# 2D-第六名-One Piece-技术说明

## 引言

医学图像分割是医学图像处理领域的重要任务，旨在将医学图像中的感兴趣区域（如病变、器官等）从背景中准确地分割出来。本报告将介绍基于2D 全景图像的牙齿分割任务的开发过程、训练技巧以及创新思路。

## 数据收集和预处理

将官方提供的复赛数据集的有标签部分划分4个文件夹。对图像和标签进行Resize，大小为640\*640，并对图像进行归一化：

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),

    ]

## 模型选择和设计

使用segmentation\_models\_pytorch定义模型，可配置选项包括Unet、Unet++、FPN、PSPNet、PAN等图像分割模型。编码器包括ResnNet、SeNet、MixTransformer、efficientnet、mobileone等。

最终模型选择Unet++，并使用灵活的编码器配置对Unet++进行多模型融合，encoder部分选择·包括efficientnet-b5、mobileone\_s4、 se\_resnext101\_32x4d的编码器进行训练。

model = smp.UnetPlusPlus(encoder\_name=encoder\_name, activation='sigmoid', decoder\_channels=CFG.decoder\_channels, encoder\_depth=CFG.encoder\_depth).to(CFG.device)

## 数据增强

使用随机水平、垂直翻转；随机亮度对比度；高斯噪声；网格扭曲；随机遮挡；随机中心裁剪；平移缩放旋转。

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),

    ]

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]

## 训练策略和技巧

采用GradualWarmupSchedulerV2学习率调整策略。

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()

采用使用DiceLoss、IOULoss、HdLoss和BCELoss的加权损失函数。

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

    return iou\_loss

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

使用AdamW优化器，初始学习率设置为5e-5，正则项设置为2e-5，并使用5-fold交叉验证训练50个epoch，取最终本地cv分数最高的模型作为最终提交模型。

## 测试

选出编码器为efficientnet-b5、mobileone\_s4、 se\_resnext101\_32x4d的Unet++最佳模型进行多模型融合。将多个模型的结果取平均值后作为最终结果。

配置：

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

构建多模型融合器：

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

结果：

|  |  |  |  |
| --- | --- | --- | --- |
| score | dice | iou | hausdorff\_distance |
| 0.9606 | 0.9334 | 0.9812 | 0.239 |

