Identify the category of foliar diseases in apple trees

Kang Huang
Xin Ma
Yongchao Qiao

Content

Introduction

Part One

Background

Part One

Diagnose by human scouting

- Time-Consuming
- Expensive

Diagnose by computer-vision Tech.

- Efficiency
- Accuracy

EDA

Part Two

Metadata

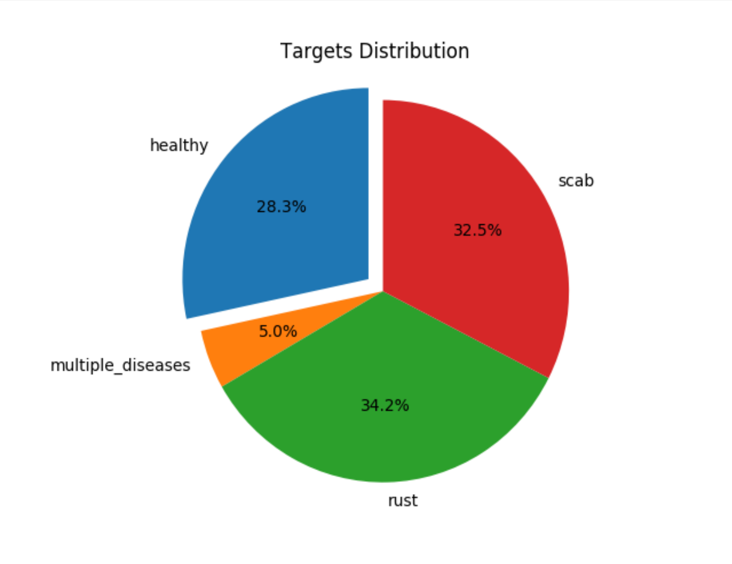
Part Two

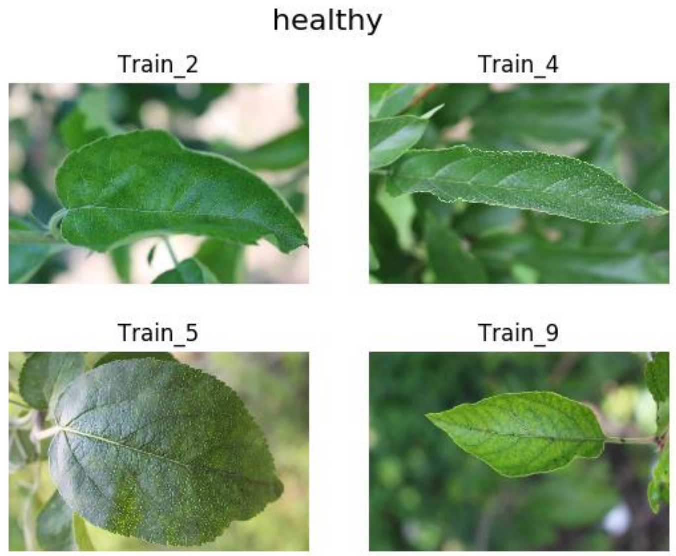
- Data Source: https://www.kaggle.com/c/plant-pathology-2020-fgvc7
- Training Data: 1821 Images and Labels
- Test Data: 1821 Images
- 4 Classes: ‘Healthy’ , ‘Multiple_Diseases’ , ‘Rust’ , ‘Scab’

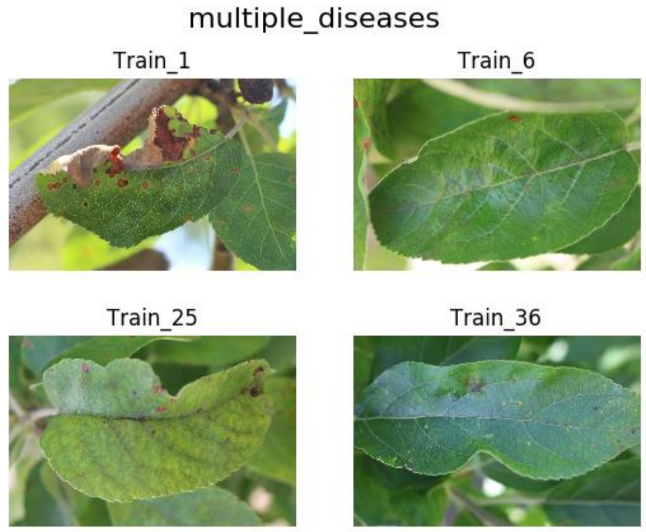
	image_id	healthy	multiple_diseases	rust	scab
0	Train_0	0.00000	0.00000	0.00000	1.00000
1	Train_1	0.00000	1.00000	0.00000	0.00000
2	Train_2	1.00000	0.00000	0.00000	0.00000
3	Train_3	0.00000	0.00000	1.00000	0.00000
4	Train_4	1.00000	0.00000	0.00000	0.00000
5	Train_5	1.00000	0.00000	0.00000	0.00000
6	Train_6	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

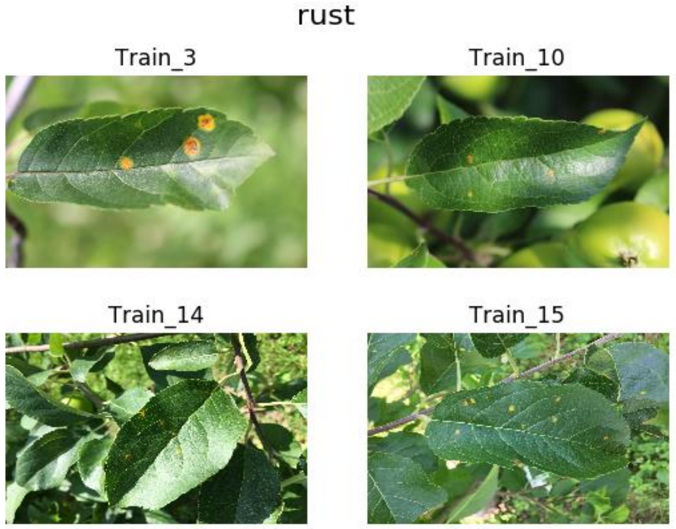
Targets Distribution and Visualize Data
Part Two

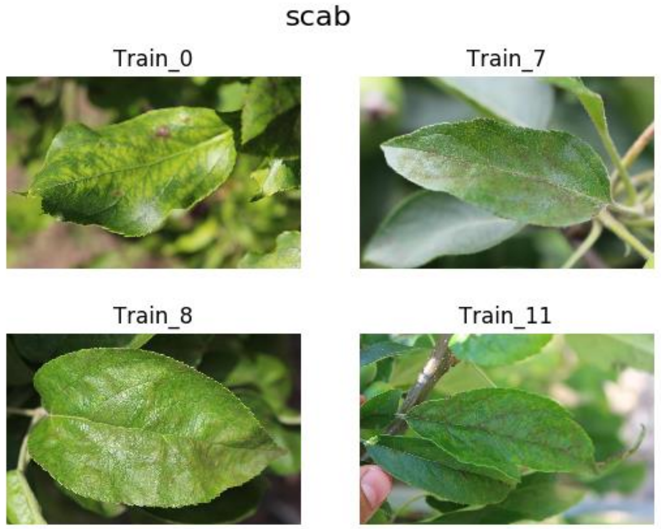
- Imbalanced Classes











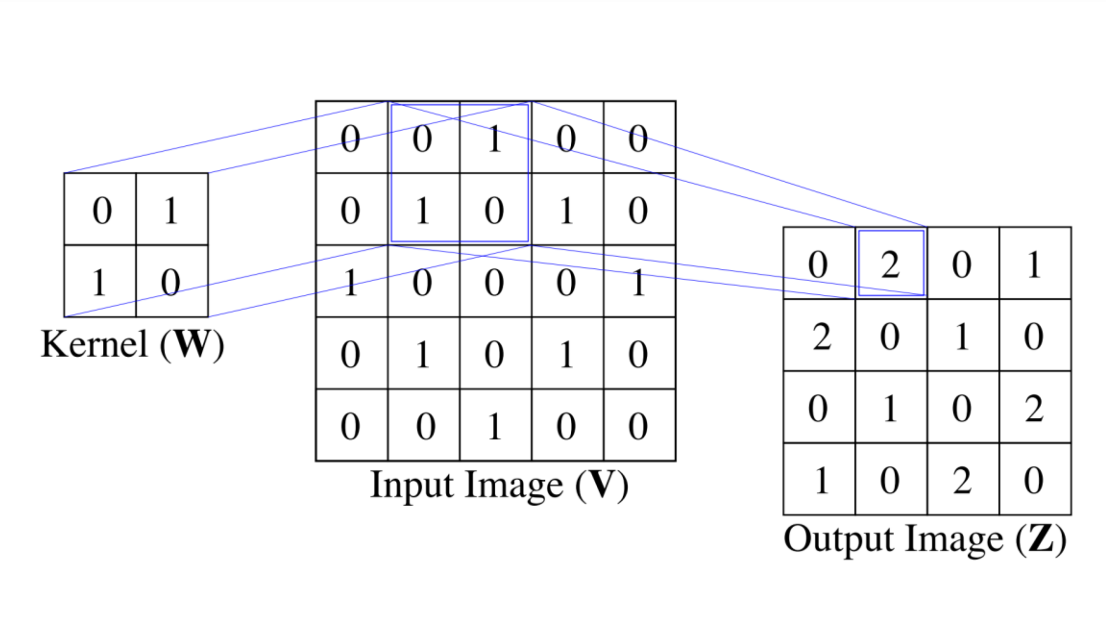
Theory

Part three

CNN

Part Three

1. Convolution layer



2. Sample layer

The pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.

Max pooling:

$$z_{i,j} = \max\{v_{r(i-1)+k,c(j-1)+l} | k = 1, \dots, r; l = 1, \dots, c\}$$

Why CNN?

Part Three

1. Sparse interaction

2. Parameter sharing

3. Equivariant representations

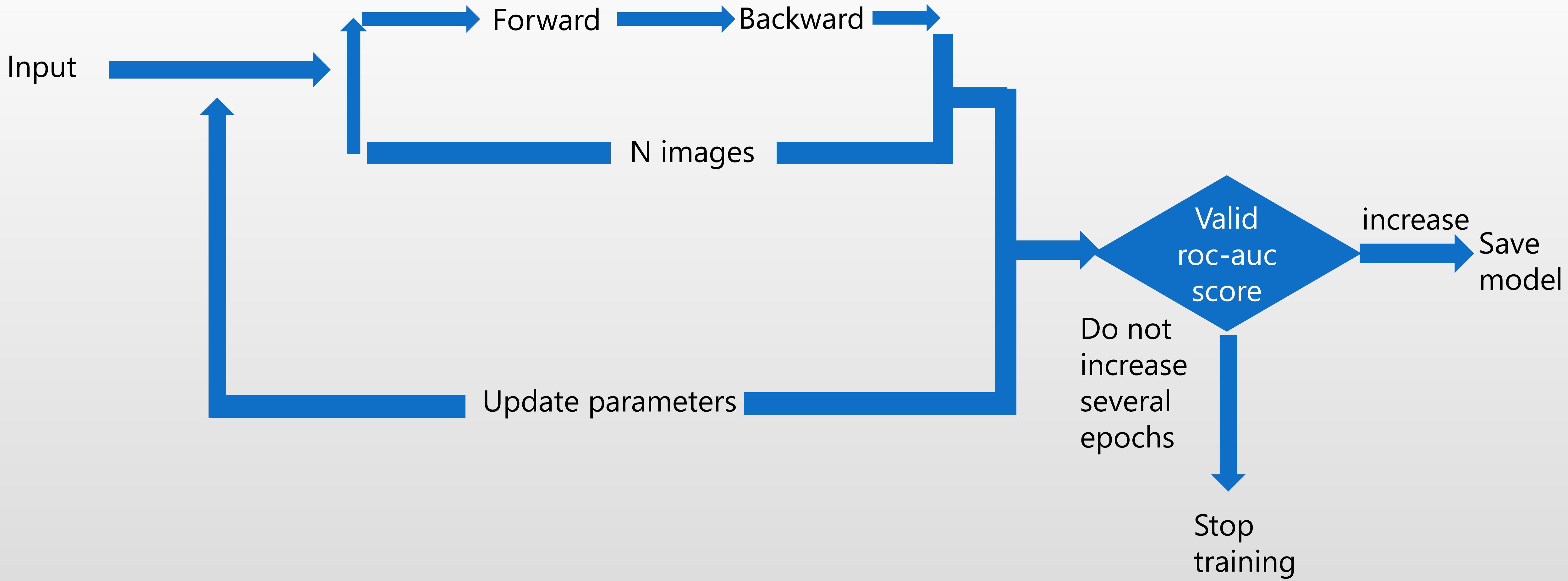
Project: identify the category of foliar disease in apple trees based on the images

Image classification

CNN

Algorithm

Part Three



Experiment Setup

Part Four

DATA Preprocessing

Firstly, upload the plant-pathology-2020-fgvc7.zip to the cloud.

Table1 Check original dataset

Data size	(1365, 2048, 3)	(2048, 1365, 3)	Total
Original Train	1819	2	1821
Original Test	1801	20	1821

Resize both train and test set into (110, 164, 3)

Secondly, augment the original train data to **10926** observations:

Method:

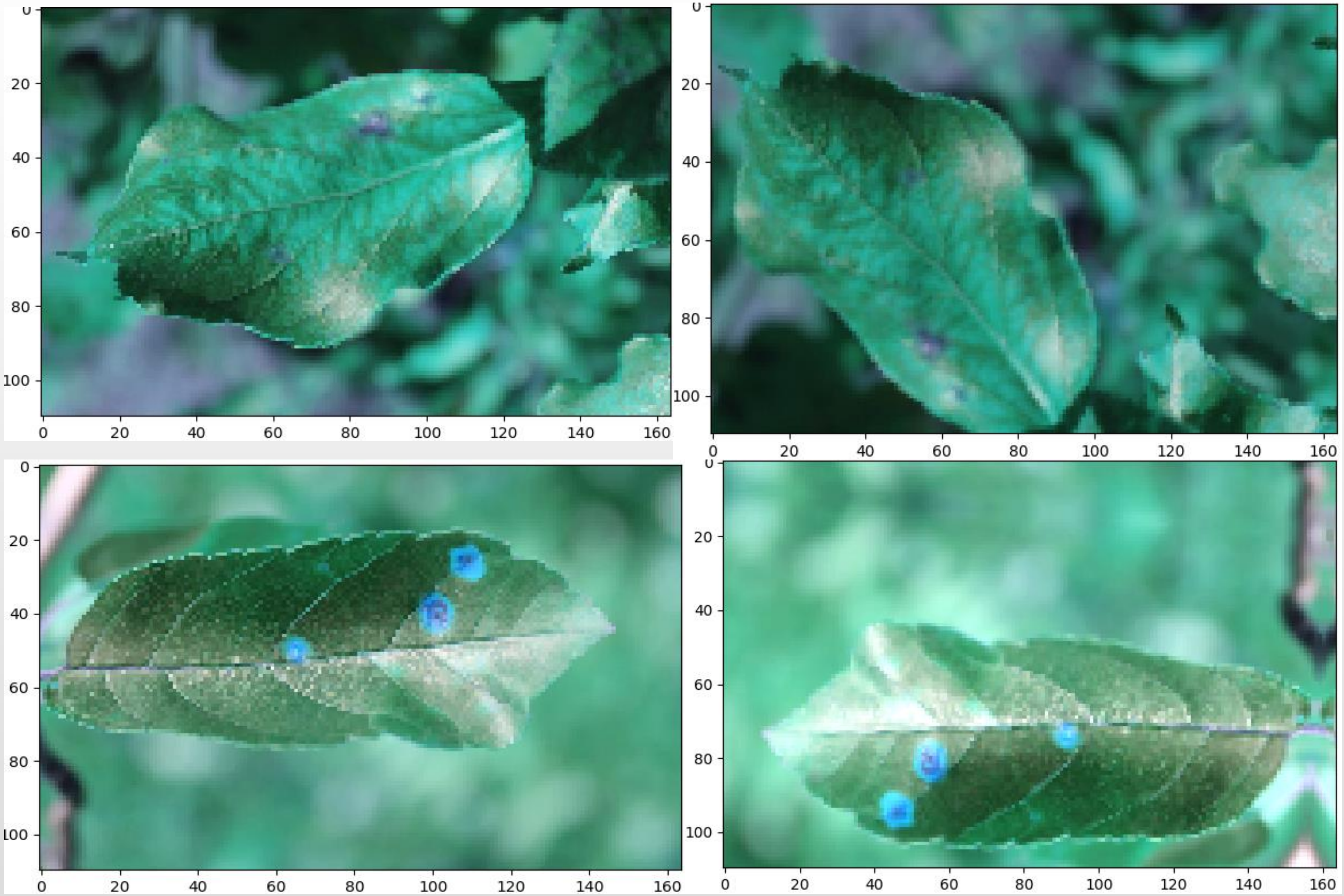
Rotation, Horizontal and vertical flip, Width and Height shift, Feature-wise_center, Feature-wise_std_normalization, Brightness adjustment Fill_mode as “reflect”

Thirdly, switch the one-hot encoding target Into four numbers representing four categories(strategy of split)

Fourthly, randomly split the original train into train : test = 0.7 : 0.3 = 7648 : 3278

DATA Preprocessing

Figure1: Original images vs augmented image examples

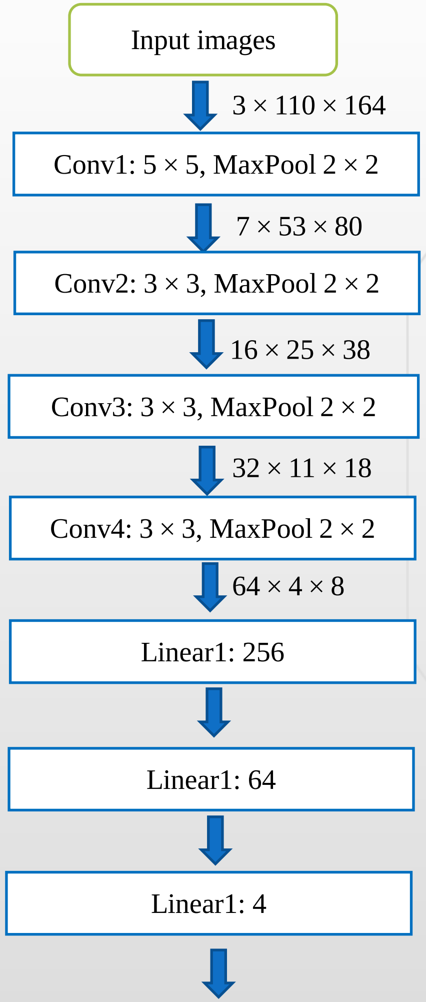


Left side:
Original images

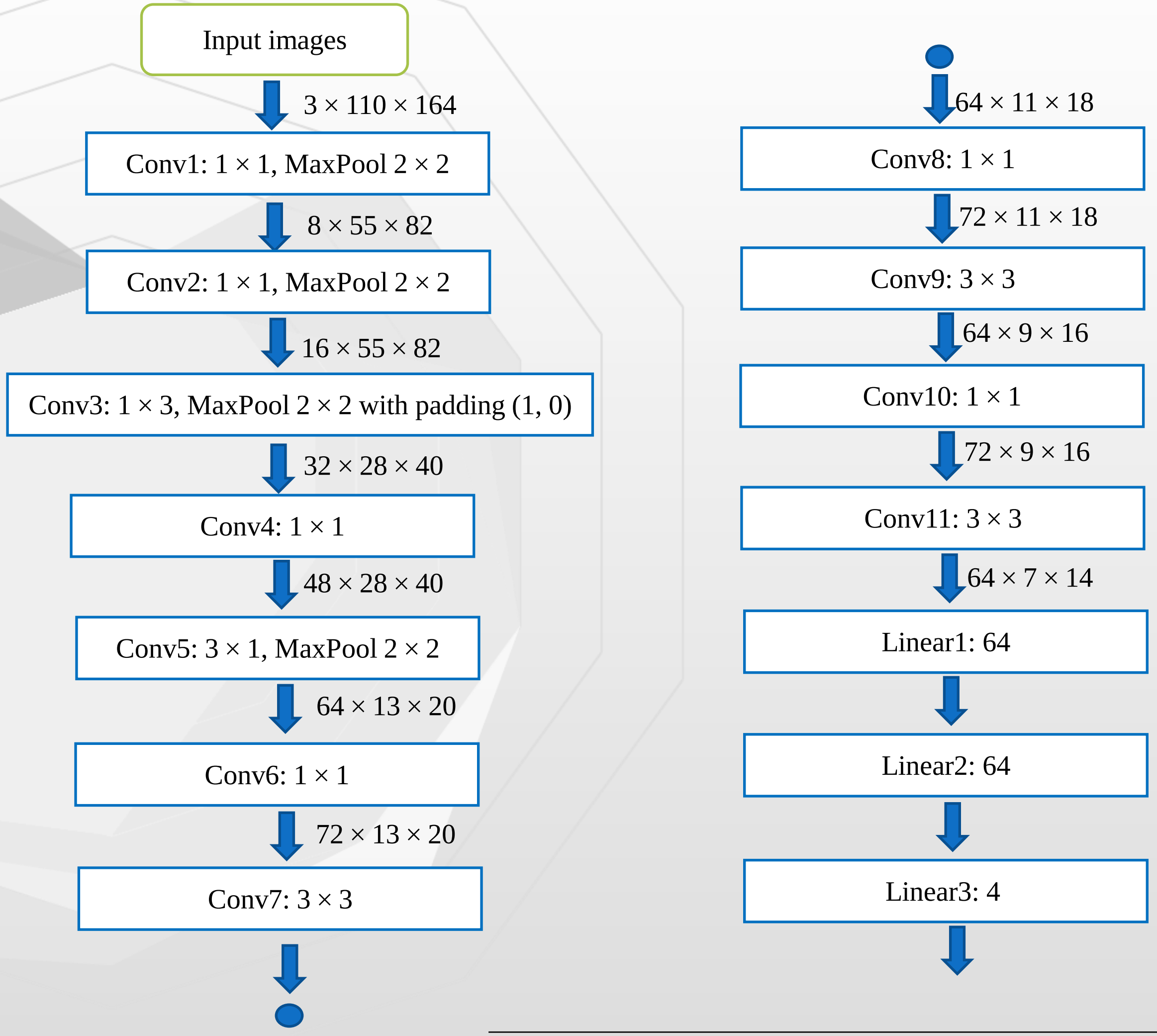
Right side:
Augmented
images

Model Construction

The original CNN model:



The Best CNN model:



Parameters tuning

First part: Training operations

1. Use the train set to feed the model and test set to check the model performance.
2. Performance Index: Crossentroploss; Matrics: AUC score
3. Add early stopping operation to save the best model weight on the test set;
4. Plot the train loss and test loss against the epochs to detect the overfitting.
5. Dropout, Batchnormal layers, group values in the Convo2d to prevent overfitting

Second part: Minibatch Size

Try and compare: 200, 412, 430(Max memory for GPU)

Third part: Learning rate pattern and value

Try and compare: Static LR: 0.1; Dynamic LR: cosine annealing schedular: max=0.1

Parameters tuning

Fourth part: Initial weight method
Try and compare: Xavier normal, Xavier uniform, Kaiming normal and Kaiming Uniform; with bias~N(0, 0.02).

Fifth part: Optimizers
Try and compare: SGD and Adam

Sixth part: Image Size
Try and compare: (110, 164, 3) & (82, 123, 3)

Seventh part: Original and deeper CNNs
Try and compare: The original CNN & The best CNN

Eighth part: Ensemble model
Combine different model weights with high AUC score

Results

Part Five

Results

Table 2: The result of Minibatch comparison

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

Minibatch comparison

There is no obvious relationship between the minibatch size and test AUC score.

Prefer 412 as the minibatch size since it provided the highest AUC score and reduce the training time.

Results

Table 3: The result of Learning rate comparison

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

Learning rate comparison

When the initial learning rate is 0.1, the dynamic learning rate schedular performed better than static learning rate.

There is still no obvious relationship between the learning rate and test AUC score.

Prefer max =0.1 with Cosine LR schedular

Results

Weights initialization method comparison

Table 4: The result of different weights initialization

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

Xavier normal weight initialization is the best one among these 4 initialization methods.

Prefer Xavier normal weight initialization method

Results

Table 5: The result of Optimizers comparison

Optimizers	SGD	Adam
Test $AUC_{(kaggle)}$	0.901	0.937

Optimizers comparison

Prefer Adam since it provided highest AUC score and less running time.

Results

Table 6: The result of Image size comparison

Image Size	(110, 164, 3)	(82, 123, 3)
Test $AUC_{(kaggle)}$	0.937	0.912

Image size comparison

The bigger image size, the higher AUC score.

Prefer (110, 164, 3) as the image size since it provided the highest AUC score and enough augmented images under the memory limit of GPU

Results

Original and deeper CNNs comparison

Table 7: The result of different CNNs comparison

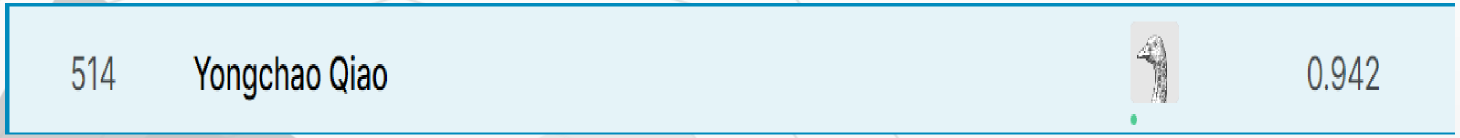
CNNs	The original one	The best one
Test $AUC_{(kaggle)}$	0.920	0.937

In this case, the deeper CNNs the better AUC scores.

Based on this result, I decided to tune the parameters of the best CNN to get a higher AUC score.

Results

Figure 2: The highest AUC score in this project



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.

Summary

Part Six

Summary & Conclusions

In this project

A) Each parameter in the model can have a big influence on the model performance.

B) 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.

C) For the learning rate, using the dynamic learning rate is a good path to find the best AUC score. In this project, the Cosine annealing schedular worked well.

D) The weights initialization is also an important part. The Xavier normal weights initialization method performed well.

E) Adam is still a good optimizer to train the model.

F) Bigger image size will lead to a good performance.

G) Deeper network will lead to a good performance.

H) The ensemble of some model weights with high AUC is a good way to get a higher AUC.

Summary & Conclusions

Possible improvement

A) Try the CNN with some branches

B) Find efficient methods to reduce the occupied GPU memory during model training, which means a more efficient method still need to be found.

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

[4] Kaggle notebook: https://www.kaggle.com/pestipeti/plant-pathology-2020-pytorch

[5] Pytorch tutorial: https://pytorch.org/tutorials/

[6]Liu, T., Fang, S., Zhao, Y., Wang, P., & Zhang, J. (2015). Implementation of training convolutional neural networks. arXiv preprint arXiv:1506.01195.

[7]Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.