技术报告

杨奕鑫、白景琦
队伍：煲仔饭在煲

数据准备：

我们的对比赛数据的准备大致可分为初赛、复赛两个阶段：

1. 初赛阶段，我们使用标准的 2000 张训练集图像进行训练。我们对训练集和测试集进行了均值和标准差的评估，并利用所有总体图像的均值和标准差对训练数据进行归一化，具体数值为均值 0.4044，标准差 0.1550（根据 Albumentations .Normalize 函数中的要求，数值为实际值除以 255），并在数据增强的最后步骤对输入图像进行标准化操作。
2. 复赛阶段，我们使用标准的 900 张+2100 张训练集图像进行训练。其中，2100 张图像的数据来源于微调模型推理的伪标签：具体而言，我们使用初赛的模型权重初始化一个相同的网络，随后在 900 张带标签数据上进行微调，得到新的用于推理伪标签的网络模型。利用 12 个微调后的模型，得到 900+2100 的混合标签数据后，我们完整训练了一次复赛网络模型，并根据提交的最优结果阈值，重新推理 2100 张数据的伪标签，在得到迭代后的 2100 张伪标签后，我们对其进行了轻微的微调，主要利用下图所示形状的 mask(320,640)与原图相乘，剔除了两侧部分区域(:30,610:)，并对剩余区域进行了检查和微调，具体可见于上传 docker 中的相关数据集文件。

图 1 伪标签处理 Mask

我们也同样使用和初赛类似的方法对所有图像数据进行均值和标准差的评估，并以此作为归一化的数值基准。在复赛阶段具体的均值与标准差分别为 0.408 及 0.163。

值得注意的是，我们在临近比赛结束时发现第二阶段计算的均值及标准差有所遗漏，并不完全包含所有图像。在我们之前的其他同类比赛“Vesuvius Challenge - Ink Detection | Kaggle”实践中，均值和标准差的轻微不同可能会引起 1e-3 数量级上的指标性能差异，因此我们对于实际理论值与我们实际训练模型时的使用值性能孰优孰劣之间存疑。

算法模型:

由于在 Kaggle 比赛中有过相似实践，在构建算法模型时我们选择了 MONAI 库作为我们构建并改进算法基础模型的 baseline 库。我们构件的模型基于 SwinUNETR[1][1]，并在其基础上作了较多探索，原 SwinUNETR 网络的基础架构如下:

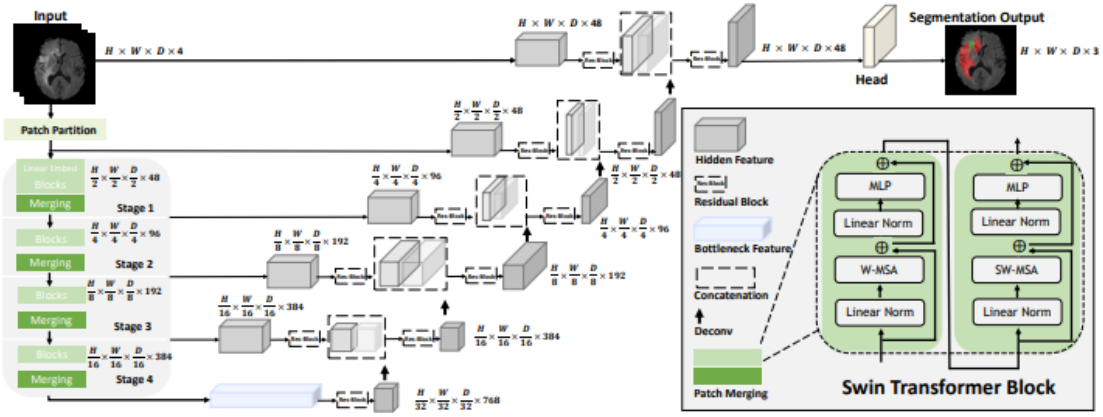


图 2. SwinUNETR 架构

我们尝试过的网络改进方向包括:

1. 增加 BackBone 的规模，并在前向过程中嵌入子模型，构成类似于集成模型+二阶段的算法网络。
2. 增加深监督，对上采样过程中的每一步都施加监督信号。
3. 在特定层增加通道注意力机制。
4. 受 PointRend 思想的启发，设计额外的特征提取模块，从输入图像上提取特征并补充连接到最终输出模块前。

在我们有限的实践中，最终方案 4 显示出了值得额外关注的提升。因此在初赛的最后阶段及复赛中，我们均以方案 4 为引导来设计改进算法。

在初赛的实践中，我们修改了一版 SwinUNETR 网络，体现于提交依赖文件中 monai.networks.nets 中的 swin_unetr_sim.py 我们在 Input 的特征尺度上增加了一个 1x1 卷积的"encoder_top"提取低维度的信息，并在输出路径"out_top"上将其与上采样的输出融合，前向过程代码如下:

```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
```

图 3 SwinUNETR_Sim 的前向流程

在复赛的实践中，我们根据先前的改进方向改写了最顶层的特征提取模块，同时对融合后的过程进行了加深。具体而言，我们对 encoder_top 使用了 13x13 的卷积，卷积步长为 1。虽然使用该卷积参数会产生较大的 padding，但由于此次竞赛数据集的特点——即前景区域集中在图像的中心部分，我们认为图像边缘的大幅度 padding 并不会产生较大的负面影响。与此同时，我们加深了低维特征与高维度上采样特征融合后的模块，增加了一个标准的带有残差连接的 MONAI UnetrBasicBlock 模块，以期能更好地融合两支信息。这版修改的网络体现在提交文件中 monai.networks.nets 的 swin_unetr_hs.py。最终的网络结构图如下：

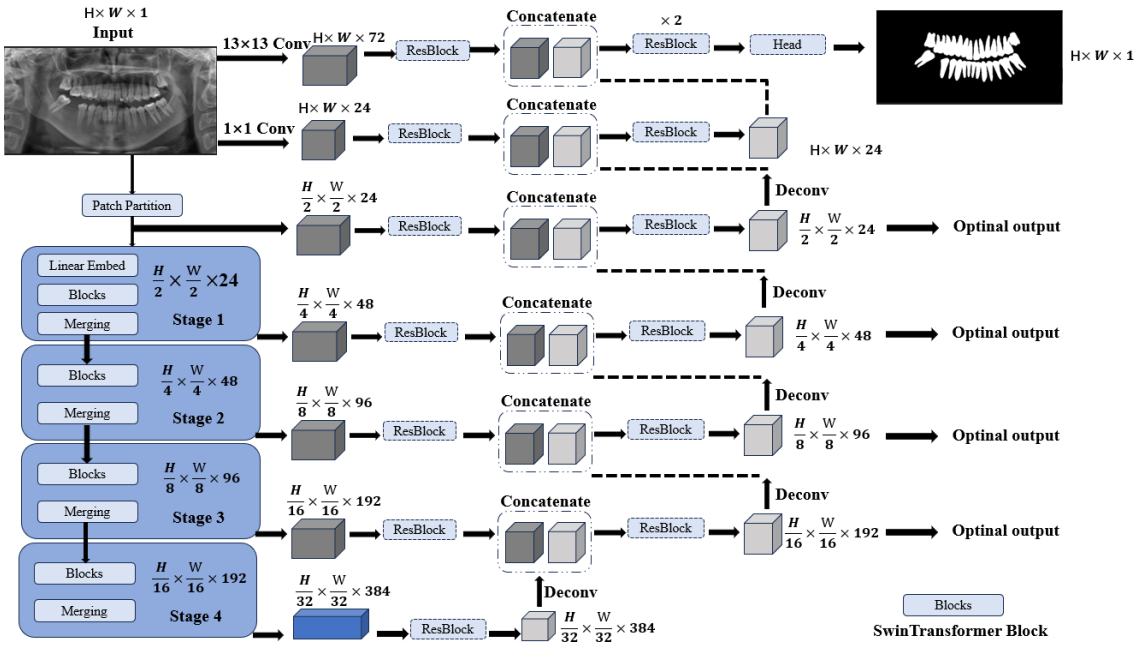


图 4 SwinUNETR_HS 结构示意

训练流程:

我们的训练过程较为普通，大致如下:

1.分折。在初赛中，我们根据官方提供的 2000 张训练集图像进行训练，并把 2000 张图像平均分为 5 折，每折取 1600 张作为训练集，400 张作为验证集，在此基础上训练 5 折不同的模型。与此同时，我们还训练了一折使用所有数据的模型，因此在一整次的训练过程中，我们会训练产生六个网络模型以供后续待介绍的推理。在复赛中，我们根据官方提供的 3000 张训练集图像进行训练，并把 900 张官方标注图像平均分为 5 折，每折取 720 张作为训练集，180 张作为验证集。因此复赛过程中单折的训练图像为 2820 张，验证图像为 180 张。

2.预热。我们跟随了 Kaggle 比赛上前有的 GradualWarmupSchedulerV2，具体代码如下:

```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]
```

图 5 WarmUp 流程

3.优化器及损失函数。我们使用了 AdamW 作为模型训练的优化器，作为学习率的衰退流程，我们使用 torch.optim.lr_scheduler.CosineAnnealingLR。基础学习率为 4e-4，最小学习率为 6e-5。在损失函数上，我们使用 pos_weight 为 2.0 的 BCE 以及 MONAI 中的 Dice 损失作为集合损失函数。
4.训练。我们使用单张 NVIDIA A100 80GB 进行模型训练。训练的 BatchSize 为 24，训练轮数为 72，训练过程中，我们使用自动混合精度以节省显存占用。
5.模型评估。参照给出的评分公式，我们在模型训练的过程中利用 MONAI 库中的 DiceMetric，HausdorffDistanceMetric，MeanIoU 构建了类似的评分标准，并在验证时采用其作为评分标准保模型。但在实践中我们发现，自行构建的评分公式与官方的评分公式指标可能有所差异，譬如本地评估时的最优阈值往往与线上提交时相异，因此本地的验证分数往往只能作为模型效果的粗略评估。

数据增强：

我们尝试了多种数据增强组合策略，均来源于 albumentations 库。我们曾尝试激进的数据增强，但发现其对模型效果持负面影响。我们提交最优模型的数据增强策略如下：

```
# ============== 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),
]
```

图 6 数据增强策略

模型集成：

在众多训练流程结束后，我们得到了为数不少的模型文件。我们发现模型间对测试集的最优推理阈值存在较大差异，因此如何选择模型一直是个棘手的问题。一般而言，我们在进行推

理时选择的模型个数为 6 的倍数——因单一整次训练会产生（五折+一折完整数据训练）六个模型。在“数据准备”环节提及的伪标签生成过程中，我们使用了 12 个模型进行集成，其余情况下我们使用 6 个模型进行集成。
具体而言，在集成推理过程中，我们首先应用图像高，宽方向上的翻转作为测试数据增强（TTA），并得到单个模型的 TTA 结果，随后再将六个模型的 TTA 结果取 Sigmoid，并取最终六个模型预测概率的平均值。

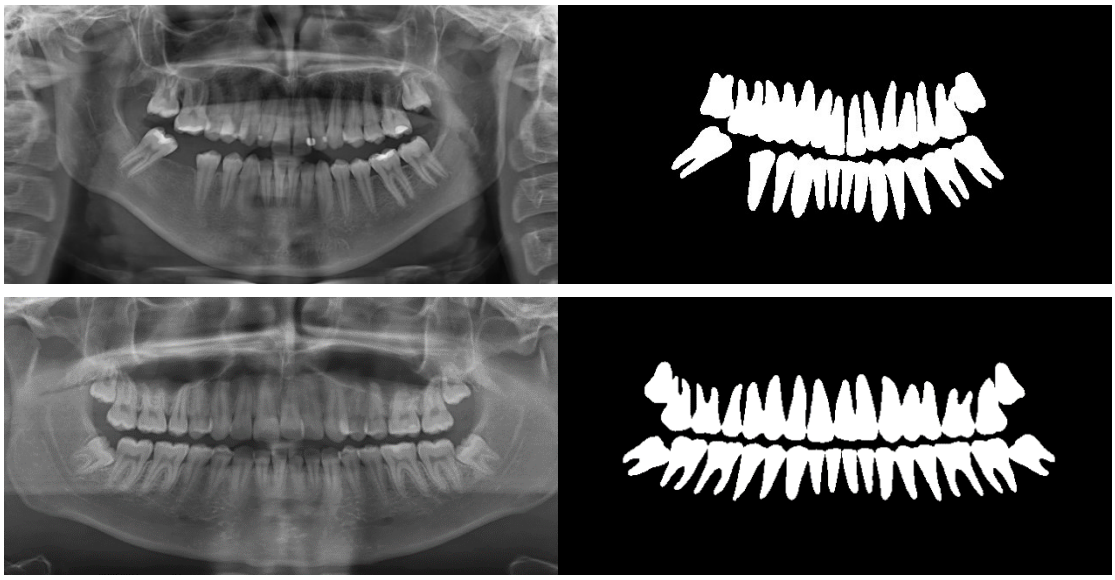
阈值选择:

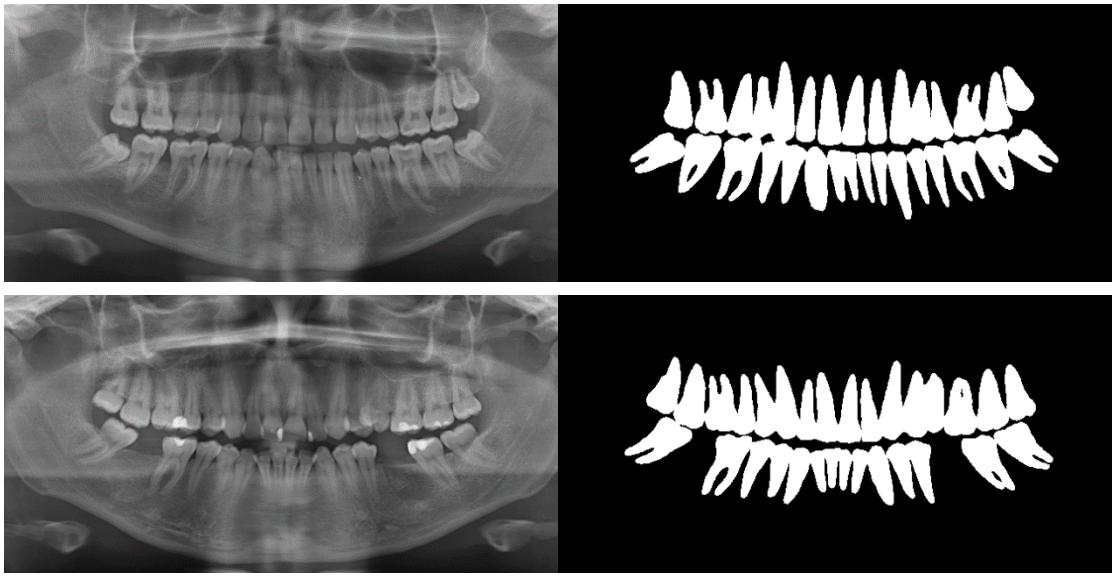
如前所述，我们发现模型间对预测集的最优阈值存在较大差异。在初赛中，我们的最优模型最优阈值大概为 0.2，但在复赛过程中，我们曾得到过最优阈值大概为 0.9 的模型。对于我们提交的最优模型，我们的分割阈值设定为 0.3.

后处理:

由于团队成员时间有限，我们只遵循“数据处理”环节中的 mask，对输出结果的两侧进行了强硬的屏蔽处理。但在先前的 Kaggle 比赛实践中，我们还曾经探索过二值化图像的进一步优化后处理，例如剔除一些较小的区域。我们相信如果使用了妥善的后处理方法，最终输出的结果仍然有进一步提升的空间。

结果展示:





score: 0.9624

dice: 0.9371

iou: 0.9823

hausdorff_distance: 0.0239

参考文献:

[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.