CIS 680: Advanced Machine Perception
Assignment 3: Instance Segmentation with MaskRCNN
Due: October 31, 2019 at 11:59

1 Overview

In this exercise you will implement MaskRCNN [1], an algorithm that addresses the task of instance segmentation, which combines object detection and semantic segmentation into a per-pixel object detection framework.

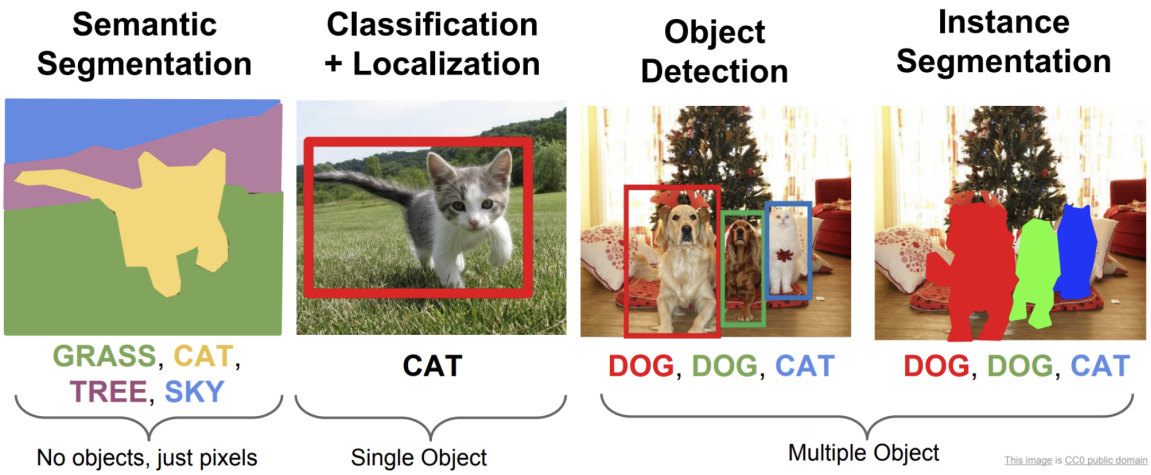

Figure 1: Instance Segmentation

The full implementation of [1] would take many days to train, so you will implement a simpler version that keeps all the necessary components. However, these simplifications affect the performance of the algorithm. Later, we will provide you with pretrained parts to boost the performance.

The exercise consists of three parts:

1. Implement a Region Proposal Network (RPN) [2].
2. Implement a RoI Align [1] which extends RoI pooling [2].
3. Complete the MaskRCNN architecture by adding the object detection heads of FasterRCNN and a parallel mask branch [1, 2].

You are strongly encouraged to read the papers.

2 Dataset Description

In this assignment your task is to detect 3 types of objects: Vehicles, People and Animals.Your dataset consists of:
(1) a numpy array of all the RGB Images ($3 \times 300 \times 400$)
(2) a numpy array of all the masks (300×400)
(3) list of ground truth labels per image.

(4) list of ground truth bounding boxes per image. The four numbers are the upper left and lower right coordinates.
You should use the labels list to figure out which mask belongs to which image.

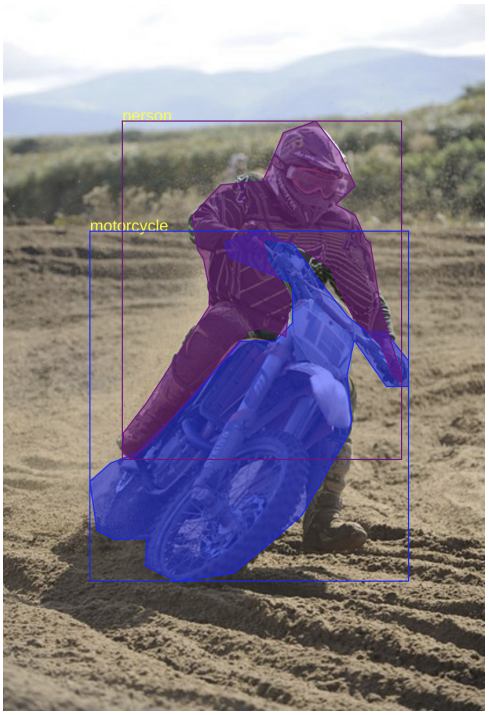

Figure 2: Datapoint

3 Region Proposal Network

Region Proposal Networks are "attention mechanisms" for the object detection task, performing a crude but inexpensive first estimation of where the bounding boxes of the objects should be. They were first proposed in [2] as a way to address the issue of expensive greedy algorithms like Selective Search ([3]), opening new avenues to end-to-end object detection tasks. They work through classifying the initial anchor boxes into object/background and refine the coordinates for the boxes with objects. Later, these boxes will be further refined and tightened by the instance segmentation heads as well as classified in their corresponding classes. The architecture is shown below:

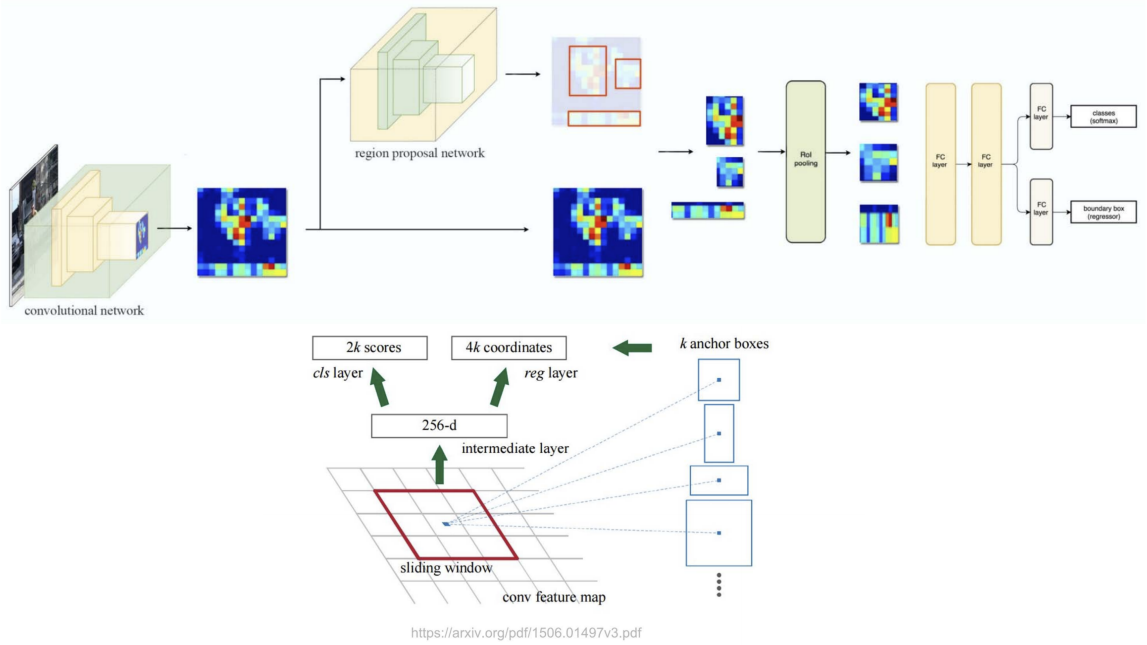

Figure 3: FasterRCNN architecture and detailed RPN

As you can see the convolutional network in the beginning serves as a shared backbone. Its output is the input to the RPN, which in turn, outputs the objectness and bounding boxes for each anchor box, for each feature of the input feature map. You learned in class that the anchor boxes are reference boxes, relative to which, we parametrize the outputs of the RPN. We will use just 1 anchor box to simplify the assignment. Feel free to use more for performance boost.

1. Build the following backbone architecture:

Layers	**Hyperparameters**
Convolution 1	Kernel Size = (5,5,16), BatchNorm, ReLU, Padding=Same
Pooling 1	Kernel Size = (2,2), Stride = 2, Padding=0
Convolution 2	Kernel Size = (5,5,32), BatchNorm, ReLU, Padding=Same
Pooling 1	Kernel Size = (2,2), Stride = 2, Padding=0
Convolution 3	Kernel Size = (5,5,64), BatchNorm, ReLU, Padding=Same
Pooling 1	Kernel Size = (2,2), Stride = 2, Padding=0
Convolution 4	Kernel Size = (5,5,128), BatchNorm, ReLU, Padding=Same
Pooling 1	Kernel Size = (2,2), Stride = 2, Padding=0
Convolution 5	Kernel Size = (5,5,256), BatchNorm, ReLU, Padding=Same

2. Make a histogram of the scales and aspect ratios of the bounding boxes of the dataset provided. Your anchor boxes will be defined by the modes of your histograms. What kind of observations can you make from these histograms that you expect will affect (positively or negatively) the performance?

3. Create ground truth for RPN training. We will assign a binary label (object or not) to each anchor. Firstly, cross-boundrary anchors are removed. Out of the remaining, positive labels will get the anchors that have either: (1) highest IOU with a ground truth box, or (2) anchors with $IOU > 0.7$ with ANY ground truth box. Note that a single ground truth box may assign positive labels to multiple anchors. Negative labels will get the anchors that (1) are non-positive and (2) have $IOU < 0.3$ with EVERY ground truth box. Anchors with neither positive or negative label are excluded from training both for the classification and the regression branch.

4. Use the feature map from the last layer of the base network to build a proposal classifier. This classifier outputs an objectness score (in the figure there are 2 outputs binded by a softmax layer. We

can alternatively use a single sigmoid). Specifically, add a standard convolution layer (referred as the intermediate layer) with kernel size (3, 3, 256) with same padding (followed by BatchNorm and ReLU) and a convolution layer with kernel size (1, 1, 1) with a sigmoid rectification layer. Thus, RPN outputs one objectness score per each feature in the feature map. Use the point-wise binary cross entropy loss to train the proposal classifier. Plot the training loss over training iterations.

5. What is the receptive field of each neuron in the intermediate layer? Is it enough for the network to locate the objects? Make an observation about how this receptive field together with the choice of the anchor boxes affect the performance. Consider cases like very big objects.

6. Build a proposal regressor. On top of the intermediate layer, add another convolution layer with kernel size (1, 1, 4) (without BatchNorm and rectification). The first two channels are for the row/column coordinates of the object center and the third and fourth channel for the width and height of the object.

Parameterize the coordinates as follows:

$$t_x = (x - x_a)/w_a \quad t_y = (y - y_a)/h_a \quad t_w = \log(w/w_a) \quad t_h = \log(h/h_a)$$

$$t_x^* = (x^* - x_a)/w_a \quad t_y^* = (y^* - y_a)/h_a \quad t_w^* = \log(w^*/w_a) \quad t_h^* = \log(h^*/h_a)$$

where x,y,w,h denote the box's center coordinates and its width and height. Variables x, x_a, x^* are for the predicted box, anchor box and groundtruth box respectively (likewise for y,w,h).

The regressor is trained with the Smooth L1 loss between t, t^*, which is defined as:

$$L_{reg}(t_i) = \frac{1}{N_{reg}} \sum_{i=1}^{N_{reg}} p_i^* smooth_{L1}(t_i - t_i^*)$$

where,

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & |x| \leq 1 \\ |x| - 0.5 & else \end{cases}$$

and N_{reg} is the number of anchor locations. Note that the regressor loss is computed only on positive anchors ($p_i^* = 1$). Plot the training regression loss over training iterations.

Remark: In [1], the authors propose an "image-centric" sampling strategy, where each mini-batch arises from a single image that contains many positive and negative example anchors. In order, to minimize the bias towards the negative classes they subsample the positive and negative anchors in a ratio as close to 1:1 as possible.

7. Train the whole network with *equally* weighted losses from the proposal classifier and the regressor.

Mind that the losses in Pytorch are usually normalized (reduction='mean'), but maybe they are normalized differently between classification and regression (eg. over batch versus over anchor boxes) in your implementation. If this is the case find the multiplication factor in one of them, so that the losses are normalized equally. Plot the training curves of each loss and the total loss. Report the (point-wise) test accuracy of the proposal classifier and regressor. Also from a small subset of your test images, plot the positive and some negative region proposals overlayed on the image with different colours.

8. After decoding the boxes, throw away the boundary crossing ones (only for training, clip for testing). Then, apply Non negative maximum suppression with IOU=0.7, like you did in the previous assignment and finally choose the top N proposals. Leave N as a hyperparameter for now. These will be used for training the rest of the network.

References

[1] He, K., Gkioxari, G., Dollar, P., Girshick, R.: Mask R-CNN. In: ICCV. (2017)

[2] Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. (2015)

[3] Girshick, R.: Fast R-CNN. In: ICCV. (2015)