Indian Institute of Information Technology Vadodara
Government Engineering College, Sector-28, Gandhinagar,
Gujarat - 382028

Design Project Report

on

Krishi Unnati: A Mobile Application for Plant and Crop Disease Detection using Convolutional Neural Networks

Submitted by

Milind Pandey	**Prateek Senapati**	**Sudhanshu Pandey**	**Sumit Kumar Mishra**
201851068	**201851090**	**201851131**	**201851133**

under the supervision of

Dr. Bhupendra Kumar

Abstract- **Convolutional Neural Networks have been observed to be very successful and efficient when it comes to image classification. Our design project aims at using a state-of-the-art CNN architecture, VGG16, to build a model for detecting plant and crop diseases using the images of the diseased leaves. We have used transfer learning and trained our model on a publicly available dataset consisting of 61,486 images categorised into 39 different classes of [plant, disease] combinations, including healthy plants. Out of all the various configurations of the VGG16 architecture we have tried, the best performing model architecture gives a training accuracy of 100% and a test accuracy of 96.61%. We have also developed a mobile application, named Krishi Unnati, for the same purpose which can be conveniently used by farmers for detecting crop and plant diseases - without any reliance on human expertise.**

I. Introduction

Crop diseases are a serious threat to food security. If not taken care of, crop diseases can lead to food shortage and wastage. These can also result in financial losses. Therefore, controlling and preventing crop diseases are very vital to agriculture and related activities. Deep learning models can be used for rapid identification of crop diseases, which can help in early prevention. Since mobile phones are an essential part of life in today's world, having a mobile application for the same purpose can potentially work as an early warning tool for farmers. This increases productivity as well as reduces the reliance on human expertise for crop disease detection. Previous research papers on the same topic have been referred to for ideas. Different variations of VGG16, a convolutional neural networks architecture for image classification, were trained (using transfer learning method) on a public dataset of 61,486 coloured images (collected under controlled conditions) - spread across 39 distinct classes of [plant, disease] combinations, including healthy plants. The best performing configuration of VGG16 (in our case) achieved a training accuracy of 100% and a test accuracy of 96.61%. The mobile application provides a minimalistic and easy-to-use interface which can be used by farmers effortlessly. The results can be further improved by making enhancements to our model as well as the whole project, but as of now, this is a good start and shows great promise for the future where mobile-assisted crop disease detection might become an indispensable part of the agriculture industry.

II. Literature Survey

We have referred to a few research papers wherein people have worked on the same or a similar problem statement, viz., Using Deep Learning for Image-Based Plant Disease Detection [1], Deep learning models for plant disease detection and diagnosis [2] and On Using Transfer Learning For Plant Disease Detection [3].

III. THE PRESENT INVESTIGATION

- *Technologies Used*

  For computational requirements, we have used Google Colaboratory as it provides free access to GPUs which are powerful enough for our use case. We have used Keras, a deep learning framework built on top of TensorFlow 2.0, for building, training and testing our deep learning model. For mobile application development, we have used React Native + Expo (for frontend development) and Django (for backend development).

- *Dataset*

  We have used a dataset titled Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network [4]. This is a public dataset of 61,486 coloured images (collected under controlled conditions) - spread across 39 distinct classes of [plant, disease] combinations, including healthy plants. This dataset includes augmented images. Six different data augmentation techniques (image flipping, Gamma correction, noise injection, PCA color augmentation, rotation, and scaling) have been used to achieve data augmentation in this dataset.

  The 39 different classes are:

  Apple scab, Apple black rot, Apple cedar apple rust, Apple healthy, Background without leaves, Blueberry healthy, Cherry powdery mildew, Cherry healthy, Corn gray leaf spot, Corn common rust, Corn northern leaf blight, Corn healthy, Grape black rot, Grape black measles, Grape leaf blight, Grape healthy, Orange haunglongbing, Peach bacterial spot, Peach healthy, Pepper bacterial spot, Pepper healthy, Potato early blight, Potato healthy, Potato late blight, Raspberry healthy, Soybean healthy, Squash powdery mildew, Strawberry healthy, Strawberry leaf scorch, Tomato bacterial spot, Tomato early blight, Tomato healthy, Tomato late blight, Tomato leaf mold, Tomato septoria leaf spot, Tomato spider mites two-spotted spider mite, Tomato target spot, Tomato mosaic virus, Tomato yellow leaf curl virus.

- *Data Pre-processing*

  All the images have been reshaped to a size of 224 * 224 * 3 and rescaled by a factor of 1/255. We have used a batch size of 128 and split the dataset into training and validation subsets with a train-test split ratio of 80:20.

- *Model Architecture*

  We have used a pre-trained VGG16 model trained on the imagenet dataset. First, we removed all the dense (fully connected) layers present after the last max-pooling layer in the original architecture (Figure 1) and froze all the remaining layers. Then, we flattened the output from the max-pooling layer. Finally, we fed the output to a dense layer with 39 units (for 39 classes) and softmax activation. We trained the remaining trainable weights for 25 epochs, and we used Adam optimizer for updating the trainable weights during backpropagation.

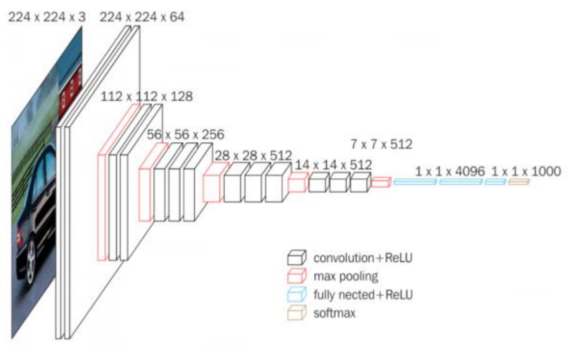


Figure 1: VGG16 model architecture

We tried some variations in configuring our model's architecture, such as adding dropout, keeping the dense layers (after the final max-pooling layer) from the original VGG16 architecture, adding multiple trainable dense layers after the final max-pooling layer, changing the batch size and the number of epochs. But the model architecture we mentioned in the previous paragraph gave us the best results. Hence, we finalised that model architecture (Figure 2) built using transfer learning.

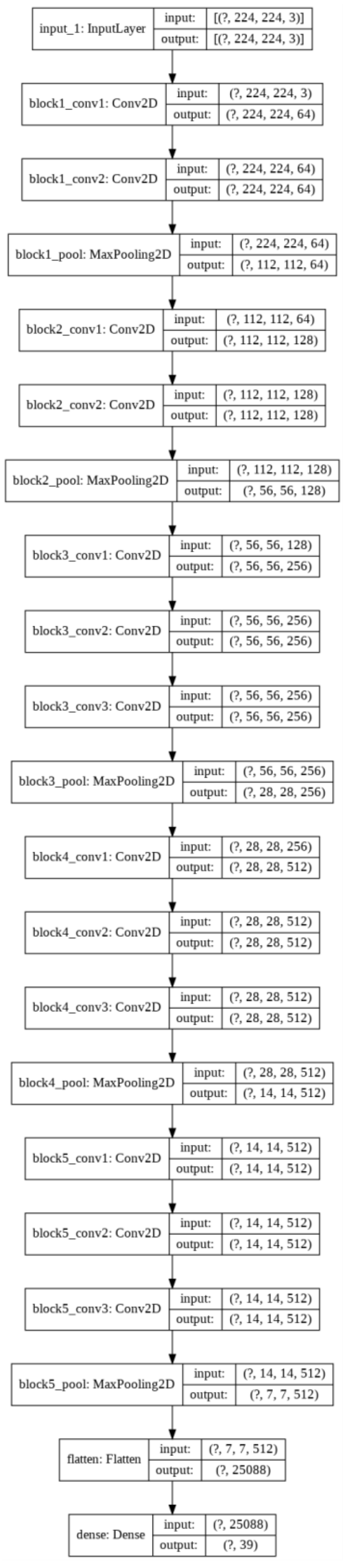
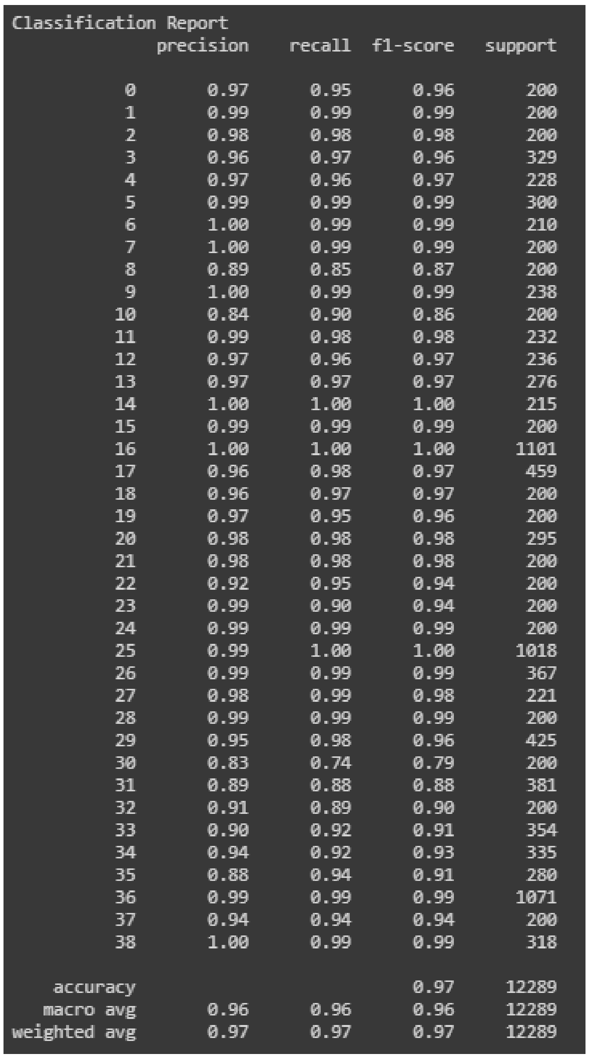

input_1: InputLayer | input: [(?, 224, 224, 3)] | output: [(?, 224, 224, 3)]
block1_conv1: Conv2D | input: (?, 224, 224, 3) | output: (?, 224, 224, 64)
block1_conv2: Conv2D | input: (?, 224, 224, 64) | output: (?, 224, 224, 64)
block1_pool: MaxPooling2D | input: (?, 224, 224, 64) | output: (?, 112, 112, 64)
block2_conv1: Conv2D | input: (?, 112, 112, 64) | output: (?, 112, 112, 128)
block2_conv2: Conv2D | input: (?, 112, 112, 128) | output: (?, 112, 112, 128)
block2_pool: MaxPooling2D | input: (?, 112, 112, 128) | output: (?, 56, 56, 128)
block3_conv1: Conv2D | input: (?, 56, 56, 128) | output: (?, 56, 56, 256)
block3_conv2: Conv2D | input: (?, 56, 56, 256) | output: (?, 56, 56, 256)
block3_conv3: Conv2D | input: (?, 56, 56, 256) | output: (?, 56, 56, 256)
block3_pool: MaxPooling2D | input: (?, 56, 56, 256) | output: (?, 28, 28, 256)
block4_conv1: Conv2D | input: (?, 28, 28, 256) | output: (?, 28, 28, 512)
block4_conv2: Conv2D | input: (?, 28, 28, 512) | output: (?, 28, 28, 512)
block4_conv3: Conv2D | input: (?, 28, 28, 512) | output: (?, 28, 28, 512)
block4_pool: MaxPooling2D | input: (?, 28, 28, 512) | output: (?, 14, 14, 512)
block5_conv1: Conv2D | input: (?, 14, 14, 512) | output: (?, 14, 14, 512)
block5_conv2: Conv2D | input: (?, 14, 14, 512) | output: (?, 14, 14, 512)
block5_conv3: Conv2D | input: (?, 14, 14, 512) | output: (?, 14, 14, 512)
block5_pool: MaxPooling2D | input: (?, 14, 14, 512) | output: (?, 7, 7, 512)
flatten: Flatten | input: (?, 7, 7, 512) | output: (?, 25088)
dense: Dense | input: (?, 25088) | output: (?, 39)

Figure 2: Model architecture used in our project

IV. RESULTS AND DISCUSSIONS

- **Batch size:** 128
- **Number of epochs:** 25
- **Train-test split ratio:** 80:20
- **Training accuracy:** 100%
- **Test accuracy:** 96.61%
- **Precision (weighted average):** 97%
- **Recall (weighted average):** 97%
- **F1 score:** 97%

The classification report (Figure 3) shows class-wise evaluation metric values along with other important details.



```
Classification Report
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       200
           1       0.99      0.99      0.99       200
           2       0.98      0.98      0.98       200
           3       0.96      0.97      0.96       329
           4       0.97      0.96      0.97       228
           5       0.99      0.99      0.99       300
           6       1.00      0.99      0.99       210
           7       1.00      0.99      0.99       200
           8       0.89      0.85      0.87       200
           9       1.00      0.99      0.99       238
          10       0.84      0.90      0.86       200
          11       0.99      0.98      0.98       232
          12       0.97      0.96      0.97       236
          13       0.97      0.97      0.97       276
          14       1.00      1.00      1.00       215
          15       0.99      0.99      0.99       200
          16       1.00      1.00      1.00      1101
          17       0.96      0.98      0.97       459
          18       0.96      0.97      0.97       200
          19       0.97      0.95      0.96       200
          20       0.98      0.98      0.98       295
          21       0.98      0.98      0.98       200
          22       0.92      0.95      0.94       200
          23       0.99      0.90      0.94       200
          24       0.99      0.99      0.99       200
          25       0.99      1.00      1.00      1018
          26       0.99      0.99      0.99       367
          27       0.98      0.99      0.98       221
          28       0.99      0.99      0.99       200
          29       0.95      0.98      0.96       425
          30       0.83      0.74      0.79       200
          31       0.89      0.88      0.88       381
          32       0.91      0.89      0.90       200
          33       0.90      0.92      0.91       354
          34       0.94      0.92      0.93       335
          35       0.88      0.94      0.91       280
          36       0.99      0.99      0.99      1071
          37       0.94      0.94      0.94       200
          38       1.00      0.99      0.99       318

    accuracy                           0.97     12289
   macro avg       0.96      0.96      0.96     12289
weighted avg       0.97      0.97      0.97     12289
```

Figure 3: Classification report

An example of our model's prediction can be seen in Figure 4, wherein our model has successfully predicted a leaf of a corn plant suffering from common rust with 100% confidence value.

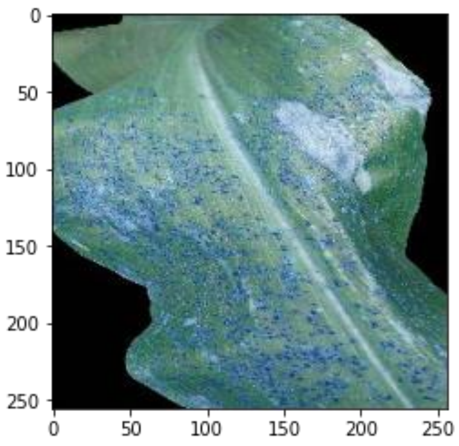
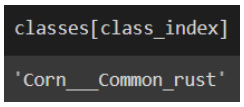


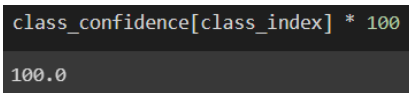
Figure 4 (a): An image of a corn plant leaf suffering from common rust



```
classes[class_index]

'Corn___Common_rust'
```

Figure 4 (b): Our model's prediction for Figure 4 (a)



```
class_confidence[class_index] * 100

100.0
```

Figure 4 (c): Our model's class confidence value for Figure 4 (a)

V. CONCLUSIONS AND FUTURE WORK

Our model architecture, though simple, resulted in a 96.61% test accuracy while detecting crop diseases - showing how suitable convolutional neural networks are for this purpose. Moreover, a deep learning model like this requires very little computational power which makes it very efficient to be integrated with mobile devices. There is a lot of scope for improvement: a much larger and varied dataset can improve the model's performance, images with real world background or images captured in real world (and not in controlled conditions) will help the application as well as the model perform better in real world. Our final goal is to make this application as helpful as possible for the Indian farmers and the Indian agriculture sector. This can be achieved by adding images of crop and plant diseases specifically found/seen in India. We can also enhance the user experience of Indian farmers by adding Indian languages to our mobile application which will make our application more accessible and more convenient to use for them.

ACKNOWLEDGMENT

We are thankful to Google for providing Google Colaboratory which gives free access to GPUs – without which completing this project would have been really difficult for us. We would also like to thank our mentor, Dr. Bhupendra Kumar, who encouraged us to explore freely and try things out by taking risks. This open-minded and less constraining approach towards the project has helped us to learn new technologies more effectively and enabled us to enjoy the learning process while working on our design project.

REFERENCES

[1] Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in plant science, 7, 1419.

[2] Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. Computers and Electronics in Agriculture, 145, 311-318.

[3] Sagar, A., & Dheeba, J. (2020). On Using Transfer Learning For Plant Disease Detection. bioRxiv.

[4] J, ARUN PANDIAN; GOPAL, GEETHARAMANI (2019), "Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network", Mendeley Data, V1, doi: 10.17632/tywbtsjrjv.1