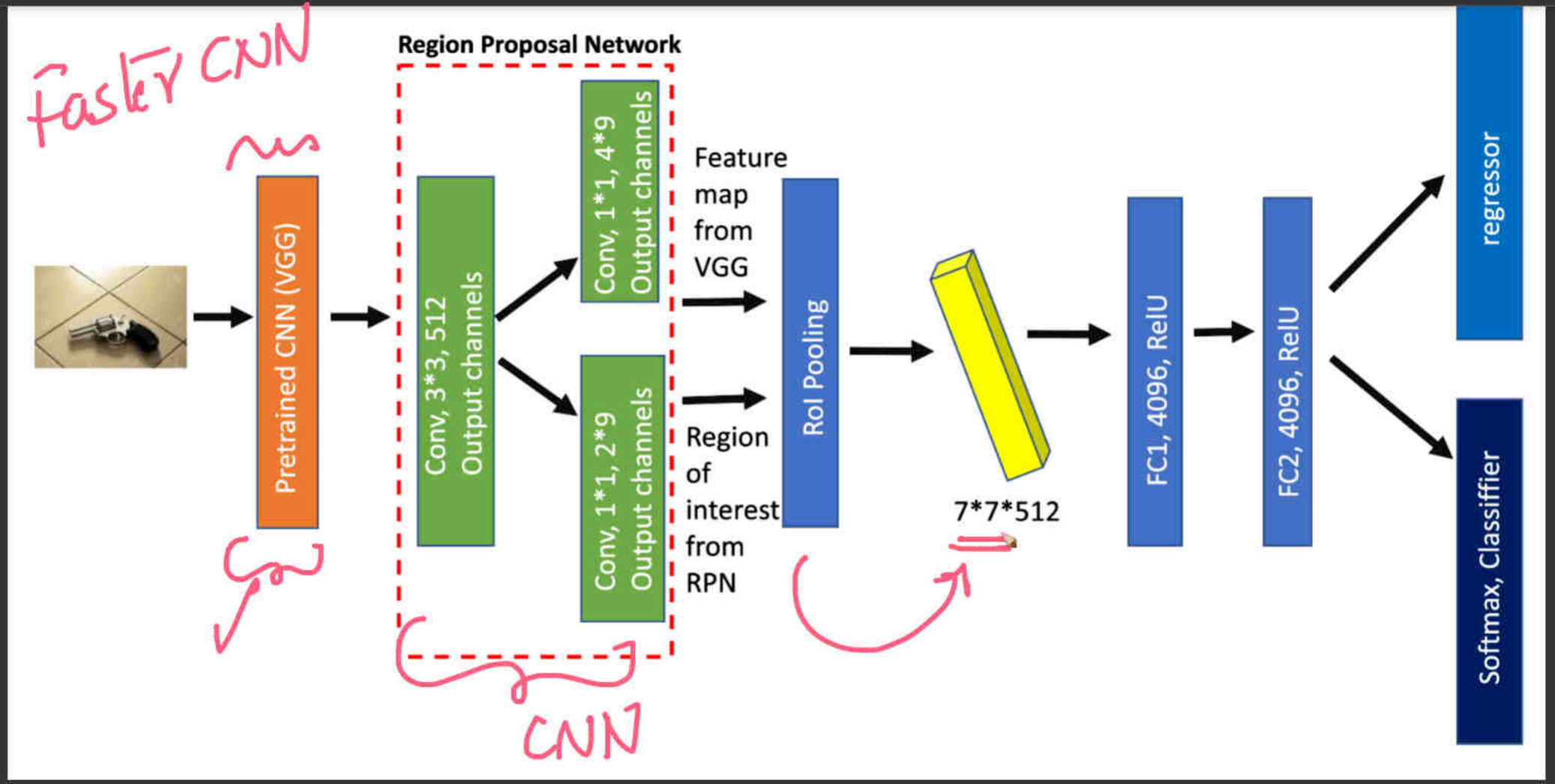
1. `We input an image and associated ground-truth bounding boxes.`
2. **Extract the feature map**
3. Apply **ROI pooling** and obtain **the ROI feature vector**.
4. And finally use two sets of fully-connected l

Given below is comparison of Speed and mAP metric for all three versions of RCNN on VOC2007 Dataset

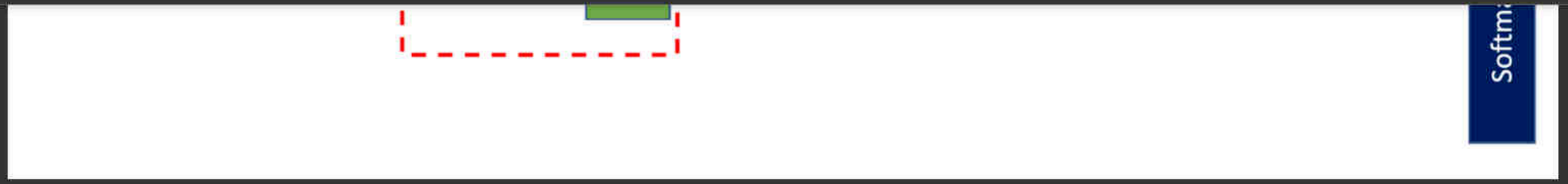


Given below is comparison of Speed and mAP metric for all three versions of RCNN on VOC2007 Dataset





External region proposals method



External region proposals method (Selective Search Algorithm)	✓	✓	✗
	R-CNN	**Fast R-CNN**	**Faster R-CNN**
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x
mAP (VOC 2007)	66.0%	66.9%	66.9%

24 fps

surveillance

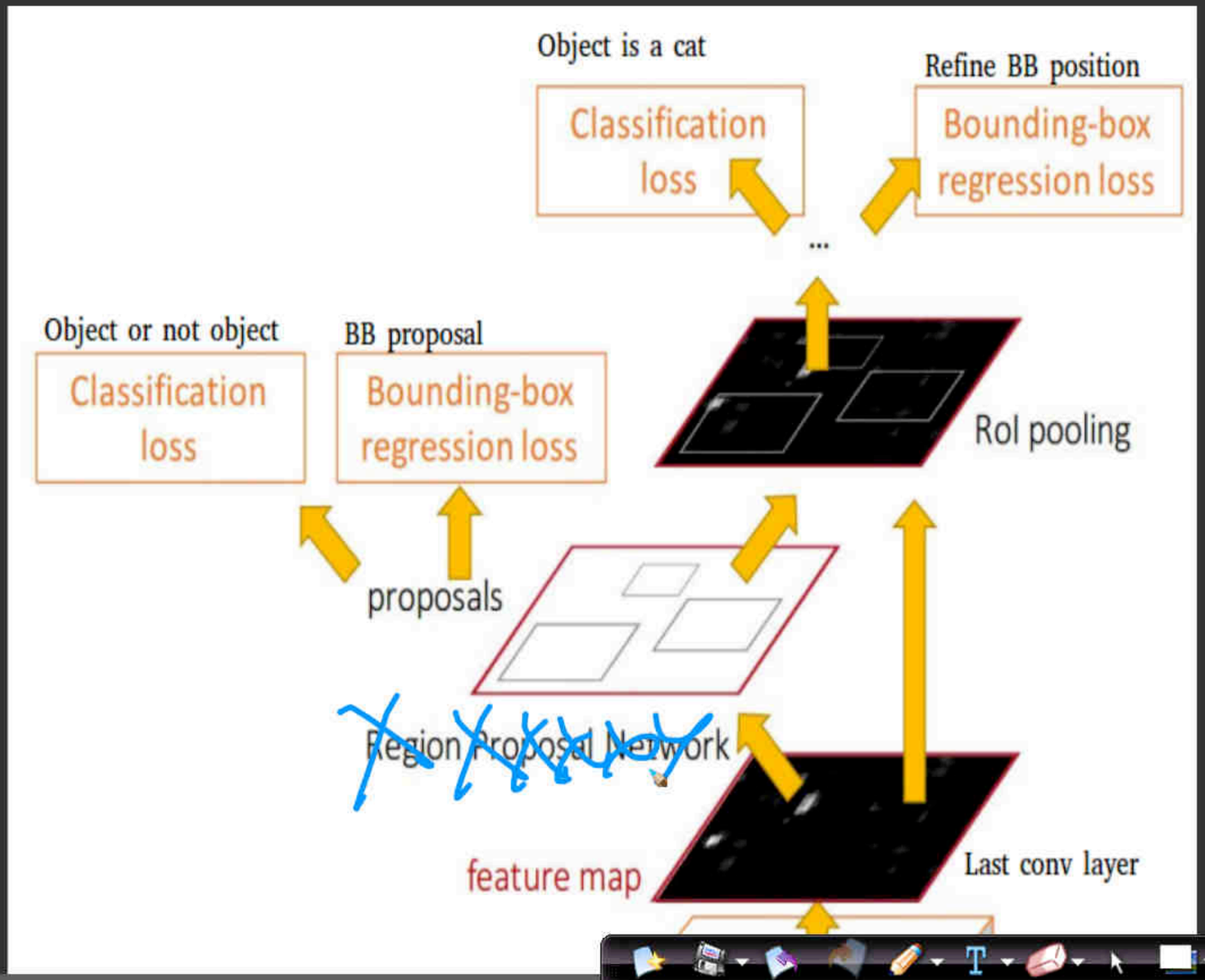
5

$\frac{1}{30}$ sec = 0.033 sec

self-driving car

[Below part is Optional]

Code Implentation of Backbone for FasterRCNN(VGG:



2 stages
→ RPN
→ RoI + FC → BBOX
→ Class label

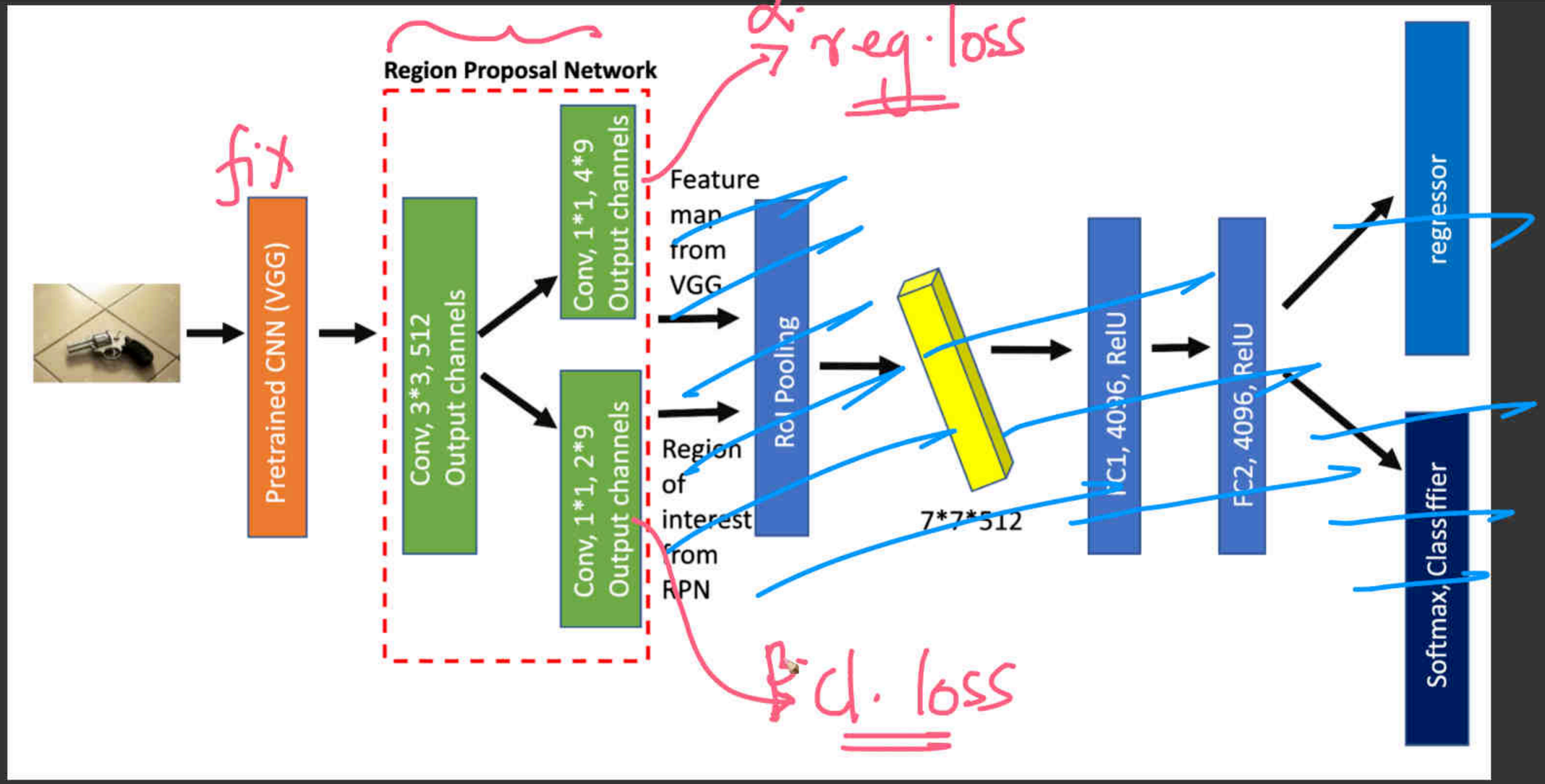
```
        x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(x)
        # x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)

        # We are not using fully connected layers (3 fc layers) as we need feature maps as output from this network.

        return x
```

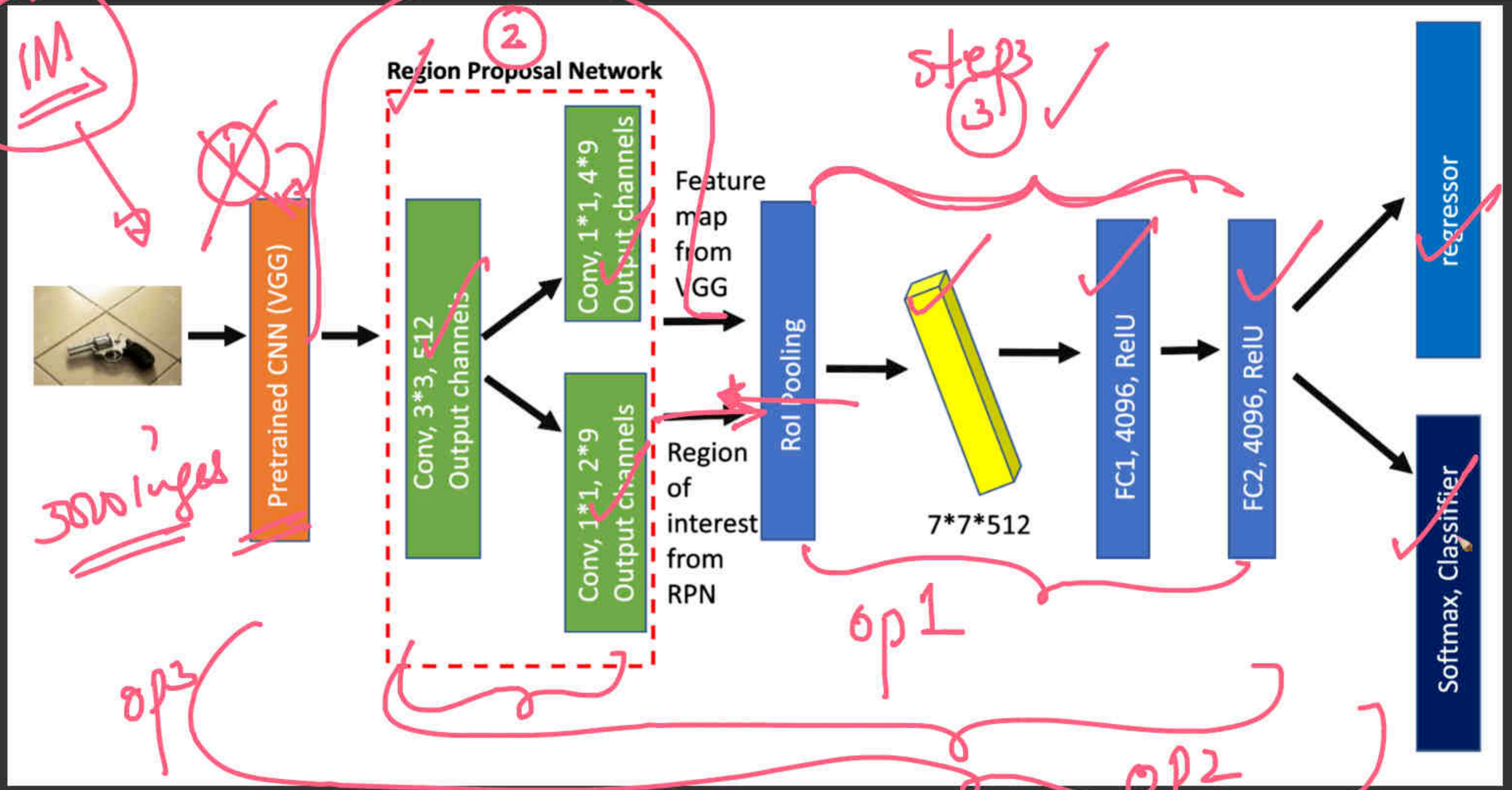
Code Implementation OF RPN:

```
[ ] ""
    #RPN layer
    def rpn_layer(base_layers, num_anchors):

        #cnn_used for creating feature maps: vgg, num_anchors: 9
        x = Conv2D(512, (3, 3), padding='same', activation='relu')(base_layers)

        #classification layer: num_anchors (9) channels for 0, 1 sigmoid activation output
        x_class = Conv2D(num_anchors, (1, 1), activation='sigmoid')(x)

        #regression layer: num_anchors*4 (36) channels for computing the regression of bboxes
        x_regr = Conv2D(num_anchors * 4, (1, 1), activation='linear')(x)

        return [x_class, x_regr, base_layers] #classification of object(0 or 1),compute bounding boxes, base layers vgg
```

Code Implementation of ROI Pooling Layer:

```
[ ] class RoiPoolingConv(Layer):
```


Region Proposal Network
fix
Pretrained CNN (VGG)
Conv, 3*3, 512 Output channels
Conv, 1*1, 4*9 Output channels
Conv, 1*1, 2*9 Output channels
Feature map from VGG
Region of interest from RPN
RoI Pooling
7*7*512
FC1, 4096, ReLU
FC2, 4096, ReLU
regressor
Softmax, Classifier
reg. loss
cl. loss


Region Proposal Network
Pretrained CNN (VGG)
Conv, 3*3, 512 Output channels
Conv, 1*1, 4*9 Output channels
Conv, 1*1, 2*9 Output channels
Feature map from VGG
Region of interest from RPN
RoI Pooling
7*7*512
FC1, 4096, ReLU
FC2, 4096, ReLU
regressor
Softmax, Classifier

- These models don't have the region proposal stage **Region Proposal Network**.

There are majorly two single Stage Detecion algorithms:

1. **SSD**: Single Shot Detector
2. **Yolo**: You Only Look Once (much more popular and faster)

Let's Dive Deeper into Each one of them:

[Optional Post-READ] Introduction to SSD:

- Original paper: https://arxiv.org/pdf/1512.02325.pdf
- The **core idea behind the SSD network** is to have a **CNN** that takes **in an image as input and produce detections at different scales, shapes, and locations.**

  Let's understand this in detail:

↳ 14 cells hidden

Introduction to YOLO algorithm

- These models don't have the region proposal stage **Region Proposal Network**.

There are majorly two single Stage Detecion algorithms:

1. **SSD**: Single Shot Detector
2. **Yolo**: You Only Look Once (much more popular and faster)

Let's Dive Deeper into Each one of them:

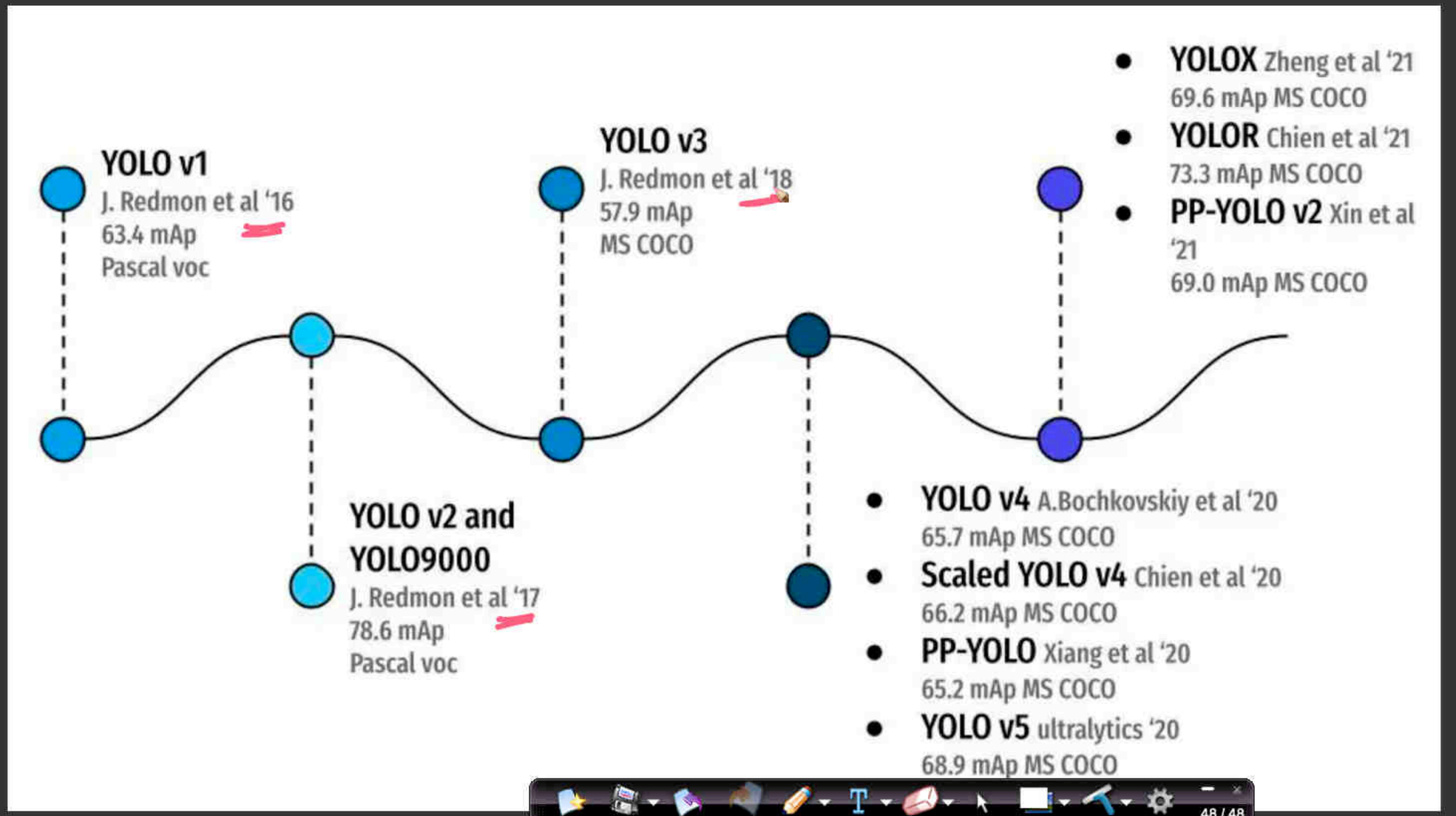
[Optional Post-READ] Introduction to SSD:

- Original paper: https://arxiv.org/pdf/1512.02325.pdf
- The **core idea behind the SSD network** is to have a **CNN** that takes **in an image as input and produce detections at different scales, shapes, and locations.**

  Let's understand this in detail:

↳ 14 cells hidden

Introduction to YOLO algorithm

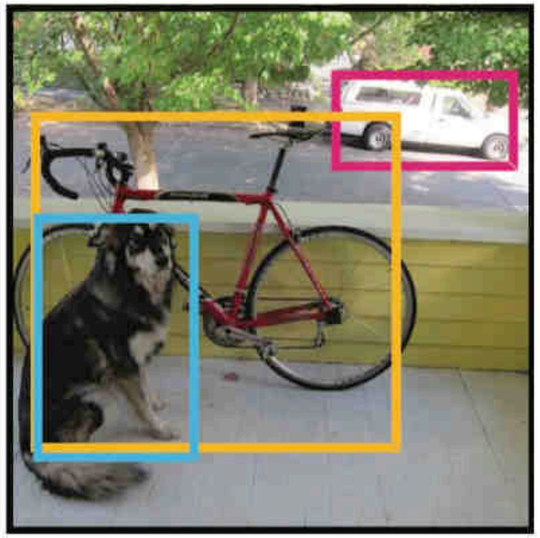

YOLO v1
J. Redmon et al '16
63.4 mAp
Pascal voc
YOLO v2 and
YOLO9000
J. Redmon et al '17
78.6 mAp
Pascal voc
YOLO v3
J. Redmon et al '18
57.9 mAp
MS COCO
YOLO v4 A.Bochkovskiy et al '20
65.7 mAp MS COCO
Scaled YOLO v4 Chien et al '20
66.2 mAp MS COCO
PP-YOLO Xiang et al '20
65.2 mAp MS COCO
YOLO v5 ultralytics '20
68.9 mAp MS COCO
YOLOX Zheng et al '21
69.6 mAp MS COCO
YOLOR Chien et al '21
73.3 mAp MS COCO
PP-YOLO v2 Xin et al '21
69.0 mAp MS COCO
48 / 48

19x19

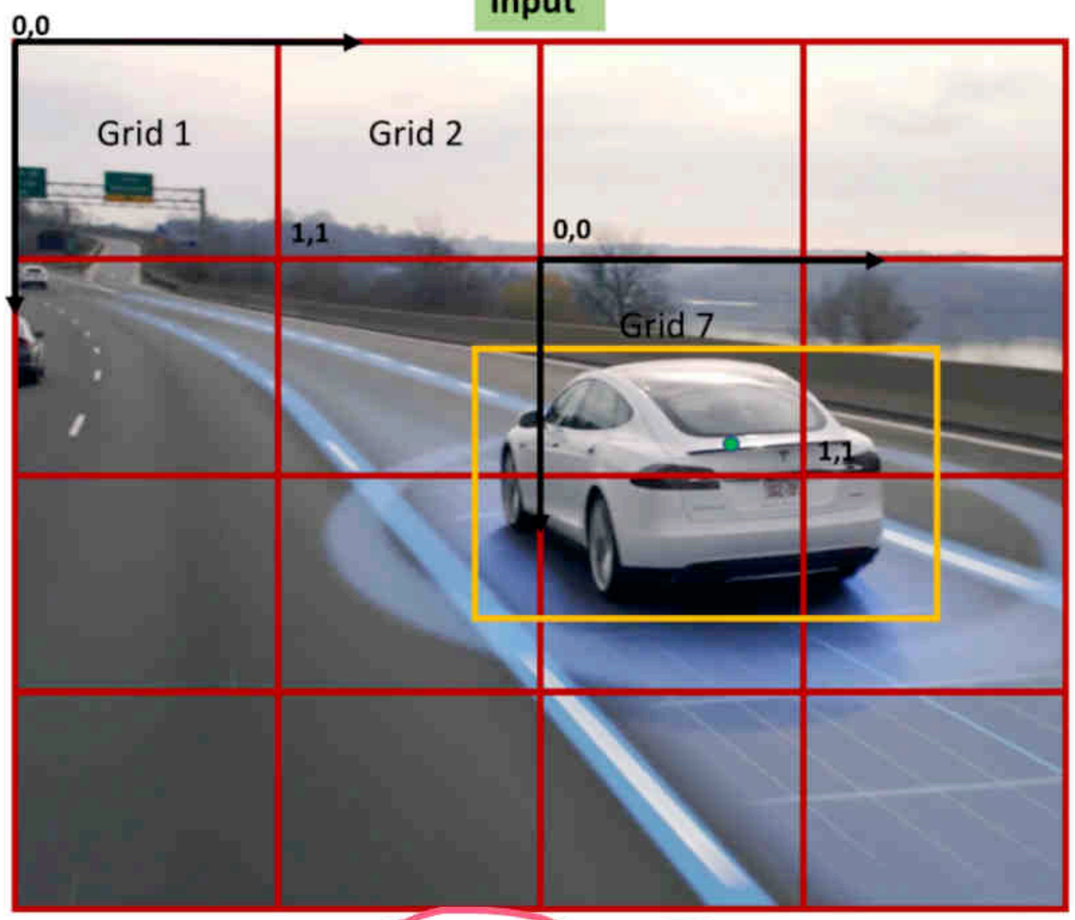
10x10

S × S grid on input

Bounding boxes + confidence

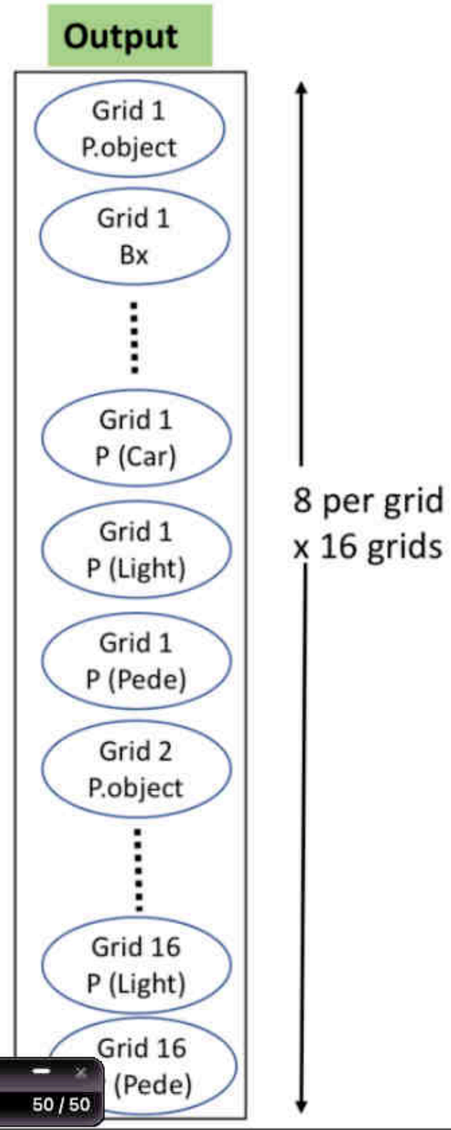


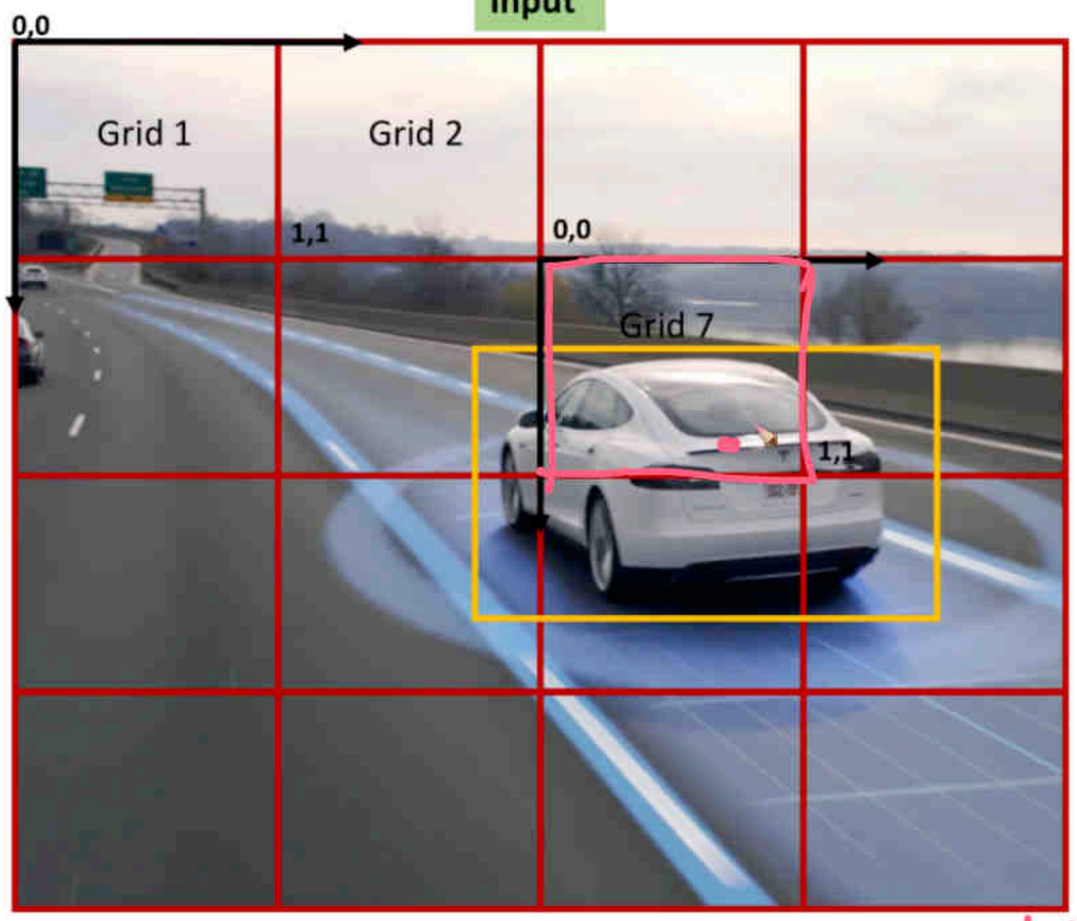
Final detections



Sample output
Grid 7
P.Object : 1
Bx : 0.8
By : 0.9
Bw : 1.5
Bh : 1.2
P.Car : 1
P.Light : 0
P.Pedestrian : 0

P.Object : 0 for all other grids and we don't care about other values if P.Object is 0

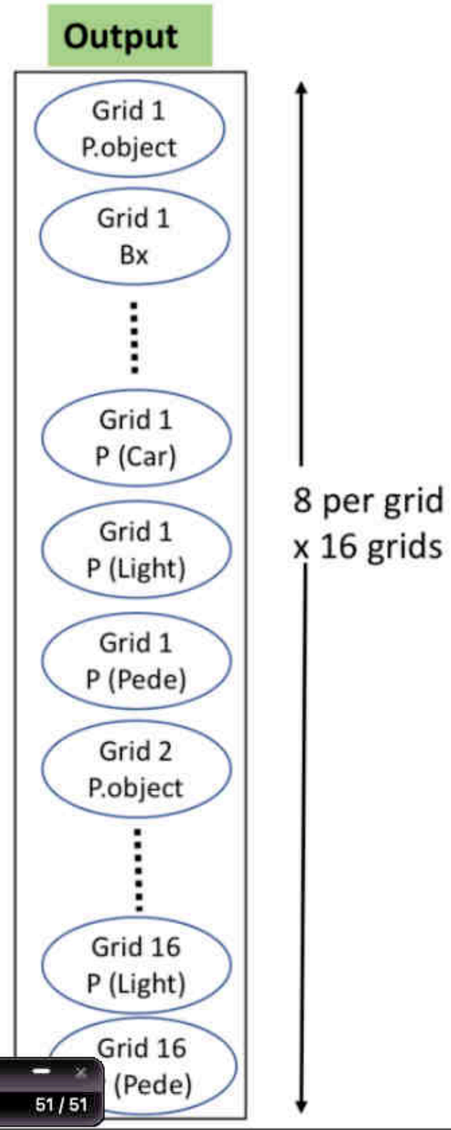


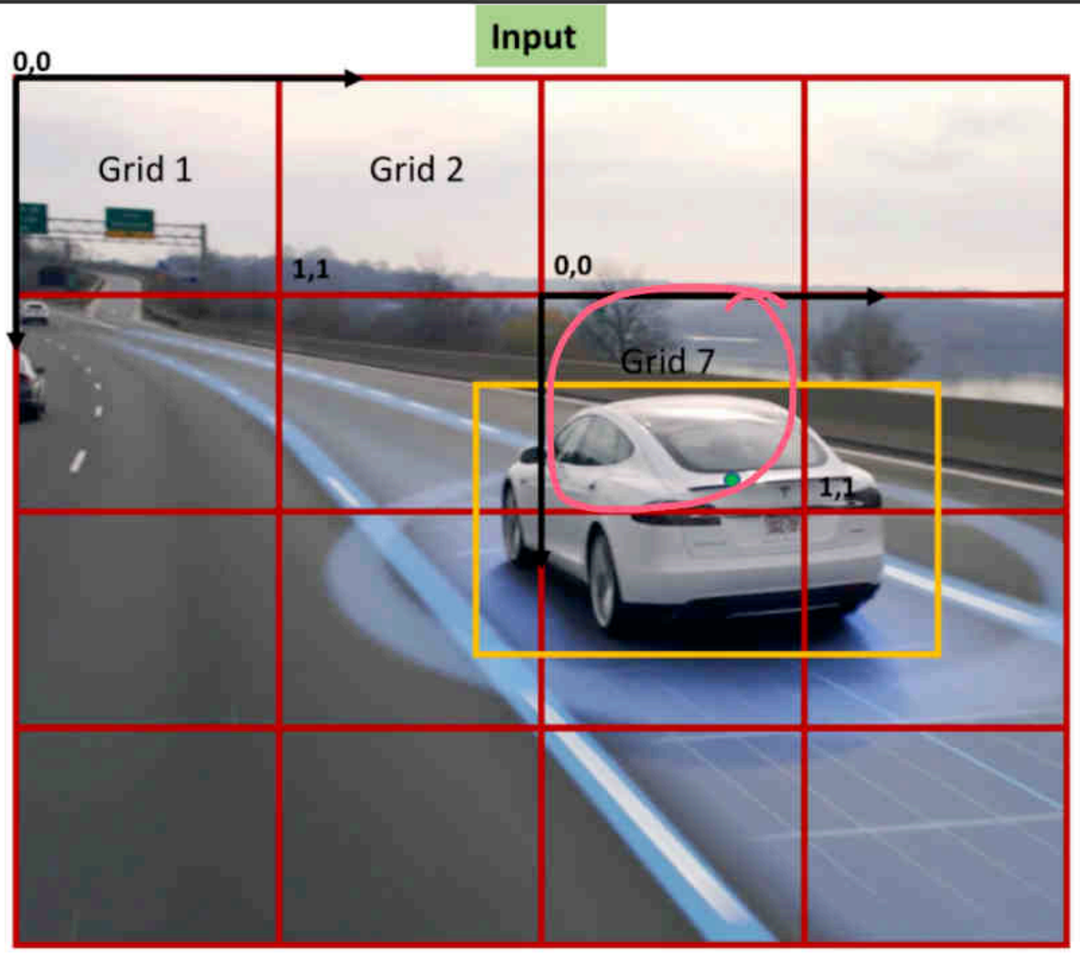


Sample output

Grid 7
P.Object : 1
Bx : 0.8
By : 0.9
Bw : 1.5
Bh : 1.2
P.Car : 1
P.Light : 0
P.Pedestrian : 0

P.Object : 0 for all other grids and we don't care about other values if P.Object is 0





2150 x 1613 x 3 x 16

Sample output

Grid 7
P.Object : 1
Bx : 0.8
By : 0.9
Bw : 1.5
Bh : 1.2
P.Car : 1
P.Light : 0
P.Pedestrian : 0

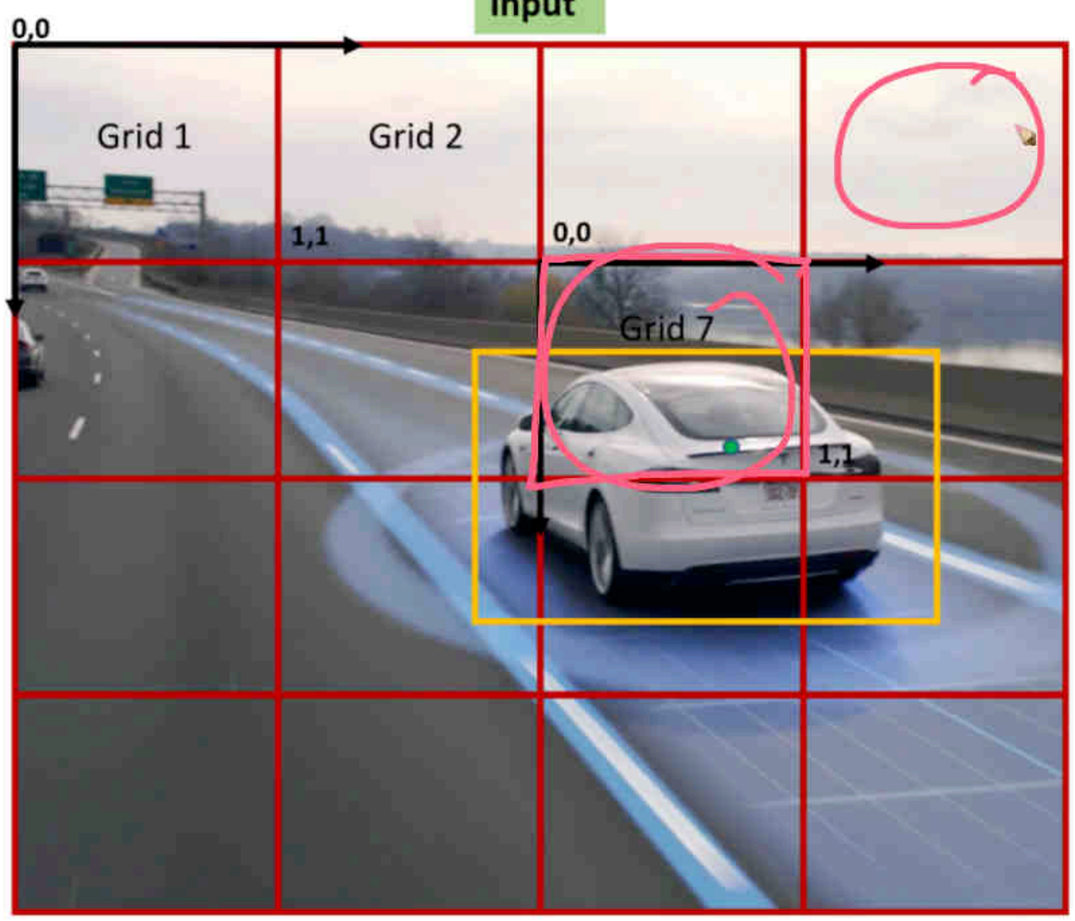
P.Object : 0 for all other grids and we don't care about other values if P.Object is 0

Output

- Grid 1 P.object
- Grid 1 Bx
- Grid 1 P (Car)
- Grid 1 P (Light)
- Grid 1 P (Pede)
- Grid 2 P.object
- Grid 16 P (Light)
- Grid 16 (Pede)

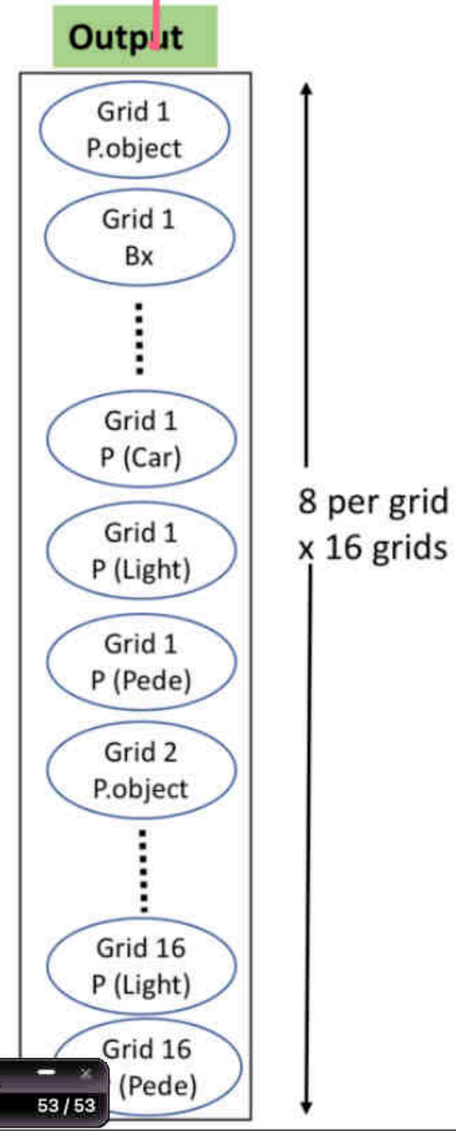
8 per grid x 16 grids

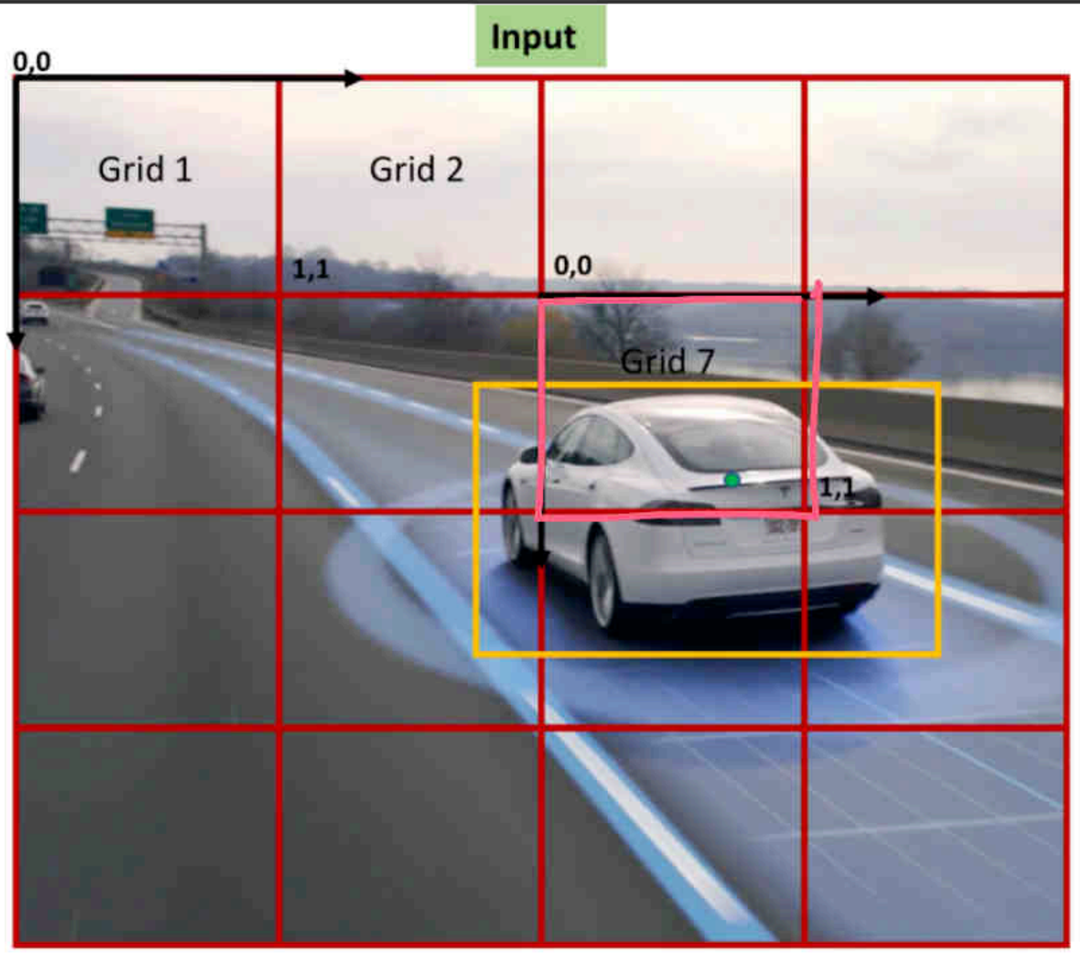
2 anchor boxes per grid cell



Sample output
Grid 7
P.Object : 1
Bx : 0.8
By : 0.9
Bw : 1.5
Bh : 1.2
P.Car : 1
P.Light : 0
P.Pedestrian : 0

P.Object : 0 for all other grids and we don't care about other values if P.Object is 0

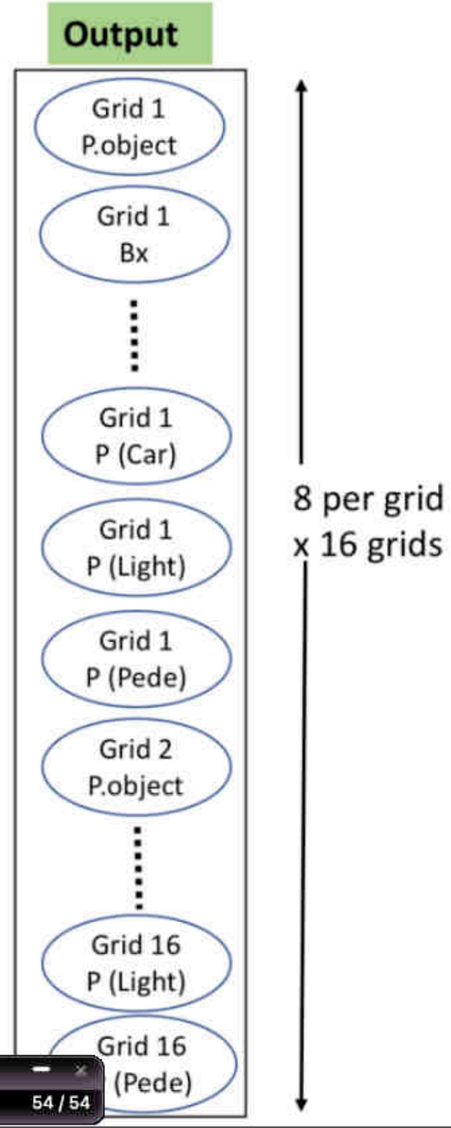


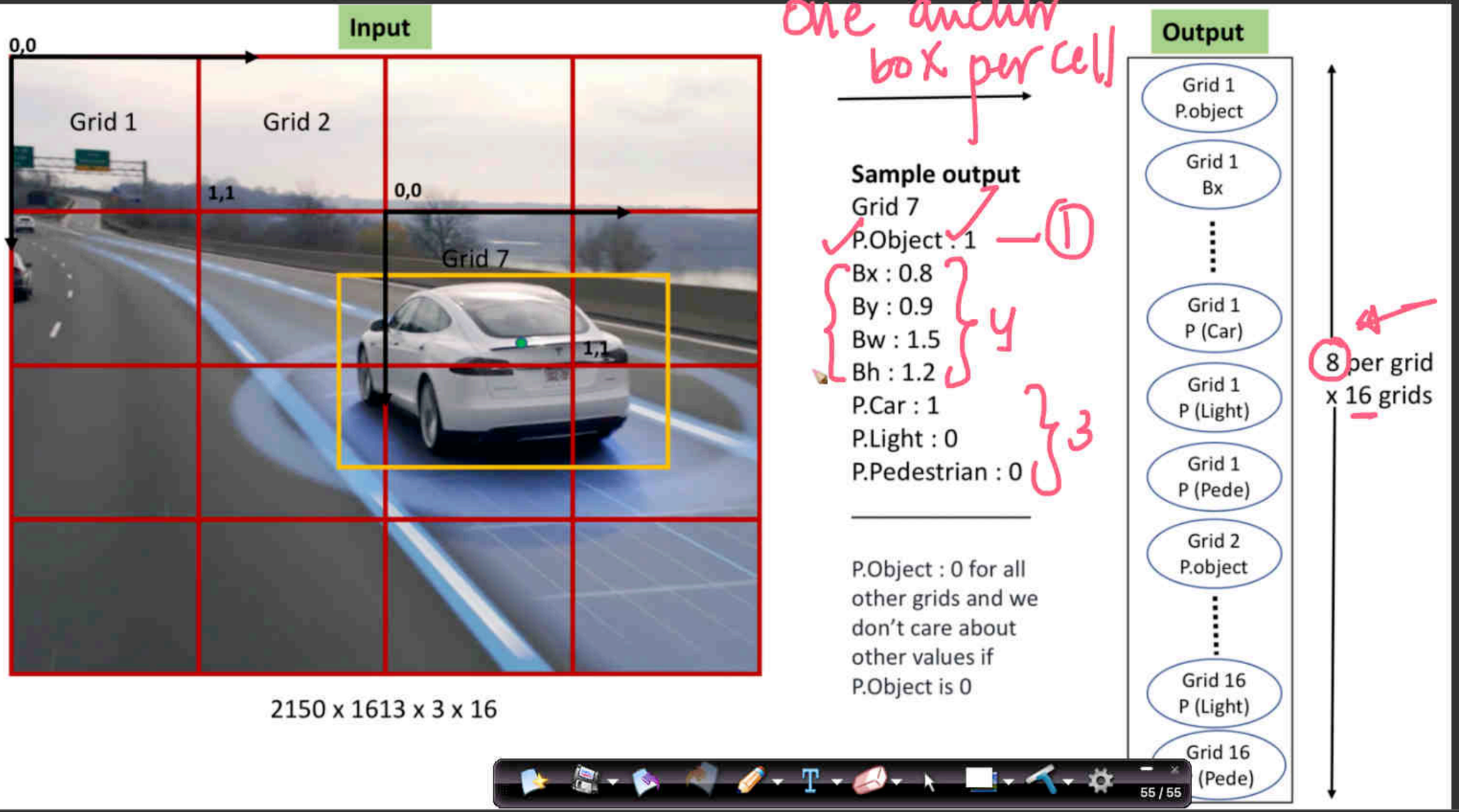


Sample output

Grid 7
P.Object : 1
Bx : 0.8
By : 0.9
Bw : 1.5
Bh : 1.2
P.Car : 1
P.Light : 0
P.Pedestrian : 0

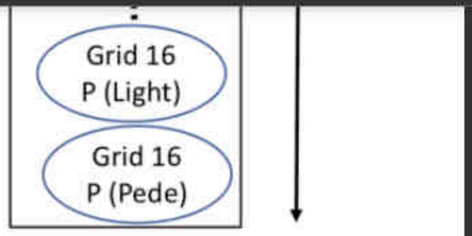
P.Object : 0 for all other grids and we don't care about other values if P.Object is 0





2150 x 1613 x 3 x 16

other values if P.Object is 0



Yolo V3 Architecture details:

As mentioned in the original paper, YOLOv3 has 53 convolutional layers called Darknet-53 is shown in the bellow image. The figure is mainly composed of Convolutional and Residual structures. It should be noted that the last three layers Avgpool, Connected, and softmax layer, are used for classification training on the Imagenet dataset. When using the Darknet-53 layer to extract features from the picture, these three layers are not used:

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 ×

Back-bone
Image → Conv. layers

fastest calculation speed. Compared with ResNet-101, the speed of the Darknet-53 network is 1.5 times that of the former; although ReseNet-152 and its performance are similar, it takes more than two times.

2. it has highest measurement floating-point operation per second, which means that the network structure can better use the GPU, thereby making it more efficient and faster.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

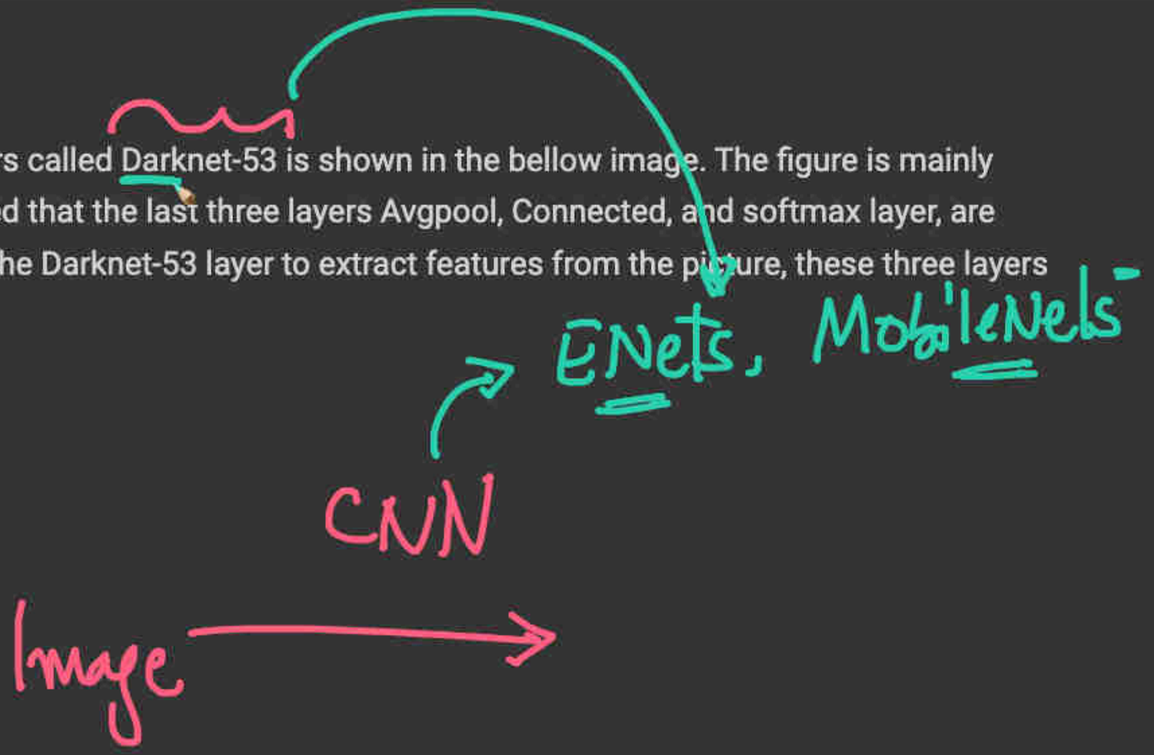
YOLO in easy steps:

1. Divide the image into multiple grids. For illustration, I have drawn 4x4 grids in above figure.
2. Label the training data as shown in the above figure.
   - If **C is number of unique objects** in our data, `S*S is number of grids into which we split our image`, then our output vector will be of length `S*S*(C+5)`.
   - **For e.g.** in above case, our target vecto ng for `3 unique`

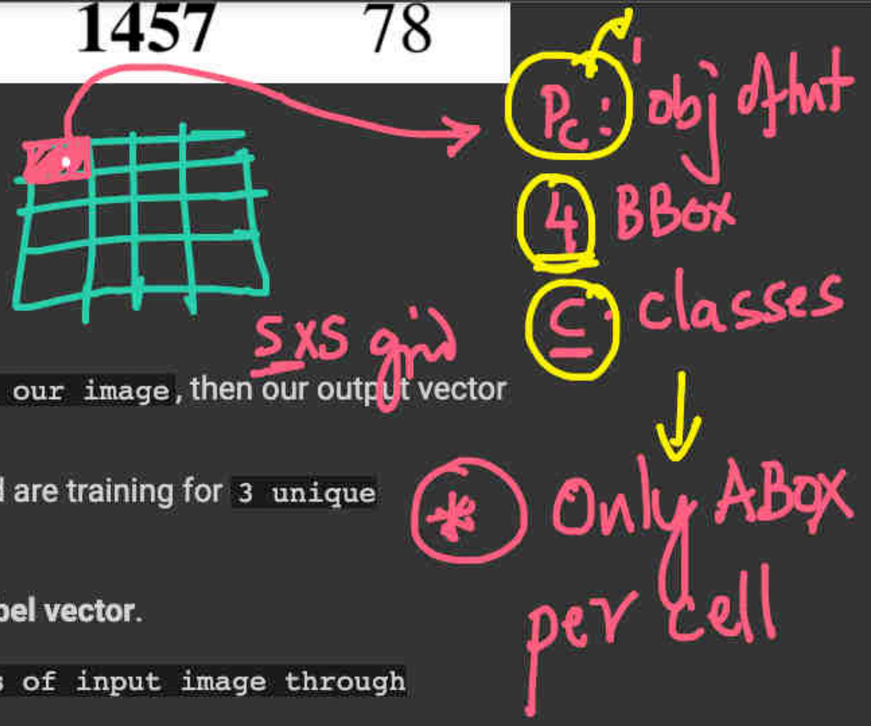
Yolo V3 Architecture details:

As mentioned in the original paper, YOLOv3 has 53 convolutional layers called Darknet-53 is shown in the bellow image. The figure is mainly composed of Convolutional and Residual structures. It should be noted that the last three layers Avgpool, Connected, and softmax layer, are used for classification training on the Imagenet dataset. When using the Darknet-53 layer to extract features from the picture, these three layers are not used:

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 ×



Darknet-53	77.2	**93.8**	18.7	**1457**	78



YOLO in easy steps:

1. Divide the image into multiple grids. For illustration, I have drawn 4x4 grids in above figure.
2. Label the training data as shown in the above figure.
   - If **C is number of unique objects** in our data, `S*S is number of grids into which we split our image`, then our output vector will be of length `S*S*(C+5)`.
   - **For e.g.** in above case, our target vector is 4x4x(3+5) as we divided our images into 4*4 grids and are training for `3 unique objects: Car, Light and Pedestrian`.
3. Make one deep convolutional neural net with **loss function as error between output activations and label vector**.
   - Basically, `the model predicts the output of all the grids in just one forward pass of input image through ConvNet`.
4. **Keep in mind that the label for object being present in a grid cell (P.Object) is determined by the presence of object's centroid in that grid.**
   - This is important to not allow one object to be counted multiple times in different grids.

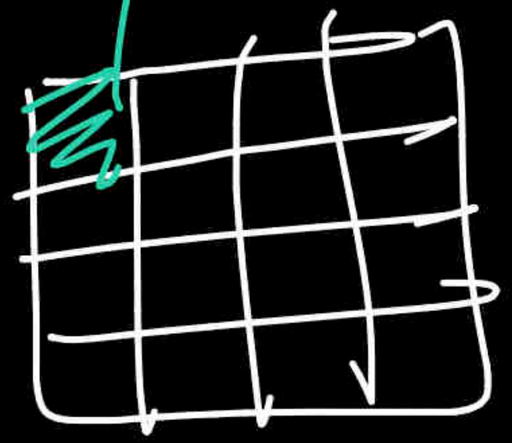
The **main advantage Of YOLO** is, it is very **fast** as the model predicts the output of all the grids in just `one forward pass of input image through ConvNet.`

9

4x4

Case 1

1 PC → 4BBox → S + C outputs

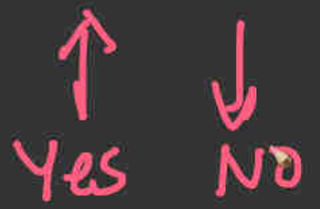


SXS

SXS x (S+C)

YOLO in easy steps:

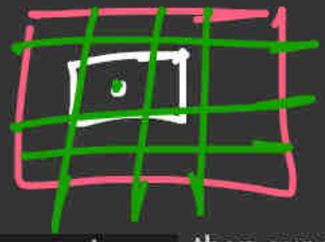
1. Divide the image into multiple grids. For illustration, I have drawn 4x4 grids in above figure.
2. Label the training data as shown in the above figure.
   - If **C is number of unique objects** in our data, `S*S is number of grids into which we split our image`, then our output vector will be of length `S*S*(C+5)`.
   - **For e.g.** in above case, our target vector is 4x4x(3+5) as we divided our images into 4*4 grids and are training for `3 unique objects: Car, Light and Pedestrian`.
3. Make one deep convolutional neural net with **loss function as error between output activations and label vector**.
   - Basically, `the model predicts the output of all the grids in just one forward pass of input image through ConvNet`.
4. **Keep in mind that the label for object being present in a grid cell (P.Object) is determined by the presence of object's centroid in that grid.**
   - This is important to not allow one object to be counted multiple times in different grids.



The **main advantage Of YOLO** is, it is very **fast** as the model predicts the output of all the grids in just `one forward pass of input image through ConvNet.`

YOLO Loss:

YOLO in easy steps:



1. Divide the image into multiple grids. For illustration, I have drawn 4x4 grids in above figure.
2. Label the training data as shown in the above figure.
   - If **C is number of unique objects** in our data, `S*S is number of grids into which we split our image`, then our output vector will be of length `S*S*(C+5)`.
   - **For e.g.** in above case, our target vector is 4x4x(3+5) as we divided our images into 4*4 grids and are training for `3 unique objects: Car, Light and Pedestrian`.
3. Make one deep convolutional neural net with **loss function as error between output activations and label vector**.
   - Basically, `the model predicts the output of all the grids in just one forward pass of input image through ConvNet`.
4. **Keep in mind that the label for object being present in a grid cell (P.Object) is determined by the presence of object's centroid in that grid.**
   - This is important to not allow one object to be counted multiple times in different grids.

The **main advantage Of YOLO** is, it is very **fast** as the model predicts the output of all the grids in just `one forward pass of input image through ConvNet.`

YOLO Loss:

YOLO in easy steps:

1. Divide the image into multiple grids. For illustration, I have drawn 4x4 grids in above figure.
2. Label the training data as shown in the above figure.
   - If **C is number of unique objects** in our data, `S*S is number of grids into which we split our image`, then our output vector will be of length `S*S*(C+5)`.
   - **For e.g.** in above case, our target vector is 4x4x(3+5) as we divided our images into 4*4 grids and are training for `3 unique objects: Car, Light and Pedestrian`.
3. Make one deep convolutional neural net with **loss function as error between output activations and label vector**.
   - Basically, `the model predicts the output of all the grids in just one forward pass of input image through ConvNet`.
4. **Keep in mind that the label for object being present in a grid cell (P.Object) is determined by the presence of object's centroid in that grid.**
   - This is important to not allow one object to be counted multiple times in different grids.

The **main advantage Of YOLO** is, it is very **fast** as the model predicts the output of all the grids in just `one forward pass of input image through ConvNet.`

YOLO Loss:

YOLO in easy steps:

1. Divide the image into multiple grids. For illustration, I have drawn 4x4 grids in above figure.
2. Label the training data as shown in the above figure.
   - If **C is number of unique objects** in our data, `S*S is number of grids into which we split our image`, then our output vector will be of length `S*S*(C+5)`.
   - **For e.g.** in above case, our target vector is 4x4x(3+5) as we divided our images into 4*4 grids and are training for `3 unique objects: Car, Light and Pedestrian`
3. Make one deep convolutional neural net with **loss function as error between output activations and label vector**.
   - Basically, `the model predicts the output of all the grids in just one forward pass of input image through ConvNet`.
4. **Keep in mind that the label for object being present in a grid cell (P.Object) is determined by the presence of object's centroid in that grid.**
   - This is important to not allow one object to be counted multiple times in different grids.

The **main advantage Of YOLO** is, it is very **fast** as the model predicts the output of all the grids in just `one forward pass of input image through ConvNet.`

YOLO Loss:

YOLOV3

Image size
object size

Image
(grid 5×5)

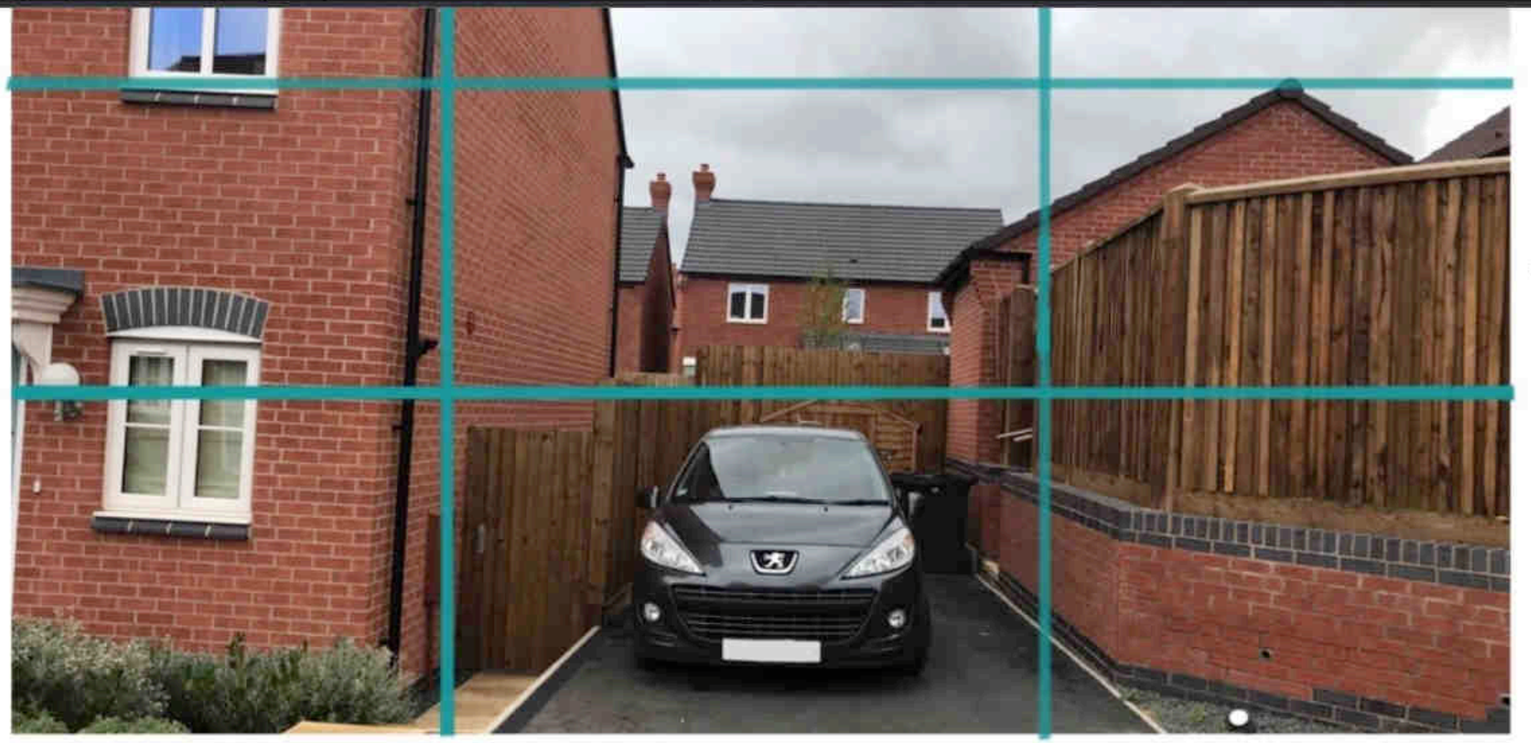
back-bone
pretrain model
(Darknet-53)
Conv

GAP
FC

5×5×(5+C)

grid
Anchor-box

log-loss ← P_0:
reg-loss ← B_x, B_y, B_h, B_w
CE-loss ← $C_1\ C_2\ C_3 \cdots C_C$



$\lambda_{noobj} = 0,5$

j

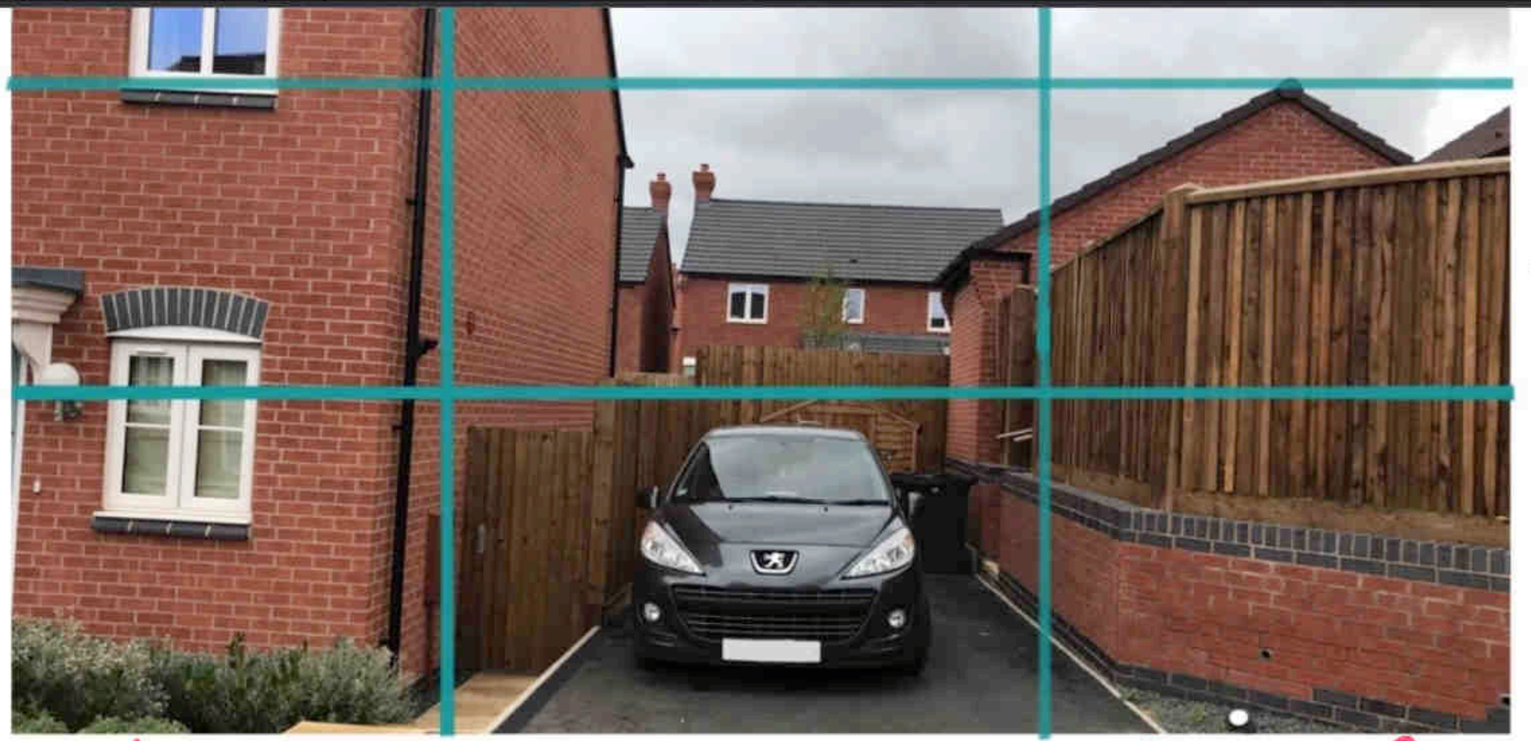
$$Loss_{yolo} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] + \longrightarrow \text{Bounding Box coord}$$

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left[(C_i - \hat{C}_i)^2 + \longrightarrow \text{Confidence} \right. \right.$$

$$\sum_{i=0}^{S^2} 1_{ij}^{noobj} \sum_{C \in classes} \left[(p_i(C) - \hat{p}_i(C))^2 \right] \longrightarrow \text{Classification}$$

$1_{ij}^{noobj} \longrightarrow$ 1 if box j and cell i no match.
0 otherwise

$\lambda_{noobj} = 0,5$



j

$$Loss_{yolo} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 \right] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] + \longrightarrow \text{Bounding Box coord}$$

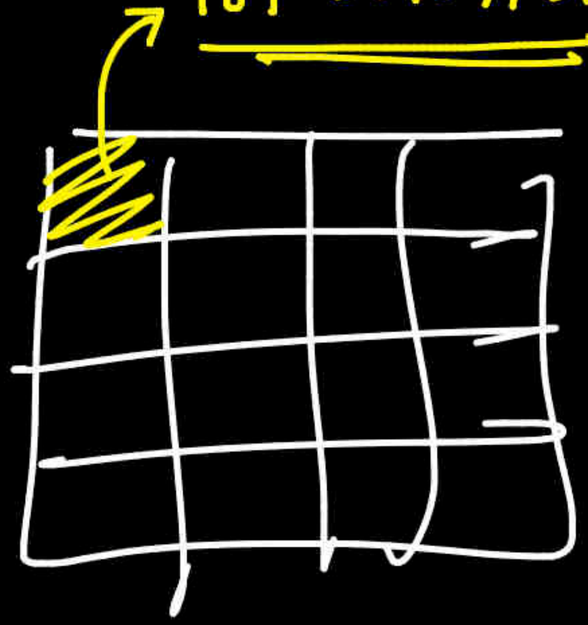
$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(C_i - \hat{C}_i\right)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left[\left(C_i - \hat{C}_i\right)^2 + \longrightarrow \text{Confidence} \right. \right.$$

$$\sum_{i=0}^{S^2} 1_{ij}^{noobj} \sum_{C \in classes} \left[\left(p_i(C) - \hat{p}_i(C)\right)^2 \right] \longrightarrow \text{Classification}$$

$1_{ij}^{noobj} \longrightarrow$ 1 if box j and cell i no match.
0 otherwise

Q

P_0, B_x, B_y, B_h, B_w ⊆ pack



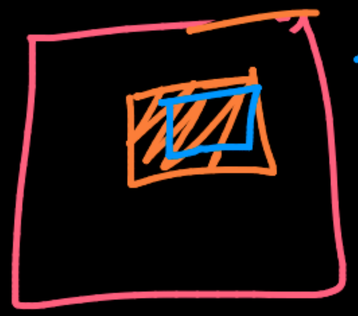
$S \times S \times (C + 5)$

one anchor box per cell

Q

Co-ordinates

Train



CNN ✓

B_x, B_y, B_h, B_w

$l_0, 120, 20, 40$

→ Cl. head

GAP

FC

→ Reg. head

$\hat{l_0}, \hat{121}, \hat{12}, \hat{31}$

$\hat{B_x}$ $\hat{B_y}$ $\hat{B_h}$ $\hat{B_w}$

L2-loss

(MSE)

Problem Statement:

You are working as a Machine learning Engineer in Tesla as part of the Autonomous driving team focused on building prototype ML model to identify and detect location of:

- Traffic Lights, Stop signs
- Other Vehicles such as Car, Bicycles, Truck etc
- Pedestrian(People)
- Animals

on the road in realtime using the Camera installed in the deck of the owners car.

Real time constraints:

- Our Object Detection Algorithm needs to be Fast enough to **run inference in RealTime** on commodity hardware(CPU only) and **highly accurate to ensure safety.**

Let's see how we can achieve this Balance using new family of Object Detection Algorithms: **Single Stage Detectors**.