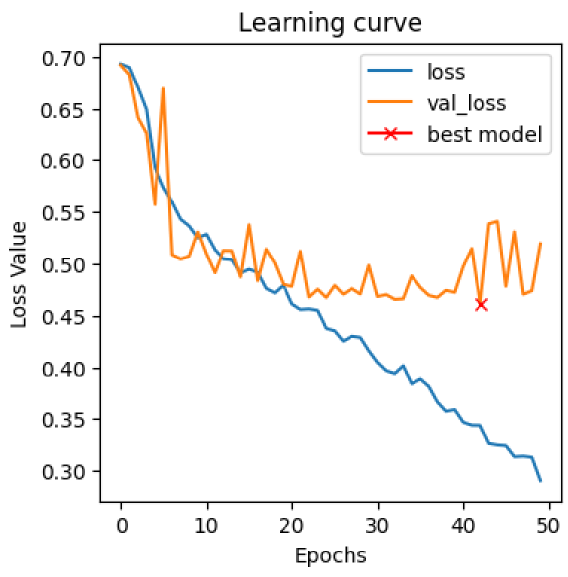
Task 1a)

Employ the AlexNet model with the following architecture: five convolutional blocks [with feature maps of the size of base, base*2, base*4, base*4, base*2 where base=8], followed by three dense layers with 64, 64, and 1 neuron, respectively. Add three max-pooling layers after 1st, 2nd, and 5th convolutional blocks. Set the following parameters: learning rate=0.0001, batch size=8, 'relu' as activation function, 'Adam' as optimizer, and image size=(128,128,1). Train this model for 50 epochs on skin images. **What are the values of the train and validation accuracy? How do you interpret the learning curves?**

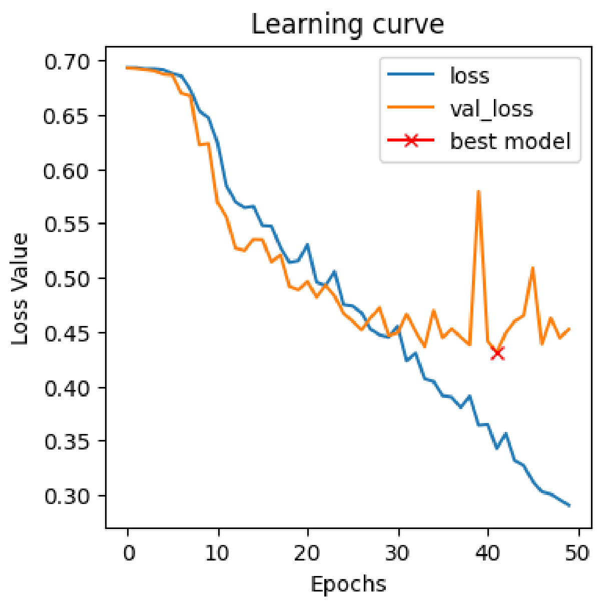
Add two drop out layers after the first two dense layers with the dropout rate of 0.4 and repeat the experiments and compare the results. What is the effect of adding drop out layers?

Answer:
See below for the train and validation accuracy, train accuracy named 'accuracy' and validation accuracy 'val_accuracy'. The accuracy of training and validation after 50 epochs are similar for with or without dropout layers. If we compare the graphs however, we can see that when we do not have dropout layers in the first figure, the two curves diverge sooner. Dropout layers delay the epoch for the divergence, which reduces the risk for overfitting in this case.

Without dropout layers
learning rate=0.0001, batch size=8, 'relu' as activation function, 'Adam' as optimizer
n_base = 8, epoch = 50
- loss: 0.2906 - binary_accuracy: 0.8860 - val_loss: 0.5191 - val_binary_accuracy: 0.7850



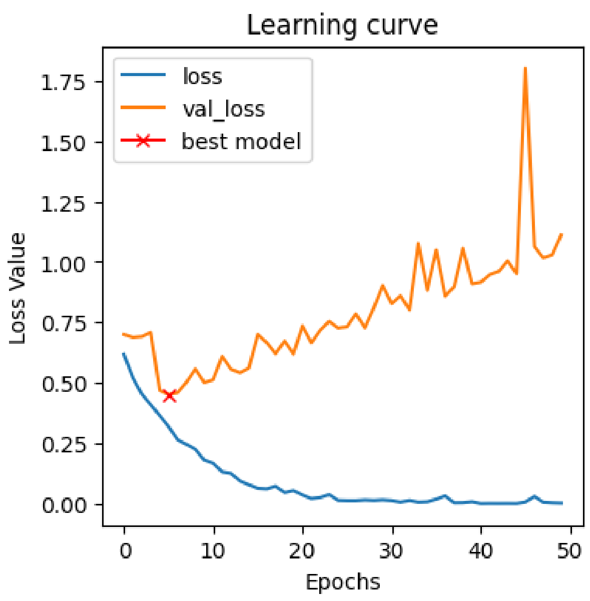
With 2 dropout layers
learning rate=0.0001, batch size=8, 'relu' as activation function, 'Adam' as optimizer
n_base = 8, epoch = 50
- loss: 0.2903 - binary_accuracy: 0.8830 - val_loss: 0.4524 - val_binary_accuracy: 0.8300



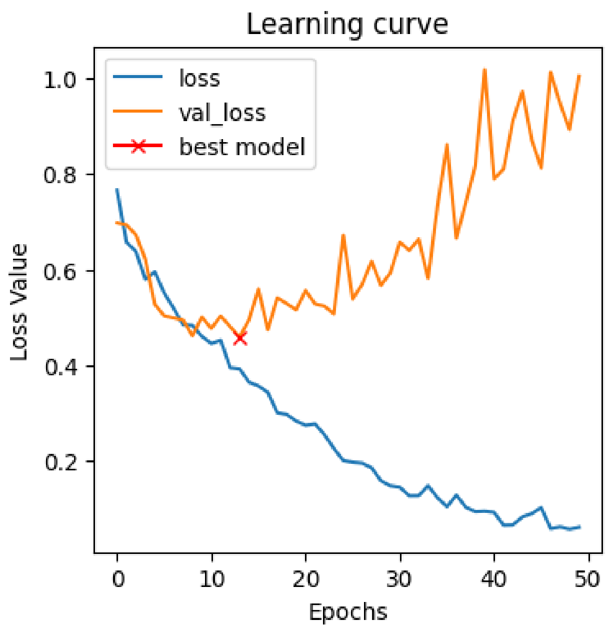
Task1b) With the same model and same settings, now insert a batch normalization layer at each convolutional block (right after convolution layer and before activation function). At which epoch do you observe the same training accuracy as task1a? **What is the value of final training accuracy? What is the effect of the batch normalization layer?** Similar to task1a, do this task with and without drop out layers.

Answer:
The final training accuracy is very high, despite only having 50 epochs. This is because of the batch normalization layers speeding up training. With dropout layers we see a slower convergence for training.

With batch normalization, without dropout layers
learning rate=0.0001, batch size=8, 'relu' as activation function, 'Adam' as optimizer
n_base = 8, epoch = 50
- loss: 0.0037 - binary_accuracy: 1.0000 - val_loss: 1.1110 - val_binary_accuracy: 0.7950



With batch normalization, with dropout layers
learning rate=0.0001, batch size=8, ‘relu’ as activation function, ‘Adam’ as optimizer
n_base = 8, epoch = 50
- loss: 0.0606 - binary_accuracy: 0.9780 - val_loss: 1.0045 - val_binary_accuracy: 0.7900

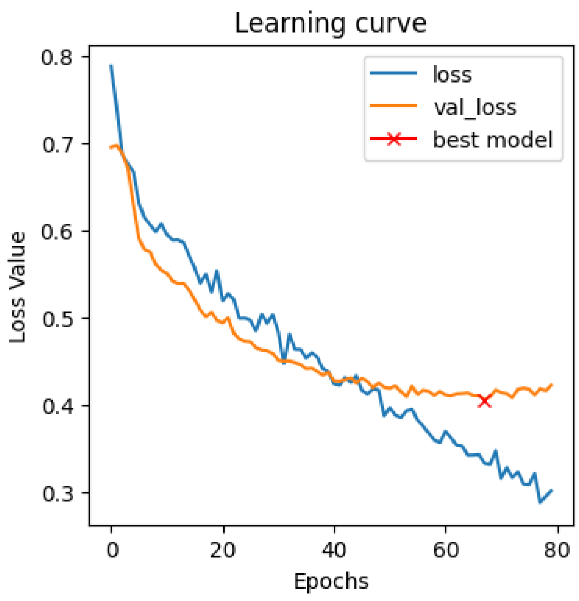


Task1c) Train again the same model with precisely the same parameters except learning rate = 0.00001 and epochs = 80 with and without batch normalization layers (in both cases, use the drop out layers). Focus on validation loss & accuracy. **Which model resulted in higher validation accuracy? How do you explain the effect of batch normalization?**

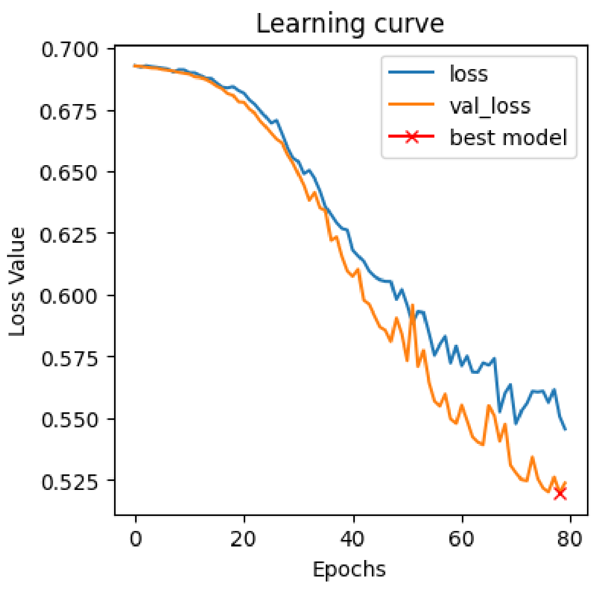
Answer:

With BN layers gave us a higher validation accuracy. BN layers speed up the convergence of the validation (and training) accuracy, giving us more reliable results. However, 80 epochs is still too few to make any conclusions.

With batch normalization layers, with dropout layers
- loss: 0.3016 - binary_accuracy: 0.8730 - val_loss: 0.4225 - val_binary_accuracy: 0.8300



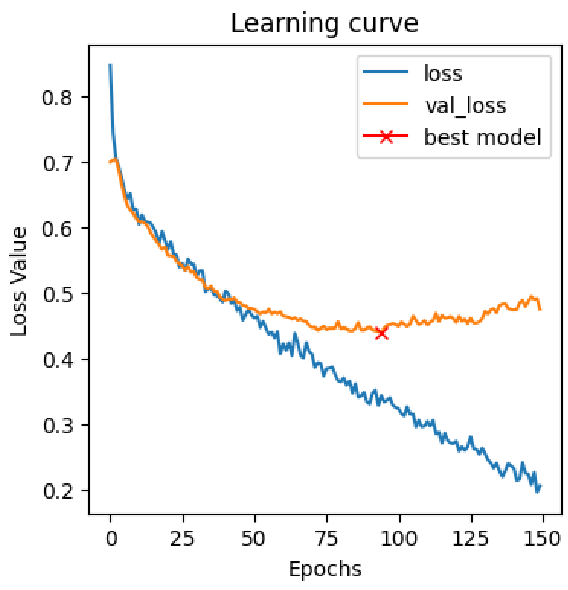
Without batch normalization layers, with dropout layers
- loss: 0.5456 - binary_accuracy: 0.7450 - val_loss: 0.5238 - val_binary_accuracy: 0.7950



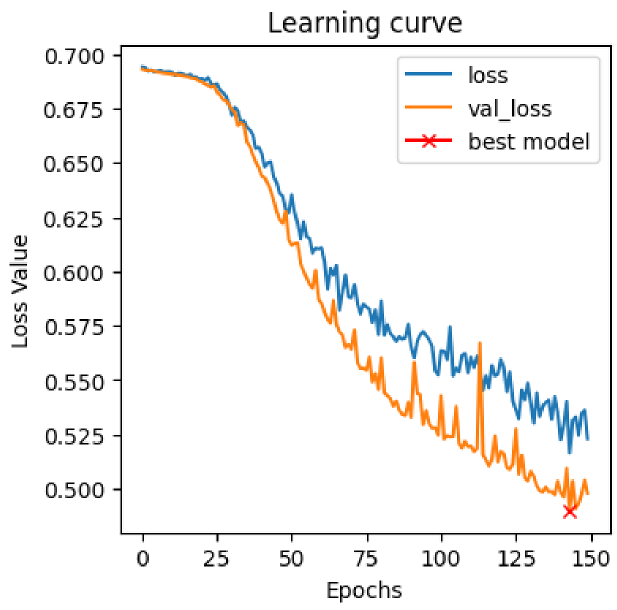
Task1d) Keep the settings from the task1c unchanged and train the model for 150 epochs with and without batch normalization layers (in bose cases, use the drop out layers). **Which model yields more accurate results? Which of them has more generalization power?**

Answer:
We get a slightly higher validation accuracy with BN layers, however we reach our lowest validation loss value only around 90 epochs. With BN layers we also have overfitting, and the validation loss value is increasing after epoch 90. Comparing this to when we do not have BN layers, the graph seems to be converging and our best validation loss value is close to epoch 150. We would argue that we have better generalization power without BN layers since the model seems to be more stable and converging.

With batch normalization layers, with dropout layers
- loss: 0.2049 - binary_accuracy: 0.9200 - val_loss: 0.4742 - val_binary_accuracy: 0.8100



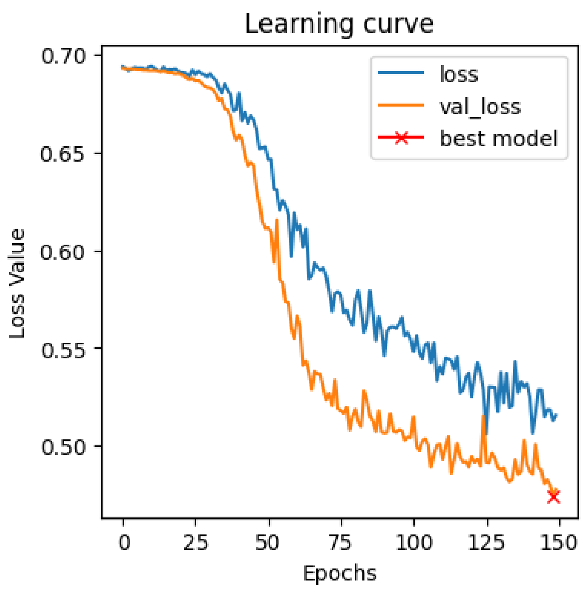
Without batch normalization layers, with dropout layers
- loss: 0.5229 - binary_accuracy: 0.7610 - val_loss: 0.4979 - val_binary_accuracy: 0.7850



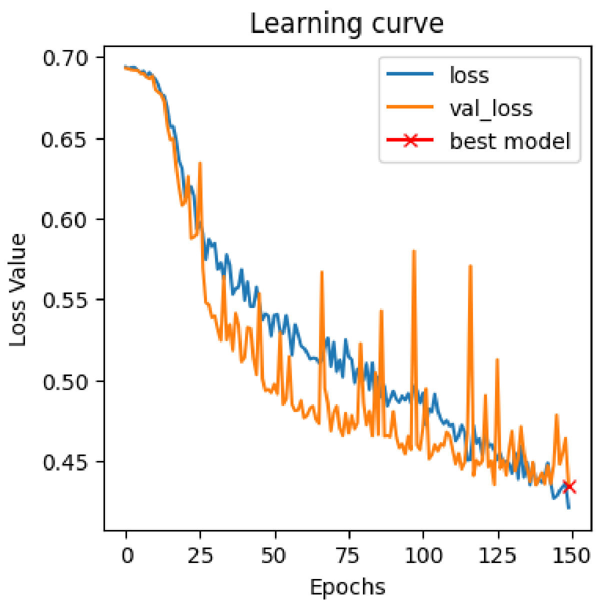
Task2a) Use the same model as task1d but set the “base” parameter as 32 and replace the batch normalization layers with spatial dropout layers at each convolutional blocks (after activation function, and before max-pooling). Set the dropout rate of spatial drop out layers as 0.1 and the rate of 0.4 for the normal drop out layers after the first two fully connected layers. Then let the model runs for 150 epochs with LR=0.00001. Save the loss and accuracy values for the validation data. Then, run the same model with the same settings but remove all the spatial drop out layers. **Which of them converges faster? Why?**

Answer:
Without spatial dropout layers converge faster. Adding dropout layers in general makes the model slower. Here we can see that the validation loss curve has a much steeper slope when we do not have spatial dropout layers.

With spatial dropout layers
learning rate=0.00001, batch size=8
Epoch = 150, n_base = 32
Spatial dropout = 0.1 normal dropout = 0.4
- loss: 0.5156 - binary_accuracy: 0.7710 - val_loss: 0.4776 - val_binary_accuracy: 0.8100



Without spatial dropout layers
learning rate=0.00001, batch size=8
Epoch = 150, n_base = 32
Spatial dropout = 0.1 normal dropout = 0.4
- loss: 0.4212 - binary_accuracy: 0.8180 - val_loss: 0.4350 - val_binary_accuracy: 0.8300

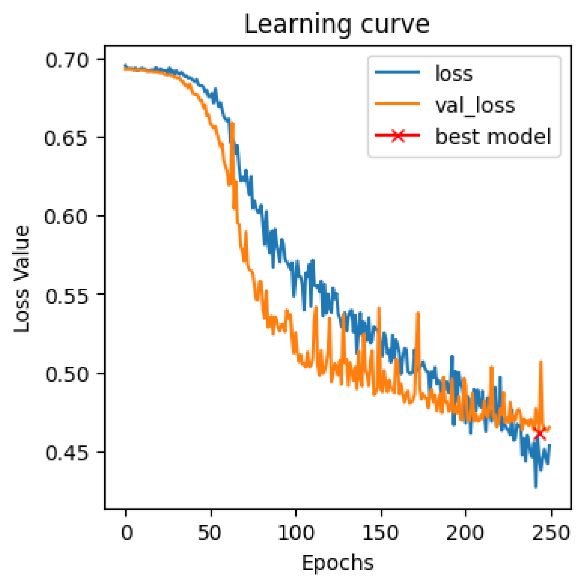


Task2b) Repeat the task2a for 250 epochs with and without spatial dropout layers. In general, **discuss how the drop out technique (spatial and normal one) would help the learning procedure.**

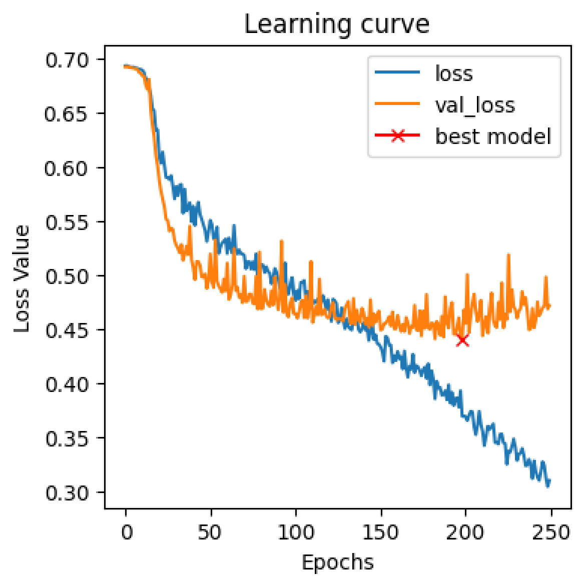
Answer:
We can use dropout layers to slow down the model and see if a slower model will reach a higher validation accuracy. It is also good to visualize and experiment with the convergence of the validation curve. After 150 epochs the effect of the spatial dropout layers was not as clear, but here for 250 epochs we can clearly see how when we remove the spatial dropout layers the validation curve converges and stabilizes. Adding spatial dropout layers here also delays the divergence of the two curves and lessens the risk of overfitting.

With spatial dropout layers
learning rate=0.00001, batch size=8
Epoch = 250, n_base = 32
Spatial dropout = 0.1 normal dropout = 0.4

```
- loss: 0.4537 - binary_accuracy: 0.8060 - val_loss: 0.4649 - val_binary_accuracy: 0.8000
```

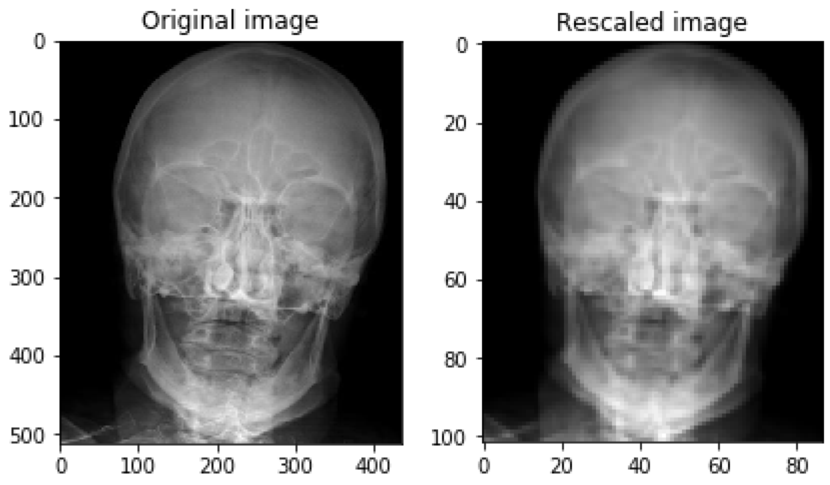


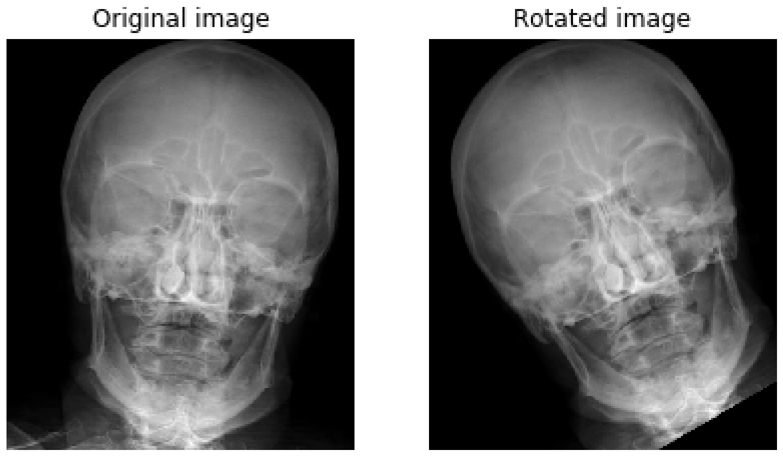
Without spatial dropout layers
learning rate=0.00001, batch size=8
Epoch = 250, n_base = 32
Spatial dropout = 0.1 normal dropout = 0.4
- loss: 0.3102 - binary_accuracy: 0.8670 - val_loss: 0.4718 - val_binary_accuracy: 0.8150

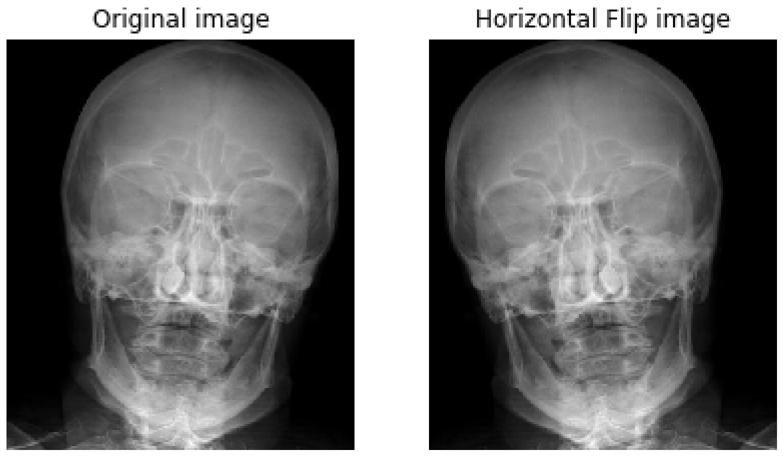


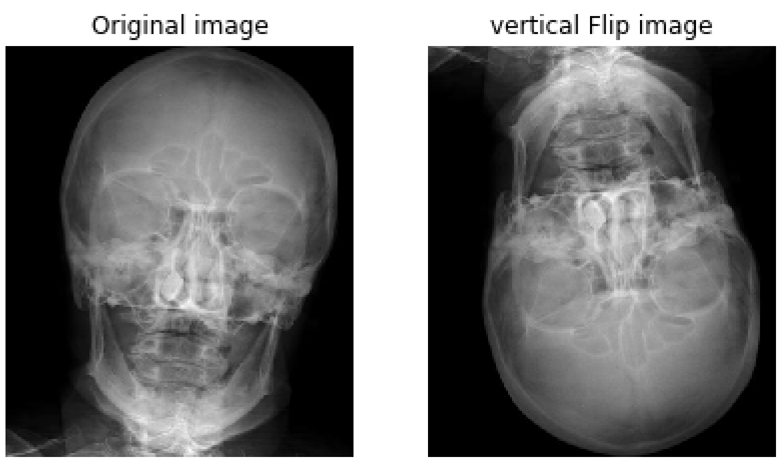
Task3a) Read the following set of codes and find out how they work. Then, change the following parameters: scale_factor, Angle, low/high bounds to see their effects.

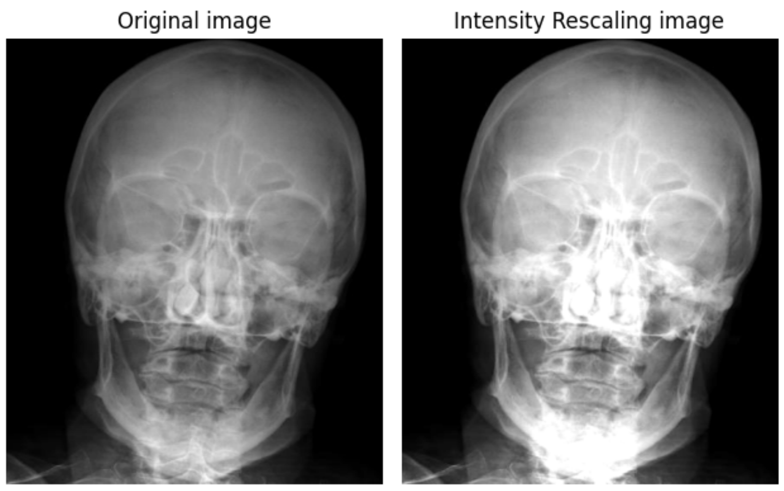
scale_factor = 0.2, angle = 30, horizontal_flip = img[:, ::-1]
vertical_flip = img[::-1, :], low_bound, high_boud = 5, 95


Original image
Rescaled image
0
100
200
300
400
500
0
100
200
300
400
0
20
40
60
80
100
0
20
40
60
80


Original image
Rotated image

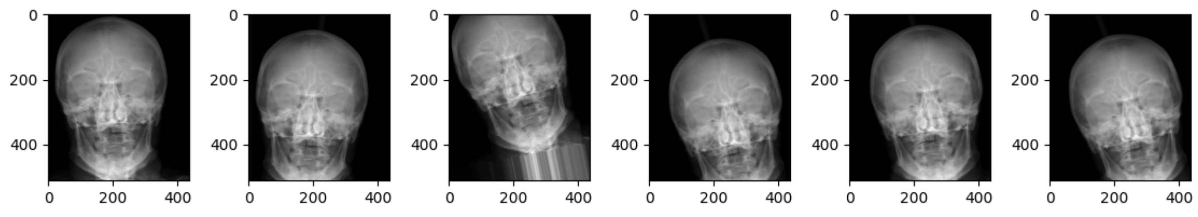

Original image
Horizontal Flip image





(In the code file Task3-4-5.ipybn)

Task3b) A practical way to perform the data augmentation technique is to develop a generator.
The following code is an example of how you can generate augmented images randomly with a TensorFlow built-in generator.



(In the code file Task3-4-5.ipybn)

Task4) Develop a framework for training the models with augmenting the training data as:

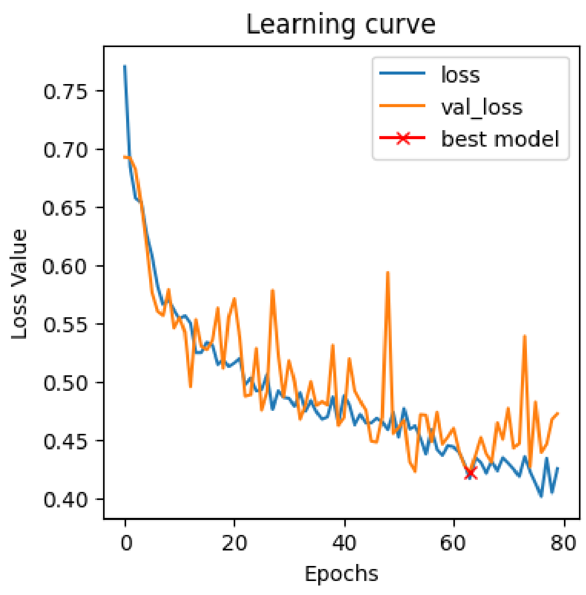
(In the code file Task3-4-5.ipybn)

How does the data augmentation impact model training? Why?

Data augmentation forces the model to learn quicker. After 10 epochs we are already below loss values of 0.55. If we compare this to the results from task 1c) (without BN layers) the model learns way slower with almost constant values the first 20 epochs. Adding data augmentation and with that more variation to the images forces the model to learn the features better and thus the performance will become better quicker.

Results:

AlexNet with image augmentation
n_epochs = 80, Batch_Size = 8, LR = 1e-5, n_base = 64, dropout = 0.4
- loss: 0.4256 - accuracy: 0.8000 - val_loss: 0.4729 - val_accuracy: 0.8250



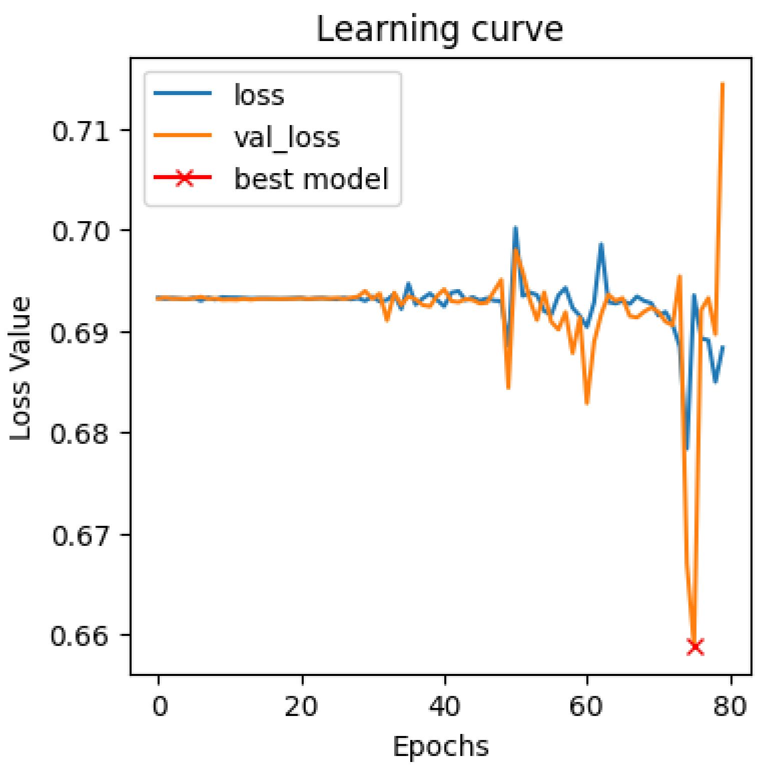
Task5) Repeat task6 for VGG model for both of skin and bone data set.

Which VGG model should be used is unclear for this task. So we used the normal VGG from Lab2.

Results of Skin:

- **Normal VGG from Lab2:**
  Base = 16, n_epochs = 80, Batch_Size = 8 ,LR = 1e-5, Adam.
  Dropout(0.2)

  - loss: 0.6883 - accuracy: 0.4917 - val_loss: 0.7143 - val_accuracy: 0.3333

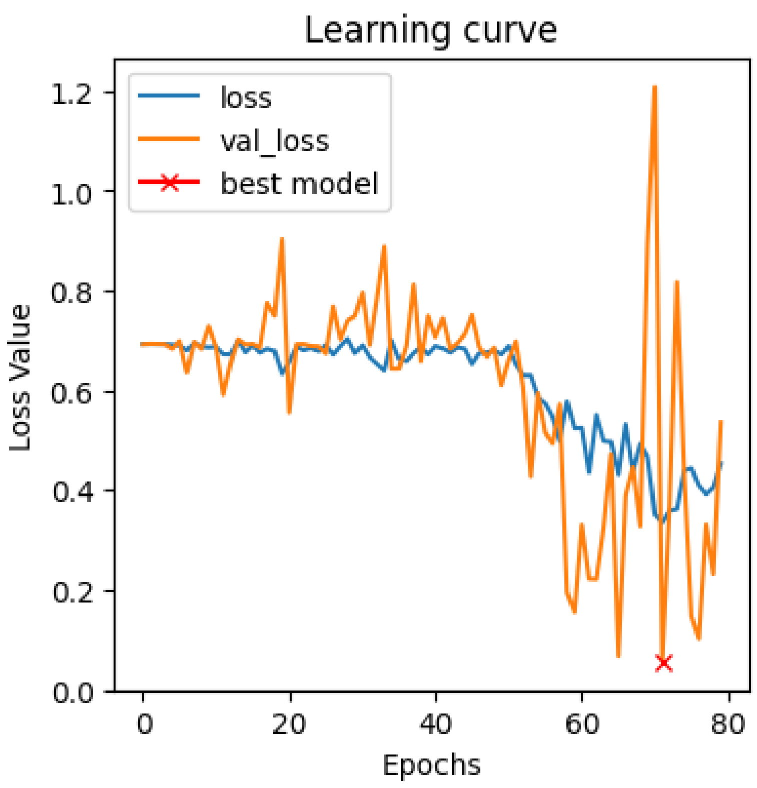


Results of Bones:

- **Normal VGG from Lab2:**
  Base = 64, n_epochs = 80, Batch_Size = 8 ,LR = 1e-5, Adam.
  Dropout(0.2)

The performance of this VGG model for bone images is not stable by now, which indicates the result is not reliable yet.

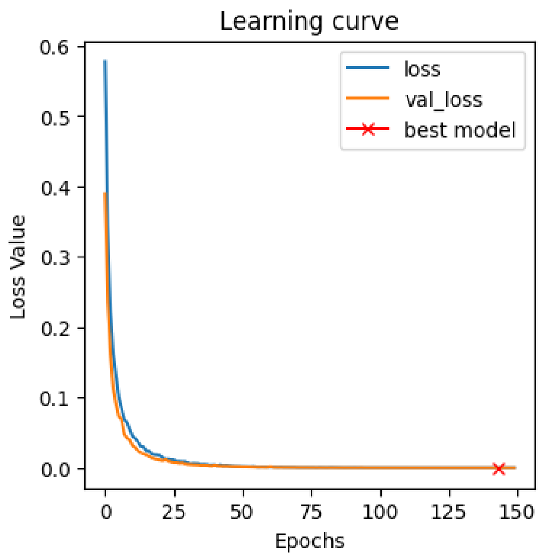
- loss: 0.4540 - accuracy: 0.8456 - val_loss: 0.5366 - val_accuracy: 0.7500

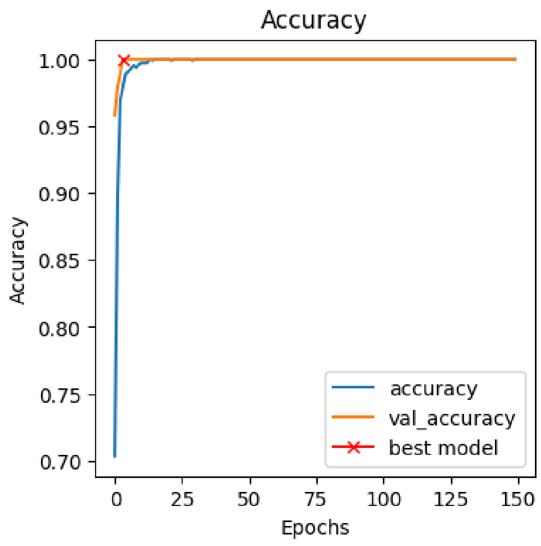


Task6 : In this task, you will employ a pre-trained VGG16 model that was trained on the ImageNet database. By fine-tuning this model, you will classify the skin and bone data set again. To do so, as the first step, the pre-trained model should be loaded. This pretrained model will be used as a feature 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.

Classification of Bones
With hyperparameters as :
epochs = 150
batch_size = 8
LR = 0.00001
optimizer=Adam
loss='binary_crossentropy'
(In the code file Task6.ipynb)

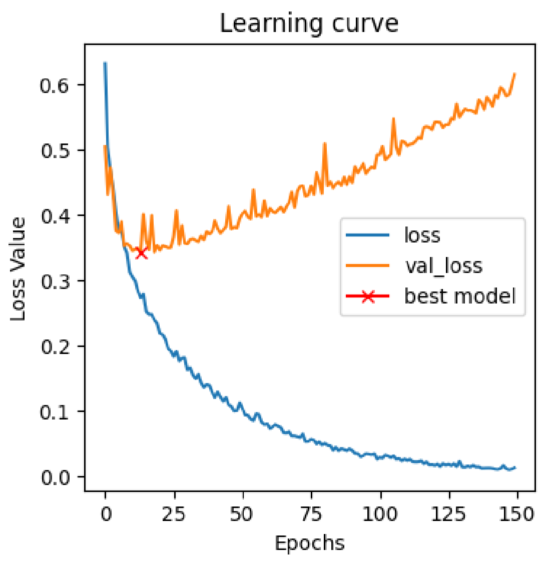
```
139/139 [==============================] - 0s 3ms/step - loss: 3.1536e-05 - accuracy: 1.0000 - val_loss: 1.9735e-05 - val_accuracy: 1.0000
```

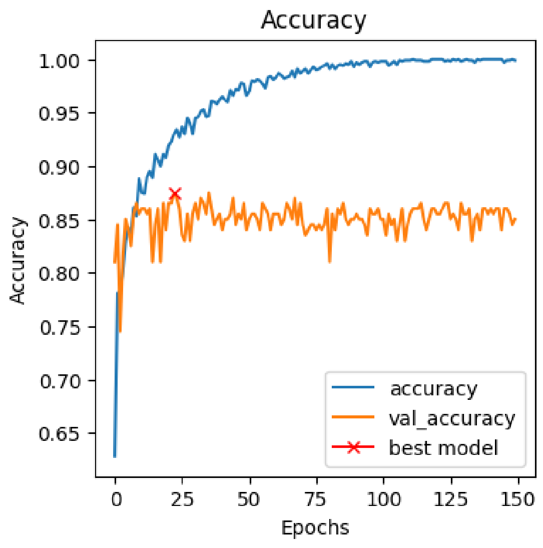




Classification of skin images
With hyperparameters as :
epochs = 150
batch_size = 8
LR = 0.00001
optimizer=Adam
loss='binary_crossentropy'

```
125/125 [==============================] - 1s 7ms/step - loss: 0.0123 - accuracy: 0.9990 -
val_loss: 0.6137 - val_accuracy: 0.8500
```





Task7) 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?

Answers:

(Results of models for bones and skins are shown in task 6)

Using the fine-tuned model, the input train_data and validation_data are the features information extracted from the pretrain model.

- We could tell that the curves are much smoother than the performance without a pretrained model. The models perform better with transfer learning.
- The convergence is much quicker with a pre-trained model.
- By comparing Adam and SGD as optimizers for models, we also found that the model with Adam converges more quickly than the other.

Using transfer learning simplifies the procedure of training. Since convolutional layers in the pre-trained model extract low-level features. In transfer learning, the pre-trained model is reused as the starting point for another model. The pre-trained model in this task is based on bone image. So the new model for bone classification performs much better using pre-trained features. For skin classification, the convergence of the model is quicker as well. But the accuracy in classifying skins is not as high as bones.
Overall, pre-trained models simplify the model for new tasks and serve for specific tasks and users.

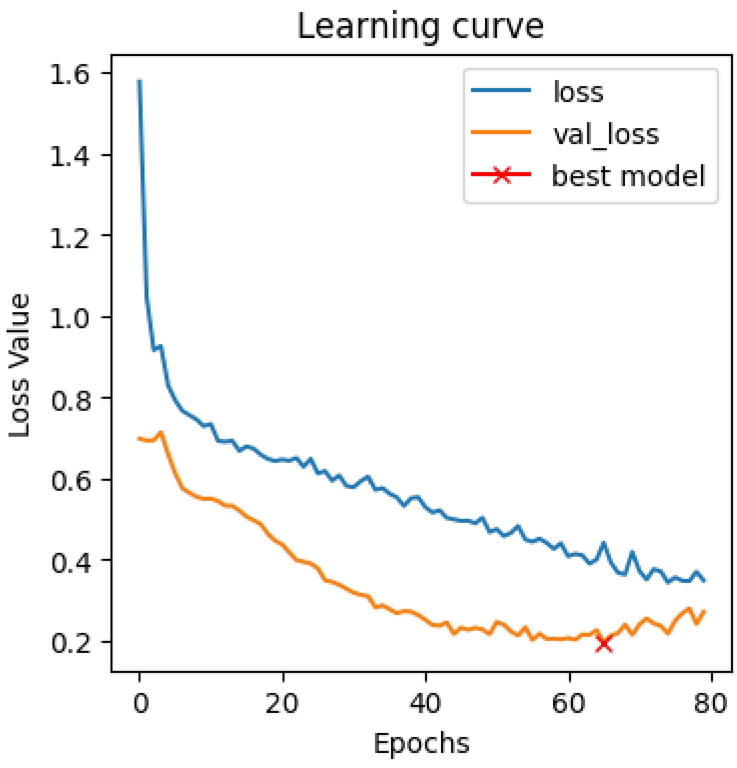
From the results below, the model with Adam as optimizer seems to converge and be stable. So the result is reliable.
While the curves from the model with SGD as optimizer are still changing, the result is not reliable yet. Training with more epochs may give us a reliable result.

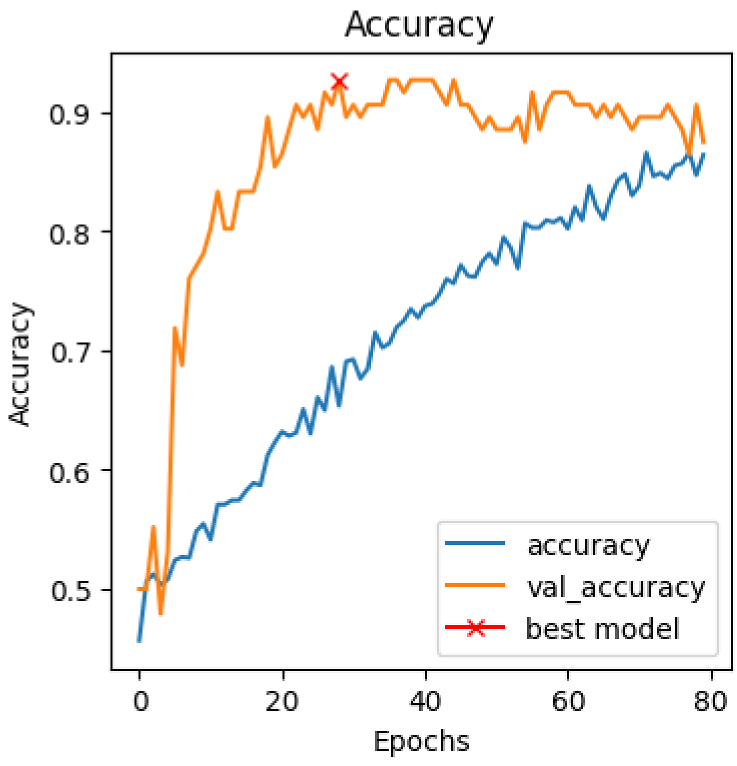
Task8) What can you infer from visualization of the class activation maps?

Answer:
The results from this task are attention maps which indicates which part of information in the images is mainly used for prediction. As we could see that the text at the corner in yellow is the area with more attention than the bones, we include that this model doesn't learn from the bones' features from the image. Instead the model extracts features from the text.

Results:
VGG model:
Base = 8, batch_s = 8; LR=1e-5; img_size = 128*128, epoch = 80

```
428ms/step - loss: 0.6497 - accuracy: 0.6736 - val_loss: 0.9618 - val_accuracy: 0.4375
```

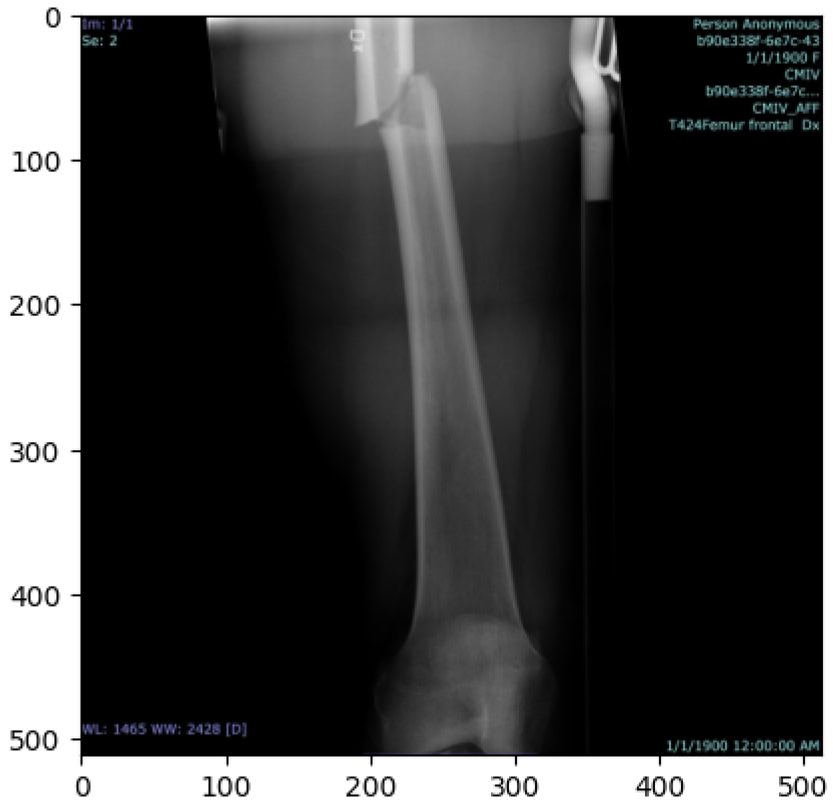


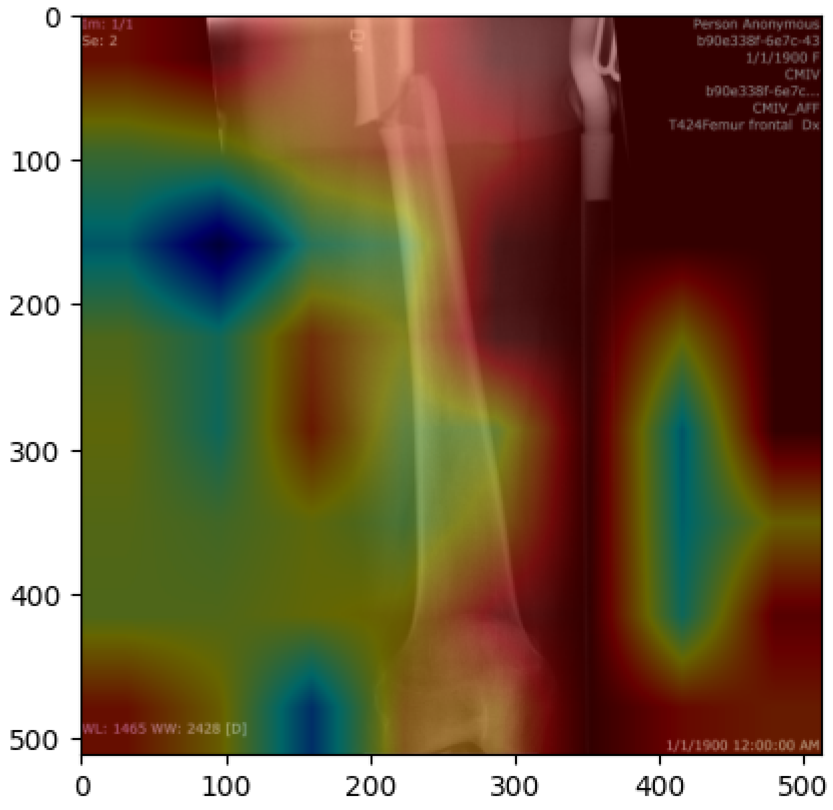


Results:

The predicted class label is 1.
The attention map shows the regions with high attention in the last convolutional layer from the model. The region of red color is of high attention while the region of green or blue is of

lower attention. On the top right corner of the image, there is text with the information of the label which is in the red region as well. The cracked bone region is yellow. It indicates that in this image, the model pays more attention to the text region and makes the prediction, which shows that the model probably gets trained with the text information.





Bonus Task)

Design a N-Layer Residual network for the classification task. Try to find the optimum hyperparameters such as batch size, depth of model, learning rate etc. Train your model with data augmentation for X-ray images. Apply the followings in your implementation:

B1) As you might notice, the number of image data in each class of X-ray images are not equal. Therefore, you need to handle this imbalanced class problem.

We implemented class weight and included it into the model. (See from codes file)

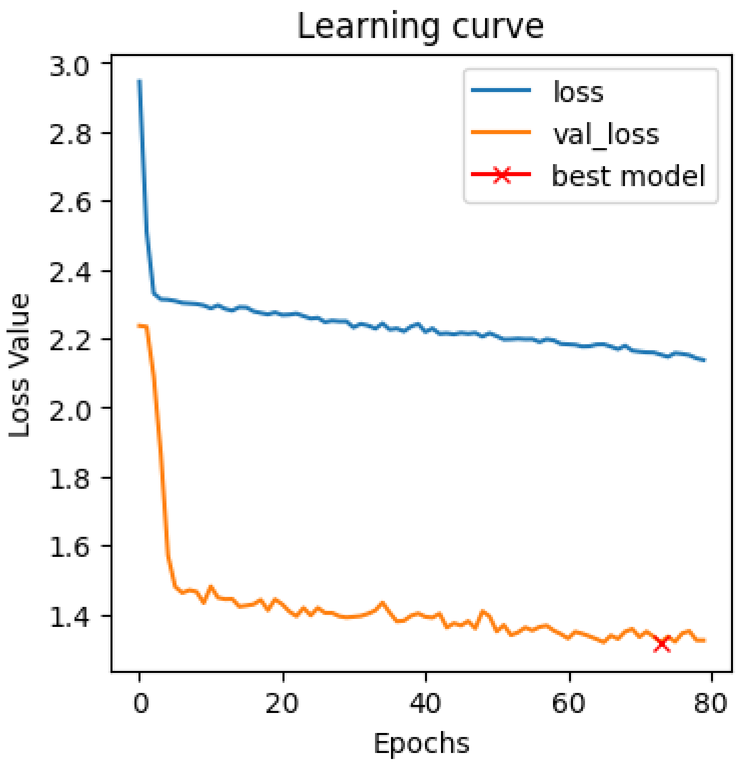
B2) In previous tasks, for simplicity, you performed all the experiments on randomly selected train and validation data. Split your data through a 3-fold cross-validation approach and report the classification accuracy of your model as mean∓variation scores over the validation sets.

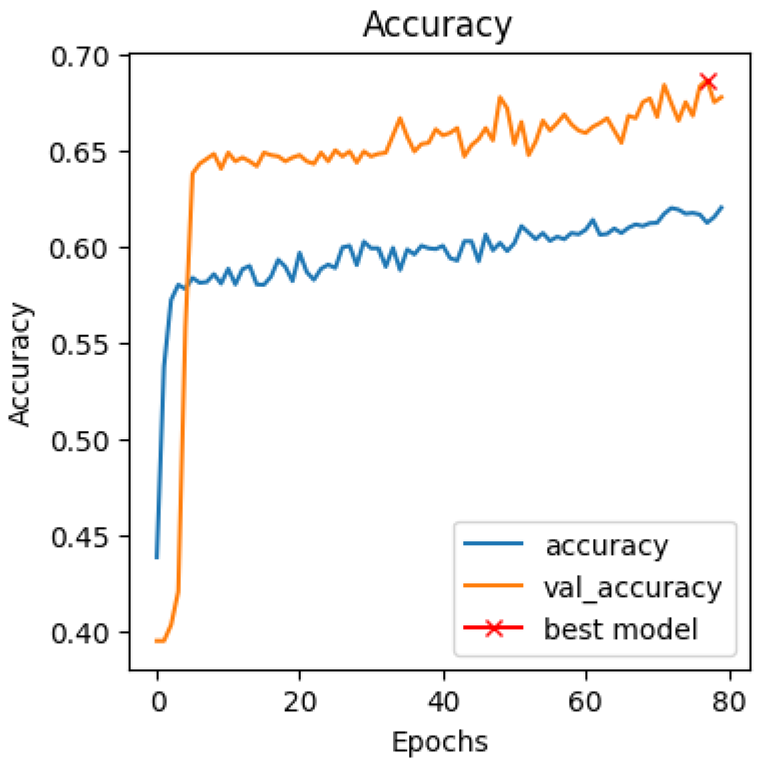
From 4 folds:
67.46% (+/- 7.88%)
(See from codes file)

B3) Optimize your implementations by adding early stopping criteria and learning rate schedule. You can also save the whole model or model weights only. This is a very useful strategy by which you can train your model for a few epochs, and later, you can load the weights and continue the training process.

We implemented a learning rate schedule method to control it during the training.
Also we added checkpoints into the model to save the model with best performance.

When batch_size = 8


Learning curve
Loss Value
Epochs
loss
val_loss
best model
3.0
2.8
2.6
2.4
2.2
2.0
1.8
1.6
1.4
0
20
40
60
80


Accuracy
Accuracy
Epochs
accuracy
val_accuracy
best model
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
20
40
60
80