Smart Agriculture using Image Processing

MINOR PROJECT I

Submitted By:

Apurav Sharma [9917103079]

Sawan Kumar [9917103098]

Krishna Bharadwaj [9917103134]

Under the supervision of

Dr. Ashish Tripathi

Department of CSE/IT
Jaypee Institute of Information Technology University, Noida

November 2019

ACKNOWLEDGEMENT

I would like to place on record my deep sense of gratitude to Dr. Ashish Kumar Tripathi, Assistant Professor (Senior Grade), Jaypee Institute of Information Technology, India for his generous guidance, help and useful suggestions.

I express my sincere gratitude to Computer Science and Engineering Dept. of Jaypee Institute of Information Technology, India, for his/her stimulating guidance, continuous encouragement and supervision throughout the course of present work.

I also wish to extend my thanks to Vaibhav Dwivedi, Milind Kumar and other classmates for their insightful comments and constructive suggestions to improve the quality of this project work.

Signature(s) of Students

Apurav Sharma (9917103079)

Sawan Kumar (9917103098)

Krishna Bharadwaj (9917103134)

Abstract:

Agriculture is essential to the continued existence of human life as they directly depend on it for the production of food. The exponential rise in population calls for a rapid increase in food with the application of technology to reduce the laborious work, maximize production and reduce the impact of pressure on the environment. Precision Agriculture is thought to be solution required to achieve the production rate required. There has been a significant improvement in the area of image processing and data processing which has being a major challenge previously in the practice of precision Agriculture. Over the years, several new challenges have emerged before the sector. With fragmentation of agricultural holdings and depletion of water resources, the adoption of a combination of resource efficient methods, dynamic cropping patterns, farming that is responsive to climate change and intensive use of information and communication technology should be the backbone of smallholder farming in India. For a safe and food secure future, the agriculture landscape has to undergo tremendous transformation and shift from the philosophy of green revolution led productivity to green methods led sustainability in agriculture. In smallholder firms, resource efficiency can be brought about through adoption of appropriate technologies. However, use of technology, investment in costly farm machinery or scaling up the existing technology may not economically feasible for small and marginal farmers. Hence, there is need to promote use of environment-friendly high-tech machinery (like robots, drones etc) for the mechanization of small and marginal farm holdings. Robots are quite smart by design. However, to fully understand and properly navigate a task like identification, spraying, cutting and dis-rooting of weed plants from crops, they need inputs. These inputs can be provided using artificial intelligence. The main area of application of robots in agriculture today is at the harvesting stage. Emerging applications of robots or drones in agriculture include weed control, cloud seeding, planting seeds, harvesting, environmental monitoring and soil analysis. Disease identification, Plant leaves classification, Seed classification, Pollen leaves classification, Increase Crop Productivity are some of the main areas where image processing can be used. The project aims to automate the identification of weed plants among crops in agriculture using artificial intelligence by processing captured images and has the potential for implementation in areas mentioned above.

Table of Contents

List of Figures

1. Introduction

Can you differentiate a weed from a crop plant in an agricultural field? The ability to do so effectively can mean better crop yields and better stewardship of the environment. Food is an essential necessity of human life and requires its continuous supply and production to cater the needs of the growing population through sustainable agriculture. This can be achieved with the application of emerging technologies in the sector to maximize production across a vegetation. Technology can aid/improve agriculture in several ways through pre-planning and post-harvest by the use of deep learning techniques through image processing to determine the soil nutrient composition, right amount, right time, right place application of farm input resources like fertilizers, herbicides, water, weed detection, early detection of pest and diseases etc.

Our project is based on a data-set we obtained from Hare Krishna Farms containing two classes of plants – one of them is tomato and the other one is a weed. The data set contains two folders test set and training set. Test set contains 144 images and training set contains 460 images of each plant. The images are in colored format also called RGB format with three channels Red, Blue and Green.

Now a days, whole world is facing various problems like global warming, biodiversity loss, effects of fast urban development, and various environmental damages. Hence there is an urgent need to apply deep learning techniques to obtain the botanical knowledge & various features of plant and make this information accessible and useful to different kinds of people like researchers, farmers, botanists, and students. Hence plant identification is the first and important task.

There are many plant organs like leaves, flowers, fruits, seeds which can be used for plant identification. In this project leaves are selected to obtain the features of plant. Because leaf can be easily obtained and scanned and also it consists of more excluding information which is useful for plant classification. These leaf images are sent to computer and then by using image processing tools, leaf features are extracted to identify the plant.

2. Background study

Generally speaking, CNNs attempt to learn the relationship between the input and the output and store the learned experience in their filter weights. One challenge to understand CNNs is the role played by the nonlinear activation unit after the convolution operation. The pooling operation mainly provides a spatial dimension reduction technique and its role is not as critical.

In the recent literature, three activation functions are commonly used by CNNs. They are the sigmoid function, the rectified linear unit (ReLU) and the parameterized ReLU (PReLU). The PReLU is also

known as the leaky ReLU. All of them play a clipping-like operation. The sigmoid clips the input into an interval between 0 and 1. The ReLU clips negative values to zero while keeping positive values unchanged. The leaky ReLU has a role similar to the ReLU but it maps larger negative values to smaller ones by reducing the slope of the mapping function. It is observed experimentally that, if the nonlinear operation is removed, the system performance drops by a large margin [1].

When we applied our model on the data set by making a sequential model of Convolution-layers and Max-Pool layers and attained some accuracy on cross-validated data set, the only way to improve the accuracy without changing the model architecture or without using any bigger Network (that may increase accuracy) was by exploring how learning rate affects model training.

Since, Adam learns the fastest and is more stable than the other optimizers, it doesn't suffer any major decreases in accuracy. [2]

In spite of all the data availability, fetching the right type of data which matches the exact use-case of our experiment is a daunting task. Moreover, the data has to have good diversity as the object of interest needs to be present in varying sizes, lighting conditions and poses if we desire that our network generalizes well during the testing phase. To overcome this problem of limited quantity and limited diversity of data, we generate(manufacture) our own data with the existing data which we have [3]. To state a few of the frameworks, Keras has ImageDataGenerator (needs least amount of work from us), Tensor flow has TFLearn's Data Augmentation and MXNet has Augmenter classes.

3. Requirement Analysis

3.1 Product Perspective

The purpose of this project work is to provide a proper methodology and efficient algorithms for better crop yields and better stewardship of the environment . In this world of evolving machine learning and deep learning techniques it is becoming increasingly important for us to understand and implement in the correct fashion the proper software for the identification purpose. A need for the increase in a low cost and low overhead system is very much in demand for the reduction of the net customer costs and survival in the market of intense competitions. So we aim at developing algorithms which will prove to be more and more fruitful for this purpose and will be of immense applicability in agricultural fields. The employment of newer concepts and algorithms will prove to be a much better headway in this field.

3.2 User Characteristics

The users who are developing this whole application or are trying to make an evaluation of the proper working of the newest ideas and the algorithms implemented need to be proficient in the following areas

- Deep learning including all concepts of Convolutional Neural Networks.
- Data augmentation techniques in different frameworks such as keras, tensor flow, Mxnet.
- Python programming with important data structures and libraries.

3.3 Specific Requirements

The whole of this project work will be developed using the Jupyter Notebook in anaconda environment which uses the Python language for the purpose of the development of the network and the keras with tensor flow in back-end for the module definitions. So the specific requirements for this project work are :

- Python version 3.6.5
- Jupyter Notebook
- Conda version 4.6.14
- Linux or any compatible Operating Systems.

As an end product the algorithm developed by us will be implemented in mobile or other embedded devices. In that scenario the whole internal working will be abstracted from the actual users of the product and it is of no importance for them to understand the specific requirements of the programs executing inside their devices.

3.4 Performance Requirements

The algorithms developed will be implemented on embedded devices as an end product. So it is not our concern about the performance of the embedded devices which is rather a concern of the company who will be implementing this technology in their products. For the purpose of development the only performance requirements are related to the specific software requirements mentioned above and a system which is capable of smoothly running these above said software is good enough for the project members to work on the proposed algorithms.

3.5 Design Constraints

The data-set is used as a visual aid only and can never bring in front of our eyes the actual scenario where millions of nodes exist in a single Network. So, we must have this thing in mind that there exists a close approximation between the actual experiments developed and the practical measurements.

4. Detailed design

The main design procedure in this project work of ours has been carried out with Keras. Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it

offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

- This makes Keras easy to learn and easy to use. This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular Tensor Flow), it enables us to implement anything we could have built in the base language.

- We are already constantly interacting with features built with Keras -- it is in use at Netflix, Uber and many others. It is especially popular among startups that place deep learning at the core of their products.

4.1 Approach

First, the Keras package was needed to be studied. After that, the models were developed as a slow and gradual process. The ImageDataGenerator [3] was studied and the model for this was executed in Keras. The outcomes from these tests have been compared with the published results and theoretical outcomes.

4.2 Tensor flow Backend

Keras is a model-level library, providing high-level building blocks for developing deep learning models. It does not handle low-level operations such as tensor products, convolutions and so on itself. Instead, it relies on a specialized, well optimized tensor manipulation library to do so, serving as the "backend engine" of Keras. Rather than picking one single tensor library and making the implementation of Keras tied to that library, Keras handles the problem in a modular way, and several different backend engines can be plugged seamlessly into Keras.

At this time, Keras has three backend implementations available: the TensorFlow backend, the Theano backend, and the CNTK backend.

- TensorFlow is an open-source symbolic tensor manipulation framework developed by Google.

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 on November 9, 2015.

4.3 Model Architecture

The first step in image processing is to understand, how to represent an image so that the machine

can read it? In simple terms, every image is an arrangement of dots (a pixel) arranged in a special order. If we change the order or color of a pixel, the image would change as well. The machine will basically break this image into a matrix of pixels and store the color code for each pixel at the representative location.

We need three basic components to define a basic convolutional network.

- The convolutional layer
- The Pooling layer[optional]
- The output layer

4.3.1 The Convolution Layer

Suppose we have an image of size 6*6. We define a weight matrix which extracts certain features from the images.

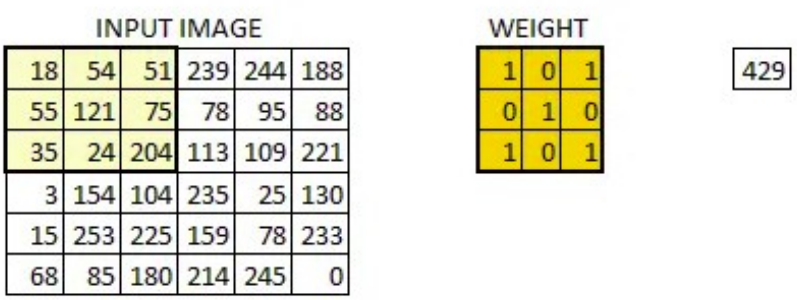

Fig 4.1

We have initialized the weight as a 3*3 matrix. This weight shall now run across the image such that all the pixels are covered at least once, to give a convoluted output. The value 429 above, is obtained by the adding the values obtained by element wise multiplication of the weight matrix and the highlighted 3*3 part of the input image. The 6*6 image is now converted into a 4*4 image. Think of weight matrix like a paint brush painting a wall. The brush first paints the wall horizontally and then comes down and paints the next row horizontally. Pixel values are used again when the weight matrix moves along the image. This basically enables parameter sharing in a convolutional neural network.

The weight matrix behaves like a filter in an image extracting particular information from the original image matrix. A weight combination might be extracting edges, while another one might a particular color, while another one might just blur the unwanted noise.

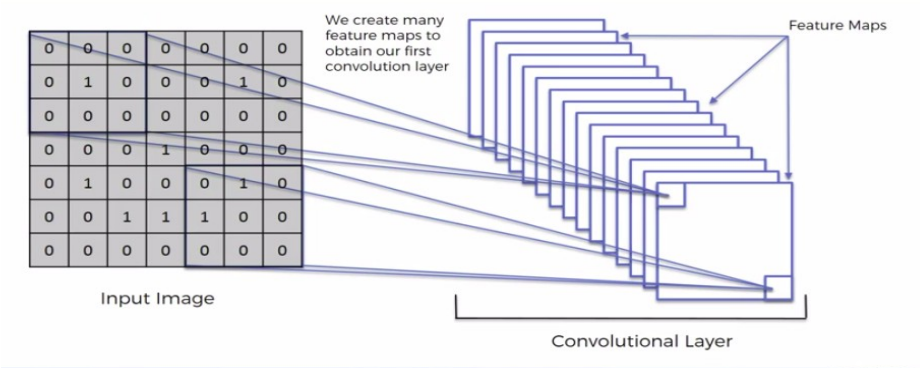

Fig 4.2

The weights are learnt such that the loss function is minimized similar to a (MLP) Multi layer Perceptron. We are using relu activation function. Therefore weights are learnt to extract features from the original image which help the network in correct prediction. When we have multiple convolution layers, the initial layer extract more generic features, while as the network gets deeper, the features extracted by the weight matrices are more and more complex and more suited to the problem at hand.

4.3.2 Pooling layer

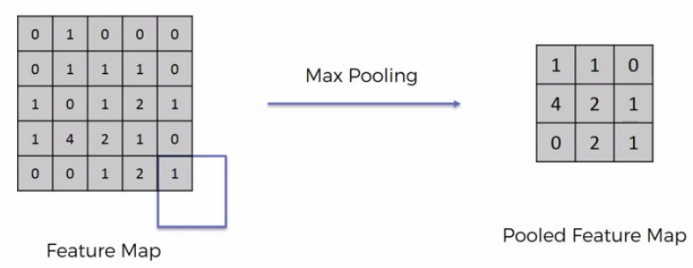

Fig 4.3

Sometimes when the images are too large, we would need to reduce the number of trainable parameters. It is then desired to periodically introduce pooling layers between subsequent convolution layers. Pooling is done for the sole purpose of reducing the spatial size of the image. Pooling is done independently on each depth dimension, therefore the depth of the image remains unchanged. The most common form of pooling layer generally applied is the max pooling.

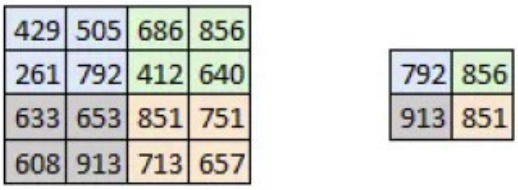

Fig 4.4

Here we have taken stride as 2, while pooling size also as 2. The max operation is applied to each depth dimension of the convoluted output. As you can see, the 4*4 convoluted output has become 2*2 after the max pooling operation.

Output dimensions

It might be getting a little confusing for you to understand the input and output dimensions at the end of each convolution layer. I decided to take these few lines to make you capable of identifying the output dimensions. Three hyper parameter would control the size of output volume.

- **The number of filters** – the depth of the output volume will be equal to the number of filter applied. The depth of the activation map will be equal to the number of filters.

- **Stride** – When we have a stride of one we move across and down a single pixel. With higher stride values, we move large number of pixels at a time and hence produce smaller output volumes.

- **Zero padding** – This helps us to preserve the size of the input image. If a single zero padding is added, a single stride filter movement would retain the size of the original image.

We can apply a simple formula to calculate the output dimensions. The spatial size of the output image can be calculated as **([W-F+2P]/S)+1.** Here, W is the input volume size, F is the size of the filter, P is the number of padding applied and S is the number of strides. Suppose we have an input image of size 32*32*3, we apply 10 filters of size 3*3*3, with single stride and no zero padding.

Here W=32, F=3, P=0 and S=1. The output depth will be equal to the number of filters applied i.e. 10. The size of the output volume will be ([32-3+0]/1)+1 = 30. Therefore the output volume will be 30*30*10.

4.3.3 Flattening

In between the pooling layer and the fully connected layer, there is a 'Flatten' layer. Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.

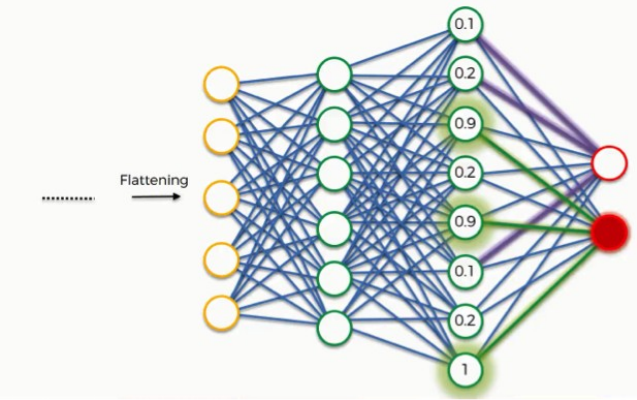

Fig 4.5

4.3.4 Full Connection

Here's where artificial neural networks and convolutional neural networks collide as we add the former to our latter. It's here that the process of creating a convolutional neural network begins to take a more complex and sophisticated turn.

As you see from the image below, we have three layers in the full connection step:

- Input layer
- Fully-connected layer
- Output layer

The input layer contains the vector of data that was created in the flattening step. The features that we distilled throughout the previous steps are encoded in this vector.

At this point, they are already sufficient for a fair degree of accuracy in recognizing classes. We now want to take it to the next level in terms of complexity and precision.

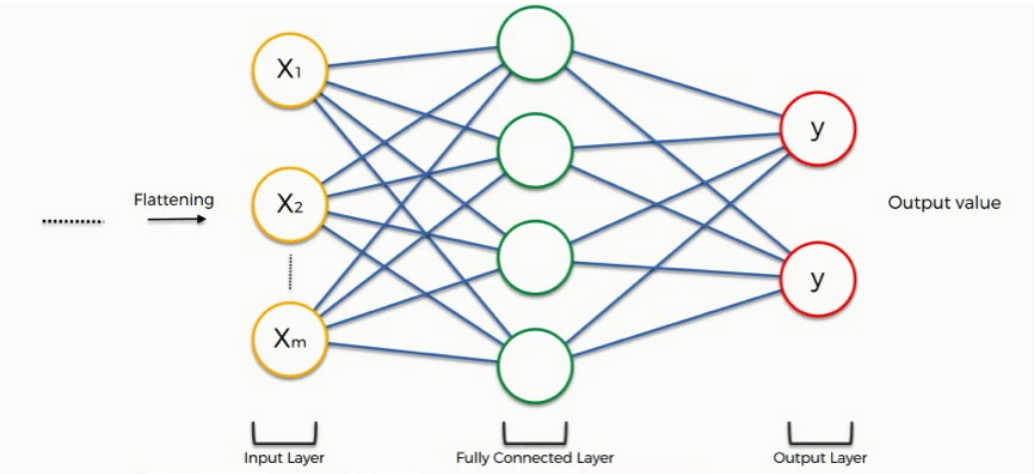

Fig 4.6

4.3.5 The Output layer

After multiple layers of convolution and padding, we would need the output in the form of a class. The convolution and pooling layers would only be able to extract features and reduce the number of parameters from the original images. However, to generate the final output we need to apply a fully connected layer to generate an output equal to the number of classes we need. It becomes tough to reach that number with just the convolution layers. Convolution layers generate 3D activation maps while we just need the output as whether or not an image belongs to a particular class. The output layer has a loss function like categorical cross-entropy, to compute the error in prediction. Once the forward pass is complete the back propagation begins to update the weight and biases for error and loss reduction.

5. Implementation

Initialization of CNN model is the first thing to do after importing important packages. There are basically two ways to initialize a network: Using Graph or by using Layers. We use the latter one and create an object of sequential class.

Step 1 - Convolution layer to detect features (relationships of a pixel with its neighbors) in images
here we create 32 feature detectors of 3*3 dimension. Perform batch normalization[4].

Step 2 - Pooling to obtain pooled/smaller feature map and dropout of 30%[5].

Step 3 - Repeat step 1 and 2.

Step 4 - Flattening converts pooled feature maps to a vector/array by storing cells of these maps

Step 5 – Perform batch normalization and dropout again and full connection to add hidden layer

Step 6 - Compiling the CNN

Step 7 - Fitting the CNN to the images

6. Experimental Results and Analysis

- For Data set obtained from Hare Krishna Farms, the model is able to achieve an accuracy upto 99.54% in 5 epochs only as shown in the figure below.
- This is greater than the recognition rate in [6], [7] but less than that of [8].

```
Found 920 images belonging to 2 classes.
Found 288 images belonging to 2 classes.
Epoch 1/5
960/960 [==============================] - 361s 376ms/step - loss: 0.1635 - acc: 0.9370 - val_loss: 0.0940 - val_ac
c: 0.9620
Epoch 2/5
960/960 [==============================] - 351s 366ms/step - loss: 0.0394 - acc: 0.9867 - val_loss: 0.1856 - val_ac
c: 0.9345
Epoch 3/5
960/960 [==============================] - 353s 368ms/step - loss: 0.0205 - acc: 0.9931 - val_loss: 0.0736 - val_ac
c: 0.9788
Epoch 4/5
960/960 [==============================] - 355s 370ms/step - loss: 0.0129 - acc: 0.9953 - val_loss: 0.1181 - val_ac
c: 0.9686
Epoch 5/5
960/960 [==============================] - 353s 368ms/step - loss: 0.0132 - acc: 0.9954 - val_loss: 0.0981 - val_ac
c: 0.9618
: <keras.callbacks.History at 0x7f7456bc9d68>
```

Fig 6.1

- where acc is the accuracy of a batch of training data and val_acc is the accuracy of a batch of testing data.

```
10/10 [==============================] - 4s 410ms/step - loss: 0.6265 - accuracy: 0.9688 - val_loss: 3.7431 - val_accura
cy: 0.4500
Epoch 24/25
10/10 [==============================] - 4s 412ms/step - loss: 0.8764 - accuracy: 0.9688 - val_loss: 4.9936 - val_accura
cy: 0.4500
Epoch 25/25
10/10 [==============================] - 4s 410ms/step - loss: 0.5257 - accuracy: 0.9781 - val_loss: 4.6652 - val_accura
cy: 0.4500
```

Fig 6.2 No dropout

```
10/10 [==============================] - 34s 3s/step - loss: 0.4401 - accuracy: 0.9531 - val_loss: 25.2797 - val_accurac
y: 0.5500
Epoch 25/25
10/10 [==============================] - 34s 3s/step - loss: 0.7189 - accuracy: 0.9469 - val_loss: 23.5857 - val_accurac
y: 0.5500
```

Fig 6.3 With dropout

When we introduced a dropout of rate 30% the accuracy improved from 94.69 to 97.81% which means the dropout is helping to limit overfitting in out dataset.

On introducing the same model to a larger and better dataset with some additional methods we obtained following results:

- No Chnage in Model

```
Epoch 22/25
73/73 [==============================] - 3s 35ms/step - loss: 0.0693 - acc: 0.9730 - val_loss: 0.0851 - val_acc: 0.9679
Epoch 23/25
73/73 [==============================] - 3s 35ms/step - loss: 0.0658 - acc: 0.9749 - val_loss: 0.0804 - val_acc: 0.9704
Epoch 24/25
73/73 [==============================] - 3s 35ms/step - loss: 0.0692 - acc: 0.9733 - val_loss: 0.0740 - val_acc: 0.9743
Epoch 25/25
73/73 [==============================] - 3s 34ms/step - loss: 0.0564 - acc: 0.9785 - val_loss: 0.0685 - val_acc: 0.9748
```

Fig 6.4

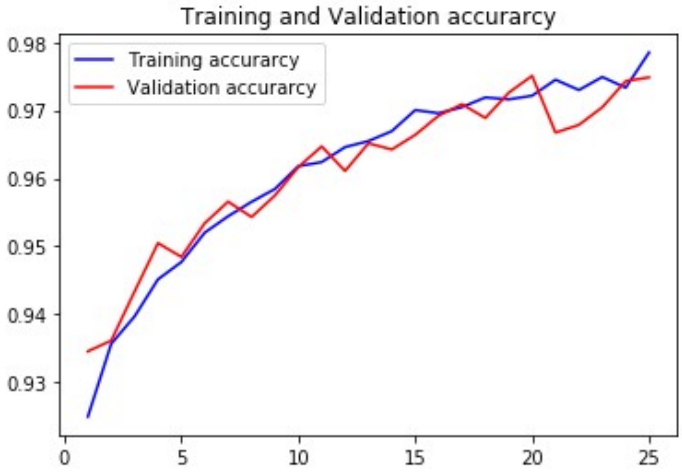

Fig 6.5

- With increase in image size from 64*64 to 256*256 – the accuracy improved to 98.82%

```
Epoch 21/25
73/73 [==============================] - 28s 388ms/step - loss: 0.0563 - acc: 0.9787 - val_loss: 0.1293 - val_acc: 0.960
9
Epoch 22/25
73/73 [==============================] - 29s 392ms/step - loss: 0.0511 - acc: 0.9810 - val_loss: 0.0928 - val_acc: 0.969
3
Epoch 23/25
73/73 [==============================] - 28s 389ms/step - loss: 0.0520 - acc: 0.9807 - val_loss: 0.0787 - val_acc: 0.970
6
Epoch 24/25
73/73 [==============================] - 28s 388ms/step - loss: 0.0529 - acc: 0.9802 - val_loss: 0.0933 - val_acc: 0.966
9
Epoch 25/25
73/73 [==============================] - 29s 401ms/step - loss: 0.0487 - acc: 0.9817 - val_loss: 0.1150 - val_acc: 0.964
0
```

Fig 6.6

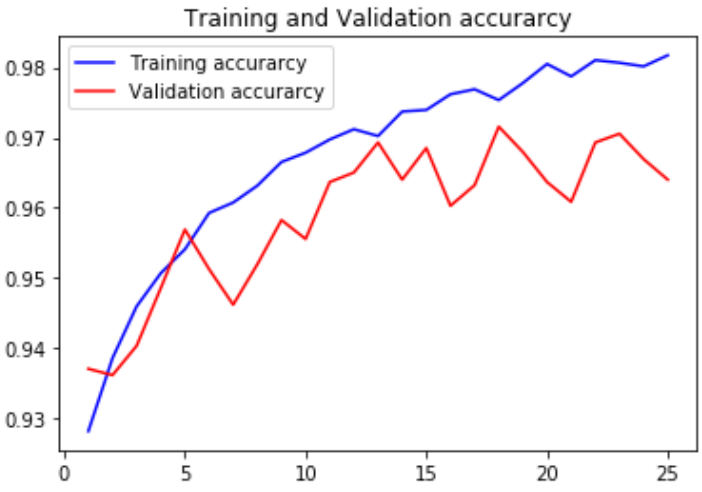

Fig 6.7

- With Batch Normalization, No Dropout

```
Epoch 21/25
73/73 [==============================] - 34s 470ms/step - loss: 1.0403 - acc: 0.9352 - val_loss: 1.0673 - val_acc: 0.933
4
Epoch 22/25
73/73 [==============================] - 35s 483ms/step - loss: 1.0448 - acc: 0.9348 - val_loss: 1.1081 - val_acc: 0.931
0
Epoch 23/25
73/73 [==============================] - 34s 472ms/step - loss: 1.0229 - acc: 0.9362 - val_loss: 0.9958 - val_acc: 0.938
2
Epoch 24/25
73/73 [==============================] - 34s 472ms/step - loss: 1.0038 - acc: 0.9375 - val_loss: 1.0900 - val_acc: 0.932
2
Epoch 25/25
73/73 [==============================] - 34s 471ms/step - loss: 1.0221 - acc: 0.9363 - val_loss: 1.0716 - val_acc: 0.933
2
```

Fig 6.8

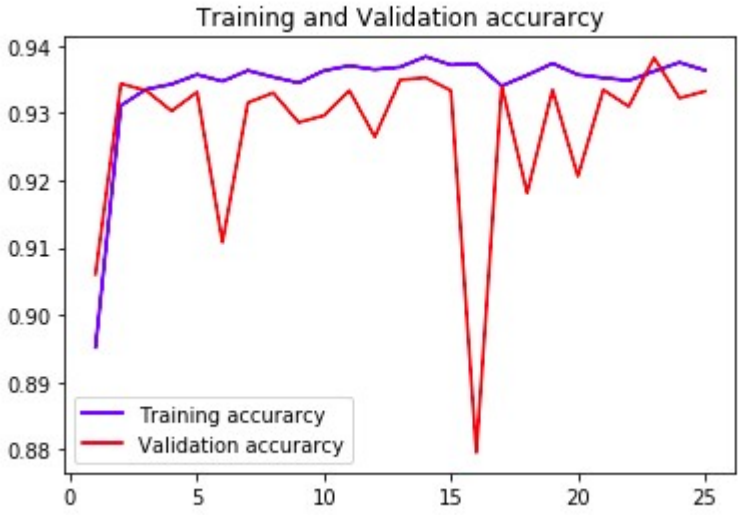

Fig 6.9

- Without Batch Normalization, Dropout 30%

```
9
Epoch 22/25
73/73 [==============================] - 35s 475ms/step - loss: 0.0786 - acc: 0.9710 - val_loss: 0.1080 - val_acc: 0.960
7
Epoch 23/25
73/73 [==============================] - 35s 477ms/step - loss: 0.0786 - acc: 0.9706 - val_loss: 0.0997 - val_acc: 0.962
1
Epoch 24/25
73/73 [==============================] - 36s 490ms/step - loss: 0.0755 - acc: 0.9710 - val_loss: 0.0961 - val_acc: 0.962
4
Epoch 25/25
73/73 [==============================] - 35s 478ms/step - loss: 0.0744 - acc: 0.9721 - val_loss: 0.1768 - val_acc: 0.945
0
```

Fig 6.10

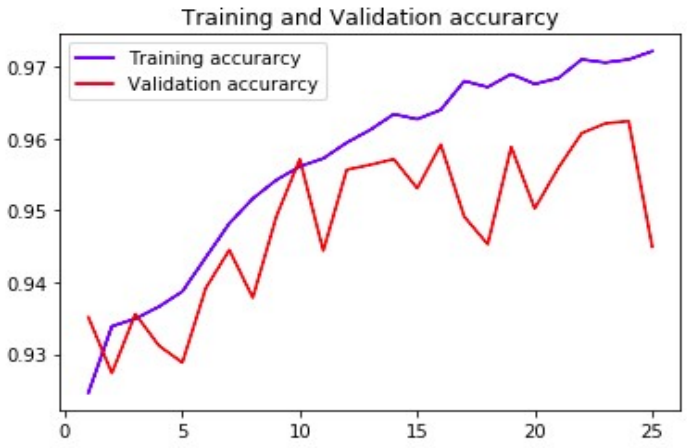

Fig 6.11

- With Both Batch Norrmalization and Dropout 30%- their use proved to be counter productive

```
Epoch 23/25
73/73 [==============================] - 30s 414ms/step - loss: 0.6454 - acc: 0.9552 - val_loss: 0.9133 - val_acc: 0.936
7
Epoch 24/25
73/73 [==============================] - 30s 410ms/step - loss: 0.6191 - acc: 0.9568 - val_loss: 0.7332 - val_acc: 0.947
1
Epoch 25/25
73/73 [==============================] - 30s 411ms/step - loss: 0.6312 - acc: 0.9563 - val_loss: 0.7846 - val_acc: 0.945
7
```

Fig 6.12

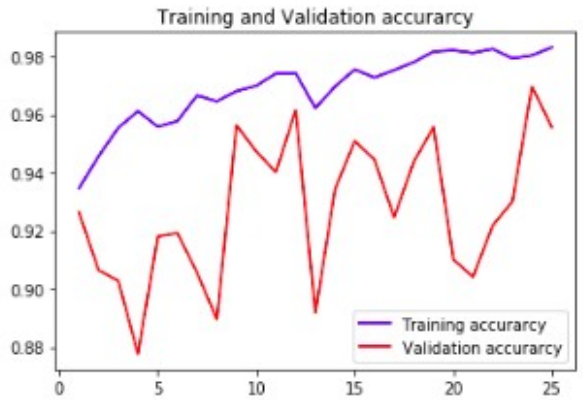

Fig 6.13

7. Conclusion of the report and Future Scope

- In this project, we trained a Convolutional Neural Network (CNN) Model using the Keras deep learning library. We prepared and obrained our datasets from Hare Krishna Farms, Noida and PlantVillage, a research and development unit of Penn State University, US for weed classification and disease identification respectively.
- In particular, our datasets consists of 1108 images for weed plants and tomato crop for first dataset and 20,654 images of 15 separate classes namely 'Pepper_bell_Bacterial_spot' , 'Pepper__bell___healthy', 'Potato_Early_blight' 'Potato_Late_blight' , 'Potato_healthy', 'Tomato_Bacterial_spot', 'Tomato_Early_blight' , 'Tomato_Late_blight', 'Tomato_Leaf_Mold' , 'Tomato_Septoria_leaf_spot', 'Tomato_Spider_mites_Two_spotted_spider_mite' , 'Tomato__Target_Spot', 'Tomato_healthy', 'Tomato_Tomato_Yellow_Leaf_Curl_Virus' , 'Tomato_Tomato_mosaic_virus', for disease data-set.
- Using our Convolutional Neural Network and Keras, we were able to obtain 97.81 and 98.82% accuracy, which is quite respectable given the limited size of our first dataset and the number of classess in our disease dataset respectively.
- In future the project enables an individual to use our trained Keras Convolutional Neural Network model and deploy it to a smartphone.

8. References:

Technical Report:

[1] C.-C. Jay Kuo Ming-Hsieh , "*Understanding Convolutional Neural Networks with A Mathematical Model",* Nov 2016*,* University of Southern California, Los Angeles, CA , USA

Online:

[2] David Mack, co-founder SketchDeck , researcher
https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2

[3] Prasad Pai, https://github.com/Prasad9
https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9

[4] https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

[5] https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Proceedings paper:

[6] Sachin D. Chothe, V.R.Ratnaparkhe,Vol. 4, Special Issue 6,*"Plant Identification Using Leaf Image*s " Government College of Engineering, Aurangabad, Maharashtra, India , May 2015

Journal Article:

[7] Jeon, Wang-Su & Rhee, Sang-Yong. (2017). "*Plant Leaf Recognition Using a Convolution Neural Network"* The International Journal of Fuzzy Logic and Intelligent Systems. 17. pp 26-34 March 2017

[8] Abdullahi, Halimatu & Sheriff, Ray & Mahieddine, Fatima "*Convolution neural network in precision agriculture for plant image recognition and classification."* 1-3. University of Bradford, Bradford August 2017