2 Semantic segmentation of Biomedical images

2.1 Semantic segmentation

2.1 a) Defining the Dice Score function

```
def DSC(a,b):
  """
  Returns the Dice SCore between two tensors
  param a: one of the tensors
  param b: the second tensor
  """
  return 2*torch.sum(a==b) / (a.numel()+b.numel())
```

2.1 b) Building the model :
After trying to convert the previous network used in task 1 and failing to train it, presumably because of vanishing gradients, a more shallow network was found to work :

```
class shallower_CNN_segment(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(16, 50, 3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        self.tconv = nn.ConvTranspose2d(50, 1, 4, stride=2, padding=1)
        self.sig = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.conv2(x)
        x = self.pool(x)

        x = self.tconv(x)
        x = self.sig(x)
        x = x.squeeze(dim=1) # Matching shape of the labels

        return x
```

Training the model :

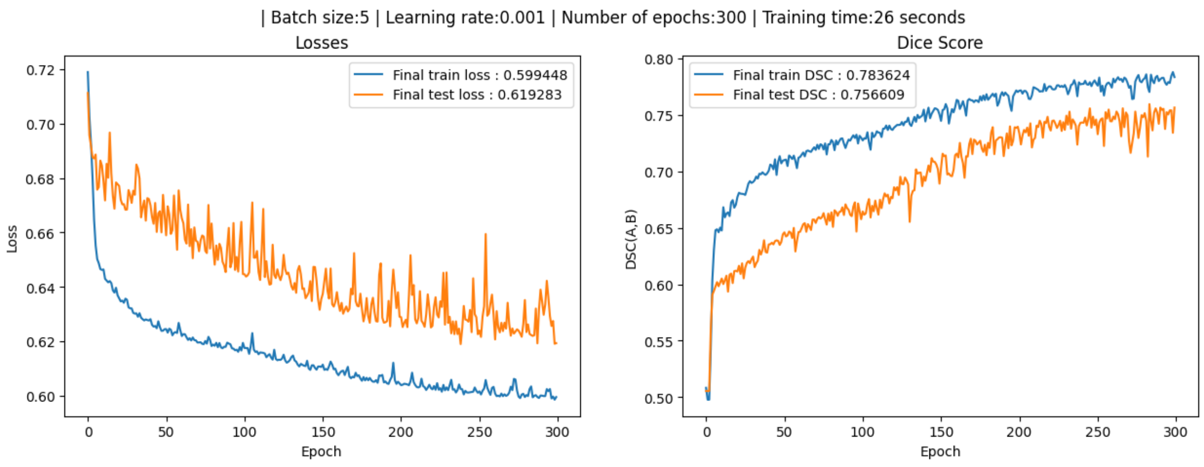

Figure 12: Learning plots for the image segmentation network
This network was trained using ADAM, without a learning rate scheduler.

2.1 c) Showing outputs :

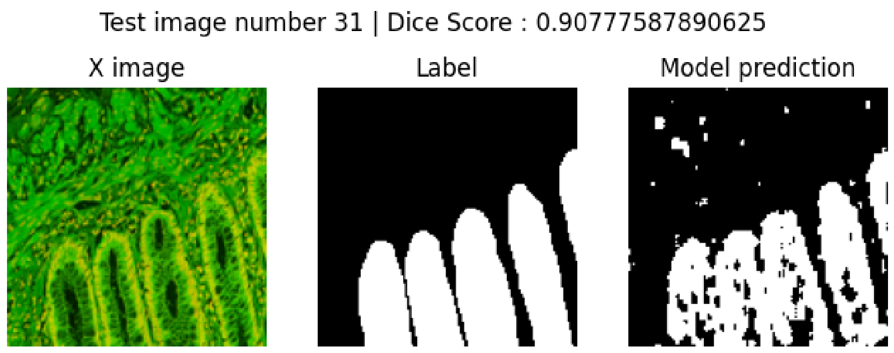

Figure 13: Best DSC on test data

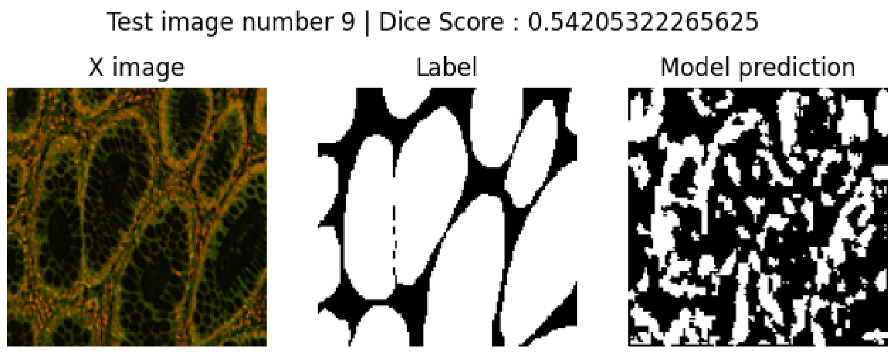

Figure 14: Worst DSC on test data

When comparing the two figures above, we can see that the borders of the tumors are more clearly marked on image 31 where they are glowing green than on image 9 where they are of the same color than the background.
As a result, the network appears to be detecting borders of tumors all over the place for this second image.

2.2 Segmentation, three variations

2.2 a) Adding noise to input during training

Adding noise is helpful both for Data Augmentation, since we do not have that large of a training set, but especially as a regularization technique to make the network less dependant on the specificities of the training data.

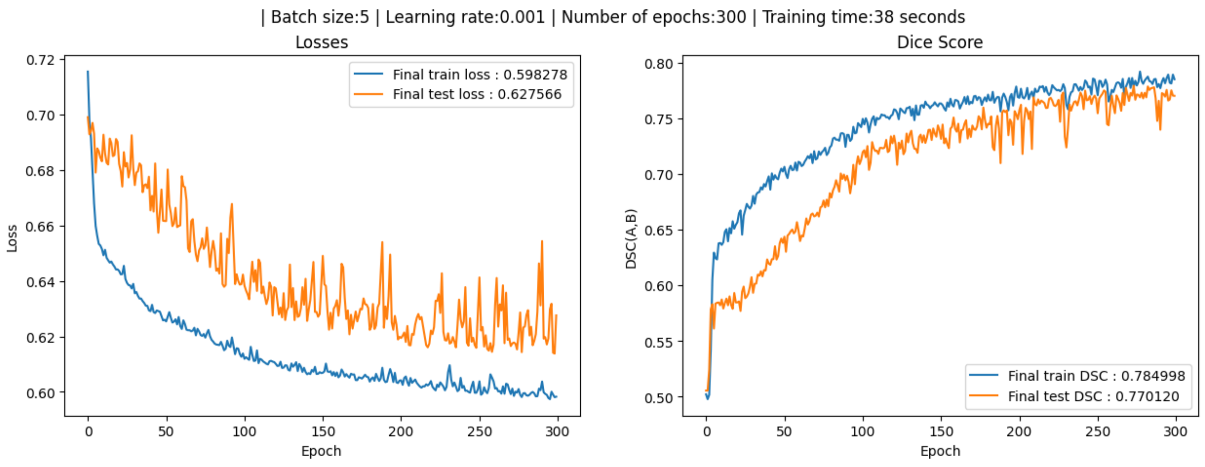

Figure 15: Learning plots for the segmentation network trained with noise augmented data

We used Gaussian noise with 0 mean and 0.03 standard deviation.

We can see in plot 15 that we achieved slighlty better performances on testing and shrunk the gap to training performances. The surprising thing is that seem not have degraded our training performances while we perturbed the input.
This suggests that the data augmentation was useful in general, maybe in helping the network get out of a local minimum by making it focus on more important general features rather than ones specific to the training data.

Here the network is definitely not overfitting, as training and testing performances are really close (at least in terms of Dice Score)

2.2 b) Dropout

Dropout randomly deactivates certain nodes of the network during training. Although more demanding in training time, this technique limits the ability of the network to learn too specific representations and thus overfitting to the training data.

The following learning plot was realized after adding a 0.5 probability of deactivating nodes of the second convolution layer. See code :

```python
class dropout_CNN_segment(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(0.5) # Dropout layer
        self.conv2 = nn.Conv2d(16, 50, 3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        self.tconv = nn.ConvTranspose2d(50, 1, 4, stride=2, padding=1)
        self.sig = nn.Sigmoid()

    def forward(self, x):

        x = self.conv1(x)
        x = self.relu1(x)
        x = self.dropout1(x) # Dropout layer
        x = self.conv2(x)
        x = self.pool(x)

        x = self.tconv(x)
        x = self.sig(x)
        x = x.squeeze(dim=1)

        return x
```

This single 0.5 dropout layer was adopted after trial and error, observing that two dropout layers with probability 0.2 was not working as well for example.

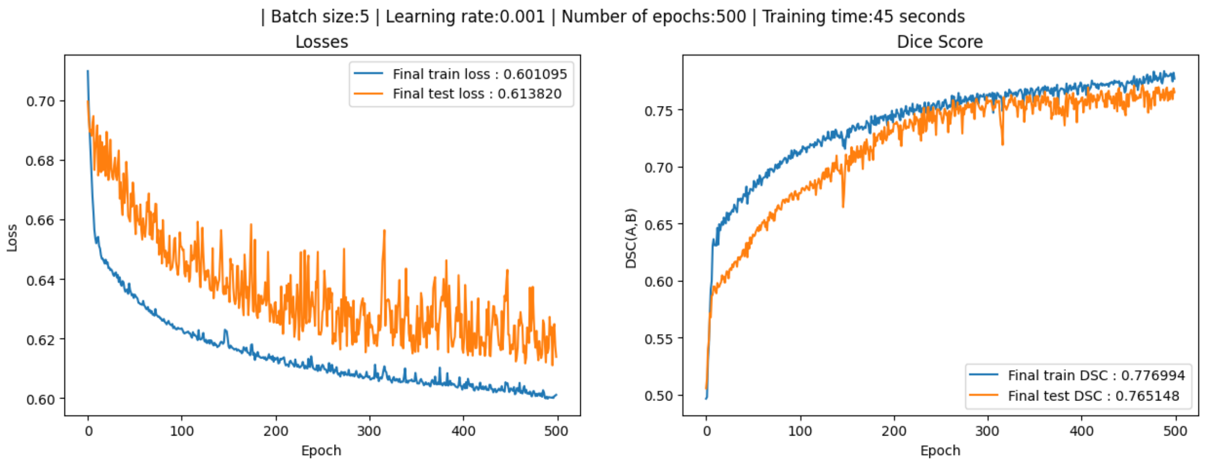

Figure 16: Learning plots for the network training with a Dropout layer

We gave more training epochs to the model to compensate for the difficulty induced by dropout.

Here, in figure 16, we can see that, similar to figure 15, the training and testing performances are much closer than in figure 12.
The conclusion is similar to that of section 2.2 a)

2.2 c) L2 Regularization

L2 (or L1) Regularization is a method that adds a term in the loss function, penalizing high (in absolute value) weight values. This coerces the network into a simpler representation of the problem and thus, is supposed to prevent overfitting.

After a bit of trial and error, the best value found for the coefficient penalizing the square of the weights is 5×10^{-5}.

Here are the results :

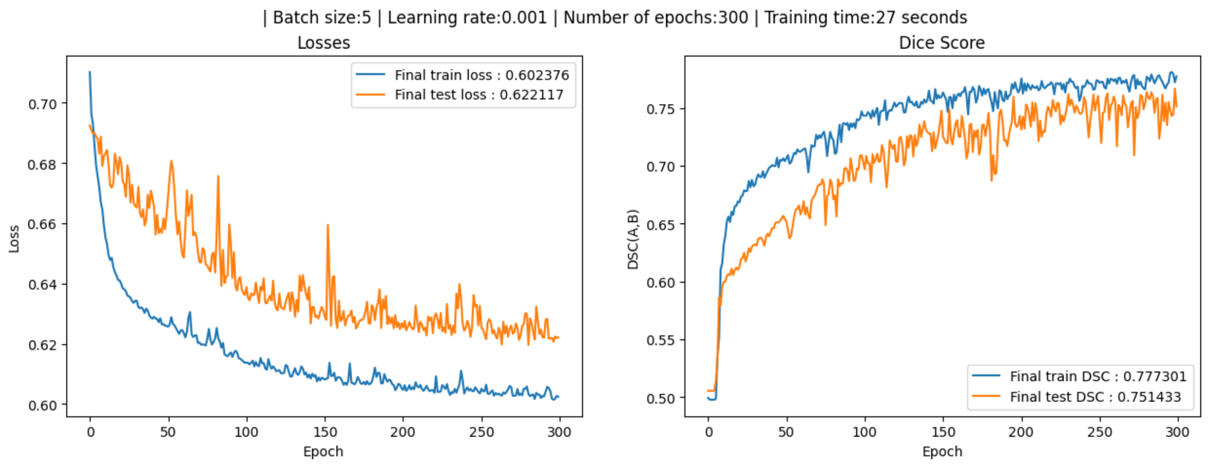

Figure 17: Learning plots for the network trained with L2 regularization

As we can see here, the results are similar to (a) and (b) in that we observe a smaller gap between training and testing performances.
However, the training score is lower than before, so we essentially closed the gap only by hurting the training performances and not by improving the ones on testing.