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

YOUHAN-WEN[1]

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

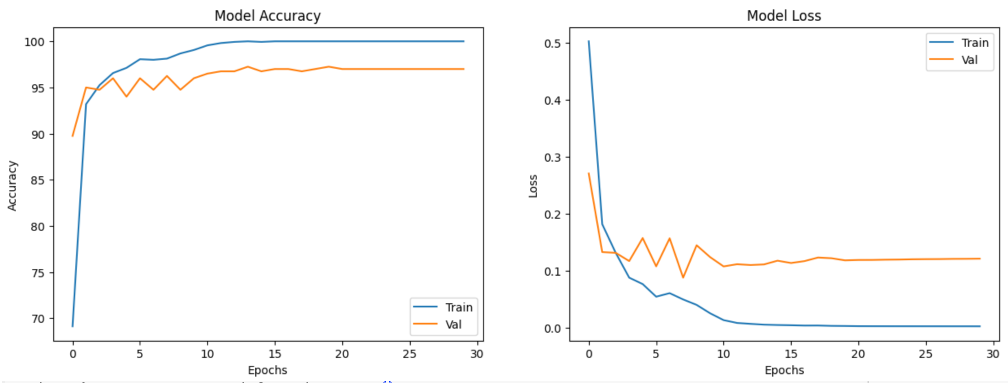
Student ID:110066554

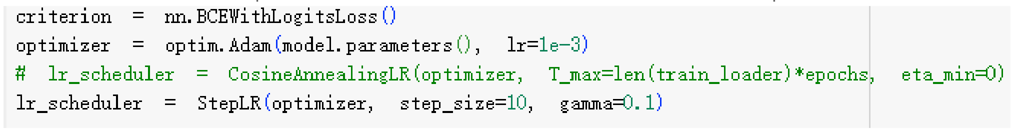
1. Introduction

In this report, we will try to demonstrate ANN and CNN model and compare the performance, in task A we will reduce the overfitting by tuning the hyperparameter and I will point out another algorithm try to improve the performance. In task B I will compare the pros and cons of the ANN and CNN model, and try to explain the meaning of the coding process, and use the diagram to visualize the movement. In tack C I will use a new CNN process method which is ConvGAP to retrain the model, and try to enhance the performance.

2. Task A: Reduce Overfitting

In this part of the report, we will try to enhance the accuracy of the CNN module, without any changing of the hyperparameter the accuracy of this module is 77%. And in fig.1 we can found out that after 15 epochs the train accuracy will be fixed at 100% and the validation accuracy will fix at 97.25% so we could reduce the epochs to increase the training time. And we also can find out the train accuracy is much higher than the test accuracy so I think this model has an over fitting, so next action we will do is to enhance the performance of this ConvModel.

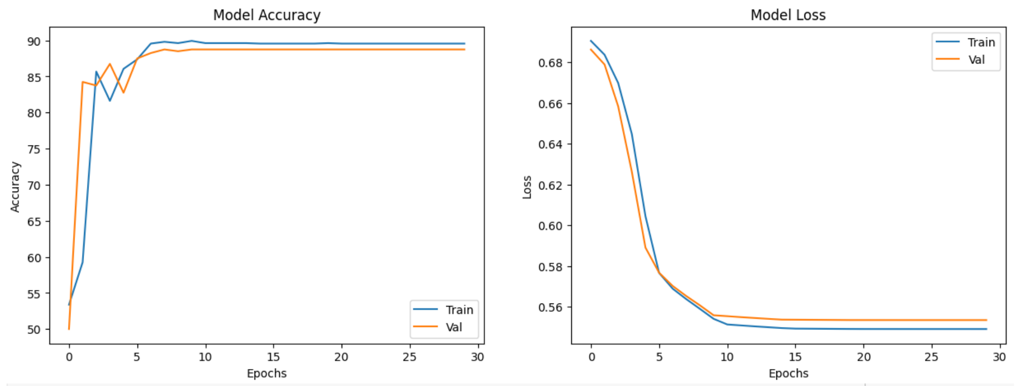


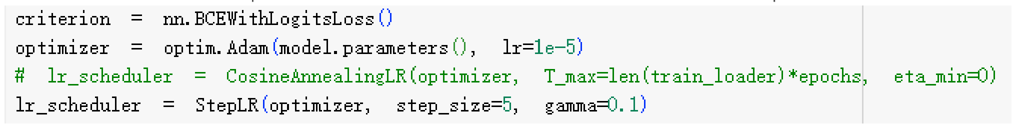


```
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
# lr_scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader)*epochs, eta_min=0)
lr_scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
```

Fig. 1. This model accuracy is 77% from this figure it shows that train accuracy reach to 100% after 15 epochs, but the validation accuracy only reach 97.25% so we can say this ConvModel has 23% higher than the test accuracy that is an over fitting.

First, I fixed the batch size to 32 only tuning the hyperparameters of the BCE loss module, I decrease the learning rate form 1e-3 to 1e-5 which can smaller the steps of the parameter space during the optimization, but in the same time it might cause the optimization process stuck in the local minima. Second, I narrow down the step size from 10 to 5, which can help the module avoid overshooting the optimal parameters by slower the convergence.





```
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
# lr_scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader)*epochs, eta_min=0)
lr_scheduler = StepLR(optimizer, step_size=5, gamma=0.1)
```

Fig. 2. This model accuracy is 82.75% after the hyperparameter changes, from the figure we can see the train accuracy and validation accuracy is much attach to each other, the train accuracy is 89.62% after 10 epochs and the validation accuracy is 88.75, and form the model loss we can see that the feedback is smoother than before, which I think is because the step size is smaller that can help the value of the weights feedback more frequently.

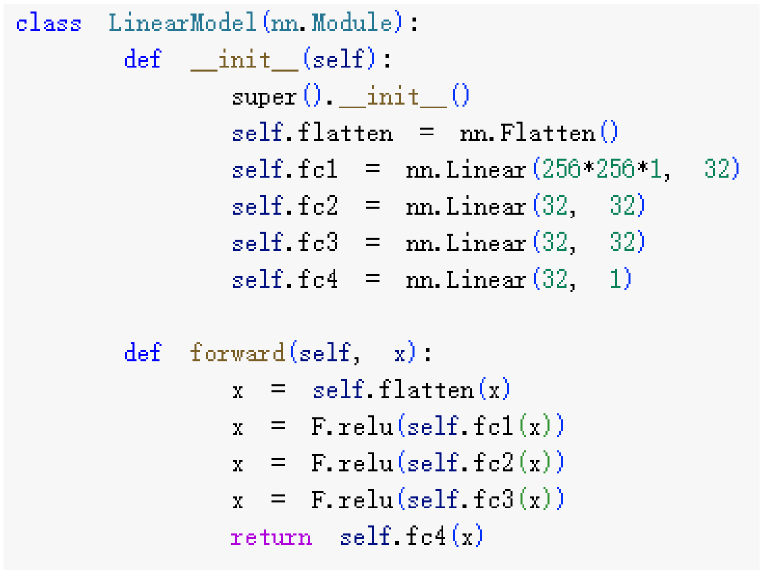
In the part we can see the spending time of the original model takes 71 seconds, and the modulate model takes 73 seconds and with higher accuracy of the test data accuracy, which reduce from 23% to 6.87% so I can say the model have conquered the over fitting issue. When I search on internet I found there is another way to overcome the over fitting issue, which is changing the optimizer we used, here in this module we used Adam for the algorithm, but if we swop to another algorithm such as SGD, RMSprop, Adamax or even AdamW, we can outperform or match Adam in another cases.

3. Task B: Performance Comparison between CNN and ANN

3.1 Artificial Neural Network (ANN)

ANN is a group of multiple neurons in each layers, which is known as feed forward neural network, and it’s also the simplest variants of neural network. ANN typically used to process and analyze the financial data, which has the advantages of ability to handle complex data and having fault tolerance that give ANN ability to work with incomplete knowledge, although ANN still have the disadvantage of highly hardware dependence for the large processors with parallel processing power of the structure.
From fig.3 we used linear model for this ANN report, we use the nn.Flatten to reshape into an one-dimensional vector and in first fully connected layer we use the image X=256, Y=256 and Z=1 for 65536 input values and applied a linear transformation to produce 32 output values and passed through an activation function where we use ReLu and then introduce non-linearity into the network. So in this linear model in input 4 hidden layer in the module.
Next I will remain the hyperparameter we used in last part and run with ANN model, in fig.4 we get the test accuracy 81%, train accuracy 87.56% and validation accuracy 87.25%, it shows the almost as good as CNN model performance the gap of the train and test accuracy is only

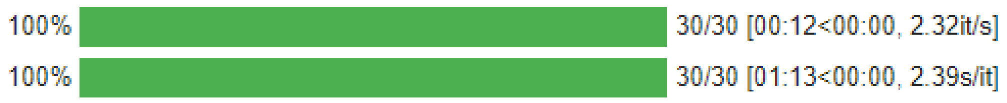
6.56% and even take less time to run the model which only takes 12 seconds compare with CNN model 73 seconds.



```
class  LinearModel(nn.Module):
        def  __init__(self):
                super().__init__()
                self.flatten  =  nn.Flatten()
                self.fc1  =  nn.Linear(256*256*1,  32)
                self.fc2  =  nn.Linear(32,  32)
                self.fc3  =  nn.Linear(32,  32)
                self.fc4  =  nn.Linear(32,  1)

        def  forward(self,  x):
                x  =  self.flatten(x)
                x  =  F.relu(self.fc1(x))
                x  =  F.relu(self.fc2(x))
                x  =  F.relu(self.fc3(x))
                return  self.fc4(x)
```

Fig. 3. Linear model we use in this ANN we import 4 hidden layer and use ReLu as activation function.



```
100%  30/30 [00:12<00:00, 2.32it/s]
100%  30/30 [01:13<00:00, 2.39s/it]
```

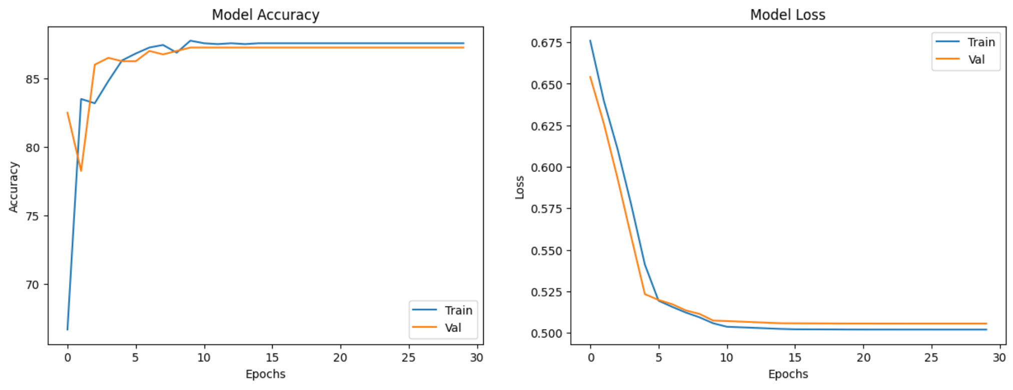


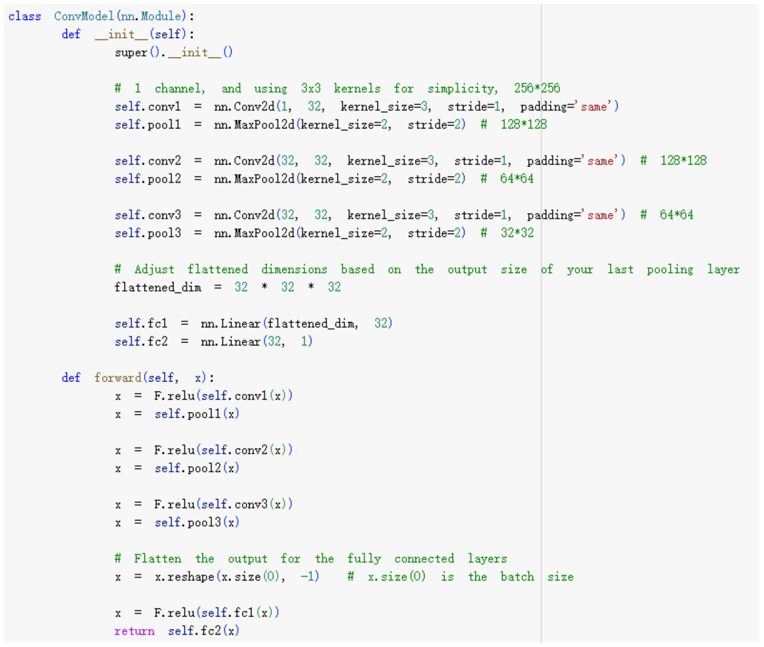
Fig. 4. The test accuracy reach 81% for this ANN model and we can see that the model didn’t over fitting compare with original code.

3.2 Convolutional Neural Network (CNN)

CNN are mainly used in image and video recognition, recommendation systems, and image analysis and classification, which can help scale the process by using linear algebra principles and to identify patterns in images, CNN are based on three main layers which is convolutional layer, polling layer and fully connected layer, CNN have the ability to see deeply of the image by stacking more layers.
Form the fig.5 I will briefly descript the code we going to use, first we will input the image from the data X=256, Y=256 and the Z=1 and after input to Conv2d the output will become 32

outputs, to emphasize that 32 doesn't mean the dimension, and the Conv2d will have the algorithm with 3*3 size of convolutional kernel and with one step of the pixel point increase, so the output of the Conv2d will have the same size compare with the input data but with 32 outputs, and MaxPool2d can downsize the input to 128*128 by using the kernel 2*2 matrix and with two step of the pixel so after the MaxPool2d the output will become 128*128*32, and after three times of convolution and pooling the output will down size to 32*32*32 we can see this result at the code flattend_dim.

In this part on conclusion we figure out that ANN is good at process text data and CNN is good at process image data, but in this report with hyperparameter modulating the performance is quite close, but the processing time has huge different which has almost five times longer for the CNN model, and the mainly reason is think is because ANN easily simplify the values to 32 outputs and maintain this 32 output until the result came out, but CNN should apply convolution algorithm and after that we slightly narrow the matrix size divide by two till the output, so I think the reality of the calculation is much larger than ANN model, that the calculation or the process function is way larger.



```
class ConvModel(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='same')
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) # 128*128

        self.conv2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same') # 128*128
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) # 64*64

        self.conv3 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same') # 64*64
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) # 32*32

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

        self.fc1 = nn.Linear(flattened_dim, 32)
        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.size(0) is the batch size

        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

Fig. 5 Linear model we use in this ANN we import 4 hidden layer and use ReLu as activation function.

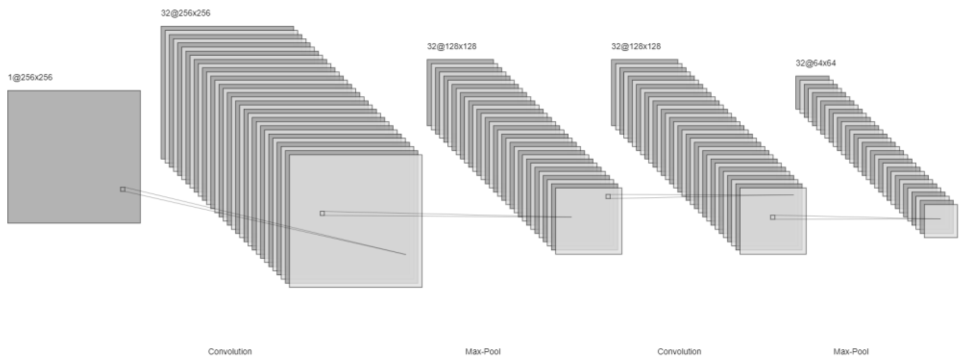


Fig. 6 CNN diagram with similar to this report, in this diagram in write two times of convolution and pooling loop, we can see we input a 256*256 with one-dimension image with first time of convolution which expend to 32 output, and with the first max pooling the matrix reduce two times by a 2*2 matrix with two step of movement, and second convolution we can recalculate the image with more eigenvalues, and downsize the image size again.

Table 1. Characteristic of the ANN and CNN

	ANN	**CNN**
BASIC	Simplest type of neural network	Most popular type of neural network
DATA TYPE	Text data	Image data
COMPLEXITY	Simple than CNN	More powerful than ANN
MAIN DRAWBACK	Hardware dependence	Large training data required
USES	Complex problem solving	Image recognition

4. Task C: Global Average Pooling in CNNs

The mainly use of the convGAP is to replace fully connected layer, with the convModel we will have the fully connected layer as a hidden layer, but in convGAP take out the hidden layer directly to the output nodes, shows in Fig.7. From chatgpt it says the main advantage of GAP is to reduce the number of parameters in the model and produces a fixed-size representation regardless of the input size. which can help mitigate overfitting and provide a more compact feature representation for making predictions, especially in image classification tasks.

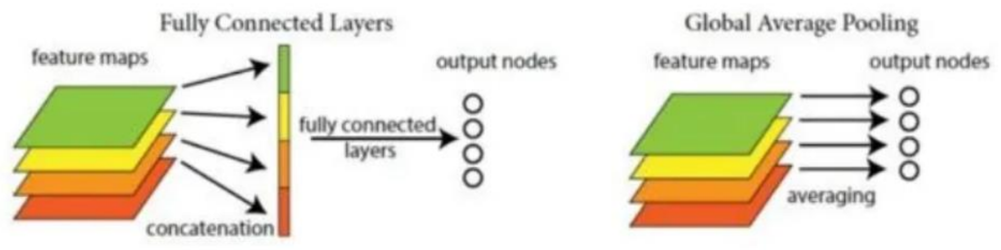


Fig. 7. The convModel and the convGAP model diagram, we can see that the GAP take out the fully connected layer with directly output.

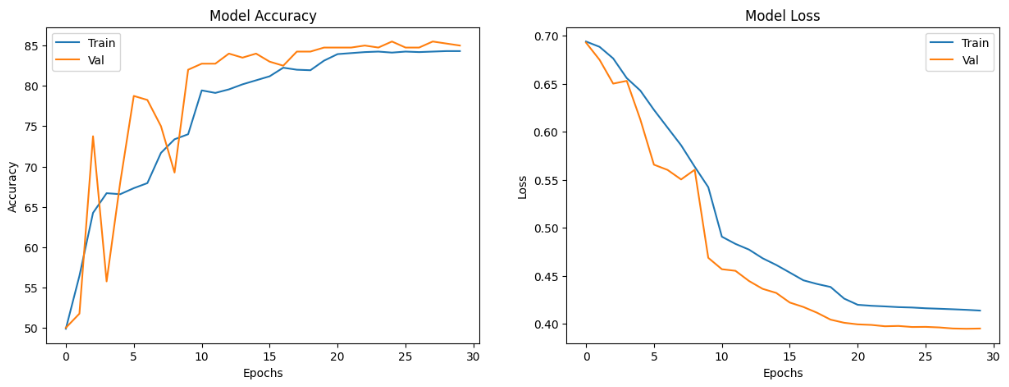


Fig. 8. The test accuracy is 70.5%, train accuracy is 84.31% and the validation accuracy is 85.5%, it shows over fitting of the model, next I will try the hyperparameter which we use in convModel, and see the performance.

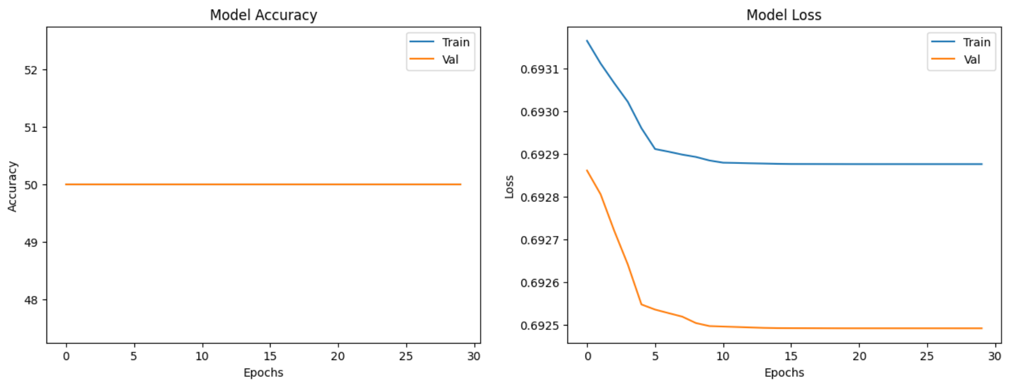


Fig. 9. The test accuracy is 50%, train accuracy is 50% and the validation accuracy is 50%, it shows over fitting of the model, I think the model stock in the local minima region, because of the step size is too small

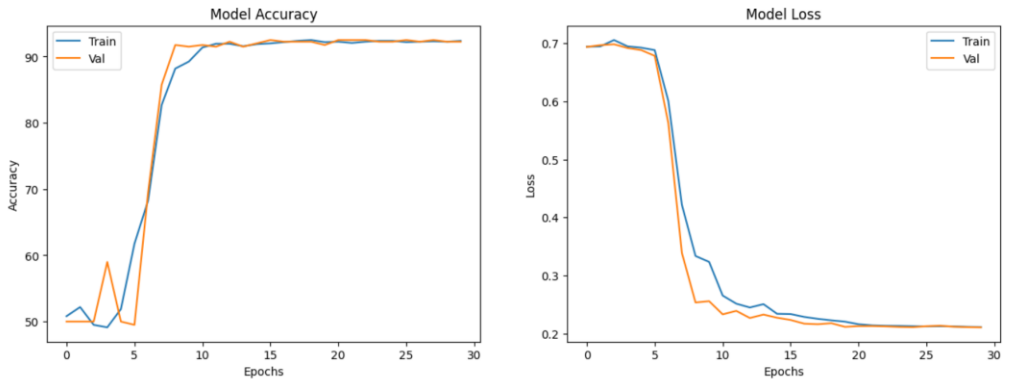
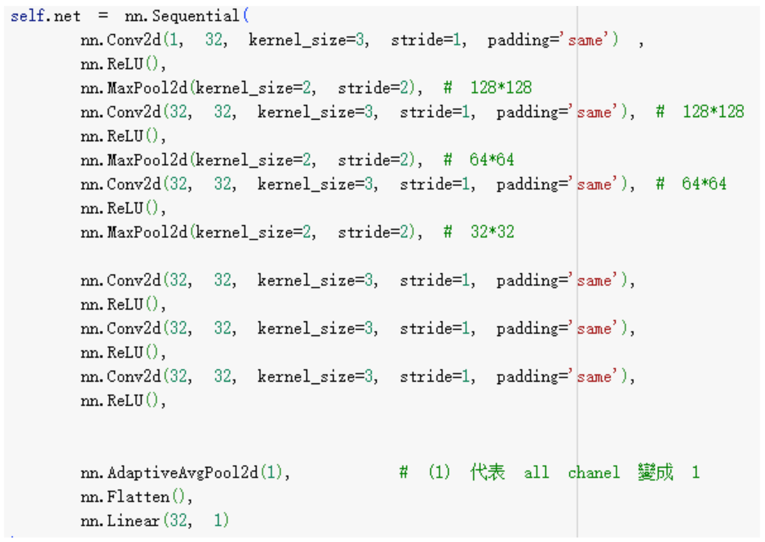


Fig. 9. The test accuracy is 75.5%, train accuracy is 92.38% and the validation accuracy is 92.25%, it shows much better performance than before.



```
self.net = nn.Sequential(
        nn.Conv2d(1, 32, kernel_size=3, stride=1, padding='same')  ,
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2), # 128*128
        nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'), # 128*128
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2), # 64*64
        nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'), # 64*64
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2), # 32*32

        nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'),
        nn.ReLU(),
        nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'),
        nn.ReLU(),
        nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'),
        nn.ReLU(),


        nn.AdaptiveAvgPool2d(1),          # (1) 代表 all chanel 變成 1
        nn.Flatten(),
        nn.Linear(32, 1)
```

Fig. 10. The code of the convGAP that I used to enhance the performance, I stack three more layer of convolution and active function, for the hidden layer has been eliminated, and the performance show much better.