DATS 6203 Final Report

Plant Pathology 2020 – FGVC7
Identify the category of foliar diseases in apple trees

Xin Ma

George Washington University
04/21/2020

Introduction

Overview of Project

Misdiagnosis of the many diseases impacting agricultural crops can lead to many serious consequences such as misusing of chemicals leading to the emergence of resistant pathogen strains, increasing input costs and more outbreaks with significant economic loss and environment impacts. Current diseases diagnosis based on human scouting is time-consuming and expensive. With the increase of computing power, especially in the past decade, we can build deep learning model to help us diagnose. The computer-vision based on deep learning models have the promise to increase efficiency. However, the great variance in symptoms due to age of infected tissues, genetic variations, and light conditions within trees might increase the complexity of this problem and decreases the accuracy of detection.

The purpose of our project is to identify the category of foliar diseases in apple trees. We participated in a competition from Kaggle where we downloaded the datasets. The detail of this competition and datasets will be discussed in next section. Before we start building model, we do some exploratory data analysis (EDA) of datasets. Through EDA, we find there are some problems of metadata such as imbalanced classes, inconsistent image size and duplicated data. To deal with these problems, we use different preprocessing methods, for example, resize, flipping, rotation to augment the data. After data preprocessing, we briefly introduce some essential theories and algorithms that we use in our model. Then, we show the architecture of our model and the process of tuning hyperparameters. Finally, we show performance of our model and summarize the results we obtained.

Outline of Shared Work

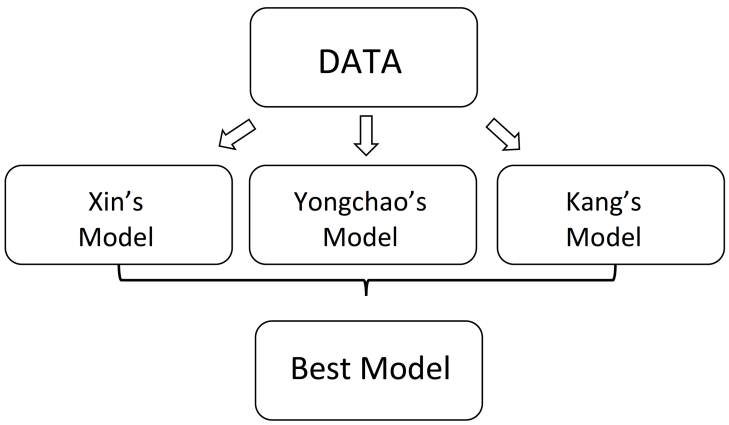

Figure 1: Outline of Shared Work

Each member builds his own model . After internal competition, the best model is selected as the group model in the group report. Each member's own model is discussed in his individual report.

Exploratory Data Analysis

Metadata

The metadata is downloaded from Kaggle (link: https://www.kaggle.com/c/plant-pathology-2020-fgvc7/data). It contains 1821 train images and corresponding labels and 1821 test images. There are four targets of this dataset: 'healthy','multiple diseases', 'rust' and 'scab'. Each image only belongs to one category. The data frame of image id and its label is shown in figure 2.

	image_id	healthy	multiple_diseases	rust	scab
0	Train_0	0.00000	0.00000	0.00000	1.00000
1	Train_1	0.00000	1.00000	0.00000	0.00000
2	Train_2	1.00000	0.00000	0.00000	0.00000
3	Train_3	0.00000	0.00000	1.00000	0.00000
4	Train_4	1.00000	0.00000	0.00000	0.00000
5	Train_5	1.00000	0.00000	0.00000	0.00000
6	Train_6	0.00000	1.00000	0.00000	0.00000
7	Train_7	0.00000	0.00000	0.00000	1.00000
8	Train_8	0.00000	0.00000	0.00000	1.00000
9	Train_9	1.00000	0.00000	0.00000	0.00000
10	Train_10	0.00000	0.00000	1.00000	0.00000

Figure 2: The data frame of $image_id$ and its label

Targets Distribution

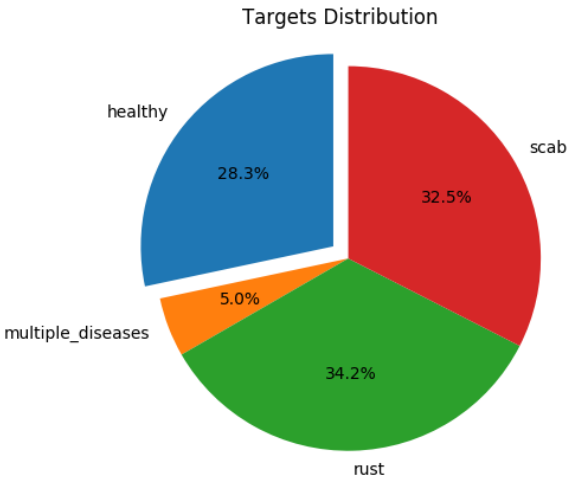

Figure 3: Targets Distribution

Since we have four targets, we need to check whether the distribution of these four classes is uniform. As figure 3 shown below, the dataset is imbalanced. There is only 5% of training data comes from 'multiple diseases' class. In preprocessing section, we will show how we deal with this problem.

Visualize Sample Images

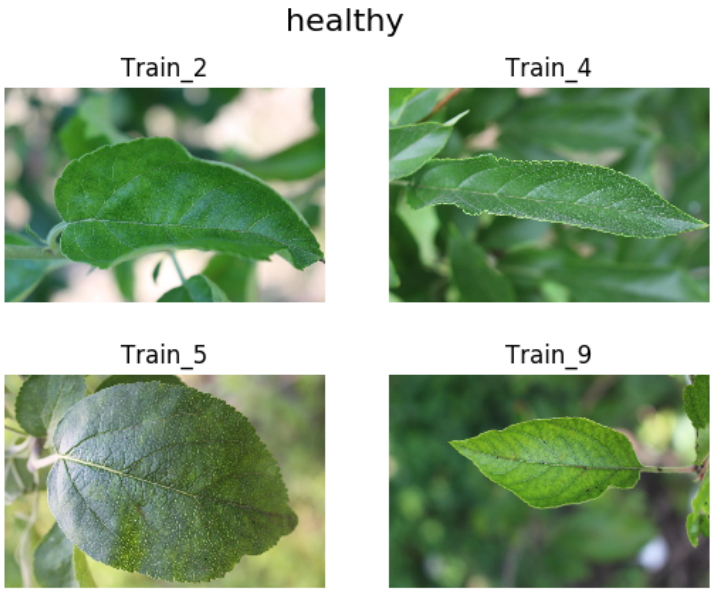

Figure 4: Healthy

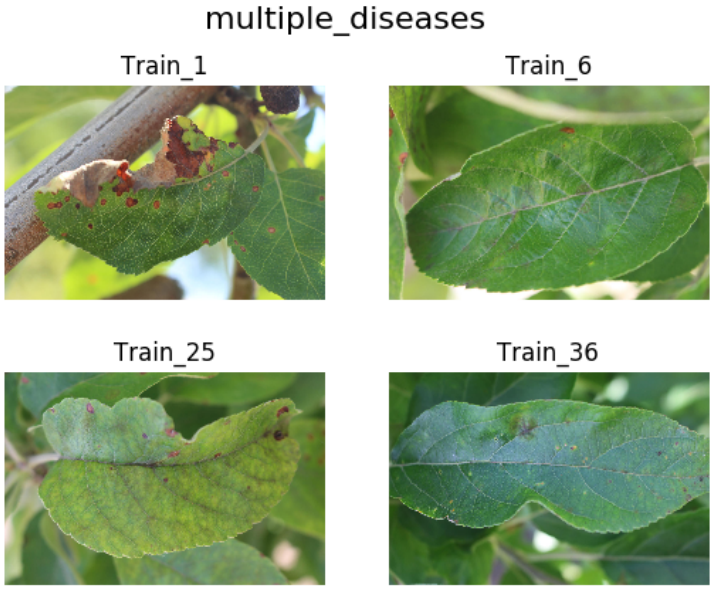

Figure 5: multiple diseases

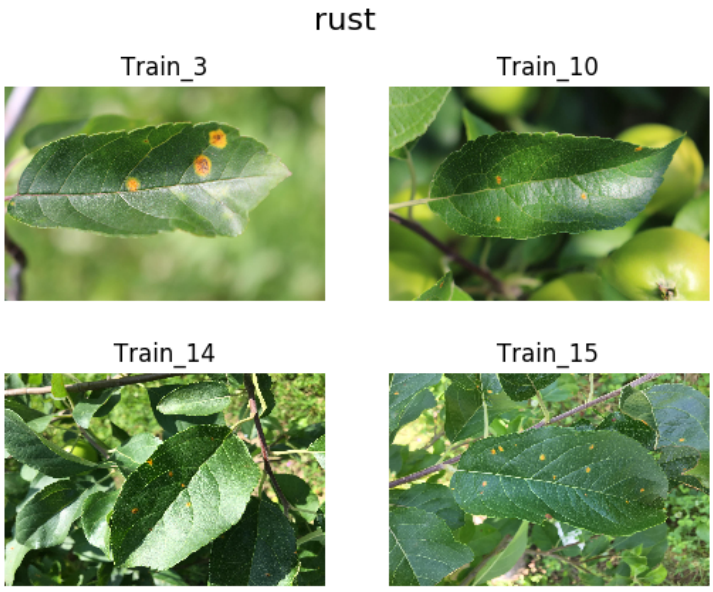

Figure 6: rust

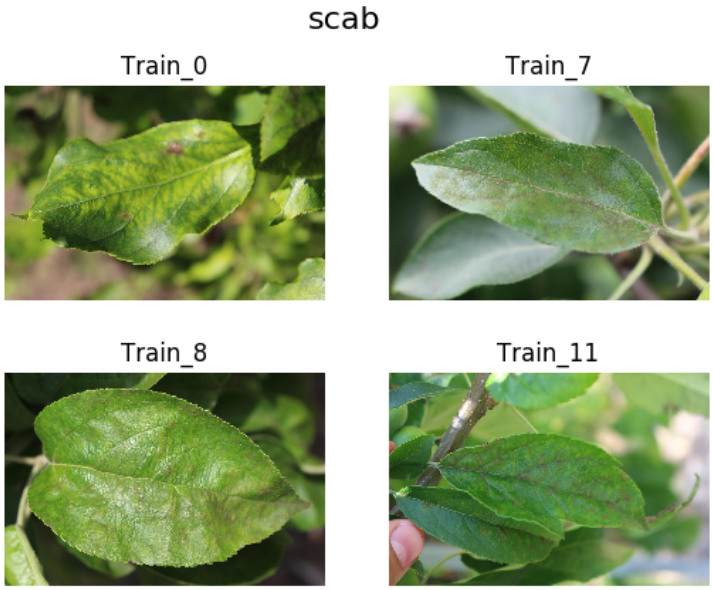

Figure 7: scab

Background Information

Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) typically consists of convolutional layers, pooling layers and fully connected layer, as figure 8 shown.

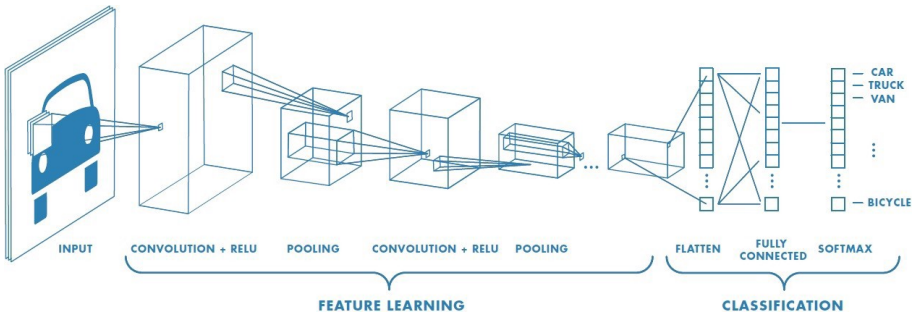

Figure 8: CNN

Convolutional layers use kernels to extract the features from input data. Each number of kernels gets multiplied with the corresponding number on the input data and them they all are summed up for the value in the corresponding position in the output matrix. Then we plug the output in an activation function. We always use ReLu function here.

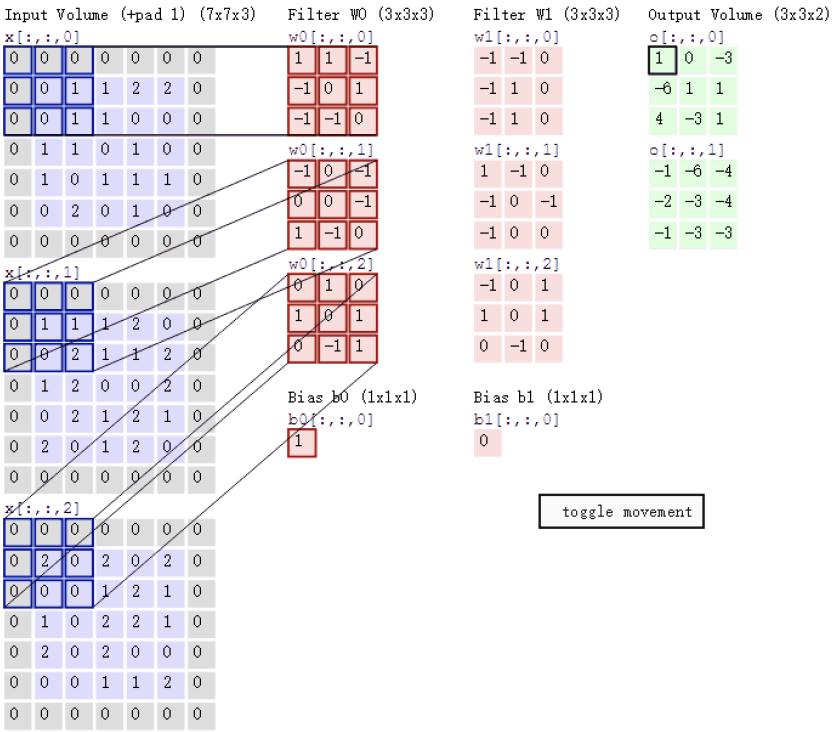

Figure 9: Convolutional Layer

After convolution layer, it is pooling layer. Its function is to reduce the size of convolutional

output.There are two pooling methods: max pooling and average pooling. Max pooling is to pick up the maximum value of a local region. Average pooling is to calculate the average value of a local region.

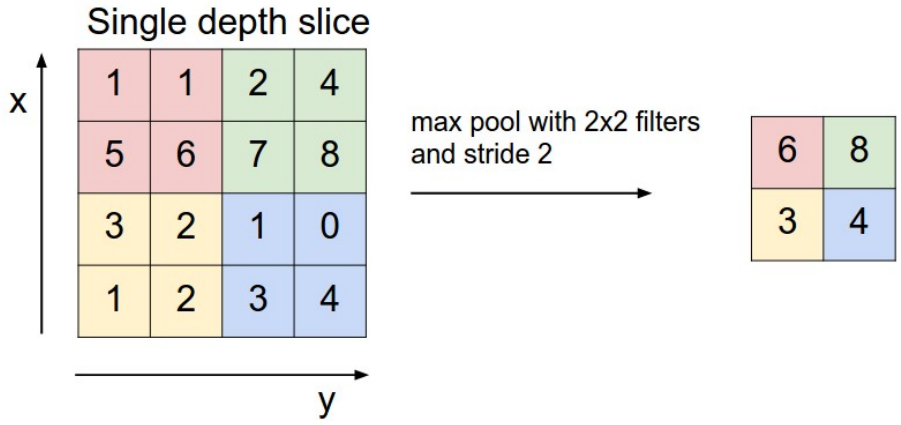

Figure 10: Max Pooling

After pooling layer, the size of the feature volume will be acceptable. we need to flatten the output of last pooling layer to convert the output volume to a vector as the input value of fully connected layer. Finally, we can use softmax or sigmoid function to obtain the score.

Process Representation

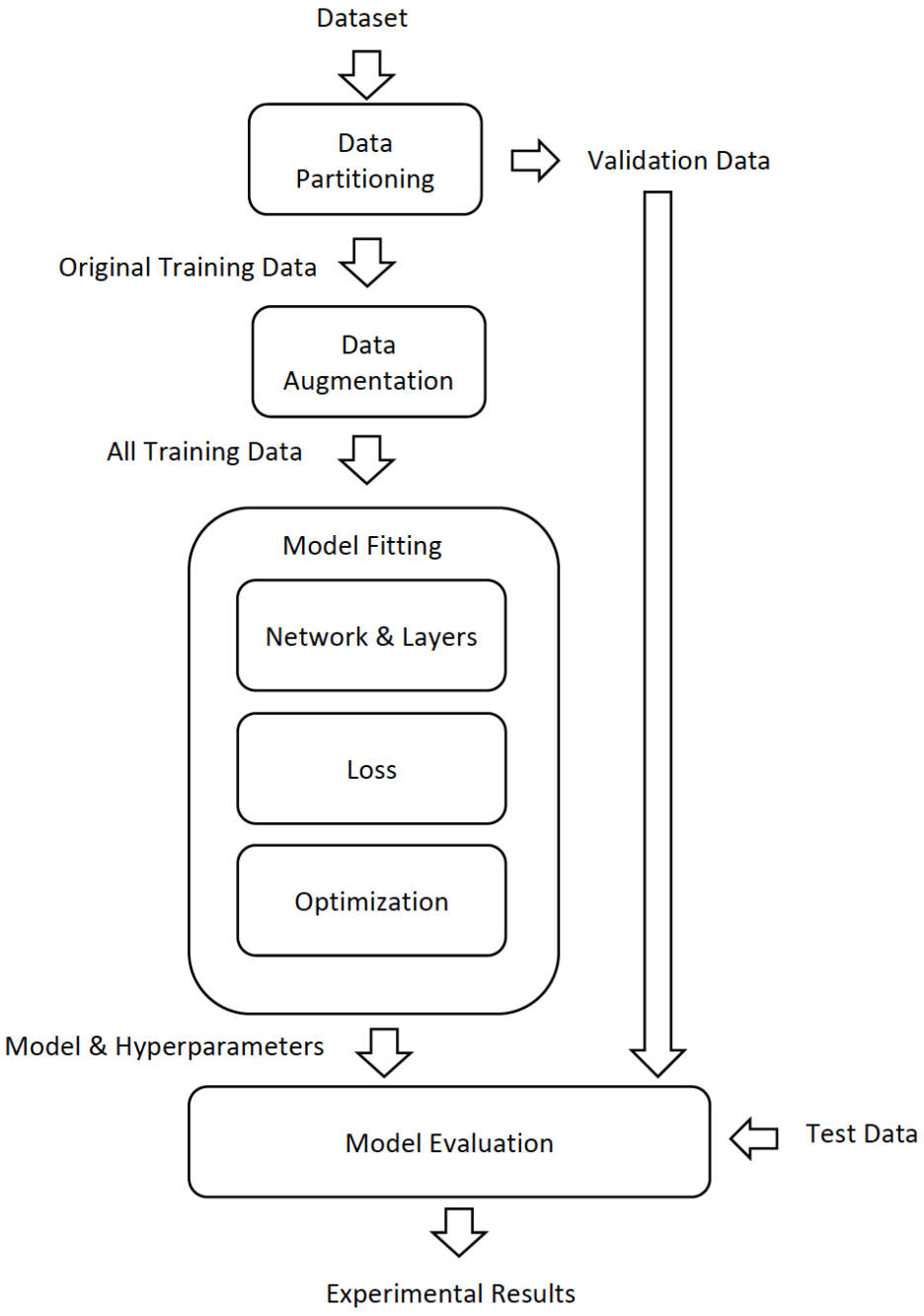

Figure 11: Process Representation

Data Preprocessing

Label Encoding

Since the CrossEntropyLoss does not accept one-hot-encoded labels, I use integers [0, 1, 2, 3] to represent different classes following the order: 'healthy','multiple diseases', 'rust' and 'scab'.

Imbalanced Classes

After EDA, the original dataset is imbalanced. In data augmentation process, I will deal with this problem.

Data Partitioning

After data augmentation, we used random sampling to split original data into training set and test set based on the proportion $8:2$.

Data Augmentation

If we use original training data to train the model, the results in the model that have poor predictive performance, especially for the minority class. This is a problem because typically, the minority classes are more important and therefore the problem is more sensitive to classification errors for the minority classes than the majority classes. To combat imbalanced classes, we use data augmentation to increase sample size of minority classes. We used many image data augmentation techniques, for example, flipping, cropping, rotation, etc. After data augmentation, the number of samples in each class is comparable and the total number of samples in training set is also expanded a lot.

After data augmentation, the number of observations in each class is about 500 and the total size of dataset is enlarged, which is very important in nerual network.

Other Preprocessing

The size fo images in the dataset are all different, so we also resized all the original images and augmentation images into 50x50.

Train Model

Network Architecture

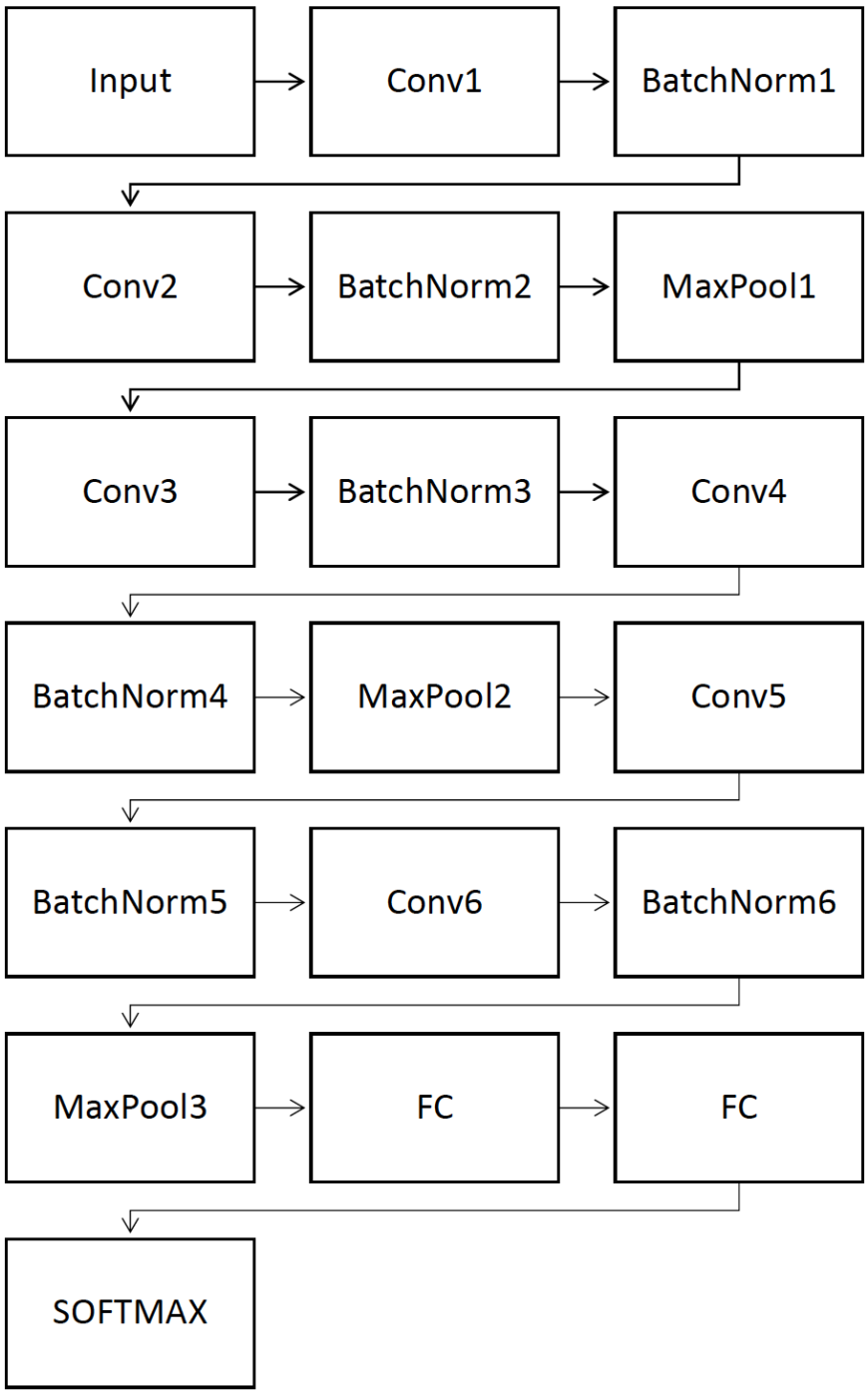

Figure 12: Architecture

```
class CNN(nn.Module):
def __init__(self):
super(CNN, self).__init__()
self.conv1 = nn.Conv2d(3, 16, (3, 3),padding=1)  # output (n_examples, 16, 50, 50)
self.convnorm1 = nn.BatchNorm2d(16)
self.conv2 = nn.Conv2d(16, 64, (3, 3),padding=1) # output (n_examples, 64, 50, 50)
self.convnorm2 = nn.BatchNorm2d(64)
self.pool1 = nn.MaxPool2d((2,2))                  # output (n_examples, 64, 25, 25)
self.conv3 = nn.Conv2d(64,128,(3,3),padding=1)    # output (n_examples, 128, 25, 25)
self.convnorm3 = nn.BatchNorm2d(128)
self.conv4 = nn.Conv2d(128,256,(3,3),padding=1)   # output (n_examples, 256, 25, 25)
self.convnorm4 = nn.BatchNorm2d(256)
self.pool2 = nn.MaxPool2d((2,2))                  # output (n_examples, 256, 12, 12)
self.conv5 = nn.Conv2d(256,256,(3,3),padding=1)   # output (n_examples, 256, 12, 12)
self.convnorm5 = nn.BatchNorm2d(256)
self.conv6 = nn.Conv2d(256,512,(3,3),padding=1)  # output (n_examples, 512, 12, 12)
self.convnorm6 = nn.BatchNorm2d(512)
self.pool3 = nn.MaxPool2d((2,2))                 # output (n_examples, 512, 6, 6)
self.linear1 = nn.Linear(512*6*6, 400)
self.linear1_bn = nn.BatchNorm1d(400)
self.drop = nn.Dropout(DROPOUT)
self.linear2 = nn.Linear(400, 4)
self.act = torch.relu

def forward(self, x):
x = self.convnorm1(self.act(self.conv1(x)))
x = self.pool1(self.convnorm2(self.act(self.conv2(x))))
x = self.convnorm3(self.act(self.conv3(x)))
x = self.pool2(self.convnorm4(self.act(self.conv4(x))))
x = self.convnorm5(self.act(self.conv5(x)))
x = self.pool3(self.convnorm6(self.act(self.conv6(x))))
x = self.drop(self.linear1_bn(self.act(self.linear1(x.view(len(x), -1)))))
return self.linear2(x)
```

Hyperparameters Tuning

According to figure 10, when lr=0.0001, the convergence speed is much faster than others.

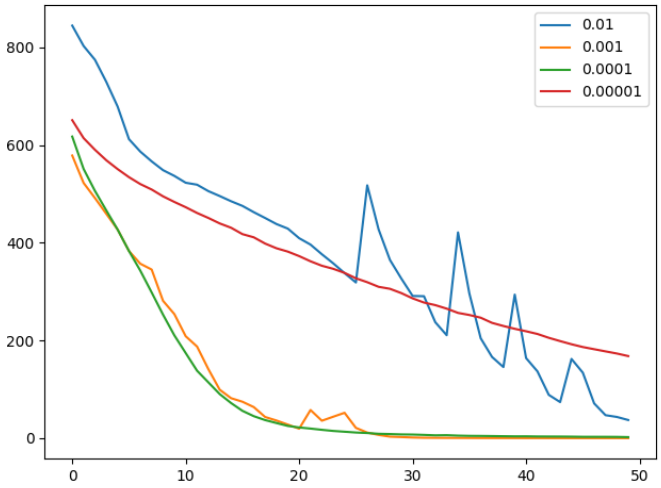

Figure 13: Train Loss vs Epochs on different LR

The learning rate also changes over time, say learning rate decay. In first 25 epochs, $lr = 0.01$, then it decayed to $lr = 0.001$ in the following 25 epochs, $lr = 0.0001$ in the last 50 epochs. Learning rate decay can mitigate the fluctuation of loss in late epochs.
The batch size is 64. If we set a large batch size, the cuda will be out of memory.
The number of epochs is 50. In training process, the loss stops decreasing after 30 epochs.

Avoid Overfitting

According to figure 11, the validation loss decreases then increases, which indicates overfitting problem.

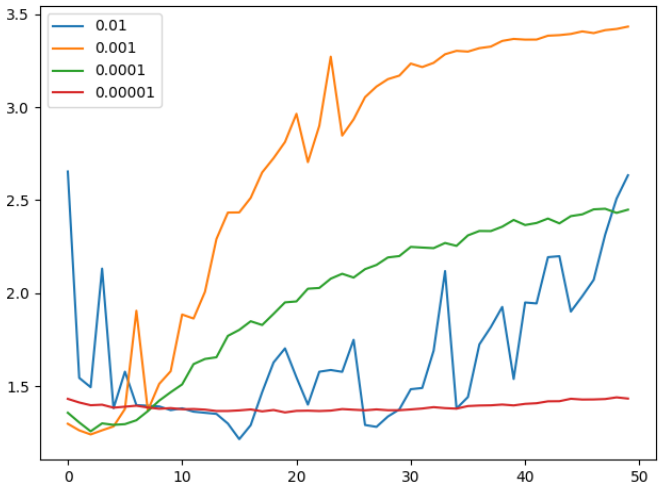

Figure 14: Val Loss vs Epochs on different LR

To avoid overfitting and enhance generalization ablility of model, we use validation loss as

monitor to select the best model during the whole process. I save the model that has minimum test loss during whole process. In addition, the dropout is 0.5 in fully connected layer and weight decay is 0.01 in SGD optimizer.
Data augmentation is also helpful for avoiding overfitting.

Performance on Validation Data and Test Data

Accuracy
74.95%

Finally, the accuracy on validation data is 74.95%. However, the accuracy on training data is 98.74%, which indicates overfitting problem again. Although I have tried many different architecture of networks, augmented training data, applied dropout and weight decay, the overfitting problem still existed.

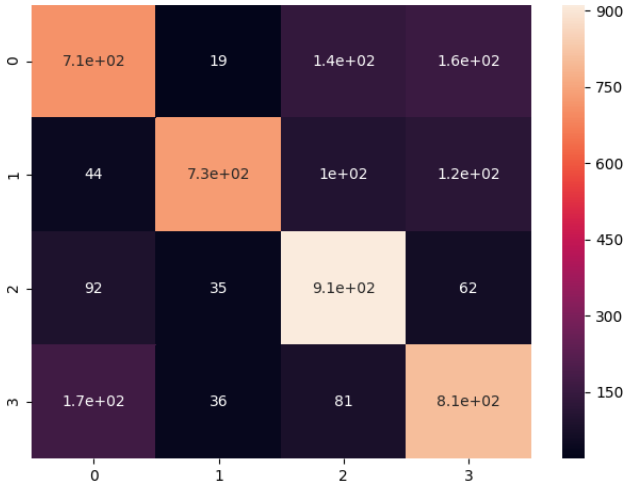

Figure 15: Confusion Matrix of Validation Data

ROC AUC Score
0.789

Figure 15 shows the confusion matrix of validation data. It is obvious that the performance of our model is not really good between class 1, multiple diseases, and class 3, scab. I think the reason is that class 1 might include many features of class 3. I think that the data creater could specify what diseases the tree has rather than use a general class to classify.

Summary

CNN is indeed a very powerful tool in computer vision area. It can extract image features automatically and deal with some high dimensional data. However, there are some things worth our attention. First, the number of parameters is really large so a big dataset is required. Second, due to many parameters, it is easy to meet overfitting problem. Tuning hyperparameters is very important during the training process. Third, the quality of data is also important. For example, in this project data, the object is a leaf in an image. The background is also some

leaves. The color of object and background is very similar. If we can remove background, I think the performance will be better. In addition, depth perception—angle, light and shade can also affect the performance of model. In the following work, I can focus on these problems to make model better.

Copy Percentage

Script Name	Percentage
load_data.py	0.00%
aug_data.py	27.6%
eda.py	0.00%
train.py	15.9%
predict.py	8.33%

Reference

[1]$Hangan, M.T., Demuth, H.B., Bealee, M.H., JesusOrlandodEE.(2016).Neuralnetworkdesign.S.I.: s.n.$

[2]$PyTorchForums.(n.d.).Retrievedfromhttps://discuss.pytorch.org/$

[3]$torchvisionű.(n.d.).Retrievedfromhttps://pytorch.org/docs/stable/torchvision/index.html$

[4]$Saha, S.(2018, December17).AComprehensiveGuidetoConvolutionalNeuralNetworks - theELI5way.Retrivedfromhttps://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-theeli5-way-3bd2b1164a53$