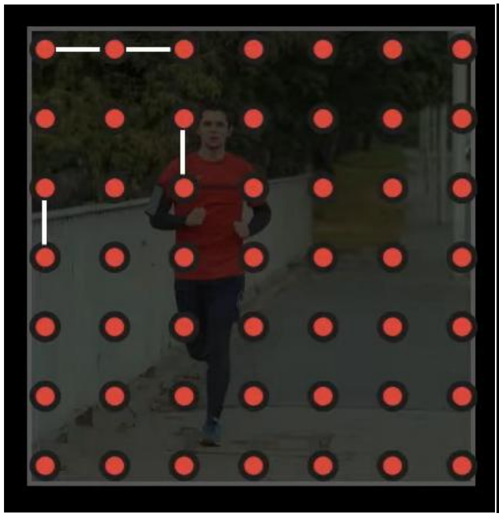
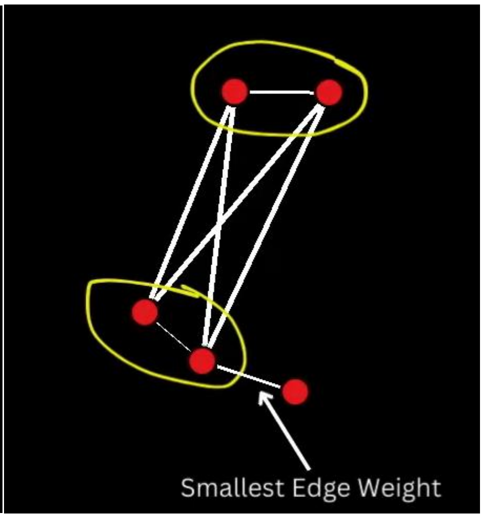
RCNN

Selective search is not a learned algorithm. It just segments images into regions and merges similar regions to create larger regions.

Step 1) Segment into different regions. Start assuming the image as a graph with pixels being the vertices and edges representing neighborhood of each pixel. Each edge is given a weight measured by some sort of distance. It can be color difference, texture difference etc. If the difference is greater, then larger weight. If the difference is lower, then smaller weight. At the start of segmentation, each vertex is a component by itself, then we pick the components with the smallest edge weights, if the edge weight is smaller than a predefined threshold, merge those two. Just keep doing this until the smallest edge weight is larger than the predefined threshold.





So, at the end, elements of one component are similar, elements in different components are dissimilar.

Step 2) Merge similar regions to create larger regions. Take into account different parameters like color similarity, texture similarity, size similarity and shape similarity. Can also do this on the same image, but using different color spaces like HSV, Lab and Grayscale

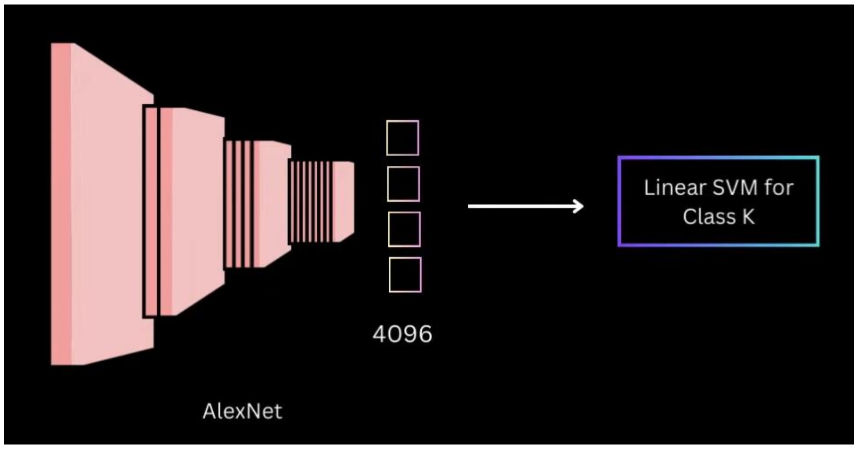
Now, the actual deep learning stuff. First extract feature representations for these proposals. Do by training a deep NN, AlexNet, on ImageNet, remove the last layer and add a new classification layer that caters to classes present in our dataset. Here, we add the background class manually. So we have K + 1 classes.

When warping the image down to 227x227, the authors first dilate the region so that they get p (=16) pixels of context. If the original box was at the boundary, they pad using mean pixels instead.

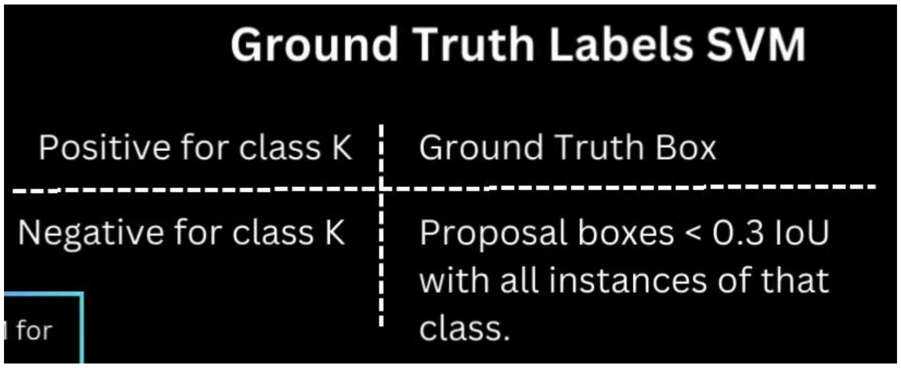
Next, to assign labels to the boxes generated by selective search, just simply select all boxes with an IOU > threshold with a ground truth box and label it with that ground truth label. Others become background proposals. During training, just randomly sample some positive and background proposals and this will be the batch.

After we have trained this new network (with the new classification layer). We remove that final classification layer and use the FC layer's output as the feature representations.

Next, you train a linear SVM for each class to classify feature representations as positive or negative.

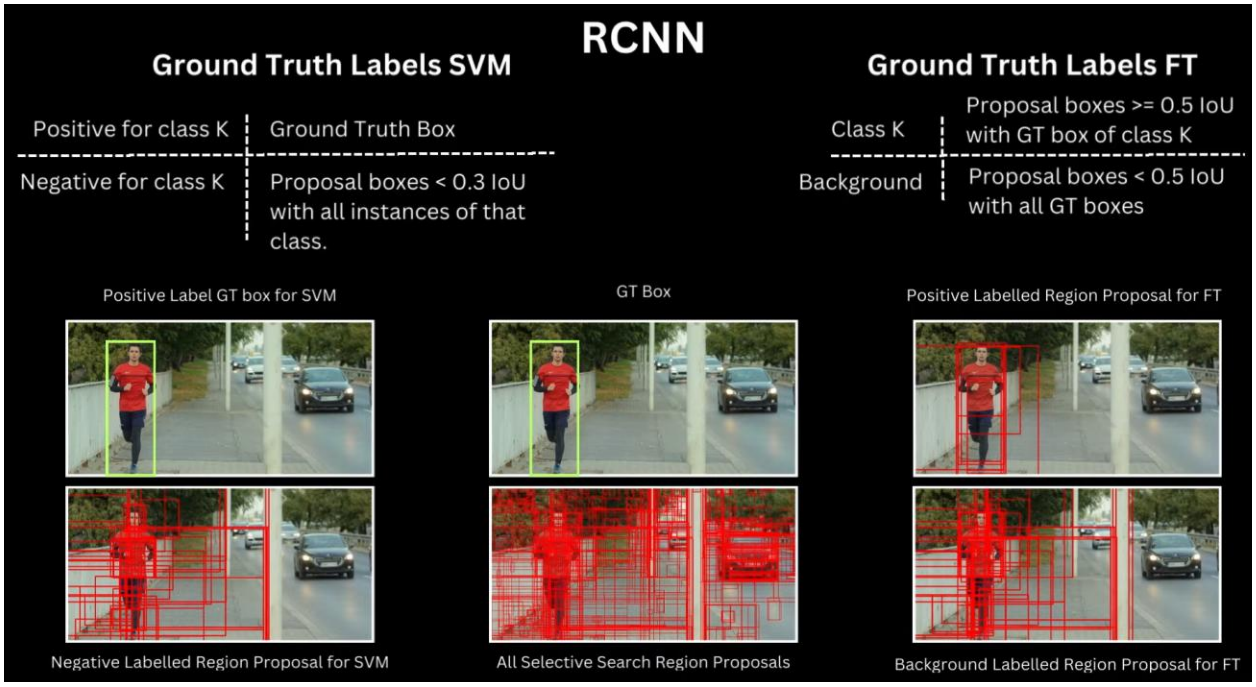


Here, the NN is frozen at this stage.



Ground Truth Labels SVM

Positive for class K	Ground Truth Box
Negative for class K	Proposal boxes < 0.3 IoU with all instances of that class.


RCNN

Ground Truth Labels SVM

Positive for class K	Ground Truth Box
Negative for class K	Proposal boxes < 0.3 IoU with all instances of that class.

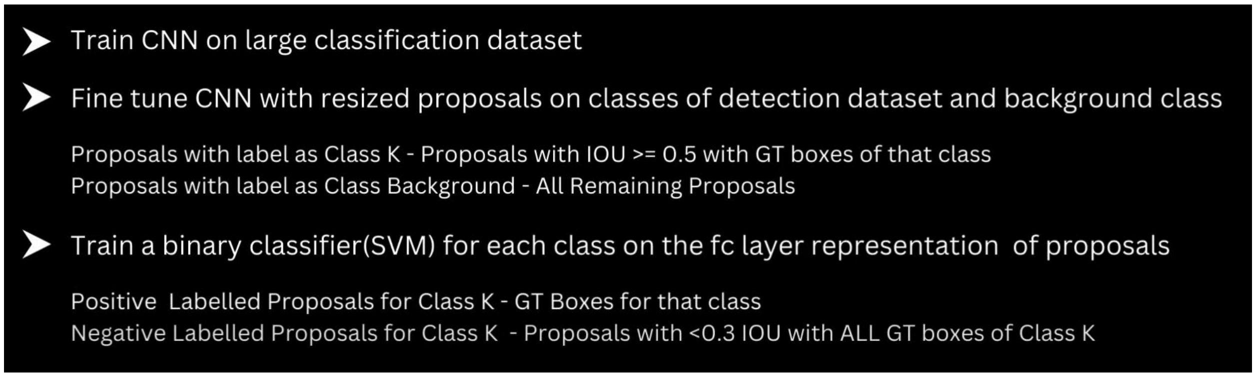
Ground Truth Labels FT

Class K	Proposal boxes >= 0.5 IoU with GT box of class K
Background	Proposal boxes < 0.5 IoU with all GT boxes

Positive Label GT box for SVM
GT Box
Positive Labelled Region Proposal for FT
Negative Labelled Region Proposal for SVM
All Selective Search Region Proposals
Background Labelled Region Proposal for FT

During fine tuning, they feel that this increase number of positive examples helps prevent overfitting at that stage.

Note that there is also a bbox regressor that morphs the selective search proposals.


- Train CNN on large classification dataset
- Fine tune CNN with resized proposals on classes of detection dataset and background class

  Proposals with label as Class K - Proposals with IOU >= 0.5 with GT boxes of that class
  Proposals with label as Class Background - All Remaining Proposals

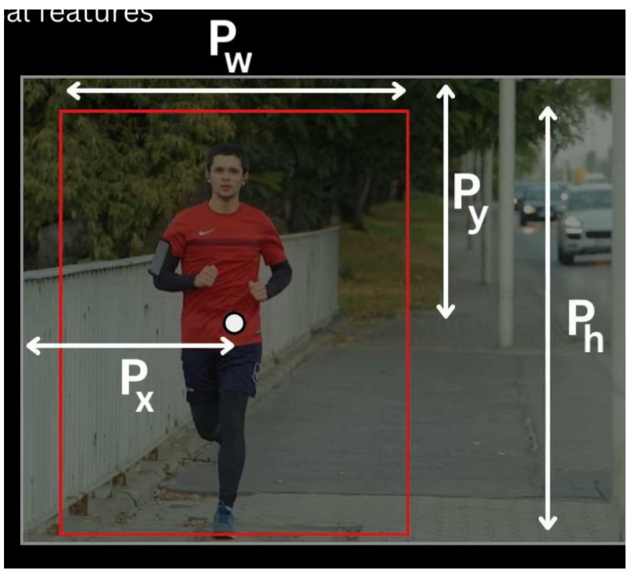
- Train a binary classifier(SVM) for each class on the fc layer representation of proposals

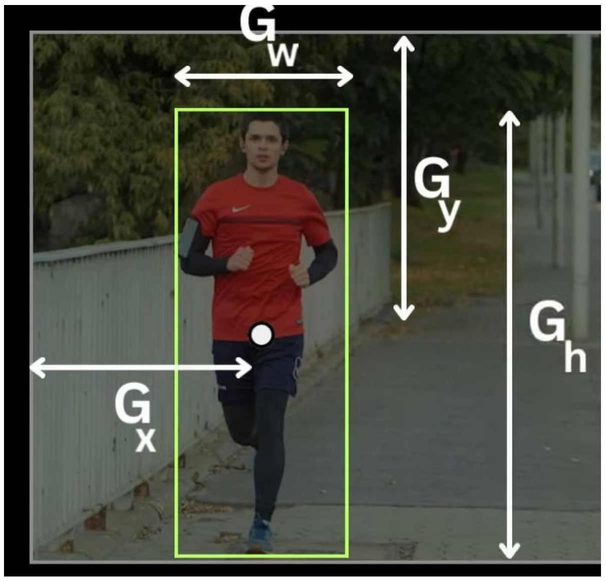
  Positive Labelled Proposals for Class K - GT Boxes for that class
  Negative Labelled Proposals for Class K - Proposals with <0.3 IOU with ALL GT boxes of Class K

At the very end, train a CLASS-SPECIFIC bounding box regressor. This is just a linear layer that pooling layer representation of boxes and modifies the original selective search proposal boxes to get better localized boxes for the objects of interest.

To do this, given a selective search box, we get the following information

The x,y coordinate of the center of the box and width and height of the selective search box.





We also do this for the ground truth box. Then parameterize the transformation from P to G in the following way (left image):

$$G_x = P_w * t_x + P_x$$

$$G_y = P_h * t_y + P_y$$

$$G_w = P_w * exp(t_w)$$

$$G_h = P_h * exp(t_h)$$

$$t_x = \frac{G_x - P_x}{P_w}$$

$$t_y = \frac{G_y - P_y}{P_h}$$

$$t_w = log\left(\frac{G_w}{P_w}\right)$$

$$t_h = log\left(\frac{G_h}{P_h}\right)$$

And the linear regressor must then essentially predict the tx, ty, tw, th expressed in the right

Note that these linear layers are CLASS-SPECIFIC, so you would have different set of linear layers for each class. Then during inference for some non-background prediction, use t_hat predicted for the class assigned to the ground truth to get the G_hat (the modified boxes).

For this, wee need to mention which ground truth is to be used for which proposals. To do this, we simply assign each selective search proposal to the ground truth that has the maximum overlap. But only if the IOU > 0.6 Otherwise that proposal box is not even used for training the regressor.

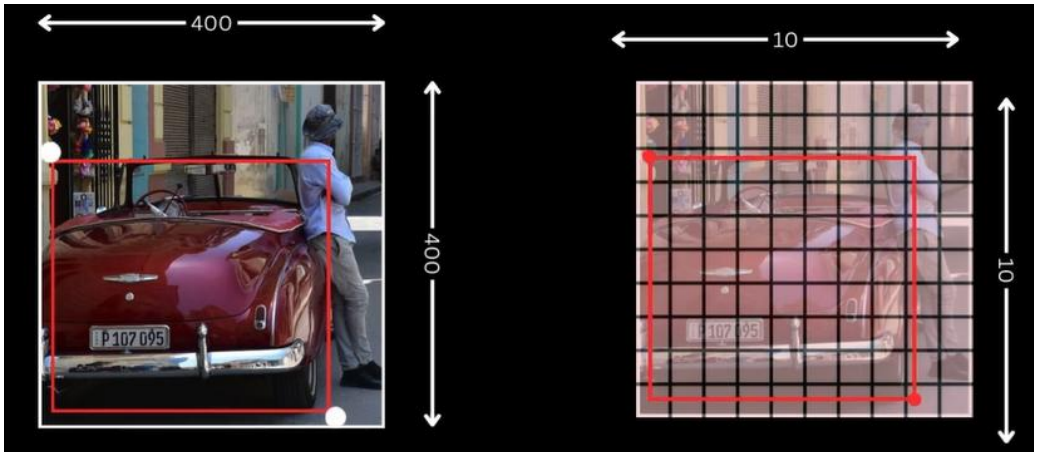
Fast-RCNN

Problems with RCNN

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 re- quire 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)

To address Problem 3:

In Fast RCNN, we forward the entire image in one go (without cropping and then forwarding). Suppose image was 400x400, then after the final convolution, suppose the feature map is 10x10.

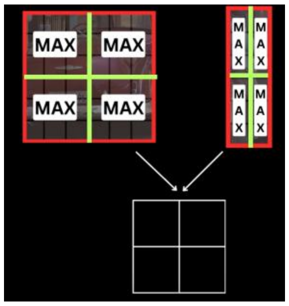


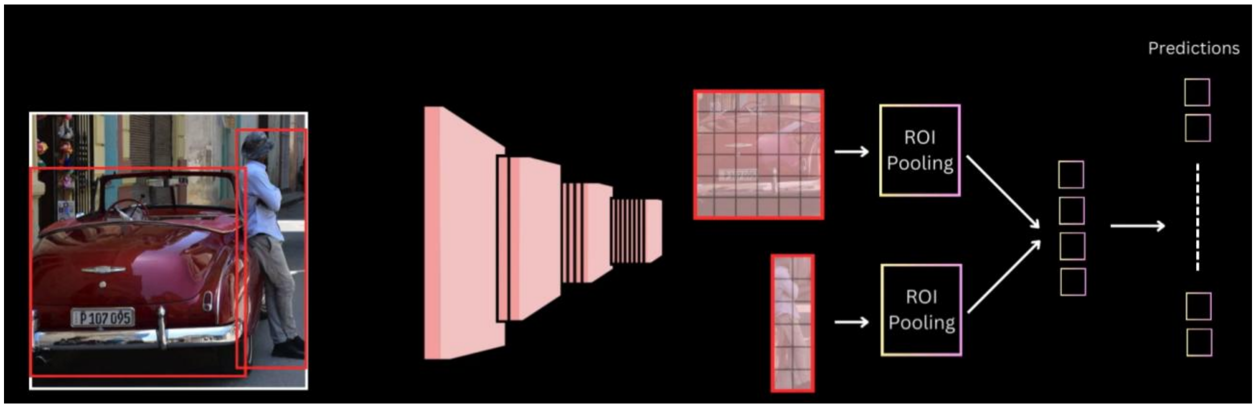
Then we just identify the original ground truth boxes in this convolved feature map. Essentially the original ground truth coordinates / 40. Need to quantize the location of proposal in the feature map by rounding. This would include extraneous regions and exclude parts of the original box.

Then for these extracted proposal maps, we get the proposal feature using the ROI pooler.

In Fast RCNN we introduce an ROI pooling layer that takes whatever feature proposal map and extract a uniform output size from the convolutional feature map.

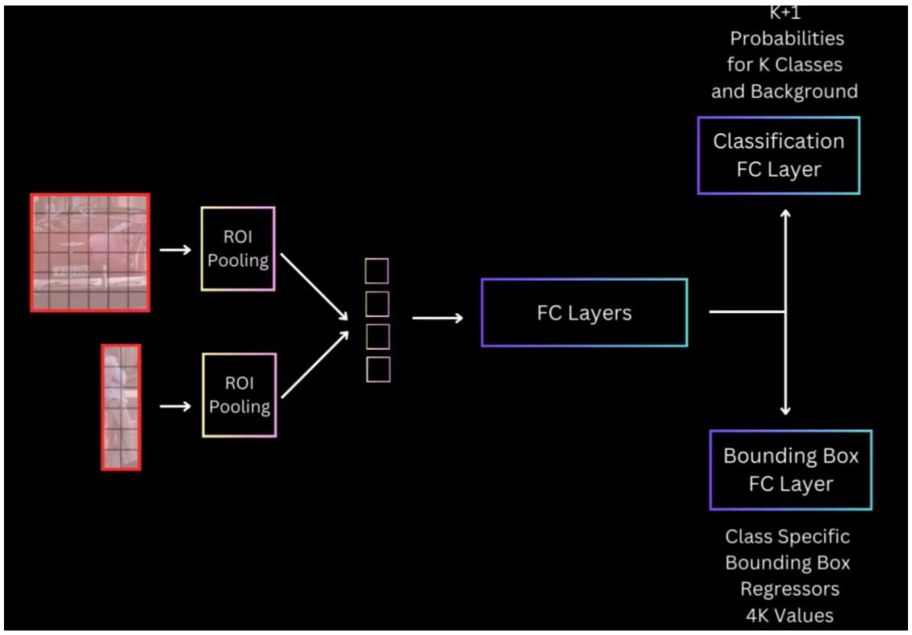
This is identical to the SPP net pooling.





To address Problem 1 (Multi stage training)

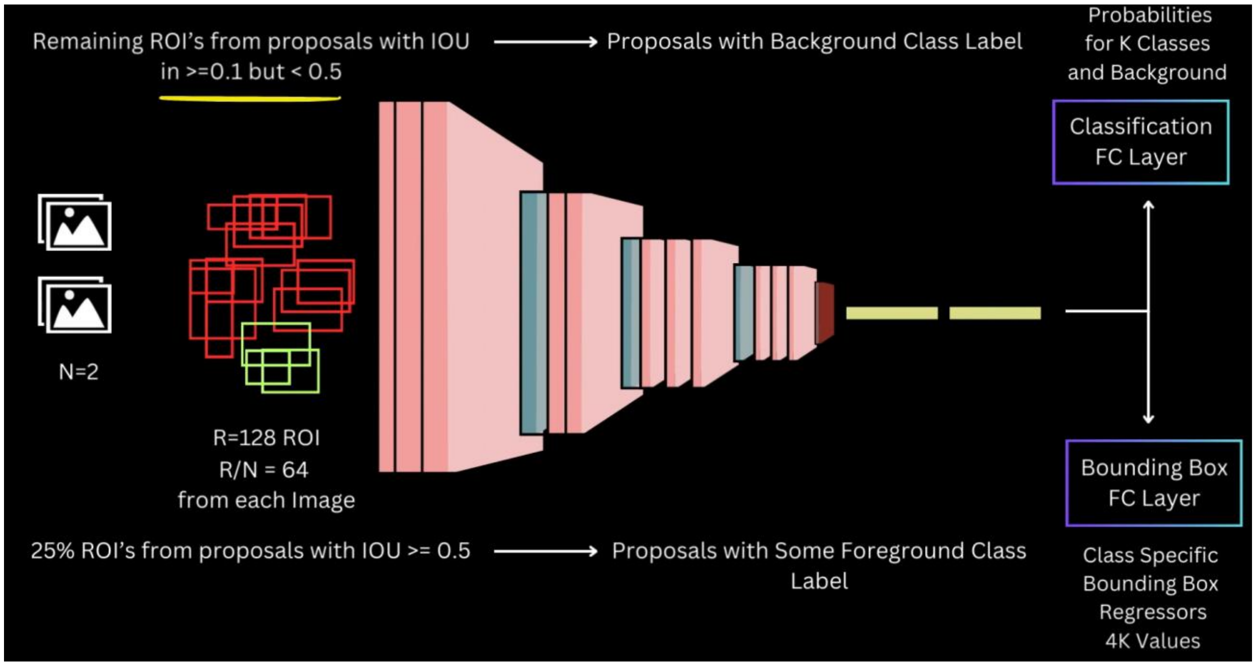
Then they take the flattened ROI pooling output, feed it to a couple of FC layers and have two branches of FC layers. One is a classification FC layer and the other is a bounding box regressor



Finetuning the above pipeline for detection

First we sample N images (N=2), from these N images, we sample R proposals by sampling R/N proposals from each image. During sampling ensure that >=25% ROI's from proposals with >= 0.5 (these are proposals with some foreground class label).

Remaining ROI's are sampled from proposals with IOU >= 0.1 but < 0.5 to any ground truth box (these are proposals with background class).



So, now we have single stage training but multi task loss

Loss is a combination of classification loss and localization loss.

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

The classification loss is the cross entropy loss using the K+1 output probabilities from the classification and layer and the ground truth class. So similar the apple, banana example.

Localization part is similar to what RCNN had, but instead of MSE, we use smooth L1 loss. The smooth L1 loss will be between the 4 transformation parameters predicted by the network and for THIS ground truth class u and the target transformation parameters.

$$\sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(\hat{t}_i^u - t_i)$$

The Identity function [u >= 1] ensures that loss only kicks in for the non-background proposals as the authors used the conventions that u = 0 means background class. Because only for the non-background proposals do we have a valid transformation parameter.