DATS 6203 Final Project Report - Group
Cotton Plant Disease Prediction

Nigel Martis, Pranay Bhakthula, Rohan Paul
The George Washington University
6th December 2021

Table of Contents

Introduction:

The goal of our project is to identify Diseased cotton plants when the photo of the plant is given.

Agriculture is one of the major industries and contributes to everyday life of people all around the world. Diseases on plantations is a major problem to all the farmers, which damages the crop and affects their yield, if unnoticed may even affect people who consume those products. Finding diseased plants in the acres of crop is a tedious task and requires a lot of manpower, time and mundane work and is prone to human error.

Our project aims to train a model on images of fresh, diseased cotton plants and develop an interface to identify plants containing disease or not.

Dataset Description:

For our project we used the 'Cotton Disease Dataset'. This dataset was taken from Kaggle. This dataset has three folders with different images for train, validation and test.

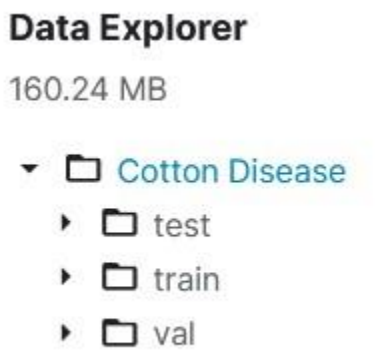

Figure1: main dataset

Inside each of these folders; test, train, val we have following folders:

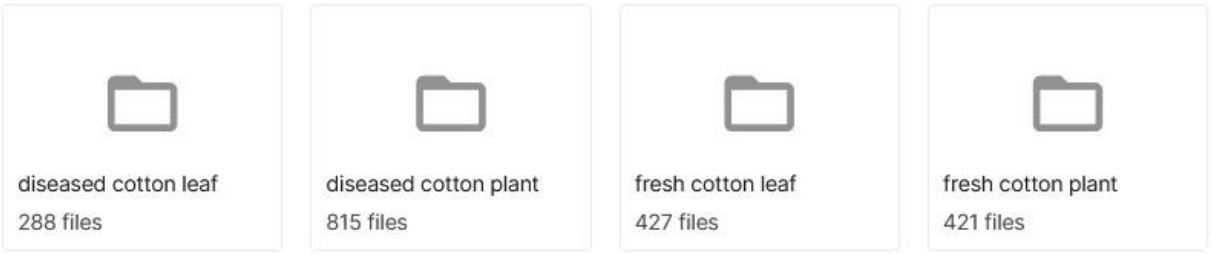

Figure2: sub folders

The Train folder contains 1951 images, Validation folder contains 253 images and test folder contains106 images. This was given as training data for Custom CNN model.

For pretrained models, through image augmentation we produced more images, total of nearly 12,000 images to train pretrained models.

This dataset contains following 4 labels:

Figure3: Diseased leaf

Figure4: Diseased plant

Figure5: Fresh leaf

Figure6: Fresh plant

Deep Learning models:

For the purpose of the project we used Convolution Neural Networks to build a custom CNN model. We also used Pre-trained models like Resnet50, VGG16, Densenet. We trained the build model with 100 epochs and pre-trained models with 30 epochs and selected the model which gave the best accuracy and f1-score. We got the highest accuracy with the resnet50 Network.

Convolution Networks:

Convolution Neural Network -- [S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.]

A convolution network is a multilayer feedforward network that has two or three-dimensional inputs. It has weight functions that are not generally viewed as matrix multiplication operations.

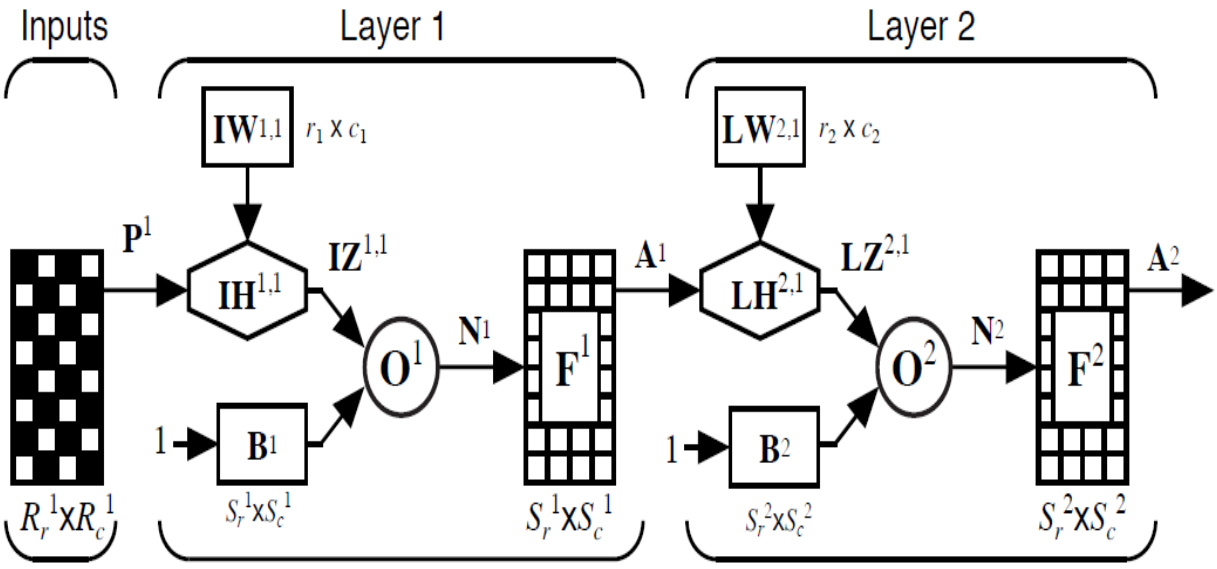

Figure7 : convolution network

Let the input image be represented by the Rr x Rc matrix V. The weight function for this layer performs a convolution kernel that is represented by the r x c matrix W.

$$z_{i,j} = \sum_{k=1}^{r} \sum_{l=1}^{c} w_{k,l} v_{i+k-1,j+l-1}$$

Figure8

In matrix form we write it as:

$$\mathbf{Z} = \mathbf{W} \circledast \mathbf{V}$$

Figure9

Resent50:

Restnet50 -- [Source: arXiv:1512.03385, Authors: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun]

ResNet50 is a variant of ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer. It has 3.8 x 10^9 Floating points operations.

Resnet 50 Architecture:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure10: Resnet 50 Architecture

VGG16:

VGG16 -- [Source: arXiv:1409.1556, Authors: Karen Simonyan, Andrew Zisserman]

VGG16 is a simple and widely used convolution neural network. In this abbreviation 16 means that this model has 16 layers. The VGG16 model achieved 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It made improvements over AlexNet architecture by replacing large kernel-sized filters with multiple 3x kernel-sized filters one after another. VGG16 was trained for weeks using NVIDIA Titan Black GPU's

VGG Architecture:

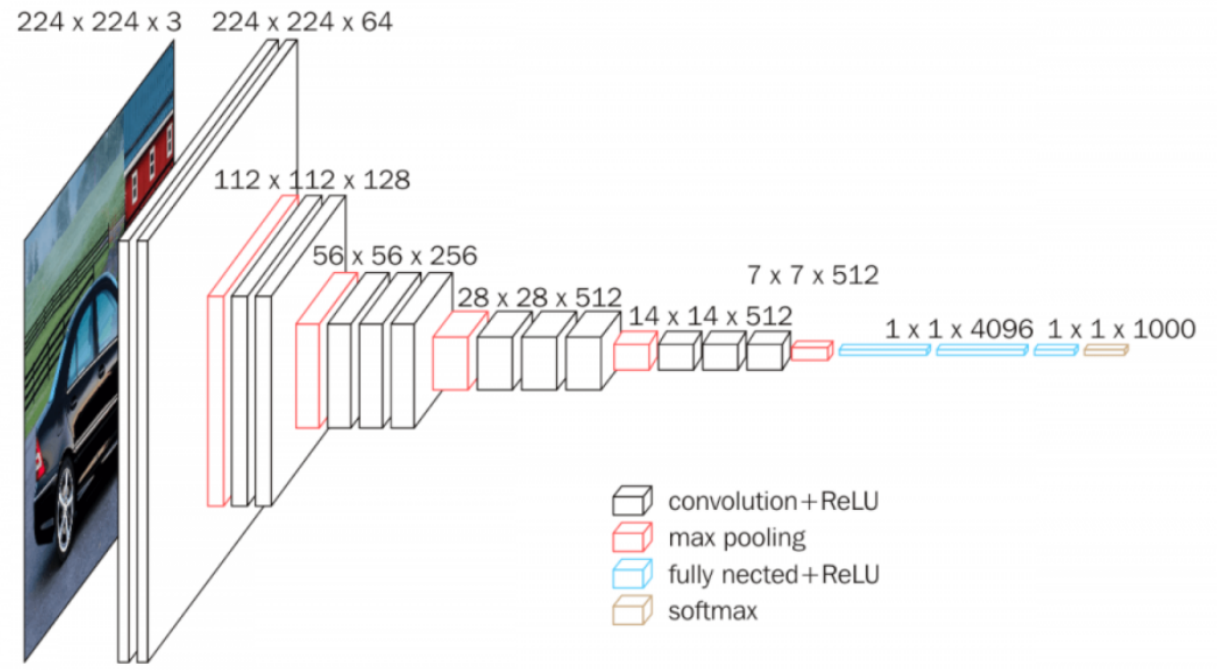

Figure11: VGG Architecture

Densesnet:

Densenet -- [Source : **arXiv:1608.06993,** Authors: Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger]

A DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect *all layers* (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers.

Densenet Architecture:

In a DenseNet architecture, each layer is connected to every other layer, hence the name Densely Connected Convolutional Network. For L layers, there are L(L+1)/2 direct connections. For each layer, the feature maps of all the preceding layers are used as inputs, and its own feature maps are used as input for each subsequent layers.

This is really it, as simple as this may sound, DenseNets essentially connect every layer to every other layer. This is the main idea that is extremely powerful. The input of a layer inside DenseNet is the concatenation of feature maps from previous layers.

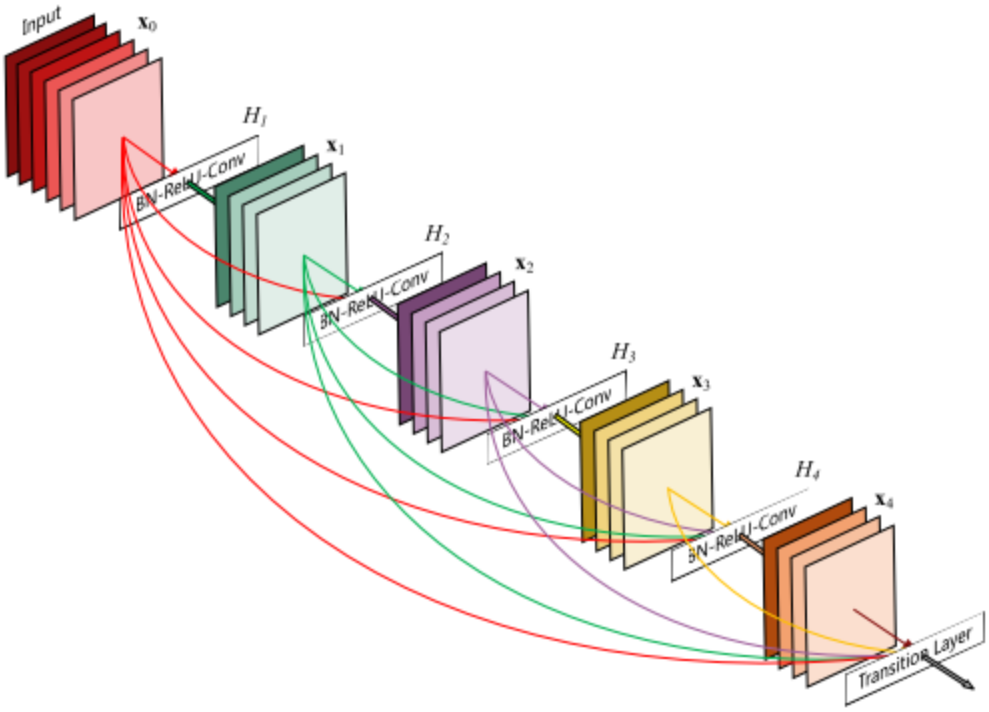

Figure12 : Densenet Architecture

Model Metrics:

We looked at the following metrics in deciding how effective our models were:

F1-score

F1-score weights precision and recall [blog.floydhub.com].

$$F1 - Score = 2 * \frac{Recall * Precision}{Recall + Precision}$$

Figure12

Accuracy

Accuracy is simply what percentage of observations were classified correctly [blog.floydhub.com].

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

Figure13

Experimental Setup:

Transformations:

The following transformations were used on input images of custom CNN model.

Rescale - we rescaled the input values with 1./255
Rotation_range - random rotation of 40 was used on the input images.
Width_shift_range - width shift range of 0.2 was used.
Height_shift_range - height shift range of 0.2 was used.
Shear_range - shear range of 0.2 was used
Zoom_range - zoom range of 0.2 was used
Horizontal flip - randomly horizontal flip was used
fill_mode - 'nearest' fill mode was used

The following transformations were used on input images of pre-trained models
Adjust_gamma -

1. we used adjust_gamma function to brighten the image by gamma=0.5
2. we used adjust_gamma function to darken the image by gamma=2

Flip left or right - we used np.fliplr() to flip the images to the left or to the right.

Flip up or down - we used np.flipud() to flip the images to up or down

Random noise - random_noise was used to induce random noise into the input images.

Hyperparameters:

Learning Rate - Learning rate of 0.001 was used. It gave good results so we continued with it.

Batch Size - Batch size of 32 was used in the built in model. Batch size of 64 was used in all the pretrained models.

Optimizer - Adam was used as the optimizer

Epochs - The built in model used 100 epochs while the pre-trained models used 30 epochs.

Dropout - For the custom model we used a dropout of 0.5.

Results:

The performance of the model was judged based on the accuracy and loss values for train, validation and test sets. The loss on train and validation sets for 30 epochs is given below for various models.

Densenet:

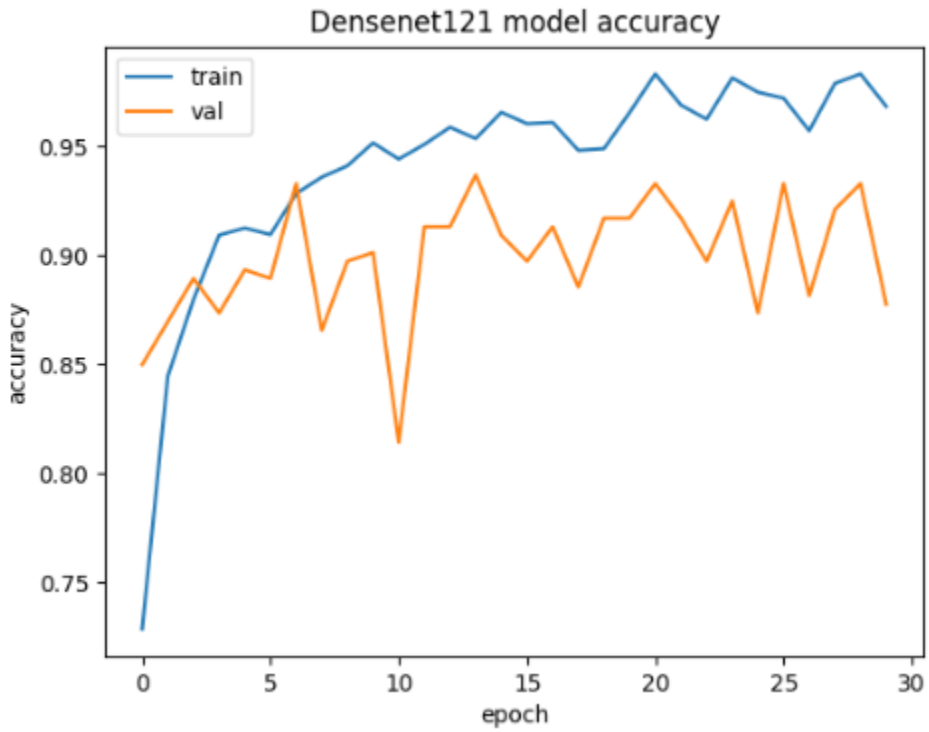

Figure14

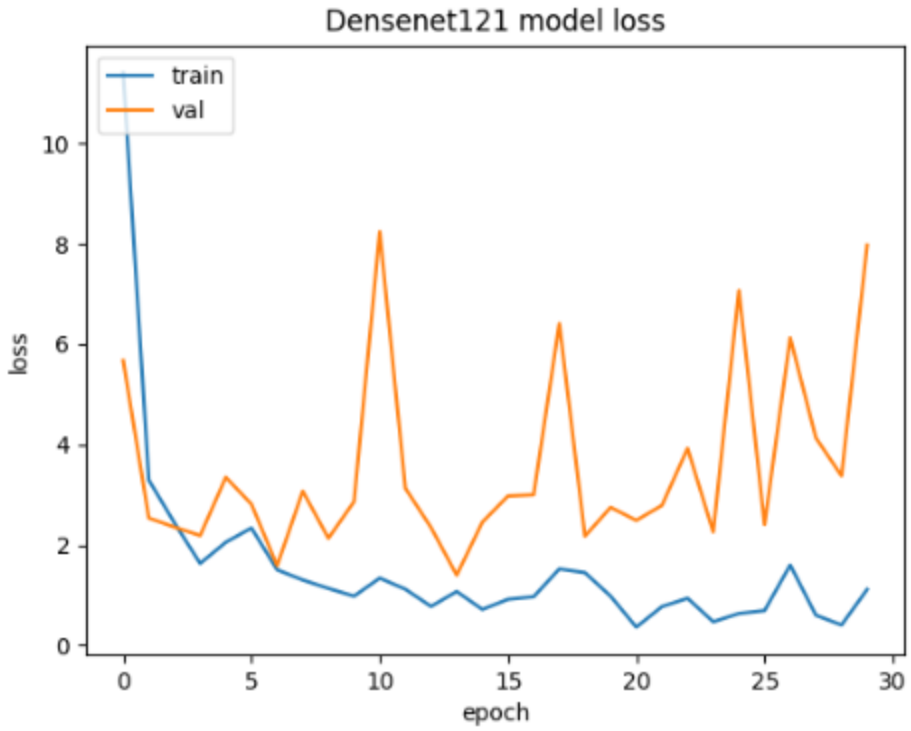

Figure15

```
Accuracy:0.932806324110672
F1 Score:0.932806324110672
Confusion Matrix:
[[40  0  3  0]
 [ 2 72  0  4]
 [ 3  1 62  0]
 [ 1  3  0 62]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.93      0.90        43
           1       0.95      0.92      0.94        78
           2       0.95      0.94      0.95        66
           3       0.94      0.94      0.94        66

    accuracy                           0.93       253
   macro avg       0.93      0.93      0.93       253
weighted avg       0.93      0.93      0.93       253
```

Figure16

```
=================================
Total params: 7,238,212
Trainable params: 200,708
Non-trainable params: 7,037,504
_________________________________
```

Figure17

VGG16:

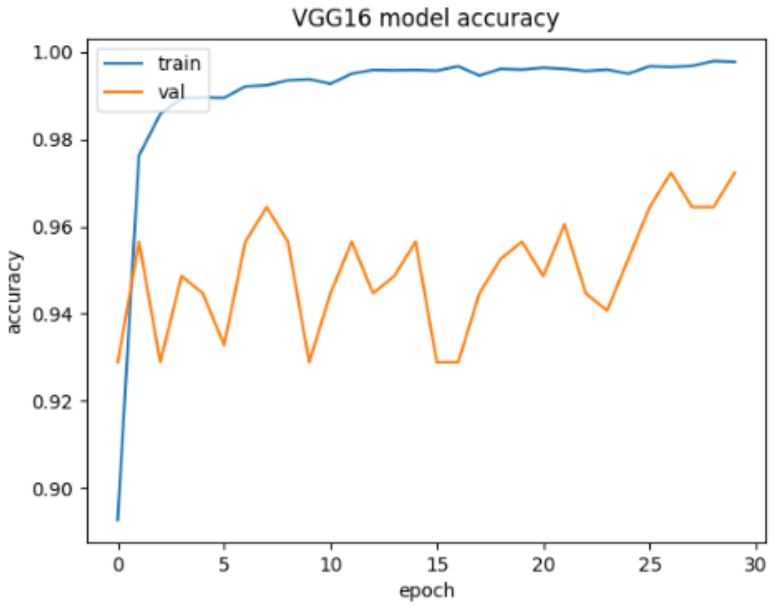

Figure18

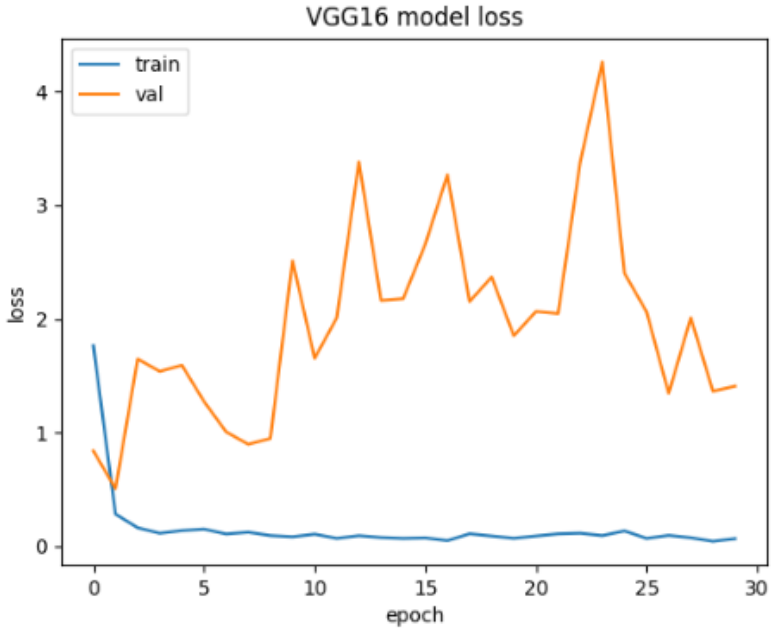

Figure19

```
Accuracy:0.932806324110672
F1 Score:0.932806324110672
Confusion Matrix:
[[43  0  0  0]
 [ 0 74  0  4]
 [ 5  1 58  2]
 [ 0  5  0 61]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        43
           1       0.93      0.95      0.94        78
           2       1.00      0.88      0.94        66
           3       0.91      0.92      0.92        66

    accuracy                           0.93       253
   macro avg       0.93      0.94      0.93       253
weighted avg       0.94      0.93      0.93       253
```

Figure20

```
==================================
Total params: 14,815,044
Trainable params: 100,356
Non-trainable params: 14,714,688
----------------------------------
```

Figure21

Resnet50:

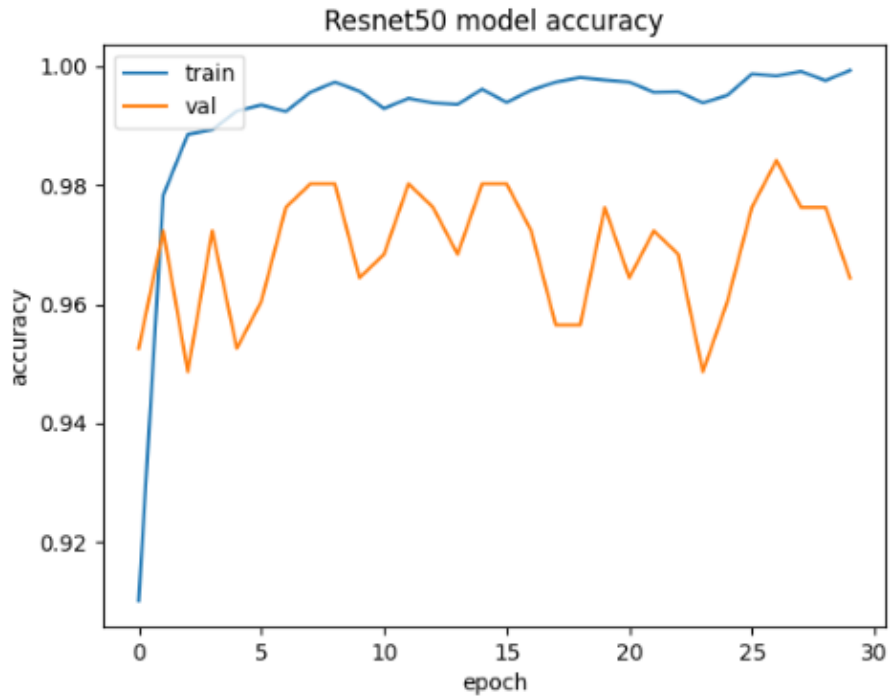

Figure22

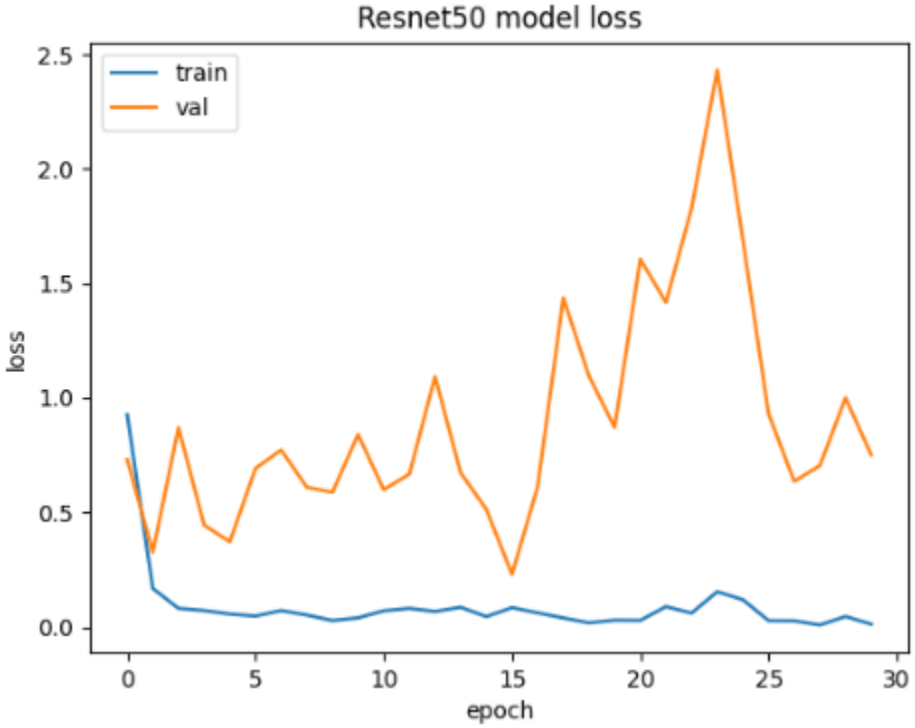

Figure23

```
Accuracy:0.9644268774703557
F1 Score:0.9644268774703557
Confusion Matrix:
[[42  1  0  0]
 [ 0 75  1  2]
 [ 0  3 63  0]
 [ 1  1  0 64]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        43
           1       0.94      0.96      0.95        78
           2       0.98      0.95      0.97        66
           3       0.97      0.97      0.97        66

    accuracy                           0.96       253
   macro avg       0.97      0.97      0.97       253
weighted avg       0.96      0.96      0.96       253
```

Figure24

```
=================================
Total params: 23,989,124
Trainable params: 401,412
Non-trainable params: 23,587,712
_________________________________
```

Figure25

Custom CNN model:

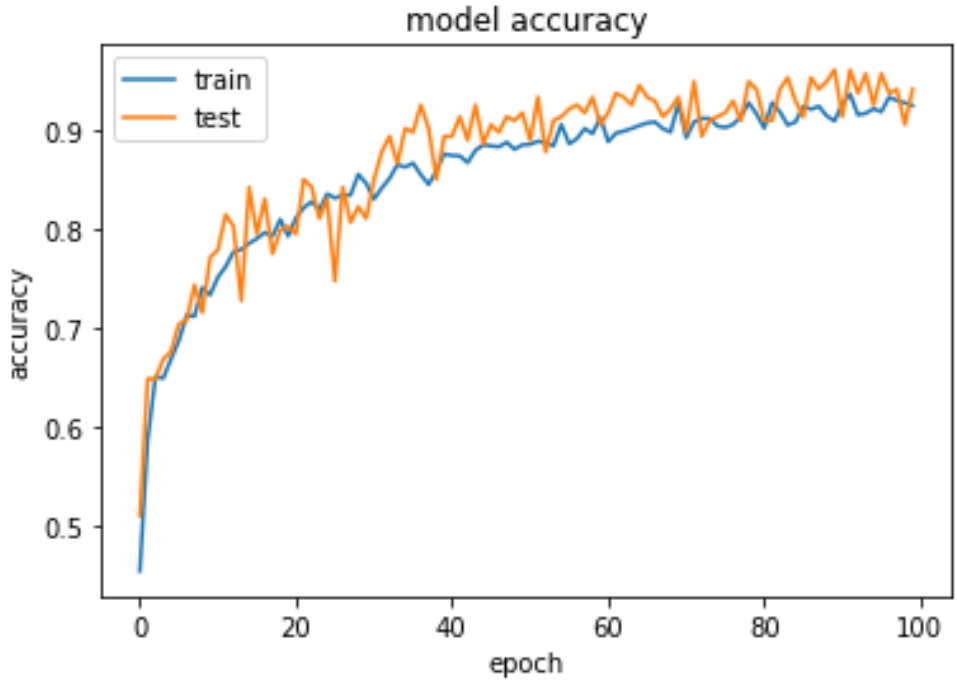

Figure26

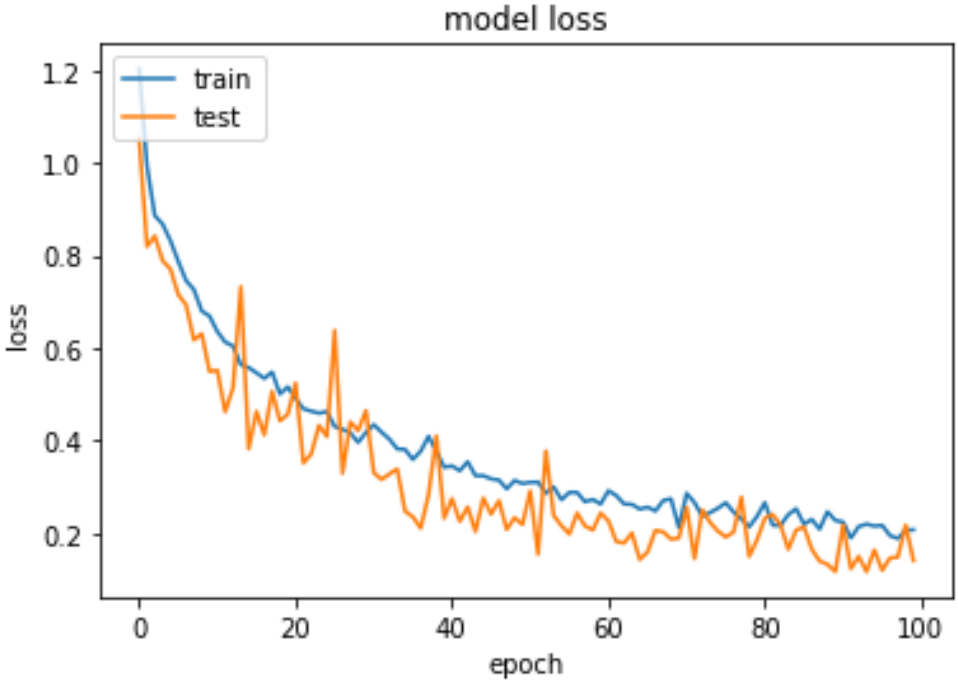

Figure27

```
In [29]: print("Accuracy:" + str(accuracy_score(val_actual, val_preds)))
Accuracy:0.9683794466403162

In [30]: print("F1 Score:" + str(f1_score(val_actual, val_preds, average='micro')))
F1 Score:0.9683794466403162

In [31]: print("Confusion Matrix:\n" + str(confusion_matrix(val_actual, val_preds)))
Confusion Matrix:
[[42  0  1  0]
 [ 0 75  1  2]
 [ 2  0 64  0]
 [ 1  1  0 64]]

In [32]: print("Classification Report:\n" + str(classification_report(val_actual, val_preds)))
Classification Report:
              precision    recall  f1-score   support

         0.0       0.93      0.98      0.95        43
         1.0       0.99      0.96      0.97        78
         2.0       0.97      0.97      0.97        66
         3.0       0.97      0.97      0.97        66

    accuracy                           0.97       253
   macro avg       0.96      0.97      0.97       253
weighted avg       0.97      0.97      0.97       253
```

Figure28

Model summary:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 298, 298, 32)	896
max_pooling2d (MaxPooling2D)	(None, 149, 149, 32)	0
conv2d_1 (Conv2D)	(None, 147, 147, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 73, 73, 64)	0
conv2d_2 (Conv2D)	(None, 71, 71, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 35, 35, 128)	0
conv2d_3 (Conv2D)	(None, 33, 33, 256)	295168
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 256)	0
dropout (Dropout)	(None, 16, 16, 256)	0

flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 128)	8388736
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33024
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 4)	1028

Total params: 8,811,204
Trainable params: 8,811,204
Non-trainable params: 0

Summary:

For Densenet50:

1. Accuracy of the model = 93.28%.
2. Overall f1-score of the model = 93.28%
3. From the classification report of models:
    1. F1 score for 0's = 0.90, f1 score for 1's = 0.94, f1 score for 2's = 0.95 and f1 score for 3's = 0.94.
    2. Precision for 0's = 0.87, Precision for 1's = 0.95, Precision for 2's = 0.95 and Precision for 3's = 0.94.
    3. Recall for 0's = 0.93, Recall for 1's = 0.92, Recall for 2's = 0.94 and Recall for 3's = 0.94.
4. From the confusion matrix we can observe which class is getting correctly predicted.

We can observe that f1-scores, precision and recall rate is high for 2's in this model.

This model has good accuracy, so we use this model for further predictions of the cotton plants images.

For VGG16:

1. Accuracy of the model = 93.28%.

2. Overall f1-score of the model = 93.28%

3. From the classification report of models:

    1. F1 score for 0's = 0.95, f1 score for 1's = 0.94, f1 score for 2's = 0.94 and f1 score for 3's = 0.92.

    2. Precision for 0's = 0.90, Precision for 1's = 0.93, Precision for 2's = 1.00 and Precision for 3's = 0.91.

    3. Recall for 0's = 1.00, Recall for 1's = 0.95, Recall for 2's = 0.88 and Recall for 3's = 0.92.

4. From the confusion matrix we can observe which class is getting correctly predicted.

We can observe that f1-scores, precision and recall rate is high for 1's in this model.

This model has good accuracy, so we use this model for further predictions of the cotton plants images.

For Resnet50:

1. Accuracy of the model = 96.44%.

2. Overall f1-score of the model = 96.44%

3. From the classification report of models:

    1. F1 score for 0's = 0.98, f1 score for 1's = 0.95, f1 score for 2's = 0.97 and f1 score for 3's = 0.97.

    2. Precision for 0's = 0.98, Precision for 1's = 0.96, Precision for 2's = 0.95 and Precision for 3's = 0.97.

    3. Recall for 0's = 0.98, Recall for 1's = 0.96, Recall for 2's = 0.95 and Recall for 3's = 0.97.

4. From the confusion matrix we can observe which class is getting correctly predicted.

This model has almost similar accuracy and f1 score as the custom model but also this is achieved by 30 epochs.

For Custom CNN model:

1. Accuracy of the model = 96.83%.

2. Overall f1-score of the model = 96.83%

3. From the classification report of models:

    1. F1 score for 0's = 0.95, f1 score for 1's = 0.97, f1 score for 2's = 0.97 and f1 score for 3's = 0.97.

    2. Precision for 0's = 0.93, Precision for 1's = 0.99, Precision for 2's = 0.97 and Precision for 3's = 0.97.

    3. Recall for 0's = 0.98, Recall for 1's = 0.96, Recall for 2's = 0.97 and Recall for 3's = 0.97.

4. From the confusion matrix we can observe which class is getting correctly predicted.

5. The total trainable parameters = 8,811,204

This model has the highest accuracy and f1 score than all the models but also this is achieved by 100 epochs.

Application -

We created an app where you can upload photos of cotton plants and show whether the plant is diseased or Fresh.

This application predicts the values based on the best model of the 4 we created above. In this case it is the Custom CNN Model.

The results of the Application are shown below.

Figure29

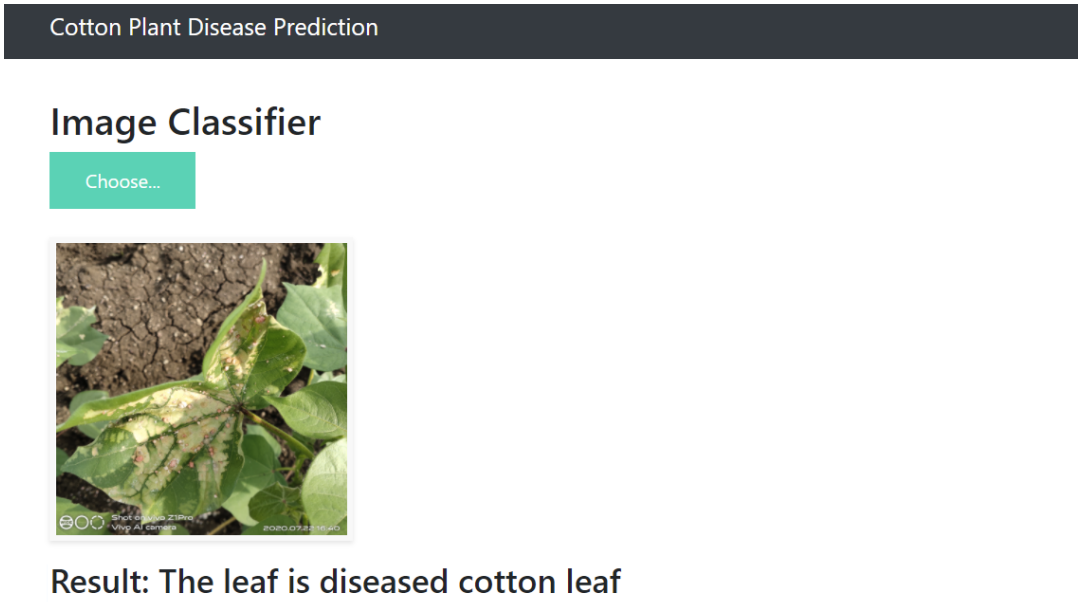

Figure30

Conclusions:

From the results we can see that the CNN model that was built performs better than the pretrained models. This can be seen by comparing the accuracy and loss functions of all the models. The manually built CNN model has 100 epochs where the closest model to that which is Resnet50 has almost the same accuracy with 30 epochs. The average training time for each of these models is approximately 1 hour.

Further work will be around interpretability of the model where one can see the features of the plant/leaf that the model thinks is important.

References:

1. https://github.com/krishnaik06/Cotton-Disease-Prediction-Deep-Learning
2. https://www.tensorflow.org/tutorials/images/data_augmentation
3. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
4. https://github.com/krishnaik06/Deployment-Deep-Learning-Model
5. https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50
6. https://www.tensorflow.org/api_docs/python/tf/keras/applications/densenet/DenseNet121
7. https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16/VGG16

8. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator