PlantDis

A Plant Disease Detector App based on Nested Transfer Learning

Saswat Panda

A report submitted for the course
COMP4560 (Advance Computing Project)
Supervised by: Prof. **Graham Williams**
The Australian National University

May 2022

Except where otherwise indicated, this report is my own original work.

Saswat Panda
27 May 2022

Acknowledgments

I am extremely grateful to my supervisor, Prof. Graham Williams for his invaluable advice, continuous support, and patience during my the project. His immense knowledge and plentiful experience have encouraged me in all the time of the thesis and daily life.

I am also thankful to team mentor Dr. Susan Elias and all my team members of the Smart India Hackathon, 2020, through which I was first introduced into the Plant Disease Detection problem.

At last, I would like to thank my Family, for providing continuous encouragement and emotional support through out the thesis and my whole life.

Abstract

The Australian Government loses millions of dollars due to plant diseases which reduces productivity thereby increasing the cost of production. Plant Diseases also lead to conditions such as food insecurity. This thesis is an attempt to minimise the losses caused due to plant diseases by developing disease detection Apps that can help detect diseases before they become incurable, or before they spread too much. In this thesis Nested Transfer Learning approach has been proposed that allows deep learning models to be reused repeatedly for different but closely related datasets. The accuracy achieved using nested transfer learning in detecting 22 classes of diseases is 99.24%. The nested transfer learning performs well on very small size dataset to achieve a 100% accuracy on held out test set. Also, the Mobile and Desktop Apps developed as a part of this thesis, may help the target users to easily access and use complex machine learning models in their daily life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Agricultural extension groups or other institutions, such as local plant clinics, have always supported disease detection. More recently, similar attempts have been aided by making illness diagnosis information available online, taking advantage of the world's growing Internet access [1]. These strategies are inefficient, time-consuming, and difficult. Therefore, quite often, farmers based on their expertise identify most of the plant diseases. Farmers with less expertise may make mistakes during the identification procedure and use medications carelessly. Environmental contamination will result from increased quality and productivity, resulting in unwarranted economic losses [2]. So, thereby if diseases are detected early, efficiently and accurately this will also promote organic farming.

The cultivation of fruits and vegetables contributes to around 13% of the overall agriculture. So, by gross value they contribute around $9.5 billion to the Australian economy [3]. Australian government loses millions of dollars due to plant diseases every year. In this thesis an attempt to develop products, that may be helpful to minimise these loses, has been made. Therefore, this thesis largely focuses on detecting the disease of the plants which are produced the most in Australia.

Infected plants frequently have visible lesions or markings on their leaves, stems, blooms, or fruits. Each illness or pest situation has its own distinct apparent pattern that may be utilised to identify anomalies. Plant leaves are usually the major source for diagnosing plant illnesses, and most disease signs can start to show on the leaves [2]. Therefore, in this thesis images of plant leaves are used to diagnose the plant diseases.

Transfer learning makes it easier to learn a new activity by reusing information from previous related ones. The purpose of transfer learning is to use existing knowledge and experience to acquire fresh but related concepts more quickly than would be feasible without it [4]. In this thesis, nested transfer learning has been proposed to learn even faster, and with better accuracy and robustness to very small datasets. The nested transfer learning approach has been used for detecting plant diseases. The approach was specifically used for increasing the number of classes of the diseases, and also for detecting diseases of a different plant with a very small amount of data.

Because of widespread smartphone use, HD cameras, and high-performance CPUs in mobile devices, illness diagnosis of plants based on automatic picture recog-

nition can be made available at an unprecedented scale [1]. Therefore, in order to testify this a mobile app was developed as a part of this thesis. This also helps in showing the proof of usability of the models developed by allowing their real-life testing. Also, in this thesis a desktop application has been developed which can be used in government offices for detecting plant diseases, and also it can be installed in public computers for general public to use it for detecting plant disease.

In this thesis there are a total of 4 chapters namely Background 2, Datasets and Machine Learning Model 3, User Applications 4, Results 5 and finally Conclusion 6.

Chapter 2

Background and Related Work

In this chapter we will discuss about the methodologies used and previous work done on them.

Section 2.1 provides background information needed to read this report, whereas Section 2.2 provides information on related work.

2.1 Background

This paper has been put in such a way that it should be easily understandable to any one with a computing background. A basic understanding of deep learning can help the readers easily understand the paper.

2.2 Related work

For leaf disease detection, a 9-layer deep convolutional neural network was proposed by the paper by Geetharamani et al., 2019 [5], which achieved a classification accuracy of 96.46% for 38 classes. In a paper by Mohanty et al, 2016 [1], the deep learning model trained by them achieved an accuracy of 99.35% percent on a held-out test set. In a research by Saleem et al., 2019 [6] numerous state of the art deep learning models have been compared and best model was stated for different plant disease detection problems.

In a research in 2020 by Chen et al [7], A VGGNet that has already been pre-trained on ImageNet and the Inception module, were used to predict rice plant images with an accuracy of 92%. They also achieved a validation accuracy of 91.83% on a public dataset. In a paper by arshad et al, 2021 [8], different transfer learning approaches were compared to find ResNet50 achieving highest accuracy of 98.7% for prediction of 16 classes.

In previous work, various models were proposed and also different models were compared to obtain a best model for plant disease detection. But, close nothing has been discussed on how to reuse the knowledge gained by the trained models to learn from closely related data quickly, which will enable easy expansion of number of classes. Past knowledge about closely related data might help the quickly learn from

very small datasets. This thesis focuses on these aspects.

2.3 Summary

So, in this chapter we discussed about the background needed and related works that has been done on plant disease detection, and how this paper is unique than the previous papers. In the next chapter we will see which datasets has been used and learn about Machine Learning Models used in this chapter.

Chapter 3

Datasets and Machine Learning Model

In this chapter we will see the datasets used, model architecture, and learn about Transfer Learning and the proposed Nested Transfer Learning.

3.1 Dataset

3.1.1 Dataset Augmentation

All of the datasets used in this thesis were either already augmented or have been augmented using techniques such as Gamma correction, PCA(Principal Component Analysis) color augmentation, noise injection, image flipping, Scaling, and rotation [9], so that the dataset size can be increased and the model developed will be more robust.

3.1.2 Dataset 1

The first dataset, which will be referred as Dataset 1 throughout the thesis, is a subset of a dataset [9], which itself is based on Plant Village dataset [10]. From whole dataset images relating to Apple, Corn, Potato, and Tomato were selected. The reason being these are some of the largest grown plants in Australia. In Australia, for the year 2019-20, Apple was the 3rd largest produced fruit, Potato and Tomato were the 1st and 2nd largest produced vegetables respectively. The Table 3.1 below shows the disease of plant leaf, and also how they were split into train and test for using in the model. Apart from this, a total of 33 images was withheld from the model to use for further testing.

Table 3.1: Dataset with 21 classes, train-valid split(80%-20%)

Plant	Disease	Train	Test
Apple	Apple scab	2016	504
	Black rot	1987	497
	Cedar rust	1760	440
	Healthy	2008	502
Corn	Gray Leaf Spot	1642	410
	Common Rust	1907	477
	Healthy	1859	465
	Northern Leaf Blight	1908	477
Potato	Early Blight	1939	485
	Healthy	1824	456
	Late Blight	1939	485
Tomato	Bacterial spot	1702	425
	Early Blight	1920	480
	Healthy	1926	481
	Late Blight	1851	463
	Leaf Mold	1882	470
	Septoria leaf spot	1742	436
	Spider mites	1741	435
	Target spot	1827	457
	Mosaic Virus	1790	448
	Yellow Leaf Curl Virus	1961	490

3.1.3 Other data sets

In this subsection four different but related datasets were created in order to test the performance of Nested Transfer Learning. The Four datasets are Expanded-Imbalanced, Expanded-Balanced, Small and Very-Small Dataset.

The second dataset that was created was a Small dataset, that contain banana leaf. The reason for including Banana leaf images is that it is the 2^{nd} largest produced fruit in Australia. The banana leaf images used are based on data from Owomugisha et al [11]. The Table 3.2 below shows the diseases of banana leaf, and also how they were split into train and test for using in the model. Apart from this, a total of 6 images belonging to the 3 categories was withheld from the model to use for further testing.

The Small dataset was appended to the Dataset 1 to create the Expanded-Imbalanced dataset. The dataset is imbalanced since the total number of images per class of the Small Dataset is not close to the total number of images per class of Dataset 1.

Table 3.2: Small Dataset, train-valid split ratio 80:20

Plant	Disease	Train	Test
Banana	Bacterial Wilt	669	168
	Black Sigatoka	468	115
	Healthy	620	154

The next dataset that was created was an expanded but balanced dataset. This dataset contains all data from Dataset 1, in addition to Orange Citrus greening (Haunglongbing) images. The reason for including Orange leaf images is that it is the largest produced fruit in Australia. The Orange leaf images used are based on Plant Village Dataset [10]. There are 2005 images for training, and 503 images for Validation(Split ratio - 80:20). Apart from this, a total of 5 images was withheld from the model to use for further testing.

The final dataset contains images of grape plant. It is based on data from Plant Village Dataset [9]. Intentionally very less number images were included in this dataset to test the performance of Nested Transfer Learning for very small datasets. The Table 3.3 below shows the diseases of Grape leaf, and also how they were split into train and test for using in the model. Apart from this, 8 images per category was withheld from the model to use for further testing.

Table 3.3: Very Small Dataset, train-valid split ratio 78:22

Plant	Disease	Train	Test
Grape	Black rot	160	45
	Black Measles(Esca)	160	45
	Healthy	160	45
	Leaf Blight	170	45

3.2 Transfer Learning

Deep learning is a machine learning approach that allows computers to learn by example in the same way that people do. Transfer learning is a deep learning strategy that focuses on preserving information learned while addressing one issue and transferring it to a similar but distinct problem. For example, If you trained a basic classifier to predict if a picture contains a dog, you might utilise the model's training knowledge to identify other animals like cats.

There are two ways we can use transfer learning. First one is finetuning, which is the process of initializing the model with the weights of the source model and then finetuning or tweaking it according to for the new dataset. The second one is using

the model as a feature extractor which is the process of using the weights as extracted features, i.e., without changing them while training on the new dataset.

In this thesis Transfer Learning was used in predicting the diseases in Dataset 1(see 3.1.2). Each of the model used had been previously trained on the ImageNet Dataset(14 million images) [12]. The knowledge obtained by the models on ImageNet Dataset has been transferred and used in classifying the plant diseases. The weights(knowledge) obtained were fine-tuned through the training process, to adapt beeter to Dataset 1.

Four models were trained and compared on Dataset 1. The model architecture for Model From Scratch is relatively simple model and is a blend of 2D convolution layers, 2D MaxPooling layers, Dense layers, Flatten Layer, Batch Normalization Layer and Dropout layers. The model architectures for Densenet [13], MobileNet [14], and EfficientNet [15] based models is inspired from my previous research [16]. Details about their Architectures can be found in the appendix(6.1).

3.2.1 Model Comparison

All the models mentioned in Table 3.4 were trained for 4 epochs(except Model From Scratch which was trained for 8 epochs) and then their performance was compared. The maximum train accuracy is 99.05% which is for DenseNet201 and the maximum validation accuracy is 99.11% which is for EfficientNetB2. Also, the difference between train and validation accuracy for the two models is very less indicating a lack of overfitting.

Table 3.4: Model Comparison after 4 epochs.

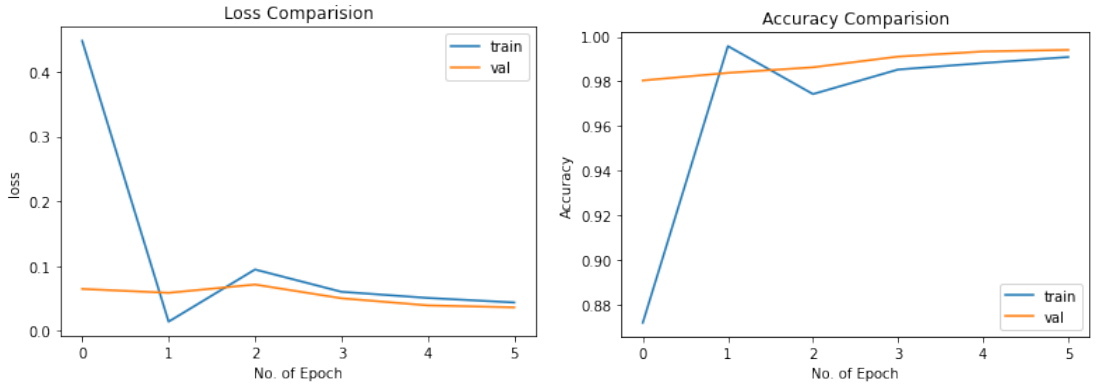
Model	Train Accuracy	Validation Accuracy
EfficientNetB2	0.9853	0.9911
DenseNet201	0.9905	0.9832
MobileNet	0.9811	0.9767
Model From Scratch(8e)	0.9760	0.9321

Therfore, both the models(EfficientNetB2 and DenseNet201) were further trained for two more epochs(Table 3.5) and then compared. As it can be observed that EfficienNettB2 clearly performs better in terms of train and validation accuracy. It can be observed that for EfficientNetB2 the validation accuracy is slightly higher than the train accuracy. The reason for this is that the dropout layers are making it difficult for the model to perform well on the train set by turning off some of the neurons. The model architecture for EfficientNetB2-based model has been discussed in the next section(3.3).

Table 3.5: Model Comparison after 6 epochs.

Model	Train Accuracy	Validation Accuracy
EfficientNetB2	0.9909	0.9941
DenseNet201	0.9869	0.9797

The next important thing to ensure about EfficientNetB2 is that it does not overfits. We can already observe the train and validation accuracy to be very close to each other indicating lack of overfitting. But, we will take the help of graphs(Fig. 3.1) containing model performance throughout six epochs to confirm this. The train and validation loss(Fig. 3.1(a)) gradually decreases becoming very close to each other. Same holds for accuracy(Fig. 3.1(b)) except that it gradually increases. These observations ensure that there is very minimum chance of overfitting. Further, a test on 33 unseen images to the model was performed, which resulted in the model achieving 100% accuracy. The results from the test on images taken by smartphone are included in the Result chapter(Fig. 5.4).



(a) Loss Comparision for 6 epochs

(b) Accuracy Comparision for 36 epochs

Figure 3.1: Loss and Accuracy Graphs for Trnsfer Learning

3.3 Model Architecture

In this paragraph wee will see which model architecture for transfer learning. Model architecture means how different layers in deep learning have been combined together to build a Deep Neural Network Model. In deep learning a layer is the highest level building block. The Fig. 3.2 shows the complete model architecture, starting from input data which are the images of plant diseases, then using the EfficientNetB2 layers(Input and hidden layers) containing the weights obtained after being trained on ImageNet, then using flatten, Batch Normalization, dropout and finally output layer.

EfficientNet is a convolutional neural network design and scaling approach that

uses a compound coefficient to scale all depth/width/resolution dimensions evenly [15]. By doing so, the unimportant training parameters are removed and due to which model size decreases which is beneficial for User Applications(4, which becomes more faster and consume less size on user device. Model Architecture for EfficientNet has been included in the appendix(6.1).

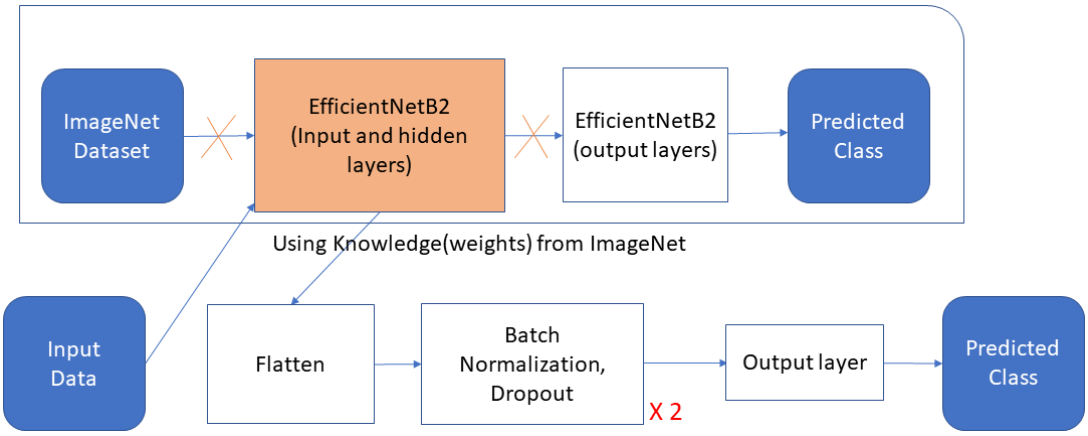


Figure 3.2: The Model Architecture for Transfer Learning

Using the flatten layer, generated 3-Dimensional arrays from pooled feature maps, from EfficientNetB2 are flattened into a single long continuous linear vector. Then the output from the flatten layer is fed into Batch Normalization. The Batch Normalization layer makes the learning easy for the model, thereby making the training process faster. The output is then into the dropout layer. It performs regularization which helps in preventing overfitting. It operates by deactivating neurons and their associated connections at random, which keeps the network from depending too heavily on single neurons and drives all neurons to improve their generalisation abilities.

Now coming to the final layer, the output layer which contains the dense layer. It maps the output from previous layer to 21 features. A dense layer is one that is deeply connected to the layer before it, meaning that the layer's neurons are connected to every neuron in the layer before it. The activation function used in the dense layer is softmax because of its ability to give disease prediction output as a probability vector. Next, we have Adam optimizer, to optimize efficiency, with a learning rate of 0.0001. The learning rate was obtained using a hit and trial method.

3.4 Nested Transfer Learning

Nested Transfer Learning approach is an approach where we can apply transfer learning on the top of results of another transfer learning, which can go on repeatedly. The Fig. 3.3 below shows how nested transfer learning has been used in this thesis for different datasets. First the knowledge(weights) were transferred from Imagenet to train model on dataset 1, then knowlege and weights were futher transferred to train model on Expanded-Balanced dataset, then again model and weights were further

transferred to train model on Small and Very Small Dataset. The knowledge(weights) obtained from dataset 1 were also used to train the model on expanded-imbalanced dataset.

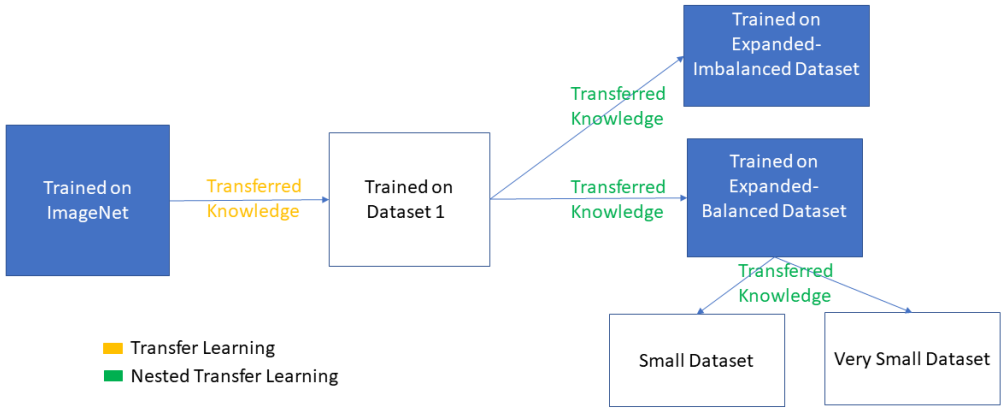


Figure 3.3: Concept Map for Nested Transfer Learning

The model architecture for using nested transfer learning approach is the same as Transfer learning except for the fact that nested transfer learning is being implemented. The architecture(3.4) below is the architecture used for training on Balanced and Imbalanced datasets.

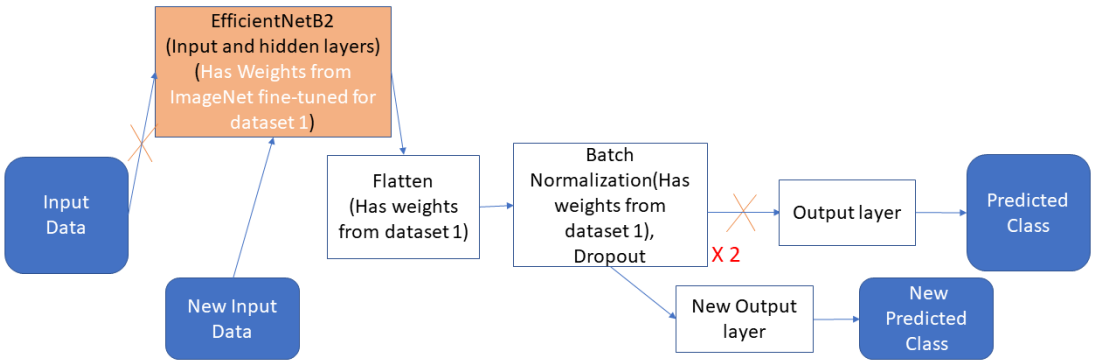


Figure 3.4: Model architecture for Nested Transfer Learning, especially for Balanced and Imbalanced Dataset

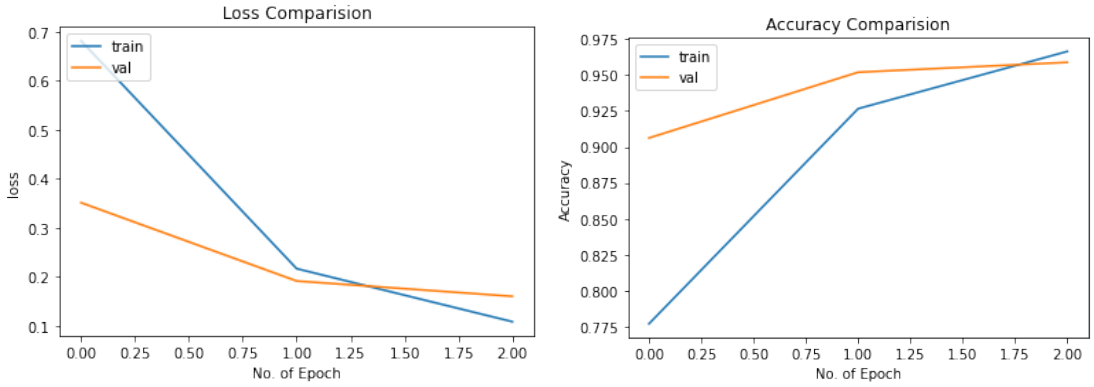
The architecture for very small dataset is also same (3.4) and just differs by weights. The efficientnet layers has weights from Imagenet finetuned on dataset 1 and further finetuned on balanced dataset. The flatten layer has weights from dataset 1 finetuned on balanced dataset. Similarly, batch normalization layers has weights

from dataset 1 finetuned on balanced dataset. The dropout layers donot have any weights to be transferred. While training the model the weights of the efficientnet layers were freezed(used as feature extractor) because of the very small size of the dataset.

Similarly, the architecture for small dataset is also same (3.4) and just differs by weights. Same as very small datasets, the efficientnet layers has weights from Imagenet finetuned on dataset 1 and further finetuned on balanced dataset. But, the flatten layer and batch normalization layers have randomly intialized weights to adapt well to a dataset of different quality(image resolution and background).

3.4.1 Applying on small dataset

Let's have a look at the performance of Nested Transfer Learning on small-sized dataset with 3 classes. We can see(Fig. 3.8) in the graph below that the model achieved a good accuracy(3.8(b)) and low loss(3.8(a)), with only three epochs. The graphs also suggests lack of overfitting, since training and validation lines almost converge into one another. The validation accuracy achieved is 96.64%, and the train accuracy achieved is 95.88%.



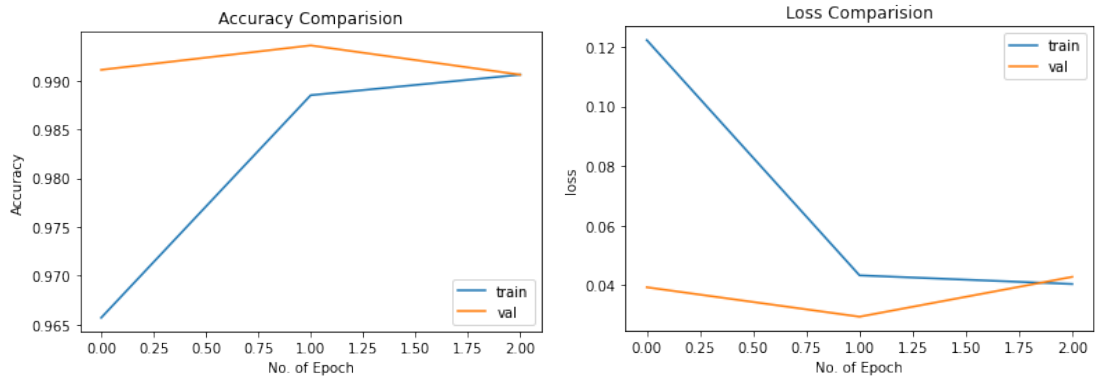
(a) Loss Comparision for 3 epochs

(b) Accuracy Comparision for 3 epochs

Figure 3.5: Loss and Accuracy Graphs for Nested Transfer Learning on on small dataset

3.4.2 Applying on Expanded-imbalanced dataset

Let us examine its performance on expanded but imbalanced dataset. From the graph below we can observe(Fig. 3.6) the model achieved a very good accuracy and low loss, with just 3 epochs. The graphs also suggests lack of overfitting, since training(3.6(b)) and validation(3.6(a)) lines converge into one another. The validation accuracy achieved is 99.06%, and the train accuracy achieved is also 99.06%. The nested transfer learning approach is not suitable for imbalanced dataset check the result section to know why.(Section 5.2.4)

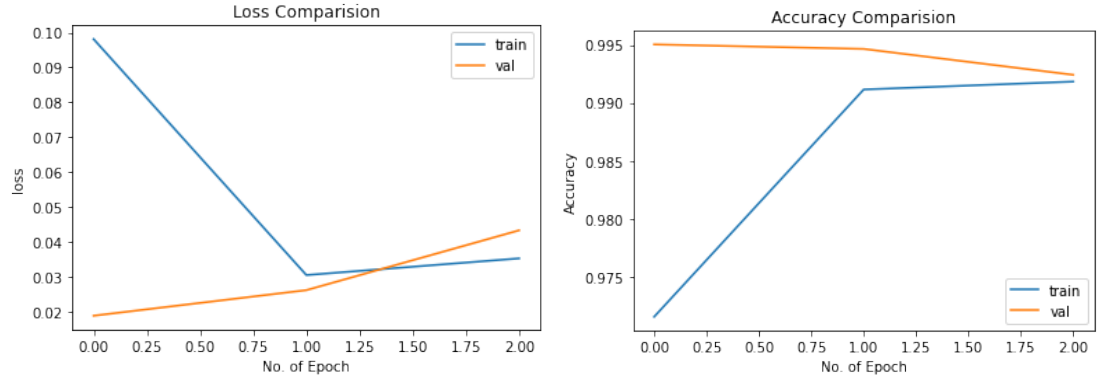


(a) Accuracy Comparision for 3 epochs (b) Loss Comparision for 3 epochs

Figure 3.6: Loss and Accuracy Graphs for Nested Transfer Learning on Imbalanced Dataset

3.4.3 Applying on Expanded-balanced dataset

Let us test its performance on expanded but balanced dataset. As shown in the graph below(Fig. 3.7) the model achieved a very good accuracy and low loss, with just 3 epochs. The graphs also suggests lack of overfitting, since training(3.7(b)) lines almost converge into each another. The validation loss(3.7(a)) slightly increases(0.005) from epoch 1 to epoch 3, but finally the difference between train and validation loss is very less(0.008). The validation accuracy is 99.24%, and the train accuracy achieved is 99.18%, being very close to each other. The validation accuracy is greater than train accuracy because of multiple dropout layers.



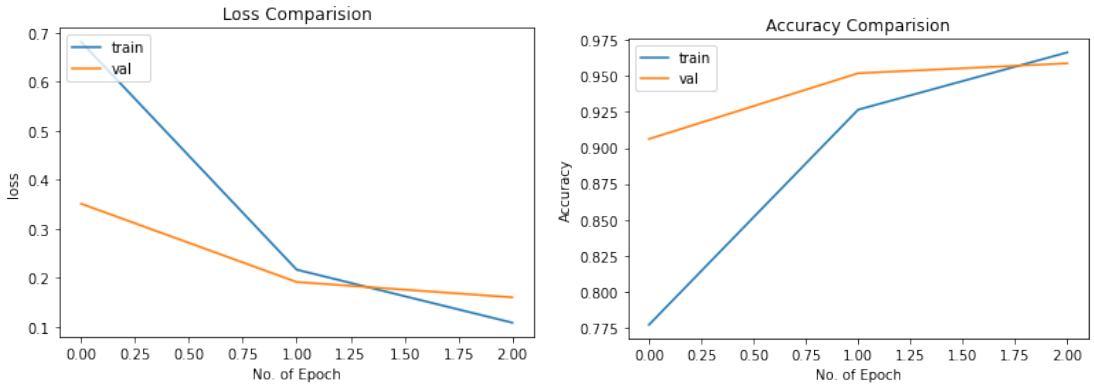
(a) Loss Comparision for 3 epochs (b) Accuracy Comparision for 3 epochs

Figure 3.7: Loss and Accuracy Graphs for Nested Transfer Learning on Imbalanced Dataset

3.4.4 Applying on very small dataset

Let's have a look at the performance of Nested Transfer Learning on a very small-sized dataset with 3 classes. We can see(Fig. 3.8) in the graph below that the model achieved a good accuracy(3.8(b)) and low loss(3.8(a)), with only three epochs. The

graphs also suggests lack of overfitting, since training and validation The lines are almost touching each other. The validation accuracy achieved is 98.15%, and the train accuracy achieved is 97.22%.



(a) Loss Comparision for 3 epochs

(b) Accuracy Comparision for 3 epochs

Figure 3.8: Loss and Accuracy Graphs for Nested Transfer Learning on on small dataset

3.5 Summary

In this chapter we discussed several important topics starting from Datasets, to Transfer Learning and its results, and finally discussing about the proposed nested transfer learning and its performance on the various datasets. In next chapter we will see about the applications developed to make the machine learning models easily accessible and usable.

Chapter 4

User Applications

This chapters covers about the user applications developed as a part of this thesis. The motivation behind developing user applications is to support easily accessible open source tools for complex Machine Learning Models. The working of these applications in real life acts as a proof of usability in real-world scenario. Flutter has been used for application development to enable cross-platform support. Currently Flutter Supports six different popularly used platforms. The link to the code can be found in the appendix(6.1).

4.1 Mobile Application

The frontend for the model application was developed using flutter, which was then connected to the Machine Learning Model using the flutter tfilte package(Fig. 4.1). The mobile app is currently tested only on android platforms. The roles of frontend is to help user capture/collect image and then provide it to the tflite package which then does some required image preprocessing and passes the preprocessed image to the model. The ML model then uses the image and calculates probability vector and sends them back to the tflite package which then filters the class with highest probability and sends it back to the frontend. The frontend then displays the predicted class(disease) to the user. Another important package that the front-end depends upon is the Image Picker package to help it capture/ collect image.

Figure 4.1: Mobile App Pipeline

The demo can be seen in Fig. 4.2 below. The user can either capture the image from camera or can choose it from gallery. After capturing the image through camera, the user has to press the OK button, and then the central part of the capture image

will be shown on the screen. Then, clicking on the Diagnose Button the user gets the result of the disease prediction which is Tomato Late Blight for the demo shown.

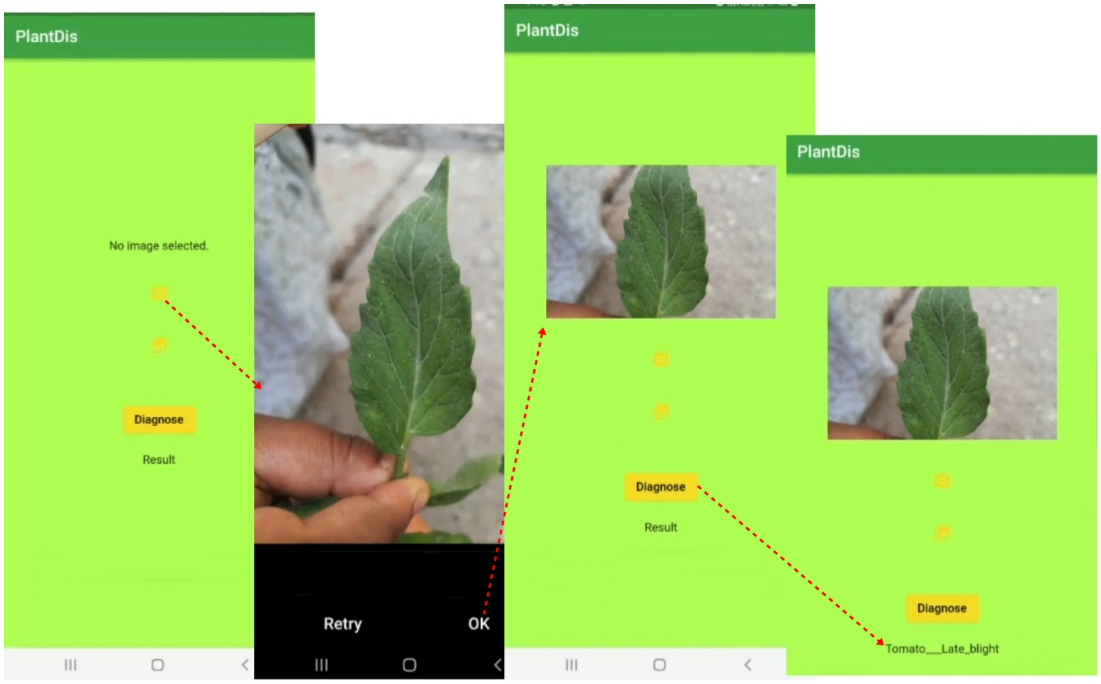


Figure 4.2: Mobile App Demo

4.2 Desktop Application

We have a mobile app then why do we need a desktop app then? The reason is that some models may require complex preprocessed image which is not well supported by flutter packages. The flutter frontend for the desktop application was adapted from the mobile app, and then the mlhub package was used to connect it to the Machine Learning Model(Fig. 4.3). The desktop app is currently tested only on Ubuntu.

Figure 4.3: Desktop App Pipeline

MlHub is an opensource package that supports both python and R. The MlHub python package was used in this project, since MlHub supports python thereby it supports complex image preprocessing and may be able to handle more complex models than tflite. By writing simple python scripts the model can be connected to

MlHub. MlHub also has a command line interface and it can be used independently without flutter to interact with a machine learning model.

Similar to what discussed in the mobile app section, the frontend's job is to assist users in collecting images and then submitting them to the mhub package, which performs any necessary image preprocessing before passing the preprocessed picture to the model. The ML model then utilises the image to build probability vectors, which it then sends back to the mlhub package, which filters the class with the greatest probability and returns it to the frontend. The predicted class(disease) is then displayed to the user by the frontend. The File Picker package is another crucial package that the front-end relies on to collect images.

The demo can be seen in Fig. 4.4 below. The user can choose the image from using the gallery icon. After selecting the image, the user has to press the OK button, and then the name of the image selected will be shown to the user. Then, clicking on the Diagnose Button the user gets the result of the disease prediction along with the image selected. The result for demo is healthy corn plant.

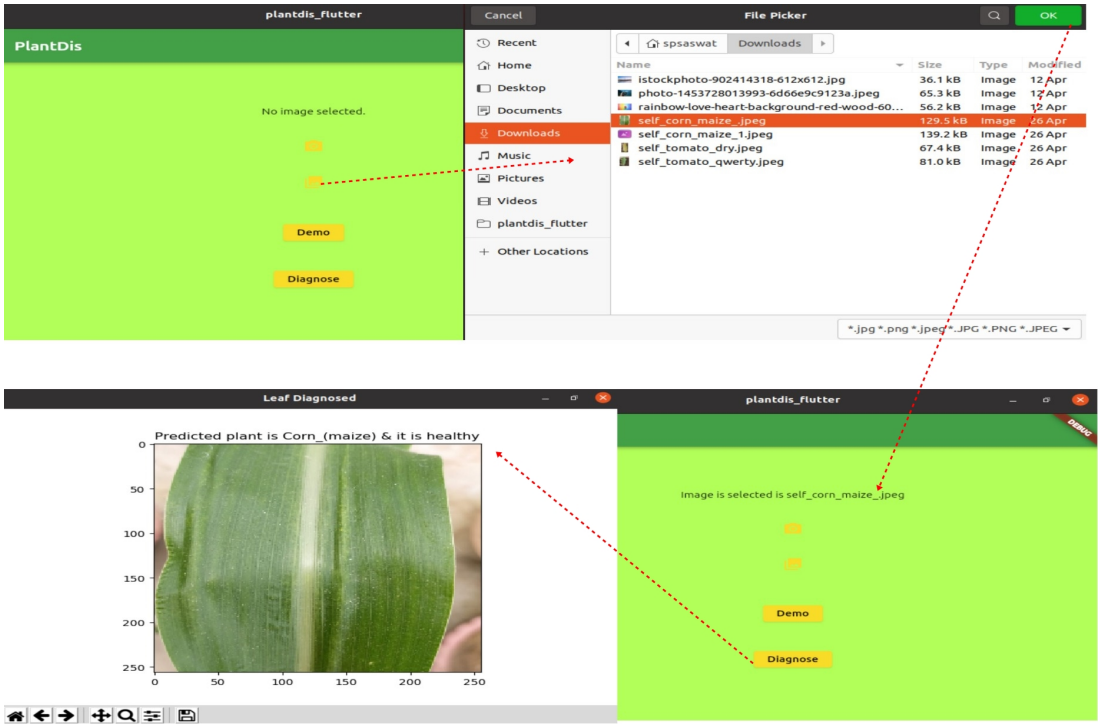


Figure 4.4: Desktop App Demo

It can be seen in Fig. 4.4, that there is a Demo button. On pressed, it runs the demo script of MlHub in the background. The demo script has been designed(during this thesis) as such that it will randomly select three images from withheld test set pass the images, one by one to the model, to get the predicted output for each image and show it to the user along with the the image. Another Mlhub script is the script for diagnose whose functionality has already been covered in the third paragraph of this section.

4.3 Summary

In this chapter we discussed about the functionality of the applications developed as a part of this thesis, and also saw demo of their use. In the next chapter we will see the Results from the experiment conducted on Nested Transfer learning.

Chapter 5

Results

This chapter contains the Results from the different experiments conducted on Nested Transfer Learning on the labelled test images witheld from the dataset, and as well on the images collected using smartphone during this research.

5.1 Results from the test of labelled withheld images

5.1.1 Small Dataset - 3 classes

The Nested Transfer Learning approach achieves an accuracy of 100% on the witheld test set consisting of 6 images as shown in Fig. 5.1.

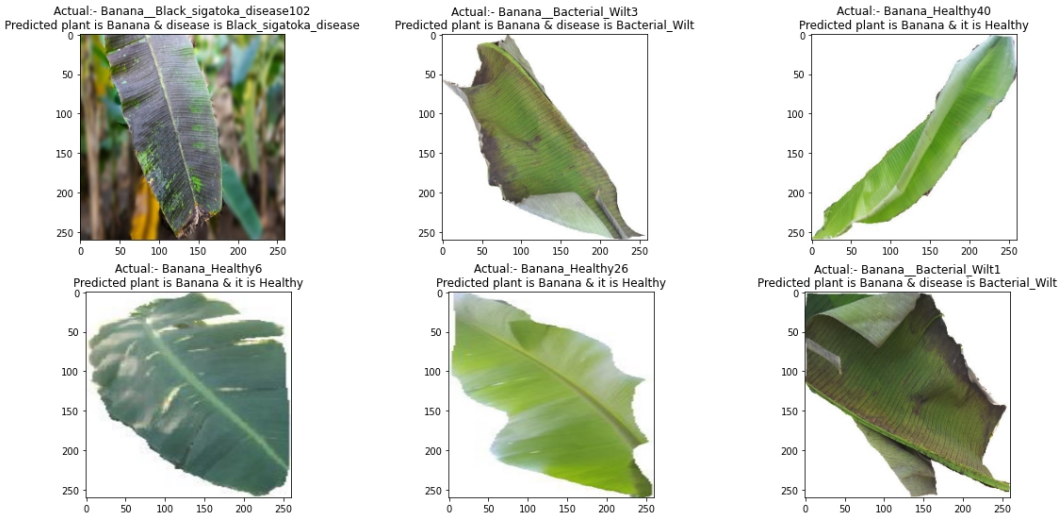


Figure 5.1: The Results from the withheld test set using Nested Transfer Learning

5.1.2 Expanded ImBalanced Dataset - 24 classes

The Nested Transfer Learning approach achieves an accuracy of 100% on the witheld test set consisting of 39 images.

5.1.3 Expanded Balanced Dataset - 22 classes

Fig. 5.2 shows the results of nested transfer learning on the witheld test set. The accuracy achieved on the set is 100%.

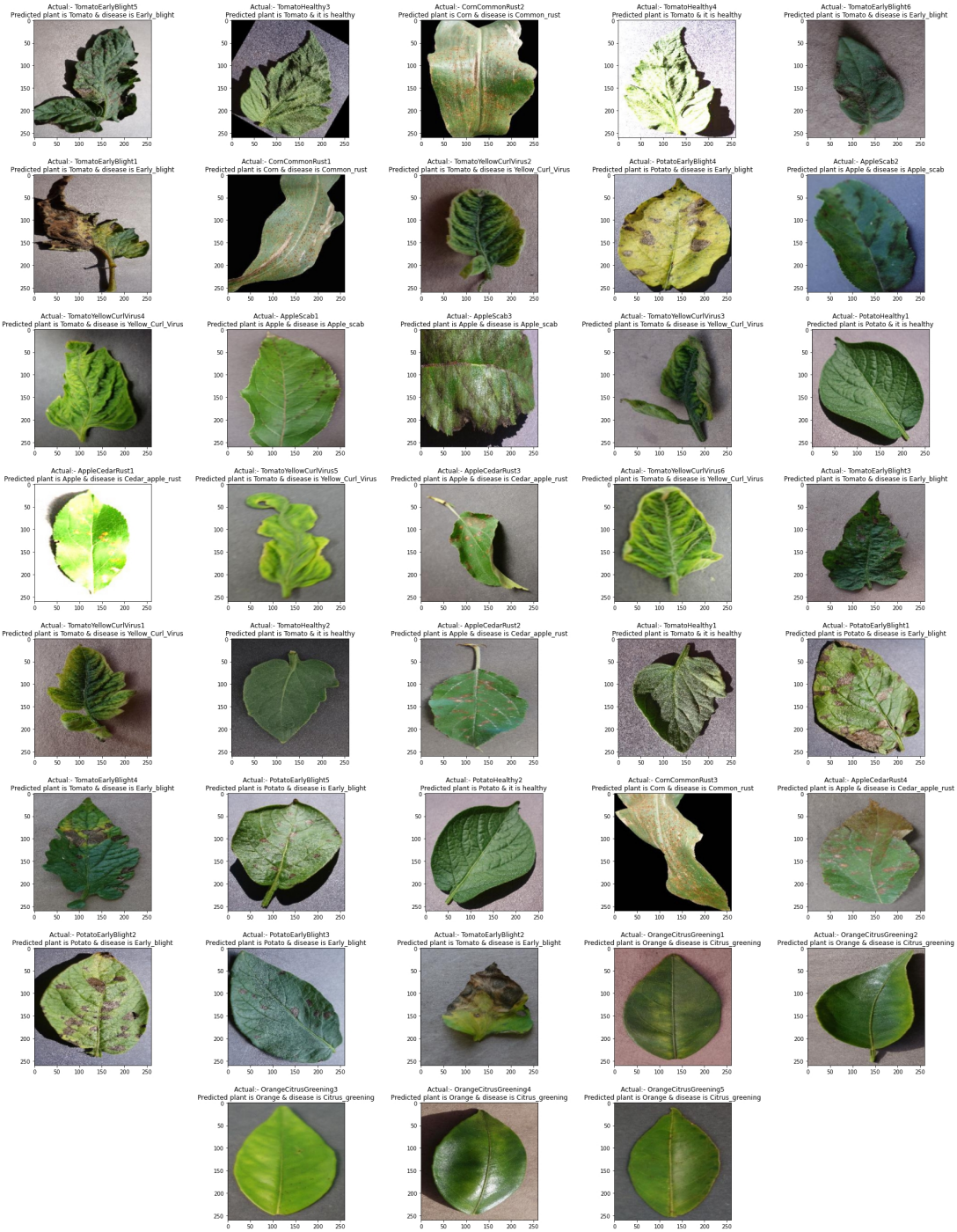


Figure 5.2: The Results from the withheld test set of 38 images, predicted by the model using nested transfer learning, covering most of the 22 classes.

5.1.4 Very Small Dataset

Here nested transfer learning was rigorously tested with 32 witheld test images belonging to the 4 classes to find out a 100% accuracy. In the Fig. 5.3 below, only one prediction from each class has been included for simplicity.

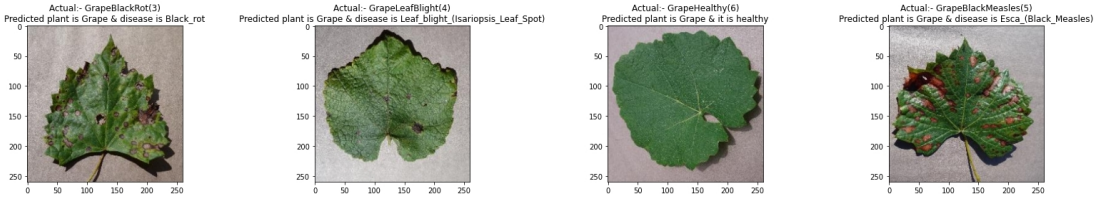


Figure 5.3: The Results from the withheld test set of 38 images, predicted by the model using transfer learning, covering most of the 22 classes.

5.2 Results from images taken by smartphone

5.2.1 Using Transfer Learning on 21 Classes

Fig. 5.4 below includes results images on images taken by smartphone during this thesis, and they are included for comparison with nested transfer learning approach.

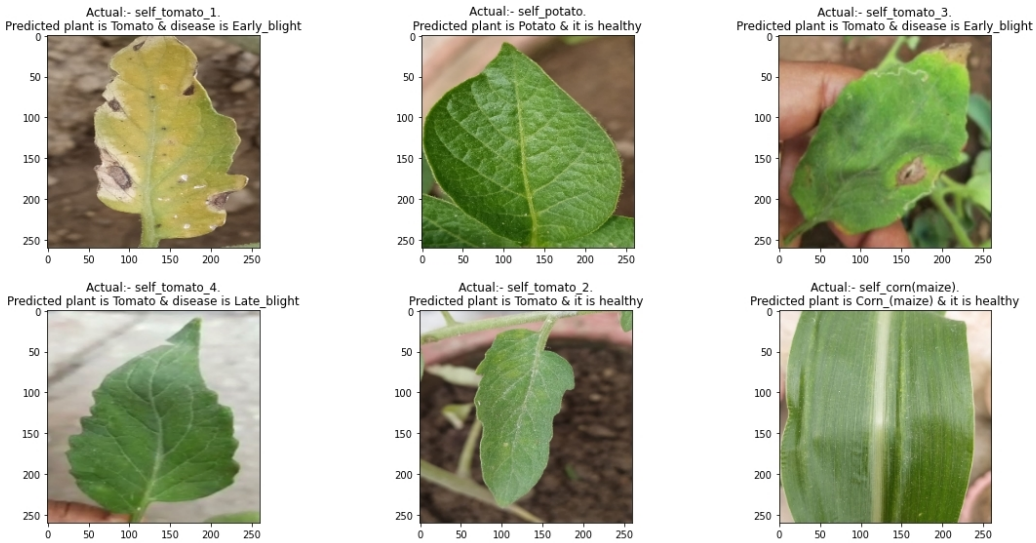


Figure 5.4: The Results of prediction by transfer learning from images taken by smartphone.

5.2.2 Expanded Balanced Dataset - 22 classes

The results are exactly the same as can be seen in Fig. 5.4.

5.2.3 Small Dataset - 3 classes

In the Fig. 5.6 below we can see the possible correct predictions using the nested transfer learning approach.

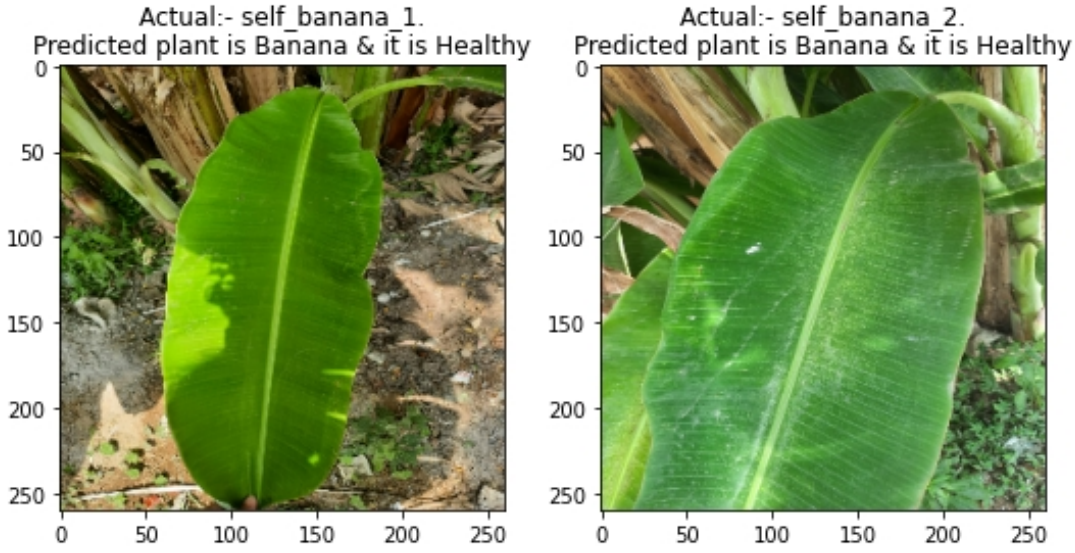


Figure 5.5: Possibly correct predictions by Nested Transfer Learning on Small Dataset

5.2.4 Expanded ImBalanced Dataset - 24 classes

In the Fig. 5.6 below we can see some of the wrong predictions by the nested transfer learning approach. These wrong predictions indicate that model might be overfitting on the dataset.

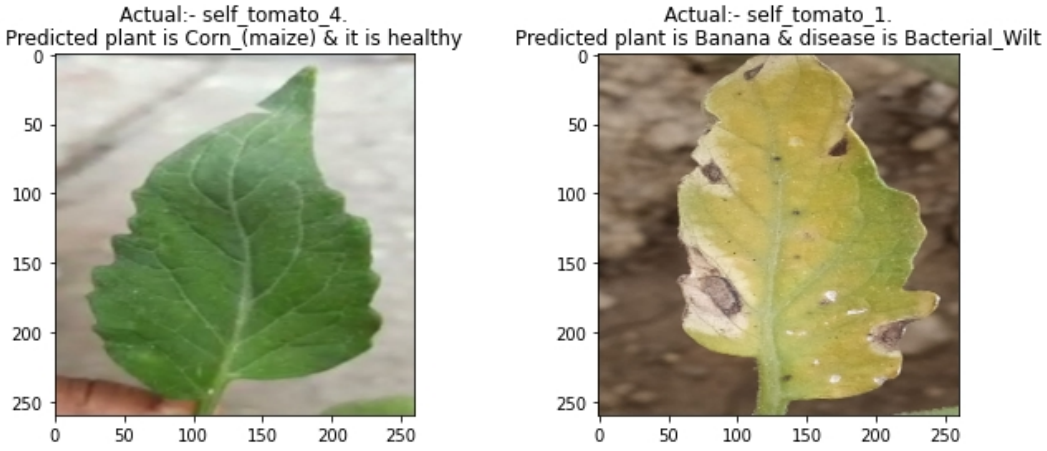


Figure 5.6: Wrong predictions by Nested Transfer Learning for imbalanced dataset

The Fig. 5.7 below shows the possible wrong predictions, because the banana

leaf images seem healthy, which if true further indicates overfitting. The wrong and possibly wrong predictions may also be caused due to the noisy background.

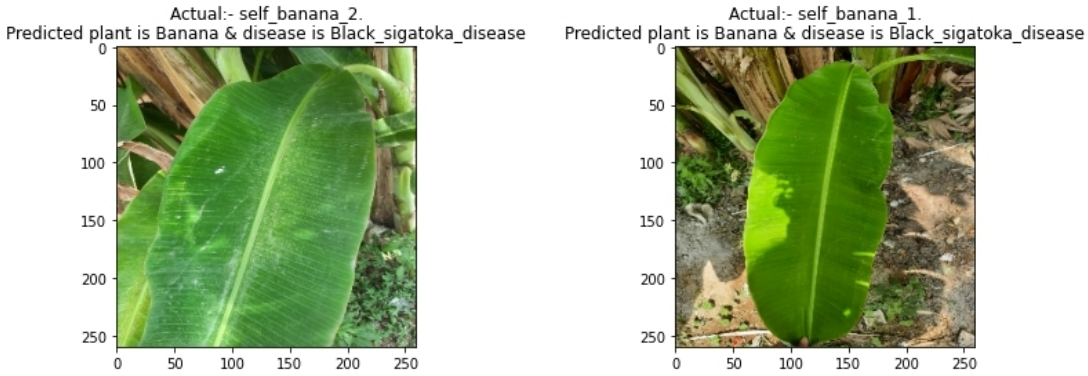


Figure 5.7: Possibly wrong predictions by Nested Transfer Learning for imbalanced dataset

5.3 Summary

In this chapter we saw the results from different experiments on Nested Transfer learning. The results on the withheld test set has an accuracy of 100% for every but, in images taken by smart phone, the Nested transfer learning was found to be performing good in all scenarios except for imbalanced datasets. This indicates that nested transfer learning approach is not suitable for imbalanced datasets, and may also be not suitable for merging datasets with different quality(dataset 1 and small dataset).

Chapter 6

Conclusion

In this thesis, the main highlights were using transfer learning to predict plant diseases for 21 classes with a validation accuracy of 99.4%. Also, nested transfer learning was used in various datasets, and was found to be performing very well especially on small sized datasets achieving 100% accuracy on witheld test sets. It achieved a validation accuracy of 99.24% with just 3 epochs for expanded datasets(22 classes). Nested Transfer Learning was also found to be very quick in learning requiring almost the half number of epochs as compared to transfer learning to achieve a high accuracy. The softwares developed as a part of the thesis are open-sourced, easily accessible, and should be easily usable.

6.1 Future Work

The deep learning models were useful in getting a high accuracy for disease prediction. Now, this work can be expanded to train deep or machine learning models that can correlate between the yield loss and disease detected taking the help of other factors related to plants.

Different approaches to control over fitting such as adding new regularization layers can be tried while performing Nested Transfer Learning on Expanded datasets. Nested Transfer learning can be tested on different large datasets with more number of classes for plant disesases(>21). Nested transfer learning can be explored for various datasets apart from plant diseases to test its performance. Also, various comparision can be done between nested transfer learning and transfer learning for various fields.

The softwares developed during this project though usable in real world scenarios, are still not scalable for a large farmland. A methodology can be developed to automatically collect the data from large farmloads, pre-process and feed them to the software for a large scale application.

Bibliography

[1] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, 2016. (cited on pages 1, 2, and 3)

[2] L. Li, S. Zhang, and B. Wang, "Plant disease detection and classification by deep learning—a review," *IEEE Access*, vol. 9, pp. 56683–56698, 2021. (cited on page 1)

[3] ABARES, "Snapshot of australian agriculture 2022," 03 2022. (cited on page 1)

[4] S. Yang, L.and Hanneke and J. Carbonell, "A theory of transfer learning with applications to active learning," 07 2012. (cited on page 1)

[5] G. G. and A. P. J., "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers Electrical Engineering*, vol. 76, pp. 323–338, 2019. (cited on page 3)

[6] M. H. Saleem, J. Potgieter, and K. M. Arif, "Plant disease detection and classification by deep learning," *Plants*, vol. 8, no. 11, 2019. (cited on page 3)

[7] J. Chen, J. Chen, D. Zhang, Y. Sun, and Y. Nanehkaran, "Using deep transfer learning for image-based plant disease identification," *Computers and Electronics in Agriculture*, vol. 173, p. 105393, 2020. (cited on page 3)

[8] M. S. Arshad, U. A. Rehman, and M. M. Fraz, "Plant disease identification using transfer learning," in *2021 International Conference on Digital Futures and Transformative Technologies (ICoDT2)*, pp. 1–5, 2021. (cited on page 3)

[9] A. P. J and G. GOPAL, "Data for: Identification of plant leaf diseases using a 9-layer deep convolutional neural network," 04 2019. (cited on pages 5 and 7)

[10] D. P. Hughes and M. Salathe, "Data for: An open access repository of images on plant health to enable the development of mobile disease diagnostics," 04 2016. (cited on pages 5 and 7)

[11] G. Owomugisha, J. Quinn, E. Mwebaze, and J. Lwasa, "Data for: Automated vision-based diagnosis of banana bacterial wilt disease and black sigatoka disease," 12 2014. (cited on page 6)

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. (cited on page 8)

[13] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016. (cited on page 8)

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. (cited on page 8)

[15] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019. (cited on pages 8 and 10)

[16] S. Panda, A. S. Tiwari, and M. R. Prusty, "Comparative study on different deep learning models for skin lesion classification using transfer learning approach," IJSRP, 01 2021. (cited on page 8)

Appendix

Model Architectures

The model architectures contained the layers, output shape and number of trainable parameters.

Model From Scratch

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 conv2d_1 (Conv2D)           (None, 256, 256, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 85, 85, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 85, 85, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 28, 28, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

 conv2d_4 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 9, 9, 128)        0
 2D)

 conv2d_5 (Conv2D)           (None, 9, 9, 128)         147584

 conv2d_6 (Conv2D)           (None, 9, 9, 256)         295168

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 256)        0
 2D)

 conv2d_7 (Conv2D)           (None, 3, 3, 256)         590080

 conv2d_8 (Conv2D)           (None, 3, 3, 512)         3277312

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 1536)              7079424

 dropout (Dropout)           (None, 1536)              0

 batch_normalization (BatchN  (None, 1536)             6144
 ormalization)

 dropout_1 (Dropout)         (None, 1536)              0

 dense_1 (Dense)             (None, 24)                36888

=================================================================
Total params: 11,572,024
Trainable params: 11,568,952
Non-trainable params: 3,072
```

DenseNet based

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet201 (Functional)    (None, 8, 8, 1920)        18321984

 flatten (Flatten)           (None, 122880)            0

 batch_normalization (BatchN  (None, 122880)           491520
 ormalization)

 dropout (Dropout)           (None, 122880)            0

 batch_normalization_1 (Batc  (None, 122880)           491520
 hNormalization)

 dropout_1 (Dropout)         (None, 122880)            0

 dense (Dense)               (None, 21)                2580501

=================================================================
Total params: 21,885,525
Trainable params: 21,164,949
Non-trainable params: 720,576
_________________________________________________________________
```

MobileNet based

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenet_1.00_224 (Functio  (None, 8, 8, 1024)       3228864
 nal)

 flatten_1 (Flatten)         (None, 65536)             0

 batch_normalization_1 (Batc  (None, 65536)            262144
 hNormalization)

 dropout_1 (Dropout)         (None, 65536)             0

 dense_1 (Dense)             (None, 21)                1376277

=================================================================
Total params: 4,867,285
Trainable params: 4,714,325
Non-trainable params: 152,960
_________________________________________________________________
```

EfficientNet based

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb2 (Functional)  (None, 9, 9, 1408)       7768569

 flatten_1 (Flatten)         (None, 114048)            0

 batch_normalization_1 (Batc  (None, 114048)           456192
 hNormalization)

 dropout (Dropout)           (None, 114048)            0

 batch_normalization_2 (Batc  (None, 114048)           456192
 hNormalization)

 dropout_1 (Dropout)         (None, 114048)            0

 dense (Dense)               (None, 21)                2395029

=================================================================
Total params: 11,075,982
Trainable params: 10,552,215
Non-trainable params: 523,767
_________________________________________________________________
```

EfficientNet

Github Repository

The open source github repository can be found using the link:- github.com/spsaswat/plantdis. The Fig. 1 currently has six folders. The folder ipynb has the notebooks used for training the models. The op_m_readme has some output images used in readme of main page. The mlhub folder contains the scripts for using the mlhub package and connecting it to the model. The plantdis_flutter has all the code used for developing desktop application. The plantdis_mob has code for the mobile application. The self_tests folder contain images collected using smartphone for testing purposes. The test folder contains the images used for testing.

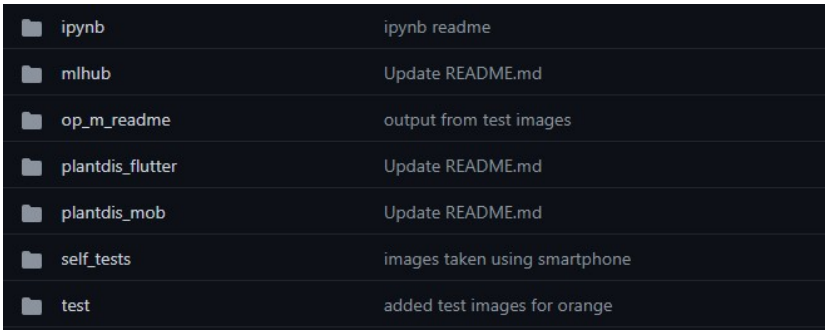


Figure 1: Folders in Github Repo

MLHUB

The official website of mlhub is mlhub.ai. The Fig. 2 is the yaml file of mlhub which includes all the packages and files used in the mlhub python script.

```yaml
--- # This is a comment and otherwise ignored.
meta:
  name         : plantdis
  title        : Detect plant disease from an image of a leaf.
  keywords     : transfer learning, tensorflow, deep learning, plant disease, python
  version      : 0.1.3
  languages    : py
  license      : gpl3
  author       : u7156387@anu.edu.au
  url          : https://github.com/spsaswat/plantdis
dependencies:
  python3:
    - opencv
    - numpy
    - requests
    - matplotlib
  pip3:
    - tensorflow-cpu>=2.8.0
    - gdown
    - argparse
    - pathlib
  files:
    - mlhub/README.md
    - mlhub/demo.py
    - mlhub/diagnose.py
    - test/
    - https://drive.google.com/uc?id=1m0bU-NCqzfjO37HFBEm4-2YmTQwQzil6: pd_densenet201_6.h5
commands:
  demo : Demonstrate disease classification for a default image
  diagnose : Predict disease for a supplied image of a diseased leaf
```

Figure 2: mlhub YAML

Development Environment

The development tools used were Tensorflow, Flutter, Jupyter Lab and Google Colab. Since, I did not have GPU I used Google Colab to access the free GPU for 12 hrs. The problem with google colab is that, the model can be trained for a few epochs(for large datasets) at a time, therefore you need to train save and reload the model. Visual Studio Code and Android Studio were used for User Applications developments. Also, a virtual environment was setup for Ubuntu in Windows using Oracle Virtual machine. Finally, to write this report overleaf was used.