HomeWork 3

Mask R-CNN Model

Instance Segmentation

國立陽明交通大學

NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Prepared by

110550124 林家甫

1 INTRODUCTION

The core objective of this homework is to use the Mask R-CNN machine learning model to identify four different types of cells in the given images. The model not only detects which category each object belongs to but also localizes where each object appears in the image through pixel-level segmentation.

The dataset consists of 209 training images and 101 test images, all in **.tif** format. Each training image is accompanied by up to four corresponding mask images. In each mask image, multiple objects are labeled with unique integer values starting from 1, while the background pixels are labeled as 0.

To tackle this task, I used PyTorch's built-in **maskrcnn_resnet50_fpn** model. I also experimented with **maskrcnn_resnet50_fpn** and **maskrcnn_resnet50_fpn_v2**, and I will present the final results and performance comparisons in the Additional Experiments section.

Code Reliability: pep8
GitHub Link: https://github.com/chiafu2018/Instance-Segmentation

2 METHOD

I used the **maskrcnn_resnet50_fpn** model to implement this task. The key difference between Mask R-CNN and Faster R-CNN is that Mask R-CNN adds an additional branch for semantic segmentation, enabling instance-level pixel-wise prediction. Although this model is more complex than the one used in HW2 for object detection, the training time is actually shorter. This is mainly because the dataset used in this task is much smaller than the one in HW2. With the same GPU resources, each epoch takes approximately one minute to complete.

- **Data Preprocessing**

For the input to the Mask R-CNN model, each input image must correspond to a single target dictionary containing six components. Among these, four are required: boxes, labels, masks, and image_id. The remaining two components, area and iscrowd, are optional but often recommended for more precise training behavior.

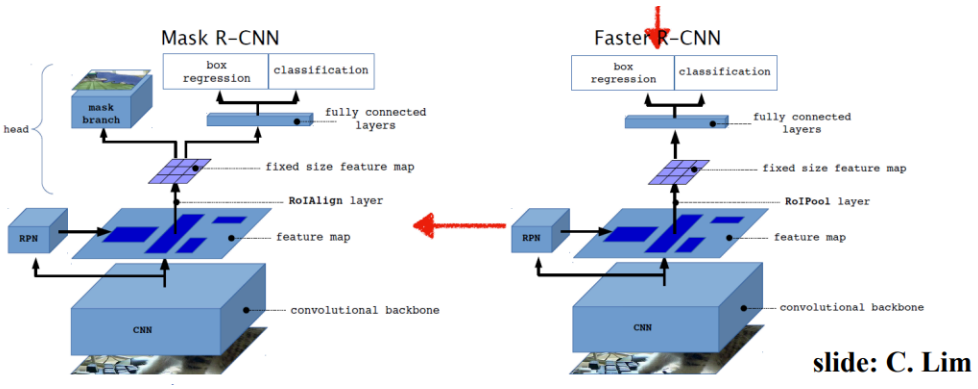
If an image contains multiple objects—as is the case in our training set—then the labels, boxes, and masks should each be a tensor list where each element corresponds to one object. Specifically, each label should correspond to one bounding box and one binary mask, ensuring that the annotations are properly aligned for each detected instance.

```
target = { "boxes": torch.as_tensor(boxes, dtype=torch.float32),
           "labels": torch.as_tensor(labels, dtype=torch.int64),
           "masks": torch.as_tensor(np.stack(binary_masks, axis=0),
dtype=torch.uint8),
           "image_id": torch.tensor([idx]),
           "area": torch.as_tensor(areas, dtype=torch.float64),
            "iscrowd": torch.zeros(0, dtype=torch.int64)
         }
```

- **Model Architecture**

The only difference between Mask R-CNN and Faster R-CNN is the additional segmentation head. Faster R-CNN, as I have introduced in hw2, composed of three main components: the backbone, the neck (Region Proposal Network or RPN), and the head. The **backbone** is typically a convolutional neural network such as ResNet, which extracts feature maps from the input image. These feature maps are then passed to the **RPN**, which acts as the **neck** of the architecture. The RPN slides over the feature map and generates a set of region proposals by predicting objectness scores and bounding box coordinates for multiple anchors at each spatial location. These proposals are then refined and filtered based on their confidence scores. Finally, the **classification head** component takes these proposed regions and applies ROI Pooling (or ROI Align) to extract fixed-size feature representations, which are passed through fully connected layers to perform object classification and further bounding box regression.

In Mask R-CNN, an additional **segmentation head** is introduced to perform instance segmentation. This segmentation head is a **fully convolutional network (FCN)** that takes the fixed-size features produced by ROI Align as input and predicts a **binary mask** for each object instance. The typical output size of each mask is 28×28 pixels. This architecture allows Mask R-CNN to simultaneously perform both **object detection** and **pixel-wise instance segmentation** in a single forward pass. Picture 1 indicates the differences between Faster R-CNN and Mask R-CNN.



Picture 1. Mask R-CNN and Faster R-CNN model architecture

As mentioned earlier, I used the **maskrcnn_resnet50_fpn model** with pretrained weights as the base for training. The model's backbone is ResNet-50, which extracts hierarchical features from the input image. These features are then passed to a Feature Pyramid Network (FPN), which enhances them at multiple spatial resolutions to improve the detection of objects at different scales. A Region Proposal Network (RPN) then uses these multi-scale features to suggest potential object regions.

The entire structure is provided by the PyTorch library, and the model contains approximately 44 million parameters.

```
model = torchvision.models.detection.maskrcnn_resnet50_fpn(weights
=MaskRCNN_ResNet50_FPN_Weights.DEFAULT)
```

- **Hyperparameters**

Using a smaller learning rate not only improves the final performance but can also stabilize and accelerate convergence in some cases. In this experiment, the batch size is set to 1 or 2 to avoid out-of-memory (OOM) errors. Since model parameters are updated after each batch, a smaller learning rate (1e-4) helps prevent overshooting optimal values during training. Additionally, I employ the **ReduceLROnPlateau** scheduler to automatically decrease the learning rate when the validation loss plateaus, enabling more controlled and adaptive fine-tuning.

- **Output Format**

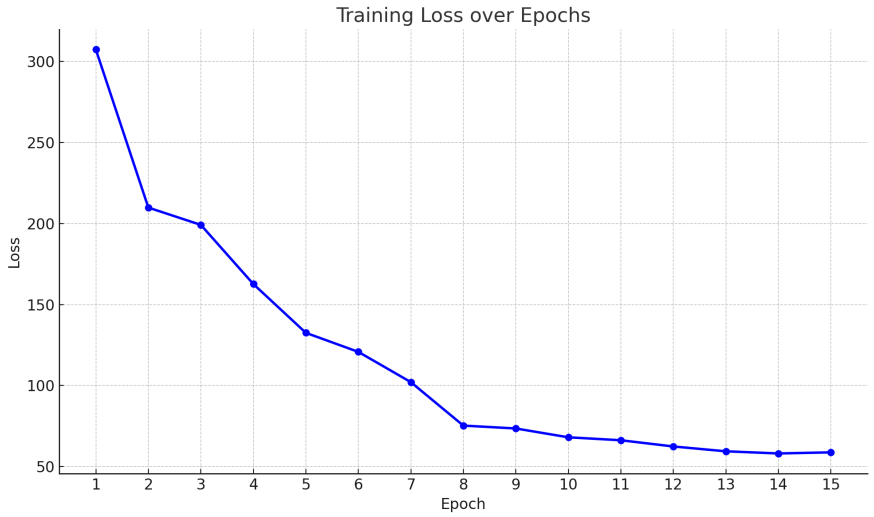
The output in COCO format should include the following fields: image_id, bbox, score, category_id, and segmentation. For the segmentation field, Run Length Encoding (RLE) is commonly used. RLE is a lossless data compression technique that represents

consecutive occurrences of the same value with a single count. This method is particularly suitable for instance segmentation tasks, where most of the pixels in an image belong to the background. These background regions result in long sequences of zeros, making RLE an efficient way to compress binary masks.

```
result = {
    "image_id": int(image_id),
    "bbox": bbox,
    "score": float(scores[i]),
    "category_id": int(labels[i]),
    "segmentation": {
        "size": [height, width],
        "counts": rle['counts']
    }
}
```

3 RESULTS

Since the training set is very small, I did not split it into separate training and validation sets. Instead, I relied on the training loss to track and select the best-performing models. However, I observed that even when the training loss was very low, the model's performance on the test set was not satisfactory. This suggests that the model may be overfitting to the training data. In fact, I test all the epochs and find out at epoch 7 will reach the best performance on the public test, which mAP is 0.4075 on Codabench.



4 REFERENCES

[1] Kaiming et al. “Mask R-CNN” [Online] Available: https://arxiv.org/pdf/1703.06870

5 ADDITIONAL EXPERIMENTS

- Different Mask R-CNN version Models

The table below compares the performance of two built-in Mask R-CNN models. As shown, Mask R-CNN version 1 with a batch size of 1 outperforms version 2 in terms of mean Average Precision (mAP). This result highlights the importance of batch size in this task, especially when observing the bad performance of the version 2 model.

Model / Batch Size	Public	Private
Version 1 / 1	0.3460909	0.3651972
Version 1 / 2	0.3019649	0.356298
Version 2 / 1	0.076132	0.121394
Version 2 / 2	0.3078241	0.316041

After selecting the optimal model and hyperparameters, I conducted an additional experiment by normalizing all input images. The results, presented below, indicate that normalization does not significantly improve performance for this task.

Model / Normalization	Public	Private
Version 1 w/o	0.3460909	0.3651972
Version 1 w/	0.32431	0.360542