National Tsing Hua University
Fall 2023 11210IPT 553000
Deep Learning in Biomedical Optical Imaging
Cancer Histology Image Classification

JUNQIANG-TANG

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

Student ID: 110066544

1. Abstract

The purpose of this project is to clearly identify six types of historical images of cancerous tissues. During the process, I used the method of transfer learning to achieve this task. I chose a pre-train model, GoogleNet, as my model. With this, it got a test accuracy of 84.67%. It has already been a suitable model to do this classification. To improve the performance, I do the fine-tuning process by adjust the learning rate and the number of epochs. Then I also added additional hidden layer to see whether the performance is better, which contains two batch normalizations before two ReLU functions to avoid overfitting. In the end, with learning rate of 0.005 and 50 epochs, the best performance I got is 95.50%.

2. Introduction

In this report, we are going to accurately classify the histological images of cancerous tissues. There are six distinct tissue textures in our dataset, which contains tumor, stroma, complex, lympho, debris, mucosa. For each sample in the dataset, it is a 150*150-pixel RGB image, which represents one of six types of tissue textures. Figure 1 shows some sample from the dataset.

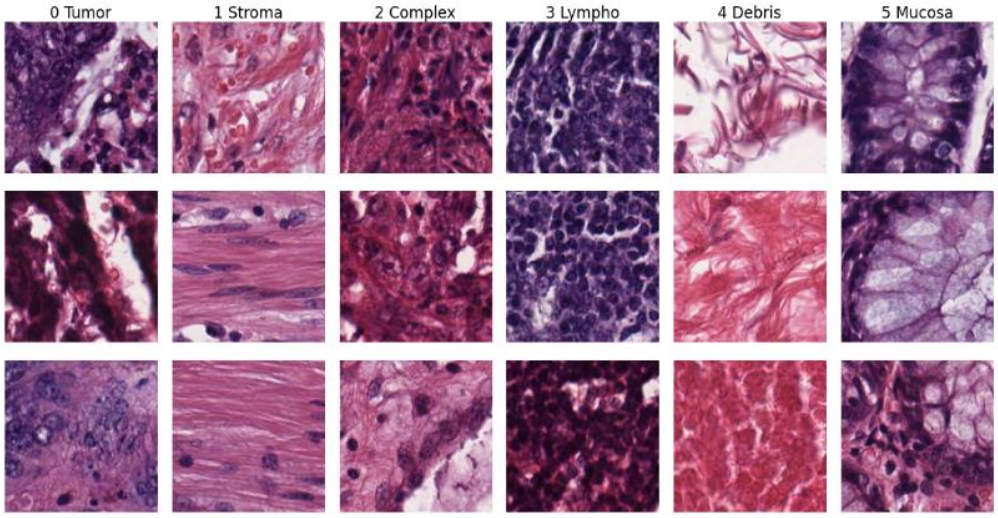

Fig. 1 The pictures of six different tissue textures from dataset

3. Result

3.1 Model Architecture

I employed the pre-train GoogleNet model and modified its fully connected layers to suit my classification task. Here, I added two hidden layers inside the model and two batch normalizations before each ReLU function to avoid overfitting. At the same time, I did a fine-tuning for different learning rate and a larger epoch to make this pre-train model adapt our specific dataset and have a better performance. To do the fine-tuning, it is necessary to unfreeze the pre-train model by typing 'param.requires_grad = True', shown in Fig. 2. So that, it allows us to adjust model's parameters and execute it.

```
for param in model.parameters():
        param.requires_grad = True
```

Fig. 2 This change allows us to unfreeze the model and do fine-tuning.

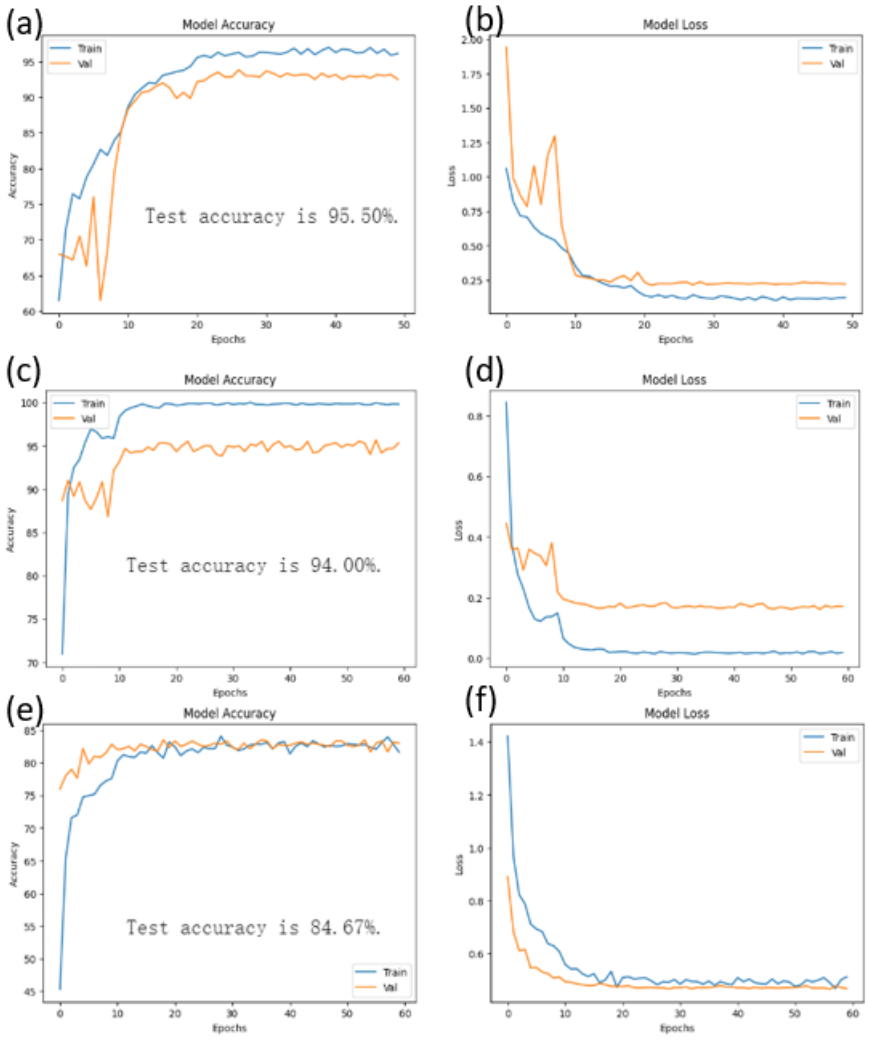

Fig. 1 (a) The plot of training and validation accuracy with learning rate of 0.005. (b) The plot of training and validation loss with learning rate of 0.005. (c) The plot of training and validation accuracy with learning rate of 0.0005. (d) The plot of training and validation loss with learning rate of 0.0005. (e) The plot of training and validation accuracy without fine-tuning. (f) The plot

of training and validation loss without fine-tuning. Blue line is training curve and orange line is validation curve.

3.2 Performance Comparison

Before fine-tuning, the test accuracy is only 84.67%, shown in Fig. 1. It is obvious to see that the performance is worse than the fine-tuning one. The best result with fine-tuning I got is 95.50%. Actually, when I increased the number of epochs from the original code to 50, which has already provided a better result than before. However, the test accuracy is just about 90%. In order to get a higher accuracy, I tried different learning rate to find out which learning rate could cause a best performance. From the result, it looks like that the learning rate of 0.005 is the best.

```
model.fc  =  nn.Sequential(
#model.classifier  =  nn.Sequential(

        nn.Linear(num_ftrs,  256),
        nn.BatchNorm1d(256),
        nn.ReLU(),
        nn.Dropout(0.5),
        nn.BatchNorm1d(256),
        nn.ReLU(),
        nn.Dropout(0.5),
        nn.Linear(256,  6),
)
```

Fig. 3 I add batch normalizations before ReLU function to avoid overfitting.