Technical Report

Yang Yixin and Bai Jingqi Team: Clay pot rice is being cooked

Data preparation:

Our preparation for the game data can be roughly divided into two stages: the preliminary and the semi-finals:

1. In the preliminary stage, we used standard 2000 training set images for training. We evaluate the mean and standard deviation of the training set and the test set, and normalize the training data using the mean and standard deviation of all population images, with the specific values being mean 0.4044 and standard deviation 0.1550 (according to the requirements in the Albumentations .Normalize function, the value is the actual value divided by 255), and standardize the input image at the last step of data augmentation.
2. In the rematch phase, we used standard 900 +2100 training set images for training. Among them, the data of 2100 images comes from the pseudo-labels of fine-tuning model inference: Specifically, we initialize an identical network using the model weights of the preliminary round, and then fine-tune it on 900 labeled data to obtain a new network model for inference pseudo-labels. After obtaining the mixed label data of 900+2100 with 12 fine-tuned models, we trained the rematch network model in a complete manner, and re-inferred the pseudo-labels of 2100 data based on the submitted optimal result threshold. After obtaining the 2100 pseudo-labels after iteration, we made a slight fine-tuning of them, mainly using the mask (320,640) of the shape shown in the figure below to multiply the original image, and eliminated some areas on both sides (:30,610:) and checked and fine-tuned the remaining areas. Specifically, we can see the relevant dataset files in uploading docker.

Figure 1 Pseudo-tagging Mask

We also used a similar method to the preliminary round to evaluate the mean and standard deviation of all image data and used it as a normalized numerical benchmark. The specific mean and standard deviations in the rematch phase are 0.408 and 0.163 respectively.

It is worth noting that as we approached the end of the game, we found that the mean and standard deviation calculated in the second phase were omitted and did not completely include all images. In our other previous similar competitions, Vesuvius ~~Challenge - Ink Detection | Kaggle, sligh~~t differences in mean and standard deviations may cause metric performance differences on the order of 1e-3, so we have doubts about which performance is better and what we use when we actually train our model.

Algorithm model:

Due to similar practices in Kaggle competitions, when building algorithm models, we chose the M ONAI library as the baseline library for us to build and improve the basic model of the algorithm. Our component model is based on SwinUNETR[i][1] and has made many explorations based on it. T he infrastructure of the original SwinUNETR network is as follows:

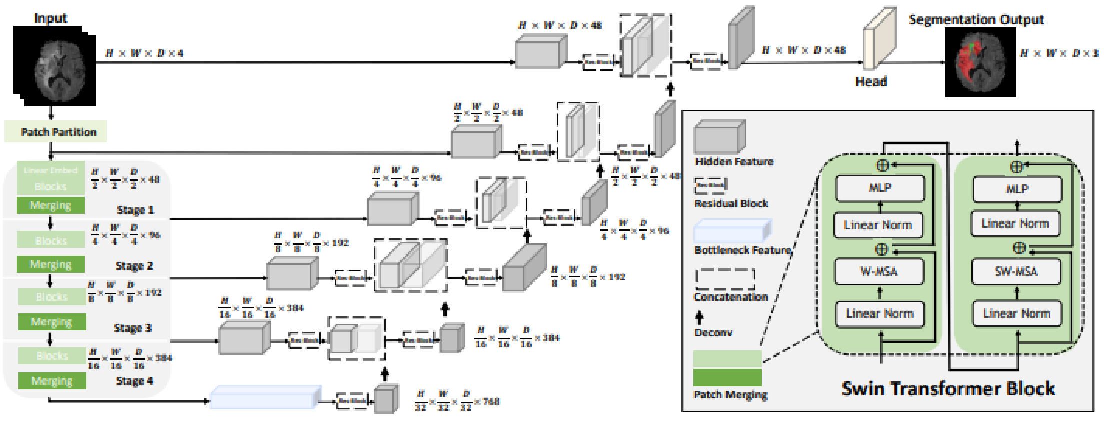


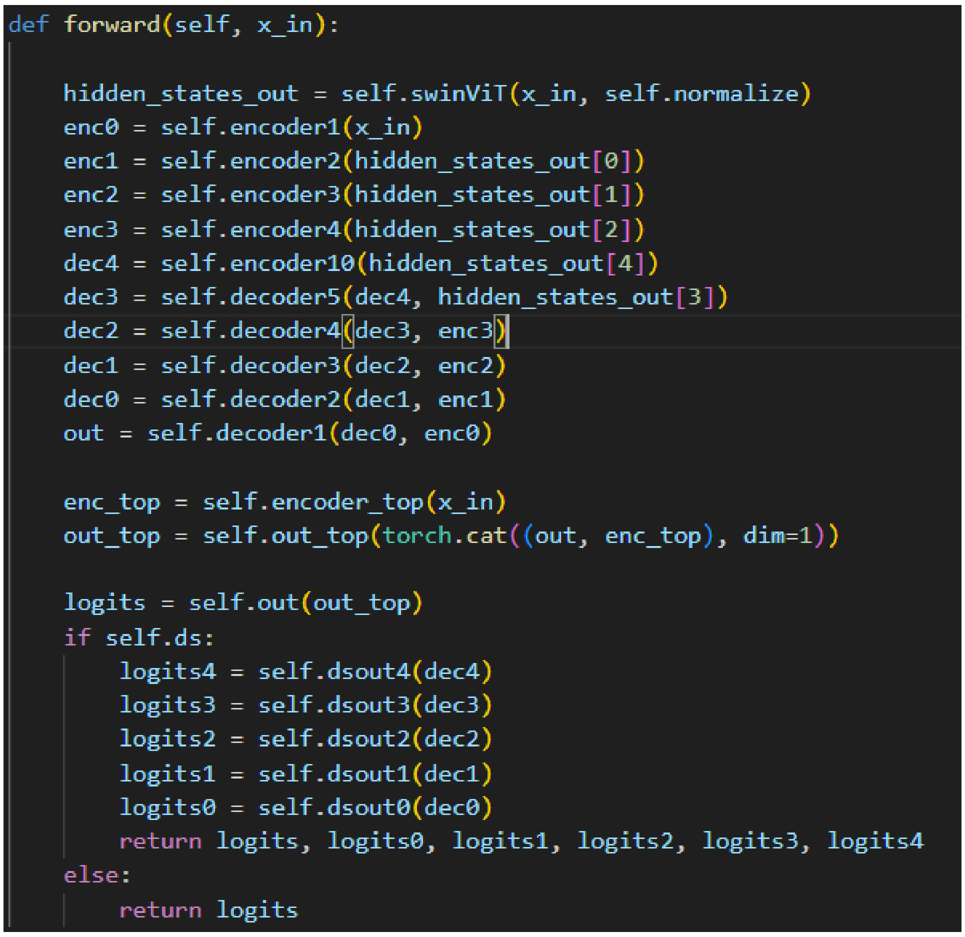
Figure 2. SwinUNETR architecture

The network improvement directions we have tried include:

1. Increase the scale of BackBone and embed the sub-model in the forward process to form an alg orithm network similar to the two-stage ensemble model +.

2. Increase deep supervision and apply supervision signals to every step in the upsampling process. 3. Add channel attention mechanis m at specific layers.
4. Inspired by PointRend's idea, an additional feature extraction module is designed to extract feat ures from the input image and supplement the connection to the final output module.

In our limited practice, Final Option 4 shows an improvement worthy of additional attention. Ther efore, in the final stage of the preliminary round and the semi-finals, we used Solution 4 as the gui de to design and improve the algorithm.
In the practice of the preliminary round, we modified the first version of the SwinUNETR network, which is reflected in swin_unetr_sim.py in monai.networks.nets in the submission dependency file. We added a 1x1 volume "encoder_top" to extract low-dimensional information on the feature scale of Input, and fuse it with the upsampled output on the output path "out_top". The forward process c ode is as follows:



```python
def forward(self, x_in):

    hidden_states_out = self.swinViT(x_in, self.normalize)
    enc0 = self.encoder1(x_in)
    enc1 = self.encoder2(hidden_states_out[0])
    enc2 = self.encoder3(hidden_states_out[1])
    enc3 = self.encoder4(hidden_states_out[2])
    dec4 = self.encoder10(hidden_states_out[4])
    dec3 = self.decoder5(dec4, hidden_states_out[3])
    dec2 = self.decoder4(dec3, enc3)
    dec1 = self.decoder3(dec2, enc2)
    dec0 = self.decoder2(dec1, enc1)
    out = self.decoder1(dec0, enc0)

    enc_top = self.encoder_top(x_in)
    out_top = self.out_top(torch.cat((out, enc_top), dim=1))

    logits = self.out(out_top)
    if self.ds:
        logits4 = self.dsout4(dec4)
        logits3 = self.dsout3(dec3)
        logits2 = self.dsout2(dec2)
        logits1 = self.dsout1(dec1)
        logits0 = self.dsout0(dec0)
        return logits, logits0, logits1, logits2, logits3, logits4
    else:
        return logits
```

Figure 3 SwinUNETR_Sim's forward process

In the practice of the rematch, we rewritten the top-level feature extraction module according to the previous improvement direction, and at the same time deepened the process after fusion. Specifically, we used 13x13 convolution for encoder_top, with a convolution step size of 1. Although using this convolution parameter will produce a larger padding, due to the characteristics of this competition dataset - that is, the foreground area is set in the center of the image, we believe that the large padding at the edge of the image will not have a major negative impact. At the same time, we deepened the module after the low-dimensional feature is fused with high-dimensional upsampled features, and added a standard MONAI UnetrBasicBlock module with residual connections in order to better integrate the two pieces of information. This version of the modified network is reflected in the swin_unetr_hs.py of monai.networks.nets in the submission file. The final network structure diagram is as follows:

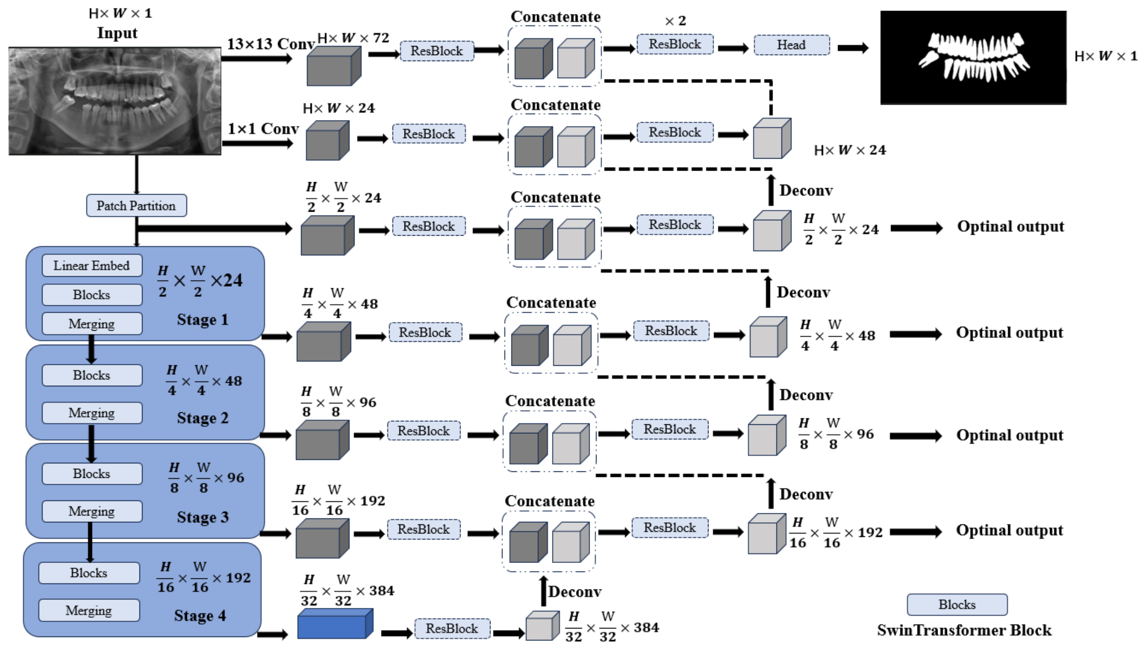
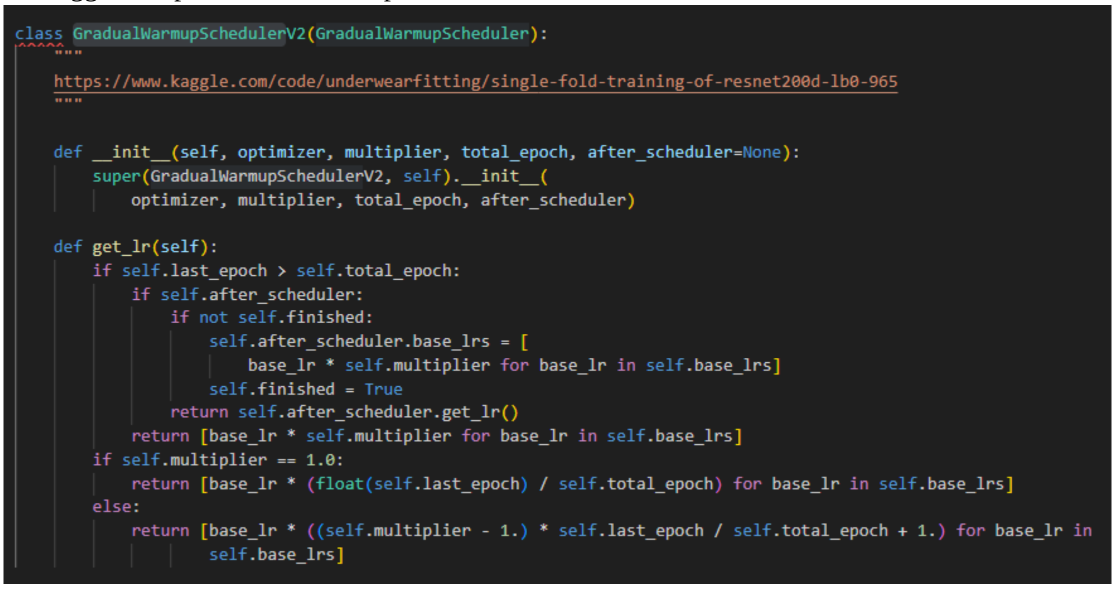


Figure 4 SwinUNETR_HS structure diagram

Training process:

Our training process is relatively ordinary, roughly as follows:

1. Divided. In the preliminary competition, we trained based on the 2,000 training set images provi ded by the official, and divided the 2,000 images into 50% off. Each change is taken as the training set and 400 as the verification set. On this basis, we trained different models with 50% off. At the s ame time, we also train a model that uses all data at a discount, so during the entire training proces s, we will train six network models for subsequent inferences to be introduced. In the rematch, we trained based on the 3,000 training set images provided by the official, and divided the 900 official annotated images into 50% off. Each discount was 720 as the training set and 180 as the verificatio n set. Therefore, the single-fold training images during the rematch were 2820, and the verification images were 180. 2. Preheat. We followed the GradualWarmupSchedulerV2 that was in front of t he Kaggle competition, with the specific code as follows:



```
class GradualWarmupSchedulerV2(GradualWarmupScheduler):
    """
    https://www.kaggle.com/code/underwearfitting/single-fold-training-of-resnet200d-lb0-965
    """

    def __init__(self, optimizer, multiplier, total_epoch, after_scheduler=None):
        super(GradualWarmupSchedulerV2, self).__init__(
            optimizer, multiplier, total_epoch, after_scheduler)

    def get_lr(self):
        if self.last_epoch > self.total_epoch:
            if self.after_scheduler:
                if not self.finished:
                    self.after_scheduler.base_lrs = [
                        base_lr * self.multiplier for base_lr in self.base_lrs]
                    self.finished = True
                return self.after_scheduler.get_lr()
            return [base_lr * self.multiplier for base_lr in self.base_lrs]
        if self.multiplier == 1.0:
            return [base_lr * (float(self.last_epoch) / self.total_epoch) for base_lr in self.base_lrs]
        else:
            return [base_lr * ((self.multiplier - 1.) * self.last_epoch / self.total_epoch + 1.) for base_lr in
                    self.base_lrs]
```

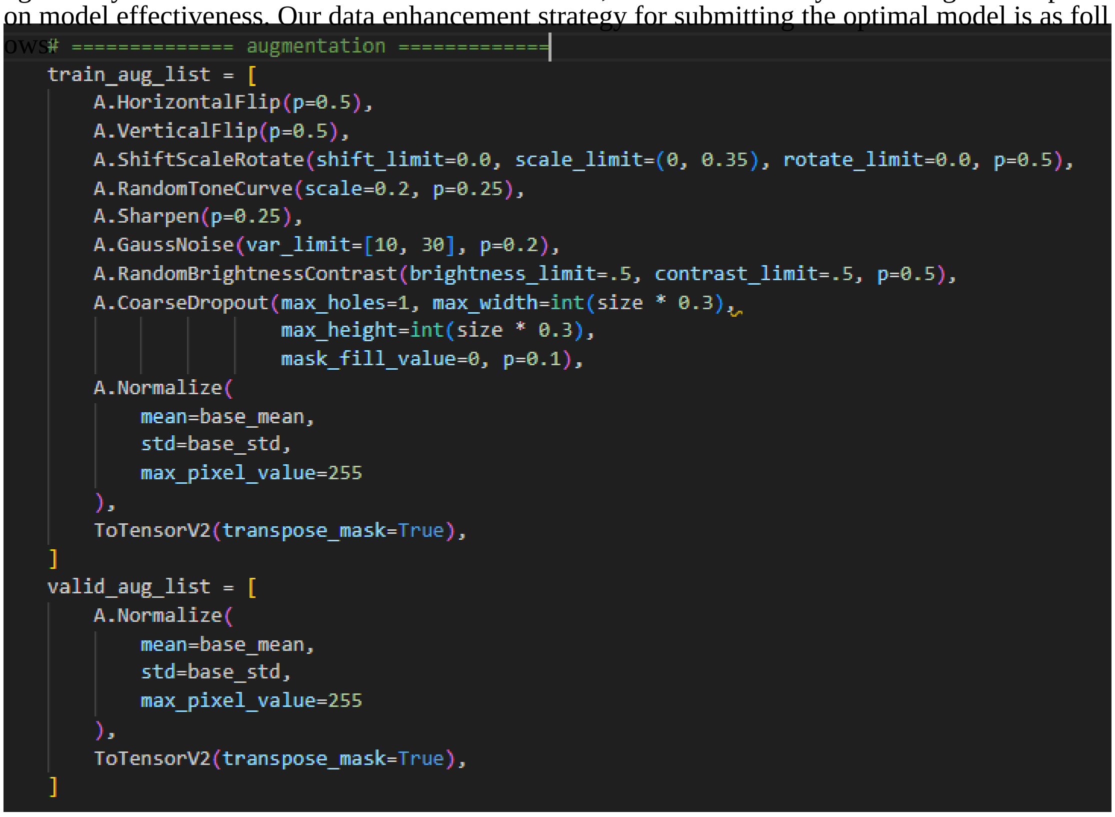
Figure 5 WarmUp process

3. Optimizer and loss function. We used AdamW as the optimizer for model training and as the de cay process for learning rate, we used torch.optim.lr_scheduler.CosineAnnealingLR. The basic lea rning rate is 4e-4 and the minimum learning rate is 6e-5. On the loss function, we use BCE with p os_weight of 2.0 and Dice loss in MONAI as the set loss function.

4. Training. We used a single NVIDIA A100 80GB for model training. The training BatchSize is 24 and the training rounds is 72. During the training process, we use automatic mixing accuracy to save memory footprint.
5. Model evaluation. Referring to the given scoring formula, we used DiceMetric, HausdorffDistan ceMetric, and MeanIoU in the MONAI library to build a similar scoring standard during model trai ning, and used it as the scoring standard guarantee model during verification. However, in practice, we found that the self-constructed scoring formula may differ from the official scoring formula indi cators. For example, the optimal threshold during local evaluation is often different from that durin g online submission, so local verification scores can often only be used as a rough evaluation of the model effect.

Data Enhancement:

We have tried a variety of data enhancement combination strategies, all derived from the accounti ngs library. We have tried radical data enhancements, but found that they have a negative impact on model effectiveness. Our data enhancement strategy for submitting the optimal model is as foll ows



```
# ============== augmentation ============
    train_aug_list = [
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),
        A.ShiftScaleRotate(shift_limit=0.0, scale_limit=(0, 0.35), rotate_limit=0.0, p=0.5),
        A.RandomToneCurve(scale=0.2, p=0.25),
        A.Sharpen(p=0.25),
        A.GaussNoise(var_limit=[10, 30], p=0.2),
        A.RandomBrightnessContrast(brightness_limit=.5, contrast_limit=.5, p=0.5),
        A.CoarseDropout(max_holes=1, max_width=int(size * 0.3),
                        max_height=int(size * 0.3),
                        mask_fill_value=0, p=0.1),
        A.Normalize(
            mean=base_mean,
            std=base_std,
            max_pixel_value=255
        ),
        ToTensorV2(transpose_mask=True),
    ]
    valid_aug_list = [
        A.Normalize(
            mean=base_mean,
            std=base_std,
            max_pixel_value=255
        ),
        ToTensorV2(transpose_mask=True),
    ]
```

Figure 6 Data Enhancement Strategy

Model Integration:

After many training processes are over, we get a large number of model files. We found that there are large differences in the optimal inference thresholds for test sets between models, so how to ch oose a model has always been a difficult problem. Generally speaking, we are pushing

The number of models selected during the theory is a multiple of 6 - because a single whole training will produce six models (50% off +10% off full data training). In the pseudo-label generation process mentioned in the "Data Preparation" section, we used 12 models for integration, and in the remaining cases we used 6 models for integration.

Specifically, in the integrated inference process, we first apply the flip in the height and width direction of the image as test data augmentation (TTA), and obtain the TTA results of a single model, and then take the Sigmoid results of the six models, and take the average of the predicted probability of the final six models.

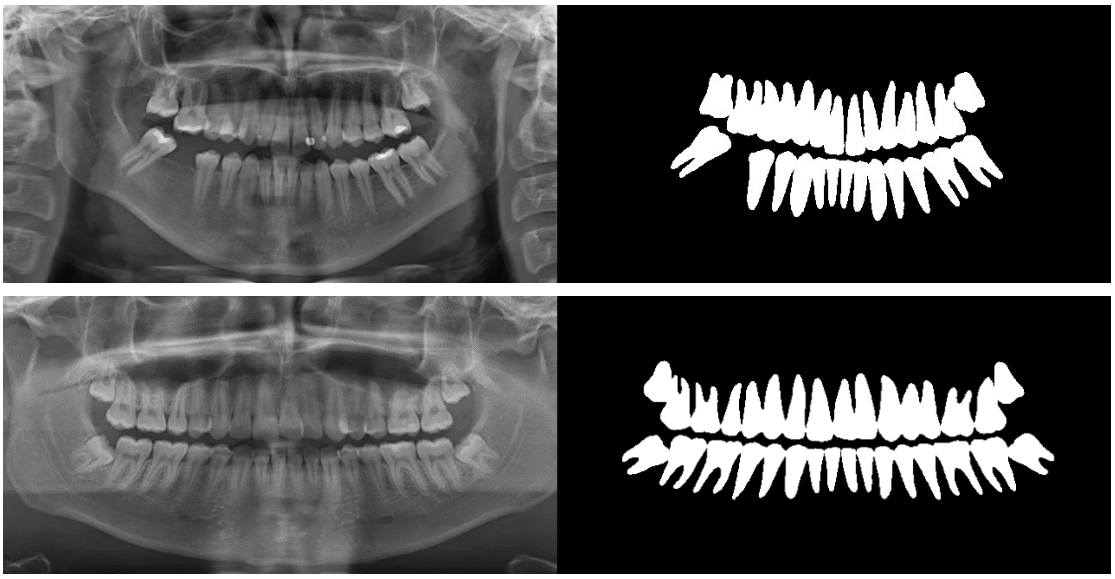
Threshold selection:

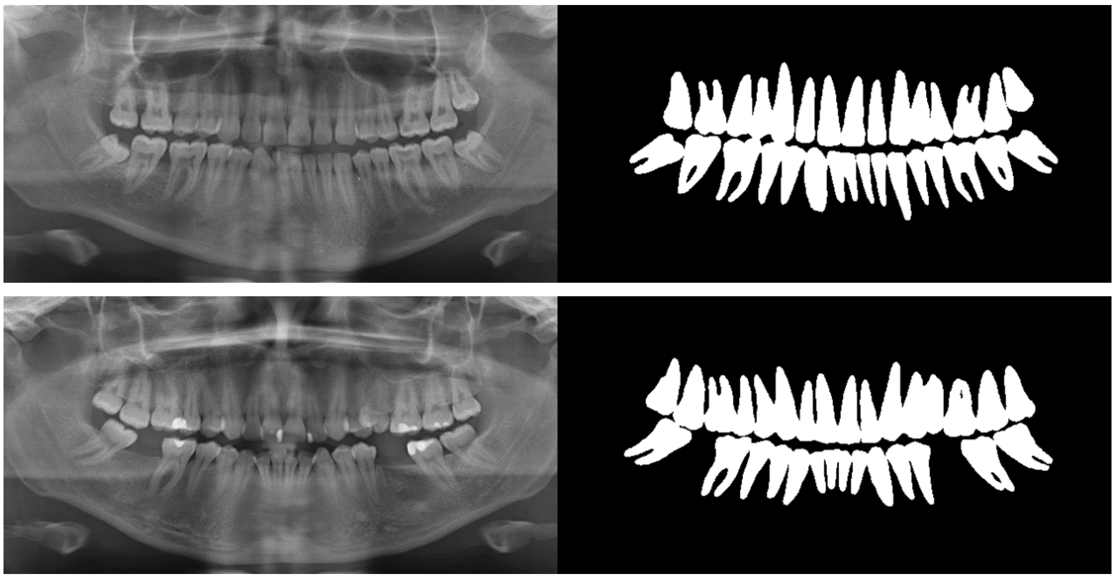
As mentioned earlier, we found that there are large differences in the optimal threshold for prediction sets among models. In the preliminary round, our optimal model has an optimal threshold of about 0.2, but during the rematch, we have obtained a model with an optimal threshold of about 0.9 . For the optimal model we submitted, our segmentation threshold is set to 0.3.

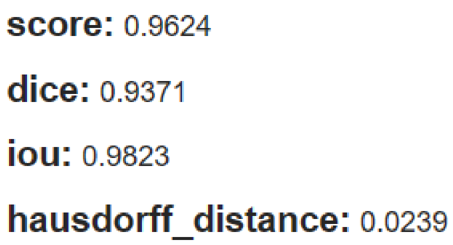
Post-processing:

Due to the limited time of team members, we only follow the mask in the "data processing" link and use tough masking on both sides of the output result. But in previous Kaggle competition practices, we have also explored further optimization post-processing of binarized images, such as culling some smaller areas. We believe that if proper post-processing methods are used, there is still room for further improvement in the final output results.

Results show:







score: 0.9624

dice: 0.9371

iou: 0.9823

hausdorff_distance: 0.0239

References:

[i][1] Hatamizadeh A , Nath V , Tang Y ,et al.Swin UNETR: Swin Transformers for Semantic Segmentation of Brain Tumors in MRI Images[J]. 2022.DOI:10.48550/arXiv.2201.01266.