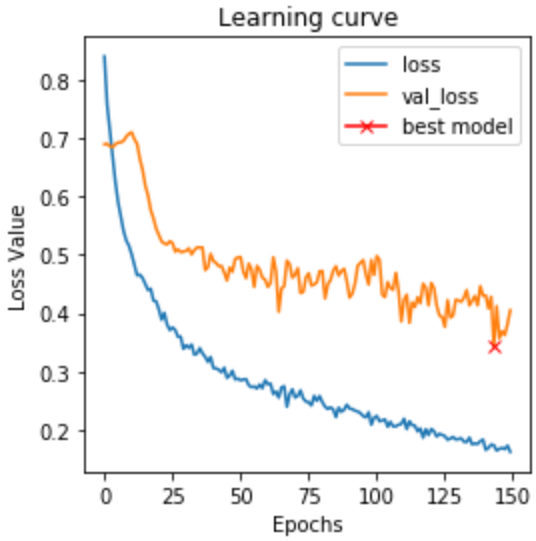
Students: Blanca Cabrera Gil & Laila Niazy
Course: CM2003 Deep Learning Methods for Medical Image Analysis

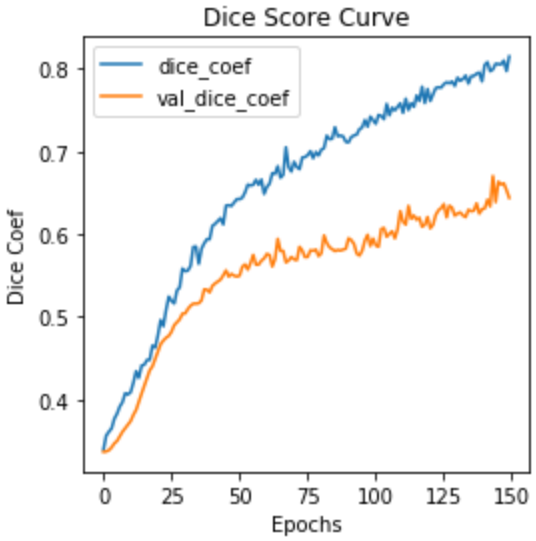
LAB 3

Lung segmentation in chest X-ray images

Task 1a)

Lung segmentation in chest X-ray images: Instantiate your pipeline with the following parameters: # of filters at first layer (base)=16; image size=256; batch size=8; learning rate = 0.0001; drop out rate=0.5; batch normalization = True; Metric = Dice Coefficient; no data augmentation; and loss function = binary cross entropy. Reading the 'X_ray' images (images and masks), randomly shuffle them and assign 80 percent of them for training and the rest of 20 percent for validation. Train the model for 150 epochs and save your final results.

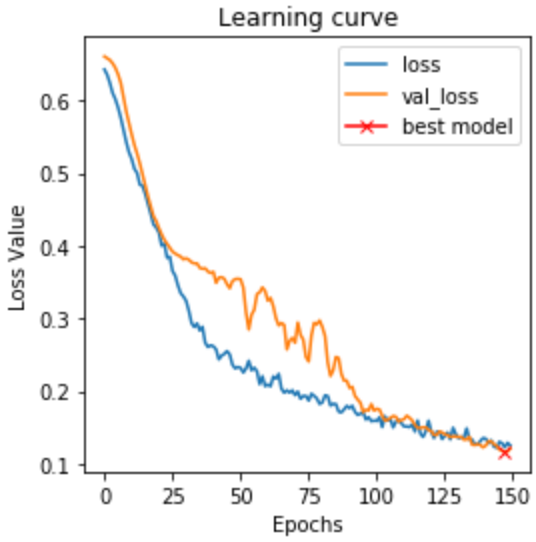


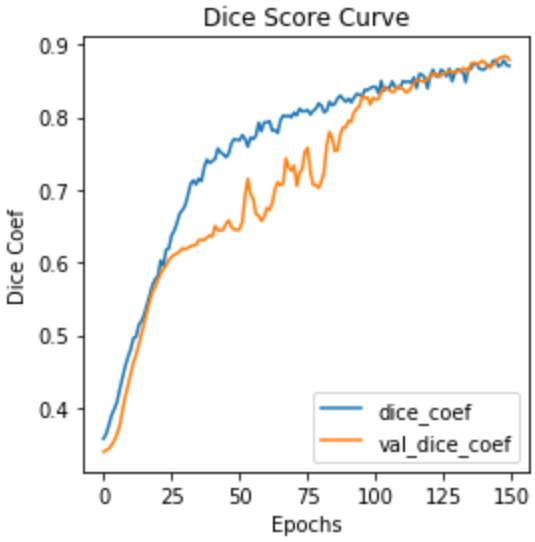


Task 1b)

Keep all the parameters from previous exercise the same, but only change the loss function to "Dice loss" and repeat the exercise. Is there any difference between model performance over validation set when you changed the loss functions? Discuss it.

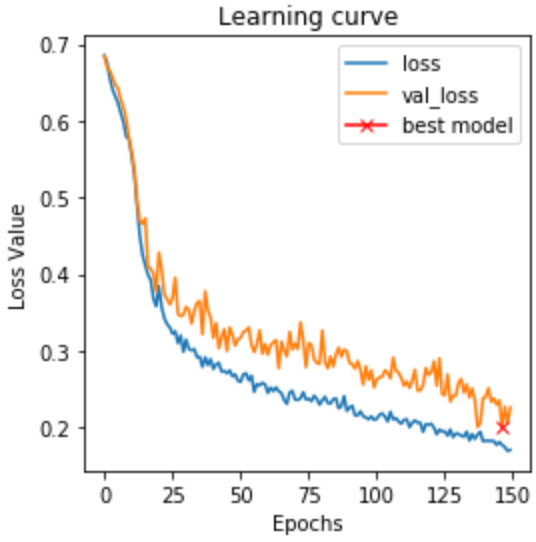
As one can see in the figures below, the loss decreases and dice score increases when using the dice loss compared to the figures above, which we created using the binary crossentrophy loss. With the dice loss, we achieve a higher generalization ability as can be seen in the validation loss curve. The main reason for the better results, could be that dice coefficient performs better at class imbalanced problems than other loss functions, which is the case here.

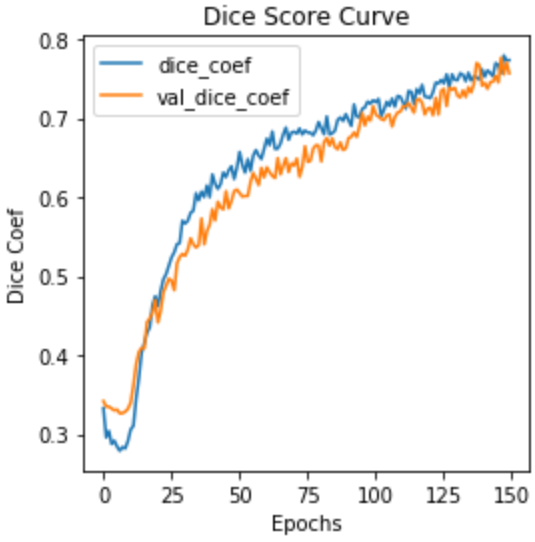


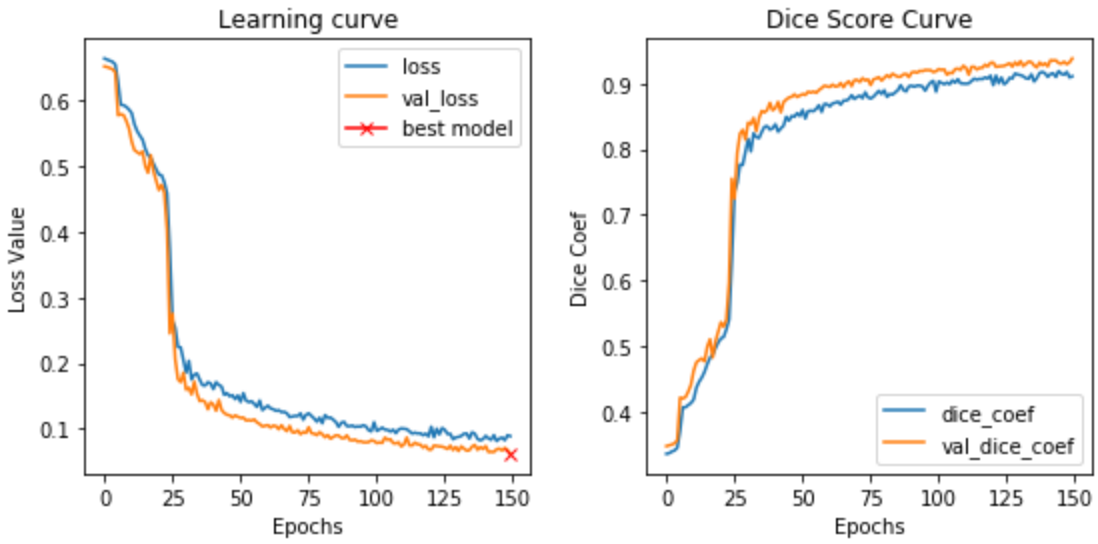


Task 2)

Repeat the tasks 1a and 1b without applying the batch normalization technique. Does it affect the model performance? What about the learning curves?





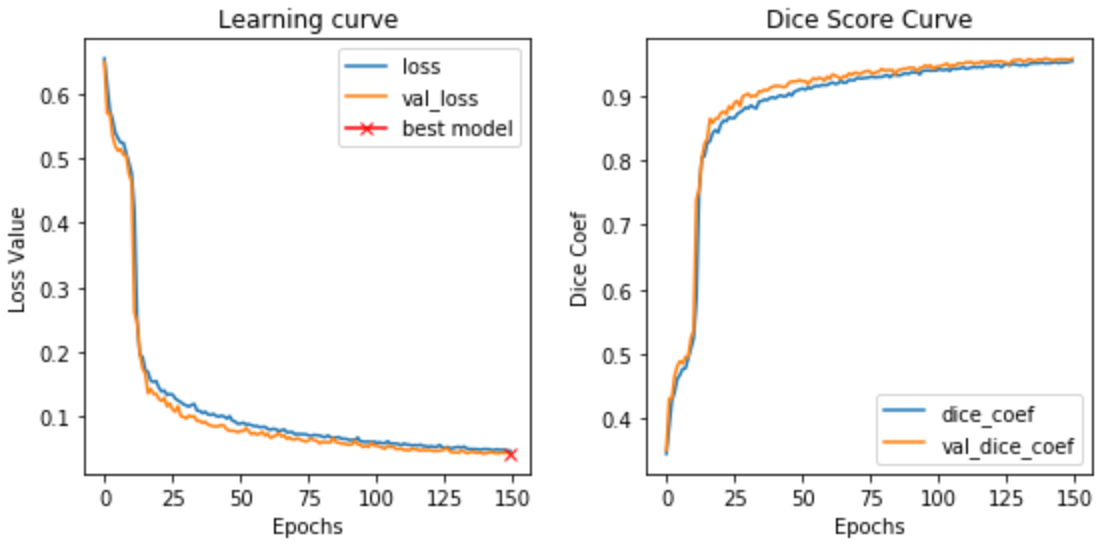


The first two figures were created using binary crosstrophy loss, while the lower figures were created using dice loss. As one can see here again the model trained using the dice loss performs better due to the imbalance in class distribution of the training data. Further comparing the four plottes with the plottes of task 1, we can see that without batch normalization we have better results. The learning curve and the dice score curve are smoother in task 2 and don't pendle. The model converges fast and generalizes better in task 2 compared to Task 1. However, batch normalization does indeed speed up the learning process but in this case at the expense of model performance. In this case, using Dropout was enough in reducing overfitting and adding more regularization and batch normalization was not needed to be added to increase model performance.

Task 3)

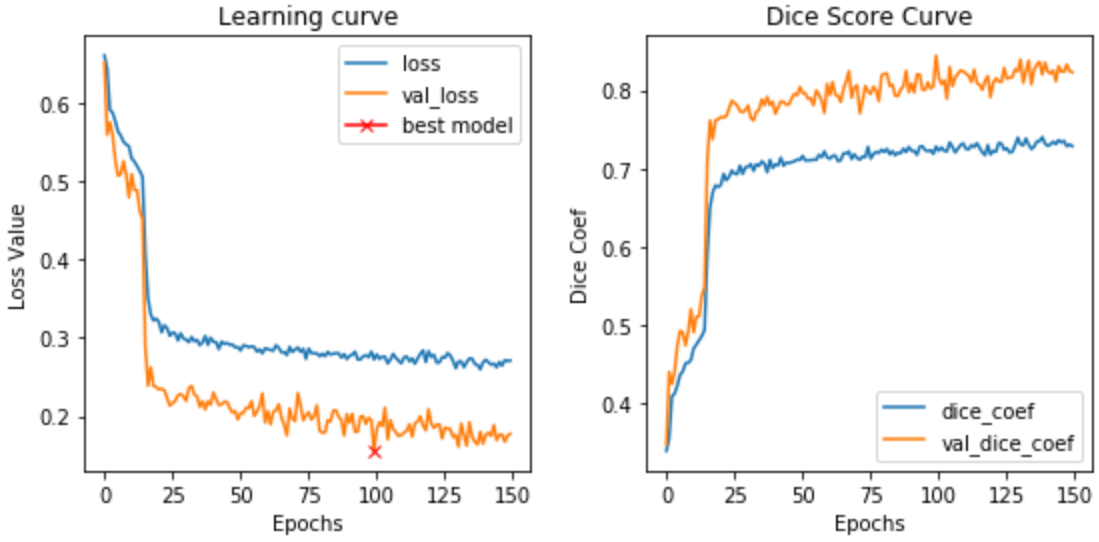
In tasks 1,2) you already worked with 4 different settings. Choose the setting with the best performance and set the # of filters at the first layer (base) as 32 and train the model for 150 epochs. Does it change the segmentation accuracy? In general, how do you choose the number of feature maps?

We choose the setting without batch normalization and using dice lose function to train the model. Looking at the figures below, we can see a slight improvement in the learning curve and dice score compared to task 2 (which had the best performance). The curves have smoothed out even more. On both training and validation set, the model performed well, showing that it generalizes well on new data. When choosing feature maps, one has to be careful and choose not to many features, so the model overfits and not to few features that the model underfits. We have to experiment with different values like we did here to determine the right number of features needed for this specific problem.



Task 4)

Similar to task3, employ the setting with the best performance and apply the augmentation technique on both images and masks: rotation range=10; width and height shift ranges=0.1; zoom range = 0.2 and horizontal flip. Could data augmentation improve the model performance? What about generalization power?



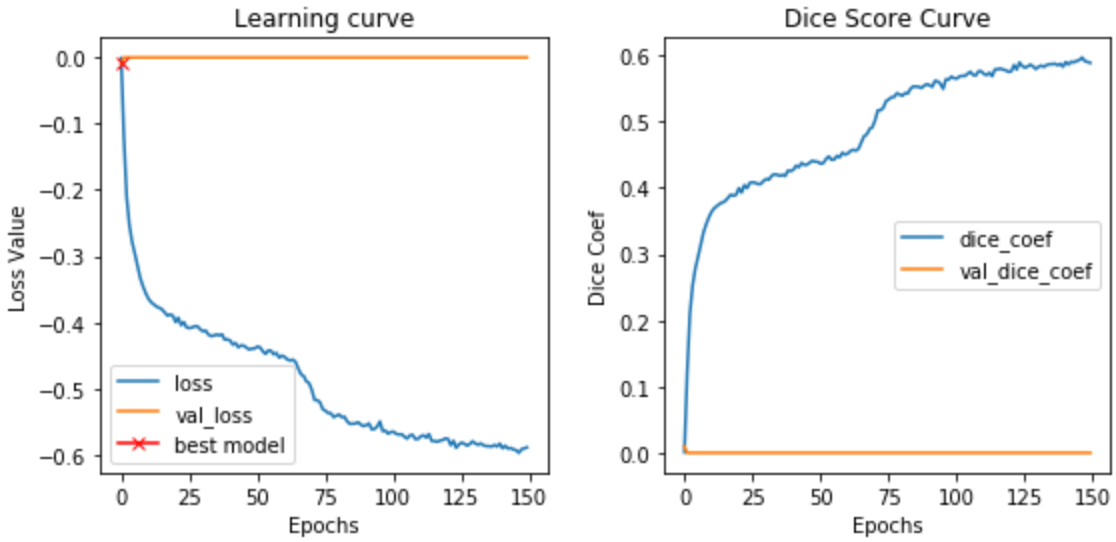
In the figures above, we can see that the model performed better on the test data than on the training data, which choose that the model has a high generalization power. However, using

data augmentation both training and test data had a lower dice score and a higher loss function compared to task 3, which has the same settings as task 4 without data augmentation. For this dataset data augmentation is not necessary.

Task 5a)

Lung segmentation in CT images: Reuse the exact settings of the task1 for "CT" segmentation. Which of the X-ray or CT images yielded more accurate segmentation results? Why?

Epoch 150/150
6735/6735 [==============================] - 50s 7ms/sample - loss: -0.5882 - dice_coef: 0.5882 - val_loss: -1.7059e-04 - val_dice_coef: 1.7099e-04

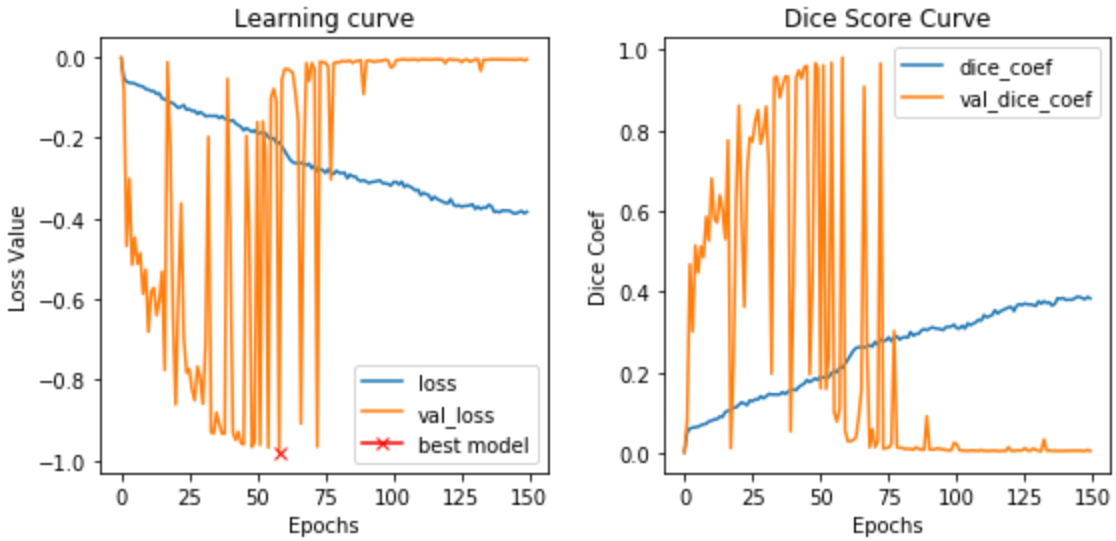


When training the model to segment the lungs out of the CT images it can be seen that the model is learning from the train set but it has low generalization capability. Moreover, the model is learning slowly and after 150 epochs has not yet overfitted.

Task 5b)

Evaluating the segmentation performance is often done by Dice coefficient; however, it is quite essential to assess the level of false positives and false negatives too. Repeat the task 5a by including data augmentation technique (similar to task4). Moreover, implement "precision" and "recall" metrics and along with "dice coefficient" apply all of them on your model. (Metric = [dice_coef, precision, recall]). Interpret the observed values of these three metrics over the validation set.

Epoch 150/150
842/842 [==============================] - 53s 63ms/step - loss: -0.3834 - dice_coef: 0.3834 - recall: 0.6068 - precision: 0.2698 - val_loss: -0.0052 - val_dice_coef: 0.0052 - val_recall: 0.0000e+00 - val_precision: 0.0000e+00

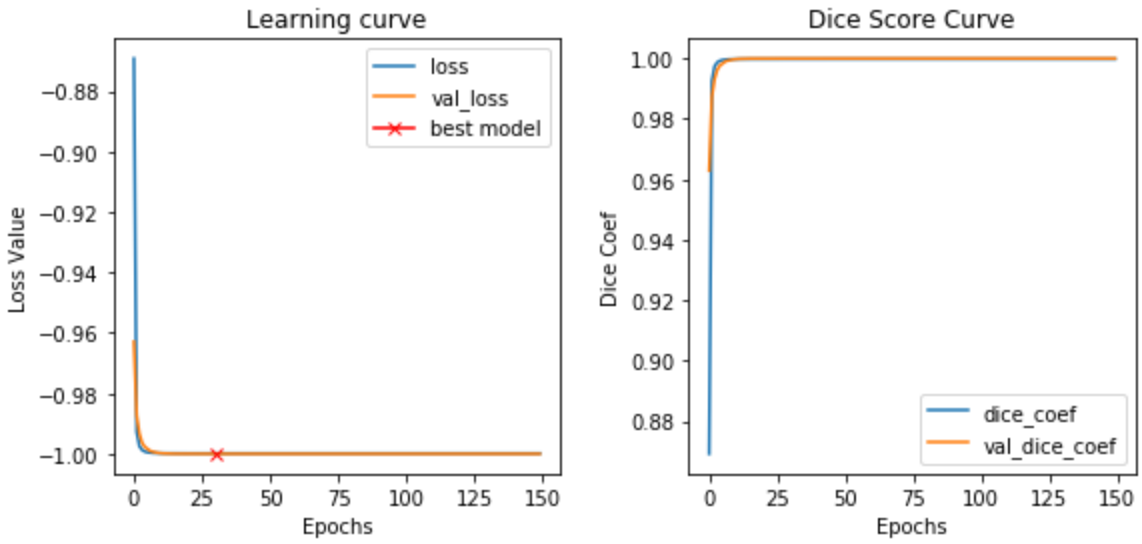


When using data augmentation it can be seen that the model learns from the train set and the generalization power is increased. However, it reaches a point where the model overfits the train data.

Task 6)

As you might have already noticed, the labels of the left and right lungs in the CT masks are not the same values. We can consider the two lungs as two different organs and perform a multi-organ segmentation. Modify your model and adapt it for a multi-organ segmentation task and segment the left and right lungs separately in one framework.

Epoch 150/150
842/842 [==============================] - 56s 66ms/step - loss: -0.9998 - dice_coef: 0.9998 - recall_1: 0.9998 - precision_1: 0.9998 - val_loss: -1.0000 - val_dice_coef: 1.0000 - val_recall_1: 1.0000 - val_precision_1: 1.0000



When performing multiclass segmentation, the model learns the segmentation task with a very good generalization power, reaching a recall rate of 0.9998 and a precision of 0.9998.

Task 7)

Brain tumor segmentation in MRI images: In contrast to the lungs that entails specific shapes, appearances, and intensity patterns, brain tumors can be presented in a variety of shape, appearance, texture and intensity patterns. More importantly, they can be presented in any regions inside the brain therefore they do not have a fixed location. In this exercise you are asked to optimize your model by tuning the hyperparameters for the challenging task of brain tumor segmentation. Please note the original size of the brain MRI are 240 by 240.

The code for this task has been implemented, however due to the large amount of parameters or the environment limitations, the kernel dies before finishing the training at epoch 49/150.

After reducing the image size to 96 and reducing the number of epochs to 80 the model run until epoch 66 before the server went down again obtaining the following results.

Epoch 66/80
1881/1881 [==============================] - 48s 26ms/step - loss: 0.0000e+00 - dice_coef: 1.0000 - recall_1: 0.0000e+00 - precision_1: 0.0000e+00 - val_loss: 0.0000e+00 - val_dice_coef: 1.0000 - val_recall_1: 0.0000e+00 - val_precision_1: 0.0000e+00