CIS 680: Advanced Machine Perception
Project 4: FasterRCNN (Part A)
Due: November 2, 2020 at 11:59pm

1 Instructions

- This is a group assignment.
- The provided code template is not definitive, feel free to modify the details in your implementation.
- One option for the final project will be extending the simplified FasterRCNN implementation presented in this project to a full MaskRCNN implemention. More details on this option will come later.
- There is no single answer to most problems in deep learning, therefore the questions will often be underspecified. You need to fill in the blanks and submit a solution that solves the (practical) problem. Document the choices (hyperparameters, features, neural network architectures, etc.) you made in the write-up.
- All the code should be written in Python. You should use PyTorch only to complete this homework.

2 Overview

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

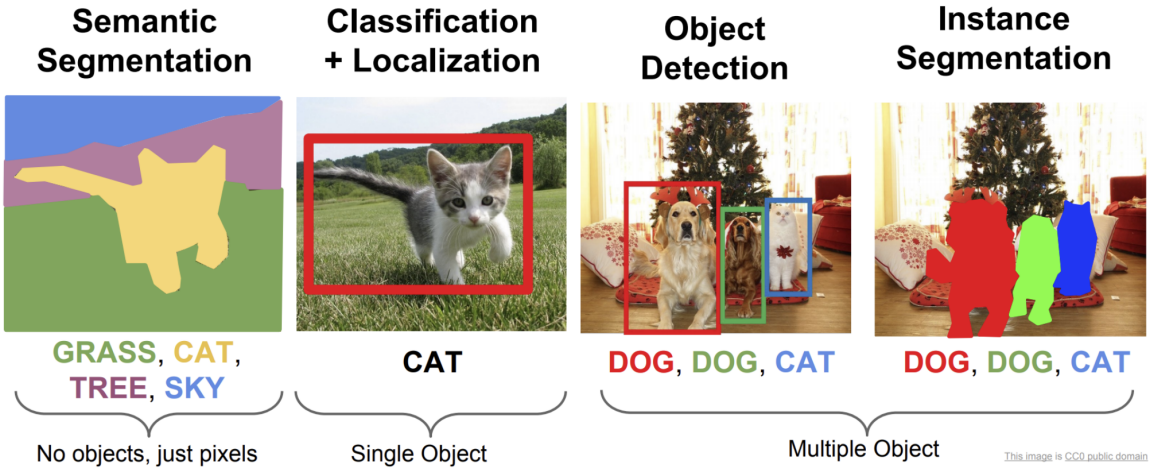


Figure 1: Instance Segmentation

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. In the later parts of the assignment, we will provide you with pretrained parts to boost the performance.

We will also provide code templates for the different parts of the assignment. These templates are provided only to help as a starting point and they are not restrictive. Feel free to structure your implementation however you want. If you choose to add a new function or class, try to add comments that are as informative as possible.

The exercise consists of two parts:

1. Part A: Implement a Region Proposal Network (RPN) [2]. (This will be provided pretrained for part B)
2. Part B: Implement and train the object detection heads [2],[1].

You are strongly encouraged to read the papers.

3 Dataset Description and Preprocessing

In this assignment your task is to detect 3 types of objects: Vehicles, People and Animals. The dataset that will be used can be found here. It 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 label list to figure out which mask belongs to which image, the same way you did it for SOLO.

After the grouping of the masks into each image, we also want to preprocess our data as follows (same way that you did in SOLO)

- normalize image pixel value to [0,1]
- rescale the image to 800x1066 (we want to keep the aspect ratio the same)
- normalize each channel with mean: [0.485,0.456,0.406], std:[0.229,0.224,0.225]
- add zero padding to get an image of size 800x1088

Do not forget to also apply the rescaling to the bounding boxes and the masks. Check the pre processing by visualizing the images of the dataset and the corresponding bounding boxes, masks. (similar to figure (2))

Useful functions in the given code template: BuildDataset.__getitem__, BuildDataset.pre_process_batch, BuildDataLoader.collect_fn

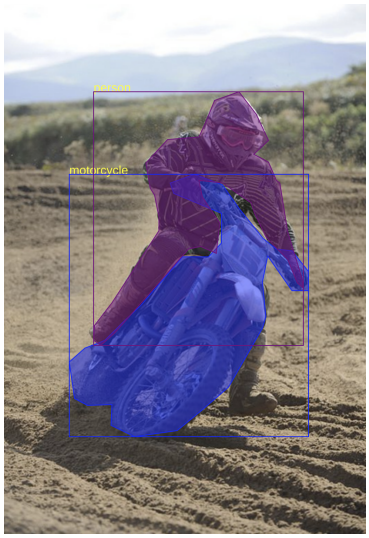


Figure 2: Datapoint

4 Region Proposal Network

Region Proposal Networks (RPNs) 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 for the RPN and the later refinement of the proposals is shown below:

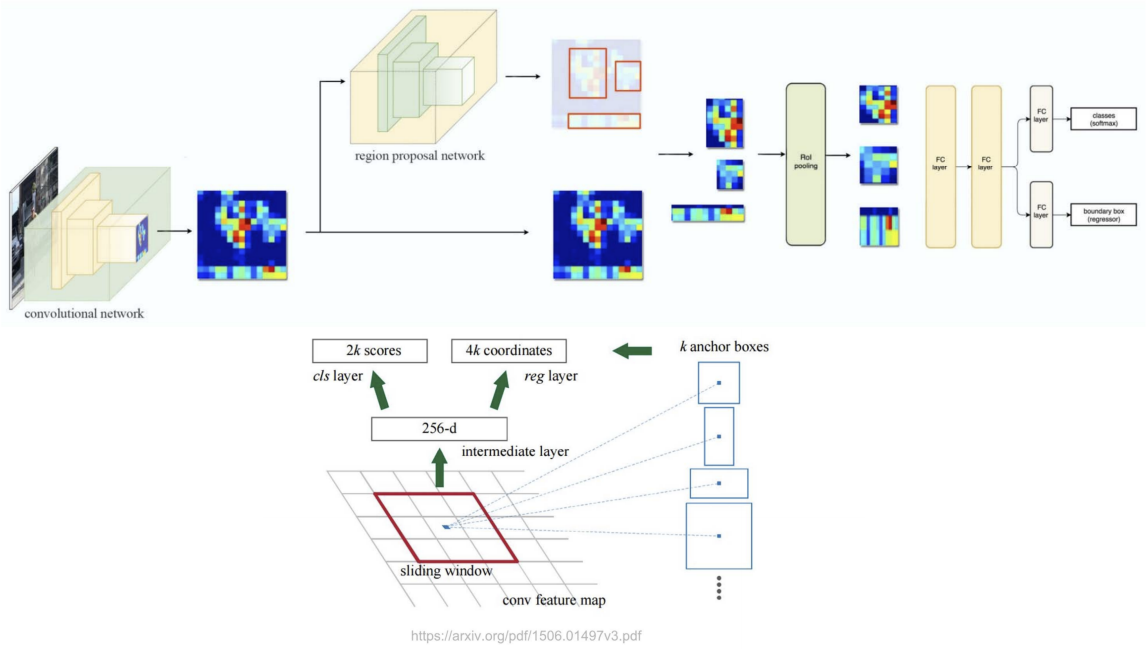


Figure 3: FasterRCNN architecture and detailed RPN

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 (This will be done on Part A).
Then the proposed bounding boxes are further refined and classified. Also inside the refined boxes a mask is proposed for the object that the bounding box contains. (This will be done on Part B).

4.1 Anchor Box Creation

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 per grid cell to simplify the assignment.

The output of the backbone produces a feature map with spatial dimensions $S_y \times S_x$. So it segments the input image into a $S_y \times S_x$ grid, and each point of the feature map corresponds to one of these grid cells (in the rest of the handout we will use the terms point of the feauture map and grid cell interchangeably).

Given an anchor box with width w and height h we define:

$$\text{aspect ratio} = \frac{w}{h}, \quad \text{scale} = \sqrt{wh}$$

Because we will use only one anchor box per grid cell, all of the anchor boxes will have the same predefined scale and aspect ratio. So for each grid cell, we assign an anchor box with the predefined scale and aspect ratio and with center the center of the grid cell. As a result for a feature map with spatial dimensions $S_y \times S_x$ we will get $S_y * S_x$ anchor boxes.

Choice of scale and aspect ratio
Make a histogram of the scales and aspect ratios of the bounding boxes of the dataset. Using these histograms choose the scale and aspect ratio of the anchor boxes that you will use. What kind of observations can you make from these histograms, how the choice of the scale and aspect ratio can effect the performance?

Using the scale and aspect ratio of your choice, assign to each point of the feature map an anchor box.

Useful functions in the given code template: RPNHead.create_anchors

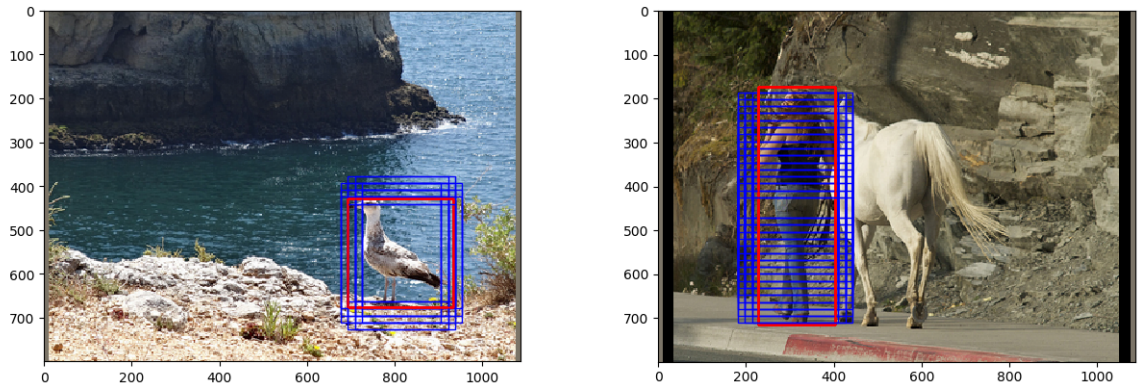
4.2 Ground truth creation

We will assign a binary label p^* (object or not) to each anchor. Firstly, cross-boundrary anchors are removed. Out of the remaining:

- positive labels ($p^* = 1$) 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 ($p^* = 0$) 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 labels are excluded from training both for the classification and the regression branch. After this assignment each anchor with a positive label is assigned a ground truth box, which is the box that the anchor has the highest IOU with. (a ground truth box can be assigned to multiple anchors, one anchor is assigned to a single ground truth box)

In figure (4) we can see some examples of anchors (blue) with positive labels and the corresponding ground truth boxes (red). To make sure that your ground truth assignment is correct, visualize the anchor box assignment similar to (4).



(a) (b)

Figure 4: Positive anchors boxes (blue) and the corresponding ground truth bounding boxes (red) [scale=256, aspect ratio=0.8]

Useful functions in the given code template: RPNHead.create_batch_truth, RPNHead.create_ground_truth

4.3 RPN Backbone

For the backbone of the RPN we will use the following convolutional network

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

The output of this backbone will then be forwarded to the RPN heads that will produce the box proposals.

Useful functions in the given code template: RPNHead.__init__, RPNHead.forward, RPNHead.forward_backbone

4.4 RPN Heads

Intermediate Layer: Define a standard convolutional layer with kernel size (3,3,256) with same padding (followed by BatchNorm and ReLU). The input of the intermediate layer is the feature map produced by the backbone.

Classifier Head: Define a convolutional layer with kernel size (1,1,1) and same padding, followed by a Sigmoid function. The input of the classifier head is the output of the intermediate layer.

If we pass the backbone's feature map of size $S_y \times S_x$ with 256 channels through the intermediate layer and the classifier head, the output will be a tensor of size $(1, S_y, S_x)$.

So the classifier outputs one objectness score p for each grid cell. (to be precise it outputs one objectness score for each anchor but in this simplified implementation we have only one anchor per grid cell)

Regressor Head: Define a convolutional layer with kernel size (1,1,4) and same padding. Similar to the classifier's head, its input is the output of the intermediate layer.

If we pass the backbone's feature map of size $S_y \times S_x$ with 256 channels through the intermediate layer and the regressor head, the output will be a tensor of size $(4, S_y, S_x)$.

So the regressor outputs one bounding box proposal for each grid cell. (again to be precise it outputs a box proposal for each anchor but here we use one anchor per grid cell)

The proposals of the regressor for one anchor are not the raw box coordinates but coordinates relative to the anchor. In particular for a single anchor, the regressor outputs a vector $\mathbf{t} = (t_x, t_y, t_w, t_h)$, which encodes the raw box proposal as follows:

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

where x, y, w, h denote the proposed box's center coordinates and its width,height. Also x_a, y_a, w_a, h_a denote the anchor box's center coordinates and its width, height.

So for one grid cell/anchor, we can use the output $\mathbf{t}$ and the anchor's coordinates to get the raw coordinates of the proposed box.

Similarly the ground truth for the regressor will be parameterized as:

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

where x^*, y^*, w^*, h^* correspond to the coordinates of the ground truth bounding box

Useful functions in the given code template: RPNHead.__init__, RPNHead.forward

4.5 Losses

Classifier's Loss:
For the classifier's loss, we compute the binary cross entropy between the predicted objectness p and the ground truth objectness p^*.
We take the binary cross entropy loss across all the anchors with positive and negative ground truth labels in the sampled mini batch. (see the remark in the end of the section for details about the sampling process)

Regressor's Loss:
For the regressor's loss we use the Smooth L1 loss between $\mathbf{t}, \mathbf{t}^*$, which is defined as:

$$L_{reg}(t_i) = \frac{1}{N_{reg}} \sum_{i=1}^{N_{reg}} p_i^* \sum_{j=1}^{4} smooth_{L1}(\mathbf{t}_i[j] - \mathbf{t}_i^*[j])$$

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 in the sampled mini-batch. Note that the regressor loss is computed only on the positive anchors in the sampled mini-batch (where p_i^* is 1).

Remark: In [2], 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. Also they set a constant size M for the mini-batch and if the positive anchors are not enough to get M total samples and keep the 1:1 ratio, they complete the batch with negative anchors.

We will follow the same approach but we will not limit the sampling to only the anchors of one image, but we will use the anchors from a small batch of images (e.g. batch size is 1 to 4 images). So the sampling process is the following

- Set a constant size M for the mini-batch
- Randomly choose M/2 anchors with positive ground truth labels. If there are not enough positive anchors choose as much as possible.
- Fill the rest of the batch with negative anchors so you will have a mini batch with M anchors

- For the mini batch that you have constructed compute the losses.

This sampling can be done inside the compute_loss function, you don't need to change your dataset's functions.

Useful functions in the given code template: RPNHead.compute_loss, RPNHead.loss_class, RPNHead.loss_reg

4.6 Training

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. Also from a small subset of your test images, plot the top 20 proposals with respect to their predicted objectness score.

4.7 Post-Processing

After decoding the boxes, clip the boundary crossing ones and keep the top K boxes (the boxes with the top objectness scores from the classifier). Then apply NMS (you can use either original NMS or the matrixNMS) and then keep the top N boxes. Leave K, N as hyperparameters for now. For a choice of K,N, plot the proposed boxes for some images of the test set, before and after the NMS.

Useful functions in the given code template: RPNHead.postprocess, RPNHead.postprocessImg

5 Submission

Write up submission must contain

1. Images with the visual correctness check similar to (2) [at least 3] (Section 2)
2. Histograms of scales and aspect ratios, along with explanation about the choice of scale and aspect ratio in your implementation. (Section 4.1)
3. Images that show the positive anchors and the corresponding ground truth boxes, similar to figure (4)[at least 4]. (Section 4.2)
4. Training and Validation curves that show the total loss, the loss of the classifier and the loss of the regressor. (Section 4.6)
5. Report of the point-wise accuracy of the proposal classifier. Plots of the top 20 proposals for some images of the test set. [at least 4]. (Section 4.6)
6. Visualization of the proposed boxes before and after the NMS [at least 4]. (Section 4.7)
7. (optional) Brief explanation about specific choices in the implementation that were not mentioned in the rest of the report.

Code submission

1. A zip file containing all the files required to run your code (data is not required). In your submitted implementation there must be a way to train your network and also to do inference and visualize the results after the post-processing.
2. A README.md file with instructions on how to run the code

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)

[4] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie: Feature Pyramid Networks for Object Detection In: CVPR (2017)

6 Optional Implementation of the RPN using FPN

The following sections describe the full implementation of the RPN for Mask RCNN, that will be required if you choose to do Mask RCNN as your final project. Even if you choose to do Mask RCNN as your final project you are not required to submit the extended implementation for this assignment. We are just providing the extra details in case you want to implement your solution in a way that is easier to transition to the full implementation. On the other hand if you end up implementing the extension of the RPN before the assignment's deadline you can submit that instead of the simplified version, you do not need to implement both (this is optional).

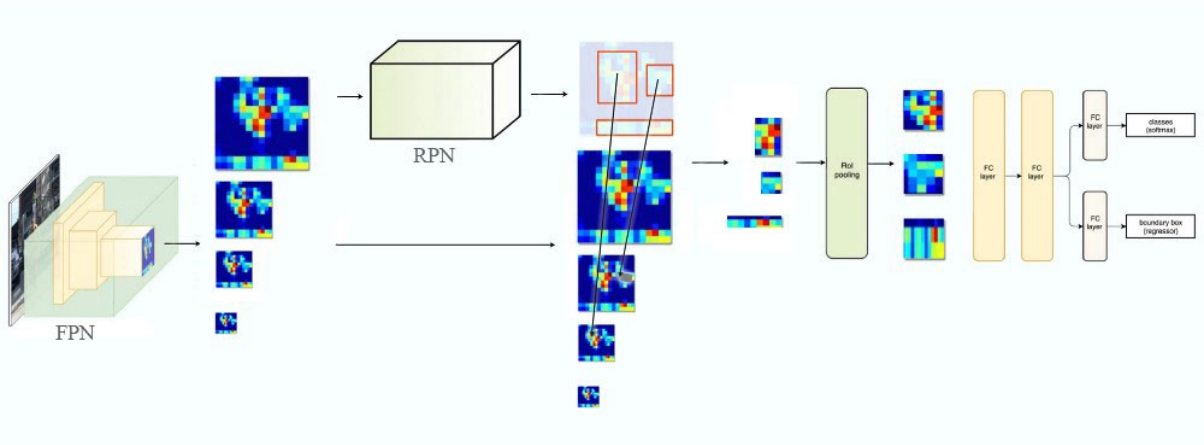


Figure 5: Detailed RPN using the FPN

6.1 Anchor Box Creation

Anchor boxes for a single level of the FPN:

The i^{th} level of the FPN produces a feature map with spatial dimensions $S_{y,i} \times S_{x,i}$. So it segments the input image into a $S_{y,i} \times S_{x,i}$ grid, and each point of the feature map corresponds to one of these grid cells. In a single level of the FPN, for each grid cell we will assign 3 anchor boxes with the same scale and aspect ratios of 1:2, 1:1, 2:1. Also their centers will be the center of their assigned grid cell.

As a result if we have a FPN level with a $S_{y,i} \times S_{x,i}$ feature map we will get $3 * S_{y,i} * S_{x,i}$ anchor boxes.

Anchor boxes for all the levels of the FPN:

To assign the anchor boxes across all the levels of the FPN we just have to repeat the process of the single level anchor box assignment for all the levels. Notice that the scale of the anchor boxes varies between the FPN levels. For the FPN levels with feature maps P_2, P_3, P_4, P_5, P_6, the corresponding anchor boxes have scales $32, 64, 128, 256, 512$ respectively.

So when we have 5 FPN feature maps, the total number of anchor boxes that we will assign will be $\{\text{Total number of Anchor Boxes}\} = \sum_{i=1}^{5} 3 \cdot S_{y,i} \cdot S_{x,i}$.

6.2 Ground Truth Creation

The ground truth is exactly the same as in section 4.2. The only difference is that now we have more anchor boxes.

6.3 Backbone

For the backnone we will use the same pretrained resnet50 FPN that we used in SOLO. It is the pretrained backbone from **torchvision.models.detection.maskrcnn_resnet50_fpn**.

6.4 RPN Heads

Intermediate Layer: Define a standard convolutional layer with kernel size (3,3,256) with same padding (followed by BatchNorm and ReLU). The input of the intermediate layer is the feature maps produced by the FPN. (one feature map at a time)

Classifier Head: Define a convolutional layer with kernel size (1,1,1*3) and same padding, followed by a Sigmoid function. The input of the classifier head is the output of the intermediate layer. Because we use same padding if we run the intermediate layer and the classifier head on a feature map of size $S_y \times S_x$ with 256 channels, the output will be a tensor of size $(3, S_y, S_x)$. So when we pass all the FPN feature maps through the RPN Heads, the classifier outputs one objectness score for each anchor for each grid cell of all the feauture maps. This means that in our architecture each channel of the classifier's output corresponds to the objectness score of one of the 3 anchor boxes in the specific grid cell.

Regressor Head: Define a convolutional layer with kernel size (1,1,4*3) and same padding. Similar to the classifier's head, its input is the output of the intermediate layer. Also if we pass a feature map of size $S_y \times S_x$ with 256 channels through the intermediate layer and the regressor the output will be a tensor of size $(12, S_y, S_x)$. So the regressor outputs one bounding box prediction for each anchor for each grid cell of all the feature maps.
Similar with the simplified version the proposals of the regressor for one anchor are not the raw box coordinates but coordinates relative to the anchor. In particular for a single anchor, the regressor outputs a vector $\mathbf{t} = (t_x, t_y, t_w, t_h)$, which encodes the raw box proposal as follows:

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

where x, y, w, h denote the proposed box's center coordinates and its width,height. Also x_a, y_a, w_a, h_a denote the anchor box's center coordinates and its width, height.
This means that in our architecture the output of the regressor which has 12 channels is splitted into 3 groups of 4 channels. Each one of the 3 groups corresponds to the encoded proposed coordinates for one of the 3 anchor boxes of the specific grid cell.

Remark: For all the feature maps of the FPN we use the same intermediate layer, classifier and regressor head.

6.5 Losses

The loss computation is exactly the same as in section 4.5

6.6 Training

The training is the same as in section 4.6

6.7 PostProcessing

The post processing is the same as in section 4.7