Selected Topics in Visual Recognition using Deep Learning, Homework 3

111550061, 邱冠崴

GitHub: HW3_LINK

Part. 1, Introduction

In this assignment, the objective is to perform cell instance segmentation using Mask R-CNN. After training, predictions must be converted into RLE (Run-Length Encoding) format for submission to Codabench, where model performance is evaluated based on mean Average Precision (mAP).

I selected Mask R-CNN with a ResNet-50 v2 backbone as the base model. Due to the severe class imbalance in the dataset, I used a WeightedRandomSampler during training to rebalance the data distribution, which resulted in an initial mAP of 0.3519 on the public testing set. Since the target cells are relatively small, I also adjusted the anchor sizes to better match the object scale. This modification further improved the mAP to 0.3741.

To boost performance even more, I conducted an additional experiment using an ensemble method, which significantly increased the final mAP to 0.4555.

Part. 2, Method

(1) Data Pre-processed:

The training dataset shows a significant class imbalance, with 14,537 instances in category 1, 15,653 in category 2, but only 630 and 587 instances in categories 3 and 4, respectively. This imbalance may cause the model to overfit to the dominant classes (1 and 2) while underperforming on the minority classes (3 and 4).

To address this issue, I first computed the number of instances in each category and applied log-balanced class weighting to reduce the impact of extreme class frequencies. Then, for each training image, I calculated the mean class weight based on the categories present in that image. This yielded a per-image weight list, which I used to initialize a WeightedRandomSampler function in torch.utils.data library.

(2) Model architecture and hyperparameters:

Model	maskrcnn_resnet50_fpn_v2 (in torchvision)
Optimizer	Adamw
Learning rate	$4*10^{-4}$
Scheduler	StepLR (LR *0.1 every 3 epochs)
# of parameters	45,671,213

Model architecture (maskrcnn_resnet50_fpn_v2):

I chose this architecture because it provides pretrained weights specifically for instance segmentation, available directly in the torchvision library. Although deeper and more advanced backbones like ResNet-101 or ResNeXt-50 exist, their pretrained weights are mainly designed for image classification, not for instance segmentation tasks. I believe that using weights pretrained on instance segmentation helps the model perform better than using classification-based weights.

The main advantage of this model is that its deep structure allows it to capture complex and high-level features, which is important for achieving accurate segmentation results. However, the main drawback is its high computational cost—it requires a large amount of GPU memory and takes more time to train and run inference.

(a) backbone:

The backbone of this model is ResNet-50, a 50-layer deep convolutional neural network that uses residual connections to enable efficient training of deep networks. It is responsible for extracting high-level semantic features from input images. In this implementation, ResNet-50 is modified to use FrozenBatchNorm2d instead of the standard BatchNorm2d. This means the batch normalization layers use precomputed statistics and do not update during training, which helps improve stability and performance, especially when training with small datasets or small batch sizes.

(b) neck (FPN):

The neck component is the Feature Pyramid Network (FPN) v2, which improves the backbone by generating a pyramid of feature maps at multiple scales. FPN combines low-level, high-resolution features with high-level, low-resolution features using a top-down pathway and lateral connections, allowing the model to better detect objects of different sizes. In the v2 implementation provided by torchvision, improvements include more consistent use of FrozenBatchNorm2d for stability, better integration with the backbone, and slight efficiency optimizations in how feature maps are processed. These changes make the model more stable and effective, especially for segmentation task.

(c) head of Mask R-CNN:

The head in this architecture includes two main components: the Region Proposal Network (RPN) and the Region of Interest (RoI) Heads. The RPN takes the multi-scale feature maps from the FPN and generates candidate object regions.

The RoI heads consist of fully connected layers for classification (predicting the object category) and bounding box regression (refining the position of the box). In addition, there is a mask prediction branch that generates a binary segmentation mask for each detected instance.

In my implementation, I adjusted the number of output classes to 5, which includes 4 cell types (class1 to class4) and 1 background class, in accordance with the task requirements.

Moreover, because the cells in the images are relatively small, I adjusted the anchor sizes to better capture fine-grained details and improve detection accuracy.

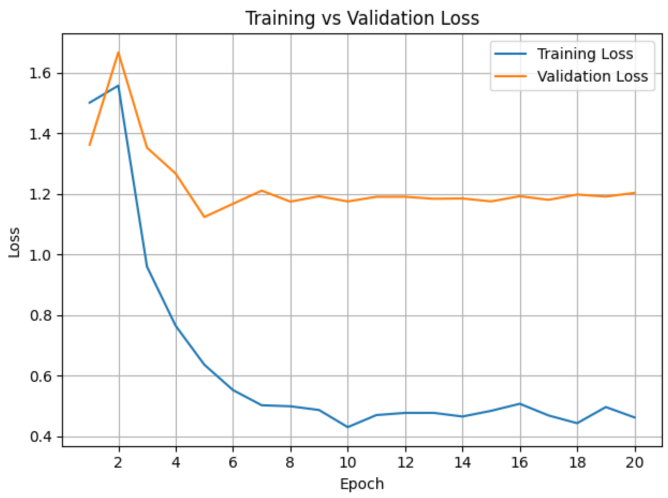
Train strategy:

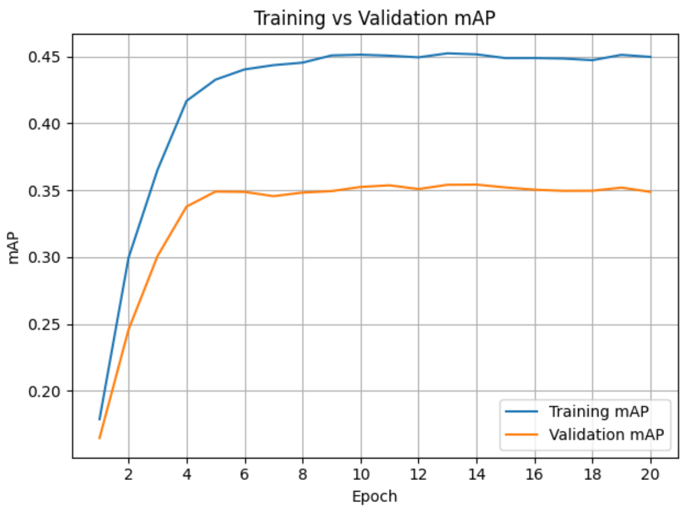
I noticed that the dataset is highly imbalanced across categories. To address this, I used the WeightedRandomSampler to assign different sampling weights to each image based on the class distribution, ensuring that underrepresented classes are sampled more frequently during training.

In addition, the default anchor sizes used in Mask R-CNN are [32, 64, 128, 256, 512]. However, since the target cells are relatively small, I reduced the anchor sizes to [8, 16, 32, 64, 128] to better match the scale of the objects and improve detection accuracy.

Part. 3, Results

(1) Training curve

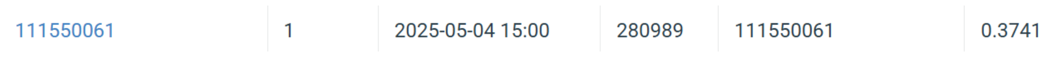




First, we can see the graph on the left, which shows the training loss and validation loss over 20 epochs. The blue curve represents the training loss, which shows a steady decrease in the first 10 epochs. After epoch 10, the training loss stops declining and starts to plateau, hovering around 0.4 to 0.5. This suggests that the model has reached a point where it is possibly approaching convergence. The orange curve represents the validation loss. Unlike the training loss, the validation loss decreases rapidly in the first few epochs but then plateaus around epoch 6. From epoch 6 to 20, the validation loss remains relatively flat and does not improve further. This could indicate that the model is beginning to overfit.

Next, take a look at the graph on the right, which shows the training mAP and validation mAP across 20 epochs. The blue curve represents the training mAP, while the orange curve represents the validation mAP.
We can observe that the training mAP increases rapidly during the first 5 epochs and starts to plateau around epoch 8, reaching a peak of 0.451. This indicates that the model is able to improve its performance on the training data relatively quickly. The validation mAP also improves steadily in the initial epochs, rising from 0.165 to 0.349 by epoch 5. However, after that, it shows little to improvement and plateaus for the remainder of the training process. This suggests that while the model learns to perform better on the training set, its generalization ability on unseen data has not significantly improved after the 5 epochs.

Based on the insights from the two plots, I eventually chose the model weight obtained after fine-tuning for 9 epochs. This checkpoint achieved a mAP of 0.3741.



111550061	1	2025-05-04 15:00	280989	111550061	0.3741

Part. 4, References

[1] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778.

[2] T. -Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature Pyramid Networks for Object Detection," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 936-944.

[3] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 2980-2988.

[4] Y. Li, S. Xie, X. Chen, P. Dollár, K. He, and R. Girshick, "Benchmarking Detection Transfer Learning with Vision Transformers," arXiv:2111.11429.

Part. 5, Additional experiments

After multiple rounds of parameter tuning, I observed that the mAP performance of a single model plateaued, suggesting that it may have been learning only a subset of features while failing to capture others. Based on this observation, I hypothesized that combining multiple Mask R-CNN models—each with different backbones or configurations—could improve instance segmentation performance. To explore this, I experimented with models that varied in settings such as pre_nms_top_n_train, post_nms_top_n_train, NMS thresholds, batch sizes, and partial backbone frozen. My intuition was that different model configurations might capture diverse aspects of the data, enabling the ensemble to extract complementary features and finer details that a single model might overlook.

In my ensemble experiment, I combined predictions from 17 different models, each configured with distinct backbones or hyperparameters. These 17 models were selected based on their relatively higher mAP scores on the public testing dataset. To ensemble their outputs, I utilized the nms (non-maximum suppression) function from the torchvision.ops library to merge the resulting .json prediction files. The NMS function works by first sorting all predicted bounding boxes in descending order of their confidence scores. It then iteratively selects the highest-scoring box and adds it to a keep list. For the remaining boxes, it calculates the Intersection-over-Union (IoU) with the selected box. If the IoU exceeds a threshold of 0.5 (as set in my experiment), the overlapping box is discarded. This process repeats until all boxes are either retained or suppressed. Finally, the indices in the keep list are used to reconstruct the filtered ensemble prediction set, which is expected to reduce redundancy while preserving high-confidence, non-overlapping detections.

As a result, the mAP on the public testing dataset improved significantly—from 0.3741 to 0.4555—after applying the ensemble method. This substantial gain in performance supports my hypothesis that combining multiple models with diverse architectures or configurations can lead to a more comprehensive understanding of the data.

Mask R-CNN with ResNet50-FPN v2:

111550061	1	2025-05-04 15:00	280989	111550061	0.3741

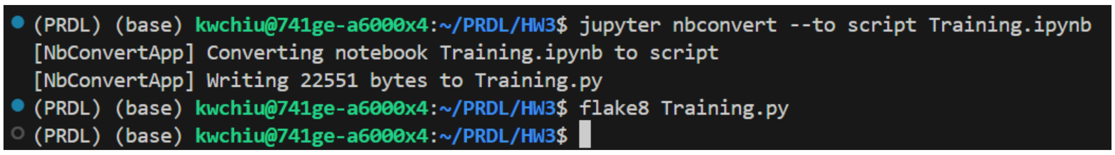
nms function by using 17 different models distinct backbones or hyperparameters:

111550061	1	2025-05-06 19:08	282476	111550061	0.4555

Part. 6, Code Reliability

Lint the code:

Since I am using the .ipynb file, which does not support Flake8 directly, I first convert it to a .py file before running Flake8 for code checks.



```
(PRDL) (base) kwchiu@741ge-a6000x4:~/PRDL/HW3$ jupyter nbconvert --to script Training.ipynb
[NbConvertApp] Converting notebook Training.ipynb to script
[NbConvertApp] Writing 22551 bytes to Training.py
(PRDL) (base) kwchiu@741ge-a6000x4:~/PRDL/HW3$ flake8 Training.py
(PRDL) (base) kwchiu@741ge-a6000x4:~/PRDL/HW3$
```