Object Classification: An Introduction

Prof. Dr. Shamim Akhter

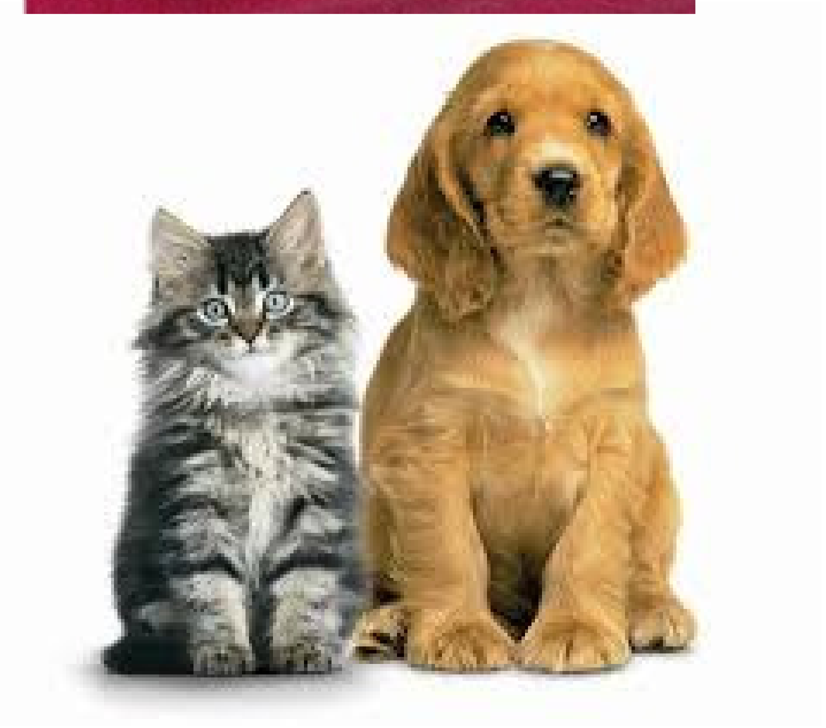
Professor, Dept. of CSE

Ahsanullah University of Science and Technology

Object Classification

CAT

DOG



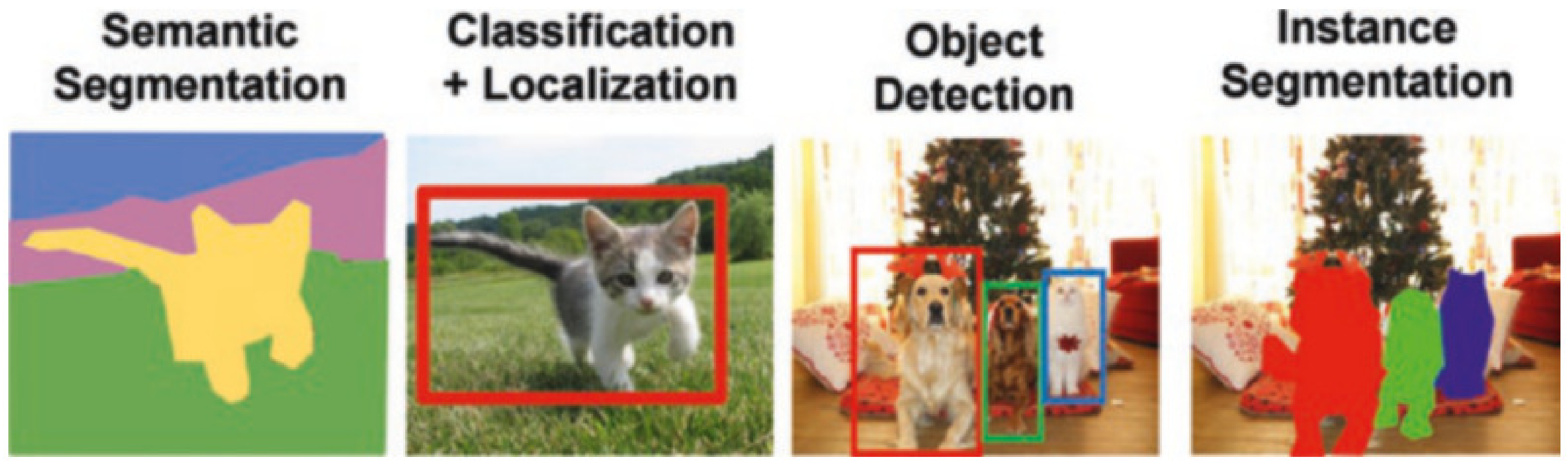
?

Typically, recognizing that you have many instances of an object in an image involves finding their position in an image and being able to distinguish between them.
To do that, we need to be able to find the positions of each instance in the image and their borders.

Object localization(single)/detection(multiple)

Object Localization

- Determine the location of one or more objects (for example, people or cars) in an image and draw a rectangular bounding box around it.
  - Localization typically refers to when an image contains only one instance of an object, while detection is when several instances of an object in an image.

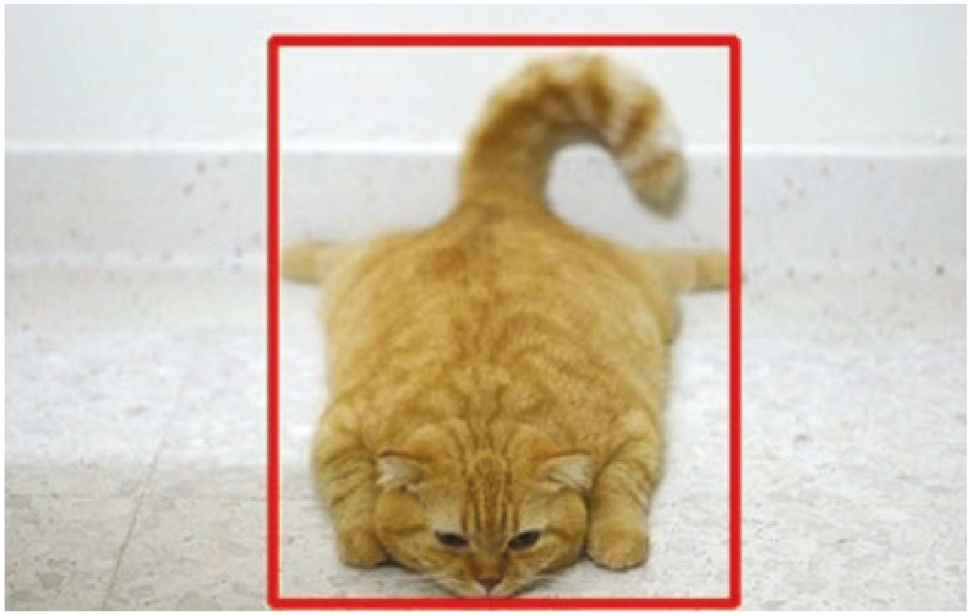


- **Classification:** Give a label to an image, or in other words, “understand” what is in an image.
  - For example, an image of a cat may have the label “cat”.
- **Classification and localization:** Give a label to an image and determine the borders of the object contained in it (and typically draw a rectangle around the object).
- **Object detection:** This term is used when you have multiple instances of an object in an image.
  - In object detection, you want to determine all the instances of several objects (for example, people, cars, signs, etc.) and draw bounding boxes around them.
- **Instance segmentation:** You want to label each pixel of the image with a specific class for each separate instance, to be able to find the exact limits of the object instance.

- **Semantic segmentation:** You want to label each pixel of the image with a specific class.
  - The difference with instance segmentation is that you don’t care if you have several instances of a car as examples. All pixels belonging to the cars will be labelled as “car”.
  - In instance segmentation, you will still be able to tell how many instances of a car you have and where they are exactly.

Object Classification and Localization

We will assume that in the images we have only one instance of a specific object, and the task is to determine what kind of object it is and draw a bounding box (a rectangle) around it.



How to create a bounding box around the object?

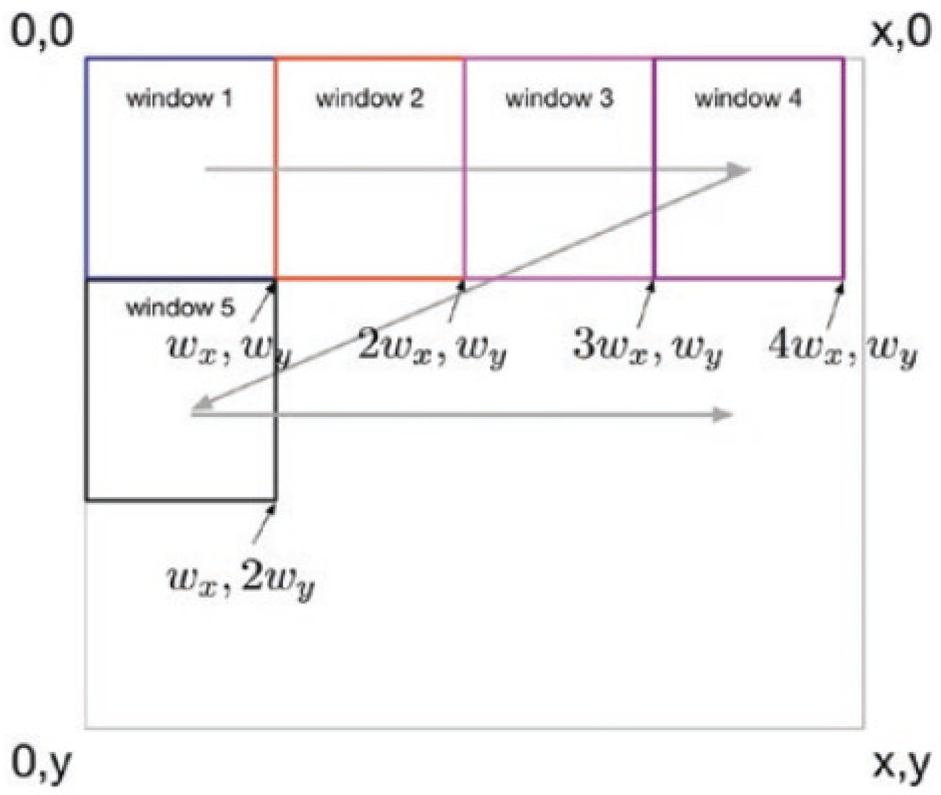
Sliding Window Approach

- Cut a small portion of your input image(x,y) starting from the top-left corner. Has dimensions wx, wy, with wx < x and wy<y.

- Use a pre-trained network and let it classify the image portion that you cut.

- Now shift this window by an amount we call stride **S** toward the right and then below. You use the network to classify this second portion.

- Once the sliding window has covered the entire image, you choose the position of the window that gives you the **highest classification probability.** This position will give you the bounding box of your object.



A graphical illustration of the sliding window approach

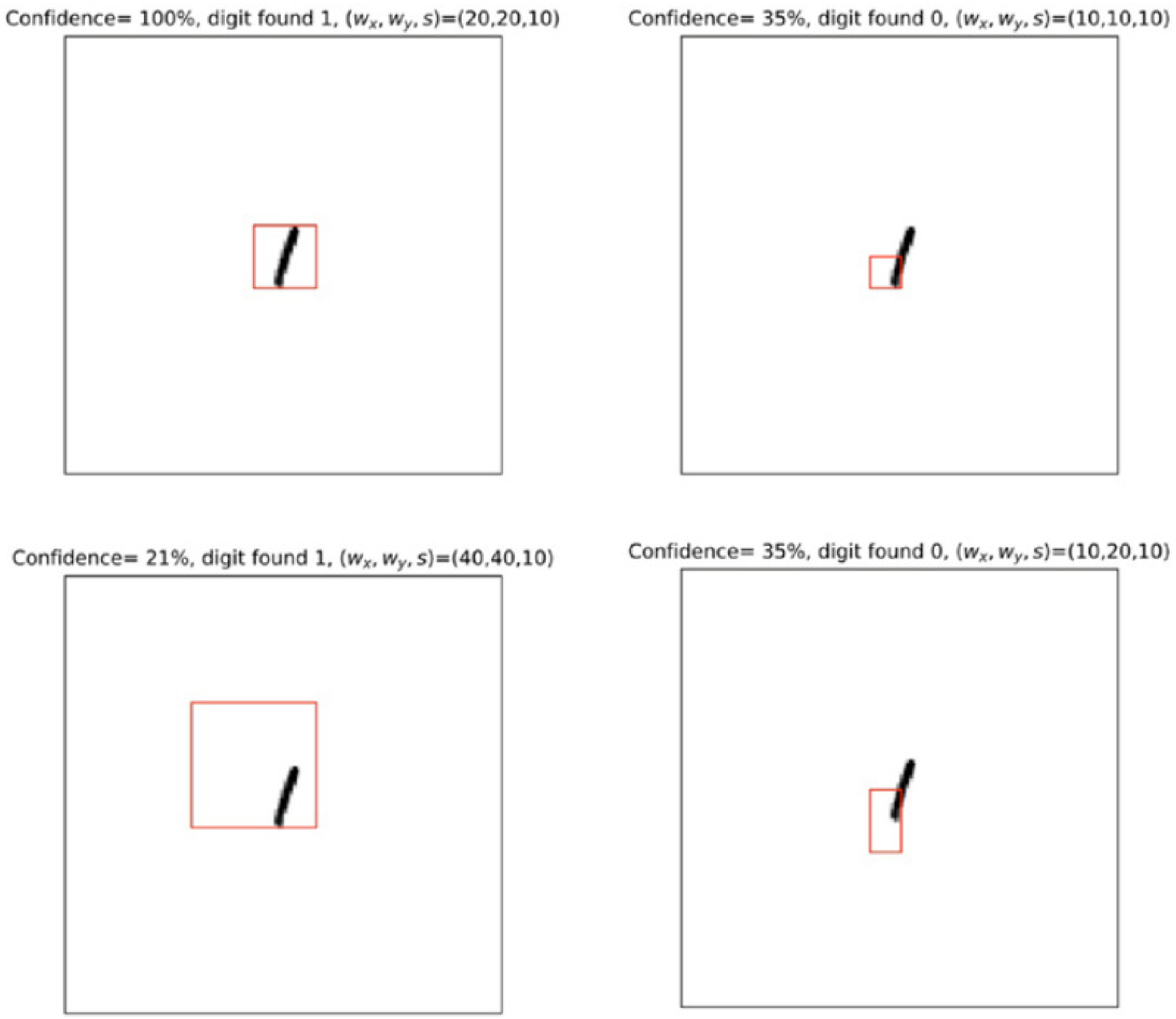
Problems and Limitations: Sliding Window Approach

- Depending on the choice of wx, wy, and s, we may not be able to cover the entire image.
- How do you choose wx, wy, and s? What if the object is larger or smaller?
- What if our object flows across two windows?

We could solve the third problem by using s = 1 to be sure that we cover all possible cases, but the first two problems are not so easy to solve.

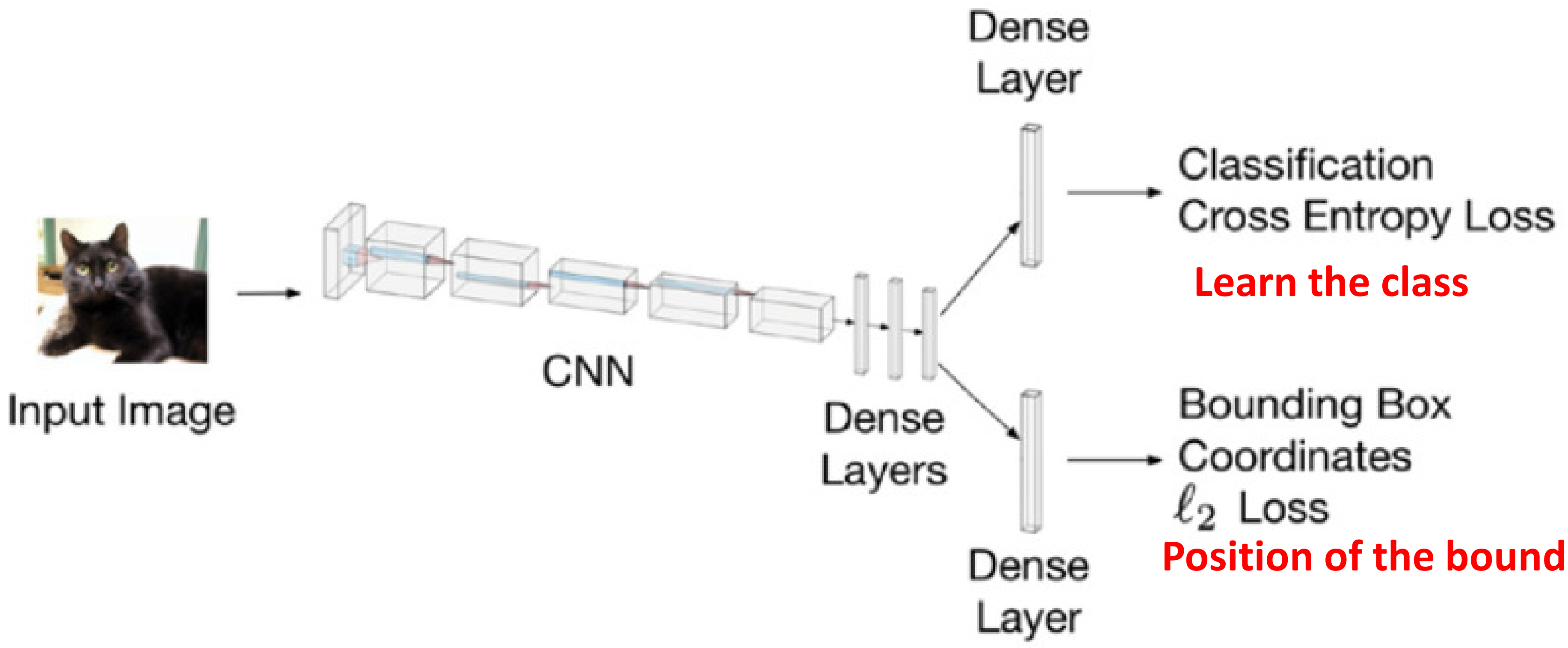
To address the window size problem, we should try all possible sizes and all possible proportions.

Do you see any problem here?
The number of evaluations that you will need to do with your network is getting out of control and will become quickly computationally infeasible.


Confidence= 100%, digit found 1, (w_x, w_y, s)=(20,20,10)
Confidence= 35%, digit found 0, (w_x, w_y, s)=(10,10,10)
Confidence= 21%, digit found 1, (w_x, w_y, s)=(40,40,10)
Confidence= 35%, digit found 0, (w_x, w_y, s)=(10,20,10)

Multi-task Learning

A better approach is to use multi-task learning. The idea is that we can build a network that will learn at the same time the class and the position of the bounding box.


Input Image
CNN
Dense Layers
Dense Layer
Classification Cross Entropy Loss
Dense Layer
Bounding Box Coordinates ℓ_2 Loss

Learn the class

Position of the bounding box

Need to minimize a linear combination of the two loss functions:

$$J_{classification} + \alpha J_{BB}$$

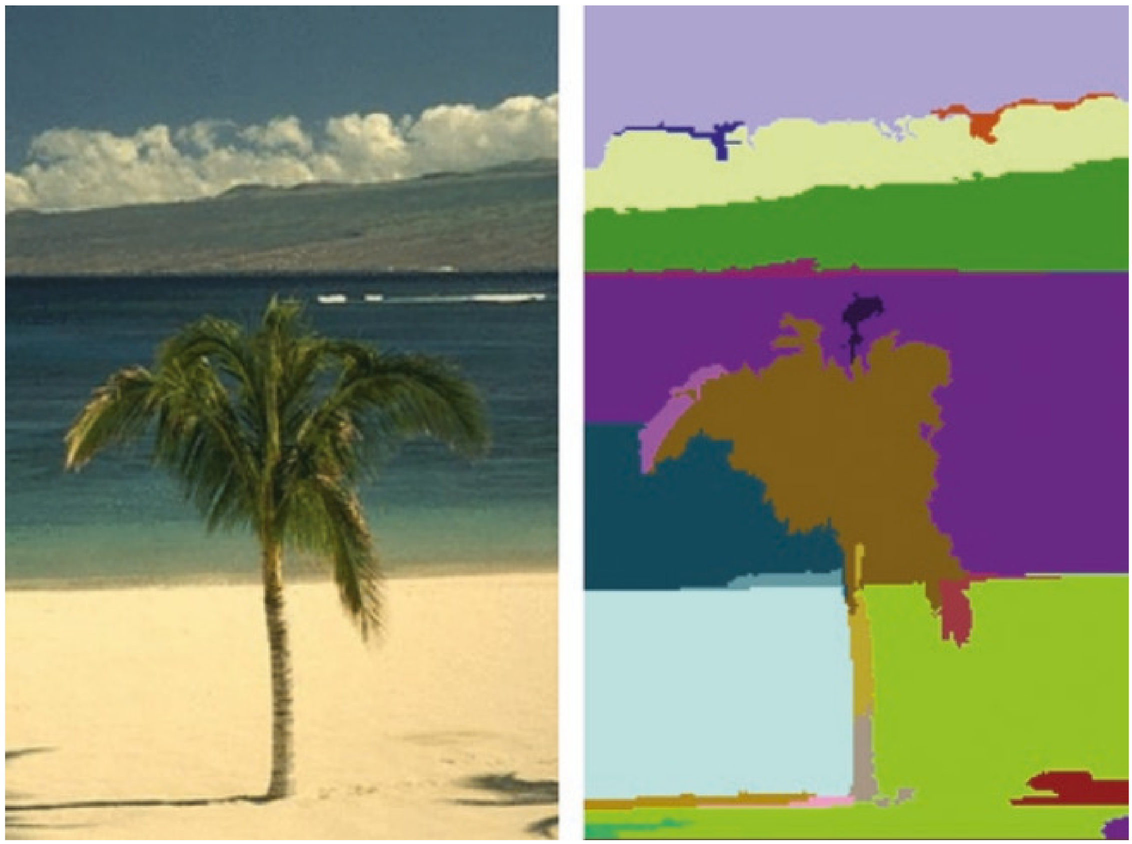
α is an additional hyper-parameter that needs to be tuned.

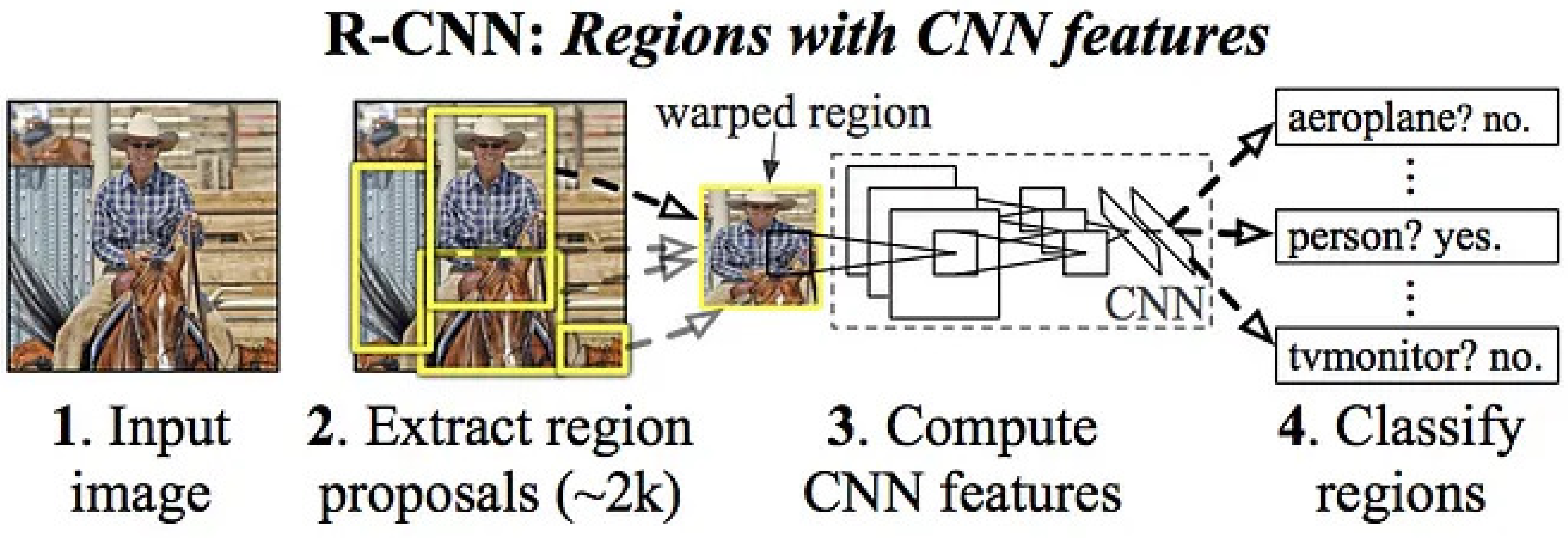
$$\ell_2 \; Loss\ Function = \sum_{i=1}^{m} \left(y_{true}^{(i)} - y_{predicted}^{(i)} \right)^2$$

Region Based CNN(R-CNN)

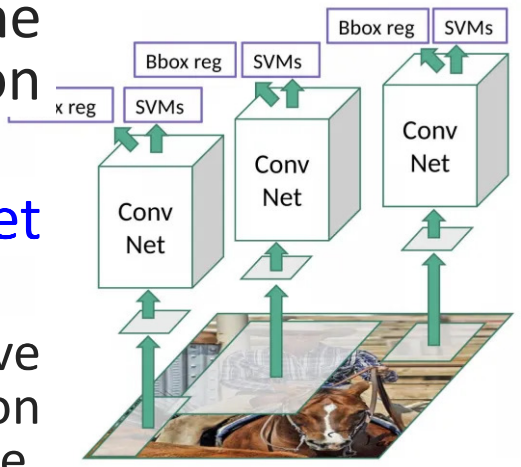
- Girshick proposed a Selective search: first propose 2000 regions from the image and then, instead of classifying a huge number of regions, they classified just those 2000 regions.
- uses a classical approach to determine which regions may contain an object.

- ❑ The first step in the algorithm is to segment an image, using pixel intensities and graph-based methods
- ❑ After this step, adjacent regions are grouped based on similarities of the following features: Color similarity, Texture similarity, Size similarity, and Shape compatibility.
- ❑ Produces 2000 candidate region proposals





- These 2000 candidate region proposals are warped into a square and fed into a pre-trained convolutional neural network (Alexnet) that produces a 4096-dimensional feature vector as output.
- The extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal.
- The algorithm also predicts four(4) values which are offset values to increase the precision of the bounding box.
  - For example, given a region proposal, the algorithm would have predicted the presence of a person but the face of that person within that region proposal could've been cut in half. Therefore, the offset values help in adjusting the bounding box of the region proposal.



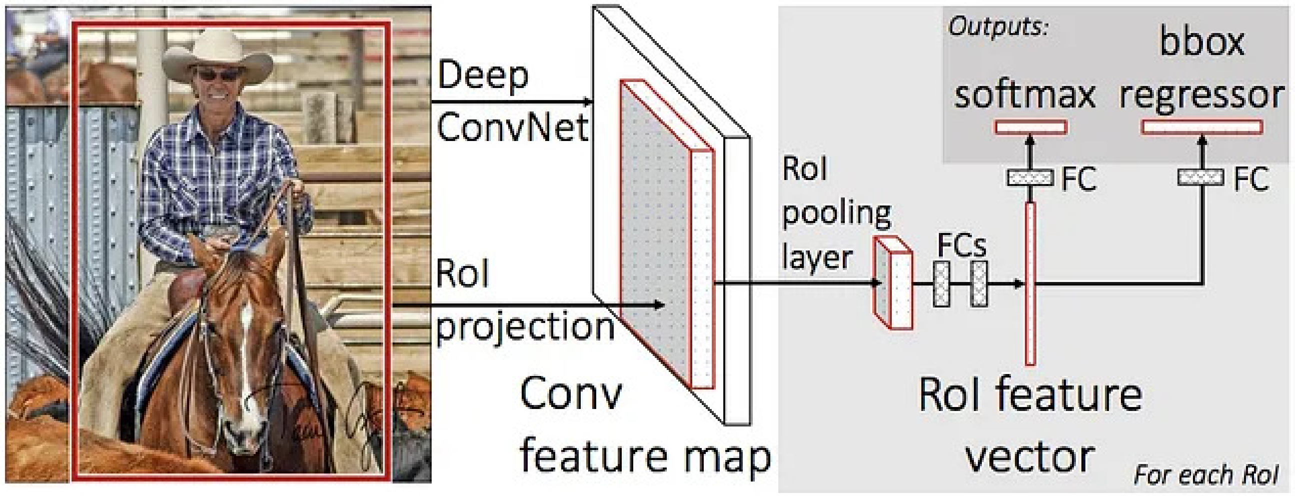
Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.

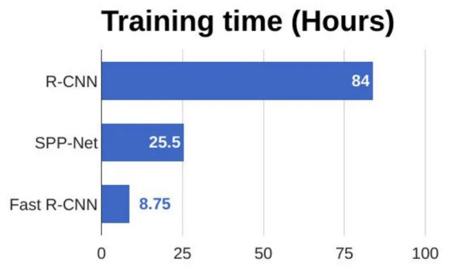
- It cannot be implemented in real time as it takes around 47 seconds for each test image.

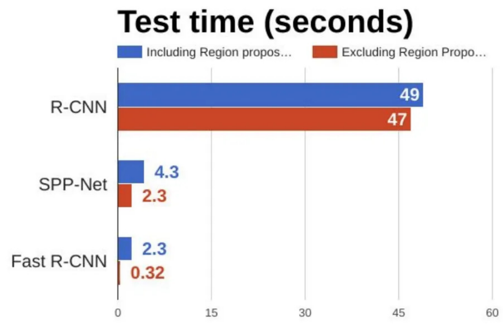
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

Fast R-CNN



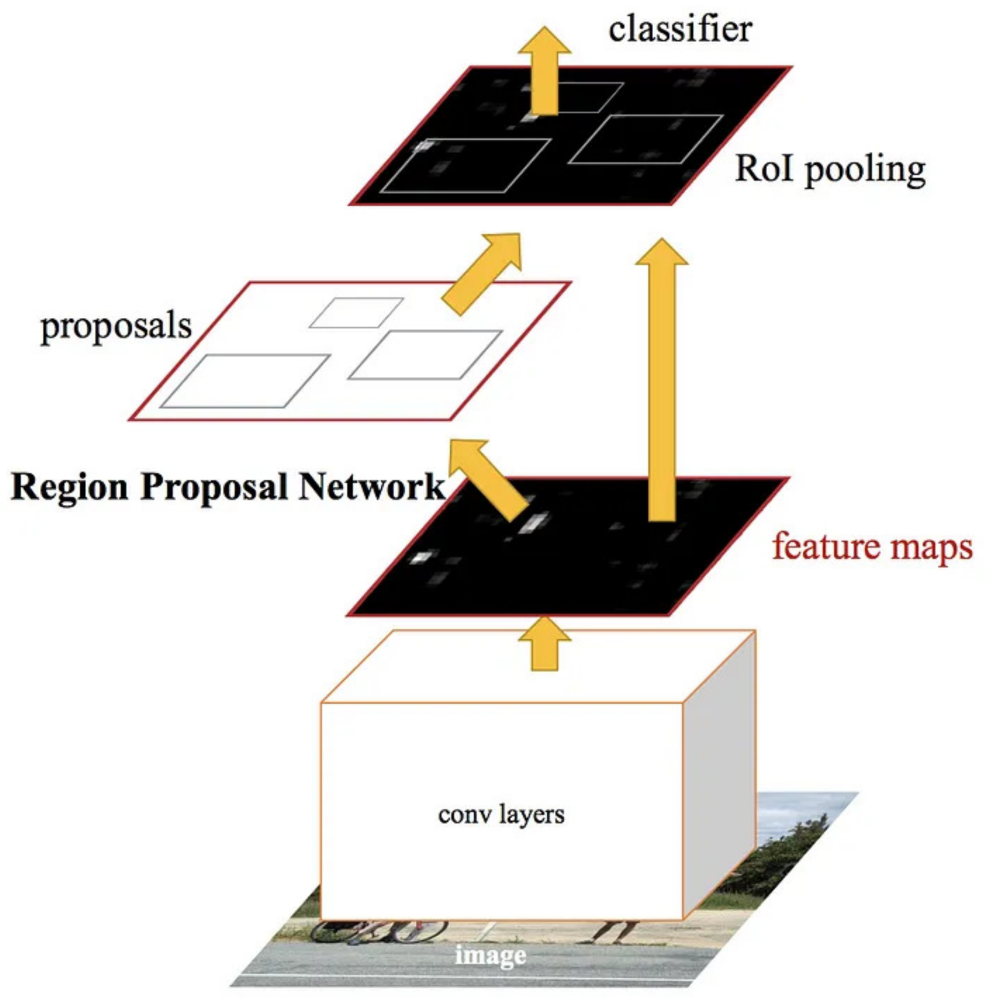
- The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map.
- From the convolutional feature map, we identify the region of proposals and warp them into squares, and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.
- From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.

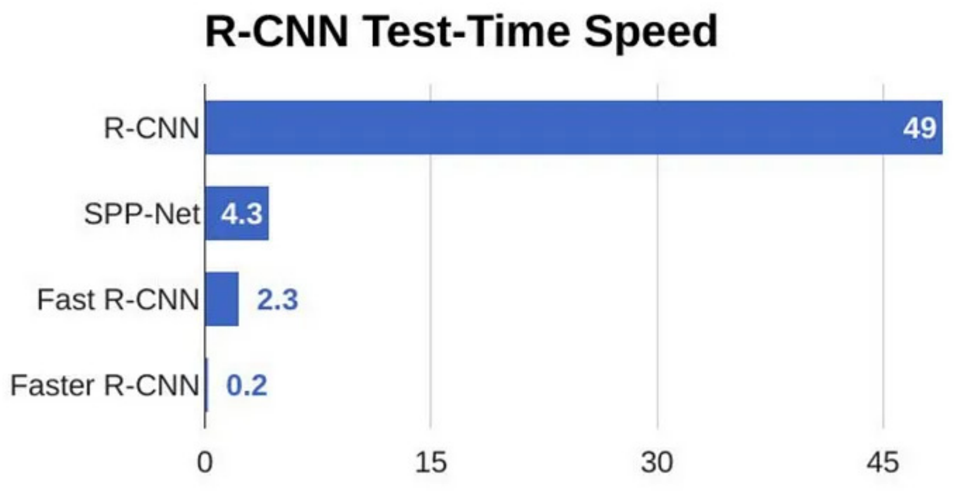




Faster R-CNN

- Shaoqing Ren et al. developed an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.
- Similar to Fast R-CNN, the image is provided as input to a convolutional network, providing a convolutional feature map.
- Instead of using a selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals.
- The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.





You Only Look Once (YOLO) Method

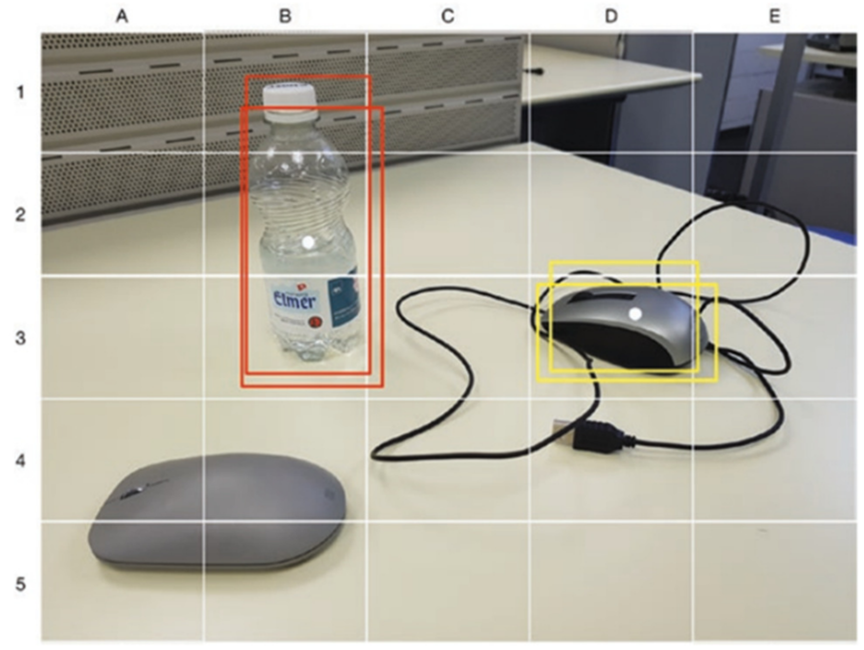
- In 2015, Redmon J. et al. proposed a new method to do object detection: they called it YOLO (You Only Look Once).
- This method is fast and is used often in real-time applications
  - The network can perform all the necessary tasks (detect where the objects are, classify multiple objects, etc.) in one pass.

- The main idea of the method is to reframe the detection problem as one single regression problem,
  - from the pixels of the image as inputs, to the bounding box coordinates and class probabilities.

How YOLO Works

- Dividing the Image Into Cells
  - The first step is to divide the image into S × S cells. For each cell, we predict what (and if an) object is in the cell.
    - Only one object will be predicted for each cell, so one cell cannot predict multiple objects.
  - Then for each cell, a certain number (B) of bounding boxes that should contain the objects are predicted.

    The likelihood of each object class (C).
  - And predicts a class confidence (a number) for each bounding box.



Let's take as an example cell D3 in the given Figure. This cell will predict the presence of a mouse and then it will predict a certain number B of bounding boxes (the yellow rectangles). Similarly, cell B2 will predict the presence of the bottle and B bounding boxes (the red rectangles) all at the same time.

- For each bounding box (B in total), there are four values: x, y, w, h. These are the position of the center, its width, and its height. Note that the position of the center is given with a relationship to the cell position, not as an absolute value.

- For each bounding box (B in total), there is a confidence score, which is a number that reflects how likely the box contains the object. In particular, at training time, if we indicate the probability of the cell containing the object as Pr(Object), the confidence is calculated as follows:

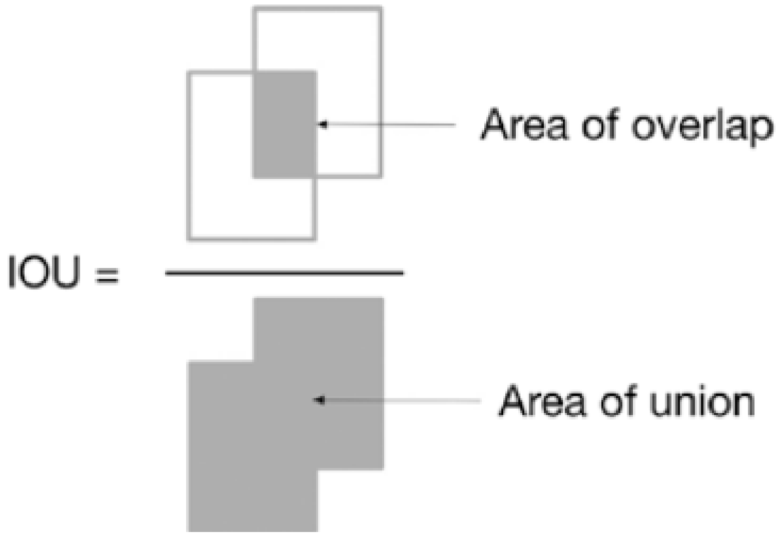
$$\Pr(Object) \times IOU$$

Where IOU indicates the Intersection Over Union, which is calculated using the training data

IOU (Intersect Over Union)

- This is a fully supervised task.
- This means that we will need to learn where the bounding boxes are and compare them to some given ground truth. We need a metric to quantify how good the overlap is between the predicted bounding boxes and the ground truth. This is typically done with the IOU (Intersect Over Union) .

$$IOU = \frac{Area\ of\ overlap}{Area\ of\ union}$$



In the ideal case of perfect overlap, we have IOU = 1, while if there is no overlap at all, we have IOU = 0.

Therefore, supposing we have $S = 5$, $B = 2$ and supposing the network can classify $N_c = 80$ classes, the network will have an output of size of the following:

$$S \times S \times (B \times 5 + N_c) = 5 \times 5 \times (2 \times 5 + 80) = 2250$$

?

In the original paper, the authors used $S = 7$, $B = 2$ and used the VOC dataset[2] with 20 labelled classes. Therefore, the output of the network was as follows:

$$S \times S \times (B \times 5 + N_c) = 7 \times 7 \times (2 \times 5 + 20) = 1470$$

?

In the original paper, the authors were inspired by the GoogLeNet model. The network has 24 layers followed by two dense layers (the last one having 1470 neurons; do you see why?).

4 box parameters 1 cs

Non-Maxima Suppression

Once you have all the predicted bounding boxes, you need to choose the best one. Remember that for each cell and object, the model predicts several bounding boxes (regardless of which version you use). Basically, you choose the best bounding box by following this procedure (called *non-maxima suppression*):

1. It first discards all cells in which the probability of an object being present is less than a given threshold (typically 0.6).

2. It takes all the cells with the highest probability of having an object inside.

3. It takes the bounding boxes that have the highest score and removes all other bounding boxes that have an IOU greater than a certain threshold (typically 0.5) with each other. That means that it removes all bounding boxes that are very similar to the chosen one.

Classification Loss

The classification loss used is determined by

$$\sum_{i=0}^{S^2} \mathbb{I}_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

Where
$\mathbb{I}_i^{obj}$ *is* 1 if an object is in cell *i*, or 0 otherwise.
$\hat{p}_i(c)$ denotes the probability of having class *c* in cell *i*.

Localization Loss

This loss measures the error of the predicted bounding boxes with respect to the expected ones.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_i^{obj} \left[\left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 \right] +$$
$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_i^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right]$$

Confidence Loss

The confidence loss measures the error when deciding if an object is in the box or not.

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$

Where

$\hat{C}_i$ is the confidence of the box *j* in cell *i*.

$\mathbb{I}_{ij}^{obj}$ *is* 1 if the j^{th} bounding box in cell *i* is responsible for detecting the object.

Since most cells does not contain an object, we must be careful. The network could learn that the background is important. We need to add a term to the cost function to remedy this. This is done with the additional term:

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2$$

Where $\mathbb{I}_{ij}^{noobj}$ is the opposite of $\mathbb{I}_{ij}^{obj}$.

Total Loss Function

The total loss function is simply the sum of all the terms:

$$L = \sum_{i=0}^{S^2} \mathbb{I}_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2 + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_i^{obj} \left[\left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 \right] +$$

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_i^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] +$$

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left(\mathrm{C}_i - \hat{\mathrm{C}}_i\right)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{noobj} \left(\mathrm{C}_i - \hat{\mathrm{C}}_i\right)^2$$