2022 AI CUP Competition:
Spread Through Air Spaces Segmentation

Jia-Wei Liao, Kuok-Tong Ng, Yi-Cheng Hung

Department of Applied Mathematics, NYCU

February 4, 2024

Abstract

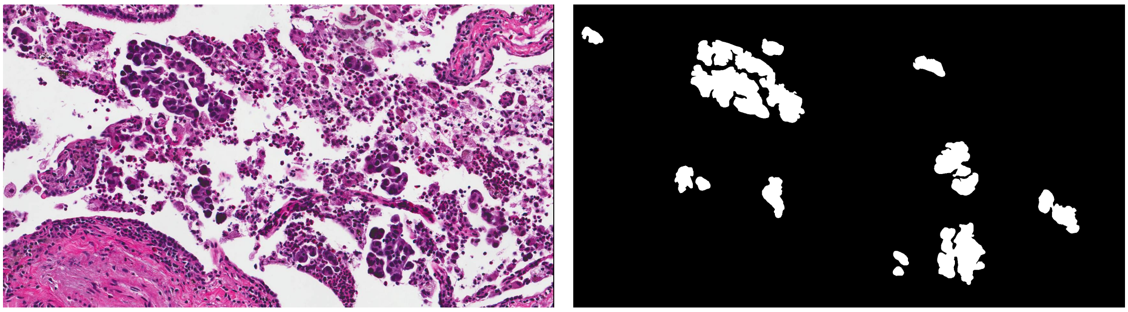
There are many tasks about pattern recognition in computer vision, such as image classification, object detection, semantic segmentation, and so on. They are widely used in medical image analysis. In this competition, we implement the deep learning-based techniques to do the Spread Through Air Spaces (STAS) Segmentation. We use the UNet with the backbone of EfficientNet, which is the state-of-the-art semantic segmentation model. UNet is proposed in 2015. It is an encoder-decoder-based model which can extract the critical features and recover the label of the goal. EfficientNet is proposed in 2020. The Google researcher through the Neural Architecture Search (NAS) to find the best model based on performance. So we use it as a feature extractor. After our experiment, we got a public score of 0.913332. Then we attempted to design the post-processing. Finally, the public score raises to 0.920027, and we got the rank of 3. Unfortunately, our best private score only had 0.910871, and our ranking dropped to 16th. We release the code to GitHub. It is available at https://github.com/YCHung1998/Spread-Through-Air-Spaces-Segmentation.git.

Contents

Chapter 1

Introduction

The competition is about tumor airway diffusion detection in pathological section images of lung adenocarcinomas. It period from April 6th to June 1st in 2022. Semantic segmentation will be used to identify lung adenocarcinomas' pathological features. By integrating deep learning for pathological analysis and analyzing pathological tumor images in conjunction with artificial intelligence, the wealth of tumor-related information that can be gathered can be applied to clinical medicine in order to aid doctors. Spread through air spaces (STAS) is a newly discovered pathological feature of lung adenocarcinoma in recent years, which refers to the spread of tumor cells from the edge of the STAS along the alveolar cavity to adjacent normal lung tissue. For segmentation, we will process 1053 stained sections of size $1716 \times 942 \times 3$. We aim to train a segmentation model using the EfficientNet backbone to segment STAS regions through routine preprocessing model training, cross-validation, and a final inference using test time augmentation and aggregated model predictions.



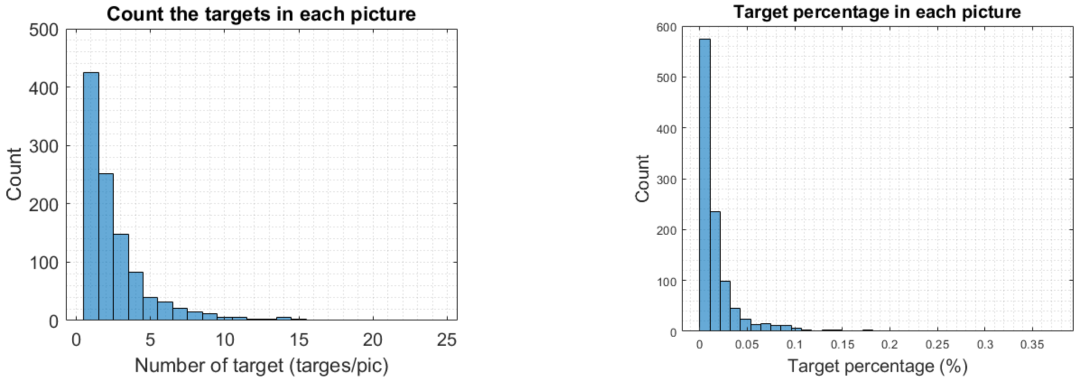
(a) image

(b) mask

Figure 1.1: STAS image 169.jpg image and mask 169_mask.png

1.1 Data Exploration

According to the exploration mask data, all data labels are not empty in the sample, and the mask has several simple objects (which we called ”targets”). We know that most images have at least one object in the label, and a few data have more than ten objects.



(a)

(b)

Figure 1.2: Histogram

We count down the number of each target pixel in the mask and display them as a boxplot in Figure 1.3. Mask pixels range from 1 to 531,379, and some of the labels might not be connected.

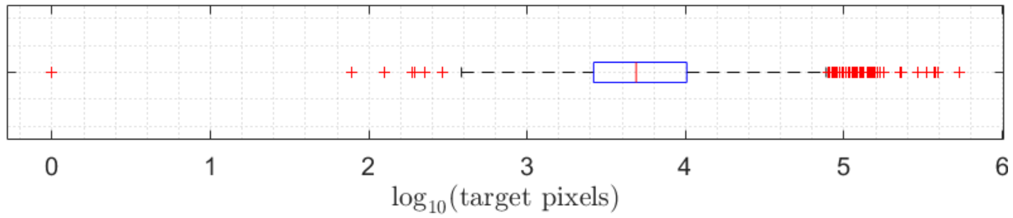
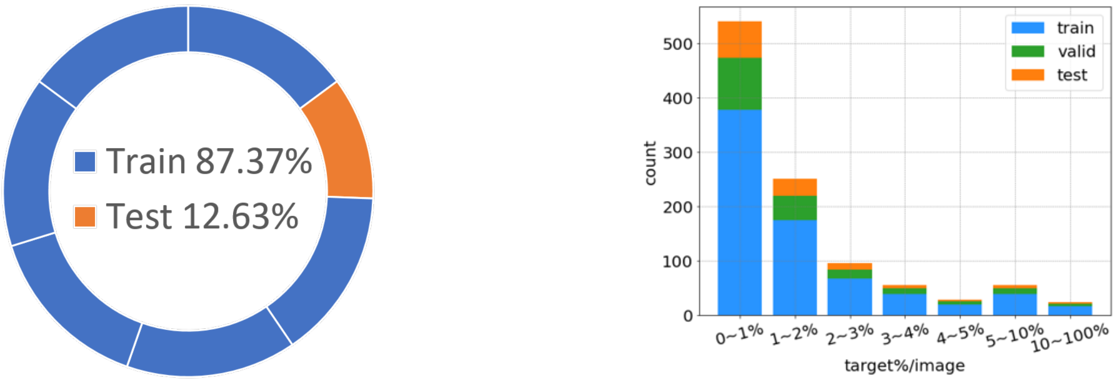


Figure 1.3: Box plot

To begin with, we only have training images. In the days leading up to the competition’s end, more information was released, and now there are Training images, Public images, and Private images. First, we split the 1,053 data into a training set and a testing set. 12.6% of the data is used for testing, and the remaining 87.3% is used for training. We use a 5-fold cross-validation strategy for training. The proportion is shown in Figure 1.4. On the previous statistical chart, we presented the target area to image size ratio, and then we grouped the data at seven intervals. Ensure each group of data exists in each fold and test set.



(a) Pie plot

(b) Bar plot

Figure 1.4: Pie plot and bar plot

Compared to public and private images, the former ones contain 131 images and the latter ones contain 184 images, training data are all labeled. In contrast, public image and private image do not with label. Moreover, the public prediction score can be obtained by uploading the prediction results, while the private prediction score will not be published until the end of competition.

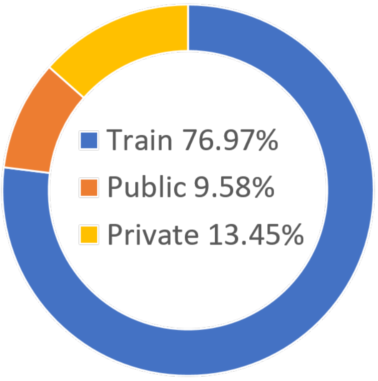


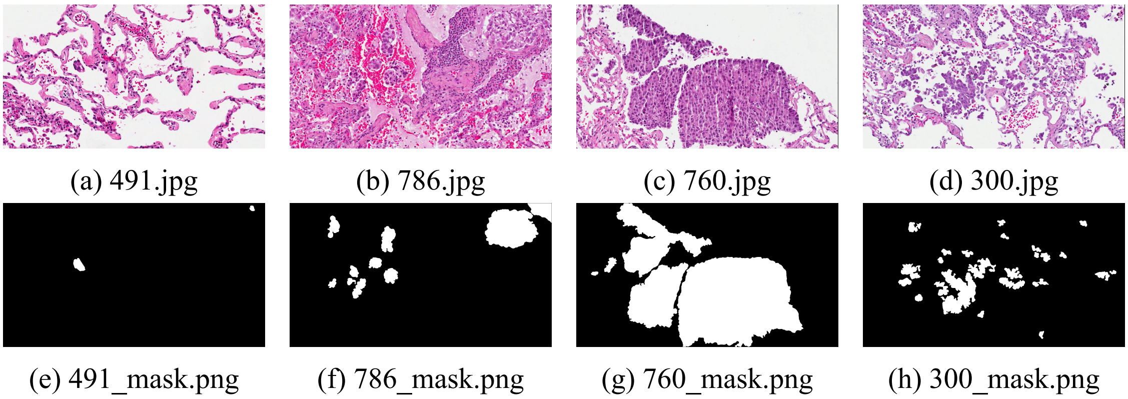
Figure 1.5: Pie plot

1.2 Difficulties

The following are obstacles when carrying out this project:

1. There are various sizes of targets in the mask: The smallest object is only 1 pixel, while the largest one is as large as 531,379 pixels. If setting a threshold to remove those target pixels of predictions below it, FN (False Negative) will rise, and label information may be lost when the image is resized to a small image. Therefore, this scheme needs to pad the original data to reduce the loss of information.
2. Some Label masks are either close to the border or close to the corner: Since the lung slice image is an image obtained by a high-magnification electron microscope,

the data is formed by splitting into multiple photographs. It is inherently possible to split the target in two when generating multiple images, which adds some difficulties in prediction.



(a) 491.jpg (b) 786.jpg (c) 760.jpg (d) 300.jpg

(e) 491_mask.png (f) 786_mask.png (g) 760_mask.png (h) 300_mask.png

Figure 1.6: Training images and corresponding masks

Chapter 2

Related Work

Semantic segmentation is an important topic in computer vision. In 2014, Fully Convolutional Networks (FCN) [6] has been proposed. FCN substitute the fully conneted layer (last layer) by transposed convolution layer in the classification model. Therefore, the model can accept variant size of input image and can be trained end-to-end. FCN has establish the foundation of semantic segmentation models. In 2015, UNet [8] has been proposed. UNet use a symmetric encoder-decoder structure, which means the decoder part is more robust than FCN. Moreover, skip connection between encoder and decoder that can keeps the finer feature (from the encoder), mitigate the problem of information loss that FCN faced.

UNet has been widely used in medical image segmentation. There are many variant of UNet in this recent years. For example, inspired by ResNet and DenseNet, ResUNet [**?**] and DenseUNet [2] has been proposeed in 2018. nnUNet (no-new-Net) [4] proposed in 2019, which concentrate on data pre-processing, training scheme and inference-scheme and get a great improvement. This proves the importance of understanding of the data. TransUNet [1], which proposed in 2021, try to enhance the encoder part by combining CNN and transformer.

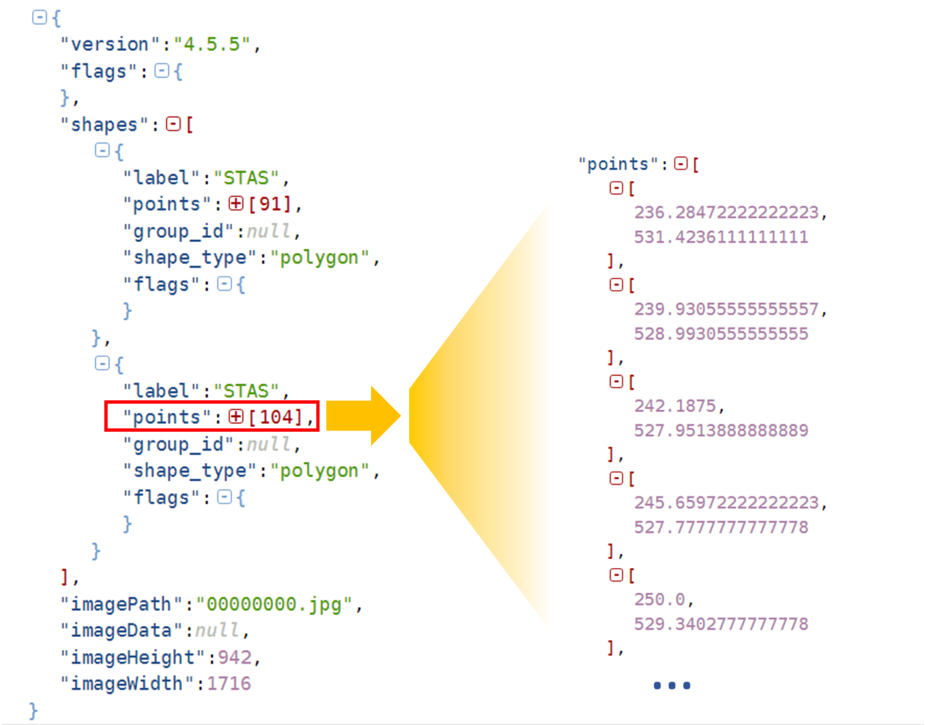
Chapter 3

Proposed Approach

3.1 Data Pre-processing

3.1.1 Create image mask file

In Figure 3.1, we shows the original annotation data which is converted to the segmentation mask.



```
⊟{
   "version":"4.5.5",
   "flags":⊟{
   },
   "shapes":⊟[
      ⊟{
         "label":"STAS",
         "points":⊞[91],
         "group_id":null,
         "shape_type":"polygon",
         "flags":⊟{
         }
      },
      ⊟{
         "label":"STAS",
         "points":⊞[104],
         "group_id":null,
         "shape_type":"polygon",
         "flags":⊟{
         }
      }
   ],
   "imagePath":"00000000.jpg",
   "imageData":null,
   "imageHeight":942,
   "imageWidth":1716
}
```

```
"points":⊟[
   ⊟[
      236.28472222222223,
      531.4236111111111
   ],
   ⊟[
      239.93055555555557,
      528.9930555555555
   ],
   ⊟[
      242.1875,
      527.9513888888889
   ],
   ⊟[
      245.65972222222223,
      527.7777777777778
   ],
   ⊟[
      250.0,
      529.3402777777778
   ],
      •••
```

Figure 3.1: JSON format

3.1.2 Splitting training and validation set in 5-fold

There are 1,053 first-hand training materials divided into training sets and test sets. We use 5 folders to manage training sets and take one of the folders as a validation set at a time. Training set, validation set and test set account for 69.90%, 17.47% and 12.63%, respectively.

3.2 Data Transform and Augmentation

Hematoxylin and eosin staining (H & E staining) procedure is the principal method in histology. The hematoxylin stains cell nuclei purplish blue, and eosin stains the extracellular matrix and cytoplasm pink. We follow this property and apply the approach, [7], to increase the diversity of our train dataset. According to the table below, we can understand different dyes with corresponding properties and effects.

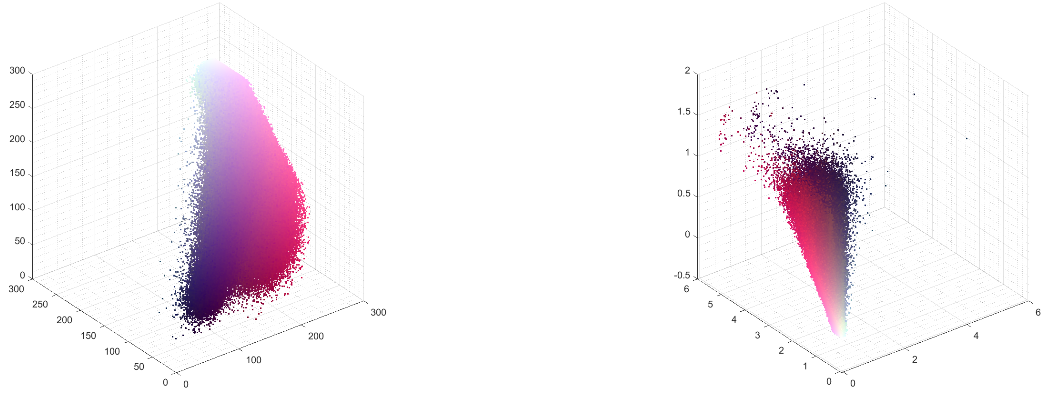
Stain	Hematoxylin	Eosin stain
Properties of Dye	Basic	Acidic
Dyed Color	Blue	Pink-red
Where the color appears	nucleus (basophilic)	cytoplasm (eosinophilic)

Table 3.1: H & E

3.2.1 H & E staining extract

Translate the RGB image to optical density (OD) value space, and divide it into the two primary colors by solving the singular value decomposition (SVD).

```
Algorithm H & E Stain Extraction
Set: $\alpha = 1$, $\beta = 0.15$, and $I_0 = 240$
Input: $RGB$ image $I$
Convert RGB image $I$ to OD density space $OD = -\log_{10}\left(\frac{I}{I_0}\right)$.
Remove data with $OD$ intensity less than $\beta$
$S = V^{-1}OD \Leftarrow OD = VS$
Project data onto the plane, and normalize to unit length
Calculate angle of each point wrt the first SVD direction
Find robust extremes ($\alpha^{th}$ and $(100-\alpha)^{th}$ percentiles of the angle)
Convert extreme values back to OD space
return norm, H, and E images
```

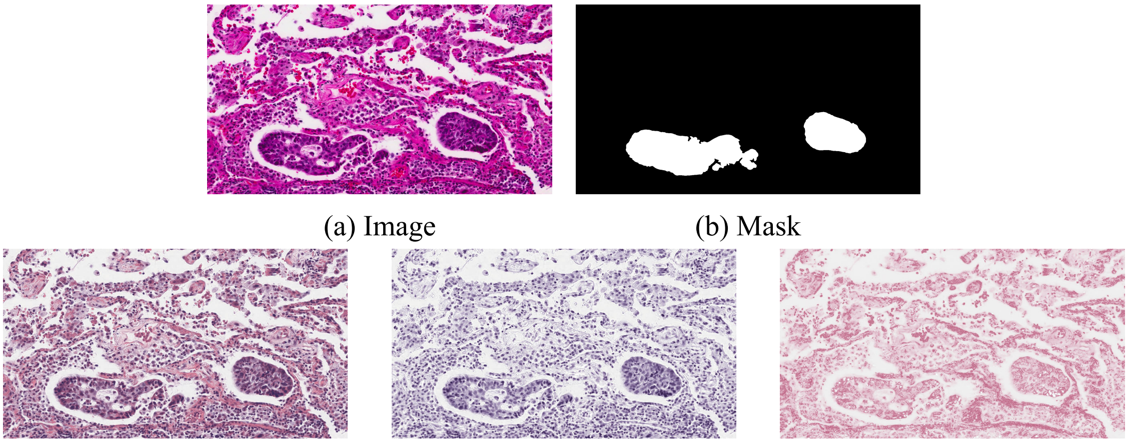


(a) RGB space

(b) OD space

Figure 3.2: RGB space vs OD space: The left is the RGB space, and the right is the OD space by observing the two spaces below. Converting an RGB image to OD (optical density) space is more efficiently represented by combining two principal vectors. We need to give a threshold and find the combination coefficients.

Here we take (a) norm image and (b) H image in training process. Let's look at the effect presented below. It's clear that nucleus present on H images. Moreover, there will be no overly vivid colors on the Norm image (compare to original image in id 297).



(a) Image

(b) Mask

(c) Norm Image

(d) H Image

(e) E Image

Figure 3.3: Augmentation data of 297.jpg

3.2.2 Auto-augmentation

The method of adding data based on reinforcement learning, adding an automatic augmentation strategy, establishes multiple sub-strategies under the strategy to make the data change in different color type or geometric type.

Here, we divide transform into two categories. One is for eight color transformations. The other is based on five geometric transformations. 2 of 13 are merged into sub-strategies, and 25 kinds of sub-strategies are combined. Select sub-policies with uniform probability in each mini-batch as the data augmentation method.

Color Space Transform		Geometry Transform
Brightness	Invert	Rotate
Color	Equalize	TranslateX
Contrast	Solarize	TranslateY
Autocontrast	Posterize	ShearX
		ShearY

Table 3.2: Auto-augmentation

3.2.3 Random Add Noise

To increase the generalization of our model, We add the noise with probability by $I(i, j) + \sigma \cdot s$, where $s \sim \mathcal{N}(0, 1)$ and I is the grayscale.

3.2.4 Random FlipLR or FlipUD

Random flip photos horizontally or vertically. Separated from auto augmentation to avoid doing two equivalents without making changes.

3.2.5 Padding

In order to be divisible by 32 and maintain the original image ratio, we pad it from a size of 1716×942 to 1728×960.

3.3 Model Architecture

In this task, we adopt a UNet based model. First, we may want to generalize the architecture of UNet to a abstract topological structure. As shown in the figure below, we may divide UNet architecture into three parts: Encoder, Bridge and Decoder. Therefore, in the next subsections, we will introduce them one by one.

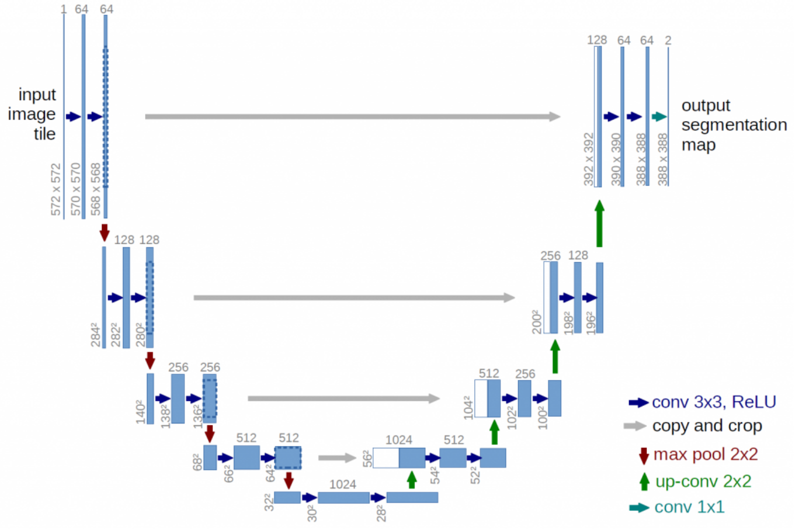


Figure 3.4: UNet architecture

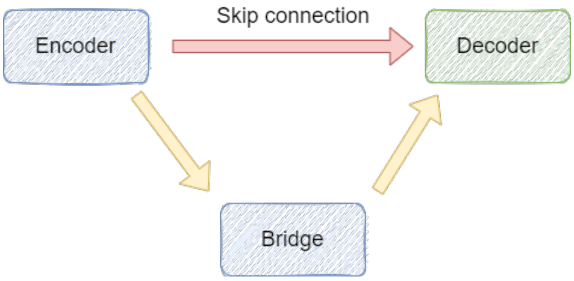


Figure 3.5: Abstract topological structure of UNet

3.3.1 Encoder and Bridge

The goal of the encoder is to extract useful features. However, it is not necessary to follow the encoder in the original UNet. Any CNN based networks (encoder) with down-sample steps can be used as an encoder. Therefore, we may choose pre-trained model as our encoder in order to get a better result. In this task, we choose ResNet [3] and EfficientNet [9] as our encoder.

ResNet [3] has been widely used as a pre-trained backbone. It adopted residual learning to get a faster convergence. As shown in Figure [3.6, 3.7]. ResNet is composed by residual blocks. Moreover, the deeper network will get a better performance. However, the deeper network will be time consuming, we only try ResNet34 and ResNet50 in this task.

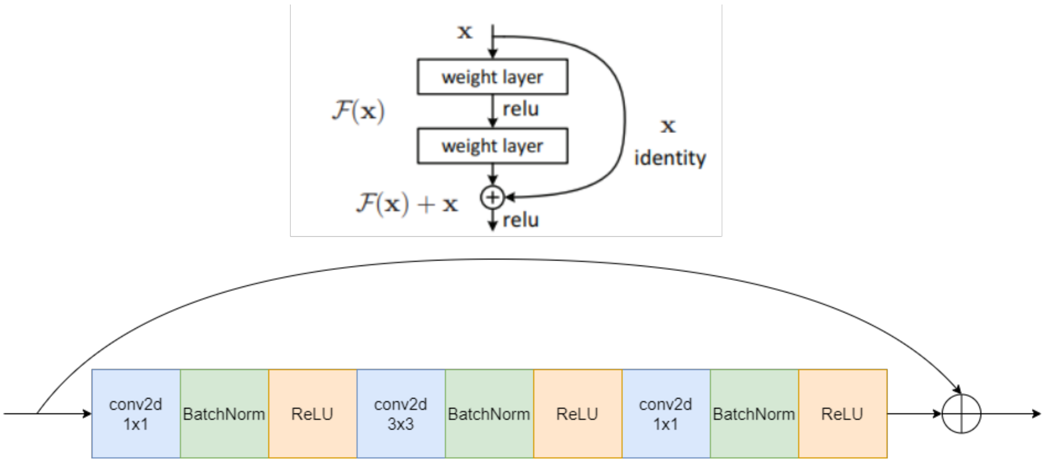
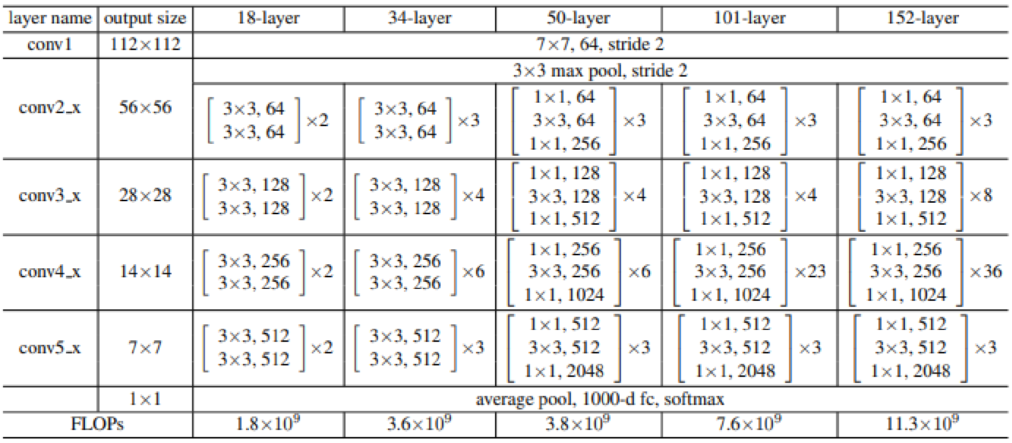


Figure 3.6: Residual learning



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.7: ResNet architecture

EfficientNet [9] is proposed in 2019. Scaling up convolution network is widely used to achieve better accuracy. For example, ResNet [3] scaling up the network's depth (ResNet18 to ResNet152). Wide Residual Network (WRN) [10] scaling up the network's width (increasing the number of output channels). Moreover, scaling up models by image resolution also gives a better accuracy too. The goal of EfficientNet is to scaling up depth/width/resolution simultaneously, as shown in Figure **??**. Intuitively, scaling up all of them makes sense. Once input image become larger, the model needs more layers to ensure the receptive fields is large enough and more channels to capture more features. Finally, EfficientNet gives a great accuracy, as shown in Figure **??**.

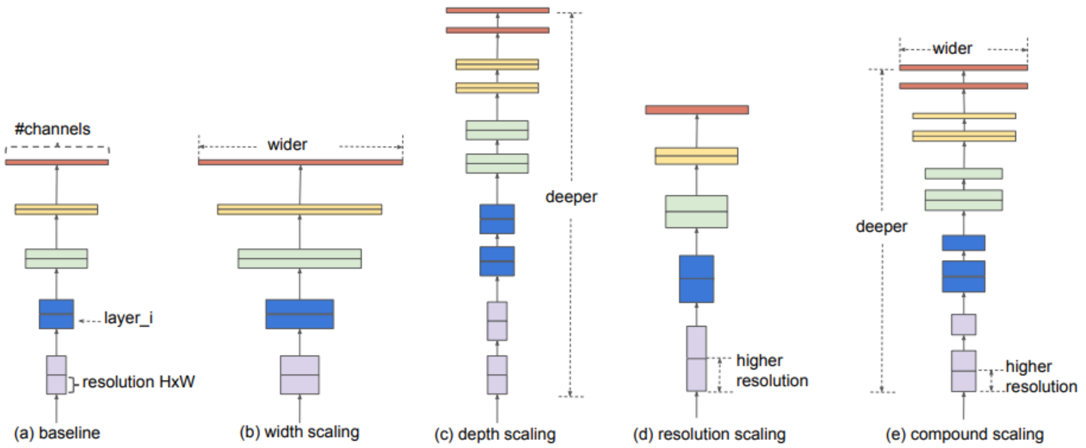
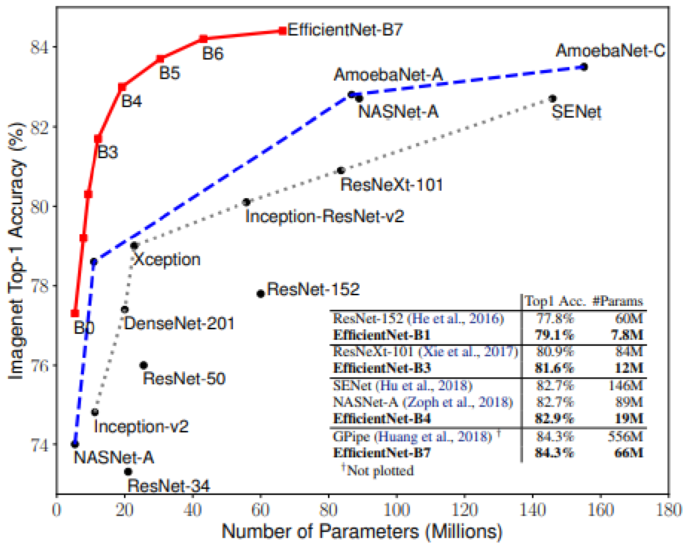


Figure 3.8: Model scaling



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure 3.9: EfficientNet accuracy

Both ResNet and Efficient has down-sampled five times, with the identity mapping, we got six stages here, see Figure 3.10. The last stage of ResNet and EfficientNet will be treated as the Bridge in Figure 3.5.

3.3.2 Decoder

For the decoder part, we follow the original UNet architecture. The only difference is that the transpose convolution is replaced by nearest interpolation.

Combining the Encoder, Bridge and Encoder that we mentioned above, the model architecture looks like the following Figure.

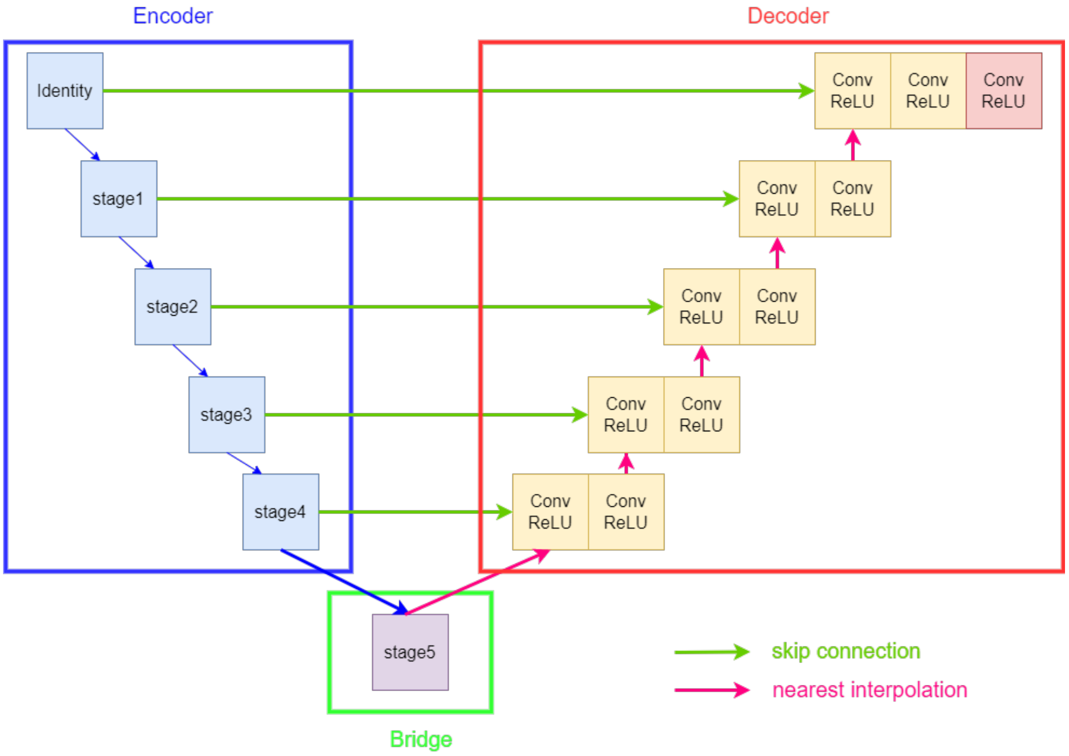


Figure 3.10: Model architecture

3.4 Loss Function

Let $Y, \hat{Y} \in \mathbb{R}^{m \times n}$ be the ground truth with one-hot label and prediction. The last dimension of Y and $\hat{Y}$ indicate the category. We use the linear combination with dice loss and focal loss as the segmentation loss.

$$\mathcal{L}_{Seg}(\hat{Y}, Y) = \mathcal{L}_{DSC}(\hat{Y}, Y) + \mathcal{L}_{FL}(\hat{Y}, Y)$$

We will introduce in detail as the following.

3.4.1 Dice Loss

The dice loss is defined as

$$\mathcal{L}_{DSC}(\hat{Y}, Y) = 1 - \frac{2 \cdot \sum_{i,j,c} Y_{i,j,c} \cdot \hat{Y}_{i,j,c}}{\sum_{i,j,c} Y_{i,j,c} + \sum_{i,j,c} \hat{Y}_{i,j,c}}$$

3.4.2 Focal Loss

The candidate object in a picture has most of the proportions of the background rather than the foreground, so there will be an imbalance in calculating the loss. The focal loss is down-weighted for the easy example. The hard example can be trained as much as possible during the training process and ignored those easy examples. So it can solve the category imbalanced problem. The focal loss [5] is defined as

$$\mathcal{L}_{FL}(\hat{Y}, Y) = -\sum_{i,j} \alpha \left(1 - \hat{Y}_{i,j,C}\right)^{\gamma} \log \hat{Y}_{i,j,C}$$

where C is category of $Y_{i,j}$ and $\alpha, \gamma \geq 0$ are hyper-parameter.

Why the focal loss can address data imbalance? We believe that the data which rarely appear would get the lower probability of ground class. On the contrary, the data which frequent appear would get the high probability of ground class. We will explain the reason why the focus loss can improve the data imbalance. First, we focus on the blue line and purple link in the Figure 3.11. The blue line is cross-entropy loss which is the special case of the focal loss. the purple line is focal loss with $\gamma = 2$. Compared with two lines, if we set 0.2, 0.6 to the probability of ground truth class, the blue line would get 1.6, 0.5 of the loss and the purple line would get 1.05, 0.05 of the loss. We discover the differences of the purple line are more significant by the following equation $\frac{1.6}{0.5} = 3.2 < 21 = \frac{1.05}{0.05}$ This shows the focal loss is able to highlight the data which perform worst, and it doesn't have to work hard on good performance data.

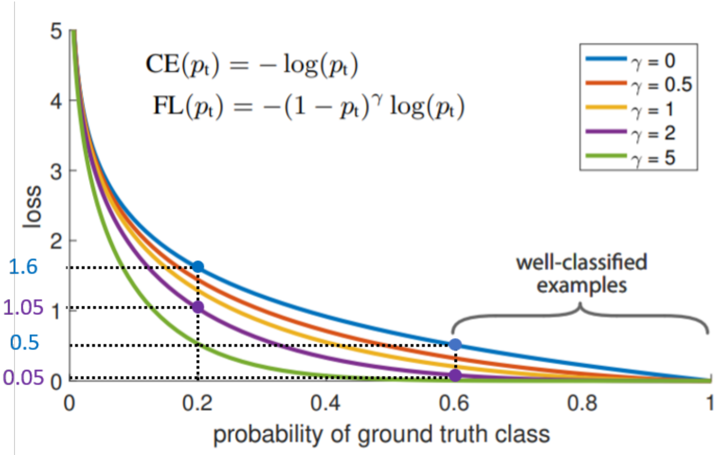


Figure 3.11: Focal loss

3.4.3 Deep Supervised

By computing loss functions on different decoder levels, deep supervision helps to improve gradient flow. Let Y_0, Y_1, Y_2 are outputs sequence of the last three decoder and Y_1, Y_2, Y_3 are downsampled using nearest neighbor interpolation to an half size of Y_0, Y_1, Y_2. The loss of deep supervised is defined as

$$\mathcal{L}_{DS} = \mathcal{L}(\hat{Y}_0, Y_0) + \frac{1}{4}\mathcal{L}(\hat{Y}_1, Y_1) + \frac{1}{16}\mathcal{L}(\hat{Y}_2, Y_2)$$

3.5 Optimization

We use AdamW as the optimizer. It is a stochastic optimization method that modifies the typical implementation of weight decay in Adam, by decoupling weight decay from the gradient update. We give the algorithm as the following:

Algorithm AdamW

```
Input: $f(\theta)$ (objective), $\gamma$ (lr), $\beta_1, \beta_2, \theta_0, \varepsilon, \lambda$ (weight decay)
Initial: $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment),
for $t = 1$ to ... do
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$
    $\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t}$
    $\theta_t \leftarrow \theta_{t-1} - \gamma \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$
end for
return $\theta_t$
```

3.6 Learning Rate Scheduler

We tried step and cosine learning scheduler, their rate of change was concentrated in different parts. The former with a high rate of change in the first 20 epochs and tends to be stable in the remaining periods. The latter is smooth in the front and tail, decreasing in the middle part.

- **Step decay:** Decay in each five step by 0.75 factor.
- **Cosine decay:** Use half of the cosine period to change the learning rate.

3.7 Metrics

In this task, the measurement indicator is evaluated on the mean dice coefficient. The Dice coefficient is equivalence to the f1-score. When applied to Boolean data, using the definition of true positive (TP), false positive (FP), and false negative (FN), it can be written as

$$DSC = \frac{2TP}{2TP + FP + FN}$$

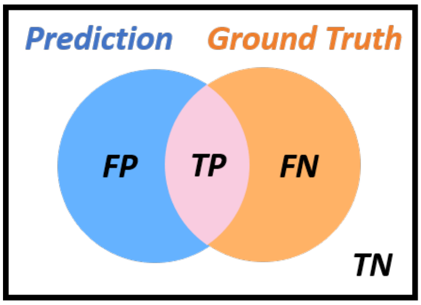


Figure 3.12: Venn diagram of prediction and ground truth

3.8 Post Processing

First, inference with test time augmentation (TTA) enabled will typically take about 2-3X the time of normal inference as the images are being up-down flipped, left-right flipped, and processed. Next, average the predictions of each corresponding image and return the probability mask. Final, we create an algorithm and convert the probability mask to a binary mask by analyzing it thoroughly and taking that as our final guess.

To make description clear, we call a connected component in non-zero area ”target” and call the union of connected targets ”mask” as shown in Figure 3.13.

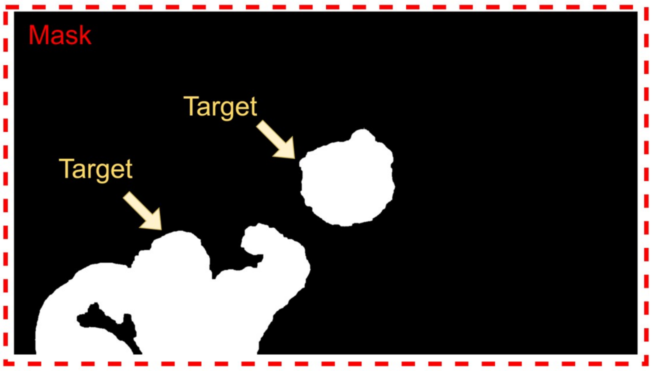
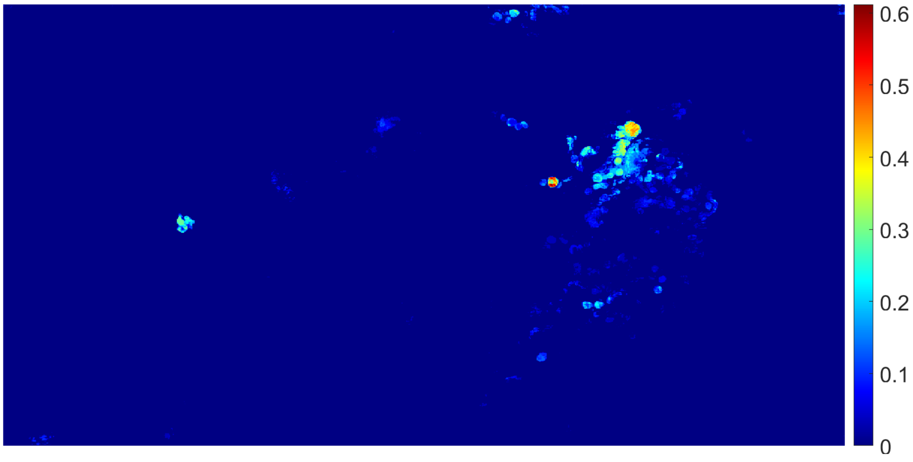


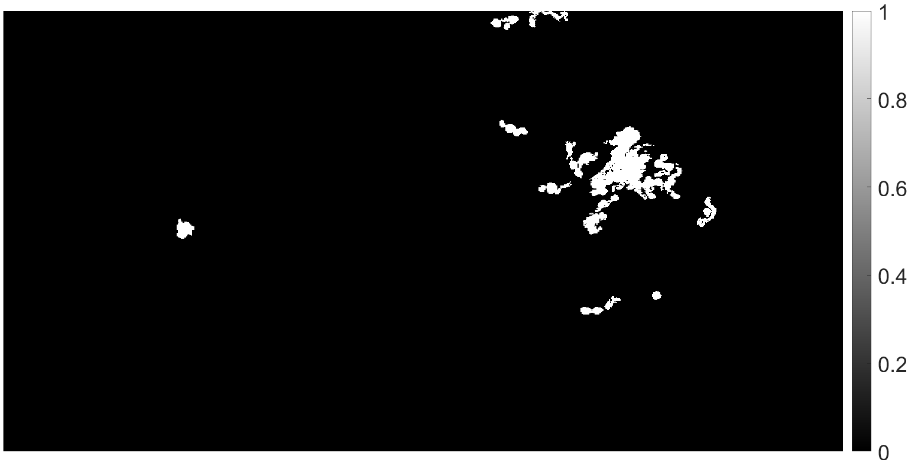
Figure 3.13: Predicted image

1. **Probability to Binary Selection:** Given some thresholds to decide the probability map’s value change to 0 or 1.
   (a) **Drop threshold:** Only retain those prob. bigger than it in prediction mask.
   (b) **High threshold:** Only concerned about those target which exist probability higher than it in prediction mask.
   (c) **Lower threshold:** According to previous interested target, screen out those higher than lower threshold pixel.

We present the probability mask and convert to binary mask image below Figure 3.14.



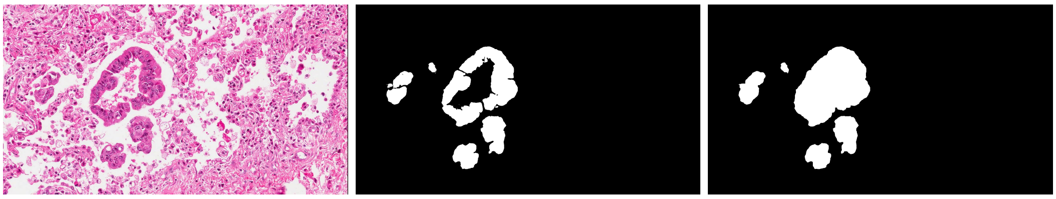
(a) Probability mask



(b) Binary mask

Figure 3.14: Convert probability mask to binary mask

2. **Morphological Image Processing:** Apply the image closing morphology, doing dilation and erosion in order to patch unconnected regions together and fill the holes. Example is shown in Figure 3.15. In (a) is an original image, (b) fill the hole without used image closing morphology, and (c) is with image closing.

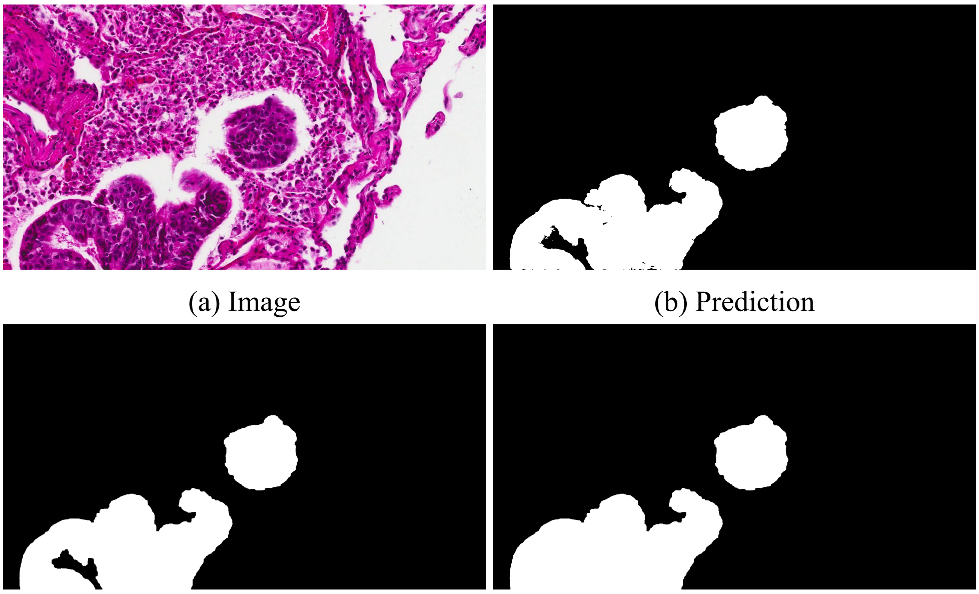


(a) Image (b) w/o post-process (c) w/ post-process

Figure 3.15: Comparison of Public_003.jpg

3. **Removal of the tiny targets:** Once filled in, check each target and remove targets with fewer than 250 target pixels in the mask.

4. **Padding and Filling:** To compensate for the same but unfillable boundaries as the target, we added the value of each edge of the bounding box and then patched it, which further enhanced the effect of post-processing. In (a) is original image, (b) is the model inference result, (c) is with morphological image process without padding edge, and (d) is with padding edge and fill the hole.



(a) Image

(b) Prediction

(c) w/o pad edge

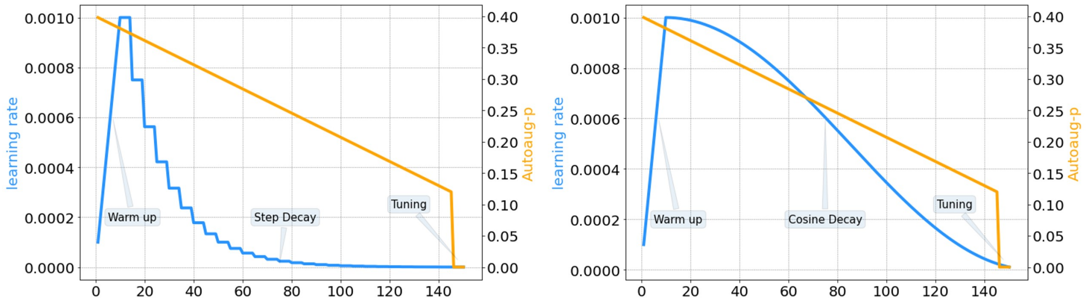
(d) w/ pad edge

Figure 3.16: Comparison of Public_100.jpg

Chapter 4

Experimental Analysis and Conclusion

We set the number of training steps to 150. The learning rate curve is shown below. The applied cosine decay and step decay are reduced by 0.75 every five steps, with warm-up used for the initial six steps. The auto-enhancement probability is initially set to 0.4. We calculate that approximately 30% of the expected data will execute the sub-policy in each epoch. In the early stage, let the model learn a lot of diverse data, reduce the trigger probability by increasing the number of steps, and reset the probability to zero in the last five steps, focusing on standard data for final fine-tuning.

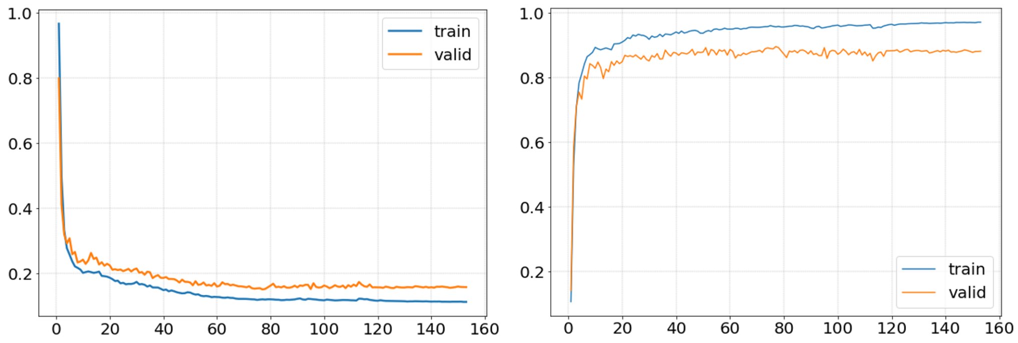


(a) Step decay

(b) Cosine decay

Figure 4.1: Learning rate schedule

After trying many encoders, we concluded that using the Efficient-B3 has better accuracy, and the loss value will also be decline steadily. In Table 4.1, we display our best model Hyperparameter.



(a) Loss

(b) Accuracy

Figure 4.2: Curve of loss and accuracy

Epoch	200
Batch size	48 (12 × 4)
Backbone	Efficient-B3
Activation function	Mish
Learning rate	0.001
Learning rate scheduler	cosine decay
Warm up epoch	6

Table 4.1: Hyperparameter

The goal of backbone is to extract the features from the image. In this task, our model uses UNet-based model and made a trial in a variety of backbone. We use the data augmentation, learning scheduler for training and implement the different backbones. For the activation function, we had tried the various function like LReLU, mish and swish but not change much. According to the information of Figure in previous, we set the threshold to eliminate the prediction mask which predict less than 250 pixels. Our experiments result display as Table 4.2.

Model	Backbone	Activation	Fold ID	Validation accuracy	Test accuracy
UNet	EfficientNet-b3	mish	0	0.90557	0.89706
UNet	EfficientNet-b3	mish	1	0.89629	0.89045
UNet	EfficientNet-b3	mish	2	0.89717	0.89919
UNet	EfficientNet-b3	mish	3	0.90199	0.89746
UNet	EfficientNet-b3	leaky relu	0	0.89005	0.90104
UNet	EfficientNet-b3	leaky relu	1	0.8777	0.90605
UNet	EfficientNet-b3	leaky relu	2	0.89861	0.90235
UNet	EfficientNet-b3	leaky relu	3	0.89216	0.90082

Table 4.2: Validation and test DSC

Conclusion. In this project, we implemented a UNet based model with different backbone. For the imbalance of foreground and background, we try to mitigate this problem by combining DiceLoss and Focal Loss. Moreover, we use and auto-augmentation increase the model robustness. For the post-processing, we design some strategy which improve the accuracy powerfully. In Table 4.3, we show the final score and rank on the leaderboard.

Team Name	Public			Private	Final Rank (total teams)
	Dice Score	Precision	Recall	Dice Score	
TEAM_1137	0.919385	0.927361	0.911546	0.910871	16 (307)

Table 4.3: TBrain ranking

Bibliography

[1] Jieneng Chen, Yongyi Lu, Qihang Yu, Xiangde Luo, Ehsan Adeli, Yan Wang, Le Lu, Alan Loddon Yuille, and Yuyin Zhou. Transunet: Transformers make strong encoders for medical image segmentation. *ArXiv*, abs/2102.04306, 2021.

[2] Steven Guan, Amir A. Khan, Siddhartha Sikdar, and Parag V. Chitnis. Fully dense unet for 2-d sparse photoacoustic tomography artifact removal. *IEEE Journal of Biomedical and Health Informatics*, 24(2):568–576, 2020.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[4] Fabian Isensee, Paul F. Jager, Simon A. A. Kohl, Jens Petersen, and Klaus Maier-Hein. Automated design of deep learning methods for biomedical image segmentation. *arXiv: Computer Vision and Pattern Recognition*, 2019.

[5] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[7] Marc Macenko, Marc Niethammer, J. S. Marron, David Borland, John T. Woosley, Xiaojun Guan, Charles Schmitt, and Nancy E. Thomas. A method for normalizing histology slides for quantitative analysis. *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1107–1110, 2009.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[9] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[10] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

Appendix A

Environment

Operating System

We develop the code on various OS, such as Windows10 and Linux (CentOS Linux release 7.8.2003, Red Hat release 6.4).

Language

For programming language, we implement our method by using MATLAB (r2021b) and python (3.8.5).

Pre-processing	MATLAB
Training	Python
Post-processing	MATLAB

Table A.1: Language

Third Party Libraries

We have used third party libraries only python, we have used the following third-party libraries:

torch	1.11.0 +cu113
torchaudio	0.11.0+cu113
monai	0.8.1
timm	0.4.12
numpy	1.19.3
matplotlib	3.3.3
Pillow	8.1.0
PyYaml	6.0
yacs	0.1.8
tqdm	4.55.1

Table A.2: Version of libraries

Pretrained Model

We apply pretrained Efficient (B0 to B3) in timm package as our model backbone.

Appendix B

Cloud Calculation

We use WinSCP[1] to manage data, monitor the data stored in the \home or \work folders, and download the completed data to the local end. The container selects the **Interactive Container** with the **Image File** `pytorch22.02-py3:latest`, and the **container computing type** is `c.2xsuper` to provide 4GPU + 16CPU 360GB Memory. Here are the following five-step we set up to make insurance that there is no wrong and waste while applying cloud server.

1. **Set up container:** Set up a development container, install the required packages, save it as a copy, and use the Custom Image File for the new container to complete the internal settings (Subsequent containers directly apply a copy).

2. **Set up monitoring interface:** Use tmux[2] to split the screen to record execution, GPU usage, and background execution in a single window. (Because the pricing starts after opening the container, it is necessary to ensure that every second of the use period will not be wasted, so we will also estimate the time to close the container within 3 minutes after completing the training).

3. **Prepare pre-written scripts:** Execute pre-written scripts through bash first, and use the background execution method to ensure that disconnection will not cause unnecessary waste in the middle. (example below)

```
#!/bin/bash
nohup python3 STEP3_Train.py --config
    B3-batch-adamw-step/Fold0.yaml > batchadamwfold0.txt &
```

4. **Confirm correct execution:** The inspection procedure is as follows.

   (a) Tracking output information in the left half of the window. (`tail -f filename`)

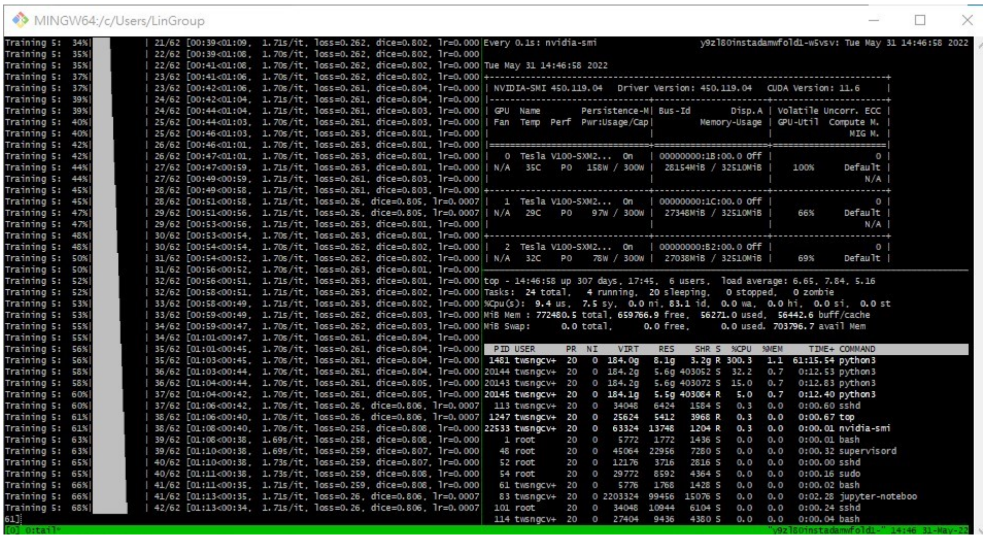
[1] WinSCP (Windows Secure Copy) is a free and open-source SSH File Transfer Protocol (SFTP), File Transfer Protocol (FTP), and secure copy protocol (SCP) client for Microsoft Windows.(from wiki)

[2] tmux is an open-source terminal multiplexer for Unix-like operating systems.(from wiki).

(b) The GPU usage in the upper right corner is correct.

(c) Enter the \home directory from WinSCP, and check whether the model parameter, record, and configuration information files are stored correctly in the checkpoint folder.

(d) Observe the execution command in the real-time monitoring system load status (`top`). The assigned command appears in the lower right block.



```
MINGW64:/c/Users/LinGroup
Training 5:  34%|███     | 21/62 [00:39<01:09,  1.71s/it, loss=0.262, dice=0.802, lr=0.000|Every 0.1s: nvidia-smi                    y9zl80instadamwfold1-w5vsv: Tue May 31 14:46:58 2022
Training 5:  35%|███     | 22/62 [00:39<01:08,  1.70s/it, loss=0.262, dice=0.802, lr=0.000|
Training 5:  35%|███     | 22/62 [00:41<01:08,  1.70s/it, loss=0.262, dice=0.802, lr=0.000|Tue May 31 14:46:58 2022
Training 5:  37%|███     | 23/62 [00:41<01:06,  1.70s/it, loss=0.262, dice=0.802, lr=0.000|+-----------------------------------------------------------------------------+
Training 5:  37%|███     | 23/62 [00:42<01:06,  1.70s/it, loss=0.261, dice=0.804, lr=0.000|| NVIDIA-SMI 450.119.04   Driver Version: 450.119.04   CUDA Version: 11.6     |
Training 5:  39%|███     | 24/62 [00:42<01:04,  1.71s/it, loss=0.261, dice=0.804, lr=0.000||-------------------------------+----------------------+----------------------+
Training 5:  39%|███     | 24/62 [00:44<01:04,  1.71s/it, loss=0.261, dice=0.803, lr=0.000|| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
Training 5:  40%|███     | 25/62 [00:44<01:03,  1.70s/it, loss=0.261, dice=0.803, lr=0.000|| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
Training 5:  40%|███     | 25/62 [00:46<01:03,  1.70s/it, loss=0.263, dice=0.801, lr=0.000||                               |                      |               MIG M. |
Training 5:  42%|███     | 26/62 [00:46<01:01,  1.70s/it, loss=0.263, dice=0.801, lr=0.000||===============================+======================+======================|
Training 5:  42%|███     | 26/62 [00:47<01:01,  1.70s/it, loss=0.263, dice=0.801, lr=0.000||   0  Tesla V100-SXM2...  On   | 00000000:1B:00.0 Off |                    0 |
Training 5:  44%|███     | 27/62 [00:47<00:59,  1.71s/it, loss=0.263, dice=0.801, lr=0.000|| N/A   35C    P0   158W / 300W |  28154MiB / 32510MiB |    100%      Default |
Training 5:  44%|███     | 27/62 [00:49<00:59,  1.71s/it, loss=0.261, dice=0.803, lr=0.000||                               |                      |                  N/A |
Training 5:  45%|███     | 28/62 [00:49<00:58,  1.71s/it, loss=0.261, dice=0.803, lr=0.000|+-------------------------------+----------------------+----------------------+
Training 5:  45%|███     | 28/62 [00:51<00:58,  1.71s/it, loss=0.26, dice=0.805, lr=0.0007||   1  Tesla V100-SXM2...  On   | 00000000:1C:00.0 Off |                    0 |
Training 5:  47%|███     | 29/62 [00:51<00:56,  1.71s/it, loss=0.26, dice=0.805, lr=0.0007|| N/A   29C    P0    97W / 300W |  27348MiB / 32510MiB |     66%      Default |
Training 5:  47%|███     | 29/62 [00:53<00:56,  1.71s/it, loss=0.263, dice=0.801, lr=0.000||                               |                      |                  N/A |
Training 5:  48%|███     | 30/62 [00:53<00:54,  1.70s/it, loss=0.263, dice=0.801, lr=0.000|+-------------------------------+----------------------+----------------------+
Training 5:  48%|███     | 30/62 [00:54<00:54,  1.70s/it, loss=0.262, dice=0.802, lr=0.000||   2  Tesla V100-SXM2...  On   | 00000000:B2:00.0 Off |                    0 |
Training 5:  50%|███     | 31/62 [00:54<00:52,  1.70s/it, loss=0.262, dice=0.802, lr=0.000|| N/A   32C    P0    78W / 300W |  27038MiB / 32510MiB |     69%      Default |
Training 5:  50%|███     | 31/62 [00:56<00:52,  1.70s/it, loss=0.263, dice=0.801, lr=0.000|────────────────────────────────────────────────────────────────────────────────────────
Training 5:  52%|███     | 32/62 [00:56<00:51,  1.71s/it, loss=0.263, dice=0.801, lr=0.000|top - 14:46:58 up 307 days, 17:45,  6 users,  load average: 6.65, 7.84, 5.16
Training 5:  52%|███     | 32/62 [00:58<00:51,  1.71s/it, loss=0.263, dice=0.802, lr=0.000|Tasks:  24 total,   4 running,  20 sleeping,   0 stopped,   0 zombie
Training 5:  53%|███     | 33/62 [00:58<00:49,  1.71s/it, loss=0.263, dice=0.802, lr=0.000|%Cpu(s):  9.4 us,  7.5 sy,  0.0 ni, 83.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
Training 5:  53%|███     | 33/62 [00:59<00:49,  1.71s/it, loss=0.262, dice=0.803, lr=0.000|MiB Mem : 772480.5 total, 659766.9 free,  56271.0 used,  56442.6 buff/cache
Training 5:  55%|███     | 34/62 [00:59<00:47,  1.70s/it, loss=0.262, dice=0.803, lr=0.000|MiB Swap:      0.0 total,      0.0 free,      0.0 used. 703796.7 avail Mem
Training 5:  55%|███     | 34/62 [01:01<00:47,  1.70s/it, loss=0.261, dice=0.804, lr=0.000|
Training 5:  56%|███     | 35/62 [01:01<00:45,  1.70s/it, loss=0.261, dice=0.804, lr=0.000|  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
Training 5:  56%|███     | 35/62 [01:03<00:45,  1.70s/it, loss=0.261, dice=0.804, lr=0.000| 1481 twsngcv+  20   0  184.0g   8.1g   3.2g R 300.3   1.1  61:15.54 python3
Training 5:  58%|███     | 36/62 [01:03<00:44,  1.70s/it, loss=0.261, dice=0.804, lr=0.000|20144 twsngcv+  20   0  184.2g   5.6g 403052 S  32.2   0.7   0:12.53 python3
Training 5:  58%|███     | 36/62 [01:04<00:44,  1.70s/it, loss=0.261, dice=0.805, lr=0.000|20143 twsngcv+  20   0  184.2g   5.6g 403072 S  15.0   0.7   0:12.83 python3
Training 5:  60%|███     | 37/62 [01:04<00:42,  1.70s/it, loss=0.261, dice=0.805, lr=0.000|20145 twsngcv+  20   0  184.1g   5.5g 403084 R   5.0   0.7   0:12.40 python3
Training 5:  60%|███     | 37/62 [01:06<00:42,  1.70s/it, loss=0.26, dice=0.806, lr=0.0007|  113 twsngcv+  20   0   34048   6424   1584 S   0.3   0.0   0:00.60 sshd
Training 5:  61%|███     | 38/62 [01:06<00:40,  1.70s/it, loss=0.26, dice=0.806, lr=0.0007| 1247 twsngcv+  20   0   25624   5412   3968 R   0.3   0.0   0:00.67 top
Training 5:  61%|███     | 38/62 [01:08<00:40,  1.70s/it, loss=0.258, dice=0.808, lr=0.000|22533 twsngcv+  20   0   63324  13748   1204 R   0.3   0.0   0:00.01 nvidia-smi
Training 5:  63%|███     | 39/62 [01:08<00:38,  1.69s/it, loss=0.258, dice=0.808, lr=0.000|    1 root      20   0    5772   1772   1436 S   0.0   0.0   0:00.01 bash
Training 5:  63%|███     | 39/62 [01:10<00:38,  1.69s/it, loss=0.259, dice=0.807, lr=0.000|   48 root      20   0   45064  22956   7280 S   0.0   0.0   0:00.32 supervisord
Training 5:  65%|███     | 40/62 [01:10<00:38,  1.73s/it, loss=0.259, dice=0.807, lr=0.000|   52 root      20   0   12176   3716   2816 S   0.0   0.0   0:00.00 sshd
Training 5:  65%|███     | 40/62 [01:11<00:38,  1.73s/it, loss=0.259, dice=0.808, lr=0.000|   54 root      20   0   29772   8592   4364 S   0.0   0.0   0:00.16 sudo
Training 5:  66%|███     | 41/62 [01:11<00:35,  1.71s/it, loss=0.259, dice=0.808, lr=0.000|   61 twsngcv+  20   0    5776   1768   1428 S   0.0   0.0   0:00.02 bash
Training 5:  66%|███     | 41/62 [01:13<00:35,  1.71s/it, loss=0.26, dice=0.806, lr=0.0007|   83 twsngcv+  20   0 2203324  99456  15076 S   0.0   0.0   0:02.28 jupyter-noteboo
Training 5:  68%|███     | 42/62 [01:13<00:34,  1.71s/it, loss=0.26, dice=0.806, lr=0.0007|  101 root      20   0   34048  10944   6104 S   0.0   0.0   0:00.24 sshd
61]                                                                                          |  114 twsngcv+  20   0   27404   9436   4380 S   0.0   0.0   0:00.04 bash
[0] 0:tail*                                                                                             "y9zl80instadamwfold1-" 14:46 31-May-22
```

Figure B.1: tmux interface

5. **Collect training results:**

(a) Track whether the output information stops outputting content.

(b) The GPU usage of the upper right window returns to 0.

(c) Make sure that the log file of the \home data is fully stored.

(d) Download the log file to your local computer.

(e) After the above confirmation, close the container and stop deducting the calculation resource points.

After the standardized execution steps are determined, eight containers are executed at the same time according to the process in Figure ?, and two groups of 4-fold cross-validation are trained so that they can be handed over to the machine for execution without fail.

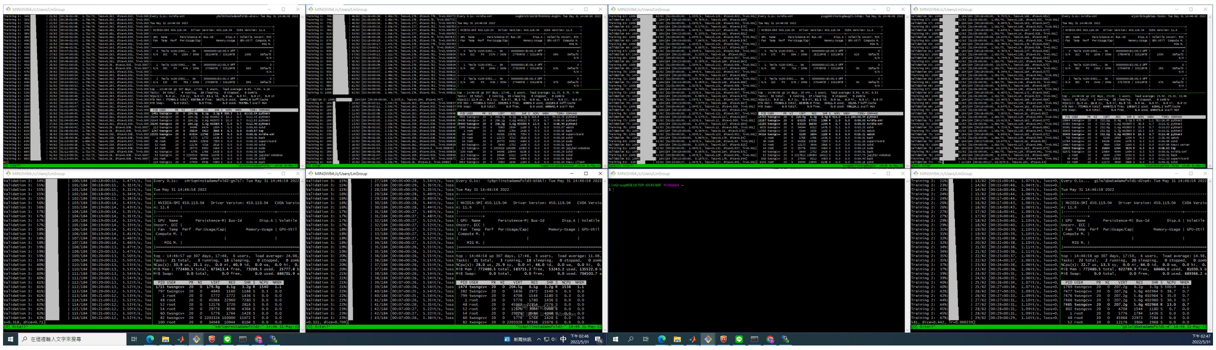


Figure B.2: tmux view

B.1 Our Experience

1. Since it is the first time to use the TWCC online resources, one person is mainly responsible for the overall operation, and the part of the shared memory is not used. The advantage is that the data is integrated into and out of the same host account, and it will be further used when there are more opportunities to use it in the future. Try using multiple accounts to use computing resources and online integration.

2. Since it is impossible to monitor the GPU usage of containers real-time simultaneously in TWCC, we turn to tmux to simultaneously render on the screen.

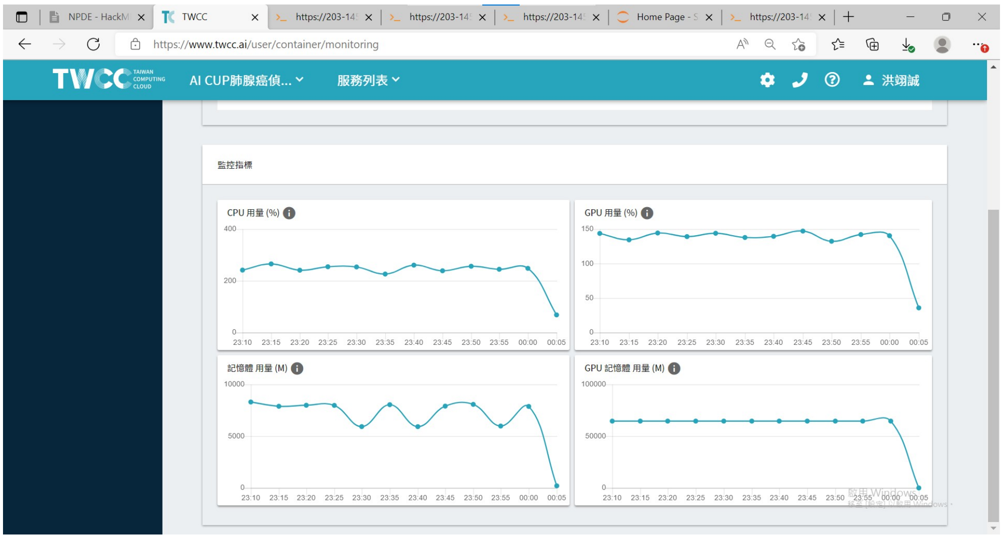


Figure B.3: GPU monitor in TWCC website

The screen only shows the GPU time usage curve of a specific container, and we are more concerned about the timely use of information so that we will choose the previous tmux monitoring method.

3. Overall computing resource usage is approximately 34,578.

Appendix C

Contact Information

- **Team**

Team Name	Private leaderboard score	Private leaderboard rank
TEAM_1137	0.910871	16

- **Team Member**

Name	Institution	Phone	Email
廖家緯 (Jia-Wei Liao)	NYCU AM	0939580280	sam23582211@gmail.com
吳國棟 (Kuok-Tong Ng)	NYCU AM	0910306618	l19so18h13k10@gmail.com
洪翊誠 (Yi-Cheng Hung)	NYCU AM	0958618583	yicheng.sc10@nycu.edu.tw

- **Advisor**

Advisor Name	Course (Course Number)	Institution	Email
Wen-Wei Lin	-	NYCU AM	wwlin@g2.nctu.edu.tw
Yuh-Jye Lee	Machine Learning (IAM5816)	NYCU AM	yuhjye@math.nctu.edu.tw