2D-Sixth place-One Piece-Tech Note

one. introduction

Medical image segmentation is an important task in the field of medical image processing, aiming to accurately segment the areas of interest (such as diseases, organs, etc.) in medical images from the background. This report will introduce the development process, training skills and innovative ideas of tooth segmentation tasks based on 2D panoramic images.

two. Data collection and preprocessing

Divide the tagged part of the official rematch data set to 4 folders. Resize the image and label , with a size of 640*640, and normalize the image:

```
valid_aug_list = [
        A.Resize(size, size),
        A.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        ),
        ToTensorV2(transpose_mask=True),
    ]
```

three. Model selection and design

Use segmentation_models_pytorch to define the model, and configurable options include image segmentation models such as Unet, Unet++, FPN, PSPNet, PAN, etc. Encoders include ResnNet, SeNet, MixTransformer, efficientnet, mobileone, etc.

The final model is selected by Unet++, and the multi-model fusion of Unet++ is performed using a flexible encoder configuration. The encoder part selects and encoders including efficientnet-b5 ,

```
model = smp.UnetPlusPlus(encoder_name=encoder_name, activation='sigmoid',
decoder_channels=CFG.decoder_channels,
encoder_depth=CFG.encoder_depth).to(CFG.device)
```

Four. Data Enhancement

Use random horizontal and vertical flips; random brightness contrast; Gaussian noise; grid distortion; random occlusion; random center clipping; translation zoom rotation.

```python
train_aug_list = [
        # A.RandomCrop(height=size, width=size, p=0.5),
        A.Resize(size, size),
        # A.Rotate(limit=90, p=0.5),
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),
        A.RandomBrightnessContrast(p=0.75),
        A.ShiftScaleRotate(p=0.75),
        A.OneOf([
            A.GaussNoise(var_limit=[10, 50]),
            A.GaussianBlur(),
            A.MotionBlur(),
        ], p=0.4),
        A.GridDistortion(num_steps=5, distort_limit=0.3, 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=0.5),
        A.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        ),
        ToTensorV2(transpose_mask=True),
```

```python
def get_randon_cnter_crop(self, center, shape, min_size):
        max_width = shape[1]
        max_height = shape[0]
        max_size = min(min(max_width - center[0], center[0]),
min(max_height - center[1], center[1])) * 2
        # 生成逐渐增加的边框大小
        size = random.randint(min(min_size, max_size), max_size)
        x = center[0] - size // 2
        y = center[1] - size // 2
        self.selection = [y, y+size, x ,x+size]
```

five. Training strategies and techniques

Adopt GradualWarmupSchedulerV2 learning rate adjustment strategy.

```python
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]
def get_scheduler(cfg, optimizer):
    scheduler_cosine = torch.optim.lr_scheduler.CosineAnnealingLR(
        optimizer, cfg.epochs, eta_min=CFG.min_lr)
    scheduler = GradualWarmupSchedulerV2(
        optimizer, multiplier=10, total_epoch=1, after_scheduler=scheduler_cosine)
    return scheduler
def scheduler_step(scheduler):
    scheduler.step()
```

```
def iou_loss(pred, target):
    # 计算预测和目标的交集和并集
    intersection = torch.sum(pred * target)
    union = torch.sum(pred) + torch.sum(target) - intersection

    # 计算 IOU
    iou = intersection / (union + 1e-6)
    # 计算 IOU 损失
    iou_loss = 1.0 - iou
```

```
def h_loss(pred, target):
    distances = torch.cdist(pred, target, p=1)
    # 计算 set1 中的每个点到 set2 的最小距离
    min_dist_set1_to_set2, _ = torch.min(distances, dim=2)
    # 计算 set2 中的每个点到 set1 的最小距离
    min_dist_set2_to_set1, _ = torch.min(distances, dim=3)
    min_dist_set = min_dist_set1_to_set2 + min_dist_set2_to_set1
    # 计算最大值和最小值
    max_dist = torch.max(min_dist_set)
    min_dist = torch.min(min_dist_set)
    # 归一化距离
    normalized_dist = min_dist / max(max_dist, 1e-5)
    # normalized_dist = transform(min_dist_set)
    # 取两个集合中的最小距离之和作为二维豪斯多夫距离
    hausdorff_dist = torch.min(normalized_dist) 
    return hausdorff_dist
class DHILoss():
    def __init__(self, dice_wight=0.4, iou_weight=0.3, h_weight=0.3, bce_weight=0.3):
        self.dice_wight = dice_wight
        self.iou_weight = iou_weight
        self.bce_weight = bce_weight
        self.h_weight = h_weight
        self.dice = DiceLoss(mode='binary')
        self.bce = nn.BCELoss()
    def do(self, pred, true):
        loss = self.dice(pred, true) * self.dice_wight + iou_loss(pred, true) * self.iou_weight + self.bce(pred, true) * self.bce_weight + self.h_weight * h_loss(pred, true)
        return loss
```

Using the AdamW optimizer, the initial learning rate is set to 5e-5, the regular term is set to 2 e-5, and 50 epochs are trained using 5-fold cross-validation, and the model with the highest final l ocal cv score is taken as the final submitted model.

six. test

Select the Unet++ best model with encoders efficientnet-b5, mobileone_s4, se_resnext101_3 2x4d for multi-model fusion. The results of multiple models are averaged as the final result.

Configuration:

```
chepoint_dir = 'result/logs/tst/last/' # 模型文件目录
test_encoder_list = ['efficientnet-b5', 'mobileone_s4', 'se_resnext101_32x4d'] # 混合模型
```

Building a multi-model fusion:

```python
class EnsembleModel:
    def __init__(self):
        self.models = []

    def __call__(self, x):
        outputs = [model(x) for model in self.models]
        outputs = torch.stack(outputs, dim=0)
        avg_preds = torch.mean(outputs, dim=0)
        return avg_preds
    def add_model(self, model):
        self.models.append(model)
def build_ensemble_model():
    model = EnsembleModel()
    for encoder in CFG.test_encoder_list:
        model_list = os.listdir(CFG.chepoint_dir + encoder + '/')
        for model_dir in model_list:
            model_path = CFG.chepoint_dir + encoder + '/' + model_dir + '/checkpoint/best.pth'
            _model = smp.UnetPlusPlus(encoder_name=encoder, encoder_weights=None)
            _model.to(CFG.device)
            print(model_path)
            try:
                state = torch.load(model_path, map_location=CFG.device)
                _model.load_state_dict(state['model_state_dict'])
            except:
                print('checpoint not load')
            _model.eval()
            model.add_model(_model)
    return model
```

result:

score	dice	iou	hausdorff_distance
0.9606	0.9334	0.9812	0.239