Visual Recognition Using Deep Learning Lab3 多工所碩一 313553024 蘇柏叡

---

0. Github link of this Lab:

https://github.com/rmd926/NYCU_CV2025_Spring/tree/main/Lab3

1. Introduction

In this laboratory assignment, the objective is to perform **instance segmentation** on **310 colour microscopic images** of medical specimens—**209 reserved for training** and **101 for testing**—containing **four distinct cell categories**. Instance segmentation unifies the merits of semantic segmentation and object detection: it requires **pixel-level semantic labelling** of the image while simultaneously **localising and categorising every individual object**. Moreover, for each discrete instance belonging to the same category, the model must produce an **accurate pixel-wise mask** and assign a **unique instance identifier**.

Following the specification, we adopt **Mask R-CNN [1]** as the overall framework and tailor its three principal components—backbone, neck, and heads—to our task. For the backbone we evaluate two pre-trained networks: **ResNet-101 [2]**, **initialised with *ResNet101_Weights.IMAGENET1K_V2***, and **EfficientNet-V2 [3]**, **initialised with *EfficientNet_V2_M_Weights.IMAGENET1K_V1*.** In both cases the first two stages are kept frozen to preserve low-level representations, whereas the deeper layers are unfrozen—**three stages for ResNet-101 and six blocks for EfficientNet-V2-M—to allow training;** the latter configuration serves as our baseline. Departing from the default Mask R-CNN design, we further integrate lightweight attention-enhancement modules into the neck and head: a **Transformer-augmented feature-pyramid neck that promotes global multi-scale interaction**, and a **spatial-channel attention module inserted at the entrance of the mask head** to refine instance-level decoding. These modifications collectively aim to strengthen semantic fusion and boundary precision while retaining the computational efficiency of the original architecture.

Our **proposed attention-enhancement module** introduces two **architectural modules** tailored to improve **semantic representation** and **instance-level discrimination** within the **Mask R-CNN [1] pipeline**. In the **neck**, the **multi-scale feature maps** produced by the **Feature Pyramid Network (FPN)** [10] are first projected into a **shared channel dimension** and compressed into **fixed spatial resolutions** using **adaptive average pooling**. These are then reshaped into **token sequences** and enriched with **positional information** via

sinusoidal encodings and **learnable level embeddings**. The resulting sequence undergoes **global contextual modeling** through a **compact Transformer encoder**, wherein **self-attention** captures **dependencies** across both **spatial positions** and **semantic scales**. To further consolidate the **multi-scale features**, we incorporate a **Gated Feature Fusion mechanism** [4], which learns **scale-adaptive weights** through **global average pooling** followed by **1×1 convolution** and **softmax activation**. This allows the network to **dynamically balance contributions** from each level and propagate a **semantically coherent**, **globally fused feature map**.

In the **mask head**, we prepend a **Spatial–Channel Transformer module** prior to the original convolutional layers. This module integrates **Efficient Channel Attention (ECA)** [5], which applies **1D convolution** to capture **local inter-channel dependencies** and recalibrates feature responses to emphasize **informative channels**. In parallel, a **deformable convolutional block** [6] **adaptively samples spatial locations** based on **learned offsets**, enabling **precise alignment** with **non-rigid and fine-grained cell contours**. These attention components are followed by **batch normalization** and **residual addition** to preserve the **stability of feature representations**. By combining **channel-wise** and **spatially adaptive mechanisms**, this module enhances the **semantic expressiveness** and **boundary sensitivity** of the **region-of-interest features**, thereby improving the **overall accuracy** of **pixel-wise instance segmentation**.

Regarding the training and data preprocessing procedures in this study, the 209 training images were partitioned into training and validation subsets in an **80:20 ratio**. This split enabled systematic performance monitoring at each training epoch, thereby mitigating the risk of blind submission without validation. **Given the limited dataset size, data augmentation techniques were applied to the training subset to improve generalization.** Specifically, the augmentation pipeline included **color jitter**, **horizontal and vertical flipping**, and **CutOut** [7], the latter of which randomly occludes regions of the image with zero-valued square patches to simulate occlusion and increase robustness. For the optimization strategy, we adopted a **warm-up phase** during the early epochs, followed by a **cosine annealing schedule** to dynamically adjust the learning rate. Empirical results demonstrate that our improvements—introduced atop the baseline architecture—**enabled the model to not only outperform both weak and strong baselines** defined in the benchmark, **but also exceed the performance of the originally designated baseline model.**

2. Method

2.1 Data Preprocessing

(1) Data Loader

In this laboratory assignment, due to the heterogeneous spatial resolutions across the dataset and the requirement to preserve original aspect ratios for accurate mAP evaluation, no resizing was applied to the training, validation, or test images. Instead, two custom Dataset classes and a simple collate_fn were implemented within dataloader.py to support variable-sized inputs. The MedicalInstanceDataset loads each sample's image.tiff —converting single-channel or RGBA images to RGB when necessary—and sequentially reads all corresponding class*.tif label files to generate binary masks for each instance. **These masks are then used to compute bounding boxes, object areas, and is crowd annotations**, which are collectively packaged into a target dictionary compatible with the **Mask R-CNN [1]** framework. Conversely, the MedicalTestDataset is responsible solely for loading images and retrieving COCO-style image IDs via an external JSON mapping file, yielding outputs in the form of (image, image_id, filename).

As no global resizing is performed, the custom collate_fn aggregates each image and its annotation into tuples without attempting to standardize dimensions. This ensures that the DataLoader can batch together inputs of varying sizes while maintaining compatibility with **Mask R-CNN [1]**'s input requirements. Consequently, the model is trained and evaluated on images in their original resolution, aligning with both the evaluation protocol and the nature of the input data.

(2) Data Augmentation

In the data augmentation phase, our implementation includes the following three techniques:

a. **ColorJitter:**

To introduce photometric variations, **we apply torchvision.transforms.ColorJitter to adjust brightness, contrast, and saturation.** The augmentation is conditionally applied based on a predefined probability threshold for HSV jitter; when a random value falls below this threshold, the transformation is triggered.

b. Horizontal Flip and Vertical Flip:

Images are randomly flipped along the horizontal and vertical axes according to preset flip probabilities. Since flipping alters the spatial arrangement of objects, we also update the corresponding bounding box annotations. Specifically, horizontal flips reflect the left and right coordinates with respect to the image width, while vertical flips reflect the top and bottom coordinates relative to the image height. This ensures that spatial annotations remain consistent with the transformed image.

c. CutOut[7]:

The CutOut**[7]** strategy involves randomly masking out square regions of the image during training, encouraging the model to rely on more global contextual cues rather than overfitting to specific local patterns. This helps improve generalization. In our implementation, CutOut is applied with a probability of 0.5. **When triggered, a fixed-size square region of 50×50 pixels is randomly selected and set to zero (black),** effectively removing both the visual and annotated information in that area from the training input. This jointly modifies the image and its corresponding binary mask, enforcing stronger robustness against occlusion.

2.2 Model Architecture

In this study, we employed two distinct backbone networks, both initialized with pretrained weights from the ImageNet dataset provided by the torchvision library. Nevertheless, the core framework throughout our experiments remains based on the **Mask R-CNN [1]** architecture for instance segmentation. Accordingly, this section begins with an overview of the standard **Mask R-CNN [1]** pipeline, followed by a detailed description of the two backbone variants used in our experiments. Lastly, we elaborate on the additional experiments conducted to enhance the performance of the neck and head modules through the integration of attention-based modifications.

2.2.1 Backbone Network

In the **Mask R-CNN framework** [1], the **backbone** is responsible for extracting **hierarchical convolutional features** from the input images. A variety of network architectures have been proposed as potential backbones, including **ResNet** [2], **EfficientNet** [3], and **ConvNeXt** [8], among others. Considering **hardware constraints** and a **model parameter limit** of 200 million, this study adopts **ResNet-101** [2] and **EfficientNet-V2** [3] as backbone networks. Both models are initialized with **pretrained weights** obtained from the **ImageNet-1K** [14] dataset to leverage prior semantic knowledge and **accelerate convergence** during training.

(1) Using ResNet-101[2] as the backbone.

ResNet-101 [2] begins with a **7×7 convolutional layer (stride=2)** followed by a **3×3 max-pooling operation**, serving as the **initial feature extractor**. The network proceeds through **four main stages—Conv2_x to Conv5_x—each composed of bottleneck blocks** that utilize a sequence of **1×1, 3×3, and 1×1 convolutions**. These blocks **reduce, process, and then restore the channel dimensions**, and incorporate **identity shortcuts via residual connections** to **mitigate gradient vanishing** and **facilitate deep network training** [2].

In this work, we employ the **ResNet101_Weights.IMAGENET1K_V2 pretrained on the ImageNet-1K dataset** [14] to **initialize the backbone**. During training, we set **trainable_layers=3**, enabling **gradient updates only for the final three stages—Conv3_x, Conv4_x, and Conv5_x**—which together contain **30 bottleneck blocks**. In contrast, the **initial stem module and Conv2_x are kept frozen** to **retain stable low-level representations**. This **selective fine-tuning strategy** allows the model to maintain **robust**

general-purpose features while **adapting higher-level semantics** to the **domain-specific task of instance segmentation** in **high-resolution medical microscopy images**.

(2) Using EfficientNet_V2[3] as the backbone.

EfficientNetV2 [3] utilizes an enhanced compound scaling approach to jointly optimize network depth, width, and input resolution. The architecture begins with a 3×3 convolutional stem followed by BatchNorm and SiLU activation, then proceeds through a mix of **Fused-MBConv and MBConv blocks**. Each block expands channels via a 1×1 pointwise convolution, applies depthwise convolution for spatial encoding, and includes a **Squeeze-and-Excitation module [9]** to adaptively reweight channels. Residual connections are used throughout to stabilize training. Compared to EfficientNetV1, the V2 variant leverages Fused-MBConv more heavily in early layers to **accelerate convergence across tasks. We initialize the model with *EfficientNet_V2_M_Weights.IMAGENET1K_V1***, pretrained on ImageNet-1K [14]. During training, the stem and early Fused-MBConv blocks are frozen to retain general low-level features, while later layers remain trainable to adapt semantic representations. Four feature maps at strides of 4, 8, 16, and 32 are extracted and passed to the FPN [10] for multi-scale integration.

2.2.2 Feature Pyramid Network[10]

In Mask R-CNN[1], the Feature Pyramid Network (FPN) [10] constructs a multi-scale feature pyramid with progressively decreasing spatial resolutions and increasing semantic richness through a top-down information flow coupled with lateral connections. Specifically, feature maps from different stages of the backbone network are first projected into a uniform number of channels via 1×1 convolutions. Then, starting from the deepest stage, high-level semantic features are upsampled via bilinear interpolation and fused with lower-level features at the corresponding resolution through element-wise addition. This process enables the preservation of both low-level spatial detail and high-level semantic context. Additionally, to enhance the representation of very large objects, an extra top-level feature map (P6) is often appended. By maintaining a consistent channel dimension across all levels, this hierarchical structure enables the **Region Proposal Network (RPN)**, classification and regression heads, and the **Mask Head** to **simultaneously consume features at multiple scales** (from P2 to P6). As a result, the model achieves both fine-grained localization for small instances and holistic understanding of large objects, thereby significantly improving the **accuracy and robustness of instance segmentation across varying object sizes**.

2.2.3 Region Proposal Network

Within the Mask R-CNN[1] framework, the Region Proposal Network (RPN) remains responsible for extracting candidate object regions from the multi-scale feature maps generated by the backbone–FPN hierarchy [10]. However, its outputs not only serve the downstream classification and bounding box regression tasks, but also critically **determine the sampling quality of Regions of Interest (RoIs) for the mask prediction branch.**The RPN begins by applying a shared 3×3 convolution with ReLU activation to each FPN level, thereby capturing local spatial context. Two parallel 1×1 convolutional heads then predict the objectness scores and bounding box regression offsets, respectively. At each spatial location, a set of predefined anchors—configured with various scales and aspect ratios—is evaluated in parallel to accommodate a wide range of cell morphologies, from minute to large. All anchors are ranked based on objectness, and a two-stage **non-maximum suppression (NMS)** process is performed: the first within each scale level to remove redundant proposals, and the second across levels to consolidate the top-ranked proposals globally. A fixed number of high-recall **region proposals** is then retained. These proposals are **aligned via RoIAlign** to a unified spatial resolution and passed, along with their corresponding FPN features, to the **Box Head** and **Mask Head** for fine-grained classification, bounding box refinement, and pixel-wise mask prediction.

2.2.4 Heads

In the Mask R-CNN[1] architecture implemented in this work, the **Detection Head comprises a sequential inference pipeline consisting of three interconnected submodules.** The process begins with RoIAlign, which identifies the appropriate level within the feature pyramid for each candidate region and performs bilinear sampling on the corresponding feature map. This operation precisely aligns and reshapes each region of interest (RoI) to a fixed spatial resolution, effectively eliminating the spatial misalignments introduced by quantization in traditional RoI Pooling. As a result, the subsequent modules receive RoI features that are semantically consistent and resolution-normalized.

Next, the Box Head applies two fully connected layers to the aligned RoI features in order to capture global semantic information. This representation then bifurcates into two parallel branches: a classification branch, which produces class probabilities (including background) via a softmax activation, and a regression branch, which outputs four-dimensional bounding box offsets to refine the initial anchor positions and scales. **These two branches share the high-level RoI representation and are jointly optimized using a multi-task loss**, allowing the network to simultaneously minimize classification and localization errors.

Running **parallel to the Box Head**, the Mask Head is dedicated to pixel-level instance segmentation. It first employs a series of convolutional layers to extract localized texture and shape cues within each RoI, **followed by a deconvolution (upsampling) step to restore higher spatial resolution**. A final pointwise convolution then produces per-class mask logits. During inference, the network selects the mask corresponding to the predicted class and remaps it to the original image coordinates, thereby generating an accurate binary mask for each detected instance.

2.2.5 Our proposed methods in additional experiments

(1) Introduce FPNTransformer between FPN[10] and RPN

In the conventional Feature Pyramid Network (FPN) [10], high-level semantics are progressively propagated from deep to shallow layers through top-down lateral connections. However, this design only permits interactions between adjacent levels, limiting its ability to model long-range dependencies across multiple scales and spatial positions. To overcome this constraint, we propose the **FPNTransformer**, **inserted between the FPN and RPN stages**, to enable global interactions across all scales and locations in a single step, while employing a dynamic gating mechanism to adaptively weight the contribution of each scale based on image-specific context.

The FPNTransformer first standardizes the channel dimensions of the **five pyramid levels (C2–C6), then applies adaptive average pooling to compress each feature map into a fixed resolution of $r \times r$ (where $r = 14$)**, which is subsequently flattened into a sequence of tokens. To retain spatial and hierarchical information, each token is augmented with 2D sinusoidal positional encodings and a learnable level embedding. The concatenated sequence is passed through a **multi-head self-attention encoder [11],** enabling any pair of tokens—regardless of scale or position—to compute attention weights and establish global cross-scale interactions. The enhanced sequences are reshaped back to their original spatial resolutions and merged with the original FPN outputs via residual connections.

To further refine semantic consistency, we introduce **Gated Feature Fusion** [4], which extracts scale descriptors via global average pooling and generates adaptive weights **through 1×1 convolution and softmax normalization.** These weights guide the fusion of multi-scale features into a unified semantic representation, which is then redistributed back to each level. By combining global attention with scale-aware gating, the FPNTransformer enhances both spatial precision and semantic coherence, yielding a more informative feature foundation for downstream RPN and prediction heads.

(2) Introduce Spatial–Channel Transformer in the Mask Head

Prior to the upsampling convolutional layers in the original Mask Head, we incorporate a residual attention module that integrates both channel-wise and spatial attention mechanisms to suppress redundant signals and highlight discriminative regions at the RoI level. The **module first applies Efficient Channel Attention (ECA) [5]**, **which captures local inter-channel dependencies using lightweight 1D convolution**, enabling dynamic recalibration of feature maps to emphasize informative channels related to object boundaries or textures, while attenuating background noise.

Following this, **a Deformable Convolution block [6] generates learnable sampling offsets in the spatial domain**, allowing the convolutional kernel to adapt its receptive field to irregular and fine-grained cellular contours. This flexibility enables more precise modeling of non-rigid object boundaries, beyond the limitations of fixed-grid convolutions.

To ensure feature stability, the attention-enhanced output undergoes Batch Normalization and is combined with the original path through residual addition, preserving the underlying semantic content and preventing overfitting to the recalibrated features. **Subsequently, a lightweight 1×1 feedforward network (Conv → GELU → Dropout → Conv)** introduces additional nonlinearity and **is again fused via residual connection**. The final output tensor thus encodes both global semantics and local detail, which empirically improves the pixel-level segmentation accuracy of densely packed instances within the Mask Head.

2.3 Hyperparameters Settings and Training Configurations

2.3.1 Hyperparameters Settings

The experiments were conducted using an **NVIDIA RTX 4060 Ti 16GB GPU**. Due to hardware and time constraints, we set the **batch size to 2** throughout training. The total number of training epochs was configured to **40**, with an initial **5-epoch warmup phase** to facilitate stable optimization. For learning rate scheduling, we adopted the **cosine annealing strategy** to progressively reduce the learning rate and improve convergence. The table below summarizes the **pretrained weights** used for the two backbone networks as well as the **hyperparameter settings** employed in the additional experiments.

Hyperparameter	Value
Batch size	2
Initial learning rate (LR)	2×10^{-4}
η(min)	5×10^{-6}
Epochs	40
Warmup Epochs	5
Weight decay	1×10^{-4}
Optimizer	AdamW

Table 1. Hyperparameters Settings

2.3.2 Training Configurations

(1) Data Augmentation and Usage

As described in the Introduction and Section 2.1, we applied several data augmentation techniques to enhance the diversity of the training set. Given that the augmentation strategies adopted in this study are relatively lightweight, we chose to retain all augmentation operations throughout the entire training process, without disabling any of them during the later epochs.

(2) Warmup and Cosine Annealing

We configured the warm-up phase to span the first five epochs of training, during which the learning rate increases linearly from zero to the predefined initial learning rate. Following this phase, the learning rate is adjusted using a cosine annealing schedule, gradually decaying from the initial value to a pre-specified minimum value (eta_min) as the total number of training epochs is reached.

(3) Calculation of the Loss Function

The loss function employed in this study comprises five components: (1) objectness loss, (2) RPN bounding-box regression loss, (3) classification loss, (4) RoI bounding-box regression loss, and (5) pixel-wise segmentation loss.

Specifically, the objectness loss is defined as **binary cross-entropy** to evaluate the Region Proposal Network (RPN)'s ability to **distinguish foreground anchors from background**, ensuring a high recall rate. The RPN bounding-box regression loss adopts the Smooth L1 function to regress the anchor's positional offsets and scales, providing coarse localization of candidate regions. The classification loss is computed as **softmax cross-entropy over the Box Head's predicted classes**, assigning each RoI to the correct object category. The bounding-box regression loss is again formulated **using Smooth L1 to refine RoI-level box coordinates and improve localization accuracy**. Lastly, the mask loss **applies binary cross-entropy on a per-pixel basis for positive RoIs**, encouraging the model to produce accurate instance-level segmentation masks. During training, these five losses are summed to form the total loss, and each component is logged individually at every epoch. This allows us to monitor the contribution of each loss term and to diagnose learning bottlenecks within the model.

3. Results & Additional experiments

3.1 The baseline experiment and additional experiment results

(1) Using ResNet-101[2] as the backbone of our model

In this experiment, we adhered to the hyperparameter configuration outlined in the previous section. As illustrated in the training curves, the **overall loss converges with moderate oscillations but remains generally smooth**. The training loss stabilizes around **0.51**, while the validation loss converges near **0.92**. In terms of detection performance, the **best validation mAP was achieved at epoch 34**, reaching **0.3398**.

It is worth noting, however, that the validation loss **plateaus after approximately epoch 30**, whereas the mAP continues to show marginal improvement beyond that point. This divergence between the loss and performance metrics suggests a potential onset of **overfitting**, warranting caution in subsequent training strategies. Finally, the output result file was submitted to the **CodaBench** evaluation platform, where it attained a **public score of 0.3739 mAP** and a **private score of 0.3839 mAP**, both of which **surpass the strong baseline performance of 0.231 mAP**. This outcome demonstrates the effectiveness of our proposed architecture in improving instance segmentation accuracy under the given constraints.

{"public_score": 0.37394833518166837, "private_score": 0.3839289582328426}

Fig 1. The mAP score using ResNet-101[2].

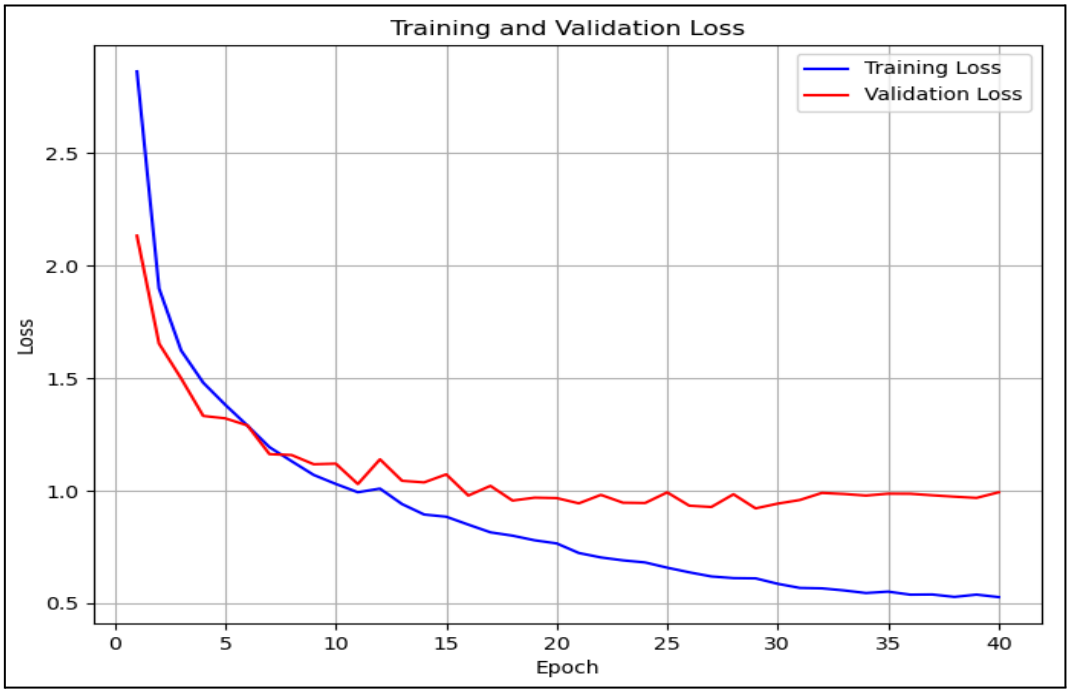

Fig 2. The plot of Training and Validation Loss using ResNet-101[2] as the backbone of our model.

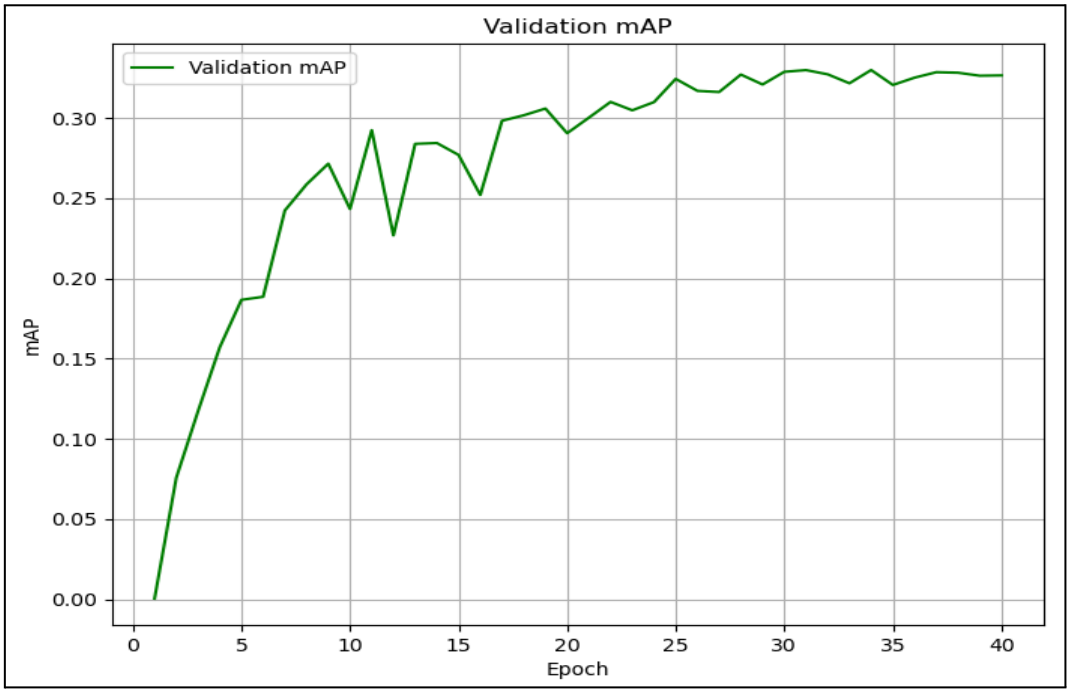

Fig 3. The plot of Validation mAP using ResNet-101[2] as the backbone of our model.

(2) Using ResNet-101[2] as the backbone of our model with Attention-Enhanced FPN[10] & Heads (ours)

In our experiment, we adhered to the hyperparameter configurations described in the preceding section. As shown in the loss curves, the overall training loss converges smoothly despite minor fluctuations, stabilizing around 0.50, while the validation loss converges **near 0.89**. The model achieved its **best validation mAP of 0.3492 at epoch 26**.

However, it is noteworthy that the validation loss plateaued around the same epoch, indicating a potential **risk of overfitting in the latter stages of training.** The final output result file was submitted to the **CodaBench** evaluation platform, where it achieved a **public score of 0.4132 mAP** and a **private score of 0.4250 mAP**. These results not only **surpass the strong baseline of 0.231 mAP**, but also **exceed our own baseline model by approximately 4.1%**, demonstrating the efficacy of our proposed improvements.

{"public_score": 0.4131693210294622, "private_score": 0.42496044726858234}

Fig 4. The plot of the mAP score using ResNet-101[2] as the backbone of our model with Attention-Enhanced FPN[10] & Heads (ours).

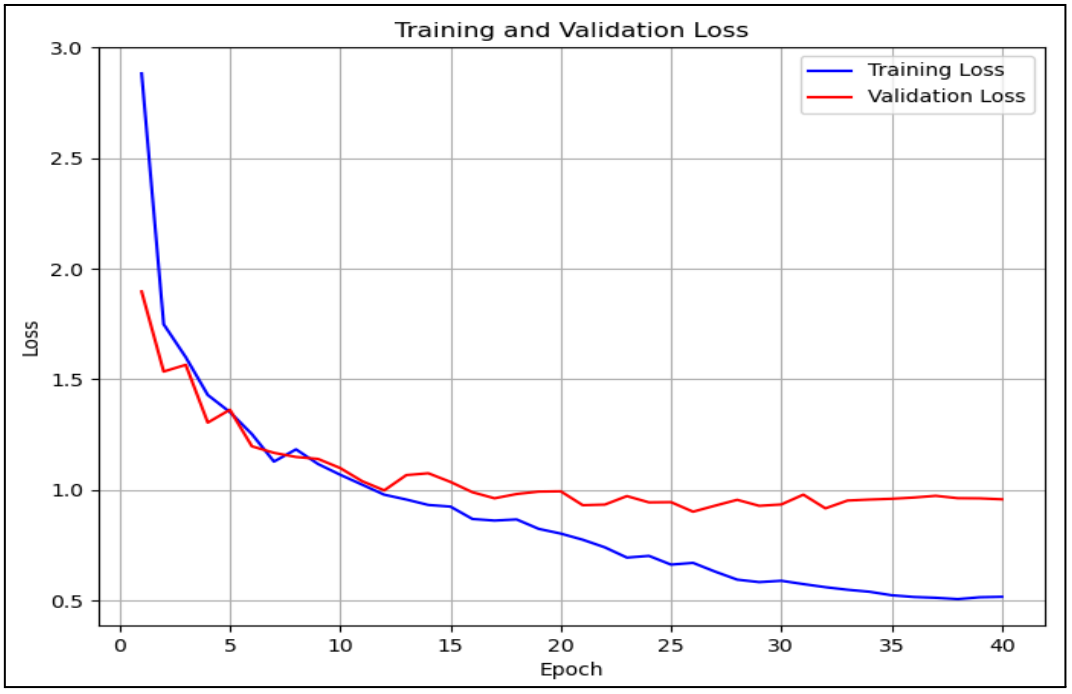

Fig 5. The plot of Training and Validation Loss using ResNet-101[2] as the backbone of our model with Attention-Enhanced FPN[10] & Heads (ours).

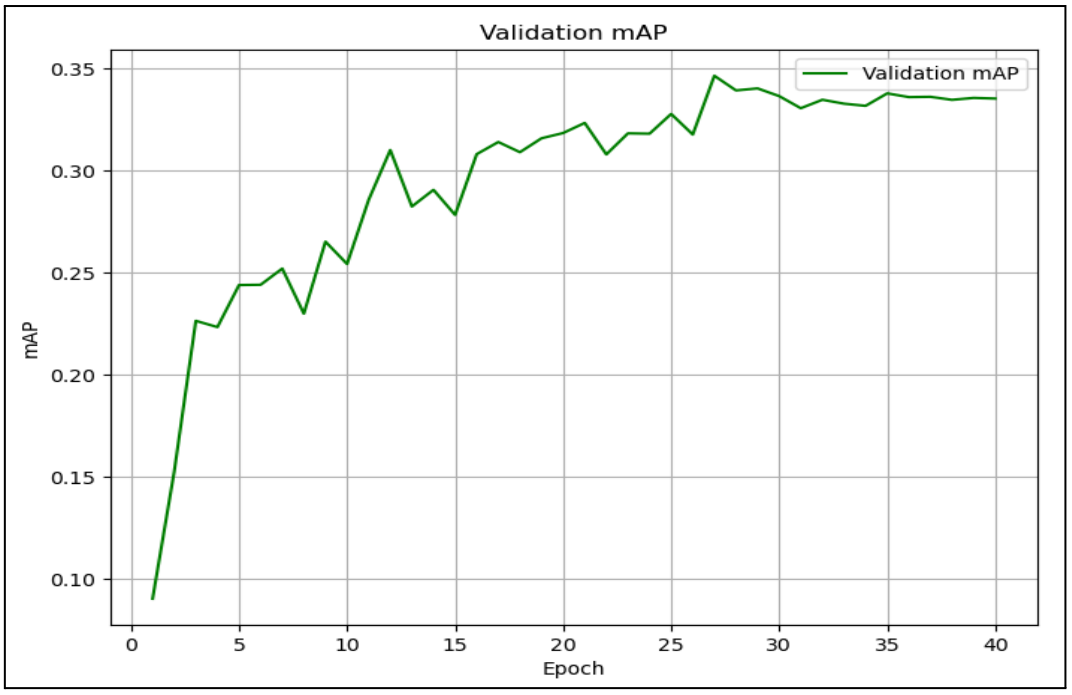

Fig 6. The plot of Validation mAP using ResNet-101[2] as the backbone of our model with Attention-Enhanced FPN[10] & Heads (ours).

(3) Using EfficientNetV2[3] as the backbone of our model

In this experiment, we continued to follow the hyperparameter settings outlined in the previous section. As illustrated by the loss curves, the overall training process exhibited smooth convergence, with the training loss stabilizing around 0.54 and the **validation loss converging near 0.93**. The model achieved its peak validation mAP of **0.3516 at epoch 29**, and no clear signs of overfitting were observed throughout training. The final output result file was submitted to **CodaBench**, where it obtained a **public score of 0.4015 mAP** and a **private score of 0.3940 mAP**. These results represent a **slight improvement (approximately 1% mAP)** over the baseline model using **ResNet-101 [2]** as the backbone, and **significantly surpass the strong baseline performance of 0.231 mAP**.

{"public_score": 0.40148627701741896, "private_score": 0.3939815741895322}

Fig 7. The mAP score using EfficientNetV2[3].

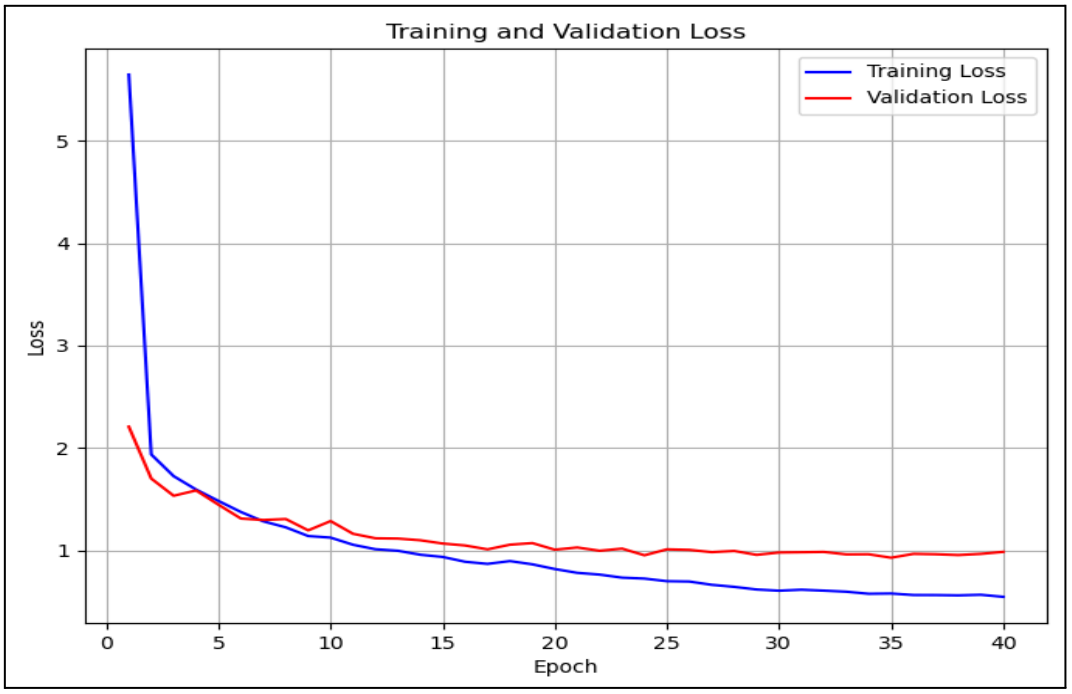

Fig 8. The plot of Training and Validation Loss using EfficientNetV2[3] as the backbone of our model.

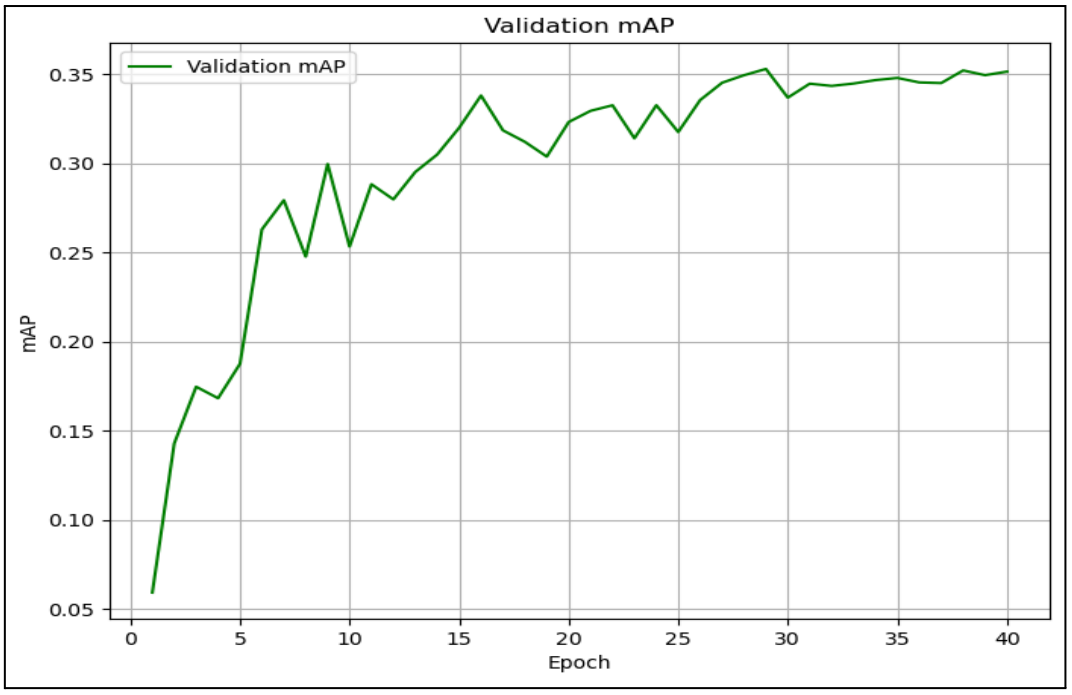

Fig 9. The plot of Validation mAP using EfficientNetV2[3] as the backbone of our model.

(4) Using EfficientNetV2[3] as the backbone of our model with Attention-Enhanced FPN[10] & Heads (ours)

In this experiment, we followed the hyperparameter settings described in the previous section. As shown in the loss curve, the **overall convergence was relatively smooth**, with the **validation loss stabilizing around 0.92** and the **training loss converging near 0.52**. The **best validation mAP of 0.3528 was achieved at epoch 30**. Upon submitting the final model to **CodaBench**, we obtained a **public score of 0.4126 mAP** and a **private score of 0.4148 mAP**, which not only **surpassed the strong baseline score of 0.231 mAP**, but also **outperformed our previous baseline using EfficientNetV2 [3] by approximately 2.1% mAP**.

{"public_score": 0.4126099669955071, "private_score": 0.41484973823851573}

Fig 10. The mAP score using EfficientNetV2[3] with Attention-Enhanced FPN[10] & Heads (ours).

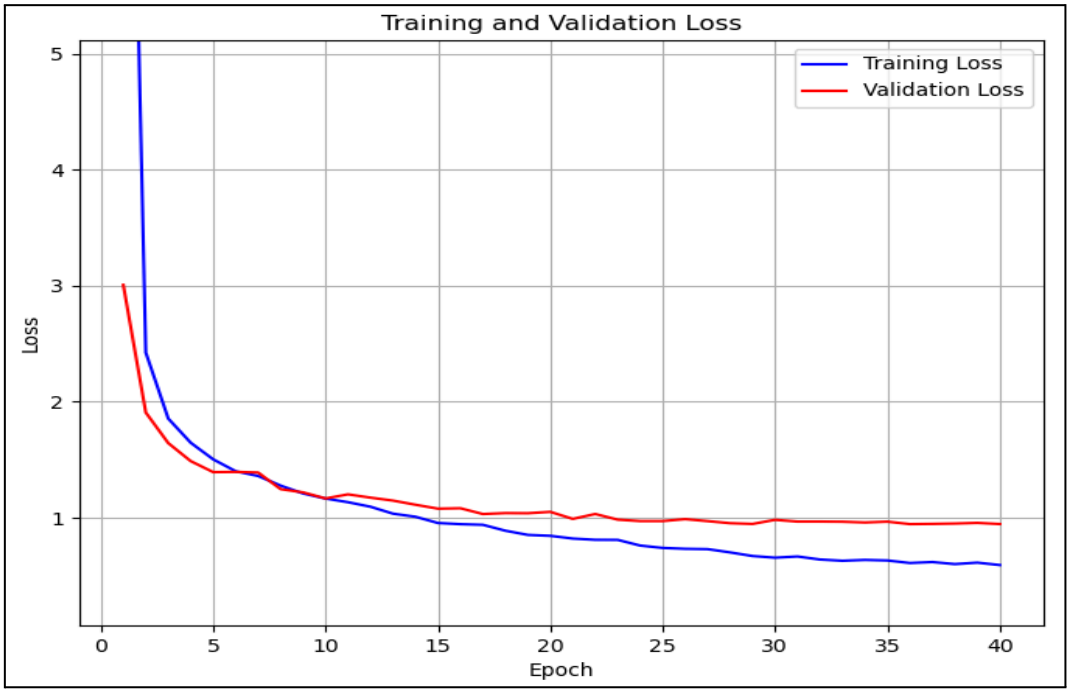

Fig 11. The plot of Training and Validation Loss using EfficientNetV2[3] as the backbone of our model with Attention-Enhanced FPN[10] & Heads (ours).

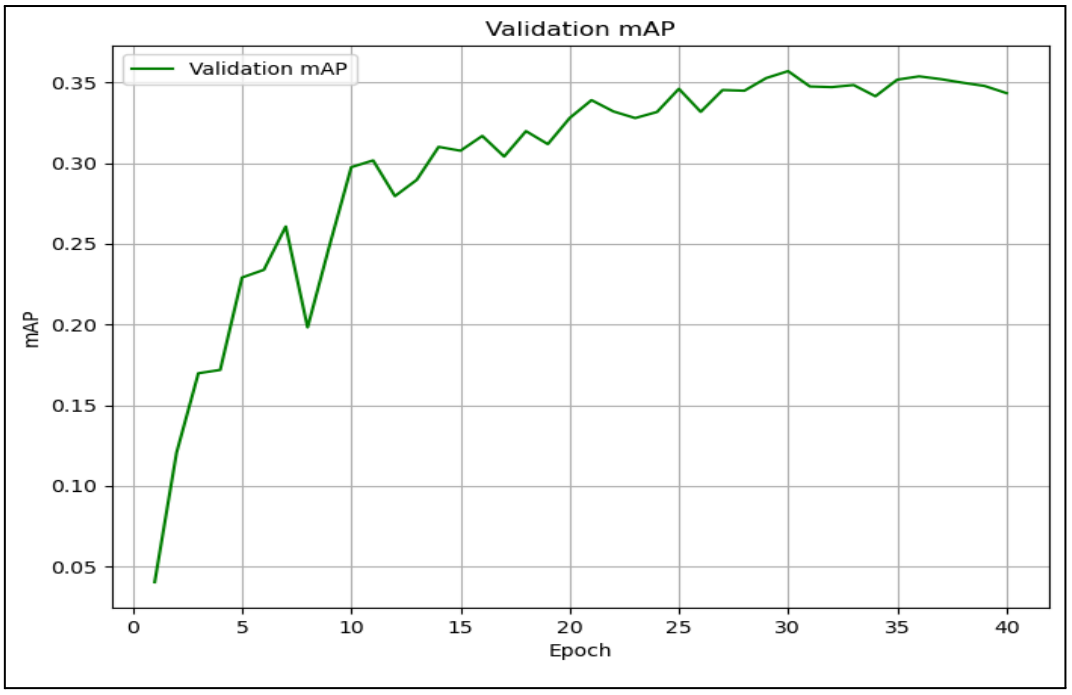

Fig 12. The plot of Validation mAP using EfficientNetV2[3] as the backbone of our model with Attention-Enhanced FPN[10] & Heads (ours).

(5) Visualization of Experiment Results

To qualitatively evaluate model performance, we applied the best-performing weights trained with **ResNet-101 [2] and the proposed attention module** to the annotated validation set, and visualized the inference results using **visualization.py**. As shown in **Figure 13** (with **ground truth on the left** and **predicted masks on the right**), the model demonstrates **reasonable segmentation quality overall**. However, it tends to **miss smaller or densely clustered and overlapping instances**, leading to occasional false negatives. Additionally, we observe some **false positives**, where the model predicts instances in regions **not annotated in the ground truth**, suggesting either annotation incompleteness or model overgeneralization.

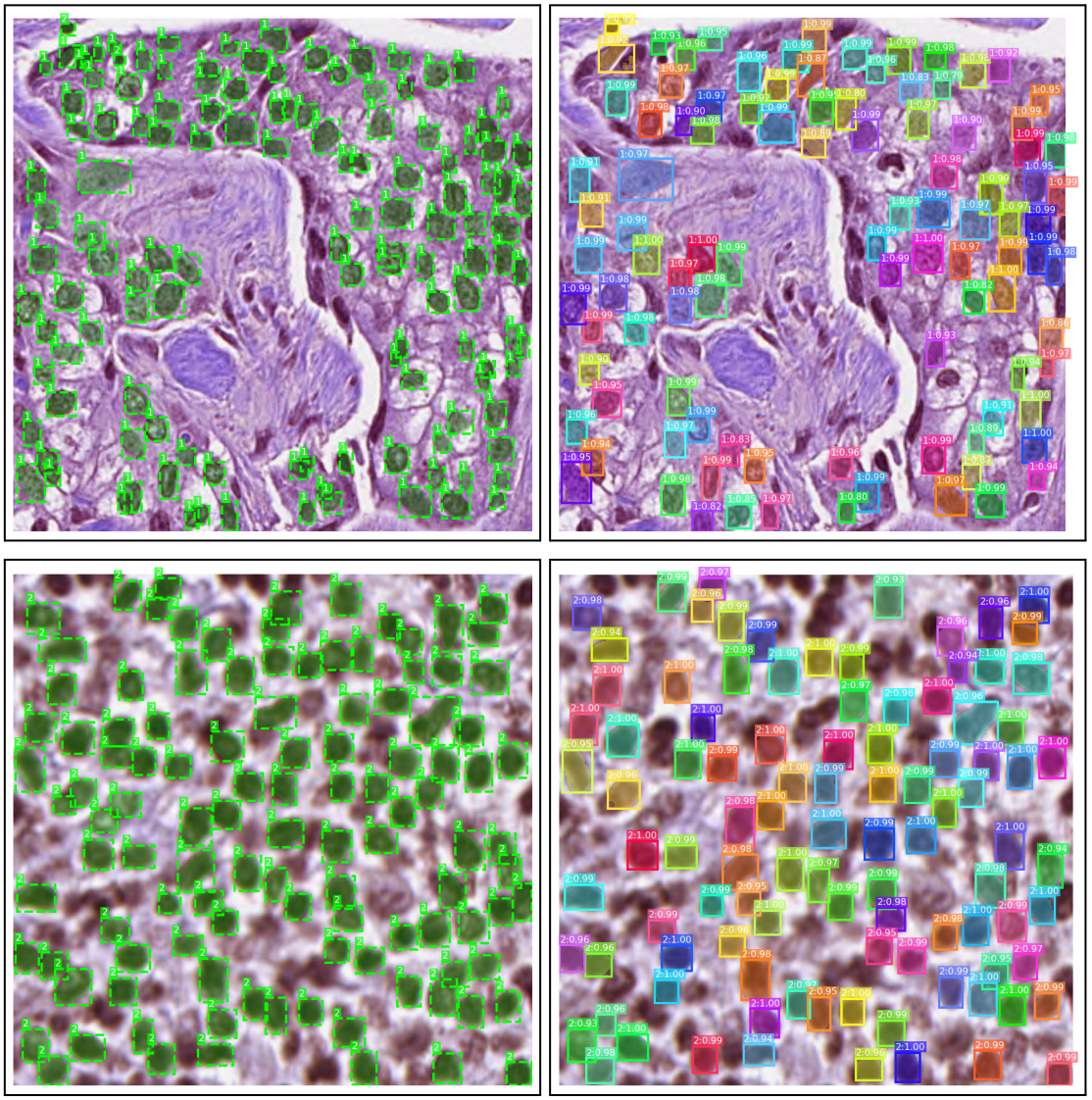

Fig 13. Visualization of the Validation Results (best weight of ResNet-101[2] w/attn)

4. Discussion and Conclusion

4.1 Experiment Results

Model Name	val mAP	test mAP (public)	test mAP (private)
ResNet-101[2] o/attn	0.3398	0.3739	0.3839
ResNet-101[2] w/attn	0.3492 **(+0.94%)**	**0.4132** **(+3.83%)**	**0.4250** **(+4.11%)**
EfficientNetV2[3] o/attn	0.3516	0.4015	0.3940
EfficientNetV2[3] w/attn	**0.3528** **(+0.12%)**	0.4126 **(+1.11%)**	0.4148 **(+2.08%)**

Table 2. All experiments results

In this experiment, both selected baseline models and their variants augmented with our proposed attention modules **consistently outperform the official strong baseline** (0.231 mAP) by a significant margin.

To begin with, a comparison between the two baseline models shows that **EfficientNetV2 [3] slightly outperforms ResNet-101 [2]** on the test set's private score by approximately **1% mAP**. When evaluating the effect of our attention modules, we observe a notable improvement in both models. For **ResNet-101 [2]**, the attention-augmented version achieved a **0.96% increase in validation mAP** and a **substantial 4.11% gain** on the private score. Similarly, for **EfficientNetV2 [3]**, although the validation mAP improvement was modest (**+0.12%**), the **private score improved by 2.08%**, demonstrating its practical benefit.

When comparing the two attention-augmented architectures based on their test set performance, **ResNet-101 [2] w/ attention** achieved the best result, surpassing **EfficientNetV2 [3] w/ attention** by **1.02% mAP** on the private leaderboard. These findings confirm that introducing the **FPNTransformer between the FPN and RPN stages**, along with inserting the **Spatial–Channel Transformer at the first layer of the Mask Head**, substantially enhances the model's ability to capture **cross-scale semantic dependencies** and **refine instance boundaries**, thereby improving the overall performance of instance segmentation on colour microscopic medical images.

4.2 Model Parameters Comparison

Model Name	Trainable Parameters	Total Parameters
ResNet-101[2] o/attn	62,656,045	62,878,445
ResNet-101[2] w/attn	65,262,941	65,485,341
EfficientNetV2[3] o/attn	54,138,637	54,155,029
EfficientNetV2[3] w/attn	56,745,533	56,761,925

Table 3. Model Parameters Comparison

In this study, the total number of parameters for all four experimental configurations was computed and summarized in **Table 3**, with all models remaining within the constraint of **200 million parameters**. Notably, the addition of our proposed attention modules resulted in only a modest increase of approximately **2.6 million parameters**, without introducing significant training overhead. This demonstrates that our design achieves a favorable balance between computational efficiency and performance gain, effectively **enhancing the baseline models with minimal parameter cost**.

4.3 Future Work

Due to **hardware limitations**, we restricted training to **models under 65 million parameters**, preventing evaluation of **larger, more computationally intensive architectures**. To more thoroughly assess our **attention modules**, future work should explore **diverse backbones** and conduct evaluations on **benchmark datasets**. We also observed **significant class imbalance by executing check_stats.py**, which may hinder **generalization**, especially for **rare categories**. To mitigate this, **stratified validation sampling** is recommended. Moreover, incorporating **Focal Loss [12]** or **gIoU loss [13]** into the existing objective could improve **learning from underrepresented instances** and enhance **segmentation robustness**.

Class	Num of Instances
cell 1	14,537
cell 2	15,653
cell 3	630
cell 4	587

Table 4. Counts of each class in the training and validation sets

Reference

[1] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[3] Tan, M., & Le, Q. (2021, July). Efficientnetv2: Smaller models and faster training. In International conference on machine learning (pp. 10096-10106). PMLR.

[4] Sun, Z., Jin, D., Deng, J., Zhang, M., & Shao, Z. (2023, December). Feature Fusion Module Based on Gate Mechanism for Object Detection. In 2023 IEEE International Conference on Robotics and Biomimetics (ROBIO) (pp. 1-6). IEEE.

[5] Wang, Q., Wu, B., Zhu, P., Li, P., Zuo, W., & Hu, Q. (2020). ECA-Net: Efficient channel attention for deep convolutional neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 11531–11539).

[6] Zhu, X., Hu, H., Lin, S., & Dai, J. (2019). Deformable convnets v2: More deformable, better results. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 9308-9316).

[7] DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.

[8] Woo, S., Debnath, S., Hu, R., Chen, X., Liu, Z., Kweon, I. S., & Xie, S. (2023). Convnext v2: Co-designing and scaling convnets with masked autoencoders. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 16133-16142).

[9] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132-7141).

[10] Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2117-2125).

[11] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[12] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).

[13] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 658-666).

[14] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). *ImageNet: A large-scale hierarchical image database*. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). IEEE.

Appendix

(1) File Structures:

The repository is structured with a datasets/ directory containing the decompressed data—namely the train/ and test_release/ folders and the test_image_name_to_ids.json mapping file—and ten Python modules that implement the full instance segmentation pipeline. The attention_modules.py file extends the baseline Mask R-CNN[1] with **our proposed Transformer- and channel/spatial-attention enhancements**, while augment.py applies a suite of lightweight augmentations (Color Jitter, flips, CutOut[7]) to the training split following an 80/20 train–validation partition. We assess model complexity in cal_param.py by computing total and trainable parameter counts, and use check_stats.py to quantify per-class instance frequencies within the annotated dataset. Data ingestion at native resolution is handled by dataloader.py, which provides custom Dataset classes and a collate function. Training scripts—train_resnet.py and train_effnet_v2.py—instantiate Mask R-CNN [1]with ResNet-101 [2] and EfficientNet-V2 [3] backbones, respectively, and integrate the attention modules into both the FPN and mask head. Inference is performed by inference.py, which produces COCO-style RLE-encoded JSON outputs, while plot_curve.py generates training/validation loss and mAP curves. Finally, visualization.py loads the trained weights to produce qualitative overlays of ground truth and predicted masks on validation or test images.

Fig 14. Our File Structure with 10 python file and a dataset folder (which would be removed before we upload our .zip file to E3 and Github)

(2) Command Line:

a. Overview of Python Files Used by the Command-Line Interface:

--train_dir:
Directory containing the training data (default: dataset/train).

--test_dir:
Directory containing the test images (default: dataset/test_release).

--id_map_json:
JSON file mapping test filenames to COCO image IDs (default: dataset/test_image_name_to_ids.json).

--output_dir:
Directory in which to save checkpoints and output JSON (default: inference_output).

--epochs:
Total number of training epochs (default: 50).

--batch_size:
Number of samples per batch (default: 2).

--lr:
Initial learning rate (default: 1e-4).

--weight_decay:
Weight decay (L2) factor (default: 1e-4).

--val_split:
Fraction of the training set to hold out for validation (default: 0.2).

--num_workers:
Number of DataLoader worker processes (default: 0).

--seed:
Random seed for reproducibility (default: 42).

--patience:
Number of epochs with no validation mAP improvement before reducing LR (when not using cosine annealing).

--early_stop:
Number of consecutive epochs without mAP improvement to trigger early stopping.

--num_classes:
Total number of classes, including background (default: 5).

--checkpoint:
Path to a pretrained model checkpoint for fine-tuning.

--use_attn:
Enable the FPN→Transformer and Spatial–Channel attention modules.

--use_amp:
Use mixed-precision (FP16) training via AMP.

--use_cosine:
Use a CosineAnnealingLR scheduler instead of ReduceLROnPlateau.

--eta_min:
Minimum learning rate for cosine annealing (default: 5e-6).

--warmup_epochs:
Number of linear warm-up epochs before starting cosine decay (default: 5).

--backbone:
Specify the backbone model to use at **inference time**; choose either resnet or effnet.

--mode:
Choose whether to visualize validation (val) or test (test) images.

--num_images:
Specify how many images to process (e.g. --num_images 5 will visualize the first 5 samples).

--score_thresh:
Set the minimum confidence score for displaying predicted masks and boxes (e.g. 0.5 means only predictions ≥50% are shown).

--draw_gt_mask:
When visualizing ground-truth, overlay the GT masks (in semi-transparent green) on the image.

--vis_dir:
Put the output visualized images in the folder.

b. **Training command line:**

python train_resnet.py --use_cosine --warmup_epochs 5 --eta_min 5e-6 --epochs 50 --use_amp --batch_size 2 --lr 2e-4 --use_attn

c. **Inference command line:**

python inference.py --backbone resnet --checkpoint path/to/resnet_best_model.pth --use_attn

d. **Visualization Command line:**

python visualization.py --backbone resnet --checkpoint path/to/resnet_best_model.pth --mode val --num_images 3 --use_attn --draw_gt_mask

e. **Visualization Command line:**

python cal_param.py --backbone effnet --num_classes 5 --use_attn