CIS 680: Advanced Machine Perception
Assignment 4: FasterRCNN implemantation (Part B)
Due: November 16, 2020 at 11:59pm

1 Instructions

- This is a group assignment of 2.
- The provided code template is not restrictive, feel free to implement your solution however you want
- 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 project you will implement FasterRCNN [2] which is similar to MaskRCNN [1] with the exception that it doesn't have a Mask Head that produces a mask prediction for the detected objects. FasterRCNN performs object detection and improves over the FastRCNN [3] architecture by introducing the Regional Proposal Network that you implemented in Part A of this assignment.
In figure (1) we can see the diagram of the full system. It contains:

- A FPN [4] that plays the role of a shared backbone that will be used by the other parts of the system as a share feature extractor.
- A Regional Proposal Network [2] that produces Region of Interest Proposals that are used as a fast estimation of the regions in the image that contain objects.
- A Box Head [3] that refines and classifies the boxes produced by the RPN network (this is what you will implement in this part)

You are strongly encouraged to read the papers. This way you will get a better understanding of the various choices that result in the Faster RCNN architecture.

In Part A you implemented a simplified version of the RPN and the backbone that doesn't utilize an FPN network. For this part we will provide a pretrained FPN network and a RPN that utilizes it. We will also provide a code template that you can use as a basis to write your code. Using the template is not necessary and you are free to change it and structure your code however you want.

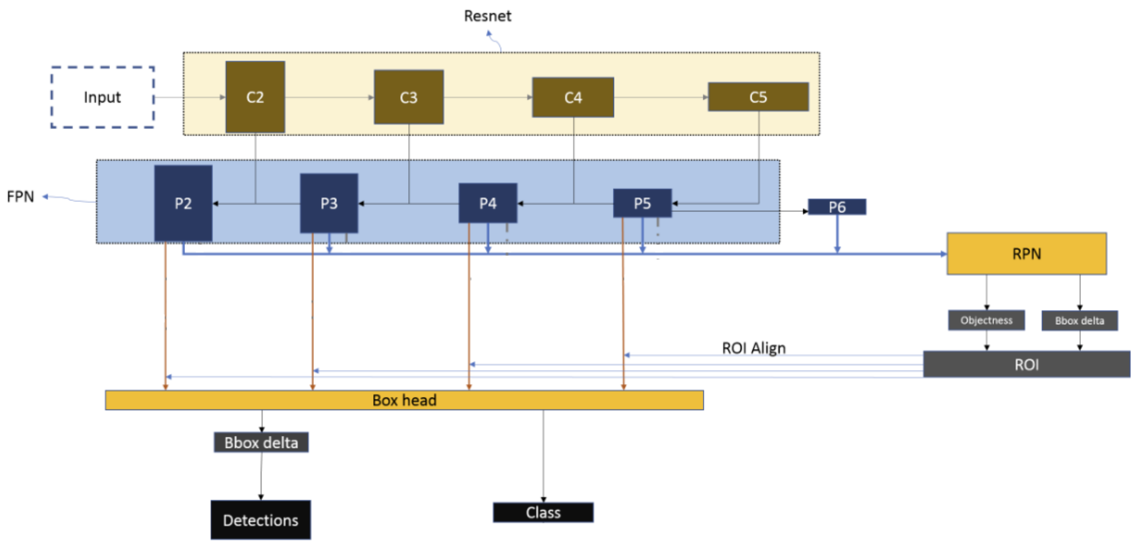


Figure 1: FasterRCNN

3 Dataset Description and Preprocessing (implemented in Part A)

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.

After the grouping of the masks into each image, we also want to preprocess our data as follows

- 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))

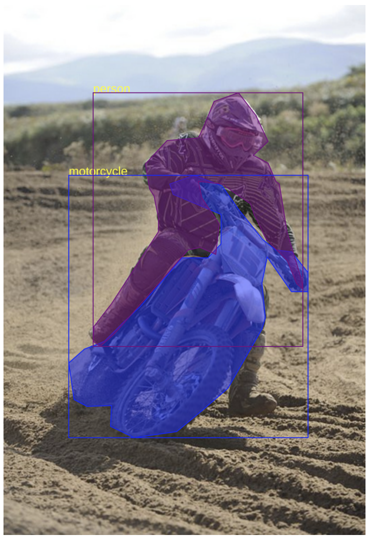


Figure 2: Datapoint

4 Pretrained Backbone and RPN

You can find a checkpoint for the pretrained Backbone and the RPN here. In the given code template we are showing how to load and use the pretrained models in the **pretrained_models.py** file.

The pretrained RPN produces for each image 1000 proposal boxes encoded in the $[x_1, y_1, x_2, y_2]$ format, which contains the coordinates of the upper left and lower right corner of the proposal. These proposals are sorted according to their confidence score in descending order. Because the images in our dataset do not contain many objects we don't want to keep all of the 1000 proposals, but we want to get the top L=200. This is easily done as shown in the given code by just choosing the first L=200 proposals for each image.

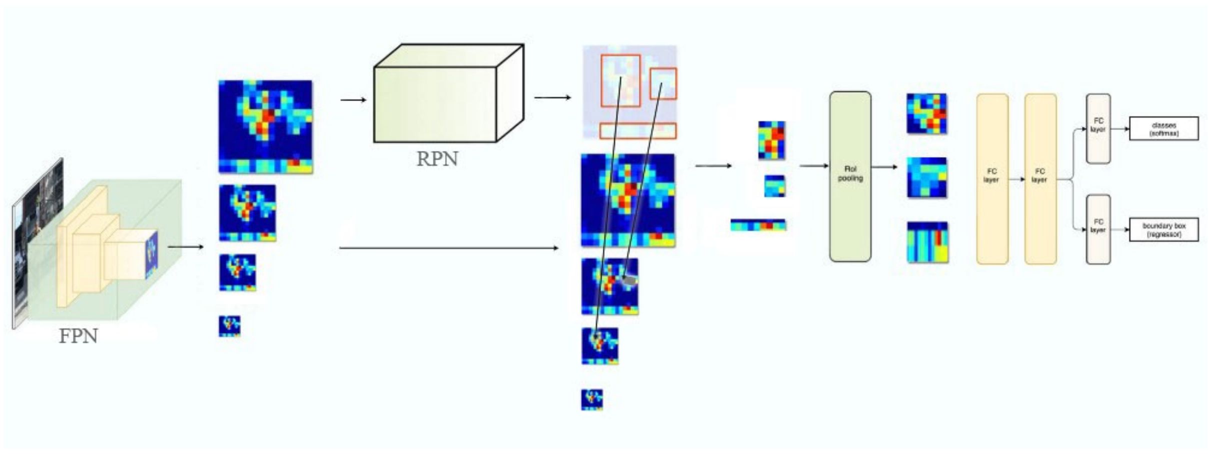
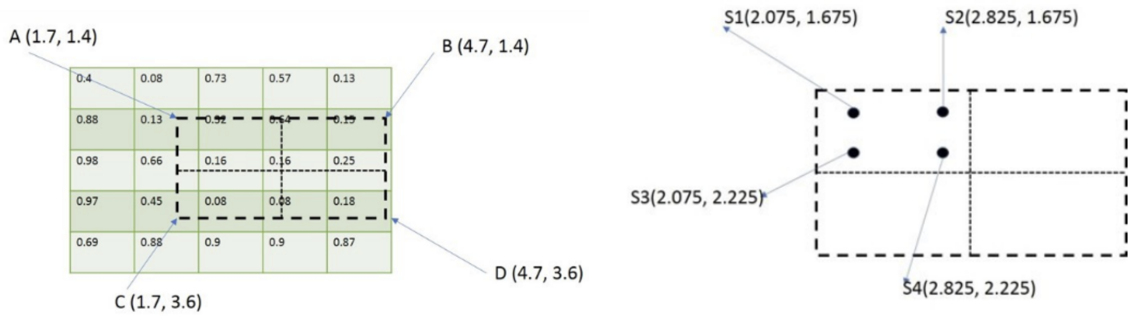


Figure 3: FasterRCNN that uses FPN as a backbone

5 Preparation of the RPN outputs for the second stage

5.1 ROI Align

The outputs of the Region Proposal Network may not be integers, which means that the corresponding predicted boxes do not align perfectly with the image. Moreover, we need to transform the region that each box surrounds into a fixed $P \times P$ matrix to train the network further.

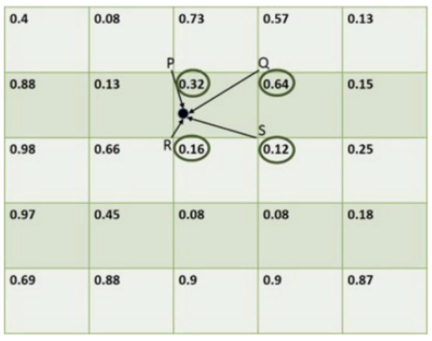


(a) (b)

Figure 4: (a) Proposed Bounding Box from RPN,(b) Region Sampling

In order to avoid quantization both for the alignment and the value assignment RoI Align was introduced in [1]. RoI Align divides the given boxes into P by P regions. Then it assigns values into those P by P cells by sampling 4 points inside each cell.
The value of each cell is the average values of $\{S_1, S_2, S_3, S_4\}$. The value of each S_i, on the other hand, can be found through bi-linear interpolation. The assumption is that the underlying image is continuous inside each pixel and it scales linearly in both directions x,y until we hit the next pixel. Each one of the S_i's is inside one cell of the feature map. So, we can use the values of the neighbouring pixels to reconstruct its value, as shown in figure (5).

For this assignment you do not need to implement ROI Align. Instead you can use the implementation given in **torchvision.ops.RoiAlign**. If we apply ROI Align for one bounding box in a feature map of the FPN, the output will be a map with the same number of channels so it will be a map represented by a tensor of size (256,P,P)



$$f(x,y) = \frac{1}{(x2-x1)(y2-y1)} \begin{bmatrix} x2-x & x-x1 \end{bmatrix} \begin{bmatrix} f(x1,y1) & f(x1,y2) \\ f(x2,y1) & f(x2,y2) \end{bmatrix} \begin{bmatrix} y2-y \\ y-y1 \end{bmatrix}$$

$$f(S1) = 0.21778 = \frac{1}{(1)(1)} \begin{bmatrix} 9.25 & 0.75 \end{bmatrix} \begin{bmatrix} 0.32 & 0.16 \\ 0.64 & 0.12 \end{bmatrix} \begin{bmatrix} 3.25 \\ 6.75 \end{bmatrix}$$

Figure 5: Bilinear Interpolation

5.2 Choose the appropriate FPN level to pool features

As shown in figure (1), for each proposal we will use one of the feature maps $P2, P3, P4, P5$ of the FPN to pool features that will be used from the later stages.
Given a proposal box with width w, height h we will pool features from the feature map P_k where:

$$k = \lfloor 4 + log_2\left(\frac{\sqrt{wh}}{224}\right) \rfloor$$

where k has a range limited from 2 to 5. Notice here that the first level of the FPN is the P2 feature map. This means that in the implemantation P2 is the first element in the list of feauture maps outputed from the backbone so it will have index 0. So if you get k=2 you will need to pool from the feature map with index 0, if you get k=3 you will need to pool from the feature map with index 1 and so on.

After finding the appropriate feature map P_k, we must find the region on the feature map that corresponds to the proposal box (the region of the proposal box is given in the image coordinates, so we have to rescale them to the coordinates of the feature map).

Finally having found the correct region in the feature map, we use ROI Align to pool a feature map with 256 channels and spatial dimensions $P \times P$. (we will use P=7 for the preparation of the features of the Box Head). For each proposal we flatten the (256,P,P) output of the ROI Align to get a feature vector of size $256 * P^2$. This feature vector will be the input of the Box Head that will refine the proposal box.

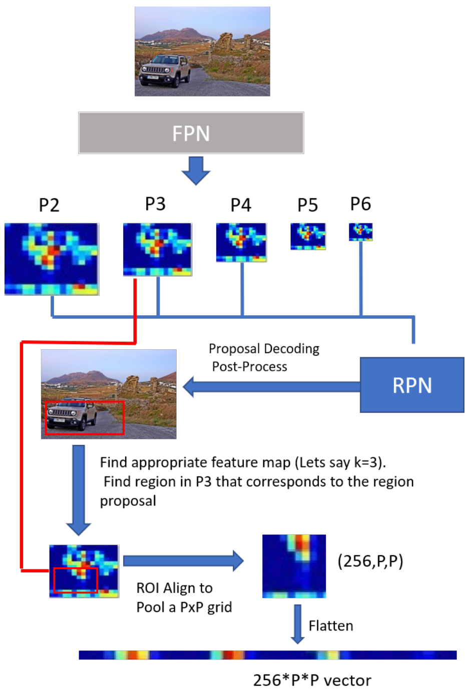


Figure 6: Preparation of feature vector for the 2nd stage of Faster RCNN

Useful functions in the given code template: BoxHead.MultiScaleRoiAlign

6 Box Head

This part of the architecture is responsible for refining and classifying the $L \approx 200$ proposals produced by the RPN. Similar to the RPN it consists of an intermediate layer a classifier and a regressor. Its input is the feature vectors with size $256 * P^2$ that was described in Section 5.

Remark 1: In order to simplify the notation we will use the symbols p, p^*, t, t^*, to denote the outputs and the ground truths of the classifier and the regressor of the Box Head. The same symbols were used in Part A to define the output and ground truth of the RPN. Make sure you do not confuse the two. In this section when we are using these symbols we are referring to the outputs and ground truths of the classifier and the regressor of the Box Head, not the RPN.

Remark 2: The Box Head processes each proposal independently but in your implementation you can stack them together in a single batch, this way you can benefit from the batch parallel computation of PyTorch. So for a single image the Box Head input is a tensor of size $(N, 256 * P^2)$

6.1 Creation of the Ground Truth

For the ground truth creation of the Box Head each proposal produced by the RPN is either assigned to a ground truth box or to the background.
Assuming we have C number of classes (in the given dataset C=3), we will give each proposal a ground truth class label p^* that assigns the proposal to one of the $C+1$ possible classes (C classes plus background). So $p^* \in \{0, \dots, C\}$, where we will use 0 as the background class.

For the ground truth assignment we have the following conditions:

1. A proposal is assigned to a ground truth box and its class, if it has an $IOU > 0.5$ with that box.
2. If a proposal has $IOU > 0.5$ with multiple ground truth boxes then it is assigned to the ground truth box that it has the highest IOU with
3. if a proposal doesn't have $IOU > 0.5$ with any ground truth box then it is assigned to the background class, so it gets $p^* = 0$.

For some images plot the proposals that are assigned to a no-background class along with their corresponding ground truth box.

Useful functions in the given code template: BoxHead.create_ground_truth

6.2 Box Head Network

6.2.1 Intermediate Layer

For the intermediate layer define a simple 2 layer MLP (2 linear layers in Pytorch) with ReLU activation functions. The size of the output of both of the 2 layers are 1024. The input of the intermediate layer is the flattened feature vector of dimension $256 * P^2$ that was produced in Section 5, after the ROI Align.

6.2.2 Classifier

For the classifier define another fully connected layer (Linear Layer) with output size of $C+1$, followed by a softmax layer. The input of the classifier is the output of the intermediate layer.
For each proposal the output of the classifier is a vector p with size $C+1$, which represents the predicted class probabilities for the proposal box.

6.2.3 Regressor

For the regressor define another fully connected layer with output size of $4*C$. The input of the regressor is the output of the intermediate layer.

In figure (1) you can see that the regressor predicts the final bounding boxes in parallel with the classifier that predicts the classes of the bounding boxes. For this reason for each proposals it predicts one bounding box for each class. During testing we keep the bounding box that corresponds to the class predicted by the classifier. Because for class i the regressor outputs a vector of encoded coordinates $\mathbf{t^{(i)}}$ with 4 elements, the total output of the regressor has the form:

$$\mathbf{t}_{\text{total}} = [t_x^{(1)}, t_y^{(1)}, t_w^{(1)}, t_h^{(1)}, \dots, t_x^{(C)}, t_y^{(C)}, t_w^{(C)}, t_h^{(C)}]$$

and as a result this is the reason the output has size $4*C$ for each proposal.

Notice that similar to the RPN, the output of this regressor uses the encoded coordinates of the bounding boxes, not the raw ones. Here the coordinates are relative to the proposal box (in the RPN they were relative to the anchor box). So

$$t_x = (x - x_p)/w_p \quad t_y = (y - y_p)/h_p \quad t_w = \log(w/w_p) \quad t_h = \log(h/h_p)$$

where x, y, w, h are the center coordinates and the width, height of the predicted bounding box. And x_p, y_p, w_p, h_p are the center coordinates and the width, height of the corresponding proposed box from the RPN. Similarly the ground truth for the regressor will be parameterized as:

$$t_x^* = (x^* - x_p)/w_p \quad t_y^* = (y^* - y_p)/h_p \quad t_w^* = \log(w^*/w_p) \quad t_h^* = \log(h^*/h_p)$$

where x^*, y^*, w^*, h^* corresponds to the center coordinates and the width, height of the ground truth box that the proposal was assigned to.

6.3 Losses

Classifier's Loss:
For the classifier's loss, we compute the cross entropy loss between the predicted class probability vector p and the ground truth class p^*. It is the same with the loss in the RPN with the only difference that now we don't have binary cross entropy because we have $C + 1$ classes.
We take the cross entropy loss across all the proposals in the sampled mini batch.

Regressor's Loss:
For the regressor loss for each proposal we use the Smooth L1 loss between the $\mathbf{t}^{(\mathbf{p}^*)}$ and t^*. This means that we compare the encoded coordinates of the ground truth bounding box with the specific prediction of the regressor that corresponds to the ground truth class. If the ground truth class is the background then we do not compute the Smooth L1 loss for this proposal. We can write the previous description as the following formula:

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

where N_{reg} is the number of proposals of the sampled minibatch, and $\mathbb{1}$ is an indicator function

For sampling the mini batch we can use a similar policy with the RPN but now we sample proposals with no-background ground truth and proposals with background ground truth with a ratio as close to 3:1 as possible. Again you should set a constant size for the mini-batch.

Useful functions in the given code template: BoxHead.compute_loss

6.4 Training

You should train the whole Box Head using the appropriate weighting so that the classifier's and regressor's loss have similar values (what we want to avoid is one of the losses having much lower value than the other one, which will result in being ignored during training). Plot the training and validation curves of the losses. For some images of the test set plot the top 20 boxes produced by the Box Head.

6.5 PostProcessing

Crop the cross boundary boxes and remove the ones where the confidence for all the no-background classes are below 0.5. Then keep the top K boxes, apply NMS for each class independently and from the remaining keep the top N boxes. You should tune K,N to find which values give you the best results during testing.

After the post processing compute the Average Precision for each category and then compute the overall mAP. Also for some images of the test set plot the predicted bounding boxes after the NMS.

Useful functions in the given code template: BoxHead.postprocess_detections

7 Submission

Write up submission must contain

1. Image plots that showcase the proposals with the no-background classes and their ground truth boxes. (Section 6.1) [at least 4] (15%)
2. Training and Validation curves that show the total loss, the loss of the classifier and the loss of the regressor of the Box Head. (Section 6.4) (20%)

3. Image plots that contain the top 20 boxes produced by the Box Head for some images of the test set (Section 6.4) [at least 4] (15%)
4. Report of the APs and the mAP. More details about how you should evaluate your model will be published soon. Points will be provided both for the implemantation of the mAP computation and the performance of your network (Section 6.5) (30%)
5. Image plots of the regressed boxes after the postprocessing. (Section 6.5) [at least 4] (20%)
6. (optional) Brief explanation about specific choices of the implemantation 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 are 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)