Extending Faster R-CNN to Mask R-CNN

Submitted by : Raima Sen, Shreyas Ramesh

Abstract

The primary goal achieved in this project is that of object detection along with semantic segmentation in order to perform object instance segmentation on the MS COCO dataset. The approach detects objects in an image and additionally generates a segmentation mask for each instance. Instance segmentation is challenging because it requires the correct detection of all objects in an image while also precisely segmenting each instance. It therefore combines elements from the classical computer vision tasks of object detection, where the goal is to classify individual objects and localize each using a bounding box, and semantic segmentation, where the goal is to classify each pixel into a fixed set of categories without differentiating object instances. As an extension of Faster-RCNN, we have added a branch to predict the object mask along with existing branches of class category and bounding box regression. We start off with a ResNet 50 Feature Pyramid Network as the backbone of this entire pipeline. The feature maps created by this FPN are further utilized in training the Region Proposal Network, Box Head and Mask Head. Each head is trained separately over a subset of 300 images of the MS COCO dataset due to time and computation constraints. Despite the low number of instances that the models are trained on, the results are fairly accurate.

Problem Statement

Given an image with multiple objects, Mask RCNN accurately detects multiple objects in the image by classifying them into categories, predicting their location in the image and also segmenting each instance within the image. Mask R-CNN is built over Faster R-CNN. While Faster R-CNN has two outputs for each candidate object, a class label and a bounding-box offset, Mask R-CNN is the addition of a third branch that outputs the object mask. The additional mask output is distinct from the class and box outputs, requiring the extraction of a much finer spatial layout of an object. Mask R-CNN is an extension of Faster R-CNN and works by adding a branch for predicting an object mask (Region of Interest) in parallel with the existing branch for bounding box recognition.

Related work

FPN : In Region-based Convolutional Network method (RCNN), Spatial pyramid pooling networks (SPPnets) [1] were proposed to speed up R-CNN by sharing computation. Similar to Feature Pyramid Network used in Mask RCNN, the SPPnet method computes a convolutional feature map for the entire input image and then classifies each object proposal using a feature vector extracted from the shared feature map. Features are extracted for a proposal by maxpooling the portion of the feature map inside the proposal into a fixed-size output (e.g., 6 × 6). Multiple output sizes are pooled and then concatenated as in spatial pyramid pooling. Using SPPNets, R-CNN speeds up by 10 to 100 times during testing. Training is also significantly decreased by 3 times due to faster proposal feature extraction. However SPPNets cannot be used for very deep networks as it limits the accuracy due to fixed convolutional layers.

R-CNN : The Region-based Convolutional Network method (RCNN) [2] was one of the initial object detection algorithms to achieve excellent object detection accuracy by using a deep network of convolution layers only to classify object proposals. However, the multi-stage pipeline makes it difficult to train. R-CNN first uses a deep ConvNet for object proposals using log loss. It then uses SVM to the ConvNet output. The SVM is used as an object detector in place of the softmax classifier to classify object proposals. Finally in the third stage the bounding boxes are learnt using regression. This makes training also spatially and temporally expensive. Additionally the resulting model predicts the objects slowly. The main reason why R-CNN is slow is because it performs a forward pass on ConvNet for each object proposal without sharing computation. Mask R-CNN, on the other hand curbs this by parallelly allocating computation in detecting the masks along with boxes and class labels.

Instance Segmentation : DeepMask [3] learns to propose segment candidates, which are then classified by Fast R-CNN. In these methods, segmentation precedes recognition, which is slow and less accurate. In Fully Connected Instance Segmentation (FCIS) [4] , using fully convolution layers, position sensitive output channels are predicted. These channels predict object classes boxes and masks parallely. Some issues in FCIS are : occlusion is poorly handled and edges are sometimes incorrect. Mask R-CNN is based on an instance-first strategy. It is built upon Faster R-CNN for object detections and a parallel mask head is implemented for instance segmentation.

Experiments

The overall two stage pipeline of Mask R-CNN consists of RPN (Region Proposal Network) as the first stage and a parallel bounding box and class detector and binary mask generator for each region of interest

1. Experiments with backbone architecture

a) ResNet-50-C4

b) ResNet-50-FPN

This was also recommended in the paper [5], where ResNet-101-C4 and ResNet-101-FPN were used. We noted that ResNet-50-FPN improved the speed of training and also gave more reliable region proposals in the RPN (Region Proposal Network) head. Additionally, the proposals were accurate for multiple scales of objects in different images, therefore enabling a more robust initial stage of the model.

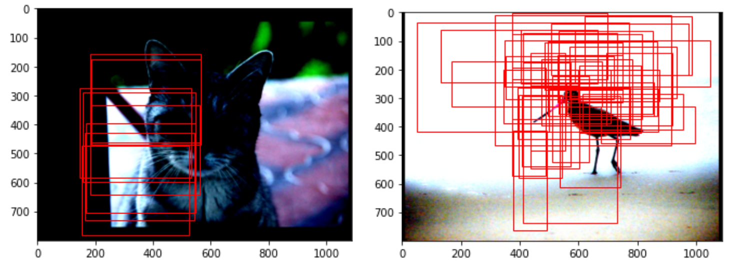

Fig.1 RPN outputs with a) ResNet-50-C4 and b) ResNet-50-FPN

2. Experiments with RoI Align, RoI pooling and RoI warping

Similar to the procedure adopted in the paper of passing the outputs of the RPN to the next stage for bounding box and mask prediction, we also experimented with RoI Align, RoI pooling and RoI warping to check which interpolation technique gave the best results. The torchvision implementation of RoI pooling and RoI align was used off the shelf; but RoI warping was implemented from scratch. It was noted that using RoI align, the masks were more pixel wise accurate than both RoI pooling and RoI warping. The effects can be seen in the image below. We only performed qualitative evaluation and noted that the absence of harsh quantizations in RoI align allowed for accurate overlap of the mask in the mask head.

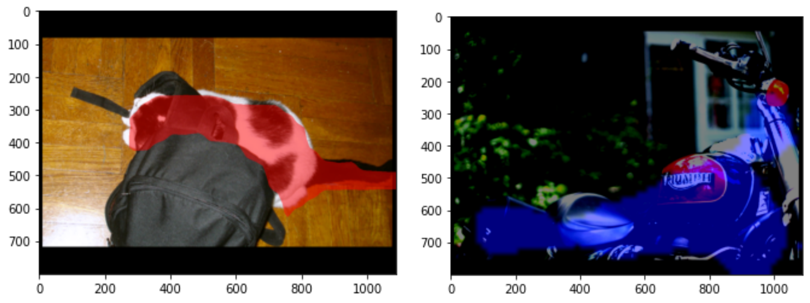

Fig.2 Mask predictions with a) RoI pooling and b) RoI align

3. Focal loss instead of pixelwise BCE loss in mask head

For the last set of experiments we used 2 different losses in the mask head : pixel wise BCE loss and Focal loss. We noted that since the dataset is highly imbalanced with most predictions detecting the background class, most of the pixels are devoted to the background. Since Focal Loss has a tunable focusing parameter ɣ, setting it to 2 allowed us to downweigh the background class and generate better masks.

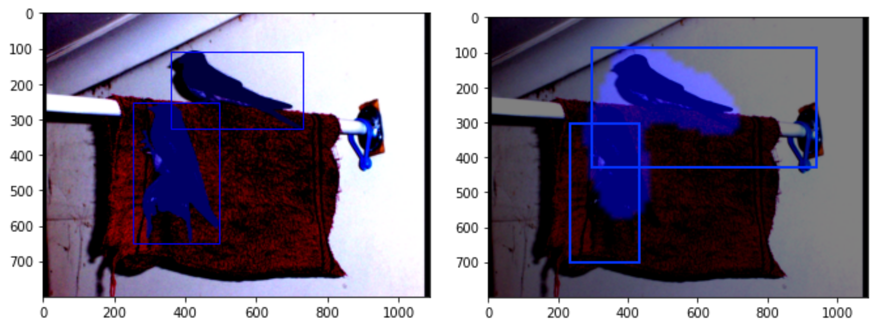
Fig.3 Mask predictions with a) Focal Loss and b) pixel-wise BCE Loss

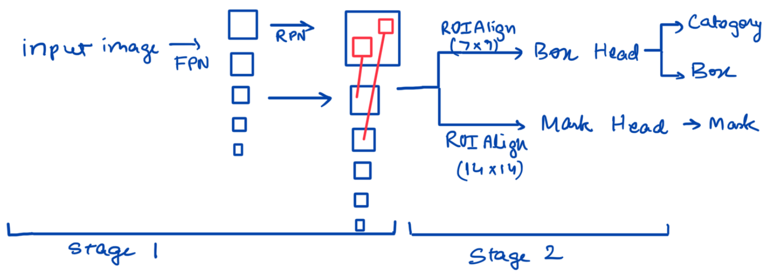

Fig.4 Mask R-CNN Pipeline

Proposed Methodology

In this project we implement an end to end object detection and instance segmentation method called Mask R-CNN by building a parallel mask detector on top of the Faster R-CNN architecture which contains a bounding box and class detector.We first implement Faster R-CNN from scratch and subsequently do stagewise training while freezing the parameters of the previous parts.

Data Preprocessing and input

The MS COCO dataset contains 328000 images and their corresponding bounding boxes,labels and masks. Due to lack of computational resources (limited to Google Colab Pro), we trained the different stages of Mask R-CNN on a small subset of 300 images and their corresponding bounding boxes, labels and masks.

Image Preprocessing :

The raw images in the dataset had 3 channels corresponding to RGB and each image was of size (300 x 400). We normalized the pixel values between [0,1], rescaled the image to size (800 x 1066), further normalized each channel with mean : [0.485,0.456,0.406], std:[0.229,0.224,0.225] and added zero padding to get an image of size (800 x 1088)

Mask Preprocessing :

The masks from the dataset were resized to (800 x 1066) and a zero padding was applied to form masks the same size as the images

Bounding Box Preprocessing :

The previous bounding boxes in the dataset were of the form (x_1,y_1,x_2,y_2) with respect to the previous dimensions of the image. These 4 dimensions were scaled as per the new image dimensions (800 x 1088) and again a zero padding was applied

Design and Architecture

The following implementation uses a batch size = 2 for all the heads.

Stage 1 : Implementation of the RPN Head

The RPN Head gives a first broad estimate of the object/background in the image. Using the feature maps generated by ResNet-50-FPN an initial set of 50 proposal boxes were predicted by this head. The low number of proposals selected after NMS was due to the small set of data that we were training on. The following steps were adopted to implement the RPN Head :

1. To parameterize the outputs of the RPN, we first generated a set of achor boxes using 5 scales [32, 64, 128, 256, 512] and 3 aspect ratios [1:2, 1:1, 2:1]
2. Next we defined the ground truth across each FPN level

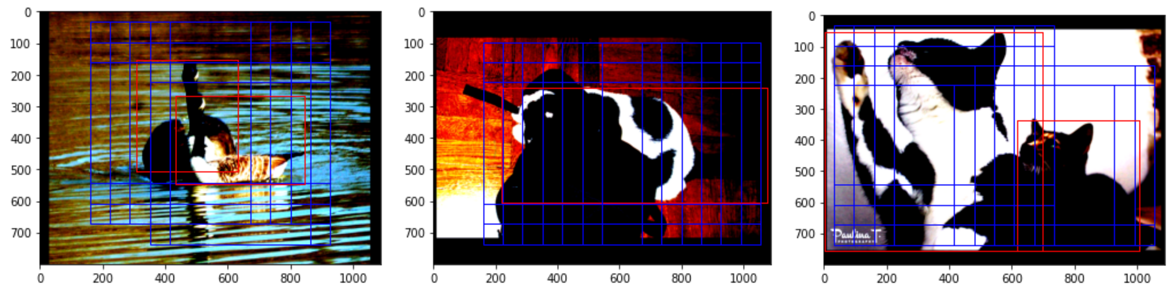

Fig.5 Ground Truths : RPN Head

3. Using the architecture shown in Table.1, we defined the intermediate layer containing convolutional layers and the subsequent classifier and regressor branches
4. The loss function has 2 components : classifier loss and regressor 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. The mini batch sampling is done such that the positive and negative anchors are spread in 1:1 ratio to reduce bias towards background class. Subsequently, the regressor loss is calculated as the Smooth L1 loss between the encoded ground truth bounding boxes and the predicted bounding boxes from the regressor branch. Regressor loss is only calculated on the positive anchors in the sample minibatch. The following equations are used :

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1-y_i)\cdot log(1-\hat{y}_i)$$

$$L_{\text{loc}}(t^u, v) = \sum_{i\in\{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

5. We use Adam optimizer with learning rate 0.001 over 20 epochs in training the network. It took about 45 seconds per epoch.
6. It must be noted that the outputs of the regressor branch are bounding boxes in encoded form :
$t_x = (x - x_a)/w_a$ $t_y = (y - y_a)/h_a$ $t_w = \log(w/w_a)$ $t_h = \log(h/h_a)$
Therefore before post processing in validation mode, we must decode them to make them comparable to the ground truth bounding boxes.
7. Finally in the post processing step, we find the top 50 object proposal boxes using iterative Non Max Suppression and an IoU threshold of 0.5

Preparation for Stage 2

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 × P matrix to train the network further.
RoIPool is a standard operation for extracting a small feature map (e.g., 7×7) from each RoI. RoIPool first quantizes a floating-number RoI to the discrete granularity of the feature map, this quantized RoI is then subdivided into spatial bins which are themselves quantized, and finally feature values covered by each bin are aggregated (usually by max pooling). These quantizations introduce misalignments between the RoI and the extracted features. While this may not impact classification, which is robust to small translations, it has a large negative effect on predicting pixel-accurate masks. Therefore, RoI Align uses bilinear interpolation to compute the exact values of the input features at four regularly sampled locations in each RoI bin, and aggregate the

result using average. We also compare to the RoIWarp operation. Unlike RoIAlign, RoIWarp overlooked the alignment issue quantized RoI just like RoIPool. So even though RoIWarp also adopts bilinear resampling, it performs on par with RoIPool. We apply the ROI Align for each channel of the feature map independently. So 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).
Next, we have to correctly choose the FPN level to pool features which is done with the following formula,

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor$$

where $k_0 = 4$

k has a range limited from 2 to 5. After finding the feature map Pk, we must find the region on the feature map that corresponds to the proposal box. 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.

<u>Stage 2.A : Implementation of the Box Head</u>

The flattened feature vectors of shape $(50, 256 * 7^2)$ are passed into the box head network for category and bounding box prediction. The following steps were adopted :

1. 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). We will use 0 as the background class. A proposal is assigned to a ground truth box and its class, if it has an IOU > 0.4 with that box.

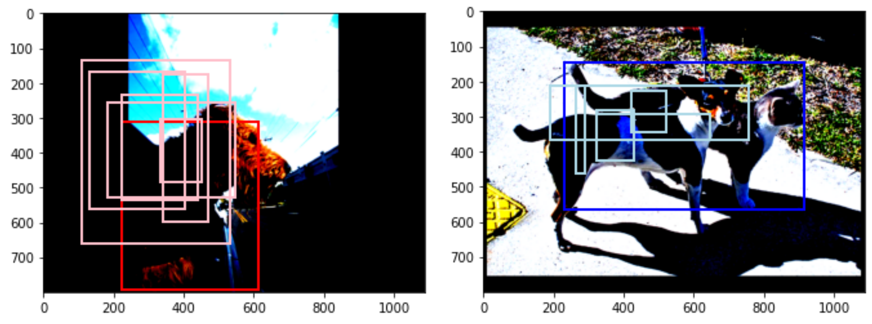

Fig.6 Ground Truths : Box Head

2. Using the architecture shown in Table.2, we defined the intermediate layer containing linear layers and the subsequent classifier and regressor branches
3. The loss function has 2 components : classifier loss and regressor loss. In this step we compute the cross entropy loss across all the proposals. For sampling the mini batch we 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. Subsequently, the regressor loss is calculated as the Smooth L1 loss between the encoded ground truth bounding boxes and the positive predicted bounding boxes. The following equations are used :

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes.}$$

4. We use Adam optimizer with learning rate 0.001 over 20 epochs in training the network. It took about 30 seconds per epoch.
5. Using decoding of bounding boxes and post processing similar to the one adopted in RPN head, we get top 20 bounding boxes from the Box Head.

Stage 2.B : Implementation of the Mask Head

Similar to the RoI Align step taken to transform the region and generate feature vectors to pass into the box head, this time we used P=14 to create new feature vectors from the RPN head and flatten them. The flattened feature vectors of shape $(20, 256 * 14^2)$ are passed into the mask head network for pixel wise mask prediction. The following steps were undertaken :

1. Using the architecture shown in Table. 3, we defined the layers of the mask branch
2. The mask target is the intersection between the bounding box produced by the Box Head and its associated ground-truth mask. The original mask target does not have the same dimensions with the output of the Mask head which has dimensions of 28×28. So after finding the intersection between the bounding box and the ground-truth mask we need to rescale it to the same dimensions as the output of the Mask head.
3. Pixel wise BCE loss and Focal loss is computed between the mask targets and the outputs of the mask branch. The following equations are used:

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

$$\mathrm{FL}(p_t) = -\alpha_t(1 - p_t)^{\gamma} \log(p_t)$$

4. The SGD optimizer with a weight decay of 1e-4 and a momentum of 0.9 is used. It took ~1 minute per epoch
5. In validation mode, post processing is done by rescaling the predicted masks back to the original image using bilinear interpolation and rescaling the boxes too.

Results and Potential Work

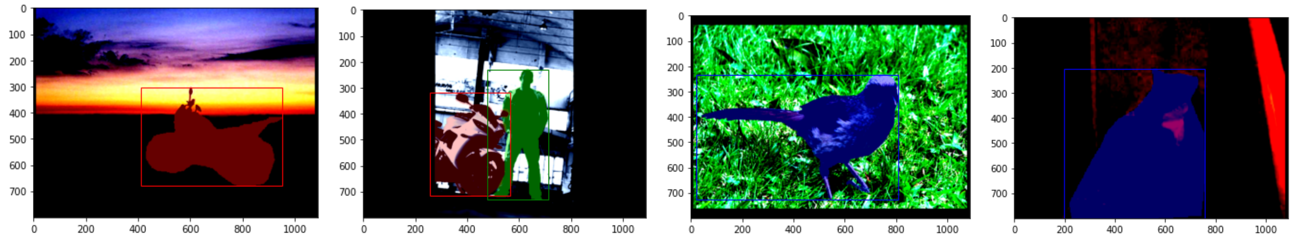

Fig.7 Mask R-CNN outputs

It was noted that the model performed fairly accurately despite training on a small subset of the dataset. Our observation is that the reason could be because of a RPN network that generated reliable initial proposals. Additionally the use of RoI Align to extract a smaller feature map from each RoI improved the quality of input feature vectors at each stage. Lastly the use of varied aspect ratios of anchor boxes in the RPN head led to more robust detections.

As a part of future work over this project, we firstly aim to train all heads together. Additionally, the use of mask branch along with different FPN levels can be extended to depth estimation. The paper recommends a fairly simple architecture of the mask branch, but we would like to explore how more complex architectures affect the performance of Mask R-CNN.

Conclusion

Mask R-CNN is an intuitive extension from Faster R-CNN with a few unique corrections for instance segmentation task, including RoIAlign and a parallel FCN mask head. RoIAlign is proposed to combat quantization from RoIPool to protect the pixel-to-pixel alignment. Decoupling the segmentation task from classification and bounding box regression is preferred than the coupling of multiple tasks. However it is uncertain as to why decoupling is more desirable than making masks across classes compete against each other. An implicit trade-off of the Mask R-CNN design is the accuracy vs. speed balance, since Mask R-CNN uses the Faster R-CNN network, it also inherent the speed limitation from it as a two-stage network. Compared to a single-stage network YOLO, Mask R-CNN is slower in the object detection task, but more accurate thanks to its a localization step that preserves the spatial coherence for the segmentation. Overall, this extension is intuitive, and it generates non-trivial improvement on multiple tasks including object detection and instance segmentation with the same framework.

References

1. K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in European Conference on Computer Vision (ECCV), 2014.
2. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014
3. P. O. Pinheiro, R. Collobert, and P. Dollar, “Learning to segment object candidates,” in Neural Information Processing Systems (NIPS), 2015.
4. J. Dai, Y. Li, K. He, and J. Sun. R-FCN: Object detection via region-based fully convolutional networks. In NIPS, 2016.
5. He, K., Gkioxari, G., Dollar, P., Girshick, R.: Mask R-CNN. In: ICCV. (2017)

Appendices

Branch	Layer	Hyperparameters
Intermediate	Conv2d	Kernel size =3×3×256, pad =’same’ followed by BatchNorm and ReLU
Classifier	Conv2d	Kernel size =1×1×num_anchors, pad =’same’ followed by Sigmoid
Regressor	Conv2d	Kernel size =1×1×4*num_anchors, pad =’same’

Table 1: RPN Head structure

Branch	Layer	Hyperparameters
Intermediate	Conv1d	in_channels = 256 x P^2, out_channels = 1024, followed by ReLU
	Conv1d	in_channels = 1024, out_channels = 1024, followed by ReLU
Classifier	Conv1d	in_channels = 1024, out_channels = C + 1
Regressor	Conv1d	in_channels = 1024, out_channels = 4*C

Table 2: Box Head structure

Branch	Layer	Hyperparameters
Layer 1	Conv2d	Kernel size =3×3×256, pad =’same’ followed by ReLU
Layer 2	Conv2d	Kernel size =3×3×256, pad =’same’ followed by ReLU
Layer 3	Conv2d	Kernel size =3×3×256, pad =’same’ followed by ReLU
Layer 4	Conv2d	Kernel size =3×3×256, pad =’same’ followed by ReLU
Layer 5	ConvTranspose2d	Kernel size =3×3×256, pad =1 followed by ReLU
Layer 6	Conv2d	Kernel size =1×1×C,followed by Sigmoid

Table 3: Mask Head structure