National Tsing Hua University

Fall 2023 11210IPT 553000

Deep Learning in Biomedical Optical Imaging

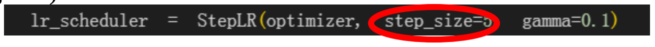
Homework 3

CHUN-HAO LI[1]

[1]*Ultrafast Photonics Lab, Institute of Photonics Technologies, Tsing Hua University.*
Student ID:112066513

1. Task A: Reduce Overfitting

In the original Lab 4's code, we have encountered a problem that we named "Overfitting". So, I looked into the code and made some hypothesizes, the first is the size of the mini-batch is too small, then the nodes of each hidden layers are too less, then lastly, the learning rate is too quick. So, I have changed three hyperparameters, but for the first two hyperparameters, I didn't see a bit of improve of overfitting. So, I decided to try on changing learning rate. Learning rate is a hyperparameter very hard to tune, it controls how much to change the model with the result of estimated error each iteration when the weights are updated. It's hard to tune because if it is too small, it may take a long time to train or the training process may get stuck; but if it is too large, it may lead to an unstable training.
The figure1. is the changed learning rate. I changed the step from 10 to 5 (one half of the original learning rate).



```
lr_scheduler = StepLR(optimizer, step_size=5, gamma=0.1)
```

Fig.1. Changed learning rate

After I trained the model with modified learning rate, I observed a sightly improvement of the occasion. With comparing the two results between original and modified code (shown in the following Figs.). From the original code, the training accuracy of the plateaus part is 100% and the validation accuracy is 96.75%, with a difference of 3.25%. But after I modified the learning rate, though there is still a plateaus part in training, and the accuracy is reduced, the training accuracy is 98.94%, and the validation accuracy is 97%, with a difference of 1.94%. We can see it is lower than the original code. But this is the best result I can get. On average, the difference between training accuracy and validation accuracy is around 2.5-3%. Sometimes there is no improvement, I think it may be the problem of the dataset isn't balanced, or we don't have enough data to produce a good result.
The test accuracy is only 75.25%.

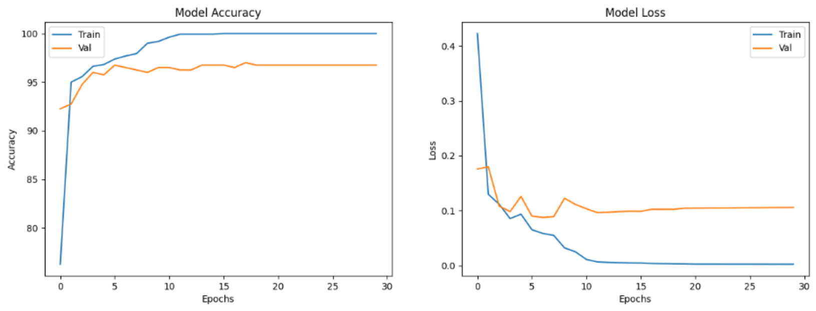


Fig.2. Result of original code

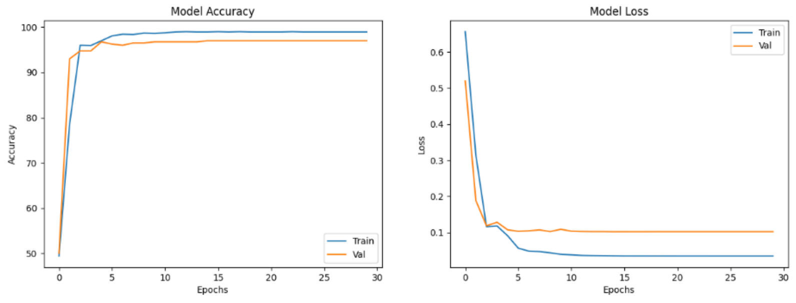


Fig.3. Result of modified code

2. *Task B: Performance Comparison between CNN and ANN*

2.1 CNN and ANN working principle

In the process of learning machine learning mechanisms, we come across Artificial Neural Network (ANN) and Convolutional Neural Network (CNN), the structures of the two neural networks are shown in Fig.4.

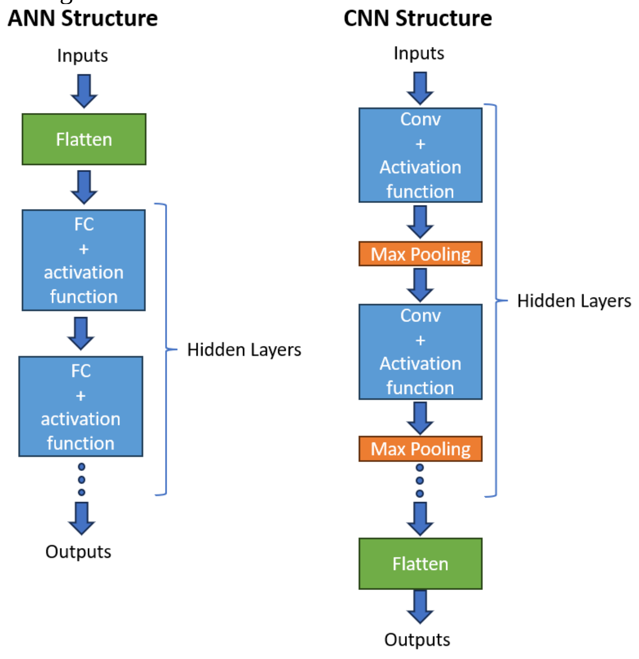


Fig.4. Two types of NN structure

For all neural networks, we will need forward propagation to calculate the output, and back propagation to calculate the validation loss, accuracy and weights, in order to check if our model can well classify the data. For ANN, it can only process one dimensional data, so we need to have this flatten procedure if the input datasets are 2-dim or 3-dim. Then we can start training the neural network with the flatten dataset. In the hidden layers of ANN, we only need fully connected layers with activation functions during the training process; the data in ANN only flows forward, so we call that “Forward Propagation”. As for CNN, the advantage is we can directly start the training process right away without a flatten sequence. But in hidden layers of CNN, we can have plural convolutional layers (the calculation of convolution is shown in Fig.5.), activation function and pooling layers (ex: maxpooling function, Fig.6. shows how maxpooling works.). The normal order of these three functions is, convolutional layer, followed by activation function and pooling layer is the last, we need to put this order as a combination before we flatten the results into a one-dimensional vector. Then we can put the one-dimensional vector into a fully connected layer for further processing before we output the results. The other difference between CNN and ANN is that CNN's weights are stored on the kernel we used for convolution, while ANN's weights are stored on each fully connected layer.

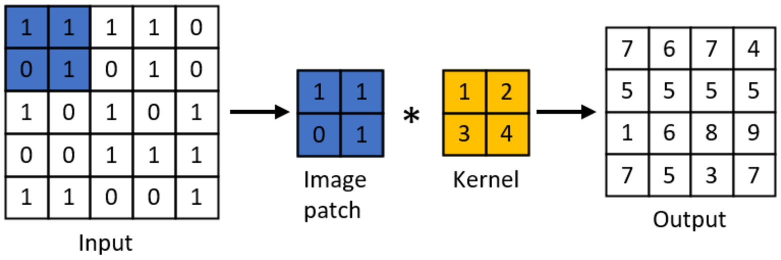


Fig.5. How convolution works

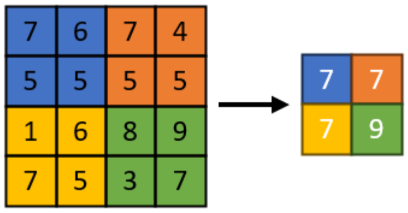


Fig.6. How max-pooling works

CNN training time exceeds 1 minute, while ANN training time only reaches maximum 20 second

2.2 Results of CNN and ANN with similar model

In CNN training, with convolution and pooling these two more pre-works before fully connected layers, the complexity is higher, so then the training speed is slower than ANN. We use a similar model to train CNN and ANN, and the training time for CNN exceeds 1 minute, while ANN only needs a maximum of 15 seconds. But we can compare Fig.7. and Fig.8., training accuracy, training loss, validation accuracy, validation loss of CNN is all better than ANN, and when I test the two models, CNN's test accuracy is 74%, but ANN only has 72%. The results show CNN has better performance than ANN, just as the training results show. But the training takes longer time, and we can refer from the figures that CNN needs further tuning to improve its overfitting problem.

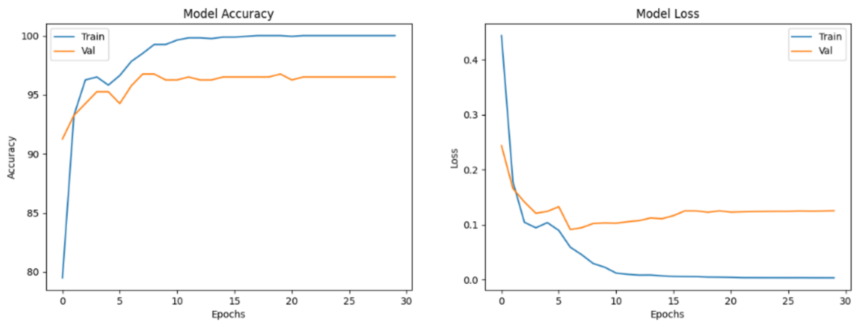


Fig.7. CNN training result

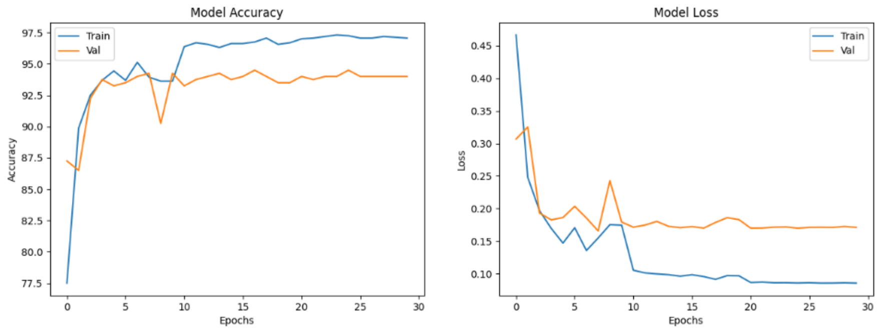


Fig.8. ANN training result

3. *Task C: Global Average Pooling in CNNs*

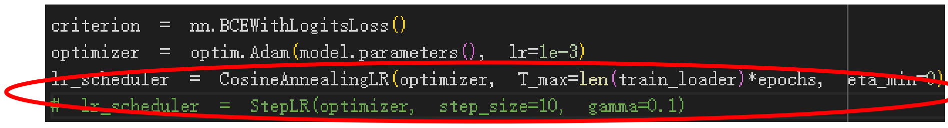
3.1 Introduce GAP and discuss on performance decline

Global Average Pooling (known as GAP) is a technique for replacing traditional fully connected layers in CNN with the advantage of reducing the effect of overfitting caused by fully connected layers. GAP produces a feature map for each corresponding category of the classification in the last convolution layer. The reason why GAP can avoid overfitting is that there isn't any parameter to tune. It takes the average of each input plane. In our code, we use "nn. AdaptiveAvgPooling2d(1)" as GAP technique, the 1 in brackets means we want to have only one output with each feature, so it will calculate the average of the previous outputs (which is the result of maxpool2d in the previous layer).

I assume the reason why GAP causes a slight performance decline is that when we average each feature, we presume the correctness of the feature with losing accuracy as a drawback (compare with the same structure as fully connected layer structure).

3.2 Increase Performance

While observing the result of the original code, I found that apart from performance decline, there is a plateaus part of the training. I assume this is because the StepLR learning scheduler we use will cause a plateau when calculating the gradient descent of the learning scheduler, and hence trapped the program at the unwanted place. The method I implemented is changing the learning scheduler to CosineAnnealingLR, with the input set as Fig.9. shows. After several training and testing, I got much better performance for each train (the results are listed in Table 1.) .



```
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
lr_scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader)*epochs, eta_min=0)
# lr_scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
```

Fig.9. Learning scheduler Setting

Table 1. Results of each train (only best results)

Train Num	Train accuracy	Train loss	Validation accuracy	Validation loss
1	91.96%	0.2525	92.5%	0.2286
2	90.62%	0.2457	91.75%	0.2184
3	90.38%	0.2634	92.25%	0.2419
4	90.25%	0.2673	92.5%	0.2367
5	92.25%	0.2175	93.75%	0.1942

As we can see from the table, the average of training accuracy is 91.092%, of training loss is 0.24928, of validation accuracy is 92.55%, of validation loss is 0.22396. Compared with the result of StepLR, training accuracy improves from 85% to 90%, training loss improves from 0.3 to 0.24, validation accuracy improves from 85% to 92%, validation loss improves from 0.35 to 0.22. For testing, the accuracy for all 5 tests is: 79.25%, 75.75%, 77.5%, 75.25%. While the testing result of StepLR is 71%. All the results are being visualized to graphs and inserted in the below.

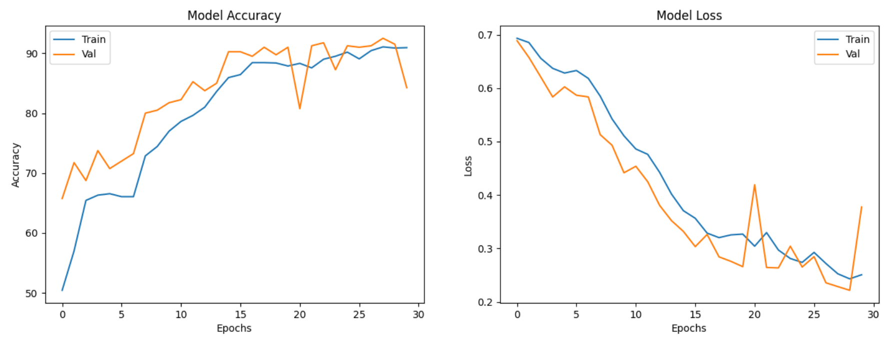


Fig.10. CosineAnnealingLR train 1

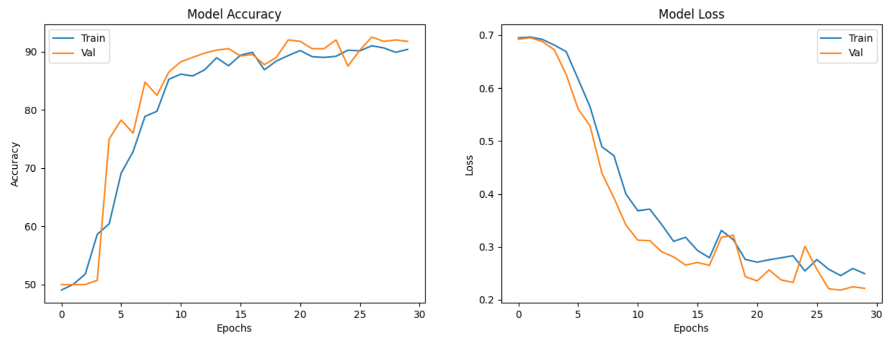


Fig.11. CosineAnnealingLR train 2

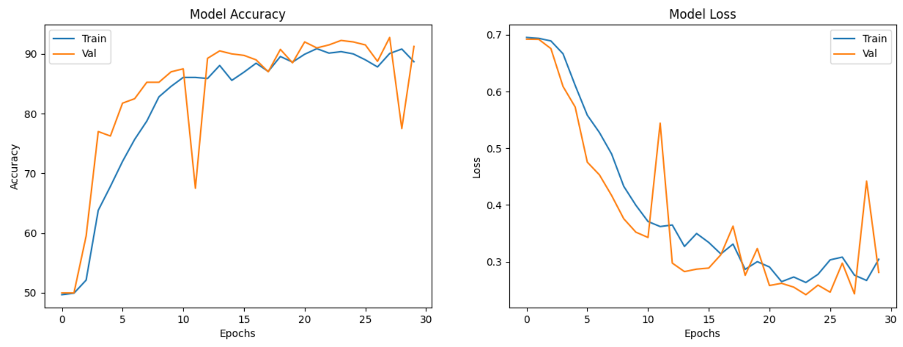


Fig.12. CosineAnnealingLR train 3

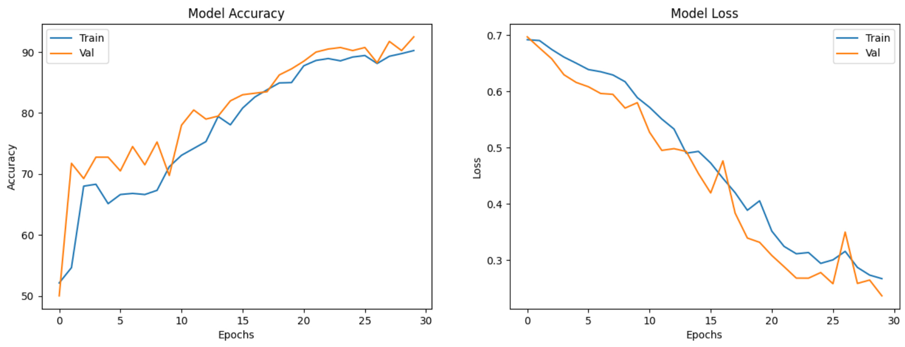


Fig.13. CosineAnnealingLR train 4

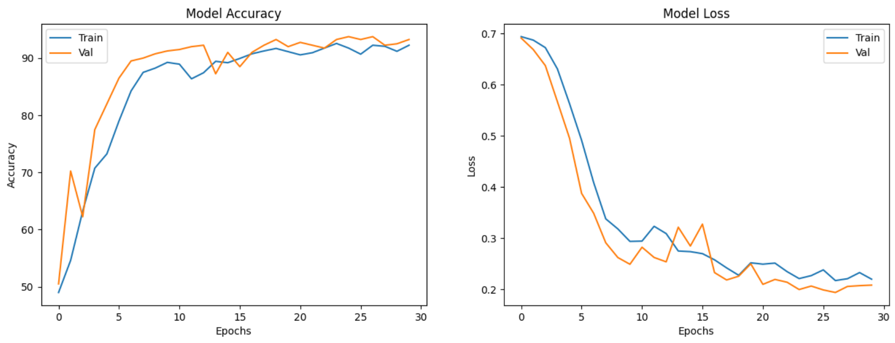


Fig.14. CosineAnnealingLR train 5

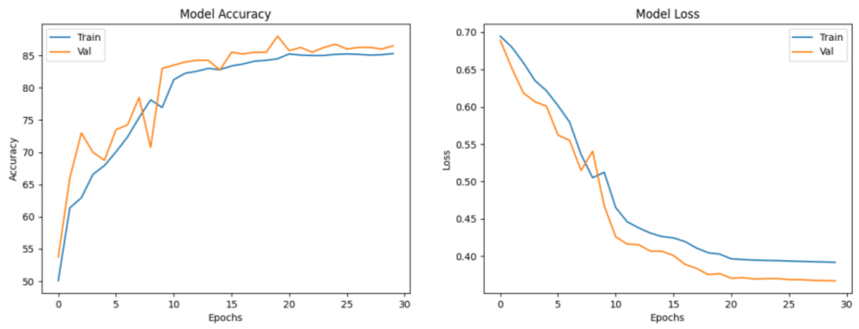


Fig.15. StepLR train

So, we can conclude that CosineAnnealingLR does improve the performance in training models constructed with GAP technique.