实验四

脑部MRI图像分割

2022年10月12日

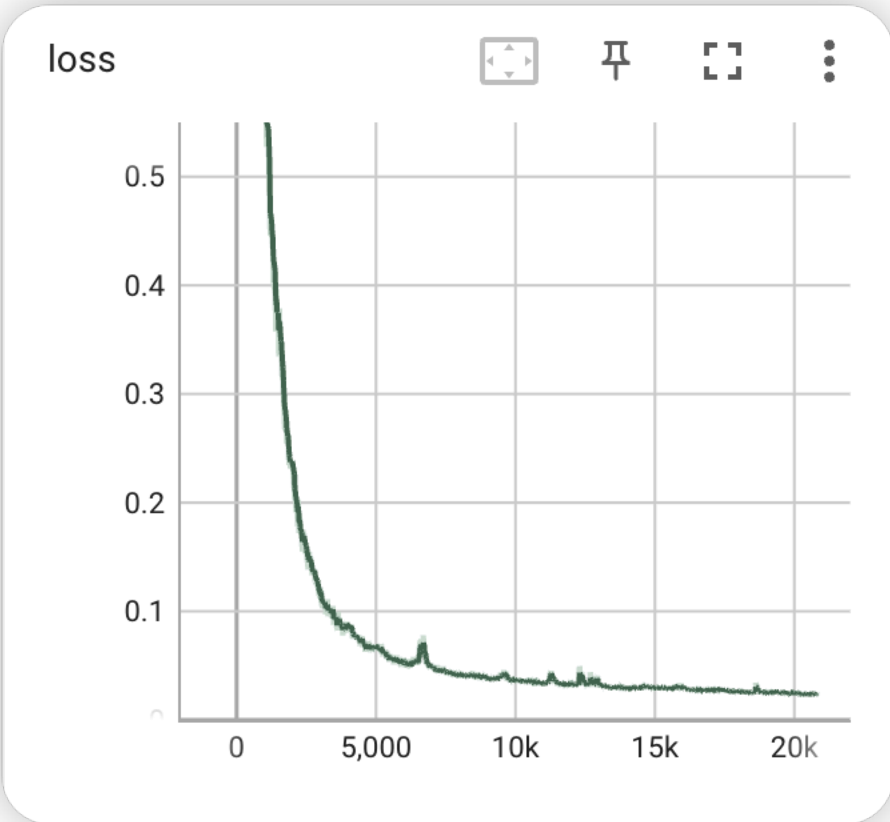
1. Unet 基准

使用DiceLoss作为损失函数，Adam作为优化器

dsc为0.9108

```
[5]   unet = UNet(in_channels=Dataset.in_channels, out_channels=Dataset.out_channels)
      optimizer = optim.Adam(unet.parameters(), lr=args.lr)
      train(args, optimizer, unet)
      epoch 95 | val_loss: 0.234698216120402
      epoch 95 | val_dsc: 0.8143511399818392
      epoch 96 | val_loss: 0.24109460626329696
      epoch 96 | val_dsc: 0.8104362380982385
      epoch 97 | val_loss: 0.2747120289575486
      epoch 97 | val_dsc: 0.9029547921997331
      epoch 98 | val_loss: 0.24866984287897745
      epoch 98 | val_dsc: 0.9009163097958627
      epoch 99 | val_loss: 0.25969226019723074
      epoch 99 | val_dsc: 0.8119596617335582
      epoch 100 | val_loss: 0.2457147552853539
      epoch 100 | val_dsc: 0.8102490039128755
      Best validation mean DSC: 0.910859
```

训练过程的loss变化如下图所示：



验证的dsc与loss变化如下图所示

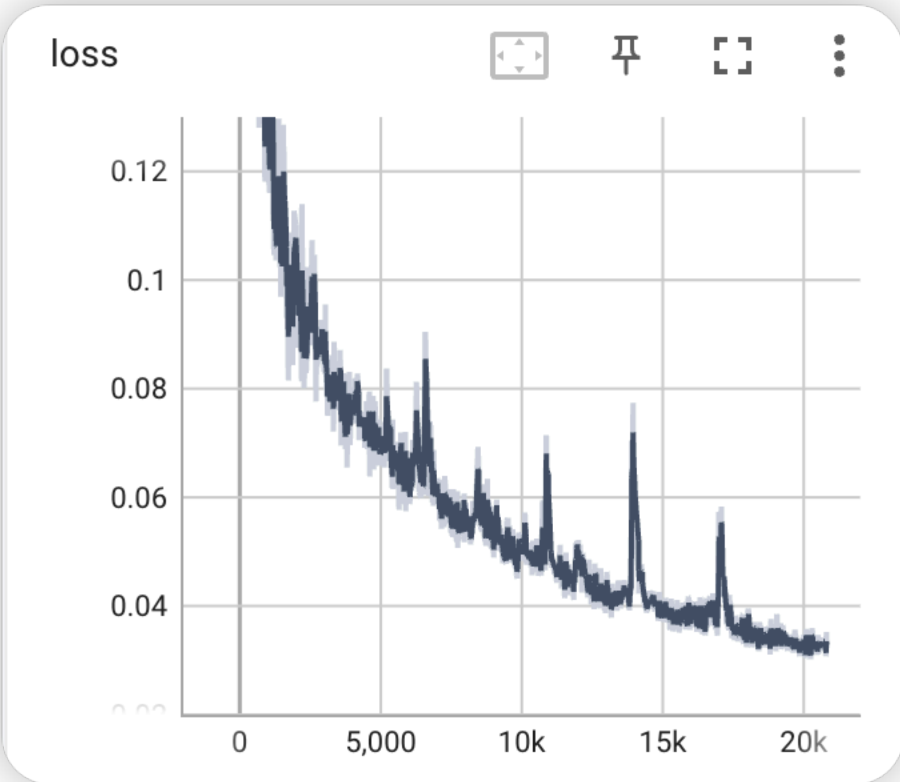
考虑到Adam优化器可能会有不收敛的情况，下面用sgd训练一次。

2. 使用SGD作为优化器

Dsc为0.806，不如基准的Unet模型。

```
[6]  args.weights = './sgdweights'
     args.logs = './sgdlogs'
     args.lr = 0.001
     unet = UNet(in_channels=Dataset.in_channels, out_channels=Dataset.out_channels)
     optimizer = optim.SGD(unet.parameters(), lr=args.lr, weight_decay=0.0001, momentum=0.9)
     train(args, optimizer, unet)

     epoch 95 | val_loss: 0.38191692034403485
     epoch 95 | val_dsc: 0.7983014418467671
     epoch 96 | val_loss: 0.38078703199114117
     epoch 96 | val_dsc: 0.7966183774336605
     epoch 97 | val_loss: 0.3766747258958362
     epoch 97 | val_dsc: 0.7947089589775882
     epoch 98 | val_loss: 0.37810280209495906
     epoch 98 | val_dsc: 0.7920366863824088
     epoch 99 | val_loss: 0.38482984758558725
     epoch 99 | val_dsc: 0.7890226666504566
     epoch 100 | val_loss: 0.38565200567245483
     epoch 100 | val_dsc: 0.7790618344346647
     Best validation mean DSC: 0.806657
```

原因可能是初始学习率设置过大了，loss看起来并不收敛。



3. 修改网络结构

将原来的VGG块替换成具有ResNet结构。

```python
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()

        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels * BasicBlock.expansion, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels * BasicBlock.expansion)
        )
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != BasicBlock.expansion * out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * BasicBlock.expansion, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels * BasicBlock.expansion)
            )

    def forward(self, x):
        return nn.ReLU(inplace=True)(self.residual_function(x) + self.shortcut(x))
```

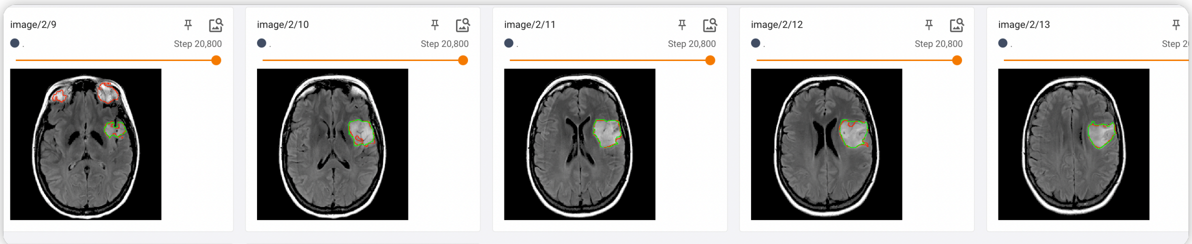
```python
class ResUNet(nn.Module):

    def __init__(self, in_channels=3, out_channels=1, init_features=32):
        super(ResUNet, self).__init__()

        features = init_features
        self.encoder1 = VGGBlock(in_channels, features, features)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.encoder2 = BasicBlock(features, features * 2)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.encoder3 = BasicBlock(features * 2, features * 4)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.encoder4 = BasicBlock(features * 4, features * 8)
        self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.bottleneck = BasicBlock(features * 8, features * 16)

        self.upconv4 = nn.ConvTranspose2d(
            features * 16, features * 8, kernel_size=2, stride=2
        )
        self.decoder4 = VGGBlock((features * 8) * 2, features * 8, features * 8)
        self.upconv3 = nn.ConvTranspose2d(
            features * 8, features * 4, kernel_size=2, stride=2
        )
        self.decoder3 = VGGBlock((features * 4) * 2, features * 4, features * 4)
        self.upconv2 = nn.ConvTranspose2d(
            features * 4, features * 2, kernel_size=2, stride=2
        )
        self.decoder2 = VGGBlock((features * 2) * 2, features * 2, features * 2)
        self.upconv1 = nn.ConvTranspose2d(
            features * 2, features, kernel_size=2, stride=2
        )
        self.decoder1 = VGGBlock(features * 2, features, features)

        self.conv = nn.Conv2d(
            in_channels=features, out_channels=out_channels, kernel_size=1
        )
```

训练结果为：

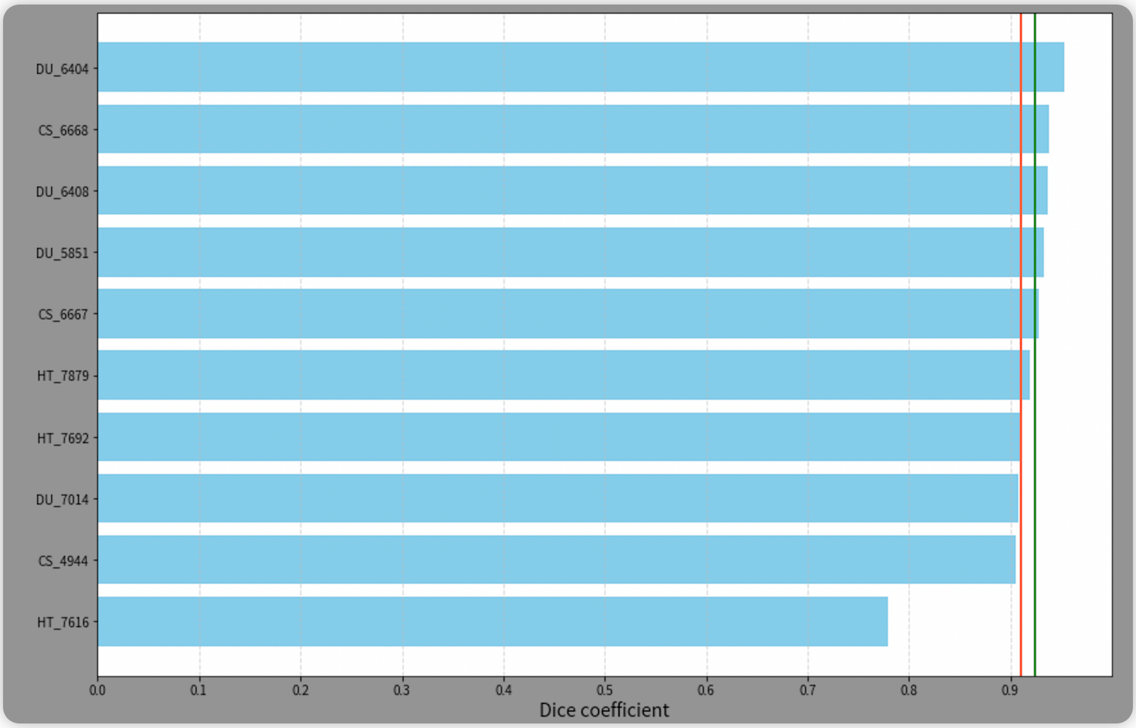
```
[7]  args.weights = './resweights'
     args.logs = './reslogs'
     resunet = ResUNet(in_channels=Dataset.in_channels, out_channels=Dataset.out_channels)
     optimizer = optim.Adam(resunet.parameters(), lr=args.lr)
     train(args, optimizer, resunet)
     epoch 95 | val_loss: 0.25042128279095605
     epoch 95 | val_dsc: 0.8151411748797817
     epoch 96 | val_loss: 0.2808646389416286
     epoch 96 | val_dsc: 0.8131370658647242
     epoch 97 | val_loss: 0.22985850061689103
     epoch 97 | val_dsc: 0.8121397950643188
     epoch 98 | val_loss: 0.2270130855696542
     epoch 98 | val_dsc: 0.9045896768351511
     epoch 99 | val_loss: 0.25673952556791757
     epoch 99 | val_dsc: 0.8115125565824132
     epoch 100 | val_loss: 0.30242357935224226
     epoch 100 | val_dsc: 0.8128705982450899
     Best validation mean DSC: 0.911568
```

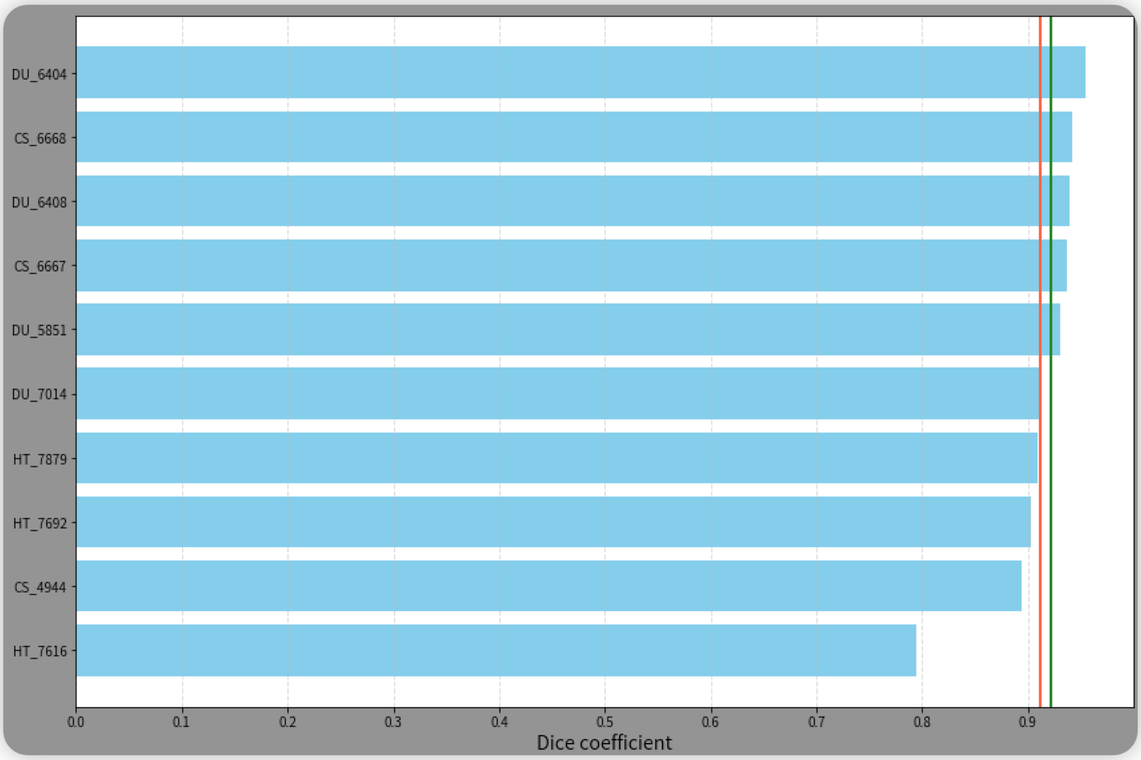
Dsc为0.9115，相较于基准的0.9108，有少量的提升。

部分图像的分割效果为：



4. 测试结果





第一个是基准Unet测试的结果，第二个是改造后的ResUnet的测试结果，可以看出，在部分数据集上，比如HT_7616，ResUnet的测试结果更好。