UDC 004.93'12

OBJECT DETECTION USING FASTER R-CNN, YOLO AND SSD

ILLIA IVANOU, KATSIARYNA HATSIKHA
Polotsk State University, Belarus

The paper describes and compares one of the most popular and effective lightweight object detection algorithms: faster R-CNN, R-FCN, YOLO and SSD. The article presents the intuitive approach and salient features of each method. The article gives an understanding of how deep learning is applied to object detection, and how these object detection models both inspire and diverge from one another.

Faster R-CNN is now a canonical model for deep learning-based object detection. It helped inspire many detection and segmentation models that came after it, including the two others we are going to examine in the article. Unfortunately, it is not possible to begin to understand Faster R-CNN without examining its predecessors, R-CNN and Fast R-CNN [1].

R-CNN, or Region-based Convolutional Neural Network, consisted of 3 simple steps (Fig. 1):

1. Scanning the input image for possible objects using an algorithm called Selective Search, generating ~2000 region proposals
2. Running a convolutional neural net (CNN) on top of each of these region proposals
3. Taking the output of each CNN and feed it into a) an SVM to classify the region and b) a linear regressor to tighten the bounding box of the object, if such an object exists.

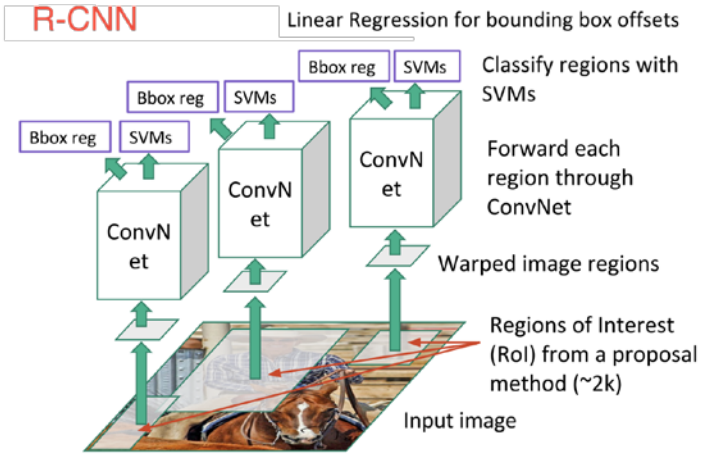

Fig. 1. Region-Based Convolutional Neural Network

In other words, we first propose regions, then extract their features, and then classify the regions based on their features. In essence, we have turned object detection into an image classification problem. R-CNN was very intuitive, but very slow.

R-CNN's immediate descendant was **Fast-R-CNN**. Fast R-CNN resembled the original in many ways, but improved on its detection speed through two main augmentations (Fig. 2):

1. Performing feature extraction over the image before proposing regions, thus only running one CNN over the entire image instead of 2000 CNN's over 2000 overlapping regions.
2. Replacing the SVM with a softmax layer, thus extending the neural network for predictions instead of creating a new model.

As we can see from the image, we are now generating region proposals based on the last feature map of the network, not from the original image itself. As a result, we can train just one CNN for the entire image.

In addition, instead of training many different SVM's to classify each object class, there is a single softmax layer that outputs the class probabilities directly. Now we only have one neural net to train, as opposed to one neural net and many SVM's.

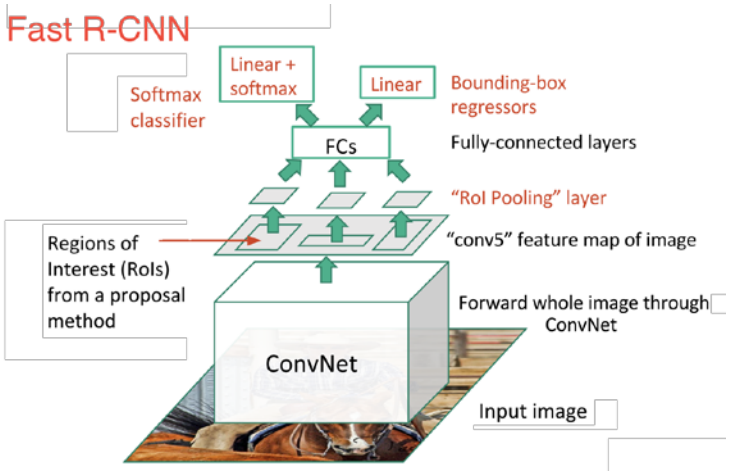

Fig. 2. Fast Region-Based Convolutional Neural Network

Fast R-CNN performed much better in terms of speed. There was just one big bottleneck remaining: the selective search algorithm for generating region proposals [1].

At this point, we are back to our original target: **Faster R-CNN**. The main insight of Faster R-CNN was to replace the slow selective search algorithm with a fast neural net. Specifically, it introduced the region proposal network (RPN) (Fig. 3).

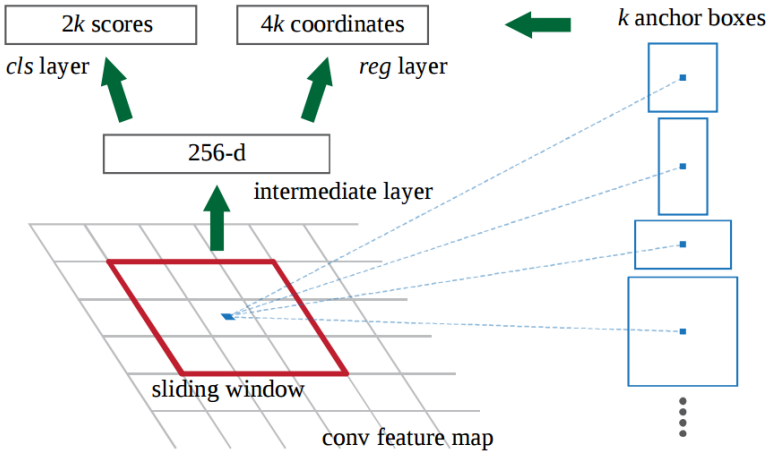

Fig. 3. Faster Region-Based Convolutional Neural Network

At the last layer of an initial CNN, a 3x3 sliding window moves across the feature map and maps it to a lower dimension (e.g. 256-d).

For each sliding-window location, it generates multiple possible regions based on k fixed-ratio anchor boxes (default bounding boxes).

Each region proposal consists of

1. An "objectness" score for that region.
2. Four coordinates representing the bounding box of the region.

In other words, we look at each location in our last feature map and consider k different boxes centered around it: a tall box, a wide box, a large box, etc. For each of these boxes, we output whether or not we think it contains an object, and what the coordinates for this box are. This is what it looks like at one sliding window location:

The 2k scores represent the softmax probability of each of the k bounding boxes being on 'object'. It should be taken into account that although the RPN outputs bounding box coordinates, it does not try to classify

any potential objects: its sole task is still proposing object regions. If an anchor box has an 'objectness' score above a certain threshold, the box's coordinates get passed forward as a region proposal.

Once we have our region proposals, we feed them straight into what is essentially a Fast R-CNN. We add a pooling layer, some fully-connected layers, and finally a softmax classification layer and bounding box regressor. In a sense, Faster R-CNN = RPN + Fast R-CNN (Fig. 4).

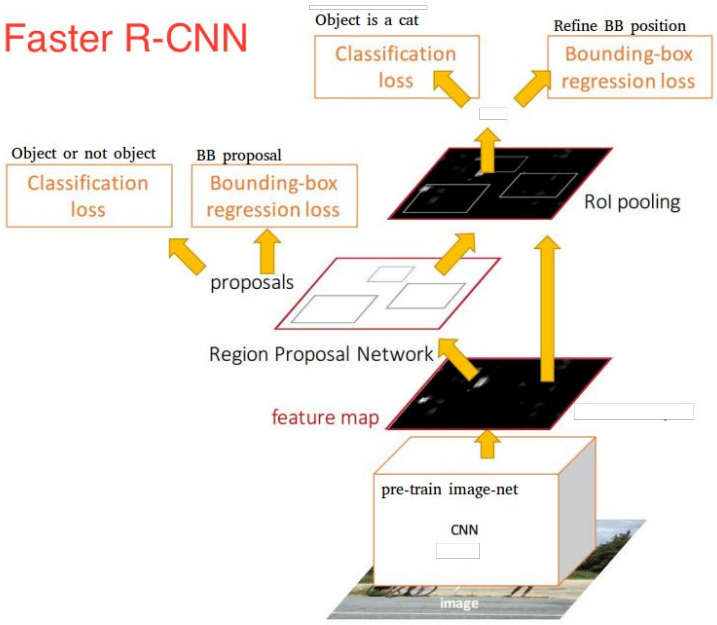

Fig. 4. Faster Region-Based Convolutional Neural Network as the Combination of RPN + Fast R-CNN

Altogether, Faster R-CNN has much better speeds and a state-of-the-art accuracy. It is worth noting that although future models did a lot to increase detection speeds, few models managed to outperform Faster R-CNN by a significant margin. In other words, Faster R-CNN may not be the simplest or fastest method for object detection, but it is still one of the best performing. A case in point, Tensorflow's Faster R-CNN with Inception ResNet is their slowest but most accurate model.

At the end of the day, Faster R-CNN may look complicated, but its core design is the same as the original R-CNN: to hypothesize object regions and then classify them. This is now the predominant pipeline for many object detection models, including our next one [1].

R-FCN, or Region-Based Fully Convolutional Net, shares 100% of the computations across every single output. Being fully convolutional, it ran into a unique problem in model design.

On the one hand, when performing classification of an object, we want to learn location invariance in a model: regardless of where a cat appears in the image, we want to classify it as a cat. On the other hand, when performing detection of the object, we want to learn location variance: if the cat is in the top left-hand corner, we want to draw a box in the top left-hand corner.

R-FCN's solution is position-sensitive score maps.

Each position-sensitive score map represents one relative position of one object class. For example, one score map might activate wherever it detects the top-right of a cat. Another score map might activate where it sees the bottom-left of a car. You get the point. Essentially, these score maps are convolutional feature maps that have been trained to recognize certain parts of each object [2]:

1.Run a CNN (in this case, ResNet) over the input image;

2.Add a fully convolutional layer to generate a score bank of the aforementioned 'position-sensitive score maps'. There should be $k^2(C+1)$ score maps, with k^2 representing the number of relative positions to divide an object (e.g. 3^2 for a 3 by 3 grid) and C+1 representing the number of classes plus the background;

3.Run a fully convolutional region proposal network (RPN) to generate regions of interest (RoI's);

4.For each RoI, divide it into the same k^2 "bins" or subregions as the score maps;

5.For each bin, check the score bank to see if that bin matches the corresponding position of some object. For example, if I am on the 'upper-left' bin, I will grab the score maps that correspond to the 'upper-left' corner of an object and average those values in the RoI region. This process is repeated for each class;

6.Once each of the k^2 bins has an "object match" value for each class, average the bins to get a single score per class;

7.Classify the RoI with a softmax over the remaining C+1 dimensional vector.

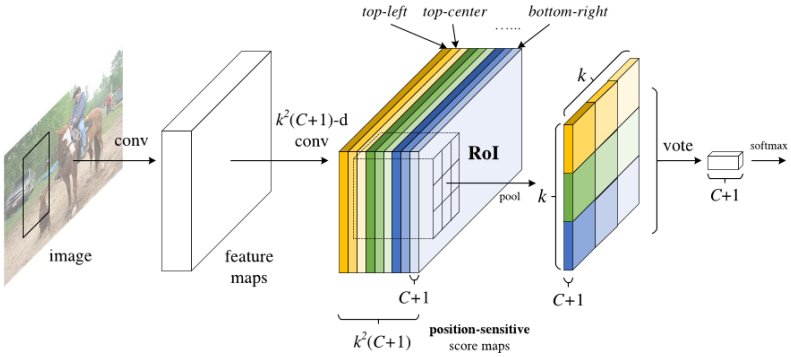

Fig. 5. Region-Based Fully Convolutional Net

Simply put, R-FCN considers each region proposal, divides it up into sub-regions, and iterates over the sub-regions asking: 'Does this look like the top-left of a baby?', 'Does this look like the top-center of a baby?' 'Does this look like the top-right of a baby?', etc. It repeats this for all possible classes. If enough of the sub-regions say 'yes, I match up with that part of a baby!', the RoI gets classified as a baby after a softmax over all the classes.

With this setup, R-FCN is able to simultaneously address location variance by proposing different object regions, and location invariance by having each region proposal refer back to the same bank of score maps. These score maps should learn to classify a cat as a cat, regardless of where the cat appears. Best of all, it is fully convolutional, meaning all of the computation is shared throughout the network [2].

As a result, R-FCN is several times faster than Faster R-CNN and achieves comparable accuracy.

SSD, which stands for Single-Shot Detector. Like R-FCN, it provides enormous speed gains over Faster R-CNN, but it does so in a markedly different manner.

Our first two models performed region proposals and region classifications in two separate steps. First, they used a region proposal network to generate regions of interest; then, they used either fully-connected layers or position-sensitive convolutional layers to classify those regions. SSD does the two in a "single shot," simultaneously predicting the bounding box and the class as it processes the image [3].

Concretely, given an input image and a set of ground truth labels, SSD does the following:

It passes the image through a series of convolutional layers, yielding several sets of feature maps at different scales (e.g. 10x10, then 6x6, then 3x3, etc.).

1.For each location in each of these feature maps, a 3x3 convolutional filter to evaluate a small set of default bounding boxes is used. These default bounding boxes are essentially equivalent to Faster R-CNN's anchor boxes.

2.For each box, simultaneously predict a) the bounding box offset and b) the class probabilities.

During training, the ground truth box is matched with the predicted boxes based on IoU. The best predicted box will be labeled 'positive', along with all the other boxes that have an IoU with the ground truth >0.5.

SSD sounds straightforward, but training it has a new challenge. With the previous two models, the region proposal network ensured that everything we tried to classify had some minimum probability of being an "object." With SSD, however, we skip that filtering step. We classify and draw bounding boxes from every single position in the image, using multiple different shapes, at several different scales. As a result, we generate a much greater number of bounding boxes than the other models, and nearly all of the them are negative examples [3].

To fix this imbalance, SSD does two things. Firstly, it uses non-maximum suppression to group together highly-overlapping boxes into a single box. In other words, if four boxes of similar shapes, sizes, etc. contain the same dog, NMS would keep the one with the highest confidence and discard the rest. Secondly, the model uses a technique called hard negative mining to balance classes during training. In hard negative mining, only a subset

of the negative examples with the highest training loss (i.e. false positives) are used at each iteration of training. SSD keeps a 3:1 ratio of negatives to positives (Fig. 6).

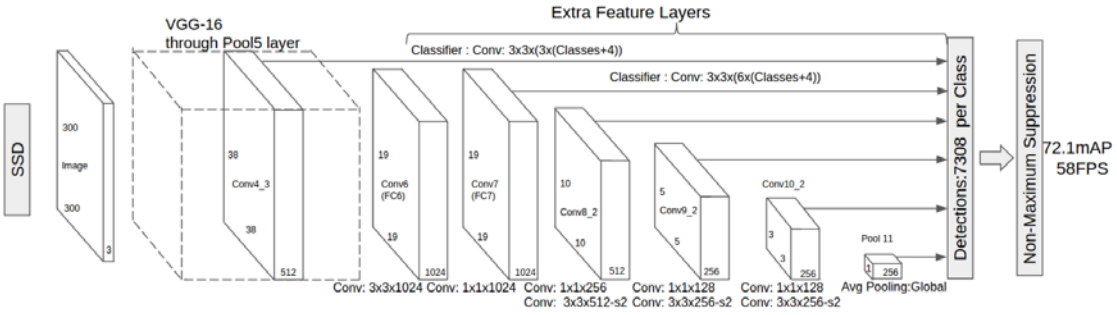

Fig. 6. Region-Based Fully Convolutional Net

As it was mentioned above there are 'extra feature layers' at the end that scale down in size. These varying-size feature maps help capture objects of different sizes (Fig. 7).

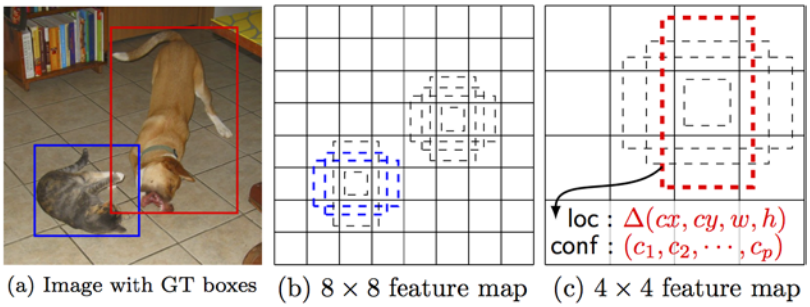

Fig. 7. SSD in Action

In smaller feature maps (e.g. 4x4), each cell covers a larger region of the image, enabling them to detect larger objects. Region proposal and classification are performed simultaneously: given p object classes, each bounding box is associated with a (4+p)-dimensional vector that outputs 4 box offset coordinates and p class probabilities. In the last step, softmax is again used to classify the object.

Ultimately, SSD is not so different from the first two models. It simply skips the 'region proposal' step, instead of considering every single bounding box in every location of the image simultaneously with its classification. Because SSD does everything in one shot, it is the fastest of the three models, and still performs quite comparably [3].

YOLO divides each image into a grid of S x S and each grid predicts N bounding boxes and confidence. The confidence reflects the accuracy of the bounding box and whether the bounding box actually contains an object (regardless of class). YOLO also predicts the classification score for each box for every class in training. It is possible to combine both the classes to calculate the probability of each class being present in a predicted box.

So, total SxSxN boxes are predicted. However, most of these boxes have low confidence scores and if we set a threshold say 30% confidence, we can remove most of them as shown in the example below (Fig. 8) [4].

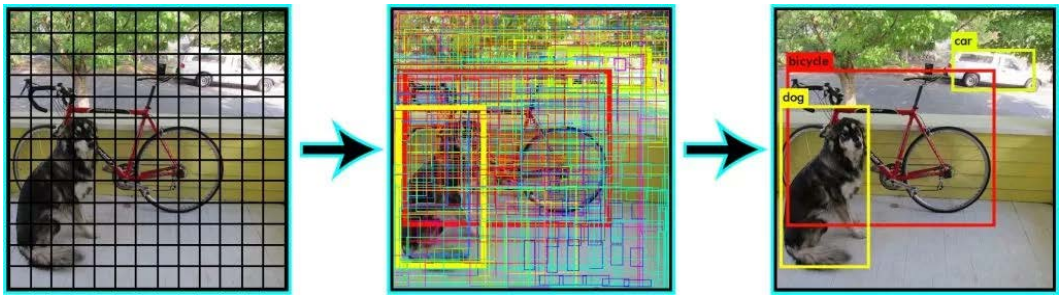

Fig. 8. YOLO in action

Notice that at runtime, we have run our image on CNN only once. Hence, YOLO is super fast and can be run real time. Another key difference is that YOLO sees the complete image at once as opposed to looking only at a generated region proposals in the previous methods. So, this contextual information helps in avoiding false positives. However, one limitation for YOLO is that it only predicts 1 type of class in one grid hence, it struggles with very small objects.

The choice of the right object detection method is crucial and depends on the problem to solve and the set-up. Object Detection is the backbone of many practical applications of computer vision such as autonomous cars, security and surveillance, and many industrial applications.

Faster R-CNN, R-FCN, YOLO, and SSD are four of the best and most widely used object detection models out there right now. Other popular models tend to be fairly similar to these three ones, all relying on deep CNN's to do the initial heavy lifting and largely following the same proposal/classification pipeline.

Currently, Faster-RCNN is a good choice for accuracy numbers. However, if you are strapped for computation (probably running it on Nvidia Jetsons), SSD is a better recommendation. Finally, if accuracy is not too much of a concern but you want to go super fast, YOLO will be the way to go.

SSD seems to be a good choice as we are able to run it on a video and the accuracy trade-off is very little. However, it may not be that simple, as it is shown in the chart that compares the performance of SSD, YOLO, and Faster-RCNN on various sized objects. At large sizes, SSD seems to perform similarly to Faster-RCNN. However, you should look at the accuracy numbers, when the object size is small the gap widens.

REFERENCES

1. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks / Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2016. – P. 1137–1149
2. R-FCN: Object Detection via Region-based Fully Convolutional Networks / Jifeng Dai, Yi Li, Kaiming He, Jian Sun // NIPS. – 2016. – P. 150–156.
3. SSD: Single Shot MultiBox Detector / Wei Liu, Dragomir Anguelov, Dumitru Erhan // Proceedings of the European Conference on Computer Vision. – 2016.
4. You Only Look Once: Unified, Real-Time Object Detection / Joseph Redmon, Santosh Divvala, Ross Girshick // Computer Vision and Pattern Recognition, IEEE Conference. – 2016.