Visual Recognition Using Deep Learning 2025 HW3: Instance Segmentation

313551073 顏琦恩

https://github.com/miayan0110/NYCU-Visual-Recognitionusing-Deep-Learning-2025-Spring/tree/master/hw3

I. Introduction

Instance segmentation is a computer vision task that involves simultaneously detecting and precisely delineating each object instance within an image. This task presents significant challenges due to object occlusions, variations in scale, diverse poses, and complex backgrounds. The assignment involves several key steps, starting with data preprocessing, which includes loading the image dataset and applying necessary transformations to images. Next, the model is designed using Mask R-CNN as the base model. Finally, I implement the training, validation, and evaluation processes to enable the model to accurately segment each object instance in real-world images.

To train the model, I use the dataset provided by the TAs. During training, I validate the model at every epoch to monitor the model's performance. Further implementation details will be discussed in the next section.

II. Method

I implemented the instance segmentation model using the Mask R-CNN architecture with a ResNet-50 backbone, a Feature Pyramid Network (FPN) neck, and heads consisting of the Region Proposal Network (RPN) and Region of Interest (RoI) heads, realized via the maskrcnn_resnet50_fpn_v2 model from torchvision. In this homework, I did not apply any additional preprocessing to the images apart from basic normalization.

The backbone ResNet-50 is pretrained on ImageNet. It is composed of an initial convolution and pooling layer followed by four residual blocks. In my setting, the last three stages of the backbone are trainable by default, while the earlier layers are frozen to retain general low-level feature extraction. This setup allows the model to adapt high-level features to the instance segmentation task while preserving efficient convergence.

The Feature Pyramid Network (FPN) acts as the neck of the model. It enhances multi-scale feature representation by constructing a top-down pyramid with lateral connections, allowing the model to detect and segment objects of various sizes more effectively.

To adapt the model for this specific instance segmentation task, I replaced the original classification and mask heads. The box head is replaced with a new FastRCNNPredictor, and the mask head is replaced with a MaskRCNNPredictor, both modified to output the number of target classes as required. The model is trained for 100 epochs with mixed precision enabled through PyTorch's autocast and GradScaler for efficiency. A batch size of 2 is used during training. The optimization is performed using Stochastic Gradient Descent (SGD) with a learning rate of 0.005, momentum of 0.9, and weight decay of 5e-4. A MultiStepLR scheduler is applied to decay the learning rate at 80% and 90% of the total number of epochs.

```
def get_maskrcnn_model_v2(args, num_classes):
    model = 
maskrcnn_resnet50_fpn_v2(weights=MaskRCNN_ResNet50_FPN_V2_Weights.DEFAULT)
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
    in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
    model.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask, 256, 
num_classes)
    model.to(args.device)

    optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=0.9, 
weight_decay=0.0005)

    start_epoch = 0
    if args.resume_path:
        print(f"Loaded model weights from {args.resume_path}")
        ckpt = torch.load(args.resume_path, map_location=args.device)
        model.load_state_dict(ckpt['model_state_dict'])
        optimizer.load_state_dict(ckpt['optimizer_state_dict'])

        start_epoch = ckpt.get('epoch', 0)
        print(f"Resuming from epoch {start_epoch}")
    print(f'model parameters: {sum(p.numel() for p in model.parameters() if 
p.requires_grad)}')
    return model, optimizer, start_epoch
```

III. Results

During training, I saved a checkpoint at the end of each epoch and kept track of the training loss to generate a loss curve (Fig. 1). I also computed the mean Average Precision (mAP) at each epoch to evaluate the model's detection performance (Fig. 2). As shown in Fig. 1, the loss steadily decreases over 100 epochs, indicating that the model is learning effectively. Although there are fluctuations in Fig. 2, the overall mAP value stabilizes around 0.48–0.50 after approximately 20 epochs. The initial instability is expected as the model adjusts from random initialization toward meaningful segmentation outputs.

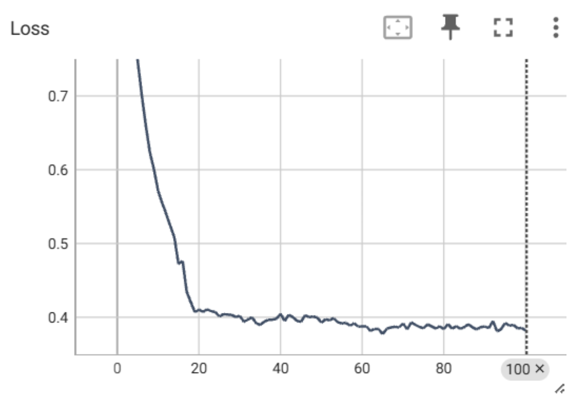


Figure 1. Loss curve.

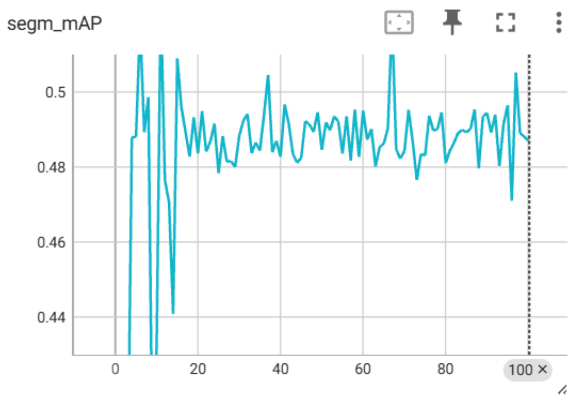


Figure 2. mAP curve.

IV. References

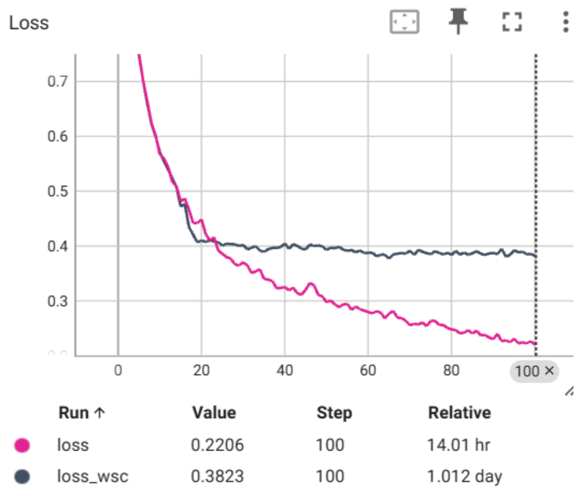
1. Mean Average Precision (mAP)
    - https://lightning.ai/docs/torchmetrics/stable/detection/mean_average_precision.html
    - https://stackoverflow.com/questions/78359307/how-to-use-meanaverageprecision-metric-from-torchmitrics-detection-on-an-object
2. CocoAPI
    - https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/cocoeval.py

V. Additional Experiments

To further investigate the impact of learning rate scheduling, I conducted an additional experiment by training the model with and without applying a learning rate scheduler. In the "wsc" (with scheduler) setting, a MultiStepLR scheduler was used to decay the learning rate at 80% and 90% of the total epochs. In contrast, another setting is trained without any scheduler, using a fixed learning rate throughout.

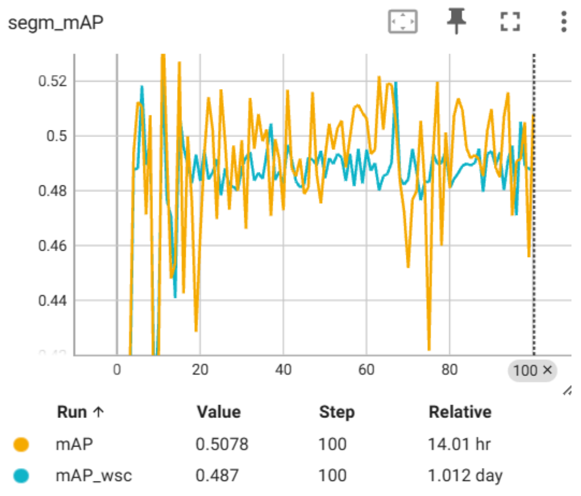
As shown in Figure 1, removing the scheduler resulted in a lower final training loss (0.2206) compared to training with a scheduler (0.3823). However, as shown in Figure 2, the mAP of the model without a scheduler (0.5078) is slightly higher than the mAP of the model with a scheduler (0.487).

These results suggest that while removing the scheduler leads to better convergence in terms of training loss, it only slightly improves segmentation performance. The scheduler, although leading to a higher training loss, may still provide better training stability and prevent potential overfitting in longer training runs.



	Run ↑	Value	Step	Relative
●	loss	0.2206	100	14.01 hr
●	loss_wsc	0.3823	100	1.012 day

Figure 3. Loss curve of experiment.



	Run ↑	Value	Step	Relative
●	mAP	0.5078	100	14.01 hr
●	mAP_wsc	0.487	100	1.012 day

Figure 4. mAP curve of experiment.