CM2003 Lab2

Libo Xu and Md Rajibul Islam

Task 1a: The final train accuracy is 88 %

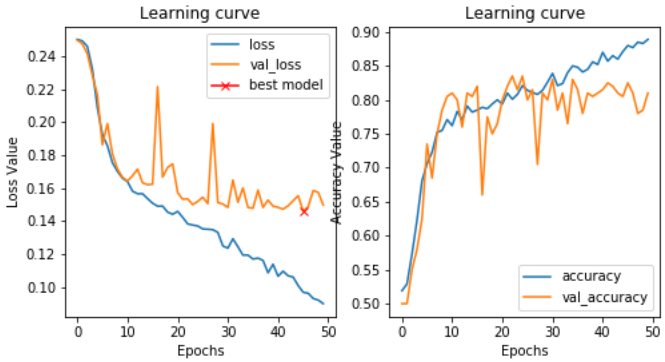


Figure 1: Loss and accuracy at epochs 50

Task 1b: The same training accuracy as task 1a can be observed at epochs 8. The final train accuracy is 99.9 %. The batch normalization accelerate the training process and stabilized the learning process within small number of epochs.

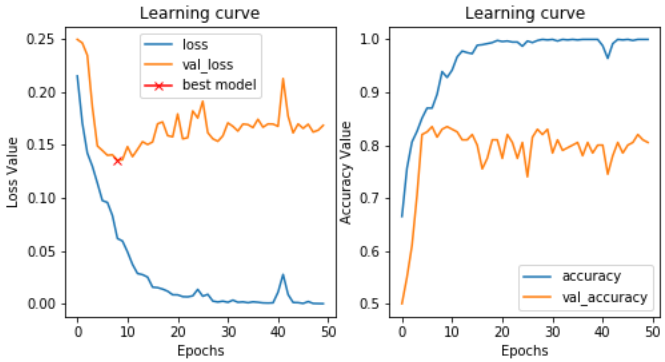
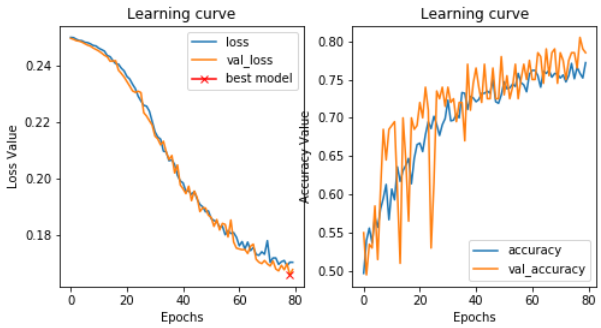
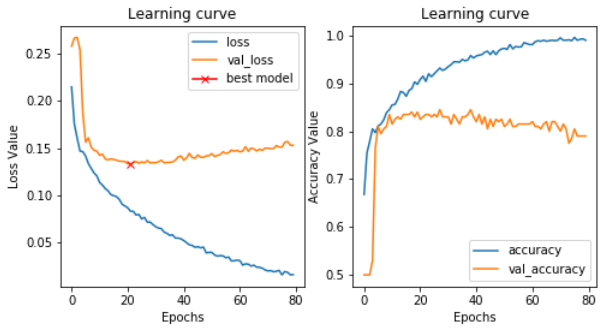


Figure 2: Batch normalization

Task 1c: The AlexNet model with batch normalization results in higher validation accuracy. Which is 82 % (at epoch 46). On the other hand, AlexNet without batch normalization results is 79.5 %. Moreover, in the previous task it has already pointed that the AlexNet with batch normalization accelerate the learning process. Therefore it can be said that the AlexNet with batch normalization has higher generalization power and it is more effective for this type of experiments.



(a)



(b)

Figure 3: Batch normalization vs without batch normalization

Task2: For this experiment we have added drop out layers with the the AleNet model with batch normalization. The drop out layers increases validation accuracy a bit and the accuracy seems more consistent. But it seems, more epochs could be useful for better understanding.

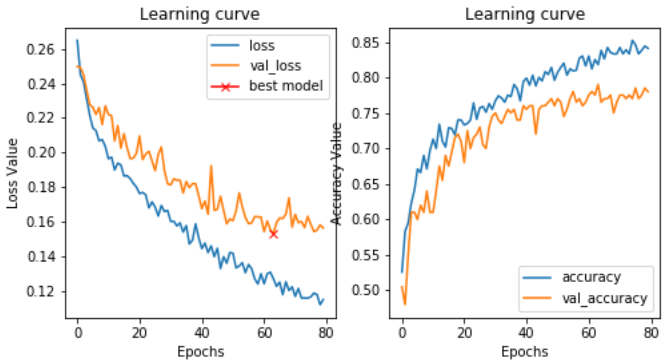
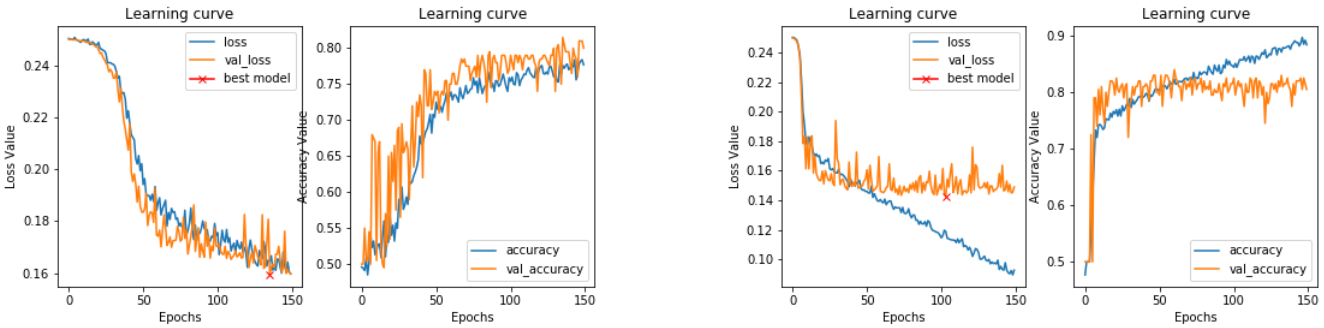


Figure 4: AlexNet with drop out and batch normalization

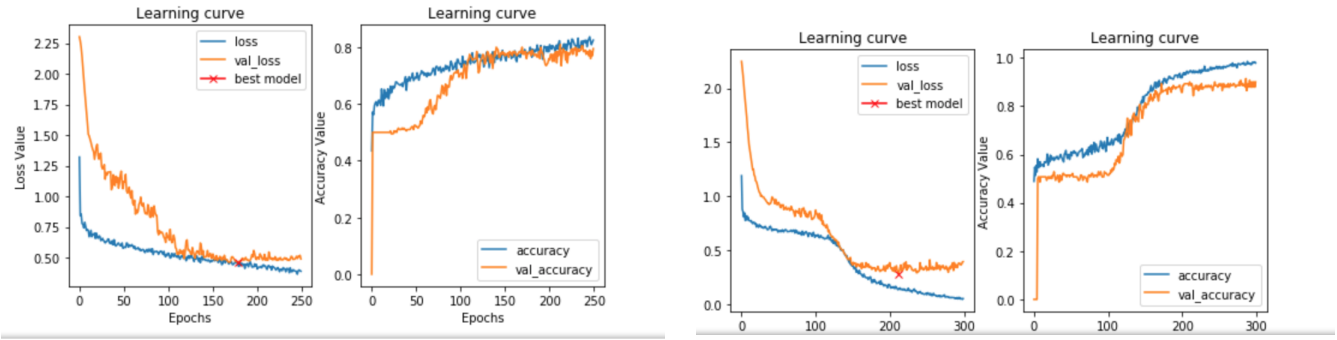
Task3:
From the Figure [5], we can say that the model with drop out (left picture) perform better than the other one. But a higher epochs might be shown better scenario. On the other hand, on the right picture, with without drop out, the validation accuracy stabilized after few epochs but the training accuracy is increasing. It indicates that the network is over fitted. So from this two experiment we can say that the drop out layers helps concerning the over fitted problem.



(a) (b)

Figure 5: Drop out (left picture) vs without drop out

Task4: During this experiments, we have designed the VGG16 by considering batch normalization and spatial dropout. Then we have played with different parameters. With the following parameters setting the learning curves are presented in Figure[6].
epochs = 250 (skin data) and 300 (for bone data)
batch size = 32
Base = 32
loss= sparse_categorical_crossentropy
optimizer = Adam(lr = 0.00001)



(a) (b)

Figure 6: Improved VGG16 for skin (left) and bone data

From the picture of the skin data, we can see that the overfitting problem has almost solved. On the other hand, for the bone data, the validation accuracy and the training accuracy curves are close to parallel. But for both of the cases, more epochs might be produced better outcomes.

Task5a: During this experiment, by changing different parameters like scale-factor, rotational angle, horizontal and vertical flip, and intensity rescaling, images are observed. Moreover, by changing the parameters values, it has been confirmed that the effects on pictures is working properly.

Task5b: In this part, a new augmentation generator has been developed.

Task6: A framework for training the model has been developed by considering augmentation. The learning curve for the Alexnet model based on this framework are given below

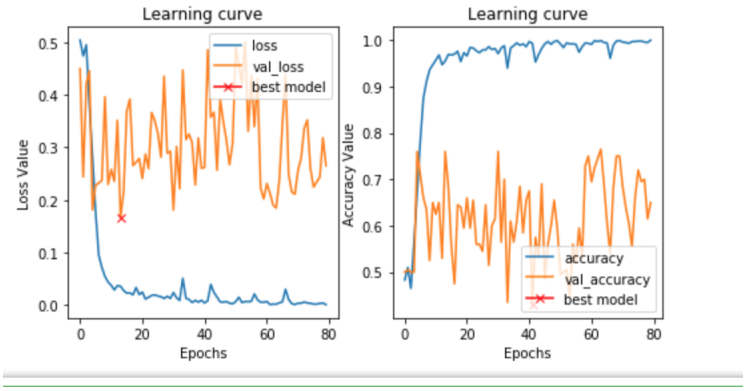
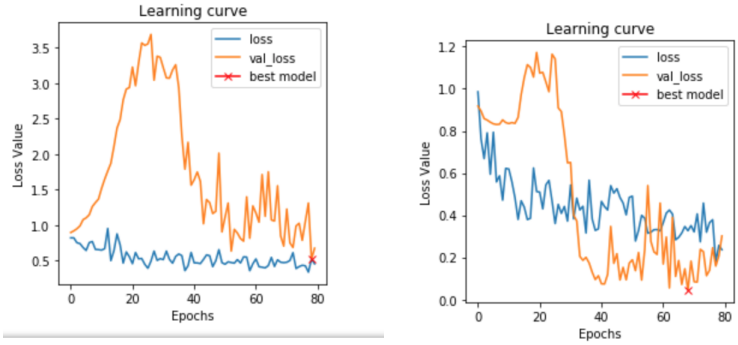


Figure 7: Learning curve of train with data augmentation

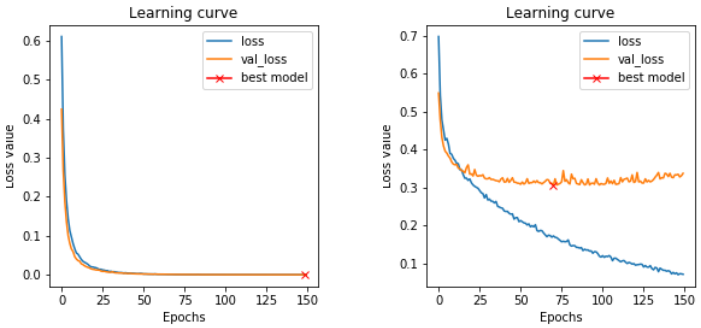
Task7: The VGG16 model has been performed for the skin and bone data. The learning curve has been presented in [8]



(a) (b)

Figure 8: Transfer learning for Skin (left) and Bone (right) images

Task8 and 9:



(a) (b)

Figure 9: Transfer learning for Bone (left) and Skin (right) images

Loss:
Bone: val_loss : 1.9106e-05 - val_accuracy:1.0000
Skin: val_loss : 0.3063 - val_accuracy:0.8800
From the figures we can see, transfer learning speeds up training and improves the model performance of the of the model. It works on Bone images but the model is overfitted on the skin images.

Task10: We have used VGG to train the model with data augmentation. But we can not get the activation maps by the following implementation in the instruction, even we add the line *tf.compat.v1.disable_eager_execution(*

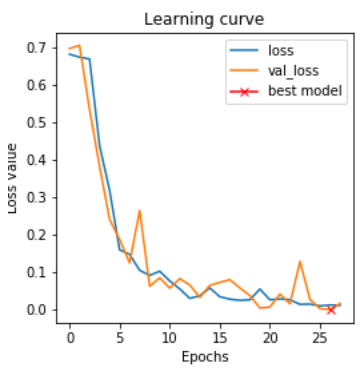


Figure 10: Learning curve of train with data augmentation