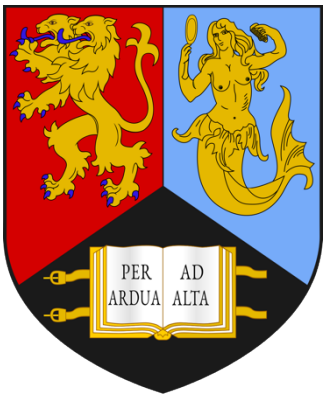
Plant Diseases Classification Using Convolutional Neural Network

Zirui Wang – 1935080

Supervised by

Dr. Kashif Rajpoot

BSc Computer Science



School of Computer Science

University of Birmingham

Report words count: 8087

Abstract

Plant Diseases are harmful to the growth of plant tissues and organs, and even cause withering and death. It is necessary to diagnose and treat the disease timely. Convolutional neural networks have excellent ability in visual tasks and can react objectively and quickly. This project is based on the plant diseases dataset provided on Kaggle, using convolutional neural networks to achieve accurate disease classification.

This project starts with the task of plant diseases, learns and explores data processing technology, convolutional neural network architectures, implementation, training and evaluation. By combining the advantages of some classic CNN models, a new CNN architecture based on separate channels (ChannelSplitNet) is designed. The innovative proposal uses the Conv-Flatten structure to replace the traditional dense layer to further reduce the model parameters. This project details multiple experiments to evaluate the performance of ChannelSplitNet and the feasibility of replacing the dense layer with the Conv-Flatten structure. Then use the ensemble model to further improve the accuracy of diagnosing diseases. At the end of the report, some suggestions for future work are put forward.

Acknowledgments

I would like to express my special thanks to Dr. Kashif Rajpoot, who is my project supervisor, for the academic supports, professional advice and the supervision of my project throughout my third year.

Glossary of Terms

ANN Artificial Neural Network

CNN Convolutional Neural Network

Conv Convolutional

CF Conv-Flatten

K-NN K-Nearest Neighbor

SVM Support Vector Machine

TfS Training from Scratch

Contents

1. INTRODUCTION **7**

1.1 Motivation 7

1.2 Aim of Project 7

1.3 Report Structure 8

2. BACKGROUND **9**

2.1 ANN, Machine Learning and Image Classification 9

2.2 Convolutional Neural Network 10

2.2.1 The Convolution Operation 12

2.2.2 The Pooling Operation 14

2.2.3 Activation Function 15

2.2.4 Dropout 16

2.2.5 BatchNormalization 16

2.2.6 Fully Connected and SoftMax Classification 17

2.2.7 Optimizers 17

2.3 Related Work 18

3. DATA **21**

3.1 Data Integration 21

3.2 Image Resize 21

3.3 Train-Valid-Test split 21

3.4 Data Augmentation 22

4. METHODOLOGY **25**

4.1 Classic CNN Models 25

4.1.1 Training from Scratch 25
4.1.2 Transfer Learning 27
4.2 Design a CNN Architecture 27
4.2.1 Experiment – Number of Convolutional Layers 28
4.2.2 ChannelSplitNet 28
4.2.3 Improvement of ChannelSplitNet 31
4.3 Ensemble of Models 33
4.4 Training Methodology 33
4.5 Evaluation Methodology 34
4.6 Extra Experiments – Conv-Flatten Output 35
4.6.1 Experiment One – Apply CF Output Structure to Other CNN Architecture 35
4.6.2 Experiment Two – Compare the Performance of Dense Output, CF Output and Improved CF Output 36
4.7 Summary 37
5. RESULT AND ANALYSIS 38
5.1 Classic CNN Model and ChannelSplitNet Model 38
5.1.1 Discussion 46
5.2 Extra Experiments 49
5.2.1 Discussion 50
6. DISCUSSION 51
6.1 Summary of Achievement 51
6.2 Limitations, Improvement and Future Work 51
7. CONCLUSION 52
8. REFERENCE 53
APPENDIX 56

Chapter 1

1. Introduction

1.1 Motivation

Plant diseases can be classified into two categories according to whether they are caused by organisms. About 85% of diseases are caused by organisms, and most of the organisms are fungi, bacteria and viruses. The symptoms of plant diseases are mainly discoloration, necrosis, decay, wilting and deformity, which are all visible by the naked eye. (Isleib, J. 2012) Plants that are invaded by pathogens can lead to reduced yields, economic losses, and even the elimination of high-quality varieties. Some diseased crops can also cause human or animal poisoning. However, most of these diseases are infectious, and a wide range of plants may be affected.

The key step to stop transmission is timely detection and treatment. The traditional way of identification is that experts rely on eyes and experience to estimate these diseases, which are slow, tiring, and subjective. However, experts are limited and expensive, and it is difficult to hire in areas with underdeveloped economies.

With the development of science and technology, computers have performed well in the field of vision. Deep learning technique has outstanding performance in computer vision, including image classification. If the computer can be applied to the identification of plant diseases, it will speed up the diagnosis, reduce costs, and convenient to use.

With the in-depth research and combination of deep learning techniques and computer vision, it has been applied to many fields. This project plans to apply deep learning to agriculture to make accurate and speedy classification of plant diseases.

1.2 Aim of Project

The aim of this project is to learn the knowledge of deep learning and use convolutional neural networks to complete the identification and classification of plant diseases dataset. It explores the classification performance of some classic CNN models for plant diseases, including construction, training, analysis and evaluation. Combining the advantages of these models, a new CNN architecture is designed for classification tasks and compared with classic models. A comparison experiment between a fully connected network and a CNN is carried out. Finally,

the ensemble model is used to classify plant diseases. This project also has extra experiments to investigate if convolutional layers can be used to replace the dense layer.

1.3 Report Structure

This report consists of eight chapters. **Chapter 1** states the motivation and aims of the project, and the structures of this report. **Chapter 2** provides the technical background on machine learning, image classification and neural networks, as well as the previous work on plant diseases. **Chapter 3** discusses the data processing and distribution of data. **Chapter 4** details some classic CNN models, a new CNN architecture and experiments related to it, and training and evaluation methods. **Chapter 5** evaluates the result, outlines the discussion of model comparisons. **Chapter 6** highlights the success, shortcomings and future work of this project. **Chapter 7** states the conclusions of this project. **Chapter 8** lists the references used in this report.

Chapter 2

2. Background

This chapter first discusses the basic concepts of artificial neural networks, machine learning and image classification. Then introduces the architecture of convolutional neural network and some basic operations of it. Meanwhile, the previous work of convolutional neural network and the related work of plant diseases classification are reviewed.

2.1 ANN, Machine Learning and Image Classification

Artificial Neural Network (ANN) has been developed rapidly in the past few years and have been widely used in computer vision, such as face recognition, image classification and object detection. These applications have been commercialized into people's side, which greatly facilitates people's life.

The idea of ANN can be traced back to 1940s (McCulloch et al., 1943). However, due to the limitation of technology and database, this method has not been well practiced.

ANN is a type of machine learning. Machine learning does not put every scene or action that needs to be completed in the code, but in a way without explicit programming. The computer learns by itself from the input data and extracts the key information to complete the task, so as to complete various complex tasks that are difficult for algorithms to solve. Experiments have shown that ANN performs much better than traditional machine learning models (Yilmaz et al., 2020), especially in the case of large amounts of data. This is why ANN has recently grown rapidly and will play an important role in the future.

Image classification belongs to supervised learning, which means that when given a set of images, each image has a corresponding label. The task of the computer is to extract and learn the key features from the training dataset and make it as accurate as possible, finally uses these features to complete the label prediction of unseen images.

An image is composed of many pixels, and each pixel is a dimension in image processing, and most images in life are multidimensional data. If the traditional machine learning model or fully connected neural network is used, this will cause too many parameters to be adjusted during the training process, resulting in a surge of computation, the huge memory requirements and dimensional disasters. In addition, if there is too much noise in the image, the result of the prediction will be affected.

2.2 Convolutional Neural Network

The proposed of convolutional neural network solves the problem of excessively large data dimensions. The basic architecture of CNN has been determined when it was invented, i.e., LeNet (LeCun et al., 1998), and it has not been changed much until now. CNN consists of three type of layers, which are convolutional layer, pooling layer and fully connected layer. When these layers are combined together, plus the input and output layers, these five layers form a simplified CNN architecture, as shown in **Figure 2.1** (O'Shea et al., 2015).

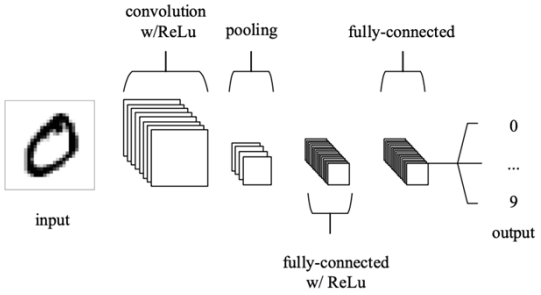


Figure 2.1: A simplified CNN architecture (O'Shea et al., 2015).

Subsequently, the proposal of AlexNet greatly improved the performance of the CNN model. AlexNet innovatively uses the ReLU function instead of the Sigmoid function to solve the vanishing gradient that may occur during back propagation. In order to prevent overfitting, dropout and data augmentation methods are used. Also use max pooling instead of average pooling.

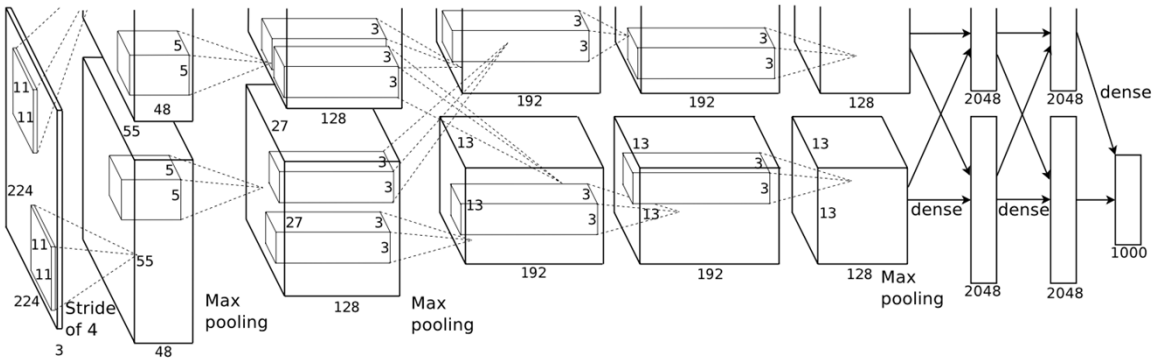


Figure 2.2: AlexNet Architecture of Two GPUs (Krizhevsky at el., 2012)

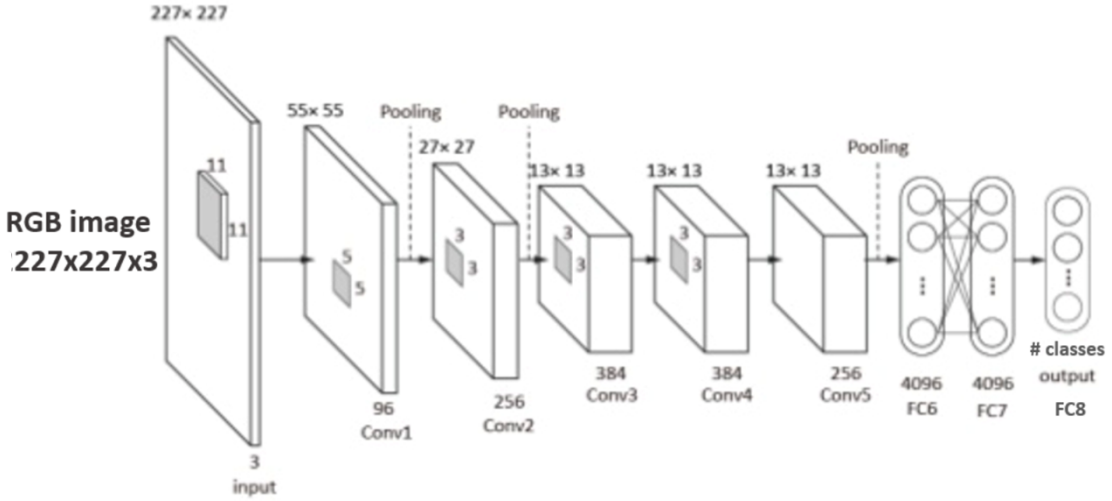


Figure 2.3: AlexNet Architecture of One GPU (Khvostikov et al., 2018)

AlexNet used two GPUs to accelerate the training, as shown in **Figure 2.2**. If the two GPUs are equivalent to one GPU (**Figure 2.3**), then the shape of the input image is changed from $224 \times 224 \times 3$ to $227 \times 227 \times 3$, and the channel number of the feature map of the same layer becomes the sum of the original two GPUs channels. In other words, the two GPUs accelerate training by separate channels.

The VGG16 network (Simonyan et al., 2015) is deeper than AlexNet, which means there are more convolutional layers. It uses multiple convolutional layers with the kernel size of 3×3 in series instead of using a single large convolution kernel, which has the advantages of reducing the number of parameters and has more nonlinear transformations than a single convolutional layer.

Then GoogLeNet invented the inception module. Inception module, as shown in **Figure 2.4**, it uses four different parts of the feature map that output by the previous layer to perform convolution or pooling operations, which can extract features as much as possible and improve the classification accuracy. In order to reduce the number of parameters and dimensions, 1×1 convolution is used to change the number of channels. Finally, the average pooling layer is used instead of the fully connected layer, thus greatly reducing the number of parameters.

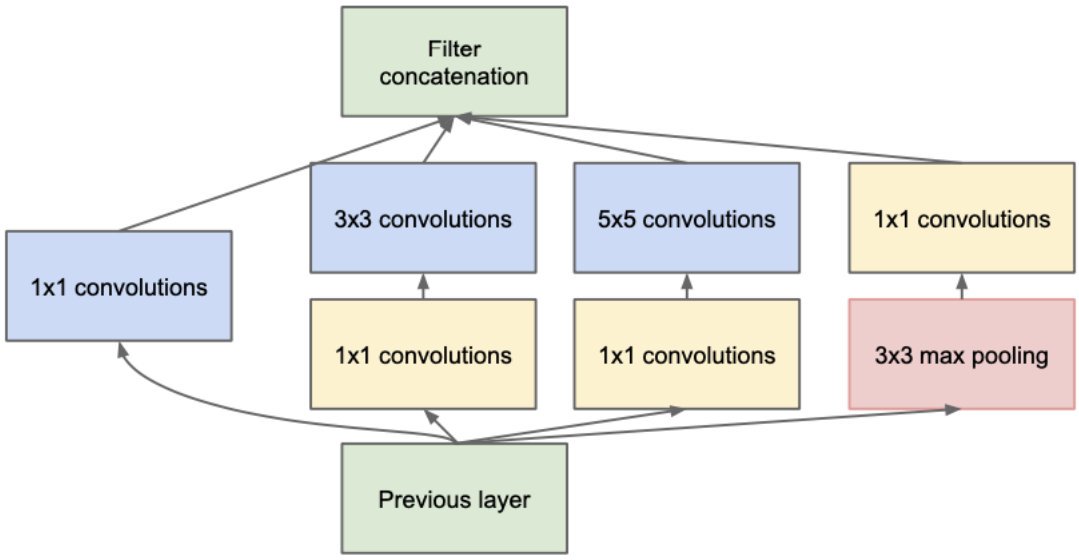
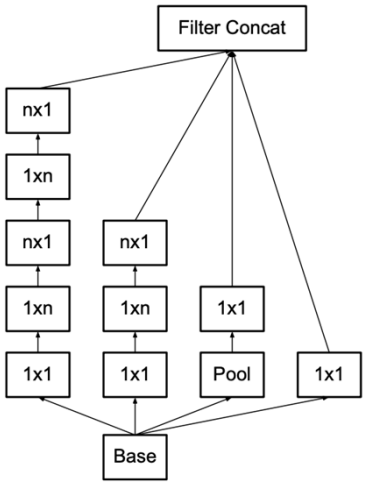


Figure 2.4: Inception module of GoogLeNet (Szegedy et al., 2015)

Inception V3 network has further improvements to the inception module. In order to further reduce the number of parameters, as shown in **Figure 2.5**, Inception V3 uses a combination of two convolution kernels of $N \times 1$ and $1 \times N$ to replace the convolutional layer with a convolution kernel of $N \times N$. And the experiment shows that this structure can also achieve a fairly performance after reducing the amount of calculation (Szegedy et al., 2016).



s

Figure 2.5: Inception module improved by Inception V3 (Szegedy et al., 2016)

2.2.1 The Convolution Operation

The purpose of the convolution operation is to extract the features from the image and output the feature maps through the operation of the convolution kernel and the feature image. In general, different convolution kernels are used to obtain different feature maps. During the

convolution operation, the convolution kernel slides on the feature graph, with a preset stride for each slide. The formula of feature map size after the convolution operation is:

$$Output\ size = \left(\frac{H - F_H + 2 \times P}{S} + 1, \frac{W - F_W + 2 \times P}{S} + 1\right) \text{ (cs231n.github.io, n.d.)}$$

$$where\ H = feature\ map\ height, W = feature\ map\ width, F_H = filter\ kernel\ height, F_W = filter\ kernel\ width, P = padding, S = strides$$

The one of the calculations of convolution operation is shown in **Figure 2.6**.

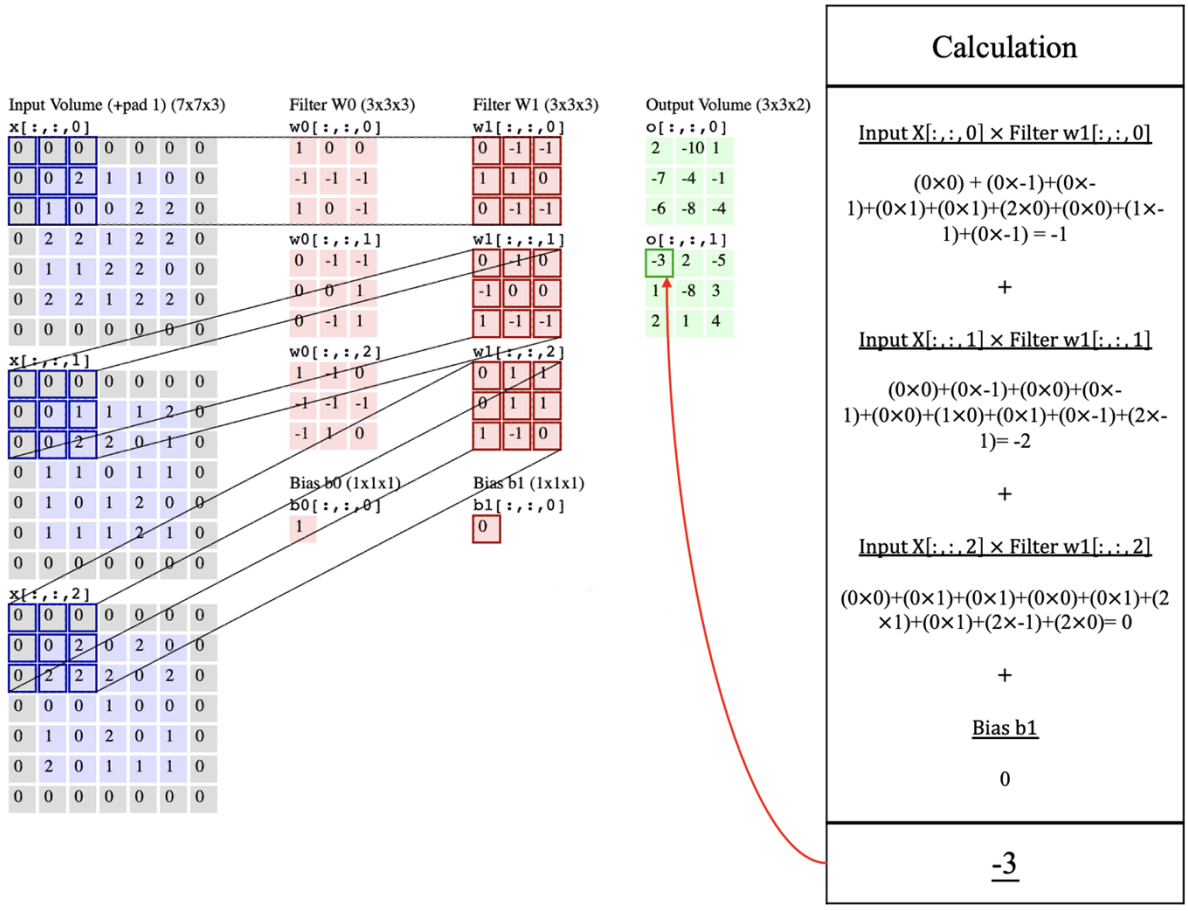


Figure 2.6: The Convolution Operation Procedure (cs231n.github.io, n.d.)

In **Figure 2.6**, the input volume is of size $W = H = 5$ (the input volume applies one padding on each side, then the size is from $5x5$ to $7x7$. This prevents the loss of the features at the edges.). Convolutional layer parameters are $F_H = F_W = 3, S = 2$. Therefore, the output size should be $(3,3)$. The output (green) is calculated by elementwise multiplying the highlighted input (blue) with the filter (red), summing it up, and then offsetting the result by the bias (cs231n.github.io, n.d.).

Figure 2.7 visualizes the features extracted by each convolutional layer. It can be seen that the feature maps become abstract after several convolution operations.

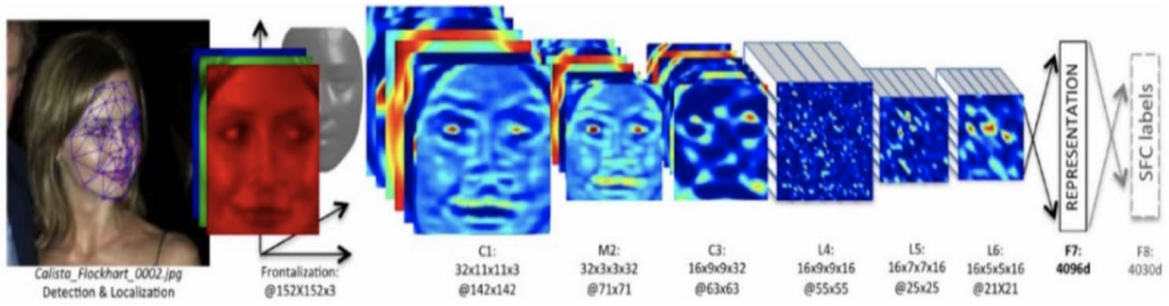


Figure 2.7: Visualize the Feature Maps (Albawi et al., 2017)

2.2.2 The Pooling Operation

The pooling layer is usually added after the convolutional layer. The main idea is down-sampling, reduce the dimension of the feature maps, remove the useless information and reduce the number of parameters. Pooling does not affect the number of channels. The most commonly used method is Max-Pooling. As shown in **Figure 2.8**, when Max-Pooling is performed in the top-left blocks (pink) with a 2x2 filter, all the other values are going to be discarded except for the largest one. Then filter moves 2 blocks and focus on top-right blocks (green). This indicates stride 2 is used in this Max-Pooling operation.

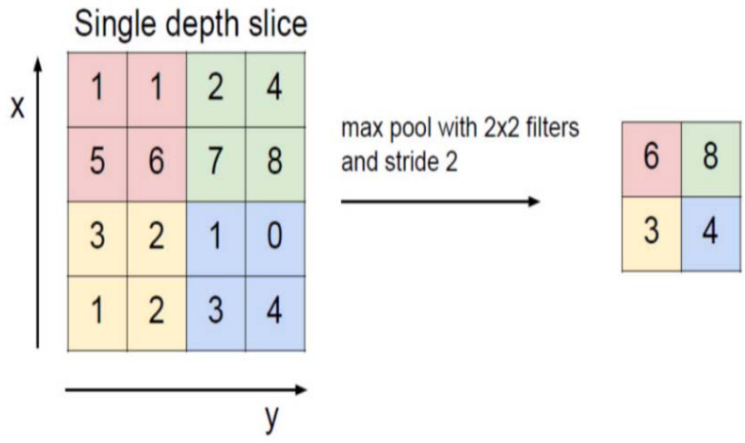


Figure 2.8: The Max-Pooling Operation (Albawi et al., 2017)

The Unusual stride 1 can be used to avoid down sampling. It should be considered that down sampling does not retain the position of the information. Therefore, it should only be applied when the presence of information (rather than spatial information) is important. (Albawi et al., 2017)

2.2.3 Activation Function

The activation function converts the input signal into an output signal, and the output signal is input to the next layer as an input. The prediction accuracy of the neural network depends on the number of layers used, and more importantly, the type of activation function used. The most commonly used activation function is the non-linear activation function. If a linear activation function is used, the network can only adapt to linear changes in the input. However, in the real world, errors have nonlinear characteristics. Therefore, nonlinear activation functions are better than linear activation functions in neural networks. (Sharma et al., 2020)

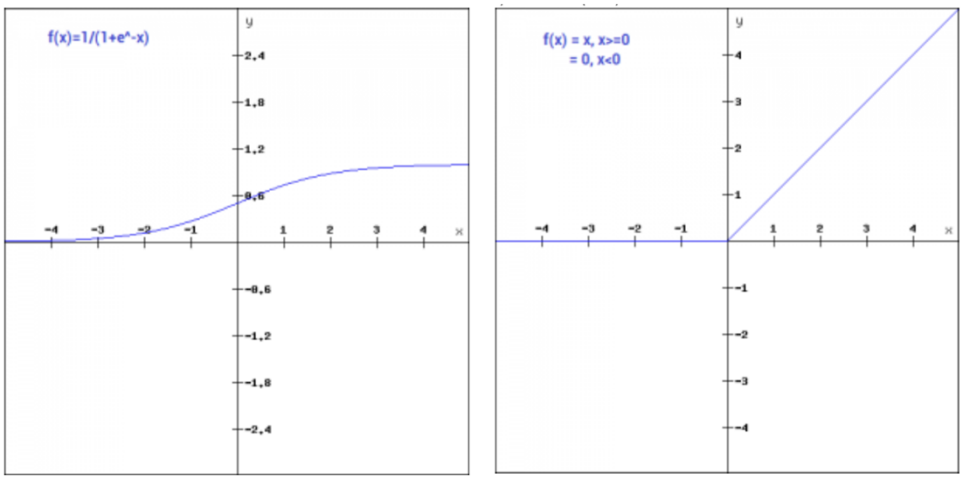


Figure 2.9: Sigmoid (Left) and ReLU (Right) Function (Sharma et al., 2020)

Sigmoid is a non-linear function and is the most widely used activation function. The sigmoid function converts values from 0 to 1. Can be defined as:

$$f(x) = \frac{1}{(1+e)^{-x}}$$

The sigmoid function is a continuously differentiable smooth s-shaped function. The derivative of the function is: $f(x) = 1 - sigmoid(x)$

ReLU stands for rectifier liner unit, which is a non-linear activation function, which has been widely used in neural networks. The advantage of the ReLU function is that it does not need to activate all neurons at the same time. This means that the neuron will be deactivated only when the output of the linear transformation is zero. The ReLU function is:

$$f(x) = \max(0,x)$$

In summary, the ReLU function is the most widely used function. In most cases, its performance is better than other activation functions. And it can only be used in the hidden layers rather than outer layer. However, due to vanishing gradient problem, i.e., gradient reaching the value zero, sigmoid functions are avoided. (Sharma et al., 2020)

2.2.4 Dropout

In deep learning, if the training set is small and the complexity of the model is high, which makes the prediction accuracy of the model in the training set high, but the accuracy in the test data low, which is over-fitted. Dropout sets a probability that each neuron has a probability of deactivating and not participating in the training of the network.

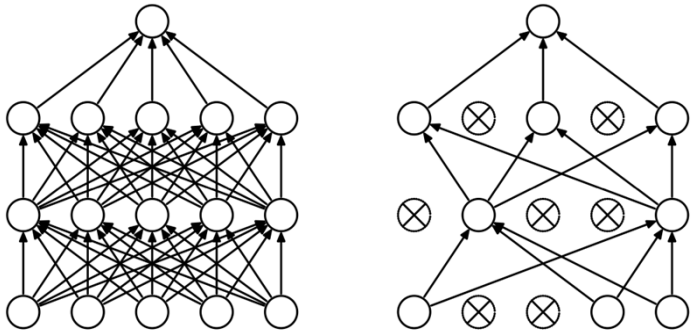


Figure 2.10: Standard Neural Network (Left) and Neural Network with Dropout (Right) (Srivastava et al., 2014)

The core idea of dropout is to take a large model that easily overfit, and repeatedly sample and train smaller sub-models from it. Because all sub-models share parameters with the large model, this process trains the large model, and then uses the large model for testing. Dropout improves the performance of neural networks in various application areas, this shows that dropout is very common, and not specific to a field. (Srivastava, 2013)

2.2.5 BatchNormalization

BatchNormalization was described by Ioffe in 2015. The principle is that each neuron is normalized, which can smooth the optimization landscape in the training stage of the network, reduce the dependence on dropout, and allow a higher learning rate to accelerate the training speed.

However, the disadvantage is that when the batch-size is too small, the mean and variance of the batch data are less representative. The reliability of the statistical information calculated by the BatchNormalization is getting worse, which will easily lead to an increase of the error rate.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 2.11: Batch Normalizing Transform, applied to activation x over a mini-batch.

2.2.6 Fully Connected and SoftMax Classification

After the features of the input images are extracted through the convolutional layer, the last step is to enter the fully connected network layer for classification. At this point, the parameter has been greatly reduced. Finally, the activation function of the output layer is SoftMax, which converts the output of the fully connected layer into a number between 0 and 1 through the formula in **Figure 2.12**, which is also called the probability of each category. The one with the highest probability is then output as the predicted category.

$$y_i^{(l)} = f\left(z_i^{(l)}\right),$$

$$\text{where } z_i^{(l)} = \sum_{i=1}^{m_i^{(l-1)}} w_{i,j}^{(l)} y_i^{(l-1)},$$

Figure 2.12: SoftMax function as output layer

where $w_{i,j}^{(l)}$ are the weights that should be tuned by the complete fully connected layer.

2.2.7 Optimizers

The optimizer is used to update and calculate the network parameters that affect the model training and output, try to approximate or reach the optimal value, finally minimizing (or maximizing) the loss function.

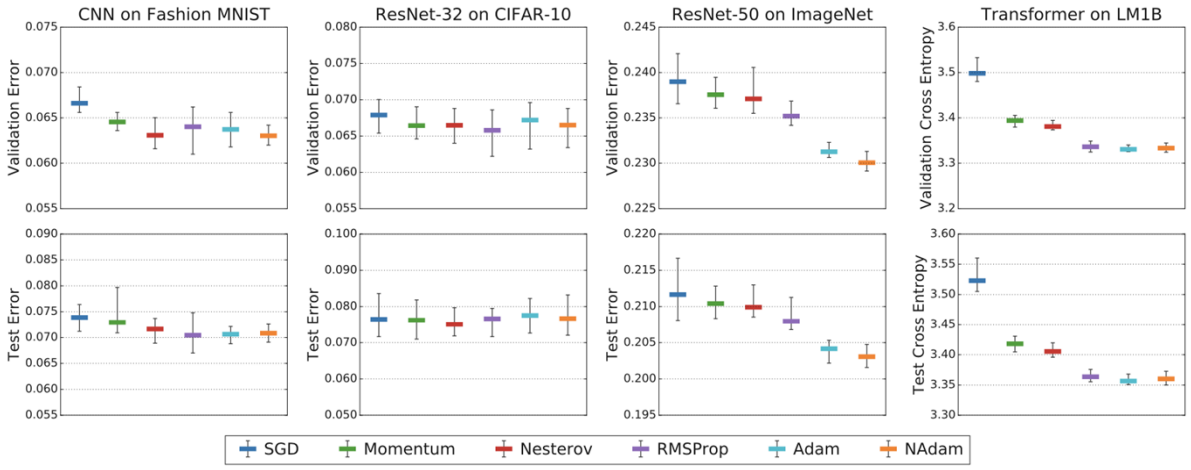


Figure 2.13: Comparation of Different Optimizers.

"In particular, we found that RMSPROP, ADAM, and NADAM never underperformed SGD, NESTEROV, or MOMENTUM under our most exhaustive tuning protocol." (Choi et al, 2020)

The experiments by (Choi et al, 2020), as shown in **Figure 2.13**, prove that in most cases, the performances of RMSPROP, ADAM and NADAM are better than others. And Adam becomes popular as an optimizer in neural network.

2.3 Related Work

There are many kinds of plant diseases, and the traditional way of identification is that experts rely on observation and experience to distinguish the diseases, which is slow and subjective. The products of plants are one of the main food sources for human beings. If plants get sick, it will affect the yield. It is necessary to make rapid and accurate identification of plant diseases and carry out corresponding treatments. There are already many works that use machine learning techniques to classify or detect plant diseases based on images. The following is a summary of some research on the classification of plant diseases.

Plant Disease Classification: A Comparative Evaluation of Convolutional Neural Networks and Deep Learning Optimizers (Saleem et al., 2020)

This article uses a large number of common deep learning frameworks to classify plant diseases (**Figure 2.14**). Then use a variety of deep learning optimization algorithms to further improve the accuracy of the model. Finally, it was found that Cascaded AlexNet with GoogLeNet, Improved GoogLeNet and Xception models performed best (**Figure 2.15**), and the two optimizers Adam and Adadelta can also be used to strengthen the research of other agricultural applications.

Deep Learning Architectures	Parameters (in Millions)	Epochs Required to Train the Model	Training Time (in Hours)	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss	Precision	Recall	F1-score
LeafNet	0.324 M	59	5.95	0.8590	0.7961	0.4563	0.6658	0.7946	0.7971	0.7958
VGG-16	138 M	59	38.13	0.8339	0.8189	0.5328	0.5651	0.8182	0.8194	0.8188
OverFeat	141.8 M	58	6.75	0.8995	0.8603	0.3201	0.4330	0.8592	0.8628	0.8610
Improved Cifar-10	2.43 M	58	6.08	0.9256	0.8974	0.2628	0.3205	0.8944	0.8960	0.8952
Inception ResNet v2	54.3 M	58	32.83	0.9551	0.9091	0.1530	0.3047	0.9075	0.9105	0.9089
Reduced MobileNet	0.5 M	55	11.72	0.9570	0.9278	0.1860	0.2442	0.9269	0.9267	0.9268
Modified MobileNet	0.5 M	53	6.38	0.9534	0.9297	0.1632	0.2385	0.9278	0.9265	0.9271
ResNet-50	23.6 M	55	26.33	0.9873	0.9423	0.0468	0.1923	0.9351	0.9358	0.9354
MLCNN	78 M	57	67.33	0.9583	0.9402	0.1335	0.1820	0.9386	0.9411	0.9398
Inception v4	41.2 M	59	52.92	0.9586	0.9489	0.1410	0.1828	0.9410	0.9466	0.9438
Improved GoogLeNet	6.8 M	53	9.67	0.9829	0.9521	0.0522	0.1038	0.9528	0.9539	0.9533
AlexNet	60 M	54	6.10	0.9689	0.9578	0.1046	0.1298	0.9563	0.9570	0.9566
DenseNet-121	7.1 M	56	28.75	0.9826	0.9580	0.0758	0.1323	0.9581	0.9569	0.9575
MobileNet	3.2 M	47	14.70	0.9764	0.9632	0.0903	0.1090	0.9624	0.9612	0.9618
Hybrid AlexNet with VGG (AgroAVNET)	238 M	54	49.90	0.9841	0.9649	0.0546	0.1078	0.9626	0.9674	0.9650
ZFNet	58.5 M	47	6.47	0.9752	0.9717	0.0746	0.1139	0.9746	0.9751	0.9748
Cascaded AlexNet and GoogLeNet	5.6 M	57	6.5	0.9931	0.9818	0.0229	0.0592	0.9749	0.9751	0.9750
Xception	22.8 M	34	56.28	0.9990	0.9798	0.0140	0.0621	0.9764	0.9767	0.9765

Figure 2.14: Comparison of Variety Models (in the order of the lowest to the highest F1-score) (Saleem et al., 2020)

Optimizers	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss	Precision	Recall	F1-score
Cascaded AlexNet with GoogLeNet							
SGD	0.9931	0.9818	0.0229	0.0592	0.9749	0.9751	0.9750
RMSProp	0.9894	0.9757	0.0482	0.1479	0.9746	0.9613	0.9679
Adagrad	0.9956	0.9824	0.0153	0.0547	0.9815	0.9782	0.9798
Adamax	0.9990	0.9859	0.0029	0.0574	0.9828	0.9795	0.9811
Adam	0.9989	0.9857	0.0039	0.0750	0.9836	0.9836	0.9836
Adadelta	0.9993	0.9873	0.0024	0.0696	0.9846	0.9856	0.9851
Improved GoogLeNet							
SGD	0.9829	0.9521	0.0522	0.1038	0.9528	0.9539	0.9533
RMSProp	0.9723	0.9685	0.1780	0.2272	0.9692	0.9666	0.9679
Adagrad	0.9889	0.9718	0.0350	0.0930	0.9651	0.9618	0.9634
Adamax	0.9998	0.9847	8.782×10^{-4}	0.0875	0.9792	0.9826	0.9809
Adam	0.9992	0.9904	0.0026	0.0434	0.9859	0.9872	0.9864
Adadelta	0.9991	0.9905	0.0022	0.0567	0.9828	0.9879	0.9861
Xception							
SGD	0.9990	0.9798	0.0140	0.0621	0.9764	0.9767	0.9765
RMSProp	0.9998	0.9924	6.922×10^{-4}	0.0433	0.9877	0.9920	0.9900
Adagrad	0.9987	0.9621	0.0164	0.1460	0.9682	0.9505	0.9593
Adamax	1.0000	0.9889	0.0012	0.0415	0.9902	0.9874	0.9888
Adam	1.0000	0.9981	6.890×10^{-4}	0.0178	0.9981	0.9975	0.9978
Adadelta	1.0000	0.9906	8.407×10^{-4}	0.0364	0.9926	0.9887	0.9906

Figure 2.15: Performance of Varity Optimizers (Saleem et al., 2020)

Apple Leaf Diseases Recognition Based on An Improved Convolutional Neural Network (Yan et al., 2020)

This study improved the VGG16 to recognize apple diseases. In improved-VGG16, the BatchNormalization layer, global average pooling layer and fully connected layer are added. In most cases, the accuracy of improved-VGG16 is better than other models (**Figure 2.16**). The experimental result is that the improved version of VGG16 has an accuracy of 99.01%, which is 6.3% higher than the classic VGG16, while the parameters are reduced by 119,534,592.

Classes	Model	Precision	Recall	F1-Score	Accuracy
Healthy	AlexNet	95.27%	95.83%	95.55%	95.83%
	ResNet34	98.14%	94.05%	96.05%	94.05%
	GoogleNet	98.80%	98.21%	98.51%	98.21%
	VGG16	93.14%	97.02%	95.04%	97.02%
	Alex + Inception	98.80%	98.21%	98.51%	98.21%
	Our Work	**98.82%**	**99.40%**	**99.11%**	**99.40%**
Scab	AlexNet	90.70%	76.47%	82.98%	76.47%
	ResNet34	84.21%	94.12%	88.89%	94.21%
	GoogleNet	97.96%	94.12%	96.00%	94.12%
	VGG16	90.91%	78.43%	84.21%	78.43%
	Alex + Inception	97.87%	90.20%	93.88%	90.20%
	Our Work	**98.04%**	**98.04%**	**98.04%**	**98.04%**
Frogeye Spot	AlexNet	92.06%	96.67%	94.31%	96.67%
	ResNet34	98.36%	**100%**	**99.17%**	96.67%
	GoogleNet	96.77%	**100%**	98.36%	**100%**
	VGG16	94.92%	93.33%	94.12%	93.33%
	Alex + Inception	95.24%	**100%**	97.56%	100%
	Our Work	**100%**	98.00%	99.16%	98.33%
Cedar rust	AlexNet	86.67%	**100%**	92.86%	**100%**
	ResNet34	**100%**	**100%**	**100%**	**100%**
	GoogleNet	92.59%	96.15%	94.34%	96.15%
	VGG16	92.59%	96.15%	94.34%	96.15%
	Alex + Inception	89.29%	96.15%	92.59%	96.15%
	Our Work	**100%**	**100%**	**100%**	**100%**

Figure 2.16: Comparison of Different Model (Out Work = Improved VGG16) (Yan et al., 2020)

Leaf Image based Plant Disease Identification using Color and Texture Features (Ahmed et al., 2021)

This research not only used neural networks, but also used traditional machine learning methods to train image-based plant disease recognition models. This model has two stages. First, the leaf images are divided into healthy and diseased categories, and then in the second stage, only unhealthy leaves are identified. An ensemble of classifiers such as SVM, K-NN, Naïve Bayes, random forest and ANN are trained on the selected feature set and select the classifier with the best performance. Among them, the Support Vector Machine has the best performance with a polynomial kernel of degree three. The result of final model is that the classification accuracy of diseased and healthy plant is 91.40% and for disease category is 82.47%.

Chapter 3

3. Data

An outstanding model is based on high-quality data. This chapter is about the data processing, including data integration, image resize, data allocation and data augmentation technology.

3.1 Data Integration

In the original Plant Diseases Dataset (Bhattarai, S., 2019), it has 38 categories with 70,296 images for training, 17,573 images for validation and 33 images for test. For the test set, this is a very small number, which does not reflect the performance of a model very well. For validation sets, there are too many images generated from the training set using data augmentation technique. That is, for a model, there are too many images in the validation set that have already been seen during training, although the size and orientation of the objects in the images may different, it's still hard to give a correct indication of the model's performance. In order to overcome this issue, all the images (include training, validation and test) are put together as new, delete the augmented images, then prepare for re-dividing the training set, validation set and test set.

3.2 Image Resize

The raw data has shape of $256 \times 256 \times 3$. However, in order to facilitate the comparison of model performance, the image size needs to be resized to the input shapes of most common CNN architectures, namely $224 \times 224 \times 3$ or $227 \times 227 \times 3$. The shape $227 \times 227 \times 3$ is only for AlexNet because its architecture is equivalent to one GPU instead of the original two GPUs.

3.3 Train-Valid-Test split

In this resized unassigned dataset, each category is assigned at the scale of 65% for training, 15% for validation and 20% for test. The training set is used to train the CNN model. The validation set is to roughly check the performance of the current model after each training epoch. This is convenient to see whether it is currently underfit or overfit, and the accuracy of the validation set is used for early stopping. The test set is used for the final model evaluation.

3.4 Data Augmentation

After the allocation, the number of images in different training set categories is unbalanced. This is unfavorable for training. If there is not enough data for a category, it will lead to the insufficient ability of the classifier to describe sparse samples, and it is difficult to effectively classify these sparse samples. If the number of images in one category is much greater than that other categories, the classification boundary tends to be more inclined to this category, resulting in a shift of the classification boundary and poor performance of classification. Therefore, data augmentation is necessary. The augmented images are shown in **Figure 3.2**.

Figure 3.1: Sample Image

Figure 3.2: Augmented Images

Data augmentation is also performed on the validation set. First of all, 100 images from training set are generated to validation set. This is to check the performance of the model on the data that has already seen. Then generate data from validation set to make sure each category has [500,600] images.

Now the data is ready for training and evaluation. **Figure 3.3** is some sample images in training set, and **Figure 3.4** is the quantity information of the dataset.

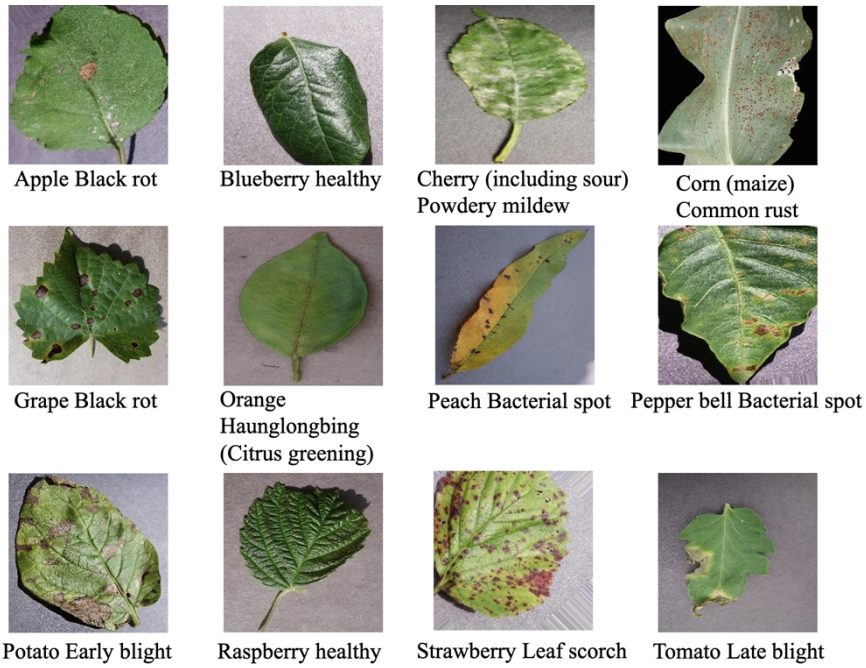


Figure 3.3: Sample Images in Training Set

Plant Name	Condition	Training Samples	Validation Samples	Test Samples
Apple	Apple Scab	2,200	520	128
	Black Rot	2,200	570	124
	Cedar Apple Rust	2,199	528	59
	Healthy	2,200	566	328
Blueberry	Healthy	2,200	522	301
Cherry (Including Sour)	Healthy	2,199	572	170
	Powdery Mildew	2,200	526	210
Corn(Maize)	Cercospora/Gray Leaf Spot	2,200	548	103
	Common Rust	2,199	588	241
	Healthy	2,200	568	232
	Northern Leaf Blight	2,200	592	197
Grap	Black Rot	2,200	538	235
	Esca (Black Measles)	2,200	520	276
	Healthy	2,200	502	84
	Leaf Blight (Isariopsis Leaf Spot)	2,200	563	215
Orange	Huanglongbing (Citrus Greening)	2,200	578	502
Peach	Bacterial Spot	2,200	546	458
	Healthy	2,200	513	72
Pepper/Bell	Bacterial Spot	2,200	564	199
	Healthy	2,200	529	295

Figure 3.4 (1): Quantity Information of Dataset

Plant Name	Condition	Training Samples	Validation Samples	Test Samples
Potato	Early Blight	2,200	586	205
	Healthy	2,198	519	32
	Late Blight	2,200	575	200
Raspberry	Healthy	2,199	571	74
Soybean	Healthy	2,200	580	504
Squash	Powdery Mildew	2,200	590	366
Strawberry	Healthy	2,199	500	91
	Leaf Scorch	2,200	588	221
Tomato	Bacterial Spot	2,200	520	424
	Early Blight	2,200	571	206
	Healthy	2,200	586	321
	Late Blight	2,200	556	381
	Leaf Mold	2,200	546	190
	Septoria Leaf Spot	2,200	576	353
	Spider Mites/Two-Spotted Spider Mite	2,200	501	335
	Target Spot	2,200	503	280
	Tomato Mosaic Virus	2,200	536	75
	Tomato Yellow Leaf Curl Virus	2,200	569	495

Figure 3.4 (2): Quantity Information of Dataset

In summary, there are totally 83,593 images for training, 20,926 images for validation and 9,182 images for test, distributed in 38 categories.

Chapter 4

4. Methodology

This chapter first discusses some common CNN architectures and they are trained to check the performance on plant diseases dataset. Then move to design a CNN, describe its architecture and training procedure. Finally focus on the training, evaluation and comparison methods of these models. In addition, an extra experiment on convolutional layer to replace the fully connected layer is outlined.

4.1 Classic CNN Models

In order to know the performance of the convolutional neural network on the plant diseases dataset, some classic CNN architectures are selected for training and evaluation. AlexNet, VGG16 and GoogLeNet are built and trained from scratch. VGG16, Inception V3 and ResNet50 use transfer training. Training procedures and architectures are discussed below.

4.1.1 Training from Scratch

There are three architectures are built from scratch. First is AlexNet, illustrated in **Figure 4.1**, uses five convolutional layers. First convolutional layer uses 11×11 kernel size to have a bigger receptive field, which can extract more features. Then followed by a 5×5 and three 3×3 convolutional layers. All these five layers uses ReLU activation function and BatchNormalization. Pooling occurs in the first, second and fifth layer, which reduce the feature maps size by half.

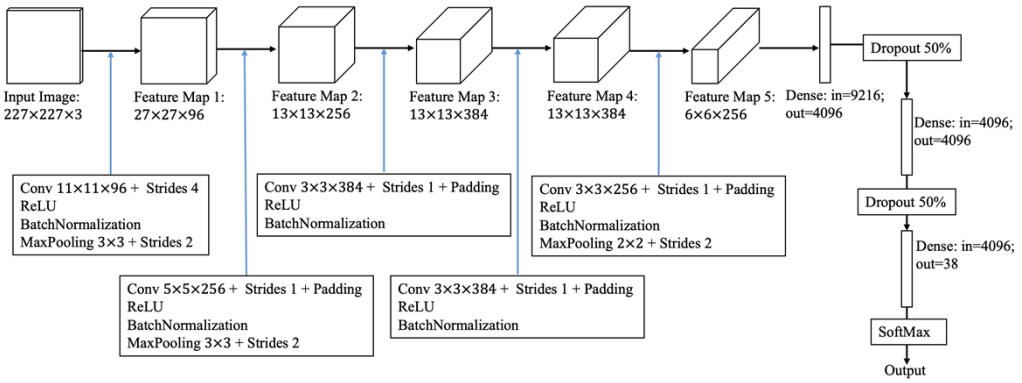


Figure 4.1: Architecture and Parameters of AlexNet

Then is VGG16, illustrated in **Figure 4.2**, has five convolution parts. In first two parts have two convolutional layers, and the rest three parts have three convolutional layers. All conv layers have same parameters with kernel size 3×3 followed by BatchNormalization and ReLU function except the filter numbers. Through the stacking of small kernel size of convolutional layers, it is used to replace large kernel size to obtain the same receptive field. For example, the receptive field of two 3×3 kernel size stacks and a 5×5 kernel size is the same, three conv layers with 3×3 kernel size stacks instead of 7×7 kernel size.

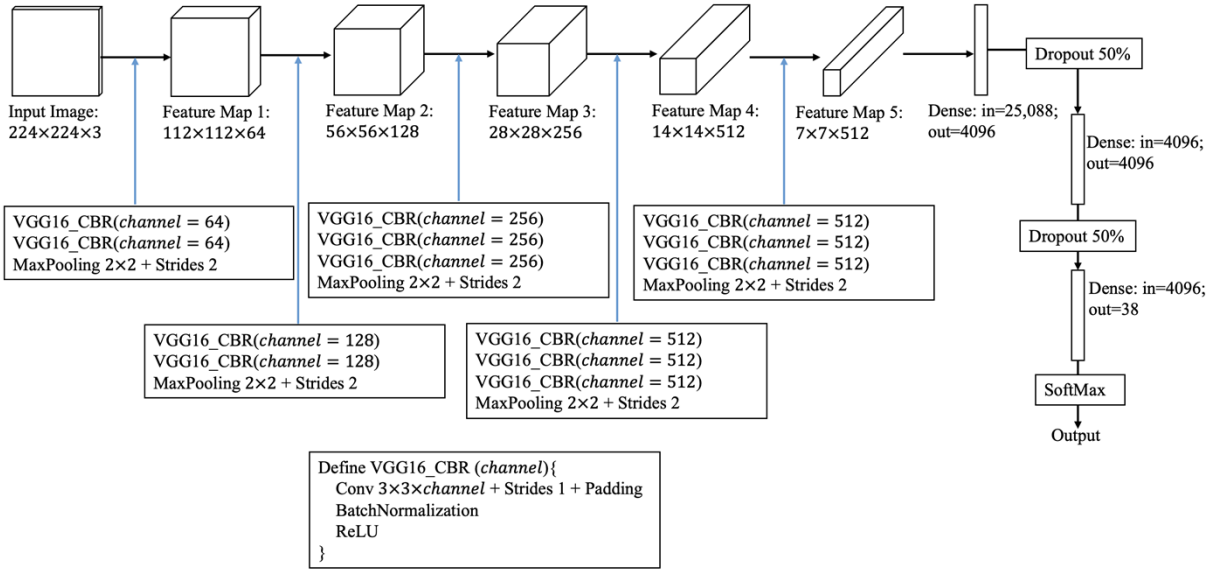


Figure 4.2: Architecture and Parameters of VGG16

The last one is GoogLeNet, illustrated in **Figure 4.3**, after two convolution parts and a pooling layer, the feature map enters the inception module (Inception Module shown in **Figure 2.4**). There are two auxiliary output layers after Feature Map 7 and Feature Map 10. This is to prevent the vanishing gradient due to the network being too deep and these two output layers are not involved in prediction. After the Feature Map 5 and Feature map 11, the pooling operation is performed, and the size of the feature map is halved.

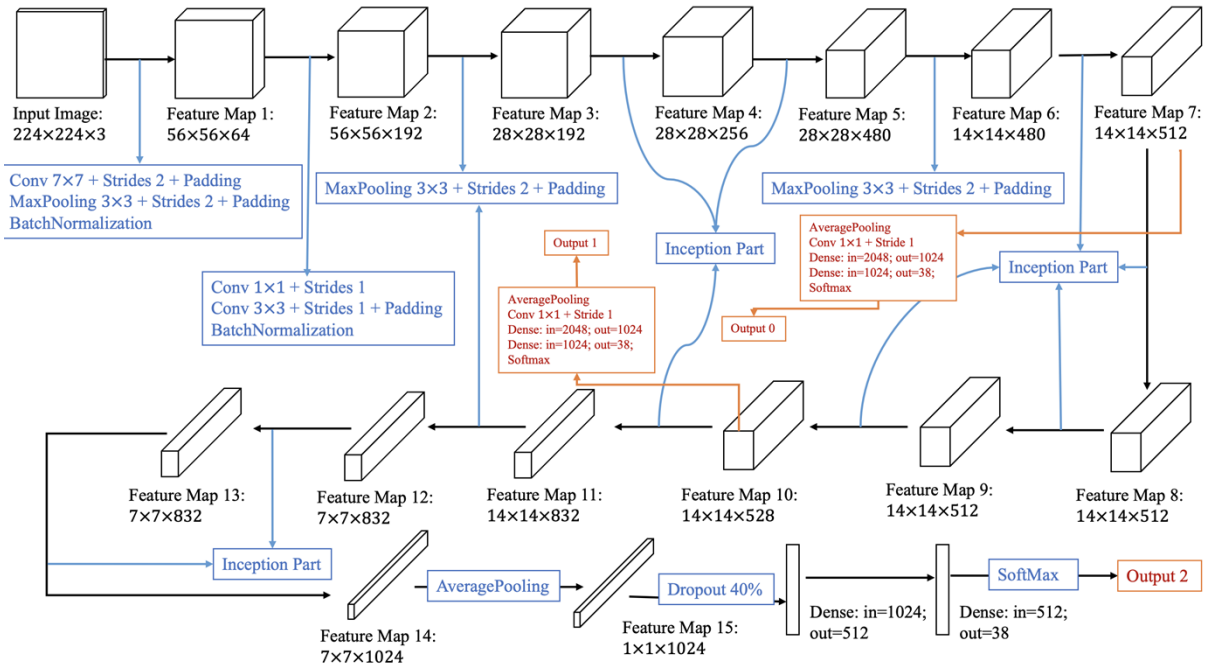


Figure 4.3: Architecture and Parameters of GoogLeNet

In summary, Googlenet has the least trainable parameters, which are 10,925,474. Alexnet has 58,439,782 trainable parameters, and VGG16 has 134,424,678 trainable parameters which is the most.

4.1.2 Transfer Learning

Transfer learning is a model that is trained on a task then use it to solve another similar problem. Generally, the input type of the model should be the same, and the output can be changed according to the needs of the task.

"This is typically understood in a supervised learning context, where the input is the same but the target may be of a different nature. For example, we may learn about one set of visual categories, such as cats and dogs, in the first setting, then learn about a different set of visual categories, such as ants and wasps, in the second setting." (Goodfellow et al., Page 534, Chapter 15, Deep Learning. 2016)

There are 26 pre-trained models on Keras. These models only have convolutional layers, and the pre-trained weights are provided by ImageNet. The pre-trained models of VGG16, Inception V3 and ResNet50 are selected, the weights of convolutional layers are frozen, and the appropriate fully connected layer and output layer are added for training.

4.2 Design a CNN Architecture

After investigating some classic CNN architectures, a new CNN architecture is designed and

named ChannelSplitNet, which combines the ideas of AlexNet, GoogLeNet and Inception V3. In order to improve model performance and reduce the number of parameters, ChannelSplitNet uses a convolutional layer to replace the original fully connected layer, that is, the Conv-Flatten output replaces the Dense output.

4.2.1 Experiment – Number of Convolutional Layers

An experiment is designed to explore how many convolutional layers are needed to obtain an acceptable performance on the plant disease dataset. In addition, this experiment also reveals the influence of the number of convolutional layers on the performance of the model.

Four architectures are designed for this experiment (**Figure 4.4**). First is no convolutional layer, that is, fully connected network. Then followed by one, two and four convolutional layers. After each convolution operation is the ReLU, Pooling and BatchNormalization layer. Finally coming by one dense layer and SoftMax output.

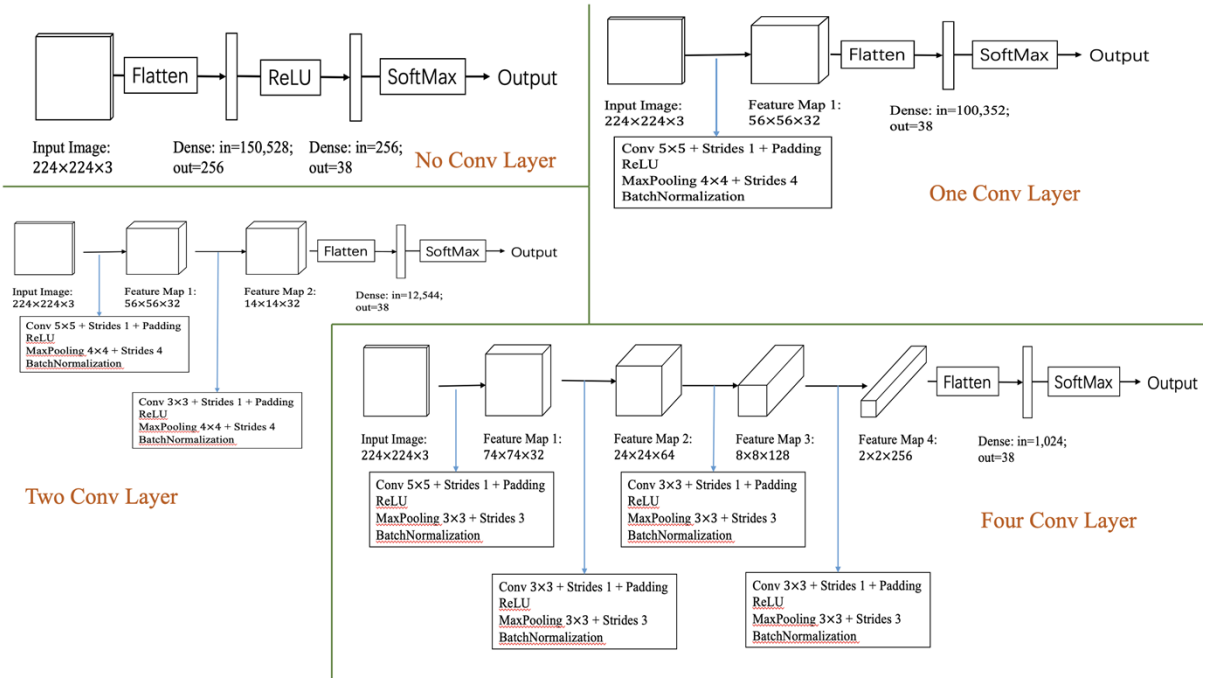


Figure 4.4: Architectures and Parameters of Conv Layer Experiment

4.2.2 ChannelSplitNet

As can be seen from **Section 2.1**, GoogLeNet proposes the inception module, so that the feature maps can extract multiple features from different parts. AlexNet splits the channel into two parts in order to use two GPUs for training. Combining the idea of splitting channels with the idea of the inception module, a new inception module can be created - use different convolution or pooling structures to extract features from different channels.

The structure of inception module with channel split idea is illustrated in Figure 4.5. From previous layer with feature map shape $a \times a \times c$, channel c is evenly divided into 4 parts. From 0 to one-quarter (not included) channels are trained by a 1×1 conv layer, which is mainly to change the number of channels. From one-quarter to two-quarters (not included) of the channels are trained by 3×3 conv layer. Then use 5×5 conv layer to extract features from two-quarter to three-quarter (not included) channels. Finally, the 3×3 pooling followed by a 1×1 conv layer perform on three-quarters to the last channel.

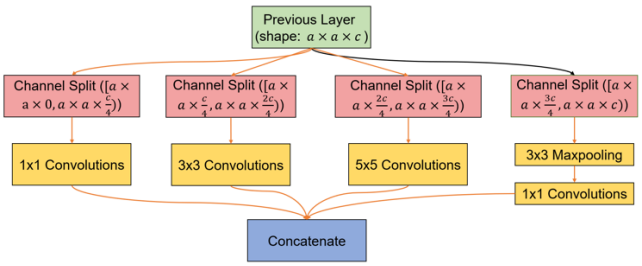


Figure 4.5: Naïve Version of Channel Split Inception Module

Figure 2.5 (In **Section 2.1**) introduces that the inception module has been further improved by Inception V3. By using two continuous $1 \times n$ and $n \times 1$ conv layers to replace $n \times n$ convolutional layers, the parameters are reduced. This improvement can also be applied to channel split inception module, as shown in **Figure 4.6**, 5×5 conv layer is changed to two continuous 1×5 and 5×1 conv layers. This is the Architecture 1 of Channel Split Inception Module.

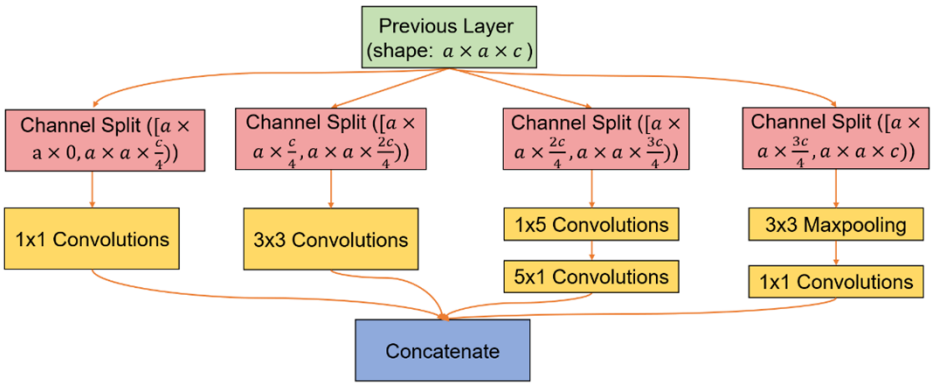


Figure 4.6: Channel Split Inception Module Architecture 1

In order to increase the non-linearity of the network, the second split channel structure is designed (**Figure 4.7**). It just flips the channel separation part (pink) in structure 1. This makes the four parts of the channels use different convolution or pooling operation to extract features.

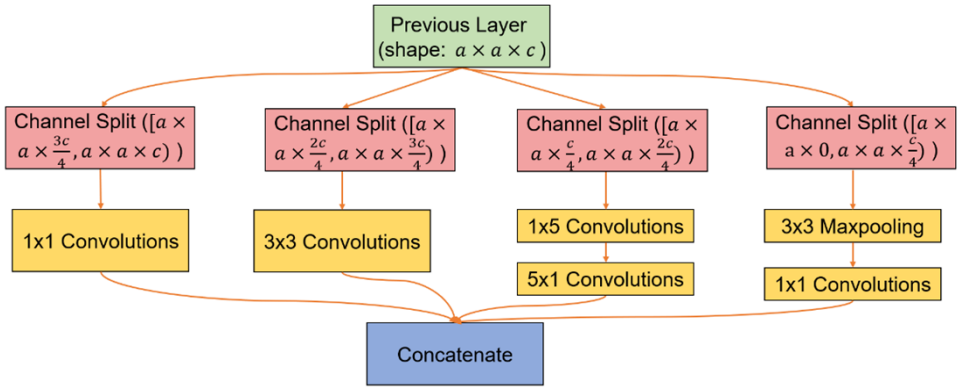


Figure 4.7: Channel Split Inception Module Architecture 2

The complete ChannelSplitNet architecture is shown in **Figure 4.8** and some details are shown in **Figure 4.9**. The input image first passes through two 5×5 convolutional layers and a pooling layer, then enters the Inception Architecture 1 and the pooling layer. Next go through Architecture 2 followed by a pooling layer. After that, the feature map re-enters Architecture 1 and the pooling layer. At the end it passes through 1×1 and 3×3 convolutional layer. The feature map at this time concatenate with other two outputs of Architecture 1 and Architecture 2, that is, the outputs of feature map generated by first Inception Architecture 1 and Inception Architecture 2 both pass through the 1×1 and 3×3 convolutional layer followed by the pooling layer. Finally, after concatenating and a 3×3 convolutional layer followed by the pooling layer, feature map enters dense layer for classification.

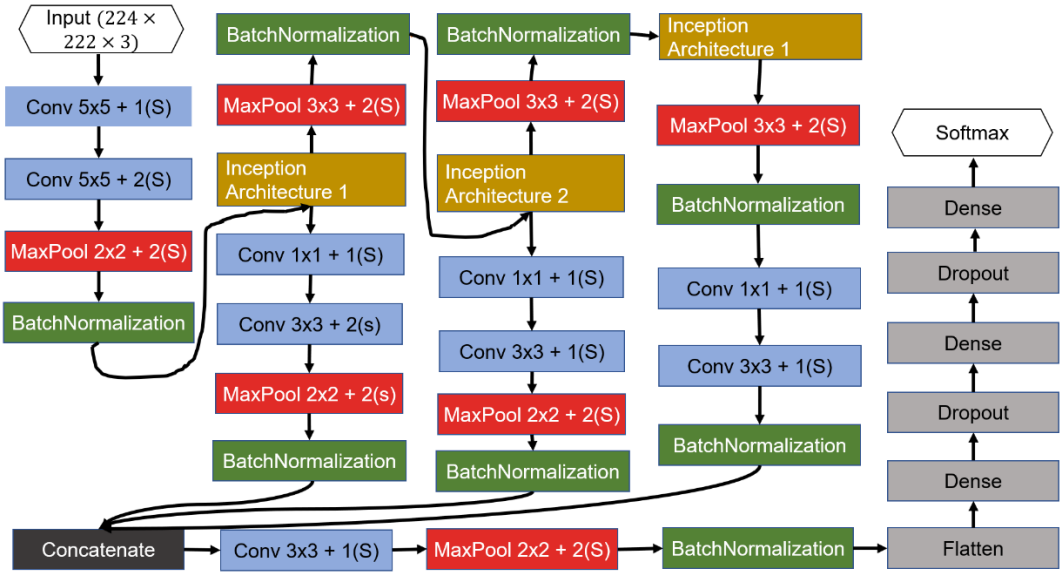
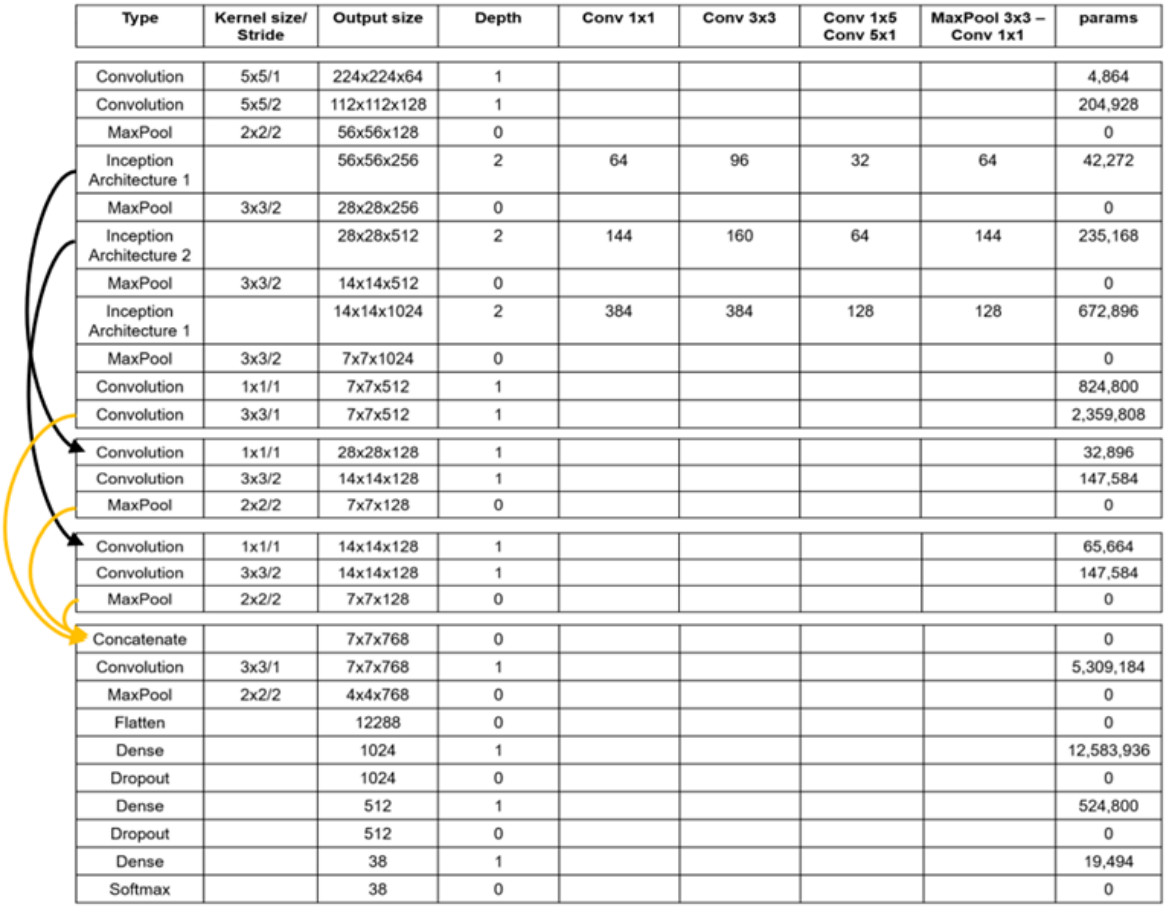


Figure 4.8: Complete Architecture of ChannelSplitNet



Type	Kernel size/ Stride	Output size	Depth	Conv 1x1	Conv 3x3	Conv 1x5 Conv 5x1	MaxPool 3x3 – Conv 1x1	params
Convolution	5x5/1	224x224x64	1					4,864
Convolution	5x5/2	112x112x128	1					204,928
MaxPool	2x2/2	56x56x128	0					0
Inception Architecture 1		56x56x256	2	64	96	32	64	42,272
MaxPool	3x3/2	28x28x256	0					0
Inception Architecture 2		28x28x512	2	144	160	64	144	235,168
MaxPool	3x3/2	14x14x512	0					0
Inception Architecture 1		14x14x1024	2	384	384	128	128	672,896
MaxPool	3x3/2	7x7x1024	0					0
Convolution	1x1/1	7x7x512	1					824,800
Convolution	3x3/1	7x7x512	1					2,359,808
Convolution	1x1/1	28x28x128	1					32,896
Convolution	3x3/2	14x14x128	1					147,584
MaxPool	2x2/2	7x7x128	0					0
Convolution	1x1/1	14x14x128	1					65,664
Convolution	3x3/2	14x14x128	1					147,584
MaxPool	2x2/2	7x7x128	0					0
Concatenate		7x7x768	0					0
Convolution	3x3/1	7x7x768	1					5,309,184
MaxPool	2x2/2	4x4x768	0					0
Flatten		12288	0					0
Dense		1024	1					12,583,936
Dropout		1024	0					0
Dense		512	1					524,800
Dropout		512	0					0
Dense		38	1					19,494
Softmax		38	0					0

Figure 4.9: Details of ChannelSplitNet

In summary, ChannelSplitNet has 3 inception modules, 9 convolutional layers (conv layers in inception modules not included), 7 MaxPooling layers and 3 fully connected layers. There are 22,882,790 trainable parameters in total.

4.2.3 Improvement of ChannelSplitNet

From the "param" column of **Figure 4.9**, it can be seen that there are 13,128,230 parameters in dense layer, which account for about 57% of entire network. Therefore, for the purpose of improving ChannelSplitNet, one can start with optimizing the number of parameters, that is, optimizing the dense layer. The most straightforward way is to remove the redundant dense layer, and only keep one dense layer. In this case, parameters in dense layer becomes 466,982, which reduced by 96% compared to the original dense layers.

The convolutional layer is used to replace this dense layer. As shown in **Figure 4.10**, the feature map shape before the Flatten layer is $4 \times 4 \times 768$. Then it can be trained by a 4×4 conv layer with 4 strides and 38 filters. Through this layer, a $1 \times 1 \times 38$ feature map is generated. Then after Flatten, it can be classified by SoftMax. The parameters of this structure are $4 \times 4 \times 768 \times 38 + 38 = 466{,}982$ which is exactly the same as one dense layer. This structure is called Conv-Flatten (CF) output.

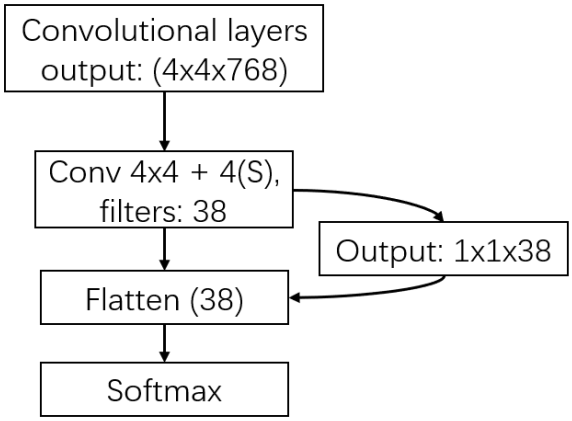


Figure 4.10: Conv-Flatten Output Structure

The schematic diagram of Conv-Flatten structure working in the network is shown in **Figure 4.11**. The shape of the feature map output by the previous convolution part is $4 \times 4 \times 768$ (red box). Then 38 $4 \times 4 \times 768$ filters are used for convolution operation (blue box). Each convolution operation produces a number, hence 38 numbers are generated, that is, from x to y in the green box (x, y are all represent a number). Finally, after Flatten, these numbers are used to classify through the SoftMax layer. Theoretically, this structure works.

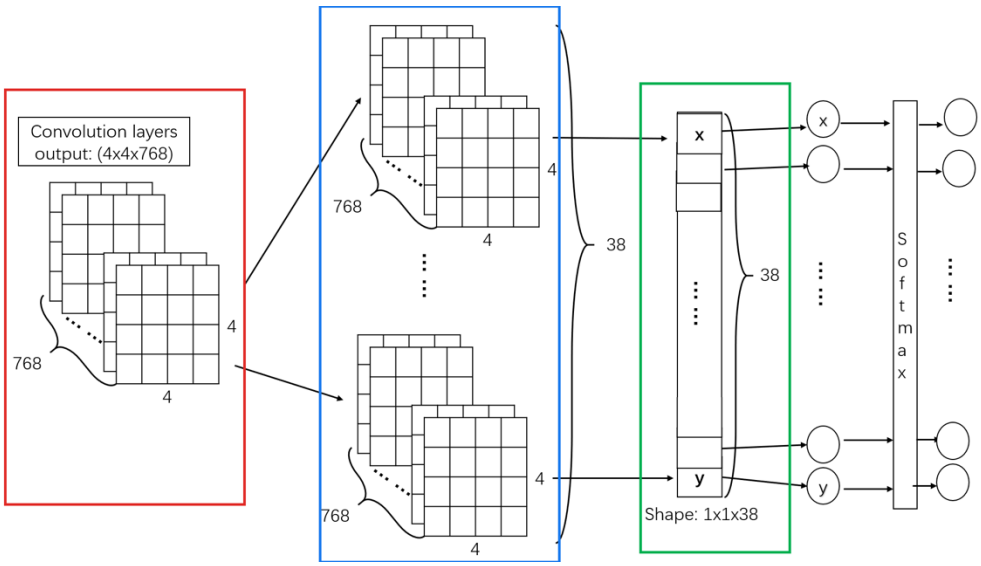


Figure 4.11: Conv-Faltten Output Structure Works in Network

Finally, the improved architecture of ChannelSplitNet is illustrated in **Figure 4.12**. The only difference is use one 4×4 conv layer instead of dense layers. It has 10,221,542 trainable parameters which is reduced by 55% compared with the first version.

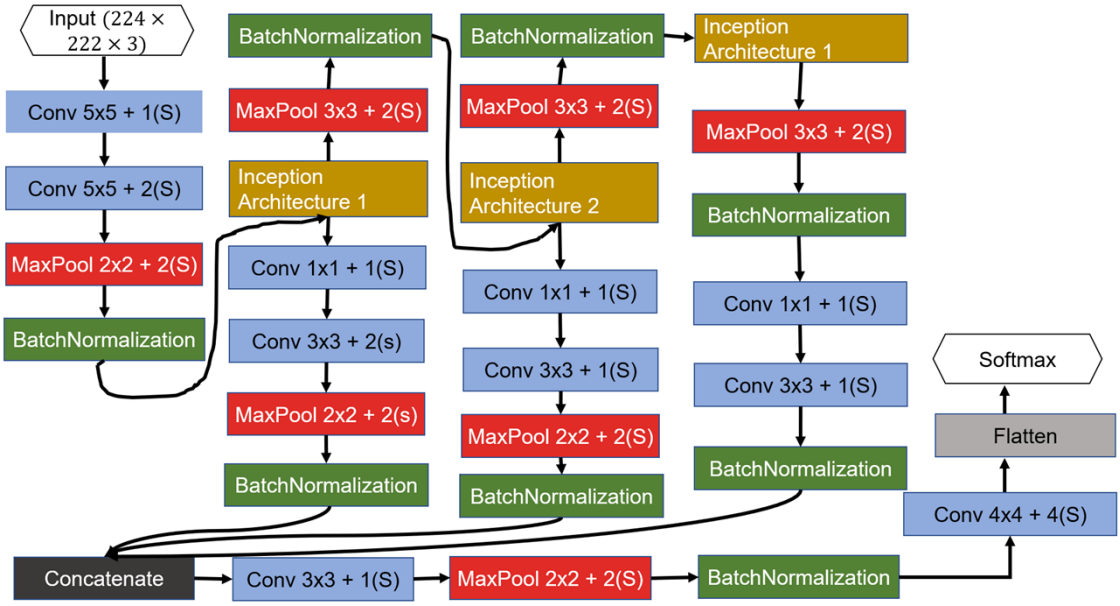


Figure 4.12: Improved Architecture of ChannelSplitNet

4.3 Ensemble of Models

In the *Data Science – Concepts and Practice (Second Edition)* book, the ensemble model is described as follows:

"Ensemble modelling is a process where multiple diverse base models are used to predict an outcome. The motivation for using ensemble models is to reduce the generalization error of the prediction. As long as the base models are diverse and independent, the prediction error decreases when the ensemble approach is used." (Kotu et al., Page 33, Section 2.3.4, Data Science – Concepts and Practice (Second Edition). 2019)

Therefore, in an attempt to get a better accuracy, several models above with the high test accuracy is used as base model to build an ensemble model. Each base model participates in the prediction, and the final prediction is determined by the majority vote.

4.4 Training Methodology

All these networks are trained using Nvidia GPU. With the aim of obtaining a better model, all training procedures use early stopping, that is, when the validation accuracy does not improve within 3 or 5 epochs, it is stopped. Or stop is performed manually if the validation accuracy does not improve significantly after many epochs (usually after 20 epochs). Then choose the model with the highest validation accuracy as the best model for evaluation.

The learning rate is also a very important hyperparameter. If the learning rate is too large, the model may skip the optimal solution and cannot converge. If it is too small, the training time

is longer. Therefore, the staged learning rate is used, that is, different learning rates are used in different epochs. Details are shown in **Figure 4.13**.

Model Name	Epoch Index									
	1	2	3	4	5	6	7	8	9	10…End
AlexNet	0.0001					0.00001			0.000001	
VGG16	0.0001			0.00001		0.000001				
GoogLeNet	0.0001					0.00001			0.000001	
Pretrained-ResNet 50	0.0001		0.00001			0.000001				
Pretrained-Inception V3	0.00001		0.000001			0.0000001				
Pretrained-VGG16	0.0001			0.00001		0.000001				
ChannelSplitNet – Dense Output	0.001		0.0001		0.00001		0.000001			
ChannelSplitNet – Conv-Flatten Output	0.001			0.0001			0.00001			0.000001

Figure 4.13: Staged Learning Rate Details

4.5 Evaluation Methodology

For each training described above, a table is built using the information of the confusion matrix first, and the accuracy, average precision, average recall and average F1-score will be calculated using this table for evaluation. Then the size of the model and the prediction time are considered to make a comprehensive evaluation. The evaluation is performed on test set.

Accuracy is the percentage of results that the model predicts correctly, the formula is:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

The average precision formulas are:

$$Class\ i\ precision = \frac{TP}{TP + FP}$$

$$Average\ Precision = \frac{\sum_{i=0}^{37} Class\ i\ precision}{38}$$

Average recall formulas are:

$$Class\ i\ recall = \frac{TP}{TP + FN}$$

$$Average\ Recall = \frac{\sum_{i=0}^{37} Class\ i\ recall}{38}$$

Average F1-score formulas are:

$$Class\ i\ F1 - score = 2 \times \frac{Class\ i\ precision \times Class\ i\ recall}{Class\ i\ precision + Class\ i\ recall}$$

$$Average\ F1 - score = \frac{\sum_{i=0}^{37} Class\ i\ F1 - score}{38}$$

where TP is the number of True Positive, FP is the number of False Positive, FN is the number of False Negative.

4.6 Extra Experiments – Conv-Flatten Output

In **Section 4.2.3** mentioned that Conv-Flatten output is used to replace the fully connected layer. In order to further explore the applicability and performance of this structure, two extra experiments are designed. These two experiments training procedure are both using early stop and staged learning rate.

4.6.1 Experiment One – Apply CF Output Structure to Other CNN Architecture

Conv-Flatten output structure is only used on ChannelSplitNet. This experiment explores whether this structure can also work on other CNN architecture. AlexNet, Pretrained-VGG16, Pretrained-ResNet50 and Pretrained-Inception V3 are chosen for this experiment. The architecture details of AlexNet are shown in **Figure 4.14** (modified part in red box), and architectures details of Pretrained-VGG16, Pretrained-ResNet50 and Pretrained-Inception V3 are shown in **Figure 4.15**. The learning rate details are shown in **Figure 4.16**.

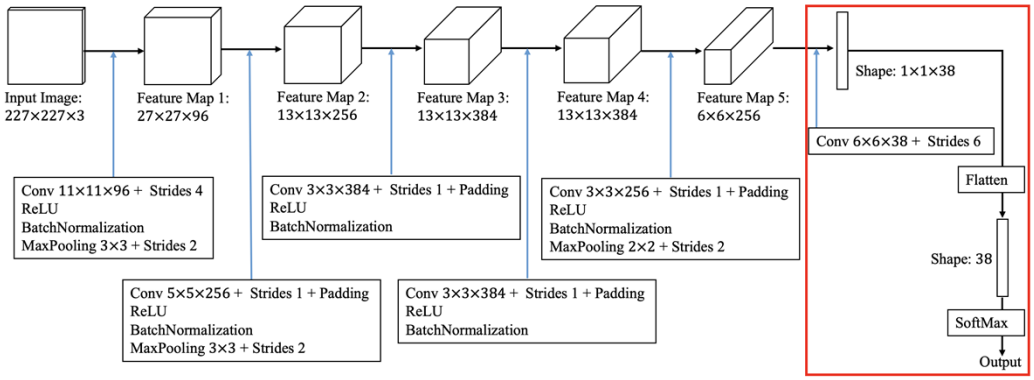


Figure 4.14: AlexNet Architecture for Experiment One

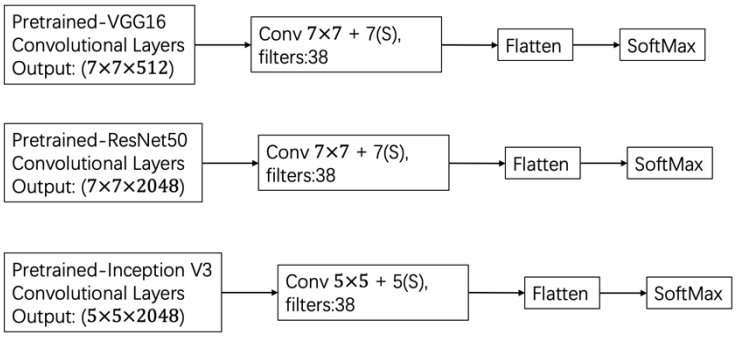


Figure 4.15: Pretrained-VGG16, Pretrained-ResNet50 and Pretrained-Inception V3 Architectures for Experiment One

Model Name	Epoch Index					
	1	2	3	4	5	6…End
AlexNet	0.0001		0.00001		0.000001	
Pretrained-VGG16	0.0001		0.00001		0.000001	
Pretrained-ResNet50	0.0001		0.00001		0.000001	
Pretrained-Inception V3	0.0001		0.00001		0.000001	

Figure 4.16: Staged Learning Rate for Experiment One

4.6.2 Experiment Two – Compare the Performance of Dense Output, CF Output and Improved CF Output

The previous CF output structure uses a convolutional layer with $n \times n$ kernel size and 38 filters to replace the dense layer. Inspired by the inception module structure of Inception v3, that is, a $n \times n$ conv layer can be replaced with two $n \times 1$ and $1 \times n$ convolutional layers to further reduce the parameters. This experiment aims to compare the performance of dense layer output, $n \times n$ conv layer and two $n \times 1$ and $1 \times n$ conv layers.

The experiment uses pretrained-VGG16 as convolution part, then connected with dense output

or CF output structure. The experiment is divided into two parts, the first part is one dense layer and its corresponding CF layer, architectures of this part is shown in **Figure 4.17**. The second part is two dense layers and its corresponding CF layers, architectures of this part is shown in **Figure 4.18**.

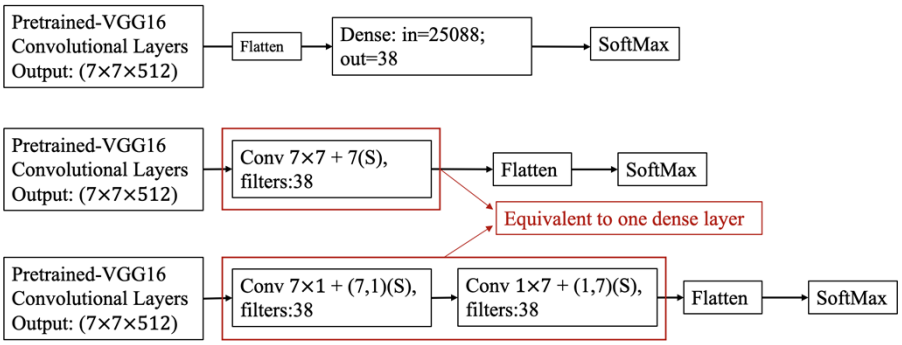


Figure 4.17: Architectures of Experiment Two (First Part)

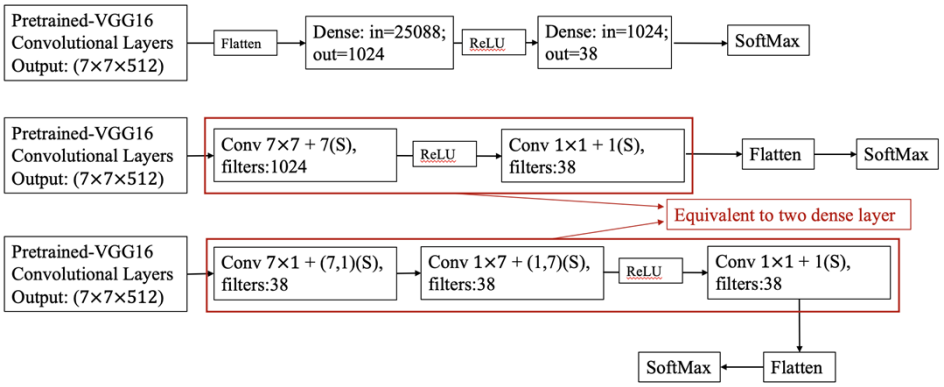


Figure 4.18: Architectures of Experiment Two (Second Part)

4.7 Summary

To summaries, in **Section 4.1**, three classic models (AlexNet, GoogLeNet, VGG16) are trained from scratch, and three classic models (VGG16, ResNet50, Inception V3) are trained using pretrained convolution part followed by appropriate dense layers. Then in **Section 4.2** details a new CNN architecture – ChannelSplitNet, and its improvement – using a convolutional layer to replace the dense layers (called Conv-Flatten structure). **This Section** also includes an experiment to explore the performance between different number of convolutional layers. **Section 4.3** discusses about ensemble. **Section 4.4** and **4.5** are the training and evaluation methods. Finally in **Section 4.6**, two extra experiments are designed to further investigate Conv-Flatten structure.

Chapter 5

5. Result and Analysis

5.1 Classic CNN Model and ChannelSplitNet Model

As discussed in **Section 4.4** and **4.5**, all results are obtained based on the test set, by using the model with the highest validation accuracy during training. These results are showed using a table generated by the confusion matrix. Each table has six columns, first is the index number from 0 to 37, as there are 38 categories. The second and third are plant names and corresponding diseases. The fourth column is the number of correct predictions by the model. The fifth column is the index numbers of mis-predicted categories.

The models trained from scratch, that is AlexNet, VGG16, GoogLeNet, ChannelSplitNet with dense output and ChannelSplitNet with Conv-Flatten output are illustrated in **Table 5.1**, **Table 5.2**, **Table 5.3**, **Table 5.4 and Table 5.5** respectively. And the models with pretrained conv parts weights, that is VGG16 and ResNet50 are illustrated in **Table 5.6** and **Table 5.7** respectively. Due to the poor performance of Inception V3, its result is not shown separately. Finally, a comparison table of these models and ensemble model that contains more details is shown in **Table 5.8** (TfS in table stands for Training from Scratch). In addition, the results of conv layers experiment (described in **Section 4.2.1**) are illustrated in **Table 5.9**.

Index	Plant Name	Condition	Correct Predictions	Misclassified Index List	Accuracy
0	Apple	Apple Scab	123	[1,3,4,16,31]	96.09%
1		Black Rot	124	[]	100%
2		Cedar Apple Rust	59	[]	100%
3		Healthy	322	[0,4,4,17,27,30]	98.17%
4	Blueberry	Healthy	301	[]	100%
5	Cherry (Including Sour)	Healthy	169	[24]	99.41%
6		Powdery Mildew	209	[29]	99.52%
7	Corn(Maize)	Cercospora/Gray Leaf Spot	97	[2,10,10,10,10,10]	94.17%
8		Common Rust	241	[]	100%
9		Healthy	232	[]	100%
10		Northern Leaf Blight	184	[7,7,7,7,7,7,7,7,7,7,7,7]	93.40%
11	Grap	Black Rot	232	[28,28,29]	98.72%
12		Esca (Black Measles)	247	[11,27]	99.28%
13		Healthy	84	[]	100%
14		Leaf Blight (Isariopsis Leaf Spot)	215	[]	100%
15	Orange	Huanglongbing (Citrus Greening)	500	[19,28]	99.60%
16	Peach	Bacterial Spot	454	[17,17,17,17]	99.13%
17		Healthy	72	[]	100%
18	Pepper/Bell	Bacterial Spot	198	[28]	99.50%
19		Healthy	293	[6,24]	99.32%
20	Potato	Early Blight	205	[]	100%
21		Healthy	31	[19]	96.88%
22		Late Blight	194	[20,20,29,31,32,35]	97.00%
23	Raspberry	Healthy	74	[]	100%
24	Soybean	Healthy	494	[3,5,9,19,19,19,19,34,34,35]	98.02%
25	Squash	Powdery Mildew	366	[]	100%
26	Strawberry	Healthy	91	[]	100%
27		Leaf Scorch	220	[31]	99.55%
28	Tomato	Bacterial Spot	420	[37,37,37,37]	99.06%
29		Early Blight	195	[17,28,31,31,31,31,31,33,35,37,37]	94.66%
30		Healthy	319	[8,33]	99.38%
31		Late Blight	366	[2,8,17,28,29,29,29,29,29,29,29,29,29,30]	96.06%
32		Leaf Mold	187	[3,31,34]	98.42%
33		Septoria Leaf Spot	349	[20,28,32,35]	98.67%
34		Spider Mites/Two-Spotted Spider Mite	334	[37]	99.70%
35		Target Spot	271	[3,30,33,34,34,34,34,34,36]	96.79%
36		Tomato Mosaic Virus	75	[]	100%
37		Tomato Yellow Leaf Curl Virus	493	[32,32]	99.60%

Table 5.1: AlexNet Result

Index	Plant Name	Condition	Correct Predictions	Misclassified Index List	Accuracy
0	Apple	Apple Scab	126	[3,16]	98.44%
1		Black Rot	124	[]	100%
2		Cedar Apple Rust	59	[]	100%
3		Healthy	324	[4,4,15,17]	98.78%
4	Blueberry	Healthy	300	[3]	99.67^
5	Cherry (Including Sour)	Healthy	169	[19]	99.41%
6		Powdery Mildew	210	[]	100%
7	Corn(Maize)	Cercospora/Gray Leaf Spot	100	[10,10,10]	97.08%
8		Common Rust	240	[31]	99.58%
9		Healthy	232	[]	100%
10		Northern Leaf Blight	187	[7,7,7,7,7,7,7,7,7,7]	94.92%
11	Grap	Black Rot	235	[]	100%
12		Esca (Black Measles)	275	[11]	99.64%
13		Healthy	84	[]	100%
14		Leaf Blight (Isariopsis Leaf Spot)	213	[11,11]	99.07%
15	Orange	Huanglongbing (Citrus Greening)	502	[]	100%
16	Peach	Bacterial Spot	458	[]	100%
17		Healthy	72	[]	100%
18	Pepper/Bell	Bacterial Spot	198	[28]	99.48%
19		Healthy	293	[23,24]	99.32%
20	Potato	Early Blight	204	[30]	99.51%
21		Healthy	30	[24,34]	93.75%
22		Late Blight	198	[29,35]	99.00%
23	Raspberry	Healthy	74	[]	100%
24	Soybean	Healthy	498	[4,5,21,34,34,35]	98.81%
25	Squash	Powdery Mildew	366	[]	100%
26	Strawberry	Healthy	91	[]	100%
27		Leaf Scorch	221	[]	100%
28	Tomato	Bacterial Spot	418	[33,35,37,37,37,37]	98.58%
29		Early Blight	196	[11,28,28,31,31,31,33,33,35,37]	95.15%
30		Healthy	319	[31,32]	99.38%
31		Late Blight	354	[0,15,16,20,22,22,22,22,22,22,22,22,22,22,22,22,22,29,29,29,29,29,29,30,30,34]	92.91%
32		Leaf Mold	189	[34]	99.47%
33		Septoria Leaf Spot	352	[0]	99.72%
34		Spider Mites/Two-Spotted Spider Mite	334	[35]	99.70%
35		Target Spot	274	[30,30,33,33,34,34]	97.86%
36		Tomato Mosaic Virus	75	[]	100%
37		Tomato Yellow Leaf Curl Virus	495	[]	100%

Table 5.2: VGG16 (Training from Scratch) Result

Index	Plant Name	Condition	Correct Predictions	Misclassified Index List	Accuracy
0	Apple	Apple Scab	127	[3]	99.22%
1		Black Rot	124	[]	100%
2		Cedar Apple Rust	59	[]	100%
3		Healthy	328	[]	100%
4	Blueberry	Healthy	299	[3,17]	99.34%
5	Cherry (Including Sour)	Healthy	170	[]	100%
6		Powdery Mildew	208	[7,29]	99.05%
7	Corn(Maize)	Cercospora/Gray Leaf Spot	91	[10,10,10,10,10,10,10,10,10,10,10]	88.35%
8		Common Rust	241	[]	100%
9		Healthy	232	[]	100%
10		Northern Leaf Blight	186	[7,7,7,7,7,7,7,7,7,7]	94.42%
11	Grap	Black Rot	232	[12,12,29]	98.72%
12		Esca (Black Measles)	276	[]	100%
13		Healthy	84	[]	100%
14		Leaf Blight (Isariopsis Leaf Spot)	212	[11,11]	98.60%
15	Orange	Huanglongbing (Citrus Greening)	500	[5,28]	99.60%
16	Peach	Bacterial Spot	455	[1,3,17]	99.34%
17		Healthy	71	[16]	98.61%
18	Pepper/Bell	Bacterial Spot	196	[1,1,16]	98.49%
19		Healthy	293	[5,17]	99.32%

Index	Plant Name	Condition	Correct Predictions	Misclassified Index List	Accuracy
20	Potato	Early Blight	205	[]	100%
21		Healthy	32	[]	100%
22		Late Blight	193	[20,21,21,29,29,30,31]	96.50%
23	Raspberry	Healthy	74	[]	100%
24	Soybean	Healthy	499	[3,19,34,35,35]	99.01%
25	Squash	Powdery Mildew	365	[11]	99.73%
26	Strawberry	Healthy	91	[]	100%
27		Leaf Scorch	221	[]	100%
28	Tomato	Bacterial Spot	420	[16,29,29,35]	99.06%
29		Early Blight	180	[11,18,20,28,28,28,28,28,31,31,31,31,31,31,31,31,31,31,31,33,33,35,35,35,36]	87.38%
30		Healthy	319	[31,35]	99.38%
31		Late Blight	363	[6,9,10,17,18,20,22,29,29,29,29,29,29,32,33]	95.28%
32		Leaf Mold	188	[33,33]	98.95%
33		Septoria Leaf Spot	347	[14,31,31,31,31,32]	98.30%
34		Spider Mites/Two-Spotted Spider Mite	325	[18,24,35,35,35,35,35,35,35,35]	97.01%
35		Target Spot	262	[24,28,28,30,30,30,33,33,33,34,34,34,34,34,34,34,34,36,]	93.57%
36		Tomato Mosaic Virus	75	[]	100%
37		Tomato Yellow Leaf Curl Virus	492	[28,28,34]	99.40%

Table 5.3: GoogLeNet Result

Index	Plant Name	Condition	Correct Predictions	Misclassified Index List	Accuracy
0	Apple	Apple Scab	113	[1,3,3,3,4,4,4,16,16,16,16,28,31,32,33]	88.28%
1		Black Rot	122	[5,12]	98.39%
2		Cedar Apple Rust	58	[28]	98.31%
3		Healthy	311	[0,4,4,4,4,4,4,4,4,4,4,16,17,17,24,29,34]	94.82%
4	Blueberry	Healthy	299	[3,3]	99.34%
5	Cherry (Including Sour)	Healthy	170	[]	100%
6		Powdery Mildew	208	[7,34]	99.05%
7	Corn(Maize)	Cercospora/Gray Leaf Spot	92	[10,10,10,10,10,10,10,10,10,10]	89.32%
8		Common Rust	241	[]	100%
9		Healthy	232	[]	100%
10		Northern Leaf Blight	178	[7,7,7,7,7,7,7,7,7,7,7,7,7,7,7]	90.36%
11	Grap	Black Rot	223	[12,12,12,12,12,12,12,12,14,28,28,33]	94.89%
12		Esca (Black Measles)	271	[11,11,11,11,11]	98.19%
13		Healthy	83	[6]	98.81%
14		Leaf Blight (Isariopsis Leaf Spot)	215	[]	100%
15	Orange	Huanglongbing (Citrus Greening)	499	[0,5,28]	99.40%
16	Peach	Bacterial Spot	443	[0,2,2,4,10,17,17,17,17,17,17,18,22,28,33]	96.72%
17		Healthy	72	[]	100%
18	Pepper/Bell	Bacterial Spot	196	[1,1,28]	98.49%
19		Healthy	292	[23,23,34]	98.98%
20	Potato	Early Blight	202	[29,31,33]	98.54%
21		Healthy	29	[19,34,34]	90.63%
22		Late Blight	190	[0,10,10,20,31,31,31,31,35]	95.00%
23	Raspberry	Healthy	74	[]	100%
24	Soybean	Healthy	488	[4,4,5,5,19,19,19,19,21,21,21,21,23,34,35]	96.83%
25	Squash	Powdery Mildew	361	[6,6,6,26,28]	98.63%
26	Strawberry	Healthy	91	[]	100%
27		Leaf Scorch	219	[10,31]	99.10%
28	Tomato	Bacterial Spot	402	[3, 15, 16, 24, 29, 29, 29, 29, 29, 29, 30, 31, 33, 35, 37, 37, 37, 37, 37, 37, 37, 37]	94.81%
29		Early Blight	168	[3, 3, 4, 11, 14, 15, 18, 22, 22, 28, 28, 28, 28, 28, 28, 31, 31, 31, 31, 31, 31, 33, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35, 35, 37]	81.55%
30		Healthy	320	[3]	99.69%
31		Late Blight	325	[0, 0, 0, 3, 6, 10, 10, 10, 14, 16, 16, 16, 20, 20, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 28, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 30, 32, 33, 34, 34, 35, 35, 37]	85.30%
32		Leaf Mold	181	[17,31,31,33,34,34,34,34,37]	95.26%
33		Septoria Leaf Spot	330	[2, 18, 20, 20, 20, 28, 28, 29, 31, 31, 31, 32, 32, 32, 32, 32, 32, 34, 34, 35, 35, 36]	93.48%
34		Spider Mites/Two-Spotted Spider Mite	325	[3.29,30,35,35,35,35,35,36]	97.01%
35		Target Spot	236	[0, 16, 28, 28, 29, 29, 29, 29, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 36]	84.29%
36		Tomato Mosaic Virus	73	[34,34]	97.33%
37		Tomato Yellow Leaf Curl Virus	487	[15,15,15,28,28,34,34,34]	98.38%

Table 5.4: ChannelSplitNet with Dense Output Result

Index	Plant Name	Condition	Correct Predictions	Misclassified Index List	Accuracy
0	Apple	Apple Scab	126	[3,3]	98.44%
1		Black Rot	124	[]	100%
2		Cedar Apple Rust	59	[]	100%
3		Healthy	323	[4,4,17,17,24]	98.48%
4	Blueberry	Healthy	300	[31]	99.67%
5	Cherry (Including Sour)	Healthy	169	[24]	99.41%
6		Powdery Mildew	209	[34]	99.52%
7	Corn(Maize)	Cercospora/Gray Leaf Spot	99	[10,10,10,10]	99.12%
8		Common Rust	213	[0,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,24]	88.38%
9		Healthy	232	[]	100%
10		Northern Leaf Blight	186	[2,7,7,7,7,7,7,7,7,31,31]	94.42%
11	Grap	Black Rot	233	[14,28]	99.15%
12		Esca (Black Measles)	275	[14]	99.64%
13		Healthy	84	[]	100%
14		Leaf Blight (Isariopsis Leaf Spot)	212	[11,11,11]	98.60%
15	Orange	Huanglongbing (Citrus Greening)	501	[28]	99.80%
16	Peach	Bacterial Spot	456	[11,17]	99.56%
17		Healthy	72	[]	100%
18	Pepper/Bell	Bacterial Spot	196	[2,20,33]	98.49%
19		Healthy	293	[23,23]	99.32%
20	Potato	Early Blight	204	[22]	99.51%
21		Healthy	31	[22]	96.87%
22		Late Blight	195	[20,31,31,31,31]	97.50%
23	Raspberry	Healthy	74	[]	100%
24	Soybean	Healthy	499	[5,17,19,35,35]	99.01%
25	Squash	Powdery Mildew	365	[6]	99.73%
26	Strawberry	Healthy	91	[]	100%
27		Leaf Scorch	221	[]	100%
28	Tomato	Bacterial Spot	417	[15,33,33,35,37,37,37]	98.35%
29		Early Blight	192	[3,3,11,16,16,28,28,31,31,31,31,33,35,37]	93.20%
30		Healthy	319	[6,29]	99.38%
31		Late Blight	361	[5,6,6,8,8,8,8,8,8,8,10,10,22,28,29,29,29,35,36]	94.75%
32		Leaf Mold	186	[3,31,34,34]	97.89%
33		Septoria Leaf Spot	347	[31,31,31,32,32,35]	98.30%
34		Spider Mites/Two-Spotted Spider Mite	330	[31,31,35,35,35]	98.51%
35		Target Spot	279	[34]	99.64%
36		Tomato Mosaic Virus	75	[]	100%
37		Tomato Yellow Leaf Curl Virus	490	[15,28,28,32,35]	98.99%

Table 5.5: ChannelSplitNet with CF Output

Index	Plant Name	Condition	Correct Predictions	Misclassified Index List	Accuracy
0	Apple	Apple Scab	124	[1,2,3,3]	96.88%
1		Black Rot	122	[20,33]	98.39%
2		Cedar Apple Rust	59	[]	100%
3		Healthy	321	[0,1,1,17,17,31,33]	97.87%
4	Blueberry	Healthy	300	[33]	97.87%
5	Cherry (Including Sour)	Healthy	170	[]	100%
6		Powdery Mildew	207	[0,3,10]	98.57%
7	Corn(Maize)	Cercospora/Gray Leaf Spot	89	[10,10,10,10,10,10,10,10 ,10,10,10,10,10,10]	86.41%
8		Common Rust	241	[]	100%
9		Healthy	232	[]	100%
10		Northern Leaf Blight	186	[7,7,7,7,7,7,7,7,7,8]	94.42%
11	Grap	Black Rot	222	[12,12,12,12,12,12,12,12 ,12,12,12,12,12]	94.47%
12		Esca (Black Measles)	271	[11,11,11,11,11]	98.19%
13		Healthy	84	[]	100%
14		Leaf Blight (Isariopsis Leaf Spot)	215	[]	100%
15	Orange	Huanglongbing (Citrus Greening)	501	[4]	99.80%
16	Peach	Bacterial Spot	454	[17,17,18,33]	99.13%
17		Healthy	71	[16]	98.61%
18	Pepper/Bell	Bacterial Spot	191	[19,19,19,19,19,19,19,29]	95.98%
19		Healthy	288	[17,18,18,18,18,18,18]	97.63%
20	Potato	Early Blight	203	[16,31]	99.02%
21		Healthy	29	[22,22,22]	90.63%
22		Late Blight	191	[20,20,20,20,21,31,31,31,31]	95.50%
23	Raspberry	Healthy	74	[]	100%
24	Soybean	Healthy	498	[4,17,22,29,34,35]	98.81%
25	Squash	Powdery Mildew	364	[13,31]	99.45%
26	Strawberry	Healthy	91	[]	100%
27		Leaf Scorch	221	[]	100%
28	Tomato	Bacterial Spot	412	[29,29,29,31,31,31,31,33,33, 37,37,37]	97.17%
29		Early Blight	163	[22, 22, 22, 28, 28, 28, 28, 28, 28, 28, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 33, 33, 33, 33, 33, 34, 34, 35, 35, 35, 35, 35, 35, 35, 35, 36, 36, 37]	79.13%
30		Healthy	318	[29,35,35]	99.07%
31		Late Blight	355	[3, 13, 17, 22, 22, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 30, 30, 32, 32, 32, 33, 35, 35, 35, 37]	93.18%
32		Leaf Mold	181	[4,29,31,31,33,34,34,34,36]	95.26%
33		Septoria Leaf Spot	331	[19, 28, 28, 28, 28, 28, 28, 31, 31, 31, 31, 32, 32, 32, 32, 32, 34, 34, 35, 35, 36, 36]	93.77%
34		Spider Mites/Two-Spotted Spider Mite	325	[35,35,35,35,35,35,35,35, 35]	97.01%
35		Target Spot	242	[28, 29, 29, 30, 30, 30, 30, 30, 30, 31, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34]	86.43%
36		Tomato Mosaic Virus	75	[]	100%
37		Tomato Yellow Leaf Curl Virus	489	[28,28,33,34,34,35]	98.79%

Table 5.6: Pretrained-VGG16 Result

Index	Plant Name	Condition	Correct Predictions	Misclassified Index List	Accuracy
0	Apple	Apple Scab	128	[]	100%
1		Black Rot	124	[]	100%
2		Cedar Apple Rust	58	[3]	98.31%
3		Healthy	322	[0,0,0,1,1,1]	98.17%
4	Blueberry	Healthy	300	[5]	99.67%
5	Cherry (Including Sour)	Healthy	169	[24]	99.41%
6		Powdery Mildew	210	[]	100%
7	Corn(Maize)	Cercospora/Gray Leaf Spot	89	[10,10,10,10,10,10,10,10,10,10,10,10,10]	86.41%
8		Common Rust	241	[]	100%
9		Healthy	232	[]	100%
10		Northern Leaf Blight	188	[7,7,7,7,7,7,7,7,7]	95.43%
11	Grap	Black Rot	226	[12,12,12,12,12,12,12,12,12]	96.17%
12		Esca (Black Measles)	275	[11]	99.64%
13		Healthy	84	[]	100%
14		Leaf Blight (Isariopsis Leaf Spot)	215	[]	100%
15	Orange	Huanglongbing (Citrus Greening)	502	[]	100%
16	Peach	Bacterial Spot	456	[17,28]	99.56%
17		Healthy	71	[16]	98.61%
18	Pepper/Bell	Bacterial Spot	197	[6,33]	98.99%
19		Healthy	291	[18,18,24,29]	98.64%
20	Potato	Early Blight	204	[33]	99.51%
21		Healthy	30	[19,22]	93.75%
22		Late Blight	195	[21,31,31,31,34]	97.50%
23	Raspberry	Healthy	73	[3]	98.65%
24	Soybean	Healthy	501	[17,33,35]	99.40%
25	Squash	Powdery Mildew	366	[]	100%
26	Strawberry	Healthy	91	[]	100%
27		Leaf Scorch	219	[31,33]	99.10%
28	Tomato	Bacterial Spot	419	[29,31,37,37,37]	98.82%
29		Early Blight	178	[16, 20, 28, 28, 28, 28, 31, 31, 31, 31, 31, 31, 31, 31, 31, 33, 33, 34, 35, 35, 35, 35, 35, 35, 36, 37, 37]	86.41%
30		Healthy	315	[29,31,33,34,35,35]	98.13%
31		Late Blight	357	[16, 22, 22, 22, 22, 26, 29, 29, 29, 29, 29, 29, 29, 29, 29, 32, 33, 33, 33, 33, 34, 35, 35]	93.70%
32		Leaf Mold	181	[24,29,29,34,34,35,35,36,36]	95.26%
33		Septoria Leaf Spot	332	[14, 27, 28, 28, 28, 28, 29, 29, 29, 29, 29, 30, 31, 31, 31, 32, 35, 35, 35, 36, 37]	94.50%
34		Spider Mites/Two-Spotted Spider Mite	324	[29,32,35,35,35,35,35,35,35,35]	96.72%
35		Target Spot	261	[29,30,30,32,33,33,33,33,33,33,34,34,34,34,34,34,34,34,34]	93.21%
36		Tomato Mosaic Virus	74	[34]	98.67%
37		Tomato Yellow Leaf Curl Virus	491	[15,31,32,34]	99.19%

Table 5.7: Pretrained-ResNet50 Result

Network	Weight	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	Test Accuracy	Precision	Recall	F1-Score	Model Size	Training Time Per Epoch	Prediction Time	Trainable Parameters
AlexNet	TfS	98.99%	0.0309	97.90%	0.0725	98.75%	0.9845	0.9869	0.9856	223MB	294s	7.56s	58,439,782
GoogLeNet	TfS	99.63%	0.0120	98.09%	0.0694	98.40%	0.9807	0.9833	0.9819	41.9MB	743s	26.67	10,925,474
ChannelSplitNet – Dense Output	TfS	91.54%	0.2683	94.22%	0.1905	96.04%	0.9540	0.9603	0.9568	87.4MB	1832s	47.87s	22,882,790
ChannelSplitNet – CF Output	TfS	99.45%	0.0189	97.56%	1.2989	98.43%	0.9826	0.9844	0.9833	39.1MB	1580s	46.14s	10,221,542
VGG16	TfS	98.45%	0.0450	98.29%	0.0570	98.99%	0,9883	0.9893	0.9887	512MB	4535s	96.33s	134,433,126
	Pretrained	99.74%	0.0087	93.92%	0.2469	97.04%	0.9674	0.9682	0.9676	512MB	965s	74.33s	119,701,542
ResNet50	Pretrained	99.70%	0.0174	97.08%	0.0903	97.90%	0.9772	0.9766	0.9768	102MB	666s	57.22s	3,186,726
Inception V3	Pretrained	61.84%	1.2975	50.53%	1.8000	63.18%	0.5823	0.6095	0.5870	287MB	550s	44.30s	53,518,374
Ensemble						99.39%	0.9922	0.9937	0.9929	918MB		213s	

Table 5.8: Comparison Table of Models

Number of Convolutional Layers	Weights	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	Test Accuracy	Model Size	Prediction Time	Trainable Parameters
No Convolutional Layer	TfS	43.65%	1.9160	40.50%	2.1195	47.24%	149MB	2.78s	6,021,158
One Convolutional Layer	TfS	74.27%	0.8500	56.75%	1.5955	69.69%	43.7MB	4.58s	3,815,910
Two Convolutional Layers	TfS	96.71%	0.1002	84.37%	0.6845	87.98%	5.75MB	4.36s	497,830
Four Convolutional Layers	TfS	97.28%	0.0796	93.52%	0.2259	96.28%	5.01MB	5.68s	429,862

Table 5.9: Number of Convolutional Layers Experiment Result

5.1.1 Discussion

5.1.1.1 Classic CNN Models

It can be seen from the results that most of the CNN models perform well on the plant diseases dataset. VGG16 achieved the highest test accuracy and F1-score of all models. But its cost is also huge. This model occupies 512 Megabytes of storage space. The training time of each epochs is several times that of other models, and the prediction time is also the longest. One of the reasons is that it has a huge number of parameters to be trained.

Although the AlexNet model size is not small, it has the fastest training and prediction time. The test accuracy and F1-score are closely behind VGG16. GoogLeNet benefits from the inception module, which greatly reduces the number of parameters and is the smallest one in the classic models. Its test accuracy and F1-score are in the middle of these models.

The performance of the pre-trained Inception V3 model is the worst, with an accuracy of only 63.18% on the test set. The parameters in convolution part are pretrained and frozen, and only the added fully connected layer is trained. Even the different layers or parameters of the fully connected layer are tried, the performance is still not greatly improved. Therefore, the most likely problem is the convolution part. By looking at the structure of Inception V3 (**Figure 5.10**), the original input shape is $299 \times 299 \times 3$, and the input size of this model is $224 \times 224 \times 3$. When $224 \times 224 \times 3$ is input to the first convolutional layer with 3×3 kernel size and 2 strides. By calculation (Formula in **Section 2.1.1**):

$$Output\ size: \frac{224 - 3}{2} + 1 = 111.5$$

It can be known that the output size is not an integer, and this also happens latter in Inception V3 Architecture. In this case, Keras would not extract features from the edges, and output size becomes 111×111. Therefore, the efficiency of extracting features of the convolution part is poor. As a result, the performance of the model is not satisfactory.

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 5.10: Inception V3 Architecture (Szegedy et al., 2016)

5.1.1.2 Trained from Scratch vs. Pretrained

After comparison, all pre-trained models do not perform as well as models trained from scratch. This is more intuitive on VGG16, with a difference of 1.95% in accuracy on the test set and a difference in F10-score of 0.0211. Because the pretrained convolution part is to extract features from a wide range of inputs, the convolution part that trained from scratch can specifically extract features that are more effective for this model.

The advantage of pre-training is that it can effectively reduce the training time, as VGG16, the time per epoch is reduced by 78.72%.

5.1.1.3 Number of Convolutional Layers

The purpose of convolution is to extract features. If there is no convolutional layer, that is, a

fully connected network, each pixel is trained as a feature. From the results, it can be seen that this performance is inferior and requires more storage.

With the convolutional layers increase, accuracy becomes better and when it comes to four conv layers, the performance is good on this dataset. Also model size decreases about 96% compared to no conv layer. However, the prediction time is proportional to the number of convolutional layers. This proves CNN is generally better than fully connected network in vision tasks.

5.1.1.4 ChannelSplitNet vs. Classic CNN Models

The accuracy of the ChannelSplitNet with dense output by is the lowest except for the Inception V3 model, and the training and prediction time is longer than most of the models. The only advantage is that it occupies the third smallest space among all models.

When ChannelSplitNet output replaced Dense with Conv-Flatten structure, the test accuracy increased by 2.39%, becoming the third of all models, precision increased by 0.0286, recall increased by 0.0241, and F1-score increased by 0.0255. Meanwhile, because the dense layers are removed, the number of parameters is greatly reduced, the model size becomes the smallest, the training time is slightly reduced, and the prediction time is basically the same.

This proves that the Conv-Flatten output structure works, and it can be inferred why the Dense output is inferior to the CF output for ChannelSplitNet. First of all, during training, the use of dense layers makes the network trapped in the local optimal and cannot escape, resulting in lower accuracy than other networks. It may also be that after the convolution part of the ChannelSplitNet, the extracted features are nonlinear, so too many dense layers make the classification effect poor.

From the table, it can be seen that most models confuse *Cercospora/Gray Leaf Spot* and *Northern Leaf Blight* of Corn (maize), *Early Blight* and *Late Blight* of Tomato *and Spider mites/Two Spotted Spider Mite* and *Target Spot* of Tomato. The sample images of these categories are illustrated in **Figure 5.11**. The images of these categories are similar, which leads to misclassification.

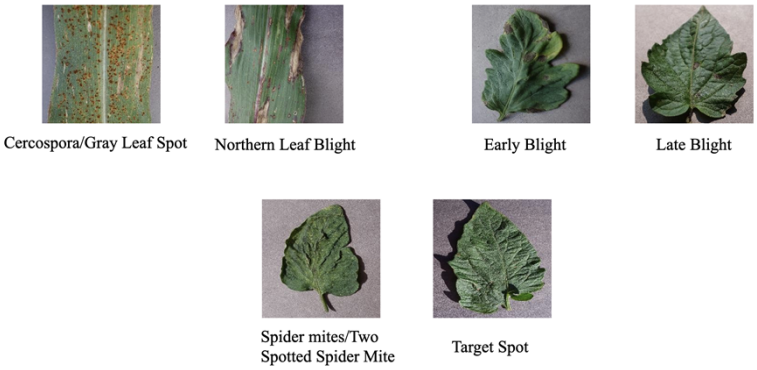


Figure 5.11: Sample Images of Misclassified Categories

5.1.1.6 Ensemble Model

The ensemble model is composed of four base models: GoogLeNet, ChannelSplitNet with CF output, VGG16 training from scratch, and ResNet50. Although AlexNet performs well, its input is $227 \times 227 \times 3$, while the input size of other models is $224 \times 224 \times 3$, which cannot be uniformly predicted, so AlexNet is abandoned.

As expected, the ensemble model achieved the best performance. In the test accuracy, precision, recall and F1-score all reached the first. However, since it needs the prediction results of each model, the prediction time and model size are the integration of all the base models, which makes the ensemble model occupy a huge space and the longest prediction time.

If the performance in other models can meet the demand and the extreme performance is not required, the ensemble model is not recommended. Because it costs a lot. However, if the performance of the models is poor, then the ensemble model would be a very convenient choice, which existing model can be used without retraining a new model.

5.2 Extra Experiments

Two extra experiments are designed to further explore the Conv-Flatten output structure (as described in **Section 4.6**). The comparison table made by result of extra experiment one and dense output is illustrated in **Figure 5.12** and extra experiment two is illustrated in **Figure 5.13**.

Network	Weights	Output Type	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	Test Accuracy	Precision	Recall	F1 – Score	Model Size	Training Time Per Epoch	Prediction Time	Trainable Parameters
AlexNet	TfS	Conv-Flatten	99.43%	0.0278	96.16%	0.1139	98.18%	0.9748	0.9769	0.9757	15.7MB	294s	7.27s	58,439,782
		Dense	98.99%	0.0309	97.90%	0.0725	98.75%	0.9845	0.9869	0.9856	223MB	220s	7.01s	4,100,198
VGG16	Pretrained	Conv-Flatten	100%	1.855e-05	91.44%	0.9087	95.28%	0.9505	0.9500	0.9500	59.8MB	842s	67.27s	953,382
		Dense	99.74%	0.0087	93.92%	0.2469	97.04%	0.9674	0.9682	0.9676	512MB	965s	73.48s	119,701,542
ResNet50	Pretrained	Conv-Flatten	95.65%	0.1812	94.20%	0.2171	95.13%	0.9508	0.9498	0.9500	90.5MB	668s	55.93s	77,862
		Dense	99.70%	0.0174	97.08%	0.0903	97.90%	0.9772	0.9766	0.9768	102MB	666s	57.35s	3,186,726
Inception V3	Pretrained	Conv-Flatten	74.56%	2.7659	56.41%	7.4987	69.44%	0.6493	0.6713	0.6543	91MB	504s	41.94s	1,945,638
		Dense	61.84%	1.2975	50.53%	1.8000	63.18%	0.5823	0.6095	0.5870	287MB	550s	44.04s	53,518,374

Figure 5.12: Comparison Table of CF Output and Dense Output

Network	Weights	Output Type	(Correspond to) Number of dense layers	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	Test Accuracy	Model Size	Training Time Per Epoch	Prediction time	Trainable parameters
VGG 16	Pretrained	CF 7x7	1	100%	1.1855e-05	91.44%	0.9087	95.28%	59.8MB	842s	67.27	953,382
		CF 7x1, 1x7		97.71%	0.0899	86.51%	0.7227	92.00%	56.7MB	806s	67.29s	146,376
		Dense		100%	2.5404e-04	91.07%	1.1069	94.88%	59.8MB	805s	66.97s	953,382
		CF 7x7,1x1	2	100%	3.1585e-04	92.87%	0.3564	96.12%	154MB	1022s	75.65s	25,730,086
		CF 7x1, 1x7, 1x1		100%	2.4955e-04	93.40%	0.4198	96.44%	98.3MB	904s	69.17s	11,051,046
		Dense		100%	1.9711e-04	92.31%	0.4174	96.07%	154MB	893s	67.06s	25,730,086

Figure 5.13: Result of Extra Experiment Two

5.2.1 Discussion

5.2.1.1 Extra Experiment One

It can be seen from the results that the structure of Conv-Flatten output can be used not only in ChannelSplitNet, but also in some common CNN models. Compared with the accuracy of Dense output, the performance of CF output has decreased by 0.57% to 2.77%, but the performance is still acceptable. The main thing is that the dense layers are removed, which significantly reduces the model parameters and model size. The most obvious one is AlexNet, the model size has dropped by 88.47%.

For models that do not perform well in Dense output, such as InceptionV3, using CF output cannot greatly improve the accuracy. Therefore, the CF output structure is suitable for models that perform well on Dense output and not suitable for greatly improving accuracy.

5.2.1.2 Extra Experiment Two

When using one or two Dense output to be replaced by 7×7 Conv-Flatten output, the parameters and model size are exactly the same, and the performance of the model is also similar. However, when 7×1 and 1×7 are used to replace the 7×7 convolutional layer, the accuracy drops slightly or remains the same, but the model size drops by at most 36.16%, and other indicators are similar.

Therefore, it can be concluded that when a model performs well on one or two dense output, the CF output can be used instead, and the performance will not fluctuate greatly. If the model size requirements are very strict, two $N \times 1$ and $1 \times N$ convolutional layers can be used instead of the $N \times N$ convolutional layer. In this case, the accuracy may be slightly reduced, but a smaller model is obtained.

Chapter 6

6. Discussion

6.1 Summary of Achievement

Throughout the project, the author has gained experience in data processing, construction of deep learning image classification architectures, GPU-based model training, results/model analysis and Python skills.

During the project, some classic convolutional neural network architectures were trained from scratch or used pre-trained models to check their performance in the classification of plant diseases. The author designed a new CNN architecture combining the advantages of multiple classic architectures and compared this with the classic models. A new structure that replaced the Dense layer was tested. The training time for all experiments ranged from 10 minutes to 38 hours.

According to the table generated by the confusion matrix, the classes that are easily confused by the model are analyzed. The models are compared and evaluated comprehensively by combining multiple indicators. Use the idea of controlling variables to explore the Conv-Flatten output structure.

6.2 Limitations, Improvement and Future Work

Due to the limitation of GPU performance, this experiment uses a dataset of 113,701 images with 38 categories. However, many real-world problems are much larger than this. And some experiments have to be stopped earlier because of the long training time, so they may not be the best model.

In future work, more datasets are needed to evaluate the performance of ChannelSplitNet. At the same time, the evaluation work can start from the feature map and model weights, and have a deep understanding of the feature extraction of the model, so as to make reasonable improvements to the network. In addition, more experiments can be designed to compare dense output, $N \times N$ Conv-Flatten output and $N \times 1$, $1 \times N$ Conv-Flatten output in order to have more accurate conclusions.

Chapter 7

7. Conclusion

In this project, the author first trained a number of classic CNN models and tested their performance in plant disease classification. The best model was VGG16 trained from scratch, which achieved 98.99% accuracy. Subsequently, combining the advantages of these models, a new CNN architecture was designed, and after training, it finally achieved an accuracy of 96.04%. In order to improve this model, the author started from the perspective of reducing the parameters, removed the dense layer and replaced it with the Conv-Flatten structure. The final accuracy was increased to 98.43%, and the model size was only 39.1 Megabytes, which was the smallest of all models. Finally, in order to further increase the accuracy, an ensemble model was established, and achieved an astonishing 99.39% accuracy.

In order to further explore the Conv-Flatten structure, the author put the Conv-Flatten structure in AlexNet, VGG16, ResNet50 and Inception V3. The results show that this structure is similar to the performance of the dense layer, and when the dense layer does not perform well, the Conv-Flatten layer cannot help a lot. After that, the author further improved the $N \times N$ Conv-Flatten structure and disassembled it into two $N \times 1$ and $1 \times N$ convolutional layers. Using pre-trained VGG16 combined with these structures, it was found that $N \times 1$ and $1 \times N$ convolutional layers compared with the same dense layer or $N \times N$ layer, the accuracy is about similar or slightly reduced, but the size is reduced. Therefore, when the accuracy of one or two Dense layers is acceptable, $N \times 1$ and $1 \times N$ structures can be used to replace the dense layer in order to further reduce the model size.

Chapter 8

8. Reference

Abiyev, R.H., and Ma'aitah, M.K.S. (2018), 'Deep convolutional neural networks for chest diseases detection', *Journal of Healthcare Engineering*, pp. 1-11.

Ahmed, N., Asif, H.M.S., and Saleem, G. (2021) 'Leaf image based plant disease identification using color and texture features', arXiv:2102.04515. Available online at: https://arxiv.org/abs/2102.04515 [Accessed 23 Apr. 2021]

Albawi, S., Mohammed, T.A., and Al-Zawi, S. (2017) 'Understanding of a convolutional neural network', *2017 International Conference on Engineering and Technology (ICET)*, pp. 1-6

Choi, D., Shallue, C.J., Nado, Z., Lee, J., Maddison, C.J., and Dahl, G.E. (2020) 'On empirical comparisons of optimizers for deep learning', arXiv:1910.05446. Available online at: https://arxiv.org/abs/1910.05446 [Accessed 23 Apr. 2021]

cs231n.github.io, (n.d.) 'Convolutional neural networks for visual recognition', Available online at: https://cs231n.github.io/convolutional-networks/ [Accessed 23 Apr. 2021]

Huilgol, P. (2020) 'Precision vs. Recall – An intuitive guide for every machine learning person', Available online at: https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/ [Accessed 23 Apr. 2021]

Ioffe, S., and Szegedy, C. (2015), 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', arXiv:1502.03167. Available online at: https://arxiv.org/abs/1502.03167 [Accessed 23 Apr. 2021]

Isleib, J. (2012, December 19), *Signs and symptoms of plant disease: Is it fungal, viral or bacterial?* Available online at: https://www.canr.msu.edu/news/signs_and_symptoms_of_plant_disease_is_it_fungal_viral_or_bacterial [Accessed 23 Apr. 2021]

Kabir, M.M., Ohi, A.Q., and Mridha, M.F. (2020). 'A multi-plant disease diagnosis method using convolutional neural network', arXiv:2011.05151. Available online at: https://arxiv.org/abs/2011.05151 [Accessed 23 Apr. 2021]

Khvostikov, A., Aderghal, K., Benois-Pineau, J., Krylov, Andrey., and Catheline, G. (2018).

‘3D CNN-based classification using sMRI and MD-DTI images for Alzheimer disease studies’, arXiv:1801.05968v1*[Preprint]*. Available online at: http://arxiv.org/abs/1801.05968 [Accessed 23 Apr. 2021]

Kotu, V., and Deshpande, B. (2018) *Data science – concepts and practice*. 2nd edn. Jonathan Simpson

Krizhevsky, A., Sutskever, I., and Hinton, G.E. (2012) ‘ImageNet classification with deep convolutional neural netowkrs’, *In Proceeding of NIPS*, pp. 1106-1114

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998) ‘Gradient-based learning applied to document recognition’, *Proceedings of the IEEE*, 86(11), pp. 2278-2324

McCulloch, W.S., and Pitts, W.H. (1943). ‘A logical calculus of the ideas immanent in nervous activity’, *Bulletin of Mathematical Biophysics*, 5, pp. 115-133.

O’Shea, K., and Nash, R. (2015). ‘An introduction to convolutional neural networks’, arXiv:1511.08458. Available online at: https://arxiv.org/abs/1511.08458 [Accessed 23 Apr. 2021]

Saleem, M.H., Potgieter, J., and Arif, K.M. (2020) ‘Plant disease classification: A comparative evaluation of convolutional neural networks and deep learning optimizers’, *Plants*, 9(10), pp. 1319

Saleem, M.H., Potgieter, J., and Arif, K.M. (2019) ‘Plant disease detection and classification by deep learning’, *Plants*, 8(11), pp. 468

Sharma, S., Sharma, S., and Athaiya, A. (2020). ‘Activation functions in neural networks’, *‘International Journal of Engineering Applied Sciences and Technology’*, 4(12), pp. 310-316.

Simonyan, K., and Zisserman, A. (2015) ‘Very deep convolutional networks for large-scale image recognition’, arXiv:1409.1556. Available online at: https://arxiv.org/abs/1409.1556 [Accessed 23 Apr. 2021]

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). ‘Dropout: A simple way to prevent neural networks from overfitting’, *Journal of Machine Learning Research*, 15(56), pp. 1929-1958.

Srivastava, N. (2013) ‘Improving neural networks with dropout’, *Msters’s thesis, Unviersity of Toronto.*

Szegedy, C., Liu, W., Jia, Y., Sermanet, Pierre., Reed, Scott., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015) ‘Going deeper with convolutions’, arXiv:1409.4842. Available online at: https://arxiv.org/abs/1409.4842 [Accessed 23 Apr. 2021]

Szegedy, C., Vanhoucke, V., Ioffe, S., Shelens, J., and Wojna, Z. (2015) ‘Rethinking the

Inception architecture for computer vision', arXiv:1512.00567. Available online at: https://arxiv.org/abs/1512.00567 [Accessed 23 Apr. 2021]

Yan, Q., Yang, B., Wang. W., Wang, B., Chen, Peng., and Zhang, J. (2020). 'Apple leaf diseases recognition based on an improved convolutional neural network', *Sensors (Basel)*, 20(12), PMCID: PMC7349496.

Yilmaz, A., Demircali, A.A., Kocaman, S., and Uvet, H. (2020), 'Comparison of deep learning and traditional machine learning techniques for classification of pap smear images', arXiv:2009.06366. Available online at: https://arxiv.org/abs/2009.06366 [Accessed 23 Apr. 2021]

Appendix

- Version Control:
  https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2020/zxw780

- Plant Diseases Dataset used in this project
  https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2020/zxw780/-/tree/master/Dataset

- Plant Diseases Dataset on Kaggle Page
  https://www.kaggle.com/vipoooool/new-plant-diseases-dataset