Miracle-boyi 技术报告

任务一分割：

1 数据的预处理

将全部的训练数据放入训练数据集，从中选取 1/3 的数据集作为验证数据集。将训练数据集做如下预处理：

1、首先算出所有训练集的三通道的均值和方差由于是灰色图片三通道相等 mean=（0.226，0.226，0.226），std=（0.160，0.160，0.160）；

2、对训练数据做随机剪裁剪裁尺寸为 800*800、做随机水平反转、做随机垂直反转、做角度旋转 limit=40 发生概率为 p=0.5、做模糊化 blur_limit=3 发生概率 p=0.3、做色彩增强发生概率为 p=0.5；

3、最后转为 tensor 格式并加入归一化操作；

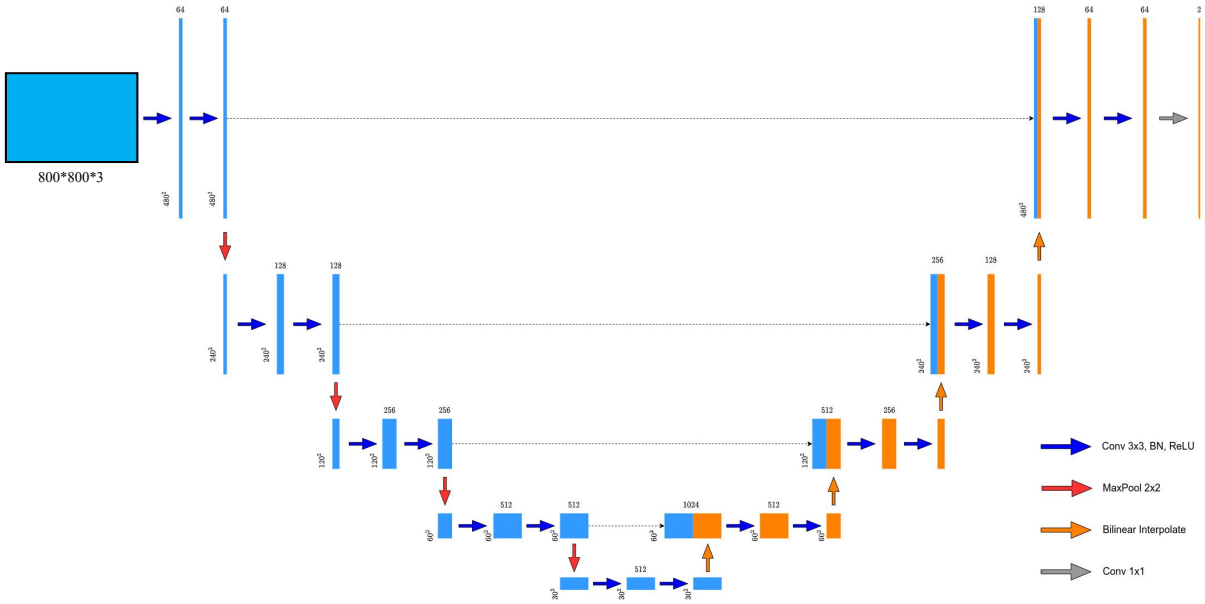
代码如下：

```
class SegmentationPresetTrain:
    def __init__(self, base_size, crop_size, hflip_prob=0.5, vflip_prob=0.5,
                 mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)):
        min_size = int(0.5 * base_size)
        max_size = int(1.2 * base_size)

        trans = [T.RandomResize(min_size, max_size)]

        if hflip_prob > 0:
            # 随机水平翻转
            trans.append(T.RandomHorizontalFlip(hflip_prob),)
        if vflip_prob > 0:
            # 随机垂直翻转
            trans.append(T.RandomVerticalFlip(vflip_prob))
        trans.extend([
            T.RandomCrop(crop_size),
            # 数据增强------------------------------->
            A.Compose([A.Rotate(limit=40, p=0.5),
                       A.OneOf([
                           A.Blur(blur_limit=3, p=0.3),
                           A.ColorJitter(p=0.5),
                       ], p=0.5),
            ]),

            T.ToTensor(),
            T.Normalize(mean=mean, std=std),
        ])
        self.transforms = T.Compose(trans)
```

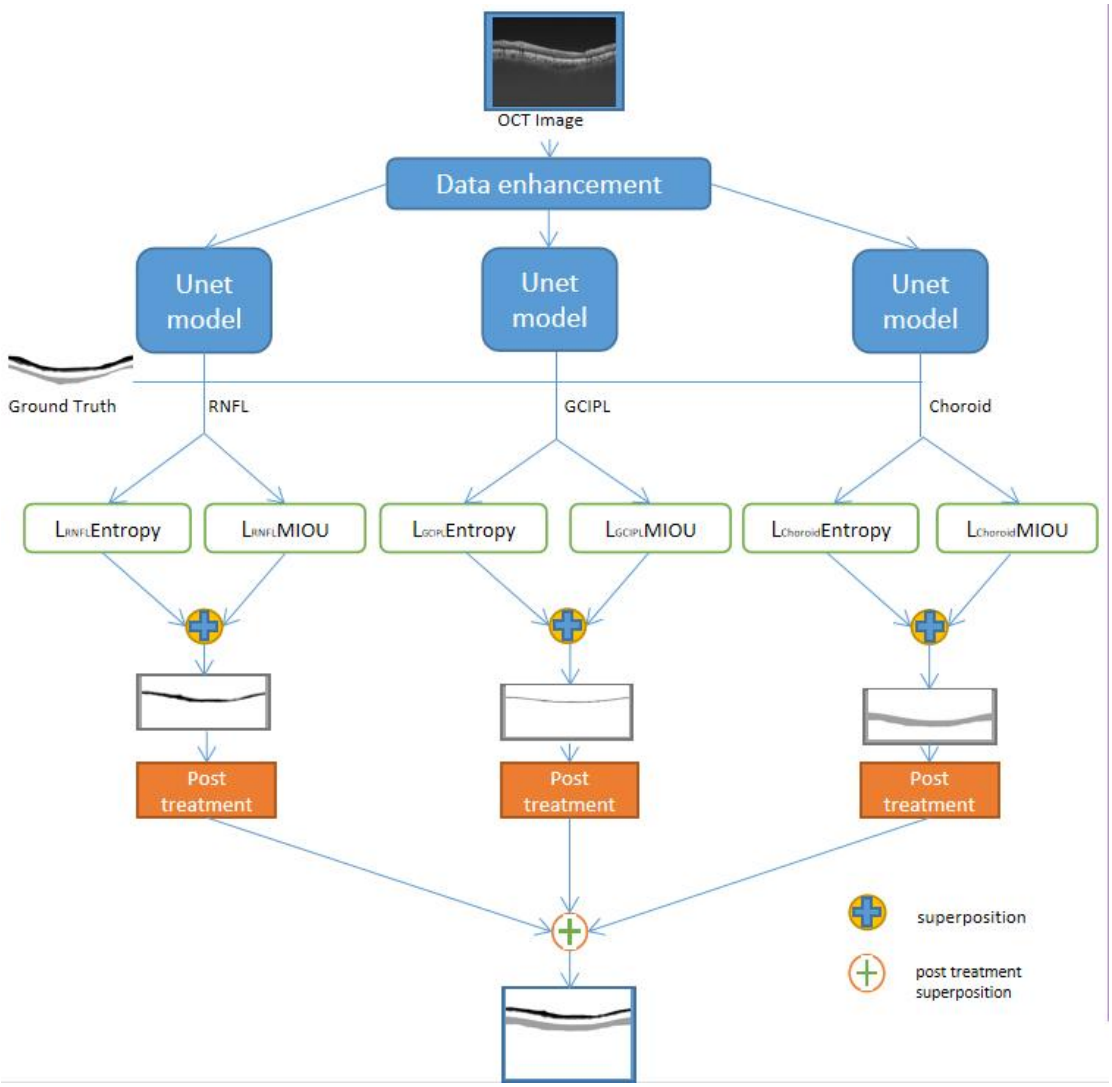
2 模型定义

使用标准的 Unet 网络，所有的上采样全部使用双线性插值，第一次特征提取的 feature map 为 64 层，输入的图片为 800*800，模型图如下：

Unet 模型结构如下：



整体模型结构如下：



以上模型为对三个不同的层进行用同样的模型进行单独训练，这样训练的好处是：1、

可以直接提升准确率；2、对于未合成之前的图片中，只有背景和一个类别，便于进行后处理；3、在对于边界的归属问题在后处理合成的时候可以考虑使用准确率更高的类别填充，也可间接的提高准确率。

3 损失函数和优化器设置

定义损失函数：使用交并比和交叉熵作为损失函数；交并比用于主要对焦分割的准确度，交叉熵主要对焦系统整体的混乱程度。

代码如下：

```
def criterion(inputs, target, loss_weight=None, num_classes: int = 2, dice: bool = True, ignore_index: int = -100):
    losses = {}
    for name, x in inputs.items():
        loss = nn.functional.cross_entropy(x, target, ignore_index=ignore_index, weight=loss_weight)
        if dice is True:
            dice_target = build_target(target, num_classes, ignore_index)
            loss += dice_loss(x, dice_target, multiclass=True, ignore_index=ignore_index)
        losses[name] = loss
    if len(losses) == 1:
        return losses['out']
    return (losses['out'] + losses['out3'] + losses['out2'] + losses['out1'])/4
```

优化器：使用 SGD 作为优化器，传入参数有 lr=0.01，momentum=0.9，weight_decay=1e-4。

代码如下：

```
    params_to_optimize = [p for p in model.parameters() if p.requires_grad]
    # 优化器的选择
    optimizer = torch.optim.SGD(
        params_to_optimize,
        lr=args.lr, momentum=args.momentum, weight_decay=args.weight_decay
    )
```

4 模型训练

由于分割目标有 4 个类别，分别是背景 255，前景 0、80、160，这里采用三个模型对其分别进行训练，三个模型均共用同一套网络模型以及所有的预处理操作。

学习率 lr 的更新如下：

```python
def create_lr_scheduler(optimizer,
                        num_step: int,
                        epochs: int,
                        warmup=True,
                        warmup_epochs=1,
                        warmup_factor=1e-3):
    assert num_step > 0 and epochs > 0
    if warmup is False:
        warmup_epochs = 0

    def f(x):
        """
        根据step数返回一个学习率倍率因子,
        注意在训练开始之前，pytorch会提前调用一次lr_scheduler.step()方法
        """
        if warmup is True and x <= (warmup_epochs * num_step):
            alpha = float(x) / (warmup_epochs * num_step)
            # warmup过程中lr倍率因子从warmup_factor -> 1
            return warmup_factor * (1 - alpha) + alpha
        else:
            # warmup后lr倍率因子从1 -> 0
            # 参考deeplab_v2: Learning rate policy
            return (1 - (x - warmup_epochs * num_step) / ((epochs - warmup_epochs) * num_step)) ** 0.9

    return torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda=f)
```

像素值为 0 和 160 的类的训练均采用 epoch=1000，由于学习率的是递减的所以经过测试 epoch=400 轮的效果是最佳的，但是不可直接训练 epoch=400 轮，这样的效果没有训练 1000 轮保存 400 轮内的最佳模型效果好；像素值为 80 类的面积较小且特征不明显，所以收敛略快一点且容易过拟合，则采用训练 epoch=800 轮，但保存 400 轮内的最好模型。

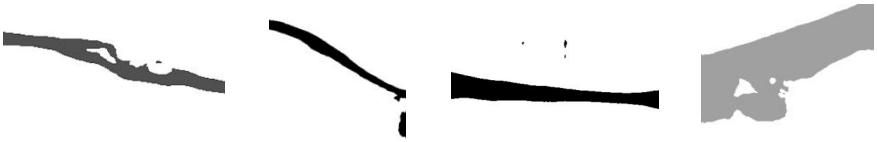
由于分开训练时的背景和前景面积比相差过大，所以采用不同的权重，背景和前景的损失权重比为 torch.as_tensor（1.0，3.0）

训练的 batch_size=6.

5 实验结果及后处理

用三个模型分别预测三个类别，测试集有 100 张图片，则对应的生成 300 张图片。

对于得到的 300 张图片的结果观察可以发现，有少量的小孔洞会被错误的预测，此时采用空洞填充，用联通区面积的方式来去除孔洞，对于像素点 0 类别，去除前景和背景面积小于 800 个像素点的孔洞；对于像素点 80 的类别，去除前景面积小于 200 背景面积小于 100 的孔洞；对于像素点 160 的类别，去除前景面积小于 5000 背景面积小于 3000 的孔洞。样例如下：



1、 2、 3、 4、

具体操作如下：

```
def fillHole(pre_mask, classes, size_b, size_v):
    # pre_mask是包含有黑色孔洞的二值图像
    # 这里将原图像黑白反转
    pre_mask_rever = pre_mask <= classes
    # 这里的min_size是表示需要删除的孔洞的大小，可以根据需要设置
    pre_mask_rever = morphology.remove_small_objects(pre_mask_rever, min_size=size_v)
    # 将删除了小连通域的反转图像再盖回原来的图像中
    pre_mask[pre_mask_rever == False] = 255
    pre_mask[pre_mask_rever == True] = classes

    pre_mask_rever = pre_mask > classes
    pre_mask_rever = morphology.remove_small_objects(pre_mask_rever, min_size=size_b)
    pre_mask[pre_mask_rever == False] = classes
    pre_mask[pre_mask_rever == True] = 255

    return pre_mask
```

去除孔洞后，将三个类别合成到一张图片上，在合成的时候，由于像素值为 0 和像数值为 80 的类别两者是紧挨着的，所以会出现两种明显的错误情况：1、临界区域同时属于两个类别；2、临界区域同时不属于两个区域。（情况 1 不便于以图像的方式表现，以下展示情况 2）

对于此类情况的处理方式如下：由于像素值为 80 的类别的准确率最低，所以在合成的时候将像素值为 0 的和像素值为 160 的直接全部重叠至像素值为 80 的类别的图上。合成后会发现，像素值为 0 的类别和像素值为 80 的类别中间可能会有缝隙，由于像素值为 0 的类别和像素值为 80 的类别是紧挨着的，所以采用准确率低的类别进行填充，即使用像素值为 0 的类别进行填充，具体操作如下：

```python
def fill(img, fill_num):
    for i in range(len(img[0])):
        start = False
        end = False
        place = []
        for j in range(len(img)):
            if img[j, i] == 0:
                start = True
            if img[j, i] == 255 and start == True:
                place.append([j, i])
            if img[j, i] == 0 and len(place) != 0:
                place = []
                start = False
            if img[j, i] == 80:
                end = True
                break
        if end == True:
            for k in range(len(place)):
                img[place[k][0], place[k][1]] = fill_num
    return img
```

最后合成后生成的图片则为最终的结果图片。

任务二分类：

1 数据的预处理

将全部的训练数据放入训练数据集，从中选取 1/3 的数据集作为验证数据集。将训练数据集做如下预处理：

1、随机剪裁尺度为 224、随机反转

2、转为 tensor 格式，并归一化 mean=（0.485，0.456，0.406）std=（0.229,0.224,0.225）

代码如下：

```python
class SegmentationPresetTrain:
    def __init__(self, base_size, crop_size, hflip_prob=0.5, vflip_prob=0.5,
                 mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)):
        min_size = int(0.5 * base_size)
        max_size = int(1.2 * base_size)

        trans = [T.RandomResize(min_size, max_size)]

        if hflip_prob > 0:
            # 随机水平翻转
            trans.append(T.RandomHorizontalFlip(hflip_prob),)
        if vflip_prob > 0:
            # 随机垂直翻转
            trans.append(T.RandomVerticalFlip(vflip_prob))
        trans.extend([
            T.RandomCrop(crop_size),
            # 数据增强---------------------------------->
            A.Compose([A.Rotate(limit=40, p=0.5),
                       A.OneOf([
                           A.Blur(blur_limit=3, p=0.3),
                           A.ColorJitter(p=0.5),
                       ], p=0.5),
            ]),

            T.ToTensor(),
            T.Normalize(mean=mean, std=std),
        ])
        self.transforms = T.Compose(trans)
```

将全部的训练数据放入训练数据集，从中选取 1/3 的数据集作为验证数据集。将训练数据集做如下预处理：

3、随机剪裁尺度为 224、随机反转

4、转为 tensor 格式，并归一化 mean=（0.485，0.456，0.406）std=（0.229,0.224,0.225）

代码如下：

```python
"train": transforms.Compose([transforms.RandomResizedCrop(224),
                             transforms.RandomHorizontalFlip(),
                             transforms.ToTensor(),
                             transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])]),
```

2 模型定义

使用的网络为标准的 resnet34 网络，使用官方模型，未做任何改动。

3 损失函数和优化器设置

损失函数：使用交叉熵作为损失函数。

代码如下：

```python
# define loss function
loss_function = nn.CrossEntropyLoss()
```

优化器：使用 Adam 优化器，lr=0.0001。

代码如下：

```
# construct an optimizer
params = [p for p in net.parameters() if p.requires_grad]
optimizer = optim.Adam(params, lr=0.0001)
```

4 模型训练

batch_size=16，epoch=30 轮，保存 30 轮中效果最好的模型。

5 实验结果及后处理

对于生成的模型直接进行预测，对预测的结果不做 softmax 操作，因为模型预测了两个类，0 为不是青光眼，1 为是青光眼，所以直接将网络模型输出的第二个结果浮点型小数保存至 CSV 文件中。