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

JUNQIANG-TANG

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

Student ID: 110066544

1. Abstract

There are mainly three sections in this project. First, I try to reduce the overfitting by adding two dropout layers after the fully connected layers. Second, with different type of neural networks, Artificial Neural Network (ANN) and Convolutional Neural Network (CNN), I did not see a dramatical difference for the performance. But for the training speed, ANN is faster than CNN in this code. Third, I explained what role Global Average Pooling(GAP) plays in CNNs and added additional three layers into the model to increase the performance, which is increased from 66.5% to 81.75%.

2. Introduction

This code presents a deep learning-based solution for the classification of chest X-ray images into two simple categories, normal and pneumonia picture. With deep learning, we are going to use two different neural networks, Artificial Neural Network (ANN) and Convolutional Neural Network (CNN), to do the comparison and analyzation between each other and improve the performance in the end.

The dataset used in this code also consists of chest X-ray images with and without pneumonia. The primary objective is to explore and evaluate the performance of these two models in image classification. The ANN represents a traditional neural network architecture, while the CNN is designed for image data.

The code includes a detailed analysis of both model architectures, training procedures, and the integration of dropout layers to solve overfitting problems. Through this work, we will try to identify the strengths and weaknesses of each ANN and CNN models. By understanding the capabilities of ANN and CNN models, we can make a way for more accurate and efficient systems to do such a diagnosis.

3. Results

3.1 Task A: Reduce Overfitting

The figure 1 is the accuracy and loss plots for the Lab 4's original code. The technique I choose to reduce overfitting is to add dropout layers with dropout rate of 0.5 after the convolutional layers and fully connected layers in the 'ConvModel' class, which is shown in figure 2, and the adjusted results are shown in figure 1. In the beginning, we can see that during the training, the training loss continues to decrease dramatically while the validation loss plateaus. After adding dropout layers, it is obvious to observe that the phenomenon of overfitting has been not significant although the training accuracy is slightly larger than the validation accuracy.

Overfitting occurs when a model becomes too complex and starts to fit the background noise and random variations in the data. In deep neural networks, neurons in hidden layers can become highly specialized and can co-adapt to the noise in the training data. When dropout is

applied, it prevents co-adaptation by randomly deactivating neurons during training, which means that different sets of neurons are trained on different parts of the data. Thus, it encourages the network to learn more robust and generalized features. In my code, dropout layers are added after the first fully connected layer in the Convolutional Neural Network (CNN), this is where the model starts making high-level abstractions from the features extracted in the convolutional layers. Therefore, applying dropout at this stage could prevent the network from overfitting to these abstractions. So, it also improves the model's generalization.

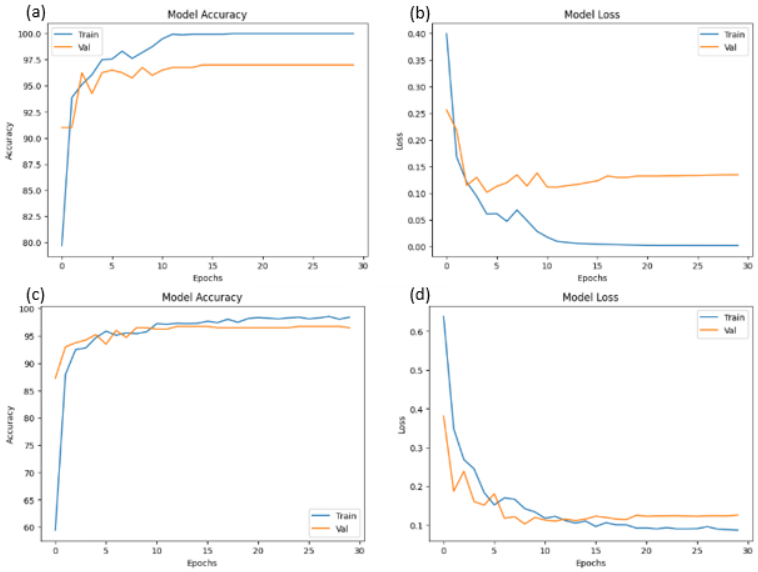

Fig. 1 (a) The plot of training and validation accuracy with original code. (b) The plot of training and validation loss with original code. (c) The plot of training and validation accuracy with adjusted code. (d) The plot of training and validation loss with adjusted code. Blue line is training curve and orange line is validation curve.

```
        # Add dropout layers
        self.dropout1 = nn.Dropout(0.5)    # You can adjust the dropout rate as needed
        self.dropout2 = nn.Dropout(0.5)    # You can adjust the dropout rate as needed

    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))

        # Apply dropout after the first fully connected layer
        x = self.dropout1(x)

        return self.fc2(x)
```

Fig. 2 The section I adjust the code. (The red rectangle is dropout layers.)

3.2 Task B: Performance Comparison between CNN and ANN

For my ANN model, the input layer takes the images with a size of 256*256 pixels, which are flattened into a 1D tensor of 256*256*1=65536 input nodes. Then, there are three hidden fully connected layers in ANN. Three of them all have 32 neurons with ReLU (Rectified Linear Unit) activation. And the output layer is a fully connected layer with a single neuron. It uses a linear activation function to generate continuous values, which represents the model's prediction for binary classification.

For the CNN model, the input layer also takes images of 256*256 pixels. Similar from above, there are three convolutional layers inside and each of them follows a max-pooling layer. All convolution layers have a kernel size of 3*3 and be applied with ReLU activation. Then, two fully connected layers come out. The features are extracted by the convolutional layers are flattened before entering the first fully connected layer. As for output fully connected layer, there is a single neuron with a linear activation function. Here I add two dropout layers after the first fully connected layer to prevent overfitting.

The results from ANN (LinearModel) and CNN (ConvModel) are the accuracies of 75.25% and 76.75%, respectively. It looks like the performance between these two different models do not have much different (although overfitting occurs in ANN), but the training speed has a significant difference. The overfitting maybe mean that CNN has a better feature extraction capability. The one for ANN only costs me around ten seconds, while the training speed for CNN needs more than one minute. However, as far as I know, CNNs are known for their faster convergence and efficient training on image data, and ANNs can be slower to train because of the large number of parameters, which will cost longer training time to reach convergence. In my code, the CNN model includes convolution layers, pooling layers, and fully connected layers, which makes this model more complex than ANN. For my ANN model, it is a relatively simple feedforward network. That why my CNN is taking longer to train compared to the ANN.

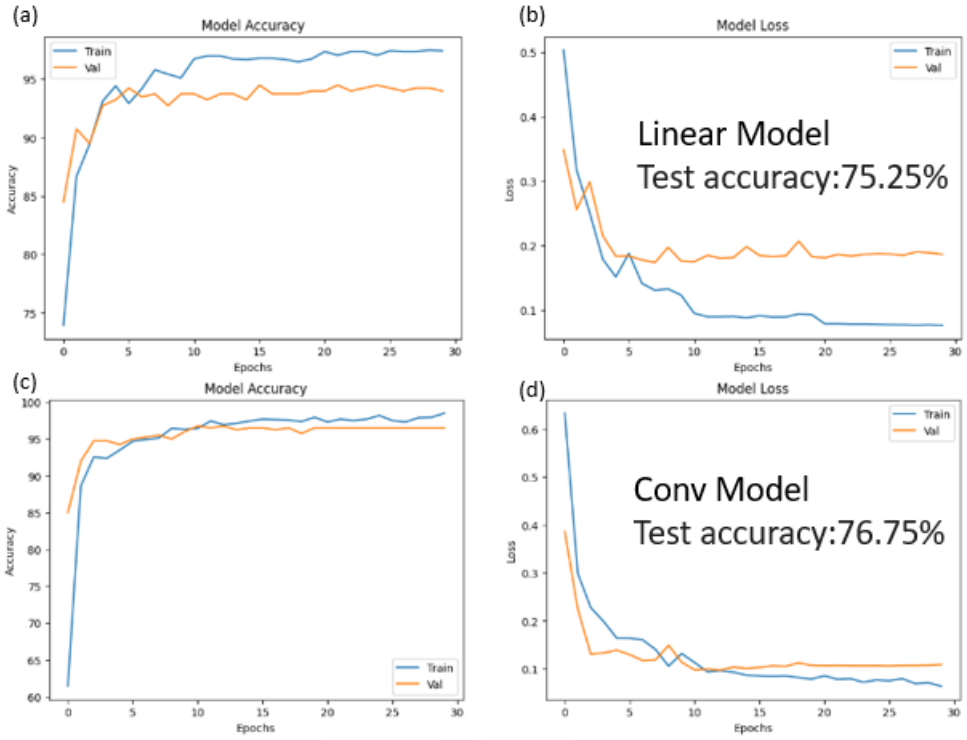

Fig. 3 The plots of accuracy and loss with different Neural Network. (Above is ANN and below is CNN.)

Fig. 4 The training speed of CNN and ANN. (Above is CNN's result and below is ANN's result)

3.3 Task C: Global Average Pooling in CNNs

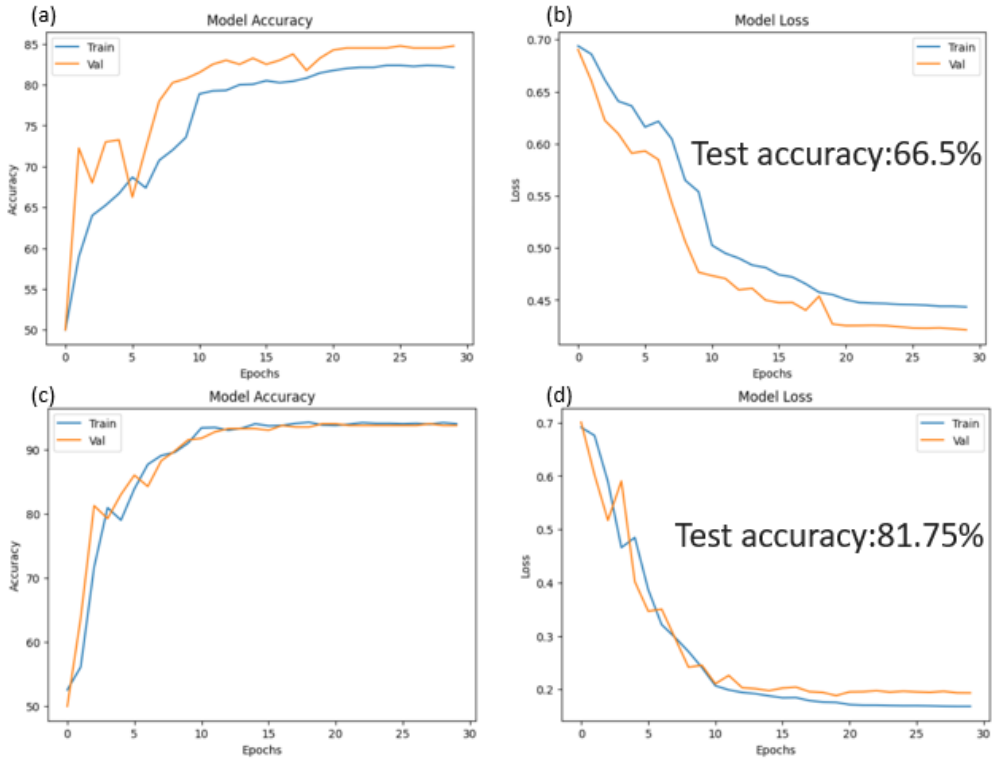

Fig. 5 The plots of accuracy and loss before and after improvement.

Global Average Pooling plays a role of reducing the spatial dimensions of the feature maps to create a compact and fixed-size representation, so which is used for feature extraction and dimension reduction in CNNs. It can convert a spatial grid of values into a single value for each output of a convolutional layer, thus this technique can replace the familiar FC (fully connected) layer and reduce overfitting. GAP eliminates the need for manual dimension calculations because it automatically adapts to the size of the input feature maps.

The way I improve the performance is to add more layers in the GAP model. Originally, the performance is quite poor, which is a test accuracy of 66.5%. After adding three more layers inside the model, the test accuracy is increased to 81.75%. By doing this, the model is allowed to learn more complex and abstract representations of the input data. I also tried to add more layers, however, the result shows that overfitting occurs. Therefore, it is important to strike a balance between model complexity and data availability when adding more layers to the model to avoid overfitting and ensure better generalization to unseen data.

```
class ConvGAP(nn.Module):
    def __init__(self):
        super().__init__()

        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'), # 64*64
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'), # 64*64
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'), # 64*64
            nn.ReLU(),



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

Fig. 6 By adding more layers in the model can improve the performance. (Inside the red rectangle is the code I added.)