Gnana Spandana Akumalla Serkan Arda Yilal Patrik Svensson

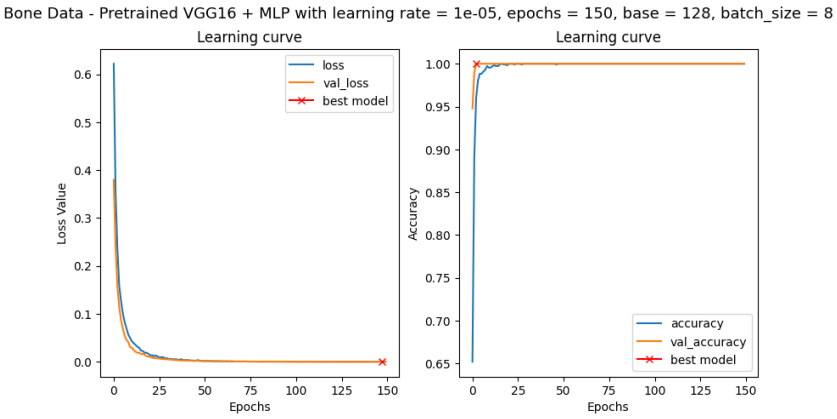
CM2003-Lab 4

Task 1) The code for this task can be found in *Lab4 - Part 1 & 2.ipynb* file and the functions used such as **MLP_model**, **loss_accuracy_plot**, **get_length**, **pretrain_data_label** can be found under *Models.py* and *Utils.py* files.

Task 2) In this task, we trained bone and skin datasets by using extracted features from pretrained VGG16 convolutional layers. The extracted features are trained on the specified MLP model. Additionally, new VGG16 models were trained on bone and skin dataset from scratch to compare their performance with pretrained VGG16.

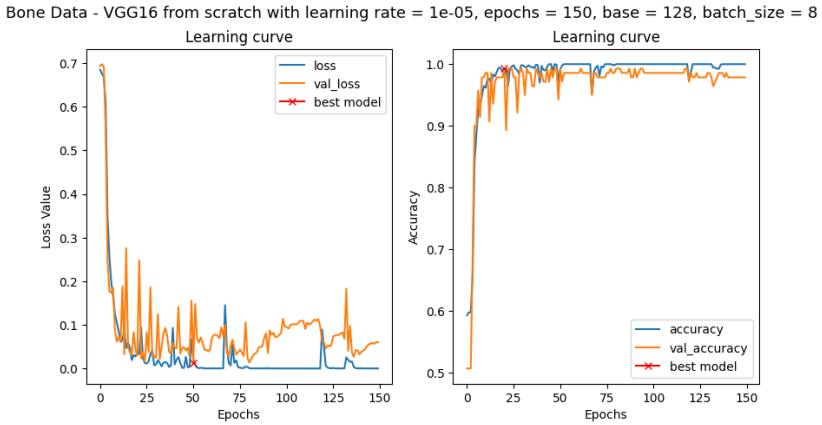
- Bone dataset classification with pretrained VGG16:

Training duration = 63 seconds



- Bone dataset classification with VGG16 trained from scratch:

Training duration = 27.5 minutes

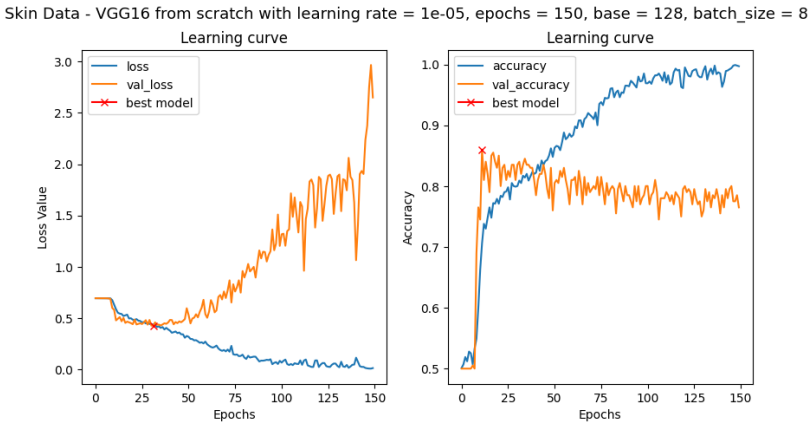


- Skin dataset classification with pretrained VGG16:

Training duration = 59 seconds

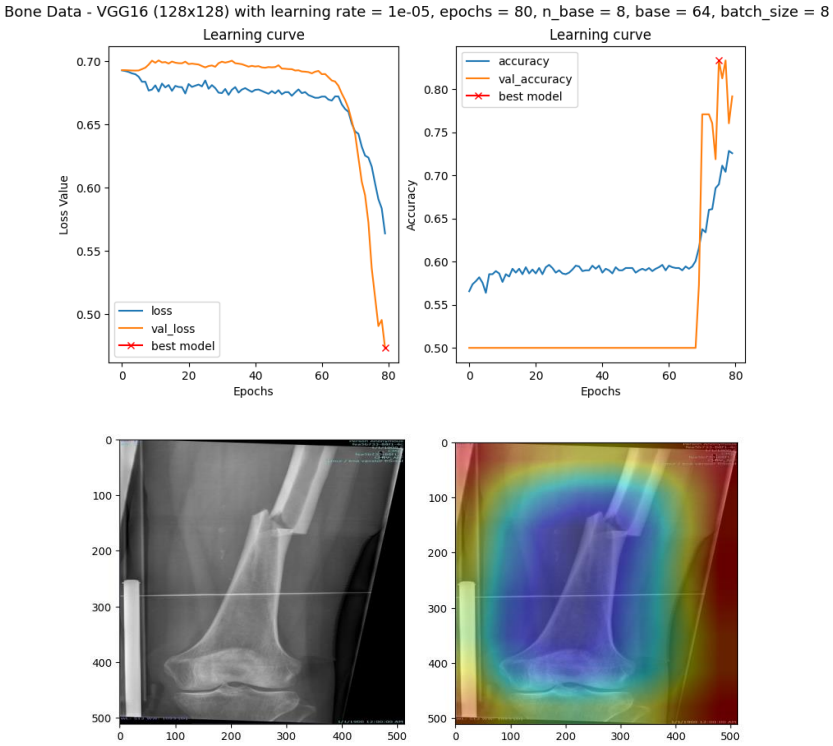
- Skin dataset classification with VGG16 trained from scratch:

Training duration = 26.1 minutes

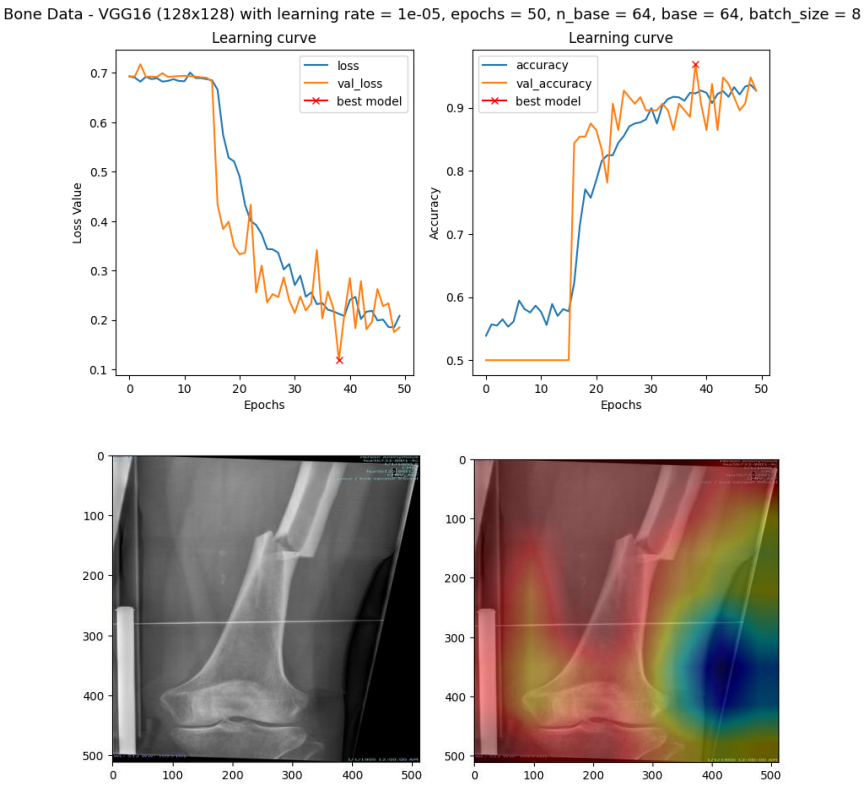


When we compare the models with pretrained VGG16 and models trained from scratch, we see that pretrained models achieves low validation loss much faster and maintain a more stable validation accuracy with slightly higher values. Also, it seems like models trained from scratch seem to be overtrained during training while the pretrained models are stronger against overtraining. We think another difference that should be considered is the training time that is spent for training both models. For both datasets, while pretrained models took 1 minute to train, the models trained from scratch took 26-27 mins. This shows to for generalizability and time efficiency, transfer learning is very useful. To check its reliability, more datasets can be checked and different settings can be used in order to see if it always performs better than models trained from scratch.

Task 3) In this task, we trained VGG16 models with specified settings and checked activation maps on for the last convolution layer. The code for this task can be found in *Lab4 - Part 1 & 2.ipynb* file. After training the model, the loss and accuracy plot along with the activation map were as follows:



We see that the model had some trouble on learning from the data. The activation map showed that the model did not concentrate on the features of the bone considering the edges of the image are red. This might be the reason why the model couldn't learn it well. We though that it might be due to low base values in convolutional layers so we decided to train another model with base = 64.



We see that in this model, we achieved higher validation accuracy and lower validation loss. The activation map is also showing that the model concentrated on the features around the bone area which would be useful for our classification task. This experiment might indicate that number of neurons is important in extracting useful features.

Task 4) The functions implemented for this task can be find in the following files:

Functions for image and mask loading in *Utils.py* = load_data, load_image, load_mask

Functions for data augmentation in *Utils.py* = create_generator, image_mask_generator

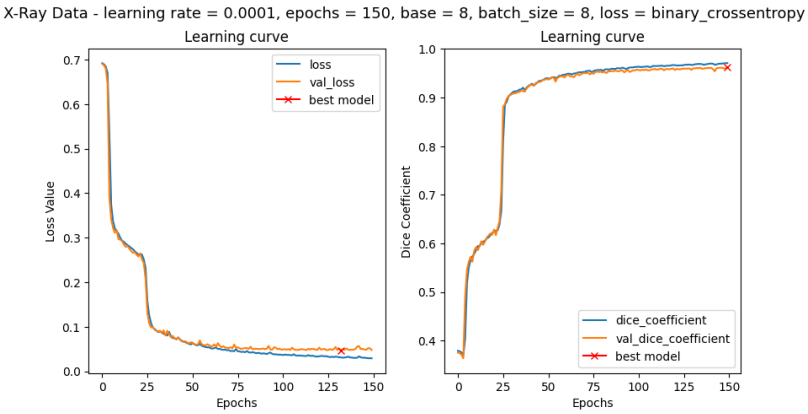
Functions for metrics in *Utils.py* = dice_coef, dice_coef_loss

Functions for model architecture in *Models.py* = get_unet, conv_block

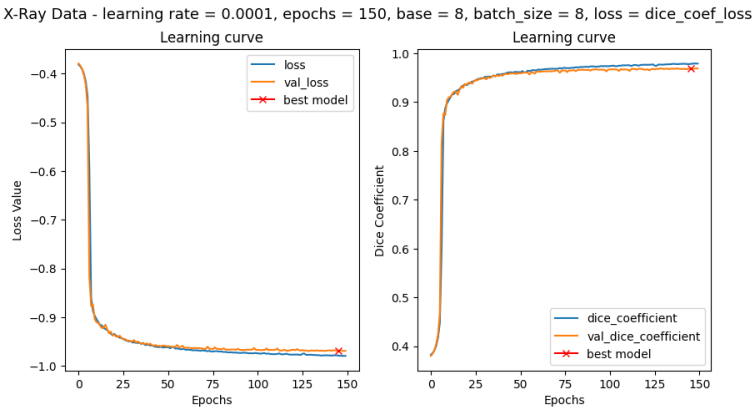
Function for training u-net model in *Utils.py* = train_unet

The way they are used can be checked in file *Lab4 - Part 3.ipynb*.

Task 4 A) In this task, we read X Ray data and used u-net architecture to train and validate the performance of segmentation. The loss function used in this task was binary cross entropy. The loss and dice coefficient plots of the model are as follows:



Task 4 B) In this task, instead of binary cross entropy loss, we use dice loss. The dice loss and dice coefficient plots of the model are as follows:

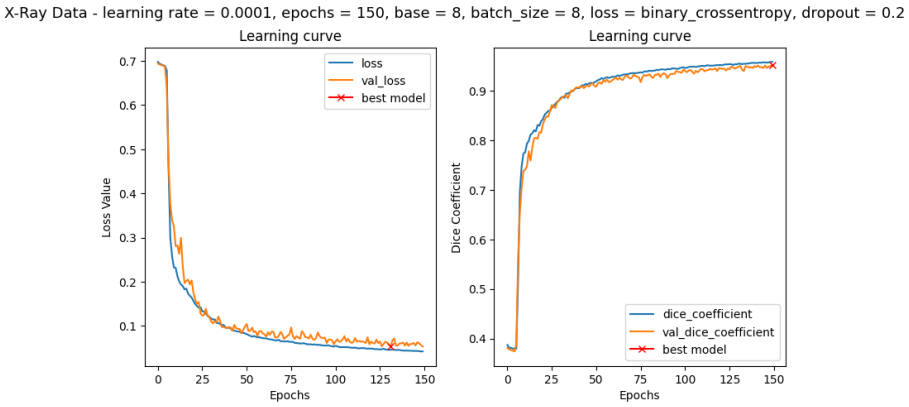


We see that the dice loss achieved a high validation accuracy and low validation loss much faster. It might be an expected result as with dice loss, we are directly aiming to have a higher dice coefficient.

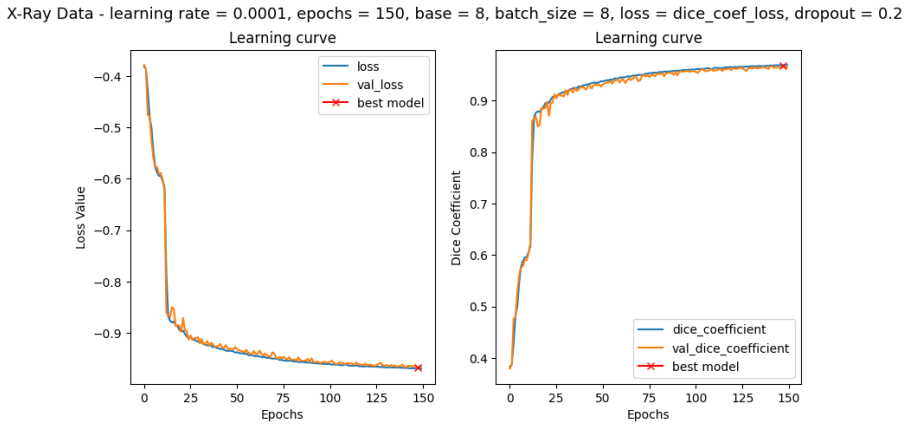
Considering it aims to have the highest overlap between the images and their corresponding masks, that might be the reason of performing better. In case of imbalance between background and foreground, it would make more sense to use dice loss as it penalizes if there is not intersection. Hence, we would go with dice loss.

Task 4 C) In this task, we repeat task 4 A and 4 B with a dropout rate of 0.2. The loss and dice coefficient plots are as follows:

- U-net with BCE and dropout = 0.2:



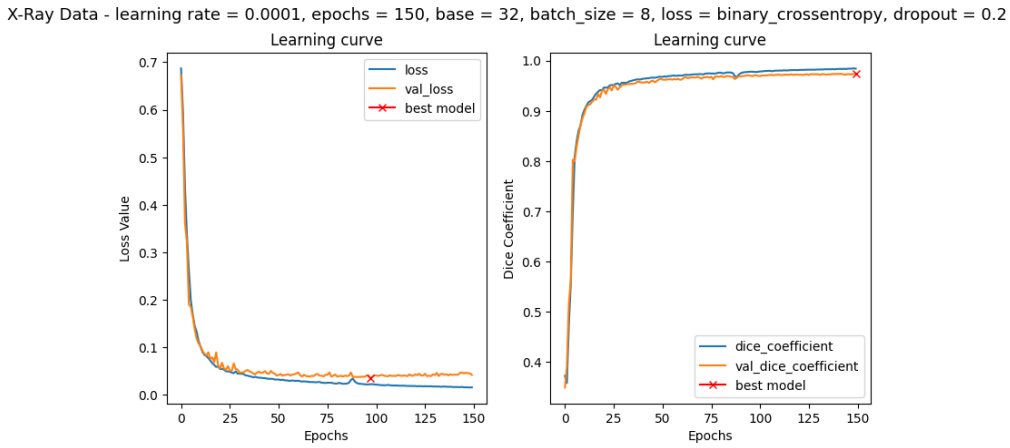
- U-net with dice loss and dropout = 0.2:



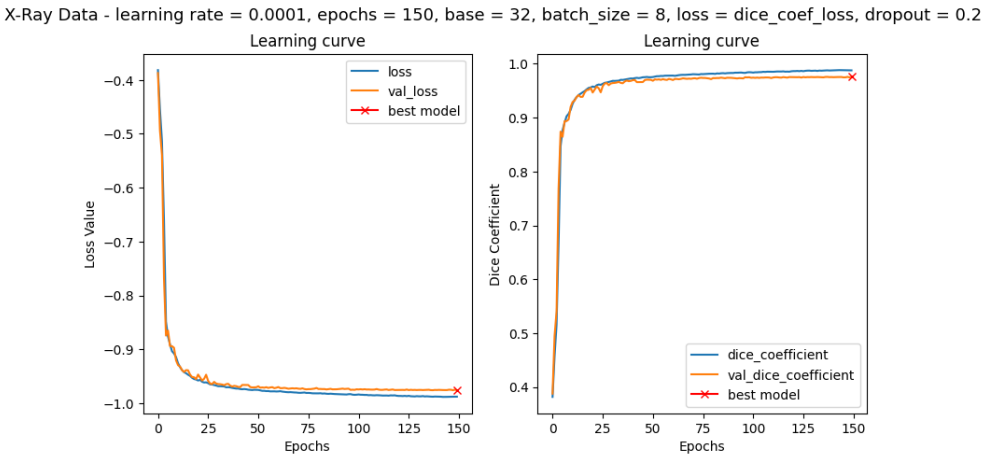
When we check the results compared to previous tasks, it seems like dropout layers helped model trained with binary cross entropy loss to have a more linear increase in validation accuracy although it reaches a validation accuracy of 0.9 slightly later. On the other hand, dropout layers seem to slowed the increase in validation accuracy down for model trained with dice loss as it reaches a value of 0.9 slightly later. It seems to slightly change the learning curve as well. Also, the validation loss does not go as low as it did without dropout for model trained with dice loss.

Task 4 D) In this task, we repeat the previous task with a base value of 32.

- U-net with BCE and dropout = 0.2, base = 32:



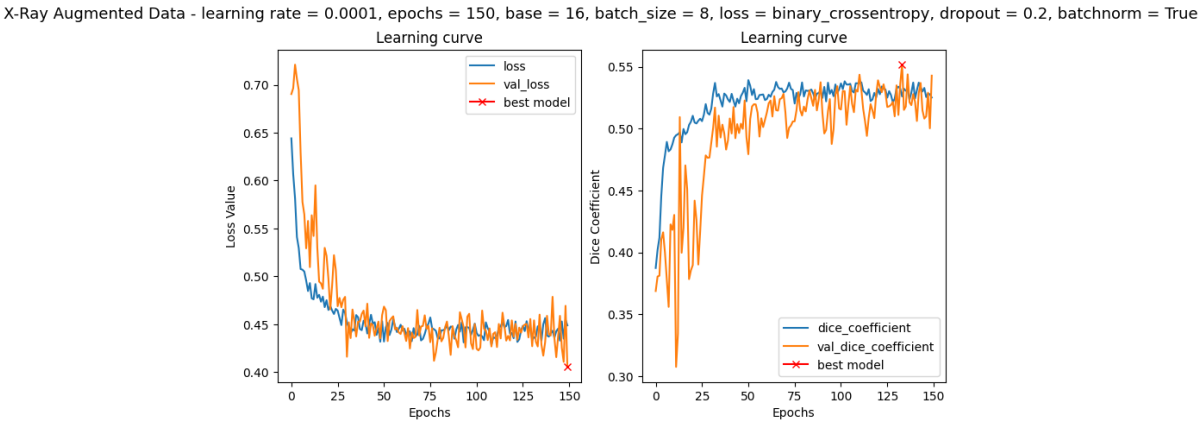
- U-net with dice loss and dropout = 0.2, base = 32:



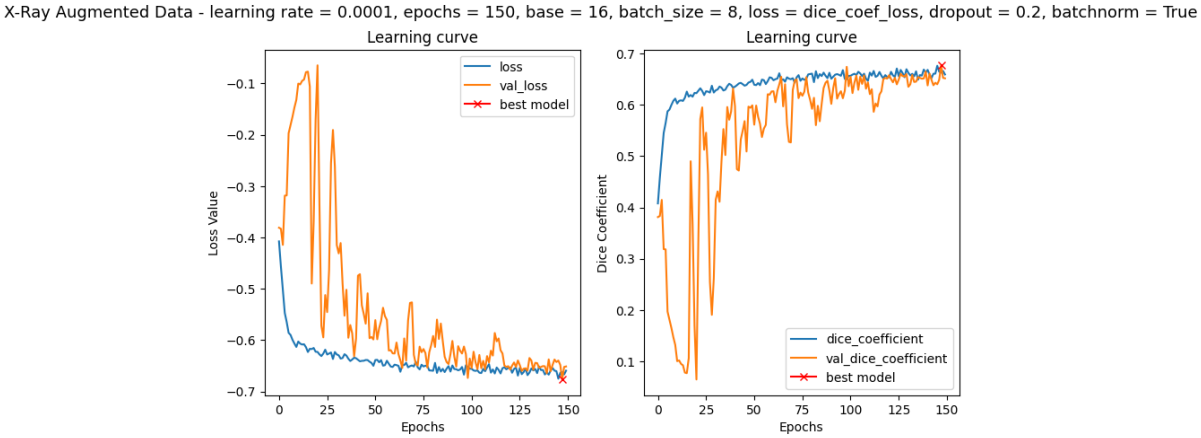
When we check the results this time with higher base value, it definitely improved both models to train and achieve higher validation accuracy and lower validation loss much faster. But it should be also noted that there seems to be a gap between training and validation loss especially for the model trained with binary cross entropy loss. This might indicate a possible overtraining for further increase in base.

Task 4 E) In this last task, we repeat the task 4c with a base value of 16 and with an activated batch normalization. Also, we applied the data augmentation on the dataset and analysed the results with these settings. The loss and dice coefficient plots are as follows:

- U-net with BCE and dropout = 0.2, base = 16, batch normalization = True, Data Augmentation = True:



- U-net with dice loss and dropout = 0.2, base = 16, batch normalization = True, Data Augmentation = True:



When we check the results based on the changes we applied on the data and the model settings, we see that the training process started to fluctuate a lot in validation loss and validation accuracy especially. This might be because of the various data augmentations we applied on the image. This might show that data augmentation affects segmentation process a lot. Still, after some time, the validation loss achieves to catch training loss. This contrasts with the previous task which had a gap between validation and training loss. If we check this setting for a longer epoch, we can have a better idea on its durability against overtraining. But it should be considered that the validation loss is higher than previous models and validation accuracy is not as high as previous models. This might be because the model is learning much slower in these settings. If we check the end result, we might say that with batch normalization and data augmentation, u-net trained for segmentation process can avoid overfitting although it might take longer time to train. Hence, this model can be more generalizable compared to other models with more epochs.