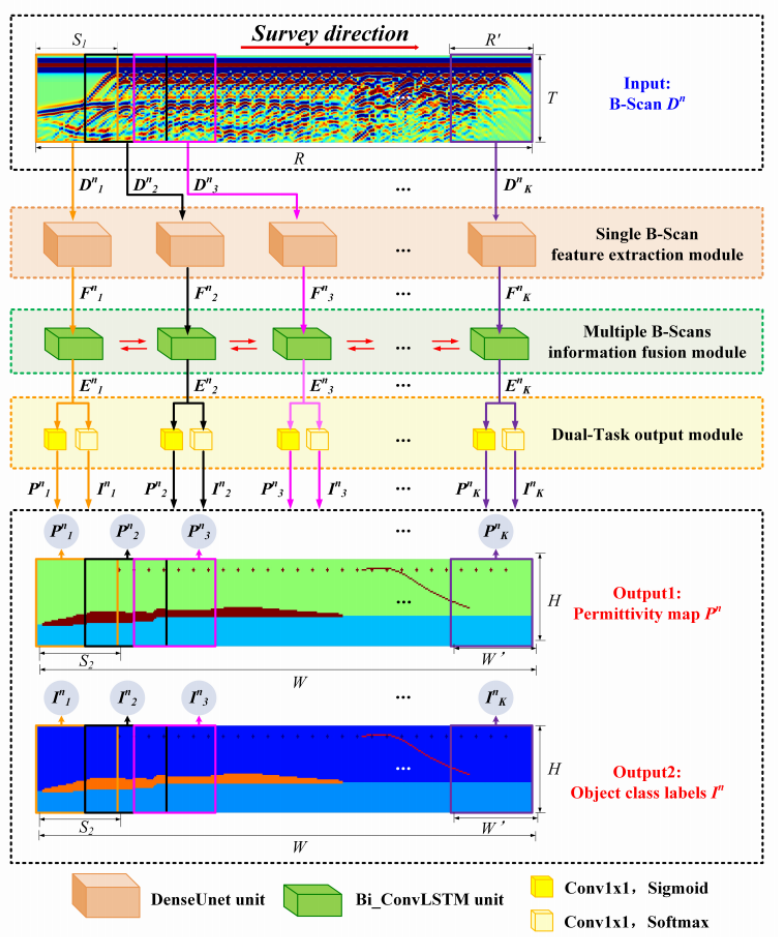
GPRI2Net实现文档

1. 核心框架

a. 

   i. 对应文件GPRI2Net.py
   ii. 第一层每一个橙色都是一个DenseUNet，详细见b.
   iii. 第二层是由两层ConvLSTM组成，详细见c.
   iv. 第三层是两个1x1Conv用于把通道减少到所需的个数，P图为衍生图，只有一个通道，I图为分类图，有C个通道，C指不同种类的介质或者缺陷

b.

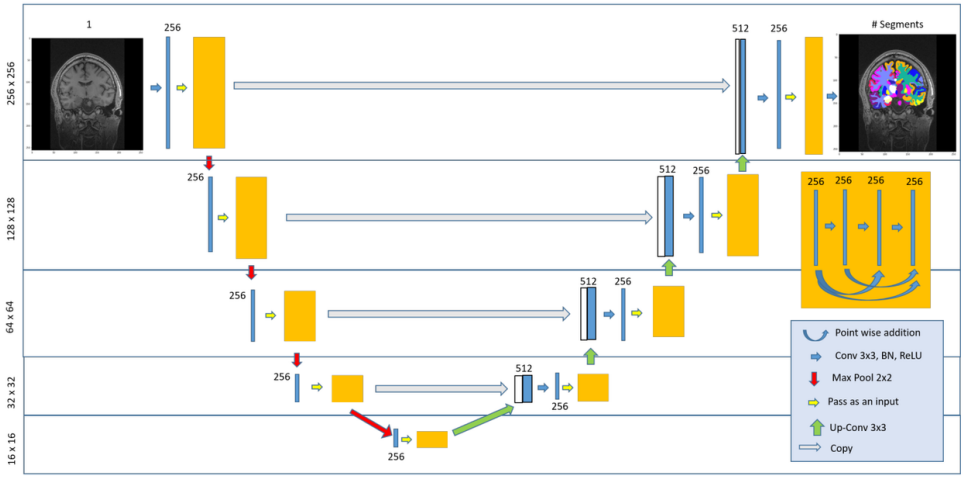
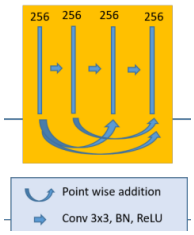


Figure 1: Schematic Diagram of DenseUNet

i. 对应文件DenseUNet.py

ii. 这张图里面的黄橙色方块是一个DenseBlock

1. 

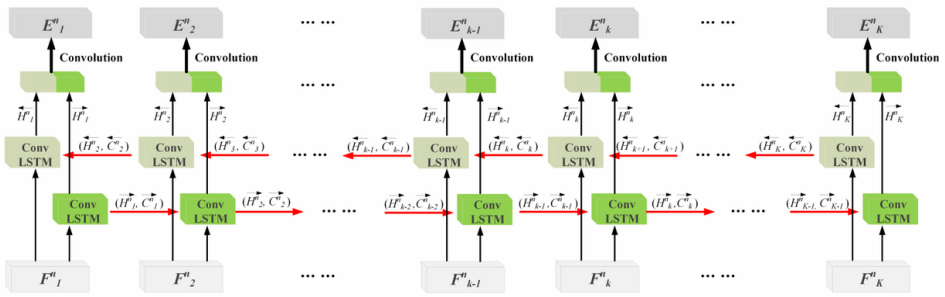
2. 对应类DenseBlock
3. 每后面一个层就会累加前面的层的结果，不是concat，对每一个元素做加法
    a. 比如第三层就要加上第一，二层，第四层加上第一，二，三层
4. 其中每一层到后一层之间要经过Conv 3x3 BN Relu（对应类ConvToBNToRelu）三个步骤
    a. 第一层 -> 第二层， 第二层 -> 第三层，第三层 -> 第四层

iii. 红色箭头对应代码里面的Downsampling（对应类DownSample）

iv. 黄色箭头指直接把上一个的输出输入到下一个块里面

v. 绿色箭头对应的upsampling，实际上是transposed convolution的操作（对应类UpSampleAndConcat，但这个类里面同时把每个downsampling的denseblock的输出和upsample的输出concat到一起，如图灰色长箭头所示）

c.



i. 对应文件ConvLSTM.py

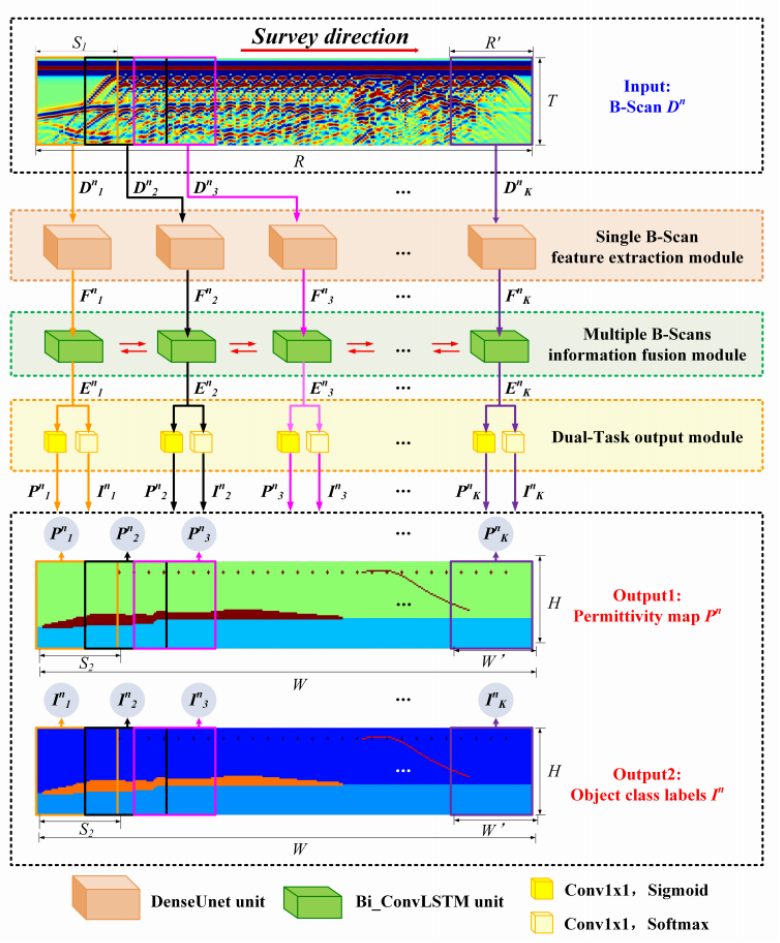
ii. 文件包含两个类：

1. ConvLSTMUnit
    a. 对应图标里面绿色的小盒子
    b. 基本由公式组成

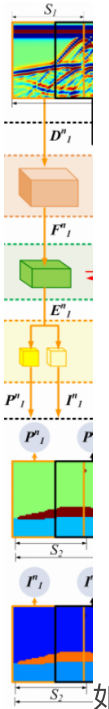
$$
\begin{aligned}
i_k^n &= \delta\left(W_{fi} * F_k^n + W_{hi} * H_{k-1}^n + b_i\right) \\
f_k^n &= \delta\left(W_{ff} * F_k^n + W_{hf} * H_{k-1}^n + b_f\right) \\
o_k^n &= \delta\left(W_{fo} * F_k^n + W_{ho} * H_{k-1}^n + b_o\right) \\
C_k^n &= f_k^n \odot C_{k-1}^n + i_k^n \odot \tanh\left(W_{fc} * F_k^n + W_{hc} * H_{k-1}^n + b_c\right) \\
H_k^n &= o_k^n \odot \tanh\left(C_k^n\right) \qquad (4)
\end{aligned}
$$

    c.
    d. 其中 δ 为sigmoid函数
    e. “*”为convolution操作符
    f. ⊙为hadamard product（对两个tensor中的每个元素一一对应相乘，elementwise product）
    g. 所有的W为3x3 Conv 拥有64个通道
    h. 所有的b都是bias
2. ConvLSTM
    a. 由红色箭头串联起来的多个ConvLSTMunit组成的一层，每一个ConvLSTMUnit的输出都会作为下一个ConvLSTMUnit输入的数据之一
    b. 拥有两个方向"forward"和"backward"对应图中的两层
    c. 两层的输入会concat到一起（共128个通道），最后通过一个convolution压缩到64个通道

2. 整个模型运行流程

a. 

b. 整个模型除开红色箭头竖着看，是一条线一条线产出的如下图



c. 如果需要，这个模型可以横向无限扩展下去

d. 一个长图会被分成很多个正方形图片，论文中采用128x128的大小作为输入D

e. D被输入到DenseUNet输出128x128单通道图片，作为基本特征提取，输出为F

f. F输入到两个ConvLSTMUnit，一个"forward"，一个"backward"如上面提及1.c，输出E
g. E输入到两个不同的层里面（论文称为双任务输出层），一个为1x1Conv + sigmoid作为衍生图输出，一个为1x1Conv + softmax作为分类图输出
h. 模型整体输入为[D1,D2,.....,Dn]，通过整个第一层输出为[F1,F2,F3,.....,Fn]，通过整个第二层输出为[E1,E2,E3,.....,En]，最后通过双任务输出层输出两个列表，[P1,P2,P3,.....,Pn]为衍生图，[I1,I2,I3,.....,In]为分类图

3. 损失函数
   a. LossFunction.py
   b. $$L = \lambda_1 L_{\mathrm{MAE}} + \lambda_2 L_{\mathrm{SSIM}} + \lambda_3 L_{\mathrm{LZS}}$$

$$L_{\mathrm{MAE}} = \frac{1}{H * W} \sum_{h=1}^{H} \sum_{w=1}^{W} \left| P_k^n(h, w) - \overline{P_k^n}(h, w) \right|$$

   c. $$L_{\mathrm{SSIM}} = -\frac{1}{H * W} \sum_{h=1}^{H} \sum_{w=1}^{W} \mathrm{SSIM}\left(P_k^n(x_{(h,w)}), \overline{P_k^n}(y_{(h,w)}) \right)$$ 论文对SSIM（图片相似度）的描述并不详细，目前采用网上已有开源代码库实现，本文实现的描述的SSIM与SSIM原论文有所不同这里直接用负号，SSIM原论文使用1 - SSIM()作为loss function
   d. L_{LZS} was employed to optimize the mean intersection over union (MIoU) of the identification results, and L_{LZS} is expressed as follows:

$$L_{\mathrm{LZS}} = \frac{1}{C} \sum_{c=1}^{C} \overline{\Delta J_c}(m_c) \qquad (3)$$

where C is the total number of object classes in the datasets, $\overline{\Delta J_c}$ is the Lovász extension to the Jaccard index of the class c, and m_c is a vector of pixel identification errors for class c. 对LZS（lovasz softmax）描述也不详细，目前采用网上已有的开源代码库实现。

4. 输入输出处理
   a. 原论文的数据为700个采样点，250道数据，共5M，也有10M的数据，共500到数据
   b. 输出图为60 x 500对应5m的图，60 x 1000对应10M的图
   c. 700 x 50为一个D数据，resize到128x128进入模型