Weighted Loss Functions

You may be wondering: how can we train a network with two different outputs (a class and a bounding box) and different losses for those outputs?

We know that, in this case, we use categorical cross entropy to calculate the loss for our predicted and true classes, and we use a regression loss (something like smooth L1 loss) to compare predicted and true bounding boxes. But, we have to train our whole network using one loss, so how can we combine these?

There are a couple of ways to train on multiple loss functions, and in practice, we often use a weighted sum of classification *and* regression losses (ex. `0.5*cross_entropy_loss + 0.5*L1_loss`); the result is a single error value with which we can do backpropagation. This does introduce a hyperparameter: the loss weights. We want to weight each loss so that these losses are balanced and combined effectively, and in research we see that another regularization term is often introduced to help decide on the weight values that best combine these losses.

R-CNN Outputs

The R-CNN is the least sophisticated region-based architecture, but it is the basis for understanding how multiple object recognition algorithms work! It outputs a class score and bounding box coordinates for every input RoI.

An R-CNN feeds an image into a CNN with regions of interest (RoI's) already identified. Since these RoI's are of varying sizes, they often need to be **warped to be a standard size**, since CNN's typically expect a consistent, square image size as input. After RoI's are warped, the R-CNN architecture, processes these regions one by one and, for each image, produces 1. a class label and 2. a bounding box (that may act as a slight correction to the input region).

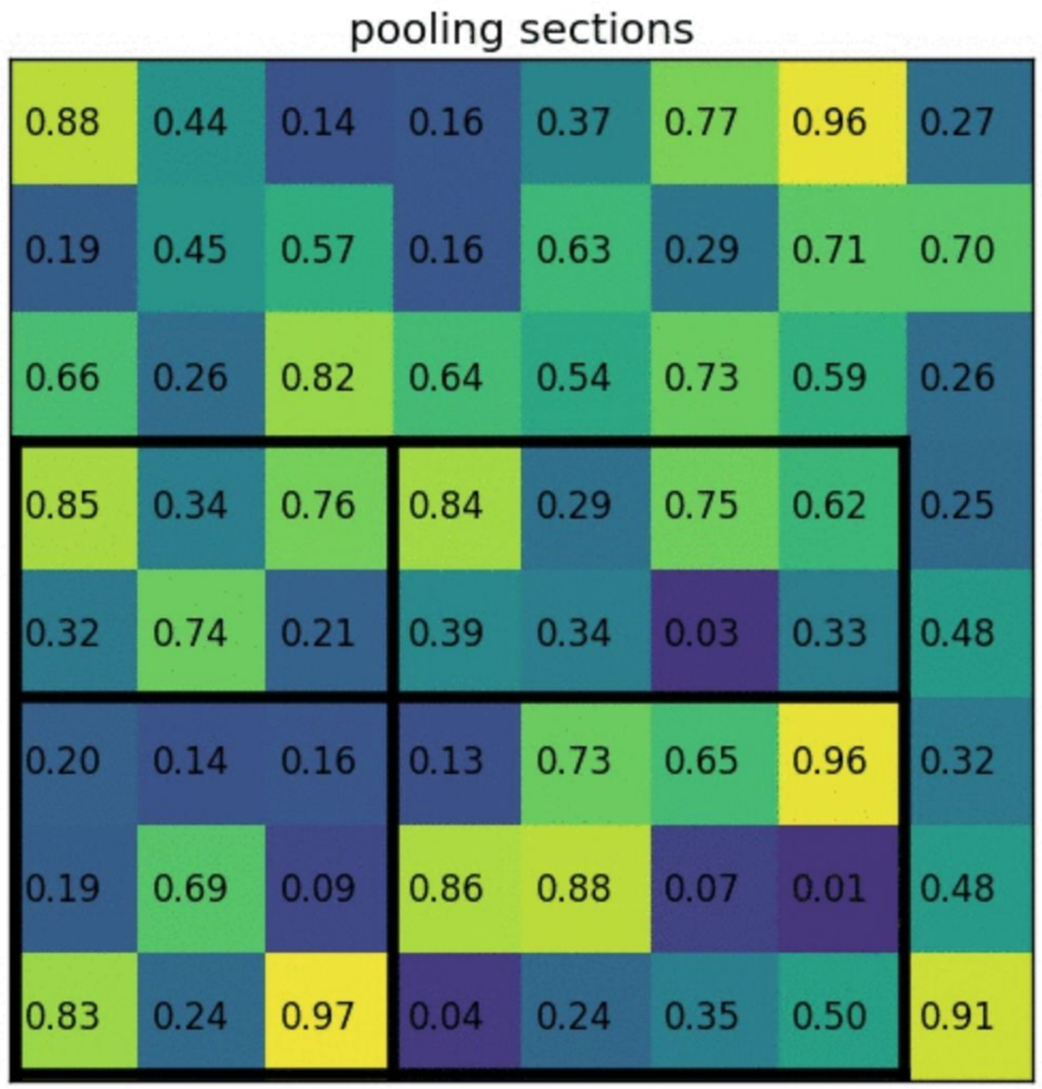
1. R-CNN produces bounding box coordinates to reduce localization errors; so a region comes in, but it may not perfectly surround a given object and the output coordinates `(x,y,w,h)` aim to *perfectly* localize an object in a given region.
2. R-CNN, unlike other models, does not explicitly produce a confidence score that indicates whether an object is in a region, instead it cleverly produces a set of class scores for which one class is "background". This ends up serving a similar purpose, for example, if the class score for a region is `Pbackground = 0.10`, it likely contains an object, but if it's `Pbackground = 0.90`, then the region probably doesn't contain an object.

RoI Pooling

To warp regions of interest into a consistent size for further analysis, some networks use RoI pooling. RoI pooling is an additional layer in our network that takes in a rectangular region of any size, performs a maxpooling operation on that region in pieces such that the output is a fixed shape. Below is an example of a region with some pixel values being broken up into pieces which pooling will be applied to; a section with the values:

```
[[0.85, 0.34, 0.76],
 [0.32, 0.74, 0.21]]
```

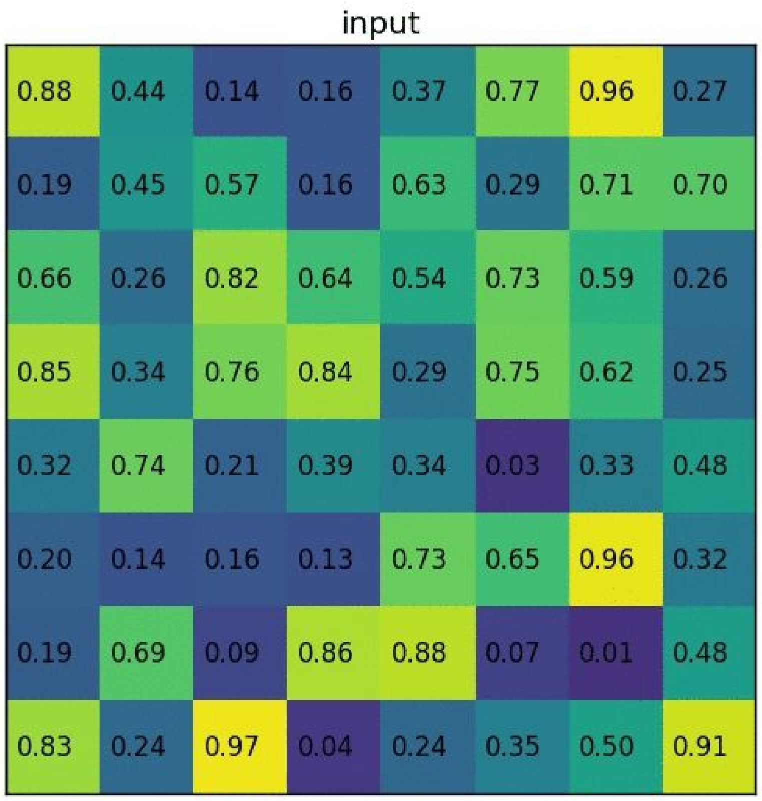
Will become a single max value after pooling: `0.85`. After applying this to an image in these pieces, you can see how any rectangular region can be forced into a smaller, square representation.



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

An example of pooling sections, credit to this informational resource on RoI pooling [by Tomasz Grel].

You can see the complete process from input image to region to reduced, maxpooled region, below.



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

Credit to this informational resource on RoI pooling.

Speed

Fast R-CNN is about 10 times as fast to train as an R-CNN because it only creates convolutional layers once for a given image and then performs further analysis on the layer. Fast R-CNN also takes a shorter time to test on a new image! It's test time is dominated by the time it takes to create region proposals.

Region Proposal Network

You may be wondering: how exactly are the RoI's generated in the region proposal portion of the Faster R-CNN architecture?

The region proposal network (RPN) works in Faster R-CNN in a way that is similar to YOLO object detection, which you'll learn about in the next lesson. The RPN looks at the output of the last convolutional layer, a produced feature map, and takes a sliding window approach to possible-object detection. It slides a small (typically 3x3) window over the feature map, then for *each* window the RPN:

1. Uses a set of defined anchor boxes, which are boxes of a defined aspect ratio (wide and short or tall and thin, for example) to generate multiple possible RoI's, each of these is considered a region proposal.
2. For each proposal, this network produces a probability, `Pc`, that classifies the region as an object (or not) and a set of bounding box coordinates for that object.
3. Regions with too low a probability of being an object, say `Pc < 0.5`, are discarded.

Training the Region Proposal Network

Since, in this case, there are no ground truth regions, how do you train the region proposal network?

The idea is, for any region, you can check to see if it overlaps with any of the ground truth objects. That is, for a region, if we classify that region as an object or not-object, which class will it fall into? For a region proposal that does cover some portion of an object, we should say that there is a high probability that this region has an object init and that region should be kept; if the likelihood of an object being in a region is too low, that region should be discarded.

I'd recommend this blog post if you'd like to learn more about region selection.

Speed Bottleneck

Now, for all of these networks *including* Faster R-CNN, we've aimed to improve the speed of our object detection models by reducing the time it takes to generate and decide on region proposals. You might be wondering: is there a way to get rid of this proposal step entirely? And in the next section we'll see a method that does not rely on region proposals to work!

Faster R-CNN Implementation

If you'd like to look at an implementation of this network in code, you can find a peer-reviewed version, at this Github repo.