Detection Model

1. Data preprocessing

Before inputting data into models, we use the csv profile “Train_GC_GT.csv” to find the gt-label of the image by locating the name of each image in the csv profile.

2. Train-Validation-Test dataset split

We use **sklearn.model_selection.KFold** to split 100 cases) of training process into 5 consecutive folds (with random shuffling). Each fold (20 cases) is then used once as validation while the 4 remaining folds (80 cases) form the training set. The 100 cases of the preliminary competition process are set as the test dataset.

3. Detection

For layer segmentation, we train a modification of Res-Net[1] named resnet50(Fig 1). It captures more high-level information while maintaining a good property of forward and backward propagation. Though it is an old network but it is an efficient and widely used one. resnet50 mainly contains five major stage: The first one is a pooling struct while the other four captures high-level information by using convolution to shrink the size of the input image to 1/4 and double the numbers of the kernel. The resnet50 we use is a pre-trained network.

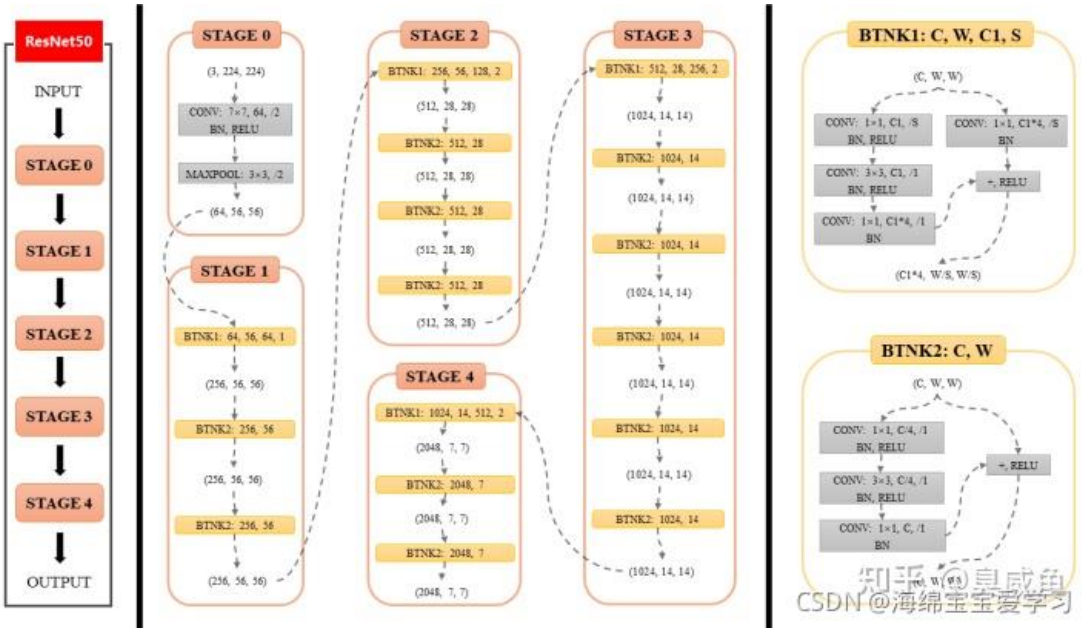

Fig 1. The framework of resnet50

4. Implementation Details

Our whole framework was implemented with Python based on Pytorch. And all models were trained on a 24GB RTX3090Ti GPU.

In experiment, the optimizer was Adam, and the **learning rate** was decayed by 0.2 every 60 epochs with an initial value of 0.0005. For each fold, the model was trained by **bce loss** with a **batch size 16** for **200 epochs** (1000 iterations). The model was evaluated on the validation set to get the **validation loss**, the **RNFL dice score, GCIPL dice score, and the Choroid layer dice score** every 5 epochs**.** We saved the model checkpoint with the highest validation metric.

5. Reference

[1] Kaiming He, et al. "Deep Residual Learning for Image Recognition." arXiv:1512.03385 [cs.CV]

Layer Segmentation Model

1. Train-Validation-Test dataset split (same as tasks 1 and 2)

We use **sklearn.model_selection.KFold** to split 100 cases of training process into 5 consecutive folds (with random shuffling). Each fold (20 cases) is then used once as validation while the 4 remaining folds (80 cases) form the training set. The 100 cases of the preliminary competition process are set as the test dataset.

2. Data preprocessing

Before inputting data into models, we crop two sub-image patches with the size of 800×800 based on the left and right of the input image. For every input, we applied the same data augmentations to all images and masks. In detail, **Resize, RandomResizedCrop, RandomHorizontalFlip, RandomVerticalFlip, RandomShiftScaleRotate, RandomBlur, RandomGaussNoise, RandomBrightnessContrast, CoarseDropout,** and **Normalize**. For validation and test set, only **Resize** and **Normalization** was used.

```python
if applied_types == None:
    data_transforms = albumentations.Compose([
        albumentations.Resize(New_size[0], New_size[1]),
        ToTensor()
        ])

elif applied_types == "train":
    data_transforms = albumentations.Compose([
        albumentations.Resize(New_size[0], New_size[1]),
        albumentations.RandomResizedCrop(height = New_size[0], width = New_size[1], scale=(0.95, 1.05), ratio=(0.95, 1.05), p=0.25),
        albumentations.HorizontalFlip(p=0.5),
        albumentations.VerticalFlip(p=0.25),
        albumentations.ShiftScaleRotate(shift_limit=0.0625,
                                        scale_limit=0.05,
                                        rotate_limit=10,
                                        p=0.25),
        albumentations.OneOf([
            albumentations.Blur(blur_limit=5),
            albumentations.GaussianBlur(blur_limit=5),
            albumentations.MedianBlur(blur_limit=5),
            albumentations.MotionBlur(blur_limit=5)
            ], p=0.25),
        transforms.RandomBrightnessContrast(brightness_limit=0.1, contrast_limit=0.1, p=0.25),
        albumentations.OneOf([
            albumentations.JpegCompression(), albumentations.GaussNoise()], p=0.25),
        albumentations.CoarseDropout(p=0.1),
        albumentations.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225], max_pixel_value=255.0),
        ToTensor()])

elif applied_types == "val" or applied_types == "test":
    data_transforms = albumentations.Compose([
        albumentations.Resize(New_size[0], New_size[1]),
        albumentations.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225], max_pixel_value=255.0),
        ToTensor()
        ])
```

Fig 1. The data augmentations

3. Layer Segmentation

For layer segmentation, we train four frameworks. The first framework is CEA-Net[1] (Fig 2). It captures more high-level information and preserves spatial information for 2D medical image segmentation. CEA-Net mainly contains three major components: a feature encoder module, a context extractor, and a feature decoder module. In the feature encoder module, the model proposes an Efficient Channel Attention (ECA) module, which avoids dimensionality reduction and captures cross-channel interaction in an efficient way. Specifically, ECA module first adaptively determines kernel size k, and then performs 1D convolution followed by a Sigmoid function to learn channel attention. We use a pre-trained ResNet34 block as the fixed feature extractor. The context extractor module is formed by a newly proposed dense atrous convolution (DAC) block and residual multi-kernel pooling (RMP) block.

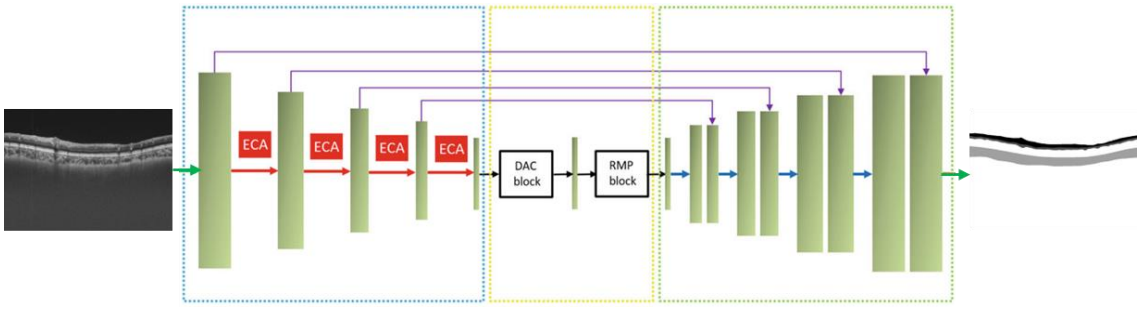

Fig 2. The framework of CEA-Net

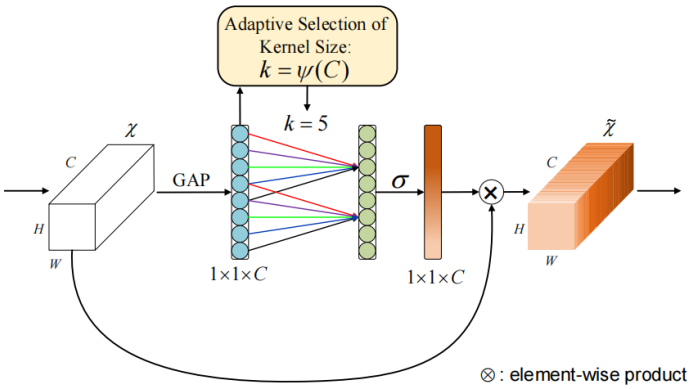

Fig 3. ECA block

The second framework is TransUNet[2] (Fig 4), which merits both Transformers and U-Net, as a strong alternative for medical image segmentation. On the one hand, the Transformer encodes tokenized image patches from a convolution neural network (CNN) feature map as the input sequence for extracting global contexts. On the other hand, the decoder upsamples the encoded features, which are then combined with the high-resolution CNN feature maps to enable precise localization.

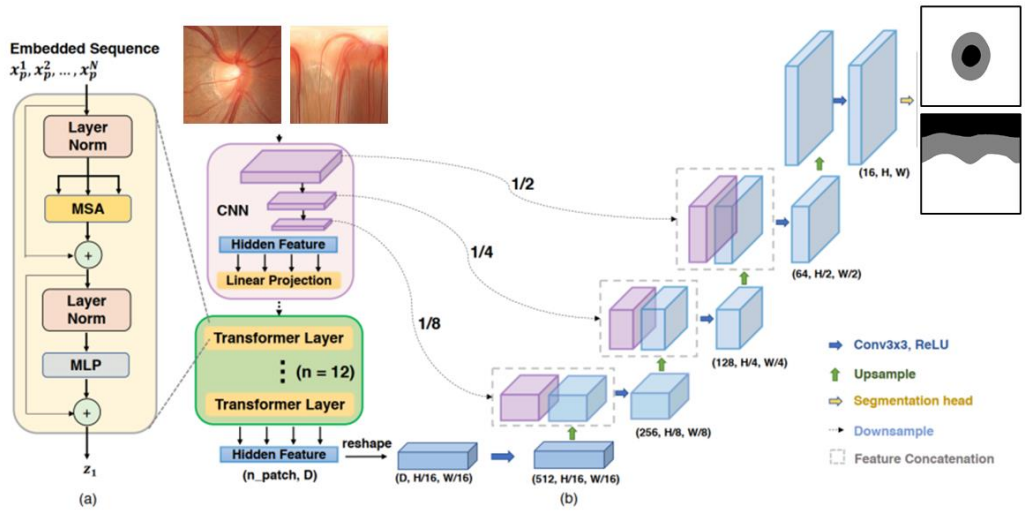

Fig 4. The framework of TransUNet

The third framework we used is based on the method[3] (Fig 5). Specifically, the input is a three channel image. The features extracted by the CEA net (CE net with ECA module) are shared by the two output branches. The top branch (conv-m) uses channel-wise softmax to output segmentation probability maps for the layers, backgrounds (choroid or vitreous), and lesions. The bottom branch (conv-s) uses column-wise soft-max to produce variational qi and then uses soft-argmax and iterative ReLU to produce M structured surfaces.

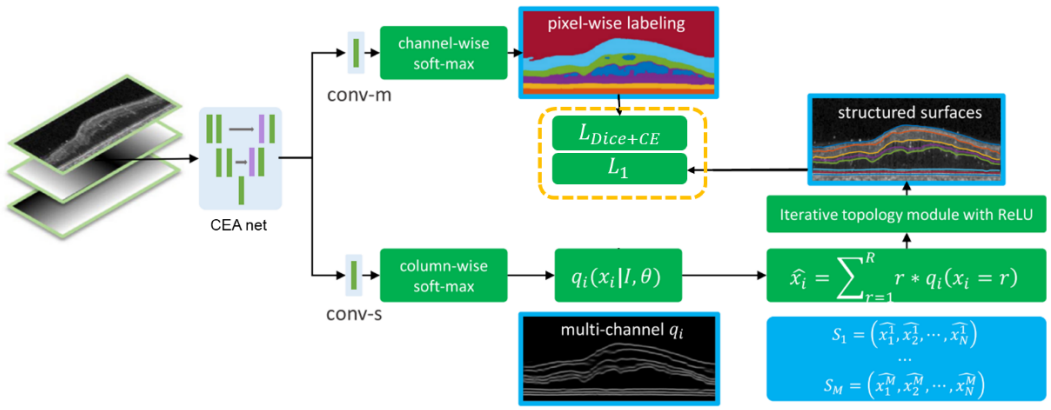

Fig 5. The framework of third framework

The fourth framework is MSnet[4] (Fig 6). In the network, with multi-level and multi-stage cascaded subtraction operations, the complementary information from lower order to higher order among different levels can be effectively obtained, thereby comprehensively enhancing the perception of target areas.

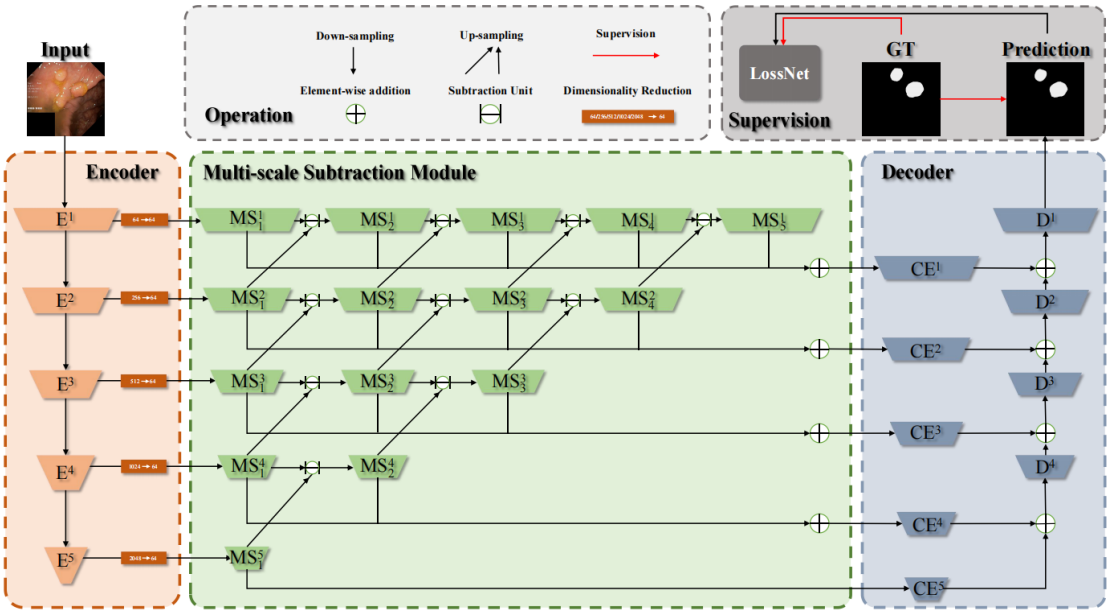

Fig 6. The framework of MSnet

4. Implementation Details

Our whole framework was implemented with Python based on Pytorch. And all models were trained on a 24GB RTX3090Ti GPU.

In experiment, the optimizer was Adam, and the **learning rate** was decayed by 0.2 every 60 epochs with an initial value of 0.0005. For each fold, the model was trained by (**multilabel dice loss+multilabel bce loss+Focal loss)** with a **batch size 8** for **200 epochs** (4000 iterations). The model was evaluated on the validation set to get the **validation loss**, the **RNFL dice score, GCIPL dice score, and the Choroid layer dice score** every 5 epochs**.** We saved the model checkpoint with the highest validation metric.

5. Ensemble Stage

After the training stage, we ensembled all model predictions (**five folds, four frameworks, and two kinds of inputs, a total of 30 models**) to get the final outputs. Firstly, we loaded the saved best metric checkpoint for each model and made predictions on the test set, respectively. So, for each case in the test set, we had 30 independent prediction probability maps. We apply the sigmoid operation on each map and add them together. Finally, we use max voting to get the final layer segmentation masks.

According to the crop coordinates, we map the patch segmentations masks into the whole image masks.

6. Post-processing Stage

After getting the ensemble results, we use morphological opening and closing operations to process the results. Three different structural element size, 5 × 5, 12 × 12 and 20 × 20, are used here for morphological operations on the predicted results from our network.

7. Reference

[1] Wang, Qilong, et al. "Supplementary material for 'ECA-Net: Efficient channel attention for deep convolutional neural networks." Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, Seattle, WA, USA. 2020.

[2] Chen, Jieneng, et al. "Transunet: Transformers make strong encoders for medical image segmentation." arXiv preprint arXiv:2102.04306 (2021).

[3] He, Y. , et al. "Structured layer surface segmentation for retina OCT using fully convolutional regression networks." Medical Image Analysis 68.136295(2020):101856.

[4] Zhao, Xiaoqi , L. Zhang , and H. Lu . "Automatic Polyp Segmentation via Multi-scale Subtraction Network." 2021.