IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

IOT-BASED INTELLIGENT FARMING EARLY DISEASE DETECTION IN RICE PADDY USING CNN

[1]Apoorva, [2]Kiran BB
[1]Student, [2]Student
[1]Department of Computer Science and Engineering,
[1]Kvg College Of Engineering, Sullia, D.K, India

Abstract *:* The agricultural industry faces significant economic losses each year due to bacterial, viral, or fungal infections in crops, resulting in a reduction of 15-20% in profits for farmers. India is a leading producer and exporter of rice, making early detection of crop diseases crucial. Smart Farming, which integrates Machine Learning with an IoT network, is a crucial area of research to prevent further damage to crops. In this study, an intelligent model based on Smart Farming has been proposed, which uses real-time data, image recognition, and pre-processing A self-designed image-processing tool was used for data pre-processing and feature extraction. Tensorflow and Keras frameworks were implemented on manually collected training and testing data from rice fields, resulting in an accuracy of 97.70%. Additionally, an app has been designed for farmers to utilize this technology.
***IndexTerms* - Convolution neural networks, smart farming, Internet of Things, Rice Paddy, AI**

1. INTRODUCTION

This text discusses the issue of crop diseases and how Machine Learning (ML) and Deep Learning (DL) can be used to detect these diseases. The importance of agriculture for human population is highlighted, with rice being the most important food crop in the world. The use of IoT and cloud computing for disease detection is explained, with drones used for image collection and ML algorithms used for prediction. The advantages of DL over traditional ML techniques are discussed, including its ability to extract features automatically and avoid complex image pre-processing. The use of Convolutional Neural Networks (CNNs) is described in detail, with CNNs being a subclass of DL neural networks that can automatically extract features from images. The advantages of using CNNs for disease detection in paddy are explained, including their ability to limit the number of parameters and avoid overfitting. The text concludes by highlighting the importance of DL in smart farming and its potential to improve crop production efficiency and food security.

2. Background and Related works

The article discusses the use of IoT and machine learning techniques for disease detection in crops, specifically Brown spot disease in rice paddy. The paper presents the need for a disease detection mechanism and highlights the losses caused by the disease to rice yields. The article also presents previous work in disease detection using image processing and machine learning techniques, including works on apple plants, strawberries, and cotton leaf. However, the article claims that the previous techniques used were computationally expensive and not practical. The article proposes an approach that uses IoT-based smart sensing and machine learning algorithms for disease detection in rice paddy. The proposed approach involves placing sensors on farms to collect data, using a drone to collect images, transmitting the data for analysis and prediction, and monitoring crops from an analytical dashboard or mobile app. The article suggests that the proposed approach can be used to detect Brown spot disease in rice paddy, which can significantly reduce losses caused by the disease.

3. Proposed model framework

The proposed model's framework is presented in Fig. 1a and b, with data acquisition from IoT networks relying on Edge Intelligence and Fog Intelligence. Images are captured and stored in Cloud servers, and pre-processing is conducted before feeding data into the machine learning algorithms. To collect data from rice paddies, a drone was used for real-time detection, and a Java-based image processing tool was developed for feature selection. The tool was used to mark the areas of the rice paddy affected by Brown spot disease and saved in XML files. The training and testing datasets were prepared using the same tool, and the training model was created in Python. The CNN model trains itself in the feature learning network after model fitting. Finally, the model was evaluated using various metrics, including training and validation accuracy, training and validation loss, and an app was developed for farmers to validate instantly and make informed decisions.

4. Methodology

The experiment was conducted using a Lenovo ideapad 510 laptop running Windows 10 Enterprise 64-bit as the edge device. The images of the diseased rice paddy were captured using a DJI Tello drone camera. The hardware specifications of the laptop and the drone are listed in Table 1

To create the model, Python 3.7 was used with the Anaconda distribution. The necessary packages including TensorFlow and Keras were imported into a virtual environment created in Anaconda navigator. The Future module was utilized to prevent confusion for existing tools that analyze import statements and expect to find the modules they are importing. NumPy, OpenCV, and TensorFlow were used to prepare the training model. Matplotlib was used for data visualization, and Keras framework with TensorFlow as its backend was used for model fitting and evaluation since Keras supports almost all models of a neural

4.1 Data acquisition

The initial stage of building the Deep Learning model is data acquisition, which involves collecting real-world data samples under physical conditions. In this project, no pre-existing dataset was used, and images of rice paddy affected by Brown spot disease were manually taken from various angles and positions in the field. The symptoms of Brown spot disease were observed to be small circular dark brown to greyish spots on the leaves of rice paddy. Toxin produced by the fungi caused fully developed lesions that were circular or oval brown spots with a greyish center. Fig. 2 provides a sample of some of the images collected

Drone shots and IoT sensors were employed to collect images under different environmental conditions, and gateway devices facilitated the communication between the IoT network and the cloud. Cloud computing, an emerging technology in smart farming, was used to prepare the dataset, and cloud computing servers were employed for storing the collected data.

4.2 Image preprocessing

Data pre-processing is a crucial step in preparing data collected from the fields for machine learning algorithms. To achieve this, the researchers designed an image processing tool using Java Platform (JDK 8) and Eclipse IDE to perform feature selection. The tool allowed for manual markup of brown spot areas on the rice paddy leaves, which were saved as objects (markups) in Extensive Markup Language (XML) data files. Using the same tool, the researchers loaded the XML file to fragment the marked and unmarked portions of the image uniformly with a horizontal shift, vertical shift, height, and width of 28. The resulting image fragments were then archived into zip files, which constituted the training and testing datasets for the model. Some of the resulting image fragments of brown spots and fresh paddy were shown in Figs. 4 and 5, respectively.

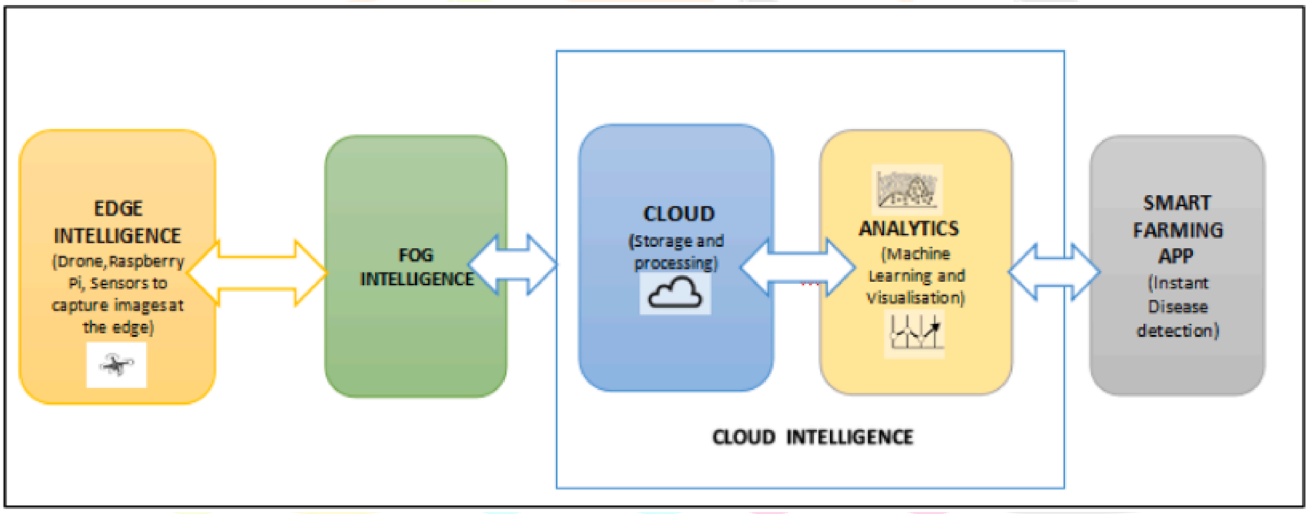

EDGE INTELLIGENCE
(Drone, Raspberry Pi, Sensors to capture images at the edge)
FOG INTELLIGENCE
CLOUD
(Storage and processing)
ANALYTICS
(Machine Learning and Visualisation)
SMART FARMING APP
(Instant Disease detection)
CLOUD INTELLIGENCE

fig 1. a) basic flow of the model incorporated

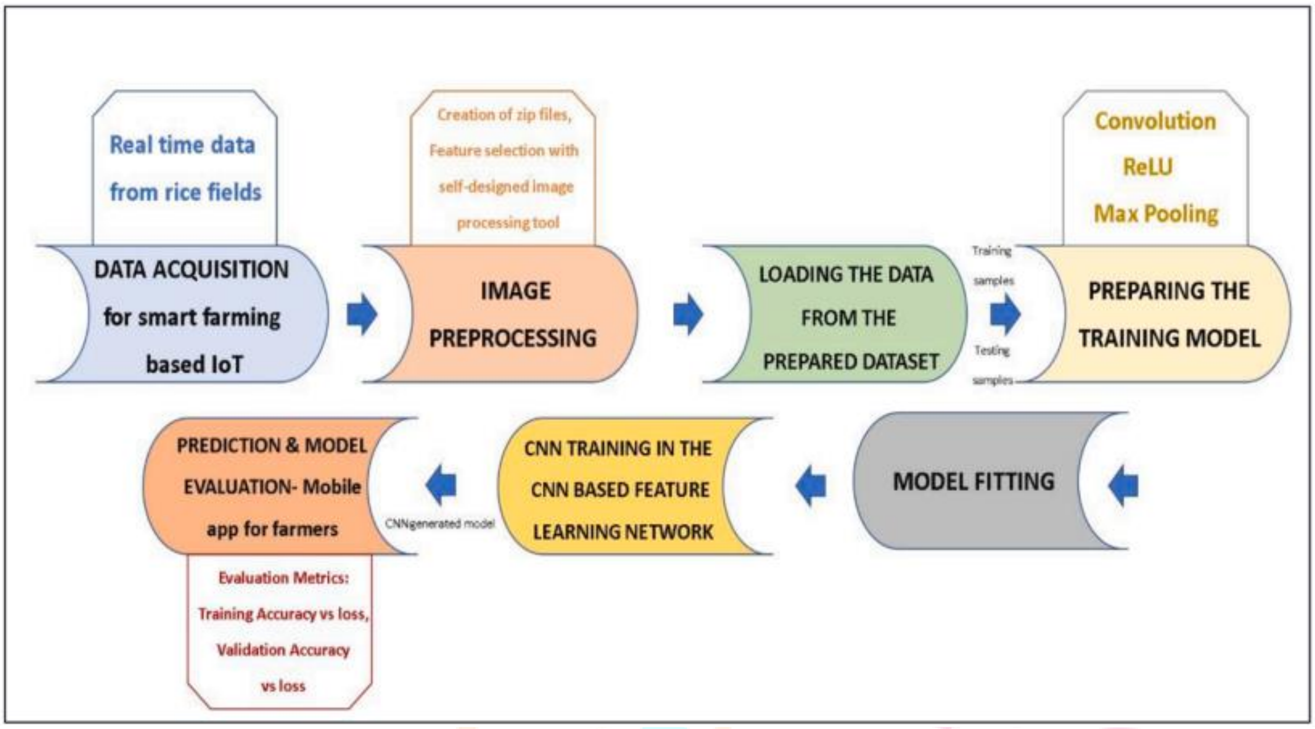

Real time data from rice fields
DATA ACQUISITION for smart farming based IoT
Creation of zip files, Feature selection with self-designed image processing tool
IMAGE PREPROCESSING
LOADING THE DATA FROM THE PREPARED DATASET
Training samples
Testing samples
Convolution
ReLU
Max Pooling
PREPARING THE TRAINING MODEL
MODEL FITTING
CNN TRAINING IN THE CNN BASED FEATURE LEARNING NETWORK
CNNgenerated model
PREDICTION & MODEL EVALUATION- Mobile app for farmers
Evaluation Metrics: Training Accuracy vs loss, Validation Accuracy vs loss

fig 1. b) the detailed framework of the proposed model

Table 1 Hardware Specification of the system and the camera

Elements	**Hardware specification**
Processor	Intel(R) core(TM) i5–7200U
CPU	@ 2.50 GHz 2.70GHz
RAM	8.00 GB
Graphics card	NVIDIA GeForce 940 MX 2GB
Photo ISO Range	100 – 3200 (Auto)
Video ISO Range	100 – 6400 (Auto)
Video Resolution of Dji Tello Drone	720p

Next using the same tool, we loaded the xml file to fragment the marked and the unmarked portions of the image . The horizontal shift, vertical shift, height and width of all the image fragments were kept uniform as 28. Archives (zip files) were created to store the image fragments for each image. The image fragments of the brown spots constituted our Brownspot class in the training model and the image fragments of the fresh areas constituted the Fresh class. Thus, the training and testing dataset were prepared.

4.3 Loading the data from the prepared dataset

After performing image preprocessing, the dataset obtained from the IoT network was loaded into the training model. The training model was created using Python version 3.7.4. To store the dataset, zip files, which are the most commonly used type of archive files, were utilized. The training and testing dataset were stored in separate zip files, and a Python code was developed using the Zipfile library to unzip and read the contents of these files.

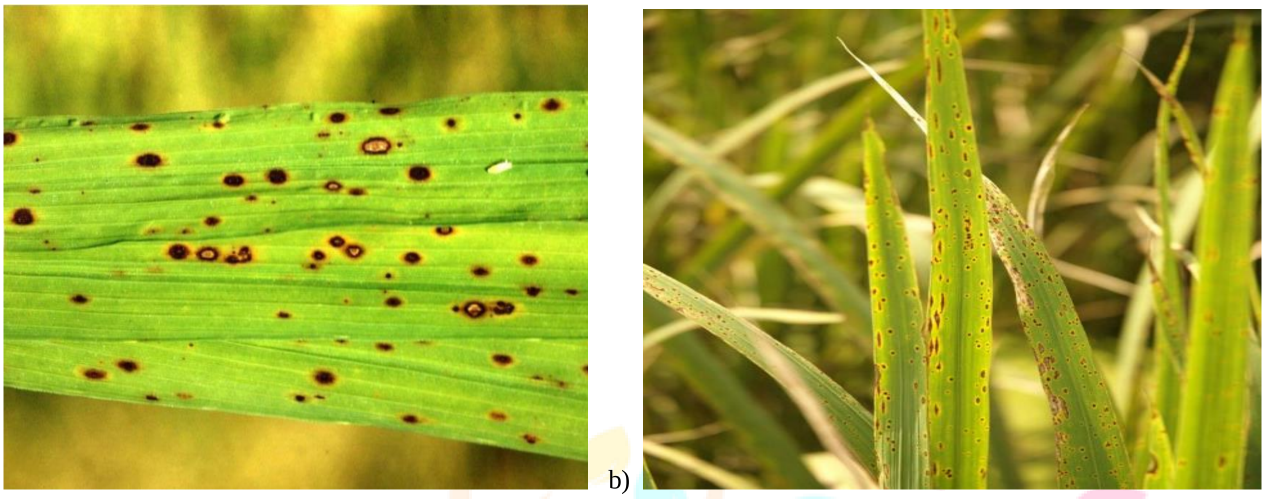
The algorithm for this is as follows:

1) From zipfile import Zipfile
2) Declare a file_name variable to store the path of the zipfile.
3) Open the zipfile in read mode : " with ZipFile(file_name, 'r') as zip:
4) Use the extractall() function to extract the content stored in the zip : " zip.extractall()"

4.4 Preparing the training model

Following are the steps which is used to prepare training model using CNN:

1) **Define a function to encode the labels:** The labels "Brown spot" and "Fresh" have been employed in the design of this model.



a) b)

fig 2. (a, b) few samples of the images captured from the field for brown spot disease detection

2) **Pixel values to be normalised between 0 to 1:** Normal viewing of images is possible when their pixel values fall within the 0-1 range which can be attained by dividing all pixel values by the highest pixel value, which is typically 255.
3) **Converting the input dataset to an array:** np.asarray() which is used to convert the input dataset to an array for the testing images, training labels and test labels. np.asarray() function which is used to convert input to an array.
4) **Load the image from Opencv:** function from the Opencv() library which is used to read the locations of the training and Testing images. cv2.imread() method loads image from an specified file. If image can't be read then it returns the empty.
5) **Resizing the image with interpolation**: Resize the training and the testing image to 28×28 pixel and read it as greyscale Image Using the resize function from Opencv library
6) **Split the training and the testing data:** 80% of the data collected was considered for training and 20% of the data for testing In the model of depicted area.
7) **Calling Keras layers on TensorFlow tensors** : A tensorflow session is created registering it with Keras. Tensorflow allows The developers to create dataflow graphs which describe how the data moves through the graph. Each node in the graph represents a series of mathematical operations and each connected edge between the nodes is a multidimensional data array or a tensor. To actually evaluate the nodes, computational graph must be run in a session. A session encapsulates the control and state of the Tensorflow runtime. " sess = tf.compat. v1.Session() followed by "K.set_session(sess) " creates a Session object and runs it with Keras API. Keras layers can then be used to define the model. The convolutional neural networks model is built on several layers of convolution, relu and pooling and a fully connected layer. The layers used in the proposed model model uses convolutional neural network. The sequential model is a linear stack of layers. We have added the layers using the add() method in our model.

The following table which shows how layers have been stacked up in the sequential model built on keras for our CNN. The sequential model is linear stack of layers. We have added the layers sing add() method in our model.

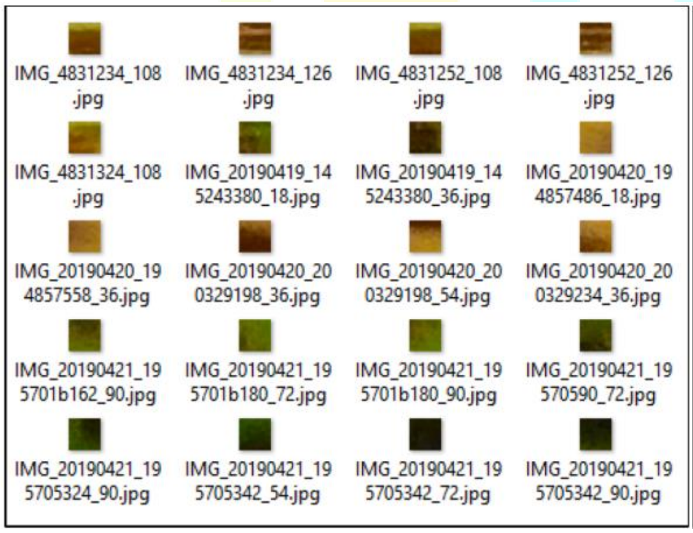

IMG_4831234_108.jpg
IMG_4831234_126.jpg
IMG_4831252_108.jpg
IMG_4831252_126.jpg
IMG_4831324_108.jpg
IMG_20190419_145243380_18.jpg
IMG_20190419_145243380_36.jpg
IMG_20190420_194857486_18.jpg
IMG_20190420_194857558_36.jpg
IMG_20190420_200329198_36.jpg
IMG_20190420_200329198_54.jpg
IMG_20190420_200329234_36.jpg
IMG_20190421_195701b162_90.jpg
IMG_20190421_195701b180_72.jpg
IMG_20190421_195701b180_90.jpg
IMG_20190421_19570590_72.jpg
IMG_20190421_195705324_90.jpg
IMG_20190421_195705342_54.jpg
IMG_20190421_195705342_72.jpg
IMG_20190421_195705342_90.jpg

Fig. 4. Image fragments of brown spot

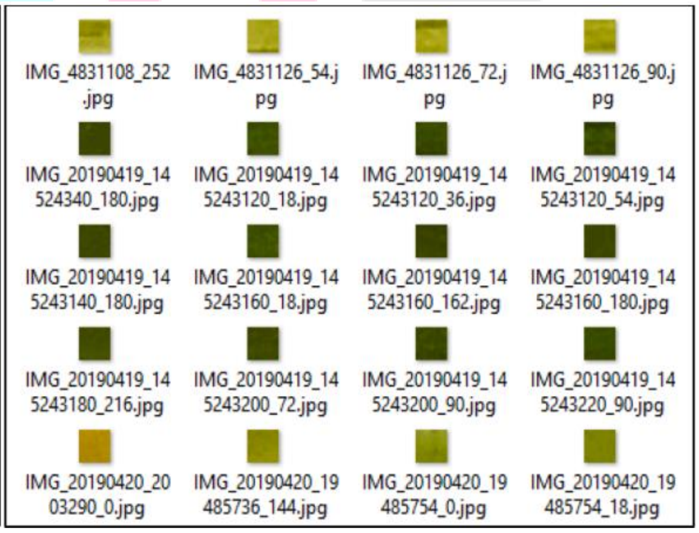

IMG_4831108_252.jpg
IMG_4831126_54.jpg
IMG_4831126_72.jpg
IMG_4831126_90.jpg
IMG_20190419_14524340_180.jpg
IMG_20190419_145243120_18.jpg
IMG_20190419_145243120_36.jpg
IMG_20190419_145243120_54.jpg
IMG_20190419_145243140_180.jpg
IMG_20190419_145243160_18.jpg
IMG_20190419_145243160_162.jpg
IMG_20190419_145243160_180.jpg
IMG_20190419_145243180_216.jpg
IMG_20190419_145243200_72.jpg
IMG_20190419_145243200_90.jpg
IMG_20190419_145243220_90.jpg
IMG_20190420_2003290_0.jpg
IMG_20190420_19485736_144.jpg
IMG_20190420_19485754_0.jpg
IMG_20190420_19485754_18.jpg

Fig. 5. Image fragments of fresh paddy.

Table 2 Layers used in training the experimental model

Layer 1	Convolution 2D
Layer 2	Activation = relu
Layer 3	Max Pooling 2D
Layer 4	Convolutional 2D
Layer 5	Activation = relu
Layer 6	Max Pooling 2D
Layer 7	Convolutional 2D
Layer 8	Activation = relu
Layer 9	Flatten(Fully connected Layer)
Layer 10	Dense(1st hidden Layer)
Layer 11	Activation = relu
Layer 12	Dense(2nd hidden Layer)
Layer 13	Activation = softmax

The layers and the parameters used in them are described as follows:

- The proposed convolutional neural network model consists of multiple layers including Conv2D, MaxPooling2D, Flatten, and Dense
- The Conv2D layer has 32 filters and a kernel size of (3,3).
- The input shape parameter specifies the dimensions of the input volume, which is a tuple/list of 2 integers that determine the step of the convolution along with the height and width.
- The activation parameter to the Conv2D class specifies the activation function to apply after performing the convolution which in this case is ‘relu’
- MaxPooling2D with a window size of (2,2) is used to reduce the dimensions of the output volume.
- Flatten() function is used to flatten the input, which is the first step of the fully connected layer.
- The first hidden layer of the fully connected layer is a Dense layer with 64 neurons, and the activation function 'relu’ is applied to the first and output layer.
- The second hidden layer is also a Dense layer with 10 neurons, and the activation function 'softmax' is applied to the output layer
- The model.summary() method is used to examine the architecture of the network and the number

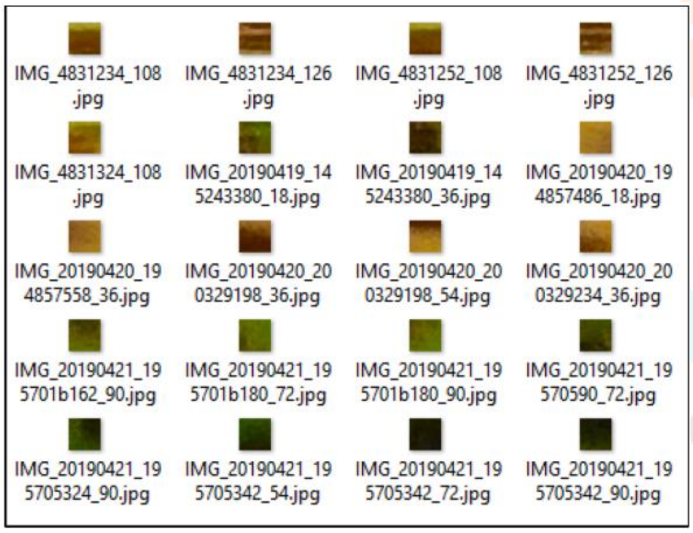

IMG_4831234_108.jpg
IMG_4831234_126.jpg
IMG_4831252_108.jpg
IMG_4831252_126.jpg
IMG_4831324_108.jpg
IMG_20190419_145243380_18.jpg
IMG_20190419_145243380_36.jpg
IMG_20190420_194857486_18.jpg
IMG_20190420_194857558_36.jpg
IMG_20190420_200329198_36.jpg
IMG_20190420_200329198_54.jpg
IMG_20190420_200329234_36.jpg
IMG_20190421_195701b162_90.jpg
IMG_20190421_195701b180_72.jpg
IMG_20190421_195701b180_90.jpg
IMG_20190421_19570590_72.jpg
IMG_20190421_195705324_90.jpg
IMG_20190421_195705342_54.jpg
IMG_20190421_195705342_72.jpg
IMG_20190421_195705342_90.jpg

fig. 4. image fragments of brown spot

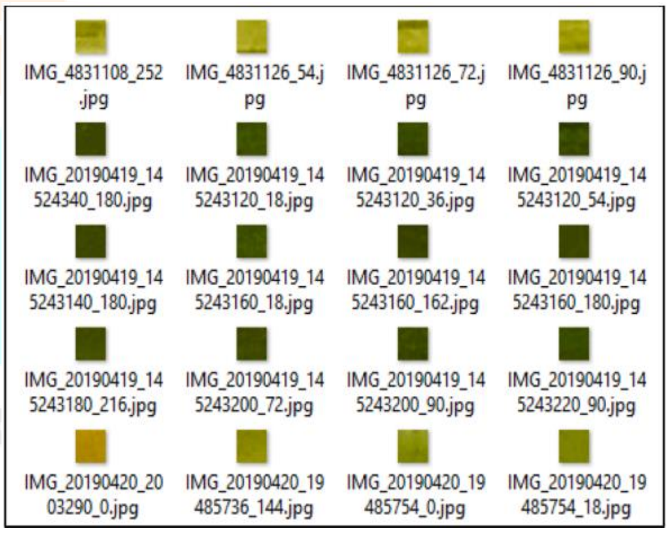
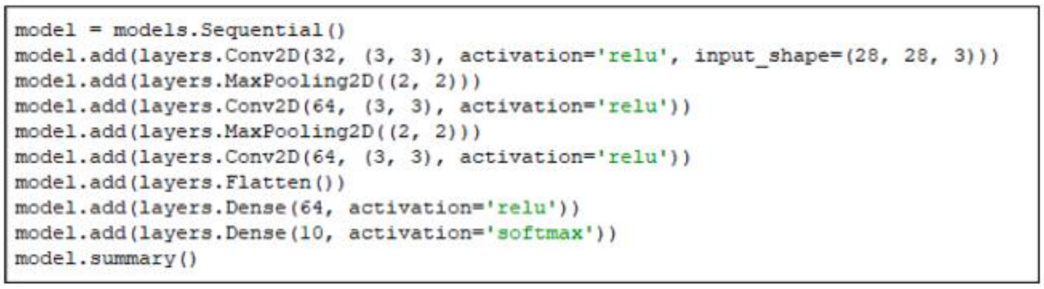

IMG_4831108_252.jpg
IMG_4831126_54.jpg
IMG_4831126_72.jpg
IMG_4831126_90.jpg
IMG_20190419_14524340_180.jpg
IMG_20190419_145243120_18.jpg
IMG_20190419_145243120_36.jpg
IMG_20190419_145243120_54.jpg
IMG_20190419_145243140_180.jpg
IMG_20190419_145243160_18.jpg
IMG_20190419_145243160_162.jpg
IMG_20190419_145243160_180.jpg
IMG_20190419_145243180_216.jpg
IMG_20190419_145243200_72.jpg
IMG_20190419_145243200_90.jpg
IMG_20190419_145243220_90.jpg
IMG_20190420_2003290_0.jpg
IMG_20190420_19485736_144.jpg
IMG_20190420_19485754_0.jpg
IMG_20190420_19485754_18.jpg

fig. 5. image fragments of fresh paddy.

4.5 Model fitting

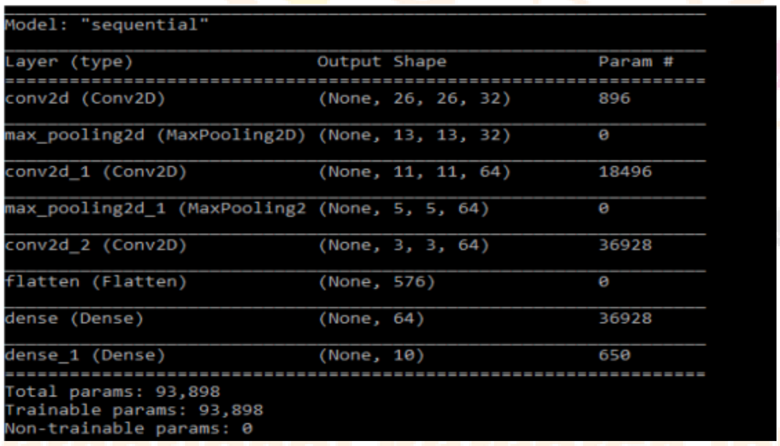
The loss function is used by the machine to assess how well the CNN algorithm has represented the data, with a higher value indicating greater deviation between predicted and actual results. On the other hand, accuracy is a metric provided by Keras API that calculates the percentage of predicted values that match the actual values. The total number of accurate predictions is divided by the total number of predictions made to compute accuracy.
When using Keras, the model.fit() method returns a history callback containing a history attribute that holds lists of losses and accuracies for each epoch. The CNN is trained for 1000 epochs, and the model.history. history method is used to obtain accuracy and losses for both the training and testing datasets across each epoch.

4.5.1 Compiling the model: Before training a model, it is necessary to configure the learning process using the compile() method which takes three arguments. One of these arguments is the optimizer, which in this case is 'adam'. This optimizer is designed to adjust the weights and learning rates of the CNN in order to minimize losses during training.
Another argument is the loss function, which in this CNN model is the categorical cross entropy. For validation data, the sparse_categorical_entropy function is used along with the test dataset and labels. The number of iterations performed by the ML algorithm is represented by epochs.



```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

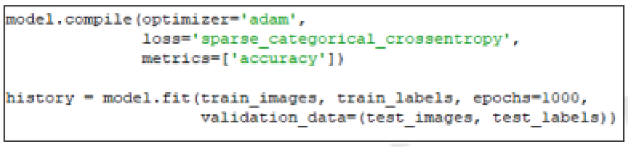
fig 6. code snippet of stacking up of the layers in the training model on keras API



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,898
Trainable params: 93,898
Non-trainable params: 0
```

Fig 7. the model summary

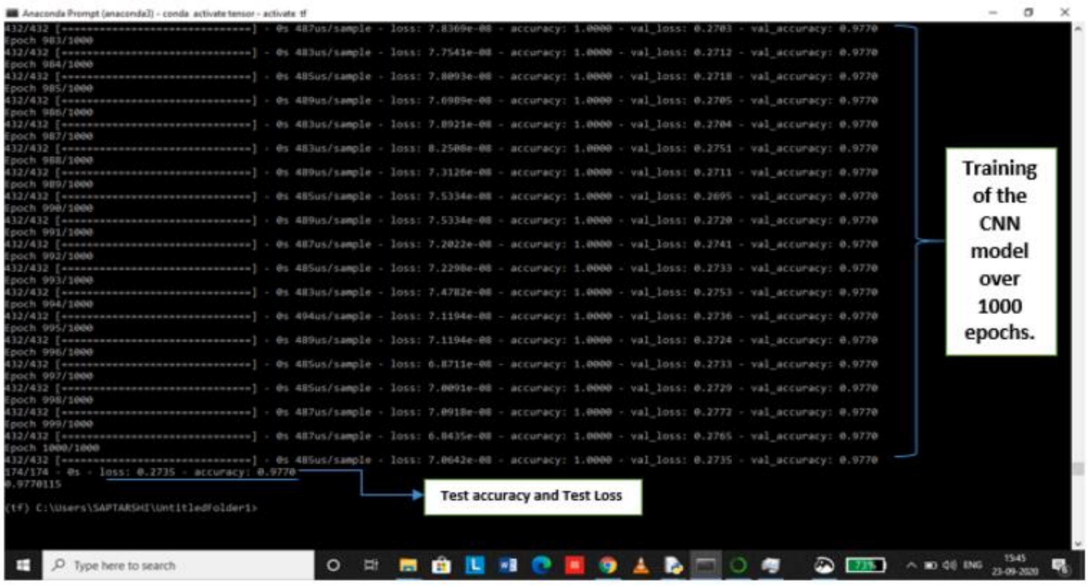
4.5.2 Fitting : The fit() function has been adopted to train the neural network on a set of inputs which are the training dataset along the training labels for 1000 epochs and produce a set of target outputs based on the CNN is a type of Deep Learning process which is implemented through neural networks. The code snippet for compiling and model fitting is depicted in Fig. 8.through code snippet.



```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=1000,
                    validation_data=(test_images, test_labels))
```

Fig 8. a code snippet of the methods used during model fitting.


Anaconda Prompt (anaconda3) - conda activate tensor - activate tf
Training of the CNN model over 1000 epochs.
Test accuracy and Test Loss
Type here to search

```
432/432 [==============================] - 0s 487us/sample - loss: 7.8369e-08 - accuracy: 1.0000 - val_loss: 0.2703 - val_accuracy: 0.9770
Epoch 983/1000
432/432 [==============================] - 0s 483us/sample - loss: 7.7541e-08 - accuracy: 1.0000 - val_loss: 0.2712 - val_accuracy: 0.9770
Epoch 984/1000
432/432 [==============================] - 0s 485us/sample - loss: 7.8093e-08 - accuracy: 1.0000 - val_loss: 0.2718 - val_accuracy: 0.9770
Epoch 985/1000
432/432 [==============================] - 0s 489us/sample - loss: 7.6989e-08 - accuracy: 1.0000 - val_loss: 0.2705 - val_accuracy: 0.9770
Epoch 986/1000
432/432 [==============================] - 0s 483us/sample - loss: 7.8921e-08 - accuracy: 1.0000 - val_loss: 0.2704 - val_accuracy: 0.9770
Epoch 987/1000
432/432 [==============================] - 0s 483us/sample - loss: 8.2508e-08 - accuracy: 1.0000 - val_loss: 0.2751 - val_accuracy: 0.9770
Epoch 988/1000
432/432 [==============================] - 0s 489us/sample - loss: 7.3126e-08 - accuracy: 1.0000 - val_loss: 0.2711 - val_accuracy: 0.9770
Epoch 989/1000
432/432 [==============================] - 0s 485us/sample - loss: 7.5334e-08 - accuracy: 1.0000 - val_loss: 0.2695 - val_accuracy: 0.9770
Epoch 990/1000
432/432 [==============================] - 0s 489us/sample - loss: 7.5334e-08 - accuracy: 1.0000 - val_loss: 0.2720 - val_accuracy: 0.9770
Epoch 991/1000
432/432 [==============================] - 0s 487us/sample - loss: 7.2022e-08 - accuracy: 1.0000 - val_loss: 0.2741 - val_accuracy: 0.9770
Epoch 992/1000
432/432 [==============================] - 0s 485us/sample - loss: 7.2298e-08 - accuracy: 1.0000 - val_loss: 0.2733 - val_accuracy: 0.9770
Epoch 993/1000
432/432 [==============================] - 0s 483us/sample - loss: 7.4782e-08 - accuracy: 1.0000 - val_loss: 0.2753 - val_accuracy: 0.9770
Epoch 994/1000
432/432 [==============================] - 0s 494us/sample - loss: 7.1194e-08 - accuracy: 1.0000 - val_loss: 0.2736 - val_accuracy: 0.9770
Epoch 995/1000
432/432 [==============================] - 0s 489us/sample - loss: 7.1194e-08 - accuracy: 1.0000 - val_loss: 0.2724 - val_accuracy: 0.9770
Epoch 996/1000
432/432 [==============================] - 0s 485us/sample - loss: 6.8711e-08 - accuracy: 1.0000 - val_loss: 0.2733 - val_accuracy: 0.9770
Epoch 997/1000
432/432 [==============================] - 0s 485us/sample - loss: 7.0091e-08 - accuracy: 1.0000 - val_loss: 0.2729 - val_accuracy: 0.9770
Epoch 998/1000
432/432 [==============================] - 0s 487us/sample - loss: 7.0918e-08 - accuracy: 1.0000 - val_loss: 0.2772 - val_accuracy: 0.9770
Epoch 999/1000
432/432 [==============================] - 0s 487us/sample - loss: 6.8435e-08 - accuracy: 1.0000 - val_loss: 0.2765 - val_accuracy: 0.9770
Epoch 1000/1000
432/432 [==============================] - 0s 485us/sample - loss: 7.0642e-08 - accuracy: 1.0000 - val_loss: 0.2735 - val_accuracy: 0.9770
174/174 - 0s - loss: 0.2735 - accuracy: 0.9770
0.9770115

(tf) C:\Users\SAPTARSHI\UntitledFolder1>
```

Fig 9. Training of the model over 1000 epochs showing the training and the testing accuracy and losses

4.6 CNN training based on the CNN feature learning network

The CNN feature learning network is used to train the model. During the training process, an output snippet is shown in Figure 9 as the model undergoes 1000 epochs.

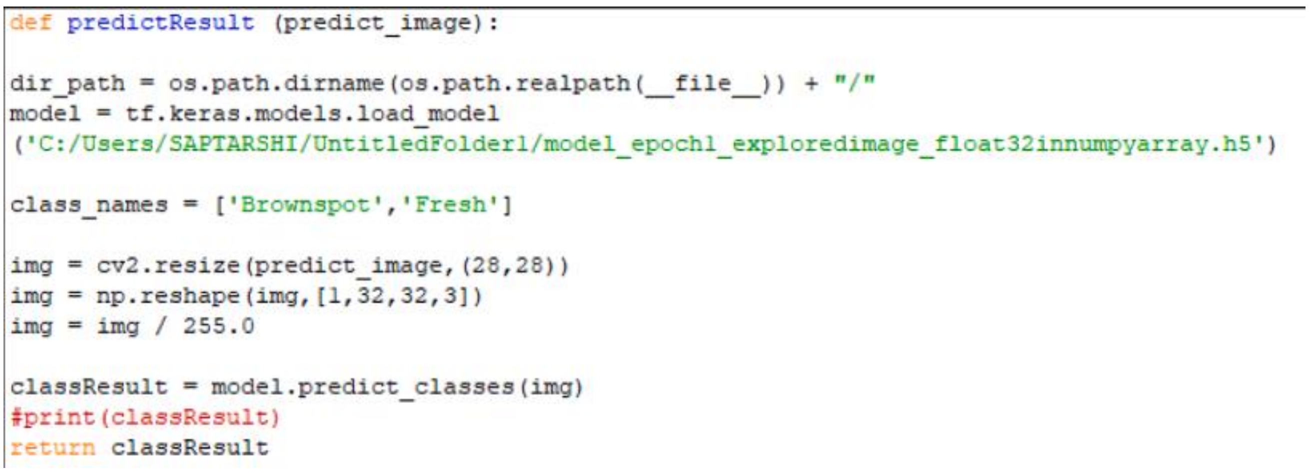
After the training phase, the model is tested and the test accuracy and loss are evaluated. The proposed model in this paper successfully detected Brown spot disease in rice paddy with a testing accuracy of 97.70115%. The minimum testing loss obtained was 5.42%. The results of the test accuracy and loss are illustrated in Figure 9.

4.7 Prediction and model evaluation

The paper describes a CNN-based approach for the early detection of Brown spot disease in rice paddy. The proposed method involves training a CNN on a dataset of images of healthy and diseased rice paddy plants. The trained model is then used to predict the presence of the disease in new images. A custom mobile app has also been developed to allow farmers to easily use the model to detect the disease in their crops.The training process involves defining a CNN architecture using the Keras library and fitting it to the training dataset for 1000 epochs. The resulting model is then saved as an H5 file for later use. To make predictions on new images, a separate Python module is created that loads the saved model and defines a method for predicting the result. The method takes an image as input, pre-processes it in the same way as during training, and predicts the presence of the disease.

The proposed method achieves a high level of accuracy in detecting Brown spot disease, with a testing accuracy of 97.70115% and a lowest testing loss of 5.42%. The performance of the model is evaluated using line graphs that show the training and testing accuracy and losses over 100 epochs. The results show that the model is successful in detecting the disease and outperforms existing methods.

Overall, the proposed approach is a promising solution for the early detection of Brown spot disease in rice paddy. It has the potential to be deployed in the agriculture sector to prevent further damage to crops and assist farmers in making better decisions about fertilizers and chemicals. The custom mobile app also makes the method more accessible and user-friendly for farmers.



```
def predictResult (predict_image):

dir_path = os.path.dirname(os.path.realpath(__file__)) + "/"
model = tf.keras.models.load_model
('C:/Users/SAPTARSHI/UntitledFolder1/model_epoch1_exploredimage_float32innumpyarray.h5')

class_names = ['Brownspot','Fresh']

img = cv2.resize(predict_image,(28,28))
img = np.reshape(img,[1,32,32,3])
img = img / 255.0

classResult = model.predict_classes(img)
#print(classResult)
return classResult
```

Fig 10. The functionality of the predictResult() method

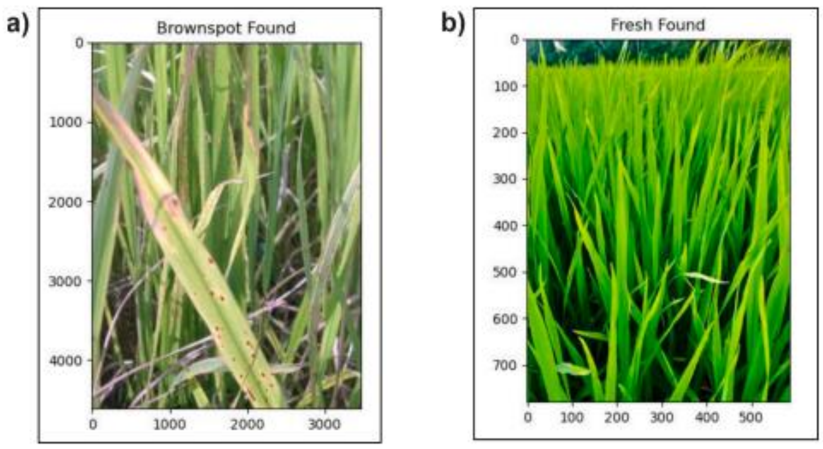


fig 11 (a, b) evaluation of the prediction model with foreign images

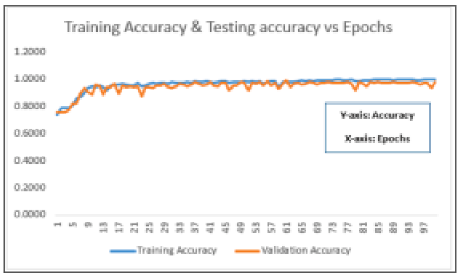


fig 12. training accuracy and validation accuracy plotted against the number of epoch

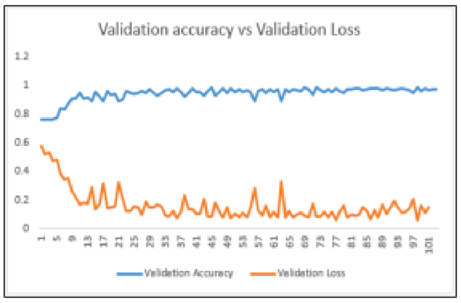


fig 13. accuracy vs loss(testing dataset)

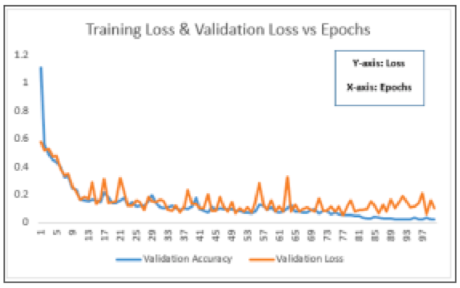


fig 14. training loss and validation loss plotted against the number of epochs

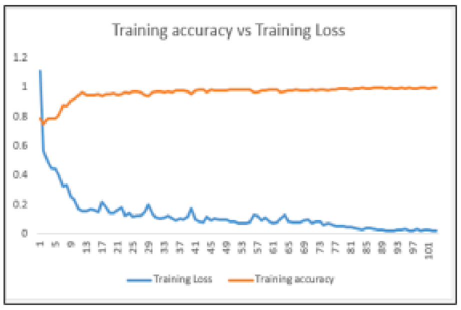


fig 15. accuracy vs loss (training).

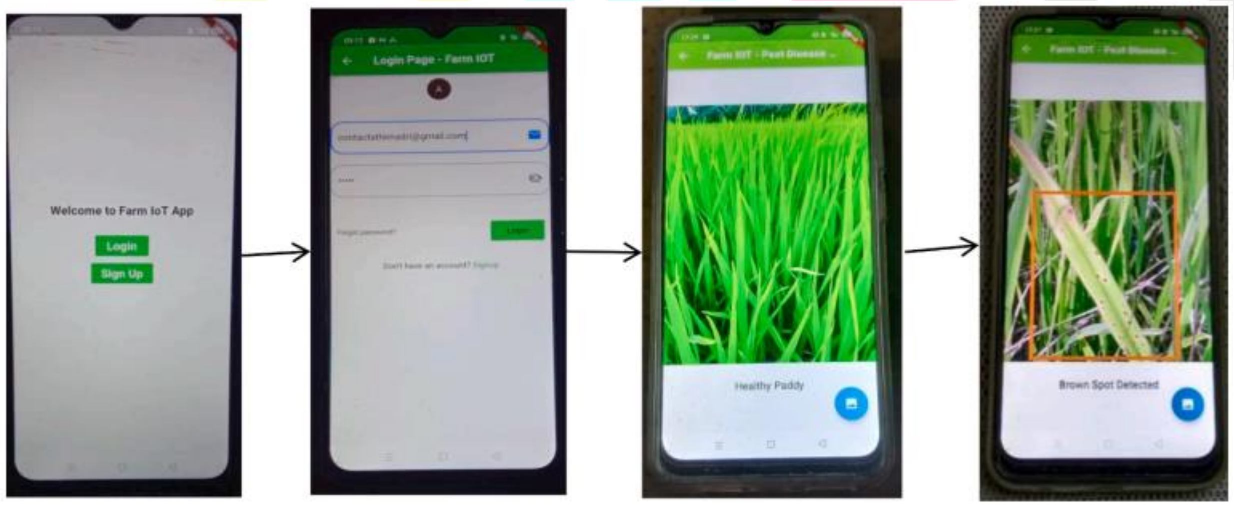


fig. 16. step by step windows of our smart farming app named "farm iot".

6. Conclusion and future work

The paper proposes a new method for detecting Brown spot, a fungal disease occurring in rice paddy, using a convolutional neural network (CNN) integrated with IoT. The novelty of this work lies in using real-time data for training and testing the model, without any predefined dataset. Cloud computing has been used to store the data, and an app has been developed for farmers to detect diseases easily and instantly. The proposed model has achieved high testing accuracy of 97.7%, which is better than existing CNN models for disease detection in rice paddy. The model has been built on advanced frameworks like TensorFlow and Keras API, making it highly flexible. In addition, the dataset preparation involved data pruning and cleaning using a self-designed tool, labelling of the dataset, and catalogue files to ease data sharing.

This proposed system is expected to help farmers worldwide with little or no knowledge of crop pathology to detect and understand emerging crop diseases. Early detection of disease can lead to adopting preventive measures to prevent further damage to the crop, resulting in increased crop production and decreased losses. This work is a significant contribution to the agricultural industry and has the potential to minimize crop losses due to various crop diseases. The proposed model can be extended in the future to detect multiple classes of diseases and the intensity of the attack in crops.

ACKNOWLEDGMENT

The successful fulfilment of this paper depends on many people whom I would like to take a moment to thank. First, I would like to thank Department of Computer science and engineering Kvg College Of Engineering Sullia, Manglore. My heartfelt gratitude is Kiran B.B Co-author Department of Computer science and engineering Kvg College Of Engineering Sullia, Manglore.For his guidance, motivation and for sparing his extremely precious time for me.

REFERENCES

[1] M. Dutot, L.M. Nelson, R.C. Tyson, ‘Predicting the spread of postharvest disease in stored fruit, with application to apples, Postharvest Biol. Technol. 85 (2013) 45–56. Nov.

[2] C. Wu, C. Luo, N. Xiong, W. Zhang, T. Kim, ‘A greedy deep learning method for medical disease analysis, IEEE Access 6 (2018) 20021–20030

[3] M. Islam, A. Dinh, K. Wahid, P. Bhowmik, ‘Detection of potato diseases using image segmentation and multiclass support vector machine, in: Proceedings of the IEEE 30th Canadian Conference on Electrical and Computer Engineering, IEEE, 2017, pp. 1–4. Apr./May.