VRDL-Homework3

Name: Kuok-Tong Ng
Student ID: 309652030

Project repository: K-T-Ng/VRDL-Homework3 (github.com)

1. Introduction

This homework is about **instance segmentation**. We are dealing with a **Nuclear segmentation dataset** contains 24 training images with 14,598 nuclear and 6 test images with 2,360 nuclear. The following figure shows one image from the dataset, there are many nuclei in this image. Our goal is to train an instance segmentation model in order to segment all the nuclei in the image.

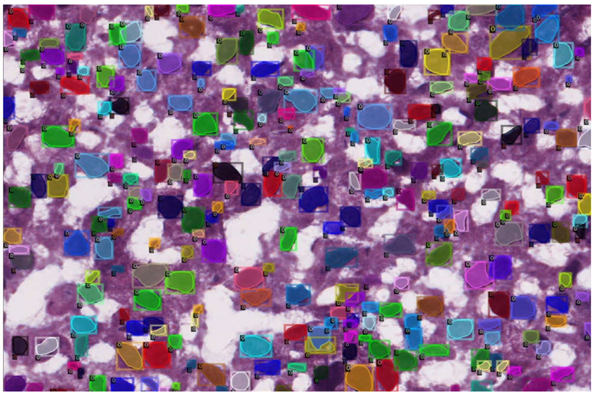


Figure 1: An image from dataset

2. Methodology

In this homework, we train a Mask-RCNN model by using Detectron2 [3]. However, Detectron2 is officially not supported on Windows10, so we use [4].

2.1 Model architecture

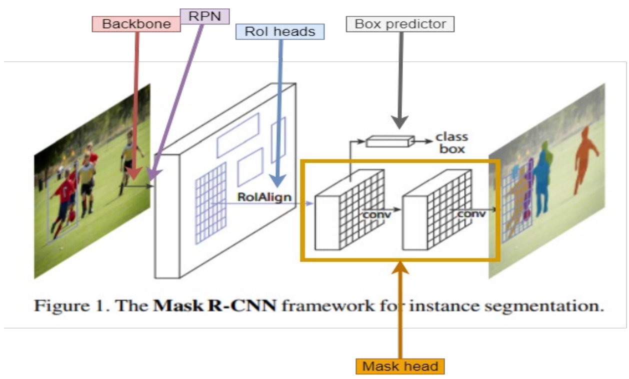


Figure 2 Mask R-CNN framework (from [1])

There are several main components in Mask-RCNN framework (see Figure2). The following subsection will explain the details based on the source code that we used (Detectron2 [4]). ***Suppose input image size is (3, 800, 800), let's see what will happen in the following.***

2.1.1 Backbone

The goal of backbone is to extract the features from the image. We use ResNet-101 as our backbone architecture. We use the output feature map of conv4_x as the output feature map. Moreover, we freeze conv1 and conv2_x during the training procedure. ***As we see in Figure3, the down sample factor (until conv4_x) is 16. Therefore, the size of the output feature map is (1024, 50, 50).***

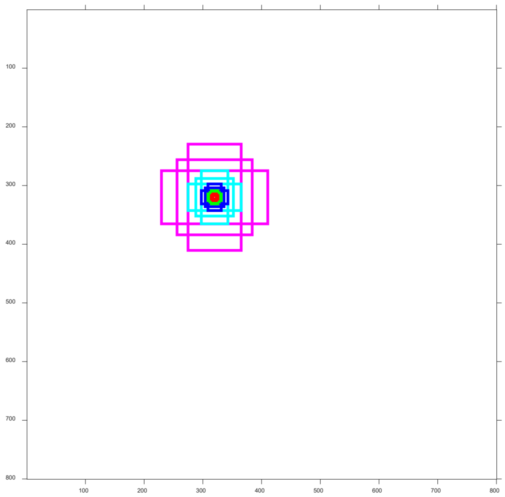
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

Figure 3: ResNet architecture (from ResNet paper)

2.1.2 Region Proposal Network (RPN)

The goal of RPN is to generate proposal. In 2.1.1, we obtained a feature map of size (2048, 25, 25). At each position in the feature map, k=15 anchors are sampled (Five scales: 8*8, 16*16, 32*32, 64*64, and 128*128 with three aspect ratios: 1*1, 1*2, and 2*1). Since the down sample factor is 32, this procedure can be interpreted as "start form (0, 0), assigned 15 anchors for stride=32". For example, our image size is (800, 800), so there are 15 anchor boxes centered at (320, 320).



First step: For each position of the feature map, RPN network will output

- k objectless score: In order to indicate is there any object in those anchors. The source code is slightly different with the paper, it only returns k scores rather than 2k scores, sigmoid rather than softmax.
- 4k regression deltas: Indicate the offset and the size changes of predict bounding box from the anchor box. From now on, suppose every bounding box is expressed as(x, y, w, h), indicates the box center, width and height. Suppose for an anchor box (x_a, y_a, w_a, h_a), and the corresponding delta (t_x, t_y, t_w, t_h), then the predict bounding box can be computed by

$$x = x_a + t_x * w_a, \qquad y = y_a + t_y * w_a$$
$$w = w_a * \exp(t_w), \qquad h = h_a * \exp(t_h)$$

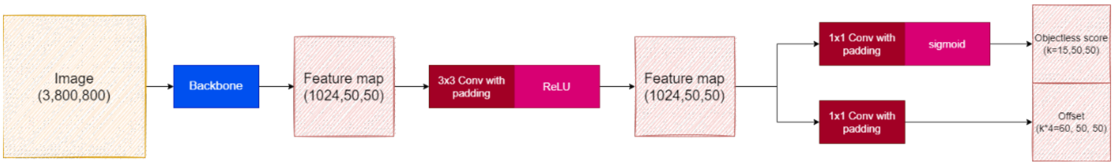


Figure 4: RPN

Second step: Calculate the loss of RPN network. There are many anchors in the image (in our example, there are 50*50*15=37500 anchor boxes in total). Each anchor box will be assigned a label (Positive, Negative, Ignore).

- Positive: anchors with highest IOU with one of the ground-truth box or IOU > 0.7 with the ground-truth box.
- Negative: IOU < 0.3 with all ground-truth box.
- Ignore: neither positive nor negative. These anchors will not contribute to the loss.

After assigning these labels, the loss contains of two parts:

- Loss_rpn_cls: For the class layer (upper part of Figure4), the loss function is binary cross entropy, the label of (positive/negative) anchor box is (1/0), respectively.
- Loss_rpn_loc: For the regression layer (lower part of Figure4), the network is optimize by using the deltas(t_x, t_y, t_w, t_h), not the exact locations. Therefore, we need to convert the ground-truth boxes (x^*, y^*, w^*, h^*) to the deltas by using the following formula

$$t_x^* = \frac{x^* - x_a}{w_a}, \qquad t_y^* = \frac{y^* - y_a}{h_a}$$
$$t_w^* = \log\left(\frac{w^*}{w_a}\right), \qquad t_h^* = \log\left(\frac{h^*}{h_a}\right)$$

The loss function of this layer is smooth L1 loss (see Figure5), compare to L2 loss, the gradient of the outlier is smaller, which makes the training procedure become more stable (avoid exploding gradients).

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

Figure 5: Smooth L1 loss.

The regression loss of this layer is

$$L_{loc}\left((t_x, t_y, t_w, t_h), (t_x^*, t_y^*, t_w^*, t_h^*)\right) = \sum_{\mathrm{i} \in \{\mathrm{x,y,w,h}\}} \mathrm{smooth}_{\mathrm{L}_1}(t_i, t_i^*)$$

Note that this sum is only calculated for positive anchors.

Final step: Before go into RoI head, we decode the deltas and apply non-max suppression (NMS) with threshold 0.7 to remove those highly coincide boxes. Moreover, we choose at most (12000/6000) proposals before applying NMS and (2000/1000) after NMS during (training/testing), respectively, with the highest objectless score.
Hence, we got a ***list of bounding boxes with objectless scores*** here.

2.1.3 RoI head (RoI align + Res5 block)

Since Box head (2.1.4) contains fully connected layer (which requires same input size) and our predicted bounding boxes (and hence RoI) above having various sizes. Therefore, before feeding into Box head and Mask head, we may need to resize the feature map to a fixed size (2048*7*7 in this homework) from each RoI.
The RoI head we use in this homework is shown in Figure 6, it consists RoI Align and ResNet stage 5 (last stage of ResNet101, backbone does not use).

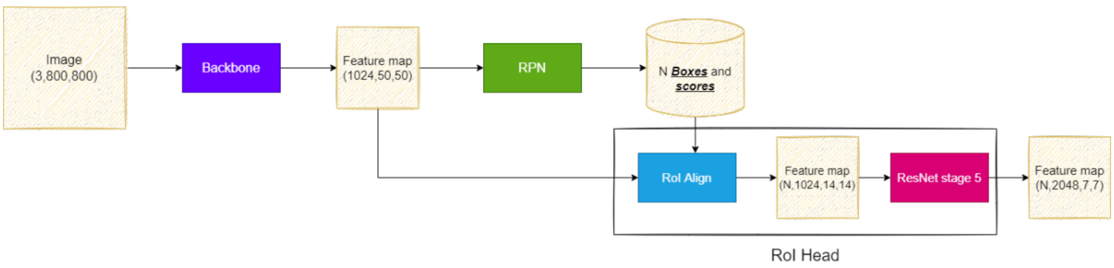


Figure 6: RoI Head

- RoI Align: Suppose we have a (1024, 50, 50) feature map (in sec 2.1.1) and one of the predict bounding box is $(x, y, w, h) = (202, 112, 189, 74)$, it indicate the center coordinate, width and height of the bounding box. We know there is a down sampled factor 16 in the backbone. Therefore, the corresponding position in the feature map is $(x, y, w, h) = (12.625, 7, 11.1825, 4.625)$. We use this example and Figure 7 to illustrate what happened in RoI align.
  Our goal is to resize this RoI into a (1024, 7, 7) feature map. It can be explained

in multiple steps in Figure 7.

(a). This indicate how the box looks like in the feature map.

(b). Split this area to 14*14=196 bins.

(c). Sample 2*2=4 points in each bin.

(d). Use bilinear interpolation to find the sample value.

Finally, perform average pooling for those 4 points to represent the value of that bin. Compare with RoI pooling, RoI align doesn't apply quantization (rounding) procedure to the box. Quantization will lead to misalignment (For example round(12.625)=13, by multiplying the down sample factor 16, there are (13-12.625)*16=6 pixels loss of the original position information). RoI align solves this problem.

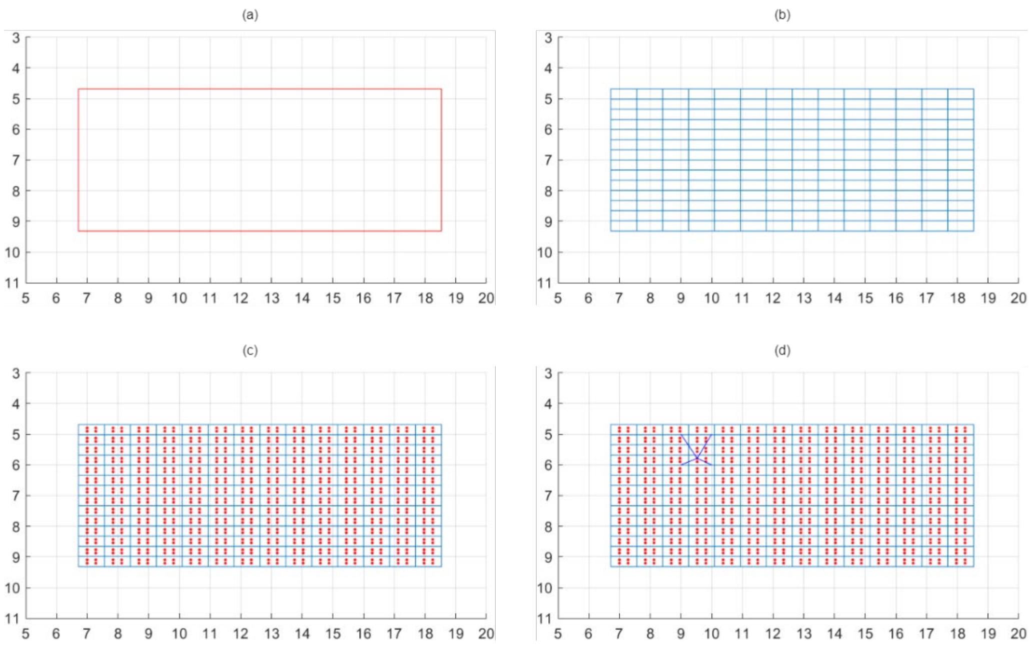


Figure 7: RoI align

Finally we got ***N feature maps of size (1024, 14, 14) under N bounding boxes***.

- ResNet stage 5

  The code in Detectron2 add ResNet stage 5 (conv5_x in Figure 3) after RoI align, it also down sampled by a factor 2. Therefore, we got ***N feature maps of size (2048, 7, 7) under N bounding boxes.***

2.1.4 Box head

The goal of box head is to predict which class and box delta (The second time regression) for each proposal (bounding boxes) is. The architecture is simple, apply average pooling to the feature maps for each proposal in (2.1.3), then apply two fully-connected layer. The loss function of this layer consists of two parts:

- Loss_cls: Apply softmax to the class score and compute the cross entropy loss with ground-truth label.
- Loss_box_reg: It is similar to what RPN regression loss (2.1.2) does. There are some slightly difference. First, only foreground object will be contribute to this loss. Second, RPN's regression is optimize the delta between "delta(Predict, Anchor)" and "delta(GroundTruth, Anchor)". Here, we don't need anchor anymore, we only optimize the delta between Predict and Anchor.

So, if the predict bounding box in RPN is (x, y, w, h) and we get the delta $t = (t_x, t_y, t_w, t_h)$ from the output, then the final bounding box should be

$$x \leftarrow x + t_x * w, \qquad y \leftarrow y + t_y * w$$
$$w \leftarrow w * \exp(t_w), \qquad h \leftarrow h * \exp(t_h)$$

2.1.5 Mask head

The goal of mask head is perform semantic segmentation for each proposals. The architecture is shown in Figure 8. We perform segmentation in 28*28 resolution. The loss of this layer is computed as follows:

- Mask_loss: apply sigmoid function of the mask prediction, and apply binary cross entropy with the ground truth. The ground truth is obtained by applying RoI align to the original ground truth, in order to matching the size (28*28).

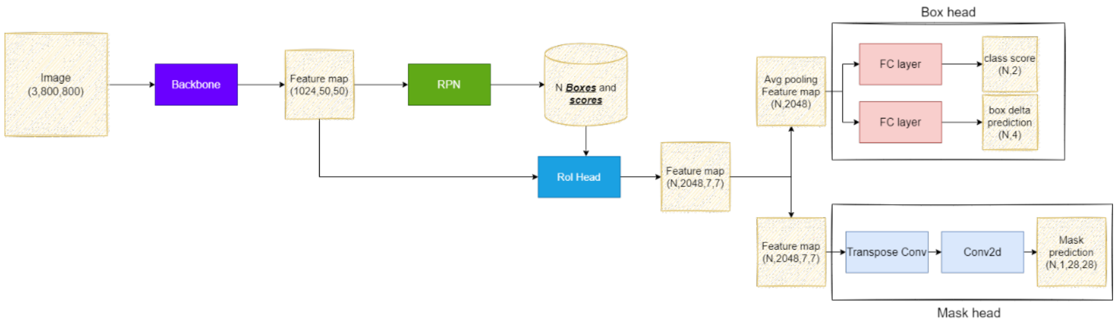


Figure 8: Box head and Mask head

2.2 Data Processing and data augmentation

All images size in the training dataset are 1000*1000. In order to avoid out of memory(OOM) issue, we did the following:

1. Random crop the image to size (500, 500)
2. Resize to one of the following size: (608, 640, 672, 704, 736, 768, 800)
3. Random flip horizontally with probability 0.5.
4. Subtract by the mean (103.530, 116.280, 123.675) (In BGR order, we didn't divide anything since detectron2 pretrain weighted required.)

2.3 Settings

Optimizer: SGD, with momentum=0.9 and weight decay=0.0001.

Iterations: 2400 iterations.

Learning rate: Warmup for first 72 iterations (linear increasing from 0 to 0.01).
Divide 10 at the 480, 1200, 1920 and 2160 iterations.

2.4 Loss function

The loss function composed by 5 parts. Actually we have mentioned all of them in section 2.1

- RPN class loss, RPN regression loss (sec 2.1.2)
- Class loss, box regression loss (sec 2.1.4)
- Mask loss (sec 2.1.5)

Our loss function is adding them directly.

2.5 Inference

All testing images size are (1000, 1000), with one testing image in hand, the inference procedure contains following steps (Default predictor of Detectron2):

1. Subtract by the mean (103.530, 116.280, 123.675) (As we stated in sec2.2)
2. Obtain a feature map from the backbone network.
3. Obtain a list of proposals (bounding box + objectless scores) from RPN. As we stated in sec2.1.2 (RPN), we will apply NMS with threshold 0.7 to the proposals and choose at most 1000 of them with the highest objectless scores.
4. Apply RoI align in order to have the same size of feature map for proposals.
5. Apply Box head to update the proposals (the second time regression of bounding box and predict which class for the bounding box belongs to).
6. Apply NMS with threshold 0.5 and filter out those proposals with scores < 0.05. Then, we only keep most 500 proposals with the highest scores.
7. With these proposals (at most 500) in hand, run Box head and Mask head to obtain the final predictions (Bounding box + Class id + Mask).

3. Summary

In this homework, we try to read the source code of Detectron2 and understand a part of the implementation detail of mask R-CNN (we didn’t try a backbone model with Feature Pyramid Network at this homework). We only try two different backbones: ResNet50 and ResNet101. The remaining part of the model are the same that we introduced above.

For this two backbones, we run three models for each with the same setting above. The result (on codalab) is as follows:

	First time	Second time	Third time
ResNet50	0.243622	0.244502	0.243495
ResNet101	0.244704	0.244325	0.244567

It seems like there is a slightly improvement if we use a larger backbone here.

4. Reference

[1] Mask RCNN paper: [1703.06870] Mask R-CNN (arxiv.org)

[2] Faster RCNN paper: [1506.01497] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (arxiv.org)

[3] Detectron2: facebookresearch/detectron2: Detectron2 is a platform for object detection, segmentation and other visual recognition tasks. (github.com)

[4] Detectron2 (for windows): DGMaxime/detectron2-windows: Detectron2 is FAIR's next-generation platform for object detection and segmentation. (github.com)