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

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.

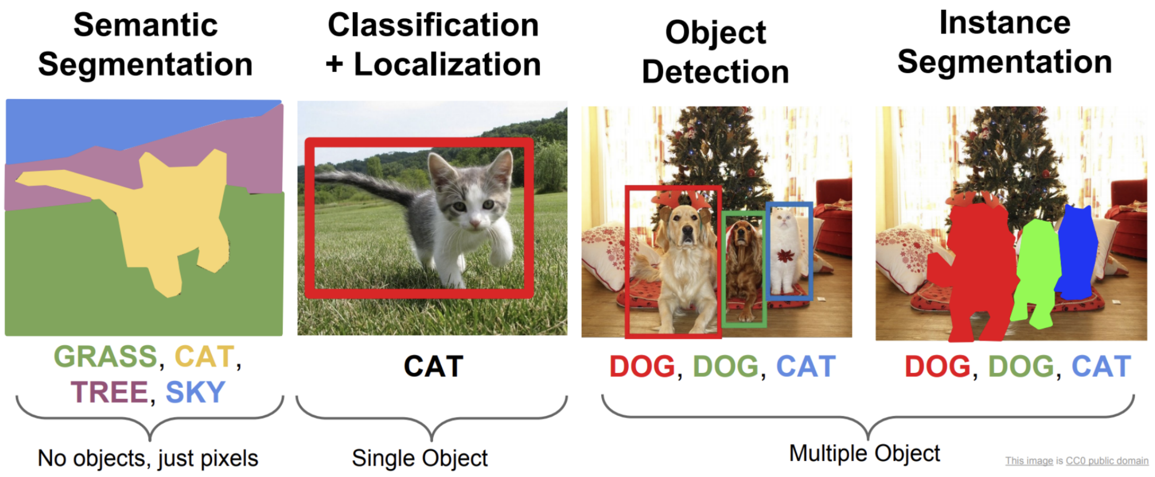


Figure 1: Instance Segmentation

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.

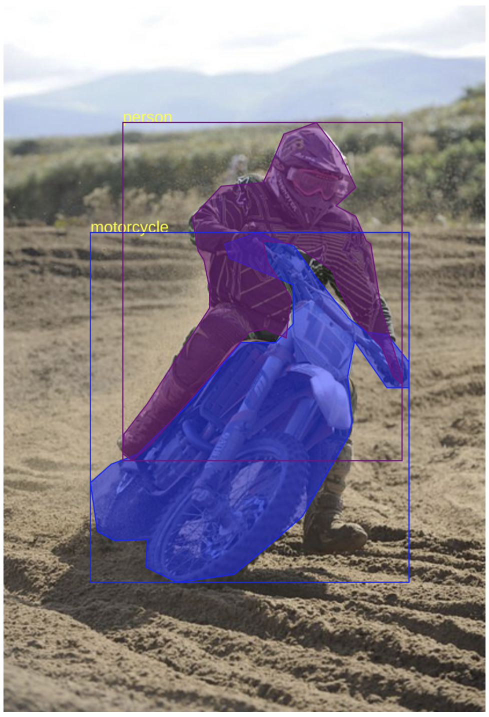


Figure 2: Datapoint

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:

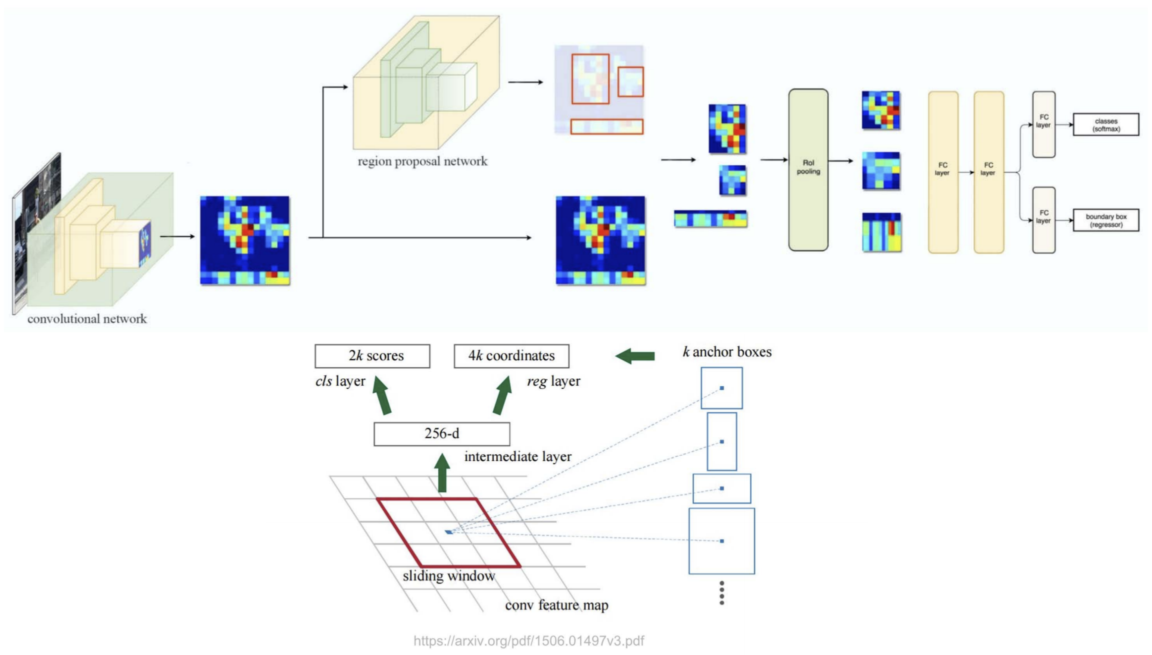


Figure 3: FasterRCNN architecture and detailed RPN

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:

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

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. 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.

4 RoI Align

You may find this article https://medium.com/@fractaldle/mask-r-cnn-unmasked-c029aa2f1296 very enlightening for the rest of the homework. This is the global picture you should have in mind.

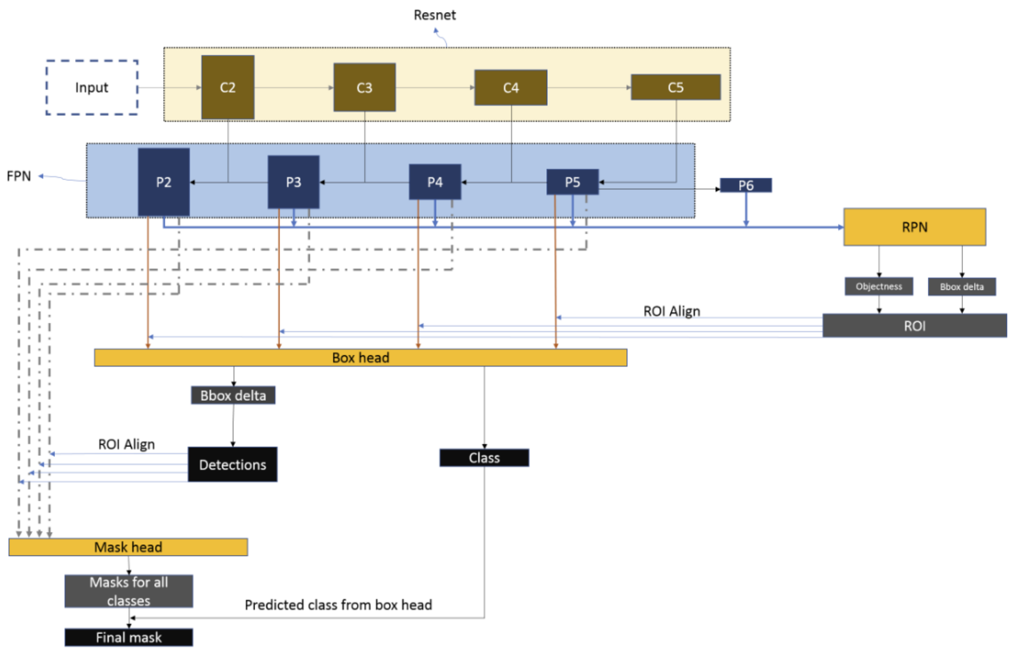


Figure 4: MaskRCNN

From this section onward you will be given a pretrained backbone, that consists of a Resnet and a Feature Pyramid Network (FPN), as well as, the Region Proposal Network that you should have implemented in part a), only with more anchors per feature in the feature map. This model is pretrained on COCO and fine-tuned in your dataset.

8. Why is it important to fine-tune our model ? Could we just skip that step, since the backbone and RPN are already pretrained on COCO and we use just a subset of COCO? What behaviour would you expect then for localization, classification and mask tasks?

With the backbone and RPN ready, we are now one step closer to completing the MaskRCNN network. The next step is a spatial transformer network, that will warp the features of a variable sized ROI region into a fixed-size feature map. Whereas, ROI pooling used in [2], worked well for object detection, it fails terribly in case of instance segmentation due to too many quantization steps which affect generation of masks, where pixel to pixel correspondence matters.

9. Prepare your images for the pre-trained model. Resize the images (together with the bounding boxes and masks) into 800x1066 images (we keep the aspect ratio of the actual images). Normalize them with mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225] and pad the width dimension to create a `[800,1088]` image, making sure that the image is still centered. Plot one image that has 2 objects (clipped, since its values will not be in [0,1]) together with its masks, labels and ground truth boxes.

10. This part is for you to get a better experience on how you can use pretrained models. Following the instructions from Google Colab and the specification of the model from the corresponding class, load the state dictionary. The inputs of the backbone are your images, transformed from step 9. What are the outputs? Explain. The input of your rpn is 1) an ImageList, which is a class with `self.tensors`(tensor) and `self.image_sizes`(List[Tuples]) fields. In this context ImageList will contain the minibatch of your images and their sizes, before this padding (be careful that maybe your dataset form part a) performs another padding too, to keep the aspect ratios, we care about this step's padding). An example of how to use it is given in the notebook. 2) A List of Tensors. In this context all the feature maps that the backbone outputs. You can use just one feature map (usually P_2 from Figure 4. 3) (when training) the target List[Dict[Tensor]] dictionary with a field `boxes` with the locations of the ground truth boxes in the same format. The output of the rpn is 1000 proposals per image, in a List of Tensors, already decoded as `[x_l,y_l,x_r,y_r]` in the image reference size and (when training) the 2 losses of the regressor and the classifier that you trained during part a). For the image from step 9, plot some of the proposals on

top of the image.

11. In the paper [2] there are 2 training schemes proposed by the authors. One is to alternate between the training of the RPN and the Heads until both converge and the next is full end-to-end training from the losses to the input. Explain the advantages and disadvantages of each.

12. (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.

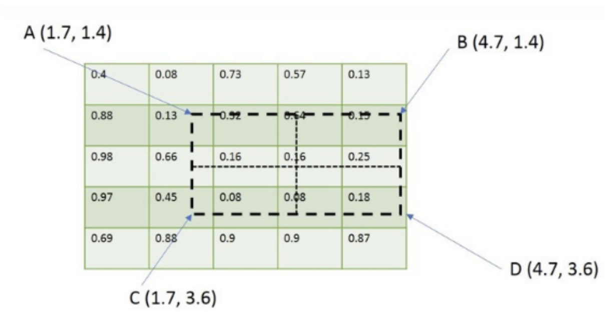


Figure 5: Proposed Bounding Box from RPN

In order to avoid quantization both for the alignment and the value assignment RoI Align was introduced in [1]. To implement a RoI align you will have to divide your boxes into P by P regions. To assign values into those P by P cells you sample (regularly or randomly) 4 points inside each cell (their coordinates not their value).

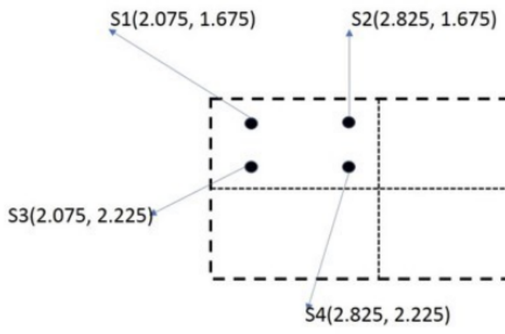
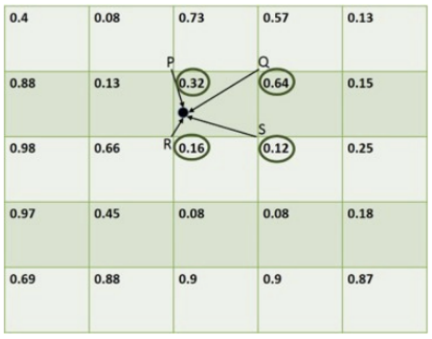


Figure 6: Region Sampling

The value of each cell will be 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.



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

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

Figure 7: Bilinear Interpolation

The output of that part should be a $P \times P$ matrix for each bounding box. $P = 7$ here. Feel free to use just one or all the feature maps of the FPN. Plot the same image from before and many of its crops after RoI Align. Try to vectorize your code as much as possible because this function will be used many times.

5 Regressor, Classifier and Mask Heads

If you see that the training takes too much time you can freeze the weights (no gradient) of the backbone and rpn for the rest of the homework.

13. Why do you think is important to have a two stage architecture (Proposal(where) and Detection(what+where)). You already implemented YOLO which is an one stage architecture. Compare two stage architectures and one stage architectures for the problem of object detection motivated by Faster-RCNN and YOLO.

14. (Classifier and Regressor)

1. (Architecture) Unroll your output into a CP^2 vector, where C is the number of channels of the FPN and create a 2 layer MLP with ReLU activation functions. The output of both layers is 1024. This is another intermediate layer, shared between the regressor and the classifier, like the one for the RPN. Now, the tasks get separated. Create two more fully connected layers, one for the classifier and one for the regressor. Do not forget (as always) to put a softmax layer to the classifier and do not forget to add the background in the classes. As before, the classification loss is cross entropy and the regression loss is smooth L1 loss.

2. (Ground Truth) Beware to assign the ground truth correctly, like you did for RPN. The same criteria apply for IOU overlap between the proposal and the ground truth. You can alternatively, not use a gray-zone and assign label (resp. background) if $IOU > 0.5$ (resp. $IOU < 0.5$). Keep in mind that the classifier now will need the actual class (or background for negative). For speed, you can subsample the ROIs. Remember that this is again your effective batch size for the regressor and the classifier. You can commit to a batch size (e.g. 128) and subsample negative and positive regions in a ratio as close as e.g. 3:1

3. (Box Encoding) From the point that proposals were produced and expressed in image coordinates we do not need the anchor boxes. However, it is easier for the training process if we regress deltas. The role of the anchors is now played by the proposals. Create the same encoding of the ground truth boxes with respect to the proposals that you used in RPN. Now you have your ground truth boxes encoded into t^* vectors and its values are relative to their corresponding proposal.

4. As you can see from the figure above the difference of the FasterRCNN heads with the RPN heads is that the regressor's head is parallel to the classifier's head (and classifier is not binary like before),

so the output of the regressor is a actually a quadruple describing the bounding box, for each class for each proposal, unrolled into a [#proposals,#classes*4] vector.

For this step report the training curve of the classifier and the regressor

15. (Post processing and metrics) After decoding, remove the invalid boxes (empty or cross boundary) and apply Non-negative Maximum Suppression at IOU=0.5 (independently for each class, except background). If the detections are many (usually 100 are enough) sort them by some confidence criterion and remove the rest. Report the test accuracy/precision/recall for the classifier. Also, look at the boxes refined by the regressor. Plot the box predicted by the regressor together with its proposal from the RPN superimposed on the image. Explain if the regressor could ever find an object that the RPN has underestimate it by (let's say) cutting it in half.

16. (Masks) At this point we have already refined the bounding boxes. After applying again the RoI Align step only with $P = 14$ this time, we end up with a `[D,256,14,14]`, where `D` < 100 tensor. Pass this tensor from a 6 layer CNN. The first four preserve the channels and the size (use a kernel size of 3×3) whereas the last two are: 1) A deconvolution layer which doubles the region's size and 2) A fully convolutional layer with kernel `[1,1,4]`. So, we produce one mask for each class, for each detection from the regressor. Since this is a pixelwise classification problem, as always we will use sigmoid to suppress the output in $[0, 1]$ range.

In the next step, filter these masks, keeping only the one that matches the output class of the classifier, for testing. For training, you will use the mask associated with the ground truth class (if it is not background) to train the mask branch. This branch is trained with binary cross entropy (BCE) loss pixelwise.

$$L_{mask} = -\frac{1}{m^2} \sum_{i,j} y_{ij} \log \hat{y}_{ij} + (1 - y_{ij}) \log(1 - \hat{y}_{ij})$$

After that, rescale the masks back to the original image, using bilinear interpolation and rescale the boxes too. Choose a threshold (e.g. 0.5) to turn the network's output into a binary mask. Report the training curves for the whole network (masks,classifier,regressor). As a reminder, you may need (like in part a)) to weight the losses differently, if they do not converge in the same speeds. You can think of this weight, as if you were training those two tasks with different learning rates. Do not only plot the total loss but the individual losses too, in the same or different figures.

17. For some images plot the predicted bounding boxes, the predicted masks and the confidence scores of the classifier on top of the image. You may have observed by now that for some images there are objects for which the ground truth boxes and masks are not given (like horses). However, at test time we do not specify how many objects each image has, so MaskRCNN may put more masks. Find one such image from the test set where MaskRCNN predicts an object even though it is not in the ground truth and plot its mask and bounding box and report which category it gets classified to and with what probability.

18. Take a picture on your own and report your MaskRCNN output.

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)