313551139 Gary Chen

Github: https://github.com/Gary123fff/NYCU-Computer-Vision-2025-Spring-HW3.git

Introduction

In this report, I talk about my approach to tackling the instance segmentation task for detecting and classifying four types of cells in medical images. Instance segmentation is a more advanced computer vision task—it involves not just finding objects in an image, but also outlining them precisely at the pixel level. When it comes to medical image analysis, accurate cell segmentation plays a key role in diagnosis, treatment planning, and research.

I chose to use a Mask R-CNN architecture and made several customizations to improve its performance, specifically for identifying and segmenting four kinds of cells in microscopic images. I focused on enhancing the model's ability to extract features and predict masks, as medical cell images come with their own set of challenges—like overlapping cells, differences in size, and subtle visual differences between cell types.

Method

Dataset Overview

The dataset consists of 209 colored medical images for training and validation, with an additional 101 images for testing. Each image contains instances of four different cell types (labeled as class1, class2, class3, and class4). The goal is to identify each cell instance and create an accurate segmentation mask for it.

Data Preprocessing

My preprocessing pipeline includes several steps designed to normalize the data and enhance the model's ability to learn from it:

1. **Image Loading**: Images are loaded in RGB format from TIFF files using OpenCV, with color space conversion from BGR to RGB.
2. **Mask Extraction**: For each of the four cell classes, I extract individual instance masks from the corresponding class file, ensuring each instance is properly isolated.
3. **Bounding Box Calculation**: For each instance mask, I compute the bounding box coordinates by finding the minimum and maximum x and y positions of non-zero pixels.
4. **Data Augmentation**: I apply a comprehensive set of transformations to increase the diversity of my training data and improve model generalization:
5. **Image Normalization**: Images are normalized using the standard ImageNet mean and standard deviation values to enhance model generalization.

6. **Size Standardization**: All images are resized to a fixed 400×400 pixel dimension, ensuring consistent input size for the model.
7. **Color Augmentation**: Color jitter is applied to training images, varying brightness, contrast, and saturation to improve robustness to lighting and color variations.
8. **Geometric Transformations**: I use affine transformations with subtle scaling and translation to further augment the dataset while preserving cell shapes.

Model Architecture

This project is about segmenting medical cell images, and I built the entire Mask R-CNN pipeline myself using PyTorch.

Backbone: Swin Transformer with FPN

I chose a Swin Transformer as the backbone because it's really good at capturing both local and global features — which is perfect for medical images where cells can vary a lot in shape and size. On top of that, I used a Feature Pyramid Network (FPN) to help the model detect objects at multiple scales more effectively.

Custom Box and Mask Heads

Instead of using the default heads in Mask R-CNN, I built my own:

- **Box Head**: I replaced the original box predictor with a custom CustomBoxPredictor that has two fully connected layers followed by classification and bounding box regression layers. This gives me more control over how the boxes are predicted.

- **Mask Head**: I also replaced the mask head with a custom CustomMaskHead, which stacks multiple convolution layers and a transposed convolution to upsample the masks. It outputs one mask per class.

This customization helps the model focus more on my specific task and improves performance.

Anchors and RoI Align

To generate proposals, I defined anchors with sizes ranging from 4 to 64 and aspect ratios of 0.5, 1.0, and 2.0. Then I used MultiScaleRoIAlign for both box and mask regions, which ensures that features from different FPN levels are aligned properly before feeding into the heads.

Training Strategy

My training strategy was designed to optimize model performance while preventing overfitting on the limited medical imaging dataset:

Optimizer:

AdamW with weight decay to reduce overfitting. I chose AdamW over standard SGD or Adam because it provides adaptive learning rates while effectively implementing weight decay regularization, which helps prevent overfitting on my relatively small medical dataset.

Learning Rate Schedule:

Cosine annealing scheduler to gradually reduce learning rate. The cosine schedule provides a smooth learning rate decay, reducing oscillations and helping the model converge to better local minima compared to step-based schedules.

Hyperparameters:

- Learning rate: 5e-4 (carefully tuned to balance convergence speed and stability)
- Batch size: 4 (optimized for training stability given GPU memory constraints)
- Epochs: 50 (sufficient for convergence while avoiding overfitting)
- Weight decay: 5e-3 (strong regularization to prevent overfitting)
- Positive fraction for ROI sampling: 0.3 (balanced sampling for RPN training)

Data Split:

80% training, 20% validation using random_split

Model Size Control:

Ensuring the model has fewer than 200M parameters to meet efficiency requirements

References

Without references

Additional experiments

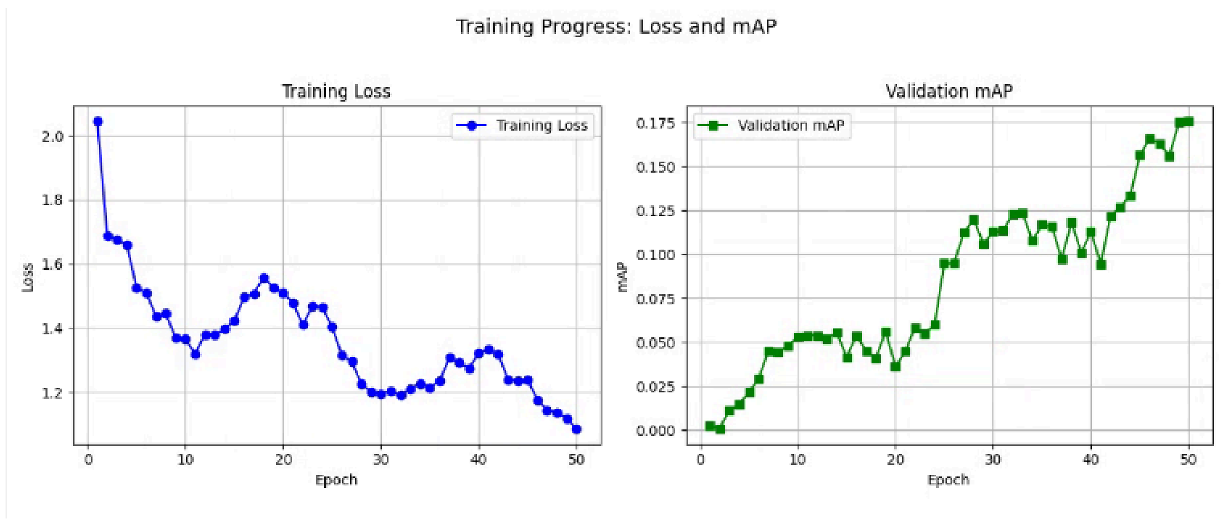
1.Switching Backbone to Swin Transformer

Hypothesis

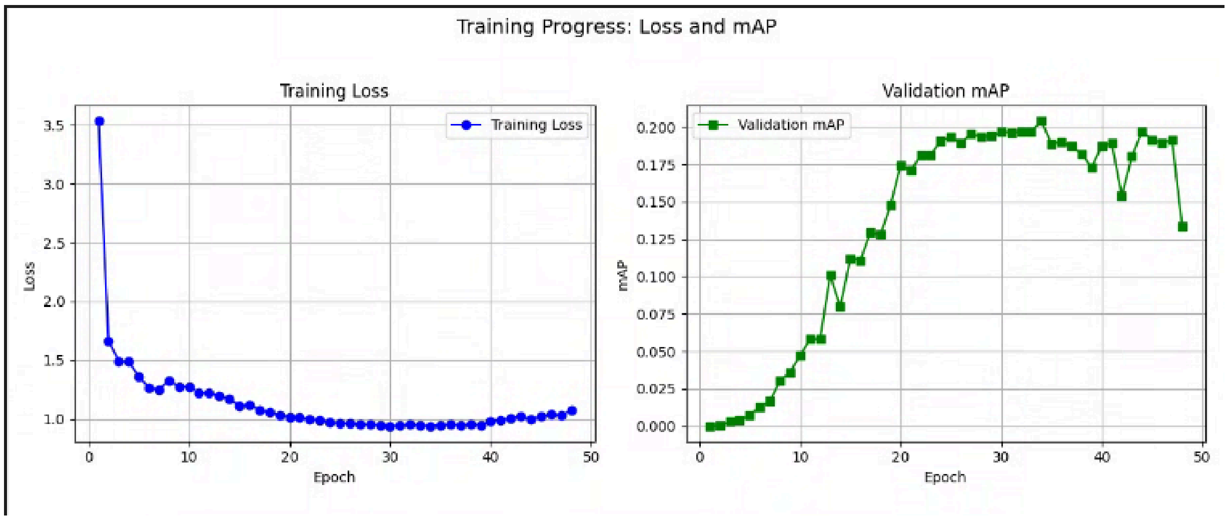
I started out using ResNet-50, but honestly, the performance wasn’t great — the mAP stayed pretty low and didn’t improve much. Since the images I am working with (cells) have lots of overlaps and look very similar, I thought CNNs like ResNet might not be the best fit. Swin Transformer, on the other hand, is designed to capture both local and global information thanks to its attention mechanism. So I figured it might help with the tricky parts of this dataset.

How this may (or may not) work

The idea was that Swin could better understand the overall structure and tiny details in the images, which should help with detection. That said, I wasn’t 100% sure — Swin is a larger model, and I was a bit worried about overfitting or slower training.



Resnet50



SWIN

Experiment Results

Turns out it worked pretty well:

- The training loss dropped more smoothly compared to ResNet.
- Validation mAP improved noticeably — from around **0.175** with ResNet to about **0.26** with Swin.

- Visually, Swin did a better job separating overlapping or unclear cells in the segmentation output.

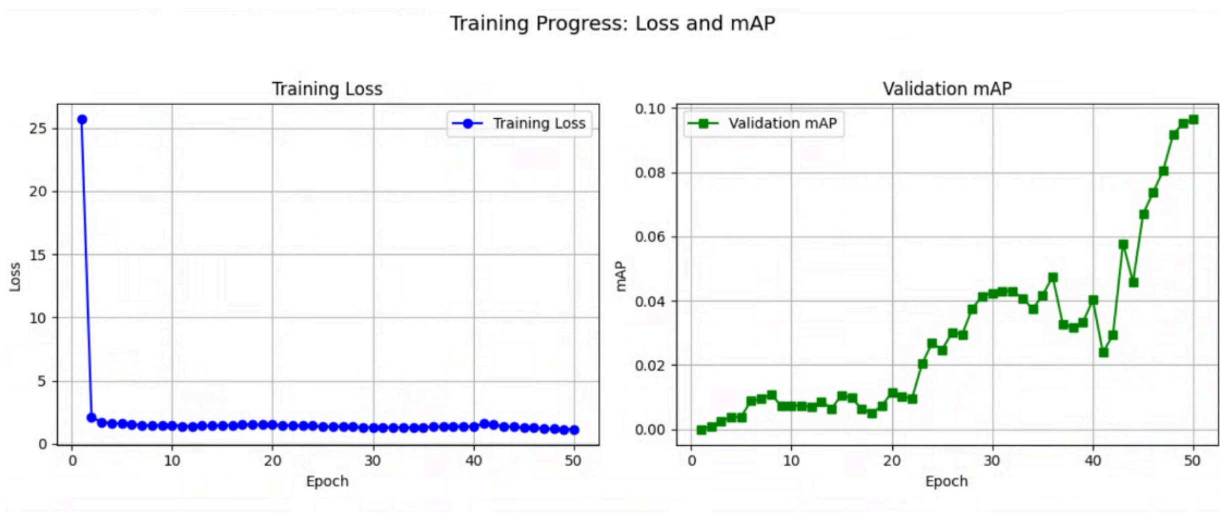
Conclusion:
Switching to Swin Transformer clearly helped. While it takes more resources to train, the boost in accuracy is totally worth it, especially for complex medical images like these. Going forward, it might be interesting to test lighter Swin versions or combine this with better loss functions for even more improvement.

2.Swin Transformer Custom Head

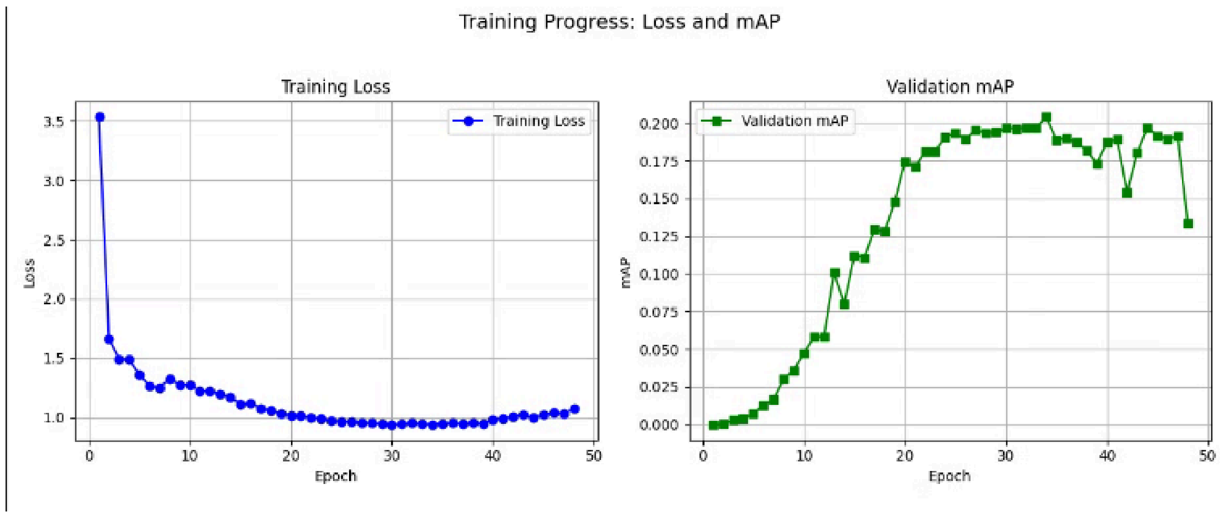
Hypothesis
I thought that by adding a head to the model, it could learn features more effectively and give better results—especially on the validation set. The head is like an extra layer on top of the main feature extractor (the backbone), and it's supposed to help the model focus on the actual task better, like classification or detection.

How this may (or may not) work

Tthe head can make features more meaningful for the task, help with more stable training, and allow the model to learn more complex patterns.
But it might not help if the head is too complicated or not designed well—it could even make the model overfit, especially with limited data. And of course, if the backbone is already strong enough, the extra head might not make a big difference.



SWIN without custom head



SWIN with custom head

Results
Looking at the charts, the model with the head clearly performed better.

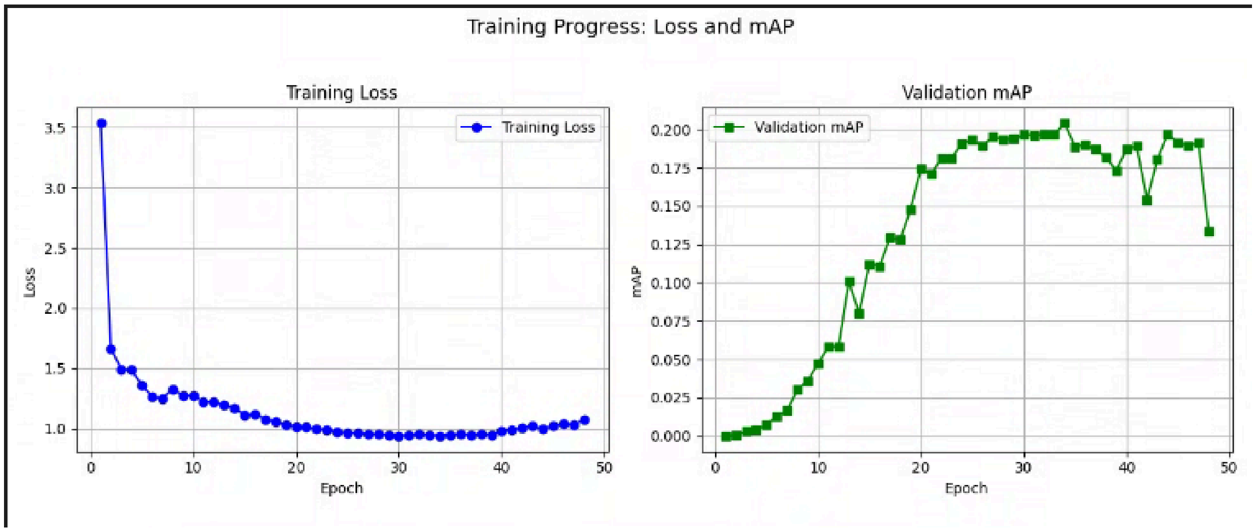
For training loss: without the head, the loss started really high and then stayed kind of flat; with the head, the loss was lower from the start and dropped smoothly.

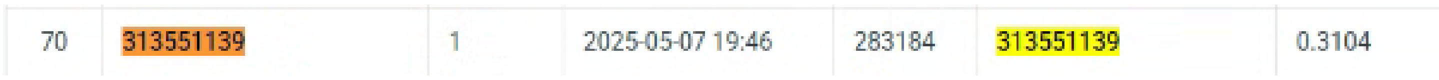
For validation mAP: without the head, the mAP stayed low the whole time—barely 0.1. With the head, the mAP went up quickly and reached around 0.2.

So overall, adding the head made the training more stable and the results more accurate.

Conclusion:
In short, adding a head helped a lot. It made the model train better and gave much better validation accuracy. This shows that a well-designed head can really boost performance. In the future, I could look into trying other head designs to see if I can push it even further.

Results





70	313551139	1	2025-05-07 19:46	283184	313551139	0.3104

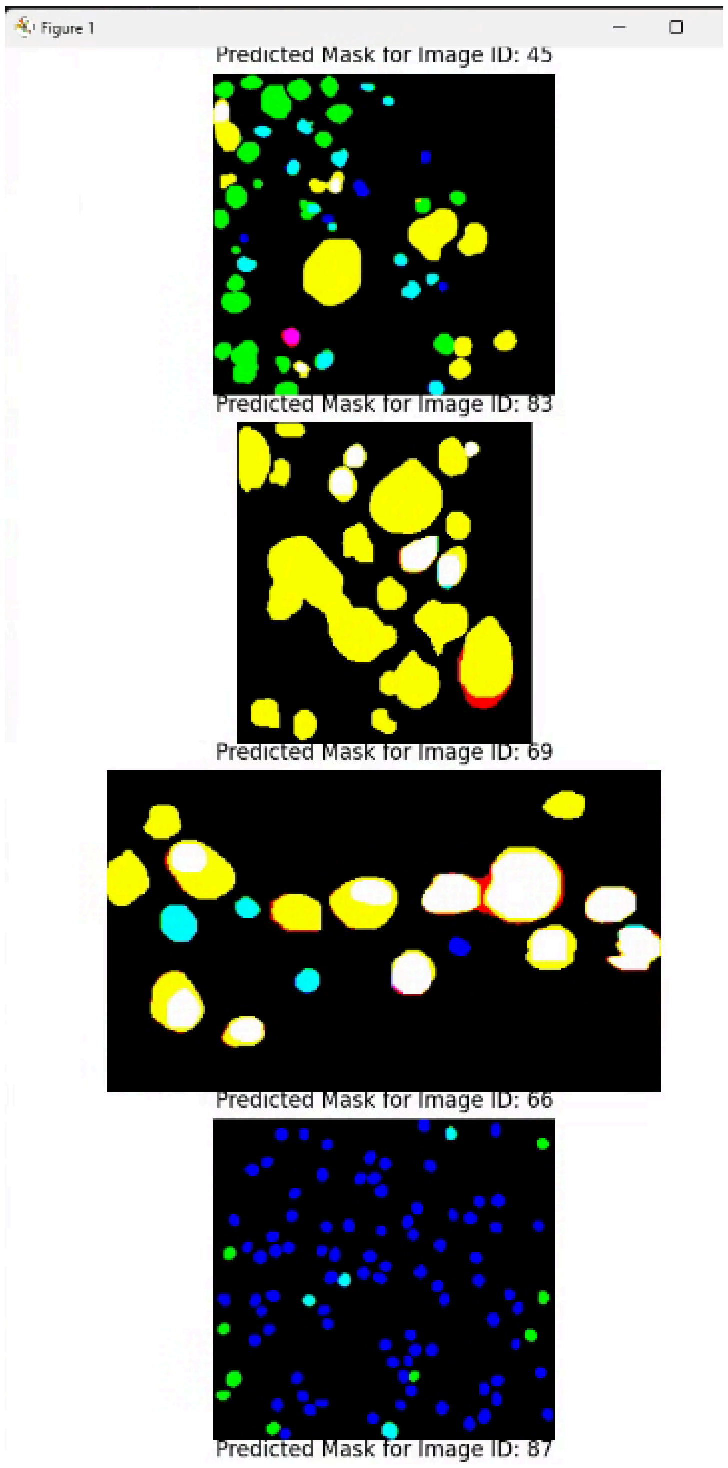


Figure 1
Predicted Mask for Image ID: 45
Predicted Mask for Image ID: 83
Predicted Mask for Image ID: 69
Predicted Mask for Image ID: 66
Predicted Mask for Image ID: 87

I trained a custom Mask R-CNN with a Swin backbone and my own box and mask heads. As you can see from the graph, the training loss dropped quickly at first and then leveled off around 1.0. That means the model learned the basics pretty fast. The validation mAP steadily improved, especially between epochs 10 to 30, and reached around 0.20, which shows the model was getting better at detecting and segmenting. Toward the end, there were some small ups and downs in both loss and mAP, probably due to overfitting or limited data, but overall the training went smoothly and the results were pretty stable. Since I worked on this alone, I also handled the entire model design and training pipeline by myself.