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

SHIU-FENG CHENG[1]

[1] *Brain Research Center, National Tsing Hua University, Hsinchu 30013, Taiwan.*

Student ID:111080527

1. Introduction

For image processing convolution neural networks (CNNs) are usually adopted instead of artificial neural networks (ANN). This report aims to get a deeper understanding of CNN models. The three tasks included are A: overfitting of CNN, B: ANN and CNN comparison, and C: global average pooling function.

2. Result

2.1 TaskA: Reducing Overfitting in Convolution Neural Model

The issue to be solved in this section is the overfitting that occurs in the given convolution neural network (ConvModel in lab 4). The method I tried out here is applying L2 regularization to the loss function. The adjusted loss function is shown in the following equation where $Loss'$ is the new loss function of the system while $Loss$ is the original loss function (BCEloss, in this model). The λ term is where L2 regularization happens, the w_n represents the weights in the network.

$$Loss' = Loss + \lambda \sum_n {w_n}^2$$

Considering the update of the weight with the new loss function with L2 regularization we get:

$$w_i^{t+1} = w_i^t + \frac{-\eta \partial L'}{\partial w_i} = (1 - 2\eta\lambda) w_i^t + \frac{-\eta \partial L}{\partial w_i}$$

The η here represents the learning rate and w_i^{t+1}, w_i^t respectively represents the updated and current synaptic weights. In this form, you can find that the weight was first decayed before it followed the gradient. Thus, the L2 regularization is also called weight decay. The way I induce L2 regularization is by adjusting the weight decay of the optimizer function.

$$weight\ decay = 2\eta\lambda$$

```
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=0.01)
# lr_scheduler = CosineAnnealingLR(optimizer, T_max=len(train_loader)*epochs, eta_min=0)
lr_scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
```

The loss of the model with and without weight decay during training is shown in Fig. 1a. To analyze if the weight decay adjustment helped avoid overfitting, I did a linear regression between average loss and the training epochs for the 31~50 epochs (fig.1b). You can find that the validation loss of the model with weight decay is continuously decreasing (slope < 0) while the model without weight decay has increased loss (slope > 0) validation loss. The other thing that can be observed in this result is the comparison between the average loss of each group. First, both train loss and validation loss with weight decay converge to a higher value compared

with the loss without weight decay. This difference makes sense because weight decay gives an additional positive L2 regularization term. Second, the difference between training loss and validation loss is decreased after including weight decay. The less difference suggests that the trained model has less bias (the performance on training and validation is more similar).

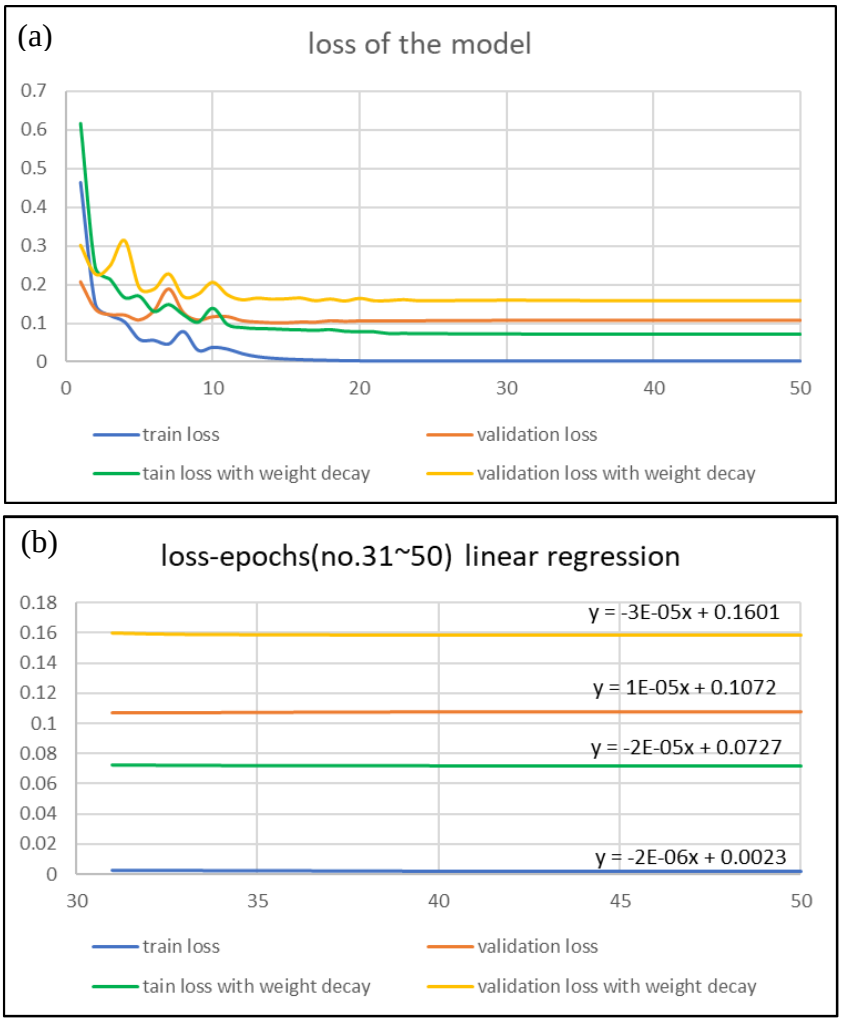

Fig. 1. The average loss of the model with and without weight decay. (a) the average loss during training in the whole training process (b) Linear regression on the 31~50 training epochs to analyze overfitting.

2.2 Task B: Performance Comparison between ANN and CNN

The structure of the two neuron network that is compared here is shown in Fig. 2a. The ANN model has 3 layers of 32 linear neurons and a linear neuron output layer. The activation function of each layer connection is the ReLU function. The CNN model consists of 3 2D convolution layers with 32 kernels (3*3) each followed by ReLU activation function and 2*2 max pooling. Then the 32 output 2d tensors are flattened and forward to a 32 linear neuron layer and a 1 neuron output layer. The loss function of both models are binary cross-entropy function. The number of variables in the ANN and CNN model are respectively 2099329 and 1049505.

The performance is shown in Fig. 2b, c. I found that although CNN has fewer variables, it has performance even better than the ANN model. This indicates that convolution and max pooling are a more efficient way to extract image features compared to fully connected NN.

Though CNN has fewer variables, training and image processing with it is slower. In my experiment training CNN with 1600 images and 50 epochs take 109.3sec and validation with 400 images take 10.6sec. while ANN take 16.4sec and 2.8sec. This suggested that either image processing (convolution and pooling) with CNN or updating weight (gradient calculation) spent more time compared to the linear computation of ANN.

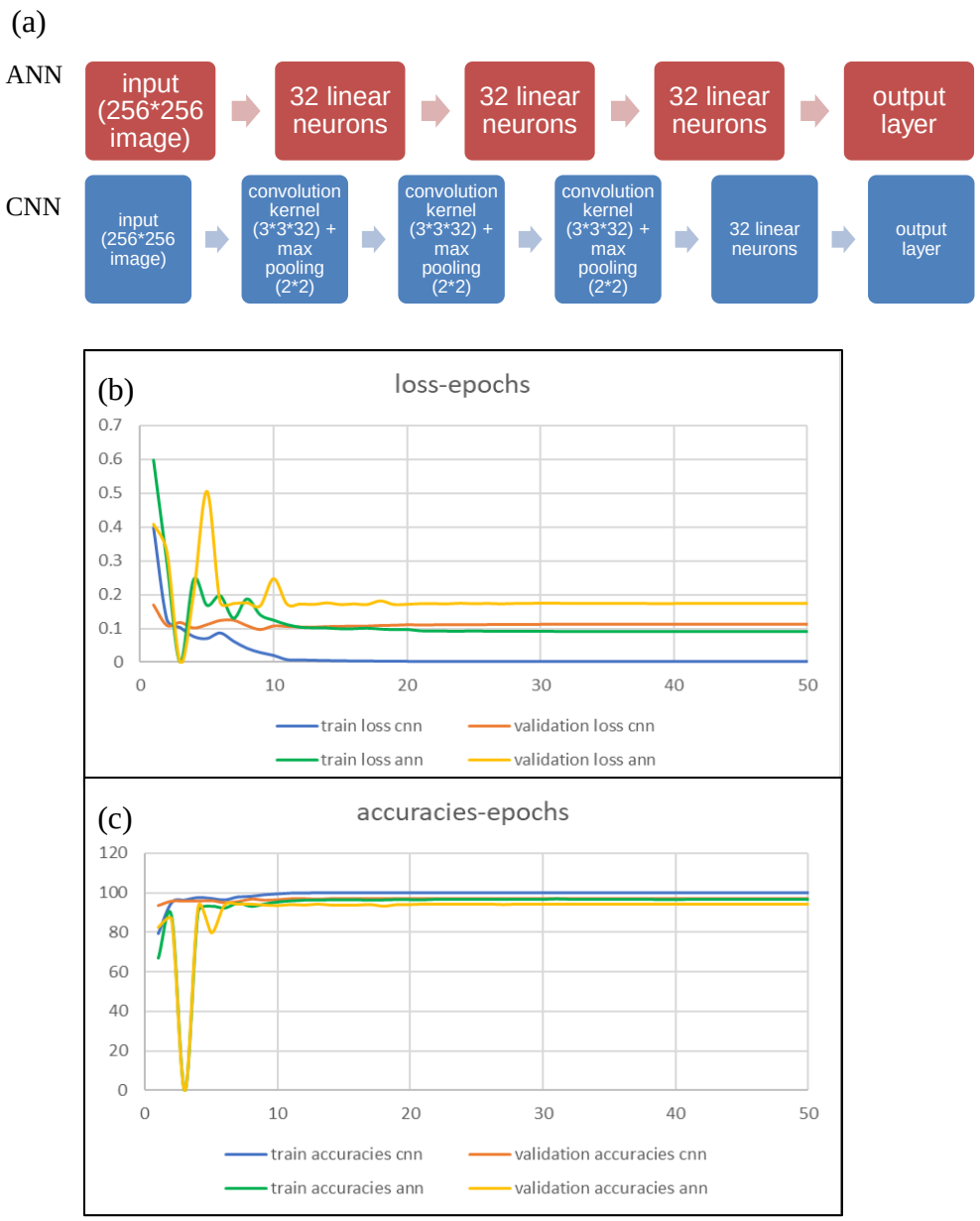

Fig. 2. Task B: (a)The neuron network structure used in the task B. (b, c) the performance of the ANN (green, yellow) and CNN (blue, orange) model

2.3 Task C: Global Average Pooling in CNNs

In this task, I'm going to discuss the global average pooling (nn.AdaptiveAvgPool2d) function in the ConvGAP model of lab4 code. What global average pooling does is output an averaged tensor with a given dimension. Fig.3a is a schematic diagram of global average pooling. Because the output of nn.AdaptiveAvgPool2d has the given dimension, it can be applied without manual dimension calculations. The ConvGAP model has nn.AdaptiveAvgPool2d(1), thus every (32*32) image plane only generates an average number. The reason for the performance drop may be that the average of the 32*32 pixels is not that representative. According to this hypothesis, I adjust every max pooling layer setting so that the tensor size gets 1/4*1/4 (from 1/2*1/2) of the input. The actual code modification is:

```
nn.MaxPool2d(kernel_size=2,  stride=2) → (kernel_size=4,  stride=4)
```

With this modification the input dimension of the global average pooling layer becomes 4*4, thus the output of average pooling is more representative of the input compared to the origin model.

The performance of the origin model and the modified model is shown in Fig. 3b, c. The training and validation accuracies of the modified model are enhanced to 91.9% and 92.3% respectively (from 83.9% and 85.8%). Also, both training loss and validation loss of the modified model decreased by ~0.23. The result suggests that the strategy I adopted is effective in resuming some amount of performance drop while using global average pooling.

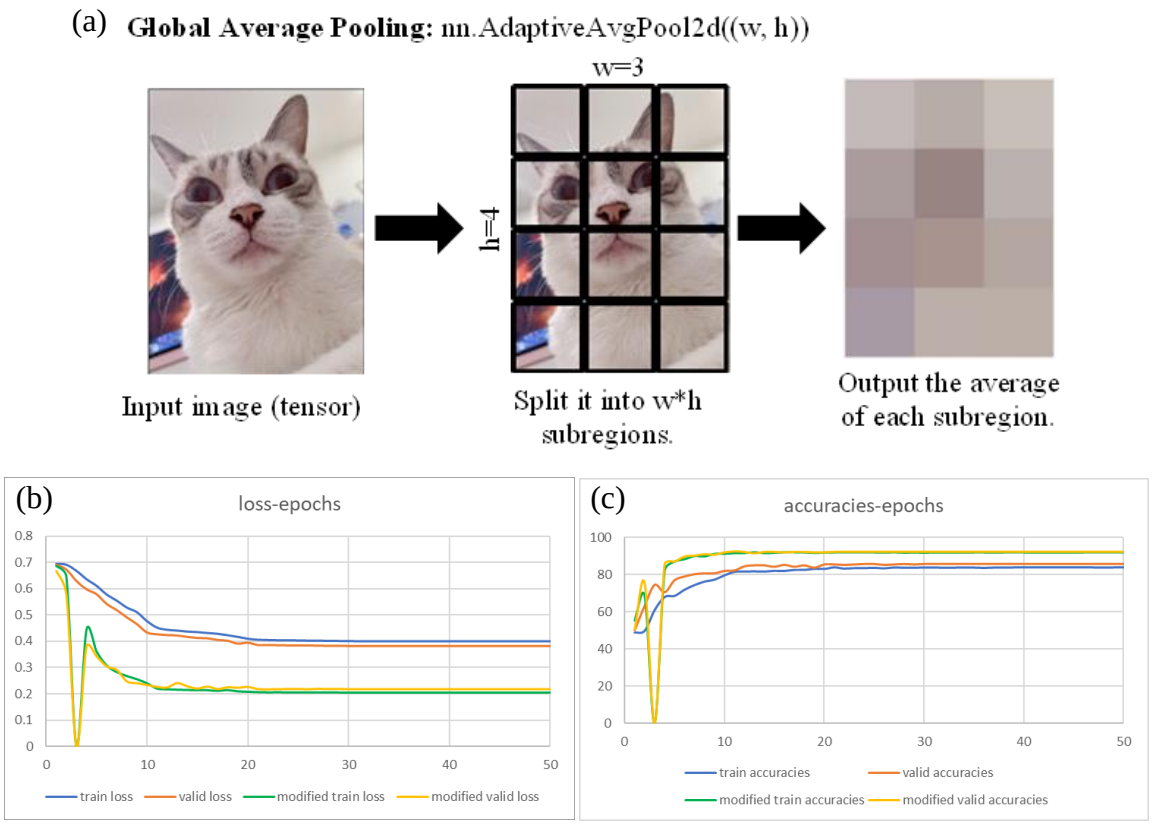

Fig. 3. Task C: (a) The schematic diagram of global average pooling. (b, c) Performance of the origin ConvGAP model (blue, orange) and the modified model (yellow, green). The modified model has every max pooling layer modified so every 4*4 pixel gives a maximum value (the original setting: 2*2 pixels gives a value).