The Description of the Algorithm Evaluated in the MICCAI 2022 GOALS Challenge

Xiaoqi Zhao
Dalian University of Technology
Dalian, Liaoning, China
zxq@mail.dlut.edu.cn

Youwei Pang
Dalian University of Technology
Dalian, Liaoning, China
lartpang@mail.dlut.edu.cn

1 Full name and abbreviated name of the algorithm

We propose the **GMM**Net by integrating our three models **G**ateNet [11], **MI**Net [8] and **M**SNet [12] to predict the OCT layer segmentation results.

2 Tasks in the MICCAI 2022 GOALS Challenge

Task1: OCT Layer Segmentation.
Task2: Glaucoma Detection.

3 Description of the Model

3.1 OCT Layer Segmentation

It is well known that different models usually have different characteristics and expertise. To make the segmentation of the OCT layer more robust and accurate, we follow the ensemble learning strategy by integrating our three high-performance segmentation networks [11, 8, 12]. GateNet [11] was published in ECCV 2020 (oral), we design the gated dual branch structure to build the cooperation among different levels of features and improve the discriminability of the whole network. Fig. 1 shows the architecture of GateNet. Please refer to the literature [11] for more details. MINet [8] was published in CVPR 2020, we investigate the multi-scale issue to propose an effective and efficient network with the transformation-interaction-fusion strategy. Fig. 2 shows the architecture of MINet. Please refer to the literature [8] for more details. MSNet [12] was published in MICCAI 2021, we propose a subtraction unit to produce the difference features between adjacent levels in encoder. Fig. 3 shows the architecture of MSNet. Please refer to the literature [12] for more details. For the loss function, we use the pixel position aware loss L_{ppa} [9] which have been widely adopted in segmentation tasks.

3.2 Glaucoma Detection

We adopt the SE-ResNet-50 [4, 5] as the classifier to finish the glaucoma diagnosis task based on OCT images.

4 Implementation Details

4.1 OCT Layer Segmentation

- Our training sets are consisted of the GOALS2022-Train [3] with a total of 100 pairs of OCT images and the gold standard. We do not set extra validation data for selecting the best model. We use the PyTorch framework

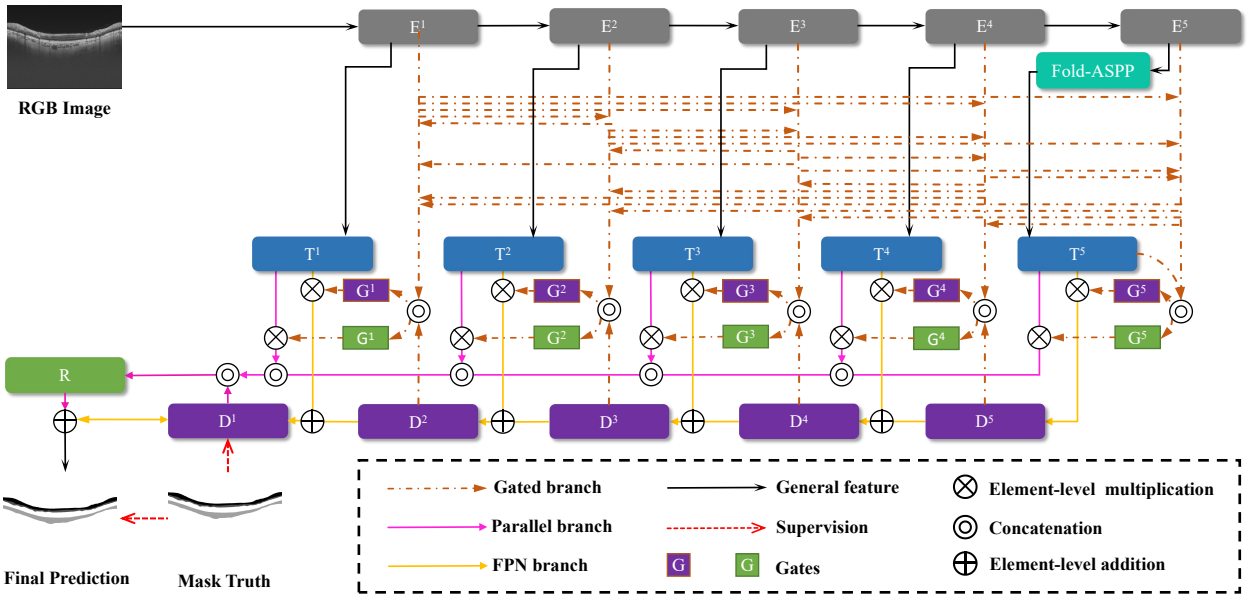


Figure 1: The overall architecture of GateNet.

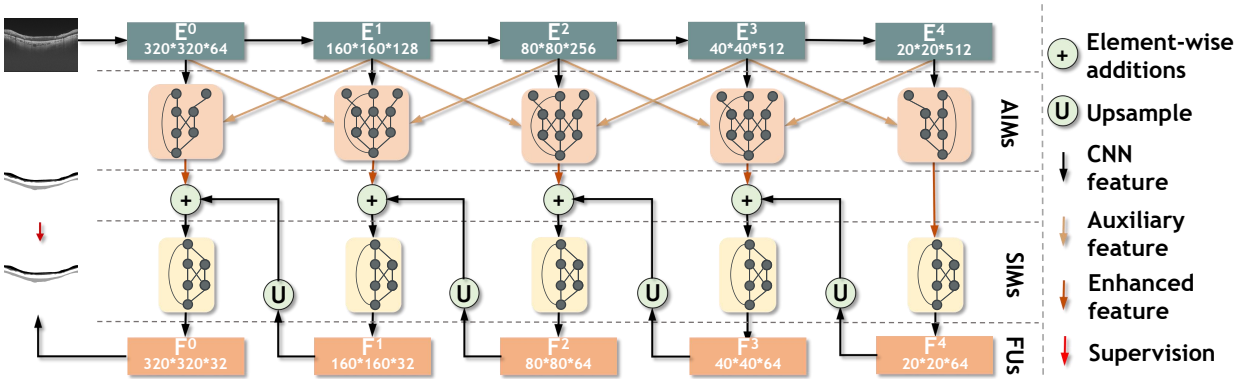


Figure 2: The overall architecture of MINet.

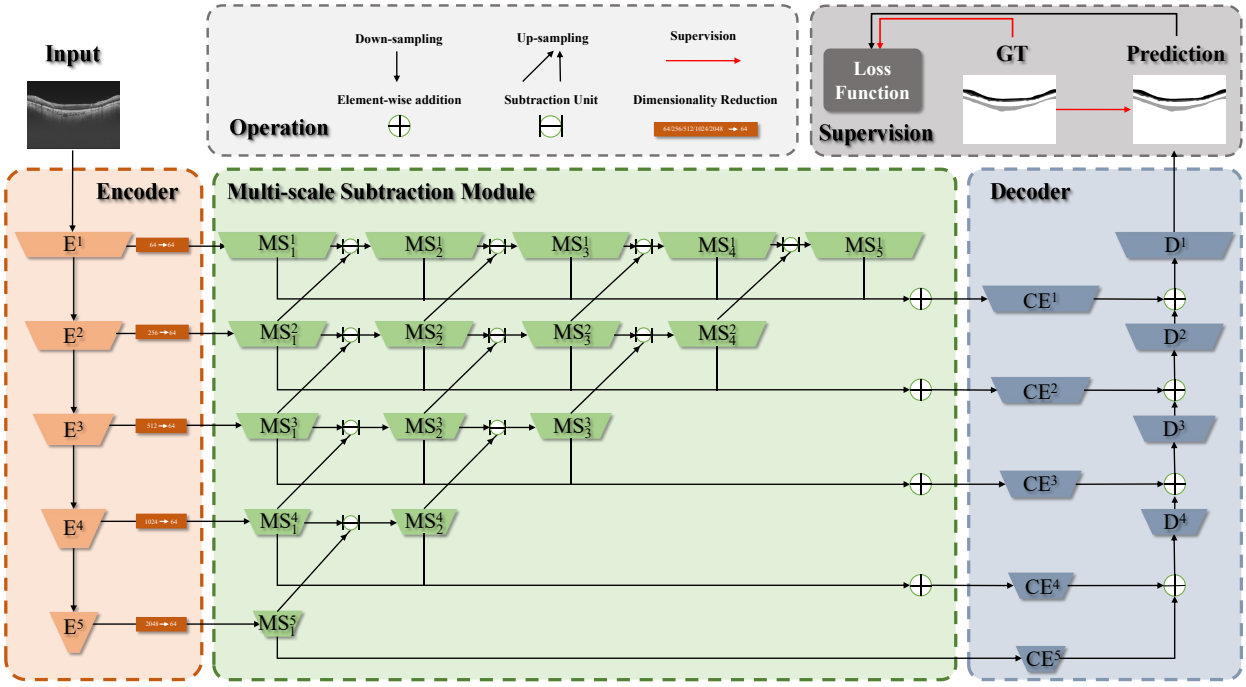


Figure 3: The overall architecture of MSNet.

to implement our models on one RTX 3090 GPU for 100, 200 and 300 epochs.

- We load the ResNeXt-101 [10] imagenet pre-trained weights as the initialization for each encoder of the models.
- The input resolutions of images are resized to 880×640 and we employ a general multi-scale training strategy as most methods [9, 1, 7, 13, 2, 12].
- We adopt some image augmentation techniques to avoid overfitting, including random cropping, flipping, rotating, and color enhancing.
- The mini-batch size settings in the GateNet, MINet and MSNet are 2, 4, and 4, respectively.
- For the optimizer, we use the Adam [6]. For the learning rate, initial learning rate is set to 0.0001. We adopt the "step" learning rate decay policy, and set the decay size as 30, 80, 150 and decay rate as 0.9 to train 100, 200, 300 epochs.
- To enable each network to focus more on specific tasks. During the training phase, we split the overall task into three sub-tasks and train them one by one. Fig. 4 shows two ways of task splitting. Unlike the left, it directly splits the task into RNFL, GCIPL and Choroid. From our observations, RNFL and GCIPL are seamlessly connected, so we decide to fuse RNFL and GCIPL into one area as a second task. Finally, the RNFL was removed from it in the testing phase to get the pure GCIPL. We show the visual results under two different task splitting schemes in Fig. 5. It can be seen that, the right one can get a better performance.
- During the testing phase, we adopt the multi-scale testing strategy and ensemble learning strategy to merge each model prediction.
- We adopt conditional random fields (CRF) as post-processing to help the distinction between foreground and background clearer. Then, we further binarize the result with a threshold of 20.

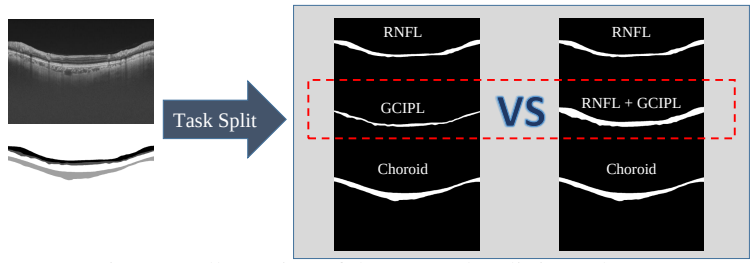


Figure 4: Illustration of the two task splitting schemes.

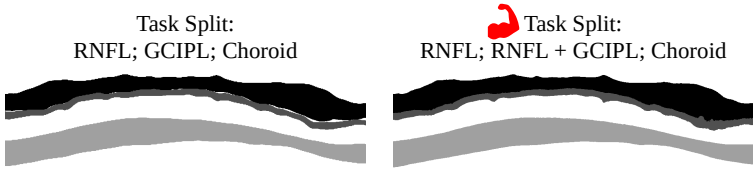


Figure 5: The prediction results corresponding to the two task splitting schemes.

- According to our observations, we find that the segmentation map for each sub-task should be a connected component without holes. However, the network may predict the segmentation map with multi-regions and exists some holes due to the OCT image background noise. To this end, we design a maximum connected component filter and a hole filling algorithm to make the prediction results more complete. The implemented code is as follows:

```
def filter(img_np,filter_thr):
    img_np = np.array(img_np, dtype="uint8")
    img_np = img_np//255
    num_labels, labels, stats_pre, centroids = cv2.connectedComponentsWithStats
                                                (img_np, ltype=cv2.CV_32S)
    area = []
    real_index = []
    for i in stats_pre:
        area.append(i[-1])
    if len(area)==1:
        flag_ring = 0
    else:
        flag_ring = 1
        max_area = max(area[1:])
        for i, v in enumerate(area):
            if v >= max_area/filter_thr:
                real_index.append(i)
    pre_single = np.zeros((img_np.shape[0], img_np.shape[1]),
                          np.uint8)
    for i in real_index:
          if i!=0:
              mask = labels == i
              pre_single[:, :][mask] = 1

    return flag_ring, pre_single*255
```

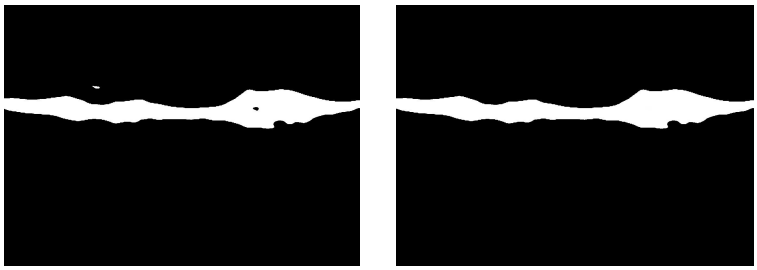
```python
def fillhole(mask):
    mask = mask.astype(dtype="uint8")
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE,
                                cv2.CHAIN_APPROX_SIMPLE)
    len_contour = len(contours)
    contour_list = []
    for i in range(len_contour):
        drawing = np.zeros_like(mask, np.uint8)  # create a black image
        img_contour = cv2.drawContours(drawing, contours, i, (255, 255, 255), -1)
        contour_list.append(img_contour)

    out = sum(contour_list)
    return out
```

Fig. 6 shows the comparison between initial prediction (Left) and the prediction with the filter and fillhole function (Right).

- For the Choroid layer, there are some hard examples make our network can not predict the overall layer well. According to our empirical observations, Choroid and GCIPL share the same morphological trend and closer geometrical areas. Therefore, we translate and fill the missing Choroid layers according to the prediction results of GCIPL, thus obtaining the complete prediction result map of the Choroid layers. The implemented code is as follows:

```python
    duan_index = []
    for i in range(1100):
        if sum(res3[0:800,i])==0:
            duan_index.append(i)
    if duan_index != []:
        duan_index_left = duan_index[1:]
        duan_index_left.append(0)
        difference_x_list = list(map(lambda x: x[0]-
                    x[1],zip(duan_index_left,duan_index)))
        duan_location_index = [i for(i,v) in
                        enumerate(difference_x_list)if v!=1]
        duan_location_x = []
        for i in range(len(duan_location_index)):
            if i ==0:
                start_duan = duan_index[0]
                end_duan = duan_index[duan_location_index[i]]
```



Initial Prediction **Prediction (with the filter and fillhole function)**

Figure 6: Visual results of the connected component filter and the hole filling algorithm.

```
                duan_location_x.append((start_duan,end_duan))
            else:
                start_duan = duan_index[duan_location_index[i-1]+1]
                end_duan = duan_index[duan_location_index[i]]
                duan_location_x.append((start_duan, end_duan))
        duan_location_y_res2 = []
        for i in duan_location_x:
            duan_res2 = np.zeros((res3.shape[0], res3.shape[1]),
                                 np.uint8)
            duan_res2[0:800,i[0]:i[1]] = res2[0:800,i[0]:i[1]]
            duan_res2_y_index = []
            for j in range(800):
                if sum(duan_res2[j, 0:1100]) != 0:
                    duan_res2_y_index.append(j)
            if duan_res2_y_index==[]:
                duan_res2 = np.zeros((res3.shape[0], res3.shape[1]),
                                     np.uint8)
                duan_res2[0:800, i[0]:i[1]] = res1[0:800, i[0]:i[1]]
                duan_res2_y_index = []
                for i in range(800):
                    if sum(duan_res2[i, 0:1100]) != 0:
                        duan_res2_y_index.append(i)
            start_duan_res2_y = duan_res2_y_index[0]
            end_duan_res2_y = duan_res2_y_index[-1]
            duan_location_y_res2.append((start_duan_res2_y,
                                         end_duan_res2_y))
        for i in range(len(duan_location_y_res2)):
            if i!=len(duan_location_y_res2)-1 or duan_location_x[i]
                                                              [0]==0:
                a = np.zeros((res3.shape[0], res3.shape[1]), np.uint8)
                a[0:800,duan_location_x[i][1]:duan_location_x[i][1]+2] =
                res2[0:800, duan_location_x[i][1]:duan_location_x[i][1]+2]
                b = np.zeros((res3.shape[0], res3.shape[1]), np.uint8)
                b[0:800, duan_location_x[i][1]:duan_location_x[i][1] + 2] =
                res3[0:800,duan_location_x[i][1]:duan_location_x[i][1] + 2]
                a_list = []
                b_list = []
                for j in range(800):
                    if sum(a[j, 0:1100]) != 0:
                        a_list.append(j)
                    if sum(b[j, 0:1100]) != 0:
                        b_list.append(j)
                if a_list != []:
                    distance_res2_res3 = b_list[0]-a_list[0]
                else:
                    distance_res2_res3 = 10

            else:
                a = np.zeros((res3.shape[0], res3.shape[1]), np.uint8)
                a[0:800,duan_location_x[i][0]-2:duan_location_x[i][0]] =
                res2[0:800, duan_location_x[i][0]-2:duan_location_x[i][0]]
```

```
                b = np.zeros((res3.shape[0], res3.shape[1]), np.uint8)
                b[0:800,duan_location_x[i][0]-2:duan_location_x[i][0]] =
                res3[0:800, duan_location_x[i][0]-2:duan_location_x[i][0]]
                a_list = []
                b_list = []
                for j in range(800):
                    if sum(a[j, 0:1100]) != 0:
                        a_list.append(j)
                    if sum(b[j, 0:1100]) != 0:
                        b_list.append(j)
                if a_list != []:
                    distance_res2_res3 = b_list[0]-a_list[0]
                else:
                    distance_res2_res3 = 10
            res3[duan_location_y_res2[i][0]+distance_res2_res3:
            duan_location_y_res2[i][1]+distance_res2_res3,
            duan_location_x[i][0]:duan_location_x[i][1]] =
            res2[duan_location_y_res2[i][0]:duan_location_y_res2[i][1],
            duan_location_x[i][0]:duan_location_x[i][1]]
```

Fig. 7 shows the comparison between initial Choroid layer prediction (Left) and the prediction with the the Choroid layer filling strategy (Right).

4.2 Glaucoma Detection

- Our training sets are consisted of the GOALS2022-Train [3] with a total of 50 Glaucoma and 50 non-Glaucoma images. We do not set extra validation data for selecting the best model. We use the PyTorch framework to implement our models on one RTX 3090 GPU for 20 epochs.
- We load the SE-ResNet-50 [4, 5] imagenet pre-trained weights as the initialization for training the two-class classifier.
- We adopt some image augmentation techniques to avoid overfitting, including random cropping and flipping.
- The mini-batch size setting is 4.
- For the optimizer, we use the SGD. For the learning rate, initial learning rate is set to 0.001. We adopt the "step" learning rate decay policy, and step size is 7, gamma is 0.1.

References

[1] Z. Chen, Q. Xu, R. Cong, and Q. Huang. Global context-aware progressive aggregation network for salient object detection. In *AAAI*, pages 10599–10606, 2020.

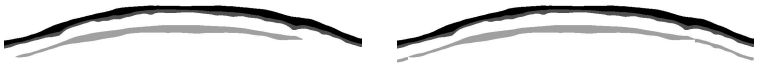
[2] D.-P. Fan, G.-P. Ji, T. Zhou, G. Chen, H. Fu, J. Shen, and L. Shao. Pranet: Parallel reverse attention network for polyp segmentation. In *MICCAI*, pages 263–273, 2020.

[3] H. Fang, F. Li, H. Fu, J. Wu, X. Zhang, and Y. Xu. Dataset and evaluation algorithm design for goals challenge. *arXiv preprint arXiv:2207.14447*, 2022.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[5] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *CVPR*, pages 7132–7141, 2018.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.



Initial Prediction **Prediction (filling the Choroid region with the GCIPL region)**

Figure 7: Visual results of the Choroid layer filling strategy.

[7] Y. Lv, J. Zhang, Y. Dai, A. Li, B. Liu, N. Barnes, and D.-P. Fan. Simultaneously localize, segment and rank the camouflaged objects. In *CVPR*, pages 11591–11601, 2021.

[8] Y. Pang, X. Zhao, L. Zhang, and H. Lu. Multi-scale interactive network for salient object detection. In *CVPR*, pages 9413–9422, 2020.

[9] J. Wei, S. Wang, and Q. Huang. F^3net: Fusion, feedback and focus for salient object detection. In *AAAI*, pages 12321–12328, 2020.

[10] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 1492–1500, 2017.

[11] X. Zhao, Y. Pang, L. Zhang, H. Lu, and L. Zhang. Suppress and balance: A simple gated network for salient object detection. In *ECCV*, pages 35–51, 2020.

[12] X. Zhao, L. Zhang, and H. Lu. Automatic polyp segmentation via multi-scale subtraction network. In *MICCAI*, pages 120–130, 2021.

[13] T. Zhou, H. Fu, G. Chen, Y. Zhou, D.-P. Fan, and L. Shao. Specificity-preserving rgb-d saliency detection. In *ICCV*, pages 4681–4691, 2021.