Files and run time instruction:

1. There are totally 7 pythons files for the code:
   1) `CE_Training.py`: training code for the cross entropy loss model.
   2) `CE_Inference.py`: inference code for the cross entropy loss model.
   3) `DL_Training.py`: training code for the dice loss model.
   4) `DL_Inference.py`: inference code for the dice loss model.
   5) `DL_Bonus_Conv.py`: training and testing code for the dice loss model in which max pooling layers are replaced with convolution layers.
   6) `DL_Bonus_BatchNorm.py`: training and testing code for the dice loss model in which batch norm is used for every layer.
   7) `DL_Bonus_BatchNorm_Conv.py`: training and testing code for the dice loss model in which batch norm is used for every layer and max pooling layers are replaced with convolution layers.

2. There are 5 folders for the best weights:
   1) `CE_training`: best weights for the cross entropy loss model. Inference result is also included.
   2) `DL_training`: best weights for the dice loss model. Inference result is also included.
   3) `DL_Bonus_BatchNorm`: best weights for the dice loss model in which batch norm is used.
   4) `DL_Bonus_Conv`: best weights for the dice loss model in which max pooling layers are Inference result is also included. replaced with convolution layers. Inference result is also included.
   5) `DL_Bonus_BatchNorm_Conv`: best weights for the dice loss model in which batch norm is used for every layer and max pooling layers are replaced with convolution layers. Inference result is also included.

Part 1 – Modeling:

1. UNet:
   1) It is written as `class UNet(nn.Module)`.

   2) The dataset class has 2 modified functions:
      (1) `__init__(self)` defines the components of the network.
      (2) `forward(self, x)` defines how the image flow through the network.

   3) 4 classes are defined for the `UNet class`:
      (1) `class DoubleConv(nn.Module)` is defined for each convolutional block.
      (2) `class Down(nn.Module)` is defined for each downward step.
      (3) `class Up(nn.Module)` is defined for each upward step.
      (4) `class OutConv (nn.Module)` is defined for output layer:
         (a) The output layer has depth of 2 for cross entropy loss model.
         (b) The output layer has depth of 1 for dice loss model.

2. Best weights:
   Best weights during each training is defined as the weights providing highest dice metric for validation dataset.

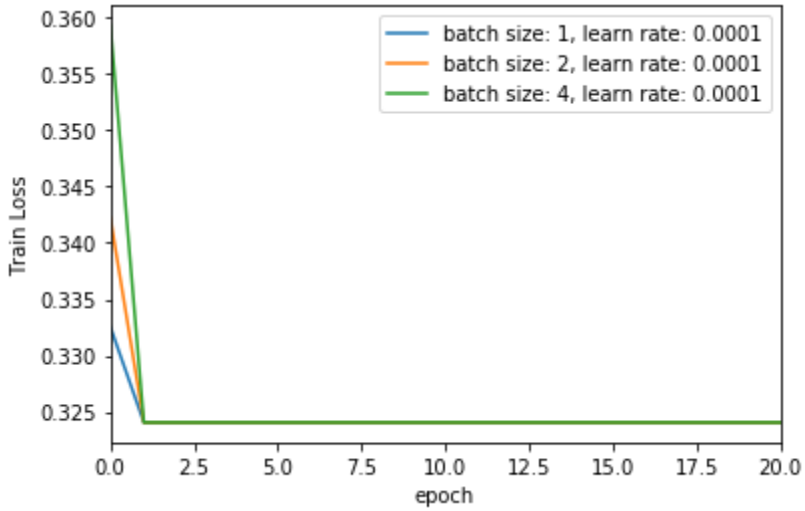
Part 2 – Hyperparameter Tuning:

1. Tuning for cross entropy loss UNet model:

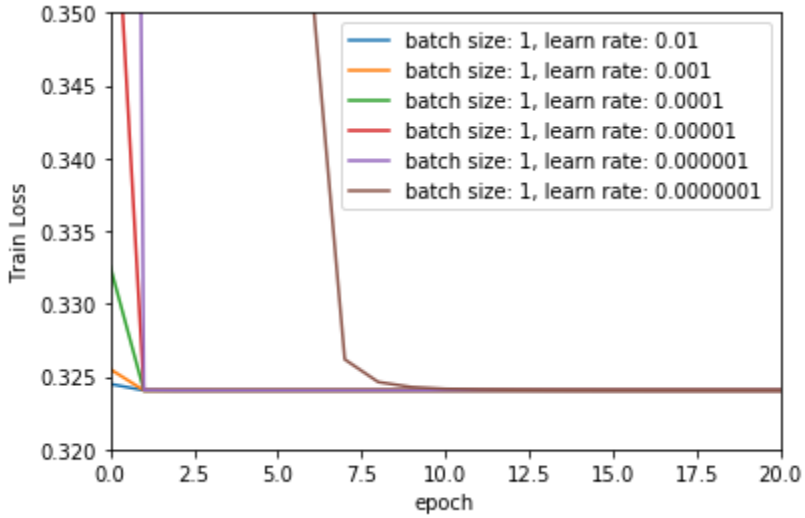
1) Optimizer:
   Adam is selected as the optimizer.

2) Each training includes 50 epochs.

3) Batch size:
   Using learning rate of 0.0001, we have tried batch size of 1, 2, 4 because we ran out of GPU memory (8G) once the batch size is higher than 8. It turned out that batch size of 1 resulted in the fastest learning. Thus, we decided to use batch size of 1.



4) Learning rate:
   Using batch size of 1, we compared the learning rate of 0.01, 0.001, 0.0001, 0.00001, 0.000001, and 0.0000001. It turned out that learning rate of 0.01 provided the best learning curve. Thus, we decided to use learning rate of 0.01.



5) Using the best weights from the training using the tuned hyperparameters (batch size = 1, and learning rate = 0.01), we have the inference results:

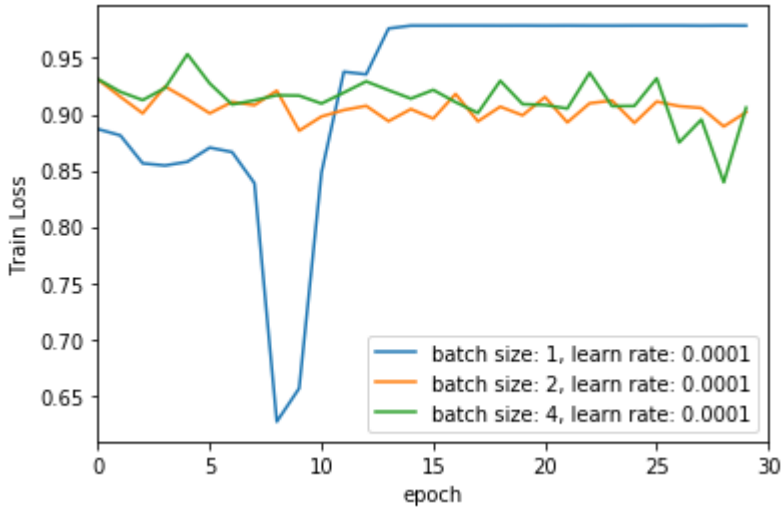
	Cross Entropy Loss	Dice Loss	Dice Metric
Training Dataset	0.3240	1	0
Validation Dataset	0.3237	1	0

Testing Dataset	0.3237	1	0

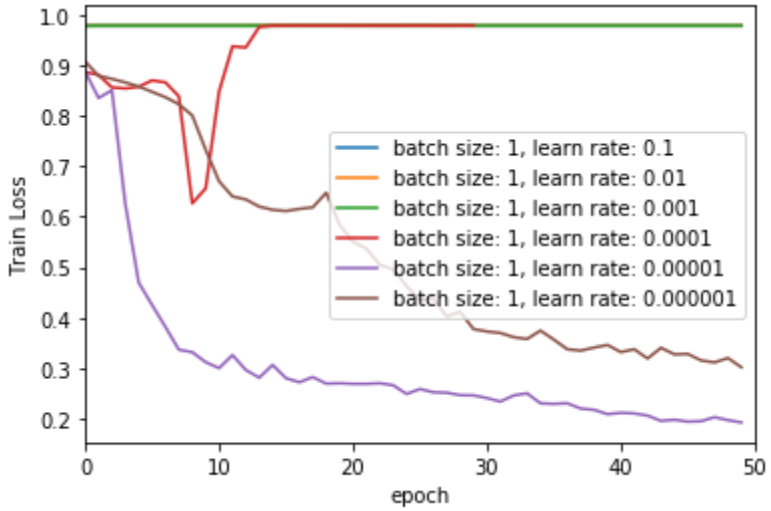
2. Tuning for dice loss UNet model:
   1) Optimizer:
   Adam is selected as the optimizer.

   2) Each training includes 30-50 epochs.

   3) Batch size:
   Using learning rate of 0.0001, we have tried batch size of 1, 2, 4 because we ran out of GPU memory (8G) once the batch size is higher than 8. It turned out that batch size of 1 resulted in the best loss for training dataset. Thus, we decided to use batch size of 1.



   4) Learning rate:
   Using batch size of 1, we compared the learning rate of 0.1, 0.01, 0.001, 0.0001, 0.00001, and 0.000001. It turned out that learning rate of 0.00001 provided the best learning curve. Thus, we decided to use learning rate of 0.00001.



5) Using the best weights from the training using the tuned hyperparameters (batch size = 1, and learning rate = 0.00001), we have the inference results:

	Dice Loss	Dice Metric
Training Dataset	0.8570	0.1429
Validation Dataset	0.7945	0.2055
Testing Dataset	0.7879	0.2121

3. Improvement of dice loss UNet model:

1) We first tried to use batch norm for every layer to see if the dice loss UNet model can be improved. Unfortunately, the inference result is worse.

	Dice Loss	Dice Metric
Training Dataset	0.863	0.1377
Validation Dataset	0.7668	0.2332
Testing Dataset	0.7673	0.2327

2) We then tried to replace max pooling with convolution (kernel 2x2, stride 2) to see if the dice loss UNet model can be improved. The inference result is quite similar to original UNet.

	Dice Loss	Dice Metric
Training Dataset	0.8049	0.1951
Validation Dataset	0.7965	0.2035
Testing Dataset	0.7851	0.2149

3) We also tried to use batch norm for every layer and replace max pooling with convolution (kernel 2x2, stride 2) to see if the dice loss UNet model can be improved. Unfortunately, the inference result is worse.

	Dice Loss	Dice Metric
Training Dataset	0.8413	0.1587
Validation Dataset	0.7362	0.2638
Testing Dataset	0.7215	0.2785