I would recommend you to use Dice loss when faced with class imbalanced datasets, which is common in the medicine domain, for example. Also, Dice loss was introduced in the paper "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation" and in that work the authors state that Dice loss worked better than multinomial logistic loss with sample re-weighting

Dice Loss is widely used in medical image segmentation tasks **to address the data imbalance problem**. However, it only addresses the imbalance problem between foreground and background yet overlooks another imbalance between easy and hard examples that also severely affects the training process of a learning model.

If you have a 2 class problem, output only 1 channel, use a sigmoid function (outputs values between 0 and 1). Then you can calculate your dice loss with output (continuous values) and target(single channel one-hot-encoded, discrete values). If your network outputs 2 channels use a softmax function and calculate your loss with your output (continous values) and target (2 channel one-hot-encoded). The former is preferred, as you will have less parameters.

Model Unet 2 :

# Data

error = 20

Area = 100

image\_shape = (128, 128)

batch\_size = 2

# Model

input\_channels = 3

num\_classes = 3

learning\_rate = 5e-4

epochs = 101

loss\_fn  =  DiceLoss()

folder\_name = 'model\_unet\_2'

Model Unet 1:

# Data

error = 25

Area = 80

learning\_rate = 1e-3

image\_shape = (128, 128)

batch\_size = 2

# Model

input\_channels = 3

num\_classes = 3

learning\_rate = 1e-3

epochs = 101

loss\_fn  =  DiceLoss()

folder\_name = 'model\_unet\_1'

'''

UNET

For both models: DICE LOSS FUNCTION

Model 4

beta = 10

epochs = 101

Model 5

beta = 20

epochs = 301

Model 6

LOSS Function: nn.BCEWithLogitsLoss(size\_average = False, reduce=False, reduction=None)

beta = 20

epochs = 101

'''

# beta  = 10

# epochs = 101

from unet\_3block\_conv import \*

from outils\_prepro import \*

from data\_loader\_seg import \*

from model\_prob\_unet\_init import \*

beta = 20

epochs = 301

input\_channels = 3

num\_classes = 3

filters = 8

if filters == 8:

    featureDim = 16384

if filters ==4:

    featureDim = 8192

z\_dim = 10

image\_shape = (128, 128)

net = Probabilistic\_UNET(input\_channels, num\_classes, filters, z\_dim, image\_shape, featureDim)

net.to(device)

optimizer = torch.optim.Adam(net.parameters(), lr=1e-4, weight\_decay=0)

I would recommend you to use Dice loss when faced with class imbalanced datasets, which is common in the medicine domain, for example. Also, Dice loss was introduced in the paper "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation" and in that work the authors state that Dice loss worked better than mutinomial logistic loss with sample re-weighting

**Semantic Segmentation**

* *BCEWithLogitsLoss* (binary cross-entropy)
* *DiceLoss* (standard DiceLoss defined as 1 - DiceCoefficient used for binary semantic segmentation; when more than 2 classes are present in the ground truth, it computes the DiceLoss per channel and averages the values)
* *BCEDiceLoss* (Linear combination of BCE and Dice losses, i.e. alpha \* BCE + beta \* Dice, alpha, beta can be specified in the loss section of the config)
* *CrossEntropyLoss* (one can specify class weights via the weight: [w\_1, ..., w\_k] in the loss section of the config)
* *PixelWiseCrossEntropyLoss* (one can specify per pixel weights in order to give more gradient to the important/under-represented regions in the ground truth)
* *WeightedCrossEntropyLoss* (see 'Weighted cross-entropy (WCE)' in the below paper for a detailed explanation)
* *GeneralizedDiceLoss* (see 'Generalized Dice Loss (GDL)' in the below paper for a detailed explanation) Note: use this loss function only if the labels in the training dataset are very imbalanced e.g. one class having at least 3 orders of magnitude more voxels than the others. Otherwise use standard *DiceLoss*.

## Supported Evaluation Metrics

### Semantic Segmentation

* MeanIoU (mean intersection over union)
* DiceCoefficient (computes per channel Dice Coefficient and returns the average) If a 3D U-Net was trained to predict cell boundaries, one can use the following semantic instance segmentation metrics (the metrics below are computed by running connected components on thresholded boundary map and comparing the resulted instances to the ground truth instance segmentation):
* BoundaryAveragePrecision (Average Precision applied to the boundary probability maps: thresholds the output from the network, runs connected components to get the segmentation and computes AP between the resulting segmentation and the ground truth)
* AdaptedRandError (see <http://brainiac2.mit.edu/SNEMI3D/evaluation> for a detailed explanation)
* AveragePrecision (see <https://www.kaggle.com/stkbailey/step-by-step-explanation-of-scoring-metric>)

If not specified MeanIoU will be used by default.