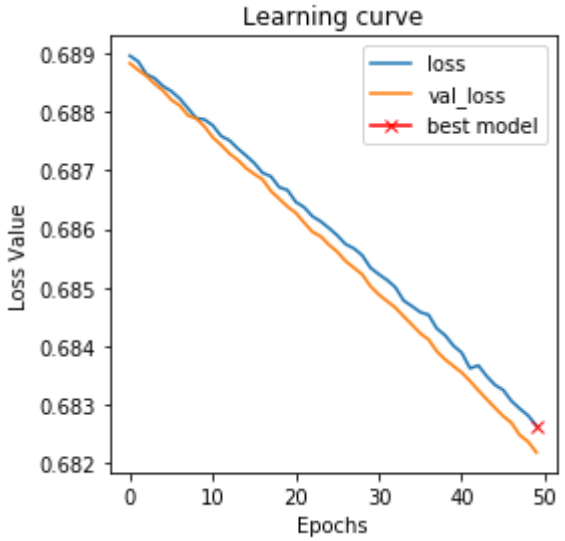
Task6A) Read the skin images with the size of 128*128 and train the AlexNet model with the following parameter: batch size = 8; epochs = 50; n_base(Base) = 32; learning rate = 0.0001, and Adam as optimizer. **Evaluate the model performance.**

Answer :

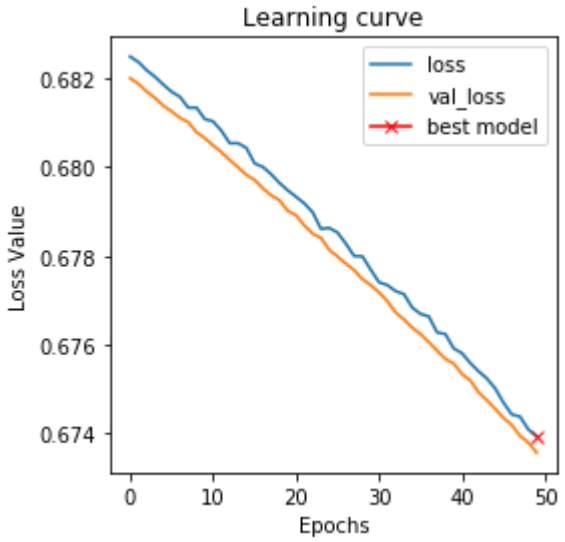


Here the training and validation loss values follow each other closely, but they have not converged, at least not after 50 epochs.

Task6B) Change the n_base parameter as 16 and 8 and run the model for 50 epochs. **How do you interpret the observed results?** Now, with n_base = 8, after each of the dense layer add a “drop out layer” with a drop out rate of 0.4 and train the model for 50 epochs. **What is the effect of the drop out layer?** Increase the number of epochs to 150. **How do you explain the effect of increasing the number of epochs?**

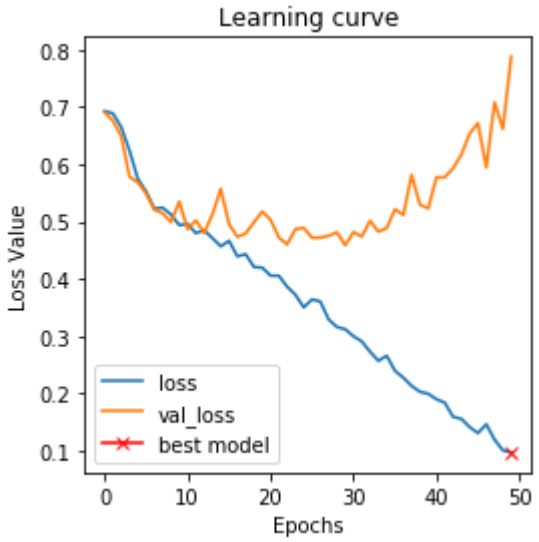
Answer :

n_base(Base) = 16
Epoch 50 : loss: 0.6739 - binary_accuracy: 0.6720 - val_loss: 0.6736 - val_binary_accuracy: 0.7200



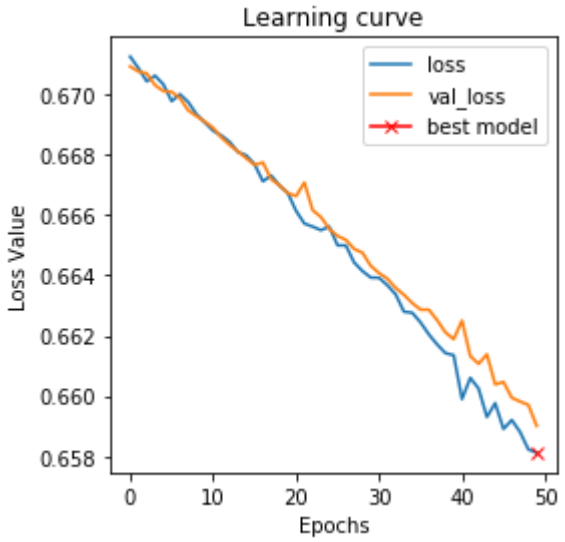
Same here as in 6A), we have no convergence yet.

n_base(Base) = 8
Epoch 50: loss: 0.0968 - binary_accuracy: 0.9750 - val_loss: 0.7878 - val_binary_accuracy: 0.7450



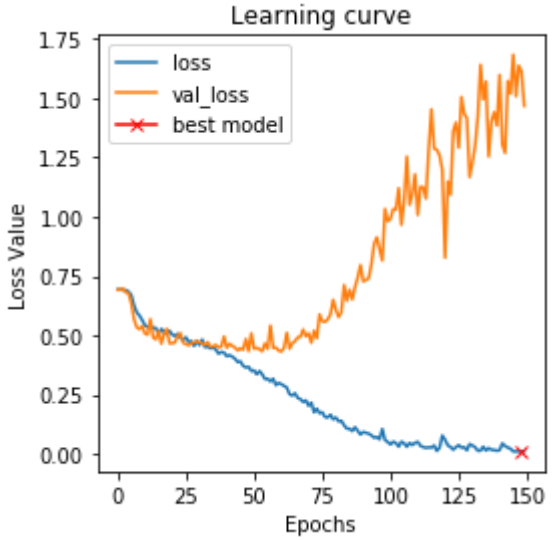
When lowering the number of features the model does not have enough features to actually learn, it only remembers the training data. Therefore the loss value for training is good but when the model is presented to new images it performs poorly.

Dropout layer, n_base = 8, epochs = 50



Adding dropout layers forces the model to learn from the limited features. So in the beginning of the training, two performance curves are close to each other. But by the end of the 50th epoch, the performance of the model is still not able to be judged, because the loss values are changing in the process.

Dropout layer, n_base = 8, epochs = 150
loss: 0.0114 - binary_accuracy: 0.9970 - val_loss: 1.4683 - val_binary_accuracy: 0.7750



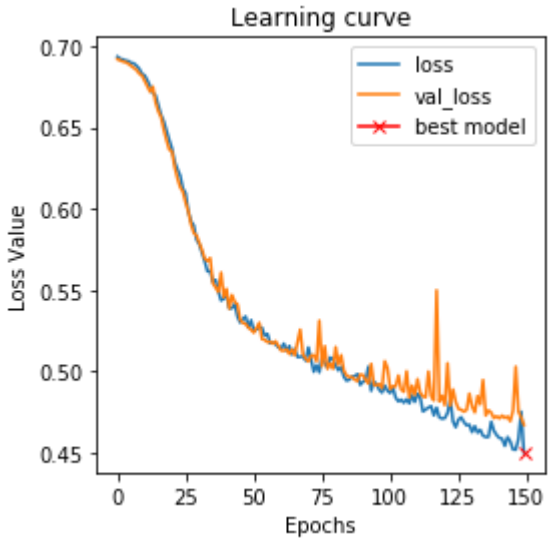
When training the model with more epochs, the model is overfitting in the end. From the results, it indicates that when lacking features from convolution layers, the model is prone to remember data instead of learning from it.

Task6C) Remove the drop out layers, set the parameters n_base=8, learning_rate = 1e-5 and run the model for n_epochs =150,350 epochs. **How changing the learning rate parameter affects model performance?**

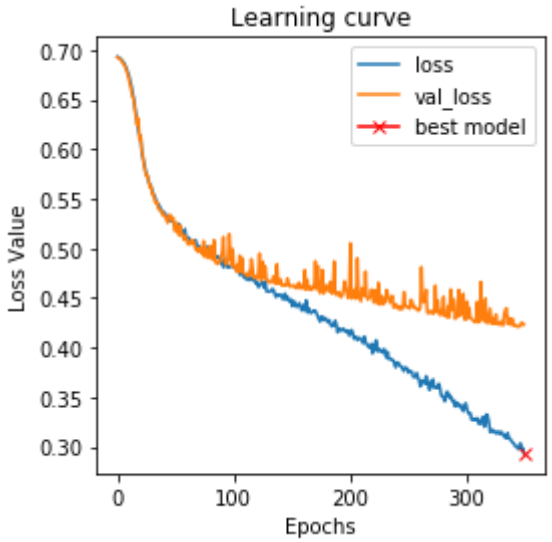
Answer :

The grid of the figure of results is very small. The training loss of the model is slightly decreasing. The validation loss during 150 epochs changes slightly as well. And at the end of the 50th epoch, we can't really judge the performance of the model because the curves are not converging and training loss is still decreasing.

n_base=8, learning_rate = 1e-5, n_epochs =150,
1000/1000： loss: 0.5853 - binary_accuracy: 0.6990 - val_loss: 0.6246 - val_binary_accuracy: 0.7000



n_base=8, learning_rate = 1e-5, n_epochs =350
Epoch 350: loss: 0.2933 - binary_accuracy: 0.8850 - val_loss: 0.4233 - val_binary_accuracy: 0.8200

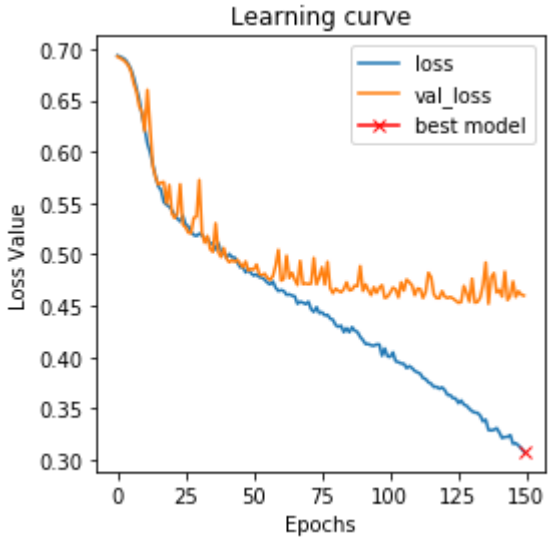


When adding the number of epochs to be 350, the difference of performance between training data and testing data becomes bigger during the process. With a small learning rate, two curves are similar in the beginning. But in the end, the model is overfitting, which indicates that the performance is not improved with a smaller learning rate when the model lacks features.

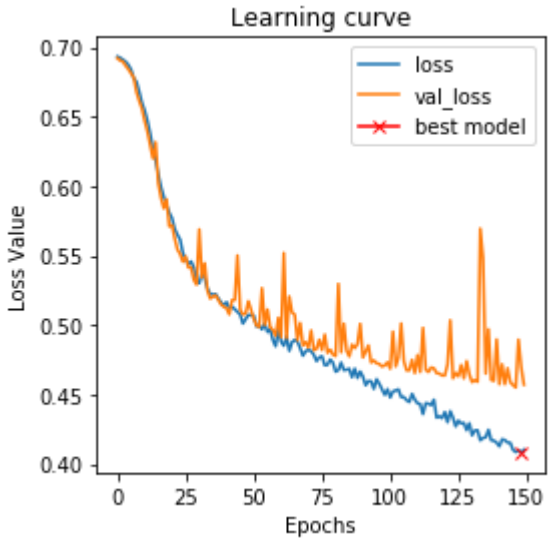
Task6D) For the fixed parameters of learning_rate = 1e-5, n_base=8, n_epochs = 150, change the batch size parameters as 2,4,8. **Do you see any significant differences in model performance?**

Answer :

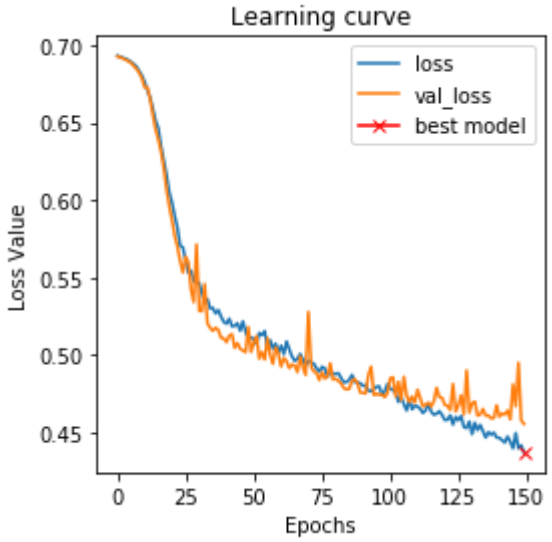
learning_rate = 1e-5, n_base=8, n_epochs = 150, batch size = 2
- loss: 0.3080 - binary_accuracy: 0.8710 - val_loss: 0.4599 - val_binary_accuracy: 0.8300



learning_rate = 1e-5, n_base=8, n_epochs = 150, batch size = 4
loss: 0.4103 - binary_accuracy: 0.8150 - val_loss: 0.4571 - val_binary_accuracy: 0.8150



learning_rate = 1e-5, n_base=8, n_epochs = 150, batch size = 8
loss: 0.5853 - binary_accuracy: 0.6990 - val_loss: 0.6246 - val_binary_accuracy: 0.7000

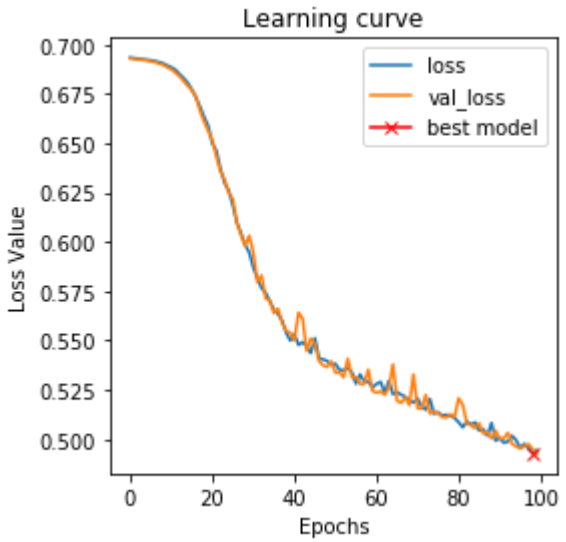


For batch sizes 2 and 4 we see a clear case of overfitting. However the loss value for training is lower with lower batch size, but this is quite irrelevant considering when presented with new data during validation all batch sizes have pretty much the same validation loss value after 150 epochs, with batch sizes 2 and 4 being slightly better.

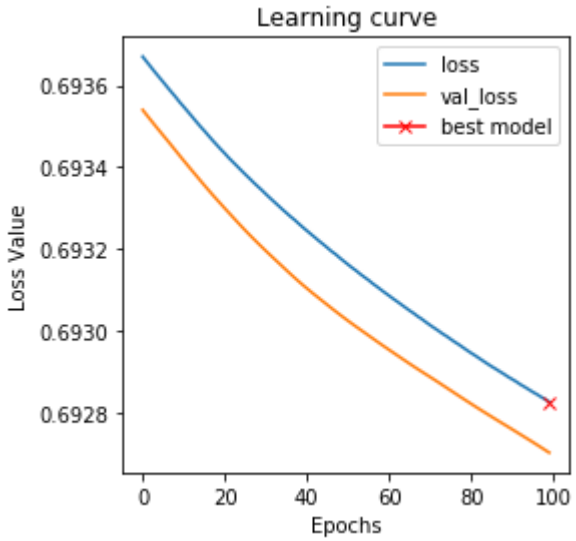
Task6E) By finding the optimum values of batch_size, learning rate, and base parameters, train the model for 100 epochs and make sure *it is not overfitted*. Report the classification accuracy of the model. **Then, for this model, only change the optimizer algorithm from Adam to SGD and RMSprop and compare the observed results.**

Answer :

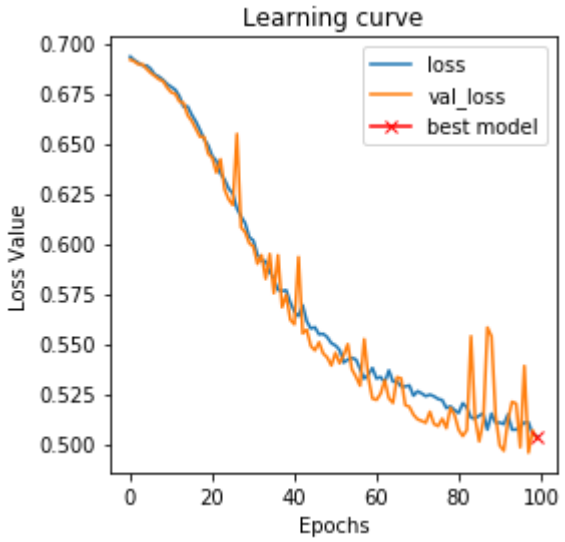
Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam
- loss: 0.4945 - binary_accuracy: 0.7790 - val_loss: 0.4927 - val_binary_accuracy: 0.8050



Batch_size = 8, learning rate = 1e-5, base parameters = 8, SGD
- loss: 0.6928 - binary_accuracy: 0.5000 - val_loss: 0.6927 - val_binary_accuracy: 0.5000



Batch_size = 8, learning rate = 1e-5, base parameters = 8, RMSprop
- loss: 0.5038 - binary_accuracy: 0.7790 - val_loss: 0.5023 - val_binary_accuracy: 0.8100

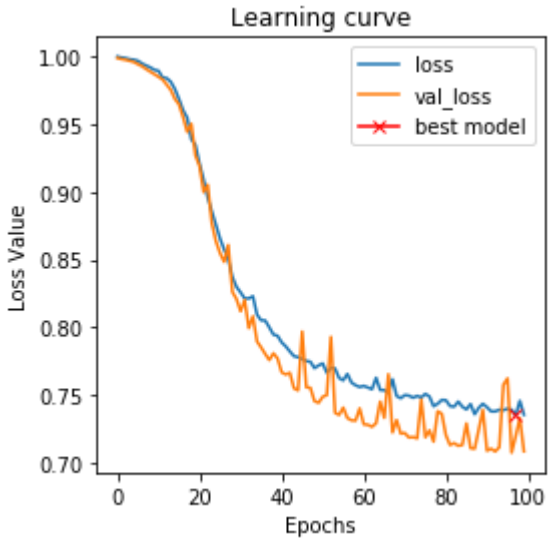


Adam looks to be the best optimizer, the validation and training values follow each other closely and is converging. For RMS prop the validation loss value seems quite unstable and for SGD the graph does not seem to be converging.

Task6F) “Binary cross entropy (BCE)” is not the only loss function for a binary classification task. Run the code again by changing the loss func into “hinge” with Adam optimizer. **What is the major difference between the “BCE” and “hinge” in terms of model performance?** Please note, you need to have class labels as [-1, 1] therefore, you need to change the labels as :
y_test[y_test == 0] = -1
y_train[y_train == 0] = -1

Answer :

Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam, loss func = hinge
- loss: 0.7356 - binary_accuracy : 0.7356 - val_loss: 0.7087 - val_binary_accuracy: 0.7087 ??



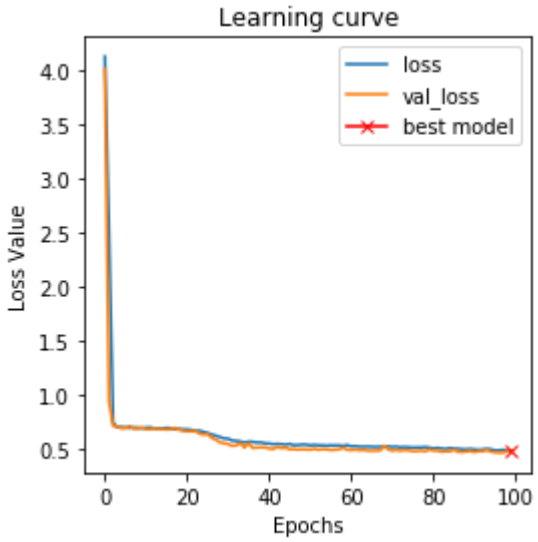
The hinge loss function does not seem to reach as low loss values compared to binary cross entropy loss function.

Taks7A) The purpose of this task is to implement an architecture from scratch. You have to implement the architecture of the VGG16 model, as shown in the following figure. Please note 3 ×3 represents the size of the convolutional kernels, “Base” shows the number of feature maps, “pool/2” indicates the max-pooling operator, and “fc 64” means fully connected (dense) layer with 64 neurons.

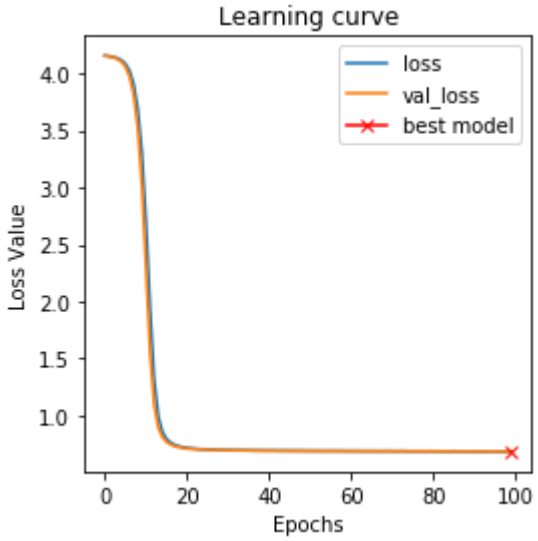
Task7B) Load the Skin data set with the size of 128*128. For the following fixed parameters, compile the model and find the optimum value of the learning rate that will lead to reliable classification performance over the validation set. **What is the proper learning rate?**

Answer :

Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam,
loss func = sparse_categorical_crossentropy
- loss: 0.4817 - accuracy: 0.7780 - val_loss: 0.4837 - val_accuracy: 0.8200



Batch_size = 8, learning rate = 1e-6, base parameters = 8, Adam,
loss func = sparse_categorical_crossentropy
- loss: 0.6851 - accuracy: 0.5630 - val_loss: 0.6842 - val_accuracy: 0.6000



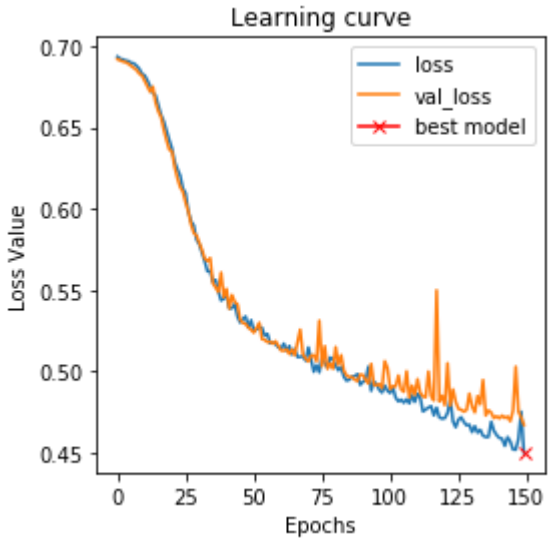
Both of the graphs have converged, but the learning rate 1e-5 has reached a lower loss value and higher accuracy.

Task7C) For the same setting as task6C (only for n_epoch=150) compare the classification accuracy over the validation set between the AlexNet and VGG16 models. **How do you justify the observed results?**

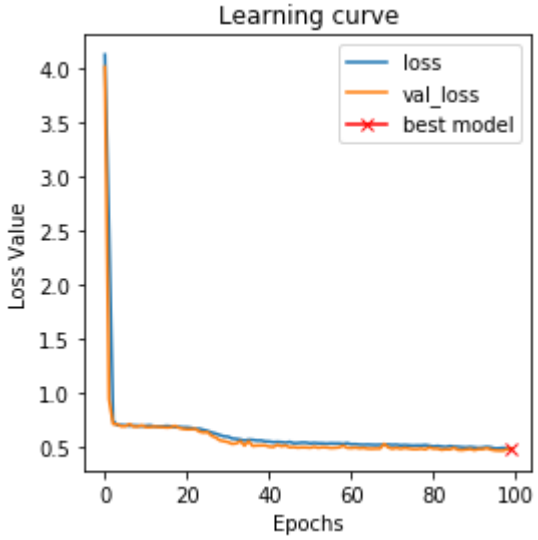
Answer :

(AlexNet)
n_base=8, learning_rate = 1e-5, n_epochs =150,

1000/1000： loss: 0.5853 - binary_accuracy: 0.6990 - val_loss: 0.6246 - val_binary_accuracy: 0.7000



(VGG16)
Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam,
loss func = sparse_categorical_crossentropy
- loss: 0.4817 - accuracy: 0.7780 - val_loss: 0.4837 - val_accuracy: 0.8200

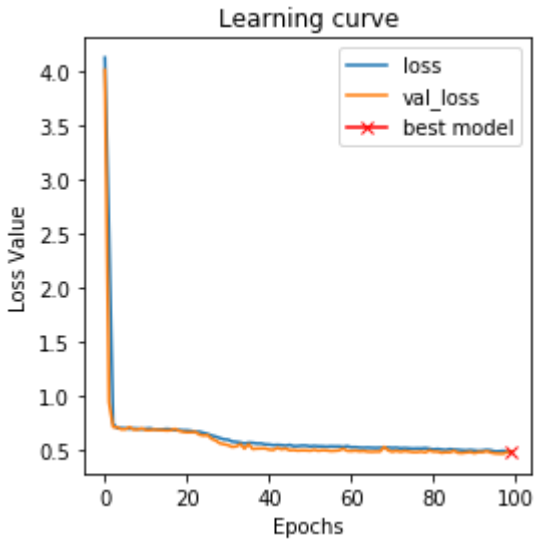


The VGG16 model reaches a higher validation accuracy. This model includes more convolutional layers than AlexNet, and therefore is better at classifying features.

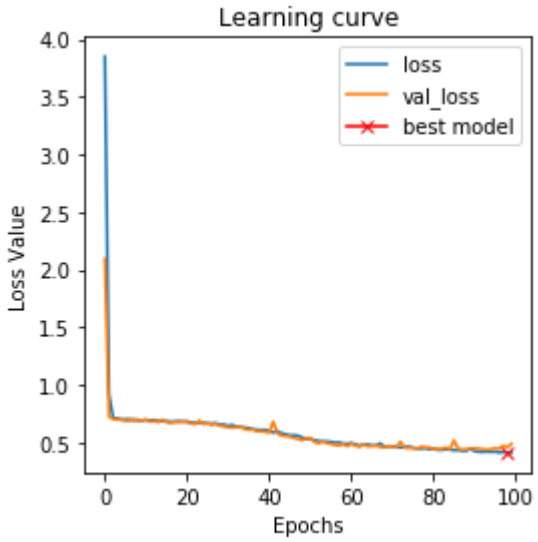
Task7D) Change the parameter n_base to 16. Train the model for 100 epochs with LR = 1e-5 and batch size of 8. **Does increasing the number of feature maps affect model performance**? Then, add a dropout layer with a rate of 0.2 for each of the dense layer. **What was the effect of adding the dropout layer?**

Answer :
Increasing the number of feature maps doesn't affect model performance that much.
When adding dropout layers for each dense layer, first the learning process of the model slows down a bit, which could be found from the curve.
And the performance on validation data is better than on training data. But two curves seem to converge at the end of the training. But it takes longer for the model to reach a stable performance until convergence.

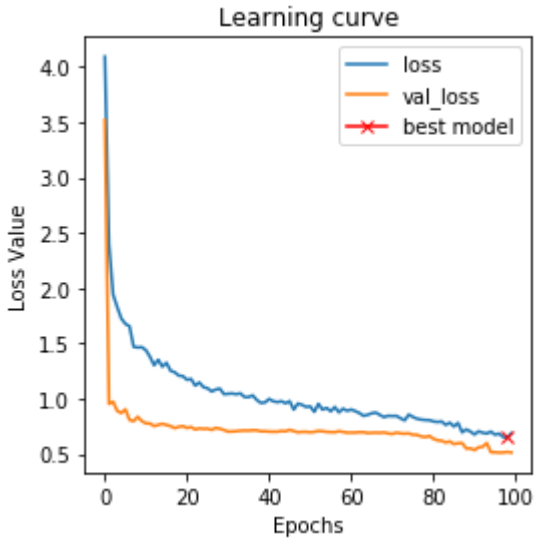
Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam,
loss func = sparse_categorical_crossentropy, without dropout layers
- loss: 0.4817 - accuracy: 0.7780 - val_loss: 0.4837 - val_accuracy: 0.8200



Batch_size = 8, learning rate = 1e-5, base parameters = 16, Adam,
loss func = sparse_categorical_crossentropy, without dropout layers
- loss: 0.4110 - accuracy: 0.8130 - val_loss: 0.4873 - val_accuracy: 0.7900



Batch_size = 8, learning rate = 1e-5, base parameters = 16, Adam, loss func = sparse_categorical_crossentropy, with dropout layers of 0.2
- loss: 0.6889 - accuracy: 0.6760 - val_loss: 0.5144 - val_accuracy: 0.8300



Task7E) So far, you classified the Skin cancer dataset with 3 different models named as LeNet, AlexNet as well VGG16. **In general, what is the difference between these three models? Which of them yields more accurate classification results? Why? To evaluate the model performance, how do you assess the loss values? How can you prevent the model training from overfitting?**

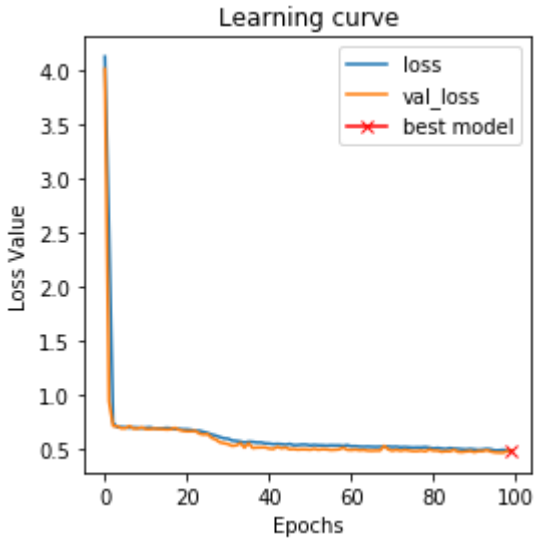
Answer :

From the angle of architecture, LeNet is a basic convolutional network. And AlexNet is more complex than LeNext which has more convolutional layers. And the problem with both LeNet and Alex is that these two models have a large number of parameters.
While VGG16 is a very deep convolutional network and has fewer parameters than the other two. VGG has the best results among the three. We understand that VGG has better ability in feature extraction with more convolutional layers.

For evaluation of loss values, first, we check if the loss value is prone to be stable. Second, the difference of loss value between training set and validation set is compared. And conclusion could be overfitting, underfitting or good performance according to the gap.

Adjusting hyperparameters could avoid overfitting, but adjustment will highly depend on the properties of datasets. For example, using a small learning rate could sometimes avoid overfitting.

VGG16
Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam,
loss func = sparse_categorical_crossentropy, without dropout layers
- loss: 0.4817 - accuracy: 0.7780 - val_loss: 0.4837 - val_accuracy: 0.8200



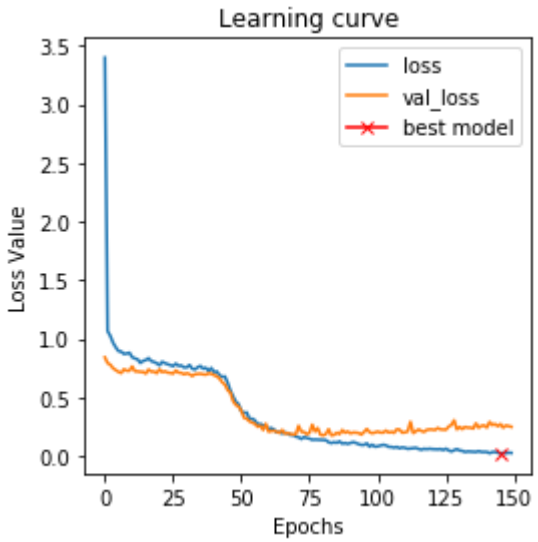
Task8) Train the VGG16 model developed in Task7 with the following parameters to classify two types of bone fractures from “Bone” dataset. Please note the size of the Bone images is quite large, so that it would take a longer time to read and load all the images.

Task9) With the implemented VGG models, **which of the Skin/Bone image sets classified more accurately? Why? How do you make sure that achieved results are reliable?**

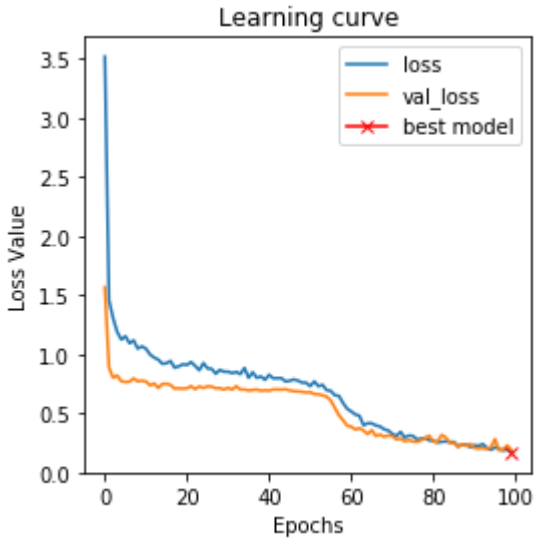
Answer :
For bone images, the VGG model is more accurate.

From the first figure, we can tell that the loss curves converge. But since the training epochs are not enough to see more training data and performance.
So the results by now are not completely reliable.
When we try without dropout layers, the model is overfitting, which indicates that dropout layers should be kept in this case.
Although curves seem to converge with dropout layers, the performance is not reaching a stable status. To investigate the performance, we changed the epochs to be 150. And from the curves, it is overfitting.

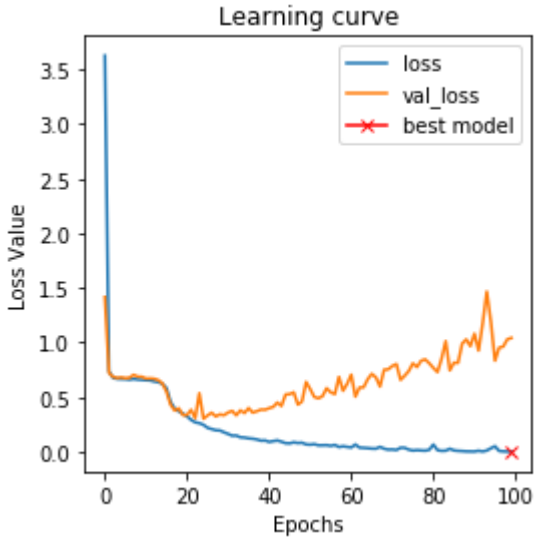
VGG n-base = 16 with dropout layers, 150 epochs
Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam,
loss func = sparse_categorical_crossentropy
- loss: 0.0247 - accuracy: 0.9916 - val_loss: 0.2474 - val_accuracy: 0.9643



VGG n-base = 16 with dropout layers
Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam,
loss func = sparse_categorical_crossentropy
- loss: 0.0482 - accuracy: 0.9897 - val_loss: 0.2643 - val_accuracy: 0.9000



VGG n-base = 16 without dropout layers
Batch_size = 8, learning rate = 1e-5, base parameters = 8, Adam,
loss func = sparse_categorical_crossentropy
- loss: 0.0020 - accuracy: 1.0000 - val_loss: 1.0425 - val_accuracy: 0.9214



Task10) In previous exercises, you conducted some experiments with binary classification tasks by implementing three deep networks named as LeNet, AlexNet, VGG. In this task, the data set includes X-ray images of 9 different organs. Therefore, you are expected to extend the implemented models into a multi-class classification task. Modify the data loader to load the images along with their class labels properly. Extend the LeNet and AlexNet models for multi- class classification tasks. Tune these two models by finding the optimum values of hyperparameters to get the best performance for each of the models and, then, compare the observed results between the two models.

Report the learning curves for both of the loss and accuracy values (for train and test data).

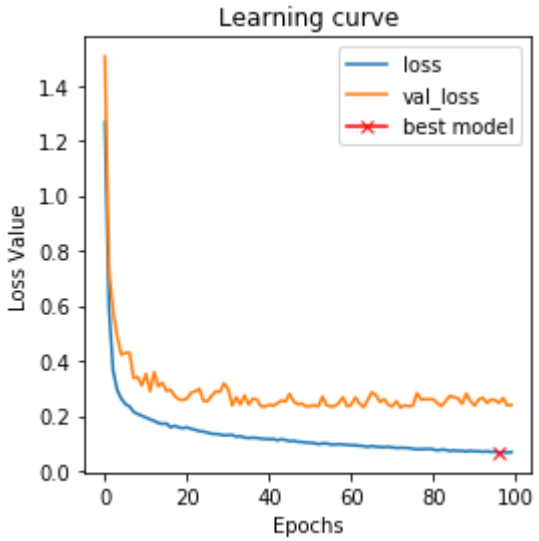
Data path is: '.../Lab1/X_ray/'

Answer :
For the AlexNet model, the best hyperparameters include n_base = 8, batch_size = 8, LR = 1e-5, dropout layers(0.4) = 2 and n_epochs = 150. In the end, the performance of the AlexNet model reaches convergence and the validation accuracy turns out to be 93%. During the testing, we found out that the model is prone to be overfitting without dropout layers or with higher numbers of features in the AlexNet model.

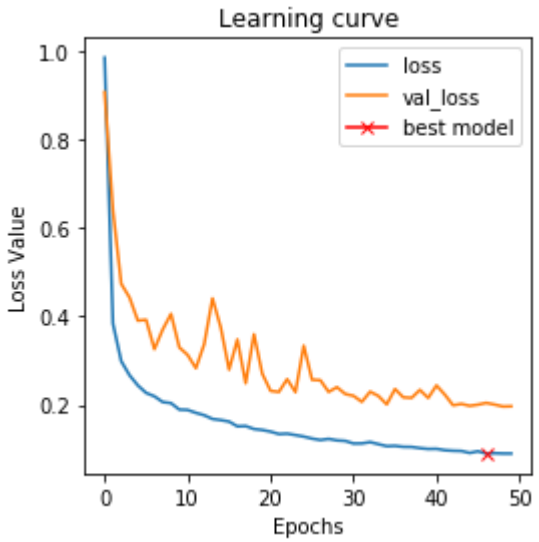
For the LeNet model, we tried multiple hyperparameters and found that the model is overfitting without dropout layers. The accuracy on the training set could be very high while the validation accuracy can't reach the same level.
So we tried to add a dropout layer to the model. From the results, it shows that the performance of the model seems to be stable after 100 epochs of training. And the difference of accuracy between training data and validation data becomes smaller compared to the one without a dropout layer. In the end, the best model is with validation accuracy around 93% while the training accuracy is around 97.5%.

Comparing the results from LeNet and AlexNet, there is no big difference in the best performance they can reach. But from the curves of loss value, LeNet seems to converge faster than AlexNet with similar hyperparameters in this dataset. And in our view, the architecture of LeNet is simpler than Alex.

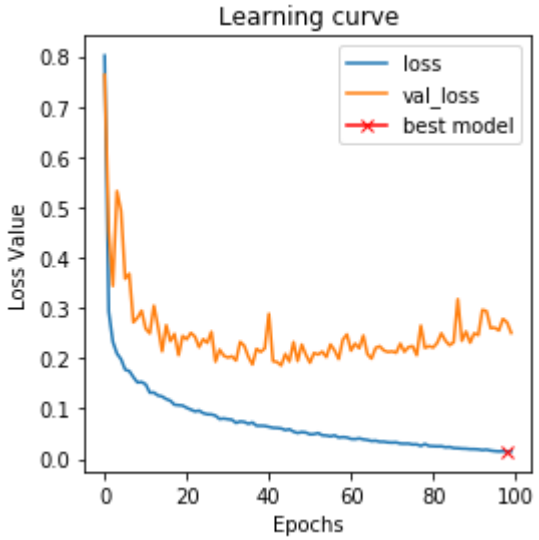
LeNet with 1 dropout layer (0.4) **(Best Performance)**
Base = 16, n_epochs = 100, Batch_Size = 4, LR = 1e-5
- loss: 0.0686 - accuracy: 0.9746 - val_loss: 0.2395 - val_accuracy: 0.9356



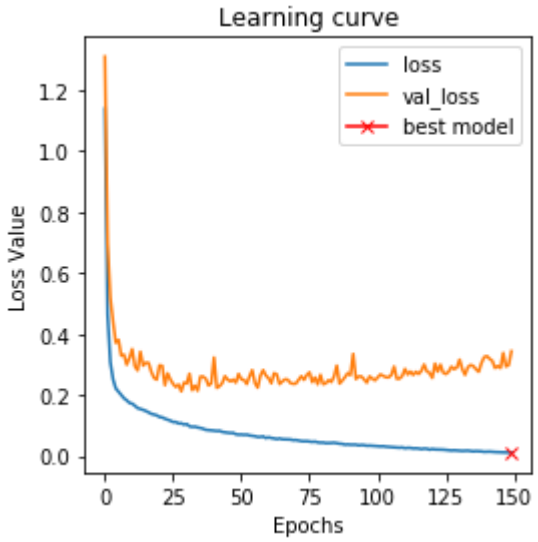
LeNet with 1 dropout layer (0.4)
Base = 16, n_epochs = 50, Batch_Size = 2, LR = 1e-5
- loss: 0.0896 - accuracy: 0.9678 - val_loss: 0.1966 - val_accuracy: 0.9333



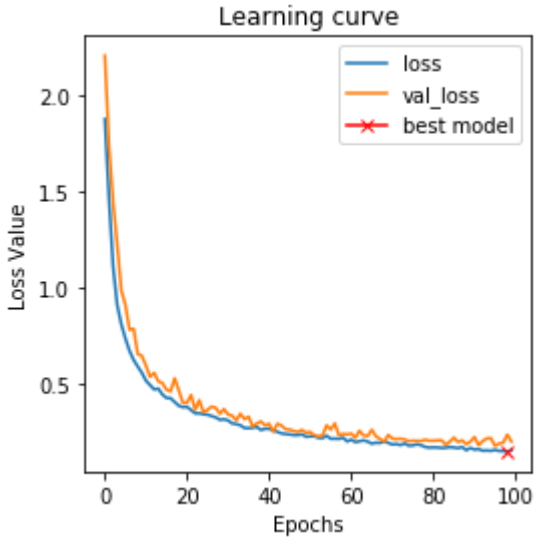
LeNet
Base = 32, n_epochs = 100, Batch_Size = 8, LR = 1e-5
- loss: 0.0152 - accuracy: 0.9957 - val_loss: 0.2504 - val_accuracy: 0.9222



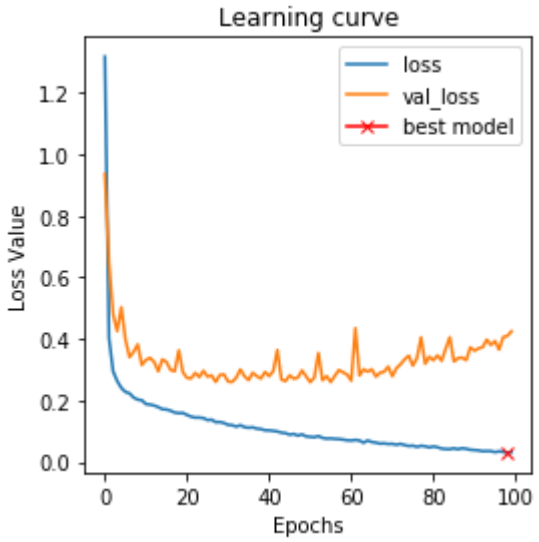
LeNet
Base = 16, n_epochs = 150, Batch_Size = 2, LR = 1e-5
- loss: 0.0105 - accuracy: 0.9979 - val_loss: 0.3432 - val_accuracy: 0.9178



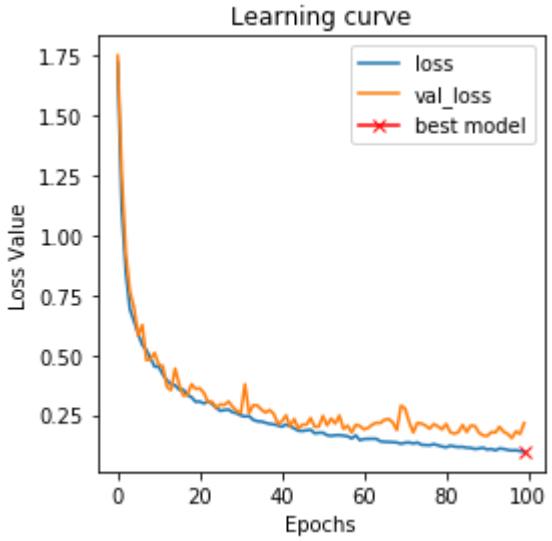
AlexNet with 2 dropout layers (0.4) **(Best Performance)**
Base = 8, n_epochs = 100, Batch_Size = 8, LR = 1e-5
- loss: 0.1558 - accuracy: 0.9434 - val_loss: 0.2008 - val_accuracy: 0.9289



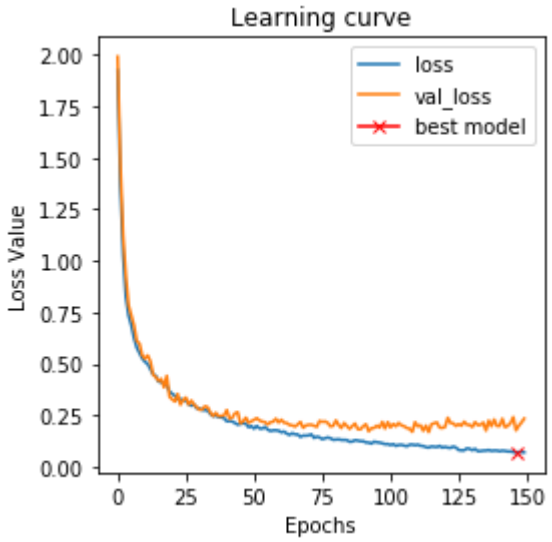
AlexNet
Base = 16, n_epochs = 100, Batch_Size = 8, LR = 1e-5
- loss: 0.0314 - accuracy: 0.9869 - val_loss: 0.4246 - val_accuracy: 0.9222



AlexNet with 2 dropout layers (0.4)
Base = 16, n_epochs = 100, Batch_Size = 8, LR = 1e-5
- loss: 0.0966 - accuracy: 0.9638 - val_loss: 0.2158 - val_accuracy: 0.9400



AlexNet with 2 dropout layers (0.4)
Base = 16, n_epochs = **150**, Batch_Size = 8, LR = 1e-5
- loss: 0.0703 - accuracy: 0.9746 - val_loss: 0.2338 - val_accuracy: 0.9311



AlexNet with 2 dropout layers (0.4)
Base = 32, n_epochs = **200**, Batch_Size = 8, LR = 1e-5
- loss: 0.0208 - accuracy: 0.9917 - val_loss: 0.2868 - val_accuracy: 0.9489