**The methods and results of the traditional radiomics analysis**

To investigate the advantage of our proposed method, we conducted traditional radiomics analysis using the same full data of deep learning radiomics. The traditional radiomic features were automatically extracted from PyRadiomics1 and implemented in Python (version 3.7.6; <https://www.python.org/>) for each sinus. The details of these features were available in the documentation for PyRadiomics (<http://PyRadiomics.readthedocs.io/en/latest/>). For each sample, a total of 1218 features including intensity features, shape features, texture features, and wavelet features were extracted from the segmented sinus regions. For features in the training cohort, each feature for a specific patient was subtracted by the mean value and divided by standard deviation value from this cohort. The same normalization method was applied to features in the testing cohort using the mean values and standard deviation values calculated based on the training cohort. Least absolute shrinkage and selection operator (Lasso) was used to select the most important features. Finally, 10 features from the CT imagines were extracted, consisting of 3 first-order statistical features, 4 grey-level size zone matrix features, 1 grey level dependence matrix, 2 gray level co-occurrence matrix. Logistic regression model was used for the recurrence prediction. A risk-score was built based on the coefficients of the selected features. The risk-score was then combined with clinical factors for nomogram construction. Finally, we got an AUC of 0.496 for the radiomics model and an AUC of 0.690 for the combined nomogram (Figure S1).

**The details of our proposed network**

Let be the input of our proposed network where  denotes the spatial height, width, and depth of the input volume, respectively, and  is the number of input channels. For each CT scan in this study, the input channel number  is 1. We crop each CT scan to a size of  as . We first use a convolutional layer with a kernel size of  to produce the initial feature maps with 16 channels. Then, four residual convolution blocks and three downsampling layers are used to gradually generate feature maps . We refer to the depth of the residual convolution blocks as level. Each level consists of two convolution blocks and a residual connection between the input and output of this level. Each convolution block consists of a batch normalization, leaky ReLU, and  convolutional layer. For each downsampling layer, a max-pooling operation and  convolutional layer are used to reduce the resolution of the feature maps and double the number of feature channels. Finally, at the end of the encoder, we obtain .

The decoder consists of multiple upsampling operations and residual convolution blocks to output the final sinus segmentation mask. We first perform two convolutional blocks on . Then, the upsampling operation and residual convolution blocks are alternately used to gradually generate high-resolution semantic feature maps. Each upsampling operation consists of a trilinear interpolation operation to upsample the low-resolution feature map and a convolution to reduce the channel number. The spatial information lost due to downsampling in the encoder can be restored by skip connections that skip features of the same resolution from the decoder to the corresponding encoder level. Finally, a  convolution followed by a softmax function is applied to generate the segmentation mask.

For , we design a classification module for the recurrent prediction task. As illustrated in Figure S2 the feature maps are transformed to the same size in each channel using global average pooling (GAP) and global max pooling (GMP). Then, these two types of pooled features are fused by concatenation:



where  is the fusion features. We then use three fully connected layers composed of 256, 64, and 2 neurons to gradually extract features. Finally, we use a softmax function to generate the final classification probability.

For the segmentation, we use generalized Dice loss (GDL) to learn the shape similarity between the segmentation result and ground truth, which is denoted as



where  is the output of the decoder, and  is the ground truth.  denotes the classes, and  denotes the voxels.  is the weight that balances the foreground and the background. For the classification task, we use the cross-entropy loss, which can be denoted as:



where  is the true label and is the softmax probability.

The final multitask loss is the linear combination of the segmentation loss and the classification loss, which is defined as



where  is the multitask loss and  is the learnable balance weight between  and . By optimizing the multitask loss, the segmentation task and the classification prediction task can be jointly trained.

The proposed network is built with PyTorch and runs on an NVIDIA V100 GPU. We employ the Adam optimizer with an initialized learning rate set to 2e-4. The bath size is set to 4, and the maximum number of epochs is set to 300. Figure S2 shows the architecture of our proposed network.