Image Based Plant Disease detection Using Deep Learning

Group number 24

Group member details:-

Shashank Pratap Singh	2019B4A70956H
Koteswarudu Akula	2019A7PS0035H
G SREEKAR VIJAY	2020A7PS0266H
S SURYA KIRAN	2019A7PS0014H

Introduction

Vegetable and fruit plants facilitate around 7.5 billion people around the globe, playing a crucial role in sustaining life on the planet. The rapid increase in the use of chemicals such as fungicides and bactericides to curtail plant diseases is causing negative effects on the agro-ecosystem. The high scale prevalence of diseases in crops affects the production quantity and quality. Solving the problem of early identification/diagnosis of diseases by exploiting a quick and consistent reliable method will benefit the farmers. In this context, we shall be looking at papers that focus on classification and identification of plant leaf diseases using convolutional neural network (CNN) techniques.

Research Paper Review

We selected research paper 4 from our initial report to work on. This paper is a review of different papers which uses different ways of deep learning for plant leaf disease detection. In this paper different approaches are categorized into groups such as

1) Classic Deep Learning Architectures For Leaf-disease Detection
2) New/Modified DL Architectures For Leaf-disease Detection
3) Target Detection of Plant Diseases For Leaf-disease Detection
4) Hyper-Spectral Imaging(HSI) With DL Models etc…

First group describes the classical deep learning techniques of using traditional convolutional architectures such as VGG-16, 19, Inception V3, Xception, Resnet etc..
Second group contains papers which use modified deep learning architectures such as introducing some new attention modules into the traditional convolution to make them more effective.
Third group of papers mainly concentrate on visualizing or to locate the lesion areas where disease started for which they use faster R-CNN, R-FCN and SSD architectures.

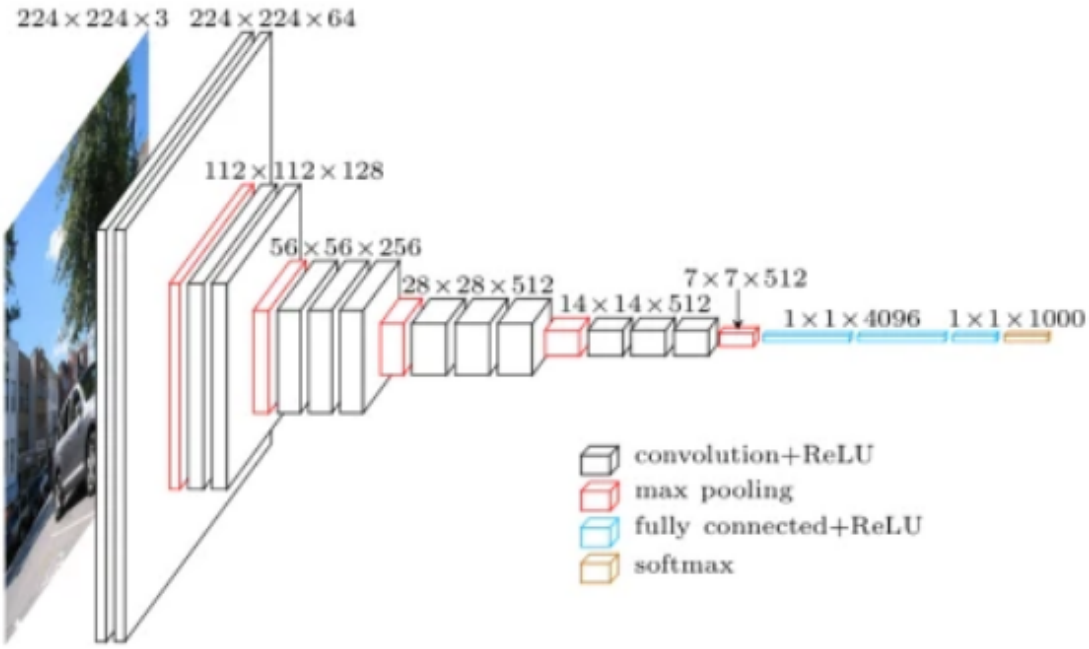
Fourth group addresses the problem of two different diseases having the same kind of symptoms and damages for the leaf which makes it difficult for our naked eye or even computer vision to know the difference. So Instead of using normal images they used HSI images which are sensitive to little leaf changes and can classify the diseases better.

Paper Selection:

Among all the different given papers, we decided to pick the one about [Optimizing Pre Trained Convolutional Neural Networks for Tomato Disease detection].

This paper considers 4 well known convolutional neural networks (CNNs) architectures for identification and classification of diseases in the tomato plant leaf. The first two being two VGG models (VGG-16 and VGG-19), Residual Neural Network (ResNet50), and Inception V3.

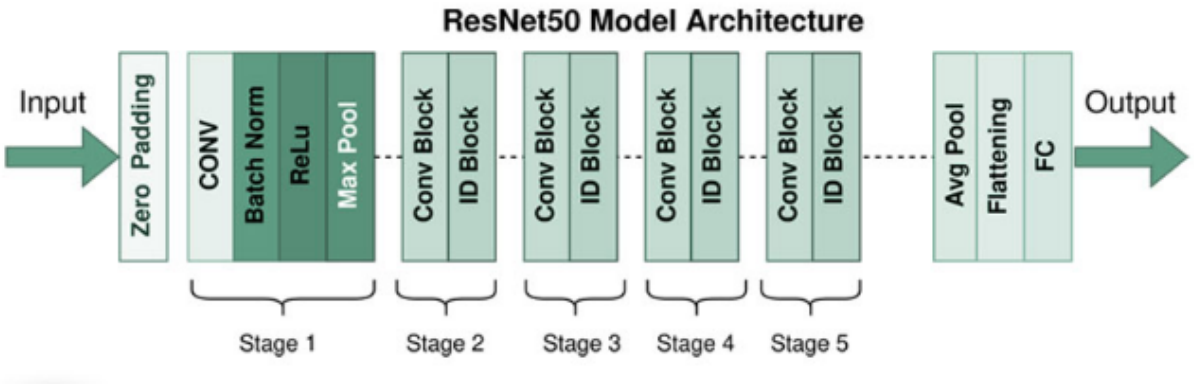
1. **VGG Net :** This particular pre-trained model was introduced by the Visual Geometric Group (VGG) at the University of Oxford. The basic working principle of this model is that it uses deeper layers with smaller filters.



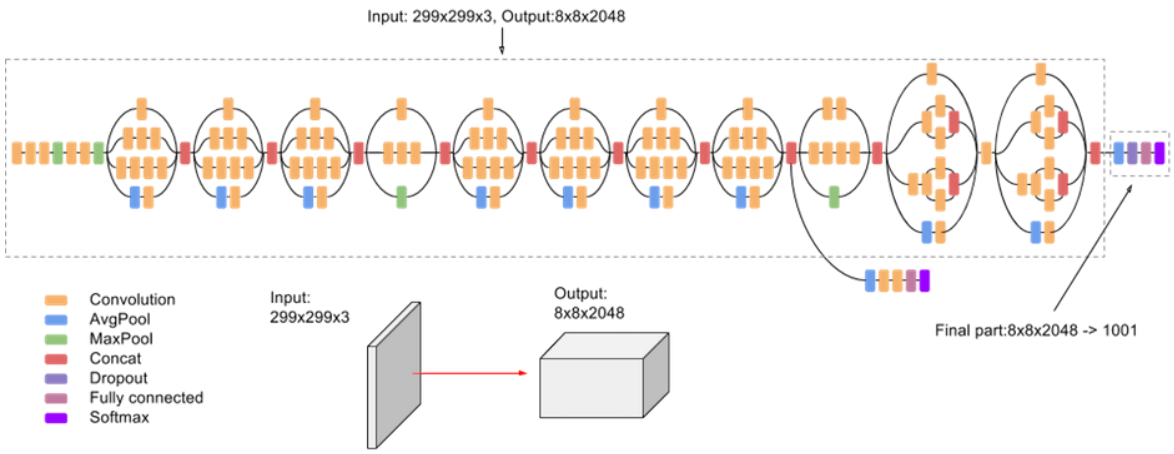
The input layer dimension of the VGG architecture is set for an image size of 244 × 244. Preprocessing involves subtraction of the mean RGB value from each pixel of the input image. Preprocessing is followed by a stack of 5 convolutional layers, each of which is followed by a MaxPool layer, i.e., each set of convolutional layers is followed by a MaxPool layer. (e final MaxPool layer precedes three fully connected (FC) layers. The first two FC layers have 64 × 64

(4096) channels, whereas the last FC layer has 1000 channels, which is followed by a softmax activation function. VGG network comes in multiple different forms. Two of them are VGG-16 and VGG-19. They both use the same architecture with different numbers of layers. VGG-16 uses 16 layers, while VGG-19 uses 19 layers. The differentiating factor is the number of convolution layers in the 3rd, 4th, and 5th layers of convolutional layers stacks.

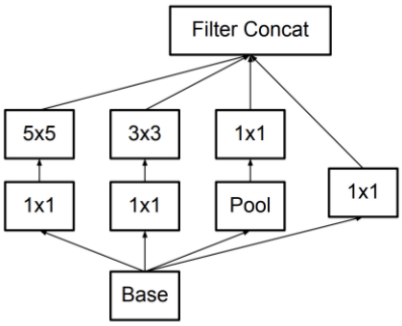
2. **ResNet :** Residual network (ResNet) addressed the problem of training and overfitting in deep neural networks by introducing the concept of residual learning. As neural networks become deeper and more complex, degradation occurs. Degradation is the phenomenon where the training error keeps increasing as we keep adding on more layers to the network. To combat this problem, the makers of ResNet added a thing called the Residual Block. ResNet attempts to identify a residual mapping between the input to the convolutional layers and the output at the MaxPool layer, thus eliminating the computational cost of input being processed by the convolutional layer stack. We shall be training the ResNet50 model. It is a CNN that is 50 layers deep.

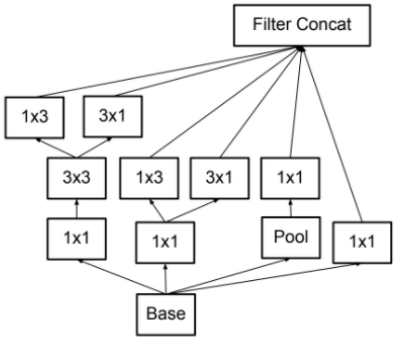


3. **Inception :** Inception is a modified CNN model that tries to achieve improved performance by increasing the depth of the network and keeping the computational cost low. It is a culmination of several ideas developed by various researchers throughout history. The downside of the original Inception network was its limited application adaptability in new use cases. It makes several improvements such as factorized 7 x 7 convolutions, for instance. The model is made up of symmetric and asymmetric building blocks, including convolutions, concatenations, average pooling, max pooling, and fully connected layers along with extensive usage of batch normalization.



Inception V3 involves using many smaller convolutions to replace a single larger convolution which leads to faster training. In comparison with its previous counterparts like inception V1 and V2 models, the V3 model has a deeper network without compromising the speed. Inception V3 was made more effective by using asymmetric convolutions of the form 1 x n followed by n x 1 to replace an n x n convolution.





Dataset

The research paper used two different sets of data. The first one is one with pictures taken in the laboratory in a controlled environment , and the second set is pictures taken from various different crop fields. The author did provide the [laboratory dataset], but did not provide the field dataset, making us unable to test our models on that.

Paper Results

The research paper had the implementation of the 4 given models by two different methods. Parameter tuning and Feature extraction. They had it tested on the two different datasets as well, them being the lab dataset, and the field dataset. The results showed that fine turning usually produces the better results among the two. InceptionV3 was the model that performed the best among all in all of the cases. One final observation was noted. It was the comparatively worse accuracy of all the models, in both of the methods, for the field dataset against the lab dataset.

Implementation

Feature Extraction:

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process. Feature extraction is the name for methods that select and /or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set. This process is mainly used when we want to reduce the number of resources needed for processing without using important data or relevant information. It can also lead to other advantages such as accuracy improvements, overfitting risk reduction and improved data visualization

The following procedure is the most typical manifestation of transfer learning in the context of deep learning:

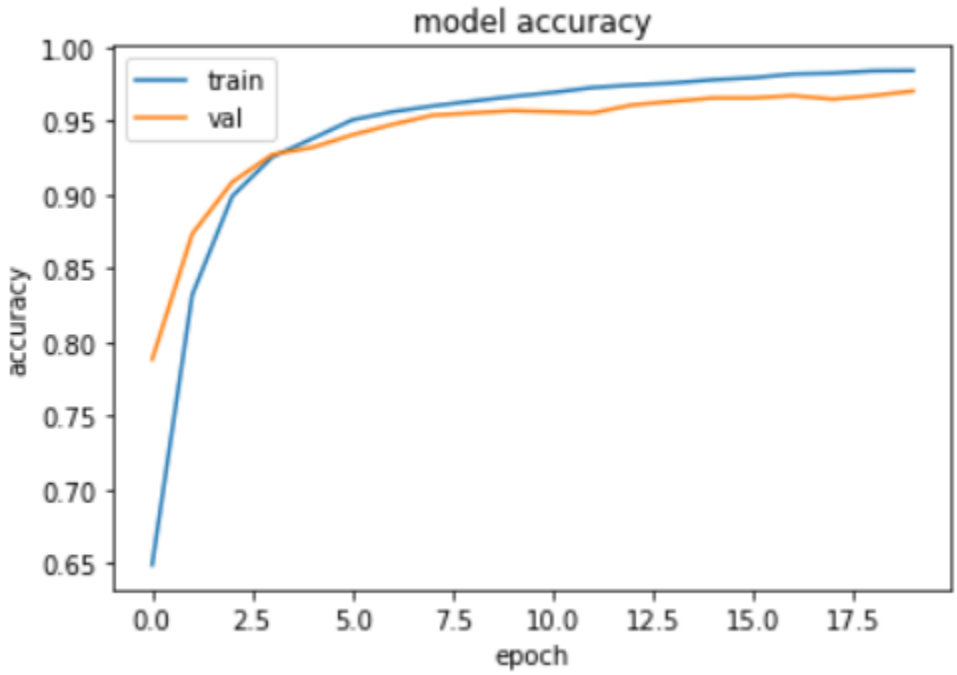
- Layers from a previously learned model are used.
- Freeze them to prevent subsequent training cycles from deleting any of the information they contain.
- On top of the frozen layers, add some additional trainable layers. They'll figure out how to transform previous characteristics into predictions on a fresh dataset.
- On your dataset, train the new layers.

Here we are using VGG, Inception V3, and ResNet as previously trained models, and on top we are adding two dense layers with activation functions relu and sigmoid, respectively. The first dense layer contains 512 neurons, and the second layer contains 8 neurons, representing the number of classes.

The results are as shown below

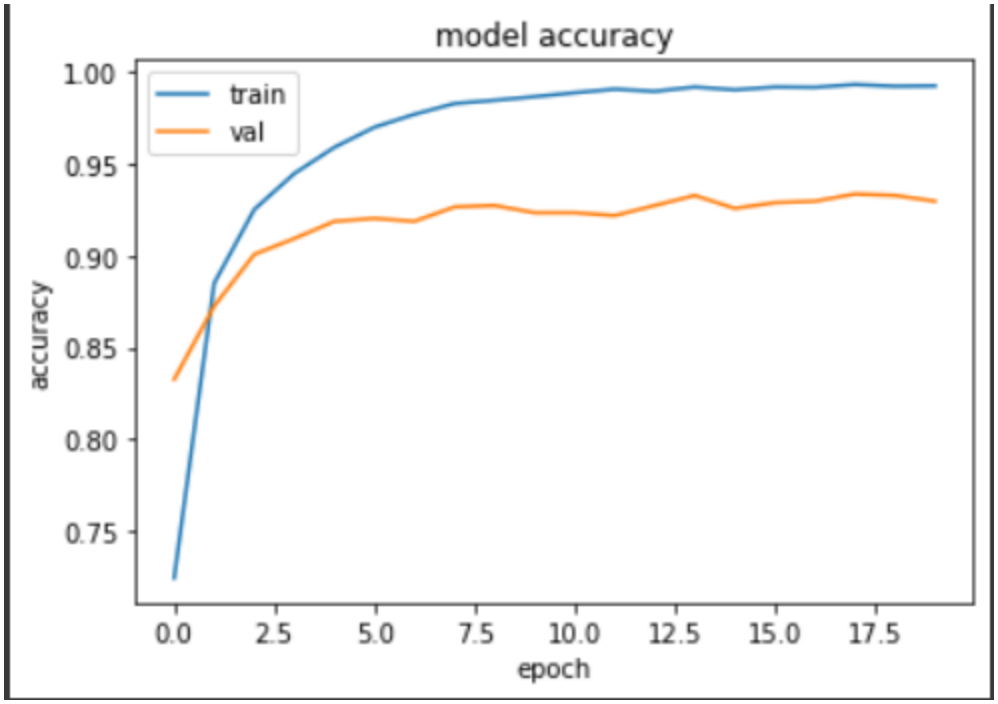
VGG:

```
40/40 [==============================] - 2s 56ms/step - loss: 0.1170 - accuracy: 0.9703
Loss = 0.11696521192789078
Test Accuracy = 97.02892899513245%
```



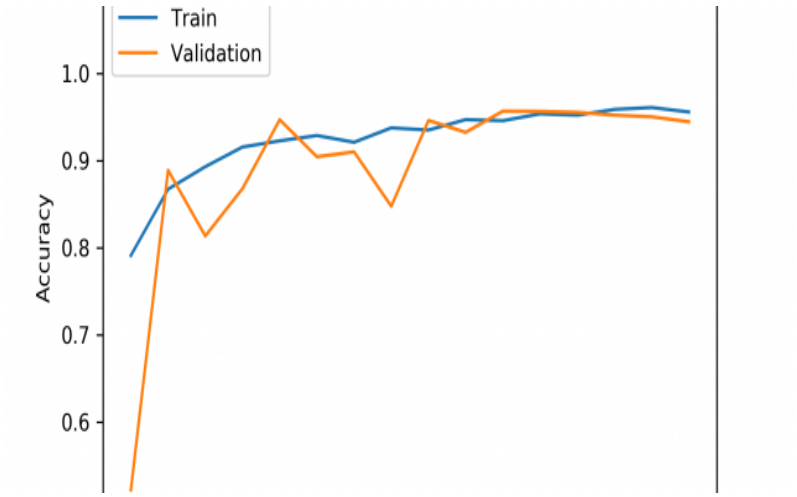
INCEPTION-V3

```
40/40 [==============================] - 1s 34ms/step - loss: 0.1925 - accuracy: 0.9335
Loss = 0.19246521592140198
Test Accuracy = 93.35418343544006
```



RESNET-50:

```
40/40 [==============================] - 1s 32ms/step
Loss = 0.29887962341308594
Test Accuracy = 90.2578579902649%
```



Code link : [Feature Extraction]

Fine Tuning Parameters

Fine-tuning parameters is a super-powerful method to obtain image classifiers on your own custom datasets from pre-trained CNNs. It is a multi step method that requires us to not only have updated the CNN architecture, but also re-trained it to have learned new object classes. The given paper requires us to take pre-trained neural networks, and apply parameter tuning to it, trying to achieve a much better accuracy than the pre-trained model. This specific method requires us to :

1. Remove the fully connected nodes at the end of the network
2. Replace the fully connected nodes with freshly initialized ones
3. Freeze certain CONV layers in the network
4. Start training , but only train the fully connected layer heads
5. Optionally unfreeze some/all of the CONV layers in the network and perform a second pass of training.

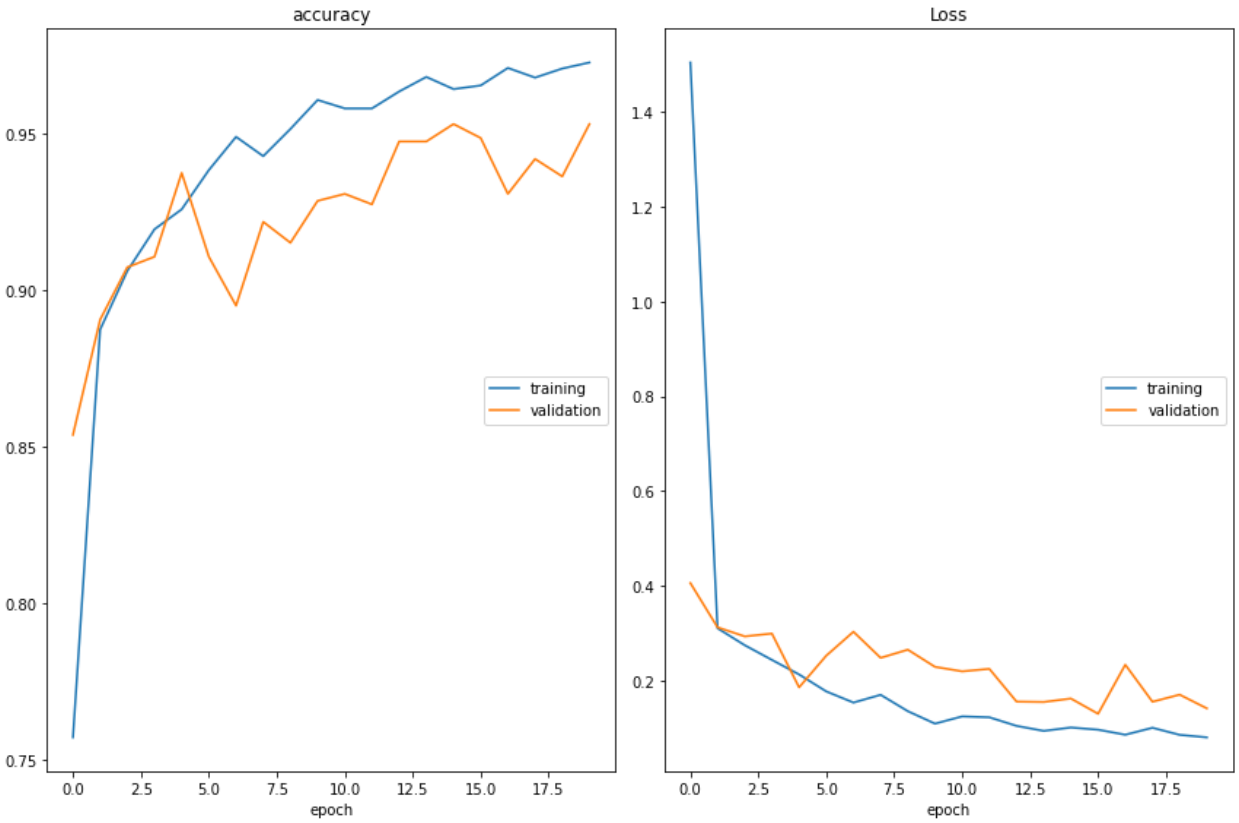
It is very important that we take the learning rate as something very small so that the newly connected FC layers can learn patterns from the previously learned CONV layers. This method almost always generates better accuracy than the method implementing feature extraction.

For the implementation of fine tuning, we took pre-trained models for VGG-16, VGG-19 and ResNet50 , while not deciding to include the fully connected layers from the networks. After doing that, we decided to freeze all but 2 convolutional layers, which allowed them to update their parameters during the training session. That was followed by adding a new fully connected system to the model. We started off by flattening the inputs, and then adding 2 dense layers with the ReLu activation function. We also decided to add a dropout function, with the parameter 0.2, which would help with the overfitting of the model. In the end, we decided to add an output layer with the SoftMax activation function.
For the inceptionV3 model, we made all layers except batch normalization layers trainable. Then we flattened the input, and added two dense layers, one with ReLu activation function, and the other with Sigmoid function.
The learning rate was decided to be 0.0001 for all 4 models.

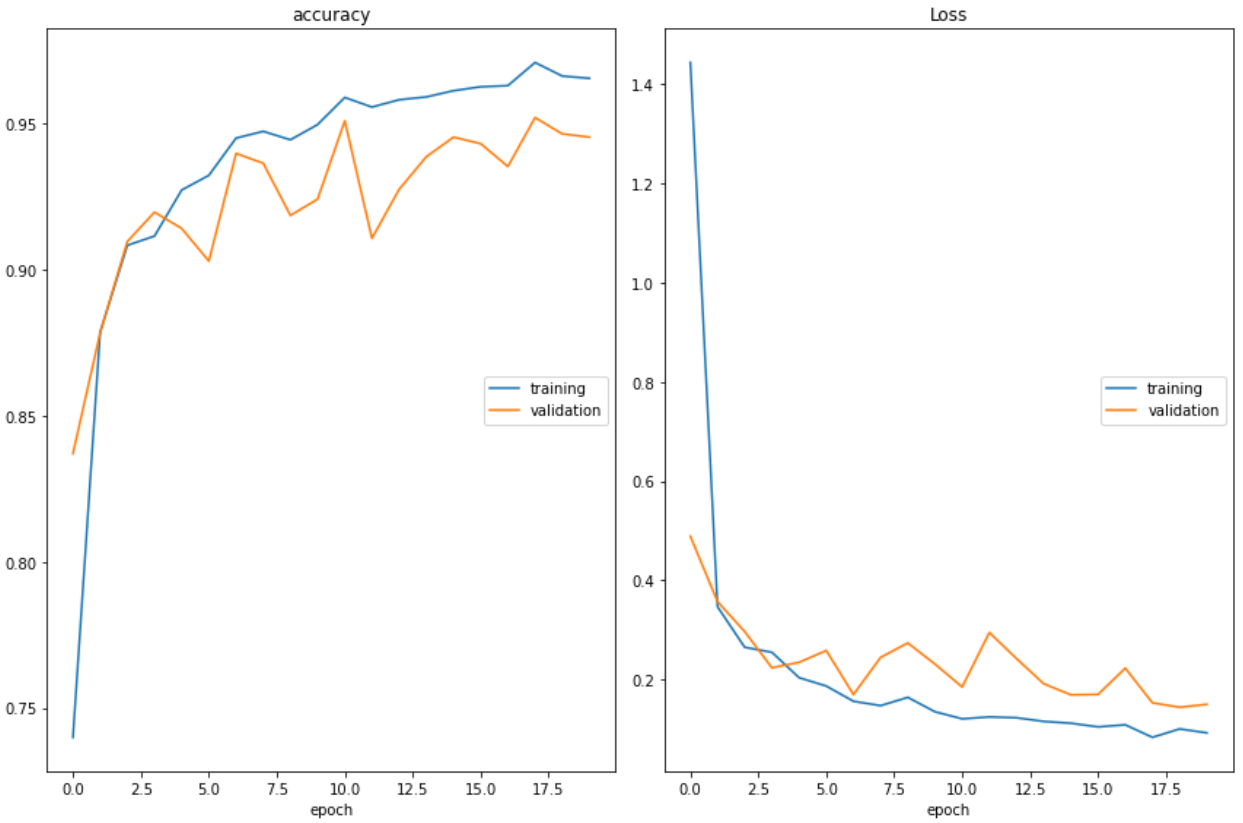
Results

1. **VGG-16 :-**

```
accuracy
        training                (min:    0.757, max:    0.973, cur:    0.973)
        validation              (min:    0.854, max:    0.953, cur:    0.953)
Loss
        training                (min:    0.080, max:    1.505, cur:    0.080)
        validation              (min:    0.130, max:    0.406, cur:    0.141)
```

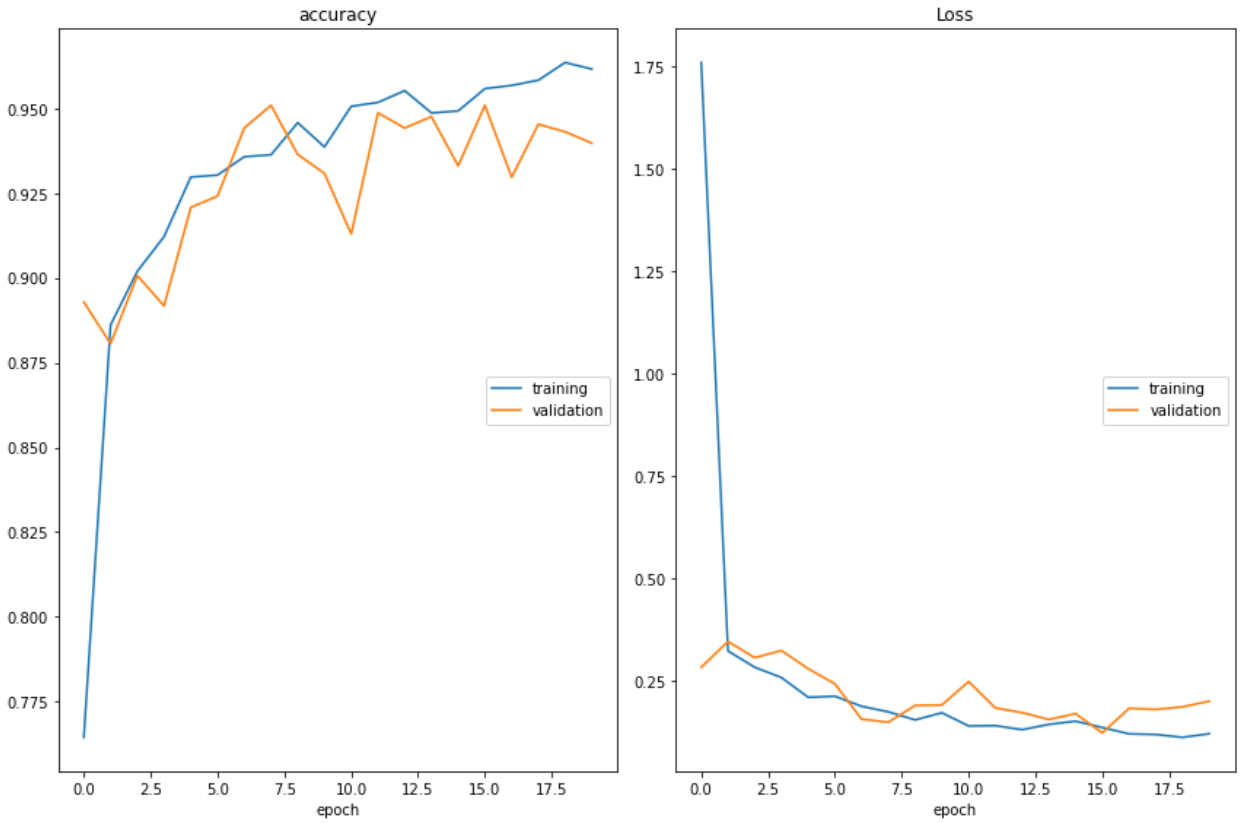

accuracy
0.95
0.90
0.85
0.80
0.75
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch
training
validation
Loss
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch
training
validation

2. VGG-19 :-



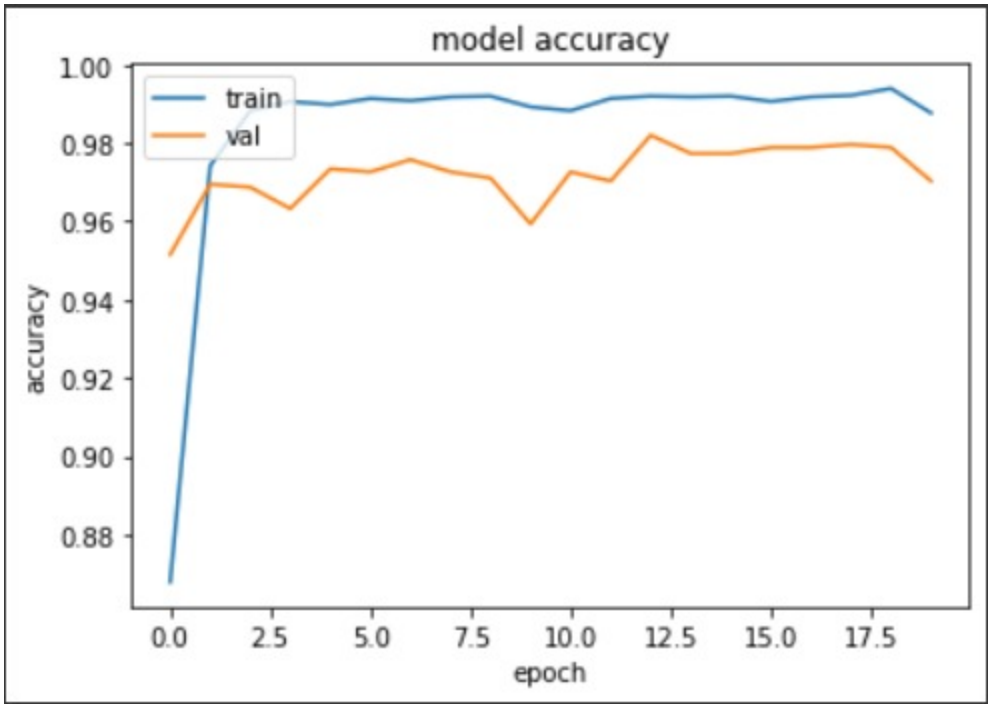

```
accuracy
        training                (min:    0.740, max:    0.971, cur:    0.965)
        validation              (min:    0.837, max:    0.952, cur:    0.945)
Loss
        training                (min:    0.084, max:    1.444, cur:    0.093)
        validation              (min:    0.144, max:    0.489, cur:    0.150)
```

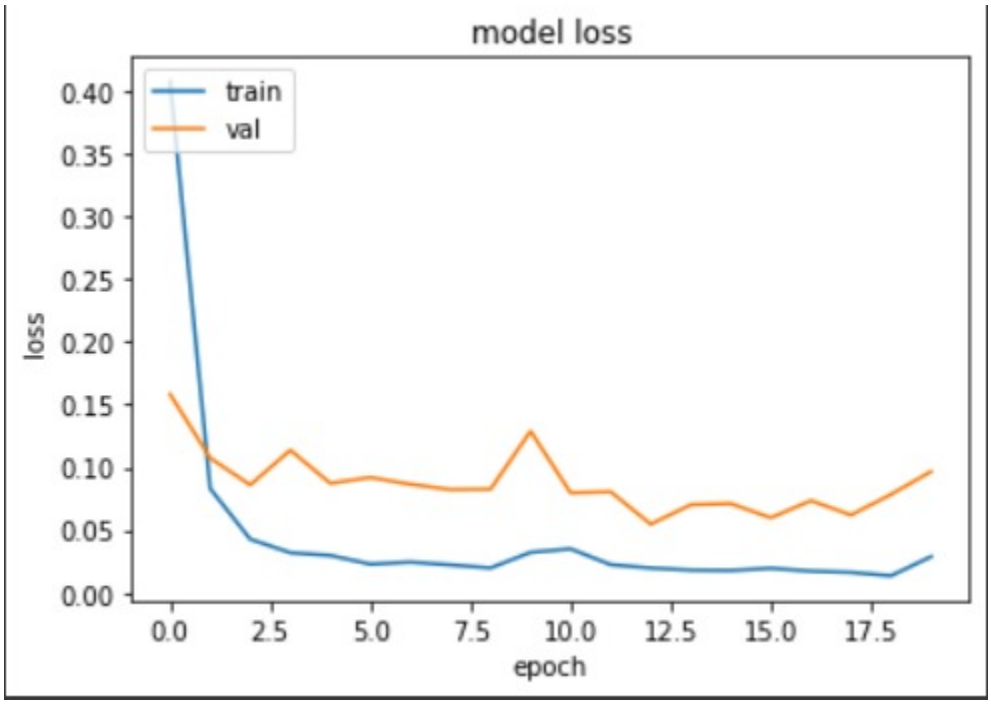
3. ResNet50 :-




```
accuracy
        training                (min:    0.764, max:    0.964, cur:    0.962)
        validation              (min:    0.881, max:    0.951, cur:    0.940)
Loss
        training                (min:    0.113, max:    1.759, cur:    0.122)
        validation              (min:    0.124, max:    0.347, cur:    0.201)
```

4. **InceptionV3 :-**






```
40/40 [==============================] - 1s 33ms/step - loss: 0.0547 - accuracy: 0.9820
Loss = 0.05468590185046196
Test Accuracy% = 98.20172190666199
```

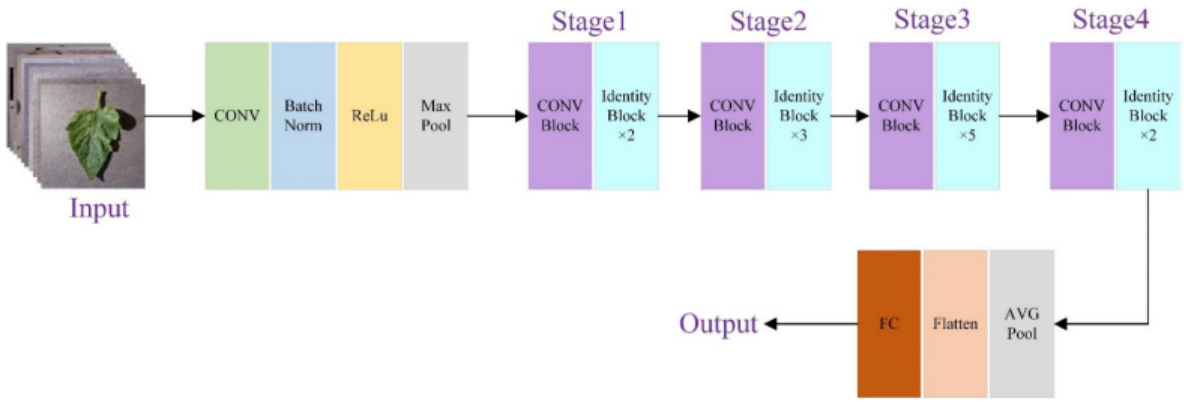
Code :- [Fine Tuning Parameter Implementation]

Implementation Results

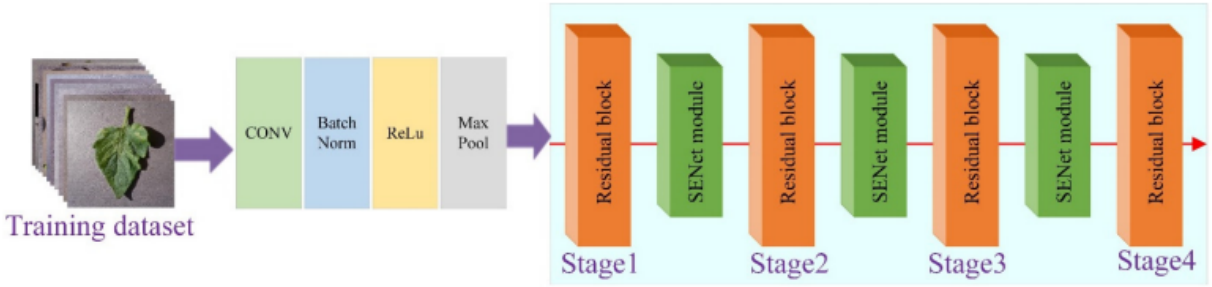
The inceptionv3 model proved to be the most accurate for the parameter tuning. Almost all of the accuracies in fine tuning seem to be more than the ones obtained in feature extraction, except for the one in the vgg-19 model. We tried various different methods, but failed to get a better accuracy for the fine tuned model. The research paper did not also mention the exact methods used for fine tuning and feature extraction, so that might have led to us taking different approaches while building the model, compared to the ones taken by the authors. We were unable to test the models on the field data, as it was not available to us.

Improvement

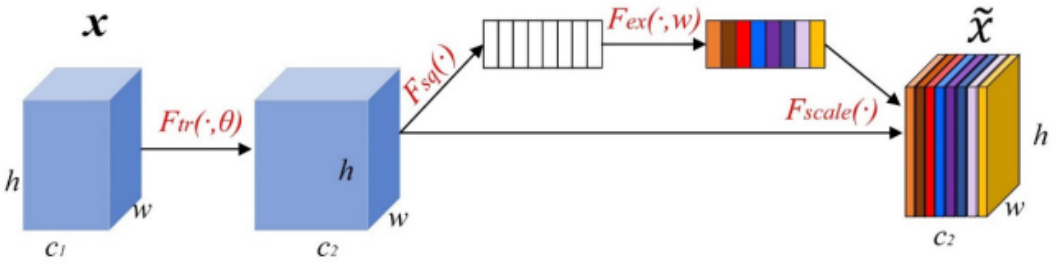
We have used SE-ResNet-50 architecture for improving results better than the previous architectures. Picture below shows how a typical Resnet architecture looks.



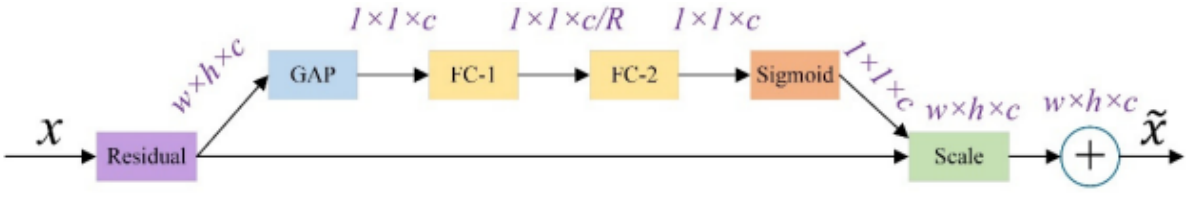
In our new architecture we introduce a new attention module called Squeeze and Excitation Module (SE) at the end of every stage as shown below.



SE module takes the output from residual block and does the global average pooling which is averaging all the values of a specific feature mapping. For example, dimensions are 100 x 50 x 50. We have 100 feature mappings so we get 100 average values corresponding to 100, 50x50 feature mappings. We send this 100 length vector into a FC which squeezes the dimensions to a smaller size and then it passes through another FC which excites the dimensions to original size and then we multiply these values with original feature mapping values. So from the SE module we can learn different weights for different feature mapping which actually improves the performance of our algorithm. Pictures below describe the above process.



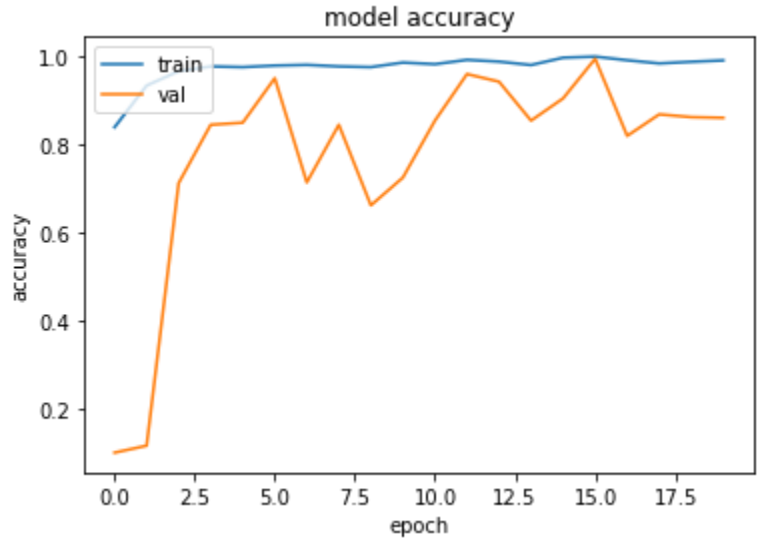
SE Module



SE Module after every stage in Resnet.

We have implemented the SE-Resnet-50 algorithm from scratch by introducing SE modules wherever necessary. We applied this architecture on laboratory dataset only as the field dataset is not available. The accuracy is better than the traditional deep learning architectures.

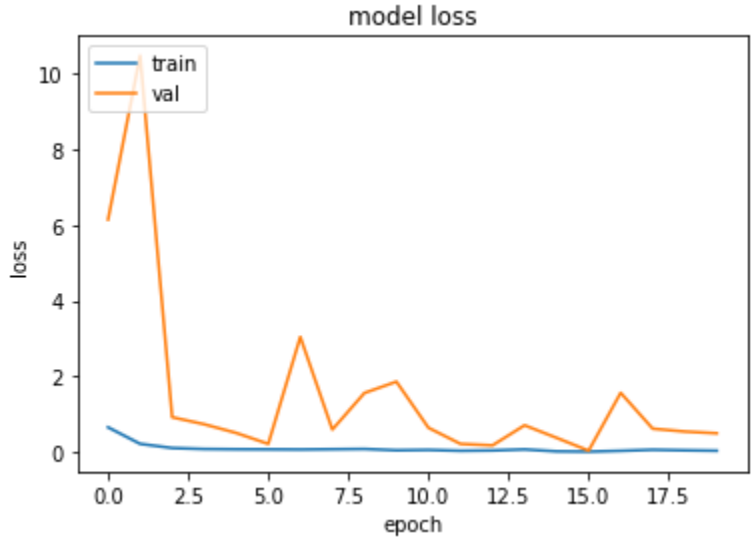
Picture Below shows the plot of accuracies for validation and training data.



The highest validation accuracy : 99.37
Training accuracy : 99.86

We saved the model when it got the highest validation score. And it gave almost 100% accuracy on test data.

Picture Below shows the plot of losses for validation and training data.



Lowest validation loss: 0.0327
Training loss : 0.0073

Code Link: Se-ResNet-50 Tomato Leaf Disease Detection

Contributions

Fine Tuning Parameters + Feature Extraction	- Surya and Shashank
Improvement + Report	- Koteswarudu and Sreekar