National Tsing Hua University
Fall 2023 11210IPT 553000
Deep Learning in Biomedical Optical Imaging
Homework 2

YOUHAN-WEN[1]

[1] *Institute of Photonics Technologies, National Tsing Hua University, Hsinchu 30013, Taiwan*

Student ID:110066554

1. Introduction

In this report, after learning the neural network for several lessons, now we are going to establish a module to distinguish the normal X-ray image and pneumonia X-ray image in grayscale data. First we will build a BCE loss module with Leaky-Relu as an active function and create a BC loss module with Leaky-Relu active function as well, now we will have the tools to see the performance between BCE and CE. Second we will try to adjust the hyperparameters to see the performance between BCE and CE, and in this part we will tune the batch size of the sequential function and the parameter of the step function such as step size and discuss the performance of CE loss module.

2. Performance between BCE loss and BC loss

First in BCE loss module, I will input three hidden-layer with Leaky-Relu as the active function, and the batch size is set as 128, which means each layer will have 128 nodes inside, and after five times testing I took the average of it, the output average train accuracy is 74.95%, after few tests I discover that while changing the epoch to 60 the accuracy doesn't become better, but

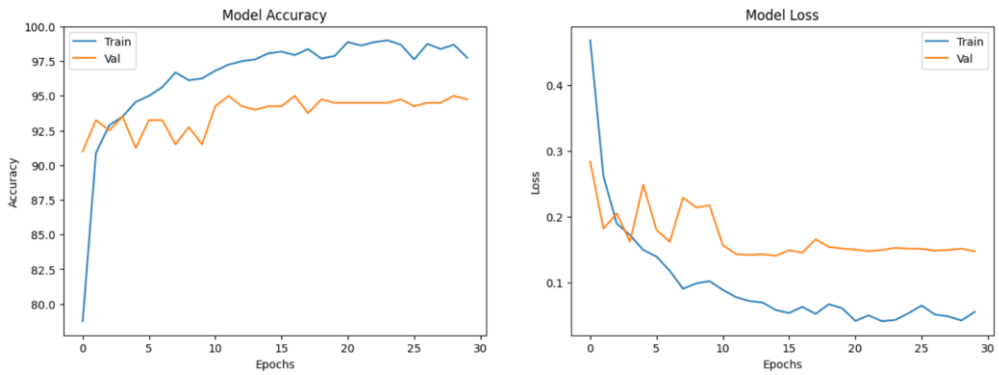

Fig. 1. This model accuracy and model loss of each epoch from test accuracy at 76.5% BCE loss module, it's seems like this BCE loss module is quite smooth form the train model loss, because the step learning rate or the batch size isn't fit to this module.

Second I also build three layer for CE loss module with Leaky-Relu, after five times of training average is 74.85%, in this CE loss module I can't understand why the accuracy is starts at 60%, in next part I will change the hyperparameters to flatten the validation to train slope, to have a better output figure or accuracy.

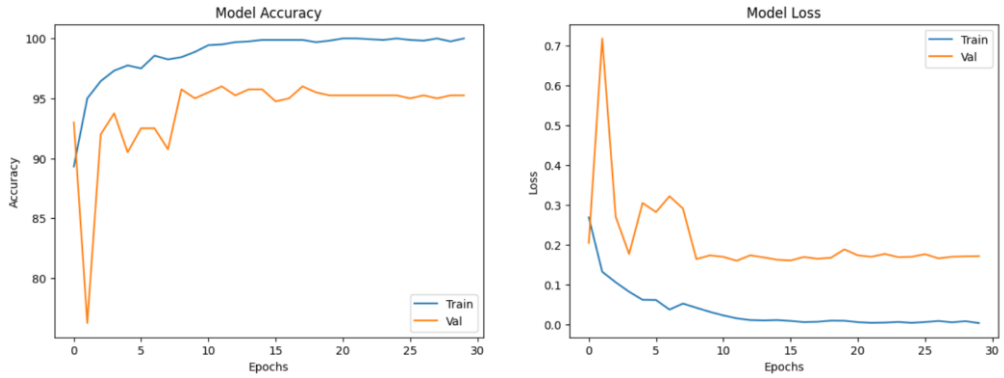

Fig. 2. This model accuracy and model loss of each epoch from test accuracy at 75.5% CE loss module, it's seems like validation loss is close to train loss in this CE loss module, but I can't figure out why it's sometimes start at around 60% accuracy.

After training these two module it's shows that BCE loss module will perform better output accuracy than CE loss module, I think the mainly reason is because I this topic we only want to detect is the lung is infecting or not, so BCE loss module can simply output the result by the function sigmoid is larger than 0.5, but in CE loss module case is commonly us in different multi output and when we us the code argmax(-1) the output will only feedback the largest value of the matrix, so for this reason the CE loss module can't easily beat BCE loss module in this condition.

3. Performance between Different Hyperparameters

3.1 Batch size

In this part we know that batch size in data loader which means that how many X-ray picture input to the for-loop of the train model, so in this section I'm going to tune CE loss module parameters of the batch size in 32,64 and 128 to see the different.

The batch size 64 after five times training the average is 72.8%, it's show that the accuracy isn't as high as batch size 32, which accuracy is 73.75%, but from fig.2 we can discover that the train loss is much smoother than batch size 32, but the gap between train accuracy and validation accuracy is still 5%.

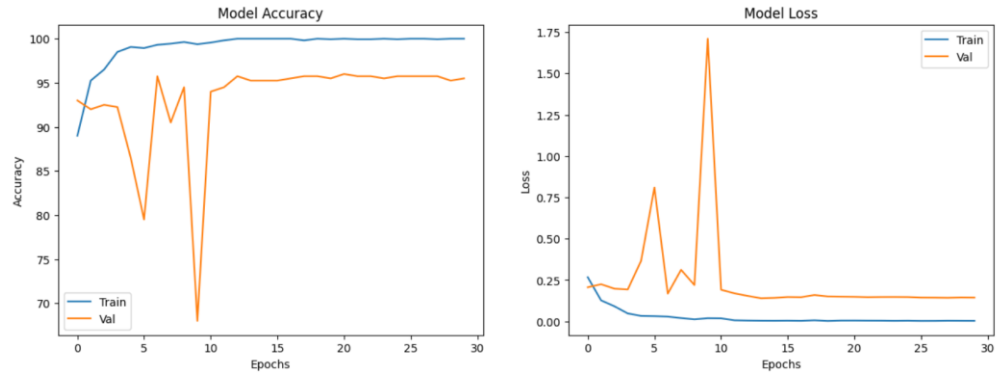

Fig. 3. Batch size 64 with 74% accuracy, this figure shows that the accuracy smoother than batch 32 the first 5 epoch slope isn't dramatic than before.

Next we change the batch size to 128, after five times average we got the accuracy 76.05%, the performance finally become higher than batch size 32 of BCE loss module, but we can see there are still few problems from fig.4, it shows that though the first epoch starts at almost the same accuracy but the gap between train accuracy and validation accuracy is still 5%, so to narrow this gap I have to change more hyperparameter in order to archive this goal, I will show this result at the conclusion.

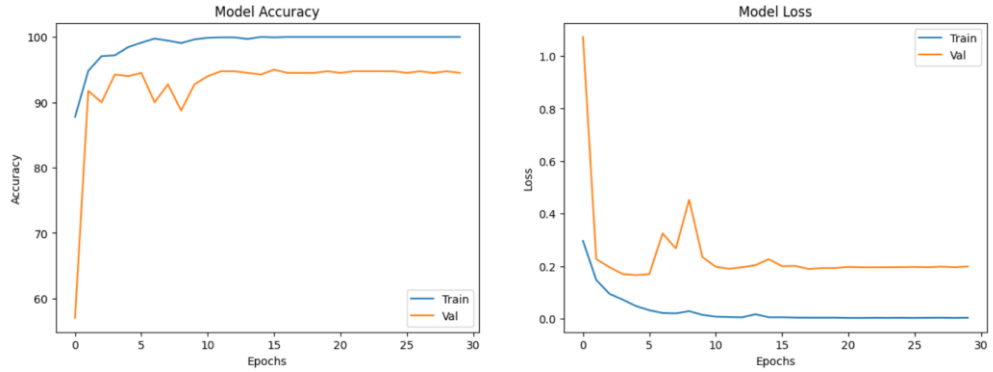

Fig. 4. Batch size 128 with 76.05% accuracy, this figure shows that the accuracy smoother than before which the jitter is smaller, but still the accuracy is still not high enough.

3.2 Step size

In this part I will fix the batch size back to 32, but change the step size inside the stepLR which can extend or downsize the period of learning rate decay, I will tune to 5 and 20 from the original parameter 10 to show the different.

When the step size period narrow to 5 the average accuracy is 74.75% after five times training, the accuracy is higher than original parameter, but from fig.5 it shows there is a pick of validation loss, I think which is because after 3 epochs the accuracy is quite good but at epoch 4 and 5 the validation data accuracy drop to 65%, so the feedback of the value is larger than usually even with gamma 0.1 can't flatten the curve, but over all the performance is acceptable the module didn't over fit or over train.

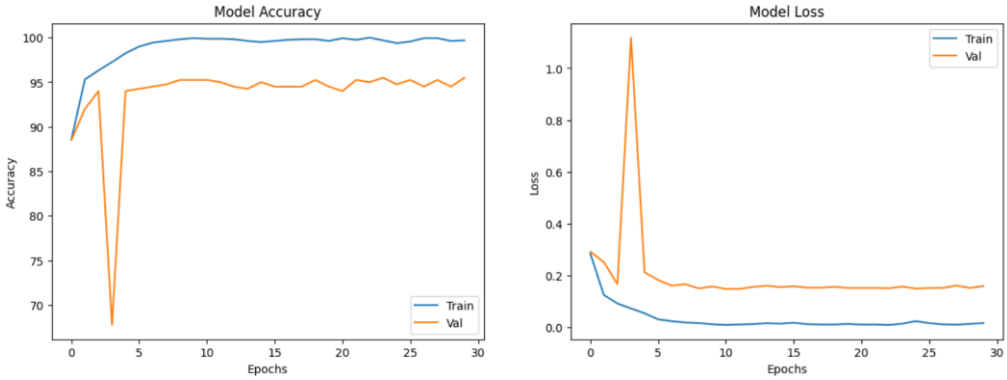

Fig. 5. Step size narrow to 5 the accuracy is 76%, the figure shows that after five data has been train the value will feedback to the node so the slope could become more attach to the train accuracy.

Next the change the step size to 20, after 5 times average the accuracy drop to 72.2%, from fig.6 we can see that the accuracy become extremely dramatic, though after almost 20 epochs it become stable, I think the problem is because when we extend the period of decay to 20 which means that in one train we reduce the number of the feedback, so this cause the dramatic picks, the validation loss should be very large to drag back the accuracy.

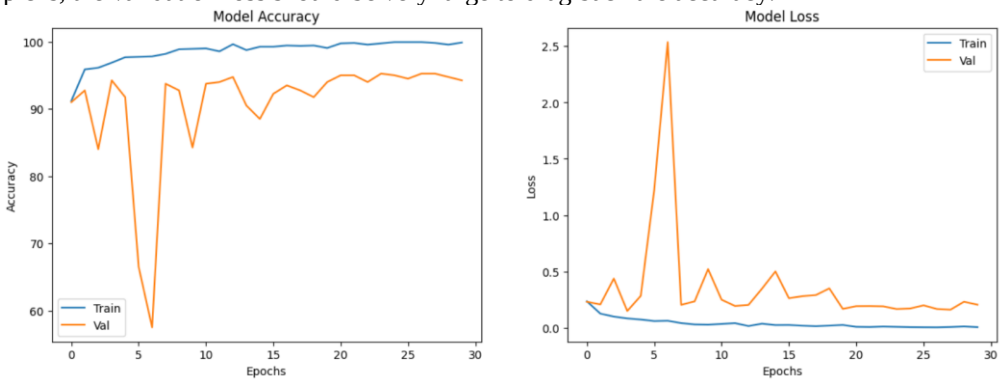

Fig. 6. Step size extend to 20 the accuracy is 79.5%, the figure shows that because the times of the feedback has been reducing, the curve line become shaking.

3.3 Combination of the several hyperparameters changes the performance

In this part I am going to tune several hyperparameter in the same time, in order to get the best result of the accuracy, and after reach the highest performance of CE loss module I will swop back to BCE with same hyperpaprmeter to see the performance of it.

First in CE loss module we change the batch size to 128, and we change the step learning rate hyperparameter, such as step size narrow to 3 and the learning rate to 1e-6 which can downsize each step of the X-ray image. Then we get the performance reach to 83% accuracy.

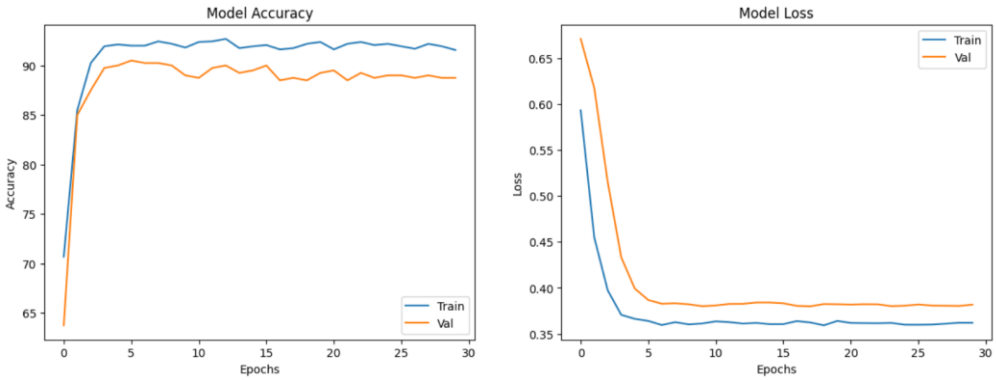

Fig. 7. The multiple hyperparameter changes the output accuracy performance to 83%, from the figure we can we the validation accuracy is quite attach to the train accuracy, which narrow the gap to 3%.

Second, we use the same hyperpaprmeter in to BCE loss module, in fig.8 we can see that the accuracy drops to 76.9%, it shows that the module is useless because and the validation accuracy is much higher than train accuracy, and the train accuracy and the validation accuracy gap is around 20%, I think it over fitting the module so that the performance isn't that well.

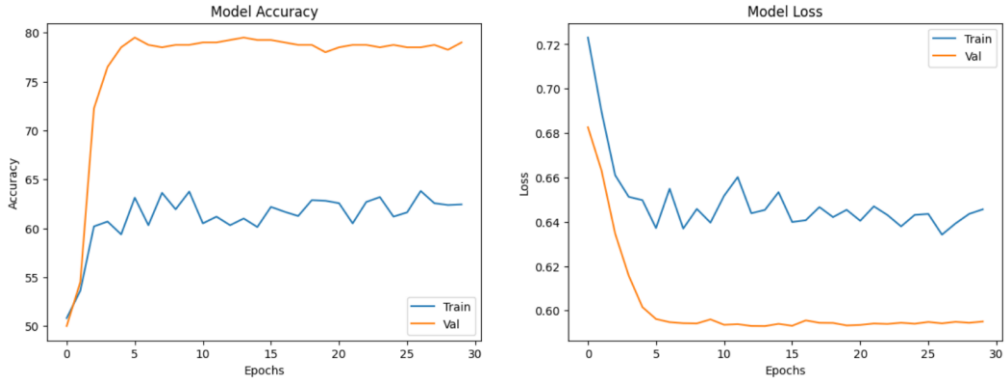

Fig. 8. The multiple hyperparameter changes to BCE loss module, it shows that the module is over fitting, though the accuracy average is 76.9%, this module seems like useless at all.

4. Conclusion

In this course we learn the full module of the X-ray image to detect is the lung is infecting or not, this technic can help doctors rapidly diagnosis the patient, the more time doctor can save the more human life cane be safe.

Homework 2 is quite difficult to people who don't know how to use python like me, but it's really interesting to learn the meaning behind the code, by reading the discerption by teacher, by change the parameters inside the loop, and much fun to discuss with classmates, the performance I have the best is 83% of accuracy, I think maybe change the learning function could have higher performance such as CosineAnnealingLR or ReduceLROnPlateau even MultiStepLR, and discover that hyperparameter in CE loss module perform well, but not the same perform in BCE loss module even I set the nn.Linear(128, 2) which I think the 2 represent to 2 output nodes.