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

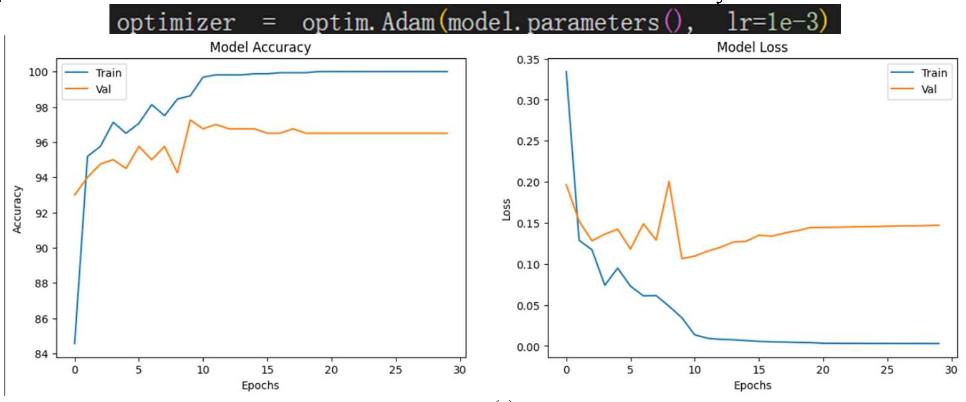
WEN CHI CHEN[1]

[1]*National Tsing Hua University, IPT, HOPE Lab.*

Student ID: 110066515

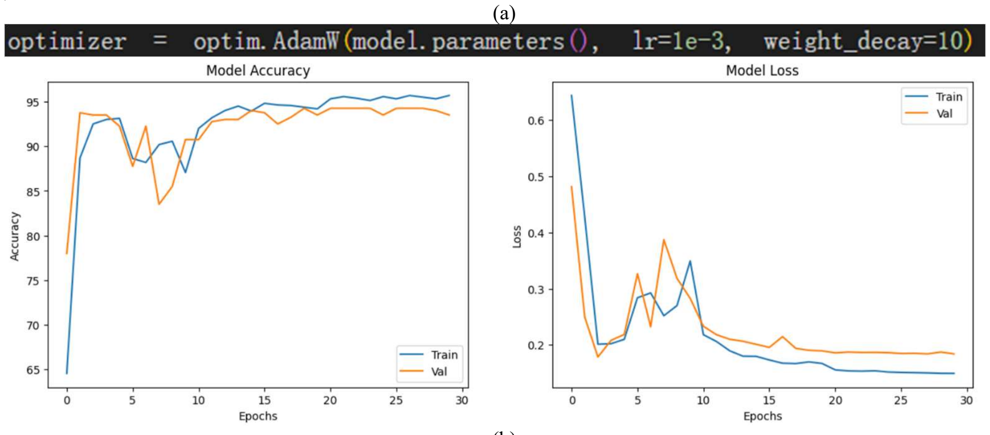
1. Reduce Overfitting

Reducing overfitting in deep learning models is crucial to ensure good generalization performance on unseen data. There are some methods: dropout of neurons, penalty on weights by regularization, batch normalization on layers, early stopping on training. In my analysis, I would like to discuss the effect of overfitting reduction by weights regularization. Following Fig. 1 is the unmodified & modified codes and their metrices of accuracy and loss.



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

(a)



```
optimizer  =  optim.AdamW(model.parameters(),  lr=1e-3,  weight_decay=10)
```

(b)

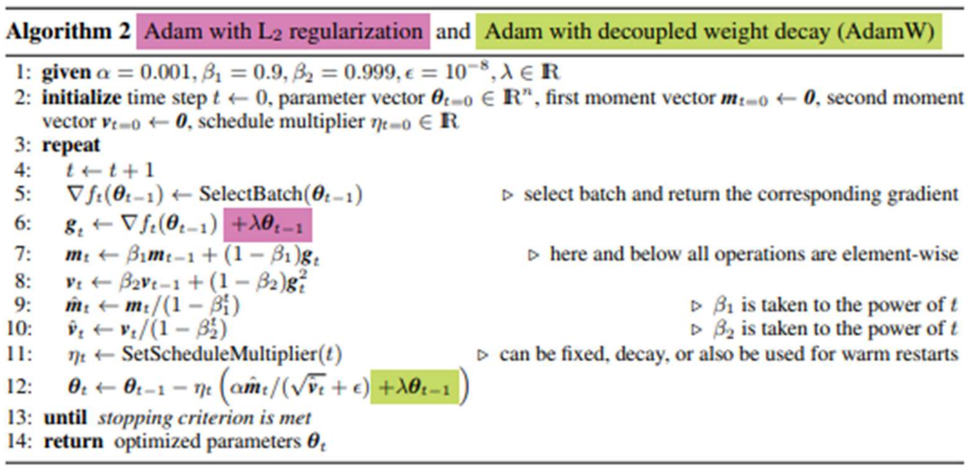
Fig. 1. (a) Original optimizer provided by TA, (b) implemented optimizer to reduce overfitting. The curves represent their metrices of training accuracy and model loss.

From Fig 1, the optimizer is modified from Adam to AdamW, and the weight decay of the optimizer is modified from 0 to 10. The convergence speed of modified model is much slower than that of before due to the effect of weight decay. The penalty of weight makes the model converge slowly compared to that of before. The test accuracy is slowly increasing while running out of all epoch; the loss is still decreasing as well. However, the original model converges in an incredible speed: the training accuracy reach 100% around 18th epoch and the loss are close to 0.00 at the same iteration. Note that in Fig 1(a), the validation loss curve reveals that the validation loss can be improve if there is early stop at around 10th iteration; therefore, the original model must be over-trained. As for the test result, the test accuracy improves from 71.15% to 78.5%.

The motivation that drives me to find the difference of weight decay is due to its difficulty for me, making me want to understand the importance of optimizer.

In general, Adam is an optimizer based on momentum. Momentum physically means "maintaining inertia". In the optimization of deep learning, it is to make the optimization process "memorable" and inert. inertia gives the past direction and speed. In Adam, "Ada" means adaptive. "m" is momentum. Adaptive weighting of past momentum in Adam prevents us from being overly influenced by the past, so the current gradient direction and the historical results are weighted to calculate by exponential smoothing (RMSprop optimizer). In short, the closer the gradient direction is to the present, the greater its influence on the weight.

To add penalty on weights (weight regularization), there is an inner parameter of Adam: weight_decay. In general, weight decay (L2 regression) uses the same value to update all weights. The gradients corresponding to larger weights will also be larger, leading to larger penalty, which is contradiction to the system of Adam. Since Adam will adjust the step size by dividing it by the accumulation of the square of the gradient, this causes the actual penalty and weight decay of the originally large weights to be different. Therefore, based on [1] (Fig. 2), AdamW adjusts the position of the regularization term to not interact with the momentum parameter This is equivalent to adding the square of the weights to the loss with plain (non-momentum) SGD. Add weight decay at the end.



Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

```
1: given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
2: initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment
   vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3: repeat
4:     $t \leftarrow t + 1$
5:     $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow$ SelectBatch($\boldsymbol{\theta}_{t-1}$)        ▷ select batch and return the corresponding gradient
6:     $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
7:     $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$        ▷ here and below all operations are element-wise
8:     $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
9:     $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1 - \beta_1^t)$        ▷ $\beta_1$ is taken to the power of $t$
10:    $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \beta_2^t)$        ▷ $\beta_2$ is taken to the power of $t$
11:    $\eta_t \leftarrow$ SetScheduleMultiplier($t$)        ▷ can be fixed, decay, or also be used for warm restarts
12:    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t/(\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) +\lambda\boldsymbol{\theta}_{t-1}\right)$
13: until *stopping criterion is met*
14: return optimized parameters $\boldsymbol{\theta}_t$
```

Fig. 2. [1] Weight decay and loss-based gradient updates in Adam.
Purple: Adam with its weight decay. Green: AdamW with its weight decay.

2. Performance Comparison between CNN and ANN

The accuracy & loss metrices performance of CNN is shown in Fig 1(a), and that of ANN is shown in Fig 3. First of all, the ANN model converges slightly slower than CNN model: the train accuracy converges at 97% and the loss at 0.15. Those value are not so extreme compared to that of CNN (100% & 0.001 in Fig 1.). However, it doesn't mean that ANN is fitted better, the test result is 73.25%, similar to the result of CNN: 71.15. Therefore, the ANN model in this report is still overfitting: the trained models are both so matched to the training set that cannot used in other test sets. The second comparison between CNN & ANN is running time. In Fig 4, there are the running time of these two models. Running time of ANN is about 10 sec, which is much smaller than that of CNN, 70 sec. This indicates that the running speed of ANN is much faster than CNN. The reason of running speed is the third comparison: feature extraction capabilities. It seems that the CNN model is more complex and complicate, because it uses kernel convolution to capture the feature of image characteristic, unlike the process of flatten the image data in ANN. The convolution increases the running time and also the captured feature. The detail comparison will be shown in below.

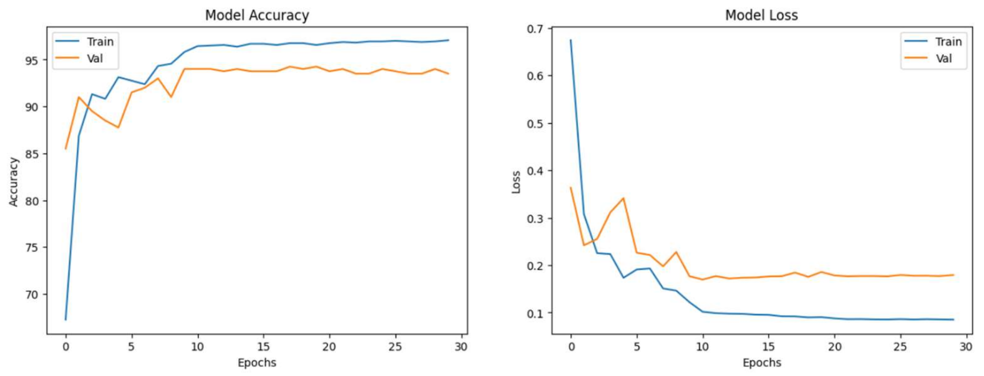
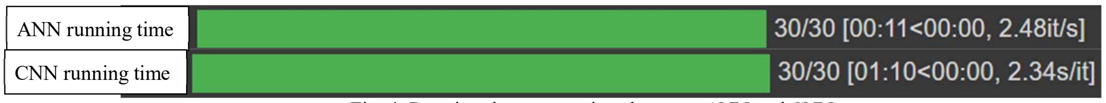


Fig. 3. Model accuracy & model loss metrices of ANN (linear model)



ANN running time 30/30 [00:11<00:00, 2.48it/s]

CNN running time 30/30 [01:10<00:00, 2.34s/it]

Fig. 4. Running time comparison between ANN and CNN

The architectures of the CNN are shown in Fig 5. Here we are predicting digits based on the input image given, note that here the image is of dimensions height = 28 pixels, width = 28 pixels, and a depth of 1. Then take the input and convoluting with filters of size 5 x 5 thereby producing an output of size 28 x 28. The output dimensions conv1 defines the filters number, so the depth of depth of conv1 is 6, or we can call the channel number 6. Next, the pooling layers kernel and stride define the output size, here the 2*2 kernel max pooling with stride 2 will give us half the size of original image. Following conv2 and max pooling 2 repeat the process. The function of convolution and max pooling are described in course videos.

Finally, we flatten all the 4 x 4 x n2 to a single layer of size n3. Input them to a feed-forward neural network of n3 neurons having a weight matrix of size [4 x 4 x n2, n3]. Following hidden layer and the output layer is the regime of fully connected layer, which is exactly the architectures of the ANN.

Table 1 shows the characteristic of CNN and ANN. There is their pros and cons, features capturing an efficiency, etc.

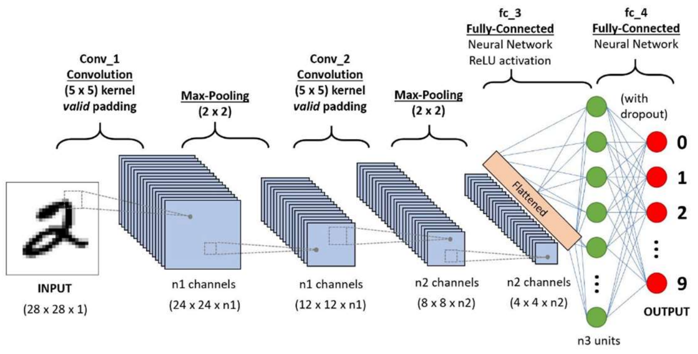


Fig. 5. The architectures of the CNN[2]

Table 1. ANN vs CNN

ANN	CNN
1. Use fully connected layers	1. Uses partially connected layers, size of connection depends on size of filter
2. can be used only for small image	2. can be used for any image
3. consider entire image for learning the patterns	3. consider only the part of the image, which is in the respective field of the filter.
4. number of parameters will be very high. For 28*28 image, and if the layer has 32 neurons, number of parameters for one layer will be 28*28*32 =25088. Consequently, computations involved in ALL are more.	4. number of parameters will be very less, compared to ANN. For 28*28 image, and if the layer has 32 filters, with each filter size as 5*5. Number of parameters for the layer will be (5*5*1+1) *32=832. Computation involved are less compared to that of ANN.
5. not very efficient for image inspite of being expensive	5. very efficient for images and very less expensive compared to ANN
6. doesn't learn the patterns in spatial hierarchy. i.e., Higher layers of ANN don't combine the lower layer outputs.	6. Learns spatial hierarchy of patterns. i.e., Higher layers of CNN are formed by combining lower layers. This helps in identifying the patterns more effectively than ANN.
7. Not translation invariant. Once a regular ANN has learned to recognize a pattern in one location, it can recognize it only in that particular location. In short, it we need to train the image again (can't reuse the previous weights) if it is rotated or shifted.	7. Translation invariant. Once the CNN has learned to recognize a pattern in one location, it can recognize it in any other location. In short, learning (weights) can be reused even if the image is rotated or shifted.

3. Global Average Pooling in CNNs

The role of GAP is to generate one feature map for each corresponding classification in the last mlpconv layer. MLP, multilayer perceptron, is suitable for CNN which use backward propagation. From its calculation, the first layer of MLP is a usual convolution. Each subsequent layer of MLP operation is equivalent to mixing and weighting the information of each channel and feature map of the previous layer. This is equivalent to performing Cascaded Cross Channel Parametric Pooling in the traditional CNN Conv Layer. Through layers of MLP layers, messages between different channels can be communicated again and again, thus enabling complex and learnable cross-channel information communication.

In traditional CNN, the value after the last convolutional layer is formed into a vector, and then fed into a fully connected layer as an input layer, and finally softmax is used for classification. The fully connected layer connects the convolutional structure with the traditional neural network, using the convolutional layer as feature extraction and classification. This method is very easy to produce overfitting.

Global Average Pooling abandons the traditional fully connected layer, outputs a number of feature maps equal to the number of categories in the last convolutional layer, and then takes the average of each feature map and outputs it into a vector then drop into softmax. This approach has several advantages: I. Enhanced the relationship between feature map and category. II. There are no parameters in GAP that need to be optimized to avoid overfitting. III. GAP combines spatial information and is more robust to spatial transformation of input features.

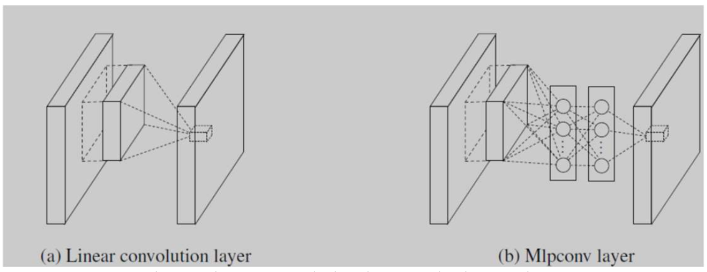


Fig. 6. Linear convolution layer and mlpconv layer

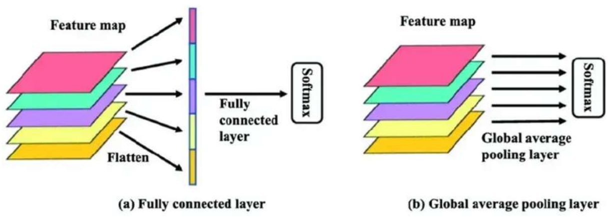

Feature map
Flatten
Fully connected layer
Softmax
(a) Fully connected layer
Feature map
Softmax
Global average pooling layer
(b) Global average pooling layer

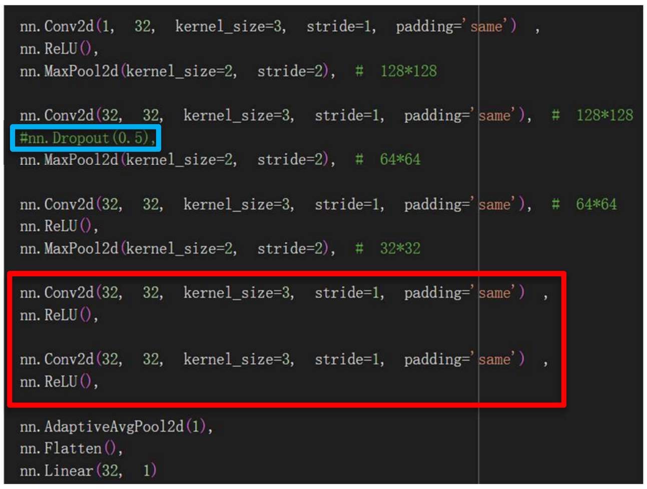
Fig. 7. Simple diagram of GAP

The key function that GAP can eliminate the need for manual dimension calculations is:

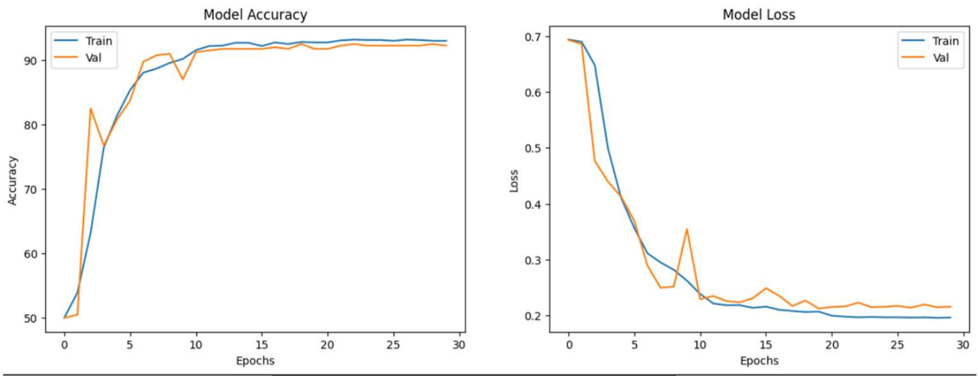
nn.AdaptiveAvgPool2d(1)

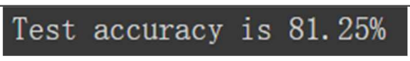
This function applies a 2D adaptive average pooling over an input signal composed of several input planes. By averaging over the spatial dimensions, GAP eliminates the need for manual calculation of spatial dimensions in subsequent layers. The output is of size H x W, for any input size. The number of output features is equal to the number of input planes.

The input and output characteristics feature of the function makes GAP has several features (from GPT): Dynamic Input Size, Simplified Network Design, Scalability, Reduced Sensitivity to Input Size.



```
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.Dropout(0.5),
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.AdaptiveAvgPool2d(1),
nn.Flatten(),
nn.Linear(32,  1)
```





```
Test accuracy is 81.25%
```

Fig. 8. Modified GAP code (add the content in the red box, eliminate the content in the blue box) and its accuracy improvement.

From GPT, it told me that to improve the performance of GAP, I can adjust network depth, fine-tune hyperparameters, add more convolutional layers, etc. for me the most understandable method is “add more convolutional layers”. Therefore, I just put 2 more Conv2d() cascade with ReLU() in the model. But I found that the result still run not singular (train acc~80%, test acc~65%).

Then I do remember Dropout() is a method to reduce overfitting; however, my model is now absolutely underfitting, so I eliminate the Dropout() in model. Finally, the result gets a boosting improvement, as shown in the accuracy & loss metrices in Fig 8. The train & validation accuracy is ~92%, and the test accuracy is 81.25%, meanwhile, the curves of train & validation metrices looks consistent, which is much better than the original curves.

References

1. Decoupled weight decay regularization, Ilya Loshchilov & Frank Hutter, University of Freiburg, Germany.
2. Convolutional Neural Network Architecture | CNN Architecture (analyticsvidhya.com)
3. What are the differences between Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN) - Stack Overflow
4. https://aitechtogether.com/python/62151.html#google_vignette