Individual Final Report

Yongchao Qiao

Introduction

Misdiagnosis of the many diseases impacting agricultural crops can lead to misuse of chemicals leading to the emergence of resistant pathogen strains, increased input costs, and more outbreaks with significant economic loss and environmental impacts. Current disease diagnosis based on human scouting is time-consuming and expensive, so computer-vision based models can help to relief this. And our purpose for this project is to use the knowledge we learnt from this class to solve the real world problems. In this project, the dataset of "Plant Pathology Challenge" from Kaggle are used to train the CNN model to accurately classify a given image from testing dataset into different diseased category or a healthy leaf. In detail, this problem is a one-label classification problem with four categories: "healthy", "multiple diseases", "rust" and "scab", which are four different possible situations where a given leaf image is probably under. And the CNN network based on PyTorch framework will be used in this project.

In this final project, we generally follow the format of Exam 1 and Exam 2, which means each member needs to build his own model individually. After several days we will exchange experience or ideas based on the test score of the Kaggle to get the best model, which will be used as the final model in our final project. During this final project, I got the highest score of the test set on Kaggle within our group, so my individual model is also the best model of our final project. Generally, the work I did this final project contains three parts: experimental setup which including data preprocessing, model constructing and parameter tuning, result and summary.

Description of my individual work

I. Background of Convolution Neural Network

In this final project, since my model got the highest test score within our group, so I was responsible for almost anything that related to the codes or models. In this project our model is Convolution Neural Network so some background information of Convolution Neural Network will be introduced as below. Like other neural networks, Convolution Neural Network contains several layers, where convolution layer and pooling layer are the most important layers. In this project I just play with these two layers so I will introduce these two layers first.

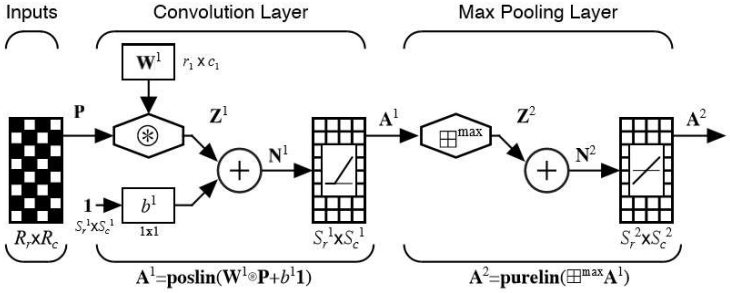


Figure 1: The convolution layer and pooling layer

The Convolution Layer

A convolution network is a multilayer feedforward network that has two- or three-dimensional inputs. In this project, I just use the two-dimensional inputs since the inputs are leaf images. The convolution layer has weight functions that performs a convolution operation on the image, using the convolution kernels, which is represented by the r × c matrix W. The input image is represented by the $R_r \times R_c$ matrix V. Then the output Z which is named as feature map will be calculated as :

$$z_{i,j} = \sum_{k=1}^{r} \sum_{l=1}^{c} w_{k,l} v_{i+k-1,j+l-1}$$

Also, in matrix form, it will be represented as: $\mathbf{Z} = \mathbf{W} \circledast \mathbf{V}$

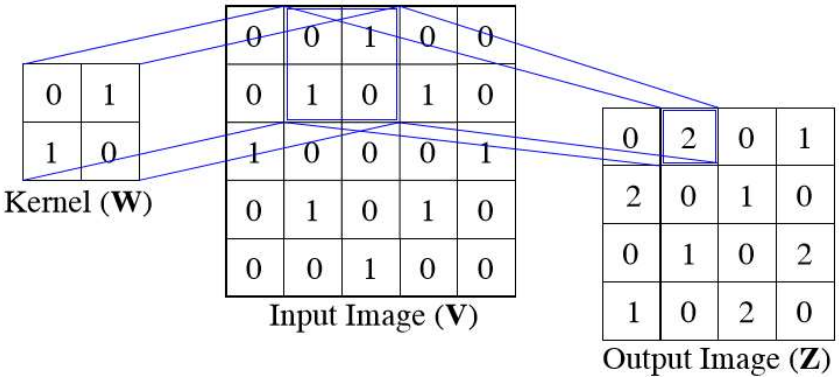


Figure 2: Convolution operation

The Pooling Layer

The pooling layer usually follows a convolution layer and consolidates r × c elements in the input image to 1 element in the output image. The purpose for the polling layer is to reduce the spatial size of the feature map, which will reduce the number of parameters in the network. Although there are several types of pooling methods, since in my model I only use the max pooling method, I will only introduce the max pooling method and average pooling method.

1. Max pooling method

In max pooling method, the maximum of the elements in the consolidation window is used, following the formula: $z_{i,j} = max\left\{v_{r(i-1)+k,c(j-1)+l} | k = 1, ..., r; l = 1, ..., c\right\}$, and the matrix form is : $\mathbf{Z} = \boxplus_{r,c}^{max} \mathbf{V}$

2. Average pooling method

In average pooling method, the mean of the elements in the consolidation window is used, following the formula: $z_{i,j} = \left\{\sum_{k=1}^{r} \sum_{l=1}^{c} v_{r(i-1)+k,c(j-1)+l}\right\} w$, and the matrix form is : $\mathbf{Z} = w \boxplus_{r,c}^{ave} \mathbf{V}$.

II. My detail work in the final project

Data preprocessing

1.Training part:

There are 1821 observations in the train set which is not that big. In order to make the model more robust, I decided to make the data augmentation. First, I downloaded the original train set: plant-pathology-2020-fgvc7.zip to the computer, uploaded it to cloud and read images as well as target with the one-hot encoding format. Then I saved images of original train set

into a tuple group by the matched file keyword, like “Train” and transformed the one-hot encoding target csv file into numpy array. There are total 1821 images, 1819 of whose size are (1365, 2048, 3) with 2 belonging to (2048, 1365, 3). So I transformed the 2 images in (2048, 1365, 3) to (1365, 2048, 3) first. Since the diseases pattern in these images are not that big, I resized them into one same size as (110, 164, 3) to avoid losing much information, fortunately this size was verified as the most suitable one for my model and the memory of cloud CPU and GPU which helped me train the model precisely. Based on the GPU memory, I tried many times and found that augmenting the original dataset to 10926 observations was suitable. That is, each image must generate 5 other new images by nine reasonable random methods like rotation within 30 degrees, horizontal flip, vertical flip, width_shift_range within 0.2, height_shift_range with 0.2, feature-wise_center(Set input mean to 0 over the dataset), feature-wise_std_normalization(Divide inputs by std of the dataset), brightness adjustment in the range [0.8, 1.2] and fill_mode as “reflect”. After this augmentation, I just changed the number of observations but not the original distribution since I thought the given imbalanced distribution is the true distribution. Some augmented images are shown as below.

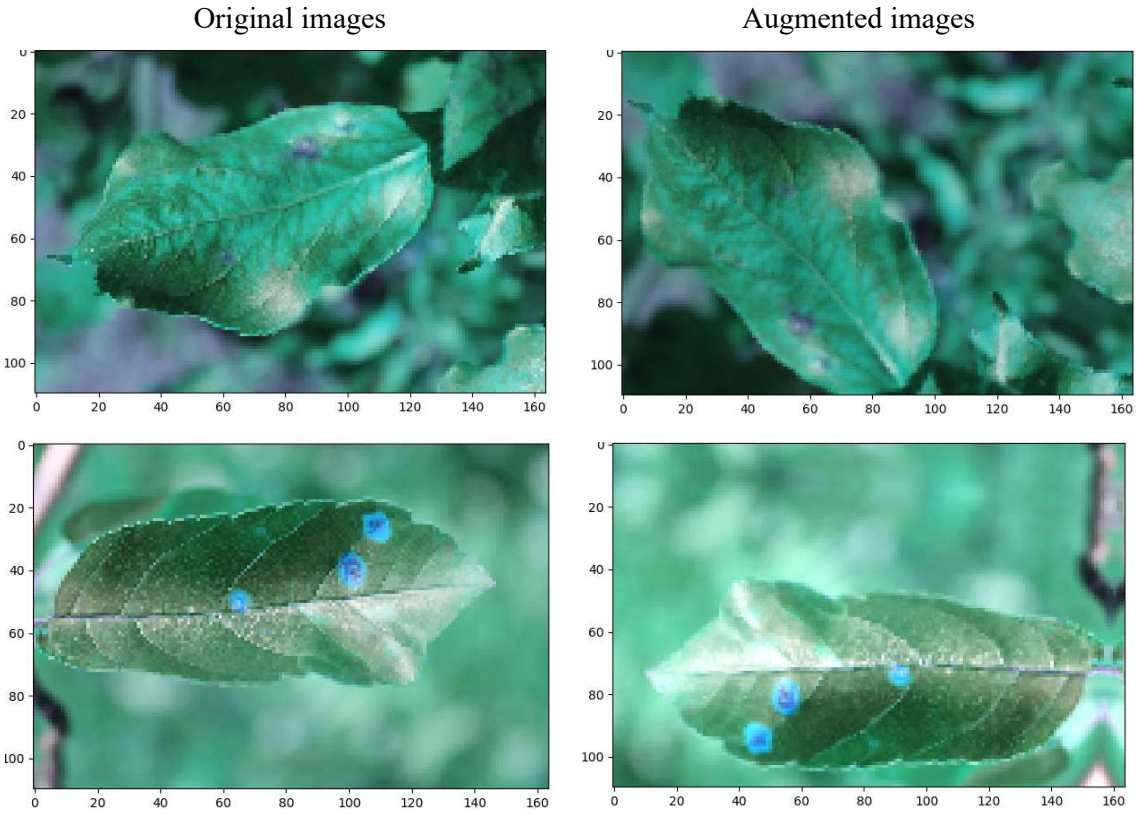


Figure 3: Original vs Augmented images

Also I needed to switch the one-hot encoding target into four numbers representing four categories as the strategy to split the original data set. After augmenting the data, I needed to split the augmented data into train and test set by the generated strategy. Here based on the memory of the GPU, the best randomly stratified separation is: train : test = 0.7 : 0.3 = 7648 : 3278.

2.Test part:

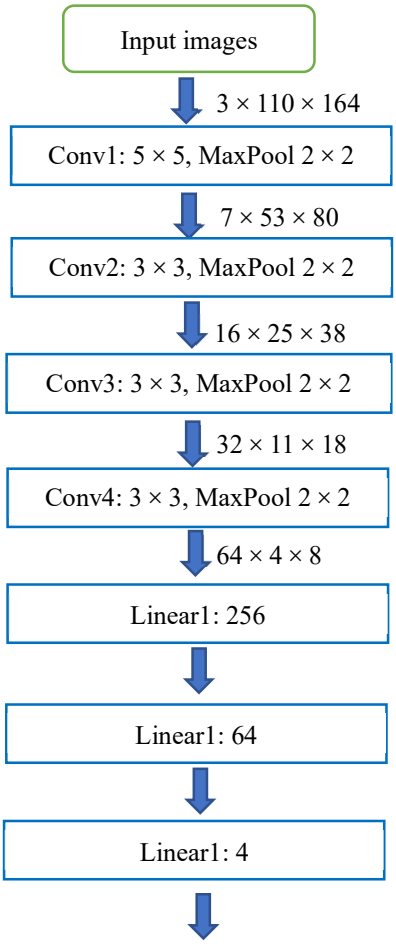
Just like the processing of training part, I read images and saved images of original train set into a tuple group by the matched file keyword, like “Test”. There are total 1821 images, 1801 of whose size are (1365, 2048, 3) with 20 belonging to (2048, 1365, 3). So I transformed the 2 images in (2048, 1365, 3) to (1365, 2048, 3) and then resized these test images into (110, 164, 3) to match the model input size.

Model construction

In this project, I have tried many convolution neural networks and played with the convolution layer and pooling layer as what I said, like more layers or less layers. Finally, I found one best single CNN model which is shown as below. This is a deep network containing 11 convolution layers and 3 linear layers, with activation function as “relu” and 14 batchnormalization layers. Also I will show an original convolution neural network that modified from Exam 2.

1. The original CNN

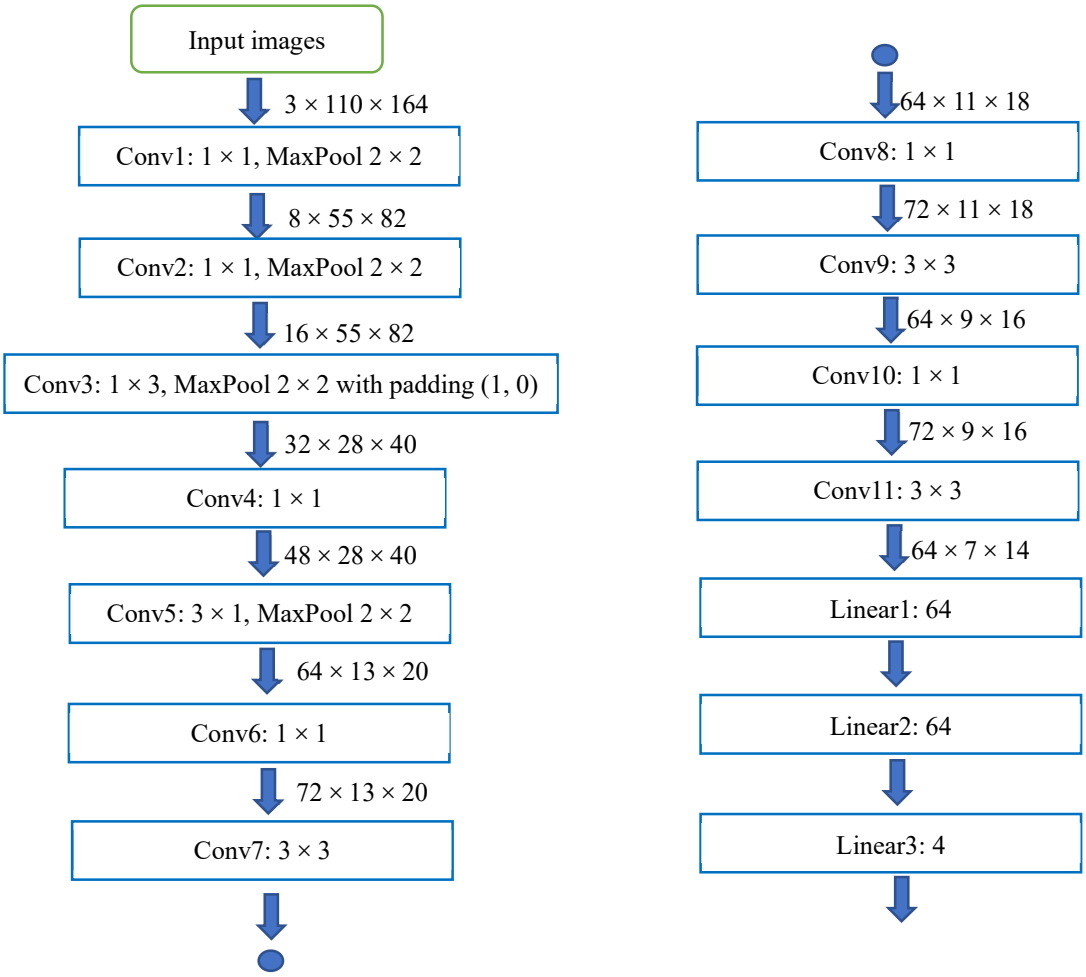
Figure 4: The original CNN



There are 4 general convolution layers and 3 linear layers in this CNN, also I add the batch normalization layers after each of these 7 layers. And the activation function is “relu”.

2.The best single model

Figure 5: The best CNN



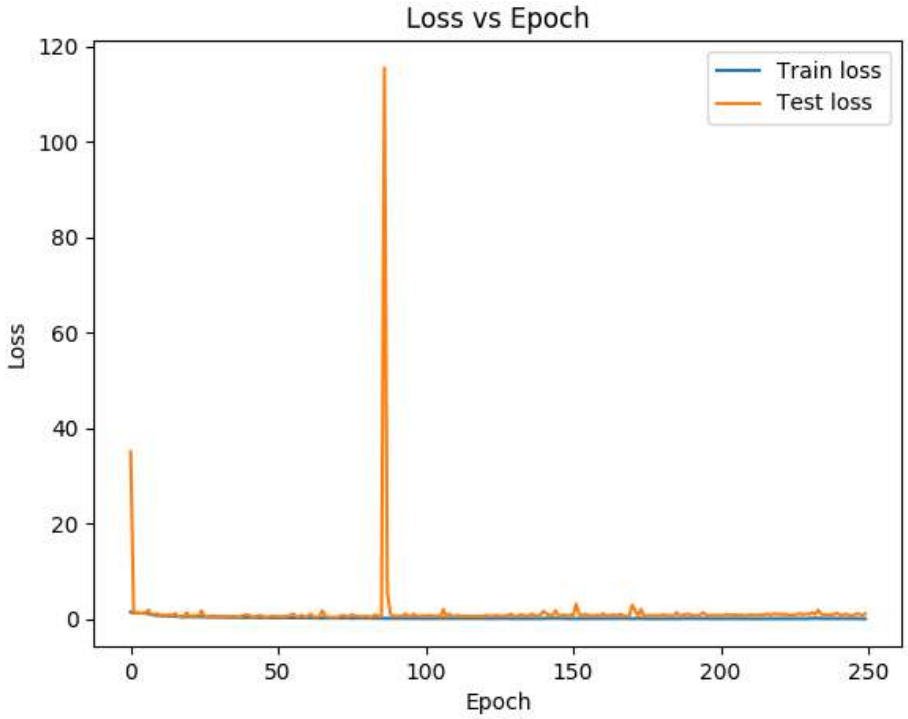
There are 11 general convolution layers and 3 linear layers in this CNN, also I add the batch normalization layers after each of these 14 layers. And the activation function is “relu”.

Parameter tuning

1. Training operations

I used the train set and test set which are split from the original train dataset, to train the model. I fed the model with train set and test it on the test set during the training procedure. The performance index is cross entropy loss but I will use the AUC score on the test set to check the performance of the model. Also, I added the early stopping operation to prevent the overfitting and save the model weight which own the highest AUC score during the training step. Besides, there are other methods that I used to prevent the overfitting, like the batchnormal layers in the model, dropout and specify the group parameters in the convolution layers. For detecting the overfitting, I just plot the train loss and test loss against the epochs as show below.

Figure 6: Detect the overfitting



From the plot, it is still hard to detect the overfitting pattern, but if we zoom the bottom part, there may exist overfitting pattern since the test loss has a slight upward tendency while the train loss is decreasing.

2. Minibatches

Based on the memory of the GPU, the max value of minibatch is 430 so I tried several different values of the minibatch size, like 200, 412, 430. The comparison is shown in the results part.

3.Learning rate

For the learning rate, based on the experience of Exam 1 and Exam 2, first I needed to check which form of the learning rate schedular is the best. So with other parameters being the same, I tried one static leaning rate as 0.1 for the whole training loop and one dynamic learning rate as cosine annealing schedular with max value as 0.1. Then I found the dynamic learning rate is better than the static learning rate so I began to tuning the specific value of the max learning rate in the cosine annealing schedular. Besides, I also tried other dynamic learning rate schedule like, ReduceLROnPlateau, CosineAnnealingWarmRestarts and so on. However, for my best model the best dynamic learning schedule was cosine annealing learning rate schedular so I decide to tune the max value of it. The comparison will be shown in the results section.

4. Initial weight method

In this project, I did not use the default initial weight for my CNN model but checked four different initial weight methods. They are xavier_normal, xavier_uniform, kaiming_normal and kaiming_uniform. For initial biases, I just let them follow the Norm(0, 0.02) distribution

which is also the experience from Exam 2. Then the comparison is shown in the result section.

5. Optimizers

In this project, I only tried Adam and SGD, since SGD is the first optimizer I learnt from this class and Adam is a quite fast optimizer. Also, the comparison is shown in the results section.

6. Image size

In this project, I tried 2 types of image size: (110, 164, 3) and (82, 120, 3). To make the results comparable, I changed the image size and rebuilt the model to match the input size with other parameters being the same. And the comparison is shown in the results section.

7. *Original CNN and deeper CNN*

In this project, I tried many types of convolution neural network. For example, the original CNN and the best single model as a deeper one. To make the results comparable, I only changed the model with other parameters being the same. And the comparison is shown in the results section.

8. *Ensemble model*

In this project, I tried to combine different model weights with high AUC score to get the highest AUC score on the Kaggle test set.

9. Confusion matrix and classification report

Based on the confusion matrix and the classification report, I may find the problem in the trained model.

III. Results

Minibatch comparison

The test AUCs for minibatch comparison on the Kaggle are shown in table 1, which indicates that the there is no obvious relationship between the minibatch size and test AUC score. Based on the results and with the consideration of the training time, I chose 412 as the minibatch size since it provided the highest AUC score in this comparison.

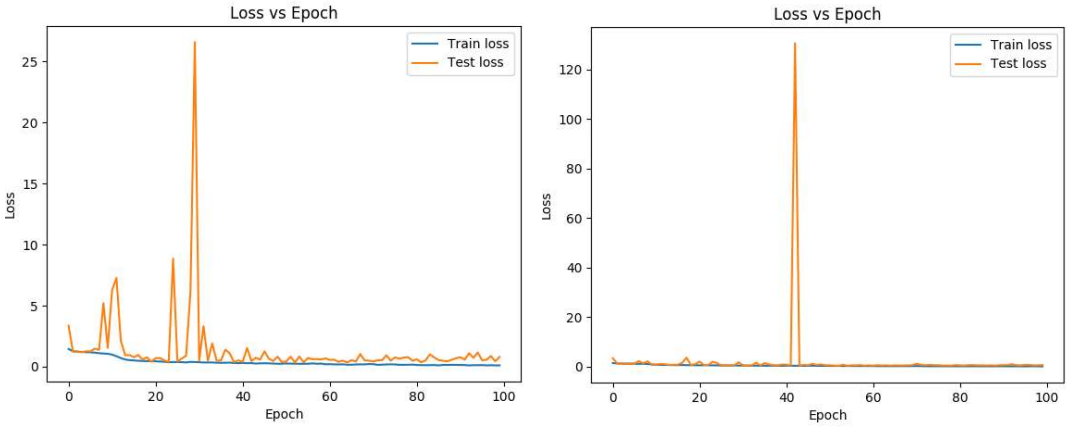
Table 1: The influence of minibatch size

Minibatch Size	200	412	430
Test $AUC_{(kaggle)}$	0.935	0.937	0.915

Learning rate comparison

Before comparing, I did a test on the Adam optimizer to find answer of: If I use Adam do I still need a learning rate schedular, since Adam is an adaptive learning rate optimizer? The figure 4, show the train loss and test loss of under different learning rate situation: Left one does not have the learning schedular and right one has the Cosine learning schedular. From the plot, I can see the loss are different which means learning rate schedular can still impact the loss curves when the optimizer is Adam.

Figure 7: Test for the Adam



The test AUCs for Learning rate comparison on the Kaggle are shown in table 2. Firstly, when the initial learning rate is 0.1, the dynamic learning rate schedular performed better than static learning rate. So I decided to use the dynamic learning rate schedular to find the best weight of the model. Then I chose the annealing schedular, since its values are quite wider. Besides, I changed the max value of the cosine annealing schedular to find the best learning rate. However, there is still no obvious relationship between the learning rate and test AUC score. Based on the results and with the consideration of the training time, I chose 0.1 as the max value of the cosine annealing schedular since it provided the highest AUC score in this comparison and would reduce my running time.

Table 2: The influence of learning rate

Static LR	0.1	Cosine LR	0.15	0.1	0.01
Test $AUC_{(kaggle)}$	0.911	Test $AUC_{(kaggle)}$	0.923	0.937	0.909

Weights initialization method comparison

The test AUCs for Initial weight method comparison on the Kaggle are shown in table 3. The result clearly indicates that Xavier normal is the best method of weights initialization.

Table 3: The influence of weights initialization

Weights Initialization	Xavier normal	Xavier uniform	Kaiming normal	Kaiming uniform
Test $AUC_{(kaggle)}$	0.937	0.928	0.927	0.922

Optimizers comparison

The test AUCs for optimizers comparison on the Kaggle are shown in table 4. The result clearly indicates that Adam's performance is better than SGD.

Table 4: The influence of optimizers

Initial weight method	SGD	Adam
Test $AUC_{(kaggle)}$	0.901	0.937

Image size comparison

The test AUCs for image size comparison on the Kaggle are shown in table 5. The result clearly indicates that the bigger size of image will lead to the higher AUC score.

Table 5: The influence of image size

Image size	(110, 164, 3)	(82, 120, 3)
Test $AUC_{(kaggle)}$	0.937	0.912

The original CNN and the deeper CNN

The test AUCs for different CNNs' comparison on the Kaggle are shown in table 6. The result clearly indicates that the deeper convolution neural network, the higher AUC score, in some this case.

Table 6: The influence of image size

CNN	The original one	Deeper one
Test $AUC_{(kaggle)}$	0.920	0.937

Ensemble model

This method really worked well. After combining top three high AUC score's model weights, I got the best AUC in this project which is 0.942.

Figure 8: The highest AUC score

Confusion matrix and classification report

Table 7: Confusion matrix

	Healthy	Multiple diseases	Rust	Scab
Healthy	847	28	2	59
Multiple diseases	11	98	26	29
Rust	6	53	1059	2
Scab	38	21	0	1006

Table 8: Classification report

	Precision	Recall	F1-score	Support
Healthy	0.94	0.91	0.93	929
Multiple diseases	0.49	0.60	0.54	164
Rust	0.97	0.95	0.96	1120
Scab	0.92	0.94	0.93	1065
Accuracy			0.92	3278
Macro avg	0.83	0.85	0.84	3278
Weighted avg	0.92	0.92	0.92	3278

Table 7 and table 8 show the confusion matrix and the classification report of the best single model with 0.937 on the Kaggle. From these two tables, I can diagnose the best single model. One possible improvement direction may exist in the category "Multiple diseases". If I can improve the accuracy of category "Multiple diseases", the model may perform better on Kaggle. Since there are least observations for "Multiple diseases" in these four categories,

one class weights or more augmented images in this category may be needed.

IV. Summary

Generally, in this project each parameter in the model can have a big influence on the model performance. A)Specifically, in this project, there is no obvious relationship between minibatch size and the performance of the model. With the consideration of running time, I prefer a bigger minibatch size with high AUC score. B)For the learning rate, using the dynamic learning rate is a good path to find the best AUC score. Although Adam is an adaptive learning rate optimizer, the learning rate schedular can still help improve the performance. In this project, the Cosine annealing schedular worked well. Besides, the max value of the learning rate in this schedular is also important. C)Then the weights initialization is also an important part. In this project, the Xavier normal weights initialization method performed well. D)Adam is still a good optimizer to train the model and E)bigger image size as well as F)deeper network will lead to a good performance. G)Finally, the ensemble of some high performance model weights is a good way to get a higher performance.

All these things above are what I learnt in this project. Based on this I found that the experiment is a good way to learn some new parameter tuning rules in this class. Also, if given more time, I want to try the CNN with some branches and find efficient methods to reduce the occupied GPU memory during model training, which means a more efficient method still need to be found. Finally, since there are least observations for "Multiple diseases" in these four categories leading to lowest accuracy, one class weights or more augmented images in this category may be needed.

Percentage of code that I copied from internet: 5%.

V. Reference

[1] Mingxing Tan, Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, ICML 2019

[2] Hagan, Demuth etc. Neural Network Design 2nd edition.

[3] https://pytorch.org/docs/stable/torch.html