National Tsing Hua University
Fall 2023 11210IPT 553000
Deep Learning in Biomedical Optical Imaging
Homework 3

陳芷韻 *Student ID:112066514*

1. Abstract

In the report, we first try to reduce the over fitting in the training case with CNN model by adding the dropout layer and increasing the regularization, however, we cannot observe the significant improvements. Then, we aim to compare the structure of CNN and ANN model with the structure diagram and performance. Finally, we improve the overfitting problem of CNN_Global Average Pooling model by changing the learning rate scheduler.

2. Reduce Overfitting

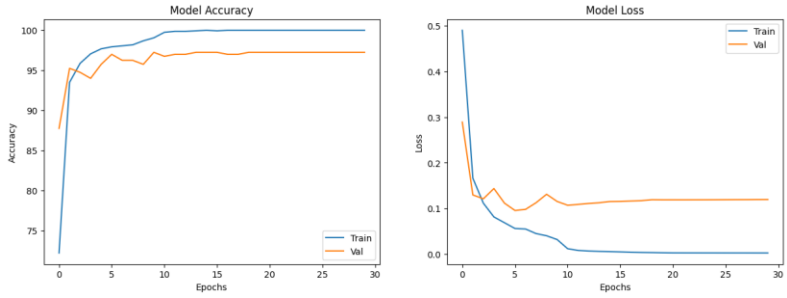
2.1 Fixed Parameters in the Test

The following (Table 1) shows the fixed parameters of the CNN training model that we were used.

Table 1. Fixed parameters in the comparison.

Parameters	Model	Loss	Epoch	Batch size	Learning rate scheduler
	CNN	BCE	30	32	StepLR

Original performance:
The performance of the above training model and parameter as (Figure 1) shows, the loss and accuracy are shown in Table 2.



(a) Model accuracy evolution (b) Model loss evolution

Figure 1. Training evolutions in the original CNN model.

Table 2.. Fixed parameters in the comparison

Train Loss	Train Accuracy	Best Val Loss	Best Val Accuracy	Test Accuracy	Training time
0.0022	100.00%	0.1194	97.25%	76.5%	1:11

Comparing the training and validation accuracy 100% and 97.25%, we can find that the validation accuracy is a slightly smaller than training one, which indicate it may exist the overfitting problem. In the following section, we try to improve the fitting problem.

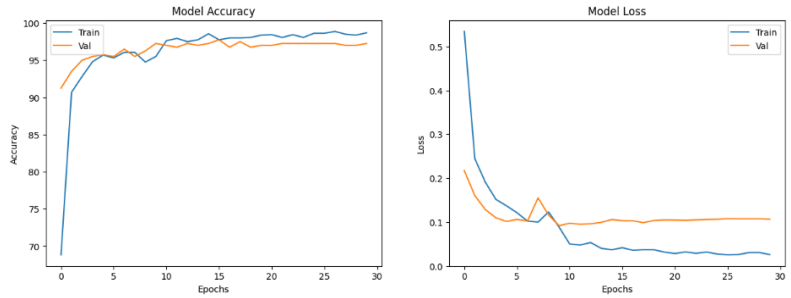
2.2 Performance Visualization of the Methods Reduced Overfitting

There are some common ways to reduce overfitting, here, we utilize the two methods respectively, adding dropout layer and increasing the regularization.

Method-1: Add Dropout

First, we apply the common used dropout layer, with the parameter **0.5** and find the validation accuracy didn't change. However, the training accuracy decrease to 98% as (Figure 2) and (Table 3) shows. It may since our dataset is smaller than common used, the large dropout 0.5 may loss some important information. Thus, we change the parameter to 0.1, however, the performance didn't be improved (Figure 3). Add dropout layer looks useless in this model and training dataset.

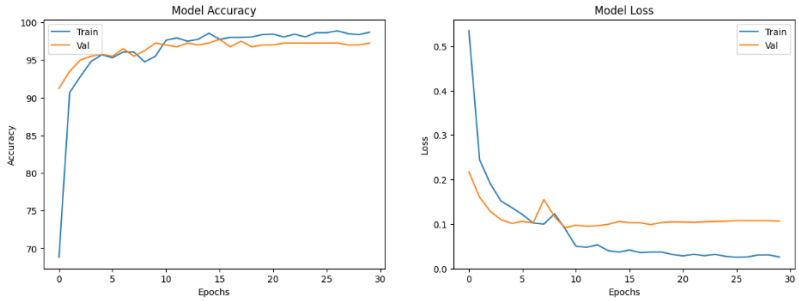
Dropout 0.5:



(a) Model accuracy evolution (b) Model loss evolution

Figure 2. Training evolutions with the Dropout 0.5.

Dropout 0.1:



(a) Model accuracy evolution (b) Model loss evolution

Figure 3. Training evolutions with the Dropout 0.1.

Table 3. Train, validation and test results in different dropout.

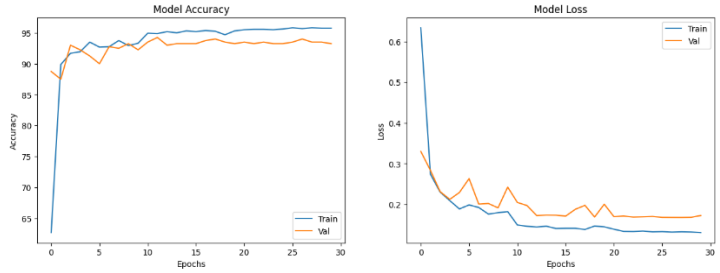
Dropout	Train Loss	Train Accuracy	Best Val Loss	Best Val Accuracy	Test Accuracy	Training time
(non)	0.0022	100.00%	0.1194	97.25%	76.5%	1:11
0.5	0.0258	98.69%	0.1064	97.75%	76.25%	1:11
0.1	0.0340	98.44%	0.1227	96.25%	75.00 %	1:10

Method-2: increase regularization

Second method we use to solve overfitting is increasing the regularization by adding the weight-decay in the **optimizer function** (the code is shown in section 2.3).

Weight decay 0.1:

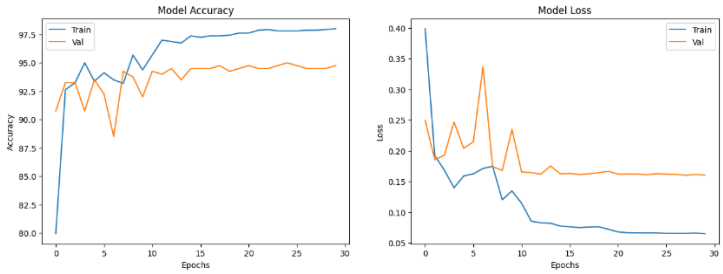
We also start from the common use weight decay number 0.1, the performance as Figure 4 and Table 4 shows. From the result, we can observe both training and validation accuracy are decrease to~95%. The number 0.1 may too large for our dataset size. Thus, we decrease the number to 0.01, and the performance is improved compare to the num 0.1. However, it still doesn't better than the original model.



(a) Model accuracy evolution (b) Model loss evolution

Figure 4. Training evolutions with the increased regularization (weight decay 0.1).

Weight decay 0.01:



(a) Model accuracy evolution (b) Model loss evolution

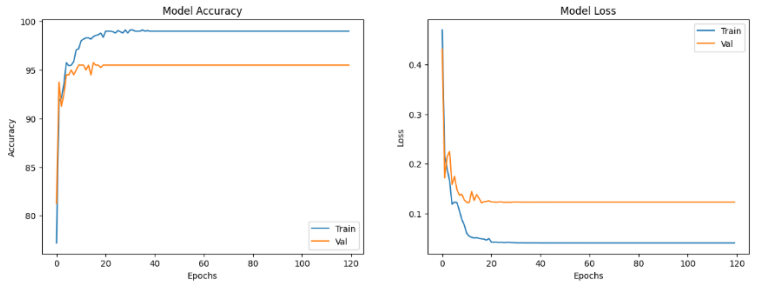
Figure 5. Training evolutions with the increased regularization (weight decay 0.01).

Table 4. Train, validation and test results in different weight decay.

Weight decay	Train Loss	Train Accuracy	Best Val Loss	Best Val Accuracy	Test Accuracy	Training time
(non)	0.0022	100.00%	0.1194	97.25%	76.5%	1:11
0.1	0.1307	95.75%	0.1679	94.25%	73.75	1:10
0.01	0.0652	98.00%	0.1601	95.00%	75.5%	1:10

Until now, we just change a parameter at a training process, however, in the above cases, it may have better performance with other parameters. Now, based on the last mode and parameter, we increase the training epoch to120, and find the lower training accuracy can be improved (Figure 6) (Table 5), and the above low training accuracy is just still not converged yet. Overall, the methods increasing the regularization is unsuitable in the model.

Weight decay & increase epoch number



(a) Model accuracy evolution (b) Model loss evolution

Figure 6. Training evolutions with the dropout 0.1.

Table 5. Train, validation and test results as weight decay 0.01 and epoch 180.

Weight decay	Train Loss	Train Accuracy	Best Val Loss	Best Val Accuracy	Test Accuracy	Training time
0.01	0.0408	99.00%	0.1217	95.75%	75.25%	4:42

2.3 Code

Method-1: Different from original ConvModel, we add the **dropout layer** into the new CNN model, both in the initial and forward part. Because the size of our data set is not too large, we only use one layer of dropout to modify.

```
class ConvModel_Re(nn.Module):
    def __init__(self):
        super().__init__()
        # 1 channel, and using 3x3 kernels for simplicity, 256*256
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) # 128*128

        self.conv2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) # 64*64

        self.conv3 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) # 32*32

        # Adjust flattened dimensions based on the output size of your
        flattened_dim = 32 * 32 * 32

        self.fc1 = nn.Linear(flattened_dim, 32)
        self.dropout = nn.Dropout(0.5)  # 添加dropout層
        self.fc2 = nn.Linear(32, 1)
```

```
def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.pool1(x)

    x = F.relu(self.conv2(x))
    x = self.pool2(x)

    x = F.relu(self.conv3(x))
    x = self.pool3(x)
    # Flatten the output for the fully connected layers
    x = x.reshape(x.size(0), -1)

    x = F.relu(self.fc1(x))
    x = self.dropout(x)  # 在fc1之後應用dropout
    return self.fc2(x)
```

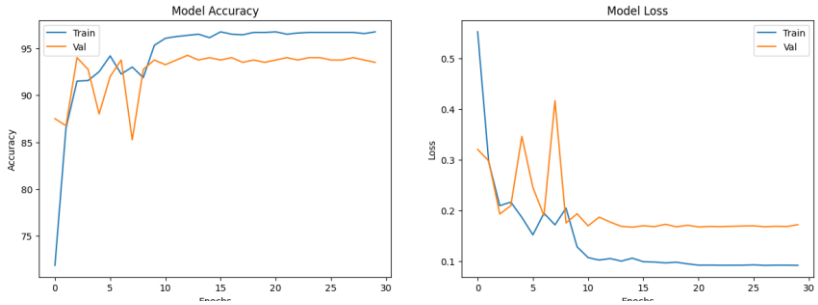
Method-2:

```
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=0.1)
```

3. Performance Comparison between CNN and ANN

3.1 Discussion

The following (Figure 7) and (Table 6) shows the performance after using the same parameters as (Figure 1) but changing the trained model from CNN to ANN.



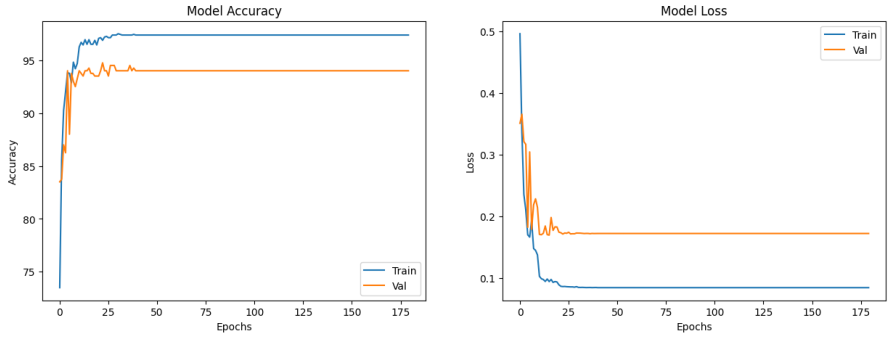
(a) Model accuracy evolution (b) Model loss evolution

Figure 7. Training and validation evolutions with ANN model.

From the results, it appears that with both CNN and ANN models set to train for 30 epochs, the training and validation accuracy of ANN is lower than that of CNN. However, it's worth noting that ANN training takes only one-sixth of the time compared to CNN. Therefore, if you have a large amount of training data and time constraints, you might consider using ANN.

Increase the epoch number:

Next, we wanted to investigate if setting the training time for ANN equal to that of CNN (by increasing the number of epochs to six times the original) would lead to similar accuracy. The results were not as expected; the accuracy began to converge quickly at around 40 epochs and didn't reach the same level as the ANN.



a) Model accuracy evolution (b) Model loss evolution

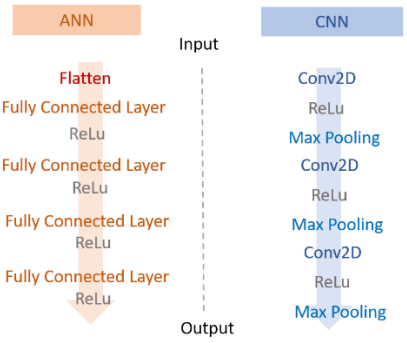
Figure 8. Training and validation evolutions with ANN model, increase the epoch number.

Table 6. Train, validation and test results of ANN and CNN.

Model	Train Loss	Train Accuracy	Best Val Loss	Best Val Accuracy	Test Accuracy	Training time
CNN	0.0022	100.00%	0.1194	97.25%	76.5%	1:11
ANN	0.0919	96.75%	0.1673	94.25%	72.75%	00:12
ANN	0.0846	97.38%	0.1698	94.75%	72.75%	1:11

3.4 Architecture Description of CNN and ANN

ANN structure is constructed by the fully connect layer, it usually is applied as we need to solve the data in the table form, and CNN is useful as processing the image data, which is consist of convolution layer and pooling layers. For the data processing, it is need to flatten the data to 1-dimension. During image training, if we use ANN (fully connected layer), it may cause larger computations.



Architecture diagram of CNN and ANN model.

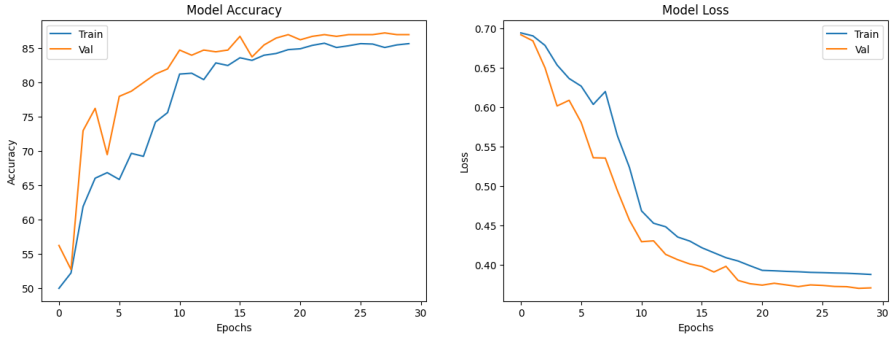
4. Global Average Pooling in CNNs

4.1 Explanation of GAP

Global Average Pooling (GAP) allows you to aggregate information from each channel into a single average value to form 1D data, without the need for additional flattening. This reduces the complexity of the network architecture. In the model we add the layer nn.AdaptiveAvgPool2d() to achieve GAP.

4.2 Increase Performance

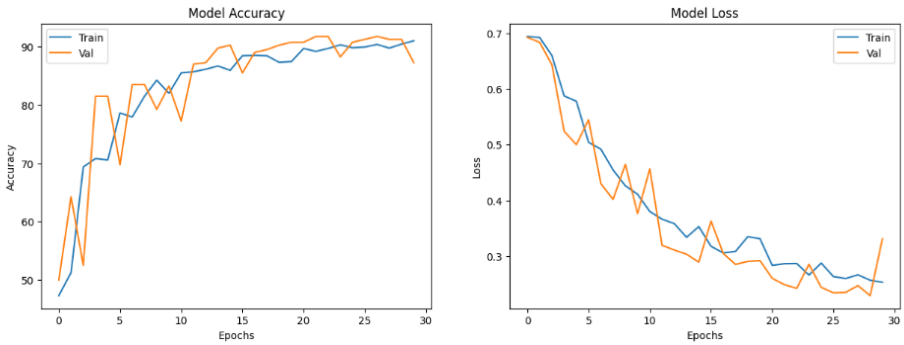
In this section, we try to discuss the performance of GAP CNN, the original model with the learning rate scheduler, StepLR, as (Figure 9) and (Table 7) shows.



a) Model accuracy evolution (b) Model loss evolution

Figure 9. Training and validation evolutions with CNN_GAP model (ori).

Therefore, when we changed the learning rate scheduler to CosineAnnealingLR, we can observe the improved alues, but there was a significant oscillation, (Figure 10). Thus, we let the model train longer (epoch: 200), and find the improvement.



(a) Model accuracy evolution (b) Model loss evolution

Figure 10. Training and validation evolutions with CNN_GAP model (change learning).

Table 7. Train, validation and test results of different learning rate scheduler in CNN_GAP.

Model	Train Loss	Train Accuracy	Best Val Loss	Best Val Accuracy	Test Accuracy	Training time
StepLR	0.3875	85.69%	0.3698	87.25%	72.75%	1:11
CosineAnnealingLR	0.2533	91.00%	0.2294	91.75%	78.5%	1:12
CosineAnnealingLR	0.1018	96.44%	0.1410	95.00%	70%	7:50