Gerard Castells and Mariona Carrasco

Deep Learning Methods for Medical Image Analysis (CM2003)

LAB 4

Transfer Learning

Task 1

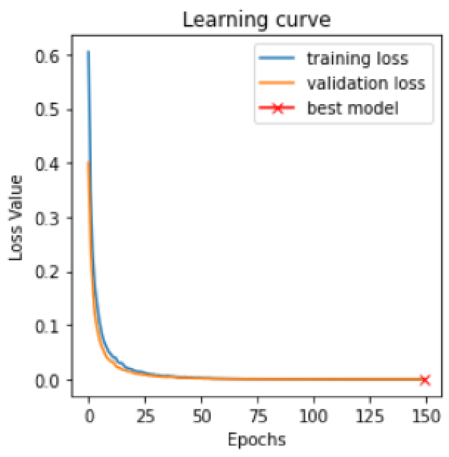
In this task, you will employ a pre-trained VGG16 model that was trained on the ImageNet database. By fine-tuning this model, you will classify the skin and bone data set again. To do so, as the first step, the pre-trained model should be loaded. This pretarined model will be
used as a feature extractor model. In other words, training and validation data should be passed through the layers of the pre-trained VGG network, and the output of last convolutional block will be used as the extracted features. These features, then, are fed to the new layer(s) in a way that only the newly added layer(s) will be trained. Follow the code and complete it as:

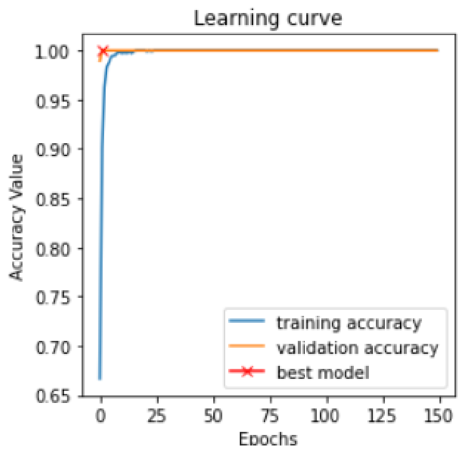
Done in the python file.

Task 2

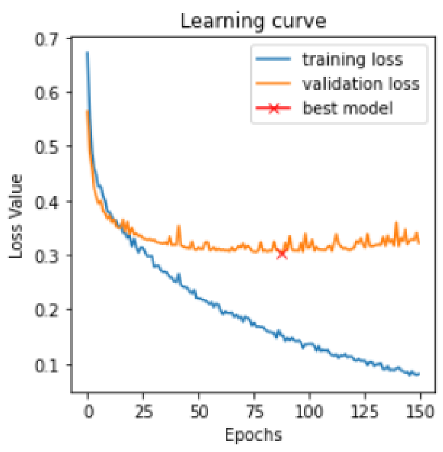
Train the fine-tuned model with skin and bone images. Compare the observed results from transfer learning against the ones you already achieved by training the models from scratch. Describe how transfer learning would be useful for training. How can you make sure that the results are reliable?

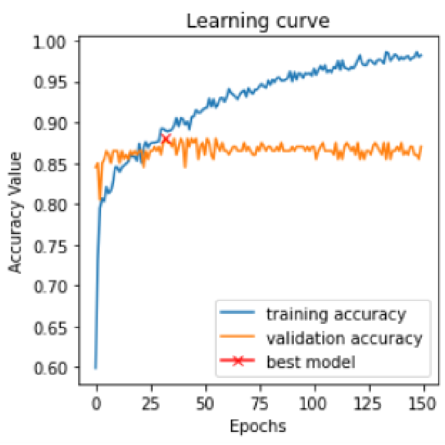
Bone:





Skin:





Transfer learning is useful for training because it leverages pre-existing knowledge, accelerates training, improves performance, enhances generalization, and can be applied efficiently even in scenarios with limited data. Transfer learning saves time and computational resources, leading to more efficient and effective training processes.

As we can see, in both types of images we can observe that the values obtained for accuracy and loss are high and considered positive. In bone images transfer learning has worked so well with values of 0 on loss and 1 of accuracy with few epochs. In contrast, on skin images it occurs overfitting.

To ensure reliable results in transfer learning, we need to choose an appropriate pre-trained model, use high-quality data, find parameters carefully, validate and test rigorously, apply data augmentation and explore ensemble methods when needed. To sum up, if we properly implement this practice of transfer learning, it can be a powerful tool in deep learning.

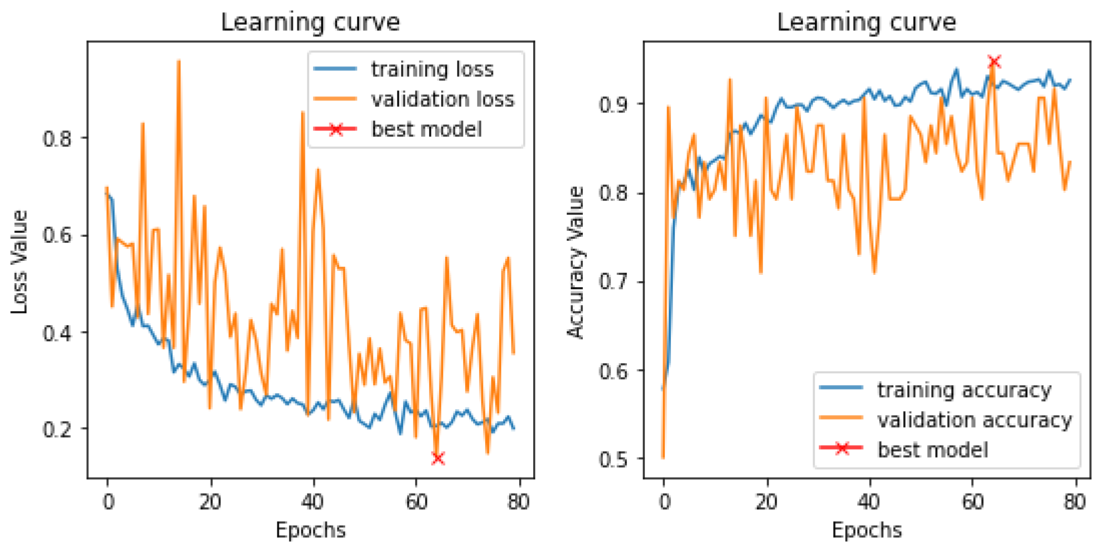
Visualizing Activation Maps

Task 3

Design a VGG16 model similar to what you already implemented in the lab 2 task 7a. Please note the model should contain 3 dense layers at the top each of which contain 64 neurons as well as dropout layer (rate=0.4). Finally, a dense layer with 2 neurons at the last layer so that the activation function should be set as “softmax” and

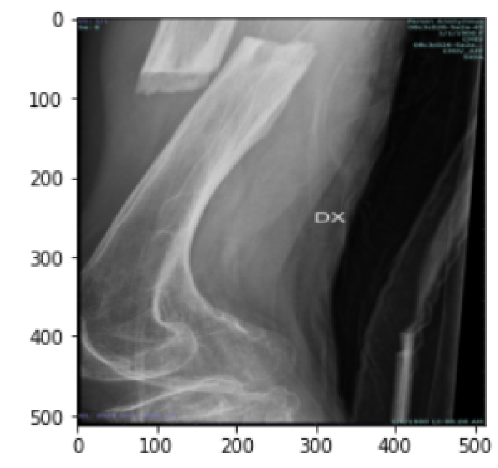
categorical cross entropy as loss function. After the training process, follow the implementation below and interpret the observed results. What can you infer from visualization of the class activation maps?

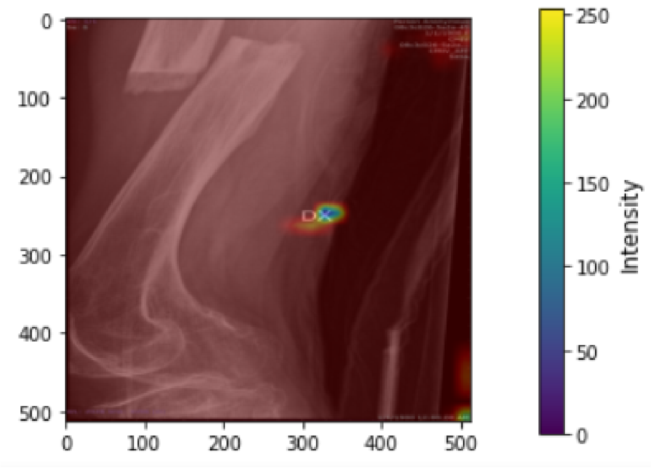
First, the training process has presented highly acceptable values in accuracy and loss (0.92 and 0.17), even though the process took around 33 seconds per epoch. In order to make this process work, we've used a simplified VGG16 architecture, and also increased the learning rate to 1e-3. With those modifications, the learning curve obtained is the following:



Class activation maps are a valuable tool for interpreting and analyzing the behavior of deep learning models in image classification tasks. They provide insights into where the model is looking and what features it finds important, which can be crucial for improving model performance, diagnosing issues, and ensuring model fairness and transparency.

From the visualization of the class activation map we can observe which are the regions which infer more in the task of the image classification. We can see that the region where the image details are exposed, is highlighted in cooler colors like blue, whereas the rest of the image is in a more monoton red color. So, in those given images the regions used to do the classification task are where the labels describing the type of image appear.





Segmentation Task

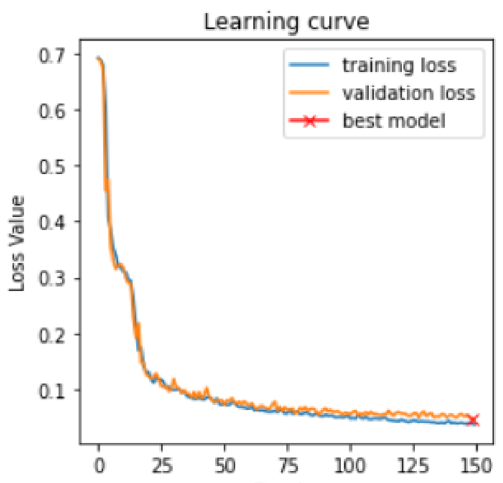
Task 4

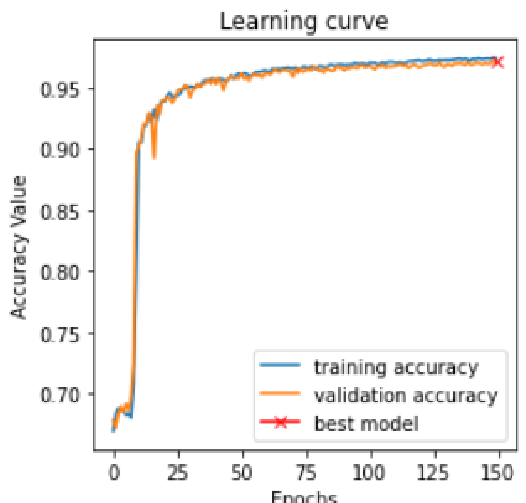
Develop a modular segmentation pipeline in a way that your pipeline contains separate functions for different steps such as image loading, segmentation mask loading, augmentation generator, similarity metrics, model architecture, and model training.m Therefore, you should have a main file containing only a few lines of codes for calling the functions and setting the parameters to run the experiments. Please remember in segmentation tasks, in general, each image has a corresponding segmentation mask. Therefore, it is quite essential to load the image and corresponding mask with the same order.

Done in Unet.py python file.

Task 4a

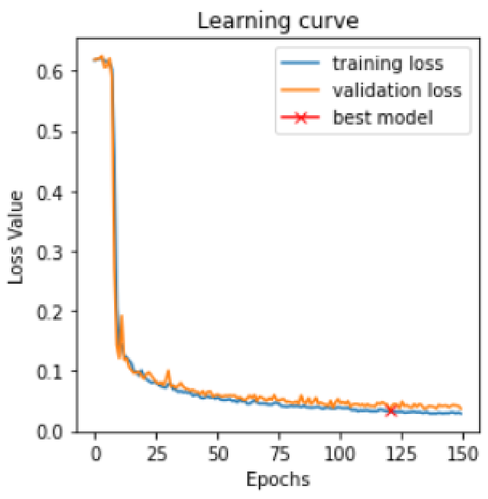
Lung segmentation in chest X-ray images

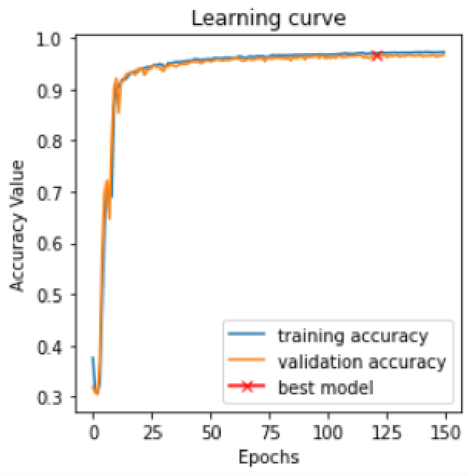




Task 4b

Keep all the parameters from the previous exercise the same, only replace the BCE loss function with "Dice loss" and repeat the exercise.





Is there any difference between model performance over validation set when you changed the loss functions? Do you expect to observe similar results when you deal with more challenging segmentation tasks? Dealing with cases in which the size of the target to be segmented would be much smaller than the image (imbalance between the background and foreground). Which loss function would you choose? Discuss it.

Both BCE (Binary Cross-Entropy) loss and Dice loss are commonly used loss functions for image segmentation tasks, and their performance can vary depending on the specific problem and dataset. As we can observe, yes it is different in our case, with BCE loss values being a little bit higher than with the Dice loss function. It occurs the same with accuracy values. Even though, with the two functions we obtained similar and good results too.

On the one hand, BCE loss is a pixel-wise loss function that treats each pixel in the segmented mask independently. It measures the dissimilarity between the predicted pixel values and the ground truth mask. It is good at penalizing false positives and false negatives individually, making it suitable for tasks where precision and recall are important.

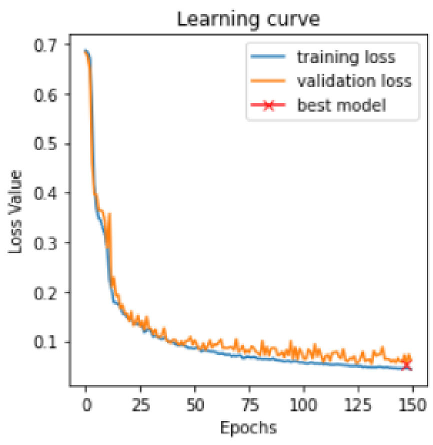
On the other hand, Dice loss is preferable for challenging segmentation tasks with imbalanced foreground and background classes. Because it handles class imbalance better by emphasizing overlap between predicted and ground truth masks. It encourages smoother segmentations with well-defined object boundaries, which is important for smaller target objects.

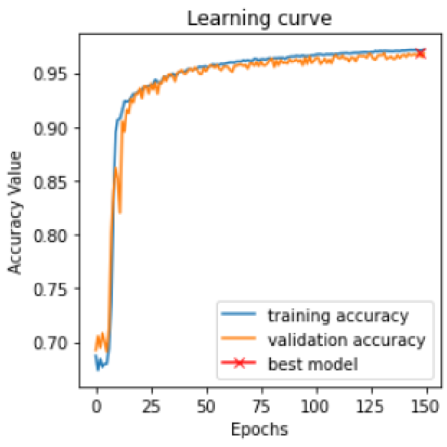
When dealing with cases in which the size of the target to be segmented would be much smaller than the image, we would typically expect better performance with Dice loss compared to BCE loss. Dice loss is particularly useful for imbalanced datasets because it focuses on the intersection of the predicted and ground truth masks, which helps address the class imbalance problem. Dice loss encourages the model to produce better boundary delineation and can lead to more accurate segmentation, especially when dealing with smaller foreground objects.
However, BCE loss can have problems with class imbalance issues, such as when the background class significantly outweighs the foreground class. This can lead to predicting the majority class (background) and may result in poor segmentation of the smaller, foreground objects.

Task 4c

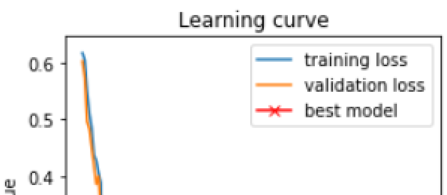
Repeat the tasks 1a and 1b by including the dropout layer (rate=0.2). Does it affect model performance? What about the learning curves?

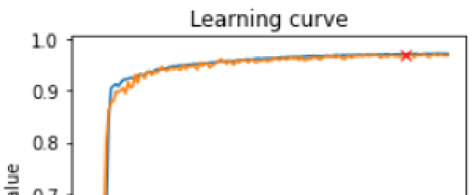
BCE





Dice loss



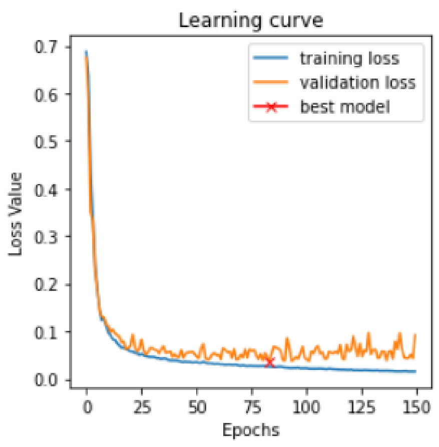


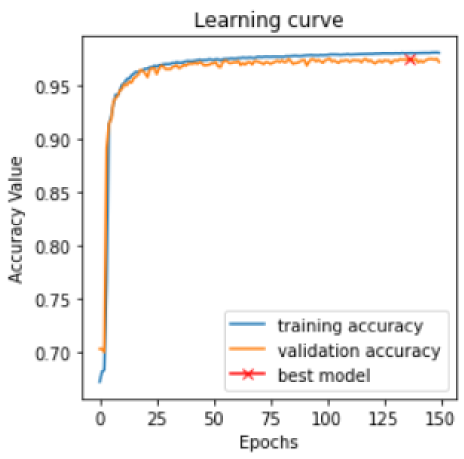
The addition of a dropout layer with a rate of 0.2 to our neural network can affect both the model performance and the learning curves. Dropout is a regularization technique used to prevent overfitting in neural networks by randomly setting a fraction of input units to zero during each forward and backward pass.
However, in our case it has not affected much with the two different functions. Because our model is already performing well without overfitting, so we don't see any improvement with dropout.

Task 4d

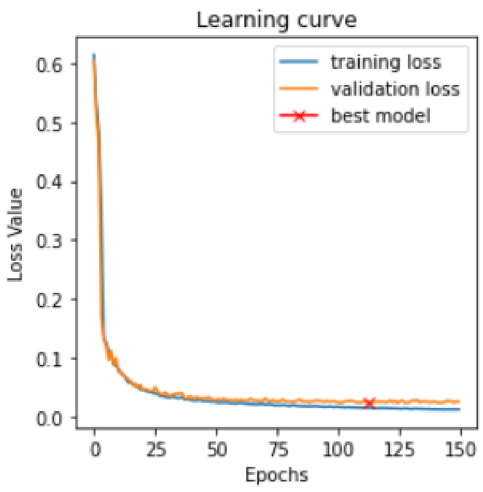
Increase the model capacity by setting base=32. Repeat tasks 1c and evaluate the results.

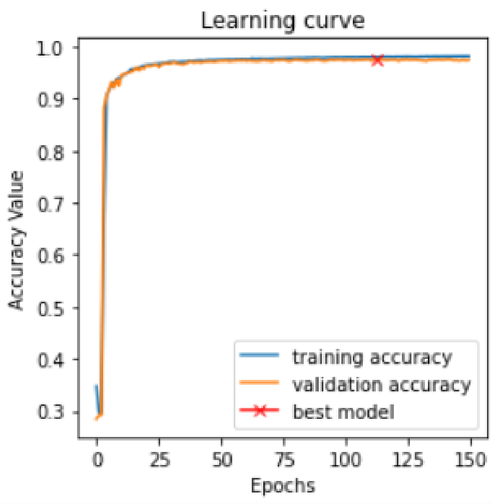
BCE





Dice loss



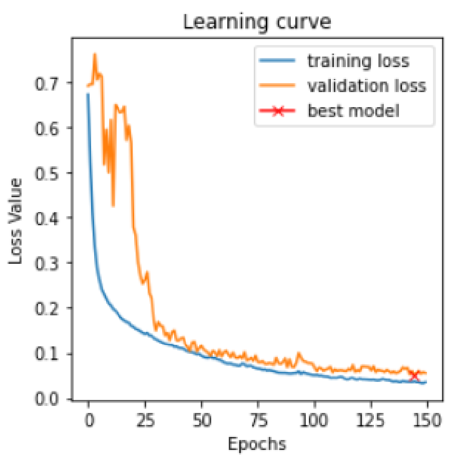


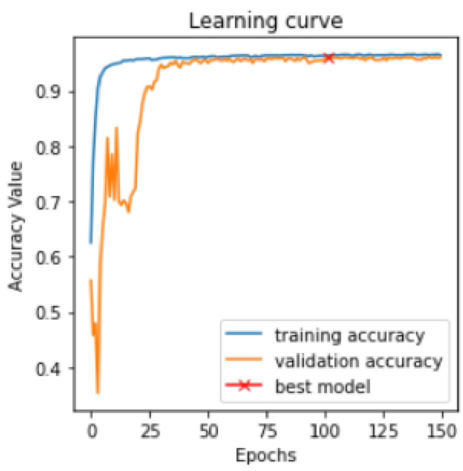
Increasing the base value from 8 to 32 in the Unet architecture has significantly increased the model's capacity and complexity. With more filters in each layer, the total number of trainable parameters in the model increases substantially. However, this can make the model more prone to overfitting if you have limited training data. In our case, we obtain similar results but validation loss and accuracy gets more stabilized so, the larger model capacity has had a positive impact on the model's ability to generalize and learn complex features from the data.

Task 4e

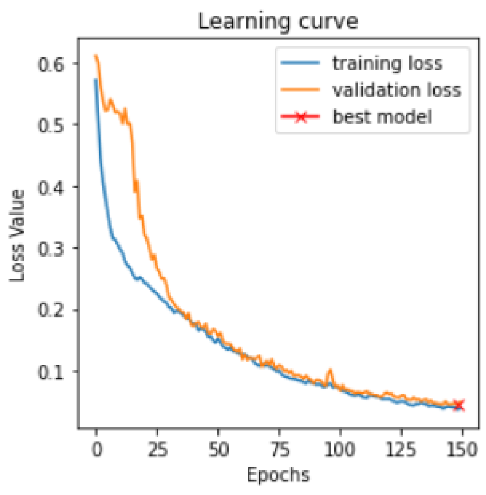
Repeat the task 1c with setting the base=16, batch norm=True, and train the model by applying augmentation technique on both images and masks: rotation range=10; width and height shift ranges=0.1; zoom range = 0.2, and horizontal flip. Could image augmentation improve the generalization power of the model?

BCE





Dice loss



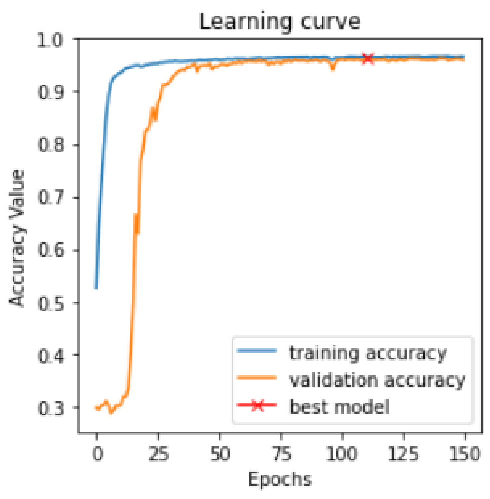


Image augmentation is a valuable tool for improving the generalization power of deep learning models. When used appropriately, it can lead to more robust and accurate models, especially when the training dataset is limited or when the real-world data can vary in ways that are hard to anticipate.
It also reduces overfitting by introducing randomness and noise, making the model less likely to memorize training examples. Additionally, augmentation helps the model become more robust to real-world conditions like changes in lighting or perspective. Moreover, it effectively increases the effective dataset size, which can lead to better model performance, even when the original dataset is limited.

However, it's important to strike a balance when applying image augmentation. Too much augmentation or inappropriate transformations may lead to over-regularization or introduce unrealistic data, which can harm model performance. It's essential to monitor the model's performance on a validation set to ensure that augmentation is benefiting generalization.

Evaluating our learning curves we could see that data augmentation has not improved our result because we have obtained worse results of accuracy and loss value in comparison to task 4c. Moreover, the regulation on curves is worse than without data augmentation, so our arguments inputs are not accurate. We may need to experiment with different augmentation strategies, model architectures, and hyperparameters to obtain better results.