CIS680: Vision & Learning
Assignment 2.b: RPN, Faster R-CNN and Mask R-CNN
Due: Nov. 21, 2018 at 11:59 pm

Instructions

- This is an **individual** assignment. "**Individual**" means each student must hand in their **own** answers, and each student must write their **own** code in the homework. It is admissible for students to collaborate in solving problems. To help you actually learn the material, what you write down must be your own work, not copied from any other individual. You must also list the names of students (maximum two) you collaborated with.
- 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.
- The assignment will describe the task on a high level. You are supposed to find out how to complete the assignment in the programming framework of your choice. While the text of the assignment should be sufficient to understand the task, you are welcome to read the references that will describe the used concepts in more detail.
- All the code should be written in Python. You should use either Tensorflow or PyTorch to complete this homework.
- You must submit your solutions online on **Canvas**. You should submit **2 folders with code**, one for each part'. Submit your code compressed into a single ZIP file named "<penn_key>.zip". Jupyter notebooks are acceptable. Submit your **PDF report** to a separate assignment called "HW2.b PDF Submission". Note that you should include all results (answers, figures) in your report.

Overview

This homework is a step-by-step guide for implementing Mask RCNN, which you have seen the first homework. We will walk you through feature extraction, object localization, classification and segmentation. You will implement a simplified version of Mask R-CNN [2].

This homework consists of two parts.

1. Build Region Proposal Network (RPN) on top of the given base network architecture. RPN consists of an object classifier that determines if there is an object in the proposed region and a bounding box regression branch to predict exact object location.
2. Use ROI pooling to obtain features from proposed region and add a classifier to determine the class identity of the detected object. By now you have a simplified but proper Faster R-CNN [1]. Add a third branch that produces instance segmentation masks.

Note that the full training of a network takes much time using only CPUs. You should observe the trend of training over the first couple hundreds of iterations and decide whether to finish training or not.

Layers	**Hyper-parameters**
Convolution 1	Kernel size = (3, 3, 8). Followed by BatchNorm and ReLU.
Pooling 1	Max operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Convolution 2	Kernel size = (3, 3, 16). Followed by BatchNorm and ReLU.
Pooling 2	Max operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Convolution 3	Kernel size = (3, 3, 32). Followed by BatchNorm and ReLU.
Pooling 3	Max operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Convolution 4	Kernel size = (3, 3, 64). Followed by BatchNorm and ReLU.
Pooling 4	Max operation. Kernel size = (2, 2). Stride = 2. Padding = 0.
Convolution 5	Kernel size = (3, 3, 128). Followed by BatchNorm and ReLU.

Table 1: base network.

1 Region Proposal Network (30%)

In this part, you start building Faster RCNN by first implementing the Region Proposal Network (RPN, see Fig. 1). The base network/feature extractor architecture is specified

in the table 1 above. It corresponds to the "conv layers" part in Fig. 1, left. Region proposal network runs on top of the base network and localizes objects in an image. You will implement a simplified version of region proposal network which consists of only one set of anchors (as opposed to nine sets in the paper). It's highly recommended to read the papers [1, 2] beforehand. We will use the synthetic pedestrian and car dataset (P&C) for all experiments.

Pedestrian and Car Dataset

The P&C dataset is a toy dataset that was created by pasting images of pedestrians and cars (cropped from COCO dataset) onto synthetic backgrounds, no more than two objects per image.

Images: Images are of size $(128, 128)$. All images can be found under the `image` folder. Each image is named after one index (e.g. `000000.png`) using which you can access its corresponding annotations.

Annotations: For each image, we provide bounding boxes and segmentation masks for all objects in the image. For an example of the provided data, see Fig. 2.

Bounding boxes are parametrized by (x, y, w, h), namely column index, row index, width, height and saved in `label_car.txt` and `label_people.txt`. For example, to access the bounding box of car for image `123456.png`, read the 123456th line in `label_car.txt` and parse it into (x, y, w, h). Note that (x, y) is the position of the **upper left corner** of the bounding box. You will want to compute the center of the box from (x, y, w, h) for training.

The segmentation masks are saved as binary images for each class. For training image `123456.png`, one can find car masks as `/label/car/123456.png`.

1. (15%) Build a base network as shown in Table 1

   Use the features from the base network to build a proposal classifier (as the cls branch in Fig. 1, right) that predicts how confident the network is that a particular region contains objects (the "objectness" of a region). Specifically, add a standard convolution layer (referred as the intermediate layer) with kernel size (3, 3, 128) (followed by BatchNorm and ReLU) and a convolution layer with kernel size (1, 1, 1) without ReLU.

   You need to use the location of the ground truth bounding box to generate a 0-1 mask that will be used to train the cls branch. The pixel value of this mask is set to be 0 if anchor box at this location has less that 0.1 IoU with any ground truth object bounding box and is set to be 1 if IoU is greater than 0.5 (you can also change these

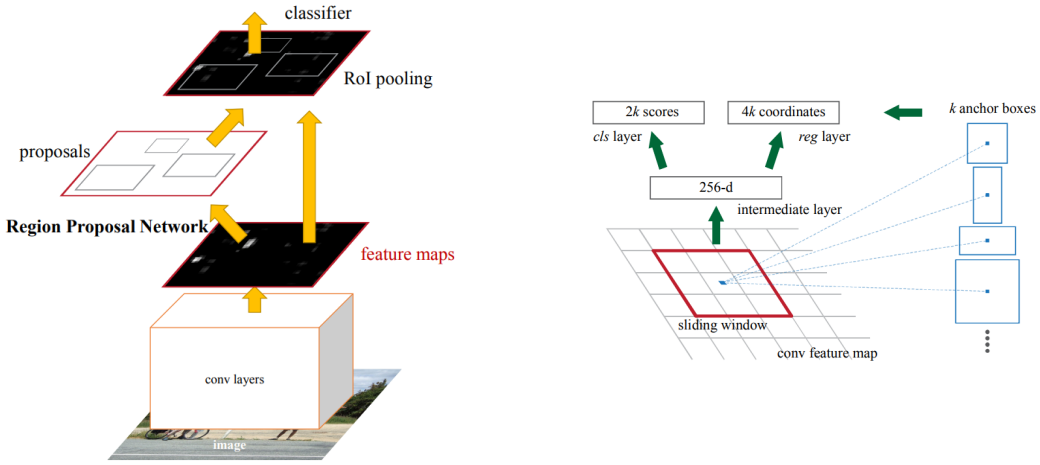


Figure 1: Faster R-CNN and its region proposal network.

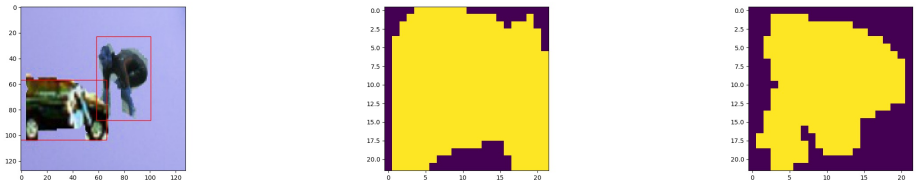


Figure 2: From left to right: image and bounding boxes, mask for car, mask for person

values if you find it essential). We simply ignore the regions for which IoU is between 0.1 and 0.5. To do this, one suggested way to compute another mask to zero out the loss in such regions.

Plot the training loss over training iterations. Report the (region-wise) test accuracy of the proposal classifier. You may also want to look at your recall and precision of the positive regions to determine if the model naively classify everything as background.

Also, there is a common pitfall in train classification network, sometimes the loss function you use have softmax embedded in it (like `torch.nn.CrossEntropyLoss`). If you already add a softmax layer at your output layer, then you are computing the softmax twice and the gradient is diminished severely. As a result, your model barely trains (and you may want to look something like `torch.nn.NLLLoss`)

2. (15%) In this question, you will build a proposal regressor (as the reg branch in Figure 1). On top of the intermediate layer, add another convolution layer with kernel size (1, 1, 4) (without BatchNorm and rectification). You can initialize the

bias of each channel to be $(0, 0, 1, 1)$, as we assume that at the very beginning, all region proposals stays where they are (anchor box locations).

The network outputs t_x, t_y, t_w, t_h defined as follows:

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

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

where x, y, and w, h denote the box's center coordinates and its width. Variables x, x_a, and x^* are for the predicted box, anchor box, and groundtruth box respectively (likewise for y and w). You can imagine the network doing the following: the network is a having a uniform sized window (the anchor box) on each region and we have a convolution layer convolving at each of these regions, predicting a (t_x, t_y) for the center of the anchor box window and (t_w, t_h) for the height and the width of the anchor box. The prediction process is as if we are moving the anchor box and stretching it to cover the object right next it.

The regressor is trained with the Smooth L1 loss defined as:

$$L_{reg}(t_i, t_i^*) = \frac{1}{N_{reg}} \sum_i \text{smooth}_{L1}(t_i - t_i^*)$$

where

$$\text{smooth}_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise.} \end{cases}$$

Note that the regressor loss is computed only on where the proposal masks predict a value > 0.5.

Train the whole network with two equally weighted losses from the proposal classifier and regressor. The training procedure is the same as the previous question.

Plot the training regression loss over training iterations. Report the final testing regression loss.

2 Faster R-CNN, Mask R-CNN (30%)

With the base network and region proposal network ready, you are now only one step from completing Faster R-CNN[4]. After finishing the whole pipeline of Faster R-CNN, you can also change the base network to see the effect of different learned features.

1. (10%) One important layer that transforms the features of the proposal into the final object classifier is the ROI pooling layer. The ROI pooling layer warps the features of an ROI region into a fixed-sized feature map.

   To save some effort, we provide a spatial transformer layer that can be used as ROI pooling layer. Check the usage in the Python file (spatial_transformer.py).

   10% extra credits will be given if you program the ROI pooling layer without using spatial transformer layer or any other bilinear sampling layer.

   The parameter θ to be fed into spatial transformer layer is as follows:

   $$\theta[:,0] = w/128,\ \theta[:,1] = 0, \theta[:,2] = (x-64)/64,$$

   $$\theta[:3] = 0,\ \theta[:,4] = w/128,\ \theta[:,5] = (y-64)/64$$

   where x, y, w are the ground truth column/row coordinates and width of the bounding box.

   Feed the original images into the spatial transformer layer with output resolution $(22, 22)$. You should get an image in which it has the bounding box region cropped and magnified. If you are implementing ROI pooling yourself, this is a good and easy sanity check to make sure you code is working properly. If you are using the provided code, you can use the ground truth box and input image to fiddle around to get familiar with the code.

2. (10%) Use the same θ parameter for the spatial transformer layer; however, feed in the feature map (conv4) from the base network and set the output size as $(4, 4)$.

   With a $(4, 4)$ output, you either

   - Flatten out the tensor to be $128 \times 4 \times 4$ dimensions and use two fully connected (followed by BatchNorm and ReLU) and a 2-way Softmax layer (with fully connected layer) for classification of people and cars.
   - OR you can add two convolution layer with 128 output channels (followed by BatchNorm and ReLU) to convolve the feature map to be $(1, 1)$. Add another layer to arrive at the classification output.

   You can experiment with this choice of architecture to see which works better. Train the network (consisting of proposal classifier, regressor, and object classifier) end-to-end with the same training procedure. Use the ground truth box to train this branch. Namely, use two sets of θ parameteres per image corresponding to the two ground boxes.

   Plot the 3 training losses over training iterations. Report the final per-pixel classification test accuracy.

3. (10%) Now you can add one more branch to implement Mask R-CNN. On top of the ROI features, add two fully convolution layer with 128 output channels (followed by BatchNorm and ReLU) and one convolution layer that predicts per-pixel class labeling. Use a binary cross entropy loss for the loss of each class binary mask. At inference time, output the class binary mask according to the classification result from the cls branch (implemented earlier).

   You have to output 22×22 masks (or roughly this size). You can do it in a couple of ways:

   - The simple is to resize the base network features two 22×22 before passing it through the mask branch.
   - Alternatively, you can use a smaller resolution as 8×8 and 4×4. Then, you will have to upsample this tensor to 22×22 by other means. You can use e.g. transposed convolutions to do this. See Odena et al. [3] for a detailed description of neural upsampling methods.

   As you cannot train the network with mask with incorrect proposals, one way is to train the mask branch separately:

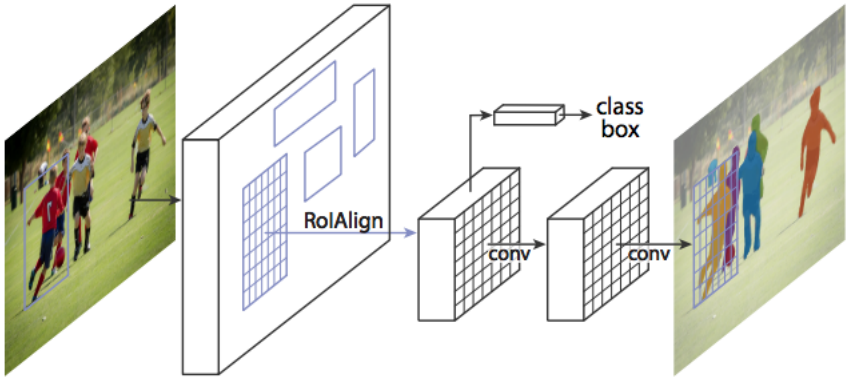
   (a) Take the computed feature from the base network.

   (b) Do ROI pooling using the ground truth boxes for that image to obtain crop and scaled features (in previous question you use this to train the classification branch).

   (c) Pass the features into the mask branch and output the mask prediction.

   (d) Resize the ground truth masks to be the same size as the predicted ones. Compute the Cross Entropy Loss for the prediction.

   Here is a figure showing the added mask head and how it processed features from a region of interest:



Plot the training losses over training iterations. Report the final classification test accuracy (i.e. the average accuracy over all pixels in the test set labels, after resizing to 22×22).

References

[1] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[2] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[3] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.

[4] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.