DL-HW3 Report: Mask R-CNN Based Instance Segmentation

My Git Link: https://github.com/IaTsai/DL-HW3

1. Introduction

In this project, I address the task of instance segmentation for biological cell images using Mask R-CNN. I aim to identify and delineate individual object instances belonging to four categories. The main challenge lies in detecting fine-grained and overlapping instances with limited training data and memory constraints. We focus on improving the baseline architecture by integrating different backbones and attention modules, optimizing training procedures, and debugging COCO evaluation mismatches.

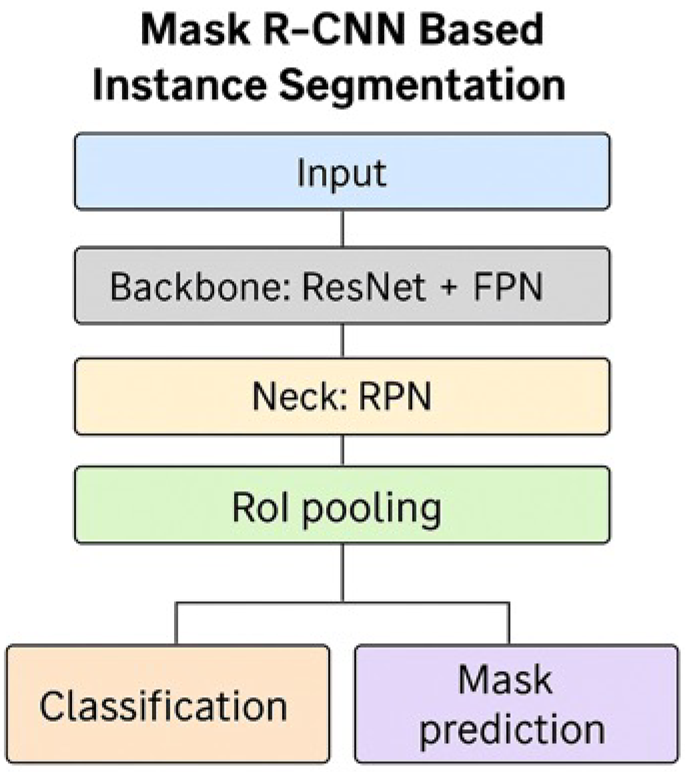
---

2. Method

2.1 Data Preprocessing

- The dataset follows COCO format with annotations in `train_gt.json`.
- A custom dataset class is used to parse instance masks from class-wise .tif masks.
- The training and validation sets are split 90%:10% with a consistent random seed.
- `val_gt.json` is auto-generated to align with the COCOEval API.

2.2 Model Architecture



I adopt the Mask R-CNN framework with a customizable backbone and FPN neck:

- **Backbone**: Tested with the following options:
  - resnet50
  - resnet50 (+ SE / CBAM)
  - resnet50v2 (+ SE / CBAM)
  - resnext50_32x4d (+ SE / CBAM)
  - resnet101 (too large for memory)
  - efficientnet_b0, efficientnet_b1 (via timm with FPN integration)
- **Neck**: Feature Pyramid Network (FPN), channels = [256, 512, 1024, 2048] → 256
- **Head**:
  - Region Proposal Network (RPN)
  - FastRCNN Head for box prediction
  - Mask Head for pixel-wise segmentation

2.3 Training Configuration

- **Losses**: Standard Mask R-CNN loss (classifier, box reg, mask, RPN losses)
- **Optimizer**: AdamW
- **Initial LR**: 1e-3
- **Learning Rate Warmup**: First 3 epochs use 10% LR
- **LR Decay**: ReduceLRonPlateau (factor=0.2, patience=30)
- **Early Stopping**: Stop after 3 decays with no improvement
- **AMP**: Automatic Mixed Precision via torch.cuda.amp

2.4 Evaluation

- COCOEval used for mAP calculation.
- Score threshold: 0.05 (post-filtering)
- Detected bounding boxes are checked for finite values to avoid core dump.

---

3. Results

Backbone Variant	Attention	Scheduler + Stop	mAP
ResNet50	None	Yes	0.3403
ResNet50	SE	Yes	0.3405
ResNet50	CBAM	Yes	**0.3617** *(Best)*
ResNet50V2	None	Yes	0.348
ResNet50V2	SE	Yes	0.3486
ResNet50V2	CBAM	Yes	0.3288
ResNeXt50-32x4d	CBAM	No	0.3413
EfficientNet-B0	None	No	0.2682
ResNet101	-	Not used	OOM

Training curves and memory usage logs indicate that attention modules introduce overhead. However, with LR warmup and AMP, training remains stable below 18 GB VRAM.

---

4. Additional Experiments & Debugging

4.1 Architecture Modifications

- Integrated `SELayer` and `CBAM` into backbone's bottleneck blocks
- EfficientNet-FPN via `timm` and `FeaturePyramidNetwork`

4.2 COCO Eval Issues Debugging

- Segmentation faults were caused by invalid `bbox` values (e.g., NaN or Inf). A filtering step np.isfinite(d['bbox']).all() fixed the issue.
- Val dataset JSON mismatch debugged by explicitly generating `val_gt.json` subset

4.3 AMP Debugging

- Memory was reduced from >20GB to ~14GB per epoch with AMP
- Added `try-except` to skip AMP-triggered exceptions (e.g., non-finite losses)

---

5. Reference

- [1] K. He, G. Gkioxari, P. Dollár, and R. Girshick, Mask R-CNN, in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2961–2969.
- [2] R. Wightman, PyTorch Image Models, GitHub Repository. [Online]. Available: https://github.com/huggingface/pytorch-image-models
- [3] S. Woo, J. Park, J. Lee, and I. S. Kweon, CBAM: Convolutional Block Attention Module, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 3–19.
- [4] J. Hu, L. Shen, and G. Sun, Squeeze-and-Excitation Networks, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 7132–7141.
- [5] TorchVision Detection Models, PyTorch Documentation. [Online]. Available: https://pytorch.org/vision/stable/models.html
- [6] COCO API, GitHub Repository. [Online]. Available: https://github.com/cocodataset/cocoapi

---

6. Architecture Choice Rationale

I selected **ResNet50 with CBAM + FPN + WarmupLR** as our final model based on the following:

- **Pros**:
  - ResNet50 strikes a balance between accuracy and VRAM usage.
  - CBAM introduces lightweight attention, boosting contextual awareness.
  - Warmup avoids early instability, and ReduceLROnPlateau ensures steady convergence.
- **Cons**:
  - Slight increase in computation and memory (~1-2GB with CBAM).
  - Model tuning becomes more sensitive with attention.

I also explored EfficientNet and ResNeXt variants, but ResNet50-based versions proved most effective under our constraints.