Chen Siyuan

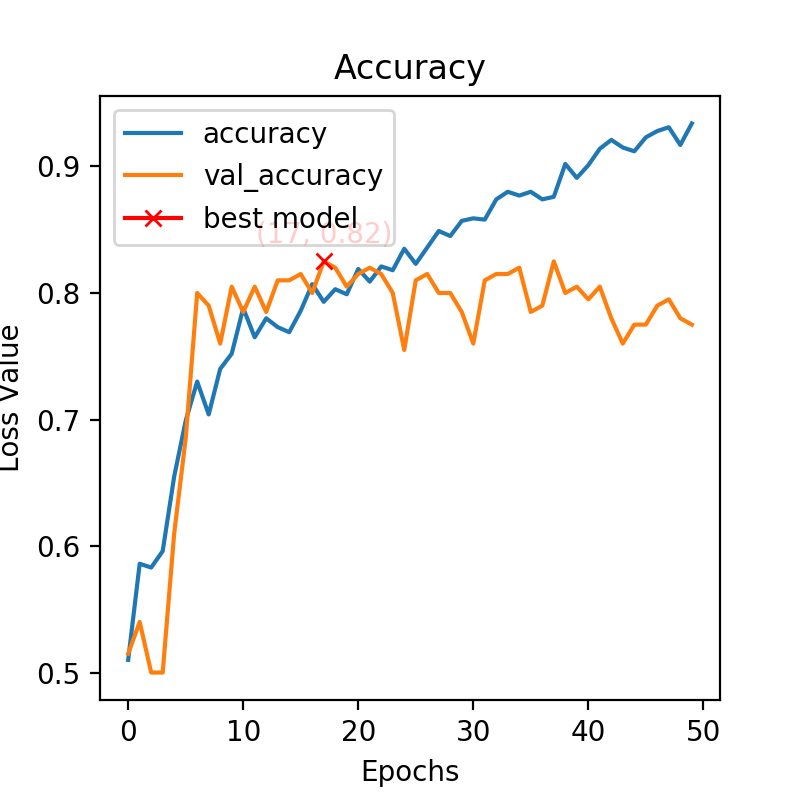
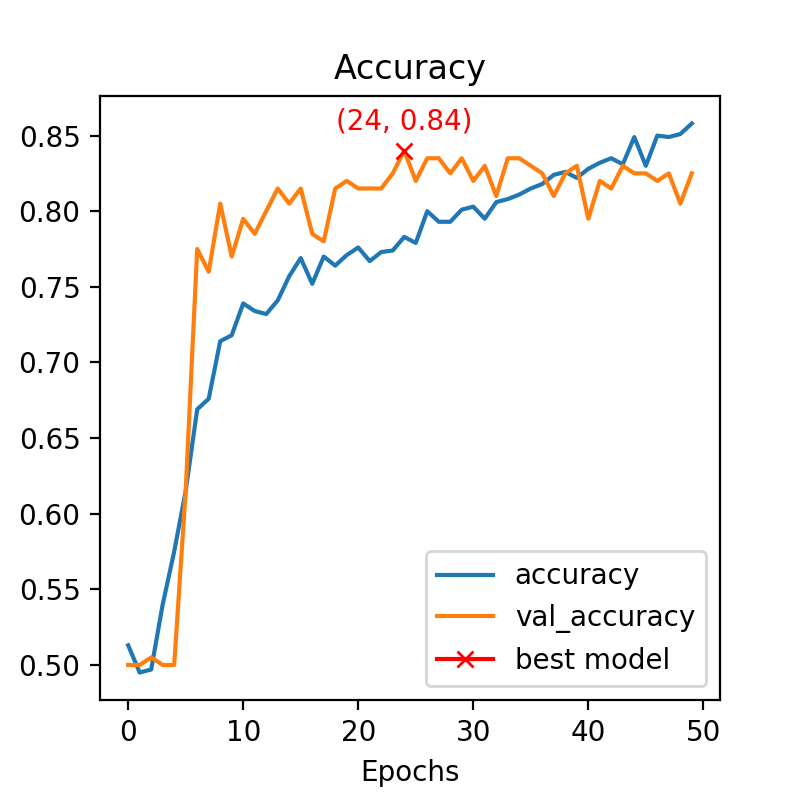
Darçot Benjamin

**Lab 3 : Report**

**Task 1A**

In this first task, the AlexNet architecture is used with a learning rate of 0.0001, a batch size of 8, a base parameter of 8 and ‘Adam’ as optimizer. By compiling and fitting this model, we find a best accuracy of approximately 0.95 for the training, and 0.82 for the validation. This means that there is an overfitting as the fitting is significantly more accurate than the validation. To counter this effect, two dropout layers are added after each dense layer in the network. Indeed, this reduces the overfitting because the accuracy of validation is about 0.84 and the one of the training is about 0.86. This reduction in the overfitting is normal as the dropout layer randomly deletes some neurons in the dense laker, thus reducing the learning power of the network which can no longer fit as well as before the training dataset.

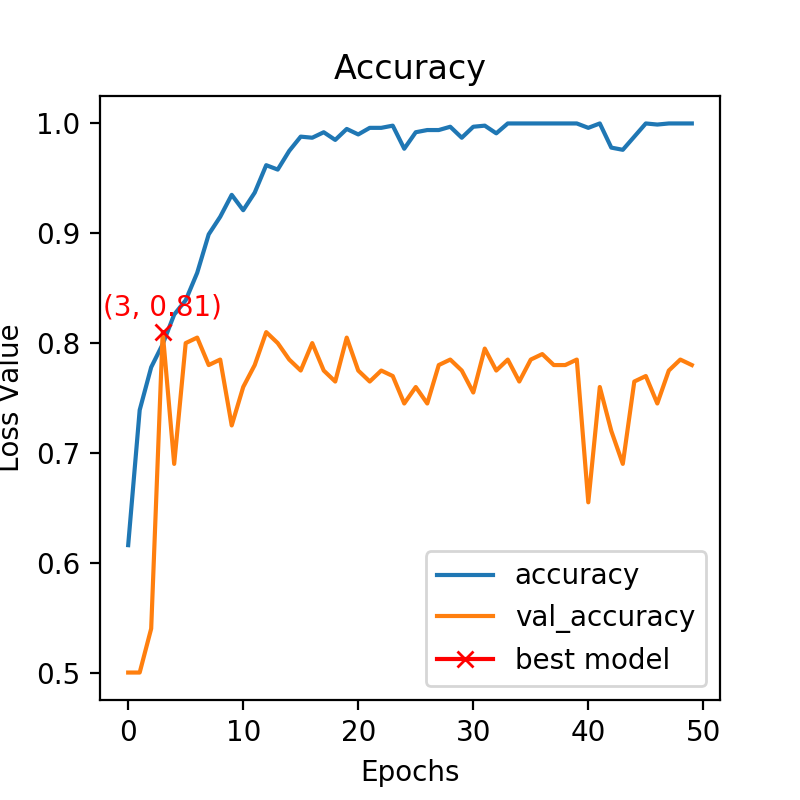
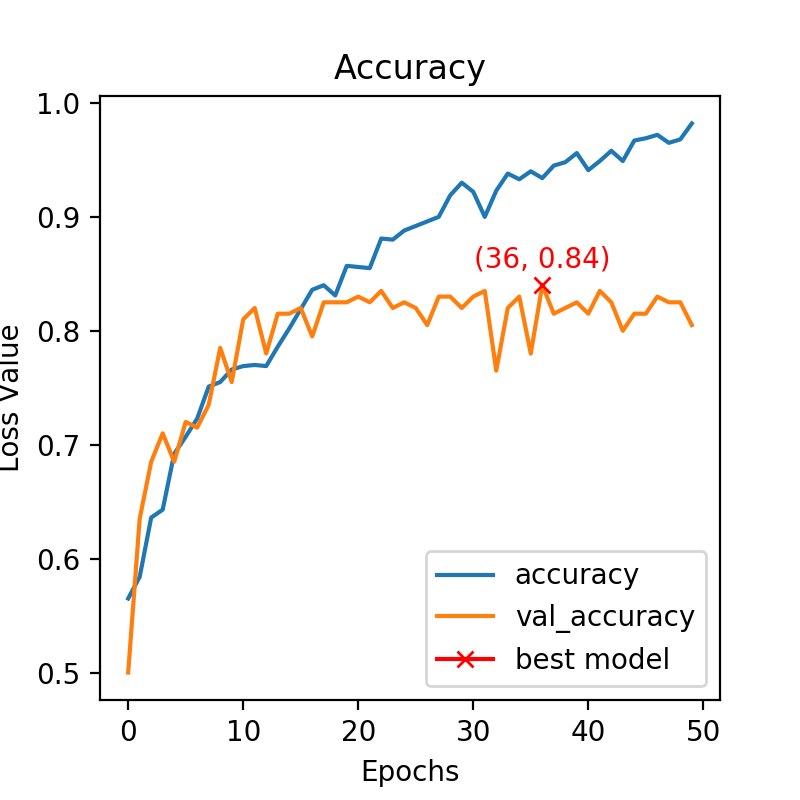
The accuracy curves are plotted below without (left) and with (right) dropout layers.



**Task 1B**

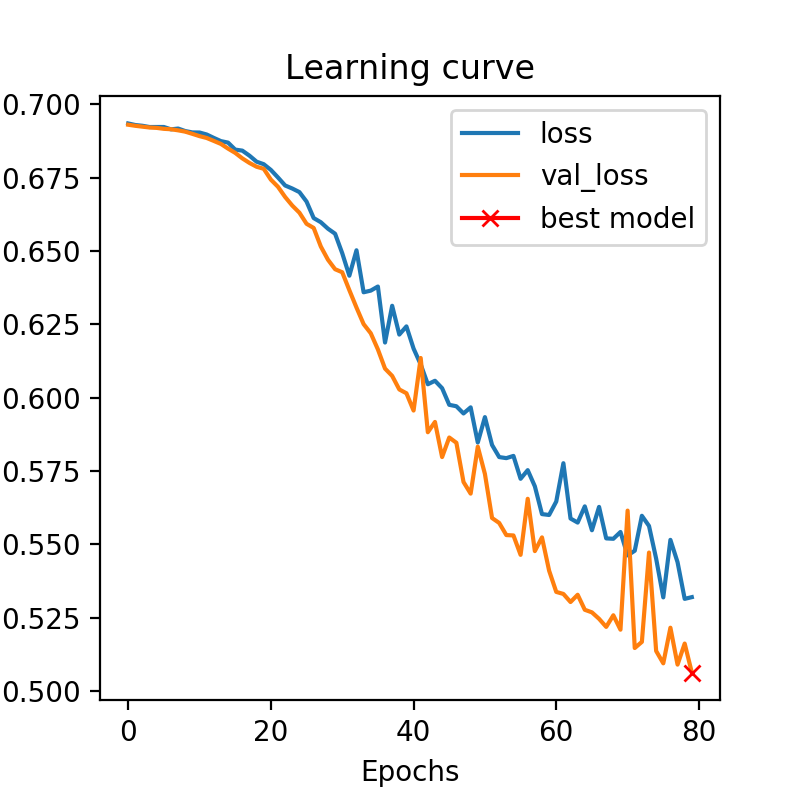
The same model is performed but a batch normalization is added after each convolutional block. Then we can observe the same results as the previous question, the dropout layers once again reduce the overfitting. But we can now note that the batch normalization seems to speed up the learning because the accuracy reaches its convergence value around the 3rd epoch while in the previous question it was around the 7th. But the value of this accuracy is the same as before, so the batch normalization didn’t improve the model (in terms of accuracy).

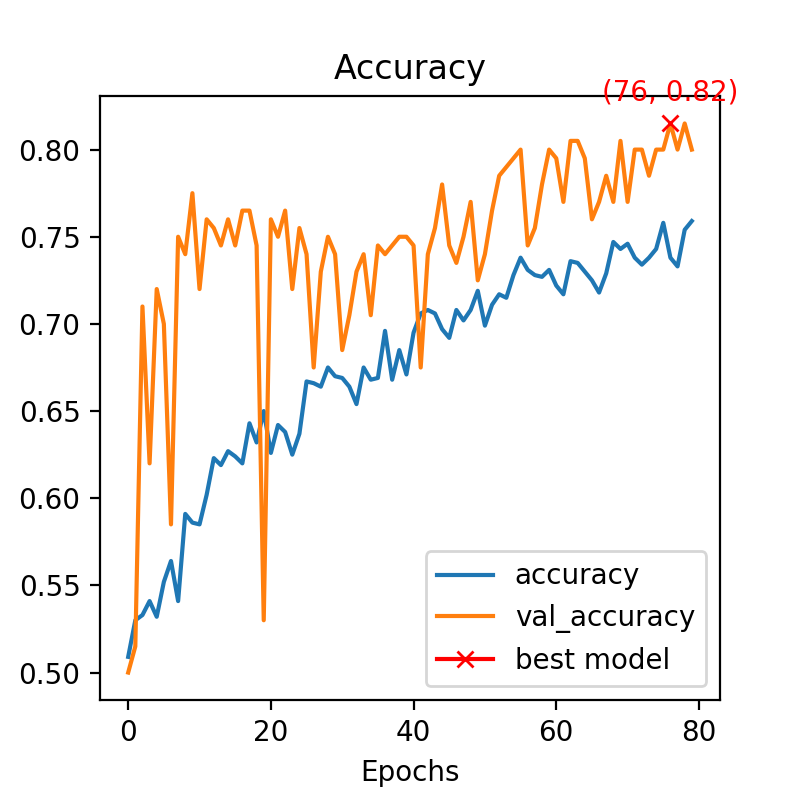
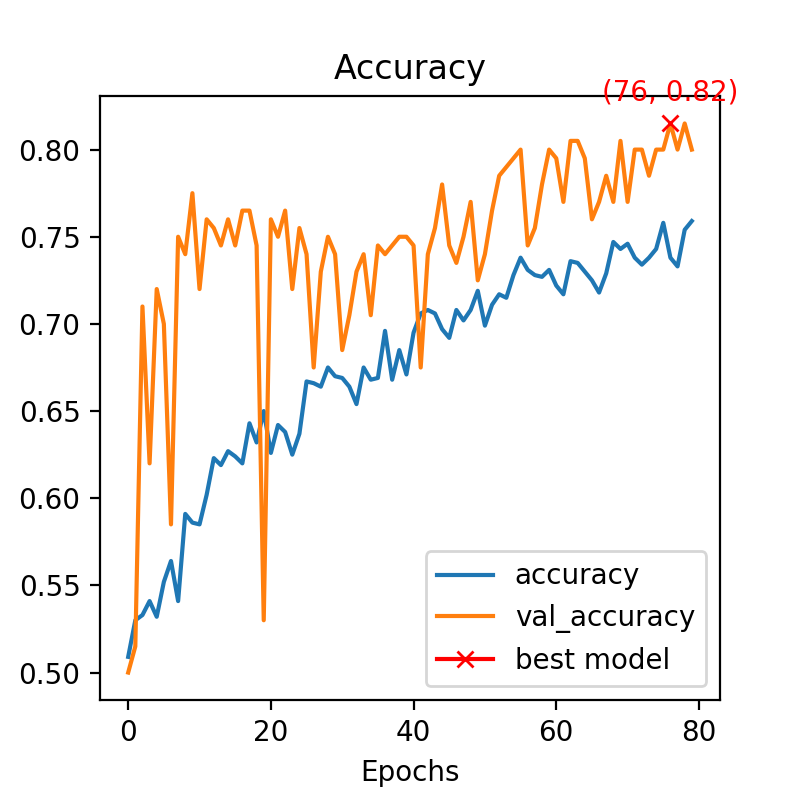
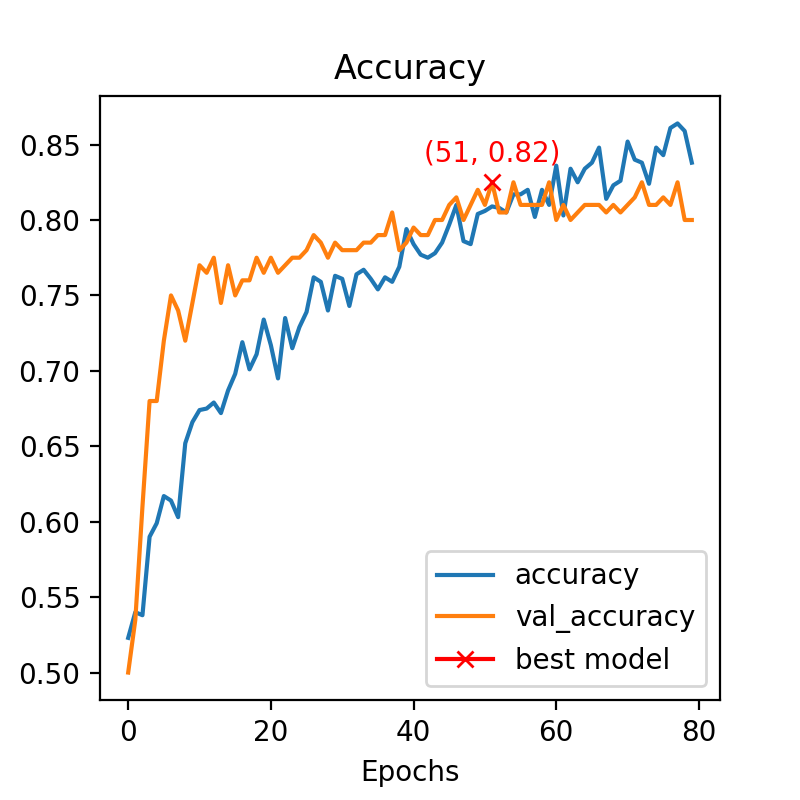
Below are the accuracy curves without (left) and with (right) dropout layers.



**Task 1C**

The same model has been used again with and without batch normalization and with a learning rate of 0.00001 and 80 epochs. By comparing the two loss curves, we can easily see the speed up effect of the batch normalization, because with batch normalization the loss has enough epochs to reach a convergence value while it is not the case without it. However, the two best validation accuracy are the same, which means that the batch normalization doesn’t improve the final accuracy, it just speeds up the learning to reach it sooner.

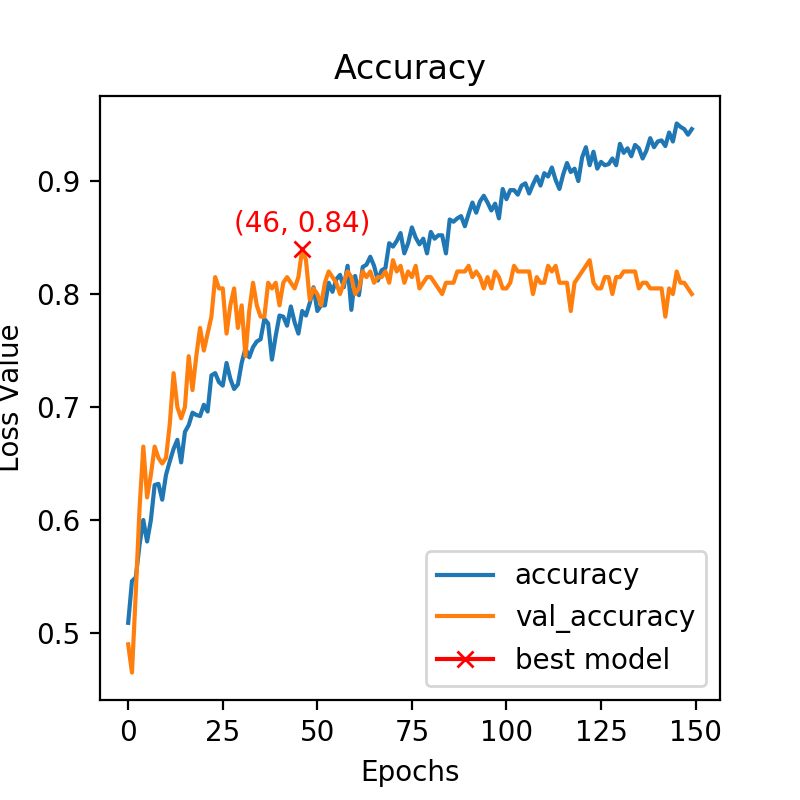
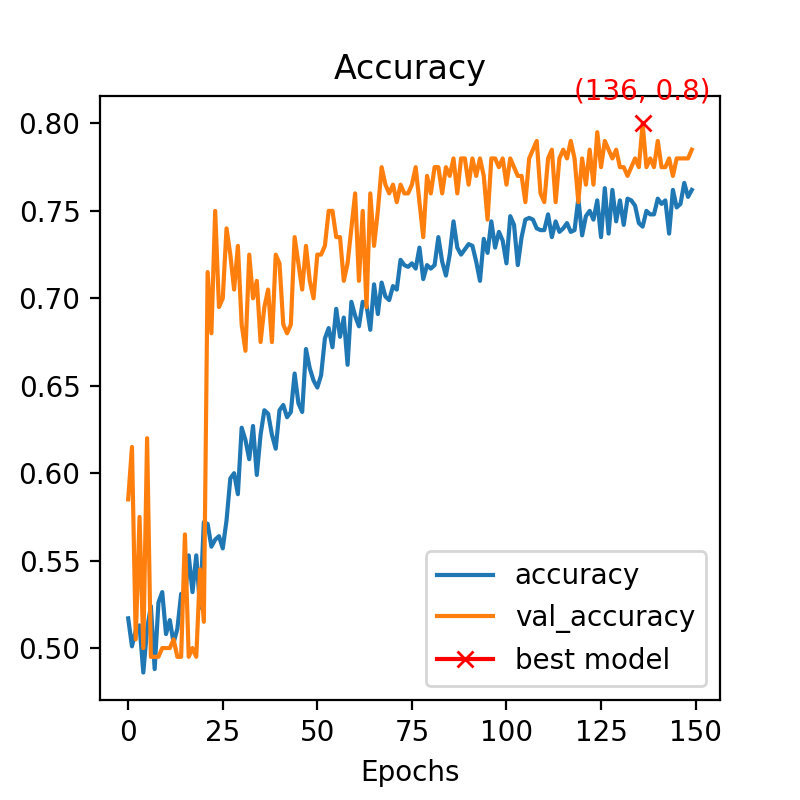
Here are the two loss curves without (left) and with (right) batch normalization.

And here are the two accuracy curves without (left) and with (right) batch normalization.

**Task 1D**

The same model as the previous question is run with and without batch normalization for 150 epochs. We can therefore now see that with the batch normalization the accuracy is slightly better (0.84 versus 0.8 without batch normalization) but it is not really significant. However without batch normalization the model has more generalization power, which is understandable because, as the batch normalization speeds up the learning, it overfits quicker.

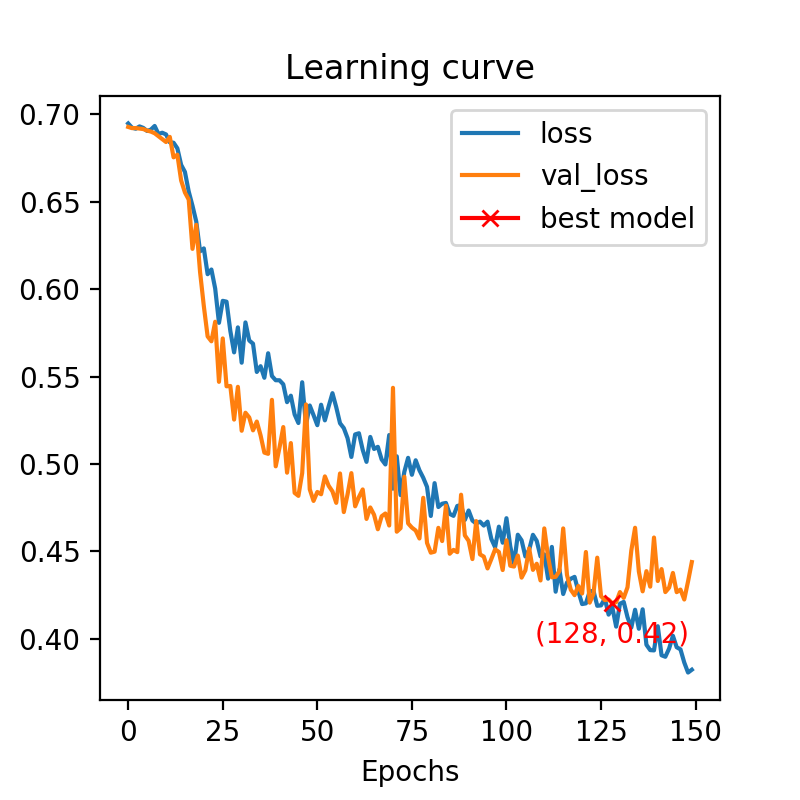
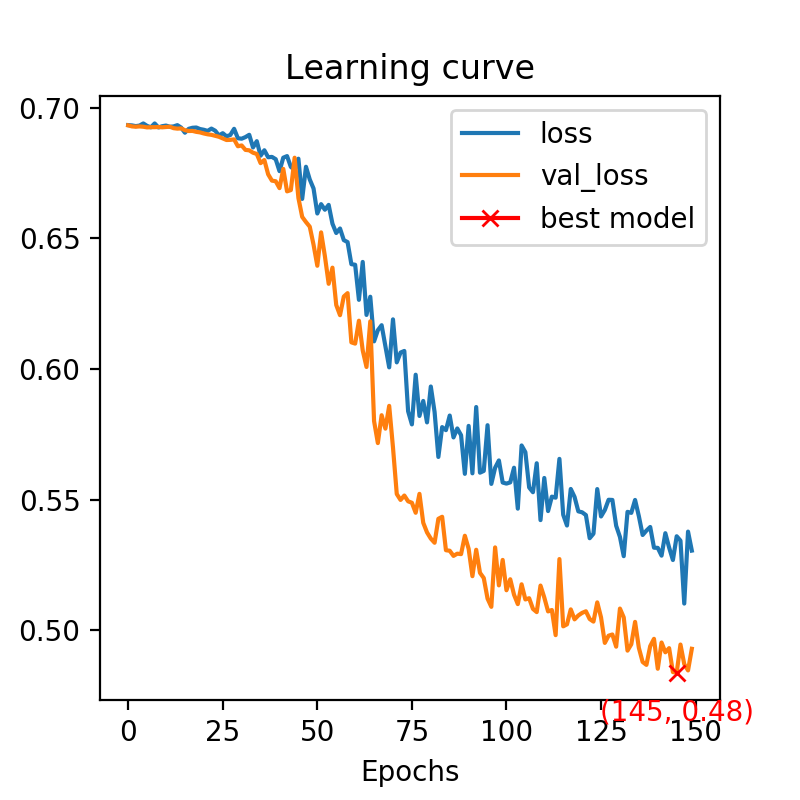
Below are the two accuracy curves without (left) and with (right) batch normalization.



**Task 2A**

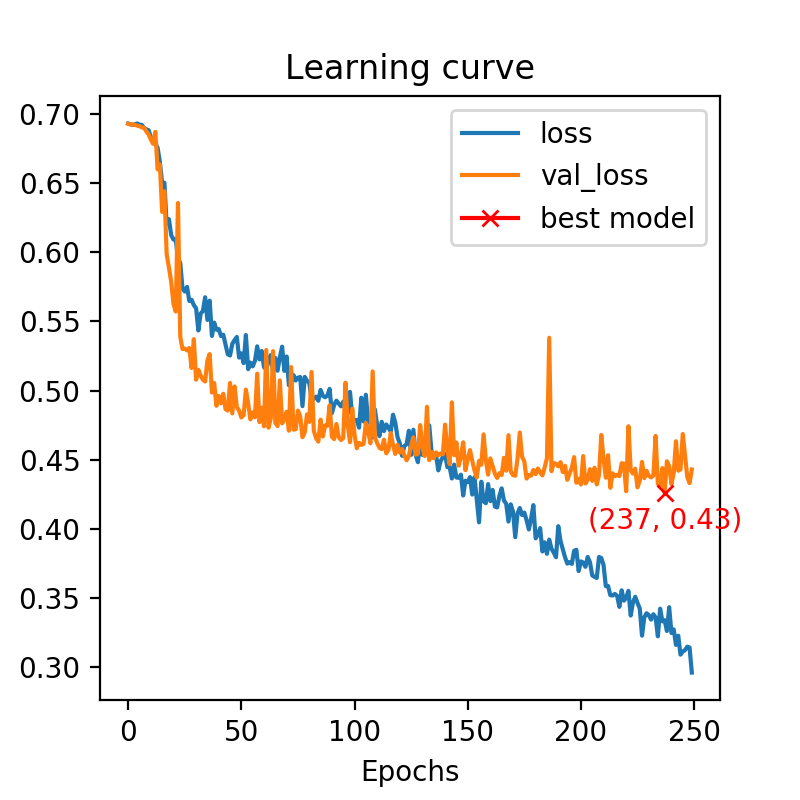
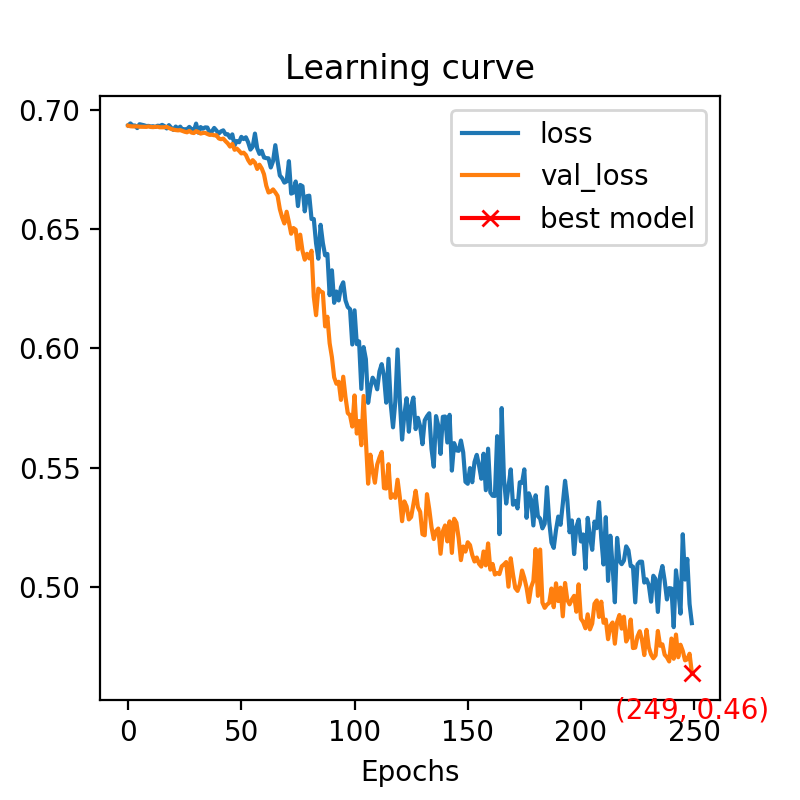
The AlexNet architecture is once again used but instead of using a batch normalization, spatial dropout layers are used at each convolutional block. By comparing the loss curves with and without the spatial dropouts, we can see that the spatial drop induces a higher generalization power. We can also note that the one without spatial dropouts converges faster, which seems normal because it has more learning power.

Below are the loss curves without (left) and with (right) spatial dropouts.



**Task 2B**

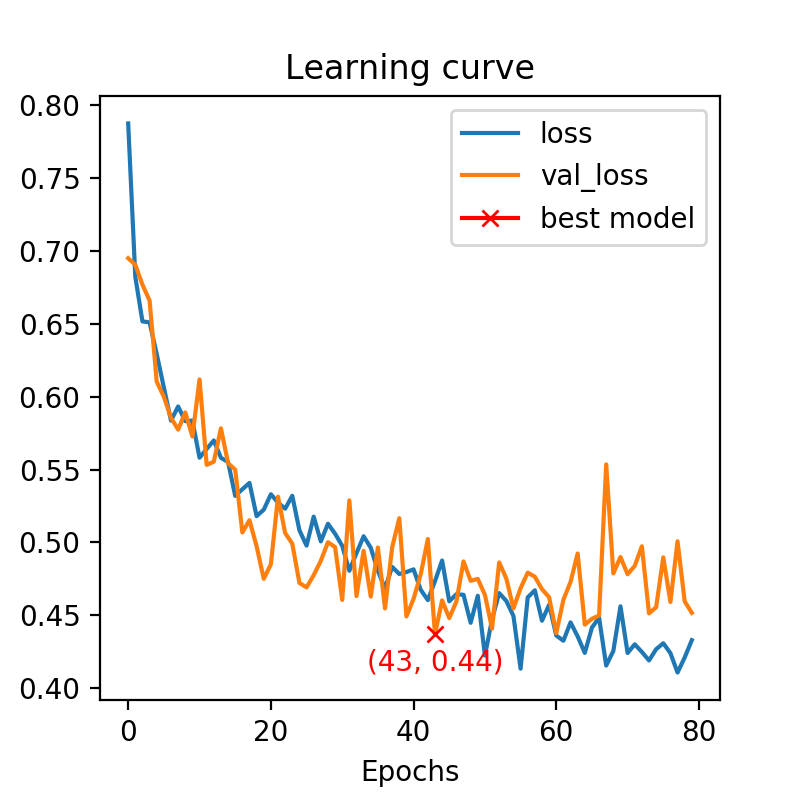
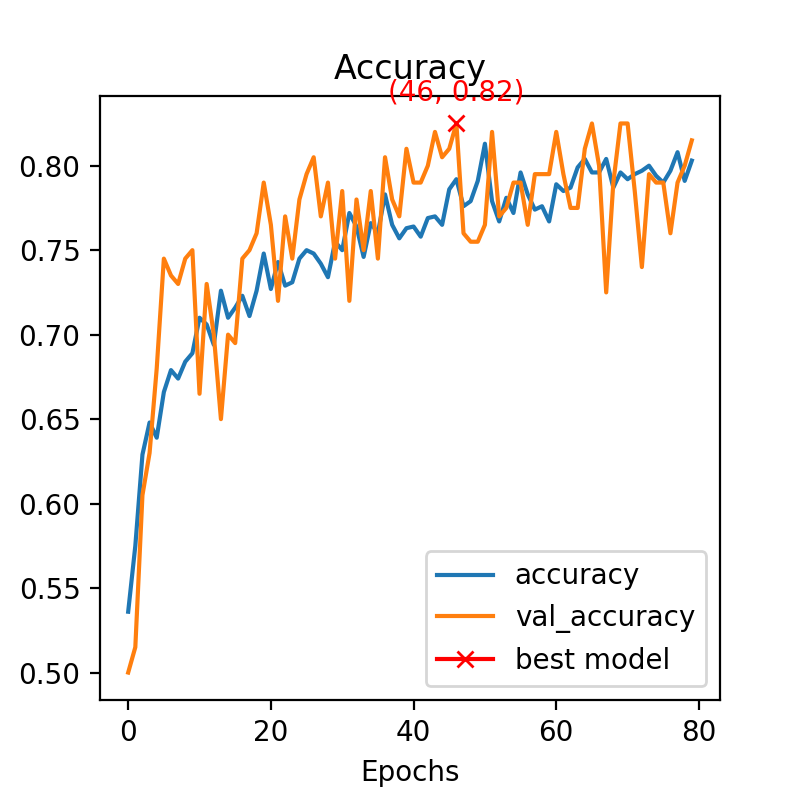
The same results are observed if we do it with 250 epochs. Below are presented the loss curves without (left) and with (right) spatial dropouts.



So, dropout techniques could help a learning procedure in case of overfitting. If the fitting is too good on the training dataset, dropouts may solve this issue.

**Task 4**

The AlexNet architecture is used again but with a base parameter of 64, a learning rate of 0.00001 and a batch-size of 8. This model is used with data augmentation and the learning curves of loss (left) and accuracy (right) are presented below.

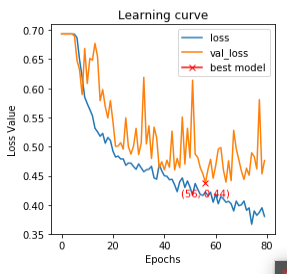
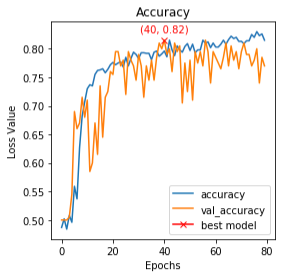


Data augmentation is a way to increase the amount of data by adding a modified image (in our case) into the dataset. The main impact of data augmentation is a reduction of the overfitting.

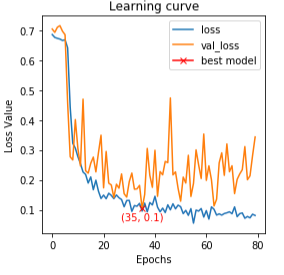
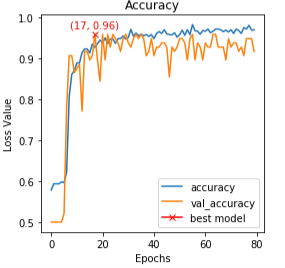
**Task 5**

In this task, we trained the VGG model in task6 for both of skin and bone data set.

Below is the results for skin data set, the accuaray on validation group of the best model is 0.82.

Below is the results for bone data set, the accuaray on validation group of the best model is 0.96.

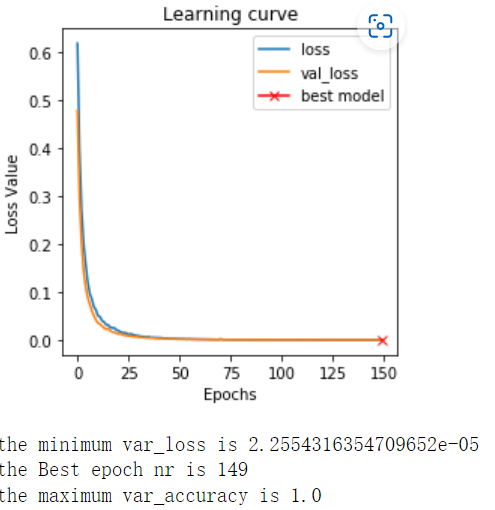
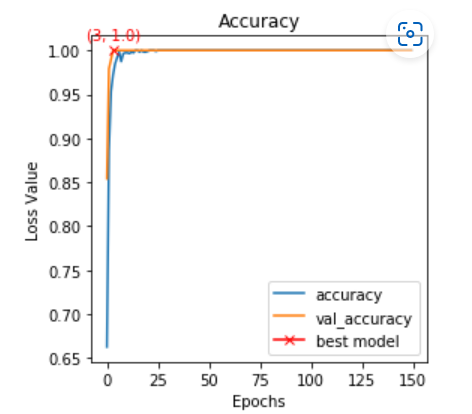
 

**Task 6**

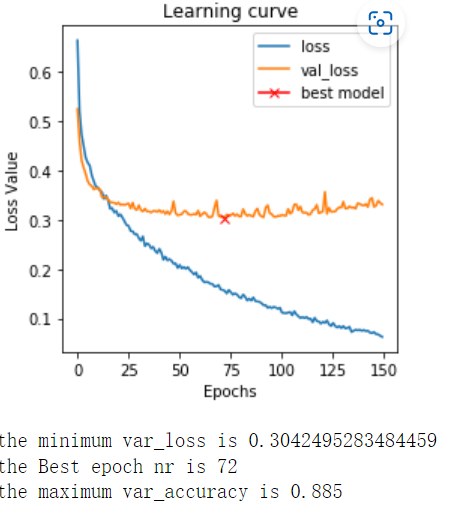
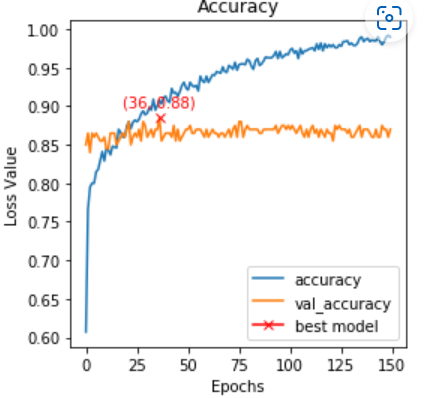
In this task, we learnt the idea of transfer learning. And we tried two methods. The first one is to creat one model combined with the downloaded VGG model and the wroten MLP model. And keep the parameters in VGG model freeze. The second one is to create two training model. The frist one is VGG model and the second one is MLP model. The input of MLP model is the output of VGG model. Both methods are tried with the same result but in our code we finally only put the code of second method.

**Task 7**

In this task, after training the fine-tuned model in task6. Below are the results of bone data,

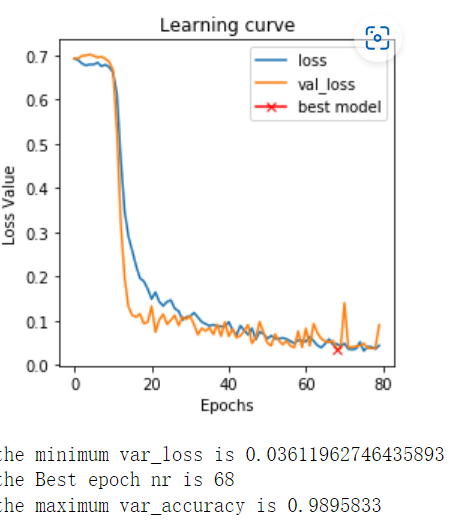
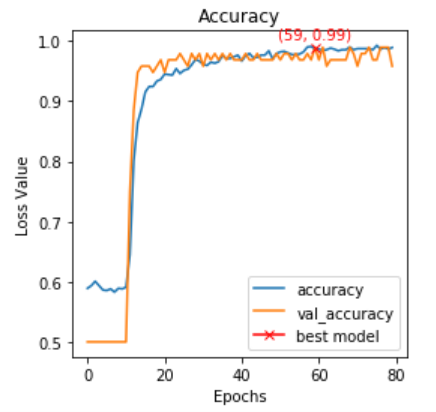
Below are the results of Skin dataset.

By compraing the observed results from transfer learning against the ones we used before, there are two main differences. The first one is the accuracy of bone and the skin dataset are much better, another one is that the training speed is much faster than before which comes from the fewer trainable parameters. We can use AUC and ROC indexes to evaluate the model so that can make sure the results are reliable.

**Task 8**

In this task, we designed the required model and used the specifc activation function and loss function. The validation accuracy is 0.99.

Class activation maps in Keras can easily let us see which regions in the image were relevant to this class.